

Problem 1

Take any number a in your mind and move it to R1 register. Add 57 with it. Then multiply (unsigned) with 100. If the result is (signed) greater than 0x1F40, store 1 to mem[8], otherwise store 0 to mem[9]. If you assume $a > 23$, mem[25] should be 1 and mem[26] should be 0. Else mem[25] should be 0 and mem[26] should be 1.

AREA MYCODE, CODE, READONLY

; Write Your Code Here

LABEL B LABEL; Program gets stuck here

END; End of the program

Solution:

The screenshot shows the Keil uVision IDE with the assembly code for 'tt1.s' and the register window. The register window on the left shows the current state of the registers. The assembly code on the right is as follows:

```
1  AREA MYCODE, CODE, READONLY
2  MOV R1, #28
3  MOV R3, #0
4  MOV R4, #1
5  ADD R1, R1, #57
6  MOV R0, #100
7  UMULL R6, R5, R1, R0
8  CMP R5, R3 ;testing if R5 has anything
9  BEQ COMPARISONONE ;if yes, then its surely bigger
10 B STOREONE
11
12 COMPARISONONE
13 CMP R6, #0x1F40
14 BGT STOREONE
15 B STOREZERO
16 STOREONE
17 STR R4, [R3, #32];8*4
18 B COMPARISONTWO
19 STOREZERO
20 STR R3, [R3, #36];9*4
21
22 COMPARISONTWO
23 CMP R2, #23
24 BGT BIGGER
25 B SMALLER
26 BIGGER
27 STR R4, [R3, #100];25*4
28 STR R3, [R3, #104];26*4
29 B DONE
30 SMALLER
31 STR R3, [R3, #100];25*4
32 STR R4, [R3, #104];26*4
33 DONE
34 END
```

The register window on the left shows the following values:

Register	Value
R0	0x00000064
R1	0x00000055
R2	0x00000000
R3	0x00000000
R4	0x00000001
R5	0x00000000
R6	0x00002134
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000
R15 (PC)	0x00000034
CPSR	0x200000D3
SPSR	0x00000000

Memory 1		Memory 1	
Address: 32		Address: 100	
0x00000020: 01 00 00 00		0x00000064: 00 00 00 00 01 00 00 00	
		0x0000007E: 00 00 00 00 00 00 00 00	

Problem 2

Write a code in Arm Assembly language that will store an integer value in register R0. If the value is positive, it will store 1 in R1, if negative then it will store 2 in R1, otherwise R1 will have 0 value.

Solution:

The screenshot displays an ARM assembly simulator interface. On the left, a 'Register' table shows the current state of various registers. On the right, the assembly code for a program named 'e6t2.s' is shown, with the current instruction highlighted.

Register	Value
Current	
R0	0x00000005
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)	0x00000030
CPSR	0x200000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000030
Mode	Supervisor
States	11
Sec	0.00000018

Address	Hex	Dec	Op	Op2
0x00000030	00000000	ANDEQ	R0, R0, R0	
0x00000034	00000000	ANDEQ	R0, R0, R0	
0x00000038	00000000	ANDEQ	R0, R0, R0	
0x0000003C	00000000	ANDEQ	R0, R0, R0	
0x00000040	00000000	ANDEQ	R0, R0, R0	

e6t2.s

```
1  AREA MYCODE, CODE, READONLY
2  MOV R0, #5
3  MOV R2, #0
4  CMP R0, R2
5  BEQ ZEROO
6  BGT ONEE
7  B TWOO
8  ZEROO
9  MOV R1, #0
10 B DONEE
11 ONEE
12 MOV R1, #1
13 B DONEE
14 TWOO
15 MOV R1, #2
16 B DONEE
17 DONEE
18 END
```

Register	Value
Current	
R0	0xFFFFFFFF
R1	0x00000002
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)	0x00000030
CPSR	0xA00000D3
SPSR	0x00000000
+ User/System	
+ Fast Interrupt	

0x00000030 00000000 ANDEO R0,R0,R0	
e6t2.s	
1	AREA MYCODE, CODE, READONLY
2	MOV R0, #-5
3	MOV R2, #0
4	CMP R0, R2
5	BEQ ZEROO
6	BGT ONEE
7	B TWOO
8	ZEROO
9	MOV R1, #0
10	B DONEE
11	ONEE
12	MOV R1, #1
13	B DONEE
14	TWOO
15	MOV R1, #2
16	B DONEE
17	DONEE
18	END

Register	Value
Current	
R0	0x00000000
R1	0x00000000
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)	0x00000000
CPSR	0x000000D3
SPSR	0x00000000
+ User/System	
+ Fast Interrupt	

2: MOV R0, #0	
e6t2.s	
1	AREA MYCODE, CODE, READONLY
2	MOV R0, #0
3	MOV R2, #0
4	CMP R0, R2
5	BEQ ZEROO
6	BGT ONEE
7	B TWOO
8	ZEROO
9	MOV R1, #0
10	B DONEE
11	ONEE
12	MOV R1, #1
13	B DONEE
14	TWOO
15	MOV R1, #2
16	B DONEE
17	DONEE
18	END

Problem 3

Write an Assembly Language Code to compute the factorial of a given integer n.

