MOV Instruction

MOV RD, op2 ; op2 can be a register or 8 bit constant

- MOVT instruction that can load any value in the range 0x0000 to 0xFFFF into the most significant half of a register, without altering the contents of the least significant half.
- The LDR Rd,=const pseudo-instruction generates the most efficient single instruction to load any 32-bit number.
- MOV32 instruction can also be used to load any 32-bit number.

MOV32 R2, #5

Arithmetic and Logic Instructions

Instruction (Flags unchanged)		Instruction (Flags updated)	
ADD	Add	**ADDS**	Add and set flags
ADC	Add with carry	**ADCS**	Add with carry and set flags
SUB	SUBS	**SUBS**	Subtract and set flags
SBC	Subtract with carry	**SBCS**	Subtract with carry and set flags
MUL	Multiply		
UMULL	Multiply long		
RSB	Reverse subtract	**RSBS**	Reverse subtract and set flags
RSC	Reverse subtract with carry	**RSCS**	Reverse subtract with carry and set flags
Note: The above instruction affect all the N, Z, C, and V flag bits of CPSR (current program status register) but the N and V flags are for signed data and are discussed in Chapter 5.			

Arithmetic Instructions and Flag Bits for Unsigned Data

If suffix S is used after the opcode, CPSR register will be effected by the result. Instructions without S executes without having any effect on the flags

ADD Rd,Rn,Op2 ;Rd = Rn + Op2
ADC Rd,Rn,Op2 ;Rd = Rn + Op2 + C

The instructions ADD and ADC are used to add two operands. The destination operand must be a register. The Op2 operand can be a register or immediate. Remember that memory-to-register or memory-to-memory arithmetic and logic operations are never allowed in ARM Assembly language since it is a RISC processor. The instruction could change any of the Z, C, N, or V bits of the status flag register, as long as we use the ADDS or ADCS instead of ADD or ADC. ADC is used in the addition of multiword data.

SUB Rd,Rn,Op2 ;Rd = Rn - Op2

In ARM SUB instruction is executed as follows:

1. Take the 2's complement of the subtrahend (Op2 operand).
2. Add it to the minuend (Rn operand).
3. Place the result in destination Rd.
4. Set the carry flag if there is a carry.

These four steps are performed for every SUBS instruction by the internal hardware of the ARM CPU. It is after these four steps that the result is obtained and the flags are set. We must look at the carry flag (not the sign flag) to determine if the result is positive or negative. After the execution of SUBS, if C=1, the result is positive; if C = 0, the result is negative and the destination has the 2's complement of the result.

Analyze the following instructions:

```
MOV R1,#0x4C ;R1 = 0x4C
MOV R2,#0x6E ;R2 = 0x6E
SUBS R0,R1,R2 ;R0 = R1 – R2
```

Solution:

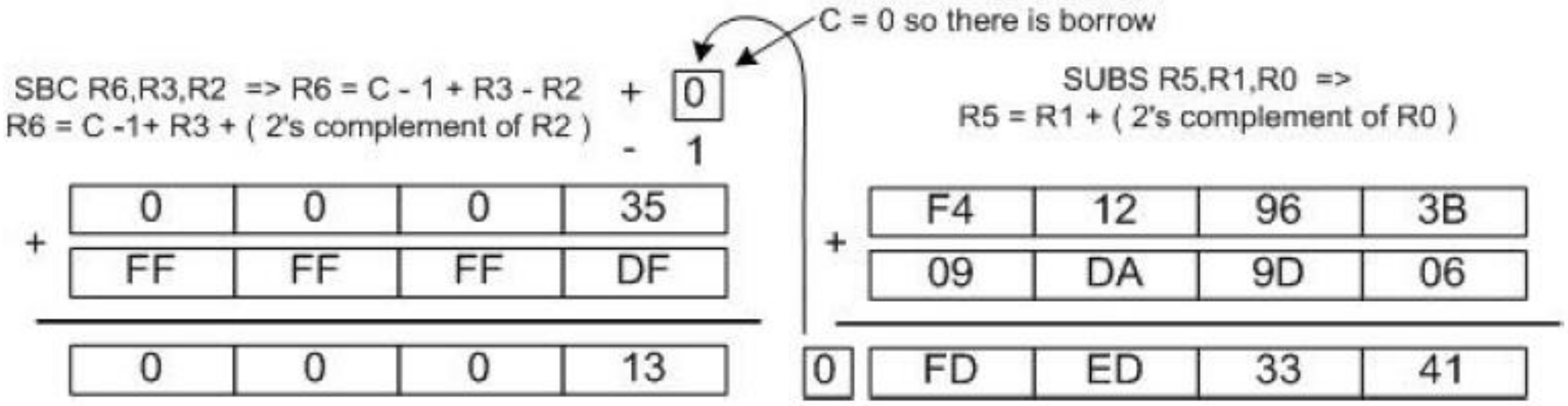
Following are the steps for “SUB R0,R1,R2”:

```
  4C          0000004C
–6E         + FFFFFF92 (2’s complement of 0x6E)
– 22        0 FFFFFFDE (C = 0 step 4) result is negative
```

