1ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΑ : Εξοικείωση με τον Η/Υ BGC-8088
Εξοικείωση με τις εντολές του προγράμματος MONITOR
Εντολές MOV, XCHG, LAHF, SAHF
Εκτέλεση απλών προγραμμάτων

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Χρησιμοποιούμενες εντολές:

- Εντολή **MOV** (Move – Μεταφορά δεδομένων μεταξύ καταχωρητών και μνήμης)
- Εντολή **XCHG** (Exchange – Ανταλλαγή δεδομένων μεταξύ καταχωρητών και μνήμης)
- Εντολή **LAHF** (Load AH with Flags – Φόρτωσε τις σημαίες στον AH)
- Εντολή **SAHF** (Store AH into Flags – Αντέγραψε τον AH στις σημαίες)

Γνωριμία με τον Η/Υ BGC-8088

- Περιγραφή του Η/Υ BGC-8088, των τμημάτων που τον απαρτίζουν και των δυνατοτήτων του.
- Εισαγωγή στον Μ/Ε Intel 8088. Καταχωρητές (Γενικοί, Ειδικοί, Καταχωρητές Τμημάτων).
- Αναφορά στον τρόπο προσπέλασης μνήμης segment:offset. Τμήματα Κώδικα, Δεδομένων και Στοίβας.
- Ο Καταχωρητής Σημαιών (FlaG register) και η σημασία των bits του.
- Ο χάρτης μνήμης του BGC-8088

Εξοικείωση με το πρόγραμμα MONITOR

- Γνωριμία με το πρόγραμμα MONITOR και το πληκτρολόγιο. Prompt, πλήκτρο CR, Soft Reset, Hard Reset
- Εντολή R (Registers) και Rxx (μεταβολή τιμής register), Εντολή RF (Register Flags)
- Εντολές D (Display) και E (Edit) για εμφάνιση και μεταβολή της μνήμης.
- Εντολές Συμβολομετάφρασης A (Assemble), I (Insert), U (Unassemble). Δοκιμή με απλό πρόγραμμα με εντολές MOV. Η σημασία του INT 3.
- Εντολή G (Go – Εκτέλεση) και T (Trace – βηματική εκτέλεση).

Η εντολή MOV και οι τρόποι σύνταξής της

(Move – Μεταφορά δεδομένων μεταξύ καταχωρητών και μνήμης)
Απλά μεταφέρει (αντιγράφει) δεδομένα από καταχωρητή/μνήμη προς άλλο καταχωρητή ή μνήμη

- MOV καταχωρητής1, καταχωρητής2 MOV AX,BX
- MOV καταχωρητής, [θέση μνήμης] MOV AX,[200]
- MOV [θέση μνήμης], καταχωρητής MOV [200],CX
- MOV καταχωρητης, τιμή MOV DX, 1AF4
- MOV BY/WO[διεύθυνση μνήμης], τιμή MOV BY[200], 7A

Η εντολή XCHG και οι τρόποι σύνταξής της

(Exchange – Ανταλλαγή δεδομένων μεταξύ καταχωρητών και μνήμης)

Ανταλλάσσει δεδομένα μεταξύ καταχωρητών ή καταχωρητών και μνήμης. Π.χ. XCHG AX,BX σημαίνει AX=BX και BX=AX. Η ανταλλαγή γίνεται με χρήση ενδιάμεσου εσωτερικού καταχωρητή.

- XCHG καταχωρητής1, καταχωρητής 2 XCHG CX,DX
- XCHG καταχωρητής , [διεύθυνση μνήμης] XCHG AX,[200]
- XCHG [διεύθυνση μνήμης] , καταχωρητής XCHG [300],BX

Η εντολή LAHF οι τρόποι σύνταξής της

(Load AH with Flags – Φόρτωσε τις σημαίες στον AH)

Επιτρέπει την καταχώρηση του χαμηλής τάξης byte των σημαιών (σημαίες Carry, Parity, Auxiliary Carry, Zero και Sign) στον καταχωρητή AH, έτσι ώστε να μπορέσουμε να τις επαναφέρουμε μετά από την εκτέλεση μικρού τμήματος εμβόλιμου κώδικα (π.χ. υπορουτίνας) που μπορεί να τις αλλοιώσει.

- LAHF

Η εντολή SAHF οι τρόποι σύνταξής της

(Store AH into Flags – Αντέγραψε τον AH στις σημαίες)

Επαναφέρει τις σωσμένες στον AH τιμές των σημαιών στον καταχωρητή σημαιών. Αντιγράφει τον AH στο χαμηλής τάξης byte του καταχωρητή σημαιών επηρεάζοντας έτσι τις σημαίες (Carry, Parity, Auxiliary Carry, Zero και Sign).

- SAHF

ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Άσκηση 1.1: Εισάγετε στην διεύθυνση 0100:0000 το παρακάτω πρόγραμμα :

```
MOV AX,1234
MOV BX,AX
MOV [200],BX
MOV CX,[200]
MOV BY[202],FF
INT 3
```

- Δείτε το πρόγραμμα με την εντολή U και διορθώστε τυχόν σφάλματα.
- Μέσω της εντολής Rxx βάλτε στους καταχωρητές AX,BX,CX τις τιμές 1111, 2222 και 3333 αντίστοιχα
- Μέσω της εντολής E Βάλτε στις θέσεις μνήμης 0200 έως και 0202 τις τμές 55, 66, 77
- Εκτελέστε το πρόγραμμα βήμα-βήμα με την εντολή T.
- Δείτε ενδιάμεσα και την μνήμη για επιβεβαίωση της σωστής εκτέλεσης των εντολών.

Άσκηση 1.2: Εισάγετε στην διεύθυνση 0100:0000 το παρακάτω πρόγραμμα :

```
MOV AX,[300]
MOV BX,[302]
XCHG AX,BX
XCHG AX,[300]
XCHG BX,[302]
LAHF
MOV [304],AH
INT 3
```

- Δείτε το πρόγραμμα με την εντολή U και διορθώστε τυχόν σφάλματα.
- Μέσω της εντολής Rxx βάλτε στους καταχωρητές AX,BX τις τιμές 1111 και 2222 αντίστοιχα
- Μέσω της εντολής E Βάλτε στις θέσεις μνήμης 0300 και 0301 τον αριθμό AAAA και στις θέσεις μνήμης 0302 και 0303 τον αριθμό BBBB

- Εκτελέστε το πρόγραμμα βήμα-βήμα με την εντολή T.
- Δείτε ενδιάμεσα και την μνήμη για επιβεβαίωση της σωστής εκτέλεσης των εντολών.
- Εξηγήστε τα αποτελέσματα

2ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΟ : Δεκαδικό, Δυαδικό και Δεκαεξαδικό Σύστημα αρίθμησης.
Μετατροπές αριθμών από ένα σύστημα σε άλλο.
Πρόσθεση δυαδικών αριθμών. Αναπαράσταση αρνητικών αριθμών.
Εντολές ADD, ADC, CLC, STC, CMC, CBW, CWD,
Πρόσθεση προσημασμένων ή μη προσημασμένων αριθμών

ΘΕΩΡΗΤΙΚΗ ΕΙΣΑΓΩΓΗ

2.1 Δεκαδικό και δυαδικό σύστημα αρίθμησης

Στο δεκαδικό σύστημα αρίθμησης που έχουμε όλοι συνηθίσει η "**βάση αρίθμησης**" είναι ο αριθμός **10**. Αυτό σημαίνει ότι το δεκαδικό σύστημα αρίθμησης χρησιμοποιεί 10 διαφορετικά σύμβολα για την παράσταση αριθμών (0..9), και ότι το κάθε ψηφίο ενός αριθμού που γράφεται στο δεκαδικό σύστημα πολλαπλασιάζεται με αυξανόμενες δυνάμεις του 10. Για παράδειγμα :

$$274_{10} = \mathbf{2} \times 10^{\mathbf{2}} + \mathbf{7} \times 10^{\mathbf{1}} + \mathbf{4} \times 10^{\mathbf{0}}$$

Στους ηλεκτρονικούς υπολογιστές χρησιμοποιείται ευρέως το δυαδικό σύστημα με βάση αρίθμησης τον αριθμό 2 και 2 μόνο σύμβολα (0 και 1) τα οποία εύκολα αναπαρίστανται "ηλεκτρονικά" (π.χ. με διακόπτες on/off ή το επίπεδο τάσης 0 ή 5 Volt). Με παρόμοιο τρόπο το κάθε ψηφίο ενός δυαδικού αριθμού πολλαπλασιάζεται με αυξανόμενες δυνάμεις του 2. Για παράδειγμα :

$$1011_2 = \mathbf{1} \times 2^{\mathbf{3}} + \mathbf{0} \times 2^{\mathbf{2}} + \mathbf{1} \times 2^{\mathbf{1}} + \mathbf{1} \times 2^{\mathbf{0}} = 8 + 0 + 2 + 1 = 11_{10}$$

Να σημειωθεί ότι η "βάση αρίθμησης" σε ένα σύστημα αρίθμησης δεν αποτελεί βασικό ψηφίο του συστήματος για την αναπαράσταση αριθμών (π.χ. στο δεκαδικό ο αριθμός 10 δεν είναι βασικό ψηφίο (τα ψηφία είναι 0..9) και στο δυαδικό σύστημα ομοίως ο αριθμός 2 δεν είναι βασικό ψηφίο.

2.2 Μετατροπή δυαδικού αριθμού σε δεκαδικό

Με βάση τα παραπάνω ένας εύκολος τρόπος για την μετατροπή σε δεκαδική μορφή ενός δυαδικού αριθμού είναι ο εξής (δίνεται παράδειγμα για δυαδικό αριθμό 8 bit) : Σε κάθε θέση bit αντιστοιχίζεται και μία δύναμη του 2 η οποία είτε προστίθεται (όταν το αντίστοιχο bit είναι 1) είτε όχι (όταν το αντίστοιχο bit είναι 0)

Θέση bit	7	6	5	4	3	2	1	0
Δύναμη του 2	128	64	32	16	8	4	2	1
Δυαδικός Αριθμός	1	0	1	1	0	0	1	1

Άρα ο αριθμός 10110011_2 είναι ο : $128 + 32 + 16 + 2 + 1 = 179_{10}$

2.3 Μετατροπή δεκαδικού αριθμού σε δυαδικό

Η μετατροπή αυτή γίνεται με διαδοχικές διαιρέσεις του αριθμού με τον αριθμό 2 όπου μετά από κάθε διαίρεση ξαναδιαιρούμε με 2 το πηλίκο που έβγαλε η προηγούμενη διαίρεση έως ότου το πηλίκο μηδενιστεί. Επίσης σε κάθε διαίρεση κρατάμε το υπόλοιπο της διαίρεσης (που είναι είτε 0 είτε 1 εφόσον διαιρούμε δια 2). Παράδειγμα μετατροπής του δεκαδικού 179 σε δυαδικό :

Διαίρεση	**Πηλίκο**	**Υπόλοιπο**
179 / 2	89	1
89 / 2	44	1

44 / 2	22	0
22 / 2	11	0
11 / 2	5	1
5 / 2	2	1
2 / 2	1	0
1 / 2	0	1

Από τις παραπάνω διαιρέσεις λαμβάνουμε τα υπόλοιπα των διαιρέσεων με ανάποδη σειρά (δηλαδή το πρώτο υπόλοιπο είναι το bit μικρότερης σημασίας και θα τοποθετηθεί στο δεξί άκρο του αριθμού, ενώ το τελευταίο υπόλοιπο είναι το bit μεγαλύτερης σημασίας και τοποθετείται στο αριστερό άκρο του αριθμού). Έτσι παίρνουμε τον αντίστοιχο δυαδικό αριθμό και επομένως για το παράδειγμά μας :

$$179_{10} = 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1_{\ 2}$$

2.4 Πρόσθεση δυαδικών αριθμών

Η πρόσθεση δυαδικών αριθμών γίνεται όπως ακριβώς και η πρόσθεση των δεκαδικών αριθμών με χρήση "κρατούμενου". Για παράδειγμα, η πρόσθεση των αριθμών $0101_2 = 5_{10}$ και $1101_2 = 13_{10}$ γίνεται ως εξής :

Πράξη	Επεξήγηση
0 1 0 **1** + 1 1 0 **1** **1** **0**	Ένα συν ένα κάνει 2_{10} δηλαδή 10_2 άρα γράφουμε ως αποτέλεσμα το 0 και έχουμε 1 ως κρατούμενο.
0 1 **0** 1 + 1 1 **0** 1 **0** **1** 0	Μηδέν και μηδέν και ένα το κρατούμενο μας δίνει 1, άρα γράφουμε ως αποτέλεσμα το 1 και έχουμε 0 κρατούμενο
0 **1** 0 1 + 1 **1** 0 1 **1** **0** 1 0	Ένα συν ένα κάνει 2_{10} δηλαδή 10_2 άρα γράφουμε ως αποτέλεσμα το 0 και έχουμε 1 ως κρατούμενο.
0 1 0 1 + **1** 1 0 1 **1** 0 0 1 0	Ένα συν μηδέν συν ένα το κρατούμενο κάνει 2_{10} δηλαδή 10_2 άρα γράφουμε ως αποτέλεσμα το 0 και έχουμε 1 ως κρατούμενο. Η πράξη τελείωσε και το αποτέλεσμα είναι $1\ 0\ 0\ 1\ 0_2 = 18_{10}$

2.5 Αναπαράσταση αρνητικών δυαδικών αριθμών

Για την αναπαράσταση των αρνητικών αριθμών ακολουθείται η σύμβαση του **συμπληρώματος ως προς 2** που διευκολύνει εξαιρετικά τις πράξεις στους Η/Υ :

Ο αρνητικός δυαδικός αριθμός ενός θετικού (π.χ. του $3_{10} = 0011_2$) είναι εκείνος ο αριθμός που προστιθέμενος με τον αντίστοιχο θετικό να δώσει αποτέλεσμα 0 (αγνοώντας το κρατούμενο). Έτσι το –3 ως δυαδικός είναι ο 1101 γιατί :

```
      1 1 0 1
+     0 0 1 1
-------------
1     0 0 0 0
```

Για να βρούμε τον αρνητικό δυαδικό αριθμό ενός θετικού δυαδικού αριθμού (με την σύμβαση του συμπληρώματος ως προς 2) αρκεί να κάνουμε τα εξής απλά βήματα :

1. Αντιστρέφουμε όλα τα ψηφία του θετικού αριθμού (π.χ. 0011 → 1100)
2. Προσθέτουμε μία μονάδα (π.χ. 1100 → **1101**)

Αντίστροφα, όταν έχουμε έναν αρνητικό δυαδικό αριθμό και θέλουμε να βρούμε τον αντίστοιχο θετικό αριθμό το λογικό θα ήταν να κάνουμε ακριβώς τα αντίστροφα βήματα :

1. Αφαιρούμε μία μονάδα από τον (αρνητικό) αριθμό (π.χ. 1101 → 1100)

2. Αντιστρέφουμε όλα τα ψηφία του αριθμού (π.χ. 1100 → **0011**)

Τα παραπάνω αντίστροφα βήματα έχουν όμως την ίδια λειτουργία με τα βήματα της πρώτης περίπτωσης (δηλαδή αφαίρεση και αντιστροφή = αντιστροφή και πρόσθεση) και έτσι μπορούμε κάνοντας και πάλι **τα ίδια** βήματα να μετατρέψουμε έναν αρνητικό σε θετικό :

1. Αντιστρέφουμε όλα τα ψηφία του θετικού αριθμού (π.χ. 1101 → 0010)
2. Προσθέτουμε μία μονάδα (π.χ. 0010 → **1101**)

Έτσι, προκύπτει εύκολα ότι ο αρνητικός αριθμός του 00000001 είναι ο 11111111, ενώ ο αρνητικός αριθμός για το 00000000 είναι πάλι το 00000000. Προσέξτε ότι το πλήθος των ψηφίων ενός αρνητικού αριθμού εξαρτάται από το πόσα bits χρησιμοποιούμε. Έτσι αν έχουμε 4μπιτους αριθμούς, ο αριθμός 1 αντιστοιχεί με τον 0001 και ο –1 με τον 1111. Αν έχουμε 8μπιτους αριθμούς, ο αριθμός 1 αντιστοιχεί με τον 00000001 και ο –1 με τον 11111111. Αν έχουμε 16μπιτους αριθμούς, ο αριθμός 1 αντιστοιχεί με τον 0000000000000001 και ο –1 με τον 1111111111111111 κ.ο.κ.

Προφανώς για να αναπαραστήσουμε και θετικούς και αρνητικούς αριθμούς με δεδομένο αριθμό bits, θα πρέπει από τον συνολικό αριθμό συνδυασμών που μπορούμε να πάρουμε με αυτά τα bits, οι μισοί αριθμοί να αντιστοιχούν σε θετικούς και οι άλλοι μισοί σε αρνητικούς. Για παράδειγμα όταν έχουμε αριθμούς των 8 bits αυτοί μπορεί να έχουν 2^8=256 διαφορετικούς συνδυασμούς. Αν θεωρήσουμε ότι όλοι οι αριθμοί είναι θετικοί, τότε μπορούμε με 8 bits να αναπαραστήσουμε τους αριθμούς από 0 (00000000) έως και 255 (11111111). Αν όμως θέλουμε να αναπαραστήσουμε προσημασμένους αριθμούς, δηλαδή και θετικούς αλλά και αρνητικούς, τότε από τους 256 συνδυασμούς οι μισοί (δηλαδή 128) θα είναι θετικοί και οι άλλοι μισοί (128) θα είναι αρνητικοί. Έτσι λοιπόν θεωρώντας και το 0 στους θετικούς αριθμούς οι προσημασμένοι αριθμοί που αναπαρίστανται με 8 bits είναι στην περιοχή :

$$\underbrace{-128, \ldots -2, -1,}_{\text{αρνητικοί}} \quad \underbrace{0\ , 1\ , 2, \ldots ,+127}_{\text{θετικοί}}$$

Η χαρακτηριστική διαφορά θετικών και αρνητικών δυαδικών αριθμών είναι ότι στους θετικούς αριθμούς το bit μεγαλύτερης σημασίας είναι 0 ενώ στους αρνητικούς είναι 1. Π.χ για τους 8μπιτους αριθμούς οι θετικοί αναπαρίστανται ως :

$$\begin{aligned} 00000000_2 &= 0_{10} \\ 00000001_2 &= 1_{10} \\ 00000010_2 &= 2_{10} \\ \ldots &= \ldots \\ 01111110_2 &= 126_{10} \\ 01111111_2 &= 127_{10} \end{aligned}$$

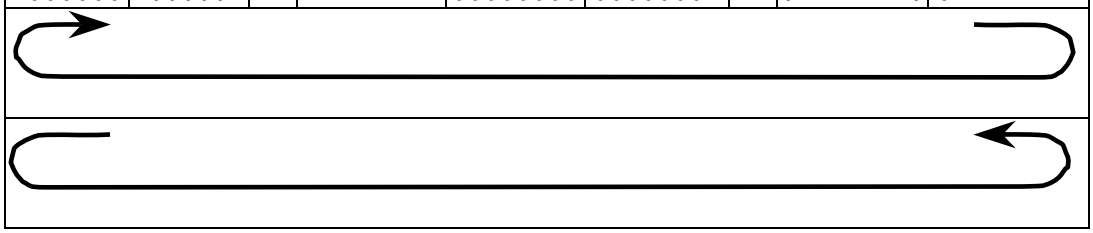
δηλαδή ως κανονικοί δυαδικοί αριθμοί αλλά με τα πρώτα 7 ψηφία ενεργά (2^7=128). Προφανώς οι αρνητικοί αριθμοί (που είναι 128) παράγονται επίσης με συνδυασμούς των πρώτων 7 ψηφίων (2^7=128) αλλά με το $8^{ο}$ ψηφίο ίσο με 1, ώστε να διαφοροποιούνται από τους θετικούς. Έτσι οι αρνητικοί θα είναι ως εξής :

$$\begin{aligned} 11111111_2 &= -1_{10} \\ 11111110_2 &= -2_{10} \\ 11111101_2 &= -3_{10} \\ \ldots &= \ldots \\ 10000001_2 &= -127_{10} \\ 10000000_2 &= -128_{10} \end{aligned}$$

Παρατηρούμε ότι αν στον αριθμό $01111111_2 = 127_{10}$ προσθέσουμε μία μονάδα παίρνουμε τον αριθμό $10000000_2 = -128_{10}$, δηλαδή αν περάσουμε προς τα πάνω το άνω όριο των προσημασμένων αριθμών τότε ξαναγυρνάμε στο κάτω όριο. Ομοίως αν από τον αριθμό $10000000_2 = -128_{10}$, αφαιρέσουμε μία μονάδα, προκύπτει ο αριθμός $01111111_2 = 127_{10}$. δηλαδή αν περάσουμε

προς τα κάτω το κάτω όριο των προσημασμένων αριθμών ξαναγυρνάμε στο πάνω όριο. Αυτές οι δύο περιπτώσεις κατά τις οποίες ξεπερνάμε την δεδομένη περιοχή αριθμών (θετικών-αρνητικών) που μπορούν να αναπαρασταθούν με δεδομένο αριθμό bits, ονομάζονται περιπτώσεις **υπερχείλισης (overflow)** και προφανώς μας δίνουν εσφαλμένα αποτελέσματα (αλλαγή προσήμου και λάθος νούμερο). Οι περιπτώσεις υπερχείλισης πρέπει να ελέγχονται από τον προγραμματιστή σε κάθε αριθμητική πράξη ενός προγράμματος, όταν βέβαια είναι πιθανόν να συμβούν.

-128	-127	...	-1	0	1	...	126	127
1000000	1000001		11111111	00000000	00000001		011111110	011111111



Αντίστοιχα όταν έχουμε αριθμούς των 16 bits αυτοί μπορεί να έχουν 2^{16}=65536 διαφορετικούς συνδυασμούς. Αν θεωρήσουμε ότι όλοι οι αριθμοί είναι θετικοί, τότε μπορούμε με 16 bits να αναπαραστήσουμε τους αριθμούς από 0 (00000000 00000000) έως και 65535 (11111111 11111111). Αν όμως θέλουμε να αναπαραστήσουμε προσημασμένους αριθμούς, δηλαδή και θετικούς αλλά και αρνητικούς, τότε από τους 65536 συνδυασμούς οι μισοί (δηλαδή 32768) θα είναι θετικοί και οι άλλοι μισοί θα είναι αρνητικοί. Έτσι λοιπόν θεωρώντας και το 0 στους θετικούς αριθμούς οι προσημασμένοι αριθμοί που αναπαρίστανται με 16 bits είναι στην περιοχή :

$$\underbrace{-32768, \ldots -2, -1}_{\text{αρνητικοί}}, \quad \underbrace{0\ , 1\ , 2, \ldots ,+32767}_{\text{θετικοί}}$$

2.6 Το δεκαεξαδικό σύστημα

Στους ηλεκτρονικούς υπολογιστές χρησιμοποιείται ευρέως και το δεκαεξαδικό (hexadecimal) σύστημα αρίθμησης που χρησιμοποιεί ως βάση αρίθμησης τον αριθμό 16, δηλαδή χρησιμοποιεί 16 διαφορετικά σύμβολα (0..9, A, B, C, D, E, F) για αναπαράσταση αριθμών. Το κάθε ψηφίο ενός δεκαεξαδικού αριθμού πολλαπλασιάζεται με αυξανόμενες δυνάμεις του 16. Για παράδειγμα :

$$2FA_{16} = \mathbf{2} \times 16^{\mathbf{2}} + \mathbf{F} \times 16^{\mathbf{1}} + \mathbf{A} \times 16^{\mathbf{0}} = 2 \times 256 + 15 \times 16 + 10 = 762_{10}$$

Το δεκαεξαδικό σύστημα χρησιμοποιείται στους Η/Υ για δύο λόγους :

- μπορεί και αναπαριστά μεγάλους αριθμούς (π.χ. διευθύνσεις μνήμης) με λίγα ψηφία, και
- οι δεκαεξαδικοί αριθμοί μετατρέπονται εύκολα σε δυαδικούς και αντίστροφα.

2.7 Μετατροπή δεκαεξαδικών αριθμών σε δυαδικούς και αντίστροφα

Η μετατροπή δεκαεξαδικών αριθμών σε δυαδικούς γίνεται ως εξής : Κάθε ψηφίο του δεκαεξαδικού αριθμού αντιστοιχεί σε 4 ψηφία του δυαδικού αριθμού. Έτσι ένας 8μπιτος δυαδικός αριθμός αναπαρίσταται με μόνο 2 ψηφία στο δεκαεξαδικό σύστημα (π.χ. 255_{10} = 1111 1111_2 = FF_{16}). Έτσι για παράδειγμα ο αριθμός D7 μετατρέπεται σε δυαδικός ως εξής :

Αρχικός δεκαεξαδικός αριθμός	$D7_{16}$	
Σε δεκαεξαδικό σύστημα ανά ψηφίο	D_{16}	7_{16}
Σε δεκαδικό σύστημα ανά ψηφίο	13_{10}	7_{10}
Σε δυαδικό σύστημα ανά ψηφίο	1101_2	0111_2
Τελικός δυαδικός	1101 0111_2	

Με τον ακριβώς αντίστροφο τρόπο γίνεται η μετατροπή από δυαδικό σε δεκαεξαδικό. Για παράδειγμα η μετατροπή του δυαδικού 10100101_2 γίνεται ως εξής

Αρχικός δυαδικός αριθμός	10100101_2	
Χωρισμός ανά 4 ψηφία	1010_2	0101_2
Σε δεκαδικό σύστημα ανά ψηφίο	12_{10}	5_{10}
Σε δεκαεξαδικό σύστημα ανά ψηφίο	C_{16}	5_{16}
Τελικός δεκαεξαδικός αριθμός	$C5_{16}$	

Χρησιμοποιούμενες εντολές:

- **ADD** (Add – Πρόσθεση χωρίς κρατούμενο)
- **ADC** (Add with Carry – Πρόσθεση με κρατούμενο)
- **CLC** (Clear Carry – Μηδενίζει το κρατούμενο)
- **STC** (Set Carry – Θέτει το κρατούμενο)
- **CMC** (Complement Carry – Συμπληρωματικό κρατούμενο)
- **CBW** (Convert Byte to Word – Μετατροπή Byte σε Word)
- **CWD** (Convert Word to Double Word – Μετατροπή Word σε Double Word)

Η εντολή ADD και οι τρόποι σύνταξής της
(Add – Πρόσθεση χωρίς κρατούμενο)
Προσθέτει τα ορίσματα χωρίς την χρήση κρατούμενου και βάζει το αποτέλεσμα στο πρώτο όρισμα. Αν προκύψει κρατούμενο τότε γίνεται Carry=1. Π.χ. ADD AX,CX σημαίνει AX=AX+CX

- ADD καταχωρητής1, καταχωρητής2 ADD AX,BX
- ADD καταχωρητής, [θέση μνήμης] ADD AX,[200]
- ADD [θέση μνήμης], καταχωρητής ADD [300],BX
- ADD καταχωρητης, τιμή ADD DX, 1AF4
- ADD BY/WO[διεύθυνση μνήμης], τιμή ADD BY[0456],6B

Η εντολή ADC και οι τρόποι σύνταξής της
(Add with Carry– Πρόσθεση με κρατούμενο)
Προσθέτει τα ορίσματα και το κρατούμενο και βάζει το αποτέλεσμα στο πρώτο όρισμα. Αν προκύψει κρατούμενο τότε γίνεται Carry=1. Π.χ. ADC DX,BX σημαίνει DX=DX+BX+Carry

- ADC καταχωρητής1, καταχωρητής2 ADC DX,CX
- ADC καταχωρητής, [θέση μνήμης] ADC BX,[9876]
- ADC [θέση μνήμης], καταχωρητής ADC [1234],CX
- ADC καταχωρητης, τιμή ADC AX, 3B6A
- ADC BY/WO[διεύθυνση μνήμης], τιμή ADC WO[FF2D],9CA8

Η εντολή CLC και οι τρόποι σύνταξής της
(Clear Carry – Μηδενίζει το κρατούμενο)
Κάνει το κρατούμενο 0 (Carry=0 ή NC)

- CLC

Η εντολή STC και οι τρόποι σύνταξής της
(Set Carry – Θέτει το κρατούμενο)
Κάνει το κρατούμενο 1 (Carry=1 ή CY)

- STC

Η εντολή CMC και οι τρόποι σύνταξής της
(Complement Carry – Συμπληρωματικό κρατούμενο)
Αν το κρατούμενο είναι 0 γίνεται 1, και αν είναι 1 γίνεται 0.

