Τίτλος Ασκησης:	Υλοποίηση απλών διαδικασιών, δομών επανάληψης και χρήση εντολών αλλαγής ροής προγράμματος υπό συνθήκη σε γλώσσα Assembly x86
Εργαστήριο:	5
Απαραίτητα Εργαλεία:	Visual Studio Professional 2008 (προτεινόμενο) /2010/2012 (και εκδόσεις express)
Απαιτούμενες Γνώσεις:	Προγραμματισμός C/C++, Διαγράμματα ροής
Στόχοι:	1) Προώθηση – ανάκτηση δεδομένων από την stack 2) Ορισμός διαδικασιών (procedure) 3) Activation record – Stack frame διαδικασίας 4) Πέρασμα παραμέτρων σε διαδικασίες μέσω register 5) Ανάλυση καταχωρητών κατάστασης CPU (Flag register) 6) Εντολές αλμάτων υπό συνθήκη 7) Activation Record (AR) συνάρτησης 8) Πέρασμα παραμέτρων by value, by reference σε συναρτήσεις 9) Πρότυπο κλήσης συναρτήσεων stdcall 10) Δήλωση τοπικών μεταβλητών (LOCAL directive) συναρτήσεων
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. Επιπλέον ο φοιτητής κατανοεί την αναπαράσταση του Activation Record μιας συνάρτησης στην stack του προγράμματος, εξασκείται με τον τρόπο περάσματος παραμέτρων by value και by reference, τον τρόπο αναφοράς στο περιεχόμενο των παραμέτρων καθώς και στον τρόπο δήλωσης και αναφοράς τοπικών μεταβλητών σε μία συνάρτηση. 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 μιας διαδικασίας αναζήτησης σε γλώσσα assembly, η οποία θα δέχεται ως παραμέτρους (μέσω register) α) την διεύθυνση ενός πίνακα στοιχείων ακέραιων προσημασμένων αριθμών 32bit, β) τον αριθμό των στοιχείων του πίνακα, γ) μία ακέραια προσημασμένη τιμή 32bit προς αναζήτηση και θα επιστρέφει (μέσω register), α) την διεύθυνση του (πρώτου) στοιχείου του πίνακα, στην περίπτωση που έχει την ίδια τιμή με την τιμή προς αναζήτηση, β) μηδενική τιμή σε περίπτωση που η τιμή προς αναζήτηση δεν βρεθεί. Για την δοκιμή της διαδικασίας δημιουργήστε κυρίως πρόγραμμα με έναν πίνακα 10 στοιχείων με αρχικές τιμές -5d, 5d, -6d, -7d, 9d, -6d, 10d, -3d, 5d, -6d. Ο χρήστης θα εισάγει στο κυρίως πρόγραμμα μία τιμή αναζήτησης από το πληκτρολόγιο μεταξύ των τιμών [-10, +10] και το πρόγραμμα θα καλεί την διαδικασία αναζήτησης, κατόπιν θα εμφανίζει σχετικό μήνυμα ενημερώνοντας τον χρήστη για την επιτυχία (με εμφάνιση της διεύθυνσης του στοιχείου) ή όχι της αναζήτησης. Επίσης θα πραγματοποιείται ο έλεγχος οριακών τιμών εισόδου στην τιμή αναζήτησης που εισάγει ο χρήστης, όπου στην περίπτωση μη αποδεκτής τιμή ο χρήστης θα καλείται εκ νέου για την εισαγωγή τιμής.

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

b) Δημιουργία πηγαίου αρχείου assembly κώδικα με όνομα **Ergasia_5_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

c) Αρχικά, θα δηλώνονται σταθερές με συμβολικό όνομα **max_limit**=10d και **min_limit**=-10d (αποδεκτά όρια τιμών εισόδου) και μία σταθερά με συμβολικό όνομα **array_size**=10d (αριθμός στοιχείων πίνακα) με χρήση της οδηγίας **EQU**

d) Στη συνέχεια, στο data segment του προγράμματος, θα δηλώνονται α) ένας πίνακας ακεραίων προσημασμένων αριθμών 32bit με αρχικές τιμές -5d, 5d, -6d, -7d, 9d, -6d, 10d, -3d, 5d, -6d, β) τα κατάλληλα αλφαριθμητικά δεδομένα (πίνακες χαρακτήρων) για την εμφάνιση των σχετικών μηνυμάτων προς τον χρήστη

e) Κατόπιν με κλήση των procedures **WriteString**, και **ReadInt** της βοηθητικής βιβλιοθήκης θα εμφανίζεται το μήνυμα στην οθόνη «**Enter a value (-10 to +10):**», όπου στη συνέχεια ο χρήστης θα εισάγει μία τιμή (πατώντας στο τέλος enter). Η τιμή που εισήγαγε ο χρήστης θα συγκρίνεται με το μέγιστο επιτρεπτό όριο (**max_limit**) με χρήση της εντολής **CMP** και ακολούθως με χρήση κατάλληλης εντολής άλματος υπό συνθήκη (JE, JNE, JA, JNA, JAE, JNAE, JB, JNB, JBE, JNBE, JG, JNG, JGE, JNGE, JL, JNL, JLE, JNLE), σε περίπτωση που (η τιμή) είναι εκτός ορίων η διαδικασία εισαγωγής θα επαναλαμβάνεται. Αντίστοιχα, η τιμή που εισήγαγε ο χρήστης θα συγκρίνεται με το ελάχιστο επιτρεπτό όριο (**min_limit**) με χρήση της εντολής **CMP** και ακολούθως με χρήση κατάλληλης εντολής άλματος υπό συνθήκη, σε περίπτωση που (η τιμή) είναι εκτός ορίων η διαδικασία εισαγωγής θα επαναλαμβάνεται.

