



Department of Computer Science & Engineering (AI & ML)
Microprocessor & Computer Architecture Lab

Lab 1 Programs

UE23CS251B

- 1 Write an ALP to perform Addition for of two numbers of size
a)64 bit
b)128 bit
save the result in register (reuse the register to store the result)

Hint: For 128 bit given below

R3 R2 R1 R0
+R7 R6 R5 R4

a)

.text

.global _start

_start:

LDR R0, =0x12345678

LDR R1, =0x87654321

LDR R2, =0x11111111

LDR R3, =0x22222222

ADDS R0, R0, R2

ADC R1, R1, R3

ARMSim - The ARM Simulator Dept. of Computer Science

File View Cache Debug Watch Help

RegistersView 1a.s

General Purpose

Hexadecimal

Unsigned Decimal

Signed Decimal

R0 :23456
R1 :a9876
R2 :11111
R3 :22222
R4 :00000
R5 :07800
R6 :00000
R7 :00000
R8 :00000
R9 :00000
R10 (s1) :00000
R11 (fp) :00000
R12 (ip) :00000
R13 (sp) :00005
R14 (lr) :00000
R15 (pc) :00011

CPSR Register
Negative (N) :0
Zero (Z) :1
Carry (C) :0
Overflow (V) :0
IRQ Disable:1
FIQ Disable:1
Thumb (T) :0
CPU Mode :S

0x400000df

```
.text
.global _start

00001000:      _start:
00001000:E59F0010      LDR R0, =0x12345678
00001004:E59F1010      LDR R1, =0x87654321
00001008:E59F2010      LDR R2, =0x11111111
0000100C:E59F3010      LDR R3, =0x22222222

00001010:E0900002      ADDS R0, R0, R2
00001014:E0A11003      ADC R1, R1, R3
```

```

b)
.text
.global _start_128
_start_128:
    LDR R0, =0x11111111
    LDR R1, =0x22222222
    LDR R2, =0x33333333
    LDR R3, =0x44444444

    LDR R4, =0x55555555
    LDR R5, =0x66666666
    LDR R6, =0x77777777
    LDR R7, =0x88888888

    ADDS R0, R0, R4
    ADCS R1, R1, R5
    ADCS R2, R2, R6
    ADC R3, R3, R7

