【 기술 노트 22 】

ATmega1281과 ATmega2561의 소프트웨어 호환성에 대하여

ATmega1281과 ATmega2561은 단지 프로그램 메모리로 사용되는 플래시 메모리의 용량이 각각 128KB와 256KB라는 차이가 있을 뿐이며 그외의 모든 사양이나 기능이 같고 명령 세트도 동일하므로 이들 2가지 모델 사이에는 완벽한 소프트웨어의 호환성이 있을 것이라고 생각하기 쉽다. 그러나, 이들 사이에는 매우 중요한 기술적인 차이점이 1가지 더 있어서 상당히 제한적으로만 소프트웨어의 호환성을 가지게 된다. 즉, ATmega1281은 프로그램 카운터가 2바이트 구조이고 ATmega2561은 프로그램 카운터가 3바이트 구조(실제로는 이중에서 17비트만 유효)라는 것이다.

이 때문에 스택을 사용하는 일부 프로그램에서는 소프트웨어의 호환성이 없게 된다. 이를 어셈블리 프로그램의 경우와 C언어 프로그램의 경우로 나누어 설명하면 다음과 같다. 여기서 설명하는 모든 내용은 단지 이 2가지 모델 뿐만 아니라 ATmega1280과 ATmega2560에 대하여도 동일하게 적용된다.

(1) AVR Studio의 매크로 어셈블러를 사용하는 경우

ATmega1281은 프로그램 카운터가 2바이트로 되어 있으므로 CALL/ICALL 명령을 수행하는 경우 스택에 2바이트가 저장되고 스택 포인터도 2만큼 감소하며, 반대로 RET/RETI 명령을 수행할 때는 스택 포인터가 2만큼 증가한다. 그러나, ATmega2561은 프로그램 카운터가 3바이트이므로 CALL/ICALL/EICALL 명령을 수행하는 경우 스택에 3바이트가 저장되고 스택 포인터도 3만큼 감소하며, 반대로 RET/RETI 명령을 수행할 때는 스택 포인터가 3만큼 증가한다.(당연히 각 모델에서 이와같이 2바이트와 3바이트의 프로그램 카운터를 저장 또는 복구하므로 이들 명령의 실행시간도 달라진다. 이에 관해서는 “AVR ATmega1281/2561 정복” 책의 <표 1.3.7> 명령표를 참조하라.)

이에 따라 서브루틴 내에서 스택을 사용하는 프로그램에서는 스택 영역을 액세스할 때 이를 서로 다르게 처리해 주어야 한다. “AVR ATmega1281/2561 정복” 책의 예제 프로그램에서 사용하는 플래시 메모리의 내용을 상수로 읽어오기 위한 서브루틴 LCD_STRING, GLCD_STRING, TX0_STRING, TX1_STRING 등이 이에 해당하는데, 이중에서 LCD_STRING을 보면 다음의 <표 22.1>과 같다.

이 표에서 ATmega1281과 ATmega2561에서 서로 다르게 처리해 주어야 하는 부분은 ★표로 표시하였다. 여기서 보듯이 서브루틴을 호출한 후에 스택에 저장되어 있는 복귀주소인 프로그램 카운터 값을 꺼내려면 ATmega1281에서는 2차례 POP 명령을 사용해야 하지만 ATmega2561에서는 3차례 POP 명령을 사용해야 한다. 반대로 모든 처리가 끝나고 나중

에 다시 복귀주소를 스택에 저장할 경우에도 ATmega1281에서는 2차례 PUSH 명령을 사용해야 하지만 ATmega2561에서는 3차례 PUSH 명령을 사용해야 한다.

<표 22.1> ATmega1281과 ATmega2561에서 서브루틴 LCD_STRING의 차이

ATmega1281의 경우(*OK-1281* 키트)

```
LCD STRING:
  OUT   GPIOR0,ZL      ; store RAMPZ:ZH:ZL
  OUT   GPIOR1,ZH
  IN    ZL,RAMPZ
  OUT   GPIOR2,ZL
  CLR   RAMPZ_BUFFER   ; ★ get start address of string
  POP   ZH             ; ★
  POP   ZL             ; ★
  LSL   ZL             ; convert into byte address
  ROL   ZH
  ROL   RAMPZ_BUFFER
  OUT   RAMPZ,RAMPZ_BUFFER
LCD STR1:
  ELPM  LCD_BUFFER,Z+ ; read a character. end string ?
  CPI   LCD_BUFFER,0
  BREQ  LCD_STR2       ; if yes, return
  CALL  LCD_DATA       ; if no, display a character
  RJMP  LCD_STR1
LCD STR2:
  SBRC  ZL,0           ; if odd address, skip 1 address
  ELPM  LCD_BUFFER,Z+
  IN    RAMPZ_BUFFER,RAMPZ ; convert into word address
  LSR   RAMPZ_BUFFER
  ROR   ZH
  ROR   ZL
  PUSH  ZL             ; ★ store new return address
  PUSH  ZH             ; ★
                       ; ★
  IN    ZL,GPIOR2      ; restore RAMPZ:ZH:ZL
  OUT   RAMPZ,ZL
  IN    ZH,GPIOR1
  IN    ZL,GPIOR0
  RET
```

