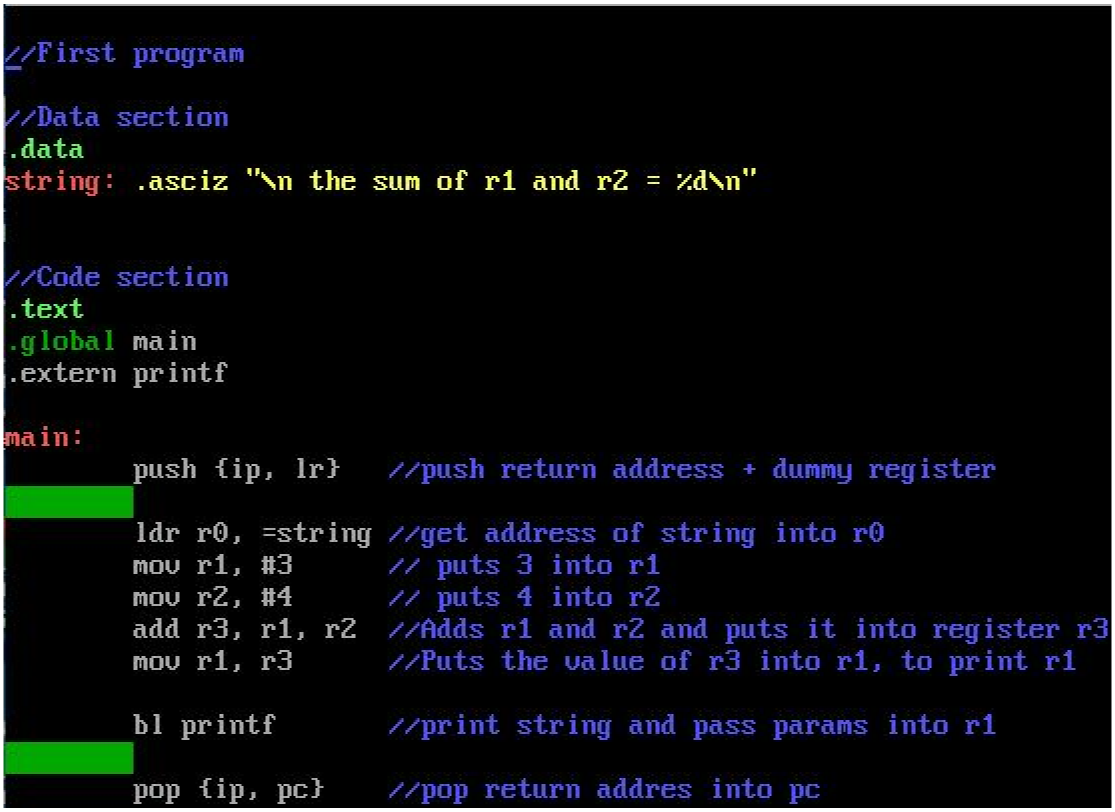
Rasmus Bäck
Andreas Johansson
Civilingenjör DataTeknik Year 3

Computer Science Laboration 2

Task 1

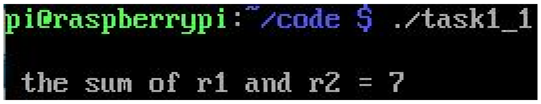


```
//First program

//Data section
.data
string: .asciz "\n the sum of r1 and r2 = %d\n"


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

main:
        push {ip, lr}   //push return address + dummy register

        ldr r0, =string //get address of string into r0
        mov r1, #3      // puts 3 into r1
        mov r2, #4      // puts 4 into r2
        add r3, r1, r2  //Adds r1 and r2 and puts it into register r3
        mov r1, r3      //Puts the value of r3 into r1, to print r1

        bl printf       //print string and pass params into r1

        pop {ip, pc}    //pop return addres into pc
```

Image 1

To find out how we should write the assembly code we used the example that Todor gave us and the references that the lab instructions provided.

After finishing the code we first used the terminal to compile the .as code into and object file (.o) with this command: **as task1_1.s -o task1_1.o**. Then we compiled the object file into and executable with this command: **gcc task1_1.o -o task1_1**. The output that our program gave us can be found in image 2.

