ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Οδηγός Εκμάθησης στην Assembly 8086[1]

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http:// arch.icte.uowm.gr/mdasyg

[1] Μετάφραση του κειμένου **8086 Assembler Tutorial for Beginners**
http:// www.itipacinotti.it/pagine/sistemi2008/documentation_emulator/asm_tutorial_01.html
Σε συνεργασία με τους: Βεντούρη Αντώνιο & Μπάτσου Ελευθερία

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα του Πανεπιστημίου Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Περιεχόμενα

1. Βοήθημα στην Assembler για αρχαρίους

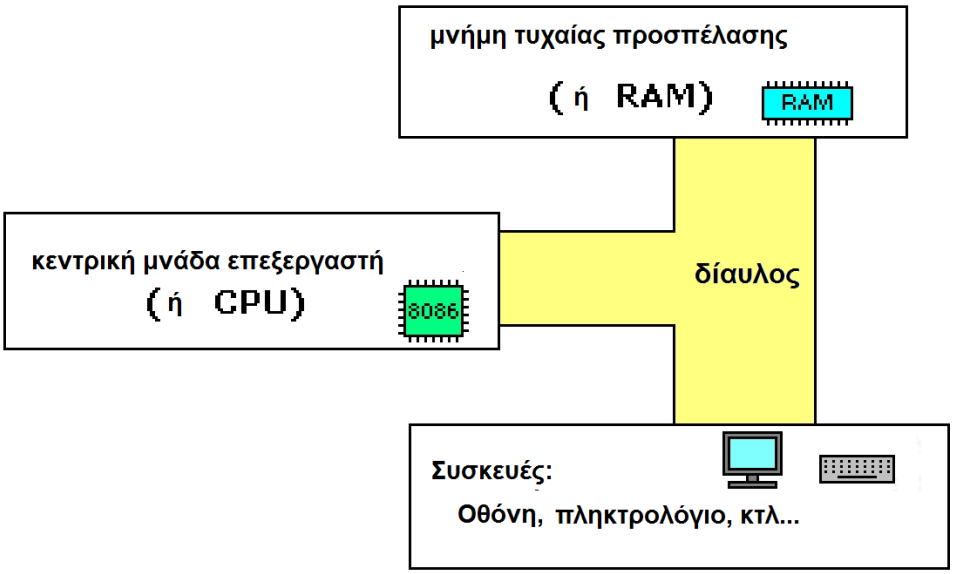
Το βοήθημα εκμάθησης αναφέρεται σε αυτούς που δεν γνωρίζουν καθόλου την assembly ή έχουν πολύ μικρή ιδέα για αυτήν. Φυσικά αν υπάρχει γνώση κάποιας άλλης γλώσσας προγραμματισμού *(Basic, C/C++, Pascal,...)* θα μπορούσε να βοηθήσει αρκετά.

Αλλά ακόμα και αν κάποιος γνωρίζει την assembler είναι καλό να δει αυτό το έγγραφο, ώστε να μελετήσει την σύνταξη του emu8086.

Θεωρείται ότι υπάρχει γνώση σχετικά με την αναπαράσταση των αριθμών *(HEX/BIN),* αν δεν υπάρχει προτείνεται ιδιαίτερα να μελετηθεί το **Numbering Systems Tutorial** πριν συνεχίσετε.

1.1 Τι είναι μια γλώσσα assembly;

Η γλώσσα assembly είναι μια γλώσσα προγραμματισμού χαμηλού επιπέδου. Θα χρειαστεί κάποια γνώση για την δομή του υπολογιστή ώστε να κατανοηθεί το οτιδήποτε. Ένα απλό μοντέλο υπολογιστή είναι αυτό:

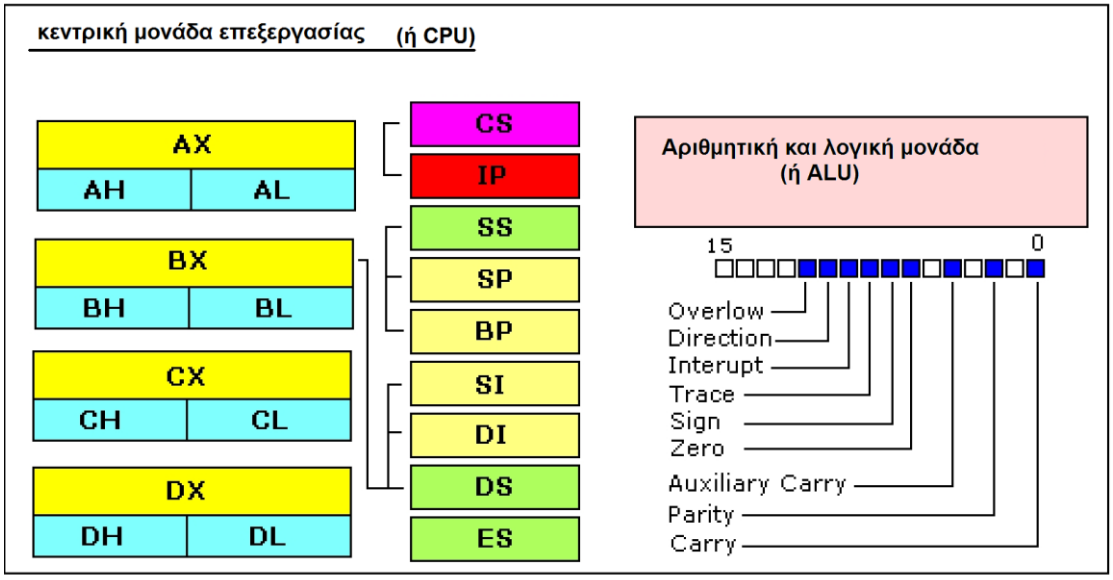


Ο **Δίαυλος** του συστήματος *(φαίνεται με το κίτρινο χρώμα)* συνδέεται με τα διάφορα μέρη του υπολογιστή.

Η **CPU** είναι η καρδιά του υπολογιστή, οι περισσότεροι υπολογισμοί εκτελούνται μέσα σε αυτήν.

Η **RAM** είναι το μέρος που φορτώνονται τα προγράμματα ώστε να εκτελεστούν.

1.2 Εσωτερικό της CPU



1.2.1 Καταχωρητές γενικού σκοπού

Στην 8086 CPU υπάρχουν 8 καταχωρητές γενικού σκοπού, κάθε καταχωρητής έχει το δικό του όνομα.

- **AX** – καταχωρητής συσσώρευσης *(χωρίζεται σε AH/AL).*
- **BX** – καταχωρητής βάσης *(χωρίζεται σε BH/BL).*
- **CX** – καταχωρητής μέτρησης *(χωρίζεται σε Ch/CL).*
- **DX** – καταχωρητής δεδομένων *(χωρίζεται σε DH/DL).*
- **SI** – καταχωρητής δείκτης προέλευσης.
- **DI** – καταχωρητής δείκτης προορισμού.
- **BP** – δείκτης βάσης.
- **SP** – δείκτης σωρού.

Ανεξάρτητα από το όνομα ενός καταχωρητή, ο προγραμματιστής είναι αυτός που καθορίζει την χρήση κάθε γενικού σκοπού καταχωρητή. Ο κύριος σκοπός ενός καταχωρητή είναι να κρατήσει έναν αριθμό (μεταβλητή). Το μέγεθος των παραπάνω καταχωρητών είναι 16 bit, είναι κάτι σαν: **0011000000111001b** *(σε δυαδική μορφή),* ή **12345** σε δεκαδική *(ανθρώπινη)* μορφή.

Οι 4 καταχωρητές γενικού σκοπού *(AX, BX, CX, DX)* είναι φτιαγμένοι από δύο ξεχωριστούς 8 bit καταχωρητές, για παράδειγμα αν **AX = 0011000000111001b**, τότε είναι **AH = 00110000b** και **AL = 00111001b**. Συνεπώς αν τροποποιήσουμε κάποιον από τους 8 bit καταχωρητές αλλάζει και ο 16 bit καταχωρητής και το αντίθετο. Το ίδιο ισχύει και για τους άλλους 3 καταχωρητές, το **"H"** είναι για το high *(υψηλό)* και το **"L"** για το low *(χαμηλό)* μέρος.

Επειδή οι καταχωρητές βρίσκονται στο εσωτερικό της CPU, είναι πολύ γρηγορότεροι από τη μνήμη. Η πρόσβαση σε μία θέση μνήμης απαιτεί τη χρησιμοποίηση ενός διαύλου, οπότε και χρειάζεται παραπάνω χρόνο. Η πρόσβαση στα δεδομένα ενός καταχωρητή συνήθως δεν απαιτεί χρόνο. Γι αυτό πρέπει να γίνεται προσπάθεια να χρησιμοποιούνται οι καταχωρητές για την αποθήκευση των δεδομένων. Το σύνολο των καταχωρητών είναι πολύ μικρό και οι περισσότεροι έχουν μια ειδική λειτουργία, κάτι που περιορίζει την χρήση τους σαν μεταβλητές, αλλά εξακολουθούν να αποτελούν ένα κάλο μέρος για προσωρινή αποθήκευση των δεδομένων.

1.2.2 Καταχωρητές τμήματος

- **CS** – δείχνει το τμήμα που περιέχει το τρέχων πρόγραμμα.
- **DS** – γενικά δείχνει το τμήμα που ορίζονται οι μεταβλητές.
- **ES** – καταχωρητής extra τμήματος, η χρήση του εξαρτάται από τον προγραμματιστή.
- **SS** – δείχνει το τμήμα που περιέχεται η σωρός.

Ακόμα και αν είναι δυνατή η αποθήκευση όλων των δεδομένων στους καταχωρητές τμήματος, αυτό δεν είναι ποτέ καλή ιδέα. Οι καταχωρητές τμήματος έχουν πολύ συγκεκριμένο σκοπό-να δείχνουν σε προσβάσιμα τμήματα της μνήμης.

Οι καταχωρητές ειδικού σκοπού λειτουργούν μαζί με τους καταχωρητές γενικής χρήσης για να έχουν πρόσβαση σε οποιαδήποτε θέση μνήμης. Για παράδειγμα αν θέλουμε να έχουμε πρόσβαση στην μνήμη με φυσική διεύθυνση **12345h** *(δεκαεξαδικό)*, θα πρέπει να θέσουμε τον **DS=1230h** και τον **SI=0045h**. Αυτό είναι επιθυμητό, καθώς με αυτό τον τρόπο μπορούμε να έχουμε πρόσβαση σε πολύ περισσότερη μνήμη από ότι με έναν μόνο καταχωρητή που περιορίζεται σε τιμές 16 bit.

Η CPU κάνει τον υπολογισμό της φυσικής διεύθυνσης πολλαπλασιάζοντας το τμήμα κώδικα με 10h και προσθέτοντας σε αυτό τον καταχωρητή γενικού σκοπού *(1230h * 10h + 45h = 12345h)***:**

```
 12300
+ 0045
-------
 12345
```

Η διεύθυνση που αποτελείται από 2 καταχωρητές ονομάζεται **ενεργός διεύθυνση**. Εξ ορισμού οι καταχωρητές **BX**, **SI** και **DI** ταιριάζουν με τον καταχωρητή τμήματος **DS**, οι καταχωρητές **BP** και **SP** ταιριάζουν με τον καταχωρητή τμήματος **SS**. Οι άλλοι καταχωρητές γενικού σκοπού δεν μπορούν να έχουν ενεργό διεύθυνση. Ακόμα και αν ο **BX** έχει ενεργό διεύθυνση, ο **BH** και **BL** δεν έχουν**!**

1.2.3 Καταχωρητές ειδικού σκοπού

- **IP** – δείκτης εντολής.
- **Flags Register** – καταχωρητής σημαιών.

