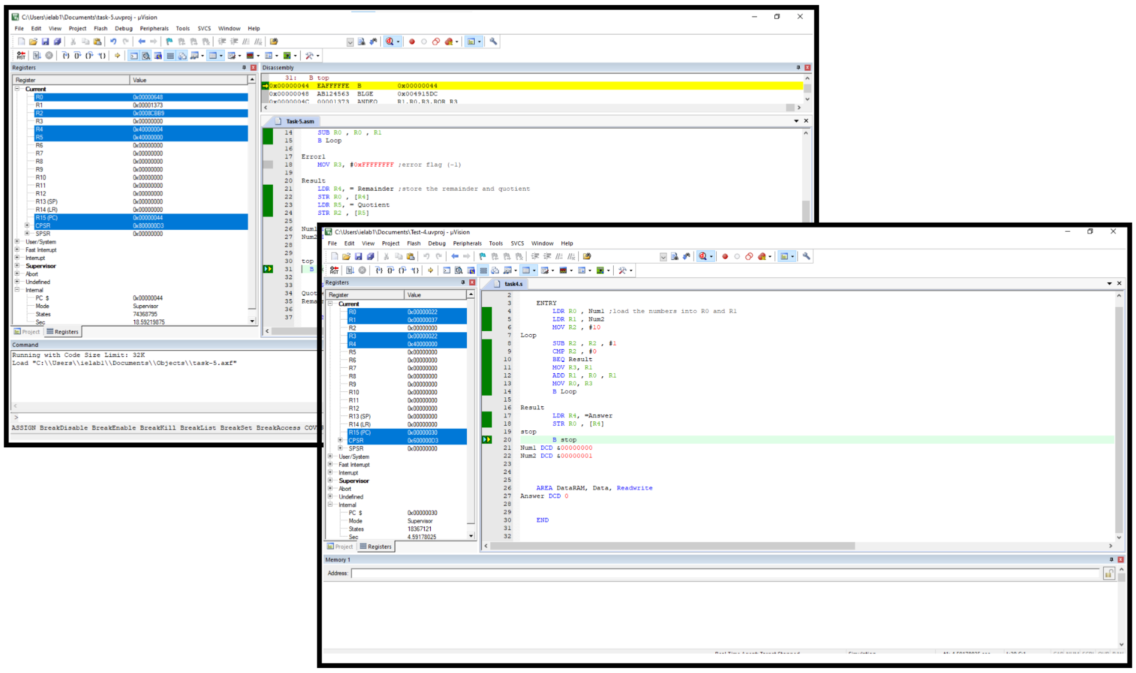

C:\Users\ielab1\Documents\task-5.uvproj - µVision

```
Register        Value
Current
  R0            0x00000648
  R1            0x00001373
  R2            0x0008CBB9
  R3            0x00000000
  R4            0x40000004
  R5            0x40000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x00000044
  CPSR          0x800000D3
  SPSR          0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
Internal
  PC $          0x00000044
  Mode          Supervisor
  States        74368795
  Sec           18.59219875
```

```
     31:     B top
0x00000044  EAFFFFFE  B        0x00000044
0x00000048  AB124563  BLGE     0x004915DC
0x0000004C  00001373  ANDEQ    R1,R0,R3,ROR R3
```

Task-5.asm

```
        SUB R0 , R0 , R1
        B Loop

Error1
        MOV R3, #0xFFFFFFFF ;error flag (-1)

Result
        LDR R4, = Remainder ;store the remainder and quotient
        STR R0 , [R4]
        LDR R5, = Quotient
        STR R2 , [R5]
```

```
Running with Code Size Limit: 32K
Load "C:\\Users\\ielab1\\Documents\\Objects\\task-5.axf"
```

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COV
```

C:\Users\ielab1\Documents\Test-4.uvproj - µVision

```
Register        Value
Current
  R0            0x00000022
  R1            0x00000037
  R2            0x00000000
  R3            0x00000022
  R4            0x40000000
  R5            0x00000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x00000000
  R14 (LR)      0x00000000
  R15 (PC)      0x00000030
  CPSR          0x600000D3
  SPSR          0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
Internal
  PC $          0x00000030
  Mode          Supervisor
  States        18367121
  Sec           4.59178025
```

task4.s

```

    ENTRY
        LDR R0 , Num1 ;load the numbers into R0 and R1
        LDR R1 , Num2
        MOV R2 , #10
Loop
        SUB R2 , R2 , #1
        CMP R2 , #0
        BEQ Result
        MOV R3, R1
        ADD R1 , R0 , R1
        MOV R0, R3
        B Loop

Result
        LDR R4, =Answer
        STR R0 , [R4]
stop
        B stop
Num1 DCD &00000000
Num2 DCD &00000001



