ΟΜΑΔΑ: Δ15

macros.asm:

```
IS_ODD macro
LOCAL:
        CLC
        TEST AX,01H
        JZ MY_EXIT
        STC
MY_EXIT:
ENDM


SAFE_CALL macro THE_PROC
        pushf           ; store the flags
        push AX
        push BX
        push CX
        push DX

        call THE_PROC

        pop DX
        pop CX
        pop BX
        popf
endm


IS_HEX macro CHAR
LOCAL _HEX, _MYEXIT
        CLC
        CMP CHAR, '0'
        JB, _MYEXIT
        CMP CHAR, '9'
        JBE _HEX
        CMP CHAR, 'A'
        JB, _MYEXIT
        CMP CHAR, 'F'
        JBE _HEX
        CMP CHAR, 'a'
        JB, _MYEXIT
        CMP CHAR, 'f'
        JG _MYEXIT
_HEX:
        STC
_MYEXIT:
```

```
ENDM

;___________________________
;______I/O_________________
;___________________________

BACKSP macro
        PUSH AX
        PUSH DX
        MOV DL, 0x08
        MOV AH, 0x02
        INT 0x21
        POP DX
        POP AX
ENDM


READ macro
        mov ah,8
        int 21h
endm


READ_ECHO macro
        mov ah,01
        int 21h
endm

PRINT macro CHAR
        push ax
        push dx
        mov dl,CHAR
        mov ah,2
        int 21h
        pop dx
        pop ax
endm

PRINT_STR macro STRING
        push ax
        push dx
        push ax
        mov dx, offset STRING
        mov ah,9
        int 21h
        pop ax
        pop dx
```

```
	pop ax
endm

EXIT macro
	mov ax,4c00h
	int 21h
endm

RESETREG macro
	mov ax,0
	mov bx,0
	mov cx,0
	mov dx,0

	mov di,0
	mov si,0
endm
```

