

Q1. Explain briefly the data processing instructions for ARM processor.

Answer:

The data processing instructions manipulate data within registers. They are move instructions, arithmetic instructions, logical instructions, comparison instructions and multiply instructions.

Most data processing instructions can process one of their operands using the barrel shifter.

If S is suffixed on a data processing instruction, then it updates the flags in the cpsr.

Move instructions:

It copies N into a destination register Rd, where N is a register or immediate value.

Syntax: <instruction> {<cond>} {S} Rd,N

MOV	Move a 32-bit value into a register	Rd=N
MVN	Move the NOT of the 32-bit value into a register	Rd= ~N

In the example shown below, the MOV instruction takes the contents of register r5 and copies them into register r7.

PRE r5=5

r7=8

MOV r7,r5

POST r5=5

r7=5

Barrel shifter: Data processing instructions are processed within the arithmetic and logic unit(ALU). A unique and powerful feature of the ARM processor is the ability to shift the 32-bit binary pattern in one of the source registers left or right by a specific number of positions before it enters the ALU. This shift increases the power and flexibility of many data processing operations.

For example, We apply a logical shift left (LSL) to register Rm before moving it to the destination register.

PRE r5=5

r7=8

MOV r7,r5,LSL #2

POST r5=5

r7=20

The above example shift logical left r5=5 (00000101 in binary) by two bits and then r7=20 (00010100 in binary).

Following table shows barrel shifter operation

Mnemonics	Description	Shift	Result
LSL	Logical shift left	xLSLy	$x \ll y$
LSR	Logical shift right	xLSRy	$(\text{unsigned}) x \gg y$
ASR	Arithmetic right shift	xASRy	$(\text{signed}) x \gg y$
ROR	Rotate right	xRORy	$((\text{unsigned})x \gg y) (x \ll (32-y))$
RRX	Rotate right extended	xRRX	$(c \text{ flag} \ll 31) ((\text{unsigned}) x \gg 1)$

Arithmetic instruction: The arithmetic instructions implement addition and subtraction of 32-bit signed and unsigned values.

Syntax: <instruction>{<cond>} {S} Rd, Rn,N

ADC	Add two 32-bit values and carry	Rd=Rn+N+carry
ADD	Add two 32-bit values	Rd=Rn+N
RSB	Reverse subtract of two 32-bits values	Rd=N-Rd
RSC	Reverse subtract with carry of two 32-bits values	Rd=N-Rd-!(carry flag)
SBC	Subtract with carry of two 32-bits values	Rd=Rn-N-!(carry flag)
SUB	Subtract two 32-bit values	Rd=Rn-N

In the following example, subtract instruction subtracts a value stored in register r2 from a value stored in the register r1. The result is stored in register r0.

```
PRE  r0=0x00000000
      r1=0x00000002
      r2=0x00000001
      SUB r0,r1,r2
```

```
POST r0=0x00000001
```

In the following example, the reverse subtract instruction (RSB) subtract r1 from the constant value #0, writing the result in r0.

```
PRE  r0=0x00000000
      r1=0x00000077
      RSB r0,r1,#0 ; Rd=0x0-r1
```

```
POST r0=-r1=0xfffff89
```

Using the barrel shifter with arithmetic instructions: Example below illustrate the use of the inline barrel shifter with an arithmetic instruction. The instruction multiplies the value stored in register r1 by three.

```
PRE  r0=0x00000000
      r1=0x00000005
      ADD r0, r1, r1, LSL #1
```

```
POST r0=0x0000000f
      R1=0x00000005
```

Logical instructions: logical instructions perform bitwise operations on the two source registers.

Syntax: <instruction> {<cond>} {S} Rd, Rn, N

AND	Logical bitwise AND of two 32 bits values	Rd=Rn & N
ORR	Logical bitwise OR of two 32-bits values	Rd= Rn N
EOR	Logical exclusive OR of two 32-bits values	Rd=Rn^N
BIC	Logical bit clear (AND NOT)	Rd= Rn^~N

In the example shown below, a logical OR operation between registers r1 and r2 and the result is in r0.

```
PRE  r0=0x00000000
```

r1=0x02040608

r2=0x10305070

ORR r0, r1, r2

POST r0=0x12345678

Comparison instructions: The comparison instructions are used to compare or test a register with a 32-bit value. They update the cpsr flag bits according to the result, but do not affect other registers. After the bits have been set, the information can be used to change program flow by using conditional execution.

