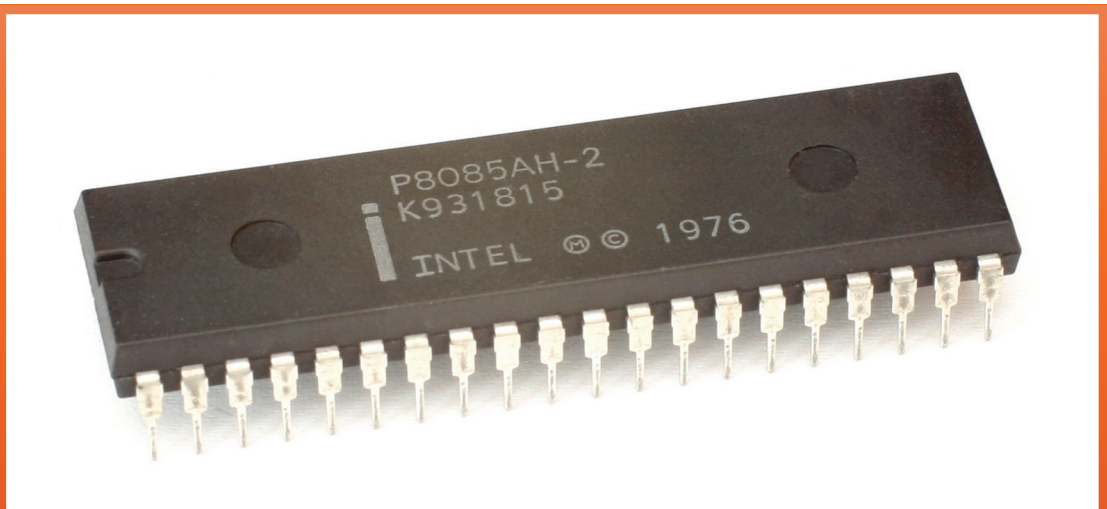
Τοπάλης Ευάγγελος (Δρ. Ηλεκτρολόγος Μηχανικός & Τεχνολογίας Υπολογιστών)
καθ. Χαδέλλης Λουκάς

Μικροϋπολογιστικά Συστήματα

Εργαστηριακές Ασκήσεις Μικροεπεξεργαστής Intel 8085

2024



ISBN 978-618-00-5292-3

ISBN 978-618-00-5292-3

Περιεχόμενα

Κεφάλαιο 1 – Εργαστηριακές Ασκήσεις

ΑΣΚΗΣΗ 1 – Αποθήκευση σε Μνήμη και Μεταφορά Περιεχομένων Μνήμης, Αριθμητικές πράξεις (Πρόσθεση δεκαεξαδικών 8-bit (διψήφιων) αριθμών)

Παράδειγμα 1.1

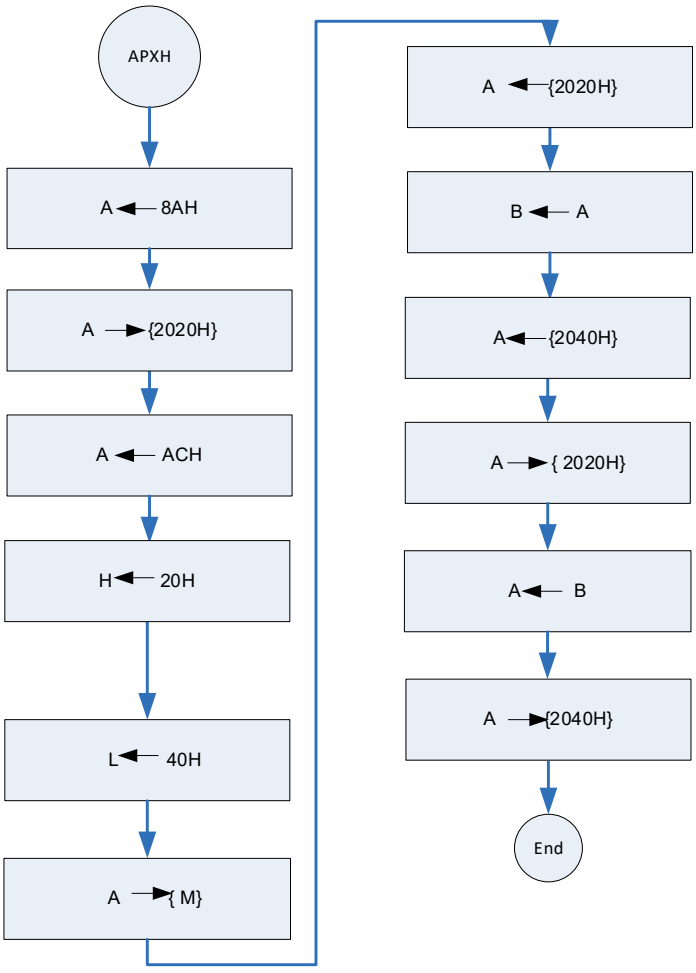
Χρησιμοποιώντας τον απευθείας τρόπο μεταφοράς δεδομένων από/προς την μνήμη (immediate addressing mode), γράψτε ένα πρόγραμμα για να φορτώσετε άμεσα τον αριθμό 8AH στην θέση μνήμης 2020H.

Ύστερα, φορτώστε τον αριθμό ACH στον καταχωρητή A και στην συνέχεια χρησιμοποιώντας τον τρόπο μεταφοράς δεδομένων από/προς την μνήμη με χρήση των καταχωρητών διεύθυνσης (καταχωρητές H και L) να τον μεταφέρετε στην θέση μνήμης 2040H.

Στην συνέχεια, με χρήση των καταχωρητών διεύθυνσης (καταχωρητές H και L), ανταλλάξτε αμοιβαία τα περιεχόμενα των θέσεων μνήμης 2020H και 2040H, δηλαδή ότι είχε η θέση μνήμης 2020 να μεταφερθεί στην θέση μνήμης 2040H και ότι είχε η 2040H να μεταφερθεί στην 2020H.

Τρέξτε το πρόγραμμα και διαπιστεύστε την ορθή μεταφορά όλων των δεδομένων (μνήμης και καταχωρητών).

Στην συνέχεια δίνεται το διάγραμμα ροής του παραδείγματος



Ακολουθεί ο κώδικα για το gnusim8085.

```
;*************************************************************************
********
;LABORATORY EXERCISE 1.1
;FILE LAB1_1.ASM
;*************************************************************************
********
MVI A,8AH   ;μεταφορά του 8AH στον καταχωρητή A
STA 2020H   ;αποθήκευση των περιεχομένων του A στην θέση μνήμης 2020H
MVI A,0ACH  ;μεταφορά του ACH στον καταχωρητή A
MVI H,20H   ;μεταφορά του 20H στον καταχωρητή H
MVI L,40H   ;μεταφορά του 40H στον καταχωρητή L
        ;εναλλακτικός τρόπος φόρτωσης σε ζευγάρι καταχωρητών-LXI H, 2040H
MOV M,A ;αποθήκευση των περιεχομένων του A στην θέση μνήμης
        ;που ορίζεται από HL
LDA 2020H   ;φόρτωση των περιεχομένων της θέσης μνήμης 2020H στον A
MOV B,A ;αντιγραφή των περιεχομένων του καταχωρητή A
        ;στον καταχωρητή B για να μην χαθούν από την
        ;επόμενη εντολη [ΤΑ ΠΕΡΙΕΧΟΜΕΝΑ ΤΟΥ ΚΑΤΑΧΩΡΗΤΗΣ Β ΕΙΝΑΙ ΙΔΙΑ ΜΕ
        ;ΑΥΤΑ ΤΗΣ ΘΕΣΗΣ ΜΝΗΜΗΣ 2020H]
LDA 2040H   ;φόρτωση των περιεχομένων της θέσης μνήμης 2040H
        ;στον A [ΤΑ ΠΕΡΙΕΧΟΜΕΝΑ ΤΟΥ ΚΑΤΑΧΩΡΗΤΗ Α ΜΕΤΑ ΑΠΟ ΑΥΤΗ
        ;ΤΗΝ ΕΝΤΟΛΗ ΕΙΝΑΙ ΙΔΙΑ ΜΕ ΑΥΤΑ ΤΗΣ ΘΕΣΗΣ ΜΝΗΜΗΣ 2040H]
STA 2020H   ;αποθήκευση των περιεχομένων του A στην θέση μνήμης 2020H
MOV A,B ;αντιγραφή των περιεχομένων του καταχωρητή B στον καταχωρητή A
STA 2040H   ;αποθήκευση των περιεχομένων του A στην θέση μνήμης 2040H
HLT
```

Παράδειγμα 1.2

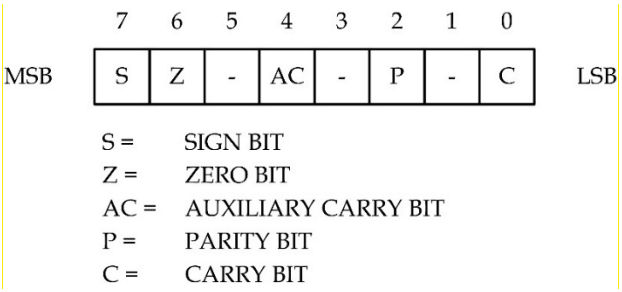
Παράδειγμα 1 – Πρόσθεση 8bit αριθμών

Γράψτε ένα πρόγραμμα για να προσθέσετε τους αριθμούς 3AH & 08H. Τοποθετήστε το άθροισμά τους στην θέση μνήμης 2030H. Έπειτα προσθέστε τα περιεχόμενα των θέσεων μνήμης 2030H και της 2031H και τοποθετείστε το αποτέλεσμα στη θέση μνήμης 2032H.

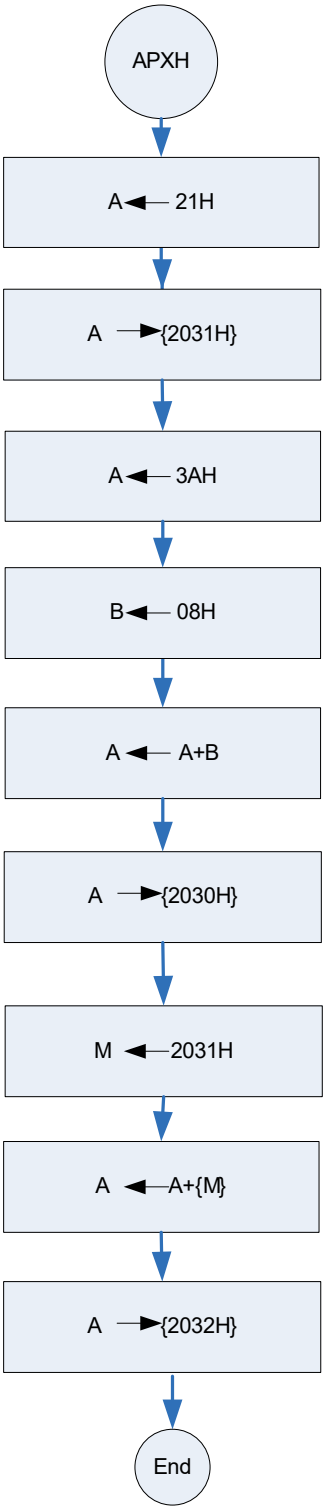
Αρχικοποιήστε τα περιεχόμενα της 2031H με 21H είτε βάζοντας το μέσω εντολών ή μέσω του προγράμματος GNUSim8085.

Υπολογίστε τα αθροίσματα μόνοι σας και επιβεβαιώστε την σωστή λειτουργία του προγράμματος

Η σημασία των bits του καταχωρητή F (FLAGS) είναι η εξής:



ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ

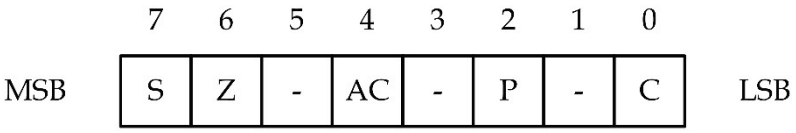


Ακολουθεί ο κώδικας για το gnusim8085

```
;*************************************************************************
********
;LABORATORY EXERCISE 1.2
;FILE LAB1_2.ASM
;*************************************************************************
********
MVI A,21H   ;μεταφορά του 21H στον καταχωρητή A
STA 2031H   ;αποθήκευση του περιεχομένου του A στην θέση μνήμης 2030H
MVI A,3AH   ;μεταφορά του 3AH στον καταχωρητή A
MVI B,08H   ;μεταφορά του 08H στον καταχωρητή B
ADD B       ;A = A + B
STA 2030H   ; αποθήκευση του περιεχομένου του A στην θέση μνήμης 2030H
LXI H, 2031H    ; φόρτωση του M με το 2031H
ADD M       ;A = A + {M}
STA 2032H   ; αποθήκευση του περιεχομένου του A στην θέση μνήμης 2030H
HLT
```

ΑΣΚΗΣΗ 2 –Αριθμητικές πράξεις (Πρόσθεση δεκαεξαδικών - 16-bit (τετραψήφιων) αριθμών), αφαίρεση και λογικές πράξεις

Η σημασία των bits του καταχωρητή F (FLAGS) είναι η εξής:



S = SIGN BIT
Z = ZERO BIT
AC = AUXILIARY CARRY BIT
P = PARITY BIT
C = CARRY BIT

Παράδειγμα 2.1 – Πρόσθεση 16bit αριθμών και αφαίρεση

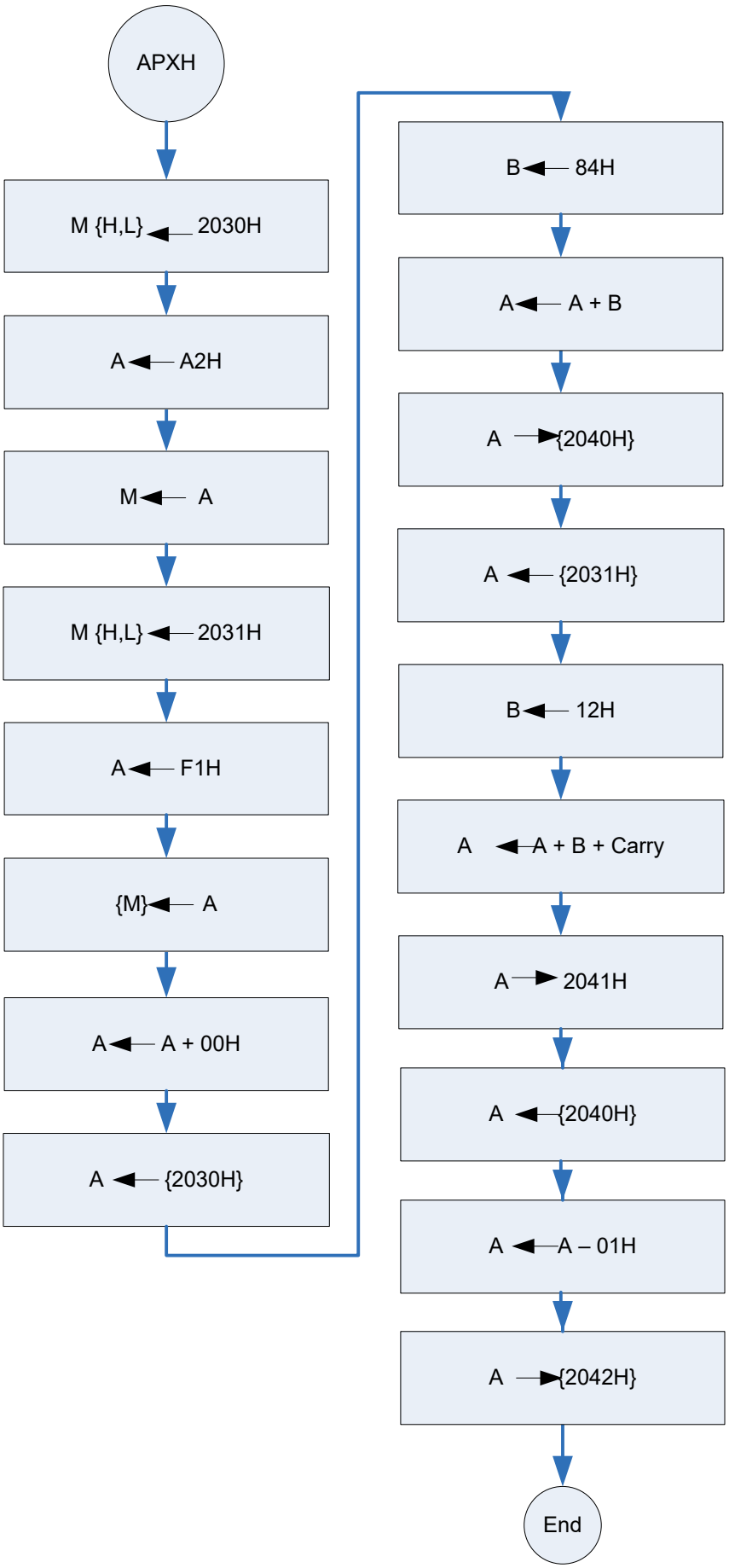
Γράψτε ένα πρόγραμμα για να αποθηκεύσετε τον 16-bit αριθμό F1A2H σπάζοντάς τον στις θέσεις μνήμης 2030H, 2031H (δηλαδή ο A2H στην θέση μνήμης 2030H και ο F1H στην θέση μνήμης 2031H).

Στη συνέχεια να προσθέσετε στον αριθμό που αποθηκεύσατε τον 16-bit αριθμό 1284H και τοποθετήστε το αποτέλεσμα στις θέσεις μνήμης 2040H, 2041H. Θα γίνει η πράξη της πρόσθεσης μεταξύ των 8 λιγότερων σημαντικών bit του κάθε 16-bit αριθμού και μεταξύ των 8 περισσότερων σημαντικών bit του κάθε 16-bit αριθμού (δηλαδή A2H + 84H και το αποτέλεσμα στην 2040H και F1H + 12H και το αποτέλεσμα στην 2041H).

Τέλος, από το περιεχόμενο της θέσης μνήμης 2040H να αφαιρέσετε τον αριθμό 01H και να αποθηκεύσετε το αποτέλεσμα στη θέση μνήμης 2042H.

Υπολογίστε τα αποτελέσματα μόνοι σας και δείτε στον flag register αν δημιουργήθηκε κρατούμενο - δανεικό. Αιτιολογείστε την απάντηση σας.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ

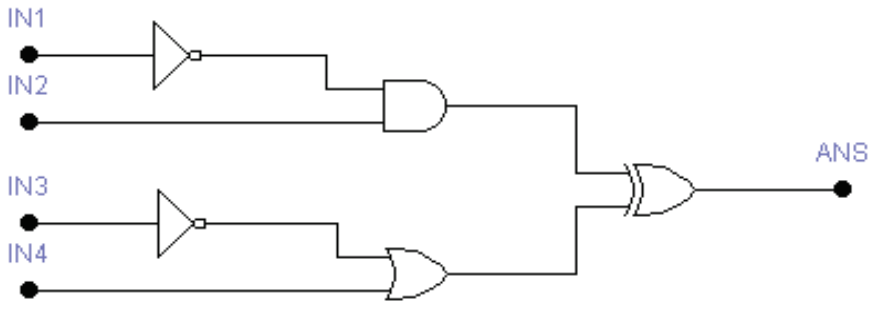

APXH
M {H,L} ← 2030H
A ← A2H
M ← A
M {H,L} ← 2031H
A ← F1H
{M} ← A
A ← A + 00H
A ← {2030H}
B ← 84H
A ← A + B
A → {2040H}
A ← {2031H}
B ← 12H
A ← A + B + Carry
A → 2041H
A ← {2040H}
A ← A – 01H
A → {2042H}
End

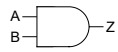
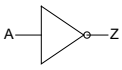
Ακολουθεί ο κώδικας στο gnusim8085

```
;*************************************************************************
********
;LABORATORY EXERCISE 2.1
;FILE LAB2_1.ASM
;*************************************************************************
********
LXI H,2030H ;ορισμός του M (HL) ως δείκτης στην 2030H
MVI A,0A2H  ;μεταφορά του A2H στον καταχωρητή A
MOV M,A ;μεταφορά του A στην θέση μνήμης που δείχνει ο H-L
LXI H,2031H ;ορισμός του M (HL) ως δείκτης στην 2031H
MVI A,0F1H  ;μεταφορά του F1H στον καταχωρητή A
MOV M,A ;μεταφορά του A στην θέση μνήμης που δείχνει ο H-L
ADI 00H     ;μηδενισμός του flag carry  (η άμεση πρόσθεση με το 00H
             ;μηδενίζει το Carry), Γιατί δεν είναι απαραίτητη να μπει?????
LDA 2030H   ;φόρτωση των περιεχομένων της θέσης μνήμης 2030H στον A
MVI B,84H   ;μεταφορά του 84H στον καταχωρητή B
ADD B       ;προσθεση των περιεχομένων του A και του B και
             ;αποθήκευση του αποτελέσματος στο A  (A = A + B)
STA 2040H   ;αποθήκευση του A (αποτελέσματος) στην θέση μνήμης 2040H
LDA 2031H   ;φόρτωση των περιεχομένων της θέσης μνήμης 2031H στον A
MVI B,12H   ;μεταφορά του 12H στον καταχωρητή B
ADC B       ;A = A + B + C
STA 2041H   ;αποθήκευση του A (αποτελέσματος) στην θέση μνήμης 2041H
LDA 2040H   ;φόρτωση των περιεχομένων της θέσης μνήμης 2040H στον A
SUI 01H     ;αφαιρεση από τον A του αριθμού 3AH και αποθήκευση του
             ;αποτελέσματος στον A (A=A-3AH)
STA 2042H   ;αποθήκευση του A (αποτελέσματος) στην θέση μνήμης 2042H
HLT
```

Παράδειγμα 2.2 – Λογικές Πράξεις

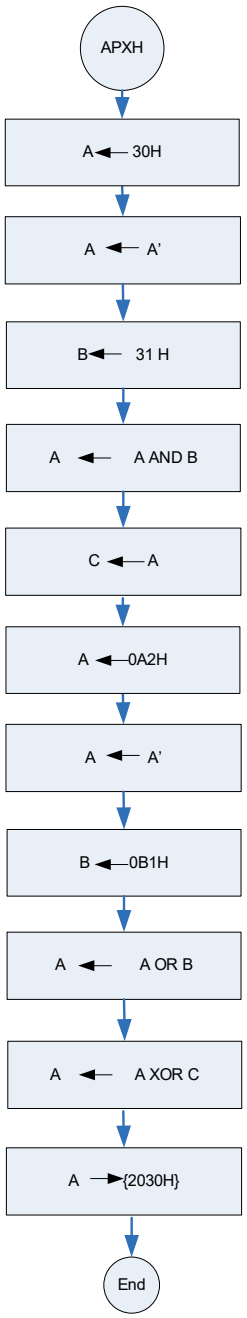
Γράψτε ένα πρόγραμμα για να υπολογίζει το αποτέλεσμα του παρακάτω λογικού κυκλώματος και να τοποθετεί το αποτέλεσμα στην 2030H. Θεωρείστε αρχικές τιμές στα IN1=30H, IN2=31H, IN3=0A2H, IN4=0B1H.



ΟΝΟΜΑΣΙΑ	ΣΥΜΒΟΛΟ ΠΥΛΗΣ	ΣΧΕΣΗ ΑΛΓΕΒΡΑΣ BOOLE	ΠΙΝΑΚΑΣ ΑΛΗΘΕΙΑΣ
AND		**Z=A•B**	A B | Z 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		**Z=A+B**	A B | Z 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
XOR		**Z=A ⊕ B**	A B | Z 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
NOT		**Z=A'= $\overline{A}$**	A | Z 0 | 1 1 | 0

Στην συνέχεια δίνεται το διάγραμμα ροής του παραδείγματος

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ



Ακολουθεί ο κώδικας στο gnusim8085

```
;*************************************************************************
********
;LABORATORY EXERCISE 2.2
;FILE LAB2_2.ASM
;*************************************************************************
********
MVI A,30H  ;μεταφορά του 30H στον καταχωρητή A
CMA     ;συμπλήρωμα καταχωρητή A
MVI B,31H  ;μεταφορά του 31H στον καταχωρητή B
ANA B  ;A AND B
MOV C,A ;μεταφορά του καταχωρητή A στον καταχωρητή C
MVI A,0A2H  ;μεταφορά του 0A2H στον καταχωρητή A
CMA     ;συμπλήρωμα καταχωρητή A
MVI B,0B1H  ;μεταφορά του 0B1H στον καταχωρητή B
ORA B   ;A OR B
XRA C ; A XOR C
STA 2030H  ;αποθήκευση του A (αποτελέσματος) στην θέση μνήμης 2030H
HLT
```

ΑΣΚΗΣΗ 3 – Βρόχοι, Συγκρίσεις, Διακλαδώσεις

Οι τρόποι σύγκρισης δύο αριθμητικών ποσοτήτων και οι διακλαδώσεις ή βρόγχοι που, συνήθως, εκτελούνται ανάλογα με το αποτέλεσμα της σύγκρισης αποτελούν σημαντικό κομμάτι των υποστηριζόμενων προγραμματιστικών τεχνικών και πολλές φορές προκαλούν σύγχυση μέχρι να κατανοηθούν πλήρως.

Έτσι λοιπόν το **πρώτο βήμα** είναι να κατανοήσουμε τους τρόπους (εντολές με άλλα λόγια) με τους οποίους μπορούμε να συγκρίνουμε δύο ποσότητες. Δύο βασικά χαρακτηριστικά που διέπουν όλες τις εντολές συγκρίσεις είναι ότι

1) το πρώτο όρισμα είναι πάντα ο accumulator και

2) η σύγκριση είναι ουσιαστικά μια αφαίρεση του δεύτερου ορίσματος από το πρώτο (που πάντα είναι ο accumulator) το αποτέλεσμα της οποίας **ΔΕΝ** αποθηκεύεται πουθενά.