Ο καταχωρητής **IP** είναι πάντα με τον καταχωρητή κώδικα **CS** και δείχνει την τρέχουσα εντολή.

Οι σημαίες των καταχωρητών τροποποιούνται αυτόματα από την CPU μετά από μαθηματικές πράξεις, αυτό επιτρέπει να καθοριστεί ο τύπος του αποτελέσματος και να καθοριστούν οι συνθήκες για να μεταφερθεί ο έλεγχος σε άλλα σημεία του προγράμματος. Γενικά δε μπορούμε να έχουμε άμεση πρόσβαση σε αυτούς τους καταχωρητές.

2. Πρόσβαση στην Μνήμη

Για να έχουμε πρόσβαση στην μνήμη μπορούμε να χρησιμοποιήσουμε αυτούς τους 4 καταχωρητές: **BX**, **SI**, **DI**, **BP**. Συνδυάζοντας αυτούς τους καταχωρητές μέσα στις **[ ]** αγκύλες, μπορούμε να έχουμε διαφορετικές θέσεις μνήμης. Αυτοί οι συνδυασμοί είναι *(διευθυνσιοδότηση):*

[BX + SI]	[SI]	[BX + SI] + d8
[BX + DI]	[DI]	[BX + DI] + d8
[BP + SI]	d16 *(ενεργός διεύθυνση)*	[BP + SI] + d8
[BP + DI]	[BX]	[BP + DI] + d8
[SI] + d8	[BX + SI] + d16	[SI] + d16
[DI] + d8	[BX + DI] + d16	[DI] + d16
[BP] + d8	[BP + SI] + d16	[BP] + d16
[BX] + d8	[BP + DI] + d16	[BX] + d16

d8 – δηλώνει 8 bit μετατόπισης.

d16 – δηλώνει 16 bit μετατόπισης.

Η μετατόπιση μπορεί να είναι μια σταθερά ή ενεργός διεύθυνση ή και τα δύο. Ο υπολογισμός μιας άμεσης τιμής γίνεται μέσω του compiler.

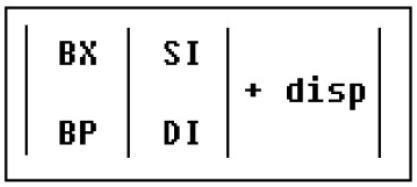
Η μετατόπιση μπορεί να είναι εσωτερικά ή εξωτερικά του συμβόλου **[ ]**, ο compiler δημιουργεί τον ίδιο κώδικα μηχανής και για τις δύο περιπτώσεις.

Η μετατόπιση είναι μια **προσημασμένη** τιμή, οπότε μπορεί να είναι είτε θετική είτε αρνητική.

Γενικά ο compiler αναγνωρίζει την διαφορά μεταξύ του **d8** και του **d16** και δημιουργεί τον κατάλληλο κώδικα μηχανής.

Για παράδειγμα, ας υποθέσουμε ότι **DS=100, BX=30, SI=70**. Η παρακάτω διευθυνσιοδότηση **[BX + SI] + 25** είναι υπολογισμένη με επεξεργασία της φυσικής διεύθυνσης: **100 * 16 + 30 + 70 + 25 = 1725**.

Εξ ορισμού ο καταχωρητής τμήματος **DS** χρησιμοποιείται για όλες τις λειτουργίες, αλλά δεν συνδυάζεται με τον καταχωρητή **BP**, γι αυτόν χρησιμοποιείται ο καταχωρητής τμήματος **SS**. Υπάρχει ένας εύκολος τρόπος να θυμάστε όλους τους πιθανούς συνδυασμούς, χρησιμοποιώντας το παρακάτω διάγραμμα:



Μπορείτε να σχηματίσετε όλους τους έγκυρους συνδυασμούς με την λήψη μόνο ενός στοιχείου από κάθε στήλη ή παρακάμπτοντας την στήλη χωρίς να παίρνετε κάτι από αυτή. Όπως είναι φανερό ο **BX** και ο **BP** δεν χρησιμοποιούνται ποτέ μαζί. Το ίδιο ισχύει και για το **SI** και το **PI.** Εδώ είναι ένα παράδειγμα έγκυρης διευθυνσιοδότησης: **[BX+5].**

Η τιμή του καταχωρητή τμήματος (CS, DS, SS, ES) ονομάζεται **"τμήμα"** *(segment)*, και η τιμή του καταχωρητή δείκτη ονομάζεται **"ενεργός διεύθυνση"** *(offset)*. Όταν ο DS περιέχει την τιμή **1234h** και ο SI την τιμή **7890h** μπορεί να καταγραφεί σαν **1234:7890**. Η φυσική διεύθυνση θα είναι 1234h * 10h + 7890h = 19BD0h.

Για να μπορέσει ο compiler να αναγνωρίσει τον τύπο δεδομένων πρέπει να χρησιμοποιηθούν τα παρακάτω προθέματα:

BYTE PTR – για byte.

WORD PTR – για λέξη *(δύο bytes)*.

Για παράδειγμα:

```
        BYTE PTR [BX]       ;πρόσβαση σε byte.
ή
        WORD PTR [BX]       ;πρόσβαση σε λέξη.
```

Επίσης υποστηρίζονται από το *MicroAsm* μικρότερα προθέματα:

b. – για BYTE PTR

w. – για WORD PTR

Μερικές φορές ο compiler υπολογίζει τον τύπο δεδομένων αυτόματα, ωστόσο δεν θα πρέπει να βασίζεστε σε αυτό ειδικά όταν ένας από τους τελεστές έχει άμεση τιμή.

Εντολή MOV

- Αντιγράφει τον **δεύτερο τελεστή** *(πηγή)* στον **πρώτο τελεστή** *(προορισμό).*
- Ο τελεστής της πηγής μπορεί να είναι μία άμεση τιμή, καταχωρητής γενικού σκοπού ή μια θέση μνήμης.
- Ο καταχωρητής προορισμού μπορεί να είναι καταχωρητής γενικού σκοπού ή θέση μνήμης.
- Και οι δύο τελεστές πρέπει να έχουν το ίδιο μέγεθος, που μπορεί να είναι ένα byte ή μία λέξη.

Επιτρέπονται οι εξής μετακινήσεις**:**

```
MOV ΚΑΤΑΧΩΡΗΤΗΣ, θέση μνήμης
MOV θέση μνήμης, ΚΑΤΑΧΩΡΗΤΗΣ
MOV ΚΑΤΑΧΩΡΗΤΗΣ, ΚΑΤΑΧΩΡΗΤΗΣ
MOV θέση μνήμης, άμεση τιμή
MOV ΚΑΤΑΧΩΡΗΤΗΣ, άμεση τιμή
```

ΚΑΤΑΧΩΡΗΤΕΣ: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

Θέση μνήμης: [BX], [BX+SI+7], μεταβλητή, κτλ...

Άμεση τιμή: 5, -24, 3Fh, 10001101b,κτλ...

Οι καταχωρητές τμήματος υποστηρίζουν μόνο τρείς τύπους MOV **:**

```
MOV ΚΑΤ. ΤΜΗΜΑΤΟΣ, θέση μνήμης
MOV θέση μνήμης, ΚΑΤ. ΤΜΗΜΑΤΟΣ
MOV ΚΑΤΑΧΩΡΗΤΗΣ, ΚΑΤ. ΤΜΗΜΑΤΟΣ
MOV ΚΑΤ. ΤΜΗΜΑΤΟΣ, ΚΑΤΑΧΩΡΗΤΗΣ
```

ΚΑΤ. ΤΜΗΜΑΤΟΣ: DS, ES, SS, και μόνο σαν δεύτερη παράμετρος: CS.

ΚΑΤΑΧΩΡΗΤΕΣ: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

Θέση μνήμης: [BX], [BX+SI+7], μεταβλητή, κτλ...

Η εντολή **MOV** δεν μπορεί να χρησιμοποιηθεί για να αλλάξει την τιμή του καταχωρητή **CS** και **IP**.

Εδώ είναι ένα μικρό πρόγραμμα που δείχνει την χρήση της εντολής MOV:

```
#MAKE_COM# ; ο compiler κάνει το COM file.
ORG 100h   ; άμεση τιμή για το πρόγραμμα COM. MOV AX, 0B800h
           ; θέτει στον AX την
           ; δεκαεξαδική τιμή B800h.
MOV DS, AX ; μεταφορά τιμή από AX στο DS.
MOV CL, 'A';θέτει στο CL την ASCII τιμή
           ;του 'A', που είναι41h.
MOV CH, 01011111b     ; θέτει στο CH σε
                      ;δυαδική μορφή.
MOV BX, 15Eh      ; θέτει στο BX το 15Eh.
MOV [BX], CX      ; μεταφορά του περιεχομένου του CX
                  ; στην θέση μνήμης  B800:015E
RET   ;επιστροφή στο λειτουργικό σύστημα.
```

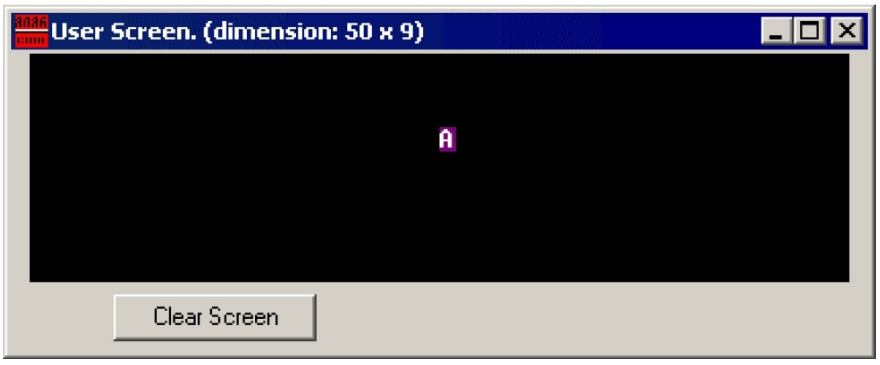
Μπορείτε να αντιγράψετε και να επικολλήσετε το παραπάνω πρόγραμμα στην επεξεργασία κώδικα *MicroAsm*, και να πιέσετε το πλήκτρο [**Compile**] *(ή να πιέσετε F5 από το πληκτρολόγιο σας).*

Πώς να κάνετε **αντιγραφή & επικόλληση :**

1. Επιλέξτε το παραπάνω κείμενο χρησιμοποιώντας το ποντίκι, κάντε κλικ στην αρχή του κειμένου και τραβήξτε το προς τα κάτω μέχρι να είναι όλα επιλεγμένα.
2. Πιέστε τον συνδυασμό πλήκτρων **Ctrl + C** για να αντιγραφούν.
3. Πηγαίνετε στο πρόγραμμα επεξεργασίας κώδικα *MicroAsm* και πατήστε τον συνδυασμό πλήκτρων **Ctrl + V** για να επικολληθεί.

Όπως είναι φανερό, το "**;**" χρησιμοποιείται για σχόλια, οτιδήποτε μετά το σύμβολο "**;**" αγνοείται από τον compiler.

Όταν τελειώσει το πρόγραμμα πρέπει να δείτε κάτι τέτοιο:



(Έτσι φαίνεται στο **emu8086** microprosessor emulator).

Η βασική λειτουργία αυτού του προγράμματος είναι η εγγραφή απευθείας στην μνήμη, όποτε πιθανό να καταλαβαίνετε ότι η MOV είναι μια πολύ ισχυρή εντολή.

