Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχ. Και Μηχ. Υπολογιστών
Εργαστήριο Μικροϋπολογιστών , 7ο εξάμηνο - Ροή Υ
Ακαδημαϊκή Περίοδος : 2011-2012

4η ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ

Ομάδα: C16
Γερακάρης Βασίλης Α.Μ.: 03108092
Διβόλης Αλέξανδρος Α.Μ.: 03107238
Ίεντσεκ Στέργιος Α.Μ.: 03108690

Εισαγωγικά

Σε αυτήν την σειρά ασκήσεων χρησιμοποιήθηκαν κάποιες μακροεντολές ώστε ο κώδικας να είναι πιο κατανοητός και ο προγραμματισμός εύκολος. Παρακάτω βλέπουμε το αρχείο **MACROS.TXT:**

```
;This macro change registers AH,AL
READ MACRO
    MOV AH,8
    INT 21H
ENDM

;This macro changes registers AH,DL
PRINT MACRO CHAR
    PUSH AX
    PUSH DX
    MOV DL,CHAR
    MOV AH,02H
    INT 21H
    POP DX
    POP AX
ENDM

;This macro change registers AH,DX
PRINT_STRING MACRO STRING
        PUSH AX
        PUSH DX
    MOV DX,OFFSET STRING ;Assume that string is a variable or constant, NOT
an address
    MOV AH,09H
    INT 21H
    POP DX
    POP AX
ENDM

PRINT_NUM MACRO CHAR
    MOV DL, CHAR
    ADD DL, 30H
    MOV AH, 2
    INT 21H
ENDM

PAUSE MACRO
    PUSH AX
    PUSH DX
    LEA DX,PKEY        ;<=>MOV DX, OFFSET PKEY;GIVES THE OFFSET OF PKEY TO DX
    MOV AH,9
    INT 21H            ;OUTPUT STRING AT DS:DX
    MOV AH,8           ;WAIT FOR PRESSING OF A KEY
    INT 21H        ;WITHOUT ECHO->8
    PRINT 0AH
    PRINT 0DH
    POP DX
    POP AX
ENDM

EXIT MACRO
    MOV AH,4CH
    INT 21H
ENDM
```

EXTRA_MACROS.TXT:

```
SCROLL_UP_WIN MACRO START_LIN START_COL END_LIN END_COL UP_NUM
;messes with AX,BH,CX,DX
    PUSH AX
    MOV AH,06H
    MOV AL,UP_NUM       ;number of lines to scroll up|0->all lines
    MOV CH,START_LIN
    MOV CL,START_COL
    MOV DH,END_LIN
    MOV DL,END_COL
    MOV BH,07H          ;attribute:0000(black) bckgrnd clr, 0111(light
grey)char clr
    INT 10H
    POP AX
ENDM

SCROLL_DOWN_WIN MACRO START_LIN START_COL END_LIN END_COL UP_NUM
    PUSH AX
    MOV AH,07H
    MOV AL,UP_NUM       ;number of lines to scroll up|0->all lines
    MOV CH,START_LIN
    MOV CL,START_COL
    MOV DH,END_LIN
    MOV DL,END_COL
    MOV BH,07H          ;attribute:0000(black) bckgrnd clr, 0111(light
grey)char clr
    INT 10H
    POP AX
ENDM

READ_NW MACRO
;messes with AX,DL,returns in AL=char, if ZF=0(there was something to read)
;reads without echo
    MOV AH,06H
    MOV DL,0FFH
    INT 21H
ENDM

LOCATE MACRO LIN COL PAGE
;messes with AH,DX,BH
    MOV AH,02H
    MOV DH,LIN
    MOV DL,COL
    MOV BH,PAGE
    INT 10H
ENDM

PRINT_BIOS MACRO CHAR
    MOV AH,0AH      ;funct code
    MOV AL,CHAR
    MOV BH,00H      ;page num
    MOV CX,1        ;times we print char
    INT 10H
ENDM
```

Άσκηση i

Στην άσκηση αυτή υλοποιούμε έναν calculator που κάνει πρόσθεση και αφαίρεση δύο (το πολύ 4ψήφιων) δεκαεξαδικών αριθμών. Χρησιμοποιήσαμε για αυτό τον emu8086 και την βιβλιοθήκη που φτιάξαμε, με τις κατάλληλες υπορουτίνες. Η βιβλιοθήκη που δημιουργήσαμε φαίνεται παρακάτω.