2.1.asm:

```
include 'macros.asm'

data segment
	input		db	"GIVE 4 OCTAL DIGITS: $"
	output		db	0AH,0DH,"DECIMAL: $"
	new_line		db	0AH,0DH,"$"
	array_flp		dw	0000h,1250h,2500h,3750h,5000h,6250h,7500h,8750h
ends

;array_flp contains the results of 0/8,1/8,2/8,3/8 etc in hex form but with the right decimal numeric value

; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
stack segment
	dw 128 dup(0)
ends
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
code segment

main proc FAR
	mov ax,data
	mov ds,ax
	mov es,ax

;knowing that each octal digit is up to three bits
;for each one we take as input we are gonna shift it 3 times left in order to
```

```
;prepare it to merge with the next one
;however,since we are gonna take 3 octal sto akeraio meros the msb of the first octal
;is gonna overflow and disappear
;so we deal with this problem by saving the first octal also
;in BH and shift it 2 times right to only keep its msb

start:
        mov ax,0          ;reset all regesters
        mov bx,ax
        mov cx,bx

        print_str input    ;print message input

        call in_oct        ;AL = first octal or 'D'
        cmp al,'D'         ;if user enters 'D' in in_oct proc
        je finish          ;go to finish
        mov ah,al          ;move first octal to AH
        shr ah,2           ;and shift right 2 times in order to keep its msb
        mov bh,ah          ;move AH to BH
        shl al,3           ;shift first octal 3 times left in order to prepare it
        mov bl,al          ;for the next octal and store it to BL

        call in_oct        ;AL = second octal or 'D'
        cmp al,'D'         ;if user enters 'D' in in_oct proc
        je finish          ;go to finish
        or al,bl           ;merge first and second octal to AL
        shl al,3           ;and then shift the new AL 3 times
                           ;in order to prepare it for the next
        mov bl,al          ;move AL to BL

        call in_oct        ;al = third octal or 'D'
        cmp al,'D'         ;if user enters 'D' in in_oct proc
        je finish          ;go to finish
        or al,bl           ;merge them all and put them in BL
        mov bl,al
                           ;now BX has the first three octals
        print '.'

        call in_oct        ;al = forth octal or 'D'
        cmp al,'D'         ;if user enters 'D' in in_oct proc
        je finish          ;go to finish
        mov cl,al          ;move AL to CL
        mov ch,0

        print_str output   ;print output message

        mov ax,bx
        call print_dec      ;print first three octals as decimal
        print '.'

        mov bx,cx           ;double the value of the final octal
   shl bx,1            ;in order to point at the right address of the
        mov cx,array_flp[bx];look up table which contains every case scenario in hex form

        mov al,ch
        call print_hex      ;print the first two hex digits which however are made to
                    ;have the right decimal numeric value :P

        shr cl,4            ;shift right 4 times in order to isolate the third hex digit
        mov dl,cl           ;move CL to DL
        call out_hex        ;and print it as hex digit
        print_str new_line
        print_str new_line  ;go to next line
        jmp start           ;go to start label and start over

finish:
        exit                ;invoke DOS software interrupt
main endp

;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
; ==In_Oct==
; Repeatedly requests a character from keyboard until user
; enters an octal digit. The octal digit is echoed on the screen
; The value of the digit is returned in AL.
; Routine is terminated if user enters 'D'
; MODIFIES: FLAGS, AX
; REQUIRES: <iolib.asm>: PRINT, READ

in_oct proc NEAR
_OIGNORE:
        READ                ;read a character from keyboard
        cmp AL, 'D'  ;if user enters 'D'
        je _OQUIT    ;terminate program
        cmp AL,'0'
        jl _OIGNORE  ;if chr(AL)<'0' or chr(AL)>'9'
        cmp AL,'7'   ;get new input
        jg _OIGNORE
        PRINT AL     ;else print number on screen and
        sub AL, '0'  ;get numeric value
_OQUIT:
        ret
endp
```

```
; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; == out_hex ==
; Prints DL as a hex digit.
; ASSUMES: 0x00 <= DL <= 0x0f
; MODIFIES: FLAGS
; REQUIRES: <iolib.asm>: PRINT
out_hex proc NEAR
    push dx
    cmp DL, 9       ; DL <= 9?
    jle _DEC        ; yes: jump to appropriate fixing code.
    add DL, 0x37    ; no : Prepare DL by adding chr(A) - 10 = 0x37.
    jmp _HEX_OUT    ; ... and go to output stage.
_DEC:
    add DL, '0'     ; Prepare DL by adding chr(0) = 0x30.
_HEX_OUT:
    PRINT DL        ; Print char to screen.
    pop dx
    ret             ; Terminate routine.
endp

; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; == out_hex_byte ==
; Prints AL as 2 hex digits.
; MODIFIES:
; REQUIRES: <numlib.asm>: out_hex
PRINT_HEX proc NEAR
    push ax
    push cx
    push dx
    mov CH, AL      ; Save AL in CH.
    mov CL, 4       ; Set rotation counter.
    shr AL, CL      ; Swap high & low nibble of AL, to print MSH first.
    and AL, 0x0f    ; Mask out high nibble (low nibble is single hex digit).
    mov DL, AL      ; Copy AL to DL.
    call out_hex    ; ... and print as hex.
    mov AL, CH      ; Recover unswapped AL from CH.
    and AL, 0x0f    ; Mask out high nibble (already printed).
    mov DL, AL      ; Copy AL to DL.
    call out_hex    ; ... and print as hex.
    pop dx
    pop cx
    pop ax
    ret             ; Terminate routine.
endp

; ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
; == PRINT_DEC ==
; Prints number in AX as sequence of decimal digits.
; MODIFIES: FLAGS, AX, BX, CX, DX
; REQUIRES: <iolib.asm>: PRINT_UNSAFE

PRINT_DEC proc NEAR
   pusha
   ;mov AX,BX
    mov CX, 0       ; CX will be used as counter for decimal digits.
_DCALC:             ; Digit-calculation loop.
    mov DX, 0       ; Zero DX.
    mov BX, 10      ; Divide DX:AX by 10 to find next decimal digit.
    div BX          ; Quotient in AX, remainder in DX.
    push DX         ; Store decimal digit.
    inc CX          ; Increase digit counter.
    cmp AX, 0       ; Repeat until there are no more digits (AX = 0).
    jnz _DCALC
_DOUT:              ; Digit-printing loop (from MSD to LSD).
    pop DX          ; Pop a decimal digit.
    add DX, '0'     ; Generate ASCII code
    PRINT DL ; ... and print as char.
    loop _DOUT      ; Loop until no decimal digits left (CX = 0).
    popa
    ret             ; Terminate routine.
endp

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

MACROS_PRINTFINAL.ASM:

```
READ_AND_PRINT_CHAR MACRO
   PUSH CX
   PUSH DX
READ:
   MOV AH,8     ;read char
   INT 21H
   CMP AL,30H
   JL  ENTER
   CMP AL,39H
   MOV CL,0          ;If char is number carry = 0 else =1
   JLE PCHAR
   CMP AL,41H
   JL READ
   CMP AL,5AH
   MOV CL,1
   JG READ
   JMP PCHAR
ENTER:
    CMP AL,0DH
    JNZ READ
    MOV AX,4C00H
    INT 21H

PCHAR:
    MOV DL,AL
    MOV AH,2H   ;print char
    INT 21H
    CMP CL,0
    CLC
    JZ LEAVE
    STC
LEAVE:
    POP DX
    POP CX
ENDM


READ macro
            mov ah,8
            int 21h
endm

PRINT_DEC MACRO
    PUSH DX
   PUSH AX
   ADD DL,30H
   MOV AH,2
   INT 21H
   POP AX
   POP DX
ENDM



PRINT macro CHAR
        push ax
        push dx
        mov dl,CHAR
        mov ah,2
        int 21h
        pop dx
        pop ax
endm

PRINT_REG macro cl
        push ax
        push dx
        mov dl,cl
        mov ah,2
        int 21h
        pop dx
        pop ax
endm


PRINT_STR macro STRING
        push ax
        push dx
        mov dx, offset STRING
        mov ah,9
        int 21h
        pop dx
        pop ax
endm


;Printing hex with two digits
PRINT_HEX_MAC macro DL
   PUSH DX     ;saving previous values
   PUSH CX
   PUSH AX
   MOV AL,DL
   MOV CH,2
SRT0:
   MOV CL,4
   ROL DL,CL
   AND DL,00001111B
   CMP DL,09H
   JG  IS_LET
   ADD DL,30H
   JMP OUT1
IS_LET:
   ADD DL,37H

OUT1:
   MOV AH,02H
   INT 21H
   MOV DL,AL
   ROL DL,CL   ;prepare 4 lsbs
   DEC CH
   JNZ SRT0   ;Do the procedure two times first with msbs then with
            ;lsbs

   POP AX
   POP CX
   POP DX
ENDM

;here we print the result of a register pair in hex
PRINTX_HEX macro
   PUSH AX
   PUSH DX
   PUSH CX
   PUSH BX

   MOV BL,00H         ;initial zeros flag
   MOV BH,00H         ;flag that checks if both DH,DL has been printed
   MOV AL,DL          ;most significant reg needs to be printed first
   MOV DL,DH          ;so DH <-> DL
   MOV DH,AL
BEGIN:
   MOV AL,DL
   MOV CH,2           ;printed digit counter
SRT0:
   MOV CL,4           ;flip the digits of DL (get the msb last)
   ROL DL,CL          ;isolate and print it
   AND DL,00001111B
   CMP DL,09H
   JG  IS_LET
   CMP DL,00H          ;if 0 is to be printed check if another number has
   JNZ CONT            ;been printed before it
   CMP BL,00H
   JNZ CONT
   JMP SKIP0
CONT:
   ADD DL,30H
   JMP OUT1
IS_LET:
   ADD DL,37H

OUT1:
   MOV BL,01H          ;a non 0 number has been printed
   MOV AH,02H
   PUSH AX             ;AFTER THE INTR AL <-DL?????????????
   INT 21H
   POP AX
SKIP0:
   MOV DL,AL           ;restore DL
   ROL DL,CL           ;prepare 4 lsbs so when you flip them after srt0 lsb get to bet last
   DEC CH
   JNZ SRT0
   INC BH
   CMP BH,02H          ;if bh=01 then DH has been printed successfully so move DL to DH (we
reverted then in the beggining) and repeat the process
   JZ LEAVE
   MOV DL,DH
   JMP BEGIN
LEAVE:
   CMP BL,00H          ;if DX = 0000 then no number has been printed so print a 0
   JNZ DOL
   PRINT '0'
DOL:
   POP BX
   POP CX
   POP DX
   POP AX
ENDM