3. Μεταβλητές

Η μεταβλητή είναι μία θέση μνήμης. Για έναν προγραμματιστή είναι πολύ πιο εύκολο να έχει μια τιμή αποθηκευμένη σε μια μεταβλητή με το όνομα **"var1"** στην διεύθυνση 5A73:235B, ειδικά όταν έχει 10 ή παραπάνω μεταβλητές.

Ο compiler υποστηρίζει δύο τύπους μεταβλητών: **BYTE** και **WORD**.

Σύνταξη για δήλωση μιας μεταβλητής:

Όνομα **DB** τιμή

Όνομα **DW** τιμή

DB – προέρχεται από το **D**efine **B**yte.

DW– προέρχεται από το **D**efine **W**ord.

όνομα- μπορεί να είναι οποιοδήποτε γράμμα ή συνδυασμός ψηφίων, ωστόσο πρέπει να ξεκινάει από γράμμα. Μπορείτε να δηλώσετε μεταβλητή χωρίς συγκεκριμένο όνομα *(αυτή η μεταβλητή θα έχει διεύθυνση αλλά όχι όνομα)*.

τιμή – μπορεί να είναι οποιαδήποτε αριθμητική τιμή *(δεκαεξαδική, δυαδική, δεκαδική)*, ή το σύμβολο "?" για μεταβλητές που δε έχουν αρχικοποιηθεί.

Όπως γνωρίζουμε από το 2ο μέρος αυτού του προγράμματος εκμάθησης, η εντολή **MOV** χρησιμοποιείται για να μεταφέρει τις τιμές από την πηγή στον προορισμό.

Ας δούμε ένα ακόμα παράδειγμα με την εντολή MOV:

```
#MAKE_COM#

ORG 100

MOV AL, var1
MOV BX, var2
RET ;σταματάει το πρόγραμμα.

VAR1 DB 7
var2 DW 1234h
```

Αντιγράψτε το παραπάνω πρόγραμμα στο *MicroAsm* source editor, και πιέστε F5 για να γίνει compile. Στην συνέχεια ανοίξτε το εκτελέσιμο σε οποιαδήποτε πρόγραμμα assembly *(**emu8086** ή κάποιο άλλο).*

Ο compiler δεν είναι ιδιαίτερα ευαίσθητος, δηλαδή η **"VAR1"** και η **"var1"** αναφέρονται στην ίδια τιμή.

Η ενεργός διεύθυνση της **VAR1** είναι **0108h**. Η ενεργός διεύθυνση της **VAR2** είναι **0109h**, αυτή η μεταβλητή είναι **WORD** οπότε και καταλαμβάνει χώρο **2 BYTES**. Θεωρείται δεδομένο ότι το low byte τοποθετείται στην χαμηλότερη διεύθυνση, οπότε το **34h** τοποθετείται πριν το **12h**.

Μπορείτε να δείτε ότι υπάρχουν κάποιες εντολές πριν την οδηγία **RET**, αυτό συμβαίνει διότι η assembly δεν γνωρίζει που αρχίζουν τα δεδομένα, επεξεργάζεται μόνο τις τιμές στην μνήμη και τις καταλαβαίνει σαν έγκυρες εντολές της 8086 *(θα τις μάθουμε αργότερα).*

Μπορείτε να γράψετε ένα πρόγραμμα χρησιμοποιώντας απευθείας μόνο οδηγίες DB:

```
#MAKE_COM#
ORG 100

DB 0A0h
DB 08h
DB 01h

DB 8Bh
DB 1Eh
DB 09h
DB 01h

DB 0C3h
DB 7

DB 34h
DB 12h
```

Αντιγράψτε το παραπάνω πρόγραμμα στην επεξεργασία κώδικα *MicroAsm*, και πιέστε **F5** για να γίνει compile, στην συνέχεια φορτώστε το στον προσομοιωτή *(emulator).* Θα πρέπει να εμφανιστεί ο ίδιος κώδικας συμβολομεταφρασμένος και με την ίδια λειτουργικότητα.

Όπως μπορεί να μαντεύετε, ο compiler μόλις μετέτρεψε τον πηγαίο κώδικα σε ένα σύνολο από bytes, αυτό το σύνολο ονομάζεται **κώδικας μηχανής**, ο επεξεργαστής κατανοεί τον **κώδικα μηχανής** και τον εκτελεί.

Το **ORG 100h** είναι μια οδηγία προς τον compiler *(λέει στον compiler πώς να χειρίζεται τον πηγαίο κώδικα).* Αυτή η οδηγία είναι πολύ σημαντική όταν χρησιμοποιούνται μεταβλητές. Γνωστοποιεί στον compiler ότι το εκτελέσιμο αρχείο θα φορτωθεί στην **ενεργό διεύθυνση** 100h *(256 bytes)*, οπότε ο compiler πρέπει να υπολογίσει την άμεση διεύθυνση για όλες τις μεταβλητές όταν αντικαθιστά τα ονόματα των μεταβλητών με τις ενεργές διευθύνσεις τους. Οι οδηγίες δεν μετατρέπονται ποτέ σε αληθινό κώδικα μηχανής.

Γιατί το εκτελέσιμο αρχείο φορτώνεται στην **ενεργό διεύθυνση 100h**; Το λειτουργικό σύστημα κρατάει κάποια δεδομένα για το πρόγραμμα στα πρώτα 256 bytes του **CS** *(κώδικας τμήματος),* όπως τις παραμέτρους της γραμμής εντολών κτλ. Ενώ αυτό ισχύει μόνο για τα αρχεία **COM**, τα αρχεία **EXE** φορτώνονται στην διεύθυνση **0000**, και γενικά χρησιμοποιούν το ειδικό τμήμα για τις μεταβλητές. Μπορεί να μιλήσουμε για **EXE** αρχεία αργότερα.

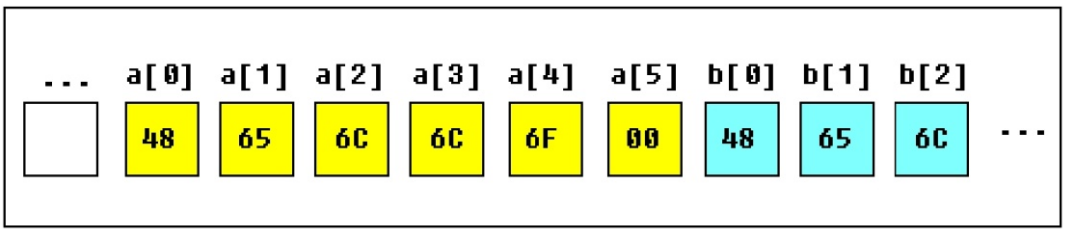
3.1 Πίνακες

Οι πίνακες μπορούν να θεωρηθούν σαν αλυσίδες μεταβλητών. Μια συμβολοσειρά κειμένου είναι ένα παράδειγμα ενός byte πίνακα, κάθε χαρακτήρας έχει μια ASCII τιμή (0..255).

Παραδείγματα αρχικοποίησης πινάκων:

```
a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h
b DB 'Hello', 0
```

Το b είναι ακριβές αντίγραφο του a πίνακα, όταν ο compiler βλέπει σε εισαγωγικά μια συμβολοσειρά, αυτόματα την μετατρέπει σε ένα σύνολο από bytes. Το παρακάτω διάγραμμα δείχνει ένα μέρος της μνήμης που δηλώνονται οι πίνακες:



Μπορείτε να έχετε πρόσβαση στην τιμή κάθε στοιχείου του πίνακα χρησιμοποιώντας αγκύλες, για παράδειγμα:

```
MOV AL, a[3]
```

Μπορείτε επίσης να χρησιμοποιήσετε οποιοδήποτε δείκτη καταχωρητών **BX, SI, DI, BP,** για παράδειγμα:

```
MOV SI, 3

MOV AL, a[SI]
```

Αν χρειάζεται να δηλώσετε έναν μεγάλο πίνακα μπορείτε να χρησιμοποιήσετε την οδηγία **DUP**.

Η σύνταξη της **DUP** είναι:

αριθμός DUP (τιμή/ες)

Αριθμός - αριθμός που θα φτιαχτούν τα διπλότυπα *(οποιαδήποτε σταθερά τιμή).*

Τιμή – έκφραση που ο DUP θα κάνει διπλότυπα.

Παράδειγμα:

```
c DB 5 DUP(9)
```

εναλλακτικός τρόπος δήλωσης:

```
c DB 9, 9, 9, 9, 9
```

Ακόμα ένα παράδειγμα:

```
d DB 5 DUP(1, 2)
```

εναλλακτικός τρόπος δήλωσης:

```
d DB 1, 2, 1, 2, 1, 2, 1, 2, 1, 2
```

Φυσικά, μπορείτε να χρησιμοποιήσετε **DW** αντί για **DB**, αν χρειάζεστε τιμές μεγαλύτερες από 255, ή τιμές μικρότερες από -128. Το **DW** δεν μπορεί να χρησιμοποιηθεί για δήλωση συμβολοσειρών.

Η έκταση **DUP** δεν θα πρέπει να είναι πάνω από 1020 χαρακτήρες! *(η έκταση του προηγούμενου παραδείγματος είναι 13 χαρακτήρες)*, αν χρειάζεται να δηλώσετε τεράστιους πίνακες χωρίστε την δήλωση σε δύο γραμμές *(θα έχετε έναν τεράστιο πίνακα στην μνήμη).*

3.2 Διεύθυνση Μεταβλητής

Υπάρχει η οδηγία **LEA** *(Load Effective Address)* και εναλλακτικά η οδηγία **OFFSET**. Τόσο η **OFFSET** όσο και η **LEA** μπορούν να χρησιμοποιηθούν για την απόκτηση της ενεργής διεύθυνσης της μεταβλητής. Η **LEA** είναι πιο ισχυρή γιατί σας επιτρέπει να πάρετε την τιμή της διεύθυνσης του δείκτη της μεταβλητής. Η απόκτηση της διεύθυνσης μιας μεταβλητής μπορεί να είναι πολύ χρήσιμη σε κάποιες περιπτώσεις, για παράδειγμα όταν χρειάζεται να περάσετε παραμέτρους σε μια συνάρτηση.

Υπενθύμιση:

Για να μπορέσει ο compiler να αναγνωρίσει τον τύπο δεδομένων πρέπει να χρησιμοποιηθούν τα παρακάτω προθέματα:

BYTE PTR – για byte.

WORD PTR – για λέξη *(δύο bytes).*

Για παράδειγμα:

```
BYTE PTR [BX]    ; πρόσβαση σε byte.

ή

WORD PTR [BX]    ; πρόσβαση σε λέξη.
```

Επίσης υποστηρίζονται από το *MicroAsm* μικρότερα προθέματα:

b. – για **BYTE PTR**

w. – για **WORD PTR**

Μερικές φορές ο compiler υπολογίζει τον τύπο δεδομένων αυτόματα, ωστόσο δεν θα πρέπει να βασίζεστε σε αυτό ειδικά όταν ένας από τους τελεστές έχει άμεση τιμή.

