Date: 11/28/2021

CSC - 212

ASSIGNMENT - 8

Ex 15.2

Ans @ reverse/array.s

@ Gets 10 integers from user then prints in reverse order.

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

@ Constant program data
- section .rodata
- align 2

prompt:
- asciz "Enter an integer: \n"

display:
- asciz "In reverse order: \n".

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

main:
sub sp, sp, 16 @ space for saving
@ - my regs
str x19, [sp, 0] @ save x19
str x5, [sp, 4] @ x5
str fp, [sp, 8] @ fp
str lr, [sp, 12] @ lr

P.T.O

```
add  fp, sp, #2     @ set new frame
                    @ pointer
sub  sp, sp, locals @ for the array

add  r4, fp, intArray  @ address of
                       @ array beg
                       @ - knowing
mov  r5, 0          @ index = 0;
fillLoop:
     cmp  r5, nElements  @ all filled?
     bge  allFull        @ yes
     @ldr  r0, =promptAddr  @ no, prompt
                            @ user
     bl   writeStr
     bl   getDecInt      @ get integer
     lsl  r1, r5, 2      @ offset is 4* in-
                         @ - dex
     str  r0, [r4, r1]   @ in index -th
                         @ element
     add  r5, r5, 1      @ index ++;
     b    fillLoop
allFull:
     ldr  r0, displayAddr @ new messag
     bl   writeStr

     add  r4, fp, intArray  @ address
                            @ of array beg-
                            @ - in
     mov  r5, 9          @ index = 9;
printLoop:
     lsl  r1, r5, 2      @ no, offset
                         @ is 4 * index
     ldr  r0, [r4, r1]   @ at index-th
                         @ element
```

bl put_Decimal @ print integer
bl newline
subs r5, r5, 1 @ index--;
bge printloop

allDone:
mov r0, 0 @ return 0;
add sp, sp, locals @ deallocate locals
@ mov
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

promptAddr:
.word prompt

displayAddr:
.word display

Ex 15.5

@ structBoss3.s
@ Allocates two structs and gets values for user for
@ each struct, then displays the values.

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

@ Constants for assembler
- include "thegStruct.s" @ the
@ eg. struct defs.

.equ y, -36 @ y struct
.equ x, -24 @ x struct
.equ locals, 24 @ space for the structs

@ Constant program data
.section .rodata
.align 2
display X:
.asciz "x fields: \n"
display Y:
.asciz "y fields: \n"

@ 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, locals @ for the structs

@ fill the x struct
add r0, fp, x @ address of x struct
bl getstruct

@ fill the y struct
add r0, fp, y @ address of y struct
bl getstruct

Date: 11/28/2021

```
@ display x struct
        ldr r0, displayXaddr
        bl  writeStr
        add r0, fp, 2       @ address of x struct
        bl  putstruct
        bl  newline

@ display y struct
        ldr r0, displayYaddr
        bl  writeStr
        add r0, fp, 7       @ address of y struct
        bl  putstruct
        bl  newline

        mov r0, 0       @ return 0;
        add sp, sp, locals  @ deallocate local
                            @ var
        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
@ addresses of messages
displayXaddr:
        .word displayX
displayYaddr:
        .word displayY

@ getstruct_s
@ gets values for a the 2dg struct from key-
@ - board
```


P.T.O

Date: 11/28/2021

```
@ Calling sequence:
@       r0 <- address of the struct
@       bl getStruct
@ Returns 0

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

@ Constants for assembler
    .include "the3asgStruct.s"  @ the3asg
                                @ struct defs.

@ Constant program data
    .section .rodata
    .align 2
charPrompt:
    .asciz  "Enter a character: "
intPrompt:
    .asciz  "Enter an integer: "

@ The program
    .text
    .align 2
    .global getStruct
    .type getStruct, %function
getStruct:
    sub sp, sp, 16  @ space for saving
                    @ regs
                    @ (keeping 8-byte
                    @ sp align)
    str x4, [sp, 4] @ save x4
```

