ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Αρχιτεκτονική Υπολογιστών

Ασκήσεις Εργαστηρίου

Ενότητα: ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ No 12

Δρ. Μηνάς Δασυγένης

mdasyg@ieee.org

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών

http:// arch.ece.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.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Περιεχόμενα

1. Σκοπός της άσκησης

- Γραφικά VGA *(συνέχεια προηγούμενου εργαστηρίου γραφικών).* Σχεδιασμός γραμμών με βελτιστοποίηση. Σχεδιασμός χρωματικής παλέτας. Σχεδιασμός & γέμισμα ορθογωνίου.
- Assembly στους σύγχρονους υπολογιστές

(A) 2 Ερωτήσεις

(C) 4 Ασκήσεις

(B) 12 Ερωτήσεις bonus (για +300%)

2. Εργαστηριακή άσκηση

(Θα πρέπει να έχει ολοκληρωθεί το προηγούμενο εργαστήριο με τις συναρτήσεις γραφικών για να συνεχίσετε σε αυτό.)

(C1) Επιπρόσθετη βελτιστοποίηση στο σχεδιασμό γραμμής

Οι συναρτήσεις που κατασκευάσατε στο προηγούμενο εργαστήριο μπορούν να βελτιστοποιηθούν για την επίτευξη ακόμη μεγαλύτερης ταχύτητας. Θα κάνουμε τη βελτιστοποίηση για μια συνάρτηση. Η ίδια τεχνική μπορεί να χρησιμοποιηθεί και για τις άλλες συναρτήσεις.

- Αντιγράψτε όλη τη συνάρτηση **`drawline_horizontal_right`** στη συνάρτηση **`drawline_horizontal_right2`**.
- Τροποποιήστε κατάλληλα τις ετικέτες μνήμης που υπάρχουν δηλωμένες μέσα στη συνάρτηση για να μη δημιουργείται πρόβλημα με τις ετικέτες τις αρχικής συνάρτησης **`drawline_horizontal_right.`**
- **1η βελτιστοποίηση:**Στα προηγούμενα παραδείγματα κάθε φορά γράφονταν 1 Byte με την εντολή `mov es:[si],9`. Μπορούμε να τροποποιήσουμε αυτή την εντολή με το να **γράφουμε 2 byte** κάθε φορά με το να γράψουμε **`mov es:[si],0909h`**. Βέβαια θα πρέπει τότε το si να αυξάνεται κατά 2. Δηλαδή να υπάρχουν δυο συνεχόμενες εντολές **`inc si`**, αντί για 1 που υπήρχε τώρα.
- **2η βελτιστοποίηση:** Αμέσως μετά τα push υπάρχει ο υπολογισμός του αρχικού pixel με μια πράξη πολλαπλασιασμού. Να αντικαταστήσετε τον πολλαπλασιασμό, ώστε να γίνεται με πράξεις ολίσθησης και πρόσθεσης, όπως έχει αναφερθεί στη διάλεξη.
- Αντικαταστήστε την κλήση της αρχικής συνάρτησης στο πρόγραμμά σας με την κλήση αυτής της συνάρτησης και επιβεβαιώστε την ορθή (και πιο γρήγορη) λειτουργία.

(C2) Η συνάρτηση print_color_palette

Η συνάρτηση θα εκτυπώνει όλα τα χρώματα της παλέτας μας *(από 0 έως 255)*. Όπως είδαμε, η τιμή που γράφεται στη μνήμη της οθόνης **es:[si]** απεικονίζει το χρώμα που θα χρησιμοποιήσουμε. Ως τώρα τοποθετούσαμε την τιμή 9 *(γαλάζιο)*. Σε αυτή τη συνάρτηση θα έχουμε ένα βρόχο εκτύπωσης που θα αυξάνει κατά 1 κάθε φορά την τιμή του χρώματος. Συγκεκριμένα στη συνάρτηση αυτή:

- Κάθε δύο γραμμές των pixels *(οι γραμμές θα εκτυπώνονται ταυτόχρονα)* θα έχουν δύο χρώματα. Δηλαδή, εδώ θα υπάρχει ένας βρόχος 160 *επαναλήψεων (320 είναι τα pixel σε μια γραμμή, άρα τα μισά είναι 160).* Αυτός θα είναι ο εσωτερικός βρόχος επανάληψης.
- Μόλις φτάσουμε στο τέλος της γραμμής, τότε θα πηγαίνουμε 2 γραμμές παρακάτω για να μη σβήσουμε τη 2η γραμμή που τοποθετήσαμε στο παραπάνω βρόχο. Αυτό θα γίνεται με την προσθήκη της τιμή 320 *(δηλαδή μιας γραμμής)* στο δείκτη.
- Θα υπάρχει ένας βρόχος 256 επαναλήψεων (τόσες είναι οι δυνατές τιμές χρώματος). Αυτός θα είναι ο εξωτερικός βρόχος επανάληψης.
- Βέβαια τα παραπάνω σημαίνουν ότι θα βγούμε εκτός από τα όρια της οθόνης, αφού η οθόνη μας έχει διαστάσεις 360 * 200. Για αυτό το λόγο στην 200 επανάληψη θα μηδενίσετε τη μετατόπιση από την αρχή, ώστε το επόμενο χρώμα να αρχίσει από την πάνω αριστερή γωνία (μετατόπιση 0).
- Θα επιλέξουμε τον καταχωρητή DL ως καταχωρητή χρώματος. Δώστε αρχική τιμή 1 για να ξεκινήσουμε από το χρώμα με τιμή 1.
- Θα επιλέξουμε τον καταχωρητή SI ως δείκτη στη μνήμη VGA που θα γράφουμε. Έτσι λοιπόν θα γράφουμε στην οθόνη ως: `mov es:[si],dl` ενώ ταυτόχρονα θα γράφουμε και στο αντίστοιχο pixel στη γραμμή ακριβώς από κάτω με την εντολή `mov es:[si+320],dl`
- Και οι δυο βρόχοι θα χρησιμοποιούν τον καταχωρητή CX. Επειδή ο ένας βρόχος είναι εμφωλιασμένος στον άλλο, θα πρέπει να χρησιμοποιήσουμε το σωρό για να διατηρούμε την τιμή του CX ως εξής:

```
MOV CX,255 ;for all the colors
_external_loop:
PUSH CX
;first half
mov CX,160 ;for half-the-line
_internal_loop:
…..εγγραφή pixel
…..αύξηση του SI κατά 1
loop _internal_loop
pop CX
inc dl ;use next color
```

; {λείπει κώδικας...}

```
;second-half
mov CX,160 ;for half-the-line
_internal_loop:
…..εγγραφή pixel
…..αύξηση του SI κατά 1
loop _internal_loop
pop CX
inc dl ;use next color

add si,320 ;jump one line
loop _external_loop
```

(C3) Σχεδιασμός ορθογώνιου

Κατασκευή συνάρτησης με όνομα **print_a_box** που θα δέχεται τις εξής εισόδους:

- DL color
- BX,AX (x,y) center of base line
- CX length
- DH height

Πρώτα θα σχεδιάσουμε τις δυο οριζόντιες γραμμές με τον πρώτο βρόχο επανάληψης και στη συνέχεια τις δυο κατακόρυφες γραμμές με το δεύτερο βρόχο.

- Υπολογίζουμε τη διεύθυνση μνήμης για το pixel (BX,AX) με τον τύπο AX*320 + BX.
- Το αποθηκεύουμε σε ένα δείκτη π.χ. di
- Υπολογίζουμε το ύψος *(αριθμός pixel)* που πρέπει να αφαιρέσουμε με τον τύπο DH * 320
- Δημιουργούμε ένα βρόχο από CX έως 0 στο οποίο κάθε φορά σχεδιάζουμε 4 σημεία:
    - το σημείο DI + CX
    - το σημείο DI – CX
    - το σημείο DI + CX – DH*320
    - το σημείο DI – CX – DH*320

Προσέξτε ότι για τη διευθυνσιοδότηση μνήμης δε μπορούμε να έχουμε CX και DX, αλλά συγκεκριμένους καταχωρητές.

