5. 외부 인터럽트

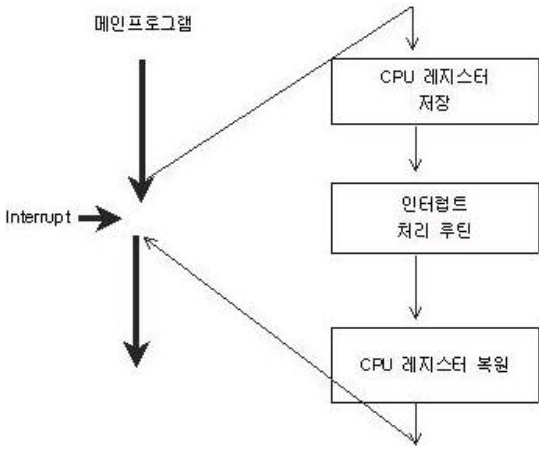
5.1 외부 인터럽트의 이해

이 절에서는 ATmega128의 인터럽트 처리 기능에 대하여 설명한다. 인터럽트 처리는 MCU의 매우 중요한 기능이므로 잘 알아두어야 하며 이를 제대로 응용할 수 있는 능력을 기르는 것이 필요하다. 그러나, 마이크로프로세서의 초보자에게는 인터럽트란 무엇인지 이해하는 것부터 쉽지 않기 때문에 여기서는 ATmega128의 인터럽트 처리 기능을 설명하기 전에 먼저 인터럽트의 기본을 알아보기로 한다.

5.1.1 인터럽트의 개요

5.1.1.1 인터럽트의 개념

CPU 내부의 하드웨어적인 요구에 의해서 정상적인 프로그램의 실행 순서를 변경하여 보다 시급한 작업을 먼저 수행한 후에 다시 원래의 프로그램으로 복귀하는 것을 인터럽트(interrupt, 가로채기)라고 한다. 이것은 흔히 우리가 책을 읽고 있는 도중에 전화가 와서 책의 읽던 페이지에 책갈피를 끼워 표시해두고 전화를 받은 다음에 다시 표시해 두었던 페이지를 찾아 책을 계속 읽어가는 경우에 비유될 수 있다. 인터럽트는 주변장치의 서비스 요청에 CPU가 가장 빠르게 대응할 수 있는 방법이며, 이것을 이용하면 주변장치 측으로 부터의 발생 시기를 예측하기 어려운 비동기적인 일(Event, 사건)을 CPU가 빠르게 처리할 수 있어서, 인터럽트는 서로 비동기적으로 동작하는 CPU(매우 고속으로 동작)와 주변장치(비교적 저속으로 동작) 사이에서 효율적으로 일을 수행하는 중요한 수단이 된다.



[그림 5.1] 인터럽트 처리의 개념도

인터럽트가 발생하면 서브루틴의 경우에서처럼 나중에 되돌아 올 복귀주소(Return Address)가 자동적으로 스택에 저장되었다가 인터럽트 서비스 루틴(Interrupt service routine)의 마지막에서 복귀(Return) 명령을 만나면 다시 자동적으로 이 복귀주소가 되 찾아져서 인터럽트 발생전의 위치로 정확하게 되돌아간다.

5.1.1.2 인터럽트의 개념

인터럽트의 종류는 마이크로프로세서에 따라 다르고, 또한 이를 보는 관점에 따라 여러 가지의 방법으로 분류할 수 있으나 일반적으로 다음과 같은 것들이 있다.

① 인터럽트 발생 원인에 따른 분류
- ▣ 하드웨어 인터럽트 : 내부 인터럽트, 외부 인터럽트
- ▣ 소프트웨어 인터럽트

② 인터럽트 발생시 마이크로프로세서의 반응 방식에 따른 분류
- ▣ 차단 가능 인터럽트(INT : Maskable Interrupt)
- ▣ 차단 불가능 인터럽트(NMI : Non-Maskable Intrrupt)

②인터럽트를 요구한 입/출력 기기를 확인하는 방법에 따른 분류
- ▣ 벡터형 인터럽트 (Vectored Interrupt)
- ▣ 조사형 인터럽트 (Polled Interrupt)

가. 내부 인터럽트

CPU에 정의되어 있지 않은 명령의 실행, 영(zero)으로 나눗셈을 시도하는 것과 같은 나눗셈 에러, 보호된 메모리 영역에의 접근 등의 원인에 의해 마이크로프로세서에서 내부적으로 발생되는 인터럽트로서, 마이크로프로세서에 따라서는 이를 execption 이라고 부르기도 한다. 그러나, 마이크로컨트롤러 소자에 내장되어 있는 타이머나 직렬 포트, A/D 컨버터, DMA 컨트롤러 등과 같은 주변장치에 의하여 발생된 인터럽트는 물리적으로는 내부 인터럽트가 될 것이지만, 이것들의 기본적인 성격이 CPU의 외부적 기능에 해당하는 것으로 보아 외부 인터럽트로 분류하는 것이 타당할 것이다.

이러한 내부 인터럽트는 신뢰성이 매우 중시되는 고성능의 마이크로컨트롤러나 다중 프로그래밍용의 CPU에서 많이 사용되며, ATmega128의 인터럽트 중에는 이러한 내부 인터럽트에 해당되는 것이 없다

나. 외부 인터럽트

타이머에서의 지정된 시간 경과, 입력 장치에서의 서비스 요구, 출력 장치의 작업 종료, A/D 변환의 완료, DMA 동작의 종료, 멀티프로세서간의 통신 요구 등 마이크로프로세서와 독립되어 있는 외부장치에 의해 발생되는 순수한 의미에서의 인터럽트이다. 일반적으로는 그냥 인터럽트라고 하면 대부분 이것을 지칭한다.

대부분의 마이크로프로세서에서 사용하고 있는 $\overline{\text{INT}}$ 및 $\overline{\text{NMI}}$ 인터럽트가 이에 해당하는데, ATmega128에서는 매우 특이하게 $\overline{\text{NMI}}$ 인터럽트를 사용하지 않으며 8개의 차단 가능한 외부 인터럽트 INT0~INT7만을 가지고 있다. ATmega128에서 사용하는 그밖의 주변장치 인터럽트들도 넓은 의미에서 보면 모두 외부 인터럽트로 분류할 수 있다.

