【 기술 노트 17 】

마이크로콘트롤러용의 C컴파일러에서 printf() 함수의 사용

오늘날 C언어는 마이크로콘트롤러에서도 가장 널리 사용되는 프로그래밍 언어가 되었다. 그러나, 적지 않은 사용자들은 단지 퍼스널컴퓨터에서 배운 범용 C언어 프로그래밍 경험만으로 마이크로콘트롤러에서도 별다른 새로운 노력 없이 C언어로 프로그래밍하겠다고 달려드는 것을 볼 수 있다. 이보다 심한 경우는 8051용의 C소스 프로그램을 Turbo C로 컴파일하려는 넌센스 같은 일도 있다. 이러한 사례들은 참으로 어리석은 생각들이다.

C언어는 매우 이식성이 우수한 언어로 알려져 있다. 하지만 C언어의 이러한 장점은 이를 마이크로콘트롤러에 적용하는 경우에는 고스란히 단점이 된다. C언어가 이식성이 우수하다는 것은 뒤집어 보면 하드웨어 의존적인 기능이 매우 취약하다는 것인데, 마이크로콘트롤러는 이름 그대로 하드웨어적인 콘트롤러로 사용하려는 목적의 CPU이므로 그 속성상 매우 하드웨어 의존적인 사항이 많고, 따라서 C언어를 마이크로콘트롤러에서 사용하려면 표준적인 C언어 기능만으로는 처리할 수 없는 일들이 많다는 이야기이기 때문이다.

뿐만 아니라 마이크로콘트롤러에서는 그 구조와 기능, 응용 분야가 천차만별로 다르고 표준적인 하드웨어 시스템 구성을 생각할 수 없기 때문에 범용 C언어에서 사용하는 파일처리 기능이나 모니터 출력 기능, 인터럽트 처리 기능 등과 같이 특정 하드웨어에 관련되는 기능 및 운영체제와 연계되는 기능은 여기에 그대로 적용하기 어렵다.

이러한 이유로 하여 마이크로콘트롤러용의 C컴파일러는 가능하면 표준 ANSI C 언어의 문법을 따르지만, ANSI C언어의 기능 중에서 상당히 많은 내용이 필요 없어서 삭제되었으며, 반대로 하드웨어적인 처리기능은 많은 내용들이 새로 추가되었다. 따라서, 마이크로콘트롤러용의 C컴파일러를 사용하려는 사람은 당연히 이러한 새로운 기능들을 추가로 익혀야만 한다. 하지만, 이렇게 추가되는 내용은 표준의 ANSI C언어와는 다르기 때문에 C컴파일러를 개발하는 각 소프트웨어 회사별로 상당히 다른 방법을 사용하게 되었고, 결국 이 소프트웨어의 사용자들은 각 제품의 특징과 기능을 세밀하게 익히지 않으면 이를 올바르게 사용할 수 없게 된다. 결과적으로 C언어는 어디서나 거의 유사하다는 믿음은 적어도 마이크로콘트롤러 분야에 이르면 전혀 잘못된 내용이 될 수밖에 없다는 것이다.

그러나, 범용 C언어의 기능 중에는 이를 적절히 변형하여 마이크로콘트롤러에서 아주 유용하게 사용할 수 있도록 해 놓은 것도 있다. printf() 함수가 바로 그러한 예 중의 하나이다. 한 번 생각해보자. C언어를 사용해본 사람이라면 아마도 누구나 가장 먼저 배우는 기능의 하나가 바로 이 printf() 함수일 것이다. 범용 C 언어에서 이 함수는 가장 기본적인 기능중의 하나이며 그 실행 결과가 콘솔 즉 퍼스널컴퓨터의 경우는 모니터로 출력되도록 정의되어 있다. 그러면, 마이크로콘트롤러용의 C컴파일러에서는 이를 어떻게 처리하였을까? 여기서는 모니터라는 주변장치가 없으니 이 함수를 삭제하였을까? 아니면 C언어에서 너무나

도 기본적인 함수이므로 그냥 살려두었을까? 그냥 두었다면 이를 실행할 때 출력은 과연 어디로 나갈까?

대부분의 마이크로콘트롤러용 C컴파일러에서는 이 printf() 함수를 지원한다. 그 출력은 디폴트로 표준의 비동기 직렬통신 포트로 출력되도록 설정하여 놓았으며, 만약 필요하면 사용자가 이를 변경할 수 있도록 하는 기능을 가지고 있다. 예를 들면 printf() 함수의 출력을 LCD 모듈로 변경 지정하여 여기에 %를 이용한 서식 지정의 데이터를 편리하게 출력(formatted output)할 수도 있는 것이다.

여기서는 8051과 80C196KC용의 C컴파일러에서 각각 이와 같이 편리한 printf() 함수를 올바르게 사용하는 방법을 알아보기로 한다.

1. 8051 MCU용의 Keil C컴파일러 C51에서 printf() 함수의 올바른 사용

Keil사의 MCS-51 C컴파일러에서는 printf() 함수가 표준적인 %서식 지정 기능을 모두 지원하며, 이것의 실행 결과가 디폴트로 8051에 내장된 직렬 포트를 통하여 출력된다. Keil C컴파일러의 매뉴얼에는 이 함수를 사용하는 방법과 출력이 다른 장치로 이루어지도록 이 함수를 사용자가 수정하여 사용하는 방법에 대하여도 비교적 자세히 취급하고 있다.

이러한 문제에 대하여는 참고문헌 1과 2에서 자세히 설명하고 있으며, 아래는 이 책들에 있는 내용을 요약한 것이다. 여기서 *OK-8051* 키트의 사용에 관해서는 취급하지 않으니 필요하면 참고문헌 1, 2를 찾아보기 바란다.

(1) 저수준 스트림 I/O의 Customization

Keil사의 C컴파일러 C51에서는 사용자가 몇가지의 소스 파일을 자신의 하드웨어 시스템에 적합하도록 수정(customization)하여 사용해야 한다. 이러한 파일에는 C언어 프로그램이 올바르게 기동할 수 있도록 해주는 STARTUP.A51 파일, 정적 변수를 초기화하는데 사용되는 INIT.A51 파일, 저수준의 스트림 I/O를 수행하는 GETKEY.C 및 PUTCHAR.C 파일 등이 있으며, 기타 메모리 할당 기능에 관련되는 CALLOC.C, FREE.C, INIT_MEM.C, MALLOC.C, REALLOC.C 파일 등이 있다.

이러한 함수의 소스 파일들은 이미 어셈블 또는 컴파일되어 기본 소프트웨어 팩키지의 런타임 라이브러리(runtime library) 파일에 포함되어 있으므로 사용자 프로그램을 링크할 때 자동으로 이것들이 링크된다. 그러나, 만약 사용자가 이를 수정하여 번역한 오브젝트 파일들을 링크시키려면 BL51 링커를 사용할 때 이들 오브젝트 파일의 이름을 지정하여 링크하면 된다.

