PROGRAM - I

Logical left shift

```
        AREA program, code, readonly        // assembler directive
                                              req to set up program
entry
        LDR R1, value
        MOV R1, R1, LSL #0X01
        SWI &11
        AREA program, data, readonly
_value  DCD &00000002
        END
```

area refers to the segment of code
'program' is the name given to it

CODE indicates executable code rather than data

READONLY state that it cannot be modified at run time

entry is a LABEL used to refer to that line

END is an assemble directive to assembler - end of program

~~DCD is~~

Program - 2

Logical Right Shift

```
        AREA program, code, readonly
entry
        LDR R1, value
        MOV R1, R1, LSR #0X01
        SWI &11
        AREA program, data, readonly
~~va~~value  DCD &00000004
        END
```

Program - 3

Addition of 2 nos.

```
AREA program, code, readonly
entry
LDR R1, Value 1
LDR R2, value 2
ADD R1, R1, R2
AREA program, data, readonly
Value1 DCD &00000002
Value2 DCD &00000004
END
```

Disassembly Window shows the program execution in assembly code or intermixed with the source code (device dependent)

when Disassembly window ~~shows the~~ is the active window, then all debug-stepping commands work on assembly level.

DW shows the code in memory and converts it into instructions

It shows the actual ARM instructions that are created by your instructions.

DCD - directive allocates one or more words of memory and defines the initial runtime contents of the memory.

Immediate operands

Replace the second source operand with an immediate operand, which is a literal constant, preceded by #

```
ADD R3, R3, #1        ;R3 = R3+1
```

(Immediate value 8 bit number with a range of 0 -255)

Shift Register operands

```
ADD R3, R2, R2, LSL# 3
```

→ LSL logical shift left by 0 to 31 places, 0 filled at the LSB end.

#Program-4 Find complement of a number

```
        AREA program, code, readonly
entry
        LDR R1, value1
        MVN R1,R1
      AREA program, data, readonly
value1 DCD &C123

     END
```

Addition using Barrel Shifter

```
LDR R1, value1
LDR R2, value2
ADD R0, R1 ,R2, LSL #0x02
  SWI &11
value1 DCD &0001000
value2 DCD &00001010
  END
```

```
LDR R1, value1
 LDR R2, value2
MOV R3, R2, LSL#0x0
ADD R4, R1, R3
```

⑤ PRG-5

Program to find difference (subtraction of 2 nos.)

```
        AREA program, code, readonly
entry
        LDR R1, value1
        LDR R2, value2
        SUB R3,R1,R2
        AREA program, data, readonly
value1  DCD  &00000004
value2  DCD  &00000002
        END
```

Program-6 Addition using Indirect Addressing Mode

```
        AREA program , code, readonly
entry
        LDR R0, value1
        LDR R1,[R0]
        LDR R2, value 2
        LDR R3, [R2]
        ADD R4, R1,R3
        AREA program, data, readonly
value1  DCD  & 00000002
value2  DCD  &00000004
        END
```

Condition Code Mnemonics

compare condition	signed	unsigned
greater than or equal	BGE	BSC BHS
greater than	BGT	BHI
equal	BEQ	BEQ
not equal	BNE	BNE
less than or equal	BLE	BLS
less than	BLT	BLO

Find larger of 2 nos.

```
        LDR    R1, value1
        LDR    R2, value2
        CMP    R1, R2
        BHI    Done
        MOV    R1, R2
Done
        STR    R1, Result
        SWI    &11
value1  DCD    &FEDCA987
value2  DCD    &12345678
Result  DCD    0
        END
```

CPSR

31	30	29	28	27 . . . 8	7	6	5	4 0
N	Z	C	V	SBZ	I	F	SBZ	Mode

Program to multiply using addition

```
        AREA  program, code, readonly
entry
        LDR   R0, value1
        LDR   R1, value2
        MOV   R2, ~~R~~#0x01
        MOV   R3, R0
loop
          ADD   R2, #0x01
          ADD   R3, R3, R0
          CMP   R2, R1
          BNE   loop
          SWI   &11
          AREA program, data , readonly
  value1    DCD    &00000005
  value2   DCD  &00000006
        END
```

