**Lab 8: ARM Assembly Instructions Advanced**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

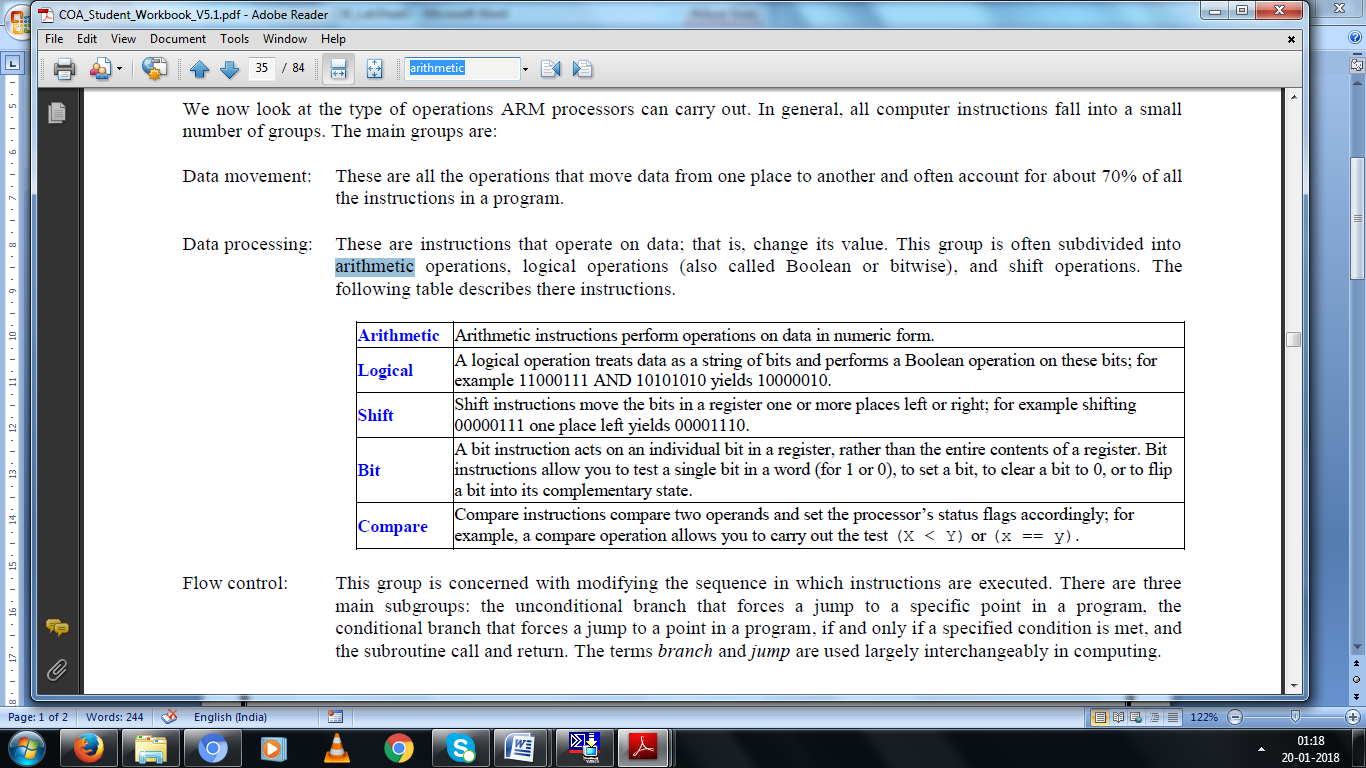
**Name: Vaisakh K V Roll Number:AM.EN.U4CSE17342**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

