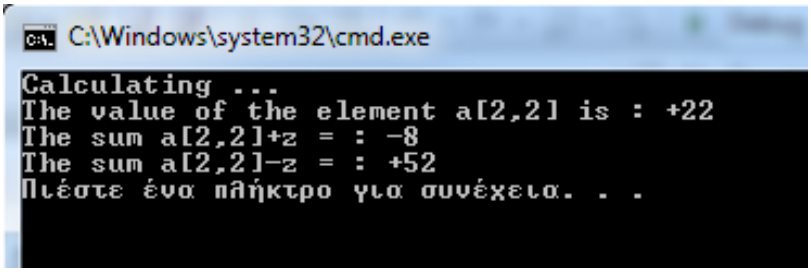
Τίτλος Άσκησης:	Δήλωση αλφαριθμητικών και πινάκων ακεραίων με άμεση και έμμεση αναφορά στη μνήμη σε γλώσσα Assembly x86
Εργαστήριο:	3
Απαραίτητα Εργαλεία:	Visual Studio Professional 2008 (προτεινόμενο) /2010/2012 (και εκδόσεις express)
Απαιτούμενες Γνώσεις:	Βασικοί τύποι δεδομένων Assembly, C/C++ Οι υποδείξεις των προηγούμενων ασκήσεων 1 και 2
Στόχοι:	1) Δήλωση και προβολή αλφαριθμητικών (string) 2) Προσδιορισμός μεγέθους, τύπου και διεύθυνσης μεταβλητών (TYPE, SIZEOF, LENGTHOF, OFFSET, PTR) 3) Δήλωση και πρόσβαση πινάκων ακεραίων αριθμών 4) Άμεση αναφορά τιμών (**direct memory access**) στη μνήμη 5) Έμμεση αναφορά τιμών (**indirect memory access**) στη μνήμη 6) Δήλωση και χρήση δεικτών (pointer) ως πολλαπλές έμμεσες αναφορές μνήμης (indirect memory access)
Διορία Παράδοσης:	Εντός της διδακτική ώρας του επόμενου εργαστηρίου
Παραδοτέα:	1) Επίδειξη 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 απλού υπολογισμού, ορίζοντας τους απαραίτητους πίνακες δεδομένων καθώς και αλφαριθμητικά (Strings) για την εμφάνιση σχετικών μηνυμάτων. Παράλληλα ο φοιτητής εξοικειώνεται με τον ορισμό και αρχικοποίηση δεδομένων πινάκων, τον ορισμό και την χρήση δεικτών (Pointer) σε δεδομένα, την άμεση και έμμεση αναφορά στη μνήμη για την προσπέλαση δεδομένων πινάκων από τις Assembly εντολές, καθώς και την χρήση ειδικών οδηγιών (directives) προς τον Assembler για τον προσδιορισμό του μεγέθους, του τύπου και της διεύθυνσης των μεταβλητών του προγράμματος.

Άσκηση 1: Δημιουργία ενός προγράμματος σε γλώσσα assembly στο οποίο θα ορίζεται ένας πίνακας (a) ακέραιων προσημασμένων αριθμών 32-bit, διαστάσεων 4x4 με συγκεκριμένες αρχικές τιμές (00d,10d,20d,30d,01d,11d,21d,31d,02d,12d,22d,32d,03d,13d,23d,33d) ανά γραμμή και στήλη, θα ορίζονται επίσης δύο μη προσημασμένες μεταβλητές 8-bit (x=2, y=2) με συγκεκριμένες αρχικές τιμές, μια μεταβλητή τύπου δείκτη (p_xy) και μια μεταβλητή 16-bit με αρχική τιμή (z=-30d). Το πρόγραμμα αρχικά θα υπολογίζει τη διεύθυνση (μετατόπιση) μνήμης του στοιχείου του πίνακα a(x,y) την οποία και θα αποθηκεύει στον δείκτη p_xy = offset(a(x,y)). Κατόπιν έχοντας ως αναφορά τον δείκτη p_xy θα ανακτά την αριθμητική τιμή a(x,y) στην οποία α) θα προσθέτει την τιμή z, δηλαδή (a(x,y)+z) και β) θα αφαιρεί την τιμή z, δηλαδή (a(x,y)-z). Τα δύο αποτελέσματα καθώς και το σύνολο των υπολογισμών θα εμφανίζονται στο console window του προγράμματος (με κλήση κατάλληλων procedure της βοηθητικής βιβλιοθήκης) με εμφάνιση κατάλληλων μηνυμάτων (θα πρέπει να ορισθούν τα κατάλληλα αλφαριθμητικά δεδομένα) με την παρακάτω μορφή εξόδου:

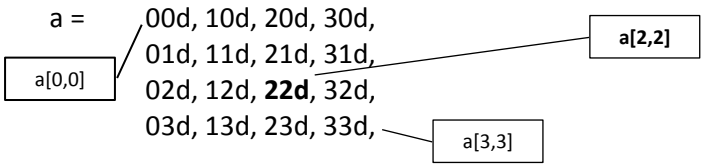


α) Δημιουργία νέου project για προγράμματα assembly ή χρήση προτύπου από προηγούμενη άσκηση

b) Δημιουργία πηγαίου αρχείου assembly κώδικα με όνομα **Ergasia_3_a.asm** (συμπερίληψη του αρχείου επικεφαλίδων irvine32.inc προκειμένου να μπορούν να κληθούν οι i/o procedures της βοηθητικής βιβλιοθήκης)

c) Αρχικά, στο data segment, θα δηλώνονται / ορίζονται τα δεδομένα των αλφαριθμητικών (Strings) μηνυμάτων που θα εμφανίζονται στο console window

c) Στη συνέχεια, επίσης στο data segment, θα δηλώνονται ο πίνακας a διαστάσεων 4x4 με συγκεκριμένες αρχικές τιμές προσημασμένου ακεραίου, μεγέθους 32-bit, θα δηλώνονται οι μεταβλητές με συμβολικά ονόματα x, y με συγκεκριμένες αρχικές τιμές τύπου μη προσημασμένου ακέραιου, μεγέθους 8-bit, η μεταβλητή z τύπου προσημασμένου ακεραίου, μεγέθους 16-bit με συγκεκριμένη αρχική τιμή, καθώς και η μεταβλητή p_xy τύπου δείκτη (μεγέθους 32-bit):



x = 2d, y = 2d , z = -30d, p = *χωρίς τιμή*

d) Κατόπιν, στο code segment, με δεδομένο τις τιμές x,y υπολογίζουμε με εντολές (run time) τη διεύθυνση (μετατόπιση από τη θέση a) του στοιχείου a[x,y] και κατόπιν το συνολικό offset του στοιχείου offset(a[x,y]), το οποίο και αποθηκεύουμε στον δείκτη p_xy.

1. μεταφέρουμε στον ESI τον αριθμό των στοιχείων ανά γραμμή του πίνακα (4 στην περίπτωσή μας – ο αριθμός μπορεί να εισαχθεί απευθείας σαν immediate value ή να υπολογισθεί με συνδυαστική χρήση των ειδικών οδηγιών π.χ. TYPE, LENGTHOF, SIZEOF, $)

2. πολλαπλασιάζουμε***** (αριστερή ολίσθηση κατά 1 θέση) τον ESI με τον δείκτη γραμμής πίνακα (2 στην περίπτωση μας). Δεν χρησιμοποιούμε την τιμή της μεταβλητής x προκειμένου να αποφύγουμε τη χρήση εντολών πολλαπλασιασμού. Πραγματοποιούμε έμμεσο πολλαπλασιασμό διαμέσου αριστερής ολίσθησης κατά ένα bit με την εντολή SHL. Στο σημείο αυτό ο ESI αναπαριστά τον αριθμό των στοιχείων 2 γραμμών του πίνακά μας (δηλαδή 8)

