Interrupt & Polling

HCS12의 인터럽트

- CCR (Condition Code Register)
  - X : /XIRQ mask bit
    - X = 0 : /XIRQ 인터럽트 Enable
    - X = 1 : /XIRQ 인터럽트 Disable
  - I : 인터럽트 mask bit
    - I = 0 : 마스크가능 인터럽트 Enable
    - I = 1 : 마스크가능 인터럽트 Disable

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	S	X	H	I	N	Z	V	C
Reset:	1	1	0	1	0	0	0	0

  - CLI 명령어 : I = 1로 설정하는 어셈블러
  - SEI 명령어 : I = 0 으로 설정하는 어셈블러

HCS12의 인터럽트

- 마스크 불가능 인터럽트
  - /XIRQ (Non-maskable interrupt request) PE0
    - 하드웨어 인터럽트
    - 항상 pull-up
    - CCR 레지스터의 X = 0 일 때 활성화
    - 처리 여부를 묻지 않고 CPU가 무조건 처리하는 인터럽트
    - 시스템 리셋에 의해 비활성화
  - 소프트웨어 인터럽트
    - Software Interrupt (SWI) 명령에 의해 수행
    - 실행되면 벡터 테이블에서 SWI의 해당 주소로 이동
    - CCR 레지스터의 I 비트 값에 상관없이 마스크 불가능

HCS12의 인터럽트

- 마스크 가능 인터럽트
  - /IRQ (Maskable Interrupt Request)
    - 하드웨어 인터럽트
    - 항상 pull-up
    - 처리 여부 설정 가능한 인터럽트
    - CCR 레지스터의 I = 0 일 때 활성화
  - 인터럽트 제어 레지스터 (INTCR)

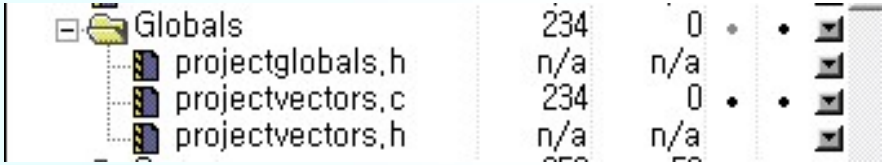
IRQE	IRQEN	0	0	0	0	0	0

    - IRQE
      - 0 : low 레벨에서 /IRQ 핀 응답
      - 1 : 하강 엣지에서 /IRQ 핀 응답
    - IRQEN
      - 0 : /IRQ 핀 Enable
      - 1 : /IRQ 핀 Disable

인터럽트 핸들러 추가

- 인터럽트 핸들러 추가
  - projectvectors.c
    - 인터럽트 핸들러 정의
    - DP512 프로세서의 모든 인터럽트 핸들러가 정의
    - 초기 상태는 software_trap() 함수
    - software_trap() 함수를 사용자가 작성한 핸들러로 변경
  - projectvectors.h
    - 인터럽트 핸들러 선언
    - 사용자가 작성한 인터럽트 핸들러가 선언된 헤더 파일 추가

인터럽트 핸들러 추가



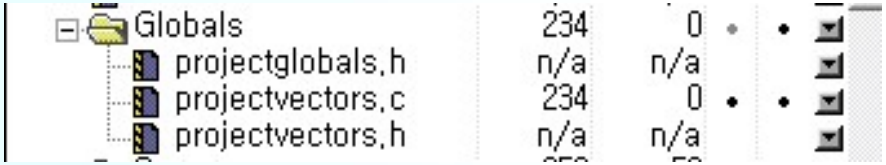
projectvectors.c 파일 열기

```
//**************************************************************************
// PORTJ ISR
// DESCRIPTION:
// Interrupt asserted on PORTJ pin edge, saved as a flag.
// All port pins serviced by one vector.
//
#pragma CODE_SEG NON_BANKED
interrupt void portj_isr(void){ (void) software_trap(); }
#pragma CODE_SEG DEFAULT

//**************************************************************************
```

software_trap() 함수를 인터럽트 핸들러 함수로 변경

```
//**************************************************************************
// PORTJ ISR
// DESCRIPTION:
// Interrupt asserted on PORTJ pin edge, saved as a flag.
// All port pins serviced by one vector.
//
#pragma CODE_SEG NON_BANKED
interrupt void portj_isr(void){ (void) interruptJ_function(); }
#pragma CODE_SEG DEFAULT

//**************************************************************************
```

