ASSEMBLER PROGRAMMING IN RASPBIAN

Objectives

- Become familiar with Assembler instructions to manipulate registers at low level programming
- Become familiar with development tools for building executable images from Assembler coding

Pre-lab

The following activities provide testing ASM code for you to assemble, compile and run on the Raspberry Pi. Test each of the sample programs provided on your assigned device. All ASM codes are available on GitHub on the following link: https://github.com.

Activity 1. Program that returns an Exit code

```
@ first.s
@   from thinkingeek.com
@   http://thinkingeek.com/2013/01/09/arm-assembler-raspberry-pi-chapter-1/
@   Defines a main function that returns 2 as an exit code.

.global main

main:   mov r0, #2        @ load immediate value 2 into Register r0
        bx lr             @ return 2 to Operating System
```

Activity 2. Hello, World!

```
@ hello.s
@   D. Thiebaut
@   Just your regular Hello World program!
@
@ ---------------------------------------
@   Data Section
@ ---------------------------------------

.data
string: .asciz "\nHello World!\n"
@ ---------------------------------------
@   Code Section
@ ---------------------------------------

.text
.global main
.extern printf

main:
        push {ip, lr}
        ldr  r0, =string
        bl   printf
        pop  {ip, pc}
```

Activity 3. Arithmetic with integer variables

```
@ sum1.s
@   D. Thiebaut
@   add 2 variables together
@
@ ---------------------------------------
@   Data Section
@ ---------------------------------------

.data
.balign 4
string: .asciz "\na + b = %d\n"
a:      .word 33
b:      .word 44
c:      .word 0       @ will contain a+b

@ ---------------------------------------
@   Code Section
@ ---------------------------------------

.text
.global main
.extern printf

main:
        push   {ip, lr}         @ push return address + dummy register for alignment

        ldr    r1, =a           @ get address of a into r1
        ldr    r1, [r1]         @ get a into r1
        ldr    r2, =b           @ get address of b into r2
        ldr    r2, [r2]         @ get b into r2
        add    r1, r1, r2       @ add r1 to r2 and store into r1
        ldr    r2, =c           @ get address of c into r2
        str    r1, [r2]         @ store r1 into c

        ldr    r0, =string      @ get address of string into r0
        ldr    r1, [r2]         @ pass c=a+b into r1
        bl     printf           @ print string and r1 as param

        pop    {ip, pc}         @ pop return address into pc
```

Do the following:

- Indicate the section of code where the variables are created.
- Explain the purpose of the directives: ".word", ".asciz".
- Indicate the section where the instructions of the program are written.
- What is stored in registers r0 and r1 before the call to printf.
- Show a print-screen with the final result.

Activity 4. Arithmetic with integer variables, version 2

```
@   sum2.s
@   D. Thiebaut
@   add 2 variables together and print the result.
@
@ ---------------------------------------
@       Data Section
@ ---------------------------------------

        .data
        .balign 4
string: .asciz "\n%d + %d = %d\n"
a:      .word   33
b:      .word   44
```

```
c:      .word   0                @ will contain a+b

@ ---------------------------------------
@       Code Section
@ ---------------------------------------

.text
.global main
.extern printf

main:
        push    {ip, lr}         @ push return address + dummy register for alignment
        ldr     r1, =a           @ get address of a into r1
        ldr     r1, [r1]         @ get a into r1
        ldr     r2, =b           @ get address of b into r2
        ldr     r2, [r2]         @ get b into r2
        add     r1, r1, r2       @ add r1 to r2 and store into r1
        ldr     r2, =c           @ get address of c into r2
        str     r1, [r2]         @ store r1 into c

        ldr     r0, =string      @ get address of string into r0
        ldr     r1, =a           @ r1 <- a
        ldr     r1, [r1]
        ldr     r2, =b           @ r2 <- b
        ldr     r2, [r2]
        ldr     r3, =c           @ r3 <- c
        ldr     r3, [r3]
        bl      printf           @ print string and pass params into r1, r2, and r3
        pop     {ip, pc}         @ pop return address into pc
```

