



Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχ. Και Μηχ. Υπολογιστών

Εργαστήριο Μικροϋπολογιστών , 7ο εξάμηνο - Ροή Υ

Ακαδημαϊκή Περίοδος : 2011-2012

4^η ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ

Ομάδα: C16

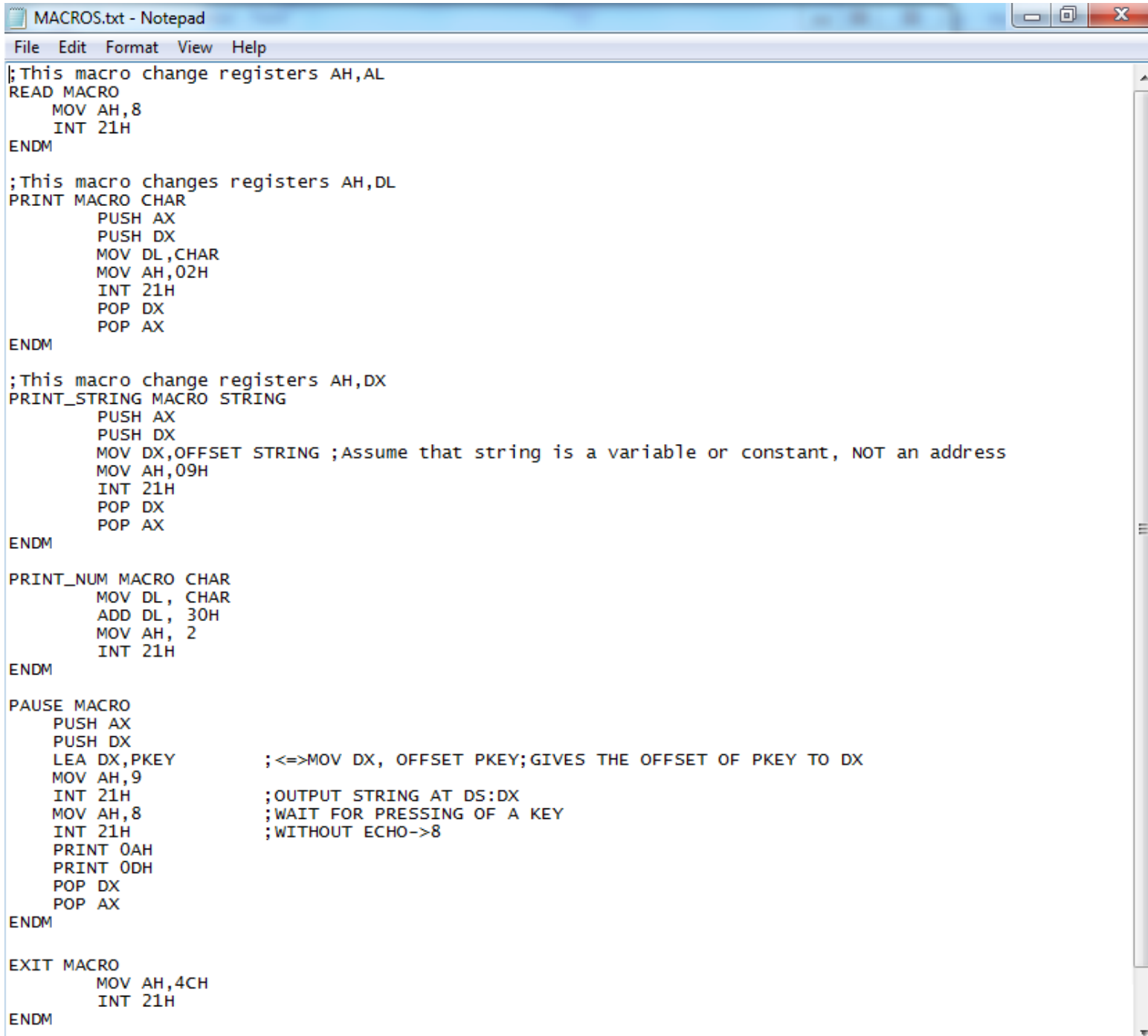
Γερακάρης Βασίλης Α.Μ.: 03108092

Διβόλης Αλέξανδρος Α.Μ.: 03107238

Τεντσεκ 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
```

Άσκηση i

Στην άσκηση αυτή εισάγουμε έναν 10bit δυαδικό αριθμό τον οποίο μετατρέπουμε σε δεκαδικό. Κατά την εισαγωγή αριθμού δεχόμαστε τιμές μόνο 0 και 1, τα οποία και τυπώνονται στην οθόνη ή ‘Q’ ή ‘q’ για έξοδο. Σε περίπτωση που πατηθεί οποιοδήποτε άλλο πλήκτρο το πρόγραμμα το αγνοεί και αναμένει το πάτημα ενός έγκυρου πλήκτρου. Οι ρουτίνες που υλοποιήθηκαν είναι η **BIN_10BIT_IN** η οποία εισάγει τον 10bit αριθμό και τον αποθηκεύει στον καταχωρητή **DX** ή βγαίνει σε περίπτωση που πατηθεί το ‘q’ ή ‘Q’, καθώς και η **PRINT_DEC**, η οποία δέχεται σαν είσοδο το περιεχόμενο του καταχωρητή **DX** και τον τυπώνει σε δεκαδική μορφή. Ο κώδικας φαίνεται παρακάτω:

```
INCLUDE MACROS.TXT
```

```
STACK_SEG SEGMENT STACK
    DW 128 DUP(?)
ENDS
```

```
DATA_SEG SEGMENT
    PKEY DB "press any key to restart or 'Q' to exit!..$"
    IN_MSG DB "GIVE A 10-BIT BINARY NUMBER: $"
    OUT_MSG DB "DECIMAL: $"
    LINE DB 0AH,0DH,"$"
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:
    PRINT_STRING IN_MSG
    CALL BIN_10BIT_IN ;DX<-000000 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
    PRINT_STRING LINE
    PRINT_STRING OUT_MSG
    CALL PRINT_DEC ;Must have in DX the number
    PRINT_STRING LINE
    PRINT_STRING PKEY
    READ
    CMP AL,'q'
    JE EXODOS
    CMP AL,'Q'
    JE EXODOS
    PRINT_STRING LINE
    JMP START
EXODOS:
    EXIT
MAIN ENDP
;-----ROUTINES-----
```

```

BIN_10BIT_IN PROC NEAR
    MOV DX,0
    MOV CX,10
IGNORE:
    READ          ;Read changes AX: puts input in AL and gives DOS function code with AH
        CMP AL,'q'
        JE EXODOS
        CMP AL,'Q'
        JE EXODOS
    CMP AL,30H
    JL IGNORE
    CMP AL,31H
    JG IGNORE     ;If we pass we have 0 or 1
    PRINT AL
    SUB AL,30H     ;AX<-00000000 00000000(0 or 1)
    MOV AH,0
    ROL DX,1
    ADD DX,AX
    LOOP IGNORE    ;loop for 10 times to import 10 bits
    RET
BIN_10BIT_IN ENDP

PRINT_DEC PROC NEAR
;   PUSH AX
;   PUSH BX
;   PUSH CX
;   PUSH DX
    MOV AX,DX      ;Put number in AX
    MOV BX,10      ;Put the divisor in BX
    MOV CX,0       ;Counts the number of decimal digits
AGAIN:
    MOV DX,0
    DIV BX         ;quotient in AX and remainder in DX
    PUSH DX
    INC CX
    CMP AX,0       ;Check if quotient = 0 (all digits stored in stack)
    JNE AGAIN
PRINT_LOOP:
    POP DX
    PRINT_NUM DL
    LOOP PRINT_LOOP
;   POP DX
;   POP CX
;   POP BX
;   POP AX
    RET
PRINT_DEC ENDP