Syntax: <instruction> {<cond>} Rn, N

CMN	Compare negated	Flags set as result of Rn+N
CMP	Compare	Flags set as result of Rn-N
TEQ	Test for equality of two 32-bit values	Flags set as result of Rn^N
TST	Test bits of a 32-bit value	Flags set as result of Rn&N

Example shown below for CMP instruction. Both r0 and r1 are equal before the execution of the instruction. The value of the z flag prior to the execution is 0 and after the execution z flag changes to 1 (upper case of Z).

PRE cpsr=nzcvqiFt_USER

r0=4

r1=4

CMP r0, r1

POST cpsr=nZcvqiFt_USER

The CMP is effectively a subtract instruction with the result discarded; similarly the TST instruction is a logical AND operation and TEQ is a logical exclusive OR operation. For each, the results are discarded but the condition bits are updated in the cpsr.

Multiply instructions: The multiply instructions multiply the contents of a pair of registers and depending upon the instruction, accumulate the results in another register. The long multiplies accumulate onto a pair of registers representing a 64-bit value.

Syntax: MLA {<cond>} {S} Rd, Rm, Rs, Rn

MUL {<cond>} {S} Rd,Rm,Rs

MLA	Multiply and accumulate	Rd=(Rm*Rs)+Rn
MUL	Multiply	Rd=Rm*Rs

Syntax: <instruction> {<cond>} {S} RdLo, RdHi, Rm, Rs

SMLAL	Signed multiply accumulate long	{RdHi, RdLo}={ RdHi, RdLo}+(Rm*Rs)
SMULL	Signed multiply long	{RdHi, RdLo}= Rm*Rs
UMLAL	Unsigned multiply accumulate long	{RdHi, RdLo}={ RdHi, RdLo}+(Rm*Rs)
UMULL	Unsigned multiply long	{RdHi, RdLo}=Rm*Rs

In the following example below shows a multiply instruction that multiplies registers r1 and r2 and places the result into the register r0.

```
PRE  r0=0x00000000
      r1=0x00000002
      r2=0x00000002
      MUL r0,r1,r2 ; r0=r1*r2
POST r0=0x00000004
      r1=0x00000002
      r2=0x00000002
```

The long multiply instructions (SMLAL, SMULL, UMLAL, and UMULL) produce a 64-bit result.

Q2. Explain briefly branch instructions for ARM processor.

Answer: A branch instruction changes the flow of execution or is used to call a routine. This type of instruction allows programs to have subroutines, if-then-else structures, and loops. The change of execution flow forces the program counter (pc) to point to a new address.

Syntax: B{<cond>}label
 BL{<cond>}label
 BX{<cond>}label
 BLX{<cond>}label

B	Branch	pc=label
BL	Branch with link	pc=label lr=address of the next instruction after the BL
BX	Branch exchange	pc=Rm & 0xffffffe, T=Rm & 1
BLX	Branch exchange with link	pc=label, T=1 pc= Rm & 0xffffffe, T=Rm & 1 lr=address of the next instruction after the BLX

The address label is stored in the instruction as signed pc-relative offset and must be within approximately 32MB of the branch instruction. T refers to the Thumb bit in the cpsr. When instruction set T, the ARM switches to Thumb state.

The example shown below is a forward branch. The forward branch skips three instructions.

```
      B      forward
      ADD    r1, r2, #4
      ADD    r0, r6, #2
      ADD    r3, r7, #4
forward
      SUB    r1, r2, #4
```

The branch with link (BL) instruction changes the execution flow in addition overwrites the link register lr with a return address. The example shows below a fragment of code that branches to a subroutine using the BL instruction.

```
      BL      subroutine      ; branch to subroutine
```

```

CMP r1, #5          ; compare r1 with 5
MOVEQ r1, #0        ; if (r1==5) then r1=0

```

Subroutine

```

MOV pc, lr          ; return by moving pc=lr

```

The branch exchange (BX) instruction uses an absolute address stored in register Rm. It is primarily used to branch to and from Thumb code. The T bit in the cpsr is updated by the least significant bit of the branch register.

