7장. 인터럽트의 동작

한국산업기술대학교
지능형 헬스케어 시스템 연구소

이응혁 교수

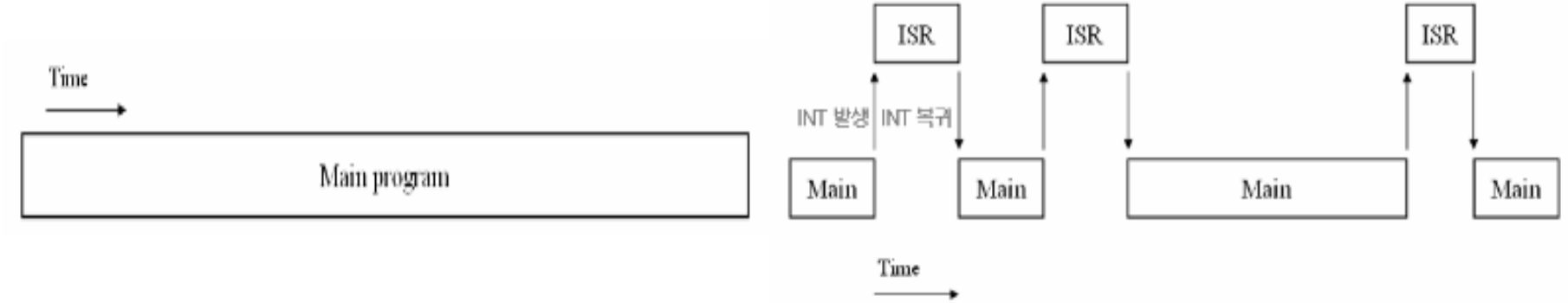
7.1 인터럽트(Interrupt) 개요

인터럽트 개념

- 프로그램이 수행되고 있는 동안에 어떤 조건이 발생하여 수행중인 프로그램을 일시적으로 중지 시키게 만드는 조건이나 사건의 발생
  - 비동기적으로 처리→다른 프로그램이 수행되는 동안 여러 개의 사건을 처리할 수 있는 메커니즘
  - 인터럽트가 발생하면 마이크로 컨트롤러는 현재 수행중인 프로그램을 일시 중단하고, 인터럽트 처리를 위한 프로그램을 수행한 후에 다시 원래의 프로그램으로 복귀
- ISR(Intertupt Service Routine) 또는 Interrupt handler : 인터럽트 처리하는 프로그램

7.1 인터럽트(Interrupt) 개요

주 프로그램과 인터럽트 프로그램의 차이



< 인터럽트가 없는 프로그램 실행 >

< 인터럽트가 있는 프로그램 실행 >

- 인터럽트가 발생할 때 주 프로그램은 일시적으로 정지하고 ISR로 분기하고, ISR이 실행되고 연산이 수행된 후에 ISR 프로그램이 종료되면, 주 프로그램의 중지된 부분부터 다시 수행
- 인터럽트의 종료는 '인터럽트로부터의 복귀(return from interrupt):RETT' 명령에 의해 수행
- Background/Foreground

7.1 인터럽트(Interrupt) 개요

인터럽트 의 종류

인터럽트 발생 원인에 의한 분류

- 하드웨어 인터럽트
  - 내부 인터럽트 : 마이크로 컨트롤러 내부의 기능에 의해 발생
  - 외부 인터럽트 : 마이크로 컨트롤러 외부에 부가된 소자에 의해 발생
- 소프트웨어 인터럽트

7.1 인터럽트(Interrupt) 개요

인터럽트 처리 방식에 의한 분류

- **일반 적인 인터럽트(/INT)**
  - 프로그램에 의하여 인터럽트의 요청을 받아들이지 않고 무시할 수 있는 구조의 인터럽트(maskable interrupt)를 의미
  - 우선처리 메커니즘
  - 우선처리 메커니즘의 경우 보통 인터럽트를 허용하는 방법은 인터럽트 마스크 레지스터 또는 인터럽트 허용 레지스터를 사용하여 각각의 인터럽트를 개별적 허용하고 이것들을 다시 전체적으로 허용함.

- **차단 불가능 인터럽트(/NMI)**
  - 프로그램에 의해 어떤 방법으로도 인터럽트 요청이 차단될 수 없는 인터럽트 (non-maskable interrupt)를 의미
  - 전원 이상이나 비상 정지 스위치등과 같이 시스템에 치명적인 오류를 대비하기 위해 주로 사용

