CSCI - 212

ASSIGNMENT 5

NAME :- SHREYAS SRINIVASA
PALOMAR ID :- 01299127

Exercise 10.4

1. hw:-

```
@ helloShreyas.s
@ Hello Shreyas program, in assembly language.

@ Define my Raspberry Pi
    .cpu    cortex-a53
    .fpu    neon-fp-armv8
    .syntax unified                @ modern syntax

@ Useful source code constant
    .equ    STDOUT, 1

@ Constant program data
    .section .rodata
    .align 2
helloMsg:
    .ascii  "Hello Shreyas!\n"
    .equ    helloLength, - helloMsg

@ Program code
    .text
    .align 2
    .global main
    .type   main, %function

main:
        sub     sp, sp, 8       @ space for fp, lr
        str     fp, [sp, 0]     @ save fp
        str     lr, [sp, 4]     @ save lr
        add     fp, sp, 4       @ set our frame
                                @ pointer
        mov     r0, STDOUT      @ file number to write to
        ldr     r1, helloMsgAddr @ pointer to message
        mov     r2, helloLength  @ number of bytes to
                                 @ write
```

```
ll  write    @ write the message

mov r0,0     @ return 0;
ldr fp,[sp,0]  @ restore caller fp
ldr lr,[sp,4]  @ lr
add sp,sp,8    @ and sp
bx  lr         @ return

.align 2

helloMsgAddr:
    .word helloMsg
```

Exercise 10.6

1. Ans
```
@ echo4chars.s
@ Prompts user to enter 4 characters and
@ echoes them

@ Define my Raspberry Pi
    .cpu cortex-a53
    .fpu neon-fp-armv8
    .syntax unified      @ modern syntax

@ Useful source code constants
    .equ STDIN,0
    .equ STDOUT,1
    .equ char1,-8
    .equ char2,-7
    .equ char3,-6
    .equ char4,-5
    .equ nchars,4
    .equ local,8
```

@ Constant program data

```
.section .rodata
.align 2
promptMsg:
    .ascii "Enter four characters: "
    .equ promptLength, . - promptMsg

responseMsg:
    .ascii "You entered: "
    .equ responseLength, . - responseMsg
```

@ Program code

```
.text
.align 2
.global main
.type main, %function

main:
    sub sp, sp, 8          @ space for fp, lr
    str fp, [sp, 0]        @ save fp
    str lr, [sp, 4]        @ and lr
    add fp, sp, 4          @ set our frame pointer
    sub sp, sp, local      @ allocate memory for
                           @ local vars

    mov r0, STDOUT         @ prompt user for input
    ldr r1, promptMsgAddr
    mov r2, promptLength
    bl write

    mov r0, STDIN          @ from keyboard
    add r1, fp, char1      @ address of 1st char
    mov r2, 1              @ one char
    bl read
```


P.T.O

```
mov   r0, STDIN      @ from keyboard
add   r1, fp, char2  @ address of 2nd char
mov   r2, 1          @ one char
bl    read

mov   r0, STDIN      @ from keyboard
add   r1, fp, char3  @ address of 3rd char
mov   r2, 1          @ one char
bl    read

mov   r0, STDIN      @ from keyboard
add   r1, fp, char4  @ address of 4th char
mov   r2, 1          @ one char
bl    read

mov   r0, STDOUT     @ print message for user
ldr   r1, resultsMsgAddr
mov   r2, resultsLength
bl    write

mov   r0, STDOUT     @ echo user's characters
add   r1, fp, char1  @ address of 1st char
mov   r2, nChars     @ all four characters
bl    write

mov   r0, 0          @ return 0;
add   sp, sp, locals @ delete local vars
ldr   fp, [sp, 0]    @ restore caller fp
ldr   lr, [sp, 4]    @ lr
add   sp, sp, 8      @ and sp
bx    lr             @ return

@ address of messages
.align 2
promptMsgAddr:
.word promptMsg
resultsMsgAddr:
.word resultsMsg
```


P.T.O

Newline (Enter key) is treated as one character. If we type three characters and don't do anything else after that, the terminal will wait for the input of the fourth character. Any key pressed next will be interpreted as the fourth character, and the program resumes execution and prints out the three characters along with the newline character that moves the terminal prompt to a new line. If you type less than three characters and stop, the terminal will still be waiting for the next two characters. Any two keys pressed next will be interpreted as characters including the enter key. e.g.. If you type less than three characters and press enter two times, the program will continue execution and display the two entered characters and also display the remaining two newlines by outputting a newline for each one. Extra characters which cause the input to exceed the size of four characters will be read from the input buffer by the terminal after the program finishes execution.

Ex. 12.2

Lab
@ numerals .s
@ Displays the numerals, 0 - 9

@ Define my Raspberry Pi
- cpu cortex - A53
- ram neon - fp - armv8
- syntax unified @ modern syntax

@ useful source code constants
.equ STDOUT, 1
.equ numerical, -20
.equ local, 8

@ Constant program data
.section .rodata
.align 2

@ The program
.text
.align 2
.global main
.type main, %function

main:

sub sp, sp, 16 @ space for saving
@ regs
@ (keeping 8-byte
@ sp align)
str r4, [sp, 4] @ save r4
str fp, [sp, 8] @ fp
str lr, [sp, 12] @ lr
add fp, sp, 12 @ set our frame
@ pointer
sub sp, sp, local @ allocate memory
@ for local vars

mov r4, 0 @ numerical 0

loop:
strb r4, [fp, numerical] @ check next
@ lr
@ in memory
@ for write

```
mov   r0, STDOUT      @ write to screen
add   r1, fp, numeral @ address of numeral
mov   r2, 1           @ one byte
bl    write

add   r4, r4, 1       @ next numeral
cmp   r4, 69          @ all numerals?
blt   loop            @ no, keep going

mov   r0, 0           @ yes, return 0;
add   sp, sp, local   @ deallocate local vars
ldmfd r4, [sp, 43]    @ restore r4
ldr   fp, [sp, 8]     @ fp
ldr   lr, [sp, 12]    @ lr
add   sp, sp, 16      @ sp
bx    lr              @ return
```

