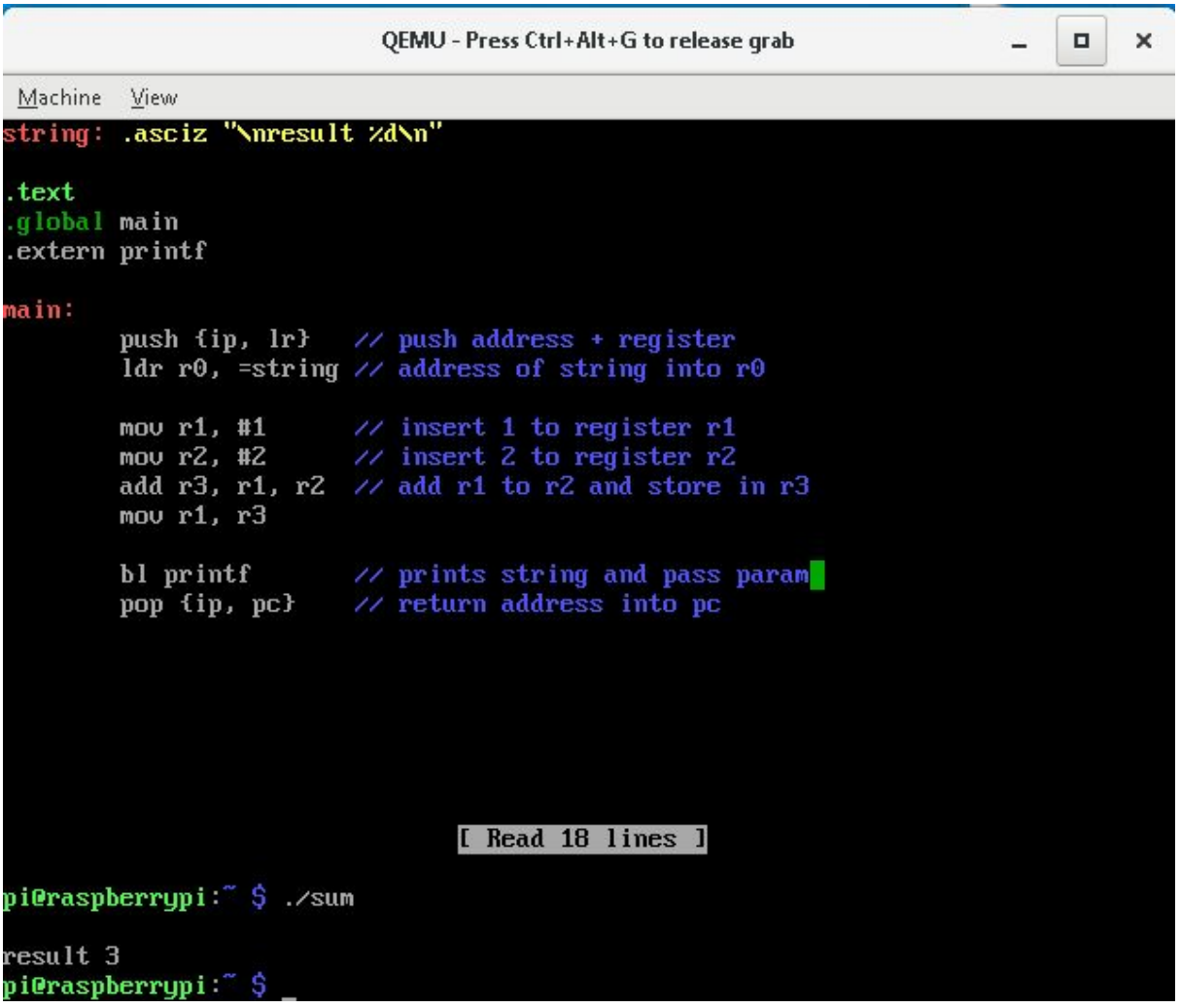
Datateknik
Lab 2: Digital Logic Circuits

Foad Ghafourian
Arash Mostafaee

The main focus of this lab is to learn how to program low level assembly code and be able to interface the assembly code from C by working in Raspbian emulator.

Task 1:

An assembly program that saves two integer into the registers, adds them together and displays the result by using external printf.



