ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός Ενσωματωμένων Συστημάτων
5η Σειρά Ασκήσεων

Χειμερινό Εξάμηνο, Ακαδ. Έτος: 2023-2024

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΔΙΑΜΑΝΤΙΔΗΣ ΘΕΟΧΑΡΗΣ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ: ΙΩΑΝΝΟΥ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 03119002

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ: 03119840

ΕΞΑΜΗΝΟ: 9

Εργασία assembly σε επεξεργαστή ARM

Σκοπός της 5ης εργαστηριακής άσκησης είναι η εξοικείωσή με γλώσσα assembly σε επεξεργαστές arm με σωστή χρήση του instruction set του ARM ώστε να έχουμε αποδοτικό αλλά και μειωμένο σε μέγεθος κώδικα. Για να προσομοιώσουμε την αρχιτεκτονική του arm επεξεργαστή με τα χαρακτηριστικά που θέλουμε και να τρέξουμε εκεί τον κώδικα μας χρησιμοποιούμε το vm qemu. Λειτουργεί με τον τρόπο ότι μεταφράζει εντολές ενός υποκείμενου επεξεργαστή σε εντολές που μπορούν να εκτελεστούν από τον υποκείμενο επεξεργαστή του υπολογιστή στον οποίο εκτελείται το QEMU. Αυτή η διαδικασία, γνωστή ως δυναμική μεταγλώττιση, επιτρέπει την εκτέλεση λογισμικού για διάφορες πλατφόρμες χωρίς την ανάγκη να γίνεται πλήρης εξομοίωση του υλικού. Το QEMU υποστηρίζει διάφορες αρχιτεκτονικές επεξεργαστών (όπως x86, ARM, MIPS κ.ά.) και μπορεί να χρησιμοποιηθεί για τη δημιουργία εικονικών μηχανημάτων για πολλά λειτουργικά συστήματα.

Τρέχοντας την συγκεκριμένη εντολή στον φάκελο που έχουμε εγκαταστήσει το qemu

```
sudo qemu-system-arm -M versatilepb -kernel vmlinuz-3.2.0-4-versatile -initrd initrd.img-3.2.0-4- versatile -hda debian_wheezy_armel_standard.qcow2 -append "root=/dev/sda1" -net nic -net user,hostfwd=tcp:127.0.0.1:22223-:22
```

Δημιουργούμε ένα εικονικό μηχάνημα για εξομοίωση arm αρχιτεκτονικής με μοντέλο μηχανήματος Versatile platform Baseboard ακόμη ορίζει τον πυρήνα του λειτουργικού συστήματος , το αρχείο initial ram disk που πρέπει να φορτωθεί , τον σκληρό δίσκο καθώς και την δικτύωση.

Ερώτημα 1ο : Μετατροπή εισόδου από τερματικό

Στην πρώτη άσκηση στόχος είναι να γραφεί πρόγραμμα σε assembly το οποίο θα λαμβάνει string μεγέθους εως 32 χαρακτήρων (αν είναι μεγαλύτερο θα αγνοεί τους υπολοίπους χαρακτήρες) θα το μετατρέπει κατάλληλα και θα το εκτυπώνει στην οθόνη. Αυτό θα γίνεται συνεχώς μέχρ να λάβει την συμβολοσειρά ‘q’ or ‘Q’.

Η μετατροπή θα είναι να αντικαταστεί τα πεζά γράμματα με κεφαλαία και αντίστροφα και όταν ο χαρακτήρας είναι ψηφίο ‘0’ εως ‘9’ να επιστρέφει άλλον χαρακτήρα ψηφίο όπως αναφέρεται στην εκφώνηση.

Αρχικά γράφουμε τον κώδικα σε C γλώσσα ώστε να καταλάβουμε πλήρως την λειτουργία του αλλά και να μας είναι πιο εύκολο να τον μετατρέψουμε σε assembly.Φυσικά το να πράξουμε αυτόν τον απλό κώδικα είναι αρκετά εύκολο καθώς υπάρχουν πολλές έτοιμες συναρτήσεις και το ζητούμενο είναι απλό για μία high level γλώσσα όπως η C.

