LABORATORY WORK NO. 8
WORKING WITH MACROS AND LIBRARIES

1. Object of laboratory

Getting used to defining and using macros, procedure defining and using LIB library librarian.

2. Theoretical considerations

2.1. Working with macros

The macros, procedures and libraries are the programmer tools, which allow the call and the using of previously written and debugged code.

The macros are facilities for assembly language programmers. A macro is a pseudo-operation that allows repeated including of code in the program. The macro, once defined, his call by name allows his insertion any time is needed. When meeting a macro name, the assembler **expands** his name in corresponding code of the macro body. For this reason, it is said the macros are executed in-line because the sequential execution flow of the program in not interrupted.

Macros can be created as a part of user program or grouped into another file as a macro library. A macro library is a usual file, which contains a series of macros and which is referred during program assembly, at the first pass of the assembler over the source program. It has to be specified that a macro library contain unassembled source lines. Because of that, macro libraries have to be included in the user source program using the INCLUDE pseudo-instruction – see Annex 12 example. This is the major difference between the macros library and a procedure library in object code that contains assembled procedures as object code and which is referred to the link-edit.

Firms offer this kind of macro libraries, for example DOS.INC and BIOS.INC by IBM.

For defining a macro it is used the sequence beneath:

```
name    MACRO       {macro parameters}
        LOCAL       local label list of the macro
                    these are expanded with different names at the
                    repeated call of the macro
                    {macro body}
        ENDM
```

Example:

```
INTIR MACRO     TIME
      LOCAL             P1,P2       ;p1 and p2 are local labels
      PUSH      DX                  ;saves the dx and cx registers
      PUSH      CX                  ;cx
      MOV       DX, TIME            ;loads a delay in dx
P1:   MOV       CX, 0FF00H          ;loads cx with 0FF00H
                                      ;counts
P2:   DEC       CX                  ;delays decrementing cx
      JNZ       P2                  ;if cx!=0 continue
      DEC       DX                  ;if cx=0 decrements dx
      JNZ       P1                  ;if dx!=0 loads again cx
      POP       CX                  ;if dx=0 remake cx
      POP       DX                  ;and dx
      ENDM                          ;end macrou
```

P1 and P2 are the local labels of the macro.

2.2. Pre-defined macros

TASM recognizes pre-defined macros. Those are IRP, IRPC and REPT. They are used for repeated defining.

Example:

```
        IRP     VAL, <2,4,6,8,10>
        DB      VAL
        DB      VAL*2
        ENDM
```

In some cases, the formal parameter substitution with actual parameters creates some problems. Let’s follow the macroinstruction, which suggests interchanging two 16 bites quantities.

```
TRANS  MACRO  X, Y
        PUSH AX
        PUSH BX
        MOV  BX, X
        MOV  AX, Y
        MOV  X, AX
        MOV  Y, BX
        POP   BX
        POP   AX
ENDM
```

Apparently, every thing is ok. However, unexpected situation can appear, like in following sequence:

```
        TRANS       AX, SI              ;interchange ax with SI
```

This referred macroinstruction will be expanded in:

```
        PUSH  AX
        PUSH  BX
        MOV   BX, AX
        MOV   AX, SI
        MOV   AX, AX
        MOV   SI, BX
        POP    BX
        POP    AX
```

and it is obviously the AX register is not modifying. Worst thing can happen, like beneath:

```
        TRANS       SP, DI              ;interchange SP with DI
```

which is expanded in:

```
        PUSH  AX
        PUSH  BX
        MOV   BX, SP
        MOV   AX, DI
        MOV   SP, AX                    ;SP is modified here
        MOV   DI, BX                    ;POPs are compromised
        POP    BX
        POP    AX
```

Danger appears, therefore, in situation in which actual parameters are conflicting with some variables or registers being used in the macroinstruction. Situations like this must be avoided.

2.3. UsingTLIB librarian

The syntax for launching TLIB librarian is:

TLIB *library_name [/C] [/E] [/P] [/O] command, listing_file_name*

where:

- *library_name* represents the path and the library file name
- *command* represents commands sequence that will be executed on the library
- *listing_file_name* represents the path and the name of the file in which you want the crossed references to be generated for PUBLIC symbols and for the library modules names. The listing is generated after the processing in the library is finished.