3. προσθέτουμε στον ESI τον δείκτη στήλης πίνακα (την τιμή y=2 στην περίπτωσή μας). Εδώ προσθέτουμε κανονικά την τιμή της μεταβλητής y στον ESI (Προσοχή κατά την πρόσθεση του 32-bit ESI με την 8-bit y μεταβλητή, μεταφέρετε (MOVZX / MOVSX) πρώτα την y στον ECX και στη συνέχεια προσθέστε τον ECX στον ESI). Στο σημείο αυτό ο ESI αναπαριστά τον αριθμό των στοιχείων 2 γραμμών και 2 στηλών του πίνακά μας (δηλαδή 10)

4. πολλαπλασιάζουμε (αριστερή ολίσθηση κατά 2 θέσεις) τον ESI με το μέγεθος σε bytes του τύπου στοιχείων του πίνακα (2^2=4 για 32-bit στην περίπτωση μας). Πραγματοποιούμε έμμεσο πολλαπλασιασμό διαμέσου αριστερής ολίσθησης κατά δύο bit με την εντολή SHL. Στο σημείο αυτό ο ESI αναπαριστά τον συνολικό μέγεθος σε bytes των στοιχείων 2 γραμμών και 2 στηλών του πίνακά μας (δηλαδή 10*4 = 40), δηλαδή την απόσταση σε bytes μεταξύ της θέσης a (πρώτο στοιχείο του πίνακα) και του επιθυμητού στοιχείου a[2,2]

5. προσθέτουμε στον ESI το offset του a (πρώτο στοιχείο του πίνακα). Στο σημείο αυτό ο ESI

αναπαριστά το offset του επιθυμητού στοιχείου του πίνακα offset(a[2,2])

6. Τέλος αποθηκεύουμε το περιεχόμενο του ESI στην μεταβλητή p_xy τύπου pointer (δείκτη)

(*) Αν ο απαιτούμενος πολλαπλασιασμός για τον υπολογισμό του offset της γραμμής του πίνακα δεν είναι με δύναμη του 2, τότε απαιτείται η χρήση εντολών πολλαπλασιασμού (ωστόσο δεν απαιτούνται στην συγκεκριμένη άσκηση).

e) Στη συνέχεια έχοντας ως αναφορά τον δείκτη p_xy, η αριθμητική τιμή a(x,y), θα μεταφέρεται στον καταχωρητή EAX και θα προστίθεται σε αυτόν η τιμή z, κατόπιν το αποτέλεσμα (a(x,y)+z) του EAX register θα εμφανίζεται στην οθόνη (console window) σε δεκαδική προσημασμένη μορφή με κλήση της procedure WriteInt (περιέχεται στη βοηθητική βιβλιοθήκη irvine32). Προκειμένου τα αποτελέσματα να εμφανίζονται με τη μορφή που ζητά η άσκηση θα γίνεται κατάλληλη κλήση των procedures της βοηθητικής βιβλιοθήκης WritwString (εμφάνιση αλφαριθμητικού), WriteInt (εμφάνιση προσημασμένου ακεραίου) και Crlf (αλλαγή γραμμής στην οθόνη)

Για την αναφορά στο περιεχόμενο (τιμή) του στοιχείου a[x,y] ή a[2,2] του πίνακα, αρχικά θα μεταφέρουμε το περιεχόμενο του pointer p_xy στον ESI και κατόπιν με έμμεση (indirect) αναφορά στη μνήμη [esi], θα μεταφέρουμε το περιεχόμενο (τιμή) του στοιχείου στον EAX