다. 차단 가능 인터럽트

프로그래머에 의하여 인터럽트 요청을 받아들이지 않고 무시할 수 있는 것으로서 시간 제약이 있는 중요한 프로그램 수행중에는 인터럽트 요청을 허용하지 않을 수 있다. 보통 인터럽트를 차단하는 방법은 인터럽트 마스크 레지스터 또는 인터럽트 허용 레지스터를 사용하여 각각의 인터럽트를 개별적으로 차단할 수도 있고, DI(Disable Interrupt)명령을 사용하여 전체적으로 차단할 수도 있다. 또한 인터럽트 마스크 레지스터에서 개별적으로 허용된 인터럽트를 전체적으로 허용하는 명령은 EI(Enable Interrupt)이다.

ATmega128에서는 외부 인터럽트 INT0~INT7을 비롯한 모든 인터럽트가 차단 가능하며, 개별적인 인터럽트의 차단에는 EIMSK와 같은 인터럽트 제어 레지스터를 사용하고, 전체적으로 인터럽트를 허용 또는 금지하는데는 각각 SEI(Set Global Interrupt) Falg, Global Interrupt Enable)와 CLI(Clear Global Interrupt Flag, Global Interrupt Disable)명령을 사용한다.

라. 차단 불가능 인터럽트

프로그래머에 의하여 어떤 방법으로도 인터럽트 요청이 차단될 수 없는 것으로 전원 이상이나 비상정지 스위치 등과 같이 중요도가 높은 심각한 돌발 사태에 대비하기 위하여 주로 사용된다. 대부분의 마이크로프로세서에는 $\overline{\text{NMI}}$ 신호를 사용하여 이 기능이 수행되지만, ATmega128에서는 차단 불가능 인터럽트가 사용되지 않는다.

마. 벡터형 인터럽트

이것은 인터럽트가 발생할 때마다 인터럽트를 요청한 장치가 인터럽트 서비스 루틴의 시작번지(Interrupt Vector)를 CPU에게 전송하거나 또는 CPU가 각 인터럽트의 종류에 따라 미리 지정된 메모리 번지에서 인터럽트 벡터를 읽어서 이를 인터럽트 서비스 루틴의 시작번지로 사용하는 방식이다. 따라서, 이 방식에서는 주변장치가 CPU에게 인터럽트 요청 신호를 보내는 방법이나 또는 인터럽트 제어기에서 이 신호를 처리하는 방법에 의하여 인터럽트의 우선순위가 결정된다.

이 방법은 인터럽트 벡터에 의하여 즉시 인터럽트 서비스 루틴을 찾아가므로 인터럽트 응답 시간이 빠르며, 이 인터럽트 응답시간이 주변장치의 수가 많고 적음에 영향을 받지 않는다. 이와 같은 장점 때문에 빠른 인터럽트 처리가 요구되는 마이크로컨트롤러에서는 대부분 이 방식을 사용하며, ATmega128에서의 인터럽트 처리도 모두 이와 같은 벡터형 인터럽트 방식으로 되어 있다.

바. 조사형 인터럽트

이것은 인터릅트가 발생하면 이 인터럽트를 요청한 장치를 찾아내기 위하여 CPU가 각 주변장치를 소프트웨어적으로 차례로 조사(Polling)하는 방식이다. 이 때 각 장치는 상태 레지스터(Status Register)의 특정한 비트에 인터럽트 요청 사실을 표시해 놓음으로써 자신이 인터럽트를 요청하였음을 CPU가 알 수 있도록 한다. 따라서, 이 방식에서는 폴링의 순서에 의하여 소프트웨어적으로 인터럽트의 우선순위가 결정된다.
이 방법은 하드웨어가 간단한 것이 장점이므로 주변 단말장치가 많은 범용 컴퓨터에서 많이 사용하지만, 인터럽트가 발생될 때마다 각 장치를 조사해야 하므로 주변장치의 수가 많을수록 소프트웨어적인 처리 시간이 길어지고 따라서 인터럽트 응답이 늦어지는 단점이 있다. ATmega128에서는 이를 사용하지 않는다.

5.1.1.3인터럽트의 우선순위 제어

인터럽트는 CPU의 의지가 아니라 주변장치의 필요에 의하여 CPU와 비동기적으로 또한 비정기적으로 발생되는 것이 일반적이므로 우연히 2개 이상의 주변장치가 동시에 CPU에게 신호를 보내 인터럽트를 요청하는 경우가 있을 수 있다. 이러한 경우에 CPU는 이들 인터럽트를 한꺼번에 처리할 수가 없으므로 한번에 1개만의 인터럽트를 선택하여 처리하게 되는데 이를 인터럽트 우선순위(Interrupt Priority) 제어라고 한다. 이밖에 하나의 인터럽트가 서비스되고 있는 동안에 또 다른 인터럽트가 요청되면 이를 어떻게 처리할 것인지도 인터럽트 우선순위 제어에 해당한다.
동시에 요청된 인터럽트의 우선순위를 결정하는 방법은 마이크로프로세서에 따라 또는 인터럽트 방식에 따라 다르다.

가. 인터럽트 제어기를 사용하는 벡터형 인터럽트의 경우

대부분의 마이크로컨트롤러에서는 많은 종류의 인터럽트를 사용하고 있으며, 이를 효율적으로 관리하기 위하여 내부에 인터럽트 제어기를 가지고 있다. 이 인터럽트 제어기에는 인터럽트 마스크 레지스터 또는 인터럽트 허용 레지스터를 가지고 있어서 인터럽트 허용 여부를 설정할 수도 있고, 인터럽트 우선순위 제어 레지스터를 가지고 있기도 하다.

Interl 사의 80x86 계열 범용 마이크로프로세서에서는 CPU의 내부에 인터럽트 제어기를 가지고 있지 않고, 외부에 주변 LSI로서 8259A 인터럽트 제어기를 사용하여 인터럽트를 제어한다. 모든 주변장치들은 CPU에게 직접 인터럽트를 요청하지 않고 8259A에게 인터럽트를 요구하며, 1개 또는 2개 이상의 인터럽트가 요청되면 8259A는 미리 CPU에 의하여 소프트웨어로 지정된 우선순위에 따라 1개의 주변장치에 대한 인터럽트만을 선택하여 CPU에게 인터럽트 요구를 전달한다. CPU로부터 인터럽트 요청이 받아들여지면 역시 미리 초기화되어 있던 해당 주변장치에 대한 인터럽트 벡터를 CPU로 전송한다.