Library.inc:

```
;***********************************************************************************************
;Library file has some usual macros and some definition macros. We have to call these macros,
;which only make the wanted code of Our Procedures, before "END MAIN" instruction.
;
;FORM
;_______________________________________________________________________________________________
;***********************************************************************************************
;DEFINE_? MACRO
;LOCAL
;LOCAL
;LOCAL
;LOCAL SKIP_?
;JMP SKIP_?
;
;? PROC NEAR
;   [/CODE]
;? ENDP
;
;SKIP_?:
;   DEFINE_? ENDM
;***********************************************************************************************;
;***********************************************************************************************
;--------------------------------R-E-G-U-L-A-R-----M-A-C-R-O-S----------------------------------
SWAP_W MACRO WRD_1,WRD_2
    ;swap 2 16bit, except "mem-mem" or "SI-smthing" or "smthing-SI"
    PUSH SI
    PUSH DI
    MOV SI,WRD_1
    MOV DI,WRD_2
    MOV WRD_1,DI
    MOV WRD_2,SI
    POP DI
    POP SI
SWAP_W ENDM
;----------------------------D-E-F-I-N-I-T-I-O-N------M-A-C-R-O-S-------------------------------
DEFINE_AL_ITOA MACRO
LOCAL HIGH_HEX,LOW_CHECK,LOW_HEX
LOCAL AL_ITOA_END,SKIP_AL_ITOA
JMP SKIP_AL_ITOA

AL_ITOA PROC NEAR
;This routine takes the AX number and returns the two hex_chars of it at AH-AL
;Register Affections:AX
    MOV AH,AL       ;AH=AL= h3 h2 h1 h0 l3 l2 l1 l0
    SHR AH,4        ;AH<-0 0 0 0 h3 h2 h1 h0
    AND AL,0FH      ;AL<-0 0 0 0 l3 l2 l1 l0
    CMP AH,9        ;If AH>9 is hex_char
    JA HIGH_HEX
    ADD AH,30H      ;else is dec_char
    JMP LOW_CHECK   ;go on!
```

```
HIGH_HEX:
    ADD AH,37H
LOW_CHECK:
    CMP AL,9
    JA LOW_HEX
    ADD AL,30H
    JMP AL_ITOA_END
LOW_HEX:
    ADD AL,37H
AL_ITOA_END:
    RET
AL_ITOA ENDP

SKIP_AL_ITOA:
    DEFINE_AL_ITOA ENDM
;************************************************************************************************
DEFINE_AX_ATOI MACRO
LOCAL ERROR_END,CHECK_AL,AX_ATOI_END,NORMAL
LOCAL CHECK_HEX_AH_CAPITAL,CHECK_HEX_AH_LOWERCASE
LOCAL CHECK_HEX_AL_CAPITAL,CHECK_HEX_AL_LOWERCASE
LOCAL SKIP_AX_ATOI
JMP SKIP_AX_ATOI

AX_ATOI PROC NEAR
;This routine takes the 2 hex_chars in AH-AL and returns the integer  in AL
;When returns, if  (AH==0) then everything is ok,number in AL.
;Else if (AH==1) there where no hexadecimal chars (error case)!
;Register Affections:AX
    CMP AH,30H                  ;
    JB ERROR_END                ;error case:AH<30H
    CMP AH,39H                  ;
    JA CHECK_HEX_AH_CAPITAL     ;
    SUB AH,30H                  ;if we are here AH='0'-'9'
    JMP CHECK_AL
CHECK_HEX_AH_CAPITAL:
    CMP AH,41H
    JB ERROR_END                ;error case:39H<AH<41H
    CMP AH,46H
    JA CHECK_HEX_AH_LOWERCASE
    SUB AH,37H                  ;if we are here AH='A'-'F'
    JMP CHECK_AL
CHECK_HEX_AH_LOWERCASE:
    CMP AH,61H
    JB ERROR_END                ;error case:46H<AH<61H
    CMP AH,66H
    JA ERROR_END                ;error case:AH>66H
    SUB AH,57H                  ;if we are here AH='a'-'f'
;=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
CHECK_AL:
    CMP AL,30H
    JB ERROR_END
    CMP AL,39H
    JA CHECK_HEX_AL_CAPITAL
    SUB AL,30H
    JMP NORMAL
CHECK_HEX_AL_CAPITAL:
    CMP AL,41H
    JB ERROR_END
    CMP AL,46H
    JA CHECK_HEX_AL_LOWERCASE
    SUB AL,37H
    JMP NORMAL
```

```
CHECK_HEX_AL_LOWERCASE:
    CMP AL,61H
    JB ERROR_END
    CMP AL,66H
    JA ERROR_END
    SUB AL,57H
;=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