Προσοχή πάλι κατά την πρόσθεση του 32-bit EAX με την 16-bit z μεταβλητή, μεταφέρετε (MOVZX / MOVSX) πρώτα την z στον ECX και στη συνέχεια προσθέστε τον ECX στον EAX

f) Αντίστοιχα με τα προηγούμενα, έχοντας ως αναφορά τον δείκτη p_xy, η αριθμητική τιμή a(x,y), θα μεταφέρεται στον καταχωρητή EAX και θα αφαιρείται η τιμή z, κατόπιν το αποτέλεσμα (a(x,y)-z) του EAX register θα εμφανίζεται στην οθόνη (console window)

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

Για λόγους ομοιομορφίας ακολουθείστε το παρακάτω πρότυπο κώδικα:

```
TITLE  Expression calculator       (Ergasia_3.asm)
; This program declares and reads array elements
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_3_b.asm**) ώστε μετά την εμφάνιση όλων των αποτελεσμάτων όπως ζητούνται στην άσκηση 1, επιπρόσθετα θα εμφανίζει την τιμή του a(3,1) στοιχείου του πίνακα. Το μήνυμα που θα εμφανίζεται στο χρήστη θα έχει ανάλογη μορφή : "**The value of the element a[3,1] is : ….**".Η διεύθυνση (μετατόπιση) μνήμης του στοιχείου του πίνακα a(3,1) δεν θα υπολογίζεται run time αλλά θα δίνεται ως immediate τιμή σε έναν register της

CPU και στη συνέχεια με indirect αναφορά θα μεταφέρουμε την τιμή του στοιχείου στη CPU για εμφάνιση στην οθόνη.

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

b) Δημιουργία πηγαίου αρχείου assembly κώδικα με όνομα **Ergasia_3_b.asm** (συμπερίληψη του αρχείου επικεφαλίδων irvine32.inc προκειμένου να μπορούν να κληθούν οι i/o procedures της βοηθητικής βιβλιοθήκης)

c) Προσθέστε τον επιπλέον κώδικα (εντολές assembly) για τη λύση της άσκησης

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

Υποδείξεις :

1) **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. Ο τρόπος με τον οποίο οι γραμμές και οι στήλες ενός πολυδιάστατου πίνακα διατάσσονται στη γραμμική (μονοδιάστατη) μνήμη εξαρτάται από την υλοποίηση που επιλέγει ο κάθε μεταγλωττιστής. Συνηθέστερο πρότυπο διάταξης είναι η αποθήκευση ολόκληρων γραμμών διαδοχικά στη μνήμη:

   [3x3] = 1,2,3,
4,5,6,
7,8,9 → [9] = 1,2,3,4,5,6,7,8,9

   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

2) **Strings** : οι συμβολοσειρές στην assembly είναι πίνακες με τις ASCII 8-bit τιμές των χαρακτήρων τους ακολουθούμενα από το null byte (null-terminated string). Ο assembler μετατρέπει αυτόματα τους χαρακτήρες στις αντίστοιχες ASCII τιμές τις οποίες και αποθηκεύει στις αντίστοιχες θέσεις μνήμης

```
π.χ.    string1 BYTE 'H', 'e', 'l', 'l', 'o', 0
ή       string1 BYTE 'Hello', 0
ή       string1 BYTE "Hello", 0
```

οι hexadecimal τιμές 0Dh, 0Ah καλούνται CR/LF (carriage-return, line-feed ή EOL character) και μετακινούν τον κέρσορα της εξόδου στην αρχή της επόμενη γραμμής

```
π.χ.    string1 BYTE "Hello", 0Dh, 0Ah, 0
```

3) **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')
```

4) **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
```

5) **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
```

6) **LENGTHOF** : οδηγία που επιστρέφει τον αριθμό των στοιχείων ενός πίνακα

```
π.χ.    array3 WORD 10, 20, 30, 40      ; LENGTHOFF array3 = 4
        string4 BYTE "Hello",0;         ; LENGTHOFF array3 = 6
```