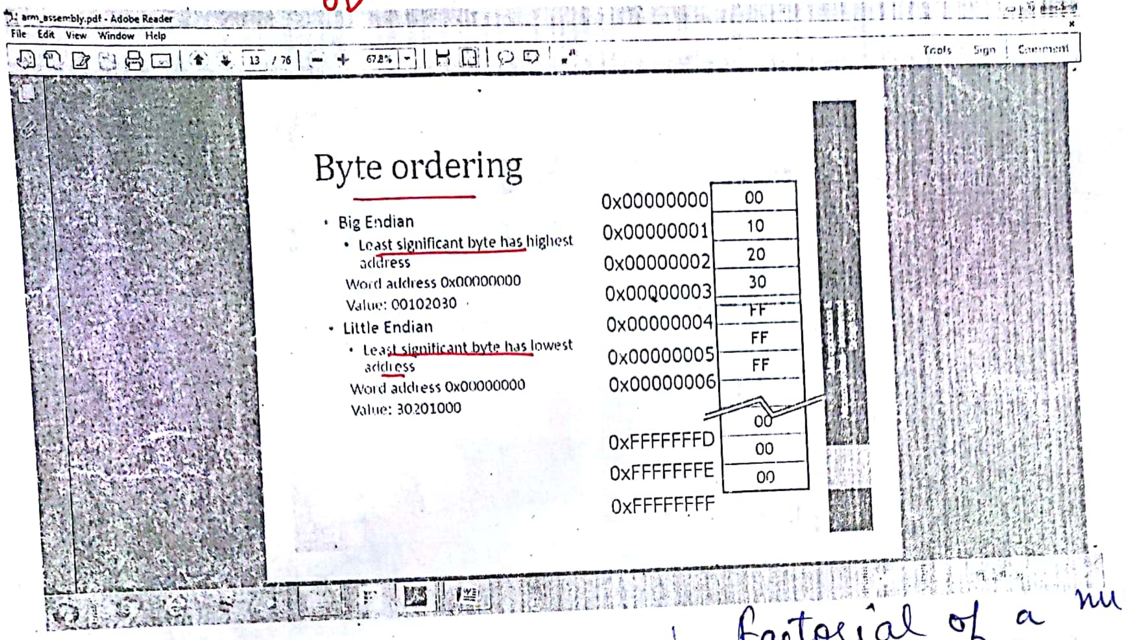
Program 8   Program to ~~perform~~ multiplication table

```
        AREA & program, code, readonly
entry
          LDR  R0, value1
          MOV  R3, R0
          LDR  R1, value2
          MOV    R2, #0X0A
loop
            STR   R0, [R1]
            ADD  R0, R0, R3
            SUB  R2, R2 , #0X01
             ADD    R1,R1 , #0X04
            CMP    R2, # 0X00
            BNE  loop
            SWI  &11
          AREA program, data, readonly
 value1  DCD   &00000005  /    value 2 DCD  &[illegible] / END
```

0A 0B 0C 0D

0A
0B
0C
0D

0D a
0C a+1
0B a+2
0A a+3



Byte ordering

- Big Endian
  - Least significant byte has highest address

  Word address 0x00000000

  Value: 00102030
- Little Endian
  - Least significant byte has lowest address

  Word address 0x00000000

  Value: 30201000

0x00000000	00
0x00000001	10
0x00000002	20
0x00Q00003	30
0x00000004	FF
0x00000005	FF
0x00000006	FF
	00
0xFFFFFFFD	00
0xFFFFFFFE	00
0xFFFFFFFF	

Instruction set
- Data processing (Arithmetic and Logical)
- Data movement
- Flow control

Arithmetic
```
• ADD R0, R1, R2 @ R0 = R1+R2
• ADC R0, R1, R2 @ R0 = R1+R2+C
• SUB R0, R1, R2 @ R0 = R1-R2
• SBC R0, R1, R2 @ R0 = R1-R2+C-1
• RSB R0, R1, R2 @ R0 = R2-R1
• RSC R0, R1, R2 @ R0 = R2-R1+C-1
```

Bitwise logic
```
• AND R0, R1, R2 @ R0 = R1 and R2
• ORR R0, R1, R2 @ R0 = R1 or R2
• EOR R0, R1, R2 @ R0 = R1 xor R2
• BIC R0, R1, R2 @ R0 = R1 and (~R2)
```
bit clear: R2 is a mask identifying which bits of R1 will be cleared to zero

```
R1=0x11111111 R2=0x01100101
BIC R0, R1, R2
R0=0x10011010
```

Register movement
```
• MOV R0, R2 @ R0 = R2
```

$\underline{5 \ \ \cancel{X}1}$

R2=5 R1=5 $\underline{R0=4}$

20

Factorial of a number

```
AREA program, code, readonly
entry
    LDR R0, value1
    MOV R1, #0x01
loop
    MUL R2, R1, R0
    MOV R1, R2
    SUB R0, R0, #0x01
    CMP R0, #0x01
    BNE loop
    SWI &11
    AREA program, data, readonly
value1 DFD &00000003
    END
```

Click Project

· Open New µVision Project

.uvproj

File → new

save as .asm or .s

Project → Manage → Env. Books.

