

Lab 2



Task 1: Basic Setup

The task was to load two immediate values into two registers and add them. Then you print the result with the printf function that you import.

```
.data
string: .asciz "\na + b = %d\n"
a: .word 0 @ for storing answer

.text
.global main
.extern printf

main:
    push {ip, lr}
    ldr r1, = #5
    ldr r2, = #7

    add r1,r1,r2

    ldr r0, = string
    bl printf

    pop { ip, pc }
```

Code 1

```
pi@raspberrypi: $ ./adder2
a + b = 12
```

Code 2

Task 2: Shifting Integers

Task 2

The first task here was to write a C-program that takes an integer and returns that integer in

```
#include <stdio.h>

void int_out(int i)
{
    printf("\nNumber = 0x%x\n", i);
}
```

Code 3

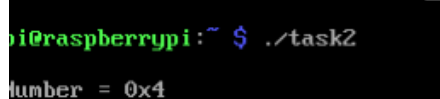
hexadecimal notation. We then tested the so that the function worked as expected, which it did.

The next assignment was to include the above figured c-function into an assembly program.

```
.text
global main

extern int_out

main:
    push {ip, lr}
    mov r0, #4
    bl int_out
    pop { ip, pc }
```



Code 4

Here, int_out was declared as an external function. The value 4 was then stored into register r0, then int_out was branched into taking r0 as an argument. The program was then executed to verify the result.

```
.text
.global main

.extern int_out

main:
    push {ip, lr}

    ldr r0, = 0xBD5B7DDE
    asr r0, r0, #1

    bl int_out

    pop { ip, pc }
```



```
pi@raspberrypi:~ $ ./task2_2
Number = 0xdeadbeef
```

Code 5

For the next task, the integer 0xBD5B7DDE was loaded into memory and the instructions was to bit shift it and then print the result in hexadecimal form. This was done by using the asr-function, which bit shifted the integer one step to the right. The function used in the previous task, int_out, was then called in the same way as above, which gave us the result, deadbeef.

```

string: .asciz "\nNumber = %x\n"
.text
.global main

@.extern int_out
.extern printf
main:
    push {ip, lr}

    ldr r0, = 0xBD5B7DDE
    asr r0, r0, #1

    mov r1, r0
    ldr r0, = string
    bl printf

    pop {ip, pc }

pi@raspberrypi:~ $ ./task2_3
Number = deadbeef

```

The last assignment in Task 2 was to print out the hexadecimal answer directly in the assembly-program. This was done defining a string for the extern printf-function, using the %x to format the output into hexadecimal.

Code 6

Task 3: Assembly in C

In this task, we were asked to implement a c-function, xor, that calculated the bit-wise xor for two integers. This was done by using the built in command ^, which performs bit-wise xor.

Bit-wise XOR is performed by comparing the binary representation of the integers bit by bit. These bits are then compared with an exclusive or, which truth table can be seen below. The result is that if the bits are different, the output will be one, otherwise zero. This is then done for each bit, and then the output for each bit is summed up.

0	0	0
0	1	1
1	0	1
1	1	0

```

int xor(int a, int b)
{
    int result = a^b;
    return result;
}

int main()
{
    int a = 2;
    int b = 4;
    int res = xor(a,b);
    printf("%d\n", res);
    return 0;
}

```

Code 7

Then, we wrote a function to calculate the bitwise xor using assembly only. This was done using the function EOR and then load the result into register 1, the string for the output is loaded into register 0, and then branched into printf.

```

.data
string: .asciz "\nNumber = %x\n"
.text
.global main
main:
    push {ip, lr}

    mov r0, #2
    mov r1, #4

    EOR r1, r0, r1

    ldr r0, = string
    bl printf

    pop { ip, pc }

```

Code 8

For the last task, the problem was to integrate an assembly function to be executable inside c. This was done by declaring a function in assembly that was then called inside a c-file. These files were then compiled and the result can be seen in picture *Code 9*.

```

#include <stdio.h>
extern int axor(int a, int b);
int main()
{
    int a = 2;
    int b = 4;
    printf("\nXor A = %d and B = %d result = %d\n", a, b, axor(a,b));
    return 0;
}

```

```

.type axor%function
axor:
    .fnstart
    EOR r0, r0, r1
    bx      lr
    .fnend

pi@raspberrypi:~ $ ./task3
Xor A = 2 and B = 4 result = 6
pi@raspberrypi:~ $

```

Code 9

Code 10

Conclusion

We thought that it was interesting to try out assembly and see how it can be integrated with C. Most of the lab went well, the biggest problem that we had was to get information about how to compile the programs, especially when trying to compile with C-files. We discovered that we had to compile the C-file on the raspberry with the correct flags.