ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Αρχιτεκτονική Υπολογιστών

Εργαστήριο

Ενότητα: ΠΑΡΑΔΕΙΓΜΑ ΑΠΟΣΦΑΛΜΑΤΩΣΗΣ

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

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

Άδειες Χρήσης

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

Χρηματοδότηση

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

Περιεχόμενα

1. Σκοπός της ενότητας 4
2. Εργαλεία στον emulator 4
3. Παράδειγμα 5

Πίνακας εικόνων

Εικόνα 1: single step και step back 4

Εικόνα 2: Μενού debug του emulator 4

Εικόνα 3: Emulator Screen 5

Εικόνα 4: Αρχική οθόνη του emu8086 6

Εικόνα 5: Ο κώδικας της άσκησης 6

Εικόνα 6: Το παράθυρο του emulator με "φωτισμένα" τα αντίστοιχα bytes 7

Εικόνα 7: Τα περιεχόμενα των καταχωρητών 8

Εικόνα 8: Το κουμπί vars του emulator 8

Εικόνα 9: Το παράθυρο με τις τιμές των μεταβλητών 8

Εικόνα 10: Step back 2 φορές 9

Εικόνα 11: RUN UNTIL 10

Εικόνα 12: Single step 10

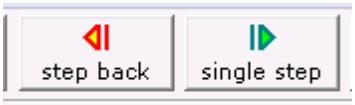
Εικόνα 13: Εκτέλεση της επόμενης εντολής 10

1. Σκοπός της ενότητας

Από τα πιο σημαντικά θέματα στον προγραμματισμό αποτελεί το θέμα της αποσφαλμάτωσης ενός προγράμματος. Σε αυτό το κείμενο θα γνωρίσουμε τα βασικά εργαλεία αποσφαλμάτωσης που μας παρέχει το emu8086.