Πρώτο παράδειγμα:

```
ORG 100h

MOV   AL, VAR1    ; έλεγχος VAR1 μεταφέροντας την στο AL.

LEA   BX, VAR1    ; βάζει την διεύθυνση του VAR1 στο  BX.

MOV   BYTE PTR [BX], 44h ; τροποποιεί το περιεχόμενο του VAR1.

MOV   AL, VAR1    ; έλεγχος VAR1 μεταφέροντας την στο AL.

RET

VAR1  DB 22h

END
```

Ακόμα ένα παράδειγμα που χρησιμοποιείται η **OFFSET** αντί της **LEA**:

```
ORG 100h

MOV   AL, VAR1    ; έλεγχος VAR1 μεταφέροντας την στο AL.

MOV   BX, OFFSET VAR1  ;διεύθυνση της VAR1 στο BX.

MOV   BYTE PTR [BX], 44h ;τροποποιεί το περιεχόμενο VAR1.

MOV   AL, VAR1    ; έλεγχος VAR1 μεταφέροντας την στο AL.

RET

VAR1  DB 22h

END
```

Και τα δύο παραδείγματα έχουν την ίδια λειτουργικότητα. Αυτές οι σειρές:

```
LEA BX, VAR1

MOV BX, OFFSET VAR1
```

Έχουν στο compiler τον ίδιο κώδικα μηχανής:

```
MOV BX, num
```

Ο num έχει τιμή 16 bit της μεταβλητής offset.

Παρακαλώ προσέξτε ότι μόνο οι καταχωρητές μπορούν να χρησιμοποιηθούν μέσα σε αγκύλες *(σαν δείκτες μνήμης)*: **BX, SI, DI, BP!** *(δείτε προηγούμενα μέρη της διαδικασίας εκμάθησης).*

3.3 Σταθερές

Οι σταθερές είναι σαν τις μεταβλητές, αλλά δημιουργούνται μονάχα μέχρι το πρόγραμμα να κάνει compile *(assembled).* Μετά τον ορισμό μιας σταθεράς η τιμή της δεν μπορεί να αλλάξει. Για να ορίσουμε την σταθερά οδηγία EQU χρησιμοποιείται:

```
Όνομα EQU < έκφραση >
```

Παράδειγμα:

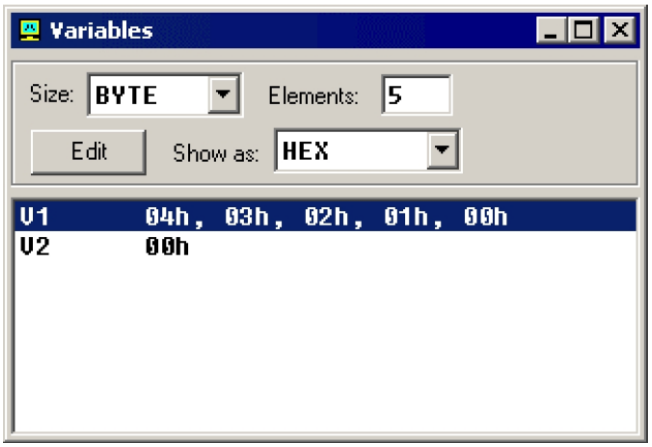
```
k EQU 5

MOV AX, k
```

```
MOV AX, 5
```

Διαβάστε την παρακάτω ενότητα μόνο αν χρησιμοποιείτε τον επεξεργαστή προσομοίωσης **emu8086:**

Μπορείτε να δείτε τις μεταβλητές ενώ το πρόγραμμα εκτελείται επιλέγοντας **"Variables"** από το **"View"** στο μενού του emulator.



Για να δείτε τους πίνακες πρέπει νε κάνετε κλικ σε μια μεταβλητή και να θέσετε στο **elements** τον κατάλληλο αριθμό δεδομένων στον πίνακα. Στην γλώσσα Assembly δεν υπάρχουν άμεσοι τύποι δεδομένων, οπότε κάθε μεταβλητή μπορεί να αναπαρασταθεί σαν πίνακας.

Η μεταβλητή μπορεί να είναι σε οποιοδήποτε αριθμητικό σύστημα.

- **HEX** – δεκαεξαδικό *(βάση το 16).*
- **BIN** – δυαδικό *(βάση το 2).*
- **OCT** – οκταδικό *(βάση το 8).*
- **SIGNED** – προσημασμένος δεκαδικός *(βάση το 10).*
- **UNSIGNED** – μη προσημασμένος δεκαδικός *(βάση το 10).*
- **CHAR** – ASCII χαρακτήρας
  (υπάρχουν 256 σύμβολα, κάποια σύμβολα είναι αόρατα).

Μπορείτε να επεξεργαστείτε την τιμή μιας μεταβλητής όταν ακόμα το πρόγραμμα σας τρέχει, απλώς πατήστε διπλό κλικ, ή επιλέξτε την και στην συνέχεια πατήστε το πλήκτρο **Edit**.

Μπορείτε να εισάγετε αριθμούς σε οποιοδήποτε αριθμητικό σύστημα, οι δεκαεξαδικοί αριθμοί πρέπει να έχουν κατάληξη **"h"**, οι δυαδικοί **"b"** και οι οκταδικοί **"o"**, ενώ οι δεκαδικοί δεν χρειάζονται κάποια κατάληξη. Οι συμβολοσειρές μπορούν να εισαχθούν κατά τον παρακάτω τρόπο: **'hello world', 0** *(η συμβολοσειρά αυτή είναι μηδενικού τερματισμού).*

Οι πίνακες μπορούν να εισαχθούν κατά τον παρακάτω τρόπο:

> **1, 2, 3, 4, 5** *(ο πίνακας μπορεί να αποτελείται από byte ή words, εξαρτάται από το αν είναι επιλεγμένο το **BYTE** ή το **WORD**).*

Οι εκφράσεις μετατρέπονται αυτόματα, για παράδειγμα: όταν εισαχθεί η παρακάτω πρόταση:

> **5 + 2** θα μετατραπεί σε **7** κτλ...

4. Διακοπές

Οι διακοπές *(interrupts)* μπορούν να θεωρηθούν σαν μια σειρά λειτουργιών που διευκολύνουν τον προγραμματισμό. Έτσι αντί για τη συγγραφή κώδικα για την εμφάνιση ενός χαρακτήρα, να καλείται η διακοπή και αυτή να τα κάνει όλα. Υπάρχουν επίσης κάποιες λειτουργίες διακοπών που σχετίζονται με τη μονάδα δίσκου *(disc drive)* και άλλο υλικό *(hardware)*. Αυτές οι λειτουργίες διακοπών ονομάζονται **διακοπές λογισμικού**.

Οι διακοπές προκαλούνται από διαφορετικό hardware, αυτές ονομάζονται **διακοπές hardware**. Αυτή τη στιγμή ενδιαφερόμαστε μόνο για τις **διακοπές λογισμικού**.

Για να υπάρξει μια **διακοπή λογισμικού** χρειάζεται μια οδηγία **INT**, η σύνταξη της είναι απλή:

```
INT τιμή
```

Όπου **τιμή** μπορεί να είναι οποιοσδήποτε αριθμός μεταξύ του 0 ως 255 *(ή 0 ως 0FFh)*, γενικά χρησιμοποιούνται δεκαεξαδικοί αριθμοί.

Μπορεί να νομίζετε ότι υπάρχουν μονάχα 256 λειτουργίες, αλλά αυτό δεν αληθεύει. Κάθε διακοπή μπορεί να έχει επίσης υπο-λειτουργίες.

Για να ορίσουμε μία υπο-λειτουργία πρέπει πρώτα να τεθεί ο καταχωρητής **AH** σε μία τιμή πριν κληθεί η διακοπή. Κάθε διακοπή μπορεί να έχει πάνω από 256 υπολειτουργίες *(οπότε προκύπτουν 256 * 256 = 65536 λειτουργίες)*. Γενικά ο καταχωρητής **AH** χρησιμοποιείται, αλλά μερικές φορές μπορεί άλλοι καταχωρητές να μπουν επίσης σε χρήση. Γενικά οι άλλοι καταχωρητές χρησιμοποιούνται για πέρασμα παραμέτρων και δεδομένων στις υπολειτουργίες. Το επόμενο παράδειγμα χρησιμοποιεί το **INT 10h** με υπολειτουργία το **0Eh** για εμφάνιση μηνύματος `"Hello!"`. Αυτή η λειτουργία εμφανίζει ένα χαρακτήρα στην οθόνη και μετακινεί τον κέρσορα και την οθόνη όπου είναι αναγκαίο.

```
#MAKE_COM# ; εντολή σε compiler για δημιουργία COM file.
ORG 100h
Η υπο-λειτουργία που χρησιμοποιείται δεν τροποποιεί στην επιστροφή
τον καταχωρητή AH, μπορεί να τεθεί μόνο μία φορά.
MOV  AH, 0Eh    ;επιλογή υπολειτουργίας.
← INT 10h/0Eh το υποπρόγραμμα λαμβάνει τον ASCII χαρακτήρα που θα
εμφανιστεί
← στον καταχωρητή
← AL.
MOV  AL, 'H'    ; ASCII κώδικας: 72
INT  10h        ;εμφάνισε το!
MOV  AL, 'e'    ; ASCII κώδικας:101
INT  10h        ;εμφάνισε το!
```

```
MOV  AL, 'l'    ; ASCII κώδικας: 108
INT  10h        ; εμφάνισε το!
MOV  AL, 'l'    ; ASCII κώδικας: 108
INT  10h        ; εμφάνισε το!
MOV  AL, 'o'    ; ASCII κώδικας: 111
INT  10h        ; εμφάνισε το!
MOV  AL, '!'    ; ASCII κώδικας: 33
INT  10h        ; εμφάνισε το!
RET             ;επιστροφή στο λειτουργικό σύστημα.
```

Επιλέξτε αντιγραφή & επικόλληση του παραπάνω προγράμματος στον επεξεργαστή πηγαίου κώδικα, κα επιλέξτε compile.

Τρέξτε το!

Δείτε το **list of basic interrupts** για πληροφορίες σχετικά με τις διακοπές.

5. Βιβλιοθήκη κοινών συναρτήσεων – emu8086.inc

Για να γίνει ευκολότερος ο προγραμματισμός υπάρχουν κάποιες κοινές συναρτήσεις που μπορούν να συμπεριληφθούν σε ένα πρόγραμμα. Για να κάνετε το πρόγραμμα σας να χρησιμοποιεί συναρτήσεις που είναι ορισμένες σε ένα άλλο αρχείο πρέπει να χρησιμοποιήσετε την οδηγία **INCLUDE** ακολουθούμενη από το όνομα του αρχείου. Ο compiler ψάχνει αυτόματα για το αρχείο στον ίδιο φάκελο που είναι αποθηκευμένο το πηγαίο αρχείο, και αν δεν μπορεί να βρεθεί εκεί ψάχνει και στο φάκελο **Inc**.

Μέχρι στιγμής μπορεί να μην είστε σε θέση να αντιληφθείτε πλήρως το περιεχόμενο του **emu8086.inc** *(βρίσκεται στο φάκελο Inc)*, αλλά δεν πειράζει, αφού το μόνο που χρειάζεται είναι να καταλάβετε τι μπορεί να κάνει.

Για να χρησιμοποιείται οποιαδήποτε συνάρτηση στο **emu8086.inc** θα πρέπει στην αρχή του πηγαίου αρχείου σας να έχετε την επόμενη γραμμή:

```
include 'emu8086.inc'
```

Το **emu8086.inc** ορίζεται με τις παρακάτω μακροεντολές *(macros):*

- **PUTC char** – μακροεντολή με 1 παράμετρο, τυπώνει έναν ASCII χαρακτήρα στην θέση που βρίσκεται ο κέρσορας.
- **GOTOXY col, row** – μακροεντολή με 2 παραμέτρους, θέτει την θέση του κέρσορα.
- **PRINT string** – μακροεντολή με 1 παράμετρο, εμφανίζει μια συμβολοσειρά.
- **PRINTN string** – μακροεντολή με 1 παράμετρο, εμφανίζει μια συμβολοσειρά. Είναι το ίδιο με την PRINT αλλά προσθέτει αυτόματα στο τέλος της συμβολοσειράς “φορτίο επιστροφής”.
- **CURSOROFF** – απενεργοποιεί τον κέρσορα κειμένου.
- **CURSORON** – ενεργοποιεί τον κέρσορα κειμένου.