```

ARMSim - The ARM Simulator Dept. of Computer Science

File View Cache Debug Watch Help

RegistersView 1b.s

General Purpose

Hexadecimal

Unsigned Decimal

Signed Decimal

R0 : 66666

R1 : 88888

R2 : aaaaa

R3 : ccccc

R4 : 55555

R5 : 00000

R6 : 00000

R7 : 88888

R8 : ccccc

R9 : 00000

R10 (s1) : 00000

R11 (fp) : 00000

R12 (ip) : 00000

R13 (sp) : 00005

R14 (lr) : 00000

R15 (pc) : 00011

CPSR Register

Negative (N) : 0

Zero (Z) : 0

Carry (C) : 1

Overflow (V) : 1

IRQ Disable : 1

FIQ Disable : 1

Thumb (T) : 0

CPU Mode : S

0x300000df

```

.text
.global _start_128
_start_128:
00001000: E59F0028 LDR R0, =0x11111111
00001004: E59F1028 LDR R1, =0x22222222
00001008: E59F2028 LDR R2, =0x33333333
0000100C: E59F3028 LDR R3, =0x44444444

00001010: E59F4028 LDR R4, =0x55555555
00001014: E59F5028 LDR R5, =0x66666666
00001018: E59F6028 LDR R6, =0x77777777
0000101C: E59F7028 LDR R7, =0x88888888

00001020: E0900004 ADDS R0, R0, R4
00001024: E0B11005 ADCS R1, R1, R5
00001028: E0B22006 ADCS R2, R2, R6
0000102C: E0A33007 ADC R3, R3, R7

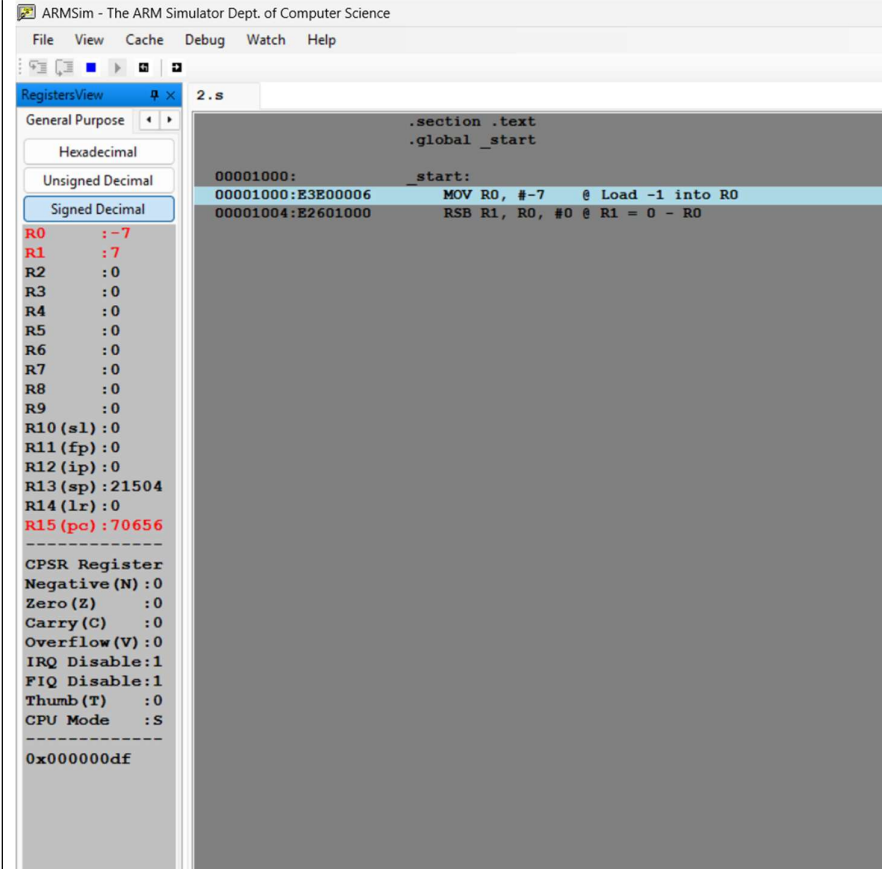
```

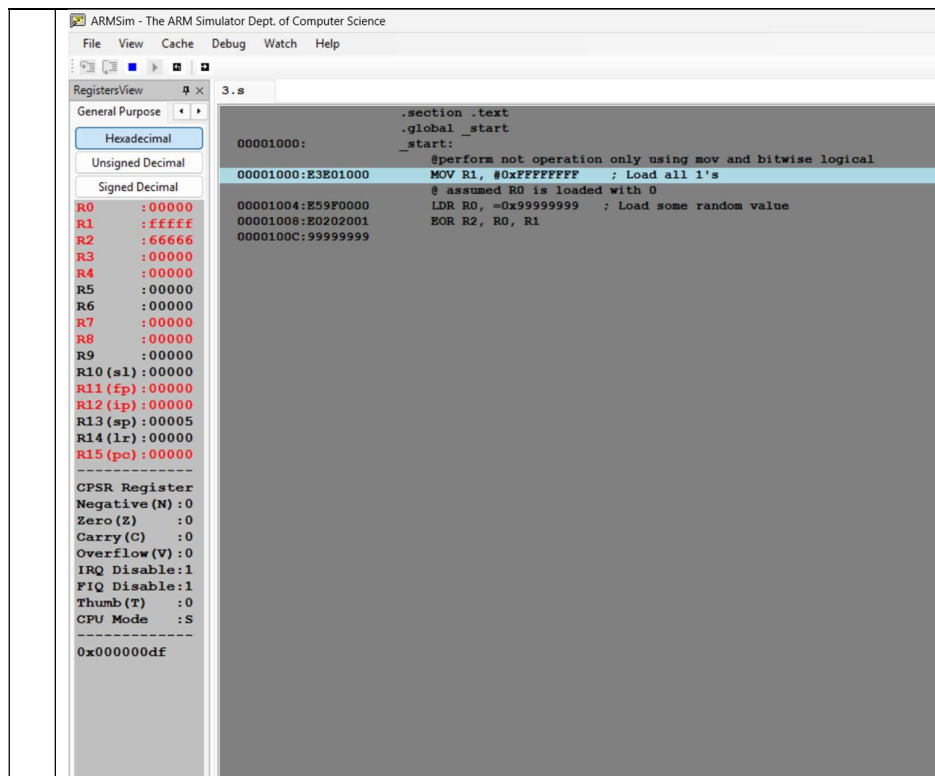
- 2 Write an ALP to perform 2's complement only using mov and RSB instruction.
- ```

.section .text
.global _start

_start:
 MOV R0, #-7 @ Load -1 into R0
 RSB R1, R0, #0 @ R1 = 0 - R0

