

Lab 2

Humam Amouri & Yonis Said

Lab 2 report

Yonis Said & Humam Amouri



Task 1

Before starting with the task we were supposed to do some preparations in order to run raspbian emulator on our device.

The task assigned to us here was that we write an assembly program which takes 2 integers, adds them, and then call the external function "printf" to display the results. The following code snippet shows the code we used to accomplish this task

```
GNU nano 2.2.6      File: program.s

.data
string: .asciz "%d"
.text
.global main
.extern printf
main:
    push {ip,lr}

    mov r2, #1
    mov r3, #2
    add r1, r2, r3
    ldr r0,=string
    bl printf

    pop {ip,pc}
```

[Read 15 lines]

^G Get Help	^O WriteOut	^R Read File	^Y Prev Page	^K Cut Text	^C Cur Pos
^X Exit	^J Justify	^W Where Is	^V Next Page	^U UnCut Text	^T To Spell

Code snippet 1

In pseudocode the main process in code snippet 1 could be described as;

- Store the value 1 in Register 2
- Store the value 2 in Register 3
- Add the values in Register 2 and 3 and store the result in Register 1
- Load the string(defined in the header) in Register 0
- Execute "printf"


The function "printf" begins always by checking the first register (r0) where it expects a string. In our case the string can be thought of as a specifier. This means that we need to state the specifier "%d" first to be able to print out integers. What "printf" does after that is to check the integer value stored in the following register (r1) and prints it out.

When we finished writing our code, we proceeded to assemble, and link the file into an executable. After running it the answer "3" was printed in the terminal which proves that our code works.

Task 2

Task 2 can be divided into several parts.

The first part is to write a C-file which will be linked together with an assembly file, see code snippet 2. The task is to write a function which takes an integer as an argument and converts it into hexadecimal. Fortunately this is already a built-in function in the library with “printf” which can be achieved by using the specifier “%x”. It requires another argument following the string and it will be converted to hexadecimal, assuming its a valid argument. So we created a function which takes an integer as argument and prints out the values corresponding to hexadecimal value.



```
GNU nano 2.2.6      File: int_out.c

#include <stdio.h>

void int_out(int num){
    printf("%x\n", num);
}
```

Code snippet 2

The following part of the task wanted us to write an assembly program which loads the number “4” in (r0) and calls the “int_out.c” function with the intention of using the value as the argument to print in hexadecimal. The program is intentionally very similar to the code from first task as we followed the tracks of using an external function with a loaded value, see code snippet 3. So we compile and link our C and assembly programs together and run it, getting the hexadecimal of 4 which is 4.

```

.text
.global main
.extern int_out
main:
    push {ip,lr}

    mov r0, #4
    ldr r0,=myvar
    mov r0,[r0]

    bl int_out

    pop {ip,pc}

[ Wrote 17 lines ]

pi@raspberrypi:~ $ as cass.s -o cass.o
pi@raspberrypi:~ $ gcc cass.o int_out.c -o cass_out
pi@raspberrypi:~ $ ./cass_out
4

```

Code snippet 3

Next task requires us to use a hexadecimal int as an input and to verify that bit shifting works as expected. Code snippet 4 below shows the code we used to solve this task.

```

myvar: .int 0xBD5B7DDE

.text
.global main
.extern int_out
main:
    push {ip,lr}

    mov r0, #4
    ldr r0,=myvar
    ldr r0, [r0]
    asr r0, #1

    bl int_out

    pop {ip,pc}

[ Wrote 19 lines ]

pi@raspberrypi:~ $ as cass.s -o cass.o
pi@raspberrypi:~ $ gcc cass.o int_out.c -o cass_out
pi@raspberrypi:~ $ ./cass_out
deadbeef

```

Code snippet 4

In pseudocode the main process could be described as:

- Store the address of the hexadecimal int.
- Store the value of the address.
- Bit shift to the right.
- Print out the result.

As seen in the picture above the output that we got of bit shifting one bit to the right was “deadbeef”.

To complete task 2 we were required to replace our function (int_out) with” printf”. Modifying the code to work with “printf” wasn’t that difficult to achieve because task 1 had the same concept, see code snippet 5.

```
.global main
.extern printf
main:
    push {ip,lr}

    //    mov r0, #4
        ldr r0,=string
        ldr r1,=myvar
        ldr r1, [r1]
        asr r1, #1

    bl printf

    pop {ip,pc}
```

[Wrote 20 lines]

```
pi@raspberrypi:~$ as cass.s -o cass.o
pi@raspberrypi:~$ gcc cass.o int_out.c -o cass_out
pi@raspberrypi:~$ ./cass_out
deadbeefpi@raspberrypi:~$ as cass.s -o cass.o
pi@raspberrypi:~$ as cass.s -o cass.o
pi@raspberrypi:~$ gcc cass.o -o cass
pi@raspberrypi:~$ ./cass
deadbeefpi@raspberrypi:~$ _
```

Code snippet 5