EC-310 Microprocessor and Microcontroller Based Design

Chapter - 6

Macros and Modules

Nazar Abbas Saqib

nazar.abbas@ceme.nust.edu.pk

Outline

1. Checksum Calculation and verification
2. Macros
3. Modules

Checksum and ASCII Subroutines

- **Checksum byte in ROM**
- To ensure the integrity of ROM contents, every system must perform a checksum calculation.
- The checksum detects any corruption of the contents of ROM.
- To ensure data integrity in ROM, the checksum process uses checksum byte.
- Checksum byte is an extra byte that is tagged to the end of series of bytes of data.

Checksum and ASCII Subroutines

- **Checksum byte in ROM**
- **To calculate checksum** byte of a series of bytes of data, following steps are performed:
  - Add bytes together and drop the carries.
  - Take the 2's complement of the total sum and that is the checksum byte, which becomes the last byte of the series.
- **To perform checksum byte** add all the bytes together. The result must be zero.
- If result is not zero, than some bytes are corrupted.

Checksum and ASCII Subroutines

Example 6-29

Assume that we have 4 bytes of hexadecimal data: 25H, 62H, 3FH, and 52H.

(a) Find the checksum byte.
(b) Perform the checksum operation to ensure data integrity.
(c) If the second byte, 62H, has been changed to 22H, show how the checksum method detects the error.

Solution:

(a) Find the checksum byte.

```
     25H
+    62H
+    3FH
+    52H
    118H
```

(Dropping the carry of 1, we have 18H. Its 2's complement is E8H. Therefore checksum byte is E8H.)

Checksum and ASCII Subroutines

Example 6-29

Assume that we have 4 bytes of hexadecimal data: 25H, 62H, 3FH, and 52H.

(a) Find the checksum byte.
(b) Perform the checksum operation to ensure data integrity.
(c) If the second byte, 62H, has been changed to 22H, show how the checksum method detects the error.

(b) Perform the checksum operation to ensure data integrity.

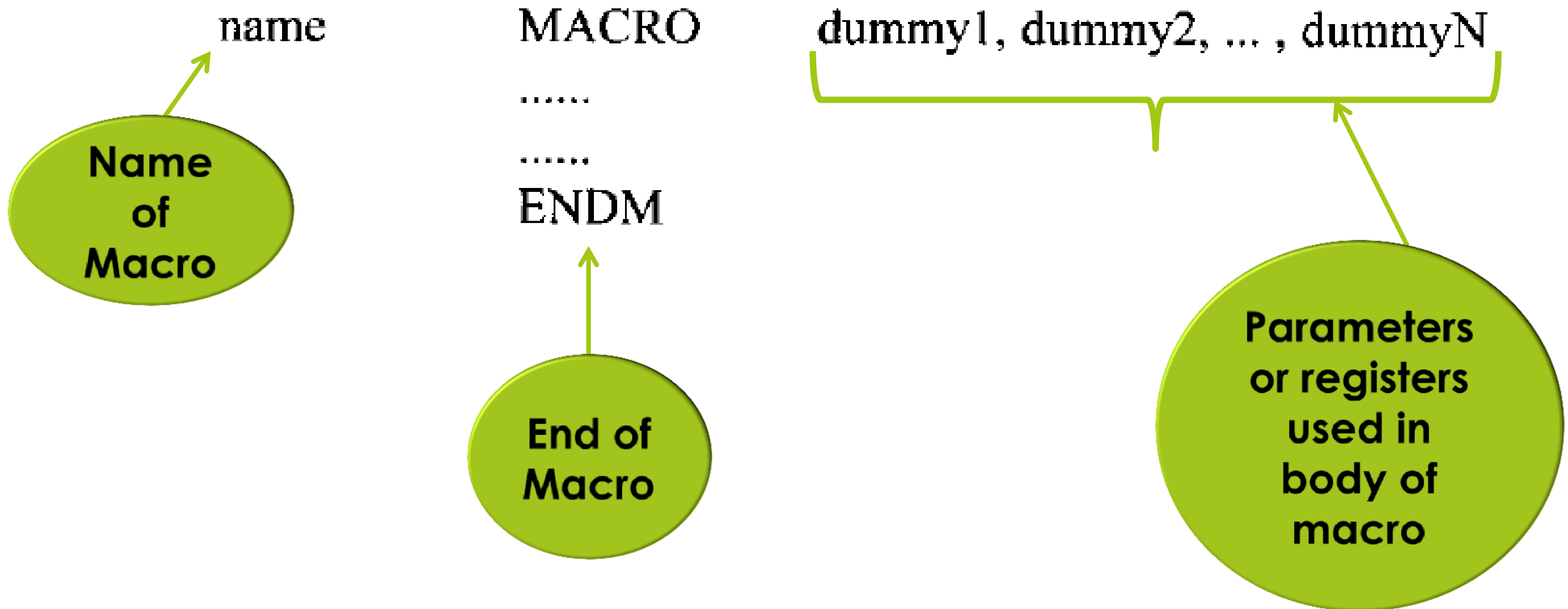
```
     25H
  +  62H
  +  3FH
  +  52H
  +  E8H
  ------
    200H    (Dropping the carries, we see 00, indicating that data is not corrupted.)
```