SBC Rd,Rn,Op2 ;Rd = Rn – Op2 – 1 + C

This instruction is used for subtraction of multiword (data larger than 32-bit) numbers. In ARM the carry flag is not inverted after subtraction and carry flag is invert of borrow. To invert the carry flag while running the subtract with borrow instruction it is implemented
as “Rd = Rn – Op2 – 1 + C”

```
LDR R0,=0xF62562FA              ;R0 = 0xF62562FA,
LDR R1,=0xF412963B              ;R1 = 0xF412963B
MOV R2,#0x21                    ;R2 = 0x21
MOV R3,#0x35                    ;R3 = 0x35
SUBS R5,R1,R0           ;R5 = R1 – R0
                ; =0xF412963B – 0xF62562FA, and C = 0
SBC R6,R3,R2                    ;R6 = R3 – R2 – 1 + C
                ; = 0x35 – 0x21 – 1 + 0 = 0x13
```



RSB Rd,Rn,Op2 ;Rd = Op2 – Rn

This instruction can be used to get 2’s complement of a 32-bit operand

```
MOV R1,#0x1  ;R1=1
RSB R0,R1,#0   ;R0= 0 – R1 = 0 – 1
```

This is one way to get a fixed value of 0xFFFFFFFF in a register. R0=0xFFFFFFFF.

RSC Rd,Rn,Op2 **;Rd = Op2 – Rn – 1 + C**

This instruction can be used to get the 2's complement of the 64-bit operand.

Show how to create 2's complement of a 64-bit data in R0 and R1 register. The R0 hold the lower 32-bit.

Solution:

```
LDR R0,=0xF62562FA          ;R0 = 0xF62562FA
LDR R1,=0xF812963B          ;R1 = 0xF812963B
RSB R5,R0,#0                ;R5 = 0 – R0
                    ; = 0 – 0xF62562FA = 9DA9D06 and C = 0
RSC R6,R1,#0                ;R6 = 0 – R1 – 1 + C
                    ; = 0 – 0xF812963B – 1 + 0 = 7ED69C4
```

Multiplication of unsigned numbers in ARM

Instruction	Source 1	Source 2	Destination	Result
MUL	Rn	Op2	Rd (32 bits)	Rd=Rn×Op2
UMULL	Rn	Op2	RdLo, RdHi (64 bits)	RdLo:RdHi=Rn×Op2

Note 1: *Using MUL for word × word multiplication preserves only the lower 32 bit result in Rd and the rest are dropped. If the result is greater than 0xFFFFFFFF, then we must use UMULL (unsigned Multiply Long) instruction.*

Note 2: *In some CPUs the C flag is used to indicate the result is greater than 32-bit but this is not the case with ARM MUL instruction.*

MUL Rd,Rn,Op2 ;Rd = Rn × Op2

All the operands must be in registers. Immediate value is not allowed as an operand.

UMULL (unsigned multiply long)

UMULL RdLo,RdHi,Rn,Op2 ;RdHi:RdLo = Rn × Op2

In unsigned long multiplication, the operands must be in registers. After the multiplication, the destination registers will contain the result. Notice that the left most register, RdLo, will hold the lower word and the higher portion beyond 32-bit is saved in the second register, RdHi.

Multiply and Accumulate Instruction in ARM

MLA Rd,Rm,Rs,Rn ;Rd = Rm × Rs + Rn

In multiplication and add, the operands must be in registers. After the multiplication and add, the destination register will contain the result.