IS_ODD macro
LOCAL:
        CLC
        TEST CL,01H
        JZ MY_EXIT
```

```
      STC
MY_EXIT:
ENDM

EXIT macro
  mov ax,4C00H
  int 21H
endm
```

2.2.asm:

```
INCLUDE "MACROS_PRINTFINAL.ASM"
DATA_SEG SEGMENT
                  MSG1 DB 0AH, 0DH, 'GIVE 3 DEC DIGITS: $'
                  MSG2 DB 0AH, 0DH, 'HEX= $'
                  BUF  DW 50 DUP (?)
DATA_SEG ENDS

CODE_SEG SEGMENT
                  ASSUME DS:DATA_SEG, CS:CODE_SEG, SS:DATA_SEG
  MAIN    PROC FAR
        MOV AX,DATA_SEG
        MOV DS,AX
       SRT:
        MOV BX,offset BUF
        PRINT_STR MSG1         ;Prints a string starting in memory address MSG1
        CALL READ_PRINT_DEC      ;Reads a 3 digit dec number and returns it in reg DX andalso
it prints in dec form the inputed number in stdout
        PRINT_STR MSG2
        PRINTX_HEX                      ;Prints hex number in reg DX in HEX form
    JMP SRT
      MAIN    ENDP

  ;Read dec digits from keyboard and store them in an array , if you get 'Q' quit the program
  ;Ignore any other character other than 0-9, enter or Q
  ;If you get enter check if you got at least 3 numbers get those 3 last numbers from the array,
  ;save them in registers in order to print them at the correct order and save the 3 digit number in DX
  READ_PRINT_DEC    PROC NEAR
    MOV CX,00H              ;number of input digits
    MOV DX,00H              ;inputed number will be saved here
   IGNORE:
    MOV AH,08H
    INT 21H
    CMP AL,0DH              ;In = 'enter' ?
    JL IGNORE
    JE CHECK
    CMP AL,30H                          ;In = (0-9) ?
    JL IGNORE
    CMP AL,39H
    JLE ISNUM
    CMP AL,51H             ;if in = 'Q' then exit
    JNE IGNORE
    MOV AX,4C00H
    INT 21H
   ISNUM:
    INC CL
    SUB AX,30H
    MOV [BX],AL
    INC BX
    JMP IGNORE
   CHECK:
    CMP CL,03H             ;got enter so check if i got at least 3 non hex numbers
    JL IGNORE
    MOV CX,100
    DEC BX                ;BX is pointning now in the last position of the array
    MOV DL,[BX-2]            ;Get ms digit and print it then do 100xmsb and store to AX
    PRINT_DEC              ; Repeat this for all the digits
    MOV AH,0
    MOV AL,DL
    MUL CX
    MOV DL,[BX-1]
    PRINT_DEC
    MOV DH,[BX]            ;get least significant digits in order to free BX
    MOV BX,AX
    MOV CL,10
    MOV AH,0
    MOV AL,DL
    MUL CL
    ADD AX,BX
    MOV DL,DH               ;print ls digit
    PRINT_DEC
    MOV DH,0
    ADD DX,AX
    RET
  READ_PRINT_DEC    ENDP
CODE_SEG ENDS
      END MAIN
```

2.3.asm:

