

제 9 장 7-세그먼트의 동적 표시

9.1 동작원리

7.5.3절 과제 2에서 사용한 7-세그먼트는 LCD와 더불어 마이크로컨트롤러의 응용에서 표시장치로 빈번히 사용되는 장치이다. 7.5.3절에서 사용한 바와 같이 하나의 7-세그먼트를 구동하는 데 8비트 출력포트 1개가 필요하다. 일반적인 응용에서는 7-세그먼트에 표시하여야 하는 수의 자리 수가 여러 개이어서 상당히 많은 출력포트가 필요함을 알 수 있다. 4 자리수를 표현하는 데 4개의 출력포트, 즉 32핀의 출력핀이 7-세그먼트를 구동하는 데 사용되어야 한다. ATmega128에서는 53핀의 입출력 핀을 제공하지만 32핀은 적은 수가 아니다. 이의 해결방법 중 하나가 동적으로 7-세그먼트를 구동하는 것이다.

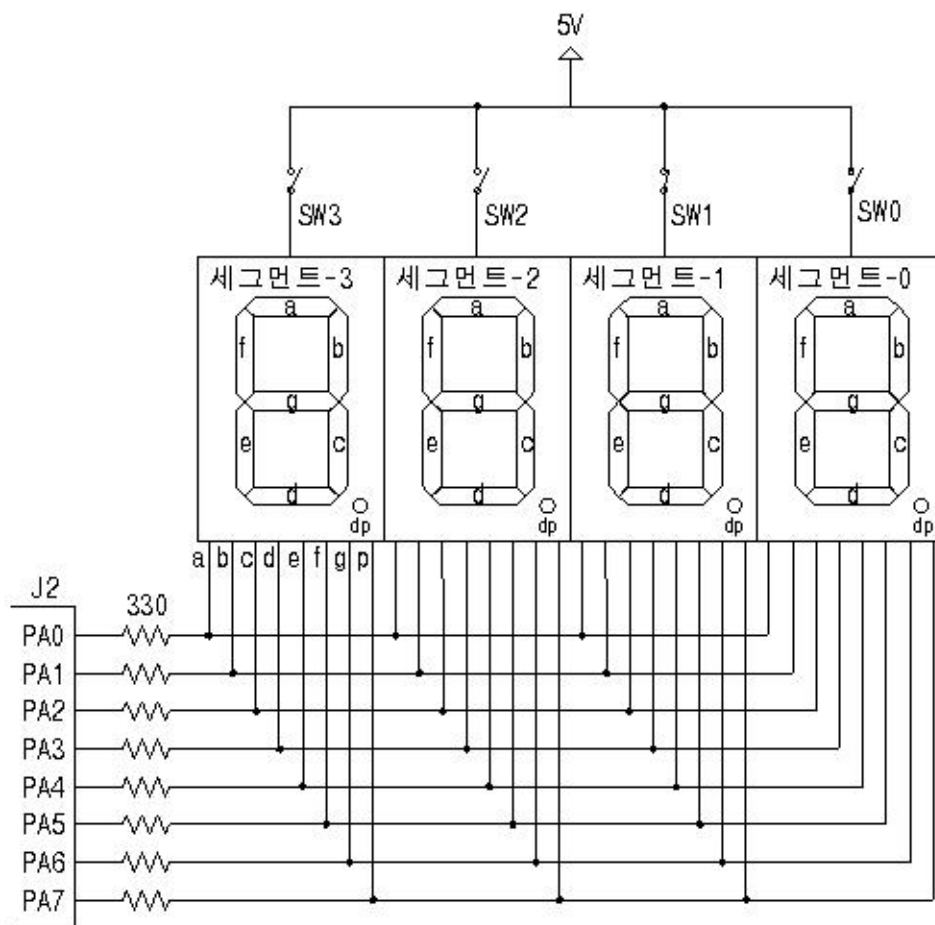


그림 9.1 동적디스플레이를 위한 7-세그먼트회로

6.7절의 실습에서 관찰된 것처럼 LED패턴의 회전을 빠른 속도로 하게 되면 모든 LED가 켜져 있는 것처럼 보인다. 이는 눈의 잔상효과 때문이다. 이를 이용하여 7-세그먼트를 한 순간에 1개씩 표시를 하고 이를 아주 빠른 속도로 자리수를 돌아가면서 수를 표시하면 사람은 모든 자리수의 수가 한꺼번에 표시된 것으로 느낀다. 이런 방법을 동적 표시(Dynamic Display)방법이라 하고 앞의 방법을 정적 표시 방법이라 한다.

네 자리수를 Anode Common형 7-세그먼트로 동적 표시하기 위해 세그먼트를 켜고/끄고 할 수 있는 스위치를 장착한 그림 9.1의 회로를 고려하여 보자. 포트-A의 데이터 출력은 스위치가 ON되어 있는 7-세그먼트에 표시된다. 그림 9.1에서는 SW1이 ON되어 있으므로 해당하는 세그먼트-1에 포트-A의 데이터가 표시된다.

다음 방법으로 스위치를 동작하고 포트-A에 데이터를 출력하여 보자.

1. 스위치 SW0을 켜고, 숫자 4을 표시하기 위한 데이터를 포트-A에 출력
=> 세그먼트-0에 '4'가 표시
2. 스위치 SW1을 켜고, 숫자 3을 표시하기 위한 데이터를 포트-A에 출력
=> 세그먼트-1에 '3'이 표시
3. 스위치 SW2를 켜고, 숫자 8을 표시하기 위한 데이터를 포트-A에 출력
=> 세그먼트-2에 '8'이 표시
4. 스위치 SW3을 켜고, 숫자 7을 표시하기 위한 데이터를 포트-A에 출력
=> 세그먼트-3에 '7'이 표시
5. 순서 1부터 다시 시작

- ☞ 세그먼트-0부터 세그먼트-3으로 돌아가면서 '4', '3', '8', '7'이 표시된다. 수가 표시되는 간격은 스위치의 ON/OFF 간격과 일치한다.
- ☞ 스위칭 간격이 길면 '4', '3', '8', '8'순으로 돌아가면서 켜진다.
- ☞ 간격이 매우 짧으면 '4', '3', '8', '7'이 한꺼번에 켜져 있는 것처럼 보인다. 즉 4개의 7-세그먼트에 수 "7834"가 표시된다.