여기서 우리가 관심을 갖는 것은 저수준의 스트림 출력(low-level stream output)을 수행하는 PUTCHAR.C 파일이다. 이 PUTCHAR.C 파일은 내부에 저수준의 문자 출력 루틴인 putchar() 함수를 포함하고 있다. printf, vprintf 등과 같이 수치 데이터를 서식지정 방식

으로 출력하는 모든 스트림 루틴들은 내부적으로 putchar() 함수를 사용하고 있으므로, 사용자는 자신의 하드웨어에 맞게 이 putchar() 함수만을 수정하면 이들 나머지의 함수들이 모두 자동으로 올바르게 수행되는 것이다.

따라서, printf() 함수를 사용하여 직렬 포트로 %서식 지정 문자를 출력하려면 소스 프로그램을 작성할 때 그냥 printf() 함수를 사용하기만 하면 된다. 그러나, 만약 이를 다른 장치로 출력하도록 수정하려면 putchar() 함수가 정의된 PUTCHAR.C 파일을 수정하고 이를 컴파일하여 얻어진 PUTCHAR.OBJ 파일을 사용자 프로그램의 오브젝트 파일과 함께 링크해 주어야 한다.

C51 소프트웨어의 기본적인 런타임 라이브러리에 제공되는 putchar() 함수는 디폴트로 8051 마이크로콘트롤러의 직렬 포트를 통하여 문자를 출력하도록 프로그램되어 있다. 이 출력제어에서는 XON/XOFF 제어를 사용하며, 라인피드 문자('\n')는 캐리지리턴/라인피드 문자열('\r\n')로 자동 변환된다. 따라서, 사용자가 이를 만약 LCD 모듈로 출력하도록 지정하려면 이에 맞게 PUTCHAR.C 파일에 들어있는 putchar() 함수를 수정하여야 한다.

◀참고▶ 이와 같이 putchar() 함수를 이용한 저수준의 스트림 출력 기능을 사용할 때 한가지 주의해야 할 것은 이들 루틴의 어디에도 하드웨어에 대한 초기화 기능이 없다는 것이다. 따라서, putchar() 함수를 이용하여 직렬 포트나 텍스트형 LCD 모듈에 출력을 수행하기 전에 이들 하드웨어는 별도의 초기화 루틴을 사용하여 적절히 초기화해 두어야 한다.

(2) printf() 함수의 기능

여기서는 putchar() 함수를 텍스트형 LCD 모듈에 맞도록 수정하여 사용하는 예를 보여주기 위하여 먼저 C51에서 printf() 함수의 기능을 자세히 설명한다. 이러한 스트림 I/O에 관련된 함수들은 모두 헤더파일 STDIO.H에 정의되어 있으며, 그 기능은 다음과 같다.

```
int printf(const char *formatstring, argument, ...)
```

여기서 formatstring은 출력할 문자열의 포맷을 지정하는 양식으로서 다음과 같은 형식으로 사용한다. 여기서 각 필드는 1개의 문자 또는 숫자로 구성된다.

```
%[flag][[0]width.precision]{b|B|l|L}type
```

여기서, type은 이 함수의 argument가 문자, 문자열, 수, 포인터 중에 어느 것으로 해석될지를 지정하는 문자이다. 이 type에 사용할 수 있는 문자와 그 의미를 요약하면 <표 1>과 같다.

선택적으로 사용하는 문자 b, B, l, L은 int형인 d, i, u, o, x, X를 char 또는 long형으로 지정하는데 사용한다.

또한, width는 표시되는 전체 데이터의 폭을 나타내는 정수로서 출력값이 width보다 짧으면 공백이나 0이 채워지고, 출력값이 width보다 길면 width를 무시하고 출력값이 모두 표시된다. 선택사항으로 width의 앞에 0이 붙으면 빈자리가 공백이 아닌 0으로 채워진다.

precision은 전체 데이터의 폭 중에서 소수부의 자리수를 지정하는 필드이다. 표시되는 자리의 마지막은 그 아래 자리에서 반올림된 것이다. 이를 0으로 지정하면 소수점도 표시하지 않는다.

<표 1> %서식 지정에서 type으로 사용되는 문자

type 문자	argument의 형	출력 포맷
d 또는 i u	int unsigned int	부호있는 10진수. 부호없는 10진수.
o	unsigned int	부호없는 8진수.
x X	unsigned int unsigned int	부호없는 16진수, 소문자 사용(0123456789abcdef). 부호없는 16진수, 대문자 사용(0123456789ABCDEF).
f	float	[-]dddd.dddd 형식의 부동소수점 수.
e E	float float	[-]d.dddde[-]dd 형식의 부동소수점 수. [-]d.ddddE[-]dd 형식의 부동소수점 수.
g G	float float	수치의 크기에 따라 f형식 또는 e형식으로 표시되는 부동소수점 수. 수치의 크기에 따라 f형식 또는 E형식으로 표시되는 부동소수점 수.
c	char	1개의 문자.
s	generic	null 문자로 끝나는 문자열.
p	generic	t:aaaa 형식으로 표현되는 포인터.(여기서 aaaa는 16진수 어드레스이며, t는 c:code, i:data/idata, x:xdata, p:pdata를 나타낸다.)

flag는 출력문자들을 정렬하거나 +, - 기호, 공백, 소수점 등을 표시하는 방법을 지정하는데 사용하는 필드이다. 이 flag에 사용할 수 있는 문자와 그 의미를 요약하면 <표 2>와 같다.

<표 2> flag으로 사용되는 문자

flag 문자	의 미
-	필드를 좌측으로 정렬하여 출력한다.
+	수치의 앞에 반드시 + 또는 -를 붙인다.
공백	양수의 앞에 공백문자를 붙인다.
#	type이 o, x, X인 경우에 각각 숫자의 앞에 0, 0x, 0X를 붙인다. type이 f, e, E, g, G인 경우에는 반드시 숫자에 소수점을 붙여 표시한다.
*	출력 포맷 지정을 무시한다.

(3) putchar() 함수의 수정

C51에서 printf() 함수를 텍스트형 LCD 모듈에 맞도록 사용하려면 먼저 C:\C51\LIB 디렉토리에 기본적으로 제공되는 PUTCHAR.C 파일의 putchar() 함수를 수정하여야 한다. C51에 기본적으로 제공되는 오리지널 PUTCHAR.C 파일은 <파일 PUTCHAR.C>와 같다. 내용을 보면 알 수 있듯이 이는 직렬 포트를 사용하는 것으로 되어 있다.