Αυτές οι εντολές συνοπτικά είναι (για περισσότερες λεπτομέρειες μπορείτε να ανατρέξετε στο Instruction Set που παρατίθεται στο τέλος του φυλλαδίου):

CMP r: Το περιεχόμενο του καταχωρητή **r** συγκρίνεται με αυτό του **A** με το να αφαιρείται από τον **A**.

CMP M: Το περιεχόμενο της θέσης μνήμης **M** συγκρίνεται με αυτό του **A** με το να αφαιρείται από τον **A**.

CPI byte: Η απόλυτη αριθμητική ποσότητα **byte** συγκρίνεται με το περιεχόμενο του **A** με το να αφαιρείται από τον **A**.

Το **δεύτερο βήμα** είναι να κατανοήσουμε πως ή που αποτυπώνεται το αποτέλεσμα της σύγκρισης (ουσιαστικά της αφαίρεσης) αφού το αποτέλεσμα αυτό καθαυτό δεν φυλάσσεται πουθενά. Η απάντηση είναι ότι κάθε πιθανό αποτέλεσμα αποτυπώνεται και μπορεί να εξαχθεί από συνδυαστική εξέταση των σημαιών Z και CY σύμφωνα με της εξής λογική.

Έστω ότι συγκρίνουμε το περιεχόμενο του A με αυτού του B μέσω της εντολής CMP B, ισχύουν τα εξής.

Αποτέλεσμα Σύγκρισης	Τιμή του bit ZERO	Τιμή του bit CARRY
A=B	Z=1	CY=0
A<B	Z=0	CY=1
A>B	Z=0	CY=0

Το **τρίτο βήμα** είναι πως μπορούμε να χρησιμοποιήσουμε τα ZERO και CARRY bit για να διακλαδωθεί η εκτέλεση του προγράμματος ανάλογα με το αποτέλεσμα της σύγκρισης. Η απάντηση είναι μέσω των εντολών JMP που λαμβάνουν υπόψη τη τιμή των Z και CY bit. Αυτές είναι:

JNZ: θα εκτελέσει την εντολή JMP αν από την προηγούμενη σύγκριση το Z=0

JZ: θα εκτελέσει την εντολή JMP αν από την προηγούμενη σύγκριση το Z=1

JNC: θα εκτελέσει την εντολή JMP αν από την προηγούμενη σύγκριση το CY=0

JC: θα εκτελέσει την εντολή JMP αν από την προηγούμενη σύγκριση το CY=1

Έχοντας κατανοήσει τα παραπάνω βήματα μπορούμε να αντιμετωπίσουμε κάθε πρόβλημα που απαιτεί τη χρήση σύγκρισης και διακλάδωσης.

Παράδειγμα 3.1

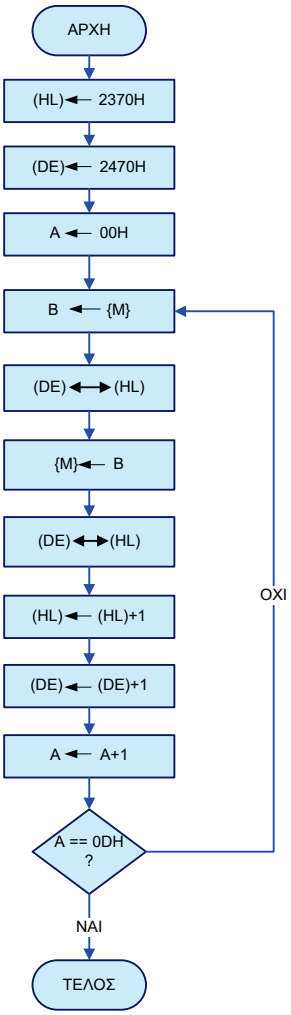
Γράψτε ένα πρόγραμμα στο οποίο θα γίνεται μεταφορά των περιεχομένου των θέσεων μνήμης από 2370H έως και 237CH στις θέσεις από 2470H έως και 247CH.

Επιβεβαιώστε την εγγραφή στις συγκεκριμένες θέσεις μνήμης.

Υπόδειξη: Αρχικά να γίνει διάγραμμα ροής. Να χρησιμοποιήστε ως μετρητή τον καταχωρητή A.

Διεύθυνση	Περιεχόμενα	Περιεχόμενα	Διεύθυνση
2370H	??H	??H XXH	2470H
2371H	??H	??H XXH	2471H
...	...	...	...
237CH	??H	??H XXH	247CH

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ



Ακολουθεί ο κώδικας στο gnusim8085.

```
;*********************************************************************
********
;LABORATORY EXERCISE 3.1
;FILE LAB3_1.ASM
;*********************************************************************
********
LXI H,2370H ;οι HL δείχνουν στην διεύθυνση μνήμης 2370H
LXI D,2470H ;οι DE δείχνουν στην διεύθυνση μνήμης 2470H
MVI A,00H   ;Αρχικοποίηση του A στο 0 για χρήση του ως μετρητή.
TEST1:  MOV B,M ;μεταφορά του περιεχομένου μνήμης που δείχνουν οι HL στον B
XCHG
MOV M,B ;μεταφορά περιεχομένου B στη διεύθυνση μνήμης που δείχνουν οι HL.
XCHG      ;ανταλλαγή των H και L με D και E
INX H     ;αύξηση του ζεύγους καταχωρητών HL.
INX D     ;αύξηση του ζεύγους καταχωρητών DE.
INR A     ;αύξηση του μετρητή
CPI 0DH    ;σύγκριση μετρητή (A) με το πλήθος των επαναλήψεων
JNZ TEST1  ;τέλος βρόχου
HLT
```

Παράδειγμα 3.2

Να γραφεί πρόγραμμα το οποίο αρχικά να αποθηκεύει δύο τυχαίους αριθμούς στις θέσεις μνήμης 2320H και 2321H. Την αποθήκευση να την κάνετε μέσω του προγράμματος GNUSim8085, πηγαίνοντας στις θέσεις μνήμης 2320H και 2321H και να βάλετε κάποια δεδομένα.

Στην συνέχεια να τοποθετεί το μεγαλύτερο (περιεχόμενο) στην θέση μνήμης 2322H και το μικρότερο (περιεχόμενο) στην 2323H.

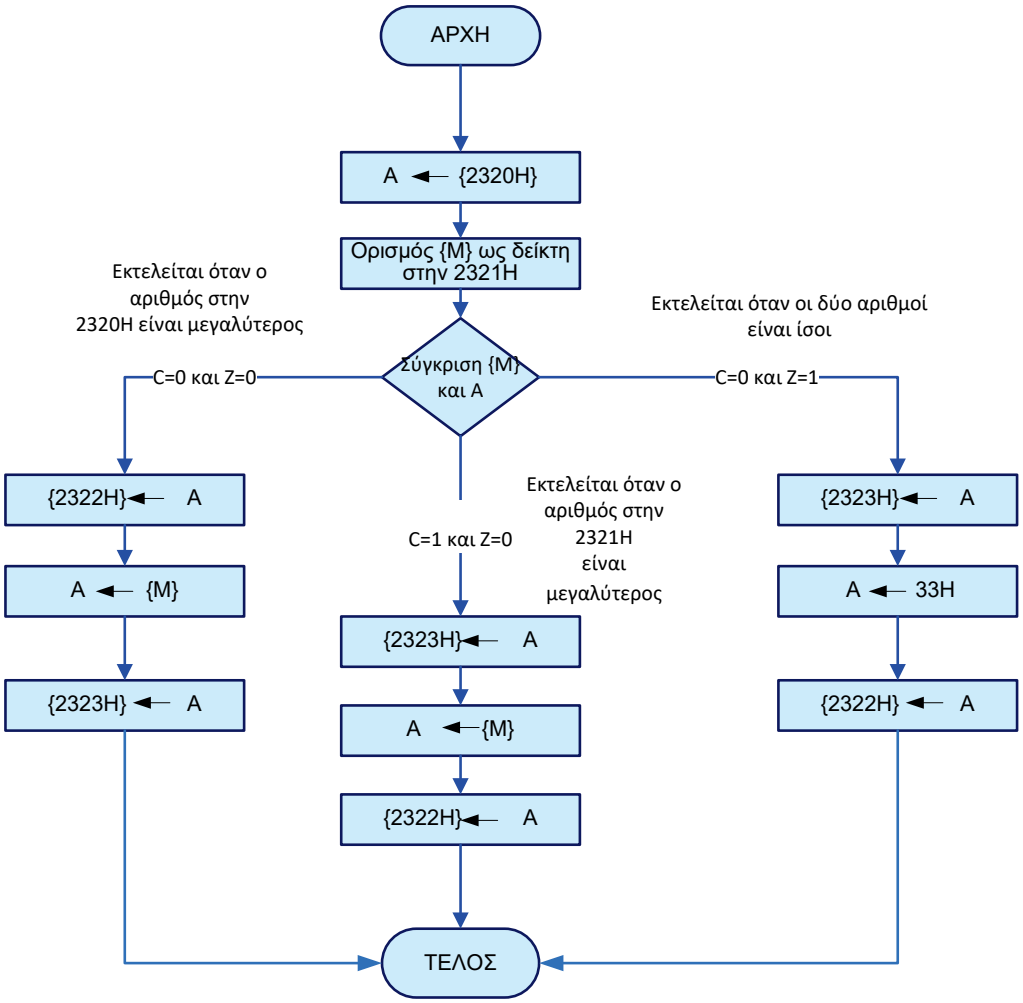
Στην περίπτωση όμως που είναι ίσα τα δύο περιεχόμενα, να το τοποθετεί το 33H στην θέση μνήμης 2322H και στη θέση μνήμης 2323H τον αριθμό (το περιεχόμενο μίας από της δύο θέσης μνήμης).

Τρέξτε το πρόγραμμα τρεις φορές,

- δύο με άνισους αριθμούς για να επιβεβαιώσετε την εύρεση του μέγιστου / ελάχιστου (μία για {2320H}>{2321H} και μία για {2320H}<{2321H}) και
- μια με ίσους ({2320H}={2321H}) για να επιβεβαιώσετε την λειτουργία του στην ισότητα.

Υπόδειξη: Να γράψετε το διάγραμμα ροής.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ



Ακολουθεί ο κώδικας στο gnusim8085

```
;*************************************************************************
********
;LABORATORY EXERCISE 3.2
;FILE LAB3_2.ASM
;*************************************************************************
********
LDA 2320H
LXI H,2321H ;άμεση φόρτωση του 2321H στον H,L
CMP M       ;Σύγκρισή του A με ότι περιέχει η μνήμη που δείχνει ο H,L
JZ LABEL1   ;σε ισότητα το Zero flag γίνετε 1. Μεταφορά στην LABEL1
JC LABEL2   ;διακλάδωση στην LABEL2 όταν ο αριθμός στην 2321H είναι μεγαλύτερος
STA 2322H   ;αποθήκευση του A στην διεύθυνση μνήμης 2322H
MOV A,M ;μεταφορά δεδομένου από την [H,L] μνήμη στον A
STA 2323H   ;αποθήκευση του A στην διεύθυνση μνήμης 2323H
JMP LABELEND    ;διακλάδωση στο LABEL_END
LABEL1: STA 2323H   ;αποθήκευση του A στην διεύθυνση μνήμης 2323H
MVI A,33H   ;Φόρτωμα του A με 33H τιμή
STA 2322H   ;αποθήκευση του A στην διεύθυνση μνήμης 2322H
JMP LABELEND    ;διακλάδωση στο LABELEND
LABEL2: STA 2323H   ;αποθήκευση του A στην διεύθυνση μνήμης 2323H
```

```
MOV A,M ;μεταφορά δεδομένου από την [H,L] μνήμη στον A
STA 2322H  ;αποθήκευση του A στην διεύθυνση μνήμης 2322H
LABELEND: HLT
```

ΑΣΚΗΣΗ 4 – Αριθμητικές Πράξεις – Πολλαπλασιασμό, Διαίρεση

Πολλοί 8-bit ή RISC μικροεπεξεργαστές και μικροελεγκτές, όπως και ο 8085, δεν περιλαμβάνουν στο σετ εντολών τους πολύπλοκές αριθμητικές πράξεις, όπως ο πολλαπλασιασμός και η διαίρεση. Σε αυτήν την άσκηση θα δείξουμε έναν από τους τρόπους με τους οποίους μπορούμε να εκτελέσουμε αυτές τις πράξεις μεταξύ 8-bit δυαδικών αριθμών.

Η Πράξη του Πολλαπλασιασμού μέσω Διαδοχικών Προσθέσεων

Η πράξη του πολλαπλασιασμού δεν είναι τίποτα άλλο από την διαδοχική πρόσθεση του πολλαπλασιαστέου τόσες φορές όσες είναι η τιμή του πολλαπλασιαστή. Για χάριν απλότητας θα θεωρήσουμε ότι πρόκειται για θετικούς (απρόσημους) αριθμούς και θα θεωρήσουμε ότι το αποτέλεσμα του πολλαπλασιασμού (γινόμενο) μπορεί να αποθηκευτεί σε πεδίο 8-bits.

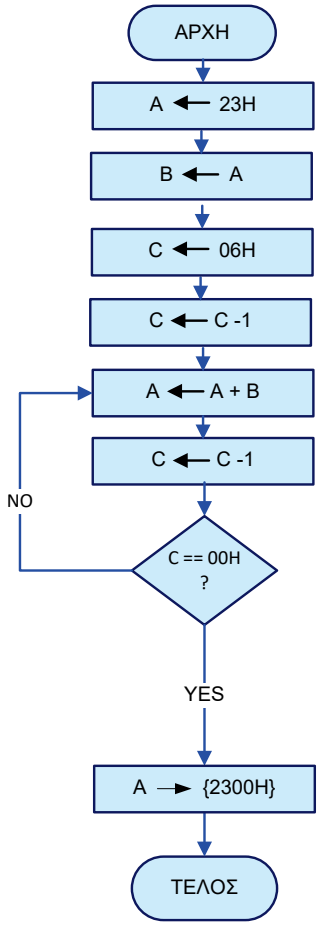
Παράδειγμα 4.1 - Πολλαπλασιασμός

Να γραφεί πρόγραμμα το οποίο να πολλαπλασιάζει τους 8-bit αριθμούς 23H και 06H και το αποτέλεσμα του πολλαπλασιασμού να αποθηκεύεται στην θέση μνήμης 2300H.

Υπόδειξη: Να γίνει μέσω μιας επανάληψης διαδοχικών προσθέσεων.

Να φτιάξετε το διάγραμμα ροής.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ



Ο κώδικας στο gnusim8085

```
;*****************************************************************
;LABORATORY EXERCISE 4.1
;FILE LAB4_1.ASM
;*****************************************************************
MVI A,23H   ;Φόρτωσε στον A τον πολλαπλασιαστέο
MOV B,A ;Μετέφερε τον στον B
MVI C,06H   ;Φόρτωσε στον C τον πολλαπλασιαστή
DCR C      ;μείωση του μετρητή
LABEL:  ADD B      ;εκτέλεση της διαδοχικής πρόσθεσης
DCR C      ;μείωση του μετρητή
JNZ LABEL   ;τέλος βρόχου
STA 2300H   ;Αποθήκευσε τον A στην θέση μνήμης 2300H
HLT
```

Άλλος τρόπος επίλυσης της άσκησης

```
MVI A,23H   ;Φόρτωσε στον A τον πολλαπλασιαστέο
MOV B,A ;Μετέφερε τον στον B
MVI C,00H
LABEL:  ADD B      ;εκτέλεση της διαδοχικής πρόσθεσης
MOV D, A
INR C      ;αύξηση του μετρητή
MOV A,C
CPI 05H
MOV A,D
JNZ LABEL   ;τέλος βρόχου
STA 2300H   ;Αποθήκευσε τον A στην θέση μνήμης 2300H
HLT
```

Παράδειγμα 4.2 - Διαίρεση

Να γραφεί πρόγραμμα το οποίο να διαιρεί τον αριθμό ADHμε τον αριθμό 13H και το αποτέλεσμα της διαίρεσης να αποθηκεύεται στην θέση μνήμης 2310H.

Υπόδειξη: α χρησιμοποιηθεί ένας μετρητής, οποίος θα μετράει τις διαδοχικές αφαιρέσεις του διαιρέτη από τον διαιρετέο. Ο βρόγχος επανάληψης θα σταματάει την στιγμή που θα προκύψει αφαίρεση με κρατούμενο. Η τιμή του μετρητή (πριν η αφαίρεση μας δώσει κρατούμενο) θα είναι το πηλίκο της διαίρεσης.

Στην συνέχεια δίνεται ο κώδικας που υλοποιεί την πράξη της διαίρεσης.

Ο κώδικας στο gnusim8085

```
;*****************************************************************
;LABORATORY EXERCISE 4.2
;FILE LAB42.ASM
;*****************************************************************
MVI A,ADH   ;Φόρτωσε στον A τον διαιρετέο
STA 2300H   ;Αποθήκευσε τον A στην θέση μνήμης 2300H
```

```
MVI A,13H   ;Φόρτωσε στον A τον διαιρέτη
STA 2301H   ;Αποθήκευσε τον A στην θέση μνήμης 2301H
MOV B,A ;Μετέφερε τον A στον B
LDA 2300H   ;Φόρτωσε στον A το περιεχόμενο της 2300H
MVI C,00H   ;Φόρτωσε στον C το 0
LOOP:   SUB B       ;εκτέλεση της διαδοχικής αφαίρεσης
INR C       ;αύξηση του μετρητή
JNC LOOP        ;Για όσο δεν έχω CARRY κάνε άλμα στην ετικέτα LOOP
DCR C       ;μείωση του μετρητή
MOV A,C ;Μετέφερε τον C στον A
STA 2310H   ; Αποθήκευσε τον A στην θέση μνήμης 2310H
HLT
```

ΑΣΚΗΣΗ 5 – Στοίβα, Υπορουτίνες, Εύρεση Μέγιστου

Στοίβα και Υπορουτίνες

Οι στοίβα και οι υπορουτίνες αποτελούν τα δύο πρωταρχικά και βασικά στοιχεία για την συγγραφή δομημένου κώδικα. Ο δομημένος ή αλλιώς τμηματικός προγραμματισμός αναφέρεται στην τεχνική σχεδίασης και ανάπτυξης προγραμμάτων ως ένα σύνολο από απλούστερα τμήματα προγραμμάτων. Η τεχνική του δομημένου προγραμματισμού μας εξασφαλίζει σε μεγάλο βαθμό την επιτυχή και εύκολη δημιουργία σωστών προγραμμάτων. Πιο συγκεκριμένα τα πλεονεκτήματα του δομημένου προγραμματισμού είναι:

- *Διευκολύνει την ανάπτυξη του προγράμματος.* Ένα σύνθετο πρόβλημα μπορεί να αναλυθεί στα επιμέρους απλούστερα υποπροβλήματα που το απαρτίζουν. Έτσι, η σταδιακή επίλυση των υποπροβλημάτων αυτών και η δημιουργία των αντίστοιχων υποπρογραμμάτων επιτρέπει την εξέταση και την επίλυση απλών προβλημάτων και όχι την απευθείας αντιμετώπιση του συνολικού προβλήματος.

- *Διευκολύνει την κατανόηση και τη διόρθωση του προγράμματος.* Ο χωρισμός του προγράμματος σε μικρότερα αυτοτελή τμήματα επιτρέπει τη γρήγορη διόρθωση καθενός τμήματος του χωρίς οι αλλαγές αυτές να επηρεάζουν το υπόλοιπο πρόγραμμα. Επίσης διευκολύνει τον καθένα να διαβάσει και να κατανοήσει τη λειτουργία του προγράμματος χωρίς ιδιαίτερα μεγάλο κόπο. Αυτό αποτελεί πολύ σημαντικό χαρακτηριστικό για τον σωστό προγραμματισμό, αφού ένα μεγάλο πρόγραμμα κατά τον κύκλο της ζωής του χρειάζεται να συντηρηθεί από διαφορετικούς προγραμματιστές.

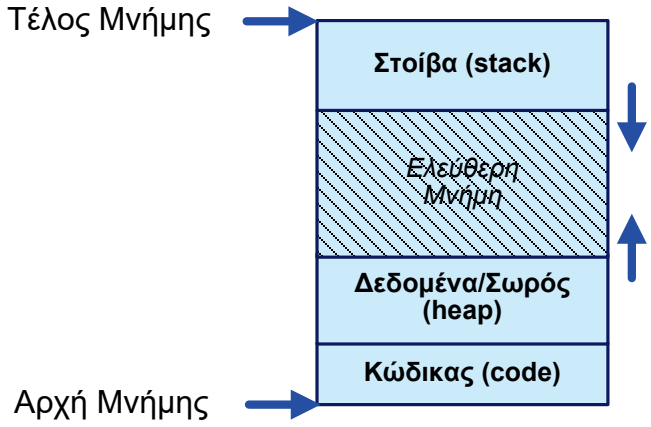
- *Απαιτεί λιγότερο χρόνο και προσπάθεια στη συγγραφή του προγράμματος.* Πολύ συχνά, μια συγκεκριμένη λειτουργία απαιτείται να επαναληφθεί σε διαφορετικά σημεία ενός προγράμματος. Από τη στιγμή που ένα υποπρόγραμμα έχει γραφεί, μπορεί να καλείται από πολλά σημεία του προγράμματος μειώνοντας με τον τρόπο αυτό το μέγεθος του προγράμματος, το χρόνο συγγραφής του και την πιθανότητα προγραμματιστικού σφάλματος, ενώ ταυτόχρονα το πρόγραμμα γίνεται πιο εύληπτο και κατανοητό.

- *Επεκτείνει τις δυνατότητες των γλωσσών προγραμματισμού.* Το υποπρόγραμμα από τη στιγμή που έχει φτιαχτεί μπορεί να χρησιμοποιηθεί από οποιοδήποτε πρόγραμμα της συγκεκριμένης γλώσσας, το οποίο κατ' επέκταση οδηγεί στη δημιουργία των βιβλιοθηκών χρήστη. Η συγγραφή πολλών υποπρογραμμάτων

και η δημιουργία βιβλιοθηκών με αυτά, υπό μία έννοια επεκτείνουν την ίδια τη γλώσσα προγραμματισμού με λειτουργίες που δεν υποστηρίζονται απευθείας από τη γλώσσα προγραμματισμού.

Στοίβα

Στην εικόνα που ακολουθεί βλέπετε το λογικό διάγραμμα της οργάνωσης της μνήμης σχεδόν όλων των υπολογιστικών συστημάτων. Εκτός από τα τμήματα του κώδικα και των δεδομένων υπάρχει και το τμήμα της στοίβας (stack). Η στοίβα επομένως δεν είναι τίποτα άλλο από ένα κομμάτι στην κύρια μνήμη (τύπου RAM) το οποίο όμως λειτουργεί με ένα ξεχωριστό τρόπο. *Η στοίβα λειτουργεί με την αρχή LIFO (Last-In-First-Out) και είναι ένα κομμάτι μνήμης που έχει μόνο ένα σημείο εισόδου/εξόδου.*



Οργάνωση της μνήμης ενός υπολογιστικού συστήματος

Στο επεξεργαστή 8085 ο stack λειτουργεί με ζευγάρια καταχωρητών (με άλλα λόγια ο stack στον 8085 είναι 16-bits). Οι δύο βασικές εντολές χειρισμού του stack στον 8085 είναι:

- PUSH rp: το περιεχόμενο του ζεύγους καταχωρητών rp μεταφέρεται στην στοίβα.
- POP rp: το περιεχόμενο των κορυφαίων θέσεων της στοίβας αποθηκεύεται στο ζεύγος καταχωρητών rp.

Σημείωση: rp (register pair ή ζευγάρι καταχωρητών) είναι ένα από τα ζευγάρια των καταχωρητών BC, DE, HL (τα ζευγάρια καταχωρητών σηματοδοτούνται πάντα από το όνομα του πρώτου καταχωρητή του ζευγαριού, δηλαδή B, D και H αντίστοιχα). Επίσης, αν το όρισμα της εντολής είναι το PSW (Processor Status Word) αναφερόμαστε στον καταχωρητή A (accumulator) μαζί με καταχωρητή F (flag register).

Υπορουτίνες

Οι υπορουτίνες είναι αυτοτελή κομμάτια κώδικα τα οποία καλούνται με την εντολή call, ενώ στο τέλος μια υπορουτίνας πρέπει να υπάρχει η εντολή RET (Return), η οποία και δηλώνει το τέλος της υπορουτίνας και την επιστροφή στο σημείο κλήσης της υπορουτίνας. Υπορούτινες μπορεί να φτιάξει ο χρήστης αλλά συνήθως υπάρχουν και έτοιμες υπορουτίνες που παρέχονται είτε από τον κατασκευαστή της γλώσσας προγραμματισμού είτε από το λειτουργικό σύστημα (π.χ. windows). + Οι εντολές

CALL και RET κάνουν χρήση της στοίβας για να μπορέσει να επιτευχθεί η επιστροφή του ελέγχου μετά την ολοκλήρωση της υπορουτίνας στην επόμενη εντολή της call από την οποία κλήθηκε. Συγκεκριμένα η CALL είναι ουσιαστικά μια JMP εντολή που όμως εκτός από το να φορτώνει τον PC με τη διεύθυνση αρχής της υπορουτίνας αποθηκεύει την προηγούμενη τιμή του PC (διεύθυνση επιστροφής) στη στοίβα. Αντίστροφα, η RET ανασύρει από την στοίβα τη διεύθυνση επιστροφής, την αποθηκεύει στον PC και με τον τρόπο αυτό η εκτέλεση του προγράμματος μπορεί να συνεχίσει από το σημείο που κλήθηκε η υπορουτίνα. Χωρίς την ύπαρξη στοίβας θα ήταν αδύνατο να υποστηριχθεί η ύπαρξη υπορουτινών και κατά συνέπεια δομημένου κώδικα τόσο σε επίπεδο συμβολικής γλώσσας όσο και σε επίπεδο γλωσσών προγραμματισμού υψηλότερου επιπέδου."