NORMAL:
    SHL AH,4
    OR AL,AH
    MOV AH,0                ;everything ok:AH<-0
    JMP AX_ATOI_END
ERROR_END:
    MOV AH,1                ;not ok:AH<-1
AX_ATOI_END:
    RET
AX_ATOI ENDP

SKIP_AX_ATOI:
    DEFINE_AX_ATOI ENDM
;***********************************************************************************************

DEFINE_INPUT_NUMS MACRO
LOCAL IN_NUM1,SIGN_EXPECT
LOCAL NEXT_CHECK,NEXT_HEX
LOCAL IN_NUM2,EQUAL_EXPECT
LOCAL END_CHECK,END_INPUT
LOCAL SKIP_INPUT_NUMS
JMP SKIP_INPUT_NUMS

INPUT_NUMS PROC NEAR
    MOV CX,4            ;3 digits input (except 1st obligatory digit input)
    MOV DX,0            ;Initialize for the first number
IN_NUM1:
    READ
    CMP AL,'Q'
    JE EXODOS
    CMP AL,'q'
    JE EXODOS           ;Check if
    CMP AL,'+'
    JE NEXT_CHECK
    CMP AL,'-'
    JE NEXT_CHECK
    MOV BL,AL           ;Keep printable form in BL
    MOV AH,30H          ;Because we have AX_ATOI procedure,
    CALL AX_ATOI        ;which returns the INTEGER in AL<-0000(from AH) c3 c2 c1 c0(from AL,if
it's hex)
    CMP AH,0            ;If (AH==0) we have acceptable character(hexadecimal)
    JNE IN_NUM1         ;Else we have NOT acceptable character
    PRINT BL            ;We are OK, so print pushed char
    SHL DX,4            ;Create space to add new digit
    ADD DX,AX           ;AX -> AH-AL -> 00000000 - 0000 c3 c2 c1 c0
    LOOP IN_NUM1
;If we are here 4 digits have been inputed and expected SIGN of the calculation
SIGN_EXPECT:
    READ
    CMP AL,'Q'
    JE EXODOS
    CMP AL,'q'
    JE EXODOS
    CMP AL,'+'
    JE NEXT_HEX         ;If '+' then go on
    CMP AL,'-'
    JNE SIGN_EXPECT
    JMP NEXT_HEX        ;If '-' then go on
NEXT_CHECK:
    CMP CX,4            ;Check if at least one digit has been inputed
    JE IN_NUM1          ;If not REREAD!
NEXT_HEX:
    MOV SIGN,AL         ;Save the SIGN of the calculation
    PRINT AL            ;Print it on screen
    MOV NUM_1,DX        ;Save the NUM_1
;Initializations for the second input
    MOV CX,4
    MOV DX,0
IN_NUM2:
    READ
    CMP AL,'Q'
    JE EXODOS
    CMP AL,'q'
    JE EXODOS           ;Check if
    CMP AL,'='
    JE END_CHECK
    MOV BL,AL           ;Keep printable form in BL
    MOV AH,30H          ;Because we have AX_ATOI procedure,
    CALL AX_ATOI        ;which returns the INTEGER in AL<-0000(from AH) c3 c2 c1 c0(from AL,if
it's hex)
    CMP AH,0            ;If (AH==0) we have acceptable character(hexadecimal)
    JNE IN_NUM2         ;Else we have NOT acceptable character
    PRINT BL            ;We are OK, so print pushed char
    SHL DX,4            ;Create space to add new digit
    ADD DX,AX           ;AX -> AH-AL -> 00000000 - 0000 c3 c2 c1 c0
    LOOP IN_NUM2
;If we are here 4 digits have been inputed and expected EQUAL to calculate
EQUAL_EXPECT:
    READ
    CMP AL,'Q'
    JE EXODOS
    CMP AL,'q'
    JE EXODOS
    CMP AL,'='
    JNE EQUAL_EXPECT    ;Repeat until '=' is given, or exit request
    JMP END_INPUT
END_CHECK:
    CMP CX,4            ;Check if at least one digit has been inputed
    JE IN_NUM2          ;If not REREAD!
END_INPUT:
    MOV NUM_2,DX        ;Save the NUM_2
    PRINT AL            ;Print EQUAL sign
    RET
INPUT_NUMS ENDP

SKIP_INPUT_NUMS:
    DEFINE_INPUT_NUMS ENDM
;**********************************************************************************************
DEFINE_CALCULATION MACRO
LOCAL ADD_NUMS,NOT_OVERFLOW
LOCAL SUB_NUMS,SUB_CALC
LOCAL END_CALCULATION
LOCAL SKIP_CALCULATION
JMP SKIP_CALCULATION

CALCULATION PROC NEAR
    MOV RES_SIGN,0      ;Initialize sign of result to NONE (positive)
