Lab Assignment 1

Mohit Maurya (22110145)

Parag Sarvoday Sahu (22110179)

Wherever specified in the table heading in the format 0x... are the memory addresses

Q1:

(a)

```
.global _start
_start:

 mov r2, #8
 mov r3, #6

 adds r4, r2, r3
 subs r5, r3, r2
 muls r6, r2, r3

 b exit

exit:
 B exit
```

Register, Memory address values that were affected by the program:

PC	R2	R3	R4	R5	R6	N	Z	C	V
00000018	00000008	00000006	0000000e	fffffffe	00000030	0	0	0	0

(b)

```
.global _start
_start:

 ldr r2, =#0x00001260
 ldr r3, =#0x12345678
 str r3, [r2]

 b exit

exit:
 B exit
```

Register, Memory address values that were affected by the program:

PC	R2	R3	0x00001260	N	Z	C	V
00000010	00001260	12345678	12345678	0	0	0	0

Q2:

Code:

```
Loop: ADD r2,r2,#1    // R2 is a iterator
ADD R3,R3,R2          // R3 is the final anwer
CMP R2,#5             // Compare the iterator value
BEQ EXIT              // If Equal to 5 break out of loop otherwise
continue
B Loop
EXIT:
B EXIT              // prevent the increment of PC
```

Register values that were affected by the program:

1st iteration:

PC	R2	R3	CPSR	N	Z	C	V
00000000	00000000	00000000	000001d3	0	0	0	0
00000004	00000001	00000000	000001d3	0	0	0	0
00000008	00000001	00000001	000001d3	0	0	0	0
0000000c	00000001	00000001	800001d3	1	0	0	0
00000010	00000001	00000001	800001d3	1	0	0	0

Final result:

PC	R2	R3	CPSR	N	Z	C	V
00000014	00000005	0000000f	600001d3	0	1	1	0

Short explanation:

- R2 is like an iterator whose value is increased by 1 every time the loop runs
- R3 is the register where the final summation is stored
- Initial N flag is 1 because of CMP (compare) operation as CMP operation performs a subtraction. So, when the final subtraction is negative, the N flag becomes 1
- When the value of R2 is finally 5, the Z (zero flag) becomes 1 as the result is 0. The carry flag becomes 1 as Borrow = ~ (carry)

Q3:

Common code to load values to the registers:

```
.global _start
_start:
 mov r1, #0x00000011
 mov r2, #0x00000022
 mov r4, #0x00000033
 mov r5, #0x00000055
 ldr r6, =0x10000000
```

- ADDS R0, R1, R2

PC	R0	R1	R2	R4	R5	R6	N	Z	C	V
00000018	00000033	00000011	00000022	00000033	00000055	10000000	0	0	0	0

- MOVNE R3, R0

PC	R0	R1	R2	R3	R4	R5	R6	N	Z	C	V
00000018	00000000	00000011	00000022	00000000	00000033	00000055	10000000	0	0	0	0

- ORREQ R0, R4, R5

PC	R0	R1	R2	R4	R5	R6	N	Z	C	V
00000018	00000033	00000011	00000022	00000033	00000055	10000000	0	0	0	0

- LSR R3, R6, #7

PC	R0	R1	R2	R3	R4	R5	R6	N	Z	C	V
00000018	00000033	00000011	00000022	00200000	00000033	00000055	10000000	0	0	0	0

- SBC R0, R4, R5

PC	R0	R1	R2	R4	R5	R6	N	Z	C	V
00000018	ffffffdd	00000011	00000022	00000033	00000055	10000000	0	0	0	0

- MOVW R5, 0x5422

PC	R0	R1	R2	R4	R5	R6	N	Z	C	V
00000018	00000033	00000011	00000022	00000033	00005422	10000000	0	0	0	0

- MOVT R6, 0x524

PC	R0	R1	R2	R4	R5	R6	N	Z	C	V
00000018	00000033	00000011	00000022	00000033	05240055	10000000	0	0	0	0

Q4:

Code 1:

```
mov r0,#0x1b     // Moving 8bit value to r0 register
mov r1,#0x1d    // Moving 8bit value to r1 register
mov r2,#0x00000020  // Assigning the address of memory
strb r0,[r2],#4              // Storing the value in memory and
//incrementing address
strb r1,[r2],#4             // Storing the value in memory and
//incrementing address
```

Register, Memory address values that were affected by the program:

PC	r0	r1	r2	CPSR	N	Z	C	V	0x00000020	0x00000024
00000000	00000000	00000000	00000000	000001d3	0	0	0	0	aaaaaaaa	aaaaaaaa
00000004	0000001b	00000000	00000000	000001d3	0	0	0	0	aaaaaaaa	aaaaaaaa
00000008	0000001b	0000001d	00000000	000001d3	0	0	0	0	aaaaaaaa	aaaaaaaa
0000000c	0000001b	0000001d	00000020	800001d3	0	0	0	0	aaaaaaaa	aaaaaaaa
00000010	0000001b	0000001d	00000024	800001d3	0	0	0	0	aaaaaa1b	aaaaaaaa
00000014	0000001b	0000001d	00000028	800001d3	0	0	0	0	aaaaaa1b	aaaaaa1d

Code 2:

```
mov r0,#0x20       // Load 8 bit value in r0
mov r1,#0x3D  // Load 8 bit value in r0
push {r0,r1}      // push into stack
mov r0,#0xf// overwrite the value of r0 and r1
mov r1,#0xf
pop {r0,r1}     // pop back from stack
add r2,r1,r0     // perform addition of ro and r1 and store into r2
ldr r3,=0x000000f0   // load the memory address in which result is to
//be stored
str r2,[r3]   store the result in the specified address
```