이처럼 벡터형 인터럽트에서는 대부분 인터럽트 제어기의 우선순위 제어 레지스터를 초기화함으로써 역시 손쉽게 동적 우선순위 제어 방식을 사용하고 있다. 그러나, ATmega128의 인터럽트에는 우선순위 제어기능이 없으며 각 인터럽트마다 하드웨어적으로 우선순위가 정해져 있다.

나. 조사형 인터럽트의 경우

조사형 인터럽트는 인터럽트가 발생하면 이 인터럽트를 요청한 장치(Interrupting Device)를 찾아내기 위하여 CPU가 각 주변장치를 소프트웨어적으로 차례로 폴링하는 방식이므로 2개 이상의 주변장치가 인터럽트를 요청한 경우에는 폴링의 순서에 의하여 소프트웨어적으로 인터럽트의 우선순위가 결정되며, 따라서 필요할 때마다 소프트웨어를 수정하여 손쉽게 우선순위를 변경할 수 있다.

5.1.1.4 인터럽트의 처리 과정

인터럽트의 처리과정은 마이크로프로세서의 종류에 따라 다르고 또한 하나의 마이크로프로세서에서도 인터럽트의 종류에 따라 그 처리과정이 달라지는 경우가 있지만 전체적인 처리 과정은 서로 상당히 유사하다. 일반적인 인터럽트의 처리 과정을 요약하면 [표 5.1]과 같으며, 물론 ATmega128에서도 전체적인 처리과정은 이와 유사하다.

◉ **[표 5.1] 인터럽트의 처리과정**

1. 인터럽트 요청 신호의 검출	CPU는 모든 명령 사이클에 한번씩 또는 모든 머신 사이클에 한번씩 지정된 순간에 인터럽트를 샘플링하여 주변장치의 인터럽트 요청을 검출한다. (ATmega128에는 모든 인터럽트마다 그에 대응되는 인터럽트 플래그 레지스터를 가지고 있어서 인터럽트가 검출되면 이 플래그가 1로 셋된다. 이 플래그 비트는 해당 인터럽트의 벡터가 인출되어 인터럽트 서비스 루틴의 실행이 개시되면 자동으로 0으로 클리어 된다.)
⬇	
2. 인터럽트 우선순위 제어 및 허용 여부 판단	CPU가 인터럽트를 허용할 수 있는 조건이 만족되는지를 판단하고 우선순위를 결정하여 인터럽트를 허용한다. 외부에서 인터럽트 벡터를 읽어들이는 방식에서는 이를 위한 인터럽트 인정 사이클을 수행한다. (ATmega128에서는 각 인터럽트에 해당되는 인터럽트 마스크 레지스터와 SREG 레지스터의 글로벌 인터럽트 허용 비트를 보고 인터럽트 허용 여부를 판단한다)
⬇	
3. 인터럽트 처리루틴의 시작 번지 확인	CPU가 인터럽트를 요청한 주변장치에 대응하는 인터럽트 서비스 루틴의 시작번지를 확인한다. 각 인터럽트의 종류에 따라 이 벡터가 지정되어 있는 번지가 미리 지정되어 있는 경우도 있고, 또 다른 방식에서는 주변장치로 부터 읽어들인 인터럽트 벡터가 이 서비스 루틴의 시작번지를 결정하는 자료로 사용된다. (ATmega128에서는 각 인터럽트별로 인터럽트 벡터의 번지가 미리 정해져 있어서 여기에 사용자가 인터럽트 서비스 루틴의 시작번지나 또는 그곳으로의 점프 명령을 저장하여 두어야 한다)
⬇	
4. 복귀주소 및 레지스터를 저장	CPU가 나중에 인터럽트 서비스 루틴을 종료하고 되돌아 올 수 있도록 복귀주소(현재의 PC 값)를 스택에 저장하고, 마이크로프로세서에 따라서는 현재의 레지스터 값을 자동으로 스택에 저장하는 것도 있다. 또한, 대부분의 경우 하나의 인터럽트가 서비스되고 있는 동안에는 다른 인터럽트가 허용되지 못하도록 일시적으로 인터럽트 금지 상태로 설정된다. (ATmega128은 복귀주소만을 스택에 저장하며 다른 레지스터는 저장하지 않는다)
⬇	
5. 인터럽트 서비스 루틴을 실행	CPU가 해당 인터럽트 서비스 루틴으로 점프하여 프로그램을 실행한다. 만약, 현재의 인터럽트 서비스 루틴이 실행되는 동안에 어떤 인터럽트를 허용하도록 하려면 여기서 그 인터럽트 설정을 새로 해주어야 한다.
⬇	
6. 인터럽트 서비스 루틴을 종료하고 원래의 주프로그램으로 복귀	CPU가 인터럽트 서비스 루틴의 실행중에 리턴 명령을 만나면 스택에서 복귀주소를 되찾아 프로그램 카운터에 로드함으로써 원래의 위치로 되돌아온다. 이 때 복귀주소와 함께 레지스터를 저장하는 방식이면 이 레지스터들의 값도 되찾으며, 일시적으로 인터럽트 금지 상태로 설정되었던 것도 원상태로 되돌려진다. (ATmega128의 경우에는 RETI 명령을 만나면 해당 인터럽트를 종료하고 주프로그램으로 복귀한다)

5.1.2 ATmega128의 인터럽트 처리 동작

ATmega128에는 리셋을 포함하여 모두 35종의 리셋 및 인터럽트 벡터를 가지고 있다. 이것들은 여러 가지의 인터럽트 마스크 레지스터를 통하여 각각 개별적으로 허용여부를 설정할 수 있으며, 상태 레지스터 SREG의 글로벌 인터럽트 허용 비트 I를 이용하여 전체적인 허용여부를 설정할 수 있다.

