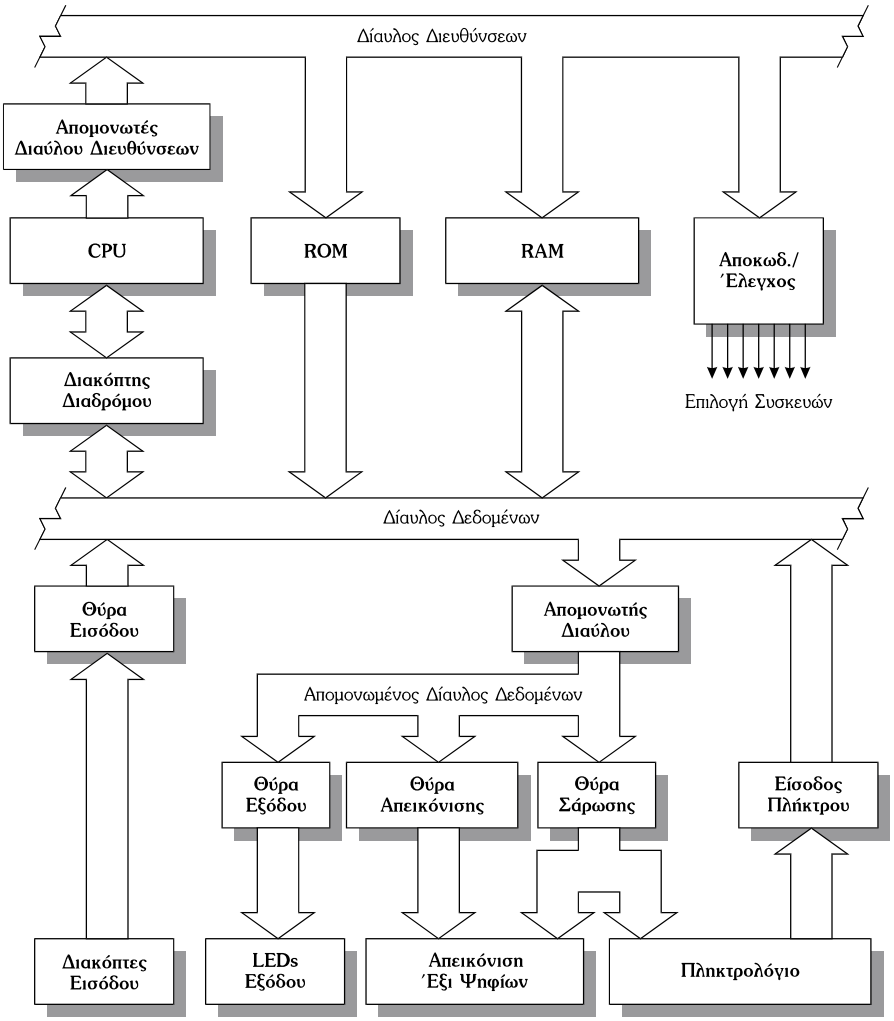
Κ. Ζ. ΠΕΚΜΕΣΤΖΗ
ΚΑΘΗΓΗΤΗΣ Ε.Μ.Π.

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ



ΑΘΗΝΑ 2011

Πρόλογος

Το βιβλίο αυτό είναι το βοήθημα του Εργαστηριακού μαθήματος "Εργαστήριο Μικροϋπολογιστών" στο 8ο εξάμηνο στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Π. Συνεπώς, η οργάνωσή του βασίζεται στη σειρά των ασκήσεων που πραγματοποιούνται στο εργαστήριο. Για να είναι το βιβλίο πιο εύχρηστο έχουν προστεθεί παραρτήματα για γρήγορη αναφορά.

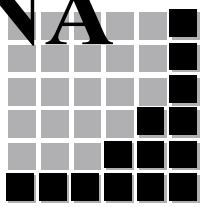
Στο βιβλίο αυτό έχουν συμβάλει κατά τα παλαιότερα έτη στο διάστημα μέχρι την τελική διαμόρφωση αρκετοί σπουδαστές. Από τη θέση αυτή θα ήθελα να τους εκφράσω τις ευχαριστίες μου.

Για τις παραπάνω εργασίες τους θα ήθελα να εκφράσω στον καθένα ξεχωριστά τις πιο θερμές μου ευχαριστίες.

Τέλος, θα ήθελα να ζητήσω την κατανόηση του αναγνώστη για πιθανά λάθη και παραλείψεις που μπορεί να έχουν υπεισέλθει στην έκδοση αυτή.

Κ. ΠΕΚΜΕΣΤΖΗ

ΠΕΡΙΕΧΟΜΕΝΑ



ΕΙΣΑΓΩΓΗ ΣΤΟ μLAB 1

1. Γνωριμία με το μLab 2
2. Χάρτης μνήμης του μLab 7
3. Block διάγραμμα του μLab 8
4. Hardware του μLab 9

1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 35

1. Βασικές γνώσεις για την εκτέλεση προγραμμάτων στο μLab..... 36
2. Τα θέματα της 1ης εργαστηριακής άσκησης..... 43

2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 45

1. Στοιχεία θεωρίας 46
2. Παραδείγματα προγραμμάτων 48
3. Τα θέματα της 2ης εργαστηριακής άσκησης..... 59

3η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 61

1. Χάρτης καταχωρητών και διακοπών 62
2. Κατηγορίες εντολών 65
3. Τεχνικές προγραμματισμού 69
4. Τα θέματα της 3ης εργαστηριακής άσκησης 77

4η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 81

1. Έλεγχος των περιφερειακών μέσω software 82
2. Έλεγχος του πληκτρολογίου με τις ρουτίνες του μLab 82
3. Απευθείας έλεγχος του πληκτρολογίου 84
4. Έλεγχος της οθόνης 7 τμημάτων με τις ρουτίνες του μLab 86
5. Απευθείας έλεγχος των οθονών 7 τμημάτων 89
6. Τα θέματα της 4ης εργαστηριακής άσκησης 95

5η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 97

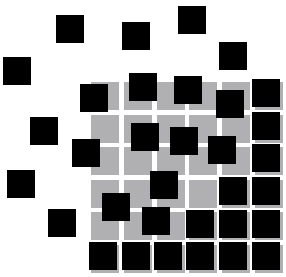
1. Περιγραφή MASM 98
2. Δημιουργώντας μια βιβλιοθήκη 112
3. Τα θέματα της 5ης εργαστηριακής άσκησης 114

6η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 117

1. Σύνδεση ρουτινών MASM με γλώσσες υψηλού επιπέδου 118
2. Ασύγχρονη επικοινωνία μέσω software σε προσωπικό υπολογιστή 133
3. Τα θέματα της 6ης εργαστηριακής άσκησης 145

ΠΑΡΑΡΤΗΜΑΤΑ 147

Παράρτημα 1: Έτοιμα προγράμματα στη ROM του μLab 147
Παράρτημα 2: Πίνακας αναφοράς κώδικα assembly 8085A 151
Παράρτημα 3: Πλήρες listing της ROM του μLab 159
Παράρτημα 4: Οι Εντολές του μΕ 80x86 205
Παράρτημα 5: Ρουτίνες BIOS και DOS 235
Παράρτημα 6: Πίνακας Χαρακτήρων ASCII 253



ΕΙΣΑΓΩΓΗ ΣΤΟ μLAB

(Αφορά τις ασκήσεις 1-4)

- 1. Γνωριμία με το μLab
- 2. Χάρτης μνήμης του μLab
- 3. Block διάγραμμα του μLab
- 4. Hardware του μLab

1. Γνωριμία με το μLab

Το μLab είναι ένας μικροϋπολογιστής σχεδιασμένος ειδικά για εκπαιδευτικούς σκοπούς. Βασίζεται στο μικροεπεξεργαστή 8085 της Intel και περιλαμβάνει μνήμη ROM 2K bytes και RAM 1K bytes.

Το σύστημα βρίσκεται μέσα σε ένα πρακτικό βαλιτσάκι. Ανοίγοντάς το, βλέπουμε την πλακέτα που φιλοξενεί τα διάφορα κυκλώματα. Πάνω της αναγράφονται τα ονόματά τους, ενώ η λειτουργία τους υποδηλώνεται με βέλη που συμβολίζουν τη ροή των δεδομένων.

Η πλακέτα περιλαμβάνει ένα πληκτρολόγιο (keyboard) για την εισαγωγή προγραμμάτων, φύλαξη δεδομένων και εκτέλεση εντολών ελέγχου του συστήματος καθώς και ένα display για την απεικόνιση των διευθύνσεων μνήμης ή των καταχωρητών του 8085 με τα περιεχόμενά τους. Επίσης, περιλαμβάνει μία πόρτα εξόδου αποτελούμενη από 8 LEDs, ένα για κάθε γραμμή εξόδου, και μία πόρτα εισόδου μέσω 8 μικροδιακοπτών (dip switches), ένα για κάθε γραμμή εισόδου.

Ένα μεγάφωνο ενσωματωμένο στην πλακέτα, ελέγχεται από το μικροϋπολογιστή. Τέλος, υπάρχουν LEDs για το address bus, το data bus και τις βασικές γραμμές ελέγχου του συστήματος έτσι ώστε κάθε στιγμή να μπορεί ο χρήστης να ελέγχει τη δραστηριότητά του.

Η μνήμη ROM περιλαμβάνει προγράμματα για την ανάγνωση του πληκτρολογίου, την εκτέλεση των εντολών από τα πλήκτρα λειτουργιών και την απεικόνιση των δεδομένων στα displays. Ολόκληρο το πρόγραμμα λειτουργίας του συστήματος καλείται monitor. Οι εφαρμογές που ακολουθούν έχουν σκοπό την επίδειξη των χαρακτηριστικών του monitor και του hardware που αυτό ελέγχει. Περισσότερες λεπτομέρειες θα δοθούν στις διάφορες εργαστηριακές ασκήσεις.

Αφού ανοίξετε το βαλιτσάκι, πατήστε το διακόπτη στο πλάι του μLab στη θέση ON. Τα displays και τα LEDs εξόδου φωτίζονται για ένα περίπου δευτερόλεπτο και ένας χαρακτηριστικός ήχος ακούγεται υποδηλώνοντας τον τερματισμό του self-test του συστήματος. Το display αναγράφει 'uLAb UP' και το σύστημα είναι έτοιμο για εισαγωγή εντολών. Στο δεξί μέρος του πληκτρολογίου υπάρχει η αριθμητική πληκτροπινακίδα για είσοδο δεκαεξαδικών (hex) δεδομένων ενώ στο αριστερό υπάρχουν τα πλήκτρα ειδικών λειτουργιών, των οποίων η λειτουργία εξηγείται παρακάτω:

RESET

Επαναφέρει το σύστημα στην αρχική του κατάσταση από οποιοδήποτε σημείο βρισκόμαστε. Μετά το πάτημα του πλήκτρου, η οθόνη του μLab πρέπει να δείχνει 'uLAb UP', που σημαίνει ότι είναι έτοιμο να δεχτεί μια καινούρια εντολή.

RUN

Με το πάτημά του αρχίζει η εκτέλεση των εντολών από το σημείο που δείχνει ο μετρητής προγράμματος (PC) τη στιγμή που πατιέται.

HDWR STEP

Με το πάτημά του εκτελείται ένας κύκλος μηχανής της εντολής που δείχνει ο PC. Με διαδοχικά πατήματα μπορούμε να βρούμε από πόσους κύκλους αποτελείται μια εντολή και τι μικρολειτουργίες εκτελεί.

INSTR STEP

Με το πάτημά του εκτελείται η εντολή στην οποία δείχνει ο PC τη στιγμή που πατιέται.

INTRP

Με το πάτημά του προκαλούμε στο μικροεπεξεργαστή μια διακοπή RST6.5 με αποτέλεσμα να τον αναγκάζουμε να μεταφέρει τον έλεγχο από οποιοδήποτε σημείο βρίσκεται σε μια προκαθορισμένη διεύθυνση στη RAM όπου είναι αποθηκευμένη η ρουτίνα εξυπηρέτησης διακοπής.

FETCH PC

Με το πάτημά του εμφανίζεται στο display η διεύθυνση της επόμενης προς εκτέλεση εντολής (το περιεχόμενο του PC).

FETCH REG

Πατώντας διαδοχικά το πλήκτρο αυτό μπορούμε να δούμε τα περιεχόμενα όλων των καταχωρητών.

FETCH ADRS

Πατώντας το πλήκτρο αυτό καθαρίζει το display και το μLab περιμένει την εισαγωγή μιας διεύθυνσης από την αριθμητική πληκτροπινακίδα. Όταν συμπληρωθεί η εισαγωγή της διεύθυνσης μεταφέρεται ο PC στην εισαχθείσα διεύθυνση και εμφανίζεται στα δύο δεξιότερα Leds το περιεχόμενό της.

STORE INCR

Πατώντας το, το μLab δείχνει την επόμενη διεύθυνση μνήμης την οποία μπορούμε είτε να εποπτεύσουμε είτε να τροποποιήσουμε (το περιεχόμενό της).

DECR

Πατώντας το μπορούμε να εποπτεύσουμε τα περιεχόμενα της μνήμης του μLab οπισθοδρομώντας.

Για το χειρισμό και την κατανόηση της λειτουργίας του μLab υπάρχουν πολύ χρήσιμες πληροφορίες στα παραρτήματα, στο τέλος του βιβλίου. Στο παράρτημα 1 δίνονται οι διευθύνσεις της ROM στις οποίες υπάρχουν έτοιμα προγράμματα (παιχνίδια) όπως και χρήσιμες ρουτίνες που μπορείτε να χρησιμοποιήσετε στη συνέχεια. Στο παράρτημα 2 δίνονται σε συγκεντρωτική μορφή οι κώδικες των εντολών του 8085. Τέλος, στο παράρτημα 3 δίνεται το πλήρες listing της ROM.

Έχοντας όλα αυτά υπ' όψη μπορούμε να εκτελέσουμε μερικές απλές, εισαγωγικές εφαρμογές. Ακολουθήστε τις παρακάτω υποδείξεις:

1. Πατήστε [FETCH ADRS]. Οι υπογραμμίσεις στα displays υποδηλώνουν ότι το μLab περιμένει από σας κάποια διεύθυνση.
2. Πληκτρολογήστε [05F9].
3. Πατήστε [RUN]. Εκκινήσατε το πρόγραμμα που βρίσκεται στη διεύθυνση 05F9 της ROM (πρόγραμμα "rocket blast-off").
4. Αν με τη λήξη του προγράμματος θέλετε να το επανεκκινήσετε πατήστε ξανά το [RUN].
5. Τώρα, πατήστε [FETCH ADRS] [053E] [RUN]. Αυτό είναι ένα πρόγραμμα τυχαίας παραγωγής ήχων.
6. Πατήστε [RESET] για να σταματήσετε το πρόγραμμα.
7. Πατήστε [FETCH ADRS] [055A] [RUN]. Άλλο ένα από τα προγράμματα της ROM εμφανίζεται στα display.

8. Πατήστε [RESET] για να σταματήσει το πρόγραμμα.

Από τα παραδείγματα αξίζει να σχολιάσουμε δύο στοιχεία:

- Οι μικροεπεξεργαστές έχουν τη δυνατότητα να εκτελούν διαφορετικές λειτουργίες με το ίδιο υλικό χρησιμοποιώντας κάθε φορά διαφορετικό λογισμικό.
- Το πρόγραμμα με τον πύραυλο έχει στο τέλος του εντολή που επαναφέρει τον έλεγχο στο πρόγραμμα ελέγχου του μLab (monitor program). Τα άλλα δύο όμως προγράμματα τρέχουν συνεχώς έως ότου πατήσετε το [RESET].

Όμως, μεταξύ των έτοιμων προγραμμάτων του μLab δεν υπάρχουν μόνο παιχνίδια. Στη διεύθυνση 04E0 της ROM υπάρχει ένα πρόγραμμα που εξομοιώνει τη λειτουργία μιας πύλης AND με την χρήση των δύο I/O ports του μLab: της input port και της output port (LEDs). Μπορείτε να τις εντοπίσετε πάνω στην πλακέτα εύκολα από τα ονόματά τους που υπάρχουν δίπλα τους. Το πρόγραμμα διαβάζει την πόρτα εισόδου και αν όλα τα switches είναι στη θέση high τότε και η έξοδος είναι high, μια και αυτή είναι η λειτουργία της πύλης. Αυτή η διαδικασία επαναλαμβάνεται διαρκώς. Σε περίπτωση που έχουν αλλάξει τα δεδομένα εισόδου αλλάζει και η κατάσταση της πόρτας εξόδου. Ακολουθεί στην επόμενη σελίδα, στο σχήμα 1, το διάγραμμα ροής για το πρόγραμμα με την πύλη AND.

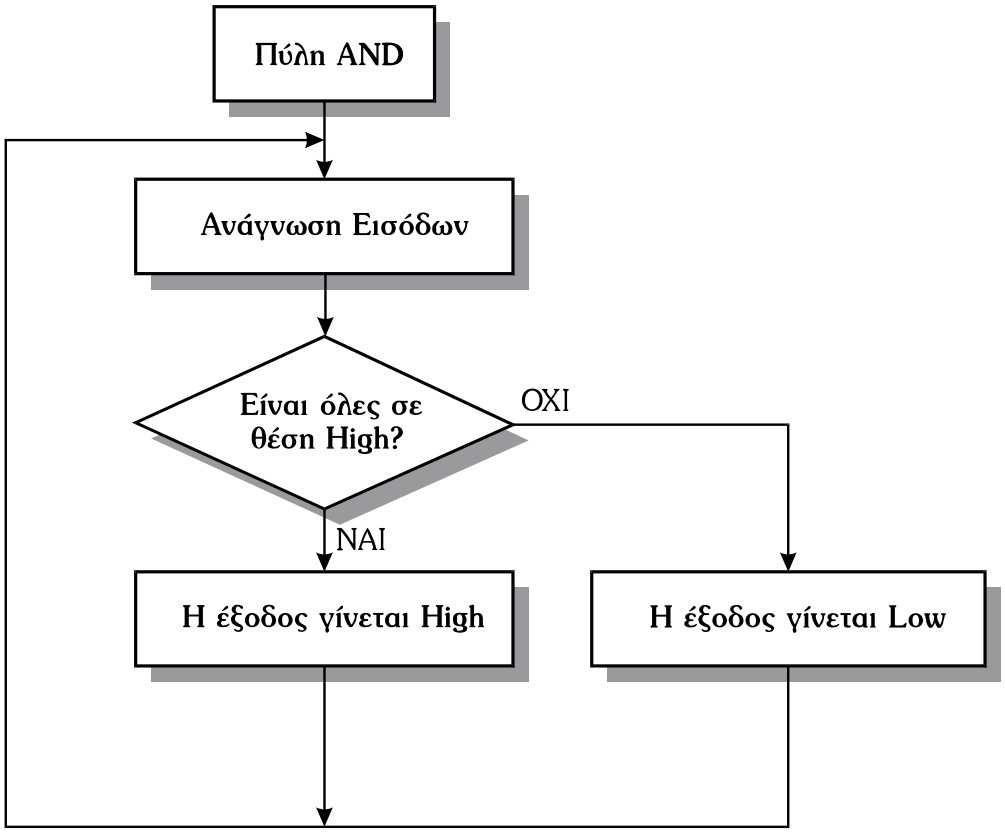
Για την εκτέλεση του προγράμματος ακολουθείστε τα παρακάτω βήματα:

1. Πατήστε το [RESET].
2. Πατήστε [FETCH ADRS] [04E0]. Αυτή είναι η διεύθυνση του προγράμματος στη ROM.
3. Πατήστε [RUN]. Το πρόγραμμα αρχίζει τώρα να τρέχει.
4. Θέσατε όλα τα input switches στην θέση high. Τώρα πρέπει το πιο δεξί από τα leds εξόδου να είναι στην κατάσταση OFF μια και το led αυτό προσομοιώνει το bit εξόδου της πύλης AND.
5. Αλλάξτε την κατάσταση οποιουδήποτε από τα input switches και το led 0 θα τεθεί σε κατάσταση ON αμέσως, όπως θα έπρεπε σε μια πύλη AND.
6. Πατήστε [RESET]. Αν τώρα θέσετε τα input switches στη θέση ON το led 0 δεν θα μεταβληθεί μια και το πρόγραμμα δεν τρέχει πλέον.

Δύο σημεία πρέπει να σχολιάσουμε:

- Τα leds εξόδου είναι συνδεδεμένα σε αρνητική λογική και όταν είναι σβηστά βρίσκονται σε κατάσταση high. Αυτή η ιδιαιτερότητα ίσως σας ξάφνιασε κατά την εκτέλεση του προγράμματος. Βέβαια είναι εύκολο σε κάποιο άλλο πρόγραμμα να περάσει απαρατήρητη.

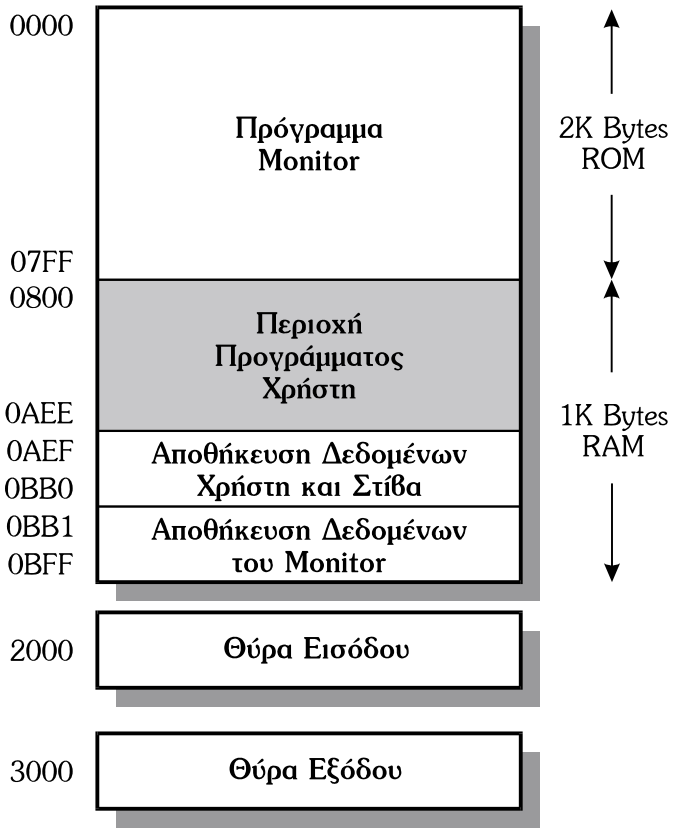
- Όταν προγραμματίζουμε το μLab και κατ' επέκταση τον 8085, ο κώδικας assembly που γράφουμε πρέπει να μετατραπεί σε κώδικα μηχανής για κάθε εντολή με βάση τον πίνακα που υπάρχει στο παράρτημα 2 και στη συνέχεια να εισαχθεί στο σύστημα (σε hex μορφή). Η μετατροπή από hex σε binary γίνεται αυτόματα.



Σχήμα 1. Διάγραμμα ροής για το πρόγραμμα εξομοίωσης πύλης AND

2. Χάρτης μνήμης του μLab

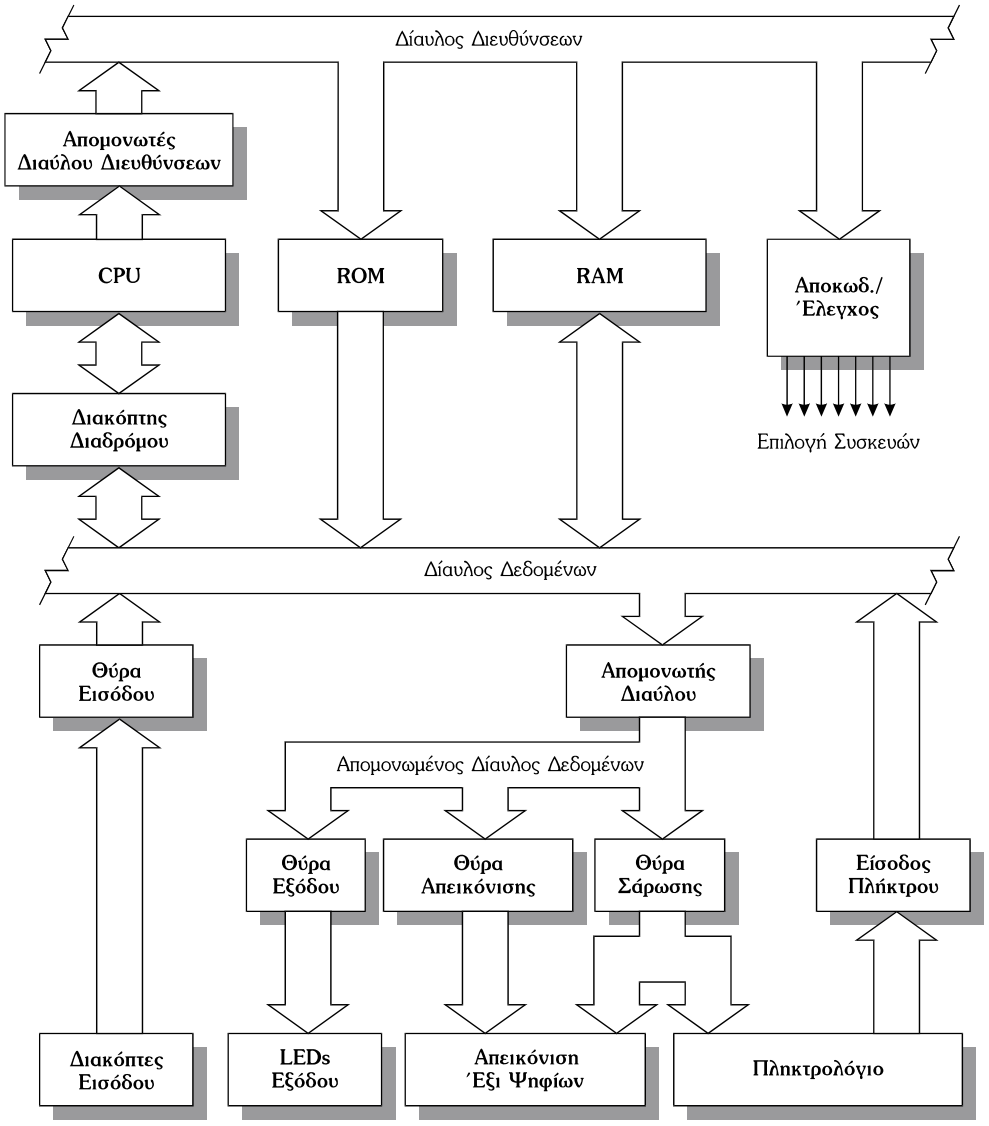
Για τη σωστή χρήση του εκπαιδευτικού συστήματος απαιτείται η γνώση του χάρτη μνήμης. Δηλαδή σε ποιες διευθύνσεις βρίσκονται οι μνήμες (ROM και RAM), οι πόρτες I/O καθώς και ο χώρος για την αποθήκευση του προγράμματος, των δεδομένων και του σωρού. Στο παρακάτω σχήμα δίνονται οι πληροφορίες αυτές.



Σχήμα 2. Χάρτης μνήμης του μLab

3. Block διάγραμμα του μLab

Στο επόμενο σχήμα δίνονται τα κύρια τμήματα του μLab και οι διάδρομοι επικοινωνίας μεταξύ τους. Λεπτομερής περιγραφή του συστήματος θα γίνει στην επόμενη παράγραφο.



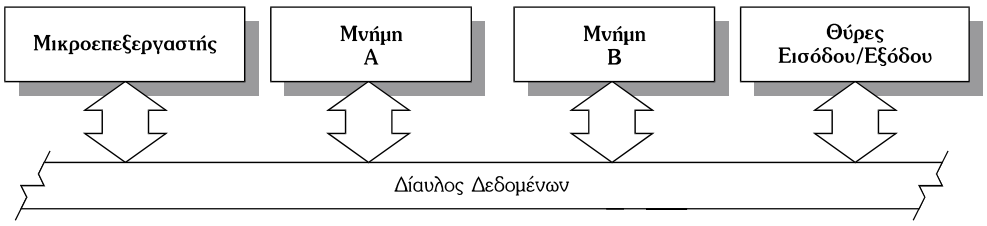
Σχήμα 3. Block διάγραμμα του μLab

4. Hardware του μLab

Το σύστημα του μLab είναι βασισμένο στη δομή των διαδρόμων (busses). Υπάρχει ο διάδρομος δεδομένων (data bus) και ο διάδρομος διεύθυνσης (address bus).

Ο 8085 δημιουργεί σήματα διευθύνσεων και ελέγχου. Είναι συνδεδεμένος με τον διπλής κατεύθυνσης διάδρομο δεδομένων (bidirectional data bus), μέσα από τον οποίο δέχεται ή στέλνει πληροφορίες. Η ROM, η RAM και οι πόρτες I/O χρησιμοποιούν το διάδρομο δεδομένων για να μεταφέρουν πληροφορίες από και προς τον 8085. Η επιλογή μιας από αυτές τις συσκευές γίνεται με τη βοήθεια των σημάτων διεύθυνσης (address-data). Τη δουλειά αναλαμβάνουν ειδικά κυκλώματα, τα κυκλώματα αποκωδικοποίησης διευθύνσεων (address decoding circuits), που αποκωδικοποιούν το περιεχόμενο του διαδρόμου διεύθυνσης και δημιουργούν ένα σήμα επιλογής για κάθε συσκευή του συστήματος. Το σήμα ενεργοποιεί το επιλεγέν memory ή I/O chip ώστε να μπορεί να πάρει ή να στείλει δεδομένα μέσω του data bus.

Η επικοινωνία του 8085 με τις μνήμες και τις πόρτες I/O γίνεται χρησιμοποιώντας το ίδιο data bus από και προς το μικροεπεξεργαστή, όπως φαίνεται στο παρακάτω σχήμα.



Σχήμα 4. Η διασύνδεση μέσω κοινού bus

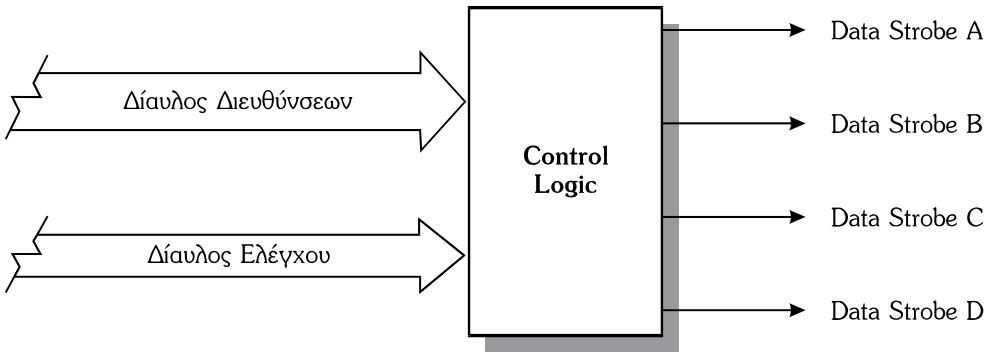
Η επικοινωνία αυτής της μορφής απαιτεί λιγότερες γραμμές data από αυτές που θα χρειάζονταν αν χρησιμοποιούσαμε ξεχωριστές συνδέσεις για κάθε πιθανό δρόμο από τον 8085 σε μνήμη ή πόρτα I/O. Με το κοινό bus όλες οι συσκευές είναι μονίμως συνδεδεμένες πάνω του, σε κάθε χρονική στιγμή όμως μόνο μία μπορεί να το χρησιμοποιήσει για να επικοινωνήσει με το μικροεπεξεργαστή.

Η επιλογή μιας από περισσότερες συσκευές που είναι συνδεδεμένες στο data bus γίνεται με τη βοήθεια των 3-state buffer (ή driver) που παρεμβάλλονται μεταξύ των συσκευών και του bus. Γενικά, ένας 3-state buffer ανάλογα με την τιμή

ενός σήματος επίτρεψης (enable) αποκαθιστά ή διακόπτει τη σύνδεση μεταξύ των εισόδων και των εξόδων του.

Τα σήματα enable δημιουργούνται από τη μονάδα ελέγχου (control unit), η οποία δέχεται εισόδους απευθείας από το μικροεπεξεργαστή. Η λειτουργία της είναι τέτοια ώστε όταν μία συσκευή είναι σε κατάσταση enable όλες οι άλλες να είναι σε κατάσταση disable, δηλαδή να παρουσιάζουν υψηλή αντίσταση.

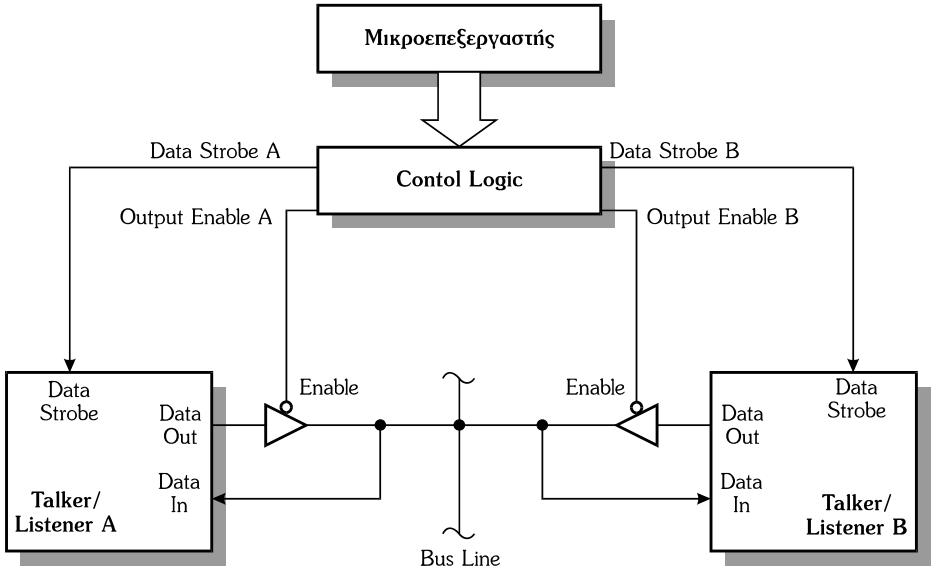
Εκτός από τα σήματα enable μια μονάδα ελέγχου μπορεί να δίνει στο σύστημα και πολλά άλλα σήματα, όπως σήματα strobe για να ειδοποιήσει τις ενεργοποιημένες συσκευές ότι το bus περιέχει έγκυρα δεδομένα. Ένα γενικό σχέδιο μονάδας ελέγχου φαίνεται στο επόμενο σχήμα.



Σχήμα 5. Συσκευή επιλογής control logic

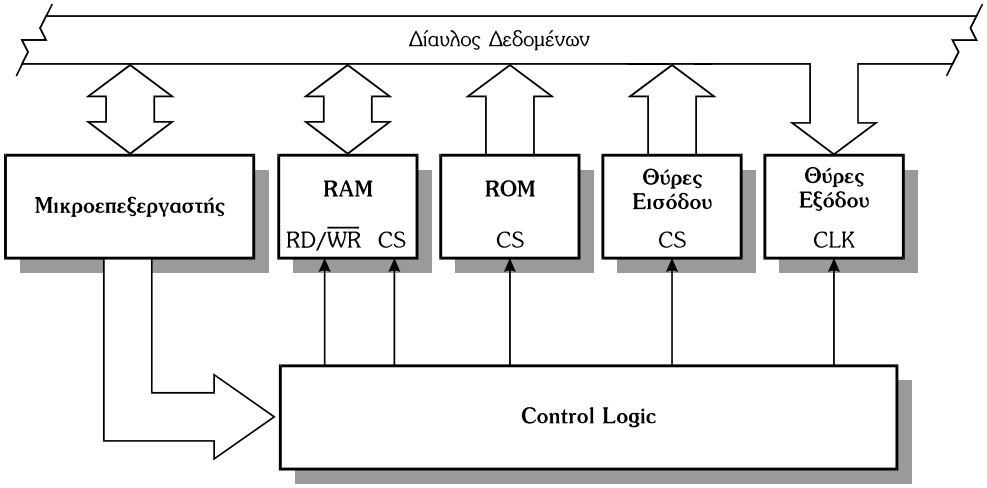
Ένα παράδειγμα διασύνδεσης διπλής κατεύθυνσης με κοινό bus δύο συσκευών A και B, φαίνεται στο σχήμα 6.

Για τη μεταφορά δεδομένων από τη συσκευή A στη B, η μονάδα ελέγχου θέτει την έξοδο enable A, true και την έξοδο enable B false. Μετά από ένα χρόνο settling, ώστε να φτάσουν τα δεδομένα στην είσοδο δεδομένων της B, στέλνει ένα παλμό στη γραμμή data strobe B ώστε να διαβάσει τα δεδομένα. Σε όλη αυτή τη διαδικασία καμία άλλη συσκευή δε συνδέεται με τον διάδρομο δεδομένων. Στο συγκεκριμένο παράδειγμα ο μικροεπεξεργαστής δε συνδέεται με το bus αλλά απλά ελέγχει τις συσκευές.



Σχήμα 6. Σύνδεση διπλής κατεύθυνσης σε κοινό bus

Στα μικροϋπολογιστικά συστήματα η είσοδος που δέχεται το σήμα enable είναι η CS (chip select) που υπάρχει στις RAM, ROM και πόρτες input. Ο μικροεπεξεργαστής ενεργεί σαν ελεγκτής και δεν επιτρέπει παρά μόνο σε μία συσκευή (εκτός από αυτόν) να χρησιμοποιήσει το data bus. Μια γενική συνδεσμολογία φαίνεται στο παρακάτω σχήμα.



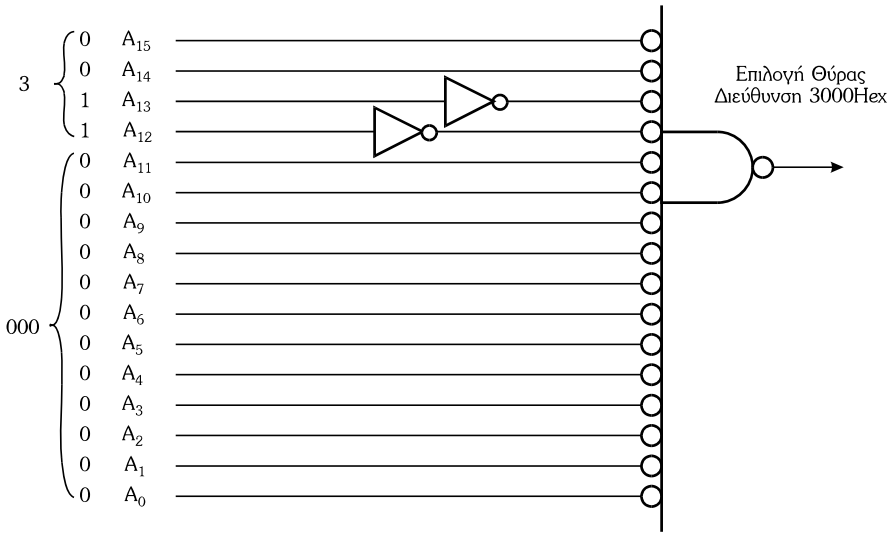
Σχήμα 7. Επικοινωνία συσκευών με τον μΕ μέσω του data bus

Αυτά είναι τα γενικά χαρακτηριστικά της συνδεσμολογίας κοινού bus του μLab. Ο 8085 τα χρησιμοποιεί ως εξής. Για να διαβάσει δεδομένα δημιουργεί (μέσω της control logic) σήμα ελέγχου για την ενεργοποίηση της αντίστοιχης συσκευής. Αμέσως οι έξοδοι της συσκευής παρέχουν δεδομένα στο data bus τα οποία και παραλαμβάνονται από τον μικροεπεξεργαστή.

Για να γράψει δεδομένα στη RAM ή να στείλει στην πόρτα εξόδου, ο 8085 τα φέρνει πρώτα πάνω στο data bus. Στέλνει στη συνέχεια ένα παλμό WRITE στην επιλεγείσα συσκευή, ώστε να τα παραλάβει. Όταν δεν ενεργοποιείται καμία συσκευή το data bus βρίσκεται σε κατάσταση high και τα data bus LEDs ανάβουν. Υπάρχουν 8 γραμμές στο διάδρομο δεδομένων γιατί ο μικροεπεξεργαστής 8085 είναι 8 bits.

4.1 Διάδρομος διευθύνσεων

Η επιλογή της συσκευής που θα επικοινωνήσει με το διάδρομο δεδομένων γίνεται με την βοήθεια του διαδρόμου διευθύνσεων. Ο 8085 έχει διάδρομο διευθύνσεων 16 γραμμών, αντιπροσωπεύοντας 2^{16}=65536 θέσεις μνήμης και πόρτες I/O. Τα pins του διαδρόμου διευθύνσεων συμβολίζονται A_0, A_1, A_2, ..., A_{15} με το A_0 σαν το λιγότερο σημαντικό ψηφίο.



Σχήμα 8. Αποκωδικοποίηση διεύθυνσης για τον έλεγχο της πόρτας 3000H

Ο αποκωδικοποιητής διευθύνσεως (address decoder) είναι μέρος του control logic. Δημιουργεί σήματα επιλογής συσκευής όταν η διεύθυνση του address bus βρίσκεται μέσα στην περιοχή διευθύνσεων που αντιστοιχεί σε κάθε

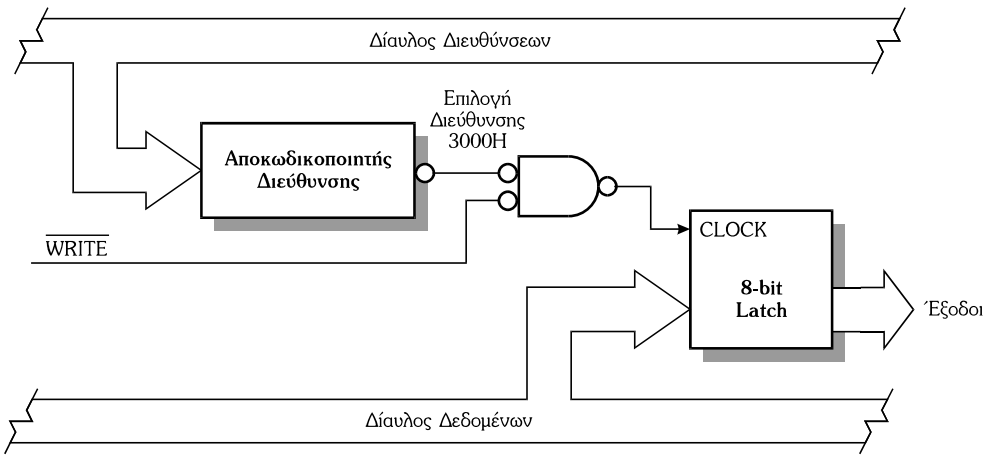
συσκευή. Σύμφωνα με τα παραπάνω θα δούμε πώς γίνεται η επιλογή, με τη βοήθεια του address decoder για τη διεύθυνση 3000H (0011 0000 0000 0000 binary). Η έξοδος του αποκωδικοποιητή είναι αληθής (λογικό 0) μόνο όταν αυτή ακριβώς η διεύθυνση εμφανίζεται στο διάδρομο διευθύνσεων. Αυτή η έξοδος χρησιμοποιείται για να ενεργοποιήσει μία πόρτα I/O, όπως φαίνεται στο σχήμα 8.

4.2 Διάδρομος ελέγχου

Ο 8085 παρέχει δύο κύρια σήματα ελέγχου, τα READ και WRITE. Εάν το READ είναι low αυτό δείχνει ότι έχουμε διαδικασία ανάγνωσης, και τα σήματα του μικροεπεξεργαστή δείχνουν στην υποδειχθείσα συσκευή να δώσει δεδομένα στο διάδρομο δεδομένων. Εάν το WRITE είναι low, τότε είμαστε στη διαδικασία εγγραφής, οπότε ο μικροεπεξεργαστής παρέχει δεδομένα στο διάδρομο δεδομένων και ειδοποιεί την υποδειχθείσα συσκευή να τα αποθηκεύσει. Τα READ και WRITE LEDs κατάστασης του μLab συνδέονται με αυτά τα σήματα. Τα LEDs ανάβουν όταν το αντίστοιχο σήμα είναι αληθές (low).

4.3 Πόρτες εισόδου/εξόδου

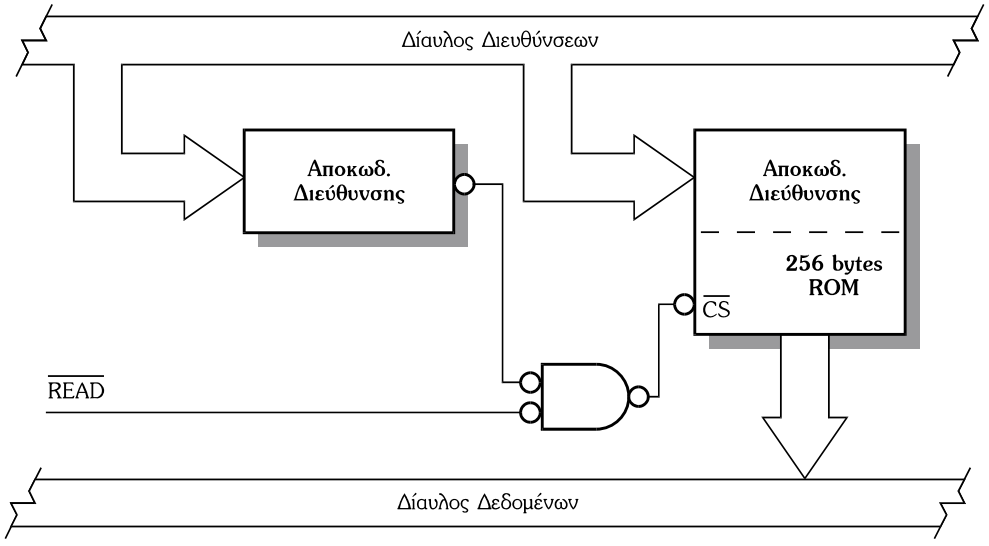
Το επόμενο σχήμα δείχνει ένα μανδαλωτή (latch) εξόδου μαζί με τον τρόπο επιλογής της διεύθυνσης 3000H. Ο μανδαλωτής δέχεται παλμό ρολογιού όταν η διεύθυνση 3000H παρουσιάζεται στο διάδρομο διευθύνσεων και έχουμε αλλαγή από low σε high στο σήμα ελέγχου WRITE. Τότε τα δεδομένα από το data bus αποθηκεύονται στο μανδαλωτή και τροφοδοτούν τις εξόδους.



Σχήμα 9. Αποθήκευση στον μανδαλωτή που βρίσκεται στη διεύθυνση 3000H

Ανάλογα λειτουργούν και οι πόρτες εισόδου μόνο που την έξοδο του αποκωδικοποιητή διευθύνσεων οδηγούμε σε μία NAND μαζί με το σήμα READ. Η έξοδος της NAND ενεργοποιεί την είσοδο enable του 8 bit 3-state driver ώστε τα δεδομένα εισόδου να μπουν στον διάδρομο δεδομένων και να αποθηκευτούν προσωρινά στους καταχωρητές του 8085.

Το σήμα READ χρησιμοποιείται και για την ανάγνωση από την ROM. Στο σχήμα 10 έχουμε ένα παράδειγμα μιας μικρής ROM 256 θέσεων όπου τα περισσότερα σημαντικά bits (A_8-A_{15}) της διεύθυνσης αποκωδικοποιούνται από εξωτερικό αποκωδικοποιητή, ώστε μαζί με το σήμα READ να επιτρέψουν στην ROM να δώσει δεδομένα. Τα bits A_0-A_7 δείχνουν ποια από τις 256 θέσεις του chip της ROM έχει επιλεγεί και αποκωδικοποιούνται μέσα στο chip.



Σχήμα 10. Εσωτερική αποκωδικοποίηση της ROM

Για τη λειτουργία READ της RAM ισχύουν τα ίδια με τη ROM.

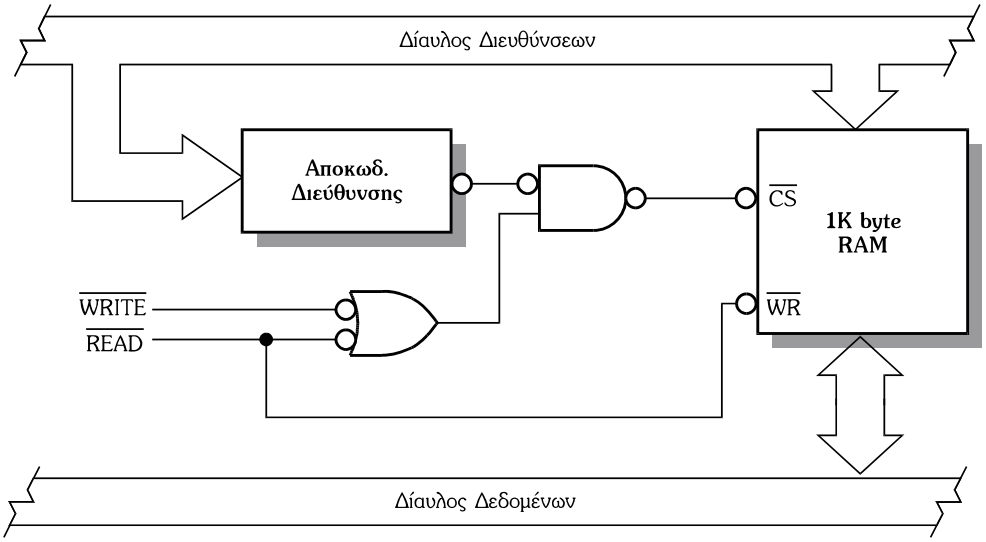
Στη RAM, όμως, έχουμε και την λειτουργία WRITE (αποθήκευση δεδομένων). Για αυτό η μνήμη αυτή έχει στην είσοδο CS (chip select) και το σήμα WRITE περασμένο από μια NAND.

Το σχήμα 11 δείχνει τον πίνακα αλήθειας ελέγχου της RAM.

Το σχήμα 12 δείχνει τον αποκωδικοποιητή διευθύνσεων και τον έλεγχο μιας RAM 1KB.

$\overline{CS}$	$\overline{WR}$	**Λειτουργία**
0	0	Εγγραφή
0	1	Ανάγνωση
1	X	Καμία Λειτουργία

Σχήμα 11. Πίνακας αλήθειας για τον έλεγχο της RAM



Σχήμα 12. Κύκλωμα ελέγχου 1K byte RAM

Για το μLab υπάρχει ο παρακάτω χάρτης διευθύνσεων πάνω στον οποίο φαίνεται ο χώρος που καταλαμβάνει κάθε κατασκευή.

Πίνακας 1: Χάρτης διευθύνσεων του μLab

Bit:								Διεύθυνση	
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**HEX**	**Συσκευή**
0	0	0	0	0	0	0	0	0000	ROM
0	0	0	0	0	1	1	1	07FF	
0	0	0	0	1	0	0	0	0800	RAM
0	0	0	0	1	1	1	1	0FFF	
0	0	0	1	0	0	0	0	1000	Έλεγχος
0	0	0	1	0	1	1	1	17FF	
0	0	0	1	1	0	0	0	1800	Δεδομένα
0	0	0	1	1	1	1	1	1FFF	Πληκτρολογίου
0	0	1	0	0	0	0	0	2000	Θύρα Εισόδου
0	0	1	0	0	1	1	1	27FF	
0	0	1	0	1	0	0	0	2800	Σάρωση
0	0	1	0	1	1	1	1	2FFF	(Keyboard+Display)
0	0	1	1	0	0	0	0	3000	Θύρα Εξόδου
0	0	1	1	0	1	1	1	37FF	
0	0	1	1	1	0	0	0	3800	Τμήμα
0	0	1	1	1	1	1	1	3FFF	Απεικόνισης
0	1	0	0	0	0	0	0	4000	Αχρησιμοποίητα
1	1	1	1	1	1	1	1	FFFF	

Η ROM καταλαμβάνει 2KB, η RAM 2KB (αν η RAM είναι 1KB, όπως στο μLab του εργαστηρίου, τότε μένει ελεύθερο 1KB). Τα επόμενα 6 τμήματα των 2KB, αντιπροσωπεύουν τα I/O ports.

Η πόρτα ελέγχου (control) χρησιμοποιείται από το monitor πρόγραμμα προσφέροντας μερικές ειδικές λειτουργίες.

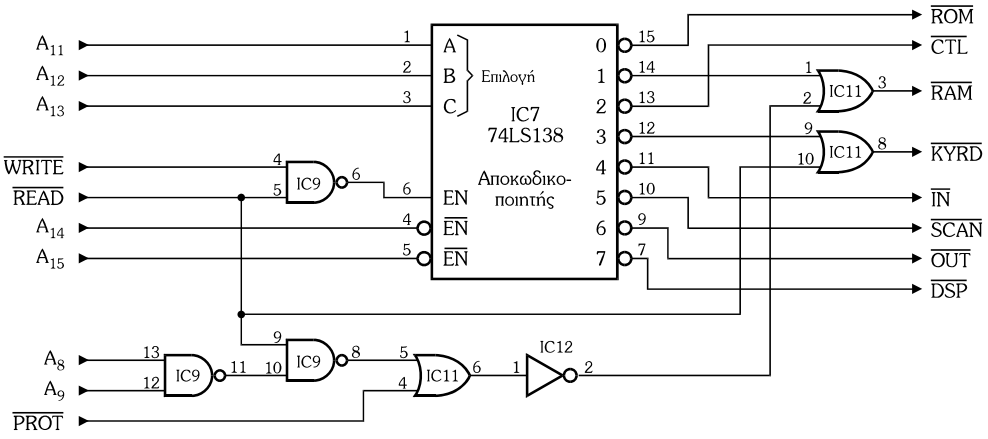
Οι πόρτες key data, scan και display segments ελέγχουν το πληκτρολόγιο και το display. Οι πόρτες εισόδου, εξόδου χρησιμοποιούνται για τους dip-switches

και για τα LEDs. Όλες οι πόρτες δέχονται ένα byte δεδομένων διεύθυνσης και ενεργοποιούν τη συσκευή που αντιστοιχεί σε αυτήν την πόρτα, αν το byte διεύθυνσης περιέχεται στα 2K byte αυτής της πόρτας. Αυτό σημαίνει ότι σε κάθε πόρτα αντιστοιχούν 2047 (2KB-1) πλεονάζουσες διευθύνσεις.

Βέβαια όπως βλέπουμε μένουν αχρησιμοποίητα 48KB διευθύνσεις για επέκταση του συστήματος, αφού χρησιμοποιούνται 2KB για ROM, 2KB για RAM και (6×2KB) για I/O πόρτες, δηλαδή συνολικά 16KB.

4.4 Hardware αποκωδικοποίησης

Το παρακάτω κύκλωμα είναι ο αποκωδικοποιητής διεύθυνσης του μLab.



Σχήμα 13. Αποκωδικοποίηση διευθύνσεων στο μLab

Οι γραμμές A_{11}, A_{12}, A_{13} όπως φαίνεται στο χάρτη διευθύνσεων δείχνουν ποιος από τους 8 τομείς των 2K έχει επιλεγεί. Αυτές οι τρεις γραμμές είναι οι είσοδοι δυαδικής λογικής του IC 74LS138, ενός δυαδικού αποκωδικοποιητή τρία σε οκτώ. Αυτό το κύκλωμα δίνει 8 εξόδους μία για κάθε τομέα. Επιπλέον έχει 3 εισόδους ενεργοποίησης: δύο ενεργές low και μία ενεργή high. Και οι τρεις πρέπει να είναι αληθείς για να φέρουν σε κατάσταση αληθή μία έξοδο. Οι γραμμές A_{14} και A_{15} (συνδεσμολογημένες στις δύο ενεργές low εισόδους ενεργοποίησης), απαγορεύουν να γίνει αληθής κάποια έξοδος εκτός αν είναι και οι δύο low. Αυτός είναι ο έλεγχος των κατώτερων 16 από τα 64KB του πεδίου διευθύνσεων.

Με τη βοήθεια των γραμμών READ, WRITE γίνεται επιλογή λειτουργίας εγγραφής ή διαβάσματος δεδομένων μέσω της πύλης NAND στην τρίτη είσοδο enable. Έτσι δεν αναγκαζόμαστε να χρησιμοποιούμε τις γραμμές αυτές κατευθείαν στα περισσότερα από τα σήματα επιλογής συσκευής. Για να λειτουργήσει κάποια

συσκευή πρέπει να έχουμε κατάσταση READ ή WRITE. Οι ενεργοποιήσεις αυτές προστατεύουν τις συσκευές για να μην πάρουν δεδομένα σε λάθος χρόνο.

Επιπλέον, οι πόρτες εισόδου και η ROM πρέπει να ενεργοποιούνται μόνο σε κατάσταση READ. Αν π.χ. η ROM ενεργοποιηθεί σε κατάσταση WRITE τότε, εκτός από τα δεδομένα στο διάδρομο δεδομένων που είναι υπό εγγραφή, θα θελήσει να δώσει δεδομένα και η ROM, γεγονός που θα μπορούσε να προκαλέσει σοβαρή βλάβη στο σύστημα.

Για να λυθεί το πρόβλημα αυτό, πρέπει το σήμα READ να περάσει από μία πύλη OR μαζί με το σήμα επιλογής συσκευής. Συνήθως οι ROM έχουν δύο εισόδους επίτρεψης, οπότε η σύνδεση του READ σε μια από αυτές έχει το ίδιο αποτέλεσμα. Στο σχήμα 13 η παραπάνω τεχνική επιδεικνύεται στο σήμα KYRD.

Για τις πόρτες εξόδου όμως δεν χρειάζεται OR, γιατί μή λειτουργία READ σε αυτές δε θα προκαλέσει βλάβη στο κύκλωμα, απλά θα έχει σαν αποτέλεσμα τη μεταφορά μη έγκυρων δεδομένων στη συσκευή εξόδου.

Ιδιαίτερη φροντίδα υπάρχει για τις RAM. Η πύλη OR (IC 11) αποτελεί την λεγόμενη προστασία εγγραφής. Αυτή υπάρχει για να προστατεύει το περιεχόμενο των RAM από προγράμματα που κατά την λειτουργία τους προσπαθούν να γράψουν δεδομένα στο τμήμα της μνήμης που είναι αποθηκευμένος ο κώδικάς τους.

Για αυτό υπάρχει ο μανδαλωτής προστασία μνήμης. Η έξοδός του δίνει το σήμα PROT στην είσοδο 4 του IC 11. Όταν ο μανδαλωτής είναι set η RAM προστατεύεται, δηλαδή μπορούμε να διαβάσουμε αλλά δεν μπορούμε να γράψουμε δεδομένα. Το monitor πρόγραμμα κάνει set τον μανδαλωτή προστασίας όταν τρέχουμε ένα πρόγραμμα. Το αντίθετο συμβαίνει (reset) κατά την εισαγωγή δεδομένων από το πληκτρολόγιο.

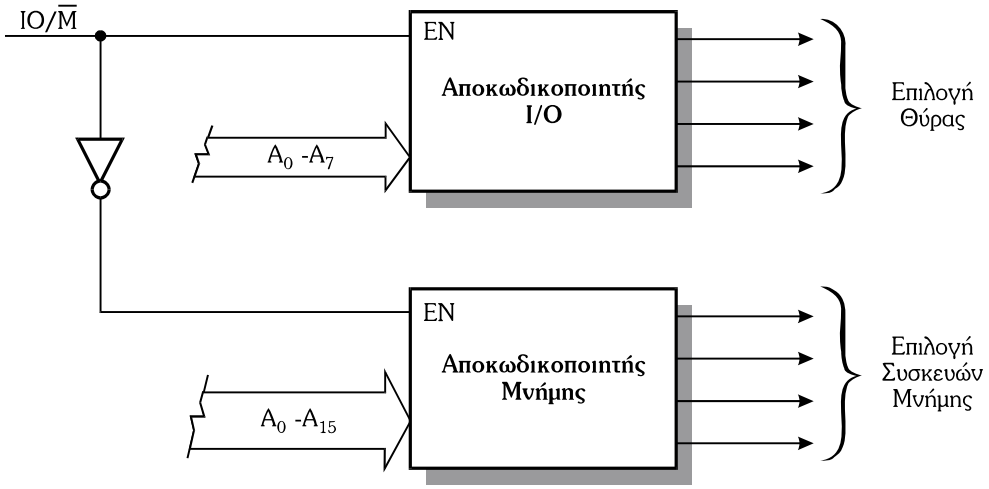
Επειδή όμως θέλουμε να αποθηκεύουμε δεδομένα κατά την εκτέλεση προγραμμάτων, μόνο τα πρώτα τρία τέταρτα της RAM προστατεύονται (για κάθε 1KB RAM). Οι γραμμές διευθύνσεων A_8 και A_9 δείχνουν σε ποιο τέταρτο της RAM αναφερόμαστε. Όταν είναι και τα δύο high ελευθερώνεται το τελευταίο τέταρτο της RAM από την προστασία. Τα IC 9, 11 και 12 δίνουν έξοδο το σήμα ενεργοποίησης της RAM.

Σύμφωνα λοιπόν με τα παραπάνω, κατά την εκτέλεση ενός προγράμματος, στο μLab του εργαστηρίου με 1KB RAM, μπορεί να γίνει εγγραφή MONO στις διευθύνσεις 0B00H-0BFFH, εκτός εάν τεθεί ο μανδαλωτής PROT 0, με τρόπο που θα παρουσιαστεί στη συνέχεια.

4.5 Αποκωδικοποίηση I/O

Μερικοί μικροεπεξεργαστές όπως ο 8080 και 8085 έχουν μία επιπλέον γραμμή ελέγχου για την επιλογή λειτουργιών σε συσκευές I/O ή μνήμης. Στον 8085 αυτή η γραμμή λέγεται IO/M (pin 34). Στην κατάσταση high εκτελούνται I/O λειτουργίες ενώ στην κατάσταση low λειτουργούν οι μνήμες. Χρησιμοποιώντας αυτή τη μέθοδο η μνήμη και οι πόρτες I/O έχουν ξεχωριστά διαστήματα διευθύνσεων, αυξάνοντας το συνολικό διάστημα διευθύνσεων του συστήματος κατά 2^8=256 bytes. Έτσι δίνεται μεγαλύτερη ευελιξία στη σχεδίαση του αποκωδικοποιητή διευθύνσεων.

Το μLab δε χρησιμοποιεί αυτή την τεχνική, γι' αυτό και η γραμμή IO/M δε χρησιμοποιείται. Όπως αναφέραμε προηγούμενα, για τις λειτουργίες I/O χρησιμοποιείται τεχνική memory map.

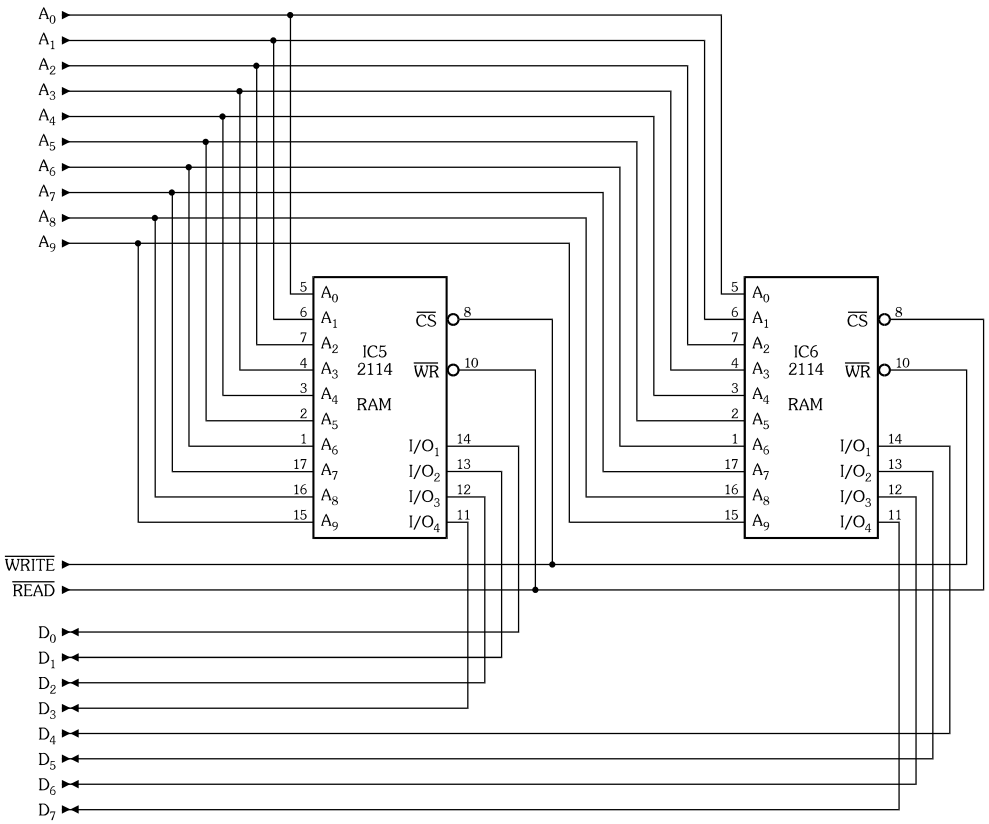


Σχήμα 14. Διαχωρισμός του standard I/O από το memory map I/O

4.6 Μνήμη RAM

Το μlab χρησιμοποιεί δύο στατικές μνήμες RAM 2114. Οι στατικές αυτές RAM έχουν ένα flip-flop για κάθε στοιχείο μνήμης και όταν το flip-flop βρίσκεται στην κατάσταση set αυτή ισοδυναμεί με το λογικό "1", ενώ στην κατάσταση reset ισοδυναμεί με το λογικό "0".

Όπως φαίνεται στο σχήμα που ακολουθεί τα δύο chips 2144 συμπεριφέρονται σαν μνήμη 1Kbyte. Οι γραμμές διευθύνσεων και ελέγχου είναι παράλληλα συνδεδεμένες, ενώ οι γραμμές δεδομένων D_0-D_3 εξυπηρετούνται από το IC5 και οι γραμμές D_4-D_7 από το IC6. Αυτό γίνεται γιατί η RAM 2114 είναι μεγέθους 4 bit, οπότε για 8 bit πληροφορία χρειαζόμαστε δύο IC 2114.

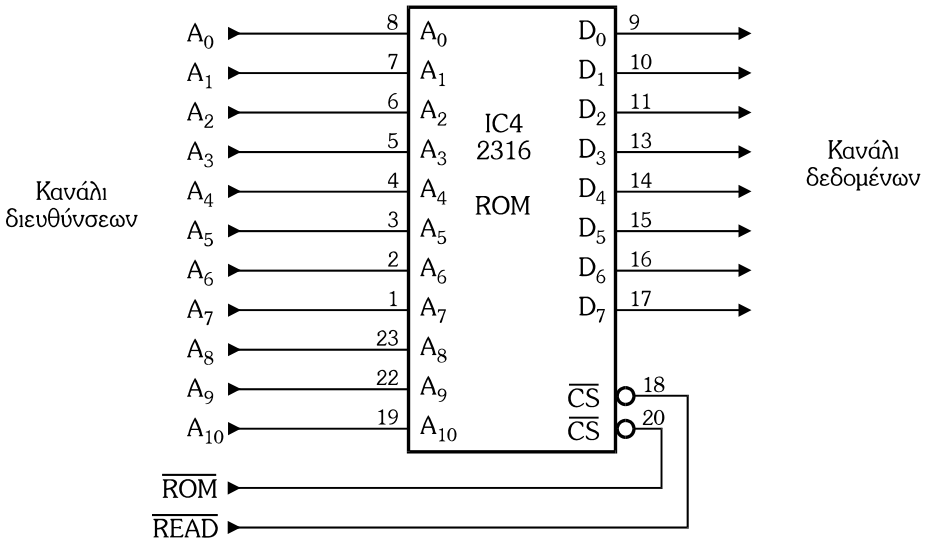


Σχήμα 15. Το κύκλωμα της RAM στο μLab

4.7 Μνήμη ROM

Η ROM του συστήματος πάνω στην οποία είναι γραμμένο το monitor πρόγραμμα του μLab είναι η 2316E ROM μεγέθους 2Kbytes και είναι τεχνικής προγραμματιζόμενης μάσκας. Η ROM παρέχει δεδομένα εφόσον και οι δύο είσοδοι CS (Chip Select), ROM και READ είναι true. Επίσης από τις 16 γραμμές διευθύνσεων που είναι διαθέσιμες, χρησιμοποιούνται οι 11 κατώτερες γραμμές σύμφωνα με αυτά που αναφέραμε προηγουμένως.

Στο σχήμα που ακολουθεί εικονίζεται το ολοκληρωμένο 2316 με τις εισόδους και τις εξόδους του.



Σχήμα 16. Η ROM του μLab

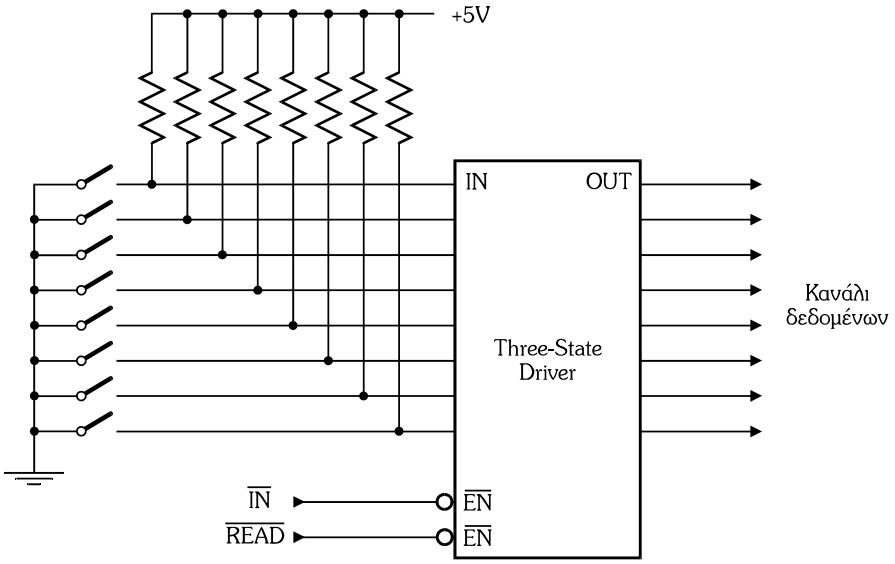
4.8 Περιφερειακές μονάδες

Η επικοινωνία του μικροεπεξεργαστή με τον έξω κόσμο γίνεται τουλάχιστον με μία συσκευή εισόδου και μία συσκευή εξόδου που συνδέονται στις πόρτες Ε/Ε. Οι συσκευές αυτές αποτελούν τα περιφερειακά, καθόσον δεν είναι απευθείας συνδεδεμένες με τον μικροεπεξεργαστή.

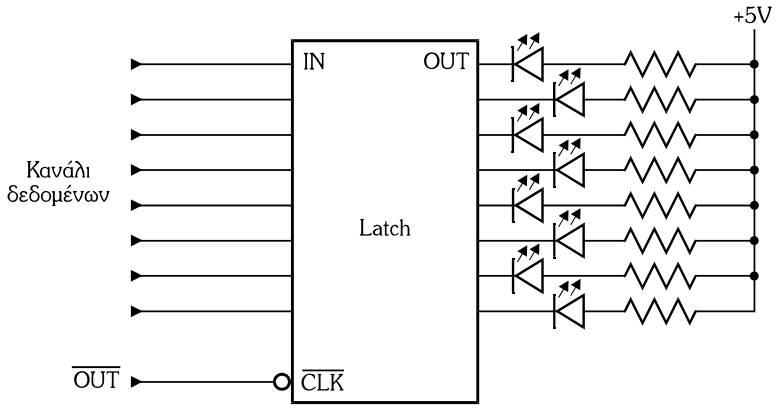
4.9 Είσοδοι/Εξοδοι

Το μlab έχει δύο απλά περιφερειακά: τα dip-switches της πόρτας εισόδου και τα LEDs της πόρτας εξόδου. Αν κάποιος από τους διακόπτες είναι κλειστός θέτει την είσοδο σε κατάσταση low, ενώ αν είναι ανοικτός σε κατάσταση high.

Κάθε έξοδος από την πόρτα εξόδου οδηγεί ένα LED, ενώ οι αντιστάσεις περιορίζουν το ρεύμα που τα διαρρέει. Όταν η έξοδος είναι low τα LEDs άγουν και φωτοβολούν. Έτσι, από τη μια μεριά μπορούμε να επηρεάσουμε την είσοδο και από την άλλη να "δούμε" την έξοδο. Στα σχήματα που ακολουθούν φαίνονται, αναλυτικότερα, οι πόρτες Ε/Ε του μlab.



Πόρτα εισόδου με μικροδιακόπτες

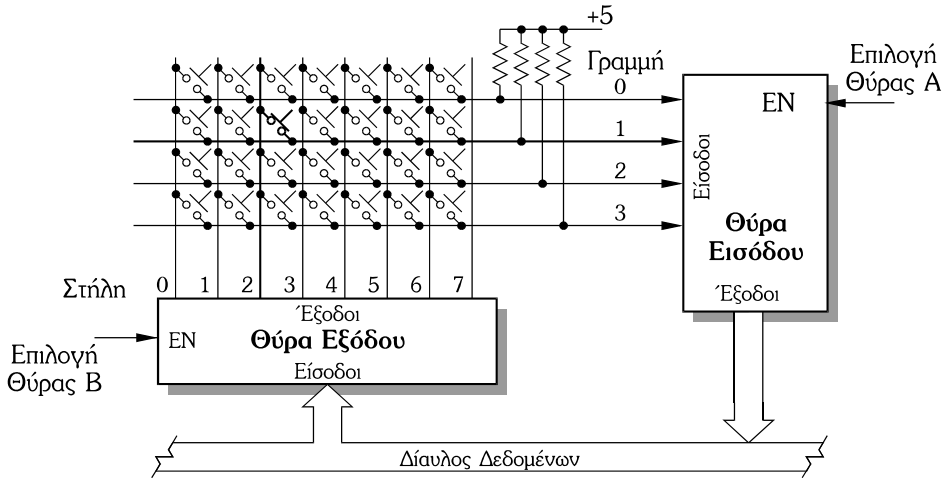


Πόρτα εξόδου με LEDs

Σχήμα 17. Θύρες Ε/Ε του μLab

4.10 Πληκτρολόγιο

Βασίζεται στην τεχνική σάρωσης (scanning) που χρησιμοποιείται και για τη φωτεινή οθόνη (display). Το πληκτρολόγιο του μlab έχει 26 πλήκτρα. Στο σχήμα 18 της επόμενης σελίδας φαίνεται ένα interface πληκτρολογίου. Παρατηρούμε ότι για τα πλήκτρα σχηματίζεται ένας πίνακας (matrix) συνδέσεων από γραμμές και στήλες. Μία πόρτα εξόδου οδηγεί τις στήλες και μία πόρτα εισόδου διαβάζει τις οριζόντιες γραμμές. Στο σχήμα 19 δίνεται ένα παράδειγμα για το πώς ανιχνεύεται ένα πλήκτρο με τη μέθοδο αυτή.



Σχήμα 18. Κύκλωμα σύνδεσης πληκτρολογίου στο μLab

Κατάσταση	Θύρα εξόδου Στήλη	Θύρα εισόδου Γραμμή
	0 1 2 3 4 5 6 7	0 1 2 3
0	0 1 1 1 1 1 1 1	1 1 1 1
1	1 0 1 1 1 1 1 1	1 1 1 1
2	1 1 **0** 1 1 1 1 1	1 **0** 1 1
3	1 1 1 0 1 1 1 1	1 1 1 1
4	1 1 1 1 0 1 1 1	1 1 1 1
5	1 1 1 1 1 0 1 1	1 1 1 1
6	1 1 1 1 1 1 0 1	1 1 1 1
7	1 1 1 1 1 1 1 0	1 1 1 1

Το κουμπί που πατήθηκε, βρίσκεται στη Γραμμή 1, Στήλη 2

Σχήμα 19. Ο τρόπος σάρωσης του πληκτρολογίου

Για να γίνει σάρωση στο πληκτρολόγιο χρειάζεται ένα πρόγραμμα το οποίο να θέτει την πόρτα εξόδου σε κατάσταση τέτοια ώστε μόνο ένα bit είναι low και όλα τα άλλα high. Το bit αυτό επιλέγει μία στήλη στον πίνακα των πλήκτρων. Στη συνέχεια διαβάζεται η πόρτα εισόδου. Αν κάποιο από τα πλήκτρα στη στήλη που έχει επιλεγεί είναι πατημένο, το bit της πόρτας εισόδου που αντιστοιχεί στη γραμμή του πλήκτρου αυτού γίνεται low (τα υπόλοιπα παραμένουν high). Έτσι, το πρόγραμμα γνωρίζει ποια στήλη και ποια γραμμή έχουν ενεργοποιηθεί. Αυτό αποτελεί και τη διαδικασία για την αναγνώριση του πλήκτρου που πατήθηκε. Για να γίνει έλεγχος σε όλα τα πλήκτρα κάθε ένα bit της πόρτας εξόδου γίνεται low και στη συνέχεια γίνεται scanning στα πλήκτρα, ελέγχοντας σε κάθε βήμα μόνο τέσσερα πλήκτρα.

