Instructions_02

CS2400

Spring 2020

Program Review

- Write an instruction to solve:

Z= (a << 2) | (c & 15)

```
        AREA Logical,CODE,READONLY
         LDR R4,=A
        LDR R0,[R4]
        MOV R0,R0,LSL #2
        LDR R4,=C
        LDR R0,[R4]
        AND R1,R1,#15
        ORR R1,R0,R1
          LDR R4,=Z
          STR R1,[R4]
st          B  st
A DCD 0x04
C DCD 0x45
                AREA data1,DATA,READWRITE
Z  DCD 0
                END
```

Multiplication

- Multiplicand X Multiplier = Product

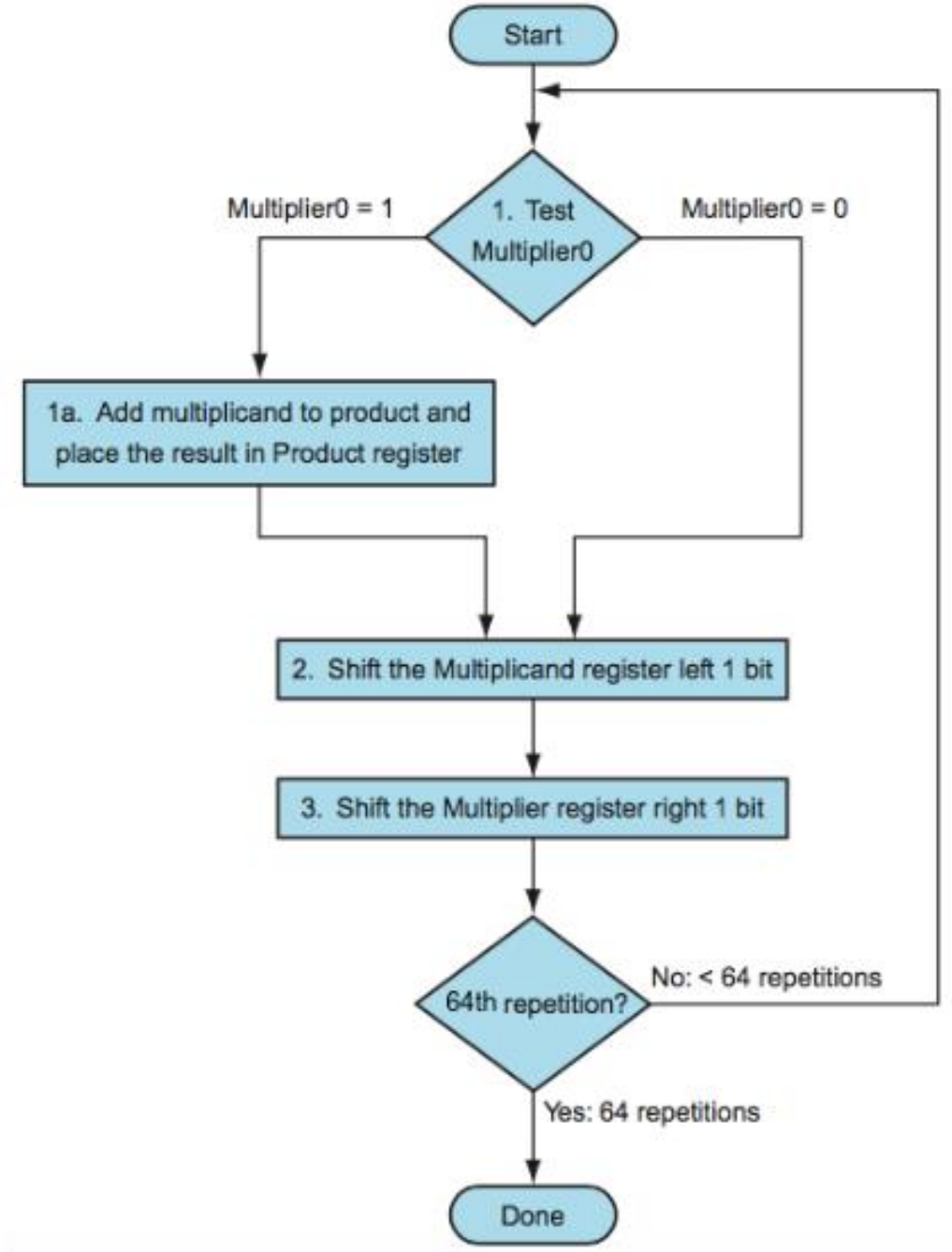
- Number of bits in Multiplicand is n
- Number of bits in Multiplier is m
- Number of bits in Product is n+m