Παράδειγμα 5.1

DISPLAY: Θεωρήστε ότι υπάρχει υπορουτίνα DISPLAY, η οποία όταν την καλέσετε εμφανίζει το περιεχόμενο του Καταχωρητή A στο data field μιας οθόνης (display).

Προσοχή, ΥΠΟΘΕΣΤΕ ΟΤΙ ΜΕ ΤΗΝ ΚΛΗΣΗ ΤΗΣ συγκεκριμένης υπορουτίνας σε ένα σύστημα με πραγματική οθόνη, όλοι οι καταχωρητές χάνουν τα περιεχόμενά τους, συμπεριλαμβανομένου και του καταχωρητή σημαιών, γιατί οι καταχωρητές χρησιμοποιούνται για την αποθήκευση τοπικών τιμών. Έτσι για να μη χαθούν οι τιμές που είχαν οι καταχωρητές πριν την κλήση της display είναι επιτακτικό να αποθηκεύουν οι τιμές των καταχωρητών. Οι τιμές αυτές των καταχωρητών θα πρέπει να επανέλθουν μετά την κλήση της display για να συνεχίσει ομαλά το πρόγραμμα που την κάλεσε. Για λογούς απλότητας στην άσκηση Δηλώστε την μόνο με το όνομα της και μόνο με την εντολή RET (δηλαδή ως εξής: DISPLAY: RET).

Προσοχή, η συγκεκριμένη υπορουτίνα καταστρέφει όλους τους καταχωρητές, συμπεριλαμβανομένου και του καταχωρητή σημαιών. Δηλώστε την μόνο με το όνομα της και μόνο με την εντολή RET (δηλαδή ως εξής: DISPLAY: RET).

Μπορείτε εάν θέλετε να την εξομοιώσετέ με ένα STA 2110H, δηλαδή

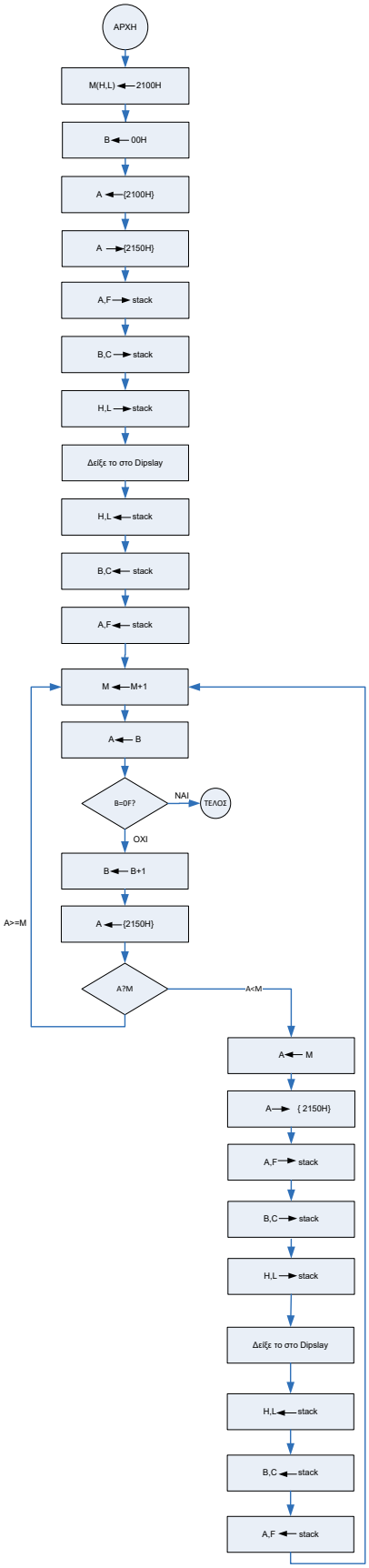
DISPLAY: STA 2110H

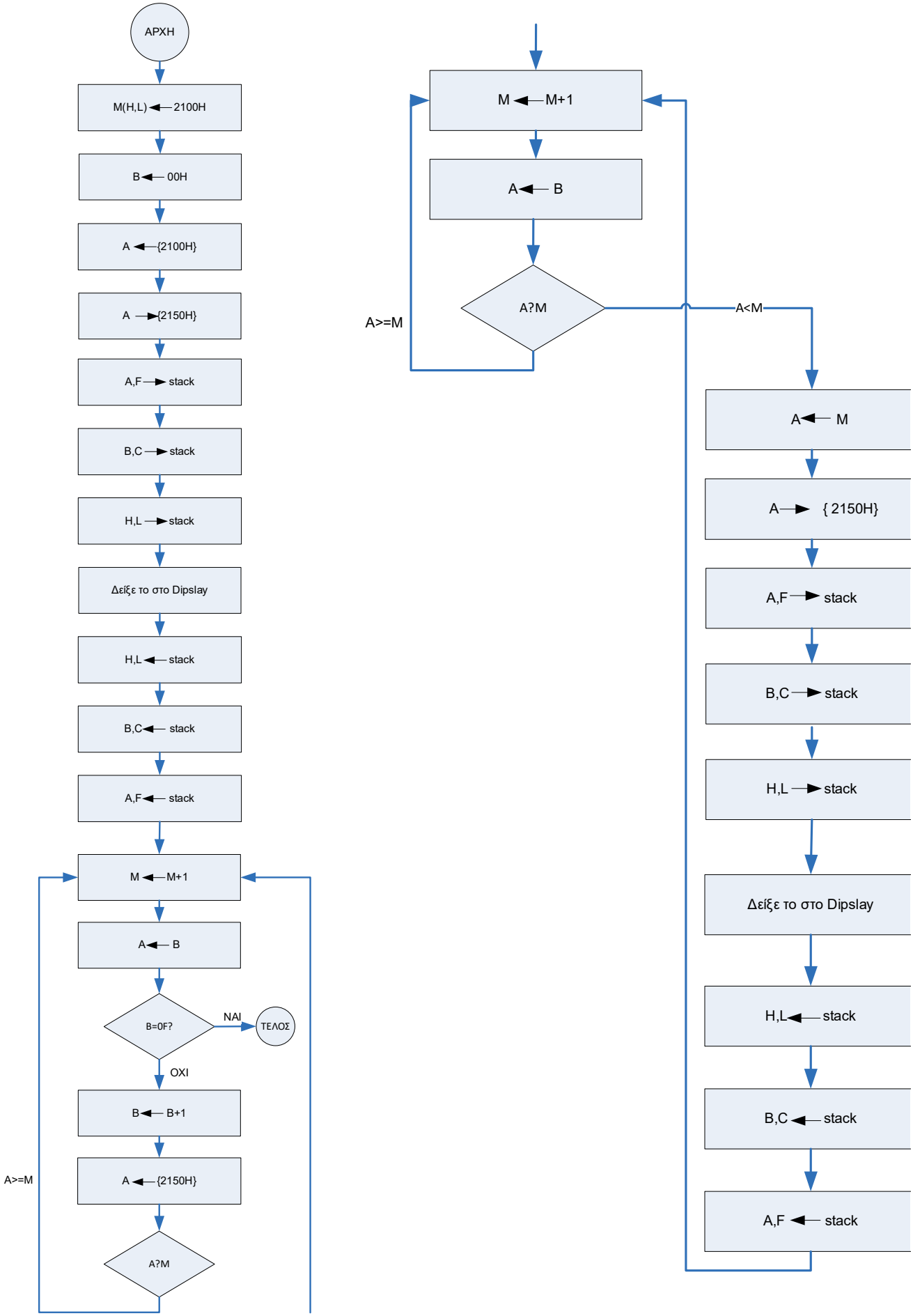
RET

Άσκηση

Βρείτε το μέγιστο περιεχόμενο μεταξύ των περιεχομένων των θέσεων μνήμης 2100H έως 210FH και να το αποθηκεύσετε στην θέση μνήμης 2150H. Βγάλτε το μέγιστο περιεχόμενο στο data field του display.

Σχεδιάζουμε το διάγραμμα ροής του προγράμματος:




ΑΡΧΗ
M(H,L) ◄— 2100H
B◄— 00H
A ◄—{2100H}
A —►{2150H}
A,F —► stack
B,C —► stack
H,L —► stack
Δείξε το στο Dipslay
H,L ◄— stack
B,C◄— stack
A,F◄— stack
M ◄—M+1
A◄— B
B=0F?
ΝΑΙ
ΤΕΛΟΣ
ΟΧΙ
B◄— B+1
A ◄—{2150H}
A>=M
A?M
M ◄—M+1
A◄— B
A?M
A>=M
A<M
A◄— M
A—► { 2150H}
A,F —► stack
B,C —► stack
H,L —► stack
Δείξε το στο Dipslay
H,L◄— stack
B,C ◄— stack
A,F ◄— stack

Ακολουθεί ο κώδικας στο gnusim8085.

Η υπορουτίνα DISPLAY εμφανίζει το περιεχόμενο του Καταχωρητή A στο data field του display. Προσοχή, η συγκεκριμένη υπορουτίνα καταστρέφει όλους τους καταχωρητές, συμπεριλαμβανομένου και του καταχωρητή σημαιών. Δηλώστε την μόνο με το όνομα της και μόνο με την εντολή RET (DISPLAY: RET).

```
;*****************************************************************
********
;LABORATORY EXERCISE 5.1
;FILE LAB5_1.ASM
;*****************************************************************
********
LXI H,2100H    ;φόρτωσε την 2100H σαν διεύθυνση στην M
MVI B,00H      ;φόρτωσε στον B το 00H
LDA 2100H      ;φόρτωσε στον A το περιεχόμενο της 2100H
STA 2150H      ;αποθήκευσε στην 2150H το περιεχόμενο του A
PUSH PSW
PUSH B
PUSH H
CALL DISPLAY       ;εμφάνισε τον A στο Display
POP H
POP B
POP PSW
LOOP1:  INX H      ;Αύξησε το ζεύγος καταχωρητών H,L (M)
MOV A,B    ;μετέφερε το περιεχόμενο του B στον A
CPI 0FH        ;σύγκρινε το περιεχόμενο του A με το 0FH
JZ LOOP3       ;εάν το περιεχόμενο του A είναι ίσο με το 0FH, πήγαινε στο LOOP3
INR B          ;αλλιώς αύξησε το περιεχόμενο του καταχωρητή B κατά 1
LDA 2150H      ;φόρτωσε στον A το περιεχόμενο της 2150H
CMP M         ;σύγκρινε το περιεχόμενο του M με το περιεχόμενο του A
JC LOOP2       ;εάν το M > A πήγαινε στο LOOP2, να τον βάλεις στην 2150H
JMP LOOP1      ;αλλιώς πήγαινε στο LOOP1
LOOP2:  MOV A,M ;μετέφερε το περιεχόμενο της μνήμης M στον A
STA 2150H      ;αποθήκευσε στην 2150H το περιεχόμενο του A
PUSH PSW
PUSH B
PUSH H
CALL DISPLAY       ;εμφάνισε τον A στο Display
POP H
POP B
POP PSW
JMP LOOP1      ;πήγαινε στο LOOP1
DISPLAY: RET
LOOP3:  HLT    ;Τέλος προγράμματος
```

Παράδειγμα 5.2

Υπολογισμός της περιόδου του 8085

Κατά την εκτέλεση διαφόρων προγραμμάτων, ορισμένες φορές απαιτείται από τον προγραμματιστή, αλλά και από τη φύση του προγράμματος, να καθυστερήσει η εκτέλεση μιας εντολής ή μια ομάδας εντολών. Αυτό στον επεξεργαστή 8085 μπορεί να γίνει εκτελώντας μια αλληλουχία εντολών, οι οποίες στο σύνολό τους δημιουργούν την επιθυμητή καθυστέρηση. Τα προγράμματα τα οποία δημιουργούν την επιθυμητή καθυστέρηση ονομάζονται προγράμματα καθυστέρησης και συχνά εμφανίζονται ως υπορουτίνες στο πρόγραμμα.

Ο χρόνος εκτέλεσης μια εντολής εξαρτάται από:

- Τους κύκλους μηχανής (cycle time) που χρειάζεται κάθε εντολή για την εκτέλεση της. Αυτή η πληροφορία είναι διαθέσιμη από τον κατασκευαστή του επεξεργαστή και υπάρχει στο εγχειρίδιο (manual) του κάθε επεξεργαστή.
- Από την συχνότητα του ρολογιού/κρυστάλλου του επεξεργαστή.

Στη περίπτωση που χρησιμοποιείται η Mikrokit αναπτυξιακή πλακέτα, ο εξωτερικός κρύσταλλος έχει συχνότητα 6.144 MHz, και το ρολόι που λειτουργεί ο μικροϋπολογιστής είναι το μισό της συχνότητας του κρυστάλλου, δηλαδή 3.072 MHz. Επομένως η περίοδος του ρολογιού (cycle time) είναι 0.325 μsec.

Έχοντας υπόψη την παραπάνω περίοδο ρολογιού στον 8085 η υλοποίηση μιας ρουτίνας εισαγωγής καθυστέρησης DELAY σε ένα πρόγραμμα assembly μπορεί να υλοποιηθεί ως εξής:

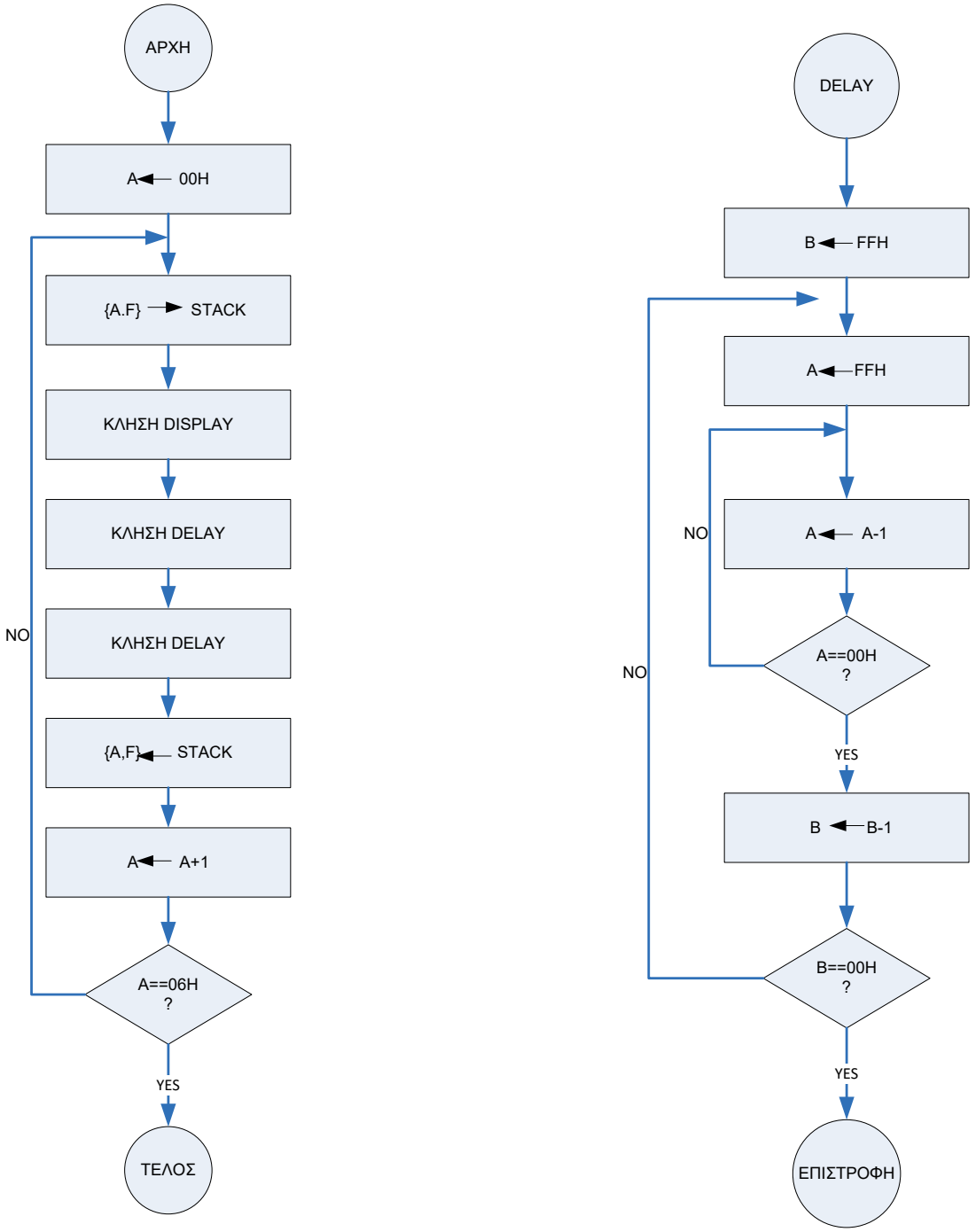
DELAY: για την υλοποίηση μιας καθυστέρησης 1'' μπορεί να χρησιμοποιηθεί διπλό Loop με τους μετρητές ΚΑΘΕ ΒΡΟΓΧΟΥ ΣΤΗΝ ΤΙΜΗ FFH. ΜΕ ΑΝΑΛΟΓΟ ΤΡΟΠΟ ΓΙΑ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΚΑΘΥΣΤΕΡΗΣΗΣ 0,5'' διπλό Loop με τον ένα μετρητή στο FFH και τον άλλο στο 7FH, 0,25'' διπλό Loop με τον ένα μετρητή στο FFH και τον άλλο στο 3FH ή διπλό Loop με τον ένα μετρητή στο 7FH και τον άλλο στο 7FH, 0,125'' διπλό Loop με τον ένα μετρητή στο FFH και τον άλλο στο 1FH, ...

Άσκηση

Να γραφεί πρόγραμμα το οποίο να εμφανίζει στο Data Field ενός display (όπως αυτό του αναπτυξιακού συστήματος Mikrokit του εργαστηρίου) τους αριθμούς 00H έως 05H. Θεωρήστε ότι για την εμφάνιση στο data field αρκεί η κλήση της DISPLAY που περιγράφηκε νωρίτερα. Μεταξύ των διαδοχικών εμφανίσεων να υπάρχει καθυστέρηση ίση με 2 seconds.

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ

Στην συνέχεια δίνεται το διάγραμμα ροής του παραδείγματος (στο διάγραμμα ροής δεν υπάρχουν οι δύο περιττές εντολές).



Ο κώδικας στο gnusim8085

Η υπορουτίνα DISPLAY εμφανίζει το περιεχόμενο του Καταχωρητή A στο data field του display. Προσοχή, η συγκεκριμένη υπορουτίνα καταστρέφει όλους τους

καταχωρητές, συμπεριλαμβανομένου και του καταχωρητή σημαιών. Δηλώστε την μόνο με το όνομα της και μόνο με την εντολή RET (DISPLAY: RET).

```
;*************************************************************************
********
;LABORATORY EXERCISE 5.2
;FILE LAB5_2.ASM
;*************************************************************************
********
MVI A,00H      ;Αρχικοποίηση του A (μετρητής) στο 0
LOOP1:  PUSH PSW      ;αποθήκευση του A στον stack
CALL DISPLAY      ;εμφάνισε τον A στο Data Field (ή CALL DELAY ξανά)
POP PSW       ;επαναφορά του A από τον stack (περιττή)
PUSH PSW      ;αποθήκευση του A στον stack (περιττή)
CALL DELAY    ;καθυστέρηση ίση με 1 second
CALL DELAY    ;καθυστέρηση ίση με 1 second
POP PSW       ;επαναφορά του A από τον stack
INR A         ;αύξηση του μετρητή
CPI 06H          ;σύγκριση μετρητή με το πλήθος των επαναλήψεων
JNZ LOOP1     ;τέλος βρόχου
JMP END
DISPLAY: RET
DELAY:  MVI B,0FFH     ;φόρτωσε το FFH στον B
LABEL1: MVI A,0FFH     ;φόρτωσε το FFH στον A
LABEL2: DCR A         ;μείωσε τον A κατά 1
JNZ LABEL2    ;διακλάδωση στο LABEL2 αν το Z flag είναι 0
DCR B         ;μείωσε τον B κατά 1
JNZ LABEL1    ;διακλάδωση στο LABEL1 αν το Z flag είναι 0
RET
END:    HLT
```

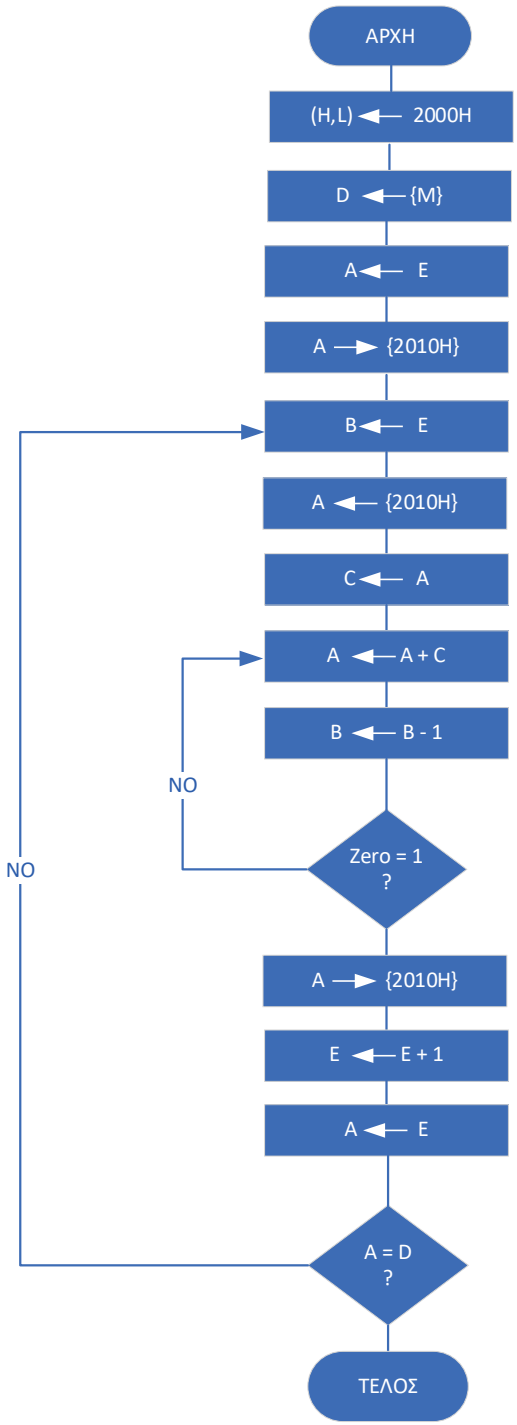
ΑΣΚΗΣΗ 6 – Παραγοντικό

Γράψτε ένα πρόγραμμα που να υπολογίζει το 4! (4 παραγοντικό).

Στην 2000H αποθηκεύεται το n και στην 2010H το n!

Ο παρακάτω κώδικας μπορεί να υπολογίσει μέχρι και το 5 παραγοντικό (επειδή το αποτέλεσμα δεν υπερβαίνει το FFH)

ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ



Ο κώδικας στο gnusim8085

```
;*****************************************************************
********
;LABORATORY EXERCISE 6.0
;FILE LAB6_0.ASM
;*****************************************************************
********
lxi h,2000h
mov d,m ; ο καταχωρητής D θα αποθηκεύει την τιμή n
mvi e,01h ; το τρέχων γινόμενο είναι 1
mov a,e ; ο καταχωρητής E θα έχει διαδοχικά τις τιμές 1, 2, 3, ..., n
sta 2010h ; αποθήκευση του τρέχοντος γινομένου
main: mov b,e ; η επόμενη τιμή με την οποία θα πολλαπλασιαστεί το τρέχον
γινόμενο να μεταφερθεί στον B
lda 2010h ; φόρτωση του A με το τρέχων γινόμενο
mov c,a ; αντίγραφο του τρέχοντος γινομένου στον C
addloop: add c ; υλοποίηση του πολλαπλασιασμού με διαδοχικές προσθέσεις
dcr b
jnz addloop
sta 2010h ; αποθήκευση του τρέχοντος γινομένου
inr e ; αύξηση του E κατά 1
mov a,e
cmp d ; έλεγχος αν το E έφτασε την τιμή n που είναι αποθηκευμένη στον D
jnz main ; αν δεν έχει ολοκληρωθεί ακόμα, ο υπολογισμός του n!, επανάληψη
hlt
```

ΑΣΚΗΣΗ 7 – Λειτουργίες I/O – Θύρες Εισόδου / Εξόδου

Αρχικοποίηση Θυρών (Α είσοδος, Β έξοδος)

```
MVI A,02H   ; Φόρτωσε στον Α το 02Η
OUT 40H ; Αρχικοποίηση θυρών (αρχικοποιείται η θύρα Α ως είσοδος και η θύρα Β ως έξοδος)
```

Διάβασμα Θύρας Α (Είσοδος)

```
IN 41H ; Διάβασμα θύρας Α. Τα δεδομένα της θύρας Α μεταφέρονται στον καταχωρητή Α.
```

Γράψιμο Θύρας Β (Έξοδος)

```
MVI A,XXH   ; Φόρτωσε στον Α το XXH
OUT 42H ; Αποστολή δεδομένων του καταχωρητή Α στην θύρα Β (έξοδος)
```

Παράδειγμα 7

Αρχικά προγραμματίστε το KIT, ώστε να έχει ΘΥΡΑ Α ΣΑΝ ΕΙΣΟΔΟ , ΘΥΡΑ Β ΣΑΝ ΕΞΟΔΟ.