    MOV AX,NUM_1
    MOV BX,NUM_2        ;RESULT=RES_HIGH - RES_LOW
    MOV RES_HIGH,0      ;The result is 32bit for the case of adding big 16bit-nums
    CMP SIGN,'-'        ;If calculation sign is '-' we hanve subtraction
    JE SUB_NUMS         ;2 cases: i.'-' or ii.'+'
ADD_NUMS:
    ADD AX,BX
    JNC NOT_OVERFLOW
    MOV RES_HIGH,1
NOT_OVERFLOW:
    MOV RES_LOW,AX
    JMP END_CALCULATION
SUB_NUMS:
    CMP AX,BX
    JAE SUB_CALC
    SWAP_W AX,BX
    MOV RES_SIGN,1      ;The result is negative
SUB_CALC:
    SUB AX,BX           ;AX (minus) BX > 0 in any case
    MOV RES_LOW,AX
END_CALCULATION:
    ;Here we have the result in RES_HIGH - RES_LOW
    ;and the sign of the result in RES_SIGN (0->'+', 1->'-'
    RET
CALCULATION ENDP

SKIP_CALCULATION:
    DEFINE_CALCULATION ENDM
;***********************************************************************************************;
DEFINE_PRINT_RESULTS MACRO
LOCAL PRINT_HEX_RESULT
LOCAL PRINT_DEC_RESULT
LOCAL SKIP_PRINT_RESULTS
JMP SKIP_PRINT_RESULTS

PRINT_RESULTS PROC NEAR
    MOV DX,RES_HIGH
    MOV BX,RES_LOW
    CMP RES_SIGN,0
    JE PRINT_HEX_RESULT
    PRINT '-'
PRINT_HEX_RESULT:
    CALL PRINT_HEX_SPEC
    PRINT '='
    CMP RES_SIGN,0
    JE PRINT_DEC_RESULT
    PRINT '-'           ;RES_SIGN can only be 0->positive, or 1->negative
PRINT_DEC_RESULT:
    CALL PRINT_DEC_SPEC
    PRINT_STRING NEW_LINE
    RET
PRINT_RESULTS ENDP

SKIP_PRINT_RESULTS:
    DEFINE_PRINT_RESULTS ENDM
;***********************************************************************************************;
DEFINE_PRINT_HEX_SPEC MACRO
LOCAL NEXT_PRINT0
LOCAL END_PRINT_HEX_SPEC
LOCAL SKIP_PRINT_HEX_SPEC
JMP SKIP_PRINT_HEX_SPEC
```

```
PRINT_HEX_SPEC PROC NEAR
    ;We have the num in DX-BX, but DX(=RES_HIGH) is surely 1 or 0
    MOV 1ST_NZ,0        ;Initialize 1ST_NZ - first-non-zero flag
    CMP DL,0
    JE NEXT_PRINT0
    PRINT 31H
    MOV 1ST_NZ,1
NEXT_PRINT0:
    MOV AL,BH
    CALL PRINT_AL
    MOV AL,BL
    CALL PRINT_AL
    CMP 1ST_NZ,0
    JNE END_PRINT_HEX_SPEC
    PRINT 30H
END_PRINT_HEX_SPEC:
    RET
PRINT_HEX_SPEC ENDP

SKIP_PRINT_HEX_SPEC:
    DEFINE_PRINT_HEX_SPEC ENDM
;***********************************************************************************************;
DEFINE_PRINT_AL MACRO
LOCAL CHECK_1,NEXT_PRINT1
LOCAL CHECK_2,END_PRINT_AL
LOCAL SKIP_PRINT_AL
JMP SKIP_PRINT_AL

PRINT_AL PROC NEAR
    ;We produce from AL(hex) AX<-2hex chars=AH:High Hex Char  - AL:Low Hex Char
    ;1ST_NZ(byte) 0:haven't printed yet non-zero digit 1:have already printed
    CALL AL_ITOA
    CMP AH,30H
    JE CHECK_1
    MOV 1ST_NZ,1
    PRINT AH
    JMP NEXT_PRINT1
CHECK_1:
    CMP 1ST_NZ,0
    JE NEXT_PRINT1
    PRINT AH
NEXT_PRINT1:
    CMP AL,30H
    JE CHECK_2
    MOV 1ST_NZ,1
    PRINT AL
    JMP END_PRINT_AL
CHECK_2:
    CMP 1ST_NZ,0
    JE END_PRINT_AL
    PRINT AL
END_PRINT_AL:
    RET
PRINT_AL ENDP

SKIP_PRINT_AL:
    DEFINE_PRINT_AL ENDM
;***********************************************************************************************;
DEFINE_PRINT_DEC_SPEC MACRO
LOCAL DIV_LOOP
LOCAL PRINT_LOOP
LOCAL SKIP_PRINT_DEC_SPEC
```

```
JMP SKIP_PRINT_DEC_SPEC

PRINT_DEC_SPEC PROC NEAR
    ;We have in DX-BX the 32bit result
    MOV AX,BX
    MOV BX,10
    MOV CX,0
DIV_LOOP:
    DIV BX
    PUSH DX
    INC CX
    MOV DX,0
    CMP AX,0
    JNE DIV_LOOP
PRINT_LOOP:
    POP DX
    ADD DL,30H
    PRINT DL
    LOOP PRINT_LOOP
    RET
PRINT_DEC_SPEC ENDP

SKIP_PRINT_DEC_SPEC:
    DEFINE_PRINT_DEC_SPEC ENDM
;***********************************************************************************************;
```

