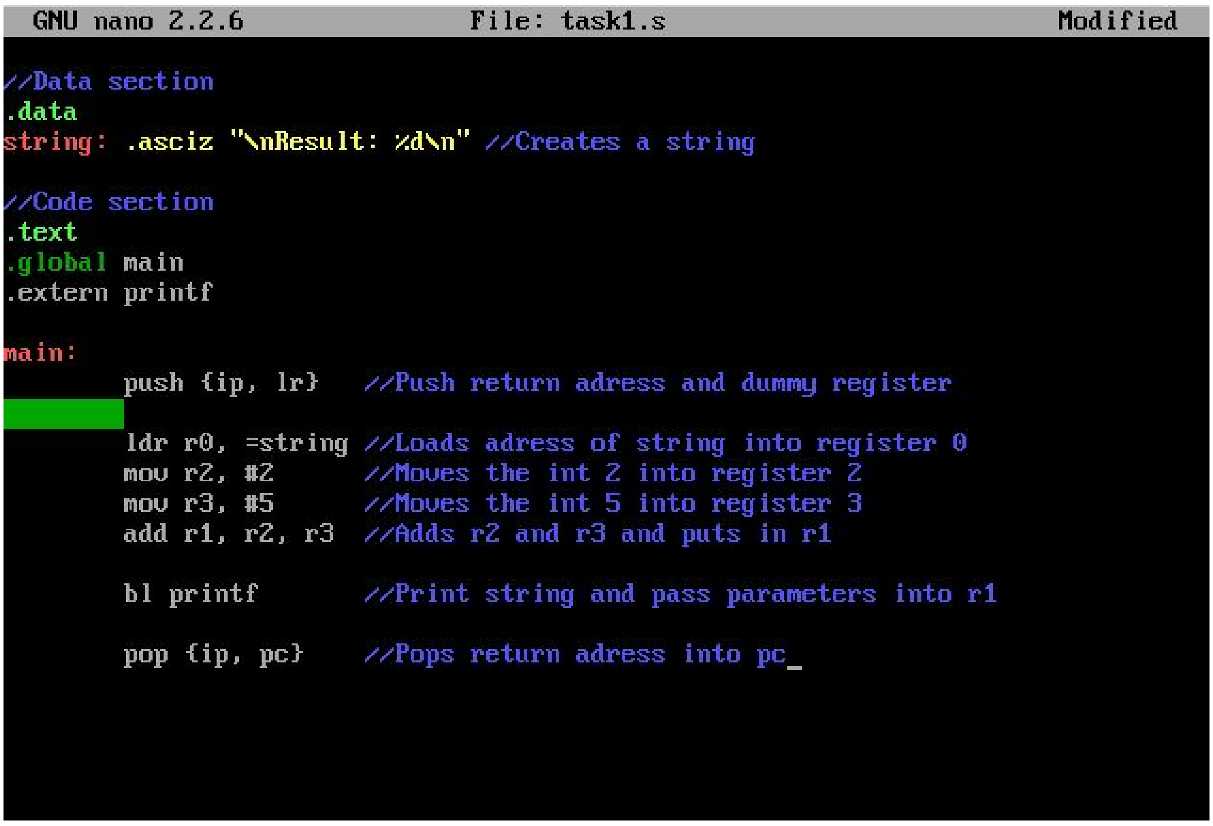
Lab 2: ARM Assembly Language

Task 1

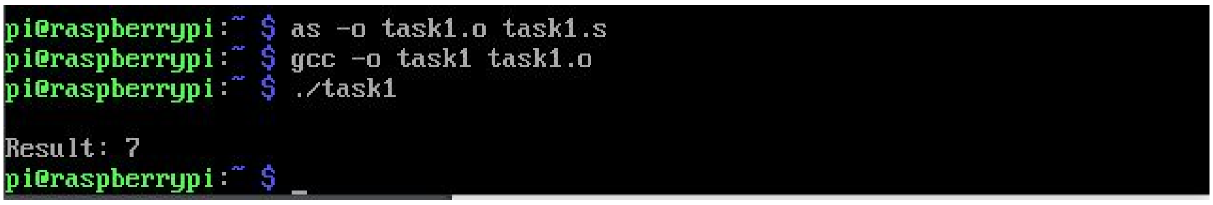
The task was to write an assembly program which adds two integers and displays the result using the external printf function. Writing the code was quite simple, except for calling the C function printf() in assembly. After viewing the example put up on blackboard, as well as using the links provided with the lab instructions, a solution was found.



```
  GNU nano 2.2.6               File: task1.s                        Modified

//Data section
.data
string: .asciiz "\nResult: %d\n" //Creates a string

//Code section
.text
.global main
.extern printf

main:
        push {ip, lr}   //Push return adress and dummy register

        ldr r0, =string //Loads adress of string into register 0
        mov r2, #2      //Moves the int 2 into register 2
        mov r3, #5      //Moves the int 5 into register 3
        add r1, r2, r3  //Adds r2 and r3 and puts in r1

        bl printf       //Print string and pass parameters into r1

        pop {ip, pc}    //Pops return adress into pc_
```