7.1 인터럽트(Interrupt) 개요

인터럽트 의 제어 및 처리 절차

벡터형 인터럽트

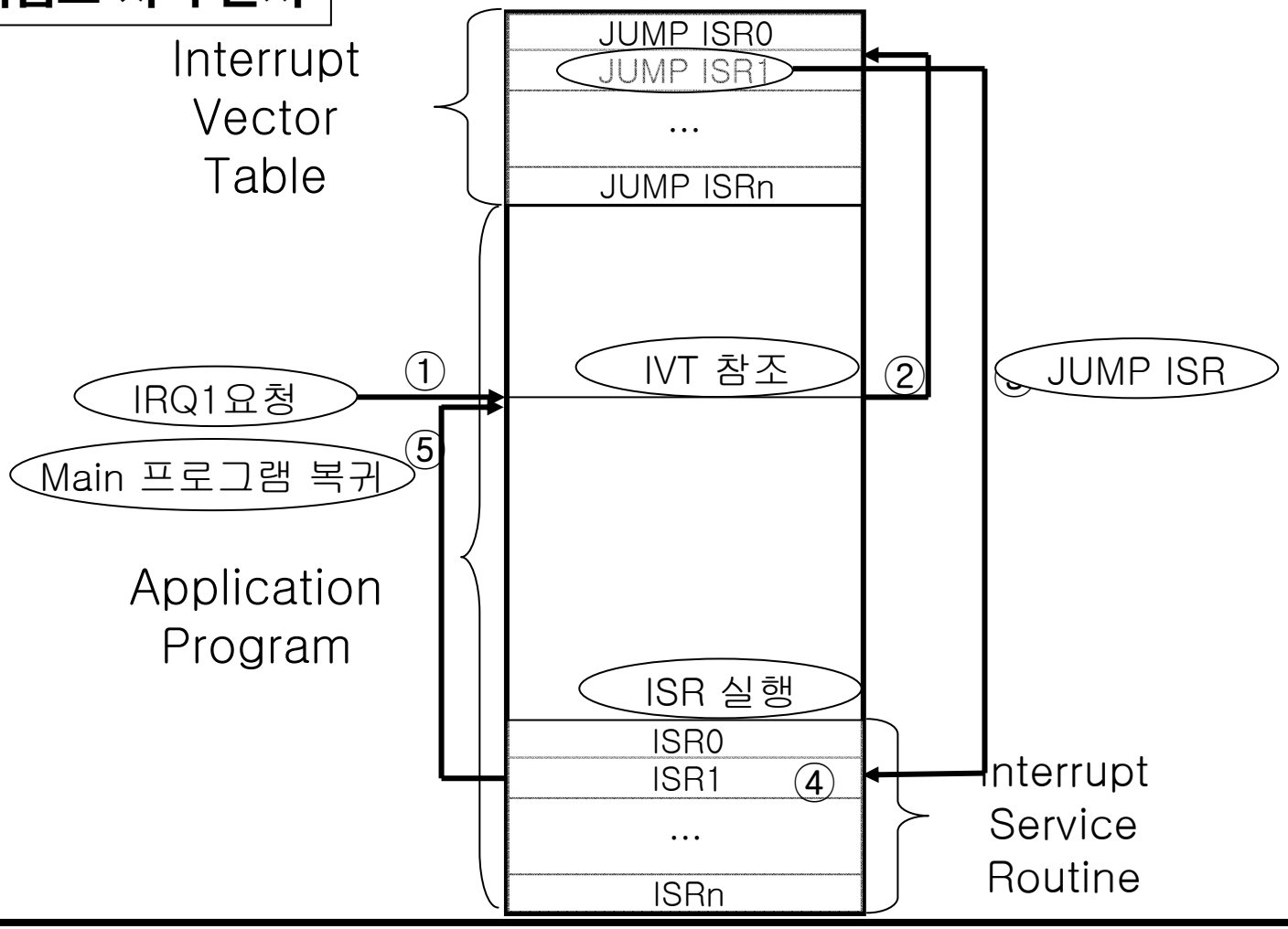
- 인터럽트가 발생할 때마다 인터럽트를 요청한 장치가 인터럽트 벡터를 마이크로 컨트롤러에게 전송하는 방식
- 각 주변장치가 각각의 인터럽트 신호선을 가지고 있고, 각 주변장치가 인터럽트를 요청하면 마이크로컨트롤러는 각각의 인터럽트에 따라 미리 지정된 인터럽트 벡터를 가지고 있어 즉시 해당 인터럽트 서비스 루틴을 찾아가는 방식
  - 인터럽트 처리 응답시간이 빠르고, AVR에 구현된 방식

인터럽트의 우선순위

- 2개 이상의 주변장치가 동시에 마이크로 컨트롤러에 인터럽트를 요구하는 경우 우선 순위를 미리 정하여 번에 하나의 인터럽트를 선택하여 처리
- 우선순위가 높은 인터럽트의 처리 중 -> 낮은 순위의 인터럽트는 대기상태
- 우선순위가 높은 인터럽트의 처리 끝 -> 낮은 순위의 인터럽트는 미리 지정된 우선순위에 의해 처리

