

Datateknik

Lab 2: Digital Logic Circuits



Foad Ghafourian
Arash Mostafaei

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.

```
Machine  View
string: .asciz "\nresult %d\n"

.text
.global main
.extern printf

main:
    push {ip, lr}    // push address + register
    ldr r0, =string  // address of string into r0

    mov r1, #1       // insert 1 to register r1
    mov r2, #2       // insert 2 to register r2
    add r3, r1, r2    // add r1 to r2 and store in r3
    mov r1, r3

    bl printf        // prints string and pass param
    pop {ip, pc}     // return address into pc

[ Read 18 lines ]

pi@raspberrypi:~ $ ./sum
result 3
pi@raspberrypi:~ $
```

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
```

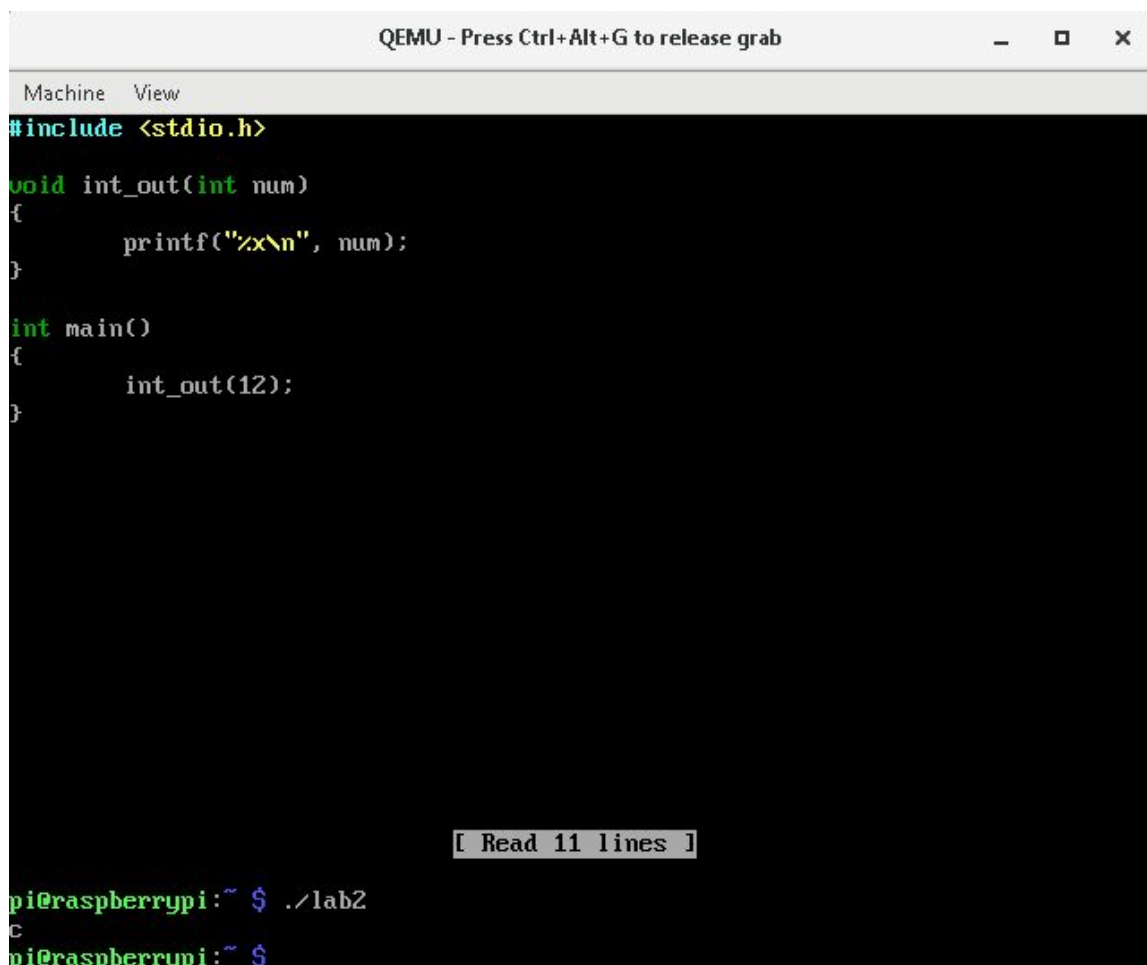
to generate the executable file

```
./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.

