Datateknik
Lab 2: ARM Assembly Language

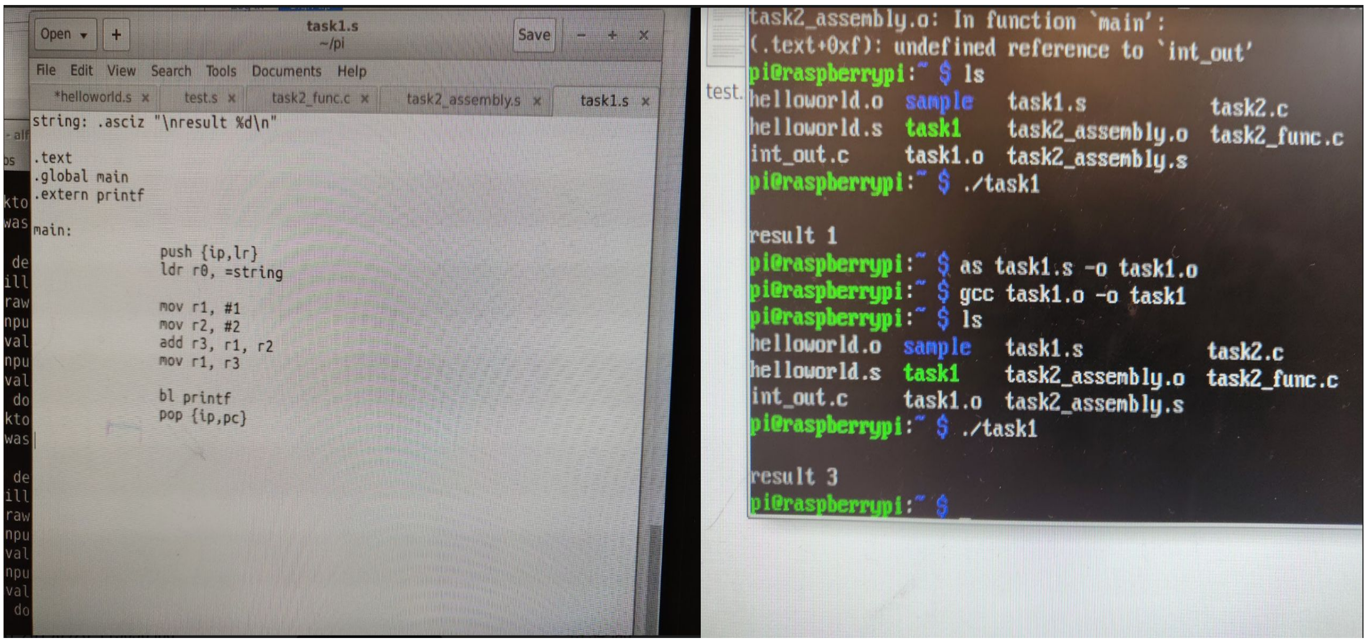
Alfred Elfving
Fahad Rejabo

Materials

• ThinLinc client on windows PC
• Raspbian kernel for qemu
• Disk image with raspbian

Task 1: Basic Setup

The first task is about to write an assembly program which loads two immediate integer values into the registers, adds them, and then calls the external function printf to display the result.



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

.text
.global main
.extern printf

main:
                push {ip,lr}
                ldr r0, =string

                mov r1, #1
                mov r2, #2
                add r3, r1, r2
                mov r1, r3

                bl printf
                pop {ip,pc}
```

```
task2_assembly.o: In function `main':
(.text+0xf): undefined reference to `int_out'
pi@raspberrypi:~ $ ls
helloworld.o  sample    task1.s           task2.c
helloworld.s  task1     task2_assembly.o  task2_func.c
int_out.c     task1.o   task2_assembly.s
pi@raspberrypi:~ $ ./task1

result 1
pi@raspberrypi:~ $ as task1.s -o task1.o
pi@raspberrypi:~ $ gcc task1.o -o task1
pi@raspberrypi:~ $ ls
helloworld.o  sample    task1.s           task2.c
helloworld.s  task1     task2_assembly.o  task2_func.c
int_out.c     task1.o   task2_assembly.s
pi@raspberrypi:~ $ ./task1