```
PRINT_STR MACRO STRING
   PUSH DX
   PUSH AX
   LEA DX,STRING
   MOV AH,09H
   INT 21H
   POP AX
   POP DX
ENDM

EXIT MACRO
   MOV AH,4CH
   INT 21H
ENDM

PRINT MACRO CHAR
   PUSH DX
   PUSH AX
   MOV DL,CHAR
   MOV AH,02H
   INT 21H
   POP AX
   POP DX
ENDM

READ MACRO
   MOV AH,08H
   INT 21H
ENDM

DATA SEGMENT
   MSG DB 0AH,0DH,'GIVE UP TO 16 CHARACTERS: $'
   INPUT DB 16 DUP(?)
   NEW_LINE DB 0AH,0DH, '$'
ENDS

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

MAIN PROC FAR
   MOV AX,DATA
   MOV DS,AX

ENTRY:
   PRINT_STR MSG
   MOV CX,10H     ; CX=16

INIT:               ; INITIALIZATION
   LEA DI,INPUT   ; DI POINTS TO INPUT
   ADD DI,CX      ; DI POINTS TO INPUT+16
   DEC DI         ; DI POINTS TO INPUT+15 (LAST MEM LOCATION)
   MOV AL,00H        ; INITIAL VALUE = 0
   MOV  [DI],AL    ; [DI] = 0
   LOOP INIT       ; [INPUT .. INPUT+15] = 0

CONTINUE:
   MOV CX,0                        ; INIT
   MOV BX,0
   MOV DX,0
   LEA DI,INPUT    ; DI POINTS TO INPUT

READ_NEXT:
   READ            ; READ ASCII CHAR INTO AL REGISTER
   CMP AL,0DH      ; ENTER PRESSED ?
   JE RESULT      ; IF YES THEN SHOW RESULT

   CMP AL,'*'      ; IF * TERMINATE
   JE TERMINATE

   CMP AL,20H      ; WHITESPACE
   JE SAVE

   CMP AL,'0'      ; IF LESS THAN 0 (ASCII)
   JL READ_NEXT   ; READ AGAIN
   CMP AL,39H      ; IF LESS THAN OR EQUAL TO '9'
   JLE SAVE_DIG    ; AL BETWEEN [30H .. 39H] (DIGITS)

   CMP AL,'A'      ; UPPERCASE
   JL READ_NEXT   ; IF LESS THAN 41H READ AGAIN
   CMP AL,'Z'      ; IF LESS THAN OR EQUAL TO 5AH
   JLE SAVE        ; SAVE UPPERCASE

   CMP AL,'a'      ; LOWERCASE
   JL READ_NEXT   ; IF < 61H READ AGAIN
   CMP AL,'z'      ; IF > 7AH READ AGAIN
   JG READ_NEXT

SAVE:                 ; SAVE CHARACTERS
   PRINT AL
   MOV [DI],AL
   INC DI            ; MEMORY LOCATION INDEX
   INC BL            ; COUNTER
   CMP BL,16
   JL READ_NEXT
   JMP WAIT1         ; IF >= 16 CHARS GIVEN WAIT FOR ENTER OR *

SAVE_DIG:            ; SAVE DIGITS
   PRINT AL
   MOV [DI],AL     ; SAVE CHARACTER IN INPUT BUFFER [DI]
   INC DI          ; INCREMENT MEMORY INDEX LOCATION
   CMP BH,0        ; MIN1
   JE MIN1        ; BH == 0 => FIND MIN1
   CMP BH,1        ; MIN2
   JE MIN2        ; BH == 1 => FIND MIN2
   JMP DIG_OUT

MIN1:
   MOV CH,AL       ; STORE MIN1 IN CH

MIN2:
   MOV CL,AL       ; STORE MIN2 IN CL
   PUSH CX         ; SAVE (MIN1,MIN2) IN STACK

DIG_OUT:
   INC BH          ; BH {0,1: MIN NOT FOUND, 2: MIN1 FOUND, 3+: MIN1,MIN2 FOUND}
   INC BL          ; CHAR COUNTER ++
   CMP BL,16       ; COUNTER < 16
   JL READ_NEXT   ; CONTINUE READING


WAIT1:             ; READ 16 CHARACTERS NO MORE SPACE
   READ            ; READ ASCII CHAR INTO AL REGISTER
   CMP AL,0DH      ; ENTER RECEIVED
   JE RESULT

   CMP AL,'*'      ; TERMINATION SIGNAL
   JE TERMINATE

   JMP WAIT1

RESULT:
   CMP BL,0
   JE ENTRY
   PRINT_STR NEW_LINE
   LEA DI,INPUT     ; POINT TO [INPUT]
   PUSH BX          ; SAVE TO STACK (MIN_STATUS, NUM_OF_CHARS)