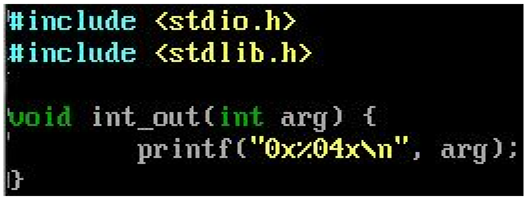


```
pi@raspberrypi:~/code $ ./task1_1

 the sum of r1 and r2 = 7
```

Image 2

Rasmus Bäck
Andreas Johansson
Civilingenjör DataTeknik Year 3

Task 2

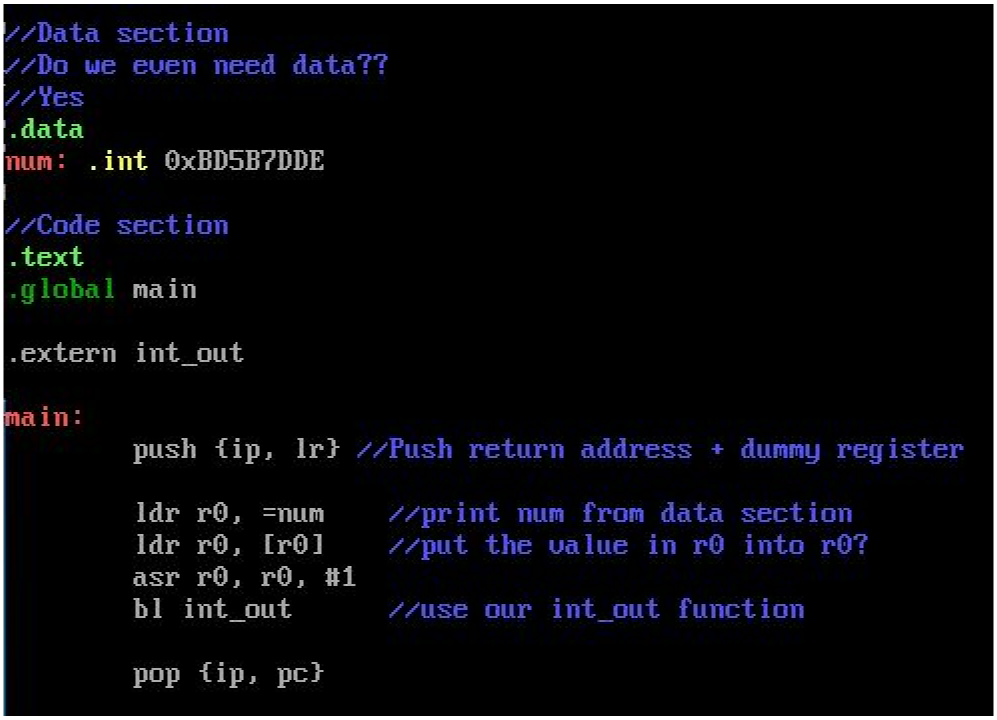


```
#include <stdio.h>
#include <stdlib.h>

void int_out(int arg) {
        printf("0x%04x\n", arg);
}
```

Image 3

First we wrote our c-function. The code can be seen in image 3. It takes an integer as argument and prints it with hexadecimal notation. Then we coded a main function to test our print function and it worked as we expected.