- CMC

Η εντολή CBW και οι τρόποι σύνταξής της
(Convert Byte to Word – Μετατροπή Byte σε Word)
Μετατρέπει τον προσημασμένο 8-bit αριθμό που βρίσκεται στον καταχωρητή AL σε προσημασμένο 16-bit αριθμό που αποθηκεύεται στον AX. Π.χ. 01→0001, 80→FF80

- CBW

Η εντολή CWD και οι τρόποι σύνταξής της
(Convert Word to Double Word – Μετατροπή Word σε Double Word)
Μετατρέπει τον προσημασμένο 16-bit αριθμό που βρίσκεται στον καταχωρητή AX σε προσημασμένο 32-bit αριθμό που αποθηκεύεται στους καταχωρητές DX: AX. Π.χ. 7FFF→0000:7FFF, C5A8→FFFF:C5A8

- CWD

ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Άσκηση 2.1: Γράψτε ένα πρόγραμμα που να προσθέτει δύο μη προσημασμένους αριθμούς των 8 bits που είναι περιεχόμενα των θέσεων μνήμης **0200** και **0201** και να αποθηκεύει το άθροισμα στη θέση μνήμης **0202**.

- Εκτελέστε το πρόγραμμά σας για τις τρεις περιπτώσεις που υπάρχουν στον παρακάτω πίνακα και συμπληρώστε τον.

0200		**0201**		**0202**		
16-δικός	10-δικός	16-δικός	10-δικός	16-δικός	10-δικός	Κρατούμενο (Ναι/Όχι)
8F	143	83	131			
2E	46	41	65			
3B	59	C5	197			

- Στις περιπτώσεις που υπάρχει κρατούμενο η θέση **0202** κρατάει το σωστό αποτέλεσμα;

Άσκηση 2.2: Γράψτε ένα πρόγραμμα που να προσθέτει δύο προσημασμένους αριθμούς των 8 bits που είναι περιεχόμενα των θέσεων μνήμης **0200** και **0201** και να αποθηκεύει το άθροισμα στη θέση μνήμης **0202**.

- Το πρόγραμμα αυτό σε τι διαφέρει από το προηγούμενο;
- Εκτελέστε το πρόγραμμά σας για τις τιμές του παρακάτω πίνακα και συμπληρώστε τον.
- Ποια αποτελέσματα στη **0202** θέση είναι σωστά; Γιατί;

0200		**0201**		**0202**		
16-δικός	10-δικός	16-δικός	10-δικός	16-δικός	10-δικός	Υπερχείλιση ? (Ναι/Όχι)
6A	106	59	89			
7F	127	03	3			
20	32	FD	-3			
A6	-90	C2	-62			

Άσκηση 2.3: Γράψτε ένα πρόγραμμα που να προσθέτει δύο 16-bit προσημασμένους αριθμούς που ο ένας είναι αποθηκευμένος στις θέσεις μνήμης **0200** (χαμηλής τάξης BYTE), **0201** (υψηλής τάξης BYTE), και ο άλλος στις **0202** (Χ.Τ.Β.) και **0203** (Υ.Τ.Β.). Το αποτέλεσμα να αποθηκευθεί στις θέσεις μνήμης **0204** (Χ.Τ.Β.) και **0205** (Υ.Τ.Β.).

Για παράδειγμα. εάν ο πρώτος προσθετέος είναι ο **186F** και ο δεύτερος ο **235C** οι θέσεις μνήμης θα περιέχουν :

0200: 6F
0201: 18
0202: 5C
0203: 23

και μετά την άθροιση :

186F
+235C
3BCB

Οι θέσεις **0204** και **0205** θα περιέχουν:

0204: CB
0205: 3B

- Σε τι διαφέρει το πρόγραμμα αυτό από το προηγούμενο ;
- Εκτελέστε το πρόγραμμά σας για τα δεδομένα του παρακάτω πίνακα:

Προσθετέος **0200,0201**	Προσθετέος **0202,0203**	Άθροισμα **0204,0205**	**Υπερχείλιση ?** (Ναι/Όχι)
2563	41AB		
83C7	F1B4		

Άσκηση 2.4: Γράψτε ένα πρόγραμμα που να προσθέτει δύο 32-bit προσημασμένους αριθμούς που ο ένας είναι αποθηκευμένος στις θέσεις μνήμης **0200...0203**, και ο άλλος στις **0204...0207**. Το αποτέλεσμα να αποθηκευθεί στις θέσεις μνήμης **0208...020B**.

- Σε τι διαφέρει το πρόγραμμα αυτό από το προηγούμενο ;
- Εκτελέστε το πρόγραμμά σας για τα δεδομένα του παρακάτω πίνακα:

Προσθετέος **0200...0203**	Προσθετέος **0204...0207**	Άθροισμα **0208...020B**	**Υπερχείλιση ?** (Ναι/Όχι)
1234 5678	2345 6789		
7FFF FFFF	0000 0001		

3ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΟ : Αφαίρεση προσημασμένων ή μη προσημασμένων αριθμών

Εντολές SUB, SBB, NEG

Πολλαπλασιασμός προσημασμένων ή μη προσημασμένων αριθμών

Εντολές MUL, IMUL

Διαίρεση προσημασμένων ή μη προσημασμένων αριθμών

Εντολές DIV, IDIV

ΘΕΩΡΗΤΙΚΗ ΕΙΣΑΓΩΓΗ

3.1 Αφαίρεση δυαδικών αριθμών

Η αφαίρεση δυαδικών αριθμών γίνεται κατ' αντιστοιχία με την πρόσθεση των δυαδικών αριθμών που είδαμε στο προηγούμενο κεφάλαιο. Για παράδειγμα, η αφαίρεση των αριθμών $1101_2 = 13_{10}$ και $0110_2 = 6_{10}$ γίνεται ως εξής :

Πράξη	Επεξήγηση
1 1 0 **1** - 0 1 1 **0** **0** **1**	Μηδέν από ένα μας δίνει 1. Δεν χρειάστηκε δανεικό (δανεικό=0).
1 **1** 0 1 - 0 1 **1** 0 **1** **1** 1	Ένα από Μηδέν δεν αφαιρείται. Έτσι παίρνουμε δανεικό και λέμε Ένα (1_2) από Δύο (10_2) μας δίνει 1, άρα γράφουμε ως αποτέλεσμα το 1 και έχουμε 1 δανεικό
1 **1** 0 1 - 0 **1** 1 0 **1** **1** 1 1	Ένα το δανεικό και Ένα μας κάνουν Δύο (10_2), από Ένα (1_2) δεν αφαιρείται. Έτσι παίρνουμε δανεικό και λέμε Δύο (10_2) από Τρία (11_2) μας δίνουν Ένα, άρα γράφουμε ως αποτέλεσμα το 1 και έχουμε 1 ως δανεικό.
1 1 0 1 - **0** 1 1 0 **0** **0** 1 1 1	Ένα το δανεικό και 0 μας κάνουν ένα (1_2), από Ένα μας δίνει 0 άρα γράφουμε ως αποτέλεσμα το 0 και δεν έχουμε δανεικό. Η πράξη τελείωσε και το αποτέλεσμα είναι $1\ 1\ 1_2 = 7_{10}$

Κατά την αφαίρεση, αν ο αφαιρετέος είναι συνολικά μεγαλύτερος από τον μειωτέο (π.χ. A-B και A<B) τότε προκύπτει δανεικό (Borrow) το οποίο εκφράζεται με την σημαία κρατούμενου (Carry). Όταν προκύψει δανεικό τότε Carry=1.

3.2 Πολλαπλασιασμός και Διαίρεση αριθμών

Το σετ εντολών του 8088 περιλαμβάνει και εντολές πολλαπλασιασμού και διαίρεσης ακέραιων αριθμών. Οι πράξεις αυτές γίνονται πάντα με την συμμετοχή του καταχωρητή AX. Συγκεκριμένα μπορεί να πολλαπλασιαστεί ο AX με έναν από τους καταχωρητές (AX,BX,CX,DX,BP,SI,DI,SP) και το αποτέλεσμα καταχωρείται στο 32 bit ζευγάρι καταχωρητών DX:AX (DX:AX = AX * καταχωρητής). Μπορεί βέβαια η πράξη να γίνει με μικρότερους αριθμούς, δηλαδή να πολλαπλασιαστεί ο AL με έναν 8bit καταχωρητή (π.χ. BL) και το αποτέλεσμα να πάει στον AX (AX = AL * καταχωρητής).

Έχει βέβαια μεγάλη σημασία το αν οι αριθμοί θεωρούνται προσημασμένοι ή όχι. Για παράδειγμα αν πολλαπλασιάσουμε τους αριθμούς :

$FFFF_{16}$ X 0002_{16}

Τότε, αν δεχθούμε ότι οι αριθμοί είναι χωρίς πρόσημο (65535_{10} X 2_{10}) το αποτέλεσμα θα είναι 131070_{10}, δηλαδή DX:AX = 0001 FFFE.

Αν όμως οι αριθμοί είναι προσημασμένοι τότε πρόκειται για την πράξη (-1_{10} X 2_{10}) και το αποτέλεσμα είναι -2_{10}, δηλαδή DX:AX = FFFF FFFE.

Με παρόμοιο τρόπο γίνεται και η διαίρεση ακέραιων αριθμών με συμμετοχή των καταχωρητών AX και DX. Συγκεκριμένα διαιρείται ο 32 bit αριθμός που βρίσκεται στο ζευγάρι καταχωρητών DX:AX με τον καταχωρητή που θέλουμε (AX,BX,CX,DX,BP,SI,DI,SP), και το πηλίκο πηγαίνει στον AX ενώ το υπόλοιπο στον DX (AX = DX:AX / καταχωρητής, DX=υπόλοιπο). Μπορεί βέβαια η πράξη να γίνει με μικρότερους αριθμούς, δηλαδή να διαιρεθεί ο AX με έναν 8bit καταχωρητή (π.χ. BL) και το πηλίκο να πάει στον AL ενώ το υπόλοιπο πάει στον AH (AL = AX / καταχωρητής, AH=υπόλοιπο).

Χρησιμοποιούμενες εντολές:

- **SUB** (Subtract – Αφαίρεση χωρίς δανεικό)
- **SBB** (Subtract with Borrow – Αφαίρεση με δανεικό)
- **NEG** (Negative - Αρνητικό)
- **MUL** (Multiply unsigned – Πολλαπλασιασμός αριθμών χωρίς πρόσημο)
- **IMUL** (Integer Multiply signed – Πολλαπλασιασμός αριθμών με πρόσημο)
- **DIV** (Divide unsigned – Διαίρεση αριθμών χωρίς πρόσημο)
- **IDIV** (Integer Divide signed – Διαίρεση αριθμών με πρόσημο)

Η εντολή SUB και οι τρόποι σύνταξής της

(Subtract – Αφαίρεση χωρίς δανεικό)

Αφαιρείται από το πρώτο όρισμα το δεύτερο και το αποτέλεσμα πηγαίνει στο πρώτο όρισμα. Π.χ. SUB AX,BX σημαίνει AX=AX-BX. Αν προκύψει δανεικό "ανάβει" το Carry.

- SUB καταχωρητής1, καταχωρητής2 — SUB AX,BX
- SUB καταχωρητής, [θέση μνήμης] — SUB AX,[200]
- SUB [θέση μνήμης], καταχωρητής — SUB [300],BX
- SUB καταχωρητης, τιμή — SUB DX, 1AF4
- SUB BY/WO[διεύθυνση μνήμης], τιμή — SUB BY[0456],6B

Η εντολή SBB και οι τρόποι σύνταξής της

(Subtract with Borrow – Αφαίρεση με δανεικό)

Αφαιρείται από το πρώτο όρισμα το δεύτερο και το δανεικό και το αποτέλεσμα πηγαίνει στο πρώτο όρισμα. Π.χ. SUB CX,DX σημαίνει CX=CX-DX-Carry. Αν προκύψει δανεικό "ανάβει" το Carry.

- SBB καταχωρητής1, καταχωρητής2 — SBB AX,BX
- SBB καταχωρητής, [θέση μνήμης] — SBB AX,[200]
- SBB [θέση μνήμης], καταχωρητής — SBB [300],BX
- SBB καταχωρητης, τιμή — SBB DX, 1AF4
- SBB BY/WO[διεύθυνση μνήμης], τιμή — SBB BY[0456],6B

Η εντολή NEG και οι τρόποι σύνταξής της

(Negative - Αρνητικό)

Αλλάζει πρόσημο στην τιμή του καταχωρητή με βάση το συμπλήρωμα ως προς 2.

- NEG καταχωρητής NEG BX

Η εντολή MUL και οι τρόποι σύνταξής της
(Multiply unsigned – Πολλαπλασιασμός αριθμών χωρίς πρόσημο)
Πολλαπλασιάζει τον καταχωρητή AX με τον καταχωρητή που δίνουμε (χωρίς πρόσημο) και βάζει το αποτέλεσμα στο 32-bit ζευγάρι καταχωρητών DX:AX (DX:AX = AX * καταχωρητής) ή (AX = AL * καταχ/τής).
- MUL καταχωρητής MUL DX

Η εντολή IMUL και οι τρόποι σύνταξής της
(Integer Multiply signed – Πολλαπλασιασμός αριθμών με πρόσημο)
Πολλαπλασιάζει τον καταχωρητή AX με τον καταχωρητή που δίνουμε (προσημασμένοι αριθμοί) και βάζει το αποτέλεσμα στο 32-bit ζευγάρι καταχωρητών DX:AX (DX:AX = AX * καταχωρητής) ή (AX = AL * καταχ/τής).
- IMUL καταχωρητής IMUL BP

Η εντολή DIV και οι τρόποι σύνταξής της
(Divide unsigned – Διαίρεση αριθμών χωρίς πρόσημο)
Διαιρεί το 32-bit ζευγάρι καταχωρητών DX:AX με τον καταχωρητή που δίνουμε (χωρίς πρόσημο) και βάζει το πηλίκο στον AX και το υπόλοιπο στον DX (AX = DX:AX / καταχωρητής, DX=υπόλοιπο) ή (AL = AX / καταχωρητής, AH=υπόλοιπο).
- DIV καταχωρητής DIV AX

Η εντολή IDIV και οι τρόποι σύνταξής της
(Integer Divide signed – Διαίρεση αριθμών με πρόσημο)
Διαιρεί το 32-bit ζευγάρι καταχωρητών DX:AX με τον καταχωρητή που δίνουμε (προσημασμένοι αριθμοί) και βάζει το πηλίκο στον AX και το υπόλοιπο στον DX (AX = DX:AX / καταχωρητής, DX=υπόλοιπο) ή (AL = AX / καταχωρητής, AH=υπόλοιπο).
- IDIV καταχωρητής IDIV DX

ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Άσκηση 3.1: Γράψτε ένα πρόγραμμα που να αφαιρεί τους μη προσημασμένους αριθμούς **9** και **5** (9-5) με χρήση της εντολής **SBB** μια φορά με Carry=0 πριν από την αφαίρεση και μια φορά με Carry=1. Ποιο αποτέλεσμα είναι το σωστό?

Άσκηση 3.2: Γράψτε ένα πρόγραμμα που να αφαιρεί δύο προσημασμένους αριθμούς των 8 bit που βρίσκονται στις θέσεις μνήμης **200** και **201** και να τοποθετεί το αποτέλεσμα στην θέση μνήμης **202**. Κάντε δοκιμή για τα παρακάτω δεδομένα :

200	201	202	Carry	Overflow	Negative	Σωστό αποτέλ. ?
$6F_{16}$ (111_{10})	70_{16} (112_{10})					
FF_{16} (-1_{10})	02_{16} (2_{10})					
80_{16} (-128_{10})	01_{16} (1_{10})					

Άσκηση 3.3: Γράψτε ένα πρόγραμμα που να αφαιρεί τον 16-bit προσημασμένο αριθμό που βρίσκεται στις θέσεις μνήμης **0202**, **0203** (Υ.Τ.Β. στην 0203) από τον 16-bit προσημασμένο αριθμό που βρίσκεται στις θέσεις μνήμης **0200**, **0201**, (Υ.Τ.Β. στην 0201). Βάλτε το αποτέλεσμα της αφαίρεσης στις θέσεις **0204**, **0205,** και το αρνητικό του στις θέσεις μνήμης **0206,0207**.
Κάντε δοκιμή για τα παρακάτω δεδομένα :

200,201	202,203	204,205	206,207	CY	OV	NG	Σωστό ?

4321_{16} (17185_{10})	3210_{16} (12816_{10})						
$ABCD_{16}$ (-21555_{10})	1111_{16} (4369_{10})						
8000_{16} (-32768_{10})	0002_{16} (2_{10})						

Άσκηση 3.4: Γράψτε ένα πρόγραμμα που να πολλαπλασιάζει τους μη προσημασμένους 16 bit αριθμούς που βρίσκονται στις θέσεις μνήμης 0200-0201 και 0202-0203 και να βάζει το αποτέλεσμα στις θέσεις μνήμης 0204-0207.
Εκτελέστε το πρόγραμμά σας για τα δεδομένα του παρακάτω πίνακα:

Πολ/στής **0200...0201**	Πολ/στής **0202...0203**	Γινόμενο **0204...0207**	**Σωστό ?** (Ναι/Όχι)
1234_{16} (4660_{10})	0100_{16} (256_{10})		
$FFFF_{16}$ (65535_{10})	$FFFF_{16}$ (65535_{10})		

Άσκηση 3.5: Γράψτε ένα πρόγραμμα που να διαιρεί τον 32 bit προσημασμένο αριθμό που βρίσκεται στις θέσεις μνήμης 0200-0203, με τον 16-bit προσημασμένο αριθμό που βρίσκεται στις θέσεις μνήμης 0204-0205 και να βάζει το Πηλίκο στις θέσεις μνήμης 0206-0207 και το Υπόλοιπο στις θέσεις μνήμης 0208-0209.
Εκτελέστε το πρόγραμμά σας για τα δεδομένα του παρακάτω πίνακα:

Διαιρετέος **0200...0203**	Διαιρέτης **0204...0205**	Πηλίκο **0206...0207**	Υπόλοιπο **0208...0209**	**Σωστό ?** (Ναι/Όχι)
0123 4001_{16} (19087361_{10})	1000_{16} (4096_{10})			
FFFF 0000_{16} (-65536_{10})	8000_{16} (-32768_{10})			

Είναι όλες οι διαιρέσεις εφικτές με τις εντολές DIV και IDIV και με τις προδιαγραφές που αυτές έχουν ?

Άσκηση 3.6: Γράψτε ένα πρόγραμμα που να αφαιρεί τον 32-bit προσημασμένο αριθμό που βρίσκεται στις θέσεις μνήμης **0204**, **0205, 206, 207** (Υ.Τ.Β. στην 0207) από τον 32-bit προσημασμένο αριθμό που βρίσκεται στις θέσεις μνήμης **0200**, **0201**, **202, 203,** (Υ.Τ.Β. στην 0203). Βάλτε το αποτέλεσμα της αφαίρεσης στις θέσεις **0208**, **0209, 020A, 020B**.
Εκτελέστε το πρόγραμμά σας για τα δεδομένα του παρακάτω πίνακα:

Μειωτέος **0200...0203**	Αφαιρετέος **0204...0207**	Διαφορά **0208...020B**	**Υπερχείλιση ?** (Ναι/Όχι)
2345 6789	1234 5678		
8000 0000	0000 0001		

4ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΟ : Σύγκριση δύο αριθμών

Εντολή CMP

Διακλάδωση προγράμματος υπό συνθήκη

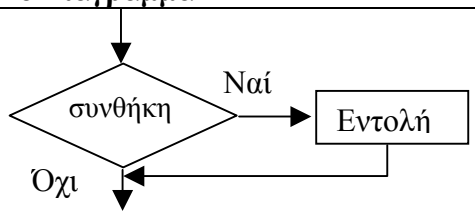
Εντολές Jxx, JMP

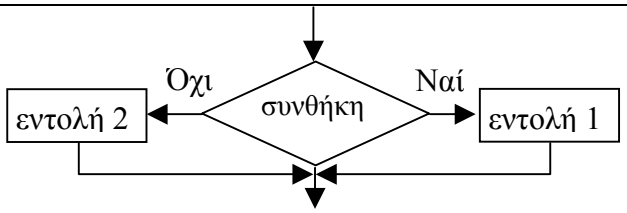
Υλοποίηση δομών if-then, if-then-else, case/switch

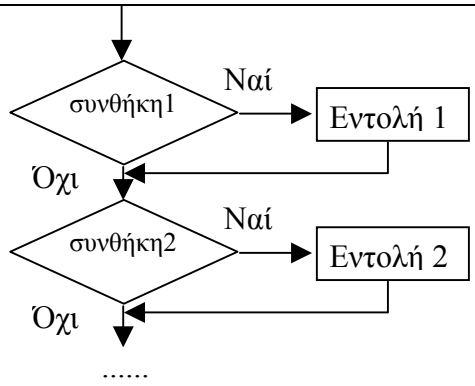
ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Μία από τις σημαντικότερες δυνατότητες των γλωσσών προγραμματισμού είναι η υλοποίηση δομών «ελέγχου ροής προγράμματος». Οι δομές αυτές επιτρέπουν την αλλαγή της ομαλής ροής του προγράμματος και την εκτέλεση εμβόλιμων τμημάτων κώδικα σε περίπτωση που ισχύει μία συνθήκη. Έτσι μπορούμε να φτιάχνουμε «έξυπνα» προγράμματα που εκτελούν διαφορετικά μπλόκ εντολών ανάλογα με τις συνθήκες ή που απλά προσαρμόζουν την εκτέλεσή τους ανάλογα με τις παραμέτρους που δέχονται από τον χρήστη ή το περιβάλλον (Λειτουργικό Σύστημα). Οι κοινές δομές που χρησιμοποιούνται στις γλώσσες προγραμματισμού είναι :

Α/Α	**Δομή**	**Λογικό Διάγραμμα**	**Παράδειγμα σε C**

1	If - then		…
			if (k>10)
			{
			k=k-10;
			reset=true;
			}
			…

2	If – then - else		…
			if (A[i]==Search)
			{ Found=true;
			Position=i; }
			else i=i+1;
			…

3	Case / Switch		…
			switch (key)
			{
			case ‘U’ : y--;break;
			case ‘D’ : y++; break;
			case ‘L’ : x--; break;
			case ‘R’ : x++; break;
			}
			…

Οι δομές αυτές προϋποθέτουν την αποτίμηση μίας «συνθήκης». Αυτό σε επίπεδο γλώσσας μηχανής γίνεται με την εντολή CMP (Compare) η οποία μπορεί να συγκρίνει δύο αριθμούς που βρίσκονται σε καταχωρητές, στη μνήμη ή είναι σταθερά νούμερα. Η εντολή CMP μετά την σύγκριση δεν κάνει τίποτα άλλο παρά να επηρεάζει τις τιμές των bits του καταχωρητή σημαιών (Flag Register). Για να μπορούμε όμως να αλλάζουμε την ροή του προγράμματος, υπάρχουν μία σειρά από εντολές διακλάδωσης υπό συνθήκη (Εντολές Jxx – Jump Conditional) από τις οποίες η κάθε μία ελέγχει και διαφορετικές σημαίες του καταχωρητή σημαιών και μεταφέρει την εκτέλεση του προγράμματος σε κάποια άλλη διεύθυνση μνήμης. Έτσι οι παραπάνω δομές γενικά υλοποιούνται με συνδυασμό εντολών CMP και Jxx. Επειδή όμως και άλλες εντολές επηρεάζουν τα bits του καταχωρητή σημαιών, δεν είναι πάντα απαραίτητο να προηγείται εντολή CMP πρίν από τις εντολές Jxx. Για παράδειγμα η εντολή ADD μπορεί κατά την εκτέλεσή της να επηρεάσει τις σημαίες Carry, Zero, Negative και Overflow, οπότε θα μπορούσαμε κατευθείαν να ελέγξουμε το αποτέλεσμα της ADD για υπερχείλιση χωρίς την CMP ως εξής :

```
0100:0000 ADD AX,[0200]
0100:0003 JO 0050
… (Εντολές που εκτελούνται όταν ΔΕΝ υπάρχει υπερχείλιση)
0100:0050 (Εντολές που εκτελούνται όταν υπάρχει υπερχείλιση)
```

Χρησιμοποιούμενες εντολές:

- **CMP** (Compare – Σύγκριση δύο αριθμών)
- **JMP** (Jump – Μεταπήδηση προγράμματος σε άλλη διεύθυνση χωρίς συνθήκη)
- **JE** ή **JZ** (Jump on Equal/Zero)
  Διακλάδωση προγράμματος αν το αποτέλεσμα της προηγούμενης πράξης είχε αποτέλεσμα 0 (ZF=1)
- **JNE** ή **JNZ** (Jump on Not Equal/Not Zero)
  Διακλάδωση προγράμματος αν το αποτέλεσμα της προηγούμενης πράξης είχε αποτέλεσμα διάφορο του 0 (ZF=0)
- **JL** ή **JNGE** (Jump on Less/Not Greater or Equal)
  Διακλάδωση προγράμματος αν η προηγούμενη σύγκριση μεταξύ προσημασμένων αριθμών είχε αποτέλεσμα ο πρώτος να είναι μικρότερος του δεύτερου (SF <> OF)
- **JLE** ή **JNG** (Jump on Less or Equal/Not Greater)
  Διακλάδωση προγράμματος αν η προηγούμενη σύγκριση μεταξύ προσημασμένων αριθμών είχε αποτέλεσμα ο πρώτος να είναι μικρότερος ή ίσος του δεύτερου (SF <> OF ή ZF=1)
- **JNL** ή **JGE** (Jump on Not Less/Greater or Equal)
  Διακλάδωση προγράμματος αν η προηγούμενη σύγκριση μεταξύ προσημασμένων αριθμών είχε αποτέλεσμα ο πρώτος να είναι μεγαλύτερος ή ίσος του δεύτερου (SF = OF)
- **JNLE** ή **JG** (Jump on Not Less or Equal/Greater)
  Διακλάδωση προγράμματος αν η προηγούμενη σύγκριση μεταξύ προσημασμένων αριθμών είχε αποτέλεσμα ο πρώτος να είναι μεγαλύτερος του δεύτερου (ZF=0 και SF = OF)
- **JB** ή **JNAE** (Jump on Below/Not Above or Equal)
  Διακλάδωση προγράμματος αν η προηγούμενη σύγκριση μεταξύ μη προσημασμένων αριθμών είχε αποτέλεσμα ο πρώτος να είναι μικρότερος του δεύτερου (CF = 1)
- **JBE** ή **JNA** (Jump on Below or Equal/Not Above)
  Διακλάδωση προγράμματος αν η προηγούμενη σύγκριση μεταξύ μη προσημασμένων αριθμών είχε αποτέλεσμα ο πρώτος να είναι μικρότερος ή ίσος του δεύτερου (CF = 1 ή ZF = 1)
- **JNB** ή **JAE** (Jump on Not Below /Above or Equal)
  Διακλάδωση προγράμματος αν η προηγούμενη σύγκριση μεταξύ μη προσημασμένων αριθμών είχε αποτέλεσμα ο πρώτος να είναι μεγαλύτερος ή ίσος του δεύτερου (CF = 0)
- **JNBE** ή **JA** (Jump on Not Below or Equal/Above)

Διακλάδωση προγράμματος αν η προηγούμενη σύγκριση μεταξύ μη προσημασμένων αριθμών είχε αποτέλεσμα ο πρώτος να είναι μεγαλύτερος του δεύτερου (CF = 0 και ZF=0)

- **JP** ή **JPE** (Jump on Parity/Parity Even)
  Διακλάδωση προγράμματος αν το αποτέλεσμα της προηγούμενης πράξης έδωσε αριθμό με ζυγό αριθμό μονάδων (PF=1).
- **JNP** ή **JPO** (Jump on Not Parity/Parity Odd)
  Διακλάδωση προγράμματος αν το αποτέλεσμα της προηγούμενης πράξης έδωσε αριθμό με μονό αριθμό μονάδων (PF=0).
- **JO** (Jump on Overflow)
  Διακλάδωση προγράμματος αν κατά την προηγούμενη πράξη δημιουργήθηκε υπερχείλιση των προσημασμένων αριθμών (OF=1).
- **JNO** (Jump on Not Overflow)
  Διακλάδωση προγράμματος αν κατά την προηγούμενη πράξη δεν δημιουργήθηκε υπερχείλιση των προσημασμένων αριθμών (OF=0).
- **JS** (Jump on Sign)
  Διακλάδωση προγράμματος αν το αποτέλεσμα της προηγούμενης πράξης ήταν αρνητικός αριθμός (SF=1).
- **JNS** (Jump on Not Sign)
  Διακλάδωση προγράμματος αν το αποτέλεσμα της προηγούμενης πράξης ήταν θετικός αριθμός (SF=0).
- **JCXZ** (Jump on CX Zero)
  Διακλάδωση προγράμματος αν η τιμή του καταχωρητή CX είναι 0 (CX=0).

Η εντολή CMP και οι τρόποι σύνταξής της

(Compare – Σύγκριση δύο αριθμών)

Συγκρίνει δύο αριθμούς που μπορεί να βρίσκονται σε καταχωρητές, στη μνήμη ή να είναι σταθερά νούμερα και επηρεάζει τις σημαίες του καταχωρητή σημαιών. Η εντολή CMP στην ουσία εκτελεί αφαίρεση μεταξύ των δύο αριθμών, χωρίς όμως να αλλοιώνεται το περιεχόμενο κανενός καταχωρητή εκτός από τις σημαίες.

- CMP καταχωρητής1, καταχωρητής2 CMP AX,BX
  (σύγκρινε τα περιεχόμενα των καταχωρητών AX και BX επηρεάζοντας τις σημαίες.)
  Παρακάτω δίνονται παραδείγματα τιμών των AX,BX και σημαιών που προκύπτουν με την σύγκριση CMP :

A/A	AX	BX	Σημαίες
1	0002	0001	-
2	0004	0001	Parity
3	0001	0001	Zero, Parity
4	0001	0002	Carry, AuxCarry, Negative, Parity
5	0000	0002	Carry, AuxCarry, Negative

- CMP καταχωρητής, [θέση μνήμης] CMP AX,[200]
  Συγκρίνει το περιεχόμενο του καταχωρητή και το περιεχόμενο της θέσης μνήμης (2 byte), επηρεάζοντας τις σημαίες, κάνοντας εικονική αφαίρεση (καταχωρητής-μνήμη). Αν ο καταχωρητής είναι του ενός byte, π.χ. AL, τότε η σύγκριση γίνεται με 1 byte από τη μνήμη.
- CMP [θέση μνήμης], καταχωρητής CMP [200],AX
  (όπως η προηγούμενη)
- CMP καταχωρητης, τιμή CMP DX, 1AF4
  Συγκρινει το περιεχόμενο του καταχωρητή με την τιμή που δίνουμε, επηρεάζοντας τις σημαίες. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX,BX,CX,DX,BP,SI,DI,SP και όχι για καταχωρητές τμημάτων και σημαίες.
- CMP BY/WO[διεύθυνση μνήμης], τιμή CMP BY[200], 7A

Συγκρίνει το περιεχόμενο της διεύθυνσης με την τιμή που δώσαμε, επηρεάζοντας τις σημαίες.

Η εντολή JMP και οι τρόποι σύνταξής της

(Jump – Σύγκριση δύο αριθμών)

Μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται στην <διεύθυνση μνήμης> που δόθηκε, χωρίς συνθήκη (unconditional jump).

JMP <διεύθυνση μνήμης> JMP 0300

Μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται στην <διεύθυνση μνήμης> που δόθηκε, η οποία ανήκει στο ίδιο Τμήμα Κώδικα (code segment)

JMP καταχωρητής JMP BX

Μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται στην διεύθυνση μνήμης που υπάρχει σαν περιεχόμενο του <καταχωρητή> που δόθηκε, η οποία ανήκει στο ίδιο Τμήμα Κώδικα (code segment)

JMP [διεύθυνση μνήμης] ή JMP NE [διεύθυνση μνήμης] JMP [0300]

Διαβάζει από την [διεύθυνση μνήμης] 2 byte (low-high) τα οποία σχηματίζουν μία διεύθυνση και μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται σε αυτή τη διεύθυνση, η οποία ανήκει στο ίδιο Τμήμα Κώδικα (code segment)

JMP <segment:offset> JMP F000:0100

Μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται στην διεύθυνση μνήμης <segment:offset> που δόθηκε, η οποία μπορεί να είναι οπουδήποτε στη μνήμη

JMP FAR [διεύθυνση μνήμης] JMP FAR [0300]

Διαβάζει από την [διεύθυνση μνήμης] 4 byte (offset low, offset high, segment low, segment high) τα οποία σχηματίζουν μία διεύθυνση και μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται σε αυτή τη διεύθυνση, η οποία μπορεί να βρίσκεται οπουδήποτε στη μνήμη.

Η εντολές Jxx και οι τρόποι σύνταξής τους

(Jxx – Διακλάδωση υπό συνθήκη)

Οι εντολές αυτές είναι στην ουσία 29 διαφορετικές εντολές (μερικές έχουν διπλή σύνταξη). Κάθε μία από αυτές ελέγχει διαφορετικά bits του καταχωρητή σημαιών και αν έχουν τις κατάλληλες τιμές τότε μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται στην <διεύθυνση μνήμης> που δόθηκε (conditional jump).

Jxx <διεύθυνση μνήμης> JNE 0300

Προσοχή, η <διεύθυνση μνήμης> θα πρέπει να είναι κοντινή και στο διάστημα –128..+127 bytes από την αρχή της επόμενης εντολής από την Jxx. Αυτό γίνεται διότι οι εντολές Jxx παρόλο που συντάσσονται με διεύθυνση των 16 bit, όταν μεταφραστούν από την συμβολική μορφή σε κώδικα μηχανής, η <διεύθυνση μνήμης> μετατρέπεται σε <σχετική μετατόπιση> η οποία μάλιστα είναι του 1 byte. Η σύνταξη της εντολής με 16μπιτη διεύθυνση γίνεται μόνο για διευκόλυνση του προγραμματιστή. Γι αυτό και όλες οι εντολές Jxx έχουν ως παράμετρο **Σχετική Μετατόπιση (Relative Offset Operand)**. Η παράμετρος είναι ένας προσημασμένος αριθμός των 8 bit (-128 .. +127) η οποία προστίθεται στην διεύθυνση της επόμενης εντολής για να μας δώσει την απόλυτη διεύθυνση μεταφοράς της εκτέλεσης του προγράμματος. Η τελική διεύθυνση πρέπει να βρίσκεται μέσα στο ίδιο Τμήμα Κώδικα (code segment)

Παράδειγμα : …

```
0100:000C  SUB AX,BX     29 D8
0100:000E  JNE 0020      75 10        ( IF (ZF=0) IP=0020 )
0100:0010  MOV …         …
```

Η εντολή JNE 0020 αντιστοιχεί στον κώδικα μηχανής 75 10 όπου (75) είναι το opcode και (10) η παράμετρος. Το (10) είναι η σχετική μετατόπιση από την αρχή της επόμενης εντολής (0010 + 10 = 0020).

Υλοποίηση κοινών δομών της C με Assembly 8088.

Παρακάτω παραθέτουμε την υλοποίηση σε Assembly 8088 κοινών δομών ελέγχου της C

Α/Α	Δομή της C	Κώδικας Assembly 8088	Παρατηρήσεις

1	Εντολή πριν ;
	if (συνθήκη)
	{
	Εντολή 1;
	Εντολή 2;
	...
	Εντολή ν;
	}
	Εντολή μετά ;
		Εντολή πριν ;
		CMP <παράμετροι>
		Jxx <διεύθυνση>
		Εντολή 1
		Εντολή 2
		...
		Εντολή ν
		<διεύθυνση> Εντολή μετά
			Η εντολή Jxx είναι αντίθετη από την (συνθήκη) της C. Π.χ. για την συνθήκη :
			if (ax==0)
			η εντολή θα είναι :
			JNZ

2	Εντολή πριν ;
	if (συνθήκη)
	{ Εντολή 1;
	...
	Εντολή ν;}
	else
	{ Εντολή ν+1;
	...
	Εντολή μ;}
	Εντολή μετά ;
		Εντολή πριν ;
		CMP <παράμετροι>
		Jxx <διεύθυνση>
		Εντολή ν+1(περίπτωση else)
		...
		Εντολή μ
		JMP <συνέχεια>
		<διεύθυνση> Εντολή 1 (περίπτωση if)
		...
		Εντολή ν
		<συνέχεια> Εντολή μετά
			Η εντολή Jxx είναι όμοια με την (συνθήκη) της C. Π.χ. για την συνθήκη :
			if (ax==0)
			η εντολή θα είναι :
			JZ

	Δεύτερη υλοποίηση :	Εντολή πριν ;
		CMP <παράμετροι>
		Jxx <διεύθυνση>
		Εντολή 1 (περίπτωση if)
		...
		Εντολή ν
		JMP <συνέχεια>
		<διεύθυνση> Εντολή ν+1(περίπτωση else)
		...
		Εντολή μ
		<συνέχεια> Εντολή μετά
			Η εντολή Jxx είναι αντίθετη από την (συνθήκη) της C. Π.χ. για την συνθήκη :
			if (ax>=5)
			η εντολές θα είναι :
			CMP AX,05
			JB (Jump on Below)

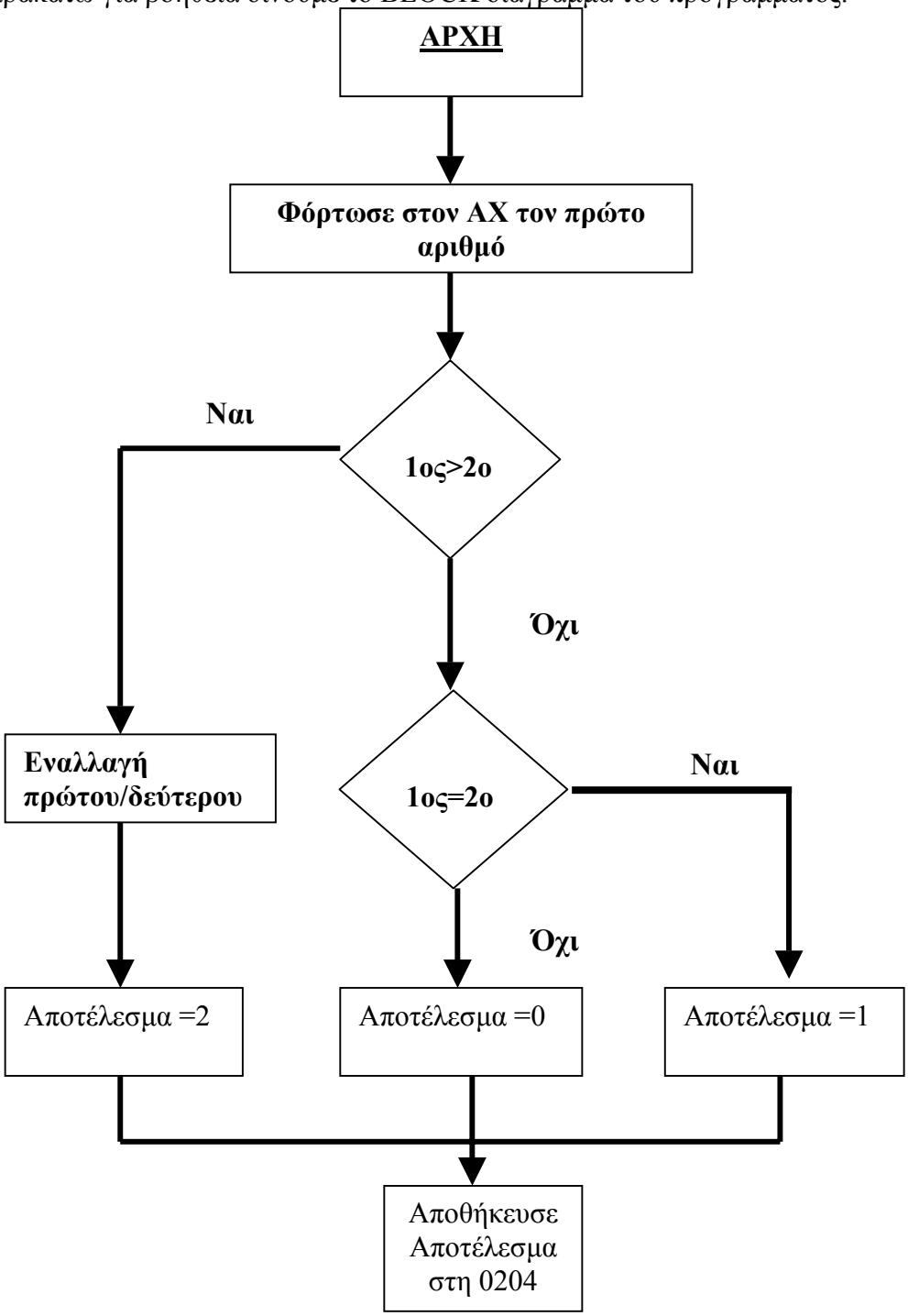
3	Εντολή πριν ;
	switch (k)
	{
	case 1: Εντολή 1;
	break;
	case 2 : Εντολή 2;
	break;
	...
	case n : Εντολή ν;
	break;
	default : Εντολή def
	}
	Εντολή μετά ;
		Εντολή πριν ;
		CMP AX,01
		Jxx <διεύθυνση1>
		CMP AX,02
		Jxx <διεύθυνση2>
		…
		CMP AX,0n
		Jxx <διεύθυνση n>
		Εντολή default
		JMP <συνέχεια>
		<διεύθυνση1> Εντολή 1
		JMP <συνέχεια>
		<διεύθυνση2> Εντολή 2
		JMP <συνέχεια>
		...
		<διεύθ. n> Εντολή ν
		<συνέχεια> Εντολή μετά
			Για κάθε περίπτωση της εντολής switch χρειάζεται και μία CMP σε assembly. Εδώ υποθέτουμε ότι η μεταβλητή (k) είναι φορτωμένη στον AX register. Η (Εντολή 1) όπως και η 2 κ.λ.π. μπορεί να αποτελούν-ται από ολόκληρα μπλόκ εντολών.

ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Άσκηση 4.1: Γράψτε ένα πρόγραμμα που θα ταξινομεί δύο **μη προσημασμένους** αριθμούς κατ' αύξουσα σειρά. Οι δύο αριθμοί περιέχονται στις θέσεις μνήμης **0200-0201** ο πρώτος και **0202-0203** ο δεύτερος. Εάν ο πρώτος αριθμός είναι μεγαλύτερος από τον δεύτερο τότε οι δύο αριθμοί θα εναλλάσσονται στη μνήμη. Εάν ο πρώτος είναι ίσος ή μικρότερος από τον δεύτερο θα παραμείνουν όπως είναι. Στο τέλος η θέση μνήμης **0204** να περιέχει μετά την εκτέλεση της σύγκρισης έναν από τους τρεις αριθμούς 0,1,2 όπως παρακάτω :

- 0 εάν ο πρώτος είναι μικρότερος από τον δεύτερο.
- 1 εάν οι δύο αριθμοί είναι ίσοι.
- 2 εάν ο πρώτος είναι μεγαλύτερος από τον δεύτερο.

Παρακάτω για βοήθεια δίνουμε το BLOCK διάγραμμα του προγράμματος.



Εκτελέστε το πρόγραμμα για τις παρακάτω τιμές και συμπληρώστε τον πίνακα για κάθε περίπτωση. Μετά την εκτέλεση της σύγκρισης να ελέγξετε επίσης και τις θέσεις μνήμης **0202-0204**, από τις οποίες οι 0200-0201 θα πρέπει να περιέχουν το μικρότερο αριθμό και οι 0202-0203 το μεγαλύτερο , ενώ η 0204 θα έχει το αποτέλεσμα της σύγκρισης.

0200-0201	**0202-0203**	**0204**
2345	1234	
AAAA	BBBB	
FFFF	FFFF	

Άσκηση 4.2: Γράψτε ένα πρόγραμμα που θα γράφει αυτόματα στη μνήμη και στις θέσεις από την 0200 και μετά τους αριθμούς 00, 01, 02, 03, ... με χρήση ενός απλού βρόχου επανάληψης. Κάθε αριθμός καταλαμβάνει 1 byte. Αρχικά εκτελέστε την επανάληψη 8 φορές. Στη συνέχεια μετατρέψτε το πρόγραμμα ώστε να εκτελείται τόσες φόρες όσο το περιεχόμενο του καταχωρητή CL. Τέλος μετατρέψτε το πρόγραμμα ώστε να εκτελείται τόσες φορές όσο η τιμή της μεταβλητής που βρίσκεται στη θέση 0300 (1 byte). Πώς μπορούμε να μετατρέψουμε το πρόγραμμα ώστε να λειτουργεί για πίνακα μεταβλητής διεύθυνσης αρχής ?

5ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΟ : Βρόχοι Επανάληψης (Loops)
Εντολές INC, DEC, LOOP, LOOPZ/LOOPE, LOOPNZ/LOOPNE
Ενδεικτική υλοποίηση βρόχων "for", "while", "do while", "repeat until", με εντολές Assembly 8088
Εκτέλεση προγραμμάτων με επαναληπτικούς βρόχους
Διαχείριση δεδομένων σε μορφή πίνακα

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Μετά τις δομές ελέγχου ροής προγράμματος οι επόμενες σημαντικές δομές προγραμματισμού είναι οι βρόχοι επανάληψης. Οι βρόχοι επανάληψης μας επιτρέπουν να συνθέσουμε προγράμματα που εκτελούν επαναληπτικές λειτουργίες για την υλοποίηση σύνθετων αλγορίθμων (π.χ. εύρεση παραγοντικού N!, αθροίσματα ή γινόμενα σειρών (ακολουθιών), επαναληπτικές αριθμητικές μέθοδοι, κ.λ.π.) ή προγράμματα που επενεργούν με τον ίδιο τρόπο σε μεγάλο πλήθος δεδομένων (π.χ. διαχείριση πινάκων, αρχείων, κ.λ.π.). Ένας βρόχος δεν είναι τίποτα άλλο από την υπό-συνθήκη μεταφορά της εκτέλεσης του προγράμματος σε προηγούμενη διεύθυνση. Συνήθως πριν από τον βρόχο θέτουμε σε μία μεταβλητή (που λέγεται «μεταβλητή ελέγχου του βρόχου» ή "loop control variable") μία αρχική τιμή και μέσα στο σώμα του βρόχου μεταβάλλουμε την μεταβλητή αυτή (την αυξάνουμε ή την μειώνουμε π.χ. κατά 1). Στο τέλος του βρόχου ελέγχουμε αν η μεταβλητή έχει φτάσει στο όριο επαναλήψεων. Π.χ.

```
0100:0000    MOV AL, 00
0100:0002    πρώτη εντολή του βρόχου ...
0100:0005    δεύτερη εντολή του βρόχου ...
0100:0009    …
0100:000B    INC AL
0100:000C    CMP AL, 08
0100:000E    JNE 0002
0100:0010    INT 3
```

Βέβαια είναι δυνατό να κατασκευάσουμε βρόχους και χωρίς "loop control variables", Οι οποίοι απλά ελέγχουν μία συνθήκη για το άν θα επαναληφθούν ξανά ή θα τερματίσουν (π.χ. ανάγνωση αρχείου με while (!feof(fp)) {…}).

Έτσι λοιπόν έχοντας εντολές ελέγχου ροής προγράμματος μπορούμε να υλοποιήσουμε και βρόχους επανάληψης. Ωστόσο στην Assembly του 8088 υπάρχουν και ειδικές εντολές για υλοποίηση βρόχων επανάληψης που διευκολύνουν σημαντικά τον προγραμματιστή. Αυτές είναι :

Χρησιμοποιούμενες εντολές:

- **INC** (Increment – Αύξηση ενός καταχωρητή κατά 1)
- **DEC** (Decrement – Μείωση ενός καταχωρητή κατά 1)
- **LOOP** (Loop – Βρόχος με απαριθμητή τον CX)
- **LOOPZ/LOOPE** (Loop while Zero/Equal - Βρόχος με απαριθμητή τον CX και συνθήκη ισότητας)

- **LOOPNZ/LOOPNE** (Loop while Not Zero/ Not Equal - Βρόχος με απαριθμητή τον CX και συνθήκη ανισότητας)

Η εντολή INC και οι τρόποι σύνταξής της
(Increment – Αύξηση ενός καταχωρητή κατά 1)
Αυξάνει την τιμή του καταχωρητή κατά 1. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX,BX,CX,DX,BP,SI,DI,SP και όχι για καταχωρητές τμημάτων και σημαίες
- INC καταχωρητής INC CX

Η εντολή DEC και οι τρόποι σύνταξής της
(Decrement – Μείωση ενός καταχωρητή κατά 1)
Μειώνει την τιμή του καταχωρητή κατά 1. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX,BX,CX,DX,BP,SI,DI,SP και όχι για καταχωρητές τμημάτων και σημαίες
- DEC καταχωρητής DEC DI

Η εντολή LOOP και οι τρόποι σύνταξής της
(Loop – Βρόχος με απαριθμητή τον CX)
Μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται στη διεύθυνση που δίνουμε, τόσες φορές όσες η τιμή του καταχωρητή CX. Σε κάθε επανάληψη μειώνεται ο CX κατά 1 και ελέγχεται αν έφτασε στο 0. Αν δεν έχει φτάσει τότε πηγαίνει πάλι στη διεύθυνση, που υποτίθεται ότι πρέπει να δείχνει την αρχή του βρόχου. Αν ο CX βρεθεί ίσος με 0, τότε ο βρόχος σταματά και το πρόγραμμα συνεχίζεται στην επόμενη εντολή μετά την LOOP. Προσοχή, η διεύθυνση θα πρέπει να είναι κοντινή και στο διάστημα –128..+127 bytes από την αρχή της επόμενης εντολής.
- LOOP <διεύθυνση> LOOP 0005

Η εντολή LOOPZ/LOOPE και οι τρόποι σύνταξής της
(Loop while Zero/Equal – Βρόχος με απαριθμητή τον CX και συνθήκη ισότητας)
Μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται στη διεύθυνση που δίνουμε, τόσες φορές όσες η τιμή του καταχωρητή CX και εφόσον η σημαία μηδενός ZF έχει ανάψει (ZF=1). Σε κάθε επανάληψη μειώνεται ο CX κατά 1 και ελέγχεται αν έφτασε στο 0. Αν δεν έχει φτάσει τότε πηγαίνει πάλι στη διεύθυνση, που υποτίθεται ότι πρέπει να δείχνει την αρχή του βρόχου. Αν ο CX βρεθεί ίσος με 0, ή η σημαία μηδενός δεν είναι αναμμένη (ZF=0) τότε ο βρόχος σταματά και το πρόγραμμα συνεχίζεται στην επόμενη εντολή μετά την LOOP. Προσοχή, η διεύθυνση θα πρέπει να είναι κοντινή και στο διάστημα –128..+127 bytes από την αρχή της επόμενης εντολής.
- LOOPZ <διεύθυνση> ή LOOPE <διεύθυνση> LOOPZ 001F

Η εντολή LOOPNZ/LOOPNE και οι τρόποι σύνταξής της
(Loop while Not Zero/Not Equal – Βρόχος με απαριθμητή τον CX και συνθήκη ανισότητας)
Μεταφέρει την εκτέλεση του προγράμματος στην εντολή που βρίσκεται στη διεύθυνση που δόθηκε, τόσες φορές όσες η τιμή του καταχωρητή CX και εφόσον η σημαία μηδενός ZF είναι σβηστή (ZF=0). Σε κάθε επανάληψη μειώνεται ο CX κατά 1 και ελέγχεται αν έφτασε στο 0. Αν δεν έχει φτάσει τότε πηγαίνει πάλι στη διεύθυνση, που υποτίθεται ότι πρέπει να δείχνει την αρχή του βρόχου. Αν ο CX βρεθεί ίσος με 0, ή η σημαία μηδενός είναι αναμμένη (ZF=1) τότε ο βρόχος σταματά και το πρόγραμμα συνεχίζεται στην επόμενη εντολή μετά την LOOP. Προσοχή, η διεύθυνση θα πρέπει να είναι κοντινή και στο διάστημα –128..+127 bytes από την αρχή της επόμενης εντολής
- LOOPNZ <διεύθυνση> ή LOOPNE <διεύθυνση> LOOPNE 002A

Υλοποίηση κοινών βρόχων της C με Assembly 8088.
Παρακάτω παραθέτουμε την υλοποίηση σε Assembly 8088 κοινών βρόχων επανάληψης της C :

Α/Α	**Βρόχος της C**	**Κώδικας Assembly 8088**	**Παρατηρήσεις**

1
 	
for με αύξηση
	Εντολή πριν ;
	for (i=αρχ; i<τελ; i++)
	{
	Εντολή 1;
	Εντολή 2;
	...
	Εντολή ν;
	}
	Εντολή μετά ;
		Εντολή πριν ;
		MOV CX, αρχ
		<διεύθυνση> Εντολή 1
		Εντολή 2
		...
		Εντολή ν
		INC CX
		CMP CX, τελ
		JNE <διεύθυνση>
		Εντολή μετά
			Η τιμή «αρχ» είναι η αρχική τιμή για το i (CX) και η «τελ» είναι η τελική τιμή. Αντί για τον CX μπορεί να χρησιμοποιηθεί οποιοσδήποτε καταχωρητής δεδομένων 8 ή 16 bit.

