```
;8086 PROGRAM HW1_2_89101089.ASM
;ABSTRACT  : This program evaluates entered expression and displays result
;REGISTERS : Uses CS , DS , ES , SS , SP , BP , AX , BX , CX , DX ,
;PORTS     : ---
;PROCEDURES: ins_char       : inserts read charachter into char_buffer
;           push_var       : calculates decimal chars in char_buff and puts result in queue
;           push_op_t      : pushes operator to temporary operator stack (i.e. texas!)
;           move_stck      : moves top element of stack to queue
;           GetRes         : evaluates queue
;           GetOp          : calculates result of desired operator on two operators which are in stack
;           UnSign         : used to unsign operands before division & multiplication and set isSigned flag
;           DispRes        : convert 32-bit binary result to signed decimal   & display it
;-------------------------------------------------

  DATA      SEGMENT

  Sen_Max    DB 127
  Sen_Real   DB ?
  Sentence   DB 127 DUP(?)

  ; stack pointer backup

  bcup_op_t  DW ?

  ;charachter read part

  isSigned   DB 0
  isNegative DB 0
  char_fnd   DB 0
  char_buff  DB 5 DUP(0)
  char_coun  DB 0
  char_sum   DW 0

  ;calculation part

   operandL DW ?
   operandH DW ?

    Divisor DW 10
   ResWord0 DW 0
   ResWord1 DW 0
   TempWord DW 0

      queue DW 200 DUP(0)                    ; queue of double words
  ptr_queue DW offset queue                  ; pointer to current double word in queue
 calc_queue DW offset queue                   ; pointer used in calculating result

  MESS1     DB 0DH , 0AH , 'Error in expression $'
  MESS2     DB 0DH , 0AH , 11 DUP(0) , '$'
  MESS3     DB 0DH , 0AH , 'Enter expression to evaluate : (Specify negative numbers with # )' , 0DH , 0AH , '$'

  DATA      ENDS
```

```
;-------------------------------------------------

  STACK     SEGMENT Stack

   DW       100   DUP(0)
   sp_func   Label Word

   DW       100   DUP(0)                    ; stack of 100       words for ascii code of operators (+ , * , / , -)
   sp_op_t   Label Word

   DW       100   DUP(0)                    ;  "       "   double words for operands during evaluation
   sp_oprnd  Label Word

  STACK     ENDS

;-------------------------------------------------

  CODE      SEGMENT

   ASSUME    CS:CODE , DS:DATA , ES:DATA , SS:STACK


    START:  MOV  AX , DATA
            MOV  DS , AX                    ; initialize segments
            MOV  ES , AX

            MOV  AX , STACK
            MOV  SS , AX
            MOV  SP , offset sp_func

            MOV  DX , offset MESS3              ; display message
            MOV  AH , 09H
            INT  21H

     read:  MOV  DX , offset Sen_Max
            MOV  AH , 0AH
            INT  21H                        ; read input

            SUB  BH , BH
            MOV  BL , Sen_Real
            MOV  Sentence[BX] , '$'             ; specify end of input

            MOV  AL , '$'                   ; specify first element in temporary operator stack
            CALL push_op_t

            SUB  CH , CH
            MOV  CL , Sen_Real
            INC  CL
            ADC  CH , 00H
            MOV  BX , offset Sentence           ; point BX to start of input

  evaluate:  MOV  AL , [BX]
                                    ; is read character number?
       char:  AND  AL , 0F0H
```

```
        CMP  AL , 30H
        JNE  not_number
        CALL ins_char
        JMP  next                       ; read next input

not_number:  MOV  AL , [BX]
        CMP  AL , '#'
        JNE  not_char

        MOV  char_fnd , 1
        MOV  isNegative , 1
        JMP  next                       ; read next input

 not_char:  CMP  char_fnd , 0
        JE   peek_op
        CALL push_var

  peek_op:  MOV  BP , bcup_op_t                 ; peek from temporary operand stack to AX
        MOV  AX , [BP]
        MOV  AH , AL                    ; AH holds operand at top of temporary operand stack

        MOV  AL , [BX]                  ; read character is not number , find operator

    ;-------------------------------------------------check for $

    try0:  CMP  AL , '$'
        JNE  try1

    nxt0:  CMP  AH , '$'
        JNE  nxt1
        JMP  next

    nxt1:  CMP  AH , '('
        JNE  nxt2
        MOV  AH , 09H
        MOV  DX , offset MESS1
        INT  21H                        ; report error
        HLT

    nxt2:  CALL move_stck
        JMP  nxt0                       ; recursive

    ;-------------------------------------------------check for + and -

    try1:  CMP  AL , '+'
        JE   nxt3
        CMP  AL , '-'
        JNE  try2

    nxt3:  CMP  AH , '$'
