Lab 2: ARM Assembly Language

Marcus Nyström, Anton Söderlund
20180928

Task 1

To test we put the integer 2 into register 1 and integer 3 into register 2 and added them into register 1. Then we put the result into a string and used printf to print the result, as can be seen in Appendix 1.

Task 2

We wrote the function int_out in C to print the value in hexadecimal notation, see Appendix 2, and tested it with some numbers to verify that it worked.

We wrote an assembly program, see Appendix 3, which loads the immediate value of 4 into a register and then used the external function int_out, which we have compiled into an object file and linked that and the object file for our assembly program into an executable. We verified that it converted 4 to 0x4, we also tested with other, larger, numbers.

We modified our program to use printf directly instead of int_out, a simple change. In the same way we used printf in Task 1, but we print two values instead of one. printf gets the string to print from r0, the first value from r1 and the second value from r2.

Figure 1: printf output

We loaded the integer 0xBD5B7DDE instead of 4, using ldr, and then used asr to bit shift 1 bit right to get the integer 0xDEADBEEF, see Appendix 4. We tested the bit-shift manually on paper to verify correct output.

Task 3

We wrote the C function xor that computes the bit-wise exclusive or of two integers a, b as a^b. To verify that it works we also used a print_binary function[1] to print the values in binary format. We wrote a main function and tested the xor function with a couple of integers. See code in Appendix 5.

[1] https://stackoverflow.com/questions/6373093/6373240#comment60290917_6373240

We wrote an assembly version of the xor function called axor where we used the instruction eor to compute the exclusive-or of two integers stored in registry, see Appendix 6.

We implemented a main function in C using our axor function and tested it with a lot of different inputs and compared it to the output of our xor function, see code in Appendix 7.

Appendices: Assembly and C code

Appendix 1: Task 1 Assembly Code

```
.data
.balign 4
string: .asciz "\n Result: %d\n "/* Output string */
c: .word 0 /* Output value to printf */
return: .word 0

.text
.global main
.extern printf

main:
    ldr r1, address_return /* r1 <- &address_return  */
    str lr, [r1]                /* *r1 <- lr  */

    mov r1, #2             /* r1 <- 2  */
    mov r2, #3             /* r2 <-  */
    add r1, r1, r2         /* r1 <- r1 + r2 */
    ldr r2, =c             /* get address of c into r2 */
    str r1, [r2]           /* store r1 into c */
    ldr r0,=string         /* get address of string into r0 */
    ldr r1, [r2]                /* pass c into r1 */
    bl printf              /* print string and r1 as param */

    ldr lr, address_return /* lr <- &address_return */
    ldr lr, [lr]                /* lr <- *lr */
    bx lr                  /* return from main using lr */
address_return : .word return
```

Appendix 2: Task 2 int_out.c

```
#include <stdio.h>

void int_out(int a){
     printf("\n %d to %#x\n",a,a); // %#x value in hex startin w
0x
}
```

Appendix 3: Task 2 Assembly program calling external int_out

```
.data
.balign 4
return: .word 0

.text
.global main
.extern int_out

main:
    ldr r1, address_return /* r1 <- &address_return  */
    str lr, [r1]                /* *r1 <- lr  */

    mov r0, #4             /* r1 <- 2  */
    bl int_out             /* int_out prints hexadecimal */

    ldr lr, address_return /* lr <- &address_return */
    ldr lr, [lr]                /* lr <- *lr */
    bx lr                  /* return from main using lr */
address_return : .word return
```

Appendix 4: Task 2 0xBD5B7DDE

```
.data
.balign 4
string: .asciz "\n Decimal %d To hexadecimal %#x \n "/* Output
string */
c: .word 0 /* First output value to printf */
d: .word 0 /* Second output value to printf */
return: .word 0

.text
.global main
.extern printf

main:
    ldr r1, address_return /* r1 <- &address_return  */
    str lr, [r1]                /* *r1 <- lr  */

    ldr r1, =#0xBD5B7DDE   /* r1 <- 0xBD5B7DDE  */
    asr r1,r1,#1                /* shift 1 bit right */

    ldr r2, =c             /* get address of c into r2 */
    str r1, [r2]           /* store r1 into c */
    ldr r3, =d             /* get address of d into r3 */
    str r1, [r3]                /* store r1 into d */
    ldr r0,=string         /* get address of string into r0 */
    ldr r1, [r2]                /* pass c into r1 */
    ldr r2, [r3]                /* pass d into r2 */
    bl printf              /* print string and r1, r2 as params*/

    ldr lr, address_return /* lr <- &address_return */
    ldr lr, [lr]                /* lr <- *lr */
    bx lr                  /* return from main using lr */
address_return : .word return
```

Appendix 5: Task 3 xor C function

```
#include <stdio.h>
#include <stdlib.h>

int xor(int a, int b){
     return  a^b;
}

int print_binary(int N){
     int numbits = 32;
     while(--numbits >= 0) {
          printf("%c",(N & ((int)1 << numbits)) ?'1' : '0');
     }
     printf("\n");
}

int main() {
     int a = 5;
     int b = 10;
     printf("a = %d, b = %d\n",a,b);
     int c = xor(a, b);
     print_binary(a);
     print_binary(b);
     print_binary(c);

     printf("%d\n",c);
}
```

Appendix 6: Task 3 axor function in assembly

```
.text
.global axor
.type axor,%function
axor:
     .fnstart
     eor r0, r0,r1
     bx lr
     .fnend
```

Appendix 7: Task 3 main.c

```
#include <stdio.h>

extern int axor(int a, int b);

int main(int argc, char *argv[]) {
     printf("\n mainprint = %d\n",
                axor(atoi(argv[1]),atoi(argv[2])));
     return 0;
}
```