Παρακάτω φαίνεται ο κώδικας:

```c
#include <stdio.h>
#include <ctype.h>

char UP(char ch) {
    if(isupper(ch))
            {
            ch = tolower(ch);
            }
```

```c
    if(islower(ch))
            {
            ch = toupper(ch);
            }
     return ch;
}

char code_digits(char ch) {
    if (ch >= '0' && ch<= '4')
            {
                ch = ch +5 ;
            }
            else if (ch >= '5' && ch<= '9')
            {
               ch =ch -5;
            }
                 return ch;
}


int main() {
    char userInput[32];
    char temp;
    size_t length ;

    printf("Enter a string: ");
    scanf("%31s", userInput);
    length = strlen(userInput);

     while( (userInput[0] != 'q' && userInput[0] != 'Q') || length >1 )
    {
        for(int i =0; i<31;i++) {
              userInput[i] =UP(userInput[i]);
              userInput[i]=code_digits(userInput[i]);
        }
             printf("Output string is: %s\n", userInput);

            printf("Enter a  string: ");
            scanf("%31s", userInput);
            printf("You entered: %s\n", userInput);
            length = strlen(userInput);

    }
 printf("Exit....: ");
    return 0;
}
```

Στον παραπάνω κώδικα φαίνεται ξεκάθαρα ότι χωρίζεται σε 2 βασικά μέρη. Αρχικά την εισαγωγή του string(τους 32 πρώτους χαρακτήρες) σε πίνακα μέσα σε ένα while loop και την εκτύπωση του στην οθόνη και το δεύτερο μέρος είναι η συνάρτηση που κάνει την μετατροπή της συμβολοσειράς για κάθε χαρακτήρα. Η μετατροπή χωρίζεται στο να αντικαταστήσει στην συνάρτηση UP που αντικαθιστά τα πεζά με κεφαλαία και ανάποδα και την συνάρτηση code_digits που που μετακινείται κυκλικά στον κώδικα asci κατά 5 θέσεις κυκλικά ώστε α πάρει το κατάλληλο ψηφίο.

