【 기술 노트 15 】

C언어에서의 인라인 어셈블

마이크로프로세서, 그 중에서도 특히 마이크로콘트롤러에서 C언어로 프로그램을 작성하다보면 가끔 어셈블리 언어의 필요성을 느끼게 되는 경우가 있다. 원래 C언어는 이식성이 좋다는 것이 그 중요한 속성이며, 이는 반대로 말하면 C언어는 그만큼 하드웨어 의존적인 기능이 취약하다는 것인데, 마이크로콘트롤러에서는 범용 컴퓨터에 비하여 하드웨어 의존적인 일을 처리해야 할 경우가 많기 때문이다. 물론 이러한 이유로 마이크로콘트롤러용의 C컴파일러에서는 범용 컴퓨터의 C컴파일러에 비하여 많은 하드웨어 처리기능을 가지도록 설계되어 있기는 하지만 그래도 이것으로는 부족할 때가 많다는 것이다.

이러한 경우에는 해당 루틴만을 별도로 어셈블리 언어 파일로 작성하여 어셈블한 이후에 이를 나중에 메인 C프로그램의 오브젝트 파일과 링크하는 것이 일반적인 방법이다. 그러나, 어셈블리 언어로 작성하고자 하는 루틴이 상당히 짧고 간단한 경우에는 이렇게 별도의 어셈블리 파일로 작성하는 것이 번거롭고 또한 여러 개의 파일을 관리하기가 불편할 수도 있다. 대부분의 C컴파일러는 이러한 경우에 사용할 수 있도록 인라인 어셈블(inline assemble) 기능을 제공한다. 이것은 C언어 소스의 중간에 어셈블리 명령을 포함시켜 프로그램을 작성하는 기능으로서, 이렇게 작성된 어셈블리 명령은 C컴파일러에 의하여 번역되지 않고 중간 생성 파일에 포함되어 있다가 나중에 어셈블러에 의하여 C소스에서 만들어진 루틴들과 함께 번역 처리되는 것이 보통이다.

인라인 어셈블 기능은 C컴파일러마다 사용 방법이 조금씩 차이가 있다. 그러나, 초기에는 불편하고 제한 사항이 많던 이들 인라인 어셈블 기능이 요즈음에 나오는 C컴파일러에서는 점차로 개선되어 기능이 강력해지고 사용이 더욱 편리하게 되어 가는 추세이다.

여기서는 TMS320C3X DSP, 80C196KC MCU, 8051 MCU 등의 C컴파일러에서 각각 인라인 어셈블 기능을 사용하는 방법을 요약하여 정리하기로 한다.

1. TMS320C3X DSP의 경우

TMS320C3X/4X C컴파일러는 최적의 성능을 발휘하기 위하여 어셈블러와 매우 밀접하게 관련되어 사용되도록 설계되었고, 따라서 TMS320C3X/4X DSP의 어셈블러가 사용하는 거의 모든 어셈블리 명령이나 지시어를 사용할 수 있는 매우 우수한 인라인 어셈블 기능을 지원하며 사용법도 아주 간단하다. 이를 통하여 이 C 컴파일러는 C언어로는 액세스가 불가능한 여러가지 레지스터의 액세스나 하드웨어적인 처리가 가능하다.

TMS320C3X/4X C컴파일러의 인라인 어셈블 명령은 다음과 같이 asm 문에 문자열 인수(string-constant argument)로 어셈블리 명령이나 지시어를 표현하는 형식으로 사용된다.

asm("*assembler text*"); [/* *comment* */]

여기서 *assembler text*는 반드시 큰 인용부호(" ")로 묶어져야 하며, 일반적인 문자열의 escape code를 사용할 수 있다. 또한, *assembler text*에는 어셈블리 언어에서 사용할 수 있는 거의 모든 DSP 명령이나 어셈블리 지시어 등이 사용되며, 어셈블리 언어 프로그램에서처럼 반드시 label, blank, tab, comment(* 또는 ;) 등으로 시작되어야 한다. 이처럼 주석문은 인라인 어셈블 명령 내부에서 사용할 수도 있지만, 위의 *comment*는 인라인 어셈블 명령 밖에서 사용하는 주석문으로서 항상 C언어 형식(/* */)으로 사용해야 한다.

