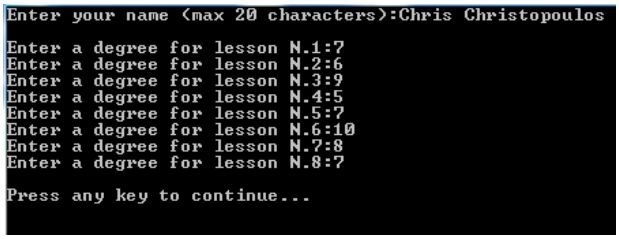
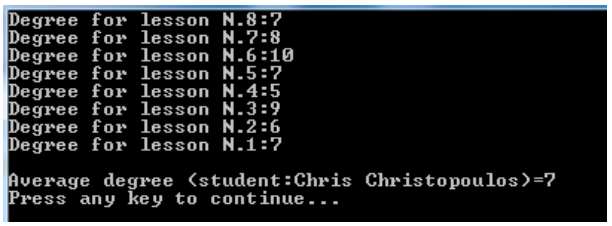
Τίτλος Ασκησης:	Υλοποίηση βασικών δομών επαναλήψεων και επεξεργασία πινάκων σε γλώσσα Assembly x86
Εργαστήριο:	4
Απαραίτητα Εργαλεία:	Visual Studio Professional 2008 (προτεινόμενο) /2010/2012 (και εκδόσεις express)
Απαιτούμενες Γνώσεις:	Προγραμματισμός C/C++, Διαγράμματα ροής
Στόχοι:	1) Ορισμός επικεφαλίδων (Labels) 2) Υλοποίηση δομής επανάληψης με χρήση της εντολής LOOP 3) Διασύνδεση – χρήση procedures εξωτερικών βιβλιοθηκών 4) Εισαγωγή και εμφάνιση αριθμητικών και αλφαριθμητικών δεδομένων με κλήση procedures της βοηθητικής βιβλιοθήκης
Διορία Παράδοσης:	Εντός της διδακτική ώρας του επόμενου εργαστηρίου
Παραδοτέα:	1) Επιδειξη λειτουργίας και Πηγαία Αρχεία (5/10) 2) Γραπτή αναφορά (στο χαρτί) των ενεργειών που έγιναν (5/10)
Διδάσκων:	Γρηγόρης Δημητρουλάκος

Στόχος εργαστηριακής άσκησης : Στην παρούσα άσκηση ο φοιτητής καλείται να γράψει ένα assembly πρόγραμμα εισαγωγής και επεξεργασίας αριθμητικών τιμών πίνακα, κάνοντας χρήση της βασικής εντολής επανάληψης LOOP. Παράλληλα ο φοιτητής εξοικειώνεται με τις άμεσες και έμμεσες αναφορές στη μνήμη, με τη δήλωση και χρήση των πινάκων, με την διαπέραση στοιχείων πινάκων, καθώς και στην κλήση procedures της βοηθητικής βιβλιοθήκης, το πέρασμα παραμέτρων σε αυτές μέσω των καταχωρητών, κατανοώντας παράλληλα τον τρόπο διασύνδεσης των βοηθητικών βιβλιοθηκών στο πρόγραμμα του. Επίσης εξοικειώνεται με τη χρήση όλων των συναρτήσεων της βιβλιοθήκης και είναι σε θέση να εισάγει και εξάγει (εμφανίζει) αριθμητικά και αλφαριθμητικά δεδομένα.

Άσκηση 1: Δημιουργία ενός προγράμματος σε γλώσσα assembly το οποίο με εμφάνιση κατάλληλων μηνυμάτων στην οθόνη να προτρέπει το χρήστη αρχικά να εισάγει το όνομά του βαθμολογούμενου (μέγιστου μεγέθους 20 χαρακτήρων) και στη συνέχεια να ζητά την εισαγωγή των βαθμών (ακεραίων τιμών 1 έως 10) για οκτώ (8) συνολικά μαθήματα. Στη συνέχεια το πρόγραμμα θα υπολογίζει τον ακέραιο (προς τα κάτω) μέσο όρο της βαθμολογίας και για τα οκτώ μαθήματα τον οποίο και θα εμφανίζει στην οθόνη (εφόσον πρώτα την καθαρίσει) με κατάλληλο μήνυμα το οποίο θα αναφέρει και το όνομα του βαθμολογούμενου. Επίσης θα εμφανίζονται και οι βαθμοί του κάθε μαθήματος με αντίστροφη σειρά σε σχέση με αυτή που εισήχθησαν. Δεν απαιτείται ο έλεγχος οριακών τιμών εισόδου από το πρόγραμμα, ο χρήστης θεωρούμε ότι εισάγει τιμές εντός των αποδεκτών ορίων.





