



**ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**  
**ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ**  
**ΥΠΟΛΟΓΙΣΤΩΝ**

Τομέας Τεχνολογίας Υπολογιστών και Υπολογιστών  
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Σχεδιασμός Ενσωματωμένων Συστημάτων  
9<sup>ο</sup> Εξάμηνο ΗΜΜΥ

**5η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ**

**Εργασία σε assembly του επεξεργαστή ARM**

Ημ/νια Παράδοσης : 10/01/2025

Ομάδα 22

Παναγιώτης Μπέλσης AM : 03120874

Θεοδώρα Εξάρχου AM : 03120865

**Ερώτημα 1ο: Μετατροπή εισόδου από τερματικό**

Ο σκοπός της άσκησης είναι η ανάπτυξη μιας εφαρμογής σε ARM Assembly, η οποία θα επεξεργάζεται μια συμβολοσειρά εισόδου από το τερματικό, πραγματοποιώντας συγκεκριμένες μετατροπές στους χαρακτήρες της. Η εφαρμογή θα ελέγχει αν οι χαρακτήρες είναι κεφαλαία ή πεζά γράμματα, και θα μετατρέπει αριθμούς σύμφωνα με έναν καθορισμένο πίνακα που μας δόθηκε. Επίσης, η εφαρμογή θα συνεχίζει να εκτελείται μέχρι να ληφθεί είσοδος που να αποτελείται από τον χαρακτήρα 'Q' ή 'q', οπότε θα τερματίσει.

Παρακάτω παραθέτουμε τον κώδικα σε ARM assembly

```
.global _start

_start:
    mov r7, #4      @syscall for write
    mov r0, #1      @file descriptor 1(stdout)
    ldr r1, =msg
    mov r2, #38
    svc 0

    mov r7, #3      @syscall for read
    mov r0, #0      @file descriptor 0(stdin)
    ldr r1, =buffer
    mov r2, #32      @max number of bytes to read
    svc 0

    @ check if buffer_size==2, "q" or "Q" + \n

    mov r4, r0      @ after svc 0, r0 has the buffer size
    cmp r4, #2
    bne process_buffer
    ldrb r3, [r1]    @ Load the first byte of the input into r3
    cmp r3, #'q'     @ Compare if the input byte is 'q'
    beq _exit
```

```

        cmp r3, #'Q'    @ Compare if the input byte is 'Q'
        beq _exit

process_buffer:
        mov r2, #0      @ current byte index = 0

process_loop:
        cmp r2, r4
        bge write_output

        add r5, r1, r2 @ Calculate the address(buffer + index)
        ldrb r3, [r5]  @ Load the byte at the current index into r3

        bl convert      @ call the function that converts chars

        add r2, r2, #1 @ index++
        b process_loop @ loop start

write_output:
        mov r7, #4      @syscall for write
        mov r0, #1      @file descriptor 1(stdin)
        ldr r1, =buffer
        mov r2, r4      @max number of bytes to read
        svc 0

        b _start        @restart the program

_exit:
        mov r7, #1
        mov r0, #0
        svc 0

@Function convert_char
.align 4
.global convert
.type convert, %function

convert:
        cmp r3, #65     @check if r3 < "A"
        blt lower_check
        cmp r3, #90     @check if r3 > "Z"
        bgt lower_check
        add r3, r3, #32 @convert capital letter to small
        strb r3, [r5]   @ Store the modified byte back into the buffer
        bx lr          @ Return from function

lower_check:
        cmp r3, #97     @check if r3 < "a"
        blt num_check0_4
        cmp r3, #122    @check if r3 > "z"
        bgt num_check0_4
        sub r3, r3, #32 @convert small letter to capital
        strb r3, [r5]   @ Store the modified byte back into the buffer
        bx lr          @ Return from function

num_check0_4:
        cmp r3, #'0'    @check if r3 < '0'
        blt num_check5_9
        cmp r3, #'4'    @check if r3 > '4'
        bgt num_check5_9
        add r3, r3, #5   @'0' -> '5', etc.
        strb r3, [r5]   @ Store the modified byte back into the buffer
        bx lr          @ Return from function

```

```

num_check5_9:
    cmp r3, #'5'    @check if r3 < '5'
    blt no_change
    cmp r3, #'9'    @check if r3 > '9'
    bgt no_change
    sub r3, r3, #5   @'5' -> '0', etc.
    strb r3, [r5]    @ Store the modified byte back into the buffer

no_change:
    bx lr           @ Return from function

@end function
.data
msg:    .ascii "Input a string of up to 32 chars long:"
buffer: .space 32   @allocate 32bytes for input storage

```

Από τα σχόλια του κώδικα εξηγείται πλήρως το σκεπτικό μας.