인터럽트 핸들러 추가



projectvectors.h 파일 열기

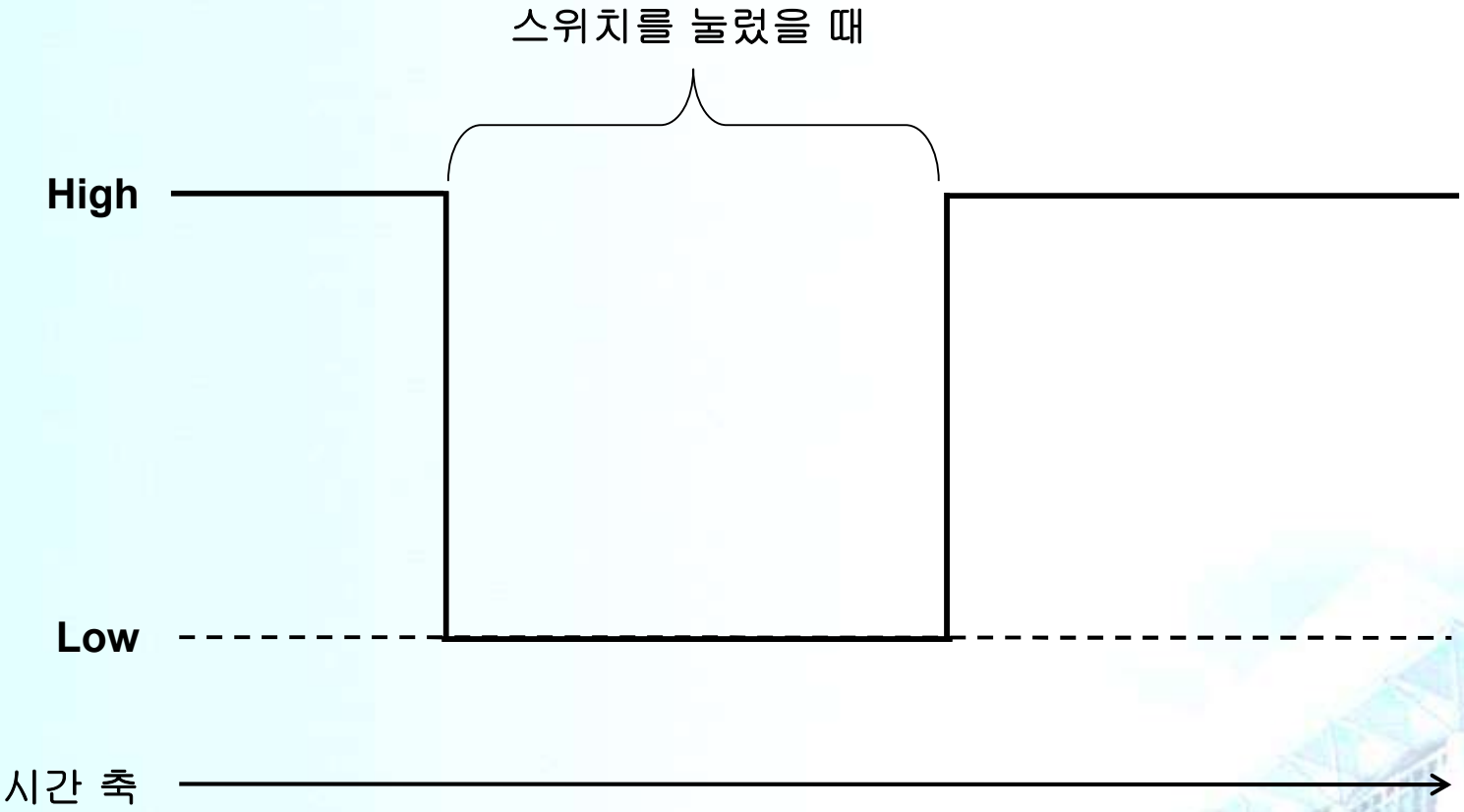
```
/*Include Files*/
#include "projectglobals.h"

/*************************/
/* 사용자가 만든 핸들러를 포함시킴 */


/*Local Prototypes*/
void software_trap(void);
```

사용자가 만든 핸들러 헤더 파일 추가

```
/*Include Files*/
#include "projectglobals.h"
#include "interrupt.h"
```