Macros and Modules

- **Modules:**
- Modules in assembly language are same as functions in C programming.
- It is a common practice to divide a program into several modules, test each module and put them into a library.
- **Macros:**
- There are applications where a group of instructions performs a task that is used repeatedly.
- It does not make sense to rewrite this code every time it is needed.
- To reduce the time that it takes to write code and reduce the possibility of errors, the concept of macros was born.

MACRO definition

Every macro definition have 3 parts:



Macros and Modules

- Example: Macro to move a literal value into a RAM location.

```
MOVLF MACRO K, MYREG
      MOVLW K
      MOVWF MYREG
      ENDM
```

- Three ways to use the macro MOVLF

```
1. MOVLF    0x55, 0x20         ;send value 55H to loc 20H
```

```
2. VAL_1    EQU 0x55
   RAM_LOC  EQU 0x20
   MOVLF    VAL_1, RAM_LOC
```

```
3. MOVLF    0x55, PORTB        ;send value 55H to Port B
```

Macro – Rules

- The following rules must be observed in the body of the macro:
  - All labels in the label field must be declared LOCAL.
  - The local directive must be right after MACRO directive.
  - The local directive can be used to declare all names and labels at once or one at a time

```
            LOCAL       name1, name2, name3
or one at a time as:
            LOCAL       name1
            LOCAL       name2
            LOCAL       name3
```

Macro for Time Delay

To clarify these points, look at the following macro for time delay:

```
DELAY_1 MACRO V1, TREG
        LOCAL BACK
        MOVLW V1
        MOVWF TREG
BACK    NOP
        NOP
        NOP
        NOP
        DECF  TREG,F
        BNZ   BACK
        ENDM
```

"BACK" label is defined as LOCAL right after the MACRO directive.

Defining this label anywhere else causes error.

The local directive allows the assembler to define the labels separately each time it counters them.

Macro for Time Delay with Nested Loop

```
DELAY_2 MACRO V1, V2, R1, R2
        LOCAL BACK
        LOCAL AGAIN
        MOVLW V2
        MOVWF R2
AGAIN   MOVLW V1
        MOVWF R1
BACK    NOP
        NOP
        NOP
        NOP
        DECF  R1,F
        BNZ   BACK
        DECF  R2,F
        BNZ   AGAIN
        ENDM
```

Program 6-4, Using Macros in a Program

```
;-----------------------------------------------------------
;Program 6-4: toggling Port B using macros
        #include P18F458.INC

;----------------sending data to fileReg macro
MOVLF   MACRO K, MYREG
        MOVLW K
        MOVWF MYREG
        ENDM

;------------------------------------time delay macro
DELAY_1 MACRO V1, TREG
        LOCAL BACK
        MOVLW V1
        MOVWF TREG
BACK    NOP
        NOP
        NOP
        NOP
        DECF  TREG,F
        BNZ   BACK
        ENDM
```