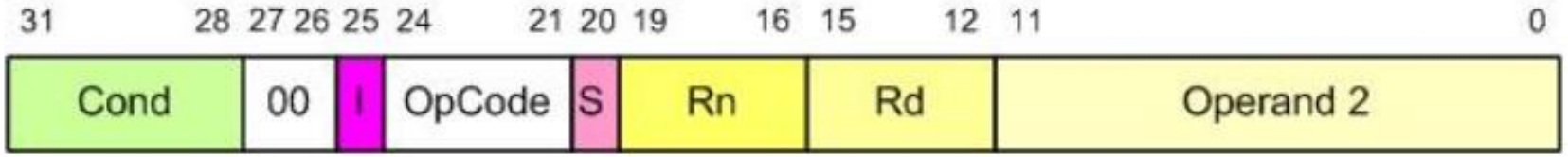
Notice that multiply and add can produce a result greater than 32-bit, if the MLA instruction is used, the destination register will hold the lower word (32-bit) and the portion beyond 32-bit is dropped.

UMLAL RdLo,RdHi,Rn,Op2 ;RdHi:RdLo = Rn × Op2 + RdHi:RdLo

In multiplication and add, the operands must be in registers. Notice that the addend and the high word of the destination use the same registers, the two left most registers in the instruction. It means that the contents of the registers which have the addend will be changed after execution of UMLAL instruction.

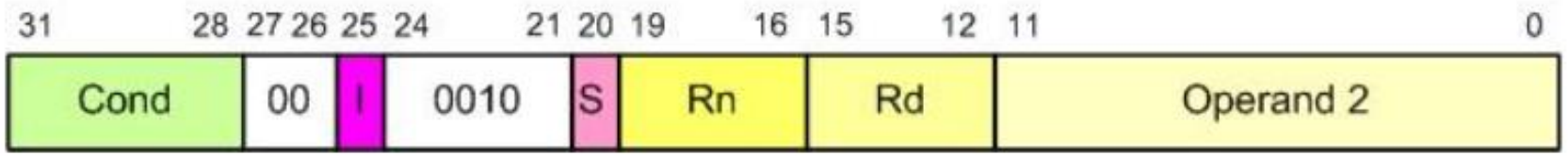
Division of unsigned numbers in ARM

It is done using repeated subtraction

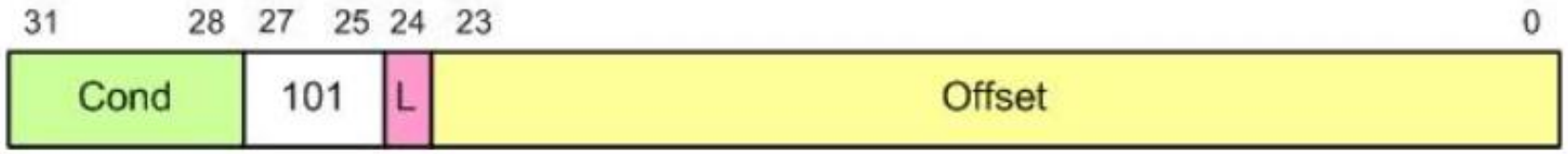


General Formation of Data Processing Instructions

ADD Instruction Formation



SUB Instruction Formation



Branch Instruction Formation

Logic Instructions

Instruction (Flags Unchanged)	Action	Instruction (Flags Changed)	Hexadecimal
AND	ANDing	**ANDS**	Anding and set flags
ORR	ORRing	**ORS**	Oring and set flags
EOR	Exclusive-ORing	**EORS**	Exclusive Oring and set flags
BIC	Bit Clearing	**BICS**	Bit clearing and set flags

```
AND Rd, Rn, Op2          ;Rd = Rn ANDed Op2
ORR Rd, Rn, Op2          ;Rd = Rn ORed Op2
EOR Rd,Rn,Op2            ;Rd = Rn Ex-ORed with Op2
BIC Rd,Rn,Op2            ;clear certain bits of Rn specified by the Op2 and place the result in Rd
```

In all these instructions Op2 can be register or immediate value

The BIC (bit clear) instruction is used to clear the selected bits of the Rn register. The selected bits are held by Op2. The bits that are HIGH in Op2 will be cleared and bits with LOW will be left unchanged. In reality, the BIC instruction performs AND operation on Rn register with the complement of Op2 and places the result in destination register

```
MOV R1,#0x0F
MOV R2,#0xAA
BIC R3,R2,R1          ;now R3 = 0xAA ANDed with 0xF0 = 0xA0
```

```
MVN Rd, Rn            ;move negative of Rn to Rd
```

The MVN (move negative) instruction is used to generate one's complement of an operand. For example, the instruction "MVN R2,#0" will make R2=0xFFFFFFFF. Look at the following example:

```
LDR R2,=0xAAAAAAAA              ;R2 = 0xAAAAAAAA
MVN R2,R2                       ;R2 = 0x55555555
```

It must be noted that the instruction "MVN Rd,#0" is widely used to load the fixed value of 0xFFFFFFFF into destination register.