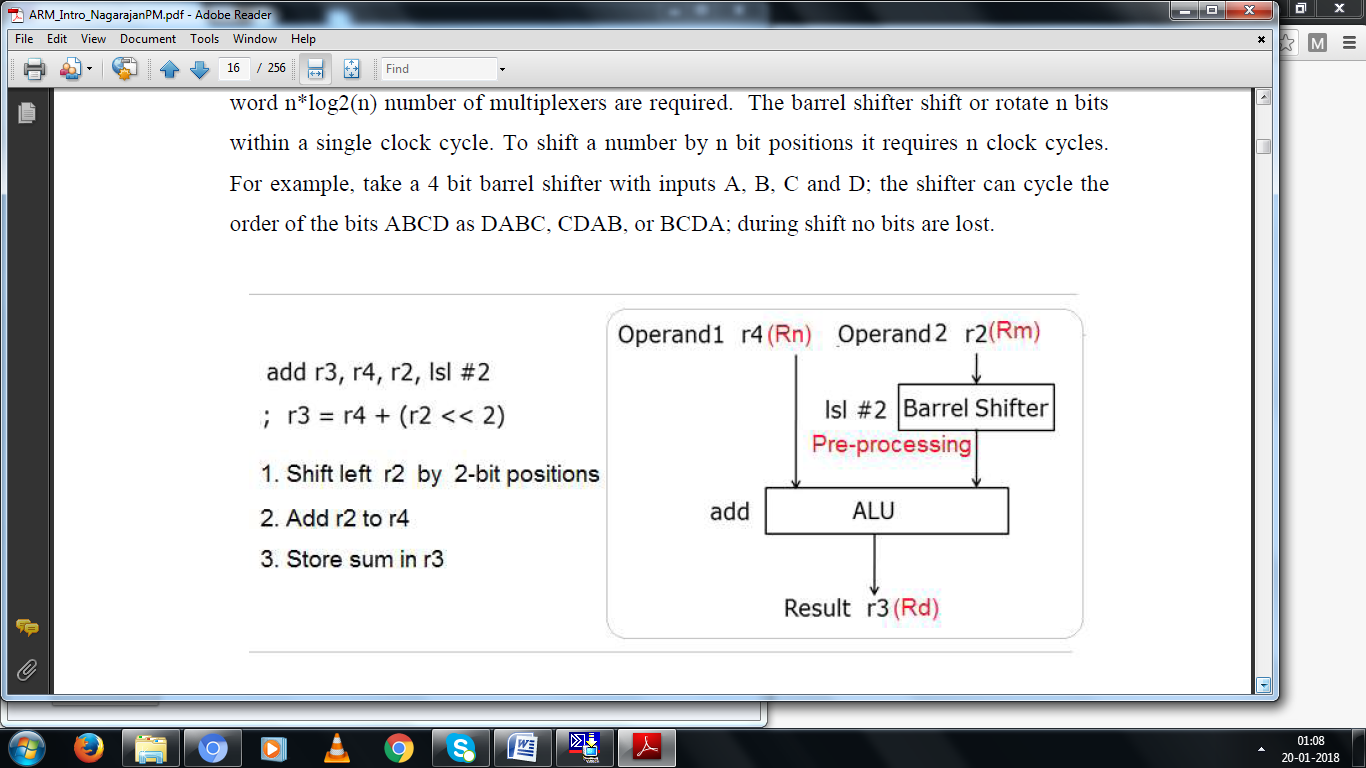
**Part 1: Data Processing ARM Instructions**

