# EE2016: Experiment 7

Group:19


| Sparsh Gupta | Charan Srikkanth | Kaushik Iyer |
|---|---|---|
| EE23B117 | EE23B127 | EE23B135 |


October 01, 2024


**Abstract**

This experiment was our first look into programming with the ARM Assembly language (using Keil $\mu$Vision studio).


## 1 Objectives

The primary objectives of this experiment were to:

1. Getting familiar with the instruction set provided in ARM Assembly.
2. Understanding how to use Keil $\mu$Vision studio to build and debug the program.

## 2 Tasks

The following are the tasks given, the results obtained and our approach to solving it.

### 2.1 Task 1: Loading numbers (part 1)

#### 2.1.1 Assembly Code

```
    AREA Program, CODE, READONLY
    ENTRY
        MOV r0, #11 ;; Load decimal 11 into r0
Stop
        B Stop
    END
```

#### 2.1.2 Result

After running this program, r0 gets the value of 0x0000000B. The disassembler shows that this was done by simply using the MOV command itself.

### 2.2 Task 2: Loading numbers (part 2)

#### 2.2.1 Assembly Code

```
    AREA Program, CODE, READONLY
    ENTRY
        MOV r0, &FFFFFFFF ;; Load hex FFFFFFFF into r0
        ; Assembler smartly converts this to
        ; MVN r0, #0 ;; Which does the same thing :)
Stop
        B Stop
    END
```

### 2.2.2 Result

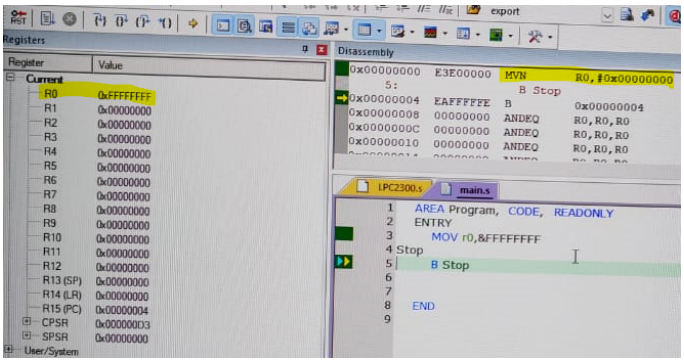


Figure 1: Output for task 2

After running this program, r0 gets the value of 0xFFFFFFFF. This might be surprising as there is no way this MOV command can be stored in a 32 bit instruction (since the number we are loading is simply too big). In this case the assembler was smart enough to notice this and instead use the MVN command to load this number. (0xFFFFFFFF is the negation of 0x00000000)

## 2.3 Task 3: Simple arithmetic

### 2.3.1 Assembly Code

```
    AREA Reset, CODE, READONLY
    ENTRY
        LDR r0, =7
        MUL r1, r0, r0
        LDR r2, =4
        MUL r1, r2, r1
        LDR r3, =3
        MUL r3, r0, r3
        ADD r1, r1, r3
stop
        B stop
    END
```

### 2.3.2 Result

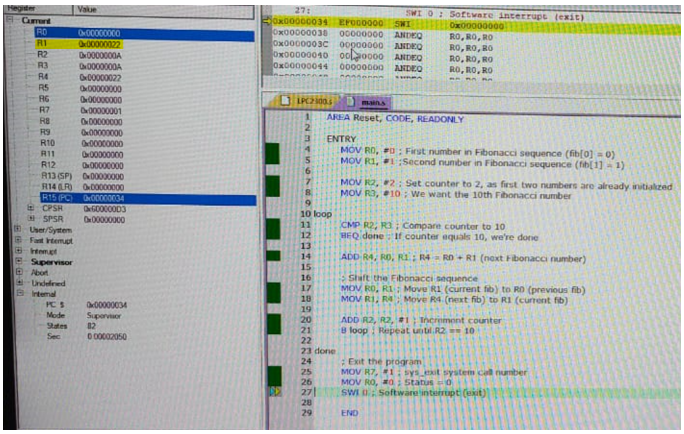


Figure 2: Output for task 3

The final value of r1 is 0x000000D9

1. After the line2, r1 = 0x00000031
2. After the line4, r1 = 0x000000C4
3. After the line7, r1 = 0x000000D9

## 2.4 Task 4: Fibonacci numbers

### 2.4.1 Pseudo Code

```
int fib0 = 0; // R0
int fib1 = 1; // R1

int iter = 2; // R2
int max_iter = 10; // R3

do { // loop
    if (iter == max_iter) break

    int tmp = fib0 + fib1; // R4
    fib0 = fib1;
    fib1 = tmp;

    ++iter;
} while(true)
```

### 2.4.2 Assembly Code

```
        AREA Reset, CODE, READONLY

        ENTRY
                MOV R0, #0 ; First number in Fibonacci sequence (fib[0] = 0)
                MOV R1, #1 ;Second number in Fibonacci sequence (fib[1] = 1)

                MOV R2, #2 ; Set counter to 2, as first two numbers are already initialized
                MOV R3, #10 ; We want the 10th Fibonacci number

loop
                CMP R2, R3 ; Compare counter to 10
                BEQ done ; If counter equals 10, we're done

                ADD R4, R0, R1 ; R4 = R0 + R1 (next Fibonacci number)

                ; Shift the Fibonacci sequence
                MOV R0, R1 ; Move R1 (current fib) to R0 (previous fib)
                MOV R1, R4 ; Move R4 (next fib) to R1 (current fib)

                ADD R2, R2, #1 ; Increment counter
                B loop ; Repeat until R2 == 10

done
                ; Exit the program
                MOV R0, #0 ; Status = 0

    END
```

