# Datorteknik för civilingenjörer, HT18, DT509G
# Lab 2: ARM Assembly Language

*Karl Eriksson & Simon Johansson*
*2018-09-28*

## Task 1: Basic setup

```
.data
string: .asciz "\n%d + %d = %d \n"
.text
.global main
.extern printf
main:
push {ip, lr}
mov r1, #2
mov r2, #3
add r3, r1, r2
ldr r0, =string
bl printf
pop {ip, pc}
```

```
pi@raspberrypi:~/Documents/assembler $ as task1.s -o task1.o
pi@raspberrypi:~/Documents/assembler $ gcc task1.o -o task1
pi@raspberrypi:~/Documents/assembler $ ./task1

2 + 3 = 5
```

The program stores the immediate integer values 2 and 3 into register r1 and r2 respectively. Then the values in r1 and r2 are added together and the result is stored in register r3. Then the address to “string” is stored in register r0. The external function printf then reads from register r0 to r3 and outputs the text.

## Task 2: Shifting integers

```
.global main
.extern int_out

main:
push {ip, lr}
mov r0, #15
bl int_out
pop {ip, pc}
```

The external function “int_out” is declared. Then the immediate integer value 15 is stored in register r0. Then the program jumps to “int_out”, which is the compiled function created in the c-file. It will read r0 as an integer input.

```
#include <stdio.h>

void int_out(int input){

printf("\n%#010x\n", input);

}
```

```
pi@raspberrypi:~/Documents/assembler $ gcc task2_as.s task2.c -o task2
pi@raspberrypi:~/Documents/assembler $ ./task2

0x0000000f
```

Here is the c-file for the “int_out” function. The function takes an integer as input and prints it out in hexadecimal form. The decimal 15 is 0xf in hexadecimal form.

```
.data
number: .int 0xBD5B7DDe
message: .asciz "\n%#010x\n"
.text
.global main
.extern printf

main:
push {ip, lr}
ldr r1, =number
ldr r1, [r1]
asr r1, #1
ldr r0, =message
bl printf
pop {ip, pc}
```

The address of "number" is loaded into r1 then it follows the address and stores the value in r1. The value in r1 is bit shifted by one bit to the right (divided by 2). As in task1 the "message" address is stored in r0 and the printf procedure will read from r0 and r1. The output will be hexadecimal as specified in "message".

```
pi@raspberrypi:~/Documents/assembler $ gcc task2_as.s -o task2_printf
pi@raspberrypi:~/Documents/assembler $ ./task2_printf

0xdeadbeef
```

## Task 3: Assembly in C

```
.text
.global main
.extern xor

main:
push {ip, lr}
mov r0, #0b01110
mov r1, #0b10110
bl xor
pop {ip, pc}
```

The binary digit 01110 and 10110 is stored in register r0 and r1 respectively. The program the jumps to the procedure external xor which is our c-code function in compiled form. It takes two arguments, and it will read them from r0 and r1.

```
void xor(int inA, int inB){
int a = ~(inA & inB);
int b = (inA | inB);
int c = (a & b);
printf("\n%x xor %x = %x\n", inA, inB, c);

}
```

The function preforms bitwise NAND and bitwise OR then it ANDs them together. In other words, it will only give a 1 if one of the arguments is one but not both. It then presents it as hexadecimal numbers.

```
.text
.global axor

axor:
eors r0, r0, r1
mov pc,lr
```

```
uments/assembler $ gcc task3_as.s task3_c.c -o task3_part1
uments/assembler $ ./task3_part1
```

The last part of the task was to do it the other way around. The procedure

axor is written in assembly which performs xor on the two input arguments read from r0 and r1. It then stores the result in r0 to give it back to the c-function.

```c
#include <stdio.h>
extern int axor(int a, int b);

int main(void)
{
    int x = axor(0xe, 0x16);
    printf("\n%x\n", x);
    return 0;
}
```

First the external function “axor” with two inputs is declared. The main function calls the external function “axor” and gets the result back, which was written into register r0. Finally, it displays it as a hexadecimal number.

```
pi@raspberrypi:~/Documents/assembler $ ./task3_part2

18
```