7) **SIZEOF** : οδηγία που επιστρέφει το συνολικό μέγεθος σε byte ενός πίνακα (LENGTHOFF * TYPE)

```
π.χ.    array3 WORD 10, 20, 30, 40      ; SIZEOFF array3 = 8
        string4 BYTE "Hello",0;         ; SIZEOFF array3 = 6
```

8) **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 λύση της άσκησης (ακολουθώντας τα προτεινόμενα βήματα υλοποίησης) χρησιμοποιεί τον απλούστερο τύπο indirect προσπέλασης. Η παραπάνω πλήρη ανάλυση παρουσιάζεται για εκπαιδευτικούς σκοπούς άλλα και για χρησιμοποίηση σε επόμενες ασκήσεις.

9) **PTR** : οδηγία με την οποία μπορούμε να προσπελάσουμε τα δεδομένα στη μνήμη, ανεξάρτητα του τρόπου που αυτά έχουν δηλωθεί ως μεταβλητές

```
π.χ.    var4 WORD 1234h          ; δήλωση στο data segment
        mov ax, var4             ; AX = 1234h
        mov al, var4             ; ERROR type mismatch (8-bit al <> 16bit var4)
        mov al, BYTE PTR var4    ; AL = 34h (λόγω του little endian order τρόπου αποθήκευσης)
```

Επίσης η χρήση της είναι απαραίτητη όταν από τους operands μιας εντολής ο assembler δεν μπορεί να συμπεράνει τον τύπο δεδομένων τον οποίο διαχειρίζεται. Αυτό συμβαίνει κυρίως στις έμμεσες αναφορές μνήμης όπου και δεν υπάρχει κάποιο όνομα μεταβλητής ή καταχωρητής προορισμού που να προσδιορίζει τον τύπο δεδομένων που αναφερόμαστε. Δηλαδή ο assembler δεν γνωρίζει πόσα bytes θα προσπελάσει (μεταφέρει) εκκινώντας από το offset της έμμεσης αναφοράς και χρειάζεται την PTR οδηγία να του το υποδείξει

```
π.χ.    mov ax, [esi]            ; ο τύπος του 16-bit destination register, υποδεικνύει στον
                                   assembler τον αριθμό των 2bytes που θα μεταφερθούν από
                                   τη μνήμη, δηλαδή εκκινώντας από το offset ESI, θα
                                   μεταφερθούν 2 bytes από τη μνήμη στον AX
        inc WORD PTR [esi]       ; η οδηγία WORD PTR γνωστοποιεί στον assembler ότι θα
                                   μεταφερθούν από τη μνήμη ένας ακέραιος μήκους 2 bytes
                                   προκειμένου η τιμή του να αυξηθεί από την εντολή inc
```

10) **MOVZX** : εντολή για μεταφορά δεδομένων από ένα operand μικρότερου μεγέθους σε έναν operand μεγαλύτερου μεγέθους, εισάγοντας από αριστερά μηδενικά bits για την πλήρωση των επιπλέων bytes. Διασφαλίζει ότι τα επιπλέον bytes του destination operand έχουν την επιθυμητή τιμή (χρησιμοποιείται συνήθων σε μη προσημασμένες τιμές)

```
π.χ.    mov bx, FFFFh            ; BX = FFFFh
        movzx eax, bx            ; EAX = 0000FFFFh
```

11) **MOVSX** : εντολή για μεταφορά δεδομένων από ένα operand μικρότερου μεγέθους σε έναν operand μεγαλύτερου μεγέθους, εισάγοντας από αριστερά τα bits του πρόσημου του source operand για την πλήρωση των επιπλέων bytes του destination operand. Διασφαλίζει ότι τα επιπλέον bytes του destination operand έχουν την επιθυμητή προσημασμένη τιμή

```
π.χ.    mov bx, FFFFh            ; BX = FFFFh
        movsx eax, bx            ; EAX = FFFFFFFFh
```

