Lab work no. 9

Programs with multiple segments

Object of laboratory

Procedure definition, procedure call from the same segment and from different segments; working with programs written in more, separately assembled modules.

Theoretical considerations

Procedures may be defined as FAR or NEAR type. The procedure's type determines the way in which the call is made and the information that is saved on the stack at calling.

When calling a NEAR type procedure, IP register is saved on the stack. CS register remains unmodified and is not saved on the stack. This implies that the two procedures, the called one and the one that makes the call, belong to the same code segment. If the two procedures are defined in different program modules or files, the fact that they belong to the same segment is defined in concordance with the names of code segments in which the procedures were defined. The code segment needs to have the **same name**. The link-editor knows to concatenate in a single segment code segments with the same name from different modules.

The declaration of a procedure that is defined in another program module than the one that makes the call (uses the procedure) is made through the EXTRN directive. The called procedure has to be declared with the PUBLIC directive in the module in which it is defined. EXTRN and PUBLIC declarations must be written **inside** the segment and not outside for near procedure.

When calling a FAR type procedure CS, IP are saved on the stack. In this case the two procedures must belong to **different** segments. EXTRN declaration is made **outside** the segment and the PUBLIC inside the segment. FAR type calling is used only when the NEAR type calling is not possible, because this type of call is slower due to the more references made to the stack both at calling time and return time. A FAR type call is necessary when the length of the two procedures might exceed 64K, this being the maximum admitted dimension for a segment.

Procedure definition example: NEAR type procedures with procedures in different modules/files:

The calling, main, procedure:

```
DATE SEGMENT   PARA PUBLIC 'DATA'        ; data segment definition
        ;...
DATE ENDS
STAC SEGMENT PARA STACK  'stack'         ;stack segment definition
                        db 64           dup ('MY_STACK')
STAC ENDS

COD1  SEGMENT       PARA PUBLIC 'CODE'  ; cod segment definition

EXTRN PROCED: NEAR
PRPRINC         PROC   FAR                          ; main procedure definition
ASSUME CS: COD1, DS: DATE, SS: STAC, ES: NOTHING
        PUSH   ds                       ;prepare stack
        SUB    ax, ax                   ;to return
        PUSH   ax                       ; to DOS
        MOV    AX, DATE                         ; load register
        MOV    DS, AX                   ; DS with data segment

; The instructions of the main procedure
        CALL   PROCED                   ; call procedure
; Other instructions

        RET                             ; coming back to DOS
PRPRINC         ENDP                    ; end procedure
COD1  ENDS                              ; segment's end
END PRPRINC                         ; end of the first module
```

----------end of first file

The called procedure defined in another program module:

```
COD1 SEGMENT      PARA 'CODE'        ;       segment       code
definition
PUBLIC     PROCED                   ;   declare   proced   as
PUBLIC
ASSUME      CS: COD1
PROCED     PROC       NEAR          ; procedure definition

;The instructions of the called procedure

     RET                  ;  coming  back  to  the  procedure,
which made the call
```

```
PROCED    ENDP                ; end procedure
COD1 ENDS                     ; end segment
     END                            ; end of second module
```
---------- end of second file

FAR type procedure call example, procedure in different segments with the procedures in two different modules/files:

```
EXTRN PROCED2:FAR
STAC SEGMENT PARA STACK  'stack'   ;stack            segment
definition
     db 64          dup ('MY_STACK')
STAC ENDS

DATE SEGMENT   PARA PUBLIC 'DATA'  ;       data       segment
definition
     ;…                                    data definition
DATE ENDS

COD2 SEGMENT   PARA PUBLIC 'CODE'  ;       code       segment
definition
ASSUME CS: COD2, DS: DATE, SS:STAC, ES:NOTHING

PRPRINC2 PROC       FAR                  ;  main  procedure
definition
     PUSH DS                  ; prepare stack
     SUB  AX, AX              ; to return
     PUSH AX                  ; to DOS
     MOV  AX, DATE            ; load register
     MOV  DS, AX              ; DS with data segment
; The main procedure's instructions
     CALL PROCED2             ; procedure call

; Other instructions

     RET                      ; coming back to DOS
PRPRINC2 ENDP                 ; end procedure
COD2 ENDS                     ; end segment
     END PRPRINC              ; end of the first module
```

---------- end of first file

The called procedure defined in another program module:

```
COD3 SEGMENT     PARA 'CODE'       ;       code       segment
definition
PUBLIC    PROCED2                 ;  procedure  declaration
as public
```

```
ASSUME    CS: COD3
PROCED2   PROC FAR       ; procedure definition
; The instructions of the called procedure

          RETF                                          ; back
     to the procedure which made the call
PROCED2   ENDP                ; end procedure
COD3      ENDS                ; end segment
          END                 ; end of second module
```
---------- end of second file

Passing parameters to procedures

There are three known types of parameter transfers to procedures in assembly language: through registers, through pointers and data structure and through the stack.