Για την αλλαγή των αριθμών όπως ζητείται στην εκφώνηση η αρχική ιδέα ήταν για την μετατροπή να κάνουμε +5 στο input και έπειτα mod 10 (αφού γίνουν κανονικοποιήσεις από χαρακτήρες σε νομμερα), αλλά το arm νη που φτιάξαμε δεν υποστήριζε την εντολή udiv, ώστε να μπορούσαμε στη συνέχεια να πάρουμε το mod. Συνεπώς λύθηκε όπως φαίνεται παραπάνω

## Makefile

```

# Variables
TARGET = test
OBJ = test.o
SRC = test.s

# Default target
all: $(TARGET)

# Assemble and link
$(TARGET):
    as -o $(OBJ) $(SRC)
    ld -o $(TARGET) $(OBJ)

# Clean up build files
clean:
    rm -f $(OBJ) $(TARGET)

```

```

root@debian-armel:/home/user/ex3# cd ..
root@debian-armel:/home/user# ls
ex3 Makefile test test.o test.s
root@debian-armel:/home/user# vim test.s
root@debian-armel:/home/user# ./test
Input a string of up to 32 chars long:q!@#~`AB!Cabc!
Q!@#~`ab!cABC!
Input a string of up to 32 chars long:

```

```

root@debian-armel:/home/user# ls
ex3 Makefile test test.o test.s
root@debian-armel:/home/user# vim test.s
root@debian-armel:/home/user# ./test
Input a string of up to 32 chars long:q!@#~`AB!Cabc!
Q!@#~`ab!cABC!
Input a string of up to 32 chars long:q
root@debian-armel:/home/user#

```

Το πρόγραμμα τερματίζει μόνο όταν το input είναι 1 char και αυτό είναι 'q','Q'. Αν το string μας ξεκινάει απλά με 'q' δεν τερματίζεται η λειτουργία.

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

Για την υλοποίηση της άσκησης, επιλέξαμε τη μέθοδο χρήσης της εικονικής σειριακής θύρας μέσω του εργαλείου QEMU με την επιλογή `-serial pty`. Έτσι, μπορούμε να στείλουμε δεδομένα από το πρόγραμμα στο host στο πρόγραμμα του guest, το οποίο στη συνέχεια επεξεργάζεται τα δεδομένα και επιστρέφει την απαραίτητη απάντηση.

### **1. Εκκίνηση qemu με προσθήκη `-serial pty`**

```
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
root@debian-armel:~# dmesg | grep tty
```

2. Το qemu θα κάνει τις απαραίτητες ενέργειες για την δημιουργία του αρχείου στο `/dev/pts/` από αυτό το command line βλέπουμε ποια θύρα είναι ενεργή: **`/dev/pts/3`**

Ο αριθμός 3 είναι μεταβλητός για κάθε νέα εκκίνηση μπορεί να αλλάζει. Αυτό το αρχείο θα χρησιμοποιήσουμε στο host μηχανήμα ως το ένα άκρο της σειριακής.

```
dora@DESKTOP-J824248:~/qemu/arm_vm_lab5$ sudo qemu-sys
nitrd.img-3.2.0-4-versatile -hda debian_wheezy_armel_s
tcp:127.0.0.1:22223-:22 -serial pty
[sudo] password for dora:
char device redirected to /dev/pts/3 (label serial0)
```

3. Συνδεση στο qemu με το root και βλέπουμε ποιες είναι οι διαθέσιμες σειριακές θύρες για το guest

```
root@debian-armel:~# dmesg | grep tty
[ 0.000000] console [tty0] enabled
[ 0.168499] dev:f1: ttyAMA0 at MMIO 0x101f1000 (irq = 12) is a PL011 rev1
[ 0.172382] dev:f2: ttyAMA1 at MMIO 0x101f2000 (irq = 13) is a PL011 rev1
[ 0.172985] dev:f3: ttyAMA2 at MMIO 0x101f3000 (irq = 14) is a PL011 rev1
[ 0.173548] fpga:09: ttyAMA3 at MMIO 0x10009000 (irq = 38) is a PL011 rev1
```

Εμείς θα χρησιμοποιήσουμε την πρώτη `ttyAMA0` για την άσκηση αυτή. Παράλληλα απενεργοποιήσαμε και το πρόγραμμα `getty` για τη σειριακή θύρα `ttyAMA0`, όπως αναφερόταν στις οδηγίες.

### **Για Host :**

Υλοποίηση κώδικα σε C.

Για να μην απαιτείται αλλαγή στον κώδικα, περνάμε ως όρισμα σε κάθε εκτέλεση το `/dev/pts/2`. Το εκτελούμε με την εντολή : `./host_serial /dev/pts/3`

Για να τρέξει σωστά η επικοινωνία, ακολουθούμε τα εξής βήματα:

1. Στο host τρέχουμε το αρχείο `host_serial.c`
2. Στο guest τρέχουμε το αρχείο `guest_serial.c`

### 3. Γράφουμε ένα string στο host.

Έχουμε ορίσει το MAX\_SIZE=65, καθώς υπολογίζουμε και το τελευταίο Enter που θα δώσουμε, το οποίο καταμετράται στο length.

