C - HW6

임베디드스쿨1기

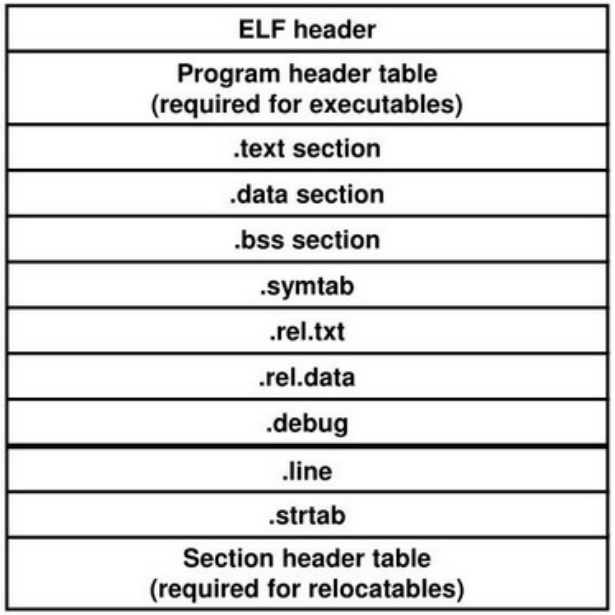
Lv1과정

2020. 09.04

박하늘

1-1. ELF & HEX ?

1) ELF(Executable and Linkable Format)
- 여러 운영체제 환경으로 확장될 수 있는 binary 인터페이스 정의들의 집합을 제공
- 다른 플랫폼에 의해서 적용 가능(링킹 가능한 파일들): executable, relocatable object file(.o), shared object file(.so)이 모두 따름



.text	실제 코드를 저장하는 섹션. objdump -drS .process.o로 볼 수 있다.
.data	전역 테이블, 변수, 등을 저장하는 섹션. objdump -s -j .data .process.o는 이것을 hex 형식으로 덤프한다.
.bss	파일에서 .bss의 bit을 찾지 말자. 왜냐하면 거기엔 아무것도 없기 때문이다. 초기화 되지 않은 배열과 변수가 거기에 있고, loader는 이들이 0으로 채워져야만 한다고 "알고 있다". 이미 있는 곳 보다 더 많은 0을 디스크에 저장할 필요가 없기 때문이다.
.rodata	프로그램의 문자열이 저장되는 섹션, 보통 링킹 과정에서 잊어버리면, 커널이 동작하지 않게 된다. objdump -s -j .rodata.process.o 는 hex 형식으로 덤프한다. 컴파일러에 따라, 이와 같은 섹션이 더 많을 수도 있다.
.comment & .note	컴파일러/링커 툴체인에 의해서 입력되는 단순 커멘트
.stab & .stabstr	디버깅 심볼과 이와 유사한 정보

Source → compile → ELF → Link → hex → Flash programmer → Target Board

1-2. ELF & HEX ?

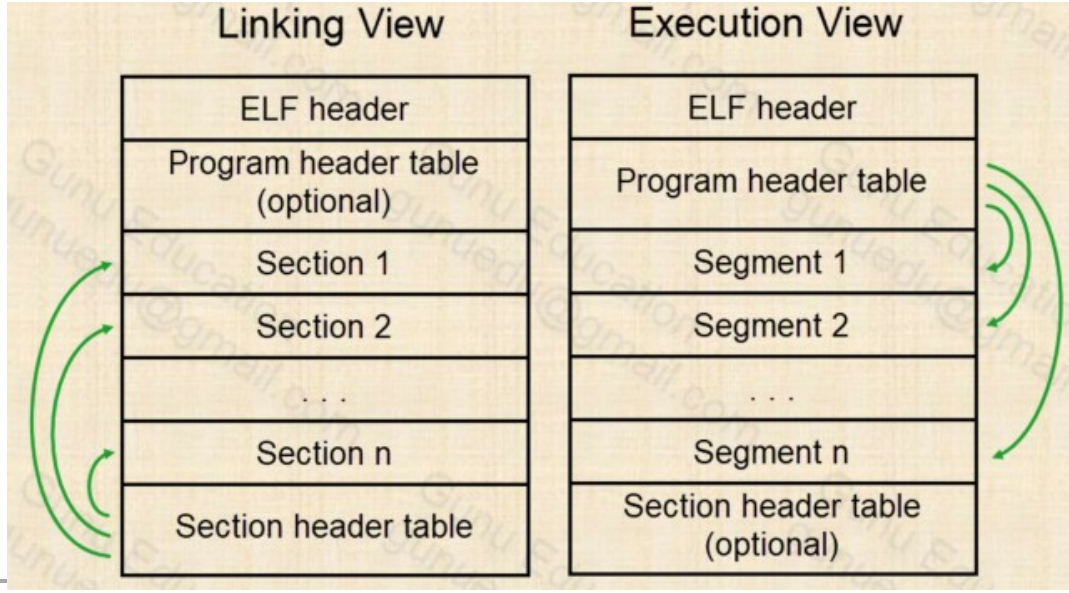
1) ELF - Object file 이란?