7.1 인터럽트(Interrupt) 개요

인터럽트 처리 순서



7.2 ATmega128의 인터럽트 구성

인터럽트의 종류

- 리셋을 포함하여 총 35종의 인터럽트 소스가 존재
  - 외부 인터럽트 8개,
  - 타이머/카운터0에 관련된 인터럽트 2개
  - 타이머/카운터1에 관련된 인터럽트 5개
  - 타이머/카운터2에 관련된 인터럽트 2개
  - 타이머/카운터3에 관련된 인터럽트 5개
  - USART0와 USART1에 관련된 인터럽트 각각 3개와 기타의 인터럽트 6개
- 인터럽트의 동작 형태
  - 인터럽트가 발생하면 관련 플래그 비트를 1로 세트하여 트리거 시키는 형태
    - ✓ 프로그램 카운터가 실제 인터럽트 벡터로 지정되어 인터럽트 처리 루틴을 수행
    - ✓ 해당 플래그는 하드웨어에 의해 자동으로 0으로 클리어됨.
    - ✓ 인터럽트 플래그는 해당 비트에 1을 써 넣음으로써 0으로 클리어할 수 있음.
    - ✓ 인터럽트 마스크 레지스터 또는 SREG 레지스터를 통해 금지 상태로 설정하여 놓았더라도 인터럽트가 발생하면 해당 인터럽트 플래그가 1로 설정되어 인터럽트 대기 상태로 되며, 나중에 인터럽트가 허가상태로 설정될 때 해당 인터럽트가 처리됨.
  - 인터럽트 조건이 발생한 동안에만 인터럽트를 트리거 하는 형태
    - 인터럽트는 인터럽트 발생조건이 사라지면 인터럽트 요청도 없어지므로 나중에 인터럽트가 다시 허용 상태로 되더라도 인터럽트는 요청되지 않음.

7.2 ATmega128의 인터럽트 구성 (1)

인터럽트의 종류 및 인터럽트 벡터

벡터 번호 (우선 순위)	벡터 주소	인터럽트 소스	인터럽트 발생 조건
0	0x0000	RESET	외부 핀, 전원 투입 리셋, 저전압 검출 리셋, 워치독 리셋, JTAG AVR 리셋
1	0x0002	INT0	외부 인터럽트 0
2	0x0004	INT1	외부 인터럽트 1
3	0x0006	INT2	외부 인터럽트 2
4	0x0008	INT3	외부 인터럽트 3
5	0x000A	INT4	외부 인터럽트 4
6	0x000C	INT5	외부 인터럽트 5
7	0x000E	INT6	외부 인터럽트 6
8	0x0010	INT7	외부 인터럽트 7
9	0x0012	TIMER2 COMP	타이머/카운터2 비교 일치
10	0x0014	TIMER2 OVF	타이머/카운터2 오버플로우
11	0x0016	TIMER1 CAPT	타이머/카운터1 입력 캡처
12	0x0018	TIMER1 COMPA	타이머/카운터1 비교 일치 A
13	0x001A	TIMER1 COMPB	타이머/카운터1 비교 일치 B
14	0x001C	TIMER1 OVF	타이머/카운터1 오버플로우
15	0x001E	TIMER0 COMP	타이머/카운터0 비교 일치

7.2 ATmega128의 인터럽트 구성 (2)

인터럽트의 종류 및 인터럽트 벡터

벡터 번호 (우선 순위)	벡터 주소	인터럽트 소스	인터럽트 발생 조건
16	0x0020	TIMER0 OVF	타이머/카운터0 오버플로우
17	0x0022	SPI, STC	SPI 시리얼 통신 완료
18	0x0024	USART0, RX	USART0, 수신 완료
19	0x0026	USART0, UDRE	USART0, 데이터 레지스터 비움
20	0x0028	USART0, TX	USART0, 송신 완료
21	0x002A	ADC	ADC 변환 완료
22	0x002C	EE READY	EEPROM 준비
23	0x002E	ANALOG COMP	아날로그 비교기
24	0x0030	TIMER1 COMPC	타이머/카운터1 비교 일치 C
25	0x0032	TIMER3 CAPT	타이머/카운터3 입력 캡처
26	0x0034	TIMER3 COMPA	타이머/카운터3 비교 일치 A
27	0x0036	TIMER3 COMPB	타이머/카운터3 비교 일치 B
28	0x0038	TIMER3 COMPC	타이머/카운터3 비교 일치 C
29	0x003A	TIMER3 OVF	타이머/카운터3 오버플로우
30	0x003C	USART1, RX	USART1, 수신 완료
31	0x003E	USART1, UDRE	USART1, 데이터 레지스터 비움
32	0x0040	USART1, TX	USART1, 송신 완료
33	0x0042	TWI	I2C 통신 인터페이스
34	0x0044	SPM READY	저장 프로그램 메모리 준비

7.2 ATmega128의 인터럽트 구성

리셋 및 인터럽트 벡터의 배치

BOOTRST	IVSEL	리셋 벡터 주소	인터럽트 벡터의 시작 주소
1	0	0x0000	0x0000
1	1	0x0000	부트 리셋 주소 + 0x0002
0	0	부트 리셋 주소	0x0002
0	1	부트 리셋 주소	부트 리셋 주소 + 0x0002

- **리셋 및 인터럽트 벡터는 BOOTRST와 IVSEL 비트의 조합에 의해 가변적으로 배치**
  - **부트 리셋 주소 : 부트 로더 섹션의 크기의 설정에 따라 달라짐. (표 2.22 참조)**
    예) BOOTSZ1~0 = 00 이면 부트 사이즈는 1024워드로 되어 부트 리셋 주소는 0x1C000이 됨.
- **일반적인 ATmega128에서는 BOOTRST 비트는 1로 설정되고, IVSEL은 0으로 설정**