<파일 PUTCHAR.C> C51에 기본적으로 제공되는 오리지널 putchar() 함수 파일

```
/***********************************************************************/
/*  This file is part of the C51 Compiler package                      */
/*  Copyright KEIL ELEKTRONIK GmbH 1990                                */
/***********************************************************************/
/*                                                                     */
/*  PUTCHAR.C:  This routine is the general character output of C51.   */
/*  To translate this file use C51 with the following invocation:      */
/*                                                                     */
/*     C51 PUTCHAR.C <memory model>                                    */
/*                                                                     */
/*  To link the modified PUTCHAR.OBJ file to your application use the  */
/*  following L51 invocation:                                          */
/*                                                                     */
/*     L51 <your object file list>, PUTCHAR.OBJ <controls>             */
/*                                                                     */
/***********************************************************************/

#include <reg51.h>

#define XON  0x11
#define XOFF 0x13

char putchar (char c)  {
  if (c == '\n')  {
    if (RI)  {
      if (SBUF == XOFF)  {
        do  {
          RI = 0;
          while (!RI);
        }
        while (SBUF != XON);
        RI = 0;
      }
    }
    while (!TI);
    TI = 0;
    SBUF = 0x0d;                        /* output CR  */
  }
  if (RI)  {
    if (SBUF == XOFF)  {
      do  {
        RI = 0;
        while (!RI);
      }
      while (SBUF != XON);
      RI = 0;
    }
  }
  while (!TI);
  TI = 0;
  return (SBUF = c);
}
```

여기서는 이를 직접 수정하지 않고 원본을 보존하여 두기 위하여 별도의 파일로 작성하기로 하는데 그 내용은 아래의 <파일 PUTCHAR1.C>와 같다. 당연히 이 파일은 C:\C51\LIB 디렉토리에서 작성하며, 이를 컴파일하여 얻은 오브젝트 파일 PUTCHAR.OBJ도 물론 이 디렉토리에 위치해야 한다.

<파일 PUTCHAR1.C> 1문자를 LCD 모듈에 출력하도록 수정한 putchar() 함수 파일

```
/*************************************************************************/
/*                                                                       */
/*  PUTCHAR1.C:  This routine is the character output to LCD module.     */
/*                                                                       */
/*************************************************************************/

#include <REG52.H>
#include "OK8051.H"

char putchar(char c)
{ unsigned int i;

  if(c == '\n')                       /* if CR, return */
    return(c);

  for(i = 0; i < 2000; i++)           /* check busy flag */
    { if((LCD_RD_STATUS & 0x80) == 0)
        break;
    }

  LCD_WR_DATA = c;                    /* output c */
  return(c);
}
```

☞ 헤더 파일 STDIO.H에서 보면 이 함수는 char putchar(char)의 형으로 되어 있으므로 여기서도 이 형식에 맞게 프로그램을 작성해야 한다.

텍스트형 LCD 모듈에 1문자를 출력하기 전에 busy flag을 체크한다. 그러나, 이를 단순한 무한 루프로 처리하면 텍스트형 LCD 모듈이 *OK-8051* 키트에서 제거되거나 또는 LCD 모듈의 고장으로 인하여 올바른 동작을 하지 않을 경우에 프로그램이 무한 루프에 빠져 홀딩될 수 있으므로 이를 방지하기 위하여 2000회 이내로 체크 횟수를 제한하였다.

이 putchar() 함수가 사용자 프로그램에서 올바르게 사용되려면 이 PUTCHAR1.C 파일을 컴파일하여 미리 오브젝트 파일 PUTCHAR.OBJ를 생성해 놓아야 한다. 이는 아래와 같이 수행한다.

```
C:\C51\LIB>C51 PUTCHAR1.C OJ(PUTCHAR.OBJ) Enter↵
```

(4) 수정된 printf() 함수의 사용

그러면 이제 사용자 프로그램에서 이렇게 수정된 printf() 함수를 사용하기 위한 준비는 모두 끝났다. 실제로 이를 수행해 보기 위하여 간단히 16비트 정수를 10진수와 16진수로 표시하는 예제를 printf() 함수를 사용한 %서식 지정으로 구현해 보기로 한다.

이렇게 작성한 예제는 아래의 <프로그램 TEST51A.C>와 같다. 이를 printf() 함수를 사용하지 않고 일반적인 방법으로 LCD 모듈에 정수 데이터를 출력하는 프로그램으로 작성하여 보면 아래의 <프로그램 TEST51AX.C>와 같다. printf() 함수를 사용하는 경우 프로그램

이 훨씬 더 간단하고 편리해진 것을 느낄 수 있지 않은가?

<프로그램 TEST51A.C> printf() 함수를 사용하여 LCD 모듈에 정수를 표시하는 프로그램

```
/* TEST51A.C */
/* ========================================================================== */
/*      Integer Number Display on Text LCD Module by printf()                 */
/* ========================================================================== */
/*                                    Programmed by Duck-Yong Yoon in 2002.   */

#include <REG52.H>
#include <STDIO.H>
#include "OK8051.H"
#include "OK8051FN.H"

main()
{ unsigned int i;

  LCD_string(0x80," Integer Number ");          /* display title */
  LCD_string(0xC0," 00000 = 0000H  ");
  Beep();

  i = 1;                                        /* initial integer number */

  while(i != 0)
    { LCD_command(0xC1);                        /* display decimal number */
      printf("%5d",i);
      LCD_command(0xC9);                        /* display hexadecimal number */
      printf("%04X",i);
      Delay(200);                               /* delay 200 ms */
      i++;
    }
}
```

☞ 이 예제에서는 %서식 지정으로 10진수에서 최초로 나타나는 0을 공백문자로 표시하고, 16진수에서는 최초로 0이 나타나더라도 이를 공백 문자로 처리하지 않고 그냥 0으로 표시하도록 하였다.

<프로그램 TEST51AX.C> 사용자 전용 함수를 사용하여 LCD 모듈에 정수를 표시하는 프로그램

```
/* TEST51AX.C */
/* ========================================================================== */
/*              Integer Number Display on Text LCD Module                     */
/* ========================================================================== */
/*                                    Programmed by Duck-Yong Yoon in 2002.   */

#include <REG52.H>
#include "OK8051.H"
#include "OK8051FN.H"

void LCD_4hex(unsigned int number)      /* display HEX number xxxxH */
{ unsigned int i;

  i = number >> 12;                             /* 16^3 */
  if(i <= 9) LCD_data(i + '0');
  else       LCD_data(i - 10 + 'A');

  i = (number >> 8) & 0x000F;                   /* 16^2 */
```

```
    if(i <= 9) LCD_data(i + '0');
    else       LCD_data(i - 10 + 'A');

    i = (number >> 4) & 0x000F;                    /* 16^1 */
    if(i <= 9) LCD_data(i + '0');
    else       LCD_data(i - 10 + 'A');

    i = number & 0x000F;                           /* 16^0 */
    if(i <= 9) LCD_data(i + '0');
    else       LCD_data(i - 10 + 'A');
  }

  void LCD_5d(unsigned int number)        /* display 5-digit decimal number */
  { unsigned int i;
    unsigned char flag;

    flag = 0;
    i = number/10000;                              /* 10^4 */
    if(i == 0) LCD_data(' ');
    else {     LCD_data(i + '0');
               flag = 1;
         }

    number = number % 10000;                       /* 10^3 */
    i = number/1000;
    if((i == 0) && (flag == 0))
               LCD_data(' ');
    else {     LCD_data(i + '0');
               flag = 1;
         }

    number = number % 1000;                        /* 10^2 */
    i = number/100;
    if((i == 0) && (flag == 0))
               LCD_data(' ');
    else {     LCD_data(i + '0');
               flag = 1;
         }

    number = number % 100;                         /* 10^1 */
    i = number/10;
    if((i == 0) && (flag == 0))
               LCD_data(' ');
    else       LCD_data(i + '0');

    i = number % 10;                               /* 10^0 */
    LCD_data(i + '0');
  }

  main()
  { unsigned int i;

    LCD_string(0x80," Integer Number ");           /* display title */
    LCD_string(0xC0," 00000 = 0000H  ");
    Beep();

    i = 1;                                         /* initial integer number */

    while(i != 0)
      { LCD_command(0xC1);                         /* display decimal number */
        LCD_5d(i);
        LCD_command(0xC9);                         /* display hexadecimal number */
        LCD_4hex(i);
        Delay(200);                                /* delay 200 ms */
        i++;
      }
  }
```