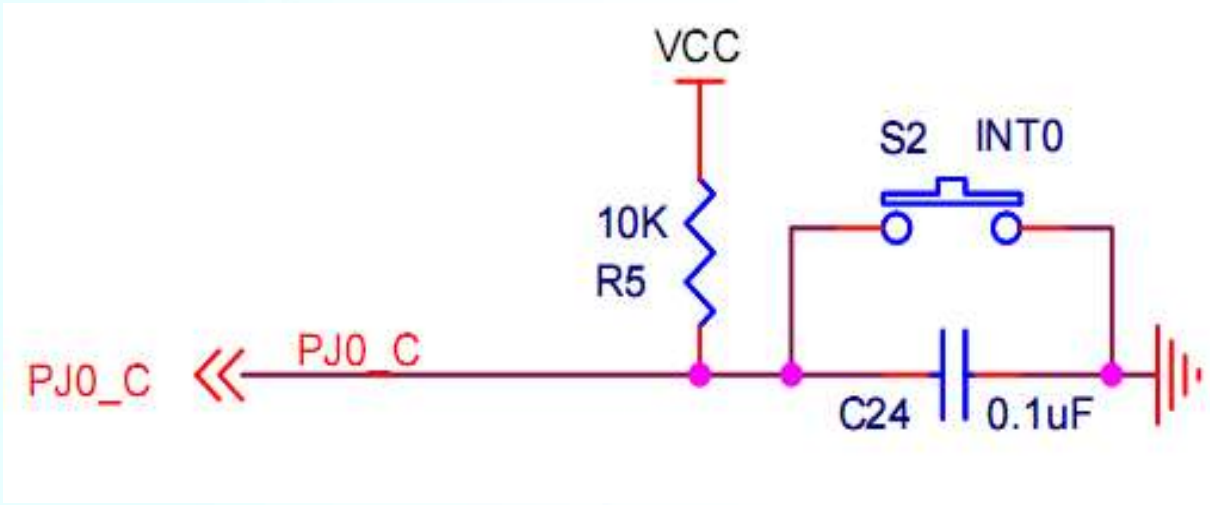
인터럽트 제어

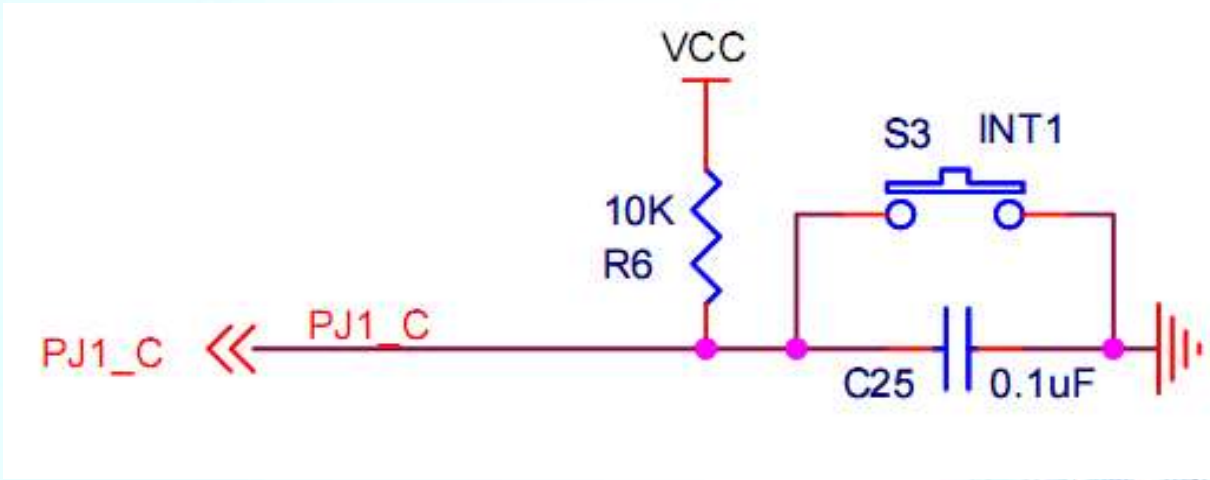
- 스위치와 인터럽트 신호



인터럽트 제어

- 스위치 회로도





인터럽트 제어

- Port J 인터럽트 활성화 레지스터 (PIEJ)
  - PIEJn = 0 : Disable
  - PIEJn = 1 : Enable

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PIEJ7	PIEJ6	0	0	0	0	PIEJ1	PIEJ0
Write:								
Reset:	0	0	-	-	-	-	0	0

