

Viktor Rask
Amanda Håkansson
DT509G



Lab 2

Computer Architectures for MSc in Engineering

Innehåll

Introduction.....	2
Task 1	3
Task 2	4
Task 3	6

Viktor Rask
Amanda Håkansson
DT509G

Introduction

The purpose of this lab was to learn how to program low-level assembly language and how to interface that with c.

Task 1

The first task of lab 2 was to simply write an assembly program which added two integers and printed the sum. This was done with the following code:

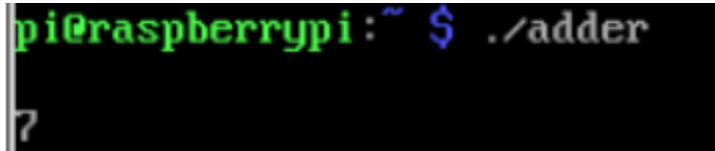
Code

```
1
2  .data
3  .balign 4
4      string: .asciz "\n%d\n"
5  .text
6  .extern printf
7
8  .global main
9  main:
10     push {ip, lr}
11
12     mov r0, #1          @ r0 <-- 1
13     mov r1, #6          @ r1 <-- 6
14     add r1, r0, r1      @ r1 = r0+r1
15     ldr r0, =string     @ load string into r0 to print value of r1
16     bl printf           @ branch to printf
17
18     pop {ip, pc}
19
20
```

At the top of the code 4 bytes is defined with `.balign 4` and initialized as a string with an integer input, this string is given the label “string”. The “`.extern`” command was used to be able to use the c function “`printf()`” externally. Push was used to push the return address “`lr`” and a dummy register “`ip`” to the stack. Pop was used to pop the return address into the program counter. The next block of code moves 1 and 6 into register `r0` and `r6`, then with the add command put $1 + 6$ into register `r1`. To be able to use the print function the string was loaded into register `r0` because the string requires an input it will take the value stored at register `r1` and print it. Lastly to execute the `printf` function the command “`bl`” was used to branch to `printf`.

Method of verification

To verify that the code worked the output was observed to see if the two numbers were correctly added or not. The correct result 7 from the $1 + 6$ can be seen in the following picture:



Task 2

The second task of the lab had several steps but the point of it was to learn how to use c functions in assembly and how to bit shift an integer.

Code

Assembly shift code:

```
1  .data
2  .balign 4
3      value: .word 0xBD5B7DDE
4  .balign 4
5      string: .asciz "%#010X\n" @"%d\n"
6  .text
7  .extern printf @int_out
8  .global main
9
10 main:
11     push {ip, lr}
12
13     ldr r1, =value           @ r0 <-- value
14     ldr r1, [r1]             @ r1 <-- *r1
15     asr r1, #1               @ shift r0 1 steg till höger
16     ldr r0, =string          @ kommer printa r1
17     bl printf @int_out       @ branch to printf
18
19     pop {ip, pc}
20
```

This is the code for the assembly program which shifts an integer one bit to the right, if the int was 010 the bit shift would make it 001. Same as in task 1, 4 bytes are defined for integers

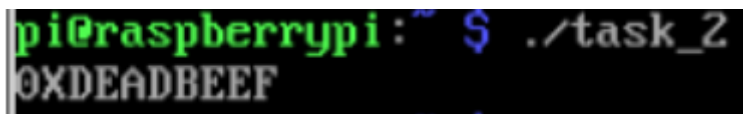
and the integer 0xBD5B7DDE is initialized to the label “value” at row 3. When using the external c function printf a string similar to task 1 is also initialized. In the main block of code at row 13 a pointer of the integer in the label value is loaded into r1. Then the pointer is dereferenced, this is done because we want the actual value not the address of a pointer. After that the command asr is used to arithmetic shift the integer one bit to the right. Then r0 is loaded with a string which accepts the value of r1 at row 16. If the printf function is used the string must be loaded into r0 to work but if the int_out function is used that step is not needed.

C function:

```
1  #include <stdio.h>
2  void int_out(int arg){
3      printf("%#010X\n", arg);
4  }
5
```

This small block of code is a c function which simply prints the input argument in hexadecimal form.

Method of verification

A terminal window on a Raspberry Pi showing the command './task_2' being executed. The output is '0XDEADBEEF' in green text on a black background.

This was the answer in hexadecimal form when running the bit shifting code with the integer 0xBD5B7DDE. To check if the shift was successful the input integer 0xBD5B7DDE and the output 0XDEADBEEF was converted to binary and compared. In the table the output is the same as input but shifted to the right with the first 1 from the input copied to be in first place on the output.

	Input	Output
Hex	0xBD5B7DDE	0xDEADBEEF
Binary	10111101010110111110111011110	11011110101011011011111011101111

Task 3

The purpose of the third task was to create a c program that uses a function which was written in assembly.

Code

```
1  #include <stdio.h>
2  extern int axor(int a, int b);
3
4  int xor( int a, int b){
5      if( a == b){
6          return 0;
7      }
8      else{
9          return 1;
10     }
11 }
12 int main(){
13     int a = 1, b = 0, c, ac;
14
15     c = xor(a, a); // use c function
16     ac = axor(a, a); // use assembly function
17     printf("1 1 c: %d ac: %d\n", c, ac); // print value of c func and asm func
18
19     c = xor(a, b);
20     ac = axor(a, b);
21     printf("1 0 c: %d ac: %d\n", c, ac);
22
23     c = xor(b, a);
24     ac = axor(b, a);
25     printf("0 1 c: %d ac: %d\n", c, ac);
26
27     c = xor(b, b);
28     ac = axor(b, b);
29     printf("0 0 c: %d ac: %d\n", c, ac);
30     return 0;
31 }
32 }
```

The function xor(int, int) in row 4- 11 returns 0 if a and b is the same and 1 otherwise. It is used to compare the values from the axor function that is written in assembly.

```
1  .data
2  .globl axor
3  .text
4
5  axor:
6      @push {ip, lr}
7
8      cmp r0, r1      @ if r0 == r1
9      bne false      @ if false branch to false
10     mov r0, #0      @ r0 <-- 0 because we want to return 0
11     bx lr          @ branch to return address
12
13 false:
14     mov r0, #1      @ r0 <-- 1 because we want to return 1
15     bx lr          @ branch to return address
16
17     @pop {ip, pc}
18
```

In row 2 the axor function is declared as global, so that it is possible to use it in the c-program. The first thing that is done in the function is comparing the values in registers r0 and r1, if they aren't equal the program branches to "false" and puts the value 1 in register r0, and then returns it. Otherwise the value 0 is put in to register r0, and then returned.

Method of verification

In the main() function all different combinations of the xor truth table is tested with both the c- and the assembly xor function and the result is printed.

a	b	xor
0	0	0
0	1	1
1	0	1
1	1	0

Viktor Rask
Amanda Håkansson
DT509G

```
pi@raspberrypi:~$ ./task_3
1 1 c: 0 ac: 0
1 0 c: 1 ac: 1
0 1 c: 1 ac: 1
0 0 c: 0 ac: 0
```