Για να χρησιμοποιήσετε οποιαδήποτε από τις παραπάνω μακροεντολές απλός πληκτρολογήστε το όνομα κάπου στον κώδικα σας και αν χρειάζεται θέστε τις παραμέτρους:

```
include emu8086.inc

ORG 100h
PRINT 'Hello World!'
GOTOXY 10, 5

PUTC 65   ; 65 -είναι ASCII κώδικας για το 'A'
PUTC 'B'
RET       ;επιστροφή στο λειτουργικό σύστημα.
END       ;οδηγία να σταματήσει ο compiler.
```

Όταν ο compiler επεξεργάζεται τον πηγαίο κώδικα ψάχνει στο φάκελο **emu8086.inc** δηλώσεις μακροεντολών και αλλάζει τα ονόματα τους με αληθινό κώδικα. Γενικά, οι μακροεντολές είναι σχετικά μικρά κομμάτια κώδικα, η συχνή όμως χρήση τους μπορεί να κάνει το εκτελέσιμο αρχείο πολύ μεγάλο *(οι διαδικασίες είναι καλύτερες για την βελτιστοποίηση του ελέγχου).*

Το emu8086.inc ορίζει τις παρακάτω διαδικασίες:

- **PRINT_STRING** – διαδικασία εμφάνισης μιας συμβολοσειράς που τερματίζει σε μηδέν στην θέση που βρίσκεται ο κέρσορας, λαμβάνει την διεύθυνση της συμβολοσειράς στον καταχωρητή **DS:SI**. Για να το χρησιμοποιήσετε δηλώστε: **DEFINE_PRINT_STRING** πριν την οδηγία **END**.

- **PTHIS** - διαδικασία εμφάνισης μιας συμβολοσειράς που τερματίζει σε μηδέν στην θέση που βρίσκεται ο κέρσορας, *(σαν το PRINT_STRING)*, αλλά λαμβάνει την διεύθυνση της συμβολοσειράς από την στοίβα. Η συμβολοσειρά μηδενικού τερματισμού θα πρέπει να ορίζεται αμέσως μετά την οδηγία CALL. Για παράδειγμα:

```
CALL PTHIS
db 'Hello World!', 0
```

  Για να το χρησιμοποιήσετε δηλώστε: **DEFINE_PTHIS** πριν την οδηγία **END**.

- **GET_STRING** - διαδικασία για να ληφθεί μιας συμβολοσειρά μηδενικού τερματισμού από ένα χρήστη. Η συμβολοσειρά που λαμβάνεται γράφεται στην προσωρινή μνήμη στο **DS:DI**, το μέγεθος της προσωρινής μνήμης πρέπει να είναι στο **DX**. Η διαδικασία σταματάει μόλις πατηθεί το 'Enter'. Για να τη χρησιμοποιήσετε δηλώστε: **DEFINE_GET_STRING** πριν την οδηγία **END**.

- **CLEAR_SCREEN** – διαδικασία καθαρισμού οθόνης, *(γίνεται με μετακίνηση ολόκληρου του παραθύρου οθόνης)*, και θέτοντας τον κέρσορα στην κορυφή της. Για να την χρησιμοποιήσετε δηλώστε: **DEFINE_CLEAR_SCREEN** πριν την οδηγία **END**.

- **SCAN_NUM** – διαδικασία που δέχεται από το πληκτρολόγιο πολλά ψηφία ΠΡΟΣΗΜΑΣΜΕΝΩΝ αριθμών και αποθηκεύει τα αποτελέσματα στον καταχωρητή **CX**.
  Για να την χρησιμοποιήσετε δηλώστε: **DEFINE_SCAN_NUM** πριν την οδηγία **END**.

- **PRINT_NUM** – διαδικασία εμφάνισης προσημασμένου αριθμού στον καταχωρητή **AX**. Για να την χρησιμοποιήσετε δηλώστε:
  DEFINE_PRINT_NUM και **DEFINE_PRINT_NUM_UNS** πριν την οδηγία **END**.

- **PRINT_NUM_UNS** – διαδικασία εμφάνισης μη προσημασμένου αριθμού στον καταχωρητή **AX**.
  Για να την χρησιμοποιήσετε δηλώστε: **DEFINE_PRINT_NUM_UNS** πριν την οδηγία **END**.

Για να χρησιμοποιήσετε οποιαδήποτε από τις παραπάνω διαδικασίες πρέπει πρώτα να το δηλώσετε στο τέλος του αρχείου *(αλλά πριν το END)*, και μετά να χρησιμοποιήστε την οδηγία CALL ακολουθούμενη από το όνομα της διαδικασίας.

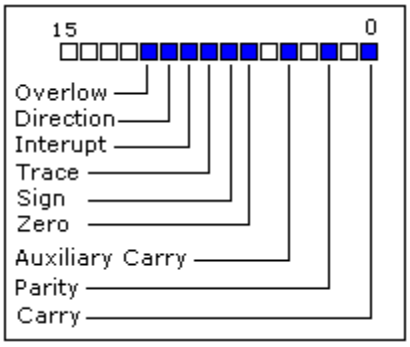
Παράδειγμα:

```
include 'emu8086.inc'
ORG  100h
LEA  SI, msg1  ;ζητάμε αριθμό
CALL print_string
CALL scan_num  ;τοποθετούμε στο CX.
MOV  AX, CX    ;μεταφέρουμε τον ;αριθμό στο AX.
; τυπώνουμε την ακόλουθη συμβολοσειρά
string: CALL pthis
DB   13, 10, 'You have entered: ', 0
CALL print_num ;εμφανίζουμε τον αριθμό από το AX.
RET            ;επιστροφή στο λειτουρικό σύστημα.
msg1 DB   'Enter the number: ', 0
DEFINE_SCAN_NUM
DEFINE_PRINT_STRING
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS ;δήλωση για print_num.
DEFINE_PTHIS
END        ;οδηγία για να σταματήσει ο compiler.
```

Ο compiler πρώτα επεξεργάζεται τις δηλώσεις *(αυτές είναι οι συνήθεις μακροεντολές που έχουν επεκταθεί στις διαδικασίες)*. Όταν ο compiler παίρνει την οδηγία CALL αντικαθιστά το όνομα της διαδικασίας με την διεύθυνση του κώδικα που δηλώνεται η διαδικασία. Όταν εκτελείται η CALL ο έλεγχος μεταφέρεται στην διαδικασία. Αυτό είναι αρκετά χρήσιμο, γιατί ακόμα και αν κληθεί η ίδια διαδικασία 100 φορές μέσα στον κώδικα, θα είναι και πάλι σχετικά μικρό το μέγεθος του εκτελέσιμου αρχείου. Δείχνει κάπως περίπλοκο, σωστά; Ωστόσο, δεν υπάρχει πρόβλημα καθώς με το πέρασμα του χρόνου θα μάθετε περισσότερα, επί του παρόντος χρειάζεται να καταλαβαίνεται τις βασικές αρχές.

6. Αριθμητικές και Λογικές Εντολές

Οι περισσότερες αριθμητικές και λογικές εντολές επηρεάζουν την κατάσταση των καταχωρητών *(ή **σημαιών**)* του επεξεργαστή.



Όπως βλέπετε υπάρχουν 16bit σε αυτόν τον καταχωρητή. Το κάθε bit ονομάζεται σημαία και μπορεί να πάρει τις τιμές 0 ή 1.

- **Σημαία Κρατουμένου (CF)** - αυτή η σημαία παίρνει την τιμή **1**, όταν υπάρχει απροσήμαστη υπερχείλιση. Για παράδειγμα, όταν προσθέτουμε τα bytes 255+1 *(το αποτέλεσμα δεν είναι μεταξύ της εμβέλειας 0...255)*. Όταν δεν υπάρχει υπερχείλιση η σημαία αυτή παίρνει την τιμή **0**.

- **Σημαία Μηδενικού (ZF)** - παίρνει την τιμή **1** όταν το αποτέλεσμα είναι **0**. Για κανένα μηδενικό αποτέλεσμα αυτή η σημαία παίρνει την τιμή **0**.

- **Σημαία Προσήμου (SF)** - παίρνει την τιμή **1** όταν το αποτέλεσμα είναι αρνητικό. Όταν το αποτέλεσμα είναι θετικό, τότε παίρνει την τιμή **0**. Στην πραγματικότητα, αυτή η σημαία παίρνει την τιμή του πιο σημαντικού ψηφίου *(MSB).*

- **Σημαία Υπερχείλισης (OF)** - παίρνει την τιμή **1** όταν υπάρχει προσημασμένο αποτέλεσμα. Για παράδειγμα, όταν προσθέτουμε τα bytes 100+50 *(η εμβέλεια του αποτελέσματος δεν είναι μεταξύ του -128...127)*

- **Σημαία Κρατουμένου (PF)** - αυτή η σημαία παίρνει την τιμή **1** όταν υπάρχει άρτια ισοτιμία στο αποτέλεσμα και **0** όταν υπάρχει περιττή ισοτιμία. Ακόμα και όταν το αποτέλεσμα είναι λέξη, μόνο τα 8 πιο σημαντικά bit αναλύονται.

- **Σημαία Βοηθητική (AF)** - παίρνει την τιμή **1** όταν υπάρχει απροσήμαστο αποτέλεσμα μικρού μεγέθους *(4 bits).*

- **Σημαία Ενεργοποίησης Διακοπών (IF)** - όταν αυτή η σημαία έχει την τιμή **1** τότε ο επεξεργαστής ανταποκρίνεται σε διακοπές από εξωτερικές συσκευές.

- **Σημαία Διαδρομής (DF)** - αυτή η σημαία χρησιμοποιείται από μερικές εντολές για τη διεργασία δεδομένων αλυσίδας. Όταν η σημαία έχει την τιμή **0** - η διεργασία έχει γίνει προς τα εμπρός, όταν η σημαία έχει την τιμή **1** η διεργασία έχει γίνει προς τα πίσω.

Υπάρχουν 3 ομάδες από εντολές.

- Πρώτη Ομάδα: **ADD, SUB,CMP, AND, TEST, OR, XOR**

Υποστηρίζονται οι παρακάτω τύποι τελεστέων:

REG (καταχωρητής), **memory** (θέση μνήμης)
memory, REG
REG, REG
memory, immediate (σταθερή τιμή)
REG, immediate

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], μεταβλητή, και άλλα…

immediate: 5, -24, 3Fh, 10001101b, και άλλα...

Μετά τον χειρισμό των τελεστέων, το αποτέλεσμα αποθηκεύεται πάντα στον πρώτο τελεστέο. Οι εντολές **CMP** και **TEST** επηρεάζουν μόνο σημαίες και δεν αποθηκεύουν κάποιο αποτέλεσμα.

Αυτές οι εντολές επηρεάζουν μόνο τις εξής σημαίες:

CF, ZF, SF, OF, PF, AF.

- **ADD** - πρόσθεσε τον δεύτερο τελεστέο στον πρώτο.
- **SUB** - αφαίρεσε τον δεύτερο τελεστέο από τον πρώτο.
- **CMP** - αφαίρεσε τον δεύτερο τελεστέο από τον πρώτο *(μόνο για σημαίες)*.
- **AND** - λογικό ΚΑΙ μεταξύ όλων των ψηφίων των 2 τελεστέων.