α) Δημιουργία νέου project για προγράμματα assembly ή χρήση προτύπου από προηγούμενη άσκηση

b) Δημιουργία πηγαίου αρχείου assembly κώδικα με όνομα **Ergasia_4_a.asm** (συμπερίληψη του αρχείου επικεφαλίδων irvine32.inc προκειμένου να μπορούν να κληθούν οι i/o procedures της

βοηθητικής βιβλιοθήκης), όπου με χρήση των κατάλληλων εντολών καθώς και κλήση των απαιτούμενων procedures της βοηθητικής βιβλιοθήκης θα υλοποιηθούν οι απαιτούμενες λειτουργίες του προγράμματος. Αρχικά, θα δηλώνεται μια σταθερά με συμβολικό όνομα **table_size**=8d (αριθμός μαθημάτων) και μία σταθερά με συμβολικό όνομα **name_size**=21d (αριθμός χαρακτήρων ονόματος, συμπεριλαμβανομένου του χαρακτήρα τερματισμού) με χρήση της οδηγίας **EQU**

c) Στη συνέχεια, στο data segment του προγράμματος, θα δηλώνονται α) τυχόν μεταβλητές που χρειάζονται για του υπολογισμούς, β) ένας πίνακας (μεγέθους **table_size**) ακεραίων αριθμών για την αποθήκευση των βαθμολογιών, ένα πίνακας (μεγέθους **name_size**) χαρακτήρων για την αποθήκευση του ονόματος του βαθμολογούμενου καθώς και γ) τα κατάλληλα αλφαριθμητικά δεδομένα (πίνακες χαρακτήρων) για την εμφάνιση των σχετικών μηνυμάτων προς τον χρήστη

d) Κατόπιν με κλήση των procedures **WriteString**, **ReadString** και **Crlf** της βοηθητικής βιβλιοθήκης θα εμφανίζεται το μήνυμα στην οθόνη «**Enter your name (max 20 characters):**», όπου στη συνέχεια ο χρήστης θα εισάγει ένα όνομα (πατώντας στο τέλος enter). Πριν την κλήση των procedure θα πρέπει να θέσουμε κατάλληλες αρχικές τιμές στους καταχώρητες που χρησιμοποιεί η κάθε διαδικασία για πέρασμα παραμέτρων με βάση τις οδηγίες των υποδείξεων της άσκησης

e) Στη συνέχεια με κλήση των procedures **WriteString**, **WriteDec**, **ReadDec** και **Crlf** της βοηθητικής βιβλιοθήκης και με χρήση μίας επικεφαλίδας (**label**) με όνομα **L1** και της εντολής **LOOP** θα υλοποιούμε μία δομή επανάληψης όπου αρχικά θα θέτουμε την τιμή του καταχωρητή ECX = **table_size**, και εντός της επανάληψης θα εμφανίζεται το μήνυμα στην οθόνη «**Enter a degree for lesson N.1:**», όπου ο χρήστης θα εισάγει έναν ακέραιο αριθμό – βαθμό (πατώντας στο τέλος enter). Στη συνέχεια σε νέα γραμμή θα ακολουθεί το μήνυμα «**Enter a degree for lesson N.2:**», μέχρι να εισαχθούν και τα οκτώ μαθήματα. Οι βαθμοί που θα εισάγονται θα αποθηκεύονται στον πίνακα βαθμολογιών που έχουμε δηλώσει στο data segment. Η εμφάνιση του δείκτη μαθήματος στα μηνύματα (N.1, N.2, … N.8) θα πραγματοποιείται με την εμφάνιση τιμής σχετικού μετρητή και όχι με την δήλωση ξεχωριστών μηνυμάτων στο data segment