Παρακάτω φαίνεται ο κώδικας του προγράμματος. Παρατηρούμε ότι πριν το τέλος του προγράμματος πληκτρολογούμε τα αντίστοιχα DEFINE_? Των υπορουτινών που χρειαζόμαστε.

ASK_1_LibCall.asm:

```
INCLUDE Library.inc
INCLUDE MACROS.TXT

STACK_SEG SEGMENT STACK
    DW 128 DUP(?)
ENDS

DATA_SEG SEGMENT
    NEW_LINE DB 0AH,0DH,"$"
    OUT_MSG DB "press any key to replay or 'Q','q' to exit$"
    SIGN DB ?
    RES_SIGN DB ?
    1ST_NZ DB ?
    NUM_1 DW ?
    NUM_2 DW ?
    RES_HIGH DW ?
    RES_LOW DW ?
ENDS

CODE_SEG SEGMENT
    ASSUME CS:CODE_SEG,SS:STACK_SEG,DS:DATA_SEG,ES:DATA_SEG
MAIN PROC FAR
    ;SET SEGMENT REGISTERS
    MOV AX,DATA_SEG
    MOV DS,AX
    MOV ES,AX
;=-=-=-=-==-=-=-=-=-=-=-CODE-=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=
START:
    CALL INPUT_NUMS
    CALL CALCULATION
    CALL PRINT_RESULTS
    PRINT_STRING OUT_MSG
    READ
    CMP AL,'Q'
    JE EXODOS
    CMP AL,'q'
    JE EXODOS
    PRINT_STRING NEW_LINE
    JMP START
EXODOS:
    EXIT
MAIN ENDP
ENDS

;Library definitions of procedures
DEFINE_AX_ATOI
DEFINE_AL_ITOA
DEFINE_INPUT_NUMS
DEFINE_CALCULATION
DEFINE_PRINT_RESULTS
DEFINE_PRINT_HEX_SPEC
DEFINE_PRINT_AL
DEFINE_PRINT_DEC_SPEC

END MAIN
```