Add files to Source group

PROGRAM 9 Division using subtraction

```
      AREA program, code, readonly
entry LDR  R0, value1
      LDR  R1, value2
      MOV  R2, #0x00
      MOV  R3, R0
loop
      SUB  R3, R3, R1
      ADD  R2, #0x01
      CMP  R3, R1
      BGT  loop
      BEQ  lo
lo
      ADD  R3 #0x01
      SWI  &11

      AREA program, data, readonly
value1    &0000000A
value2    &00000002
```

```
loop SUB R3, R3, R1
     ADD R2, #0x01
     CMP R3, R1
     BGE loop

     SWI &11
```

Condition Code Mnemonics

EQ	- Equal
NE	- Not Equal
GT	- Greater than
GE	- Greater than or equal
LT	- Less than
LE	- Less than or equal
HI	- Higher than
HS	- Higher or Same
LO	- Lower than
LS	- Lower or Same
CS	- Carry Set
PL	- Plus (+ve or zero)
VC	- overflow clear
VS	→ overflow set
MI	→ Minus

Conditional Execution

MOVCS R0, R1

ADDEQ

ADDNE

Offset Addressing

LDR R0, [R1] - no offset is specified.

LDR R0, [R1, #4] - load reg R0 with the word at memory address calculated by adding the constant value 4 to the memory address contained in R1. R1 is not changed by this inst.

LDR R0, [R1,R2] - loads R0 with the value at memory address calculated by adding value at R1 to value held in R2. Both R1 & R2 are not altered.

Pre-Indexed Addressing

memory address is formed in the same way as for offset addressing. Address is not only used for to access memory but the base register is also modified to hold the new value.

useful in loop to auto inc. or dec a counter

LDR R0, [R1,#4]! load R0 with the word at the address calculated by addig const value 4 to address. in R1

New memory address is placed back into the base register R1.

LDR R0, [R1,R2]! ~~R0~~ R1+R2 address
R2 is not altered
R1 is modified to hold the new addr.

Post - Indexed Addressing

It uses the value of the base register without modification. Then applies the modification to the address and writes the new address back to the base register.

LDR R0, [R1], #4 will load the reg R0 with word at the memory address contained in R1. It will then calculate the new value of R1 by adding 4 to current value.

LDR R0, [R1], R2

# [64-bit Addition]

```
LDR  R0, =value1            pointer to first value
LDR  R1, [R0]               load first part of value1
LDR, R2 [R0, ~~#0x4~~]      load lower part of value1

LDR  R0, =value2
LDR  R3, [R0]
LDR  R4, [R0, #0x04]        load lower part of value2
ADDS R6, R2, R4             ; Add lower 4 bytes & set carry flag
ADC  R5, R1, R3             upper 4 bytes including carry
LDR  R0, = Result           pointer to result
STR  R5, [R0]               store upper part of result
STR  R6 [R0, #0x04]         store lower part of result
SWI  &11
```

```
value1 DCD &12A2E640, &F2100123      Result DCD 0
value2 DCD &001019BF, &40023F51      END
value1 DCD &FFFFFFFF
END
```

WAP to count the no. of characters in a string

```
      AREA program, code, readonly
entry
      LDR    R0,=string
LOOP
      LDRB   R1, [R0], #0x01
      CMP    R1, #0x00
      ADDNE  R2, R2 , #0x01
      BNE    LOOP
      MOV    R4, R2
      SWI    &11
      AREA program, data , readonly
~~str DCD string~~
string DCB "ANJALI"
      END
```

DCB - directive defines one or more bytes of store. In addition to integer values, DCB accepts quoted strings. Each character of the string is placed in a consecutive byte

= is a synonym for DCB.

To construct a null-terminated C string using DCB

c_string DCB "c_string", 0

WAP to count a particular character in a stri

```
        AREA program, code, readonly
entry
        LDR   R0, = string
Loop
        LDRB  R1, [R0], #0x01
        CMP   R1, "a".
        ADDEQ R2, R2, #0x01
        CMP   R1, #0x00
        BNE   LOOP
        SWI   &11
        AREA  program, data, readonly
string  DCB   "ANJALI"
        END
```

WAP to add two numbers using offset addressing

```
        AREA program, code, readonly
entry
        LDR  R0, =value1          ;load the address of first value
        LDR  R1, [R0]                load what is at that address
        ADD  R0, R0, #0x04            Adjust the pointer
        LDR  R2, [R0]                load what is at the new addr
        ADD  R1, R1, R2
        LDR  R0, =Result          load the storage address
        STR  R1, [R0]                Store the result
        SWI  &11

value1   DCD  &00000007
value2   DCD  &00000008
Result   DCD  0
```