f) Στη συνέχεια με χρήση μίας επικεφαλίδας (**label**) με όνομα **L2** και της εντολής **LOOP** θα υλοποιούμε μία δομή επανάληψης όπου αρχικά θα θέτουμε την τιμή του καταχωρητή ECX = **table_size**, και εντός της επανάληψης θα υπολογίζουμε το άθροισμα των επιμέρους τιμών του πίνακα βαθμολογιών. Αμέσως μετά την ολοκλήρωση της δομής επανάληψης θα διαιρούμε το άθροισμα με τον αριθμό των μαθημάτων (8) με χρήση της εντολής **SHR**, ως ένας έμμεσος τρόπος διαίρεσης (επιστρέφει το ακέραιο μέρος της διαίρεσης με δυνάμεις του 2 – στρογγυλοποίηση προς τα κάτω)

g) Κατόπιν με κλήση των procedures **WriteString**, **WriteDec**, **Crlf** και **Clrscr** της βοηθητικής βιβλιοθήκης θα καθαρίζουμε τα περιεχόμενα της οθόνης και με χρήση μίας επικεφαλίδας (**label**) με όνομα **L3** και της εντολής **LOOP** θα υλοποιούμε μία δομή επανάληψης όπου αρχικά θα θέτουμε την τιμή του καταχωρητή ECX = **table_size**, και εντός της επανάληψης θα εμφανίζεται το μήνυμα στην οθόνη «**Degree for lesson N.8:**» και δίπλα ο αριθμός του αντίστοιχου μαθήματος. Στη συνέχεια σε νέα γραμμή θα ακολουθεί το μήνυμα «**Degree for lesson N.7:**», μέχρι να εμφανισθούν οι βαθμοί και για τα οκτώ μαθήματα. Πάλι η εμφάνιση του δείκτη μαθήματος στα μηνύματα (N.8, N.7, … N.1) θα πραγματοποιείται με την εμφάνιση τιμής σχετικού μετρητή και όχι με την δήλωση ξεχωριστών μηνυμάτων στο data segment

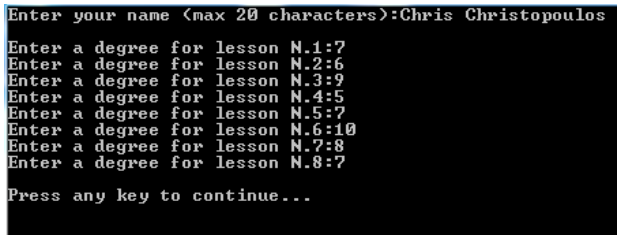
h) Τέλος με κλήση των procedures **WriteString**, **WriteDec**, και **Crlf**, ο μέσος όρος που υπολογίσθηκε σε προηγούμενο βήμα εμφανίζεται στην οθόνη με σχετικό μήνυμα «**Average degree (student: xxxxxxxxxxx)=**» ακολουθούμενο από τη σχετική τιμή.

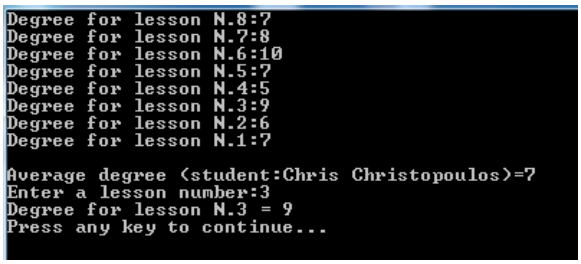
i) Μεταγλώττιση (assembling), διασύνδεση (linking) και εκτέλεση προγράμματος

j) Για λόγους ομοιομορφίας ακολουθείστε το παρακάτω πρότυπο κώδικα:

```
TITLE  Expression calculator       (Ergasia_4_a.asm)
; This program ……
INCLUDE Irvine32.inc
        {δήλωση σταθερών συμβολικών ονομάτων – compile time data}
.data
        {δήλωση συμβολικών ονομάτων μεταβλητών – run time data}
.code
main PROC
        {εντολές για την υλοποίηση της άσκησης}
        exit
main ENDP
END main
```

