ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

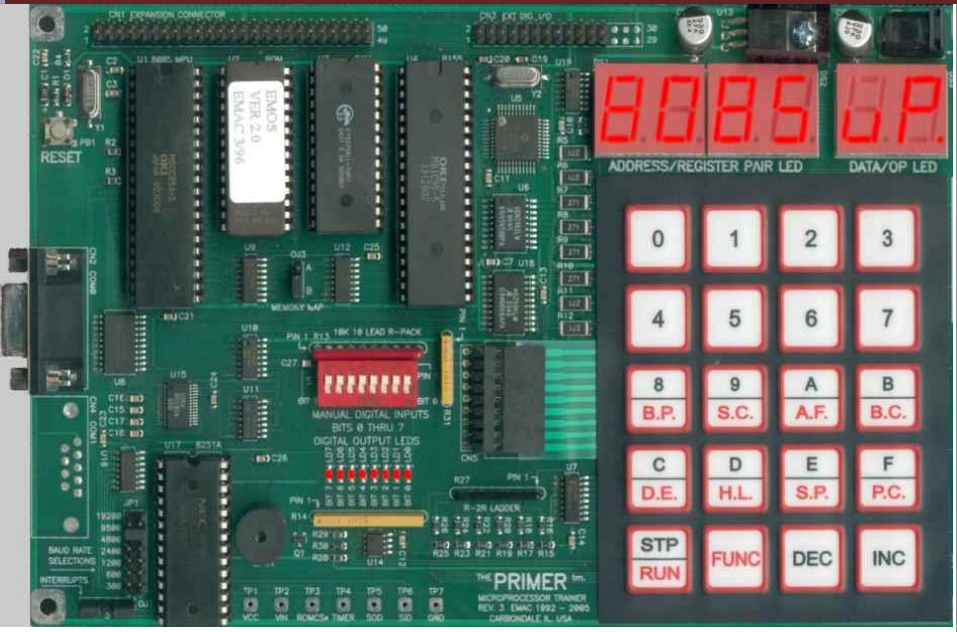
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

&

ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Συστήματα Μικροϋπολογιστών

$5^{η}$ Σειρά Ασκήσεων



6^{O} Εξάμηνο

ΡΟΗ Υ

Αθανασίου Νικόλαος

ΑΜ 03112074

Σταυρακάκης Δημήτριος

ΑΜ 03112017

«Δεν απαιτήθηκε η λύση όλων των ασκήσεων γι' αυτό λύθηκαν αναλυτικά και σε μορφή εκτελέσιμου κώδικα οι 1,2,4,5 και παρατέθηκε ενδεικτική λύση της άσκησης 3. »

Άσκηση 1

α)

```
;exercise 1.a




ADDRN = 0800h        ;some random numbers

N = 8


    mov bx,ADDRN   ;load the starting address


set_values:         ;we set our values in memory so as to check if our program works properly

    mov [bx],50    ;even

    mov [bx+1],47  ;odd

    mov [bx+2],27  ;odd

    mov [bx+3],80  ;even

    mov [bx+4],96  ;even

    mov [bx+5],137 ;odd

    mov [bx+6],84  ;even

    mov [bx+7],7   ;odd


    mov cl,N ;cl = 8

    mov ch,0 ;ch = 0

    mov dx,0 ;register for our sum

    mov ah,0 ;register for the number of even values
```

```
 mainloop:              ;loop for computing our sum
     mov al,[bx]     ;start from the bx address
     inc bx          ;point to the next memory address
     rcr al,1        ;right shift 1
     jc isodd        ;if last bit is 1 then the value is odd so dont put it in the sum
     rcl al,1        ;if it's not odd restore the starting value by shifting 1 left
     push ax         ;push double reg a in the stack
     mov ah,0
     add dx,ax       ;dx=dx+ax
     pop ax
     inc ah          ;if value is even then ah=ah+1
isodd:
     loop mainloop


     mov cl,ah       ;calculation of the average
     mov ch,0        ;double reg c now has the number of the sumed values
     mov ax,dx       ;move the sum to double reg a


     cmp cl,0        ;if we dont have any even values or the sum is equal to zero then
     jz zero         ;jump to zero and print 0
     mov dx,0
     div cx          ;else make the division
     call hexprint   ;and print the result in hex mode
     jmp end         ;go to the end and halt
zero:
     mov al,'0'      ;print 0 on screen
     mov ah,0eh
```

```
        int 10h
end:
        hlt

        hexprint proc

        push dx         ;this procedure prints the desired result on screen
        push cx         ;which is in the double reg a
        mov dx,0
        cmp ax,0h
        je finish
        mov cx,10h
        div cx
        push dx
        call hexprint   ;recursion
        pop dx
        mov ax,dx
        cmp ax,0ah
        jnc letter
        add ax,30h      ;add 30h to print the number
        jmp number
letter:
        add ax,37h      ;add 37h to print the letter
number:
        mov ah,0eh
        int 10h         ;print
finish:
        pop cx

        pop dx

        ret


        endp      ;it correctly prints 4Dhex which is 77dec, our average is 77.5
```