Comparison of unsigned numbers

CMP Rn,Op2 ;compare Rn with Op2 and set the flags

- The CMP instruction compares two operands and changes the flags according to the result of the comparison. The operands themselves remain unchanged. The second source operands can be a register or an immediate value not larger than 0xFF.
- It must be emphasized that "CMP Rn,Op2" instruction is really a subtract operation. Op2 is subtracted from Rn (Rn – Op2) and the result is discarded and flags are set accordingly.
- Although all the C, S, Z, and V flags reflect the result of the comparison, only C and Z are used for unsigned numbers, as shown below:

	C	Z
Rn > Op2	1	0
Rn = Op2	1	1
Rn < Op2	0	0

```
        LDR R1,=0x35F           ;R1 = 0x35F

        LDR R2,=0xCCC           ;R2 = 0xCCC
        CMP R1,R2               ;compare 0x35F with 0xCCC
        BCC OVER                ;branch if C = 0
        MOV R1,#0               ;if C = 1, then clear R1
OVER    ADD R2,R2,#1            ;R2 = R2 + 1 = 0xCCC + 1 = 0xCCD
```

Instruction	Flags Affected
ANDS	C, Z, N
ORRS	C, Z, N
MOVS	C, Z, N
ADDS	C, Z, N, V
SUBS	C, Z, N, V
B	No flags
Note that we cannot put S after B instruction.	

Flag Bits Affected by Different Instructions

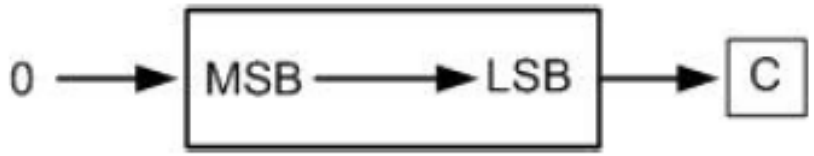
Rotate and Barrel Shifter

we can perform the shift and rotate operations as part of other instructions such as MOV

The process instructions can be used in one of the following forms:
1. opcode Rd, Rn, Rs (e.g. ADD R1,R2,R3)
2. opcode Rd, Rn, immediate Value (e.g. ADD R2,R3,#5)

ARM is able to shift or rotate the second argument before using it as the argument.

LSR Logical Shift Right




```
MOV R0,#0x9A                ;R0 = 0x9A
MOVS R1,R0,LSR #3    ;shift R0 to right 3 times
                            ;and then move (copy) the result to R1
```

```
MOV R0,#0x9A
MOV R2,#0x03
MOV R1,R0,LSR R2      ;shift R0 to right R2 times and move the result to R1
```

LSL Logical Shift Left


```
LDR R1,=0x0F000006
MOVS R2,R1,LSL #8

LDR R1,=0x0F000006
MOV R0,#0x08
MOV R2,R1,LSL R0
```

Operation	Destination	Source	Number of shifts
LSR (Shift Right)	Rd	Rn	Immediate value
LSR (Shift Right)	Rd	Rn	register Rm
LSL (Shift Left)	Rd	Rn	Immediate value
LSL (Shift Left)	Rd	Rn	register Rm
Note: Number of shift cannot be more than 32.			

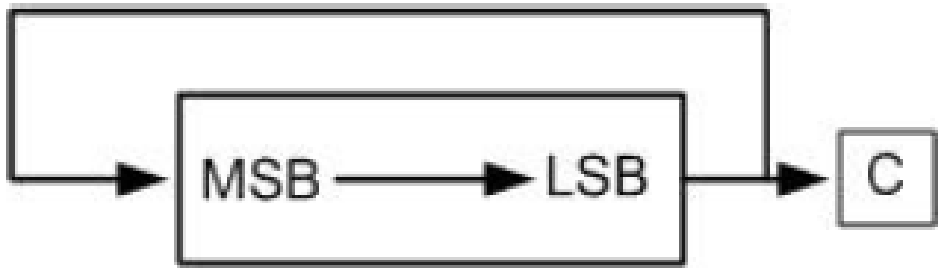
Logic Shift operations for unsigned numbers in ARM

ASR (arithmetic shift right)