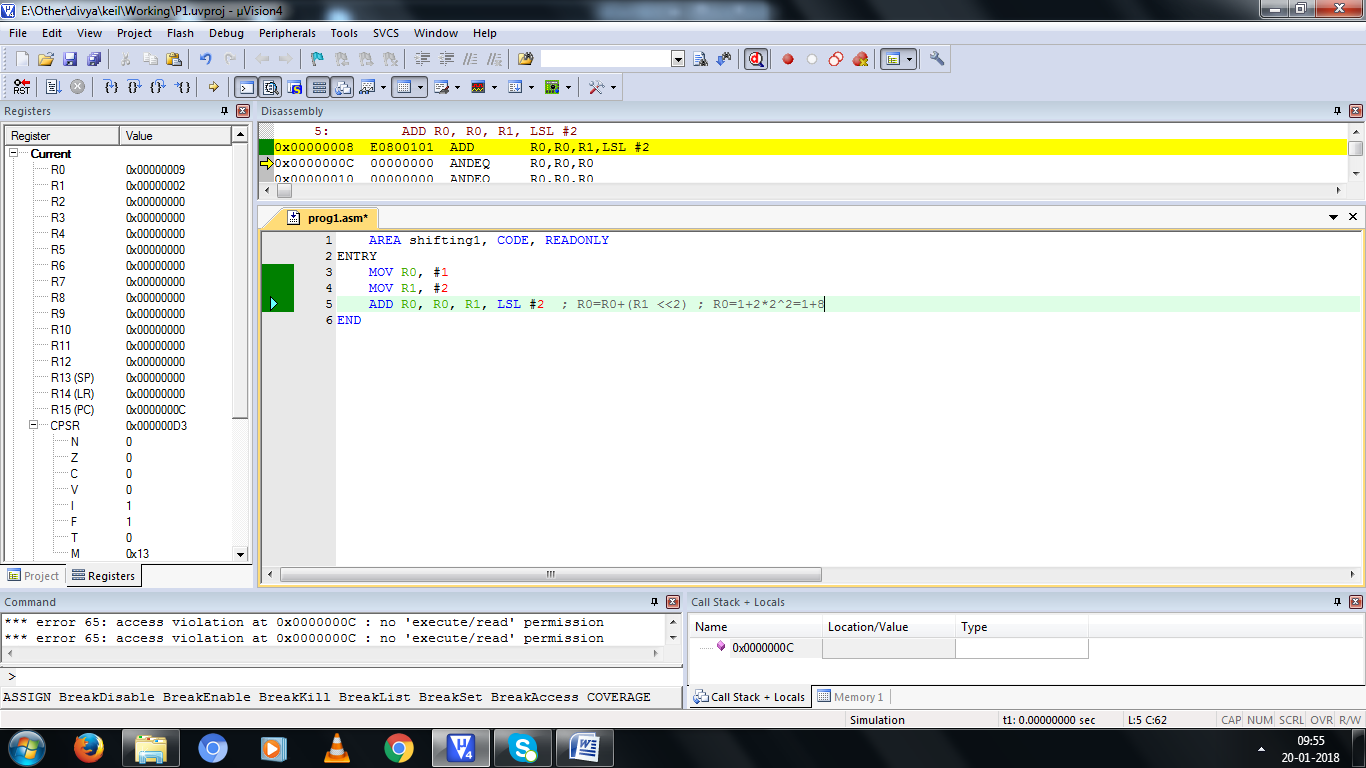
Computer instructions can be classified into Data Movement, Data Processing and Flow Control.

Data Processing Instructions can be further classified into Arithmetic, Logical, Shift, Bit and Compare.



Example and working of preprocessing instruction LSL:



****

Fill the following table by modifying the above program.

|  |  |  |  |
| --- | --- | --- | --- |
| **R0** | **R1** | **Preprocessing #i** | **R0 after executing ADD R0, R0, R1 Preprocessing #i** |
| 1 | 2 | LSL #2 | **0x000000009** |
| 1 | 2 | LSL #3 | **0x000000011** |
| 1 | 2 | LSL #5 | **0x000000041** |
| 1 | 2 | ASR #2 | **0x000000001** |
| -1 | 2 | LSR #2 | **0xFFFFFFFF** |
| -1 | 2 | ROR #2 | **0x7FFFFFFF** |

Example program using CMP to find the biggest among 2 numbers (R0, R1)