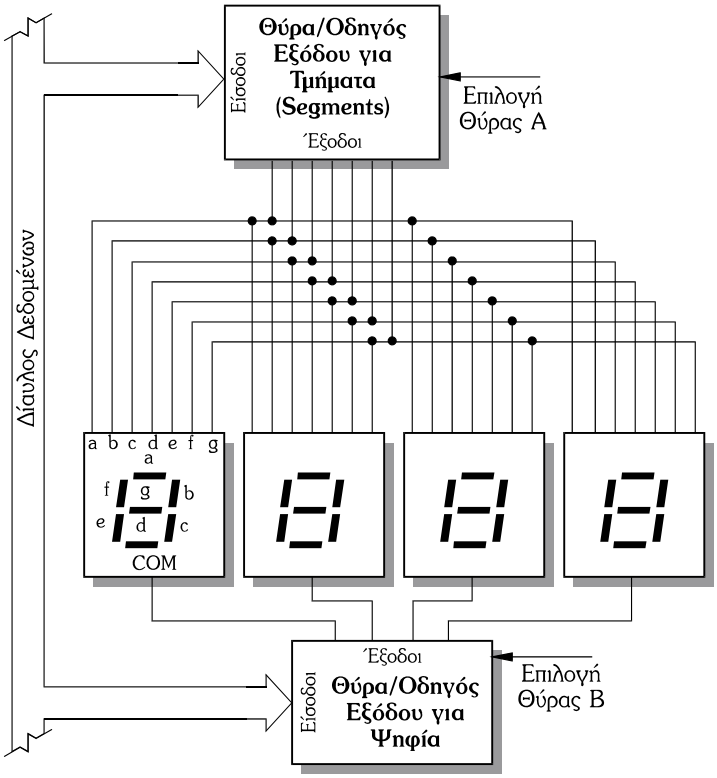
4.11 Η οθόνη (display)

Χρησιμοποιείται και εδώ, όπως και στο πληκτρολόγιο, η τεχνική σάρωσης. Το μLab έχει display 6 ψηφίων και μόνο ένα ανάβει κάθε φορά. Η διαδικασία όμως αυτή, γίνεται τόσο γρήγορα, ώστε να φαίνεται ότι όλα ανάβουν ταυτόχρονα. Κάθε ψηφίο σχηματίζεται με τη βοήθεια 7 LEDs για τον εικονιζόμενο χαρακτήρα και ένα για υποδιαστολή.

Σε κάθε ψηφίο υπάρχει μία σύνδεση για κάθε LED και μια κοινή για όλα (COM), όπως φαίνεται στο σχήμα 20. Ένας χαρακτήρας εμφανίζεται θέτοντας χαμηλή στάθμη στη κοινή σύνδεση και υψηλή στον ακροδέκτη κάθε LED που θέλουμε να ανάψει. Το ρεύμα περιορίζεται με αντιστάσεις σε σειρά με κάθε φωτεινό τμήμα (LED).

Μία πόρτα εξόδου των 8 bits τροφοδοτεί όλα τα ψηφία με τις πληροφορίες για κάθε φωτεινό τμήμα, ενώ μία άλλη πόρτα εξόδου οδηγεί τις κοινές συνδέσεις κάθε ψηφίου και καθορίζει ποιο θα ανάψει.

Για τον έλεγχο του display χρειάζεται ένα πρόγραμμα το οποίο πρώτα στέλνει την πληροφορία τμήματος για το ψηφίο 0 στην πόρτα των τμημάτων (Output Port/Driver for Segments) όπου μανδαλώνεται. Τότε η πόρτα του ψηφίου (Output Port/Driver for Digits) χρησιμοποιείται για να ενεργοποιήσει μόνο το ψηφίο 0 για ένα δεδομένο χρόνο (ελεγχόμενο από τον κύκλο χρονισμού του προγράμματος) και έπειτα για να το σβήσει. Η διαδικασία επαναλαμβάνεται για κάθε ένα από τα επόμενα ψηφία.



Σχήμα 20. Διάγραμμα σύνδεσης οθόνης 7-segment στο μLab

Τέλος, πρέπει να αναφέρουμε ότι στο μLab η πόρτα σάρωσης (βλέπε χάρτη μνήμης) χρησιμοποιείται τόσο σαν πόρτα ψηφίου για τον έλεγχο του display όσο και σαν πόρτα στηλών για τον έλεγχο του πληκτρολογίου. Αυτή η τεχνική χρησιμοποιείται συχνά στο σχεδιασμό μικροϋπολογιστικών συστημάτων για την μείωση του hardware.

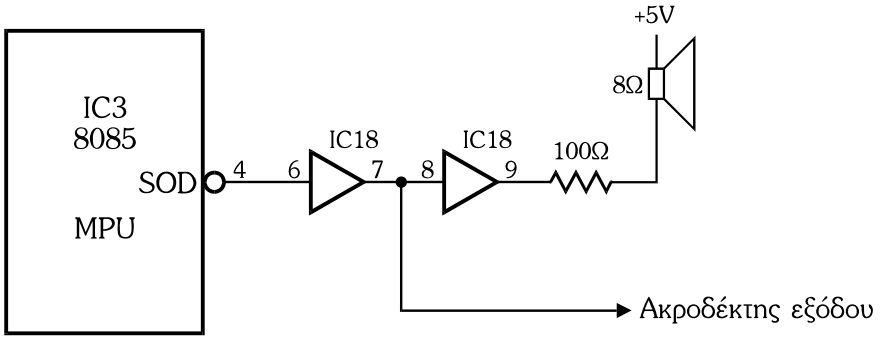
4.12 Η σειριακή θύρα εξόδου

Η σειριακή πόρτα εξόδου οδηγεί το μεγάφωνο του μlab και είναι ουσιαστικά μια πόρτα εξόδου 1 bit, ελεγχόμενη από την εντολή SIM του 8085.

Το σχήμα 21 δείχνει πώς συνδέεται το μεγάφωνο. Η έξοδος SOD του 8085 περνά από έναν απομονωτή και στέλνεται σε έναν ακροδέκτη εξόδου (edge connector), για τη χρησιμοποίησή του από εξωτερικό hardware. Στη συνέχεια περνάει από έναν ακόμα απομονωτή και μία αντίσταση 100Ω και καταλήγει στο μεγάφωνο.

Η αντίσταση σε σειρά με το μεγάφωνο περιορίζει το ρεύμα ώστε να μην καταστραφεί ο απομονωτής αλλά και ταυτόχρονα η ένταση του ήχου να βρίσκεται σε ικανοποιητικό επίπεδο. Το άκρο του μεγαφώνου βρίσκεται σε τάση +5V και αυτό γιατί ο TTL απομονωτής μπορεί να απορροφήσει περισσότερο ρεύμα παρά να δώσει.

Με χρήση software ελέγχουμε τον τόνο και τη διάρκεια του ήχου που θα παραχθεί, όχι όμως την ένταση. Το πρόγραμμα BEEP στη ROM ανοίγει και κλείνει τη σειριακή έξοδο χιλιάδες φορές το δευτερόλεπτο, οδηγώντας το μεγάφωνο με τετραγωνικό παλμό.

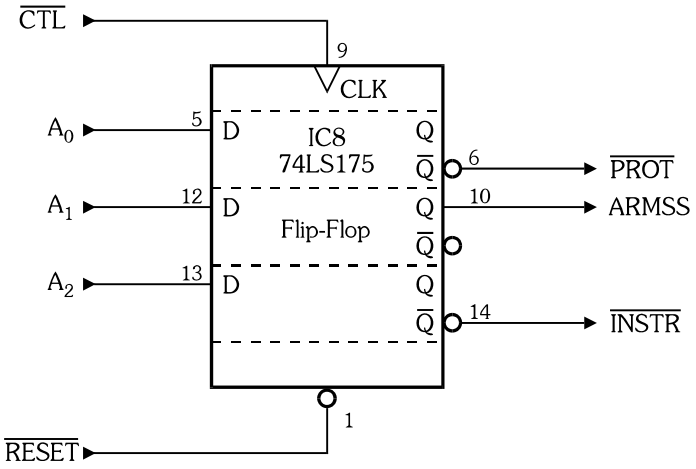


Σχήμα 21. Κύκλωμα σειριακής εξόδου του μLab

4.13 Η πόρτα ελέγχου

Το μLab περιλαμβάνει μία επιπλέον πόρτα εξόδου, την πόρτα ελέγχου που δεν είναι εμφανής στο χρήστη. Χρησιμοποιείται για να στέλνει ο μικροεπεξεργαστής σήματα σε ειδικά κυκλώματα. Έχει τρεις εισόδους και τρεις εξόδους του 1 bit. Το bit PROT ελέγχει την προστασία της μνήμης. Αν το bit αυτό είναι set, τα πρώτα 3/4 της RAM είναι προστατευμένα (write protected). Τα άλλα δύο bit ελέγχουν τα κυκλώματα για HDWR και INSTR single step, που θα μελετήσουμε αργότερα.

Το σχήμα 22 δείχνει τον καταχωρητή της πόρτας ελέγχου του μLab. Η επιλογή του γίνεται από το σήμα CTL του αποκωδικοποιητή διευθύνσεως, σύμφωνα με όσα ισχύουν και για τις άλλες πόρτες I/O.



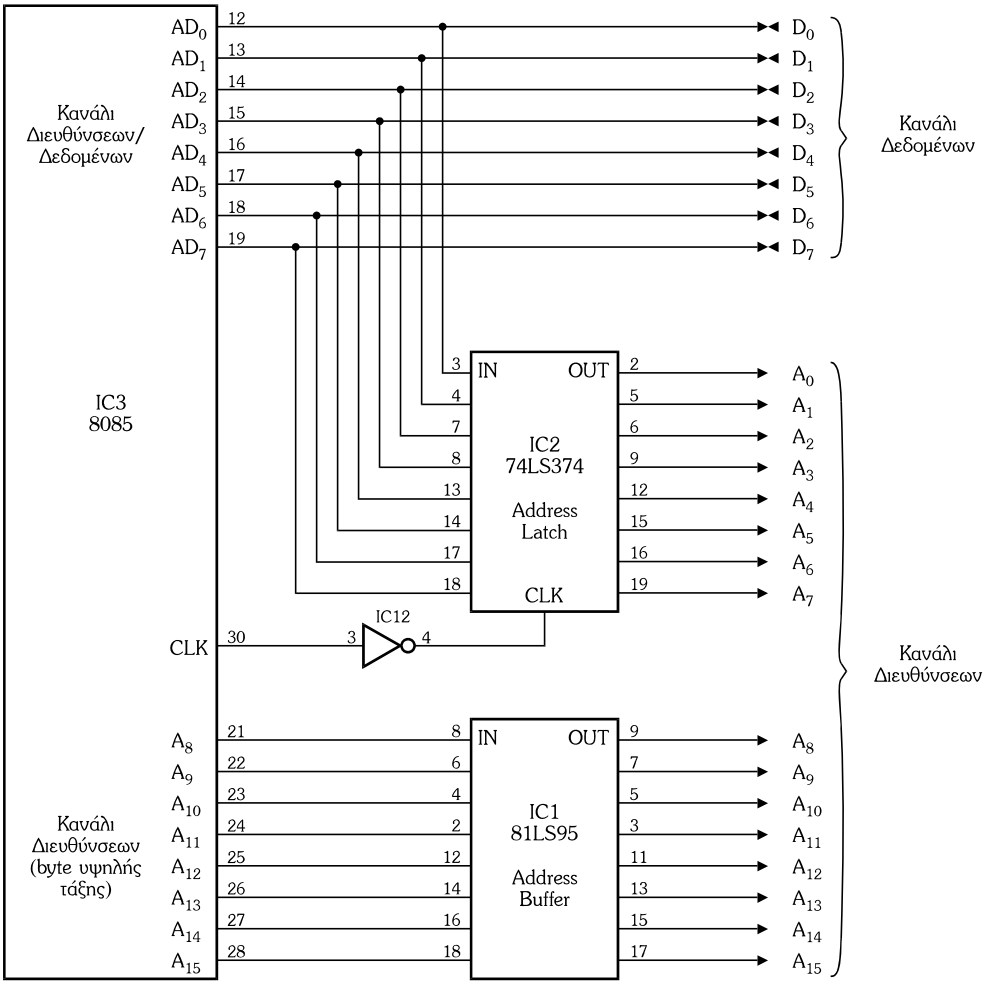
Σχήμα 22. Ο καταχωρητής της πόρτας ελέγχου του μLab

Το ασυνήθιστο για την πόρτα ελέγχου είναι ότι οι είσοδοι δεδομένων της είναι συνδεδεμένες στο διάδρομο διεύθυνσης αντί στο διάδρομο δεδομένων. Δηλαδή, τα δεδομένα που γράφονται στην πόρτα δεν εξαρτώνται από την κατάσταση του διαδρόμου δεδομένων.

Πώς όμως τελικά λειτουργεί η πόρτα ελέγχου; Κοιτάζοντας ξανά το χάρτη μνήμης παρατηρούμε πως η πόρτα επιλέγεται από οποιαδήποτε διεύθυνση από 1000H-17FFH. Αυτό επιτρέπει στα 11 λιγότερο σημαντικά ψηφία να πάρουν οποιαδήποτε τιμή. Παράλληλα από το σχήμα 22 φαίνεται πως οι είσοδοι της πόρτας ελέγχου είναι μόνο τα A_0, A_1 και A_2. Άρα η διεύθυνση στην οποία θα γίνει αναφορά καθορίζει την κατάσταση της πόρτας ελέγχου. Για παράδειγμα, μια εγγραφή στη διεύθυνση 1000H θέτει όλες τις εξόδους 0, ενώ στη διεύθυνση 1001H κάνει set την PROT. (Για τη λειτουργία του PROT, βλέπε και σχήμα 13).

4.14 Διάδρομος πολύπλεξης

Από την αρχή των σημειώσεων έχουμε υποθέσει την ύπαρξη ενός 16-bit διαδρόμου διεύθυνσης και ενός ξεχωριστού 8-bit διαδρόμου δεδομένων. Ο 8085 όμως πολυπλέκει τις ακίδες του διαδρόμου δεδομένων με τις ακίδες του κάτω μισού του διαδρόμου διεύθυνσης. Τα υπόλοιπα 8-bit του πάνω μισού του διαδρόμου διεύθυνσης αποτελούν ξεχωριστές ακίδες. Για να δημιουργηθούν οι δύο ξεχωριστοί διάδρομοι χρησιμοποιείται το κύκλωμα αποπολύπλεξης του σχήματος 23.



Σχήμα 23. Κύκλωμα αποπολύπλεξης της διεύθυνσης στο μLab

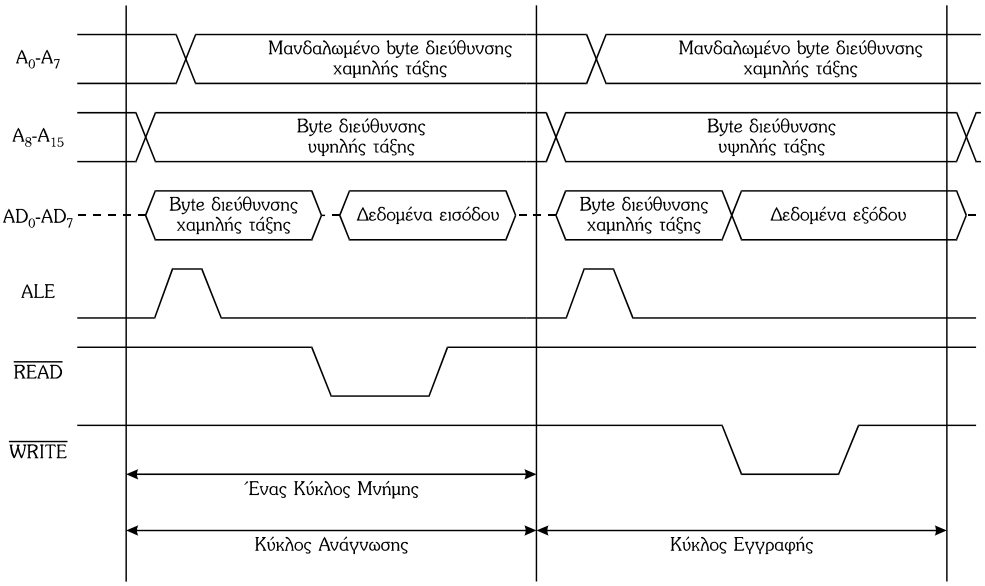
Το σήμα address latch enable (ALE), δείχνει πότε ο διάδρομος διεύθυνσης-δεδομένων (τα σήματα που παρέχει ο 8085) περιέχει πληροφορίες διεύθυνσης. Χρησιμοποιείται για να μανδαλώσει το περιεχόμενο του διαδρόμου αυτού για το κάτω μισό του διαδρόμου διεύθυνσης του συστήματος.

Το IC2 είναι ένας μανδαλωτής 8-bit με έξοδο 3-καταστάσεων. Αυτό μανδαλώνει την πληροφορία διεύθυνσης από το διάδρομο διεύθυνσης-δεδομένων στο αρνητικό άκρο του ALE (ο αναστροφέας IC12 είναι απαραίτητος για να επιλέξει αυτό το άκρο).

Το IC1 είναι ένας απλός απομονωτής 3-καταστάσεων και δεν ανήκει στην αποπολύπλεξη. Οι γραμμές A_8-A_{15} περιέχουν πάντοτε το υψηλής τάξης byte διεύθυνσης.

Στην αρχή κάθε κύκλου μνήμης, το χαμηλής τάξης byte διεύθυνσης παρέχεται στο διάδρομο διευθύνσεων-δεδομένων. Η πτώση από high σε low του παλμού ALE δείχνει την εμφάνιση του byte αυτού, κάνοντας το μανδαλωτή της αποπολύπλεξης IC2 να το αποθηκεύσει. Η πληροφορία διεύθυνσης τότε μετακινείται από το διάδρομο διευθύνσεων-δεδομένων αφήνοντας χώρο στα δεδομένα που θα μεταφερθούν.

Στη συνέχεια δίνεται ένα σχήμα στο οποίο φαίνεται ο χρονισμός των σημάτων του διαδρόμου του μLab.



Σχήμα 24. Χρονισμός των σημάτων του διαδρόμου

Αν έχουμε διαδικασία ανάγνωσης ο μικροεπεξεργαστής δέχεται τα δεδομένα που διαβάζονται και η μνήμη ή η συσκευή Ε/Ε που έχει υποδειχθεί από τη διεύθυνση παρέχει τα δεδομένα στο διάδρομο. Ο κύκλος της εγγραφής είναι παρόμοιος, εκτός από το ότι η κατεύθυνση μεταφοράς των δεδομένων αντιστρέφεται.

Με αυτή τη σύνδεση δημιουργείται το ακόλουθο πρόβλημα. Ο διάδρομος δεδομένων περιέχει στην αρχή κάθε κύκλου μνήμης πληροφορίες διεύθυνσης, δηλαδή μη έγκυρα δεδομένα. Εφόσον όμως δε χρησιμοποιείται σ' αυτό το χρόνο

(κανένα από τα READ και WRITE δεν είναι true), δεν έχουμε λάθος αποτελέσματα.

Με την προσθήκη του μανδαλωτή αποπολύπλεξης η λειτουργία των διαδρόμων είναι παρόμοια με διάδρομο χωρίς πολύπλεξη. Ο πολυπλεγμένος διάδρομος αφήνει 7 από τις 40 ακίδες του 8085 για άλλες λειτουργίες (16 ακίδες διεύθυνσης και 8 ακίδες δεδομένων αντικαθιστούνται από 8 ακίδες διευθύνσεις, 8 διεύθυνσης-δεδομένων και 1 ALE). Οι καθιερωμένες συσκευές μνήμης και Ε/Ε συνδέονται με το διάδρομο μέσω του απλού 8-bit μανδαλωτή αποπολύπλεξης.

4.15 Reset

Ο ακροδέκτης reset του 8085 όταν δεχτεί σήμα χαμηλής στάθμης "καθαρίζει" τα εσωτερικά του κυκλώματα. Ο μετρητής προγράμματος μηδενίζεται και η εκτέλεση του προγράμματος αρχίζει από τη διεύθυνση 0000Η της ROM, με ταυτόχρονο μηδενισμό των εσωτερικών καταχωρητών. Το πρόγραμμα που βρίσκεται εκεί κάνει έλεγχο του συστήματος και θέτει τα περιφερειακά σε κατάσταση έναρξης.

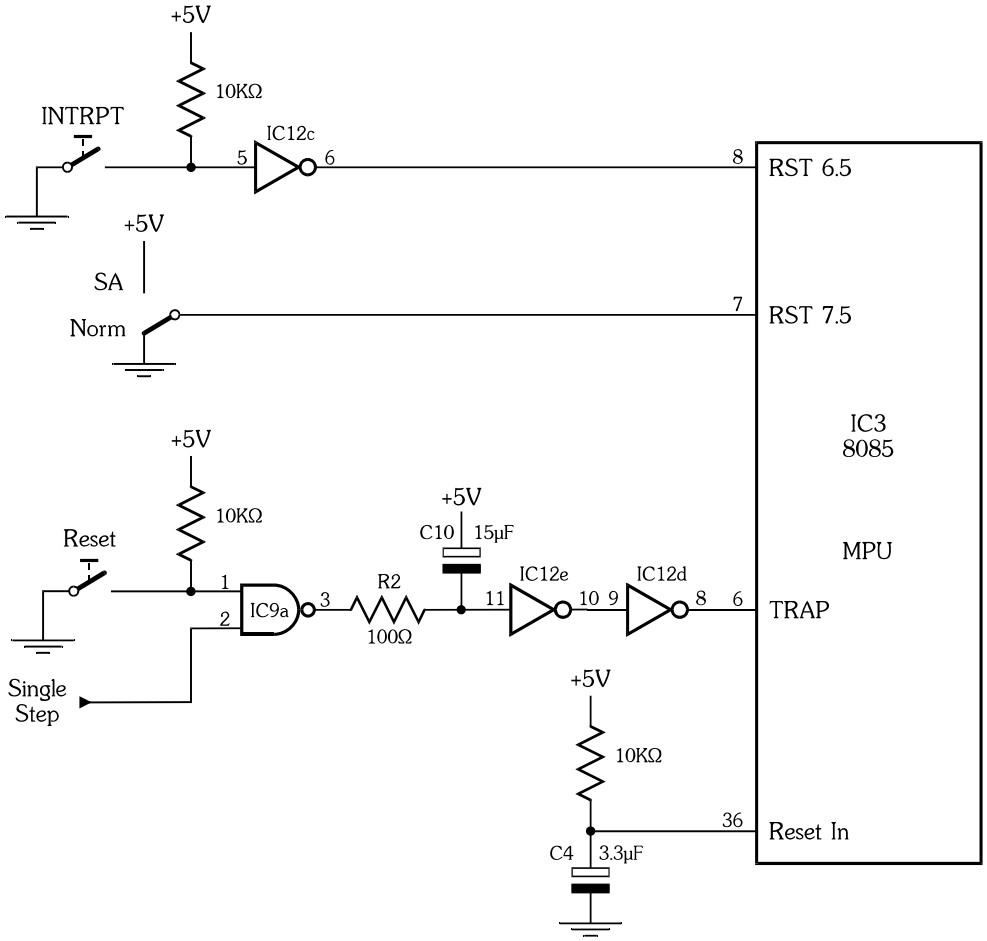
Στο σχήμα 25 φαίνεται η χρησιμοποίηση του συστήματος διακοπών του 8085 από το μLab.

Ο ακροδέκτης reset είναι συνδεδεμένος στο άκρο ενός γειωμένου πυκνωτή. Αυτός, μόλις ανοίξουμε το σύστημα, βρίσκεται σε χαμηλή στάθμη με αποτέλεσμα να γίνεται η αρχικοποίηση σύμφωνα με τα παραπάνω. Ταυτόχρονα, η αντίσταση 10Κ φέρνει γρήγορα τον πυκνωτή σε υψηλή στάθμη, πράγμα απαραίτητο για τη λειτουργία του συστήματος.

Το πλήκτρο RESET δεν συνδέεται απευθείας στην είσοδο reset του μικροεπεξεργαστή. Αν γινόταν έτσι, πατώντας το, το μLab θα πήγαινε στη ρουτίνα 0000Η στη ROM η οποία καταστρέφει όλα τα φυλαγμένα προγράμματα της RAM. Γι' αυτό το πλήκτρο reset συνδέεται στην είσοδο TRAP, που θα περιγραφεί αργότερα.

4.16 Ρολόι

Συνήθως, οι μικροεπεξεργαστές λειτουργούν σε ταχύτητες που είναι κλάσμα της συχνότητας του κρυστάλλου του μικροϋπολογιστικού συστήματος που τους φιλοξενεί. Στο μLab ο 8085 χρησιμοποιεί έναν κρύσταλλο 4MHz. Ο βασικός όμως κύκλος μηχανής είναι 2MHz, όπως και το σήμα clock out που παρέχεται από το μικροεπεξεργαστή.



Σχήμα 25. Κύκλωμα διακοπών του μLab

4.17 Status

Ο 8085 έχει δύο εξόδους κατάστασης S_0 και S_1. Αυτές παρέχουν πρόσθετες πληροφορίες γύρω από τον τρέχοντα κύκλο μηχανής. Επίσης, δείχνουν πότε ο μικροεπεξεργαστής είναι σε κατάσταση halt. Στο μLab πάντως τα σήματα αυτά δε χρησιμοποιούνται, αφού συνήθως τα βλέπουμε σε ειδικές εφαρμογές.

4.18 Ready

Όταν η είσοδος ready του 8085 γίνεται low, ο μικροεπεξεργαστής εισέρχεται σε κατάσταση αναμονής (wait state). Οι διάδρομοι παραμένουν στην κατάσταση που βρίσκονταν πριν, μέχρι να ξαναγίνει το ready high.

Το μlab χρησιμοποιεί τη γραμμή ready στη λειτουργία hardware step. Τότε η γραμμή ready πέφτει σε low αμέσως μετά από κάθε κύκλο μηχανής, παγώνοντας τελείως το σύστημα. Έτσι ο χρήστης παίρνει πληροφορίες για το status και τους διαδρόμους σε κάθε κύκλο μηχανής.

Αυτή η λειτουργία χρησιμοποιείται και για την καθυστέρηση του διαβάσματος από το μικροεπεξεργαστή από αργές μνήμες. Ο αποκωδικοποιητής διευθύνσεων μνήμης κάνει low τη γραμμή ready για όσο χρόνο χρειάζεται η μνήμη να δώσει δεδομένα.

4.19 Είσοδος ελέγχου συγκράτησης

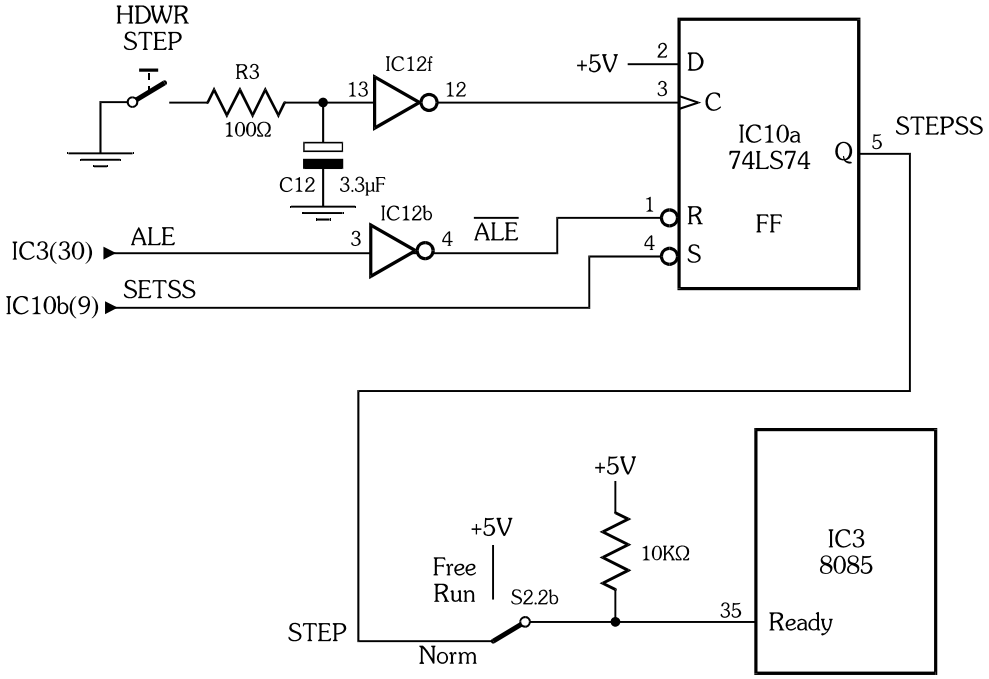
Η είσοδος ελέγχου hold βοηθά στην απευθείας μεταφορά δεδομένων από τις μνήμες σ' ένα περιφερειακό (disk controller ή CRT), παρακάμπτοντας το μικροεπεξεργαστή. Όταν η γραμμή hold είναι high ο 8085 τελειώνει τον κύκλο μηχανής που εκτελεί και σταματά να δίνει στάθμη high στη γραμμή hold acknowledge (HLDA). Όλες οι έξοδοι του μικροεπεξεργαστή είναι σε κατάσταση υψηλής αντίστασης, οπότε η περιφερειακή συσκευή μπορεί να χρησιμοποιήσει τους διαδρόμους για να μεταφέρει δεδομένα από τη μνήμη. Ο μικροεπεξεργαστής συνεχίζει από εκεί που έμεινε, όταν η συσκευή θέσει το hold low.

4.20 Διακοπές

Υπάρχουν δύο ομάδες διακοπών για τον 8085. Η πρώτη ομάδα (TRAP, RST 5.5, 6.5, και 7.5) ελέγχεται από καθορισμένες ακίδες του μικροεπεξεργαστή, μία για κάθε διακοπή. Η δεύτερη ομάδα (RST 1,2,3,4,5,6 και 7) ελέγχεται όλη από τις δύο ακίδες INTR και INTA.

Το μLab χρησιμοποιεί την είσοδο TRAP για το πλήκτρο RESET, την είσοδο RST 6.5 για το πλήκτρο INTRPT και την είσοδο RST 7.5 για το διακόπτη "SA" (θα εξεταστεί αργότερα). Σε κάθε διακοπή αντιστοιχεί και μία προτεραιότητα, με αποτέλεσμα η διακοπή με τη μεγαλύτερη προτεραιότητα να εξυπηρετείται πρώτη. Η διακοπή TRAP έχει τη μεγαλύτερη προτεραιότητα και ακολουθούν κατά σειρά οι RST 7.5, 6, 5.5, 5 και INTR.

Ακόμα το μLab έχει ειδικά κυκλώματα ελέγχου για τις λειτουργίες single step. Στο σχήμα 26 φαίνεται το κύκλωμα για λειτουργία single step με το πάτημα του πλήκτρου HDWR STEP.



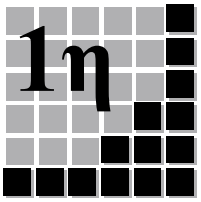
Σχήμα 26. Το κύκλωμα single step (κύκλου μηχανής)

Με το πάτημα του HDWR STEP οι διαδικασίες που γίνονται είναι:

1. Όταν το monitor πρόγραμμα τρέχει, το SETSS (set single step) σήμα ελέγχου (IC 10-9) είναι low, αναγκάζοντας το STEPSS (step single step) σήμα (IC 10-5) να γίνει high. Αυτή η γραμμή έχει συνδεθεί με την είσοδο ready του μικροεπεξεργαστή (IC3-35) διαμέσου του διακόπτη S-2. Η θετική λογική στάθμη στη γραμμή ready κάνει το σύστημα να τρέξει σε πλήρη ταχύτητα.
2. Όταν μία διεύθυνση φτάσει στο display του μLab και το πλήκτρο HDWR STEP πατηθεί, το monitor πρόγραμμα αποκρίνεται μεταφέροντας τον έλεγχο στο πρόγραμμα του χρήστη και κατόπιν θέτει τη γραμμή SETSS high.
3. Στον επόμενο κύκλο μηχανής η γραμμή ALE έχει παλμό low κάνοντας reset τον μανδαλωτή IC10A και αναγκάζοντας τη γραμμή STEPSS να γίνει low. Αυτή η πτώση, με τη σειρά της, αναγκάζει τη γραμμή ready να γίνει low.
4. Με τη γραμμή ready να είναι low όλες οι ενέργειες του συστήματος σταματούν. Το μLab τώρα είναι σε κατάσταση hardware single step και η διεύθυνση που

ήταν προηγουμένως στο display εμφανίζεται τώρα στα LEDs του διαδρόμου διεύθυνσης.

5. Όταν το πλήκτρο HDWR STEP πατηθεί ξανά, εμφανίζεται ένα θετικό μέτωπο στην ακίδα IC 10-3, κάνοντας τη γραμμή STEPSS καθώς και τη γραμμή ready high.
6. Όταν η γραμμή ready ξαναγίνει high ο μικροεπεξεργαστής συνεχίζει την κανονική εκτέλεση του προγράμματος, πηγαίνοντας στον επόμενο κύκλο μηχανής.
7. Μόλις φτάσει ο επόμενος κύκλος μηχανής και μια καινούργια διεύθυνση μανδαλωθεί, η γραμμή ALE γίνεται low. Έτσι, αυτό έχει ξανά σαν αποτέλεσμα ο μανδαλωτής IC10A να κάνει reset και η γραμμή STEPSS να γίνει low.
8. Αυτή η διαδικασία επαναλαμβάνεται κάθε φορά με το πάτημα του HDWR STEP.
9. Όταν το πλήκτρο reset πατηθεί, το IC10B κάνει reset αναγκάζοντας τη γραμμή SETSS να γίνει low, με αποτέλεσμα η γραμμή STEPSS να γίνει high. Το σήμα reset στέλνει επίσης τη διακοπή TRAP στο μικροεπεξεργαστή (IC3-6). Αυτή η διακοπή τώρα στέλνει το σύστημα στο monitor πρόγραμμα.



1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

- 1. Βασικές γνώσεις για την εκτέλεση προγραμμάτων στο μLab
  - Εφαρμογή 1: Εξετάζοντας τα περιεχόμενα της μνήμης
  - Εφαρμογή 2: Μνήμη και φύλαξη δεδομένων
  - Εφαρμογή 3: Εκτέλεση προγραμμάτων
  - Εφαρμογή 4: Οι πόρτες εισόδου και εξόδου

1. Βασικές γνώσεις για την εκτέλεση προγραμμάτων στο μLab

Οι βασικές λειτουργίες του μLab συνίστανται στη φύλαξη δεδομένων στη μνήμη, στην εξέταση των περιεχομένων της μνήμης και στην εκτέλεση προγραμμάτων. Τα διάφορα προγράμματα μπορούν να εκτελεστούν είτε σε πλήρη ταχύτητα είτε ανά βήμα έτσι ώστε να μας επιτρέπουν να παρακολουθούμε τη λειτουργία τους λεπτομερώς. Οι λειτουργίες αυτές εξετάζονται στη συνέχεια.

Εφαρμογή 1: Εξετάζοντας τα περιεχόμενα της μνήμης

Ι) Θεωρητικό Μέρος

Σ' αυτήν την εφαρμογή θα εξετάσουμε τα περιεχόμενα της μνήμης και στη συνέχεια θα τα μετατρέψουμε από γλώσσα μηχανής σε κώδικα assembly. Θα πραγματοποιήσουμε δηλαδή την αντίστροφη διαδικασία απ' ότι συνήθως. Σε αυτό θα μας χρησιμεύσει ο πίνακας του παραρτήματος 2.

ΙΙ) Πρακτικό Μέρος

1. Πατήστε [RESET].
2. Πατήστε [FETCH ADRS] [07F0]. Αυτό δηλώνει ότι θέλουμε να εξετάσουμε τη διεύθυνση 07f0. Το περιεχόμενό της είναι 3E, και το καταγράφουμε στον παρακάτω πίνακα.
3. Πατήστε [STORE INC] για να εξετάσουμε την επόμενη θέση μνήμης.
4. Καταγράψτε το περιεχόμενο της θέσης μνήμης στον πίνακα που συμπληρώνετε.
5. Επαναλάβατε τα βήματα 3 και 4 έως ότου γεμίσετε τον πίνακα.
6. Συμβουλευόμενοι τώρα το παράρτημα 2 στο οποίο καταγράφονται όλοι οι opcodes και τα αντίστοιχα mnemonics γεμίζουμε τη στήλη mnemonics του παραπάνω πίνακα. Θυμηθείτε ότι δεν περιέχει κάθε διεύθυνση και opcode. Μερικές εντολές ακολουθούνται από data ή jump address.

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
07F0	3E - opcode	START:	MVI A,0
07F1	00 - data		
07F2	3C - opcode	LOOP:	INR A
07F3	FE - opcode		CPI 10d
07F4	0A - data		
07F5	CA - opcode		JZ START
07F6	F0 - address		
07F7	07 - address		
07F8	C3 - opcode		JMP LOOP
07F9	F2 - address		
07FA	07 - address		

Εφαρμογή 2: Μνήμη και φύλαξη δεδομένων

Ι) Θεωρητικό Μέρος

Για την εξέταση και τη μετατροπή των περιεχομένων της μνήμης του μLab χρησιμοποιούνται 3 πλήκτρα. Το πλήκτρο FETCH ADRS χρησιμοποιείται για την εξέταση του περιεχομένου της διεύθυνσης μνήμης που εισάγουμε από το πληκτρολόγιο. Το πλήκτρο STORE/INCR χρησιμοποιείται για το σώσιμο δεδομένων ή τον έλεγχο διαδοχικών θέσεων μνήμης (θα αναπτυχθεί στη συνέχεια). Τέλος το πλήκτρο DECR χρησιμοποιείται στην εξέταση θέσεων μνήμης προς τα πίσω.

ΙΙ) Πρακτικό Μέρος

Σώσιμο δεδομένων στη μνήμη

1. Πατήστε [FETCH ADRS] [0800]. Αυτό ορίζει τη διεύθυνση 0800, που αποτελεί και την αρχή της μνήμης RAM. Όταν ανοίγουμε το μLab οι θέσεις μνήμης από 0800 και πάνω γεμίζουν με 00. Έτσι σ' αυτή τη θέση μνήμης εμφανίζει υπάρχει το 00.
2. Πατήστε [STORE/INCR]. Έτσι, προχωράμε στην επόμενη θέση μνήμης και αφήνουμε το 00 αποθηκευμένο στη διεύθυνση 0800.
3. Πατήστε [C3]. Παρατηρήστε ότι αυτό εμφανίζεται στα δεξιότερα ψηφία και η υποδιαστολή στο πιο δεξί display ανάβει υποδηλώνοντας ότι βρισκόμαστε σε entry mode και ότι πρόκειται να αλλάξουμε τα περιεχόμενα αυτής της θέσης μνήμης.

4. Πατήστε [STORE/INCR]. Έτσι τώρα σώσαμε το C3 στη διεύθυνση 0801 και η υποδιαστολή στο δεξιότερο ψηφίο σβήνει υποδηλώνοντας ότι δεν βρισκόμαστε πλέον σε entry mode. Τώρα η διεύθυνση 0802 εμφανίζεται στα displays με το περιεχόμενό της.
5. Πατήστε [8] [STORE/INCR]. Παρατηρήστε ότι εισάγοντας μόνο ένα ψηφίο το μLab αυτόματα θέτει το 0 πριν από αυτό.
6. Πατήστε τώρα [DECR]. Έτσι μπορούμε να εξετάσουμε τα δεδομένα που εισάγαμε προηγουμένως.
7. Επαναλάβατε το βήμα 6 για να διαπιστώσετε ότι τα περιεχόμενα της μνήμης δεν μεταβάλλονται με το πλήκτρο [DECR].

Διορθώνοντας τυχόν λάθη

1) Πατήστε [FETCH ADRS] [0000]. Αυτό επιλέγει την πρώτη διεύθυνση στη ROM.
2) Πατήστε [0] [STORE/INCR]. Προσπαθήσατε μόλις να σώσετε την τιμή 00 στην παραπάνω διεύθυνση.
3) Τι έγινε; Το σύστημα αποκρίθηκε με ένα beep, η διεύθυνση δεν αυξήθηκε κατά ένα και το περιεχόμενο της διεύθυνσης δε μεταβλήθηκε.
4) Πώς θα αντιδράσουμε σε περίπτωση λανθασμένης διεύθυνσης ή δεδομένων; Απλά, πατάμε πάλι [FETCH ADRS] και εισάγουμε την σωστή διεύθυνση ή τα σωστά δεδομένα ακολουθώντας ακριβώς τα βήματα που αναλύσαμε πρωτύτερα.

Χρήσιμες παρατηρήσεις:

1. Τα δεδομένα που μόλις εισάγαμε είναι στην ουσία το παρακάτω πρόγραμμα:

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0800	00	START:	NOP	; Καμία λειτουργία
0801	C3		JMP START	; Jump στο START
0802	00			
0803	08			

Το opcode 00 αντιστοιχεί στο mnemonic NOP. Το opcode C3 αντιστοιχεί σε εντολή jump.

2. Παρατηρήστε ότι οι διευθύνσεις που εισάγουμε γράφονται με το LSByte πρώτα και το MSByte μετά. Έτσι η διεύθυνση άλματος 0800 σώζεται σαν 00 στην θέση 0802 και 08 στη θέση 0803.

Εφαρμογή 3: Εκτέλεση προγραμμάτων

Ι) Θεωρητικό Μέρος

Το μLab έχει 3 τρόπους εκτέλεσης προγραμμάτων. Τη συνηθισμένη λειτουργία του πλήκτρου RUN για πλήρη ταχύτητα εκτέλεσης και 2 modes απλού βήματος (single stepping), δηλαδή με δυνατότητα εκτέλεσης μιας εντολής ή ενός κύκλου μηχανής τη φορά.

ΙΙ) Πρακτικό Μέρος

Εκτέλεση με χρήση του instruction step (βήμα εντολής)

1. Πατήστε [FETCH ADRS] [0800]. Ακολουθήστε τα βήματα της εφαρμογής 2 για την εισαγωγή του προγράμματος εκείνης της εφαρμογής.
2. Πατήστε [FETCH ADRS] για να επιστρέψετε στην αρχή του προγράμματος.
3. Πατήστε [INSTR STEP]. Αυτό έχει σαν αποτέλεσμα το display να δείξει την επόμενη εντολή. Θα πείτε βέβαια ότι το ίδιο φαίνεται να έκανε και το [STORE/INCR]. Η απάντηση είναι περίπου ναι. Η διαφορά συνίσταται στο ότι ενώ με το δεύτερο πλήκτρο εξετάζουμε απλώς διαδοχικές θέσεις μνήμης με το πρώτο προκαλούμε ΕΚΤΕΛΕΣΗ της εντολής που εμφανίζουν τα displays.
4. Τώρα, η εντολή jump (C3) φαίνεται στα displays. Πατήστε [INSTR STEP]. Ο επεξεργαστής εκτελεί την εντολή κάνοντας άλμα στη διεύθυνση 0800. Παρατηρήστε ότι τα addresses δεν εμφανίζονται μια και με το πλήκτρο αυτό το μLab εκτελεί ολόκληρη την εντολή, η οποία περιλαμβάνει και τα δύο address bytes.
5. Πατώντας συνεχώς το [INSTR STEP] παρατηρούμε διαδοχική εκτέλεση του loop.

Εκτέλεση με χρήση του hardware step (βήμα κύκλου μηχανής)
(Η ΛΕΙΤΟΥΡΓΙΑ ΑΥΤΗ ΔΕΝ ΥΠΆΡΧΕΙ ΣΤΟΝ ΠΡΟΣΟΜΟΙΩΤΗ)

1. Πατήστε [FETCH ADRS] [0800].
2. Πατήστε [HDWR STEP]. Τα displays σβήνουν επειδή η χρήση του hardware step προκαλεί σταμάτημα του processor μόλις εκτελεστεί ένας κύκλος μηχανής. Η ιδιαιτερότητα του hardware step συνίσταται στο ότι με την εκτέλεση μιας εντολής ο έλεγχος δεν επιστρέφει στο monitor πρόγραμμα.
3. Παρατηρήστε τα 16 LEDs με την επιγραφή ADDRESS που είναι συνδεδεμένα απ' ευθείας με το address bus. Έχουν την ένδειξη 0000 1000 0000 0000. Συμβουλευόμενοι τον πίνακα δεξιά στα LEDs μετατρέπουμε σε hex μορφή. Αυτή η τιμή πρέπει να είναι η διεύθυνση που καθορίσαμε στο βήμα Α (0800).

4. Παρατηρήστε τα LEDs με επιγραφή DATA. Αυτά αποτελούν την οπτική αναπαράσταση του data bus και δείχνουν 00 όσο και τα δεδομένα στη διεύθυνση 0800.
5. Παρατηρήστε τα 6 status LEDs. Αυτά δείχνουν αν πρόκειται για εντολή read ή write, αν αναφερόμαστε στη ROM, τη RAM, την πόρτα εισόδου ή την πόρτα εξόδου. Προς το παρόν τα LEDs READ, RAM είναι αναμμένα υποδεικνύοντας ότι πληροφορία διαβάζεται από τη RAM.
6. Πατήστε [HDWR STEP]. Η διεύθυνση αυξάνει και τα LEDs παρουσιάζουν την πληροφορία της νέας διεύθυνσης. Όπως και με το instruction step η εντολή εκτελείται.
7. Τα data LEDs τώρα δείχνουν το opcode C3 (1100 0011) της εντολής jump. Πατήστε τώρα [HDWR STEP] και δείτε ότι η διεύθυνση αυξάνεται αλλά το άλμα ΔΕΝ πραγματοποιείται. Αυτό αποτελεί και την ειδοποιό διαφορά μεταξύ hardware step και instruction step. Το πρώτο προχωρά μία διεύθυνση μνήμης τη φορά, εκτελώντας ένα κύκλο μηχανής, ενώ το δεύτερο προχωρά μια εντολή τη φορά, που πιθανόν να αποθηκεύεται σε περισσότερες από μια διευθύνσεις.
8. Πατήστε [HDWR STEP] [HDWR STEP]. Τώρα μόλις εκτελέστηκε η εντολή άλματος και τα address LEDs δείχνουν 0800.
9. Πατήστε συνεχώς το παραπάνω πλήκτρο και παρατηρήστε την συνεχή εκτέλεση του προγράμματος.
10. Πατήστε [RESET] αν κατανοήσατε πλέον πώς λειτουργεί το πλήκτρο αυτό. Δείτε όμως ότι η εντολή που περίμενε να εκτελεστεί εκτελείται και προχωράμε στην επόμενη πριν ο έλεγχος επιστρέψει στο monitor πρόγραμμα.

Εκτέλεση του προγράμματος με το πλήκτρο RUN (πλήρης ταχύτητα)

1. Πατήστε [FETCH ADRS] [0800].
2. Πατήστε RUN. Το πρόγραμμα τρέχει σε full speed. (Περίπου 2 μsec ανά εντολή).
3. Παρατηρήστε πάλι τα address LEDs. Δείχνουν 0803 (0000 1000 0000 0011). Στην πραγματικότητα μετρούν από την 0800 ως 0803 σε δυαδική μορφή και συνεχώς, αλλά αλλάζουν τόσο γρήγορα που τα δύο τελευταία LEDs φαίνονται διαρκώς αναμμένα.
4. Τα status LEDs δείχνουν ανάγνωση από RAM, όπως και πριν.
5. Τα data LEDs για λόγους που δεν μας αφορούν προς το παρόν φαίνονται όλα ON. Είναι χρήσιμα μόνο στο hardware step mode.
6. Πατήστε [RESET]. Επιστρέφουμε στο monitor και απεικονίζεται η εντολή που επρόκειτο να εκτελεστεί όταν πατήσαμε [RESET].

Παρατηρήσεις:

1. Το μLab δεν πρόκειται να ανταποκριθεί στο πλήκτρο RUN ενόσω βρισκόμαστε σε entry mode και το δεκαδικό σημείο (υποδιαστολή) είναι αναμμένο στο δεξιότερο ψηφίο.
2. Παρατηρήστε ότι το πλήκτρο hardware step έχει δύο λειτουργίες. Ενόσω δε βρισκόμαστε σε hardware step mode πίεση του μας φέρνει σ' αυτήν και δεύτερη πίεση επενεργεί όπως αναπτύξαμε προηγουμένως, δηλαδή προχωράει το πρόγραμμα κατά ένα κύκλο μηχανής.

Εφαρμογή 4: Οι πόρτες εισόδου και εξόδου

Ι) Θεωρητικό Μέρος

Το μLab, όπως αναφέραμε και προηγουμένως, έχει μια πόρτα εισόδου μέσω ενός dip switch για τον καθορισμό των δεδομένων εισόδου (8 γραμμές). Εξάλλου διαθέτει και μια πόρτα εξόδου που απαρτίζεται από 8 LEDs τριών χρωμάτων, ένα για κάθε γραμμή εξόδου. Ο 8085 μπορεί και παίρνει τα δεδομένα του από την πόρτα εισόδου και να απεικονίζει αποτελέσματα στην πόρτα εξόδου.

Στη συνέχεια θα εισάγουμε το παρακάτω πρόγραμμα:

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0900	3A	START:	LDA 2000	; Ανάγνωση πόρτας εισόδου
0901	00			
0902	20			
0903	32		STA 3000	; Έξοδος data στην πόρτα εξόδου
0904	00			
0905	30			
0906	C3		JMP START	; Loop στην αρχή
0907	00			
0908	09			

Τι κάνει το παραπάνω πρόγραμμα; Απλά, διαβάζει δεδομένα από την πόρτα εισόδου (μόλις ανακαλύψατε ότι αντιστοιχεί στη διεύθυνση 2000) και τα τοποθετεί στην πόρτα εξόδου (που αντιστοιχεί στη διεύθυνση 3000). Όταν λοιπόν χρειάζεται να διαβάσουμε δεδομένα εισόδου φορτώνουμε τα περιεχόμενα της θέσης μνήμης 2000 σαν να ήταν μία κοινή θέση μνήμης στον καταχωρητή A (accumulator). Ανάλογα για έξοδο στα LEDs, σώζουμε τον accumulator στην 3000.

ΙΙ) Πρακτικό Μέρος

1. Πατήστε [FETCH ADRS] [0900].
2. Περάστε το παραπάνω πρόγραμμα στη μνήμη όπως έχουμε μάθει σε προηγούμενες εφαρμογές.
3. Αφού ελέγξετε την ορθότητα του προγράμματος τρέξτε το εκτελώντας [FETCH ADRS] [0900] [RUN].
4. Θέσατε τα input switches σε τυχαίες θέσεις. Πάνω είναι το λογικό 1 και κάτω το 0.
5. Δείτε τα output LEDs. Απεικονίζουν ότι παριστάνει η πόρτα εισόδου. Θυμηθείτε ότι όταν μια γραμμή εξόδου είναι high, το αντίστοιχο LED είναι σβηστό (η εξήγηση είναι θέμα κατασκευής κυκλωμάτων και δε θα μας απασχολήσει. Πάντως συνηθέστερα τα LEDs συνδέονται μ' αυτόν τον τρόπο).
6. Αλλάξτε τώρα τα switches. Τα LEDs πρέπει να αλλάξουν ανάλογα.
7. Πατήστε [RESET]. Το πρόγραμμα σταματά και ο έλεγχος επιστρέφει στο monitor πρόγραμμα. Οποιαδήποτε αλλαγή στα switches πλέον δεν επιφέρει αλλαγή στα LEDs μια και το πρόγραμμα έχει σταματήσει την εκτέλεσή του.
8. Τώρα, προσθέστε την εντολή CMA που επιφέρει συμπλήρωμα ως προς 1 στον Α. Δηλαδή, εισάγετε την εντολή:

Διεύθυνση	Περιεχόμενο	Εντολή	Σχόλια
0903	2F	CMA	; Συμπληρωματικά data

   μετακινώντας τις επόμενες εντολές κατά μια διεύθυνση μνήμης.
9. Τρέξτε πάλι το πρόγραμμα και διαβάστε τα LEDs εξόδου για να δείτε αν απεικονίζουν αυτό που περιμένατε.

Παρατήρηση:

Η τελευταία αλλαγή επέτρεψε την απεικόνιση της πόρτας εισόδου στην πόρτα εξόδου, διαβάζοντας τα αναμμένα LEDs σαν ON και τα σβηστά σαν OFF. Αυτό αναδεικνύει την ευελιξία και την προσαρμοστικότητα ενός συστήματος βασισμένου σε μικροεπεξεργαστή στις απαιτήσεις εφαρμογών. Σκεφτείτε ότι αν τα switches ήταν συνδεδεμένα με τη μεσολάβηση κυκλώματος με τα LEDs θα χρειάζονταν 8 inverters για να επιτευχθεί η παραπάνω απαίτηση. Αλλά εφόσον είναι συνδεδεμένα μέσω επεξεργαστή μια μικρή αλλαγή στο πρόγραμμα είναι το μόνο που χρειάζεται.

2. Τα θέματα της 1ης εργαστηριακής άσκησης

Αφού εκτελέσετε τις εφαρμογές που περιγράφονται στο πρώτο τμήμα, προχωρήστε στην υλοποίηση των παρακάτω θεμάτων που θα πρέπει να επιδείξετε στο τέλος του εργαστηρίου, εκτός από τα θέματα iii και iv, που η ανάπτυξή τους μπορεί να γίνει στο σπίτι. Η έκθεση της εργαστηριακής άσκησης θα πρέπει να παραδίδεται κάθε φορά στο επόμενο εργαστήριο.

i. Να μελετηθούν με hardware step οι εντολές:

```
0800:  LXI H, 0BOO
       MVI M, 7A
       MVI A, 3F
       ADD M
```

Για κάθε κύκλο μηχανής να καταγραφούν και να δοθούν στην αναφορά οι τιμές του διαδρόμου δεδομένων και διευθύνσεων, τα σήματα ελέγχου, το είδος του κύκλου μηχανής και η λειτουργία που επιτελείται. Οι κωδικοί Hex της γλώσσας assembly του 8085 βρίσκονται στον πίνακα 1 του παραρτήματος 2.

ii. Να κατασκευαστεί χρονόμετρο δευτερολέπτων που θα απεικονίζει το χρόνο σε δυαδική μορφή πάνω στα LEDs εξόδου του μLab. Φροντίστε το άναμμα των LED να αντιστοιχεί σε λογικό 1 και αντίστροφα. Για την υλοποίηση της χρονοκαθυστέρησης του 1 sec μπορείτε να χρησιμοποιήσετε την έτοιμη ρουτίνα DELB που υπάρχει στο παράρτημα 1. Το χρονόμετρο όταν φτάνει στην τιμή 15_{10} , στο επόμενο βήμα να ξαναρχίζει από την αρχή. Προαιρετικά, πριν το ξεκίνημα να ενεργοποιείται η ρουτίνα BEEP. Τονίζεται το γεγονός ότι ορισμένοι καταχωρητές επηρεάζονται από τις ρουτίνες του συστήματος (βλέπε Παράρτημα 1).

iii. Αφού μελετηθεί η παράγραφος 4.13 της εισαγωγής να αιτιολογηθεί γιατί **η εντολή IN 10 αίρει την προστασία της μνήμης**. Ακόμα να δοθεί και άλλη μια εντολή που έχει το ίδιο αποτέλεσμα και να γραφεί πρόγραμμα που το επαληθεύει. Το θέμα αυτό είναι θεωρητικό. Δεν θα ασχοληθείτε με αυτό την ώρα του εργαστηρίου, αλλά θα το αναπτύξετε στην έκθεση.

iv. Να γραφεί σε assembly το παρακάτω πρόγραμμα που δίνεται σε γλώσσα μηχανής και να εξηγηθεί η λειτουργία του:

06 08 3A 00 20 17 DA 0D 08 05 C2 05 08 78 2F 32 00 30 CF

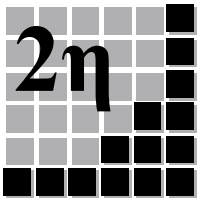
Το πρόγραμμα είναι φορτωμένο στη μνήμη με αρχή τη διεύθυνση 0800.

Η διαδικασία της αποκωδικοποίησης θα διευκολυνθεί με τη χρήση του πίνακα 2 του παραρτήματος 2. Επίσης να γίνει και το διάγραμμα ροής του προγράμματος. Τι αλλαγές πρέπει να γίνουν για να έχουμε σε συνεχόμενη μορφή τη λειτουργία του παραπάνω προγράμματος;

Υπόδειξη: Μπορείτε να ακολουθήσετε τη διαδικασία της εφαρμογής 1 της παρούσας εργαστηριακής άσκησης.

ΠΡΟΑΙΡΕΤΙΚΗ ΑΣΚΗΣΗ

v. Να γραφεί σε assembly πρόγραμμα που να απεικονίζει ένα αναμμένο led που να κινείται αριστερά και όταν φτάνει στο όγδοο να κινείται δεξιά κ.ο.κ. όταν το LSB dip switch είναι ON. Οποτεδήποτε, είτε στην αρχή είτε ενδιάμεσα, γίνεται OFF να σταματάει εκεί που βρίσκεται. Στη συνέχεια, όταν ξαναγίνει ON να συνεχίζεται η κίνηση του αναμμένου led. (Διάρκεια ανάμματος ½ sec).



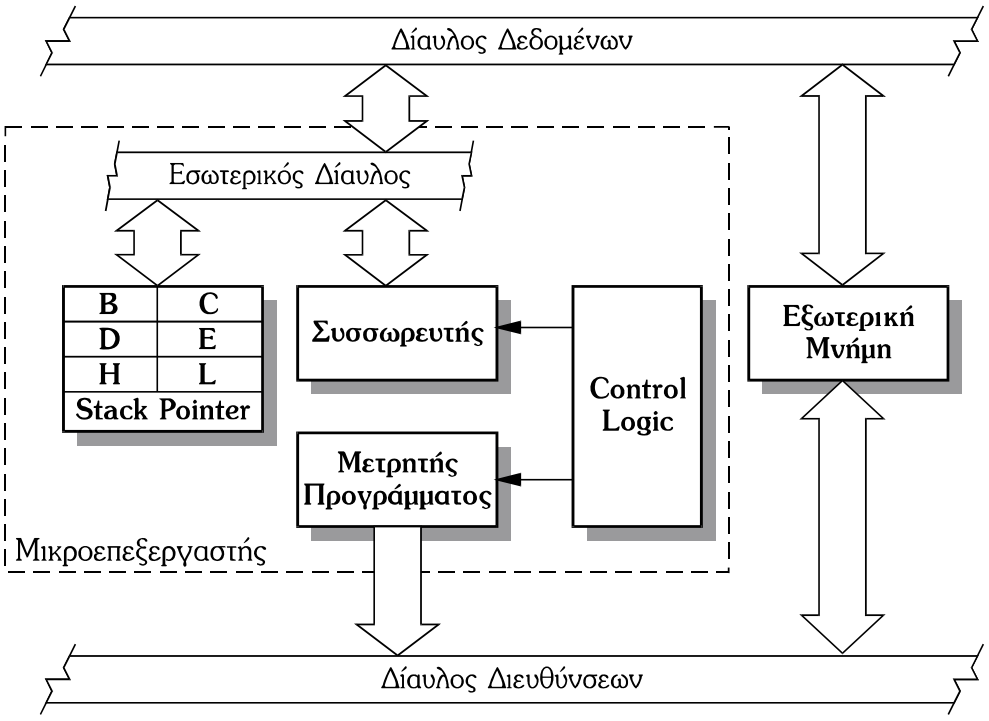
2η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

- 1. Στοιχεία θεωρίας
- 2. Παραδείγματα προγραμμάτων
  - ΕΦΑΡΜΟΓΗ 1: Εκτέλεση του προγράμματος μετρητή
  - ΕΦΑΡΜΟΓΗ 2: Υπορουτίνες
  - ΕΦΑΡΜΟΓΗ 3: Χρήση διακοπών

1. Στοιχεία θεωρίας

1.1 Οι καταχωρητές του μΕ 8085

Ο μικροεπεξεργαστής 8085 περιέχει καταχωρητές διαφόρων χρήσεων. Αυτοί δεν επιλέγονται από το διάδρομο διεύθυνσης, αλλά ελέγχονται κατευθείαν από τη λογική ελέγχου του επεξεργαστή, όπως φαίνεται στο σχήμα 2.1. Ο συσσωρευτής (accumulator, A) είναι γενικής χρήσης (για αποθήκευση αποτελεσμάτων και ορισμάτων πράξεων). Επίσης γενικής χρήσης είναι και οι καταχωρητές B, C, D, E, H, L. Ο program counter (PC) έχει ειδική χρήση και συγκεκριμένα συνεχώς δείχνει τη διεύθυνση της επόμενης εντολής που θα εκτελεστεί.



Σχήμα 2.1. Καταχωρητές και μνήμη

1.2 Έλεγχος ροής προγράμματος-καταχωρητών

Για να δούμε τα περιεχόμενα των καταχωρητών στη βηματική εκτέλεση ενός προγράμματος μπορούμε να χρησιμοποιήσουμε το ειδικό πλήκτρο [FETCH REG]. Υπάρχει επίσης το πλήκτρο [FETCH PC] με το οποίο μπορούμε να επιστρέψουμε σ' ένα σταματημένο πρόγραμμα.

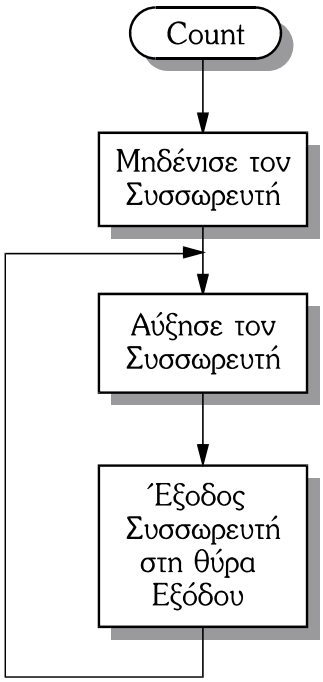
1.3 Το monitor πρόγραμμα του μLab

Η ROM του μLab περιέχει το monitor πρόγραμμα που διαβάζει το πληκτρολόγιο, εκτελεί τη ζητούμενη λειτουργία και ελέγχει το display. Το monitor αποθηκεύει τα περιεχόμενα των καταχωρητών κατά την έξοδο από ένα πρόγραμμα και έπειτα από κάθε εντολή σε ειδικές θέσεις της RAM. Από εκεί τα ανακαλεί με τη χρήση των προαναφερθέντων πλήκτρων.

2. Παραδείγματα προγραμμάτων

2.1 Ένα πρόγραμμα μέτρησης

Για να δούμε καλύτερα τη λειτουργία αυτών των πλήκτρων εισάγουμε ένα πρόγραμμα που μετράει δυαδικά από το 0 έως το 255 και ξαναρχίζει. Το διάγραμμα ροής του φαίνεται στο σχήμα 2.2.



Σχήμα 2. 2. Πρόγραμμα μετρητής

Το listing του προγράμματος φαίνεται στον πίνακα 2.1. Το πρόγραμμα αρχίζει στη διεύθυνση 0804 γιατί θα χρειαστούν κάποιες προσθήκες αργότερα. Για τον ίδιο λόγο χρησιμοποιούνται και μερικές εντολές NOP.

Πίνακας 2.1. Πρόγραμμα μετρητής

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0804	3E		MVI A,0	; Μηδένισε τον A
0805	00			
0806	32	LOOP:	STA 3000	; Μετέφερε τον A στη θύρα εξόδου
0807	00			
0808	30			
0809	00		NOP	; Για προσθήκη
080A	00		NOP	
080B	00		NOP	
080C	3C		INR A	; Αύξησε τον A
080D	C3		JMP LOOP	; Loop
080E	06			
080F	08			

ΕΦΑΡΜΟΓΗ 1: Εκτέλεση του προγράμματος μετρητή

I) Θέμα

Σ' αυτό το πείραμα θα εισάγουμε και θα εκτελέσουμε το πρόγραμμα που περιγράψαμε. Θα δούμε τη λειτουργία των πλήκτρων [FETCH REG] και [FETCH PC].

II) Διαδικασία

Εισαγωγή του προγράμματος

1. [FETCH ADRS] [0] [8] [0] [4].
2. [E] [STORE/INCR].
3. [STORE/INCR].
4. Φόρτωσε και το υπόλοιπο πρόγραμμα.
5. [DECR]. Έλεγχος των θέσεων της μνήμης με επανάληψη του βήματος

Βηματική εκτέλεση του προγράμματος με χρήση του [INSTR STEP]

1. [FETCH ADRS] [0] [8] [0] [4].
2. [INSTR STEP]. Εκτέλεση της MVI A.
3. Εμφανίζεται ο κώδικας της STA (32). [INSTR STEP]. Τα LEDs ανάβουν (αρνητική λογική).
4. [INSTR STEP] τρεις φορές. Εκτέλεση των NOP, INR A, JMP.
5. Ο A έχει αυξηθεί. Για να το δούμε [instr step].
6. [INSTR STEP] επαναληπτικά για να δεις το Loop.
7. Σταμάτα όταν εμφανιστεί η NOP. [FETCH REG]. Τώρα φαίνεται το "A" ακολουθούμενο από μια τιμή. Αυτή είναι η τιμή του συσσωρευτή μετά το τελευταίο βήμα. Τα δεδομένα (σε δυαδικό) αντιστοιχούν στα LEDs εξόδου. Γιατί;
8. [STORE/INCR]. Στο display φαίνεται ο flags register.
9. [STORE/INCR]. Επαναληπτικά ώσπου να δεις PCH. Αυτό είναι το πιο σημαντικό μέρος του PC (08).
10. [FETCH PC] για να συνεχίσουμε την εκτέλεση του προγράμματος από εκεί που είχε σταματήσει.
11. [INSTR STEP]. Τώρα βλέπουμε να συνεχίζεται η βηματική εκτέλεση.
12. Εκτελέστε διάφορα loops χρησιμοποιώντας [INSTR STEP] σταματώντας στη NOP (διεύθ. 0809) και κάθε φορά πατάτε [FETCH REG] για να δείτε την αύξηση της τιμής του A καθώς και των LEDs εξόδου.

Βηματική εκτέλεση με χρήση του [HDWR STEP]

1. [FETCH ADRS] [0] [8] [0] [4] [HDWR STEP]. Το display παραμένει κενό μια και είσαστε στο hardware step mode. Η διεύθυνση 0804 εμφανίζεται στα LEDs δυαδικής διεύθυνσης. Τα δεδομένα εμφανίζονται στα LEDs δυαδικής διεύθυνσης (κώδικας 3E). Όταν χρησιμοποιείτε [HDWR STEP] πρέπει να αναφέρεστε στα LEDS δυαδικής διεύθυνσης και δεδομένων.
2. [HDWR STEP]. Εδώ έρχεται το 2ο byte της εντολής (00).
3. [HDWR STEP]. Εκτέλεση της εντολής
4. [HDWR STEP] 4 φορές. Έτσι εκτελείται η STA 3000 και τα LEDs ανάβουν.

5. Το πρόγραμμα φτάνει στην NOP και πατάμε reset για να επιστρέψουμε στο μόνιτορ. Η NOP εκτελείται και βλέπουμε τη διεύθυνση της επόμενης εντολής (080A).
6. [FETCH REG]. Για να δούμε το περιεχόμενο του A.
7. [FETCH PC] [hdwr step] για επιστροφή στο πρόγραμμα.
8. [HDWR STEP] επαναληπτικά και επαναλάβετε τα βήματα 5, 6, 7, για να δείτε την αύξηση του A στα LEDs εξόδου.
9. [RESET] για επιστροφή στην κανονική λειτουργία του μLAB.

III) Σχόλια

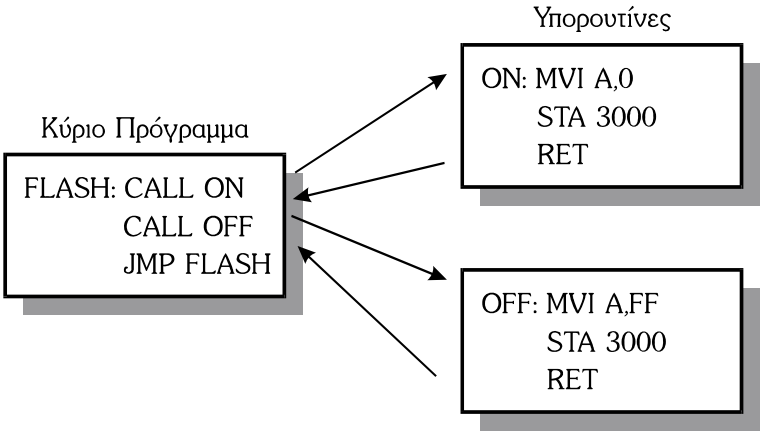
Σ' αυτό το πείραμα φορτώσαμε το πρόγραμμα μετρητή που έχουμε αναλύσει προηγουμένως. Το εκτελέσαμε βηματικά χρησιμοποιώντας και τα δυο step modes. Είδαμε τη λειτουργία των πλήκτρων: [FETCH REG], [FETCH PC] και [RESET].

Είδαμε ακόμη ότι στο hardware step mode, η εντολή STA απαιτεί ένα επιπλέον βήμα για να εκτελεστεί. Ένα βήμα hardware απαιτείται για κάθε αναφορά στη μνήμη ή στις θύρες Ε/Ε. Έτσι η STA χρειάζεται 3 βήματα για να διαβάσει τα 3 bytes της εντολής και ένα 4ο για να γράψει τα δεδομένα στη θύρα εξόδου.

2.2 Υπορουτίνες

Το σχήμα 2.3 δείχνει ένα απλό πρόγραμμα που χρησιμοποιεί υπορουτίνες για να αναβοσβήνει τα LEDs επαναληπτικά. Μια υπορουτίνα χρησιμοποιείται για να τα ανάβει και μια για να τα σβήνει.

Το κύριο πρόγραμμα έχει μόνο 3 εντολές: μια που καλεί την ρουτίνα ON, μια που καλεί την ρουτίνα OFF και μια που ξαναγυρνά στην αρχή.



Σχήμα 2.3. Πρόγραμμα που αναβοσβήνει τα LEDs

Στη συνέχεια παρουσιάζουμε αναλυτικά την εφαρμογή.

Πίνακας 2.2. Listing του προγράμματος που αναβοσβήνει τα LEDs

Κύριο πρόγραμμα

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0800	CD	FLASH:	CALL ON	; Άναψε τα LEDs
0801	19			
0802	08			
0803	00		NOP	; Για προσθήκη
0800	00		NOP	
0805	00		NOP	
0806	CD		CALL OFF	; Σβήσε τα LEDs
0807	1F			
0808	08			
0809	00		NOP	; Για προσθήκη
080A	00		NOP	
080B	00		NOP	
080C	C3		JMP FLASH	; Επανέλαβε
080D	00			
080E	08			

Υπορουτίνα για να ανάβουν τα LEDs

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0819	3E	ON:	MVI A,00	; Βάλε στον Α
081A	00			; μηδενικά
081B	32		STA 3000	; Γράψε τα
081C	00			; περιεχόμενα του Α
081D	30			; στη θύρα εξόδου
081E	C9		RET	; Τέλος υπορουτίνας

Υπορουτίνα για να σβήνουν τα LEDs

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
081F	3E	OFF:	MVI A,FF	; Βάλε στον Α άσσους
0820	FF			
0821	32		STA 3000	; Γράψε τα
0822	00			; περιεχόμενα του Α
0823	30			; στη θύρα εξόδου
0824	C9		RET	; Γύρισε στο κύριο πρόγραμμα

Σημείωση: Για να γίνει επιστροφή από υπορουτίνα στο κύριο πρόγραμμα χρησιμοποιείται η RET. Όταν γίνεται μια κλήση CALL η διεύθυνση επιστροφής φυλάγεται σε μια στοίβα απ' όπου ανακαλείται μέσω της RET. Έτσι η ροή του προγράμματος επιστρέφει στο κύριο πρόγραμμα μόλις τελειώσει η υπορουτίνα.

ΕΦΑΡΜΟΓΗ 2: Υπορουτίνες

Ι) Θέμα

Το πρόγραμμα FLASH που περιγράφηκε εισάγεται στο μLAB και εκτελείται. Θα παρατηρήσετε το άλμα της ροής του προγράμματος από το κύριο πρόγραμμα στις υπορουτίνες και την επιστροφή σ' αυτό.

ΙΙ) Διαδικασία

1. Πληκτρολογήστε το πρόγραμμα του πίνακα 2.2 και επαληθεύστε αν αποθηκεύτηκε σωστά.
2. [FETCH ADRS] [0800]. Στο display εμφανίζεται η πρώτη CALL. Με την εκτέλεσή της [INSTR STEP] ο έλεγχος πηγαίνει στην διεύθυνση 0819. Εκτελέστε σειριακά την υπορουτίνα χρησιμοποιώντας την [INSTR STEP]. Παρατηρήστε επίσης ότι ο έλεγχος του προγράμματος μετά την εκτέλεση της εντολής RET μεταφέρεται αμέσως μετά το αντίστοιχο CALL που προκάλεσε την εκτέλεση της υπορουτίνας.
3. [INSTR STEP] επαναληπτικά. Τώρα εκτελείται το πρόγραμμα και βλέπουμε τα LEDs της θύρας εξόδου να αναβοσβήνουν ανάλογα με την υπορουτίνα που εκτελείται.
4. Στο κύριο πρόγραμμα να προστεθούν δύο κλήσεις της DELB με καθυστέρηση 0.5sec. Για να μη χάσετε τον κώδικα συνεχίστε με το πρώτο θέμα της εργαστηριακής άσκησης.

2.3 Διακοπές

Συχνά θέλουμε ένα μΥ.Σ. να μπορεί να δρα σε γεγονότα τα οποία είναι εντελώς απεριοδικά και μη προβλεπόμενα. Η δράση αυτή ονομάζεται διακοπή. Οι μΕ έχουν εισόδους για το σκοπό αυτό.

Όταν γίνεται σε ένα μικροεπεξεργαστή διακοπή ανεξάρτητα από το τι συμβαίνει εκείνη τη στιγμή ο έλεγχος μεταφέρεται σε μια προκαθορισμένη ρουτίνα που ονομάζεται ρουτίνα εξυπηρέτησης της διακοπής. Το μLab έχει το πλήκτρο INTRPT που είναι συνδεμένο με τη διακοπή RST6.5 του μΕ 8085. Όπως γνωρίζουμε όταν συμβεί μια διακοπή τέτοιου τύπου ο επεξεργαστής "κάνει άλμα" στη διεύθυνση 0034 (hex). Οι σχεδιαστές της ROM του μLab για να επιτρέπουν να

γράφουμε δικές μας ρουτίνες εξυπηρέτησης έχουν βάλει στη διεύθυνση αυτή μια εντολή JMP 0AFCh (μπορείτε να το δείτε με [FETCH ADRS] [0034]). Όπως θα δείτε στο παράδειγμα που ακολουθεί (πίνακας 2.3), από τη διεύθυνση αυτή αρχίζουμε να γράφουμε τη ρουτίνα εξυπηρέτησης διακοπής.