A screenshot of a QEMU terminal window. The title bar reads "QEMU - Press Ctrl+Alt+G to release grab". The terminal has a menu bar with "Machine" and "View". The main area displays C code with syntax highlighting:

```
#include <stdio.h>

void int_out(int num)
{
    printf("%x\n", num);
}

int main()
{
    int_out(12);
}
```

 At the bottom, a command prompt shows

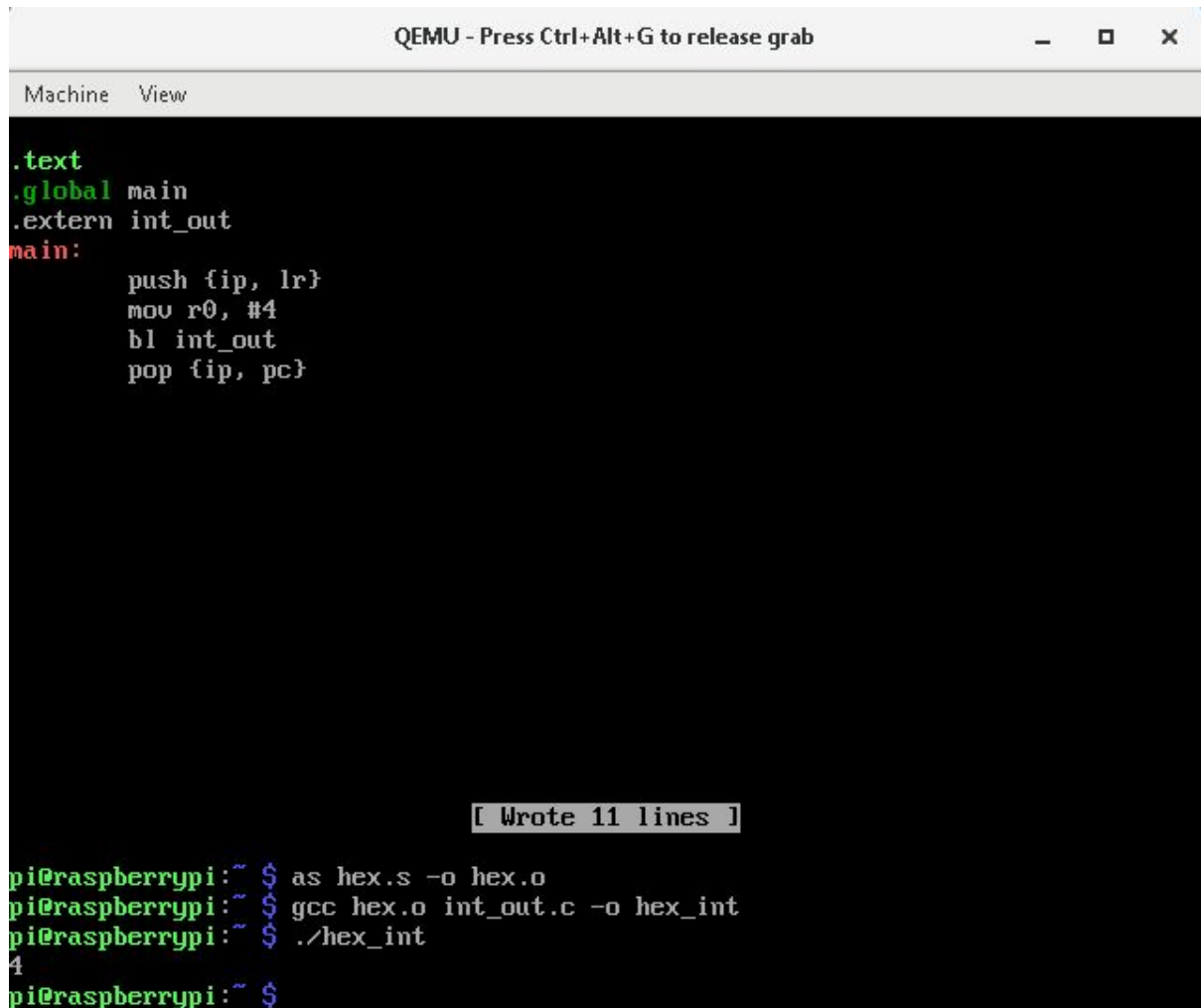
```
pi@raspberrypi:~ $ ./lab2
```

 followed by a new line. A status bar at the bottom of the terminal area says "[Read 11 lines]".