ATmega2561의 경우(*OK-2561* 키트)

```
LCD STRING:
  OUT   GPIOR0,ZL      ; store RAMPZ:ZH:ZL
  OUT   GPIOR1,ZH
  IN    ZL,RAMPZ
  OUT   GPIOR2,ZL
  POP   RAMPZ_BUFFER   ; ★ get start address of string
  POP   ZH             ; ★
  POP   ZL             ; ★
  LSL   ZL             ; convert into byte address
  ROL   ZH
  ROL   RAMPZ_BUFFER
  OUT   RAMPZ,RAMPZ_BUFFER
LCD STR1:
  ELPM  LCD_BUFFER,Z+ ; read a character. end string ?
  CPI   LCD_BUFFER,0
  BREQ  LCD_STR2       ; if yes, return
  CALL  LCD_DATA       ; if no, display a character
  RJMP  LCD_STR1
LCD STR2:
  SBRC  ZL,0           ; if odd address, skip 1 address
  ELPM  LCD_BUFFER,Z+
  IN    RAMPZ_BUFFER,RAMPZ ; convert into word address
  LSR   RAMPZ_BUFFER
  ROR   ZH
  ROR   ZL
  PUSH  ZL             ; ★ store new return address
  PUSH  ZH             ; ★
  PUSH  RAMPZ_BUFFER   ; ★
  IN    ZL,GPIOR2      ; restore RAMPZ:ZH:ZL
  OUT   RAMPZ,ZL
  IN    ZH,GPIOR1
  IN    ZL,GPIOR0
  RET
```

그러나, ATmega1281과 ATmega2561에서 이처럼 프로그램을 다르게 작성하는 것은 때로 불편할 수 있으므로 실제로 "AVR ATmega1281/2561 정복" 책의 예제 프로그램에서는 다음과 같이 조건부 어셈블리 지시어를 사용하여 동일한 소스를 사용하도록 처리하였다.

```
;-----------------------------------------------
;       Display String on LCD
;-----------------------------------------------
LCD_STRING:OUT  GPIOR0,ZL               ; store RAMPZ:ZH:ZL
        OUT     GPIOR1,ZH
        IN      ZL,RAMPZ
        OUT     GPIOR2,ZL
#if(defined(  ATmega2560  ) || defined(  ATmega2561  ))
        POP     RAMPZ_BUFFER            ; get start address of string(3-byte PC)
#else
        CLR     RAMPZ_BUFFER            ; get start address of string(2-byte PC)
#endif
        POP     ZH
        POP     ZL
        LSL     ZL                      ; convert into byte address
        ROL     ZH
        ROL     RAMPZ_BUFFER
```

```
        OUT     RAMPZ,RAMPZ_BUFFER
LCD_STR1:ELPM   LCD_BUFFER,Z+           ; read a character, end string ?
        CPI     LCD_BUFFER,0
        BREQ    LCD_STR2                ; if yes, return
        CALL    LCD_DATA                ; if no, display a character
        RJMP    LCD_STR1
LCD_STR2:SBRC   ZL,0                    ; if odd address, skip 1 address
        ELPM    LCD_BUFFER,Z+
        IN      RAMPZ_BUFFER,RAMPZ      ; convert into word address
        LSR     RAMPZ_BUFFER
        ROR     ZH
        ROR     ZL
        PUSH    ZL                      ; store new return address
        PUSH    ZH
#if(defined(  ATmega2560  ) || defined(__ATmega2561__))
        PUSH    RAMPZ_BUFFER
#endif
        IN      ZL,GPIOR2               ; restore RAMPZ:ZH:ZL
        OUT     RAMPZ,ZL
        IN      ZH,GPIOR1
        IN      ZL,GPIOR0
        RET
```

이와 같이 어셈블리 언어 프로그램에서 이러한 특별한 서브루틴을 사용하는 프로그램은 ATmega1281과 ATmega2561에서 서로 소프트웨어의 호환성이 없게 된다. 그러나, "AVR ATmega1281/2561 정복" 책에 있는 대부분의 어셈블리 언어 예제에서는 이 서브루틴들을 사용하고 있으므로 이 2가지 모델 사이에는 사실상 소프트웨어 호환성이 없다고 보아야 한다.

