Date: 11/03/2021

CSCI-212

ASSIGNMENT-6

NAME:- SHREYAS SRINIVASA PALOMAR
ID :- 012551187

Section 13.3

1. Ans. Will come back to this later.

2. Ans.
@ writestr.s
@ Writes a 0-style text string to the standard
@ output (screen).
@ Calling sequence:
@ r0 <- address of string to be written
@ bl writestr
@ returns number of characters written

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

@ Useful source code constants
.equ STDOUT, 1
.equ NUL, 0

@ The code
.text
.align 2
.global writestr
.type writestr, %function
writestr:
sub sp, sp, 16 @space for saving
@ regs

P.T.O

Date: 11/09/2021

```
str  r4, [sp, 0]   @ save r4
str  r5, [sp, 4]   @      r5
str  fp, [sp, 8]   @      fp
str  lr, [sp, 12]  @      lr
add  fp, sp, 12    @ set new frame pointer

mov  r4, r0        @ r4 = string pointer
mov  r5, 0         @ r5 = count

whileloop:
ldrb r3, [r4]      @ get a char
cmp  r3, NUL       @ end of string?
beq  alldone       @ yes, all done

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 anyway
add  r5, r5, 1     @ count++
b    whileloop     @ back to top

alldone:
mov  r0, r5        @ return count
ldr  r4, [sp, 0]   @ restore r4
ldr  r5, [sp, 4]   @         r5
ldr  fp, [sp, 8]   @         fp
ldr  lr, [sp, 12]  @         lr
add  sp, sp, 16    @ restore sp
bx   lr            @ return

@ helloWorld3.s
@ Hello world program to test writestr function
```


P.T.O

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

@ Constant program data
- section .rodata
- align 2
thestring:
- asciz "Hello World - In"

@ The program
- 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

ldr r0, thestringaddr
bl writeStr

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

thestringaddr:
- word thestring

3. (a) readstr.simple.s

@ Reads a line (through the '\n') from standard
@ input. Deletes the '\n' and creates a C-style
@ text string.
@ Calling sequence:
@ r0 <- address of place to store string
@ bl readstr
@ returns number of characters read, excluding
@ NUL.

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

@ Useful source code constants
.equ STDIN,0
.equ NUL,0
.equ LF,10 @ '\n' under Linux

@ The code
.text
.align 2
.global readstr
.type readstr, %function
readstr:
sub sp,sp,16 @ space for saving reg
str r4,[sp,4] @ some r4
str r5,[sp,8] @ r5
str fp,[sp,12] @ fp
str lr,[sp,16] @ lr
add fp,sp,12 @ set our frame
@ pointer

```
        mov     r4, r0          @ r4 = string pointer
        mov     r5, 0           @ r5 = count

        mov     r0, STDIN       @ read from keyboard
        mov     r1, r4          @ address of current
                                @ storage
        mov     r2, 1           @ read 1 byte
        bl      read
whileloop:
        ldrb    r3, [r4]        @ get just read char
        cmp     r3, LF          @ end of input?
        beq     endofstring     @ yes, input done
        add     r4, r4, 1       @ no, increment pointer
                                @ too
        add     r5, r5, 1       @ count ++
        mov     r0, STDIN       @ read from keyboard
        mov     r1, r4          @ address of current
                                @ storage
        mov     r2, #1          @ read 1 byte
        bl      read
        b       whileloop       @ and check for eof
endofstring:
        mov     r0, NUL         @ string terminator
        strb    r0, [r4]        @ write over 'n'

        mov     r0, r5          @ return count;
        ldr     r4, [sp, 4]     @ restore r4
        ldr     r5, [sp, 8]     @      r5
        ldr     fp, [sp, 12]    @      fp
        ldr     lr, [sp, 16]    @      lr
        add     sp, sp, 16      @ space for saving regs
        bx      lr              @ return
```

Date: 12/03/2021

@ echostring I.s
@ Prompts user to enter a string, then echoes
@ it.

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

@ Constant for assembler
.equ nBytes, 50 @ amount of memory
@ for string

@ Constant program data
.section .rodata
.align 2
prompt:
.asciz "Enter some text: "

@ 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 x4, [sp, 4] @ save x4
str fp, [sp, 8] @ fp
str lr, [sp, 12] @ lr
add fp, sp, 12 @ set our frame
@ pointer

mov x0, nBytes @ get memory from
@ heap

(6)

```
bl      malloc
mov     r4, r0      @ pointer to new mem-
                    @ -ory

ldr r0, promptaddr  @ prompt user
bl writestr

mov r0, r4          @ get user input
bl readstr

mov r0, r4          @ echo user input
bl writestr

mov r0, r4          @ free heap memory
bl free

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

promptaddr:
        .word   prompt
```

The space used by the stored characters is dynamically allocated by using a library call, malloc. It just returns a pointer to the first memory location allocated, but depending on operating system and CPU architecture, more memory may have been technically assigned by the OS to the process, and accessing more memory than what is allocated by malloc would

Date: 14/05/2024

be considered legal from the OS perspective. Continued use of malloc will eventually cause a buffer overflow by overwriting on the memory which is supposed to be used by the other parts of the program. In short, Nothing happens on the Raspberry Pi, and the program returns the expected output with the excess characters.