Το CX θα μειώνεται συνεχώς λόγω της εντολής loop, οπότε στο τέλος θα έχουν σχεδιαστεί πλήρως οι 2 γραμμές.

- Δημιουργούμε ένα βρόχο που θα επαναληφθεί για DH φορές.
- Αφαιρούμε από το DI το μήκος που είχε δώσει ο χρήστης CX για να δείχνει στο πάνω αριστερά σημείο του ορθογωνίου.
- Θα σχεδιάζουμε κάθε φορά 2 σημεία:

- το σημείο DI *(αντιστοιχεί στην αριστερή κατακόρυφη)*
- το σημείο DI+BX όπου BX είναι το μήκος όλης της γραμμής, δηλαδή του μήκους που είχε δώσει ο χρήστης επί 2, CX*2 *(αντιστοιχεί στη δεξιά κατακόρυφη)*
- Για να πάμε στην επόμενη γραμμή προσθέτουμε στο DI την τιμή 320 και επαναλαμβάνουμε.

Δοκιμάζουμε στο κυρίως πρόγραμμα τη συνάρτηση:

```
mov dl,9
mov bx,50
mov ax,50
mov cx,10
mov dh,20
call print_a_box

mov dl,19
mov bx,50
mov ax,71
mov cx,15
mov dh,20
call print_a_box
```

Θα πρέπει να δούμε να σχεδιάζονται 2 ορθογώνια.

(C4) Σχεδιασμός ορθογώνιου με γέμισμα

Να κατασκευάσετε συνάρτηση με όνομα **`print_a_box_with_fill`** που θα δέχεται τις εξής εισόδους:

- DL line + fill color
- BX,AX (x,y) center of base line
- CX length
- DH height

3. Ερωτήσεις κατανόησης

(A1) Αναγνωρίστε στον κώδικα σας 2 πιθανούς κινδύνους διασωλήνωσης.

(A2) Δώστε πιθανούς τρόπους να απομακρύνετε τους παραπάνω 2 κινδύνους διασωλήνωσης.

Β Μέρος Εργαστηρίου *(Bonus +300%)*[1]

4. Ερωτήσεις Bonus

4.1 Χρήση της Assembly στους σύγχρονους υπολογιστές

Ένα από τα σημαντικά πλεονεκτήματα της assembly είναι ότι μπορεί να χρησιμοποιηθεί μέσα σε μια υψηλή γλώσσα προγραμματισμού όπως ή C ή JAVA. Η ταυτόχρονη χρησιμοποίηση assembly και C έχει το έξης όφελος. Ενώ χρησιμοποιούμε για το μεγαλύτερο μέρος του κώδικά μας, το φιλικό περιβάλλον προγραμματισμού, τις συναρτήσεις, τις βιβλιοθήκες και την ευκολία αποσφαλμάτωσης της υψηλής γλώσσας προγραμματισμού, κάποια κρίσιμα κομμάτια κώδικα μπορούμε να τα γράφουμε σε assembly προκειμένου να επιτύχουμε καλύτερες επιδόσεις. Ο κώδικα assembly που παράγεται από το συμβολομεταφραστή *(compiler)* αυτόματα από μια υψηλή γλώσσα προγραμματισμού, δεν είναι τόσο βέλτιστος και υστερεί στις επιδόσεις και στην κατανάλωση ενέργειας, ως προς τον κώδικα assembly που μπορεί να γράψει ένας έμπειρος προγραμματιστής.

Για αυτό το τμήμα του εργαστηρίου απαιτείται πρόσβαση σε 32bit λειτουργικό σύστημα ή 64bit λειτουργικό σύστημα που έχει υποστήριξη για μεταγλώττιση και αποσφαλμάτωση 32bit που εκτελεί Linux ή FreeBSD. Η 64bit assembly Intel/AMD είναι υπερσύνολο της assembly x86 των προηγούμενων εργαστηρίων. Μπορείτε να διαβάσετε το κείμενο Gentle Introduction to x86-64.pdf το οποίο βρίσκεται στην διεύθυνση: http://www.x86-64.org/documentation/assembly.html .