==> 스위칭 간격을 짧게 함으로써 동적표시를 할 수 있다.

- ☞ 4자리 수를 표시할 때 정적 표시 방법을 사용하면 $4 \times 8 = 32$ 핀의 입출력이 필요하지만, 동적표시를 하면 12개의 입출력 핀이 필요하다.

9.2 쌍극트랜지스터를 사용한 스위칭

동적표시를 위해서는 스위치가 중요한 요소이다. 마이크로컨트롤러에서 동적표시를 응용하고자 하면 그림 9.1의 기계적 스위치 대신 제어신호로서 ON/OFF를 할 수 스위치가 필요하다. 제어신호로 스위칭을 할 수 있는 소자로서 쌍극트랜지스터(Bipolar Junction Transistor, BJT)가 있다. 그림 9.2는 NPN형과 PNP형 BJT를 스위치로 사용하는 회로를 보여준다.

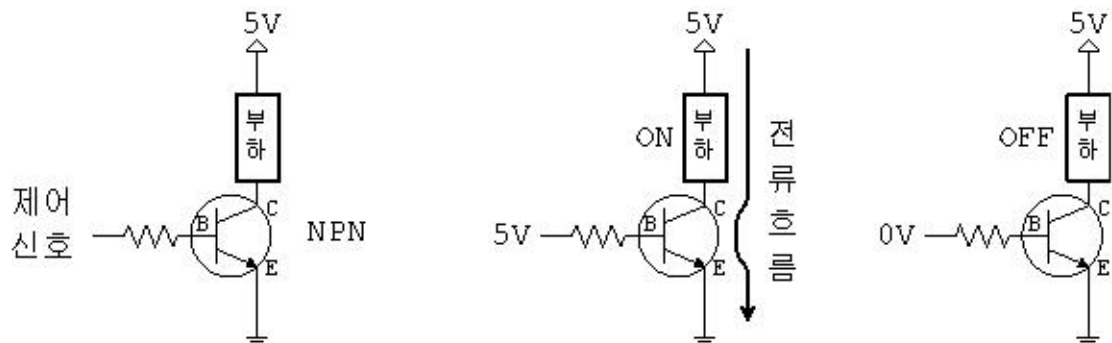


그림 9.2(a) NPN 트랜지스터를 사용한 스위칭

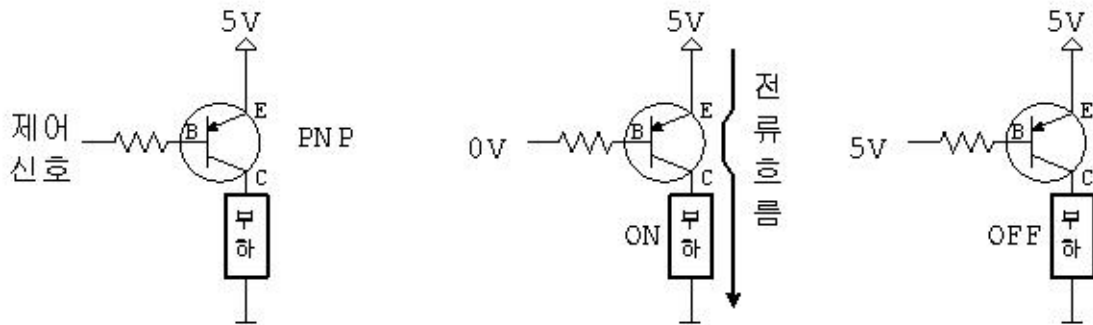



















그림 9.2(b) PNP 트랜지스터를 사용한 스위칭

- ☞ NPN형과 PNP형 모두 콜렉터에 스위칭할 부하를 연결하여야 한다.
- ☞ NPN형은 제어신호의 전압이 에미터의 전압보다 높으면 ON, 낮으면 OFF가 된다.
- ☞ PNP형은 제어신호의 전압이 에미터의 전압보다 낮으면 ON, 높으면 OFF가 된다.
- ☞ 그림 9.1의 기계적 스위치는 PNP형 트랜지스터를 사용한 전자적 스위치로 대체할 수 있다.

표 9.1 7-세그먼트의 16진수 표시 코드 값

테이블 위치	비트위치 표시	D7	D6	D5	D4	D3	D2	D1	D0	코드값
		dp	g	f	e	d	c	b	a	
0		0	0	1	1	1	1	1	1	0x3F
1		0	0	0	0	0	1	1	0	0x06
2		0	1	0	1	1	0	1	1	0x5B
3		0	1	0	0	1	1	1	1	0x4F
4		0	1	1	0	0	1	1	0	0x66
5		0	1	1	0	1	1	0	1	0x6D
6		0	1	1	1	1	1	0	1	0x7D
7		0	0	0	0	0	1	1	1	0x07
8		0	1	1	1	1	1	1	1	0x7F
9		0	1	1	0	1	1	1	1	0x6F
10		0	1	1	1	0	1	1	1	0x77
11		0	1	1	1	1	1	0	0	0x7C
12		0	1	0	1	1	0	0	0	0x58
13		0	1	0	1	1	1	1	0	0x5E
14		0	1	1	1	1	0	0	1	0x79
15		0	1	1	1	0	0	0	1	0x71
16		0	0	0	0	0	0	0	0	0x00

※ 위 코드 값은 1을 출력하면 LED가 켜질 때의 값이다. 그림 5.1의 회로에서는 0을 출력할 때 LED가 켜지므로 위 코드 값에 비트별 NOT을 취하여 출력해야 한다.

표 9.1은 7-세그먼트에 수를 표시하기 위해 포트-A로 출력하여야 할 코드를 보여주고 있다. 표 9.1은 핀에 1을 출력하면 LED가 켜지는 정논리의 코드값을 나타내는 반면 그림 9.2의 회로는 핀에 0을 출력하여야 LED가 켜지는 부논리이므로 7-세그먼트에 코드를 내보낼 때는 표9.1의 코드에 비트별 NOT을 취하여 내보내야 한다.

예) 그림 9.2의 회로에 10진수 7834를 표시하여 보자. 트랜지스터를 켜는 포트는 포트 D이고 코드를 출력하는 포트는 포트 A이다.