object 파일에는 크게 3가지가 존재한다.

– 재배치 가능 파일(relocatable file) : 이 파일은 링크를 쉽게 할 수 있는 코드 및 데이터와 함께, 실행 파일을 만들거나 공유 object파일을 생성하기위한 다른 object파일을 가지고 있는 형태의 파일이다.

– 실행 가능 파일(executable file) : 이 파일은 실행을 위해서 적합한 프로그램을 가지는 파일로서, exec 시스템 콜이 어떻게 프로세스의 이미지를 프로그램을 이용해서 만들지를 기술하는 파일이다.

– 공유 object 파일(shared object file) : 두개의 환경(context)상에서 링크에 적합한 형태의 코드와 데이터를 가지는 파일로서, 먼저 LD와 같은 링크 에디터가 이 파일과 함께, 다른 재배치 가능 파일과 공유 object파일을 처리해서 또하나의 object 파일을 생성해주게 되고, 이를 다시 동적 링커(dynamic linker)가 생성된 object 파일을 실행 가능 파일과 다른 공유 object파일을 묶어서 프로세스의 이미지를 생성해 주는 두 단계를 거치는 파일이다.



1-2. ELF & HEX ?

2) HEX ?

:	Data count	Address	Type	Data	CheckSum

-Intel 기준 레코드 구조

1. 시작코드(Start code) : ASCII 문자 ':'(colon).
2. 데이터 개수(Data count) : 16진수 문자 2 자리, 8Bit 데이터 갯수
3. 주소(Address) : 16진수 문자 4 자리, 시작 주소와의 옵셋 값을 표현한다.
시작 주소와 주소는 항상 빅 엔디언으로 표현된다.
4. 레코드 종류(Record type) : 16진수 문자 2 자리
5. 데이터(Data) : 2n 개의 16진수 문자로 표현된 n 바이트의 데이터, n은 Byte count의 값과 동일
6. 체크섬(Checksum) : 16진수 문자 2 자리, 레코드에 오류가 없음을 입증하는데 이용 될 수 있는 계산된 값

계산방법 1) 데이터를 모두 더하고 이 값을 256으로 나눈 나머지의 2의 보수(모든 비트를 뒤집고 1을 더한 값)
계산방법 2) ~((데이터를 모두 더한 값) & 0xff)
:021001001234A7 의 경우
0x02+0x10+0x01+0x00+0x12+0x34=0x059,
0x059 & 0x0FF = 0x059, 0x100 – 0x59 = 0xA7

- I8HEX 파일 : 00과 01 레코드만 사용. 16bit 주소(64K Byte)
- I16HEX 파일 : 00에서 03까지의 레코드만 사용. 20bit주소(1M Byte)
- I32HEX 파일 : 00, 01, 04, 05의 레코드만 사용. 32bit주소(4G Byte)