  Εφαρμόζονται οι παρακάτω κανόνες :
  1 AND 1 = 1
  1 AND 0 = 0
  0 AND 1 = 0
  0 AND 0 = 0

  Όπως βλέπετε παίρνουμε το **1**, όταν και τα 2 bit είναι **1**.

- **TEST** - το ίδιο με την AND μόνο που είναι για σημαίες.

- **OR** - λογικό Ή μεταξύ όλων των ψηφίων των 2 τελεστέων.

  Εφαρμόζονται οι παρακάτω κανόνες :
  1 OR 1 = 1
  1 OR 0 = 1
  0 OR 1 = 1
  0 OR 0 = 0

  Όπως βλέπετε παίρνουμε το **1**, κάθε φορά που υπάρχει τουλάχιστον ένας **1** *(άσσος)* σε κάποιο από τα bit

- **XOR** - λογικό XOR *(αποκλειστικό ή)* μεταξύ όλων των ψηφίων των 2 τελεστέων.

  Εφαρμόζονται οι παρακάτω κανόνες :
  1 XOR 1 = 0
  1 XOR 0 = 1
  0 XOR 1 = 1
  0 XOR 0 = 0

  Όπως βλέπετε παίρνουμε το **1** κάθε φορά που τα bit είναι διαφορετικά μεταξύ τους.

➢ Δεύτερη Ομάδα: **MUL, IMUL, DIV, IDIV**

Υποστηρίζονται οι παρακάτω τύποι τελεστέων:

```
REG
memory
```

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], μεταβλητή, και άλλα...

Οι εντολές **MUL** και **IMUL** επηρεάζουν μόνο τις παρακάτω σημαίες:

CF, OF

Όταν το αποτέλεσμα είναι μεγαλύτερο από το μέγεθος του τελεστέου αυτές οι σημαίες παίρνουν την τιμή **1**. Όταν το αποτέλεσμα είναι μικρότερο ή ίσο, τότε οι σημαίες παίρνουν την τιμή **0**.

Για τις εντολές **DIV** και **IDIV** δεν έχουν προσδιοριστεί σημαίες.

- **MUL** - απροσήμαστος πολλαπλασιασμός:

  όταν ο τελεστέος είναι ένα **byte**:
  AX = AL * τελεστέος.

  όταν ο τελεστέος είναι **word** *(λέξη)*:
  (DX AX) = AX * τελεστέος.

- **IMUL** - προσημασμένος πολλαπλασιασμός:

  όταν ο τελεστέος είναι ένα **byte**:
  AX = AL * τελεστέος.

  όταν ο τελεστέος είναι **word**:
  (DX AX) = AX * τελεστέος.

- **DIV** - απροσήμαστη διαίρεση:

  όταν ο τελεστέος είναι ένα **byte**:
  AL = AX / τελεστέος
  AH = υπόλοιπο.

  όταν ο τελεστέος είναι **word**:
  AX = (DX AX) / τελεστέος
  DX = υπόλοιπο.

- **IDIV** - προσημασμένη διαίρεση:

  όταν ο τελεστέος είναι ένα **byte**:
  AL = AX / τελεστέος
  AH = υπόλοιπο.

  όταν ο τελεστέος είναι **word**:
  AX = (DX AX) / τελεστέος
  DX = υπόλοιπο.

- Τρίτη Ομάδα: **INC, DEC, NOT, NEG**

Υποστηρίζονται οι παρακάτω τύποι τελεστέων:

```
REG
memory
```

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], μεταβλητή, και άλλα...

Οι εντολές **INC**, **DEC** επηρεάζουν μόνο τις παρακάτω σημαίες:

ZF, SF, OF, PF, AF.

Η εντολή **NOT** δεν επηρεάζει καμία σημαία!

Η εντολή **NEG** επηρεάζει μόνο τις παρακάτω σημαίες:

CF, ZF, SF, OF, PF, AF.

- **NOT** - αντιστροφή κάθε bit του τελεστέου.

- **NEG** - μετατροπή τελεστέου σε αρνητικό *(συμπλήρωμα ως προς 2)*. Στην πραγματικότητα αντιστρέφει το κάθε bit του τελεστέου και στη συνέχεια του προσθέτει το 1 σε κάθε ένα από αυτά. Για παράδειγμα, το 5 θα γίνει -5 και το -2 θα γίνει 2.

7. Έλεγχος Ροής Προγράμματος

Ο έλεγχος ροής προγράμματος είναι πολύ σημαντικός. Είναι το σημείο που το πρόγραμμά σας, μπορεί να πάρει αποφάσεις σύμφωνα με συγκεκριμένες καταστάσεις

- **Διακλαδώσεις χωρίς συνθήκη.**

Η βασική εντολή που μεταφέρει τον έλεγχο σε ένα άλλο σημείο του προγράμματος είναι η **JMP**. Η βασική σύνταξη της **JMP** είναι:

```
JMP ετικέτα
```

Για να δηλώσουμε μια ετικέτα στο πρόγραμμα, απλά γράφουμε το όνομά της και προσθέτουμε στο τέλος τον χαρακτήρα **":"**. Η ετικέτα μπορεί να είναι οποιοσδήποτε συνδυασμός χαρακτήρων, αλλά δεν μπορεί να ξεκινάει με αριθμό. Για παράδειγμα, δείτε 3 αποδεκτούς τρόπους δήλωσης ετικέτας:

```
label1:
label2:
a:
```

Η ετικέτα δεν μπορεί να δηλωθεί σε διαφορετική γραμμή ή πριν από κάποια άλλη εντολή. Για παράδειγμα :

```
x1:
MOV AX, 1
x2: MOV AX, 2
```

Ορίστε ένα παράδειγμα με την εντολή **JMP:**

```
ORG 100h

MOV  AX, 5       ;βάλε στον AX το 5.
MOV  BX, 2       ; βάλε στον BX το 2.

JMP  calc        ; πήγαινε στην ετικέτα 'calc'.

back:  JMP stop  ; πήγαινε στην ετικέτα 'stop'.

calc:
ADD  AX, BX      ; πρόσθεσε το BX στον AX.
JMP  back        ; πήγαινε στην ετικέτα 'back'.

stop:

RET  ; επιστροφή στο λειτουργικό σύστημα.

END  ; οδηγία προς τον compiler να σταματήσει.
```

Φυσικά, υπάρχει πιο εύκολος τρόπος να υπολογίσουμε το άθροισμα 2 αριθμών, αλλά είναι ένα καλό παράδειγμα της εντολής **JMP**.

Όπως βλέπετε από το παράδειγμα, η **JMP** μπορεί να μεταφέρει τον έλεγχο τόσο προς τα εμπρός, όσο και προς τα πίσω. Μπορεί να μεταπηδήσει οπουδήποτε μέσα στο τρέχων τμήμα κώδικα *(65, 535 bytes)*.

- **Διακλαδώσεις με συνθήκη.**

Σε αντίθεση με την εντολή **JMP** που κάνει διακλαδώσεις χωρίς συνθήκη, υπάρχουν εντολές που κάνουν διακλαδώσεις με συνθήκη *(μεταπήδηση μόνο όταν ισχύει μια συνθήκη, κατάσταση)*. Αυτές οι εντολές χωρίζονται σε 3 ομάδες. Η πρώτη ομάδα εξετάζει μόνο τις απλές σημαίες. Η δεύτερη ομάδα, συγκρίνει προσημασμένους αριθμούς και η τρίτη ομάδα, συγκρίνει απροσήμαστους αριθμούς.

Οδηγίες Μεταπήδησης που Εξετάζουν μόνο Απλές Σημαίες.

Εντολή	Περιγραφή	Κατάσταση	Αντίθετη εντολή
JE, JZ	Διακλάδωση αν είναι ίσο.	ZF = 1	JNZ, JNE
JC, JB, JNAE	Διακλάδωση αν είναι μικρότερο μη προσημασμένο.	CF = 1	JNC, JNB, JAE
JS	Διακλάδωση εάν αρνητικό αποτέλεσμα.	SF = 1	JNS
JO	Διακλάδωση εάν υπάρχει υπέρβαση κρατουμένου.	OF = 1	JNO
JPE, JP	Διακλάδωση εάν υπάρχει ισοτιμία.	PF = 1	JPO
JNZ, JNE	Διακλάδωση εάν είναι άνισα.	ZF = 0	JZ, JE
JNC, JNB, JAE	Διακλάδωση εάν δεν είναι μικρότερο μη Προσημασμένο.	CF=0	JC, JB, JNAE
JNS	Διακλάδωση εάν θετικό αποτέλεσμα.	SF = 0	JS
JNO	Διακλάδωση εάν δεν υπάρχει υπέρβαση κρατουμένου.	OF = 0	JO
JPO, JNP	Διακλάδωση εάν δεν υπάρχει ισοτιμία.	PF = 0	JPE, JP

Όπως βλέπετε υπάρχουν εντολές που κάνουν το ίδιο πράγμα, που είναι σωστό, που έχουν ακριβώς τον ίδιο κώδικα μηχανής, οπότε είναι καλό να θυμάστε πως όταν μεταγλωττίζετε την εντολή **JE** - θα την πάρετε μεταφρασμένη ως: **JZ**.

Διαφορετικά ονόματα χρησιμοποιούνται ώστε τα προγράμματα να είναι πιο εύκολα, τόσο στην κατανόηση, όσο και στον κώδικα.

Εντολές μεταπήδησης για Προσημασμένους Αριθμούς.

Εντολή	Περιγραφή	Κατάσταση	Αντίθετη εντολή
JE, JZ	Διακλάδωση αν είναι ίσο (=) Διακλάδωση αν είναι 0.	ZF = 1	JNZ, JNE
JNZ, JNE	Διακλάδωση αν είναι άνισα (<>) Διακλάδωση αν δεν είναι 0.	ZF = 0	JE, JZ
JG, JNLE	Διακλάδωση αν είναι μεγαλύτερο προσημασμένο.	ZF = 0 and SF = OF	JNG, JLE
JL, JNGE	Διακλάδωση αν είναι μικρότερο προσημασμένο.	SF<>OF	JNL, JGE
JGE, JNL	Διακλάδωση αν δεν είναι μικρότερο προσημασμένο.	SF=OF	JNGE, JL
JLE, JNG	Διακλάδωση αν δεν είναι μεγαλύτερο προσημασμένο.	ZF = 1 and SF <> OF	JNLE, JG

<> - σημαίνει όχι ίσα *(άνισα)***.**

Εντολές μεταπήδησης για Απροσήμαστους Αριθμούς.

Εντολή	Περιγραφή	Κατάσταση	Αντίθετη εντολή
JE, JZ	Διακλάδωση αν είναι ίσο (=) Διακλάδωση αν είναι 0.	ZF = 1	JNE, JNZ
JNE, JNZ	Διακλάδωση αν είναι άνισα (<>) Διακλάδωση αν δεν είναι 0.	ZF = 0	JE, JZ
JA, JNBE	Διακλάδωση αν είναι μεγαλύτερο προσημασμένο. Διακλάδωση αν είναι μικρότερο προσημασμένο.	CF = 0 and ZF = 0	JNG, JLE
JB, JNAE, JC	Διακλάδωση αν έχει κρατούμενο.	CF = 1	JNL, JGE
JAE, JNB, JNC	Διακλάδωση αν δεν είναι μικρότερο προσημασμένο.	CF = 0	JNGE, JL
JBE, JNA	Διακλάδωση αν δεν είναι μεγαλύτερο προσημασμένο.	CF = 1 or ZF = 1	JNLE, JG

Γενικά, όταν απαιτείται σύγκριση μεταξύ αριθμών χρησιμοποιείται η εντολή **CMP** *(κάνει ακριβώς το ίδιο πράγμα με την εντολή **SUB** – αφαίρεση – αλλά δεν κρατάει το αποτέλεσμα, απλά επηρεάζει τις σημαίες).*

Η λογική είναι πολύ απλή.