MOV Rn,Op2, ASR count


```
MOV R0,#-10          ;R0 = -10 = 0xFFFFFFF6
MOV R3,R0,ASR #1     ;R0 is arithmetic shifted right once
                              ;R3 = 0xFFFFFFFB = -5
```

ROR (rotate right)




```
MOV R1,#0x36
MOVS R1,R1,ROR #1
```

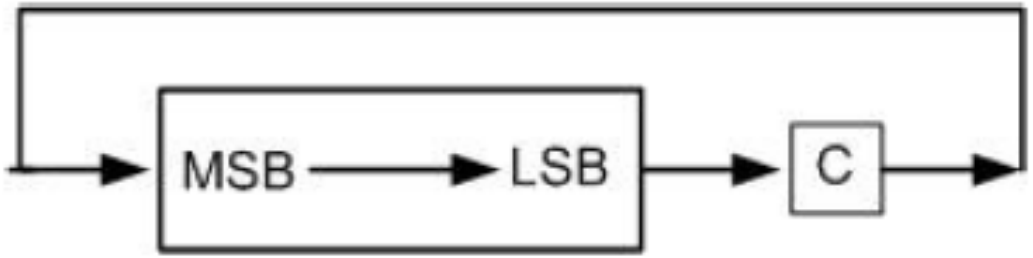
```
MOV R1,#0x36
MOV R0,#3
MOVS R1,R1,ROR R0
```

Rotate left

There is no rotate left option in ARM since one can use the rotate right (ROR) to do the job. That means instead of rotating left n bits we can use rotate right 32–n bits to do the job of rotate left. Using this method does not give us the proper carry if actual instruction of ROL was available

```
LDR R0,=0x00000072
;R0 = 0000 0000 0000 0000 0000 0000 0111 0010
MOVS R0,R0,ROR #31
;R0 = 0000 0000 0000 0000 0000 0000 1110 0100 C=0
```

RRX rotate right through carry



In RRX the LSB is moved to C and C is moved to the MSB. In reality, C flag acts as if it is part of the operand. That means the RRX is like 33-bit register since the C flag is 33rd bit. The RRX takes no arguments and the number of times an operand to be rotated is fixed at one.

```
;assume C=0
MOV R2,#0x26
;R2 = 0000 0000 0000 0000 0000 0000 0010 0110
MOVS R2,R2,RRX
;R2 = 0000 0000 0000 0000 0000 0000 0001 0011 C=0
```

Operation	Destination	Source	Number of Rotates
ROR (Rotate Right)	Rd	Rn	Immediate value
ROR (Rotate Right)	Rd	Rn	register Rm
RRX (Rotate Right Through Carry)	Rd	Rn	1 bit

Rotate operations for unsigned numbers in ARM

Shift and Rotate Instructions

In ARM Cortex M all the above shift and rotate instructions can be used as independent instructions

LSL Rd, Rm, Rn
LSL R0,R2,#8
LSL R2,R0,R1

LSLS Rd, Rm, Rn
LSLS R0,R2,#8
LSLS R2,R0,R1

Similarly, LSR and ROR

LSR Rd, Rm, Rn
LSRS Rd, Rm, Rn
ROR Rd, Rm, Rn
RORS Rd, Rm, Rn

```
RRX Rd,Rm                ;Rd=rotate Rm right 1 bit position
```

Function: Each bit of Rm register is shifted from left to right one bit. The RRX does not update the flags.

```
LDR R2,=0x00000002
RRX R0,R2                        ;R0=R2 is shifted right one bit
                                 ;now, R0=0x00000001
```

RRXS Rotate Right with extend (update the flags)

```
RRXS Rd,Rm               ;Rd=rotate Rm right 1 bit position
```

Function: Each bit of Rm register is shifted from left to right one bit. The RRXS updates the flags.

```
LDR R2,=0x00000002
RRXS R0,R2               ;R0=R2 is shifted right one bit
                                 ;now, R0=0x00000001
```

EQ	Z set	equal
NE	Z clear	not equal
CS/HS	C set	unsigned higher or same
CC/LO	C clear	unsigned lower
MI	N set	negative
PL	N clear	positive or zero
VS	V set	overflow
VC	V clear	no overflow
HI	C set and Z clear	unsigned higher
LS	C clear or Z set	unsigned lower or same
GE	N equals V	signed greater or equal
LT	N not equal to V	signed less than
GT	Z clear AND (N equals V)	signed greater than
LE	Z set OR (N not equal to V)	signed less than or equal
AL	(ignored)	always (usually omitted)

Add 10 32-bit numbers available in the code memory and store the result in data memory.