CODE_SEG ENDS

END MAIN

```

Άσκηση ii

Μας ζητείται να εισάγουμε έναν τετραψήφιο δεκαδικό αριθμό ώστε να τον τυπώσουμε στη συνέχεια σε δεκαεξαδική μορφή. Εισάγουμε τον δεκαδικό αριθμό με την βοήθεια της ρουτίνας **DEC_4DIG_IN**, η οποία ελέγχει εάν το πλήκτρο που πατήθηκε δίνει αποδεκτή τιμή (0-9 δεκαδικό), αν ναι το τυπώνει στην οθόνη, πολλαπλασιάζει την τρέχουσα τιμή του καταχωρητή στον οποίο αποθηκεύεται ο αριθμός με το 10 (εφόσον το ψηφίο που μόλις εισάχθηκε είναι πολλαπλάσιο μίας δύναμης του 10 μικρότερης κατά 1 από τον ήδη υπάρχοντα αριθμό) και προσθέτει τον αριθμό. Μόλις εισαχθούν και τα 4 ψηφία αναμένεται η πίεση του πλήκτρου **ENTER** και μόνο. Με την βοήθεια της υπορουτίνας **PRINT_HEX** τυπώνεται ο αριθμός που βρίσκεται στον **DX** καταχωρητή σε 16αδική μορφή. Στη συνέχεια παρατίθεται ο κώδικας:

```
INCLUDE MACROS.TXT
```

```
STACK_SEG SEGMENT STACK
    DW 128 DUP(?)
ENDS
```

```
DATA_SEG SEGMENT
    PKEY DB "press any key to restart or 'Q' to exit!..$"
    IN_MSG DB "GIVE FOUR NUMBERS: $"
    OUT_MSG DB "HEX = $"
    ENTER_MSG DB "PLEASE PRESS ENTER!$"
    LINE DB 0AH,0DH,"$"
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:
    PRINT_STRING IN_MSG
    CALL DEC_4DIG_IN ;DX has the inputed number!
NOT_ENTER:
    READ
    CMP AL,0DH
    JNE NOT_ENTER
    PRINT_STRING LINE
    ; PRINT_STRING ENTER_MSG
    ; PRINT_STRING LINE
    PRINT_STRING OUT_MSG
    CALL PRINT_HEX ;Must have in DX the number
    PRINT_STRING LINE
    PRINT_STRING PKEY
    READ
    CMP AL,'q'
    JE EXODOS
```

```

    CMP AL,'Q'
    JE EXODOS
    PRINT_STRING LINE
    JMP START
EXODOS:
    EXIT
MAIN ENDP
;-----ROUTINES-----
DEC_4DIG_IN PROC NEAR
    MOV DX,0
    MOV BH,0
    MOV CX,4
IGNORE:
    READ ;Read changes AX: puts input in AL and gives DOS function code with AH
    CMP AL,'q'
    JE EXODOS
    CMP AL,'Q'
    JE EXODOS
    CMP AL,30H
    JL IGNORE
    CMP AL,39H
    JG IGNORE ;If we pass we have accepted value
    PRINT AL ;We have read without echo
    SUB AL,30H ;AX<-(0-9)
    MOV BL,AL ;BX<-00000000 (AL) the new decimal digit
    MOV AX,DX ;put the number in AX
    MOV DX,10 ;DX<-10 multiplier
    MUL DX ;AX*DX : Result is returned to DX-AX, but it can't be
>9999, so it fits in AX
    MOV DX,AX
    ADD DX,BX ;Add NEW digit
    LOOP IGNORE ;loop for 10 times to import 10 bits
    RET ;Finally we have the number in DX
DEC_4DIG_IN ENDP

PRINT_HEX PROC NEAR
; PUSH AX
; PUSH BX
; PUSH CX
; PUSH DX
    MOV AX,DX ;Put number in AX
    MOV BX,16 ;Put the divisor in BX
    MOV CX,0 ;Counts the number of decimal digits
AGAIN:
    MOV DX,0
    DIV BX ;quotient in AX and remainder in DX
    PUSH DX
    INC CX
    CMP AX,0 ;Check if quotient = 0 (all digits stored in stack)
    JNE AGAIN
PRINT_LOOP:
    POP DX
    CMP DL,9 ;i know that in char is something between 00000000 and 00001111
    JBE DEC_DEC ;if A<=9 jump to DEC_DEC
    ADD DL,07H;we add total 37H, if we have something A-F
DEC_DEC:

```

```

        ADD DL,30H
        MOV AH,02H
        INT 21H           ;To print the DL
    LOOP PRINT_LOOP
;   POP DX
;   POP CX
;   POP BX
;   POP AX
    RET
PRINT_HEX ENDP