◉ [표 5.2] 리셋 및 인터럽트 벡터

벡터번호 (우선순위)	벡터번지	인터럽트 소스	인터럽트 발생 조건
0	0x0000	리셋	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
1	0x0002	INT0	External Interrupt Request 0
2	0x0004	INT1	External Interrupt Request 1
3	0x0006	INT2	External Interrupt Request 2
4	0x0008	INT3	External Interrupt Request 3
5	0x000A	INT4	External Interrupt Request 4
6	0x000C	INT5	External Interrupt Request 5
7	0x000E	INT6	External Interrupt Request 6
8	0x0010	INT7	External Interrupt Request 7
9	0x0012	TIMER2 COMP	Timer/Counter2 Comare Match
10	0x0014	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0016	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x001C	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001E	TIMER0 COMP	Timer/Counter0 Compare Match
16	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
17	0x0022	SPI, STC	SPI Serial Tranfer Complete
18	0x0024	USART0, RX	USART0, Rx Complete
19	0x0026	USART0, UDRE	USART0 Data Register Empty
20	0x0028	USART0, TX	USART0, Tx Complete
21	0x002A	ADC	ADC Conversion Complete
22	0x002C	EE READY	EEPROM Ready ∴
23	0x002E	ANALOG COMP	Analog Comparator
24	0x0030	TIMER1 COMPC	Timer/Counter1 Compare Match C
25	0x0032	TIMER3 CAPT	Timer/Counter3 Capture Event
26	0x0034	TIMER3 COMPA	Timer/Counter3 Compare Match A
27	0x0036	TIMER3 COMPB	Timer/Counter3 Compare Match B
28	0x0038	TIMER3 COMPC	Timer/Counter3 Compare Match C
29	0x003A	TIMER3 OVF	Timer/Counter3 Overflow
30	0x003C	USART1, RX	USART1, Rx Complete
31	0x003E	USART1, UDRE	USART1 Data Register Empty
32	0x0040	USART1, TX	USART1, Tx Complete
33	0x0042	TWI	Two-wire Serial Interface
34	0x0044	SPM READY	Store Program Memory Ready ∴

※ ATmega103 호환모드에서는 인터럽트 벡터 24~34가 사용되지 않음

※ ∴표시는 인터럽트 플래그 비트를 가지고 있지 않는 종류의 인터럽트임.

5.1.2.1 인터럽트의 종류

ATmega128의 인터럽트는 [표 5.2]와 같이 모두 34종이 있는데, 이것들은 모두 외부 인터럽트에 해당하며 차단 가능하다. 이들 인터럽트를 다시 구분해보면 외부 핀을 통하여 요구되는 인터럽트 8개, 타이머0 관련 2개, 타이머1 관련 5개, 타이머2 관련 2개, 타이머3 관련 5개, USART0 관련 3개, USART1 관련 3개, 기타 6개이다.

이들 인터럽트에는 크게 2가지의 동작 유형이 있다.

첫째는, 인터럽트가 발생하면 관련 플래그 레지스터의 해당 플래그 비트를 1로 셋하도록 트리거시키는 종류이다. 이러한 인터럽트에서는 인터럽트 벡터가 인출되어 프로그램 카운터가 인터럽트 서비스 루틴의 시작번지를 가리키게 되면 자동으로 해당 플래그가 다시 0으로 클리어된다. 물론 소프트웨어로 이를 0이 되도록 라이트할 수도 있다. 이들 인터럽트에서는 인터럽트 마스크 레지스터 또는 SREG 레지스터에서 이를 금지상태로 설정하여 놓았더라도 인터럽트가 발생하면 해당 인터럽트 플래그가 1로 되어 인터럽트 대기상태로 되며, 나중에라도 인터럽트가 허용상태로 설정되면 이 인터럽트가 처리된다.

둘째는, 인터럽트가 요청되더라도 인터럽트 플래그를 셋시키지 않으며 인터럽트 발생 조건이 만족되어 있는 경우에만 인터럽트를 트리거하는 종류이다. 이러한 인터럽트는 인터럽트 발생조건이 사라지면 인터럽트 요청도 없어지므로 나중에 인터럽트가 다시 허용상태로 되더라도 인터럽트는 걸리지 않는다.

5.1.2.2 인터럽트의 동작

ATmega128의 어느 인터럽트가 요청되어 이것이 허용되면 인터럽트 서비스 루틴(Interrupt Service Routine, Interrupt Handller)이 실행되면서 SREG 레지스터의 글로벌 인터럽트 허용 비트Irk 0으로 클리어되어 모든 인터럽트가 금지상태로 된다. 따라서, 여기서 사용자가 인터럽트 서비스 루틴이 실행되는 동안에 다른 인터럽트가 발생되도록 다중 인터럽트를 허용하려면 SREG 레지스터의 I 비트를 1로 설정하여야 한다. 인터럽트 서비스 루틴을 종료하기 위하여 복귀명령 RETI를 실행하면 SREG 레지스터의 글로벌 인터럽트 허용 비트 Irk 다시 1로 되살아나 인터럽트 허용상태로 복구된다.

이와 같이 하나의 인터럽트가 처리되는 동안에 SREG 레지스터의 글로벌 인터럽트 허용 비트가 잠정적으로 0으로 되었다가 나중에 인터럽트가 종료되면 다시 1로 복구되지만, 이 과정에서 SREG 레지스터가 자동으로 저장되는 것은 아니라는데 유의해야 한다. 따라서, 인터럽트 처리과정에서 SREG 레지스터의 내용이 변경되지 않고 안전하게 유지되도록 하기 위하여 인터럽트 서비스 루틴에서 이를 저장하거나 되찾는 것은 사용자의 몫이다.

※ ATmega128에서 PUSH 및 POP 명령의 오퍼랜드로는 단지 R0~R31만을 사용할 수 있다.
따라서, I/O 영역에 위치하는 상태 레지스터 SREG를 직접 스택에 PUSH/POP하는 것은 불가능하므로 다른 레지스터나 데이터 메모리에 저장해야 한다.

ATmega128이 인터럽트 서비스 루틴을 종료하고 원래의 프로그램으로 복귀하면 이미 대기 상태에 있던 다른 인터럽트를 수행하기 전에 최소한 1개 이상의 명령을 실행한다. CLI 명령을 실행하면 어떤 인터럽트도 수행되지 않으므로 이 명령이 실행될 때 동시에 요청된 인터럽트도 처리되지 않고 대기 상태로 된다. SEI 명령을 실행하면 그 다음에 있는 최소 1개의 명령을 실행하고 나서 대기 상태에 있던 인터럽트가 수행된다.