Register, Memory address values that got affected by the program:

PC	SP	r0	r1	r2	r3	N	Z	C	V	0x000000f0	0x000000f4
00000000	00000000	00000000	00000000	00000000	00000000	0	0	0	0	aaaaaaaa	aaaaaaaa
00000004	00000000	00000020	00000000	00000000	00000000	0	0	0	0	aaaaaaaa	aaaaaaaa
00000008	00000000	00000020	0000003d	00000000	00000000	0	0	0	0	aaaaaaaa	aaaaaaaa
0000000c	fffffff8	00000020	0000003d	00000000	00000000	0	0	0	0	00000020	0000003D
00000010	fffffff8	0000000f	0000003d	00000000	00000000	0	0	0	0	00000020	0000003D
00000014	fffffff8	0000000f	0000000f	00000000	00000000	0	0	0	0	00000020	0000003D
00000018	fffffff0	00000020	0000003d	00000000	00000000	0	0	0	0	00000020	0000003D
0000001c	fffffff0	00000020	0000003d	0000005d	00000000	0	0	0	0	00000020	0000003D
00000020	fffffff0	00000020	0000003d	0000005d	000000f0	0	0	0	0	00000020	0000003D
00000024	fffffff0	00000020	0000003d	0000005d	000000f0	0	0	0	0	00000020	0000003D

Q5:

(a)

Code:

```
.global _start
_start:

 ldr r1, =0x00000040  @Load the address of memory
 mov r2, #0x79        @Load the given numbers in various registers
 mov r3, #0x58
 mov r4, #0x1D
 mov r5, #0x43
 mov r6, #0x64

 strb r2, [r1]       @Use of strb, 1-byte number needs to be stored
 strb r3, [r1, #1]   @Increment the memory address by 1 byte
 strb r4, [r1, #2]   @Increment the memory address by (2*1)byte
 strb r5, [r1, #3]   @Increment the memory address by (3*1)byte
 strb r6, [r1, #4]   @Increment the memory address by (4*1)byte
```

Register, Memory address values that got affected by the program:

PC	R1	R2	R3	R4	R5	R6	0x00000040	0x00000044
00000018	00000040	00000079	00000058	0000001d	00000043	00000064	aaaaaaaa	aaaaaaaa
0000001c	00000040	00000079	00000058	0000001d	00000043	00000064	aaaaaa79	aaaaaaaa
00000020	00000040	00000079	00000058	0000001d	00000043	00000064	aaaa5879	aaaaaaaa
00000024	00000040	00000079	00000058	0000001d	00000043	00000064	aa1d5879	aaaaaaaa
00000028	00000040	00000079	00000058	0000001d	00000043	00000064	431d5879	aaaaaaaa
0000002c	00000040	00000079	00000058	0000001d	00000043	00000064	431d5879	aaaaaa64

(b):

Code:

```
.global _start
_start:

// Define the memory address of the data
ldr r10, =0xfffff200

// Load the data
mov r0,#0x1d
str r0, [r10]

mov r0,#0x43
str r0, [r10, #4]!

mov r0,#0x58
str r0, [r10, #4]!
```

```
mov r0,#0x64
str r0, [r10, #4]!

mov r0,#0x79
str r0, [r10, #4]!

ldr r10, =0xfffff200

//temp variable to load the data
mov r1,#0
mov r2,#0

// iterator
mov r6,#0    // i


LOOP:  // loop for i
CMP r6,#5   // Condition to break out of outer loop
beq exit
mov r7,#0x0    // j
mov r8, #0x4
sub r8, r8, r6
loop:  //loop for j

cmp r7, r8  // Condition to break out of inner loop
beq exit1
lsl r7,r7,#2
mov r11,r10
add r11,r11,r7
ldr r1,[r11]      // loading the data for comparison
ldr r2,[r11,#4]
CMP r1,r2
strmi r1,[r11,#4]    // sorting in the memory address
strmi r2,[r11]
lsr r7,r7,#2
add r7,r7,#0x01   // incrementing j
b loop
exit1:
add r6,r6,#0x01   // incrementing i
b LOOP
exit:   // end
```

Register, Memory address Value that got affected by the program:

1st Iteration

PC	R6 (i)	R7 (j)	fffff200	fffff204	fffff208	fffff20c	fffff210	N	Z	C	V
00000000	00000000	00000000	aaaaaaaa	aaaaaaaa	aaaaaaaa	aaaaaaaa	aaaaaaaa	0	0	0	0
0000007c	00000000	00000000	00000079	00000058	0000001d	00000043	00000064	0	0	1	0
0000007c	00000000	00000001	00000079	00000058	0000001d	00000043	00000064	0	0	1	0
0000007c	00000000	00000002	00000079	00000058	00000043	0000001d	00000064	0	0	1	0
0000007c	00000000	00000003	00000079	00000058	00000043	00000064	0000001d	0	0	1	0
00000054	00000000	00000003	00000079	00000058	00000043	00000064	0000001d	0	1	1	0
00000088	00000001	00000000	00000079	00000058	00000043	00000064	0000001d	0	1	1	0

Final result

PC	R6 (i)	R7 (j)	fffff200	fffff204	fffff208	fffff20c	fffff210	N	Z	C	V
0000008c	00000005	00000000	00000079	00000064	00000058	00000043	0000001d	0	1	1	0