UPPERCASE:          ; BH:BL HOLDS NUMBER OF UPPERCASE LETTERS
```

```
   CMP BL,0         ; WHILE BL != 0 REPEAT
   JE GROUP1       ; IF BL == 0 NEXT GROUP
   DEC BL           ; DECREMENT
   MOV DL,[DI]      ; LOAD MEMORY
   INC DI           ; NEXT MEMORY LOCATION

   CMP DL,'A'       ; WHILE DL NOT IN [0x65 .. 0x5A]
   JL UPPERCASE     ; NEXT UPPERCASE
   CMP DL,'Z'
   JG UPPERCASE     ; NEXT UPPERCASE

VALID_UPPERCASE:
   PRINT DL
   JMP UPPERCASE     ; NEXT UPPERCASE

GROUP1:               ; GROUP UPPERCASE HAS NO MORE MEMBERS
   POP BX            ; RESTORE (MIN_STATUS, NUM_OF_CHARS) FROM STACK
   PRINT '-'         ; NEXT GROUP: LOWERCASE
   LEA DI,INPUT

   PUSH BX         ; SAVE TO STACK (MIN_STATUS, NUM_OF_CHARS)
LOWERCASE:          ; PROCESS GROUP LOWERCASE
   CMP BL,0
   JE GROUP2       ; GROUP LOWERCASE HAS NO MORE MEMBERS
   DEC BL          ; COUNTER--
   MOV DL,[DI]
   INC DI          ; NEXT MEMORY LOCATION

   CMP DL,'a'      ; IF [MEM] NOT IN [0x61 .. 0x7a]
   JL LOWERCASE    ; CHECK NEXT
   CMP DL,'z'
   JG LOWERCASE    ; CHECK NEXT IF LOWERCASE

VALID_LOWERCASE:
   PRINT DL        ; VALID LOWERCASE PRINT IT
   JMP LOWERCASE   ; PROCESS NEXT LOWERCASE

GROUP2:             ; GROUP:LOWERCASE HAS NO MORE MEMBERS
   POP BX          ; RESTORE (MIN_STATUS, NUM_OF_CHARS) FROM STACK
   PRINT '-'

   LEA DI,INPUT    ; POINT TO START OF INPUT MEMORY LOCATION
   POP CX          ; RESTORE (MIN1,MIN2) FROM STACK
   MOV BH,0        ; MIN_STATUS=NOT_YET_FOUND (INIT)

NUM:               ; GROUP:NUMBERS
   CMP BL,0
   JE PRINT_RES   ; NOTHING TO PROCESS => EXIT
   DEC BL         ; COUNTER--
   MOV DL,[DI]    ; LOAD MEMORY LOCATION
   INC DI         ; MEMORY INDEX LOCATION ++

   CMP DL,'0'     ; [30H .. 39H] == ASCII[0 .. 9]
   JL NUM         ; IF < 0 PROCESS NEXT
   CMP DL,'9'
   JG NUM         ; IF > 9 PROCESS NEXT

VALID_NUM:        ; NOT SKIPPED - THEREFORE DL HOLDS A VALID DIGIT
   INC BH         ; BH++
   PRINT DL       ; PRINT DIGIT
   CMP BH,2       ; {1: NO MIN FOUND YET, 2: ONLY MIN 1 FOUND, 3+: MIN 1,2 FOUND}
   JLE NUM

   ; CX HOLDS (MIN1,MIN2)
   CMP CL,DL      ; MIN2 <= CURRENT
   JLE CHECK      ; IF TRUE GOTO CHECK

   ; AT THIS POINT # MIN2 > CURRENT #
   CMP CH,CL    ; IS MIN1 <= MIN2 ?
   JLE CHECK2   ; IF TRUE UPDATE MIN2 => (MIN2 := CURRENT)