2. Εργαλεία στον emulator

Εργαλεία στον emulator **:**

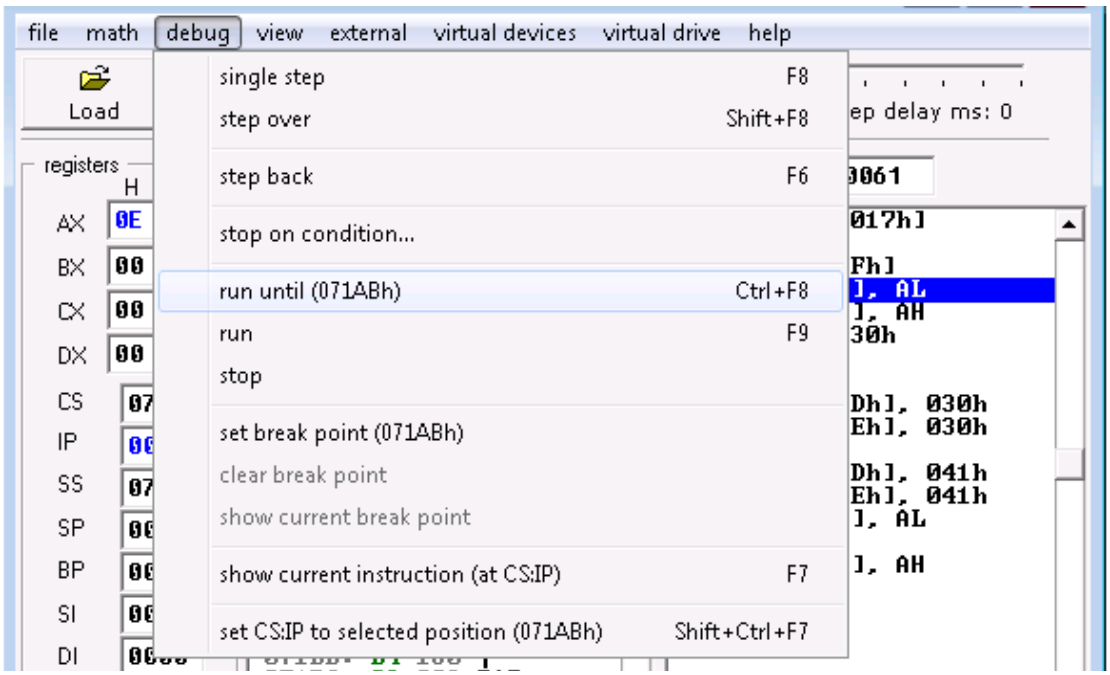


Εικόνα 1: single step και step back

- Αν πατάμε **[single step]** τότε εκτελείται μια μόνο εντολή και βλέπουμε το αποτέλεσμα στους καταχωρητές ή στην οθόνη.

- Αν πατήσουμε **[step back]** τότε αναιρείται η εκτέλεση και το πρόγραμμα πηγαίνει μια εντολή πίσω.

Από το μενού debug μπορούμε να έχουμε πρόσβαση στις εξής εντολές:



Εικόνα 2: Μενού debug του emulator

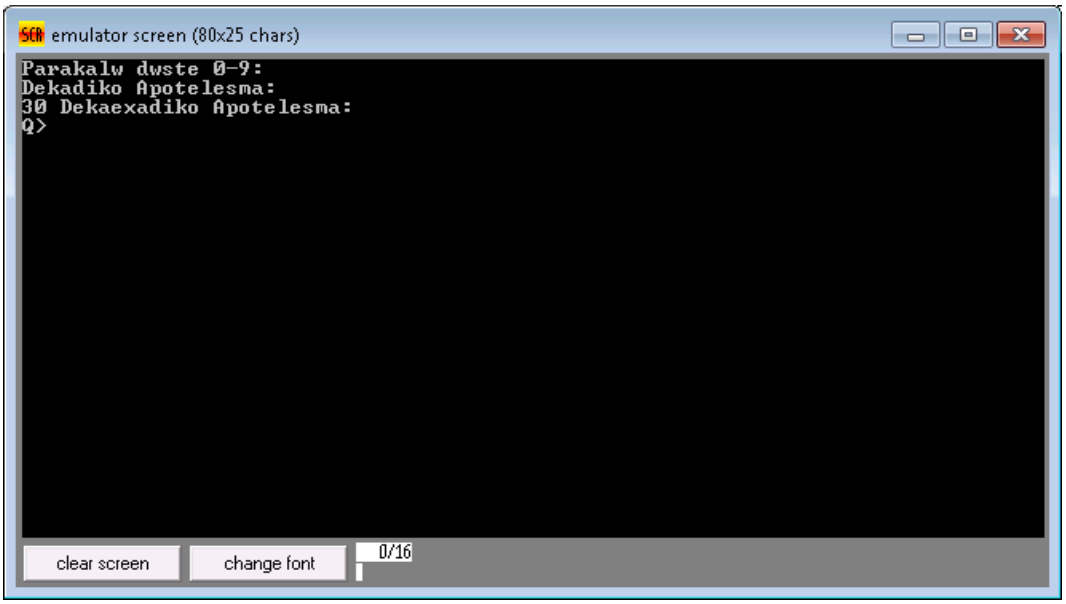
- **[single step]:** εκτέλεση της επόμενης εντολής.
- **[step over]:** εκτέλεση μιας συνάρτησης χωρίς να μπούμε μέσα στον κώδικα της συνάρτησης.
- **[run until]:** εκτέλεση μέχρι το συγκεκριμένο σημείο στο κώδικα που έχουμε επιλέξει στο παράθυρο κώδικα.
- **[set breakpoint]:** τοποθέτηση σημείου παύσης κάθε φορά που ο επεξεργαστής θα φτάνει σε αυτή την εντολή που έχουμε επιλέξει στο παράθυρο κώδικα.

Οι υπόλοιπες εντολές αποσφαλμάτωσης είναι για πιο προχωρημένες λειτουργίες.

3. Παράδειγμα

Ακολουθεί ένα παράδειγμα χρήσης των ανωτέρω εργαλείων.