	Δεύτερη υλοποίηση που ελέγχει και την περίπτωση μή εκτέλεσης :	Εντολή πριν ; MOV CX, αρχ <διεύθυνση> CMP CX, τελ JAE <τέλος> Εντολή 1 Εντολή 2 ... Εντολή ν INC CX JMP <διεύθυνση> <τέλος> Εντολή μετά	Αν ο μετρητής CX είναι με πρόσημο θα πρέπει αντί για JAE να χρησιμοποιήσουμε την JGE.

2
 	
for με μείωση
	Εντολή πριν ;
	for (i=αρχ; i>τελ; i--)
	{
	Εντολή 1;
	Εντολή 2;
	...
	Εντολή ν;
	}
	Εντολή μετά ;
		Εντολή πριν ;
		MOV CX, αρχ
		<διεύθυνση> Εντολή 1
		Εντολή 2
		...
		Εντολή ν
		DEC CX
		CMP CX, τελ
		JNE <διεύθυνση>
		Εντολή μετά
			Η τιμή «αρχ» είναι η αρχική τιμή για το i (CX) και η «τελ» είναι η τελική τιμή. Αντί για τον CX μπορεί να χρησιμοποιηθεί οποιοσδήποτε καταχωρητής δεδομένων 8 ή 16 bit.

	Δεύτερη υλοποίηση στην περίπτωση που η τελική τιμή «τελ» είναι 0 :	Εντολή πριν ; MOV CX, αρχ <διεύθυνση> Εντολή 1 Εντολή 2 ... Εντολή ν LOOP <διεύθυνση> Εντολή μετά	Η εντολή LOOP συνεργάζεται μόνο με τον καταχωρητή CX
	Τρίτη υλοποίηση που ελέγχει και την περίπτωση μη εκτέλεσης :	Εντολή πριν ; MOV CX, αρχ <διεύθυνση> CMP CX, τελ JBE <τέλος> Εντολή 1 Εντολή 2 ... Εντολή ν DEC CX JMP <διεύθυνση> <τέλος> Εντολή μετά	Αν ο μετρητής CX είναι με πρόσημο θα πρέπει αντί για JBE να χρησιμοποιήσουμε την JLE.

3
 	
while
	Εντολή πριν ;
	while (συνθήκη)
	{
	Εντολή 1;
	Εντολή 2;
	...
	Εντολή ν;
	}
	Εντολή μετά ;
		
		<διεύθυνση>
		
		<τέλος>
			Εντολή πριν ;
			CMP <παράμετροι>
			Jxx <τέλος>
			Εντολή 1
			Εντολή 2
			...
			Εντολή ν
			JMP <διεύθυνση>
			Εντολή μετά
				Η συνθήκη βάση της οποίας εκτελείται η Jxx είναι ανάποδη (αν δεν ισχύει η συνθήκη τότε τερματίζει ο βρόχος)

4 do while	Εντολή πριν ; do { Εντολή 1; Εντολή 2; ... Εντολή ν; } while (συνθήκη) Εντολή μετά ;	 <διεύθυνση>	Εντολή πριν ; Εντολή 1 Εντολή 2 ... Εντολή ν CMP <παράμετροι> Jxx <διεύθυνση> Εντολή μετά	Η συνθήκη βάση της οποίας εκτελείται η Jxx είναι κανονική (αν ισχύει η συνθήκη τότε συνεχίζει ο βρόχος)
5 repeat until (pascal)	Εντολή πριν ; repeat Εντολή 1; Εντολή 2; ... Εντολή ν; until (συνθήκη) Εντολή μετά ;	 <διεύθυνση>	Εντολή πριν ; Εντολή 1 Εντολή 2 ... Εντολή ν CMP <παράμετροι> Jxx <διεύθυνση> Εντολή μετά	Η συνθήκη βάση της οποίας εκτελείται η Jxx είναι ανάποδη (αν δεν ισχύει η συνθήκη τότε συνεχίζει ο βρόχος)

Διαχείριση Πινάκων δεδομένων με Δεικτοδοτούμενη Διευθυνσιοδότηση

Οι πίνακες αποτελούν δομές δεδομένων που αποτελούνται από ομοειδή στοιχεία σε σειριακή διάταξη. Τα στοιχεία ενός πίνακα προσπελαύνονται με την βοήθεια ενός «δείκτη» της θέσης του πίνακα. Π.χ. int A[100]; for (i=0 ; i<100 ; i++) A[i]=0;

Για την προσπέλαση πινάκων η Assembly του 8088 μας παρέχει έναν αριθμό από τρόπους σύνταξης των εντολών, όπου δίνουμε την βάση του πίνακα και την μετατόπιση. Έτσι οι περισσότερες εντολές του 8088 μπορούν να συντάξουν διευθύνσεις με την βοήθεια και συγκεκριμένων καταχωρητών που παίζουν τον ρόλο των δεικτών. Έτσι όταν συντάσσουμε μία διεύθυνση μνήμης με τη βοήθεια και ενός καταχωρητή, τότε το περιεχόμενο του καταχωρητή προστίθεται στην διεύθυνση μνήμης που δώσαμε, ώστε να προκύψει η τελική διεύθυνση. Οι καταχωρητές που μπορούν να συμμετέχουν στον προσδιορισμό διεύθυνσης ως δείκτες είναι οι BX, SI, DI, BP, οι οποίοι μπορεί να συνταχθούν και σε συνδυασμούς, δίνοντας έτσι 24 διαφορετικούς τρόπους σύνταξης διευθύνσεων μνήμης. Οι τρόποι αυτοί είναι οι εξής :

Τρόποι διευθυνσιοδότησης μνήμης

1) Διευθυνσιοδότηση [ADDR16] Παράδειγμα : MOV AX, [0200]
2) Διευθυνσιοδότηση [BP+ADDR8] Παράδειγμα : MOV AX, [BP+27]
3) Διευθυνσιοδότηση [BP+ADDR16] Παράδειγμα : MOV AX, [BP+8765]
4) Διευθυνσιοδότηση [BP+SI] Παράδειγμα : MOV AX, [BP+SI]
5) Διευθυνσιοδότηση [BP+SI+ADDR8] Παράδειγμα : MOV AX, [BP+SI+4D]
6) Διευθυνσιοδότηση [BP+SI+ADDR16] Παράδειγμα : MOV AX, [BP+SI+900A]
7) Διευθυνσιοδότηση [BP+DI] Παράδειγμα : MOV AX, [BP+DI]
8) Διευθυνσιοδότηση [BP+DI+ADDR8] Παράδειγμα : MOV AX, [BP+DI+77]
9) Διευθυνσιοδότηση [BP+DI+ADDR16] Παράδειγμα : MOV AX, [BP+DI+11EE]

10) Διευθυνσιοδότηση [BX] Παράδειγμα : MOV AX, [BX]
11) Διευθυνσιοδότηση [BX+ADDR8] Παράδειγμα : MOV AX, [BX+2B]
12) Διευθυνσιοδότηση [BX+ADDR16] Παράδειγμα : MOV AX, [BX+1234]
13) Διευθυνσιοδότηση [BX+SI] Παράδειγμα : MOV AX, [BX+SI]
14) Διευθυνσιοδότηση [BX+DI] Παράδειγμα : MOV AX, [BX+DI]
15) Διευθυνσιοδότηση [BX+SI+ADDR8] Παράδειγμα : MOV AX, [BX+SI+A8]
16) Διευθυνσιοδότηση [BX+DI+ADDR8] Παράδειγμα : MOV AX, [BX+DI+59]
17) Διευθυνσιοδότηση [BX+SI+ADDR16] Παράδειγμα : MOV AX, [BX+SI+4C7E]
18) Διευθυνσιοδότηση [BX+DI+ADDR16] Παράδειγμα : MOV AX, [BX+DI+91DE]
19) Διευθυνσιοδότηση [SI] Παράδειγμα : MOV AX, [SI]
20) Διευθυνσιοδότηση [SI+ADDR8] Παράδειγμα : MOV AX, [SI+23]
21) Διευθυνσιοδότηση [SI+ADDR16] Παράδειγμα : MOV AX, [SI+FEDC]
22) Διευθυνσιοδότηση [DI] Παράδειγμα : MOV AX, [DI]
23) Διευθυνσιοδότηση [DI+ADDR8] Παράδειγμα : MOV AX, [DI+23]
24) Διευθυνσιοδότηση [DI+ADDR16] Παράδειγμα : MOV AX, [DI+2552]

Σε μία εντολή π.χ. MOV AX, [DI+0200], η διεύθυνση 0200 είναι η «βάση» του πίνακα (η διεύθυνση του πρώτου στοιχείου) και ο DI είναι ο δείκτης (όπως το i) που μας δίνει την μετατόπιση πάνω από την βάση. Έτσι σε ένα βρόχο μπορούμε να αυξάνουμε τον DI και με την εντολή MOV να φορτώνουμε στον AX διαδοχικά στοιχεία του πίνακα. Για παράδειγμα το παρακάτω πρόγραμμα προσθέτει τα 5 στοιχεία ενός πίνακα 8μπιτων αριθμών που ξεκινούν από την διεύθυνση 0200 :

```
0100:0000   MOV SI, 0000
0100:0003   MOV AX,SI
0100:0005   ADD AL,[0200+SI]
0100:0009   INC SI
0100:000A   CMP SI, 0005
0100:000D   JNE 0005
0100:000F   INT 3
```

Με συντάξεις με 2 καταχωρητές δείκτη π.χ. MOV AX,[BX+DI] ή MOV AX,[BX+DI+91DE] μπορούμε να υλοποιήσουμε διδιάστατους πίνακες ή πίνακες μεταβλητής διεύθυνσης αρχής.

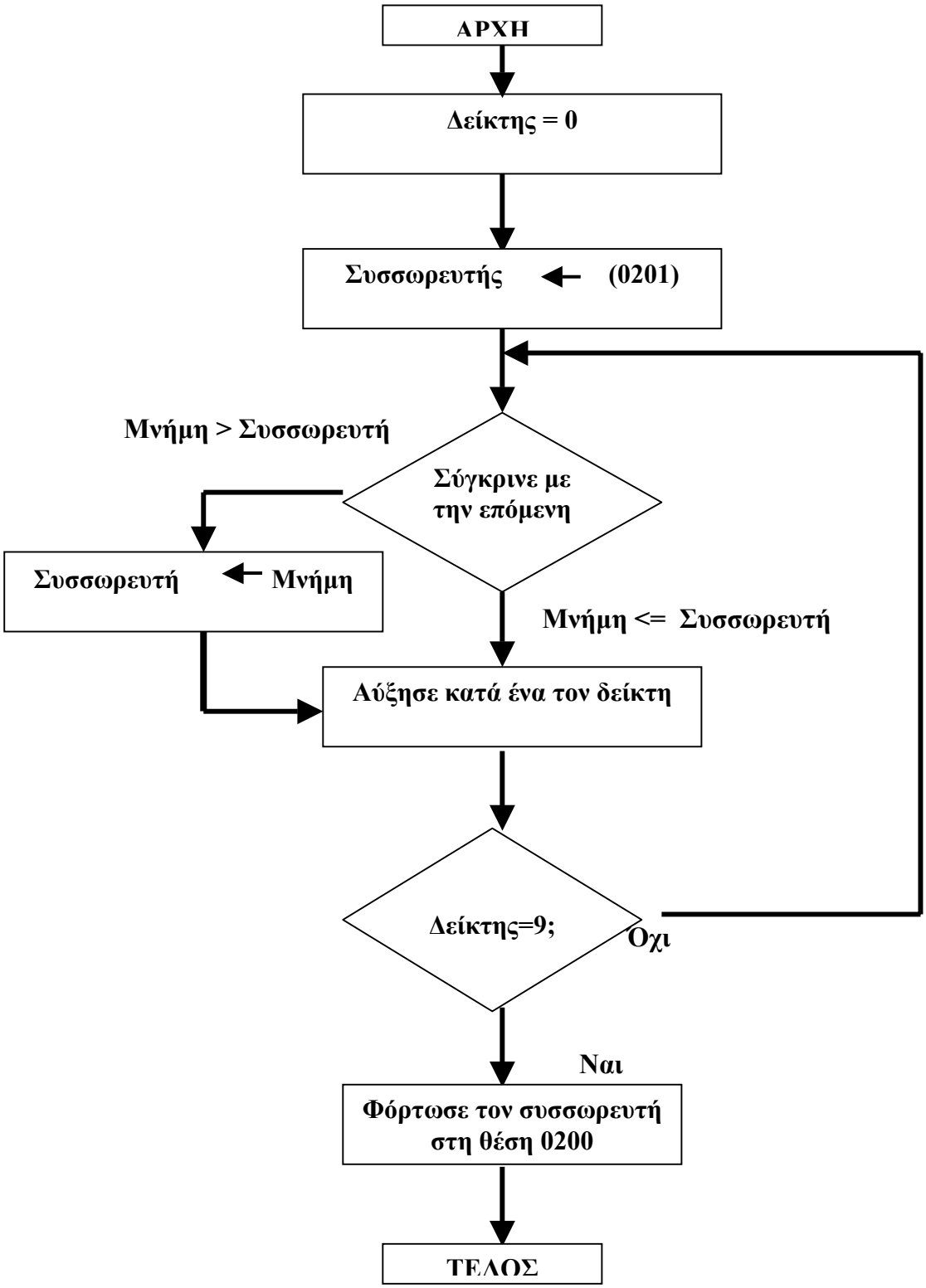
ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Άσκηση 5.1: Γράψτε το πρόγραμμα που βρίσκει τον μεγαλύτερο 8-bit μη προσημασμένο αριθμό από δέκα αριθμούς που βρίσκονται στις θέσεις μνήμης από **0201** έως και **020A** και βάλτε τον στη θέση μνήμης **0200**. Ο πίνακας των αριθμών δίνεται παρακάτω.

Θέση Μνήμης	**Περιεχόμενα**
0201	A2
0202	00
0203	40
0204	40
0205	0B
0206	5A
0207	FA
0208	55
0209	06
020A	4F

Τι θα περιέχει η θέση μνήμης **20** μετά την εκτέλεση του προγράμματος;

Στο Block διάγραμμα που υπάρχει παρακάτω χρησιμοποιούμε την θέση μνήμης **0200** για να αποθηκεύσουμε τελικά τον μεγαλύτερο αριθμό απ' όλους.



Ανάλυση : Εάν έχουμε μια ομάδα αριθμών σε κάποιες θέσεις μνήμης (έναν σε κάθε θέση) τότε για να βρούμε τον μεγαλύτερο απ' αυτούς θα κάνουμε διαδοχικές συγκρίσεις ανά δύο των αριθμών, αρχίζοντας απ' τον πρώτο με τον δεύτερο, βρίσκοντας κάθε φορά τον μεγαλύτερο, τον οποίον θα συγκρίνουμε με τον επόμενο στη σειρά και θα βρούμε τον μεγαλύτερο, τον οποίο θα συγκρίνουμε

με τον επόμενο στη σειρά κ.ο.κ. Κάθε φορά, σε κάθε σύγκριση, τον μεγαλύτερο που θα βρίσκουμε θα τον καταχωρούμε σ' έναν καταχωρητή (π.χ. τον συσσωρευτή) ούτως ώστε στο τέλος να υπάρχει ο μεγαλύτερος απ' όλους τους αριθμούς στο συγκεκριμένο καταχωρητή.

Πριν γράψουμε το πρόγραμμα πρέπει να γνωρίζουμε πόσοι είναι οι αριθμοί που θα σαρώσουμε (ψάξουμε). Έτσι στην αρχή του προγράμματος θα θέσουμε σε ένα δείκτη (έστω τον BX) τον αριθμό 0 και μετά από κάθε σύγκριση θα αυξάνουμε κατά 1 τον δείκτη αυτό και αμέσως θα συγκρίνουμε με τον αριθμό που φανερώνει το πλήθος των αριθμών μας, που για το παράδειγμά μας είναι 10_{10} ($0A_{16}$). Εάν ο δείκτης είναι ίσος μ' αυτόν τότε σημαίνει πως έχουν γίνει όλες οι συγκρίσεις και σταματάει το πρόγραμμα. Οι αριθμοί αρχίζουν από την θέση μνήμης **0201**.
Η λογική του προγράμματος έχει ως εξής:

- Θέτουμε τον δείκτη BX= 0
- Βάζουμε στον συσσωρευτή τον πρώτο αριθμό (απ' την πρώτη θέση 0201 της μνήμης).
- Συγκρίνουμε αυτόν με την επόμενη θέση μνήμης
- Είναι μεγαλύτερη η μνήμη από τον συσσωρευτή; Αν **ΝΑΙ** βάζουμε στο συσσωρευτή το περιεχόμενο της συγκεκριμένης θέσης μνήμης και αυξάνουμε έπειτα τον δείκτη κατά 1. Αν **ΟΧΙ** αυξάνουμε κατ' ευθείαν το δείκτη κατά 1.
- Συγκρίνουμε τον δείκτη με το 9 για να διαπιστώσουμε αν έγιναν οι 9 επαναλήψεις (9 διότι το $1^{ο}$ στοιχείο το πήραμε εξ'αρχής). Το έχει φτάσει; Αν **ΟΧΙ** επιστρέφουμε στην εντολή σύγκρισης για να συγκρίνουμε τον επόμενο αριθμό με αυτόν του συσσωρευτή (τον μεγαλύτερο από τους δυο προηγουμένους που συγκρίθηκαν). Αν **ΝΑΙ** βάζουμε το περιεχόμενο του συσσωρευτή σε κάποια θέση μνήμης (π.χ. 0200) και σταματάμε το πρόγραμμα.

Άσκηση 5.2: Να γραφεί πρόγραμμα που να προσθέτει, με χρήση βρόχου, τους αριθμούς 1+2+3+4+5 και να τοποθετεί το αποτέλεσμα στη διεύθυνση μνήμης **0200**.

Άσκηση 5.3: Να τροποποιήσετε το ανωτέρω πρόγραμμα ώστε να προσθέτει τους αριθμούς 1+2+…+N όπου το N να είναι μεταβλητή και να βρίσκεται στη θέση μνήμης **0200**. Το αποτέλεσμα να τοποθετείται στην διεύθυνση μνήμης **0201-0202**.

Άσκηση 5.4: Να γραφεί πρόγραμμα το οποίο θα υπολογίζει το πλήθος των αρνητικών αριθμών που περιέχονται σε μια περιοχή μνήμης, η οποία αρχίζει από τη διεύθυνση **0301**. Το σύνολο των θέσεων μνήμης είναι αποθηκευμένο στη διεύθυνση **0200**. Το πλήθος των αρνητικών αριθμών θα καταχωρηθεί στη θέση **0202**. Σαν παραλλαγή της άσκησης, μπορεί η διεύθυνση του πίνακα να είναι μεταβλητή και να περιέχεται στην διεύθυνση μνήμης **0204.**

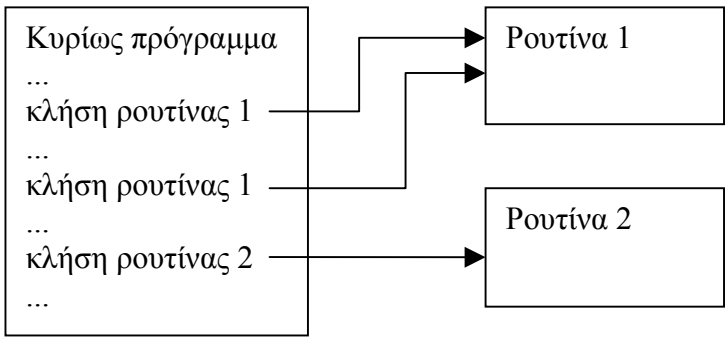
6ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΟ : Υπορουτίνες κλήση και επαναφορά
Εντολές CALL, RET, INT, IRET
Χρήση του Stack για πέρασμα παραμέτρων και αποτελεσμάτων
Εντολές PUSH, PUSHF, POP, POPF
Σύγκριση και Πρόσθεση Πινάκων Δεδομένων

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Οι υπορουτίνες αποτελούν αυτόνομα τμήματα κώδικα που διεκπεραιώνουν μία συγκεκριμένη εργασία και μπορούμε να τα καλούμε μέσα από το κυρίως πρόγραμμα όσες φορές χρειαστεί (π.χ. κώδικας που εναλλάσσει δύο θέσεις μνήμης, κώδικας που κάνει σύνθετες πράξεις, κώδικας πρόσθεσης αριθμών πολλών bytes, κ.λ.π.). Τα πλεονεκτήματα της χρήσης των υπορουτινών είναι :

- Συμβάλλουν στην μείωση του συνολικού όγκου του κώδικα σε περίπτωση ύπαρξης όμοιων επαναλαμβανόμενων τμημάτων
- Καθιστούν την διόρθωση πιο εύκολη γιατί διορθώνουμε τον κώδικα της υπορουτίνας μία φορά και η διόρθωση ισχύει για όλο το πρόγραμμα
- Κάνουν τον κώδικα πιο κατανοητό και δίνουν την δυνατότητα επαναχρησιμοποίησης του κώδικα.



Σχήμα 6.1 : Κλήση υπορουτινών μέσα από το κυρίως πρόγραμμα

Για την υλοποίηση των υπορουτινών σε προγράμματα γλώσσας μηχανής ο 8088 μας παρέχει τις εξής εντολές :

Χρησιμοποιούμενες εντολές:

- **CALL** (Call – Κλήση υπορουτίνας)
- **RET** (Return – Επιστροφή από υπορουτίνα)
- **INT** (Interrupt – Κλήση ρουτίνας διακοπής λογισμικού – software interrupt)
- **IRET** (Interrupt Return – Επιστροφή από ρουτίνα διακοπής)

Η εντολή CALL και οι τρόποι σύνταξής της
(Call – Κλήση υπορουτίνας)

Μεταφέρει την εκτέλεση του προγράμματος στην υπορουτίνα που βρίσκεται στην διεύθυνση μνήμης που δόθηκε ως παράμετρος. Ο κώδικας της ρουτίνας πρέπει να τερματίζει με RET έτσι ώστε μόλις τελειώσει η ρουτίνα να επιστρέψουμε στο σημείο του κυρίως προγράμματος από το οποίο κλήθηκε.

- CALL <διεύθυνση μνήμης> CALL 0500
  Μεταφέρει την εκτέλεση του προγράμματος στην υπορουτίνα που βρίσκεται στην <διεύθυνση μνήμης> που δόθηκε, η οποία ανήκει στο ίδιο Τμήμα Κώδικα (code segment) με την εντολή CALL που εκτελείται (Near Call Direct Relative - ο CS παραμένει ο ίδιος)
- CALL καταχωρητής CALL DX
  Μεταφέρει την εκτέλεση του προγράμματος στην υπορουτίνα που βρίσκεται στην διεύθυνση μνήμης που υπάρχει σαν περιεχόμενο του <καταχωρητή> που δόθηκε, η οποία ανήκει στο ίδιο Τμήμα Κώδικα (code segment) με την εντολή CALL που εκτελείται (Near Call Register Indirect Absolute - ο CS παραμένει ο ίδιος)
- CALL [διεύθυνση μνήμης] ή CALL NE [διεύθυνση μνήμης]
  (NE=NEAR) CALL NE[0900]
  Διαβάζει από την [διεύθυνση μνήμης] 2 byte (low-high) τα οποία σχηματίζουν μία διεύθυνση και μεταφέρει την εκτέλεση του προγράμματος στην υπορουτίνα που βρίσκεται σε αυτή τη διεύθυνση, η οποία ανήκει στο ίδιο Τμήμα Κώδικα (code segment) με την εντολή CALL που εκτελείται (Near Call Memory Indirect Absolute - ο CS παραμένει ο ίδιος)
- CALL <segment:offset> CALL F000:0100
  Μεταφέρει την εκτέλεση του προγράμματος στην υπορουτίνα που βρίσκεται στην διεύθυνση μνήμης <segment:offset> που δόθηκε, η οποία μπορεί να είναι οπουδήποτε στη μνήμη (Far Call Direct Absolute - ο CS αλλάζει)
- CALL FAR[διεύθυνση μνήμης] CALL FAR[0A00]
  Διαβάζει από την [διεύθυνση μνήμης] 4 byte (offset low, offset high, segment low, segment high) τα οποία σχηματίζουν μία διεύθυνση και μεταφέρει την εκτέλεση του προγράμματος στην υπορουτίνα που βρίσκεται σε αυτή τη διεύθυνση, η οποία μπορεί να βρίσκεται οπουδήποτε στη μνήμη (Far Call Memory Indirect Absolute - ο CS μεταβάλλεται).