Similarly, branch exchange with link (BLX) instruction updates the T bit of the cpsr with the least significant bit and additionally sets the link register with the return address.

Q3. Explain briefly the software interrupt instruction.

Answer: A software interrupt instruction (SWI) causes a software interrupt exception, which provides a mechanism for applications to call operating system routines.

Syntax: SWI {<cond>} SWI_number

SWI	Software interrupt	lr_svc=address of instruction following the SWI spsr_svc=cpsr pc=vectors+0x8 cpsr mode= SVC cpsr I=I (mask IRQ interrupt)
-----	--------------------	---

When the processor executes an SWI instruction, it sets the program counter pc to the offset 0xB in the vector table. The instruction also forces the processor mode to SVC, which allows an operating system routine to be called in a privileged mode.

Each SWI instruction has an associated SWI number, which is used to represent a particular function call or feature.

The example below shows an SWI call with SWI number 0x123456, used by ARM toolkits as a debugging SWI.

```

PRE      cpsr=nzcVqift_USER
          pc=0x00008000
          lr=0x003fffff ; lr=r14
          r0=0x12
          0x00008000 SWI 0x123456
POST     cpsr=nzcVqIfT_SVC
          spsr= nzcVqift_USER
          pc=0x000000008
          lr=0x00008004
          r0=0x12

```

Since SWI instructions are used to call operating system routines, it is required some form of parameter passing. This achieved by using registers. In the above example, register r0 is used to pass parameter 0x12. The return values are also passed back via register.

Q4. Explain briefly program status register instructions.

Answer: The ARM instruction set provides two instructions to directly control a program status register (psr). The MRS instruction transfers the contents of either the cpsr or spsr into a register; in the reverse direction, the MSR instruction transfers the contents of a register into the cpsr or spsr. Together these instructions are used to read and write the cpsr and spsr.

Syntax: MRS {<cond>} Rd <cpsr |spsr>

MSR {<cond>} <cpsr|spsr> _<fields>,Rm

MSR {<cond>} <cpsr|spsr> _<fields>, #immediate

The table shows the program status register instructions

MRS	Copy program status register to a general-purpose register	Rd=psr
MSR	Move a general-purpose register to a program status register	psr[field]=Rm
MSR	Move an immediate value to a program status register	psr[field]=immediate

Q5. Explain briefly coprocessor instructions.

Answer: Coprocessor instructions are used to extend the instruction set. A coprocessor can either provide additional computation capability or be used to control the memory subsystem including caches and memory management. The coprocessor instructions include data processing, register transfer, and memory transfer instructions. These instructions are used only by core with a coprocessor.

Syntax: CPD {<cond>} cp,opcode1, Cd, Cn {,opcode2}

<MRC|MCR>{<cond>}cp,opcode1,Rd,Cn,Cm{,opcode2}

<LDC|STC>{<cond>}cp,Cd,addressing

CDP	Coprocessor data processing-perform an operation in a coprocessor
MRC, MCR	Coprocessor register transfer-move data to/from coprocessor registers
LDC,STC	Coprocessor memory transfer-load and store blocks of memory to/from a coprocessor

In the syntax of the coprocessor instructions, the cp field represents the number between p0 and p15. The opcode fields describe the operation to take place on the coprocessor. The Cn, Cm and Cd fields describe registers within the coprocessor.

For example: The instruction below copies CP15 register into a general purpose register.

MPC p15,0,r10,c0,c0,0 ; CP15 register-0 is copied into general purpose register r10.

CP15 is called the system control coprocessor. Both MRC and MCR are used to read and write to CP15 where Rd is the core destination register, Cn is the primary register, Cm is the secondary register and opcode2 is a secondary register modifier.

For example: The instruction below moves the contents of CP15 control register c1 into register r1 of the processor core.

MRC p15,0,r1,c1,c0,0

Q6. Explain briefly the loading constants.

Answer: There are two pseudo instructions to move a 32-bit constant value to a register.

Syntax: LDR Rd,=constant

ADR Rd,label

LDR	Load constant pseudo instruction	Rd=32-bit constant
ADR	Load address pseudo instruction	Rd=32-bit relative address

The example below shows an LDR instruction loading a 32-bit constant 0xff00ffff into register r0.