CODE_SEG ENDS

END MAIN

```

Άσκηση iii

Στην άσκηση αυτή υλοποιούμε πρόγραμμα κατά το οποίο εισάγονται 20 (το πολύ) αλφαριθμητικοί χαρακτήρες (μόνο λατινικά γράμματα) και στη συνέχεια τυπώνονται με την σειρά εισαγωγής, όμως ανά είδος: **ΑΡΙΘΜΟΙ – ΠΕΖΟΙ ΧΑΡΑΚΤΗΡΕΣ – ΚΕΦΑΛΑΙΟΙ ΧΑΡΑΚΤΗΡΕΣ**. Πιέζοντας το **ENTER** ανά πάσα στιγμή δηλώνουμε τερματισμό της εισαγωγής χαρακτήρων και το πρόγραμμα τυπώνει την επιθυμητή έξοδο. Αν εισαχθούν και οι 20 χαρακτήρες αναμένεται η πίεση του πλήκτρου **ENTER** ώστε να συνεχίσει το πρόγραμμα. Αν πατηθεί κατά την είσοδο το πλήκτρο **/'** το πρόγραμμα τερματίζει.

Η είσοδος των δεδομένων πραγματοποιείται με την ρουτίνα **INPUT_ROUTINE**. Εκμεταλλευόμαστε το γεγονός ότι μπορούμε να έχουμε άμεσο έλεγχο κατά την πίεση ενός πλήκτρου, οπότε εφόσον εισαχθεί αποδεκτή τιμή ("0-9", "a-z", "A-Z"), αυτή αποθηκεύεται στον κατάλληλο πίνακα, **NUM_TABLE** αν είναι αριθμός, **LOWER_TABLE** αν είναι μικρός, ή **UPPER_TABLE** αν είναι κεφαλαίος χαρακτήρας, με την ταυτόχρονη αύξηση του αντίστοιχου μετρητή (**NUM_COUNTER**, **LOWER_COUNTER**, **UPPER_COUNTER**), ο οποίος κρατάει κάθε φορά το πλήθος των εγγεγραμμένων χαρακτήρων στον αντίστοιχο πίνακα.

Με την βοήθεια της ρουτίνας **OUTPUT_ROUTINE** τυπώνουμε με την σειρά τους τρεις πίνακες, ώστε να έχουμε το επιθυμητό αποτέλεσμα. Εντοπίζουμε το πρώτο στοιχείο του κάθε πίνακα με την βοήθεια της εντολής **MOV BX,OFFSET X_TABLE** και αφού τυπώσουμε το στοιχείο του πίνακα αυξάνουμε κατά 1 τον καταχωρητή BX με **INC BX** ώστε να εντοπίσουμε το επόμενο στοιχείο του πίνακα (λόγω της ανά byte διευθυνσιοδότησης της μνήμης και του ότι κάθε στοιχείο είναι ένα byte. Αν π.χ. έπρεπε να εντοπίσουμε λέξεις θα προσθέταμε 2 κάθε φορά). Σε περίπτωση που δεν έχει εισαχθεί κάποιου είδους χαρακτήρας δεν τυπώνεται κάποιος χαρακτήρας αυτού του είδους (ελέγχοντας τον μετρητή του αντίστοιχου πίνακα).

Η λειτουργία του προγράμματος γίνεται πιο σαφής από τον παρακάτω κώδικα:

```

INCLUDE MACROS.TXT

STACK_SEG SEGMENT STACK
    DW 128 DUP(?)
ENDS

DATA_SEG SEGMENT
    ENTAILMENT DB " => $"
    LINE DB 0AH,0DH,"$"

```



```

INPUT_MSG DB "GIVE 20 CHARACTERS (Latin characters,numbers,spaces,/ for
exit)",0AH,0DH,"$"
GIVE_MSG DB "GIVE AND PRESS ENTER:$"
    NUM_TABLE DB 20 DUP(?)
    LOWER_TABLE DB 20 DUP(?)
    UPPER_TABLE DB 20 DUP(?)
    NUM_COUNTER DW 0
    LOWER_COUNTER DW 0
    UPPER_COUNTER DW 0
    INDEX DW 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:
    PRINT_STRING INPUT_MSG
    PRINT_STRING GIVE_MSG
    CALL INPUT_ROUTINE
    CALL OUTPUT_ROUTINE
    PRINT_STRING LINE
    MOV LOWER_COUNTER,0    ;Resetting Counters for new input
    MOV UPPER_COUNTER,0
    MOV NUM_COUNTER,0
    JMP START
EXODOS:
    EXIT
MAIN ENDP
;-----ROUTINES-----
OUTPUT_ROUTINE PROC NEAR
    MOV CX,NUM_COUNTER
    CMP CX,0
    JE LOWER_START        ;Table is empty -> nothing to print
    MOV BX,OFFSET NUM_TABLE ;The start address of NUM_TABLE->Address of first
data
NUM_PRINT:
    MOV AL,DS:[BX]
    PRINT AL
    INC BX                ;BX+=1 to locate next data (1Byte-data)
    LOOP NUM_PRINT        ;Do this NUM_COUNTER times
    PRINT ' '
LOWER_START:              ;Do the same for the other two tables
    MOV CX,LOWER_COUNTER
    CMP CX,0
    JE UPPER_START
    MOV BX,OFFSET LOWER_TABLE
LOWER_PRINT:
    MOV AL,DS:[BX]
    PRINT AL
    INC BX
    LOOP LOWER_PRINT

```

```

        PRINT ' '
UPPER_START:
        MOV CX,UPPER_COUNTER
        CMP CX,0
        JE PRINT_EJECT
        MOV BX,OFFSET UPPER_TABLE
UPPER_PRINT:
        MOV AL,DS:[BX]
        PRINT AL
        INC BX
        LOOP UPPER_PRINT
PRINT_EJECT:
        RET
OUTPUT_ROUTINE ENDP

INPUT_ROUTINE PROC NEAR
;        PUSH DX
;        PUSH BX
;        PUSH CX
;    MOV DX,0        ;DH->Counter of Nums
;    MOV BX,0        ;BX->Counter of Uppercase,
;    MOV BP,0        ;BP->Counter of Lowercase
        MOV CX,20    ;Counter of maximum number of characters inputed
INPUT_LOOP:
    READ            ;Read changes AX: puts input in AL and gives DOS function code with AH
        CMP AL,0DH        ;ODH = ENTER
        JE INPUT_END
        CMP AL,20H        ;20H = SPACE
        JE SPACE_LOOP
        CMP AL,2FH        ;2FH = '/'
        JE EXODOS
        CMP AL,30H
        JL INPUT_LOOP
        CMP AL,39H
        JG UPPER_CHECK ;If we pass we have 0-9 value
        PRINT AL        ;We have read withouth echo
NUM_INPUT:
        MOV BX,OFFSET NUM_TABLE
        ADD BX,NUM_COUNTER
        MOV [BX],AL        ;Move Char in Num_Table
        INC NUM_COUNTER    ;Increase num-counter
        JMP ENDING_LOOP
UPPER_CHECK:
        CMP AL,41H
        JL INPUT_LOOP
        CMP AL,5AH
        JG LOWER_CHECK ;If we pass we have A-Z value
        PRINT AL        ;We have read withouth echo
UPPER_INPUT:
        MOV BX,OFFSET UPPER_TABLE
        ADD BX,UPPER_COUNTER
        MOV [BX],AL        ;Move Char in Num_Table
        INC UPPER_COUNTER  ;Increase upper-counter
        JMP ENDING_LOOP
LOWER_CHECK:
        CMP AL,61H

```