12) **SHL** : εντολή ολίσθησης των bits ενός operand προς τα αριστερά για συγκεκριμένο αριθμό bits, όπου το τελευταίο ολισθαίνων bit αποθηκεύεται στο carry flag (CF) και εισάγονται ‘0’ bits στο δεξιό μέρος

```
π.χ.    mov ax, 0000000011111111b    ; AX=0000000011111111b
        shl ax,1                     ; AX=0000000111111110b
        shl ax,2                     ; AX=0000011111111000b
```

Η ολίσθηση προς τα αριστερά πραγματοποιεί πολλαπλασιασμό με δυνάμεις του 2, ανάλογα με τα bits που ολισθαίνουν

13) **$** : σύμβολο το οποίο όταν εμφανίζεται στο data segment, αντικαθίσταται από τον assembler με το offset (μετατόπιση εντός του data segment) του τρέχοντος σημείου όπου και εμφανίζεται. Χρησιμοποιείται για τον ορισμό συμβολικών σταθερών στο πρόγραμμα σχετικά με το μέγεθος τμημάτων ή δομών δεδομένων

```
π.χ.   array2    DWORD 01, 02, 03
                 line_size = ($ - array2)    ; μέγεθος σε bytes μιας γραμμής του πίνακα
                 DWORD 11, 12, 13
                 DWORD 21, 22, 23
                 Total_size = ($ - array2) ; συνολικό μέγεθος σε bytes του πίνακα
```

14) **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
```

Ενδεικτικά παραδείγματα αναφοράς στο στοιχείο array2[2,1] σε πίνακα δύο διαστάσεων (3x3) :

```
π.χ.   array2  DWORD 01, 02, 03
               line_size = ($ - array2)
               DWORD 11, 12, 13
               DWORD 21, 22, 23        ; δήλωση data segment

       mov esi, line_size              ; ESI = μέγεθος bytes γραμμής πίνακα (3*4 = 12)
       shl esi, 1                      ; ESI = ESI * 2^1 =  24 = μέγεθος 2 γραμμών του πίνακα
       mov edi, 1                      ; EDI = δείκτης στήλης στοιχείου
       mov eax, [array2 + esi + edi * TYPE array2]      ; EAX = array1[2,1] = 22

       mov esi, 3                      ; ESI = 3 (αριθμός στοιχείων ανά γραμμή)
```

```
shl esi, 1                          ; ESI = ESI * 2^1 =  6 (αριθμός στοιχείων 2 γραμμών)
add esi, 1                          ; ESI = 7 (αριθμός στοιχείων 2 γραμμών + 1 στήλης)
mov eax, [array2 + esi * TYPE array2]     ; EAX = array1[2,1] = 22

mov esi, 3                  ; ESI = 3 (αριθμός στοιχείων ανά γραμμή)
shl esi, 1                  ; ESI = ESI * 2^1 =  6 (αριθμός στοιχείων 2 γραμμών)
add esi, 1                  ; ESI = 7 (αριθμός στοιχείων 2 γραμμών + 1 στήλης)
shl esi, 2                  ; ESI = ESI * 2^2 = 28 (συνολικά bytes 2 γραμμών + 1 στήλης)
add esi, OFFSET array2      ; ESI = offset(array2[2,1])
mov eax, [esi]              ; EAX = array1[2,1] = 22
```

Προφανώς και στην περίπτωση που ο απαιτούμενος πολλαπλασιασμός για τον υπολογισμό του offset της γραμμής του πίνακα δεν είναι με δύναμη του 2, τότε απαιτείται η χρήση εντολών πολλαπλασιασμού (ωστόσο δεν απαιτούνται στην συγκεκριμένη άσκηση). Γενικά για τον υπολογισμό του offset ενός στοιχείου σε πίνακα, καλό είναι να εκμεταλλευόμαστε στο έπακρο τη δυνατότητα του επεξεργαστή που παρέχει ο απευθείας (hardware) υπολογισμός – πολλαπλασιασμός δια μέσου του scale factor του indirect operand, με αποδεκτές τιμές 1, 2, 4, ή 8

15) **Pointer** : γενικά ένας δείκτης είναι μία μεταβλητή (θέση μνήμης) όπου και αποθηκεύεται η τιμή μίας διεύθυνσης μνήμης. Αντίστοιχα από την πλευρά της assembly ένας δείκτης είναι μία μεταβλητή (offset) όπου και αποθηκεύεται η τιμή ενός 32-bit offset (μετατόπιση εντός data segment). Ως μεταβλητή τύπου pointer μπορεί να θεωρηθεί οποιαδήποτε 32-bit μεταβλητή τύπου DWORD και στην οποία μπορεί να αποθηκευτεί ένα 32-bit offset

π.χ.
```
pointer1 DWORD (?)              ; δήλωση κενού δείκτη
array3 BYTE 1,2,3,4,5
pointer2 DWORD OFFSET array3    ; δήλωση δείκτη με αρχική τιμή το offset του array3