f) Στη συνέχεια προκειμένου να καλέσουμε την διαδικασία αναζήτησης, θα μεταφέρουμε τις τιμές των παραμέτρων της σε συγκεκριμένους καταχωρητές (προτείνονται ο **EAX** για την τιμή προς αναζήτηση, ο **ECX** για τον αριθμό των στοιχείων του πίνακα, ο **EDX** δια την διεύθυνση του – πρώτου στοιχείου - πίνακα). Κατόπιν θα καλείται η procedures **SearchArray** με χρήση της εντολής **CALL**

g) Κατόπιν θα συγκρίνεται η τιμή που επέστρεψε η διαδικασία αναζήτησης (διεύθυνση στοιχείου πίνακα με την επιθυμητή τιμή ή 0 αν δεν υπάρχει τέτοιο στοιχείο) με χρήση της εντολή **CMP** και ακολούθως, με χρήση κατάλληλης εντολής άλματος υπό συνθήκη, το πρόγραμμα θα εμφανίζει το μήνυμα «**The value hasn't been found**» σε περίπτωση που δεν βρέθηκε στοιχείο, ή το μήνυμα «**The value has been found at address (offset):**» ακολουθούμενο από την διεύθυνση του στοιχείου του πίνακα σε hexadecimal μορφή με χρήση της procedure **WriteHex** της βοηθητικής βιβλιοθήκης

h) Τέλος δημιουργούμε τη διαδικασία αναζήτησης με όνομα **SearchArray**, με χρήση των **PROC** και **ENDP** οδηγιών και της εντολής **RET**, η οποία δεχόμενη ως παραμέτρους τιμές από συγκεκριμένους registers (προτείνονται ο **EAX** για την τιμή προς αναζήτηση, ο **ECX** για τον αριθμό των στοιχείων του πίνακα, ο **EDX** δια την διεύθυνση του – πρώτου στοιχείου - πίνακα), θα συγκρίνει ένα προς ένα τα στοιχεία του πίνακα με την τιμή προς αναζήτηση και θα επιστρέφει σε ένα καταχωρητή (προτείνεται ο **EAX**) τη διεύθυνση στοιχείου πίνακα με την επιθυμητή τιμή ή 0 αν δεν βρεθεί τέτοιο στοιχείο

i) Η διαδικασία αναζήτησης θα περιέχει ένα βρόγχο επανάληψης για την διαπέραση όλων των στοιχείων του πίνακα (εκκινώντας από την διεύθυνση του πρώτου στοιχείου) με χρήση της εντολής **LOOP** όπου ο **ECX** είναι ο αυτόματος μετρητής. Εντός του βρόγχου η τιμή του εκάστοτε τρέχοντος στοιχείου θα συγκρίνεται με την τιμή προς αναζήτηση με χρήση της εντολή **CMP** και ακολούθως, με χρήση κατάλληλης εντολής άλματος υπό συνθήκη (σε περίπτωση που βρεθεί η αναζητούμενη τιμή), ο έλεγχος του προγράμματος θα

μεταφέρεται εκτός βρόγχου όπου και η διεύθυνση του τρέχοντος στοιχείου θα μεταφέρεται στον κατάλληλο register επιστρεφόμενης τιμής (προτείνεται ο EAX) και η διαδικασία θα τερματίζεται. Αν ο βρόγχος ολοκληρώσει την διαπέραση όλων των στοιχείων (δηλαδή δεν βρέθηκε η τιμή προς αναζήτηση) θα μεταφέρεται στον κατάλληλο register επιστρεφόμενης τιμής η τιμή 0 και η διαδικασία θα τερματίζεται. Προκειμένου η διαδικασία να καλείται με ασφάλεια, προστατέψτε το περιεχόμενο των register που χρησιμοποιεί, στην stack του προγράμματος, με χρήση είτε της οδηγία **USE** στη δήλωση της διαδικασίας, είτε με τις εντολές **PUSH** και **POP** στην αρχή και το τέλος της διαδικασίας

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

k) Για λόγους ομοιομορφίας ακολουθείστε το παρακάτω πρότυπο κώδικα:

```
TITLE  Expression calculator       (Ergasia_5_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 του κυρίως προγράμματος της άσκησης}
        exit
main ENDP
        {εντολές για την δήλωση επιπλέων διαδικασιών}
END main
```