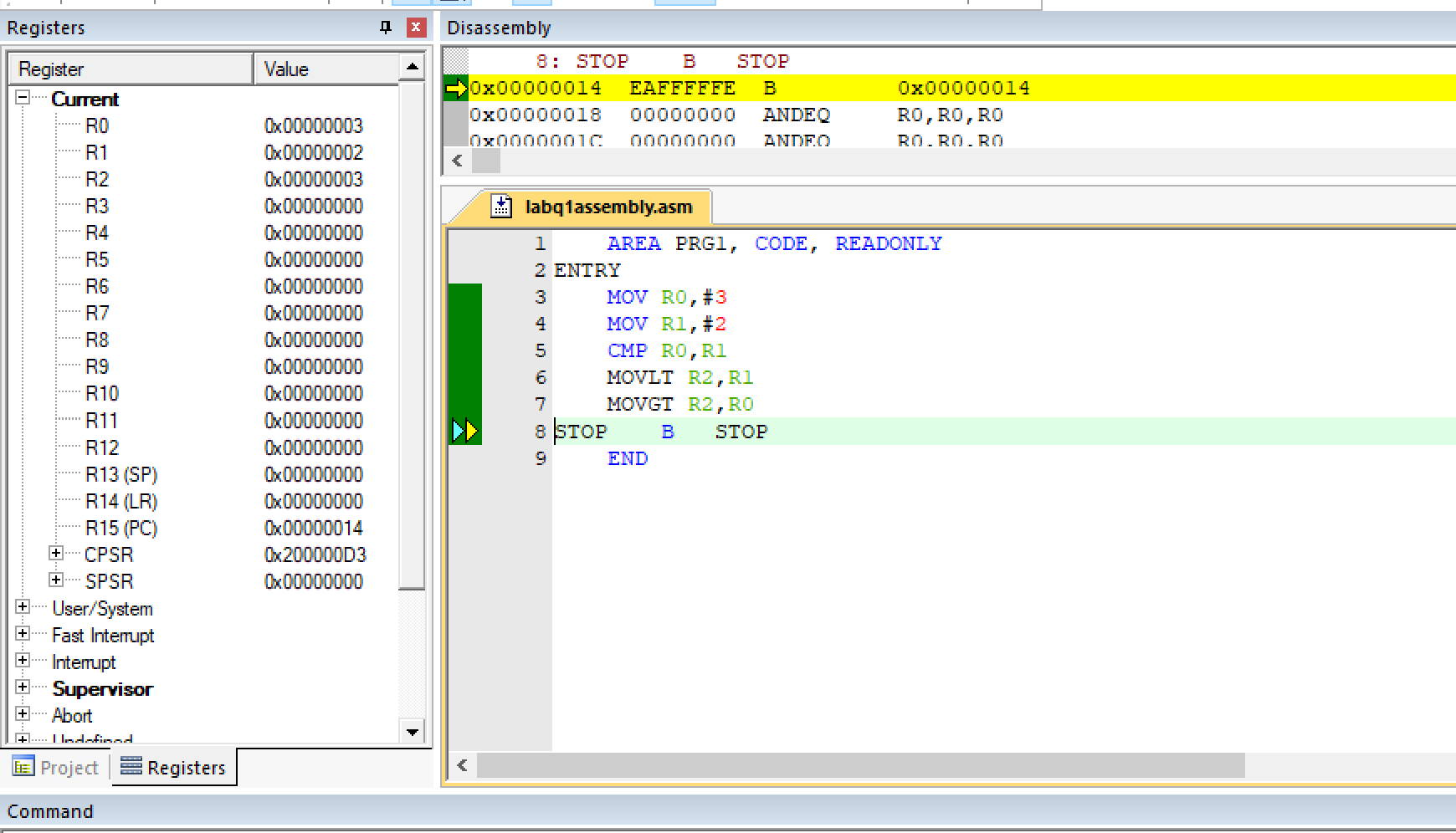
|  |  |
| --- | --- |
| **Equivalent C code**  if(r0<=r1)  r2=r1  else  r2=r0 | **Screenshot** |