Date: 11/28/2021

```
str fp, [sp, 8]      @ fp
str lr, [sp, 12]     @ lr
add fp, sp, 12       @ set new frame
                     @ pointer

mov r4, r0   @ pointer to the struct

ldr r0, =charPromptAddr
bl  writeStr     @ ask for a char
bl  getChar      @ get it
strb r0, [r4, #aChar]  @ aStruct->
                       @ aChar = first&
                       @ - char;

ldr r0, =intPromptAddr
bl  writeStr     @ ask for a char
bl  getDecInt    @ get it
str r0, [r4, #anInt]  @ aStruct->anInt=
                      @ aNumber;

ldr r0, =charPromptAddr
bl  writeStr     @ ask for a char
bl  getChar      @ get it
strb r0, [r4, #anotherChar]  @ aStruct
                             @ -> anotherChar=
                             @ secondChar;

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
```

.align 2

Date: 11/28/2021

@ addresses of messages
charPromptAddr:
    .word charPrompt
intPromptAddr:
    .word intPrompt

@ putstruct :
@ Displays values for a theTagstruct on screen
@ Calling sequence:
@   r0 <- address of the struct
@   bl putstruct
@ Returns 0

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

@ Constants for assembler
    .include "theTagstruct.s" @ the Tag
                              @ struct defs

@ Constant program data
    .section .rodata
    .align 2
dispAChar:
    .ascii "aChar = "
dispAnInt
    .ascii "anInt = "
dispOtherChar:
    .ascii "anotherChar = "

@ The program

Date: 11/28/2021

```
.text
.align 2
.global putStruct
.type putStruct, %function
putStruct:
    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

    mov  r4, r0         @ pointer to the struct

    ldr  r0, displayCharAddr  @ display achar
    bl   writeStr
    ldrb r0, [r4, achar]
    bl   putChar
    bl   newline
    ldr  r0, displayAnInt Addr  @ display anInt
    bl   writeStr
    ldr  r0, [r4, anInt]
    bl   putDecInt
    bl   newline
    ldr  r0, displayAnotherCharAddr  @ display
                                     @ anotherChar
    bl   writeStr
    ldrb r0, [r4, anotherChar]
    bl   putChar
    bl   newline

    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

.align 2
@ addresses of messages
displayCharAddr:
        .word displayChar
displayUnsignedIntAddr:
        .word displayUnsignedInt
displayOtherCharAddr:
        .word displayOtherChar

@ getChar.s
@ gets one char from keyboard.
@ Calling sequence:
@       bl getChar
@ returns char in low byte of r0

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

@ Constants for assembler
        .equ maxChars, 2  @ max input chars
        .equ inputChar, -12  @ for input chars
        .equ locals, 8  @ space for locals

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

getChar:

```
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 own frame pointer
                     @ - ter
sub   sp, sp, locals @ for the string.

add   r0, fp, inputChar @ place to
                        @ store input
mov   r1, maxChars      @ limit input
                        @ length
bl    readIn

ldrb  r0, [fp, inputChar] @ return
                          @ inputChar

add   sp, sp, locals @ deallocate local
                     @ vars
ldr   fp, [sp,0]     @ restore caller fp
ldr   lr, [sp, 4]    @          lr
add   sp, sp, 8      @ and sp
bx    lr             @ return
```

Exercise 19.7

Ans
```
@ int fraction.s
@ gets values from user for a fraction, adds
@ to the fraction, and then displays the result.

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

Date: 11/28/2019

@ Constants for assembler
.equ x, -12 @ x fraction object
.equ locals, 8 @ space for fraction

@ 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
sub sp, sp, locals @ for the structs

@ construct the fraction
add r0, fp, x @ address of x struct
bl fractionConstr

@ get user input
add r0, fp, x @ get user values
bl fractionGet

@ add 1
add r0, fp, x @ add 1 to it
mov r1, 1
bl fractionAddInt

@ display result
add r0, fp, x @ get user values
bl fractionDisplay

Date: 11/28/2021

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

@fractionObject.s
@field name definitions ; requires 8 bytes

@ fraction object definition
        .equ num, 0     @ numerator
        .equ den, 4     @ denominator

@fractionConstr.s
@ Construct a fraction object
@ Calling sequence:
@       r0 <- address of fraction variable
@       bl fractionConstr
@ Returns 0

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

@ Constants for assembler
        .include "fractionObject.s"  @ fraction
                                     @ object defs.