Άσκηση 2: Μετατροπή του ανωτέρω προγράμματος (με όνομα **Ergasia_5_b.asm**) για τη δημιουργία μιας διαδικασίας αναζήτησης σε γλώσσα assembly, η οποία θα δέχεται ως παραμέτρους (μέσω arguments), α) την διεύθυνση (by reference) ενός πίνακα στοιχείων ακέραιων προσημασμένων αριθμών 32bit, β) τον αριθμό (by value) των στοιχείων του πίνακα, γ) μία ακέραια προσημασμένη (by value) τιμή 32bit προς αναζήτηση και θα επιστρέφει (μέσω argument), α) την (by reference) διεύθυνση του (πρώτου) στοιχείου του πίνακα, στην περίπτωση που έχει την ίδια τιμή με την τιμή προς αναζήτηση, β) μηδενική τιμή σε περίπτωση που η τιμή προς αναζήτηση δεν βρεθεί. Επιπλέον (ως εξάσκηση) στη διαδικασία θα δηλώνεται μία τοπική μεταβλητή (4bytes) στην οποία και θα προστατεύεται το περιεχόμενο του EAX register στην αρχή της διαδικασίας και αντίστοιχα θα αποκαθίσταται στο τέλος της διαδικασίας. Για την δοκιμή της διαδικασίας δημιουργήστε κυρίως πρόγραμμα με έναν πίνακα 10 στοιχείων με αρχικές τιμές -5d, 5d, -6d, -7d, 9d, -6d, 10d, -3d, 5d, -6d. Ο χρήστης θα εισάγει στο κυρίως πρόγραμμα μία τιμή αναζήτησης από το πληκτρολόγιο μεταξύ των τιμών [-10, +10] και το πρόγραμμα θα καλεί την διαδικασία αναζήτησης, κατόπιν θα εμφανίζει σχετικό μήνυμα ενημερώνοντας τον χρήστη για την επιτυχία (με εμφάνιση της διεύθυνσης του στοιχείου) ή όχι της αναζήτησης. Επίσης θα πραγματοποιείται ο έλεγχος οριακών τιμών εισόδου στην τιμή αναζήτησης που εισάγει ο χρήστης, όπου στην περίπτωση μη αποδεκτής τιμή ο χρήστης θα καλείται εκ νέου για την εισαγωγή τιμής.

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

b) Δημιουργία πηγαίου αρχείου assembly κώδικα με όνομα **Ergasia_5_b.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

c) Αρχικά, θα δηλώνονται σταθερές με συμβολικό όνομα **max_limit**=10d και **min_limit**=-10d (αποδεκτά όρια τιμών εισόδου) και μία σταθερά με συμβολικό όνομα **array_size**=10d (αριθμός στοιχείων πίνακα) με χρήση της οδηγίας **EQU**

d) Στη συνέχεια, στο data segment του προγράμματος, θα δηλώνονται α) ένας πίνακας ακεραίων προσημασμένων αριθμών 32bit με αρχικές τιμές -5d, 5d, -6d, -7d, 9d, -6d, 10d, -3d, 5d, -6d, β) τα κατάλληλα αλφαριθμητικά δεδομένα (πίνακες χαρακτήρων) για την εμφάνιση των σχετικών μηνυμάτων προς τον χρήστη, γ) μία μεταβλητή (με όνομα returned_value) για την αποθήκευση της διεύθυνσης του στοιχείου του πίνακα που βρέθηκε κατά την αναζήτηση (ως τιμή διεύθυνσης θα έχει τύπο DWORD)

e) Κατόπιν με κλήση των procedures **WriteString**, και **ReadInt** της βοηθητικής βιβλιοθήκης θα εμφανίζεται το μήνυμα στην οθόνη «**Enter a value (-10 to +10):**», όπου στη συνέχεια ο χρήστης θα εισάγει μία τιμή (πατώντας στο τέλος enter). Η τιμή που εισήγαγε ο χρήστης θα συγκρίνεται με το μέγιστο επιτρεπτό όριο (**max_limit**) με χρήση της εντολής **CMP** και ακολούθως με χρήση κατάλληλης εντολής άλματος υπό συνθήκη (JE, JNE, JA, JNA, JAE, JNAE, JB, JNB, JBE, JNBE, JG, JNG, JGE, JNGE, JL, JNL, JLE, JNLE), σε περίπτωση που (η τιμή) είναι εκτός ορίων η διαδικασία εισαγωγής θα επαναλαμβάνεται. Αντίστοιχα, η τιμή που εισήγαγε ο χρήστης θα συγκρίνεται με το ελάχιστο επιτρεπτό όριο (**min_limit**) με χρήση της εντολής **CMP** και ακολούθως με χρήση κατάλληλης εντολής άλματος υπό συνθήκη, σε περίπτωση που (η τιμή) είναι εκτός ορίων η διαδικασία εισαγωγής θα επαναλαμβάνεται.

f) Στη συνέχεια προκειμένου να καλέσουμε την διαδικασία αναζήτησης, θα μεταφέρουμε τις τιμές των παραμέτρων της στη stack του προγράμματος με χρήση της εντολής **PUSH** (προτείνεται η σειρά προώθησης διεύθυνση του – πρώτου στοιχείου – πίνακα, κατόπιν ο αριθμός των στοιχείων του πίνακα, κατόπιν η τιμή προς αναζήτηση και τέλος η διεύθυνση της μεταβλητής returned_value για την επιστροφή της διεύθυνσης του στοιχείου του πίνακα που βρέθηκε κατά την αναζήτηση). Κατόπιν θα καλείται η procedures **SearchArray** με χρήση της εντολής **CALL**