7.2 ATmega128의 인터럽트 구성

인터럽트 벡터의 배치

- 인터럽트 벡터를 응용 프로그램 섹션과 부트 로더 섹션 사이에서 이동하기 위해서는 MCU 컨트롤 레지스터(MCUCR, MCU Control Register)를 사용
- MCUCR 레지스터의 비트 구성은 아래와 같으며, 여기서 IVSEL과 IVCE 비트가 이상의 목적으로 사용되고, 나머지는 외부 인터럽트를 개별적으로 허가하는 용도로 사용

Bit	7	6	5	4	3	2	1	0	
	SRE	SRW10	SE	SM1	SM0	SM2	IVSEL	IVCE	MCUCR
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	

- IVSEL과 IVCE 비트를 사용
- 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
- IVCE (인터럽트 벡터 변경 허가, Interrupt Vector Change Enable)
  - IVSEL 비트의 변경을 허가하기 위해서 IVCE 비트는 1로 설정되어 있어야 함.

7.2 ATmega128의 인터럽트 구성

외부 인터럽트의 개요

- 8개의 외부 인터럽트 입력 : INT0~7
  - INT1와 INT0은 상승/하강 에지 및 Low 상태에 의해 인터럽트 발생
- INT0~70이 에지 방식에 의한 인터럽트 발생 방법으로 설정되면, I/O클럭에 동기를 맞추어 인터럽트가 발생
- INT0~70이 레벨 변화 방식으로 설정되고, 에지 트리거 방식으로 설정된 INT2와 PCINT15~0 핀의 변화가 일어나는 경우의 PCI 인터럽트들은 비동기적으로 인터럽트가 발생
- 비동기적으로 발생하는 인터럽트들은 휴면 모드를 제외하고 슬립 모드를 해제하는 수단으로 사용 가능
- INT7와 INT4가 에지 방식에 의해 설정되면 이는 I/O 클럭을 필요로 하므로 이것들은 I/O 클럭이 차단되는 휴면 모드 이외의 슬립 모드에서는 슬립 모드를 해제하는 수단으로 사용할 수 없음.
- 레벨 변화 방식으로 사용되는 인터럽트가 전원 차단 모드의 해제 수단으로 사용되는 경우에는 좀 더 긴 인터럽트 신호가 요구됨. -> 슬립 모드를 해제하고 인터럽트가 발생되려면 충분히 긴 시간동안 인터럽트 신호가 L 상태로 입력되어야 함.

7.2 ATmega128의 인터럽트 구성

외부 인터럽트 제어 레지스터

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(INT0~3)
EICRB	외부 인터럽트 트리거 방식 설정 레지스터(INT4~7)

EIMSK 제어 레지스터

- **외부 인터럽트 마스크 레지스터 EIMSK(External Interrupt Mask Regiter)는 인터럽트 INT7~0을 개별적으로 허용하는데 사용, 1로 설정 시 인터럽트 허용, 0으로 설정 시 인터럽트 금지 <SREG레지스터의 글로벌 인터럽트 허용 비트 I가 1로 되어야만 실제 허용가능>**

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
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	

7.2 ATmega128의 인터럽트 구성

EIFR 제어 레지스터

Bit	7	6	5	4	3	2	1	0	
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
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	

- 외부 인터럽트 플래그 레지스터 EIFR(External Interrupt Flag Register)는 INT7~0핀에 인터럽트 신호가 입력되어, 해당 인터럽트가 트리거 되었음을 표시하는 데 사용한다.
- 이 비트들은 인터럽트 처리가 시작되고 마이크로 콘트롤러가 인터럽트 벡터를 인출하여 인터럽트 서비스 루틴으로 점프하게 되면 다시 0으로 클리어 된다.
- 강제로 0을 클리어 하려면, 해당 비트에 1을 라이트 하면 된다.

7.2 ATmega128의 인터럽트 구성

EICRA 제어 레지스터

- 외부 인터럽트 INT3~0 핀으로 입력되는 신호에 대한 인터럽트 트리거 방법을 설정
- 모든 레벨 트리거 인터럽트와 INT3~00이 하강 또는 상승 에지 트리거 방식으로 설정 시 인터럽트가 클럭 신호와 관계없이 비동기적으로 검출, 슬립모드를 해제하는 수단으로 사용 가능.

