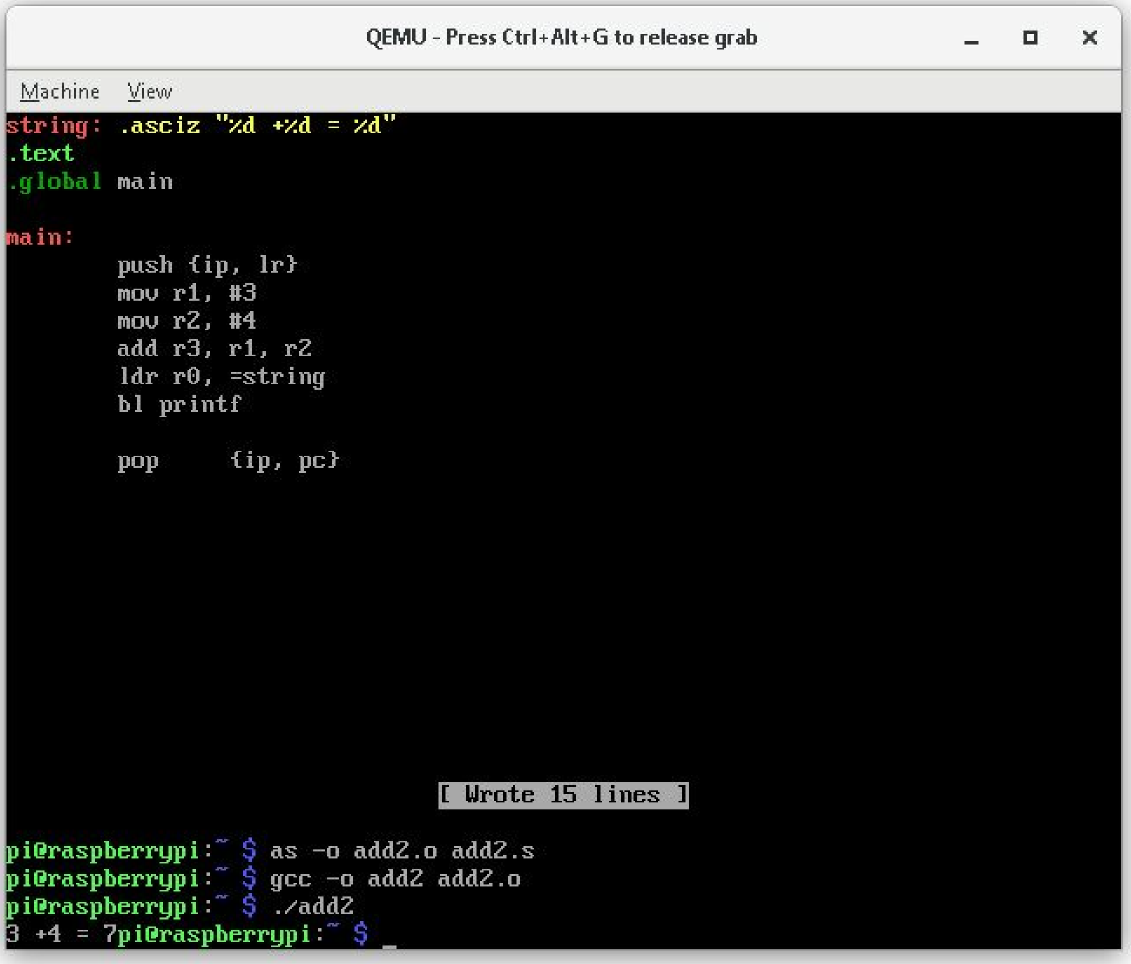
Datorteknik

Lab 2: ARM Assembly Language

Alen Gazibegovic
Thanadon Suriya

For future coding each significant line should have comments to explain but to clarify in the report we have chosen to talk about the implementation throughout.

