Assignment 5: C and Assembly Code Tutorial: Computing Sums

Σκοπός
Ο στόχος της άσκησης αυτής είναι να βοηθήσει στην εισαγωγή (1) σε assembly των επεξεργαστών της αρχιτεκτονικής x86, (2) στην αναπαράσταση της γλώσσας assembly σε κώδικα μηχανής στη μνήμη του συστήματος, και (3) στην χρήση του gdb.

Background
Η άσκηση αυτή θα γίνει σε **Linix/x86** συστήματα του τμήματος (portokali, milo, rodakino, etc.). Για θέματα της αρχιτεκτονικής του x86 και της γλώσσας assembly που χρησιμοποιεί μπορείτε να συμβουλευτείτε την online έκδοση του βιβλίου: Programming from the Ground Up, Jonathan Bartlett 2004. Ιδιαίτερα χρήσιμα θα βρείτε τα Chapter 3 και Appendix B.
Τις απαντήσεις σας στα ερωτήματα θα τις συμπληρώνετε στο README.md αρχείο που υπάρχει στο repo assignment5<\a>
Για κάθε ερώτημα που απαντάτε καλό είναι και θα πρέπει να είναι ένα ξεχωριστό commit στο git repository σας.

Η άσκηση
1. Μέσα στο repo θα βρείτε το πρόγραμμαx86asm.s.

(a) Κάντε compile και εκτελέστε το πρόγραμμα με την εντολή:

gcc -o x86asm x86asm.s

Τι τυπώνει το πρόγραμμα; Δώστε την απάντησή σας στο readme αρχείο της άσκησης σας.

(b) Αντιγράψτε το αρχείο σε ένα x86asm_comments.s αρχείο και συμπληρώστε σχόλια που δείχνουν τι κάνει το πρόγραμμα σε κάθε βήμα του. Είναι αρκετό να συμπληρώσετε σύντομα σχόλια δίπλα σε κάθε σύμβολο "#" που δεν περιέχει ήδη κάποιο σχόλιο.

(c) Εκτελέστε και εξετάστε το πρόγραμμα σας μέσα από τον gdb.

Κάντε compile το x86asm.s με την εντολή

gcc -g -o x86asm x86asm.s

Καλέστε τον gdb και εκτελέστε το πρόγραμμα x86asm.

$ gdb x86asm

(gdb) run

(d) Χρησιμοποιήστε την εντολή list "l" (help list) για να δείτε τον source κώδικα του προγράμματος από το source αρχείο x86asm.s.

(gdb) l main

Χρησιμοποιήστε την εντολή examine "x" (help examine) για να δείτε τα περιεχόμενα της μνήμης του προγράμματος ως εντολές assembly.

(gdb) x/??i *main

όπου ?? είναι ο κατάλληλος αριθμός εντολών. Συμπληρώστε στο readme αρχείο σας ποιος είναι ο αριθμός εντολών της main.

Χρησιμοποιήστε την εντολή examine "x" (help examine) για να δείτε τα περιεχόμενα της μνήμης του προγράμματος ως bytes.

(gdb) x/??b *main

όπου ?? είναι ο κατάλληλος αριθμός bytes. Συμπληρώστε στο readme αρχείο σας ποιος είναι ο αριθμός bytes που καταλαμβάνουν οι εντολές της main.

Βάλτε ένα breakpoint μετά το τέλος του loop στη main:

(gdb) b *main+?? (όπου ?? το κατάλληλο offset)

(gdb) run

Εκτελέστε την υπόλοιπη συνάρτηση main (μέχρι το τελικό μήνυμα του gdb "Program exited normally.", με διαδοχικές εντολές "ni".

(gdb) ni
...
(gdb) ni

Συμπληρώστε στο readme file σας το offset που χρησιμοποιήσατε.

(e) Εκτελέστε το πρόγραμμα πάλι και σταματήστε στο προηγούμενο breakpoint. Εξετάστε τα περιεχόμενα των θέσεων μνήμης S, Msg με την εντολή print "p".

(gdb) p (int)S
(gdb) p (int *)&S
(gdb) p (char *)&Msg

Γράψτε στο readme file σας τις τιμές τους και τις διευθύνσεις στις οποίες βρίσκονται.

Αλλάξτε την τιμή της διεύθυνσης S σε 99:

(gdb) set *(int *)&S = 99

Εκτελέστε το πρόγραμμα μέχρι το τέλος με διαδοχικές εντολές "ni". Συμπληρώστε στο readme file το μήνυμα που τυπώνει το πρόγραμμα.

2. Δίνεται το πρόγραμμα sum.c.