- Port J 극 선택 레지스터 (PPSJ)
  - PPSJn = 0 : 하강 엣지 검출
  - PPSJn = 1 : 상승 엣지 검출

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PPSJ7	PPSJ6	0	0	0	0	PPSJ1	PPSJ0
Write:								
Reset:	0	0	-	-	-	-	0	0

인터럽트 제어

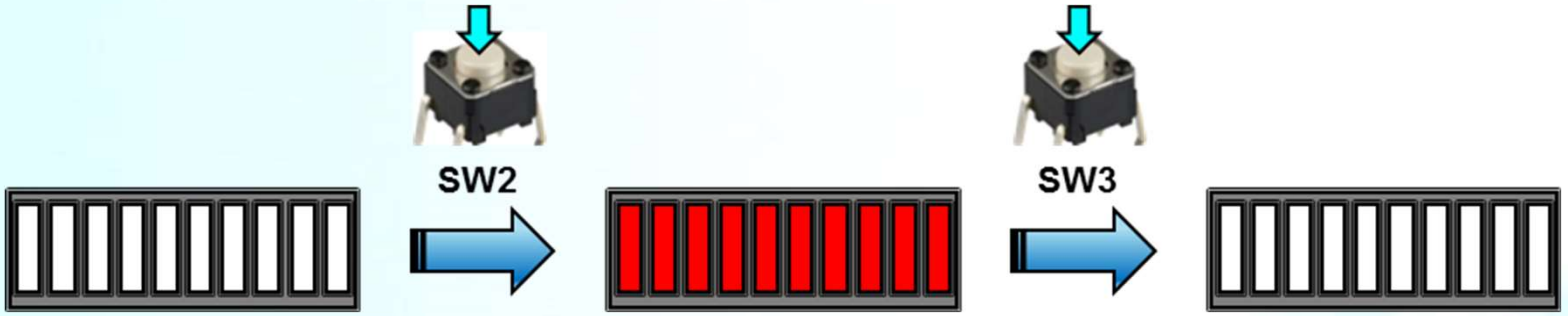
- Port J 인터럽트 플래그 레지스터 (PIFJ)
  - 인터럽트 발생 시 PIFJn = 1로 자동 설정
    - PIFJn = 0 : flag down
    - PIFJn = 1 : flag up

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PIFJ7	PIFJ6	0	0	0	0	PIFJ1	PIFJ0
Write:								
Reset:	0	0	-	-	-	-	0	0