Έχουμε ένα πρόγραμμα το οποίο υπολειτουργεί. Συγκεκριμένα αντί να υπολογίσει για N=5 (N^2+N) τις τιμές 30 και 1E, εκτυπώνει, 30 και Q> όπως παρακάτω:

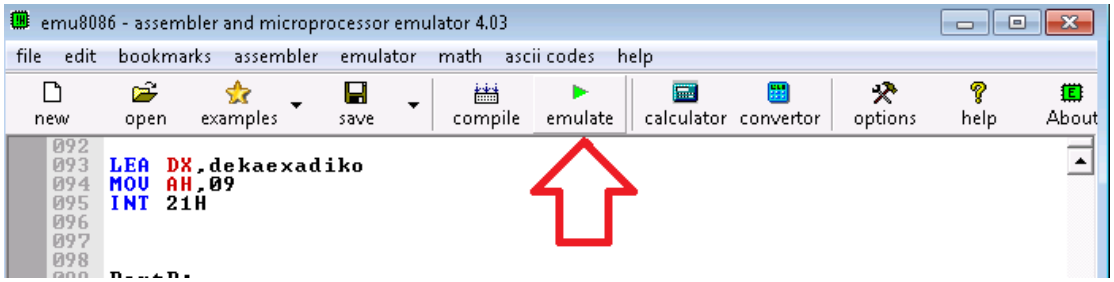


Εικόνα 3: Emulator Screen

Από ότι φαίνεται το πρόβλημα βρίσκεται στο τμήμα υπολογισμού του δεκαεξαδικού αποτελέσματος, αφού εκτυπώνεται σωστά το δεκαδικό μήνυμα.

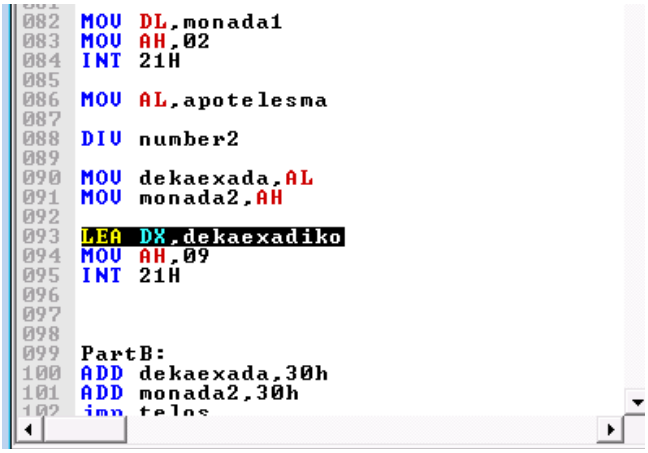
Για αυτό το λόγο απαιτείται αποσφαλμάτωση.

Πατάμε το κουμπί emulate από την αρχική οθόνη.