Άσκηση 2: Μετατροπή του ανωτέρω προγράμματος (με όνομα **Ergasia_4_b.asm**) ώστε να υλοποιεί τον υπολογισμό του μέσου όρου και την αντίστροφη εμφάνιση των βαθμών εντός της ίδιας δομής επανάληψης. Επιπλέον στο τέλος θα ζητά από τον χρήστη να εισάγει ένα αριθμό μαθήματος (από το 1 έως το 8) και θα του επιστρέφει τον αντίστοιχο βαθμό. Και πάλι θεωρούμε ότι ο χρήστη εισάγει τιμές εντός των αποδεκτών ορίων





α) Συγχώνευση των δυο δομών επανάληψης (εμφάνισης και υπολογισμού) σε μία (προσοχή στη χρήση των καταχωρητών ώστε να μην υπάρχει απώλεια δεδομένων)

β) Στο τέλος του προγράμματος με κλήση των procedures **WriteString**, **ReadDec**, **WriteDec** και **Crlf** της βοηθητικής βιβλιοθήκης θα εμφανίζεται το μήνυμα στην οθόνη «**Enter a lesson number:**», όπου στη συνέχεια ο χρήστης θα εισάγει ένα αριθμό μαθήματος (1-8, πατώντας στο τέλος enter). Κατόπιν το πρόγραμμα θα υπολογίζει τη θέση (offset) του σχετικού στοιχείου στον πίνακα βαθμολογιών και θα εμφανίζεται το μήνυμα «**Degree for lesson N.x = y**», όπου x ο αριθμός του μαθήματος και y ο αντίστοιχος βαθμός του.

β) Μεταγλώττιση (assembling), διασύνδεση (linking) και εκτέλεση του προγράμματος

Υποδείξεις :

1) **Symbol names** : τα συμβολικά ονόματα στην assembly ακολουθούν συγκεκριμένους κανόνες σύνταξης (δεν μπορούν είναι δεσμευμένες λέξεις, να ξεκινούν από ψηφίο, κλπ) και χρησιμοποιούνται για δηλώσεις συμβολικών ονομάτων σταθερών, μεταβλητών, δομών,

procedures, macros, labels, κλπ. Η ισχύ τους αφορά μόνο τον assembler και το χρόνο μεταγλώττισης του προγράμματος. Στο τελικό εκτελέσιμο κώδικα τα symbol names αντικαθίστανται από συγκεκριμένες σχετικές διευθύνσεις (offsets) ή άμεσες τιμές (immediate values)

2) **EQU directive** : οδηγία προς τον assembler για την δήλωση συμβολικών ονομάτων σταθερών που χρησιμοποιούνται και υπολογίζονται από τον προεπεξεργαστή του assembler μόνο κατά την μεταγλώττιση του προγράμματος (compile time data). Παραδείγματα δηλώσεων συμβολικών ονομάτων σταθερών :

```
sym1 EQU 3d+5d              ; expression assignment, sym1 = 8d
sym2 EQU <3d+5d >           ; text assignment, sym2 = "3d+5d"
```

3) **SHR** : εντολή ολίσθησης των bits ενός operand προς τα δεξιά για συγκεκριμένο αριθμό bits, όπου το τελευταίο ολισθαίνων bit αποθηκεύεται στο carry flag (CF) και εισάγονται '0' bits στο αριστερό μέρος

```
π.χ.   mov ax, 1111111100000000b      ; AX=1111111100000000b
       shr ax,1                       ; AX=0111111110000000b
       shr ax,2                       ; AX=0001111111100000b
```

Η ολίσθηση προς τα δεξιά πραγματοποιεί ακεραία διαίρεση (στρογγυλοποίηση προς τα κάτω) με δυνάμεις του 2, ανάλογα με τα bits που ολισθαίνουν. Εφόσον προσθέτει 0 bits από τα αριστερά, η διαίρεση αφορά μόνο μη προσημασμένους ακεραίους