(2) WinAVR의 AVR-GCC 컴파일러를 사용하는 경우

WinAVR에서 함수를 호출할 때는 인수를 전달할 때 레지스터 R25~R8이 사용된다. 이것들은 인수에 따라 사용방법이 약간 달라지는데, char형의 인수는 짝수 번호의 레지스터만 사용하고, int형의 인수는 2개의 레지스터를 사용하며, long형의 인수는 4개의 레지스터를 사용한다. 따라서, 인수에 사용되는 레지스터들을 모두 정리하면 다음과 같다.

char형 인수(9개까지 사용 가능) : R24, R22, R20, R18, R16, R14, R12, R10, R8
int형 인수(9개까지 사용 가능) : R25:R24, R23:R22, R21:R20, R19:R18, R17:R16, R15:R14, R13:R12, R11:R10, R9:R8
long형 인수(4개까지 사용 가능) : R25:R24:R23:R22, R21:R20:R19:R18, R17:R16:R15:R14, R13:R12:R11:R10

만약, 하나의 함수에서 사용하는 모든 인수들이 이처럼 1가지의 데이터 형으로 통일되어 있지 않고 여러가지의 데이터 형을 사용하고 있더라도 이 규칙에 따라 R25에서 R8까지 차례로 해당 레지스터들을 할당하여 사용하게 된다. 그러나, 하나의 함수에서 사용하는 인수가

많아져서 레지스터로 이를 모두 전달하지 못하게 되면 레지스터를 더 이상 사용할 수 없는 뒤쪽의 인수들은 스택을 통하여 전달된다.

그런데, 함수 내부에서 이 인수들을 꺼내어 사용하려면 문제가 발생하게 된다. 레지스터 R25~R8을 통하여 전달된 인수를 사용하는 것은 간단하지만, 스택을 통하여 전달된 인수를 사용하려면 스택 영역을 액세스해야 하는데, 스택에는 먼저 인수들이 차례로 저장되고 마지막에 복귀주소인 프로그램 카운터가 저장되어 있게 된다. ATmega1281에서는 2바이트의 프로그램 카운터가 저장되므로 현재 스택 포인터가 가리키는 위치에서 2번지만큼 위에 있는 영역에 인수들이 위치하지만, ATmega2561에서는 3바이트의 프로그램 카운터가 저장되므로 현재 스택 포인터가 가리키는 위치에서 3번지만큼 위에 있는 영역에 인수들이 위치하게 되므로 이를 액세스하는 프로그램은 서로 다르게 컴파일되어야 하며, 따라서 이 2모델 사이에는 소프트웨어의 호환성이 없게 된다.

이와 같이 C언어 프로그램에서 인수가 매우 많아서 일부의 인수를 스택으로 전달해야 하는 함수를 사용하는 프로그램은 ATmega1281과 ATmega2561에서 서로 소프트웨어의 호환성이 없게 된다. 그러나, "AVR ATmega1281/2561 정복" 책에 있는 모든 C언어 예제에서는 함수가 이렇게 많은 인수를 사용하지 않으므로 이 2가지 모델 사이에는 사실상 소프트웨어 호환성이 있다고 보아야 한다. 그러므로 현재 WinAVR은 아직 ATmega2561을 지원하지 않지만 ATmega1281용으로 컴파일된 실행파일을 그냥 ATmega2561에서 실행하면 되는 것이다.

그러나, "AVR ATmega1281/2561 정복" 책의 제4.20절에서 printf() 함수를 사용하는 C언어 예제 프로그램만은 ATmega1281과 ATmega2561에서 소프트웨어의 호환성이 없다. 이 함수를 지원하는 라이브러리 파일에서는 스택 영역에 저장된 내용을 액세스하여 사용하기 때문이다. 즉, ATmega1281용으로 컴파일된 실행파일은 ATmega1281에서만 정상적으로 실행되고, ATmega2561용으로 컴파일된 실행파일은 ATmega2561에서만 정상적으로 실행된다는 것이다. 그러나, 현재 WinAVR의 20060421 버전은 아직 ATmega2560/2561 모델을 지원하지 않으므로 여기서 printf() 함수를 사용하려면 이를 지원하는 새로운 버전의 WinAVR이 나올 때까지 기다려야 할 것이다.

【 참고문헌 】

1. 윤덕용, 고성능 AVR 정복 시리즈② - AVR ATmega1281/2561 정복, 도서출판 Ohm사, 2006 출간 예정.
2. 윤덕용, 고성능 AVR 정복 시리즈③ - AVR ATmega1280/2560 정복, 도서출판 Ohm사, 2006 출간 예정.