제 5 장. 인터럽트

Robotics & Artificial Intelligent Control Laboratory
로보틱스 및 인공지능제어 연구실
http://raic.kunsan.ac.kr

Index

인터럽트

-. 외부 또는 내부로부터의 긴급 서비스 요청에 의해서 CPU가 현재 실행중인 일을 중단하고 그 요청에 합당한 서비스를 해주는 기법

-. 종류
- 인터럽트 제어기에 의한 소프트? 어 인터럽트
- PTS에 의한 하드? 어 인터럽트

-. By 소프트? 어 인터럽트…
- 인터럽트 소스 : 28개, 인터럽트 벡터를 중심으로 18개로 분류

-. EXINT(P2.2) – 외부 긴급 서비스 요청선이 연결되어 있다.

-. CPU의 모든 명령의 마지막 싸이클에서 인터럽트의 요구 여부를 체크, 이 요청의 유무 판단

-. 현재 실행중인 명령만은 마친 후 일정의 순서에 따라 그 요청에 합당한 서비스 제공(인터럽트 서비스 루틴)

-. 이후 중단 지점에서부터 다시 실행

-. 용어 정리

➢ 마스크 : 인터럽트를 막기 위한 내부 프로그램 제어 과정 – 누산기에 있는 내용의 일부를 특정 기억 장소나 레지스터에 있는 자료로 대치

➢ 마스크 가능 인터럽트(Maskable Interrupt)

: 하드? 어 인터럽트의 일종

: 프로그램이 u-p의 주의를 100% 필요로 하는 동안 일시적으로 기능이 저지(mask)될 수 있는 것.

➢ 마스크 불가능 인터럽트(Non-Maskable Interrupt, NMI)

: 하드? 어 인터럽트의 일종

: 인터럽트 마스크(interrupt mask)에 영향을 받지 않는 가장 우선 순위가 높은 인터럽트

: S/W나 글쇠판 등 하드? 어 장치에 의해서 생성되는 인터럽트 요구를 우회하고 그것들보다 우선순위를 갖기 때문에 마스크 불가능 인터럽트라고 부른다.

: NMI는 다른 어떤 서비스 요구에 의해서도 저지(mask)되지 않는다.

: 심각한 기억 장치 오류나 정전 사태와 같은 급박한 상황에서만 마스크 불가능 인터럽트 요구가 m-p에 하달된다.

➢ 마스크 레지스터(mask register)

: 끼어들기의 허용 또는 금지를 제어하는 레지스터.

: 각 비트는 마스크 비트라 불리면 개개의 끼어들기 요인에 대응

: 끼어들기 요인 발생 시에 대응하는 마스크 비트가 '0'이면 끼어들기 발생,

: '1'이면 끼어들기 금지

: 마스크 레지스터의 설정과 복귀는 명령이 프로그램적(S/W)으로 행한다.

-. 인터럽트의 공통 기능

: 인터럽트는 제어를 모든 서비스 루틴의 주소를 포함하고 있는 인터럽트 벡터를 통해 서비스 루틴으로 전송한다.

: 인터럽트의 구조는 인터럽트에 의해 중단된 명령의 주소를 저장해야 한다.

: 하나의 인터럽트가 처리되고 있는 동안 다른 인터럽트가 들어와도 처리되지 않는다.
즉, 새로 들어오는 인터럽트는 대기상태에 있다.

: 트랩(trap)은 오류(0으로 나누기 또는 부적절한 기억장치 접근 등) 또는 사용자 요청에 의해 발생하는 S/W 방식의 인터럽트이다.

인터럽트 소스와 벡터

-. 인터럽트의 3요소

- 인터럽트 소스(interrupt source) : 누가 인터럽트를 요구할 수 있는가?
- 인터럽트 벡터(interrupt vector) : 인터럽트 발생시 어디로 뛰는가?
- 인터럽트 우선순위(interrupt priority) : 두 개 이상의 인터럽트가 동시에 발생할 경우 어느것부터 허용하는가?

