HYU

MICROPROCESSOR APPLICATIONS

NAM

LAB_03_ASSEMBLY

박기완
rldhks6899@hanyang.ac.kr

BITWISE

Bit types

Binary, Decimal, Hexadecimal number

Decimal Value	Hexadecimal Value	Binary Value
0	00	0000 0000
1	01	0000 0001
2	02	0000 0010
3	03	0000 0011
4	04	0000 0100
5	05	0000 0101
6	06	0000 0110
7	07	0000 0111
8	08	0000 1000
9	09	0000 1001
10	0A	0000 1010
11	0B	0000 1011
12	0C	0000 1100
13	0D	0000 1101
14	0E	0000 1110
15	0F	0000 1111
16	10	0001 0000
17	11	0001 0001
18	12	0001 0010
19	13	0001 0011
20	14	0001 0100
21	15	0001 0101
22	16	0001 0110
23	17	0001 0111
24	18	0001 1000
25	19	0001 1001
26	1A	0001 1010
27	1B	0001 1011
28	1C	0001 1100
29	1D	0001 1101
30	1E	0001 1110
31	1F	0001 1111
32	20	0010 0000
33	21	0010 0001
34	22	0010 0010
35	23	0010 0011
36	24	0010 0100
37	25	0010 0101
38	26	0010 0110
39	27	0010 0111
40	28	0010 1000
41	29	0010 1001
42	2A	0010 1010
43	2B	0010 1011

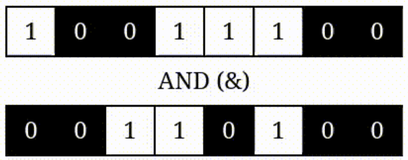
ref. https://www.oreilly.com/library/view/ccna-routing-and/9780133381351/app01.html

Bit Operations

Binary, Decimal, Hexadecimal number

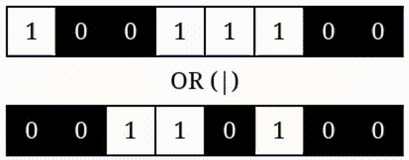
bitwise AND (&)

- The result is 1 only if both values are 1.



bitwise OR (|)

- The result is 1 if at least one of the values is 1.



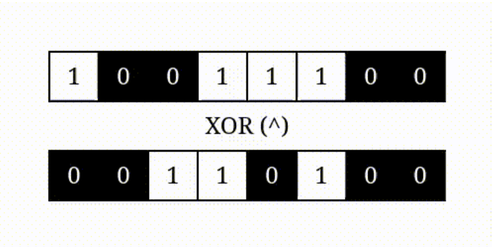
ref. https://www.oreilly.com/library/view/ccna-routing-and/9780133381351/app01.html

Bit Operations

Binary, Decimal, Hexadecimal number

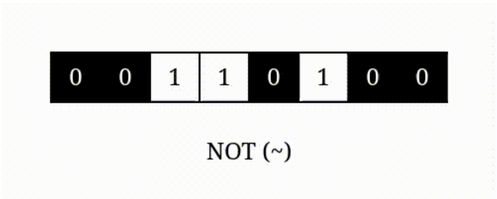
bitwise XOR (^)

- The result is 1 only if the values are different.



bitwise NOT (~)

- Flips the bits, 0 becomes 1, and 1 becomes 0.



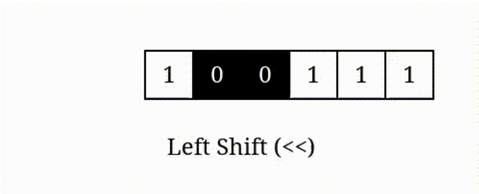
ref. https://www.oreilly.com/library/view/ccna-routing-and/9780133381351/app01.html

Bit Operations

Binary, Decimal, Hexadecimal number

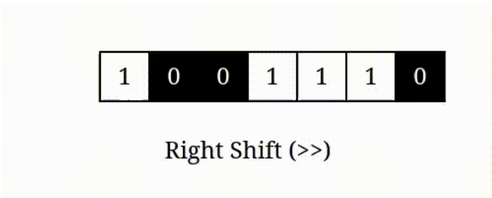
bitwise Left SHIFT (<<)

- Shifts the bits to the left. (Doubling the value)



bitwise Right SHIFT (>>)