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	

ISCn1	ISCn0	인터럽트 발생 방식
0	0	INTn 핀의 L상태 입력이 인터럽트를 트리거 한다.
0	1	사용하지 않음(Reserved)
1	0	INTn 핀에 하강 에지의 신호가 입력 시 비동기적으로 트리거
1	1	INTn 핀에 상승 에지의 신호가 입력 시 비동기적으로 트리거

7.2 ATmega128의 인터럽트 구성

EICRB 제어 레지스터

- 외부 인터럽트 INT7~4 핀으로 입력되는 신호에 대한 인터럽트 트리거 방법을 설정
- INT7~40이 에지 트리거 방식으로 설정 시, I/O클럭이 필요하게 되므로 I/O 클럭이 차단 되는 IDLE모드 이외의 슬립모드에서는 슬립모드를 해제하는 수단으로 사용 불가

Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
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	

ISCn1	ISCn0	인터럽트 발생 방식
0	0	INTn 핀의 L상태 입력이 인터럽트를 트리거 한다.
0	1	INTn 핀의 하강 에지 또는 상승 에지가 인터럽트를 트리거한다.
1	0	INTn 핀에 하강 에지의 신호가 인터럽트를 트리거
1	1	INTn 핀에 상승 에지의 신호가 인터럽트를 트리거

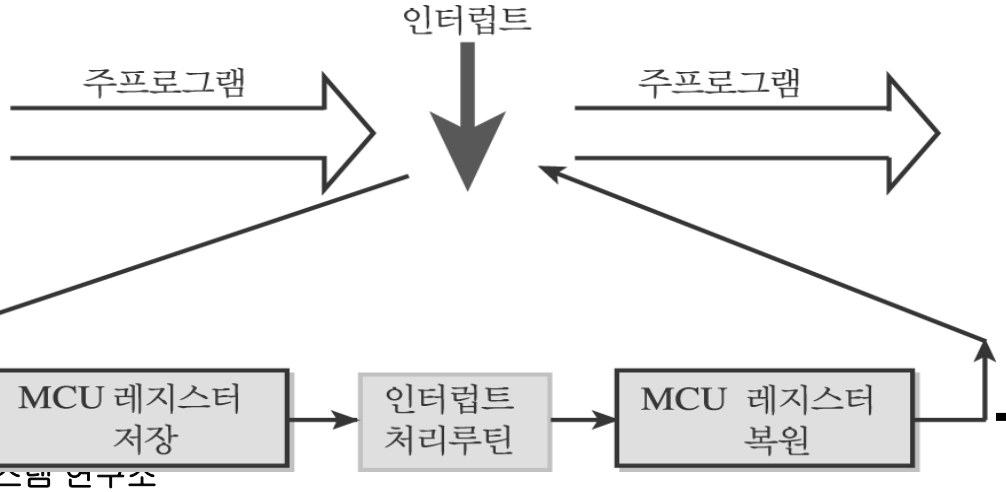
기호	파라메터	Min.	Typ.	Max.	Units
t_{INT}	비동기 외부 인터럽트에 대한 최소의 펄스 폭		50		ns

- **에지 트리거로 사용되는 인터럽트 신호의 최소 신호 폭은 50ns이상 이어야 함.**

7.3 ATmega128의 인터럽트 처리

인터럽트 처리 메카니즘

- 인터럽트가 발생하였을 때 MCU 내부에서의 동작
  - 현재 명령어의 수행을 마침
  - 스택에 PC를 저장
  - 현재 인터럽트 상태를 내부적으로 저장
  - 다른 인터럽트가 받아들여지지는 않는다. 즉, 블록킹 됨
  - ISR의 벡터 주소가 PC에 적재됨
  - ISR이 수행
- ISR은 인터럽트로 복귀(RETI)명령어로 끝나게 된다.
- 이 명령으로 인해 스택으로부터 PC의 이전 값과 인터럽트 상태의 이전 값을 되찾게 되어, 주 프로그램의 수행이 중단 되었던 곳부터 다시 계속 수행함.



7.3 ATmega128의 인터럽트 처리

ATmega 128의 인터럽트 처리 메카니즘

- ATmega128에서의 인터럽트 처리는 정해진 우선순위에 의해 처리
- ATmega128에서는 여러 인터럽트기 동시에 발생하였을 때 우선순위가 높은 인터럽트가 먼저 처리되고, 이 우선순위는 변경이 불가능
- 인터럽트가 발생하면 인터럽트에 해당하는 플래그 비트가 세트, 이 플래그 비트에 의해 인터럽트가 요청, 전체 인터럽트 허가 비트 I와 해당 인터럽트 허가 비트가 모두 1로 설정되어 있으면, 인터럽트가 요청되어 인터럽트 벡터의 주소를 찾아가 인터럽트 서비스 루틴(ISR)을 수행하게됨
- ISR이 수행되고 있을 때, ATmega128은 자동적으로 전체 인터럽트 허가 비트(SREG의 I 비트)를 클리어 모든 인터럽트의 발생을 금지 서비스 루틴의 종료와 함께 인터럽트를 허용
- ATmega128이 RETI 명령에 의하여 ISR의 실행을 마치고 주프로그램으로 복귀하는 데에도 4 클럭 사이클이 소요된다. 이 시간동안에 PC의 값이 스택으로부터 복구됨

7.4 Code Vision을 이용한 ISR의 작성

ISR의 작성

- 인터럽트의 서비스는 벡터 주소라는 교유 번지에서 시작
- 인터럽트 벡터에는 인터럽트 기능을 서비스 하기 위한 프로그램이 위치해 있어야 함
- 인터럽트 서비스 루틴이 호출되기 위해서는 C 언어에서 인터럽트 서비스 루틴이 올바르게 선언되어 있어야 한다.