4) **Memory access** : η προσπέλαση των δεδομένων αποθηκευμένα στη μνήμη του συστήματος (data segment) μπορεί να γίνει με δύο τρόπους
   a. άμεσα (**direct**) όταν γνωρίζουμε το offset (εντός του data segment) ή ισοδύναμα όταν γνωρίζουμε το όνομα της μεταβλητής κατά τη μεταγλώττιση (compile time)

```
π.χ.          mov eax, variable_name      mov eax, var1
ή             mov eax, [variable_name]    mov eax, [var1]
ή γενικά      mov eax, [offset_exp]       mov eax, [array3+2]
```

   όπου offset_exp = expression = (imm_constant | symbol_name)* | *variable_name*
   Η έκφραση offset_exp μπορεί να αποτελείται από πολλές άμεσες (immediate) τιμές και συμβολικά ονόματα σταθερών και από ένα το πολύ όνομα μεταβλητής. Υπολογίζεται κατά την μεταγλώττιση (assembling time) του προγράμματος συνοδεύοντας τον τελικό κώδικα (opcode) της εντολής με έναν σταθερό immediate operand

   b. έμμεσα (**indirect**) όταν το offset (εντός του data segment) της θέσης μνήμης που περιέχει τα επιθυμητά δεδομένα δεν είναι γνωστό κατά την μεταγλώττιση και διαμορφώνεται (υπολογίζεται) κατά το χρόνο εκτέλεσης (run time) του προγράμματος με χρήση των καταχωρητών της CPU. Για τις έμμεσες αναφορές σε θέσεις μνήμης, τα σχετικά offset (μετατόπιση εντός του data segment) έχουν μήκος 32-bit και συνεπώς μπορούν να αποθηκευτούν σε οποιοδήποτε 32-bit καταχωρητή γενικού σκοπού της CPU (EAX, EBX, ECX, EDX), ωστόσο συστήνεται αυστηρά η χρήση των καταχωρητών **ESI** (extended source index), **EDI** (extended destination index) οι οποίο και προορίζονται ειδικά για αυτό το σκοπό

```
π.χ.    array3 WORD 10, 20, 30, 40      ; δήλωση στο data segment
        mov esi, OFFSET array3          ; ESI = offset του 1ου byte του array3
        add esi, TYPE array3            ; ESI = ESI + 2 (type of word)
        mov ax, [esi]                   ; AX = 20
```

Γενικά η χρήση των αγκυλών [offset] σε έναν operand μιας εντολής, υποδηλώνει έμμεση (indirect) αναφορά στη μνήμη, δηλαδή αναφέρεται στο περιεχόμενο των δεδομένων που είναι αποθηκευμένα στο offset του operand (και δεν αναφέρεται στην ίδια την τιμή του offset, που συνήθως είναι αδιάφορη για τον χρήστη)

Παρακάτω αναφέρονται όλοι οι υποστηριζόμενοι (από τη CPU) συνδυασμοί έμμεσης αναφοράς στη μνήμη:

```
[reg]                       ; indirect
[reg*sf]                    ; indirect, scaled
[reg + offset_exp]          ; indirect, displacement
[reg*sf + offset_exp]       ; indirect, displacement, scaled
[reg + reg]                 ; indirect, base - index
[reg + reg*sf]              ; indirect, base – index, scaled
[reg + reg + offset_exp]    ; indirect, base – index, displacement
[reg + reg*sf + offset_exp] ; indirect, base-index-displacement, scaled
```

όπου sf = scale_factor = (1 | 2 | 4 | 8)
και offset_exp = expression = (imm_constant | symbol_name)* | *variable_name*

```
π.χ.    mov ax, array3 [ebp + esi*4 + 12d]
ή       mov ax, [array3 + ebp + esi*4 + 12d]
```

Η παραπάνω πλήρη ανάλυση παρουσιάζεται για εκπαιδευτικούς σκοπούς άλλα και για χρησιμοποίηση σε επόμενες ασκήσεις.