Η εντολή RET και οι τρόποι σύνταξής της

(Return – Επιστροφή από υπορουτίνα)

Επιστρέφει στην επόμενη εντολή μετά την CALL η οποία κάλεσε την υπορουτίνα. Τοποθετείται ως τελευταία εντολή των υπορουτινών, ώστε μετά την εκτέλεσή τους να επιστρέφει η εκτέλεση στο κυρίως πρόγραμμα.

- RET

Η εντολή INT και οι τρόποι σύνταξής της

(Interrupt – Κλήση ρουτίνας διακοπής λογισμικού – software interrupt)

Παράγει μία κλήση προς την ρουτίνα χειρισμού της διακοπής της οποίας ο αριθμός δίνεται ως παράμετρος. Ο αριθμός της διακοπής μπορεί να είναι στην περιοχή 00..FF (0..255 στο δεκαδικό). Κάθε αριθμός διακοπής αντιστοιχεί σε μία διεύθυνση υπορουτίνας η οποία θα εκτελεστεί. Οι διευθύνσεις των ρουτινών χειρισμού των διακοπών είναι των 4ων bytes και είναι αποθηκευμένες στις πρώτες 1024 διευθύνσεις μνήμης (00000_{16} … $003FF_{16}$).

- INT <αριθμός διακοπής> INT 84
  Καλεί την ρουτίνα της διακοπής που αντιστοιχεί στον <αριθμό διακοπής> που δόθηκε (π.χ. 84), η διεύθυνση της οποίας βρίσκεται στην ν-οστή (π.χ. $84^{η}$) κατά σειρά 32άδα bytes από την διεύθυνση 00000 (δηλαδή για INT 84 στις θέσεις 0020C..0020F).
- INTO
  Αντιστοιχεί στην διακοπή με αριθμό 4 (Overflow Interrupt). Όταν κληθεί ελέγχει την τιμή της σημαίας υπερχείλισης (OF) και αν αυτή είναι 1 τότε εκτελεί την ρουτίνα διαχείρισης της υπερχείλισης.

- INT 3

  Η διακοπή αυτή χρησιμοποιείται έξυπνα από το πρόγραμμα MONITOR, κατά την εκτέλεση ενός προγράμματος, με καθορισμό διευθύνσεων παύσης (breakpoints). Μπορεί να χρησιμοποιηθεί ως η τελευταία εντολή ενός προγράμματος, καθώς μόλις εκτελείται, εμφανίζει τις τιμές των καταχωρητών και επιστρέφει στο MONITOR (trap to debugger).

Η εντολή IRET και οι τρόποι σύνταξής της

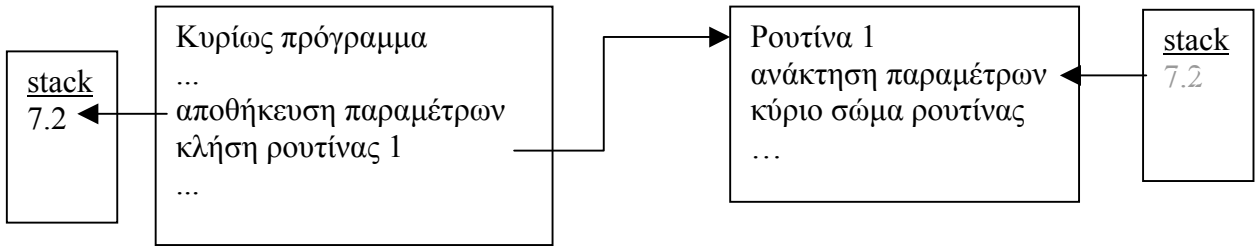
(Interrupt Return – Επιστροφή από υπορουτίνα διακοπής)

Τοποθετείται ως τελευταία εντολή μιας ρουτίνας εξυπηρέτησης διακοπής (interrupt handler routine), και επιστρέφει την εκτέλεση του προγράμματος στην επόμενη εντολή από αυτή που κάλεσε την διακοπή (INT).

- IRET

Χρήση του Σωρού (Stack)

Επειδή κατά την εκτέλεση του κώδικα της υπορουτίνας θα χρησιμοποιηθούν πιθανότατα οι καταχωρητές του επεξεργαστή (AX, BX, CX, … FG κ.λ.π.,) θα πρέπει να διασφαλίσουμε ότι οι τιμές των καταχωρητών κατά την επιστροφή από την υπορουτίνα (RET) θα πάρουν τις τιμές που είχαν λίγο πριν καλέσουμε την υπορουτίνα (CALL). Έτσι, δεν θα αλλοιωθεί η κανονική εξέλιξη του κυρίως προγράμματος από την εμβόλιμη εκτέλεση της υπορουτίνας. Για τον λόγο αυτό πριν την εκτέλεση της υπορουτίνας θα πρέπει να σώσουμε κάπου την κατάσταση των καταχωρητών και να την επαναφέρουμε αμέσως μετά το RET. Ο τόπος που σώζονται οι τιμές των καταχωρητών είναι ο **σωρός** (**stack**). Ο σωρός αποτελεί ένα τμήμα μνήμης στην βάση του οποίου δείχνει ο καταχωρητής SS (Stack Segment Register) και γεμίζει από πάνω προς τα κάτω με δομή LIFO (Last in First Out). Ο καταχωρητής SP (Stack Pointer) δείχνει πάντα την κορυφή του stack. Μία άλλη χρήση του Σωρού είναι η προσωρινή αποθήκευση των παραμέτρων που περνιούνται στις υπορουτίνες. Το κυρίως πρόγραμμα, πρίν καλέσει (CALL) μία υπορουτίνα η οποία δέχεται παραμέτρους, αποθηκεύει τις παραμέτρους στο stack. Όταν γίνει η κλήση τότε η υπορουτίνα «τραβάει» τις τιμές των παραμέτρων από το stack και προχωράει στους υπολογισμούς της. Αυτό φαίνεται στο σχήμα 6.2.



Σχήμα 6.2 : Αποθήκευση και ανάκτηση παραμέτρων υπορουτινών στο stack

Εντολές χειρισμού του Σωρού (Stack)

- **PUSH** (Push – Αποθήκευση στη στοίβα)
- **PUSHF** (Push Flags – Αποθήκευση σημαιών στη στοίβα)
- **POP** (Pop – Ανάκτηση από τη στοίβα)
- **POPF** (Pop Flags – Ανάκτηση Σημαιών από τη στοίβα)

Η εντολή PUSH και οι τρόποι σύνταξής της

(Push – Αποθήκευση στη στοίβα)

Αντιγράφει τα περιεχόμενα ενός καταχωρητή ή μίας θέσης μνήμης (2 bytes) στην στοίβα, ενημερώνοντας τον καταχωρητή SP.

- PUSH <καταχωρητής> PUSH ES

Αντιγράφει τα περιεχόμενα του καταχωρητή (2 bytes) στην στοίβα, ενημερώνοντας τον καταχωρητή SP. Ο καταχωρητής που η τιμή του πηγαίνει στην στοίβα, μπορεί να είναι και καταχωρητής τμήματος (segment register).

- PUSH [θέση μνήμης] PUSH [0700]

  Αντιγράφει τα περιεχόμενα της θέσης μνήμης και της επόμενης (2 bytes) στην στοίβα, ενημερώνοντας τον καταχωρητή SP.

Η εντολή PUSHF και οι τρόποι σύνταξής της

(Push Flags – Αποθήκευση σημαιών στη στοίβα)

Αντιγράφει τα περιεχόμενα του καταχωρητή σημαιών (Flags Register – FG) (2 bytes) στην στοίβα, ενημερώνοντας τον καταχωρητή SP.

- PUSHF

Η εντολή POP και οι τρόποι σύνταξής της

(Pop – Ανάκτηση από τη στοίβα)

Ανακτά από την στοίβα τα περιεχόμενα ενός καταχωρητή ή μίας θέσης μνήμης (2 bytes), ενημερώνοντας τον καταχωρητή SP.

- POP <καταχωρητής> POP AX

  Ανακτά από την στοίβα τα περιεχόμενα του καταχωρητή (2 bytes), ενημερώνοντας τον καταχωρητή SP. Ο καταχωρητής που η τιμή του ανακτάται από την στοίβα, μπορεί να είναι και καταχωρητής τμήματος (segment register).

- POP [θέση μνήμης] POP [0400]

  Ανακτά από την στοίβα τα περιεχόμενα μίας θέσης μνήμης, και της επόμενης (2 bytes), ενημερώνοντας τον καταχωρητή SP.

Η εντολή POPF και οι τρόποι σύνταξής της

(Pop Flags – Ανάκτηση σημαιών από τη στοίβα)

Ανακτά από την στοίβα τα περιεχόμενα του καταχωρητή σημαιών (Flags Register – FG) (2 bytes), ενημερώνοντας τον καταχωρητή SP.

- POPF

Έτσι αμέσως πριν την κλήση της υπορουτίνας θα πρέπει να καταχωρούνται στο stack όλοι οι καταχωρητές που πρόκειται να επηρεαστούν από την υπορουτίνα. Όταν η ρουτίνα επιστρέψει, θα πρέπει να γίνεται ανάκτηση των καταχωρητών που σώθηκαν εκεί, έτσι ώστε το stack να ξαναγυρίσει στην αρχική του κατάσταση. Παράδειγμα :

```
...(προηγούμενες εντολές)
PUSH AX          ;σώσιμο του AX
PUSH BX          ;σώσιμο του BX
…
PUSH ES          ; σώσιμο του ES
PUSHF            ; σώσιμο των σημαιών
CALL XXXX
POPF             ; ανάκτηση των σημαιών
POP ES           ; ανάκτηση του ES
…
POP BX           ; ανάκτηση του BX
POP AX           ; ανάκτηση του AX
...(επόμενες εντολές)
```

Παρατηρήστε ότι η ανάκτηση των καταχωρητών από τον σωρό γίνεται με ανάποδη σειρά από το σώσιμο καθώς ο σωρός έχει δομή **LIFO** (Last In First Out).

Παραδείγματα προγραμμάτων με χρήση υπορουτινών :

Παράδειγμα 1 (Η υπορουτίνα αλλοιώνει την τιμή του AX στο κυρίως πρόγραμμα)

```
0100:0000    MOV AX, FFFF
0100:0003    CALL 0200
0100:0006    INT 3

0100:0200    MOV AX,0001
0100:0203    RET
```

Παράδειγμα 2 (Η υπορουτίνα χρησιμοποιεί τον AX και μεταβάλλει τον FG αλλά με σώσιμο/ανάκτηση των καταχωρητών στο σωρό δεν επηρεάζεται το κυρίως πρόγραμμα)

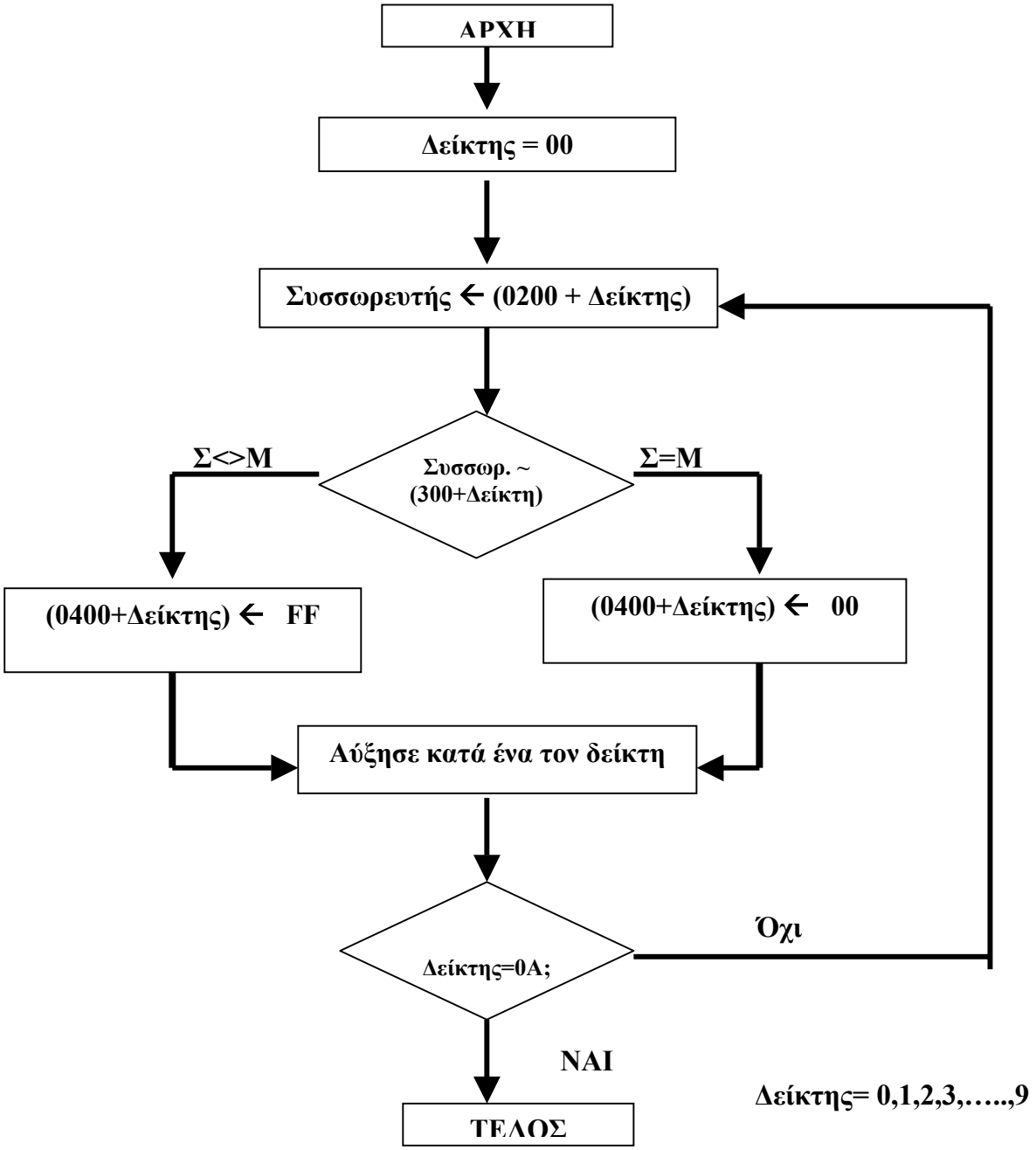
```
0100:0000    MOV AX,FFFF
0100:0003    CLC
0100:0004    PUSH AX
0100:0005    PUSHF
0100:0006    CALL 0200
0100:0009    POPF
0100:000A    POP AX
0100:000B    INT 3

0100:0200    MOV AX,CCCC
0100:0203    STC
0100:0204    RET
```

ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Άσκηση 6.1: Να γίνει πρόγραμμα που θα συγκρίνει τα περιεχόμενα δύο πινάκων 8μπιτων αριθμών που βρίσκονται στις θέσεις μνήμης **0200, 0201, 0202,….,0209** και **0300, 0301, 0302,…..0309** ανά δυο αντίστοιχα (0200 με 0300, 0201 με 0301, κ.ο.κ.) και να βάζει στις αντίστοιχες θέσεις του πίνακα **0400**, **0401**, **0402**,…,**0409** το 00 αν είναι ίσα και το FF αν είναι διαφορετικά.

Στην συνέχεια δίνεται το λογικό διάγραμμα της λύσης του προβλήματος :



Για παράδειγμα, δώστε στους πίνακες στις θέσεις **0200..0209** και **0300..0309** τις παρακάτω τιμές. Αν το πρόγραμμά σας λειτουργεί σωστά θα πρέπει στον πίνακα στις θέσεις **0400..0409** να πάρετε τις παρακάτω τιμές.

0200	FF
0201	23
0202	A0
0203	23
0204	45
0205	1A
0206	0C
0207	66
0208	99
0209	FC

0300	DD
0301	23
0302	A1
0303	32
0304	54
0305	1A
0306	AB
0307	AC
0308	AC
0309	FC

0400	FF
0401	00
0402	FF
0403	FF
0404	FF
0405	00
0406	FF
0407	FF
0408	FF
0409	00

Άσκηση 6.2: Να γραφεί πρόγραμμα που να προσθέτει δύο πίνακες 10 θέσεων που βρίσκονται στις διευθύνσεις μνήμης **0200** και **0300** αντίστοιχα και να τοποθετεί τον πίνακα-αποτέλεσμα στην θέση **0400**

Άσκηση 6.3: Να γραφεί το ίδιο πρόγραμμα με χρήση **υπορουτίνας** για την πρόσθεση κάθε στοιχείου των πινάκων

Άσκηση 6.4: Να γραφεί το ίδιο πρόγραμμα με χρήση **υπορουτίνας** για την πρόσθεση κάθε στοιχείου των πινάκων. Οι πίνακες όμως τώρα αποτελούνται από αριθμούς των 16 bits.

Άσκηση 6.5: Να γραφεί πρόγραμμα το οποίο θα υπολογίζει το άθροισμα δυο 40 bit μη προσημασμένων δεκαεξαδικών αριθμών που είναι αποθηκευμένοι στις θέσεις μνήμης **0201-0205** και **0206-020A** αντίστοιχα με διάταξη **Μεγάλου Άκρου** (Big Endian). Το αποτέλεσμα θα αποθηκεύεται στις θέσεις **0301-0305** με το περισσότερο σημαντικό BYTE στη θέση **0301**.

Αριθμητικό παράδειγμα:

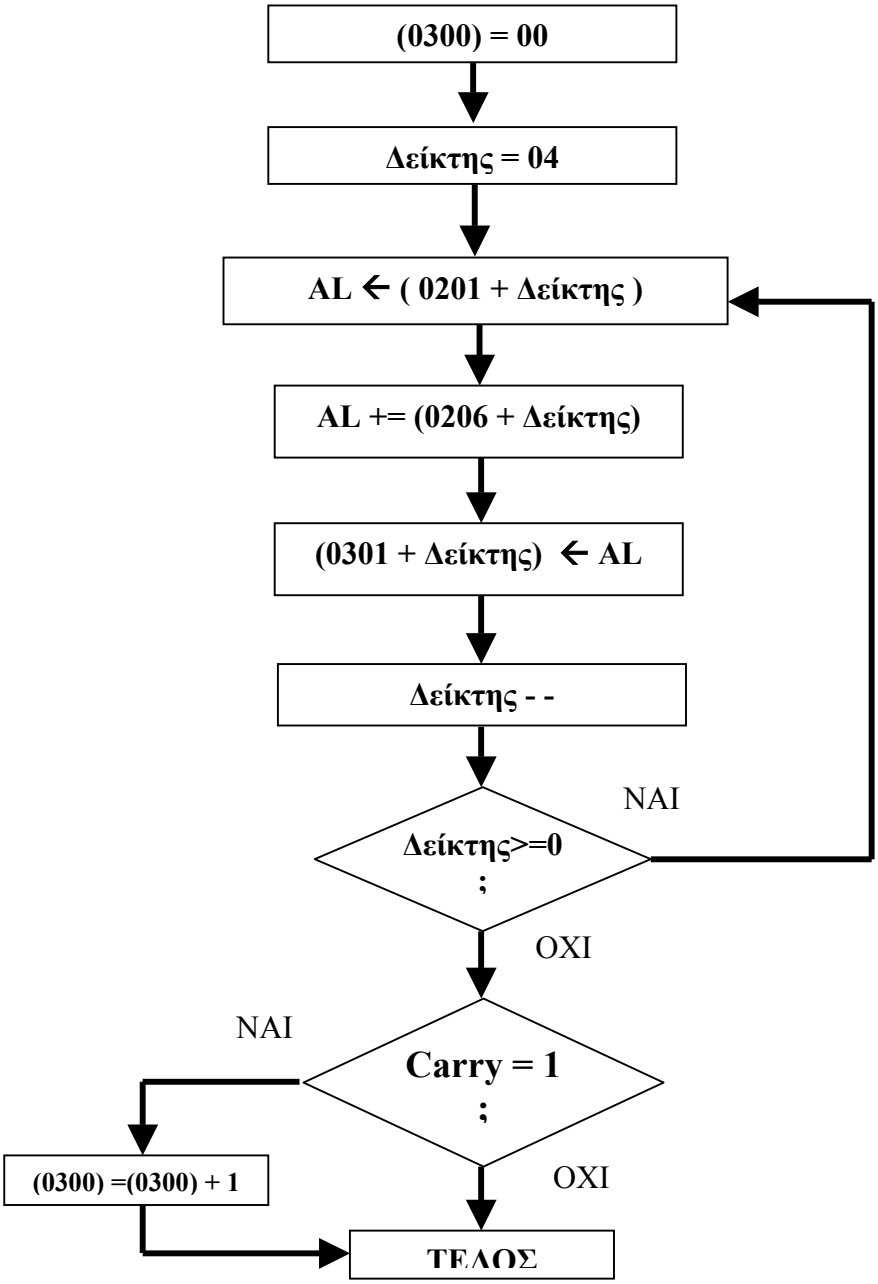
0201	**F8**
0202	**00**
0203	**35**
0204	**C2**
0205	**11**
0206	**10**
0207	**2C**
0208	**14**
0209	**33**
020A	**01**

0300	**01**
0301	**08**
0302	**2C**
0303	**49**
0304	**F5**
0305	**12**

```
     F80035C211
+    102C143301
     1082C49F512
```

Η θέση **0300** είναι αυτή που θα κρατήσει το κρατούμενο της πρόσθεσης, αν δημιουργηθεί. Γι αυτό μετά την πρόσθεση των δυο αριθμών θα ελέγξουμε το κρατούμενο και αν υπάρχει θα κάνουμε το περιεχόμενο της θέσης **0300** (που μέχρι τώρα ήταν 00) 01.

Λογικό Διάγραμμα :


(0300) = 00
Δείκτης = 04
AL ← (0201 + Δείκτης)
AL += (0206 + Δείκτης)
(0301 + Δείκτης) ← AL
Δείκτης - -
Δείκτης>=0 ;
NAI
OXI
Carry = 1 ;
NAI
OXI
(0300) =(0300) + 1
ΤΕΛΟΣ

7ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΟ : Χρήση Πινάκων ως Look-up Tables
Εντολή XLAT
Ένθετοι Βρόχοι – Διδιάστατοι Πίνακες
Ταξινόμηση Πινάκων
Εκτέλεση Προγραμμάτων με Ένθετους Βρόχους

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Οι πίνακες αναφοράς (Look-Up Tables) είναι πίνακες βάσει των οποίων γίνεται αντιστοίχιση μεταξύ δύο διατεταγμένων συνόλων, συνήθως ενός συνόλου ακέραιων θετικών αριθμών και ενός δεύτερου συνόλου που περιέχει κάποιου είδους αριθμητικές πληροφορίες ή σύμβολα κ.λ.π. (π.χ. αντιστοίχιση χαρακτήρων και αριθμών μέσω του κώδικα ASCII). Συνήθως ο προγραμματιστής έχει τον ακέραιο αριθμό που παίζει τον ρόλο του δείκτη στον πίνακα και θέλει να παίρνει το αντίστοιχο στοιχείο του πίνακα. Αυτό υλοποιείται εύκολα με την δεικτοδοτούμενη διευθυνσιοδότηση των εντολών του 8088. Όμως η assembly του 8088 μας παρέχει μία ειδική εντολή για την προσπέλαση τέτοιων πινάκων αναφοράς:

Η εντολή XLAT και οι τρόποι σύνταξής της

(Translate – Μετάφραση δεδομένων μέσω πίνακα)

Εντοπίζει ένα byte σε πίνακα στην μνήμη, την διεύθυνση βάσης του οποίου κρατά ο καταχωρητής BX (DS:BX) ενώ η θέση στον πίνακα (δείκτης) καθορίζεται από τον καταχωρητή AL. Το byte του πίνακα καταχωρείται πάλι στον AL

- XLAT

Παράδειγμα :

```
MOV BX,0200
MOV AL,05
XLAT
```

(βάζει στον AL το στοιχείο με δείκτη 5 (6ο στοιχείο) του πίνακα που αρχίζει στην διεύθυνση 0200).

Ένθετοι Βρόχοι

Κατά την ανάπτυξη εφαρμογών συναντάται η ανάγκη υλοποίησης ένθετων βρόχων. Με τον όρο αυτό εννοούμε δύο επαναληπτικές δομές (βρόχοι) ο ένας μέσα στον άλλο. Παράδειγμα σε C :

```
for (i=0 ; i<M ; i++)
{
        σώμα 1ου βρόχου
        for (j=0 ; j<N ; j++)
        {
                σώμα 2ου βρόχου
        }
        σώμα 1ου βρόχου
}
```

Εσωτερικός Βρόχος

Εξωτερικός Βρόχος

Εάν ο εξωτερικός βρόχος εκτελείται M φορές και ο εσωτερικός βρόχος εκτελείται N φορές τότε το σώμα του εσωτερικού βρόχου θα εκτελεστεί MxN φορές.

Οι ένθετοι βρόχοι υλοποιούνται στην γλώσσα Assembly με δύο διακλαδώσεις προς προηγούμενες διευθύνσεις όπου η μία περικλείει την άλλη, π.χ.

```
0100:0000 …
0100:0003 MOV CX,5
0100:0006 …
0100:0008 MOV DX,0
0100:000A …
0100:000C INC DX
0100:000D CMP DX,04
0100:0010 JNE 000A
0100:0013 …
0100:0015 LOOP 06
```

Εσωτερικός Βρόχος

Εξωτερικός Βρόχος

Οι ένθετοι βρόχοι (που μπορεί να είναι και πάνω από 2 επιπέδων) χρησιμοποιούνται σε πλήθος αλγορίθμων, όπως η επεξεργασία διδιάστατων ή γενικά ν-διάστατων πινάκων, οι αλγόριθμοι ταξινόμησης, κ.λ.π.