• 예제 파일) 16Bit Memory Address

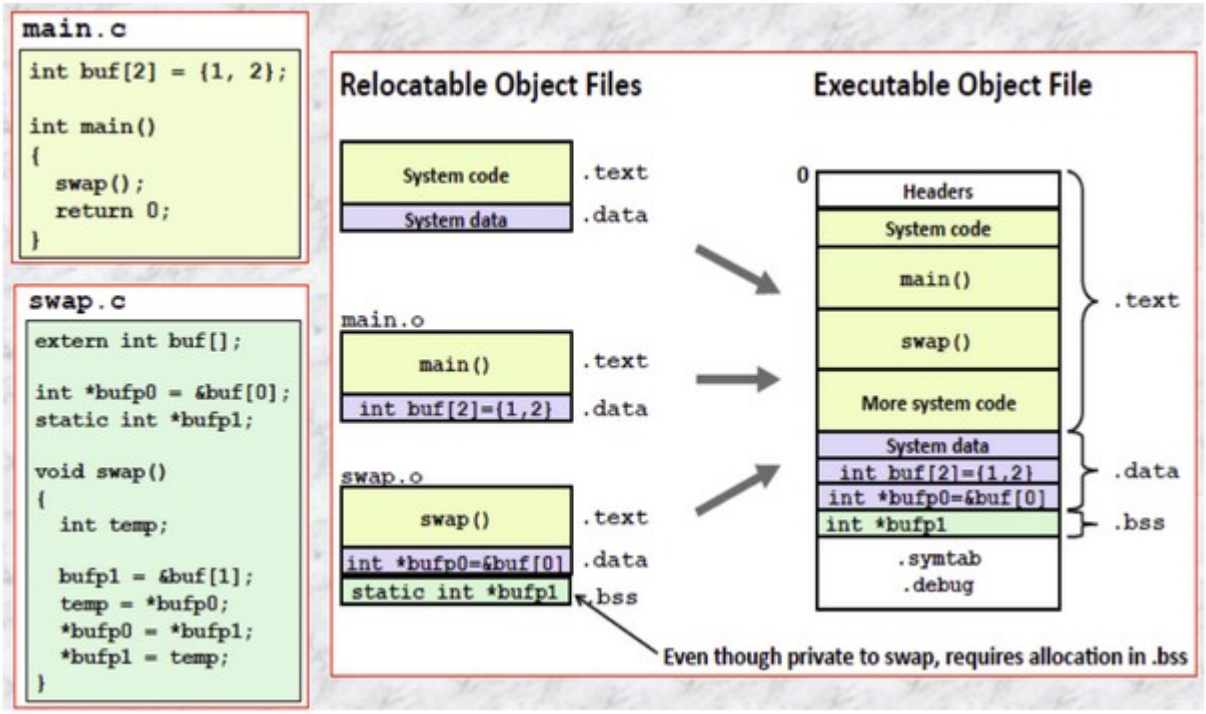
S	Type	Byte Count	Address	Data	CheckSum

```
S00F000068656C6C6F202020202000003C
S11F00007C0802A690 ····· 03863000026
S11F001C4BFFFFE539 ····· 64E800020E9
S11100384865 6C6C6F20776F726C642E0A0042
S5030003F9
S9030000FC
```

- S0 레코드 Data 값 : 'h'(0x68), 'e'(0x65), 'l'(0x6C), 'l'(0x6c), 'o'(0x6F), ' '(0x20), ' '(0x20), ' '(0x20), ' '(0x20), ' '(0x20), 0x00, 0x00
- S1 레코드 Address : 0x0000 ~ 0x0045,
- S1 레코드 Data 갯수 : 0x1C + 0x1C + 0x0E = 0x46 개
- S5 레코드 총 레코드 수 : 0x0003 개
- S9 레코드 : S1 레코드의 끝

1-2. ELF & HEX ?

2) ELF → Link → HEX ?



* 링커(Linker) : 프로그램은 여러 개의 파일로 나누어 질 수 있다. 각 파일은 각각 컴파일 되며 링커가 컴파일 된 각 파일을 합쳐서 하나의 기계어 프로그램으로 만들어줌

- 링킹 없이는 relocatable object files의 .o파일만으로는 실행이 불가능(주소 알 수 없기 때문)
- 컴파일러는 링커에게 symbol에 있을 주소를 넣어달라는 요청을 보냄
- 링커는 .o파일들을 쪼개서 실행 파일을 만든 다음 요청에 따라 적절한 주소를 적어 넣어줌