- 1) 포트 D에 ~0x10(0b11101111)을 출력한다.(Tr0를 켜다.)
- 2) 포트 A에 숫자4에 해당하는 표 9.1의 코드 값 ~0x66을 출력한다.
- 3) 포트 D에 ~0x20(0b11011111)을 출력한다.(Tr1를 켜다.)
- 4) 포트 A에 숫자3에 해당하는 표 9.1의 코드 값 ~0x4F를 출력한다.
- 5) 포트 D에 ~0x40(0b10111111)을 출력한다.(Tr2를 켜다.)
- 6) 포트 A에 숫자8에 해당하는 표 9.1의 코드 값 ~0x7F를 출력한다.
- 7) 포트 D에 ~0x80(0b01111111)을 출력한다.(Tr3를 켜다.)
- 8) 포트 A에 숫자7에 해당하는 표 9.1의 코드 값 ~0x07을 출력한다.
- 9) 절차 1)부터 다시 반복한다.

9.4 동적 표시 프로그램 I

위 예제를 실행하기 위해 프로그램 9.1을 작성할 수 있다.

- ☞ 표 9.1의 코드 값은 6절과 같이 배열을 이용한 테이블로 구성하였다.
- ☞ 7-세그먼트에 출력할 데이터 역시 배열에 저장하였다.
- ☞ 하나의 자리수를 표시한 후 시간지연을 준다. 시간지연이 너무 짧으면 표시가 너무 흐리고, 시간지연이 너무 길면 표시가 깜박인다.

실습 1 : 시간 지연을 1, 10, 20msec로 변경하고 화면표시를 관찰한다.

(※ `_delay_ms()` 함수는 최대 16msec지연만을 할 수 있음을 유의할 것)

```

#include <avr/io.h>
#include <util/delay.h>
static unsigned char SegTable[17] =
    {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
     0x7F, 0x6F, 0x77, 0x7C, 0x58, 0x5E, 0x79, 0x71,
     0x00};
int main()
{
    unsigned char cnumber[4] = {4,3,8,7};
    int j=0;

    DDRA = 0xFF; // 포트 A를 출력으로 설정
    DDRD |= 0xF0; // 포트 D의 상위니블을 출력으로 설정
    while(1)
    {
        for(j=0; j<4; j++)
        {
            PORTD = ~(0x10<<j); // j-번째 세그먼트를 켜다.
            PORTA = ~SegTable[cnumber[j]];
            _delay_ms(5); // 5msec 시간지연
        }
    }
}

```

프로그램 9.1

프로그램 9.1은 10진수 7834를 표시하기 위해 사용자가 배열에 {4, 3, 8, 7}를 저장하였다. 이와 같이 어떤 10진수가 주어지면 자리 수 별로 배열에 저장하는 것을 고려하여 보자. 예와 같이 10진수 7834에 대해

$$\begin{array}{rcl}
 7834 / 10 = \underline{783} \dots \overset{4}{\curvearrowright} & \text{나머지 } 4 \Rightarrow & \text{1자리} \quad (\text{cnumber}[0]) \\
 783 / 10 = \underline{78} \dots \overset{3}{\curvearrowright} & \text{나머지 } 3 \Rightarrow & \text{10자리} \quad (\text{cnumber}[1]) \\
 78 / 10 = \underline{7} \dots \overset{8}{\curvearrowright} & \text{나머지 } 8 \Rightarrow & \text{100자리} \quad (\text{cnumber}[2]) \\
 7 / 10 = \underline{0} \dots \overset{7}{\curvearrowright} & \text{나머지 } 7 \Rightarrow & \text{1000자리} \quad (\text{cnumber}[3])
 \end{array}$$

이다. 위 연산에서 나머지가 각 자리수의 값을 나타내며 연산은 몫이 0이 될 때까지 수행하면 된다. 위에서는 10진수로 표기하기 위하여 10으로 나누면서 연산을 하였다. 만약 16으로 나누면서 나머지를 취하면 16진수의 각 자리를 얻을 수 있다.

진법이 변수 radix로 주어졌을 때 부호 없는 2바이트 정수 num으로부터 각 자리수를 추출하여 배열에 저장하는 과정을 프로그램 9.2와 같이 작성할 수 있다.

```
// num      : 부호 없는 2바이트 입력 변수(표시할 수)
// radix    : 진법
// cnumber[] : 각 자리의 수를 저장할 배열

j = 0;           // 1-자리수
while(num)       // 몫이 0이 아니면 계속
{
    cnumber[j++] = num % radix; // 나머지
    num /= radix;              // 몫
}
```

프로그램 9.2

프로그램 9.1과 9.2를 조합하면 부호없는 2바이트 정수 num과 진법 radix를 인자로 받아 4자리수의 7-세그먼트에 동적으로 표시하는 함수를 프로그램 9.3과 같이 작성할 수 있다.

- ☞ 인수 num은 표시할 수, radix는 표시할 수의 진법을 나타낸다.
- ☞ 부호없는 2바이트정수의 최대 값은 16진수로 0xFFFF이고 10진수로 65535이어서 각각 4자리와 5자리수가 된다. 이를 모두 저장하기 위하여 배열 cnumber[]의 크기를 5로 하였다.

- ☞ 10진수일 때 cnumber[j]는 10^j -자리수의 수를 저장하고 있다. 즉 cnumber[1]은 $10^1=10$ 자리의 수를 저장하고 있다.
- ☞ 7-세그먼트가 4자리만 표시할 수 있어 5자리 10진수의 최상위 자리는 표시되지 않는다.

```
static unsigned char SegTable[17] = // 7-세그먼트 코드 값
    {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
     0x7F, 0x6F, 0x77, 0x7C, 0x58, 0x5E, 0x79, 0x71,
     0x00};
static unsigned char cnumber[5]; // 수를 저장할 배열

void Seg7DispNum(unsigned short num,
                 unsigned short radix)
{
    int j;
    // cnumber[]가 수0을 저장
    cnumber[0] = 0;
    for(j=1; j<5; j++) cnumber[j] = 16;

    j=0; // 1자리를 가리킴
    while(num) // num의 각 자리를 분리하여 배열에 저장
    {
        cnumber[j++] = num % radix; //나머지
        num /= radix; // 몫
    }
    for(j=0; j<4; j++) // 1자리부터 4개를 켜다.
    {
        // j-번째 세그먼트를 켜다.
        PORTD = (PORTD & 0x0F) | ( ~(0x10<<j) & 0xF0);
        PORTA = ~SegTable[cnumber[j]];
        _delay_ms(5); // 5msec delay
    }
}
```