β)

```
;exercise 1.b

ADDRN = 0800h         ;some random numbers

N = 8

  mov bx,ADDRN


set_values:           ;we set our values in memory so as to check if our program works properly

    mov [bx],50    ;even

    mov [bx+1],47  ;odd

    mov [bx+2],27  ;odd

    mov [bx+3],80  ;even

    mov [bx+4],96  ;even

    mov [bx+5],137 ;odd

    mov [bx+6],84  ;even

    mov [bx+7],7   ;odd




    mov cl,N       ;cl=8

    mov ch,0       ;ch=0

    mov dl,0ffh    ;max possible value since it is composed by 8 bits

    mov dh,0h      ;min possilbe value since it is composed by 8 bits
```

```
checkloop:
        mov al,[bx]     ;load starting address and
                        ;check if a value is greater than max or lower than min
        inc bx          ;point to the next memory address
        cmp al, dl
        jc min          ;if we find new min store it in dl
min_max:
        cmp al, dh
        jnc max         ;if we find new max store is in dh
        jmp end
min:
        mov dl,al
        jmp min_max
max:
        mov dh,al
end:
        loop checkloop
        mov ah,0        ;ah=0
        mov al,dh       ;al=maxvalue
        call hexprint   ;print max on screen
        mov al,' '      ;al=' '
        mov ah,0eh      ;ah=0eh
        int 10h         ;print the space that is required
        mov ah,0h       ;ah=0
        mov al,dl       ;al=minvalue
        call hexprint   ;print min on screen
        hlt
```

```
      hexprint proc
      push dx        ;this procedure prints the desired result on screen
       push cx        ;which is in the double reg a
       mov dx,0
       cmp ax,0h
       je finish
       mov cx,10h
       div cx
       push dx
       call hexprint   ;recursion
       pop dx
       mov ax,dx
       cmp ax,0ah
       jnc letter
       add ax,30h      ;add 30h to print the number
       jmp number
letter:
       add ax,37h      ;add 37h to print the letter
number:
       mov ah,0eh
       int 10h         ;print
finish:
       pop cx
       pop dx
       ret


       endp          ;it correctly print 89h=137dec and 7h=7dec with a space between them
```