이제 이를 컴파일하여 링크할 때는 C:\C51\LIB 디렉토리에 우리가 미리 생성해 두었던 PUTCHAR.OBJ 파일을 함께 링크해야 한다. 이 과정은 좀 복잡하므로 아래와 같이 일련적으로 처리할 수 있는 배치 파일로 작성하기로 한다.

<파일 BAT51.BAT> printf() 함수를 사용한 프로그램을 번역 및 링크하기 위한 배치 파일

```
C51 %1.C CODE
if errorlevel 2 goto END
if errorlevel 3 goto END
BL51 C:\C51\LIB\STARTUP.OBJ,%1.OBJ,C:\C51\LIB\PUTCHAR.OBJ TO %1.OUT CODE(8000H)
 DATA(50H) XDATA(0E000H) IXREF
echo off
if exist %1.OBJ DEL %1.OBJ
OH51 %1.OUT
if exist %1.OUT DEL %1.OUT
:END
```

◀참고▶ 이와 같이 사용자가 BL51을 사용하여 사용자 프로그램의 오브젝트 파일이나 라이브러리 파일들을 링크할 때 PUTCHAR.OBJ 파일을 별도로 링크하여 주면 BL51 링커는 런타임 라이브러리에 기본적으로 내장되어 있던 putchar() 함수를 무효화시킨다. 그렇지 않으면 링크 후에 putchar() 함수가 2개로 되어 에러가 발생하기 때문이다.

이제 이 배치 파일을 이용하여 아래와 같이 사용자 프로그램을 컴파일하고 링크하여 TEST51A.HEX 파일을 생성하고, 이를 *OK-8051* 키트에 다운로드하여 실행한다.

```
C:\C51\OK8051>BAT51 TEST51A Enter
C:\C51\OK8051>DL51 TEST51A Enter
```

다음에는 A/D 컨버터로 전압 및 온도값을 읽어들여 이를 부동소수점수로 LCD 모듈에 표시하는 <프로그램 TEST51B.C>를 printf() 함수를 사용하는 방식으로 작성하여 보기로 한다. 이것도 역시 printf() 함수를 사용하지 않고 일반적인 방법으로 프로그래밍하면 아래의 <프로그램 TEST51BX.C>와 같다. 역시, printf() 함수를 사용하면 프로그램이 훨씬 더 간단하고 편리해지는 것을 느낄 수 있다.

<프로그램 TEST51B.C> printf() 함수를 사용하여 LCD 모듈에 실수를 표시하는 프로그램

```
/* TEST51B.C */
/* ========================================================================= */
/*      Floating-Point Number Display on Text LCD Module by printf()         */
/* ========================================================================= */
```

```
/*                                    Programmed by Duck-Yong Yoon in 2002.   */

#include <REG52.H>
#include <STDIO.H>
#include "OK8051.H"
#include "OK8051FN.H"

void Delay_10us(unsigned char us10)     /* time delay for us10 * 10[us] */
{ unsigned char i, j;

  for(i = 1; i <= us10; i++)
    j = 0;
}

main()
{ unsigned char i;
  unsigned int ad_sum;

  LCD_string(0x80,"Floating Numbers");          /* display title */
  LCD_string(0xC0," 0.0[V]  00.0");
  LCD_data(0xDF);
  LCD_string(0xCE,"C ");
  Beep();

  while(1)
    { ad_sum = 0;                               /* initial sum of A/D data */
      for(i = 1; i <= 10; i++)
        { ADC_CH0 = 0;                          /* select A/D channel 0 */
          ADC_START = 0;                        /* start conversion */
          Delay_10us(15);                       /* dealy 150 us */
          ad_sum += ADC_READ;                   /* read and add A/D data */
        }
      LCD_command(0xC1);                        /* display VR1 voltage */
      printf("%3.1f",(ad_sum/10.)*(5./255.));

      ad_sum = 0;                               /* initial sum of A/D data */
      for(i = 1; i <= 10; i++)
        { ADC_CH1 = 0;                          /* select A/D channel 1 */
          ADC_START = 0;                        /* start conversion */
          Delay_10us(15);                       /* dealy 150 us */
          ad_sum += ADC_READ;                   /* read and add A/D data */
        }
      LCD_command(0xC9);                        /* display temperature */
      printf("%4.1f",(ad_sum/10.)*(100./255.));

      Delay(200);                               /* delay 200 ms */
    }
}
```

<프로그램 TEST51BX.C> 사용자 전용 함수를 사용하여 LCD 모듈에 실수를 표시하는 프로그램

```
/* TEST51BX.C */
/* ============================================================================ */
/*              Floating-Point Number Display on Text LCD Module                */
/* ============================================================================ */
/*                                    Programmed by Duck-Yong Yoon in 2002.   */

#include <REG52.H>
#include "OK8051.H"
#include "OK8051FN.H"
```

```
void Delay_10us(unsigned char us10)     /* time delay for us10 * 10[us] */
{ unsigned char i, j;

  for(i = 1; i <= us10; i++)
    j = 0;
}

void LCD_1d1(float number)              /* display floating point number x.x */
{ unsigned int i, j;

  j = (int)(number*10. + 0.5);
  i = j/10;                                     /* 10^0 */
  LCD_data(i + '0');
  LCD_data('.');
  i = j % 10;                                   /* 10^-1 */
  LCD_data(i + '0');
}

void LCD_2d1(float number)              /* display floating point number xx.x */
{ unsigned int i, j;

  j = (int)(number*10. + 0.5);
  i = j / 100;                                  /* 10^1 */
  if(i == 0) LCD_data(' ');
  else       LCD_data(i + '0');

  j = j % 100;                                  /* 10^0 */
  i = j / 10;
  LCD_data(i + '0');
  LCD_data('.');

  i = j % 10;                                   /* 10^-1 */
  LCD_data(i + '0');
}

main()
{ unsigned char i;
  unsigned int ad_sum;

  LCD_string(0x80,"Floating Numbers");          /* display title */
  LCD_string(0xC0," 0.0[V]  00.0");
  LCD_data(0xDF);
  LCD_string(0xCE,"C ");
  Beep();

  while(1)
    { ad_sum = 0;                               /* initial sum of A/D data */
      for(i = 1; i <= 10; i++)
        { ADC_CH0 = 0;                          /* select A/D channel 0 */
          ADC_START = 0;                        /* start conversion */
          Delay_10us(15);                       /* dealy 150 us */
          ad_sum += ADC_READ;                   /* read and add A/D data */
        }
      LCD_command(0xC1);                        /* display VR1 voltage */
      LCD_1d1((ad_sum/10.)*(5./255.));

      ad_sum = 0;                               /* initial sum of A/D data */
      for(i = 1; i <= 10; i++)
        { ADC_CH1 = 0;                          /* select A/D channel 1 */
          ADC_START = 0;                        /* start conversion */
          Delay_10us(15);                       /* dealy 150 us */
          ad_sum += ADC_READ;                   /* read and add A/D data */
        }
      LCD_command(0xC9);                        /* display temperature */
      LCD_2d1((ad_sum/10.)*(100./255.));

      Delay(200);                               /* delay 200 ms */
    }
}
```