ATmega128이 인터럽트 요청에 응답하는데는 최소한 4클럭 사이클이 소요된다. CPU는 이 4클럭 사이클 동안에 프로그램 카운터를 스택에 저장하고, 인터럽트 벡터를 인출하여 서비스 루틴으로 점프한다. 그러나, 만약 여러 사이클에 수행되는 명령을 실행하고 있는 동안에 인터럽트가 요청된다면 이 명령이 완료된 이후에 인터럽트가 처리되므로 전체적인 인터럽트 응답시간은 4클럭 사이클보다 더 길어진다. 또한, 슬립 모드에 있을 때 인터럽트가 발생하면 인터럽트 으답시간은 더 길어지며, 슬립모드에 따라 여기에 기동시간이 추가된다.

ATmega18이 RETI 명령에 의하여 인터럽트 서비스 루틴의 실행을 마치고 복귀하는데도 4클럭 사이클이 소요된다. 이 시간동안에 프로그램 카운터의 값이 스택에서 꺼내지고, 스택 포인터가 2만큼 증가하며, SREG 레지스터의 글로벌 인터럽트 허용 비트 I가 다시 1로 셋 된다.
ATmega128의 인터럽트 우선순위는 [표 5.2]에 표시된 것처럼 고정되어 있으며, 사용자가 이를 변경할 수 없다.

5.1.2.3 리셋 및 인터럽트 벡터의 배치

ATmega128에서 리셋 및 인터럽트 벡터는 항상 [표 5.2]처럼 위치가 고정되어 있는 것이 아니라 가변적으로 배치할 수 있다. 즉, Fuse High Byte의 BOOTRST 비트 및 MCUCR 레지스터의 IVSEL 비트에 따라 [표 5.3]과 같이 여러 가지의 조합이 가능하다. 여기서 Boot Reset Address는 부트 로더 섹션의 크기에 따라 달라지는데, Fuse High Byte의 BOOTSZ1~0 비트에 의하여 설정된다. 예를 들어 BOOTSZ1~0 비트들이 모두 0으로 설정되면 부트 사이즈는 8KB(4096워드)가 되어 Boot Reset Address는 0xF0000 번지가 된다.

◉ **[표 5.3] 리셋 및 인터럽트 벡터의 위치**

BOOTRST	IVSEL	리셋 벡터 어드레스	인터럽트 벡터의 시작 어드레스
1	0	0x0000	0x0002
1	1	0x0000	Boot Reset Address + 0x0002
0	0	Boot Reset Address	0x0002
0	1	Boot Reset Address	Boot Reset Address + 0x0002

① MCUCR(MCU Control Register) 레지스터

MCUCR 레지스터는 인터럽트의 벡터의 위치를 설정하고 또는 외부 메모리를 사용할 경우에 웨이트 사이클의 개수를 설정할 때 사용한다.

MCUCR(MCU Control Register) 0x35 (0x55)

bit	7	6	5	4	3	2	1	0
	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초기값	0	0	0	0	0	0	0	0

▣ Bit 1 : IVSEL (Interrupt Vector Select)

·IVSEL = 0 : 인터럽트 벡터는 플래시 메모리의 시작점($0000)에 위치하게 된다.

·IVSEL = 1 : 인터럽트 벡터는 플래시 메모리의 Boot Loader Section의 시작점으로 이동한다. Boot Loader Section의 시적점의 어드레스는 BOOTSZ 비트에 의해서 결정되며, 인터럽트 벡터의 예기치 않은 위험을 방지하기 위해서 IVSEL 비트를 설정하는 순서는 다음과 같다.

a. IVCE(Interrupt Vector Change Enable) 비트에 “1”을 쓴다

b. 4사이클 이내에 IVCE=0 으로 설정하고, IVSEL에는 원하는 비트를 설정한다

이와 같은 순서로 실행하면 인터럽트는 자동으로 disable 되는데, 정확하게는 IVCE가 “1‘로 셋되는 사이클에서 인터럽트가 disable로 된다.

▣ Bit 0 : IVCE (Interrupt Vector Change Enable)

·IVCE = 1 : IVSEL 비트를 변경할 수 있다. IVCE 비트는 IVSEL 비트를 변경하고 난 후 4사이클이 지나면 하드웨어에 의해서 “0”으로 클리어된다. 다음에 그에 해당하는 예제를 나타내었다.

