Lab 2

Isa Ertunga
Hannes Halm
Alexander Benteby

Task 1 (Basic Setup)

This task consisted of loading an integer into a register in assembly.

The example code shows a way to print data. In our case we needed to modify the string to include an int variable.

We chose to add 5 and 6. We load 5 into register2 and 6 into register3 and add them to register1. We call bl printf from register register 0 to make sure the string includes r1 in the print.

The code was written outside of QEMU but imported using *scp −P 5022 pi@127 . 0 . 0 . 1 : / home/ pi /.*
We compiled our .s file into an object file and then used gcc to make an executable.
When we tested our program for boundaries by testing the size of our integers. We noticed that we work with a 32 bit system and when adding values to a sum larger than 32 bit, the excess size start over from zero.

```
.syntax unified
.global main
main:
        push {ip, lr}

      mov r2, #5
      mov r3, #6
      add r1, r2, r3

      ldr r0, =string


        bl printf

        pop {ip, pc}


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

Task 2 (Shifting integers)

The various “sub-tasks” of this task will be answered chronologically.

(1) First off we were ordered to write a C function (int_out) which accepts an integer as parameter and outputs the integer in hexadecimal notation. We used the built in C method of denoting “%x” inside the printf function. We implemented this function in a C file together with a main function to make sure that it worked.

Implementation:

```c
#include <stdio.h>

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

int main() {
      int_out(100);
      return 0;
      }
```

(2) Next we wrote an assembly program that loads the immediate value 4 and prints it out using the previous int_out function.

Implementation in assembly:

```
.syntax unified
.global main
.extern int_out

main:

      push {ip, lr}

      mov r0, #4
      bl int_out

      pop {ip, pc}
```

It is important to note the line “.extern int_out” which includes the C function in the program. The be able to compile these two programs together, the C main function was needed to be removed. We compiled the program by translating them both into object files and them linking them together with GCC.

Implementation:

```
.syntax unified
.global main

main:

      push {ip, lr}

      @ Load into register 1
      ldr r1,=0xBD5B7DDE

      @ Load immediate value into r3, this is the offset for the bitshift
      mov r3, #1

      @ Perform the arithmetic right shift
      asr r1, r2, r3

      ldr r0, =string

      @ Print out the answer.
      bl printf

      pop {ip, pc}

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

The asr instruction is important, as it is responsible for actually performing the shift. We can choose how many shifts it performs with the offset. This being “arithmetic” right shift, the sign is not changed. Note that the program is modified to use the printf function as ordered by the task.

Task 3

For this task only the first three points were completed, since we had a problem with linking the programs and accessing the assembly function from C.

The C function for XOR was implemented with standard library implementation i.e a^b.

```
#include <stdio.h>

int XORfunc(int a, int b) {
            int result;
            result = a^b;
            printf("result: %d\n", result);
            return result;
      }

int main() {
      XORfunc(30,35);
      return 0;
      }
```

To test the function we calculated the result of a bitwise XOR for integers by hand and then tested the function the results were equal.

The XOR bit-wise in assembly was implemented with the operation EOR which gives the desired result.

```
.syntax unified
.global main

main:
      push {ip, lr}
      mov r2, #1
      mov r3, #0
      EOR r1, r2, r3

      ldr r0, =string
      bl printf
      pop {ip, pc}


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

For the last point which we couldn’t complete we tried linking the programs and re-writing the assembly implementation as a function so that it could be accessed from the C-program. When we tried to compile it said that we had multiple definitions of main, which we didn’t. We searched the internet for solutions and tried changing the code and tried compiling with different tags but at the end we didn’t find a solution.