Ένα θέμα που πρέπει να έχουμε επίσης κατά νου όταν σχεδιάζουμε μια ρουτίνα εξυπηρέτησης διακοπής είναι το αν η είσοδος διακοπής είναι ευαίσθητη στο επίπεδο του παλμού ή στο θετικό ή αρνητικό μέτωπο του παλμού διακοπής. Η διακοπή RST6.5 του μLab είναι ευαίσθητη στο επίπεδο του παλμού. Αυτό μπορεί να δημιουργήσει ένα πρόβλημα: αν ο χρόνος εκτέλεσης ρουτίνας εξυπηρέτησης είναι μικρότερος από τη διάρκεια του παλμού που προκαλεί τη διακοπή στον επεξεργαστή, τότε μετά την επιστροφή από τη ρουτίνα εξυπηρέτησης ο επεξεργαστής θα εκλάβει το επίπεδο του παλμού σαν μια νέα αίτηση διακοπής την οποία και θα εξυπηρετήσει. Αυτό μπορεί να δημιουργήσει προβλήματα ειδικά στην περίπτωση που θέλουμε να μετράμε τις διακοπές που γίνονται (βλ. θέμα 2).

Επειδή μερικές φορές θέλουμε ο μικροεπεξεργαστής να αγνοεί κάποιες διακοπές, αυτές μπορούν να απενεργοποιηθούν από το πρόγραμμα. Για παράδειγμα η διακοπή που παράγεται από το πλήκτρο [INTRPT] απενεργοποιείται από το πρόγραμμα monitor του συστήματος όταν αυτό είναι επιθυμητό. Για να επιδείξουμε αυτή τη διαδικασία το πρόγραμμα μετρητής που είδαμε μπορεί να διακοπεί από το [INTRPT]. Αρκεί να προσθέσουμε στην αρχή του προγράμματος τις ακόλουθες εντολές που ενεργοποιούν τη διακοπή.

Πίνακας 2.3. Εντολές που ενεργοποιούν τη διακοπή RST6.5

Διεύθυνση	Περιεχόμενο	Εντολή	Σχόλια
0800	3E	MVI A,0D	; Βάλε τιμή για μάσκα
0801	0D		; διακοπών στον A
0802	30	SIM	; Βάλε τον A στη μάσκα διακοπών
0803	FB	EI	; Ενεργοποίησε τις διακοπές RST6.5

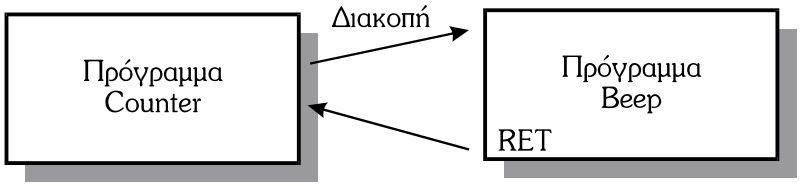
Η μάσκα διακοπών είναι ένας ειδικός καταχωρητής στον επεξεργαστή που καθορίζει ποια διακοπή θα είναι ενεργοποιημένη. Αυτή η μάσκα περιγράφεται μαζί με την εντολή SIM. Η εντολή EI προκαλεί την ενεργοποίησή της.

Όταν μια διακοπή ενεργοποιηθεί απαιτείται μια ρουτίνα εξυπηρέτησης. Το monitor περιέχει μια υπορουτίνα που προκαλεί στο ηχείο την παραγωγή ενός ήχου.

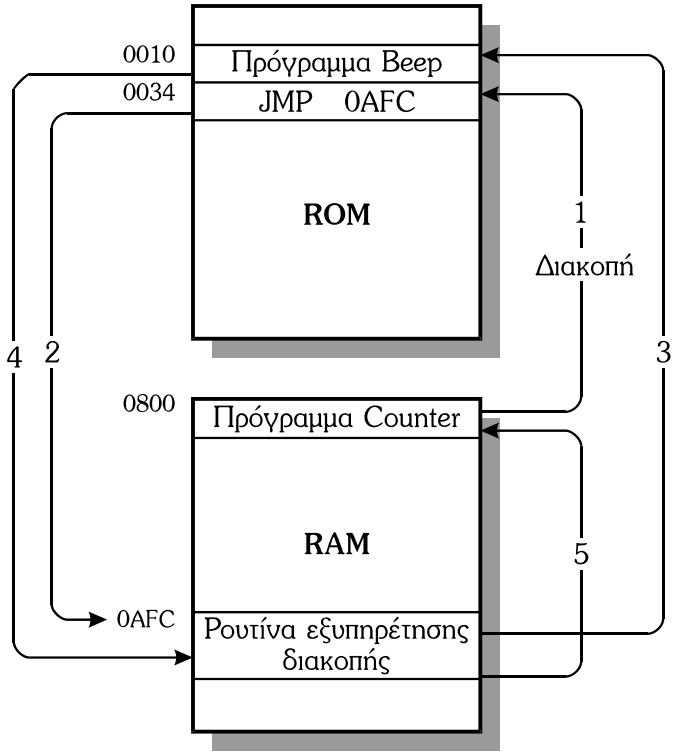
Ο πίνακας 2.4 δείχνει μια ρουτίνα εξυπηρέτησης διακοπών που καλεί την υποροutίνα beep (αποθηκευμένη στη διεύθυνση 0010).

Πίνακας 2.4. Ρουτίνα εξυπηρέτησης διακοπής

Διεύθυνση	Περιεχόμενο	Εντολή	Σχόλια
0AFC	CD	CALL BEEP	; Άλμα στη ρουτίνα
0AFD	10		; beep
0AFE	00		
0AFF	FB	EI	; Ενεργοποίησε τις διακοπές
0B00	C9	RET	; Επιστροφή στο πρόγραμμα



Σχήμα 2.4. Επικοινωνία προγράμματος counter και ρουτίνας beep με χρήση διακοπής



Σχήμα 2.5. Αναλυτική παρουσίαση επικοινωνίας με χρήση διακοπής

Η ρουτίνα beep τελειώνει με εντολή RET η οποία μετά από ένα ήχο που παράγει δίνει τον έλεγχο στη ρουτίνα εξυπηρέτησης της διακοπής.

Το σχήμα 2.4 δείχνει τα παραπάνω. Στο σχήμα 2.5 φαίνεται πιο λεπτομερειακά η ροή του προγράμματος. Η εντολή EI στο πρόγραμμα είναι απαραίτητη γιατί ο μικροεπεξεργαστής αυτόματα απενεργοποιεί τις διακοπές όταν αναγνωρίσει μία.

ΕΦΑΡΜΟΓΗ 3: Χρήση διακοπών

Ι) Θέμα

Σε αυτό το πείραμα θα τρέξουμε το πρόγραμμα μετρητή μαζί με τις εντολές που ενεργοποιούν το πλήκτρο [INTRPT]. Μια ρουτίνα εξυπηρέτησης διακοπής καλεί το BEEP πρόγραμμα του monitor. Άρα τώρα το πρόγραμμα του μετρητή μπορεί να διακόπτεται και πατώντας το [INTRPT] παράγεται ένας χαρακτηριστικός ήχος.

ΙΙ) Διαδικασία

1. Πληκτρολογήστε το πρόγραμμα του πίνακα 1.
2. Πληκτρολογήστε το πρόγραμμα του πίνακα 3 για να ενεργοποιήσετε τις διακοπές.
3. Πληκτρολογήστε τη ρουτίνα εξυπηρέτησης της διακοπής του πίνακα 4.
4. Τρέξτε το πρόγραμμα από τη διεύθυνση 0800. Η θύρα εξόδου μετράει συνεχώς αλλά τόσο γρήγορα ώστε όλα τα LEDs φαίνονται αναμμένα.
5. [INTRPT]. Το μLab εκτελεί τη BEEP ρουτίνα για όση ώρα το πλήκτρο [INTRPT] παραμένει πατημένο (γιατί;). Τα LEDs της θύρας εξόδου παραμένουν στην κατάσταση που βρίσκονταν πριν την εκτέλεση της διακοπής. Όταν αφήσετε το [INTRPT] το πρόγραμμα μετρητής συνεχίζει την εκτέλεσή του.
6. [RESET] για να σταματήσετε το πρόγραμμα.

Βρείτε τις διαδοχικές τιμές του PC και σχολιάστε τις.

3. Τα θέματα της 2ης εργαστηριακής άσκησης

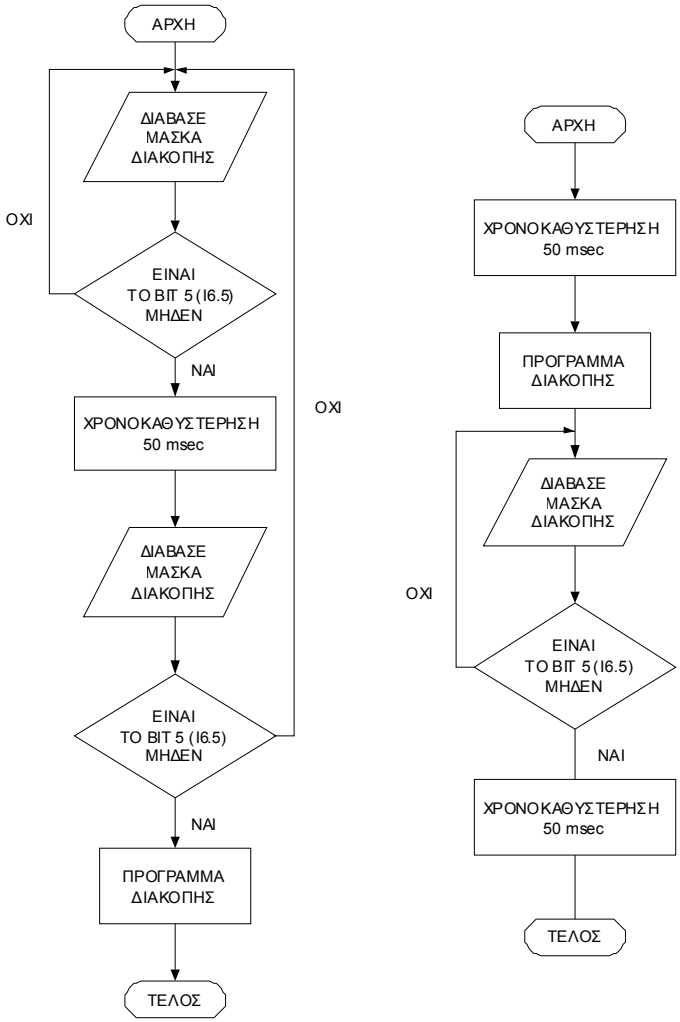
i. Μετατρέψτε το παράδειγμα του πίνακα 2, ώστε η καθυστέρηση στο άναμμα και στο σβήσιμο των LEDs (πόρτα 3000 Hex) θα καθορίζεται από τις τιμές των 4 αριστερότερων και δεξιότερων αντίστοιχα διακοπτών της πόρτας 2000 Hex. Δίνεται ότι η μικρότερη καθυστέρηση είναι 200 msec, η μεγαλύτερη 1700 msec και το βήμα 100 msec.

ii. α) Να μετατραπεί το πρόγραμμα διακοπής (πίνακες 1, 3 και 4) έτσι ώστε να επιτρέπει διακοπές μόνο όταν το MSB της πόρτας 2000 Hex είναι ON, αλλιώς όχι. Ο μετρητής που αποτελεί το κύριο πρόγραμμα να απεικονίζεται στα 4 LSB των LEDs (πόρτα 3000 Hex) και να τρέχει με ταχύτητα μίας μέτρησης ανά δέκατο του δευτερολέπτου. Η μέτρηση του πλήθους των διακοπών να δίνεται στα 4 MSB (modulo 16) των LEDs.

β) Μια άλλη ρουτίνα εξυπηρέτησης της διακοπής όταν ενεργοποιείται να απεικονίζει στα 4 MSB των LEDs τον αριθμό των διακοπτών (dip switches) που είναι ON.

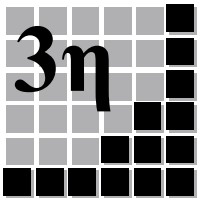
Υπόδειξη: Λάβετε υπ' όψιν την παρατήρηση της εφαρμογής 2 που αφορά σε πολλαπλές διακοπές. Μια λύση για να ξεπεραστεί το πρόβλημα είναι να εξετάζουμε το bit 5 (I6.5) της μάσκας διακοπών (η ανάγνωσή της γίνεται με την εντολή RIM), που δείχνει το αν είναι ενεργοποιημένη ή όχι κάποια διακοπή. Σημειώνουμε ότι από τη στιγμή που πατιέται το πλήκτρο διακοπής, μέχρι να σταθεροποιηθεί στην τιμή 1 το bit I6.5, μεσολαβεί κάποιο χρονικό διάστημα (~50 msec) (εξηγήστε το λόγο). Τα παρακάτω λογικά διαγράμματα δείχνουν δύο τρόπους με τους οποίους μπορούμε να εξασφαλίσουμε την ορθή λειτουργία της ρουτίνας διακοπής (εξηγείστε τη λειτουργία τους). Όσον αφορά στο κύριο πρόγραμμα καλό θα είναι να μην επιτρέπουμε διακοπές στη διάρκεια μιας χρονοκαθυστέρησης. Εξηγήστε το λόγο.

iii. Να υλοποιηθεί αυτοματισμός που να ελέγχει το άναμμα και το σβήσιμο ενός φωτιστικού σώματος. Όταν πατάμε το πλήκτρο INTR ή το LSB της πόρτας 2000 Hex (που ότι αντιστοιχεί σε ένα αισθητήρα κίνησης) να ανάβει το LED (που αντιπροσωπεύει το φωτιστικό σώμα) στο LSB πόρτας 3000 Hex (τα άλλα LED να είναι σβηστά). Το LED θα σβήνει μετά από 5 sec, εκτός αν ενδιάμεσα υπάρξει νέο πάτημα του πλήκτρου INTR ή ενεργοποίηση (ON) του διακόπτης στο LSB πόρτας 2000 Hex, οπότε και ο χρόνος των 5 sec θα ανανεώνεται.



Παρατηρήσεις:

1. Ο έλεγχος των προγραμμάτων μπορεί να γίνει τρέχοντάς τα σε single step mode. Όπου χρησιμοποιείται ρουτίνα Delay, τα τρία bytes που αντιστοιχούν στην κλήση της ρουτίνας, πρέπει να αντικατασταθούν με NOP.
2. Τα προγράμματα, όταν τρέχουν σε single step mode, αίρουν την προστασία της μνήμης και επομένως είναι δυνατό να μην τρέχουν σωστά σε πλήρη ταχύτητα, ενώ σε single step mode να μην παρουσιάζουν κανένα πρόβλημα.
3. Υπενθυμίζεται ότι η εντολή IN 10 στην αρχή του προγράμματός σας, αίρει την προστασία της μνήμης, για όση ώρα τρέχει το συγκεκριμένο πρόγραμμα και σε πλήρη ταχύτητα.



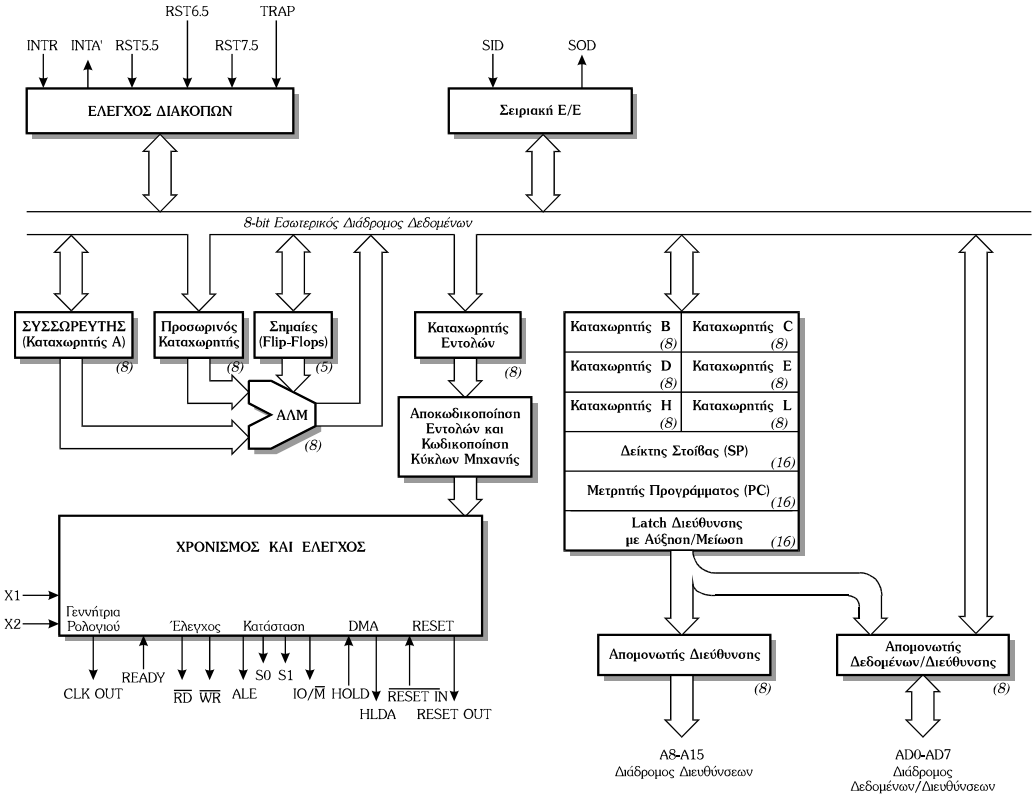
3η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

- 1. Χρήση καταχωρητών και διακοπών
  - ΕΦΑΡΜΟΓΗ 1: Βλέποντας τα περιεχόμενα των καταχωρητών
  - ΕΦΑΡΜΟΓΗ 2: Χρησιμοποιώντας τα breakpoints
- 2. Κατηγορίες εντολών
  - ΕΦΑΡΜΟΓΗ 3: Εντολές AND, OR, XOR
  - ΕΦΑΡΜΟΓΗ 4: Χρησιμότητα των λογικών εντολών
  - ΕΦΑΡΜΟΓΗ 5: Χρησιμοποιώντας εντολές ολίσθησης
- 3. Τεχνικές προγραμματισμού

1. Χρήση καταχωρητών και διακοπών

Το πρώτο μέρος αυτής της άσκησης έχει στόχο την εξοικείωση με τους καταχωρητές του μlab, καθώς και με τη χρήση των διακοπών σαν ένα μέσο για τον έλεγχο και τη διόρθωση προγραμμάτων. Υπάρχουν έξι καταχωρητές γενικής χρήσης των 8 bits στο μLab, οι B, C, D, E, H και L. Παράλληλα, υπάρχει, όπως είναι γνωστό, ο συσσωρευτής A και ο δείκτης στοίβας (stack pointer).

Στο σχήμα 3.1 βλέπουμε ένα χονδρικό διάγραμμα του 8085 όπου φαίνονται όλοι οι παραπάνω.



Σχήμα 3.1. Χονδρικό διάγραμμα του 8085

ΕΦΑΡΜΟΓΗ 1: Βλέποντας τα περιεχόμενα των καταχωρητών

Οι καταχωρητές γενικής χρήσης χρησιμοποιούνται για αποθήκευση ενδιάμεσων αποτελεσμάτων κυρίως όταν ένα πρόγραμμα χρησιμοποιεί πολλές διαφορετικές μεταβλητές. Έτσι αποφεύγεται η μεγάλης έκτασης χρήση της μνήμης ή της στοίβας. Συνίσταται η πληκτρολόγηση του παρακάτω προγράμματος και η εκτέλεση του βήμα-βήμα με τη βοήθεια των πλήκτρων [INSTR STEP] και [FETCH REG], ώστε με την εκτέλεση κάθε εντολής να ελέγχεται το περιεχόμενο των καταχωρητών.

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
0800	06		MVI B,37
0801	37		
0802	60		MOV H,B
0803	24		INR H
0804	16		MVI D,AF
0805	AF		
0806	15		DCR D

Υπενθυμίζεται ότι πατώντας το [FETCH REG] βλέπουμε αρχικά, στο display του μLab το τρέχον περιεχόμενο του A και στη συνέχεια πατώντας το [STORE/INCR] βλέπουμε τα περιεχόμενα των υπολοίπων καταχωρητών με την εξής σειρά:

A : Συσσωρευτής
FL : Flags
B : Γενικής χρήσης καταχωρητής
C : " " "
D : " " "
E : " " "
H : " " "
L : " " "
SPH : 8 MSB του δείκτη στοίβας
SPL : 8 LSB " "
PCH : 8 MSB του μετρητή προγράμματος
PCL : 8 LSB " "
I : Κατάσταση διακοπών

ΕΦΑΡΜΟΓΗ 2: Χρησιμοποιώντας τα breakpoints

Η βηματική εκτέλεση των προγραμμάτων επιτρέπει την παρατήρηση των αποτελεσμάτων κάθε μίας εντολής. Πολλές φορές, όμως, είναι πιο βολικό ένα πρόγραμμα να τρέχει σε κανονική ταχύτητα και να σταματά σε κάποιο συγκεκριμένο σημείο. Αυτό επιτυγχάνεται με τη χρήση μιας εντολής που υποχρεώνει τον επεξεργαστή να επιστρέψει στο monitor πρόγραμμα.

Στο μLab η εντολή αυτή είναι η RST 1(0CFH) που είναι ισοδύναμη με μια CALL 0008. Η διεύθυνση 0008 είναι καθορισμένη και δεν μπορεί να αλλάξει. Η υπορουτίνα που αρχίζει στη διεύθυνση αυτή της ROM σώζει τα περιεχόμενα των καταχωρητών στη RAM και επιστρέφει στο monitor πρόγραμμα.

Οι διακοπές όπως είπαμε είναι ένα πολύ χρήσιμο εργαλείο για τη διόρθωση προγραμμάτων καθώς επιτρέπουν το σταμάτημα του προγράμματος σε κάποιο συγκεκριμένο σημείο και τον έλεγχο για το αν οι καταχωρητές και η μνήμη περιέχουν τα αναμενόμενα αποτελέσματα.

Η RST 1 είναι μια διακοπή λογισμικού. Στο μLab υπάρχει και hardware διακοπή. Το παρακάτω πρόγραμμα χρησιμοποιεί την RST 1 για την παρατήρηση λειτουργίας ενός μετρητή. Πατώντας το πλήκτρο RUN μετά από κάθε διακοπή το πρόγραμμα συνεχίζει από εκεί που είχε σταματήσει.

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
0804	3E		MVI A,0
0805	00		
0806	32	LOOP:	STA 3000
0807	00		
0808	30		
0809	CF		RST 1
080A	3C		INR A
080B	C3		JMP LOOP
080C	06		
080D	08		

2. Κατηγορίες εντολών

2.1 Λογικές εντολές

ΕΦΑΡΜΟΓΗ 3: Εντολές AND, OR, XOR

Στο set εντολών του 8085 υπάρχουν εντολές που υλοποιούν τις λογικές πράξεις NOT, AND, OR και exclusive-OR. Πληκτρολογήστε το παρακάτω πρόγραμμα και τρέξτε το τρεις φορές βάζοντας στη διεύθυνση 0805 την εντολή ANA B την πρώτη φορά, ORA B τη δεύτερη και XRA B την τρίτη.

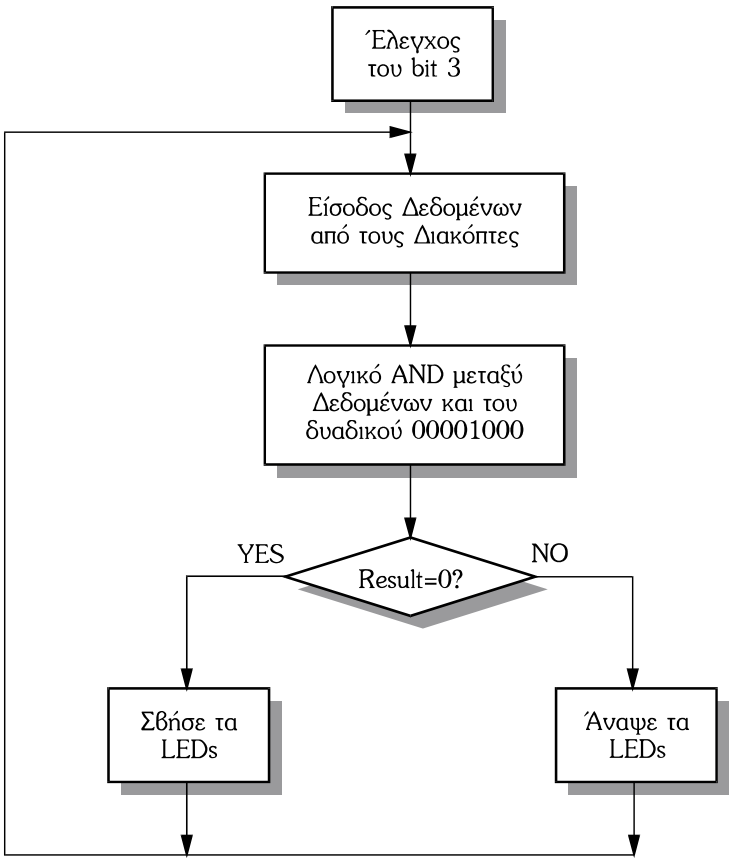
Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
0800	3A	START:	LDA 2000
0801	00		
0802	20		
0803	06		MVI B,3C
0804	3C		
0805	A0		ANA B
0806	32		STA 3000
0807	00		
0808	30		
0809	C3		JMP START
080A	00		
080B	08		

Το πρόγραμμα αυτό διαβάζει ένα δεδομένο από την πόρτα εισόδου 2000 και εκτελεί τη λογική πράξη AND (ή OR ή XOR) μεταξύ αυτού και της ποσότητας 3CH=00111100. Παρατηρούμε ότι με τη χρήση της AND μπορούμε να μηδενίσουμε συγκεκριμένα bits. Με τη χρήση της OR συγκεκριμένα bits γίνονται 1 ενώ με τη χρήση της XOR συγκεκριμένα bits αντιστρέφονται.

ΕΦΑΡΜΟΓΗ 4: Χρησιμότητα των λογικών εντολών

Μια συνηθισμένη ανάγκη στον προγραμματισμό του 8085 είναι η επιλογή συγκεκριμένων bits μιας λέξης. Το παρακάτω πρόγραμμα του οποίου το διάγραμμα ροής φαίνεται στο σχήμα 3.2 εξετάζει αν το bit 3 της εισόδου είναι 1. Αν ναι, ανάβει τα LEDs της εξόδου, αλλιώς τα σβήνει.

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
0800	3A	START:	LDA 2000
0801	00		
0802	20		
0803	06		MVI B,08
0804	08		
0805	A0		ANA B
0806	CA		JZ OFF
0807	11		
0808	08		
0809	3E	ON:	MVI A,0
080A	00		
080B	32		STA 3000
080C	00		
080D	30		
080E	C3		JMP START
080F	00		
0810	08		
0811	3E	OFF:	MVI A,FF
0812	FF		
0813	32		STA 3000
0814	00		
0815	30		
0816	C3		JMP START
0817	00		
0818	08		



Σχήμα 3.2. Διάγραμμα ροής προγράμματος που εξετάζει ένα bit της εισόδου

2.2 Ολισθήσεις

ΕΦΑΡΜΟΓΗ 5: Χρησιμοποιώντας εντολές ολίσθησης

Οι ολισθήσεις είναι από τις συχνά χρησιμοποιούμενες λειτουργίες και υλοποιούνται στον 8085 μέσω των εντολών RRC, RLC, RAL και RAR. Στο παρακάτω παράδειγμα μπορούμε να παρατηρήσουμε την εφαρμογή μιας από αυτές τις εντολές και συγκεκριμένα της RRC στο δεδομένο 11111110. Έχει χρησιμοποιηθεί η RST 1 ώστε να προλαβαίνουμε να παρατηρούμε τα αποτελέσματα της ολίσθησης στην πόρτα 3000. Έτσι κάθε φορά που πατάμε το πλήκτρο RUN το δεδομένο ολισθαίνει κατά μία θέση δεξιά.

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
0800	3E		MVI A,FE
0801	FE		
0802	32	LOOP:	STA 3000
0803	00		
0804	30		
0805	CF		RST 1
0806	0F		RRC
0807	C3		JMP LOOP
0808	02		
0809	08		

2.3 Υπορουτίνες και στοίβα

Τα σύνθετα προγράμματα μπορούν να απλοποιηθούν με τη χρήση υπορουτινών που εκτελούν επαναληπτικές λειτουργίες. Η εντολή CALL χρησιμοποιείται για την κλήση μιας υπορουτίνας. Όταν εκτελείται, ο επεξεργαστής σώζει τα περιεχόμενα του μετρητή προγράμματος στη στοίβα και συνεχίζει με την εκτέλεση της υπορουτίνας. Η εντολή RET υποδηλώνει το τέλος μιας υπορουτίνας. Όταν ο επεξεργαστής συναντήσει μια εντολή RET επιστρέφει στο σημείο από το οποίο είχε κληθεί η υπορουτίνα που την περιέχει. Η χρήση της στοίβας επιτρέπει την ύπαρξη φωλιασμένων (nested) υπορουτινών.

Για την εύκολη προσπέλαση της στοίβας χρησιμοποιείται στον 8085 ένας ειδικός καταχωρητής των 16 bits, ο δείκτης στοίβας (stack pointer) που περιέχει κάθε φορά τη διεύθυνση της κορυφής της. Στο μLab ο stack pointer παίρνει την default τιμή 0BB0 όταν το συνδέουμε στην τροφοδοσία. Έτσι μπορούμε να χρησιμοποιούμε τη στοίβα χωρίς προηγουμένως να την ορίσουμε. Η στοίβα μπορεί να χρησιμοποιηθεί και για την αποθήκευση δεδομένων μέσω των εντολών PUSH και POP. Επειδή όμως αποτελείται από θέσεις μνήμης των 16 bits σε κάθε θέση της τοποθετείται ζεύγος δεδομένων. Έτσι η εντολή PUSH B σώζει τα περιεχόμενα των B, C στην κορυφή της στοίβας και η POP B τα επαναφέρει.

3. Τεχνικές προγραμματισμού

Η "κατασκευή" ενός σύνθετου προγράμματος είναι μια επίπονη διαδικασία που απαιτεί σωστό σχεδιασμό και μεθοδικότητα. Ανεξάρτητα πάντως από την υφή του συγκεκριμένου προβλήματος μπορούμε να χαράξουμε σε γενικές γραμμές την πορεία επίλυσής του. Τα βήματα που ακολουθούμε είναι τα εξής:

1) Ορισμός του προβλήματος.
2) Σχεδιασμός της λύσης, διαίρεση σε μικρότερα αυτόνομα τμήματα.
3) Διαγράμματα ροής του κυρίου προγράμματος και των υπορουτινών.
4) Γράψιμο των προγραμμάτων.
5) Έλεγχος και διόρθωση υπορουτινών και κυρίου προγράμματος.

Το πρώτο βήμα συνίσταται στην κατανόηση του προβλήματος και στον καθορισμό των παραμέτρων του όπως οι είσοδοί του, οι έξοδοί του, ο τρόπος επεξεργασίας των δεδομένων κλπ. Το δεύτερο βήμα είναι πολύ σημαντικό καθώς η σωστή επιλογή των τμημάτων στα οποία θα χωρίσουμε το πρόβλημα μπορεί να διευκολύνει αρκετά τη διαδικασία. Τα υπόλοιπα τρία βήματα γίνονται περισσότερο μηχανιστικά και έχουν μικρότερη βαρύτητα.

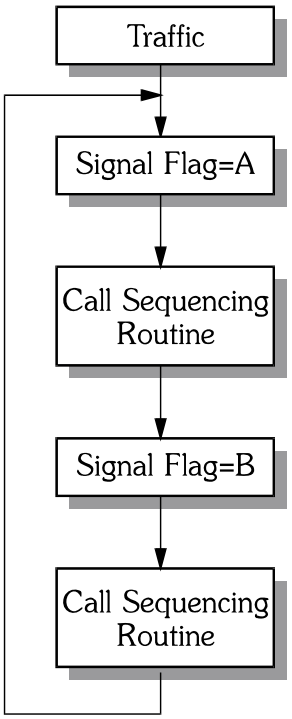
3. Τεχνικές προγραμματισμού

Στο τρίτο μέρος αυτής της εργαστηριακής άσκησης θα ακολουθηθούν τα παραπάνω βήματα προκειμένου να επιλυθεί το πρόβλημα ενός ελεγκτή σημάτων κυκλοφορίας. Συγκεκριμένα, υποθέτουμε την ύπαρξη διασταύρωσης δυο δρόμων Α και Β όπου λαμβάνει χώρα η ακόλουθη σειρά βημάτων:

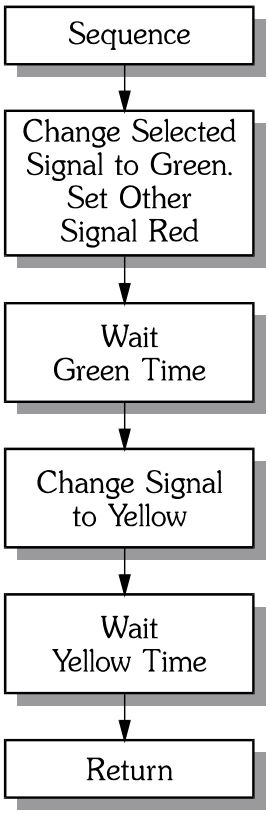
1. Σήμα Α = κόκκινο.
2. Σήμα Β = πράσινο.
3. Αναμονή για όλη τη διάρκεια του πράσινου σήματος.
4. Σήμα Β = κίτρινο.
5. Αναμονή για όλη τη διάρκεια του κίτρινου σήματος.
6. Σήμα Β = κόκκινο.
7. Σήμα Α = πράσινο.
8. Αναμονή για όλη τη διάρκεια του πράσινου σήματος.
9. Σήμα Α = κίτρινο.
10. Αναμονή για όλη τη διάρκεια του κίτρινου σήματος.
11. Πήγαινε στο βήμα 1.

Παρατηρούμε μια συμμετρία στο πρόβλημά μας αφού αποτελείται από δυο πανομοιότυπα κομμάτια, ένα για το δρόμο Α κι ένα για το δρόμο Β. Και τα δύο κομμάτια απαιτούν την αλλαγή του σήματος του δρόμου από πράσινο σε κίτρινο

και μετά σε κόκκινο. Είναι φανερή λοιπόν η ανάγκη μιας ρουτίνας που θα αναλαμβάνει τα παραπάνω και θα καλείται δυο φορές, μια για το δρόμο Α και μια για το δρόμο Β. Η ρουτίνα αυτή, που θα ονομάζεται στο εξής ρουτίνα SEQ, θα ειδοποιείται από το κύριο πρόγραμμα μέσω της τιμής κάποιου καταχωρητή για το ποιον δρόμο πρέπει κάθε φορά να ελέγξει. Το διάγραμμα ροής του κυρίου προγράμματος φαίνεται στο σχήμα 3.3 και της SEQ στο 3.4.



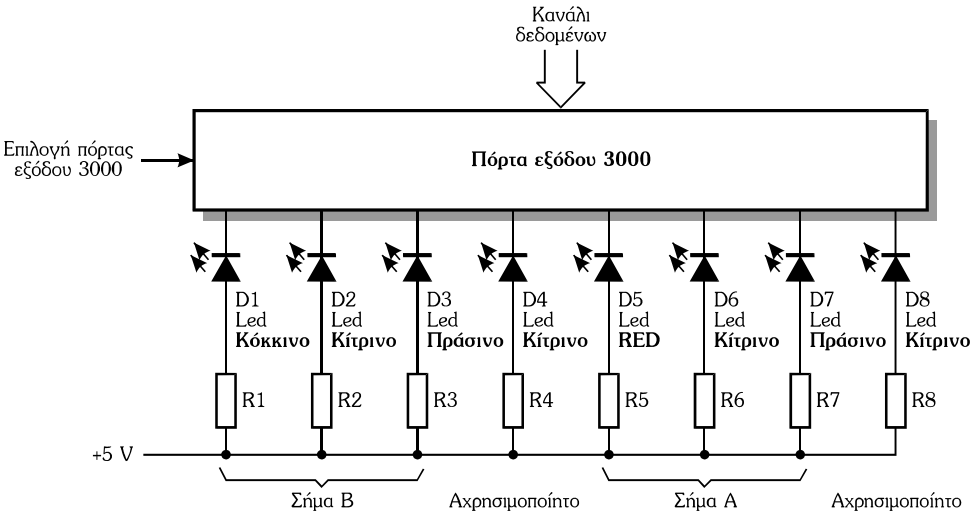
Σχήμα 3.3. Διάγραμμα ροής κυρίου προγράμματος



Σχήμα 3.4. Διάγραμμα ροής SEQ

Θα πρέπει τώρα να αναλύσουμε περισσότερο τη SEQ λαμβάνοντας υπόψη τη μορφή της εξόδου που θα χρησιμοποιηθεί. Στο μLab μπορούμε να χρησιμοποιήσουμε την πόρτα εξόδου 3000 στην οποία είναι συνδεδεμένα 8 LEDs τριών διαφορετικών χρωμάτων (πράσινο, κίτρινο, κόκκινο, ό,τι ακριβώς χρειαζόμαστε). Έτσι καθορίζουμε να χρησιμοποιηθούν τα LEDs 1-3 για το δρόμο Α, τα LEDs 5-7 για το δρόμο Β ενώ τα 0 και 4 θα μείνουν προσωρινά αχρησιμοποίητα (αργότερα θα χρησιμοποιηθούν και αυτά σε κάποια τροποποίηση

του προγράμματος του ελεγκτή σημάτων κυκλοφορίας). Όλα αυτά φαίνονται παραστατικά στο σχήμα 3.5.



Σχήμα 3.5. Χρήση της πόρτας εξόδου από το πρόγραμμα ελεγκτή σημάτων κυκλοφορίας

Με βάση το σχήμα αυτό και υπενθυμίζοντας ότι στην έξοδο του μLab χρησιμοποιείται αρνητική λογική (δηλαδή 1 = σβηστό, 0 = αναμμένο) συγκεντρώνω στον παρακάτω πίνακα τις απαιτούμενες λέξεις εξόδου για κάθε μια από τις περιπτώσεις του προβλήματος.

Πίνακας 3.1. Οι λέξεις εξόδου που αντιστοιχούν σε κάθε περίπτωση του προβλήματος

αριθμός bit :	Σήμα B 7 6 5 4	Σήμα A 3 2 1 0	Hex
A πράσινο, B κόκκινο	0 1 1 1	1 1 0 1	7D
A κίτρινο, B κόκκινο	0 1 1 1	1 0 1 1	7B
A κόκκινο, B πράσινο	1 1 0 1	0 1 1 1	D7
A κόκκινο, B κίτρινο	1 0 1 1	0 1 1 1	B7

Χρειαζόμαστε μετά από αυτά μια υπορουτίνα εξόδου που θα αναλάβει να διεκπεραιώνει την αποστολή στην πόρτα 3000 των καταλλήλων λέξεων εξόδου για κάθε περίπτωση. Η υπορουτίνα αυτή, που θα ονομάζεται στο εξής CHNG, θα καλείται από την SEQ. Μεταξύ των καθηκόντων της θα είναι και η λήψη του μηνύματος του κυρίου προγράμματος σχετικά με το ποιος δρόμος πρέπει να

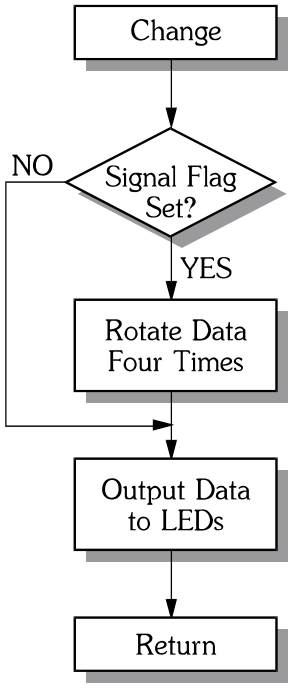
ελεγχθεί. Αυτό, όπως έχει ήδη ειπωθεί, γίνεται μέσω ενός καταχωρητή και συγκεκριμένα μέσω του Ε.

Βλέπουμε ότι το πρόβλημά μας απαιτεί να στέλνονται διαδοχικά 4 διαφορετικές λέξεις στην έξοδο, όπως δείχνει ο πίνακας 3.1. Η CHNG στέλνει μια λέξη κάθε φορά που καλείται. Όμως καλείται 2 φορές από την SEQ και η SEQ με τη σειρά της καλείται 2 φορές από το κύριο πρόγραμμα, οπότε έχουμε τελικά αποστολή 4 λέξεων. Η διαφοροποίηση των λέξεων αυτών εξασφαλίζεται από τη δομή της ρουτίνας CHNG. Συγκεκριμένα, η ρουτίνα αυτή εξετάζει αρχικά το περιεχόμενο του Ε. Αν αυτό είναι 0 στέλνει το περιεχόμενο του καταχωρητή Η στην έξοδο όπως είναι. Διαφορετικά ανταλλάσσει πρώτα τα 4 LSB με τα 4 MSB του Η. Αυτό στηρίζεται στην παρατήρηση ότι οι λέξεις εξόδου που κρατούν το σήμα του δρόμου Β κόκκινο προκύπτουν από τις αντίστοιχες λέξεις εξόδου για τον Α αν αλλάξουμε τα 4 LSB με τα 4 MSB. Έτσι τελικά όταν Ε=0 ελέγχεται ο δρόμος Α και όταν Ε=1 ο δρόμος Β. Το διάγραμμα ροής της CHNG φαίνεται στο σχήμα 3.6.

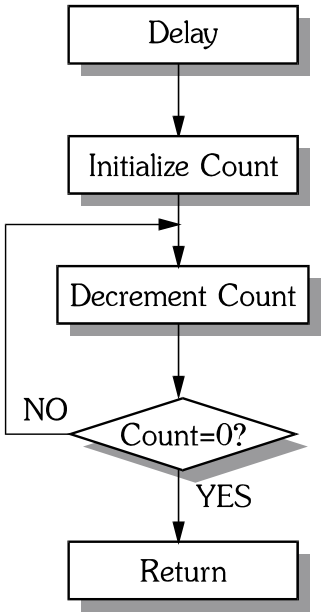
Μένει τώρα να αναφερθούμε στη ρουτίνα καθυστέρησης που πρέπει να χρησιμοποιήσουμε ώστε το πράσινο και το κίτρινο σήμα να διαρκούν κάποιο καθορισμένο χρόνο διαφορετικό βέβαια για το καθένα. Μια συνηθισμένη προγραμματιστικά ρουτίνα καθυστέρησης είναι η ακόλουθη (το διάγραμμα ροής της υπάρχει στο σχήμα 3.7):

Ετικέτα	**Εντολή**	**Σχόλια**
DELAY:	DCR A	; 4 καταστάσεις
	JNZ DELAY	; 7/10 καταστάσεις
	RET	; 10 καταστάσεις

Η ρουτίνα αυτή απαιτεί 14 καταστάσεις για κάθε επανάληψη εκτός της τελευταίας που απαιτεί 21. Η συνολική καθυστέρηση σε καταστάσεις είναι τότε: DELAY = 14·(A-1) + 21 όπου Α το περιεχόμενο του συσσωρευτή. Το μLab χρησιμοποιεί ρολόι 2 MHz, οπότε DELAY = 7·(A-1) + 10,5 μsec. Η μέγιστη καθυστέρηση που μπορεί να δώσει αυτή η ρουτίνα είναι για A=0 οπότε DELAY = 7·(256-1) + 10,5 = 1795 μsec που είναι βέβαια πολύ μικρή για τις ανάγκες του προβλήματος. Η επόμενη ρουτίνα χρησιμοποιώντας ποσότητες των 16 bits αυξάνει τη μέγιστη καθυστέρηση αλλά όχι αρκετά.



Σχήμα 3.6. Διάγραμμα ροής CHNG



Σχήμα 3.7. Διάγραμμα ροής ρουτίνας καθυστέρησης

Ετικέτα	Εντολή	Σχόλια
DELAY:	DCX B	; 6 καταστάσεις
	MOV A,B	; 4 καταστάσεις
	ORA C	; 4 καταστάσεις
	JNZ DELAY	; 7/10 καταστάσεις
	RET	; 10 καταστάσεις

Εδώ DELAY = 24·(N-1) + 31 καταστάσεις = 12·(N-1) + 15,5 μsec όπου N η 16-bit ποσότητα που βρίσκεται στους B και C. Για N=0 έχω max DELAY = 12·(65536-1) + 15,5 = 0,786 sec. Η επιθυμητή όμως καθυστέρηση στο πρόβλημά μας είναι της τάξης κάποιων δευτερολέπτων. Για να την επιτύχουμε θα πρέπει να χρησιμοποιήσουμε τη δεύτερη ρουτίνα σε συνδυασμό με έναν ακόμα καταχωρητή ώστε να μπορεί να επαναληφθεί η καθυστέρηση των 0,786 sec μέχρι 256 φορές.

Γράφουμε τώρα το πλήρες πρόγραμμα για τον ελεγκτή σημάτων κυκλοφορίας σύμφωνα με την ως τώρα ανάλυση.

Κύριο πρόγραμμα

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
0810	1E	TRAF:	MVI E,0
0811	00		
0812	CD		CALL SEQ
0813	30		
0814	08		
0815	00		NOP
0816	00		NOP
0817	00		NOP
0818	00		NOP
O819	1E		MVI E,1
081A	01		
081B	CD		CALL SEQ
081C	30		
081D	08		
081E	C3		JMP TRAF
081F	10		
0820	08		

Υπορουτίνα SEQ

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
0830	26	SEQ:	MVI H,7D
0831	7D		
0832	CD		CALL CHNG
0833	55		
0834	08		
0835	16		MVI D,6
0836	06		
0837	CD		CALL DELAY
0838	70		
0839	08		
083A	26		MVI H,7B
083B	7B		
083C	CD		CALL CHNG
083D	55		
083E	08		
083F	16		MVI D,2
0840	02		
0841	CD		CALL DELAY
0842	70		

0843	08		
0844	C9		RET

Υπορουτίνα CHNG

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
0855	7B	CHNG:	MOV A,E
0856	FE		CPI 0
0857	00		
0858	7C		MOV A,H
0859	CA		JZ OUTP
085A	60		
085B	08		
085C	07		RLC
085D	07		RLC
085E	07		RLC
085F	07		RLC
0860	32	OUTP:	STA 3000
0861	00		
0862	30		
0863	C9		RET

Υπορουτίνα DELAY

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή
0870	01	DELAY:	LXI B,0
0871	00		
0872	00		
0873	0B	LOOP:	DCX B
0874	78		MOV A,B
0875	B1		ORA C
0876	C2		JNZ LOOP
0877	73		
0878	08		
0879	15		DCR D
087A	C2		JNZ DELAY
087B	70		
087C	08		
087D	C9		RET

Στο σημείο αυτό μένει μόνο ο έλεγχος ορθής λειτουργίας των τμημάτων που αποτελούν το πρόγραμμα. Ο έλεγχος αυτός ξεκινάει από τις ρουτίνες που δεν καλούν άλλες και συνεχίζεται με αυτές που καλούν μόνο ήδη ελεγμένες, μέχρι να φτάσουμε στο κύριο πρόγραμμα. Σημαντικός όπως είδαμε είναι ο ρόλος των διακοπών στη διαδικασία αυτή.

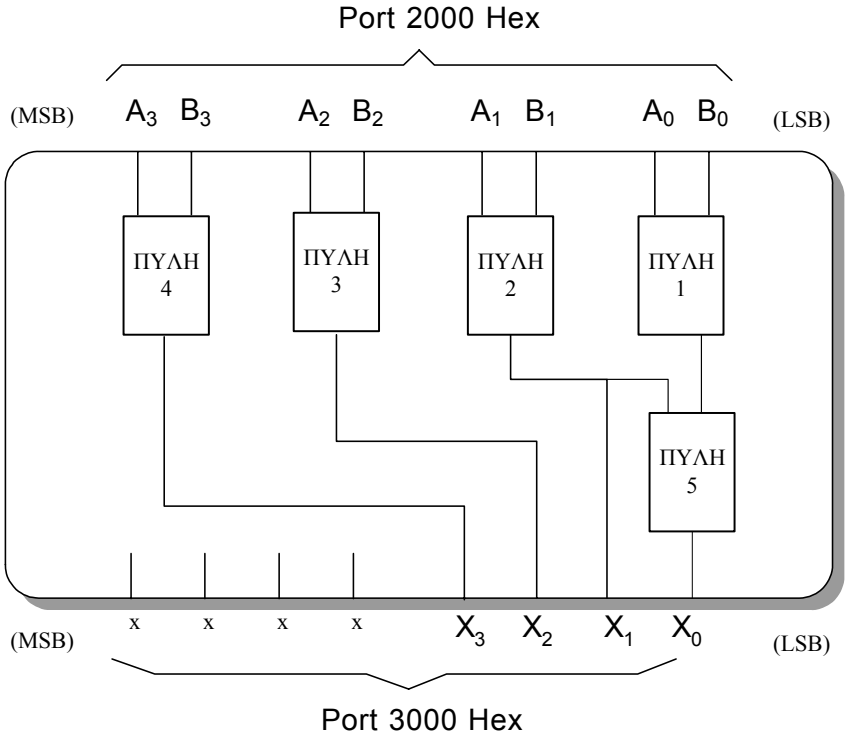
Προτεινόμενες βελτιώσεις - τροποποιήσεις για τον ελεγκτή σημάτων κυκλοφορίας.

1. Υποθέτουμε ότι οι δρόμοι Α και Β έχουν διαφορετική κίνηση. Συνεπώς, και η διάρκεια του πράσινου σήματος πρέπει να είναι διαφορετική για κάθε δρόμο.
2. Να ορισθεί διαφορετική διάρκεια κίτρινου σήματος για κάθε δρόμο.
3. Οι σηματοδότες των δύο δρόμων να γίνονται και οι δύο κόκκινοι για λίγα δευτερόλεπτα, δηλαδή η αλλαγή ενός σήματος σε πράσινο να γίνεται λίγα δευτερόλεπτα μετά την αλλαγή του άλλου σε κόκκινο, αντίθετα με το πρόγραμμα που παρουσιάστηκε στην παράγραφο 3.
4. Τα δύο αχρησιμοποίητα LEDs της εξόδου (LEDs 0,4) να χρησιμοποιηθούν σαν σηματοδότες στροφής για τους δυο δρόμους.
5. Να χρησιμοποιηθεί ένας από τους διακόπτες της πόρτας εισόδου 2000 σαν αισθητήρας ανίχνευσης αυτοκινήτων τοποθετημένος στο μικρότερο δρόμο, ο οποίος όταν ενεργοποιείται και μόνο τότε, να κάνει το σηματοδότη του μεγάλου δρόμου κόκκινο και του μικρού πράσινο αφού το πράσινο του μεγάλου δρόμου συμπληρώσει τον χρόνο του.

4. Τα θέματα της 3ης εργαστηριακής άσκησης

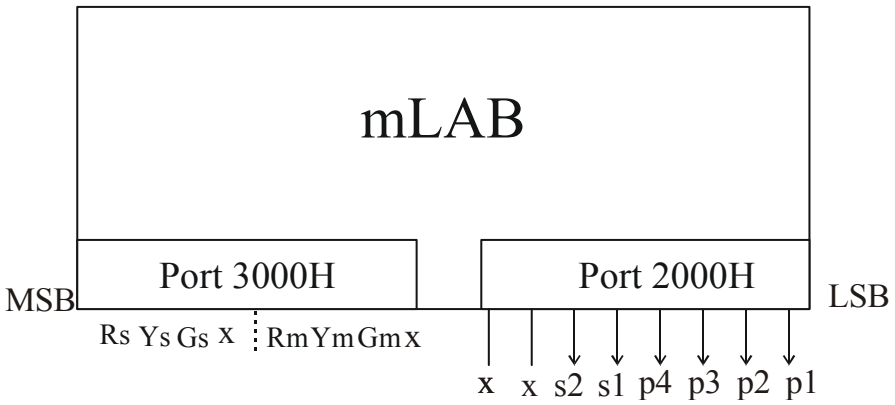
i. Να εξομοιωθεί η λειτουργία ενός υποθετικού I.C. με πύλες όπως φαίνεται στο σχήμα 8. Τα bits εισόδου πρέπει να αντιστοιχούν ακριβώς όπως φαίνονται στο σχήμα 8 με τα dip switches της πόρτας εισόδου 2000 Hex, και τα LEDs πρέπει να είναι τα τέσσερα LSB της πόρτας εξόδου 3000 Hex.

Οι πύλες 1, 2, 3, 4, 5 να υλοποιούν τις AND, AND, NOR, NXOR και OR αντίστοιχα.



Σχήμα 8. Το υποθετικό IC του θέματος (i)

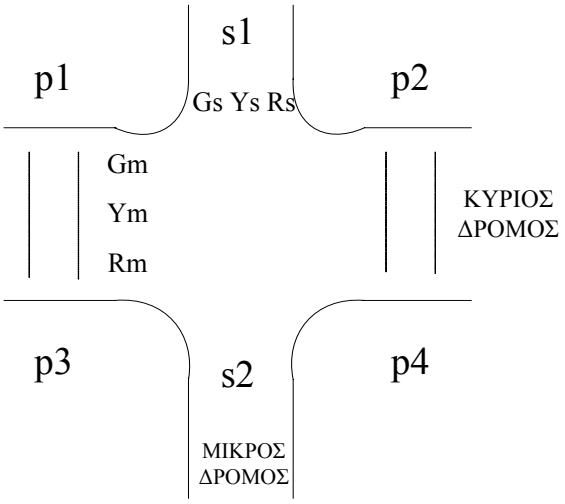
ii. Σχεδιάστε έναν ελεγκτή σημάτων κυκλοφορίας σε διασταύρωση ενός κύριου δρόμου με ένα μικρότερο. Υπάρχουν τέσσερα pushbuttons (p1, p2, p3 και p4) για τους πεζούς και δύο αισθητήρες (s1 και s2) αυτοκινήτων στο μικρότερο δρόμο. Δεδομένου ότι πολύ λίγα αυτοκίνητα διέρχονται από το μικρό δρόμο, το σύστημα πρέπει να σχεδιαστεί έτσι ώστε το πράσινο σήμα του κύριου δρόμου Gm να είναι συνεχώς αναμμένο και το πράσινο σήμα του μικρότερου δρόμου Gs να ενεργοποιείται όταν πατηθεί ένας από τους διακόπτες (p1, p2, p3, p4, s1, s2) αφού συμπληρωθεί ο χρόνος του πράσινου στο μεγάλο δρόμο (ελάχιστη διάρκεια πράσινου 8 sec). Να ληφθεί υπόψη ότι οι διακόπτες μπορούν να ξαναπατηθούν μόνο αφού επανέλθουν. Το κίτρινο σήμα πρέπει να διαρκεί 2 sec ενώ το πράσινο του μεγάλου δρόμου τουλάχιστον 8 sec. Ο ελεγκτής να υλοποιηθεί με βάση το μLab όπως φαίνεται στο επόμενο σχήμα και το σχήμα 9. Οι χρόνοι που ισχύουν για το μικρό δρόμο είναι 1 sec για κίτρινο και 4 sec για πράσινο.



Επίσης σε αυτόν τον ελεγκτή αν προκληθεί διακοπή (πάτημα του πλήκτρου INTR) να αναβοσβήνουν τα κίτρινα των 2 δρόμων με συχνότητα ½ sec (¼ sec για ON και ¼ sec για OFF). Με μια δεύτερη διακοπή να επανέρχεται η προηγούμενη λειτουργία. Σχεδιάστε με τον καλύτερο δυνατό τρόπο την εξυπηρέτηση διακοπής. Interrupt μέσα σε interrupt, όπως έχει αναφερθεί, θα πρέπει να αποφεύγεται. Δοκιμάστε λύσεις με τη χρήση κάποιου flag, και εξυπηρέτηση στο κυρίως πρόγραμμα. Θα πρέπει να δώσετε ιδιαίτερη σημασία στα παρακάτω σημεία:

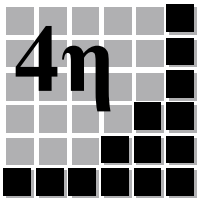
- Ο μεγάλος δρόμος θα πρέπει να έχει πράσινο για τουλάχιστον 8 sec. Αν ενδιάμεσα πατηθεί, για μικρό χρονικό διάστημα, και επανέλθει κάποιος διακόπτης, θα πρέπει το πρόγραμμά σας να το λαμβάνει υπ' όψη, πιθανά σε κάποιο flag, και **αφού** ολοκληρωθεί ο ελάχιστος χρόνος των 8 sec, τότε να αλλάζει κατάσταση. Είναι προφανές λοιπόν ότι χρειάζεται να υλοποιηθεί μια χρονοκαθυστέρηση που ανά μικρά χρονικά διαστήματα θα ελέγχει τα push-buttons και τους αισθητήρες.

- Επίσης προσέξτε να χειριστείτε διαφορετικά τους πιεστικούς διακόπτες (p1, p2, p3, p4) και διαφορετικά τους αισθητήρες προσέγγισης (s1, s2) των αυτοκινήτων. Οι διακόπτες μπορούν να ξαναπατηθούν **μόνο αφού επανέλθουν**. Αν δηλαδή ένας διακόπτης παραμείνει μόνιμα πιεσμένος, το πρόγραμμά σας θα πρέπει να τον εξυπηρετήσει μόνο μια φορά και να τον αγνοήσει στην συνέχεια. Θα πρέπει λοιπόν να ελέγχετε για την θετική ακμή στα pushbuttons, κάτι που φυσικά δεν χρειάζεται να κάνετε για τους αισθητήρες s1 και s2.



Σχήμα 9. Χρήση του μLab και μορφή του δρόμου του θέματος (ii)

iii. Να εξομοιωθεί ένας αυτοματισμός βαγονέτου που κινείται από δεξιά προς τα αριστερά και αντίστροφα. Η κίνηση του βαγονέτου θα φαίνεται πάνω στα led και η αλλαγή από led σε led θα γίνεται κάθε 0,5 sec. Το βαγονέτο θα ξεκινάει από την μία άκρη και θα καταλήγει στην άλλη όπου θα σταματά. Η κίνηση του βαγονέτου θα ελέγχεται από τον LSB διακόπτη της πόρτας 2000 Hex. Όταν αυτό είναι ON το βαγονέτο κινείται, όταν είναι OFF σταματά. Αν προκληθεί διακοπή θα πρέπει να αναστρέφεται η κατεύθυνση της κίνησης. Σε περίπτωση που το βαγονέτο έχει σταματήσει στην άκρη, συνεχίζει μόνο με πάτημα του INTR εφόσον βέβαια ο διακόπτης δεν είναι στο OFF, ενώ διπλό πάτημα του INTR κατά την στάση, δεν θα πρέπει να μεταβάλλει την κατεύθυνση κίνησης.



4η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

- 1. Έλεγχος των περιφερειακών μέσω software
- 2. Έλεγχος του πληκτρολογίου με τις ρουτίνες του μLab
  - Εφαρμογή 1: Χρησιμοποιώντας τη ρουτίνα ανάγνωσης από το πληκτρολόγιο
- 3. Απευθείας έλεγχος του πληκτρολογίου
  - Εφαρμογή 2: Εξετάζοντας το πληκτρολόγιο
- 4. Έλεγχος της οθόνης 7 τμημάτων (7-segment displays) με τις ρουτίνες του μLab
  - Εφαρμογή 3: Απεικόνιση μηνύματος
- 5. Απευθείας έλεγχος των οθονών 7 τμημάτων
  - Εφαρμογή 4: Ελέγχοντας την οθόνη
  - Εφαρμογή 5: Χρησιμοποιώντας την οθόνη

1. Έλεγχος των περιφερειακών μέσω software

Οι παρακάτω σημειώσεις περιγράφουν το software που ελέγχει το πληκτρολόγιο και την οθόνη του μLab. Περιγράφονται προγράμματα ανάγνωσης από το πληκτρολόγιο και απεικόνισης στην οθόνη. Τα σχήματα και οι μέθοδοι που περιγράφονται, έχουν εφαρμογή σε ένα ευρύ φάσμα συστημάτων που στηρίζονται σε μικροεπεξεργαστές, καθώς τα περισσότερα συστήματα περιλαμβάνουν πληκτρολόγιο και οθόνη που λειτουργούν με τον ίδιο τρόπο.

2. Έλεγχος του πληκτρολογίου με τις ρουτίνες του μLab

Το πληκτρολόγιο μπορεί να θεωρηθεί σαν ένας πίνακας πλήκτρων, κάθε ένα από τα οποία εξετάζεται αν έχει πατηθεί, ξεχωριστά. Όπως θα δούμε παρακάτω, το πλήκτρο που πατήθηκε αναγνωρίζεται από τα δεδομένα κάποιας θύρας εισόδου. Τα δεδομένα αυτά αφού διαβαστούν από τη θύρα, μετατρέπονται στον κωδικό του πλήκτρου που πατήθηκε μέσω μιας ρουτίνας που λέγεται KIND (Key INput and Decode). Η ρουτίνα αυτή είναι αποθηκευμένη στη ROM του μLab. Λεπτομέρειες της ρουτίνας αυτής, όπως και για πολλές άλλες, σας παραπέμπουμε στο αντίστοιχο παράρτημα των σημειώσεων. Η χρήση της είναι απλούστατη: εσείς καλείτε τη ρουτίνα και αυτή όταν κάποιο πατηθεί πλήκτρο επιστρέφει στο συσσωρευτή ,A, τον κωδικό του πλήκτρου που πατήθηκε (για τους κωδικούς όλων των πλήκτρων βλέπε πίνακα 4.1). Η KIND, λοιπόν, μπορεί να χρησιμοποιηθεί στα προγράμματά σας για ανάγνωση του πληκτρολογίου.

Πίνακας 4.1. Κωδικοί των πλήκτρων για τη ρουτίνα KIND

Πλήκτρο	Κωδικός	Πλήκτρο	Κωδικός
0	00	C	0C
1	01	D	0D
2	02	E	0E
3	03	F	0F
4	04	FETCH REG	80
5	05	DECR	81
6	06	FETCH ADRS	82
7	07	STORE/INCR	83
8	08	RUN	84
9	09	FETCH PC	85
A	0A	INSTR STEP	86
B	0B	HDWR STEP	F7

Εφαρμογή 1: Χρησιμοποιώντας τη ρουτίνα ανάγνωσης από το πληκτρολόγιο

Διαδικασία

1. Ελέγξτε και πληκτρολογήστε το πρόγραμμα του πίνακα 4.2. Το πρόγραμμα αυτό εικονίζει μια απλή εφαρμογή του πληκτρολογίου. Χρησιμοποιούνται δύο υπορουτίνες της ROM: η KIND (διεύθυνση 014B) και η BEEP (διεύθυνση 0010).

Πίνακας 4.2. Πρόγραμμα που διαβάζει το πληκτρολόγιο και παράγει ήχο όταν πατηθεί "7"

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0800	CD	READ:	CALL KIND	; Διάβασε
0801	4B			; πλήκτρο
0802	01			
0803	FE		CPI 07	
0804	07			
0805	C2		JNZ READ	
0806	00			
0807	08			
0808	CD		CALL BEEP	; Αν είναι 7,
0809	10			; beep
080A	00			
080B	C3		JMP READ	
080C	00			
080D	08			

Το πληκτρολόγιο διαβάζεται με την ρουτίνα KIND, η οποία περιμένει ώσπου να πατηθεί ένα πλήκτρο και τότε επιστρέφει, αφήνοντας τον κωδικό του πλήκτρου στον συσσωρευτή. Η εντολή CPI 07 δίνει στη zero flag την τιμή 1, αν το περιεχόμενο του συσσωρευτή είναι ίσο με 7. Η εντολή JNZ READ γυρίζει το πρόγραμμα πίσω στην αρχή, αν η zero flag έχει τιμή μηδέν. Αλλιώς, η εντολή JNZ δεν έχει κανένα αποτέλεσμα και τότε καλείται η ρουτίνα BEEP. Η παραπάνω διαδικασία επαναλαμβάνεται. Ήχος παράγεται κάθε φορά που πιέζεται το "7".

2. Ελέγξτε ότι το πρόγραμμα είναι σωστά αποθηκευμένο στο μLab.
3. Τρέξτε το πρόγραμμα. Σημειώστε ότι όταν πατηθεί το πλήκτρο RUN, φαίνεται σαν να μην συμβαίνει τίποτε. Όμως το πρόγραμμα τρέχει, αλλά όσο η KIND

εξετάζει το πληκτρολόγιο, η οθόνη παραμένει αναμμένη κρατώντας το προηγούμενο μήνυμα. Δεν ελέγχει όμως το monitor πρόγραμμα την οθόνη κατά την εκτέλεση του παραπάνω προγράμματος.

4. Πατήστε το πλήκτρο "7". Θα παραχθεί ήχος. Τώρα πατήστε οποιοδήποτε άλλο πλήκτρο. Μόνο το πλήκτρο "7" προκαλεί ήχο.
5. Πατήστε το πλήκτρο "RESET" για να επιστρέψει ο έλεγχος στο monitor. Τροποποιήστε το πρόγραμμα ώστε να ψάχνει αν πατήθηκε άλλο πλήκτρο (βλ. και πίνακα 1).
6. Ελέγξτε το τροποποιημένο πρόγραμμα.
7. Πιέστε "RESET" για να επιστρέψει ο έλεγχος στο monitor.

3. Απευθείας έλεγχος του πληκτρολογίου

Μπορεί μεν η ρουτίνα KIND να κάνει εύκολη τη χρήση του πληκτρολογίου, ωστόσο δεν μας δίνει τη δυνατότητα να δούμε ποια είναι ακριβώς η διαδικασία ανάγνωσης. Για να εξηγήσουμε την τεχνική που χρησιμοποιείται για την ανάγνωση από το πληκτρολόγιο, θα περιγράψουμε ένα πρόγραμμα που "διαβάζει" το πληκτρολόγιο χωρίς να χρησιμοποιεί υπορουτίνες της ROM.

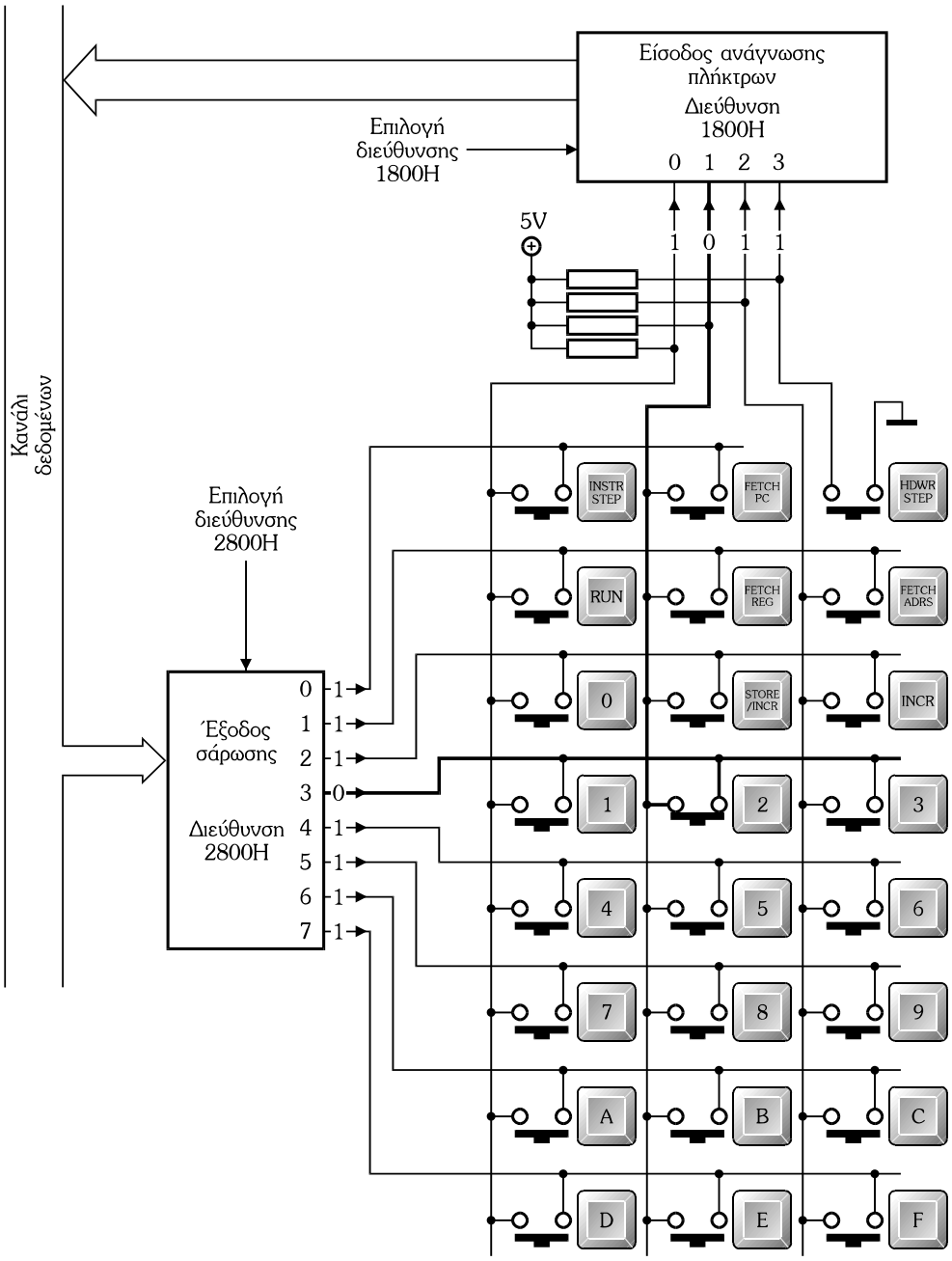
Το σχήμα 1 δείχνει ένα διάγραμμα του interface του πληκτρολογίου.

Όπως έχει περιγραφεί και προηγουμένως, η σάρωση (scanning) του πληκτρολογίου γίνεται ως εξής: μια γραμμή κάθε φορά. Για λόγους απλότητας, θεωρήστε ότι εξετάζεται μια μόνο γραμμή πλήκτρων, έστω αυτή που περιέχει τα "1", "2", "3". Η διαδικασία διαβάσματος γίνεται σε 2 βήματα:

Βήμα 1ο: Γράψε τα κατάλληλα δεδομένα στην πόρτα σάρωσης (scan port) ώστε να επιλεγεί η επιθυμητή γραμμή.

Βήμα 2ο: Διάβασε τα δεδομένα των στηλών (στις οποίες αντιστοιχούν τα πλήκτρα της γραμμής που επιλέχθηκε στο προηγούμενο βήμα) από την πόρτα ανάγνωσης των πλήκτρων (key read port).

Κάθε bit της πόρτας σάρωσης τίθεται στην τιμή "λογικού 1" εκτός από το bit που αντιστοιχεί στη γραμμή που θέλουμε να διαβαστεί, το οποίο τίθεται στην τιμή "λογικού 0". Έτσι, για να εξεταστούν τα πλήκτρα 1, 2 και 3 γράφεται στην πόρτα σάρωσης το δεδομένο 11110111 (F7 hex) (βλέπε και το σχήμα 4.1).



Σχήμα 4.1. Διάγραμμα interface του πληκτρολογίου

Ύστερα διαβάζεται η πόρτα ανάγνωσης των πλήκτρων, για να πάρουμε πληροφορίες για τις στήλες. Αν δεν πατήθηκε κανένα πλήκτρο, τα bits 0-3 έχουν όλα τιμή "λογικό 1". Αν πατήθηκε το "1", το bit 0 έχει τιμή "λογικό 0". Αν πατήθηκε το "2", το bit 1 έχει τιμή "λογικό 0" κι αν πατήθηκε το "3", τότε το bit 2 έχει τιμή "λογικό 0" (βλέπε και πίνακα 4.3).

Πίνακας 4.3. Τα δεδομένα στην πόρτα ανάγνωσης πλήκτρων.

κανένα πλήκτρο δεν πατήθηκε	XXXX X111
πατήθηκε το "1"	XXXX X110
πατήθηκε το "2"	XXXX X101
πατήθηκε το "3"	XXXX X011

Τα X στα 5 MSB δείχνουν ότι αυτά τα bits περιέχουν απροσδιόριστες τιμές. Τα δεδομένα που μας ενδιαφέρουν περιέχονται στα 3 LSB.

Προσέξτε ότι τα δεδομένα που διαβάζονται από τη θύρα ανάγνωσης πλήκτρων αποτελούν κώδικα που προσδιορίζει το πλήκτρο. Αν π.χ. πατηθεί το "2", ο συσσωρευτής (αφού διαβαστεί η θύρα ανάγνωσης πλήκτρων) θα περιέχει την τιμή 05 hex (υποθέτοντας ότι τα πέντε MSB είναι 0). Η KIND που χρησιμοποιήθηκε στο προηγούμενο πείραμα εξετάζει όλες τις γραμμές και μετατρέπει τους απλούς αυτούς κωδικούς στους κωδικούς που είδαμε στον πίνακα 4.1.

Η επόμενη εφαρμογή δείχνει ένα τμήμα προγράμματος που εκτελεί τη διαδικασία ανάγνωσης που μόλις περιγράψαμε. Η πόρτα σάρωσης έχει τεθεί να επιλέγει την επιθυμητή γραμμή και τα δεδομένα της στήλης διαβάζονται στο συσσωρευτή.

Πίνακας 4.4. Πρόγραμμα που διαβάζει μια γραμμή του πληκτρολογίου.

```
MVI A,F7      ; πόρτα σάρωσης:=1111 0111 - επιλογή γραμμής
STA 2800
LDA 1800      ; διάβασε τις στήλες των πλήκτρων
```

Εφαρμογή 2: Εξετάζοντας το πληκτρολόγιο

Διαδικασία

1. Πληκτρολογήστε το πρόγραμμα του πίνακα 4.5. Το πρόγραμμα εξετάζει μια γραμμή πλήκτρων. Αν έχει πατηθεί το "2", τότε καλείται η ρουτίνα BEEP.

Πίνακας 4.5. Πρόγραμμα ελέγχου του πλήκτρο "2"

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0800	3E		MVI A,F7	; Πόρτα σάρωσης
0801	F7			; 1111 0111
0802	32		STA 2800	
0803	00			
0804	28			
0805	3A	READ:	LDA 1800	; Διάβασε τις
0806	00			; στήλες
0807	18			
0808	06		MVI B,07	; Μηδένισε τα 5
0809	07			; MSB
080A	A0		ANA B	
080B	FE		CPI 05	; Πατήθηκε το
080C	05			; "2";
080D	C2		JNZ READ	; Αν όχι,
080E	05			; ξαναδιάβασε
080F	08			
0810	CD		CALL BEEP	
0811	10			
0812	00			
0813	C3		JMP READ	
0814	05			
0815	08			

2. Ελέγξτε ότι το πρόγραμμα είναι σωστά αποθηκευμένο στο μLab.
3. Τρέξτε το πρόγραμμα, δοκιμάζοντας διαφορετικά πλήκτρα. Δουλεύει όπως αναμένατε;
4. Πατήστε "RESET" και στη συνέχεια τροποποιήστε το πρόγραμμα, ώστε να ελέγχεται το πάτημα του πλήκτρου "3", αντί του "2" (βλ. και πίνακα 3). Τρέξτε το νέο πρόγραμμα και ελέγξτε τη λειτουργία του.

4. Έλεγχος της οθόνης 7 τμημάτων με τις ρουτίνες του μLab

Το μLab χρησιμοποιεί μια εξαψήφια 7-τμημάτων LED οθόνη. Κάθε στιγμή ανάβει μόνο ένα ψηφίο και παραμένει αναμμένο κατά το ένα έκτο του χρόνου. Έτσι σταδιακά ανάβουν όλα τα ψηφία. Η εναλλαγή αυτή, ωστόσο, γίνεται τόσο γρήγορα, ώστε σε εμάς όλα τα ψηφία φαίνονται αναμμένα ταυτόχρονα. Όλες οι οθόνες πολλών ψηφίων λειτουργούν συνήθως κατ' αυτό τον τρόπο - άλλωστε έτσι απλοποιείται και το hardware.

Το monitor πρόγραμμα του μLab περιέχει μια ρουτίνα που λέγεται DCD (Display Character Decoder), η οποία ελέγχει την οθόνη. Για να χρησιμοποιηθεί το πρόγραμμα αυτό, τα ψηφία που θα απεικονιστούν αποθηκεύονται σε έξι θέσεις μνήμης (μία για κάθε ψηφίο). Το πρόγραμμα διαβάζει τα ψηφία από τη μνήμη, μετατρέπει τα δεδομένα στον κωδικό για απευθείας έξοδο την οθόνη και έπειτα την "φρεσκάρει".