#### host\_serial.c

```
#include <termios.h> // Header file for terminal I/O control, used to configure serial ports
#include <stdio.h>
#include <fcntl.h>    // Library for file control, such as open()
#include <unistd.h>   // write(), read(), close()
#include <string.h>
#include <errno.h>    // Error integer and strerror() function
#include <stdlib.h>
#define MAX_SIZE 65 // Maximum size for the string input

int main(int argc, char **argv) {
    // Declare variables
    const char *port;
    int serial_port;
    struct termios tty;
    char *input = NULL;
    size_t length = MAX_SIZE; // Maximum size of input
    char response[2]; // response[0] = char, response[1] = count

    // Check if the port is provided as an argument
    if (argc < 2) {
        perror("Error. No port specified\n");
        return 1; // Exit if no port is provided as an argument
    }

    // Prompt for user input
    printf("Please give a string to send to host: ");
    // Get the input string from the user
    ssize_t line_length = getline(&input, &length, stdin);
    if (line_length < 0) {
        fprintf(stderr, "Error while reading input: %s\n", strerror(errno));
        return 1;
    }

    // Check if input length exceeds MAX_SIZE
    if (line_length > MAX_SIZE) {
        fprintf(stderr, "The input is too large. Please enter a string with up to 64 characters.\n");
        fflush(stderr); // Flush stderr and immediately display the message
        return 1;
    }

    // Port passed via command-line argument
    port = argv[1];
    serial_port = open(port, O_RDWR | O_NOCTTY); // Open the specified port

    // Error handling for opening the port
    if (serial_port < 0) {
        printf("Error %i from open: %s\n", errno, strerror(errno));
        return 1; // Return or handle error appropriately
    }

    // Set baud rate (input and output)
    cfsetispeed(&tty, B9600); // Set input baud rate to 9600
    cfsetospeed(&tty, B9600); // Set output baud rate to 9600
```

```

tty.c_iflag &= ~(IGNBRK | BRKINT | ICRNL | INLCR | PARMRK | INPCK | ISTRIP | IXON);
// // Output flags - Turn off output processing
// // no CR to NL translation, no NL to CR-NL translation,
// no NL to CR translation, no column 0 CR suppression,
// no Ctrl-D suppression, no fill characters, no case mapping,
// no local output processing //

tty.c_lflag=ICANON;

tty.c_cflag=CS8|CREAD|CLOCAL; // 8n1, see termios.h for more information

// Apply the changes to the serial port configuration
if (tcsetattr(serial_port, TCSANOW, &tty) != 0) {
    printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
    return 1; // Error handling
}

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

tcflush(serial_port, TCIOFLUSH);

// Send the input data to the terminal
write(serial_port, input, line_length);

// Read the response from the guest
int check_read = read(serial_port, response, sizeof(response)); // Read the response

// Check if the number of bytes read is at least 2
if (check_read < 2) {
    printf("Error: Invalid or insufficient data received from the serial port. Expected at least 2 bytes.\n");
    close(serial_port);
    return 1;
}

// Display the result
printf("The most frequent character is '%c' and it appeared %d times.\n", response[0],
response[1] - '0');
//response[1] - '0' is converting the ASCII character of a digit into its corresponding integer
value.

// Close the serial port
close(serial_port);
free(input);
return 0;}

```

### Επεξήγηση κώδικα:

Ανοίγουμε μια σειριακή θύρα που ορίζεται μέσω παραμέτρου και επιτρέπει στον χρήστη να εισάγει ένα string, το οποίο αποστέλλεται μέσω της θύρας /dev/pts/3.

Χρησιμοποιούμε τα flags

- O\_RDWR για να ανοίξουμε τη θύρα για ανάγνωση και εγγραφή
- O\_NOCTTY για να αποτρέψουμε την ανάληψη της θύρας ως "επικεφαλής" του terminal.

Η συνάρτηση fflush(stderr) διασφαλίζει ότι το μήνυμα σφάλματος θα εκτυπωθεί άμεσα στο τερματικό, χωρίς καθυστερήσεις.

Ρυθμίζουμε τις βασικές παραμέτρους της σειριακής θύρας, ορίζοντας την ταχύτητα μετάδοσης (baud rate) σε 9600 bits (μία από τις πιο κοινές τιμές) για εισαγωγή και εξαγωγή δεδομένων με τις συναρτήσεις `cfsetispeed()` και `cfsetospeed()`.

Όλες οι αλλαγές που κάνουμε εφαρμόζονται με τη συνάρτηση `tcsetattr()` στη σειριακή θύρα.

Με τη χρήση του `TCIOFLUSH` διασφαλίζουμε ότι τόσο τα εισερχόμενα όσο και τα εξερχόμενα δεδομένα στη σειριακή θύρα διαγράφονται, ώστε να αποφεύγονται ανεπιθύμητα δεδομένα ή καθυστερήσεις στην επικοινωνία.

Η συνάρτηση `write(serial_port, input, line_length)` αποστέλλει τα δεδομένα από το buffer `input` στη σειριακή θύρα.

Η συνάρτηση `read(serial_port, response, sizeof(response))` διαβάζει τα δεδομένα που επιστρέφονται από τη σειριακή θύρα και τα αποθηκεύει στον πίνακα `response`.