예
```
asm("LABEL1:  LDI     @_var1,R0      ");
asm("         .sect   \"section1\"   ");
```

C 컴파일러는 *assembler text*의 내용을 조사하지 않고 무조건 출력 파일에 복사하며, 만약 여기에 에러가 있다면 이는 나중에 어셈블러에 의하여 검출된다.

asm 문은 근본적으로 C언어로는 처리하기 어려운 하드웨어들을 액세스할 목적으로 만들어진 것이다. 그러나, C 컴파일러는 *assembler text*의 내용을 조사하지 않고 무조건 출력 파일에 복사하여 모아 두었다가 나중에 어셈블러에 의하여 어셈블하므로, 원래의 C 소스 프로그램내에서와는 전체적인 명령 코드의 순서가 상당히 달라질 수 있으며 이 때문에 뜻하지 않은 문제가 발생할 수 있다는 점에 유의해야 한다. 이러한 점에서 인라인 어셈블 명령은 C 소스에 예상치 않은 영향을 주지 않도록 주의하여 사용해야 하며, 최적화를 수행할 때 또다른 문제를 야기하지 않는지에도 주의해야 한다.

asm 문장은 마치 C언어에서의 함수와 같은 구조를 가지나 이것은 일반적인 C언어의 규칙을 따르지 않는다. 즉, asm 문은 문장이나 선언처럼 main 함수나 사용자 정의함수의 외부에 위치할 수도 있다. 따라서, 이는 컴파일러 모듈의 서두에서 어떤 지시어를 삽입할 때 유용하게 사용될 수 있고, 특히 cstartup 코드나 인터럽트 벡터 테이블 등을 작성하는 경우에도 매우 유용하다. 이 때문에 대부분의 마이크로콘트롤러에서는 C언어 프로그램을 작성할 때 별도의 어셈블리 파일에서 cstartup 코드를 만들어 링크하지만, 이 DSP에서는 사용자 C 프로그램의 서두 부분에 인라인 어셈블 명령으로 cstartup 코드를 포함시켜 훨씬 간단하게 C언어 프로그램을 작성할 수 있다.