g) Κατόπιν θα συγκρίνεται η τιμή (της μεταβλητής returned_value) που επέστρεψε η διαδικασία αναζήτησης (διεύθυνση στοιχείου πίνακα με την επιθυμητή τιμή ή 0 αν δεν υπάρχει τέτοιο στοιχείο) με χρήση της εντολή **CMP** και ακολούθως, με χρήση κατάλληλης εντολής άλματος υπό συνθήκη, το πρόγραμμα θα εμφανίζει το μήνυμα «**The value hasn't been found**» σε περίπτωση που δεν βρέθηκε στοιχείο, ή το μήνυμα «**The value has been found at address (offset):**» ακολουθούμενο από την διεύθυνση του στοιχείου του πίνακα σε hexadecimal μορφή με χρήση της procedure **WriteHex** της βοηθητικής βιβλιοθήκης

h) Τέλος δημιουργούμε τη διαδικασία αναζήτησης με όνομα **SearchArray**, με χρήση των **PROC** και **ENDP** οδηγιών και της εντολής **RET**, η οποία δεχόμενη ως παραμέτρους τιμές μέσω της stack (την διεύθυνση του – πρώτου στοιχείου – πίνακα, τον αριθμό των στοιχείων του πίνακα, την τιμή προς αναζήτηση), θα συγκρίνει ένα προς ένα τα στοιχεία του πίνακα με την τιμή προς αναζήτηση και θα επιστρέφει, δια μέσου της τελευταίας παραμέτρου (διεύθυνση μεταβλητής returned_value), τη διεύθυνση στοιχείου πίνακα με την επιθυμητή τιμή ή 0 αν δεν βρεθεί τέτοιο στοιχείο. Αναλυτικότερα υλοποιείστε τα παρακάτω βήματα με τη σειρά που εμφανίζονται:

i) εφόσον η διαδικασία δέχεται παραμέτρους μέσω της stack, δημιουργούμε το Activation Record (AR) της διαδικασίας, αποθηκεύοντας (**PUSH**) τον EBP (base pointer) register στην stack και μεταφέροντας (**MOV**) τον ESP (stack pointer) register στον EBP register

ii) εφόσον η διαδικασία περιέχει τοπική μεταβλητή, δημιουργούμε κατάλληλο χώρο στην stack, μειώνοντας (**SUB**) την τιμή του ESP (stack pointer) register κατά το μέγεθος της τοπικής μεταβλητής

(4 bytes στην περίπτωσή μας)

iii) Προκειμένου η διαδικασία να καλείται με ασφάλεια, προστατέψτε το περιεχόμενο των register που χρησιμοποιεί, στην stack του προγράμματος, με την εντολή **PUSH** στην αρχή (και μετά τη δημιουργία του AR και των τοπικών μεταβλητών) της διαδικασίας (προσοχή, **όχι** με χρήση της οδηγία USE). Ειδικά για την προστασία του EAX register, αποθηκεύστε (**MOV**) την τιμή του, στην τοπική μεταβλητή της διαδικασίας κάνοντας έμμεση (**indirect**) αναφορά στη θέση EBP-4 του AR της διαδικασίας

iv) Μεταφέρετε (**MOV**) τις τιμές των παραμέτρων εισόδου της διαδικασίας σε ξεχωριστούς register προκειμένου στη συνέχεια να υλοποιηθεί η αναζήτηση. Η Αναφορά στις παραμέτρους της διαδικασίας πραγματοποιείται με έμμεση (**indirect**) αναφορά στις θέσεις

EBP+20 ($1^{η}$ παράμετρος) για την διεύθυνση του – πρώτου στοιχείου – πίνακα,
EBP+16 ($2^{η}$ παράμετρος) για τον αριθμό των στοιχείων του πίνακα,
EBP+12 ($3^{η}$ παράμετρος) για την τιμή προς αναζήτηση,

του AR της διαδικασίας (με βάση την σειρά που εισήχθηκαν στην stack από το καλλών πρόγραμμα)

v) Στη συνέχεια η διαδικασία αναζήτησης θα περιέχει ένα βρόγχο επανάληψης για την διαπέραση όλων των στοιχείων του πίνακα (εκκινώντας από την διεύθυνση του πρώτου στοιχείου) με χρήση της εντολής **LOOP** όπου ο **ECX** είναι ο αυτόματος μετρητής. Εντός του βρόγχου η τιμή του εκάστοτε τρέχοντος στοιχείου θα συγκρίνεται με την τιμή προς αναζήτηση με χρήση της εντολή **CMP** και ακολούθως, με χρήση κατάλληλης εντολής άλματος υπό συνθήκη (σε περίπτωση που βρεθεί η αναζητούμενη τιμή), ο έλεγχος του προγράμματος θα μεταφέρεται εκτός βρόγχου. Αν ο βρόγχος ολοκληρώσει την διαπέραση όλων των στοιχείων (δηλαδή δεν βρέθηκε η τιμή προς αναζήτηση) θα τίθεται ως τελευταία διεύθυνση επίσκεψης η τιμή 0.

vi) Η τιμή της τελευταίας διεύθυνσης επίσκεψης (του στοιχείου του πίνακα που προέκυψε από την αναζήτηση) θα μεταφέρεται (**MOV**) στην παράμετρο εξόδου της διαδικασίας. Η Αναφορά στην παράμετρο εξόδου της διαδικασίας πραγματοποιείται με έμμεση (**indirect**) αναφορά στη θέση

EBP+8 ($4^{η}$ παράμετρος) για τη διεύθυνση της μεταβλητής returned_value, (προσοχή, το περιεχόμενο μια by reference παραμέτρου απαιτεί δύο indirect αναφορές για να προσπελασθεί. Η indirect αναφορά [EBP+8] μας επιστρέφει την διεύθυνση της μεταβλητής returned_value, οπότε και απαιτείται μια επιπλέον indirect αναφορά για το περιεχόμενο της returned_value. Συμβουλευτείτε τις υποδείξεις της άσκηση για το θέμα αυτό)