Υπάρχει ένα ακόμα πρόγραμμα που βοηθά στην εμφάνιση χαρακτήρων στην οθόνη. Πρόκειται για μια ρουτίνα που λέγεται STDM (Store Display Message), η οποία απλώς μετακινεί το μήνυμα (τους χαρακτήρες που θα απεικονιστούν) από τις θέσεις μνήμης στις οποίες τους έχουμε αποθηκεύσει στις θέσεις μνήμης στις οποίες η ρουτίνα απεικόνισης περιμένει να τους βρει. Με τις δύο αυτές ρουτίνες μπορεί κανείς εύκολα να χειριστεί την οθόνη.

Τέλος πρέπει να σημειωθεί ότι η παράλληλη χρήση της ρουτίνας KIND, όσο δεν πατάμε κάποιο πλήκτρο, δεν παρεμποδίζει το φρεσκάρισμα της οθόνης γιατί όπου υπάρχει αναμονή (στην KIND) καλείται συνεχώς η DCD (βλ. παράρτημα 3)

Εφαρμογή 3: Απεικόνιση μηνύματος

Διαδικασία

1. Κωδικοποιήστε και πληκτρολογήστε το πρόγραμμα του πίνακα 4.6. Η ρουτίνα STDM ξεκινά από τη διεύθυνση 0018 και η DCD από τη 01E9.

Πίνακας 4.6. Πρόγραμμα απεικόνισης μηνύματος

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0800	11		LXI D,0810	; Θέτει τη
0801	10			; διεύθυνση
0802	08			; του μηνύματος
0803	CD		CALL STDM	
0804	18			
0805	00			
0806	CD	LOOP:	CALL DCD	
0807	E9			
0808	01			
0809	C3		JMP LOOP	
080A	06			
080B	08			

Το παραπάνω πρόγραμμα αποθηκεύει αρχικά στο ζεύγος καταχωρητών DE τη διεύθυνση στην οποία αρχίζει το μήνυμα. Στον Ε αποθηκεύεται το 1ο byte της διεύθυνσης, ενώ στον D το 2ο. Στη συνέχεια καλείται η STDM, η οποία μετακινεί το μήνυμα που ξεκινά στην παραπάνω διεύθυνση στη θέση στην οποία περιμένει να το βρει η ρουτίνα DCD. Τέλος, η DCD εκτελείται συνεχώς, "φρεσκάροντας" την οθόνη.

2. Πληκτρολογήστε τα παρακάτω δεδομένα, τα οποία αποτελούν το μήνυμα:

0810	06	(το δεξιότερο -πρώτο- ψηφίο)
0811	05	(το δεύτερο ψηφίο)
0812	04	(το τρίτο ψηφίο)
0813	03	(το τέταρτο ψηφίο)
0814	02	(το πέμπτο ψηφίο)
0815	01	(το αριστερότερο -έκτο- ψηφίο)

3. Τρέξτε το πρόγραμμα που αρχίζει από τη διεύθυνση 0800. Το μήνυμα "123456" απεικονίζεται στην οθόνη.
4. Το παραπάνω πρόγραμμα μπορεί να απεικονίζει επιπλέον και ένα περιορισμένο σύνολο αλφαβητικών χαρακτήρων. Πιέστε "RESET", και τροποποιήστε το μήνυμα ως εξής:

0810	10
0811	10
0812	14
0813	12
0814	0E
0815	11

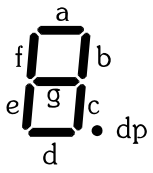
5. Αφού τρέξετε ξανά το πρόγραμμα, τροποποιήστε το και πάλι, φτιάχνοντας με τη βοήθεια του πίνακα 4.7 ένα δικό σας μήνυμα το οποίο θα αποθηκεύσετε στις θέσεις μνήμης 0810-0815.

Πίνακας 4.7. Κωδικοί χαρακτήρων για τη ρουτίνα DCD

Χαρακτήρας	**Δεκαεξαδικός κώδικας**	**Χαρακτήρας**	**Δεκαεξαδικός κώδικας**
0	00	F	0F
1	01	(κενό)	10
2	02	H	11
3	03	L	12
4	04	u	13
5	05	P	14
6	06	o	15
7	07	U	16
8	08	-	17
9	09	c	18
A	0A	I	19
b	0B	8.	1A
c	0C	r	1B
d	0D	-	1C
E	0E		

5. Απευθείας έλεγχος της οθόνης 7 τμημάτων

Η οθόνη ελέγχεται από δύο πόρτες, όπως φαίνεται και στο σχήμα 20 της εισαγωγής. Κάθε bit της πόρτας σάρωσης (scan port) ελέγχει ένα ψηφίο και κάθε bit της πόρτας τμημάτων (segments port) ελέγχει ένα τμήμα του ψηφίου. Το σχήμα 4.2 δείχνει σε τι αντιστοιχεί τα bits της κάθε πόρτας. Για την απεικόνιση ενός μηνύματος, οι χαρακτήρες μετατρέπονται πρώτα στον κώδικα των 7-τμημάτων. Κάθε ψηφίο απεικονίζεται εκ περιτροπής (θέτοντας την πόρτα σάρωσης), και τα κατάλληλα τμήματα ανάβουν για να απεικονίσουν στην οθόνη τον επιθυμητό χαρακτήρα.



Ψηφίο 7-τμημάτων

byte κώδικα 7-τμημάτων								
bit	7	6	5	4	3	2	1	0
	dp	g	f	e	d	c	b	a

Διευθύνσεις που χρησιμοποιούν οι υπορουτίνες STDM και DCD του μLab

Διεύθυνση RAM	Ετικέτα	Κωδικοποιημένο ψηφίο (δεκαεξαδικός κώδικας)
0BF0	UDSP0	0 δεξιότερο ψηφίο
0BF1	UDSP1	1
0BF2	UDSP2	2
0BF3	UDSP3	3
0BF4	UDSP4	4
0BF5	UDSP5	5 αριστερότερο ψηφίο

Διεύθυνση RAM	Ετικέτα	Αποκωδικοποιημένο ψηφίο (κώδικας 7-τμημάτων)
0BFA	DDSP0	0 δεξιότερο ψηφίο
0BFB	DDSP1	1
0BFC	DDSP2	2
0BFD	DDSP3	3
0BFE	DDSP4	4
0BFF	DDSP5	5 αριστερότερο ψηφίο

Έλεγχος Ψηφίου (digit control)

			Αριστερότερα ψηφία					Δεξιότερα ψηφία
Πόρτα σάρωσης Διεύθυνση 2800			6	5	4	3	2	1
bit	7	6	5	4	3	2	1	0
Για παράδειγμα η τιμή:	0	0	0	0	0	0	1	0

θα ανάψει το 2ο από τα δεξιά ψηφίο

Οι γραμμές εξόδου είναι θετικής λογικής: το 1 αντιστοιχεί σε απεικόνιση του ψηφίου

Έλεγχος Τμήματος (segment control)

Πόρτα τμήματος Διεύθυνση 3800	dp	g	f	e	d	c	b	a
bit	7	6	5	4	3	2	1	0
Για παράδειγμα η τιμή:	1	0	1	0	0	1	0	0

θα εμφανίσει τον αριθμό 2 στο ψηφίο που ορίζεται στην πόρτα σάρωσης

Οι γραμμές εξόδου είναι αρνητικής λογικής: το 0 αντιστοιχεί σε απεικόνιση του αντίστοιχου τμήματος

Σχήμα 4.2. Οι πόρτες που χρησιμοποιούνται για τον προγραμματισμό της οθόνης του μLab

Εφαρμογή 4: Ελέγχοντας την οθόνη

Διαδικασία

1. Κωδικοποιήστε και πληκτρολογήστε το πρόγραμμα του πίνακα 4.8. Το πρόγραμμα θέτει αρχικά την "πόρτα σάρωσης" (scan port) να ανάβει το τρίτο

από τα δεξιά ψηφίο. Τα δεδομένα που "παράγουν" το "2" στέλνονται στην πόρτα των τμημάτων (segments).

Πίνακας 4.8. Πρόγραμμα για την απεικόνιση του "2"

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0800	3E	START:	MVI A,4	
0801	04			
0802	32		STA 2800	
0803	00			
0804	28			
0805	3E		MVI A,4	; Θέτει την πόρτα
0806	A4			; των τμημάτων να
0807	32		STA 3800	; απεικονίζει
0808	00			; το "2"
0809	38			
080A	C3		JMP START	
080B	00			
080C	08			

2. Τρέξτε το πρόγραμμα. Ο χαρακτήρας "2" απεικονίζεται στο τρίτο ψηφίο από τα δεξιά. Σημειώστε ότι είναι φωτεινότερος από ό,τι συμβαίνει συνήθως. Όλα τα ψηφία εξετάζονται όμοια και κάθε ψηφίο είναι αναμμένο μόνο για το ένα έκτο του χρόνου. Κανένα ψηφίο δεν είναι αναμμένο συνέχεια.

3. Πατήστε το πλήκτρο "RESET" και στη συνέχεια αλλάξτε τα δεδομένα που στέλνονται στην πόρτα εξέτασης σε 8 hex (0000 1000 binary). Με τον τρόπο αυτό επιλέγεται το τέταρτο ψηφίο από τα δεξιά.

4. Τρέξτε το πρόγραμμα. Ο χαρακτήρας μετακινήθηκε μία θέση προς τα αριστερά.

5. Σταματήστε το πρόγραμμα και αλλάξτε τα δεδομένα που στέλνονται στην πόρτα των τμημάτων σε 9B. Βοηθούμενοι και από το σχήμα 2 προσπαθήστε να μαντέψετε ποιος χαρακτήρας θα απεικονιστεί..

6. Τρέξτε το πρόγραμμα. Ένας νέος χαρακτήρας απεικονίζεται. Σημειώστε ότι νέοι χαρακτήρες μπορούν να δημιουργηθούν κατά βούληση, αφού κάθε τμήμα ελέγχεται απ' ευθείας.

7. Με όμοιο τρόπο φτιάξτε ένα δικό σας χαρακτήρα τροποποιώντας το παραπάνω πρόγραμμα.

Το επόμενο βήμα προς τη δημιουργία ενός ολοκληρωμένου προγράμματος που να ελέγχει την οθόνη, είναι να "ανάβουν" όλα τα ψηφία. Όπως αναφέρθηκε και νωρίτερα, αυτό επιτυγχάνεται "ανάβοντας" κάθε ψηφίο εκ περιτροπής.

Ουσιαστικά για κάθε ψηφίο εκτελείται η ίδια διαδικασία, μια υπορουτίνα που να γράφει νέα δεδομένα στις πόρτες (σάρωσης και τμημάτων) θα έκανε τη δουλειά μας απλούστερη. Ο πίνακας 4.9 δείχνει μια υπορουτίνα που γράφει τα δεδομένα του καταχωρητή B στην πόρτα σάρωσης και αυτά του C στην πόρτα των τμημάτων.

Πίνακας 9. Υπορουτίνα απεικόνισης

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0830	3E	DISP:	MVI A,FF	; Σβήσε τα
0831	FF			; τμήματα
0832	32		STA 3800	
0833	00			
0834	38			
0835	78		MOV A,B	; Θέτει την
0836	32		STA 2800	; πόρτα
0837	00			; σάρωσης
0838	28			
0839	79		MOV A,C	; Θέτει την
083A	32		STA 3800	; πόρτα
083B	00			; των τμημάτων
083C	38			
083D	C9		RET	

Σημειώστε ότι το πρώτο βήμα είναι να σβήσουν όλα τα τμήματα, ώστε τα δεδομένα από το προηγούμενο ψηφίο να μην απεικονίζονται στιγμιαία. Αν δεν συνέβαινε αυτό, η πόρτα σάρωσης θα "άναβε" το υποδεικνυόμενο από το περιεχόμενο του B ψηφίο, αλλά η πόρτα των τμημάτων θα εξακολουθούσε να περιέχει δεδομένα από το προηγούμενο ψηφίο.

Για να ανάψουν και τα έξι ψηφία, χρειάζεται ένα πρόγραμμα που να δίνει τιμές στους καταχωρητές B και C και να καλεί τη ρουτίνα DISP μια φορά για κάθε ψηφίο. Το πρόγραμμα αυτό φαίνεται στον πίνακα 4.10. Ο C περιέχει κάθε φορά τα δεδομένα που στέλνονται στην πόρτα των τμημάτων από τη ρουτίνα απεικόνισης, ενώ ο B περιέχει τα δεδομένα επιλογής ψηφίου.

Πίνακας 4.10. Πρόγραμμα σάρωσης της οθόνης

Διεύθυνση	Περιεχόμενο	Ετικέτα	Εντολή	Σχόλια
0800	01	START:	LXI B,018E	; δεξιότερο
0801	8E			; ψηφίο
0802	01			
0803	CD		CALL DISP	
0804	30			
0805	08			
0806	01		LXI B,0286	; 2ο ψηφίο
0807	86			
0808	02			
0809	CD		CALL DISP	
080A	30			
080B	08			
080C	01		LXI B,04A1	; 3ο ψηφίο
080D	A1			
080E	04			
080F	CD		CALL DISP	
0810	30			
0811	08			
0812	01		LXI B,08C6	; 4ο ψηφίο
0813	C6			
0814	08			
0815	CD		CALL DISP	
0816	30			
0817	08			
0818	01		LXI B,1083	; 5ο ψηφίο
0819	83			
081A	10			
081B	CD		CALL DISP	
081C	30			
081D	08			
081E	01		LXI B,2088	; 6ο ψηφίο
081F	88			
0820	20			
0821	CD		CALL DISP	
0822	30			
0823	08			
0824	C3		JMP START	
0825	00			
0826	08			

Εφαρμογή 5: Χρησιμοποιώντας την οθόνη

Διαδικασία

1. Πληκτρολογήστε το πρόγραμμα του πίνακα 4.10 καθώς και την υπορουτίνα DISP του πίνακα 4.9.
2. Τρέξτε το πρόγραμμα. Στην οθόνη εμφανίζεται: "ABCD EF".
3. Τροποποιώντας τα δεδομένα των θέσεων μνήμης 0801, 0807, 080D, 0813, 0819 και 081F, προσπαθήστε να εμφανίσετε ένα δικό σας μήνυμα στην οθόνη. Το σχήμα 4.2 θα σας διευκολύνει σίγουρα.

6. Τα θέματα της 4ης εργαστηριακής άσκησης

i. Γράψτε ένα πρόγραμμα, όμοιο με εκείνο του πίνακα 2, το οποίο παράγει ένα beep μόνο όταν πατηθούν στη σειρά δύο συγκεκριμένα πλήκτρα (π.χ. ο διψήφιος αριθμός της ομάδας σας). Το πρόγραμμα αυτό θα μπορούσε να αποτελέσει τη βάση μιας "ηλεκτρονικής κλειδαριάς". Ξεκινήστε σχεδιάζοντας απαραίτητα το διάγραμμα ροής και μετά γράψτε το πρόγραμμα. Πληκτρολογήστε το πρόγραμμα στο μLab και ύστερα ελέγξτε το και τρέξτε το.

ii. Γράψτε ένα πρόγραμμα που να απεικονίζει στα δύο αριστερότερα displays την τιμή του κωδικού του πλήκτρου που πατήθηκε σύμφωνα με τον πίνακα 1. Χρησιμοποιήστε τη ρουτίνα KIND (βλέπε πείραμα 1) για τη λειτουργία ανάγνωσης του πληκτρολογίου και τις STDM και DCD για τη λειτουργία αποστολής των δεδομένων στα displays. Υπενθυμίζουμε ότι όλα τα προγράμματα που αποθηκεύουν δεδομένα στη RAM πρέπει να χρησιμοποιούν διευθύνσεις μεταξύ 0B00H-0BFFH, λόγω του μηχανισμού προστασίας. Συνίσταται όμως η χρησιμοποίηση διευθύνσεων μεταξύ 0B00H-0B90H, επειδή από 0B90H και κάτω αρκετές διευθύνσεις χρησιμοποιούνται από το monitor πρόγραμμα, για αποθήκευση μεταβλητών του συστήματος.

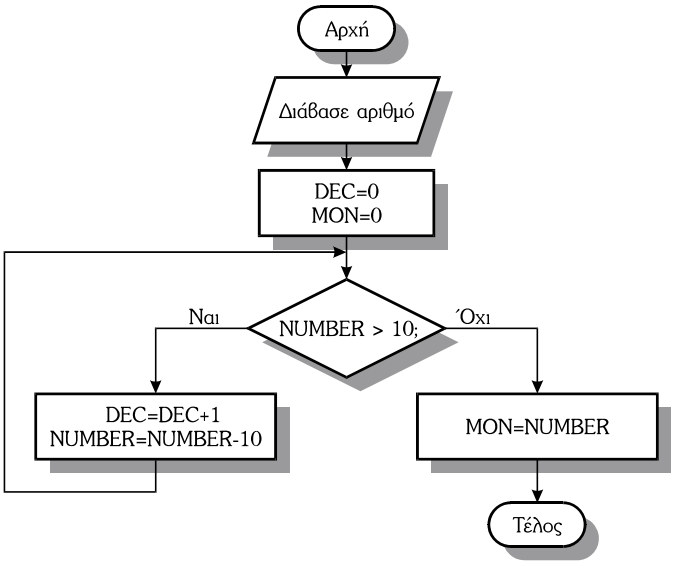
iii. Γράψτε πρόγραμμα, που να εξομοιώνει ένα σύστημα συναγερμού. Υποθέτουμε ότι τα dip switches της θύρας 2000 Hex αντιστοιχούν στις εξόδους των αισθητήρων. Αν κάποιος από αυτούς γίνει ON τότε μέσα σε 10 sec πρέπει να πληκτρολογηθεί ο αριθμός ομάδας (δυο ψηφία) και το 1ο δεκαεξαδικό γράμμα ενός επωνύμου. Αν είναι σωστά, στα τρία δεξιότερα 7-segment απεικονίζεται το μήνυμα OFF. Αλλιώς τίθεται ο συναγερμός με την ενεργοποίηση του BEEP (μια φορά) και το άναμμα του LED στο LSB της θύρας 3000H που υποθέτουμε ότι αντιστοιχεί στην ενεργοποίηση σειρήνας. Ταυτόχρονα στα δυο δεξιότερα 7-segment απεικονίζεται το μήνυμα On.

Είναι φανερό ότι θα πρέπει να υλοποιηθεί μια ρουτίνα χρονοκαθυστέρησης η οποία θα ελέγχει ταυτόχρονα και το πληκτρολόγιο. Η ρουτίνα αυτή δεν θα πρέπει να μπλοκάρει ποτέ, πχ αν πατηθεί μόνιμα ένα πλήκτρο, αλλά μετά από το προκαθορισμένο χρονικό διάστημα θα πρέπει να επιστρέψει, ώστε να χτυπήσει ο συναγερμός. Η ρουτίνα KIND δεν μπορεί να χρησιμοποιηθεί ως έχει, αφού με το που ξεκινά καλεί δύο φορές την KPU, περιμένοντας πρώτα να αφεθεί το τυχόν πατημένο πλήκτρο και μετά να πατηθεί ένα νέο, ώστε να μην πιάνει το ίδιο πάτημα πολλές φορές (περισσότερες πληροφορίες για την

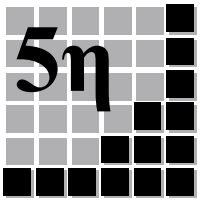
KIND στο Παράρτημα 4 με το listing του monitor). Μια λύση είναι να υλοποιηθεί μια δική σας ρουτίνα που θα κάνει scanning του πληκτρολογίου θα πρέπει όμως να προσεχτούν διάφορα θέματα, όπως το διπλό πάτημα κλπ. Μια δεύτερη λύση θα ήταν η αντιγραφή της KIND στην RAM, με τις απαραίτητες τροποποιήσεις ώστε αυτή να μην μπλοκάρει. Εφόσον όμως υπάρχει ήδη η KIND, θα ήταν καλό να προσπαθήσετε να χρησιμοποιήσετε τον ήδη υπάρχοντα κώδικα του monitor, όχι όμως από την αρχή αλλά μετά τον δεύτερο έλεγχο της KPU. Θα πρέπει να προσέξετε με ποιόν τρόπο θα το κάνετε αυτό, διότι είτε κάνετε jmp, είτε call στο μέσο της KIND τα δύο POP στο τέλος της θα σας χαλάσουν την στοίβα και η επιστροφή από την ρουτίνα θα γίνει σε κάποιο τυχαίο σημείο. Φροντίστε να 'προετοιμάσετε' την στοίβα με τα κατάλληλα δεδομένα.

iv. Γράψτε ρουτίνα που να απεικονίζει στα τρία δεξιότερα 7-segment την τιμή που διαβάζεται από τη θύρα εισόδου 2000 σε δεκαδική μορφή τριών ψηφίων (δηλαδή αν από τη θύρα διαβαστεί ο αριθμός 0110 0001 bin = 61 hex = 097 dec τότε στα τρία δεξιότερα displays να εμφανιστούν οι αριθμοί 0, 9 και 7 αντίστοιχα). Η διαδικασία να είναι συνεχόμενη.

Δίνεται στο σχήμα 3 ένα διάγραμμα ροής για τη μετατροπή ενός δυαδικού αριθμού σε BCD μορφή που αν θέλετε μπορείτε να ακολουθήσετε. Οι δεκάδες (DEC) που λαμβάνονται με βάση αυτόν τον αλγόριθμο είναι από 0 έως 25. Συνεπώς η διαδικασία πρέπει να επαναληφθεί για να προκύψουν οι εκατοντάδες.



Σχήμα 3. Διάγραμμα ροής για το πρόγραμμα του (iv) θέματος.



5η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

- 1. Περιγραφή MASM
- 2. Δημιουργώντας μια βιβλιοθήκη
- 3. . Ο debuger AFDN
  - Εφαρμογή 1: Παράδειγμα χρήσης του AFDN
  - Εφαρμογή 2: Ανάγκη χρησιμοποίησης debuger
- 4. Τα θέματα της εργαστηριακής άσκησης

1. Περιγραφή MASM

1.1 Γενική περιγραφή και παραδείγματα

Προγράμματα γλώσσας assembly δημιουργούνται από αρχεία που περιέχουν source κώδικα. Για να δημιουργήσουμε αρχεία με assembly source κώδικα χρειαζόμαστε έναν text editor ο οποίος να παράγει αρχεία τύπου ASCII (δηλαδή να μην χρησιμοποιεί κάποιο ειδικό format).

Ας ξεκινήσουμε με ένα απλό παράδειγμα assembly που τυπώνει ένα μήνυμα.

```
CODE        SEGMENT
            ASSUME
            CS:CODE,DS:CODE,SS:CODE
            ORG 100h
START:      MOV AX,CODE                 ;AX<-code segment
            MOV DS,AX                   ;DS<-AX
            MOV SS,AX                   ;SS<-AX
            JMP BEGIN
message     DB  "Microcomputer lesson"  ; μήνυμα
lmessage    EQU $-message               ; μήκος μηνύματος
BEGIN       MOV BX,1
            MOV CX,lmessage
            MOV DX,OFFSET message
            MOV AH,40H                  ; φόρτωσε αριθμό για
                                          υπορουτίνα
            INT 21h                     ; γραψίματος DOS και
                                          κάλεσε DOS
            MOV AX,4C00h                ; φόρτωσε DOS εξόδου
                                          υπορουτίνα
            INT 21h                     ; κάλεσε DOS
CODE        ENDS
            END START
```

Γράφουμε το πρόγραμμα αυτό όπως είναι σε έναν editor όπως αναφέραμε παραπάνω και σώζουμε το αρχείο με ένα όνομα έστω first.ASM. Εκτελούμε την εντολή:

MASM [path]first;

όπου path το μονοπάτι στο οποίο βρίσκεται το αρχείο first.ASM. Με αυτό τον τρόπο δημιουργείται το αρχείο first.obj σε object μορφή.
Έπειτα εκτελούμε την εντολή:

LINK [path]first;

και γίνεται το linking παράγοντας τον τελικό εκτελέσιμο κώδικα στο αρχείο first.EXE.

Ταυτόχρονα παίρνουμε το διαγνωστικό μήνυμα "no stack segment" το οποίο δεν πρέπει να μας ανησυχεί αφού οφείλεται στο ότι δεν έχουμε ορίσει stack segment στο πρόγραμμα μας (σε αυτό το απλό πρόγραμμα δεν χρειάζεται αφού δεν χρησιμοποιούμε την στοίβα). Έχοντας το first.EXE το εκτελούμε οπότε το πρόγραμμά μας τρέχει και βλέπουμε να τυπώνεται το μήνυμα:

Microcomputer lesson

Ας ξαναγυρίσουμε όμως στο πρόγραμμά μας. Στον MASM θα πρέπει να ορίσουμε τα τμήματα (segments) του προγράμματός μας. Ο ελάχιστος αριθμός segment που είναι δυνατόν να δηλώσουμε είναι ένα segment κώδικα στο οποίο αποθηκεύεται το πρόγραμμά μας. Η πρώτη εντολή CODE SEGMENT μας δείχνει ότι αρχίζει ένα τμήμα (το τμήμα κώδικα) με το συμβολικό όνομα code. Η επόμενη εντολή ASSUME δίνει στο MASM τις αντιστοιχίες των segments του προγράμματος με τα segments του μΕ 8086 (ο 8086 ως γνωστόν χρησιμοποιεί 4 segments που δείχνουν οι καταχωρητές CS (code segment), DS (data segment), SS (stack segment), ES (extra segment) και που δηλώνουν ότι και τα ονόματά τους). Ας προσέξουμε όμως ότι η εντολή ASSUME δεν φορτώνει τους αντίστοιχους καταχωρητές αλλά αυτό πρέπει να γίνει με δικές μας εντολές που θα γραφτούν στο πρόγραμμα.

Η εντολή "message DB Microcomputer lesson" δηλώνει ότι θα χρησιμοποιηθεί μια διεύθυνση μνήμης με συμβολικό όνομα message, μήκους ενός byte με αρχικό περιεχόμενο τον αντίστοιχο ASCII "M", ενώ στις επόμενες θέσεις μνήμης θα βρίσκονται οι αντίστοιχοι ASCII αριθμοί των υπόλοιπων χαρακτήρων.

Γενικά στο MASM δηλώνουμε τις θέσεις μνήμης που χρησιμοποιεί το πρόγραμμα μας ως εξής:

- Με A DW 15 δηλώνουμε ένα όνομα A που αναφέρεται σε μια θέση μνήμης μήκους μιας λέξης (word) που το αρχικό περιεχόμενό της είναι 15.
- Με B DB 20h δηλώνουμε ένα όνομα B που αναφέρεται σε μια θέση μνήμης μήκους ενός byte που το αρχικό περιεχόμενο της είναι 20 hex.

  Αντίστοιχα υπάρχουν οι τελεστές DD που ορίζει διπλή λέξη (double word - 4 bytes), DQ που ορίζει τετραπλή λέξη (8 bytes).
- Με T DB 10 DUP(?) ορίζουμε ένα πίνακα με όνομα T, μήκους 10 bytes (10 διαδοχικές θέσεις μνήμης μήκους ενός byte) του οποίου τα στοιχεία δεν αρχικοποιούνται σε κάποια τιμή. Αν θέλαμε να αρχικοποιηθούν όλα τα στοιχεία στην τιμή 0 τότε θα γράφαμε T DB 10 DUP(0).
- Με DB 'Whats now ?' δηλώνεται στον assembler ότι θα χρησιμοποιηθούν 11 θέσεις μνήμης (γενικότερα, τόσες θέσεις μνήμης όσοι είναι οι χαρακτήρες του string που ακολουθεί την εντολή DB και βρίσκονται ανάμεσα σε ' ') μήκους 1 byte, των οποίων τα περιεχόμενα αρχικοποιούνται με τις τιμές ASCII των χαρακτήρων που περιέχονται στο string.

Στο παράδειγμά μας, η εντολή EQU κάνει το αριστερά όνομα από την EQU ίσο με την τιμή που βρίσκεται δεξιά της.

Στην lmessage EQU $-message βλέπουμε μία τεχνική που χρησιμοποιείται συχνά. Ο $ παριστάνει τη διεύθυνση του προγράμματος στην οποία βρίσκεται ο τελεστής αυτός, οπότε αφαιρώντας τη διεύθυνση message από την διεύθυνση $ (και η οποία είναι, αν προσέξετε, η διεύθυνση αμέσως μετά τον τελευταίο χαρακτήρα του μηνύματος που αρχίζει στη διεύθυνση message) βρίσκουμε το μήκος του μηνύματος και το αποθηκεύουμε στη διεύθυνση lmessage.

Παρακάτω βλέπουμε τον τελεστή OFFSET που παίρνει την διεύθυνση του message, οποία ακολούθως αποθηκεύεται στον DX. Ακολουθούν 2 χρήσιμες υπορουτίνες του DOS. Η πρώτη (21h/40h), τυπώνει στην συσκευή ή στο αρχείο με file handle που περιέχεται στον BX, (εξ' ορισμού το DOS αναγνωρίζει σαν συσκευή με file handle 1 την οθόνη) το μήνυμα που αρχίζει στη διεύθυνση που δείχνει ο DS:DX και έχει μήκος τόσα bytes όσος είναι ο αριθμός που περιέχεται στον CX.

Παρατηρείστε ότι η διεύθυνση του μηνύματος περιέχεται στο ζεύγος DS:DX και ότι συνεπώς πρέπει να αρχικοποιηθεί κατάλληλα όχι μόνο ο καταχωρητής DX (offset) αλλά και ο DS (segment). Στο συγκεκριμένο παράδειγμα το μήνυμα βρίσκεται στο CODE segment. Η εντολή 'MOV AX,CODE' τοποθετεί στον καταχωρητή AX την τιμή του segment CODE. Η τιμή αυτή ΔΕΝ είναι γνωστή σε χρόνο μεταγλώττισης (compile-time) και καθορίζεται από το

λειτουργικό σύστημα σε χρόνο τρεξίματος (run-time). Με κατάλληλο τρόπο το λειτουργικό σύστημα (relocation vectors) κάθε φορά που "φορτώνει" κάποιο πρόγραμμα "ενημερώνει" παρόμοιες αναφορές. Η όλη διαδικασία δουλεύει μόνο σε εκτελέσιμα τύπου EXE (όχι COM) και δεν πρόκειται να μας απασχολήσει. Έχοντας την τιμή του CODE segment μπορούμε να αρχικοποιήσουμε τον καταχωρητή DS. Συμπληρωματικά αναφέρεται πως τα εκτελέσιμα αρχεία τύπου COM αποτελούνται από ένα μόνο segment και πως το λειτουργικό σύστημα αρχικοποιεί τους CS,DS και SS σε αυτό (δεν θα χρειαζόταν λοιπόν η αρχικοποίηση αν παρήγαμε εκτελέσιμο τύπου COM). Τα εκτελέσιμα αρχεία τύπου COM πάντως, θεωρούνται από την ίδια την Microsoft αναχρονιστικά (obsolete) και προτείνει να μην χρησιμοποιούνται πλέον (για λόγους συμβατότητας τα υποστηρίζει ακόμα).

Μια σημαντική λεπτομέρεια που αφορά μόνο το εκτελέσιμα τύπου COM είναι ότι το offset της πρώτης εντολής προς εκτέλεση (starting point) είναι πάντα σταθερή και ίση με 256 (στο δεκαδικό) ή 100h (δεκαεξαδικό). Για αυτό χρησιμοποιείται συνήθως η οδηγία 'ORG 100h' αμέσως μετά τις δηλώσεις ASSUME. Οι πρώτες 256 διευθύνσεις αρχικοποιούνται από το λειτουργικό σύστημα κάθε φορά που εκτελείται το πρόγραμμα κυρίως για λειτουργίες περάσματος τιμών από το περιβάλλον (environment) και ορισμάτων που δίνονται στη γραμμή εντολών (command line parameters).

Η δεύτερη υπορουτίνα (21h/4Ch) επιστρέφει τον έλεγχο στο DOS μετά το τέλος του προγράμματος (αν την ξεχάσουμε το πρόγραμμα θα κολλήσει). Κατά την κλήση της, υπάρχει η δυνατότητα επιστροφής στο DOS ενός κωδικού τερματισμού, μέσω του AL, που υποδηλώνει αν και τι είδους πρόβλημα προέκυψε κατά την εκτέλεση του προγράμματος. Υπό φυσιολογικές συνθήκες επιστρέφεται 0 (παράβαλε με την εντολή exit(), της γλώσσας C).

Το h ή H στο τέλος μιας σταθεράς δηλώνει ότι ο αριθμός είναι σε δεκαεξαδική μορφή. Αν δεν βάλουμε τίποτα στο τέλος μιας σταθεράς ο MASM θεωρεί ότι είναι από default δεκαδικός. Με b ή B τον θεωρεί δυαδικό και με O ή ο οκταδικό.

Το CODE ENDS δηλώνει ότι τελείωσε η δήλωση του segment με όνομα CODE και τέλος η εντολή END START δηλώνει ότι τελείωσε το πρόγραμμά μας που ξεκινά στη διεύθυνση με label START.

Ας προχωρήσουμε σε ένα παράδειγμα που δηλώνει πιο πολλά segments.

```
STACK      SEGMENT  STACK
```

```
            DW  50 DUP(?)
STACK       ENDS
DATA        SEGMENT
A1          DB  20
A2          DW  5b                                  ; 5 binary
DATA        ENDS
CODE        SEGMENT 'CODE'
            ASSUME CS:CODE, SS:STACK, DS:DATA
START:      MOV  AX, STACK
            MOV  SS, AX
            MOV  AX, CODE
            MOV  CS, AX
            MOV  AX, DATA
            MOV  DS, AX
                        .
                        .
                        .
            program statements here
                        .
                        .
            MOV  AX, 4C00h                          ; Return
            INT  21h                                ; to DOS
CODE        ENDS
            END START
```

Το παράδειγμα αυτό έχει γενική μορφή αφού αφήνει κενό τον κορμό του κυρίως προγράμματος. Εκεί μπορούν να μπουν οποιεσδήποτε εντολές. Το παράδειγμα ξεκινά με τη δήλωση του stack segment που αποτελείται από 50 bytes χωρίς αρχική τιμή (και χωρίς κάποιο όνομα). Ακολουθεί η δήλωση του data segment με το A1 byte και το A2 word. Τέλος, δηλώνεται το τμήμα κώδικα και αξίζει να προσεχθεί ότι φορτώνουμε τους καταχωρητές τμημάτων (segment registers). Αξίζει να σημειωθεί ακόμα, ότι το φόρτωμα κάθε καταχωρητή τμήματος γίνεται με 2 εντολές MOV αφού το ρεπερτόριο εντολών του 8086/88 δεν επιτρέπει το φόρτωμα των καταχωρητών τμημάτων απευθείας από μια διεύθυνση μνήμης.

Παρακάτω δίνουμε την σύνταξη των εντολών ορισμού των segments:

```
name SEGMENT [align] [combine] [use] ['class']
    statements
name ENDS
```

Το name ορίζει το όνομα του τμήματος. Αυτό μπορεί να είναι μοναδικό ή κοινό με άλλα ονόματα σε άλλα segments. Τα segments με το ίδιο όνομα θεωρούνται σαν το ίδιο segment.

Τα προαιρετικά align, combine, use και 'class' δίνουν στο linker και στον assembler εντολές για τον ορισμό των τμημάτων.

Τύπος Align

Ο τύπος align ορίζει από ποιο σημείο της μνήμης θα αρχίσει να γράφεται το segment. Χρησιμοποιούνται οι εξής τύποι:

Align Type	Σημασία
BYTE	Χρησιμοποιεί την επόμενη διαθέσιμη byte διεύθυνση.
WORD	Χρησιμοποιεί την επόμενη διαθέσιμη διεύθυνση λέξης.
DWORD	Χρησιμοποιεί την επόμενη διαθέσιμη διπλής λέξης διεύθυνση. (χρησιμοποιείται για τον 80386)
PARA	Χρησιμοποιεί την επόμενη διαθέσιμη διεύθυνση παραγράφου (16 bytes ανά παράγραφο)
PAGE	Χρησιμοποιεί την επόμενη διαθέσιμη διεύθυνση σελίδας (256 bytes ανά σελίδα)

Τύπος Combine

Ο τύπος combine ορίζει πώς θα συνδυαστούν τα segments με το ίδιο όνομα. Οι κυριότεροι τύποι είναι:

Combine Type	Σημασία
PUBLIC	Ενώνει όλα τα segment με το ίδιο όνομα.
STACK	Το ίδιο με PUBLIC με μόνη διαφορά ότι όλες οι διευθύνσεις είναι σχετικές με τον καταχωρητή SS αφού όπως σημειώσαμε παραπάνω η εντολή ASSUME δε φορτώνει τους καταχωρητές αυτούς.

Τύπος Class

Ο τύπος class χρησιμοποιείται για να συνδέσει segment με διαφορετικό όνομα αλλά με παρόμοια λειτουργία. Πρέπει να μπαίνει πάντα σε μονά εισαγωγικά (').

Για παράδειγμα έστω τα επόμενα δύο αρχεία:

```
;test1.asm
DATA        SEGMENT     PAGE  'DATA'
            DB 21 DUP(?)
DATA        ENDS

STACK             SEGMENT     PAGE  STACK
            DW 50 DUP(?)
STACK             ENDS

ANOTHER     SEGMENT     PARA  PUBLIC 'DATA'
            DB 1 DUP(?)
ANOTHER     ENDS

EXTRA             SEGMENT     WORD PUBLIC
            DB 22 DUP(?)
EXTRA             ENDS

CODE        SEGMENT BYTE
        ASSUME  CS:CODE,DS:CODE,SS:STACK
        ORG 100H
START:      JMP BEGIN
message     DB "Micro"
lmessage    EQU $-message
BEGIN:      MOV BX,1
            MOV AX,CODE
            MOV DS,AX
            MOV AX,STACK
            MOV SS,AX
        MOV CX,lmessage
        MOV DX,OFFSET message
        MOV AH,40H
        INT 21H
```

```
        MOV AX,4C00H
        INT 21H
CODE    ENDS
        END

;test2.asm
ANOTHER SEGMENT PARA PUBLIC 'DATA'
        DB 10 DUP(0)
ANOTHER ENDS

EXTRA           SEGMENT WORD PUBLIC
        DB 10 DUP(?)
EXTRA           ENDS

STACK   SEGMENT BYTE STACK
        DB 10 DUP(?)
STACK           ENDS

SOMANY  SEGMENT BYTE PUBLIC
        DB 30 DUP(?)
SOMANY  ENDS
        END
```

Περνάμε τα αρχεία αυτά πρώτα από τον assembler και τα συνδέουμε στην συνέχεια τα δύο object (.obj) αρχεία με την βοήθεια του linker (link test1+test2,test1,test1). Εξετάζοντας τώρα τα αποτελέσματα του αρχείου test1.map το οποίου και περιέχει της πληροφορίες για το πως διατάσσονται τα διάφορα segments παίρνουμε τα εξής αποτελέσματα:

Start	Stop	Length	Name	Class
00000H	00014H	00015H	DATA	DATA
00020H	00039H	0001AH	ANOTHER	DATA
00040H	000ADH	0006EH	STACK	
000AEH	000CDH	00020H	EXTRA	
000CEH	001F1H	00124H	CODE	
001F2H	0020FH	0001EH	SOMANY	

Παρατηρούμε ότι τα segments με το ίδιο όνομα ή την ίδια κλάση διασυνδέονται το ένα μετά το άλλο σύμφωνα με τον τύπο Align. Μεγαλύτερη προτεραιότητα έχει ο τύπος της κλάσης.

1.2 Υπορουτίνες

Για να υλοποιήσουμε υπορουτίνες στον 8086 μεταφέρουμε τον έλεγχο του προγράμματος σε μια υπορουτίνα με μια CALL εντολή και επιστρέφουμε από αυτή με μια εντολή RET. Ο MASM μας επιτρέπει να δηλώσουμε ακόμη με ένα όνομα μια υπορουτίνα όπως παρακάτω:

```
name    PROC
        .
        .
        .
        εντολές της υπορουτίνας
        .
        .
        .
        RET
name    ENDP
```

1.3 Το λάθος της μελλοντικής αναφοράς

Όταν ο assembler μεταφράζει ένα πρόγραμμα το κάνει σε 2 περάσματα. Στο πρώτο πέρασμα κάθε εντολή που αναφέρεται σε σύμβολα που δεν έχουν οριστεί ακόμα παράγει λάθη υποτίθεται ότι θα βρεθούν στο δεύτερο πέρασμα. Αυτό αναφέρεται σαν λάθος μελλοντικής αναφοράς. Τέτοια λάθη δε δημιουργούν πρόβλημα εφόσον τα σύμβολα βρεθούν στο δεύτερο πέρασμα. Μερικές φορές όμως όταν χρησιμοποιηθούν πολλά τμήματα στο ίδιο πρόγραμμα μπορεί να δημιουργηθεί πρόβλημα. Όταν ο assembler στο πρώτο πέρασμα βρει ένα αδήλωτο σύμβολο υποθέτει ότι η διεύθυνσή του θα βρεθεί στο ίδιο segment στο δεύτερο πέρασμα. Δηλαδή υποθέτει ότι η διεύθυνση είναι near (κοντινή). Αν κατά τη διάρκεια του δεύτερου περάσματος το σύμβολο βρεθεί σε άλλο segment (δηλαδή σε μια far διεύθυνση) ένα λάθος θα προκύψει και θα τυπωθεί ένα διαγνωστικό μήνυμα. Παρόμοιο πρόβλημα μπορεί να συμβεί με αναφορές δεδομένων. Ο assembler υποθέτει ότι οι μελλοντικές αναφορές δεδομένων θα βρεθούν στο data

segment, γεγονός το οποίο ίσως να μη συμβαίνει. Το ακόλουθο πρόγραμμα δημιουργεί ένα λάθος φάσης.

```
CODE    SEGMENT
        ASSSUME CS:CODE
        MOV    AX,COUNT
        COUNT    DW 00
        .
        .
        .
CODE    ENDS
```

Στο πρώτο πέρασμα 2 bytes δεσμεύονται για τη διεύθυνση του count υποθέτοντας ότι βρίσκεται στο data segment (οπότε χρειάζεται μόνο το offset - 2 bytes - για να προσπελαστεί το δεδομένο που είναι αποθηκευμένο στη διεύθυνση count). Στο δεύτερο πέρασμα όμως, όταν η count είναι γνωστό ότι βρίσκεται στο code segment, πρέπει να δημιουργηθεί μια υπέρβαση τμήματος (segment override), κάτι που απαιτεί άλλα δύο bytes.

Τα λάθη φάσης συμβαίνουν όταν ο αριθμός των bytes που δεσμεύονται στο πρώτο πέρασμα του assembler δεν συμφωνεί με τον αριθμό των bytes του δεύτερου περάσματος.

Για να αποφύγουμε τα προηγούμενα λάθη ο καλύτερος τρόπος είναι να γράφουμε τα προγράμματά μας με την ακόλουθη σειρά:

```
[data segment(s)]
[stack segment]
[code segment(s)
 with procedures]

[Main code
 segment
 [procedures]
 [main program]]
```

Με αυτό τον τρόπο όλα τα σύμβολα θα είναι γνωστά από το πρώτο πέρασμα και θα αποφύγουμε τα λάθη.

1.4 Γράφοντας assembly υπορουτίνες σε διαφορετικά αρχεία

Ένα πρόγραμμα assembly δεν είναι απαραίτητο να βρίσκεται σε ένα μόνο αρχείο. Αντίθετα μπορούμε να γράφουμε διάφορα κομμάτια του (συνήθως υπορουτίνες) σε διαφορετικά αρχεία. Έπειτα με τον MASM μετατρέπουμε τα αρχεία σε μορφή .OBJ και με τον LINK κάνουμε την σύνδεση ολόκληρου του προγράμματος.

Αν A.OBJ, B.OBJ και C.OBJ τα τρία αρχεία που θέλουμε να συνδέσουμε, εκτελώντας LINK A.OBJ+B.OBJ+C.OBJ θα παραχθεί το τελικό εκτελέσιμο πρόγραμμα A.EXE. Ας δούμε όμως τι απαραίτητες δηλώσεις θα πρέπει να υπάρχουν στα διαφορετικά αρχεία ώστε έπειτα να μπορούν αυτά να συνδυαστούν μεταξύ τους.

Κάθε υπορουτίνα για να μπορεί να καλείται από άλλα αρχεία θα πρέπει να δηλώνεται PUBLIC. Έτσι αν η B υπορουτίνα καλεί την A (οι A και B είναι σε διαφορετικά αρχεία) η A θα πρέπει να δηλώνεται στο αρχείο της ως:

```
PUBLIC A
```

Τότε μπορεί να εκτελεστεί από άλλα αρχεία.

Μια υπορουτίνα B που καλεί μια άλλη A σε διαφορετικό αρχείο θα πρέπει να δηλώνει την A σαν EXTRN και να καθορίζει το είδος του καλέσματος (NEAR ή FAR). Έτσι στο παραπάνω παράδειγμα πριν οριστεί η B πρέπει να δηλώσουμε:

```
EXTRN  A:FAR  (αν καλεί την A σαν FAR υπορουτίνα).
```

Η A δηλώνεται σαν NEAR αν βρίσκεται στο ίδιο segment με την B και αυτό έχει δηλωθεί σαν PUBLIC. Σαν ολοκληρωμένο παράδειγμα ας δούμε το παρακάτω:

```
CODE        SEGMENT PUBLIC
            PUBLIC  ROUTINE1
            EXTRN   ROUTINE2:NEAR
            ASSUME  CS:CODE
ROUTINE1    PROC FAR
            .
            .
            .
```

```
                CALL  ROUTINE2
                .
                .
                .
                RET
ROUTINE1        ENDP
CODE            ENDS
                END
```

Το παραπάνω πρόγραμμα βρίσκεται σε ένα αρχείο EX1.ASM. Έστω ότι στο αρχείο EX2.ASM βρίσκεται το εξής:

```
CODE            SEGMENT PUBLIC
                PUBLIC  ROUTINE2
                ASSUME  CS:CODE
ROUTINE2        PROC NEAR
                .
                .
                .
                RET
ROUTINE2        ENDS
                END
```

Το αρχείο EX1.ASM περιέχει μια υπορουτίνα ROUTINE1 που δηλώνεται PUBLIC ώστε να μπορεί να χρησιμοποιηθεί από άλλες σε διαφορετικά αρχεία. Επίσης δηλώνεται σαν FAR ώστε να μπορεί να κληθεί και από πρόγραμμα το οποίο βρίσκεται σε άλλο segment. Για να μπορεί να καλέσει η ROUTINE1 την ROUTINE2 χρησιμοποιείται η εντολή

EXTRN ROUTINE2: NEAR

Επειδή οι ROUTINE1, ROUTINE2 βρίσκονται στο ίδιο segment (με την ονομασία CODE) στην EXTRN δηλώνουμε NEAR.

Στο αρχείο EX2.ASM δηλώνεται η ROUTINE2 σαν PUBLIC και ορίζεται παρακάτω. Από τα παραπάνω καταλαβαίνουμε πως κάθε φορά που χρησιμοποιείται σε ένα αρχείο μια EXTRN εντολή, για να έχουμε δικαίωμα αναφοράς στην υπορουτίνα αυτή, σε κάποιο άλλο αρχείο (αυτό που ορίζεται η υπορουτίνα που βρίσκεται στην EXTRN) θα υπάρχει μια PUBLIC εντολή.

1.5 Macros

Τα macros χρησιμοποιούνται για να δημιουργούμε νέες εντολές που αναγνωρίζονται από τον assembler. Η γενική μορφή είναι:

```
name    MACRO   arg1 arg2 arg3 ...
        ...
        εντολές
        ...
ENDM
```

όπου name το όνομα του macro και arg1, arg2, ..., argN παριστάνει τις παραμέτρους του macro. Κάθε παράμετρος πρέπει να παριστάνει μια σταθερά. Έτσι δεν μπορούμε να χρησιμοποιήσουμε σαν παράμετρο έναν καταχωρητή.

Παράδειγμα: Να γραφεί πρόγραμμα (macro) που να κάνει λογική αντιστροφή του AX και έπειτα να τον ολισθαίνει κυκλικά προς τα δεξιά τόσες φορές όσες δείχνει ένας αριθμός που περνιέται ως όρισμα.
Λύση:

```
FIXAX   MACRO   TIMES
        NOT AX
        MOV CL,TIMES
        ROR AX,CL
ENDM
```

Αφού προσθέσουμε το macro αυτό στο πρόγραμμά μας ο assembler γνωρίζει άλλη μια εντολή, την FIXAX n που κάνει αυτό που περιγράψαμε παραπάνω.

1.6 Παράμετροι του MASM

Ο MASM εκτός από το object αρχείο παράγει και άλλα δύο. Το ένα με επέκταση .LST περιέχει τις σχετικές διευθύνσεις και τον δεκαεξαδικό κώδικα μαζί με τον αρχικό source κώδικα. Το δεύτερο αρχείο με επέκταση .CRF είναι ένα αρχείο που περιέχει με αλφαβητική σειρά όλα τα labels και όλα τα σύμβολα. Έστω ένα assembly πρόγραμμα που βρίσκεται στο αρχείο exam.ASM. Σαν παραμέτρους του MASM μπορούμε να βάλουμε τα παρακάτω:

```
MASM  /V/Z  exam,,,;
```

Οι /V, /Z δίνουν εντολή στο MASM να στείλει πρόσθετα στατιστικά και πληροφορίες λαθών στην οθόνη κατά τη διάρκεια της μετάφρασης.

Κατά τη μετάφραση αρχείων τα οποία πρόκειται να συνδεθούν με object code της γλώσσας C πρέπει να χρησιμοποιείται και η παράμετρος /mx (ή MASM -mx fname.asm για την έκδοση 5.1 και μετά) η οποία κάνει τη μετάφραση, του αρχείου assembly, case sensitive (δηλαδή ξεχωρίζει τα κεφαλαία και τα πεζά γράμματα, συνεπώς τα συμβολικά ονόματα Address1 και ADDRESS1 θεωρούνται διαφορετικά, πράγμα που δε θα συνέβαινε αν δεν είχε χρησιμοποιηθεί η παράμετρος mx). Η έξοδος της παραπάνω εντολής θα είναι 3 αρχεία. Το object αρχείο exam.obj, το listing αρχείο exam.lst και το αρχείο αναφοράς exam.crf. Υπάρχουν και άλλες παράμετροι του MASM που εδώ δεν θα μας απασχολήσουν λόγω της σύντομης παρουσίασης.

2. Δημιουργώντας μια βιβλιοθήκη

Η βιβλιοθήκη είναι μια συλλογή από ρουτίνες που μπορούν να χρησιμοποιηθούν από πολλά προγράμματα. Το αρχείο της βιβλιοθήκης καλείται από τον linker όταν γίνεται η διασύνδεση του προγράμματος.

Για να δημιουργήσουμε μια βιβλιοθήκη ξεκινάμε γράφοντας της ρουτίνες που θα την αποτελούν και αφού της περάσουμε από τον assembler χρησιμοποιούμε το πρόγραμμα LIB.

Έστω για παράδειγμα οι δύο παρακάτω ρουτίνες. Η μια διαβάζει ένα πλήκτρο από το πληκτρολόγιο και γυρίζει το χαρακτήρα που διαβάστηκε στον καταχωρητή AL. Η δεύτερη τυπώνει τον ASCII κωδικό που περιέχεται στον AL στην οθόνη:

```
;proc1.asm
LIB         SEGMENT 'CODE'
            ASSUME CS:LIB
            PUBLIC READ_KEY
READ_KEY    PROC FAR
            PUSH DX
READ_KEY1:  MOV AH,6
            MOV DL,0FFH
            INT 21H
            JE READ_KEY1
            POP DX
            RET
READ_KEY    ENDP
LIB         ENDS
            END

;proc2.asm
LIB         SEGMENT 'CODE'
            ASSUME CS:LIB
            PUBLIC ECHO
ECHO        PROC FAR
            PUSH DX
            MOV AH,6
            MOV DL,AL
            INT 21H
            POP DX
            RET
READ_KEY    ENDP
LIB         ENDS
            END
```

Ιδιαίτερα προσεκτική πρέπει να είμαστε στο γεγονός ότι τα ονόματα των ρουτινών που θα ενσωματώσουμε σε μια βιβλιοθήκη πρέπει να είναι δηλωμένα ως PUBLIC και κατά κανόνα οι ρουτίνες θα είναι FAR για να μπορούν να καλούνται και από διαφορετικά segments.

Για να φτιάξω την βιβλιοθήκη καλώ στη συνέχεια το πρόγραμμα LIB. Δίνω το όνομα της βιβλιοθήκης που θέλω να φτιάξω και στη συνέχεια στην ερώτηση Operations? Δίνω τα ονόματα των object αρχείων της ρουτίνας που θέλω να προσθέσω στη μορφή αρχειο1+αρχείο2+. Για να αφαιρέσουμε μία ρουτίνα από την βιβλιοθήκη πρέπει να χρησιμοποιήσουμε το σύμβολο '-'.

Το list αρχείο περιέχει πληροφορίες για τις ρουτίνες της βιβλιοθήκης.

Για να καλέσουμε τώρα μια ρουτίνα από την βιβλιοθήκη θα αρκεί να την καλέσουμε με το όνομα της. Βέβαια θα πρέπει πρώτα να την έχουμε δηλώσει ως EXTRN για να ειδοποιήσουμε τον assembler ότι τα συμβολικά ονόματα αυτά δεν περιέχονται στο παρόν αρχείο. Επίσης στη διάρκεια της διασύνδεσης θα πρέπει να δίνουμε στον linker την βιβλιοθήκη στην οποία περιέχεται οι ρουτίνες που χρησιμοποιούμε στο κυρίως πρόγραμμα.

Για παράδειγμα το παρακάτω πρόγραμμα διαβάζει ένα χαρακτήρα και τον τυπώνει στην οθόνη κάνοντας χρήση των ρουτινών READ_KEY και ECHO.

```
;progr.asm
                EXTRN READ_KEY:FAR;
                EXTRN ECHO:FAR;
STACK           SEGMENT      PAGE   STACK
                DW 50 DUP(?)
STACK           ENDS
CODE            SEGMENT BYTE
                ASSUME  CS:CODE,DS:CODE,SS:STACK
START:          JMP BEGIN
BEGIN:          MOV AX,CODE
                MOV DS,AX
                MOV AX,STACK
                MOV SS,AX
                CALL READ_KEY
                CALL ECHO
                MOV AX,4C00H
                INT 21H
CODE            ENDS
                END
```

4. Τα θέματα της 5ης Εργαστηριακής Άσκησης

i. Δίδεται από το πληκτρολόγιο ένας δυαδικός αριθμός των 12 bits (αρχίζοντας από το MSB) όπου να μην αναγνωρίζεται άλλο πλήκτρο δηλ. τα υπόλοιπα να αγνοούνται). Το αποτέλεσμα σε δεκαδική μορφή πρέπει να τυπωθεί στην οθόνη ως εξής:
$b_{11}b_{10} \ldots b_1b_0$ = ABCD. Η διαδικασία να είναι συνεχόμενη και να βγαίνουμε μόνο με το γράμμα Q. Κάθε νέος υπολογισμός να τυπώνεται στην αρχή της επόμενης γραμμής.

ii. Ένας τετραψήφιος BCD παρέχεται από το πληκτρολόγιο. Να μετατραπεί στην ισοδύναμη δεκαεξαδική μορφή. Ο αριθμός BCD και το αποτέλεσμα να τυπωθούν στην οθόνη. Να χρησιμοποιηθεί ο αλγόριθμος του παραδείγματος που ακολουθεί:

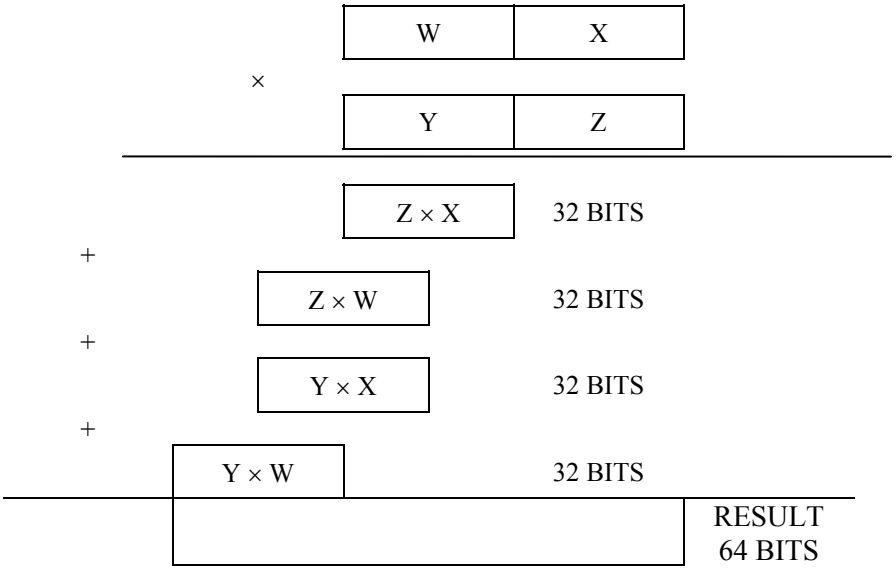
$$4596 = (4 \times 1000) + (5 \times 100) + (9 \times 10) + (6 \times 1)$$

1000	= 03E8H	άρα	4000	= 4 X 3E8H	= 0FA0H
100	= 0064H	άρα	500	= 5 X 064H	= 01F4H
10	= 000AH	άρα	90	= 9 X 00AH	= 005AH
1	= 0001H	άρα	6	= 6 X 001H	= 0006H
				άρα 4596	= 11F4H

Το πρόγραμμα να δέχεται μόνο BCD ψηφία και να αγνοεί όλα τα υπόλοιπα πλήκτρα και να τυπώνει το μήνυμα : GIVE FOUR NUMBERS: 4596
Να αναμένει [ENTER] μόνο στην περίπτωση 4 έγκυρων BCD ψηφίων δίνοντας στην επόμενη γραμμή το μήνυμα: 4596 =11F4H και να αγνοεί το [ENTER] σε όλες τις άλλες περιπτώσεις. Η διαδικασία να είναι συνεχόμενη και να τερματίζεται πάλι με το γράμμα Q. Κάθε νέος υπολογισμός με το μήνυμά του να τυπώνεται στην αρχή της επόμενης γραμμής.

iii. Η εντολή MUL του 8086/88 επιτρέπει τον πολλαπλασιασμό ενός αριθμού των 16 Bits με έναν άλλο αριθμό των 16 bits και δίνει ένα αποτέλεσμα των 32 bits. Σε μερικές περιπτώσεις όμως χρειάζεται να πολλαπλασιαστούν δύο αριθμοί των 32 bits για να δώσουν ένα αποτέλεσμα των 64 bits. Αυτό μπορεί να γίνει εύκολα με τη χρήση της εντολής MUL και μερικές προσθέσεις. Το διάγραμμα του σχήματος 1 υποδεικνύει έναν τρόπο, με τον οποίο μπορεί να γίνει αυτό.

Η βασική ιδέα είναι να χρησιμοποιηθεί η MUL για να υπολογιστούν τα μερικά γινόμενα. Και στη συνέχεια αυτά να προστεθούν όπως δείχνει το σχήμα. Με τη βοήθεια του debuger γράψτε ένα πρόγραμμα που να εκτελεί τον πολλαπλασιασμό χρησιμοποιώντας αυτόν τον τρόπο. Να διερευνήσετε, αν είναι δυνατόν, ο πολλαπλασιασμός να εκτελεστεί χωρίς τη χρήση της μνήμης για την αποθήκευση ενδιάμεσων αποτελεσμάτων.



Σχήμα 1. Υλοποίηση πολλαπλασιασμού 32 bit

iv. Να δημιουργηθεί βιβλιοθήκη που να περιέχει τις εξής ρουτίνες :

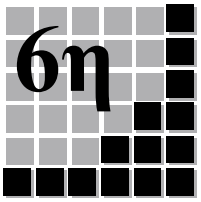
α) Μια ρουτίνα που παίρνει ως παράμετρο το byte που περιέχεται στον καταχωρητή AL και να επιστρέφει τις ASCII των δύο δεκαεξαδικών ψηφίων του. Η τιμή αυτή να επιστρέφεται μέσω του καταχωρητή AX με τον AH να περιέχει το υψηλότερης αξίας και τον AL το ψηφίο χαμηλότερης αξίας.

β) Μια ρουτίνα που να εκτελεί την αντίστροφη διαδικασία, δηλαδή να παίρνει στον κατάχωρητή AX δύο ASCII χαρακτήρες που αναπαριστούν τα δεκαεξαδικά ψηφία ενός αριθμού των 8-bits και να επιστρέφει στον καταχωρητή AL την τιμή του αριθμού αυτού. Η ρουτίνα θα πρέπει να

δέχεται και τους χαρακτήρες ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’ εκτός των αντίστοιχων κεφαλαίων.

Χρησιμοποιώντας τις πιο πάνω ρουτίνες μέσω τις βιβλιοθήκης να γραφτεί πρόγραμμα που να υλοποιεί μια αριθμομηχανή με δυνατότητες πρόσθεσης και αφαίρεσης δεκαεξαδικών αριθμών 16-bits. Το πρόγραμμα να μπορεί να δέχεται ένα δεκαεξαδικό αριθμό, ένα από τα σύμβολα ‘+’ ή ‘-‘ και ένα ακόμη δεκαεξαδικό αριθμό να τοποθετεί αυτόματα το σύμβολο του ‘=’ και να δίνει το αποτέλεσμα σε δεκαεξαδική μορφή. Οι δεκαεξαδικοί αριθμοί να είναι από 1 έως 4 ψηφία. Να αγνοούνται όλα τα υπόλοιπα πλήκτρα. Η διαδικασία να είναι συνεχιζόμενη και να διακόπτεται με το πλήκτρο Q.

Παρατήρηση: Την βιβλιοθήκη αυτή μπορείτε να την εμπλουτίζεται στη συνέχεια με ρουτίνες που θα σας χρειαστούν (π.χ. με ρουτίνες μετατροπής δυαδικών αριθμών σε δεκαδική μορφή και αντίστροφα όπως και με ρουτίνες εισαγωγής ή απεικόνισης δεκαδικών, δεκαεξαδικών και δυαδικών αριθμών).



6η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

- 1. Σύνδεση ρουτινών MASM με γλώσσες υψηλού επιπέδου
- 2. Ασύγχρονη επικοινωνία μέσω software σε προσωπικό υπολογιστή
- 3. Τα θέματα της εργαστηριακής άσκησης

1. Σύνδεση ρουτινών MASM με γλώσσες υψηλού επιπέδου

1.1 Γενικές οδηγίες για σύνδεση με υπορουτίνες.ASM

Η μέθοδος σύνδεσης των υπορουτινών assembly γλώσσας με γλώσσες υψηλού επιπέδου αποτελείται από τα παρακάτω βήματα:

1. Αρχικοποίηση της υπορουτίνας.
2. Είσοδος στην υπορουτίνα.
3. Δέσμευση τοπικών δεδομένων.
4. Σώσιμο των καταχωρητών.
5. Προσπέλαση παραμέτρων υπορουτίνας.
6. Επιστροφή τιμής (προαιρετικά).
7. Έξοδος από την υπορουτίνα.

1. Αρχικοποίηση της υπορουτίνας

Καταρχήν πρέπει να γνωρίζουμε το μοντέλο μνήμης που χρησιμοποιούμε στη γλώσσα υψηλού επιπέδου. Αν το μοντέλο μνήμης είναι small ή compact τότε δηλώνουμε την υπορουτίνα assembly NEAR (σε διαφορετική περίπτωση δηλώνεται FAR). Η assembly υπορουτίνα η οποία καλείται πρέπει να δηλωθεί PUBLIC ώστε να μπορούμε να την καλέσουμε από άλλες ρουτίνες σε διαφορετικά αρχεία. Οι καθολικές μεταβλητές ή υπορουτίνες που προσπελαύνει πρέπει να δηλωθούν σαν EXTRN. Τέλος, ανάλογα με την γλώσσα υψηλού επιπέδου χρησιμοποιούνται ειδικά ονόματα για τα διάφορα segments (βλέπε παρακάτω την αντίστοιχη παράγραφο της γλώσσας υψηλού επιπέδου).

2. Είσοδος στην υπορουτίνα

Η υπορουτίνα ξεκινά πάντα με δύο εντολές:

```
PUSH BP
MOV  BP,SP
```

Με αυτές σώζεται ο καταχωρητής BP στη στοίβα και σαν καινούργια τιμή παίρνει την τρέχουσα τιμή του δείκτη στοίβας (stack pointer) έτσι ώστε χρησιμοποιώντας τον καταχωρητή BP να μπορούμε να έχουμε πρόσβαση σε όλες τις θέσεις της στοίβας χωρίς τη χρήση της εντολής POP.

3. Δέσμευση τοπικών δεδομένων (προαιρετικά)

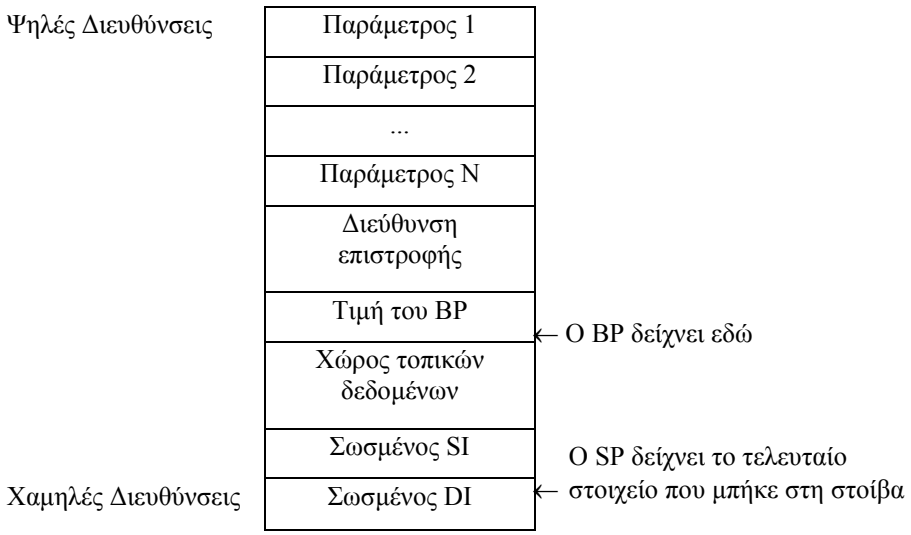
Για να δημιουργήσουμε χώρο τοπικών δεδομένων μειώνουμε τον SP (μειώνουμε, αφού, ας μην ξεχνάμε, η στοίβα των 8086/88 "επεκτείνεται προς τα κάτω" δηλαδή από τις ψηλότερες διευθύνσεις προς τις χαμηλότερες) ώστε να δημιουργηθεί κενός χώρος στη στοίβα. Αυτός ο χώρος χρησιμοποιείται για αποθήκευση τοπικών δεδομένων.

4. Σώσιμο των καταχωρητών

Οι γλώσσες υψηλού επιπέδου χρειάζονται τις τιμές των καταχωρητών SI, DI, SS και DS (μαζί με τον BP που έχει ήδη σωθεί στο πρώτο βήμα). Για αυτό κάθε ένας καταχωρητής από τους παραπάνω, που μεταβάλλεται από την υπορουτίνα assembly θα πρέπει να σώζεται αρχικά στη στοίβα με την εντολή PUSH.

5. Προσπέλαση παραμέτρων

Αφού ολοκληρώσαμε τις παραπάνω ενέργειες, γράφουμε το κύριο μέρος της υπορουτίνας. Για να γράψουμε εντολές που χρησιμοποιούν παραμέτρους της καλούσας υπορουτίνας υψηλού επιπέδου, θεωρούμε τη μορφή της στοίβας όπως αυτή είναι έπειτα από το κάλεσμα της ρουτίνας assembly (Σχήμα 1).



Σχήμα 1. Μορφή της στοίβας κατά την κλήση μιας υπορουτίνας assembly από μια γλώσσα υψηλού επιπέδου

Σημειώστε ότι στο παραπάνω σχήμα έχει απεικονιστεί το γεγονός ότι η στοίβα επεκτείνεται από τις ψηλότερες διευθύνσεις προς τις χαμηλότερες (από πάνω προς τα κάτω). Η στοίβα τροποποιήθηκε έτσι από τα παρακάτω γεγονότα:

1. Το πρόγραμμα που κάλεσε την υπορουτίνα έβαλε στη στοίβα τις παραμέτρους.
2. Το πρόγραμμα αυτό με την εντολή κλήσης υπορουτίνας (CALL) έθεσε αυτόματα στη στοίβα τη διεύθυνση επιστροφής (2 bytes για κλήση near ή 4 bytes για far).
3. Η πρώτη εντολή της υπορουτίνας έσωσε τον BP και μετέφερε την τιμή του SP στον BP. Έτσι τελικά ο BP δείχνει την παλιά τιμή του.
4. Ενώ ο BP μένει σταθερός ο SP μειώνεται για να δείχνει στο τελευταίο στοιχείο που μπήκε στη στοίβα. Έτσι για να προσπελάσουμε την παράμετρο N-1 (δες σχήμα 1) πρέπει στην τιμή του BP να προσθέσουμε:

   2 + (μέγεθος διεύθυνσης επιστροφής, 4 ή 2)
   + (μέγεθος παραμέτρου N).

   Όλα τα μεγέθη μετρούνται σε bytes.

Για παράδειγμα (σχήμα 1) ας θεωρήσουμε μια FAR υπορουτίνα που έλαβε δύο παραμέτρους. Για να φορτώσουμε την παράμετρο N-1 στον καταχωρητή BX αν θεωρήσουμε ότι η N καταλαμβάνει χώρο 2 bytes, χρησιμοποιούμε στο πρόγραμμα μας την εντολή: MOV BX,[BP+8]

6. Επιστροφή τιμής κατά τη κλήση συνάρτησης C (function)

Αν η υπορουτίνα που γράφουμε σε assembly χρειάζεται να επιστρέφει κάποια τιμή στο καλούν πρόγραμμα (που είναι γραμμένο σε γλώσσα υψηλού επιπέδου), τότε πρέπει να λάβουμε υπόψη τα επόμενα. Ανάλογα με το μέγεθος του δεδομένου αυτό επιστρέφεται όπως φαίνεται στον πίνακα που ακολουθεί.

Αν η τιμή που επιστρέφεται είναι μεγαλύτερη από 4 bytes τότε η υπορουτίνα assembly πρέπει να δεσμεύσει χώρο για αυτήν και να τοποθετήσει την διεύθυνση της στον DX:AX. Η καλούσα συνάρτηση πρέπει να φροντίσει να αντιγράψει την τιμή αυτή στην πρέπουσα θέση. Όταν επιστρέφονται near pointers τότε επιστρέφονται στον AX. Οι far pointers επιστρέφονται τοποθετώντας το segment στον DX και το offset στον AX.

Μέγεθος δεδομένου	Επιστρέφεται στον καταχωρητή
char(1 byte)	AX (με επέκταση προσήμου αν είναι προσημασμένο)
int(2 byte)	AX
enum	AX
long	DX:AX το high word στον DX και το low word στον AX
near*(2 byte)	AX
far*(4 byte)	DX:AX (segment:offset)
float - double	Επιστρέφονται στην κορυφή της στοίβας του συνεπεξεργαστή (είτε υπαρκτού είτε σε προσομοίωση)
struct(1 ή 2 byte)	AX
struct(4 byte)	DX:AX
struct (3 byte ή > 4 byte)	Εισάγεται στο stack ένας "κρυφός" δείκτης που δείχνει σε μια προσωρινή περιοχή μνήμης. Η συνάρτηση πρέπει να τοποθετήσει την τιμή στην περιοχή αυτή και να επιστρέψει το δείκτη (near δείκτες στον AX, far δείκτες στους DX:AX).

7. Έξοδος από υπορουτίνα

Ακολουθούμε τα παρακάτω βήματα:

1. Καθένας από τους καταχωρητές SS, DS, SI και DI που έχουμε σώσει στη στοίβα πρέπει να αποκατασταθεί βγάζοντας την παλιά του τιμή από την στοίβα με την ανάποδη σειρά από αυτή που είχε μπει (εντολή POP).
2. Αν χώρος τοπικών δεδομένων έχει δεσμευτεί αποδεσμεύεται με την εντολή MOV SP,BP.
3. Αποκαθιστούμε τον BP με POP BP. Αυτό είναι πάντα απαραίτητο.
4. Τέλος επιστρέφουμε στο πρόγραμμα που καλεί την ρουτίνα assembly με:
   RET αν πρόκειται για σύνδεση με τη γλώσσα C ως συνέπεια του γεγονότος ότι στην C οι παράμετροι αφαιρούνται από τη στοίβα από την καλούσα διεργασία αμέσως μετά την επιστροφή της καλούμενης.

Πώς καλούμε assembly από C

Οι οδηγίες αυτές είναι σε ένα μεγάλο μέρος ειδικά γραμμένες για την TURBO C. Το γενικό σχεδιάγραμμα της υπορουτίνας assembly που γράφουμε, είναι:

```
<code>    SEGMENT BYTE PUBLIC 'CODE'
          ASSUME CS:<code>,DS:<dseg>
          < εντολές κώδικα >
<code>    ENDS
<dseg>    GROUP -DATA,-BSS
<data>    SEGMENT WORD PUBLIC 'DATA'
          < αρχικοποιημένα δεδομένα >
<data>    ENDS
_BSS      SEGMENT WORD PUBLIC 'BSS'
          < μη αρχικοποιημένα δεδομένα >
_BSS      ENDS
END
```

Τα αναγνωριστικά <text>, <data>, <dseg> αντικαθίστανται από συγκεκριμένα ονόματα πού τα βλέπουμε στον πίνακα 1. Ας σημειωθεί ότι στο μοντέλο μνήμης huge της C δεν υπάρχει τμήμα _BSS οπότε ο ορισμός GROUP απαλείφεται. Γενικά το _BSS είναι προαιρετικό.

Πίνακας 1. Αναγνωριστικά και μοντέλα μνήμης της Turbo C

Μοντέλο μνήμης	**Αναγνωριστικό που αντικαθίσταται**
Tiny, small	<code>=_TEXT <data>=_DATA <dseg>=_DGROUP
Compact	<code>=_TEXT <data>=_DATA <dseg>=_DGROUP

Medium	<code>=_filename_TEXT <data>=_DATA <dseg>=_DGROUP
Large	<code>=_filename_TEXT <data>=_DATA <dseg>=_DGROUP
Huge	<code>=_filename_TEXT <data>=_filename_DATA <dseg>=_filename_DATA

όπου filename είναι το όνομα του module.

Ο καλύτερος τρόπος για να δημιουργήσετε ένα γενικό υποδειγματικό διάγραμμα μιας assembly υπορουτίνας είναι να μεταφράσετε ένα άδειο πρόγραμμα της C (έστω το filename.c) σε assembly γλώσσα (filename.asm), χρησιμοποιώντας τον command line compiler της turbo C, δίνοντας τη διαταγή TCC -S filename.c, το οποίο μπορείτε κατόπιν να μελετήσετε.

Όλες οι μεταβλητές ονομάτων στην υπορουτίνα assembly πρέπει να αρχίζουν με underscore (_) αφού αυτό περιμένει και ο Compiler της C.

Παράδειγμα:

Έστω ότι γράφουμε ένα module που περιέχει τις ακέραιες συναρτήσεις (functions) max και min και τις ακέραιες μεταβλητές MAXINT, lastmax και lastmin. Τότε στο τμήμα κώδικα δηλώνουμε:

```
_TEXT       SEGMENT BYTE PUBLIC 'CODE'
            ASSUME CS:_TEXT, DS:_DATA
            PUBLIC _max, _min
            .
            .
            .
            περιέχεται ορισμός των _max, _min
            .
            .
_TEXT       ENDS
```

Στο τμήμα δεδομένων δηλώνουμε:

```
_DATA       SEGMENT WORD PUBLIC 'DATA'
_MAXINT     DW 32677
_latsmin    DW 0
_lastmax    DW 0
_DATA       ENDS
```

Υποθέτουμε ότι στο παραπάνω παράδειγμα χρησιμοποιείται μοντέλο μνήμης small.

Προσοχή:

Σύμφωνα με τη σύμβαση της C, για το πέρασμα παραμέτρων (C calling convention), κατά την κλήση μιας υπορουτίνας οι παράμετροι σώζονται στη στοίβα αρχίζοντας από τη δεξιότερη.