Τώρα βασικός σκοπός της άσκησης είναι να μετατρέψουμε αυτόν τον κώδικα σε assembly ARM .

```
.section .data
output_msg1: .asciz "Input a string up to 32 characters long:"
length1=.-output_msg1
output_msg2: .asciz "Result:"
length2=.-output_msg2
output_msg3: .asciz "Exiting...\n"
length3=.-output_msg3

character:.space 1 //We use this as a variable to read every character
buffer:.space 33 //We want the buffer to keep the first 32 characters
and the last one is <ENTER>

.section .text
.global main

main:
    LDR R1,=output_msg1 //We start the programm and print the msg1
    LDR R2,=length1
    BL printf

    //We read the string with a while-loop and store it in the buffer
    MOV R11,#0 //We use R11 as a counter
    LDR R12,=buffer

READING_LOOP:
    BL scanf            //We read one letter in the character array
    LDR R0,=character   //We get the address of character in R0
    LDRB R1,[R0]        //We load the character we read in R1
    CMP R1,#'\n'        //If it is <Enter> the flag Z is set and dont
check if we have more than 32 characters
    CMPNE R11,#32       //If flag Z is 0 then we must check if we have
more than 32 characters
    ADDNE R11,R11,#1        //We increase the counter
    STRNEB R1,[R12],#1  //If either comparison do not set the flag Z
then we put it in the buffer and then increase counter
    BNE READING_LOOP    //Here the reading is over

    MOV R2,#'\n'        //We add one last <Enter> in the buffer
```

```
    STRB R2,[R12],#1    //R12 shows one address more than buffer's end
    ADD R11,R11,#1      //We count the <Enter> we add too
    CMP R1,#'\n'        //If last character is not <Enter> flash the
cache
    BLNE flush_cache    //The cache is flashed by reading all the
characters left and when it reads \n it stops

    LDR R0,=buffer      //We check is the user gave us 'Q' or 'q'
    LDRB R1,[R0]        //We get the first character of buffer
    CMP R1,#'q'         //If it is q we dont check if it is Q and we
check the number of total charcters
    CMPNE R1,#'Q'       //If it is not q we check if it is Q. If so
then we check the number of characters else the flag Z is 0 and we skip
the next 2 intructions
    CMPEQ R11,#2        //We check the number of charcters with 2
beacuse we have also included <Enter>
    BEQ END             //If (character==2 &&(buffer[0]=='q' ||
buffer[0]=='Q')) the programm ends

   LDR R0,=buffer       //R0 show in the beggining of the buffer and
R12 shows in the end
   BL manipulation

    LDR R1,=output_msg2 //We print the result
    LDR R2,=length2
    BL printf
    LDR R1,=buffer
    MOV R2,R11
    BL printf
    B main

END:
    LDR R1,=output_msg3
    LDR R2,=length3
    BL printf
    MOV R7,#1           //The syscall with R7=1 ends the program
    SWI 0

printf:
    MOV R0,#1       //We want to print the message in the stdout
    MOV R7,#4       //The syscall with R7=1 uses write()
    SWI 0           //We do the system call for printing out
    BX lr           //We return to the main program

scanf:
    MOV R0,#0
    LDR R1,=character
    MOV R2, #1
```

```
    MOV R7, #3
    SWI 0
    BX lr

flush_cache:
    PUSH {lr}       //We save lr beacuse we call scanf and lose the lr
to return into main program
    BL scanf        //We read charactres until we rea <Enter>
    LDR R0,=character
    LDRB R1,[R0]
    CMP R1,#'\n'
    BNE flush_cache
    POP {lr}
    BX lr

manipulation:
    LDRB R1,[R0]
    CMP R1,#'z'      //Compare it with 'z'
    BHI return       //If R1>'z' then we do not make changes
    CMP R1,#'a'      //We compare it with 'a' which means R1<'z'
    SUBGE R1,R1,#32  //If R1>='a' it means 'a'<=R1<='z' and we
substract 32 beavuse Capitals are 32 positions lower in Ascii table
    BGE return       //If it is >='a' then we go to the end or else we
continue

    CMP R1,#'Z'         //In this line we know R1 is not a lower case
letter so we compare it with 'Z'
    BHI return          //If it is greater than Z and it is not a lower
case it means we dont check it
    CMP R1,#'A'         //Compare with 'A' which means R1<'Z'
    ADDGE R1,R1,#32      //This command is excecuted if R1<='Z' and
R1>='A' so R1 is capital and add 32 to make it lower
    BGE return           //If we found it as capital letter go to the
return

    CMP R1,#'9'         //Compare R1 with '9'. If the code is here it
means it is not either lower case or capital
    BHI return          //If R1>'9' and taking into consideration is it
not a letter we are done
    CMP R1,#'5'         //Compare it with '5'
    SUBGE R1,R1,#5      //If R1>='5' it means we have '5'<=R1<='9' and
we sub 4 and we are done
    BGE return

    CMP R1,#'0'         //If we get here R1<'5' so we check it is >=0
    ADDGE R1,R1,#5      //If so we add 5

return:
```

```
    STRB R1,[R0],#1
    CMP R0,R12
    BNE manipulation
    BX lr
```

Αρχικά, στο τμήμα "data" αποθηκεύουμε τις συμβολοσειρές που θέλουμε να εκτυπώσουμε ως κατάλληλα μηνύματα στον χρήστη, μαζί με τα μήκη τους και έναν buffer που θα αποθηκεύει τη συμβολοσειρά (32 χαρακτήρες) και τον χαρακτήρα enter. Τα βασικά labels είναι τα εξής: αρχικά έχουμε το "main", η οποία εκτυπώνει μήνυμα εισαγωγής συμβολοσειράς, με τον register R12 να δείχνει στην πρώτη θέση του buffer. Στο εσωτερικό της main υπάρχει η ρουτίνα "READING_LOOP", η οποία εξετάζει κάθε χαρακτήρα με το scanf, τον μετατρέπει κατάλληλα, ελέγχει αν η συμβολοσειρά είναι 'Q' ή 'q' για να τερματίσει το πρόγραμμα, ή αν η συμβολοσειρά έχει τελειώσει (ή φτάσει τους 32 χαρακτήρες) για την επόμενη. Κάθε χαρακτήρας μετατρέπεται και αποθηκεύεται στην επόμενη θέση του buffer. Σημαντικό είναι ότι κάθε φορά που τελειώνουμε με μια συμβολοσειρά, κάνουμε "flush" την cache για να αδειάσει σε περίπτωση που υπάρχουν παραπάνω από 32 χαρακτήρες.