```

JL INPUT_LOOP
CMP AL,7AH
JG INPUT_LOOP      ;If we pass we have a-z value
PRINT AL           ;We have read without echo
LOWER_INPUT:
MOV BX,OFFSET LOWER_TABLE
ADD BX,LOWER_COUNTER
MOV [BX],AL        ;Move Char in Num_Table
INC LOWER_COUNTER  ;Increase upper-counter
ENDING_LOOP:
LOOP INPUT_LOOP
ENTER_LOOP:
READ
CMP AL,0DH         ;ODH = ENTER
JNE ENTER_LOOP
INPUT_END:
PRINT_STRING ENTAILMENT
RET
SPACE_LOOP:
PRINT ' '
JMP ENDING_LOOP
; POP CX
; POP BX
; POP DX
INPUT_ROUTINE ENDP
CODE_SEG ENDS
END MAIN

```

Άσκηση iv

Στην άσκηση αυτή μας ζητείται να πραγματοποιήσουμε πολλαπλασιασμό δύο 32bits αριθμών οι οποίοι εισάγονται από το πληκτρολόγιο, οπότε και προκύπτει αποτέλεσμα 64 bits. Αν χωρίσουμε τον κάθε 32bit αριθμό σε δύο 16bit μέρη (αφού μπορούμε να κάνουμε πολλ/σμο μεταξύ 16bit αριθμών). Στο παρακάτω σχήμα παρατηρούμε τους επιμέρους πολλαπλασιασμούς που πρέπει να γίνουν, καθώς και τις αντίστοιχες προσθέσεις.

Παρατηρούμε λοιπόν ότι πρέπει να προσθέσουμε τα εξής, ώστε να έχουμε το επιθυμητό

$$RESULT = (X0 \times Y0) + (X0 \times Y1 \times 2^{16}) + (X1 \times Y0 \times 2^{16}) + (X1 \times Y1 \times 2^{32})$$

αποτέλεσμα:

Οι πολλαπλασιασμοί με τις δυνάμεις του 2 (16 και 32) φαίνεται με αριστερή ολίσθηση στο σχήμα.

Οι μεταβλητές που χρησιμοποιήθηκαν είναι αυτές που φαίνονται μέσα στα πεδία των **A, B, C, D, RESULT** και έχουν μέγεθος λέξης (1W = 2B → 16bits). Η συλλογιστική της υλοποίησης φαίνεται παρακάτω και αφορά στην εύρεση των **RES_0** έως **RES_3**, με αρχικοποιημένο κρατούμενο **Carry = 0**. Όπου Carry και Temporary καταχωρητές που χρησιμοποιούνται για αυτήν την δουλειά (βλέπε κώδικα).

- **RES_0 = A_LOW**
- **RES_1 = A_HIGH + B_LOW + C_LOW**
 - Temporary = A_HIGH + B_LOW
 - If (Cflag==1) Carry= Carry + 1
 - **RES_1 = Temporary + C_LOW**
 - If (Cflag==1) Carry= Carry + 1
- **RES_2 = B_HIGH + C_HIGH + D_LOW + Carry**
 - Temporary = B_HIGH + Carry, Carry = 0
 - If (Cflag==1) Carry= Carry + 1
 - Temporary = Temporary + C_HIGH
 - If (Cflag==1) Carry= Carry + 1
 - **RES_2 = Temporary + D_LOW**
 - If (Cflag==1) Carry= Carry + 1
- **RES_3 = D_HIGH + Carry**

Οι ρουτίνες που χρησιμοποιήθηκαν φαίνονται παρακάτω:

READ_8HEX_DIG

Χρησιμοποιείται για ανάγνωση ενός 8ψήφιου (το πολύ) 16αδικού αριθμού. Με το **ENTER** τερματίζεται η είσοδος. Με το **\'/\'** τερματίζεται το πρόγραμμα. Επιστρέφει τον 32bito αριθμό στους καταχωρητές DX-BX.

ABCD_MAKE

Δημιουργεί τους **A, B, C, D** όπως αυτοί προαναφέρθηκαν, κάνοντας τους ανάλογους πολλαπλασιασμούς. Αποθηκεύονται στις αντίστοιχες μεταβλητές (***_LOW, *_HIGH**)

CALCULATION

Πραγματοποιεί τις απαιτούμενες προσθέσεις με στόχο την δημιουργία του 64bitου **RESULT**.

PRINT_HEX

Τυπώνει σε δεκαεξαδική μορφή τον αριθμό που βρίσκεται στον καταχωρητή DL (00 - 0F)

PRINT_16BIT

Τυπώνει σε δεκαεξαδική μορφή τον 16bit αριθμό που βρίσκεται στον καταχωρητή AX. Χρησιμοποιεί την PRINT_HEX

PRINT_RESULT

Τυπώνει σε δεκαεξαδική μορφή τον 64bit αριθμό που βρίσκεται στην μνήμη (RES_3 -

RES_0). Χρησιμοποιεί την PRINT_16BIT

Ο κώδικας φαίνεται παρακάτω.