Σε περίπτωση που χρησιμοποιείτε 32bit δε χρειάζεται να κάνετε κάποια αλλαγή.

Αν χρησιμοποιήσετε 64bit θα πρέπει να κάνετε compile με τις παραμέτρους `-m32` και `-L/usr/lib32` . Επίσης, για 64 bit, θα πρέπει να έχετε εγκατεστημένες τις αντίστοιχες βιβλιοθήκες, π.χ. το `gcc-multilib` ή `gcc-7-multilib` ή `lib32gcc-7-dev` ή παρόμοιο.

ΠΡΟΣΟΧΗ: Το ΛΣ πρέπει να είναι είτε FreeBSD, είτε Linux. Υβριδικές υλοποιήσεις όπως BASH σε Windows 10 ή Cygwin ενδέχεται να μη λειτουργούν (και ο διδάσκων προτείνει να τις αποφύγετε). Μπορείτε να χρησιμοποιήσετε το μηχάνημα **zafora.ece.uowm.gr** ή το **pleiades.ece.uowm.gr** **(οδηγίες: http://zafora.ece.uowm.gr)**

4.2 Κώδικας Assembly από πρόγραμμα C

Σε αυτό το κομμάτι θα προγραμματίσουμε σε υψηλή γλώσσα προγραμματισμού και θα δούμε τον κώδικα assembly που παράγεται αυτόματα.

1. Ανοίξτε μια σύνδεση ή ένα τερματικό στο μηχάνημα που θα εργαστείτε.

[1] **Υπολογισμός ποσοστού ολοκλήρωσης Bonus (ενδέχεται ο φοιτητής να εξεταστεί ατομικά αν υλοποιήσει αυτό το bonus):**
300 * (αριθμός_απαντήσεων_που_δώσατε)/(συνολικός_αριθμός_ερωτ.)

2. Στην προτροπή γράψτε **pico asm0.c** ή **nano asm0.c** για να δημιουργήστε το αρχείο **asm0.c** και να τοποθετήσετε τον παρακάτω κώδικα, ο οποίος θέτει μια αρχική τιμή στο a και στο b, εκτελεί μια μαθηματική πράξη τοποθετώντας το αποτέλεσμα στο c, και εκτυπώνει το αποτέλεσμα:

```
#include <stdio.h>
main()
{
// Variable Declaration
int a;
int b;
int c;
//Start Here
a=10;
b=25;
c=a*b+140;
printf("The result is %d\n",c);
return;
}
```

3. Πατήστε το συνδυασμό των πλήκτρων για την αποθήκευση (συνήθως **CTRL+X** και στην προτροπή για αποθήκευση πατήστε **y)**. Θα γίνει η αποθήκευση και θα επιστρέψετε στη γραμμή εντολών του λειτουργικού συστήματος.
4. Κάντε compile τον κώδικα **asm0.c** με αποθήκευση του παραγόμενου αρχείου στο **asm0**, χρησιμοποιώντας το συμβολομεταφραστή gcc ως εξής (αν σας αναφερθεί κάποιο error τότε επιστρέψτε στο βήμα 2 για να διορθώσετε τον κώδικά σας. Αν δεν σας εκτυπωθεί κανένα μήνυμα τότε ο κώδικάς σας είναι σωστός): **gcc asm0.c -o asm0**
5. Αμέσως μετά την εκτέλεση της ανωτέρω εντολής θα δημιουργηθεί το εκτελέσιμο asm0. Εκτελέστε το με την εντολή **./asm0**
6. Προκειμένου να δείτε τον κώδικα assembly θα χρησιμοποιήσετε το πρόγραμμα αποσφαλμάτωσης GNU Debugger++[2]. Δώστε **gdb ./asm0**
7. Θα σας εμφανιστεί η προτροπή **(gdb).** Δώστε την εντολή **disassemble main** για να δείτε τις εντολές assembly που έχουν δημιουργηθεί για τη συνάρτηση main. Θα δείτε ότι υπάρχουν 3 στήλες:

   1η στήλη → διεύθυνση εντολής μέσα στο εκτελέσιμο

   2η στήλη → μετατόπιση από την αρχή του προγράμματος

   3η στήλη → εντολή assembly σε συντακτικό AT&T *(att)*

   Για να μετατρέψουμε τις εντολές assembly σε συντακτικό Intel *(όπως έχουμε χρησιμοποιήσει στο εργαστήριο)* θα πρέπει να δώσετε την εντολή **set disassembly-flavor intel**

[2] Περισσότερες πληροφορίες για το debugger στη διεύθυνση:
http:// www.delorie.com/gnu/docs/gdb/gdb.html#SEC_Top

Αφού δώσετε την παραπάνω εντολή, δώστε πάλι την εντολή εμφάνισης assembly και θα παρατηρήσετε τη διαφορά. Σημειώστε ότι ο machine code *(που δεν εμφανίζεται)* είναι ίδιος, είτε μας εμφανιστεί η assembly με το συντακτικό της Intel είτε με το συντακτικό της AT&T.

(B1) Δώστε ένα screenshot **b1.png** στο οποίο φαίνεται το dissasembly σε μορφή intel.

(B2) Από το ανωτέρω listing, ποια εντολή assembly χρησιμοποιείται για την **πράξη a*b** και ποια για την **πράξη +140 .**

(B3) Πόσα Byte καταλαμβάνει η πρώτη εντολή και πόσα η δεύτερη *(και γιατί;).*

(B4) Σε ποια διεύθυνση μνήμης έχει τοποθετηθεί;

*** Προσοχή! Όλες οι παρακάτω εντολές δίνονται στο πρόγραμμα gdb, δηλαδή στην προτροπή (gdb) ***

8. Για να κάνουμε βηματική εκτέλεση *(step-by-step)* θα θέσουμε ένα breakpoint στην αρχή της συνάρτησης main με την εντολή: break main
9. Δώστε **run** για να αρχίσει η εκτέλεση έως το σημείο που έχει μπει το breakpoint. Μόλις η εκτέλεση έρθει στο σημείο που υπάρχει το breakpoint, τότε θα σταματήσει και θα μας εμφανιστεί η προτροπή του gdb.
10. Δώστε **info registers** για να σας εμφανιστούν οι τιμές των καταχωρητών του συστήματος πριν την εκτέλεση της συγκεκριμένης εντολής που έχει το **breakpoint**. Δώστε **stepi** για να εκτελεστεί η εντολή assembly.

    (B5) Τοποθετήστε ένα breakoint στη διεύθυνση της εντολής του πολλαπλασιασμού, χρησιμοποιώντας τη σύνταξη **break * 0x00000...** , *(και προαιρετικά την εντολή* **continue***)* όπου μετά το αστέρι, τοποθετείται η διεύθυνση της συγκεκριμένης εντολής. Δώστε την εντολή εμφάνισης των καταχωρητών και βρείτε ή εκτιμήστε σε ποιον καταχωρητή ή διεύθυνση μνήμης έχει τοποθετηθεί το a και που το b. Εκτελέστε την εντολή assemby, εμφανίστε τους καταχωρητές και βρείτε που έχει αποθηκευτεί το αποτέλεσμα του πολλαπλασιασμού.