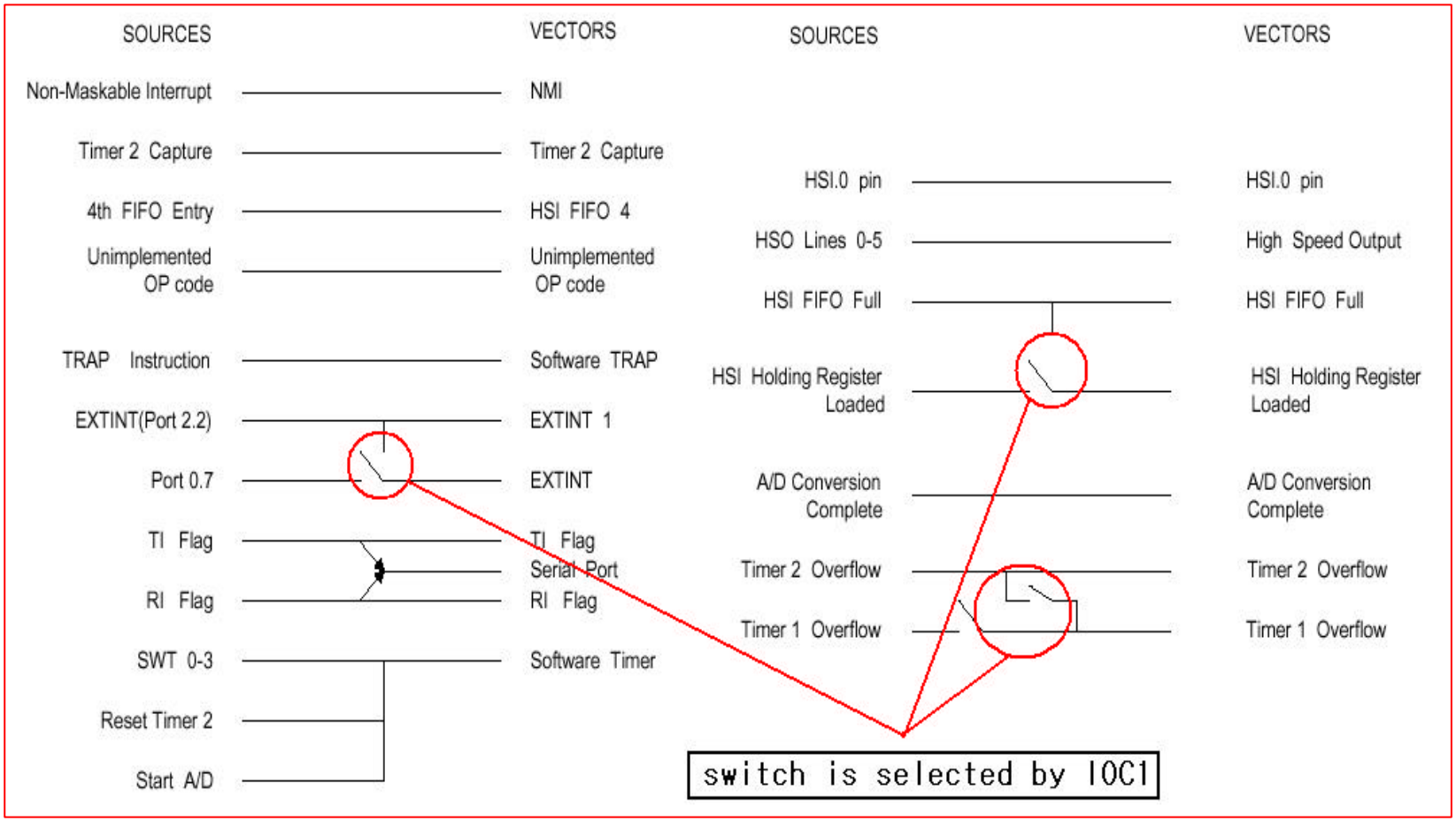
-. 인터럽트를 요구할 수 있는 장치는 28개

-. 벡터를 중심으로 INT01에서 INT15까지 16개와

-. 소프트? 어 TRAP, 정의되지 않는 명령코드 등 18개로 분류

-. Page 176, 그림 5.2 참조…

Interrupt Sources



인터럽트 번호	인터럽트 소스	인터럽트 벡터	우선순위	개별 가능비트	소스 선택비트
INT 15	NMI	203EH	15	—	—
INT 14	HSI FIFO Fall	203CH	14	IMASK1.6	—
INT 13	EXTINT(P2.2)	203AH	13	IMASK1.5	—
INT 12	TIMER2 Overflow	2038H	12	IMASK1.4	IOC1.3=1
INT 11	TIMER2 Capture	2036H	11	IMASK1.3	—
INT 10	HSI FIFO 4	2034H	10	IMASK1.2	—
INT 09	RI	2032H	9	IMASK1.1	—
INT 08	TI	2030H	8	IMASK1.0	—
특수	정의되지 않은 명령 코드	2012H	N/A	—	—
특수	Trap 명령	2010H	N/A	—	—
INT 07	EXTINT(P2.2 or P0.7)	200EH	7	INT_MASK.7	IOC1.1
INT 06	직렬포트(RI, TI)	200CH	6	INT_MASK.6	—
INT 05	소프트? 어 타이머	200AH	5	INT_MASK.5	—
INT 04	HSI.0 핀	2008H	4	INT_MASK.4	IOC0.0=0
INT 03	HSO(HSO핀 0~5)	2006H	3	INT_MASK.3	—
INT 02	HSI Data Available	2004H	2	INT_MASK.2	IOC1.7
INT 01	A/D 변환완료	2002H	1	INT_MASK.1	—
INT 00	타이머 오버플로(T1, T2)	2000H	0	INT_MASK.0	IOC1.2=0(T1) IOC1.3=1(T2)

Ex) Cstartup.asm

```
cstartup module main
;
;==================================
;interrupt vector table
;_interrupt_vector:

;       cseg  at 7f00h
;       extrn  timer_overflow
;       ljmp timer_overflow

       cseg   at  0ff10h
       extrn   ad_conversion_complete
       ljmp  ad_conversion_complete

;       cseg   at  7f20h
;       extrn   hsi_data_available
;       ljmp  hsi_data_available
```

? ? ? ? (Requirements):

1. ? ? ? ? ? ? . -> ex.) “**INT 07” EXTINT(P2.2)**
2. ? **(200EH)**.
3. ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? cstart.a96? ? ?

SOURCE	Vector Location		Priority	Vector contents
	(High Byte)	(Low Byte)		
unimplmented	2013h	2012h	Not Applicable	4010H
Software TRAP	2011h	2010h	Not Applicable	400FH
Extint	200Fh	200Eh	7(Highest)	400EH
Serial Port	200Dh	200Ch	6	400CH
Software Timers	200Bh	200Ah	5	400AH
HSI.0	2009h	2008h	4	4008H
High Speed Outputs	2007h	2006h	3	4006H
HSI Data Available	2005h	2004h	2	4004H
A/D Conversion Complete	2003h	2002h	1	4002H
Timer Overflow	2001h	2000h	0(Lowest)	4000H

Figure 1-1. Interrupt Vectors and Priorities

```
CSTART  MODULE  MAIN
        sp      equ     18H
        CSEG    AT 4100H
        ld      sp,#0c0h
        extrn   _main
        ljmp    _main
        ;
CSEG    AT 400EH
  extrn   exint
  ljmp    exint
        end
```

Ex) memory fix

```
echo rom(4000h-7fffh)          & >> t.cmd
echo ram(1ah-1ffh,8000h-0ffffh ) & >> t.cmd
```

C initialize function

```
;

        cseg  at  8000h
_c_init:
        sp        equ 18h:word
        stackline  equ 0ef00h
ld    sp, #stackline

        extrn  _main
        ljmp  _main

;
        ljmp  _exit
        rst
;
_exit:
        rst
end
```

About Interrupt

(1) INT00(Timer Overflow Interrupt) :
: 타이머 1 또는 타이머 2의 overflow 인터럽트, FFFFH~0000H로 되면 인터럽트 발생
: INT00의 인터럽트 가능 → IOC1(I/O제어 레지스터) 사용(그림 5.3)

(2) INT01(A/D Conversion Complete Interrupt)
: A/D 변환완료 인터럽트, A/D 변환이 완료되면 인터럽트 발생

(3) INT02(HSI Data Available Interrupt)
: HSI.의 data available 인터럽트
: IOC1.7 = 1 – Holding register와는 관계없이 6개의 데이터가 FIFO에 로드 될 때 인터럽트 발생
: IOC1.7 = 0 – Holding 레지스터에 사건/시간 값이 로드 될 때마다 인터럽트 발생

(4) INT03(HSO Interrupt)
: HSO 인터럽트
: HSO 명령에 의해 HSO.5~HSO.0에 대하여 각각 인터럽트 설정 가능

(5) INT04(HSI.0 pin Interrupt) : HSI.0 **핀 인터럽트**

(6) INT05(Software Timer Interrupt)
: S/W **타이머 인터럽트**
: **미리 정한 시간에 사건을 발생시키는 일은 이 인터럽트를 통하여 서비스 된다.**
: S/W **타이머는 4개가 동시에 인터럽트를 발생할 수 있다.**

(7) INT06(Serial Port Interrupt) : **직렬포트 인터럽트**

(8) INT07(EXTINT) : **외부 인터럽트**, P2.2 **또는** P0.7**의 외부 인터럽트**

(9) INT08(TI) : **직렬포트의 송신 인터럽트**

(10) INT09(RI) : **직렬포트의 수신 인터럽트**

(11) INT10(HSI. FIFO 4 Interrupt) : HSI. FIFO 4 **인터럽트**, FIFO**에 4개 이상의 데이터가 입력될 때 발생되는 인터럽트**