(a) Κάντε compile και εκτελέστε το πρόγραμμα με την εντολή:

```
gcc -Wall -pedantic -ansi -o sum sum.c
./sum
```

Τι τυπώνει το πρόγραμμα; Δώστε την απάντησή σας στο readme αρχείο της άσκησης σας.

(b) Μεταφράστε το πρόγραμμα σε assembly με τον gdb με την εντολή.

```
gcc -Wall -pedantic -ansi -O2 -S sum.c
```

Εξετάστε στο αρχείο sum.c που παράγεται και συμπληρώστε στο readme αρχείο σας από πόσες εντολές assembly αποτελείται η συνάρτηση main.

(c) Εκτελέστε και εξετάστε το πρόγραμμα μέσα από τον gdb.

Κάντε compile το πρόγραμμα με την εντολή:

```
gcc -Wall -pedantic -ansi -g -O2 -o sum sum.c
```

Καλέστε τον gdb και εκτελέστε το πρόγραμμα sum.

```
gdb sum
(gdb) run
```

(d) Χρησιμοποιήστε την εντολή list "l" (help list) για να δείτε τον source κώδικα του προγράμματος από το source αρχείο sum.c.

```
(gdb) l main
```

Χρησιμοποιήστε την εντολή examine "x" (help examine) για να δείτε τα περιεχόμενα της μνήμης του προγράμματος ως εντολές assembly.

```
(gdb) x/??i *main
```

όπου ?? είναι ο κατάλληλος αριθμός εντολών. Συμπληρώστε στο readme αρχείο σας ποιος είναι ο αριθμός εντολών της main.
Χρησιμοποιήστε την εντολή examine "x" (help examine) για να δείτε τα περιεχόμενα της μνήμης του προγράμματος ως bytes.

```
(gdb) x/??b *main
```

όπου ?? είναι ο κατάλληλος αριθμός bytes. Συμπληρώστε στο readme αρχείο σας ποιος είναι ο αριθμός bytes που καταλαμβάνουν οι εντολές της main.

Βάλτε ένα breakpoint μετά το τέλος του loop στη main:

```
(gdb) b ?? (όπου ?? το κατάλληλο offset στο source αρχείο)
(gdb) run
```

Εκτελέστε την υπόλοιπη συνάρτηση main (μέχρι το τελικό μήνυμα του gdb "Program exited normally." με διαδοχικές εντολές "ni".

```
(gdb) n
...
(gdb) n
```

Συμπληρώστε στο readme file σας το offset που χρησιμοποιήσατε για το breakpoint.

(e) Εκτελέστε το πρόγραμμα πάλι και σταματήστε στο προηγούμενο breakpoint. Εξετάστε τα περιεχόμενα των μεταβλητών Sum, n.

```
(gdb) p Sum
(gdb) p &Sum
(gdb) p n
(gdb) p &n
```

Γράψτε στο readme file σας τις τιμές τους και τις διευθύνσεις στις οποίες βρίσκονται.

Αλλάξτε την τιμή της μεταβλητής Sum σε 98 και 99 διαδοχικά με δύο διαφορετικούς τρόπους:

```
(gdb) set Sum = 98
(gdb) set *(int *)?? = 99 (όπου ?? η διεύθυνση της μεταβλητής Sum στη μνήμη)
```

Εκτελέστε το πρόγραμμα μέχρι το τέλος με διαδοχικές εντολές "n". Συμπληρώστε στο readme file το μήνυμα που τυπώνει το πρόγραμμα.

(f) Εκτελέστε το πρόγραμμα πάλι και σταματήστε με ένα breakpoint στην αρχή της main. Αλλάξτε την εντολή cmpl στη μνήμη ώστε να συγκρίνει τη μεταβλητή "n" με την τιμή 100 αντί για την τιμή 0. Μπορείτε να χρησιμοποιήσετε μια ακολουθία εντολών όπως η παρακάτω:

```
(gdb) x/??i *main (όπου ?? ο αριθμός εντολών assembly στη main)
(gdb) x/2i ?? (όπου ?? η διεύθυνση της εντολής cmpl)
(gdb) x/??b (όπου ?? ο αριθμός bytes που καταλαμβάνει η εντολή cmpl)
(gdb) set *(char *)(?? + ????) = 100 (όπου ?? η διεύθυνση της εντολής cmpl όπως παραπάνω και ???? το offset του byte της εντολής cmpl που περιέχει την τιμή 0)
(gdb) x/2i ?? (όπου ?? η διεύθυνση της εντολής cmpl, θα δείτε την εντολή cmpl να έχει διαφορετικά operands)
```

Σβήστε το breakpoint και εκτελέστε το πρόγραμμα μέχρι το τέλος:

(gdb) d
(gdb) r

Συμπληρώστε στο readme file σας το μήνυμα που τυπώνει το πρόγραμμα.