  - 해당 비트에 1을 써야 클리어
  - 예) Port J 의 0번 비트를 클리어할 때
    - PIFJ |= 0x01

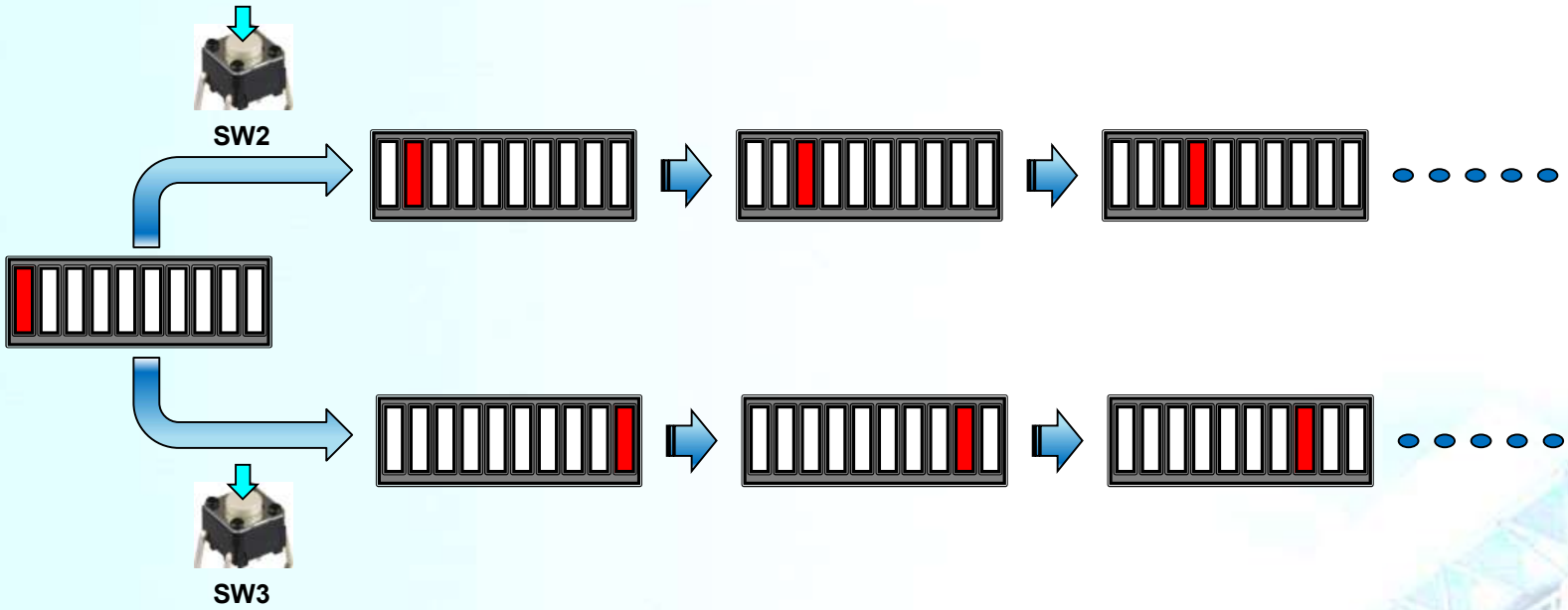
인터럽트 실습

- 실습1
  - SW2를 누르면 Bar LED 전체 점등
  - SW3을 누르면 Bar LED 전체 소등



실시간 인터럽트

- 실습2
  - SW2를 누르면 오른쪽 방향으로 Bar LED 불빛 이동
  - SW3을 누르면 왼쪽 방향으로 Bar LED 불빛 이동
  - LED가 마지막 위치 도달 후 SW입력된다면 마지막 위치에 계속 머뭄



Real-Time Interrupt (RTI) 모듈

RTI의 개념

- Real-Time Interrupt 란?
  - 주기적인 하드웨어 interrupt
  - 일정한 주기마다 동작 실행
    - 내부 클럭이 특정 값에 도달할 때마다 발생
    - 일1초 간격을 생성하여 디지털 시계 구성
    - 정 시간마다 보드의 전원을 check

RTI 모듈 레지스터

- RTICTL 레지스터
  - 인터럽트 주기 설정
  - RTR[3:0] : 배수
  - RTR[6:4] : 지수

RTR[3:0]	RTR[6:4] =							
	000 (OFF)	001 (2^{10})	010 (2^{11})	011 (2^{12})	100 (2^{13})	101 (2^{14})	110 (2^{15})	111 (2^{16})
0000 (÷1)	OFF*	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}
0001 (÷2)	OFF*	$2x2^{10}$	$2x2^{11}$	$2x2^{12}$	$2x2^{13}$	$2x2^{14}$	$2x2^{15}$	$2x2^{16}$
0010 (÷3)	OFF*	$3x2^{10}$	$3x2^{11}$	$3x2^{12}$	$3x2^{13}$	$3x2^{14}$	$3x2^{15}$	$3x2^{16}$
0011 (÷4)	OFF*	$4x2^{10}$	$4x2^{11}$	$4x2^{12}$	$4x2^{13}$	$4x2^{14}$	$4x2^{15}$	$4x2^{16}$
0100 (÷5)	OFF*	$5x2^{10}$	$5x2^{11}$	$5x2^{12}$	$5x2^{13}$	$5x2^{14}$	$5x2^{15}$	$5x2^{16}$
0101 (÷6)	OFF*	$6x2^{10}$	$6x2^{11}$	$6x2^{12}$	$6x2^{13}$	$6x2^{14}$	$6x2^{15}$	$6x2^{16}$
0110 (÷7)	OFF*	$7x2^{10}$	$7x2^{11}$	$7x2^{12}$	$7x2^{13}$	$7x2^{14}$	$7x2^{15}$	$7x2^{16}$
0111 (÷8)	OFF*	$8x2^{10}$	$8x2^{11}$	$8x2^{12}$	$8x2^{13}$	$8x2^{14}$	$8x2^{15}$	$8x2^{16}$
1000 (÷9)	OFF*	$9x2^{10}$	$9x2^{11}$	$9x2^{12}$	$9x2^{13}$	$9x2^{14}$	$9x2^{15}$	$9x2^{16}$
1001 (÷10)	OFF*	$10x2^{10}$	$10x2^{11}$	$10x2^{12}$	$10x2^{13}$	$10x2^{14}$	$10x2^{15}$	$10x2^{16}$
1010 (÷11)	OFF*	$11x2^{10}$	$11x2^{11}$	$11x2^{12}$	$11x2^{13}$	$11x2^{14}$	$11x2^{15}$	$11x2^{16}$
1011 (÷12)	OFF*	$12x2^{10}$	$12x2^{11}$	$12x2^{12}$	$12x2^{13}$	$12x2^{14}$	$12x2^{15}$	$12x2^{16}$
1100 (÷ 13)	OFF*	$13x2^{10}$	$13x2^{11}$	$13x2^{12}$	$13x2^{13}$	$13x2^{14}$	$13x2^{15}$	$13x2^{16}$
1101 (÷14)	OFF*	$14x2^{10}$	$14x2^{11}$	$14x2^{12}$	$14x2^{13}$	$14x2^{14}$	$14x2^{15}$	$14x2^{16}$
1110 (÷15)	OFF*	$15x2^{10}$	$15x2^{11}$	$15x2^{12}$	$15x2^{13}$	$15x2^{14}$	$15x2^{15}$	$15x2^{16}$
1111 (÷ 16)	OFF*	$16x2^{10}$	$16x2^{11}$	$16x2^{12}$	$16x2^{13}$	$16x2^{14}$	$16x2^{15}$	$16x2^{16}$