마찬가지로 이를 아래와 같이 배치 파일을 이용하여 컴파일하고 링크한 후에 이를 *OK-8051* 키트에 다운로드하고 실행한다.

```
C:\C51\OK8051>BAT51 TEST51B↵
C:\C51\OK8051>DL51 TEST51B↵
```

이상의 예에서 보면 알 수 있듯이 LCD 모듈에 수치 데이터를 출력할 때 printf() 함수를 사용하면 전용의 사용자 함수를 만들어 사용하는 것보다 훨씬 편리하다. 그러나, 더욱 편리한 것은 데이터의 표현 형식을 변경할 때 나타난다. 전용의 사용자 함수를 사용하는 경우에는 수치 데이터의 표시 자릿수를 변경하거나 사용하는 진법을 변경할 때마다 사용자 정의 함수를 수정 또는 다시 작성해야 하지만, printf() 함수를 사용하는 경우에는 단지 %서식만 바꾸어 지정하면 간단히 처리된다.

◀참고▶ 참고문헌 1, 2의 많은 예제에서는 텍스트형 LCD 모듈에 수치 데이터를 표시할 때 이렇게 printf() 함수를 사용하면 편리함에도 불구하고 굳이 불편하게 텍스트형 LCD 모듈에 데이터를 출력하는 루틴이나 RS-232C 통신으로 데이터를 출력하는 루틴을 자체적인 전용의 사용자 함수로 만들어서 사용한다. 그것은 프로그램에 따라서는 하나의 프로그램에서 텍스트형 LCD 디스플레이와 RS-232 통신을 모두 사용할 필요가 있는데, printf() 함수로는 이중에서 어느 한가지만을 지정하여 사용할 수밖에 없게 되고 그렇게 되면 어차피 나머지의 I/O는 사용자 정의 함수를 사용할 수밖에 없기 때문이다.

따라서, 사용자가 이와 같이 저수준의 스트림 출력 함수를 수정하여 사용하는 것은 C언어의 일반적인 기능을 이용하는 것이므로 상당히 좋다고 할 수도 있지만, C컴파일러에 따라서는 이러한 기능을 지원하지 않는 경우도 있고 이 방법이 2가지의 I/O를 동시에 지원하지 못하므로 마이크로콘트롤러용의 C언어에서는 오히려 이들 책에서처럼 printf() 함수를 사용하지 않고 전용의 사용자 함수를 만들어 사용하는 방법이 보다 일반적인 방법이라고 말할 수도 있다.

2. 80C196KC MCU용의 Intel C컴파일러 IC96에서 printf() 함수의 올바른 사용

Intel사의 MCS-96 C컴파일러에서도 printf() 함수가 표준적인 %서식 지정 기능을 대부분 지원하며, 이것의 실행 결과가 디폴트로 80C196KC에 내장된 직렬 포트를 통하여 출력된다. 마찬가지로 여기서도 이 함수의 기능을 사용자가 수정하여 이 함수의 출력을 다른 장치로 내보내도록 할 수 있다. 그러나, IC96 컴파일러의 매뉴얼에서는 이에 관한 사항을 자세히 취급하지 않고 있으므로 상당 부분을 필자의 경험과 추측에 의하여 설명하기로 한다.

인텔사의 IC96의 매뉴얼을 통하여 이러한 기술적인 방법을 수행하는데는 한계가 있으므로 본 필자는 참고문헌 3을 집필할 당시만 하더라도 이러한 printf() 함수를 수정하는 방법을 충분히 알아내지 못해 이를 사용하지 못했었다. 그러다가 오랜 시행착오 끝에 이 문제를 해결하여 참고문헌 2에서 자세히 설명할 수 있게 되었으며, 아래는 이 책에 있는 내용을 요약한 것이다. 여기서 *OK-196* 키트의 사용에 관해서는 취급하지 않으니 필요하면 참고문헌

2를 찾아보기 바란다.

(1) IC96에서 printf() 함수를 사용하는 문제의 개요

본 필자는 Intel사의 오래된 매크로 어셈블러인 ASM96에 대하여는 일찍부터 사용자 가이드(User's Guide)의 사본을 구하여 가지고 있으나, C컴파일러 IC96에 대해서는 사용자 가이드를 가지고 있지 않으며 이를 읽어본 적도 없다. 더구나, 이 인텔 소프트웨어는 이미 오래전에 단종되었으므로 지금은 이러한 문헌자료를 거의 구할 수도 없는 실정이다.

하지만, 우리는 지금 이 소프트웨어를 승계하여 발전시키고 있는 Tasking사의 C196 사용자 가이드를 볼 수가 있다. Tasking사의 ASM196 사용자 가이드가 인텔의 ASM96 매뉴얼을 거의 그대로 답습하고 있는 것으로 미루어 Tasking사의 C196 사용자 가이드도 인텔의 IC96 매뉴얼과 상당히 유사할 것으로 짐작할 수 있다. Tasking사의 ASM196 사용자 가이드와 C196 사용자 가이드는 필자의 개인 홈페이지 [문서 자료실]에 올려 두었다.

C196의 사용자 가이드에서 이 printf() 함수를 사용자가 수정하여 사용하는 방법에 대하여는 자세하게 설명하지 않고 있으며, 단지 다음과 같은 3가지의 사항만을 간단히 언급하고 있다.

① printf() 함수는 기본적으로 직렬 포트로 출력을 수행하고 있지만, 내부적으로 항상 putchar() 함수를 호출하여 동작을 수행하고 있으므로 이 putchar() 함수를 수정하면 직렬 포트가 아닌 다른 장치로 출력하는 것이 가능하다.

② printf() 함수에서는 오직 %p 형식을 제외한 모든 ANSI C 표준의 서식화된 출력(formatted output) 동작이 가능하다.

③ printf() 함수에서 부동소수점수를 출력하기 위해서는 미리 fpinit()를 실행하여 부동소수점 라이브러리 FPAL96.LIB의 기능을 초기화해 두어야 한다.