프로그램 9.3

☞ C-언어의 함수

- ▶ 특별한 일을 수행하는 코드 집합
- ▶ C-언어에서 프로그램 모듈화의 기초 구성체

☞ 함수의 정의

- ▶ 함수 머리부분 : `fctype name(type1 v1, type2 v2, . . .)`
 - `fctype` : 함수의 타입
 - 함수의 반환 값의 데이터 형에 따라 명시(`int`, `char` 등)
 - 반환 값이 없는 경우 "`void`"로 명시
 - 생략 시에는 "`int`"로 간주
 - `name` : 함수의 이름이며 함수 내용을 나타낼 수 있도록 지정
 - `v1, v2` : 인자
 - 없을 수도 있고 여러 개의 인자가 있을 수도 있음
 - 인자 앞에는 인자의 데이터 형(`int`, `char` 등)을 지정
(`type1`, `type2`는 각각 `v1`, `v2`의 데이터 형을 나타냄.)
- ▶ 함수 몸체
 - { } 안에 함수 기능 구현
 - 함수의 반환 값
 - 하나를 가지며 `return`문으로 반환 값 명시(예: `return x;`)
 - 반환 값이 없는 경우 `return` 또는 생략

예) `int add(int x, int y)`
`{`
 `return (x+y);`
`}`

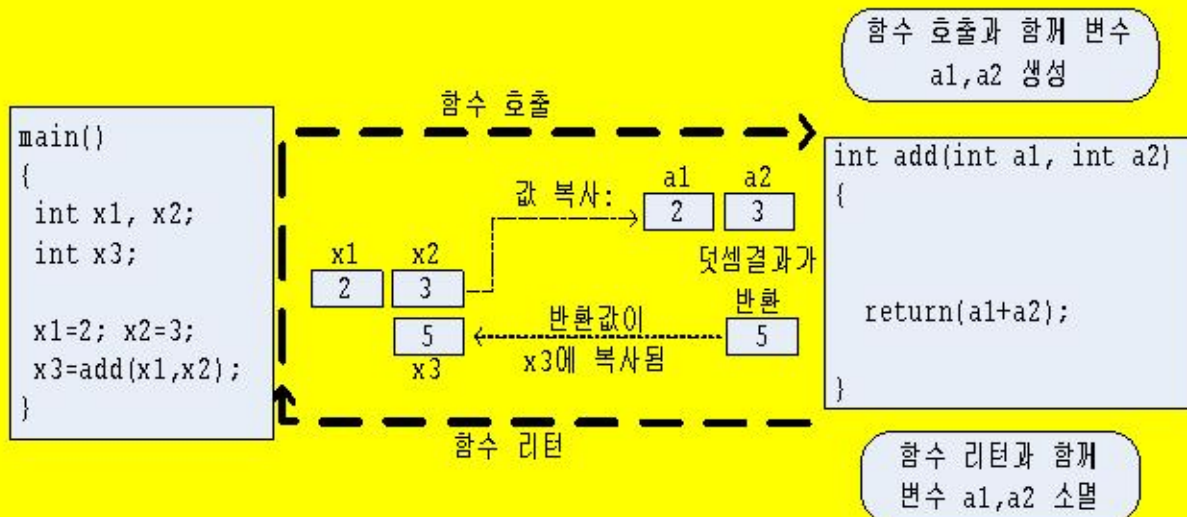
☞ 함수의 입출력

함수에 필요한 데이터의 입력은 인자를 통해 전달받고, 함수의 결과 출력은 함수의 인자 또는 반환 값으로 전달한다.

- | | |
|-----------|--|
| 인수로 입력 | - 다수의 입력을 받을 수 있음. |
| 인수로 출력 | - 다수의 출력을 할 수 있음 |
| 반환 값으로 출력 | - 하나의 출력만 할 수 있다. 주로 함수의 수행상태(성공, 실패)등을 전달하는 데 사용된다. |

☞ 인수 입력과 반환 값 출력

- ▶ 함수는 인자를 통하여 함수 호출자의 정보를 입력받는다. 아울러 함수는 반환 값으로 호출자에게 정보를 전달할 수 있다. 다음 그림은 함수 `add()`가 호출자로부터 두 값을 받아 덧셈을 한 후 덧셈 결과 값을 반환 값으로 호출자에 전달하는 과정을 보여준다.



- ▶ 함수의 호출과 동시에 함수의 인자인 변수 `a1`과 `a2`는 생성된다.
- ▶ 호출자는 함수를 호출할 때 인자에 데이터 `x1`, `x2`를 전달한다. `x1`과 `x2`의 값은 함수의 인자 `a1`과 `a2`에 각각 복사된다.
- ▶ 함수는 두 인자의 덧셈을 수행하고 반환 값을 호출자에 전달한다. 이 때 그림과 같이 반환 값을 받는 변수 `x3`가 있을 때는 반환 값이 `x3`에 복사된다. 받는 변수가 없을 때는 반환 값은 사라진다.
- ▶ 함수의 인자 `a1`과 `a2`는 자동변수이므로 함수가 리턴이 되면서 소멸된다.

☞ 인수 출력

함수는 인수를 사용하여 함수의 정보를 호출자에게 전달할 수 있다. 이 방법은 다음 10.1.2절에서 설명할 것이다.

- ☞ 프로그램 9.3에서 제어포트 D에 출력하는 부분은 프로그램 9.2와는 달리 다음과 같이 작성하였다.

```
PORTD = (PORTD & 0x0F) | ( ~(0x10<<j) & 0xF0);
```

- ☞ 이는 포트 D의 하위 4비트는 7-세그먼트와는 상관없는 다른 부분에 사용할 수 있으므로 이 비트들의 출력은 변경시키지 않고 7-세그먼트의 제어출력인 변경된 상위 4비트만 출력으로 하기위한 것이다.