5) **Arrays** : ένας πίνακας στην assembly είναι ο χώρος στη μνήμη του συστήματος όπου αποθηκεύονται σε διαδοχικές θέσεις μνήμης ένας πεπερασμένος αριθμός δεδομένων του ιδίου τύπου. Σε επίπεδο μηχανής υπάρχουν μόνο πίνακες μίας διάστασης, με τους πίνακες πολλαπλών διαστάσεων να αναπαρίστανται ως απλοί μονοδιάστατοι πίνακες π.χ. ένας πίνακας δύο διαστάσεων 6x6 αντιστοιχεί σε ένα πίνακα μίας διάστασης 36 στοιχείων.
Η δήλωση – ορισμός ενός πίνακα στην assembly τοποθετείται στο data segment του προγράμματος και επιτυγχάνεται είτε με τη χρήση της οδηγίας **DUP** προς τον assembler είτε με την παράθεση διαδοχικών αρχικών τιμών. Το όνομα της μεταβλητής τύπου πίνακα, αντιστοιχεί στην διεύθυνση (μετατόπιση εντός του data segment) του πρώτου byte του πρώτου στοιχείου του πίνακα. Παραδείγματα δηλώσεων πινάκων:

πίνακας με όνομα array1, 30 στοιχείων προσημασμένων 16-bit ακέραιων, χωρίς αρχική τιμή

```
array1 SWORD 30 DUP (?)
```

πίνακας με όνομα array2, 5x5 στοιχείων μη προσημασμένων 32-bit ακέραιων, αρχική τιμή (0)

```
array2 DWORD 25 DUP (0)
```

πίνακας με όνομα array3, 5 στοιχείων μη προσημασμένων 16-bit ακέραιων, με διακριτές αρχικές τιμές

```
array3 WORD 10, 20, 30, 40, 50
```

πίνακας με όνομα array4, 15 στοιχείων μη προσημασμένων 16-bit ακέραιων, με διάφορες αρχικές τιμές

```
array4 WORD 10, 20, 30, 40, 50, 5 DUP (0), 60, 70, 80, 90, 100
```

6) **Αναφορές σε στοιχεία πινάκων** : η αναφορά σε στοιχεία ενός πινάκα μπορεί να γίνει με διάφορους τρόπους χρησιμοποιώντας κυρίως έμμεσες (indirect) αναφορές στη μνήμη. Για τους μονοδιάστατους πίνακες και για την αναφορά στο στοιχείο a[n], αρκεί για παράδειγμα η έμμεση αναφορά [a+edi*TYPE array1] όπου EDI=n. Για τους πίνακες πολλαπλών διαστάσεων απαιτείται ο σταδιακός ανά επίπεδο υπολογισμός του offset του κάθε υποπίνακα μέχρι τον τελικό υπολογισμό του offset του επιθυμητού στοιχείου. Για παράδειγμα σε ένα πίνακα δύο διαστάσεων, η αναφορά στο στοιχείο a[n,m] όπου n=γραμμή και m=στήλη, απαιτείται πρώτα ο υπολογισμός του offset για το πρώτο στοιχείο της n οστης γραμμή και κατόπιν για το στοιχείο στην m στήλη.

Ενδεικτικά παραδείγματα αναφοράς σε στοιχείο array1[2] μονοδιάστατου πίνακα:

```
π.χ.    array1 DWORD 10, 20, 30, 40, 50          ; δήλωση data segment, dword = 4bytes
        mov eax, array1                          ; EAX = array1[0] = 10

        mov eax, [array1 + 8]                    ; EAX = array1[2] = 30

        mov esi, 2                               ; ESI = 2 = δείκτης στοιχείου
        mov eax, [array1 + esi * TYPE array1]    ; EAX = array1[2] = 30
        mov edi, OFFSET array1                   ; EDI = offset(array1)
        mov eax, [edi + esi* TYPE array1]        ; EAX = array1[2] = 30

        mov esi, 2                       ; ESI = 2 = δείκτης στοιχείου
        shl esi,2                        ; ESI = ESI *2^2 = 8 (πολλαπλασιασμός με ολίσθηση)
        add esi, OFFSET array1           ; ESI = offset(array1[2])
        mov eax, [esi]                   ; EAX = array1[2] = 30
```

7) **TYPE** : οδηγία που επιστρέφει το μέγεθος σε bytes του τύπου δεδομένων μιας μεταβλητής

```
π.χ.    var1 SDWORD      ; TYPE var1 = 4
        var2 SBYTE       ; TYPE var2 = 1
        var3 WORD        ; TYPE var3 = 2
```