Screenshot of the file sum and the result

The assembly code is explained in inline comments, and here you can see used commands for this task

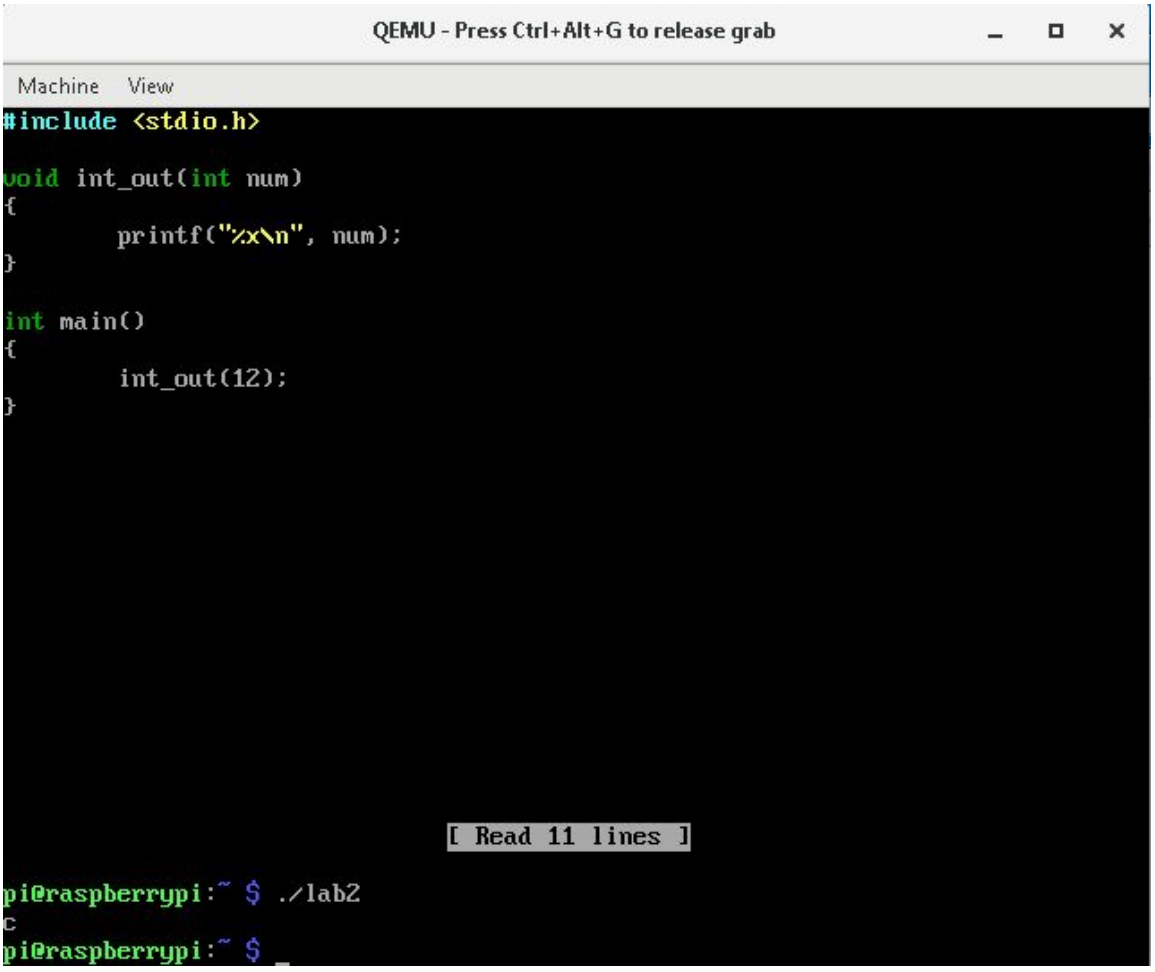
Commands:
as -o sum.o sum.s
to assemble the code to object file, which creates the object file sum.o