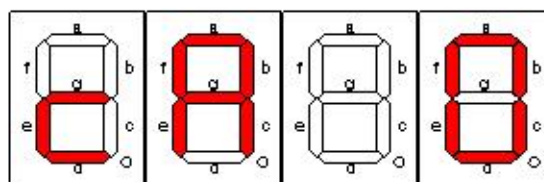
7-세그먼트 표시기능은 유용하게 쓸 수 있는 기능이므로 관련 함수 및 데이터를 하나의 파일로 작성하여 다른 응용프로그램에서 해당 파일을 프로젝트에 포함시키면 관련함수들을 사용할 수 있도록 하자. 파일명은 기능을 나타낼 수 있는 것으로 선택하여 소스파일을 seg7.c라하고, 헤더파일을 seg7.h라 하여 작성하자. 소스파일에 포함되는 함수는 7-세그먼트 구동 프로그램임을 나타내기 위해 접두어 seg7을 사용한다.

- ☞ 7-세그먼트 출력을 위한 포트 설정을 초기화함수 seg7Init()에서 수행하였다. 제어포트와 데이터포트는 매크로를 사용하여 정의한다. 제어포트와 데이터포트가 변경되면 해당 매크로만 변경하면 된다.
- ☞ 제어출력으로 사용하는 비트는 매크로 CONT_MASK와 OUTPUT_VALUE를 사용하여 지정한다. OUTPUT_VALUE는 제어출력으로 사용하는 비트 중 최하위비트로 설정한다. 제어출력으로 사용하는 포트의 비트가 변경되면 CONT_MASK와 OUTPUT_VALUE를 변경하면 된다. 7-세그먼트의 개수가 변경되면 CONT_MASK와 함께 N_SEGMENT 매크로도 함께 변경해야 한다.
- ☞ 7-세그먼트 코드 값 테이블은 파일 내의 다른 함수에서도 사용할 수 있도록 외부정적변수로 선언하였다.
- ☞ 헤더파일의 선행처리기 #ifndef - #endif에 관해서는 다음 10장에서 설명할 것이다.

실습 2 : [프로그램 prac9-1.c]는 부호없는 2바이트 정수 data를 7-세그먼트에 표시하는 프로그램이다. 수의 표시는 seg7Init()로 초기화한 후 표시를 할 수 있다. 7-세그먼트 동적 표시를 하므로 seg7.c와 seg7.h를 프로젝트 폴더에 위치시킨 후 프로젝트에 seg7.c를 포함시키고 빌드를 하여 실행파일을 얻는다. 실행파일을 수행하여 본다. 아울러 prac9-1.c를 7-세그먼트에 숫자를 16진수로 표시하도록 변경하여 본다.

과제

1. 그림 9.1에서 7-세그먼트에 출력하는 포트를 포트-A에서 포트-B로 변경하여 회로를 구성하였을 때 seg7.c에서 변경하여야 할 부분을 기술하라.
2. 소스파일 seg7.c의 함수 seg7DispNum()는 2바이트 부호없는 정수를 7-세그먼트에 표시한다. 여기서는 문자열을 받아 7-세그먼트에 표시하는 함수 seg7Dispstr()프로그램을 작성하라. 단 문자열이 4자 이상일 경우 앞 4 문자만 표시한다. 그리고 문자 '0'~'9', 'A'~'F', 'a'~'f'는 수로 표시하고 나머지 문자는 블랭크를 표시한다. 즉 문자열 "Ca#023"을 seg7Dispstr()함수에 넘기면 다음과 같이 표시한다.



작성한 프로그램으로 문자열 "3a", "Aa#2"와 "Ca#023"등 여러 문자를 표시하여 본다.

힌트: 문자로부터 수는 다음 방식으로 구할 수 있다.

$$'9' - '0' = 9, \quad 'F' - 'A' = 5, \quad 'd' - 'a' = 3$$

☞ C-언어에서 문자열(string)

- ▶ 문자열은 여러 문자의 집합을 나타낸다. C-언어에서 문자는 ' '를 사용하고 문자열은 " "를 사용하여 표시한다.

예) 문자 : 'A', '1'

문자열 : "Micom", "A", "1"

- ▶ 문자열은 문자의 집합이므로 char형의 배열에 저장된다. 그러나 일반 배열과는 달리 문자열의 마지막은 NULL 문자('\0')로 끝난다. (※ NULL 문자의 코드는 0이므로 '\0' = 0이다)

예) 문자열 "Micom"은 다음과 같이 저장된다.

'M'	'i'	'c'	'o'	'm'	'\0'				
-----	-----	-----	-----	-----	------	--	--	--	--

- ▶ 문자열의 마지막은 항상 NULL문자로 끝나므로 "Micom"를 표시하는 글자의 수는 6개이다. 적어도 6개의 char형 배열이 필요하다.
- ▶ 문자열의 끝은 NULL문자로 알 수 있으므로 문자열을 지정할 때는 문자들을 저장하고 있는 배열의 첫 주소로 나타낸다. 따라서 "Micom"는 이를 저장한 배열의 첫 주소를 나타낸다.

```

예) char ch;           // 문자
    char *pch;         // 문자열을 가리키는 포인터

    ch = 'A';          // ch에 'A'=0x41 저장
    pch = "Micom";     // pch에 "Micom"을 저장한 배열주소 저장
    ch = pch[0];       // pch[0]: "Micom"의 0-번째 글자 'M'
    ch = pch[1];       // pch[1]: "Micom"의 1-번째 글자 'i'

    ch = "A";          // 에러임 : "A"는 배열의 주소
                        // 즉 char *형이므로 char형인
                        // ch에 저장할 수 없음
  
```

[프로그램 prac9-1.c]

```

#include <avr/io.h>
#include "seg7.h"          // 7세그먼트 출력 프로그램 헤더파일

int main()
{
    unsigned short data=205;

    seg7Init();            // 7세그먼트 초기화

    while(1)
    {
        seg7DispNum(data, 10); // 10진수로 표시
    }
}

```

[소스파일 seg7.c]

```

//=====
// seg7.c : 4자리 7-세그먼트 동적 표시 프로그램
//=====
#include <avr/io.h>
#include <util/delay.h>

#include "seg7.h"          // 7-세그먼트 구동 헤더파일

#define N_SEGMENT      4          // 7-세그먼트의 개수
#define SSEG_CONT      PORTD      // 7세그먼트 제어 포트
#define DDR_CONT        DDRD
#define SSEG_DATA      PORTA      // 7세그먼트 데이터 포트
#define DDR_DATA        DDRA

#define CONT_MASK      ((1<<PD7)|(1<<PD6)|(1<<PD5)|(1<<PD4))
                        // 제어포트로 사용하는 포트 마스크
#define OUTPUT_VALUE (1<<PD4) // 제어포트 중 최하위 비트로 설정