3. Μας ζητούν να αντικαταστήσουμε στο πρόγραμμα sum.c το while loop με πιο "αποτελεσματικό" κώδικα assembly. Οπότε χρησιμοποιούμε τον κώδικα του x86asm.s. Δίνεται το αρχείο x86sum.c που περιέχει "inline assembly" κώδικα μέσα στο C πρόγραμμα.

(a) Κάντε compile το πρόγραμμα x86sum.c με την εντολή

gcc -Wall -pedantic -ansi -fasm -o x86sum x86sum.c

Εκτελέστε το πρόγραμμα και συμπληρώστε στο readme file σας το μήνυμα που τυπώνει.

(b) Κάντε compile το πρόγραμμα x86sum.c με την εντολή

gcc -Wall -pedantic -ansi -fasm -S x86sum.c

Μετρήστε και συμπληρώστε στο readme file σας τον αριθμό των εντολών assembly που υπάρχουν στην main.

(c) Αλλάξτε το όνομα των μεταβλητών "n, Sum" σε "i, S" αντίστοιχα και ονομάστε το νέο source file x86sum2.c. Κάντε compile το νέο πρόγραμμα όπως παραπάνω, εκτελέστε το, και συμπληρώστε στο readme file το μήνυμα που τυπώνει.

Logistics
Βήμα 1: Fork repository

- Κάντε fork το repository assignment5 από την ομάδα του μαθήματος στο csd gitlab. Στη συνέχεια αλλάξτε τα permissions σε private όπως αναγράφει στα Policies. Προσθέστε ως members στο repo σας τους TAs του μαθήματος.

Βήμα 2: Source Files
Χρησιμοποιήστε τον αγαπημένο σας κειμενογράφο (emacs/vim/nano) για να συμπληρωστε τα source αρχεία που βρίσκονται κάτω από το φάκελο src (createdata3.c, createdata6.c, createdata9.c, traces).
Βήμα 3: README.md

- Χρησιμοποιήστε τα συστήματα Linux/x86 του CSD και τα εργαλεία gcc, emacs, gdb.
- Χρησιμοποιήστε τον αγαπημένο σας κειμενογράφο (emacs/vim/nano) για να επεξεργαστείτε το README.md text file που υπάρχει συμπληρώνοντας το:
    - Το όνομά σας
    - Πράγματα που χειρίζεστε με διαφορετικό τρόπο από ότι ορίζει η άσκηση.
    - Μια περιγραφή της βοήθειας που είχατε από άλλους στη δημιουργία του προγράμματος σας, και σε συμφωνία με το *Policies* section του web page του μαθήματος.
    - (Προαιρετικά) Μία ένδειξη του πόσο χρόνο αφιερώσατε για την άσκηση.
    - (Προαιρετικά) Οτιδήποτε άλλο θέλετε να αναφέρετε.
- Στο repository σας θα πρεπει να υπάρχουν στο repo τα αρχεία της άσκησης: x86asm_comments.s, x86sum2.c, README.md.

Βήμα 3: Submit
Η παράδοση της άσκησής σας θα γίνει μέσω git, σύμφωνα με τις οδηγίες που περιγράφονται στο policies. Συγκεκριμένα το repository σας στο gitlab θα πρέπει να είναι fork του repository assignment5 και θα πρέπει να προσθέσετε ως members τους TAs του μαθήματος. Σιγουρευτείτε ότι ο κώδικάς σας έχει γίνει σωστά commit και ότι φαίνονται στο online repository στο account σας στο csd-gitlab. Όταν όλα είναι έτοιμα και έχετε τελειώσει με την άσκησή σας βάλτε το tag assignment5 χρησιμοποιώντας την εντολή:

git add tag assignment5

και κάντε upload το tag χρησιμοποιώντας την εντολή:

git push origin --tags

Προσοχή! Μην κάνετε commit object και executables αρχεία.

Επειδή η εξέταση θα γίνει στα μηχανήματα του Τμήματος θα πρέπει να κάνετε clone το repository στα μηχανήματα του Τμήματος και να κάνετε compile και run τις ασκήσεις σας σε αυτά τα συστήματα. Αυτή είναι η ίδια διαδικασία που θα ακολουθηθεί την ημέρα της εξέτασης.

Στην προθεσμία της παράδοσης ένα script θα τρέξει και θα κατεβάσει όλα τα repositories που έχουν γίνει fork. Αυτά είναι τα repositories που θα βαθμολογηθούν.

Βαθμολογία
Η βαθμολογία θα βασιστεί στην κατανόηση της άσκησης.

Last Modified: 12-02-2021 20:42