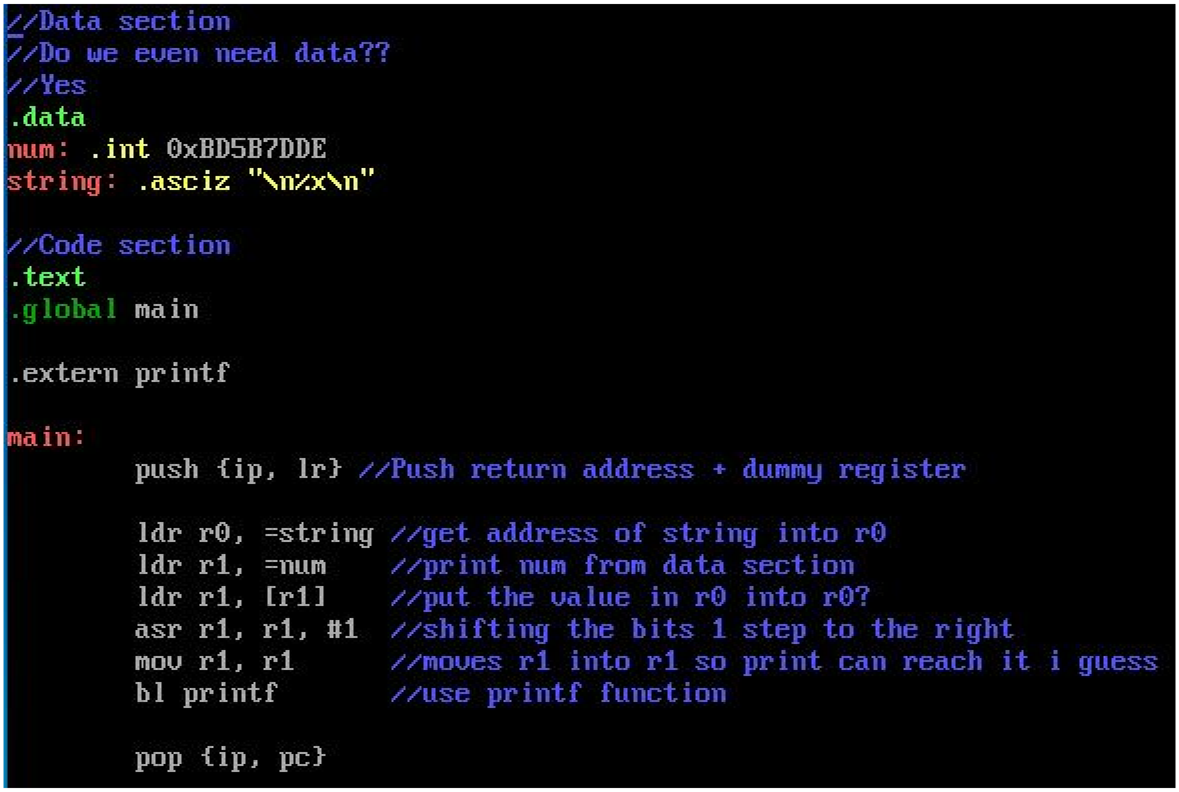
After we had tested the **int_out** function we compiled it into an object file using this command in the terminal we wrote our assembly code that can be seen in image 4. First we were asked to print the immediate value 4, so instead of **ldr ro, =num, ldr r0, [r0]** and **asr r0, r0, #1** we had **ldr r0, #4**, then we assembled the assembly file into an object file with this command: **as task2.s -o task2.o** then we linked the object file with the int_out function and made it into an executable file called task2exec with this command: **gcc int_out.c task2.o -o task2exec.**



```
//Data section
//Do we even need data??
//Yes
.data
num: .int 0xBD5B7DDE

//Code section
.text
.global main

.extern int_out

main:
        push {ip, lr} //Push return address + dummy register

        ldr r0, =num    //print num from data section
        ldr r0, [r0]    //put the value in r0 into r0?
        asr r0, r0, #1
        bl int_out      //use our int_out function

        pop {ip, pc}
```

Image 4

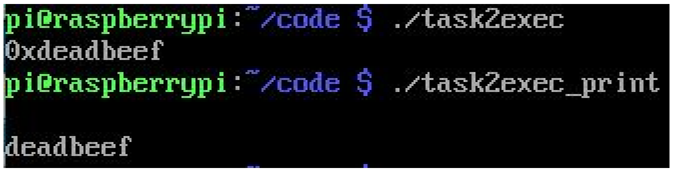
For printing the integer 0xBD5B7DDE instead of the immediate value 4 we modified the code so it looks exactly like image 4, the **asr r0, r0, #1** line of code arithmetically shifts the bit 1 step to the right. Then we compiled the code the same way we did before **as task2.s -o task2.o** first then **gcc int_out.c task2.o -o task2exec**, the output och the file task2exec can be seen in image 6.



```
//Data section
//Do we even need data??
//Yes
.data
num: .int 0xBD5B7DDE
string: .asciz "\n%x\n"

//Code section
.text
.global main

.extern printf

main:
        push {ip, lr} //Push return address + dummy register

        ldr r0, =string //get address of string into r0
        ldr r1, =num    //print num from data section
        ldr r1, [r1]    //put the value in r0 into r0?
        asr r1, r1, #1  //shifting the bits 1 step to the right
        mov r1, r1      //moves r1 into r1 so print can reach it i guess
        bl printf       //use printf function

        pop {ip, pc}
```

Image 5

Lastly instead of using or own c function **int_out** we called the external function **printf**, the assembly code for that can be seen in image 5. Then we compiled the code into an object file with this command: **as task2_print.s -o task2_print.o**, after that we compiled the object file into an executable called **task2exec_print** using this command: **gcc task2_print.o -o task2exec_print**. The output of **task2exec_print** can be seen in image 6.



```
pi@raspberrypi:~/code $ ./task2exec
0xdeadbeef
pi@raspberrypi:~/code $ ./task2exec_print

deadbeef
```

Image 6