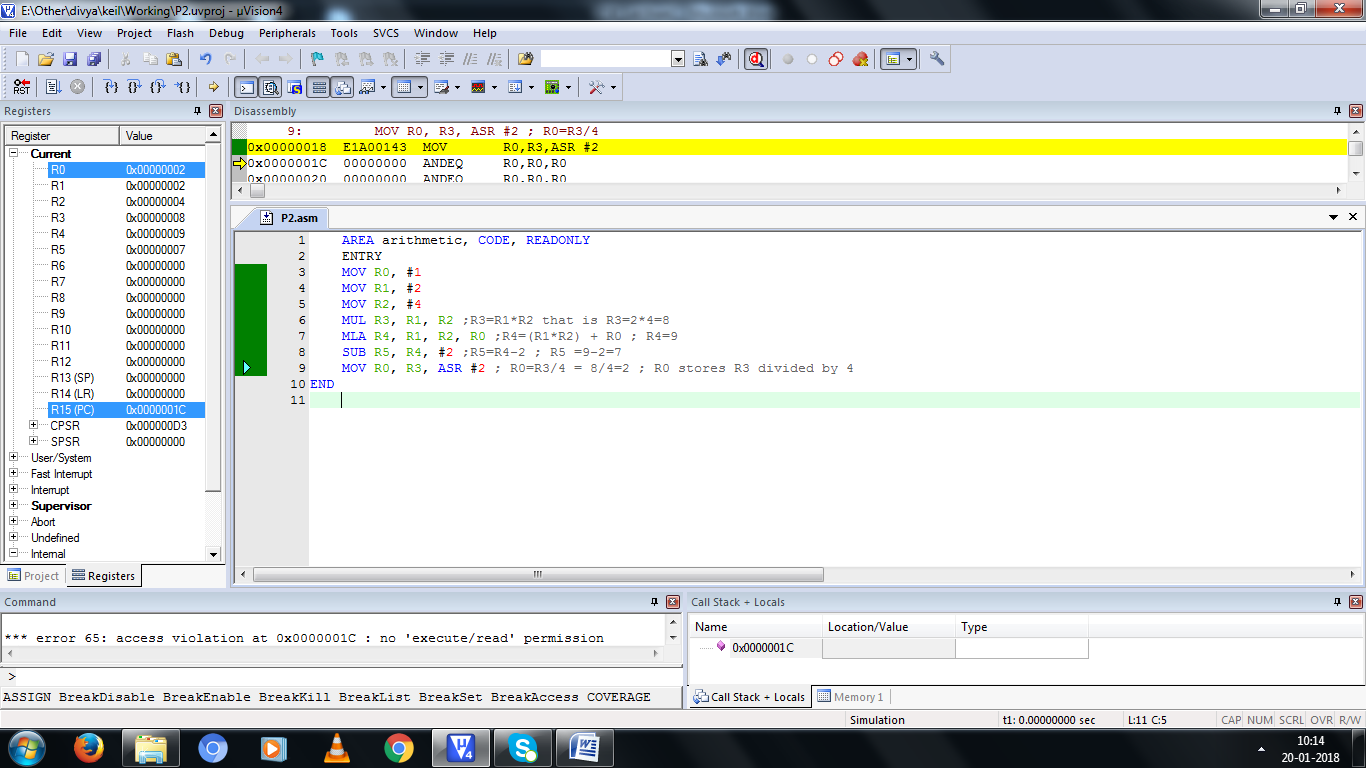
Sample program for performing arithmetic operations.

Fill the following table after the program execution.

|  |  |  |
| --- | --- | --- |
| **R0** | **R1** | **R2** |
| **5** | **3** | **5** |
| **3** | **2** | **3** |
| **5** | **5** | **5** |

Modify this assembly program to get the same result by avoing CMP by appending S to set condition codes and add 2 letters to MOV to make it as conditional execution.

**Program**

****

**Part2 Explore Addressing Modes in Data Transfer ARM Instructions**

**Single Register Load and Store operations**

AREA Pgm5, CODE, READWRITE

ENTRY

ADR R0, Table1 ;pseudo data rocessing instruction to assign address pointer

MOV R3, #6 ;immediate addressing mode

MOV R5, #0

sumlp LDR R4, [R0], #4 ;base plus offset addressing - post indexing

;LDR R4, [R0, #4] ;base plus offset addressing –pre indexing

; LDR R4, [R0, #4]! ;base plus offset addrssing - pre indexing auto - changes the base address

ADD R5, R5, R4

SUBS R3, R3, #1

CMP R3, #0

BNE sumlp

LOOP B LOOP

Table1 DCD 1, 2, 3, 4, 5, 6

END

Modify the above program which uses post indexing base plus offset addressing mode to other two addressing modes to get the same result in R5 (0x00000015)

Fill the following table for each of the loop iteration.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Loop iteration  count# | R5 | R3 | R0 | R4 |
| 1 | 0x00000001 | 0x00000005 | 0x00000028 | 0x00000001 |
| 2 | 0x00000003 | 0x00000004 | 0x0000002C | 0x00000002 |
| 3 | 0x00000006 | 0x00000003 | 0x00000030 | 0x00000003 |
| 4 | 0x0000000A | 0x00000002 | 0x00000034 | 0x00000004 |
| 5 | 0x0000000F | 0x00000001 | 0x00000038 | 0x00000005 |
| 6 | 0x00000015 | 0x00000000 | 0x0000003C | 0x00000006 |