(12) INT11(TIMER2 Capture Interrupt) : **타이머** 2 **캡처 인터럽트**

(13) INT12(TIMER2 Overflow Interrupt) : **타이머** 2 overflow **인터럽트**

(14) INT13(EXTINT1) : P2.2**의 외부 인터럽트**

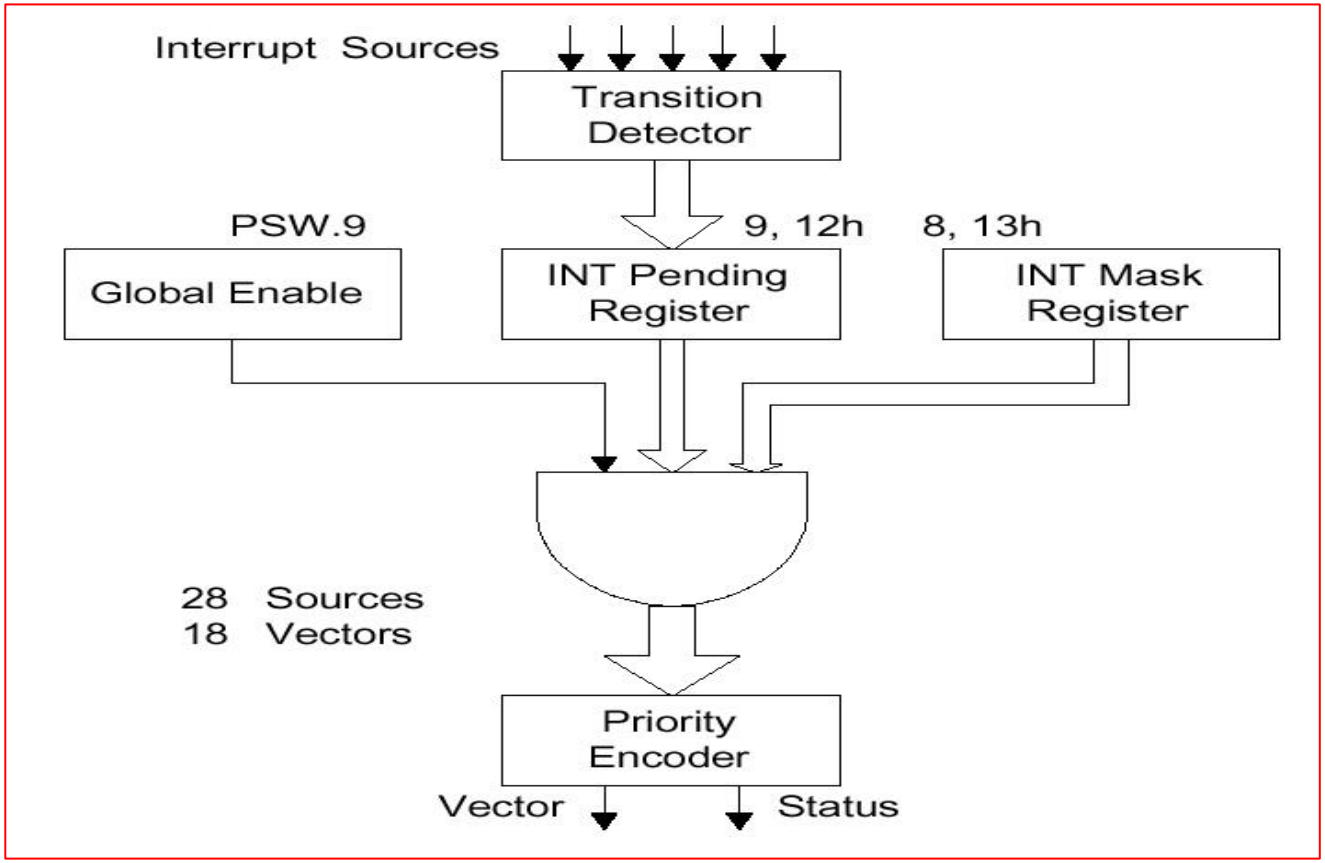
(15) INT14(HSI. FIFO Full Interrupt) : HSI.**의** FIFO**가 찰 때 발생되는 인터럽트**

(16) INT15(NMI) : **마스크 불능 인터럽트**(NMI, TRAP, **정의되지 않은 명령코드**)

(17) TRAP
: **명령코드** 0F7H**인** TRAP **명령이 실행되면** 2010H **번지를 통하여 간접으로 벡터 된다**.
: S/W **디버거를 만드는데 유용**

(18) **정의되지 않은 명령 코드** : **정의되지 않은 명령코드를 실행할 경우에 발생되는 인터럽트**

인터럽트의 구성요소


Interrupt Sources
Transition Detector
PSW.9
Global Enable
9, 12h
INT Pending Register
8, 13h
INT Mask Register
28 Sources
18 Vectors
Priority Encoder
Vector
Status

? ? ? ? ? ? ?

-. 선택된 인터럽트 소스(Interrupt Source)
: 인터럽트를 발생할 수 있는 장치 중, IOC1에 의해서 선택된 것

-. 변화 검출기(Transition Detector)
: 상승 에지(0 to 1)의 전압 변동 검출

-. 인터럽트 대기 레지스터(Interrupt Pending Register)
: 선택된 인터럽트 소스로부터 변화가 검출되면 해당 비트를 세트 시킴

-. 인터럽트 마스크 레지스터(Interrupt Mask Register)
: 인터럽트 대기 레지스터에 대기중인 인터럽트 중 실제로 발생시키고자 하는 것을 개별적으로 지정