```

```

//=====
// 수에 대한 7-세그먼트의 코드
// 마지막 코드 segTable[16]은 세그먼트를 끄는 데 사용
//=====
static unsigned char segTable[17] // 7-세그먼트 코드 값
    = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
        0x7F, 0x6F, 0x77, 0x7C, 0x58, 0x5E, 0x79, 0x71,
        0x00};

//-----
// cnumber[0] : 1자리수 저장
// cnumber[1] : 10자리수 저장
// . . .
//-----
static unsigned char cnumber[5]={16}; // 자리수별로 수를
                                         // 저장할 배열

//=====
// 7-세그먼트의 제어/데이터 포트 초기화
//=====
void Seg7Init()
{
    SSEG_CONT |= 0xF0; // 모든 7-세그먼트를 끈다.
    SSEG_DATA  = 0xFF; //

    DDR_CONT  |= 0xF0; // 제어포트의 상위니블을 출력으로 설정
    DDR_DATA   = 0xFF; // 데이터포트는 모두 출력으로 설정
}

//=====
// 2바이트 부호없는 정수를 받아 주어진 진법으로 4자리
// 7-세그먼트에 동적으로 표시하는 함수
//
// 입력 : num    - 7-세그먼트에 표시할 수
//        radix  - 사용 진법
//=====
void Seg7DispNum(unsigned short num, unsigned short radix)
{
    int j;

```



```

//-----
// cnumber[]에 수 0저장
// cnumber[0] = 0
// cnumber[1] ~ cnumber[4] = 16: 세그먼트를 끈다.
//-----
cnumber[0] = 0;
for(j=1; j<5; j++) cnumber[j] = 16;

j=0;                // 1자리를 가리킴
while(num)
{
    cnumber[j++] = num % radix; //나머지
    num /= radix;              // 몫
}

for(j=0; j<N_SEGMENT; j++)    // 하위자리부터 표시한다.
{
    // j-자리를 켜다.
    SSEG_CONT = ( SSEG_CONT & ~CONT_MASK) |
                (~(OUTPUT_VALUE<<j) & CONT_MASK) ;
    SSEG_DATA = ~SegTable[cnumber[j]];
    _delay_ms(5);    // 5msec delay
}
}

```

[헤더파일 **seg7.h**]

```

//=====
// seg7.h : 7-세그먼트 구동 프로그램의 헤더파일
//=====
#ifndef __SEG7_H__
#define __SEG7_H__

void Seg7Init(void);
void Seg7DispNum(unsigned short num,unsigned short radix);

#endif

```

9.5 동적 표시 프로그램 II - 타이머/카운터 인터럽트 사용

[프로그램 prac9-1.c]를 수정하여 다음 동작을 하도록 프로그램 9.4를 작성하여 보자.

(1) data의 초기 값은 0으로 한다.

(2) 1초마다 변수 data를 10씩 증가시켜 7-세그먼트에 표시한다.

```
#include <util/delay.h>
#include "seg7.h"          // 7세그먼트 동적 표시 헤더파일
void msec_delay(int n);   // 시간 지연 함수
int main()
{
    unsigned short data=0;
    Seg7Init();           // 7세그먼트 초기화
    while(1)
    {
        msec_delay(1000); // 1초를 기다린다.
        data += 10;       // data를 10 증가한다.
        Seg7DispNum(data, 10); // 10진수로 표시
    }
}
void msec_delay(int n)    // 시간지연함수
{
    for(; n >0; n--)      // 1msec 시간지연을 n회 반복
        _delay_ms(1);    // 1msec 시간지연
}
```

프로그램 9.4

실습 1 : 프로그램 9.4를 실행하고 동작을 관찰한다.

프로그램 9.4를 수행하면 1초 마다 갱신된 변수 data를 잠시 표시하고 사라진다. 수를 7-세그먼트에 동적으로 표시하기 위해서는 잠시도 쉬지 않고 함수 Seg7DispNum()을 호출하여야 하나, 1초를 기다림으로서 그 동안 화면 표시가 되지 않기 때문이다.

9.2절에서 작성한 동적 표시 프로그램 `seg7.c`는 `main()`함수에서 계속적으로 7-세그먼트에 표시를 해야 하므로 장시간의 시간지연을 용납하지 않는다. 이 때문에 `main()`함수에서 시간지연을 줄 필요가 있을 때는 `seg7.c`를 사용할 수 없다. 해결책으로 7-세그먼트에 수를 동적으로 표시하는 일을 `main()`함수가 아닌 인터럽트 서비스루틴이 처리하도록 하는 방법을 고려할 수 있다.

- ☞ 인터럽트 서비스루틴은 `main()`함수와는 독립적으로 동작하기 때문에 주기적으로 7-세그먼트에 데이터를 쓰는 동작을 인터럽트 서비스루틴에 할당을 하면 `main()`함수에서의 시간지연이 7-세그먼트의 동적표시에 영향을 미치지 못한다.
- ☞ 프로그램 9.3으로부터 매끄러운 동적표시를 위해서는 5msec마다 7-세그먼트에 데이터를 내보내야 한다. 이를 위해 타이머/카운터 0의 CTC모드를 사용하여 5msec마다 비교 일치 인터럽트를 걸게 한다.(8.4.1절 (다) 참조)
- ☞ 함수 `seg7DispNum()`에서 7-세그먼트에 수를 표시하였던 부분을 비교 일치 인터럽트 서비스루틴이 처리하도록 한다.

그림 9.4는 인터럽트 서비스루틴을 사용한 동적 표시 방법의 개념도를 나타낸다. 인터럽트를 사용한 동적표시프로그램의 소스파일은 `iseg7.c`에 헤더파일은 `iseg7.h`에 작성하고 함수의 접두어는 `Iseg7`를 사용한다. 여기서 'I'는 인터럽트를 사용한 동적표시라는 의미이다.