**Assembly Program for Pre Indexing – Auto**

Fill the following table for each of the loop iteration for the registers you have used.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Loop iteration  count# | R5 | R3 | R0 | R4 |
| 1 | 0x00000002 | 0x00000005 | 0x00000028 | 0x00000002 |
| 2 | 0x00000005 | 0x00000004 | 0x0000002C | 0x00000003 |
| 3 | 0x00000009 | 0x00000003 | 0x00000030 | 0x00000004 |
| 4 | 0x0000000E | 0x00000002 | 0x00000034 | 0x00000005 |
| 5 | 0x00000014 | 0x00000001 | 0x00000038 | 0x00000006 |
| 6 | 0x0000003C | 0x00000000 | 0x0000003C | 0x00000000 |

**Assembly Program for Pre Indexing**

Modify the following table for each of the loop iteration for the registers you have used.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Loop iteration  count# | R5 | R3 | R0 | R4 |
| 1 | 0x00000002 | 0x00000005 | 0x00000024 | 0x00000002 |
| 2 | 0x00000004 | 0x00000004 | 0x00000024 | 0x00000002 |
| 3 | 0x00000006 | 0x00000003 | 0x00000024 | 0x00000002 |
| 4 | 0x00000008 | 0x00000002 | 0x00000024 | 0x00000002 |
| 5 | 0x0000000A | 0x00000001 | 0x00000024 | 0x00000002 |
| 6 | 0x0000000C | 0x00000000 | 0x00000024 | 0x00000002 |