  인터럽트 서비스 루틴의 선언 : interrupt [n] void int_func_name (void)

- 타이머 0의 오버플로우 인터럽트에 대한 인터럽트 서비스 루틴의 작성 예

```
// Called automatically on TIMER0 overflow
unsigned int interrupt_cnt;
unsigned char second;
interrupt [18] void timer0_overflow(void)
{
        if (++interrupt_cnt == 4000){           // count to 4000
                second++;                       // second counter
                Interrupt_cnt = 0;   // clear int counter
        }
}
```

7.4 Code Vision을 이용한 ISR의 작성

인터럽트의 허가 및 금지 방법

- 인터럽트를 사용하려면 전체 인터럽트 허가 비트(SREG의 I 비트)를 1로 설정 → SREG의 I를 이용하여 전체 인터럽트를 허가함.
  - SREG |= 0x80; // SREG의 7비트를 1로 설정하여 전체 인터럽트를 허가함

- 어셈블리 명령어를 이용하여 인터럽트를 허가하고 금지하는 방법 → sei와 cli의 명령어를 사용
  - #asm("sei"); // 전체 인터럽트 허가
  - #asm("cli"); // 전체 인터럽트 금지

- #define 전처리기를 이용하여 C 언어 함수로 구현하는 방법 → <mega162.h>에 포함하여 사용
  - #define sei() (#asm("sei"))
  - #define cli() (#asm("cli"))

7.5 인터럽트를 이용한 실험

ISR의 초기화 과정

- EIMSK 레지스터의 비트 설정을 통한 사용하고자 하는 인터럽트의 허가
- EICRA 레지스터의 비트 설정을 통한 인터럽트 트리거 방식 설정
- SREG의 I 비트의 설정을 통한 전체 인터럽트를 허가

- 예) 외부 INT0 핀의 신호가 하강 에지에 의해 인터럽트를 발생하도록 초기화하는 과정

```
void Interrupt_init(void)
{
   EIMSK = 0x01;    // INT0 비트 설정(외부 인터럽트0 허가)
   EICRA = 0x02;    // ISC01 =1, ISC00=1(외부인터럽트 0 하강에지 비동기 트리거)
   sei();           // 전체 인터럽트 허가
}
```

7.5 인터럽트를 이용한 실험

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	

- EICRA 레지스터는 그림과 같은 비트 구성을 가지고 있으므로 각 비트 명에 대해 다음과 같이 선언

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

- ISC01 비트만 1로 설정하기 위한 과정 → 왼쪽 시프트연산자 "<<" 를 사용
  1<<ISC01
  - ISC01은 1이므로 1<<1 // 0b00000001을 1비트 시프트 → 0b00000010
  - 최종적으로 MCUCR의 ISC01 비트가 세트됨

7.5 인터럽트를 이용한 실험

- EICRA 레지스터의 ISC01과 ISC00을 세트하는 프로그램

```
EICRA = 1<<ISC01 | 1<<ISC00;
```

- 이상의 비트 제어를 통해 인터럽트 초기화 함수 Interrupt_init()의 프로그램을 재작성

```
void Interrupt_init(void)
{
        EIMSK = 0x01;
        EICRA = 1<<ISC01 | 1<<ISC00;
        sei();
}
```