Διδιάστατοι Πινακες

Οι διδιάστατοι πίνακες αποθηκεύονται στην μνήμη σειριακά. Πρώτα αποθηκεύεται η πρώτη γραμμή, κατόπιν η δεύτερη, η τρίτη κ.ο.κ. όπως φαίνεται στο παρακάτω σχήμα

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

0200	0201	0202	0203	0204	0205	0206	0207	0208
X_{11}	X_{12}	X_{13}	X_{21}	X_{22}	X_{23}	X_{31}	X_{32}	X_{33}

Έτσι λοιπόν η προσπέλαση των στοιχείων ενός διδιάστατου πίνακα μπορεί να γίνει σειριακά με απλό βρόχο (με ένα δείκτη) ο οποίος απλά θα εκτελεστεί για MxN φορές (εδώ 3x3=9 φορές). Αν όμως θέλουμε να γνωρίζουμε σε κάθε στιγμή την γραμμή και την στήλη που βρισκόμαστε, θα πρέπει να χρησιμοποιήσουμε δύο δείκτες και διπλό ένθετο βρόχο. Παραδείγματα :

Προσπέλαση πίνακα 3X3 σειριακά με απλό βρόχο και ένα δείκτη

```
0100:0000   MOV SI,0000
0100:0003   MOV AL,[0200+SI]
0100:0006   …
0100:000A   INC SI
0100:000B   CMP SI, 9
0100:000E   JNE 0003
```

Προσπέλαση πίνακα 3X3 σειριακά με απλό βρόχο ένα δείκτη και 2 καταχωρητές παρακολούθησης γραμμής και στήλης

```
0100:0000   MOV SI,0000
0100:0003   MOV BL,00       ← καταχωρητής παρακολούθησης γραμμής
0100:0005   MOV CL,00       ← καταχωρητής παρακολούθησης στήλης
0100:0007   MOV AL,[0200+SI]
0100:000A   …
0100:0010   INC SI
0100:0011   INC CL          ← αύξηση στήλης
0100:0012   CMP CL,03       ← έλεγχος τέλους στηλών
0100:0015   JB 001C
0100:0018   MOV CL,00       ← μηδενισμός στήλης
```

```
0100:001B   INC BL          ← αύξηση γραμμής
0100:001C   CMP SI, 9
0100:001F   JNE 0007
```

Προσπέλαση πίνακα 3Χ3 με διπλό βρόχο ένα δείκτη και 2 καταχωρητές παρακολούθησης γραμμής και στήλης

```
0100:0000   MOV SI,0000
0100:0003   MOV BL,00       ← καταχωρητής παρακολούθησης γραμμής
0100:0005   MOV CL,00       ← καταχωρητής παρακολούθησης στήλης
0100:0007   MOV AL,[0200+SI]
0100:000A   …
0100:0010   INC SI
0100:0011   INC CL          ← αύξηση στήλης
0100:0012   CMP CL,03       ← έλεγχος τέλους στηλών
0100:0015   JNE 007
0100:0018   MOV CL,00       ← μηδενισμός στήλης
0100:001B   INC BL          ← αύξηση γραμμής
0100:001C   CMP BL, 03      ← έλεγχος τέλους γραμμών
0100:001F   JNE 0007
```

Προσπέλαση πίνακα 3Χ3 με διπλό βρόχο και δύο δείκτες γραμμής και στήλης

```
0100:0000   MOV BP,0000
0100:0003   MOV SI,0000
0100:0006   MOV AL,[0200+BP+SI]
0100:000A   …
0100:0010   INC SI
0100:0012   CMP SI,03       ← έλεγχος τέλους στηλών
0100:0015   JNE 006
0100:0018   MOV SI,0000     ← μηδενισμός στήλης
0100:001B   ADD BP,03       ← αύξηση γραμμής κατά 3
0100:001C   CMP BP, 09      ← έλεγχος τέλους γραμμών
0100:001F   JNE 0006
```

Ταξινόμηση Πινάκων

Για την ταξινόμηση πινάκων έχουν αναπτυχθεί αρκετοί αλγόριθμοι, με διαφορετικές αποδόσεις, αναλόγως του πλήθους των στοιχείων του πίνακα και του κατά πόσο είναι ήδη μερικώς ταξινομημένα. Από τους πιο δημοφιλείς, για εκπαιδευτική χρήση κυρίως, αλγορίθμους είναι η μέθοδος **Bubblesort**, ή μέθοδος των φυσαλίδων.

Στον αλγόριθμο αυτόν, για να πραγματοποιηθεί αύξουσα ταξινόμηση, το πρώτο στοιχείο του πίνακα συγκρίνεται με το δεύτερο. Εάν το πρώτο στοιχείο είναι μεγαλύτερο, τα στοιχεία ανταλλάσσουν τις θέσεις τους. Έπειτα το δεύτερο στοιχείο συγκρίνεται με το τρίτο, ανταλλάσσονται αν χρειάζεται κ.ο.κ. Μόλις ο Μ/Ε φθάσει στο τέλος του πίνακα ($1^ο$ πέρασμα του πίνακα) το μεγαλύτερο στοιχείο θα έχει φθάσει στην τελευταία θέση του πίνακα.

Εάν στο $1^ο$ πέρασμα έχει συμβεί έστω και μια εναλλαγή τότε ο Μ/Ε θα ξανακάνει και $2^ο$ πέρασμα του πίνακα κ.ο.κ. Το εάν έγινε ή όχι εναλλαγή σε δυο στοιχεία του πίνακα το βλέπουμε σε μια θέση μνήμης η οποία πριν το πέρασμα του πίνακα είναι 00 και σε κάθε εναλλαγή θέσεων αυξάνει το περιεχόμενό της. Όταν λοιπόν μετά από ένα πέρασμα του πίνακα το περιεχόμενό της συγκεκριμένης θέσης είναι διάφορο του 0 σημαίνει ότι έγινε τουλάχιστο μια εναλλαγή και πρέπει να συνεχιστούν τα περάσματα. Τα περάσματα θα είναι τόσα ώστε στο τελευταίο πέρασμα να μη συμβεί εναλλαγή.

Παράδειγμα :

Έστω ο πίνακας:

05, 03, 04, 01, 02

μετά το 1ο πέρασμα του πίνακα η διάταξή του θα είναι:

03, 04, 01, 02, 05 (συνέβησαν 4 εναλλαγές)

μετά το 2ο πέρασμα του πίνακα η διάταξή του θα είναι:

03, 01, 02, 04, 05 (συνέβησαν 2 εναλλαγές)

μετά το 3ο πέρασμα του πίνακα η διάταξή του θα είναι:

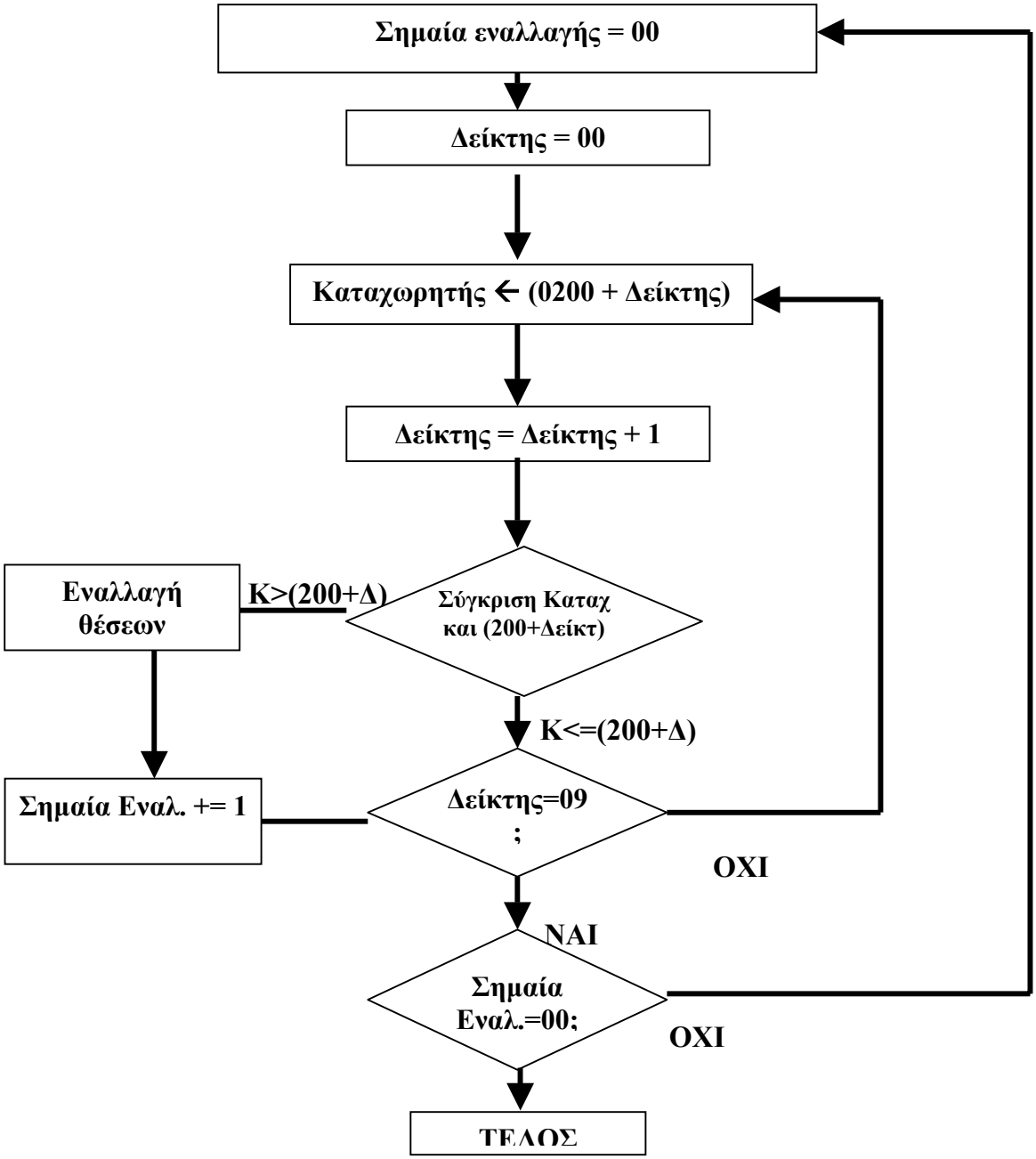
01, 02, 03, 04, 05 (συνέβησαν 2 εναλλαγές)

το 4ο πέρασμα θα είναι το τελευταίο όπου δεν θα συμβούν εναλλαγές και θα καταλάβει ο Μ/Ε ότι έχει τακτοποιηθεί ο πίνακας.

ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Άσκηση 7.1: Να γίνει πρόγραμμα που να ταξινομεί κατ' αύξουσα σειρά, με τον αλγόριθμο **Bubblesort**, τους αριθμούς–τιμές 8 bit ενός πίνακα που βρίσκεται στις θέσεις μνήμης **0200 – 0209** (10 τιμές). Δηλαδή το μικρότερο να τοποθετείται στη θέση **0200** και το μεγαλύτερο στη θέση **0209**.

Το λογικό διάγραμμα του προγράμματος θα έχει ως εξής:



Για παράδειγμα, δώστε τους αριθμούς του παρακάτω πίνακα :

Πριν την ταξινόμηση :

0200	**0201**	**0202**	**0203**	**0204**	**0205**	**0206**	**0207**	**0208**	**0209**
FF	FE	FD	FC	FB	FA	F9	F8	F7	F6

Μετά την ταξινόμηση ο πίνακας θα πρέπει να έχει τη μορφή :

0200	**0201**	**0202**	**0203**	**0204**	**0205**	**0206**	**0207**	**0208**	**0209**
F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

Μπορεί να μελετηθεί ως άσκηση η επέκταση του ανωτέρω προγράμματος ώστε να υπολογίζει και το συνολικό πλήθος των αντιμεταθέσεων που έχουν συμβεί, από την αρχή έως το τέλος του προγράμματος.

Άσκηση 7.2: Να αναπτυχθεί πρόγραμμα που να προσθέτει τα στοιχεία ενός διδιάστατου πίνακα **3X3** που βρίσκεται στην διεύθυνση μνήμης 0200 με χρήση διπλού ένθετου βρόχου. Η πρώτη γραμμή βρίσκεται στις θέσεις **0200, 0201, 0202**, η δεύτερη γραμμή στις θέσεις **0203, 0204, 0205** και η τρίτη γραμμή στις θέσεις **0206, 0207, 0208**.

Το αντίστοιχο πρόγραμμα σε γλώσσα C θα ήταν :

```
sum=0;
for (i=0 ; i<3 ; i++)
        for (j=0 ; j<3 ; j++)
                sum+=A[i][j];
```

Το πρόγραμμα να υλοποιηθεί και με απλό βρόχο αλλά με μεταβλητές παρακολούθησης της γραμμής και της στήλης.

Άσκηση 7.3: Να αναπτυχθεί πρόγραμμα που να μας δίνει το τετράγωνο ενός αριθμού μέσω look-up table για τους αριθμούς από 1..100. Το look-up table θα κατασκευάζεται υπολογιστικά στην αρχή του προγράμματος και θα αρχίζει στην διεύθυνση 0400. Στη συνέχεια το πρόγραμμα θα λαμβάνει αριθμούς από έναν πίνακα 10 θέσεων που αρχίζει στην θέση 0200 και θα υπολογίζει μέσω look-up table τα τετράγωνα των αριθμών τα οποία και θα τοποθετεί σε πίνακα που θα αρχίσει στην θέση 0300.

8ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΟ : Εργασίες με Αλφαριθμητικά,
Εντολές REP, MOVS, CMPS, SCAS, LODS, STOS,
Μετατροπή Αλφαριθμητικών,
Αναζήτηση χαρακτήρων
Εκτέλεση προγραμμάτων διαχείρισης αλφαριθμητικών

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Τα αλφαριθμητικά είναι στην ουσία πίνακες αριθμών στην μνήμη (συνήθως του 1ος byte), στους οποίους οι αριθμοί αντιστοιχούν σε χαρακτήρες μέσω της αντιστοίχισης του κώδικα ASCII. Παράδειγμα :

0200	**0201**	**0202**	**0203**	**0204**	**0205**
48_{16}	45_{16}	$4C_{16}$	$4C_{16}$	$4F_{16}$	00_{16}
H	E	L	L	O	

Τα αλφαριθμητικά συνήθως τερματίζουν με τον αριθμό 0 (χαρακτήρας '\0'). Ο κανόνας αυτός αποτελεί απλά μία σύμβαση για την χρήση των αλφαριθμητικών, που ισχύει σε αρκετές γλώσσες προγραμματισμού και πρωτίστως στη γλώσσα C. Η ύπαρξη του κανόνα είναι αναγκαία καθώς τα αλφαριθμητικά δηλώνονται με ένα μέγιστο πλήθος χαρακτήρων, αλλά μπορεί ανά πάσα στιγμή να περιέχουν μία φράση που να έχει λιγότερους χαρακτήρες από το μέγιστο. Παράδειγμα :
Δήλωση : char s[20];
(Δέσμευση χώρου στη μνήμη για 20 συνεχόμενα bytes, που αρχικά έχουν τυχαίες τιμές)

.	@	σ	#	\n	e	y		!	2	;	}	`	E	W	^	P	"	>	F

Ανάθεση : strcpy (s, "ASSEMBLY");
(Τοποθέτηση των αριθμών ASCII των γραμμάτων και τερματισμός με \0)

A	S	S	E	M	B	L	Y	\0	2	;	}	`	E	W	^	P	"	>	F

Όπως γίνεται αντιληπτό, ο τερματισμός των αλφαριθμητικών είναι κρίσιμος γιατί :

1. Όταν πρόκειται να χρησιμοποιηθεί το αλφαριθμητικό (π.χ. να εμφανιστεί στην οθόνη) λαμβάνονται μόνο οι χαρακτήρες που περιέχει έως και πριν τον χαρακτήρα τερματισμού, και όχι ολόκληρο το αλφαριθμητικό που μπορεί να περιέχει είτε τυχαίους χαρακτήρες, είτε χαρακτήρες που έχουν τοποθετηθεί εκεί από προηγούμενη ανάθεση.
2. Κερδίζουμε σε χρόνο καθώς όλοι οι βρόχοι επεξεργασίας αλφαριθμητικών σταματάνε στον χαρακτήρα τερματισμού και δεν συνεχίζουν για ολόκληρο το αλφαριθμητικό. Για παράδειγμα μία αντιγραφή αλφαριθμητικών με μία εντολή π.χ. strcpy (s2, s) θα εκτελέσει ένα βρόχο 8 επαναλήψεων (όσα και τα γράμματα της λέξης ASSEMBLY) και όχι 20 επαναλήψεων που είναι το μέγιστο μήκος του αλφαριθμητικού.

Από τα παραπάνω προκύπτει ότι το μέγιστο πλήθος χαρακτήρων που μπορούμε να τοποθετήσουμε σε ένα αλφαριθμητικό 20 θέσεων (δηλωμένο ως char s[20]) είναι 19, δηλαδή κατά ένα λιγότερο από το μέγεθος του αλφαριθμητικού. Αυτό οφείλεται στο ότι πρέπει να κρατηθεί και μία θέση για τον χαρακτήρα τερματισμού του.
Επίσης ένα «κενό» αλφαριθμητικό απλά έχει τον χαρακτήρα τερματισμού στην πρώτη θέση του πίνακα (θέση 0) :

\0	S	S	E	M	B	L	Y	\0	2	;	}	`	E	W	^	P	"	>	F

Να σημειωθεί ότι για τον 8088, τα αλφαριθμητικά δεν διαφέρουν καθόλου από τους μονοδιάστατους πίνακες αριθμών. Έτσι ότι αναφέρουμε στο κεφάλαιο αυτό για τα αλφαριθμητικά, μπορεί να χρησιμοποιηθεί και για μονοδιάστατους πίνακες αριθμών του $1^{ος}$ byte.

Για την υλοποίηση αλγορίθμων με αλφαριθμητικά μπορούμε απλά να χρησιμοποιήσουμε εντολές με δεικτοδοτούμενη διευθυνσιοδότηση όπως και στους πίνακες. Ωστόσο η γλώσσα μηχανής του 8088 μας παρέχει τις εξής ειδικές εντολές για αλφαριθμητικά :

Χρησιμοποιούμενες εντολές:

- **REP** (Repeat – Επανέλαβε)
- **MOVS** (Move String Byte/Word – Αντιγραφή byte/word από String σε String)
- **LODS** (Load String Byte/Word – Φόρτωσε byte/word ενός string)
- **STOS** (Store String Byte/Word – Αποθήκευσε byte/word σε string)
- **CMPS** (Compare String Byte/Word – Σύγκρινε byte/word από δύο string)
- **SCAS** (Scan String Byte/Word – Αναζήτηση byte/word σε string)

Η εντολή REP και οι τρόποι σύνταξής της

(Repeat – Επανέλαβε)

Επαναλαμβάνει την επόμενη εντολή αλφαριθμητικών τόσες φορές όσο η τιμή του καταχωρητή CX, ο οποίος μειώνεται σε κάθε επανάληψη κατά 1, με τελική τιμή = 0. Προσοχή, λειτουργεί μόνο όταν η επόμενη εντολή είναι μία από τις εντολές αλφαριθμητικών, δηλαδή τις MOVS, CMPS, SCAS, LODS, STOS.

- REP

Παράδειγμα :

```
MOV AX,41
MOV CX,5
REP
STOSB
```

(Αποθηκεύει 5 bytes 41_{16} ή "A" στην θέση ES:DI)

Η εντολή MOVS και οι τρόποι σύνταξής της

(Move String Byte/Word – Αντιγραφή byte/word από String σε String)

Αντιγράφει ένα byte/word ενός string από την διεύθυνση DS:SI στην διεύθυνση ES:DI. Αμέσως μετά αυξάνει ή μειώνει τους καταχωρητές SI και DI κατά 1 (2 για word), ανάλογα με την τιμή της σημαίας κατεύθυνσης (Direction Flag – DF).

Σημαία Κατεύθυνσης	Σημασία	Μεταβολή των SI, DI
0	UP	Αύξηση
1	DOWN	Μείωση

Συνδυάζεται με την εντολή "REP" η οποία εκτελεί την MOVS τόσες φορές όσο η τιμή του καταχωρητή CX, ο οποίος μειώνεται σε κάθε επανάληψη, με τελική τιμή = 0.

- MOVSB

  Αντιγράφει ένα byte ενός string από την διεύθυνση DS:SI στην διεύθυνση ES:DI.

  Παράδειγμα :

```
MOV SI, 200
MOV DI, 300
MOV CX, 4
REP MOVSB
```

  (Αντιγράφει 4 bytes από το string στην θέση DS:SI στο string που είναι στη θέση ES:DI, αυξάνοντας τους SI, DI σε κάθε επανάληψη και μειώνοντας τον CX)

- MOVSW

Αντιγράφει δύο byte (word) ενός string από την διεύθυνση DS:SI στην διεύθυνση ES:DI. Αμέσως μετά αυξάνει ή μειώνει τους καταχωρητές SI και DI κατά 2, ανάλογα με την τιμή της σημαίας κατεύθυνσης (Direction Flag – DF).

Η εντολή LODS και οι τρόποι σύνταξής της

(Load String Byte/Word – Φόρτωσε byte/word ενός string)

Φορτώνει στον καταχωρητή AL ή AX (byte/word) το περιεχόμενο της θέσης μνήμης που είναι αποθηκευμένη στους καταχωρητές DS:SI. Αμέσως μετά αυξάνει ή μειώνει τον καταχωρητή SI κατά 1 (2 για word), ανάλογα με την τιμή της σημαίας κατεύθυνσης. Συνδυάζεται με την εντολή "REP" αλλά πιο συχνά συνδυάζεται με την εντολή LOOP ώστε να υπάρχει περιθώριο επεξεργασίας των bytes που φορτώνονται πριν το επόμενο φόρτωμα.

- LODSB

  Φορτώνει στον καταχωρητή AL, 1 byte από τη θέση μνήμης που είναι αποθηκευμένη στους καταχωρητές DS:SI. Αμέσως μετά αυξάνει ή μειώνει τον καταχωρητή SI κατά 1

  Παράδειγμα :

  MOV SI,0500
  LODSB
  MOV [0200],AL

  (Φορτώνει 1 byte από την θέση DS:SI (DS:0500), και το αποθηκεύει στη θέση μνήμης DS:0200)
- LODSW

  Φορτώνει στον καταχωρητή AX, 2 byte από τη θέση μνήμης που είναι αποθηκευμένη στους καταχωρητές DS:SI. Αμέσως μετά αυξάνει ή μειώνει τον καταχωρητή SI κατά 2, ανάλογα με την τιμή της σημαίας κατεύθυνσης

Η εντολή STOS και οι τρόποι σύνταξής της

(Store String Byte/Word – Αποθήκευσε byte/word σε string)

Γράφει το περιεχόμενο του καταχωρητή AL ή AX (1 ή 2 byte) στην θέση μνήμης που είναι αποθηκευμένη στους καταχωρητές ES:DI. Αμέσως μετά αυξάνει ή μειώνει τον καταχωρητή DI κατά 1, ανάλογα με την τιμή της σημαίας κατεύθυνσης

- STOSB

  Γράφει το περιεχόμενο του καταχωρητή AL (1 byte) στην θέση μνήμης που είναι αποθηκευμένη στους καταχωρητές ES:DI. Αμέσως μετά αυξάνει ή μειώνει τον καταχωρητή DI κατά 1, ανάλογα με την τιμή της σημαίας κατεύθυνσης

  Παράδειγμα :

  MOV AX,41
  MOV CX,5
  REP STOSB

  (Αποθηκεύει 5 bytes 41_{16} ή "A" στην θέση ES:DI, αυξάνοντας τον DI σε κάθε επανάληψη ώστε τα 'A' να αποθηκεύονται σε διαδοχικές θέσεις)
- STOSW

  Γράφει το περιεχόμενο του καταχωρητή AX (2 byte) στην θέση μνήμης που είναι αποθηκευμένη στους καταχωρητές ES:DI. Αμέσως μετά αυξάνει ή μειώνει τον καταχωρητή DI κατά 2, ανάλογα με την τιμή της σημαίας κατεύθυνσης

Η εντολή CMPS και οι τρόποι σύνταξής της

(Compare String Byte/Word – Σύγκρινε byte/word από δύο string)

Συγκρίνει ένα byte (ή 2) ενός string από την διεύθυνση DS:SI με ένα byte (ή 2) ενός άλλου string στην διεύθυνση ES:DI, επηρεάζοντας τις σημαίες (flags). Αμέσως μετά αυξάνει ή μειώνει τους καταχωρητές SI και DI κατά 1 (2 για word), ανάλογα με την τιμή της σημαίας κατεύθυνσης

- CMPSB

Συγκρίνει ένα byte ενός string από την διεύθυνση DS:SI με ένα byte ενός άλλου string στην διεύθυνση ES:DI, επηρεάζοντας τις σημαίες (flags). Αμέσως μετά αυξάνει ή μειώνει τους καταχωρητές SI και DI κατά 1, ανάλογα με την τιμή της σημαίας κατεύθυνσης
Παράδειγμα :
MOV SI, 200
MOV DI, 300
CMPSB
(Συγκρίνει 1 byte από το string στην θέση DS:200 με 1 byte από το string που είναι στη θέση ES:300, επηρεάζοντας τις σημαίες, και αυξάνοντας τους SI, DI)

- CMPSW

  Συγκρίνει δύο byte (word) ενός string από την διεύθυνση DS:SI με δύο byte ενός άλλου string στην διεύθυνση ES:DI, επηρεάζοντας τις σημαίες (flags). Αμέσως μετά αυξάνει ή μειώνει τους καταχωρητές SI και DI κατά 2, ανάλογα με την τιμή της σημαίας κατεύθυνσης

Η εντολή SCAS και οι τρόποι σύνταξής της
(Scan String Byte/Word – Αναζήτηση byte/word σε string)
Συγκρίνει το περιεχόμενο του καταχωρητή AL (AX για word) με ένα byte (2 για word) ενός string στην διεύθυνση ES:DI, επηρεάζοντας τις σημαίες (flags). Αμέσως μετά αυξάνει ή μειώνει τον καταχωρητή DI κατά 1 (2 για word), ανάλογα με την τιμή της σημαίας κατεύθυνσης