του AR της διαδικασίας (με βάση την σειρά που εισήχθηκαν στην stack από το καλλών πρόγραμμα)

vii) Επαναφέρετε το περιεχόμενο των register που προστατέψατε, στην stack του προγράμματος, με την εντολή **POP** στην τέλος (και πριν τη διαγραφή του AR και των τοπικών μεταβλητών) της διαδικασίας (προσοχή, **όχι** με χρήση της οδηγία USE). Ειδικά για την επαναφορά του EAX register, μεταφέρετε (**MOV**) την τιμή του, από την τοπική μεταβλητή της διαδικασίας κάνοντας έμμεση (**indirect**) αναφορά στη θέση EBP-4 του AR της διαδικασίας

viii) εφόσον η διαδικασία περιέχει τοπική μεταβλητή, απελευθερώνουμε τον καταλαμβανόμενο χώρο στην stack, μεταφέροντας (**MOV**) την τιμή του EBP (base pointer) register στον ESP (stack pointer) register

ix) εφόσον η διαδικασία δέχεται παραμέτρους μέσω της stack, καταργούμε το Activation Record (AR) της διαδικασίας, ανακτώντας (**POP**) τον EBP (base pointer) register από την stack

x) η διαδικασία θα τερματίζεται με χρήση της εντολής **RET**, συνοδευόμενη από τον συνολικό αριθμό των bytes των παραμέτρων της, δηλαδή στην περίπτωσή μας 4 * 4bytes = **16** bytes. Με αυτόν τον τρόπο αποδεσμεύεται από την stack ο χώρος που καταλαμβάνουν οι παράμετροί της με ευθύνη της ίδια της διαδικασίας κατά το StdCall πρότυπο κλήσης

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

Υποδείξεις :

1) **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
```

2) **WriteInt** procedure: διαδικασία της βοηθητικής βιβλιοθήκης irvine32, η οποία εμφανίζει την τρέχουσα τιμή του EAX register στην οθόνη (console window) σε δεκαδική προσημασμένη μορφή (πχ. αν EAX=FFFFFFFDh τότε εμφανίζει την δεκαδική τιμή -3)

3) **WriteHex** procedure: διαδικασία της βοηθητικής βιβλιοθήκης irvine32, η οποία εμφανίζει την τρέχουσα τιμή του EAX register στην οθόνη (console window) σε hexadecimal μορφή

4) **ReadInt** 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 προσημασμένη μορφή και αποθηκεύει την τιμή του στον EAX register

5) 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]
```

6) **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

7) **CMP** : εντολή η οποία υλοποιεί αριθμητική σύγκριση μεταξύ δύο **integer** operand, χωρίς να τροποποιεί το περιεχόμενο κανενός. Πραγματοποιεί υπονοούμενη αφαίρεση του source από τον destination operand. Με τη χρήση της CMP μπορούμε υλοποιήσουμε λογικές εκφράσεις. Τροποποιεί τους Sign, Zero, Parity, Overflow, Carry και Auxiliary Carry Flags ανάλογα με την υπονοούμενη τιμή του αποτελέσματος

```
CMP destination,source
```

```
reg,reg
mem,reg
reg,mem
mem,imm
reg,imm
```

8) **JUMPS** Based on Specific Flag Values : εντολές αλμάτων υπό συνθήκη με βάση την τιμή συγκεκριμένων Flags (Zero, Carry, Overflow, Sign, Parity) της CPU. Εφαρμόζονται συνήθως μετά από αριθμητικές ή λογικές πράξεις / λειτουργίες καθώς και μετά από συγκρίσεις (εντολή CMP)

Mnemonic	Description	Flags / Registers
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

Παραδείγματα :

```
var1 DWORD 10d          ; data segment
ADD var1, -10d          ; 10+(-10) = 0
JNZ Label1              ; jumps if Zero Flag isn't enabled (not taken)
ADD var1, FFFFFFFFh     ; unsigned overflow -> enabling Cary Flag
JC Label2               ; jumps if Carry Flag  is enabled (taken)
```

9) **JUMPS** Based on Equality Comparisons : εντολές αλμάτων υπό συνθήκη με βάση το αποτέλεσμα

συγκρίσεων ισότητας. Εφαρμόζονται συνήθως μετά από συγκρίσεις (εντολές CMP, SUB)

```
CMP leftOp,rightOp
```

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

10) **JUMPS** Based on Unsigned Comparisons : εντολές αλμάτων υπό συνθήκη με βάση το αποτέλεσμα συγκρίσεων μη προσημασμένων τιμών. Εφαρμόζονται συνήθως μετά από συγκρίσεις (εντολή CMP)

```
CMP leftOp,rightOp
```

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* ≥ *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* ≤ *rightOp*)
JNA	Jump if not above (same as JBE)

Above Below

Παραδείγματα :

```
var1 DWORD 10d      ; data segment
CMP var1, 20d       ; 10 < 20
JA Label1           ; False
JAE Label1          ; False
JB Label1           ; True
JBE Label1          ; True
```