- Shifts the bits to the right. (Halving the value)



ref. https://www.oreilly.com/library/view/ccna-routing-and/9780133381351/app01.html

EXERCISE REVIEW

ref. https://www.oreilly.com/library/view/ccna-routing-and/9780133381351/app01.html

Exercise Review

Most Asked Questions

Q. LED 점등이 C언어일 때와 ASSEMBLY 언어일 때 다른 이유

C : _delay_ms(100) 함수를 통해 PORTD의 값이 0.1초마다 변하도록 의도했기 때문에 LED의 Blink를 확인 가능

ASSEMBLY : delay 함수가 따로 없었기 때문에 LED가 매우 빠른 속도로 깜빡이고 있어 계속 켜져있는 것처럼 보인 것
(실제로는 프로그램 명령어들이 실행되는 동안 자연스럽게 발생하는 매우 짧은 내부 딜레이가 있으나 육안으로 확인이 불가능)

```
.include "m128def.inc"

.CSEG
.ORG    0x00


RESET:
    LDI R16, 0xF0
    OUT DDRD, R16

LOOP:
    LDI R16, 0XF0
    OUT PORTD, R16

    LDI R16, 0X0F
    OUT PORTD, R16

    RJMP    LOOP
```

PORTD에 0xF0 값이 들어오자마자 PORTD에 0x0F 값이 들어옴 (매우 짧은 순간)

Exercise Review

Most Asked Questions

Q. 코드에서 R16에 0xF0을 로드하고, 0×0F를 로드한 이유

0xF0 = 0b11110000이고, 0x0F = 0b00001111입니다.

이번 실습에서는 상위 4bit에 연결된 PD4-7의 LED 제어에만 관심이 있기 때문에 하위 4bit의 변화는 신경쓰지 않습니다.

가령, 0x0F 대신 0x01, 0x0C, 0x0E와 같은 값을 넣는다고 하더라도 LED에는 아무런 동작을 하지 않습니다.

하지만 효율적인 코딩을 위해선 0xF0 다음 0x00 값을 넣어주는 것이 올바른 방식이긴 합니다.

```
.include "m128def.inc"

.CSEG
.ORG    0x00


RESET:
    LDI R16, 0xF0
    OUT DDRD, R16

LOOP:
    LDI R16, 0XF0
    OUT PORTD, R16

    LDI R16, 0X0F
    OUT PORTD, R16

    RJMP    LOOP
```

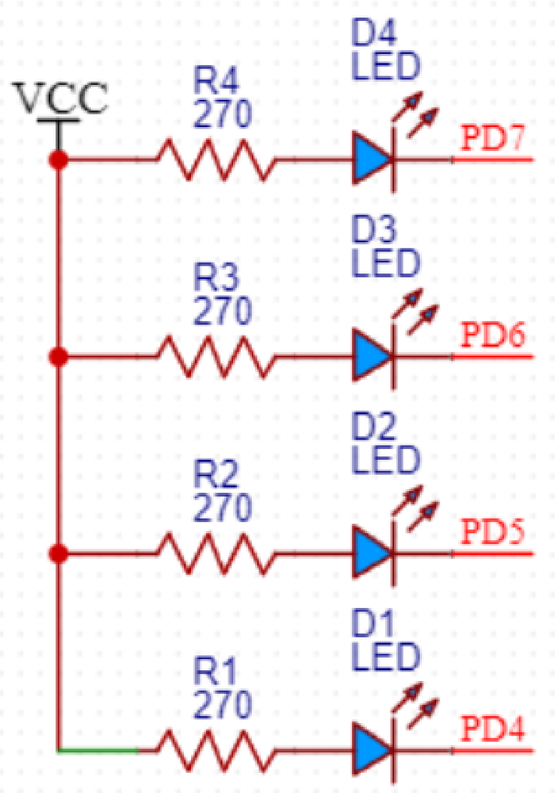
Exercise Detail Review

review in details

Q. PORTD에서 값이 LOW일때 LED가 켜지는 이유

실습 보드의 회로도 참고하면, Built-in LED는 MCU와 Pull-Up register와 연결되어 있음

(반대로 Pull-Down으로 연결된 경우, PORT 신호가 HIGH일 때 LED가 ON이 된다.)



PD4-7 포트는 LOW 일 때

전류가 흐를 수 있도록 설계 되어 있음을 알 수 있다.