A command is like:

<symbol> module_name

where <*symbol*> represents:

+	:	adds module_name to the library
-	:	deletes module_name from the library
*	:	extracts module_name from the library without deleting it
-+ or +-	:	replaces module_name in the library
-* or *-	:	extracts module_name from the library and deletes module_name from the library
/C	:	case-senzitive library
/E	:	creates extended dictionary
/P size	:	sets library page dimension to size

For moving to the next line, use ‘&’ character.

2.4. Examples of programs that are using macros and libraries

2.4.1. Program EXEMMAC.ASM

```
;PROGRAM EXAMPLE FOR USING A SIMPLE MACRO

TITLE Program with macro call

STACK SEGMENT PARA 'STACK'
        DB              64 DUP ('STACK')
STACK   ENDS

DATA SEGMENT    PARA 'DATA'
TAMP DB         2000 DUP (' ')
DATA    ENDS

INTIR MACRO     TIME
        LOCAL   P1, P2  ;;p1 and p2 are local labels
        PUSH  DX        ;;saves dx and cx registers
        PUSH  CX
        MOV   DX, TIME          ;; loads a delay in dx
P1:     MOV   CX, 0FF00H  ;;loads cx with 0FF00h
                        ;;counts
P2:     DEC   CX                ;;delays by decrementing cx
        JNZ   P2                ;;if cx!=0 continue
        DEC   DX                ;;if cx=0 decrements dx
        JNZ   P1                ;;if dx!=0 loads again cx
        POP   CX                ;;if dx=0 remake cx and dx
        POP   DX                ;;
        ENDM                    ;;end macro

MYCOD SEGMENT PARA 'CODE'       ;defines code segment
PROCED  PROC    FAR             ;procedure with proced name
        ASSUME  CS:MYCOD, ES:DATA, DS:DATA, SS:STACK
        PUSH  DS
        XOR AX,AX
        PUSH AX
        MOV   AX, DATA          ;puts data segment in ax
        MOV   ES, AX
```

```
        MOV   DS,AX                     ;loads es with data segment
;program will clear the display writing 25*80 spaces on the screen
;writing those with different values in bl the screen color will change
;intir macro will maintain this color for a time
        MOV   CX, 08H                   ;loops 8 times
        MOV   BL, 00H                   ;sets background color
LOOP1:  LEA     BP, TAMP                ;writes black string
        MOV   DX, 0000H                 ;sets the cursor to the upper
                                        :left
        MOV   AH, 19                ;writes attribute string
        MOV   AL, 1                 ;writes a character and moves
                                        ;the cursor
        PUSH CX                         ;saves cx
        MOV   CX, 07D0H                 ;writes 2000 spaces
        INT     10H                     ;call 10h
        INTIR 10000                 ;delays 10 units
        ADD   BL, 10H                   ;changes background color
        POP     CX                      ;restores  cx
        LOOP  LOOP1                     ;loops 8 times
        RET                             ;hands over the control to
                                        ;dos
PROCED      ENDP                        ;end procedure
MYCOD ENDS                              ;end code segment
        END   PROCED                    ;end program
```

2.4.2. Program EXBIMAC.ASM

```
TITLE Example of macro library using
IF1                                     ;includes a previously created
        INCLUDE C:\TASM\MLAB.MAC   ; macro library
; available on ftp.utcluj.ro/pub/users/cemil/asm/labs
ENDIF

STACK SEGMENT PARA 'STACK' ;defines a stack segment
        DB 64  DUP ('STACK')
STACK ENDS

SEGDATA SEGMENT PARA 'DATA' ;data segment definition
MESSAGE DB          'I am a simple counting program$'
TAMP  DB 2000 DUP (‘ ‘)
SEGDATA ENDS
```

```
COD1  SEGMENT PARA 'CODE'          ;code segment definition
MYPROC PROC      FAR               ;procedure with myproc name
ASSUME              CS:COD1, DS:SEGDATA, SS:STACK

        PUSH  DS                   ;saves ds
        SUB   AX, AX               ;0 in ax
        PUSH  AX                   ;0 on the stack
        MOV   AX, SEGDATA          ;adr segdata in ax
        MOV   DS, AX               ;adr segdata in ds
        DELETE                     ;clear screen macro call
        CURSOR 0019H               ;pos cursor macro call
        TYPECAR    MESSAGE         ;message type macro call
        MOV   AX, 00H              ;0 in ax for counting
REPEAT:     CURSOR      0C28H ;in middle of the screen
        TYPENUM                    ;number type macro call
        INTIR 1000          ;delay macro call
        ADD   AL, 01H              ;increment al
        DAA                        ;decimal adjustment
        CMP   AL, 50H              ;test final
        JE    SFIR                 ;after 9 executions
        JMP   REPEAT               ;else repeat
SFIR:   DELETE              ;clear screen macro call
        RET                        ;back to dos
MYPROC ENDP                        ;end procedure
COD1  ENDS                        ;end segment
        END   MYPROC               ;end program
```