사용자 가이드의 이러한 간단한 설명만을 근거로 하여 본 필자는 *OK-196* 키트에서도 *OK-8051* 키트에서처럼 printf() 함수를 이용하여 간편하게 텍스트형 LCD 모듈에 서식화된 출력을 수행하기 위하여 여러 가지의 방법으로 시행착오를 거듭하였다. 그러다가 마침내 이를 사용하는 올바른 방법을 경험적으로 알아내게 되었는데, 이를 요약하면 다음과 같다.

④ printf() 함수를 이용하여 직렬 포트가 아닌 다른 장치로 서식화된 출력을 수행하려면 사용자가 직접 putchar() 함수를 가지는 PUTCHAR.C 파일을 작성해야 하며, 이를 컴파일하여 생성된 PUTCHAR.OBJ 파일은 물론이고 IC96 소프트웨어 팩키지에 기본적으로 제공되는 PRINTF.OBJ 파일을 사용자 프로그램에 함께 링크해 주어야 한다.

⑤ PRINTF.OBJ 파일은 매우 많은 용량의 스택을 사용하므로 이를 링크할 때는 스택 메모리를 충분한 크기로 할당해 주어야 한다.

결과적으로 이렇게 하면 인텔의 IC96 컴파일러에서도 카일의 C51에서와 상당히 유사한 방법으로 사용자가 printf() 함수를 수정하여 사용할 수 있는 것이다. 그러나, 여기서 만약 PUTCHAR.OBJ 파일만을 링크하고 PRINTF.OBJ 파일은 링크하지 않으면 printf() 함수의 일부 기능을 올바르게 수행되지만 다른 기능들은 제대로 수행되지 않게 된다. 이처럼 알고 나면 단순한 내용이지만 소프트웨어 개발자가 매뉴얼에 자세히 설명해 주지 않는 것을 사용자가 경험적으로 알아내어 이를 제대로 사용하는 일은 쉽지 않았다. 사실 본 필자는 IC96을 사용해 오면서 그 동안에 이와 비슷한 일이 몇 차례 있었다. 그만큼 IC96은 사용자 매뉴얼을 구하기도 어렵거니와 이를 구한다고 하더라도 사용자가 단지 매뉴얼의 내용만으로는 중요한 기술적인 문제를 알기 어려운 경우가 많다는 이야기이다.

(2) PUTCHAR.C 파일의 작성

Keil의 8051용 C컴파일러에서는 기본적으로 putchar() 함수를 포함하는 PUTCHAR.C 소스 파일을 제공하므로 이를 사용자가 필요한대로 수정하기만 하면 되었다. 그러나, IC96 팩키지에는 이 파일을 제공하지 않으므로 printf() 함수를 직렬 포트가 아니라 LCD 모듈에 맞도록 수정하려면 먼저 사용자가 C:\80196\LIB 디렉토리에서 아래 <파일 PUTCHAR.C>와 같이 PUTCHAR.C 소스 파일을 작성하여 기본적인 런타임 라이브러리에서 제공하는 putchar() 함수의 기능을 수정하여야 한다.

<파일 PUTCHAR.C> 1문자를 LCD 모듈에 출력하는 putchar() 함수 파일

```
/*************************************************************************/
/*                                                                       */
/*  PUTCHAR.C:  This routine is the character output to LCD module.      */
/*                                                                       */
/*************************************************************************/

#include <C:\80196\INCLUDE\80C196KC.H>

extern unsigned char LCD_RD_STATUS;      /* LCD module address */
extern unsigned char LCD_WR_DATA;

int putchar(int c)
{ unsigned int i;

  if(c == '\n')                          /* if CR, return */
    return(c);

  for(i = 0; i < 2000; i++)              /* check busy flag */
    { if((LCD_RD_STATUS & 0x80) == 0)
        break;
    }

  LCD_WR_DATA = (char)c;                 /* output c */
  return(c);
}
```

☞ 헤더 파일 STDIO.H에서 보면 이 함수는 int putchar(int)의 형으로 되어 있으므로 여기서도 이 형식에 맞게 프로그램을 작성해야 한다.

텍스트형 LCD 모듈에 1문자를 출력하기 전에 busy flag을 체크한다. 그러나, 이를 단순한 무한 루프로 처리하면 텍스트형 LCD 모듈이 *OK-196* 키트에서 제거되거나 또는 LCD 모듈의 고장으로 인하여 올바른 동작을 하지 않을 경우에 프로그램이 무한 루프에 빠져 홀딩될 수 있으므로 이를 방지하기 위하여 2000회 이내로 체크 횟수를 제한하였다.

여기서 텍스트형 LCD 모듈을 액세스하는 절대번지를 위와 같은 이름으로 정의하여 사용하면 이것이 나중에 사용자 프로그램에서 사용하는 것과 중복되어 경고 메시지가 발생되므로 위에서와 같이 extern으로 선언하여 처리하였음에 유의하라.

◀참고▶ 여기서도 putchar() 함수를 이용한 저수준의 스트림 출력 기능을 사용할 때 주의해야 할 것은 이들 루틴의 어디에도 하드웨어에 대한 초기화 기능이 없다는 것이다. 따라서, putchar() 함수를 이용하여 직렬 포트나 텍스트형 LCD 모듈에 출력을 수행하기 전에 이들 하드웨어는 별도의 초기화 루틴을 사용하여 적절히 초기화해 두어야 한다.

이렇게 수정된 putchar() 함수가 사용자 프로그램에서 올바르게 사용되려면 아래와 같이 이 PUTCHAR.C 파일을 컴파일하여 미리 오브젝트 파일 PUTCHAR.OBJ를 생성해 놓아야 한다.

```
C:\80196\LIB>IC96 PUTCHAR.C[Enter↵]
```

(3) 수정된 printf() 함수의 사용

그러면 이제 사용자 프로그램에서 이렇게 수정된 printf() 함수를 사용하기 위한 준비는 모두 끝났다. 실제로 이를 수행해 보기 위하여 간단히 16비트 정수를 10진수와 16진수로 표시하는 예제를 printf() 함수를 사용한 %서식 지정으로 구현해 보기로 한다.

이렇게 작성한 예제는 아래의 <프로그램 TEST96A.C>와 같다. 이를 printf() 함수를 사용하지 않고 일반적인 방법으로 LCD 모듈에 정수 데이터를 출력하는 프로그램으로 작성하여 보면 아래의 <프로그램 TEST96AX.C>와 같다. printf() 함수를 사용하는 경우 프로그램이 훨씬 더 간단하고 편리해진 것을 느낄 수 있지 않은가?