Solution

Register	Value
Current	
R0	0x00000005
R1	0x00000006
R2	0x00000078
R3	0x00000078
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000

```
1  AREA MYCODE, CODE, READONLY
2  MOV R0,#5 ;R0 IS OUR n
3  MOV R1,#1; R1 IS OUR i
4  MOV R2,#1; R2 IS OUR result
5  FACTORIAL ;R3=R2*R1 AND THEN R1+1
6  MUL R3,R2,R1
7  MOV R2,R3
8  ADD R1,R1,#1
9  CMP R1, R0
10 BLS FACTORIAL
11 B DONEE
12 DONEE
13 END
```

Register	Value
Current	
R0	0x00000000
R1	0x00000002
R2	0x00000001
R3	0x00000001
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x00000000

```
1  AREA MYCODE, CODE, READONLY
2  MOV R0,#0 ;R0 IS OUR n
3  MOV R1,#1; R1 IS OUR i
4  MOV R2,#1; R2 IS OUR result
5  FACTORIAL ;R3=R2*R1 AND THEN R1+1
6  MUL R3,R2,R1
7  MOV R2,R3
8  ADD R1,R1,#1
9  CMP R1, R0
10 BLS FACTORIAL
11 B DONEE
12 DONEE
13 END
```


Problem 4

Write an assembly program to store all natural numbers up to 20 in an array in ascending order. Then copy the numbers to another array so the numbers appear in descending order.

Solution

The screenshot displays an ARM assembly debugger interface. On the left, a register window shows the current state of registers R0 through R15, CPSR, and SPSR. On the right, the assembly code for a program named 'r6 t4.s' is shown, with line numbers 1 through 25.

Register	Value
R0	0x0000A00
R1	0x0000015
R2	0xFFFFF0C
R3	0x0000001
R4	0x0000050
R5	0x0000F00
R6	0x0000000
R7	0x0000000
R8	0x0000000
R9	0x0000000
R10	0x0000000
R11	0x0000000
R12	0x0000000
R13 (SP)	0x0000000
R14 (LR)	0x0000000
R15 (PC)	0x0000048
CPSR	0xA00000D3
SPSR	0x00000000
User/System	
Fast Interrupt	
Interrupt	
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x0000048
Mode	Supervisor
States	422
Sec	0.00000703

```
1  AREA MYCODE, CODE, READWRITE
2  MOV R0, #0x0000A00 ;base
3  MOV R2, #0 ;OUR i
4  MOV R1, #1
5  STORINGA
6  STR R1, [R0,R2];STORE
7  ADD R1, R1,#1
8  CMP R1, #20
9  BGT STOREDA
10 ADD R2, R2,#4
11 B STORINGA
12 STOREDA
13 MOV R5, #0x0000F00
14 MOV R4, #0 ;OUR j
15 STORINGD
16 LDR R3, [R0,R2]
17 STR R3, [R5,R4]
18 ADD R4, R4,#4
19 SUB R2, R2,#4
20 CMP R2, #0
21 BLT STOREDD
22 B STORINGD
23 STOREDD
24 END
25
```

Memory 1	
Address:	0x0000A00
0x0000A00:	01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 06 00 00 00 07 00 00
0x0000A1B:	00 08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00 0C 00 00 00 0D 00 00 00 0E 00
0x0000A36:	00 00 0F 00 00 00 10 00 00 00 11 00 00 00 12 00 00 00 13 00 00 00 14 00 00 00
0x0000A51:	00 00

Memory 1	
Address:	0x0000F00
0x0000F00:	14 00 00 00 13 00 00 00 12 00 00 00 11 00 00 00 10 00 00 00 0F 00 00 00 0E 00 00
0x0000F1B:	00 0D 00 00 00 0C 00 00 00 0B 00 00 00 0A 00 00 00 09 00 00 00 08 00 00 00 07 00
0x0000F36:	00 00 06 00 00 00 05 00 00 00 04 00 00 00 03 00 00 00 02 00 00 00 01 00 00 00
0x0000F51:	00 00

Problem 5

Write an assembly code to swap two register values using stack.