Να γραφεί πρόγραμμα το οποίο ανάλογα με το ποιος διακόπτης είναι στη θέση ON, να κάνει τις παρακάτω λειτουργίες. Λειτουργίες:

Διαδικασία 1: Όταν ο διακόπτης Δ2 είναι στη θέση ON-ενεργοποιημένος (οι υπόλοιποι διακόπτες να είναι απενεργοποιημένοι) τότε όλα τα LED είναι σβηστά. Κατόπιν να ελέγχει πάλι τη θύρα Α.

Διαδικασία 2: Όταν ο διακόπτης Δ3 είναι στη θέση ON-ενεργοποιημένος (οι υπόλοιποι διακόπτες να είναι απενεργοποιημένοι) τότε να εξάγει στην θύρα Β (έξοδο) την κατάλληλη λέξη ώστε να ανάψουν τα leds L0, L1 & L2 και την κατάλληλη λέξη ώστε να ανάψουν τα leds L3 & L4 με καθυστέρηση 1 sec μεταξύ τους, δηλαδή να ανάβει ο πρώτος συνδυασμός για 1 sec και ύστερα να ανάβει ο δεύτερος συνδυασμός για 1 sec. Κατόπιν να ελέγχει πάλι τη θύρα Α.. Κατόπιν να ελέγχει πάλι τη θύρα Α.

Διαδικασία 3: Όταν οι διακόπτες Δ5 & Δ7 είναι στη θέση ON-ενεργοποιημένοι (οι υπόλοιποι διακόπτες να είναι απενεργοποιημένοι) τότε να αφαιρεί τον αριθμό 2ΑΗ από τον 6FH και το αποτέλεσμα να τοποθετείται στην 2450Η, ύστερα να κάνει την λογική πράξη OR μεταξύ του περιεχομένου της θέσης μνήμης 2450Η και του αριθμού 1FH και το αποτέλεσμα να τοποθετηθεί στην 245FH. Στην συνέχεια να τυπώνει το αποτέλεσμα στο data field Display του Mikrokit. Κατόπιν να ελέγχει πάλι τη θύρα Α.

Εάν πατηθεί άλλος συνδυασμός διακοπτών **να ξαναελέγχει πάλι τη θύρα Α.**

DISPLAY: Θεωρήστε ότι υπάρχει υπορουτίνα DISPLAY, η οποία όταν την καλέσετε εμφανίζει το περιεχόμενο του Καταχωρητή Α στο data field του display.

Προσοχή, η συγκεκριμένη υπορουτίνα καταστρέφει όλους τους καταχωρητές, συμπεριλαμβανομένου και του καταχωρητή σημαιών. Δηλώστε την μόνο με το όνομα της και μόνο με την εντολή RET (δηλαδή ως εξής: DISPLAY: RET).

Μπορείτε εάν θέλετε να την εξομοιώσετέ με ένα STA 2460H, δηλαδή

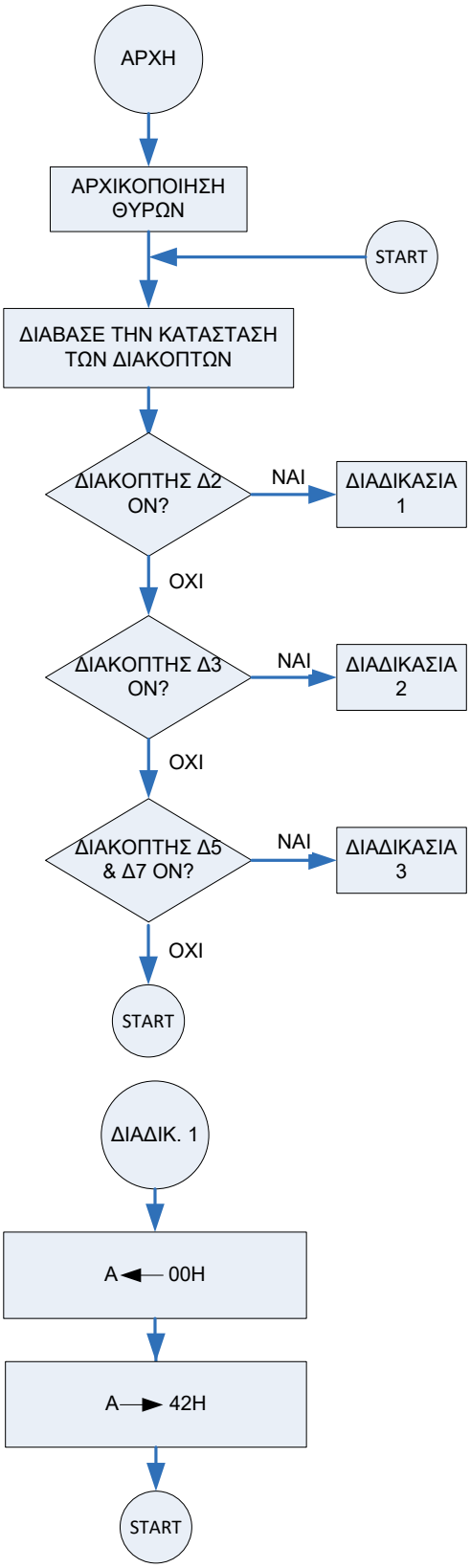
DISPLAY: STA 2460H

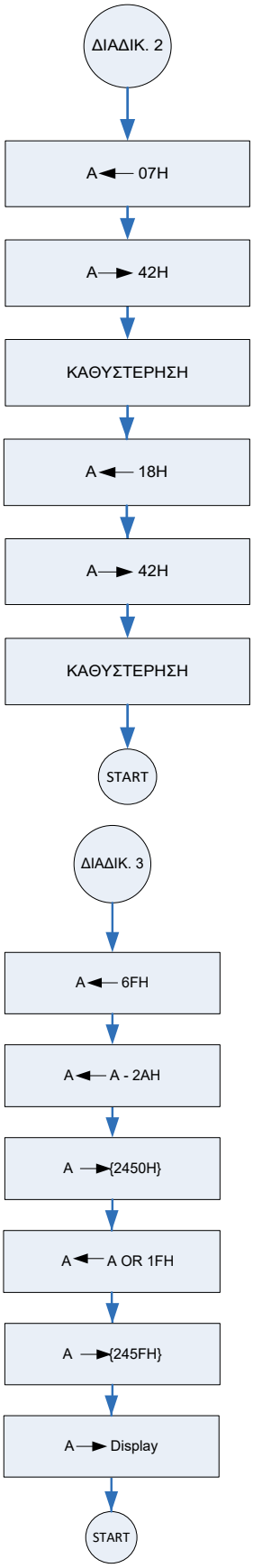
RET

DELAY: 1’’ διπλό Loop με τους μετρητές στο FFH, **0,5’’** διπλό Loop με τον ένα μετρητή στο FFH και τον άλλο στο 7FH, **0,25’’** διπλό Loop με τον ένα μετρητή στο FFH και τον άλλο στο 3FH, **0,125’’** διπλό Loop με τον ένα μετρητή στο FFH και τον άλλο στο 1FH, …

Λύση Παραδείγματος:

Σχεδιάζουμε αρχικά το διάγραμμα ροής




ΔΙΑΔΙΚ. 2
A ← 07H
A → 42H
ΚΑΘΥΣΤΕΡΗΣΗ
A ← 18H
A → 42H
ΚΑΘΥΣΤΕΡΗΣΗ
START
ΔΙΑΔΙΚ. 3
A ← 6FH
A ← A - 2AH
A → {2450H}
A ← A OR 1FH
A → {245FH}
A → Display
START

Ο κώδικας στο gnusim8085

Η υπορουτίνα DISPLAY εμφανίζει το περιεχόμενο του Καταχωρητή A στο data field του display. Προσοχή, η συγκεκριμένη υπορουτίνα καταστρέφει όλους τους καταχωρητές, συμπεριλαμβανομένου και του καταχωρητή σημαιών. Δηλώστε την μόνο με το όνομα της και μόνο με την εντολή RET (DISPLAY: RET).

```
;*****************************************************************
********
;LABORATORY EXERCISE 7
;FILE LAB7.ASM
;*****************************************************************
********
MVI A,02H   ; Φόρτωσε στον A το 02H
OUT 40H ; Αρχικοποίηση θυρών
MAIN:  IN 41H     ; Διάβασε την κατάσταση των διακοπτών
CPI 04H    ; Δ2 διακόπτης στη θέση ON?
JZ PROC_1   ; Ναι. Πήγαινε στην 1η διαδικασία
CPI 08H    ; Δ3 διακόπτης στη θέση ON?
JZ PROC_2   ; Ναι. Πήγαινε στην 2η διαδικασία
CPI 0A0H   ; Δ5-Δ7 διακόπτης στη θέση ON?
JZ PROC_3   ; Ναι. Πήγαινε στην 3η διαδικασία
JMP MAIN
PROC_1: MVI A,00H   ; Φόρτωσε στον A το 00H
OUT 42H     ; Σβήσε τα LEDs (42 ΔΙΕΥΘΥΝΣΗ ΘΥΡΑΣ B)
JMP MAIN    ; Πήγαινε στην αρχή του προγράμματος
PROC_2: MVI A,07H   ; Φόρτωσε στον A το 07H (leds L0, L1 & L2)
OUT 42H ; (42 ΔΙΕΥΘΥΝΣΗ ΘΥΡΑΣ B)
CALL DELAY  ; Κάλεσε καθυστέρηση
MVI A,18H   ; Φόρτωσε στον A το 18H (leds L3 & L4 )
OUT 42H ; (42 ΔΙΕΥΘΥΝΣΗ ΘΥΡΑΣ B)
CALL DELAY  ;Κάλεσε καθυστέρηση
JMP MAIN    ; Πήγαινε στην αρχή του προγράμματος
PROC_3: MVI A,6FH   ; Φόρτωσε στον A το 6F
SUI 2AH    ; Αφαίρεσε από τον A το 2AH (6F-2A)
STA 2450H   ; Αποθήκευση του αποτελέσματος στην 2450H
ORI 1FH    ; OR μεταξύ του αποτελέσματος της αφαίρεσης
             ; και του 1FH ({2450H}OR 1FH)
STA 245FH   ; Αποθήκευση του αποτελέσματος στην 245FH
CALL DISPLAY    ;εμφάνισε τον A στο Data Field (ή CALL DELAY ξανά)
JMP MAIN    ; Πήγαινε στην αρχή του προγράμματος
DISPLAY: RET
DELAY:  MVI B,0FFH  ; φόρτωσε το FFH στον B
LABEL1: MVI A,0FFH  ; φόρτωσε το FFH στον A
LABEL2: DCR A       ; μείωσε τον A κατά 1
JNZ LABEL2  ; διακλάδωση στην LABEL2 αν το ZERO flag είναι 0
DCR B      ; μείωσε τον B κατά 1
JNZ LABEL1  ; διακλάδωση στην LABEL1 αν το ZERO flag είναι 0
RET     ; επιστροφή από την υπορουτίνα
HLT
```

ΑΣΚΗΣΗ 8 – Λειτουργίες I/O – Θύρες Εισόδου / Εξόδου

Παράδειγμα 8

Αρχικά προγραμματίστε το ΚΙΤ, ώστε να έχει ΘΥΡΑ Α ΣΑΝ ΕΙΣΟΔΟ , ΘΥΡΑ Β ΣΑΝ ΕΞΟΔΟ.

Να γραφεί πρόγραμμα το οποίο ανάλογα με το ποιος διακόπτης είναι στη θέση ON, να κάνει τις παρακάτω λειτουργίες. Λειτουργίες:

Διαδικασία 1: Όταν ο διακόπτης Δ2 & Δ4 & Δ6 είναι στη θέση ON-ενεργοποιημένος (οι υπόλοιποι διακόπτες να είναι απενεργοποιημένοι) τότε εκτελείτε πολλαπλασιασμό μεταξύ των περιεχομένων των θέσεων μνήμης 2500H και 2501H, το αποτέλεσμα του πολλαπλασιασμού να τοποθετηθεί στην θέση μνήμης 2510H. Κατόπιν να ελέγχει πάλι τη θύρα Α.

Διαδικασία 2: Όταν ο διακόπτης Δ3 είναι στη θέση ON-ενεργοποιημένος (οι υπόλοιποι διακόπτες να είναι απενεργοποιημένοι) τότε να εξάγει στην θύρα Β (έξοδο) την κατάλληλη λέξη ώστε να ανάψουν τα leds L0, L3, L5, & L7 (τα υπόλοιπα σβηστά) και την κατάλληλη λέξη ώστε να ανάψουν τα leds L1, L2, L4 & L6 (τα υπόλοιπα σβηστά) με καθυστέρηση 1 sec μεταξύ τους, δηλαδή να ανάβει ο πρώτος συνδυασμός για 1 sec και ύστερα να ανάβει ο δεύτερος συνδυασμός για 1 sec. Κατόπιν να ελέγχει πάλι τη θύρα Α.

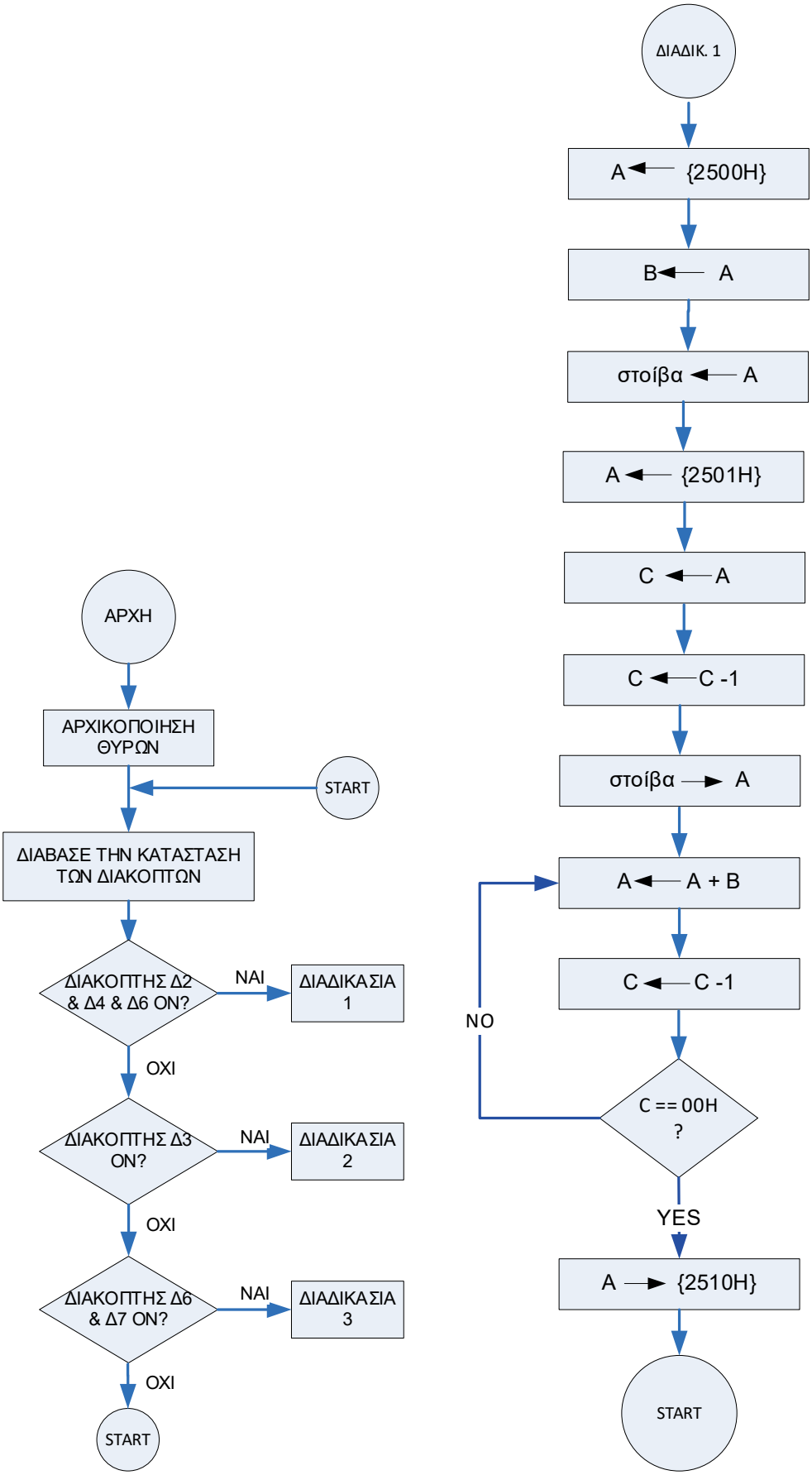
Διαδικασία 3: Όταν ο διακόπτης Δ6 & Δ7 είναι στη θέση ON-ενεργοποιημένοι (οι υπόλοιποι διακόπτες να είναι απενεργοποιημένοι) τότε να κάνει την λογική πράξη OR μεταξύ του αριθμού 0AH και του αριθμού B0H και το αποτέλεσμα να τοποθετηθεί στην 2210H, ύστερα να αφαιρεί από το περιεχόμενο της θέσης μνήμης 2220H (βάλτε με την βοήθεια του εξομοιωτή ότι περιεχόμενο θέλετε στην 2220H, όχι με χρήση εντολών, π.χ. 1AH) το περιεχόμενο της θέσης μνήμης 2230H (αφού πριν έχετε βάλει στην 2230H τον αριθμό 1DH με χρήση εντολών, μπορεί να γίνει στην αρχή της διαδικασίας) και το αποτέλεσμα να τοποθετηθεί στην 2240H. Κατόπιν να ελέγχει πάλι τη θύρα Α.

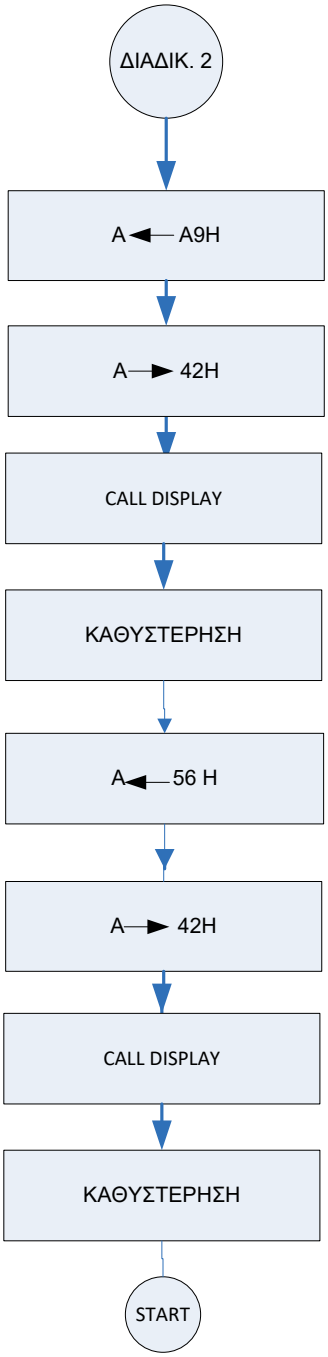
Εάν πατηθεί άλλος συνδυασμός διακοπτών **<u>να ξαναελέγχει πάλι τη θύρα Α.</u>**

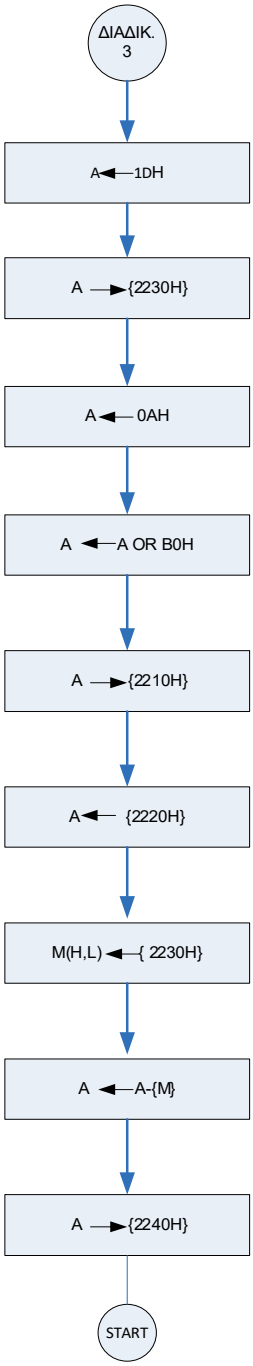
DELAY: για την Delay: 1’’ διπλό Loop με τους μετρητές στο FFH, 0,5’’ διπλό Loop με τον ένα μετρητή στο FFH και τον άλλο στο 7FH, 0,25’’ διπλό Loop με τον ένα μετρητή στο FFH και τον άλλο στο 3FH ή διπλό Loop με τον ένα μετρητή στο 7FH και τον άλλο στο 7FH, 0,125’’ διπλό Loop με τον ένα μετρητή στο FFH και τον άλλο στο 1FH, …

DISPLAY: Η υπορουτίνα DISPLAY εμφανίζει το περιεχόμενο του Καταχωρητή Α στο data field του display. Προσοχή, η συγκεκριμένη υπορουτίνα καταστρέφει όλους τους καταχωρητές, συμπεριλαμβανομένου και του καταχωρητή σημαιών. Δηλώστε την μόνο με το όνομα της και μόνο με την εντολή RET (DISPLAY: RET).

Λύση Παραδείγματος:


ΔΙΑΔΙΚ. 1
A ← {2500H}
B ← A
στοίβα ← A
A ← {2501H}
C ← A
C ← C -1
στοίβα → A
A ← A + B
C ← C -1
C == 00H ?
NO
YES
A → {2510H}
START
ΑΡΧΗ
ΑΡΧΙΚΟΠΟΙΗΣΗ ΘΥΡΩΝ
START
ΔΙΑΒΑΣΕ ΤΗΝ ΚΑΤΑΣΤΑΣΗ ΤΩΝ ΔΙΑΚΟΠΤΩΝ
ΔΙΑΚΟΠΤΗΣ Δ2 & Δ4 & Δ6 ON?
ΝΑΙ
ΔΙΑΔΙΚΑΣΙΑ 1
ΟΧΙ
ΔΙΑΚΟΠΤΗΣ Δ3 ON?
ΝΑΙ
ΔΙΑΔΙΚΑΣΙΑ 2
ΟΧΙ
ΔΙΑΚΟΠΤΗΣ Δ6 & Δ7 ON?
ΝΑΙ
ΔΙΑΔΙΚΑΣΙΑ 3
ΟΧΙ
START


ΔΙΑΔΙΚ. 2
A ◄— A9H
A —► 42H
CALL DISPLAY
ΚΑΘΥΣΤΕΡΗΣΗ
A ◄— 56 H
A —► 42H
CALL DISPLAY
ΚΑΘΥΣΤΕΡΗΣΗ
START



Ο κώδικας στο gnusim8085

Η υπορουτίνα DISPLAY εμφανίζει το περιεχόμενο του Καταχωρητή A στο data field του display. Προσοχή, η συγκεκριμένη υπορουτίνα καταστρέφει όλους τους

καταχωρητές, συμπεριλαμβανομένου και του καταχωρητή σημαιών. Δηλώστε την μόνο με το όνομα της και μόνο με την εντολή RET (DISPLAY: RET).

```
;********************************************* **********************
;LABORATORY EXERCISE 8
;FILE LAB8.ASM
```

```
;*****************************************************************
MVI A,02H  ;Φόρτωσε στον A το 02H
OUT 40H ;Αρχικοποίηση θυρών
MAIN:  IN 41H     ;Διάβασε την κατάσταση των διακοπτών
CPI 54H    ; Δ2 & Δ4 & Δ6 διακόπτης στη θέση ON?
JZ PROC_1  ;Ναι. Πήγαινε στην 1η διαδικασία
CPI 08H    ;Δ3 διακόπτης στη θέση ON?
JZ PROC_2  ;Ναι. Πήγαινε στην 2η διαδικασία
CPI 0C0H   ;Δ6 & Δ7 διακόπτης στη θέση ON?
JZ PROC_3  ;Ναι. Πήγαινε στην 3η διαδικασία
JMP MAIN
PROC_1:  LDA 2500H  ;Φόρτωσε στον A τον πολλαπλασιαστέο
MOV B,A ;Μετέφερε τον στον B
PUSH PSW      ;αποθήκευση του A στον stack
LDA 2501H
MOV C,A    ;Φόρτωσε στον C τον πολλαπλασιαστή
DCR C      ; μείωση του μετρητή
POP PSW    ;επαναφορά του A από τον stack
LABEL:  ADD B      ;εκτέλεση της διαδοχικής πρόσθεσης
DCR C      ;μείωση του μετρητή
JNZ LABEL  ;τέλος βρόχου
STA 2510H  ;Μετέφερε το αποτέλεσμα την θέση μνήμης 2300H
JMP MAIN   ;Πήγαινε στην αρχή του προγράμματος
PROC_2:  MVI A,0A9H
OUT 42H
CALL DISPLAY      ;εμφάνισε τον A στο Data Field (ή CALL DELAY ξανά)
CALL DELAY
MVI A,56H
OUT 42H
CALL DISPLAY      ;εμφάνισε τον A στο Data Field (ή CALL DELAY ξανά)
CALL DELAY
JMP MAIN   ;Πήγαινε στην αρχή του προγράμματος
PROC_3:  MVI A,1DH
STA 2230H
MVI A,0AH
ORI 0B0H
STA 2210H
LDA 2220H
LXI H,2230H
SUB M
STA 2240H
JMP MAIN   ;Πήγαινε στην αρχή του προγράμματος
DISPLAY: RET
DELAY:  MVI B,0FFH  ;φόρτωσε το FFH στον B
LABEL1: MVI A,0FFH  ; φόρτωσε το FFH στον A
LABEL2: DCR A      ;μείωσε τον A κατά 1
JNZ LABEL2  ;διακλάδωση στην LABEL2 αν το ZERO flag είναι 0
DCR B      ;μείωσε τον B κατά 1
JNZ LABEL1  ;διακλάδωση στην LABEL1 αν το ZERO flag είναι 0
```

```
RET    ;επιστροφή από την υπορουτίνα
HLT
```

Παράρτημα 1 - ΡΕΠΕΡΤΟΡΙΟ ΕΝΤΟΛΩΝ ΤΟΥ 8085

Πηγή http://www.intel.com

Instructions can be categorized according to their methodof addressing the hardware registers and/or memory.