1 Any In my opinion, the code is uncluttered and very efficient, making lesser use of space in the primary memory as the optimization level is increased. The number of instructions is also greatly reduced and operations are performed with the minimal number of instructions.

The optimised code directly makes use of the values in the registers which are accessible by both the calling and called function. It retrieves the first four arguments from the registers directly and calculates the sum, and then retrieves the remaining five arguments from the call stack. The original version of the assembly code inefficiently moves the values existing in the registers into the call stack to save them, although they are unused and re-redundantly loads all the arguments into the registers and then into the call stack to pass to the sumNine function. The sumNine function similarly wastes a lot of instructions loading values to and from the call stack instead of following the highly simplified approach of the optimised assembly code.

P.T.O

Date: 11/04/2021

The difference between the optimised code generated and the code in Listing 13.2.6 is that only five arguments are loaded onto the stack once and retrieved once in the entire program. Rest of the arguments are already present in the stack frame and are easily retrieved and serviced up. The stack frame in the optimised assembly code and listing are greatly reduced and the addition operation further simplified due to reduced load instructions.

The optimised assembly code is a condensed solution and easier to read and understand.

4 hrs

@ readln.s
@ Reads a line (through the '\n') from
@ standard input. Stops at size limit. Extra
@ characters and '\n' are ignored. Stores
@ NUL-terminated string.
@ Calling sequence:
@ r0 <- address of place to store string
@ r1 <- string size limit
@ bl readln
@ returns number of characters read, excluding
@ NUL.

@ Define my Raspberry Pi
- Pico Cortex-A53
- Pico neon-fp-armv8
- Syntax unified @ .macro,
@ .syntax

@ Useful source code constants

```
.equ STDIN, 0
.equ NUL, 0
.equ LF, 10   @ 'in' under Linux
```

@ The code

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

readLn:

```
sub sp, sp, 24   @ space for saving regs
                 @ (keeping 8-byte sp
                 @ align)
str r4, [sp, 4]   @ save r4
str r5, [sp, 8]   @      r5
str r6, [sp, 12]  @      r6
str fp, [sp, 16]  @      fp
str lr, [sp, 20]  @      lr
add fp, sp, 20    @ set our frame
                  @ pointer

mov r4, r0   @ r4 = string pointer
mov r5, 0    @ r5 = count
mov r6, r1   @ r6 = max chars
sub r6, 1    @ for NUL

mov r0, STDIN   @ read from
                @ keyboard
mov r1, r4      @ address of current
                @ storage
mov r2, 1       @ read 1 byte
bl read
```


P.T.O

```
whileLoop:
        ldrb    r3, [r4]        @ get just read
                                @ char
        cmp     r3, LF          @ end of input?
        beq     endOfString     @ yes, input done
        cmp     r5, r6          @ max chars?
        beq     ignore          @ yes, ignore rest
        add     r4, r4, 1       @ no, increment
                                @ pointer reg
        add     r5, r5, 1       @ count++
ignore:@
        mov     r0, STDIN       @ read from keyboa-
                                @- ard
        mov     r1, r4          @ address of current
                                @ storage
        mov     r2, 1           @ read 1 byte
        bl      read
        b       whileLoop       @ and check for end
endOfString:
        mov     r0, NUL         @ string terminator
        strb    r0, [r4]        @ write over '\n'

        mov     r0, r5          @ return count;
        ldr     r4, [sp, 4]     @ restore r4
        ldr     r5, [sp, 8]     @       r5
        ldr     r6, [sp, 12]    @       r6
        ldr     fp, [sp, 16]    @       fp
        ldr     lr, [sp, 20]    @       lr
        add     sp, sp, 24      @ sp
        bx      lr              @ return

@ echostring2.s
@ prompts user to enter a string, then echoes
@ it.
```

Date: 11/04/2024

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

@ Constant for assembler
- equ nBytes, 5 @ amount of memory for @ string

@ Constant program data
.section .rodata
.align 2

prompt:
- asciz "Enter some text: "

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

main:
sub sp, sp, 16 @ space for saving @ regs (keeping @ 8-byte sp alignment)
str r4, [sp, 4] @ save r4
str fp, [sp, 8] @ fp
str lr, [sp, 12] @ lr
add fp, sp, 12 @ set our frame pointer

mov r0, nBytes @ get memory from @ heap
bl malloc
mov r4, r0 @ pointer to new memory

ldr r0, promptAddr @ prompt user
bl writeStr

Date: 11/04/2021

```
mov   r0, r4    @ get user input
mov   r1, nBytes @ limit input size
bl    readStr

mov   r0, r4    @ echo user input
bl    writeStr

mov   r0, r4    @ free heap memory
bl    free

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

promptLabel:
    .word prompt
```

Q. Ans:-

@ newline.s
@ Writes a newline character to the standard
@ output (screen).
@ Calling sequence:
@ bl newline

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

@ Useful source code contents
.equ STDOUT, 1

D, T = 0

@ Constant program data
.section .rodata
.align 2
theChar:
.asciz "\n"

@ The code
.text
.align 2
.global newline
.type newline, %function
newline:
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

mov r0, STDOUT @ write to screen
ldr r1, theCharAddr @address of new-
@-line char
mov r2, 1 @write 1 byte
bl write

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

theCharAddr:
.word theChar