Άσκηση 2

```
;exercise 2


    ON_SCREEN macro screen

    push ax         ;macro that prints each message

    push dx

    mov dx, offset screen

    mov ah, 9

    int 21h

    pop dx

    pop ax

    endm

begin:  mov dl,0     ;flag for success of failure

    ON_SCREEN first_number     ;ask for the first number

    ;first digit of the first number

    mov ah,1                ;get the first number

    int 21h


    cmp al,48      ;check if our input is a valid digit

    jc fail1       ;ascii code between 48 and 58 (we include zero as first digit)

    cmp al,58      ;if it's true correct input else wrong

    jnc  fail1     ;and print message

    jmp success1


fail1:

    mov dl,1
```

```
success1:

        sub al,30h

        mov bh,10

        mul bh

        mov bh,al  ;mul first digit x10 and store it in bh

        ;second digit of the first number

        mov ah, 1

        int 21h

        cmp al,48

        jc fail2

        cmp al,58

        jnc  fail2

        jmp success2


fail2:

        mov dl,1


success2:

        sub al,30h

        add bh,al               ;add the second digit to bh now first number is ready in bh

        ON_SCREEN enter         ;print the \n

        ON_SCREEN second_number ;ask for the second number

        ;first digit of the second number

        mov ah, 1           ; get the second number

        int 21h
        cmp al,48           ;check is our input is a valid digit
        jc fail3     ;ascii code between 48 and 58 (we include zero as first digit)
        cmp al,58           ;if it's true correct input else wrong
        jnc  fail3   ;and print message
        jmp success3

fail3:
        mov dl,1

success3:
        sub al,30h
        mov bl,10
        mul bl      ;mul first digit x10 and store it in bl
        mov bl,al
        ;second digit of the second number
        mov ah, 1
        int 21h
        cmp al,48
        jc fail4
        cmp al,58
        jnc  fail4
        jmp success4

fail4:
        mov dl,1
```

```
success4:
        sub al,30h
        add bl,al   ;add the second digit to bl now second number is ready in bl

        ON_SCREEN enter                 ;print the \n
                                       ;first number in bh
        ON_SCREEN printx                ;second number in bl, print x value

        cmp dl,1
        je fail5
        mov al,bh   ;put first number in al and print its decimal value
        call decprint
        jmp success5

fail5:
        ON_SCREEN bar

success5:
        ON_SCREEN printy               ;print y value
       cmp dl,1    ;check for any failure
        je fail6
        mov al,bl   ;put second number in al and print its decimal value
        call decprint
        jmp stand_by;success6

fail6:
        ON_SCREEN bar
```

```
stand_by:                ;stand by for a user's enter

    mov ah,00h

    int 16h

    cmp al,0dh

    jne stand_by


    cmp dl,1          ;check for valid inputs

    je invalid        ;if not print an appropriate message


    ON_SCREEN enter

    push bx

    mov bl,bh

    mov bh,0

    mov cx,bx

    pop bx

    push bx

    mov bh,0

    add cx,bx

    mov ax,cx

    pop bx


    ON_SCREEN plus  ;print x+y


    mov ax,cx

    call hexprint   ;print the sum  in hex mode
```

```
        ON_SCREEN minus ;print x-y



        push bx                  ;check for negative or positive values
        mov bl,bh
        mov bh,0
        mov cx,bx
        pop bx
        mov bh,0
        sub cx,bx
        mov ax,cx
        test ax,ax               ;perform AND for double reg a with itself
        jns greater_than_zero
        push ax                  ;to check if result of sub is < or > 0
        mov al, '-'              ;if its not print minus
        mov ah,0eh
        int 10h
        pop ax
        neg ax

    greater_than_zero:
        cmp ax,0
        je zero             ;check if x-y equals to 0
        call hexprint       ;print x-y in hex mode
        jmp continue
```

```
zero:
    mov al,'0'          ;print '0' if x-y equals to zero
    mov ah,0eh
    int 10h

continue:
    ON_SCREEN enter
    jmp begin

    hlt

invalid:
    ON_SCREEN wrong      ;invalid input
    jmp begin            ;restart again


    first_number db "first number:  $"
    second_number db "second number:  $"
    enter db 0Dh,0Ah, "$"
    printx db "x=$"
    printy db "  y=$"
    plus db "x+y=$"
    minus db " x-y=$"
    wrong db 0Dh,0Ah,"Invalid Input",0Dh,0Ah,"$"
    bar db "-$"
    hexprint proc

    push dx     ;this procedure prints the desired result on screen in hex mode
    push cx
    mov dx,0
    cmp ax,0h
    je finish
    mov cx,10h
    div cx
    push dx
    call hexprint   ;recursion
    pop dx
    mov ax,dx
    cmp ax,0ah
    jnc letter
    add ax,30h      ;add 30h to print the number
    jmp number
letter:
    add ax,37h      ;add 37h to print the letter
number:
    mov ah,0eh
    int 10h         ;print
finish:
    pop cx
    pop dx
    ret
    decprint proc   ;prints the result in decimal
    push dx
    push cx
    push ax
    mov ah,0
    mov dx,0
    cmp al,0h
    je dec_finish
    mov cx,10
    div cx
    push dx
    call decprint
    pop dx
    mov al,dl
    cmp al,0ah
    jnc NaN
    add al,30h
    jmp dec_number
NaN:
    add al,37h
dec_number:
    mov ah,0eh
    int 10h
dec_finish:
    pop ax
    pop cx

    pop dx

    ret


    endp
```

Άσκηση 3

Ξεκινάμε με την κλασσική μακροεντολή που χρειαζόμαστε για την εκτύπωση μυνήματος που θα χρησιμοποιηθεί στην υλοποίηση των ζητούμενων μακροεντολών.

```
;exercise3

PRINT macro message

    push ax         ;macro that prints a message

    push dx         ;on screen

    mov dx, offset message

    mov ah, 9

    int 21h

    pop dx

    pop ax

    endm
```