Από τον πίνακα `response` παραλαμβάνουμε από το `guest` τα εξής:

`response[0]` = char και `response[1]` = count, πόσες φορές συναντάμε τον χαρακτήρα. Κανούμε χρήση του `response[1]` - '0' για κανονικοποίηση του char σε ακέραιο αριθμό.

Η χρήση της `close(serial_port)` κλείνει τη σειριακή θύρα, ενώ η `free(input)` απελευθερώνει τη δυναμική μνήμη που χρησιμοποιήθηκε για την είσοδο του χρήστη, αποφεύγοντας διαρροές μνήμης.

## Για GUEST

Για το `guest`, εντός του `root`, δημιουργούμε το αρχείο μας σε assembly και το μεταγλωττίζουμε με τις παρακάτω εντολές:

```
as -o guest_serial.o guest_serial.s
ld -o guest_serial guest_serial.o -lc -dynamic-linker /lib/ld-linux.so.3
```

Μετά, εκτελούμε το αρχείο με την εντολή: `./guest_serial`

Για την αρχιτεκτονική ARM έχουμε τα εξής :

Για τους Registers

- Registers R0-R6 και R8-R10 είναι γενικής χρήσης.
- Register R7 χρησιμοποιείται για να αποθηκεύει τον αριθμό της συστημικής κλήσης (syscall number), ο οποίος καθορίζει ποια λειτουργία του λειτουργικού συστήματος θα εκτελεστεί.
- Registers R11 χρησιμοποιείται ως frame pointer, δηλαδή για τη διαχείριση της στοίβας (stack) και την αναγνώριση του σημείου αναφοράς (frame) των τοπικών μεταβλητών στη συνάρτηση.

Για τα System Call:

### System Call Quick Reference

No	Func Name	Description
1	<a href="#">exit</a>	terminate the current process
2	<a href="#">fork</a>	create a child process
3	<a href="#">read</a>	read from a file descriptor
4	<a href="#">write</a>	write to a file descriptor
5	<a href="#">open</a>	open a file or device
6	<a href="#">close</a>	close a file descriptor
7	<a href="#">waitpid</a>	wait for process termination

## guest\_serial.s

```
.global main
.extern tcsetattr      // declares external function
main:
    ldr R0,=path        // Fd for the serial port
    MOV R1,#255         //
    ADD R1,R1,#3
    MOV R7,#5           // System call number for open (5)
    SWI 0               // Software interrupt
    MOV R10,R0          // Store the file descriptor in R10 for later use
Termios_options:
    MOV R1,#0           // Set R1 to 0 (sets options for the terminal)
    LDR R2,=options     // Load the address of the terminal options into R2
    BL tcsetattr        // Call tcsetattr to set the terminal attributes with options
    MOV R0,R10          // Load the file descriptor (R10) back into R0

    LDR R1,=buffer      //Load the address of the buffer into R1

Buffer_set:
    MOV R2,#65          // Set R2 to 65 (maximum number of characters to read)
    MOV R7,#3           // System call number for read (3)
    SWI 0
    LDR R9,=freq_table  // Load the address of the frequency table into R9
    MOV R11,#0          // Set R11 to 0,
    LDR R12,=buffer     // Load the address of the buffer into R12
READING_LOOP:
    LDRB R2,[R12,R11]   // R2 = Memory[R12+R11] = buffer[R11]
    ADD R11,R11,#1      // Increment R11 to process the next byte in the buffer
    CMP R2, #'\\n'      // Compare the character in R2 with the newline character
Processing_Char:
    SUBNE R2,R2,#32     // If the character is not newline, subtract 32 from it (adjust ASCII ange)
    LDRNEB R3,[R9,R2]   // R3 = freq_table[R2]
    ADDNE R3,R3,#1      // R3 ++
    STRNEB R3,[R9,R2]   // Store the updated frequency back to the frequency table
    BNE READING_LOOP   // Go to the next char of the string

    LDR R9,=freq_table  // R9 will have the address of the frequency table
    MOV R11,#1          // Start from index 1 to skip the space character
    MOV R0,#0           // In R0 we will keep the frequency
    MOV R1,#0           // In R1 we will keep the index of the character
    MOV R2,#0           // R2 will be the max frequency

MAX_LOOP:
    LDRB R0,[R9,R11]    // R0 = freq_table [R11]
    CMP R0,R2           // if freq_table[R11] > MAXfreq
    MOVHI R2,R0         // then MAXfreq = freq_table[R11]
    MOVHI R1,R11        // then index_of_ascii = R11
    ADD R11,R11,#1      // next freq_table element
    CMP R11,#96         // stop when checking all ASCII characters (0-255)
    BLT MAX_LOOP        // Loop if not done

    ADD R1,R1,#32        // Convert index to character (adding 32)
    LDR R3,=response    // Load the address of the response buffer
    STRB R1,[R3]
```



[illegible]

Επεξήγηση κώδικα:

Ο κώδικας ξεκινά ανοίγοντας τη σειριακή θύρα για επικοινωνία μέσω του port `/dev/ttyAMA0`. Πρώτα, φορτώνουμε τη διεύθυνση του αρχείου (δηλαδή τη διαδρομή του σειριακού port) στον καταχωρητή R0. Στη συνέχεια, ρυθμίζουμε τον καταχωρητή R7 με την τιμή 5, η οποία αντιστοιχεί στο system call `sys_open`, η οποία χρησιμοποιείται για το άνοιγμα αρχείων.