```
INCLUDE MACROS.TXT
```

```
STACK_SEG SEGMENT STACK  
    DW 128 DUP(?)  
ENDS
```

```
DATA_SEG SEGMENT  
    LINE DB 0AH,0DH,"$"  
    FIRST_NUM DB "FIRST NUMBER:$"  
            SECOND_NUM DB "SECOND NUMBER:$"  
            RES_NUM DB "RESULT:$"  
    QUIT_QUEST DB "PRESS ANY KEY TO RESTART OR '/' TO EXIT...$"  
            X0 DW ?  
            X1 DW ?  
            Y0 DW ?  
            Y1 DW ?  
            A_HIGH DW ?  
            A_LOW DW ?  
            B_HIGH DW ?  
            B_LOW DW ?  
            C_HIGH DW ?  
            C_LOW DW ?  
            D_HIGH DW ?  
            D_LOW DW ?  
            RES_0 DW ?  
            RES_1 DW ?  
            RES_2 DW ?  
            RES_3 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:  
    ;[/reading]  
    PRINT_STRING FIRST_NUM  
        CALL READ_8HEX_DIG  
        MOV X0,BX  
        MOV X1,DX  
    PRINT_STRING SECOND_NUM  
        CALL READ_8HEX_DIG  
        MOV Y0,BX  
        MOV Y1,DX  
    ;[/reading ends]  
    CALL ABCD_MAKE  
    CALL CALCULATION  
    PRINT_STRING RES_NUM  
    CALL PRINT_RESULT  
    PRINT_STRING LINE  
    PRINT_STRING QUIT_QUEST  
    READ
```

```

        CMP AL,'/'
        JE EXODOS
        PRINT_STRING LINE
        JMP START
EXODOS:
    EXIT
MAIN ENDP
;-----ROUTINES-----
;[/routine = READ_8HEX_DIG]
READ_8HEX_DIG PROC NEAR
;Messes with AX,BX,CX,DX,SI returns 32bit input at DX-BX
    MOV CX,8
    MOV BX,0
    MOV DX,0
INPUT_NUM:
    PUSH DX
IGNORE_READ:
    READ
    CMP AL,'/'
    JE EXODOS
    CMP AL,0DH                ;ODH = ENTER (ASCII)
    JE FORCED_END            ;Forced end, before 8hex digits have been inputed
    CMP AL,30H
    JL IGNORE_READ
    CMP AL,39H
    JG HEX_CHECK
    PRINT AL
    SUB AL,30H
    JMP READ_OUT
HEX_CHECK:
    CMP AL,'A'
    JL IGNORE_READ
    CMP AL,'F'
    JG IGNORE_READ
    PRINT AL
    SUB AL,37H
READ_OUT:
    POP DX                    ;Here AL <- 0000 (0000 to 1111)
    ;[/code] Shifts left DX-BX, which handled like one 32bit register
    ROL BX,4
    SHL DX,4
    MOV SI,000FH
    AND SI,BX
    OR DX,SI
    AND BX,0FFF0H
    OR BL,AL
    ;[/code ends]
    LOOP INPUT_NUM
INPUT_ENDED:
    PRINT_STRING LINE
    RET
FORCED_END:
    POP DX
    JMP INPUT_ENDED
READ_8HEX_DIG ENDP
;[/routine ends]

```


;-=====

```
[/routine = ABCD_MAKE]
ABCD_MAKE PROC NEAR
;MAKING A(32bits) = A_HIGH | A_LOW : X0*Y0
    MOV AX,X0
    MUL Y0
    MOV A_HIGH,DX
    MOV A_LOW,AX
;MAKING B(32bits) = B_HIGH | B_LOW : X0*Y1
    MOV AX,X0
    MUL Y1
    MOV B_HIGH,DX
    MOV B_LOW,AX
;MAKING C(32bits) = C_HIGH | C_LOW : X1*Y0
    MOV AX,X1
    MUL Y0
    MOV C_HIGH,DX
    MOV C_LOW,AX
;MAKING D(32bits) = C_HIGH | C_LOW : X1*Y1
    MOV AX,X1
    MUL Y1
    MOV D_HIGH,DX
    MOV D_LOW,AX
    RET
ABCD_MAKE ENDP
[/routine ends]
```

;-=====