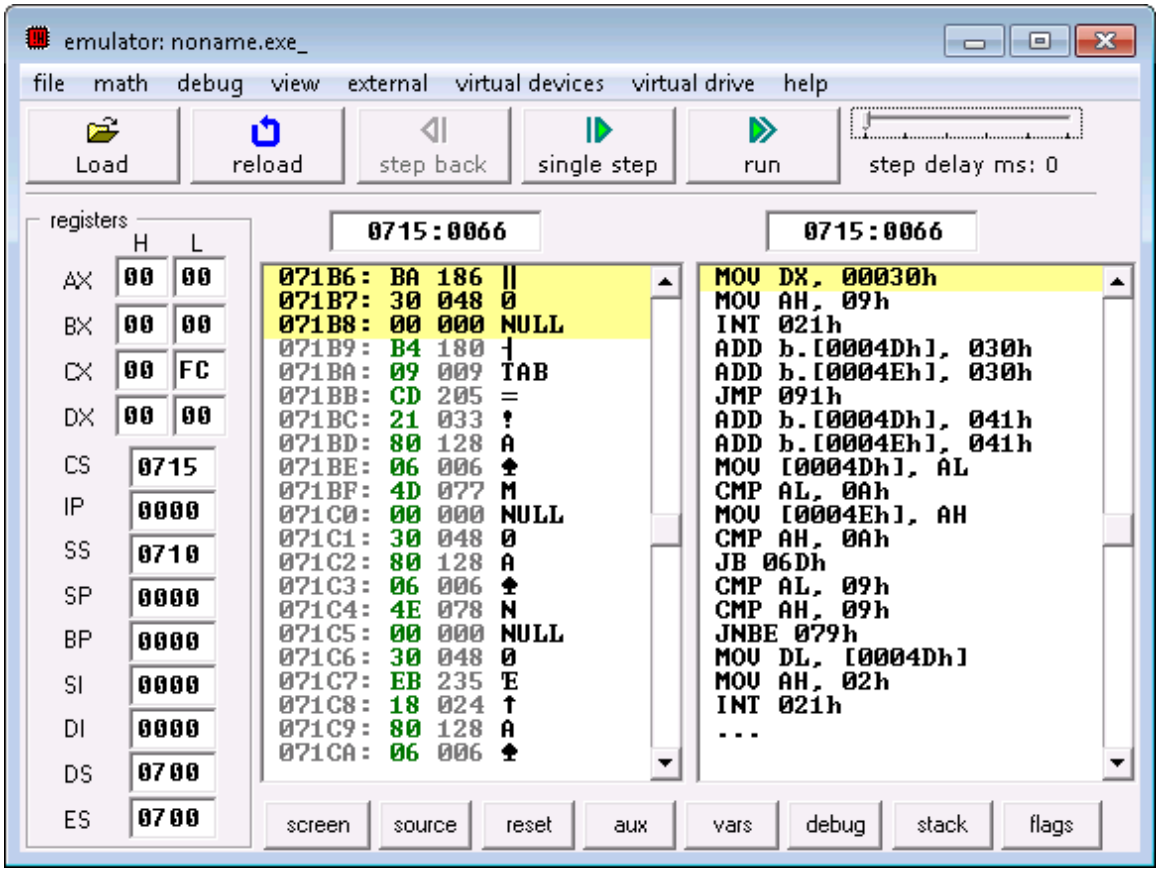
Εικόνα 4: Αρχική οθόνη του emu8086

Βρίσκουμε το σημείο του κώδικα που είναι αμέσως μετά την εμφάνιση του string δεκαεξαδικό αποτέλεσμα και πατάμε κλικ στο παράθυρο του κώδικα. Επιλέγουμε τη γραμμή με το **`LEA DX,dekaexadiko`**



Εικόνα 5: Ο κώδικας της άσκησης

Μόλις πατήσουμε πάνω στην αντίστοιχη γραμμή, τότε στο παράθυρο emulator φωτίζονται τα Bytes που αντιστοιχούν σε αυτή τη γραμμή:



Εικόνα 6: Το παράθυρο του emulator με "φωτισμένα" τα αντίστοιχα bytes

Σε αυτό το παράδειγμα, βλέπουμε ότι η γραμμή **`LEA DX,dekaexadiko`** σχετίζεται με τα 3 Byte:

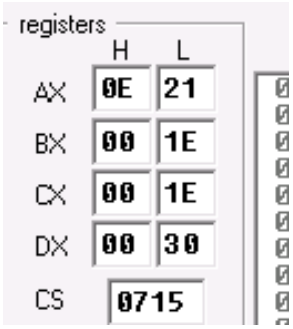
- **BA** *(βρίσκεται στη διεύθυνση μνήμης 071B6)*
- **30** *(βρίσκεται στη διεύθυνση μνήμης 071B7)* και
- **00** *(βρίσκεται στη διεύθυνση μνήμης 071B8).*

Επίσης, δίπλα στα Bytes φαίνεται εκτός από τη δεκαεξαδική τους απεικόνιση, η δεκαδική απεικόνιση και η απεικόνιση ASCII.

Το επόμενο βήμα είναι από το μενού **debug** να πατήσουμε το **[run until]**. Έτσι, γίνεται η εκτέλεση έως τη γραμμή που βρισκόμαστε.