H εντολή SWI 0 εκτελεί το system call για το άνοιγμα του αρχείου. Αφού ανοίξουμε το σειριακό port, προχωράμε στη ρύθμιση των παραμέτρων του τερματικού. Φορτώνουμε τη διεύθυνση της μεταβλητής options στον καταχωρητή R2 και καλούμε τη συνάρτηση tcsetattr, η οποία εφαρμόζει αυτές τις ρυθμίσεις στο σειριακό port, προκειμένου να διασφαλίσουμε τη σωστή επικοινωνία με το host.

Για να διαβάσουμε τους χαρακτήρες από το ανοικτό σειριακό port, θέτουμε τον καταχωρητή R7 με την τιμή 3, η οποία αντιστοιχεί στο system call `sys_read`, η οποία χρησιμοποιείται για την ανάγνωση δεδομένων.

Δηλώνουμε το `buffer` για να αποθηκεύσουμε το `Input` και το `freq_table` που θα αποθηκεύει τη συχνότητα εμφάνισης κάθε χαρακτήρα.

Στο βρόχο `READING_LOOP`, το πρόγραμμα διαβάζει κάθε χαρακτήρα από το `buffer` και τον ελέγχει.

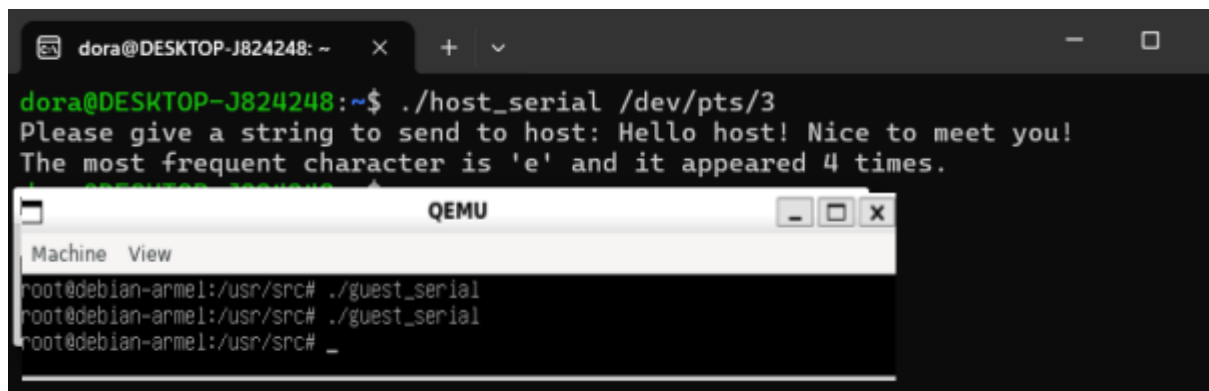
Αν ο χαρακτήρας είναι το τέλος της γραμμής (newline, ASCII 10), η επεξεργασία σταματά. Αν δεν είναι newline, συνεχίζεται η επεξεργασία του χαρακτήρα. Για κάθε χαρακτήρα που δεν είναι newline, αφαιρούμε 32 από την τιμή του ASCII για κανονικοποίηση σε ακέραιο. Η αφαίρεση αυτή μετατρέπει το κενό (ASCII 32) σε 0, το χαρακτήρα '!' (ASCII 33) σε 1, το χαρακτήρα '"' (ASCII 34) σε 2, και ούτω καθεξής. Έτσι, οι χαρακτήρες τοποθετούνται στον πίνακα συχνοτήτων freq\_table στο εύρος από 0 έως 96, με το κενό να καταλαμβάνει την πρώτη θέση (0) και τους άλλους χαρακτήρες να ακολουθούν. Με αυτό τον τρόπο, η επεξεργασία των χαρακτήρων γίνεται πιο εύκολη, καθώς το κενό είναι στην αρχή και δεν χρειάζεται να αγνοηθεί στη συνέχεια.

Ο δείκτης R11 αρχικοποιείται με MOV R11, #1, ώστε να ξεκινάμε την επεξεργασία του πίνακα από το δεύτερο στοιχείο (θέση 1), παραλείποντας το κενό (ASCII 32) που τοποθετείται στη θέση 0 του πίνακα.

Στον βρόχο MAX\_LOOP, ο κώδικας εξετάζει κάθε στοιχείο του πίνακα συχνοτήτων για να εντοπίσει τον χαρακτήρα με την υψηλότερη συχνότητα εμφάνισης. Συγκρίνει κάθε τιμή του πίνακα με την τρέχουσα μέγιστη συχνότητα. Αν βρει μεγαλύτερη τιμή, ενημερώνει τη μέγιστη συχνότητα και το αντίστοιχο index του χαρακτήρα.