Image 1

The code in Image 1 is the one we ended up with after completing the task, and the output after executing can be seen in image 2:



```
pi@raspberrypi:~ $ as -o task1.o task1.s
pi@raspberrypi:~ $ gcc -o task1 task1.o
pi@raspberrypi:~ $ ./task1

Result: 7
pi@raspberrypi:~ $ _
```

Image 2

Task 2

Like the previous task this one was also fairly simple and straightforward, especially since each step in the instructions were so short. The most difficult part where to link the **int_out** function with the assembly code into an executable, but after reading a bit on how the gcc command works it was quite intuitive. Basically the assembly code is assembled first and then the two files are linked and compiled simultaneously in the line
gcc int_out.c task2.o -o task2exec into the executable file **task2exec**.

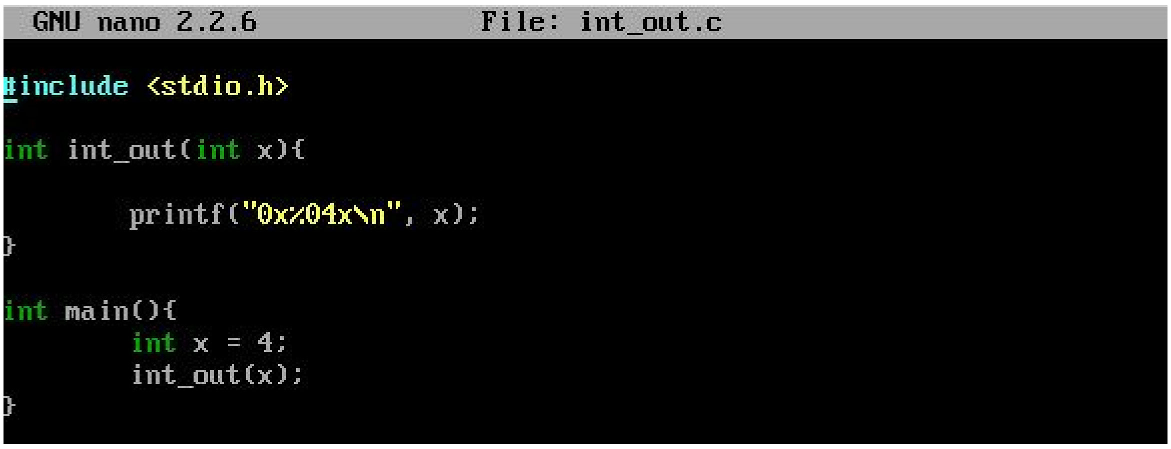
Image 3 is the C code which converts an integer to hexadecimal and prints the result. We used the function printf with %x to convert and print.

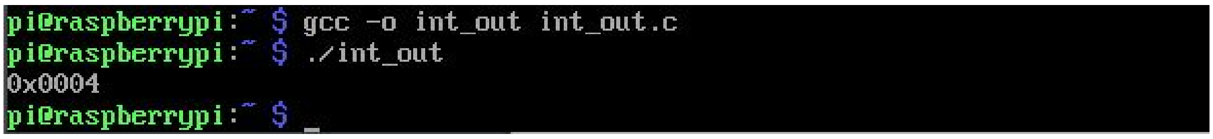
image 4 is the result displayed when the file is compiled and run.