   ; AT THIS POINT # MIN1 > MIN2 && CURRENT < MIN2 #
   MOV CH,CL   ; MIN1 := MIN2
   MOV CL,DL   ; MIN2 := CURRENT

   JMP NUM     ; PROCESS NEXT NUMBER

CHECK:         ; # CURRENT >= MIN2 #
   CMP CH,DL   ; MIN1 <= CURRENT ?
   JLE NUM     ; IF TRUE THEN PROCESS NEXT NUMBER

   ; # MIN1 > CURRENT #
   CMP CL,DL   ; IS MIN2 == CURRENT
   JE NUM      ; IF TRUE THEN PROCESS NEXT NUMBER

   ; # MIN1 > CURRENT && MIN2 > CURRENT #
   MOV CH,CL   ; MIN1 := MIN2
   MOV CL,DL   ; MIN2 := CURRENT

   JMP NUM     ; PROCESS NEXT NUMBER

CHECK2:
   MOV CL,DL   ; MIN2 := CURRENT
   JMP NUM     ; PROCESS NEXT NUMBER
```

```
PRINT_RES:
    PRINT_STR NEW_LINE  ; NEWLINE
    PRINT CH            ; MIN1
    PRINT CL            ; MIN2
    PRINT_STR NEW_LINE  ; NEWLINE

    JMP ENTRY           ; REPEAT

TERMINATE:
    EXIT    ; MACRO EXPANSION
MAIN ENDP

CODE ENDS
END MAIN
```

Notes: Για την άσκηση αυτη, ορίσαμε εναν (1) πίνακα για αποθήκευση του INPUT και εκτελουμε τρεις (3) επαναλήψεις για την δημιουργία των αντίστοιχων ομάδων. Αυτό μας δίνει πολυπλοκότητα αλγορίθμου $O(n^3)$.
Αναπτύξαμε και μια optimized εκδοχή της υλοποίησης αυτής με χρήση 3 πινάκων, όπου ανάλογα με την ομάδα που ανήκει ο χαρακτήρας που διαβάστηκε αποθηκεύεται στον κατάλληλο πίνακα, και κρατάμε το index για τον κάθε πίνακα σε ξεχωριστή θέση μνήμης. Αυτό μας δίνει συνολική πολυπλοκότητα της τάξης του O(n) – ωστόσο παρουσιαζουμε τον κωδικα της μη-optimized εκδοχης εφ'οσον αυτην παρουσιάσαμε στο εργαστήριο.
Η συγκριση για τους δυο (2) μικρότερους αριθμους γινεται κρατώντας τους σε μια "tuple" δηλαδή BX = BH:BL όπου BH =:= MIN1 και BL =:= MIN2 στο LABEL CHECK.
Επιπρόσθετη επεξήγηση του κώδικα υπάρχει σε μορφή σχολίων διπλα απο κάθε εντολή.

2.4.asm:

```
INCLUDE "MACROS_PRINTFINAL.ASM"

DATA    SEGMENT
   NEWL    DB  0AH,0DH,'$'
DATA  ENDS

CODE    SEGMENT
      ASSUME DS:DATA,CS:CODE,SS:DATA
   MAIN                              PROC FAR

      MOV AX,DATA
      MOV DS,AX
    START:
         MOV AX,00H
         MOV CX,00H                     ;input number counter
      MOV BX,0FFFFH                     ;checks if a number has been input before +, - or =
      CALL READ_PRINT_DIGITS            ;Reads and prints up to a 4 digit decimal number from
keyboard, stores it in register DX.
      MOV AX,DX                      ;Also it reads an operand '+' or '-' and stores it to CL (the first time
that is called)
      MOV BX,0FFFEH
      CALL READ_PRINT_DIGITS
      CALL CALC_AND_OUTPUT              ;It calculates the result of the operation AX (+ or -) DX
and outputs the
                                ;result in both hex and decimal forms
      JMP START                      ;The function is continuous and terminates when the letter 'M' is read
as input
   MAIN ENDP

   ;Here we get an at most 4 digit decimal number and an operator and return both of them in DX and
CL respectively
   ;If we get and operator we need to check if at least a digit has been input before it and if the operator
is fitting
   ;in the operation that we perform (e.g. the input needs to be <num1> (operator) <num2> '=', where
operator is '+' or '-')
   READ_PRINT_DIGITS PROC NEAR
      PUSH AX
      MOV DX,0            ;Current number holder
      MOV CH,04H                         ;Digit input reverse counter
    SRT:
      MOV AH,08H           ;Read input number
      INT 21H
      CMP AL,2BH           ;Check to see if input is between 0-9 or is '+', '-','=' or 'M ....here in = '+' ?
      JL SRT
```