Exercise Detail Review

review in details

Q. LDI에서 R0-R15를 사용하지 않고 R16를 사용한 이유

ATmega128에는 General Purpose Registers (이하 GPRs)는 R0-R31까지 총 32개 존재

→ Lower Register : 0x00 [R0] ~ 0x0F [R15] → 데이터 임시 저장, 레지스터 간의 데이터 이동 등

→ Upper Register : 0x10 [R16] ~ 0xF0 [R31] → 상수 값 및 주소 값 저장 등

AVR 명령어는 대부분 16bit로 구성되어 있기 때문에 가능한 공간 효율적인 처리가 필요하기 때문에
대부분의 명령어는 실제 구조 상에서 R16부터 값을 지원하도록 설계되어 있음
(비슷한 이유로 LDI 명령어 역시 R16부터 R31까지만 사용 가능)

6.69 LDI – Load Immediate

K : Constant data
Rd : destination Register

6.69.1 Description

Loads an 8-bit constant directly to register 16 to 31.

Operation:

(i) Rd ← K

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

Exercise Detail Review

review in details

Q. Routine 중에 RESET 같이 이미 역할이 정해져있는 경우

AVR MCU에는 RESET, INT0, TIMER0, OVF 등과 같은 Interrupt Routine이 이미 정의가 되어 있음

그 중 RESET 루틴만 살펴보자면, 이는 MCU에 전원이 인가되거나 즉, MCU가 시작될 때 초기화를 수행하기 위해 사용

레지스터 초기화, 데이터 메모리 초기화, 스택 초기화 등등

하지만 RESET 루틴은 가급적 권장되는 초기화 방법이 아니기 때문에 CSEG, DSEG를 사용하는 것이 일반적

→ 실습에서 배울 예정

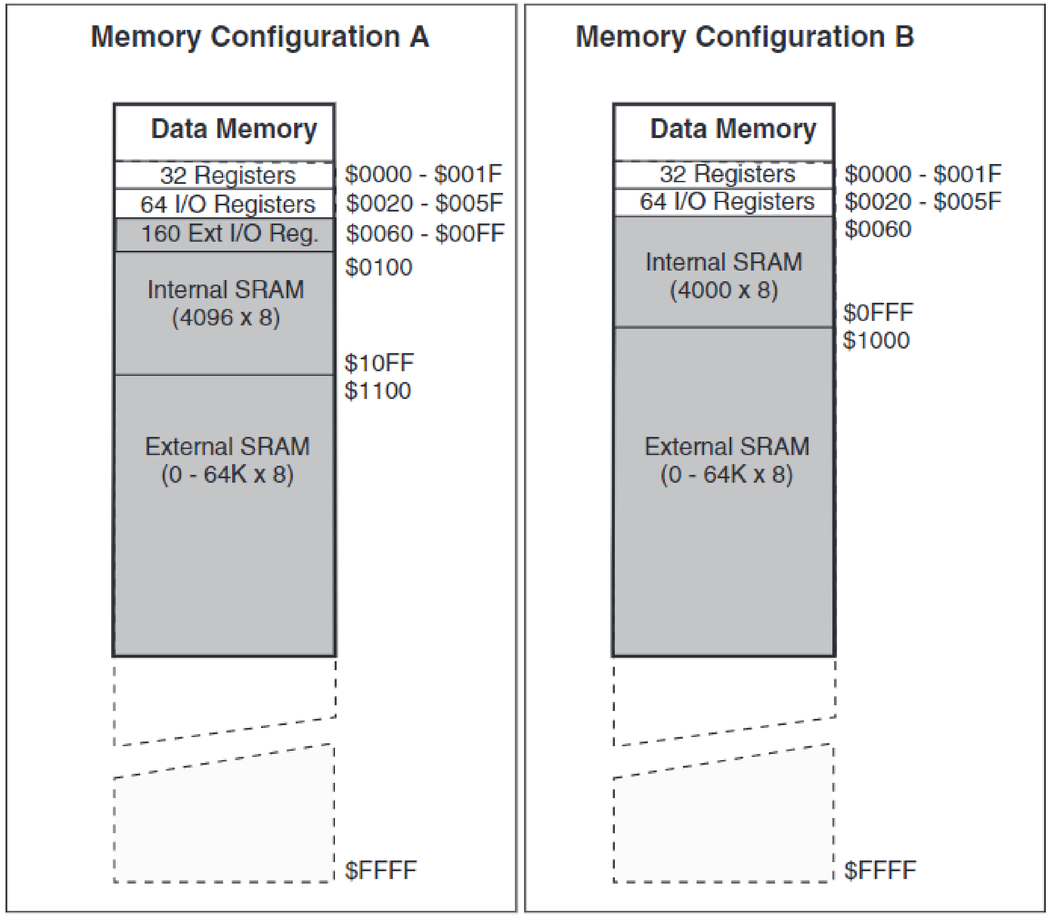
I/O PORT

ATmega128 Data Memory Address