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 prints 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
```

A screenshot of a QEMU terminal window. The window title is "QEMU - Press Ctrl+Alt+G to release grab". The terminal shows assembly code for a main function. The code starts with ".text", ".global main", and ".extern int_out". The main function is defined as "main:" and contains four instructions: "push {ip, lr}", "mov r0, #4", "bl int_out", and "pop {ip, pc}". Below the code, there is a status bar that says "[Wrote 11 lines]". At the bottom of the terminal, there are shell commands: "pi@raspberrypi:~ \$ as hex.s -o hex.o", "pi@raspberrypi:~ \$ gcc hex.o int_out.c -o hex_int", "pi@raspberrypi:~ \$./hex_int", and "4". The prompt "pi@raspberrypi:~ \$" is shown at the bottom.

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

[Wrote 11 lines]

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

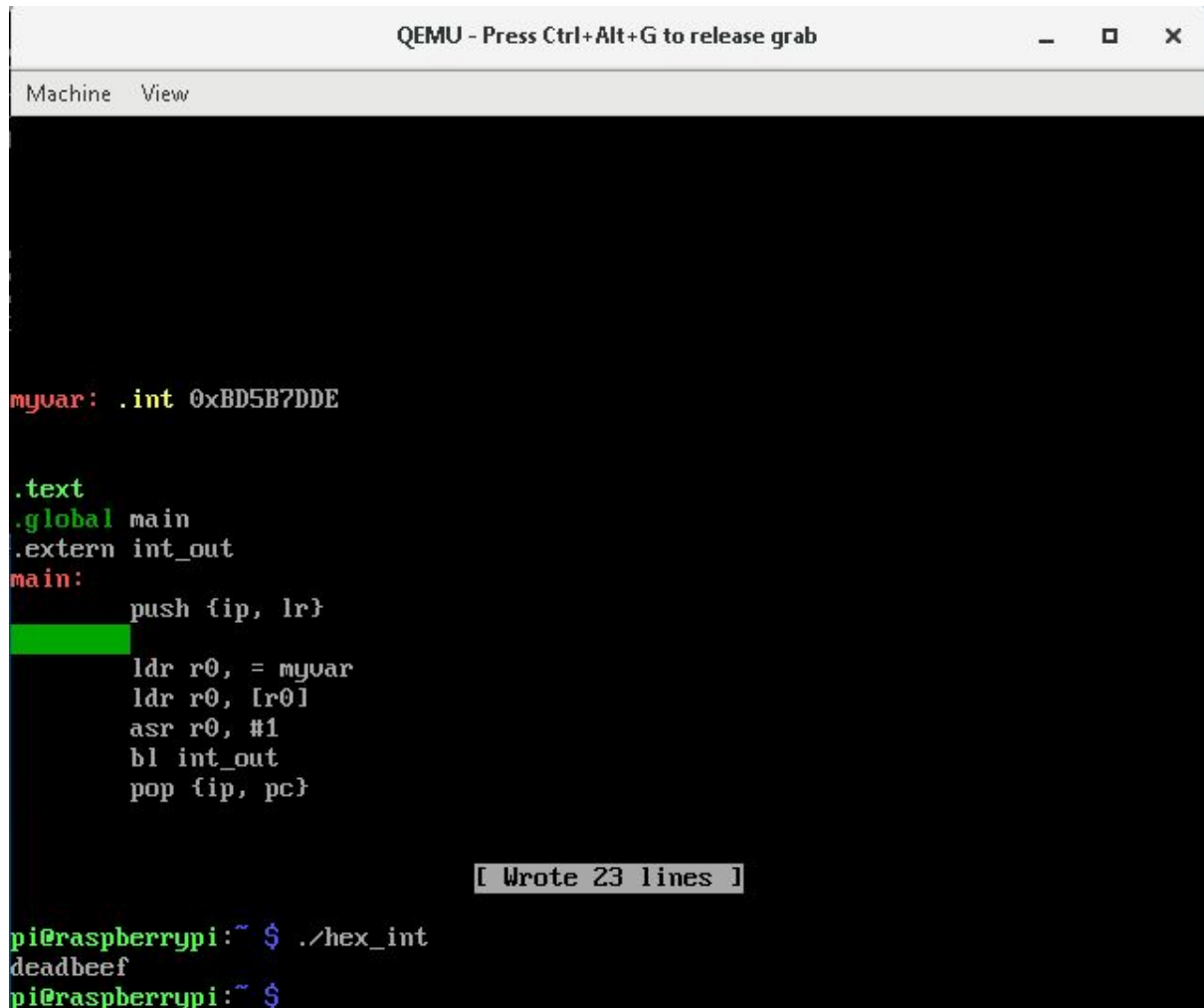
Screenshot of assembly program with external int_out

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`



The screenshot shows a QEMU terminal window with the title "QEMU - Press Ctrl+Alt+G to release grab". The terminal displays assembly code for a program named `hex_int`. The code defines a variable `myvar` with the value `0xBD5B7DDE`, then loads this value into register `r0`, shifts it right by one bit (`asr r0, #1`), and calls the `int_out` function to print the result. The output of the program is `deadbeef`. A status bar at the bottom of the terminal indicates "[Wrote 23 lines]".

```
Machine View

myvar: .int 0xBD5B7DDE

.text
.global main
.extern int_out
main:
    push {ip, lr}
    ldr r0, = myvar
    ldr r0, [r0]
    asr r0, #1
    bl int_out
    pop {ip, pc}

[ Wrote 23 lines ]

pi@raspberrypi:~ $ ./hex_int
deadbeef
pi@raspberrypi:~ $
```

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'.

```
QEMU - Press Ctrl+Alt+G to release grab
Machine View

.data
myvar: .int 0xBD5B7DDE
string: .asciz "%x"

.text
.global main
.extern printf
main:
    push {ip, lr}
    ldr r0, = string
    ldr r1, = myvar
    ldr r1, [r1]
    asr r1, #1
    bl printf
    pop {ip, pc}

[ Wrote 24 lines ]

pi@raspberrypi:~ $ as hex.s -o hex.o
pi@raspberrypi:~ $ gcc hex.o -o hex
^[[Api@raspberrypi:~ $ ./hex
deadbeefpi@raspberrypi:~ $
```

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.