Στο τέλος της διαδικασίας, τα αποτελέσματα (δηλαδή ο χαρακτήρας με τη μεγαλύτερη συχνότητα και η συχνότητα αυτή) αποθηκεύονται στο response χρησιμοποιώντας την εντολή SWI 4 (write system call).

Τέλος τα αποτελέσματα αποστέλλονται στον host.



```
dora@DESKTOP-J824248: ~  
dora@DESKTOP-J824248:~$ ./host_serial /dev/pts/3  
Please give a string to send to host: Hello host! Nice to meet you!  
The most frequent character is 'e' and it appeared 4 times.  
QEMU  
Machine View  
root@debian-armel:/usr/src# ./guest_serial  
root@debian-armel:/usr/src# ./guest_serial  
root@debian-armel:/usr/src# _
```

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

Για τη υλοποίηση των functions ακολουθήσαμε το man string της C.

char \*strcpy(char \*dest, const char \*src);  
Copy the string src to dest, returning a pointer to the start of dest.

char \*strcat(char \*dest, const char \*src);  
Append the string src to the string dest, returning a pointer dest.

int strcmp(const char \*s1, const char \*s2);  
Compare the strings s1 with s2.

size\_t strlen(const char \*s);

Return the length of the string s.

## functions.s

```
.text
.align 4
.global strlen
.type strlen, %function

strlen:
    stmfd sp!, {lr} @save the link register on the stack
    mov r1, r0      @in r0 is the string s
    mov r2, #0      @counter=0

loop:
    ldrb r3, [r1], #1 @load byte in r3, increment r1
    cmp r3, #0       @check if current char is NULL
    beq end
    add r2, r2, #1   @counter++
    b loop

end:
    mov r0, r2      @return the size of string
    ldmfd sp!, {lr} @restore the link register from the stack
    bx lr          @return to caller


.text
.align 4
.global strcpy
.type strcpy, %function

strcpy:
    stmfd sp!, {lr} @save the link register on the stack
    mov r2, r1

loop2:
    ldrb r3, [r2], #1 @ Load byte from source (r2) into r3, increment r2
    strb r3, [r0], #1 @ if the byte is NULL i want it to be stored
    cmp r3, #0
    bne loop2
    ldmfd sp!, {lr} @restore the link register from the stack
    bx lr          @return to caller


.text
.align 4
.global strcat
.type strcat, %function

strcat:
    stmfd sp!, {lr} @save the link register on the stack
    mov r2, #0 @ Initialize r2 to 0

dest_loop:
    ldrb r2, [r0, #0] @Load byte from dest(r0) into r2
    cmp r2, #0 @ Check if the byte is NULL (end of dest)
    beq src_loop
    add r0, r0, #1 @ Increment the pointer
    b dest_loop @ continue searching for the NULL terminator

src_loop:
    ldrb r2, [r1], #1 @ Load byte from src(r1) into r2, increment r1
    strb r2, [r0], #1 @ if the byte is NULL i want it to be stored
    cmp r2, #0 @ Compare the byte with NULL terminator
    bne src_loop @ If it's not NULL, continue looping
    ldmfd sp!, {lr} @restore the link register from the stack
    bx lr          @return to caller
```

```

.text
.align 4
.global strcmp
.type strcmp, %function

strcmp:
    stmfd sp!, {lr} @save the link register on the stack
read_and_compare_loop:
    ldrb r2, [r0], #1      @ Load byte from s1 (r0) into r2, increment r0
    ldrb r3, [r1], #1      @ Load byte from s2 (r1) into r3, increment r1
    cmp r2, r3
    bne min_max

    cmp r2, #0             @check if we read the whole string
    beq equal
    b read_and_compare_loop
min_max:
    cmp r2, r3             @the first non-matching char is greater or lower than r3
    bgt positive           @branch to positive if greater, else continue(to negative)

    mov r0, #0xFFFFFFFF    @ Load r0 with -1
    ldmfd sp!, {lr} @restore the link register from the stack
    bx lr                  @return to caller (r2<r3)
positive:
    mov r0, #1             @ Load r0 with +1
    ldmfd sp!, {lr} @restore the link register from the stack
    bx lr
equal:
    mov r0, #0             @ Strings are equal
    ldmfd sp!, {lr} @restore the link register from the stack
    bx lr                  @return to caller

```

## Makefile

```

FLAGS = -Wall -g -c
AS = as
CC = gcc

ASM_SRC = functions.s
C_SRC = string_manipulation.c
ASM_OBJ = functions.o
C_OBJ = string_manipulation.o
OUTPUT = string_manipulation.out

all: $(OUTPUT)

$(OUTPUT): $(C_OBJ) $(ASM_OBJ)
    $(CC) -o $@ $^

# C code
$(C_OBJ): $(C_SRC)
    $(CC) $(FLAGS) -o $@ $<

# ARM assembly code
$(ASM_OBJ): $(ASM_SRC)
    $(CC) $(FLAGS) -o $@ $<

clean:

```

```
rm -f $(ASM_OBJ) $(C_OBJ) $(OUTPUT)
:
```

rand\_str\_input\_sec.txt      concat

```
ctykifdytYabsdctXicafiquntubulqrnbxguwssuenvemlxmfionvfnlxdrut
ipltztzrjjvghlaehhqqjaszrovzkpuamc
msxebhlhgcccuzgzjqwtrsrj
oybaowcuejpaeqwqpzedsctmwykotcmdshntnhxwlfuhnlznekdxbhoknjbtlgluzwdtrtzlcvepkdcyc
pjpudlwmxfvfrgsvparvzwxwhwtbtusybhmxcrzaepivqmjjjaafxqtcejggzzegiaarxniiyyhwdfurvmrhyzjqwzowbdnissonqhuhwbzrbzjjxqxjduuwfjli
behnyxecbcnpgjnmrgpwxehxxwjamqsfoiuiyqdiifoghc
sbfvyxawngjhyoxtpbhbiterdriepwepeyfcxptztctzgluujuzdirrxhcmYhjzvezvwqaxrycmidrp11lgumiobvafiqeuhrookvvvzj
tkinuqapzvrjagugscmlroqsyrzzyhnxharodluyetgoanlpxgbhccuaaotcdedpabpbropbugeovkefcwbatzjvhg
qnvoorzdpmhglaibjblhypticewcdfidzktwhhhlxgeppddunnyaatqxybmqkhgdudzwdntfthlfajrkmizeaevna
lvggjtxyvprgipghkergemwruqincchqaknmp
nradqlkhywifcfnjbizsuixafqsbmsdgaepwsbablvjsbnbjxapdglxuqzyezxbhhtaedddlueteelk
fhstdxvhlusofasyzaylnyoxteukyqcerhupnjyedqngvwlqflzcolqmsiaynmthwmnzwireqbdvrykbiysvurpzjqlchkd
nwjxwigvgjfdgevhkowsputyxxzzqluayzjqzykrwnlpbvsjgqnamcifwjrete
qbmgtgidnvnkauzjjxluzfdlwhpjxrvdkrernltafadlgfhtylkxtbchdakqebmszkiqvacahwgrtkoicmwaavkmxsbosbzornmzbq
avjzamnfrccrzprrvrepwxjdbddknciyqjbgqnnmcocfwblncbjxqrltsbvcduhh
ocsyomazbehaqhnldzdkuxnggwvgfxqlwmmgpfcmzocykzbdtewlqjxacrhtozfvruvkck
gwaifulzpqbafygevtxhlljdpfowjlgxwfbftfyagcrfyjokuwjhjeijmwxqfbzfoxxkpvbolktpstllkn
lwfyceafxbvvabcucyuyuhlxatoxmgqsdwmsqrkq
auiddoinrygnjxdbpjukctskzjycrreremijqaltgofszjxybqqzzhxxep
jnbjgcnbnapukljadqgrvtcyfdxedvmgqvkwndzywlmrnyocwtxrdlyinnawmbnzgpbwu
serwqztvieslhxnlabcxhusxfliyjgokbdyueesjlsfujegobbnkzvdqlvlerkxkjkyktreudwauvunpxaccmnmhmxrslsoyr
cqtotvgxlqypuqblkgHrcjzegyhkyhswgosinyguql
loayyynvdgsmkpbvbxksudgoaeqix
emjefkhqecmpxgtlwddevsqznkddqvjuklgtcdimxwhvxyxkmaxblwwhpamprawrctssocakdefyosjnzojdkkiefemkfbd
riuzkieuwqylscinnoufllscsfxtixjiullbpi
tfcfndbymwgpxfcdzihovdmfvrdebjuuozudgmfeknmyvgunaazihekkxknlidtkhgdihuietychvjsarbhiiapzqotntocylzkrsksvkdd
bncupeotgjbqnkeajqcwgsuoijvwtlxlilxztputvnvowqbmtpvxbhjuraximorfwwhrwhtcpovuvkkva
fabmrdgaksdwccznozdrefufpfyhhpoarknpekpdngenkolmsncsfmttlcgbatgzrhngxxvadpypdvipnjojfxjzbdwux
fcnczbcwojzagycbmoeaxitehrlbgbkeoikrtqufdnodmglntyinbvizmcmkqdzsallbyznlafr
vervqevpnkrzrhvxqdxjvjbzefcbwoynzpvqqbkf
xclsmvjbbkynsaczbrvffxeniorjvaybwbukhrpvjqwdlthkiytmuxgtlmyapthfwxrxsv
amwxpiotilsgxuunwptldizizwewlefkeapjeqyfsifjrvxcexrhqimsLvrXov
anejxoyeienmjxzlrlrhzjjbrzrtvkqkyvhtgkqwiyucoypsyzy
anlpxgbhccuaaotcdedpabpbropbugeovkefcwbatzjvhg
aomqfhnafkjq
auiddoinry
aujauwtqxbuyfjqzylstqxcijfpeyncw
auqsorjxyolszkiglmwzpmxyauvwsnxtbdypxhkjbnaayoasmwef
avjzamnfrccrz
aximorffwwhrwhtcpovuvkkva
ayrjsmxzbjgdlaizhakczrikooincuqqtgggjmnsmrwnyZpveonmdqynehqqdon
aywytwqzqygblloezkqqr
azlnymoqndwkfyvgznmq
bbhtaedddlueteelk
bdzcwmpffwxqxrampzaok
behnyxecbcnpgjnmrgpwxehxxwjamqs
bemgzmcosrymvzyvgwglbxyyijymaqlszkrwhxvyrdouetwkvxbalccd
bfcoltqossbxjfigilmkebytdmsgbfboddmkcpqwylytnbbpilkrrhnsxep
bflbaeylwjgwkvajyaulpeqnbrtuotipxdzfkpeacdZrdwadalqurayjpi
biihmpchpajagusgjxxmzbkqximzch
bjmctkpqiffhjdcelkuoxaqv
bncupeotgjbqnkeajqcwgsuoijvwtlxlilxztputvnvowqbmtpvxbhjur
bnozgteixmertgqhwbpedmpituxgnbomdgdsxvlveploueynmryxy
bsdctxicafiquntubulqrnbxguwssuenvemlxmfionvfnlxdrut
btblguzwdtrtzlcvepkdcyc
buehoalhojlfldqexiendceblab
```