Π.χ. έστω ότι η συνάρτηση proc η οποία έχει δηλωθεί ως:

void proc(int arg1, int arg2, long arg3);

καλείται από το πρόγραμμα main ως εξής:

```
main()
{
        int i, j;
        long k;
        ...

        i = 6; j =3; k = 0x230ffd;
        proc(i, j, k);
        ...
}
```

Κατά την κλήση της συνάρτησης proc οι παράμετροι αποθηκεύονται στη στοίβα με την σειρά arg3, arg2, arg1 και η στοίβα πριν το σώσιμο της διεύθυνσης επιστροφής θα έχει την παρακάτω μορφή:

SP	0006	; i = arg1
SP + 02	0003	; j = arg2
SP + 04	0ffd	
SP + 06	0023	; k = arg3

Υπενθύμιση: Στον 8086 η στοίβα επεκτείνεται από ψηλότερες θέσεις μνήμης προς τη χαμηλότερες και έτσι το i αν και βρίσκεται στην κορυφή της στοίβας, βρίσκεται σε χαμηλότερη θέση μνήμης από τις άλλες παραμέτρους. Παρόλο που η πιο πάνω

σύμβαση φαίνεται παράξενη με την πρώτη ματιά, είναι όμως αναγκαία αφού η C επιτρέπει το πέρασμα μεταβλητού αριθμού παραμέτρων. Π.χ. η συνάρτηση:

int extern max(int counter, int arg1, int arg2, ...);

Η μεταβλητή counter που δηλώνει τον αριθμό των παραμέτρων σύμφωνα με την πιο πάνω σύμβαση, είναι η πρώτη παράμετρος που βρίσκεται κάτω από τη διεύθυνση επιστροφής. Με αυτό τον τρόπο η καλούμενη μπορεί να βρει τον αριθμό των παραμέτρων που περάστηκαν κοιτώντας την τιμή της πρώτης παραμέτρου.

Για να συνδέσουμε το αρχείο file1.c που χρησιμοποιεί μια υπορουτίνα γραμμένη σε γλώσσα assembly η οποία είναι δηλωμένη στο αρχείο file2.asm, πρώτα μεταφράζουμε το file2.asm σε file2.obj με τον MASM

MASM -mx file2.asm;

και στη συνέχεια δίνουμε την εντολή:

tcc file1 file2.obj

Inline εντολές μηχανής σε γλώσσες υψηλού επιπέδου

Πολλοί compilers γλωσσών υψηλού επιπέδου επιτρέπουν να εισάγουμε μέσα στον source κώδικα τους σύντομες υπορουτίνες assembly. Τέτοια είναι η Turbo C.

Inline εντολές assembly στη Turbo C

Μια inline εντολή assembly στην Turbo C έχει τη γενική μορφή:

asm <opcode> <operands> <; or newline>

όπου: <opcode> είναι μια νόμιμη εντολή assembly για τον 8086

<operands> τα κατάλληλα τελούμενα για τον <opcode> που μπορεί να είναι σταθερές, μεταβλητές ή labels της C. Π.χ. η ρουτίνα που ακολουθεί υπολογίζει τον ελάχιστο μεταξύ δύο αριθμών:

```
int min (int arg1, int arg2)
{
        asm mov ax,arg1
        asm cmp ax,arg2
        asm jle end
        asm mov ax,arg2
        end: return (_AX)
}
```

Όπως φαίνεται από το πιο πάνω παράδειγμα η Turbo C επιτρέπει κατευθείαν προσπέλαση στους καταχωρητές του 8086 μέσω των ψευδομεταβλητών που φαίνονται στον παρακάτω πίνακα.

Ψευδομεταβλητή	Τύπος	Καταχωρητής
_AX	unsigned int	AX
_AL	unsigned char	AL
_AH	unsigned char	AH
_BX	unsigned int	BX
_BL	unsigned char	BL
_BH	unsigned char	BH
_CX	unsigned int	CX
_CL	unsigned char	CL
_CH	unsigned char	CH
_DX	unsigned int	DX
_DL	unsigned char	DL
_DH	unsigned char	DH
_CS	unsigned int	CS
_DS	unsigned int	DS
_SS	unsigned int	SS
_ES	unsigned int	ES
_SP	unsigned int	SP
_BP	unsigned int	BP
_DI	unsigned int	DI
_SI	unsigned int	SI

Γενικά παραδείγματα

Στην παράγραφο αυτή θα παρουσιάσουμε ένα απλό πλην όμως βασικό παράδειγμα interfacing ρουτινών με γλώσσα υψηλού επιπέδου. Το πρόγραμμα είναι γραμμένο σε δυο γλώσσες υψηλού επιπέδου (Pascal, C) ενώ χρησιμοποιούνται οι ρουτίνες σε assembly κατάλληλα τροποποιημένες ανάλογα με τις συμβάσεις που ισχύουν για το interfacing κάθε γλώσσας με υπορουτίνες assembly.

Το πρόγραμμα σε γλώσσα υψηλού επιπέδου ζητάει από το χρήστη να του δώσει τον αριθμό των χαρακτήρων που επιθυμεί να πληκτρολογήσει. Στη συνέχεια

του ζητάει να εισάγει τους χαρακτήρες και καλεί την πρώτη ρουτίνα assembly, η οποία διαβάζει τους χαρακτήρες ,τους τοποθετεί σε ένα string και ψάχνει να βρει τον αριθμό των φωνηέντων, τον οποίο και επιστρέφει. Στην συνέχεια δηλώνει πως θα μετατρέψει τους κεφαλαίους χαρακτήρες σε "μικρούς" και καλεί τη δεύτερη ρουτίνα assembly. Τελικά τυπώνει τη νέα συμβολοσειρά, δηλώνει πόσα φωνήεντα βρέθηκαν και ρωτάει το χρήστη αν θέλει να επαναλάβει τη διαδικασία.

Ακολουθεί το πρόγραμμα υψηλού επιπέδου σε C:

C

```
#include <stdio.h>
#include <conio.h>

extern char rscrn(int num,char *bfr);
extern void readn(int num,char *input,char *output);

main()
 {
char find[80],m[80];
int k;
char x,ch;
char buf[255];

    do
      {
      clrscr();
      printf("\n");
      printf("HOW MANY CHARACTERS SHALL I READ?:\n")'
      gets(buf);sscanf(buf,"%d",&k);
      if(k) {
         printf("YOU MAY NOW ENTER THE LINE:\n");
         x=rscrn(k,find);
         printf("\n");
         printf("\n");
         printf(("IN LOWER CASE:");
         readn(k,find,m);
         printf((m);
         if(x)
             {
```

```
            if (x==1) printf("THERE WAS 1 VOWEL FOUND.\n");
                    else
              printf("THERE WERE %d VOWELS COUNTED.\n");
             }
         else
                    printf("THERE WERE NO VOWELS FOUND.\n");
          printf("\n");
            printf("WOULD YOU LIKE TO DO IT AGAIN? (Y/N):");
         gets(buf);ch=buf[0];
       }
      else
           ch='Y'
       if (ch=='n') ch='N'
     }
while(ch!='N');
}
```

Η ρουτίνα RSCRN για C είναι:

C

```
_DATA SEGMENT WORD PUBLIC 'DATA'
_DATA ENDS

_BSS SEGMENT WORD PUBLIC 'BSS'
_BSS ENDS

DGROUP GROUP _DATA,_BSS

_TEXT SEGMENT BYTE PUBLIC 'CODE'
    PUBLIC _rscrn
    ASSUME CS:_TEXT,DS:DGROUP,SS:DGROUP
                            ; Χρησιμοποιήσαμε small model δηλαδή near procedures
                            ; και near δείκτες οπότε έχουμε:
NUM_CHARS  EQU    [BP+4]
RET_STRING  EQU    [BP+6]
                  ;η Turbo C δίνει ως παράμετρο στο small model, μόνο το offset
```

```
                ; του πίνακα αν είχαμε large model (όλοι οι δείκτες far) θα είχαμε
                        ;NUM_CHARS EQU    [BP+6]
                        ;RET_STRING EQU   DWORD PTR [BP+8]

_rscrn  PROC NEAR                   ;FAR αν είχαμε large model
        PUSH BP
        MOV BP,SP
        PUSH SI
        PUSH DI
        PUSH ES
          MOV ES,DS
        MOV  DI,RET_STRING          ;DI -> string που θα
                                    ;  αποθηκεύσουμε τους χαρακτήρες
        MOV CX,NUM_CHARS            ;CX <- αριθμός χαρακτήρων
                                    ; για ανάγνωση
        MOV AH,0FH
        INT 10H                     ;BH <- τρέχουσα σελίδα οθόνης
        MOV DH,4                    ;DH <- γραμμή
OKAY:
        MOV DL,0                    ;DL <- στήλη
        MOV AH,2
        INT 10H                     ;Μετακίνηση του δρομέα
                                    ;(σελίδα, γραμμή, στήλη)
        MOV BL,0                    ;αριθμός φωνηέντων
READEM:
          MOV AH,1
        INT 21H                     ;ανάγνωση χαρακτήρα
        PUSH AX
        CMP AL,30H
        JL GO                       ;αν <'0' τσέκαρε
        CMP AL,39H
        JG GO                       ;αν >'9' τσέκαρε
        JMP LAST
GO:
        AND AL,0FH                  ;μηδενίζουμε τα 4 ψηλά bit του AL
        CMP AL,01H                  ;αν AL=1 τότε φωνήεν
        JNE NEXT
PROCCEED:                           ;εδώ έρχεται ο έλεγχος αν
```

```
                                    ; έχουμε φωνήεν
        INC BL                      ;αυξάνουμε τον αριθμό των φωνηέντων
        JMP LAST
NEXT:
        CMP AL,05H                  ;αν AL=5 τότε φωνήεν
        JNE NEXT1
        JMP PROCCEED
NEXT1:
        CMP AL,09H                  ;αν AL=9 τότε φωνήεν
        JNE NEXT2
        JMP PROCCEED
NEXT2:
        CMP AL,0FH                  ;αν AL=15 τότε φωνήεν
        JNE LAST
        JMP PROCCEED
LAST:                               ;εδώ έρχεται ο έλεγχος για να
                                    ; πάμε στο επόμενο γράμμα
        POP AX              ;αφού βάλουμε στο string το
                                    ;τρέχον γράμμα
        STOSB
        LOOP READEM                 ;διαβάζουμε τον επόμενο χαρακτήρα
EXIT:  POP ES
        POP DI
        POP SI
        POP BP
        MOV AL,BL
        RET
_rscrn   ENDP
_TEXT    ENDS
        END
```

Ο κώδικας για τη ρουτίνα READN είναι:

C:

```
```

```
_DATA SEGMENT WORD PUBLIC 'DATA'
_DATA ENDS

_BSS SEGMENT WORD PUBLIC 'BSS'
_BSS ENDS

_TEXT SEGMENT BYTE PUBLIC 'CODE'
   PUBLIC _readn
   ASSUME CS:_TEXT,DS:DGROUP,SS:DGROUP

        ;έχουμε small model άρα near δείκτες
NUM_CHAR EQU [BP+4]
INPUT_STRING EQU [BP+6]
RET_STRING EQU [BP+8]
        ;η Turbo C στο small model ως παράμετρο δίνει μόνο
        ; το offset του πίνακα αν είχαμε far model τότε
;NUM_CHAR EQU [BP+6]
;INPUT_STRING EQU DWORD PTR [BP+8]
;RET_STRING EQU DWORD PTR [BP+12]

_readn    PROC NEAR
        PUSH BP
        MOV BP,SP
          PUSH DI
          PUSH SI
          PUSH ES
        CLD
        MOV SI,INPUT_STRING        ;DS:SI -> INPUT_STRING[0]
         MOV ES,DS
        MOV DI,RET_STRING ;ES:DI -> RET_STRING[0]
        MOV CX,NUM_CHAR            ;CX -> αριθμός χαρακτήρων
        MOV BYTE PTR ES:[DI],CL
                        ;RET_STRING[0] = αριθμός χαρακτήρων
NEXT:
        LODSB
                        ;AL <- χαρακτήρας που δείχνει ο DS:SI
        CMP AL,41H              ;αν AL <'A' επόμενος χαρακτήρας
        JL GO
```

```
        CMP AL,5AH          ;αν AL >'Z' επόμενος χαρακτήρας
        JG GO
        ADD AL,20H          ;αλλιώς τον κάνουμε κεφαλαίο
 GO:
        STOSB          ;αποθήκευση του χαρακτήρα στο RET_STRING
        LOOP NEXT
         XOR AX,AX
         STOSB              ;  τα string στη C τελειώνουν με '/0'
EXIT:    POP ES
         POP SI
         POP DI
        POP BP
        RET                 ;καθαρίζουμε το stack (αφήνοντας όμως τον
                            ;κρυφό δείκτη στο stack)
_readn ENDP
_TEXT   ENDS
        END
```

Ιδιαίτερη προσοχή πρέπει να δοθεί στο ότι κατά τη διάρκεια της μετάφρασης θα πρέπει να δώσουμε στο MASM την επιλογή -Ml (ή -Mx) ώστε να μην αλλάξει τα γράμματα των ονομάτων των συναρτήσεων σε κεφαλαία, οπότε δεν θα μπορεί να τα εντοπίσει στη συνέχεια ο linker της C.

2. Ασύγχρονη επικοινωνία μέσω software σε προσωπικό υπολογιστή

Η ασύγχρονη επικοινωνία σε ένα μικροϋπολογιστικό σύστημα (PC) χάρη στο DOS και στο BIOS είναι σχετικά απλή και δεν απαιτεί δουλειά σε επίπεδο hardware. Από εδώ και μέχρι το τέλος αυτού του κεφαλαίου η επικοινωνία θα νοείται μεταξύ δύο συσκευών που έχουν η καθεμία μια θύρα RS-232C, π.χ.. δύο PC μεταξύ τους.

Παρακάτω θα αναπτυχθούν δύο διαφορετικές μέθοδοι, η πρώτη είναι η μέθοδος με τη χρήση των ρουτινών του BIOS (polling) και μια δεύτερη στην οποία γίνεται απευθείας προγραμματισμός του controller σειριακής επικοινωνίας 8250 που διαθέτει το PC και βασίζεται στις διακοπές. Και με τις δύο μεθόδους μπορούμε να μετατρέψουμε ένα PC σε τερματικό, δηλαδή όποιος χαρακτήρας πληκτρολογείται θα στέλνεται στη θύρα RS-232C (και θα τυπώνεται στην οθόνη του PC αν θέλετε), και όποιος χαρακτήρας έρχεται από την θύρα RS-232C θα τυπώνεται στην οθόνη του PC.

2.1 Χρήση των ρουτινών του BIOS

Αυτή η μέθοδος είναι πολύ πιο απλή από την δεύτερη. Όπως είπαμε και στην αρχή, υπάρχουν μερικές ρουτίνες του DOS και του BIOS τις οποίες θα εκμεταλλευτούμε για να πετύχουμε εύκολα το σκοπό μας. Έτσι λοιπόν θα κοιτάξουμε πρώτα τον αλγόριθμο εξομοίωσης του PC σε τερματικό, για να δούμε ποιες ρουτίνες χρειαζόμαστε για να τον υλοποιήσουμε.

```
INIT COM1
REPEAT
  IF CHARACTER READY UART THEN
  READ CHARACTER
  SEND TO CRT
  IF KEYPRESSED ON PC KEYBOARD THEN
  READ KEY
  SEND TO SERIAL PORT
UNTIL FOREVER
```

Όπου: INIT COM1 είναι η αρχικοποίηση της πόρτας COM1 του PC.
UART είναι το ολοκληρωμένο που σε επίπεδο hardware υλοποιεί την ασύγχρονη επικοινωνία.
CRT είναι η οθόνη.

Οι σχετικές ρουτίνες του DOS είναι οι ακόλουθες:

Function call 2: Ο χαρακτήρας που βρίσκεται στον καταχωρητή DL στέλνεται στην οθόνη και ο κέρσορας προωθείται μία θέση.

Function call 3: Περιμένει να λάβει ένα χαρακτήρα από τη σειριακή πόρτα και τον τοποθετεί στον καταχωρητή AL.

Function call 4: Ο χαρακτήρας που βρίσκεται στον καταχωρητή DL στέλνεται στην σειριακή πόρτα.

Function call 8: Περιμένει να πατηθεί ένα πλήκτρο στο πληκτρολόγιο και επιστρέφει τον ASCII κώδικά του στον καταχωρητή AL.

Οι ρουτίνες 2 και 4 είναι ότι χρειαζόμαστε για να στείλουμε κάποιο χαρακτήρα στην οθόνη και για να στείλουμε ένα χαρακτήρα από το τερματικό μας αντίστοιχα. Οι άλλες δύο όμως ρουτίνες περιμένουν και οι δύο για είσοδο και έτσι θα μπλοκάρουν το πρόγραμμα μέχρι να πατηθεί κάποιο πλήκτρο ή να φτάσει κάποιος χαρακτήρας στην σειριακή. Έτσι στην ουσία καταργείται η φιλοσοφία του polling.

Έτσι μας χρειάζονται διαδικασίες που θα επιτρέπουν στη ρουτίνα που θα υλοποιεί το polling να μπορεί να κοιτά και στο πληκτρολόγιο και στη θύρα εισόδου χωρίς να κολλάει μέχρι να υπάρξει είσοδος στην συσκευή που ελέγχεται.

Ας δούμε λοιπόν ποια λύση μας προσφέρει το BIOS με τη ρουτίνα INT 14H. Αυτή η ρουτίνα θα εκτελέσει μια από τις τέσσερις ρουτίνες ανάλογα με την τιμή του καταχωρητή AH:

- Αν ο καταχωρητής AH είναι ίσος με μηδέν όταν κληθεί η ρουτίνα αυτή, τότε το byte που περιέχεται στον καταχωρητή AL χρησιμοποιείται για να αρχικοποιήσει τη σειριακή πόρτα.
- Αν ο καταχωρητής AH είναι ίσος με 1, τότε ο χαρακτήρας που βρίσκεται στον καταχωρητή AL θα σταλεί στην πόρτα εξόδου.
- Αν AH=2, τότε θα διαβαστεί ένας χαρακτήρας από τη σειριακή πόρτα και θα τοποθετηθεί στον AL.
- Τέλος αν AH=3 όταν κληθεί η ρουτίνα του BIOS, τότε η κατάσταση της σειριακής πόρτας γράφεται στους AH και AL.

Με την πρώτη περίπτωση (AH=0) λύνουμε το πρόβλημα της αρχικοποίησης. Με την τελευταία (AH=3) λύνουμε το μεγαλύτερο κομμάτι του

προβλήματος δηλαδή να καθορίσουμε πότε η UART έχει ένα χαρακτήρα έτοιμο να διαβαστεί. Το bit 0 του byte κατάστασης του AH θα γίνει 1 αν η UART περιέχει χαρακτήρα έτοιμο για να διαβαστεί. Αν υπάρχει χαρακτήρας, τότε μπορεί να διαβαστεί και μετά να σταλεί στην οθόνη. Αν δεν υπάρχει, τότε το πρόγραμμα θα πάει να ελέγξει αν έχει πατηθεί κάποιο πλήκτρο στο πληκτρολόγιο.

Το αντίστοιχο πρόβλημα του polling στο πληκτρολόγιο λύνεται από τη ρουτίνα INT 16H του BIOS, η οποία εξετάζει το πληκτρολόγιο και παρέχει 3 δυνατές υπορουτίνες ανάλογα με την τιμή του AH:

- Για AH=0 τοποθετεί τον ASCII χαρακτήρα που πατήθηκε στο πληκτρολόγιο στον AL.
- Για AH=1 θέτει ZF=0 (ZF=Zero Flag) αν πατήθηκε κάποιο πλήκτρο και ο ASCII κώδικας του είναι διαθέσιμος και βρίσκεται στον AL να διαβαστεί.
- Για AH=2 προκαλεί την ρουτίνα να επιστρέψει την κατάσταση μετατόπισης (shift status) στον AL. Τα διάφορα bit αυτού του byte δείχνουν την κατάσταση των πλήκτρων Shift, Alt, Caps Lock κλπ. στο πληκτρολόγιο.

Η κλήση της ρουτίνας INT 16 με AH=1 λύνει το πρόβλημα της παρακολούθησης του πληκτρολογίου χωρίς να περιμένει σε κάποιο loop όπως κάνει η αντίστοιχη ρουτίνα του DOS. Η κατάσταση ZF=1 μπορεί απλώς να ελεγχθεί με την ρουτίνα INT 16 και αν δεν έχει πατηθεί κάποιο πλήκτρο τότε μπορεί να ελεγχθεί κατευθείαν η UART πάλι. Αν έχει πατηθεί κάποιο πλήκτρο, τότε μπορεί να διαβαστεί από τον AL και να σταλεί στη UART.

Όπως αναφέραμε και στην αρχή η μέθοδος polling προσφέρει μια εύκολη λύση στο πρόβλημά μας, πλην όμως, δε μας επιτρέπει να πετύχουμε μεγάλους ρυθμούς μεταφοράς πληροφοριών (όχι πάνω από 600 Baud (1 baud = bit/sec)). Αν προσπαθήσετε να χρησιμοποιήσετε μεγαλύτερους ρυθμούς π.χ. 1200 ή 2400 Baud τότε ο πρώτος χαρακτήρας κάθε γραμμής θα χάνεται. Το πρόβλημα δεν έχει σχέση με την ταχύτητα του 8088 ο οποίος είναι αρκετά γρήγορος και μπορεί να δουλέψει και στα 4800 Baud. Το πρόβλημα βρίσκεται στη ρουτίνα που στέλνει τους χαρακτήρες στην οθόνη. Μετά την αποστολή ενός Carriage return, η οθόνη κυλάει μια γραμμή και για να μη φανεί αυτό το τρεμοσβήσιμο, η νέα γραμμή θα εμφανιστεί με την επόμενη ενημέρωση (update) της οθόνης, που γίνεται σε συχνότητα 50 Hz στις μονοχρωματικές οθόνες. Αυτό σημαίνει ότι μπορεί να χρειαστούν μέχρι 20 ms για να συνεχιστεί το τύπωμα των χαρακτήρων στην οθόνη και έτσι όσοι χαρακτήρες φτάσουν μέσα σε αυτό το χρονικό διάστημα θα χαθούν.

2.2 Απευθείας προγραμματισμός του controller σειριακής επικοινωνίας 8250

Εισαγωγή

Όταν οι ταχύτητες που απαιτούνται για την μεταφορά δεδομένων δεν είναι υπερβολικά υψηλές μπορεί να χρησιμοποιηθεί η σειριακή επικοινωνία. Τα δεδομένα μεταφέρονται σε ομάδες των 7 ή των 8 bits που ονομάζονται λέξεις. Για λόγους συγχρονισμού μεταξύ του δέκτη και του παραλήπτη τοποθετούνται bits συγχρονισμού στην αρχή (starts bits) και στο τέλος (stop bits) κάθε λέξης. Στη σειριακή ασύγχρονη επικοινωνία ο αριθμός των bits ανά δευτερόλεπτο ονομάζεται baud rate. Για την ελαχιστοποίηση του σφάλματος λόγω χρονικής ολίσθησης και ο δέκτης και ο πομπός χρησιμοποιούν εσωτερικά χρονόμετρα με συχνότητα 16 φορές το baud rate. Όταν ανιχνευθεί η οδηγούσα ακμή του start bit ο δέκτης δειγματοληπτεί την αφικνούμενη κυματομορφή κάθε 16 περιόδους αρχίζοντας από την όγδοη περίοδο μετά την οδηγούσα ακμή του start bit. Αυτό εξασφαλίζει ότι η κυματομορφή δειγματοληπτείται περίπου στο μέσο κάθε baud περιόδου και επομένως το σύστημα επικοινωνίας μπορεί να αντέξει μικρές χρονικές ολισθήσεις και διαφορές ανάμεσα στις συχνότητες των χρονομέτρων του πομπού και του δέκτη.

Μια ειδική ψηφίδα καλούμενη UART (Universal Asynchronous Receiver Transmitter) πραγματοποιεί σειριακή ασύγχρονη μεταφορά και μάλιστα και προς τις δύο κατευθύνσεις (full dublex). Η ψηφίδα αυτή μετατρέπει τα δεδομένα που πρόκειται να σταλούν, από παράλληλη σε σειριακή μορφή, προσθέτει το parity, το start και τα stop bits σε κάθε λέξη και εκτελεί την αντίστροφη διαδικασία στα σειριακά δεδομένα που λαμβάνει. Επειδή η μεταφορά δεδομένων σειριακά δεν μπορεί να γίνει χωρίς λάθη (κυρίως εξαιτίας θορύβου) πρέπει η έξοδος μιας UART να μετατραπεί σε κατάλληλη μορφή. Η πιο διαδεδομένη μορφή είναι η RS232, όπου το λογικό 0 αντιστοιχεί σε επίπεδο τάσης από +3V έως +15V και το λογικό 1 σε επίπεδο τάσης από -3V έως -15V.

Στο PC (Personal Computer) για την επίτευξη σειριακής επικοινωνίας χρησιμοποιείται η UART 8250 και για τη μεταφορά των σειριακών δεδομένων η μορφή RS232. Ακολουθεί μια σύντομη περιγραφή της 8250 και στη συνέχεια περιγράφεται ο προγραμματισμός της σειριακής επικοινωνίας για το PC.

Περιγραφή των ακίδων της UART 8250

D_0-D_7:	Γραμμές δεδομένων
CS_0, CS_1, CS_2:	Γραμμές επιλογής ψηφίδας (chip select)
DISTR, DISTR:	(Data Input Strobe) Όταν ενεργοποιηθούν μπορεί η CPU να διαβάσει δεδομένα από την UART.
DOSTR, DOSTR:	(Data Output Strobe) Όταν ενεργοποιηθούν μπορεί η CPU να γράψει δεδομένα στη UART.
A_0, A_1, A_2:	Γραμμές διευθύνσεων που προσδιορίζουν τον εσωτερικό καταχωρητή της UART που θα προσπελαστεί.
ADS:	Μανταλώνει τις γραμμές διευθύνσεων και τις γραμμές επιλογής.
INTR:	Στέλνει σήμα διακοπής προς τη CPU.
RI:	(Ring Indicator) Επιτρέπει στον υπολογιστή να απαντήσει σε ένα τηλεφώνημα.
RLSD:	(Received Line Signal Detect) Μέσω αυτής το modem δηλώνει στη UART ότι ανιχνεύθηκε κάποια κυματομορφή στην τηλεφωνική γραμμή.
DTR, RTS:	(Data Terminal Ready και Request To Send) Με τις γραμμές αυτές το τερματικό δηλώνει στο modem ότι είναι έτοιμο και επιθυμεί την αποστολή δεδομένων. Οι γραμμές αυτές είναι συνδεδεμένες με τις γραμμές DSR (Data Send Ready) και CTS (Clear To Send) του modem. Ομοίως οι DTR και RTS του modem συνδέονται με τις DTR και RTS του τερματικού.
SIN:	(Serial Input) Είσοδος σειριακών δεδομένων.
SOUT:	(Serial Output) Έξοδος σειριακών δεδομένων.
OUT1, OUT2:	Δύο ακόμα έξοδοι. Στο PC χρησιμοποιείται μόνο η OUT1 για την ενεργοποίηση των απαιτήσεων διακοπών στη γραμμή INTR.
XTAL1, XTAL2:	Συνδέεται ένας κρύσταλλος που ελέγχει έναν εσωτερικό ταλαντωτή. Επίσης μπορεί να συνδεθεί ένα εξωτερικό χρονόμετρο στην XTAL1.
BAUDOUT:	Αποδίδει συχνότητα (16 χ baud rate).
RCLCK:	(Receiver CloCK) Είσοδος χρονομέτρου παραλήπτη. Στο PC στην είσοδο RCLCK της 8250 συνδέεται η BAUDOUT.
MR:	(active high Master Reset)

Από τις ακίδες αυτές σε μια απλή σειριακή μεταφορά δεδομένων μπορούμε να περιοριστούμε στις παρακάτω τρεις: SIN, SOUT και GROUND.

Προγραμματισμός της UART 8250 στο PC

Στον ακόλουθο πίνακα περιέχονται οι καταχωρητές της 8250 που πρέπει να προγραμματιστούν για την επίτευξη σειριακής επικοινωνίας καθώς και οι διευθύνσεις που αντιστοιχούν σε πόρτες I/O στο PC.

ΔΙΕΥΘΥΝΣΗ I/O		**Καταχωρητής**	**Κατάσταση του**
COM 1	**COM 2**	**που επιλέγεται**	**DLAB**
3F8	2F8	Απομονωτής εξόδου	DLAB = 0
3F8	2F8	Απομονωτής εισόδου	DLAB = 0
3F8	2F8	Διαιρέτης του baud rate LSB	DLAB = 1
3F9	2F9	Διαιρέτης του baud rate MSB	DLAB = 1
3F9	2F9	Ενεργοποίηση διακοπών	
3FA	2FA	Αναγνώριση διακοπών	
3FB	2FB	Καταχωρητής ελέγχου γραμμής	
3FC	2FC	Καταχωρητής ελέγχου modem	
3FD	2FD	Καταχωρητής κατάστασης γραμμής	
3FE	2FE	Καταχωρητής κατάστασης modem	

Τα bits διευθύνσεων A_0, A_1, A_2 επιλέγουν τον συγκεκριμένο καταχωρητή σε συνδυασμό με το DLAB (Divisor Latch Access Bit), το 7^0 bit του latch διαιρέτη. Ακολουθεί σύντομη περιγραφή της λειτουργίας των παραπάνω καταχωρητών.

Απομονωτής εξόδου: Σε αυτόν φορτώνεται μια λέξη παράλληλα για να μεταφερθεί σειριακά.

Απομονωτής εισόδου: Στον απομονωτή εισόδου συγκεντρώνεται μια λέξη, που φτάνει σειριακά, για να διαβαστεί παράλληλα.

Διαιρέτης του baud rate LSB και MSB: Η UART περιέχει μια προγραμματιζόμενη γεννήτρια baud rate, που παίρνει τη συχνότητα του χρονομέτρου (1.8432 MHz) και τη διαιρεί με έναν οποιοδήποτε διαιρέτη από 1 έως 2^{16}-1. Η συχνότητα εξόδου της γεννήτριας είναι 16×baud rate. Ο διαιρέτης δίνεται από τον τύπο:

$$\text{διαιρέτης} = 1.8432 / (16 \times \text{baud rate})$$

Ο διαιρέτης αυτός αποθηκεύεται σε δύο latches των 8 bit που ονομάζονται καταχωρητής διαιρέτη με τα λιγότερο σημαντικά ψηφία (LSB) και καταχωρητής διαιρέτη με τα περισσότερο σημαντικά ψηφία (MSB).

Καταχωρητής ελέγχου γραμμής: Αποτελείται από 8 bit από τα οποία τα δύο πρώτα ορίζουν το μήκος λέξης. Για παράδειγμα αν είναι bit0=1 και bit1=1 ορίζεται λέξη των 8 bit. Το επόμενο bit ορίζει τον αριθμό των stop bit, που για bit2=0 είναι ένα. Τα bit3, bit4, bit5 ενεργοποιούν τον μηχανισμό της ισοτιμίας. Το bit6 λέγεται bit απομόνωσης. Όταν είναι 1 η SOUT τίθεται και παραμένει σε λογικό 0 ανεξάρτητα από τη δραστηριότητα της επικοινωνίας. Τέλος το bit7 είναι το bit προσπέλασης των καταχωρητών διαιρέτη το DLAB.

Στη συνέχεια δίνονται τα βασικά στοιχεία για την επίτευξη της απλούστερης σειριακής επικοινωνίας. Σε επίπεδο hardware αρκεί η χρησιμοποίηση μόνο τριών συρμάτων (receiver, transmitter και ground) ενώ σε επίπεδο software δεν χρησιμοποιούνται οι καταχωρητές κατάστασης και ελέγχου modem.

Με βάση τα παραπάνω, η ρουτίνα που δίνει αρχικές τιμές στη UART έχει ως εξής:

```
open_RS232      proc near
                jmp start
BAUD_RATE       LABEL WORD              ; Διαιρέτης για το ακόλουθο baud rate
                DW 1047                 ; 110 baud rate
                DW 768                  ; 150 baud rate
                DW 384                  ; 300 baud rate
                DW 192                  ; 600 baud rate
                DW 96                   ; 1200 baud rate
                DW 48                   ; 2400 baud rate
                DW 24                   ; 4800 baud rate
                DW 12                   ; 9600 baud rate
START:          STI                     ; Set interrupt flag
                                        ; ΑΡΧΙΚΕΣ ΤΙΜΕΣ ΣΤΗΝ RS232
                MOV AH,AL               ; Save in it parameters in AH
                MOV DX,3FBH             ; Δείξε στον καταχωρητή ελέγχου  γραμμής
                MOV AL,80H
                OUT DX,AL
                                ; ΚΑΘΟΡΙΣΜΟΣ ΔΙΑΙΡΕΤΗ
                MOV DL,AH               ; Μετέφερε τις παραμέτρους στον DL
                MOV CL,4
```

```
                ROL DL,CL
                AND DX,0EH
                MOV DI,OFFSET BAUD_RATE
                ADD DI,DX           ; Δείξε στα σημαντικά ψηφία του διαιρέτη
                MOV DX,3F9          ; Δείξε στον καταχωρητή διαιρέτη MSB
                MOV AL,CS:[DI]+1    ; Πάρε τα σημαντικά ψηφία του διαιρέτη
                OUT DX,AL           ; Τοποθέτησε τα στον καταχωρητή διαιρέτη
                                      MSB
                MOV DX,3F8          ; Δείξε στον καταχωρητή διαιρέτη LSB
                MOV AL,CS:[DI]      ; Πάρε τα λιγότερο σημαντικά bits του διαιρέτη
                OUT DX,AL           ; Τοποθέτησε τα στον καταχωρητή διαιρέτη
                                      LSB
                MOV DX,3FBH         ; Διεύθυνση του καταχωρητή ελέγχου γραμμής
                MOV AL,AH           ; Βάλε τις παραμέτρους πίσω στον AL
                AND AL,01FH         ; Παράκαμψε τα bits που δηλώνουν το baud rate
                OUT DX,AL           ; Γράψε στον καταχωρητή ελέγχου γραμμής τις
                                      παραμέτρους
                MOV DX,3F9          ; Δείξε καταχωρητή ενεργοποίησης διακοπών
                MOV AL,0H
                OUT DX,AL           ; Απενεργοποίησε τις διακοπές
                RET
OPEN_RS232      ENDP
```

Οι παράμετροι που αρχικοποιούν την RS232 τοποθετούνται στον AL πριν την κλήση της OPEN_RS232. Η σημασία των bits του AL είναι η ακόλουθη:

bits 7, 6, 5 = οι τιμές από 000 μέχρι 111 δίνουν baud rates: 110, 150, 300, 600, 1200, 2400, 4800, 9600 αντίστοιχα.
bit 4 = άρτια ισοτιμία με 1, περιττή με 0.
bit 3 = ενεργοποίηση ισοτιμίας (1 ενεργοποίηση, 0 απενεργοποίηση).
bit 2 = αριθμός των stop bits (0 δίνει ένα stop bit και 1 δίνει 2 stop bits).
bits 1, 0 = μήκος λέξης (10 για μήκος 7 bits και 11 για 8 bits).

Παράδειγμα:

Για AL=0E3H=11100011 έχουμε:
μήκος λέξης 8 bits, όχι ισοτιμία, 1 stop bit και baud-rate=9600.

Ο καταχωρητής καταστάσεως γραμμής (πόρτα 3FD) μας πληροφορεί για την κατάσταση των γραμμών. Η σημασία των bit του καταχωρητή αυτού είναι η ακόλουθη:

bit 0 ή DR (Data Received):	αν είναι 1 σημαίνει ότι ένας χαρακτήρας έχει ληφθεί και μπορεί να διαβαστεί από τη CPU.
bit 1 ή OE (Overrun Error):	αν είναι 1 δηλώνει ότι τα δεδομένα στον απομονωτή εισόδου δεν διαβάστηκαν από τη CPU πριν έλθει ο νέος χαρακτήρας που κατέστρεψε τον προηγούμενο.
bit 2 ή PE (Parity Error):	αν είναι 1 δηλώνει ότι ο χαρακτήρας που λάβαμε δεν έχει τη σωστή ισοτιμία.
bit 3 ή FE (Framing Error):	αν είναι 1 δηλώνει ότι ο χαρακτήρας που λάβαμε δεν έχει έγκυρο stop bit.
bit 4 ή BI (Break Interrupt):	λαμβάνει την τιμή 1 κάθε φορά που τα λαμβανόμενα δεδομένα κρατούνται σε λογικό 0 για χρόνο περισσότερο από τον χρόνο μεταφοράς μιας λέξης. Δηλαδή τον συνολικό χρόνο των start bit + baud rate + parity + stop bits.
bit 5 ή THRE (Transmitter Holding Register Empty):	δηλώνει ότι ο καταχωρητής εξόδου είναι κενός. Το THRE=1 σημαίνει ότι ο τελευταίος χαρακτήρας που στάλθηκε έχει ήδη μεταφερθεί και άρα μπορούμε να στείλουμε και άλλον χαρακτήρα.
bit 6 ή TSRE (Trasmitter Shift Register Empty):	όταν η τιμή του είναι 1 σημαίνει ότι δύο χαρακτήρες μπορούν να σταλούν, ένας που πηγαίνει στον καταχωρητή ολισθήσεως και στέλνεται στη γραμμή SOUT και ένας που περιμένει τη σειρά του στον καταχωρητή εισόδου.

Ακολουθούν δύο υπορουτίνες:

- Η πρώτη στέλνει έναν χαρακτήρα στην σειριακή έξοδο και διαβάζει ένα χαρακτήρα, αν έχει ληφθεί κάποιος, από τη σειριακή είσοδο και τον επιστρέφει στον AL. Αν κανένας χαρακτήρας δεν έχει ληφθεί, ο AL επιστρέφει με την τιμή 0.
- Η δεύτερη στέλνει στη σειριακή έξοδο ένα χαρακτήρα ο οποίος πρέπει προηγουμένως να τοποθετηθεί στον AL.

```
RXCH_RS232      PROC NEAR
                MOV             ; Δείξε καταχωρητή καταστάσεως
                DX,3FDH           γραμμής
                IN AL,DX        ; Πάρε την κατάσταση γραμμής στον
                                  AL
                TEST AL,1       ; Υπάρχει χαρακτήρας εκεί ?
                JZ NOTHING      ; Αν όχι κάνε τον AL 0
                SUB DX,5        ; Αν ναι δείξε στον απομονωτή εισόδου
                IN AL,DX        ; και πάρε τον χαρακτήρα στον AL
                JMP EXIT2
NOTHING:        MOV AL,0
EXIT2:          RET
RXCH_RS232      ENDP
```

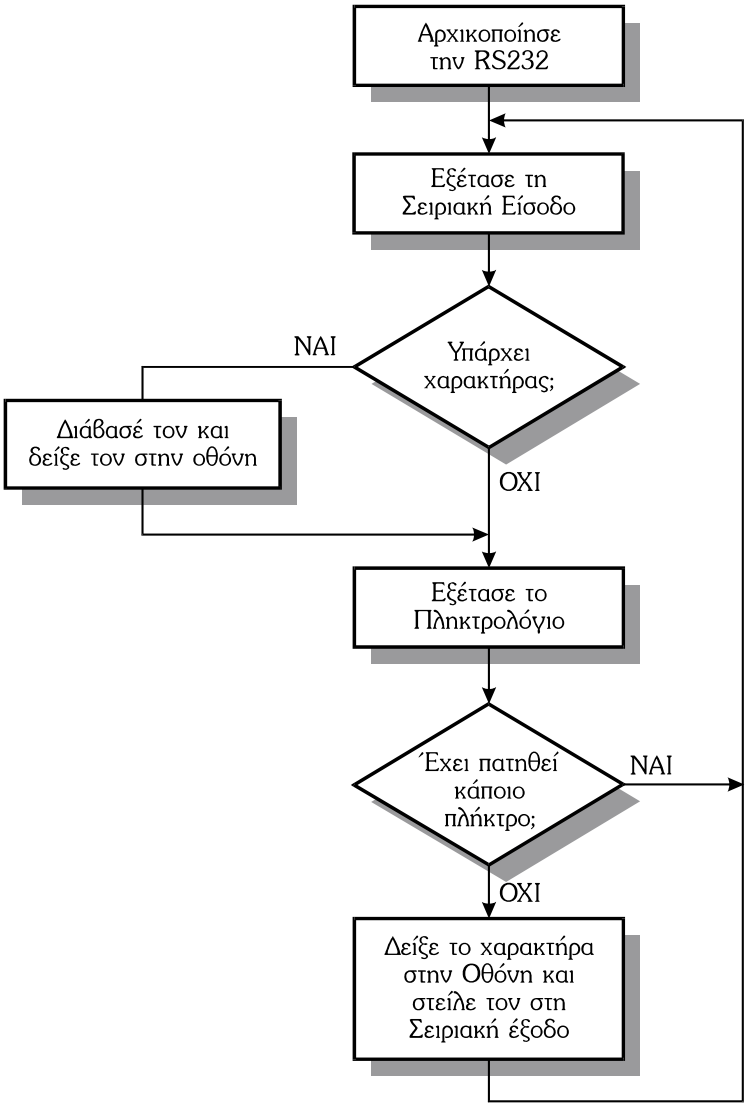
```
TXCH_RS232      PROC NEAR
                PUSH AX         ; Ο χαρακτήρας που θα σταλεί βρίσκεται
                                  στον AL
                MOV DX,3FD      ; Δείξε στον καταχωρητή κατάστασης
                                  γραμμής
TXCH_RS232_2:
                IN AL,DX        ; Πάρε την κατάσταση στον AL
                TEST AL,20H     ; Έλεγχος του bit THRE
                JZ TXCH_RS232_2
                SUB DX,5        ; Αν είναι άδειος ο καταχωρητής
                POP AX          ; εξόδου, δείξε σε αυτόν
                OUT DX,AL       ; και στείλε του τον χαρακτήρα
                RET
TXCH_RS232      ENDP
```

Με βάση τα όσα αναφέρθηκαν παραπάνω, το PC μπορεί να μετατραπεί σε ένα απλό τερματικό. Αυτό σημαίνει ότι:

α) Αν εισαχθεί κάποιος χαρακτήρας από το πληκτρολόγιο θα στέλνεται στη σειριακή έξοδο και αν έχει ορισθεί αντήχηση (echo) θα εμφανίζεται και στην οθόνη.

β) Αν ληφθεί κάποιος χαρακτήρας από τη σειριακή είσοδο θα εμφανίζεται στην οθόνη.

Ένας τρόπος για να γίνει αυτό είναι να χρησιμοποιηθεί η μέθοδος polling όπως δείχνει το ακόλουθο διάγραμμα ροής:



Για την ανάγνωση του χαρακτήρα από το πληκτρολόγιο μπορεί να χρησιμοποιηθεί η λειτουργία του DOS:

```
MOV  AH,06H
MOV  DL,FFH
INT  21H
```

Στην επιστροφή αν ZF=0 σημαίνει ότι έχει πατηθεί κάποιο πλήκτρο και ο αντίστοιχος χαρακτήρας περιέχεται στον AL. Διαφορετικά θα είναι ZF=1 και AL=0.

Επίσης, για την εμφάνιση χαρακτήρα στην οθόνη μπορεί να χρησιμοποιηθεί η κλήση:

```
MOV  AH,2
MOV  DL, char(ascii)
INT  21H
```

Τέλος, παρουσιάζεται σύντομα η διαδικασία για τη δημιουργία ενός COM εκτελέσιμου αρχείου. Ο source code για ένα τέτοιο αρχείο έχει την ακόλουθη γενική μορφή:

```
CODE_SEG    SEGMENT
            ASSUME CS:CODE_SEG, DS:CODE_SEG, ES:CODE_SEG
            ORG 100H                ; very important
PROGRAM     PROC FAR
            JMP START               ; jump around the data
            .
            .   (DB, DW, etc.  data fields)
            .
START:      CALL LOCAL_PROC         ; sample CALL to local procedure
            .
            .
            .   (insert main program logic here)
            .
            MOV AX,4C00h            ; direct exit to DOS
            INT 21H
PROGRAM     ENDP
CODE_SEG    ENDS
END         PROGRAM                 ; specify starting address in END statement
```

Για τη δημιουργία του πρέπει να εκτελεστούν τα ακόλουθα βήματα:

1. Με κάποιον editor δημιουργείται το source code αρχείο π.χ. το TERMINAL.ASM
2. Γίνεται assembling π.χ. MASM TERMINAL και παράγεται το αρχείο TERMINAL.OBJ
3. Γίνεται linking π.χ. LINK TERMINAL και παράγεται το αρχείο TERMINAL.EXE. Αγνοείστε το μήνυμα "no stack segment".
4. Γίνεται μετατροπή του EXE αρχείου σε COM με το πρόγραμμα EXE2BIN.COM εκτελώντας: EXE2BIN TERMINAL TERMINAL.COM και παράγεται το τελικό εκτελέσιμο αρχείο TERMINAL.COM.

3. Τα θέματα της 6ης Εργαστηριακής Άσκησης

i. Μετατροπή PC σε τερματικό: Για να μετατραπεί ένα PC σε τερματικό αρκεί να τυπώνει στην οθόνη του τους χαρακτήρες που δέχεται από τη σειριακή πόρτα RS232 και να στέλνει από την ίδια πόρτα τους χαρακτήρες που πληκτρολογούνται. Το πρόγραμμα που θα γραφεί για το σκοπό αυτό θα πρέπει να ζητάει από το χρήστη τις ακόλουθες παραμέτρους:

   α) Echo (on ή off): Echo on σημαίνει ότι οι χαρακτήρες που πληκτρολογούνται τυπώνονται και στην οθόνη.

   β) Ταχύτητα επικοινωνίας (baud rate): Οι δυνατές τιμές είναι 300, 600, 1200, 2400, 4800 και 9600.

   Επίσης το πρόγραμμα αυτό θα πρέπει να σταματάει και να επιστρέφει στο DOS με το πάτημα κάποιου πλήκτρου που εσείς θα καθορίσετε. Στο κεφάλαιο αυτό δόθηκαν οι απαραίτητες πληροφορίες και ρουτίνες για την υλοποίηση που ζητείται. Για τη λύση του προβλήματος υποδεικνύεται και ένα διάγραμμα ροής. Προαιρετικά μπορείτε να προτείνετε ένα δικό σας διάγραμμα ροής ή και να τροποποιήσετε τις ρουτίνες που δίνονται.

Με τη βοήθεια του προγράμματος αυτού δημιουργήστε ένα περιβάλλον επικοινωνίας δύο PCs όπου στη μισή οθόνη να μπορούμε να πληκτρολογούμε μηνύματα και στην άλλη μισή να λαμβάνουμε μηνύματα.

ii. Επεξεργασία αρχείων: Δημιουργήστε ένα αρχείο 256 δυαδικών δεδομένων με τιμές από το 0 ως το 255, το οποίο να αποθηκεύσετε πάνω στο δίσκο. Το κάθε δεδομένο του αρχείου (byte) να διαβαστεί και αφού χωριστεί σε δύο τετράδες από bits, η κάθε μια από αυτές τις τετράδες (αρχίζοντας από τη λιγότερο σημαντική) να μετατραπεί στον ASCII χαρακτήρα που συμβολίζει τη δεκαεξαδική τιμή της. Το νέο αρχείο (διπλάσιου μεγέθους) να αποθηκευτεί επίσης στο δίσκο. Τη διαχείριση των δύο αρχείων θα πρέπει να αναλαμβάνουν ρουτίνες υψηλού επιπέδου, ενώ η ρουτίνα assembly θα δέχεται ως είσοδο ένα χαρακτήρα και θα κάνει τη μετατροπή που ζητείται.

   Κάντε το ίδιο χωρίς να χρησιμοποιήσετε γλώσσα υψηλού επιπέδου. Θα χρειαστείτε κάποιες ρουτίνες του MS-DOS που θα τις βρείτε στο βιβλίο Συστήματα Μικροϋπολογιστών του ομώνυμου μαθήματος του 7ου εξαμήνου.

ΠΑΡΑΡΤΗΜΑ 1

Έτοιμα προγράμματα στη ROM του μLab

- 1. Προγράμματα επίδειξης
- 2. Utilities

1. Προγράμματα επίδειξης

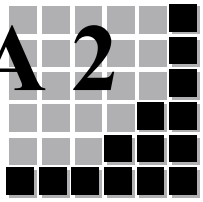
Ονομασία	Διεύθυνση εκκίνησης	Περιγραφή
ECHO	04D7	Παρουσιάζει data από τα input switches στα output LEDs.
ANDGT	04E0	Χρησιμοποιεί τα input switches και τα output LEDs για την υλοποίηση μιας πύλης AND.
CONV	04F8	Ελεγκτής ιμάντα μεταφοράς.
WTM	053E	Παραγωγή τυχαίων μουσικών τόνων από τη ROM.
SQRL	055A	Παιχνίδι με σκίουρο.
ORGAN	0599	Γεννήτρια ήχων με χρήση του keyboard.
ROCT	05F9	Παιχνίδι με πύραυλο.
STW	0662	Παιχνίδι με ψηφιακό ρολόι.
SNAKE	06C2	Παιχνίδι με φιδάκι.

2. Utilities

Ονομασία	Επηρρεαζόμενοι καταχωρητές	Δεδομένα	Διεύθυνση εκκίνησης	Περιγραφή
BEEP	Όλοι	Κανένα	0010	Παράγει ένα απλό beep καθορισμένης συχνότητας και διάρκειας.
BEEP 1	Όλοι εκτός του B	B: συχνότητα	0012	Παράγει ένα απλό beep καθορισμένης διάρκειας και συχνότητας που καθορίζει η τιμή του B. Χαμηλή τιμή του B =>υψηλή συχνότητα (minimum τιμή 01).
BEEP 2	Όλοι εκτός του B και του D	B: συχνότητα & D: διάρκεια	0447	Όμοια με την BEEP 1 αλλά με διάρκεια την τιμή του D. Μικρή τιμή => μικρή διάρκεια.
KIND (Key Input & Decode)	A	είσοδος από πληκτρολόγιο	014B	Τροποποιεί τα displays διαβάζοντας και αποκωδικοποιώντας το πληκτρολόγιο. Η τιμή του πλήκτρου (0-F) αφήνεται στον A.
KPU (Key Pushed)	A, H και L	Είσοδος από πληκτρολόγιο	0185	Ελέγχει το πληκτρολόγιο και αν πατηθεί πλήκτρο καθαρίζει το zero flag.
SDS (Scan Display Segments)	Κανείς	display segments	01C8	Τα segment data που βρίσκονται στις θέσεις RAM 0BFA-0BFF στέλνονται στα displays 0-5. Το κάθε ένα είναι ON για 1 ms (βλέπε σχήμα 1).
DELA (Fixed Delay)	Κανείς	Κανένα	0429	Προκαλεί προκαθορισμένη καθυστέρηση 1 ms.
DELB (Variable Delay)	Κανείς	Ζεύγος καταχωρητών BC	0430	Προκαλεί μεταβλητή καθυστέρηση ίση με την τιμή του ζεύγους BC επί 1 ms.
RS1	Κανείς	Κανένα	0008	Breakpoint για επιστροφή στο monitor. Ισοδυναμεί με CF που είναι η εντολή RST 1.

DCD (Display Character Decoder)	Κανείς	display digits	01E9	Παίρνει τους κωδικούς των χαρακτήρων των displays (00-1C) που είναι στη RAM στις θέσεις 0BF0-0BF5 και τους απεικονίζει στα displays. Στη συνέχεια τους σώζει στις 0BFA-0BFF και εξετάζει το display μια φορά. Η αποκωδικοποίηση γίνεται με βαση τον πίνακα DDC στη διεύθυνση 0218 (βλέπε σχήματα 2 και 3).
STDM (Store Display Message)	Όλοι	Ζεύγος καταχωρητών DE	0018	Παίρνει το μήνυμα 6 χαρακτήρων από τη διεύθυνση του DE και το σώζει στη RAM (0BF0-0BF5). Ο 1ος χαρακτήρας εμφανίζεται στο δεξιότερο ψηφίο του display. Αυτή η ρουτίνα ακολουθείται συνήθως από την DCD. Μπορεί εξάλλου να κληθεί γράφοντας DF στο πρόγραμμα. Το DF είναι η εντολή RST 3 (βλέπε σχήματα 2 και 3).

ΠΑΡΑΡΤΗΜΑ 2



Πίνακας αναφοράς κώδικα assembly 8085

- 1. Εντολές μεταφοράς δεδομένων
- 2. Εντολές αριθμητικών και λογικών πράξεων
- 3. Εντολές ελέγχου διακλάδωσης
- 4. Εντολές στοίβας, ελέγχου και I/O

ΠΙΝΑΚΑΣ 1

1. Εντολές μεταφοράς δεδομένων

MOV r1, r2							
(r1) ← (r2)							
A		**B**		**C**		**D**	
MOV A,A	7F	MOV B,A	47	MOV C,A	4F	MOV D,A	57
MOV A,B	78	MOV B,B	40	MOV C,B	48	MOV D,B	50
MOV A,C	79	MOV B,C	41	MOV C,C	49	MOV D,C	51
MOV A,D	7A	MOV B,D	42	MOV C,D	4A	MOV D,D	52
MOV A,E	7B	MOV B,E	43	MOV C,E	4B	MOV D,E	53
MOV A,H	7C	MOV B,H	44	MOV C,H	4C	MOV D,H	54
MOV A,L	7D	MOV B,L	45	MOV C,L	4D	MOV D,L	55
MOV A,M	7E	MOV B,M	46	MOV C,M	4E	MOV D,M	56

E		**H**		**L**		**M**	
MOV E,A	5F	MOV H,A	67	MOV L,A	6F	MOV M,A	77
MOV E,B	58	MOV H,B	60	MOV L,B	68	MOV M,B	70
MOV E,C	59	MOV H,C	61	MOV L,C	69	MOV M,C	71
MOV E,D	5A	MOV H,D	62	MOV L,D	6A	MOV M,D	72
MOV E,E	5B	MOV H,E	63	MOV L,E	6B	MOV M,E	73
MOV E,H	5C	MOV H,H	64	MOV L,H	6C	MOV M,H	74
MOV E,L	5D	MOV H,L	65	MOV L,L	6D	MOV M,L	75
MOV E,M	5E	MOV H,M	66	MOV L,M	6E	MOV M,M	76

MVI r, byte		**LXI rp, dble**		**XCHG**	
(r) ← (byte2)		(rh) ← (byte3) (rl) ← (byte2)		(HL) ↔ (DE)	
MVI A,byte	3E	LXI B, dble	01	XCHG	EB
MVI B,byte	06	LXI D, dble	11		
MVI C,byte	0E	LXI H, dble	21		
MVI D,byte	16	LXI SP,dble	31		
MVI E,byte	1E				
MVI H,byte	26				
MVI L,byte	2E				
MVI M,byte	36				

LDAX B	0A	(A) ← [(BC)]
LDAX D	1A	(A) ← [(DE)]
STAX B	02	[(BC)] ← (A)
STAX D	12	[(DE)] ← (A)

LHLD adr	2A	(L) ← (adr) , (H) ← (adr+1)
SHLD adr	22	(adr) ← (L) , (adr+1) ← (H)
LDA adr	3A	(A) ← (adr)
STA adr	32	(adr) ← (A)

2. Εντολές αριθμητικών και λογικών πράξεων

Add / Subtract *

ADD r		**ADC r**		**SUB r**		**SBB r**	
(A)←(A)+(r)		(A)←(A)+(r)+(CY)		(A)←(A)-(r)		(A)←(A)-(r)-(CY)	
ADD A	87	ADC A	8F	SUB A	97	SBB A	9F
ADD B	80	ADC B	88	SUB B	90	SBB B	98
ADD C	81	ADC C	89	SUB C	91	SBB C	99
ADD D	82	ADC D	8A	SUB D	92	SBB D	9A
ADD E	83	ADC E	8B	SUB E	93	SBB E	9B
ADD H	84	ADC H	8C	SUB H	94	SBB H	9C
ADD L	85	ADC L	8D	SUB L	95	SBB L	9D
ADD M	86	ADC M	8E	SUB M	96	SBB M	9E

Increment / Decrement **

INR r		**DCR r**		**INX rp**		**DCX rp**	
(r)←(r)+1		(r)←(r)-1		(rp)←(rp)+1		(rp)←(rp)-1	
INR A	3C	DCR A	3D	INX B	03	DCX B	0B
INR B	04	DCR B	05	INX D	13	DCX D	1B
INR C	0C	DCR C	0D	INX H	23	DCX H	2B
INR D	14	DCR D	15	INX SP	33	DCX SP	3B
INR E	1C	DCR E	1D				
INR H	24	DCR H	25				
INR L	2C	DCR L	2D				
INR M	34	DCR M	35				

Logical *

ANA r		XRA r		ORA r		CMP r	
(A)←(A)and(r)		(A)←(A)xor(r)		(A)←(A) or(r)		(A)=(r) Z=1 (A)<(r) CY=1	
ANA A	A7	XRA A	AF	ORA A	B7	CMP A	BF
ANA B	A0	XRA B	A8	ORA B	B0	CMP B	B8
ANA C	A1	XRA C	A9	ORA C	B1	CMP C	B9
ANA D	A2	XRA D	AA	ORA D	B2	CMP D	BA
ANA E	A3	XRA E	AB	ORA E	B3	CMP E	BB
ANA H	A4	XRA H	AC	ORA H	B4	CMP H	BC
ANA L	A5	XRA L	AD	ORA L	B5	CMP L	BD
ANA M	A6	XRA M	AE	ORA M	B6	CMP M	BE

Specials		Double Add +		Rotate +		Arith. & Logical Immediate	
DAA *	27	DAD B	09	RLC	07	ADI byte	C6
CMA +	2F	DAD D	19	RRC	0F	ACI byte	CE
STC +	37	DAD H	29	RAL	17	SUI byte	D6
CMC +	3F	DAD SP	39	RAR	1F	SBI byte	DE
						ANI byte	E6
						XRI byte	EE
						ORI byte	F6
						CPI byte	FE

3. Εντολές ελέγχου διακλάδωσης

JMP adr		CALL adr		Return		Restart	
(PC)←(adr)		[(SP)-1]←(PCH) [(SP)-2]←(PCL) (SP)←(SP)-2 (PC)←(adr)		(PCL)←[(SP)] (PCH)←[(SP)+1] (SP)←(SP)+2		[(SP)-]←(PCH) [(SP)-]←(PCL) (SP)←(SP)-2 (PC)←8*(NNN)	
JMP adr	C3	CALL adr	CD	RET	C9	RST 0	C7
JNZ adr	C2	CNZ adr	C4	RNZ	C0	RST 1	CF
JZ adr	CA	CZ adr	CC	RZ	C8	RST 2	D7
JNC adr	D2	CNC adr	D4	RNC	D0	RST 3	DF
JC adr	DA	CC adr	DC	RC	D8	RST 4	E7
JPO adr	E2	CPO adr	E4	RPO	E0	RST 5	EF
JPE adr	EA	CPE adr	EC	RPE	E8	RST 6	F7
JP adr	F2	CP adr	F4	RP	F0	RST 7	FF
JM adr	FA	CM adr	FC	RM	F8		

PCHL	E9	(PC)←(HL)

4. Εντολές στοίβας, ελέγχου και I/O

PUSH rp	POP rp	XHTL	SPHL
[(SP)-1] ← (rh) [(SP)-2] ← (rl) (SP) ← (SP)-2	(rl) ← [(SP)] (rh) ← [(SP)+1] (SP) ← (SP)+2	(L) ← [(SP)] (H) ← [(SP)+1]	(SP) ← (HL)
PUSH B C5 PUSH D D5 PUSH H E5 PUSH PSW F5	POP B C1 POP D D1 POP H E1 POP PSW F1	XHTL E3	SPHL F9

Disable interrupts	DI	F3
Enable interrupts	EI	FB
No operation	NOP	00
Halt	HLT	76

RIM	20
SIM	30
IN byte	DB
OUT byte	D3

Σημείωση:

byte - constant or logical expression that evaluates to an 8-bit data quantity. (Second byte of 2-byte instructions).

Dble - constant or logical /arithmetic expression that evaluates to a 16-bit data quantity. (2nd and 3rd bytes of 3-byte instructions).

Adr - 16-bit address (2nd and 3rd bytes of 3-bytes instructions).

* - all flags (C, Z, S, P, AC) affected.

** - all flags except CARRY affected. (exception: INX and DCX affect no flags).

\+ - only CARRY affected.

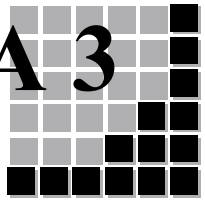
M - memory location addressed by the contents of HL pair

ΠΙΝΑΚΑΣ 2

00	NOP	2F	CMA	5E	MOV E,M	8D	ADC L
01	LXI B,*dble*	30	SIM	5F	MOV E,A	8E	ADC M
02	STAX B	31	LXI SP,*dble*	60	MOV H,B	8F	ADC A
03	INX B	32	STA *adr*	61	MOV H,C	90	SUB B
04	INR B	33	INX SP	62	MOV H,D	91	SUB C
05	DCR B	34	INR M	63	MOV H,E	92	SUB D
06	MVI B,*byte*	35	DCR M	64	MOV H,H	93	SUB E
07	RLC	36	MVI M,*byte*	65	MOV H,L	94	SUB H
08	---	37	STC	66	MOV H,M	95	SUB L
09	DAD B	38	---	67	MOV H,A	96	SUB M
0A	LDAX B	39	DAD SP	68	MOV L,B	97	SUB A
0B	DCX B	3A	LDA *adr*	69	MOV L,C	98	SBB B
0C	INR C	3B	DCX SP	6A	MOV L,D	99	SBB C
0D	DCR C	3C	INR A	6B	MOV L,E	9A	SBB D
0E	MVI C,*byte*	3D	DCR A	6C	MOV L,H	9B	SBB E
0F	RLC	3E	MVI A,*byte*	6D	MOV L,L	9C	SBB H
10	---	3F	CMC	6E	MOV L,M	9D	SBB L
11	LXI D,*dble*	40	MOV B,B	6F	MOV L,A	9E	SBB M
12	STAX D	41	MOV B,C	70	MOV M,B	9F	SBB A
13	INX D	42	MOV B,D	71	MOV M,C	A0	ANA B
14	INR D	43	MOV B,E	72	MOV M,D	A1	ANA C
15	DCR D	44	MOV B,H	73	MOV M,E	A2	ANA D
16	MVI D,*byte*	45	MOV B,L	74	MOV M,H	A3	ANA E
17	RAL	46	MOV B,M	75	MOV M,L	A4	ANA H
18	---	47	MOV B,A	76	HLT	A5	ANA L
19	DAD D	48	MOV C,B	77	MOV M,A	A6	ANA M
1A	LDAX D	49	MOV C,C	78	MOV A,B	A7	ANA A
1B	DCX D	4A	MOV C,D	79	MOV A,C	A8	XRA B
1C	INR E	4B	MOV C,E	7A	MOV A,D	A9	XRA C
1D	DCR E	4C	MOV C,H	7B	MOV A,E	A	XRA D
1E	MVI E,*byte*	4D	MOV C,L	7C	MOV A,H	A	
1F	RAR	4E	MOV C,M	7D	MOV A,L	AB	XRA E
20	RIM	4F	MOV C,A	7E	MOV A,M	AC	XRA H
21	LXI H,*dble*	50	MOV D,B	7F	MOV A,A	A	XRA L
22	SHLD *adr*	51	MOV D,C	80	ADD B	D	
23	INX H	52	MOV D,D	81	ADD C	AE	XRA M
24	INR H	53	MOV D,E	82	ADD D	AF	XRA A
25	DCR H	54	MOV D,H	83	ADD E	B0	ORA B
26	MVI H,*byte*	55	MOV D,L	84	ADD H	B1	ORA C
27	DAA	56	MOV D,M	85	ADD L	B2	ORA D
28	---	57	MOV D,A	86	ADD M	B3	ORA E
29	DAD H	58	MOV E,B	87	ADD A	B4	ORA H
2A	LHLD *adr*	59	MOV E,C	88	ADC B	B5	ORA L
2B	DCX H	5A	MOV E,D	89	ADC C	B6	ORA M
2C	INR L	5B	MOV E,E	8A	ADC D	B7	ORA A
2D	DCR L	5C	MOV E,H	8B	ADC E	B8	CMP B
2E	MVI L,*byte*	5D	MOV E,L	8C	ADC H	B9	CMP C

BA	CMP D	CC	CZ *adr*	D	---	EE	XRI *byte*
BB	CMP E	CD	CALL *adr*	D		EF	RST 5
BC	CMP H	CE	ACI *byte*	DE	SBI *byte*	F0	RP
BD	CMP L	CF	RST 1	DF	RST 3	F1	POP PSW
BE	CMP M	D0	RNC	E0	RPO	F2	JP *adr*
BF	CMP A	D1	POP D	E1	POP H	F3	DI
C0	RNZ	D2	JNC *adr*	E2	JPO *adr*	F4	CP *adr*
C1	POP B	D3	OUT *byte*	E3	XTHL	F5	PUSH PSW
C2	JNZ *adr*	D4	CNC *adr*	E4	CPO *adr*	F6	ORI *byte*
C3	JMP *adr*	D5	PUSH D	E5	PUSH H	F7	RST 6
C4	CNZ *adr*	D6	SUI *byte*	E6	ANI *byte*	F8	RM
C5	PUSH B	D7	RST 2	E7	RST 4	F9	SPHL
C6	ADI *byte*	D8	RC	E8	RPE	FA	JM *adr*
C7	RST 0	D9	---	E9	PCHL	FB	EI
C8	RZ	D	JC *adr*	EA	JPE *adr*	FC	CM *adr*
C9	RET	A		EB	XCHG	FD	---
CA	JZ	DB	IN *byte*	EC	CPE *adr*	FE	CPI *byte*
CB	---	DC	CC *adr*	ED	---	FF	RST 7

ΠΑΡΑΡΤΗΜΑ 3



Πλήρες listing της ROM του μLab

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0000			ORG 0	
0000	26 08	RESET:	MVI H,8	;PAGE ADRS OF WRITE TEST RAM
0002	7E		MOV A,M	;TEST RAM CELL DATA
0003	2F		CMA	;COMPLEMENT IT
0004	77		MOV M,A	;STORE IT BACK IN RAM
0005	C3 40 00		JMP STRT	;TO CONTINUE
;				
;RS1 IS MAIN ENTRY POINT TO MONITOR				
0008			ORG 8	
0008	22 D3 0B	RS1:	SHLD TSAVH	;SAVE USER HL CONTENTS IN RAM
000B	D3 10		OUT 10H	;UNPROTECT RAM
000D	C3 F3 00		JMP TRP	;CONTINUE
;				
;BEEP PRODUCES A FIXED TONE FREQ+DURATION				
0010			ORG 10H	
0010	06 06	BEEP:	MVI B,FREQ	;DEFAULT FREQUENCY
0012	16 04	BEEP1:	MVI D,DURA	;DEFAULT DURATION
0014	C3 47 04		JMP BEEP2	;CONTINUE
0017	00			NOP
;				
;STDM STORES DISP MESSAGE AT DE ADRS IN UDSP RAM				
0018			ORG 18H	
0018	C5	STDM:	PUSH B	
0019	21 F0 0B		LXI H,UDSP0	;ADRS OF UNDECODED DISP. DIGIT 0
001C	C3 35 02		JMP SDM	;CONTINUE
001F	00		NOP	
0020			ORG 20H	
0020	C3 F0 0A	RS4:	JMP RS4C	;USER ROUTINE
0023	00		NOP	
0024			ORG 24H	
0024	C3 08 00	TRAP:	JMP RS1	;SAV USER REGS+RETURN TO MONITOR

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0027	00		NOP	
0028			ORG 28H	
0028	C3 F3 0A	RS5:	JMP RS5C	;USER ROUTINE
002B	00		NOP	
002C			ORG 2CH	
002C	C3 F6 0A	RS55:	JMP RS55C	;USER ROUTINE
002F	00		NOP	
0030			ORG 30H	
0030	C3 F9 0A	RS6:	JMP RS6C	;USER ROUTINE
0033	00		NOP	
0034			ORG 34H	
0034	C3 FC 0A	RS65:	JMP RS65C	;USER INTERRUPT KEY ROUTINE
0037	00		NOP	
0038			ORG 38H	
0038	C3 A6 00	RS7:	JMP STRT6	;RETURN TO MONITOR
003B	00		NOP	
003C			ORG 3CH	
003C	D7	RS75:	RST 2	;BEEP
003D	C3 8D 04		JMP SATL1	;SA TEST LOOP
;				
;POWER-UP SELF TEST AND INITIALIZE				
0040			ORG 40H	
0040	BE	STRT:	CMP M	;SEE IF DATA STORED IN RAM
0041	C2 C8 00		JNZ PPER	;IF RAM WAS PROTECTED (RUN MODE)
0044	31 CE 0B		LXI SP,MSP	;INITIALIZE MONITOR SP
0047	AF		XRA A	;CLEAR A
0048	67		MOV H,A	;CLEAR H FIRST ADRS
0049	6F		MOV L,A	;CLEAR L OF ROM
004A	D3 30		OUT LOUT	;TURN ON OUTPUT LEDS
;				
;ROM SELF TEST				
004C	86	STRT1:	ADD M	;ADD ROM DATA TO A

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
004D	23		INX H	;POINT TO NEXT ROM ADRS
004E	4F		MOV C,A	;SAVE A RESIDUE IN C
004F	3E 08		MVI A,08H	;LAST ADRS OF ROM+1 (MS BYTE)
0051	BC		CMP H	;COMPARE IT TO H
0052	79		MOV A,C	;RESTORE RESIDUE TO A
0053	C2 4C 00		JNZ STRT1	;IF LAST ROM ADRS NOT 0800
0056	2B		DCX H	;POINT HL TO 07FF CHECKSUM VALUE
0057	96		SUB M	;SUBTRACT CHECKSUM FOR A RESIDUE
0058	BE		CMP M	;COMPARE RESIDUE TO CHECKSUM
0059	06 04		MVI B,04	;IC 4 (ROM) MESSAGE
005B	C2 E5 00		JNZ MERR1	;IF NOT A MATCH
;				
;RAM SELF TEST				
005E	AF		XRA A	;CLEAR A
005F	21 00 08		LXI H,0800H	;1ST RAM ADRS
0062	06 03		MVI B,3	;ADD CONSTANT
0064	77	STRT2:	MOV M,A	;STORE DATA IN RAM
0065	80		ADD B	;ADD 3 TO A
0066	23		INX H	;POINT TO NEXT RAM ADRS
0067	4F		MOV C,A	;SAVE A
0068	7C		MOV A,H	;GET MS BYTE ADRS
0069	FE 0C		CPI 0CH	;LAST RAM ADRS+1
006B	79		MOV A,C	;RESTORE A
006C	C2 64 00		JNZ STRT2	;IF NOT LAST RAM ADRS
006F	AF		XRA A	;CLEAR A
0070	21 00 08		LXI H,0800H	;1ST RAM ADRS
0073	BE	STRT3:	CMP M	;DID DATA GET STORED IN RAM?
0074	C2 DC 00		JNZ MERR	;IF DATA NOT SAVED
0077	2F		CMA	
0078	77		MOV M,A	;STORE COMPLEMENT BACK IN RAM
0079	BE		CMP M	;DID IT STORE?
007A	C2 DC 00		JNZ MERR	;IF NOT

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
007D	2F		CMA	;UNCOMPLEMENT A
007E	80		ADD B	;ADD 3 TO A
007F	23		INX H	;NEXT RAM ADRS
0080	4F		MOV C,A	;SAVE A
0081	7C		MOV A,H	;GET MS BYTE ADRS
0082	FE 0C		CPI 0CH	;LAST RAM ADRS+1
0084	79		MOV A,C	;RESTORE A
0085	C2 73 00		JNZ STRT3	;TO CHECK NEXT RAM LOCATION
;				
;DISPLAY TEST				
0088	06 00		MVI B,0	;CLEAR LOOP COUNTER
008A	11 81 02	STRT4:	LXI D,ALL	;DISPLAY MESSAGE POINTER ALL SEGS
008D	DF		RST 3	;GET MESSAGE
008E	CD E9 01		CALL DCD	;UPDATE DISPLAY
0091	05		DCR B	;DECR LOOP COUNTER
0092	C2 8A 00		JNZ STRT4	;IF NOT DONE
;				
;CLEAR RAM (STORE 00 IN ALL LOCATIONS)				
0095	06 00		MVI B,0	;CLEAR B
0097	3E 0C		MVI A,0CH	;MS BYTE ADRS OF TOP OF RAM +1
0099	21 00 08		LXI H,0800H	;1ST ADRS OF RAM
009C	70	STRT5:	MOV M,B	;CLEAR RAM LOCATION
009D	23		INX H	;POINT TO NEXT LOCATION
009E	BC		CMP H	;TO LAST RAM ADRS+1
009F	C2 9C 00		JNZ STRT5	;IF NOT DONE CLEARING RAM
00A2	3E FF		MVI A,0FFH	;SET A TO ALL ONES
00A4	D3 30		OUT LOUT	;TURN OFF OUTPUT LEDS
00A6	D7	STRT6:	RST 2	;SIGNAL START-UP DONE
;				
;INITIALIZE REGISTERS				
00A7	21 B0 0B		LXI H,USP	;USER SP DEFAULT VALUE
00AA	22 DE 0B		SHLD SAVSL	;STORE IT IN RAM

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
00AD	21 D6 0B		LXI H,RS	;RUN STATUS WORD ADRS
00B0	36 00		MVI M,0	;SET STATUS TO MONITOR
00B2	3E 0B		MVI A,DIM	;DEFAULT INTERRUPT MASK
00B4	32 DB 0B		STA SAVIM	;STORE IT IN RAM
00B7	21 00 08	STRT7:	LXI H,PC	;DEFAULT PC
00BA	22 DC 0B		SHLD SAVPC	;STORE IT IN RAM
00BD	3E FF		MVI A,0FFH	;RST 7 INSTR CODE
00BF	32 FF 0A		STA TPR	;STORE IT IN TOP OF PROTECTED RAM
00C2	32 EF 0A		STA UR	;STORE IT IN UPPER RAM BELOW LINKS
00C5	C3 11 01		JMP TRP3	;JUMP TO MONITOR
;				
;PUSH-POP ERROR ROUTINE				
00C8	31 CE 0B	PPER:	LXI SP,MSP	;SET MONITOR SP
00CB	21 00 08		LXI H,PC	;DEFAULT PC
00CE	22 DC 0B		SHLD SAVPC	;STORE IT IN RAM
00D1	D7		RST 2	;SIGNAL AN ERROR
00D2	AF		XRA A	;CLEAR A
00D3	32 D6 0B		STA RS	;SET RUN STATUS TO MONITOR
00D6	11 8D 02		LXI D,PPM	;PUSH-POP ERROR MESSAGE ADRS
00D9	C3 15 01		JMP TRP4	
;				
;MEMORY ERROR SORT				
00DC	06 06	MERR:	MVI B,6	;IC6 RAM FAIL MESSAGE
00DE	AE		XRA M	;GET DIFFERENCE INTO A
00DF	E6 0F		ANI 0FH	;TEST 4 LSB'S OF IC6
00E1	CA E5 00		JZ MERR1	;IF PROBLEM IN IC6
00E4	05		DCR B	;SET B TO IC5
00E5	11 87 02	MERR1:	LXI D,ICX	;ICX MESSAGE ADRS
00E8	DF		RST 3	;GET MESSAGE
00E9	21 F2 0B		LXI H,UDSP2	;ADRS OF ICX IN UDSP RAM
00EC	70		MOV M,B	;STORE IC NUMBER IN UDSP2
00ED	CD E9 01	MERR2:	CALL DCD	;DISPLAY MESSAGE