```

|   |                                                                                                                                                                                                                                                                                                                                                           |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   |                                                                                                                                                                                                                                                                        |
| 3 | <p>Write an ALP to perform not operation only using mov and bitwise logical instructions.</p> <pre>.section .text .global _start _start:     @perform not operation only using mov and bitwise logical     MOV R1, #0xFFFFFFFF ; Load all 1's     @ assumed R0 is loaded with 0     LDR R0, =0x99999999 ; Load some random value     EOR R2, R0, R1</pre> |



4 Write an ALP to subtract if the numbers are equal, otherwise add them.

```
.text
.global _start

_start:
 MOV R0, #20 @ Load 20 into R0
 MOV R1, #10 @ load 10 into R1

 CMP R0, R1 @ Compare R0 and R1
 BEQ subtract
 ADD R2,R0,R1 @ R2 = R0 + R1
 B end

subtract:
 SUB R2,R0,R1 @ R2 = R0 - R1

end:
```

ARMSim - The ARM Simulator Dept. of Computer Science

File View Cache Debug Watch Help

RegistersView 4.s

General Purpose

Hexadecimal

Unsigned Decimal

Signed Decimal

R0 : 20  
R1 : 10  
R2 : 30  
R3 : 0  
R4 : 0  
R5 : 0  
R6 : 0  
R7 : 0  
R8 : 90  
R9 : 0  
R10 (s1) : 0  
R11 (fp) : 0  
R12 (ip) : 0  
R13 (sp) : 21504  
R14 (lr) : 0  
R15 (pc) : 70656

CPSR Register

Negative (N) : 0  
Zero (Z) : 0  
Carry (C) : 1  
Overflow (V) : 0  
IRQ Disable : 1  
FIQ Disable : 1  
Thumb (T) : 0  
CPU Mode : S

0x200000df

```

.text
.global _start
00001000: start:
00001000:E3A00014 MOV R0, #20 @ Load 20 into R0
00001004:E3A0100A MOV R1, #10 @ load 10 into R1
00001008:E1500001 CMP R0, R1 @ Compare R0 and R1
0000100C:0AD00001 BEQ subtract
00001010:E0802001 ADD R2,R0,R1 @ R2 = R0 + R1
00001014:EAD00000 B end
00001018: subtract:
00001018:E0402001 SUB R2,R0,R1 @ R2 = R0 - R1
0000101C: end:

```

- 5 Check if a given number is even or odd.
- Note: at the end of the program execution R2 contains 0 if number is even, otherwise R2 contains 1.

```

.text
.global _start
_start:
 MOV R0, #21 @ Load number to check
 AND R1, R0, #1 @ R1 = R0 & 1 (remainder when divided by 2)
 MOV R2, R1 @ Move result to R2 (0 for even, 1 for odd)