Data Memory

- General Purpose Registers, GPRs (R0-R31) → $0000 - $001F
  - 0×00(R0) ~ 0×0F (R15)
  - 0×10(R16) ~ 0×1F(R31)
- I/O Registers (DDR, PORT, PIN) → $0020 - $005F
  - For compatibility, Register address start at 0x20

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$03 ($23)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0
$02 ($22)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0
$01 ($21)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0
$00 ($20)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0



I/O Ports

port architecture

Port Architecture

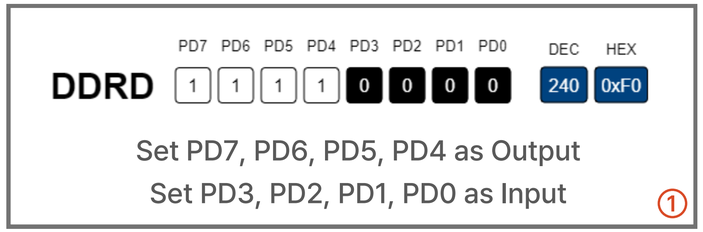
- Data Direction Register, DDR : 해당 PORT의 각 Pin을 Input Port로 사용할 지, Output Port로 사용할지를 결정
- Data Register, PORT : Register에서 받은 Data 값을 Output으로 출력
- Port Input Pins Register, PIN : 입력 PIN으로 부터 Data를 받아 Register로 전달
- DDR를 통해 PORT를 Input/Output 중 어떤 역할로 사용할 건지 결정하고, 역할에 따라 Reg를 적절히 사용
  - DDRA = 0xFF → Port A 8-bit는 모두 Output 역할 → PORTA = 0x01 → PA1를 Output port로 사용
  - DDRA = 0x00 → Port A 8-bit는 모두 Input 역할 → PINA = 0x01 → PA1를 Input port로 사용