LDR r0,=0xff00ffff

Q7. If r0=0x00000000, r1=0x80000004

Find the content of the registers r0 and r1 after the following instructions are executed in isolation. Mention if the CPSR register is updated or not.

(i) MOV r0,r1 (ii) MOV r0,r1,LSL #1 (iii) MOVS r0,r1,LSL #1 (iv) MVN r0,r1 (v) MOV r0,r1,LSR #1

Answer:

(i) r0=0x80000004 and r1=0x80000004. CPSR is not updated.

(ii) Before execution of the instruction, r1=0x80000004=1000 0000 0000 0000 0000 0000 0000 0100 in binary.

After 1 bit logical shift left(LSL), the value is 0000 0000 0000 0000 0000 0000 0000 1000 which is copied to r0.

After the execution of the instruction, r0=0x00000008 and r1=0x80000004. CPSR is not updated.

(iii) Before execution of the instruction, r1=0x80000004=1000 0000 0000 0000 0000 0000 0000 0100 in binary.

After 1 bit logical left shift (LSL), the value is 0000 0000 0000 0000 0000 0000 0000 1000 which is copied to r0. Most significant bit (MSB) is copied to the carry flag and carry flag is set. After the execution of the instruction, r0=0x00000008 and r1=0x80000004. CPSR is updated.

(iv) NOT of r1 is 0111 1111 1111 1111 1111 1111 1111 1011=0x7ffffffb is copied to r0.

After the execution, r0=0x7ffffffb and r1=0x80000004. CPSR is not updated.

(v) Before execution of the instruction, r1=0x80000004=1000 0000 0000 0000 0000 0000 0000 0100 in binary.

After 1 bit logical shift right (LSR), the value is 0100 0000 0000 0000 0000 0000 0000 0010 which is copied to r0.

After the execution of the instruction, r0=0x40000002 and r1=0x80000004. CPSR is not updated.

Q8. If r0=0x00000000, r1=0x00000002 and r2=0x00000001

Find the content of the register r0, r1 and r2 after the following instructions are executed in isolation. Mention if the CPSR register is updated or not.

(i) ADD r0,r1,r2,LSL #1 (ii) SUB r0,r1,r2 (iii) RSB r0,r1,#0 (iv) SUBS r1,r1,#2

Answer:

(i) $r2=0x00000001=0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001$. After logic shift left (LSL) it is $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010=0x00000002$ which is added to $r1$ and the result is at $r0$.

$r0=0x00000004$, $r1=0x00000002$ and $r2=0x00000001$. CPSR is not updated.

(ii) $r0=0x00000001$, $r1=0x00000002$ and $r2=0x00000001$. CPSR is not updated.

(iii) $r0=r1$ which is copied to $r0$ as 2's complement of $r1$.

$r1=0x00000002=0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010$

1's complement = $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101$

2's complement = $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110=0xffffffe$

Hence, $r0=0xffffffe$, $r1=0x00000002$ and $r2=0x00000001$. CPSR is not updated.

(iv) $r0=0x00000000$, $r1=0x00000000$ and $r2=0x00000001$. CPSR is updated (Zero flag is set).

Q9. If $r0=0x00000000$, $r1=0x02040608$ and $r2=0x10305070$

Find the content of the register $r0$ after the following instructions are executed in isolation and also mention if the CPSR register updated or not.

(i) AND $r0,r1,r2$ (ii) ORR $r0,r1,r2$ (iii) (iv) BIC $r0,r1,r2$

Answer:

(i) $r1=0x02040608=0000\ 0010\ 0000\ 0100\ 0000\ 0110\ 0000\ 1000$

$r2=0x10305070=0001\ 0000\ 0011\ 0000\ 0101\ 0000\ 0111\ 0000$

Hence, $r0=0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000=0x00000000$

After execution, $r0=0x00000000$, $r1=0x02040608$ and $r2=0x10305070$

(ii) $r1=0x02040608=0000\ 0010\ 0000\ 0100\ 0000\ 0110\ 0000\ 1000$

$r2=0x10305070=0001\ 0000\ 0011\ 0000\ 0101\ 0000\ 0111\ 0000$

Hence, $r0=0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000=0x12345678$

After execution, $r0=0x12345678$, $r1=0x02040608$ and $r2=0x10305070$

(iii) $r1=0x02040608=0000\ 0010\ 0000\ 0100\ 0000\ 0110\ 0000\ 1000$

$r2=0x10305070=0001\ 0000\ 0011\ 0000\ 0101\ 0000\ 0111\ 0000$

Hence, $r0=0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000=0x12345678$

After execution, $r0=0x12345678$, $r1=0x02040608$ and $r2=0x10305070$

(iv) $r2=0x10305070=0001\ 0000\ 0011\ 0000\ 0101\ 0000\ 0111\ 0000$

NOT $r2=1110\ 1111\ 1100\ 1111\ 1010\ 1111\ 1000\ 1111$

$r1=0x02040608=0000\ 0010\ 0000\ 0100\ 0000\ 0110\ 0000\ 1000$

$r0=r1\ \text{AND}\ (\text{NOT}\ r2)=0000\ 0010\ 0000\ 0100\ 0000\ 0110\ 0000\ 1000=0x02040608$

After execution, $r0=0x02040608$, $r1=0x02040608$ and $r2=0x10305070$. CPSR is not updated.

Q10. If $r1=0x00000002$ and $r2=0x00000002$

Find the content of the register $r1$ and $r2$ after the following instructions are executed in isolation and also mention if the CPSR register is updated or not.

(i) CMP $r1,r2$ (ii) CMN $r1,r2$ (iii) TST $r1,r2$ (iv) TEQ $r1,r2$

Answer:

(i) After execution, $r1=0x00000002$ and $r2=0x00000002$. CPSR is updated (Zero flag is set if $r1-r2=0$).

(ii) After execution, $r1=0x00000002$ and $r2=0x00000002$. CPSR is not updated (Zero flag is set if $r1+r2=0$).

(iii) After execution, $r1=0x00000002$ and $r2=0x00000002$. CPSR is updated (Zero flag is set if Exclusive OR of $r1$ and $r2=0$).

(iv) After execution, $r1=0x00000002$ and $r2=0x00000002$. CPSR is not updated (Zero flag is set if AND of $r1$ and $r2=0$).

Q11. If $r0=0x00000000$, $r1=0x00000001$, $r2=0x00000002$ and $r3=0x00000003$

Find the content of the register $r0$, $r1$, $r2$ and $r3$ after the following instructions are executed in isolation. Mention if the CPSR register is updated or not.

(i) MUL $r0, r1, r2$ (ii) MLA $r0, r1, r2, r3$ (iii) UMULL $r0, r1, r2, r3$

Answer:

(i) After execution, $r0=r1*r2$

Hence, after execution, $r0=0x00000002$, $r1=0x00000001$, $r2=0x00000002$ and $r3=0x00000003$

(ii) After execution, $r0=r1*r2+r3$

Hence, after execution, $r0=0x00000005$, $r1=0x00000001$, $r2=0x00000002$ and $r3=0x00000003$

(iii) After execution, $[r0, r1]=r2*r3$ where $r0$ contains lower 32 bits and $r1$ contains higher 32 bits.

Hence, after execution, $r0=0x00000006$, $r1=0x00000000$, $r2=0x00000002$ and $r3=0x00000003$

Q12. If $r0=0x00000000$, $r1=0x00090000$

$\text{mem32}[0x00090000]=0x01010101$ and $\text{mem32}[00090004]=0x02020202$

Find the content of the register $r0$, $r1$ after the following instructions are executed in isolation.

(i) LDR $r0, [r1, \#4]$ (ii) LDR $r0, [r1, \#4]!$ (iii) LDR $r0, [r1], \#4$

Answer:

(i) After the execution, $r0=0x02020202$ and $r1=0x00090000$

(ii) After the execution, $r0=0x02020202$ and $r1=0x00090004$

(iii) After the execution, $r0=0x01010101$ and $r1=0x00090004$

Q13. If $r0=0x00000010$, $r1=0x00000000$, $r2=0x00000000$, $r3=0x00000000$

$\text{mem32}[0x00000010]=0x00000001$ and $\text{mem32}[00000014]=0x00000002$

$\text{mem32}[0x00000018]=0x00000003$ and $\text{mem32}[0000001c]=0x00000004$

Find the content of the register $r0$, $r1, r2$ and $r3$ after the following instructions are executed in isolation.

