

ARM Assembly Programming

Aneesh Kandi EE20B009

October 2021

1 Aim

To (a) learn the architecture of ARM processor (b) learn basics of ARM instruction set, in particular the ARM instructions pertaining to computations (c) go through example programs and (d) write assembly language programs for the given set of (computational) problems

2 Program Codes

2.1 Problem 1

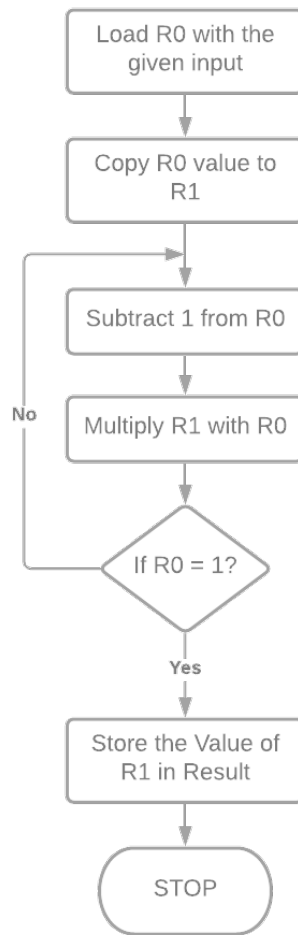
Compute the factorial of a given number using ARM processor through assembly programming

Code

```
1 ; Compute the factorial of a given number using ARM
2 ; processor through assembly programming
3
4 ;-----PROGRAM 1-----
5         AREA Program, CODE, READONLY
6         LDR R0, Value
7         MOV R1, R0
8 Loop    SUB R0, R0, #1
9         MUL R1, R0, R1
10        CMP R0, #1
11        BNE Loop
12        LDR R2, Result
13        STR R1, [R2]
14        SWI &11
15
16 Value   DCD &000A
17 Result  DCD &40000000
18        END
```

Code Explained

In this program, we're using ARM assembly programming to calculate the factorial of a number. Here's the flowchart explaining the logic of the code:



So R0 value decreases every iteration and is multiplied with R1 and at the end, the factorial is stored in R1

Result

Here's the screenshot of the result:

The screenshot displays a debugger interface with two main panels: **Registers** and **Memory 1**.

Registers Panel:

Register	Value
Current	
R0	0x00000001
R1	0x00375F00
R2	0x40000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000024
R15 (PC)	0x00000008
CPSR	0x600000D3
SPSR	0x600000D3
User/System	

Memory 1 Panel:

Address: 0x40000000

Address	Value (Hex)
0x40000000	00 5F 37 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000025	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000004A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000006F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000094	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x400000B9	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x400000DE	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000103	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000128	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000014D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000172	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000197	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x400001BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x400001E1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000206	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x4000022B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000250	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x40000275	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

The bottom of the window shows a tabbed interface with **Project**, **Registers**, **Disassembly**, **Call Stack + Locals**, and **Memory 1** selected.

2.2 Problem 2

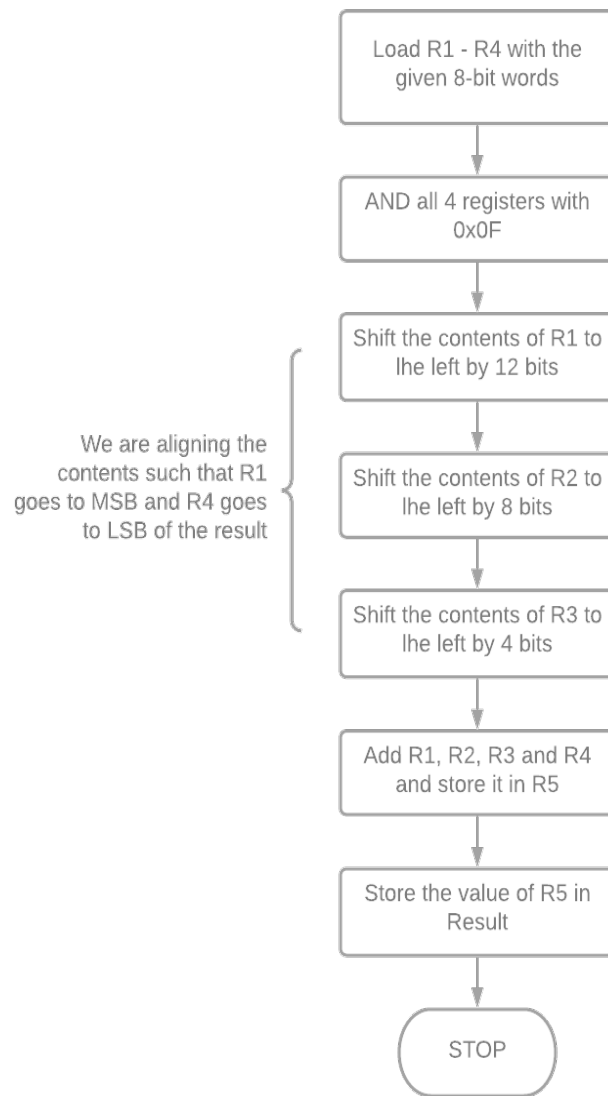
Combine the low four bits of each of the four consecutive bytes beginning at LIST into one 16-bit halfword. The value at LIST goes into the most significant nibble of the result. Store the result in the 32-bit variable RESULT.