gcc -o sum sum.o
to generate the executable file

./sum
to run the program

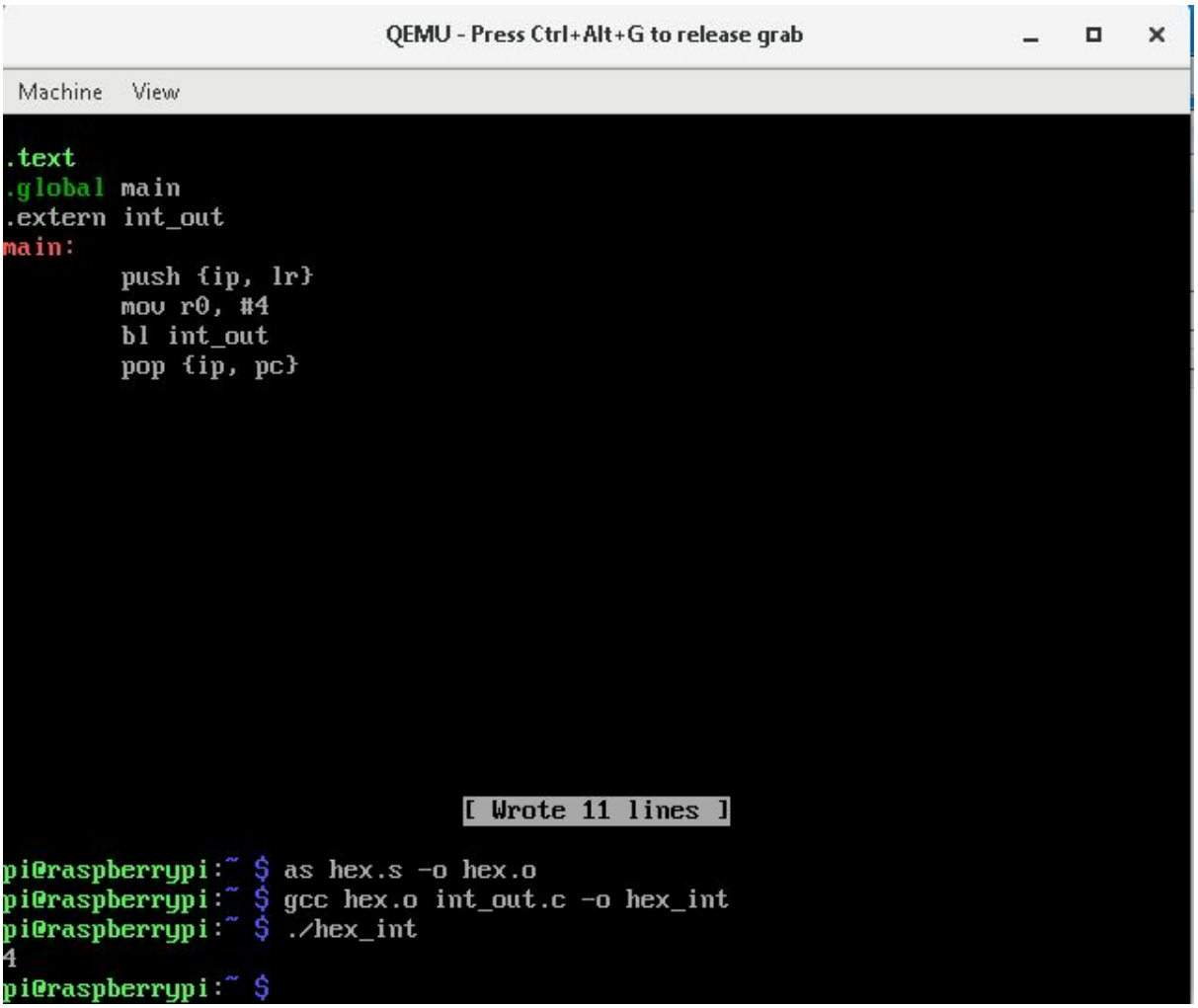
Task 2:

An assembly program that shifts an integer to the right and calls a C function to display the resulting value of the integer in hexadecimal notation.



C function that prints the value of an integer argument in hexadecimal notation.