Εναλλακτικές χρήσεις

11) **JUMPS** Based on Signed Comparisons : εντολές αλμάτων υπό συνθήκη με βάση το αποτέλεσμα συγκρίσεων προσημασμένων τιμών. Εφαρμόζονται συνήθως μετά από συγκρίσεις (εντολή CMP)

```
CMP leftOp,rightOp
```

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* ≥ *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* ≤ *rightOp*)
JNG	Jump if not greater (same as JLE)

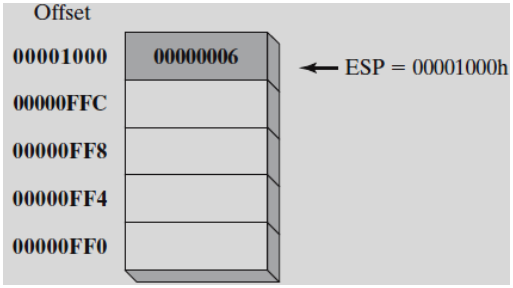
Greater Less

Παραδείγματα :

```
var1 SDWORD -30d    ; data segment
CMP var1, 20d       ; -30 < 20
JG Label1           ; False
JGE Label1          ; False
JL Label1           ; True
JLE Label1          ; True
```

Εναλλακτικές χρήσεις

12) **Runtime Stack** : πίνακας στην μνήμη διαχειριζόμενος απευθείας από την CPU (σε επίπεδο hardware), δια μέσου του ESP (Stack Pointer) register, ο οποίος περιέχει πάντα το 32-bit Offset της θέσης μνήμης όπου τοποθετήθηκε το τελευταίο στοιχείο της stack, δηλαδή δείχνει την κορυφή (top) της stack. Σπάνια τροποποιούμε απευθείας την τιμή του ESP register, η οποία έμμεσα μεταβάλλεται από εντολές όπως CALL, RET, PUSH, POP. Κάθε στοιχείο της stack έχει μέγεθος 32-bit σε protected mode



13) **PUSH** : εντολή για την αποθήκευση τιμών από την stack. Αρχικά μειώνει τον ESP register και στη συνέχεια αντιγράφει τον source operand στη stack

```
PUSH reg/mem16
PUSH reg/mem32
PUSH imm32
```

14) **POP** : εντολές για την ανάκτηση τιμών από την stack. Αρχικά αντιγράφει το περιεχόμενο του στοιχείου της stack που δείχνει ο ESP σε έναν 16bit ή 32bit operand και στη συνέχεια αυξάνει τον ESP register

```
POP reg/mem16
POP reg/mem32
```

15) **PROC** / **ENDP** / **USES** / **RET** : η δήλωση μιας procedure χρησιμοποιεί τις directives PROC και ENDP συνοδευόμενες από έναν identifier. Η startup procedure τελειώνει πάντα με την δήλωση exit. Όλες οι άλλες procedures τελειώνουν πάντα με την εντολή RET, η οποία οδηγεί τη CPU να επιστρέψει τον έλεγχο (ροή) του προγράμματος στη θέση από την οποία κλήθηκε. Ο USES operator συνδυαζόμενος με την PROC directive, επιτρέπει τον καθορισμό των register που μεταβάλλονται από μια procedure και επιθυμούμε να προστατεύσουμε τις τιμές τους (προσοχή δεν πρέπει να συμπεριλάβουμε τον register επιστρεφόμενης τιμής, αν υπάρχει τέτοιος). Ο USE operator στην πράξη παράγει εντολές PUSH (στην αρχή της procedure) και POP (στο τέλος της procedure) για την αποθήκευση και ανάκτηση των συγκεκριμένων register στην stack

```
main PROC
.
.
main ENDP
```

```
sample PROC
     .
     .
     ret
sample ENDP
```

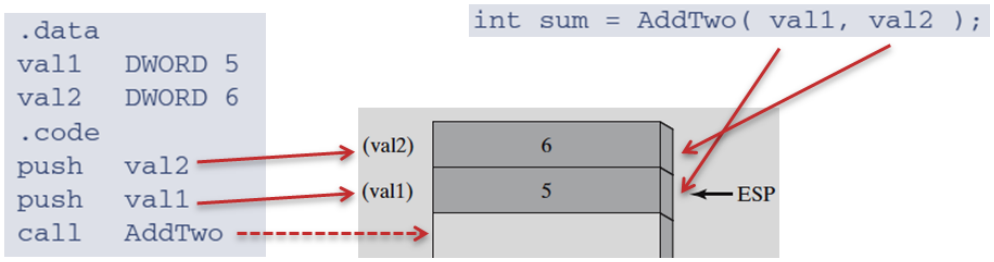
```
ArraySum PROC USES esi ecx
     mov   eax,0                ; set the sum to zero
L1:
     add   eax,[esi]            ; add each integer to sum
     add   esi,TYPE DWORD       ; point to next integer
     loop  L1                   ; repeat for array size

     ret                        ; sum is in EAX
ArraySum ENDP
```

16) **Activation Record (AR) / Stack Frame** : είναι το τμήμα της stack στο οποίο αποθηκεύονται τα arguments, η subroutine return address, οι local variables καθώς και οι saved registers μιας subroutine. Δημιουργείται από τα ακόλουθα κατά σειρά βήματα :