Program 6-4 (continued)

```
;-------------------------------program starts
        ORG     0
        CLRF      TRISB                  ;Port B as an output
OVER    MOVLF     0x55,PORTB
        DELAY_1   0x200,0x10
        MOVLF     0xAA,PORTB
        DELAY_1   0x200,0x10
        BRA       OVER
        END
;--------------------end of file
```

Main Program

MOVLF MACRO

DELAY_1 MACRO

INCLUDE Directive

- **Assume that several macros are used in every program. For this, each macro must be written at least once in every program.**

- **This issue can be resolved using INCLUDE directive.**

- **INCLUDE directive allows the programmer to write a macro and save it in a file. Later this macro can be used in every program by using INCLUDE directive.**

- **Macros are saved in file with extension .mac.**

mymicro1.mac

```
;----------------sending data to fileReg macro
MOVLF MACRO K, MYREG
      MOVLW K
      MOVWF MYREG
      ENDM

;------------------------------time delay macro
DELAY_1 MACRO V1, TREG
      LOCAL BACK
      MOVLW V1
      MOVWF TREG
BACK  NOP
      NOP
      NOP
      NOP
      DECF  TREG,F
      BNZ   BACK
      ENDM
```

INCLUDE Directive

Macros are stored in this file

```
;Program 6-5: toggling Port B using macros
        #include P18F458.INC
        #include "MYMACRO1.MAC" ;get macros from macro file

;----------------------------program starts
        ORG     0
        CLRF    TRISB                     ;Port B as an output
OVER    MOVLF   0x55,PORTB
        DELAY_1         0x200,0x10
        MOVLF   0xAA,PORTB
        DELAY_1         0x200,0x10
        BRA     OVER
        END
;---------------------end of file
```

Both MOVLF and DEALY_1 are macros

NOEXPAND/EXPAND Directive

- **EXPAND Directive:**
- Expand directive is used to turn on the macro display in list file. By default macros are fully displayed.

- **NOEXPAND Directive:**
- For limited iterations, macro expansion is ok. But for greater number of iterations, this becomes cumbersome.
- Using noexpand directive, we can turn off the display of macros in the list file.

NOEXPAND/EXPAND Directive

```
                  00001 ;Program 6-4:toggling Port B using macros
                  00002        #include P18F458.INC
                  00003        NOEXPAND
                  00004 ;---------------sending data to fileReg macro
                  00005 MOVLF MACRO K, MYREG
                  00006        MOVLW    K
                  00007        MOVWF    MYREG
                  00008        ENDM
                  00009
                  00010 ;----------------------------time delay macro
                  00011 DELAY_1 MACRO V1, TREG
                  00012        LOCAL        BACK
                  00013        MOVLW        V1
                  00014        MOVWF        TREG
                  00015 BACK   NOP
                  00016        NOP
                  00017        NOP
                  00018        NOP
                  00019        DECF         TREG,F
                  00020        BNZ          BACK
                  00021        ENDM
                  00022
                  00023 ;----------------------------program starts
000000            00024        ORG          0
000000 6A93       00025        CLRF         TRISB        ;Port B as an output
                  00026 OVER   MOVLF        0x55,PORTB
                  00027        DELAY_1      0x200,0x10
                  00028        MOVLF        0xAA,PORTB
                  00029        DELAY_1      0x200,0x10
00002A D7EB       00030        BRA          OVER
                  00031        END
```