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
00F0	C3 ED 00		JMP MERR2	;LOOP MESSAGE
;				
;RS1 IS MAIN ENTRY POINT TO MONITOR				
00F3	21 00 00	TRP:	LXI H,0	;CLEAR H-L
00F6	D2 FA 00		JNC TRP1	;NO USER CARRY WILL BYPASS DCXH
00F9	2B		DCX H	;SET H-L TO FF IF CARRY PRESENT
00FA	39	TRP1:	DAD SP	;GET SP VALUE INTO HL, RESTORE CY
00FB	D2 FF 00		JNC TRP2	;IF JNC TO TRP1 OCCURED
00FE	23		INX H	;IF DCXH OCCURED BECAUSE OF CARRY
00FF	22 D1 0B	TRP2:	SHLD TSAVSP	;SAVE USER SP IN RAM
0102	31 D1 0B		LXI SP,TSAVSP	;TSAVA+1 ADRS
0105	F5		PUSH PSW	;SAVE PSW AND A IN RAM
0106	21 D6 0B		LXI H,RS	;RUN STATUS ADRS
0109	AF		XRA A	;CLEAR A
010A	BE		CMP M	;FOR RUN STATUS=MONITOR
010B	32 F6 0B		STA UDSP6	;CLEAR DATA MODIFY FLAG
010E	C2 24 01		JNZ TRP6	;IF CAME FROM USER PROGRAM
0111	11 41 02	TRP3:	LXI D,DMT	;ULAB MESSAGE ADRS
0114	FB		EI	
0115	3E 0B	TRP4:	MVI A,DIM	;DEFAULT INTERRUPT MASK
0117	30		SIM	;SET IT TO ENABLE RST 7.5 ONLY
0118	D3 10		OUT 10H	;UNPROTECT RAM
011A	DF		RST 3	;GET MESSAGE
011B	CD 4B 01	TRP5:	CALL KIND	;INPUT KEYS
011E	CD B8 02		CALL CFETA	;LOOK FOR ACCEPTABLE KEYS
0121	C3 1B 01		JMP TRP5	;TRY AGAIN
0124	77	TRP6:	MOV M,A	;STORE 0 IN RS TO SET MONITOR
;				
;SAVES REGISTERS				
0125	F1		POP PSW	;FROM TSAVPSW IN RAM
0126	E1		POP H	;GETS USER SP VALUE FROM RAM
0127	23		INX H	;USER SP

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0128	23		INX H	;USER SP
0129	22 DE 0B		SHLD SAVSL	;SAVE SP IN RAM
012C	2B		DCX H	;USER SP
012D	2B		DCX H	;USER SP
012E	F9		SPHL	;RESTORE SP
012F	E1		POP H	;GET RETURN ADRS TO USER PROGRAM
0130	22 DC 0B		SHLD SAVPC	;STORE IT IN RAM
0133	31 E8 0B		LXI SP,0BE8H	;ADRS OF SAVA+1
0136	2A D3 0B		LHLD TSAVH	;RESTORE H-L
0139	F5		PUSH PSW	;INTO SAVPSW
013A	C5		PUSH B	;INTO SAVB
013B	D5		PUSH D	;INTO SAVD
013C	E5		PUSH H	;INTO SAVH
013D	20		RIM	;GET IM
013E	32 DB 0B		STA SAVIM	;STORE IT IN RAM
0141	31 DC 0B	TRP7:	LXI SP,SAVPC	;POINT IT TO USER SP
0144	C1		POP B	;AND POP IT IN BC
0145	31 CE 0B		LXI SP,MSP	;RESTORE MONITOR SP
0148	C3 64 03		JMP FETA3	
;				
;KEY INPUT AND DECODE				
014B	D5		KIND:	PUSH D
014C	E5		PUSH H	
014D	CD E9 01	KIND1:	CALL DCD	;UPDATE DISPLAY AND WAIT
0150	CD 85 01		CALL KPU	;CHK FOR PUSHED KEY
0153	C2 4D 01		JNZ KIND1	;IF KEY STILL PUSHED
0156	CD E9 01	KIND2:	CALL DCD	;UPDATE DISP AND WAIT
0159	CD 85 01		CALL KPU	;SHK FOR PUSHED KEY
015C	CA 56 01		JZ KIND2	;IF KEY NOT PUSHED
015F	21 E8 0B		LXI H,UDKY	;ADRS OF FIRST KEY ROW SCAN
0162	16 FF		MVI D,0FFH	;LOAD ROW COUNTER TO 0-1
0164	7E	KIND3:	MOV A,M	;GET ROW N KEY DATA

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0165	FE F7		CPI 0F7H	;IS IT THE HDWR STEP KEY?
0167	CA 82 01		JZ KIND5	;YES JUMPS
016A	2F		CMA	;INVERT KEY DATA
016B	2C		INR L	;NEXT ROW
016C	14		INR D	;NEXT TABLE BLOCK
016D	A7		ANA A	;TEST ROW N FOR 0
016E	CA 64 01		JZ KIND3	;JUMP IF KEY NOT PUSHED
0171	FE 04		CPI 4	;SEE IF D3=1
0173	C2 77 01		JNZ KIND4	;IF SO
0176	3D		DCR A	;ELSE SET A=3
0177	82	KIND4:	ADD D	;ADD 3X THE ROW N TO
0178	82		ADD D	;GET THE TABLE OFFSET
0179	82		ADD D	
017A	5F		MOV E,A	;STORE TABLE INDEX
017B	16 00		MVI D,0	;CLEAR MS BYTE OF DE
017D	21 AF 01		LXI H,KIT-1	;ADRS OF KEY CODE TABLE
0180	19		DAD D	;ADD INDEX TO TABLE ADRS
0181	7E		MOV A,M	;PUT KEY CODE IN A
0182	E1		KIND5:	POP H
0183	D1		POP D	
0184	C9		RET	
;				
;DETERMINES IF ANY KEY IS PUSHED				
0185	C5	KPU:	PUSH B	
0186	CD 9A 01		CALL KRD	;READ THE KEYBOARD
0189	06 08		MVI B,8	;SET THE LOOP COUNTER
018B	21 E8 0B		LXI H,UDKY	;ADDRESS OF UNDECODED KEY SCAN
018E	3E FF		MVI A,0FFH	;UNPUSHED KEY CODE
0190	A6	KPU1:	ANA M	;ANY PUSHED KEY CODE CHANGE A
0191	2C		INR L	;NEXT RAM KEY ROW
0192	05		DCR B	;LOOP COUNTER
0193	C2 90 01		JNZ KPU1	;IF KEY ROWS NOT ANDED WITH A

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0196	FE FF		CPI 0FFH	;SET FLAG IF ALL KEYS NOT PUSHED
0198	C1		POP B	
0199	C9		RET	
;				
;READS KEYS AND STORES THEM IN RAM (UDKY)				
019A	21 E8 0B	KRD:	LXI H,UDKY	;ADRS OF UNDECODED KEY SCAN
019D	3E FF		MVI A,0FFH	;BLANK DISPLAY CODE
019F	D3 38		OUT DSP	;CLEAR DISPLAY
01A1	3D		DCR A	;SET A TO 1111 1110 SCAN POINTER
01A2	37		STC	;TO PRESET END OF SCAN LOOP FLAG
01A3	D3 28	KRD1:	OUT SCAN	;SCAN ONE KEY ROW
01A5	47		MOV B,A	;SAVE SCAN POINTER
01A6	DB 18		IN KEY	;INPUT A KEY ROW
01A8	77		MOV M,A	;STORE IT IN RAM
01A9	78		MOV A,B	;RESTORE SCAN POINTER
01AA	2C		INR L	;POINT TO NEXT RAM ADRS
01AB	17		RAL	;MOVE SCAN POIN. TO NEXT KEY ROW
01AC	DA A3 01		JC KRD1	;IF LAST KEY ROW NOT SCANNED
01AF	C9		RET	
;				
;KEY INPUT DECODE TABLE (HDWR STEP IS F7)				
01B0	86	KIT:	DB 86H	;INSTR STEP KEY CODE
01B1	85		DB 85H	;FETCH PC KEY CODE
01B2	00		DB 0	;(UNDEFINED) KEY CODE
01B3	84		DB 84H	;RUN KEY CODE
01B4	80		DB 80H	;FETCH REG KEY CODE
01B5	82		DB 82H	;FETCH ADRS KEY CODE
01B6	00		DB 0	;0 KEY CODE
01B7	83		DB 83H	;STORE/INCR KEY CODE
01B8	81		DB 81H	;KEY CODE
01B9	01		DB 1	;1 KEY CODE
01BA	02		DB 2	;2 KEY CODE

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
01BB	03		DB 3	;3 KEY CODE
01BC	04		DB 4	;4 KEY CODE
01BD	05		DB 5	;5 KEY CODE
01BE	06		DB 6	;6 KEY CODE
01BF	07		DB 7	;7 KEY CODE
01C0	08		DB 8	;8 KEY CODE
01C1	09		DB 9	;9 KEY CODE
01C2	0A		DB 0AH	;A KEY CODE
01C3	0B		DB 0BH	;B KEY CODE
01C4	0C		DB 0CH	;C KEY CODE
01C5	0D		DB 0DH	;D KEY CODE
01C6	0E		DB 0EH	;E KEY CODE
01C7	0F		DB 0FH	;F KEY CODE
;				
;SCAN DISPLAY SEGMENTS				
01C8	F5	SDS:	PUSH PSW	
01C9	E5		PUSH H	
01CA	C5		PUSH B	
01CB	21 FF 0B		LXI H,DDSP5	;ADRS OF DECODED DISP DIGIT 5
01CE	06 20		MVI B,20H	;DISP DIGIT 5 SCAN POINTER
01D0	AF	SDS1:	XRA A	;CLEAR A
01D1	D3 28		OUT SCAN	;TURN DISP DIGIT OFF
01D3	7E		MOV A,M	;GET SEGMENT DATA
01D4	D3 38		OUT DSP	;STORE IT IN DSP LATCH
01D6	78		MOV A,B	;GET SCAN DIGIT POINTER
01D7	D3 28		OUT SCAN	;TURN ON DISP DIGIT
01D9	CD 29 04		CALL DELA	;STRETCH DIGIT IMS
01DC	2D		DCR L	;ADRS OF NEXT DIGIT IN RAM
01DD	1F		RAR	;NEXT SCAN DIGIT POINTER
01DE	47		MOV B,A	;SAVE IT IN B
01DF	D2 D0 01		JNC SDS1	;IF NOT LAST SCANNED DIGIT (LSD)
01E2	2F		CMA	;SET A=FF BLANK DISP CODE

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
01E3	D3 38		OUT DSP	;TURN OFF DSP DIGITS
01E5	C1		POP B	
01E6	E1		POP H	
01E7	F1		POP PSW	
01E8	C9		RET	
;				
;DISPLAY CHARACTER DECODER				
01E9	F5	DCD:	PUSH PSW	
01EA	C5		PUSH B	
01EB	D5		PUSH D	
01EC	E5		PUSH H	
01ED	01 FA 0B		LXI B,DDSP0	;DECODED DIGIT FIRST ADRS
01F0	11 F0 0B		LXI D,UDSP0	;UNDECODED DIGIT FIRST ADRS
01F3	21 18 02	DCD1:	LXI H,DCC	;DISPLAY CODE CONVERTER TABLE ADRS
01F6	1A		LDAX D	;GET UNDECODED DATA FOR OFFSET
01F7	D5		PUSH D	;SAVE ITS ADRS
01F8	5F		MOV E,A	;TABLE OFFSET VALUE TO E
01F9	16 00		MVI D,0	;CLEAR D
01FB	19		DAD D	;ADD OFFSET VALUE TO DCC ADRS
01FC	7E		MOV A,M	;GET DECODED DATA FROM TAB. ADRS
01FD	02		STAX B	;STORE IT IN DECODED RAM
01FE	D1		POP D	;RESTORE UDSPX ADRS
01FF	1C		INR E	;POINT TO NEXT UPSP ADRS
0200	0C		INR C	;POINT TO NEXT DDSP ADRS
0201	C2 F3 01		JNZ DCD1	;IF NOT LAST DIGIT
0204	21 FA 0B		LXI H,DDSP0	;DECODED DIGIT 0
0207	1A		LDAX D	;UDSP6 DATA MODIFY FLAG
0208	A7		ANA A	;CHECK FOR SET FLAG
0209	CA 10 02		JZ DCD2	;IF DATA NOT BEING MODIFIED
020C	7E		MOV A,M	;GET DDSP0 DATA
020D	E6 7F		ANI 7FH	;SET ITS DECIMAL POINT DISP BIT
020F	77		MOV M,A	;STORE IT IN DDSP0

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0210	E1	DCD2:	POP H;	
0211	D1		POP D;	
0212	C1		POP B;	
0213	F1		POP PSW;	
0214	CD C8 01		CALL SDS	;UPDATE DISPLAY
0217	C9		RET	;
;				
;DISPLAY CODE CONVERTER TABLE				
0218	C0	DCC:	DB 0C0H	;0
0219	F9		DB 0F9H	;1
021A	A4		DB 0A4H	;2
021B	B0		DB 0B0H	;3
021C	99		DB 099H	;4
021D	92		DB 092H	;5
021E	82		DB 082H	;6
021F	F8		DB 0F8H	;7
0220	80		DB 080H	;8
0221	90		DB 090H	;9
0222	88		DB 088H	;A
0223	83		DB 083H	;B
0224	C6		DB 0C6H	;C
0225	A1		DB 0A1H	;D
0226	86		DB 086H	;E
0227	8E		DB 08EH	;F
0228	FF		DB 0FFH	;BLANK
0229	89		DB 089H	;H
022A	C7		DB 0C7H	;L
022B	E3		DB 0E3H	;U SMALL
022C	8C		DB 08CH	;P
022D	A3		DB 0A3H	;O
022E	C1		DB 0C1H	;U LARGE
022F	F7		DB 0F7H	;_

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0230	A7		DB 0A7H	;C SMALL
0231	CF		DB 0CFH	;1 LEFT
0232	00		DB 000H	;ALL SEGS
0233	AF		DB 0AFH	;R SMALL
0234	BF		DB 0BFH	;-
;				
;STDM STORES DISP MESSAGE AT DE ADRS IN UDSP RAM				
0235	06 06	SDM:	MVI B,6	;LOOP COUNTER FOR 6 DISPLAY DIGITS
0237	1A	SDM1:	LDAX D	;DISPLAY CHARACTER
0238	77		MOV M,A	;STORE IT IN UPSPX IN RAM
0239	2C		INR L	;NEXT UDSP ADRS
023A	13		INX D	;NEXT MESSAGE TABLE ADRS
023B	05		DCR B	;LOOP COUNTER
023C	C2 37 02		JNZ SDM1	;IF NOT LAST DIGIT
023F	C1		POP B	;
0240	C9		RET	
;				
;DISPLAY MESSAGE TABLES				
0241	14	DMT:	DB 14H	;P
0242	16		DB 16H	;U
0243	0B		DB 0BH	;B
0244	0A		DB 0AH	;A
0245	12		DB 12H	;L
0246	13		DB 13H	;U SMALL
0247	10	FETCH:	DB 10H	;SP
0248	10		DB 10H	;SP
0249	17		DB 17H	;_
024A	17		DB 17H	;_
024B	17		DB 17H	;_
024C	17		DB 17H	;_
024D	0A	MA:	DB 0AH	;A
024E	10		DB 10H	;SP

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
024F	10		DB 10H	;SP
0250	10		DB 10H	;SP
0251	12	FLG:	DB 12H	;L
0252	0F		DB 0FH	;F
0253	10		DB 10H	;SP
0254	10		DB 10H	;SP
0255	0B	MB:	DB 0BH	;B
0256	10		DB 10H	;SP
0257	10		DB 10H	;SP
0258	10		DB 10H	;SP
0259	0C	MC:	DB 0CH	;C
025A	10		DB 10H	;SP
025B	10		DB 10H	;SP
025C	10		DB 10H	;SP
025D	0D	MD:	DB 0DH	;D
025E	10		DB 10H	;SP
025F	10		DB 10H	;SP
0260	10		DB 10H	;SP
0261	0E	ME:	DB 0EH	;E
0262	10		DB 10H	;SP
0263	10		DB 10H	;SP
0264	10		DB 10H	;SP
0265	11	MH:	DB 11H	;H
0266	10		DB 10H	;SP
0267	10		DB 10H	;SP
0268	10		DB 10H	;SP
0269	12	ML:	DB 12H	;L
026A	10		DB 10H	;SP
026B	10		DB 10H	;SP
026C	10		DB 10H	;SP
026D	11	SPH:	DB 11H	;H
026E	14		DB 14H	;P

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
026F	05		DB 05H	;S
0270	10		DB 10H	;SP
0271	12	SPL:	DB 12H	;L
0272	14		DB 14H	;P
0273	05		DB 05H	;S
0274	10		DB 10H	;SP
0275	11	PCH:	DB 11H	;H
0276	0C		DB 0CH	;C
0277	14		DB 14H	;P
0278	10		DB 10H	;SP
0279	12	PCL:	DB 12H	;L
027A	0C		DB 0CH	;C
027B	14		DB 14H	;P
027C	10		DB 10H	;SP
027D	19	IM:	DB 19H	;I
027E	10		DB 10H	;SP
027F	10		DB 10H	;SP
0280	10		DB 10H	;SP
0281	1A	ALL:	DB 1AH	;ALL
0282	1A		DB 1AH	;ALL
0283	1A		DB 1AH	;ALL
0284	1A		DB 1AH	;ALL
0285	1A		DB 1AH	;ALL
0286	1A		DB 1AH	;ALL
0287	10	ICX:	DB 10H	;SP
0288	10		DB 10H	;SP
0289	10		DB 10H	;SP
028A	0C		DB 0CH	;C
028B	01		DB 01H	;I
028C	10		DB 10H	;SP
028D	1B	PPM:	DB 1BH	;R SMALL
028E	0E		DB 0EH	;E

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
028F	14		DB 14H	;P
0290	05		DB 05H	;S
0291	10	BLNKM:	DB 10H	;SP BLANK CHARACTER
0292	10		DB 10H	;SP
0293	10		DB 10H	;SP
0294	10		DB 10H	;SP
0295	10		DB 10H	;SP
0296	10		DB 10H	;SP
;				
;BLANK THE DISPLAY				
0297	11 91 02	BLNK:	LXI D,BLNKM;	ADRS OF BLANK MESSAGE TABLE
029A	DF		RST 3	;GET MESSAGE
029B	CD E9 01		CALL DCD	;SEND TO DISPLAY
029E	C9		RET	
;				
;CONTROL KEY JUMP TABLE				
029F	FE F7	CHDSS:	CPI 0F7H	;HARDWARE SINGLE STEP PUSHED?
02A1	CA F7 03		JZ HDSS	;HARDWARE SINGLE STEP ROUTINE
02A4	FE 86	CINSS:	CPI 86H	;SOFTWARE SINGLE STEP PUSHED?
02A6	CA F1 03		JZ INSS	;SOFTWARE SINGLE STEP ROUTINE
02A9	FE 84	CRUN:	CPI 84H	;RUN PUSHED?
02AB	CA C2 03		JZ RUN	;RUN ROUTINE
02AE	FE 83	CSTRM:	CPI 83H	;STORE MEMORY PUSHED?
02B0	CA AF 03		JZ STRM	;STORE MEMORY ROUTINE
02B3	FE 81	CDCRM	CPI 81H	;DECREMENT MEMORY PUSHED?
02B5	CA AA 03		JZ DCRM	;DECREMENT MEMORY ROUTINE
02B8	FE 82	CFETA:	CPI 82H	;FETCH ADRS PUSHED?
02BA	CA 28 03		JZ FETA	;FETCH ADRESS ROUTINE
02BD	FE 85	CFETP:	CPI 85H	;FETCH PROGRAM COUNTER PUSHED?
02BF	CA 41 01		JZ TRP7	;FETCH PROGRAM COUNTER ROUTINE
02C2	FE 80	CFETR:	CPI 80H	;FETCH REGISTER PUSHED?
02C4	CA D5 02		JZ FETR	;FETCH REGISTER ROUTINE

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
02C7	C9		RET	;IF NO LEGAL KEY WAS PUSHED
02C8	FE 83	CSTRR:	CPI 83H	;STORE REGISTER PUSHED?
02CA	CA 13 04		JZ STRR	;STORE REGISTER ROUTINE
02CD	FE 81	CDCRR:	CPI 81H	;DECREMENT REGISTER PUSHED?
02CF	CA FD 03		JZ DCRR	;DECREMENT REGISTER ROUTINE
02D2	C3 B8 02		JMP CFETA	
;				
;FETCH REGISTER MODE				
02D5	E1	FETR:	POP H	;UNDO STACK CALL
02D6	21 4D 02		LXI H,MA	;A REG MESSAGE ADRS
02D9	01 E7 0B		LXI B,SAVA	;ADRS OF USER A REG CONTENTS
02DC	2B	FETR1:	DCX H	;6-2=4 CHARACTERS IN REG MESSAGE
02DD	2B		DCX H	
02DE	22 F8 0B	FETR2:	SHLD RMP	;STORE IN REGISTER MESSAGE POINTER
02E1	EB		XCHG	;PUT MESSAGE ADRS IN DE
02E2	DF		RST 3	;STORE MESSAGE IN RAM
02E3	0A		LDAX B	;USERS REG CONTENTS
02E4	5F		MOV E,A	;TRANSFER IT TO E
02E5	21 F1 0B		LXI H,UDSP1	;ADRS WHERE DISPLAY REG DATA GOES
02E8	CD 9C 03		CALL FETA7	;FORMAT REG BYTE DATA + STORE IT
02EB	3E DB		MVI A,0DBH	;LS BYTE OF SAVIM ADRS
02ED	B9		CMP C	;IS REG EXAMINED THE INTRPT MASK?
02EE	CA 10 03		JZ FETR6	;IF IT IS- HEX KEY INPUT NOT LEGAL
02F1	CD 4B 01	FETR3:	CALL KIND	;INPUT KEYS
02F4	CD C8 02		CALL CSTRR	;LOOK FOR CONTROL
02F7	D2 F1 02		JNC FETR3	;IF NOT A HEX KEY OR NEW CONTROL
02FA	5F		MOV E,A	;SAVE 1ST HEX KEY IN E
02FB	2C		INR L	;POINT TO UDSP1
02FC	36 00		MVI M,0	;CLEAR IT TO DISPLAY A 0
02FE	2D	FETR4:	DCR L	;POINT BACK TO UDSP0
02FF	73		MOV M,E	;STORE NEW HEX KEY THERE
0300	CD 19 03	FETR5:	CALL DPS	;TO SET DP AND INPUT KEYS

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0303	CD C8 02		CALL CSTRR	;LOOK FOR CONTROL
0306	D2 00 03		JNC FETR5	;IF NOT A HEX KEY OR NEW CONTROL
0309	2C		INR L	;POINT TO UDSP1
030A	53		MOV D,E	;PUT OLD HEX CHARACTER INTO D
030B	5F		MOV E,A	;PUT NEW HEX CHARACTER INTO E
030C	72		MOV M,D	;STORE OLD HEX CHAR. INTO DSP1
030D	C3 FE 02		JMP FETR4	;CONTINUE
0310	CD 4B 01	FETR6:	CALL KIND	;INPUT KEYS AND UPDATE DISPLAY
0313	CD C8 02		CALL CSTRR	;LOOK FOR CONTROL ONLY
0316	C3 10 03		JMP FETR6	;KEEP LOOKING
;				
;DECIMAL POINT SET				
0319	3E 01	DPS:	MVI A,1	;FLAG SET DATA
031B	32 F6 0B		STA UDSP6	;SET DATA MODIFY FLAG
031E	CD 4B 01		CALL KIND	;GET ANOTHER KEY
0321	F5		PUSH PSW	;SAVE KEY CODE
0322	AF		XRA A	;CLEAR A
0323	32 F6 0B		STA UDSP6	;CLEAR DATA MODIFY FLAG
0326	F1		POP PSW	;RECOVER KEY CODE
0327	C9		RET	
;				
;FETCH MEMORY ADDRESS				
0328	D1	FETA:	POP D	;UNDO STACK CALL
0329	11 47 02	FETAR:	LXI D,FETCH	;DISPLAY MESSAGE ADRS
032C	DF		RST 3	;STORE MESSAGE IN RAM
032D	0E 04		MVI C,4	;ADRS DIGIT COUNTER
032F	CD 4B 01	FETA1:	CALL KIND	;READ KEYS AND SCAN DISPLAY
0332	CD B8 02		CALL CFETA	;LOOK FOR CONTROL
0335	D2 2F 03		JNC FETA1	;IF NOT A HEX OR NEW CONTROL
0338	21 F6 0B		LXI H,UDSP6	;ADRS OF DISPLAY DIGIT 4+2
033B	47		MOV B,A	;SAVE HEX KEY INPUT
033C	FE 01		CPI 1	;1 KEY PUSHED?

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
033E	C2 4A 03		JNZ NCTL	;IF NOT
0341	3E 04		MVI A,4	;MS ADRS POSITION VALUE
0343	B9		CMP C	;ADRS POSITION POINTER
0344	C2 4A 03		JNZ NCTL	;IF 1 KEY NOT MS ADRS BYTE
0347	C3 29 03		JMP FETAR	;WAIT FOR ANOTHER KEY
034A	78	NCTL:	MOV A,B	;RESTORE NON-1 KEY VALUE
034B	06 04		MVI B,4	;DISPLAY POSITION COUNTER
034D	2D	FETA2:	DCR L	;POINT TO DISP DIGIT ON RIGHT
034E	2D		DCR L	;POINT TO DISP DIGIT ON RIGHT
034F	56		MOV D,M	;PUT THIS CHARACTER IN D
0350	2C		INR L	;POINT TO DISP DIGIT ON LEFT
0351	72		MOV M,D	;STORE THE DIGIT SHIFTED 1 TO LEFT
0352	05		DCR B	;POSITION COUNTER
0353	C2 4D 03		JNZ FETA2	;IF NOT DONE ENTERING ADRS
0356	77		MOV M,A	;KEY CODE TO DISP DIGIT 2 ADRS
0357	0D		DCR C	;DIGIT COUNTER
0358	C2 2F 03		JNZ FETA1	;IF NOT DONE
035B	CD 93 03		CALL FETA6	;MERGE LS ADRS BYTE IN DISP TO A
035E	4F		MOV C,A	;STORE MERGED BYTE IN C
035F	2C		INR L	;POINT TO MS ADRS BYTE IN DISP
0360	CD 93 03		CALL FETA6	;MERGE IT IN A
0363	47		MOV B,A	;STORE IT IN C
0364	21 F5 0B	FETA3:	LXI H,UDSP5	;ADRS OF DISP DIGIT 5
0367	58		MOV E,B	;PUT MS ADRS BYTE IN E
0368	CD 9C 03		CALL FETA7	;SEPARATE AND STORE IT IN RAM
036B	2D		DCR L	;POINT TO UDSP3
036C	59		MOV E,C	;SAVE LS ADRS BYTE
036D	CD 9C 03		CALL FETA7	;SEPARATE AND STORE IT IN RAM
0370	2D		DCR L	;POINT TO UDSP1
0371	0A		LDAX B	;GET DATA AT FETCH ADRS
0372	5F		MOV E,A	;PUT IT IN E
0373	CD 9C 03		CALL FETA7	;SEPARATE IT AND DISP IT AS DATA

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
;				
;MODIFY THE DATA				
0376	CD 4B 01		CALL KIND	;GET A KEY
0379	CD 9F 02		CALL CHDSS	;LOOK FOR ANY CONTROL KEY
037C	5F		MOV E,A	;STORE HEX KEY IN E
037D	77		MOV M,A	;AND UDSP0
037E	2C		INR L	;POINT TO UDSP1
037F	36 00		MVI M,0	;CLEAR IT
0381	2D	FETA4:	DCR L	;POINT TO UDSP0
0382	73		MOV M,E	;STORE LATEST KEY ENTRY THERE
0383	CD 19 03	FETA5:	CALL DPS	;TO SET DP AND INPUT KEYS
0386	CD AE 02		CALL CSTRM	;LOOK FOR CONTROL KEY
0389	D2 83 03		JNC FETA5	;IF NOT A VALID KEY
038C	2C		INR L	;POINT TO UDSP1
038D	53		MOV D,E	;LAST KEY ENTRY
038E	5F		MOV E,A	;NEW KEY ENTRY
038F	72		MOV M,D	;LAST ENTRY SHIFTED TO UDSP1
0390	C3 81 03		JMP FETA4	;UPDATE NEW KEY AND CONTINUE
;				
;MERGES 2 HEX NUMBERS INTO A REG				
0393	5E	FETA6:	MOV E,M	;LS HEX CHAR
0394	23		INX H	;POINT TO MS HEX CHAR
0395	7E		MOV A,M	;MS HEX CHAR
0396	07		RLC	;MOVE IT TO 4 MS BITS IN A
0397	07		RLC	;AND CLEAR 4 LS BITS IN A
0398	07		RLC	;
0399	07		RLC	;
039A	B3		ORA E	;MERGE MS AND LS HEX CHARS IN A
039B	C9		RET	
;				
;SEPARATES 2 HEX CHARACTERS IN A + STORES IN M				
039C	7B	FETA7:	MOV A,E	;PUT MERGED HEX CHARS INTO A

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
039D	0F		RRC	;MOVE MS HEX CHAR TO 4 LS BITS A
039E	0F		RRC	;
039F	0F		RRC	;
03A0	0F		RRC	;
03A1	16 0F		MVI D,0FH	;MASK TO CLEAR 4 MS BITS OF A
03A3	A2		ANA D	;DO IT
03A4	77		MOV M,A	;STORE MS HEX CHAR
03A5	2D		DCR L	;LS CHAR DESTINATION ADRS
03A6	7B		MOV A,E	;GET MERGED HEX CHARS
03A7	A2		ANA D	;CLEAR MS HEX CHAR
03A8	77		MOV M,A	;STORE LS HEX CHAR
03A9	C9		RET	
;				
;DECREMENTS MEMORY ADRS				
03AA	0B	DCRM:	DCX B	;POINT TO NEXT LOWER ADRS
03AB	E1		POP H	;UNDO STACK CALL
03AC	C3 64 03		JMP FETA3	;TO FETCH NEW ADRS DATA
;				
;STORES DATA IN MEM AND INCREMENTS ADRS				
03AF	D1	STRM:	POP D	;UNDO STACK CALL
03B0	CD 93 03		CALL FETA6	;MERGE 2 HEX DATA VALUES INTO A
03B3	5F		MOV E,A	;SAVE IN E
03B4	02		STAX B	;STORE DATA BYTE IN RAM ADRS IN B
03B5	0A		LDAX B	;READ IT BACK
03B6	BB		CMP E	;IS IT THE SAME? DID IT STORE?
03B7	03		INX B	;POINT TO NEXT RAM ADRS
03B8	CA 64 03		JZ FETA3	;FETCH NEW ADRS IF LAST STORE OK
03BB	0B		DCX B	;ELSE POINT BACK TO LAST RAM ADRS
03BC	C5		PUSH B	;SAVE THIS ADRS
03BD	D7		RST 2	;BEEP A FAIL TO STORE ERROR
03BE	C1		POP B	;RESTORE RAM ADRS
03BF	C3 64 03		JMP FETA3	;AND FETCH IT AGAIN

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
;				
;RUNS THE PROGRAM STARTING AT ADRS-IN DISPLAY				
03C2	21 32 01	RUN:	LXI H,0132H	;INSTR CODES FOR STA 01XX (RUN)
03C5	22 D5 0B	RUN1:	SHLD RAML1	;JUMP LINK IN RAM
03C8	21 10 C3		LXI H,0C310H	;INSTR CODES FOR 00 AND JMP XXXX
03CB	22 D7 0B		SHLD RAML2	;JUMP LINK IN RAM
03CE	31 DB 0B		LXI SP,SAVIM	;USER PROG START ADRS LINK+1
03D1	C5		PUSH B	;STORE USER START ADRS IN RAM LINK
03D2	21 DF 0B		LXI H,SAVSH	;RAM ADRS OF MS BYTE
03D5	36 0B		MVI M,0BH	;MSBYTE USER SP
03D7	2B		DCX H	;RAM ADRS OF LSBYTE USER SP
03D8	7E		MOV A,M	;LSBYTE USER SP
03D9	FE 40		CPI 40H	;<=40
03DB	D2 E0 03		JNC RUN2	;IF AVAIL STACK SPACE
03DE	36 B0		MVI M,0B0H	;IF NOT, RESET POINTER
03E0	31 E2 0B	RUN2:	LXI SP,SAVE	;PREPARE TO RESTORE USER REGS
03E3	D1		POP D	;RESTORE D-E
03E4	C1		POP B	;RESTORE B,L
03E5	F1		POP PSW	;RESTORE PSW,A
03E6	31 DE 0B		LXI SP,SAVSL	;RAM ADRS OF USER SP
03E9	E1		POP H	;PUT SPH-L IN H-L
03EA	F9		SPHL	;TRANSFER SP VAL TO CPU SP
03EB	2A E0 0B		LHLD SAVL	;RESTORE H-L
03EE	C3 D5 0B		JMP RAML1	;LINK TO USER PROGRAM
;				
;INSTRUCTIONS SINGLE STEP AND RETURN TO MONITOR				
03F1	21 32 06	INSS:	LXI H,0632H	;INSTR CODES FOR STA 06XX (INSS)
03F4	C3 C5 03		JMP RUN1	;SET LINKS,RESTORE REGS,USER PROG
;				
;HARDWARE SINGLE STEP ONE LINE OF CODE				
03F7	21 32 03	HDSS:	LXI H,0332H	;INSTR CODES FOR STA 03XX (HDSS)
03FA	C3 C5 03		JMP RUN1	;SET LINKS,RESTORE REGS,USER PROG

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
;				
;DECREMENTS REGISTER DISPLAYED				
03FD	D1	DCRR:	POP D	;UNDO STACK CALL
03FE	03		INX B	;POINT TO LAST SAVX REG RAM ADRS
03FF	2A F8 0B		LHLD RMP	;REGISTER MESSAGE POINTER
0402	2B		DCX H	;SUBTRACT 2 FROM IT
0403	2B		DCX H	;
0404	3E E8		MVI A,0E8H	;LS BYTE OF SAVA+1 ADRS IN RAM
0406	B9		CMP C	;SEE IF IT'S IM REG
0407	C2 DC 02		JNZ FETR1	;FETCH NEW NON-IM REG+DISP IF NOT
040A	01 DB 0B		LXI B,SAVIM	;ADRS OF IM VALUE IN RAM
040D	21 7D 02		LXI H,IM	;ADRS OF IM DISP MESSAGE
0410	C3 DC 02		JMP FETR1	;FETCH IM REG + DISP IF IT IS IM
;				
;STORES REGISTER DATA AND INCREMENTS				
0413	D1	STRR:	POP D	;UNDO STACK CALL
0414	CD 93 03		CALL FETA6	;MERGE 2 HEX DATA VALUES IN A
0417	02		STAX B	;STORE DATA BYTE IN SAVX REG ADRS
0418	0B		DCX B	;POINT TO NEXT SAVX REG ADRS
0419	2A F8 0B		LHLD RMP	;REGISTER MESSAGE POINTER
041C	23		INX H	;POINT TO NEXT REG MESSAGE
041D	23		INX H	;
041E	23		INX H	;
041F	23		INX H	;
0420	3E DA		MVI A,0DAH	;LS BYTE OF SAVIM-1 ADRS IN RAM
0422	B9		CMP C	;SEE IF IT'S A REG
0423	CA D5 02		JZ FETR	;FETCH A REG + DISP IF IT IS
0426	C3 DE 02		JMP FETR2	;FETCH NEXT REG + DISP IF NOT
;				
;DELAYS APPROX 1MS				
0429	C5	DELA:	PUSH B	
042A	01 01 00		LXI B,0001H	;FIXED 1 MS VALUE

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
042D	C3 31 04		JMP DEL1	;START TIMING LOOP
;				
;DELAYS APPROX 1MS TIMES VALUE IN BC				
0430	C5	DELB:	PUSH B	
0431	F5	DEL1:	PUSH PSW	
0432	AF		XRA A	;CLEAR A
0433	D5		PUSH D	
0434	16 80	DEL2:	MVI D,TIME	;1MS SMALL LOOP TIME CONSTANT
0436	15	DEL3:	DCR D	;1MS LOOP
0437	C2 36 04		JNZ DEL3	;IF NOT 1MS WORTH OF COUNTS
043A	0B		DCX B	;LARGE LOOP COUNTER
043B	B8		CMP B	;MS BYTE=0?
043C	C2 34 04		JNZ DEL2	;IF NOT, LOOP FOR 1 MORE MS
043F	B9		CMP C	;LS BYTE=0?
0440	C2 34 04		JNZ DEL2	;IF NOT, LOOP
0443	D1		POP D	;WHEN DONE
0444	F1		POP PSW	
0445	C1		POP B	
0446	C9		RET	
;				
;BEEP PRODUCES A FIXTED TONE FREQ. + DURATION				
0447	2E FF	BEEP2:	MVI L,0FFH	;DURATION MULTIPLIER
0449	26 00		MVI H,0	;SPEAKER FLAG
044B	48		MOV C,B	;SET COUNT REG B
044C	5A		MOV E,D	;SET COUNT REG E
044D	E5		PUSH H	;INIT. DONE FLAG
044E	0D	BEEP3:	DCR C	;DECR FREQ
044F	C2 80 04		JNZ BEEP8	;
0452	48		MOV C,B	;RESTORE FREQ
0453	7C		MOV A,H	;CHG SPKR FLAG
0454	2F		CMA	;
0455	B7		ORA A	;

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0456	67		MOV H,A	;
0457	C2 60 04		JNZ BEEP4	;TEST SPKR FLAG
045A	3E C0		MVI A,0C0H	;TURN SPKR OFF
045C	30		SIM	;
045D	C3 64 04		JMP BEEP5	;
0460	3E 40	BEEP4:	MVI A,40H	;TURN SPKR ON
0462	30		SIM	;
0463	BE		CMP M	;TIME DELAY INSTR
0464	F1	BEEP5:	POP PSW	;GET DONE FLAG
0465	F5		PUSH PSW	;
0466	B7		ORA A	;
0467	CA 6F 04		JZ BEEP6	;CONTINUE IF NOT DONE
046A	7C		MOV A,H	;RETURN IF SPKR OFF
046B	B7		ORA A	;
046C	CA 7E 04		JZ BEEP7	;
046F	1D	BEEP6:	DCR E	;DURATION CNTR
0470	C2 88 04		JNZ BEEP9	;
0473	5A		MOV E,D	;RESTORE DURATION
0474	2D		DCR L	;TIME COUNT
0475	C2 4E 04		JNZ BEEP3	;
0478	F1		POP PSW	;GET DONE FLAG
0479	2F		CMA	;
047A	F5		PUSH PSW	;SET DONE FLAG
047B	C3 4E 04		JMP BEEP3	;
047E	F1	BEEP7:	POP PSW	;
047F	C9		RET	;
0480	E3	BEEP8:	XTHL	;DELAY 81 CYCLES
0481	E3		XTHL	;
0482	E3		XTHL	;
0483	E3		XTHL	;
0484	BE		CMP M	;
0485	C3 6F 04		JMP BEEP6	;

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0488	00	BEEP9:	NOP	;DELAY 14 CYCLES
0489	00		NOP	;
048A	C3 4E 04		JMP BEEP3	;
;				
;SIGNATURE ANALYSIS TEST LOOP				
048D	F3	SATL1:	DI	;TURN OFF INTERRUPTS
048E	DB 80		IN 80H	;PULSE A15 READ START-STOP LINE
0490	D3 80		OUT 80H	;PULSE A15 WRITE START-STOP LINE
0492	31 CE 0B		LXI SP,0BCEH	;SET TO MONITOR VALUE
;				
;RAM PROGRAM TEST				
0495	D3 11		OUT 11H	;SET RAM PROTECT
0497	32 11 0B		STA 0B11H	;WRITE TO UNPROTECTED RAM
049A	32 11 09		STA 0911H	;WRITE TO PROTECTED RAM
049D	D3 10		OUT 10H	;UNPROTECTED RAM
;				
;OUTPUT PORT TEST				
049F	AF		XRA A	;CLEAR A
04A0	37		STC	;SET CARRY BIT TO 1 FOR 8 LOOPS
04A1	17	SATL2:	RAL	;MOVE 1 BIT TO LEFT
04A2	D3 30		OUT LOUT	;OUTPUT PORT LEDS
04A4	D2 A1 04		JNC SATL2	;IF NOT DONE
;				
;DISPLAY LATCH TEST				
04A7	17	SATL3:	RAL	;MOVE 1 BIT TO LEFT
04A8	D3 38		OUT DSP	;OUTPUT DISPLAY SEGMENTS
04AA	D2 A7 04		JNC SATL3	;IF NOT DONE
;				
;SCAN LATCH TEST				
04AD	2F		CMA	;SET A TO FF
04AE	3F		CMC	;CLEAR CARRY
04AF	17	SATL4:	RAL	;MOVE 0 BIT TO LEFT

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
04B0	D3 28		OUT SCAN	;OUTPUT SCAN LINES
04B2	CD 29 04		CALL DELA	;STRETCH EACH DISP DIGIT
04B5	DA AF 04		JC SATL4	;IF NOT DONE
;				
;SPEAKER SERIAL OUT TEST				
04B8	3E 40		MVI A,40H	;SPEAKER ON MASK
04BA	30		SIM	;TURN SPKR ON
04BB	3E C0		MVI A,0C0H	;SPKR OFF MASK
04BD	30		SIM	;TURN SPKR OFF
;				
;KEY INPUT TEST				
04BE	AF		XRA A	;CLEAR A
04BF	D3 28		OUT SCAN	;ALL SCAN LINES TO LOGIC 0
04C1	DB 18		IN KEY	;RESPOND TO ALL KEYS
;				
;INPUT PORT TEST				
04C3	DB 20		IN SIN	;INPUT THE INPUT SWITHES
;				
;INTERRUPT KEY TEST				
04C5	21 10 00		LXI H,BEEP	;ADRS OF BEEP ROUTINE
04C8	22 FD 0A		SHLD 0AFDH	;STORE IT IN INTRPT RAM LINK 6.5
04CB	3E C3		MVI A,0C3H	;JUMP OP CODE
04CD	32 FC 0A		STA 0AFCH	;STORE IT IN INTRPT RAM LINK 6.5
04D0	3E 1D		MVI A,1DH	;INTRPT MAST FOR RST 6.5
04D2	30		SIM	;SET MASK
04D3	FB		EI	;ENABLE INTERRUPTS
04D4	C3 8D 04		JMP SATL1	;LOOP BACK TO START OVER AGAIN
;				
;PRESENTS INPUT SWITCH DATA ON OUTPUT LEDS				
04D7	3A 00 20	ECHO:	LDA 2000H	;READ INPUT PORT SWITHES
04DA	32 00 30		STA 3000H	;WRITE TO OUTPUT PORT LEDS
04DD	C3 D7 04		JMP ECHO	;REPEAT

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
;				
;AND GATE PROGRAM				
04E0	3A 00 20	ANDGT:	LDA 2000H	;READ INPUT PORT
04E3	FE FF		CPI 0FFH	
04E5	CA F0 04		JZ ON	;JUMP IF ALL BITS ARE ONE
04E8	3E FF		MVI A,0FFH	
04EA	32 00 30		STA 3000H	;TURN OUTPUT LEDS OFF
04ED	C3 E0 04		JMP ANDGT	
04F0	3E FE	ON:	MVI A,0FEH	;TURN LED ON
04F2	32 00 30		STA 3000H	
04F5	C3 E0 04		JMP ANDGT	
;				
;CONVEYOR BELT CONTROLLER DEMO PROGRAM				
04F8	CD 97 02	CONV:	CALL BLNK	;BLANK THE DISPLAY
04FB	AF		XRA A	;CLEAR A
04FC	32 F0 0B		STA UDSP0	;SET DISPLAY COUNTER
04FF	CD 4B 01	LOOP:	CALL KIND	;DISPLAY MESSAGE & READ KEYS
0502	FE 00		CPI 0	;CHECK FOR "0" KEY
0504	C2 FF 04		JNZ LOOP	
0507	21 F0 0B		LXI H,NUM	;INCREMENT COUNT
050A	34		INR M	
050B	7E		MOV A,M	;TEST FOR COUNT=10
050C	FE 0A		CPI 10	
050E	CA 14 05		JZ MOVE	
0511	C3 FF 04		JMP LOOP	
0514	3E 7F	MOVE:	MVI A,7FH	
0516	16 F0		MVI D,MIN	;SET LOW FREQ VALUE
0518	32 00 30	MLP:	STA 3000H	;WRITE DATA TO LEDS
051B	01 50 00		LXI B,DTIME	;WAIT DELAY TIME
051E	D5		PUSH D	
051F	CD 30 04		CALL DELB	
0522	D1		POP D	

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0523	CD 31 05		CALL TONE	;GENERATE BEEP
0526	37		STC	
0527	1F		RAR	;SHIFT PATTERN
0528	DA 18 05		JC MLP	
052B	32 00 30		STA 3000H	;TURN OFF LEDS
052E	C3 F8 04		JMP CONV	
0531	5F	TONE:	MOV E,A	;SAVE A
0532	D5		PUSH D	
0533	42		MOV B,D	
0534	CD 12 00		CALL BEEP1	;GENERATE BEEP
0537	D1		POP D	
0538	7A		MOV A,D	;INCREASE FREQUENCY
0539	D6 10		SUI INCR	
053B	57		MOV D,A	
053C	7B		MOV A,E	;RESTORE A
053D	C9		RET	
	F0 0B	NUM	EQU 0BF0H	;RAM BUFFER LOCATION
	10 00	INCR	EQU 10H	;FREQ INCREMENT
	F0 00	MIN	EQU 0F0H	;FREQUENCY MINIMUM
	50 00	DTIME	EQU 80	;TIME BETWEEN SHIFTS
;				
;WELL TEMPERED MICROPROCESSOR				
;GENERATES PSEYDO RANDOM TONES FROM ROM				
053E	21 80 01	WTM:	LXI H,0180H	;SET ROM POINTER TO 0180
0541	7E	WTM1:	MOV A,M	;ROM CONTENTS
0542	E6 7E		ANI 7EH	;MASK BITS
0544	47		MOV B,A	;SET BEEP FREQ. REG
0545	E5		PUSH H	;SAVE ADRS POINTER
0546	CD 12 00		CALL BEEP1	;GENERATE BEEP
0549	E1		POP H	;RESTORE ADRS POINTER
054A	01 50 00		LXI B,0050H	;SET DELAY REG
054D	CD 30 04		CALL DELB	;PAUSE

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0550	23		INX H	;NEXT ADRS OF ROM
0551	3E 03		MVI A,03H	;LAST ADRS OF LOOP
0553	BC		CMP H	;LAST ADRS
0554	CA 3E 05		JZ WTM	;IF LAST ADRS
0557	C3 41 05		JMP WTM1	;IF NOT LAST ADRS
;				
;SQUIRREL FEEDBACK SHIFT REGISTER DISPLAY				
055A	CD 97 02	SQRL:	CALL BLNK	;CLEAR THE DISPLAY
055D	AF		XRA A	;CLEAR A AND CARRY
055E	06 01		MVI B,1	;SEED
0560	17	SQRL1:	RAL	;SHIFT A A7 TO CARRY
0561	57		MOV D,A	;SAVE IT
0562	78		MOV A,B	;GET B FSR REGISTER
0563	1F		RAR	;SHIFT A7 INTO B7
0564	4F		MOV C,A	;SAVE IT
0565	A8		XRA B	;XOR B0 AND B1
0566	E6 01		ANI 1	;SET B7 TO B1 TO 0
0568	B2		ORA D	;INSERT B0 XOR B1 IN A0
0569	41		MOV B,C	;RESTORE NEW B
056A	21 FC 0B		LXI H,DDSP2	;ADRS OF DECODED DISPLAY D2
056D	F5		PUSH PSW	;SAVE A FSR REG
056E	0F		RRC	;GET DIGIT 2 BITS IN POSITION
056F	0F		RRC	;
0570	CD 86 05		CALL SQRL3	;FORMAT AND STORE DIGITS 2,3
0573	2C		INR L	;ADRS DIGIT 4
0574	78		MOV A,B	;GET B FSR REGISTER
0575	07		RLC	;GET DIGIT 4 BITS IN POSITION
0576	CD 86 05		CALL SQRL3	;FORMAT AND STORE DIGITS 4,5
0579	0E 0F		MVI C,0FH	;DISPLAY TIMER SEGMENTS
057B	CD C8 01	SQRL2:	CALL SDS	;DISPLAY SEGMENTS
057E	0D		DCR C	;DCR TIMER
057F	C2 7B 05		JNZ SQRL2	;IF TIME NOT UP

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0582	F1		POP PSW	;RESTORE A FSR REGISTER
0583	C3 60 05		JMP SQRL1	;DO IT AGAIN
0586	4F	SQRL3:	MOV C,A	;SAVE SHIFTED VALUE
0587	F6 F0		ORI 0F0H	;BLANK E,F,G,DP SEGS
0589	77		MOV M,A	;STORE IN DIG 2 OR 4
058A	79		MOV A,C	;RECALL SHIFTED VALUE
058B	2C		INR L	;ADRS OF DIG 3,5
058C	07		RLC	;GET A SEG IN D0
058D	E6 01		ANI 01	;CLEAR D7-D1
058F	57		MOV D,A	;SAVE A SEG
0590	79		MOV A,C	;RECALL SHIFTED VALUE
0591	0F		RRC	;MOVE A,F,E,D TO D6-D3
0592	E6 38		ANI 38H	;CLEAR D7-D6,D2-D0
0594	B2		ORA D	;INSERT A SEG IN D0
0595	F6 C6		ORI 0C6H	;BLANK B,C,H,DP SEGS
0597	77		MOV M,A	;STORE IN DIG 3 OR 5
0598	C9		RET	
;				
;ORGAN GENERATES TONES FROM KEYBOARD				
0599	16 40	ORGAN:	MVI D,40H	;INITIALIZE SPKR FLAG
059B	CD 9A 01	ORGAN 1:	CALL KRD	;READ KEYS
059E	CD BB 05		CALL CODE	;DECODE KEYS & LOOK-UP DELAY VALUE
05A1	B7		ORA A	;CHECK FOR NO KEY
05A2	CA 9B 05		JZ ORGAN1	
05A5	CD AE 05		CALL DLY	;TIME DELAY
05A8	CD B5 05		CALL SPKR	;CHANGE SPEAKER STATE
05AB	C3 9B 05		JMP ORGAN1	;REPEAT
;				
;DELAY ROYTINE				
05AE	3D	DLY:	DCR A	;DECREMENT A UNTIL ZERO
05AF	00		NOP	

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
05B0	00		NOP	
05B1	C2 AE 05		JNZ DLY	
05B4	C9		RET	
;				
;SPKR ROUTINE TO CHANGE SPEAKER STATE				
05B5	7A	SPKR:	MOV A,D	;GET SPKR FLAG
05B6	EE 80		XRI 80H	;COMPLEMENT BIT 7
05B8	57		MOV D,A	;SAVE FLAG
05B9	30		SIM	;OUTPUT TO SPKR
05BA	C9		RET	
;				
;CODE ROUTINE DETERMINES WHICH KEY IS				
;PRESSED AND LOOKS UP DELAY VALUE				
05BB	06 07	CODE:	MVI B,7	;INITIALIZE ROW COUNT
05BD	21 EF 0B		LXI H,0BEFH	;INITIALIZE DATA ADDRESS
05C0	7E	READ:	MOV A,M	;GET KEY DATA
05C1	2F		CMA	
05C2	B7		ORA A	;ANY KEY PRESSED?
05C3	CA D8 05		JZ NOKEY	
05C6	FE 04		CPI 4H	;DATA=100?
05C8	C2 CC 05		JNZ SHIFT	;IF YES CHANGE TO 011
05CB	3D		DCR A	
05CC	4F	SHIFT:	MOV C,A	;SAVE KEY DATA
05CD	78		MOV A,B	;SHIFT ROW COUNT
05CE	07		RLC	
05CF	07		RLC	
05D0	B1		ORA C	;COMBINE ROW COUNT & DATA
05D1	21 D9 05		LXI H,TABLE9	;SET LOOK-UP ADDRESS
05D4	85		ADD L	
05D5	6F		MOV L,A	
05D6	7E		MOV A,M	;GET DELAY VALUE
05D7	C9		RET	

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
05D8	05	NOKEY:	DCR B	;NO KEY PRESSED-GO TO NEXT ROW
05D9	2B		DCX H	
05DA	78		MOV A,B	;DONE?
05DB	FE 01		CPI 1	
05DD	C2 C0 05		JNZ READ	;IF NOT, READ NEXT ROW
05E0	AF		XRA A	;NO KEY - SET DELAY TO 0
05E1	C9		RET	
;				
;LOOK-UP TABLE FOR ORGAN DELAY VALUES				
05E2	E6	TABLE:	DB 0E6H	;E3
05E3	00		DB 0	;
05E4	00		DB 0	;
05E5	00		DB 0	;
05E6	DA		DB 0DAH	;F3
05E7	BD		DB 0BDH	;G3
05E8	A3		DB 0A3H	;A3
05E9	00		DB 0	;
05EA	8F		DB 8FH	;B3
05EB	85		DB 85H	;C4
05EC	72		DB 72H	;D4
05ED	00		DB 0	
05EE	64		DB 64H	;E4
05EF	5C		DB 5CH	;F4
05F0	4D		DB 4DH	;G4
05F1	00		DB 0	
05F2	43		DB 43H	;A4
05F3	38		DB 38H	;B4
05F4	33		DB 33H	;C5
05F5	00		DB 0	
05F6	2D		DB 2DH	;D5
05F7	24		DB 24H	;E5
05F8	20		DB 20H	;F5

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
;				
;ROCKET BLAST-OFF DEMO PROGRAM				
05F9	3E AA	ROCT:	MVI A,0AAH	;ALL AMBER LED MASK
05FB	D3 30		OUT LOUT	;TURN ON AMBER LEDS
05FD	11 5C 06		LXI D,ROCM	;ROCKET DISPLAY MESSAGE
0600	DF		RST 3	;STORE MESSAGE IN RAM
0601	06 80	ROCT1:	MVI B,80H	;1 SECOND DELAY LOOP COUNTER
0603	CD E9 01	ROCT2:	CALL DCD	;UPDATE DISPLAY
0606	05		DCR B	;COUNTER
0607	C2 03 06		JNZ ROCT2	;REPEAT IF <1 SECOND
060A	D7		RST 2	;BEEP
060B	21 F0 0B		LXI H,UDSP0	;COUNT DOWN SECONDS DIGIT
060E	35		DCR M	;DECREMENT IT
060F	C2 01 06		JNZ ROCT1	;IF NOT LAST COUNT (0 SECONDS)
0612	06 00		MVI B,0	;START BLAST-OFF FREQUENCY
0614	16 02		MVI D,2	;START BLAST-OFF DURATION
0616	3E E7	ROCT3:	MVI A,0E7H	;LED PATTERN
0618	CD 55 06		CALL ROCT5	;SEQUENCE
061B	3E DB		MVI A,0DBH	;LED PATTERN
061D	CD 55 06		CALL ROCT5	;SEQUENCE
0620	3E BD		MVI A,0BDH	;LED PATTERN
0622	CD 55 06		CALL ROCT5	;SEQUENCE
0625	3E 7E		MVI A,07EH	;LED PATTERN
0627	CD 55 06		CALL ROCT5	;SEQUENCE
062A	C2 16 06		JNZ ROCT3	;REPEAT IF B70, FREQ <MAX
062D	AF		XRA A	;CLEAR A
062E	2F		CMA	;SET A TO FF
062F	D3 30		OUT LOUT	;TURN OFF ALL LEDS
0631	01 00 05		LXI B,0500H	;1 SECOND DELAY VALUE
0634	CD 30 04		CALL DELB	;FOR 1 SECOND PAUSE
0637	3E 03		MVI A,3	;NOTE COUNTER
0639	06 34	ROCT4:	MVI B,34H	;1st 3 NOTE FREQ

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
063B	F5		PUSH PSW	;SAVE NOTE COUNT
063C	CD 12 00		CALL BEEP1	;PLAY NOTE
063F	F1		POP PSW	;RESTORE NOTE COUNT
0640	01 80 00		LXI B,0080H	;PAUSE VALUE
0643	CD 30 04		CALL DELB	;PAUSE BETWEEN NOTES
0646	3D		DCR A	;NOTE COUNTER
0647	C2 39 06		JNZ ROCT4	;IF NOT LAST OF 3 FIRST NOTES
064A	06 85		MVI B,85H	;LAST NOTE FREQ
064C	16 40		MVI D,40H	;LAST NOTE DURATION
064E	CD 47 04		CALL BEEP2	;PLAY LAST NOTE
0651	CF		RST 1	;RETURN TO MONITOR
0652	C3 F9 05		JMP ROCT	;RUN PROGRAM AGAIN
0655	D3 30	ROCT5:	OUT LOUT	;UPDATE LEDS
0657	CD 47 04		CALL BEEP2	;BEEP
065A	05		DCR B	;INCREASE FREQ BY 1
065B	C9		RET	;
065C	09	ROCM:	DB 09H	;9
065D	1C		DB 1CH	;-
065E	18		DB 18H	;SMALL C
065F	15		DB 15H	;SMALL O
0660	1B		DB 1BH	;SMALL R
0661	10		DB 10H	;SPACE
;				
;STOPWATCH				
0662	11 BC 06	STW:	LXI D,STWM	;ADRS OF DISPLAY MESSAGE
0665	DF		RST 3	;ZERO DISPLAY
0666	11 FF 00		LXI D,00FFH	;RUN AND 3-KEY STATUS
0669	AF	STW1:	XRA A	;CLEAR A
066A	32 F6 0B		STA UDSP6	;TURN OFF DECIMAL POINT FLAG
066D	CD E9 01		CALL DCD	;UPDATE DISPLAY
0670	06 F5		MVI B,0F5H	;DELAY CONSTANT
0672	05	STWL:	DCR B	;DELAY COUNTER

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0673	29		DAD H	;DELAY INSTRUNCTION
0674	C2 72 06		JNZ STWL	;IF NOT DONE
0677	3E FB		MVI A,0FBH	;0-KEY SCAN DATA
0679	D3 28		OUT SCAN	;0-KEY ROW
067B	DB 18		IN KEY	;INPUT
067D	FE FE		CPI 0FEH	;CHECK FOR 0-KEY
067F	CA 62 06		JZ STW	;IF PUSHED
0682	3E F7		MVI A,0F7H	;3-KEY SCAN DATA
0684	D3 28		OUT SCAN	;3-KEY ROW
0686	DB 18		IN KEY	;INPUT
0688	FE FB		CPI 0FBH	;CHECK FOR 3-KEY
068A	C2 B7 06		JNZ STW6	;IF NOT PUSHED
068D	BB		CMP E	;WAS IT PUSHED BEFORE?
068E	CA 93 06		JZ STW2	;IF IT WAS
0691	5F		MOV E,A	;PUSHED STATUS CODE
0692	14		INR D	;CHANGE RUN MODE RUN/STOP
0693	7A	STW2:	MOV A,D	;CURRENT RUN MODE
0694	1F		RAR	;FLAG MODE BIT TO CARRY
0695	D2 69 06		JNC STW1	;IF IN STOP MODE
0698	21 F0 0B		LXI H,UDSP0	;.01 SEC MEM LOCATION
069B	34	STW3:	INR M	;INCREMENT DIGIT COUNT
069C	7E		MOV A,M	;LOAD DIGIT IN A
069D	FE 0A		CPI 0AH	;OVERFLOW
069F	C2 A8 06		JNZ STW5	;IF DIGIT 9 OR LESS
06A2	36 00		MVI M,0	;SET DIGIT TO 0
06A4	2C	STW4:	INR L	;NEXT HIGHER DIGIT ADRS
06A5	C3 9B 06		JMP STW3	;REPEAT SEQUENCE ON THIS DIGIT
06A8	21 F3 0B	STW5:	LXI H,UDSP3	;10 SEC MEM LOCATION
06AB	7E		MOV A,M	;LOAD DIGIT IN A
06AC	FE 06		CPI 6	;OVERFLOW
06AE	C2 69 06		JNZ STW1	;IF DIGIT 5 OR LESS
06B1	36 00		MVI M,0	;SET DIGIT TO 0

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
06B3	2C		INR L	;MIN DIGIT ADRS
06B4	C3 A4 06		JMP STW4	;REPEAT SEQUENCE FOR MIN DIGIT
06B7	1E FF	STW6:	MVI E,0FFH	;3-KEY NOT PUSHED CODE
06B9	C3 93 06		JMP STW2	;CONTINUE IN CURRENT RUN MODE
06BC	00	STWM:	DB 0	;ZERO DISPLAY MESSAGE TABLE
06BD	00		DB 0	
06BE	00		DB 0	
06BF	00		DB 0	
06C0	10		DB 10H	
06C1	00		DB 0	
;				
;SNAKE PADDLE GAME				
06C2	21 00 FF	SNAKE:	LXI H,0FF00H	;INITIALIZE SCORE DISPLAY
06C5	06 F7		MVI B,RIGHT	;RIGHT PLAYER CODE
06C7	E5		PUSH H	;SAVE SCORE
06C8	3E 40	SERV:	MVI A,40H	;SERVE SPEED
06CA	32 00 0B		STA SPEED	;CURRENT BALL SPEED
06CD	CD 97 02		CALL BLNK	;CLEAR THE DISPLAY
06D0	E1		POP H	;GET SCORE
06D1	0E 1C		MVI C,1CH	;CODE FOR "-" CHAR
06D3	78		MOV A,B	;PLAYER CODE
06D4	FE F7		CPI RIGHT	;RT PLAYER LOST POINT?
06D6	79		MOV A,C	;DASH CHAR CODE
06D7	C2 E4 06		JNZ LLOSE	;IF LEFT PLAYER LOST POINT
06DA	32 F5 0B		STA UDSP5	;LEFT SERVE DISPLAY MESSAGE
06DD	24		INR H	;LEFT PLAYER GETS POINT
06DE	7C		MOV A,H	;LEFT SCORE
06DF	06 FB		MVI B,LEFT	;LEFT PLAYER GETS SERVE
06E1	C3 EB 06		JMP SERV1	;CONTINUE
06E4	32 F2 0B	LLOSE:	STA UDSP2	;RIGHT SERVE DISPLAY MESSAGE
06E7	2C		INR L	;RIGHT PLAYER GETS POINT
06E8	7D		MOV A,L	;RIGHT SCORE

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
06E9	06 F7		MVI B,RIGHT	;RIGHT PLAYER GETS SERVE
06EB	FE 0A	SERV1:	CPI 0AH	;LAST POINT
06ED	E5		PUSH H	;SAVE SCORE
06EE	22 F0 0B		SHLD UDSP0	;DISPLAY SCORE
06F1	CD E9 01		CALL DCD	;UPDATE DISPLAY
06F4	C2 00 07		JNZ SERV2	;IF NOT LAST GAME POINT
06F7	CD 99 07	ENDGM:	CALL RSTGM	;NEW GAME REQUEST?
06FA	CD C8 01		CALL SDS	;UPDATE DISPLAY
06FD	C2 F7 06		JNZ ENDGM	;WAIT FOR NEW GAME REQUEST
0700	CD 99 07	SERV2:	CALL RSTGM	;NEW GAME REQUEST?
0703	CA C2 06		JZ SNAKE	;IF NEW GAME REQUESTED
0706	CD A2 07		CALL PADL	;CHECK PADDLE
0709	C2 00 07		JNZ SERV2	;IF NOT PUSHED
070C	C5	RBND:	PUSH B	;SAVE HITTING PLAYER CODE
070D	78		MOV A,B	;HITTING PLAYER CODE
070E	FE FB		CPI LEFT	;LEFT PLAYER CODE
0710	01 D4 07		LXI B,LSM	;LEFT PLAYER MESSAGE POINTER
0713	CA 19 07		JZ RSRV	;IF LEFT PLAYER WAS HITTER
0716	01 E2 07		LXI B,RSM	;RIGHT PLAYER MESSAGE CODE
0719	CD 38 07	RSRV:	CALL PASS	;START BALL MOVING
071C	C1		POP B	;RESTORE LAST HITTER CODE
071D	78		MOV A,B	;HITTING PLAYER CODE
071E	FE FB		CPI LEFT	;LEFT PLAYER CODE
0720	06 F7		MVI B,RIGHT	;RIGHT PLAYER CODE
0722	CA 27 07		JZ RBND1	;IF BALL CAME FROM LEFT PLAYER
0725	06 FB		MVI B,LEFT	;LEFT PLAYER NEXT HITTER
0727	CD 5A 07	RBND1:	CALL PLAY	;CHECK PADDLE
072A	7A		MOV A,D	;POINT LOSS REGISTER
072B	FE FF		CPI PNTLS	;POINT LOSS CODE
072D	C2 0C 07		JNZ RBND	;IF NO LOSS POINT, REVERSE BALL
0730	0E 28		MVI C,LOSTN	;LOSE POINT TRILL CODE
0732	CD B9 07		CALL TRIL	;PLAY TRILL

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0735	C3 C8 06		JMP SERV	;NEW SERVE
;				
;PASSES BALL FROM ONE PLAYER TO THE OTHER				
0738	CD 97 02	PASS:	CALL BLNK	;CLEAR THE DISPLAY
073B	26 0B		MVI H,0BH	;MSBYTE DISP MEM
073D	16 0E		MVI D,0EH	;BALL PATTERN LENGTH
073F	0A	PASS1:	LDAX B	;BALL POSITION DISPLAY CODE
0740	5F		MOV E,A	;SAVE A
0741	E6 12		ANI 12H	;MASK
0743	0F		RRC	;POSITION
0744	6F		MOV L,A	;SAVE
0745	0F		RRC	;POSITION
0746	0F		RRC	;POSITION
0747	B5		ORA L	;MASK
0748	E6 03		ANI 3	;MASK
074A	C6 FC		ADI 0FCH	;OFFSET FOR RAM
074C	6F		MOV L,A	;LSBYTE DISP MEM
074D	7B		MOV A,E	;RESTORE MESSAGE
074E	F6 92		ORI 92H	;SET DP,E,B SEGS OFF
0750	77		MOV M,A	;SEND TO UDSPX
0751	CD AD 07		CALL TIMER	;DISPLAY IT
0754	03		INX B	;NEXT MESSAGE
0755	15		DCR D	;MESSAGE COUNTER
0756	C2 3F 07		JNZ PASS1	;IF LAST BALL POSITION
0759	C9		RET	
;				
;PLAY ROUTINE WHEN PASS IS COMPLETE				
075A	16 FF	PLAY:	MVI D,PNTLS	;POINT LOSS CODE
075C	CD A2 07		CALL PADL	;CHECK PADDLE
075F	C8		RZ	;IF PUSHED TOO SOON
0760	3A 00 0B		LDA SPEED	;PRESENT BALL SPEED
0763	5F		MOV E,A	;SAVE IT

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0764	57		MOV D,A	;SET REACTION SPEED COUNTER VALUE
0765	0F		RRC	;HALVE BALL SPEED,TIME
0766	4F		MOV C,A	;SAVE IT
0767	7A	PLAY1:	MOV A,D	;RESTORE COUNTER VALUE
0768	3D		DCR A	;COUNTER
0769	B9		CMP C	;HALF TIME
076A	CA 82 07		JZ FSTR	;IF PADL NOT PUSHED BY THIS TIME
076D	57		MOV D,A	;SAVE COUNT
076E	CD A2 07		CALL PADL	;CHECK PADDLE
0771	C2 67 07		JNZ PLAY1	;IF NOT PUSHED
0774	0E 0B		MVI C,SLOTN	;SLOWER TRILL CODE
0776	CD B9 07		CALL TRIL	;PLAY TRILL
0779	7B		MOV A,E	;ORIGINAL BALL SPEED
077A	FE 40		CPI 40H	;SLOWEST SPEED
077C	C8		RZ	;IF ALREADY AT SLOWEST SPEED
077D	07		RLC	;HALVE BALL SPEED
077E	32 00 0B		STA SPEED	;STORE IT
0781	C9		RET	
;				
;FASTER INCREASES RETURN SPEED				
0782	7A	FSTR:	MOV A,D	;RESTORE COUNTER VALUE
0783	16 FF		MVI D,PNTLS	;LOSE SIGNAL CODE
0785	3D		DCR A	;COUNTER
0786	C8		RZ	;IF PADDLE NOT PUSHED IN TIME
0787	57		MOV D,A	;SAVE A
0788	CD A2 07		CALL PADL	;CHECK PADDLE
078B	C2 82 07		JNZ FSTR	;IF NOT PUSHED
078E	0E 01		MVI C,FSTTN	;FASTER SIGNAL CODE
0790	CD B9 07		CALL TRIL	;PLAY TRILL
0793	7B		MOV A,E	;ORIGINAL BALL SPEED
0794	0F		RRC	;DOUBLE BALL SPEED
0795	32 00 0B		STA SPEED	;STORE IT

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
0798	C9		RET	
;				
;RESET GAME BUTTON PUSHED?				
0799	3E FB	RSTGM:	MVI A,0FBH	;0 KEY INPUT CODE
079B	D3 28		OUT SCAN	;SET SCAN LATCH
079D	DB 18		IN KEY	;INPUT KEYS
079F	FE FE		CPI 0FEH	;0 KEY INPUT CODE
07A1	C9		RET	
;				
;PADDLE PUSHED?				
07A2	CD C8 01	PADL:	CALL SDS	;UPDATE DISPLAY
07A5	78		MOV A,B	;PADDLE KEY SCAN MASK
07A6	D3 28		OUT SCAN	;SET SCAN LATCH
07A8	DB 18		IN KEY	;INPUT KEYS
07AA	FE FB		CPI 0FBH	;ACCEPTABLE KEY INPUT CODE
07AC	C9		RET	
;				
;TIME CONTROLS BALL SPEED				
07AD	3A 00 0B	TIMER:	LDA SPEED	;PRESENT BALL SPEED
07B0	CD C8 01	TIME1:	CALL SDS	;UPDATE DISPLAY/DELAY
07B3	DE 02		SBI 2	;COUNTER
07B5	C2 B0 07		JNZ TIME1	;IF COUNTER NOT DONE
07B8	C9		RET	
;				
;TRILL MAKES SPEAKER SOUNDS				
07B9	D5	TRIL:	PUSH D	;
07BA	C5		PUSH B	;
07BB	06 06		MVI B,6	;START FREQ
07BD	16 01		MVI D,1	;TONE DURATION
07BF	C5	TRIL1:	PUSH B	;SAVE START FREQ
07C0	CD 47 04		CALL BEEP2	;TONE
07C3	C1		POP B	;START FREQ

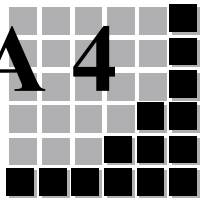
Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
07C4	04		INR B	;DECREASE FREQ
07C5	79		MOV A,C	;TRILL CODE
07C6	FE 01		CPI FSTTN	;FASTER CODE
07C8	C2 CD 07		JNZ TRIL2	;IF NOT
07CB	05		DCR B	;RESTORE FREQ
07CC	05		DCR B	;INCREASE FREQ
07CD	B8	TRIL2:	CMP B	;LAST BEEP
07CE	C2 BF 07		JNZ TRIL1	;IF NOT DONE
07D1	C1		POP B	;
07D2	D1		POP D	;
07D3	C9		RET	
;				
;LEFT SERVE MESSAGE TABLE				
07D4	F7	LSM:	DB 0F7H	;DISPLAY SEGMENT CODES
07D5	F5		DB 0F5H	
07D6	E7		DB 0E7H	
07D7	E5		DB 0E5H	
07D8	E1		DB 0E1H	
07D9	A1		DB 0A1H	
07DA	A7		DB 0A7H	
07DB	B5		DB 0B5H	
07DC	B7		DB 0B7H	
07DD	97		DB 097H	
07DE	96		DB 096H	
07DF	B4		DB 0B4H	
07E0	A6		DB 0A6H	
07E1	A0		DB 0A0H	
;				
;RIGHT SERVE MESSAGE TABLE				
07E2	EC	RSM:	DB 0ECH	;DISPLAY SEGMENT CODES
07E3	EE		DB 0EEH	
07E4	FC		DB 0FCH	

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
07E5	FE		DB 0FEH	
07E6	DE		DB 0DEH	
07E7	9E		DB 09EH	
07E8	BC		DB 0BCH	
07E9	AE		DB 0AEH	
07EA	AC		DB 0ACH	
07EB	A8		DB 0A8H	
07EC	A0		DB 0A0H	
07ED	A6		DB 0A6H	
07EE	B4		DB 0B4H	
07EF	96		DB 096H	
	00 0B	SPEED	EQU 0B00H	
	FF 00	PNTLS	EQU 0FFH	
	FB 00	LEFT	EQU 0FBH	
	F7 00	RIGHT	EQU 0F7H	
	28 00	LOSTN	EQU 028H	
	0B 00	SLOTN	EQU 00BH	
	01 00	FSTTN	EQU 001H	
;				
;LESSON 3 COUNTER PROGRAM				
07F0	3E 00	XSTART:	MVI A,0	;SET A REGISTER TO 0
07F2	3C	XLOOP:	INR A	;INCREMENT A REGISTER
07F3	FE 0A		CPI 10	;COMPARE A REGISTER TO 10
07F5	CA F0 07		JZ XSTART	;GO TO BEGINNING IF A=10
07F8	C3 F2 07		JMP XLOOP	;INCREMENT AGAIN
07FB	00		NOP	
07FC	00		NOP	
07FD	00		NOP	
07FE	00		NOP	
07FF	DE	CHXSN:	DB 0DEH	
;				

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
;EQUATES				
	18 00	KEY	EQU 18H	;KEY INPUT PORT ADRS
	20 00	SIN	EQU 20H	;KEY INPUT PORT ADRS
	28 00	SCA	EQU 28H	;KEY SCAN PORT ADRS
	30 00	LOU	EQU 30H	;KEY OUTPUT PORT ADRS
	38 00	DSP	EQU 38H	;KEY DISPLAY PORT ADRS
	0B 00	DIM	EQU 0BH	;DEFAULT INTERRUPT MASK
	00 08	PC	EQU 080	;DEFAULT PROGRAM COUNTER
	B0 0B	USP	EQU 0BB	;DEFAULT USER STACK POINTER
	CE 0B	MSP	EQU 0BC	;MONITOR STACK POINTER
	80 00	TIM	EQU 80H	;1MS TIME CONSTANT FOR DELAY LOOP
	06 00	FRE	EQU 06H	;BEEP DEFAULT FREQUENCY CONSTANT
	04 00	DUR	EQU 04H	;BEEP DEFAULT DURATION CONSTANT
	F0 0A	RS4	EQU 0AF	;RST4 LINK
	F3 0A	RS5	EQU 0AF	;RST5 LINK
	F6 0A	RS5	EQU 0AF	;RST5.5 LINK
	F9 0A	RS6	EQU 0AF	;RST6 LINK
	FC 0A	RS6	EQU 0AF	;RST6.5 LINK FOR INTRPT KEY
	FF 0A	TPR	EQU 0AF	;TOP OF PROTECTED RAM
	EF 0A	UR	EQU 0AE	;UPPER RAM BELOW RST LINKS
	D1 0B	TSA	EQU 0BD	;TEMPORARY STACK POIN. SAVE ADRS
	D3 0B	TSA	EQU 0BD	;TEMPORARY H-L REG SAVE ADRS
	D5 0B	RAM	EQU 0BD	;RAM LINK TO USER PROG
	D6 0B	RS	EQU 0BD	;RUN STATUS ADRS
	D7 0B	RAM	EQU 0BD	;RAM LINK TO USER PROG
	DB 0B	SAV	EQU 0BD	;SAVE INTERRUPT MASK ADRS
	DC 0B	SAV	EQU 0BD	;SAVE PROGRAM COUNTER ADRS
	DE 0B	SAV	EQU 0BD	;SAVE STACK POINTER LOW ADRS
	DF 0B	SAV	EQU 0BD	;SAVE STACK POINTER HIGH ADRS
	E0 0B	SAV	EQU 0BE	;SAVE L REG ADRS
	E2 0B	SAV	EQU 0BE	;SAVE E REG ADRS
	E7 0B	SAV	EQU 0BE	;SAVE A REG ADRS

Διεύθ.	Περιεχ/νο	Ετικέτα	Εντολή	Σχόλια
	E8 0B	UDK	EQU 0BE	;UNDECODED KEY SCAN 0
	F0 0B	UDS	EQU 0BF	;UNDECODED DISPLAY DIGIT 0 ADRS
	F1 0B	UDS	EQU 0BF	;UNDECODED DISPLAY DIGIT 1 ADRS
	F2 0B	UDS	EQU 0BF	;UNDECODED DISPLAY DIGIT 2 ADRS
	F3 0B	UDS	EQU 0BF	;UNDECODED DISPLAY DIGIT 3 ADRS
	F5 0B	UDS	EQU 0BF	;UNDECODED DISPLAY DIGIT 5 ADRS
	F6 0B	UDS	EQU 0BF	;UNDECODED DISPLAY DIGIT 6 ADRS
	F8 0B	RMP	EQU 0BF	;REGISTER MESSAGE POINTER
	FA 0B	DDS	EQU 0BF	;DECODED DISPLAY DIGIT 0 ADRS
	FB 0B	DDS	EQU 0BF	;DECODED DISPLAY DIGIT 1 ADRS
	FC 0B	DDS	EQU 0BF	;DECODED DISPLAY DIGIT 2 ADRS
	FD 0B	DDS	EQU 0BF	;DECODED DISPLAY DIGIT 3 ADRS
	FE 0B	DDS	EQU 0BF	;DECODED DISPLAY DIGIT 4 ADRS
	FF 0B	DDS	EQU 0BF	;DECODED DISPLAY DIGIT 5 ADRS
0800			END 0H	

ΠΑΡΑΡΤΗΜΑ 4



Οι Εντολές του μΕ 80x86

- Οι εντολές του 8086/88 και οι κυριότερες οδηγίες προς τον assembler
- Οδηγίες προς τον assembler
- Οι εντολές του 8088/86
- Επιπρόσθετες εντολές του 80386
- Επιπρόσθετες εντολές του 80486

Οι εντολές του 8086/88 και οι κυριότερες οδηγίες προς τον assembler

Το τμήμα αυτό του παραρτήματος χωρίζεται σε δύο μέρη. Στο πρώτο μέρος αναφέρονται οι πιο κοινές οδηγίες προς τον assembler (assembler directives). Στο δεύτερο περιλαμβάνονται όλες οι εντολές των μΕ 8086/88, 80386 και 80486 ομαδοποιημένες ανάλογα με τη λειτουργία τους. Για κάθε εντολή, δίνεται το σωστό συντακτικό, πλήρης περιγραφή της λειτουργίας της, πώς επηρεάζονται οι σημαίες και ένα παράδειγμα. Δίπλα σε κάθε παράδειγμα και μέσα σε παρενθέσεις δίνονται με τη σειρά όλες οι λειτουργίες που επιτελούνται κατά την εκτέλεση του συγκεκριμένου παραδείγματος. Για την περιγραφή αυτή πρέπει να αναφέρουμε τα εξής:

Το σύμβολο "←" σημαίνει εκχώρηση.