2. Ans: @ Letters.s

@ Displays the upper case alphabet, A - Z

```
@ Defines my Raspberry Pi
.cpu cortex-a53
.fpu neon-fp-armv8
.syntax unified @ modern syntax

@ Useful source code constants
.equ STDOUT, 1
.equ alpha, -20
.equ local, 8

@ Control program data
.section .rodata
.align 2
```

@ The program

```
.text
.align 2
.global main
.type main, %function

main:
        sub     sp, sp, 16      @ space for saving
                                @ regs
                                @ (keeping 8-byte sp
                                @ align)
        str     r4, [sp, 4]     @ save r4
        str     fp, [sp, 8]     @ fp
        str     lr, [sp, 12]    @ lr
        add     fp, sp, 12      @ set over frame
                                @ pointer
        sub     sp, sp, local   @ allocate memory
                                @ for local vars
        mov     r4, 'A          @ beginning of alphabet

loop:
        strb    r4, [fp, alpha] @ char must be
                                @ in memory for
                                @ write
        mov     r0, STDOUT      @ write to screen
        add     r1, fp, alpha   @ address of alpha
        mov     r2, 1           @ one byte
        bl      write

        add     r4, r4, 1       @ next alpha
        cmp     r4, 'Z          @ whole alphabet?
        ble     loop            @ no, loop again
```


P.T.O

```
mov   r0, 0        @ yes, return 0;
add   sp, sp, local  @ deallocate local vars
ldr   r4, [sp, 4]    @ restore r4
ldr   lr, [sp, 8]    @ lr
ldr   lr, [sp, 12]   @ lr
add   sp, sp, 16     @ sp
bx    lr             @ return
```

Ex. 12.4

1. Ans @ range3.s
@ Checks if user entered a numeral.

```
@ Define my Raspberry Pi
.cpu cortex-a53
.fpu neon-fp-armv8
.syntax unified          @ modern syntax

@ Useful source code constants
.equ STDOUT, 1
.equ STDIN, 0
.equ NUL, 0
.equ response, -20
.equ local, 8

@ Constant program data
.section .rodata
.align 2

prompt:
.asciz "Enter a single character:"
.align 2
```

numeral:

.asciz "you entered a numeral. \n"
.align 2

other:

.asciz "you entered a non-numeric character \n"

@ The program
.text
.align 2
.global main
.type main, @function

main:

sub sp, sp, 16 @ space for saving reg
@ (keeping 8 byte sp
@ align)
str r4, [sp, 4] @ save r4
str fp, [sp, 8] @ fp
str lr, [sp, 12] @ lr
add fp, sp, 12 @ set our frame pointer
sub sp, sp, local @ local variable
ldr r4, promptaddr @ prompt user

promptloop:
ldrb r3, [r4] @ get a char
cmp r3, NUL @ end of string?
bneq getResponse @ yes, get response

mov r0, STDOUT @ we, write to screen
mov r1, r4 @ address of current char
mov r2, 1 @ write 1 byte
bl write

P.T.O

```
add   r4, r4, 1        @ increment pointer mem
b     promptLoop       @ back to top

getResponse:
      mov   r0, STDIN        @ from keyboard
      add   r1, fp, response @ address of response
      mov   r2, 2            @ one char plus enter
                             @ key
      bl    read

      ldrb  r3, [fp, response] @ load response
      cmp   r3, '9'          @ check high end
      bhi   notNumeral       @ > '9', otherwise
      cmp   r3, '0'          @ check low end
      blo   notNumeral       @ < '0', other char

      ldr   r4, numericalAddr @ "you entered a
                              numerical"

numLoop:
      ldrb  r3, [r4]         @ get a char
      cmp   r3, NUL          @ end of string?
      beq   endThen          @ yes, end of then block
      mov   r0, STDOUT       @ no, write to screen
      mov   r1, r4           @ address of current char
      mov   r2, 1            @ write 1 byte
      bl    write
      add   r4, r4, 1        @ increment pointer mem
      b     numLoop          @ back to top

endThen:
      b     endElse          @ branch over else block
notNumeral:
      ldr   r4, otherAddr    @ "you entered some
                             other character."
```

```
notNumCheck:
        ldrb    r3, [r4]        @ get a char
        cmp     r3, NUL         @ end of string?
        beq     endElse         @ yes, end of if-else
        mov     r0, STDOUT      @ no, write to screen
        mov     r1, r4          @ address of current char
        mov     r2, 1           @ write 1 byte
        bl      write
        add     r4, r4, 1       @ increment pointer
        b       notNumCheck     @ back to top

endElse:
        mov     r0, 0           @ return 0;
        add     sp, sp, local   @ deallocate local vars
        ldr     r4, [sp, 4]     @ restore r4
        ldr     fp, [sp, 8]     @        fp
        ldr     lr, [sp, 12]    @        lr
        add     sp, sp, 16      @        sp
        bx      lr              @ return

        .align  2
promptAddr:
        .word   prompt
numeralAddr:
        .word   numeral
otherAddr:
        .word   other
```