Solution:

Register	Value
Current	
R0	0x00000008
R1	0x00000005
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000

r6 t5.s	
1	AREA MYCODE, CODE, READONLY
2	MOV R0, #5
3	MOV R1, #8
4	PUSH{R0}
5	PUSH{R1}
6	POP{R0}
7	POP{R1}
8	END

Problem 6

The following snippet shows an example of non-leaf function calling, where function *f1* calls another function *f2*. However, the idea is the similar, we just need to differentiate which registers are preserved and non-preserved with respect to the caller *f1* and callee *f2*, and save the non-preserved registers on the stack before we call *f2* inside of *f1* and then restore those registers afterward. Implement the following C code snippet in ARM assembly language.

```
int main() {
    int y;
    y = f1(1,2);
}
int f1(int a, int b) {
    int i, x;
    x = (a + b)*(a - b);
    for (i=0; i<a; i++) x = x + f2(b+i);

    return x;
}
int f2(int p) {
    int r;
    r = p + 5;
    return r + p;
}
```

Solution:

Register	Value
Current	
R0	0x00000006
R1	0x00000001
R2	0x00000002
R3	0x00000002
R4	0x00000006
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x00000000
R14 (LR)	0x0000000C
R15 (PC)	0x00000064
CPSR	0x600000D3
SPSR	0x00000000
Supervisor	
Abort	
Undefined	
Internal	
PC \$	0x00000064
Mode	Supervisor
States	52
Sec	0.00000087


```

1  AREA MYCODE, CODE
2  MAIN
3      MOV R1, #1 ;a
4      MOV R2, #2 ;b
5      BL FONE
6      MOV R4,R0 ;R4 IS RESULT
7      B MAINDONE
8
9  FONE
10     PUSH {LR}
11     PUSH {R4,R5,R6}
12     ADD R4,R1,R2 ;TEMP
13     SUB R5,R1,R2 ;TEMP
14     MUL R6,R4,R5 ;R6 IS x NOW
15     MOV R4,#0 ;R4 IS i NOW
16  LOOP
17     CMP R4,R1
18     BGE LOOPDONE
19     ADD R3,R2,R4 ;R3 IS ARG FOR F2
20     BL FTWO
21     ADD R6,R6,R0
22     ADD R4,R4,#1
23     B LOOP
24  LOOPDONE
25     MOV R0,R6
26     POP {R4,R5,R6}
27     POP {LR}
28     MOV PC,LR
29
30  FTWO
31     ADD R0,R3,#5
32     ADD R0,R0,R3
33     MOV PC,LR
34  MAINDONE
35  END
  
```


Implement the following C code snippet in ARM assembly language.

// C code

```
void setArray(int num) {
    int i;
    int array[10];
    for (i = 0; i < 10; i = i + 1)
        array[i] = compare(num, i);
}

int compare(int a, int b) {
    if (sub(a, b) >= 0)
        return 1;
    else
        return 0;
}

int sub(int a, int b) {
    return a - b;
}
```

Solution:

Register	Value		
Current			
R0	0x00000000	1	AREA MYCODE, CODE
R1	0x00000005	2	MOV R1, #5 ;OUR num
R2	0x0000000A	3	BL SETARRAY
R3	0x00000000	4	B MAINDONE
R4	0x00000000	5	SETARRAY
R5	0x00000000	6	PUSH {LR}
R6	0x00000000	7	PUSH {R4, R5}
R7	0x00000000	8	MOV R4, #0x0000A000 ;base
R8	0x00000000	9	MOV R2, #0 ;OUR i
R9	0x00000000	10	LOOP
R10	0x00000000	11	CMP R2, #10
R11	0x00000000	12	BGE LOOPDONE
R12	0x00000000	13	BL COMPARE ;a=R1,b=R2
R13 (SP)	0x00000000	14	LSL R5, R2, #2
R14 (LR)	0x00000008	15	STR R0, [R5, R4]
R15 (PC)	0x00000074	16	ADD R2, R2, #1
CPSR	0x600000D3	17	B LOOP

Address	Disassembly	Comment
0x00000000	SPSR	
0x00000004	PC \$	
0x00000008	User/System	
0x0000000C	Fast Interrupt	
0x00000010	Interrupt	
0x00000014	Supervisor	
0x00000018	Abort	
0x0000001C	Undefined	
0x00000020	Internal	
0x00000074	PC \$	
0x00000078	Mode	Supervisor
0x0000007C	States	366
0x00000080	Sec	0.00000610
18	LOOPDONE	
19	POP {R4,R5}	
20	POP {LR}	
21	MOV PC,LR	
22	COMPARE	
23	PUSH {LR}	
24	BL SUBB	
25	CMP R0,#0	
26	BLT ZERO	
27	B ONE	
28	ZERO	
29	MOV R0,#0	
30	B COMPAREDONE	
31	ONE	
32	MOV R0,#1	
33	COMPAREDONE	
34	POP {LR}	
35	MOV PC,LR	
36	SUBB	
37	SUB R0,R1,R2	
38	MOV PC,LR	
39		
40	MAINDONE	
41	END	

```

18      LOOPDONE
19          POP {R4,R5}
20          POP {LR}
21          MOV PC,LR
22      COMPARE
23          PUSH {LR}
24          BL SUBB
25          CMP R0,#0
26          BLT ZERO
27          B ONE
28      ZERO
29          MOV R0,#0
30          B COMPAREDONE
31      ONE
32          MOV R0,#1
33      COMPAREDONE
34          POP {LR}
35          MOV PC,LR
36      SUBB
37          SUB R0,R1,R2
38          MOV PC,LR
39
40      MAINDONE
41          END

```

[illegible]