Άσκηση ii

Στην άσκηση αυτή υλοποιούμε ένα terminal PC το οποίο επικοινωνεί με την βοήθεια του UART (Universal Asynchronus Receiver Transmitter) 8250 και επικοινωνούμε σύμφωνα με το πρωτόκολο RS232. Εξομοιώσαμε την επικοινωνία με την βοήθεια της εφαρμογής DOSBOX, την οποία τρέχουμε δύο φορές με διαφορετική ρύθμιση στο serial1.

Πρώτα serial1=nullmodem

Στη συνέχεια serial1=nullmodem server:localhost

Για την υλοποίηση της παραπάνω εφαρμογής δηιουργήσαμε δύο βιβλιοθήκες, μία για την αρχικοποίηση της RS232 και επικοινωνία με αυτήν, και μία για τις υπόλοιπες συναρτήσεις που απαιτούνται. Παρακάτω φαίνονται οι δύο βιβλιοθήκες.

RS232_ROUTINES.inc:

```
;************************R*S*2*3*2*-*-*-*-*-*B*A*S*I*C**L*I*B*R*A*R*Y****************************
;*             This library defines the three basic procedures we need to communicate          *
;*             via UART(Universal Asynchronus Receiver Transmitter) 8250,RS232 standard        *
;*                     1.OPEN_RS232    initializes RS232 standard communication                 *
;*                     2.RXCH_RS232    READS a char from serial port                            *
;*                     3.TXCH_RS232    SENDS a char to serial port                              *
;************************************************************************************************
DEFINE_OPEN_RS232 MACRO
LOCAL START,SKIP_OPEN_RS232
JMP SKIP_OPEN_RS232

;This routine initializes RS232 standard communication
;Messes with AX,DX,DI
OPEN_RS232 PROC NEAR
    JMP START
    BAUD_RATE_DIVISOR LABEL WORD    ;divisor=115200/baud_rate, same declaration as
    DW 1047 ;110  baud rate     (OFFSET BAUD_RATE_DIVISOR)+0    BR=000
    DW 768  ;150  baud rate     (OFFSET BAUD_RATE_DIVISOR)+2    BR=001
    DW 384  ;300  baud rate     (OFFSET BAUD_RATE_DIVISOR)+4    BR=010
    DW 192  ;600  baud rate     (OFFSET BAUD_RATE_DIVISOR)+6    BR=011
    DW 96   ;1200 baud rate     (OFFSET BAUD_RATE_DIVISOR)+8    BR=100
    DW 48   ;2400 baud rate     (OFFSET BAUD_RATE_DIVISOR)+10   BR=101
    DW 24   ;4800 baud rate     (OFFSET BAUD_RATE_DIVISOR)+12   BR=110
    DW 12   ;9600 baud rate     (OFFSET BAUD_RATE_DIVISOR)+14   BR=111 "+14->LSByte, +15->MSByte"
START:
    STI     ;Set interrupt flag != CLI; Clear Interrupt Flag (?)
; Initial Values of RS232
    MOV AH,AL       ;AH<-AL parameters:
;BR2|BR1|BR0|EVEN_OR_ODD_PARITY|PARITY_ON|NUM_STOP_BIT|WORD_LENGTH_1|WORD_LENGTH_0
    MOV DX,3FBH     ;Line Control REGISTER address
    MOV AL,80H      ;AL<-1000 0000 : DLAB=1
    OUT DX,AL       ;send to register
    MOV DL,AH       ;DL<- Parameters
    ROL DL,4
    AND DX,0EH      ;DH<-00H, DL<-0000 BR2|BR1|BR0|0 --->offset=0,2,4,6,8,10,12,14
    MOV DI,OFFSET BAUD_RATE_DIVISOR
    ADD DI,DX       ;DI<-memory address of correct divisor
    MOV DX,3F9H     ;MSByte of Baudrate divisor REGISTER adddress (DLAB=1)
    MOV AL,CS:[DI]+1;CS:[DI]+1 -> MSByte of divisor
    OUT DX,AL       ;send to register
    MOV DX,3F8H     ;LSByte of Baudrate divisor (DLAB=1)
    MOV AL,CS:[DI]  ;CS:[DI]   -> LSByte of divisor
    OUT DX,AL       ;send to register
    MOV DX,3FBH     ;Line Control REGISTER address
    MOV AL,AH       ;AL<-parameters
    AND AL,1FH      ;AL<-
;0(DLAB)|0(SOUT not deactivated)|0(normal parity bit)|
;EVEN_OR_ODD_PARITY|PARITY_ON|NUM_STOP_BIT|WORD_LENGTH_1|WORD_LENGTH_0
    OUT DX,AL       ;send to register
    MOV DX,3F9H     ;Interrupt Enable REGISTER address
    MOV AL,0        ;disabled interrupts 0  Rx data int. enable
                                        ;1  Tx holding reg. empty int.
                                        ;2  Rx status int. enable (ie Parity, Framing, overrun and
                                       ;BREAK enable).
                                        ;3  Modem signal change int. enable.
    OUT DX,AL
    RET
OPEN_RS232 ENDP

SKIP_OPEN_RS232:
    DEFINE_OPEN_RS232 ENDM
;*************************************************************************************************
DEFINE_RXCH_RS232 MACRO
LOCAL END_RXCH_RS232
LOCAL SKIP_RXCH_RS232
JMP SKIP_RXCH_RS232

;This routine READS a char from serial port
;Messes with AL,DX
RXCH_RS232 PROC NEAR
    MOV DX,3FDH         ;Line Status REGISTER Address
    IN AL,DX            ;Input Status of Line (to check if there is something to read)
    AND AL,1            ;AL (AND) 00000001 ->IF NonZero => DR=1 => something has come
    JZ END_RXCH_RS232   ;AL<-0(NUL) means there is nothing to Read
    MOV DX,3F8H         ;Data Read/Write REGISTER address.
    IN AL,DX            ;READ IT!
END_RXCH_RS232:
    RET
RXCH_RS232 ENDP

SKIP_RXCH_RS232:
    DEFINE_RXCH_RS232 ENDM
;*************************************************************************************************
DEFINE_TXCH_RS232 MACRO
LOCAL SKIP_TXCH_RS232
LOCAL TXCH_RS232_2
JMP SKIP_TXCH_RS232

;This routine SENDS a char to serial port
;Messes with AL(there is the CHAR_2_SEND),DX
TXCH_RS232 PROC NEAR
    PUSH AX
    MOV DX,3FDH         ;Line Status Register Address
TXCH_RS232_2:
    IN AL,DX            ;Input Status of Line (to check if TRANSMITTER REGISTER is clear to send)
    TEST AL,20H         ;AL (AND) 0010 0000 ->IF NonZero => THRE=1 => Transmitter Holding Register
                       ;is empty, we can send
    JZ TXCH_RS232_2     ;Loop from proc_begin, until Transmitter Register is empty!
    MOV DX,3F8H         ;Data Read/Write REGISTER address.
    POP AX              ;Retrieve AL<-CHAR_2_SEND
    OUT DX,AL           ;Send it to Transmitter Register(=Data Read/Write Register)
    RET
TXCH_RS232 ENDP
```

```
SKIP_TXCH_RS232:
    DEFINE_TXCH_RS232 ENDM
```