μετά ελέγχουμε αν ειναι το πρώτο δοθέν ψηφίο 0 γιατί θέλουμε δηψήφιους με cmp με το 0

και ζητάμε έγκυρη είσοδο με μύνημα του τύπου “enter valid decimal number:”

έπειτα τύπωνουμε τον αριθμό αναδρομικά σε δεκαδική , δεκαεξαδική , οκταδική και δυαδική μορφή

Με κόκκινο παρατίθενται οι αλλαγές ετικετών και οι προσθήκες εντολών για τύπωμα της εκάστοτε μορφής της εισόδου.

```
print_dec proc

    push dx

    push cx

    push bx
```

```
        push ax
        mov bh,0
        mov dx,0
        cmp bl,0h
        je proc_end_dec
        mov cx,10      ;base 10
       mov cx,10h    ;base 16
       mov cx,8    ;base 8
       mov cx,2   ;base 2
       mov ax,bx
        div cx
        mov bx,ax
        push dx
        call print_dec/hex/oct/bin
        pop dx
        mov bl,dl
        cmp bl,0ah
        jnc not_number_dec/hex/oct/bin
        add bl,48
        jmp number_dec/hex/oct/bin
not_number_dec/hex/oct/bin:
        add bl,55
number_dec/hex/oct/bin:
        mov al,bl
        mov ah,0eh
        int 10h
proc_end_dec:
        pop ax

        pop bx

        pop cx

        pop dx

        ret
```

Επιπρόσθετα χρησιμοποιούνται κάποια ακόμα labels για έλεγχο εγκυρότητας της εισόδου και σβήσιμο ακατάλληλων χαρακτήρων τα οποία χρησιμοποίηθηκαν σχεδόν αυτούσια και στις υπόλοιπες ασκήσεις.

Άσκηση 4

```
        ;Exercise 4


        PRINT macro message

        push ax                 ;macro that prints a message

        push dx                 ;on screen

        mov dx, offset message  ;for the messages we need to print

        mov ah, 9

        int 21h

        pop dx

        pop ax

        endm



begin:  mov cl,20

        mov dl,20

        mov bx,0800h

read:                   ;read  the valid input 0-9 a-z

        call read_char  ;and store them through bx
```

```
    mov [bx],al             ;in memory
    inc bx
    loop read

    PRINT newline
    mov cl,dl
    mov bx,0800h

write:                      ;print the numbers as they are
    mov al,[bx]             ;and change letters to
    inc bl                  ;capital letters from
    cmp al,97               ;small
    jc arithmos
    cmp al,123
    jnc arithmos
    sub al,32
arithmos:
    mov ah,0eh
    int 10h
    loop write

    PRINT newline
    jmp begin
end_program:
    hlt

    newline db 0Dh,0Ah, "$" ;messages
```

```
  erase_char proc     ;this routine erases the last printed
  push ax             ;character
  mov ah,0eh
  mov al,8
  int 10h
  mov al,32
  int 10h
  mov al,8
  int 10h
  pop ax
  ret



  read_char proc       ;reads one character as input
read_again:                ;and stores it if it's ok (a-z)(0-9)
  mov ah, 1           ;if it's not reads again .
  int 21h
  cmp al,61           ;if input is "=" 61 in ascii table end program
  je end_program
  cmp al,13           ;if input is ENTER (ascii code 13) we stop reading
  jne wait_enter       ;by making cl=1 and store the number of the characters
  mov dl,20           ;we have read in dl through substraction
  sub dl,cl           ;so we can print them
```

```
    mov cl,1
    cmp dl,0          ;check if enter pressed with no valid characters pressed
    jne wait_for_correct_input     ;to restart
    PRINT newline
    jmp begin
wait_for_correct_input:
    jmp correct
wait_enter:
    cmp al,48
    jc wrong                ;we check if the input is wrong
    cmp al,58                ;for the numbers
    jnc check_if_wrong_input
    jmp correct
check_if_wrong_input:
    cmp al,97
    jc wrong                ;and for the letters
    cmp al,123
    jnc wrong
    jmp correct
wrong:
    call erase_char     ;erase the invalid character
    jmp read_again
correct:
    ret


    endp
```