**Multiple Register Load and Store Instructions**

**Stack Addressing**

AREA MoveMultiple, CODE, READWRITE

ENTRY

LDR R1,=0x11111111

LDR R2,=0x22222222

LDR R3,=0x33333333

LDR R5,=0x55555555

ADR R0, Stack

STMIA R0!, {R1-R3, R5}

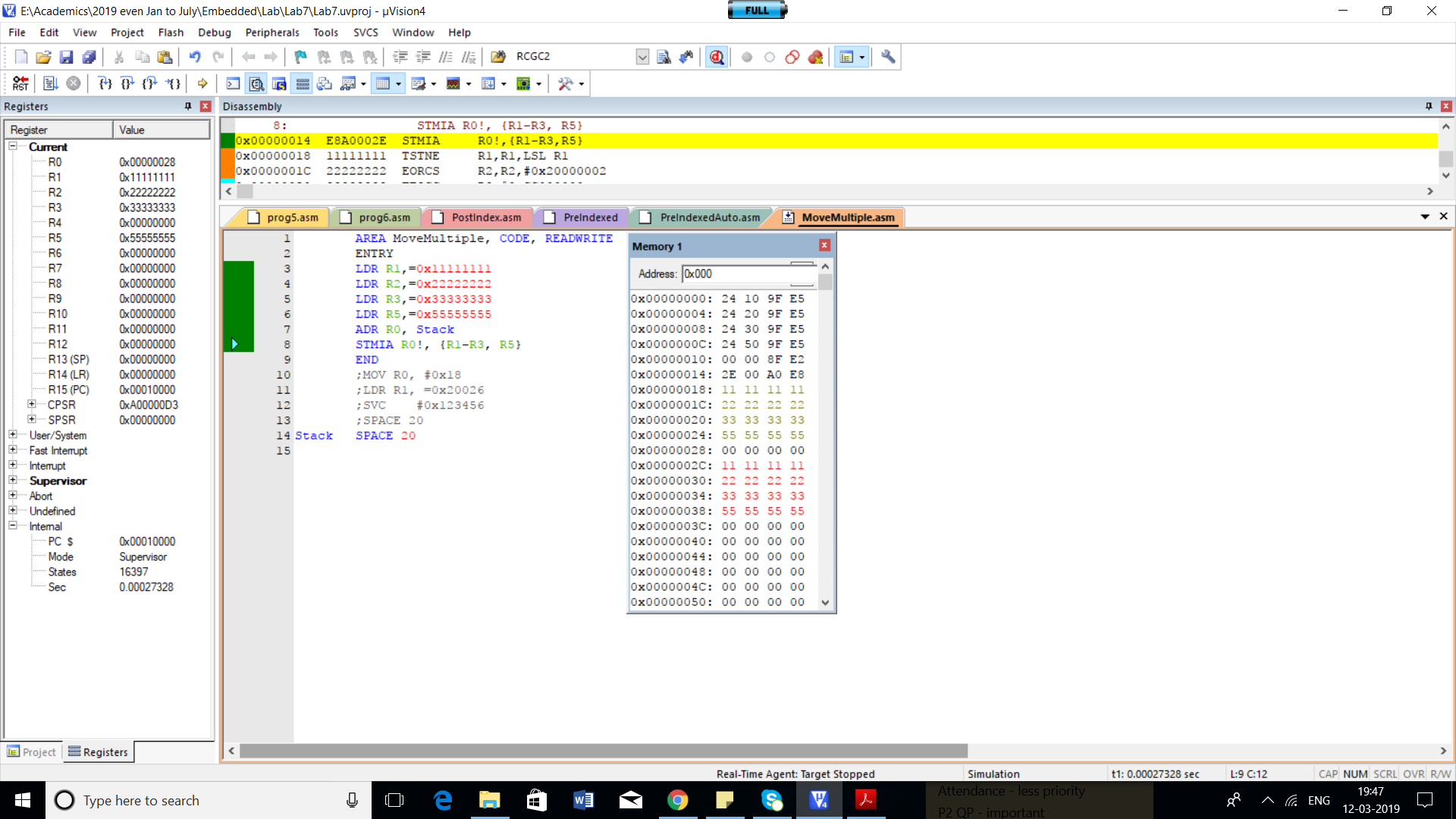
END

Stack SPACE 20

What is the stack addressing mode used in STMIA R0!, {R1-R3, R5}?

a) Empty Ascending mode is used

Explore other 3 stack addressing modes.



Trace the R0 value after pushing each item in the stack for STMIA-empty ascending

|  |  |  |
| --- | --- | --- |
|  | R0 | Remarks |
| Before pushing any value | 0x00000028 | Base address of Stack |
| After pushing R1 value | 0x0000002C |  |
| After pushing R2 value | 0x00000030 |  |
| After pushing R3 value | 0x00000034 |  |
| After pushing R5 value | 0x00000038 |  |

Trace the R0 value after pushing each item in the stack for other 3 stack modes

1)STMIB-full ascending

|  |  |  |
| --- | --- | --- |
|  | R0 | Remarks |
| Before pushing any value | 0x00000028 | Base address of Stack |
| After pushing R1 value | 0x0000002C |  |
| After pushing R2 value | 0x00000030 |  |
| After pushing R3 value | 0x00000034 |  |
| After pushing R5 value | 0x00000038 |  |

2)Empty Descending

|  |  |  |
| --- | --- | --- |
|  | R0 | Remarks |
| Before pushing any value | 0x00000028 | Base address of Stack |
| After pushing R1 value | 0x00000024 |  |
| After pushing R2 value | 0x00000020 |  |
| After pushing R3 value | 0x0000001C |  |
| After pushing R5 value | 0x00000038 |  |

3)Full Descending

|  |  |  |
| --- | --- | --- |
|  | R0 | Remarks |
| Before pushing any value | 0x00000028 | Base address of Stack |
| After pushing R1 value | 0x00000024 |  |
| After pushing R2 value | 0x00000020 |  |
| After pushing R3 value | 0x0000001C |  |
| After pushing R5 value | 0x00000038 |  |

**Part3: Explore array related assembly programs**

**Q1.** What does the following program computes? Mention the need of each and every instruction as comment. Debug for any error.

AREA pgm3, CODE

ENTRY

MAIN

ldr r0,=data1

;ldr r3,=0x40000000

ldr r4,=0x05

ldr r1,[r0],#4

sub r4,r4,#1

BACK ldr r2,[r0]

cmp r1,r2

bls LESS

mov r1,r2

LESS add r0,r0,#4

sub r4,r4,#1

cmp r4,#0

bne BACK

;str r1,[r3]

;AREA data , DATA

data1 dcd &65,&05,&96,&01,&65

END

a) The Program finds the smallest in the array data1. R0 will have the smallest value of the array.