<프로그램 TEST96A.C> printf() 함수를 사용하여 LCD 모듈에 정수를 표시하는 프로그램

```
/* TEST96A.C */
/* ========================================================================= */
/*      Integer Number Display on Text LCD Module by printf()                */
/* ========================================================================= */
/*                                   Programmed by Duck-Yong Yoon in 2002.   */

#include <80C196KC.H>
#include <STDIO.H>
#include "OK196IO.H"
#include "OK196FN.H"
```

```
main()
{ unsigned int i;

  LCD_string(0x80," Integer Number ");          /* display title */
  LCD_string(0xC0," 00000 = 0000H  ");
  Beep();

  i = 1;                                        /* initial integer number */

  while(i != 0)
    { LCD_command(0xC1);                        /* display decimal number */
      printf("%5d",i);
      LCD_command(0xC9);                        /* display hexadecimal number */
      printf("%04X",i);
      Delay(200);                               /* delay 200 ms */
      i++;
    }
}
```

☞ 이 예제에서는 %서식 지정으로 10진수에서 최초로 나타나는 0을 공백문자로 표시하고, 16진수에서는 최초로 0이 나타나더라도 이를 공백 문자로 처리하지 않고 그냥 0으로 표시하도록 하였다.

<프로그램 TEST96AX.C> 사용자 전용 함수를 사용하여 LCD 모듈에 정수를 표시하는 프로그램

```
/* TEST96AX.C */
/* ============================================================================ */
/*              Integer Number Display on Text LCD Module                       */
/* ============================================================================ */
/*                                    Programmed by Duck-Yong Yoon in 2002.   */

#include <80C196KC.H>
#include "0K196IO.H"
#include "0K196FN.H"

void LCD_4hex(unsigned int number)      /* display HEX number xxxxH */
{ unsigned int i;

  i = number >> 12;                             /* 16^3 */
  if(i <= 9) LCD_data(i + '0');
  else       LCD_data(i - 10 + 'A');

  i = (number >> 8) & 0x000F;                   /* 16^2 */
  if(i <= 9) LCD_data(i + '0');
  else       LCD_data(i - 10 + 'A');

  i = (number >> 4) & 0x000F;                   /* 16^1 */
  if(i <= 9) LCD_data(i + '0');
  else       LCD_data(i - 10 + 'A');

  i = number & 0x000F;                          /* 16^0 */
  if(i <= 9) LCD_data(i + '0');
  else       LCD_data(i - 10 + 'A');
}

void LCD_5d(unsigned int number)        /* display 5-digit decimal number */
{ unsigned int i;
  unsigned char flag;
```

```
    flag = 0;
    i = number/10000;                          /* 10^4 */
    if(i == 0) LCD_data(' ');
    else {     LCD_data(i + '0');
               flag = 1;
         }

    number = number % 10000;                   /* 10^3 */
    i = number/1000;
    if((i == 0) && (flag == 0))
               LCD_data(' ');
    else {     LCD_data(i + '0');
               flag = 1;
         }

    number = number % 1000;                    /* 10^2 */
    i = number/100;
    if((i == 0) && (flag == 0))
               LCD_data(' ');
    else {     LCD_data(i + '0');
               flag = 1;
         }

    number = number % 100;                     /* 10^1 */
    i = number/10;
    if((i == 0) && (flag == 0))
               LCD_data(' ');
    else       LCD_data(i + '0');

    i = number % 10;                           /* 10^0 */
    LCD_data(i + '0');
  }

  main()
  { unsigned int i;

    LCD_string(0x80," Integer Number ");       /* display title */
    LCD_string(0xC0," 00000 = 0000H  ");
    Beep();

    i = 1;                                     /* initial integer number */

    while(i != 0)
      { LCD_command(0xC1);                     /* display decimal number */
        LCD_5d(i);
        LCD_command(0xC9);                     /* display hexadecimal number */
        LCD_4hex(i);
        Delay(200);                            /* delay 200 ms */
        i++;
      }
  }
```

이제 이를 컴파일하여 링크할 때는 C:\80196\LIB 디렉토리에 기본적으로 제공되어 있는 PRINTF.OBJ 파일과 여기에 우리가 미리 생성해 두었던 PUTCHAR.OBJ 파일을 함께 링크해야 한다. 이 과정은 좀 복잡하므로 아래와 같이 일련적으로 처리할 수 있는 배치 파일로 작성하기로 한다.

<파일 BAT196.BAT> printf() 함수를 사용한 프로그램을 번역 및 링크하기 위한 배치 파일

```
IC96 %1.C MODEL(KC) SI(C:\80196\INCLUDE) CO
if errorlevel 2 goto END
if errorlevel 3 goto END
RL96 C:\80196\LIB\CSTART.OBJ,%1.OBJ, & < PRINTF.CMD
echo off
if exist %1.OBJ DEL %1.OBJ
OH TEST96.OUT TO %1.HEX
if exist TEST96.OUT DEL TEST96.OUT
:END
```

<파일 PRINTF.CMD> RL96을 위한 링커 커맨드 파일

```
C:\80196\LIB\PRINTF.OBJ,C:\80196\LIB\PUTCHAR.OBJ, &
C:\80196\LIB\C96.LIB,C:\80196\LIB\FPAL96.LIB TO TEST96.OUT &
SS(256) RAM(001AH-00FFH,0100H-01FFH(STACK),1000H-1001H,1200H-1203H,1400H-1403H, &
1600H-160DH,1800H-1800H,1A00H-1A03H,0F000H-0FEFFH) ROM(8000H-0EFFFH,0FF00H-0FFFFH)
```

◀참고▶ 이와 같이 사용자가 RL96을 사용하여 사용자 프로그램의 오브젝트 파일이나 라이브러리 파일들을 링크할 때 PRINTF.OBJ 파일과 PUTCHAR.OBJ 파일을 별도로 링크하여 주면 RL96 링커는 런타임 라이브러리에 내장되어 있던 이들 함수를 무효화시키는 것으로 추정된다. 그렇지 않으면 링크 후에 이들 함수가 각각 2개씩으로 되어 에러가 발생하기 때문이다.

여기서 RL96 명령을 실행하는 행이 매우 길어져서 127문자를 초과하므로 별도의 링커 커맨드 파일로 처리하였으며, 이러한 링커 커맨드 파일에서는 배치 파일에서처럼 %에 의한 환경변수를 사용하지 못하므로 파일명 TEST96.OUT을 직접 사용하였다. 여러 가지의 오브젝트 파일 및 라이브러리 파일을 링크하는 순서에도 유의하고, 스택 용량을 상당히 큰 256바이트로 지정하였다는 점도 주목하라.

이제 이 배치 파일을 이용하여 아래와 같이 사용자 프로그램을 컴파일하고 링크하여 TEST96A.HEX 파일을 생성하고, 이를 *OK-196* 키트에 다운로드하여 실행한다.

```
C:\80196\OK-196>BAT196 TEST96A Enter
C:\80196\OK-196>DL196 TEST96A Enter
```

다음에는 A/D 컨버터로 전압 및 온도값을 읽어들여 이를 부동소수점수로 LCD 모듈에 표시하는 <프로그램 TEST96B.C>를 printf() 함수를 사용하는 방식으로 작성하여 보기로 한다. 이것도 역시 printf() 함수를 사용하지 않고 일반적인 방법으로 프로그래밍하면 아래의 <프로그램 TEST96BX.C>와 같다. 역시, printf() 함수를 사용하면 프로그램이 훨씬 더 간단하고 편리해지는 것을 느낄 수 있다.