- SCASB

  Συγκρίνει το περιεχόμενο του καταχωρητή AL με 1 byte ενός string στην διεύθυνση ES:DI, επηρεάζοντας τις σημαίες (flags). Αμέσως μετά αυξάνει ή μειώνει τον καταχωρητή DI κατά 1, ανάλογα με την τιμή της σημαίας κατεύθυνσης
  Παράδειγμα :
  MOV AL, 41
  MOV DI, 300
  SCASB
  (Συγκρίνει τον AL με 1 byte από το string στην θέση ES:DI, επηρεάζοντας τις σημαίες, και αυξάνοντας τον DI)

- SCASW

  Συγκρίνει το περιεχόμενο του καταχωρητή AX (δύο byte - word) με δύο byte ενός string που βρίσκεται στην διεύθυνση ES:DI, επηρεάζοντας τις σημαίες (flags). Αμέσως μετά αυξάνει ή μειώνει τον καταχωρητή DI κατά 2, ανάλογα με την τιμή της σημαίας κατεύθυνσης

ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

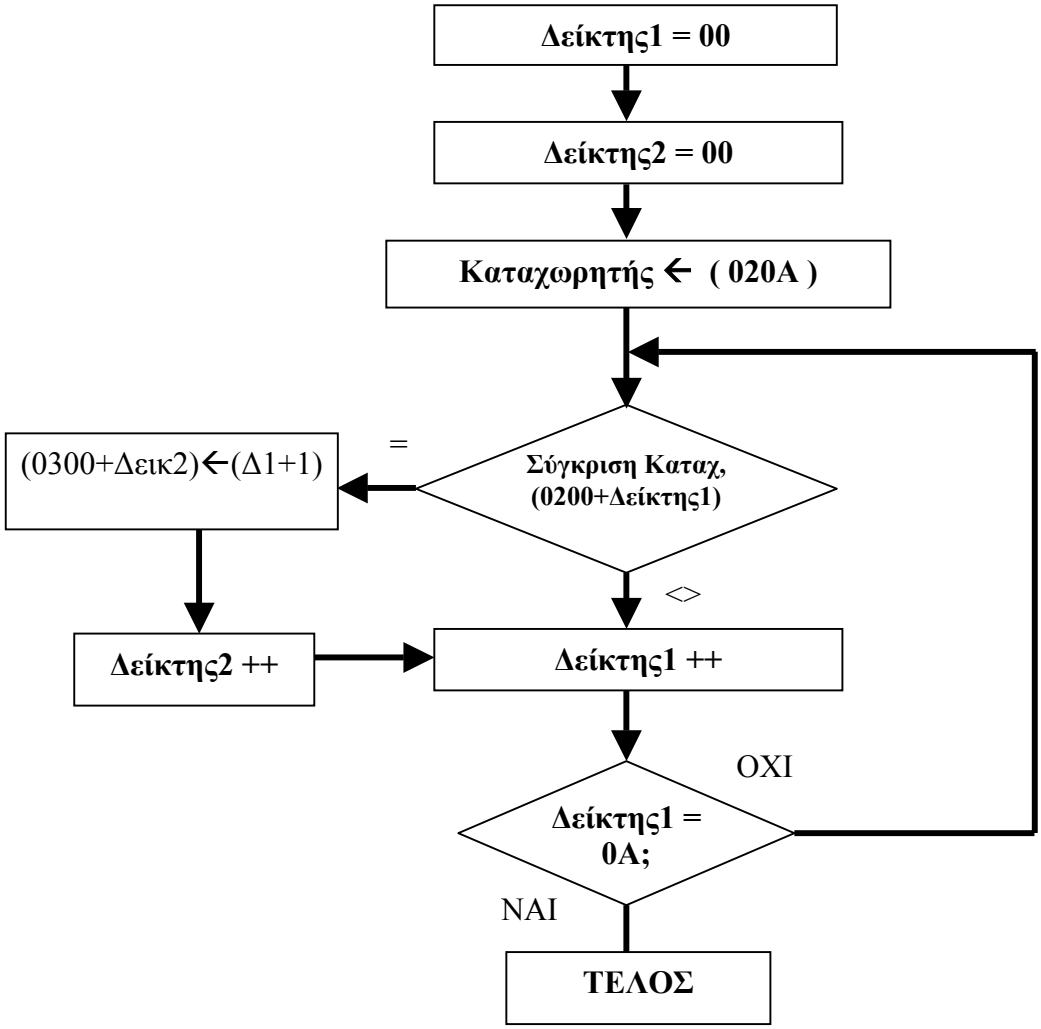
Άσκηση 8.1: Να γίνει πρόγραμμα που θα ψάχνει εάν σ' έναν πίνακα που βρίσκεται στις θέσεις **0200-0209** υπάρχει η τιμή που βρίσκεται στη θέση **020A** και θα σημειώνει στις θέσεις **0300** και πέρα σε ποια θέση ($1^{η}$,$2^{η}$, $3^{η}$, ….) από τις 10 (0200-0209) βρίσκεται αυτή. Π.χ. εάν η ζητούμενη τιμή βρίσκεται στις θέσεις **0203**, **0207** τότε το πρόγραμμα θα βάλει στις θέσεις **0300**, **0301** τις τιμές:

0300: 04 ($4^{ο}$ στοιχείο του πίνακα)
0301: 08 ($8^{ο}$ στοιχείο του πίνακα)

Το πρόγραμμα να υλοποιηθεί μία φορά με δεικτοδοτούμενη διευθυνσιοδότηση και μία φορά με εντολές αλφαριθμητικών.

Το BLOCK ΔΙΑΓΡΑΜΜΑ του προγράμματος θα έχει ως εξής:

BLOCK ΔΙΑΓΡΑΜΜΑ



Για παράδειγμα, δώστε τους αριθμούς του παρακάτω πίνακα :

020A

FF

0200	**0201**	**0202**	**0203**	**0204**	**0205**	**0206**	**0207**	**0208**	**0209**
FF	66	FD	78	FB	FF	FF	F8	F7	F6

Αποτέλεσμα :

0300	**0301**	**0302**	**0303**	**0304**	**0305**	**0306**	**0307**	**0308**	**0309**
01	06	07							

Υπόδειξη : Πριν εκτελέσετε το πρόγραμμα γεμίστε τις διευθύνσεις μνήμης **0300..0309** με την τιμή **00**, ώστε να μην επηρεαστεί το αποτέλεσμα από προηγούμενες τιμές.

Άσκηση 8.2: Να αναπτυχθεί πρόγραμμα που θα αντιγράφει ένα string (αλφαριθμητικό), με μέγιστο μήκος 8 χαρακτήρες, από τη θέση **0200** στη θέση **0208**. Θεωρείστε ότι το τέλος του string σηματοδοτείται με το '\0' (αριθμός 0 όχι χαρακτήρας '0'). Η υλοποίηση να γίνει μία φορά με την εντολή MOVS και 1 φορά με τις εντολές LODSB, STOSB.
Μπορεί να γίνει δοκιμή με το string :

0200	**0201**	**0202**	**0203**	**0204**	**0205**
T	O	D	A	Y	
54_{16}	$4F_{16}$	44_{16}	41_{16}	59_{16}	00_{16}

Άσκηση 8.3: Να αναπτυχθεί πρόγραμμα που θα μετατρέπει ένα string (αλφαριθμητικό) από κεφαλαία σε πεζά. Θεωρείστε ότι το αρχικό string με τα κεφαλαία βρίσκεται στη θέση μνήμης **0200** και έχει μέγιστο μήκος 8 χαρακτήρες ενώ το τέλος του string σηματοδοτείται με το '\0' (αριθμός 0 όχι χαρακτήρας '0'). Το τελικό string να αποθηκεύεται στη διεύθυνση μνήμης **0208**.

Επεξήγηση : Τα strings περιέχουν γράμματα αριθμούς και σύμβολα με κωδικοποίηση ASCII. Κάθε γράμμα του string σώζεται ως ένα byte και αντιστοιχεί σε ένα συγκεκριμένο αριθμό 0..255. Π.χ. για τα κεφαλαία Αγγλικά γράμματα 'A'=41_{16}, 'B'=42_{16}, 'C'=43_{16}, …, 'Z'=$5A_{16}$. Για τα πεζά Αγγλικά γράμματα 'a'=61_{16}, 'b'=62_{16}, 'c'=63_{16}, …, 'z'=$7A_{16}$. Παρατηρούμε ότι τα κεφαλαία και τα μικρά γράμματα είναι κωδικοποιημένα στη σειρά και δύο αντίστοιχα γράμματα διαφέρουν μεταξύ τους κατά $\mathbf{20}_{16}$.
Σημείωση : Ο έλεγχος για τέλος του string να γίνεται στην αρχή του βρόχου γιατί το string μπορεί να είναι και εξαρχής άδειο και να περιέχει ως πρώτο χαρακτήρα τον \0 = 'τέλος του string'.

Μπορεί να γίνει δοκιμή με το string :

0200	**0201**	**0202**	**0203**	**0204**	**0205**
H	E	L	L	O	
48_{16}	45_{16}	$4C_{16}$	$4C_{16}$	$4F_{16}$	00_{16}

9ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΟ : Λογικές Εντολές, AND, OR, XOR, NOT, TEST,
Εντολές ολίσθησης και περιστροφής, SHL/SAL, SHR, SAR, ROL, ROR, RCL, RCR,
Έλεγχος τιμής bit,
Καθορισμός τιμής bit,
Εκτέλεση προγραμμάτων με λογικές εντολές.

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Εκτός από τις εντολές αριθμητικών πράξεων που επενεργούν στα περιεχόμενα των καταχωρητών ή της μνήμης, τα οποία θεωρούνται συνολικά ως ακέραιοι αριθμοί, η Assembly του 8088 μας παρέχει και μία ομάδα εντολών για την εκτέλεση **λογικών πράξεων** πάνω στα δυαδικά δεδομένα που περιέχουν οι καταχωρητές ή η μνήμη. Οι λογικές πράξεις αντιμετωπίζουν τα περιεχόμενα των καταχωρητών ή της μνήμης ως πίνακες από bits, και πραγματοποιούν πράξεις με τα αντίστοιχα bits των δύο τελεσταίων που συμμετέχουν στην λογική πράξη. Οι λογικές πράξεις που υποστηρίζει η Assembly του 8088 είναι :

Πράξη AND (Λογικό «και») : αν και τα δύο αντίστοιχα bits είναι «1» δίνει «1», αλλιώς «0»
Πράξη OR (Λογικό «ή») : αν και τα δύο αντίστοιχα bits είναι «0» δίνει «0», αλλιώς «1»
Πράξη XOR (Αποκλειστικό «ή») : αν τα δύο αντίστοιχα bits είναι ίδια δίνει «0», αλλιώς «1»
Πράξη NOT (Λογική αντιστροφή) : αν το bit είναι «0» δίνει «1», ενώ αν είναι «1» δίνει «0»

Με τις λογικές πράξεις μπορούμε να κάνουμε τα ακόλουθα :

1. Να υπολογίσουμε τιμές λογικών συναρτήσεων (συναρτήσεων Boole).
2. Να αλλάξουμε την τιμή ενός μόνο bit σε ένα καταχωρητή ή σε μία θέση μνήμης, χωρίς να επηρεαστούν τα υπόλοιπα bits.
3. Να ελέγξουμε την τιμή ενός bit ενός καταχωρητή ή μίας θέσης μνήμης.

Τα παραπάνω είναι ιδιαίτερα σημαντικά για τον προγραμματισμό περιφερειακών συσκευών, καθώς οι λέξεις που αποστέλνονται προς τις θύρες (ports) εξόδου, ή λαμβάνονται στις θύρες εισόδου των συσκευών αυτών αποτελούνται από bits, που το καθένα έχει ειδική σημασία για την κατάσταση και την λειτουργία της περιφερειακής συσκευής.
Μερικές από τις σημαντικές ιδιότητες των λογικών πράξεων είναι οι ακόλουθες :

1. Αριθμός AND 11111111 = Αριθμός
2. Αριθμός AND 00000000 = 00000000
3. Αριθμός AND Αριθμός = Αριθμός
4. Αριθμός OR 11111111 = 11111111
5. Αριθμός OR 00000000 = Αριθμός
6. Αριθμός OR Αριθμός = Αριθμός
7. Αριθμός XOR 11111111 = NOT Αριθμός
8. Αριθμός XOR 00000000 = Αριθμός
9. Αριθμός XOR Αριθμός = 00000000

Αλλαγή τιμής ενός bit

α) Μηδενισμός ενός bit

Για να μηδενίσουμε ένα bit ενός δυαδικού αριθμού χωρίς να επηρεαστούν τα υπόλοιπα πρέπει κάνουμε λογικό "**ΚΑΙ**" (**AND**) μεταξύ του δυαδικού αριθμού και μιας **μάσκας** που έχει "0" στη θέση του δυαδικού ψηφίου που θέλουμε να μηδενίσουμε και "1" σε όλες τις υπόλοιπες θέσεις.

Παράδειγμα : Ας υποθέσουμε ότι ο καταχωρητής AL έχει το παρακάτω περιεχόμενο :

	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	
(AL) :	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**→ $(F5)_{16}$**

και ότι θέλουμε να μηδενίσουμε το δυαδικό ψηφίο B_5, χωρίς να αλλάξουμε τις τιμές των υπολοίπων. Το πρόγραμμα θα περιλαμβάνει ένα λογικό "**ΚΑΙ**" μεταξύ του καταχωρητή AL και ενός αριθμού που θα έχει όλα τα ψηφία του "1" εκτός από το B_5, που θα είναι "0", δηλαδή με τον $(\mathbf{11011111})_2$ ή $(\mathbf{DF})_{16}$.Το επιθυμητό αποτέλεσμα θα είναι ο αριθμός $(\mathbf{11010101})_2$ ή $(\mathbf{D5})_{16}$.
Έτσι το πρόγραμμα θα κάνει την εξής πράξη:

```
      11110101                 (F5)16
AND   11011111   ή     AND     (DF)16
      11010101                 (D5)16
```

Ο πίνακας αληθείας του λογικού "ΚΑΙ" (AND) είναι ο παρακάτω:

X	Y	X (AND) Y
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

β) Τοποθέτηση μονάδας «1» σε ένα bit

Για να τοποθετήσουμε μονάδα «1» σε ένα bit ενός δυαδικού αριθμού χωρίς να επηρεαστούν τα υπόλοιπα πρέπει να εκτελέσουμε λογικό "**Η**" μεταξύ του αριθμού και μίας **μάσκας** που πρέπει να έχει "1" στη θέση που πρέπει να γίνει "1" και "0" σε όλες τις υπόλοιπες θέσεις.

Παράδειγμα : Ας υποθέσουμε ότι ο καταχωρητής AL έχει το παρακάτω περιεχόμενο :

	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	
(AL) :	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**→ $(12)_{16}$**

και ότι θέλουμε να τοποθετήσουμε μονάδα στο bit 6 χωρίς να αλλάξουμε τις τιμές των υπολοίπων.
Για να γίνει αυτό κάνουμε λογικό "OR" μεταξύ αυτού του αριθμού και της μάσκας **«01000000»** που έχει το bit 6 μονάδα και όλα τα άλλα bits μηδέν.

```
      00010010                 (12)16
OR    01000000   ή     OR      (40)16
      01010010                 (52)16
```

Ο πίνακας αληθείας του λογικού "Η" (OR) είναι ο παρακάτω:

X	Y	X (OR) Y
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**

γ) Συμπλήρωμα ενός bit ως πρός 1 (0→1, 1→0)

Εκμεταλλευόμενοι τις ιδιότητες της πράξης XOR (Αριθμός XOR 11111111 = NOT Αριθμός, Αριθμός XOR 00000000 = Αριθμός), μπορούμε να αντιστρέψουμε ένα bit ενός δυαδικού αριθμού χωρίς να επηρεαστούν τα υπόλοιπα. Για να γίνει αυτό πρέπει να εκτελέσουμε λογικό "**Αποκλειστικό Ή**" (XOR) μεταξύ του αριθμού και μίας **μάσκας** που πρέπει να έχει "1" στη θέση που πρέπει να γίνει η αντιστροφή του bit και "0" σε όλες τις υπόλοιπες θέσεις.

Παράδειγμα : Ας υποθέσουμε ότι ο καταχωρητής AL έχει το παρακάτω περιεχόμενο :

	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	
(AL) :	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**→ $(BA)_{16}$**

και ότι θέλουμε να αντιστρέψουμε το bit 4 του αριθμού, χωρίς να αλλάξουμε τις τιμές των υπολοίπων.

Για να γίνει αυτό κάνουμε λογικό "XOR" μεταξύ αυτού του αριθμού και της μάσκας **«00010000»** που έχει το bit 4 μονάδα και όλα τα άλλα bits μηδέν.

```
      10111010                    (BA)16
XOR   00010000   ή        OR      (10)16
      10101010                    (AA)16
```

Ο πίνακας αληθείας του λογικού "Αποκλειστικού Η" (XOR) είναι ο παρακάτω:

X	Y	X (XOR) Y
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**0**

Έλεγχος τιμής ενός bit

Για να ελέγξουμε αν ένα bit ενός αριθμού είναι «0» ή «1», ανεξάρτητα από τις τιμές των υπόλοιπων bits, αρκεί να κάνουμε λογικό AND (ΚΑΙ) μεταξύ του αριθμού και μίας **μάσκας** που θα έχει σε όλες τις θέσεις «0» εκτός από τη θέση που μας ενδιαφέρει, όπου η μάσκα θα έχει «1». Έτσι λόγω των «0» όλα τα υπόλοιπα bits θα μηδενιστούν, και το αν το τελικό αποτέλεσμα θα είναι ολόκληρο ίσο με μηδέν ή ένας αριθμός διάφορος του μηδενός (δύναμη του 2), θα εξαρτηθεί από την τιμή του bit που μας ενδιαφέρει.

Παράδειγμα : Ας υποθέσουμε ότι ο καταχωρητής AL έχει το παρακάτω περιεχόμενο :

	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	
(AL) :	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**→ $(57)_{16}$**

και ότι θέλουμε να ελέγξουμε αν το δυαδικό ψηφίο B_2 είναι 0 ή 1, ανεξάρτητα από τις τιμές των υπολοίπων bits. Θα πρέπει να κάνουμε ένα λογικό "**ΚΑΙ**" μεταξύ του καταχωρητή AL και μίας **μάσκας** που θα έχει όλα τα ψηφία του "0" εκτός από το B_2, που θα είναι "1", δηλαδή με τον $(00000100)_2$ ή **$(04)_{16}$**. Έτσι το πρόγραμμα θα κάνει την εξής πράξη:

```
      01010101                    (57)16
AND   00000100   ή        AND     (04)16
      00000100                    (04)16
```

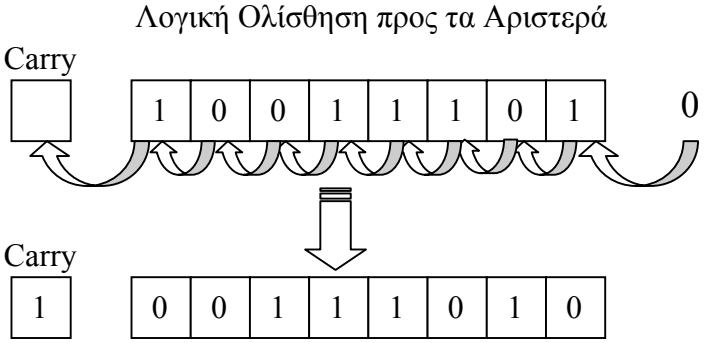
Το αποτέλεσμα είναι διάφορο του 0 άρα το bit 2 είναι μονάδα. Άν ήταν «0» τότε θα είχαμε:

```
      01010001                    (53)16
AND   00000100   ή        AND     (04)16
      00000000                    (00)16
```

Το αποτέλεσμα είναι 0 άρα το bit 2 ήταν «0».

Εντολές ολίσθησης και περιστροφής

Η Assembly του 8088 μας παρέχει επίσης μία ομάδα εντολών για ολίσθηση και περιστροφή των bits ενός καταχωρητή. Κατά την ολίσθηση τα bits μετακινούνται κατά μία θέση αριστερά ή δεξιά, όπως φαίνεται στο σχήμα 9.1. Τα bits που υπερχειλίζουν πηγαίνουν στο Carry, ενώ ο αριθμός συμπληρώνεται συνήθως με «0».



Σχήμα 9.1. Λογική Ολίσθηση προς τα αριστερά. Το υπερχειλίζον bit πηγαίνει στο Carry, και ο αριθμός συμπληρώνεται από δεξιά με μηδενικά.

Με ολίσθηση αριστερά μπορούμε να φτιάξουμε εύκολα μία **μάσκα** σαν αυτές που χρειάζονται για τις εργασίες με bits. Για παράδειγμα :

Για να φτιάξουμε την παρακάτω μάσκα για μηδενισμό του bit 4:

11101111

ξεκινάμε από την μονάδα

00000001

την οποία ολισθαίνουμε αριστερά 4 φορές και έχουμε :

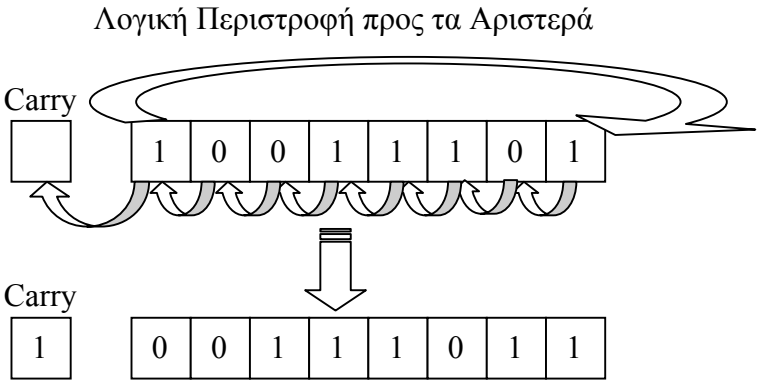
00010000

και την οποία τελικά αντιστρέφουμε με NOT και παίρνουμε :

11101111

Επίσης η ολίσθηση προς τα αριστερά πολλαπλασιάζει τον αριθμό επί την βάση του συστήματος αρίθμησης (επί 2), ενώ η ολίσθηση δεξιά διαιρεί τον αριθμό δια την βάση του συστήματος αρίθμησης (δια 2).

Κατά την **περιστροφή** τα bits μετακινούνται κατά μία θέση αριστερά ή δεξιά, όπως φαίνεται στο σχήμα 9.2. Τα bits που υπερχειλίζουν επανεισάγονται από την άλλη μεριά, ενώ αντιγράφονται και στο Carry.



Σχήμα 9.2. Λογική Περιστροφή προς τα αριστερά. Το υπερχειλίζον bit επανεισάγεται από δεξιά και αντιγράφεται στο Carry.

Η ολίσθηση και η περιστροφή μπορούν να χρησιμεύσουν στις εξής περιπτώσεις

α) Δημιουργία μασκών για εργασίες με bits.

β) Πολλαπλασιασμό επί 2 ή δυνάμεις του 2.
γ) Διαίρεση δια 2 ή δυνάμεις του 2.
δ) Μετατροπή παράλληλου σε σειριακό

Χρησιμοποιούμενες εντολές:

- **AND** (AND – Λογικό ΚΑΙ)
- **OR** (OR – Λογικό Ή)
- **XOR** (Exclusive Or – Αποκλειστική Διάζευξη)
- **NOT** (NOT – Λογική Αντιστροφή, συμπλήρωμα ως προς 1)
- **TEST** (TEST – Έλεγχος τιμής bits)
- **SHL/SAL** (Shift Logical/Arithmetic Left – Μετακίνηση των bits αριστερά)
- **SHR** (Shift Logical Right – Λογική Μετακίνηση των bits Δεξιά)
- **SAR** (Shift Arithmetic Right – Αριθμητική Μετακίνηση των bits Δεξιά)
- **ROL** (Rotate Left – Περιστροφή των bits αριστερά)
- **ROR** (Rotate Right – Περιστροφή των bits Δεξιά)
- **RCL** (Rotate Through Carry Flag Left – Περιστροφή των bits αριστερά μέσω του κρατούμενου)
- **RCR** (Rotate Through Carry Flag Right – Περιστροφή των bits δεξιά μέσω του κρατούμενου)

Η εντολή AND και οι τρόποι σύνταξής της
(AND – Λογικό ΚΑΙ)
Πραγματοποιεί λογικό ΚΑΙ (AND) στα περιεχόμενα των δύο τελεσταίων (καταχωρητές/μνήμη) και βάζει το αποτέλεσμα στον πρώτο τελεσταίο.

- AND καταχωρητής1, καταχωρητής2 AND AX,BX
  Πραγματοποιεί λογικό ΚΑΙ (AND) στα περιεχόμενα του καταχωρητή1 και του καταχωρητή 2 και βάζει το αποτέλεσμα στον καταχωρητή 1.
- AND καταχωρητής, [θέση μνήμης] AND AX,[0200]
  Πραγματοποιεί λογικό ΚΑΙ (AND) στα περιεχόμενα του καταχωρητή και της θέσης μνήμης (2 byte), και βάζει το αποτέλεσμα στον καταχωρητή. Αν ο καταχωρητής είναι του ενός byte, π.χ. AL, τότε η λογική πράξη ΚΑΙ γίνεται με 1 byte από τη μνήμη.
- AND [θέση μνήμης], καταχωρητής AND [0300],BX
  Πραγματοποιεί λογικό ΚΑΙ (AND) στα περιεχόμενα του καταχωρητή και της θέσης μνήμης (2 byte), και βάζει το αποτέλεσμα στη θέση μνήμης (και στην επόμενη, low-high bytes). Αν ο καταχωρητής είναι του ενός byte, π.χ. AL, τότε η λογική πράξη ΚΑΙ γίνεται με 1 byte από τη μνήμη, και το αποτέλεσμα αποθηκεύεται σε 1 byte στην μνήμη.
- AND καταχωρητής, τιμή AND CX, FA3D
  Πραγματοποιεί λογικό ΚΑΙ (AND) στον καταχωρητή και την τιμή που δίνουμε, και βάζει το αποτέλεσμα στον καταχωρητή. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX,BX,CX,DX,BP,SI,DI,SP και όχι για καταχωρητές τμημάτων και σημαίες.
- AND BY/WO[διεύθυνση μνήμης], τιμή AND BY[0400], 4C
  Πραγματοποιεί λογικό ΚΑΙ (AND) στο περιεχόμενο της διεύθυνσης μνήμης και την τιμή που δώσαμε και βάζει το αποτέλεσμα στην διεύθυνση μνήμης.

