Computer Lab 2

ARM Assembly Language

Computer Architectures for MSc in Engineering

HT19, DT509G

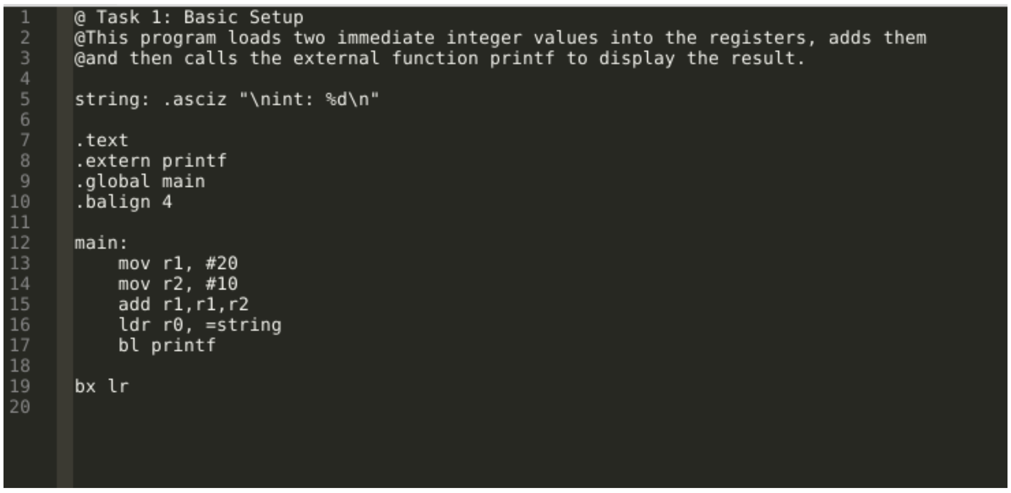
Abshir Abdulrahman & Fria Khorshid

2019-10-10

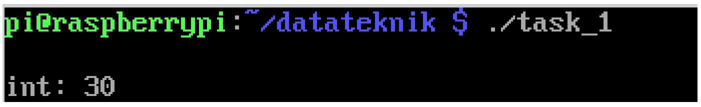
Task 1: Basic Setup

The main idea of this task is to make a program in assembly. This program should take two integer values, load them into two different registers and then add them together to get the sum. After the sum is obtained, the result should be displayed by using the external function printf.

We load integer 20 in register 1 and integer 10 in register 2, add them together by using "add" keyword and store them back in register 1. After that we load a prewritten string into r0 which we use to call printf with. Printf then goes into the next register, r1, and prints the value stored there which is 30 in our case.



```
@ Task 1: Basic Setup
@This program loads two immediate integer values into the registers, adds them
@and then calls the external function printf to display the result.

string: .asciz "\nint: %d\n"

.text
.extern printf
.global main
.balign 4

main:
    mov r1, #20
    mov r2, #10
    add r1,r1,r2
    ldr r0, =string
    bl printf

bx lr
```



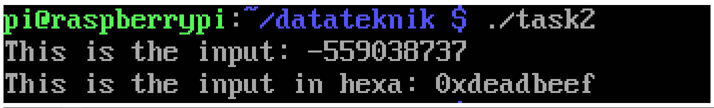
```
pi@raspberrypi:~/datateknik $ ./task_1

int: 30
```

Task 2: Shifting Integers

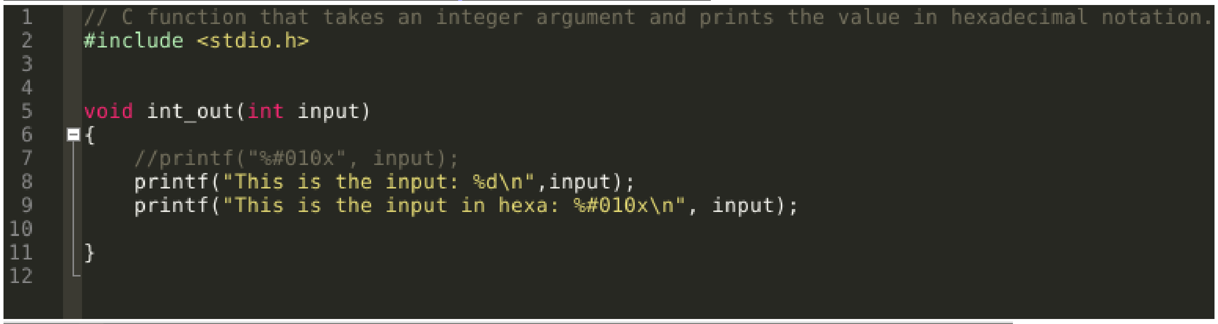
The objective of this task was to write our own function that takes in a hexadecimal value and shifts the value by 1 to the right. And also outputs the new, shifted, value.