- ✓ Για παράδειγμα:
  Απαιτείται η σύγκριση μεταξύ των 5 και 2, 5-2=3.
  Το αποτέλεσμα δεν είναι μηδέν *(η σημαία μηδενικού παίρνει την τιμή 0)*.

- ✓ Ένα άλλο παράδειγμα:
  Απαιτείται η σύγκριση μεταξύ των 7 και72, 7-7=0.
  Το αποτέλεσμα είναι μηδέν! *(η σημαία μηδενικού παίρνει την τιμή 1 και οι εντολές **JZ** ή **JE** δεν θα κάνουν την μεταπήδηση)*.

Ένα παράδειγμα με εντολές **CMP** και διακλαδώσεις με συνθήκη:

```
include emu8086.inc

ORG 100h

MOV   AL, 25      ; βάλε στον AL το 25.
MOV   BL, 10      ; βάλε στον BL το 10.

CMP   AL, BL      ; σύγκρινε τους AL - BL.

JE    equal       ; διακλάδωση αν AL = BL (ZF = 1).

PUTC  'N'         ; αν έρθει εδώ, τότε  AL <> BL,
JMP   stop        ; τύπωσε το 'N' και πήγαινε στην ετικέτα stop.

equal:      ; αν έρθει εδώ,
PUTC  'Y'   ; τότε  AL = BL, οπότε τύπωσε το 'Y'.

stop:

RET         ; θα φτάσει εδώ, ότι και αν συμβεί.

END
```

Δοκιμάστε το παραπάνω παράδειγμα με διαφορετικούς αριθμούς για τους καταχωρητές **AL** και **BL**, δείτε τις σημαίες, κάνοντας click στο πλήκτρο **[FLAGS]**. Χρησιμοποιείστε το **[Single Step]** και δείτε τι συμβαίνει. Μην ξεχάσετε να ξαναμεταγλωττίσετε, να επαναφορτώσετε μετά από κάθε αλλαγή *(χρησιμοποιείστε την συντόμευση **F5**)*.

Όλες οι διακλαδώσεις με συνθήκη έχουν ένα μεγάλο περιορισμό. Σε αντίθεση με την εντολή **CMP** μπορούν να μεταπηδήσουν **127** bytes προς τα εμπρός και **128** bytes προς τα πίσω *(σημειώστε ότι οι περισσότερες εντολές συμβολομεταφράζονται σε 3 ή περισσότερα bytes)*.

Μπορούμε εύκολα αποφύγουμε αυτούς τους περιορισμούς με ένα έξυπνο κόλπο:

- Πάρτε μια αντίθετη εντολή διακλάδωσης με συνθήκη από τον παραπάνω πίνακα και κάντε μεταπήδηση στην `ετικέτα_x`.
- Χρησιμοποιείστε την εντολή **JMP** για να μεταπηδήσετε σε συγκεκριμένη τοποθεσία.
- Ορίστε την `ετικέτα_x:` αμέσως μετά την εντολή **JMP**.
  `ετικέτα_x:` - μπορεί να έχει οποιοδήποτε έγκυρο όνομα.

Ορίστε ένα παράδειγμα:

```
include emu8086.inc

ORG 100h

MOV   AL, 25      ; βάλε στον AL το 25.
MOV   BL, 10      ; βάλε στον BL το 10.

CMP   AL, BL      ; σύγκρινε AL - BL.

JNE   not_equal   ; μεταπήδηση  if AL <> BL (ZF = 0).
JMP   equal
not_equal:

; ας υποθέσουμε ότι εδώ έχουμε
; ένα κομμάτι κώδικα που συμβολομεταφράζεται
; σε περισσότερα από 127 bytes...

PUTC  'N'   ; αν έρθει εδώ,  τότε  AL <> BL,
JMP   stop  ; οπότε τύπωσε το  'N', και πήγαινε στην ετικέτα stop.
equal:      ; αν έρθει εδώ,
PUTC  'Y'   ; τότε  AL = BL, οπότε τύπωσε το  'Y'.
stop:
RET         ; θα φτάσει εδώ, ότι και αν συμβεί.
END
```

Μια άλλη, ακόμα πιο σπάνια χρησιμοποιούμενη περίπτωση, προσφέρει σταθερή τιμή αντί μιας ετικέτας. Όταν η σταθερή τιμή ξεκινάει με τον χαρακτήρα "$", πραγματοποιείται όμοια μεταπήδηση, διαφορετικά ο μεταγλωττιστής μετράει τις εντολές που πραγματοποιούν απευθείας μεταπήδηση στην ενεργό διεύθυνση.

Για παράδειγμα:

```
ORG 100h
;διακλάδωση χωρίς συνθήκη προς τα εμπρός:
; προσπερνάμε για τα επόμενα 2 byte,
JMP $2
a DB 3      ; 1 byte.
b DB 4      ; 1 byte.

; JCC διακλάδωση προς τα πίσω κατά 7 bytes:
; (JMP παίρνει 2 byte από μόνο του)
MOV BL,9
DEC BL      ; 2 bytes.
CMP BL,     ; 3 bytes.
JNE $-7
RET
END
```

8. Διαδικασίες

Η διαδικασία είναι μέρος του κώδικα που μπορεί να καλεστεί από το πρόγραμμά σας, ώστε να πραγματοποιήσει ένα συγκεκριμένο έργο. Οι διαδικασίες καθιστούν το πρόγραμμα πιο δομημένο και πιο εύκολο στην κατανόηση. Γενικά, η διαδικασία επιστρέφει στο ίδιο σημείο από το οποίο καλέστηκε.

Η σύνταξη για τη δήλωση διαδικασίας είναι:

```
όνομα PROC
       ; εδώ μπαίνει ο κώδικας
       ; της διαδικασίας...
RET
όνομα ENDP
```

όνομα - είναι το όνομα της διαδικασίας. Το ίδιο όνομα πρέπει να είναι τόσο στην κορυφή, όσο και στο βάθος. Αυτό χρησιμοποιείται για τον έλεγχο σωστού κλεισίματος διαδικασιών.

Πιθανόν, ήδη να ξέρετε ότι η εντολή **RET** χρησιμοποιείται για την επιστροφή στο λειτουργικό σύστημα. Η ίδια εντολή χρησιμοποιείται για την επιστροφή από την διαδικασία.

(Στην πραγματικότητα, τα λειτουργικά συστήματα βλέπουν τα προγράμματα σαν διαφορετικές διαδικασίες).

Οι εντολές **PROC** και **ENDP** είναι οδηγίες προς τον μεταγλωττιστή, ώστε να μην συμβολομεταφραστούν σε οποιοδήποτε πραγματικό κώδικα μηχανής. Ο μεταγλωττιστής απλά θυμάται τη διεύθυνση της διαδικασίας.

Η εντολή **CALL** χρησιμοποιείται για να καλέσουμε την διαδικασία.

Ορίστε ένα παράδειγμα:

```
ORG 100h
CALL   m1
MOV   AX, 2
RET         ; επιστροφή στο λειτουργικό σύστημα.
m1    PROC
MOV   BX, 5
RET         ; επιστροφή στο σημείο από το οποίο καλέστηκε.
m1    ENDP
END
```

Το παραπάνω παράδειγμα καλεί την διαδικασία **m1**, εκτελεί την εντολή

MOV BX, 5 και επιστρέφει στην επόμενη εντολή μετά την **CALL** την **MOV AX, 2**.

Αυτοί είναι οι διάφοροι τρόποι να περάσουμε παραμέτρους στις διαδικασίες. Ο πιο εύκολος τρόπος, είναι μέσω καταχωρητών. Ορίστε και ένα παράδειγμα μιας διαδικασίας που δέχεται 2 παραμέτρους στους καταχωρητές **AL** και **BL** αντίστοιχα, τις πολλαπλασιάζει και επιστρέφει το αποτέλεσμα στον καταχωρητή **AX**:

```
ORG 100h

MOV   AL, 1
MOV   BL, 2

CALL   m2
CALL   m2
CALL   m2
CALL   m2

RET         ; επιστροφή στο λειτουργικό σύστημα.

m2    PROC
MUL   BL    ; AX = AL * BL.
RET         ; επιστροφή στο σημείο από το οποίο καλέστηκε.
m2    ENDP

END
```

Στο παραπάνω παράδειγμα η τιμή του καταχωρητή **AL** ενημερώνεται κάθε φορά που η διαδικασία καλείται. Ο **BL** καταχωρητής παραμένει ίδιος. Αυτός ο αλγόριθμος υπολογίζει το 2 σε δύναμη του 4, οπότε το τελικό αποτέλεσμα στον καταχωρητή **AX** είναι το 16 *(ή 10h).*

Ορίστε και ένα άλλο παράδειγμα , που χρησιμοποιεί μια διαδικασία για να εμφανίσει το μήνυμα "Hello World!":

```
ORG 100h
LEA   SI, msg     ; ο SI παίρνει την διεύθυνση της μεταβλητής msg.
CALL   print_me
RET               ; επιστροφή στο λειτουργικό σύστημα.
; ========================================================
; η διαδικασία αυτή εκτυπώνει ένα αλφαριθμητικό που έπρεπε να είναι
κενό .
; έπρεπε να έχει στο τέλος 0 ,
; η διεύθυνση του αλφαριθμητικού έπρεπε να είναι στον καταχωρητή SI:
print_me   PROC
next_char:
      CMP  b.[SI], 0   ; έλεγχος μέχρι να βρει 0.
      JE   stop
      MOV  AL, [SI]    ; πάρε τον επόμενο ASCII χαρακτήρα.
      MOV  AH, 0Eh     ; τηλέτυπο αριθμού λειτουργίας .
      INT  10h   ; χρήση διακοπής για να εκτυπώσει το
      ; περιεχόμενο του AL.
      ADD  SI, 1       ; αυξάνουμε κατά ένα ώστε να πάμε στο
      ; επόμενο στοιχείο του αλφαριθμητικού
      JMP  next_char   ; πήγαινε πίσω και τύπωσε τον επόμενο
      ; χαρακτήρα.
stop:
RET   ; επιστροφή στο σημείο από το οποίο καλέστηκε.
print_me   ENDP
; ========================================================
Msg   DB  'Hello World!', 0  ;τερματικός χαρακτήρας αλφ/κού.
END
```

"b." - το πρόθεμα αυτό πριν το [SI] σημαίνει ότι χρειάζεται να συγκρίνουμε 2bytes και όχι λέξεις . Όταν αντίθετα χρειάζεται να συγκρίνετε λέξεις προσθέτετε το πρόθεμα **"w."**. Όταν ένας από τους δύο τελεστέους είναι καταχωρητής, δεν χρειάζεται να βάλετε πρόθεμα γιατί ο μεταγλωττιστής γνωρίζει το μέγεθος του κάθε καταχωρητή.

9. Ο σωρός

Ο σωρός είναι μια περιοχή στη μνήμη που κρατάει προσωρινά τα δεδομένα. Ο σωρός χρησιμοποιείται από την εντολή **CALL** για να κρατήσει τη διεύθυνση επιστροφής για τη διαδικασία. Η εντολή **RET** παίρνει τη τιμή από το σωρό και επιστρέφει στην ενεργό διεύθυνση. Σχεδόν το ίδιο συμβαίνει όταν η εντολή **INT** καλεί μια διακοπή. Αποθηκεύει στο σωρό σε σημαία καταχωρητή το τμήμα κώδικα και την ενεργό διεύθυνση. Η εντολή **IRET** χρησιμοποιείται για να επιστρέψουμε από κλήση διακοπής.

