Sara Kazemi
CSIS 212
10/22/2019

Assignment #5

10.4, 10.6, 12.2, and 12.4

10.4 Programming Exercise

1. Modify the assembly language program in Listing 10.3.1 so that it prints "Hello, *yourName*!" on the screen. Remember to change the documentation such that it accurately describes your program.

```
@ helloWorld2.s
@ Hello World program, in assembly language.
@ 2017-09-29: Bob Plantz

@ 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:
        .asciz   "Hello, World!\n"
        .equ    helloLngth,.-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]     @   and 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, helloLngth  @ number of bytes to write
        bl      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
```

```
IchiGo:A5 Sarie$ cat helloSara.s
@ helloSara.s
@ Writes Hello, Sara! to screen
@ Sara Kazemi

@ 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:
        .asciz   "Hello, Sara!\n"
        .equ    helloLngth,.-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]     @   and 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, helloLngth  @ number of bytes to write
        bl      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
```

```
[pi@kazemi:~/Assembly-Programming/Assignments/A5 $ as --gstabs -o helloSara.o helloSara.s
[pi@kazemi:~/Assembly-Programming/Assignments/A5 $ gcc -o helloSara helloSara.o
[pi@kazemi:~/Assembly-Programming/Assignments/A5 $ ./helloSara
Hello, Sara!
```

10.6 Programming Exercise

1. Write an assembly language program that allows the user to enter four characters and then echoes them. What happens if the user enters three characters? What happens if the user enters fewer than three characters?

```
Enter four characters: abcd
You entered: abcdpi@kazemi:~/Assembly-Programming/Assignments/A5 $
```

If the user enters three characters, the program echoes back those three characters plus the RETURN character.

```
Enter four characters: abc
You entered: abc
pi@kazemi:~/Assembly-Programming/Assignments/A5 $
```

If the user enters fewer than three characters, the program continues to wait for input, since it is expecting four characters before moving on to the writing to standard output. You can see below that I entered two characters, a and b and hit return. This means that the characters a, b, and RETURN are currently in memory. But it will continue to wait until a fourth character is input and stored to memory.

```
pi@kazemi:~/Assembly-Programming/Assignments/A5 $ ./echo4chars
Enter four characters: ab
```

After entering an additional character and RETURN, the program writes ab<RETURN>d to the terminal screen.

```
pi@kazemi:~/Assembly-Programming/Assignments/A5 $ ./echo4chars
Enter four characters: ab
d
You entered: ab
dpi@kazemi:~/Assembly-Programming/Assignments/A5 $
pi@kazemi:~/Assembly-Programming/Assignments/A5 $
```

```
@ echo4chars.s
@ Prompts user to enter a 4 characters and echoes them.
@ Sara Kazemi

@ 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    nChars,4 @ number of characters to read and echo
        .equ    char1,-8
        .equ    char2,-7
        .equ    char3,-6
        .equ    char4,-5
        .equ    local,8

@ Constant program data
        .section  .rodata
        .align  2
promptMsg:
        .asciz   "Enter four characters: "
        .equ    promptLngth,.-promptMsg
responseMsg:
        .asciz   "You entered: "
        .equ    responseLngth,.-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 var

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

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

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

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

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

        mov     r0, STDOUT      @ nice message for user
        ldr     r1, responseMsgAddr
        mov     r2, responseLngth
        bl      write

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

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

@ Addresses of messages
        .align  2
promptMsgAddr:
        .word   promptMsg
responseMsgAddr:
        .word   responseMsg
```