result 3
pi@raspberrypi:~ $
```

Screenshot of the code and the result

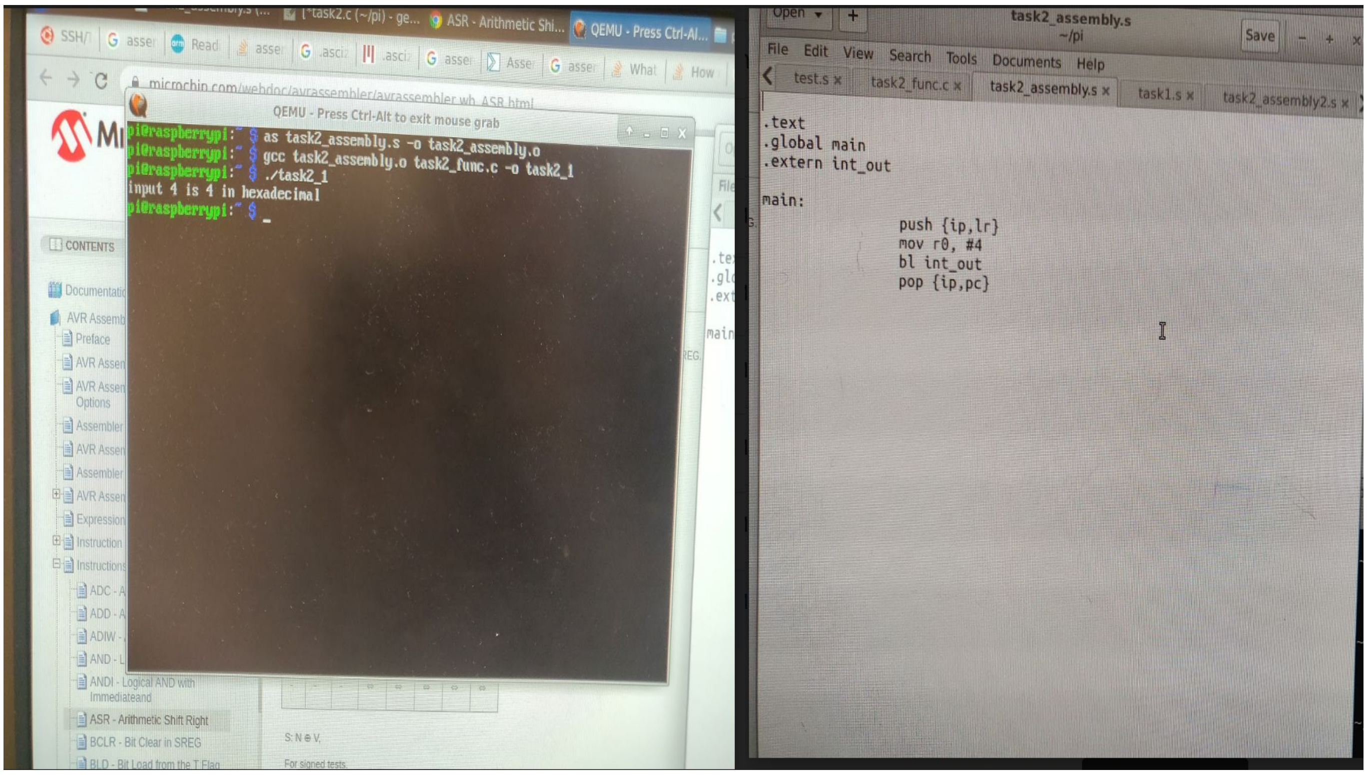
As we see in the code we start with to push return address and the register, after that we declare the address of the string into r0. We insert 1 and 2 to register r1 and r2 and also add and store them into r3, which thereafter, we store the sum back to register 1. In the end we print the string and return the address into pc, the printf function will print what is stored in register 1. After compiling and running the program we get “result 3”, as expected, the sum of 2 and 1 which register 1 and register 2 had in the beginning.

Task 2: Shifting Integers

Part 1 - printing hex value 4 with int_out.

The second task is about to write 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. And after that we have to load an integer 4 into a register r0 and then prints it by calling the external function int_out.

We start with creating a C function *int-out* that takes an integer argument and pints the value in hexadecimal. After that we create the assembly program which loads integer 4 into a register r0 and then prints it by calling the external function *int_out*, as we see in the screenshot. We start in the assembly program with pushing return address and register, after that inserting 4 to register r0, and printing by calling on *int_out* and last return address into pc. After compiling and running the program we get “input 4 is 4 in hexadecimal”.



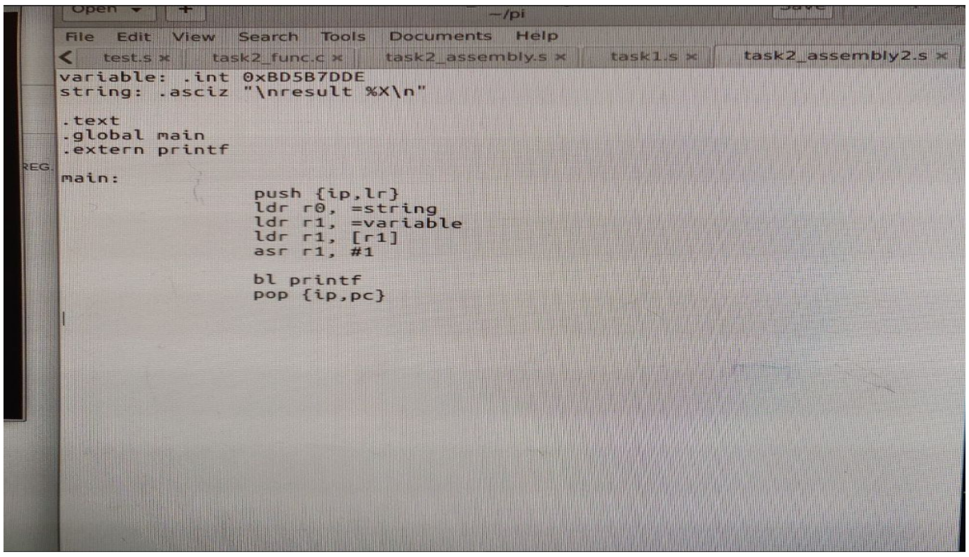
```
pi@raspberrypi:~ $ as task2_assembly.s -o task2_assembly.o
pi@raspberrypi:~ $ gcc task2_assembly.o task2_func.c -o task2_1
pi@raspberrypi:~ $ ./task2_1
input 4 is 4 in hexadecimal
pi@raspberrypi:~ $ _
```