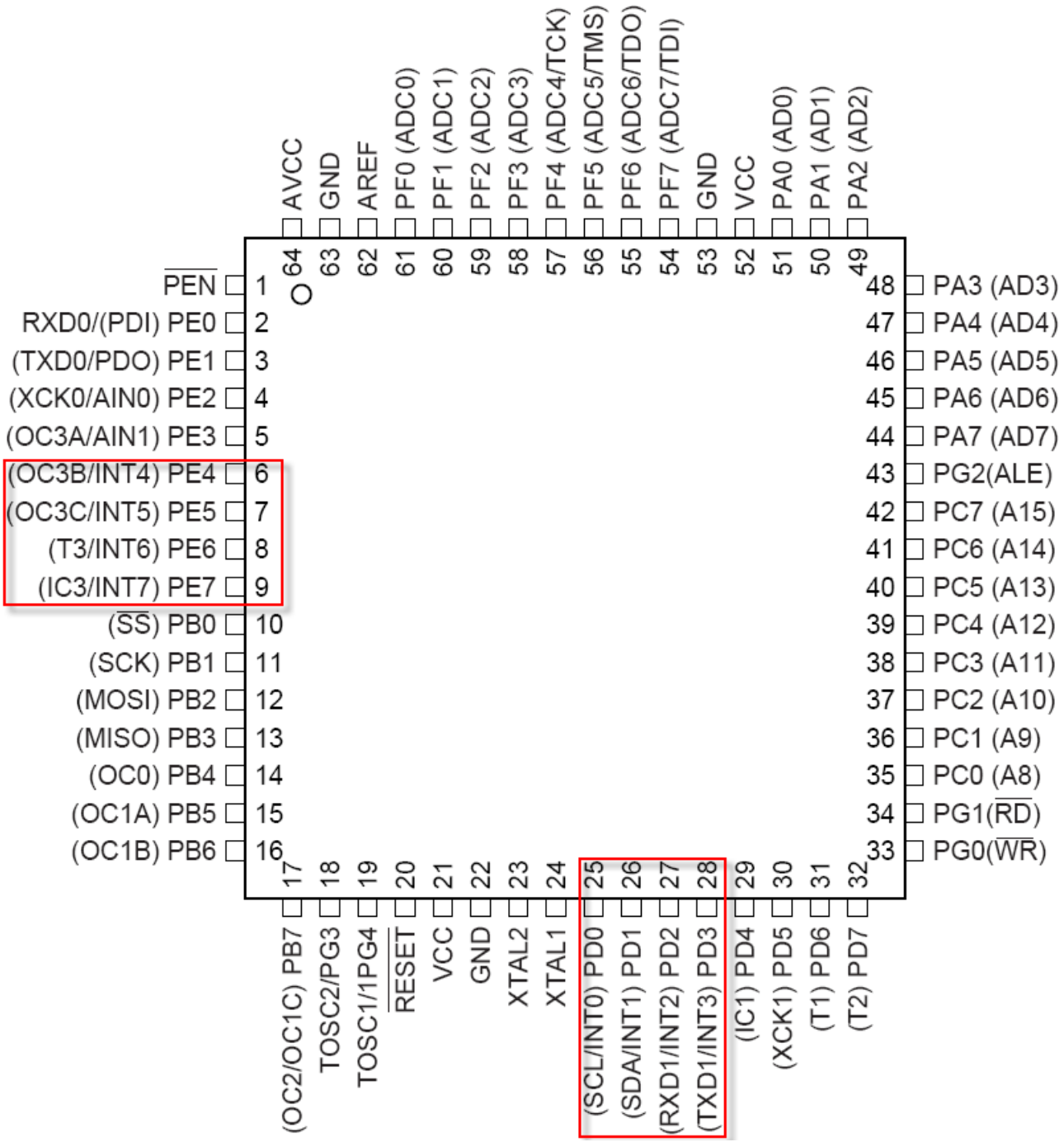
MCUCR 레지스터의 IVCE 비트 예제

```
void Move_interrupt(void)
{
    //Enable change of interrupt vectors
    MCUCR = (1<<IVCE);

    // Move interrupt to boot Flash section
    MCUCR = (1<<IVSEL);
}
```

5.1.2.4 AVR ATmega128의 외부 인터럽트

AVR ATmega128의 외부 인터럽트는 INT0~INT7 핀에 이벤트가 인가됨으로써 발생한다. 이벤트(Event)는 크게 레벨 방식(Level mode)과 에지 방식(Edge mode)으로 나뉜다. AVR ATmega128의 외부 인터럽트는 하강 에지, 상승 에지, 로 레벨(Low level)에서 발생될 때 인터럽트가 발생된다. 이것을 설정하기 위해서 EICRA(External Interrupt Control Register A)와 EICRB(External Interrupt Control Register B)를 선택한다. [그림 5.2]에 AVR ATmega128의 외부 인터럽트 핀을 나타내었다. INT0..7 핀이 출력으로 설정되어 있어도 인터럽트는 발생된다. 즉 DDRn을 통해 I/O의 방향을 별도로 설정할 필요가 없다.



[그림 5.2] AVR ATmega128의 외부 인터럽트 핀

① EICRA(External Interrupt Control Register A) 레지스터

EICRA(External Interrupt Control Register A) 레지스터는 interrupt sense control 및 MCU의 일반적인 기능을 설정하는 데 사용한다.

EICRA(External Interrupt Control Register A) 0x6A

bit	7	6	5	4	3	2	1	0
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초기값	0	0	0	0	0	0	0	0

▣ Bit 7~0-ISC31, ISC30~ISC01, ISC00 (External Interrupt 3~0 Sense Control 비트)

외부 인터럽트 3~0은 전역 인터럽트(SREG의 I 비트)와 EIMSK의 개별 인터럽트(INT3~INT0)를 설정함으로써 발생된다. 레벨과 에지에 대해서는 [표 5.5]에 나타내었다.

◉ [표 5.5] Interrupt Sense Control

ISCn1	ISCn0	설 명
0	0	INTn의 Low level에서 인터럽트를 발생한다.
0	1	예약
1	0	INTn의 하강 에지에서 인터럽트를 발생한다.
1	1	INTn의 상승 에지에서 인터럽트를 발생한다.

② EICRB(External Interrupt Control Register B) 레지스터

EICRB(External Interrupt Control Register B) 레지스터는 다기능 비트로 구성되어 있다.

EICRB(External Interrupt Control Register B) 0x5A

bit	7	6	5	4	3	2	1	0
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초기값	0	0	0	0	0	0	0	0

▣ Bit 7~0-ISC71, ISC70~ISC41, ISC40 (External Interrupt 7~4 Sense Control 비트)

외부 인터럽트 7~4는 전역 인터럽트(SREG의 I 비트)와 EIMSK의 개별 인터럽트(INT7~INT4)를 설정함으로써 발생된다. 레벨과 에지에 대해서는 [표 5.6]에 나타내었다.

◉ [표 5.6] Interrupt Sense Control

ISCn1	ISCn0	설 명
0	0	INTn의 Low level에서 인터럽트를 발생한다.
0	1	INTn의 핀에 논리적인 변화가 발생할 경우
1	0	INTn의 하강 에지에서 인터럽트를 발생한다.
1	1	INTn의 상승 에지에서 인터럽트를 발생한다.

③ EIMSK(External Interrupt Mask Register) 레지스터

EIMSK(External Interrupt Mask Register) 레지스터는 INT0~INT7의 개별 인터럽트를 설정한다.

EIMSK(External Interrupt Mask Register) 0x39 (0x59)

bit	7	6	5	4	3	2	1	0
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초기값	0	0	0	0	0	0	0	0

▣ Bit 7~0-INT7~INT0 : External Interrupt Request 7~0 Enable

이 비트에 “1”을 쓰고, SREG 레지스터의 I 비트가 “1”로 설정되어 있으면 외부 인터럽트는 enable된다. EICRA와 EICRB레지스터의 ISCn1과 ISCn0의 비트를 설정함으로써 레지 또는 레벨 방식을 선택할 수 있다.