Task 1: Basic Setup was completed according to instructions given in lab assignment sheet. The task is to loads two immediate integer values into the registers, add them, and then call the external function printf to display the result. Instructions said to compile the program using the assembler as and then link the object file into an executable, using gcc. We have verified that our code produces correct values.



```
string: .asciz "%d +%d = %d"
.text
.global main

main:
        push {ip, lr}
        mov r1, #3
        mov r2, #4
        add r3, r1, r2
        ldr r0, =string
        bl printf

        pop     {ip, pc}
```

```
[ Wrote 15 lines ]

pi@raspberrypi:~ $ as -o add2.o add2.s
pi@raspberrypi:~ $ gcc -o add2 add2.o
pi@raspberrypi:~ $ ./add2
3 +4 = 7pi@raspberrypi:~ $ _
```

This is done by the following explanation of the code
push{ip, lr} //push return address + register
mov r1, #3 //moves value 3 into register1
mov r2, #4 //move value 4 into register r2
add r3, r1, r2 //add r1 and r2 and store the result in r3

Datorteknik

Lab 2: ARM Assembly Language

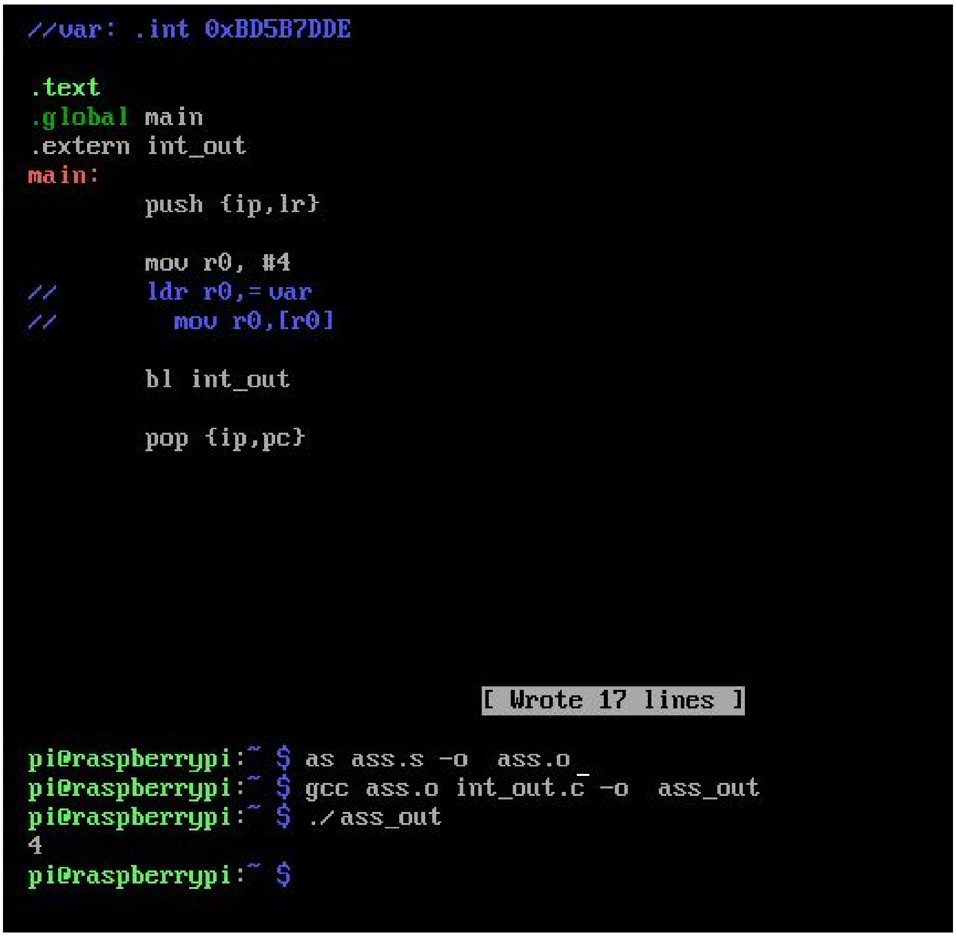
Alen Gazibegovic
Thanadon Suriya

bl printf print the result
pop {ip, pc} //return the address
Run the code by these commands
as - add2.o add2.s //assemble the file
gcc -o add2 add2.o //create add2.o and link this file to get an executable
./add2 for the result.

Task 2:

The second task is to write a C function int_out which takes an integer argument and returns the value in hexadecimal notation and writing a separate main function to test that int_out produces the expected output. When compiling - make int_out into object file. Lastly, we gonna load integer 0xBD5B7DDE and bitshift, both in bl int_out return of a word and using printf function.