1-3. 프로그램 개발 전체 과정

1단계: PC에서 편집기를 통해 C나 어셈블리 소스파일 작성
C파일 확장자 .c / 어셈블리 확장자 .asm

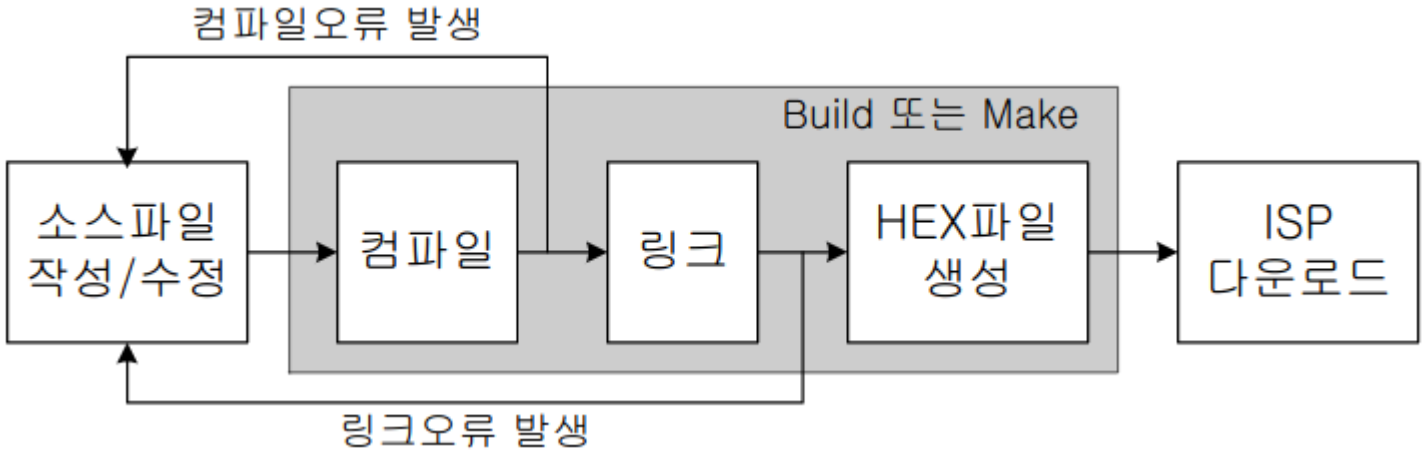
2단계: 크로스 컴파일러를 사용하여 소스파일을 컴파일함
컴파일 결과 생성되는 파일 .o
→ 오브젝트 파일은 주소관련 부분을 제외하고 모두 기계어로 번역된 파일
→ 주소 관련 부분은 링커가 정리

3단계: 링커를 사용해 실행파일을 만듦
실행파일 확장자 .elf

4단계: hex파일 컨버터를 사용해 hex파일 생성
hex파일 확장자 .hex

5단계: hex파일을 ISP를 사용하여 FLASH 메모리에 다운

여기서, winAVR, CodeVisionAVR같은 통합 환경에서는 2,3,4 단계를 “Build” or “Make”라는 명령으로 한 번에 수행



2. JTAG ?

1) Debugger 란?

- 프로그램 실행 중 여러 변수 레지스터의 상태 등을 보여주고, 프로그램의 내부 상황을 손쉽게 파악할 수 있어서 버그(문제점)수정 가능.

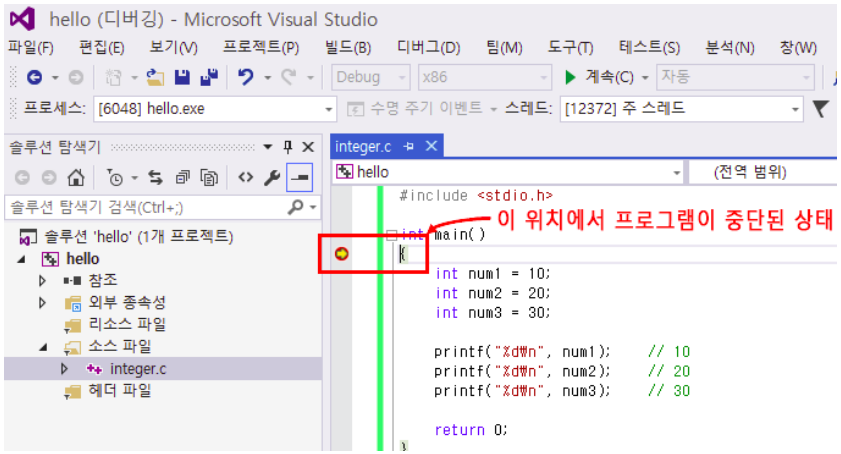
2) JTAG 란?