Οι βασικές συναρτήσεις που χρησιμοποιούνται είναι η printf για τον εκτυπωτή, η scanf για την ανάγνωση εισόδου ενός χαρακτήρα κάθε φορά , η flush_cache για τον έλεγχο της μνήμης cache η οποία διαβάζει τους χαρακτήρες που απέμειναν σε περίπτωση που έδωσε παραπάνω από 32 χαρακτήρες ώστε να αδειάζει η μνήμη , και η manipulation για τη μετατροπή του χαρακτήρα σύμφωνα με την ζητούμενη μέθοδο. Ο κώδικας assembly που παρέχεται με κόκκινα γράμματα χρησιμοποιεί εντολές που εκτελούνται απευθείας αν μια συνθήκη ισχύει, διευκολύνοντας τον κώδικα.

Είναι εμφανές ότι ο συγκεκριμένος κώδικας έχει σχεδιαστεί για arm επεξεργαστές, καθώς χρησιμοποιεί απευθείας εντολές που τους αφορούν. Η χρήση αυτών των εντολών επιτρέπει την εκτέλεση απευθείας συγκρίσεων και πράξεων, χωρίς την ανάγκη για επιπλέον ενδιάμεσες συνθήκες και jumps, κάνοντας τον κώδικα πιο συμπαγή και αποδοτικό.

Στην συνέχεια δημιουργούμε το εκτελέσιμο αρχείο και το τρέχουμε(με το παρακάτω bash script) :

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
  echo "Usage: $0 <filename>"
  exit 1
fi

filename=$1

as -o "${filename}.o" "${filename}.s"
ld -o "${filename}" "${filename}.o"
qemu-arm "./${filename}"
```

Δοκιμάζουμε διάφορες συμβολοσειρές και παρατηρούμε ότι τα αποτελέσματα είναι τα σωστά ,δηλαδή η μετατροπή γίνεται σωστά , το πρόγραμμα δεν αγνοεί τους χαρακτήρες όταν αυτοί είναι πάνω από 32 και η cache δεν υπερχειλίζει ενώ επιπλέον το πρόγραμμα τερματίζει αν και μόνο αν η συμβολοσειρά είσοδος είναι ο χαρακτήρας ‘q’ ή ‘Q’.

Ερώτημα 2ο : Επικοινωνία των guest και host μηχανημάτων μέσω σειριακής θύρας

Σε αυτό το ερώτημα σκοπός της άσκησης είναι να δημιουργήσουμε έναν κώδικα HOST(σε γλώσσα C) ο οποίος θα διαβάζει συμβολοσειρές από τον χρήστη και στην συνέχεια θα της στέλνει στον GUEST(σε γλώσσα assembly) ο οποίος θα λαμβάνει το μήνυμα του host και θα επιστρέφει ποιον χαρακτήρα συνάντησε πιο συχνά σε αυτήν την συμβολοσειρά καθώς και πόσες φορές τον συνάντησε. Συνεπώς έχουμε δύο θέματα που πρέπει να διευθετήσουμε το ένα είναι να διασφαλίσουμε μέσω σωστού configuration την επικοινωνία μεταξύ host και guest και το δεύτερο το string manipulation που πρέπει να γίνει σε arm assembly.

Για να πραγματοποιήσουμε αυτήν την επικοινωνία επιλέξαμε την πρώτη μέθοδο δηλαδή την δημιουργία pseudo terminal.

Τρέχοντας την συγκεκριμένη εντολή στον φάκελο που έχουμε εγκαταστήσει το qemu