```
.text
.global main
.extern int_out

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

Screenshot of the code and the result

Part 2 - bit shifting hex value and printing with printf.
The second part of the task we had to redesign the assembly code to work as previously but instead use the *printf* function instead of our *int_out* function. We also changed the variable from the value 4 to 0xBD5B7DDE instead. On top of doing this, we also bit shifted this value once to the right. The last part of the code will in the end use the function *printf* to print the bit shifted value of the hexadecimal value. After compiling with gcc and running the program, it printed out the word "*DEADBEEF*".



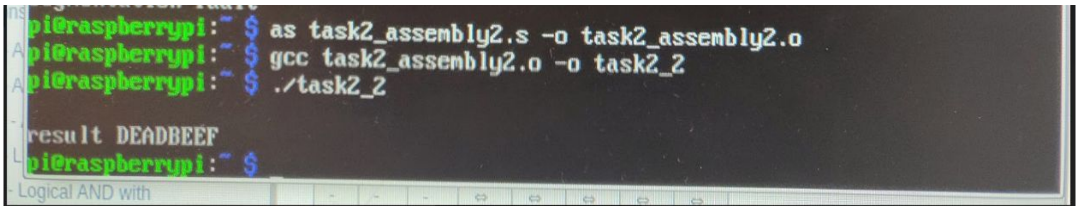
```
variable: .int 0xBD5B7DDE
string: .asciz "\nresult %X\n"

.text
.global main
.extern printf

main:
                push {ip,lr}
                ldr r0, =string
                ldr r1, =variable
                ldr r1, [r1]
                asr r1, #1

                bl printf
                pop {ip,pc}
```

The assembly code for part 2 of the task



```
pi@raspberrypi:~ $ as task2_assembly2.s -o task2_assembly2.o
pi@raspberrypi:~ $ gcc task2_assembly2.o -o task2_2
pi@raspberrypi:~ $ ./task2_2

result DEADBEEF
pi@raspberrypi:~ $
```

The commands we ran to compile and run the part 2 of the task

helloworld.s file:

```
.data
string: .asciz "\nHello World!\n"


.text
.global main
.extern printf

main:
        push {ip,lr}
        ldr r0, =string
        bl printf
        pop{ip,pc}
```

task1.s file:

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

.text
.global main
.extern printf

main:
                push {ip,lr}
                ldr r0, =string

                mov r1, #1
                mov r2, #2
                add r3, r1, r2
                mov r1, r3

                bl printf
                pop {ip,pc}
```

task2_assembly.s file:

```
.text
.global main
.extern int_out

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

task2_assembly2.s file:

```
variable: .int 0xBD5B7DDE
string: .asciz "\nresult %X\n"

.text
.global main
.extern printf

main:
                push {ip,lr}
                ldr r0, =string
                ldr r1, =variable
                ldr r1, [r1]
                asr r1, #1

                bl printf
                pop {ip,pc}
```

task2_printFunc file:

```
#include <stdio.h>

int int_out(unsigned int input){

        printf("input %d is %X in hexadecimal\n", input, input);
}
```

task2.c file:

```
#include <stdio.h>

int int_out(unsigned int input){

        printf("input %d is %X in hexadecimal\n", input, input);
}

int main(){
        int a = 15;
        int b = 16;
        int c = 30;

        int_out(a);
        int_out(b);
        int_out(c);
        return 0;
}
```