@ The code
        .text
        .align 2
        .global fractionConstr
```

```
.type fractionConstr, %function
fractionConstr:
    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   r1, 1          @ reasonable numerator
    str   r1, [r0, num]

    mov   r1, 2          @ reasonable denominator
    str   r1, [r0, den]

    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

@ fractionGet.s
@ Gets values for a fraction from keyboard
@ Calling sequence:
@       r0 <- address of the struct
@       bl getStruct
@ Returns 0

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

@ Constants for assembler
    .include "fractionObject.s" @ fraction
                                @ object defs.
```

```
@ Constant program data
        .section .rodata
        .align 2
numBrompt:
        .asciz  "Enter numerator: "
denBrompt:
        .asciz  "Enter denominator: "

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

        mov  r4, r0         @ pointer to the object

        ldr  r0 numBromptAddr
        bl   writeStr       @ ask for numerator
        bl   getDecInt      @ get it
        str  r0, [r4, num]  @ store at the ->
                            @ num

        ldr  r0, denBromptAddr
        bl   writeStr       @ ask for denominator
        bl   getDecInt      @ get it
        str  r0, [r4, den]  @ store at the ->
                            @ den
```

```
    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

    .align 2
@ addresses of messages
numPromptAddr:
    .word numPrompt
denPromptAddr:
    .word denPrompt

@ fractionAddInt.s
@ adds an integer to a fraction
@ assumes (int x den) + num fits into 32 bits.
@ calling sequence:
@       r0 <- address of the object
@       r1 <- integer to add
@       bl fractionAddInt
@ Returns 0

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

@ Constants for assembler
    .include "fractionObject.s" @ fraction
                                @ object defs.

@ The code
    .text
    .align 2
```

```
.global fractionAddInt
.type fractionAddInt, %function
fractionAddInt:
        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
        mov   r4, r0        @ this pointer
        ldr   r0, [r4, den] @ get denominator
        mul   r2, r3, r0    @ integer * denominator
        ldr   r0, [r4, num] @ get numerator
        add   r2, r2, r0    @ numerator + (int * den)
        str   r2, [r4, num] @ save new numerator
        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

@ fractionDisplay.s
@@ Displays a fraction struct on screen
@ Calling sequence:
@       r0 <- address of the struct
@       bl  fractionDisplay
@ Returns 0

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

@ Constants for assembler
        .include "fractionObject.s"  @ fraction object def

(17)
```

@ The code

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

fractiondisplay:

```
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 own
               @ frame pointer
mov r4, r0 @ this pointer
ldr r0, [r4, num] @ display
                  @ numerator
bl putDecInt
mov r0, '/' @ slash for
            @ fraction
bl putChar
ldr r0, [r4, den] @ dis-
                  @ play
                  @ denomi-
```

(19)

```
bl  putDecInt
bl  newline

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
```

```
@ putChar.s
@ Writes a character to the standard
@ output (screen).
@ Calling sequence:
@       r0 <- the character
@       bl  putChar

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

@ Constants for assembler
        .equ STDOUT, 1
        .equ theChar, -5 @ for
                         @ storing
        .equ locals, 8 @ space for
```

@ local var

@ the code

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

putchar:
    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, locals
    strb r0, [fp, -thechar]
          @ write needs addr
          @ - reg
    mov r0, STDOUT @ write
                   @ to screen
    add r1, fp, -thechar
          @ address of the
          @ - char
    mov r2, 1 @ write 1 byte
    bl write
          @ r0)
```

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