Στο παράθυρο που θα μας ζητηθεί είσοδος χαρακτήρα, πατάμε 5.

Παρατηρούμε ότι στο αριστερό παράθυρο του emulator που φαίνονται οι καταχωρητές, ο **AH** έχει την τιμή 0E και ο **AL** την τιμή 21.



Εικόνα 7: Τα περιεχόμενα των καταχωρητών

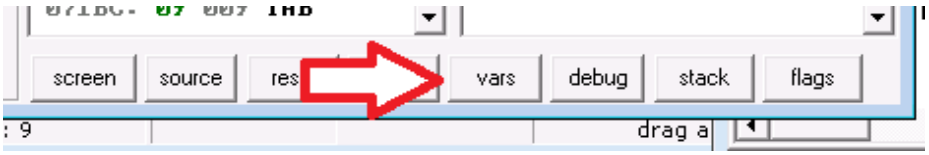
Παρατηρώντας τον κώδικα *(δες εικόνα με κώδικα λίγα σχήματα παραπάνω)* βλέπουμε ότι σε αυτό το σημείο έχουμε ήδη τοποθετήσει στις θέσεις μνήμης `dekaexada` και `monada2` το αποτέλεσμα που έχει προκύψει από τη διαίρεση 30 δια 16.

(γραμμή 90: `MOV dekaexada,AL`)

(γραμμή 91: `MOV monada2,AH`)

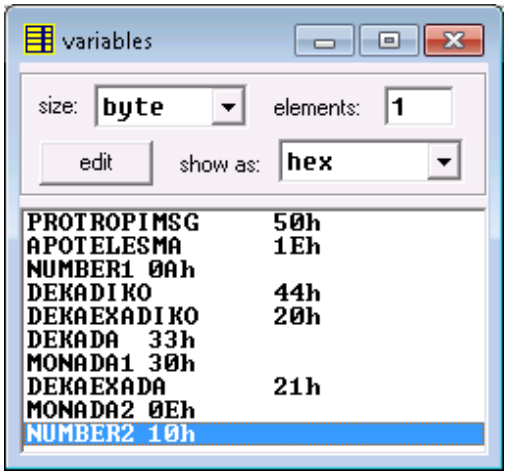
Δηλαδή, αν γίνονταν σωστά η διαίρεση 30h/16 η δεκαεξάδα θα πρέπει να έχει την τιμή 1 *(1 δεκαξάδα)* και η μονάδα την τιμή 0E *(14 μονάδες)*. Συνολικά 16+14=30.

Πατάμε το κουμπί **VARS** που βρίσκεται στο παράθυρο του emulator.



Εικόνα 8: Το κουμπί vars του emulator

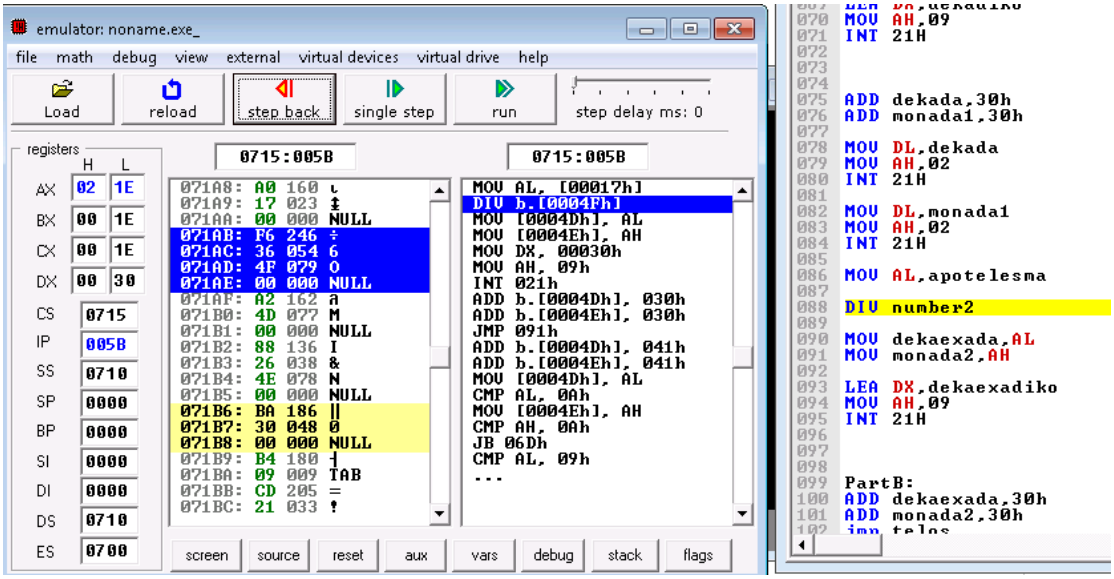
Και εμφανίζεται το παρακάτω παράθυρο:



Εικόνα 9: Το παράθυρο με τις τιμές των μεταβλητών

Δυστυχώς το `dekaexada` έχει τιμή 21h ενώ το `monada2` τιμή **0Eh**. Δηλαδή, η διαίρεση δεν έχει γίνει σωστά.

Πατάμε στο emulator **[step back]**, 2 φορές μέχρι να φτάσουμε στην διαίρεση *(div number2)*. Εκεί παρατηρούμε ότι το number2 είναι Byte και έχει τιμή 10h. Οπότε θα γίνει η διαίρεση 1Byte δηλαδή **AX** δια 10h. Το **AX** θα πρέπει να έχει το αποτέλεσμα 30.



Εικόνα 10: Step back 2 φορές

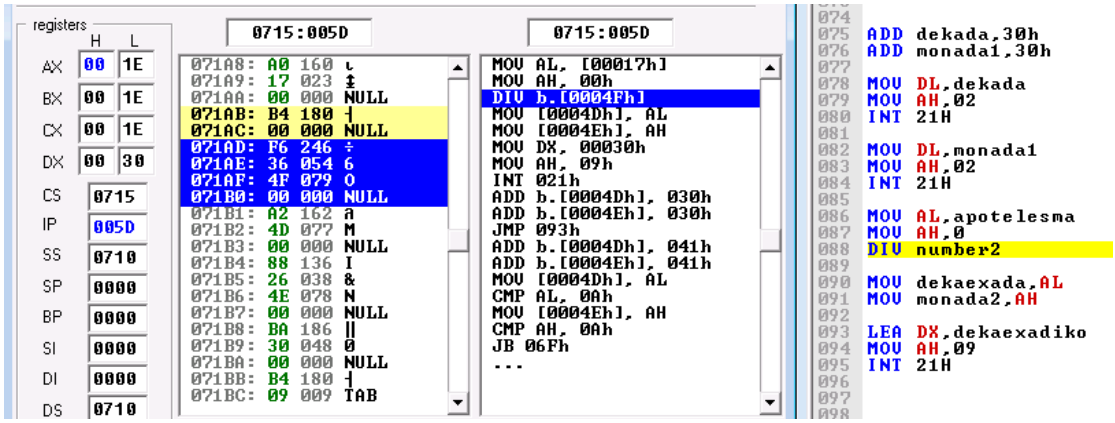
Όπως βλέπουμε το AX έχει τις εξής τιμές**:**

Στον **AH** έχει τιμή **02** και στο **AL** έχει τιμή **1E**.

Εμείς όμως είχαμε υπολογίσει το αποτέλεσμα 30 στο δεκαδικό, δηλαδή 1Eh στο δεκαεξαδικό. Δηλαδή, αντί να διαρέσουμε το 001Eh με το δεκαέξι, διαιρούμε το 021Eh με το δεκαέξι.

Όπως καταλαβαίνουμε θα πρέπει να μηδενίσουμε το high byte του AX, δηλαδή να τοποθετήσουμε μια γραμμή αμέως πριν τη διαίρεση **MOV AH,0**.