```

ARMSim - The ARM Simulator Dept. of Computer Science

File View Cache Debug Watch Help

RegistersView 5.s

General Purpose

Hexadecimal

Unsigned Decimal

Signed Decimal

R0 : 21  
R1 : 1  
R2 : 1  
R3 : 0  
R4 : 0  
R5 : 0  
R6 : 0  
R7 : 0  
R8 : 0  
R9 : 0  
R10 (s1) : 0  
R11 (fp) : 0  
R12 (ip) : 0  
R13 (sp) : 21504  
R14 (lr) : 0  
R15 (pc) : 70656

CPSR Register

Negative (N) : 0  
Zero (Z) : 0  
Carry (C) : 0  
Overflow (V) : 0  
IRQ Disable : 1  
FIQ Disable : 1  
Thumb (T) : 0  
CPU Mode : S

0x000000df

```

.text
.global _start
00001000: start:
00001000:E3A00015 MOV R0, #21 @ Load number to check
00001004:E2001001 AND R1, R0, #1 @ R1 = R0 & 1 (remainder when divided by 2)
00001008:E1A02001 MOV R2, R1 @ Move result to R2 (0 for even, 1 for odd)

```

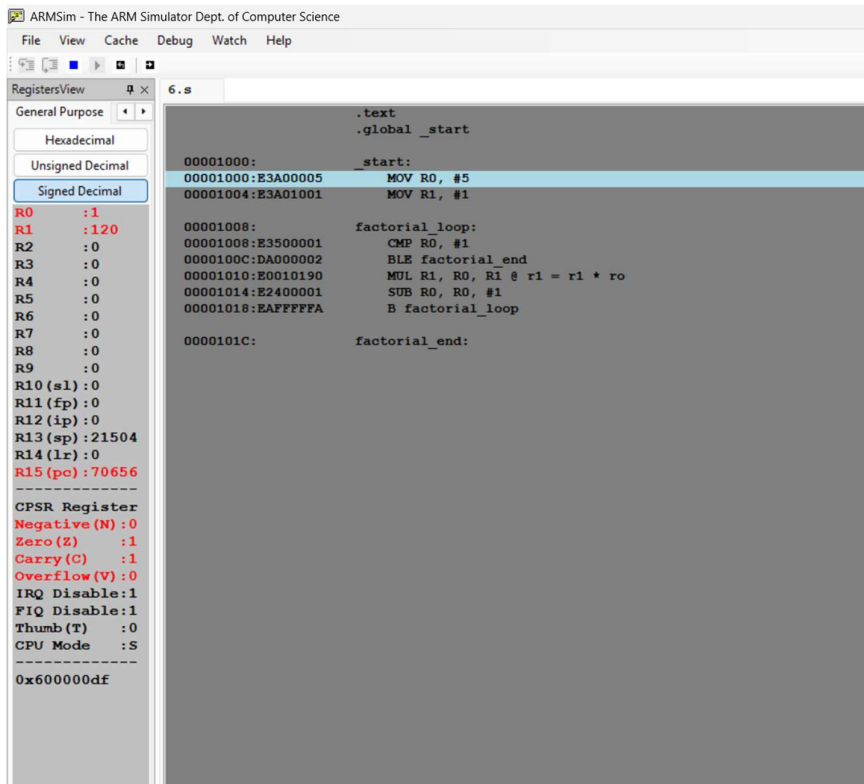
6 Write a program to find the factorial of a given number.

```
.text
.global _start

_start:
 MOV R0, #5
 MOV R1, #1

factorial_loop:
 CMP R0, #1
 BLE factorial_end
 MUL R1, R0, R1 @ r1 = r1 * ro
 SUB R0, R0, #1
 B factorial_loop

factorial_end:
```