Κάθε θέση μνήμης συμβολίζεται με "[...]" μέσα στις οποίες περιέχεται η διεύθυνση (offset) ή ο τρόπος υπολογισμού της, ο οποίος εξαρτάται από τον τρόπο αναφοράς στη μνήμη, που χρησιμοποιείται. Π. χ. [2000] ή [BX+SI+5]. Όταν η θέση μνήμης αντιπροσωπεύει ολόκληρη λέξη, τότε ανάμεσα στις διευθύνσεις περιέχονται δύο διευθύνσεις, που χωρίζονται με ":". Η πρώτη είναι η διεύθυνση του πιο σημαντικού byte και η δεύτερη είναι η διεύθυνση του αμέσως προηγούμενου. Υπενθυμίζεται, ότι στον 8088/86, όταν αποθηκεύεται μια λέξη, αποθηκεύεται πρώτα το λιγότερο σημαντικό byte και στην αμέσως επόμενη θέση το πιο σημαντικό. Π.χ. [2001:2000] ή [BX+SI+1:BX+SI]. Τα παραπάνω ισχύουν μόνο για την περιγραφή των παραδειγμάτων και όχι για τη σύνταξη των εντολών. Κατά τη φραστική περιγραφή των εντολών, όπου αναφέρεται θέση μνήμης, εννοείται πως αυτή μπορεί να προσδιορίζεται με οποιοδήποτε τρόπο αναφοράς. Όπου υπάρχουν εξαιρέσεις αυτές θα αναφέρονται κατά την περιγραφή της εντολής.

Τέλος, θα πρέπει να επισημανθεί, ότι κατά την περιγραφή κάθε εντολής, η οποία έχει δύο ορίσματα, αυτά θα πρέπει να είναι ομοειδή, δηλαδή ή και τα δύο μήκους ενός byte ή και τα δύο μήκους μιας λέξης.

Οδηγίες προς τον assembler

ASSUME

Χρησιμοποιείται για να πληροφορήσει τον assembler για το όνομα του λογικού τμήματος (logical segment), που θα αντιστοιχιστεί σε ένα από τα τέσσερα φυσικά τμήματα (segments).

Παράδειγμα:

```
ASSUME      CS:CODE_SEG,
            DS:DATA_SEG,
            ES:DATA_SEG
```

Η οδηγία αυτή δηλώνει, ότι το λογικό τμήμα CODE_SEG θα χρησιμοποιηθεί σαν τμήμα κώδικα, ενώ το DATA_SEG σαν τμήμα δεδομένων και έκτακτο τμήμα.

***DB** (Define Byte)*

Χρησιμοποιείται για να κρατηθούν στη μνήμη ένα ή περισσότερα bytes, τα οποία θα χρησιμοποιηθούν σαν μεταβλητή (ένα byte), ή για τη φύλαξη κάποιων δεδομένων (περισσότερα bytes) όπως strings.

Παραδείγματα:

Με την οδηγία

```
VAR DB 42H
```

ορίζουμε μια μεταβλητή ενός byte και της δίνουμε αρχική τιμή 42H.

Με την οδηγία

```
TABLE 100  DUP(?)
```

ορίζουμε έναν πίνακα μήκους 100 bytes χωρίς να ορίσουμε αρχικές τιμές.

***DW** (Define Word)*

Για την οδηγία αυτή ισχύουν τα ίδια με την DB μόνο που τώρα κρατιούνται θέσεις μνήμης μήκους μιας λέξης. Με την DW μπορούμε να ορίσουμε μεταβλητές μιας λέξης.

***DD** (Define Double word)*

Ισχύουν τα ίδια με τις δύο παραπάνω οδηγίες μόνο που τώρα κρατιούνται θέσεις μνήμης των 4 bytes.

***DQ** (Define Quarter)*

Ισχύουν τα ίδια με τις παραπάνω οδηγίες μόνο που τώρα κρατιούνται θέσεις μνήμης μήκους 4 λέξεων.

***DT** (Define Ten bytes)*

Κρατάει θέσεις μνήμης μήκους 10 bytes. Ισχύουν και εδώ τα ίδια με τις παραπάνω οδηγίες.

***EQU** (EQUate)*

Ορίζει σταθερές μέσα στο πρόγραμμα. Αντιστοιχεί κάποιο όνομα σε συγκεκριμένη τιμή, και στο στάδιο της μετάφρασης ο assembler αντικαθιστά, όπου βρίσκει αυτό το όνομα με την αντίστοιχη τιμή.

Παράδειγμα:

```
FACTOR EQU 78H
```

Η οδηγία αυτή αντιστοιχεί στην αριθμητική τιμή 78H το όνομα FACTOR.

TYPE

Πρόκειται για τελεστή, ο οποίος υπολογίζει το μήκος μιας μεταβλητής σε bytes.

Παράδειγμα:

```
MOV BX,TYPE WORD_ARRAY
```

όπου η WORD_ARRAY είναι μεταβλητή μήκους δύο byte. Με την εντολή αυτή εκχωρείται στον BX το μήκος της μεταβλητής, δηλαδή ο αριθμός δύο.

OFFSET

Ο τελεστής OFFSET υπολογίζει τη σχετική διεύθυνση μέσα στο τμήμα (offset) μιας μεταβλητής.

Παράδειγμα:

MOV BX,OFFSET VAR

Με την εντολή αυτή η διεύθυνση (offset) της μεταβλητής VAR υπολογίζεται και αποθηκεύεται στον BX. Το ίδιο μπορεί να γίνει και με την εντολή LEA του 8088/86.

PTR

Με τον τελεστή αυτό καθορίζεται σε μια εντολή, αν ένα από τα ορίσματά της το οποίο είναι θέση μνήμης, είναι μήκους μιας λέξης ή ενός byte, όταν αυτό δεν είναι προφανές.

Παράδειγμα:

INC PTR WORD [BX]

Εδώ είναι αναγκαίο να καθοριστεί, αν η θέση μνήμης της οποίας το περιεχόμενο θα αυξηθεί κατά 1, είναι μήκους ενός ή δύο bytes. Αντίθετα στην εντολή

ADD AL,[BX]

αυτό είναι προφανές από το όρισμα AL, που είναι του ενός byte..

LENGTH

Αυτός ο τελεστής υπολογίζει το μήκος σε bytes μιας ακολουθίας δεδομένων.

Παράδειγμα: Με την εντολή

MOV CX,LENGTH STRING1

τοποθετείται στον CX το μήκος σε bytes του string STRING1.

SEGMENT

Η οδηγία αυτή δηλώνει την αρχή κάποιου λογικού τμήματος.

Παράδειγμα: Με την οδηγία

SEGNENT CODE_SEG

δηλώνεται η αρχή του λογικού τμήματος CODE_SEG.

ENDS

Η οδηγία αυτή δηλώνει το τέλος ενός λογικού τμήματος η αρχή του οποίου έχει δηλωθεί με τη SEGMENT.

Παράδειγμα:

Η εντολή

CODE_SEG ENDS

δηλώνει το τέλος του λογικού τμήματος CODE_SEG.

PROC

Με την οδηγία αυτή δηλώνεται μια υπορουτίνα (procedure). Το όνομά της προηγείται της PROC ενώ ο τύπος της αν δηλαδή είναι τύπου NEAR ή FAR δηλώνεται μετά την PROG.

Παράδειγμα:

SUBROUTINE PROC FAR

δηλώνεται ότι η SUBROUTINE είναι υπορουτίνα τύπου FAR.

ENDP (END Procedure)

Δηλώνει το τέλος μιας υπορουτίνας της οποίας η αρχή έχει δηλωθεί με την PROC.

Παράδειγμα:

DIVIDE ENDP

δηλώνει το τέλος μιας υπορουτίνας της οποίας η αρχή έχει δηλωθεί με την PROC.

PUBLIC

Με την οδηγία αυτή πληροφορείται ο assembler, ότι ορισμένες υπορουτίνες, επιγραφές, μεταβλητές, λογικά τμήματα όπως και άλλα στοιχεία του προγράμματος μπορούν να χρησιμοποιηθούν από άλλα προγράμματα, ή τμήματα του ίδιου προγράμματος, που βρίσκονται σε ξεχωριστά αρχεία και μεταφράζονται χωριστά. Η ενσωμάτωση όλων αυτών των στοιχείων, που δηλώνονται σαν public, γίνεται στο στάδιο του linking.

Παράδειγμα: Η οδηγία

PUBLIC VAR1,SUBROUTINE1,ADDR1

δηλώνει ότι η μεταβλητή VAR1, η υπορουτίνα SUBROUTINE1 και την επιγραφή ADDR1 είναι διαθέσιμες και σε κώδικα, ο οποίος βρίσκεται σε διαφορετικά αρχεία.

EXTRN

Η οδηγία αυτή χρησιμοποιείται για να δηλώσει ότι κάποιες επιγραφές, μεταβλητές ή υπορουτίνες, ορίζονται σε άλλο αρχείο, όπου έχουν δηλωθεί σαν PUBLIC. Αν μάλιστα πρόκειται για υπορουτίνα ή για επιγραφή, τότε πρέπει να δηλωθεί αν είναι τύπου FAR ή NEAR.

Παράδειγμα: Η δήλωση

EXTRN SUBR:NEAR

σημαίνει ότι η υπορουτίνα είναι τύπου near και βρίσκεται σε άλλο αρχείο.

ORG

Με την οδηγία αυτή δηλώνεται η σχετική διεύθυνση μέσα στο τμήμα, όπου θα αρχίζει το τμήμα κώδικα του προγράμματος που ακολουθεί αμέσως μετά την ORG.

Παράδειγμα:

ORG 100h

Οι εντολές που ακολουθούν τη δήλωση θα τοποθετηθούν από τη διεύθυνση (offset) 100H και κάτω.

Οι εντολές του 8088/86

MOV προορισμός, προέλευση

(MOVe)

Μεταφέρει μια λέξη ή ένα byte από την προέλευση στον προορισμό. Ο προορισμός μπορεί να είναι οποιοσδήποτε καταχωρητής ή θέση μνήμης. Η προέλευση μπορεί να είναι καταχωρητής, θέση μνήμης ή αριθμός δηλαδή άμεσο δεδομένο. Δεν μπορούν όμως να είναι ταυτόχρονα η προέλευση και ο προορισμός θέσεις μνήμης. Καμιά σημαία δεν επηρεάζεται κατά την εκτέλεση της εντολής.

IN συσσωρευτής, πόρτα

(INput)

Μεταφέρει δεδομένα από μια πόρτα I/O προς τον συσσωρευτή, δηλαδή τον καταχωρητή AX αν η πόρτα είναι των 16 bits ή τον AL αν η πόρτα είναι των 8 bits. Η εντολή IN έχει δύο διαφορετικές μορφές ανάλογα με το τι είναι το όρισμα "πόρτα" της εντολής. Αν αυτό είναι αριθμός, τότε αυτή είναι η διεύθυνση της πόρτας απ' όπου θα διαβαστούν τα δεδομένα, οπότε έχουμε την εντολή IN με τη μορφή της σταθερής πόρτας (fixed port format). Αν το όρισμα "πόρτα" είναι ο καταχωρητής DX, τότε έχουμε την εντολή IN με τη μορφή της μεταβλητής πόρτας (variable port format) γιατί η διεύθυνσή της μπορεί να καθοριστεί κατά τη διάρκεια εκτέλεσης του προγράμματος από την τιμή του καταχωρητή DX. Κατά την εκτέλεση της εντολής δεν επηρεάζεται καμιά σημαία.

Παραδείγματα:

```
IN AL,0C8H,
IN AX,34H,
IN AL,DX,
IN AX,DX
```

Στις δύο πρώτες εντολές έχουμε την εντολή IN με τη μορφή σταθερής πόρτας, ενώ στις δύο επόμενες με τη μορφή της μεταβλητής πόρτας.

OUT πόρτα, συσσωρευτής

(OUTput)

Μεταφέρει δεδομένα από το συσσωρευτή προς μια πόρτα εξόδου. Ο συσσωρευτής είναι ο AX, αν πρόκειται για πόρτα των 16 bits, ή ο AL αν πρόκειται για πόρτα των 8 bits. Και εδώ, όπως συμβαίνει και με την εντολή IN, ανάλογα με το αν το όρισμα "πόρτα" είναι αριθμός ή ο καταχωρητής DX, έχουμε την εντολή OUT με τη μορφή της σταθερής και της μεταβλητής πόρτας. Κατά την εκτέλεση της εντολής δεν επηρεάζεται καμιά σημαία.

Παραδείγματα:

```
OUT 0C8H,AL,
OUT 34H,AX,
OUT DX,AL,
OUT DX,AX
```

Στις δύο πρώτες εντολές έχουμε την εντολή OUT με τη μορφή της σταθερής πόρτας, ενώ στις δύο επόμενες με τη μορφή της μεταβλητής πόρτας.

XCHG προρισμός, προέλευση

(eXCHanGe)

Ανταλλάσσει τα περιεχόμενα της προέλευσης και του προορισμού. Τόσο η προέλευση όσο και ο προορισμός μπορεί να είναι θέσεις μνήμης, ή καταχωρητές (εκτός από καταχωρητές τμήματος). Καμιά σημαία δεν επηρεάζεται.

Παράδειγμα:

```
XCHG [SI+200],DX
```

Η ερμηνεία της παραπάνω εντολής είναι:

DS:[SI+201:SI+200]↔DX

LAHF

(Load to AH low byte of Flag register)

Φορτώνει το λιγότερο σημαντικό byte του καταχωρητή σημαιών στον AH. Το byte αυτό είναι το ίδιο με τον 8 bits καταχωρητή σημαιών του 8085. Έτσι η εντολή αυτή, όταν ακολουθείται από την εντολή PUSH AX, έχει το ίδιο αποτέλεσμα με την εντολή PUSH PSW του 8085 και έχει προστεθεί στο set των εντολών του 8088/86 για να επιτρέψει την εύκολη προσομοίωση της εντολής PUSH PSW του 8085. Καμιά σημαία δεν επηρεάζεται.

Παράδειγμα:

Η ερμηνεία της εντολής LAHF είναι:

(AH ← FlagsLow)

SAHF

(Store AH register to low byte of Flags)

Ανάλογη με την εντολή LAHF. Αποθηκεύει το περιεχόμενο του AH στο λιγότερο σημαντικό byte του καταχωρητή σημαιών. Αν προηγείται η εντολή POP AX, τότε ισοδυναμεί με την εντολή POP PSW του 8085. Μόνο οι σημαίες του λιγότερο σημαντικού byte του καταχωρητή σημαιών επηρεάζονται.

LEA προορισμός, προέλευση

(Load Effective Address)

Ο προορισμός είναι ένας 16-bits καταχωρητής και η προέλευση μια θέση μνήμης. Η εντολή φορτώνει στον προορισμό το offset της προέλευσης. Η εντολή αυτή δεν επηρεάζει καμιά σημαία.

Παραδείγματα:

LEA BX,VAR

φορτώνει στον BX το offset της θέσης μνήμης, που αντιπροσωπεύει για τον assembler η μεταβλητή VAR. Επίσης η εντολή

LEA CX,[BX][DI]

υπολογίζει το offset, που προκύπτει από την έκφραση [BX][DI], και το αποθηκεύει στον CX.

LDS προορισμός, προέλευση

LES προορισμός, προέλευση

(Load register DS/ES with word of memory)

Ο προορισμός είναι ένας καταχωρητής των 16 bits και η προέλευση μια θέση μνήμης. Η εντολή αυτή φορτώνει μια λέξη από την προέλευση στον προορισμό και στη συνέχεια φορτώνει και την αμέσως επόμενη λέξη στον DS/ES. Η LDS/LES χρησιμοποιείται συνήθως για να φορτώσει κάποια διεύθυνση στους DS/ES και προορισμό, απ' όπου αρχίζει κάποιο string, το οποίο στη συνέχεια θα υποστεί επεξεργασία με εντολές string. Καμιά σημαία δεν επηρεάζεται κατά την εκτέλεση της εντολής.

Παραδείγματα:

Η εντολή

LDS SI,WORD PTR[BX]

έχει την σημασία

SI ← DS:[BX+1:BX],
DS ← DS:[BX+3:BX+2]

Η εντολή

LES DI,DWORD PTR[BX]

έχει τη σημασία

DI ← DS:[BX+1:BX],
ES ← DS:[BX+3:BX+2]

XLAT

(Translate a byte in AL)

Χρησιμοποιεί το περιεχόμενο του AL για να προσπελάσει κάποιο byte μέσα σε έναν πίνακα μήκους 256 bytes, του οποίου η αρχική διεύθυνση βρίσκεται στον BX. Στη συνέχεια επιστρέφει το byte αυτό στον AL. Χρησιμοποιείται συνήθως για τη μετατροπή ενός κώδικα των 8 bits σε άλλο κώδικα των 8 bits. Καμιά σημαία δεν επηρεάζεται.

Παράδειγμα:

XLAT

με λειτουργία

AL ← DS:[BX+AL]

ΕΝΤΟΛΕΣ STRING

STOSB/STOSW

(STOre String Byte/Word)

Αποθηκεύει ένα byte/λέξη, που περιέχεται στον AL/AX, σε μια θέση στο έκτακτο τμήμα, της οποίας το offset περιέχεται στον DI. Στη συνέχεια αν η σημαία DF είναι μηδέν το περιεχόμενο του DI αυξάνει κατά ένα/δύο, διαφορετικά αν DF=1 μειώνεται κατά ένα/δύο για να δείχνει στο επόμενο ή στο προηγούμενο byte/λέξη του string. Καμιά σημαία δεν επηρεάζεται.

Παράδειγμα: Η εντολή STOSB
έχει ερμηνεία

ES:[DI] ← AL και
αν DF=0 τότε DI ← DI+1
ενώ αν DF=1 τότε DI ← DI - 1

Η εντολή STOSW
έχει την εξής ερμηνεία

ES:[DI+1:DI] ← AX και
αν DF=0 τότε DI ← DI+2
ενώ αν DF=1 τότε DI ← DI-2

***STOS** προορισμός*

(STOre String)

Ο προορισμός είναι μια θέση μνήμης. Ισχύουν ακριβώς τα ίδια με αυτά που ισχύουν για την εντολή STOSB, οπότε η εντολή χειρίζεται ένα byte, ή με όσα ισχύουν για την εντολή STOSW, οπότε χειρίζεται μια λέξη. Η μοναδική διαφορά που υπάρχει είναι η χρησιμοποίηση αντί του DI του προορισμού. Ο assembler ξεχωρίζει αν η STOS είναι εντολή χειρισμού ενός byte ή μιας λέξης από το αν ο προορισμός έχει μήκος ένα byte ή μια λέξη. Καμιά σημαία δεν επηρεάζεται.

Παραδείγματα:

STOS [1000]
STOS [BP+DI]

LODSB/LODSW

(LOaD String Byte/Word)

Φορτώνει στον AL/AX ένα byte ή μια λέξη από μια θέση μνήμης μέσα στο τμήμα δεδομένων, της οποίας το offset περιέχεται στον SI. Στη συνέχεια, αν η σημαία DF είναι 0 το περιεχόμενο του SI αυξάνεται κατά ένα/δύο, διαφορετικά αν DF=1 μειώνεται κατά ένα/δύο για να δείχνει στο επόμενο ή στο προηγούμενο byte/λέξη του string. Καμιά σημαία δεν επηρεάζεται.

Παράδειγμα: Η εντολή LODSB
σημαίνει

AL ← DS:[SI] και
αν DF=0 SI ← SI+1
ενώ αν DF=1 SI ← SI-1

Επίσης για την εντολή LODSW
έχουμε την εξής ερμηνεία:

AX ← DS:[SI+1:SI] και
αν DF=0 SI ← SI+2
ενώ αν DF=1 SI ← SI - 2

***LODS** προέλευση*

(LOaD String)

Η προέλευση είναι μια θέση μνήμης μήκους μιας λέξης. Ισχύουν ακριβώς τα ίδια με αυτά που ισχύουν για την εντολή LODSW ή την LODSB με μόνη διαφορά τη χρησιμοποίηση αντί του SI της προέλευσης. Για το πώς ο assembler ξεχωρίζει για ποια περίπτωση πρόκειται, ισχύουν τα ίδια με την εντολή STOS. Καμιά σημαία δεν επηρεάζεται.

Παράδειγμα:

LODS [2000]
LODS SS:[BP]

MOVSB/MOVSW

(MOve String Byte/Word)

Αντιγράφει ένα byte/λέξη από κάποια θέση μνήμης στο τμήμα δεδομένων, της οποίας το offset περιέχεται στον SI, σε μια άλλη θέση μνήμης στο έκτακτο τμήμα, της οποίας το offset περιέχεται στον DI. Στη συνέχεια αν DF=0 τα περιεχόμενα των SI και DI αυξάνονται κατά ένα/δύο διαφορετικά αν DF=1 μειώνονται κατά ένα/δύο για να δείχνουν στο επόμενο ή στο προηγούμενο byte/λέξη των αντίστοιχων strings. Καμία σημαία δεν επηρεάζεται.

Παράδειγμα: Η εντολή MOVSB
σημαίνει
ES:[DI] ← DS:[SI] και
αν DF=0 τότε SI ← SI+1 και DI ← DI+1
ενώ αν DF=1 τότε SI ← SI-1 και DI ← DI-1
Ομοίως, η εντολή MOVSW
σημαίνει
ES:[DI+1:DI] ← DS:[SI+1:SI] και
αν DF=0 τότε SI ← SI+2 και DI ← DI+2
ενώ αν DF=1 τότε SI ← SI-2 και DI ← DI-2

MOVS προορισμός, προέλευση

(MOVe String)

Ο προορισμός και η προέλευση είναι δύο θέσεις μνήμης. Για την εντολή αυτή ισχύουν τα ίδια με αυτά που ισχύουν και για την εντολή MOVSW ή τη MOVSB με μόνη διαφορά τη χρησιμοποίηση του προορισμού αντί του DI και της προέλευσης αντί του SI. Για το πώς ο assembler ξεχωρίζει για ποια περίπτωση πρόκειται ισχύουν τα ίδια με την εντολή STOS. Καμία σημαία δεν επηρεάζεται.

Παράδειγμα:

MOVS [SI],ES:[DI+BX+8]

SCASB/SCASW

(SCAn String Byte/Word)

Συγκρίνει αφαιρώντας ένα byte/λέξη από κάποιο string, που βρίσκεται στο έκτακτο τμήμα, από ένα byte/λέξη που βρίσκεται στον AL/AX, χωρίς όμως να αλλάζει το περιεχόμενο του string ή του AL/AX. Το offset του προς σύγκριση byte/λέξης περιέχεται στον DI. Μετά τη σύγκριση το περιεχόμενο του DI αυξάνει κατά ένα/δύο αν DF=0 ή μειώνεται κατά ένα/δύο αν DF=1 για να δείχνει στο επόμενο ή στο προηγούμενο byte/λέξη του string. Η εκτέλεση της εντολής επηρεάζει τις σημαίες AF, CF, OF, PF, SF και ZF μέσω των οποίων γίνονται γνωστά τα αποτελέσματα της σύγκρισης. Οι υπόλοιπες σημαίες δεν επηρεάζονται.

Παραδείγματα: Η εντολή SCASB σημαίνει
AL-ES:[DI] και
αν DF=0 τότε DI ← DI+1
ενώ αν DF=1 τότε DI ← DI-1
ενώ η εντολή SCASW σημαίνει
AX-ES:[DI+1:DI] και
αν DF=0 τότε DI ← DI+2
ενώ αν DF=1 τότε DI ← DI-2

SCAS προορισμός

(SCAn String)

Ο προορισμός είναι μια θέση μνήμης μήκους μιας λέξης. Για την εντολή αυτή ισχύουν τα ίδια με αυτά που ισχύουν για την εντολή SCASW και SCASB με μόνη διαφορά τη χρησιμοποίηση του προορισμού αντί του DI. Το πως ο assembler ξεχωρίζει για το ποια περίπτωση πρόκειται ισχύουν τα ίδια με την εντολή STOS. Για τις σημαίες ισχύουν τα ίδια με την SCASW/SCASB.

Παραδείγματα:

SCAS SS:[2000]

CMPSB/CMPSW

(CoMPare String Byte/Word)

Συγκρίνει αφαιρώντας ένα byte/λέξη ενός string, που βρίσκεται στο έκτακτο τμήμα από ένα byte/λέξη άλλου string, που βρίσκεται στο τμήμα δεδομένων. Το offset του byte/ λέξης για το string που βρίσκεται στο τμήμα δεδομένων, περιέχεται στον SI, ενώ για το

byte/λέξη του string, που βρίσκεται στο έκτακτο τμήμα, περιέχεται στον DI. Τα δύο strings μετά τη σύγκριση παραμένουν ανεπηρέαστα. Επίσης, αν DF=0 τα περιεχόμενα των DI και SI αυξάνονται κατά ένα/δύο, διαφορετικά αν DF=1 μειώνονται κατά ένα/δύο. Η εκτέλεση της εντολής επηρεάζει τις σημαίες AF, CF, OF, PF, SF και ZF μέσω των οποίων γίνονται γνωστά τα αποτελέσματα της σύγκρισης. Οι υπόλοιπες σημαίες μένουν ανεπηρέαστες.

Παραδείγματα: Με την εντολή CMPSB προκαλεί τη λειτουργία

DS:[SI]-ES:[DI] και
αν DF=0 τότε DI ← DI+1
ενώ αν DF=1 τότε DI ← DI-1

ενώ αντίστοιχα με την CMPSW προκαλεί τη λειτουργία

DS:[SI+1:SI]-ES:[DI+1:DI] και
αν DF=0 τότε DI ← DI+2
ενώ αν DF=1 τότε DI ← DI-2

CMPS προορισμός, προέλευση

(SCAn String)

Ο προορισμός και η προέλευση είναι θέσεις μνήμης μήκους μιας λέξης. Για την εντολή αυτή ισχύουν τα ίδια με αυτά που ισχύουν για την εντολή CMPSW και CMPSB με μόνη διαφορά τη χρησιμοποίηση του προορισμού αντί του DI και της προέλευσης αντί του SI. Για το πώς ο assembler ξεχωρίζει ποια περίπτωση πρόκειται ισχύουν τα ίδια με την εντολή STOS. Για τις σημαίες ισχύουν τα ίδια με την CMPSW/CMPSB.

Παραδείγματα:

CMPS [DI+2],BP

ΠΡΟΘΕΜΑΤΑ ΕΠΑΝΑΛΗΨΗΣ ΕΝΤΟΛΩΝ STRING

Αν μπροστά από οποιαδήποτε εντολή string τοποθετηθεί ένα πρόθεμα (prefix) επανάληψης η εντολή αυτή επαναλαμβάνεται και ο αριθμός των επαναλήψεων περιέχεται στον CX. Σε κάθε επανάληψη μετά την εκτέλεση της εντολής το περιεχόμενο του CX μειώνεται κατά ένα. Ελέγχεται αν ο CX είναι διάφορος του μηδενός και αν ισχύει η συνθήκη που καθορίζει το πρόθεμα. Η μείωση του CX δεν επηρεάζει τις σημαίες. Σε περίπτωση που ληφθεί σήμα διακοπής, η επανάληψη διακόπτεται για να συνεχιστεί μετά την εκτέλεση της διακοπής. Ακολουθούν όλα τα διαθέσιμα προθέματα.

REP εντολή-string

Η εντολή που ακολουθεί τη REP επαναλαμβάνεται μέχρι να μηδενιστεί ο CX.

Παραδείγματα: Η εντολή REP STOSB σημαίνει

STOSB
CX ← CX-1
repeat until CX=0

Ομοίως η εντολή REP MOVSW σημαίνει

MOVSW
CX ← CXX-1
repeat until CX=0

REPE (REPZ) εντολή-string

Η εντολή που ακολουθεί επαναλαμβάνεται συνεχώς όσο ισχύει ZF=1. Η επανάληψη σταματά όταν μηδενιστεί η σημαία ZF ή ο μετρητής CX.

Παράδειγμα: Η εντολή REPZ SCSW αντιστοιχεί στις λειτουργίες

SCASW
CX ← CX - 1
repeat until (ZF=0 or CX=0)

REPNE (REPNZ) *εντολή-string*

Η εντολή-string επαναλαμβάνεται συνεχώς όσο η σημαία ZF παραμένει μηδενισμένη. Η επανάληψη σταματά όταν γίνει ZF=1 ή όταν μηδενιστεί ο μετρητής CX.

Παράδειγμα: Η REPNZ CMPSW

αντιστοιχεί σε

CMPSW
CX ← CX-1
repeat until (ZF=1 or CX=0)

Παρατήρηση: Όταν το πρόθεμα REP χρησιμοποιείται με τις εντολές SCAS και CMPS τότε είναι ισοδύναμο με τα προθέματα REPZ και REPE.

ΕΝΤΟΛΕΣ ΛΟΓΙΚΩΝ ΠΡΑΞΕΩΝ

NOT *προορισμός*

Ο προορισμός μπορεί να είναι οποιοσδήποτε καταχωρητής ή θέση μνήμης. Η εντολή NOT αντιστρέφει κάθε bit του byte ή της λέξης, που βρίσκεται στον προορισμό, σχηματίζοντας έτσι το συμπλήρωμα ως προς 1, το οποίο αποθηκεύεται πάλι στον προορισμό. Καμιά σημαία δεν επηρεάζεται.

Παραδείγματα: Η εντολή

NOT BYTE PTR[BP]

μεταφράζεται σε

$SS:[BP] \leftarrow SS:\overline{[BP]}$

και η εντολή NOT DL σε

$DL \leftarrow \overline{DL}$

AND *προορισμός, προέλευση*

Ο προορισμός μπορεί να είναι ένας καταχωρητής ή θέση μνήμης. Η προέλευση μπορεί να είναι ένας καταχωρητής ή θέση μνήμης ή άμεσο όρισμα δηλαδή ένας αριθμός. Η πηγή και ο προορισμός δεν μπορεί να είναι θέσεις μνήμης στην ίδια εντολή. Η εντολή αυτή εκτελεί τη λογική πράξη AND μεταξύ του κάθε bit της προέλευσης και του αντίστοιχου bit του προορισμού. Το περιεχόμενο αποθηκεύεται στον προορισμό ενώ το περιεχόμενο της προέλευσης παραμένει αμετάβλητο. Η εκτέλεση της εντολής μηδενίζει τις σημαίες CF και OF, επηρεάζει ανάλογα με το αποτέλεσμα τις σημαίες PF, ZF και SF ενώ η AF είναι ακαθόριστη.

Παράδειγμα:

Η εντολή

AND BH,BYTE PTR[2000]

ερμηνεύεται ως

BH ← BH AND DS:[2000]

OR *προορισμός, προέλευση*

Για τον προορισμό και την προέλευση καθώς και για την κατάσταση των σημαιών μετά την εκτέλεση της εντολής ισχύουν ακριβώς τα ίδια με την εντολή AND. Η εντολή αυτή εκτελεί τη λογική πράξη OR του κάθε bit της προέλευσης με το αντίστοιχο bit του προορισμού. Το αποτέλεσμα αποθηκεύεται στον προορισμό ενώ το περιεχόμενο της προέλευσης παραμένει αμετάβλητο.

Παράδειγμα: Η σημασία της εντολής

OR AX,1000H

είναι

AX ← AX OR 1000H

TEST *προορισμός, προέλευση*

Για την εντολή TEST ισχύουν όσα αναφέρθηκαν στην εντολή AND σχετικά με την προέλευση τον προορισμό και τις σημαίες. Η εντολή αυτή εκτελεί τη λογική πράξη AND του κάθε bit της προέλευσης με το αντίστοιχο bit του προορισμού με μόνη διαφορά το ότι ο προορισμός και η προέλευση παραμένουν αμετάβλητα. Με άλλα λόγια η εντολή TEST εκτελεί μεταξύ των δύο ορισμάτων, το αποτέλεσμα της οποίας γνωστοποιείται μέσω των σημαιών.

Παραδείγματα:

Η σημασία της εντολής TEST CX,DX είναι

$CX \wedge DX$ (ενημέρωση των σημαιών),

ενώ για την εντολή

TEST [SI],BH

έχουμε ερμηνεία

DS:[SI] ∧ BH (ενημέρωση σημαιών)

ΕΝΤΟΛΕΣ ΟΛΙΣΘΗΣΗΣ ΚΑΙ ΠΕΡΙΣΤΡΟΦΗΣ

SAL προορισμός, μετρητής

SHL προορισμός, μετρητής

(SHift Left)

Ο προορισμός μπορεί να είναι καταχωρητής ή θέση μνήμης και κατά την εκτέλεση της εντολής ολισθαίνει προς τ' αριστερά. Αν θέλουμε να ολισθήσει μόνο κατά μία θέση τότε στη θέση του μετρητή πρέπει να βάλουμε 1. Π.χ.

SHL AX,1

Αν θέλουμε να ολισθήσει κατά περισσότερες θέσεις πρέπει να τοποθετήσουμε τον αριθμό των θέσεων στον CL και στην εντολή SHL να θέσουμε σαν μετρητή τον CL. Π.χ.

SHL BH,CL

Κατά την ολίσθηση όταν ένα bit μετακινείται από το λιγότερο σημαντικό ψηφίο στη θέση του μπαίνει το 0, ενώ το bit του ΠΣΨ ολισθαίνει στο CF. Όταν γίνεται ολίσθηση κατά περισσότερες από μία θέσεις, μετά την εκτέλεση της εντολής το CF θα περιέχει το τελευταίο bit, που μετακινήθηκε από το ΠΣΨ ενώ τα προηγούμενα θα έχουν χαθεί. Οι σημαίες επηρεάζονται ως εξής: Η CF θα περιέχει το ΠΣΨ που ολίσθησε τελευταίο. Για ολίσθηση κατά μία θέση η OF γίνεται 1 αν έχει αλλάξει το ΠΣΨ του προορισμού ενώ για ολίσθηση κατά περισσότερες από μία θέσεις παραμένει ακαθόριστη. Οι σημαίες ZF, SF και PF επηρεάζονται από την κατάσταση του προορισμού και τέλος η AF παραμένει ακαθόριστη.

Παραδείγματα:

SHL AX,1
SAL BH,CL

SAR προορισμός, μετρητής

SHR προορισμός, μετρητής

(SHift Right)

Για τον προορισμό και τον μετρητή ισχύουν εδώ τα ίδια με αυτά που ισχύουν για την εντολή SHL. Η εκτέλεση της εντολής προκαλεί ολίσθηση του προορισμού κατά μία ή περισσότερες θέσεις προς τα δεξιά. Όταν ένα bit ολισθήσει από το ΠΣΨ προς τα δεξιά το ΠΣΨ παραμένει αμετάβλητο. Δηλαδή το ΠΣΨ αντιγράφεται σε όλες τις θέσεις κατά τις οποίες γίνεται η ολίσθηση προς τα δεξιά. Το ΛΣΨ ολισθαίνει προς το CF. Μετά από ολίσθηση κατά περισσότερες θέσεις το CF θα περιέχει το τελευταίο ΛΣΨ που ολίσθησε. Για τις υπόλοιπες σημαίες ισχύουν τα ίδια με την εντολή SAL.

Παραδείγματα:

SHR [BX+SI+200],CL
SAR AX,1

RCL προορισμός, μετρητής

Για τον προορισμό και τον μετρητή ισχύουν τα ίδια με την εντολή SHL. Η εντολή RCL προκαλεί περιστροφή του προορισμού προς τα αριστερά μέσω κρατουμένου κατά μία ή περισσότερες θέσεις. Δηλαδή το CF ολισθαίνει στο ΛΣΨ του προορισμού, το περιεχόμενο του οποίου έχει ήδη ολισθήσει προς τα αριστερά, ενώ το ΠΣΨ του προορισμού ολισθαίνει προς το CF. Από τις σημαίες επηρεάζονται μόνο οι CF και OF ενώ οι υπόλοιπες παραμένουν ανεπηρέαστες. Για περιστροφή κατά μία θέση η OF γίνεται ένα αν αλλάξει το ΠΣΨ του προορισμού, ενώ μετά από μία ολίσθηση κατά περισσότερες θέσεις είναι ακαθόριστη.

Παραδείγματα:

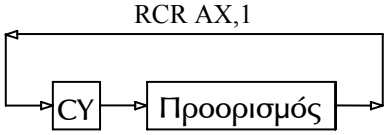
RCL BL,CL
RCR AX,1

***RCR** προορισμός, μετρητής*

Ισχύουν τα ίδια με αυτά που ισχύουν για την εντολή RCL μόνο που τώρα η περιστροφή μέσω κρατουμένου γίνεται προς τα δεξιά.

Παραδείγματα:

RCR BL,CL
RCR AX,1

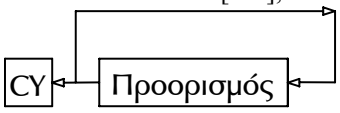


***ROL** προορισμός, μετρητής*

Για τον προορισμό και τον μετρητή ισχύουν τα ίδια με αυτά που ισχύουν για την εντολή SHL. Η εντολή αυτή περιστρέφει τον προορισμό προς τα αριστερά κατά μία ή περισσότερες θέσεις. Η περιστροφή αυτή γίνεται μέσω κρατουμένου. Δηλαδή το ΠΣΨ στο ΛΣΨ του προορισμού το οποίο με τη σειρά του έχει προηγουμένως ολισθήσει. Επιπλέον, το ΠΣΨ ολισθαίνει και στο CF. Μετά από μία ολίσθηση κατά περισσότερες από μία θέσεις το CF περιέχει το τελευταίο ΠΣΨ που ολίσθησε. Για τις σημαίες ισχύουν τα ίδια με την εντολή RCL.

Παραδείγματα:

ROL BYTE PTR[DI],1
ROL WORD PTR[BX],CL

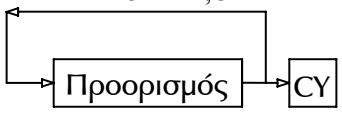


***ROR** προορισμός, μετρητής*

Ισχύουν τα ίδια με την εντολή ROL μόνο που τώρα η περιστροφή γίνεται προς τα δεξιά δηλαδή το ΛΣΨ ολισθαίνει στο ΠΣΨ που έχει ήδη ολισθήσει προς τα δεξιά καθώς και στο CF.

Παραδείγματα:

ROR AL,1
ROR DH,CL



ΕΝΤΟΛΕΣ ΠΡΟΣΘΕΣΗΣ ΚΑΙ ΑΦΑΙΡΕΣΗΣ

***ADD** προορισμός, προέλευση*

Ο προορισμός μπορεί να είναι καταχωρητής ή θέση μνήμης. Η προέλευση μπορεί να είναι καταχωρητής, θέση μνήμης ή άμεσο όρισμα δηλαδή αριθμός. Η προέλευση και ο προορισμός δεν επιτρέπεται να είναι και τα δύο θέσεις μνήμης. Η εντολή ADD προσθέτει τα δύο ορίσματα και τοποθετεί το αποτέλεσμα στον προορισμό. Επηρεάζονται οι σημαίες AF, CF, OF, PF, SF και ZF.

Παραδείγματα: Η εντολή ADD SI,DX λειτουργεί ως εξής:

SI ← SI+DX

Η εντολή ADD BYTE PTR[BX],CH λειτουργεί ως:

DS:[BX] ← DS:[BX]+DH

Τέλος, η εντολή ADD DI,8000H επιτελεί τη λειτουργία:

DI ← DI+8000H

***ADC** προορισμός, προέλευση*

(ADd with Carry)

Ισχύουν τα ίδια με την εντολή ADD εκτός του ότι τώρα μαζί με τον προορισμό και την προέλευση προστίθεται και το περιεχόμενο του CF.

Παραδείγματα: Η εντολή

ADC WORD PTR[2000],800H

λειτουργεί ως

DS:[2001:2000] ←DS:[2001:2000]+800H+CF

Επίσης, η εντολή ADC BL,AH λειτουργεί ως:

BL ← BL+AH+CF

SUB προορισμός, προέλευση

(SUBtract)

Για την προέλευση και τον προορισμό ισχύουν τα ίδια με την εντολή ADD. Η εντολή SUB αφαιρεί το περιεχόμενο της προέλευσης από το περιεχόμενο του προορισμού και αποθηκεύει το αποτέλεσμα στον προορισμό. Αν κατά την αφαίρεση του ΠΣΨ απαιτείται δανεικό κρατούμενο, δηλαδή ο αφαιρέτης είναι μεγαλύτερος από τον αφαιρετέο τότε το CF γίνεται 1 διαφορετικά μηδενίζεται. Για τις σημαίες ισχύουν τα ίδια με την εντολή ADD.

Παραδείγματα: Η εντολή

SUB BYTE PTR[BP+DI],CH

σημαίνει

SS:[BP+DI] ← SS:[BP+DI]+CH

Όμοια η εντολή SUB AX,3FH

σημαίνει

AX ← AX - 3FH

SBB προορισμός, προέλευση

(SuBctract with Borrow).

Ισχύουν τα ίδια με την εντολή SUB μόνο που τώρα εκτός από την προέλευση αφαιρείται από τον προορισμό και το κρατούμενο δηλαδή το περιεχόμενο του CF.

Παραδείγματα: Παρακάτω δίδονται δύο εντολές και η λειτουργική σημασία τους

SBB BYTE PTR[BP],CH
SS:[BP] ← SS:[BP]+CH

SBB DI,6000H
DI ← DI+6000H

INC προορισμός

(INCrement)

Ο προορισμός μπορεί να είναι καταχωρητής ή θέση μνήμης. Η εντολή INC προσθέτει 1 στο περιεχόμενο του προορισμού. Οι σημαίες AF, OF, PF, SF, ZF επηρεάζονται ενώ η CF δεν επηρεάζεται. Δηλαδή αν ο προορισμός γίνει FF (ή FFFF) το CF δεν γίνεται ένα.

Παραδείγματα: Παρακάτω δίδονται δύο εντολές και η λειτουργική σημασία τους

INC WORD PTR CS:[DI]
CS:[DI+1:DI] ← CS:[DI+1:DI]+1

INC BL
BL ← BL+1

DEC προορισμός

(DECrement)

Ισχύουν τα ίδια με την εντολή INC με τη διαφορά ότι τώρα το περιεχόμενο του προορισμού μειώνεται κατά ένα. Για τις σημαίες ισχύουν τα ίδια με την INC. Και εδώ το CF παραμένει ανεπηρέαστο. Δηλαδή αν ο προορισμός γίνει 0 και εκτελεστεί η DEC τότε αυτός γίνεται FFH (ή FFFFH) χωρίς να αλλάξει το κρατούμενο.

NEG προορισμός

(NEGative)

Ο προορισμός μπορεί να είναι καταχωρητής ή θέση μνήμης. Η εντολή NEG αντικαθιστά το περιεχόμενο του προορισμού με το συμπλήρωμά του ως προς 2 και χρησιμοποιείται κυρίως για να αλλάζει το πρόσημο προσημασμένων αριθμών. Αν το περιεχόμενο του προορισμού είναι το -128 ή -32768 η εντολή NEG θα αφήσει αμετάβλητο τον προορισμό και αυτό γιατί οι μεγαλύτεροι προσημασμένοι θετικοί αριθμοί είναι ο +127 και ο +32767 αντίστοιχα. Στην περίπτωση αυτή η σημαία OF γίνεται 1 (η πράξη δεν μπορεί να εκτελεστεί) ενώ σε κάθε άλλη περίπτωση μηδενίζεται. Επίσης το CF γίνεται 1 εκτός αν ο προορισμός είναι μηδέν. Ακόμη επηρεάζονται οι σημαίες AF, SF, PF και ZF.

Παραδείγματα: Παρακάτω δίδονται δύο εντολές και η λειτουργική σημασία τους

NEG WORD PTR[BX]
DS:[BX+1:BX] ← 0-DS:[BX+1:BX]
NEG BL
BL ← 0-BL

CMP προορισμός, προέλευση

(CoMPare)

Η προέλευση και ο προορισμός μπορεί να είναι θέσεις μνήμης, καταχωρητές ή αριθμοί. Δεν μπορεί όμως και τα δύο στην ίδια εντολή να είναι θέσεις μνήμης. Η εντολή αυτή συγκρίνει αφαιρώντας το περιεχόμενο της προέλευσης από το περιεχόμενο του προορισμού χωρίς όμως να τα αλλάζει. Το αποτέλεσμα της σύγκρισης φαίνεται στις σημαίες AF, OF, SF, ZF και CF οι οποίες επηρεάζονται από την εκτέλεση της εντολής.

Παραδείγματα: Παρακάτω δίδονται δύο εντολές και η λειτουργική σημασία τους

CMP [BX+8],CX
DS:[BX+8]-CX (ενημέρωση σημαιών)
CMP DI,8000H
DI-8000H; (ενημέρωση σημαιών)

ΕΝΤΟΛΕΣ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ ΚΑΙ ΔΙΑΙΡΕΣΗΣ

MUL προέλευση

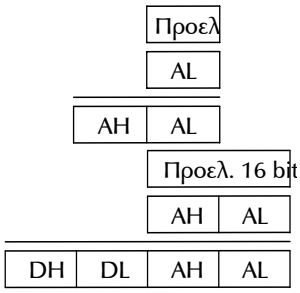
(MULtiplicate)

Η προέλευση μπορεί να είναι καταχωρητής ή θέση μνήμης αλλά όχι αριθμός. Η εντολή MUL θεωρεί το περιεχόμενο του AX αν η προέλευση είναι των 16 bits ή του AL αν είναι των 8 bits και της προέλευσης σαν απρόσημη ποσότητα και τα πολλαπλασιάζει μεταξύ τους. Στην πρώτη περίπτωση το αποτέλεσμα είναι των 32 bits και αποθηκεύεται η πιο σημαντική λέξη στον DX και η άλλη στον AX. Στην δεύτερη περίπτωση το αποτέλεσμα είναι των 16 bits και τοποθετείται στον AX. Αν η πιο σημαντική λέξη ή το πιο σημαντικό byte του αποτελέσματος είναι μηδέν τότε οι σημαίες CF και OF είναι μηδέν. Το γεγονός αυτό μας επιτρέπει να ανιχνεύσουμε και να απαλλαγούμε από τα άχρηστα μηδενικά στην αρχή του αποτελέσματος. Μετά την εκτέλεση της εντολής τα AF, PF, SF και ZF είναι ακαθόριστα.

Παραδείγματα: Παρακάτω δίδονται δύο εντολές και η λειτουργική σημασία τους

MUL BYTE PTR[BX]
AX ← AL*DS:[BX]

MUL CX
DX:AX ← AX*CX



IMUL προέλευση

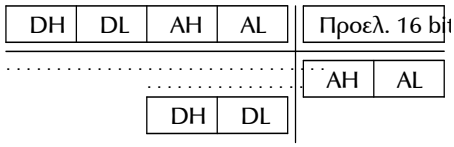
Ισχύουν τα ίδια με την εντολή MUL με τη διαφορά ότι τώρα το περιεχόμενο της προέλευσης και του AL ή του AX θεωρούνται προσημασμένοι αριθμοί και το αποτέλεσμα λαμβάνεται με επέκταση προσήμου. Οι σημαίες OF και CF γίνονται πάλι ένα αν το σημαντικότερο byte ή η σημαντικότερη λέξη του αποτελέσματος περιέχει μέρος του γινομένου.

Διαφορετικά αν περιέχει απλώς την επέκταση προσήμου οι σημαίες αυτές είναι μηδέν. Μετά την εκτέλεση της εντολής τα AF, PF, SF και ZF είναι ακαθόριστα.

Παραδείγματα: Παρακάτω δίδονται δύο εντολές και η λειτουργική σημασία τους

IMUL WORD PTR[SI+100]
DX:AX ← AX*DS:[SI+1001:SI+1000]

IMUL BH
AX ← AL*BH



DIV προέλευση

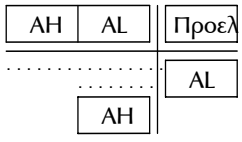
(DIVide)

Η προέλευση μπορεί να είναι θέση μνήμης, καταχωρητής αλλά όχι αριθμός. Η εντολή DIV θεωρεί το περιεχόμενο του AX ή του AX και του DX μαζί καθώς και της προέλευσης σαν απροσήμαστες ποσότητες και κάνει τη διαίρεση. Η προέλευση είναι ο διαιρέτης και αν είναι των 8 bits ο διαιρετέος είναι ο AX και μετά τη διαίρεση το πηλίκο τοποθετείται στον AL και το υπόλοιπο στον AH. Αν ο διαιρέτης είναι των 16 bits ο διαιρετέος είναι ένας αριθμός των 32 bits η πιο σημαντική λέξη του οποίου περιέχεται στον DX και η λιγότερο σημαντική στον AX. Μετά τη διαίρεση το πηλίκο τοποθετείται στον AX και το υπόλοιπο στον DX. Και στις δύο περιπτώσεις αν γίνει διαίρεση με το 0 ή αν το πηλίκο δεν χωράει στον AL ή στον AX αντίστοιχα τότε προκαλείται διακοπή τύπου 0. Όταν το πηλίκο δεν είναι ακέραιος τότε γίνεται αποκοπή του δεκαδικού μέρους και όχι στρογγύλευση. Μετά την εκτέλεση της DIV οι σημαίες είναι ακαθόριστες.

Παραδείγματα: Παρακάτω δίδονται δύο εντολές και η λειτουργική σημασία τους

DIV WORD PTR[1000]
DX:AX ← AX/DS:[1001:1000]

DIV BX
AX ← DX:AX/BX



IDIV προέλευση

Ισχύουν τα ίδια όπως και για την εντολή DIV μόνο που τώρα διαιρέτης και διαιρετέος θεωρούνται προσημασμένοι αριθμοί και το αποτέλεσμα λαμβάνεται με επέκταση προσήμου.

ΕΝΤΟΛΕΣ ΑΡΙΘΜΗΤΙΚΗΣ ΔΙΟΡΘΩΣΗΣ

DAA

(Decimal Adjust Accumulator)

Η εντολή αυτή μετατρέπει το αποτέλεσμα της πρόσθεσης δύο συνεπτυγμένων (packed) ή μη συνεπτυγμένων (unpacked) BCD αριθμών, το οποίο βρίσκεται στον AL, στο σωστό BCD αποτέλεσμα σε αντίστοιχη μορφή. Αυτό γίνεται με τον ακόλουθο τρόπο. Αν το λιγότερο σημαντικό τμήμα του AL (bit 0-3) είναι μεγαλύτερο από το 9 ή αν AF=1 τότε η DAA θα προσθέσει 6 στον AL. Αν τώρα στο αποτέλεσμα που προκύπτει το ανώτερο τμήμα του AL (bit 7-4) είναι μεγαλύτερο του 9 ή το CF κατά την προηγούμενη πρόταση έχει γίνει 1 τότε στον AL προστίθεται το 60H. Οι σημαίες AF, CF, PF και ZF επηρεάζονται από την εντολή, ενώ η OF γίνεται ακαθόριστη.

Παραδείγματα: Η εντολή

DAA

ερμηνεύεται ως:

```
IF ((AL AND 0FH) >9) OR (AF=1) THEN
        AL ← AL+6
        AF=1
IF (AL>9FH) OR (CF=1) THEN
        AL ← AL+60 H
        CF=1
```

DAS

(Decimal Adjust after Subtraction)

Η εντολή DS μετατρέπει το αποτέλεσμα της αφαίρεσης δύο συνεπτυγμένων (packed) ή μη συνεπτυγμένων (unpacked) αριθμών, το οποίο βρίσκεται στον AL, στο σωστό BCD αποτέλεσμα σε αντίστοιχη μορφή. Για να το επιτύχει αυτό εργάζεται ακριβώς όπως και η DAA με τη διαφορά ότι τώρα αφαιρεί αντί να προσθέτει το 6Η και το 60Η από τον AL. Για τις σημαίες ισχύουν ακριβώς τα ίδια με την DAA.

Παραδείγματα: Η λειτουργία της

DAS

είναι η παρακάτω

```
        IF (AL AND 0FH >9) OR (AF=1) THEN
                AL ← AL-6
                AF=1
        IF (AL>9FH) OR (CF=1) THEN
                AL ← AL-60 H
                CF=1
```

AAA

(ASCII Adjust after Addition)

Η εντολή AAA μετατρέπει το άθροισμα των ASCII κωδικών δύο δεκαδικών ψηφίων, που βρίσκεται στον AL σε έναν BCD αριθμό, που τοποθετείται στα bits 0 έως 3 του AL. Τα 4 πρώτα bits (7-4) μηδενίζονται ακόμα και στην περίπτωση που το αποτέλεσμα της πρόσθεσης των ASCII είναι διψήφιος (π.χ. 14). Στην περίπτωση αυτή ενημερώνεται το CF, που είναι αρκετό, αφού το πρώτο ψηφίο αν υπάρχει θα είναι 1. Τα παραπάνω γίνονται με τον ακόλουθο τρόπο. Αν ο αριθμός που περιέχεται στα 4 λιγότερο σημαντικά bits του AL είναι μεγαλύτερος του 9 ή AF=1 τότε στον AL προστίθεται το 6 και το CF γίνεται 1. Στη συνέχεια τα 4 σημαντικότερα bits του AL μηδενίζονται. Η εντολή ενημερώνει μόνο τα CF και AF ενώ οι υπόλοιπες γίνονται ακαθόριστες.

Παράδειγμα: Ο μηχανισμός της εντολής

AAA

είναι ο εξής:

IF (AL AND 0FH >9) OR (AF=1) THEN
AL ← AL+6
CF ← AF
AL ← AL AND 0FH

AAS

ASCII Adjust after Subtraction

Η εντολή AAS μετατρέπει το αποτέλεσμα της αφαίρεσης δύο ASCII δεκαδικών ψηφίων το οποίο βρίσκεται στον AL σε έναν αριθμό BCD, ο οποίος τοποθετείται στα 4 λιγότερο σημαντικά bits του AL ενώ τα 4 πιο σημαντικά bits μηδενίζονται. Αυτό γίνεται όπως και στην AAA μόνο που τώρα αντί να προστίθεται αφαιρείται το 6 από τον AL. Για τις σημαίες ισχύουν τα ίδια με την AAA.

Παράδειγμα: Ο μηχανισμός της εντολής

AAS

είναι ο παρακάτω

IF (AL AND 0FH >9) OR (AF=1) THEN
AL ← AL-6
CF ← AF
AL ← AL AND OFH

AAM

(Adjust After Multiplication)

Η εντολή AAM μετατρέπει το αποτέλεσμα του πολλαπλασιασμού δύο μη συνεπτυγμένων BCD αριθμών, το οποίο βρίσκεται στον AL, σε έναν διψήφιο μη συνεπτυγμένο BCD αριθμό το σημαντικότερο ψηφίο του οποίου βρίσκεται στον AH και το δεύτερο στον AL. Για να γίνει αυτό ο AL διαιρείται με το 0Ah και το πηλίκο αποθηκεύεται στον AH ενώ το υπόλοιπο στον AL. Όλες οι σημαίες μετά την εκτέλεση της AAM εκτός από τις PF, ZF και SF είναι ακαθόριστες.

Παράδειγμα: Η εντολή

AAM

λειτουργεί ως εξής

AH ← AL/0AH
AL ← υπόλοιπο

AAD

(Adjust before Division)

Πριν διαιρεθεί ο AX με έναν μη συνεπτυγμένο BCD αριθμό η εντολή αυτή μετατρέπει τον διψήφιο BCD μη συνεπτυγμένο BCD αριθμό που περιέχεται στον AX στον ισοδύναμο δυαδικό αριθμό, που τοποθετείται στον AL, ενώ ο AH μηδενίζεται. Έτσι μετά τη διαίρεση με έναν μονοψήφιο μη συνεπτυγμένο BCD αριθμό το πηλίκο θα περιέχεται σε μη συνεπτυγμένη BCD μορφή στον AL και ομοίως και το υπόλοιπο στον AH. Αυτό γίνεται πολλαπλασιάζοντας τον AH με 0Ah και προσθέτοντας το αποτέλεσμα στον AL. Όλες οι σημαίες μετά την εκτέλεση της AAD εκτός από τις PF, ZF και SF είναι ακαθόριστες.

Παράδειγμα: Ο μηχανισμός της

AAD

είναι ο εξής

AL ← (AH*0Ah +AL)
AH ← 0

CBW

(Convert signed Byte to signed Byte)

Η εντολή αυτή κάνει επέκταση προσήμου του περιεχομένου του AL στον AH. Δηλαδή αντιγράφει το πρόσημο του AL σε όλα τα bits του AH. Δηλαδή η CBW μετατρέπει το προσημασμένο byte, που βρίσκεται στον AL, σε προσημασμένη λέξη, που τοποθετείται στον AX. Καμιά σημαία δεν επηρεάζεται.

Παράδειγμα: Η εντολή CBW λειτουργεί ως εξής:

IF AL≤7Fh THEN AH ← 0
IF AL>7Fh THEN AH ← FFh

CWD

(Convert Signed Word to Signed Double Word)

Η εντολή αυτή κάνει επέκταση προσήμου του περιεχομένου του AX στον DX. Δηλαδή αντιγράφει το πρόσημο του AX σε όλα τα bits του DX. Δηλαδή η CBW μετατρέπει την προσημασμένη λέξη, που βρίσκεται στον AL, σε προσημασμένη διπλή λέξη, που τοποθετείται στους DX (σημαντικότερη λέξη) και AX. Καμιά σημαία δεν επηρεάζεται.

Παράδειγμα: Για την λειτουργία της CBW έχουμε

IF AX≤7FFFh THEN DX←0
IF AX>7FFFh THEN DX←FFh

ΕΝΤΟΛΗ ΑΛΜΑΤΟΣ ΧΩΡΙΣ ΣΥΝΘΗΚΗ

JMP ετικέτα

Απλή εντολή άλματος χωρίς συνθήκη. Η ετικέτα είναι κάποιο μνημονικό όνομα που βρίσκεται μπροστά από κάποια εντολή του προγράμματος. Έχουμε δύο είδη εντολών JMP. Όταν με την JMP ο έλεγχος μεταφέρεται σε κάποια εντολή, που βρίσκεται στο τρέχον τμήμα κώδικα τότε πρόκειται για εντολή JMP κοντινού στόχου (JMP near target). Το άλμα επιτυγχάνεται αλλάζοντας το περιεχόμενο του IP. Στην περίπτωση αυτή η επιτρεπόμενη απόσταση μετρούμενη σε bytes της επιγραφής από την αντίστοιχη εντολή JMP βρίσκεται στα όρια -32768 έως +32767 δηλαδή 32768 bytes πριν από την εντολή και 32767 bytes μετά την εντολή. Όταν η εντολή αυτή μεταφραστεί από τον assembler σε κώδικα έχει μήκος κανονικά τρία bytes. Στην περίπτωση όμως που η παραπάνω απόσταση βρίσκεται στα όρια -128 έως +127 τότε κατά τη μετάφραση έχει μήκος δύο bytes.

Όταν με την JMP ο έλεγχος μεταφέρεται σε άλλο τμήμα κώδικα, τότε έχουμε εντολή JMP μακρινού στόχου (JMP far target). Σ' αυτήν την περίπτωση εκτός από το περιεχόμενο του IP πρέπει να αλλαχτεί και το περιεχόμενο του CS. Η εντολή JMP δεν επηρεάζει καμιά σημαία.

Παραδείγματα: Παρακάτω δίνονται ζεύγη εντολών και αντίστοιχων εκφράσεων σχετικών με τη λειτουργία τους όπου δεν είναι τετριμμένες:

JMP [2000]
IP ← DS:[2001:2000]) (near target)
JMP FAR CS:[2000]
IP ← CS:[2001:2000]
CS ← DS:[2003:2002] (far target)
JMP AX
IP ← AX (near target)

ΕΝΤΟΛΕΣ ΑΛΜΑΤΟΣ ΥΠΟ ΣΥΝΘΗΚΗ

Οι εντολές άλματος υπό συνθήκη χρησιμοποιούνται συνήθως μετά τις εντολές σύγκρισης, και σε συνδυασμό με αυτές επιτυγχάνεται η εξομοίωση των εντολών διακλάδωσης υπό συνθήκη του τύπου IF...THEN, που διαθέτουν γλώσσες προγραμματισμού υψηλότερου επιπέδου. Με τις εντολές αυτές μπορεί να γίνει άλμα σε απόσταση που μετρούμενη σε bytes βρίσκεται μέσα στα όρια -128 έως +127. Επίσης, οι εντολές αυτές δεν επηρεάζουν τις σημαίες. Παρακάτω αναφέρονται όλες οι εντολές άλματος υπό συνθήκη. Για τις περισσότερες από τις εντολές υπάρχουν δύο μνημονικά τα οποία είναι εντελώς ισοδύναμα. Δίπλα σε κάθε εντολή αναγράφεται η συνθήκη, που πρέπει να ικανοποιείται, για να πραγματοποιηθεί το άλμα. Στην αντίθετη περίπτωση, που η συνθήκη δεν ικανοποιείται η εκτέλεση του προγράμματος συνεχίζεται με την αμέσως επόμενη εντολή.

JCXZ ετικέτα	[CX=0]
JG ετικέτα	[(SF XOR OF) OR ZF)=0]
JNLE ετικέτα	(Greater/Not Less Equal)
JGE ετικέτα	[(SF XOR OF)=1]
JNL ετικέτα	(Greater OR Equal/Not Less)

JL ετικέτα	[(SF XOR OF)=1]
JNGE ετικέτα	(Less/Not Greater Not Equal)
JLE ετικέτα	[((SF XOR OF) OR ZF)=1]
JNG ετικέτα	(Less or Equal/Not Greater)
JO ετικέτα	[OF=1] (Overflow)
JS ετικέτα	[SF=1] (Sign)
JNO ετικέτα	[OF=0] (Not Overflow)
JNS ετικέτα	[SF=0] (Not Sign)
JA ετικέτα	[(CF OR ZF)=0]
JNBE ετικέτα	(Above/Not Below nor Equal)
JAE ετικέτα	[CF=0]
JNB ετικέτα	(Above or Equal/Not Below)
JB ετικέτα	[CF=1]
JNAE ετικέτα	(Below/Not Above nor Equal)
JBE ετικέτα	[(CF OR ZF)=1]
JNA ετικέτα	(Below or Equal/Not Above)
JC ετικέτα	[CF=1] (Carry)
JE ετικέτα	[ZF=1]
JZ ετικέτα	(Equal/Zero)
JP ετικέτα	[PF=1]
JPE ετικέτα	(Parity Even)
JNC ετικέτα	[CF=0] (No Carry)
JNE ετικέτα	[ZF=0]
JNZ ετικέτα	(Not Equal/Not Zero)
JNP ετικέτα	[PF=0]
JPO ετικέτα	(Parity Odd)

Παράδειγμα: Η παρακάτω εντολή

JZ [2000]

ως αποτέλεσμα θα έχει

IF Z=1 THEN

IP ← [2001:2000]

ΕΝΤΟΛΕΣ ΒΡΟΧΟΥ (LOOP)

Με τις εντολές αυτές επιτυγχάνεται κάτω από ορισμένες προϋποθέσεις η συνεχής επανάληψη μιας ακολουθίας εντολών, που περικλείεται από την εντολή βρόγχου και την εντολή του προγράμματος, μπροστά από την οποία υπάρχει η επιγραφή που αναφέρεται στη συγκεκριμένη εντολή βρόγχου. Οι εντολές αυτές είναι ισοδύναμες με τη δομή repeat... until, που διαθέτουν γλώσσες προγραμματισμού υψηλότερου επιπέδου. Ο αριθμός των επαναλήψεων που θα εκτελεστούν, τοποθετείται από πριν στον CX. Μετά από κάθε εκτέλεση βρόγχου το CX μειώνεται κατά 1 και στη συνέχεια εξετάζεται αν έγινε 0 και ακόμη αν ικανοποιείται η συνθήκη, που καθορίζεται στην εντολή βρόγχου. Αν δεν συμβαίνει ούτε το ένα ούτε το άλλο τότε η επανάληψη συνεχίζεται διαφορετικά εκτελείται η αμέσως επόμενη εντολή. Η εκτέλεση των εντολών αυτών δεν επηρεάζει τις σημαίες. Στη συνέχεια αναφέρονται όλες οι εντολές βρόγχου. Δίπλα σε κάθε εντολή αναφέρεται το δεύτερο ισοδύναμο μνημονικό αν υπάρχει και η συνθήκη που πρέπει να ικανοποιείται προκειμένου να συνεχιστεί η επανάληψη της εκτέλεσης των εντολών.

LOOP ετικέτα	[CX<>0]
LOOPE ετικέτα	[CX<>0 AND ZF=1]
LOOPZ ετικέτα	
LOOPNE ετικέτα	[CX<>0 AND ZF=0]
LOOPNZ ετικέτα	

Παραδείγματα:

Η εντολή LOOP LABEL1 ισοδυναμεί με:

CX ← CX-1

IF CX<>0 THEN IP ← [LABEL1]

Η εντολή LOOPNZ [2000] ισοδυναμεί με:

CX ← CX-1

IF CX<>0 AND ZF=0 THEN

IP ← [2001:2000])

ΕΝΤΟΛΕΣ ΣΤΟΙΒΑΣ

PUSH προέλευση

Η προέλευση μπορεί να είναι καταχωρητής των 16 bits (εκτός από καταχωρητή τμήματος) η θέση μνήμης μήκους

μιας λέξης. Η εντολή PUSH μειώνει τον SP κατά δύο και αποθηκεύει τη λέξη, που λαμβάνεται από την προέλευση, στη στοίβα. Πριν τη χρήση αυτής της εντολής πρέπει να λάβουν κατάλληλη τιμή ο SP και ο SS. Οι σημαίες δεν επηρεάζονται.

Παράδειγμα:

Εκτέλεση της PUSH [DI+2]

επιφέρει εκτέλεση των λειτουργιών

SP ← SP-2

SS:[SP+1:SP] ← DS:[DI+3:DI+2]

POP προορισμός

Ο προορισμός μπορεί να είναι καταχωρητής των 16 bits (εκτός από καταχωρητή τμήματος) ή θέση μνήμης μήκους μιας λέξης. Η POP αντιγράφει μια λέξη από τη στοίβα στον προορισμό και κατόπιν αυξάνει τον SP κατά δύο. Τα περιεχόμενα της στοίβας δεν αλλάζουν. Οι σημαίες δεν επηρεάζονται.

Παράδειγμα: Η POP DX ισοδυναμεί με

DX ← SS: [SP+1:SP]

SP ← SP+2

PUSHF

Ισχύουν τα ίδια με την PUSH με τη διαφορά ότι τώρα αποθηκεύονται στη στοίβα τα περιεχόμενα του καταχωρητή σημαιών.

Παράδειγμα: Η PUSHF εκτελείται σαν

SP ← SP-2

SS:[SP+1:SP] ← flags

POPF

Ισχύουν τα ίδια με την εντολή POP με τη διαφορά, ότι τώρα η λέξη που ανακτάται από τη στοίβα αποθηκεύεται στον καταχωρητή σημαιών.

Παράδειγμα: Η εντολή POPF επιφέρει την εκτέλεση των λειτουργιών

flags ← SS:[SP+1:SP]

SP ← SP+2

ΕΝΤΟΛΕΣ ΚΛΗΣΕΩΣ ΚΑΙ ΕΠΙΣΤΡΟΦΗΣ ΥΠΟΡΟΥΤΙΝΩΝ

CALL όνομα υπορουτίνας

Μεταφέρει τον έλεγχο του προγράμματος σε κάποια υπορουτίνα, της οποίας το όνομα αναφέρεται στην εντολή. Η υπορουτίνα αυτή μπορεί να βρίσκεται μέσα στο τρέχον τμήμα κώδικα οπότε έχουμε

ενδοτμηματική κλήση υπορουτίνας ή σε διαφορετικό τμήμα κώδικα οπότε έχουμε διατμηματική κλήση υπορουτίνας. Στην περίπτωση της ενδοτμηματικής κλήσης αποθηκεύεται πρώτα το περιεχόμενο του IP στη στοίβα και στη συνέχεια μεταφέρεται ο έλεγχος στην καλούμενη υπορουτίνα. Στην περίπτωση της διατμηματικής κλήσης αποθηκεύεται πρώτα το περιεχόμενο του CS στη στοίβα, στη συνέχεια το περιεχόμενο του IP και τέλος μεταφέρεται ο έλεγχος. Και στις δύο περιπτώσεις ο SP ενημερώνεται κατάλληλα. Οι σημαίες δεν επηρεάζονται.

Παραδείγματα: Εκτελώντας την

CALL FAR DIVIDE

πραγματοποιούνται οι εξής λειτουργίες

SP ← SP-2
SS:[SP+1:SP] ← CS
CS ← 09D3h
SP ← SP-2
SS:[SP+1:SP] ← IP
IP ← 1000h

Στο παράδειγμα αυτό υποθέσαμε, ότι η procedure DIVIDE βρίσκεται στο τμήμα κώδικα με αρχική διεύθυνση 09D3h και αρχίζει στη διεύθυνση 1000h.

Για δεύτερο παράδειγμα έστω η εντολή

CALL NEAR DIVIDE

τότε αυτή προκαλεί τις εξής λειτουργίες

SP ← SP-2
SS:[SP+_1:SP] ← IP
IP ← 1000h

Στο παράδειγμα αυτό υποθέσαμε ότι η υπορουτίνα DIVIDE βρίσκεται στο τρέχον τμήμα κώδικα και αρχίζει από τη διεύθυνση 1000h.

RET

Η εντολή αυτή τοποθετείται σαν τελευταία εντολή κάθε υπορουτίνας για να εξασφαλιστεί η επιστροφή στο σημείο του προγράμματος απ' όπου αυτή κλήθηκε. Και εδώ υπάρχουν δύο περιπτώσεις. Να γίνει επιστροφή ύστερα από ενδοτμηματική κλήση ή ύστερα από διατμηματική. Ο assembler ξεχωρίζει για ποια περίπτωση πρόκειται από την εντολή CALL, με την οποία κλήθηκε η υπορουτίνα. Στην πρώτη περίπτωση η RET αντιγράφει την πρώτη λέξη της στοίβας στον IP ενώ στη δεύτερη περίπτωση η RET εκτός από τον IP αντιγράφει και τη δεύτερη λέξη της στοίβας στον CS. Τα περιεχόμενα της στοίβας δεν αλλάζουν. Ο SP ενημερώνεται κατάλληλα. Οι σημαίες δεν επηρεάζονται.

Παράδειγμα: Με την εντολή RET πραγματοποιούνται οι εξής λειτουργίες:

IP ← [SP+1:SP]
SP ← SP+2

Εδώ υποθέσαμε, ότι η εντολή RET ανήκει σε υπορουτίνα, που κλήθηκε από το ίδιο τμήμα στο οποίο περιέχεται. Η εντολή RET μπορεί όμως να σημαίνει και

IP ← [SP+1:SP]
SP ← SP+2
CS ← [SP+1:SP]
SP ← SP+2

όπου εδώ υποθέσαμε, ότι η εντολή RET ανήκει σε διατμηματική υπορουτίνα.

ΕΝΤΟΛΕΣ ΔΙΑΚΟΠΩΝ ΤΟΥ SOFTWARE

INT τύπος διακοπής

Ο τύπος διακοπής είναι ένας ακέραιος μεταξύ 0 και 255. Η εντολή INT κάνει τα ακόλουθα. Αποθηκεύει στη στοίβα τα περιεχόμενα του καταχωρητή σημαιών, του CS και τη διεύθυνση (IP) της επόμενης μετά την INT εντολής. Ο SP ενημερώνεται κατάλληλα. Στον CS μεταφέρεται το περιεχόμενο της θέσης μνήμης με διεύθυνση την (τύπος διακοπής)*4, που είναι η διεύθυνση του τμήματος κώδικα, στο οποίο βρίσκεται η ρουτίνα εξυπηρέτησης. Οι σημαίες TF και IF μηδενίζονται. Τέλος ο έλεγχος μεταφέρεται στη ρουτίνα εξυπηρέτησης της συγκεκριμένης διακοπής με την τοποθέτηση στον IP του περιεχομένου της θέσης μνήμης η

οποία έχει διεύθυνση την (τύπος διακοπής)*4+2, όπου περιέχεται η διεύθυνση (offset) αυτής της ρουτίνας.

Παράδειγμα: Η διακοπή του λογισμικού

INT 23h

αναλύεται στις εξής λειτουργίες:

SP ← SP-2
SS:[SP+1:SP] ← flags
SP ← SP-2
SS:[SP+1:SP] ← CS
SP ← SP-2
SS:[SP+1:SP] ← IP
CS ← [8D:8C]
IF ← 0
TF ← 0
IP ← [8F:8E]

INTO

Το αποτέλεσμα αυτής της εντολής είναι μια διακοπή τύπου 4 αν υπάρχει υπερχείλιση, δηλαδή αν OF=1.

Παράδειγμα: Η εντολή INTO σημαίνει:

```
if OF=1 then
        SP ← SP-2
        SS:[SP+1:SP] ← flags
        SP ← SP-2
        SS:[SP+1:SP] ← CS
        SP ← SP-2
        SS:[SP+1:SP] ← IP
        CS ← [19:18]
        IF ← 0
        TF ← 0
        IP ← [17:16]
else
        nothing
```

IRET

Τοποθετείται πάντα σαν τελευταία εντολή σε κάθε ρουτίνα εξυπηρέτησης διακοπής, για να εξασφαλίσει την επιστροφή στο σημείο του προγράμματος, όπου έγινε η διακοπή. Όταν εκτελεστεί, αντιγράφει από τη στοίβα στους αντίστοιχους καταχωρητές τα παλιά περιεχόμενα του IP, του CS και του καταχωρητή σημαιών.

Παράδειγμα: Για την εντολή IRET έχουμε:

IP ← SS:[SP+1:SP]
SP ← SP+2
CS ← SS:[SP+1:SP]
SP ← SP+2
flags ← SS:[SP+1:SP]
SP ← SP+2

ΕΝΤΟΛΕΣ ΕΛΕΓΧΟΥ ΤΟΥ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΗ

Οι επόμενες εντολές δεν έχουν ορίσματα και επηρεάζουν τις σημαίες.

STC

Θέτει τη σημαία κρατουμένου CF=1.

CLC

Μηδενίζει τη σημαία κρατουμένου CF=0.

CMC

Αντιστρέφει το κρατούμενο CF=CF'.