First we made a C function that takes in an integer and converts it to hexadecimal before outputting it. Then we tested this function with a simple main function. After that we continued by writing an assembly program that loads an integer value into a register and calls our external function. The next step was to load a hexadecimal value into a register and call our function but bit shifting the value before outputting it. Finally we modified our program so it only uses printf and not our own function.

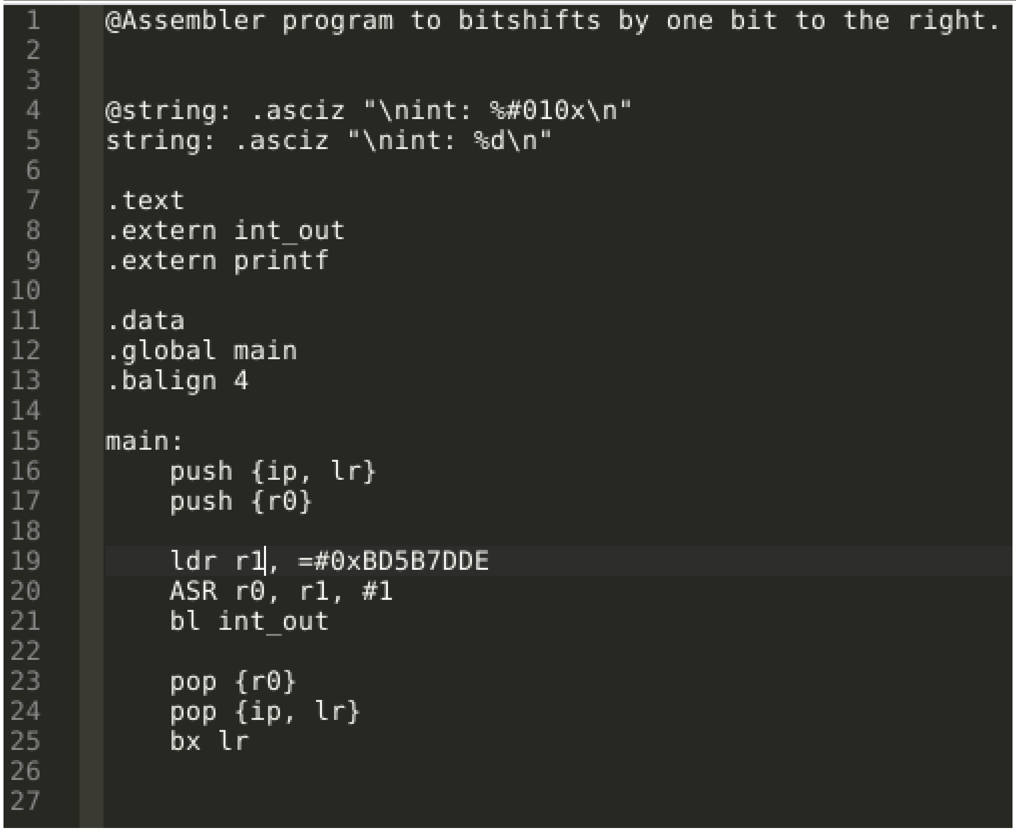
These pictures below shows the Result, C function, Assembly code. This is the final version of the task:



```
pi@raspberrypi:~/datateknik $ ./task2
This is the input: -559038737
This is the input in hexa: 0xdeadbeef
```



```c
// C function that takes an integer argument and prints the value in hexadecimal notation.
#include <stdio.h>


void int_out(int input)
{
    //printf("%#010x", input);
    printf("This is the input: %d\n",input);
    printf("This is the input in hexa: %#010x\n", input);

}