```
sudo qemu-system-arm -M versatilepb -kernel vmlinuz-3.2.0-4-versatile -initrd initrd.img-3.2.0-4- versatile -hda debian_wheezy_armel_standard.qcow2 -append "root=/dev/sda1" -net nic -net user,hostfwd=tcp:127.0.0.1:22223-:22 -serial pty
```

Δημιουργούμε λοιπόν ένα virtual machine με arm επεξεργαστή που λειτουργεί ως pseudoterminal και εδώ θα τρέξει ο κώδικα του **guest** που θα λαμβάνει τα μνήματα. Συγκεκριμένα όταν δημιουργούμε το **pseudoterminal(στο qemu)** εμφανίζεται κατάλληλο μήνυμα που εμφανίζει σε ποιο αρχείο γίνεται το αρχείου είναι το /dev/pts/X , όπου το X αλλάζει σε κάθε δημιουργία του VM οπότε πρέπει να αλλάζει κατάλληλα και στον κώδικα. Τέλος με την εντολή dmesg | grep tty βλέπουμε ότι σειριακή θύρα που θα χρησιμοποιήσουμε εμείς (καθώς επιλέγουμε την πρώτη) είναι πάντα η ttyAMA0.

Ο κώδικα του **host** θα τρέξει κανονικά **σε terminal του Linux** και απλώς χρειάζεται να επικοινωνήσει με το κατάλληλο αρχείο (/dev/pts/X) ώστε να έχει πρόσβαση στο αρχείο του pseudoterminal.

Στην συνέχεια θα ασχοληθούμε με το configuration ώστε να διασφαλίσουμε την επικοινωνία host και guest (βοήθησε αρκετά το παρακάτω link https://en.wikibooks.org/wiki/Serial_Programming/termios)

Στην συνέχει ακολουθεί ο κώδικα του host(τρέχει τοπικά στα linux)

```c
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include"termios.h"

int main()
{
    int fd;            //File descriptor used for communication
    char input=NULL;        //Pointer to the user's string in terminal
```

```
    char output[3]; //The most frequent character and the frequency in
this array
    ssize_t counter;
    struct termios tty;
    charc;
    size_t length = 65;

    printf("Input of up to 64 bit to send:\n");
    counter = getline(&input, &length, stdin);

    if (counter<0){
        perror("read");
          exit(1);
    }
    length = counter;
    if (length >65) {
          fprintf(stderr, "The input is too large\n");
           fflush(0);
          exit(1);
    }
    else
        fd = open("/dev/pts/2", O_RDWR | O_NOCTTY);
    if (fd == -1) {
        printf("Failed to open port\n");
        return 1;
    }
    if(tcgetattr(fd, &tty) < 0) {
        printf("Couldn't get the data from the terminal\n");
        return 1;
    }
    tty.c_iflag &= ~(IGNBRK | BRKINT | ICRNL | INLCR | PARMRK | INPCK |
ISTRIP | IXON);

    tty.c_lflag=ICANON;
    tty.c_cflag=CS8|CREAD|CLOCAL;

    if(cfsetispeed(&tty, B9600) < 0 || cfsetospeed(&tty, B9600) < 0) {
        printf("Problem with baudrate\n");
        return 1;
    }

    if(tcsetattr(fd, TCSANOW, &tty) < 0) {
        printf("Couldn't apply settings\n");
         return 1;
    }

    tcflush(fd, TCIOFLUSH);
    write(fd, input, counter);
    read(fd, output,3);
```

```
    int frequency;
    frequency = output[1]-'0';
    printf("The most frequent character is "%c" \nand it appeared %d
times.\n", output[0], frequency);
    close(fd);
    return 0;

}
```

Διαβάζει την συμβολοσειρά που εισάγει ο χρήστης ,έπειτα αφού κάνει τους απαραίτητους ελέγχους για το μήκος της συμβολοσειράς ανοίγει το file descriptor για το αρχείο που βρίσκεται ο pseudoterminal και στέλνει το string στον guest με την εντολή write() ενώ με την συνάρτηση read έχει ως default blocking operation και εμείς δεν αλλάζουμε αυτήν την λειτουργία με κάποιο flag οπότε περιμένει μέχρι να λάβει κάποιο μήνυμα από τον guest στην συνέχεια τυπώνει μήνυμα με τα δεδομένα που έλαβε κλείνει το αρχείο και τερματίζει ο κώδικας. Επίσης, παρατηρούμε τη ρύθμιση των χαρακτηριστικών του τερματικού (baudrate, control flags) μέσω της συνάρτησης tcgetattr() και των σχετικών ρυθμίσεων. Είναι σημαντικό να σημειωθεί ότι στη γραμμή char input=NULL;, το input θα πρέπει να οριστεί ως πίνακας χαρακτήρων, όχι ως απλός χαρακτήρας.

Στην συνέχεια αφού τρέξουμε τον κώδικα host.c (στο linux) τρέχουμε στο qemu τον κώδικα guest.s ώστε να λάβει το μήνυμα και να το επεξεργαστεί κατάλληλα.