Code

```
1 ; For example, 0C 42 36 09
2 ; Result = 0000C269
3
4 ;-----PROGRAM 2-----
5     AREA Program, CODE, READONLY
6     LDR R0, =LIST
7     LDMIA R0, {R1,R2,R3,R4}
8     AND R1, R1, #&0F
9     AND R2, R2, #&0F
10    AND R3, R3, #&0F
11    AND R4, R4, #&0F
12
13    ADD R5, R5, R1, LSL#12
14    ADD R5, R5, R2, LSL#8
15    ADD R5, R5, R3, LSL#4
16    ADD R5, R5, R4
17
18    LDR R6, Result
19    STR R5, [R6]
20    SWI &11
21
22 LIST    DCD &0C, &42, &36, &09
23 Result  DCD &40000000
24    END
```

Code Explained

This program exploits the use of command 'AND'. According to the problem, we need the lower nibble of the 8-bit word. We use AND command to extract it from each word and then align them in the given order(value at LIST goes to MSB of the halfword). Here's the flowchart:



Result

Here's the screenshot of the result:

The screenshot displays a debugger window with two main panes: 'Registers' on the left and 'Disassembly' on the right. The 'Registers' pane shows a list of registers with their current values. The 'Disassembly' pane shows the memory contents at address 0x40000000 and the corresponding assembly instructions.

Registers

Register	Value
R0	0x00000034
R1	0x0000000C
R2	0x00000002
R3	0x00000006
R4	0x00000009
R5	0x0000C269
R6	0x40000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000030
CPSR	0x000000D3
SPSR	0x00000000
User/System	