(i) LDMIA $r0!, \{r1-r3\}$ (ii) LDMIB $r0!, \{r1, r2, r3\}$

Answer:

(i) After the execution, $r0=0x0000001c$, $r1=0x00000001$, $r2=0x00000002$ and $r3=0x00000003$

(ii) After the execution, $r0=0x0000001c$, $r1=0x00000002$, $r2=0x00000003$ and $r3=0x00000004$

Q.14 write ALP program for ARM7 demonstrating the data transfer.

Answer:

```
        AREA DATATRANSFER, CODE, READONLY
        ENTRY
        LDR R0, =0x40000000
        LDR R1, =0x40000080
        LDR R2, =0x0000000A
UP       LDRB R3, [R0]
        STRB R3, [R1]
        ADD R0, R0, #1
        ADD R1, R1, #1
        ADD R2, R2, #-1
        CMP R2, #0
        BEQ NEXT
        B UP
NEXT     MOV R0, #0x18
        LDR R1, =0x20026
        SVC #0x123456
        END
```

Q.15 Write ALP program for ARM7 demonstrating logical operation.

Answer:

```
AREA LOGIC , CODE, READONLY
ENTRY
LDR R0, =0x40000000
LDR R1, =0x40000010
LDRB R2, [R0]
LDRB R3, [R0, #1]
AND R4, R2, R3
STRB R4, [R1]
ORR R4, R2, R3
STRB R4, [R1, #1]
MOV R0, #0x18
LDR R1, =0x20026
SVC #0x123456
END
```

Q.16 Write ALP program for ARM7 demonstrating arithmetic operation.

Answer:

```
AREA MULTIPLY, CODE, READONLY
ENTRY
LDR R0, =0x40000000
LDRB R1, [R0]
LDRB R2, [R0, #1]
ADD R3, R1, R2
STRH R3, [R0, #2]
MUL R4, R1, R2
STRH R4, [R0, #4]
SUB R5, R1, R2
STR R5, [R0, #6]
MOV R0, #0x18
LDR R1, =0x20026
SVC #0x123456
END
```

Q17. Write ALP program for ARM7 to find factorial of a positive number.

Answer:

```
AREA Factorial, CODE, READONLY
ENTRY
LDR R0, =0X40000000
LDRB R1, [R0]
MOV R2, R1
UP    ADD R1, R1, #-1
      CMP R1, #0
      BEQ NEXT
      MUL R3, R2, R1
      MOV R2, R3
      B UP
NEXT  STR R2, [R0, #4]
      MOV R0, #0X18
      LDR R1, =0X20016
      SVC #0123456
      END
```

Q18. Write ALP program for ARM7 to find number of zeros and number of ones in a 32-bit number.

Answer:

```
                AREA CountOneZero, CODE, READONLY
                ENTRY
                LDR R0, =0X40000000
                LDR R1, [R0]
                MOV R2, #32
AGAIN           RORS R1, #1
                BCS ONES
                ADD R3, R3, #1
                B NEXT
ONES           ADD R4, R4, #1
NEXT           ADD R2, R2, #-1
                CMP R2, #0
                BNE AGAIN
                ADD R0, R0, #4
                STRB R3, [R0]
                STRB R4, [R0, #1]
                MOV R0, #0X18
                LDR R1, =0X20026
                SVC #0123456
                END
```

Q19. Write ALP program to add array of 16 bit numbers and store the result in 32 bit memory.

Answer:

```

                AREA  ArrayAddition, CODE, READONLY
                ENTRY
                LDR   R0, =0X40000000
                LDRB  R1, [R0]
                MOV   R5, #0
UP              ADD   R0, R0, #2
                LDRH  R2, [R0]
                ADD   R1, #-1
                ADD   R0, R0, #2
                LDRH  R3, [R0]
                ADD   R1, #-1
                ADD   R4, R2, R3
                ADD   R5, R5, R4
                CMP   R1, #0
                BNE   UP
                STR   R5, [R0, #4]
                MOV   R0, #0X18
                LDR   R1, =0X20026
                SVC   #0123456
                END
```