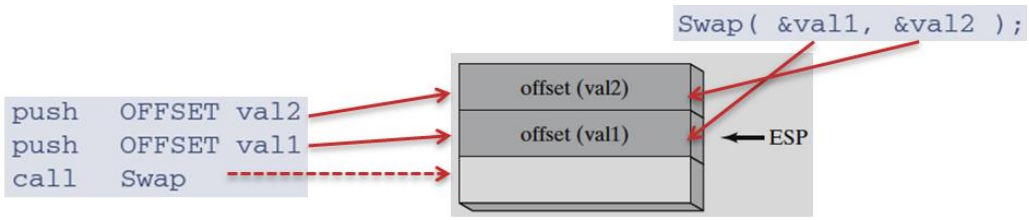
    a. Τα arguments, αν υπάρχουν προωθούνται (pushed) στην stack
    b. Καλείται (call) η subroutine, προκαλώντας την προώθηση της return address της subroutine στην stack
    c. Καθώς ξεκινά ή εκτέλεση της subroutine, ο EBP register προωθείται (pushed) στην stack
    d. Τίθεται ο EBP = ESP register, ώστε από αυτό το σημείο ο **EBP** (base pointer) να δρα ως **base register reference** για όλες τις παραμέτρους της subroutine
    e. Αν υπάρχουν local variables, ο ESP register μειώνεται για την παρακράτηση του απαιτούμενου χώρου για την φύλαξη των μεταβλητών στη stack
    f. Αν χρειάζεται η φύλαξη κάποιων registers, προωθούνται (pushed) στην stack

17) **Arguments** : Δύο γενικοί τύποι arguments τοποθετούνται (pushed) στην stack κατά την κλήση μιας subroutine α) **Value arguments** (τιμές μεταβλητών και σταθερές), β) **Reference arguments** (διευθύνσεις – offset μεταβλητών)

    Κατά το πέρασμα των τιμών (**passing by value**), ένα αντίγραφο της κάθε τιμής προωθείται (pushed) στην stack με <u>αντίστροφη σειρά</u> (σε σχέση με τη δήλωσή τους)



    Κατά το πέρασμα των αναφορών (**passing by reference**), η διεύθυνση (offset) των αντίστοιχων αντικειμένων προωθείται (pushed) στην stack, με <u>αντίστροφη σειρά</u> (σε σχέση με τη δήλωσή τους)

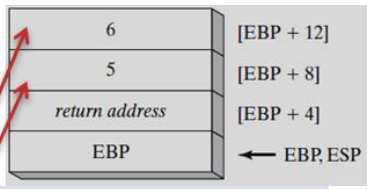


    Κατά το πέρασμα των πινάκων (**passing arrays**), χρησιμοποιείται η τεχνική **passing by reference,**

ώστε μόνο η διεύθυνση (offset) του πίνακα να προωθείται (pushed) στην stack

```
.data
array  DWORD 50 DUP(?)
.code
push   OFFSET array
call   ArrayFill
```

Για την προσπέλαση των παραμέτρων μπορούμε να χρησιμοποιήσουμε **base-offset (indirect) addressing** με βάση τον **EBP** (base of stack frame) και offset μια σταθερά (**constant**)




```
AddTwo PROC
    push  ebp
    mov   ebp,esp              ; base of stack frame
    mov   eax,[ebp + 12]       ; second parameter
    add   eax,[ebp + 8]        ; first parameter
    pop   ebp
    ret
AddTwo ENDP
```

ΠΡΟΣΟΧΗ: για την αφαίρεση των παραμέτρων από τη stack (με βάση το πρότυπο κλήσης STDCALL), στην εντολή ret της καλούμενης subroutine προσθέτουμε μία integer parameter με μια τιμή που ισούται με το συνολικό μέγεθος (bytes) των παραμέτρων. Η τιμή αυτή προστίθεται στον ESP μετά την επιστροφή στην αρχική procedure

```
AddTwo PROC
    push  ebp
    mov   ebp,esp              ; base of stack frame
    mov   eax,[ebp + 12]       ; second parameter
    add   eax,[ebp + 8]        ; first parameter
    pop   ebp
    ret   8                    ; clean up the stack
AddTwo ENDP
```

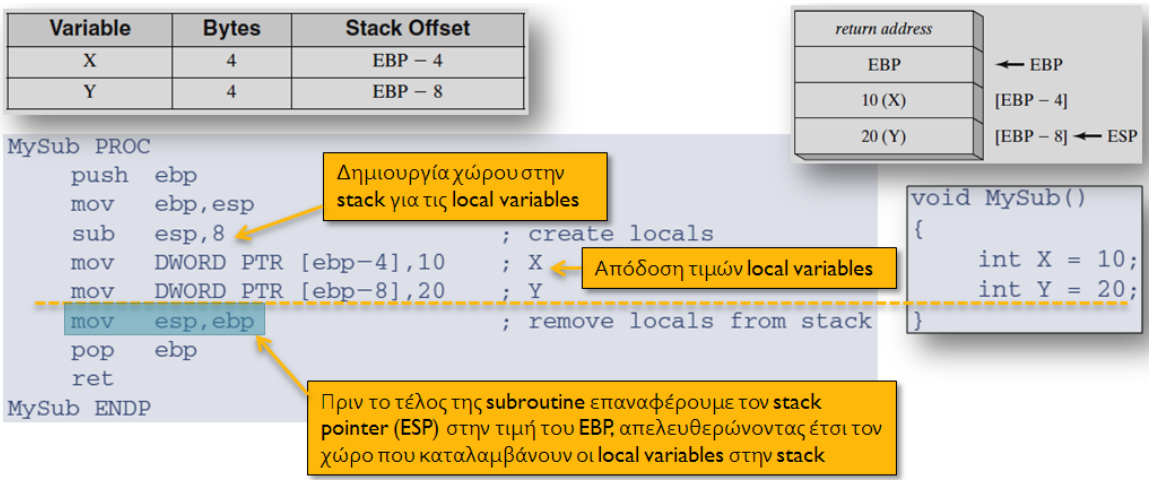
ΠΡΟΣΟΧΗ: Οι procedures που χρησιμοποιούν ρητές αναφορές παραμέτρων στην stack θα πρέπει να αποφεύγουν τη χρήση του USES operand