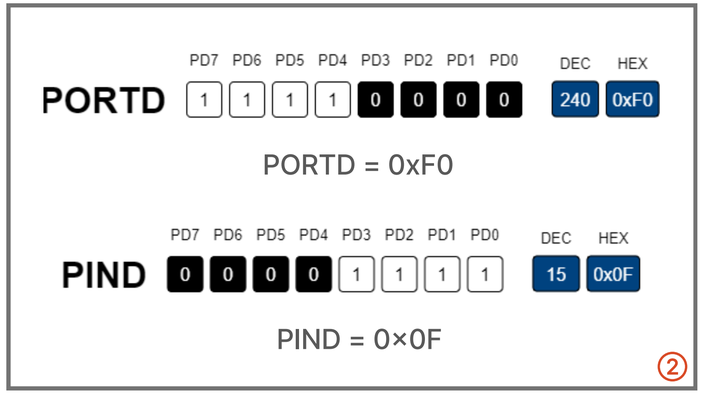
I/O Ports example

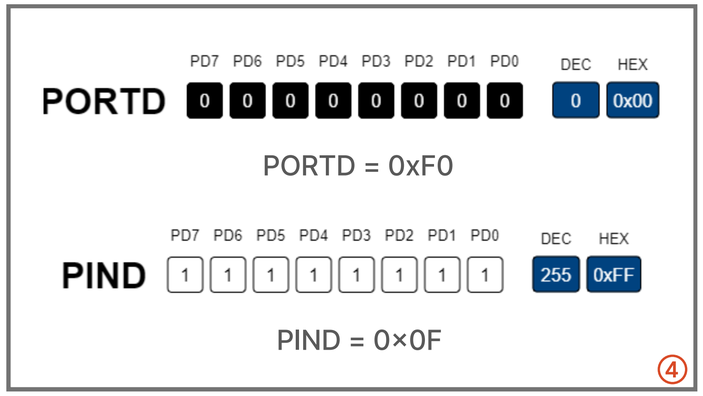
port architecture

LED BLINK

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

int main(void)
{
    DDRD = 0xF0;                ①

    while (1)
    {
        PORTD= 0xF0;            ②
        _delay_ms(100);         ③
        PORTD = 0x00;           ④
        _delay_ms(100);         ⑤
    }
    return 0;
}
```







I/O Ports example

port architecture

LED BLINK sequentially

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

int main(void)
{
    DDRD = 0xF0;
    uint8_t LED = 0;                ①

    while (1)
    {
        PORTD = 0x10 << LED;        ②
        _delay_ms(200);             ③
        PORTD = 0x00;               ④
        LED++;                      ⑤
        if(LED==4)LED=0;
    }
    return 0;
}
```

LED = 0 ①

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
0	0	0	1	0	0	0	0	16	0x10

PORTD = 0×10 << LED ②

delay 0.2 sec ③

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
0	0	0	0	0	0	0	0	0	0x00

PORTD = 0×00 ④

LED++ ⑤

Bitwise Shift Left

0×10 << 1 = 0×20 = 0010 0000

0×10 << 2 = 0×40 = 0100 0000

0×10 << 3 = 0×80 = 1000 0000

0X10 << 4 = 0X100 = 1 0000 0000

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
0	0	1	0	0	0	0	0	32	0x20

PORTD = 0×10 << LED(1)

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
0	1	0	0	0	0	0	0	64	0x40

PORTD = 0×10 << LED(2)

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
1	0	0	0	0	0	0	0	128	0x80

PORTD = 0×10 << LED(3)

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
0	0	0	1	0	0	0	0	16	0x10

PORTD = 0×10 << LED

LED is "4" → LED = 0

처음으로 돌아가기

I/O Ports example

port architecture

LED BLINK with SWITCH

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

int main(void) {
  DDRD = 0xF0;  ①

  while (1) {
    if (PINA & 0X02) { ②
      PORTD |= 0xF0; // (Turn OFF) ③
    }
    else { // If the button is pressed ④
      PORTD &= 0x00; // (Turn ON) ⑤
    }
    _delay_ms(10); ⑥
  }
}
```

DDRA

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	DEC	HEX
0	0	0	0	0	0	0	0	0	0x00

DDRA data is not changed

→ Set all PA as Input

DDRD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
1	1	1	1	0	0	0	0	240	0xF0

Set PD7, PD6, PD5, PD4 as Output

Set PD3, PD2, PD1, PD0 as Input ①

PINA

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	DEC	HEX
0	0	0	0	0	0	1	0	2	0x02

if PINA & 0×02 → input 상태인 PA1를 read

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
1	1	1	1	0	0	0	0	240	0xF0

PORTD |= 0xF0 ③

②

PINA

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	DEC	HEX
0	0	0	0	0	0	0	0	0	0x00

else (except PINA & 0×02)

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
0	0	0	0	0	0	0	0	0	0x00

PORTD &= 0xF0 ⑤

④

10ms delay ⑥

Bitwise OR

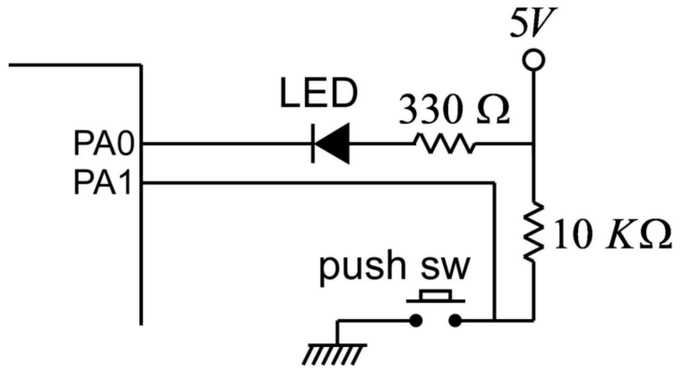
0×00 |= 0×01 → 0×01

0×00 = 0×00|0×01 = 0×01

Bitwise AND

0×00 &= 0×01 → 0×00

0×00 = 0×00&0×01 = 0×00



I/O Ports example

port architecture

LED Sequentially with SWITCH toggle

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

int main(void) {
    DDRD = 0xF0;
    uint8_t toggle = 0; // toggle 변수 선언 및 초기화

    while (1) {
        if (PINA & 0X02) {
            toggle = 0;
        }
        else { // If the button is pressed
            if(toggle == 0) {
                PORTD ^= 0xF0; // XOR
                toggle = 1;
            }
        }
        _delay_ms(10); // Input value delay for chattering
    }
}
```