TERM_LIB.inc:

```
;********************************PROJECT****4-2***LIBRARY***************************************
;*                  This library defines three procedures                                     *
;*                                                                                             *
;*                          1.INPUT_CHOOSE      initializes ECHO CHOICE and BAUD RATE          *
;*                          2.PRINT_START_SCRN  prints the main screen                         *
;*                          3.MAIN_LOOP         main loop procedure of our program             *
;***********************************************************************************************
DEFINE_INPUT_CHOOSE MACRO
LOCAL ECHO_ERR,BAUD_RATE_ERR
LOCAL SKIP_INPUT_CHOOSE
JMP SKIP_INPUT_CHOOSE

INPUT_CHOOSE PROC NEAR
    SCROLL_UP_WIN 0 0 24 80 0
    LOCATE 0 0 0
    PRINT_STRING ECHO_MSG
ECHO_ERR:
    READ
    CMP AL,30H
    JB ECHO_ERR
    CMP AL,31H
    JA ECHO_ERR
    PRINT AL
    SUB AL,30H
    MOV ECHO_FLG,AL
    PRINT_STRING NEW_LINE
    PRINT_STRING BAUD_RATE_MSG
BAUD_RATE_ERR:
    READ
    CMP AL,31H
    JB BAUD_RATE_ERR
    CMP AL,36H
    JA BAUD_RATE_ERR
    PRINT AL
    SUB AL,2FH          ;example(gave '1'):31h=29h=2h->010->baud rate 300
    SHL AL,5                        ;AL<-xxx0 0000
    AND AL,0E0H
    ADD AL,3                        ;AL<-xxx0 0011
(xxx||EVEN_OR_ODD_PARITY|PARITY_ON|NUM_STOP_BIT|WORD_LENGTH_1|WORD_LENGTH_0)
    MOV B_R_CHOICE,AL
    PRINT_STRING NEW_LINE
    PRINT_STRING PKEY
    READ
    SCROLL_UP_WIN 0 0 3 80 0
    RET
INPUT_CHOOSE ENDP

SKIP_INPUT_CHOOSE:
    DEFINE_INPUT_CHOOSE ENDM
;***********************************************************************************************
DEFINE_PRINT_START_SCRN MACRO
LOCAL SKIP_PRINT_START_SCRN
JMP SKIP_PRINT_START_SCRN
```

```
PRINT_START_SCRN PROC NEAR
    LOCATE 0 0 00H
    PRINT_STRING LOC_MSG
    MOV LOCAL_LIN,1
    LOCATE 11 0 00H
    PRINT_STRING SEPERATOR
    LOCATE 12 0 0
    PRINT_STRING REM_MSG
    MOV REMOTE_LIN,13
    RET
PRINT_START_SCRN ENDP

SKIP_PRINT_START_SCRN:
    DEFINE_PRINT_START_SCRN ENDM
;***********************************************************************************************
DEFINE_MAIN_LOOP MACRO
LOCAL FULL_REM_WIN,KEY_RECEIVED
LOCAL FULL_REM_WIN_2,GO_PRINT_RECEIVED
LOCAL SEND_CHECK,FULL_LOC_WIN
LOCAL KEY_PUSHED,FULL_LOC_WIN_2
LOCAL GO_PRINT,GO_ON_SEND
LOCAL SKIP_MAIN_LOOP
JMP SKIP_MAIN_LOOP

MAIN_LOOP PROC NEAR
    CALL RXCH_RS232     ;AL<-0 (NUL) means there is nothing to Read
    CMP AL,0            ;else AL<-char received
    JE SEND_CHECK
;[section=CHAR RECEIVED]
    CMP AL,0DH          ;check if ENTER received
    JNE KEY_RECEIVED    ;if not ENTER jump to KEY_PUSHED
    CMP REMOTE_LIN,22   ;Lines can be printed-limit
    JE FULL_REM_WIN
    ADD REMOTE_LIN,1
    MOV REMOTE_COL,0
    JMP SEND_CHECK
FULL_REM_WIN:
    SCROLL_UP_WIN 13 0 22 79 1
    MOV REMOTE_COL,0
    JMP SEND_CHECK
KEY_RECEIVED:
    CMP REMOTE_COL,80   ;0-79 column have been written (80 chars)
    JNE GO_PRINT_RECEIVED
    CMP REMOTE_LIN,10   ;Lines can be printed-limit
    JE FULL_REM_WIN_2
    ADD REMOTE_LIN,1
    MOV REMOTE_COL,0
    JMP GO_PRINT_RECEIVED
FULL_REM_WIN_2:
    SCROLL_UP_WIN 13 0 22 79 1
    MOV REMOTE_COL,0
GO_PRINT_RECEIVED:
    LOCATE REMOTE_LIN REMOTE_COL 0
    PRINT AL
    ADD REMOTE_COL,1
;[\section]
SEND_CHECK:
    READ_NW             ;if ZF=0 there was something to read (in AL)
    JZ MAIN_LOOP        ;if ZF=1 loop!
    CMP AL,1BH          ;check if ESC
    JE EXODOS
    CMP ECHO_FLG,1
    JNE GO_ON_SEND
;[section=ECHO ON]
    CMP AL,0DH          ;check if ENTER
    JNE KEY_PUSHED      ;if not ENTER jump to KEY_PUSHED
    CMP LOCAL_LIN,10    ;Lines can be printed-limit
    JE FULL_LOC_WIN
    ADD LOCAL_LIN,1
    MOV LOCAL_COL,0
    JMP GO_ON_SEND
FULL_LOC_WIN:
    SCROLL_UP_WIN 1 0 10 79 1
    MOV LOCAL_COL,0
    JMP GO_ON_SEND
KEY_PUSHED:
    CMP LOCAL_COL,80    ;0-79 column have been written (80 chars)
    JNE GO_PRINT
    CMP LOCAL_LIN,10    ;Lines can be printed-limit
    JE FULL_LOC_WIN_2
    ADD LOCAL_LIN,1
    MOV LOCAL_COL,0
    JMP GO_PRINT
FULL_LOC_WIN_2:
    SCROLL_UP_WIN 1 0 10 79 1
    MOV LOCAL_COL,0
GO_PRINT:
    LOCATE LOCAL_LIN LOCAL_COL 0
    PRINT AL
    ADD LOCAL_COL,1
;[\section]
GO_ON_SEND:
    CALL TXCH_RS232
    JMP MAIN_LOOP
    RET             ;not necessary, because it's infinite loop(ends with jump to EXODOS)
MAIN_LOOP ENDP

SKIP_MAIN_LOOP:
    DEFINE_MAIN_LOOP ENDM
;****************************************************************************************
```