7.5 인터럽트를 이용한 실험

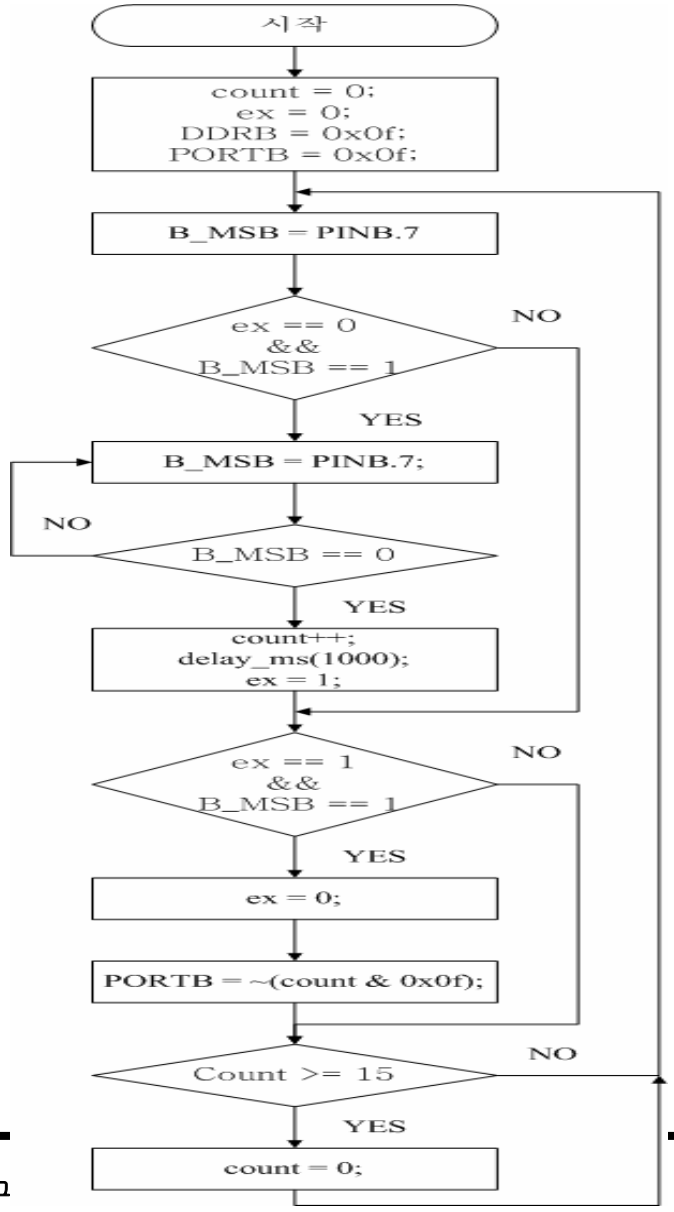
예제 7-1: 폴링 방식의 프로그램과 인터럽트 프로그램의 차이점

- **PORTB의 스위치의 신호를 계수하여 PORTB의 LED에 카운트된 값을 출력하는 프로그램 작성**

```
#include <mega128.h>
#include <delay.h>

Byte count, change;
bit key7;

Byte Exch(void)
{
      while(1)
      {
          key7 = PINB.7;
                if(key7 == 0)
                 {
                    count++;
                    delay_ms(1000);
                    return 1;
                 }
      }
}
```

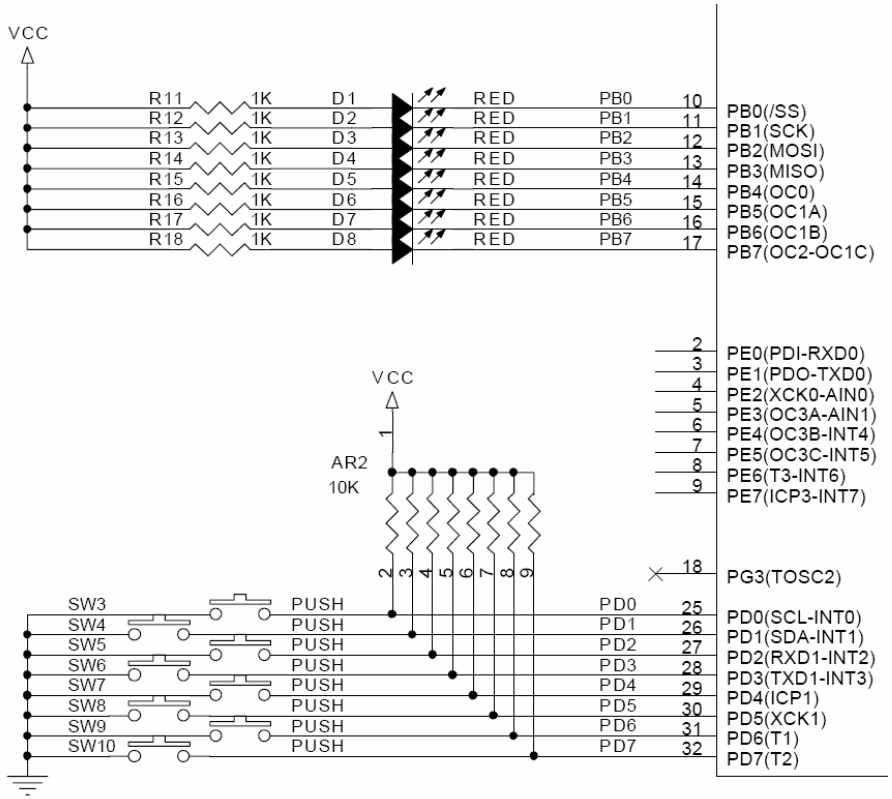
```
void main(void)
{
          // count 변수의 선언 및 초기화
          Count = 0;
          ex = 0;
          // 포트 B의 상위 니블을 입력, 하위 니블을 출력으로 설정
          // 포트 B의 LED를 모두 OFF
          DDRB = 0x0f;
          PORTB = 0x0f;

   while(1)
   {
            key7 = PINB.7;

                if(change == 0 && key7 == 1) {
                          change = Exch();
                         }
                if(change == 1 && key7 == 1) {
                          change = 0;
                         }

                   PORTB = ~(count&0x0f);
                   If(count>= 15) count = 0;
   }
}
```