A simple C function int-out that takes an integer argument and pints the value in hexadecimal. As seen in the screenshot of the code above, a main function has been added to test int-out. The command used for compiling int_out into an object file:
gcc lab2t2.c -o lab2



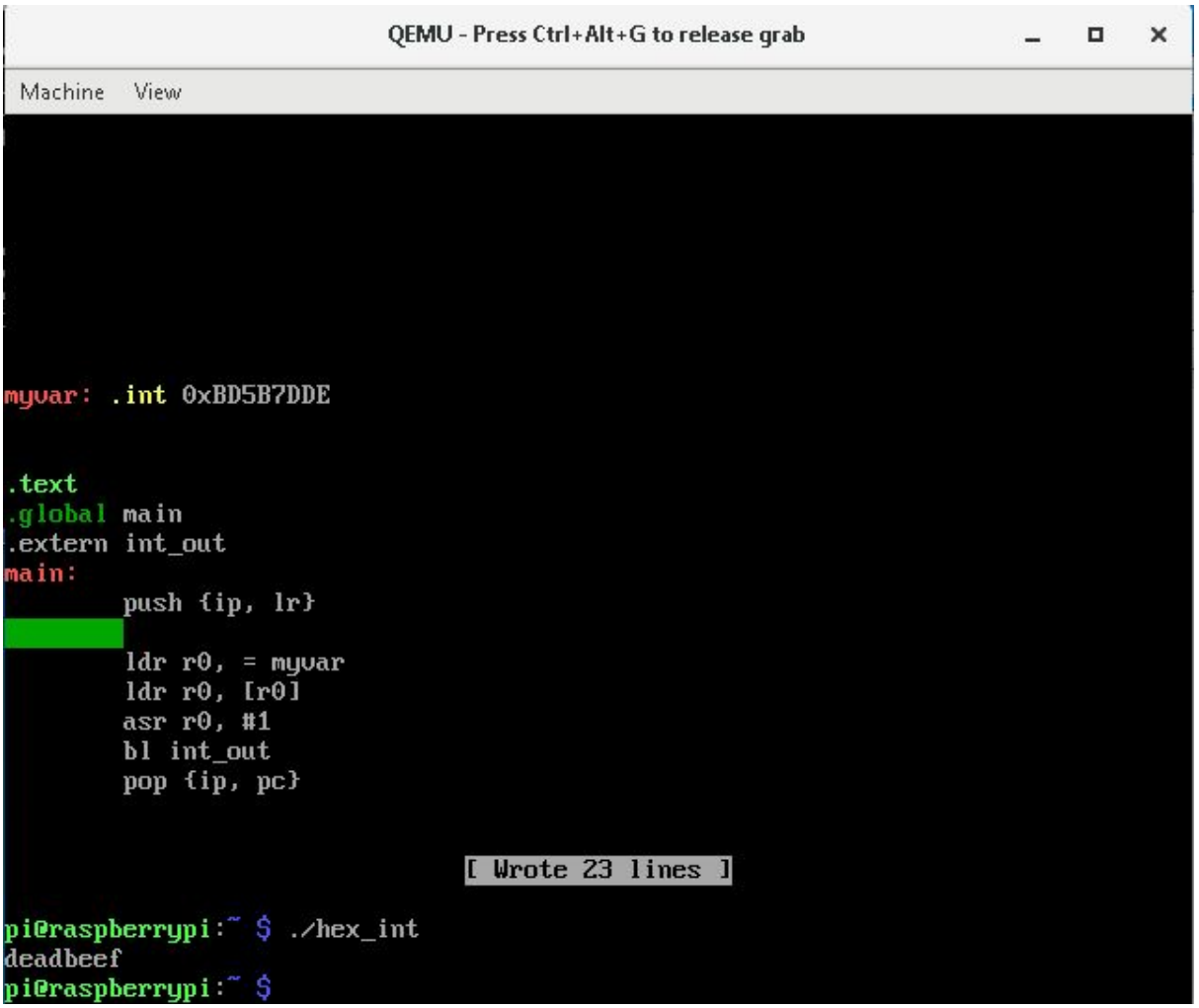
Screenshot of assembly program with external int_out

The screenshot above is from the assembly program which loads integer 4 into a register r0 and then prints it by calling the external function int_out.

- push {ip, lr}: push return address and register
- mov: insert integer 4 to register r0
- bl int_out: prints by calling on int_out
- pop{ip,pc}: return address into pc

The major difference with calling the external function int:out rather than printf is that printf checks the first register “r0” for a string. Our external function “int_out” doesn't work in the same way.