Παρακάτω φαίνεται ο κώδικας του κυρίως προγράμματος. Η παρακάτω υλοποίηση είναι για παραγωγή .com αρχείου στον emu8086. Βέβαια δουλεύει και δημιουργώντας .exe αρχείο, το οποίο παρατίθεται στο τέλος της εργασίας.

ask_2.asm:

```
INCLUDE MACROS.TXT
INCLUDE EXTRA_MACROS.TXT
INCLUDE RS232_ROUTINES.INC
INCLUDE TERM_LIB.INC

org 100h

.data
    PKEY DB "Press any key...$"
    NEW_LINE DB 0AH,0DH,"$"
    LOC_MSG DB "LOCAL$"
    REM_MSG DB "REMOTE$"
    SEPERATOR DB 80 DUP(0C4H),"$"
    ECHO_MSG DB "With(1) or Without(0) ECHO? $"
    BAUD_RATE_MSG DB "Give Baud Rate:(1)300,(2)600,(3)1200,(4)2400,(5)4800,(6)9600:$"
    LOCAL_LIN DB 0
    LOCAL_COL DB 0
    REMOTE_LIN DB 12
    REMOTE_COL DB 0
    WHERE_2_WRITE DB 0
    ECHO_FLG DB 0
    B_R_CHOICE DB 0

.code

MAIN PROC FAR
;=-=-=-=-==-=-=-=-=-=-=-CODE-=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=
START:
    CALL INPUT_CHOOSE
    MOV AL,B_R_CHOICE               ;sthing 0000 0xxx
    CALL OPEN_RS232
;    MOV AH,00H
;    MOV AL,7
;    INT 10H
    CALL PRINT_START_SCRN
    CALL MAIN_LOOP
;    CALL INPUT_METHOD
;    READ
EXODOS:
    SCROLL_UP_WIN 0 0 24 80 0       ;to clear screen
    LOCATE 0 0 0                    ;to locate at the begining
    EXIT
MAIN ENDP
;*****************************************************************


DEFINE_OPEN_RS232
DEFINE_RXCH_RS232
DEFINE_TXCH_RS232
DEFINE_INPUT_CHOOSE
DEFINE_PRINT_START_SCRN
DEFINE_MAIN_LOOP
```

Termin_2.asm:

```
INCLUDE MACROS.TXT
INCLUDE EXTRA_MACROS.TXT
INCLUDE RS232_ROUTINES.INC
INCLUDE TERM_LIB.INC

STACK_SEG SEGMENT STACK
    DW 128 DUP(?)
ENDS

DATA_SEG SEGMENT
    PKEY DB "Press any key...$"
    NEW_LINE DB 0AH,0DH,"$"
    LOC_MSG DB "LOCAL$"
    REM_MSG DB "REMOTE$"
    SEPERATOR DB 80 DUP(0C4H),"$"
    ECHO_MSG DB "With(1) or Without(0) ECHO? $"
    BAUD_RATE_MSG DB "Give Baud Rate:(1)300,(2)600,(3)1200,(4)2400,(5)4800,(6)9600:$"
    LOCAL_LIN DB 0
    LOCAL_COL DB 0
    REMOTE_LIN DB 12
    REMOTE_COL DB 0
    WHERE_2_WRITE DB 0
    ECHO_FLG DB 0
    B_R_CHOICE DB 0
ENDS

CODE_SEG SEGMENT
    ASSUME CS:CODE_SEG,SS:STACK_SEG,DS:DATA_SEG,ES:DATA_SEG
MAIN PROC FAR
    ;SET SEGMENT REGISTERS
    MOV AX,DATA_SEG
    MOV DS,AX
    MOV ES,AX
;=-=-=-=-==-=-=-=-=-=-=-CODE-=-=-=-=-=-=-=-=-=-=-==-=-=-=-=-=-=-=
START:
    CALL INPUT_CHOOSE
    MOV AL,B_R_CHOICE               ;sthing 0000 0xxx
    ROL AL,5                        ;AL<-xxx0 0000
    ADD AL,3                        ;AL<-xxx0 0011
;(xxx||EVEN_OR_ODD_PARITY|PARITY_ON|NUM_STOP_BIT|WORD_LENGTH_1|WORD_LENGTH_0)
    CALL OPEN_RS232
    MOV AH,00H
    MOV AL,7
    INT 10H
    CALL PRINT_START_SCRN
    CALL MAIN_LOOP
EXODOS:
    SCROLL_UP_WIN 0 0 24 80 0       ;to clear screen
    LOCATE 0 0 0                    ;to locate at the begining
    EXIT
MAIN ENDP
;*****************************************************************
ENDS

DEFINE_OPEN_RS232
DEFINE_RXCH_RS232
DEFINE_TXCH_RS232
DEFINE_INPUT_CHOOSE
DEFINE_PRINT_START_SCRN
DEFINE_MAIN_LOOP

END MAIN
```