- ☞ 초기화 함수 `Iseg7Init()`에서는 타이머/카운터 0이 CTC모드로 동작하도록 하고 5msec주기로 비교일치 인터럽트를 걸도록 설정한다.
- ☞ 그림 9.4에서 `main()`함수가 함수 `Iseg7DispNum()`를 호출하여 415를 10진법으로 표시하려면 이 데이터가 `Iseg7DispNum()`함수의 자동변수인 `num`과 `radix`로 각각 복사된다.
- ☞ 함수 `Iseg7DispNum()`는 2바이트 변수인 `num`을 요구된 진법 `radix`에 따라 자리수별로 구별하여 `cnumber[]`에 저장한다. 그림 9.4에서는 십진수 415를 십진법으로 표시한다. 저장하는 동안 타이머/카운터

0의 비교일치 인터럽트를 금지한다.

- ☞ 함수 `Seg7DispNum()`과는 달리 `Iseg7DispNum()`은 7-세그먼트에 값을 표시하지 않는다. 저장된 `cnumber[]`를 7-세그먼트에 표시하는 작업은 비교일치 인터럽트 서비스 루틴으로 이양되었다.
- ☞ 배열 `cnumber[]`는 정적외부변수로 설정되어 있으므로 `iseg7.c`내의 모든 함수는 `cnumber[]`의 값을 액세스할 수 있다. 따라서 함수 `Iseg7DispNum()`에서 설정한 `cnumber[]`를 인터럽트 서비스루틴에서도 읽을 수 있다.
- ☞ 비교일치 인터럽트 서비스루틴은 `main()`함수와는 독립적으로 동작하며 초기화 함수인 `Iseg7Init()`의 설정에 따라 5msec마다 수행된다.
- ☞ 인터럽트 서비스루틴의 내부정적변수 `index`는 현재 표시하여야 할 자리수를 나타낸다. 인터럽트 서비스루틴이 한번 수행될 때마다 이 변수는 한자리씩 이동하게 된다.
- ☞ 현재 표시하여야 할 수는 `cnumber[index]`이고 이를 표시하기 위해 7-세그먼트에 출력하여야 할 데이터는 `SegTable[cnumber[index]]`이다. 인터럽트 서비스루틴은 5msec마다 변수 `index`를 이동하면서 `index`-번째 7-세그먼트에 데이터 `SegTable[cnumber[index]]`를 출력하여 동적표시를 작업을 독립적으로 수행한다.

프로그램 9.4를 인터럽트를 사용한 동적 표시 프로그램인 `iseg7.c`를 사용하여 새로 작성하면 프로그램 9.5와 같다.

- ☞ `iseg.c`는 인터럽트를 사용하므로 초기화후 함수 `sei()`를 호출하여 전역인터럽트를 허용하여야 한다.