DDRD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
1	1	1	1	0	0	0	0	240	0xF0

Set PD7, PD6, PD5, PD4 as Output
Set PD3, PD2, PD1, PD0 as Input

①

PORD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
0	0	0	0	0	0	0	0	0	0x00

PORTD initial state
(아무것도 안넣음)

if toggle is "zero"

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
1	1	1	1	0	0	0	0	240	0xF0

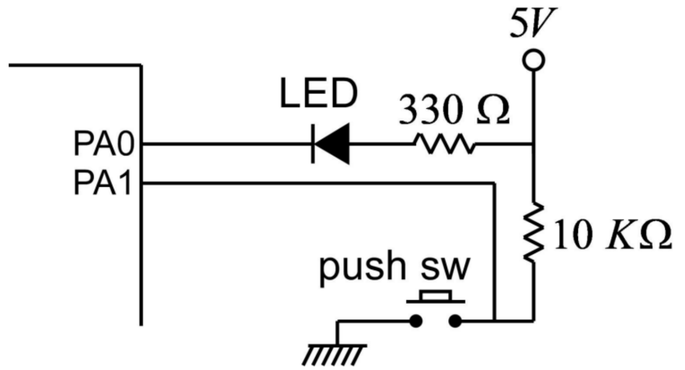
PORTD ^= 0xF0
[Flip bits]

if toggle is "one"

PORTD

PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	DEC	HEX
0	0	0	0	1	1	1	1	15	0x0F

PORTD ^= 0xF0
[Flip bits]



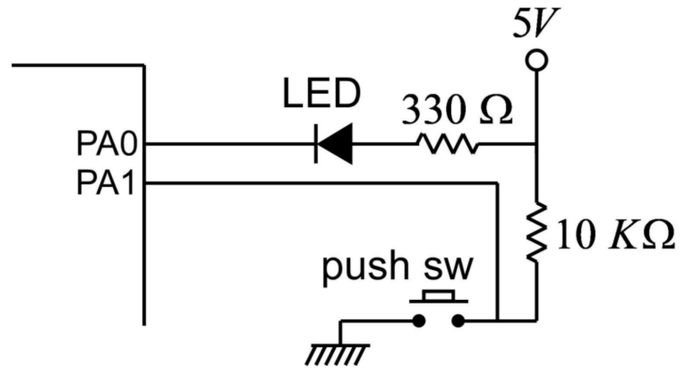
Exercise 1

> LED CONTROL <

Make the LED blink sequentially-serially when the button is pressed

I/O Ports Exercise

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

int main(void) {
  DDRD |= 0xF0;

  uint8_t LED = 0;
  uint8_t toggle = 0;

  while (1) {
    if(LED==4)LED=0;
    if (PINA & 0x02) {
      toggle = 0;
    } else {
      if (toggle == 0) {
        PORTD = 0x10 << LED;
        LED++;
        toggle = 1;
      }
    }