Επίσης μπορούμε να χρησιμοποιήσουμε το σωρό για να κρατήσουμε οποιοδήποτε άλλο δεδομένο . Υπάρχουν δύο εντολές που δουλεύουν με τον σωρό:

- ✓ **PUSH** - αποθηκεύει στο σωρό τιμές 16byte.
- ✓ **POP** - παίρνει τιμές από το σωρό 16byte.

Σύνταξη για την εντολή **PUSH**:

```
PUSH REG
PUSH SREG
PUSH memory
PUSH immediate
```

REG: AX, BX, CX, DX, DI, SI, BP, SP.

SREG: DS, ES, SS, CS.

memory: [BX], [BX+SI+7], 16 bit μεταβλητή , και άλλα...

immediate: 5, -24, 3Fh, 10001101b, και άλλα...

Σύνταξη για την εντολή **POP**:

```
POP REG
POP SREG
POP memory
```

REG: AX, BX, CX, DX, DI, SI, BP, SP.

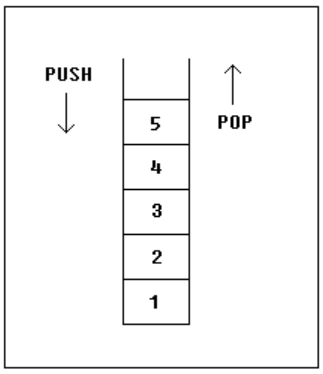
SREG: DS, ES, SS, (εκτός από τον CS).

memory: [BX], [BX+SI+7], 16 bit μεταβλητή , και άλλα...

Ο σωρός χρησιμοποιεί αλγόριθμο **LIFO** *(Ο τελευταίος βγαίνει πρώτος)*. Αυτό σημαίνει πως όταν κάνουμε **PUSH** τιμές μία προς μία στο σωρό:

1, 2, 3, 4, 5

Η **πρώτη** τιμή που θα εξαχθεί θα είναι το **5**, μετά το 4, 3, 2, και τελευταία η 1.



Είναι πολύ σημαντικό όσα **PUSH** γίνονται να έχουμε αντίστοιχα και τόσα **POP**, διαφορετικά ο σωρός μπορεί να φθαρεί και θα είναι απίθανο να επιστρέψει στο λειτουργικό σύστημα, ώστε όταν ξεκινάει το πρόγραμμα να υπάρχει η διεύθυνση επιστροφής στο σωρό *(γενικά είναι η 0000h)*.

Οι εντολές **PUSH** και **POP** είναι ιδιαίτερα χρήσιμες επειδή δεν έχουμε τόσους πολλούς καταχωρητές να χρησιμοποιήσουμε. Οπότε να ένα κόλπο:

- Αποθήκευση της αρχικής τιμής του καταχωρητή στο σωρό *(χρησιμοποιώντας **PUSH**)*.
- Χρήση του καταχωρητή για οποιοδήποτε λόγο.
- Επαναφορά της αρχικής τιμής του καταχωρητή από το σωρό *(χρησιμοποιώντας **POP**)*.

Ορίστε ένα παράδειγμα:

```
ORG   100h
MOV   AX, 1234h
PUSH   AX          ; αποθήκευση του AX στο σωρό .
MOV   AX, 5678h    ; τροποποίηση του AX .
POP   AX           ; επιστροφή αρχικής τιμής του AX στον AX .
RET
END
```

Μια άλλη χρήση του σωρού είναι για ανταλλαγή τιμών. Ορίστε ένα παράδειγμα:

```
ORG   100h

MOV   AX, 1212h   ; αποθήκευση της τιμής 1212h στον AX .
MOV   BX, 3434h   ; αποθήκευση της τιμής 3434h στον BX .

PUSH   AX  ; αποθήκευση του AX στο σωρό .
PUSH   BX  ; αποθήκευση του BX στο σωρό .
POP   AX   ; επιστροφή αρχικής τιμής στον AX .
POP   BX   ; επιστροφή αρχικής τιμής στον BX .

RET
END
```

Η ανταλλαγή συμβαίνει επειδή ο σωρός χρησιμοποιεί αλγόριθμο **LIFO**, οπότε όταν κάνουμε **PUSH** την τιμή **1212h** και μετά την **3434h**, στο **POP** θα πάρουμε πρώτα την τιμή **3434h** και μετά την τιμή **1212h**.

Η περιοχή μνήμης του σωρού ορίζεται από τους καταχωρητές **SS** *(τμήμα σωρού)* και **SP** *(δείκτης σωρού)*. Γενικά το λειτουργικό σύστημα θέτει τιμές σε αυτούς τους καταχωρητές με την εκκίνηση του προγράμματος.

Η **"PUSH πηγή"** εντολή κάνει τα εξής**:**

- Αφαιρεί το **2** από τον καταχωρητή **SP**.
- Γράφει την τιμή της πηγής στη διεύθυνση **SS:SP**.

Η **"POP διαδρομή"** εντολή κάνει τα εξής**:**

- Γράφει την τιμή από τη διεύθυνση **SS:SP** στην διαδρομή.
- Προσθέτει το **2** στον καταχωρητή **SP**.

Η τωρινή διεύθυνση που δείχνετε από το **SS:SP** καλείται κορυφή του σωρού. Για **COM** αρχεία το τμήμα σωρού , είναι γενικά το τμήμα κώδικα και ο δείκτης σωρού παίρνει τη τιμή **0FFFEh** . Στη διεύθυνση **SS:0FFFEh** έχει αποθηκευτεί η διεύθυνση επιστροφής από την εντολή **RET** που εκτελείται στο τέλος του προγράμματος. Στον emu8086 μπορείτε να οπτικά να δείτε τη λειτουργία του σωρού κάνοντας click στο πλήκτρο **[Stack]** στο παράθυρο του εξομοιωτή. Η κορυφή του σωρού συμβολίζεται με το χαρακτήρα **"<"** .

10. Μακροεντολές

Οι μακροεντολές είναι σαν τις διαδικασίες, αλλά όχι πραγματικά. Οι μακροεντολές μοιάζουν με διαδικασίες, αλλά υπάρχουν μόνο μέχρι τη μεταγλωττιστεί ο κώδικας. Μετά τη μεταγλώττιση όλες οι μακροεντολές αντικαθίστανται με πραγματικές εντολές. Αν έχετε δηλώσει μια μακροεντολή και δεν την έχετε χρησιμοποιήσει πότε στο κώδικα, ο μεταγλωττιστής απλά θα την αγνοήσει.

Emu8086.inc είναι ένα πολύ καλό παράδειγμα του πως χρησιμοποιούνται οι μακροεντολές. Αυτό το αρχείο περιλαμβάνει μακροεντολές που κάνουν τον προγραμματισμό πιο εύκολο.

Ορισμός μακροεντολής:

```
όνομα   MACRO  [παράμετροι,...]

     <εντολές>

ENDM
```

Αντίθετα με τις διαδικασίες, οι μακροεντολές πρέπει να οριστούν πάνω από το κώδικα που τις χρησιμοποιεί. Για παράδειγμα:

```
MyMacro   MACRO  p1, p2, p3

     MOV AX, p1
     MOV BX, p2
     MOV CX, p3

ENDM

ORG 100h

MyMacro 1, 2, 3

MyMacro 4, 5, DX

RET
```

Ο πιο πάνω κώδικας αναλύεται σε:

```
MOV AX, 00001h
MOV BX, 00002h
MOV CX, 00003h
MOV AX, 00004h
MOV BX, 00005h
MOV CX, DX
```

Μερικά σημαντικά στοιχεία για τις **μακροεντολές** και τις **διαδικασίες:**

- Όταν θέλετε να χρησιμοποιήσετε μια διαδικασία πρέπει να χρησιμοποιήσετε την εντολή **CALL**. Για παράδειγμα**:**

```
CALL MyProc
```

- Όταν θέλετε να χρησιμοποιήσετε μια μακροεντολή απλά γράφετε το όνομά της. Για παράδειγμα:

```
MyMacro
```

- Η διαδικασία βρίσκεται σε συγκεκριμένη διεύθυνση της μνήμης και αν θέλετε να χρησιμοποιήσετε την ίδια διαδικασία 100 φορές, ο επεξεργαστής πρέπει να μεταφέρει τον έλεγχο σε αυτό το μέρος της μνήμης. Ο έλεγχος θα επιστραφεί πίσω στο πρόγραμμα από την εντολή **RET**. Ο **σωρός** χρησιμοποιείται για να κρατάμε αυτή τη διεύθυνση. Η εντολή **CALL** παίρνει 3byte , οπότε το μέγεθος του εκτελέσιμο αρχείου μεγαλώνει, χωρίς να μας νοιάζει πόσες φορές χρησιμοποιείται η διαδικασία .

- Η μακροεντολή αναλύεται άμεσα στο κώδικα του προγράμματος. Οπότε αν χρησιμοποιήσουμε την ίδια μακροεντολή 100 φορές, ο μεταγλωττιστής αναλύει 100 φορές τη μακροεντολή, κάνοντας το εκτελέσιμο αρχείο όλο και μεγαλύτερο, κάθε φορά που εισάγονται όλες οι εντολές της μακροεντολής.

- Θα πρέπει να χρησιμοποιείται **σωρό** ή οποιοδήποτε καταχωρητή γενικού σκοπού για να περνάτε παραμέτρους στις διαδικασίες.

- Για να περάσετε παραμέτρους σε μακροεντολές, απλά τις γράφετε μετά το όνομα της μακροεντολής. Για παράδειγμα:

```
MyMacro 1, 2, 3
```

- Για να δηλώσετε το τέλος μια μακροεντολής η εντολή **ENDM** είναι αρκετή.

- Για να δηλώσετε το τέλος μια διαδικασίας, πρέπει να γράψετε το όνομα της διαδικασίας πριν την εντολή **ENDP**.

Οι μακροεντολές αναλύονται αμέσως στον κώδικα, ως εκ τούτου αν υπάρχουν ετικέτες μέσα στη δήλωση της μακροεντολής μπορεί να πάρετε σφάλμα "δήλωσης εις διπλούν" όταν η μακροεντολή χρησιμοποιείται περισσότερες από 2 φορές. Για να αποφύγετε αυτό το πρόβλημα, χρησιμοποιείστε την εντολή **LOCAL** ακολουθούμενη από τα ονόματα των μεταβλητών, ετικετών ή ονόματα διαδικασιών.

Για παράδειγμα:

```
MyMacro2  MACRO
     LOCAL label1, label2

     CMP  AX, 2
     JE label1
     CMP  AX, 3
     JE label2

     label1:
          INC  AX
     label2:
          ADD  AX, 2
     ENDM
     ORG 100h

     MyMacro2

     MyMacro2

RET
```

Αν σχεδιάζετε να χρησιμοποιήσετε μακροεντολές σε αρκετά προγράμματα, μπορεί να είναι καλή ιδέα να τοποθετήσετε όλες τις μακροεντολές σε ξεχωριστό αρχείο. Τοποθετήστε αυτό το αρχείο στο φάκελο **Inc** και χρησιμοποιείστε την εντολή **INCLUDE file-name** για να χρησιμοποιήσετε τις μακροεντολές. Δείτε τη βιβλιοθήκη των κοινών συναρτήσεων - **emu8086.inc** για ένα παράδειγμα τέτοιου αρχείο