8) **OFFSET** : οδηγία που επιστρέφει την 32-bit διεύθυνση (μετατόπιση σε bytes εντός του data segment) μιας μεταβλητής

```
π.χ.    mov eax, OFFSET string1      ; EAX = offset του 1ου byte του string1 ('H')
```

9) **WriteString** : συνάρτηση της βοηθητικής βιβλιοθήκης η οποία εμφανίζει τους χαρακτήρες ενός πίνακα (string) στο console window του προγράμματος. Δέχεται ως παράμετρο το 32-bit offset του string (δηλαδή τη μετατόπιση του πρώτου χαρακτήρα) στον EDX register. Το string θα πρέπει πάντα να ακολουθείται από το null byte

```
π.χ.    string1 BYTE "Hello", 0      ; δήλωση στο data segment
        mov edx, OFFSET string1
        call WriteString             ; εμφανίζει το string1
```

10) **ReadString** : συνάρτηση της βοηθητικής βιβλιοθήκης η οποία διαβάζει από την είσοδο (πληκτρολόγιο, μέχρι ο χρήστης να πατήσει το enter) μία ακολουθία χαρακτήρων (string). Δέχεται ως παραμέτρους α) το 32-bit offset του string όπου και θα αποθηκευθεί η είσοδος, (δηλαδή τη μετατόπιση του πρώτου χαρακτήρα) στον EDX register και β) το μέγιστο αριθμό των χαρακτήρων (αυξημένο κατά ένα) προς είσοδο στον ECX register. Στο τελικό αποθηκευμένο string προστίθεται αυτόματα το null byte

```
π.χ.    string1 BYTE 21 DUP(?)          ; δήλωση στο data segment
        mov edx, OFFSET string1
        mov ecx, 21d
        call ReadString          ; εισάγει το πολύ 20 χαρακτήρες (+null byte) από την
                                                                είσοδο στο string1
```

11) **WriteDec** procedure: διαδικασία της βοηθητικής βιβλιοθήκης irvine32, η οποία εμφανίζει την τρέχουσα τιμή του EAX register στην οθόνη (console window) σε δεκαδική **μη** προσημασμένη μορφή (πχ. αν EAX=FFFFFFFDh τότε εμφανίζει την δεκαδική τιμή 4.294.967.293)

12) **ReadDec** procedure: διαδικασία της βοηθητικής βιβλιοθήκης irvine32, η οποία διαβάζει από την είσοδο (πληκτρολόγιο, μέχρι ο χρήστης να πατήσει το enter) έναν ακέραιο αριθμό σε δεκαδική **μη** προσημασμένη μορφή και αποθηκεύει την τιμή του στον EAX register

13) **Crlf** procedure : διαδικασία της βοηθητικής βιβλιοθήκης irvine32, η οποία μεταφέρει τον κέρσορα της οθόνης (console window) στην αρχή της επόμενης γραμμής

14) **Crlscr** procedure : διαδικασία της βοηθητικής βιβλιοθήκης irvine32, η οποία καθαρίζει το περιεχόμενο της οθόνης (console window) και μεταφέρει τον κέρσορα στην αρχή της πρώτης γραμμής

15) **WaitMsg** procedure : διαδικασία της βοηθητικής βιβλιοθήκης irvine32, η οποία διακόπτει προσωρινά την εκτέλεση του προγράμματος εμφανίζοντας σχετικό μήνυμα στο χρήστη προκειμένου να πατήσει ένα οποιοδήποτε πλήκτρο, κατόπιν η εκτέλεση του προγράμματος συνεχίζεται κανονικά

16) Code **Label**: είναι ένα συμβολικό όνομα το οποίο όταν τίθεται μπροστά από μια εντολή της assembly αντιστοιχεί στο offset της εντολής εντός του code segment του προγράμματος. Χρησιμοποιείται κυρίως στις εντολές μεταφοράς ελέγχου του προγράμματος (π.χ. JMP, LOOP) για την αλλαγή της ροής εκτέλεσής του. Υφίσταται μόνο κατά το χρόνο μεταγλώττισης και στον παραγόμενο κώδικα αντικαθίσταται από τον assembler με την διεύθυνση (offset) της σχετικής εντολής

```
[label:] mnemonic [operands] [;comment]
```