### 2.4.3 Result

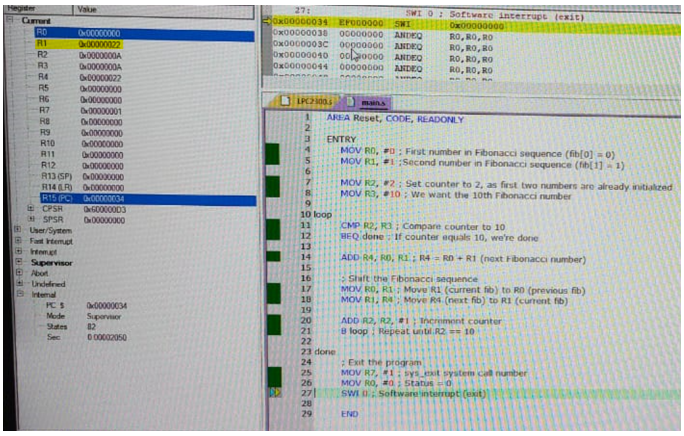


Figure 3: Output for task 4

The 10th fibonacci number comes out to be 0x00000022.

## 2.5 Task 5: Division

### 2.5.1 Pseudo Code

```
int* num_ptr = Num1; // R2
int num1 = *num_ptr; // R0
int* num_ptr = Num2; // R2
int num2 = *num_ptr; // R1
int quotient = 0; // R2

do { // Loop
    if (num2 == 0) goto Error1;
    if (num1 < num2) goto Result;

    ++quotient;
    num1 -= num2;
} while(true)

Error1:
    quotient = -1;

Result:
    int* remainder_ptr = Remainder; // R4
    *remainder_ptr = num1;
    int* quotient_ptr = Quotient; // R5
    *quotient_ptr = quotient;

    exit(0)
```

### 2.5.2 Assembly Code

```
    AREA Program, CODE, READONLY
    ENTRY
        LDR R2, =Num1
        LDR R0, [R2]  ;load the numbers into R0 and R1
```

```
        LDR R2, =Num2
        LDR R1, [R2]
        MOV R2, #0; assign and clear the quotient register
Loop
        CMP R1, #0 ;test division by 0
        BEQ Error1
        CMP R0, R1 ;is the dividend less than the divisor ?
        BLO Result ;if yes, then we are done
        ADD R2, R2, #1 ;add one to quotient
        SUB R0, R0, R1
        B Loop
Error1
        MOV R2, #0xFFFFFFFF ;error flag (-1)
Result
        LDR R4, =Remainder ;store the remainder and quotient
        STR R0, [R4]
        LDR R5, =Quotient
        STR R2, [R5]
        SWI &11
Num1 DCD 121
Num2 DCD 12
ALIGN
    AREA Data, DATA, Readwrite
Quotient DCD 0
Remainder DCD 0
    END
```

### 2.5.3 Result

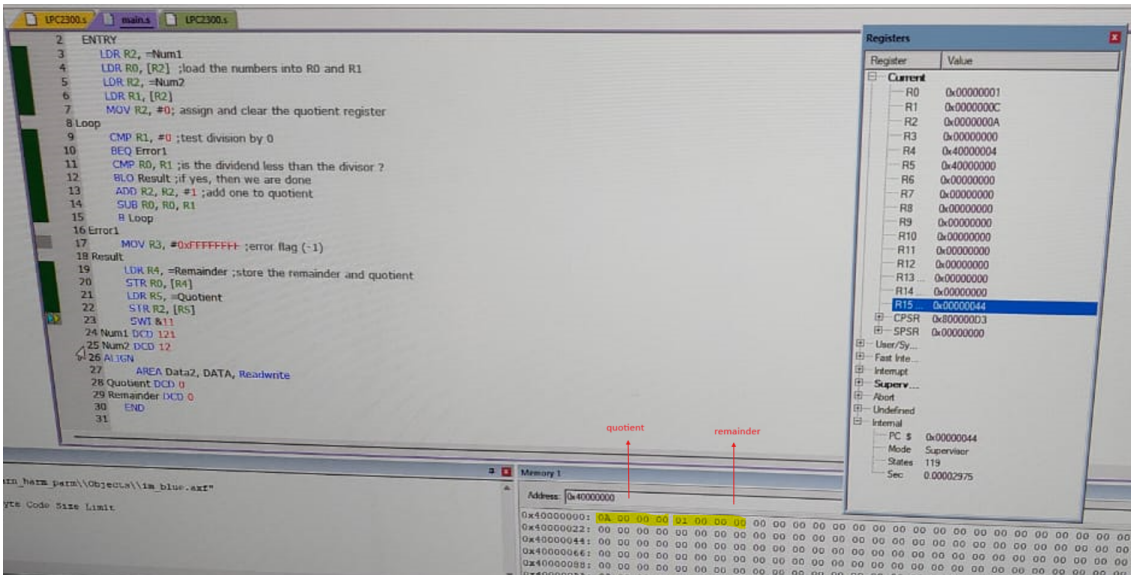


Figure 4: Output for task 5 (NOTE: It is in little endian)

When 121 is divided by 12, we get quotient 10 and remainder 1.

# 3 Comments

1. We made the mistake of including the LPC.s file while creating the project. This gave some random errors while we were trying to execute our code.

2. Getting used to the indentation rules took some time.