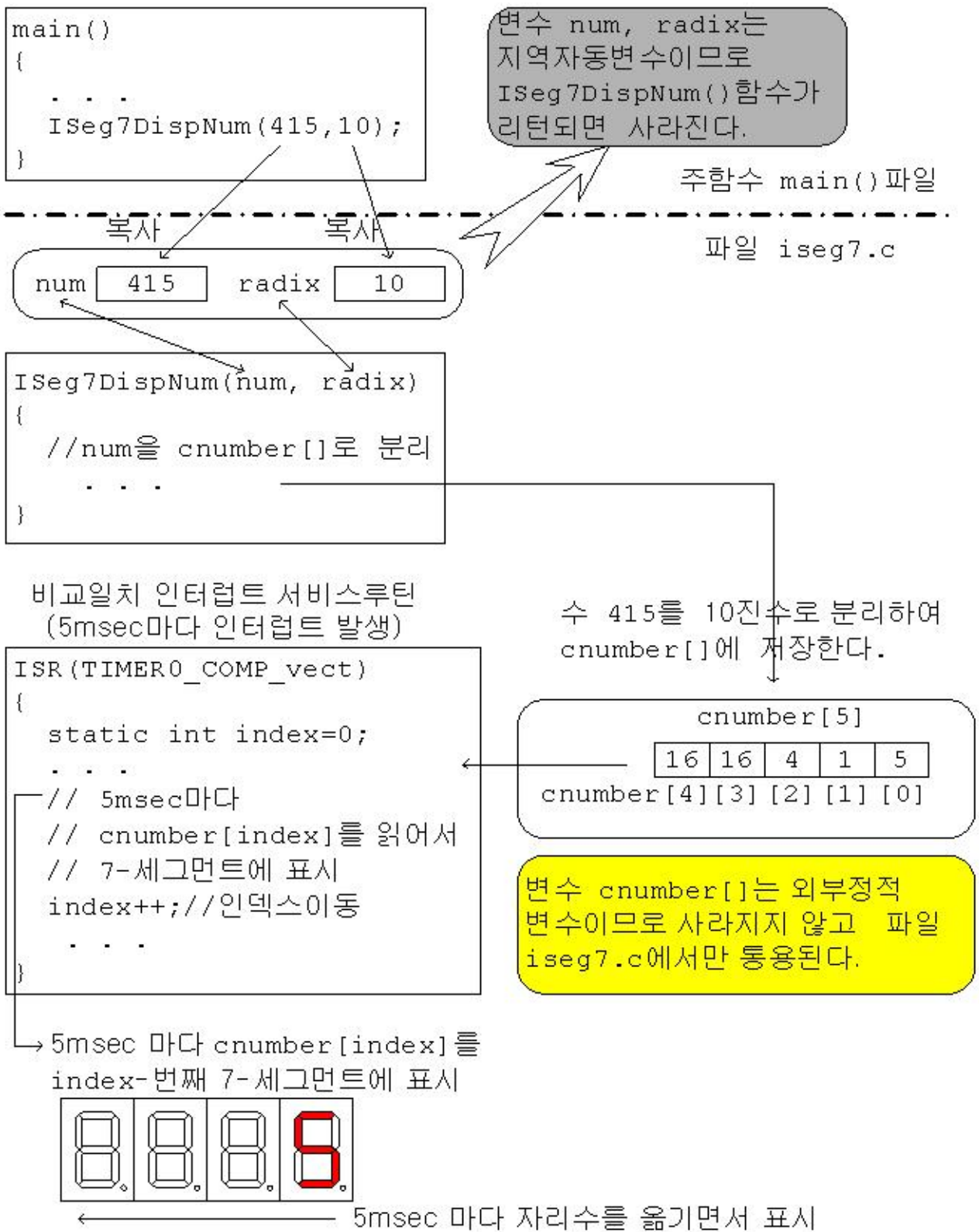


그림 9.4 인터럽트를 사용한 동적 표시

```

#include <avr/interrupt.h>
#include <util/delay.h>
#include "iseg7.h"          // 인터럽트를 사용한 7세그먼트
                             // 동적 표시 프로그램 헤더파일
void msec_delay(int n); // 시간지연함수
int main()
{
    unsigned short data=0;

    Iseg7Init();           // 인터럽트를 사용한
                             // 7세그먼트 동적표시 초기화
    sei();                 // 7-세그먼트 표시를 위해
                             // 인터럽트를 허용한다.

    while(1)
    {
        msec_delay(1000);    // 1초를 기다린다.
        data += 10;          // data를 10 증가한다.
        Iseg7DispNum(data, 10); // 10진수로 표시
    }
}

void msec_delay(int n)
{
    for(; n >0; n--)          // 1msec 시간지연을 n회 반복
        _delay_ms(1);        // 1msec 시간지연
}

```

프로그램 9.5

실습 : 프로그램 9.5를 수행하고 동작을 관찰하라.

과제 : 인터럽트를 사용하는 동적 표시 프로그램에서 9.4절 과제에서 작성한 seg7DispStr()와 유사한 동작을 하는 Iseg7DispStr()함수를 추가하고 처음 1초는 Iseg7DispNum()함수를 사용하여 숫자 123을 표시하고 다음 1초는 Iseg7DispStr()함수를 사용하여 "Ca#023"을 표시하고 이를 반복하는 프로그램을 작성하라.

[소스파일 iseg7.c]

```
//=====
// iseg7.c : 인터럽트를 사용한 4자리
//          7-세그먼트 동적 표시 프로그램
//=====
#include <avr/io.h>
#include <avr/interrupt.h>

#include "iseg7.h"          // 7-세그먼트 구동 헤더파일

#define N_SEGMENT 4        // 7-세그먼트의 개수

#define SSEG_CONT PORTD    // 7세그먼트 제어 포트
#define DDR_CONT  DDRD
#define SSEG_DATA PORTA    // 7세그먼트 데이터 포트
#define DDR_DATA  DDRA

#define CONT_MASK ((1<<PD7)|(1<<PD6)|(1<<PD5)|(1<<PD4))
// 제어포트로 사용하는 포트 마스크
#define OUTPUT_VALUE (1<<PD4)// 제어포트 중 최하위 비트로 설정

//=====
// 수에 대한 7-세그먼트의 코드
// 마지막 코드 segTable[16]은 세그먼트를 끄는 데 사용
//=====
static unsigned char segTable[17] // 7-세그먼트 코드 값
    = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
        0x7F, 0x6F, 0x77, 0x7C, 0x58, 0x5E, 0x79, 0x71,
        0x00};

//-----
// cnumber[0] : 1자리수 저장
// cnumber[1] : 10자리수 저장
// . . .
//-----
static unsigned char cnumber[5] // 자리수별로 수를 저장할 배열
    = {16,16,16,16,16};      // 초기에 모두 끝다.
```

```

//=====
// 7-세그먼트의 제어/데이터 포트 초기화 및 타이머/카운터 초기화
//=====
void Iseg7Init()
{
    SSEG_CONT = 0xFF; // 모든 7-세그먼트를 끈다.
    SSEG_DATA = 0xFF; //

    DDR_CONT |= 0xF0; // 제어포트의 상위니블을 출력으로 설정
    DDR_DATA = 0xFF; // 데이터포트는 모두 출력으로 설정

    // 타이머/카운터0 비교 일치 인터럽트 설정
    // 타이머/카운터0는 CTC모드로 동작
    TCCR0 = (1<<WGM01); // CTC모드/타이머 정지, oc0핀 미사용
    OCR0 = 78; // 5msec마다 비교일치 인터럽트 발생
    TIMSK |= (1<<OCIE0); // 타이머0 비교일치 인터럽트 허용

    TCCR0 |= 0x07; // 분주비 1024로 타이머 시작
}

//=====
// 2바이트 부호없는 정수를 받아 주어진 진법으로 4자리
// 7-세그먼트에 동적으로 표시하는 함수
// (이 함수는 각 세그먼트에 표시할 숫자를 cnumber[]배열에
// 저장만 한다. 실제 표시는 인터럽트 서비스루틴에서 수행한다.)
// 입력 : num - 7-세그먼트에 표시할 수
//         radix - 사용 진법
//=====
void Iseg7DispNum(unsigned short num,
                  unsigned short radix)
{
    int j;

    //-----
    // cnumber[]에 수 0저장
    // cnumber[0] = 0
    // cnumber[1] ~ cnumber[4] = 16: 세그먼트를 끈다.
    //-----

```



```

TIMSK &= ~(1<<OCIE0);    // 타이머0 비교일치 인터럽트 금지
cnumber[0] = 0;
for(j=1; j<5; j++) cnumber[j] = 16;

j=0;                      // 1자리를 가리킴
while(num)
{
    cnumber[j++] = num % radix; //나머지
    num /= radix;              // 몫
}
TIMSK |= (1<<OCIE0);      // 타이머0 비교일치 인터럽트 허용
}

//=====
// 인터럽트 서비스 루틴
//
// 인터럽트가 걸릴 때마다 index가 가리키는 세그먼트에 데이터를
// 표시하고, index는 계속 회전이동하게 된다.
//=====
//
ISR(TIMER0_COMP_vect)    // 카운터/타이머0 비교일치 인터럽트
{
    static int index = 0;

    // 현재 index가 가리키는 자리수를 표시한다.

    SSEG_CONT = (SSEG_CONT & ~CONT_MASK) |
                (~(OUTPUT_VALUE<<index) & CONT_MASK);
    SSEG_DATA = ~SegTable[cnumber[index]];

    index++;          // 인덱스 이동
    if(index == N_SEGMENT) index = 0;
}

```

[헤더파일 iseg7.h]

```
//=====
// iseg7.h : 인터럽트를 사용한 7-세그먼트
//          동적 표시 프로그램의 헤더파일
//=====
//
#ifndef __ISEG7_H__
#define __ISEG7_H__

void Iseg7Init(void);
void Iseg7DispNum(unsigned short num,
                  unsigned short radix);
#endif
```