    _delay_ms(10); // Input value delay for chattering
  }
}
```



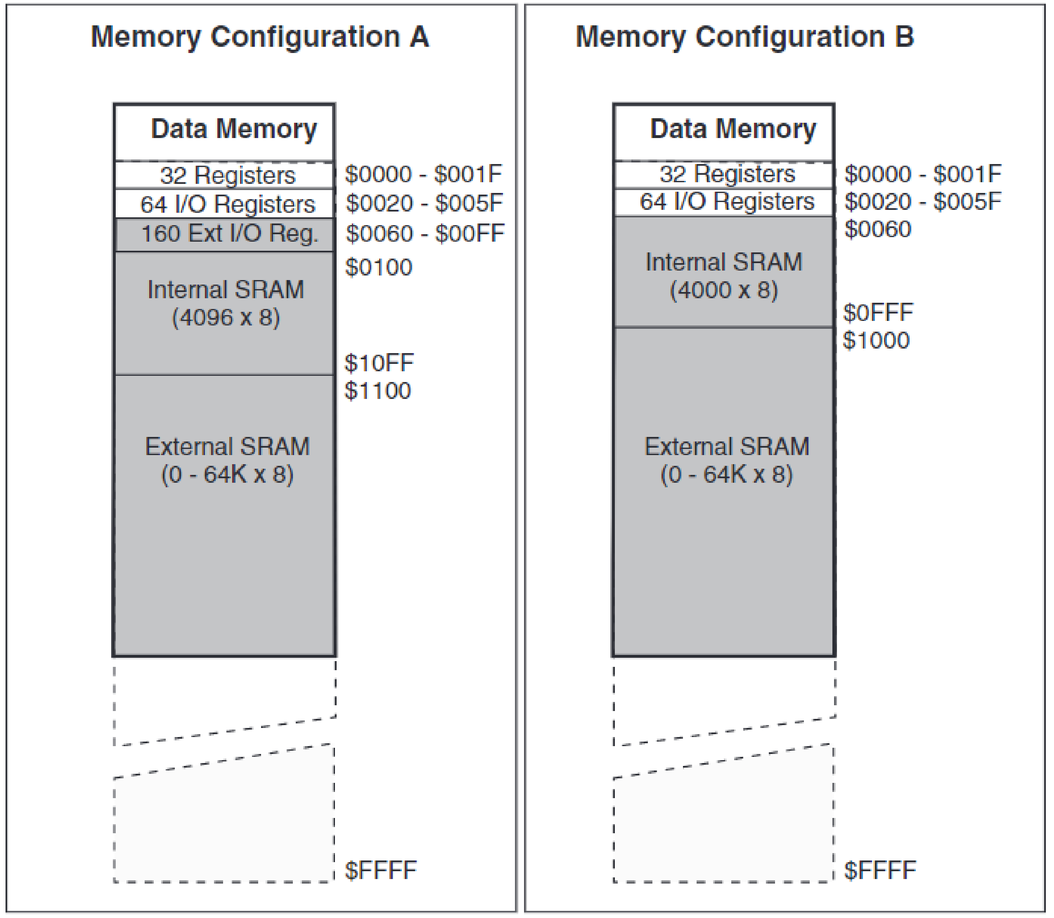
ASSEMBLY

ATmega128 Data Memory Address

Data Memory

- General Purpose Registers, GPRs (R0-R31) → $0000 - $001F
  - 0×00(R0) ~ 0×0F (R15)
  - 0×10(R16) ~ 0×1F(R31)
- I/O Registers (DDR, PORT, PIN) → $0020 - $005F
  - For compatibility, Register address start at 0x20

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
$03 ($23)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0
$02 ($22)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0
$01 ($21)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0
$00 ($20)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0



PROGRAM DIRECTIVES

프로그램 지시어

프로그램의 구조와 동작을 정의하는 명령어 종류

HEADER

.DEF	Define a symbolic name on a register	.DEF myReg = R16
.EQU	Set a symbol and later re-definition is possible	.EQU tempe = 1004
.SET	Set a symbol and later re-definition is not possible	.SET temps = 369
.INCLUDE	Include a file	.INCLUE "m128def.inc

CODE

.CSEG	Start of the code segment	
.DB	Define a constant Byte(s) data	.DB 0×12, 0×34 or .DB "Hello"
.DW	Define a constant Words data	.DW 0×1234, 0×5678

PROGRAM DIRECTIVES

프로그램 지시어

프로그램의 구조와 동작을 정의하는 명령어 종류

SRAM

.DSEG	Start of Data Segment	
.BYTE	Reserve byte to a variable	.BYTE 0×01, 0×02, 0×03

Everywhere

.ORG	Set the address of program origin	.ORG 0×00
.EXIT	Exit from a file	

PROGRAM DIRECTIVES

프로그램 지시어

프로그램의 구조와 동작을 정의하는 명령어 종류

Advanced

.IF <Condition> ~ .ENDIF

.IF <Condition> ~ .ELSE ~ .ENDIF

.IF <Condition1> ~ .ELIF <Condition2> ~ .ENDIF

.IFDEF <Symbol> : Check if a symbol has been defined.

"TRUE" → 다음 코드 시퀀스 실행 "FALSE" → 다음 코드 시퀀스 실행안됨

.IFNDEF <Symbol> : 위에거 반대

PROGRAM INSTRUCTIONS

프로그램 지시어

100개 이상의 명령어가 있지만, 예제에 쓰이는 것만 하나씩 알아보기

IN (Input from)