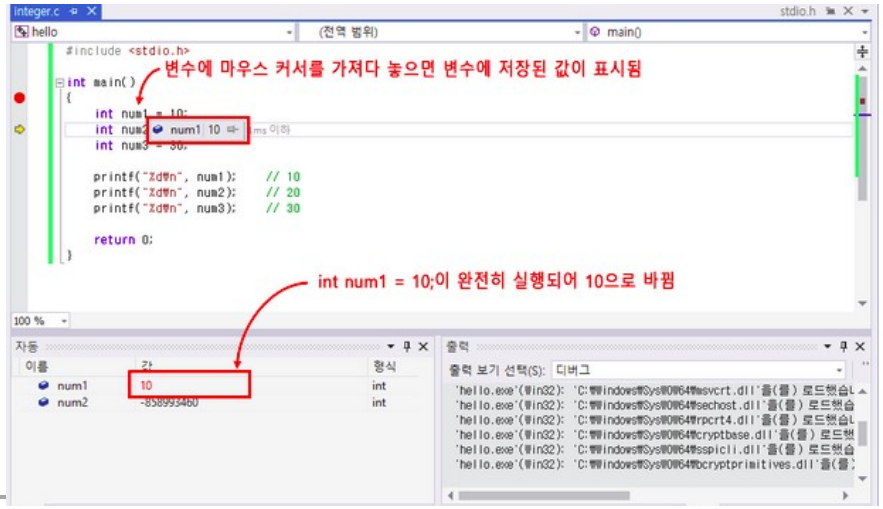
1. 디버깅용

- 임베디드 시스템에서 디버거는 소프트웨어 디버거와 하드웨어 디버거로 나누어짐
- 소프트웨어 디버거: 주로 시리얼 포트로 메시지 출력해서 디버깅을 함(단점: printf를 사용하여 출력확인, 매번 컴파일 해야함)
- 하드웨어 디버거: CPU를 브레이크(Break) 하거나 러닝(Running) 하면서 개발자가 직접 CPU를 조절하면서 C 소스 레벨 디버깅 가능(디버거툴 JTAG 등 사용)

2. 플래시 다운로더용

개발자가 C로 프로그램을 한 후 컴파일하고 나면 통합바이너리가 만들어지는데, 이 통합바이러리를 플래시 메모리에 다운로드하기 위해서 JTAG포트 사용.





3. Interrupt?

1) AVR Interrupt 란?

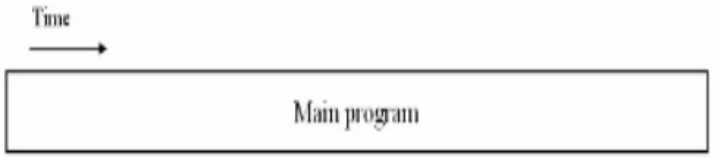
- 외부 인터럽트: 전원, 입출력장치, 주변장치 등과 같은 외부 장치에 의해서 발생하는 인터럽트다. (전원, CPU, 타이머 오버플로우, A/D변환완료, 시리얼 통신 수신 완료 등의 인터럽트)
-차단기능(INT): 인터럽트가 발생하였을때 무시할 수 있는 차단기능을 가짐
-벡터방식: 하드웨어 구조가 복잡하지만 응답 속도가 매우 빠르고 장치수에 영향을 거의 안받음

2) EI/DI 명령

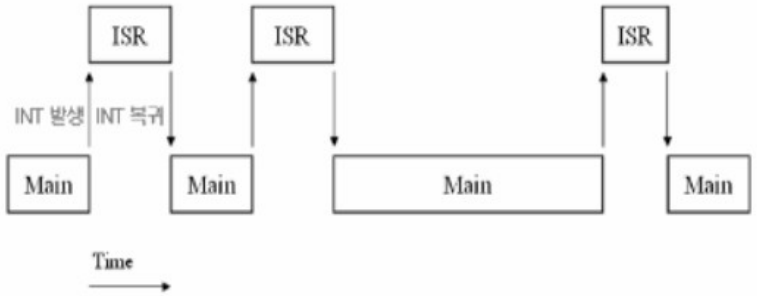
- 전체 인터럽트를 허용(EI:Enable Interrupt)/차단(Disable Interrupt) 명령을 사용
- EI명령: SEI / DI명령: CLI (C에서는 sei(),cli())
- 위 명령어 대신 SREG 레지스터의 I비트를 세트/ 클리어 해도 동일한 결과를 얻음