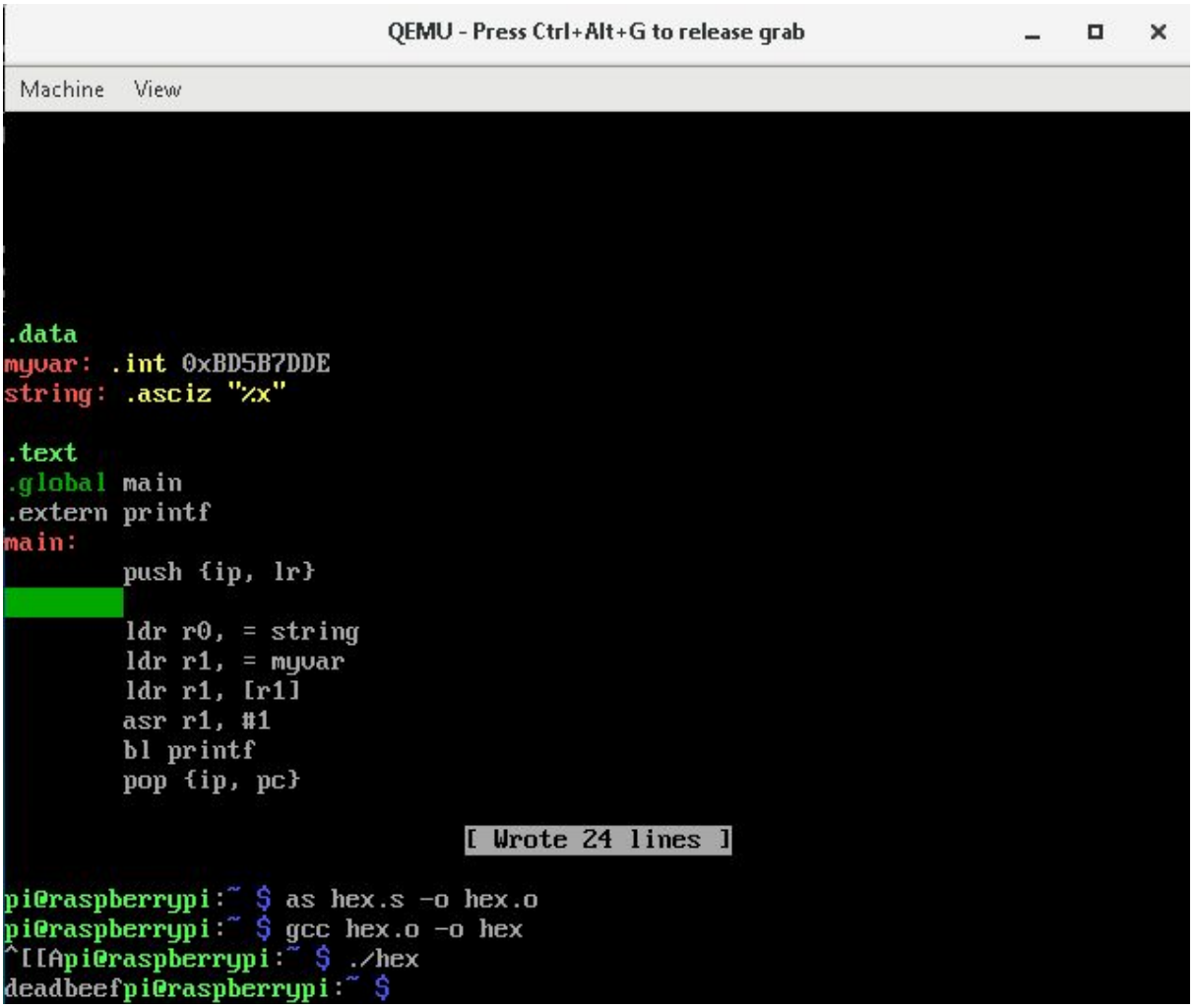
The command used to link the two object files into an executable:
gcc hex.o int_out.c -o hex_int



Assembly code with integer 0xBD5B7DDE instead of 4

- myvar: store address of the hexadecimal
- ldr r0, [r0]: store value of address
- asr r0, #1: bitshift to the right
- bl int_out: prints

We followed the note given to us regarding ‘ldr’ and ‘asr’ which load data from memory and does arithmetic shift for signed numbers. The result was ‘deadbeef’.



Assembly code with integer 0xBD5B7DDE instead of 4 and external print

As mentioned before, the difference here is using printf which checks the first register.