RTI 모듈 레지스터

- RTI 제어 레지스터 (RTICTL)

	Bit 7	6	5	4	3	2	1	0
Read:	0	RTR6	RTR5	RTR4	RTR3	RTR2	RTR1	RTR0
Write:								
Reset:	0	0	0	0	0	0	0	0

  - Timeout 주기 공식

$$T_{RTI} = \frac{1}{\left\{\frac{\text{OSC clock}}{\text{RTR}[3:0]\times\text{RTR}[6:4]}\right\}}\,[sec]$$

  - Timeout 주파수 공식

$$f_{RTI} = \frac{\text{OSC clock}}{\text{RTR}[3:0]\times\text{RTR}[6:4]}\,[\text{Hz}]$$

RTI 모듈 레지스터

- RTI 제어 레지스터
  - 16MHz 크리스탈 클럭 사용시 1ms 주기 계산 방법
    - $T_{RTI} = \frac{1}{\left\{\frac{\text{OSC clock}}{\text{RTR[3:0]}\times\text{RTR[6:4]}}\right\}} = 1ms,$
    - $\frac{1}{\left\{\frac{16\text{MHz}}{x}\right\}} = 1ms,$
    - $\frac{1}{16\times10^6}x = 10^{-3},$
    - $x = 16 \times 10^3$ $(10^3 \approx 2^{10}$ 이므로)
    - $\therefore x = 16 \times 2^{10}$
    - RTR[3:0] = 16
    - RTR[6:4] = 2^{10}

RTI 모듈 레지스터

- CRG 인터럽트 활성화 레지스터 (CRGINT)
  - Clock and Reset Generator module (CRG) interrupt 요청 활성화
  - RTIE = 1 : 실시간 인터럽트 활성화

	Bit 7	6	5	4	3	2	1	0
Read:	RTIE	0	0	LOCKIE	0	0	SCMIE	0
Write:								
Reset:	0	0	0	0	0	0	0	0

- CRG 플래그 레지스터 (CRGFLG)
  - RTI 주기마다 자동으로 RTIF=1 설정
    - RTIF = 0: RTI 미발생
    - RTIF = 1: RTI 발생
  - RTIF에 1을 쓰면 클리어

	Bit 7	6	5	4	3	2	1	0
Read:	RTIF	PORF	0	LOCKIF	LOCK	TRACK	SCMIF	SCM
Write:								
Reset:	0	(1)	0	0	0	0	0	0

실시간 인터럽트

- 실습3
  - SW2를 누르면 시계 방향으로 세그먼트 하나씩 반복하여 켜기
  - SW3을 누르면 반 시계 방향으로 세그먼트 하나씩 반복하여 켜기
  - 스위치를 한 번만 누르면 해당 방향에 따라 세그먼트 계속 변함
  - SW2나 SW3을 누를 때에는 위의 동작이 멈추지 않고 SW1을 눌러야 멈춤