다음은 C언어 프로그램에서 다양한 용도로 인라인 어셈블 명령을 사용한 예를 보여주는 프로그램이다. 여기서는 인라인 어셈블 기능을 이용하여 cstartup 코드를 C소스 내에서 처리하였고, DSP를 초기화하기 위하여 여러가지 레지스터들을 액세스하며, 일반적으로 C언어에서는 구현하기 어려운 정확한 시간지연 루틴 함수를 쉽게 프로그램하는 등의 예를 살펴볼 수 있다.

```
/* HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH */
/* H                                                                    H */
/* H                TMS-32 V1.0 :  LED Control Program                  H */
/* H                                                                    H */
/* H               Programmed by Duck-Yong Yoon in 1998                 H */
/* H                                                                    H */
/* HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH */

/* ==================================================================== */
/*      Boot Loader for Startup in C Program                            */
/* ==================================================================== */
  asm("        .text                  ");
  asm("        .global _c_int00       ");
  asm("        BR      _c_int00       ");

/* ==================================================================== */
/*      TMS-32 V1.0 Internal I/O Address                                */
/* ==================================================================== */
int     *Timer0_GCR     = (int *)0x808020;      /* timer 0 global control register */
int     *Timer0_PR      = (int *)0x808028;      /* timer 0 period register */
int     *Timer1_GCR     = (int *)0x808030;      /* timer 1 global control register */
int     *Timer1_PR      = (int *)0x808038;      /* timer 1 period register */
int     *Serial_TX      = (int *)0x808042;      /* serial FSX/DX/CLKX control register */

/* ==================================================================== */
/*      TMS-32 V1.0 External I/O Address                                */
/* ==================================================================== */
int     *PPI_PA         = (int *)0x810000;      /* 8255A PPI */
int     *PPI_PB         = (int *)0x810001;
int     *PPI_PC         = (int *)0x810002;
int     *PPI_CW         = (int *)0x810003;

int     *PCI_DATA       = (int *)0x811000;      /* 8251A PCI */
int     *PCI_CW         = (int *)0x811001;

int     *ADC_CS         = (int *)0x812000;      /* MAX122 A/D converter */
int     *ADC_START      = (int *)0x813000;      /* MAX122 conversion start */
int     *ADC_MUX        = (int *)0x814000;      /* multiplxer for MAX122 */

int     *DAC_CH0        = (int *)0x815000;      /* AD664 D/A converter */
int     *DAC_CH1        = (int *)0x815001;
int     *DAC_CH2        = (int *)0x815002;
int     *DAC_CH3        = (int *)0x815003;

int     *SRAM_CS        = (int *)0x900000;      /* KM68257 SRAM */

/* ==================================================================== */
/*      Common User Functions                                           */
/* ==================================================================== */
void Init_DSP()                         /* initialize TMS320C32 DSP */
{ asm("        LDI     0000H,IF       ");      /* disable all interrupts */
  asm("        LDI     0000H,IE       ");
  asm("        LDI     0800H,ST       ");      /* enable cache */
  *Serial_TX = 0x0222;                          /* BUZZER = PWM = FOR/-REV = 0 */
}

int Delay_count;                        /* declare global variable */

void Delay_us(int us)                   /* delay by [us] unit */
{ Delay_count = us;
  asm("        LDI     @_Delay_count,R7");
  asm("TIME1US: RPTS   15             ");
  asm("        NOP                    ");
  asm("        SUBI    1,R7           ");
  asm("        BNZ     TIME1US        ");
}
```

```
void Delay_ms(int ms)                   /* delay by [ms] unit */
{ Delay_count = 1000*ms;
  Delay_us(Delay_count);
}

void Beep()                             /* beep sound for 50 ms */
{ *Serial_TX = 0x0404 | *Serial_TX;             /* buzzer ON */
  Delay_ms(50);
  *Serial_TX = 0x0BFB & *Serial_TX;             /* buzzer OFF */
}

void Init_8255()                        /* initialize 8255A PPI */
{ *PPI_CW = 0x82; Delay_us(3);                  /* port A & C = output, B = input */
  *PPI_PA = 0x00; Delay_us(3);                  /* output port A and C with 0 */
  *PPI_PC = 0x00; Delay_ms(1);
}

void LCD_command(int command)          /* output a command to LCD module */
{ *PPI_PC = 0x8F & *PPI_PC;                     /* Rs = 0, R/-W = 0, E = 0 */
  *PPI_PA = command;
  *PPI_PC = 0x40 | *PPI_PC;                     /* Rs = 0, R/-W = 0, E = 1 */
  *PPI_PC = 0x8F & *PPI_PC;                     /* Rs = 0, R/-W = 0, E = 0 */
  Delay_us(50);
}

void Init_LCD()                         /* initialize LCD module */
{ int i, LCD_InitWord[4] = { 0x38, 0x0C, 0x06, 0x01 };

  for(i = 0; i <= 3; i++)
    { LCD_command(LCD_InitWord[i]); Delay_ms(2);
    }
}

void LCD_data(int character)            /* display a character on LCD module */
{ *PPI_PC = 0x10 | (0x8F & *PPI_PC);            /* Rs = 1, R/-W = 0, E = 0 */
  *PPI_PA = character;
  *PPI_PC = 0x40 | *PPI_PC;                     /* Rs = 1, R/-W = 0, E = 1 */
  *PPI_PC = 0x9F & *PPI_PC;                     /* Rs = 1, R/-W = 0, E = 0 */
  *PPI_PC = 0x8F & *PPI_PC;                     /* Rs = 0, R/-W = 0, E = 0 */
  Delay_us(50);
}

void LCD_string(int command, char *string) /* output string to LCD module */
{ LCD_command(command);                         /* set cursor position */
  while(*string != '\0')                        /* output string */
    { LCD_data(*string); string++;
    }
}

int Key_input()                         /* input key SW1 - SW4 */
{ int key;
  key = 0x0F & *PPI_PB;                         /* read key */
  if(key == 0x0F) return key;                   /* if no key, return */

  while((0x0F & *PPI_PB) != 0x0F)               /* wait for key-off */
    Delay_us(3);
  Delay_ms(50); return key;
}


/* ================================================================== */
/*      Main Program                                                  */
/* ================================================================== */
void main()
{ int i;
  int LED[16] = { 0x01, 0x02, 0x04, 0x08, 0x04, 0x02, 0x01, 0x00,
                  0x01, 0x03, 0x07, 0x0F, 0x07, 0x03, 0x01, 0x00 };
```

```
    Init_DSP(); Delay_ms(50);                    /* initialize DSP, 8255A, LCD */
    Init_8255();
    Init_LCD();
    Beep();

    LCD_string(0x80, "LED Control by C");        /* display title */

    for(;;)
      { for(i = 0; i <= 15; i++)
          { *PPI_PC = LED[i];                    /* display LED1 - LED4 */
            Delay_ms(500);                       /* delay 500 ms */
          }
        Beep();
      }
}
```