Disassembly

Memory 1

Address: 0x40000000

0x40000000: 69 C2 00

0x40000015: 00

0x4000002A: 00

0x4000003F: 00

0x40000054: 00

0x40000069: 00

0x4000007E: 00

0x40000093: 00

0x400000A8: 00

0x400000BD: 00

0x400000D2: 00

0x400000E7: 00

0x400000FC: 00

Call Stack + Locals

Memory 1

25 LIST DCD &0C, &42, &36, &09

26 Result DCD &40000000

27 END

2.3 Problem 3

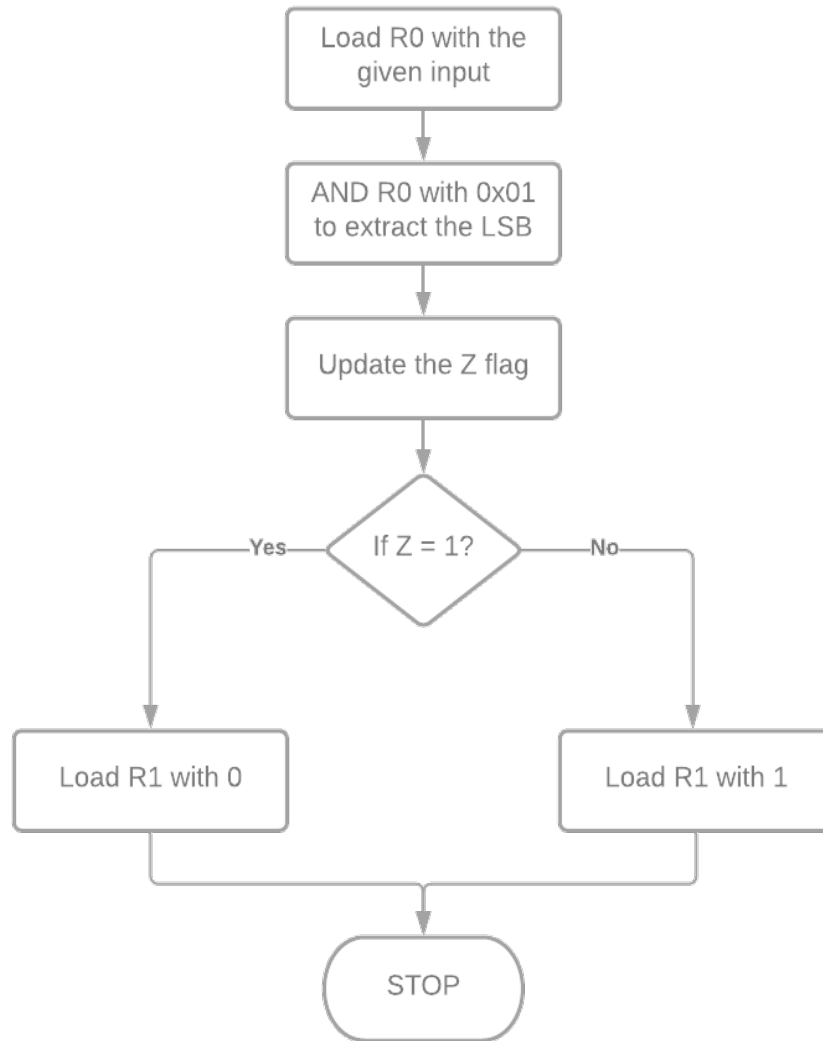
Given a 32 bit number, identify whether it is an even or odd. (Your implementation should not involve division).

Code

```
1 ; Given a 32 bit number, identify whether it is an even or odd.
2 ; Note : If the given number is Odd, then Register R1 will have
3 ; 1 and if the given number is even, the Register R1 will have 0
4
5 ; -----PROGRAM 3-----
6         AREA Program, CODE, READONLY
7         LDR R0, Value
8         ANDS R0, #0x01
9         BEQ even
10        MOV R1, #1
11        B odd
12 even    MOV R1, #0
13 odd     SWI &11
14
15 Value   DCD &12910301
16        END
```

Code Explained

To find out if a given number is odd or even, we have to check the LSB bit. If the LSB is 1, then it is an odd number because it is not a multiple of 2. Similarly, if LSB is 0, then it is an even number. Here's the flowchart:



Result

Here's the screenshot of the result:

The screenshot displays a debugger interface with two main panels. The left panel, titled 'Registers', shows a list of registers from R0 to R15, along with CPSR and SPSR. R15 (PC) is highlighted with a blue bar, showing a value of 0x00000018. The right panel, titled 'Disassembly', shows the assembly code for a program named 'evenodd.s'. The code includes comments and instructions for checking if a number is odd or even. The instruction at address 0x00000018 is highlighted in yellow: 'SWI 0x00000011'. Below this, the instruction at address 0x0000001C is highlighted in green: 'ADDNES R0, R1, #0x04000000'. The instruction at address 0x00000020 is also highlighted in green: 'ANDREQ R0, R0, R0'. The code ends with 'END' at address 0x00000016.

Register	Value
R0	0x00000001
R1	0x00000001
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000018
CPSR	0x000000D3
SPSR	0x00000000
User/System	

```
13: odd      SWI &11
0x00000018 EF000011 SWI 0x00000011
0x0000001C 12910301 ADDNES R0,R1,#0x04000000
0x00000020 00000000 ANDREQ R0,R0,R0

evenodd.s
2 ; Note : If the given number is Odd, then Reg
3 ; 1 and if the given number is even, the Regi
4
5 ; -----PROGRAM 3-----
6 AREA Program, CODE, READONLY
7 LDR R0, Value
8 ANDS R0, #0x01
9 BEQ even
10 MOV R1, #1
11 B odd
12 even MOV R1, #0
13 odd SWI &11
14
15 Value DCD &12910301
16 END
```

0x12910301 is an Odd Number

3 Inferences and Learnings

- Learnt various commands used in ARM Programming.
- Explored ARM architecture and Instruction Set.
- Learnt about the different directives present in ARM like DCW, DCD, etc.
- Using shifting and looping operations in ARM
- Learnt how to use Keil software

4 Link to My Codes

https://drive.google.com/file/d/1KS3QAA1tocm_Ffqp1SZS0Npj_NYdP3qQ/view?usp=sharing

This drive link consists of a zipped file containing all the files related to the codes.

—————-THANK YOU—————-