7.5 인터럽트를 이용한 실험



7.5 인터럽트를 이용한 실험

예제 7-2: 외부 인터럽트 0 서비스 루틴의 작성



-외부 인터럽트 실험 회로-

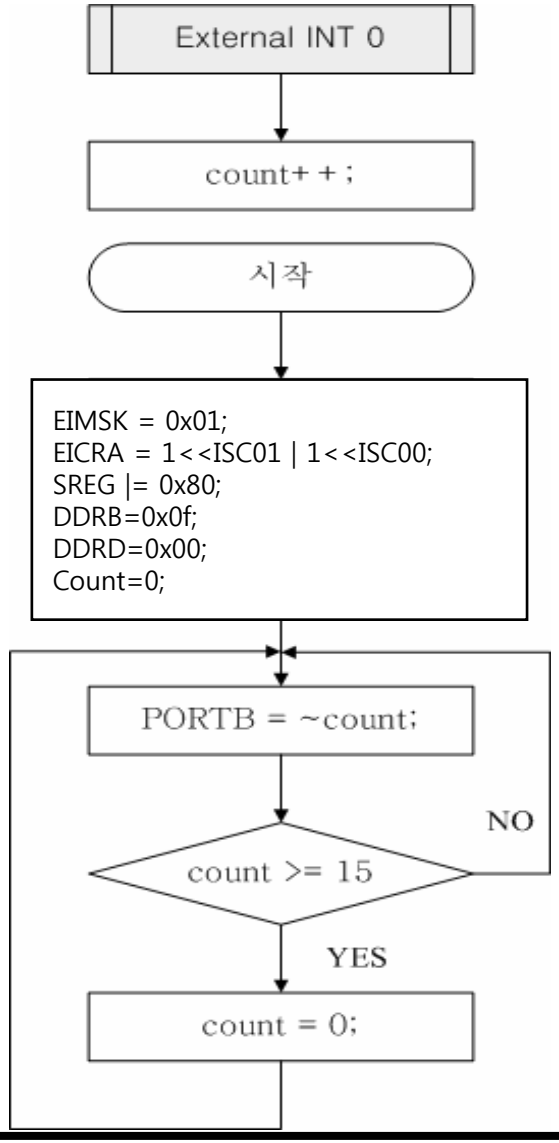
7.5 인터럽트를 이용한 실험

- PORTB에 연결된 스위치 대신에 INT0 핀에 연결된 스위치의 입력을 계수하여 PORTB의 LED에 출력하는 프로그램 작성

```
#include <mega128.h>
#include <delay.h>

Byte count;

interrupt [EXT_INT0] void ext_int0(void)
{
        count++;
}

void Interrupt_init(void)
{

        EIMSK = 0x01;
        EICRA = 1<<ISC01 | 1<<ISC00;
        sei();

}
```

```
void main(void)
{
    DDRB = 0x0f;
    DDRD = 0x00;

    count = 0;

    Interrupt_init();

  while(1)
  {
    PORTB = ~count;
    if(count >= 15) count = 0;
  }
}
```

7.5 인터럽트를 이용한 실험



7.5 인터럽트를 이용한 실험

예제 7-3: 외부 인터럽트 0 서비스 루틴의 작성(인터럽트 모드의 변경)

- **회로에서 INT0 핀에 Low 신호가 입력되면 PORTB 포트에 연결되어 있는 모든 LED를 켜고, 인터럽트가 해제되면 LED는 off 상태를 그대로 유지하는 프로그램 작성**

```
#include <mega128.h>
#include <delay.h>

bit exchange;

interrupt [EXT_INT0] void ext_int0(void)
{
        exchange = ~exchange;
}

void Interrupt_init(void)
{

            EIMSK = 0x01;
            EICRA = 1<<ISC01 | 1<<ISC00;
            sei();

}
```

```
void main(void)
{
    DDRB = 0x0f;
    DDRD = 0x00;

        Interrupt_init();

        PORTB = 0x0f;

        exchange = 0;

  while(1)
  {
     if(exchange) PORTB = 0x00;
     else PORTB = 0x0f;
   }
}
```

7.5 인터럽트를 이용한 실험

예제 7-4: 외부 인터럽트 0의 활용

- **LED가 처음에는 시프트 동작을 수행하고 있다. 이 상황에서외부 INT0 키가 눌릴 때마다 반대의 순서로 LED 점등하도록 하는 프로그램 작성**

```
#include <mega128.h>
#include <delay.h>

bit Direction;

interrupt [EXT_INT0] void ext_int0(void)
{
        Direction = ~Direction;
}
void PB_LShift(void)
{
   int i;
   Byte Temp;
   Temp = 0xfe;
         for(i=0; i<4;i++)
         {
              delay_ms(500);
              PORTB = Temp;
              Temp = (Temp<<1)|0x01;
         }
}
```

```
void PB_RShift(void)
{
   int i;
   Byte Temp;
   Temp = 0xef;
         for(i = 0; i < 4; i++)
         {
              delay_ms(500);
              Temp = (Temp >> 1);
              PORTB = Temp;
         }
}

void Interrupt_init(void)
{
   EIMSK = 0x01;
   EICRA = 1<<ISC01 | 1<<ISC00;
   sei();

}
```

```
void main(void)
{
    DDRB = 0x0F;
    DDRD = 0x00;

    Interrupt_init();

    PORTB = 0x0f;

    Direction = 0;

  while(1)
  {
      if(Direction) PB_RShift();
      else PB_LShift();
  }
}
```