2. 80C196KC 마이크로콘트롤러의 경우

인텔사에서 오래전에 개발한 80C196KC와 같은 MCS-96용 C컴파일러인 IC96.EXE는 1990년대 초반에 개발된 상당히 오래된 소프트웨어임을 감안하면 비교적 우수한 인라인 어셈블 기능을 지원하고 있다. 그러나, 이것은 상당히 사용하기가 편리한 장점이 있는 반면에 ASM96.EXE에서 사용하는 어셈블리 언어에 비하여 많은 제약 사항을 가지고 있다. 이것은 어셈블리 언어 기능의 대부분을 인라인 어셈블에서도 그대로 사용할 수 있는 DSP의 경우와 상당히 대조적이다.

IC96 C컴파일러의 인라인 어셈블 명령은 아래와 같이 asm 지시어 다음에 1개의 어셈블리 명령을 사용하거나 또는 여러개의 어셈블리 명령을 { }안에 묶어서 나열하는 형식으로 사용할 수 있다.

asm *assembly instruction* ; [/* *comment* */]

또는

asm { *assembly instruction* ; [/* *comment* */]
.
.
.
}

여기서 *assembly instruction*은 모두 어셈블리 명령과 같으나 항상 세미콜론(;)으로 끝나야 하며, *comment*는 주석문으로서 항상 C언어 형식(/* */)으로 사용해야 하고 어셈블리 프로그램에서처럼 세미콜론으로 시작하는 주석문은 사용할 수 없다.

이와 같은 인라인 어셈블 기능이 사용되려면 IC96에서 항상 extend 제어 옵션이 지정되어야 한다. 그렇지 않으면 C컴파일러는 asm 지시어와 그 뒤에 있는 어셈블리 명령을 올바르게 인식하지 못한다.

IC96의 인라인 어셈블 기능에서 사용되는 어셈블리 명령은 여러가지의 제한을 가지고 있는데, 이것들의 특징과 제약 사항을 요약하면 다음과 같다.

① 인라인 어셈블의 어셈블리 명령에서는 대소문자를 구분하지 않는다.

② 인라인 어셈블 명령에서 어셈블리 언어 형식의 라벨은 사용할 수 없으며, 새로운 심볼(symbol)을 정의할 수도 없다. 따라서, 제너릭 브랜치(generic branch) 명령에서 사용되는 점프 어드레스는 반드시 C 라벨이어야 한다. C 라벨은 인라인 어셈블 명령의 내부가 아니라 외부에 사용된다.

```
예  label1:  return(1);
    data1:   dcw  0x1234;
```

③ 제너릭 CALL 명령으로 호출하는 서브루틴은 C 언어 소스의 함수명이어야 한다.

④ 수치 상수는 사용되지만 표현식은 사용할 수 없다. 이들 수치 상수는 10진수를 제외하고는 항상 C언어의 형식으로 사용해야 한다. 즉, 3AH와 같은 표현은 사용할 수 없으며 0x3a, 0x3A, 0X3a, 0X3A 등으로 사용해야 한다.

⑤ 상수를 정의하는 지시어는 오직 DCB, DCW, DCL의 3가지만 사용할 수 있으며, 이들 지시어에서 오퍼랜드는 오직 1개씩만 허용된다.