2.4.3 Calling a procedure defined in a different source file

Main program:

```
;Program example for procedure use procedure defined in a different source
;file
TITLE Program with procedure call

STACK       SEGMENT    PARA 'STACK'
        DB      64 DUP ('STACK')
STACK       ENDS

DATA SEGMENT PARA `DATA`
TAMP DB     2000 DUP ('  ')
DATA        ENDS
```

```
COD1 SEGMENT PARA 'CODE'          ;code segment definition
PROCED    PROC FAR                ;procedure with proced name
        ASSUME    CS:COD1, ES: DATA, DS:DATA, SS:STACK
        EXTRN     INTIRP:NEAR ;extern declaration for INTIRP
                              ;procedure
        PUSH DS                   ;saves ds
        SUB  AX, AX               ;0 in ax
        PUSH AX                   ;puts 0 on the stack
        MOV  AX, DATA             ;puts seg data in ax
        MOV DS,AX
;main program
        MOV  AX, 100              ;parameter in ax
        CALL INTIRP               ;intirp procedure call
        RET                       ;gives the control to dos
PROCED    ENDP                    ;procedure end
COD1 ENDS                         ;code segment end
        END  PROCED               ;end program
;End of first source file _____________   _____   _______

;Start of second sourse file_____________________________
;called procedure
COD1 SEGMENT PARA 'CODE'    ;defines code segment
             PUBLIC INTIRP ;public declaration for INTIRP
                           ;procedure
        ASSUME      CS:COD1
INTIRP      PROC NEAR       ;intirp procedure name
        PUSH DX             ;saves dx şi cx registers
        PUSH CX             ;
        MOV  DX, AX         ;loads a delay in dx
P1:     MOV  CX, 0FF00H  ;loads 0FF00h in cx
                            ;counts
P1:     DEC  CX             ;delays decrementing cx
        JNZ  P2             ;if cx!=0 continue
        DEC  DX             ;if cx=0 decrements dx
        JNZ  P1             ;if dx!=0 loads again cx
        POP  CX             ;if dx=0 restore cx and
        POP  DX             ;dx
        RET                 ;return to the main procedure
```

```
INTIRP ENDP                    ;procedure end
COD1 ENDS
END
```

3. Lab tasks

1. Study the given example and exemmac.asm program.
2. Assemble this program with TASM and create EXEMMAC.LST file, study the way INTIR macro has been expanded.
3. Edit the links with LINK and execute exemmac.exe generated program.
4. Modify INTIR macro TIME parameter with different values with an edit program and repeat the steps from 1 to 3.
5. Study the case in which the macro is written into a separate file and it is included with INCLUDE directive (see previously example); notice the difference from a module included before compilation (with INCLUDE), a macro (which is similar) and a library (which contains compiled modules) – point out the similarity with .h files from C which are being compiled in the same time with the program.
6. Study the example of using a macro library MLIB.MAC in the exbimac.asm program.
7. Study the expand mode of PUSHALL and POPALL macros in TASM created listing of the program from the step 5.
8. Edit the links with TLINK program and execute EXBIMAC.EXE program.
9. Write a procedure with the same function as INTIR macro with INTIRP name. Include this procedure into a library with BIBLIO.LIB name. TIME parameter will be passed to the procedure in AX register.
10. Modify exmmac.asm to axmlib.asm and replace macros with procedure calls to INTIRP procedure, which initially has been included in BIBLIO.LIB.
11. Trace the program from steps 3 and 10 and follow the differences of generated code and change in instruction flow.