Implied Addressing:

The addressing mode of certain instructions is implied bythe instruction's function. For example, the STC (set carry flag) instruction deals only with the carry flag, the DAA (decimal adjust accumulator) instruction deals with the accumulator.

Register Addressing:

Quite a large set of instructions call for register addressing. With these instructions, you must specify oneof the registers A through E, H or L as well as the operation code. With these instructions, the accumulatoris implied as a second operand. For example, the instruction CMP E may be interpreted as 'compare the contents of the E register with the contents of the accumulator.

Most of the instructions that use register addressingdeal with

8-bit values. However, a few of these instructions deal with 16-bit register pairs. For example, the PCHL instruction exchanges the contents of the program counterwith the contents of the H and L registers.

Immediate Addressing:

Instructions that use immediate addressing have data assembled as a part of the instruction itself. For example, the instruction CPI 'C' may be interpreted as 'compare the contents of the accumulator with the letter

C. When assembled, this instruction has the hexadecimal value FE43. Hexadecimal 43 is the internal representationfor the letter C. When this instruction is executed, theprocessor fetches the first instruction byte and determines that it must fetch one more byte. The processor fetches the next byte into one of its internalregisters and then performs the compare operation.

Notice that the names of the immediate instructions indicate that they use immediate data. Thus, the name ofan add instruction is ADD; the name of an add immediate instruction is ADI.

All but two of the immediate instructions uses the accumulator as an implied operand, as in the CPI instruction shown previously. The MVI (move immediate) instruction can move its immediate data to any of the working registers including the accumulator or to memory. Thus, the instruction MVI D, OFFH moves the hexadecimal value FF to the D register.

The LXI instruction (load register pair immediate) is even more unusual in that its

immediate data is a 16-bitvalue. This instruction is commonly used to load addresses into a register pair. As mentioned previously,your program must initialize the stack pointer; LXI is the instruction most commonly used for this purpose. Forexample, the instruction LXI SP,3OFFH loads the stack pointer with the hexadecimal value 30FF.

Direct Addressing:

Jump instructions include a 16-bit address as part of theinstruction. For example, the instruction JMP 1000H causes a jump to the hexadecimal address 1000 by replacing the current contents of the program counter with the new value 1000H.

Instructions that include a direct address require threebytes of storage: one for the instruction code, and twofor the 16-bit address

Register Indirect Addressing:

Register indirect instructions reference memory via a register pair. Thus, the instruction MOV M,C moves the contents of the C register into the memory address storedin the H and L register pair. The instruction LDAX B loads the accumulator with the byte of data specified bythe address in the B and C register pair.

Combined Addressing Modes:

Some instructions use a combination of addressing modes.A CALL instruction, for example, combines direct addressing and register indirect addressing. The directaddress in a CALL instruction specifies the address of the desired subroutine; the register indirect address is the stack pointer. The CALL instruction pushes the current contents of the program counter into the memorylocation specified by the stack pointer.

Timing Effects of Addressing Modes:

Addressing modes affect both the amount of time requiredfor executing an instruction and the amount of memory required for its storage. For example, instructions thatuse implied or register addressing, execute very quicklysince they deal directly with the processor's hardware orwith data already present in hardware registers. Most important, however is that the entire instruction can befetched with a single memory access. The number of memoryaccesses required is the single greatest factor in determining execution timing. More memory accesses therefore require more execution time. A CALL instructionfor example, requires five memory accesses: three to access the entire instruction and two more to push the contents of the program counter onto the stack.

The processor can access memory once during each processor cycle. Each cycle comprises a variable number of states. (See below and the appendix of "USING THE SDK-

85 MICROPROCESSOR TRAINER"). The length of a state depends on the clock frequency specified for your system,and may range from 480 nanoseconds to 2 microseconds. Thus, the timing for a four state instruction may range from 1.920 microseconds through 8 microseconds. (The 8085have a maximum clock frequency of 5 MHz and therefore a minimum state length of 200 nanoseconds.)

Instruction Naming Conventions:

The mnemonics assigned to the instructions are designed to indicate the function of the

instruction. The instruc-tions fall into the following functional categories:

Data Transfer Croup:

The data transfer instructions move data betweenregisters or between memory and registers.

MOV	Move
MVI	Move Immediate
LDA	Load Accumulator Directly from Memory
STA	Store Accumulator Directly in Memory
LHLD	Load H & L Registers Directly from MemorySHLD Store H & L Registers Directly in Memory

An 'X' in the name of a data transfer instruction impliesthat it deals with a register pair (16-bits);

LXI	Load Register Pair with Immediate data LDAX Load Accumulator from Address in Register Pair
STAX	Store Accumulator in Address in RegisterPair
XCHG	Exchange H & L with D & E
XTHL	Exchange Top of Stack with H & L

Arithmetic Group:

The arithmetic instructions add, subtract, increment, ordecrement data in registers or memory.

ADD	Add to Accumulator
ADI	Add Immediate Data to Accumulator
ADC	Add to Accumulator Using Carry Flag
ACI	Add Immediate data to Accumulator UsingCarry
SUB	Subtract from Accumulator
SUI	Subtract Immediate Data from Accumulator
SBB	Subtract from Accumulator Using Borrow(Carry) Flag
SBI	Subtract Immediate from Accumulator UsingBorrow (Carry) Flag
INR	Increment Specified Byte by One
DCR	Decrement Specified Byte by One
INX	Increment Register Pair by One
DCX	Decrement Register Pair by One
DAD	Double Register Add; Add Content ofRegister Pair to H & L Register Pair

Logical Group:

This group performs logical (Boolean) operations on datain registers and memory and on condition flags.

The logical AND, OR, and Exclusive OR instructions enableyou to set specific bits in the accumulator ON or OFF.

ANA	Logical AND with Accumulator
ANI	Logical AND with Accumulator Using Immediate Data
ORA	Logical OR with Accumulator
OR	Logical OR with Accumulator Using Immediate Data
XRA	Exclusive Logical OR with Accumulator
XRI	Exclusive OR Using Immediate Data

The Compare instructions compare the content of an 8-bitvalue with the contents of the accumulator;

CMP	Compare
CPI	Compare Using Immediate Data

The rotate instructions shift the contents of theaccumulator one bit position to the left or right:

RLC	Rotate Accumulator Left
RRC	Rotate Accumulator Right
RAL	Rotate Left Through Carry
RAR	Rotate Right Through CarryComplement and

carry flag instructions:

CMA	Complement Accumulator
CMC	Complement Carry Flag
STC	Set Carry Flag

Branch Group:

The branching instructions alter normal sequential program flow, either unconditionally or conditionally. The unconditional branching instructions are as follows:

JMP	Jump
CALL	Call
RET	Return

Conditional branching instructions examine the status ofone of four condition flags to determine whether the specified branch is to be executed. The conditions thatmay be specified are as follows:

NZ	Not Zero (Z = 0)
Z	Zero (Z = 1)
NC	No Carry (C = 0)
C	Carry (C = 1)
PO	Parity Odd (P = 0)
PE	Parity Even (P= 1)
P	Plus (S = 0)
M	Minus (S = 1)

Thus, the conditional branching instructions arespecified as follows:

Jumps	Calls	Returns	
C	CC	RC	(Carry)
INC	CNC	RNC	(No Carry)
JZ	CZ	RZ	(Zero)
JNZ	CNZ	RNZ	(Not Zero)
JP	CP	RP	(Plus)
JM	CM	RM	(Minus)
JPE	CPE	RPE	(Parity Even)
JP0	CPO	RPO	(Parity Odd)

Two other instructions can affect a branch by replacingthe contents or the program counter:

PCHL	Move H & L to Program Counter
RST	Special Restart Instruction Usedwith Interrupts

Stack I/O, and Machine Control Instructions:

The following instructions affect the Stack and/or StackPointer:

PUSH	Push Two bytes of Data onto the Stack
POP	Pop Two Bytes of Data off the Stack
XTHL	Exchange Top of Stack with H & L
SPHL	Move content of H & L to Stack Pointer

The I/0 instructions are as follows:

IN	Initiate Input Operation
OUT	Initiate Output Operation

The Machine Control instructions are as follows:

EI	Enable Interrupt System
DI	Disable Interrupt System
HLT	Halt
NOP	No Operation

Mnemonic	Op	SZAPC	~s	Description	Notes
ACI n	CE	*****	7	Add with Carry Immediate	A=A+n+CY
ADC r	8F	*****	4	Add with Carry	A=A+r+CY(21X)
ADC M	8E	*****	7	Add with Carry to Memory	A=A+[HL]+CY
ADD r	87	*****	4	Add	A=A+r (20X)
ADD M	86	*****	7	Add to Memory	A=A+[HL]
ADI n	C6	*****	7	Add Immediate	A=A+n
ANA r	A7	**0*0	4	AND Accumulator	A=A&r (24X)
ANA M	A6	**0*0	7	AND Accumulator and Memory	A=A&[HL]
ANI n	E6	**0*0	7	AND Immediate	A=A&n
CALL a	CD	-----	18	Call unconditional	-[SP]=PC,PC=a
CC a	DC	-----	9	Call on Carry	If CY=1(18~s)
CM a	FC	-----	9	Call on Minus	If S=1 (18~s)
CMA	2F	-----	4	Complement Accumulator	A=~A
CMC	3F	----*	4	Complement Carry	CY=~CY
CMP r	BF	*****	4	Compare	A-r (27X)
CMP M	BF	*****	7	Compare with Memory	A-[HL]
CNC a	D4	----------	9	Call on No Carry	If CY=0(18~s)
CNZ a	C4	----------	9	Call on No Zero	If Z=0 (18~s)
CP a	F4	----------	9	Call on Plus	If S=0 (18~s)
CPE a	EC	----------	9	Call on Parity Even	If P=1 (18~s)
CPI n	FE	*****	7	Compare Immediate	A-n
CPO a	E4	----------	9	Call on Parity Odd	If P=0 (18~s)
CZ a	CC	----------	9	Call on Zero	If Z=1 (18~s)
DAA	27	*****	4	Decimal Adjust Accumulator	A=BCD format
DAD B	09	----*	10	Double Add BC to HL	HL=HL+BC
DAD D	19	----*	10	Double Add DE to HL	HL=HL+DE
DAD H	29	----*	10	Double Add HL to HL	HL=HL+HL
DAD SP	39	----*	10	Double Add SP to HL	HL=HL+SP
DCR r	3D	****-	4	Decrement	r=r-1 (0X5)

DCR	M	35	****-	10	Decrement Memory	[HL]=[HL]-1	
DCX	B	0B	-----	6	Decrement BC	BC=BC-1	
DCX	D	1B	-----	6	Decrement DE	DE=DE-1	
DCX	H	2B	-----	6	Decrement HL	HL=HL-1	
DCX	SP	3B	-----	6	Decrement Stack Pointer	SP=SP-1	
DI		F3	-----	4	Disable Interrupts		
EI		FB	-----	4	Enable Interrupts		
HLT		76	-----	5	Halt		
IN p		DB	----------	10	Input	A=[p]	
INR	r	3C	****-	4	Increment	r=r+1 (0X4)	
INR	M	3C	****-	10	Increment Memory	[HL]=[HL]+1	
INX	B	03	-----	6	Increment BC	BC=BC+1	
INX	D	13	-----	6	Increment DE	DE=DE+1	
INX	H	23	-----	6	Increment HL	HL=HL+1	
INX	SP	33	-----	6	Increment Stack Pointer	SP=SP+1	
JMP	a	C3	----------	7	Jump unconditional	PC=a	
JC a		DA	----------	7	Jump on Carry	If CY=1(10~s)	
JM a		FA	----------	7	Jump on Minus	If S=1 (10~s)	
JNC a		D2	-----	7	Jump on No Carry	If CY=0(10~s)	
JNZ a		C2	-----	7	Jump on No Zero	If Z=0 (10~s)	
JP a		F2	-----	7	Jump on Plus	If S=0 (10~s)	
JPE a		EA	-----	7	Jump on Parity Even	If P=1 (10~s)	
JPO a		E2	-----	7	Jump on Parity Odd	If P=0 (10~s)	
JZ a		CA	-----	7	Jump on Zero	If Z=1 (10~s)	
LDA a		3A	-----	13	Load Accumulator direct	A=[a]	
LDAX B		0A	-----	7	Load Accumulator indirect	A=[BC]	
LDAX D		1A	-----	7	Load Accumulator indirect	A=[DE]	
LHLD a		2A	-----	16	Load HL Direct	HL=[a]	
LXI B,nn		01	-----	10	Load Immediate BC	BC=nn	
LXI D,nn		11	-----	10	Load Immediate DE	DE=nn	
LXI H,nn		21	-----	10	Load Immediate HL	HL=nn	
LXI SP,nn		31	-----	10	Load Immediate Stack Ptr	SP=nn	
MOV r1,r2		7F	-----	4	Move register to register	r1=r2 (1XX)	
MOV M,r		77	-----	7	Move register to Memory	[HL]=r (16X)	
MOV r,M		7E	-----	7	Move Memory to register	r=[HL] (1X6)	
MVI r,n		3E	-----	7	Move Immediate	r=n (0X6)	
MVI M,n		36	-----	10	Move Immediate to Memory	[HL]=n	
NOP		00	-----	4	No Operation		
ORA r		B7	**0*0	4	Inclusive OR Accumulator	A=Avr (26X)	
ORA M		B6	**0*0	7	Inclusive OR Accumulator	A=Av[HL]	
ORI n		F6	**0*0	7	Inclusive OR Immediate	A=Avn	
OUT p		D3	-----	10	Output	[p]=A	
PCHL		E9	-----	6	Jump HL indirect	PC=[HL]	
POP B		C1	-----	10	Pop BC	BC=[SP]+	
POP D		D1	-----	10	Pop DE	DE=[SP]+	
POP H		E1	-----	10	Pop HL	HL=[SP]+	
POP PSW		F1	-----	10	Pop Processor Status Word	{PSW,A}=[SP]+	

Mnemonic	Op	SZAPC	~s	Description	Notes
PUSH B	C5	-----	12	Push BC	-[SP]=BC
PUSH D	D5	-----	12	Push DE	-[SP]=DE
PUSH H	E5	-----	12	Push HL	-[SP]=HL
PUSH PSW	F5	-----	12	Push Processor Status Word	-[SP]={PSW,A}
RAL	17	----*	4	Rotate Accumulator Left	A={CY,A}<-
RAR	1F	----*	4	Rotate Accumulator Righ	A=->{CY,A}
RET	C9	----------	10	Return	PC=[SP]+
RC	D8	-----	6	Return on Carry	If CY=1(12~s)
RIM	20	-----	4	Read Interrupt Mask	A=mask
RM	F8	----------	6	Return on Minus	If S=1 (12~s)
RNC	D0	----------	6	Return on No Carry	If CY=0(12~s)
RNZ	C0	----------	6	Return on No Zero	If Z=0 (12~s)
RP	F0	----------	6	Return on Plus	If S=0 (12~s)
RPE	E8	----------	6	Return on Parity Even	If P=1 (12~s)
RPO	E0	----------	6	Return on Parity Odd	If P=0 (12~s)
RZ	C8	----------	6	Return on Zero	If Z=1 (12~s)
RLC	07	----*	4	Rotate Left Circular	A=A<-
RRC	0F	--------*	4	Rotate Right Circular	A=->A
RST z	C7	----------	12	Restart (3X7)	-[SP]=PC,PC=z
SBB r	9F	*****	4	Subtract with Borrow	A=A-r-CY
SBB M	9E	*****	7	Subtract with Borrow	A=A-[HL]-CY
SBI n	DE	*****	7	Subtract with Borrow Immed	A=A-n-CY
SHLD a	22	-----	16	Store HL Direct	[a]=HL
SIM	30	-----	4	Set Interrupt Mask	mask=A
SPHL	F9	-----	6	Move HL to SP	SP=HL
STA a	32	-----	13	Store Accumulator	[a]=A
STAX B	02	-----	7	Store Accumulator indirect	[BC]=A
STAX D	12	-----	7	Store Accumulator indirect	[DE]=A
STC	37	----1	4	Set Carry	CY=1
SUB r	97	*****	4	Subtract	A=A-r (22X)
SUB M	96	*****	7	Subtract Memory	A=A-[HL]
SUI n	D6	*****	7	Subtract Immediate	A=A-n
XCHG	EB	-----	4	Exchange HL with DE	HL<->DE
XRA r	AF	**0*0	4	Exclusive OR Accumulator	A=Axr (25X)
XRA M	AE	**0*0	7	Exclusive OR Accumulator	A=Ax[HL]
XRI n	EE	**0*0	7	Exclusive OR Immediate	A=Axn
XTHL	E3	-----	16	Exchange stack Top with HL	[SP]<->HL

PSW	-*01	Flag unaffected/affected/reset/set
S	S	Sign (Bit 7)
Z	Z	Zero (Bit 6)
AC	A	Auxilary Carry (Bit 4)
P	P	Parity (Bit 2)
CY	C	Carry (Bit 0)

a p	Direct addressing
M z	Register indirect addressing

n nn	Immediate addressing
r	Register addressing
DB n(,n)	Define Byte(s)
DB 'string'	Define Byte ASCII character string
DS nn	Define Storage Block
DW nn(,nn)	Define Word(s)
A B C D E H L	Registers (8-bit)
BC DE HL	Register pairs (16-bit)
PC	Program Counter register (16-bit)
PSW	Processor Status Word (8-bit)
SP	Stack Pointer register (16-bit)
a nn	16-bit address/data (0 to 65535)
n p	8-bit data/port (0 to 255)
r	Register (X=B,C,D,E,H,L,M,A)
z	Vector (X=0H,8H,10H,18H,20H,28H,30H,38H)
+ -	Arithmetic addition/subtraction
& ~	Logical AND/NOT
v x	Logical inclusive/exclusive OR
<- ->	Rotate left/right
<->[ ]	Exchange
[ ]	Indirect addressing
[ ]+ -[	Indirect address auto-inc/decrement
{ }	Combination operands
(X)	Octal op code where X is a 3-bit code
If (~s)	Number of cycles if condition true

Παράρτημα 2 - ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ ΤΟΥ 8085 ΜΙΚΡΟΚΙΤ

2.1 Η δομή του ΜΙΚΡΟΚΙΤ

Στο ποιο κάτω σχήμα φαίνεται η κάτοψη του MIKPOKIT 8085, το οποίο αποτελείται από το μΕ 8085, μνήμη SRAM, μνήμη EPROM, την USART 8251, το προγραμματιζόμενο περιφερειακό παράλληλης διασύνδεσης 8155, τον ελεγκτή πληκτρολογίου 8279, έξι seven segment displays (θα το ονομάζουμε εν συντομία *"display"*) και μια σειρά από συνδέσμους (connectors).

Με τη βοήθεια του MIKPOKIT μπορεί να αναπτυχθεί, να ελεγχθεί και να εκσφαλματωθεί γρήγορα και αποτελεσματικά το λογισμικό εφαρμογής για τον μΕ 8085. Ο σκοπός αυτός υποστηρίζεται από ένα ειδικού σκοπού λογισμικό το οποίο έχει αναπτυχθεί και αποθηκευτεί στην μνήμη EPROM του MIKPOKIT. Το πρόγραμμα ονομάζεται εν συντομία *"monitor"* και έχει τις πιο κάτω λειτουργίες – δυνατότητες:

Εξέταση του περιεχομένου όλων των θέσεων μνήμης (RAM και ROM) του συστήματος και απεικόνιση στο *"display"*.

Εξέταση του περιεχομένου όλων των καταχωρητών του μΕ και απεικόνιση στο *"display"*.

Αλλαγή του περιεχομένου όλων των θέσεων μνήμης RAM.

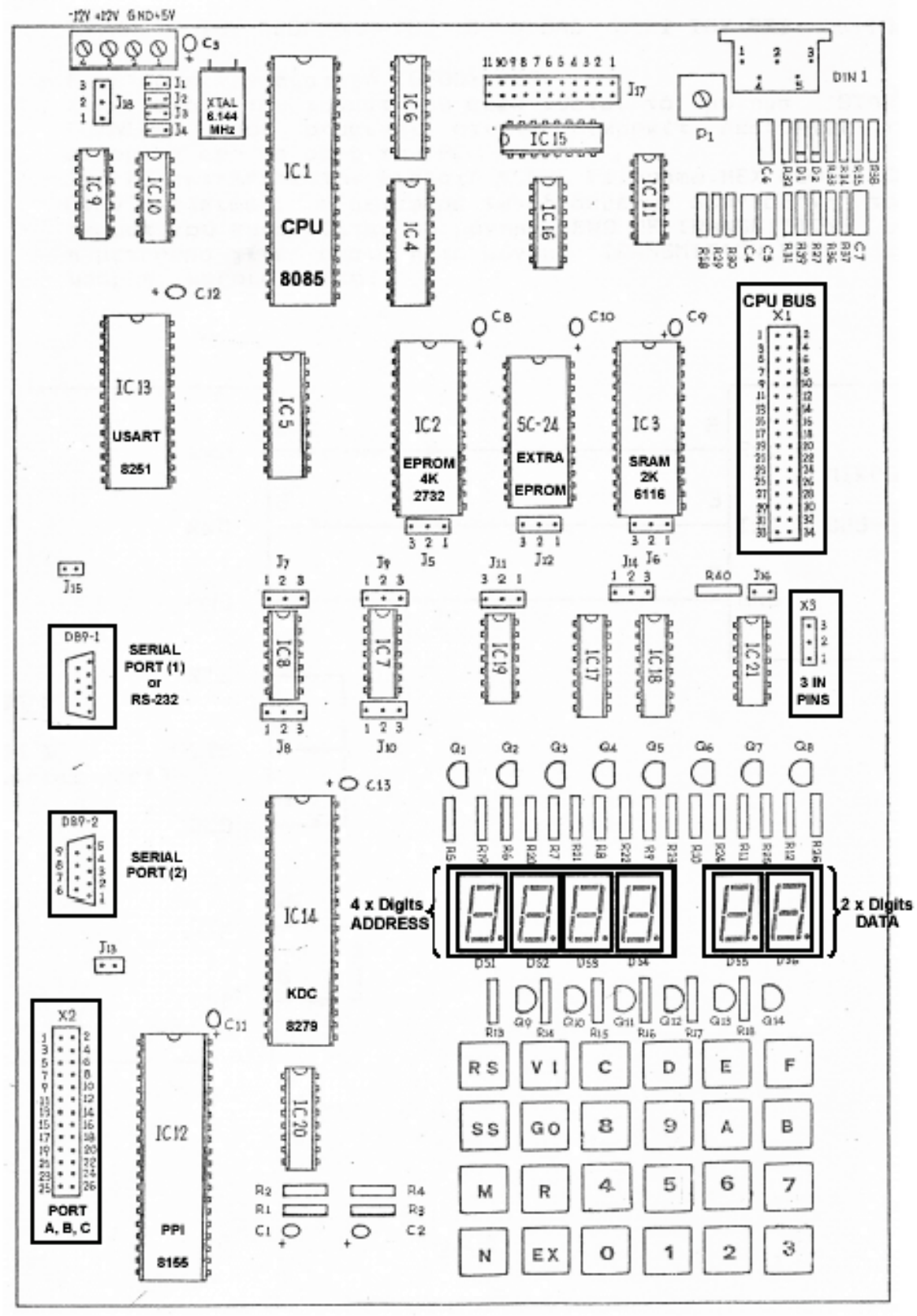
Αλλαγή του περιεχομένου όλων των καταχωρητών του μΕ.

Εκτέλεση προγράμματος.

Εκτέλεση προγράμματος βήμα – βήμα.

Επανεκκίνηση (RESET) με το πάτημα ενός πλήκτρου.