Activity 5. Passing parameters by value

```
@  sum3.s
@  D. Thiebaut
@  Illustrates how to pass 2 ints by value
@  to a function that adds them up and returns
@  the sum in r0.
@ ---------------------------------------
@    Data Section
@ ---------------------------------------

        .data
        .balign 4
string: .asciz "\n%d + %d = %d\n"
a:      .word   33
b:      .word   44
c:      .word   0               @ will contain a+b

@ ---------------------------------------
@    Code Section
@ ---------------------------------------

.text
.global main
.extern printf

@ ---------------------------------------
@ sumFunc: gets 2 ints in r1 and r2, adds
@ them up and saves the results in
@ r0.
sumFunc:
        push    {ip, lr}
        add     r0, r1, r2
        pop     {ip, pc}

@ ---------------------------------------
@ main: passes 2 ints to sumFunc and prints
@ the resulting value using printf
main:   push    {ip, lr}         @ push return address + dummy register for alignment
        ldr     r1, =a           @ get address of a into r1
        ldr     r1, [r1]         @ get a into r1
        ldr     r2, =b           @ get address of b into r2
        ldr     r2, [r2] @ get b into r2

        bl      sumFunc          @ pass (r1, r2) to sumFunc
@ gets sum back in r0
        ldr     r2, =c           @ get address of c into r2
        str     r0, [r2]         @ store r0 into c

@ printf( "%d + %d = %d\n", r1, r2, r3 )
@ (format-string address passed in r0)
        Ldr     r0, =string      @ get address of string into r0
        Ldr     r1, =a           @ r1 <- a
        Ldr     r1, [r1]
        Ldr     r2, =b           @ r2 <- b
        Ldr     r2, [r2]
        Ldr     r3, =c           @ r3 <- c
        Ldr     r3, [r3]
        Bl      printf           @ print string and pass params into r1, r2, and r3
@ return to OS
        pop     {ip, pc} @ pop return address into pc
```

At **steps 7 and 8**, explain the difference between passing the parameters by value or by reference to the function *sumFunc*. Show a printout of the correct execution of both examples.

Activity 6. Passing parameters by reference

```
@ sum4.s
@   D. Thiebaut
@   Illustrates how to pass 2 ints by reference
@   to a function that adds them up and returns
@   the sum in r0.

@ ---------------------------------------
@       Data Section
@ ---------------------------------------

        .data
        .balign 4
string: .asciz "\n%d + %d = %d\n"
a:      .word   33
b:      .word   44
c:      .word   0                @ will contain a+b
d:      .word   55
e:      .word   22

@ ---------------------------------------
@       Code Section
@ ---------------------------------------

.text
.global main
.extern printf

@ ---------------------------------------
@ sumFunc: gets 2 ints in r1 and r2, adds
@       them up and saves the results in
@       r0.
sumFunc:
        push    {ip, lr}
        ldr     r1, [r1]
        ldr     r2, [r2]
        add     r0, r1, r2
        pop     {ip, pc}

@ ---------------------------------------
@ printFunc: prints [r1], [r2], r3 in this way.
@ printf( "%d + %d = %d\n", r1, r2, r3 )
@ (format-string address passed in r0)
printFunc:
        push    {ip, lr}
        ldr     r0, =string     @ get address of string into r0
        ldr     r1, [r1]
        ldr     r2, [r2]
        mov     r3, r3          @ not necessary...
        bl      printf          @ print string and pass params into r1, r2, and r3

        pop     {ip, pc}

@ ---------------------------------------
@ main: passes 2 ints to sumFunc and prints
@ the resulting value using printf
main:   push    {ip, lr}        @ push return address + dummy register for alignment

@ c = a + b
        ldr     r1, =a          @ get address of a into r1
        ldr     r2, =b          @ get address of b into r2

        bl      sumFunc         @ pass (r1, r2) to sumFunc
@ gets sum back in r0
        ldr     r3, =c          @ get address of c into r2
        str     r0, [r3]        @ store r0 into c

@ print a + b = c
        ldr     r1, =a
        ldr     r2, =b
        ldr     r3, =c
        ldr     r3, [r3]
        bl      printFunc

@ return to OS
        pop     {ip, pc}        @ pop return address into pc
```

At **steps 7 and 8**, explain the difference between passing the parameters by value or by reference to the function *sumFunc*. Show a printout of the correct execution of both examples.

Activity 7. Using the C-function *scanf()* for User input

At **step 9**, explain which parameters are passed to the *scanf* function and which are the registers needed for this purpose. Show a printout of the correct execution of the example.

Activity 8. Pausing the program using the *sleep()/usleep()* C functions

At **step 10**, explain what are the parameters passed to the *sleep()* and *usleep()*. Indicate what are the registers needed. Show a printout of the correct execution of the example.

Activity 9. Recursive Towers of Hanoi

Activity 10. Blinking LED