Άσκηση 5

```
;Exercise 5


PRINT macro message

push ax              ;macro for the messages message

push dx              ;START (..): ..

mov dx, offset message  ;errors  etc


mov ah, 9

int 21h

pop dx

pop ax

endm





PRINT startornot

begin_decision:

mov ah, 1           ;get Y(y) to begin the execution of the program

int 21h             ;or N(n) to hlt and below are the

                    ;management of each input hlt or continue

cmp al,78           ;N

je end_of_execution

cmp al,110          ;n

je end_of_execution

cmp al,89           ;Y

je start

cmp al,121          ;y
```

```
    je start

    call erase_char

    jmp begin_decision


                ;every time we need to print on the screen

                ;we call the proper print procedure

                ;eg. for hex numbers in our example

                ;or for characters like -

                ;the same thing if we want to read

                ;from user as in start

start:

    PRINT newline

restart:

    mov ch,0

    mov dx,0

    PRINT input

    call read_hex     ;we want 3 hexadecimal digits so we call

    mov bl,al         ;read_hex 3 times

    call read_hex

    mov bh,al

    call read_hex

    mov cl,al


    mov dh,bl         ;here we put the contents from bx(bh,bl)

    mov ax,10h        ;and cl to dx as one hex number

    mul bh

    add dx,ax
    add dx,cx       ;dx=x


    cmp dx,0BB8h

    jnc check1

    mov ax,5

    mul dx          ;y (Temperature) is y=5/3*x from 0(0h) to 3(BB8h) Volts

    mov bx,3

    div bx          ;ax=y

    jmp tupwse

check1:

    mov ax,5

    mul dx          ;y (Temperature) is y=5*x-10000 from 3 to 4 Volts (BB8h-FA0h)

    sub ax,10000    ;ax=y

tupwse:

    cmp ax,270Fh    ;check if the temperature reaches 999.9

                ;999.9d=270Fh

    jnc lathos

    PRINT newline   ;print integral part

    mov bx,10

    mov dx,0        ;check for possible jumps

    div bx

    cmp ax,0

    je check2

    call print_ax_dex

    jmp dekadiko

check2:

    mov ah,0eh
    mov al,'0'

    int 10h


dekadiko:

    mov ah,0eh

    mov al,'.'

    int 10h


    mov ax,dx          ;print dekadiko

    cmp ax,0

    je check3

    call print_ax_dex

    jmp again

check3:

    mov ah,0eh

    mov al,'0'

    int 10h


again:

    PRINT newline      ;restarts program

    jmp restart


end_of_execution:

    hlt


                ;print the error message

                ;and the initial message for input
```

```
lathos:
        PRINT newline
        PRINT errormsg
        jmp restart


        errormsg db "lathos"
        newline db 0Dh,0Ah, "$" ;messages
        startornot db "START (Y, N):",0Dh,0Ah, "$"
        input db "insert input: $"






        erase_char proc    ;this routine erases the last character pressed
        push ax
        mov ah,0eh
        mov al,8
        int 10h
        mov al,32
        int 10h
        mov al,8
        int 10h
        pop ax
        ret
```

```
    read_hex proc       ;read a valid hexadecimal digit
ksanadiavase:              ;if it's not valid read again
    mov ah, 1          ;until valid
    int 21h
    cmp al,'n'
    je end_of_execution
    cmp al,'N'
    je end_of_execution
    cmp al,48
    jc lathoseisodos
    cmp al,58
    jnc  mallon_lathoseisodos
    jmp swsth_eisodos
lathoseisodos:
    call erase_char     ;erases the invalid character
    jmp ksanadiavase
mallon_lathoseisodos:
    cmp al,65
    jc lathoseisodos
    cmp al,103
    jnc lathoseisodos
    cmp al,71
    jnc check_lathos2
    sub al,7
    jmp swsth_eisodos
check_lathos2:
    cmp al,97

    jc lathoseisodos

    sub al,39

swsth_eisodos:

    sub al,30h          ;in al we have a valid hex number pressed

    ret




   print_ax_dex proc

    push dx             ;here is the function that shows

    push cx             ;result on screen (ax is the result)

    mov dx,0            ;it works recursively

    cmp ax,0h           ;the result is decimal

    je proc_end

    mov cx,10           ;base 10

    div cx

    push dx

    call print_ax_dex

    pop dx

    mov ax,dx

    cmp ax,0ah

    jnc not_number

    add ax,48

    jmp number
```

```
not_number:
        add ax,55           ;the usual check for number or
                            ;characters and offset in ascii table
number:
        mov ah,0eh
        int 10h
proc_end:
        pop cx
        pop dx
        ret
        endp
```