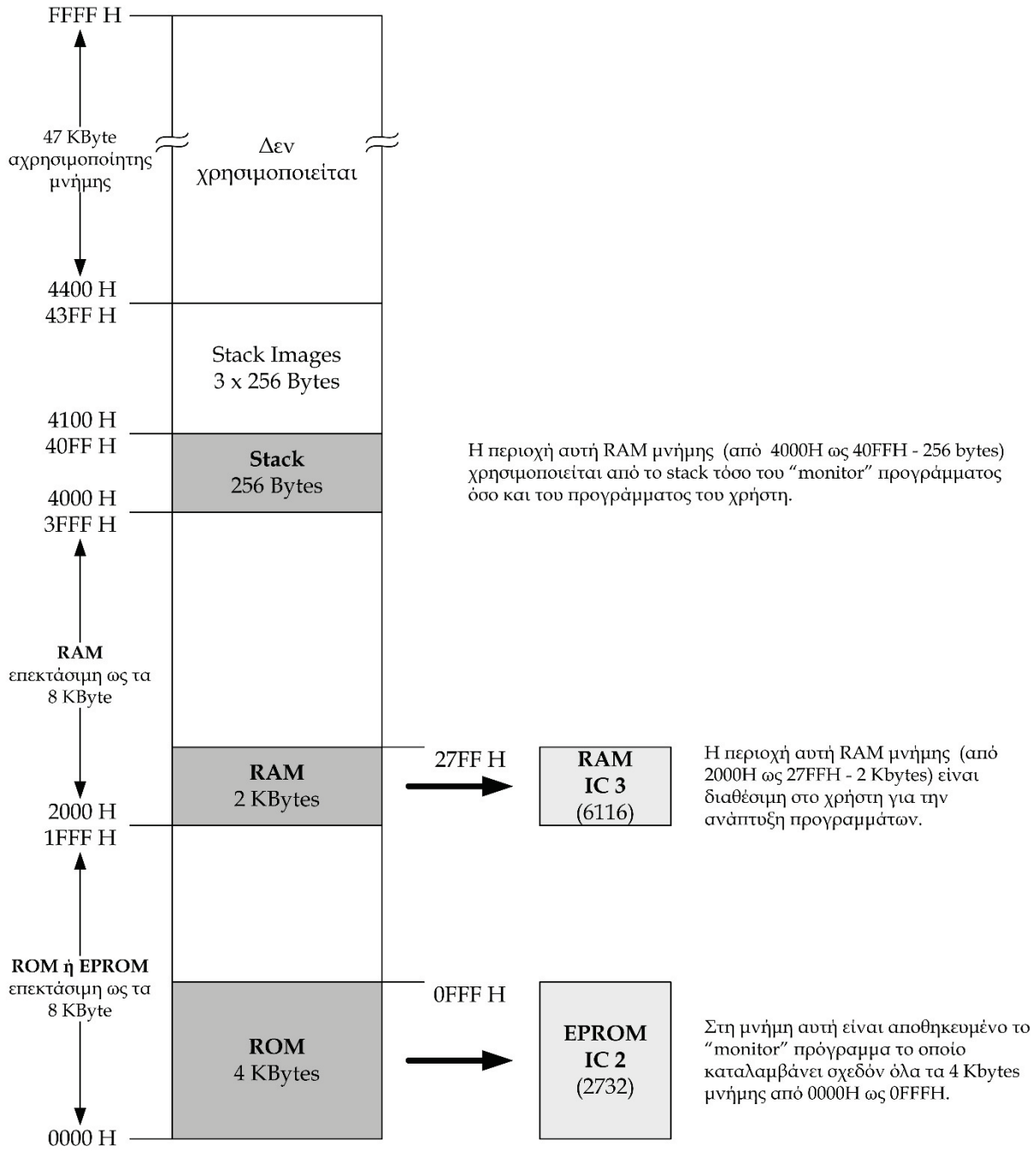
Διακοπή της λειτουργίας ενός προγράμματος (INTERRUPT) και εκτέλεση άλλου υποπρογράμματος με το πάτημα ενός πλήκτρου.



Σχήμα 8 Κάτοψη του ΜΙΚΡΟΚΙΤ 8085

2.2 Χαρτογράφηση μνήμης

Στο πιο κάτω εικόνα φαίνεται παραστατικά η δομή της μνήμης του ΜΙΚΡΟΚΙΤ.



Σχήμα 9 Χαρτογράφηση μνήμης

2.3 Είσοδοι – Έξοδοι ΜΙΚΡΟΚΙΤ

Το ΜΙΚΡΟΚΙΤ έχει τις εξής δυνατότητες εισόδου – εξόδου:

3 παράλληλες θύρες (IC12 – 8155C) – σύνδεσμος X1

Σειριακή θύρα USART (IC13 – 8251A) – σύνδεσμος DB9-1

Σειριακή θύρα η οποία υποστηρίζεται από τον μΕ8085 μέσω των ακίδων της SID και SOD – σύνδεσμος DB9-2

Κύκλωμα ελέγχου πληκτρολογίου και Display (IC14 - KDC 8279)

Όλα τα βασικά σήματα του μΕ8085 είναι διαθέσιμα στο σύνδεσμο X1

Στον πιο κάτω πίνακα φαίνονται αναλυτικά οι I/O διευθύνσεις που έχουν ανατεθεί σε διάφορα ολοκληρωμένα κυκλώματα για τον έλεγχο της λειτουργίας τους και για την διακίνηση δεδομένων.

Ολοκληρωμένο	Διεύθυνση	Διεύθυνση
USART 8251 (IC13)	1EH 1FH	Data Register Control/Status Register
KDC 8279 (IC14)	2EH 2FH	Data Register Control/Status Register
PPI 8155 (IC12)	40H 41H 42H 43H 44H 45H	Command/Status Register Data Register PORT A Data Register PORT B Data Register PORT C Timer λιγότερο σημαντικό byte Timer περισσότερο σημαντικό byte
74125 input port (IC21)	0FH	Παράλληλο θύρα εισόδου 3 bits. Η θύρα αυτή είναι συνδεδεμένη απ ευθείας στο data bus του μΕ και διαθέσιμη στο χρήστη μέσω του συνδέσμου X3

Πίνακας 1. Ανάθεση I/O διευθύνσεων στη διαχείριση των θυρών εισόδου – εξόδου.

Στους πιο κάτω πίνακες φαίνεται η ανάθεση των ακίδων στους διαθέσιμους συνδέσμους εισόδου – εξόδου.

Ακίδες	Λειτουργία
1	Ασύνδετο
2	RxD (input)
3	TxD (output)
4	CTS# (input)
5	RTS# (output)
6	Ασύνδετο
7	GND

8	Ασύνδετο
9	Ασύνδετο

Πίνακας 2. Ανάθεση ακίδων στο σύνδεσμο DB9-1 (σειριακή θύρα 8251).

Ακίδες	Λειτουργία	Ακίδες	Λειτουργία
25	PA0	13	PB4
26	PA1	14	PB5
23	PA2	11	PB6
24	PA3	12	PB7
21	PA4	9	PC0
22	PA5	10	PC1
19	PA6	7	PC2
20	PA7	8	PC3
17	PB0	5	PC4
18	PB1	6	PC5
15	PB2	2 & 4	VCC (+5V)
16	PB3	1 & 3	GND

Πίνακας 3. Ανάθεση ακίδων στο σύνδεσμο X2 (παράλληλη θύρα 8155).

Ακίδες	Λειτουργία	Ακίδες	Λειτουργία
25	A0	33	AD0
26	A1	34	AD1
23	A2	31	AD2
24	A3	32	AD3
21	A4	29	AD4
22	A5	30	AD5
19	A6	27	AD6
20	A7	28	AD7

17	A8	8	IO/M#
18	A9	7	ALE
15	A10	5	RESET OUT
16	A11	10	RD#
13	A12	9	WR#
14	A13	6	CLK OUT
11	A14	1, 2, 3, 4	GND
12	A15		

Πίνακας 4. Ανάθεση ακίδων στο σύνδεσμο X1 (CPU BUS).

Ακίδες	Λειτουργία
1	Ασύνδετη
2	SID (input)
3	SOD (output)
4	Ασύνδετη
5	Ασύνδετη
6	Ασύνδετη
7	GND
8	Ασύνδετη
9	Ασύνδετη

Πίνακας 5. Ανάθεση ακίδων στο σύνδεσμο DB9-2 (σειριακή θύρα μE8085).

2.4 Διαχείριση διακοπών στο ΜΙΚΡΟΚΙΤ

Οι διακοπές στο μE8085 διακλαδίζουν το πρόγραμμα (με εξαίρεση τη διακοπή INTR) σε θέσεις μνήμης κοντά στη θέση 0000H. Αυτή η περιοχή μνήμης είναι γνωστή ως *"interrupt vector table"* και χρησιμοποιείται για την ανάπτυξη των ρουτινών εξυπηρέτησης των διακοπών. Στο ΜΙΚΡΟΚΙΤ στην περιοχή αυτή υπάρχει η μνήμη ROM η οποία περιέχει το *"monitor"* πρόγραμμα. Αποτέλεσμα αυτού είναι να μην

δίδεται η δυνατότητα στο χρήστη να τοποθετήσει τις ρουτίνες εξυπηρέτησης διακοπών στην περιοχή αυτή.

Το πρόβλημα αυτό αντιμετωπίζεται από το *"monitor"* πρόγραμμα με την προσθήκη εντολών διακλάδωσης σε συγκεκριμένες θέσεις μνήμης οι οποίες βρίσκονται σε περιοχή στην οποία υπάρχει μνήμη RAM. Με το τρόπο αυτό δημιουργείται ένα δεύτερο *"interrupt vector table"* το οποίο θα χρησιμοποιείται πλέον για την ανάπτυξη τον ρουτινών εξυπηρέτησης των διακοπών. Θα πρέπει όμως να σημειωθεί ότι δεν είναι όλες οι πηγές διακοπών διαθέσιμες στο χρήστη του ΜΙΚΡΟΚΙΤ μιας και μερικές από αυτές χρησιμοποιούνται από το *"monitor"* για την λειτουργία του ΜΙΚΡΟΚΙΤ. Στον πιο κάτω πίνακα φαίνεται αναλυτικά ο τρόπος με τον οποίο διαχειρίζεται το *"monitor"* πρόγραμμα τις διακοπές.

Διακοπή μΕ 8085	Λειτουργία	Αρχική διεύθ. εξυπηρέτησης	Τελική διεύθ. εξυπηρέτησης
TRAP	Συνδέεται με το TIMER OUT του 8155 και χρησιμοποιείται από το *"monitor"* πρόγραμμα για την υλοποίηση της λειτουργίας *"SINGLE STEP"*.	0024H	-
RST 7.5	Συνδέεται με το πλήκτρο VI χρησιμοποιείται για τη χειροκίνητη δημιουργία διακοπής. Διατίθεται στο χρήστη.	003CH	40E3H
RST 6.5	Συνδέεται στο JUMPER J2. Διατίθεται στο χρήστη.	0034H	40DDH
RST 5.5	Συνδέεται με το 8279 και χρησιμοποιείται για να ειδοποιήσει τον μΕ ότι πατήθηκε κάποιο πλήκτρο.	002CH	-
INTR	Συνδέεται στο JUMPER J3. Διατίθεται στο χρήστη. Για να χρησιμοποιηθεί η διακοπή αυτή είναι απαραίτητη η ανάπτυξη εξωτερικού HW.	-	-
RST 0	Χρησιμοποιείται από το *"monitor"* πρόγραμμα. Υλοποιεί την λειτουργία *"COLD START"* που ισοδυναμεί με HW RESET.	0000H	-

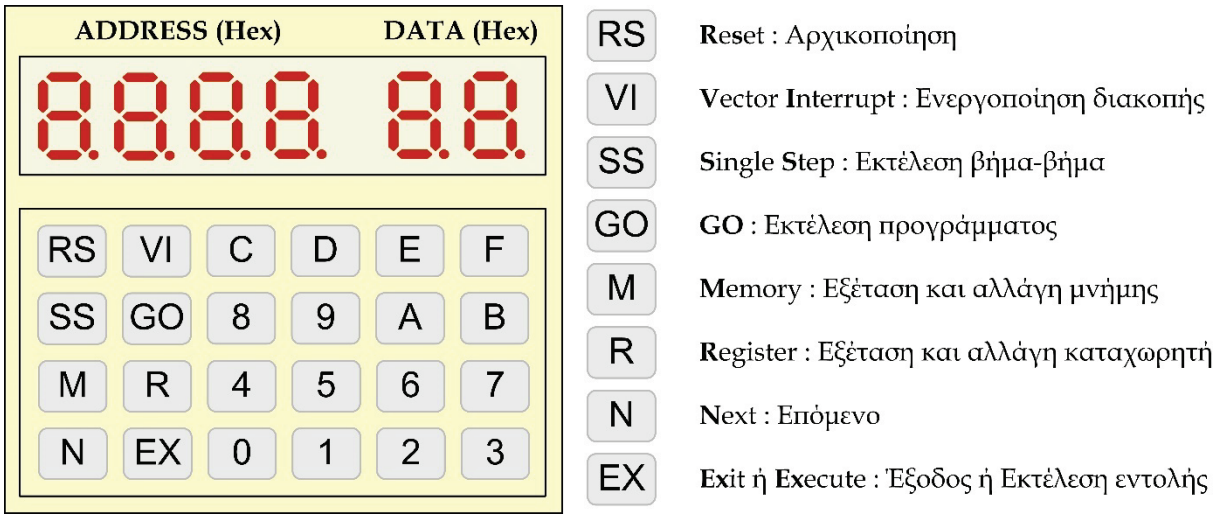
RST 1	Χρησιμοποιείται από το *"monitor"* πρόγραμμα. Υλοποιεί την λειτουργία *"WARM START"* και μπορεί να χρησιμοποιείται για την επαναφορά του ελέγχου στο *"monitor"* πρόγραμμα από πρόγραμμα του χρήστη. Για το λόγο αυτό είναι χρήσιμη επίσης αυτή η διακοπή και υλοποίηση *"break points"* στο πρόγραμμα του χρήστη.	0008H	-
RST 2	Δεν υλοποιείται, επικαλύπτεται από το *"monitor"* πρόγραμμα. Δεν πρέπει να χρησιμοποιείται.	-	-
RST 3	Δεν υλοποιείται, επικαλύπτεται από το *"monitor"* πρόγραμμα. Δεν πρέπει να χρησιμοποιείται.	-	-
RST 4	Δεν υλοποιείται, επικαλύπτεται από το *"monitor"* πρόγραμμα. Δεν πρέπει να χρησιμοποιείται.	-	-
RST 5	Διατίθεται στο χρήστη.	0028H	40D7H
RST 6	Διατίθεται στο χρήστη.	0030H	40DAH
RST 7	Διατίθεται στο χρήστη.	0038H	40E0H

Πίνακας 6. Διαχείριση διακοπών στο MIKROKIT.

2.5 Λειτουργίες του "monitor" προγράμματος

Το *"monitor"* πρόγραμμα είναι σε θέση να εκτελέσει μια σειρά από λειτουργίες, οι οποίες μπορούν να μας φανούν χρήσιμες στην ανάπτυξη προγραμμάτων εφαρμογής για τον μΕ8085 με την βοήθεια του MIKROKIT. Οι λειτουργίες αυτές εξυπηρετούνται από το πληκτρολόγιο και το display το υπάρχει στο MIKROKIT και έχει την ακόλουθη δομή.

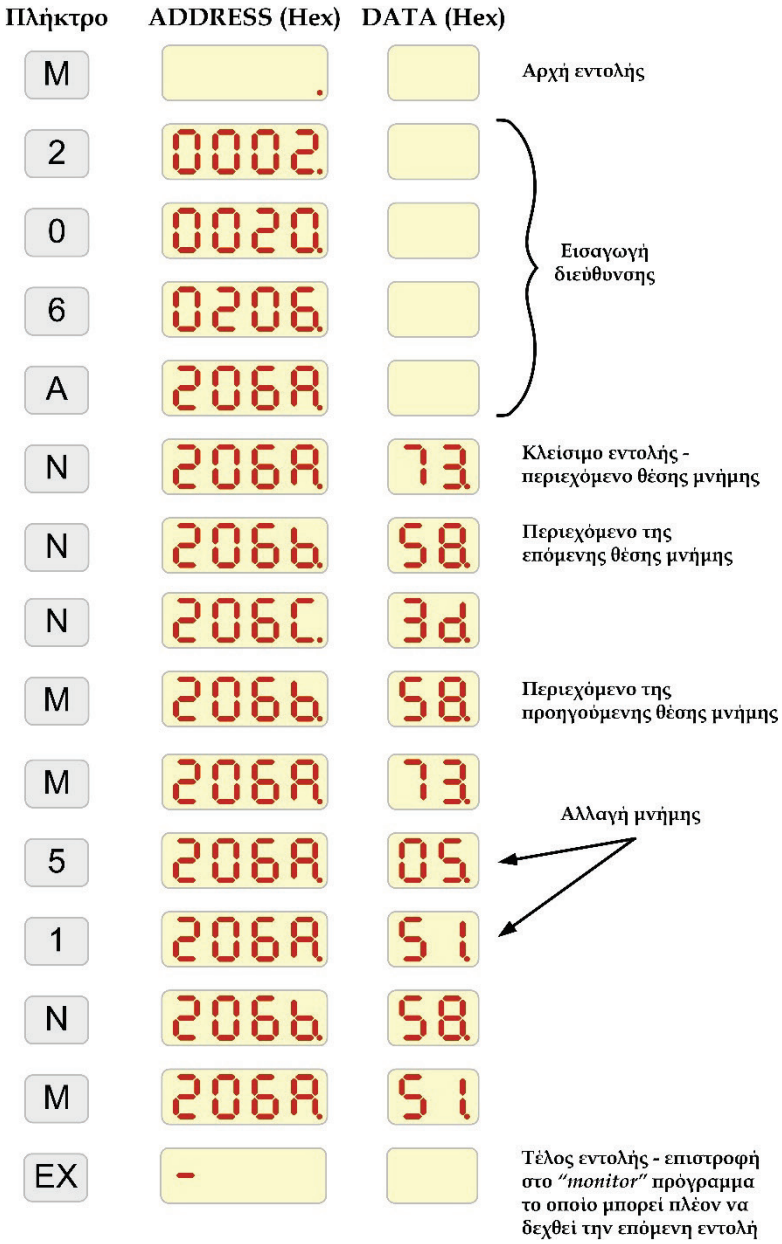
Ένα μέρος του πληκτρολογίου, το αριθμητικό, μας επιτρέπει την εισαγωγή δεκαεξαδικών αριθμών κατά την πληκτρολόγηση εντολών. Η εντολοδότηση του *"monitor"* υποστηρίζεται και από μια σειρά πλήκτρων ειδικών λειτουργιών. Το αποτέλεσμα κάθε εντολής απεικονίζεται στο display το οποίο διαιρείται σε δύο μέρη. Το πρώτο μέρος αποτελείται από τέσσερα ψηφία και απεικονίζει διευθύνσεις μνήμης ή καταχωρητές ενώ το δεύτερο απεικονίζει δεδομένα. Η δομή του πληκτρολογίου και του display καθώς επίσης και μια σύντομη επεξήγηση της λειτουργίας των πλήκτρων ειδικού σκοπού φαίνεται στην πιο κάτω εικόνα.



Εικόνα 1. Η δομή και η χρήση του πληκτρολογίου και του display.

2.5.1 Εξέταση και αλλαγή περιεχομένων μνήμης

Με την εντολή μπορεί να εξεταστεί το περιεχόμενο μιας θέσης μνήμης. Στην περίπτωση όπου η υπό εξέταση περιοχή μνήμης είναι τύπου RAM τότε υπάρχει η δυνατότητα αλλαγής του περιεχομένου της. Αν επιχειρηθεί η αλλαγή του περιεχομένου μιας θέσης ROM μνήμης το *"monitor"* πρόγραμμα το αντιλαμβάνεται και εμφανίζει στο display μήνυμα λάθους *"-Err"*. Η σύνταξη της εντολής καθώς και το αποτέλεσμα της εφαρμογής της φαίνεται παραστατικά στην πιο κάτω εικόνα.



Εικόνα 2. Εντολή εξέτασης και αλλαγής περιεχομένων μνήμης.

2.5.2 Εισαγωγή και εκτέλεση προγράμματος

Το *"monitor"* πρόγραμμα με την εντολή αλλαγής μνήμης που διαθέτει μας δίδει την δυνατότητα εισαγωγής προγράμματος από το πληκτρολόγιο. Ας υποθέσουμε ότι θέλουμε να εισάγουμε το πιο κάτω πρόγραμμα στο ΜΙΚΡΟΚΙΤ στη θέση μνήμης 2000H (περιοχή RAM) και στη συνέχεια να το εκτελέσουμε.

MVI A, 47H

RST 1

Η μετατροπή του προγράμματος αυτού σε γλώσσα μηχανής έχει ως εξής:

Διεύθυνση	Δεδομένα	Κώδικας Assembly
2000	3E	MVI A,47H

2001 47

2002 CF RST 1

Αφού εισάγουμε το πρόγραμμα σύμφωνα με τις εντολές τις παραγράφου 0 δίνουμε στο ΜΙΚΡΟΚΙΤ την εντολή εκτέλεσης προγράμματος η οποία συντάσσεται ως εξής:

ΣΥΝΤΑΞΗ: GO **Πληκτρολόγηση διεύθυνσης προγράμματος** EX

2.5.3 Εκτέλεση προγράμματος βήμα-βήμα

Το *"monitor"* πρόγραμμα μας δίδει την δυνατότητα εκτέλεσης προγράμματος βήμα. Η σύνταξη της εντολής είναι η εξής:

ΣΥΝΤΑΞΗ: GO

Πληκτρολόγηση της αρχικής διεύθυνσης του προγράμματος

N **Εκτέλεση της πρώτης εντολής**

N **Εκτέλεση της επόμενης εντολής**

⋮

EX **Τερματισνός εντολής επιστροφή στο *"monitor"* πρόγραμμα**

2.5.4 Εξέταση και αλλαγή του περιεχομένου των καταχωρητών

Το *"monitor"* πρόγραμμα μας δίδει την δυνατότητα εξέτασης του περιεχομένου των όλων καταχωρητών των καταχωρητών του μΕ καθώς επίσης και την δυνατότητα μεταβολής του περιεχομένου τους. Η σύνταξη της εντολής στην περίπτωση της εξέτασης του περιεχομένου είναι η εξής:

ΣΥΝΤΑΞΗ: R **Εμφάνιση του περιεχομένου του ACC**

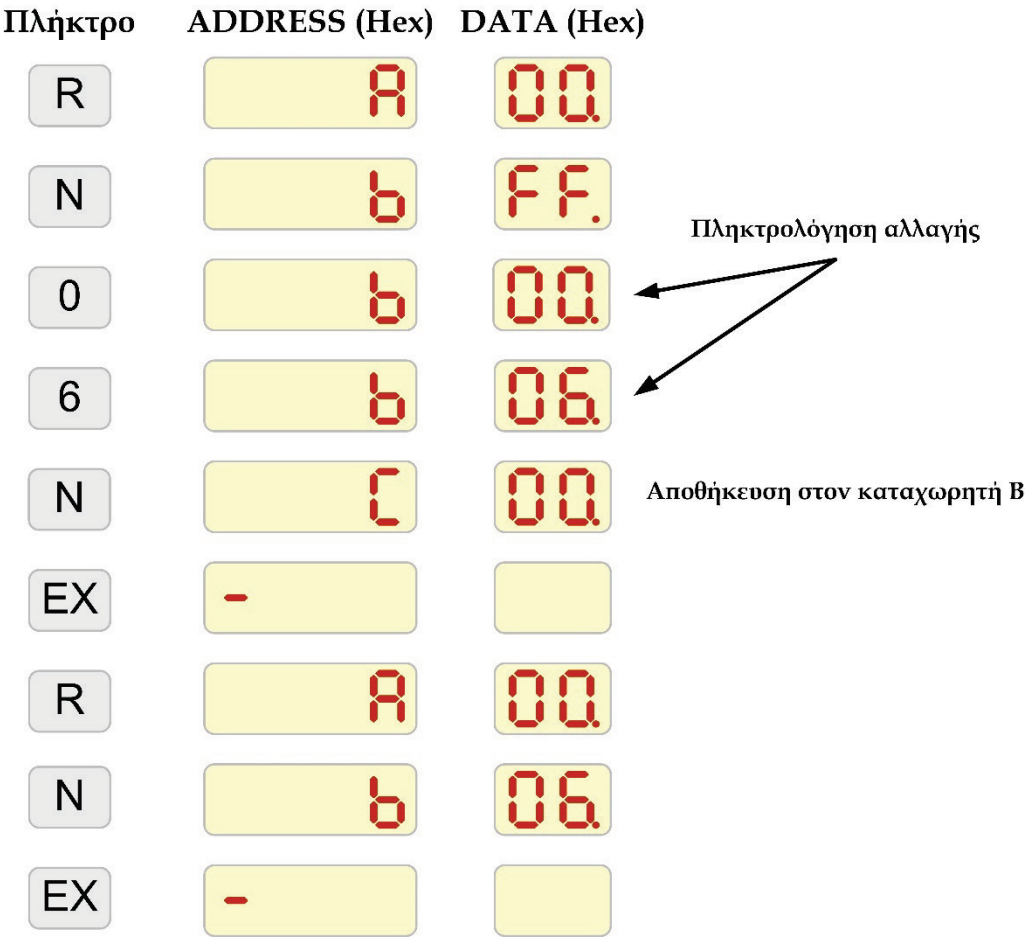
N **Επόμενος καταχωρητής B**

N **Επόμενος καταχωρητής C**

⋮

EX **Τερματισμός εντολής επιστροφή στο *"monitor"* πρόγραμμα**

Για την αλλαγή του περιεχομένου των καταχωρητών χρησιμοποιούμε τα πλήκτρα «R» και «N» προκειμένου να εμφανίσουμε τον καταχωρητή που επιθυμούμε να μεταβάλουμε, στη συνέχεια χρησιμοποιώντας το πληκτρολόγιο αλλάζουμε τιμή, η οποία αποθηκεύεται στο καταχωρητή με το πάτημα του πλήκτρου «N». Στην πιο κάτω εικόνα φαίνεται ένα παράδειγμα εξέτασης και αλλαγής της τιμής του καταχωρητή B.



Εικόνα 3. Εξέταση και αλλαγή περιεχομένου καταχωρητή

Η σειρά με την οποία εμφανίζονται οι καταχωρητές στο display φαίνεται στον πιο κάτω πίνακα.