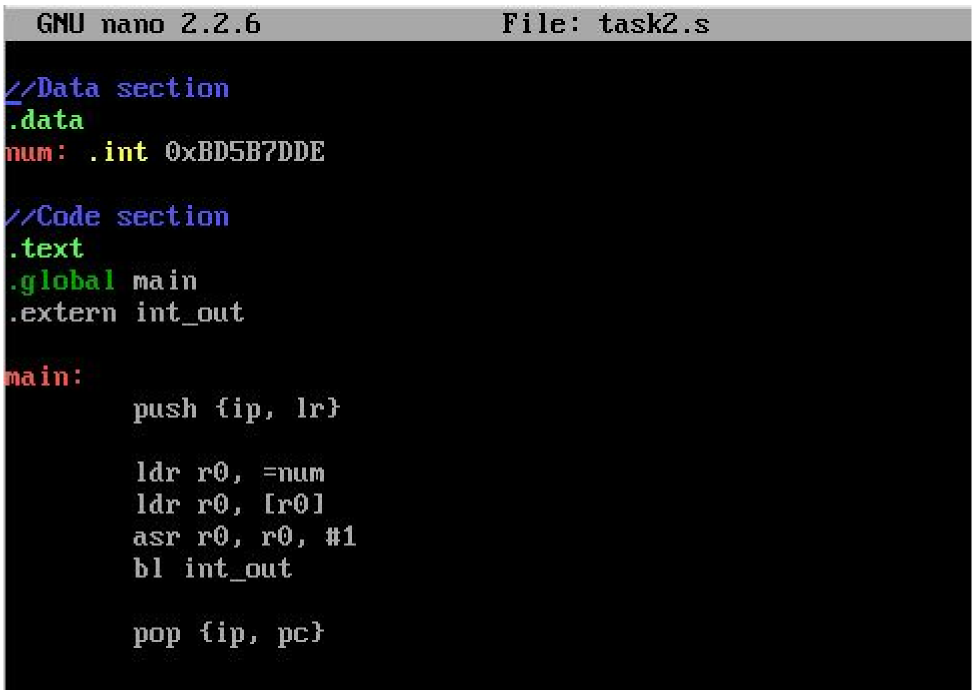
Image 5 is the assembly code which uses our C function **int_out**. In the code we have the hexadecimal number 0xBD5B7DDE which we subtract 1 from, and then send in the result into **int_out**.

Image 6 is the result displayed and the commands to assemble **task2.s** and compile/link the assembly and C code. The result is DEADBEEF.

Image 7 is the modified assembly code where we, like in task 1, created a string and loaded the string into r0 and branched the printf function to get the result in Image 8.

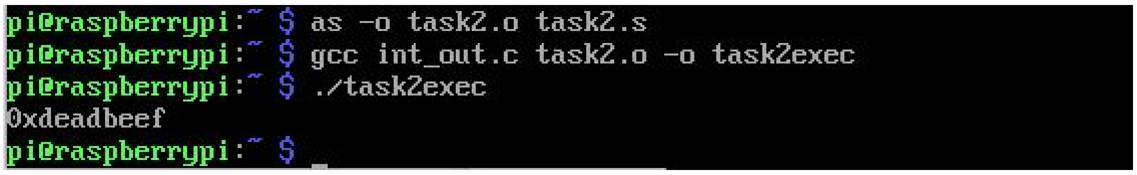

```
  GNU nano 2.2.6              File: int_out.c

#include <stdio.h>

int int_out(int x){

        printf("0x%04x\n", x);
}

int main(){
        int x = 4;
        int_out(x);
}
```
Image 3


```
pi@raspberrypi:~ $ gcc -o int_out int_out.c
pi@raspberrypi:~ $ ./int_out
0x0004
pi@raspberrypi:~ $ _
```
Image 4