Conditional Execution

- If we do not add a condition after an instruction, it will be executed unconditionally because the default is not to check the flags and execute unconditionally.
- If we want an instruction to be executed only when a condition is met, we put the condition syntax right after the instruction.
- This feature of ARM allows the execution of an instruction conditionally based on the status of Z, C, V and N flags. To do that, the ARM instructions have set aside the most significant 4 bits of the instruction field for the conditions. The 4 bits gives us 16 possible conditions.

To make an instruction conditional, simply we put the condition syntax in front of it.

```
MOV R1,#10    ;R1 = 10
MOV R2,#12    ;R2 = 12
CMP R2,R1     ;compare 12 with 10, Z = 0 because they are not equal
MOVEQ R4,#20 ;this line is not executed because the condition EQ is not met

CMP R1,#0     ;compare R1 with 0
ADDNE R1,R1,#10                ;this line is executed if Z = 0 (if in the last CMP
                               ;operands were not equal)
```

Note that we can add both S and condition to syntax of an instruction. It is common to put S after the condition. See the following examples:

```
ADDNES R1,R1,#10
;this line is executed and set the flags if Z = 0
```

BCD and ASCII Conversion

(1) unpacked BCD

(2) packed BCD.

(3) ASCII numbers

BL (branch and link)

- Subroutines are often used to perform tasks that need to be performed frequently.
- In the ARM there is only one instruction for call and that is BL (branch and link).
- To use BL instruction for call, we must leave the R14 register unused since that is where the ARM CPU stores the address of the next instruction where it returns to resume executing the program.
- ARM automatically saves in the link register (LR), the R14, the address of the instruction immediately below the BL.

- When a subroutine is called by the BL instruction, control is transferred to that subroutine, and the processor saves the PC (program counter) in the R14 register and begins to fetch instructions from the new location.

- After finishing execution of the subroutine, we must use “BX LR“ instruction to transfer control back to the caller. Every subroutine needs “BX LR” as the last instruction for return address.

Branching beyond 32M byte limit

- To branch beyond the address space of 32M bytes, we use BX (branch and exchange) instruction.
- The “BX Rn” instruction uses register Rn to hold target address. Since Rn can be any of the R0–R14 registers and they are 32-bit registers, the “BX Rn” instruction can land anywhere in the 4G bytes address space of the ARM.
- In the instruction “BX R2” the R2 is loaded into the program counter (R15) and CPU starts to fetch instructions from the target address pointed to by R15, the program counter.

TST (Test)

TST Rn,Op2 ;Rn AND with Op2 and flag bits are updated

The TST instruction is used to test the contents of register to see if any bit is set to HIGH.
After the operands are ANDed together the flags are updated.
After the TST instruction if result is zero, then Z flag is raised and one can use BEQ (branch equal) to make decision.
example:

```
        MOV R0,#0x04        ;R0=00000100 in binary
        LDR R1,=myport      ;port address
OVER    LDRB R2,[R1]   ;load R2 from myport
        TST R2,R0           ;is bit 2 HIGH?
        BEQ OVER            ;keep checking
```

In TST, the Op2 can be an immediate value of less than 0xFF.
example:

```
        LDR R1,=myport    ;port address
OVER  LDRB R2,[R1]        ;load R2 from myport
        TST R2,#0x04              ;is bit 2 HIGH?
        BEQ OVER          ;keep checking
```

TEQ (test equal)

- TEQ Rn,Op2 ;Rn EX-ORed with Op2 and flag bits are set

- The TEQ instruction is used to test to see if the contents of two registers are equal.

- After the source operands are Ex-ORed together the flag bits are set according to the result.

- After the TEQ instruction if result is 0, then Z flag is raised and one can use BEQ (branch zero) to make decision.

```
        TEMP EQU 100
        MOV R0,#TEMP            ;R0 = Temp
        LDR R1,=myport          ;port address
OVER    LDRB R2,[R1]            ;load R2 from myport
        TEQ R2,R0               ;is it 100?
        BNE OVER                ;keep checking
```

Home Work:

- Convert 8 digit ASCII into unpacked BCD

Input: 0x31323334, Output: 0x01020304

- Unpack 8 digit packed BCD

Input: 0x12345678 Output: 0102030405060708

- Pack 8 digit unpacked BCD

Input: 0x01020304, Output: 0x1234

- GCD and LCM