```
[/routine = CALCULATION]
CALCULATION PROC NEAR
;RESULT(64bits) = RES_3|RES_2|RES_1|RES_0 : X(32bits) * Y(32bits), WHERE X=X1|X0,Y=Y1|Y0
    MOV CX,0                ;CX = CARRY
;MAKING RES_0
    MOV AX,A_LOW
    MOV RES_0,AX
;MAKING RES_1
    MOV AX,A_HIGH
    ADD AX,B_LOW
    JNC NEXT00
    INC CX                ;If C=1 must increase CARRY
NEXT00:
    ADD AX,C_LOW          ;AX <- A_HIGH + B_LOW + C_LOW
    JNC NEXT01
    INC CX
NEXT01:
    MOV RES_1,AX
;MAKING RES_2
    MOV AX,B_HIGH
    ADD AX,CX
    JNC NEXT02
    MOV CX,1              ;Making CARRY=1
    JMP NEXT03
```

```

NEXT02:
    MOV CX,0
NEXT03:
    ADD AX,C_HIGH
    JNC NEXT04
    INC CX
NEXT04:
    ADD AX,D_LOW    ;AX <- CARRY + B_HIGH + C_HIGH + D_LOW
    JNC NEXT05
    INC CX
NEXT05:
    MOV RES_2,AX
;MAKING RES_3
    MOV AX,D_HIGH
    ADD AX,CX        ;Trust math, that tell us there will not be any carry
    MOV RES_3,AX
    RET
CALCULATION ENDP
;[/routine ends]

;-----

;[/routine = PRINT_HEX]
PRINT_HEX PROC NEAR
;PRINTS DL(hex) <- 0 0 0 0 (0000 - FFFF)
    CMP DL,9
    JG ADDR_42
    ADD DL,30H
    JMP ADDR_R6
ADDR_42:
    ADD DL,37H
ADDR_R6:
    PRINT DL
    RET
PRINT_HEX ENDP
;[/routine ends]

;-----

;[/routine = PRINT_16BIT]
PRINT_16BIT PROC NEAR
;Having at AX <- LMNO H, and want to print L,M,N and O
    MOV CX,4        ;repeat 4 times (4hex digits exist within a 16bit reg)
    CMP SI,0
    JNE PRINT_16_2
PRINT_16:           ;If flag is still 0 (nothing non-zero have printed yet) we are here
    ROL AX,4
    MOV DL,0FH
    AND DL,AL
    CMP DL,0
    JNE GO_ON
    LOOP PRINT_16
    JMP END_PRINT_16BIT
GO_ON:
    MOV SI,1        ;Flag that from now on must print zeros
    CALL PRINT_HEX

```

```

        JMP LOOP_LOOP          ;We don't use DEC CX, because if this has result CX = 0,
PRINT_16_2:          ;LOOP PRINT_16_2 will make CX = FFFF H and keep looping !
        ROL AX,4
        MOV DL,0FH
        AND DL,AL
        CALL PRINT_HEX
LOOP_LOOP:
        LOOP PRINT_16_2
END_PRINT_16BIT:
        RET
PRINT_16BIT ENDP
;[/routine ends]

;-----

;[/routine = PRINT_RESULT]
PRINT_RESULT PROC NEAR
        MOV SI,0              ;Flag for first non-zero number to print
        MOV AX,RES_3
        CALL PRINT_16BIT
        MOV AX,RES_2
        CALL PRINT_16BIT
        MOV AX,RES_1
        CALL PRINT_16BIT
        MOV AX,RES_0
        CALL PRINT_16BIT
        CMP SI,0              ;If the result is just zero, print it!
        JNE PRINT_RESULT_END
        PRINT 30H
PRINT_RESULT_END:
        RET
PRINT_RESULT ENDP
CODE_SEG ENDS
END MAIN

```

Τέλος αξίζει να αναφερθεί πως θα μπορούσαμε να παραλείψουμε τις ενδιάμεσες αποθηκεύσεις στην μνήμη, δηλαδή στις μεταβλητές **A,B,C,D (_LOW, _HIGH)**. Αυτό είναι δυνατόν κατι το περιγράφουμε στο επόμενο σχήμα. Βέβαια τοιουτοτρόπως η λειτουργία του προγράμματος γίνεται λιγότερο σαφής στο ενδιάμεσο στάδιο των υπολογισμών και ο κώδικας που θα προέκυπτε λιγότερο ευανάγνωστος.