Figure 6-11. List File with NOEXPAND Option for Program 6-4

NOEXPAND/EXPAND Directive

```
00001 ;Program 6-4:toggling Port B using
00002         #include P18F458.INC
00003         EXPAND
00004 ;------------------sending data to fil
00005 MOVLF MACRO K, MYREG
00006         MOVLW K
00007         MOVWF MYREG
00008         ENDM
00009
00010 ;-------------------------------time de
00011 DELAY_1 MACRO V1, TREG
00012         LOCAL BACK
00013         MOVLW V1
00014         MOVWF TREG
00015 BACK    NOP
00016         NOP
00017         NOP
00018         NOP
00019         DECF   TREG,F
00020         BNZ    BACK
00021         ENDM
00022
```

```
                 00023 ;---------------------------program starts
000000           00024         ORG    0
000000 6A93      00025         CLRF   TRISB ;Port B as an output
                 00026 OVER    MOVLF  0x55,PORTB
000002 0E55          M         MOVLW  0x55
000004 6E81          M         MOVWF  PORTB
                 00027 DELAY_1 0x200,0x10
  0000               M         LOCAL  BACK
000006 0E00          M         MOVLW  0x200
000008 6E10          M         MOVWF  0x10
00000A 0000          M BACK    NOP
00000C 0000          M         NOP
00000E 0000          M         NOP
000010 0000          M         NOP
000012 0610          M         DECF   0x10,F
000014 E1FA          M         BNZ    BACK
                 00028         MOVLF  0xAA,PORTB
000016 0EAA          M         MOVLW  0xAA
000018 6E81          M         MOVWF  PORTB
                 00029 DELAY_1 0x200,0x10
  0000               M         LOCAL  BACK
00001A 0E00          M         MOVLW  0x200
00001C 6E10          M         MOVWF  0x10
00001E 0000          M BACK    NOP
000020 0000          M         NOP
000022 0000          M         NOP
000024 0000          M         NOP
000026 0610          M         DECF   0x10,F
000028 E1FA          M         BNZ    BACK
00002A D7EB      00030         BRA    OVER
                 00031         END
```

Figure 6-12. List File with EXPAND Option for Program 6-4

Macros vs Subroutines

- Macros increase code size every time they are invoked.
  - For example, if you call a 10-instruction macro 10 times, the code size is increased 100 instructions
- Whereas, if same subroutine is called 10 times, the code size is only the size of the subroutine instructions.
- The problem with subroutine is that they use stack space when called and this can cause problem in case of nested calls.
- The nested calls can lead to a stack overflow and cause the program to crash.

Modules

- It is a common practice to break the software into small modules and distribute the task of writing those modules among several programmers.
- Advantages:
  - Each module can be written, debugged and tested individually.
  - The failure of one module does not stop the entire project.
  - The task of locating and isolating any problem is easier and less time consuming.
  - One can use the modules to link with high level languages such as C.
  - Parallel development shortens considerably the time required to complete a project.

Modules

- In previous examples, so far, subroutines have been called in main program to perform the task.
- If one of the subroutine did not work properly, the entire program would have to be rewritten and reassembled.
- A more efficient way to develop software is to **treat each subroutine as a separate program or module with separate file name.**
- Each file can be assembled and tested independently.
- After testing each program, they all can be brought together to make a single program.
- To enable these modules to be linked together, certain assembly language directives must be used.

Extern Directive

- The EXTERN directive is used to notify the assembler and linker that certain name and variables are not defined in the present module
- These names and variables are defined externally somewhere else
- In the absence of EXTERN directive, assembled would generate error

```
GLOBAL name1                ;each name can be in a separate directive
GLOBAL name2

GLOBAL name1, name2         ;or many can be listed in the same GLOBAL
```

Program 6-6

```
;-----------------------------------------------------
;PROG 6-6: MAIN.ASM - CALCULATING AND TESTING CHECKSUM BYTE
      #include P18F458.INC

RAM_ADDR      EQU 40H
COUNTREG      EQU 0x20           ;fileReg loc for counter
CNTVAL        EQU 4              ;counter value
CNTVAL1       EQU 5              ;counter value

      EXTERN CAL_CHKSUM
      EXTERN TEST_CHKSUM

PGM CODE
;------------main program
      ORG    0
      CALL   COPY_DATA           ;this subroutine is in this file
      CALL   CAL_CHKSUM          ;this sub is in external file
      CALL   TEST_CHKSUM         ;this sub is in external file
      BRA    $
```