17) **LOOP :** εντολή της assembly η οποία επαναλαμβάνει ένα μπλοκ από δηλώσεις (εντολές) για ένα συγκεκριμένο αριθμό επαναλήψεων. Ο register **ECX** χρησιμοποιείται από την εντολή αυτόματα ως μετρητής και μειώνεται κάθε φορά που η εντολή εκτελείται. Έχει τον περιορισμό ότι ο loop destination πρέπει να είναι σε απόσταση μεταξύ -128 και +127 bytes από την τρέχουσα θέση του μετρητή προγράμματος. Κατά την εκτέλεση της εντολή πραγματοποιείται πρώτα μείωση του ECX register κατά 1 και στη συνέχεια έλεγχο σύγκριση της τιμής του ECX με το 0. Αν η τιμή του ECX > 0 πραγματοποιείται άλμα (jump) στην destination label της εντολής. Αν η τιμή του

ECX = 0 ο έλεγχος περνά κανονικά στην επόμενη προς εκτέλεση εντολή

```
     mov  ax,0
     mov  ecx,5
L1:
     inc  ax
     loop L1
```

Εκτέλεση της εντολής inc ax, πέντε (5) φορές

Η αναλυτική παρουσίαση της εντολής περιέχεται στις διαφάνειες 185-192 του υποστηρικτικού υλικού

18) **Βασικές εντολές assembly :** η γενική σύνταξη των απαιτούμενων βασικών assembly εντολών για την άσκηση έχουν ως εξής :

```
MOV dest,source          ADD dest,source          SUB dest,source          CALL procedure_name
  MOV reg,reg              reg,reg                  reg,reg
  MOV mem,reg              mem,reg                  mem,reg
  MOV reg,mem              reg,mem                  reg,mem
  MOV mem,imm              mem,imm                  mem,imm
  MOV reg,imm              reg,imm                  reg,imm

SHR destination,count                LOOP destination                INC reg/mem
      reg,imm8                       destination = current offset    DEC reg/mem
      mem,imm8                       – codeLabel offset
      reg,CL
      mem,CL
```

19) Τα εκτελέσιμα αρχεία **Project_Array_Process_a.exe** και **Project_Array_Process_b.exe** (περιλαμβάνονται στο υποστηρικτικό υλικό της άσκησης) παρουσιάζουν το τελικό αποτέλεσμα εξόδου που αναμένεται από τις ασκήσεις

Περιεχόμενα Γραπτής Αναφοράς

1) Κώδικας αρχείου **Ergasia_4_a.asm** της άσκησης 1
2) Screen shot του visual studio σε κατάσταση debugging, όπου να παρουσιάζονται το πρόγραμμα και τα περιεχόμενα των register καθώς και της data segment memory όπου είναι αποθηκευμένες οι μεταβλητές του προγράμματος (ακριβώς πριν την εκτέλεση της εντολής τερματισμού του προγράμματος)
3) Κώδικας αρχείου **Ergasia_4_b.asm** της άσκησης 2
4) Αντίστοιχο Screen shot του visual studio με το (2)
5) Τα αρχεία τύπου *.inc στην assembly τι περιέχουν και που χρησιμοποιούνται ?, ποίος ο αντίστοιχος τύπος αρχείων στη γλώσσα προγραμματισμού C/C++
6) Τι συμβαίνει αν δεν συμπεριλάβουμε το αρχείο Irvine32.inc στο πρόγραμμά μας ?, είναι δυνατή η εκτέλεση του προγράμματος μας χωρίς την συμπερίληψή του ?
7) Η αυτόματη μείωση της τιμής του καταχωρητή ECX από την εντολή LOOP που λαμβάνει χώρα ? (στον assembler, με ξεχωριστές εντολές, από τη CPU, άλλο)
8) Γιατί υπάρχει ο περιορισμός της απόστασης μεταξύ της εντολής LOOP και του σημείου άλματος (code label) ? γιατί δεν μπορεί να κατευθύνει την εκτέλεση του προγράμματος σε μεγαλύτερη απόσταση ? που εξυπηρετεί ο περιορισμός ?