-. 인터럽트 우선순위 인코더(Interrupt Priority Encoder)
: 인터럽트 우선순위를 결정하는 하드? 어

-. 인터럽트 벡터(Interrupt Vector)
: 인터럽트 서비스루틴을 처리하기 위해서 뛰어가는 주소가 들어 있는 번지

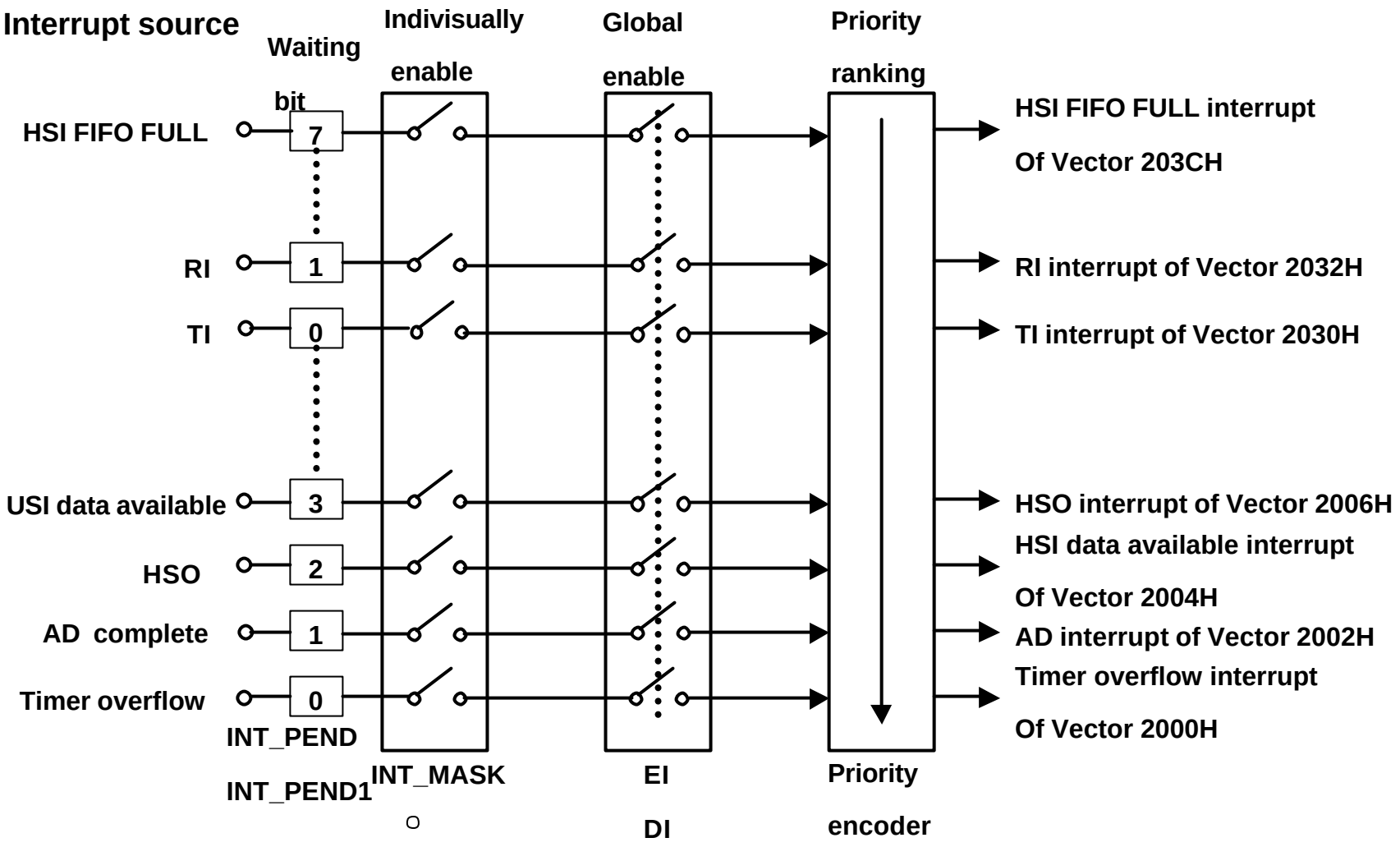
인터럽트 제어 레지스터 및 인코더

-. 인터럽트 제어 레지스터 및 인코더
: 인터럽트 대기 레지스터, 인터럽트 마스크 레지스터, 인터럽트 우선순위 인코더

-. 인터럽트 대기 레지스터
: 어느 인터럽트가 선택된 가운데 그 인터럽트가 발생된 사실을 소스별로 보존

-. 인터럽트 마스크 레지스터
: 인터럽트 가능/불능을 개별적으로 통제

-. 인터럽트 우선순위 인코더
: 인터럽트 소스간의 상대적인 우선순위를 하드? 어적으로 통제


Selected
Interrupt source
Waiting bit
Indivisually enable
Global enable
Priority ranking
HSI FIFO FULL
7
HSI FIFO FULL interrupt
Of Vector 203CH
RI
1
RI interrupt of Vector 2032H
TI
0
TI interrupt of Vector 2030H
USI data available
3
HSO interrupt of Vector 2006H
HSO
2
HSI data available interrupt
Of Vector 2004H
AD complete
1
AD interrupt of Vector 2002H
Timer overflow
0
Timer overflow interrupt
Of Vector 2000H
INT_PEND
INT_PEND1
INT_MASK
EI
DI
Priority encoder
? ? ? ? ? ? ? (2)

인터럽트 대기 레지스터

09H 번지의 INT_PENDING
12H 번지의 IPEND1(INT_PENDING1)
- 전압 변동을 검출하면 두 개의 인터럽트 대기 레지스터 중 어느 하나의 해당 비트를 세트

IPEND1(12h), IMASK1(13h)

7	6	5	4	3	2	1	0
NMI	FIFO full	EXT INT1	T2 OVF	T2 CAP	HSI4	Ri	TI

IPEND(09h), IMASK(08h)

7	6	5	4	3	2	1	0
EXT INT	SEREAL PORT	SOFT Timer	HSI.0 pin	HSO pin	HSI DATA	A/D Done	Timer OVF

? ? ? ? ? ? ? ? pm? ? ? pm? ? ? pm?

-. 인터럽트 대기 레지스터의 각 비트는 인터럽트 가능(enable), 불능(disable)과는 무관, 선택된 인터럽트 소스에 해당하는 사건이 발생하면 자동적으로 세트

-. 대기 비트는 인터럽트 벡터가 취해지면 자동적으로 클리어

-. 두 레지스터는 바이트 단위로 내용 판독/수정

-. 판독(read) : 어느 인터럽트가 대기중인가
수정(modify) : 특정 비트를 강제적으로 "0" 또는 "1"로 세트(인터럽트 발생과는 관계없음)
: 이 기능을 이용하여 s/w 인터럽트 발생

-. By. 부분 인터럽트 사이클
: 클리어 명령이 실행되는 동안에는 인터럽트가 인정되지 않도록 서로 불가분한 명령을 사용
→ 2~3개의 operand를 가진 논리명령 사용

```
Int_pending &= 0xfd      ; A/D 인터럽트 클리어

Int_pending |= 0x02      ; A/D 인터럽트 세트
```

<인터럽트 마스크 레지스터>

-. 인터럽트가 발생되려면…

: NMI를 제외한 모든 개개의 인터럽트는 인터럽트 마스크 레지스터(INT_MASK,~1)의 해당비트를 1로 세트함으로써 인터럽트가 마스크 되지 않아야 함

-. 인터럽트 개별 가능(enable) : 레지스터의 해당 비트 세트

-. 인터럽트 개별 불능(disable) : 해당 비트 클리어

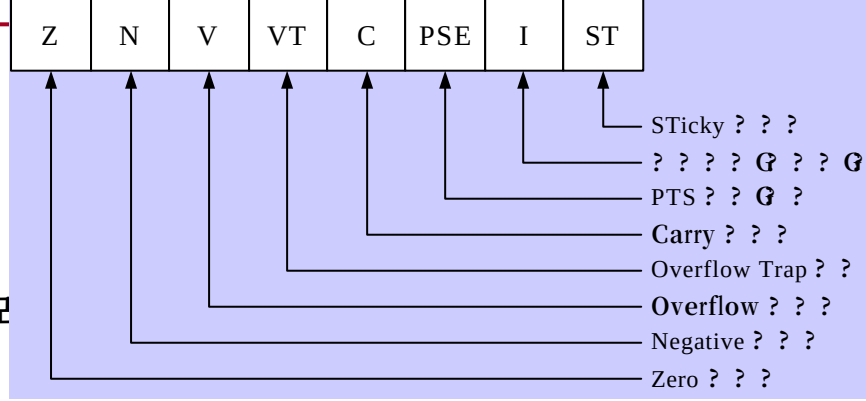
-. 인터럽트 마스크 레지스터

: 판독/기록 가능

: 클리어되어 있어도 인터럽트 발생시 대기 레지스터의 해당 비트 세트

-. INT_MASK : asm PUSHF 및 asm POPF 명령 실행시 레지스터의 내용 저장/복귀

-. asm PUSHF : 이들 명령이 실행된 후 disable



PSW

<인터럽트 전체 불능>

-. NMI, TRAP, 정의 되지 않은 명령 코드를 제외한 모
로써 전체적으로 disable됨

-. 인터럽트 마스크 레지스터가 세트 되어 있으면 인터럽트는 enable

-. 즉, 인터럽트 마스크 레지스터는 인터럽트를 개별적으로 enable
I 비트는 전체적으로 enable

<인터럽트 우선순위>

-. 2개 이상의 인터럽트가 동시에 발생할 경우에 어느 것부터 서비스를 해줄 것인가…

-. 인터럽트 우선순위 제어 : 우선순위 인코더(priority)

-. 대기중(pending)인 인터럽트 중에서 우선순위가 가장 높은 것을 선택

-. 우선순위가 결정되면 인터럽트 제어기가 프로그램 카운터로 하여금 지정된 인터럽트 벡터로 뛰어가도록 한다.