```
        JG NXT                ;Else its equal with 2Bh
        CMP CL,'+'            ;if input= '+' and CL='+' or '-' then its the second time the routine is called
        JZ SRT                ;the only acceptable operand is '='
        CMP CL,'-'
        JZ SRT
        CMP BX,0FFFFH                                                                        ;If BX
hasnt changed then no number has been input as the first value but in ='+'
        JZ SRT
        MOV CL,AL
        PRINT CL
        JMP RETURN
     NXT:
        CMP AL,2DH            ;in = '-' ?
        JL SRT
        JG NUM
        CMP CL,'+'            ;if input= '-' and CL='+' or '-' then its the second time the routine is called
        JZ SRT                ;the only acceptable operand is '='
        CMP CL,'-'
        JZ SRT
        CMP BX,0FFFFH                                                                  ;If BX hasnt
change then no number has been input as the first value but in ='+'
        JZ SRT
        MOV CL,AL
        PRINT CL
        JMP RETURN
     NUM:
        CMP AL,30H            ;in = [0,9] ?
        JL SRT
        CMP AL,39H
        JLE ISNUM
        CMP AL,3DH            ;in = '='?
        JG CHECK_EXT
        JL SRT
        CMP CL,0              ;if in = '=' and its the first time this routine is called (CL=0) its not a valid
operand
        JZ SRT
        CMP BX,0FFFEH                                                                  ;Also if no
valid value hasbeen input its incorrect
        JZ SRT
        PRINT '='
        JMP RETURN
     CHECK_EXT:               ;if in = 'M' then exit the program
        CMP AL,4DH
        JNZ SRT
        MOV AX,4C00H
        INT 21H
     ISNUM:
           CMP CH,00H            ;if 4 numbers have been read ,wait for a char to be input
           JZ SRT                                    ;If we have a new number then multiply the number that
we have x10 and then add the new number and store
        MOV AH,0             ;it to DX
        PRINT AL
        SUB AL,30H
        MOV BX,AX
        MOV AX,DX
        MOV DX,10
        MUL DX
        ADD AX,BX
        MOV DX,AX
        DEC CH
        JMP SRT
      RETURN:
        POP AX
        RET
    READ_PRINT_DIGITS ENDP


    ;here we ADD or SUBTRACT the two numbers store the result in AX
    ;then check the CArry and print '-' if its 1
    ;and then print the number in hex form and then decimal as described below
    CALC_AND_OUTPUT PROC NEAR
           MOV CH,00H                                         ;Negative sub flag
           CMP CL,'+'
           JNZ MNS
           ADD AX,DX
           JMP OUTP
      MNS:
           SUB AX,DX
           JNC OUTP
           MOV CH,01                             ;if AX-DX <0
           NEG AX                                             ;complete of 2 of the reg AX
      OUTP:
        CMP CH,00H
        JZ NO_PR
        PRINT '-'
      NO_PR:
        MOV DX,AX
           PRINTX_HEX
           PRINT '='
           CMP CH,00
           JZ PRDEC
           PRINT '-'
             PRDEC:                   ;In order to print in decimal form i  have then number in AX and
continuously divide it by 10
```

```
            MOV DX,0                    ;thus getting the last digit each time and pushing in to the stack and the
quotient in AX
                        MOV BX,10                   ;I count the number of digits i have in CX and then i'm printing
the digits top down from the stack
            MOV CX,0                    ;in a loop
              PUSH_NUM:
            INC CX
            DIV BX
            PUSH DX
            CMP AX,0
            JZ DO_PRINT
            MOV DX,0
            JMP PUSH_NUM
      DO_PRINT:
            POP DX
            PRINT_DEC
            LOOP DO_PRINT
            PRINT_STR NEWL
            RET
            CALC_AND_OUTPUT ENDP
CODE ENDS
    END MAIN
```