        JNE  nxt4
        CALL push_op_t
        JMP  next
```

```
nxt4:  CMP  AH , '('
       JNE  nxt5
       CALL push_op_t
       JMP  next

nxt5:  CALL move_stck
       JMP  nxt3

;-------------------------------------------------check for * and -

try2:  CMP  AL , '*'
       JE   nxt6
       CMP  AL , '/'
       JNE  try3

nxt6:  CMP  AH , '*'
       JNE  nxt7
       CALL move_stck
       JMP  nxt6

nxt7:  CMP  AH , '/'
       JNE  nxt8
       CALL move_stck
       JMP  nxt6

nxt8:  CALL push_op_t
       JMP  next

;-------------------------------------------------check for (

try3:  CMP  AL , '('
       JNE  try4
       CALL push_op_t
       JMP  next

;-------------------------------------------------check for )

try4:  CMP  AH , '$'
       JNE  nxt9

       MOV  DX , offset MESS1
       MOV  AH , 09H
       INT  21H         ; report error
       HLT

nxt9:  CMP  AH , '('
       JNE  nxt10
       ADD  bcup_op_t , 02H
       JMP  next

nxt10: CALL move_stck
       JMP  try4

;-----------------------------------------------
```

```
  next:  INC  BX          ; read next
         LOOP evaluate

         CALL GetRes       ; calculate result
         CALL DispRes      ; display result


         HLT


 ;functions************************************************

 ins_char  PROC NEAR

         PUSH BX

         MOV char_fnd , 31H                    ; some random none-zero number : set char_fnd

         MOV AL , [BX]
         AND AL , 0FH

         SUB BH , BH
         MOV BL , char_coun
         MOV char_buff[BX] , AL
         INC char_coun

         POP  BX
         RET

 ins_char  ENDP


 ;*******************************************************

 push_var  PROC NEAR

         PUSH BX
         PUSH DX                          ; reset char_fnd

         MOV  char_fnd , 0

         CMP char_coun , 1
         JNE two_char

one_char:  MOV AL , char_buff[0]
         SUB AH , AH
         MOV char_sum , AX

         JMP ENDE
         ;----------------------

two_char:  CMP char_coun , 2
         JNE three_char
```

```
        MOV AL , char_buff[1]
        SUB AH , AH
        MOV char_sum , AX

        MOV AL , char_buff[0]
        MOV AH , 10
        MUL AH
        ADD char_sum , AX

        JMP ENDE
        ;----------------------

three_char:  CMP char_coun , 3
        JNE four_char

        MOV AL , char_buff[2]
        SUB AH , AH
        MOV char_sum , AX

        MOV AL , char_buff[1]
        MOV AH , 10
        MUL AH
        ADD char_sum , AX

        MOV AL , char_buff[0]
        MOV AH , 100
        MUL AH
        ADD char_sum , AX

        JMP ENDE
        ;----------------------

 four_char:  CMP char_coun , 4
        JNE five_char

        MOV AL , char_buff[3]
        SUB AH , AH
        MOV char_sum , AX

        MOV AL , char_buff[2]
        MOV AH , 10
        MUL AH
        ADD char_sum , AX

        MOV AL , char_buff[1]
        MOV AH , 100
        MUL AH
        ADD char_sum , AX

        MOV AL , char_buff[0]
        SUB AH , AH
        MOV DX , 1000                ; caution:destroys DX
        MUL DX                       ; multiply AX by DX :first_digit*1000
```

```
        ADD char_sum , AX               ; since result can't exceed 9000 , we don't need high word

        JMP ENDE
        ;-----------------------

five_char:  MOV AL , char_buff[4]
        SUB AH , AH
        MOV char_sum , AX

        MOV AL , char_buff[3]
        MOV AH , 10
        MUL AH
        ADD char_sum , AX

        MOV AL , char_buff[2]
        MOV AH , 100
        MUL AH
        ADD char_sum , AX

        MOV AL , char_buff[1]
        SUB AH , AH
        MOV DX , 1000                   ; caution:destroys DX
        MUL DX                      ; multiply AX by DX :second_digit*1000
        ADD char_sum , AX               ; since input is assumed to be 16 bit , we don't need high word

        MOV AL , char_buff[0]
        SUB AH , AH
        MOV DX , 10000
        MUL DX                      ; multiply AX by DX :first_digit*10000
        ADD char_sum , AX               ; since input is assumed to be 16 bit , we don't need high word

        ;-----------------------------------

   ENDE:   MOV  char_coun    , 0
        MOV  char_buff[0] , 0
        MOV  char_buff[1] , 0
        MOV  char_buff[2] , 0
        MOV  char_buff[3] , 0
        MOV  char_buff[4] , 0

        CMP  isNegative   , 1
        JNE  goon
        XOR  char_sum     , 0FFFFH        ; form 2's complement of AX
        ADD  char_sum     , 1
        MOV  isNegative   , 0

   goon:  MOV  AX          , char_sum      ;                       Byte0 : charsumL