```
.global main
.extern tcsetattr

main:
    ldr R0,=path    //Returns the file descriptor for the serial port
    MOV R1,#255 //The file descriptor fd returns in register R0
    ADD R1,R1,#3
    MOV R7,#5
    SWI 0

    MOV R10,R0  //R10 will contain the file descriptor and will not
change

    MOV R1,#0   //Sets the options for the termios
    LDR R2,=options
    BL tcsetattr

    MOV R0,R10  //We read from file descriptor up to 65 characters and
store them in buffer. The last character will always be <Enter>
    LDR R1,=buffer
    MOV R2,#65
    MOV R7,#3
    SWI 0

    LDR R9,=counter //R9 will have the address of the counter
```

```
    MOV R11,#0  //We will use R11 to have the index of the array
    LDR R12,=buffer //We have the address of the buffer in R12

READING_LOOP:
    LDRB R2,[R12,R11] // R2 = Memory[R12+R11] = buffer[R11]
    ADD R11,R11,#1    // R11++ to take all the chars from buffer
    CMP R2,#'\n'      //check if is the end of the string  if not make
all the "NE" else continue
    SUBNE R2,R2,#32   // R2 = R2-32; take the index of the char
    LDRNEB R3,[R9,R2] // R3 = counter[R2]
    ADDNE R3,R3,#1    // R3 ++ ; because we find again this char
    STRNEB R3,[R9,R2] // counter[R2] = R3 , or acutally counter[R2]++
    BNE READING_LOOP  // go to next char of the string

    LDR R9,=counter //R9 will have the address of the counter
    MOV R11,#1  //We will use R11 as the index of array. We strat from
1 because the first is <SPACE> and do not count it
    MOV R0,#0   //In R0 we will keep the frequency
    MOV R1,#0   //In R1 we will keep the index of the character
character
    MOV R2,#0   //R2 will be the max frequency

MAX_LOOP:
    LDRB R0,[R9,R11] // R0 =  counter [R11]
    CMP R0,R2        // if Counter[R11] > MAXfreq
    MOVHI R2,R0      // then MAXfreq = counter[R11]
    MOVHI R1,R11     // then index_of_ascii = R11
    ADD R11,R11,#1   //next Counter element
    CMP R11,#96      // stop when check all ascii characters
    BLT MAX_LOOP

    ADD R1,R1,#32
    LDR R3,=output
    STRB R1,[R3]
    STRB R2,[R3,#1]
    MOV R1,#'\n'
    STRB R1,[R3,#2]

    MOV R0,R10
    LDR R1,=output
    MOV R2,#3
    MOV R7,#4
    SWI 0

End:
    MOV R7,#1
    SWI 0
```

```
.data
 options: .word 0x00000000 /* c_iflag */
             .word 0x00000000 /* c_oflag */
             .word 0x000008b0 /* c_cflag */
             .word 0x00000002 /* c_lflag */
             .byte 0x00       /* c_line */
             .word 0x00000000 /* c_cc[0-3] */
             .word 0x00000000 /* c_cc[4-7] */
             .word 0x00000000 /* c_cc[8-11] */
             .word 0x00000000 /* c_cc[12-15] */
             .word 0x00000000 /* c_cc[16-19] */
             .word 0x00000000 /* c_cc[20-23] */
             .word 0x00000000 /* c_cc[24-27] */
             .word 0x00000000 /* c_cc[28-31] */
             .byte 0x00       /* padding */
             .hword 0x0000    /* padding */
             .word 0x0000000d /* c_ispeed */
             .word 0x0000000d /* c_ospeed */