But firstly to explain; <u>Write an assembly program which loads an immediate value of 4 into a register and then calls the external function int_out to print it</u>. The implementation for next task is commented out which we will explain hereafter.



```
//var: .int 0xBD5B7DDE

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

        mov r0, #4
//      ldr r0,= var
//        mov r0,[r0]

        bl int_out

        pop {ip,pc}
```

```
                                   [ Wrote 17 lines ]

pi@raspberrypi:~ $ as ass.s -o  ass.o
pi@raspberrypi:~ $ gcc ass.o int_out.c -o  ass_out
pi@raspberrypi:~ $ ./ass_out
4
pi@raspberrypi:~ $
```

Datorteknik

Lab 2: ARM Assembly Language

Alen Gazibegovic
Thanadon Suriya

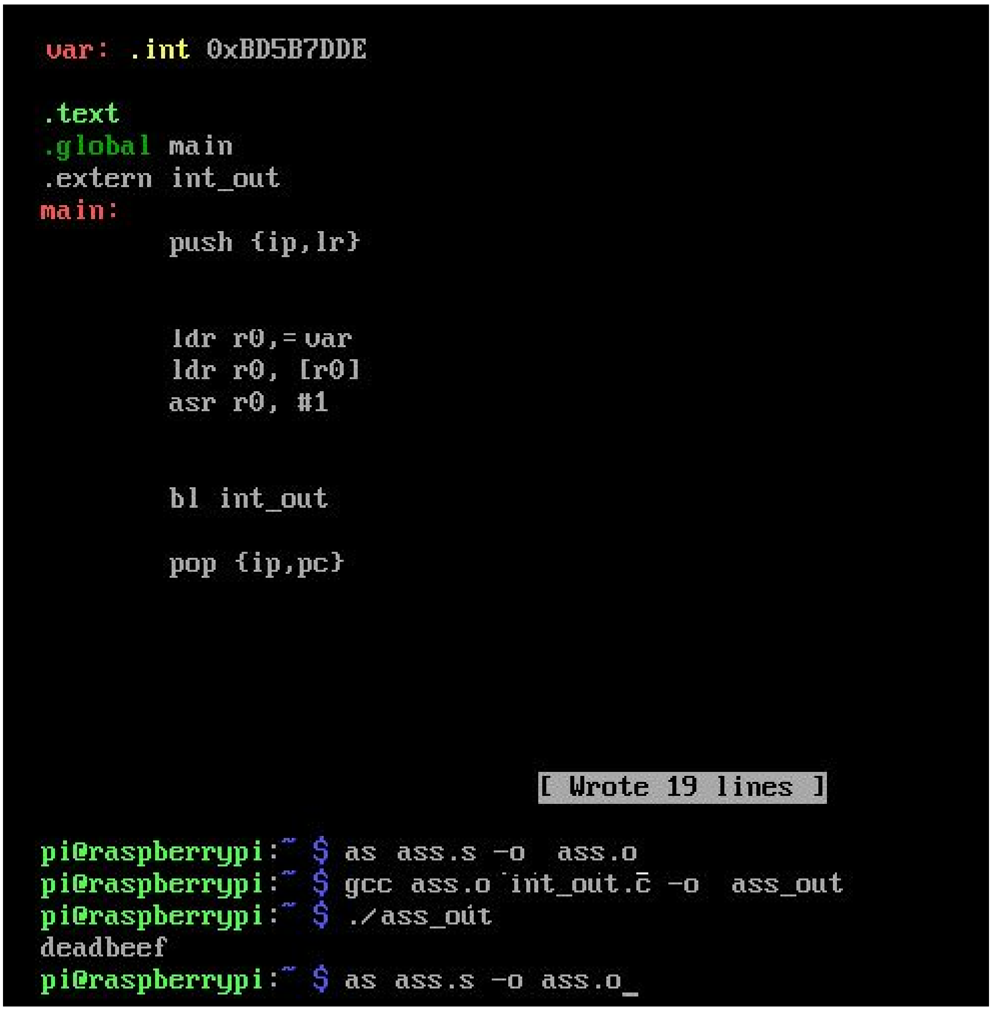
Code:

ldr instruction is used to load register with a 32-bit constant value or an adress. Here the value of 4 is moved into register 0 and printed out. bl int_out // the result
pop {ip, pc} //return the address The compilation is shown in the last 4 rows after pi@raspberrypi.. syntax.