⑥ 심볼을 정의하는 지시어 EQU, SET나 상수를 정의하는 지시어 DCR 및 메모리를 예약하는 지시어 DSB, DSW, DSL, DSR 등은 사용할 수 없다.

⑦ 모듈에 관련되는 지시어 MODULE, EXTRN, PUBLIC, END나 로케이션 카운터에 관련되는 지시어 RSEG, CSEG, DSEG, OSEG, ORG 등은 사용할 수 없다.

⑧ 매크로에 관계되는 지시어나 조건부 어셈블 지시어는 사용할 수 없다.

⑨ 점프 명령 Jxx, DJNZ, DJNZW, SJMP, LJMP 및 서브루틴 호출명령 SCALL, LCALL 등은 사용할 수 없다. 그러나, 이것들의 generic instruction인 Bxx나 CALL은 사용할 수 있다.

⑩ C언어에서 정의된 배열을 인라인 어셈블 명령으로 액세스하려면 배열의 인수를 ( )로 묶어 사용한다.

```
예  const int a[8] = {1,3,5,7,11,12,13,14};
    asm   LD  wreg,a(5);
```

다음은 IC96에서 인라인 어셈블 명령을 사용하는 간단한 예제 프로그램이다. IC96은 하드웨어 의존적인 처리를 수행하는 기능을 상당히 많이 가지고 있기 때문에 C언어 프로그램에서 특별히 꼭 인라인 어셈블 기능을 사용해야만 하는 경우는 매우 드물다. 하지만, 보다 쉽고 정확한 처리를 위하여 인라인 어셈블 기능이 필요하다고 판단되는 경우에는 이처럼 언제라도 쉽게 이를 사용할 수 있다.

```
#pragma model(kc)

#include <stdio.h>

register int ri;

main()
{
   int i;

   init_serio();

   asm { ld  ri, #10;
         st  ri, i;
         add ri, i;
       }

   printf("ri = %d\r\n", ri);
}
```

이상에서 설명한 IC96의 인라인 어셈블 기능은 IC96의 후속 제품인 오늘날 Tasking사의 C컴파일러인 C196에도 그대로 적용된다. C196은 IC96과 그 뿌리가 같으며, 상향 호환성이 유지되는 C컴파일러이기 때문이다.

3. 8051 마이크로콘트롤러의 경우

8051 마이크로콘트롤러에 대하여는 가장 많이 사용되는 Keil사의 C 컴파일러 C51.EXE를 중심으로 설명한다. C51에서는 인라인 어셈블 기능을 지원하기는 하는데, 기능은 우수하지만 그 사용 방법이 상당히 불편하고 특이하다. 즉, C51에서 인라인 어셈블 명령을 사용할 때에는 반드시 C소스 파일을 컴파일하여 중간 어셈블리 소스 파일을 생성하고, 이를 A51 어셈블러로 다시 어셈블하여 B51로 링크하는 번거로운 절차를 거쳐야 한다.

다음은 C51에서 실제로 이 인라인 어셈블 기능을 사용하는 프로그램의 기본 형식을 보여주는 예다.

```
#pragma src(test.asm)

#include <REG52.H>

main()
{
   .
   .
   .
#pragma  asm
  PORTA  EQU   1200H
  NOP
  MOV    DPTR, #PORTA
  MOVX   A, @DPTR
  ANL    A, #0FH
#pragma  endasm
   .
   .
   .
}
```

위의 형식에서 보듯이 C51에서 인라인 어셈블 명령을 사용하는 절차를 요약하면 다음과 같다.

① C소스 파일내에서는 인라인 어셈블 명령의 앞과 뒤에 각각 #pragma asm 및 #pragma endasm 지시어를 사용하여 이 부분이 인라인 어셈블로 처리하려는 부분이라는 것을 표시해야 한다. 그 사이에 들어가는 인라인 어셈블 명령의 행들은 매크로 어셈블러 A51에서의 프로그램 형식과 동일하다.

② C소스 파일의 서두에서 #pragma src(*asm_filename*) 지시어를 사용하든지 또는 C51로 컴파일할 때 옵션 스위치 src(*asm_filename*)를 사용하는 방법으로 컴파일 후에 생성될 중간 어셈블리 소스 파일명을 지정해야 한다.