<인터럽트 우선순위의 수정>

-. 인터럽트 인코더는 대기중인 인터럽트가 인터럽트 콜을 통하여 s/w로 넘어가는 순서를 제어할 뿐이다.

-. 사용자는 인터럽트 마스크 레지스터(INT_MASK, ~1)를 이용하여 실질적인 우선순위를 바꿀 수 있다.

-. 우선순위 7인 EXTINT1 인터럽트만 가능(EXTINT1의 우선순위를 가장 높게 설정) 다른 어떠한 인터럽트도 불능(disable)으로 하기 위한 예

```
void serial_ri_isr(void)
asm pushf;
disable();
int_mask1 = 0x20;
enable();
```

"RI"인터럽트 서비스

```
asm popf;
return;
```

인터럽트 처리과정

-. 소스 선택, 개별 가능, 전체 가능, 우선순위, 주변장치의 개별 가능 (교재 참조)

-. 사건 발생 → 변화 검출기에서 검출 → 인터럽트 대기 레지스터의 해당 비트에 세트 → 우선순위 인코더에 의해 우선순위 결정 → 인터럽트가 CPU에서 인정 → 처리 과정

-. 처리 과정

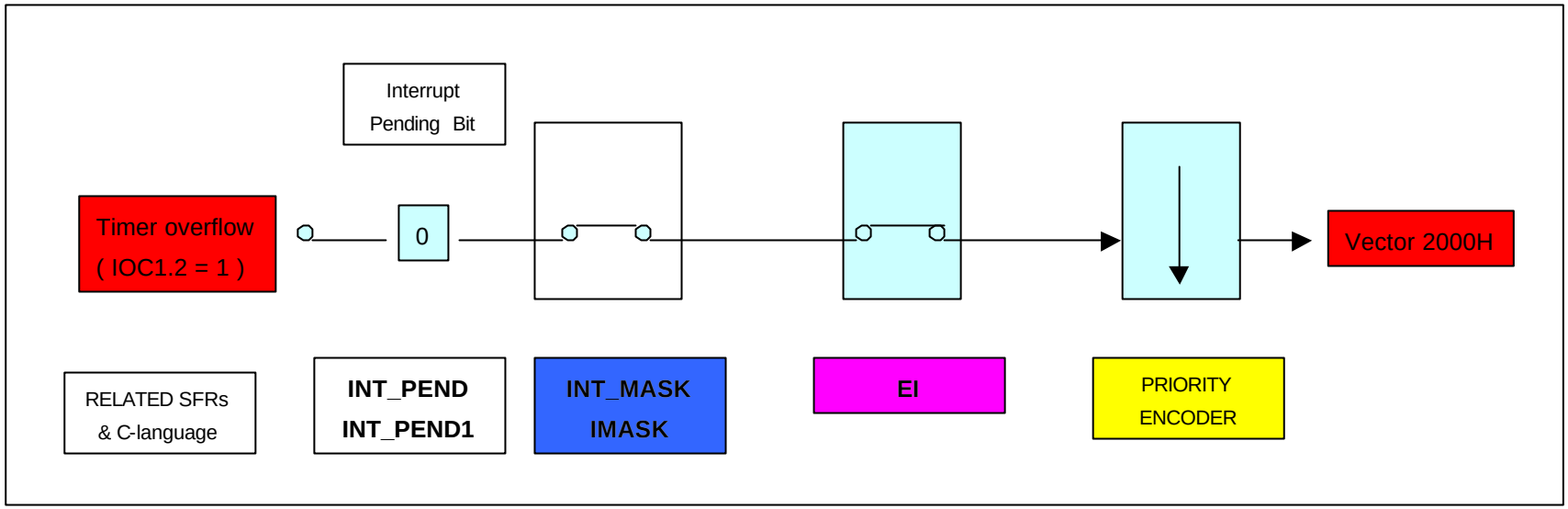
① CPU는 현재 실행중인 명령만은 마친 후, 특수 인터럽트-콜 명령을 실행

② asm pusha 명령을 실행

③ ①의 과정에서 PC에 로드 된 인터럽트 서비스루틴의 시작번지로 점프하여 인터럽트 처리루틴 실행

④ 인터럽트 CALL 명령으로 스택에 푸시 되었던 PSW, INT_MASK, WSR 등의 내용을 원상복귀

⑤ 메인 루틴에 복귀하여 중단된 명령 다음의 명령어부터 실행


Interrupt
Pending Bit
Timer overflow
(IOC1.2 = 1)
0
Vector 2000H
RELATED SFRs
& C-language
INT_PEND
INT_PEND1
INT_MASK
IMASK
EI
PRIORITY
ENCODER

인터럽트 처리 과정

인터럽트 타이밍 및 잠복시간

-. 외부 인터럽트 소스 : NMI, P2.2, HSI.0, P0.7

-. 인터럽트가 발생된다고 해서 항상 즉시 그 인터럽트가 인정되는 것은 아니다.

① 만약 현재 명령 사이클이 끝나기 전 4 상태 시간 이전에 인터럽트 신호가 발생되지 않으면 다음 명령 실행 후 인터럽트 인정

② 인터럽트 금지

; enable(), disable() – 전체 불능 비트(PSW.9)를 "1" 또는 "0"으로 하므로 인터럽트를 전체적으로 가능 또는 불능으로 한다.

; asm pushf – PSW/INT_MASK 쌍을 스택에 푸시한 후 이들을 클리어 하므로 인터럽트 불능

; asm popf – PSW/INT_MASK 쌍을 스택에 팝한다.

; asm pusha – asm pushf를 하고, IMASK1/WSR 쌍을 푸시한 후, IMASK1을 클리어 한다.

; POPA – IMASK1/WSR 쌍을 팝하고 asm popf를 한다.

③ 다음의 명령들이 실행된 후 바로 뒤에는 인터럽트가 발생되지 않는다.

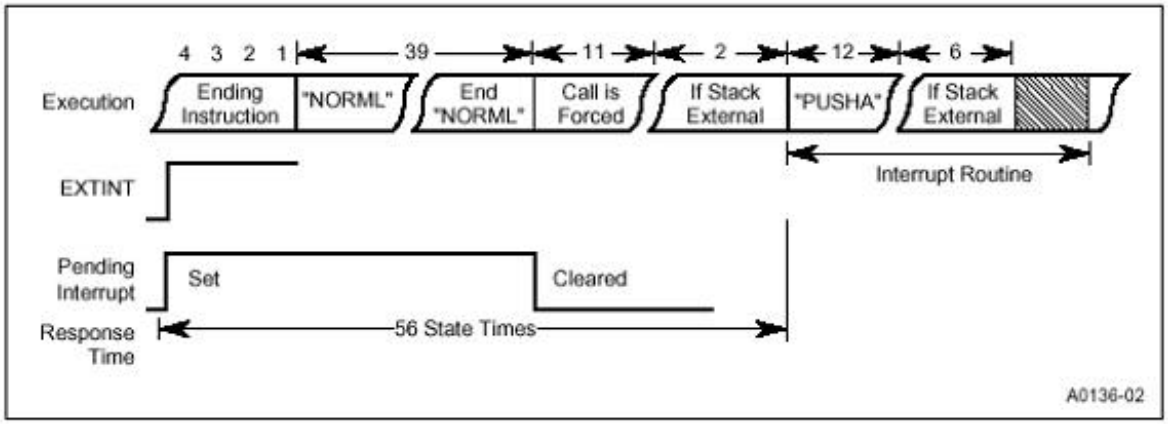
; 정의되지 않은 명령코드

; TRAP – 소프트? 어 TRAP 명령

; SIGND(EXT, EXTB) – 곱셈 및 나눗셈 명령을 위한 부호가 있는 접두부(signed prefix)

-. 인터럽트가 인정되면 인터럽트 대기 비트(INT_PENDING, IPEND1)는 자동적으로 클리어되고, 특정 인터럽트 벡터가 지시하는 번지로 CALL된다.