    AREA DataRAM, Data, Readwrite
Answer DCD 0


    END
```

EE2016: Assignment-7

By Siva(EE23B151),
Bhavesh(EE23B017)
and Naveen(EE23B113)

Group 28

Date: 8/10/2024

Indian Institute of Technology Madras

Contents

1 Introduction

The objective of this experiment is to familiarize students with ARM assembly language, one of the most commonly used low-level programming languages for embedded systems. ARM (Advanced RISC Machine) is a family of Reduced Instruction Set Computing (RISC) architectures that are widely employed in microcontrollers and processors for a variety of applications, ranging from mobile phones to industrial automation.

1.1 ARM Architecture Overview

ARM processors are based on the RISC architecture, which is designed to reduce the number of cycles per instruction and increase efficiency. ARM is known for its:

- **Simple and small instruction set**: The ARM instruction set is streamlined, allowing for more efficient execution of instructions.
- **Energy efficiency**: ARM processors are highly energy-efficient, making them ideal for battery-powered devices.
- **Flexibility**: The ARM architecture is used in a wide range of devices, from low-power microcontrollers (MCUs) to high-performance processors in mobile phones and tablets.

In this experiment, we will focus on programming in **ARM assembly language**, which gives us direct control over the hardware and provides a deep understanding of the processor's internal workings.

1.2 ARM Assembly Programming

ARM assembly language is a low-level programming language where each line of code represents a specific machine-level instruction. Writing in assembly provides better control over hardware resources, but also requires a deep understanding of the processor's architecture.

Some basic elements of ARM assembly include:

- **32-bit Registers**: ARM uses a set of general-purpose registers (R0-R15) to store temporary data, as well as special-purpose registers like the Program Counter (PC) and Link Register (LR).
- **Instructions**: ARM instructions operate on data stored in registers. Common instructions include `MOV` (move data), `ADD` (addition), and `B` (branch to a different part of the program).
- **Directives**: These are instructions for the assembler rather than the processor. For example, `.global` is a directive used to declare global symbols.

1.3 Keil µVision IDE

To run and simulate ARM assembly language programs, we will be using the **Keil µVision IDE**. This integrated development environment is specifically designed for ARM-based microcontrollers and supports multiple ARM architectures. It provides an easy-to-use interface for developing, debugging, and testing assembly programs.

2 Tasks

2.1 Task 1: Running the given ARM code

Given ARM Code

```
    AREA Program, CODE, READONLY
    ENTRY
        MOV r0,#11
Stop
        B Stop
    END
```

Procedure

In this task, we observed the content of register `R0` during the execution of a simple ARM assembly program. A breakpoint was set at a specific instruction, and the program was executed step-by-step using the `F11` key.

Result

After executing the instruction to load the value into `R0`, the content of `R0` was observed to be **11** (in decimal, or `0x0000 000B` in hexadecimal). The value remained unchanged as the program entered an infinite loop, halting at the breakpoint.

Explanation

The instruction executed loads the value `11` directly into register `R0`. Since no further instructions modify the value of `R0`, it retains the same value throughout the program's execution. The infinite loop at the breakpoint ensures that the program remains in a halt state without altering the register's content.

2.2 Task 2: Modification of Task 1 (replaced 11 by `&FFFFFFFF`)

Given ARM Code

```
    AREA Program, CODE, READONLY
    ENTRY
        MOV r0,#&FFFFFFFF
Stop
        B Stop
    END
```

Procedure

In this task, the value `11` was replaced with `&FFFFFFFF`. The program was executed in debug mode, with a breakpoint set at the loop, and the content of `R0` was observed.

Result

The following error message was encountered:

Error: Immediate value out of range for the MOV instruction.

This error occurred because the `MOV` instruction cannot handle large immediate values like `&FFFFFFFF`.

Explanation

The value `&FFFFFFFF` is too large for the `MOV` instruction, which has limitations on the size of immediate values it can process. As a result, the program failed to load `&FFFFFFFF` into `R0`.

2.3 Task 3: Running another ARM code

Given ARM Code