Transfer through registers

The advantage of this solution is that that in the procedure, the actual parameters are immediately available. For register conservation, these are saved on the stack before calling the procedure and are restored after returning from the procedure. There are 2 disadvantages of this:

- the limited number of available registers
- non-uniformity of the method – there is no ordered modality of transferring, each procedure having it's own rules for transfer

Another advantage is speed, many operations with the memory (stack) not needed.

Transfer through memory

In this transfer type a data zone is prepared previously and the address of this data zone is transmitted to the procedure.

To ease access to the parameters it is recommended to define a structure, which describes the structure of the parameters:

```
_ZONA STRUC
        VAL1        DD    ?
        VAL2        DD    ?
        RETURN      DD    ?
_ZONA ENDS

DAT SEGMENT PARA PUBLIC 'data'
        ZONE _ZONA    <10, 20, ?>
dat ends

COD SEGMENT PARA PUBLIC 'code'
Assume cs:cod, ds:dat
```

```
extrn   proce:near
        LEA    BX, ZONE
        CALL  PROCe
cod ends
end
```

Parameter transfer through stack

Transferring parameters through the stack is the most uniform transfer modality. The transfer through stack is compulsory if the applications contain both ASM modules and modules in high level languages. The standard access technique to the parameters procedure is based on based addressing using BP register, which uses by default SS register as segment register to access the data. The access is achieved through the following operations, executed when entering the procedure:

- BP register is saved on the stack
- SP is copied to BP
- the registers used by the procedure are saved on the stack
- the parameters are accessed through indirect addressing using BP

When ending the procedure, the following operations are executed:

- the saved registers are restored
- BP is restored
- Return to the program which made the call through RET

Lab tasks

1. Study the given examples, noticing the differences between the two procedure call types: FAR and NEAR.
2. Write a program which calculates the sum of a string of numbers using a NEAR and then a FAR type procedure, written in another code segment, first both segments being written in the same file and then in different files. The procedure will be called *sum* and it will get as input parameters: the address and length of the string from DS: BX and CX registers. The procedure will return the sum in AX register.

Observations:

- The procedures which are to be included in a library will be defined of the same type, FAR or NEAR, in segments with the same name (if possible), in order not to complicate any more the call and the link edition.
- It is also recommended to group procedures of the same type (mathematical, display, etc) in different libraries having suggestive names.

Solved problem: Write a recursive procedure to display a number stored in AX

Solution:

```
TIP     STRUC                    ; pattern for parameters
   _BP  DW ?
   _CS  DW ?
   _IP  DW ?
   N    DW ?
TIP     ENDS


MYSTACK SEGMENT STACK 'stack'
        DB 4096 DUP (?)      ; stack segment declaration
MYSTACK ENDS


COD SEGMENT PARA PUBLIC 'CODE'
   ASSUME CS:COD, SS:MYSTACK
   DISPL PROC FAR
        PUSH BP         ; standard access
        MOV BP, SP           ; sequence
        PUSH DX
        PUSH AX         ;   we   will   work   with   these
registers in the
                        ; procedure, so we save them
        PUSH BX

        MOV AX,[BP].N
        CMP AX, 10               ; if n<10, dl=n
        MOV DL, AL
        JB DISPLAY_1        ;  jump  to  display  (we  have
only one number)
        MOV BX, 10               ; general case
        MOV DX, 0           ;  calculates  n/10  and  n  mod
10
        DIV BX              ; AX=n/10;
                            ; dl=n mod 10
        PUSH AX             ;  recursive  call  with  n/10
parameter
        CALL FAR PTR DISPL
DISPLAY_1:

        ADD DL, '0'              ; +'0'
        MOV AH, 02H         ; Dos function for display
```

```
        INT  21H            ; display

        POP  BX             ; restore
        POP  AX                       ; registers
        POP     DX
        POP  BP
        RETF 2              ;  FAR type return
   DISPL     ENDP

   START:


        MOV  AX, 65535 ; prepare register with number to
display
        PUSH AX        ;  we  put  it  on  the  stack  as
parameter
        CALL FAR PTR DISPL  ; procedure call

        MOV  AX, 4C00H ; return  to
        INT  21H       ; DOS

COD     ENDS
        END START
```