ARMSim - The ARM Simulator Dept. of Computer Science

File View Cache Debug Watch Help

RegistersView 6.s

General Purpose

Hexadecimal

Unsigned Decimal

Signed Decimal

R0 : 1

R1 : 120

R2 : 0

R3 : 0

R4 : 0

R5 : 0

R6 : 0

R7 : 0

R8 : 0

R9 : 0

R10 (s1) : 0

R11 (fp) : 0

R12 (ip) : 0

R13 (sp) : 21504

R14 (lr) : 0

R15 (pc) : 70656

-----

CPSR Register

Negative (N) : 0

Zero (Z) : 1

Carry (C) : 1

Overflow (V) : 0

IRQ Disable: 1

FIQ Disable: 1

Thumb (T) : 0

CPU Mode : S

-----

0x600000df

```
.text
.global _start

00001000: _start:
00001000:E3A00005 MOV R0, #5
00001004:E3A01001 MOV R1, #1

00001008: factorial_loop:
00001008:E3500001 CMP R0, #1
0000100C:DAD00002 BLE factorial_end
00001010:E0010190 MUL R1, R0, R1 @ r1 = r1 * ro
00001014:E2400001 SUB R0, R0, #1
00001018:EAF00001 B factorial_loop

0000101C: factorial_end:
```

7 Write a program to find GCD of two numbers.

```
@to find gcd of 2 numbers
.text
.global _start

_start:
 MOV R0, #80 @ Load 20 into R0
 MOV R1, #90 @ load 10 into R1
```

```
gcd_loop:
 CMP R0, #0 @ Compare R0 and R1
 BEQ gcd_end
```

```
 CMP R0,R1
 BLT swap
 SUB R0,R0,R1
 B gcd_loop
```

```
swap:
 MOV R2,R0
 MOV R0,R1
 MOV R1,R2
 B gcd_loop
```

```
gcd_end:
```

The screenshot displays the ARMSim - The ARM Simulator interface. The main window shows assembly code for finding the GCD of two numbers. The code includes labels for `gcd_loop`, `swap`, and `gcd_end`. The registers view on the left shows the current state of the processor registers. The CPSR register is also visible, showing flags like Negative (N), Zero (Z), Carry (C), and Overflow (V).

```

@to find gcd of 2 numbers
.text
.global _start

00001000: _start:
00001000:E3A00050 MOV R0, #80 @ Load 20 into R0
00001004:E3A0105A MOV R1, #90 @ load 10 into R1

00001008: gcd_loop:
00001008:E3500000 CMP R0, #0 @ Compare R0 and R1
0000100C:0A000007 BEQ gcd_end

00001010:E1500001 CMP R0,R1
00001014:BA000001 BLT swap
00001018:E0400001 SUB R0,R0,R1
0000101C:EAF00009 B gcd_loop

00001020: swap:
00001020:E1A02000 MOV R2,R0
00001024:E1A00001 MOV R0,R1
00001028:E1A01002 MOV R1,R2
0000102C:EAF00005 B gcd_loop

00001030: gcd_end:

```

RegistersView: 7.s

General Purpose: ☐ Hexadecimal ☐ Unsigned Decimal ☒ Signed Decimal

|          |        |
|----------|--------|
| R0       | :0     |
| R1       | :10    |
| R2       | :10    |
| R3       | :0     |
| R4       | :0     |
| R5       | :0     |
| R6       | :0     |
| R7       | :0     |
| R8       | :0     |
| R9       | :0     |
| R10 (s1) | :0     |
| R11 (fp) | :0     |
| R12 (ip) | :0     |
| R13 (sp) | :21504 |
| R14 (lr) | :0     |
| R15 (pc) | :70656 |

CPSR Register

|              |    |
|--------------|----|
| Negative (N) | :0 |
| Zero (Z)     | :1 |
| Carry (C)    | :1 |
| Overflow (V) | :0 |
| IRQ Disable  | :1 |
| FIQ Disable  | :1 |
| Thumb (T)    | :0 |
| CPU Mode     | :S |

0x600000df