        MOV  BX          , ptr_queue     ;                       Byte1 : charsumH
        MOV  [BX]        , AX            ; enqueue number to queue       Byte2 : 00H
        ADD  ptr_queue   , 0004H         ; point ptr_queue to next double word   Byte3 : 00H

        POP  DX
        POP  BX
```

```
        RET
push_var  ENDP

 ;*********************************************

push_op_t PROC NEAR

        SUB  bcup_op_t , 02H
        MOV  BP   , bcup_op_t

        SUB  AH   , AH
        MOV  [BP] , AX                  ; push AX to temporary operand stack

        RET
push_op_t ENDP

 ;*********************************************

move_stck PROC NEAR                     ; pops temporary operand stack & enqueues result to operand queue

        PUSH BX
        PUSH CX

        MOV  CL , AL                    ; make a copy of AL
        MOV  BP , bcup_op_t             ; pop operand from temporary operand stack
        MOV  AX , [BP]
        ADD  bcup_op_t   , 02H

        ADD  ptr_queue   , 0002H        ; point   ptr_queue to upper   word        Byte0 : 00H
        MOV  BX          , ptr_queue    ;                                          Byte1 : 00H
        MOV  [BX]        , AX           ; enqueue operand   to operand queue       Byte2 : operator (AL)
        ADD  ptr_queue   , 0002H        ;                                          Byte3 : 00H     (AH)

        MOV  BP , bcup_op_t             ; peek operand from temporary operand stack
        MOV  AX , [BP]
        MOV  AH , AL
        MOV  AL , CL                    ; restore old value of AL

        POP  CX
        POP  BX

        RET
move_stck ENDP

 ;*********************************************

 GetRes  PROC NEAR

        MOV  BP   , offset sp_oprnd

  begin: MOV  AX   , calc_queue
        CMP  AX   ,  ptr_queue
        JNE  calc                       ; calc_queue <= ptr_queue
```

```
        RET

  calc: ADD  calc_queue ,  0002H          ; point calc_queue to upper word
        MOV  BX   , calc_queue
        MOV  CX   , [BX]                ; load upper word to AX
        CMP  CX   , 0
        JE   num_fnd                    ; numbers are in lower words

 op_fnd: CALL GetOp
        ADD  calc_queue ,  0002H          ; point calc_queue to next double word
        JMP  begin

num_fnd: SUB  calc_queue ,  0002H          ; point calc_queue to lower word
        MOV  BX        , calc_queue
        MOV  AX        , [BX]

        SUB  BP        ,  0004H          ; decrement stack pointer (BP)
        MOV  [BP+0]    , AX              ; push number to operand stack
        CMP  AX        , 0
        JGE  zero
        MOV  [BP+2]    , 0FFFFH
        JMP  move

 zero:  MOV  [BP+2]    ,  0000H          ; since inputs are assumed to be 16 bit , upper word is zero

 move:  ADD  calc_queue ,  0004H          ; point calc_queue to next double word
        JMP  begin

 GetRes  ENDP

 ;*********************************************

 GetOp   PROC NEAR                       ; use BP as operand stack SP


        MOV AX      , [BP+0]
        MOV operandL , AX               ; load first  operand to ( operandH operandL ) pair
        MOV AX      , [BP+2]
        MOV operandH , AX

        MOV AX      , [BP+4]            ; load second operand to ( DX       AX       ) pair
        MOV DX      , [BP+6]

 ;------------------------------

 tryadd: CMP CL      , '+'
        JNE trysub

        ADD AX      , operandL
        ADC DX      , operandH
        JMP push_res

 ;------------------------------
```

```
trysub: CMP CL      , '-'
        JNE trymul

        SUB AX      , operandL
        SBB DX      , operandH
        JMP push_res

;-----------------------------

trymul:  CMP CL     , '*'
        JNE trydiv

        CALL UnSign

        MOV [BP+4] , AX                 ; destroys operand stack
        MOV [BP+6] , DX

;[(2^16)*DX+AX]*[(2^16)*High+Low]= (2^32)*DX*High + (2^16)*[(DX*Low)+(AX*High)] + [AX*Low]
=>result is limited to 32 bits so we omit term with 2^32

        MUL  operandL                   ; form AX*Low

        PUSH DX                         ; save    DX (1)
        PUSH AX                         ; save    AX (1)

        MOV  AX , [BP+4]                ; form AX*operandH
        MUL  operandH
                                ; save    DX (2)
        PUSH DX                         ; save    AX (2)
        PUSH AX

        MOV  AX , [BP+6]
        MUL  operandL                   ; form DX*operandL
                                ; restore AX (2)
        POP  BX                         ; restore DX (2)
        POP  CX

        ADD  AX , BX
        ADC  DX , CX                    ; form (AX*High)+(DX*Low)

        CMP  DX , 0