mov eax, OFFSET array3          ; EAX = offset(array3)
add eax, 2                      ; EAX = offset(array3[2])
mov pointer1, eax               ; αποθήκευση του offset(array3[2]) στον pointer1

mov esi, pointer1       ; ESI = offset(array3[2]) ανάκτηση offset από τον pointer1
mov eax, [esi]          ; EAX=array3[2]=3 (indirect αναφορά σε συγκεκριμένο στοιχείο)
```

16) **Βασικές εντολές assembly :** η γενική σύνταξη των απαιτούμενων assembly εντολών για την άσκηση έχουν ως εξής (**ΠΡΟΣΟΧΗ,** αρκεί και ένα υποσύνολο των εντολών για την επίλυση της άσκησης) :

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

MOVZX *reg32,reg/mem8*
MOVZX *reg32,reg/mem16*
MOVZX *reg16,reg/mem8*
		MOVSX *reg32,reg/mem8*
		MOVSX *reg32,reg/mem16*
		MOVSX *reg16,reg/mem8*	

SHL *destination,count* *reg,imm8* *mem,imm8* *reg,CL* *mem,CL*			

17) Τα εκτελέσιμα αρχεία **Project_Array_Reference_a.exe** και **Project_Array_Reference_b.exe** (περιλαμβάνονται στο υποστηρικτικό υλικό της άσκησης) παρουσιάζει το τελικό αποτέλεσμα εξόδου που αναμένεται από τις ασκήσεις

Περιεχόμενα Γραπτής Αναφοράς

1) Κώδικας αρχείου **Ergasia_3_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 όλες οι μεταβλητές του προγράμματος (δηλαδή strings και variables, ακριβώς πριν την εκτέλεση της εντολής τερματισμού του προγράμματος)
3) Κώδικας αρχείου **Ergasia_3_b.asm** της άσκησης 2
4) Η μεταβλητή z (με αρχική τιμή -30) με τι τιμές αναπαριστάται στη μνήμη του προγράμματος κατά την εκτέλεσή του ?, εξηγήστε τον τρόπο αναπαράστασης και αποθήκευσης
5) Αναφέρετε τις οδηγίες (directives) του MASM οι οποίες επιστρέφουν α) το μέγεθος σε bytes του κάθε στοιχείου, β) τον αριθμό των στοιχείων, γ) το συνολικό μέγεθος σε byte του πίνακα της άσκησης
6) Αν κατά τη μεταφορά της 16-bit μεταβλητής y στον 32-bit register, αλλάξετε τη χρήση της εντολή (από MOVZX σε MOVSX ή αντίστροφα) τι θα συμβεί ?, αναφέρετε τα αποτελέσματα που επιστρέφει το πρόγραμμά σας, εξηγήστε γιατί συμβαίνει αυτό ?