-. 잠복시간(latency time)
: 인터럽트가 발생되고부터 인터럽트 처리 루틴의 최초 명령을 실행하기까지의 시간



인터럽트 응답시간

외부 인터럽트

-. **외부 인터럽트** : NMI(INT15), EXINT1(INT13), EXINT(INT07), HSI.0(INT04)

-. NMI**를 제외하고 모두** I/O **제어 레지스터** IOC1**으로 그 소스가 선택**

-. IOC1.1 : EXINT**와** EXINT1 **중 선택**

-. IOC1.7 : HSI.0 **인터럽트를 선택**

① IOC1.1=0**이면** EXINT(P2.2)**가 인터럽트 소스로 선택**

② IOC1.1=1**이면** EXINT1(P0.7)**이 인터럽트 소스로 선택**

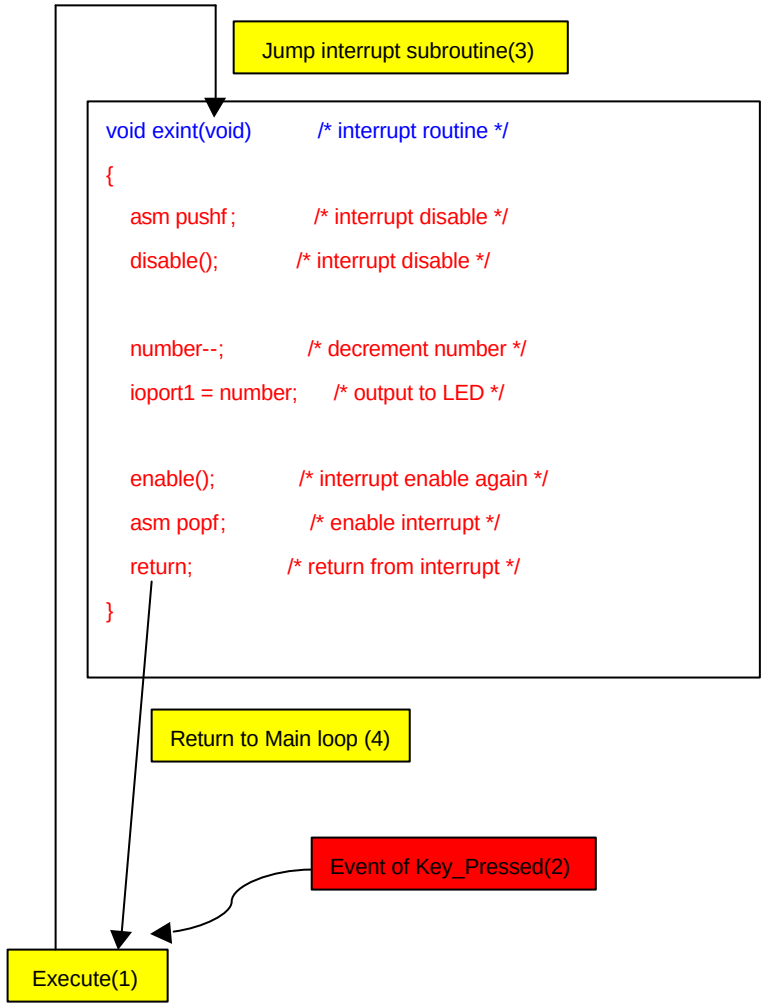
③ IOC1.7=1**이면** HSI.0**가 인터럽트 소스**

④ NMI**는 어떤 제어 레지스터의 간섭도 받지 않음**

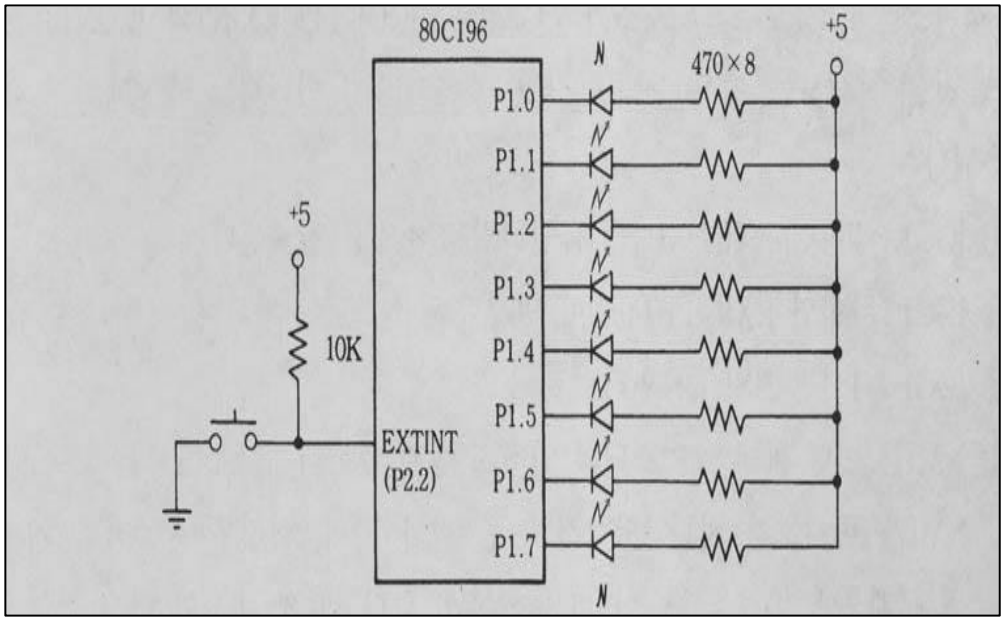
그림5.11 I/O **제어 레지스터**1(IOC1) **참조**

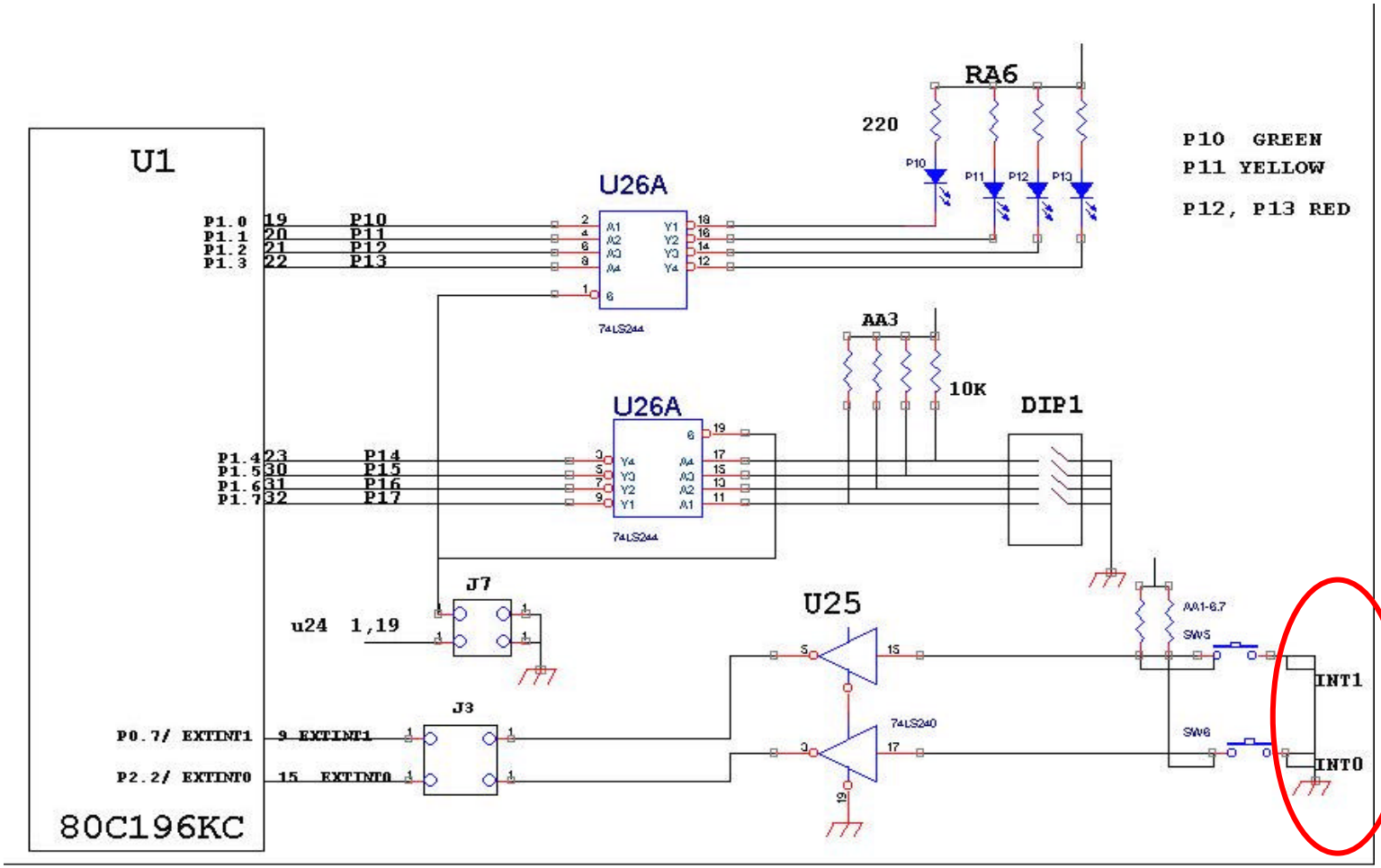
C-Language for using Interrupt

```
/*=====================================*/
/* PROGRAM EX5_1.C) [EXTERNAL INTERRUPT]*/
/*=====================================*/

#pragma model(kc)          /* CPU model=80C196KC      */