3)인터럽트 처리 과정

1. 인터럽트 요청- flag set
2. 인터럽트 허용 여부 판단 - 인터럽트 마스크 레지스터의 비트와 SREG 전역 인터럽트
3. 비트를 보고 혀용 여부 결정
4. 인터럽트 백터 주소로 점프 - 그 해당 주소로 점프함
5. 복귀 정보 저장 - 스택에 현재의 동작중인 프로그램 위치인 프로그램 카운터(PC)값을 저장
6. 인터럽트 서비스 루틴 수행 - ISP(인터럽트 함수) 수행
7. 주 프로그램으로 복귀 - 스택에서 프로그램 카운터 값을 찾아와 주 프로그램으로 복귀



〈 인터럽트가 없는 프로그램 실행 〉



〈 인터럽트가 있는 프로그램 실행 〉

3. 외부 Interrupt 제어 레지스터

* 외부 인터럽터 제어 레지스터
  - Atmega328p 외부 인터럽트는 2개
  - 외부 인터럽트 레지스터 종류 및 설명

외부 인터럽트 레지스터	설 명
MCUCR	MCU 제어 레지스터
EIMSK	외부 인터럽트 마스크 레지스터
EIFR	외부 인터럽트 플래그 레지스터
EICRA	외부 인터럽트 트리거 방식 설정 레지스터

1) MCUCR 란?

-인터럽트 벡터를 응용프로그램 색션과 부트로더 색션 사이에서 이동하기 위해 사용.

MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	BODS	BODSE	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

1. IVSEL (인터럽트 벡터 선택, Interrupt Vector Select)
   - •IVSEL = 0 : 인터럽트 벡터는 응용 프로그램 섹션인 플래시 메모리의 시작 부분에 위치
   - •IVSEL = 1 : 인터럽트 벡터는 부트 로더 섹션의 시작 부분에 위치
2. IVCE (인터럽트 벡터 변경 허가, Interrupt Vector Change Enable)
   - •IVSEL : 비트의 변경을 허가하기 위해서 IVCE 비트는 1로 설정되어 있어야 함.

3. 외부 Interrupt 제어 레지스터

2) EICRA 제어 레지스터

-외부 인터럽트 INT3~0핀으로 입력되는 신호에 대한 인터럽트 트리거 방식을 설정

-모든 레벨 트리거 인터럽트와 INT3~0이 Falling or rising 트리거 방식으로 설정시 인터럽트가 비동기적으로 검출, 슬립모드를 해제하는 수단으로 사용 가능.

The external interrupt control register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Value [ICSn1:ICSn0]	Description
00	Low level에서 Trigger 되도록 설정.
01	Rising/Falling Edge Trigger 되도록 설정.
10	Falling Edge에서 Trigger 되도록 설정.
11	Rising Edge에서 Trigger 되도록 설정.

```
/*EICRA 레지스터는 비트명에 대해 다음과 같이 선언*/

#define ISC00 0
#define ISC01 1
#define ISC10 2
#define ISC11 3
```

예) ISC11 = 1, ISC10 = 0으로 설정하면 '외부 인터럽트 1의 폴링 엣지시에 인터럽트를 발생.'

EICRA = 0x30; 또는 0b00110000; 과 같이 표현

3. 외부 Interrupt 제어 레지스터

3) EIMSK(External Interrupt Mask Register)

- INT1~0을 개별적으로 허용하는 레지스터
- 세트: 인터럽트 허용 / 클리어: 인터럽트 금지
- SREG(Status Register)의 I 비트가 set 되어야만 EIMSK가 허용 가능함

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	**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	

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	**INT1**	**INT0**	**EIMSK**
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

4) EIFR(External Interrupt Flag Register)

- INTF1~0핀에 인터럽트 신호가 입력되어 해당 인터럽트가 트리거 되었음을 표시하는 레지스터
- ISP(인터럽트 함수) 호출되면 0으로 클리어 됨
- 강제로 0을 클리어 하려면, 해당 비트에 1을 write하면 된다.