The above program has no errors.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration | R0 | R1 | R2 | R4 |
| 0 | 0x00000030 | 0x00000064 | 0x00000000 | 0x00000004 |
| 1 | 0x00000038 | 0x00000005 | 0x00000005 | 0x00000003 |
| 2 | 0x00000040 | 0x00000001 | 0x00000096 | 0x00000002 |
| 3 | 0x00000044 | 0x00000001 | 0x000000065 | 0x00000000 |

**Provide a suitable table to trace the Register values for each of the iteration.**

**Q2.** What does the following program computes? Mention the need of each and every instruction as comment. Debug for any error.

AREA pgm3, CODE

ENTRY

LDR R0, =str1

MOV R3, #0x0D ; R3 stores the ASCII value of Carriage Return

MOV R2, #0

loop LDRB R1, [R0], #1 ; used to Load 1 byte value

CMP R1, R3

BEQ done

ADD R2, R2, #1

B loop

done B done

str1 DCB "Om Namah Shivaya", 0x0D

ALIGN

END

**Provide a suitable table to trace the Register values for each of the iteration.**

A) The above program calculates the length of the string

|  |  |  |  |
| --- | --- | --- | --- |
| R0 | R1 | R2 | R3 |
| 0x00000024 | 0 | 0 | 0x0D |
| 0x00000025 | 4F | 1 | ,, |
| 0x00000026 | 6D | 2 | ,, |
| 0x00000027 | 20 | 3 | ,, |
| 0x00000028 | 4E | 4 | ,, |
| 0x00000029 | 61 | 5 | ,, |
| 0x0000002A | 6D | 6 | ,, |
| 0x0000002B | 61 | 7 | ,, |
| 0x0000002C | 68 | 8 | ,, |
| 0x0000002D | 20 | 9 | ,, |
| 0x0000002E | 53 | 10 | ,, |
| 0x0000002F | 68 | 11 | ,, |
| 0x00000030 | 69 | 12 | ,, |
| 0x00000031 | 76 | 13 | ,, |
| 0x00000032 | 61 | 14 | ,, |
| 0x00000033 | 79 | 15 | ,, |
| 0x00000034 | 61 | 0A | ,, |
| 0x00000035 | 0D | 0A | ,, |

**Part 4: Do It Yourself Assembly Programming**

1. Write a simple assembly program to evaluate (A+8B+7C-27)/4. Try to reuse the registers there by minimizing the number of registers.

a)

AREA pgm3, CODE

ENTRY

;A=r0,B-r1,C-r2

MOV R0,#1

MOV R1,#2

MOV R2,#4

MOV R1,R1,LSL #3

ADD R0,R0,R1

RSB R2,R2,R2,LSL #3

ADD R0,R0,R2

SUB R0,R0,#27

MOV R0,R0,LSR #2

done B done

END

1. Write the assembly code for the equivalent C snippet

if(a>b)

a=a-b;

else

a=a\*b

a)

MOV r0,#1

MOV r1,#2

CMP r0, r1

BLE Label

SUB r0, r0, r1

B Exit

Label: MUL r0,r1

B Exit

1. Write the assembly program for the following C code snippet. Please use SUBS instruction to see the change in condition codes during the execution of each and every statement. Also observe the change happening during the execution of the instruction BNE (Branch Not equal). Also, briefly explain the execution process.

|  |
| --- |
| int total, i;  total=0;  for(i=20;i>=0;i--){  total +=i;  } |

a) MOV r1, #20 ; Assign 20 to r1

MOV r2, #0 ;Assign 0 to r0

Loop: ADD r2, #1

SUBS r1,#1

CMP r1, #0

BGT Loop;if r1 > 0,continue Loop

Stop: B Stop

1. Write a program to find the largest of N Numbers

HINT : BHS is an instruction which gives unsigned comparison and Set C=1 when higher or same

A)

AREA pgm3, CODE

ENTRY

MAIN

ldr r0,=data1

ldr r4,=0x05

ldr r1,[r0],#4

sub r4,r4,#1

BACK ldr r2,[r0]

cmp r1,r2

bhs BIG

mov r1,r2

BIG add r0,r0,#4

sub r4,r4,#1

cmp r4,#0

bne BACK

;str r1,[r3]

;AREA data , DATA

data1 dcd &65,&05,&96,&01,&65

END