#pragma interrupt(exint = 7) /* define interrupt number(vector) */
#include <80C196.h>        /* include 80C196 header file */

unsigned char number;      /* number=8bit variable */

void main(void)            /* main function,no arguement */
{
   disable();
   ioc1 = 0x00;            /* select EXT_INT(P2.2) */
   int_pending = 0x00;     /*clear INT_PEND register */
   int_mask = 0x80;        /* EXT_INT enable */

   ioport1 = 0xff;         /*all LED is off */
   number = 0xff;          /* initial number */

   enable();               /* global interrupt enable */
   while(1);               /* wait interrupt */
}
```

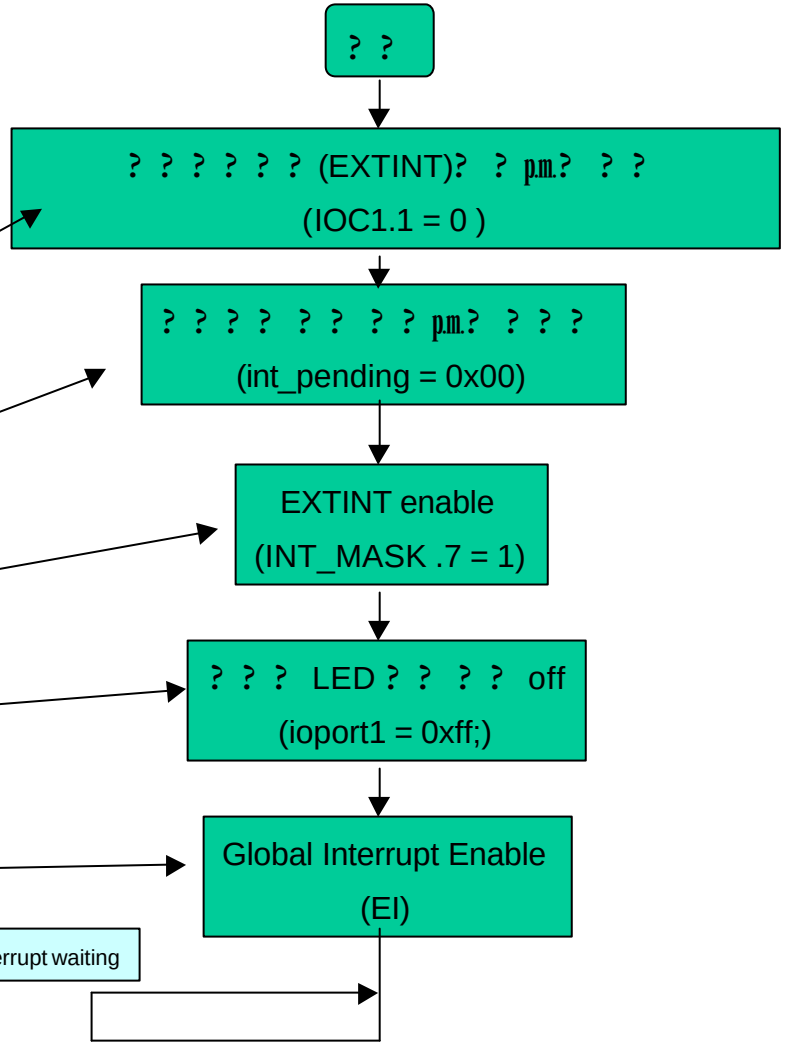


- Objective: - **Use of external interrupt : EXTINT(P2.2)**
  - If you press the button, LED connected to port1 is shifted sequentially.
  - All LEDs are dead before initial press of button.