Η εντολή OR και οι τρόποι σύνταξής της
(OR – Λογικό Ή)
Πραγματοποιεί λογικό Ή (OR) στα περιεχόμενα των δύο τελεσταίων (καταχωρητές/μνήμη) και βάζει το αποτέλεσμα στον πρώτο τελεσταίο. Οι τρόποι σύνταξης της OR είναι ακριβώς ίδιοι με τους τρόπους σύνταξης της εντολής AND, δηλαδή :

- OR καταχωρητής1, καταχωρητής2 OR AX,BX
- OR καταχωρητής, [θέση μνήμης] OR AX,[0200]

- OR [θέση μνήμης], καταχωρητής — OR [0300],BX
- OR καταχωρητής, τιμή — OR CX, FA3D
- OR BY/WO[διεύθυνση μνήμης], τιμή — OR BY[0400], 4C

Η εντολή XOR και οι τρόποι σύνταξής της

(Exclusive Or – Αποκλειστική Διάζευξη)

Πραγματοποιεί λογικό Αποκλειστικό Ή (XOR) στα περιεχόμενα των δύο τελεσταίων (καταχωρητές/μνήμη) και βάζει το αποτέλεσμα στον πρώτο τελεσταίο. Οι τρόποι σύνταξης της XOR είναι ακριβώς ίδιοι με τους τρόπους σύνταξης της εντολής AND, δηλαδή :

- XOR καταχωρητής1, καταχωρητής2 — XOR AX,BX
- XOR καταχωρητής, [θέση μνήμης] — XOR AX,[0200]
- XOR [θέση μνήμης], καταχωρητής — XOR [0300],BX
- XOR καταχωρητής, τιμή — XOR CX, FA3D
- XOR BY/WO[διεύθυνση μνήμης], τιμή — XOR BY[0400], 4C

Η εντολή NOT και οι τρόποι σύνταξής της

(Not – Λογική Άρνηση-Αντιστροφή)

Αντιστρέφει τα bit, δηλαδή κάνει τα 0 να γίνουν 1 και τα 1 να γίνουν 0:

- NOT καταχωρητής — NOT DX

  Αντιστρέφει τα bit του καταχωρητή, δηλαδή κάνει τα 0 να γίνουν 1 και τα 1 να γίνουν 0. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX, BX, CX, DX, BP, SI, DI, SP και όχι για καταχωρητές τμημάτων και σημαίες.

Η εντολή TEST και οι τρόποι σύνταξής της

(Test – Λογικό ΚΑΙ χωρίς καταχώρηση αποτελέσματος)

Πραγματοποιεί λογικό ΚΑΙ (AND) στα περιεχόμενα των δύο τελεσταίων, επηρεάζοντας τις σημαίες, αλλά ΧΩΡΙΣ να βάζει το αποτέλεσμα στον πρώτο τελεσταίο. Κατά τα άλλα οι τρόποι σύνταξης της TEST είναι ακριβώς ίδιοι με τους τρόπους σύνταξης της εντολής AND, δηλαδή :

- TEST καταχωρητής1, καταχωρητής2 — TEST AX,BX
- TEST καταχωρητής, [θέση μνήμης] — TEST AX,[0200]
- TEST [θέση μνήμης], καταχωρητής — TEST [0300],BX
- TEST καταχωρητής, τιμή — TEST CX, FA3D
- TEST BY/WO[διεύθυνση μνήμης], τιμή — TEST BY[0400], 4C

Η εντολή SHL/SAL και οι τρόποι σύνταξής της

(Shift Logical/Arithmetic Left – Μετακίνηση των bits αριστερά)

Μετακινεί (ολισθαίνει) τα bit ενός καταχωρητή προς τα αριστερά κατά 1 θέση. Ο καταχωρητής συμπληρώνεται από δεξιά με μηδενικά. Το υπερχειλίζον bit πηγαίνει στο Carry:

- SHL/SAL καταχωρητής,1 — SHL AX,1

  Μετακινεί (ολισθαίνει) τα bit του καταχωρητή προς τα αριστερά κατά 1 θέση. Ο καταχωρητής συμπληρώνεται από δεξιά με μηδενικά. Το υπερχειλίζον bit πηγαίνει στο Carry. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX, BX, CX, DX, BP, SI, DI, SP και όχι για καταχωρητές τμημάτων και σημαίες.
- SHL/SAL καταχωρητής,CL — SHL BX,CL

  Μετακινεί (ολισθαίνει) τα bit του καταχωρητή προς τα αριστερά κατά τόσες θέσεις όσο η τιμή του καταχωρητή CL. Ο καταχωρητής συμπληρώνεται από δεξιά με μηδενικά. Το υπερχειλίζον bit πηγαίνει στο Carry. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX, BX, CX, DX, BP, SI, DI, SP και όχι για καταχωρητές τμημάτων και σημαίες.

Η εντολή SHR και οι τρόποι σύνταξής της

(Shift Logical Right – Λογική Μετακίνηση των bits Δεξιά)

Μετακινεί (ολισθαίνει) τα bit ενός καταχωρητή προς τα δεξιά κατά 1 θέση. Ο καταχωρητής συμπληρώνεται από αριστερά με μηδενικά. Το υπερχειλίζον bit πηγαίνει στο Carry. Κατά τα άλλα οι τρόποι σύνταξης της SHR είναι ακριβώς ίδιοι με αυτούς της SHL/SAL:

- SHR καταχωρητής,1 SHR AX,1
- SHR καταχωρητής,CL SHR BX,CL

Η εντολή SAR και οι τρόποι σύνταξής της

(Shift Arithmetic Right – Αριθμητική Μετακίνηση των bits Δεξιά)

Μετακινεί (ολισθαίνει) τα bit του καταχωρητή προς τα δεξιά κατά 1 θέση. Το σημαντικότερο ψηφίο του αριθμού (most significant bit), που έχει σημασία προσήμου στους προσημασμένους αριθμούς, παραμένει. Το υπερχειλίζον bit πηγαίνει στο Carry. Κατά τα άλλα οι τρόποι σύνταξης της SAR είναι ακριβώς ίδιοι με αυτούς της SHL/SAL:

- SAR καταχωρητής,1 SAR AX,1
- SAR καταχωρητής,CL SAR BX,CL

Η εντολή ROL και οι τρόποι σύνταξής της

(Rotate Left – Περιστροφή των bits αριστερά)

Περιστρέφει τα bit του καταχωρητή προς τα αριστερά κατά 1 θέση. Το bit που υπερχειλίζει επανεισάγεται από δεξιά και αντιγράφεται στο carry. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX, BX, CX, DX, BP, SI, DI, SP και όχι για καταχωρητές τμημάτων και σημαίες. Κατά τα άλλα οι τρόποι σύνταξης της ROL είναι ακριβώς ίδιοι με αυτούς της SHL/SAL:

- ROL καταχωρητής,1 ROL AX,1
- ROL καταχωρητής,CL ROL BX,CL

Η εντολή ROR και οι τρόποι σύνταξής της

(Rotate Right – Περιστροφή των bits δεξιά)

Περιστρέφει τα bit του καταχωρητή προς τα δεξιά κατά 1 θέση. Το bit που υπερχειλίζει επανεισάγεται από αριστερά και αντιγράφεται στο carry. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX, BX, CX, DX, BP, SI, DI, SP και όχι για καταχωρητές τμημάτων και σημαίες. Κατά τα άλλα οι τρόποι σύνταξης της ROR είναι ακριβώς ίδιοι με αυτούς της SHL/SAL:

- ROR καταχωρητής,1 ROR AX,1
- ROR καταχωρητής,CL ROR BX,CL

Η εντολή RCL και οι τρόποι σύνταξής της

(Rotate Through Carry Flag Left – Περιστροφή των bits αριστερά μέσω του κρατούμενου)

Περιστρέφει τα bit του καταχωρητή προς τα αριστερά κατά 1 θέση. Το bit που υπερχειλίζει πηγαίνει στο Carry και το Carry (αρχική τιμή) επανεισάγεται από δεξιά. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX, BX, CX, DX, BP, SI, DI, SP και όχι για καταχωρητές τμημάτων και σημαίες. Κατά τα άλλα οι τρόποι σύνταξης της RCL είναι ακριβώς ίδιοι με αυτούς της SHL/SAL:

- RCL καταχωρητής,1 RCL AX,1
- RCL καταχωρητής,CL RCL BX,CL

Η εντολή RCR και οι τρόποι σύνταξής της

(Rotate Through Carry Flag Right – Περιστροφή των bits δεξιά μέσω του κρατούμενου)

Περιστρέφει τα bit του καταχωρητή προς τα δεξιά κατά 1 θέση. Το bit που υπερχειλίζει πηγαίνει στο Carry και το Carry (αρχική τιμή) επανεισάγεται από αριστερά. Προσοχή, λειτουργεί μόνο για τους καταχωρητές AX, BX, CX, DX, BP, SI, DI, SP και όχι για καταχωρητές τμημάτων και σημαίες. Κατά τα άλλα οι τρόποι σύνταξης της RCR είναι ακριβώς ίδιοι με αυτούς της SHL/SAL:

- RCR καταχωρητής,1 RCR AX,1
- RCR καταχωρητής,CL RCR BX,CL

ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Άσκηση 9.1: Να γραφεί πρόγραμμα το οποίο θα μετατρέπει έναν αριθμό που βρίσκεται στη διεύθυνση μνήμης **0200** σε δυαδικό αριθμό βάζοντας κάθε ψηφίο του δυαδικού αριθμού σε διαδοχικά bytes με αρχή την διεύθυνση μνήμης **0208** (0208..020F), ώστε ο δυαδικός αριθμός να είναι αναγνώσιμος από τον χρήστη με την εντολή D200.
(Βοηθητική ανάλυση : αρχικά φτιάξτε μία μάσκα με τιμή 10000000 η οποία σε κάθε επανάληψη θα ολισθαίνει δεξιά. Κάθε φορά συγκρίνετε τη μάσκα με τον αριθμό και αναλόγως τοποθετείτε 0 ή 1 στην κατάλληλη θέση μνήμης)

Άσκηση 9.2: Να γραφεί πρόγραμμα το οποίο θα μετατρέπει έναν αριθμό που βρίσκεται αναπτυγμένος ως δυαδικός αριθμός στις διευθύνσεις μνήμης **0208..020F** (ένα ψηφίο σε κάθε byte) σε κανονικό ακέραιο αριθμό του ενός byte που θα τοποθετείται στη διεύθυνση μνήμης **0210.**
(Βοηθητική ανάλυση : αρχικά φτιάξτε μία μάσκα με τιμή 10000000 η οποία σε κάθε επανάληψη θα ολισθαίνει δεξιά και μηδενίστε ένα καταχωρητή ή απευθείας τη θέση μνήμης. Αν η κατάλληλη θέση μνήμης έχει «1» τότε προστίθεται λογικά το bit της μάσκας στον καταχωρητή/μνήμη.)

10ο ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΑΘΗΜΑ

ΑΝΤΙΚΕΙΜΕΝΟ : Προγραμματισμός θυρών εισόδου / εξόδου
Εντολές IN, OUT
Το Status Port του BGC-8088
Εκτέλεση προγραμμάτων για προγραμματισμό του Status Port (LEDS, SPEAKER, κ.λ.π.)

ΘΕΩΡΗΤΙΚΟ ΜΕΡΟΣ

Ένας υπολογιστής, εκτός από τον επεξεργαστή και την μνήμη, περιλαμβάνει και μονάδες εισόδου / εξόδου (I/O), οι οποίες του επιτρέπουν να επικοινωνεί και να ανταλλάσσει δεδομένα με τον υπόλοιπο κόσμο, δηλαδή με άλλες ψηφιακές συσκευές και διατάξεις. Η επικοινωνία με τις μονάδες I/O γίνεται μέσω του Διαύλου Διευθύνσεων (Address Bus) και του Διαύλου Δεδομένων (Data Bus), όπως ακριβώς και με την μνήμη, αλλά μέσω ειδικών εντολών γλώσσας μηχανής.
Έτσι όταν ο 8088 θέλει να επικοινωνήσει με μία μονάδα I/O, απλά σχηματίζει στο Address Bus την διεύθυνση που αντιστοιχεί στην συγκεκριμένη συσκευή, και μέσω του Data Bus μπορεί να στείλει ή να λάβει δεδομένα από την περιφερειακή συσκευή (Π.Σ).
Εφόσον λοιπόν η επικοινωνία με τις Π.Σ. γίνεται μέσω των ίδιων διαύλων όπως και η επικοινωνία με την μνήμη, τίθεται το ερώτημα πώς γίνεται ο διαχωρισμός αυτών των δύο ειδών επικοινωνίας που λαμβάνουν χώρα μέσω των ίδιων διαύλων.
Για παράδειγμα, έστω ότι ο 8088 θέλει να λάβει ένα byte από την συσκευή που βρίσκεται στη διεύθυνση 0FF00. Απλά σχηματίζει στο Address Bus τον αριθμό 0FF00 δηλαδή «00001111111100000000» και λαμβάνει μέσω του Data Bus τον αριθμό που στέλνει η Π.Σ. Σχηματίζοντας όμως μία διεύθυνση στο Address Bus, η διεύθυνση αυτή οδηγείται και στα chip μνήμης, με αποτέλεσμα να πάρουμε στο Data Bus το περιεχόμενο της διεύθυνσης μνήμης 0FF00 !
Έτσι λοιπόν γίνεται αντιληπτό ότι πρέπει να υπάρχει μία μέθοδος διαχωρισμού της πρόθεσης του 8088 να επικοινωνήσει με την μνήμη ή με μονάδες I/O, έτσι ώστε να μην υπάρχουν συγκρούσεις δεδομένων πάνω στο Data Bus. Η μέθοδος που ακολουθείται στον 8088 βασίζεται στην ύπαρξη τριών pin του 8088 με τα ονόματα S0, S1 και S2. Ο συνδυασμός τιμών των τριών αυτών pins, μας δίνει 2^3=8 διαφορετικά σήματα ελέγχου τα οποία είναι :

S2	S2	S0	Σήμα Ελέγχου
0	0	0	Interrupt Acknowledge (Αποδοχή Διακοπής)
0	0	1	Read I/O Port (Ανάγνωση Θύρας I/O)
0	1	0	Write I/O Port (Εγγραφή Θύρας I/O)
0	1	1	Halt (Διακοπή λειτουργίας)
1	0	0	Code Access (Προσκόμιση opcode)
1	0	1	Read Memory (Ανάγνωση Μνήμης)
1	1	0	Write Memory (Εγγραφή Μνήμης)
1	1	1	Passive (Καμία Λειτουργία)

Έτσι όταν ο 8088 διαβάζει από την μνήμη θέτει στα pins S2, S1, S0 τις τιμές 1, 0, 1. Οι τιμές αυτές αποκωδικοποιούνται από decoders παράγοντας το σήμα R/W (ή RE, WE) προς την μνήμη, το οποίο

και ενεργοποιεί στην ουσία τα chip μνήμης, ώστε λαμβάνοντας την διεύθυνση από το Address Bus, να επιλέξουν το κατάλληλο byte και να το εξάγουν προς το Data Bus.
Όταν όμως ο 8088 διαβάζει από Π.Σ. θέτει στα pins S2, S1, S0 τις τιμές 0, 0, 1. Οι τιμές αυτές δεν ενεργοποιούν τα chip μνήμης αλλά τα chip των περιφερειακών συσκευών, έτσι ώστε με τον σχηματισμό της διεύθυνσης στο Address Bus, να επιλέγεται η κατάλληλη συσκευή (ο κατάλληλος καταχωρητής μίας συσκευής) του οποίου το περιεχόμενο εν συνεχεία εξάγεται προς το Data Bus.
Γίνεται λοιπόν φανερό ότι οι Π.Σ. και η μνήμη μοιράζονται κοινές διευθύνσεις σε έναν υπολογιστή.

Ο 8088 μπορεί να επικοινωνήσει με 65536 (2^{16}) ή $FFFF_{16}$ διαφορετικές θύρες I/O (καταχωρητές συσκευών εισόδου / εξόδου). Για την επιλογή θύρας I/O χρησιμοποιούνται τα 16 λιγότερο σημαντικά ψηφία της διεύθυνσης (A0..A15) ενώ τα 4 περισσότερο σημαντικά (A16..A19) παραμένουν ίσα με 0.
Στον BGC-8088 υπάρχουν 8 συσκευές εισόδου / εξόδου στις οποίες έχουν παραχωρηθεί 16 διευθύνσεις για κάθε συσκευή, και άρα υπάρχουν συνολικά 128 διευθύνσεις καταχωρητών I/O. Οι διευθύνσεις αυτές είναι στην περιοχή 0FF00 .. 0FF7F. Η αντιστοιχία διευθύνσεων και συσκευών είναι η εξής :

Περιοχή Διευθύνσεων	Συσκευή	Επεξήγηση
FF00 – FF0F	8254	Programmable Timer (PT). Παράγει σήματα χρονισμού των οποίων η συχνότητα αλλά και η κυματομορφή είναι προγραμματιζόμενες. Υλοποιεί προγραμματιζόμενους μετρητές – counters.
FF10 – FF1F	8255A	Programmable Peripheral Interface (PPI). Διαθέτει 3 παράλληλες θύρες των 8–bits η καθεμιά, οι οποίες είναι πλήρως ανεξάρτητες και πλήρως προγραμματιζόμενες ως προς το αν θα λειτουργούν ως είσοδοι, ως έξοδοι ή ως δίαυλοι ελέγχου.
FF20 – FF2F	8259A	Programmable Interrupt Controller (PIC). Επιτρέπει την σύνδεση 8 γραμμών interrupt από 8 Π.Σ. στην μοναδική IRQ γραμμή του 8088. Οι 8 γραμμές IRQ έχουν διαβαθμισμένες προτεραιότητες. Στέλνει στον 8088 μέσω του Data Bus τον Α/Α της συσκευής (0..7) που προκάλεσε το interrupt (Διανυσματική Διακοπή).
FF30 – FF3F	PRINTER	Η παράλληλη θύρα του εκτυπωτή υλοποιείται με ένα chip 74LS138 που αποτελεί ένα decoder, ένα 74LS374 που αποτελεί ένα 8-bit Data Latch (buffer), και ένα 74LS174 που αποτελεί ένα 4-bit Control Latch (buffer σημάτων ελέγχου).
FF40 – FF4F	LCD	Η LCD οθόνη οδηγείται τοποθετώντας εντολές στον Instruction Register του LCD και δεδομένα στον Data Register και προσπελαύνοντας την εσωτερική RAM του LCD. Διαθέτει 11 διαφορετικές εντολές.
FF50 – FF5F	8279	Ελεγκτής πληκτρολογίου και οθόνης LED (7-segment LED) που χρησιμοποιείται για σάρωση πληκτρολογίου και αποθήκευση

		των κωδικών των χαρακτήρων σε buffer με δομή ουράς FIFO.
FF60 – FF6F	NS 8250	RS-232C Interface. Υλοποιεί την σειριακή θύρα του BGC-8088
FF70 – FF7F	STATUS PORT	Αποτελείται από ένα 8-bit data latch (buffer) εκ των οποίων τα 3 bits οδηγούν τα LED "Caps", "Ins", "Print" του πληκτρολογίου καθώς και το μεγαφωνάκι του BGC-8088

Κάθε μία από τις 8 συσκευές I/O του BGC-8088 προγραμματίζεται με διαφορετικό τρόπο, καθώς η κάθε μία υλοποιεί έναν αριθμό από καταχωρητές στους οποίους ο προγραμματιστής θα πρέπει να καταχωρεί τις κατάλληλες τιμές ώστε να χειρίζεται με επιτυχία την συσκευή I/O.
Ο προγραμματισμός όμως όλων των συσκευών I/O γίνεται μέσω ειδικών εντολών γλώσσας μηχανής που είναι :

Χρησιμοποιούμενες εντολές:

- **OUT** (Output to port – Έξοδος δεδομένων προς θύρα Εξόδου)
- **IN** (Input from port – Είσοδος δεδομένων από θύρα εισόδου)

Η εντολή OUT και οι τρόποι σύνταξής της
(Output to port – Έξοδος δεδομένων προς θύρα Εξόδου)
Εξάγει στην θύρα εξόδου το περιεχόμενο του καταχωρητή AL. Η θύρα καθορίζεται δίνοντας την διεύθυνσή της ως παράμετρο της εντολής.

- OUT <αριθμός θύρας>, AL OUT A4, AL
  Εξάγει το byte που περιέχεται στον AL, στην θύρα που προσδιορίζεται από τον <αριθμό θύρας>. Η εντολή μπορεί να εφαρμοστεί μόνο για τις θύρες 00..FF (0000 .. 00FF).
- OUT DX, AL OUT DX, AL
  Εξάγει το byte που βρίσκεται στο καταχωρητή AL στην θύρα εξόδου που δίνουμε ως τιμή του καταχωρητή DX. Χρησιμοποιείται όταν έχουμε μέχρι 65536 θύρες I/O στο σύστημα (όπως στα PC και τον BGC-8088). Παράδειγμα :
  MOV AL, FE
  MOV DX, FF70
  OUT DX, AL

Η εντολή IN και οι τρόποι σύνταξής της
(Input from port – Είσοδος δεδομένων από θύρα εισόδου)
Διαβάζει το byte που βρίσκεται στην θύρα εισόδου που δίνουμε και το τοποθετεί στον καταχωρητή AL.

- IN AL, <αριθμός θύρας> IN AL, 5C
  Διαβάζει το byte που βρίσκεται στην θύρα εισόδου που προσδιορίζεται από τον <αριθμό θύρας>, και το τοποθετεί στον AL. Η εντολή μπορεί να εφαρμοστεί μόνο για τις θύρες 00..FF (0000 .. 00FF).
- IN AL, DX IN AL,DX
  Διαβάζει το byte που βρίσκεται στην θύρα εισόδου που δίνουμε ως τιμή του καταχωρητή DX, και το τοποθετεί στον AL. Χρησιμοποιείται όταν έχουμε μέχρι 65536 θύρες I/O στο σύστημα (όπως στα PC και τον BGC-8088). Παράδειγμα :
  MOV DX, FF6D
  IN AL, DX

Περισσότερα για το Status Port

Το Status Port στην ουσία αποτελείται από ένα 8-bit data latch (buffer) 74LS374 που είναι χαρτογραφημένο στην διεύθυνση FF70. Από τα 8 bits του latch χρησιμοποιούνται μόνο τα 4. Η σημασία των bits είναι η ακόλουθη :

Bit :	7	6	5	4	3	2	1	0
Σημασία:	X	X	X	X	Speaker	Print led	Ins Led	Caps lock

Bit 0 : όταν έχει την τιμή 0 το LED για το Caps Lock (Κλείδωμα Κεφαλαίων) ανάβει. Όταν πάρει την τιμή 1 τότε το LED σβήνει. Αλλάζει κατάσταση πατώντας το πλήκτρο "CAPS LOCK" του πληκτρολογίου.
Bit 1 : όταν έχει την τιμή 0 το LED για το "INS" (κατάσταση εισαγωγής εντολών) ανάβει. Όταν πάρει την τιμή 1 τότε το LED σβήνει. Αλλάζει κατάσταση πατώντας το πλήκτρο "INS" του πληκτρολογίου, ή δίνοντας την εντολή "I" του MONITOR.
Bit 2 : όταν έχει την τιμή 0 το LED για το PRINT (Εκτύπωση) ανάβει. Όταν πάρει την τιμή 1 τότε το LED σβήνει. Αλλάζει κατάσταση πατώντας τον συνδυασμό πλήκτρων Ctrl+P του πληκτρολογίου.
Bit 3 : Το bit αυτό οδηγείται σε ένα μεγαφωνάκι (buzzer). Η συχνότητα και η χροιά του ήχου που θα ακουστεί καθορίζεται από την διάρκεια που το bit θα είναι 0, και την διάρκεια που το bit θα είναι 1. Η εναλλαγή του 0 και του 1 παράγουν στη ουσία μία τετραγωνική κυματομορφή η οποία όταν οδηγείται στο μεγαφωνάκι μετατρέπεται σε ήχο. Η ένταση του ήχου είναι σταθερή. Ένα απλό παράδειγμα οδήγησης του μεγαφώνου είναι το ακόλουθο :

```
0100:0000   MOV DX,FF70     Η διεύθυνση το καταχωρητή του Status Port
0100:0003   MOV AL,0        Αρχική τιμή 0 σε όλα τα bits (και στο bit 3)
0100:0005   OUT DX,AL       Έξοδος του αριθμού στον καταχωρητή του Status Port
0100:0006   MOV CX,0100     Μετρητής καθυστέρησης
0100:0009   LOOP 0009       Βρόχος καθυστέρησης
0100:000B   MOV AL,8        Βάλε «1» στο bit 3
0100:000D   OUT DX,AL       Έξοδος του αριθμού στον καταχωρητή του Status Port
0100:000E   MOV CX,0100     Μετρητής καθυστέρησης
0100:0011   LOOP 0011       Βρόχος καθυστέρησης
0100:0013   JMP 0003        Επανάληψη επ' άπειρον
```

ΕΡΓΑΣΤΗΡΙΑΚΟ ΜΕΡΟΣ

Άσκηση 10.1: Να αναπτυχθεί πρόγραμμα που θα παράγει έναν ήχο από το μεγαφωνάκι του BGC-8088. Ο ήχος να ξεκινάει από χαμηλή συχνότητα (μπάσος) και σταδιακά να αυξάνει συχνότητα (πρίμα). Στη συνέχεια και αφού φτάσει σε μία μέγιστη συχνότητα, να ξαναελαττώνει την συχνότητα του προς χαμηλότερες συχνότητες (ανέβασμα και κατέβασμα κλίμακας). Η διαδικασία αυτή να επαναλαμβάνεται 10 φορές.

Άσκηση 10.2: Να αναπτυχθεί πρόγραμμα που θα προκαλεί το σειριακό άναμμα και σβήσιμο των LEDS του πληκτρολογίου (CAPS, INS, PRINT) με μεταβλητό ρυθμό ο οποίος θα ξεκινάει αργά και θα επιταχύνει σταδιακά. Όταν φτάσει την μέγιστη συχνότητα θα επιβραδύνει και πάλι μέχρι το κάτω όριο (αργός ρυθμός). Η διαδικασία αυτή να επαναληφθεί 3 φορές.