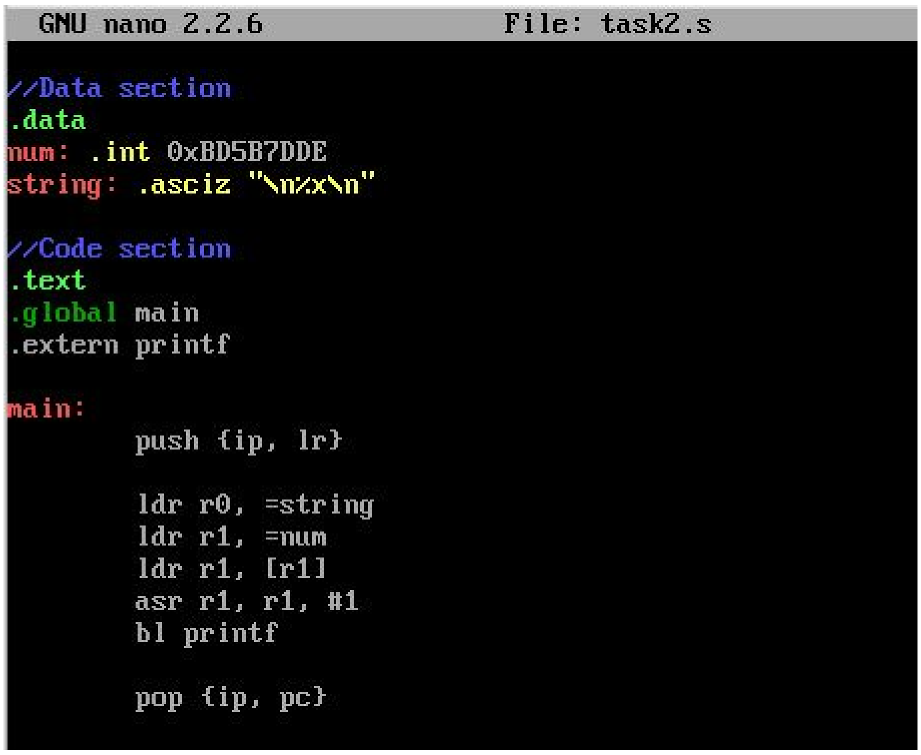
```
 GNU nano 2.2.6                File: task2.s

//Data section
.data
num: .int 0xBD5B7DDE

//Code section
.text
.global main
.extern int_out

main:
        push {ip, lr}

        ldr r0, =num
        ldr r0, [r0]
        asr r0, r0, #1
        bl int_out

        pop {ip, pc}
```

Image 5



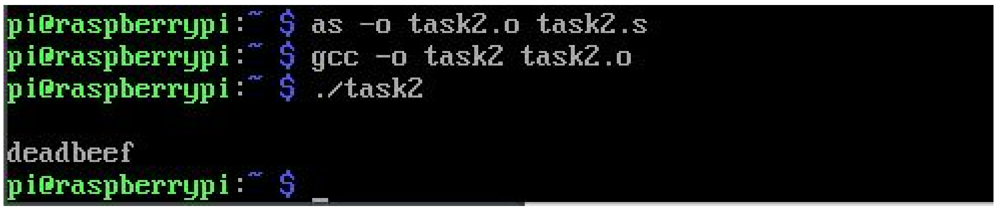
```
pi@raspberrypi:~ $ as -o task2.o task2.s
pi@raspberrypi:~ $ gcc int_out.c task2.o -o task2exec
pi@raspberrypi:~ $ ./task2exec
0xdeadbeef
pi@raspberrypi:~ $ _
```

Image 6



```
  GNU nano 2.2.6                 File: task2.s

//Data section
.data
num: .int 0xBD5B7DDE
string: .asciz "\n%x\n"

//Code section
.text
.global main
.extern printf

main:
        push {ip, lr}

        ldr r0, =string
        ldr r1, =num
        ldr r1, [r1]
        asr r1, r1, #1
        bl printf

        pop {ip, pc}
```

Image 7



```
pi@raspberrypi:~ $ as -o task2.o task2.s
pi@raspberrypi:~ $ gcc -o task2 task2.o
pi@raspberrypi:~ $ ./task2

deadbeef
pi@raspberrypi:~ $ _
```

Image 8