```
IN R16, PINA          ; Read the value of PINA and store it in register R16
```

OUT (Output to)

```
OUT PORTB, R16        ; Write the value of register R16 to PORTB
```

CLR (CLear Register)

```
CLR R16               ; Set the contents of register R16 to zero
```

EOR (Exclusive OR)

```
EOR R16, R17          ; Perform an XOR operation between R16 and R17, and store the result in R16
```

PROGRAM INSTRUCTIONS

프로그램 지시어

100개 이상의 명령어가 있지만, 예제에 쓰이는 것만 하나씩 알아보기

IN (Input from)

```
IN R16, PINA          ; Read the value of PINA and store it in register R16
```

OUT (Output to)

```
OUT PORTB, R16        ; Write the value of register R16 to PORTB
```

CLR (CLear Register)

```
CLR R16               ; Set the contents of register R16 to zero
```

EOR (Exclusive OR)

```
EOR R16, R17          ; Perform an XOR operation between R16 and R17, and store the result in R16
```

LDI (LoaD Immediate)

```
LDI R16, 0x0F         ; Load the immediate value 0×0F into register R16
```

PROGRAM INSTRUCTIONS

프로그램 지시어

100개 이상의 명령어가 있지만, 예제에 쓰이는 것만 하나씩 알아보기

CPI (Compare Immediate) : Compare register with a value

BREQ (Branch if Equal)

```
CPI R16, 0x03          ; Compare register R16 with the immediate value 0×03
BREQ EQUAL             ; Branch to the EQUAL label if R16 is equal to 0×03
```

BRNE (Branch if Not Equal)

```
CPI R16, 0x03
BRNE NOT_EQUAL         ; Branch to the NOT_EQUAL label if R16 is not equal to 0×03
```

RJMP (Relative Jump)

```
RJMP LOOP              ; Unconditional jump to a relative address in the program memory.
```

Exercise 2

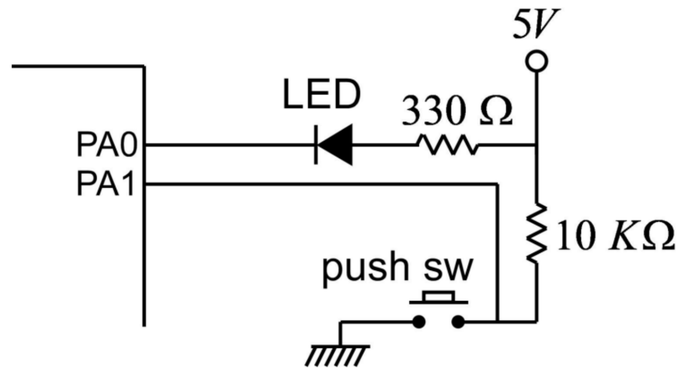
> ASSEMBLY <

Let's make Exercise 1 in Assembly Code

> Make the LED blink sequentially-serially when the button is pressed <

Assembly Exercise

```
.INCLUDE "m128def.inc"

; Register 변수 지정 (지시어 사용)
.DEF temp   = r16
.DEF toggle = r17

.ORG 0x0000
    RJMP INIT

INIT: ; 변수 초기화 단계
    LDI temp, 0xF0
    OUT DDRD, temp

    CLR toggle

LOOP:
    IN temp, PINA ;
    ANDI temp, 0x02 ;
    BREQ TOGGLE_RESET

    CPI toggle, 0
    BRNE LOOP

    IN temp, PORTD
    LDI r18, 0xF0
    EOR temp, r18

    OUT PORTD, temp

    LDI toggle, 1

    RJMP LOOP


TOGGLE_RESET:
    CLR toggle
    RJMP LOOP
```



Homework

give me the report

- Exercise 1과 Exercise 2의 동작을 비교하고, 각 Exercise 코드를 비교해주세요.
- Assembly 코드에서 사용된 명령어를 정리해주세요.

- Exercise 2에는 Delay가 없기 때문에 Switch 동작이 이상할 수 있습니다.

  그 이유에 대해 생각해보시고, Discussion에 포함해주세요.