ΠΡΟΣΟΧΗ: Στην περίπτωση που θέλουμε να προσπελάσουμε το περιεχόμενο μιας by reference παραμέτρου τότε απαιτούνται δύο indirect αναφορές. Μία τύπου πχ [EBP+8] για την ανάκτηση της διεύθυνσης της παραμέτρου και κατόπιν ακόμη μία για την πρόσβαση στο περιεχόμενο της

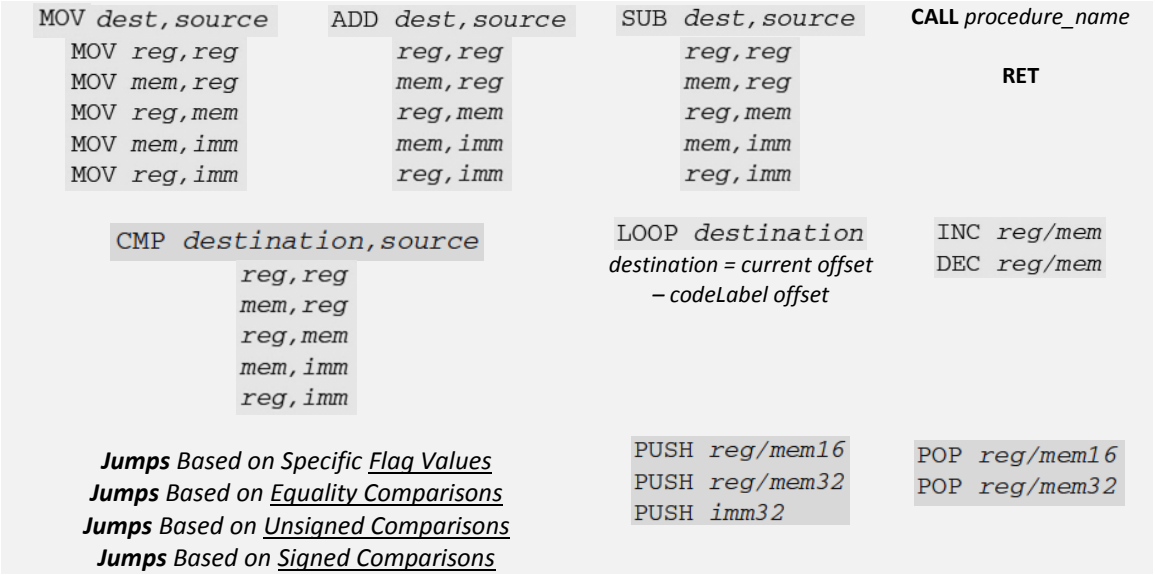
π.χ.
```
.data
        val1 DWORD ?
        val2 DWORD ?
.code
        ……
        PUSH OFFSET val2        ; by reference second arguments
        PUSH OFFSET val1        ; by reference first arguments
        CALL Swap
```

```
        ……
Swap PROC
        PUSH ebp
        MOV ebp, esp
        PUSH esi
        PUSH edi
        PUSH eax
        ……..
        MOV esi, [EBP+8]        ; ESI = offset of the first argument (val1)
        MOV eax, [esi]          ; EAX = content of the first argument (val1)
        ……..
        POP eax
        POP edi
        POP esi
        POP ebp
        RET 8
Swap ENDP
……..
```

18) **Local variables:** δημιουργούνται κατά το χρόνο εκτέλεσης του προγράμματος στην runtime stack, συνήθως κάτω από τον base pointer (EBP). Η αρχικοποίηση τους γίνεται επίσης σε πραγματικό χρόνο. Το μέγεθος αποθήκευσης (στη stack) κάθε local variable πρέπει να στρογγυλοποιείται προς τα πάνω σε πολλαπλάσιο του 4 (bytes).



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



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

Περιεχόμενα Γραπτής Αναφοράς

21) Κώδικας αρχείου **Ergasia_5_a.asm** της άσκησης 1
22) 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)
23) Κώδικας αρχείου **Ergasia_5_b.asm** της άσκησης 2
24) Αντίστοιχο Screen shot του visual studio με το (2)
25) Η εντολή CMP τι είδους υπονοούμενη λειτουργία πραγματοποιεί ?, πως οι εντολές αλμάτων υπό συνθήκη που ακολουθούν την εντολή κατανοούν το αποτέλεσμα της σύγκρισης που προηγήθηκε ?
26) Η εντολές XOR eax, ebx ακολουθούμενη από την εντολή JNZ Label1, τι είδους λειτουργία επιτελούν αναφορικά με το περιεχόμενο των εμπλεκόμενων registers ?, τεκμηριώστε την απάντησή σας