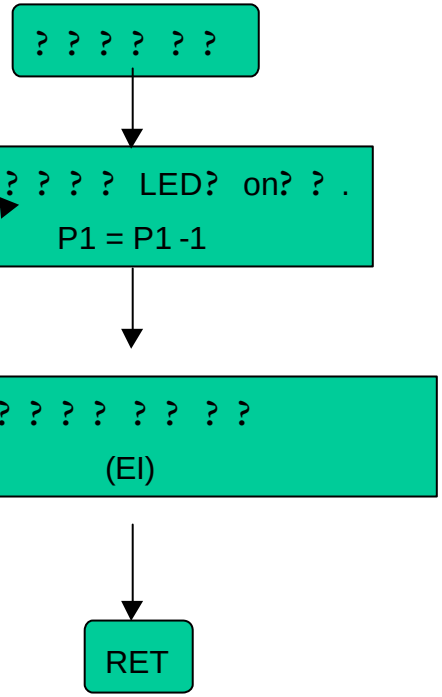



U1
80C196KC
P1.0 19 P10
P1.1 20 P11
P1.2 21 P12
P1.3 22 P13
P1.4 23 P14
P1.5 30 P15
P1.6 31 P16
P1.7 32 P17
P0.7/ EXTINT1 9 EXTINT1
P2.2/ EXTINT0 15 EXTINT0
U26A
74LS244
U26A
74LS244
RA6
220
P10
P11
P12
P13
P10 GREEN
P11 YELLOW
P12, P13 RED
AA3
10K
DIP1
J7
u24 1,19
J3
U25
74LS240
AA1-8.7
SW5
SW6
INT1
INT0

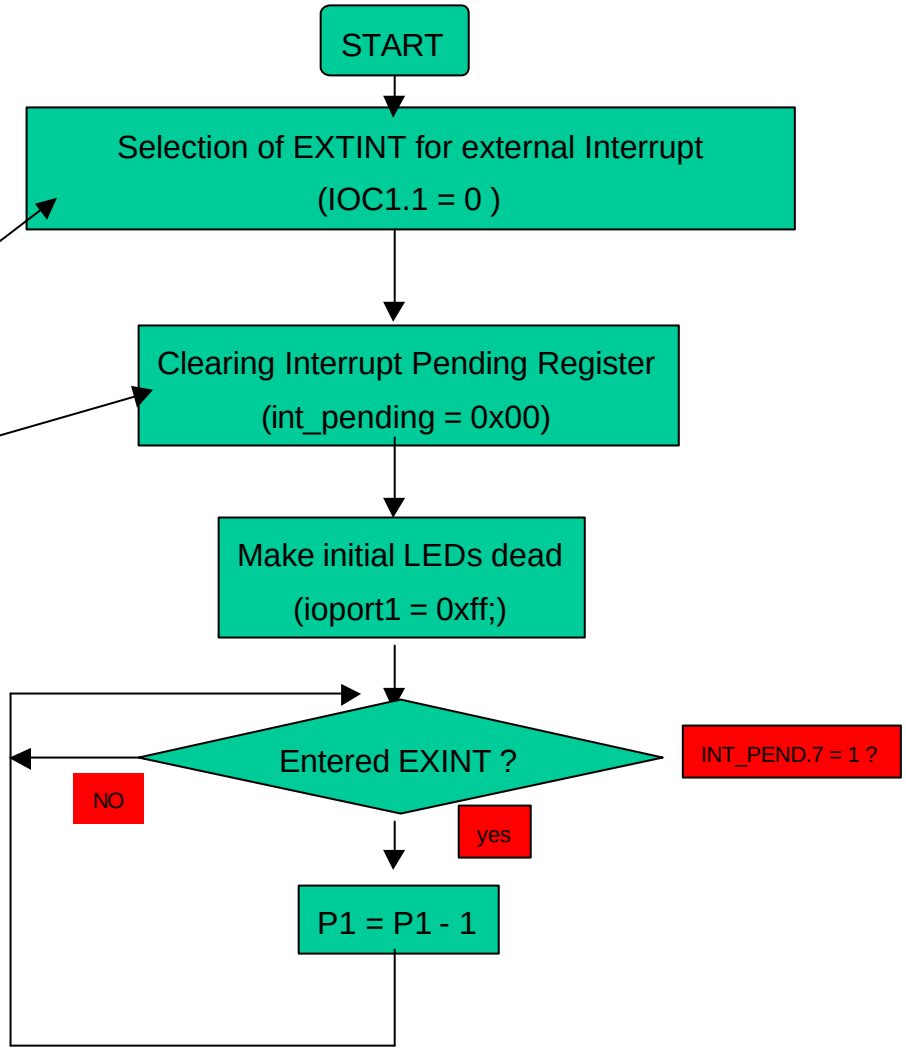
```
/*=====================================*/
/* PROGRAM EX5_1.C) [EXTERNAL INTERRUPT]*/
/*=====================================*/

#pragma model(kc)          /* CPU model=80C196KC      */
#pragma interrupt(exint = 7) /* define interrupt number(vector) */
#include <80C196.h>        /* include 80C196 header file */

unsigned char number;      /* number=8bit variable */

void main(void)            /* main function,no arguement */
{
   disable();
   ioc1 = 0x00;            /* select EXT_INT(P2.2) */
   int_pending = 0x00;     /*clear INT_PEND register */
   int_mask = 0x80;        /* EXT_INT enable */

   ioport1 = 0xff;         /*all LED is off */
   number = 0xff;          /* initial number */

   enable();               /* global interrupt enable */
   while(1);               /* wait interrupt */
}
```



```
void exint(void)          /* interrupt routine */
{
    asm pushf;            /* interrupt disable */
    disable();            /* interrupt disable */

    number--;             /* decrement number */
    ioport1 = number;     /* output to LED */

    enable();             /* interrupt enable again */
    asm popf;             /* enable interrupt */
    return;               /* return from interrupt */
}
```



```
/*=============================================*/
/* PROGRAM EX5_2.C) [EXTERNAL INTERRUPT by FLAGS]*/
/*=============================================*/
#pragma model(kc)          /* CPU model=80C196KC      */
#include <80C196.h>        /* include 80C196 header file */

unsigned char number,flag; /* number=8bit variable */

void main(void)            /* main function,no arguement */
{
   ioc1 = 0x00;           /* select EXT_INT(P2.2) */
   int_pending = 0x00;    /*clear INT_PEND register */

   number = 0xff;         /* initial number */

   do{
     do{                  /*int_pending.7=1?*/
        flag = int_pending;
     }while((flag & 0x80) == 0);

     int_pending &= 0x7f; /*int_pending.7=0*/
     number--;            /*update output data*/
     ioport1 = number;
  }while(1);              /*repeat*/
}
```



Objective: - **Use of INT_PEND register instead of external interrupt. (Polling method)**

- If you press the button, LED connected to port1 is shifted sequentially.
- All LEDs are dead before initial press of button EXTINT(P2.2)

: s/w ? ? ? ? ? ?

```
/*=====================================*/
/* PROGRAM EX5_3.C) [EXTERNAL INTERRUPT]*/
/*=====================================*/
#pragma model(kc)          /* CPU model=80C196KC      */
#pragma interrupt(exint = 7) /* define interrupt number(vector) */
#include <80C196.h>        /* include 80C196 header file */

unsigned char number;      /* number=8bit variable */
/* MAIN ROUTINE */
void main(void)            /* main function,no arguement */
{
   ioc1 = 0x00;            /* select EXT_INT(P2.2) */
   int_pending = 0x00;     /*clear INT_PEND register */
   int_mask = 0x80;        /* EXT_INT enable */

   enable();               /* global interrupt enable */
   int_pending |= 0x80;    /* set int_pending bit external */
   while(1);               /* wait interrupt */
}

/* SUBROUTINE */
delay(short i)             /* delay routine */
{
   while(i--);
}
```

```
/* INTERRUPT ROUTINE */
void exint(void)           /* interrupt routine */
{
   disable();              /* interrupt disable */
   do{
      ioport1 = 0xff;
      delay(0xffff);
      ioport1 = 0x00;
      delay(0xffff);
   }while(1);
   return;
}
```