Πεδίο διευθύνσεων του display	Πλήρης ονομασία καταχωρητή
A	Καταχωρητής A
b	Καταχωρητής B
C	Καταχωρητής C
d	Καταχωρητής D

E	Καταχωρητής E
F	Καταχωρητής των FLAGS
I	INTERRUPT BYTE
H	Καταχωρητής H
L	Καταχωρητής L
SPH	Το περισσότερο σημαντικό byte του 16-bit STACK POINTER
SPL	Το λιγότερο σημαντικό byte του 16-bit STACK POINTER
PCH	Το περισσότερο σημαντικό byte του 16-bit PROGRAM COUNTER
SPL	Το λιγότερο σημαντικό byte του 16-bit PROGRAM COUNTER

Πίνακας 7. Σειρά εμφάνισης καταχωρητών.

Μερικοί από τους πιο πάνω καταχωρητές χρίζουν ιδιαίτερης επεξήγησης διότι δεν είναι φανερός ο τρόπος με τον οποίο το *"monitor"* τους χειρίζεται. Η σημασία των bits του καταχωρητή F (FLAGS) είναι η εξής:

	7	6	5	4	3	2	1	0	
MSB	S	Z	-	AC	-	P	-	C	LSB

S = SIGN BIT
Z = ZERO BIT
AC = AUXILIARY CARRY BIT
P = PARITY BIT
C = CARRY BIT

Η σημασία των bits του INTERRUPT BYTE (I), αμέσως μετά την εκτέλεση κάποιου προγράμματος είναι η εξής:

	7	6	5	4	3	2	1	0	
MSB	SID	0	0	0	IE	M7.5	M6.5	M5.5	LSB

BIT 7: SID (Serial Input Data). Δείχνει την κατάσταση της ακίδας SID του μE8085

BIT 3: IE (Interrupt Enable). Όταν το bit αυτό έχει την τιμή 1 σημαίνει πως οι διακοπές είναι ενεργοποιημένες. Η ενεργοποίηση αυτού του bit γίνεται με εντολή EI (Enable Interrupt) και η απενεργοποίηση με την εντολή DI.

BIT 2-0: M7.5, M6.5, M5.5: Δείχνουν την παρούσα κατάσταση ενεργοποίησης των διακοπών RST7.5, RST6.5, RST5.5. Όταν κάποιο από τα bit αυτά έχει την τιμή 1 σημαίνει πως έχει τεθεί μάσκα στο αντίστοιχο interrupt και εμποδίζεται η εξυπηρέτηση του. Αντίθετα όταν τα bit αυτά έχουν την τιμή 0 και IE bit (BIT 3) έχει την τιμή 1 τότε επιτρέπεται στην CPU να εξυπηρετήσει μια αίτηση διακοπής.

Όταν το *"monitor"* αλλάζει το περιεχόμενο του INTERRUPT BYTE τότε τα bits αυτού αποκτούν την πιο κάτω σημασία.

	7	6	5	4	3	2	1	0	
MSB	SOD	SOE	0	0	IE	M7.5	M6.5	M5.5	LSB

Τα bits 3-0 έχουν την ίδια σημασία με αυτή που αναλύθηκε πιο πάνω και τροποποιώντας τα μπορούμε να ενεργοποιήσουμε ή να απενεργοποιήσουμε όλες τις διακοπές ή καθεμιά χωριστά. Η σημασία των bit 6 και 7 αναλύεται πιο κάτω.

BIT 7: SOD (Serial Output Data). Η κατάσταση του αντιστοιχεί στη στάθμη που τίθεται στην ακίδα εξόδου SOD του μE8085 εφόσον το bit 6 έχει την τιμή 1.

BIT 6: SOE (Serial Output Enable). Όταν το bit αυτό έχει την τιμή 1 τότε το bit 7 εμφανίζεται στην ακίδα εξόδου SOD του μE8085. Όταν έχει την τιμή 0 η έξοδος είναι απενεργοποιημένη.

2.5.5 Χειροκίνητη διακοπή

Το κουμπί «VI» το πληκτρολογίου είναι διασυνδεδεμένο με την ακίδα διακοπής RST 7.5 του μE8085. Στην περίπτωση όπου η συγκεκριμένη διακοπή είναι ενεργοποιημένη και πατηθεί το κουμπί «VI» αυτό θα προκαλέσει την εκτέλεση προγράμματος από τη θέση 003CH δηλαδή από την θέση εκείνη του *"interrupt vector table"* του μE8085 που προβλέπεται για τη διακοπή RST 7.5.

Η θέση 003CH όμως βρίσκεται στη ROM όπου φυλάσσεται και το *"monitor"* πρόγραμμα το οποίο και ανακατευθύνει την διακοπή αυτή στη διεύθυνση 40E3H. Η θέση αυτή βρίσκεται σε περιοχή όπου υπάρχει RAM μνήμη και είναι δυνατή η ανάπτυξη της δικιάς μας ρουτίνας εξυπηρέτησης της διακοπής.

Πρέπει να σημειώσουμε ότι στην περιοχή έχουν δεσμευθεί μόλις 3 θέσεις μνήμης για τον σκοπό αυτό οι οποίες μόλις που επαρκούν για την ανακατεύθυνση της ρουτίνας

εξυπηρέτησης διακοπής σε μια άλλη περιοχή μνήμης όπου υπάρχει διαθέσιμη μνήμη RAM (στην περιοχή από 2000H ως 27FFH).

2.5.6 Χρήσιμες παρατηρήσεις και λειτουργίες

Στις επόμενες γραμμές αναλύονται διάφορες χρήσιμες λειτουργίες του ΜΙΚΡΟΚΙΤ για την ανάπτυξη λογισμικού εφαρμογής με την βοήθεια του, ενώ παράλληλα γίνονται και χρήσιμες παρατηρήσεις πάνω στη λειτουργία του και στη λειτουργία το *"monitor"* προγράμματος.

2.5.6.1 Λειτουργία RESET

Με το πάτημα του πλήκτρου «RS» (RESET) το monitor πρόγραμμα επανεκκινεί από την θέση 0000H και εκτελεί όλες τις διαδικασίες αρχικοποίησης των ολοκληρωμένων 8279, 8155, 8251 αρχικοποιεί τον STACK POINTER (40D0H) και αποθηκεύει την κατάσταση των καταχωρητών του μΕ8085 στη μνήμη RAM (40E6H – 40F2H). Η διαδικασία αυτή ονομάζεται και *"COLD START"* και είναι ισοδύναμη με την εκτέλεση των εντολών JMP 0000H ή RST 0.

2.5.6.2 Λειτουργία WARM START ως "break point"

Η εντολή RST 1 εκτελεί το *"monitor"* πρόγραμμα από το σημείο όπου αποθηκεύεται στη RAM η κατάσταση της CPU, η λειτουργία αυτή ονομάζεται και *"WARM START"*. Το γεγονός αυτό είναι εκμεταλλεύσιμο στην εκσφαλματώση του υπό ανάπτυξη λογισμικού εφαρμογής.

Έτσι η τοποθέτηση μιας εντολής RST 1, στο σημείο του υπό ανάπτυξη κώδικα μπορούμε που επιθυμούμε να εξετάσουμε την κατάσταση της CPU (*"break point"*), έχει ως αποτέλεσμα την διακοπή της εκτέλεσης του προγράμματος εφαρμογής της αποθήκευση της κατάστασης της CPU, την στιγμή που συνέβη η διακοπή, και εκτέλεση του *"monitor"* προγράμματος. Χρησιμοποιώντας τις εντολές του *"monitor"* μπορεί να απεικονιστούν στο display όλες οι τιμές των καταχωρητών και των θέσεων μνήμης που απαιτούνται προκειμένου να γίνει ο έλεγχος του προγράμματος.

2.5.6.3 Τερματισμός προγράμματος

Αν το πρόγραμμα που αναπτύσσεται δεν μπαίνει σε κάποια ατέρμονη διαδικασία σκόπιμα θα πρέπει να τερματίζεται οπωσδήποτε. Αλλιώς η εκτέλεση του προγράμματος θα συνεχιστεί σε περιοχή μνήμης όπου το περιεχόμενο της είναι απροσδιόριστο με αποτέλεσμα και η συμπεριφορά του ΜΙΚΡΟΚΙΤ να είναι επίσης απροσδιόριστη.

Στην περίπτωση όπου το πρόγραμμα εφαρμογής δεν επηρεάζει τη λειτουργία του *"monitor"* ο τερματισμός μπορεί να γίνεται με μια εντολή RST 1. Στην αντίθετη περίπτωση θα πρέπει να γίνεται οπωσδήποτε με μια RST 0 ή JMP 0000H.

Παράρτημα 3 - ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ ΤΟΥ 8085 Virtual Kit

3.1 8085 Virtual Kit

Στα πλαίσια αυτού του κεφαλαίου θα παρουσιάσουμε έναν ιδιαίτερα χρήσιμο και φιλικό στη χρήση εξομοιωτή ο οποίος μπορεί να εξομοιώσει πλήρως την λειτουργία του 8085 αλλά και της εργαστηριακής πλακέτας που χρησιμοποιείται στα πλαίσια του μαθήματος.

Τα τελευταία χρόνια η εξομοιώση έχει γίνει αναπόσπαστο κομμάτι της σχεδίασης, ανάπτυξης και ελέγχου οποιουδήποτε μικρού ή μεγάλου συστήματος. Όλο και περισσότερο γίνεται κατανοητό ότι οποιασδήποτε κατηγορίας σύστημα (μηχανολογικό, ηλεκτρολογικό, ηλεκτρονικό, τηλεπικοινωνιακό, κατασκευαστικό κ.α.) προτού καν αρχίσει να υλοποιείται θα έχει εξομοιωθεί σε μεγάλη λεπτομέρεια και σε όσο γίνεται μεγαλύτερο φάσμα πιθανόν λειτουργικών περιπτώσεων.

Όπως, λοιπόν, είναι σαφές με τον όρο εξομοιωτή αναφερόμαστε σε ένα λογισμικό (πρόγραμμα) το οποίο μπορεί να εκτελεστεί σε έναν υπολογιστή (ανάλογα και με την πολυπλοκότητα του εξομοιούμενου συστήματος ποικίλουν και οι απαιτήσεις από τον συγκεκριμένο υπολογιστή) και να εξομοιώσει κάθε πιθανή λειτουργία του εκάστοτε συστήματος. Ένα τέτοιο πρόγραμμα παρουσιάζεται εδώ και παρέχεται δωρεάν στους φοιτητές του εργαστηρίου. Τα πλεονεκτήματα που παρέχονται στον φοιτητή μέσα από την χρήση του εν λόγω εξομοιωτή είναι πολλά και ιδιαιτέρως σημαντικά.

- Εξάσκηση στην ανάπτυξη προγράμματος για τον 8085 και στη λειτουργία της εργαστηριακής πλακέτας χρησιμοποιώντας μονό έναν προσωπικό υπολογιστή. Οι απαιτήσεις του εξομοιωτικού προγράμματος είναι ιδιαιτέρως χαμηλές και άρα μπορεί να εκτελεστεί σε σχεδόν κάθε υπολογιστικό σύστημα αρκεί να τρέχει το λειτουργικό σύστημα windows.
- Πειραματισμός και εύρεση απαντήσεων σε απορίες που δεν διασαφηνίστηκαν πλήρως στην ώρα του εργαστηρίου ή προέκυψαν μετά από αυτό.
- Προετοιμασία για εξετάσεις είτε σε επίπεδο προόδου είτε τελικές.
- Προετοιμασία για την επόμενη άσκηση ώστε η ώρα του εργαστηρίου να είναι πιο εποικοδομητική, ουσιαστική και χρήσιμη.
- Το συγκεκριμένο πρόγραμμα παρέχει ένα ιδιαίτερα φιλικό γραφικό περιβάλλον, όμοιο σχεδόν με την εργαστηριακή πλακέτα που έχετε γνωρίσει και δεν χρειάζεται καν εγκατάσταση.

Στις επόμενες γραμμές γίνεται μια συνοπτική παρουσίαση των δυνατοτήτων και των κυριότερων λειτουργιών του εξομοιωτή.

Στην παρακάτω εικόνα φαίνεται το γραφικό περιβάλλον του προγράμματος στο οποίο έχουμε δημιουργήσει μια νέα εργασία στα πλαίσια της οποία έχουμε γράψει ένα πολύ απλό πρόγραμμα αποτελούμενο από εντολές που είναι οικίες στον φοιτητή στο σημείο αυτό.

Μια νέα εργασία μπορεί να δημιουργηθεί από το μενού **File → New** ή μια ήδη υπάρχουσα να ανοιχθεί από το μενού **File → Open** με διαδικασίες, κοινές σε όλα τα προγράμματα των Windows.

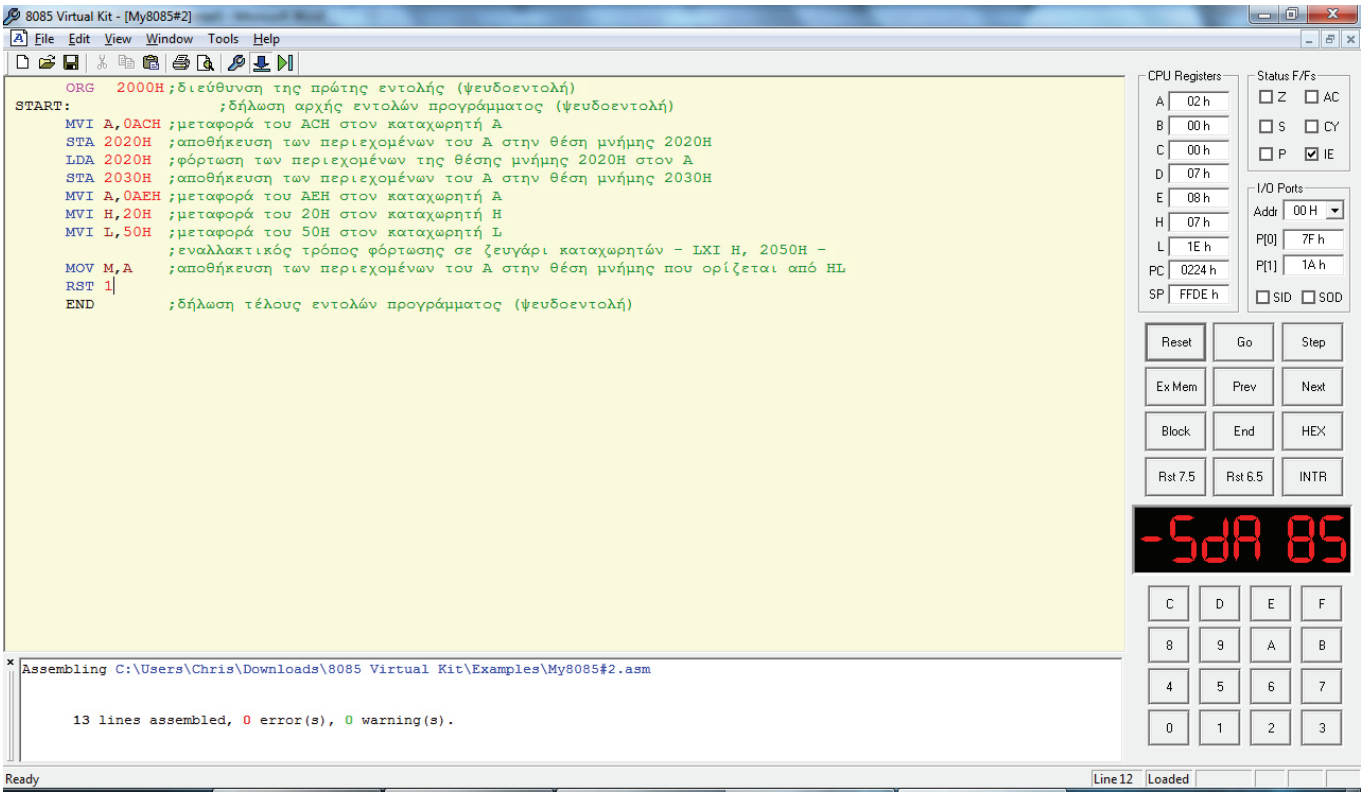
Η εισαγωγή των εντολών ακολουθεί συγκεκριμένη χρωματική επισήμανση (αν το κείμενο είναι εντολή, αριθμητική ποσότητα, σχόλιο κτλ.) , ιδιαίτερα χρήσιμη στον γρήγορο εντοπισμό ορθογραφικών και συντακτικών λαθών.

Στο δεξί πάνω μέρος το πρόγραμμα μας εμφανίζει τις τιμές που έχουν ανά πάσα στιγμή οι καταχωρητές (γενικού και ειδικό σκοπού) και τα περιεχόμενα κάθε μιας εκ των 256 διευθύνσεων εισόδου/εξόδου.

Ακριβώς από κάτω δίνονται τα πλήκτρα ελέγχουν των λειτουργιών του monitor προγράμματος η λειτουργία των οποίων είναι σχεδόν ολόιδια με το Monitor πρόγραμμα της εργαστηριακής πλακέτας.

Ακριβώς από κάτω έχουμε την οθόνης με λειτουργία όπως και στην πλακέτα. Δηλαδή στο αριστερό μέρος φαίνεται η διεύθυνση που έχουμε διαλέξει και στο δεξί το περιεχόμενό της σε δεκαεξαδική μορφή.

Τέλος ακριβώς από κάτω βλέπουμε το δεκαεξαδικό πληκτρολόγιο όμοιο με αυτό της πλακέτας το οποίο μας δίνει τη δυνατότητα να επιλέγουμε διευθύνσεις ή/και να τροποποιούμε τα περιεχόμενα τους.



Αφού εξοικειωθούμε με τα μέρη του γραφικού περιβάλλοντος και γράψουμε τις εντολές του κώδικά μας το επόμενο βήμα είναι να εκτελέσουμε της βασικές λειτουργίες του εξομοιωτικού προγράμματος, αυτές είναι:

- Δημιουργία του εκτελέσιμου αρχείου: Ως γνωστό οι εντολές που εισάγουμε πρέπει να μεταφραστούν σε δυαδικό κώδικα που καταλαβαίνουν και μπορούν να εκτελέσουν οι μικροεπεξεργαστές. Η λειτουργία αυτή ενεργοποιείται μέσα από το μενού **Tools → Assemble**. Τα πιθανά λάθη φαίνονται στο κάτω αριστερά λευκό πλαίσιο. Στην συγκεκριμένη περίπτωση μεταφραστήκαν 13 γραμμές και δεν υπήρξαν λάθη ή προειδοποιήσεις.

- "Κατέβασμα" του εκτελέσιμου στο εικονικό επεξεργαστή και εκτέλεση αυτού: Αφού τελειώσει η μετάφραση χωρίς λάθη μέσα από το μενού **Tools → Load/Unload OBJ** εκτελούμε εικονική μεταφορά του εκτελέσιμου στον επεξεργαστή. Κατόπιν η εκτέλεση γίνεται μέσα από το monitor πρόγραμμα και τα αντίστοιχα πλήκτρα όπως θα γινόταν και στην πραγματική πλακέτα. Φυσικά μετά την εκτέλεση του προγράμματος μπορούμε να εξετάσουμε και να τροποποιήσουμε τα περιεχόμενα της μνήμης όπως θα κάναμε και στην πραγματική πλακέτα του εργαστηρίου. Συγκεκριμένα πρέπει να γίνουν τα ακόλουθα βήματα

    - Πάτημα του πλήκτρου **GO**
    - Εισαγωγή της αρχικής διεύθυνσης του κώδικα, στη συγκεκριμένη περίπτωση **2000**
    - Πάτημα του πλήκτρου **NEXT**

- Εξέταση-Παρακολούθηση Λειτουργίας Προγράμματος (Trace Program): Η συγκεκριμένη λειτουργία ενσωματώνει το περιβάλλον εξομοίωσης με τον κώδικα που έχουμε γράψει και έτσι έχουμε τη δυνατότητα να "εκτελέσουμε" το πρόγραμμα με βάση τις εντολές που έχουμε γράψει στη γλώσσα assembly. Η λειτουργία αυτή ενεργοποιείται μέσα από το μενού **Tools → Trace Program**. Με την ενεργοποίηση της λειτουργίας εμφανίζεται η βοηθητική μπάρα Trace Bar (φαίνεται στην επόμενη εικόνα) η οποία παρέχει (με ανάλογα κουμπιά) τα διαφορετικά είδη εκτέλεσης του προγράμματος σε σχέση με τον κώδικα που έχουμε γράψει. Παράλληλα παρατηρήστε ότι με ειδική σήμανση (μαύρισμα γραμμής) το πρόγραμμα μας δείχνει ανά πάσα στιγμή ποια είναι η επόμενη προς

εκτέλεση εντολή. Για παράδειγμα με διαδοχικά πατήματα του αριστερού πλήκτρου το οποίο αντιστοιχεί στη εντολή προς εντολή εκτέλεση του προγράμματος (και η οποία είναι η πιο χρήσιμη λειτουργία στη παρούσα φάση για τον φοιτητή του εργαστηρίου) παρατηρούμε την εκτέλεση κάθε μιας από τις εντολές assembly που έχουμε γράψει και παράλληλα μπορούμε να έχουμε εποπτεία των δεδομένων όλων των διαθέσιμων καταχωρητών. Όταν η λειτουργία απενεργοποιηθεί (πάλι μέσα από το μενού **Tools → Trace Program)** ο χρήστης μπορεί να εξετάσει και να τροποποιήσει τα δεδομένα της μνήμης όπως θα έκανε και στην πραγματική εργαστηριακή πλακέτα.

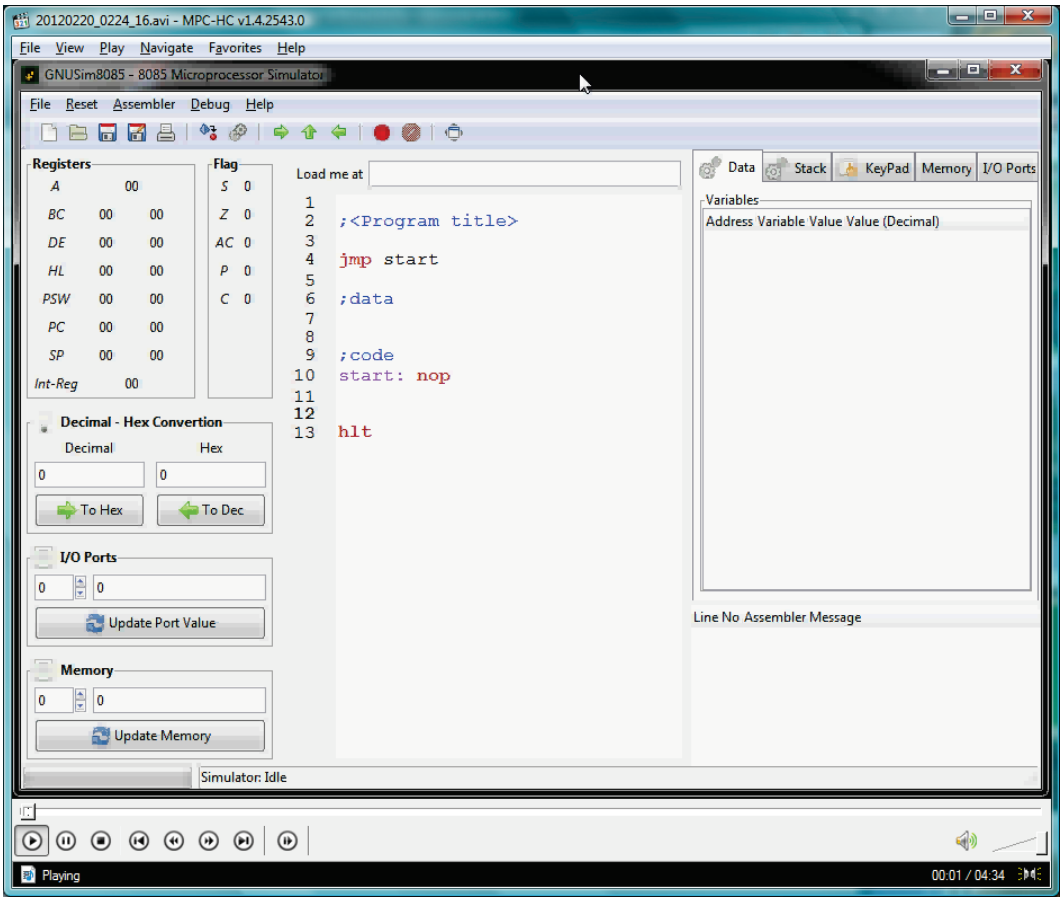
Περισσότερες λεπτομέρειες για τις δυνατότητες και λειτουργίες του συγκεκριμένου εξομοιωτή μπορείτε να βρείτε στη βοήθεια του προγράμματος (μενού HELP) αλλά και μέσα από ερωτήσεις κατά τη διάρκεια των εργαστηρίων.