12.2 Programming Exercises

1. Assume that you do not know how many numerals there are, only that the first one is '0' and the last one is '9'. Write a program in assembly language that displays all the numerals, $0, \ldots, 9$ on the screen, one character at a time. Do not allocate a separate character for each numeral.

```
pi@kazemi:~/Assembly-Programming/Assignments/A5 $ more numerals.s
@ numerals.s
@ Displays all numerals 0-9
@ Sara Kazemi

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

@ Constants
        .equ    STDOUT,1
        .equ    numeral,-20     @ for offset in register
        .equ    local,8         @ for memory allocation

@ Constant program data
        .section .rodata
        .align 2

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

@ main function
main:
        sub   sp, sp, 16      @ reserve space for regs
                              @ using 8-byte sp align
        str   r4, [sp, 4]     @ store r4
        str   fp, [sp, 8]     @ store fp
        str   lr, [sp, 12]    @ store lr
        add   fp, sp, 12      @ set our frame pointer
        sub   sp, sp, local   @ allocate mem for local var
        mov   r4, '0          @ move numeral 0 to r4

@ loop
loop:
        strb  r4, [fp, numeral] @ store register byte in r4
                                @ using numeral offset
        mov   r0, STDOUT        @ write to stdout
        add   r1, fp, numeral   @ addr of numeral in r1
        mov   r2, 1             @ 1 byte
        bl    write

        add   r4, r4, 1         @ next numeral
        cmp   r4, '9            @ have gone past max numeral?
        ble   loop              @ false, loop again
        mov   r0, 0             @ true, return 0
        add   sp, sp, local     @ deallocate local var mem
        ldr   r4, [sp, 4]       @ restore r4
        ldr   fp, [sp, 8]       @ restore fp
        ldr   lr, [sp, 12]      @ restore lr
        add   sp, sp, 16        @ restore sp
        bx    lr                @ return
```

```
pi@kazemi:~/Assembly-Programming/Assignments/A5 $ as --gstabs -o numerals.o numerals.s
pi@kazemi:~/Assembly-Programming/Assignments/A5 $ gcc -o numerals numerals.o
pi@kazemi:~/Assembly-Programming/Assignments/A5 $ ./numerals
0123456789pi@kazemi:~/Assembly-Programming/Assignments/A5 $
```

2. Assume that you do not know how many alphabetic characters there are, only that the first one is ‘A’ and the last one is ‘Z’. Write a program in assembly language that displays all the letters, A,..., Z on the screen, one character at a time. Do not allocate a separate character for each numeral.

```
[pi@kazemi:~/Assembly-Programming/Assignments/A5 $ more alphas.s
@ alphas.s
@ Displays all numerals 0-9
@ Sara Kazemi

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

@ Constants
        .equ    STDOUT,1
        .equ    alpha,-20     @ for offset in register
        .equ    local,8         @ for memory allocation

@ Constant program data
        .section .rodata
        .align 2

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

@ main function
main:
        sub   sp, sp, 16      @ reserve space for regs
                              @ using 8-byte sp align
        str   r4, [sp, 4]     @ store r4
        str   fp, [sp, 8]     @ store fp
        str   lr, [sp, 12]    @ store lr
        add   fp, sp, 12      @ set our frame pointer
        sub   sp, sp, local   @ allocate mem for local var
        mov   r4, 'A          @ move numeral 0 to r4

@ loop
loop:
        strb  r4, [fp, alpha]   @ store register byte in r4
                                @ using numeral offset
        mov   r0, STDOUT        @ write to stdout
        add   r1, fp, alpha   @ addr of numeral in r1
        mov   r2, 1             @ 1 byte
        bl    write

        add   r4, r4, 1         @ next numeral
        cmp   r4, 'Z            @ have gone past max numeral?
        ble   loop              @ false, loop again
        mov   r0, 0             @ true, return 0
        add   sp, sp, local     @ deallocate local var mem
        ldr   r4, [sp, 4]       @ restore r4
        ldr   fp, [sp, 8]       @ restore fp
        ldr   lr, [sp, 12]      @ restore lr
        add   sp, sp, 16        @ restore sp
        bx    lr                @ return
```

```
[pi@kazemi:~/Assembly-Programming/Assignments/A5 $ as --gstabs -o alphas.o alphas.s
[pi@kazemi:~/Assembly-Programming/Assignments/A5 $ gcc -o alphas alphas.o
[pi@kazemi:~/Assembly-Programming/Assignments/A5 $ ./alphas
ABCDEFGHIJKLMNOPQRSTUVWXYZpi@kazemi:~/Assembly-Programming/Assignments/A5 $
```