Appendix I: Passing Arguments into Functions

Sepehr Naimi

www.NicerLand.com

There are different ways to pass arguments (parameters) to functions. Some of them are:

- through registers
- through memory using references
- using stack

I.1: Passing arguments through registers

In the following program the BIGGER function gets two values through R21 and R22. After comparing R21 and R22, it returns the bigger value through R21.

Program I-1

```
       LDI    R16,HIGH(RAMEND)     ; SP = RAMEND
       OUT    SPH,R16
       LDI    R16,LOW(RAMEND)
       OUT SPL,R16

       LDI    R21,5  ; R21 = 5
       LDI    R22,7  ; R22 = 7
       CALL   BIGGER
HERE:
    RJMP HERE

       ; =======================================
       ; bigger returns the bigger value
       ; parameters:
       ;      R21 and R22: the values to be compared
       ; returns:
       ;      R21: containing the bigger value
       ; =======================================
BIGGER:
       CP     R21, R22
       BRSH   L1
       MOV    R21, R22
L1:    RET
```

This is a fast way of passing arguments to the function.

I.2: Passing through memory using references

We can store the data in memory and pass its address through a register. In the following program, the address of a string is passed to the function through the Z register. The string ends with 0. The function puts the contents of the string on PORTB until it reaches 0.

Program I-2

```
       ;Initializing the stack pointer
       LDI    R16,LOW(RAMEND)
       OUT    SPL,R16
       LDI    R16,HIGH(RAMEND)
       OUT    SPH,R16
```

```
        ;Z = addr. of MYDATA
        LDI     ZL,LOW(MYDATA<<1)
        LDI     ZH,HIGH(MYDATA<<1)
        CALL    MY_FUNC
HERE: RJMP HERE

MY_FUNC:
        LPM     R20,Z+
        CPI     R20,0          ;is 0 ?
        BREQ    L_END          ;return if it is 0
        OUT     PORTB,R20
        RJMP    MY_FUNC
L_END: RET             ;return

MYDATA: .DB "Hello World",0        ;a zero ended string
```

In the following program the STR_LENGTH function gets the address of a zero-ended string through Z and returns the length of the string through R20.

Program I-3

```
        ; =======================================
        ; STR_LENGTH returns the length of string
        ; parameters:
        ;      Z: address of the string
        ; returns:
        ;      r0: the length of string
        ; =======================================
STR_LENGTH:
        LDI     R20, 0         ; use R20 as string length counter
L_BEGIN:
        LD      R20, [Z+]      ; fetch a character from string
        CPI     R20, 0
        BREQ    L_END          ; return if character is null (end of string)
        INC     R20            ; increment the counter
        JMP     L_BEGIN
L_END: RET
```

I.3: Passing arguments through stack

Passing through the stack is a flexible way of passing arguments. To do so, the arguments are pushed onto the stack just before calling the function and popped off after returning.

This method of passing arguments is used in x86 computers because they have very few general purpose registers. In AVR CPU, the arguments are passed in registers. If the arguments are more than registers, the rest are passed on the stack.

It is important to remember that after returning from the call, the caller must clear the arguments on the stack.

I.4: AVR Procedure Call Standard

AVR provides a standard for implementing the functions and the function calls so that the codes made by different compilers and different programmers can work with each other. Some of the rules of the standard are:

- The arguments must be sent through R25 to R8. The left most argument goes to R25 and R24, and the next arguments go respectively to the next registers.
- An even number of registers are set aside for each argument. So, for passing a character argument, two registers are used. To pass an int argument, two registers are also used, and 4 registers are used to pass a long argument.
- The return value must be returned in R25 through R18 depending on the size of return value.
- The functions can use R2 to R17, R28 and R29. But their values must be saved upon entering the function and restored before returning. To do so, we push the registers before using them and pop them before returning from the function.
- The functions can freely use R18 to R27, R30, and R31. Functions have no responsibilities about the values of these registers. So, if there is a valuable value in each of these registers, they must be saved before calling a function, and restored after returning. To do so, we push the registers before calling the functions and pop them after returning from the function.
- The AVR C compilers assume that R1 contains 0. So, if you change the value of R1, you must clear R1 when your code ends.
- The AVR compilers use R0 as a temporary register. So, if the value of R0 is changed in an assembly code, R0 must be saved before changing and then restored afterward.

Register	Function	The caller
R0	Save and restore if using	Save and restore if using
R1	Must clear before returning	Must clear before calling
R2-R17, R28, R29	Save and restore if using	Can freely use
R18-R27, R0, R31	Can freely use	Save and restore if using

Table I-1: Summery of the register interfaces between C and Assembly (Copied from Atmel AT1886)

In Program I-4 the above rules are considered. The delay_ms function gets a byte argument through R24 (R24 and R25 are set aside to be even). It saves and restores R16 and R17 using stack.

Program I-4

```
LDI     R16,HIGH(RAMEND)
OUT     SPH,R16
LDI     R16,LOW(RAMEND)
OUT     SPL,R16
```

```
       LDI    R24, 30
       CALL DELAY_MS        ;WAIT 30 ms

HERE:  RJMP   HERE

       ; ========================================
       ; DELAY_MS waits for a few milliseconds
       ; parameters:
       ;      r0: the amount of wait in milliseconds
       ; returns:
       ;      none
       ; ========================================

DELAY_MS:
       PUSH R16      ;save R16
       PUSH R17      ;save R17

D_L0:  LDI R17, 100
D_L1:  LDI    R16, 40
D_L2:  NOP
       DEC R16
       BRNE D_L2
       DEC R17
       BRNE D_L1
       DEC    R24
       BRNE D_L0

       POP R17       ;restore R17
       POP R16       ;restore R16
       RET
```