4.3 Κώδικας assembly μέσα σε πρόγραμμα C

Υπάρχουν περιπτώσεις που απαιτείται η συγγραφή τμήματος κώδικα ενός προγράμματος σε assembly προκειμένου να επιταχυνθεί η λειτουργία του προγράμματος ή να γίνει χρήση κάποιας λειτουργίας που δεν είναι αντιστοιχεί σε εντολή υψηλού επιπέδου. Αυτό ονομάζεται inline assembly code και ο τρόπος που γίνεται αναλύεται στο έγγραφο **GCC Inline Assembly Howto.pdf** που προέρχεται από τη διεύθυνση: http:// ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html

Η επικοινωνία του προγράμματος C με τον κώδικα assembly γίνεται χρησιμοποιώντας το σωρό. Η σύνταξη της inline assembly ακολουθεί το συντακτικό της AT&T. Οι κυριότερες διαφορές είναι:

- Οι καταχωρητές έχουν δύο %% από μπροστά

- Οι καταχωρητές προορισμού τοποθετούνται στο τέλος της εντολής.
- Στο τέλος κάθε εντολής πρέπει να υπάρχει το \n\t
- Η κάθε εντολή μπαίνει μέσα σε διπλά εισαγωγικά
- **(προσοχή)** Αμέσως μετά τα διπλά εισαγωγικά υπάρχει ένα κενό μετά ο χαρακτήρας \ και μετά enter *(αλλαγή γραμμής)*. Εναλλακτικά μπορούμε να μη βάλουμε το \ και το enter και να δώσουμε όλες τις εντολές assembly σε μια σειρά.
- Μετά την τελευταία εντολή, ακολουθεί ένα πεδίο που ξεκινάει με **:** που δείχνει τους καταχωρητές εξόδου, στη συνέχεια ακολουθεί ένα πεδίο που ξεκινάει με **:** που δείχνει τους καταχωρητές εισόδου, και μετά ένα πεδίο που ξεκινάει με **:** και δείχνει τους clobbered_registers δηλαδή τους καταχωρητές που έχουμε τροποποιήσει και πρέπει ο compiler να το λάβει υπόψη για να μη δημιουργηθεί πρόβλημα στο περιβάλλον κλήσης.

Σε αυτό το κομμάτι του εργαστηρίου θα γράψουμε κάποιες εντολές assembly για να διαπιστώσουμε κάθε φορά που κάνουμε push έναν καταχωρητή της Intel/AMD 64bit πόσα byte τοποθετούνται στο σωρό. Για να το υπολογίσουμε θα κάνουμε τα εξής:

- θα αποθηκεύσουμε την τρέχουσα τιμή του **SP** *(καταχωρητής eSP για την 64bit)* σε έναν καταχωρητή.
- Θα κάνουμε push έναν καταχωρητή 64bit, όπως τον **rbx**.
- Θα αφαιρέσουμε από την αρχική τιμή του **SP** *(που είναι αποθηκευμένη στο βήμα 1)* τη νέα τιμή του **SP** που έχει τώρα.
- Θα τοποθετήσουμε στο σωρό το αποτέλεσμα, προκειμένου να παραληφθεί από το πρόγραμμα σε C.

1. Ανοίξτε μια σύνδεση ή ένα τερματικό στο μηχάνημα που θα εργαστείτε.
2. Στην προτροπή γράψτε **pico asm1.c** ή **nano asm1.c** για να δημιουργήστε το αρχείο asm1.c και να τοποθετήσετε τον παρακάτω κώδικα:

```
#include <stdio.h>
int test_pushl()
      {
      asm volatile ( "mov %%esp, %%eax\n\t" \
                     "push %%rbx\n\t" \
                     "sub %%esp, %%eax\n\t" \
                     "pop %%rbx": : :"%eax","%esp"); }
int main()
      {
      printf("It used %d bytes\n", test_pushl());
      return;
      }
```

3. Κάντε compile το παρακάτω πρόγραμμα με την παράμετρο –g (για εισαγωγή συμβόλων αποσφαλμάτωσης), με έξοδο στο αρχείο **asm1**.
4. Εκτελέστε το **./asm1** και δείτε το νούμερο που σας εμφανίζεται στην έξοδο.
5. Εκτελέστε το gdb με παράμετρο το asm1
6. Δώστε την εντολή εμφάνισης assembly για τη συνάρτηση **test_pushl().**

7. Αλλάξτε την εμφάνιση του συντακτικού της assembly σε Intel και επαναλάβετε το προηγούμενο βήμα

   (B6) Πόσα bytes καταλαμβάνει κάθε μια από τις εντολές assembly που έχετε δώσει inline μέσα στο πρόγραμμα C; Χρησιμοποιήστε την εντολή **x/?bx** διεύθυνση_μνήμης, όπου **?** είναι ο αριθμός των Bytes που θέλετε να εμφανίσετε (για παράδειγμα **x/2bx 0x0000000000400567**) για να εμφανίσετε τα Bytes του machine code κάθε inline assembly εντολής που έχετε δώσει.