        JNE  dort

        MOV  DX , AX                    ; shift (DX AX) pair by 16 bits
        SUB  AX , AX
        JMP  cont

   cont: POP  BX                        ; restore AX (1)
        POP  CX                         ; restore DX (1)

        ADD  AX , BX
        ADC  DX , CX                    ; form final result
        JMP  chk_sign
```

```
      dort: MOV  AX , 0FFFFH              ; can' represent calculated value , so representing it as max positive
number
        MOV  DX , 07FFFH
        JMP  chk_sign
    ;-----------------------------

    trydiv:  CMP  CL     , '/'
        JNE  tryerr

        CALL UnSign

        MOV  [BP+4] , AX                ; destroys operand stack
        MOV  [BP+6] , DX

        CMP  operandH , 0000H
        JNE  ex_div

     ; (DX AX) / Low
        MOV  AX , DX                    ; divide high word (DX) first
        SUB  DX , DX                    ; convert word to double word
        DIV  operandL
        PUSH AX                         ; store   high word of quotient
        MOV  AX , [BP+4]
        DIV  operandL
                                ; low  word of quotient is in AX
        POP  DX                         ; restore high word of quotient
        JMP  chk_sign


    ; (DX AX) / (High Low) =~ DX / High

     ex_div: MOV  DX , AX
        SUB  DX , DX
        DIV  operandH
        SUB  DX , DX                    ; destroy remainder

        JMP  chk_sign

    ;-------------------------------

    tryerr:  MOV  DX , offset MESS1
        MOV  AH , 09H
        INT  21H
        HLT

   chk_sign:  CMP  isSigned , 1
        JNE  push_res

        CMP  DX , 0                     ; if DX 's MSB is one , then overflow has occured
        JL   over

        XOR  AX , 0FFFFH
        XOR  DX , 0FFFFH
        ADD  AX , 1
```

```
        ADC  DX , 0
        JMP  push_res

    over: MOV  DX , 0000H
        MOV  AX , 8000H
        JMP  push_res

 push_res: ADD  BP     , 0004H              ; push result back to operands stack
        MOV  [BP+0] , AX
        MOV  [BP+2] , DX

        RET

 GetOp    ENDP

 ;**********************************

 UnSign   PROC NEAR

        MOV isSigned , 0                ; reset sign

 check_1:  CMP operandH , 0                ; is 'operand' variable negative?
        JL  negate_1
        JMP check_2

negate_1:  XOR operandL , 0FFFFH             ; 1's complement of operandL
        XOR operandH , 0FFFFH             ; 1's complement of operandH
        ADD operandL , 1                ; 2's complement of
        ADC operandH , 0                ;                operand
        MOV isSigned , 1


 check_2:  CMP DX      , 0                ; is (DX AX)   pair     negative?
        JL  negate_2
        RET                         ; end of procedure

negate_2:  XOR AX , 0FFFFH                ; 1's complement of AX
        XOR DX , 0FFFFH                ; 1's complement of DX
        ADD AX , 1                    ; 2's complement of
        ADC DX , 0                    ;                (DX AX) pair

        ;MOV [BP+4] , AX                ; destroys operand stack
        ;MOV [BP+6] , DX

        CMP isSigned , 0
        JE  signIt
        MOV isSigned , 0                ; (-) * (-) = (+)
        RET                         ; end of procedure

 signIt:  MOV isSigned , 1                ; (+) * (-) = (-)
        RET

 UnSign   ENDP
```

```
;*********************************

DispRes  PROC NEAR

        MOV  operandL  , 0000H          ; making operand zero , so it can't effect IsSign
        MOV  operandH  , 0000H

        MOV  AX        , [BP+0]
        MOV  DX        , [BP+2]

        CALL UnSign

        MOV  ResWord0  , AX
        MOV  ResWord1  , DX

        MOV  CX        , 10
        MOV  BX        , 12             ; 3+9
;------------------------------

again:   SUB  DX        , DX
        MOV  AX        , ResWord1
        DIV  Divisor

        MOV  ResWord1  , AX             ; store AX (high word of new dividend)
        MOV  AX        , ResWord0
                                ; DX already contains correct value
        DIV  Divisor                    ; Move High word of new dividend to ResWord1
        MOV  ResWord0  , AX

        OR   DL        , 30H            ; DX contains decimal number
        MOV  MESS2[BX] , DL

        DEC  BX
        LOOP again

;------------------------------
        CMP isSigned , 01H
        JE  N_Sign

        MOV MESS2[2] , '+'
        JMP done

 N_Sign: MOV MESS2[2] , '-'

   done: MOV DX , offset MESS2
        MOV AH , 09H
        INT 21H
        RET

DispRes  ENDP

;*********************************

CODE     ENDS
```

END START