```

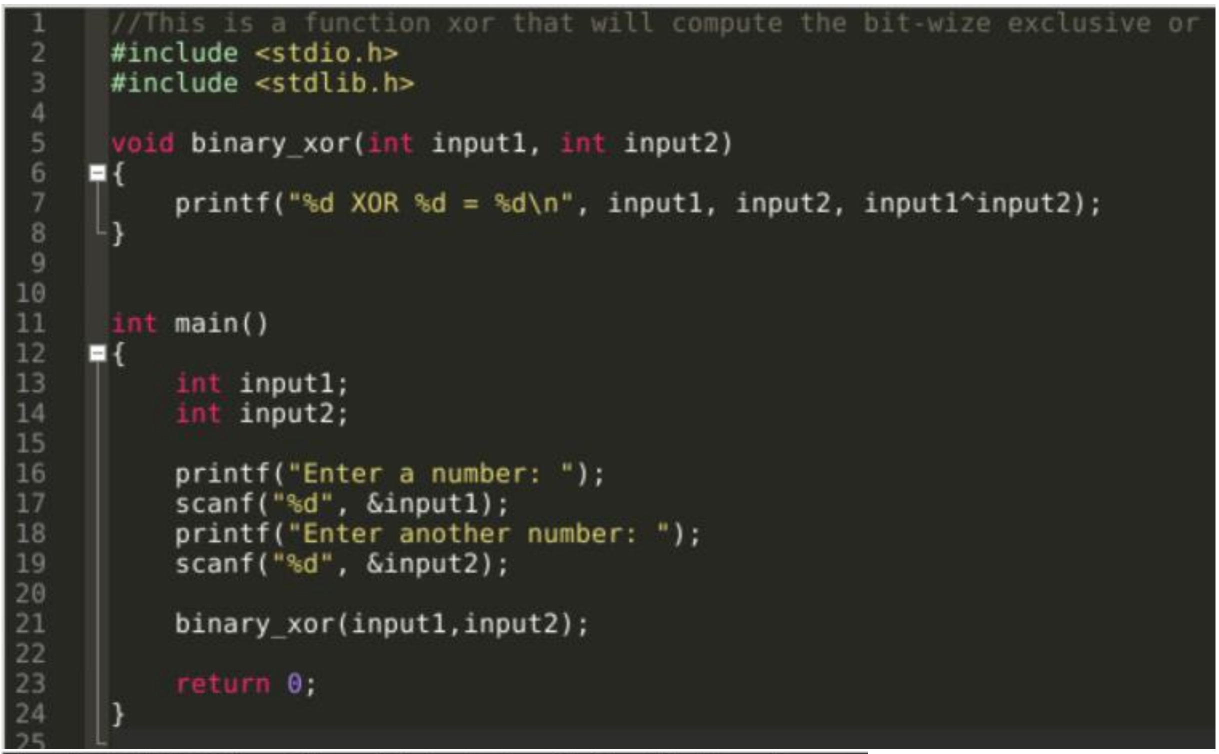


```
@Assembler program to bitshifts by one bit to the right.


@string: .asciz "\nint: %#010x\n"
string: .asciz "\nint: %d\n"

.text
.extern int_out
.extern printf

.data
.global main
.balign 4

main:
    push {ip, lr}
    push {r0}

    ldr r1, =#0xBD5B7DDE
    ASR r0, r1, #1
    bl int_out

    pop {r0}
    pop {ip, lr}
    bx lr


```

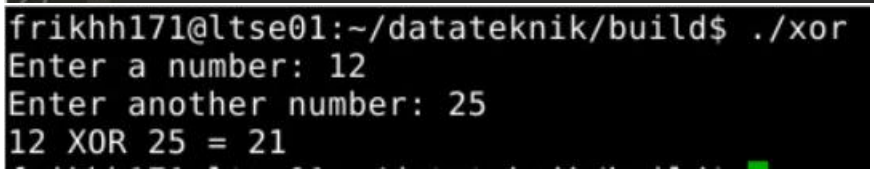
Task 3: Assembly in C

The objective of this task was to write a function in C that computes the bitwise XOR between two integers. Then we are to do the same thing but in assembly and compare our results with the C function.

We started by writing a C function that takes in two integer inputs and outputs the XOR value. We tested this function by trying it out with a simple main function. The result gets printed out as you can see below.



```c
//This is a function xor that will compute the bit-wize exclusive or
#include <stdio.h>
#include <stdlib.h>

void binary_xor(int input1, int input2)
{
    printf("%d XOR %d = %d\n", input1, input2, input1^input2);
}


int main()
{
    int input1;
    int input2;

    printf("Enter a number: ");
    scanf("%d", &input1);
    printf("Enter another number: ");
    scanf("%d", &input2);

    binary_xor(input1,input2);

    return 0;
}
```



```
frikhh171@ltse01:~/datateknik/build$ ./xor
Enter a number: 12
Enter another number: 25
12 XOR 25 = 21
```