<프로그램 TEST96B.C> printf() 함수를 사용하여 LCD 모듈에 실수를 표시하는 프로그램

```
/* TEST96B.C */
/* ========================================================================== */
/*      Floating-Point Number Display on Text LCD Module by printf()          */
/* ========================================================================== */
/*                                    Programmed by Duck-Yong Yoon in 2002.   */

#include <80C196KC.H>
#include <STDIO.H>
#include <FPAL96.H>
#include "OK196IO.H"
#include "OK196FN.H"

main()
{ unsigned char i;
  unsigned int ad_result, ad_temp;

  LCD_string(0x80,"Floating Numbers");          /* display title */
  LCD_string(0xC0," 0.0[V]  00.0");
  LCD_data(0xDF);
  LCD_string(0xCE,"C ");
  Beep();

  fpinit();                                     /* initialize FPAL96.LIB */

  ioc2 = 0x00;                                  /* A/D 80C196KB slow mode */

  while(1)
    { ad_result = 0;
      for(i = 1; i <= 10; i++)
        { ad_command = 0x0E;                    /* ACH6(VR1) 10-bit A/D start */
          asm SKIP 0x00;
          asm SKIP 0x00;
          while((ad_result_lo & 0x08) == 0x08);
          ad_temp = ad_result_hi * 256 + ad_result_lo;
          ad_temp >>= 6;
          ad_result += ad_temp;
        }
      LCD_command(0xC1);                        /* display VR1 voltage */
      printf("%3.1f",(ad_result/10.)*(5./1023.));

      ad_result = 0;
      for(i = 1; i <= 10; i++)
        { ad_command = 0x0F;                    /* ACH7(LM35DZ) 10-bit A/D start */
          asm SKIP 0x00;
          asm SKIP 0x00;
          while((ad_result_lo & 0x08) == 0x08);
          ad_temp = ad_result_hi * 256 + ad_result_lo;
          ad_temp >>= 6;
          ad_result += ad_temp;
        }
      LCD_command(0xC9);                        /* display temperature */
      printf("%4.1f",(ad_result/10.)*(100./1023.));

      Delay(200);                               /* delay 200 ms */
    }
}
```

<프로그램 TEST96BX.C> 사용자 전용 함수를 사용하여 LCD 모듈에 실수를 표시하는 프로그램

```
/* TEST96BX.C */
/* ========================================================================== */
/*              Floating-Point Number Display on Text LCD Module              */
/* ========================================================================== */
/*                                      Programmed by Duck-Yong Yoon in 2002.  */

#include <80C196KC.H>
#include "OK196IO.H"
#include "OK196FN.H"

void LCD_1d1(float number)              /* display floating point number x.x */
{ unsigned int i, j;

  j = (int)(number*10. + 0.5);
  i = j/10;                                         /* 10^0 */
  LCD_data(i + '0');
  LCD_data('.');
  i = j % 10;                                       /* 10^-1 */
  LCD_data(i + '0');
}

void LCD_2d1(float number)              /* display floating point number xx.x */
{ unsigned int i, j;

  j = (int)(number*10. + 0.5);
  i = j / 100;                                      /* 10^1 */
  if(i == 0) LCD_data(' ');
  else       LCD_data(i + '0');

  j = j % 100;                                      /* 10^0 */
  i = j / 10;
  LCD_data(i + '0');
  LCD_data('.');

  i = j % 10;                                       /* 10^-1 */
  LCD_data(i + '0');
}

main()
{ unsigned char i;
  unsigned int ad_result, ad_temp;

  LCD_string(0x80,"Floating Numbers");          /* display title */
  LCD_string(0xC0," 0.0[V]  00.0");
  LCD_data(0xDF);
  LCD_string(0xCE,"C ");
  Beep();

  fpinit();                                         /* initialize FPAL96.LIB */

  ioc2 = 0x00;                                      /* A/D 80C196KB slow mode */

  while(1)
    { ad_result = 0;
      for(i = 1; i <= 10; i++)
        { ad_command = 0x0E;                        /* ACH6(VR1) 10-bit A/D start */
          asm SKIP 0x00;
          asm SKIP 0x00;
          while((ad_result_lo & 0x08) == 0x08);
          ad_temp = ad_result_hi * 256 + ad_result_lo;
          ad_temp >>= 6;
          ad_result += ad_temp;
        }
```

```
        LCD_command(0xC1);                          /* display VR1 voltage */
        LCD_1d1((ad_result/10.)*(5./1023.));

        ad_result = 0;
        for(i = 1; i <= 10; i++)
          { ad_command = 0x0F;                      /* ACH7(LM35DZ) 10-bit A/D start */
            asm SKIP 0x00;
            asm SKIP 0x00;
            while((ad_result_lo & 0x08) == 0x08);
            ad_temp = ad_result_hi * 256 + ad_result_lo;
            ad_temp >>= 6;
            ad_result += ad_temp;
          }
        LCD_command(0xC9);                          /* display temperature */
        LCD_2d1((ad_result/10.)*(100./1023.));

        Delay(200);                                 /* delay 200 ms */
      }
  }
```

마찬가지로 이를 아래와 같이 배치 파일을 이용하여 컴파일하고 링크한 후에 이를 *OK-196* 키트에 다운로드하고 실행한다.

```
C:\80196\OK-196>BAT196 TEST96B Enter
C:\80196\OK-196>DL196 TEST96B Enter
```

이상의 예에서 보면 알 수 있듯이 LCD 모듈에 수치 데이터를 출력할 때 printf() 함수를 사용하면 전용의 사용자 함수를 만들어 사용하는 것보다 훨씬 편리하다. 그러나, 더욱 편리한 것은 데이터의 표현 형식을 변경할 때 나타난다. 전용의 사용자 함수를 사용하는 경우에는 수치 데이터의 표시 자릿수를 변경하거나 사용하는 진법을 변경할 때마다 사용자 정의 함수를 수정 또는 다시 작성해야 하지만, printf() 함수를 사용하는 경우에는 단지 %서식만 바꾸어 지정하면 간단히 처리된다.

마이크로콘트롤러용의 C컴파일러에서는 이와 같이 printf() 함수가 기본적으로 직렬 포트로 동작하게 되어 있으나, 사용자가 이를 다른 장치로 동작하도록 수정할 수 있다. 그러나, 이 함수를 그냥 직렬 포트에서 동작하도록 사용하는 경우에는 단지 소스 프로그램에서 헤더 파일 STDIO.H를 인클루드한 후에 printf() 함수를 사용하는 것으로 충분하다.

【 참고문헌 】

1. 윤덕용, 어셈블리와 C언어로 익히는 8051 마스터, Ohm사, 2001
2. 윤덕용, LCD 모듈의 철저 활용, Ohm사, 2002
3. 윤덕용, 어셈블리와 C언어로 익히는 80C196KC 마스터, Ohm사, 2000