rand\_str\_input\_sec.txt      sorted

```
panosbel@lenovo: ~/arm_vm
aagdfvlsijdzqcijcddtsxrjeac
aagncpvvytuhbworgbsdnhm
abacvtvbmbozaewgvfrftgwqaaa
agkzuhufilyizgatpmqomesozf
ahdqnsfviyysnuaqxbqoxvogsexldofoacisxsxorctfdrxneqwspujclnroif
ahtpzhqkiloHVnoqqwnmcpmrjknmensepzlxlyvbgbyjhcgvdkflkpaducwpu
ahzkhuowyyyscuokaunvnltjtvvpsywfgttixwejhqcnnuuieqtqvnnekq
aleydjirbkqzatzfbytbefxtmgjnczmnlwumkdblihwzrlhpxevpph
amwxpiotilsgxuunwptldizizwewlefkeapjeqyfsifjrvxcexrhqimsLvrXov
anejxoyeienmjxzlrlrhzjjbrzrtvkqkyvhtgkqwiyucoypsyzy
anlpxgbhccuaaotcdedpabpbropbugeovkefcwbatzjvhg
aomqfhnafkjq
auiddoinry
aujauwtqxbuyfjqzylstqxcijfpeyncw
auqsorjxyolszkiglmwzpmxyauvwsnxtbdypxhkjbnaayoasmwef
avjzamnfrccrz
aximorffwwhrwhtcpovuvkkva
ayrjsmxzbjgdlaizhakczrikooincuqqtgggjmnsmrwnyZpveonmdqynehqqdon
aywytwqzqygblloezkqqr
azlnymoqndwkfyvgznmq
bbhtaedddlueteelk
bdzcwmpffwxqxrampzaok
behnyxecbcnpgjnmrgpwxehxxwjamqs
bemgzmcosrymvzyvgwglbxyyijymaqlszkrwhxvyrdouetwkvxbalccd
bfcoltqossbxjfigilmkebytdmsgbfboddmkcpqwylytnbbpilkrrhnsxep
bflbaeylwjgwkvajyaulpeqnbrtuotipxdzfkpeacdZrdwadalqurayjpi
biihmpchpajagusgjxxmzbkqximzch
bjmctkpqiffhjdcelkuoxaqv
bncupeotgjbqnkeajqcwgsuoijvwtlxlilxztputvnvowqbmtpvxbhjur
bnozgteixmertgqhwbpedmpituxgnbomdgdsxvlveploueynmryxy
bsdctxicafiquntubulqrnbxguwssuenvemlxmfionvfnlxdrut
btblguzwdtrtzlcvepkdcyc
buehoalhojlfldqexiendceblab
```

rand\_str\_input\_sec.txt      len

```
panosbel@lenovo: ~/arm_vm
1
52
22
11
14
9
58
23
60
64
31
15
47
56
44
46
25
64
18
21
64
17
44
59
39
24
64
41
13
50
62
15
52
```

Συμπέρασμα:

Ο σκοπός του να γράψουμε αυτές τις 4 συναρτήσεις σε assembly και να βγάλουμε την βιβλιοθήκη strings.h ήταν ότι η συγκεκριμένη βιβλιοθήκη περιέχει πάρα πολλές συναρτήσεις που για τον κώδικα μας δεν χρειάζονται κάπου. Το να φορτώσουμε όλη την βιβλιοθήκη στον arm θα ήταν σπατάλη μνήμης. Τέλος, με τη χρήση assembly έχουμε έλεγχο στο memory layout και διαχειριζόμαστε το stack απευθείας. Επίσης με την assembly μειώνουμε το overhead και έχουμε μικρότερα latency.