④ EIFR(External Interrupt Flag Register) 레지스터

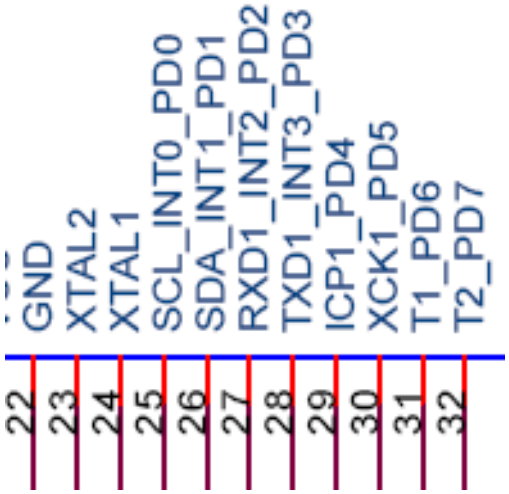
EIFR(External Interrupt Flag Register)레지스터는 EIMSK 레지스터에서 설정한 개별 인터럽트의 상태를 나타낸다.

EIFR(External Interrupt Flag Register) 0x38 (0x58)

bit	7	6	5	4	3	2	1	0
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초기값	0	0	0	0	0	0	0	0

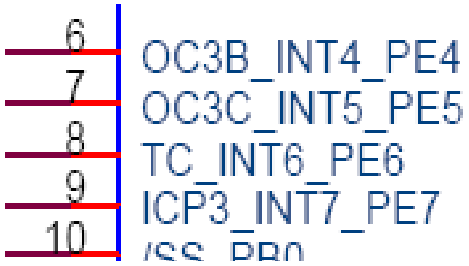
▣ Bit 7~0-INTF7~INTF0 : External Interrupt Flag 7~0

INT7~INT0 핀에 에지 또는 논리적인 변화에서 트리거되어 인터럽트가 요구되면, INTF7~INTF0 비트는 “1”로 셋된다. SREG 레지스터의 I 비트와 EIMSK 레지스터의 INT7~INT0 비트가 “1‘로 설정되어 있으면, MCU는 해당하는 인터럽트 벡터로 점프한다. 인터럽트 서비스 루틴(ISR)이 실행되면 INTF7~INTF0 비트는 자동으로 ”0“으로 클리어되고, 이 INTF7~INTF0 비트에 논리적으로 ”1“을 쓰면 클리어가 된다. INT7~INT00이 레벨 인터럽트로 설정되면, INTF7~INTF0 비트는 자동으로 클리어 된다.



[그림 5.3] INT0~INT3

[그림 5.3]은 INT0~INT3이 25~28번 핀을 사용할 수 있는 것을 보여준다. 하지만 이 핀들은 다른 용도와 겹치기 때문에 DK128에서는 INT4~INT5를 쓰는 것을 권장하고 있다. 아래 그림은 INT4~INT7이 6~9번 핀을 사용할 수 있는 것을 보여주고 있다.



[그림 5.4] INT4~INT7

5.2.1 학습예제

5.2.1.1 임의의 KEY를 눌렀다 떼는 순간부터 1초간 모든 LED가 켜졌다 꺼지는 것을 보여라

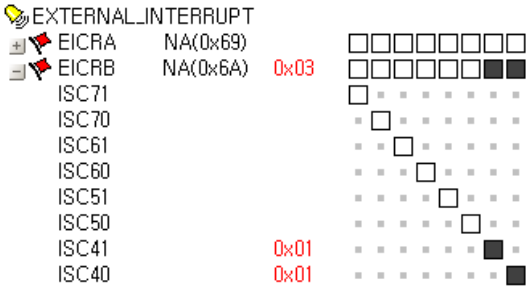
ex3_1.c 임의의 KEY를 눌렀다 떼는 순간부터 1초간 모든 LED가 켜졌다 꺼지는 예제

```
1   #include <avr/io.h>
2   #include <avr/interrupt.h>
3
4   // PF0 ~ PF7은 61 ~ 54번 핀
5   #define DDR_LED             DDRF
6   #define PORT_LED            PORTF
7   #define PIN_LED             PINF
8
9   // 전역 변수 선언
10  volatile int i, j;
11  volatile char int_flag;
12
13  // INT4의 외부 인터럽트 함수 PIN 6
14  SIGNAL(SIG_INTERRUPT4)
15  {
16          int_flag = 1;
17  }
18
19  int main(void)
20  {
21          DDR_LED = 0xFF;             // LED 포트를 출력 모드로 설정
22          PORT_LED = 0xFF;            // LED 포트 초기화
23
24          // INT4을 상승 에지 인터럽트 설정
25          EICRB = (1 << ISC41) | (1 << ISC40);
26          // INT4을 활성화
27          EIMSK = (1 << INT4);
28
29          sei();                      // 글로벌 인터럽트 활성화 설정
30
31          for (;;)
32          {
33                  if (int_flag==1)
34                  {
35                          int_flag = 0;
36                          PORT_LED = 0x00;
37
38                          // 1000ms 동안 시간 지연
39                          for(i=0; i<1000; ++i)
40                                  for(j=0; j<600; ++j)
41                                          ;
42
43                          PORT_LED = 0xFF;
44                  }
45          }
46
47          return 1;
48  }
```

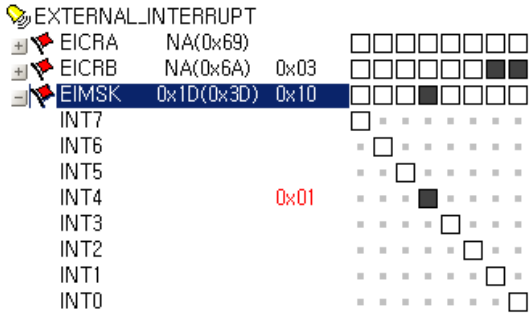
11행: volatile char형 변수 int_flag는 인터럽트가 발생한 것을 체크하기 위한 변수이다. int_flag는 인터럽트가 발생하면 1로 되고, 인터럽트 처리가 끝나면 0이 된다.

14~17행: 외부 인터럽트 INT4로 인터럽트가 발생할 때 수행하는 인터럽트 루틴이다. int_flag를 1로 설정하여 인터럽트가 발생한 것을 저장한다.