③ 이렇게 컴파일되어 만들어진 중간 어셈블리 소스 파일을 어셈블러 A51로 어셈블하여 오브젝트 파일을 생성한다.

④ 그러나, 이는 원래 C소스에서 출발한 것이기 때문에 이렇게 만들어진 오브젝트 파일 단독으로는 실행 파일이 만들어질 수 없다. 따라서, 링커 BL51을 이용하여 startup.obj, 이 사용자 프로그램의 오브젝트 파일, C51에 제공되는 런타임 라이브러리 파일 c51x.lib 등 3가지를 링크하여 실행 파일을 만든다.

이와 같이 C51에서 인라인 어셈블 명령을 사용할 때에는 반드시 src() 옵션을 사용하여 중간 어셈블리 소스 파일을 생성하고, 이를 A51 어셈블러로 다시 어셈블하여 B51로 링크하는 번거로운 절차를 밟아야 한다.

그러나, C51에서는 #pragma asm 및 #pragma endasm 지시어를 사용하여 인라인 어셈블 기능을 사용하는 부분을 블록으로 묶었으므로 그 내부에서는 자유롭게 어셈블리 언어에서와 동일한 명령 형식으로 프로그램을 할 수 있다는 점에서 편리하며, 또한 여기서 사용할 수 있는 어셈블리 명령들은 A51에서 가능한 것들을 대부분 포함하고 있다.

다음은 C51에서 인라인 어셈블 기능을 보여주기 위한 예제 프로그램이다. 먼저 사용자가 작성하는 C언어 소스 파일은 다음과 같다.(이것은 소스의 각 행에 행번호를 나타내기 위하여 실제로는 컴파일한 이후의 리스팅 파일 TEST.LST를 보인 것이다.)

```
stmt level    source

   1          #pragma src(test.src)
   2
   3          extern void test();
   4
   5          main()
   6          {
   7   1          test();
   8   1
   9   1      #pragma asm
  10   1          JMP   $        ; endless loop
  11   1      #pragma endasm
  12   1      }
```

이를 C51로 컴파일하여 생성된 중간 어셈블리 소스 파일 TEST.SRC는 다음과 같다. 사용자는 매크로 어셈블러 A51을 사용하여 이것을 다시 어셈블하여야 한다.

```
; TEST.SRC generated from: TEST.C

NAME    TEST

?PR?main?TEST        SEGMENT CODE
        EXTRN   CODE (test)
        EXTRN   CODE (?C_STARTUP)
        PUBLIC  main
; #pragma src(test.src)
;
; extern void test();
;
; main()

        RSEG  ?PR?main?TEST
        USING   0
```

```
main:
                        ; SOURCE LINE # 5
; {
                        ; SOURCE LINE # 6
;    test();
                        ; SOURCE LINE # 7
        LCALL   test
;
; #pragma asm
;    JMP   $            ; endless loop
         JMP   $               ; endless loop
; #pragma endasm
; }
                        ; SOURCE LINE # 12
        RET
; END OF main

        END
```

이와 같이 C51에서 인라인 어셈블 기능을 사용할 때 지정하는 중간 어셈블리 소스 파일의 이름은 기존에 존재하던 파일 이름과 중복을 피하기 위하여 xxx.SRC라고 하는 것이 좋은 방법이다. 이렇게 하면 원래부터 어셈블리 언어로 작성된 사용자 프로그램을 xxx.ASM이라고 하는 것과 파일명이 확실히 구별되어 관리하기도 편리해진다.

【 참고문헌 】

1. TMS320C3X/4X Optimizing C Compiler User's Guide, Texas Instruments, 1998
2. 윤덕용, TMS320C31 마스터, Ohm사, 1998
3. 윤덕용, TMS320C32 마스터, Ohm사, 1999
4. 80C196 C Compiler User's Guide, Tasking, 1999
5. 윤덕용, 어셈블리와 C언어로 익히는 80C196KC 마스터 (I) (II), Ohm사, 2000
6. Cx51 Compiler User's Guide, Keil Software, 2000
7. 윤덕용, 어셈블리와 C언어로 익히는 8051 마스터, Ohm사, 2001