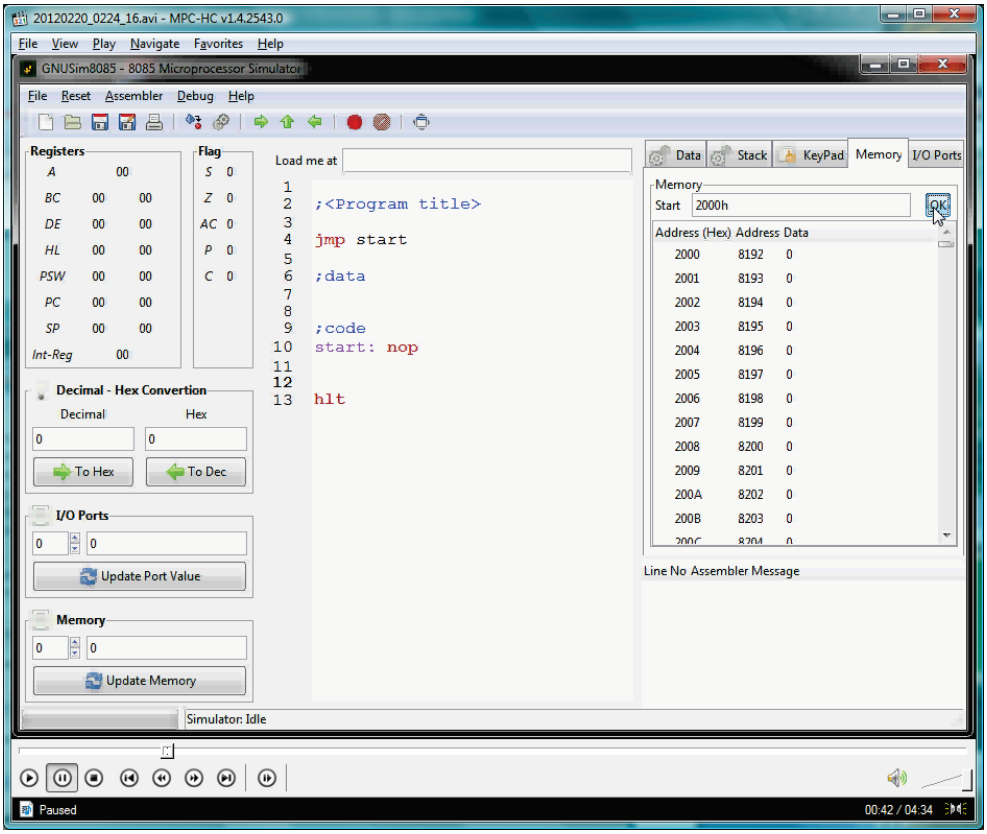
Παράρτημα 4 - ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ ΤΟΥ gnusim8085

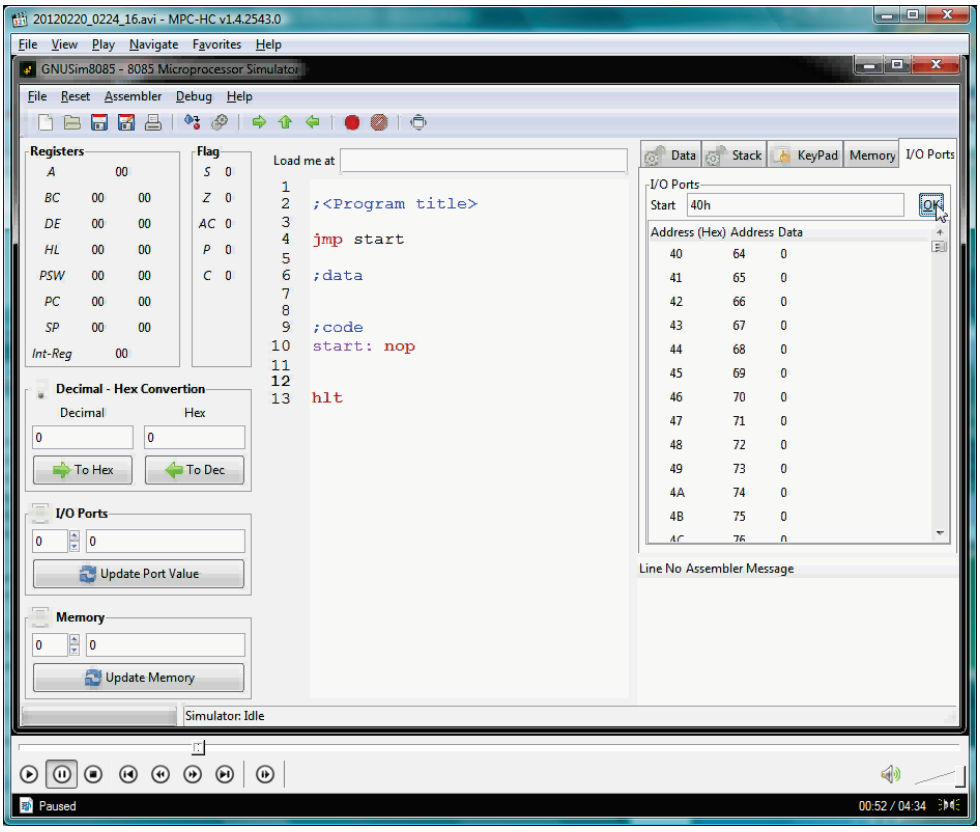
4.1 gnusim8085

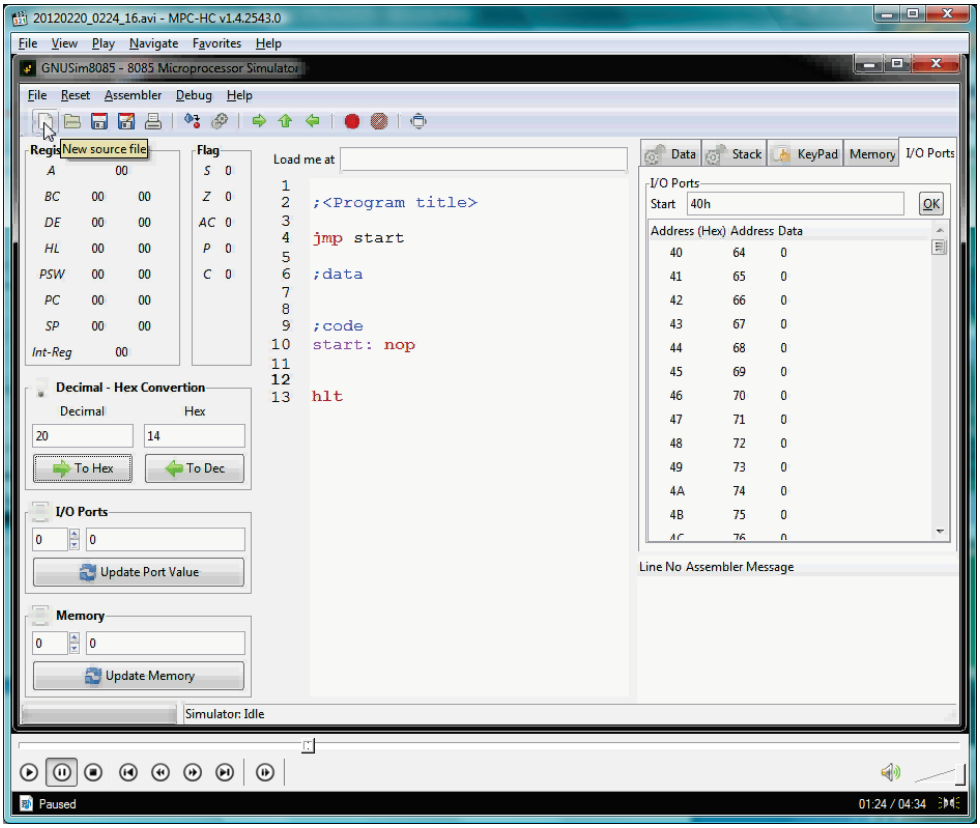
Κατεβάστε σχετικό video χρήσης από την σελίδα του μαθήματος στο eclass

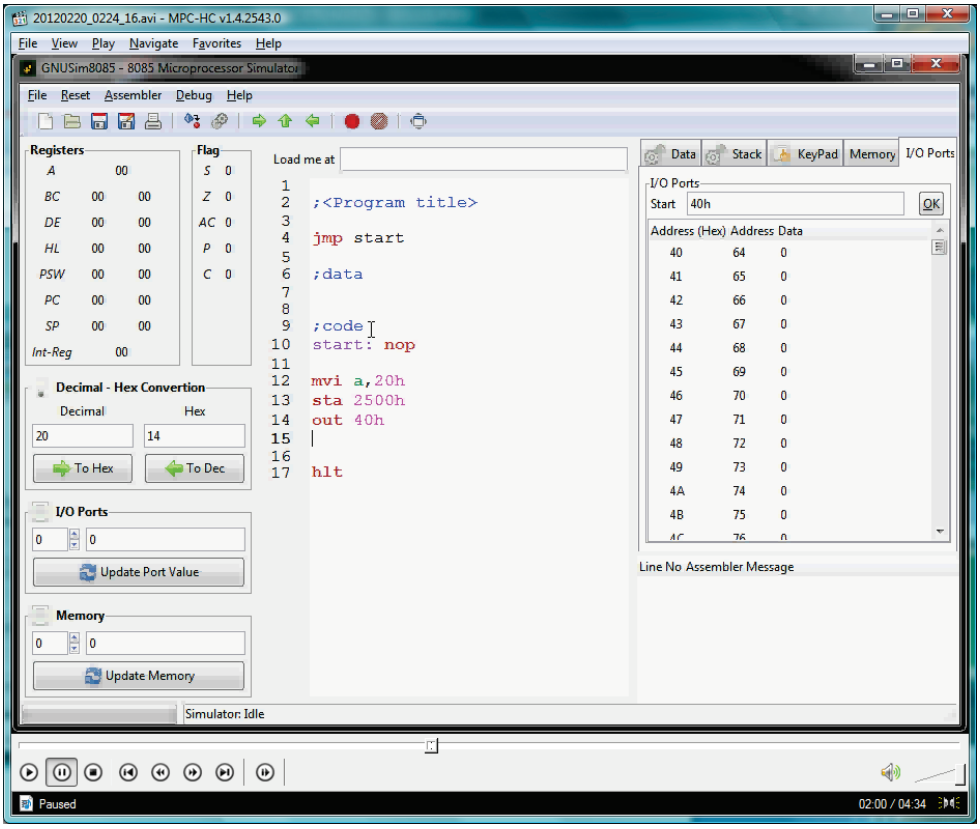
http://eclass.teipat.gr/eclass/modules/document/play.php/487136/Εργαστήριο/GNUSim8085.avi

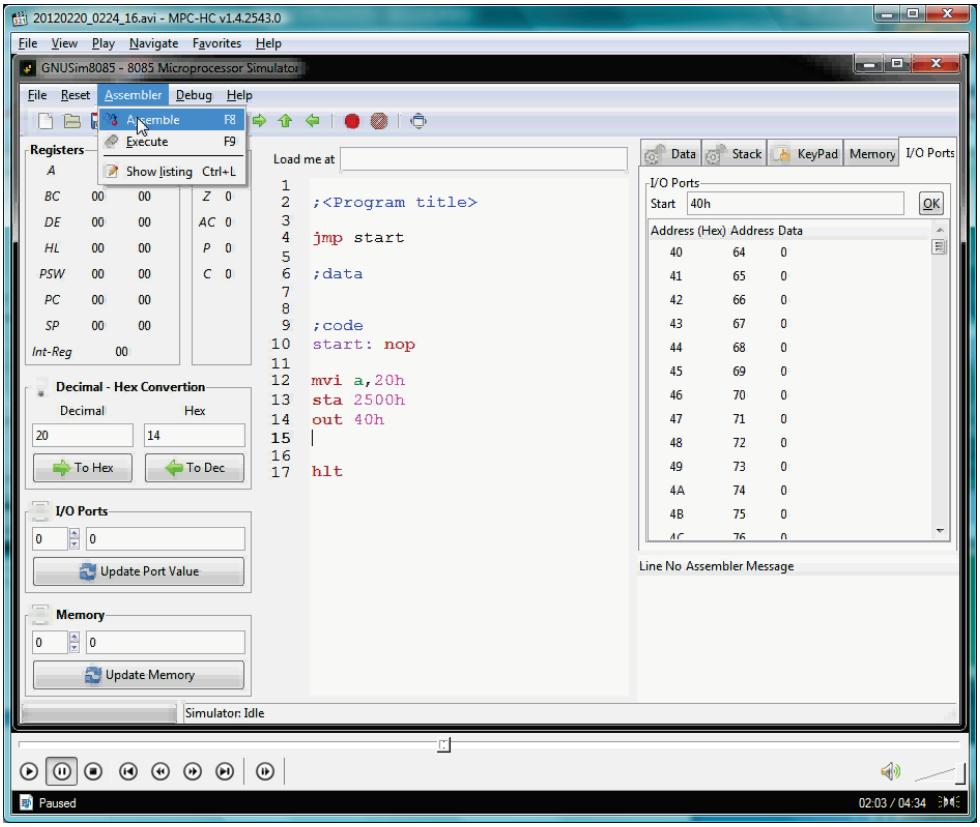


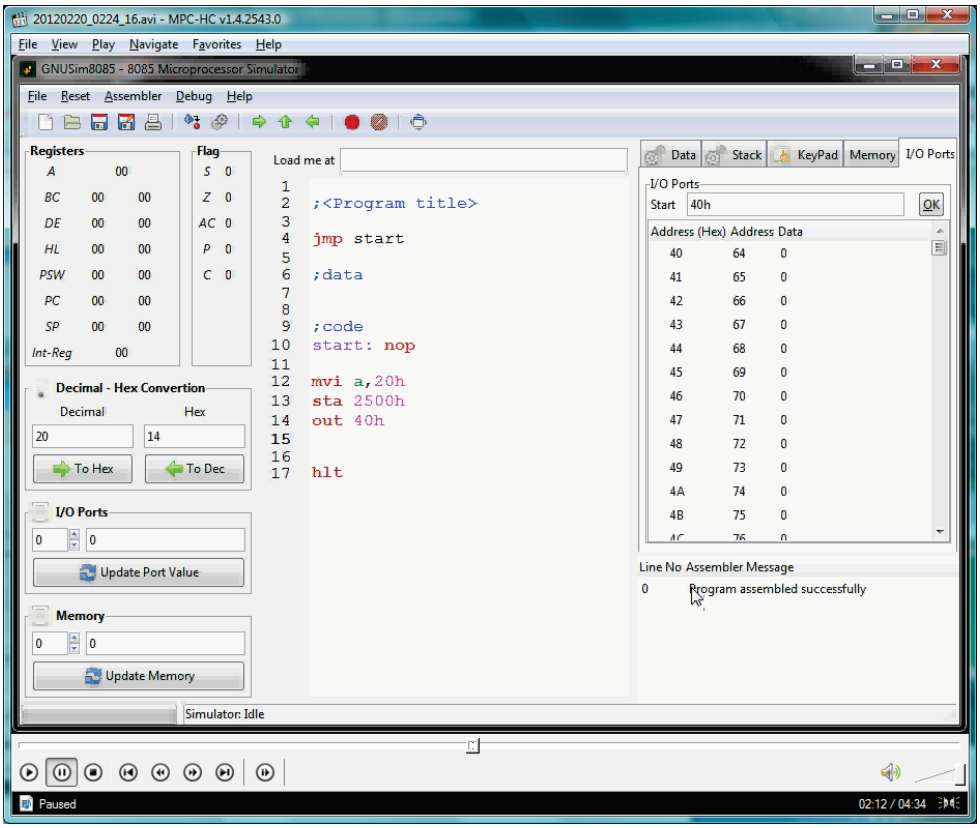

20120220_0224_16.avi - MPC-HC v1.4.2543.0
File View Play Navigate Favorites Help
GNUSim8085 - 8085 Microprocessor Simulator
File Reset Assembler Debug Help
Registers
Flag
Load me at
;<Program title>
jmp start
;data
;code
start: nop
hlt
Data Stack KeyPad Memory I/O Ports
Memory
Start 2000h
Address (Hex) Address Data
Decimal - Hex Convertion
Decimal Hex
To Hex To Dec
I/O Ports
Update Port Value
Memory
Update Memory
Line No Assembler Message
Simulator: Idle
Paused
00:42 / 04:34

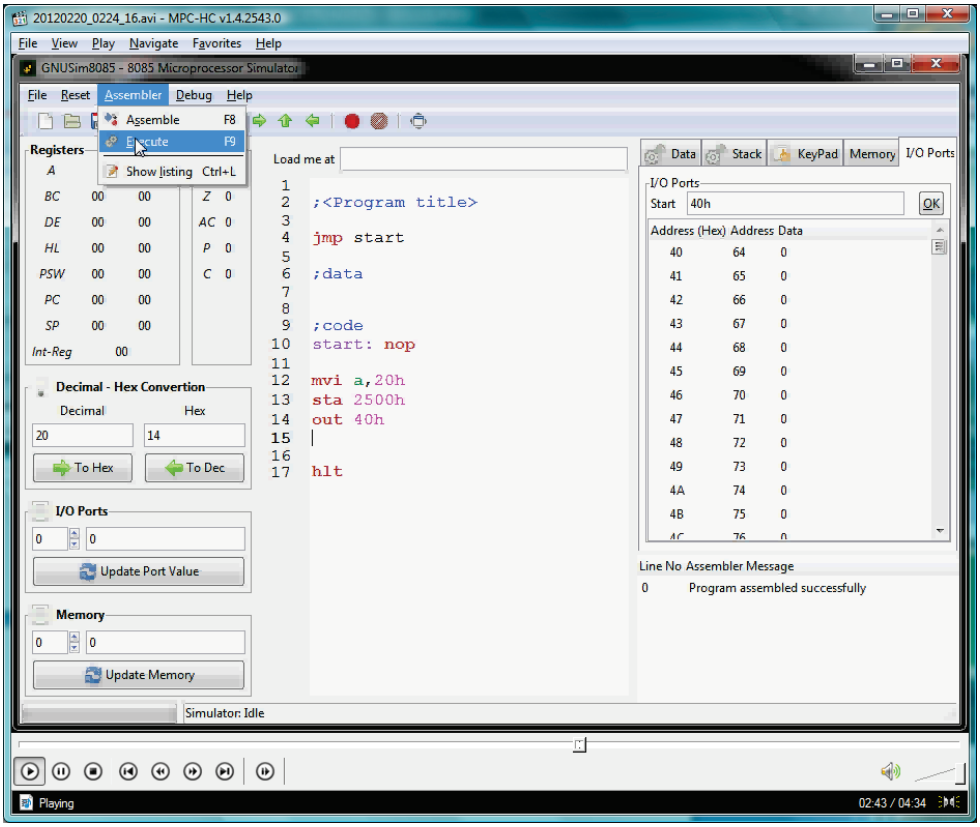

20120220_0224_16.avi - MPC-HC v1.4.2543.0
File View Play Navigate Favorites Help
GNUSim8085 - 8085 Microprocessor Simulator
File Reset Assembler Debug Help
Registers
Flag
Load me at
;<Program title>
jmp start
;data
;code
start: nop
hlt
Data Stack KeyPad Memory I/O Ports
I/O Ports
Start 40h
Address (Hex) Address Data
Decimal - Hex Convertion
Decimal Hex
To Hex To Dec
I/O Ports
Update Port Value
Memory
Update Memory
Line No Assembler Message
Simulator: Idle
Paused
00:52 / 04:34

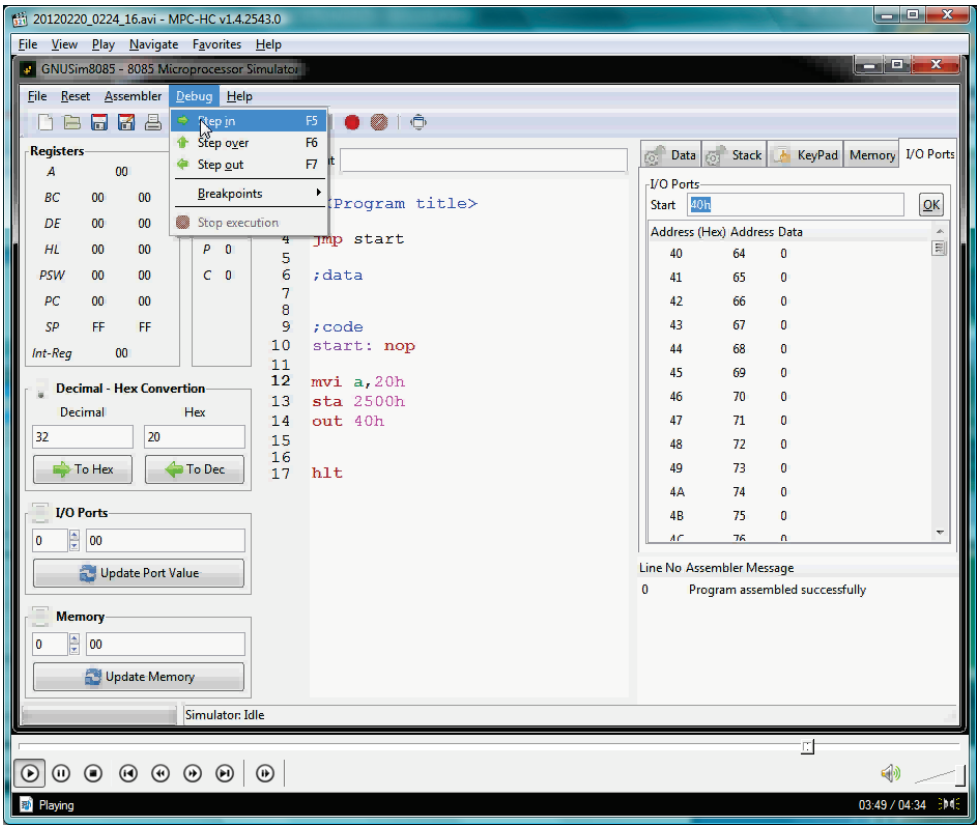

20120220_0224_16.avi - MPC-HC v1.4.2543.0
File View Play Navigate Favorites Help
GNUSim8085 - 8085 Microprocessor Simulator
File Reset Assembler Debug Help
New source file
Registers
Flag
Load me at
1 ;<Program title>
3 jmp start
5 ;data
7
8
9 ;code
10 start: nop
11
12
13 hlt
Data Stack KeyPad Memory I/O Ports
I/O Ports
Start 40h OK
Address (Hex) Address Data
40 64 0
41 65 0
42 66 0
43 67 0
44 68 0
45 69 0
46 70 0
47 71 0
48 72 0
49 73 0
4A 74 0
4B 75 0
Line No Assembler Message
Decimal - Hex Convertion
Decimal
Hex
20
14
To Hex
To Dec
I/O Ports
Update Port Value
Memory
Update Memory
Simulator: Idle
Paused
01:24 / 04:34


20120220_0224_16.avi - MPC-HC v1.4.2543.0
File View Play Navigate Favorites Help
GNUSim8085 - 8085 Microprocessor Simulator
File Reset Assembler Debug Help
Registers
Flag
Load me at
1 ;<Program title>
3 jmp start
5 ;data
7
8
9 ;code
10 start: nop
11
12 mvi a,20h
13 sta 2500h
14 out 40h
15
16
17 hlt
Data Stack KeyPad Memory I/O Ports
I/O Ports
Start 40h OK
Address (Hex) Address Data
40 64 0
41 65 0
42 66 0
43 67 0
44 68 0
45 69 0
46 70 0
47 71 0
48 72 0
49 73 0
4A 74 0
4B 75 0
Line No Assembler Message
Decimal - Hex Convertion
Decimal
Hex
20
14
To Hex
To Dec
I/O Ports
Update Port Value
Memory
Update Memory
Simulator: Idle
Paused
02:00 / 04:34


20120220_0224_16.avi - MPC-HC v1.4.2543.0
File View Play Navigate Favorites Help
GNUSim8085 - 8085 Microprocessor Simulator
File Reset Assembler Debug Help
Assemble F8
Execute F9
Show listing Ctrl+L
Registers
Decimal - Hex Convertion
Decimal
Hex
20
14
To Hex
To Dec
I/O Ports
Update Port Value
Memory
Update Memory
Load me at
;<Program title>
jmp start
;data
;code
start: nop
mvi a,20h
sta 2500h
out 40h
hlt
Data Stack KeyPad Memory I/O Ports
I/O Ports
Start 40h
OK
Address (Hex) Address Data
Line No Assembler Message
Simulator: Idle
Paused
02:03 / 04:34


20120220_0224_16.avi - MPC-HC v1.4.2543.0
File View Play Navigate Favorites Help
GNUSim8085 - 8085 Microprocessor Simulator
File Reset Assembler Debug Help
Registers
Flag
Decimal - Hex Convertion
Decimal
Hex
20
14
To Hex
To Dec
I/O Ports
Update Port Value
Memory
Update Memory
Load me at
;<Program title>
jmp start
;data
;code
start: nop
mvi a,20h
sta 2500h
out 40h
hlt
Data Stack KeyPad Memory I/O Ports
I/O Ports
Start 40h
OK
Address (Hex) Address Data
Line No Assembler Message
0 Program assembled successfully
Simulator: Idle
Paused
02:12 / 04:34


20120220_0224_16.avi - MPC-HC v1.4.2543.0
File View Play Navigate Favorites Help
GNUSim8085 - 8085 Microprocessor Simulator
File Reset Assembler Debug Help
Assemble F8
Execute F9
Show listing Ctrl+L
Registers
Decimal - Hex Convertion
Decimal
Hex
20
14
To Hex
To Dec
I/O Ports
Update Port Value
Memory
Update Memory
Load me at
;<Program title>
jmp start
;data
;code
start: nop
mvi a,20h
sta 2500h
out 40h
hlt
Data Stack KeyPad Memory I/O Ports
I/O Ports
Start 40h
OK
Address (Hex) Address Data
Line No Assembler Message
0 Program assembled successfully
Simulator: Idle
Playing
02:43 / 04:34


20120220_0224_16.avi - MPC-HC v1.4.2543.0
File View Play Navigate Favorites Help
GNUSim8085 - 8085 Microprocessor Simulator
File Reset Assembler Debug Help
Step in F5
Step over F6
Step out F7
Breakpoints
Stop execution
Registers
Decimal - Hex Convertion
Decimal
Hex
32
20
To Hex
To Dec
I/O Ports
Update Port Value
Memory
Update Memory
;Program title>
jmp start
;data
;code
start: nop
mvi a,20h
sta 2500h
out 40h
hlt
Data Stack KeyPad Memory I/O Ports
I/O Ports
Start 40h
OK
Address (Hex) Address Data
Line No Assembler Message
0 Program assembled successfully
Simulator: Idle
Playing
03:49 / 04:34

ISBN 978-618-00-5292-3