Next;

For implementation with variable int 0xBD5B7DDE comments are removed and picture below explains the procedure.

Here variable value is entered with hexadecimal notation (0xBD5B7DDE). Using same type of implementation, but now with asr instruction (arithmetic right shift) we can return result. pop {ip, pc} returns the address and push{ip, lr} pushes return address + register.



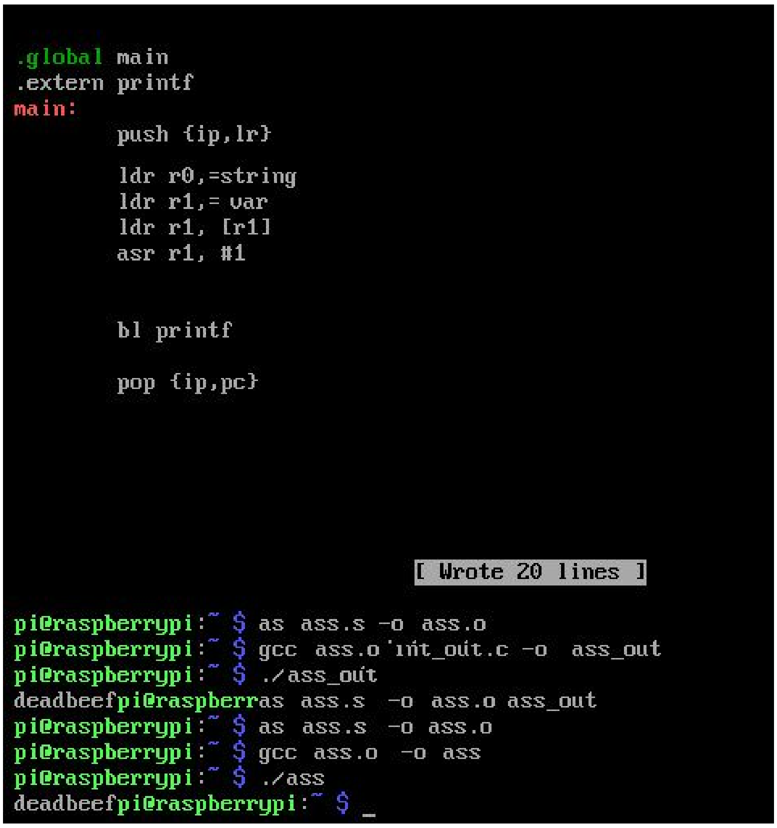
```
 var: .int 0xBD5B7DDE

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


        ldr r0,=var
        ldr r0, [r0]
        asr r0, #1


        bl int_out

        pop {ip,pc}





                               [ Wrote 19 lines ]

pi@raspberrypi:~ $ as ass.s -o  ass.o
pi@raspberrypi:~ $ gcc ass.o int_out.c -o  ass_out
pi@raspberrypi:~ $ ./ass_out
deadbeef
pi@raspberrypi:~ $ as ass.s -o ass.o_
```

Datorteknik

Lab 2: ARM Assembly Language

Alen Gazibegovic
Thanadon Suriya

This function however did not use printf function but returns via bl int_out. In last part of the task we need use the external function printf for the result. This is where we gonna add ldr r0, = string instead of ldr r0, = var, to hold our string like in first task. so r0 is string and r1 holds variable 0xBD5B7DDE. Now since r1 is ldr with variable when we call asr instruction in assembly like above but replacing bl int_out with our external print function we can return same result with string form. Picture of main and output below.



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

        ldr r0,=string
        ldr r1,= var
        ldr r1, [r1]
        asr r1, #1


        bl printf

        pop {ip,pc}







                                [ Wrote 20 lines ]

pi@raspberrypi:~ $ as ass.s -o ass.o
pi@raspberrypi:~ $ gcc ass.o int_out.c -o  ass_out
pi@raspberrypi:~ $ ./ass_out
deadbeefpi@raspberras ass.s  -o ass.o ass_out
pi@raspberrypi:~ $ as ass.s  -o ass.o
pi@raspberrypi:~ $ gcc ass.o  -o ass
pi@raspberrypi:~ $ ./ass
deadbeefpi@raspberrypi:~ $ _
```