path: .asciz "/dev/ttyAMA0"
buffer: .space 65
output: .space 3
counter: .ascii
"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"  //We will have a counter for every
possible printable ascii code
```

Αρχικά στην main γίνεται το αντίστοιχο configuration ώστε να διαβάσουμε το μήνυμα που έστειλε αυτό το κάνουμε με .extern tcsetattr ώστε να ορίσουμε σωστά τις ρυθμίσεις επικοινωνίας και μετα διαβάζουμε το string που έστειλε ο host και το αποθηκεύομε σε ένα buffer.Η βασική μετατροπή βρίσκεται στις ρουτίνες reading_loop και Max_loop.

reading_loop: επεξεργάζεται κάθε χαρακτήρα της συμβολοσειράς μέχρι να εντοπίσει τον χαρακτήρα newline ('\n'). Κάθε χαρακτήρας υποβάλλεται σε μια διαδικασία μετατροπής (σύμφωνα με τον κώδικα) και αυξάνεται η συχνότητα εμφάνισης του συγκεκριμένου χαρακτήρα.Ο counter είναι ένας buffer που περιέχει τελικά την συχνότητα όλων των χαρακτήρων ascii από το συγκεκριμένο string .

Max_loop: Βρίσκει από τον πίνακα που περιέχει τις συχνότητες των χαρακτήρων ποιο index έχει την μεγαλύτερή συχνότητα συγκρίνοντας εξαντλητικά όλους τους χαρακτήρες και στέλνει με system call τα δεδομένα στον host.

Παρατηρούμε ότι όντως οι κώδικες λειτουργούν σωστά αρκεί να τρέξουμε πρώτα τον host και στην συνέχεια τον guest.

Ερώτημα 3ο : Σύνδεση κώδικα C με κώδικα assembly του επεξεργαστή ARM

Σε αυτό το ζητούμενο σκοπός είναι να υλοποιήσουμε κάποιες βασικές συναρτήσεις της C και συγκεκριμένα της βιβλιοθήκης string.h σε assembly και να δούμε ότι έπειτα ο κώδικας της c λειτουργεί κανονικά.

Για να το κάνουμε αυτό θα φτιάξουμε ένα αρχείο assembly

```
.text
.align 4
.type strlen %function
strlen:
  MOV R1, R0      // Copy the address of the input string to R1
  MOV R0, #0      // Initialize the length counter to 0

 strlen_loop:
  LDRB R2, [R1], #1  // Load the byte at the address in R1 and
increment the address
   CMP R2, #0        // Compare the loaded byte with null terminator
   ADDNE R0,R0,#1
   BNE strlen_loop
   BXEQ LR     // return with the length in R0 regiser if end of
terminal

.align 4
.global strcpy
.type strcpy  %function
strcpy:
  //RO = output_file , R1 = input_file
  strcpy_loop:
   LDRB R2, [R1], #1  // Load the byte at the address in R1 and
icrement the address
   STRB R2,[R0],#1    // store the value in the register R0 (pointer
   CMP R2, #0        // Compare the loaded byte with null terminator
   BNE  strcpy_loop
   BX LR     // return with register R0,R1

.align 4
.global strcat
.type strcat  %function
strcat:
  // input: RO = destination(string1) , R1 = source(string2)
  // output:RO = string1+string2 , R1=string2

 // find the length of str1 -- go to the end of that string
 len_loop:
 LDRB R2,[R0],#1
```

```
 CMP R2,#0
 BNE len_loop

 SUB R0,R0,#1 // go string1 before \n at the last charachter of string1

 copy_loop:
 LDRB R2,[R1],#1
 STRB R2,[R0],#1
 CMP R2,#0
 BNE copy_loop

 BX LR

 .align 4
.global strcmp
.type strcmp  %function
strcmp:
   // input R0 = str1  R1 =str2
   // return 0 if the string are equal (the reutrn value is stored at
register R0)
   // >0 if str1 >str2   else < 0
   compare_loop:
   CMP R2,#'\n' // check if both of the string has finished
   BEQ Finish
   LDRB R2,[R0],#1 // R2 = str1[i] i++
   LDRB R3,[R1],#1 // R3 =str2[j] j++
   CMP R2,R3
   BEQ compare_loop