External

Local

By using CODE directive, we want to write instructions to be placed in the program memory. Label of CODE here (PGM) is used group instructions from one program to other

Program 6-6 (continued)

```
;--------copying data from code ROM to data RAM
COPY_DATA
      MOVLW low(MYBYTE)        ;WREG = 00 LOW-byte addr.
      MOVWF TBLPTRL            ;ROM data LOW-byte addr.
      MOVLW hi(MYBYTE)         ;WREG = 5, HIGH-byte addr.
      MOVWF TBLPTRH            ;ROM data HIGH-byte addr.
      MOVLW upper(MYBYTE)      ;WREG = 00 upper-byte addr.
      MOVWF TBLPRTRU           ;ROM data upper-byte addr.
      LFSR  0,RAM_ADDR         ;FSR0 = RAM_ADDR, place to save
C1    TBLRD*+                  ;bring in next byte and inc TBLPTR
      MOVF  TABLAT,W           ;copy to WREG (Z = 1, if null)
      BZ    EXIT               ;is it null char? exit if yes
      MOVWF POSTINC0           ;copy WREG to RAM and inc pointer
      BRA   C1
EXIT  RETURN

;----------my data in program ROM
      ORG 0x500
MYBYTE DB 0x25, 0x62, 0x3F, 0x52, 0x00
      END
```

Program 6-6 (continued)

```
;-----------------------------------------------------------
;PROG 6-6: CALCCSB.ASM - CALCULATING CHECKSUM BYTE
        #include P18F458.inc

RAM_ADDR        EQU 40H
COUNTREG        EQU 0x20              ;fileReg loc for counter
CNTVAL          EQU 4                 ;counter value
CNTVAL1         EQU 5                 ;counter value

        GLOBAL CAL_CHKSUM

PGM CODE                        ;we use this to inform the linker that
                                ;the code segment has the name PGM
CAL_CHKSUM
        MOVLW  CNTVAL                 ;WREG = 4
        MOVWF  COUNTREG               ;load the counter
        LFSR   0,RAM_ADDR             ;load pointer. FSR0 = 40H
        CLRF   WREG
C2      ADDWF  POSTINC0,W             ;add RAM to WREG and increment FSR0
        DECF   COUNTREG,F             ;decrement counter
        BNZ    C2                     ;loop until counter = zero
        XORLW  0xFF                   ;1's comp
        ADDLW  1                      ;2'compl
        MOVWF  POSTINC0
        RETURN
        END
```

Program 6-6 (continued)

```
;------------------------------------------------
;PROG 6-6: TESTCSB.ASM - TESTING CHECKSUM BYTE
      #include P18F458.inc

RAM_ADDR      EQU 40H
COUNTREG      EQU 0x20              ;fileReg loc for counter
CNTVAL        EQU 4                 ;counter value
CNTVAL1       EQU 5                 ;counter value

       GLOBAL TEST_CHKSUM
       PGM CODE
 TEST_CHKSUM
       MOVLW CNTVAL1                ;WREG = 5
       MOVWF COUNTREG               ;load the counter
       CLRF  TRISB
       LFSR  0,0x40                 ;load pointer. FSR0 = 40H
       CLRF  WREG
 C3    ADDWF POSTINC0,W             ;add RAM and increment FSR0
       DECF  COUNTREG,F             ;decrement counter
       BNZ   C3                     ;loop until counter = zero
       XORLW 0x0                    ;EX-OR to see if zero
       BZ    G_1                    ;is result zero? then good
       MOVLW 'B'
       MOVWF PORTB                  ;if not, data is bad
       RETURN
 G_1   MOVLW 'G'
       MOVWF PORTB                  ;data is not corrupted
       RETURN
       END
```

Linking Programs

Assuming that each program module in Program 6-6 is assembled separately and saved under the filenames MAIN.O, CALCCSB.O, and TESTCSB.O, the following shows how to link them together with MPLINK in order to generate a single executable file:

```
> MPLink.exe" "18f458.lkr" "MAIN.O" "CALCCSB.O" "TESTCSB.O" /o"PRG6-6.COF"
```