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	–	**INTF1**	**INTF0**	**EIFR**
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

3. 외부 Interrupt 제어 레지스터

* ISP 초기화 과정

→ EIMSK: (레지스터의 비트 설정을 통한) 사용하고자 하는 인터럽트 허가
→ EICRA: (레지스터의 비트 설정을 통한) 인터럽트 트리거 방식 설정
→ SREG의 I 비트의 설정을 통한 전체 인터럽트 허가

```
/*외부 INT0핀의 신호가 Falling edge에 의해 인터럽트를 발생하도록 초기화*/

void interrupt_init(void)
{
        EIMSK = 0X01;    //INT0 비트 설정(외부인터럽트0 허가)
        EICRA = 0x02;    //ISC01 = 1, ISC00 = 1(외부인터럽트 0 Falling edge 비동기 트리거)
        sei();           //전체 인터럽트 허가
}
```

3. 외부 Interrupt 제어 레지스터

* PCI(Pin Change Interrupt)란?

-핀이 상태를 변경 (L→H , H→ L)하는 경우 인터럽트를 트리거하는 23개의 핀이 존재함
(23개핀 = 3 개의 GPIO 포트 B, C, D = 3개의 인터럽트 그룹 (PCI 2~0))

4) PCICR(Pin Change Interrupt Control Register)

- PCIE[2:0]를 설정하여 PCINT 사용 여부를 결정 Set으로 설정할 경우 해당 Interrupt를 Enable시킴
- PCINT의 동작은 해당 Pin의 모든 입력 상태 변화에서 Interrupt가 발생되며 Group은 다음과 같음

PCIE2: PCINT[23:16] 사용여부를 결정, 해당 PCMSK2와 연동되어 사용
PCIE1: PCINT[14:8] 사용여부를 결정, 해당 PCMSK1와 연동되어 사용
PCIE0: PCINT[7:0] 사용여부를 결정, 해당 PCMSK0와 연동되어 사용

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

5) PCIFR(Pin Change Interrupt Flag Register)

- PCICEn Group에 속해있는 PCINTn에서 Interrupt 발생시 해당 PCIFn bit가 set으로 설정됨
- Clear를 위해서는 1을 write함

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

3. 외부 Interrupt 제어 레지스터

6) PCMSK(Pin Change Mask Register)

-외부 인터럽트에 대한 사용 여부 설정

-기본값은 00이며 SET설정으로 사용할 PCINTn 선택

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
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	

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

3. LED ON/OFF Code

1) C code with structure

```
#define F_CPU 16000000L             //CPU 16Mhz
#include <util/delay.h>
#include <avr/io.h>
#define PORTB_REG 0x23

struct io_port                      //구조체 선언
{

    unsigned char pin;              //8bit+8bit+8bit = 24bit
    uint8_t ddr;
    uint8_t port;

    };

int main(void)
{
    /* Replace with your application code */

    struct io_port *portB = (void*) PORTB_REG;  //Define된 0X23이 상수이므로 (void*)로 형변환을 시켜 주소 연산이 가능하게 함
    while (1)
    {
    //  DDRB = 0x20;

        PORTB -> ddr = 0x20;
        PORTB -> port = 0x20;
        _delay_ms(1000);
        PORTB -> port = 0x00;
        _delay_ms(1000);
    }
}
```

→ #define F_CPU 16000000L　　//CPU 16MHz frequency
→ #define PORTB_REG 0X23　　//직접 레지스터 할당하기
→ struct io_port *portB = (void*) PORTB_REG;　//정의해준 PORTB_REG가 상수이므로 주소 연산을 위해 (void*)형변환을 해줌

3. LED ON/OFF Code

1) iom328p.h 레지스터 확인

0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

맨앞: 물리주소
괄호안: 우리가 실제 사용하는 주소
레지스터 직접 할당: 0x23~0x25(8+8+8=24bit) 를 #define PORTB_REG 0X23 로 직접 레지스터 할당함 (주소값을 가짐)