   CMP R2,R3
   BGT  RETURN_1       //if str1[i] > str2[i]
   MOV R0,#-1
   BX LR
   RETURN_1:
   MOV R0,#1
   BX LR

Finish:
 MOV R0,#0
 BX LR
```

Αρχικά σημειώνουμε ότι χρειάζονται οι εντολές
.global function_name
 .type function_name %function
Ώστε να μπορεί να γίνει η σύνδεση με το C κώδικα στην συνέχεια.

Οι συναρτήσεις που υλοποιούνται είναι αρκετά απλές και περιγράφονται πλήρως στα σχόλια οπότε δεν θα σχολιάσουμε την λειτουργία και την λογική τους. Αξίζει όμως να σημειωθεί ότι τα ορίσματα της συνάρτησης που καλείται στην C δίνονται με την σειρά στους registers R0,R1,R2 ,… και ομοίως αυτό που επιστρέφει η συνάρτηση πρέπει να αποθηκευτεί στον register R0.

Κάνοντας λοιπόν extern τις συναρτήσεις στον κώδικα C που μας δίνεται είμαστε έτοιμοι .

```c
#include <stdlib.h>
#include <stdio.h>
// string.h removed since functions are defined by user
// #include <string.h>

// define all fucntions to be replaced as extern
extern size_t strlen(const char *c);
extern char *strcpy(char *char1, const char *char2);
extern char *strcat(char *char1, const char *char2);
extern int strcmp(const char *char1, const char *char2);
// end of defintion
```

(ο υπόλοιπος κώδικας μένει ίδιος όπως μας δόθηκε)

Στην συνέχεια φτιάχνουμε και ένα bash script για να κάνει build και link τα δύο αρχεία κατάλληλα.(build_and_link.sh)

```bash
#!/bin/bash

if [ "$#" -ne 2 ]; then
  echo "Usage: $0 <c_file> <assembly_file>"
  exit 1
fi

c_file="$1"
assembly_file="$2"
output_file="${c_file%.*}"  # Extracting the file name without
extension

# Compile C code (-c flag)
gcc -Wall -g -c "$c_file" -o "${output_file}.o"

# Compile assembly code (-c flag)
as -o "${output_file}_asm.o" "$assembly_file"

# Link object files
gcc -o "${output_file}_program" "${output_file}.o"
"${output_file}_asm.o"

# Clean up object files (optional)
# rm "${output_file}.o" "${output_file}_asm.o"

echo "Build completed. Executable: ${output_file}_program"
```

Επομένως μένει να εξετάσουμε κατά πόσο οι συναρτήσεις της assembly λειτουργούν σωστά για να το κάνουμε αυτό αρχικά τρέχουμε με τις συναρτήσεις σε C τα αρχεία με rand_str_input_first.txt και rand_str_input_sec.txt αυτό παράγει τρία αρχεία για κάθε txt τα μετονομάζουμε κατάλληλα ώστε να ξέρουμε ότι αυτά προέρχονται από τις συναρτήσεις της C.Στην συνέχεια τρέχουμε κατάλληλα το πρόγραμμα με τις συναρτήσεις που έχουν υλοποιηθεί σε assembly και παράγονται και πάλι 3 αποτελέσματα για κάθε txt .Συγκρίνουμε λοιπόν τις εξόδους της C με τις εξόδους της Assembly και παρατηρούμε ότι έλιναι ακριβώς ίδες επομένως συναρτήσεις μας σε assembly λειτουργούν κανονικά

Για να το κάνουμε αυτό φτιάχνουμε bash script που τσεκάρε αν δύο αρχεία out είναι ίδια

```
#!/bin/bash

if [ $# -ne 2 ]; then
    echo "Usage: $0 <file1.out> <file2.out>"
    exit 1
fi

file1=$1
file2=$2

# Use diff command to compare the contents of the two files
diff "$file1" "$file2" > /dev/null

if [ $? -eq 0 ]; then
    echo "Files are the same."
else
    echo "Files are different."
fi
```

*Όλοι οι κώδικες και τα script έχουν παραδοθεί μαζί με την αναφορά σε zip folder.