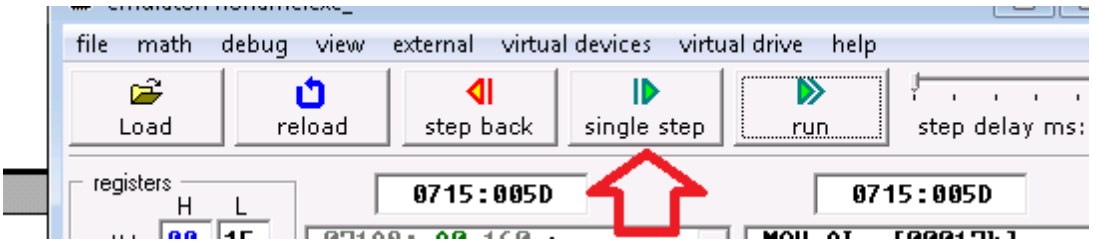
Αν το κάνουμε αυτό *(προσθέτουμε στη γραμμή 87 την εντολή MOV AH,0)*, κάνουμε πάλι assemble και φτάσουμε στην ίδια γραμμή με **[RUN UNTIL],** έχουμε:



Εικόνα 11: RUN UNTIL

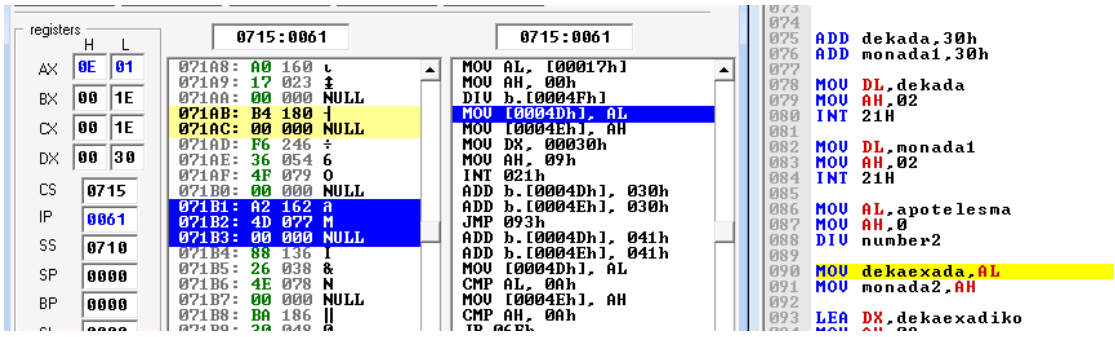
δηλαδή τώρα θα γίνει σωστά η διαίρεση AX/16, δηλαδή **001Eh/ 16.**

Αν πατήσουμε το κουμπί **[Single Step]**



Εικόνα 12: Single step

θα εκτελεστεί η επόμενη εντολή, δηλαδή η εντολή της διαίρεσης, και θα έχουμε ως αποτέλεσμα το πηλίκο να μπει στον AL και το υπόλοιπο στον AH.



Εικόνα 13: Εκτέλεση της επόμενης εντολής

Δηλαδή στον **AH** μπήκε το **0Eh** ενώ στον **AL** το **01**.

Ως αυτό το σημείο είμαστε λοιπόν σωστοί.

Το συγκεκριμένο πρόγραμμα ασφαλώς έχει και άλλα σφάλματα, αλλά με αυτό τον τρόπο έχουμε διασφαλίσει ότι ω τη γραμμή 91 είναι σωστό το πρόγραμμα.

Στη συνέχεια θα πρέπει να συνεχίσουμε με step-by-step και να παρατηρούμε τις τιμές στους καταχωρητές και στις μεταβλητές, και θα καταλάβουμε που είναι το επόμενο πρόβλημα.

Ένα πρόγραμμα μπορεί να έχει παραπάνω από ένα σφάλματα. Η διαδικασία της αποσφαλμάτωσης μερικές φορές απαιτεί πολύ περισσότερο χρόνο από το χρόνο που χρειαστήκαμε για να γράψουμε το πρόγραμμα.

Καλή επιτυχία στο debugging λοιπόν!