STD

Θέτει τη σημαία κατεύθυνσης DF=1

CLD

Μηδενίζει τη σημαία κατεύθυνσης DF=0

STI

Θέτει τη σημαία διακοπής IF=1

CLI

Μηδενίζει τη σημαία διακοπής IF=0

WAIT

Όταν εκτελείται η εντολή WAIT ο επεξεργαστής εισέρχεται σε κατάσταση αναμονής, κατά την οποία δεν εκτελεί καμιά λειτουργία, για όσο χρονικό διάστημα το σήμα εισόδου στην ακίδα "TEST" του επεξεργαστή παραμένει σε λογικό μηδέν. Εξέρχεται από την κατάσταση αυτή όταν το σήμα αυτό γίνει λογικό

μηδέν. Η εντολή αυτή χρησιμοποιείται για να συντονίζει τον επεξεργαστή με εξωτερικό hardware, όπως ο μαθηματικός συνεπεξεργαστής 8087. Αν ο επεξεργαστής βρίσκεται σε κατάσταση αναμονής και δεχθεί έγκυρο σήμα εξωτερικής διακοπής, τότε εξυπηρετεί τη διακοπή αυτή και επανέρχεται σε κατάσταση αναμονής. Οι σημαίες δεν επηρεάζονται.

NOP

Όταν εκτελεστεί η εντολή αυτή ο επεξεργαστής δεν εκτελεί καμιά λειτουργία και συνεχίζει με την επόμενη εντολή του προγράμματος.

LOCK

Η LOCK είναι ένα πρόθεμα, που τοποθετείται μπροστά από κάποια εντολή, η οποία θεωρείται κρίσιμη δηλαδή η εκτέλεσή της δεν μπορεί να διακοπεί. Με το πρόθεμα LOCK κατά τη διάρκεια εκτέλεσης της κρίσιμης εντολής, το σήμα στην ακίδα LOCK του επεξεργαστή γίνεται 1 ειδοποιώντας τους πιθανούς συνεπεξεργαστές, ότι δεν μπορούν να πάρουν υπό τον έλεγχό τους τον διάδρομο του συστήματος.

Παράδειγμα: Η εντολή

```
LOCK XCHG SEMAPHORE,AL
```

ισοδυναμεί με ανταλλαγή χωρίς κίνδυνο διακοπής.

ESC *αριθμός, προέλευση*

Αυτή η εντολή χρησιμοποιείται για την αποστολή εντολών προς τον μαθηματικό συνεπεξεργαστή 8087, ο οποίος μοιράζεται με τον επεξεργαστή τον διάδρομο του συστήματος. Οι εντολές του 8087 αναπαριστάνονται με κώδικα των 6 bits, ο οποίος ενσωματώνεται στην εντολή ESC. Κάθε φορά που ο 8088/86 διαβάζει τα bytes μιας εντολής, τα διαβάζει ταυτόχρονα και ο 8087 και τα αποθηκεύει στην ουρά του. Παρ' όλα αυτά ο 8087 χειρίζεται όλες τις εντολές του 8088/86 σαν εντολές NOP. Όταν όμως πρόκειται για την εντολή ESC ο 8087 αποκωδικοποιεί την εντολή, που προορίζεται γι' αυτόν και την εκτελεί ενώ ο 8088/86 θεωρεί την εντολή ESC σαν εντολή NOP.

HLT

Όταν εκτελεστεί η εντολή αυτή ο επεξεργαστής τίθεται σε κατάσταση αναμονής από την οποία εξέρχεται μόνον όταν δεχτεί σήμα διακοπής στις ακίδες INTR ή NMI ή όταν ενεργοποιηθεί το σήμα RESET.

Επιπρόσθετες εντολές του 80386

Ο μΕ 80386 εκτελεί όλες τις εντολές του 8086 και για τελεστές των 32 bit. Στην περίπτωση αυτή για να δηλώσουμε ότι έχουμε καταχωρητές και πράξεις των 32 bit βάζουμε το γράμμα Ε στον αντίστοιχο καταχωρητή του 8086.

BT *προορισμός, δείκτης*

(Bit Test, έλεγχος Bit**)**

BTC *προορισμός, δείκτης*

(Bit Test and Complement, έλεγχος και αντιστροφή Bit**)**

BTR *προορισμός, δείκτης*

(Bit Test and Reset, έλεγχος και μηδενισμός Bit)

BTS *προορισμός, δείκτης*

(Bit Test and Set, έλεγχος και τοποθέτηση της τιμής 1 στο Bit)

Οι εντολές αυτές επιστρέφουν μέσω της σημαίας κρατουμένου την τιμή κάποιου συγκεκριμένου bit μέσα στον προορισμό του οποίου η θέση προσδιορίζετε από το δείκτη. Επιπλέον της εντολής BT, η εντολή BTC αντικαθιστά το bit αυτό του προορισμού με το συμπλήρωμά του, η BTR το μηδενίζει ενώ η BTS το θέτει στην τιμή 1. Ο προορισμός μπορεί να είναι καταχωρητής ή θέση μνήμης των 16 ή των 32 bit, ενώ ο δείκτης είναι καταχωρητής αριθμός. Η θέση του λιγότερο σημαντικού bit είναι 0. Οι υπόλοιπες σημαίες δεν επηρεάζονται.

BSF *προορισμός, πηγή*

(Bit Scan Forward)

BSR *προορισμός, πηγή*

(Bit Scan Reverse)

Οι εντολές BSF και BSR σαρώνουν την προέλευση bit προς bit και τοποθετούν στον προορισμό τη θέση του πρώτου bit που θα συναντήσουν, το οποίο έχει την τιμή 1. Αν η προέλευση είναι 0 η σημαία μηδενισμού γίνεται 1 και ο προορισμός έχει ακαθόριστη τιμή μετά την εκτέλεση της εντολής. Διαφορετικά η σημαία αυτή γίνεται 0. Η προέλευση μπορεί να είναι καταχωρητής ή θέση μνήμης ενώ ο προορισμός είναι πάντα καταχωρητής. Η διαφορά ανάμεσα στις δύο εντολές είναι ότι η BSF αρχίζει τη σάρωση ξεκινώντας από το λιγότερο σημαντικό bit της προέλευσης, ενώ η BSR αρχίζει από το πιο σημαντικό bit. Οι υπόλοιπες σημαίες δεν επηρεάζονται.

MOVSX *προορισμός, προέλευση*

(MOVe with Signe eXtension)

MOVZX *προορισμός, προέλευση*

(MOVe with Zero eXtension)

Οι εντολές αυτές αντιγράφουν ένα δεδομένο των 8 bit σε έναν καταχωρητή των 16 ή των 32 bit και ένα δεδομένο των 16 bit σε έναν καταχωρητή των 32 bit. Κατά την αντιγραφή η MOVSX κάνει επέκταση προσήμου ενώ η MOVZX γεμίζει τα επιπλέον bit του καταχωρητή. Ο προορισμός είναι ο καταχωρητής στον οποίο γίνεται η αντιγραφή ενώ η προέλευση περιέχει το δεδομένο και μπορεί να είναι καταχωρητής ή θέση μνήμης. Καμία σημαία δεν επηρεάζεται.

SET *cc προορισμός*

(Set on condition)

Η εντολή θέτει στον προορισμό, που μπορεί να είναι καταχωρητής ή θέση μνήμης μήκους πάντα 1 byte, την τιμή 1 αν η συνθήκη που καθορίζεται από την εντολή είναι αληθής. Διαφορετικά θέτει την τιμή 0. Η συνθήκη καθορίζεται από το μετάθεμα cc στην εντολή (π.χ. SETZ, SETLE κλπ.) όπως ακριβώς συμβαίνει και με την εντολή άλματος υπό συνθήκη JC. Καμία σημαία δεν επηρεάζεται.

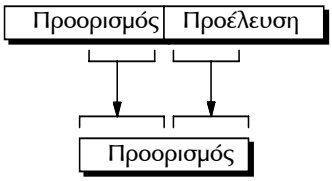
SHLD *προορισμός, προέλευση, μετρητής*

(Shift Left Double)

SHRD *προορισμός, προέλευση, μετρητής*

(Shift Left Right)

Ο προορισμός και η προέλευση συνενώνονται και ολισθαίνουν αριστερά (με την SHLD) ή δεξιά (με την SHRD) τόσες θέσεις όσες ορίζει ο μετρητής. Το αποτέλεσμα υποδιαιρείται σε τέσσερα ίσα μέρη, από τα οποία το δεύτερο και το τρίτο αποθηκεύονται στον προορισμό, όπως δείχνει και το επόμενο σχήμα.



Για τις σημαίες ισχύουν τα ίδια με τις εντολές SHR και SHL. Ο προορισμός μπορεί να είναι καταχωρητής ή θέση μνήμης, η προέλευση μόνο καταχωρητής και ο μετρητής αριθμός ή ο καταχωρητής CL. Κατά την εκτέλεση της εντολής τίθεται μάσκα με την τιμή 1FH στο μετρητή έτσι ώστε ο αριθμός των ολισθήσεων να μην υπερβεί το 31.

LDS/LES/LFS/LGS/LSS *προορισμός, προέλευση*

(Load DS\ES\FS\GS\SS)

Στη θέση μνήμης που καθορίζει η προέλευση βρίσκεται αποθηκευμένος ένας δείκτης, δηλαδή η πλήρης διεύθυνση μιας άλλης θέσης στη μνήμη. Η διεύθυνση αυτή αποτελείται από το offset (32 bit) ακολουθούμενο από τον επιλογέα (16 bit). Η εντολή αυτή φορτώνει το offset στον προορισμό και τον επιλογέα (selector) στον καταχωρητή τμήματος DS/ES/FS/GS/SS. Καμία σημαία δεν επηρεάζεται.

LGDT/LLDT/LIDT *προέλευση*

(Load GDT/LDT/IDT)

Με τις εντολές αυτές εκχωρούνται τιμές στους καταχωρητές GDTR, LDTR και LDTR αντίστοιχα. Η προέλευση πρέπει να είναι μια θέση μνήμης συνολικού μεγέθους 48 bit, από τα οποία στα 16πρώτα φυλάγεται το όριο του πίνακα και στα υπόλοιπα η διεύθυνση βάσης (αρχική διεύθυνση) του πίνακα. Καμία σημαία δεν επηρεάζεται.

SGDT/SLDT/SIDT *προορισμός*

(Save GDT/LDT/IDT)

Οι τιμές των καταχωρητών GDTR, LDTR και IDTR αποθηκεύονται στη μνήμη στη θέση που καθορίζει ο προορισμός. Καμία σημαία δεν επηρεάζεται

LTR/STR *προέλευση/προορισμός*

(Load/Save TR)

Ο καταχωρητής TR φορτώνεται ή σώζεται από ή προς τη θέση μνήμης ή τον καταχωρητή που καθορίζει η προέλευση ή ο προορισμός αντίστοιχα. Καμία σημαία δεν επηρεάζεται.

LMSW/SMSW προέλευση/προορισμός

(Load/Save MSW)

Ο καταχωρητής CR0, που περιέχει τη λέξη κατάστασης μηχανής ή MSW (Machine Status Word), φορτώνεται ή σώζεται από ή προς τη θέση μνήμης ή τον καταχωρητή που καθορίζει η προέλευση ή ο προορισμός αντίστοιχα. Καμία σημαία δεν επηρεάζεται.

CLTS

(CLear TS bit)

Μηδενίζεται το bit εναλλαγής διεργασιών TS (Task Switch bit) του καταχωρητή CR0. Δεν επηρεάζει καμία σημαία.

LSL προορισμός, προέλευση

(Load Segment Limit)

Η εντολή LSL φορτώνει σε κάποιον καταχωρητή που δίνεται σαν προορισμός το όριο (limit) ενός τμήματος, που περιέχεται στον περιγραφητή τμήματος (segment descriptor). Ο περιγραφητής αυτός προσδιορίζεται από την προέλευση, η οποία περιέχει έναν επιλογέα (selector) που δείχνει σε αυτόν. Η προέλευση μπορεί να είναι καταχωρητής ή θέση μνήμης. Επιπλέον η εντολή θέτει στη σημαία μηδενισμού την τιμή 1, ενώ αν το τμήμα αυτό δεν είναι προσπελάσιμο ή ο περιγραφητής δεν περιέχει όριο, θέτει την τιμή 0. Οι υπόλοιπες σημαίες δεν επηρεάζονται.

VERR/VERW επιλογέας

(VERify Read/Write)

Με αυτές τις εντολές ελέγχεται αν η τρέχουσα διεργασία έχει δικαίωμα προσπέλασης και ανάγνωσης/ εγγραφής στο συγκεκριμένο τμήμα που καθορίζει ο επιλογέας. Αν συμβαίνει αυτό στη σημαία μηδενισμού τοποθετείται η τιμή 1 διαφορετικά τοποθετείται η τιμή 0. Οι υπόλοιπες σημαίες δεν επηρεάζονται

BOUND προορισμός, προέλευση

Η εντολή χρησιμεύει στην κατασκευή μεταγλωττιστών (compiler) και ειδικότερα στη προσπέλαση πινάκων. Πιο συγκεκριμένα όταν πρόκειται να προσπελαστεί κάποιο στοιχείο του πίνακα, με την εντολή BOUND μπορεί να ελεγχθεί αν η τιμή του δείκτη βρίσκεται μέσα στα όρια του πίνακα. Αν δε βρίσκεται προκαλείται διακοπή τύπου 5. Ο προορισμός περιέχει την τιμή του δείκτη και είναι πάντα καταχωρητής. Η προέλευση είναι μια θέση μνήμης όπου βρίσκεται αποθηκευμένο το κάτω όριο. Στην αμέσως επόμενη θέση πρέπει να βρίσκεται το άνω όριο του πίνακα. Οι θέσεις αυτές θα είναι των 16 ή των 32 bit και αυτό καθορίζεται από το αν ο προορισμός είναι των 16 ή των 32 bit αντίστοιχα. Καμία σημαία δεν επηρεάζεται.

ENTER s, n

Η εντολή χρησιμεύει στην κατασκευή μεταγλωττιστών (compiler) και ειδικότερα στην κλήση υπορουτινών. Το όρισμα s της εντολής είναι ένας αριθμός που καθορίζει πόσος χώρος σε bytes πρέπει να δεσμευτεί στη στοίβα για τις τοπικές μεταβλητές της υπορουτίνας. Το όρισμα n είναι πάλι ένας αριθμός που δηλώνει το επίπεδο φωλιάσματος, δηλαδή σε περίπτωση που η συγκεκριμένη υπορουτίνα είναι φωλιασμένη μέσα σε μια άλλη υπορουτίνα, όπως για παράδειγμα μπορεί να συμβεί στη γλώσσα Pascal, σε ποια θέση βρίσκεται. Αν για παράδειγμα έχουμε τρεις υπορουτίνες τη μια δηλωμένη μέσα στην άλλη τότε για την πιο εσωτερική από αυτές το n είναι 3. Για τις γλώσσες που δεν υποστηρίζουν φωλιασμένες υπορουτίνες, όπως για παράδειγμα η γλώσσα C, το n είναι πάντα 0. Όταν εκτελεστεί η ENTER κάνει τα ακόλουθα:

1. Ο EBP φυλάγεται στη στοίβα.
2. Στον EBP εκχωρείται ο EBS.
3. Το βήμα αυτό εκτελείται μόνο όταν η υπορουτίνα είναι φωλιασμένη μέσα σε κάποιες άλλες, δηλ. n>0. Στην περίπτωση αυτή

τοποθετούνται με τη σειρά στη στοίβα δείκτες που δείχνει στις περιοχές της στοίβας που βρίσκονται οι παράμετροι και οι τοπικές μεταβλητές των πιο πάνω υπορουτινών, έτσι ώστε η τωρινή υπορουτίνα να μπορεί να τις προσπελάσει

4. Από τον ESP αφαιρείται ο αριθμός s, έτσι ώστε να δημιουργηθεί στη στοίβα χώρος για τις τοπικές μεταβλητές.

Προφανώς η εντολή αυτή πρέπει να τοποθετηθεί σαν πρώτη εντολή στην υπορουτίνα. Καμία σημαία δεν επηρεάζεται.

LEAVE

Η εντολή LEAVE κάνει ακριβώς τις αντίστροφες λειτουργίες που εκτελεί η ENTER και έτσι απελευθερώνει τη στοίβα από το χώρο που δέσμευσε η κλήση της υπορουτίνας. Προφανώς η εντολή αυτή πρέπει να χρησιμοποιηθεί οπωσδήποτε σε μια υπορουτίνα, που αρχίζει με την εντολή ENTER, πριν από την εντολή επιστροφής. Καμία σημαία δεν επηρεάζεται.

Επιπρόσθετες εντολές του 80486

BSWAP προορισμός

(Byte SWAP)

Με την εντολή BSWAP τα bytes που περιέχονται στον προορισμό, ο οποίος είναι πάντα ένας καταχωρητής των 32 bit, τοποθετούνται με αντίστροφη σειρά. Έτσι το πρώτο byte τοποθετείται τέταρτο, το δεύτερο τρίτο, το τρίτο δεύτερο και το τέταρτο πρώτο. Καμία σημαία δεν επηρεάζεται.

XADD προορισμός, προέλευση

(eXchange and ADD)

Η εντολή χειρίζεται ορίσματα μόνο των 16 bit. Το περιεχόμενο της προέλευσης προστίθεται στον προορισμό και το αποτέλεσμα αποθηκεύεται στον προορισμό. Στην προέλευση τοποθετείται το αρχικό περιεχόμενο του προορισμού. Ο προορισμός μπορεί να είναι καταχωρητής ή θέση μνήμης των 16 bit και η προέλευση μόνο καταχωρητής των 16 bit. Για τις σημαίες ισχύουν τα ίδια με την εντολή ADD.

CMPXCHG προορισμός, προέλευση

(CoMPare and eXCHanGe)

Το περιεχόμενο της προέλευση συγκρίνεται με το συσσωρευτή (AL, AX ή EAX). Αν είναι ίδιο τοποθετείται στον προορισμό. Διαφορετικά στο συσσωρευτή τοποθετείται το περιεχόμενο του προορισμού. Οι σημαίες επηρεάζονται όπως ακριβώς και στην εντολή CMP.

INVD

(INValiDate cache)

WBINVD

(Write Back)

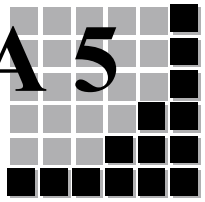
Οι εντολές αυτές ακυρώνουν τα τρέχοντα περιεχόμενα της εσωτερικής λανθάνουσας μνήμης. Με αυτό τον τρόπο δεν χρησιμοποιούνται πλέον από τον επεξεργαστή αλλά αντικαθίστανται σταδιακά με νέα κάθε φορά που ο διαβάζει από τη μνήμη. Επιπλέον η εντολή WBINVD πριν την ακύρωση γράφει όλα τα περιεχόμενα της μνήμης αυτής στην κύρια μνήμη. Καμία σημαία δεν επηρεάζεται.

INVLPG σελίδα

(INVaLidate Page)

Ακυρώνει τη συγκεκριμένη καταχώρηση στον TLB, η οποία περιέχει τη συγκεκριμένη διεύθυνση σελίδας που καθορίζεται από το όρισμα της εντολής. Καμία σημαία δεν επηρεάζεται

ΠΑΡΑΡΤΗΜΑ 5



Ρουτίνες BIOS και DOS

- 1. Ρουτίνες του BIOS
- 2. Ρουτίνες του MS-DOS

1. Ρουτίνες του BIOS

Στην παράγραφο αυτή περιγράφεται η κλήση μερικών από τις πιο βασικές ρουτίνες του BIOS. Στη συνέχεια ακολουθείται η εξής σύμβαση: με **Κλήση** δηλώνουμε το ποια θα πρέπει να είναι η κατάσταση των καταχωρητών πριν την κλήση της συνάρτησης, με **Επιστροφή** δηλώνουμε την κατάσταση των καταχωρητών μετά την κλήση και με **Σχόλια** αναφερόμαστε σε σύντομο επεξηγηματικό σχόλιο σε σχέση με τη λειτουργία της ρουτίνας.

ΡΟΥΤΙΝΕΣ ΟΘΟΝΗΣ

1. Int 10/00

Κλήση:
AH ← 00h
AL ← κωδικός mode οθόνης
0→ 40×25 B/W text CGA-EGA,
1→ 40×25 16 color text CGA-EGA,
2→ 80×25 B/W text CGA-EGA,
3→ 80×25 16 color text CGA-EGA,
4→ 320×200 4 color gr CGA-EGA,
5→ 320×200 4 grays gr CGA-EGA,
6→ 640×200 B/W gr CGA-EGA,
7→ 80×25 B/W text Monochrome,
8→ 160×200 16 color gr PCjr,
9→ 320×200 16 color gr PCjr,
10→ 640×200 4 color gr Pcjr,
13→ 320×200 16 color gr EGA,
14→ 640×200 16 color gr EGA,
15→ 640×350 4 color gr EGA
Επιστροφή: -
Σχόλια: θέτει την οθόνη στην κατάσταση που αντιστοιχεί στον κωδικό.

2. Int 10/02

Κλήση:
AH ← 02h
DH ← αριθμός γραμμής
DL ← αριθμός στήλης
BH ← αριθμός σελίδας
Επιστροφή: -
Σχόλια: τοποθετεί το δρομέα (cursor) σε συγκεκριμένη θέση σε μία από τις σελίδες της μνήμης που αντιστοιχούν στην οθόνη. Γενικά, μπορούν να υπάρχουν παραπάνω από ένα τμήματα μνήμης τα οποία αντιστοιχούν στην οθόνη (ανάλογα με την κάρτα οθόνης), οι σελίδες. Σε κάθε στιγμή μία μόνο σελίδα είναι ενεργή και τα περιεχόμενά της απεικονίζονται στην οθόνη. Μπορούμε όμως ελεύθερα να γράψουμε και στις άλλες.

3. Int 10/03

Κλήση:
AH ← 03h
BH ← αριθμός σελίδας
Επιστροφή:
DH ← αριθμός γραμμής
DL ← αριθμός στήλης
CX ← κωδικός mode
Σχόλια: επιστρέφει τη θέση του cursor στη συγκεκριμένη σελίδα καθώς και το mode της.

4. Int 10/05

Κλήση:
AH ← 05h
AL ← αριθμός σελίδας
Επιστροφή: -
Σχόλια: καθιστά μία σελίδα ενεργή.

5. Int 10/06

Κλήση:
AH ← 06h
AL ← αριθμός γραμμών για μετακίνηση. Αν AL=0 μετακινούνται όλες οι γραμμές της οθόνης.
CH ← γραμμή πάνω αριστερά άκρου
CL ← στήλη πάνω αριστερά άκρου
DH ← γραμμή κάτω δεξιά άκρου
DL ← στήλη κάτω δεξιά άκρου
BH ← χαρακτηριστικά νέων γραμμών
Επιστροφή: -
Σχόλια: μετακινεί ένα καθοριζόμενο παράθυρο της οθόνης προς τα πάνω τόσες γραμμές όσες το περιεχόμενο του AL.

6. Int 10/07

Κλήση:
AH ← 07h
Τα υπόλοιπα όπως παραπάνω
Επιστροφή: -
Σχόλια: κίνηση του παράθυρου προς τα κάτω.

7. Int 10/08

Κλήση:
AH ← 08h
BH ← αριθμός σελίδας
Επιστροφή:
AL ← ASCII κωδικός χαρακτήρα
AH ← χαρακτηριστικό χαρακτήρα
Σχόλια: επιστρέφει το χαρακτήρα και το χαρακτηριστικό που βρίσκεται στη θέση του cursor.

8. Int 10/09

Κλήση:
AH ← 09h
BH ← αριθμός σελίδας
BL ← κωδικός χαρακτηριστικού
CX ← αριθμός χαρακτήρων που θα τυπωθούν
AL ← ASCII κωδικός χαρακτήρα
Επιστροφή: -
Σχόλια: τυπώνει έναν αριθμό όμοιων χαρακτήρων ξεκινώντας από εκεί που βρίσκεται ο cursor.

9. Int 10/0A

Κλήση:
AH ← 0Ah
BH ← αριθμός σελίδας
CX ← αριθμός χαρακτήρων που θα τυπωθούν
AL ← ASCII κωδικός χαρακτήρα
Επιστροφή: -
Σχόλια: όπως παραπάνω με τη διαφορά πως δεν ορίζεται το χαρακτηριστικό.

10. Int 10/0E

Κλήση:
AH ← 0Eh
AL ← ASCII κωδικός χαρακτήρα
BL ← κωδικός χαρακτηριστικού
BH ← αριθμός σελίδας
Επιστροφή: -
Σχόλια: τυπώνει ένα χαρακτήρα στη θέση του cursor και προχωράει το δεύτερο μία θέση.

ΜΕΓΕΘΟΣ ΜΝΗΜΗΣ

11. Int 12

Κλήση: -
Επιστροφή:
AX ← αριθμός των block μνήμης μεγέθους 1K
Σχόλια: επιστρέφει τον αριθμό των συνεχόμενων block μνήμης μεγέθους 1K που μετρήθηκαν κατά την αρχικοποίηση του H/Y.

Ε/Ε ΔΙΣΚΟΥ

12. Int 13/00

Κλήση:
AH ← 00h
Επιστροφή:
DL ← αριθμός οδηγού δίσκου
(0→a:, 1→b:,...)
bit7= 0 ⇒ δισκέτα,
bit7 = 1 ⇒ σκληρός δίσκος
Σχόλια: αρχικοποιεί τον οδηγό δίσκου προετοιμάζοντας τον για ανάγνωση ή/και θέτοντας το reset flag του ελεγκτή του δίσκου και αναγκάζοντας έτσι την κεφαλή να επιστρέψει στην τροχιά 0.

13. Int 13/01

Κλήση:
AH ← 00h
Επιστροφή:
AH ← λέξη κατάστασης του οδηγού δίσκου (δες στο τέλος πιν. 1)
Σχόλια: με την ρουτίνα αυτή ο χρήστης μπορεί να βρει την κατάσταση του δίσκου μετά την τελευταία λειτουργία του.

14. Int 13/02

Κλήση:
AH ← 02H
AL ← τομείς για ανάγνωση (1,..,9)
ES:BX ← δείκτης στον buffer δίσκου του χρήστη
CH ← αριθμός της τροχιάς (0,..,39)
CL ← αριθμός του τομέα (0,..,9)
DH ← αριθμός της κεφαλής (0 ή 1)
DL ← αριθμός οδηγού δίσκου (0,..,3)
Επιστροφή:
CF ← 1 σε περίπτωση λάθους οπότε
AH ← byte κατάστασης (δες πιν. 1.)
CF ← 0 αν υπήρξε επιτυχία οπότε
AH ← 0
AL ← ο αριθμός των τομέων που μεταφέρθηκαν
Σχόλια: μεταφέρει ένα ή περισσότερους τομείς από τον δίσκο στην μνήμη. Χρειάζεται ιδιαίτερη προσοχή κατά τον ορισμό των πιο πάνω παραμέτρων πριν την κλήση.

15. Int 13/03

Κλήση:
AH ← 03H
AL ← τομείς για μεταφορά (1,..,9)
ES:BX ← δείκτης στον buffer δίσκου του χρήστη
CH ← αριθμός της τροχιάς (0,..,39)
CL ← αριθμός του τομέα (0,..,9)
DH ← αριθμός της κεφαλής (0 ή 1)
DL ← αριθμός οδηγού δίσκου (0,..,3)
Επιστροφή:
CF ← 1 σε περίπτωση λάθους οπότε
AH ← byte κατάστασης (δες πιν. 1.)
CF ← 0 σε περίπτωση επιτυχίας οπότε
AH ← 0
AL ← ο αριθμός των τομέων που μεταφέρθηκαν
Σχόλια: γράφει ένα ή περισσότερους τομείς από την μνήμη στον δίσκο. Χρειάζεται ιδιαίτερη προσοχή κατά τον ορισμό των πιο πάνω παραμέτρων πριν την κλήση.

16. Int 13/05

Κλήση:
AH ← 05h
ES:BX ← δείκτης στην λίστα των πεδίων διευθύνσεων τροχιών (track address field list)
CH ← αριθμός τροχιών
DH ← αριθμός κεφαλής
DL ← αριθμός οδηγού δίσκου
Επιστροφή:
AH ← κώδικας επιστροφής (δες πίνακα 1)
Σχόλια: πραγματοποιεί format μιας τροχιάς του δίσκου αρχικοποιώντας τα πεδία διευθύν-

σεων του δίσκου καθώς και οι τομείς του δίσκου.

Πίνακας 1 Λέξη κατάστασης ελεγκτή δίσκου

.....1	άγνωστη εντολή προς τον ελεγκτή
.....1.	χαλασμένος τομέας (δε βρέθηκε η σωστή διεύθυνση)
.....11	δίσκος write-protected
.....1..	δε βρέθηκε ο ζητούμενος τομέας
.....11.	γραμμή αλλαγής δισκέτας ON
....1...	προσπέλαση DMA
....1..1	προσπάθεια DMA πέραν του ορίου των 64K
....11..	χαλασμένος δίσκος ή δίσκος άλλου τύπου από τον οδηγό
...1....	CRC λάθος στην προσπάθεια διαβάσματος του δίσκου
..1.....	λάθος του ελεγκτή
.1.....	αποτυχής αναζήτηση
1.....	δίσκος όχι έτοιμος

Ε/Ε ΣΕΙΡΙΑΚΗΣ ΘΥΡΑΣ

17. Int 14/00

Κλήση:
AH ← 00h
AL ← παράμετρος αρχικοποίησης
DX ← αριθμός πόρτας (0=COM1, 1=COM2, 2=COM3, 3= COM4)

Επιστροφή:
AH ← κατάσταση της πόρτας
AL ← κατάσταση του modem

Σχόλια: χρήση για την αρχικοποίηση της σειριακής πόρτας που δείχνει ο DX. Ο τρόπος της αρχικοποίησης δηλώνεται από το byte στον AL:

Πεδίο	Έννοια
Bits 7, 6, 5	Baud-rate
Bits 4, 3	Ισοτιμία
Bit 2	Stop-bits
Bits 1, 0	Μήκος λέξης

Bits 7, 6, 5	Baud-rate
000	110 baud
001	150 baud
010	300 baud
011	600 baud
100	1200 baud
101	2400 baud
110	4800 baud
111	9600 baud

Bits 4, 3	Ισοτιμία
X0	Καμία
01	Περιττή
11	Άρτια

Bit 2	Ψηφία Τερμ.
0	1 bit
1	2 bits

Bits 1,0	Μήκος Λέξης
10	7 bits
11	8 bits

18. Int 14/01

Κλήση:
AH ← 01h
AL ← ASCII κωδικός χαρακτήρα
DX ← αριθμός πόρτας (0=COM1, 1=COM2, 2=COM3, 3=COM4)

Επιστροφή:
AH ← bit7 = 0 ⇒ επιτυχία,
AH ← bit7 = 1 ⇒ αποτυχία

Σχόλια: γράφει ένα χαρακτήρα στην σειριακή πόρτα που δείχνει ο DX. Πριν την κλήση αυτής της ρουτίνας η σειριακή πόρτα πρέπει να αρχικοποιηθεί με την INT 14/00.

19. Int 14/02

Κλήση:
AH ← 02h
DX ← αριθμός πόρτας (0=COM1, 1=COM2, 2=COM3, 3=COM4)

Επιστροφή:
AH ← bit7 = 0 ⇒ επιτυχία

AL ← κωδικό ASCII του χαρακτήρα που διαβάστηκε
AH ← bit7 = 1 ⇒ αποτυχία

Σχόλια: διαβάζει ένα χαρακτήρα από την οριζόμενη σειριακή πόρτα. Πριν την κλήση αυτής της ρουτίνας η σειριακή πόρτα πρέπει να αρχικοποιηθεί με την INT 14/00.

Ε/Ε ΠΛΗΚΤΡΟΛΟΓΙΟΥ

20. Int 16/00

Κλήση:
AH ← 00h
Επιστροφή:
AH ← κωδικός ανίχνευσης πληκτρολογίου (Keyboard Scan Code)
AL ← κωδικός ASCII του χαρακτήρα

Σχόλια: περιμένει μέχρι να πατηθεί κάποιο πλήκτρο και επιστρέφει τον ASCII κωδικό του καθώς και τον κωδικό ανίχνευσης του. Σε περίπτωση που πατηθεί κάποιο ειδικό πλήκτρο ο AL επιστρέφει 0 και ο AH τον εκτεταμένο ASCII (extended ASCII) κωδικό αυτού του πλήκτρου.

Πίνακας 2 Λέξη κατάστασης του πληκτρολογίου

.......0	δεξιό shift πατημένο
......0.	αριστ. shift πατημένο
.....0..	Ctrl-Shift πατημένο
....0...	Alt-Shift πατημένο
...1....	ScrollLock ενεργοποιημένο
..1.....	NumLock ενεργοποιημένο
.1.....	CapsLock ενεργοποιημένο
1.....	Insert ενεργοποιημένο

21. Int 16/02

Κλήση: -
Επιστροφή:
AL ← λέξη κατάστασης πληκτρολογίου

Σχόλια: Εξετάζει αν είναι πατημένο κάποιο από τα πλήκτρα Shift, NumLock και CapsLock. Οι πληροφορίες αυτές δίνονται από τη λέξη κατάστασης πληκτρολογίου, που επιστρέφεται μέσω του AL (δες πίνακα 2).

Ε/Ε ΕΚΤΥΠΩΤΗ

22. Int 17/00

Κλήση:
AH ← 00h
AL ← κωδικός χαρακτήρα
DX ← αριθμός εκτυπωτή (0, 1 ή 2)
Επιστροφή:
AH ← λέξη κατάστασης εκτυπωτή

Σχόλια: γράφει τον οριζόμενο χαρακτήρα στον εκτυπωτή και επιστρέφει την λέξη κατάστασης του εκτυπωτή (δες πίνακα 3.)

Πίνακας 3 Λέξη κατάστασης του εκτυπωτή

.......1	time-out (εκτυπωτής off-line ή σβηστός)
.....xx.	δε χρησιμοποιούνται
....1...	λάθος ανάγν./εγγραφής
...1....	εκτυπωτής επιλέχθηκε
..1.....	εκτυπωτής χωρίς χαρτί
.1.....	αναγνώριση αίτησης εκτύπωσης
1.....	εκτυπωτής σε αναμονή (μη απασχολημένος)

23. Int 17/01

Κλήση:
AH ← 01h
DX ← αριθμός εκτυπωτή (0, 1 ή 2)
Επιστροφή:
AH ← λέξη κατάστασης εκτυπωτή

Σχόλια: αρχικοποιεί την παράλληλη θύρα και επιστρέφει την λέξη κατάστασης του εκτυπωτή. Η ρουτίνα αυτή εκπέμπει τους χαρακτήρες 08h 02h στην θύρα του εκτυπωτή. Οι εκτυπωτές τύπου EPSON ή IBM και οι συμβατοί ανταποκρίνονται σ' αυτούς τους χαρακτήρες με αρχικοποίηση.

ΕΠΑΝΕΚΚΙΝΗΣΗ

24. Int 19

Κλήση: -
Επιστροφή: -
Σχόλια: η συνάρτηση αυτή είναι παρόμοια με το πάτημα των πλήκτρων CTRL-ALT-DEL και πραγματοποιεί ένα warm boot.

Ε/Ε ΧΡΟΝΟΥ

25. Int 1A/00

Κλήση: AH ← 00h
Επιστροφή:
AL ← flag για μεσάνυχτα
CX:DX← μετρητής ρολογιού

Σχόλια: επιστρέφει τον μετρητή του ρολογιού που "κτυπά" 18.2065 φορές το δευτερόλεπτο. Μηδέν σημαίνει μεσάνυχτα

26. Int 1A/01

Κλήση:
AH← 01h
CX:DX← τιμή του μετρητή
Επιστροφή: -
Σχόλια: για να θέσετε το ρολόι σε μια συγκεκριμένη ώρα υπολογίστε τον αριθμό των "κτύπων" (ticks) που χρειάζεται πολλαπλασιάζοντας τον αριθμό των δευτερολέπτων μετά τα μεσάνυχτα μέχρι την σωστή ώρα επί το 18.2065.

2. Ρουτίνες του MS-DOS

Στη συνέχεια θα αναφερθούμε στην κλήση (και χρήση) ενός υποσυνόλου από τις ρουτίνες του MS-DOS, τις πιο χρήσιμες.

Όσον αφορά τους συμβολισμούς **Κλήση**, **Επιστροφή** και **Σχόλια** ισχύουν τα ίδια με αυτά που αναφέρθηκαν πιο πάνω για τις ρουτίνες του BIOS. Επίσης, όπου αναφέρεται το PSP τούτο συμβολίζει το Program Segment Prefix δηλαδή τις τιμές των DS και ES κατά τη στιγμή που αρχίζει να εκτελείται το πρόγραμμα. Μια συμβολοσειρά που είναι σε μορφή ASCIIZ αποτελείται από μια σειρά χαρακτήρων που τερματίζονται με το χαρακτήρα '/0'. Τα αρχικά FCB συμβολίζουν το File Control Block το οποίο περιέχει όλες τις σχετικές με ένα αρχείο πληροφορίες που είναι απαραίτητες για την επεξεργασία του.

Οι ρουτίνες του DOS, μπορούν να ταξινομηθούν στις ακόλουθες κατηγορίες:

1) Ρουτίνες χειρισμού δίσκων και αρχείων
   INT 21/0D, 21/0E, 21/2E, 21/30, 21/36, 21/39, 21/3A, 21/3B, 21/3C, 21/3D, 21/3E, 21/3F, 21/40, 21/41, 21/42, 21/43, 21/47, 21/54, 21/56, 21/57

2) Ρουτίνες διαχείρισης διεργασιών
   INT 21/31, 21/4B, 21/4C, 21/4D

3) Ρουτίνες διαχείρισης διακοπών
   INT 21/25, 21/35

4) Ρουτίνες χειρισμού σειριακής θύρας
   INT 21/03, 21/04

5) Ρουτίνες έλεγχου timer
   INT 21/2A, 21/2B, 21/2C, 21/2D

6) Ρουτίνες χειρισμού οθόνης
   INT 21/02, 21/05, 21/06, 21/09

7) Ρουτίνες χειρισμού πληκτρολογίου
   INT 21/01, 21/06, 21/08, 21/0A, 21/0B

8) Ρουτίνες τερματισμού
   INT 20, INT 21/00, INT 27

1. Int 20

(Terminate program)
Κλήση:
CS ← διεύθυνση τμήματος (segment) του PSP
Επιστροφή: -
Σχόλια: επιτρέπει τον τερματισμό ενός προγράμματος και την απελευθέρωση της μνήμης που ήταν δεσμευμένη από αυτό. Αντί αυτής της ρουτίνας, συνιστάται η χρήση της Int 21/4C για εκδόσεις του DOS μεταγενέστερες της 2.0. Η πιο πάνω ρουτίνα παραμένει μόνο και μόνο για συμβατότητα με παλαιότερα προγράμματα τα οποία τρέχουν κάτω από εκδόσεις του DOS πριν την 2.0.

2. Int 21/00

(Terminate program)
Κλήση:
AH ← 00h
CS ← διεύθυνση τμήματος του PSP
Επιστροφή: -
Σχόλια: ίδια με την Int 20.

3. Int 21/01

(Wait key)
Κλήση:
AH ← 01h
Επιστροφή:
AL ← 8 - bits κωδικός ASCII
Σχόλια: περιμένει μέχρι να πατηθεί κάποιο πλήκτρο και αφού τυπώσει τον αντίστοιχο χαρακτήρα στην οθόνη επιστρέφει τον κωδικό του στον AL.

4. Int 21/02

(Display char)
Κλήση:
AH ← 02h
DL ← 8-bits κωδικός ASCII
Επιστροφή: -
Σχόλια: τυπώνει στην οθόνη τον χαρακτήρα του οποίου ο ASCII κωδικός βρίσκεται στον DL.

5. Int 21/03

(Wait serial port)
Κλήση: AH ← 03h
Επιστροφή: AL ← 8-bits κωδικός ASCII
Σχόλια: περιμένει μέχρι να λάβει κάποιο χαρακτήρα από την πρώτη σειριακή θύρα COM1 και επιστρέφει τον κωδικό του στον AL.

6. Int 21/04

(Output serial port)
Κλήση:
AH ← 04h
DL ← 8-bits κωδικός ASCII
Επιστροφή: -
Σχόλια: χρησιμοποιείται για να στείλει ένα χαρακτήρα στην πρώτη σειριακή θύρα COM1. Αν αυτή δεν είναι ελεύθερη τότε η ρουτίνα περιμένει μέχρι να ελευθερωθεί και τότε στέλνει τον χαρακτήρα.

7. Int 21/05

(Print char)
Κλήση:
AH ← 05h
DL ← 8-bits κωδικός ASCII
Επιστροφή: -
Σχόλια: περιμένει μέχρι ο εκτυπωτής να είναι έτοιμος και τότε στέλνει ένα byte (τον χαρακτήρα που βρίσκεται στον DL).

8. Int 21/06

Κλήση:
AH ← 06h
DL ← 00h - FEh ⇒ εκτύπωση αυτού του χαρακτήρα στην οθόνη
DL ← FFh ⇒ έλεγχος αν πατήθηκε πλήκτρο
Επιστροφή: τίποτα αν εκτυπώνεται κάποιος χαρακτήρας
Αν γίνεται έλεγχος για πάτημα πλήκτρου:
Αν ZF←1 τότε δεν πατήθηκε πλήκτρο

Αν ZF←0 πατήθηκε πλήκτρο και
στον AL περιέχεται ο 8-bits ASCII
κωδικός.

Σχόλια: εκτυπώνει ή διαβάζει χαρακτήρες ανάλογα με την τιμή του DL. Στην περίπτωση που διαβάζει κάποιο χαρακτήρα η ρουτίνα αυτή δεν περιμένει μέχρι να πατηθεί το πλήκτρο. Επιπλέον είναι φανερό ότι ο χαρακτήρας με κωδικό FFh δεν μπορεί να τυπωθεί με αυτή τη συνάρτηση.

9. Int 21/08

(Wait key)
Κλήση: AH ← 08h
Επιστροφή: AL ← 8-bits κωδικός ASCII
Σχόλια: περιμένει μέχρι να πατηθεί ένα πλήκτρο και επιστρέφει τον κωδικό του στον AL (χωρίς να τυπωθεί στην οθόνη ο χαρακτήρας).

10. Int 21/09

(Display string)
Κλήση:
AH ← 09h
DS:DX ← δείκτης σε συμβολοσειρά που
τερματίζεται με '$'
Επιστροφή: -
Σχόλια: εκτυπώνει στην οθόνη μια σειρά από συνεχόμενους χαρακτήρες όπως η Int 21/02 εκτυπώνει απλούς χαρακτήρες. Εκτυπώνονται όλοι οι χαρακτήρες από την διεύθυνση που δείχνουν οι DS:DX μέχρι να βρεθεί το σύμβολο '$'.

11. Int 21/0A

(Read string)
Κλήση:
AH ← 0Ah
DS:DX ← δείκτης στο buffer δεδομένων
Επιστροφή: -
Σχόλια: επιτρέπει την είσοδο δεδομένων από το πληκτρολόγιο και την αποθήκευση τους εκεί που δείχνουν οι DS:DX. Για να χρησιμοποιήσετε αυτή την συνάρτηση θα πρέπει στο πρώτο byte του buffer που δείχνουν οι DS:DX να σώσετε τον μέγιστο αριθμό bytes που θέλετε να διαβάσετε. Στο δεύτερο byte θα επιστραφεί ο αριθμός των bytes που διαβάστηκαν τελικά.

12. Int 21/0B

(Keyboard status)
Κλήση: AH ← 0Bh
Επιστροφή:
AL ← 00h ⇒ δεν πατήθηκε πλήκτρο
AL ← FFh ⇒ πατήθηκε κάποιο πλήκτρο
Σχόλια: ελέγχει αν πατήθηκε κάποιο πλήκτρο χωρίς όμως να επιστρέφει τον κωδικό του.

13. Int 21/0D

Κλήση: AH ← 0Dh
Επιστροφή: -
Σχόλια: αρχικοποιεί τον δίσκο γράφοντας τα περιεχόμενα των disk-buffers στα ανάλογα αρχεία. Δεν ενημερώνει τον κατάλογο του δίσκου (directory) και δεν πρέπει να χρησιμοποιείται στην θέση κάποιας ρουτίνας που κλείνει τα αρχεία.

14. Int 21/0E

Κλήση:
AH ← 0Eh
DL ← αριθμός οδηγού (A:=0, ..., Z:=25)
Επιστροφή:
AL ← αριθμός τελευταίου οδηγού
(A:=1, ..., Z:=26)
Σχόλια: θέτει τον εξ' ορισμού οδηγό (default drive) και επιστρέφει τον αριθμό των λογικά εγκατεστημένων οδηγών.

15. Int 21/25

Κλήση:
AH ← 25h
AL ← αριθμός διακοπής

DS:DX ← δείκτης σε νέα ρουτίνα
εξυπηρέτησης διακοπής

Επιστροφή: -

Σχόλια: η ρουτίνα αυτή εγγυάται την ασφαλή ενημέρωση του πίνακα ανυσμάτων διακοπής σε μια διεύθυνση που δίνετε εσείς. Αυτή είναι και η μόνη εγκεκριμένη μέθοδος για αλλαγή ενός διανύσματος διακοπής.

16. Int 21/26

Κλήση:
AH ← 26h
DX ← διεύθυνση τμήματος (segment) για το νέο PSP

Επιστροφή: -

Σχόλια: αντιγράφει το PSP του τρέχοντος προγράμματος στην οριζόμενη διεύθυνση τμήματος και ενημερώνει κατάλληλα την πληροφορία για τη δέσμευση μνήμης (updates memory allocation information).

17. Int 21/2A

(Get date)
Κλήση: AH ← 2Ah
Επιστροφή:
CX ← έτος (1980 - 2099)
DH ← μήνας (1 - 12)
DL ← ημέρα (1 - 31)
AL ← ημέρα της εβδομάδας
(0 = Κυριακή, ..., 6= Σάββατο)

Σχόλια: επιστρέφει την ημερομηνία του συστήματος που βασίζεται στο εσωτερικό ρολόι του DOS.

18. Int 21/2B

(Set date)
Κλήση:
AH ← 2Bh
CX ← έτος (1980 - 2099)
DH ← μήνας (1 - 12)
DL ← ημέρα (1 - 31)

Επιστροφή:
AL ← 00h ⇒ επιτυχής διόρθωση ημερομηνίας
AL ← FFh ⇒ αποτυχία λόγω λανθασμένης ημερομηνίας (π.χ. λανθασμένο format)

Σχόλια: τυπικά διορθώνει μόνο το τμήμα της ημερομηνίας του εσωτερικού ρολογιού του DOS αν όμως ο H/Y σας διαθέτει και CMOS ρολόι τότε διορθώνεται και η ημερομηνία και σε αυτό.

19. Int 21/2C

(Get time)
Κλήση: AH ← 2Ch
Επιστροφή:
CH ← ώρα (0 - 23)
CL ← λεπτά (0 - 59)
DH ← δευτερόλεπτα (0 - 59)
DL ← εκατοστά δευτερολέπτου (0 - 99)

Σχόλια: επιστρέφει την εσωτερική ώρα του DOS η οποία είναι τόσο ακριβής όσο και η αρχική της ρύθμιση.

20. Int 21/2D

(Set time)
Κλήση:
AH ← 2Dh
CH ← ώρα (0 - 23)
CL ← λεπτά (0 - 59)
DH ← δευτερόλεπτα (0 - 59)
DL ← εκατοστά δευτερολέπτου(0 - 99)

Επιστροφή:
AL ← 00h ⇒ επιτυχής διόρθωση ώρας
AL ← FFh ⇒ αποτυχία λόγω λάθους ώρας

Σχόλια: τυπικά διορθώνει μόνο το τμήμα το εσωτερικό ρολόι του DOS αν όμως ο H/Y σας

διαθέτει και CMOS ρολόι τότε διορθώνεται και η ώρα αυτού.

21. Int 21/2E

Κλήση:
AH ← 2Eh
AL ← 00h ⇒ ενεργοποίηση επαλήθευσης (verify on)
AL ← 01h ⇒ απενεργοποίηση επαλήθευσης (verify off)
DH ← 00h (για έκδοση του DOS προγενέστερη της 3.0)
Επιστροφή: -
Σχόλια: ενεργοποίηση της επαλήθευσης αυξάνει την ασφάλεια όταν γίνεται εγγραφή στον δίσκο, αυξάνει όμως και το χρόνο μεταφοράς δεδομένων στο δίσκο. (Δες και Int 21/54).

22. Int 21/30

(DOS version)
Κλήση: AH ← 30h
Επιστροφή:
AL ← κύριος αριθμός έκδοσης DOS (πχ. 2, 3, 4)
AH ← δευτερεύων αριθμός έκδοσης (πχ. Αν V2.10 τότε AH=10)
BX ← 00h
CX ← 00h
Σχόλια: επιστρέφει τον κύριο και δευτερεύοντα αριθμό έκδοσης του DOS κάτω από το οποίο τρέχει το πρόγραμμα σας. Εκδόσεις του DOS προγενέστερες της 2.0 επιστρέφουν στους AL και AH το 0.

23. Int 21/31

Κλήση:
AH ← 31h
AL ← κώδικας επιστροφής
DX ← μέγεθος μνήμης προς δέσμευση (σε παραγράφους)
Επιστροφή: -
Σχόλια: τερματίζει την εκτέλεση του προγράμματος αλλά δεν ελευθερώνει την μνήμη που είχε δεσμεύσει αυτό και ούτε κλείνει όσα αρχεία είχαν ανοιχθεί (terminate and stay resident). Η ρουτίνα αυτή επιτρέπει περισσότερα από 64K μνήμης καθώς και έλεγχο του κώδικα επιστροφής ο οποίος είναι προσιτός στην διεργασία πατέρα με την διακοπή Int 21/4D ή στα batch αρχεία μέσω της παραμέτρου ERRORLEVEL.

24. Int 21/35

Κλήση:
AH ← 35h
AL ← αριθμός διακοπής
Επιστροφή:
ES:BX ← δείκτης στην ρουτίνα εξυπηρέτησης διακοπής
Σχόλια: ο μόνος ορθός τρόπος για την εύρεση τις τρέχουσας τιμής του πίνακα διανυσμάτων διακοπών για την συγκεκριμένη διακοπή.

25. Int 21/36

(Get free space)
Κλήση:
AH ← 36h
DL ← αριθμός του οδηγού (0=default, 1=A:, κοκ)
Επιστροφή:
AX ← τομείς ανά συστοιχία, ή FFFFh αν δόθηκε λάθος αριθμός οδηγού
BX ← πλήθος διαθέσιμων συστοιχιώνs
CX ← bytes ανά τομέα
DX ← σύνολο συστοιχιών στο δίσκο

Σχόλια: επιστρέφει πληροφορίες σχετικές με τον ελεύθερο εναπομείναντα χώρο στον δίσκο.

26. Int 21/39

(MKDIR)
Κλήση:
AH ← 39h
DS:DX ← δείκτης σε ASCIIZ όνομα του path (μονοπατιού)
Επιστροφή:
CF ← 0 ⇒ επιτυχία
CF ← 1 ⇒ αποτυχία
AX ← 03h ⇒ δεν βρέθηκε το path
AX ← 05h ⇒ μη αποδεκτή προσπέλαση
Σχόλια: επιτρέπει την δημιουργία νέων καταλόγων (directories). Η ρουτίνα θα επιστρέψει λάθος και δεν θα δημιουργήσει τον κατάλογο αν είτε ο κατάλογος ήδη υπάρχει, είτε κάποιο μέρος του ονόματος του path δεν υπάρχει, είτε ο κατάλογος αρχίζει από την ρίζα (root) και αυτή είναι γεμάτη.

27. Int 21/3A

(RMDIR)
Κλήση:
AH ← 3Ah
DS:DX ← δείκτης σε ASCIIZ όνομα path
Επιστροφή:
CF ← 0 ⇒ επιτυχία
CF ← 1 ⇒ αποτυχία
AX ← 03h ⇒ δεν βρέθηκε το path
AX ← 05h ⇒ μη αποδεκτή προσπέλαση
AX ← 16h ⇒ τρέχων κατάλογος
Σχόλια: επιτρέπει το σβήσιμο ενός καταλόγου μόνο εάν αυτός υπάρχει, είναι άδειος και δεν είναι ο τρέχων (current).

28. Int 21/3B

(CHDIR)
Κλήση:
AH ← 3Bh
DS:DX ← δείκτης σε ASCIIZ όνομα path
Επιστροφή:
CF ← 0 ⇒ επιτυχία
CF ← 1 ⇒ αποτυχία
AX ← 03h ⇒ δεν βρέθηκε το path
Σχόλια: επιτρέχει την αλλαγή του τρέχοντος καταλόγου (παρόμοια με τις εντολές του DOS CD ή CHDIR).

29. Int 21/3C

(Create file)
Κλήση:
AH ← 3Ch
CX ← χαρακτηριστικά του αρχείου
DS:DX ← δείκτης σε ASCIIZ όνομα
Επιστροφή:
CF ← 0 ⇒ επιτυχία
AX ← χειριστής του αρχείου (file handle)
CF ← 1 ⇒ αποτυχία
AX ← 03h ⇒ δεν βρέθηκε το path
AX ← 04h ⇒ δεν υπάρχει διαθέσιμος χειριστής
AX ← 05h ⇒ μη αποδεκτή προσπέλαση
Σχόλια: δημιουργεί και ανοίγει ένα αρχείο αν αυτό δεν υπάρχει ήδη, αλλιώς το ξανανοίγει μηδενίζοντας το μήκος του. Στο όνομα του αρχείου επιτρέπεται και καθορισμός του path. Η ρουτίνα αυτή δεν μπορεί να χρησιμοποιηθεί για τη δημιουργία καταλόγων (subdirectories) ή volume labels.

30. Int 21/3D

(Open file)
Κλήση:
AH ← 3Dh
AL ← τρόπος προσπέλασης αρχείου (file access mode)
DS:DX ← δείκτης σε ASCIIZ όνομα

Επιστροφή:
CF ← 0 ⇒ επιτυχία
AX ← χειριστής του αρχείου (file handle)
CF ← 1 ⇒ αποτυχία
AX ← 01h ⇒ λανθασμένη ρουτίνα
AX ← 02h ⇒ δεν βρέθηκε το αρχείο
AX ← 03h ⇒ δεν βρέθηκε το path
AX ← 04h ⇒ δεν υπάρχουν διαθέσιμοι χειριστές
AX ← 05h ⇒ μη αποδεκτή προσπέλαση
AX ← 0Ch ⇒ λανθασμένος τρόπος προσπέλασης αρχείου

Σχόλια: για να ανοίξετε ένα αρχείο καθορίστε το όνομα του σε ASCIIZ μορφή. Το αρχείο μπορεί να είναι συνηθισμένο, hidden ή system. Ο παρακάτω πίνακας δείχνει πως θα διαλέξετε την τιμή του AL:

76543210	**Λειτουργία**
.....000	μόνο ανάγνωση
.....001	μόνο εγγραφή
.....010	ανάγνωση/εγγραφή

31. Int 21/3E

(Close file handle)
Κλήση:
AH ← 3Eh
BX ← χειριστής του αρχείου
Επιστροφή:
CF ← 0 ⇒ επιτυχία,
CF ← 1 ⇒ αποτυχία,
AX ← 06h ⇒ λανθασμένος χειριστής

Σχόλια: χρησιμοποιείται για να κλείσει αρχεία που ανοίχθηκαν ή δημιουργήθηκαν με τις ρουτίνες του DOS για τους χειριστές (handles, όχι τα FCBs). Ο χειριστής ελευθερώνεται για μετέπειτα χρήση και το αρχείο ενημερώνεται για τις οποιεσδήποτε τροποποιήσεις που έγιναν.

32. Int 21/3F

(Read from file)
Κλήση:
AH ← 3Fh
BX ← χειριστής του αρχείου
CX ← αριθμός bytes για ανάγνωση
DS:DX ← δείκτης στην περιοχή του buffer εγγραφής δεδομένων
Επιστροφή:
CF ← 0 ⇒ επιτυχία
AX ← αριθμός bytes που διαβάστηκαν
CF ← 1 ⇒ αποτυχία
AX ← 05h ⇒ μη αποδεκτή προσπέλαση
AX ← 06h ⇒ λάθος χειριστής

Σχόλια: μεταφέρει ένα συγκεκριμένο αριθμό από bytes από τον δίσκο στην περιοχή που δείχνουν οι DS:DX. Αν υπάρξει επιτυχία αλλά ο AX έχει τιμή μικρότερη από τον CX τότε έχει συμβεί μερικό διάβασμα αφού βρέθηκε το τέλος αρχείου (EOF). Αν το αρχείο βρίσκεται ήδη στο τέλος του κατά την κλήση το CF θα γίνει 1 αλλά ο AX θα μηδενιστεί.

33. Int 21/40

(Write to file)
Κλήση:
AH ← 40h
BX ← χειριστής του αρχείου
CX ← αριθμός bytes που θα γραφτούν
DS:DX ← δείκτης στην περιοχή του buffer δεδομένων
Επιστροφή:
CF ← 0 ⇒ επιτυχία
AX ← ο αριθμός των bytes που γράφτηκαν
CF ← 1 ⇒ αποτυχία
AX ← 05h ⇒ μη αποδεκτή προσπέλαση
AX ← 06h ⇒ λάθος χειριστής

Σχόλια: απλά ορίστε το χειριστή καθώς και τον αριθμό των bytes που θα γραφτούν και κάντε τους DS:DX να δείχνουν στο buffer των δεδομένων. Η ρουτίνα αυτή θα γράψει τα bytes στην τρέχουσα θέση του αρχείου.

34. Int 21/41

(Delete file)
Κλήση:
AH ← 41h
DS:DX ← δείκτης σε ASCIIZ όνομα
Επιστροφή:
CF ← 0 ⇒ επιτυχία
CF ← 1 ⇒ αποτυχία
AX ← 02h ⇒ δε βρέθηκε το αρχείο
AX ← 05h ⇒ μη αποδεκτή προσπέλαση
Σχόλια: σβήνει ένα αρχείο επιστρέφοντας τον χώρο που είχε δεσμευτεί από αυτό στο σύστημα. Στο όνομα αρχείου δεν επιτρέπονται wild-cards.

35. Int 21/42

Κλήση:
AH ← 42h
AL ← Κωδικός Μεθόδου
AL ← 00h, offset από την αρχή του αρχείου
AL ← 01h, offset από την τρέχουσα θέση του αρχείου
AL ← 02h, offset από το τέλος του αρχείου
BX ← χειριστής του αρχείου
CX:DX ← επιθυμούμενο offset
Επιστροφή:
CF ← 0 ⇒ επιτυχία
DX:DS ⇒ νέα θέση δείκτη αρχείου
CF ← 1 ⇒ αποτυχία
AX ← 01h ⇒ λανθασμένη ρουτίνα
AX ← 06h ⇒ λάθος χειριστής
Σχόλια: μετακινεί τον δείκτη ανάγνωσης/εγγραφής του αρχείου σε μια νέα θέση σε σχέση με είτε την αρχή ή τέλος του αρχείου είτε με την τρέχουσα θέση του.

36. Int 21/43

Κλήση:
AH ← 43h
AL ← 00h ⇒ εύρεση χαρακτηριστικών αρχείου
AL ← 01h ⇒ αλλαγή χαρακτηριστικών αρχείου, όπου
CX ← νέα χαρακτηριστικά αρχείου
DS:DX ← δείκτης σε ASCIIZ όνομα
Επιστροφή:
CF ← 0 ⇒ επιτυχία
CX ← τα χαρακτηριστικά αν AL ← 01h
CF ← 1 ⇒ αποτυχία
AX ← 01h ⇒ λάθος ρουτίνα
AX ← 02h ⇒ δεν βρέθηκε το αρχείο
AX ← 03h ⇒ δεν βρέθηκε το path
AX ← 05h ⇒ μη αποδεκτή προσπέλαση

Σχόλια: επιτρέπει ανάλογα με την τιμή του AL την εύρεση ή την αλλαγή των χαρακτηριστικών ενός αρχείου. Μόνο τα ακόλουθα χαρακτηριστικά είναι επιτρεπτά:

6543210	**Λειτουργία**
.....1	μόνο ανάγνωση
.....1.	hidden
....1..	system
.1.....	archive

37. Int 21/47

Κλήση:
AH ← 47h
DL ← αριθμός του οδηγού (0 = τρέχων, A: = 1, κοκ)
DS:SI ← δείκτης 64-byte scratch buffer
Επιστροφή:
CF ← 0 ⇒ επιτυχία
DS:SI ← δείκτης στο path του τρέχοντα καταλόγου
CF ← 1 ⇒ αποτυχία
AX ← 0Fh ⇒ μη επιτρεπτός οδηγός
Σχόλια: επιστρέφει το όνομα του path του τρέχοντος καταλόγου (current directory) χωρίς τα χαρακτηριστικά του οδηγού ή τα backslashes (\). Αν ο κατάλογος είναι η ρίζα τότε επιστρέφεται ο χαρακτήρας NUL.

38. Int 21/4B

Κλήση:
AH ← 4Bh

AL ← 00h ⇒ φορτώνει και εκτελεί ένα πρόγραμμα
AL ← 03h ⇒ φορτώνει ένα overlay
ES:BX ← δείκτης σε block παραμέτρων
DS:DX ← δείκτης σε ASCIIZ όνομα

Επιστροφή:
CF ← 0 ⇒ επιτυχία, όλοι οι καταχωρητές πλην των CS, IP επηρεάζονται. Οι SS, SP θα πρέπει να σώζονται πριν την κλήση της ρουτίνας.
CF ← 1 ⇒ αποτυχία
AX ← 01h ⇒ λανθασμένη ρουτίνα
AX ← 02h ⇒ δεν βρέθηκε το αρχείο
AX ← 05h ⇒ μη αποδεκτή προσπέλαση
AX ← 08h ⇒ μή αρκετή μνήμη
AX ← 0Ah ⇒ λάθος περιβάλλον (invalid environment)
AX ← 0Bh ⇒ λανθασμένο format

Σχόλια: επιτρέπει την εκτέλεση προγραμμάτων και την διαχείριση των overlays. Όταν ένα καινούργιο πρόγραμμα (διεργασία παιδί) τερματίσει, τότε το αρχικό πρόγραμμα (διεργασία πατέρας) επανακτά τον έλεγχο. Η διεργασία πατέρας μπορεί να πάρει τον κωδικό εξόδου του παιδιού αν αυτό χρησιμοποιεί μια ρουτίνα τερματισμού η οποία επιστρέφει κωδικό τέλους. Ο έλεγχος της λειτουργίας γίνεται από το block παραμέτρων το οποίο έχει την παρακάτω μορφή:

1) AL ← 00h

Byte	Μήκος	Περιεχόμενα
00h	word	Τμήμα (segment) του block περιβάλλοντος
02h	dword	Δείκτης στην "ουρά διαταγής"
06h	dword	Δείκτης στο 1ο FCB (offset 05Ch)
0Ah	dword	Δείκτης στο 2ο FCB (offset 06Ch)

2) AL ← 03h

Byte	Μήκος	Περιεχόμενα
00h	word	Τμήμα (segment) του σημείου φόρτωσης (load point) για το overlay.
02h	word	Παράγοντας επανατοποθέτησης για τα EXE αρχεία μόνο (relocation factor)

Το block περιβάλλοντος είναι μερικές ASCIIZ συμβολοσειρές οι οποίες χρησιμοποιούνται για να περάσουν πληροφορίες για το περιβάλλον στο πρόγραμμα που εκτελείται. Η "ουρά διαταγής" είναι μια συμβολοσειρά η οποία αποτελείται από οτιδήποτε θα γράφαμε στην γραμμή διαταγής (command line) μετά την εντολή προς εκτέλεση. Στο πρώτο byte έχει το μήκος ενώ τερματίζεται με ένα carriage return.

39. Int 21/4C

(Terminate program)

Κλήση:
AH ← 4Ch
AL ← κώδικας επιστροφής

Επιστροφή: -

Σχόλια: ο ορθός τρόπος με τον οποίο τερματίζεται ένα πρόγραμμα και μετά επιστρέφει τον κώδικα που βρίσκεται στον AL. Η ρουτίνα αυτή θα πρέπει να χρησιμοποιείται αντί των Int 20 ή Int 21/00.

40. Int 21/4D

Κλήση: AH ← 4Dh

Επιστροφή:
AH ← κωδικός εξόδου (exit) του συστήματος:
00h ⇒ κανονικός τερματισμός
01h ⇒ τερματισμός με Ctrl-C
02h ⇒ τερματισμός λόγω λάθους συσκευής (device error)
03h ⇒ τερματισμός με κλήση της Int 21/31
AL ← κωδικός εξόδου της θυγατρικής διεργασίας.

Σχόλια: όταν κληθεί για μία και μόνο φορά δίνει τον κωδικό εξόδου του συστήματος και της διεργασίας παιδί. Ο κωδικός εξόδου του

συστήματος, πληροφορεί για τον ομαλό ή όχι τερματισμό του προγράμματος.

41. Int 21/54

Κλήση: AH ← 54h
Επιστροφή:
AL ← 00h ⇒ ενεργοποιημένη επαλήθευση (verify on)
AL ← 01h ⇒ απενεργοποιημένη επαλήθευση (verify off)
Σχόλια: επιστρέφει την τρέχουσα τιμή του flag της επαλήθευσης. Η Int 21/2E θέτει την επαλήθευση on/off.

42. Int 21/56

(Rename file)
Κλήση:
AH ← 56h
DS:DX ← δείκτης σε ASCIIZ όνομα αρχείου
ES:DI ← δείκτης στο νέο όνομα (ASCIIZ)
Επιστροφή:
CF ← 0 ⇒ επιτυχία
CF ← 1 ⇒ αποτυχία
AX ← 02h ⇒ δεν βρέθηκε το αρχείο
AX ← 03h ⇒ δεν βρέθηκε το path
AX ← 05h ⇒ μη επιτρεπτή προσπέλαση
AX ← 11h ⇒ διαφορετική συσκευή
Σχόλια: επιτρέπει την αλλαγή ονόματος κάποιου αρχείου. Δεν επιτρέπονται τα wildcards αλλά υποστηρίζεται το πλήρες όνομα του path. Μην προσπαθήσετε να μετονομάσετε ανοικτά αρχεία γιατί αυτό μπορεί να έχει απρόβλεπτα αποτελέσματα.

43. Int 21/57

Κλήση:
AH ← 57h
AL ← 00h ⇒ εύρεση ημερομηνίας και ώρας
BX ← χειριστής του αρχείου
AL ← 01h ⇒ αλλαγή ημερομηνίας και ώρας αρχείου
BX ← χειριστής του αρχείου
CX ← ώρα
DX ← ημερομηνία
Επιστροφή:
CF ← 0 ⇒ επιτυχία
CX ← ώρα αν AL ← 00h
DX ← ημερομηνία αν AL ← 00h
CF ← 1 ⇒ αποτυχία
AX ← 01h ⇒ λανθασμένη ρουτίνα
AX ← 06h ⇒ λάθος χειριστής
Σχόλια: βρίσκει ή διορθώνει την ώρα και ημερομηνία μέσω χειριστών (handles). Η μορφή με την οποία ορίζονται η ημερομηνία και η ώρα είναι η εξής:

1) Ώρα

FEDCBA98	76543210	**Λειτουργία**
xxxxx...	.....	ώρες (0 - 23)
.....xxx	xxx.....	λεπτά (0 - 59)
.....	...xxxxx	ανά 2“ (0 - 29)

2) Ημερομηνία

FEDCBA98	76543210	**Λειτουργία**
xxxxxxx.	.....	έτος-1980
.....x	xxx.....	μήνας (1 - 12)
.....	...xxxxx	ημέρα (1 - 31)

34. Int 27

(TSR)
Κλήση:
DX ← offset του τελευταίου byte συν ένα, (σε σχέση με το PSP) του προγράμματος που θα μείνει memory resident.
CS ← Τμήμα (segment) του PSP
Επιστροφή: -
Σχόλια: επιτρέπει σ' ένα πρόγραμμα μεγέθους μέχρι 64K να παραμείνει resident μετά τον τερματισμό του. Χρησιμοποιείται συχνά για την εγκατάσταση ρουτινών εξυπηρέτησης διακοπής διαφόρων συσκευών (device-specific Interrupt handlers). Συνιστάται η χρήση της ρουτίνας Int 21/31 που επιτρέπει σε προγράμματα μεγαλύτερα των 64K να παραμείνουν στη μνήμη (resident) εκτός αν απαιτείται συμβατότητα με εκδόσεις του DOS παλαιότερες της 2.0.

Παράρτημα 6ο
Πίνακας Χαρακτήρων ASCII και Επεκτεταμένων Κωδικών Πληκτρολογίου

Πίνακας Π6.1 Χαρακτήρες ASCII

Dec	Hex	Name	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
00	00	**NUL**	32	20	**Space**	64	40	**@**	96	60	**`**
01	01	**SOH**	33	21	**!**	65	41	**A**	97	61	**a**
02	02	**STX**	34	22	**“**	66	42	**B**	98	62	**b**
03	03	**ETX**	35	23	**#**	67	43	**C**	99	63	**c**
04	04	**EOT**	36	24	**$**	68	44	**D**	100	64	**d**
05	05	**ENQ**	37	25	**%**	69	45	**E**	101	65	**e**
06	06	**ACK**	38	26	**&**	70	46	**F**	102	66	**f**
07	07	**BEL**	39	27	**‘**	71	47	**G**	103	67	**g**
08	08	**BS**	40	28	**(**	72	48	**H**	104	68	**h**
09	09	**HT**	41	29	**)**	73	49	**I**	105	69	**i**
10	A	**LF**	42	2A	*	74	4A	**J**	106	6A	**j**
11	B	**VT**	43	2B	+	75	4B	**K**	107	6B	**k**
12	C	**FF**	44	2C	**,**	76	4C	**L**	108	6C	**l**
13	D	**CR**	45	2D	**-**	77	4D	**M**	109	6D	**m**
14	E	**SO**	46	2E	**.**	78	4E	**N**	110	6E	**n**
15	F	**SI**	47	2F	/	79	4F	**O**	111	6F	**o**
16	10	**DLE**	48	30	**0**	80	50	**P**	112	70	**p**
17	11	**DC1**	49	31	**1**	81	51	**Q**	113	71	**q**
18	12	**DC2**	50	32	**2**	82	52	**R**	114	72	**r**
19	13	**DC3**	51	33	**3**	83	53	**S**	115	73	**s**
20	14	**DC4**	52	34	**4**	84	54	**T**	116	74	**t**
21	15	**NAK**	53	35	**5**	85	55	**U**	117	75	**u**
22	16	**SYN**	54	36	**6**	86	56	**V**	118	76	**v**
23	17	**ETB**	55	37	**7**	87	57	**W**	119	77	**w**
24	18	**CAN**	56	38	**8**	88	58	**X**	120	78	**x**
25	19	**EM**	57	39	**9**	89	59	**Y**	121	79	**y**
26	1A	**SUB**	58	3A	**:**	90	5A	**Z**	122	7A	**z**
27	1B	**ESC**	59	3B	**;**	91	5B	**[**	123	7B	**{**
28	1C	**FS**	60	3C	<	92	5C	\	124	7C	**|**
29	1D	**GS**	61	3D	=	93	5D	**]**	125	7D	**}**
30	1E	**RS**	62	3E	>	94	5E	^	126	7E	~
31	1F	**US**	63	3F	**?**	95	5F	**_**	127	7F	**del**

Πίνακας Π6.2 Η σημασία των χαρακτήρων ελέγχου

SOH - Start Of Header	HT - Horizontal Tabulation	DC1 - Device Control 1	EM - End of Medium
STX - Start Of teXt	LF - Line Feed	DC2 - Device Control 2	SUB - SUBstitute
ETX - End Of teXt	VT - Vertical Tabulation	DC3 - Device Control 3	ESC - ESCape
EOT - End Of Transmission	FF - Form Feed	DC4 - Device Control 4	FS - File Separator
ENQ - ENQuiry	CR - Carriage Return	NAK - Negative AcKnowledge	GS - MainForm.Group Separator
ACK - ACKnowledge	SO - Shift Out	SYN - SYNchronous idle	RS - Record Separator
BEL - BELl	SI - Shift In	ETB - End of Transmission Block	US - Unit Separator
BS - BackSpace	DLE - Data Link Escape	CAN - CANcel	

Dec	Hex	Name	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	A	160	A0	ι	192	C0	└	224	E0	ω
129	81	B	161	A1	κ	193	C1	┴	225	E1	ά
130	82	Γ	162	A2	λ	194	C2	┬	226	E2	έ
131	83	Δ	163	A3	μ	195	C3	├	227	E3	ή
132	84	E	164	A4	ν	196	C4	─	228	E4	ϊ
133	85	Z	165	A5	ξ	197	C5	┼	229	E5	ί
134	86	H	166	A6	ο	198	C6	╞	230	E6	ό
135	87	Θ	167	A7	π	199	C7	╟	231	E7	ύ
136	88	I	168	A8	ρ	200	C8	╚	232	E8	ϋ
137	89	K	169	A9	σ	201	C9	╔	233	E9	ώ
138	8A	Λ	170	AA	ς	202	CA	╩	234	EA	Ά
139	8B	M	171	AB	τ	203	CB	╦	235	EB	Έ
140	8C	N	172	AC	υ	204	CC	╠	236	EC	Ή
141	8D	Ξ	173	AD	φ	205	CD	═	237	ED	Ί
142	8E	O	174	AE	χ	206	CE	╬	238	EE	Ό
143	8F	Π	175	AF	ψ	207	CF	╧	239	EF	Ύ
144	90	P	176	B0	░	208	D0	╨	240	F0	'Ω
145	91	Σ	177	B1	▒	209	D1	╤	241	F1	±
146	92	T	178	B2	▓	210	D2	╥	242	F2	≥
147	93	Y	179	B3	│	211	D3	╙	243	F3	≤
148	94	Φ	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	X	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	Ψ	182	B6	╢	214	16	╓	246	F6	÷
151	97	Ω	183	B7	╖	215	D7	╫	247	F7	≈
152	98	α	184	B8	╕	216	D8	╪	248	F8	°
153	99	β	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	γ	186	BA	║	218	DA	┌	250	FA	·
155	9B	δ	187	BB	╗	219	DB	█	251	FB	√
156	9C	ε	188	BC	╝	220	DC	▄	252	FC	^{n}
157	9D	ζ	189	BD	╜	221	DD	▌	253	FD	^{2}
158	9E	η	190	BE	╛	222	DE	▐	254	FE	■
159	9F	θ	191	BF	┐	223	DF	▀	255	FF	(Blank)

Πίνακας Επεκτεταμένων Κωδικών Πληκτρολογίου (int 16/00 και int 16/02)

Πλήκτρο	Κωδικός	Πλήκτρο	Κωδικός	Πλήκτρο	Κωδικός	Πλήκτρο	Κωδικός
F1	3Bh	F10	44h	Shift-F7	5Ah	Ctrl-F4	61h
F2	3Ch	F11	85h	Shift-F8	5Bh	Ctrl-F5	62h
F3	3Dh	F12	86h	Shift-F9	5Ch	Ctrl-F6	63h
F4	3Eh	Shift-F1	54h	Shift-F10	5Dh	Ctrl-F7	64h
F5	3Fh	Shift-F2	55h	Shift-F11	87h	Ctrl-F8	65h
F6	40h	Shift-F3	56h	Shift-F12	88h	Ctrl-F9	66h
F7	41h	Shift-F4	57h	Ctrl-F1	5Eh	Ctrl-F10	67h
F8	42h	Shift-F5	58h	Ctrl-F2	5Fh	Ctrl-F11	89h
F9	43h	Shift-F6	59h	Ctrl-F3	60h	Ctrl-F12	8Ah
Alt-F1	68h	Alt-F7	6Eh	Alt-1	78h	Alt-7	7Eh
Alt-F2	69h	Alt-F8	6Fh	Alt-2	79h	Alt-8	7Fh
Alt-F3	6Ah	Alt-F9	70h	Alt-3	7Ah	Alt-9	80h
Alt-F4	6Bh	Alt-F10	71h	Alt-4	7Bh	Alt-0	81h
Alt-F5	6Ch	Alt-F11	8Bh	Alt-5	7Ch	Alt-πλην	82h
Alt-F6	6Dh	Alt-F12	8Ch	Alt-6	7Dh	Alt-=	83h