```
    AREA Reset, CODE, READONLY
    ENTRY
    LDR r0, =7          ; Load the value 7 into register r0
    MUL r1, r0, r0      ; r1 = r0 * r0
    LDR r2, =4          ; Load the value 4 into register r2
    MUL r1, r2, r1      ; Multiply r1 by r2 (r1 = r1 * r2)
    LDR r3, =3          ; Load the value 3 into register r3
    MUL r3, r0, r3      ; Multiply r3 by r0 (r3 = r0 * r3)
    ADD r1, r1, r3      ; Add r3 to r1 (r1 = r1 + r3)
stop
    B stop              ; Infinite loop: branch to stop
    END
```

Explanation

The code starts with setting the directives. Then, the register r0 is loaded with immediate value 7. r1 is stored with the square of 7. r2 is loaded with value 4, and then r1 is updated with product of values stored in r2 and r1. r3 is loaded with value 3, then it is updated with product of values stored previously in it and r0. r1 is then updated with sum of values in it and r3.

Result

We observe the values of r1. r1 is initially 49 (7×7), then r1 has value 196(49×4). Since r3 is updated with 21 (7×3), then finally r1 has value 217($196 + 21$).

2.4 Task 4: ARM code to find 10^{th} term in Fibonacci Sequence

Code Explanation

Load the first two Fibonacci values (0 and 1) into R0 and R1. Set a counter value of 8 in register R2. Inside the Loop label, decrement the counter and if the counter becomes 0, jump to Result. When counter is non zero, find the next Fibonacci number by adding the previous two and store in R0.

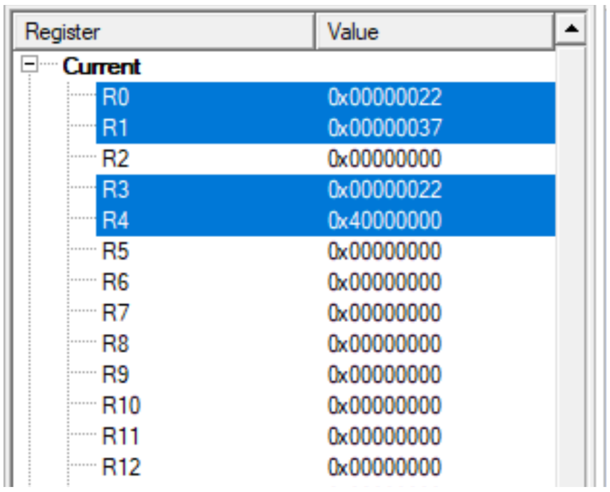
Inside the Result label, R4 is made to store the answer of our answer (10th Fibonacci number). Then store its value in R0, which has the value of the 10th Fibonaaci number.

Result

By setting the counter to 8,the Loop runs for 8 times so that it gives 8th value after the first two Fibonacci numbers (essentially, the 10th Fibonacci number). Finally, R0 stores the value 34.

Code and Output

ARM code: contains the .asm file which has the ARM code.



Register	Value
Current	
R0	0x00000022
R1	0x00000037
R2	0x00000000
R3	0x00000022
R4	0x40000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000

Figure 1: R1 contains 10th Fibonacci sequence in hexadecimal

2.5 Task 5: ARM code to find quotient and remainder when 32 bit is divided by 16 bit

Code Explanation

This ARM assembly program performs integer division, calculating both the quotient and remainder. The key components are:

Registers

- `R0`: Dividend (numerator).
- `R1`: Divisor (denominator).
- `R2`: Quotient (initialized to 0).

- `R3`: Error flag (set for division by 0).

Program Flow

1. `LDR R0, =0xAB124563` and `LDR R1, =0x00001373`: Load dividend and divisor.
2. `MOV R2, #0`: Initialize quotient.
3. `CMP R1, #0`: Check for division by 0.
4. `CMP R0, R1`: Compare dividend and divisor.
5. `ADD R2, R2, #1` and `SUB R0, R0, R1`: Loop to subtract divisor from dividend, incrementing quotient.

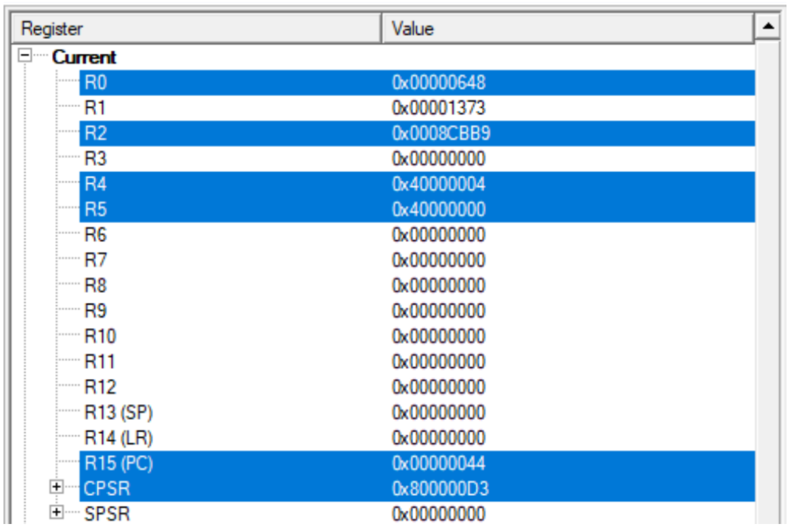
Error Handling and Result

- `MOV R3, #0xFFFFFFFF`: Set error flag if divisor is 0.
- Store quotient and remainder: `STR R2, [Quotient]`, `STR R0, [Remainder]`.

The quotient is the number of subtractions and the remaining is the leftover dividend.

Code and Output

ARM code: contains the .asm file which has the ARM code.



Register	Value
Current	
R0	0x00000648
R1	0x00001373
R2	0x0008CBB9
R3	0x00000000
R4	0x40000004
R5	0x40000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000044
CPSR	0x800000D3
SPSR	0x00000000

Figure 2: R0 contains remainder and R2 contains the quotient for the division problem: 0xAB124563÷0x00001373