Βγείτε με **quit** στην προτροπή εντολών του λειτουργικού συστήματος.

8. Εκτός από το gdb μπορείτε να χρησιμοποιήσετε το βοηθητικό εργαλείο gdbtui με την ίδια σύνταξη. Δώστε την εντολή **gdbtui ./asm1**

9. Στην προτροπή δώστε **layout asm** για να εμφανίσετε την assembly στο άνω παράθυρο.
   (B7) Δώστε ένα screenshot με όνομα **b7.png** με την assembly που εμφανίζεται.

10. Τοποθετήστε ένα **breakpoint** στο main και πατήστε **run** για να εκτελεστεί μέχρι αυτό το σημείο.

11. Μπορείτε με τα βέλη/pageup/pagedown να εμφανίζετε τις προηγούμενες ή τις επόμενες εντολές assembly.

12. Αλλάξτε το συντακτικό εμφάνισης σε Intel για να είναι πιο οικείο.

13. Μπορείτε να δώσετε **display/i $pc** για να εμφανίζεται αυτόματα η επόμενη εντολή κάθε φορά. Πατήστε μια φορά το **stepi** και κάθε φορά που θα πατάτε το enter εμφανίζεται η εντολή που θα εκτελεστεί. Δοκιμάστε το.

5. Αποσφαλμάτωση κώδικα assembly μέσα σε πρόγραμμα C

Σας δίνεται το παρακάτω πρόγραμμα σε C που έχει inline assembly.

```
#include <stdio.h>
int f1 (int a)
{ return a+5; }
void f2 (int lock, int a)
{printf("\nans = %d\n", lock + a);}
#define temp(lock) \
     asm volatile ( " push $0\n\t" \
                    " call f1\n\t" \
                    " push %0\n\t" \
                    " push %%eax\n\t" \
                    " call f2\n\t" \
                    " pop %%ax\n\t" \
                    " pop %%eax\n\t" \
```

```
                        " pop %%ds\n\t"\
          " pop %%eax": : "r" (lock) : "memory", "%eax");
int main ()
      {
      int lock = 10;
      temp (lock);
      return 0;
      }
```

1. Κάντε το compile και εκτελέστε το. Θα εμφανιστεί ένα πρόβλημα (**Illegal instruction (core dumped)**) . Αναλόγως της έκδοσης του compiler, ενδέχεται να μη γίνει καν compile, αλλά να αναφερθεί ως σφάλμα 'type mismatch'. Σε αυτή την περίπτωση θα πρέπει να κάνετε compile για 32bit και όχι 64bit.
2. Φορτώστε το στο **gdb/gdbtui** και κάντε **dissasembly**. Συγκρίνετε τις εντολές που έχετε γράψει μια προς μια με τις εντολές από το **dissasembly** και βρείτε πια εντολή είναι προβληματική *(δεν εμφανίζεται όπως τη γράψατε).* Σβήστε τελείως αυτή την εντολή από το αρχείο C, κάντε compile, εκτελέστε το.

   (B8) Ποια εντολή προκάλεσε το πρόβλημα και γιατί;

   (B9) Τι κάνει το συγκεκριμένο πρόγραμμα;

   (B10) Πως μεταφέρονται οι παράμετροι από την assembly σε κάθε συνάρτηση;

   (B11) Επιστρέφει κάποια τιμή η assembly και γιατί;

   (B12) Πως δίνουμε από το κυρίως πρόγραμμα τιμή στην assembly και μέσα στην assembly πως χρησιμοποιούμε αυτή την τιμή;