25행: INT4가 상승 에지에서 인터럽트를 발생 시키기 위한 EICRB 레지스터의 설정 값이다.



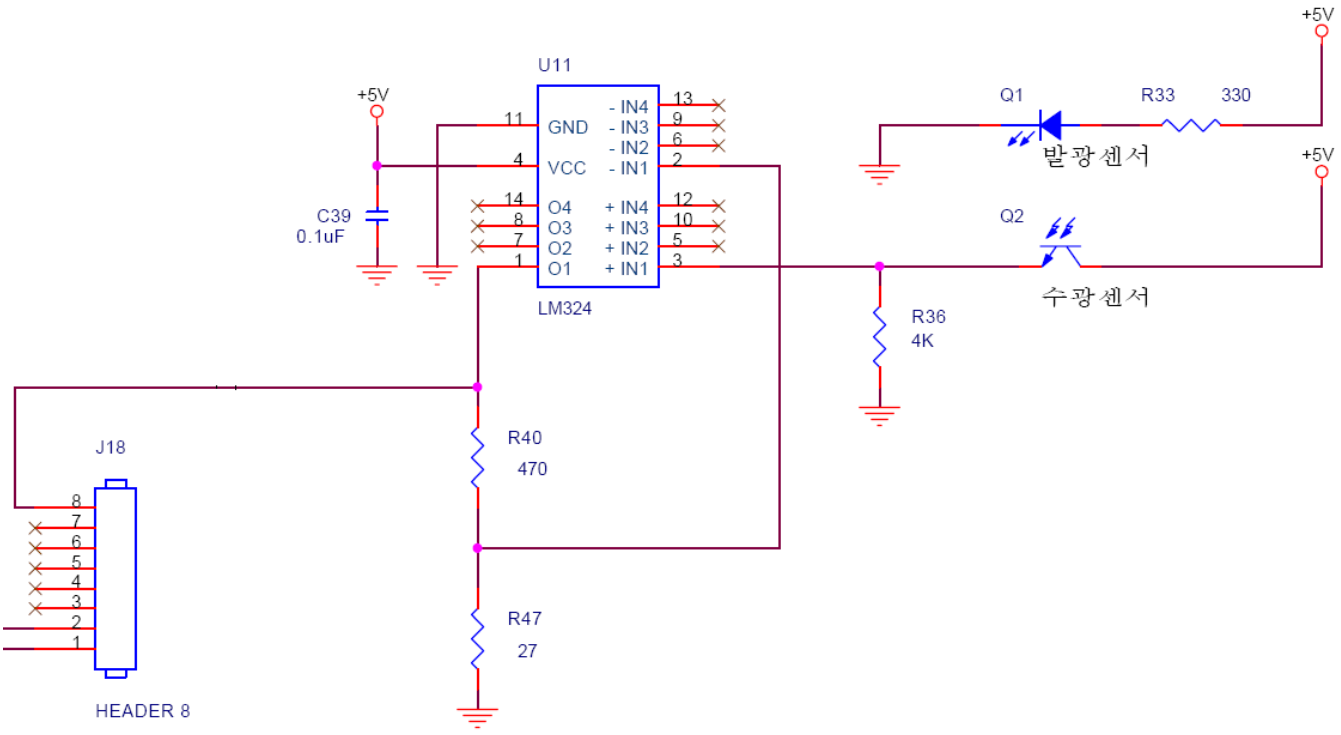
27행: EIMSK 레지스터에서 INT4인터럽트를 활성화 하기 위한 설정이다.



33~44행: 인터럽트가 발생하면 int_flag값이 1이 되기 때문에 int_flag를 체크하여 인터럽트가 발생된 것이 확인되면 int_flag는 다시 0으로 만들어 주고 모든 LED를 켠 후에 1초간 delay를 시킨다. 1초가 지난 후에 다시 모든 LED를 끈다.

5.2.1.1 임의의 KEY를 눌렀다 떼는 순간부터 1초간 모든 LED가 켜졌다 꺼지는 것을 보여라

이 예제에서 사용하는 수광 센서는 DK128-EXT 보드의 J18 헤더 핀에서 8번 핀을 사용하고 있는 것을 참조하기 바란다.



ex3_2.c 수광 센서가 차단되었을 때 모든 LED가 켜지는 예제

```
1   #include <avr/io.h>
2   #include <avr/interrupt.h>
3
4   // PF0 ~ PF7은 61 ~ 54번 핀
5   #define DDR_LED             DDRF
6   #define PORT_LED            PORTF
7   #define PIN_LED             PINF
8
9
10  volatile unsigned char int_flag;
11
12  // INT4의 외부 인터럽트 함수
13  SIGNAL(SIG_INTERRUPT4)
14  {
15          int_flag = 1;
16  }
```

ex3_2.c 수광 센서가 차단되었을 때 모든 LED가 켜지는 예제

```
17    int main(void)
18    {
19            DDR_LED = 0xFF;                 // LED 포트를 출력 모드로 설정
20            PORT_LED = 0xFF;                // LED 포트 초기화
21
22            // INT4을 low level 인터럽트 설정
23            EICRB = (0 << ISC41) | (0 << ISC40);
24            // INT4을 활성화
25            EIMSK= (1 << INT4);
26
27
28            sei();                          // 글로벌 인터럽트 활성화
29
30            for (;;)
31            {
32                    if (int_flag)
33                    {
34                            int_flag = 0;
35                            PORT_LED = 0x00;
36                    }
37                    else
38                    {
39                            PORT_LED = 0xFF;
40                    }
41            }
42
43            return 1;
44    }
45
```

23행: INT4가 low level에서 인터럽트를 발생 시키기 위한 EICRB 레지스터의 설정 값이다.

33~37행: int_flag가 1로 설정된 것이 확인되면 인터럽트가 발생한 것이므로 int_flag를 0으로 클리어 시키고 모든 LED를 켠다.

38~41행: int_flag가 1이 아니라면 인터럽트가 발생하고 있지 않은 상태이므로 모든 LED를 끈다.

5.2.2 실습예제

5.2.2.1 FND의 숫자가 0.1초 주기로 1씩 증가하다가, 수광 센서가 차단되었을 때는 멈추는 것을 보여라.(단, FND의 범위는 00~99까지)