Example

- **1 1 0 1 (Multiplicand)**
- **x 0 1 1 0 (Multiplier)**
- ----------
- ? ? ? ? (Partial product 1)
- ? ? ? ? (Partial product 2)
- ? ? ? ? (Partial product 3)
- + ? ? ? ? (Partial product 4)
- ----------
- ? ? ? ? ? ? ? ? (Product)

1 1 0 1 (Multiplicand)
x 0 1 1 0 (Multiplier)
---
0 0 0 0 (Partial product 1)
1 1 0 1 (Partial product 2)
1 1 0 1 (Partial product 3)
+ 0 0 0 0 (Partial product 4)
---
1 0 0 1 1 1 0 (Product)

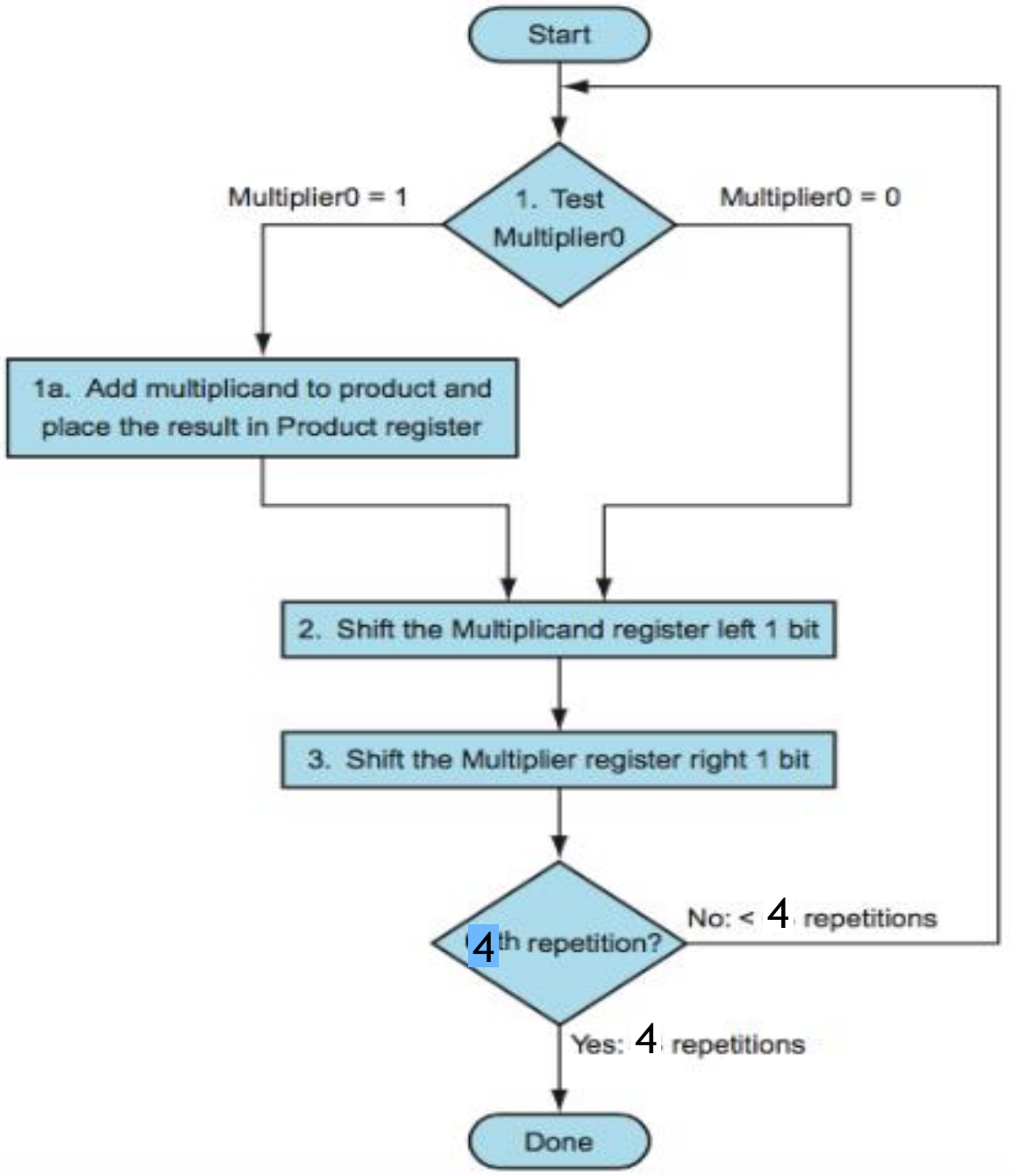
Flow Chart


Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
64th repetition?
No: < 64 repetitions
Yes: 64 repetitions
Done

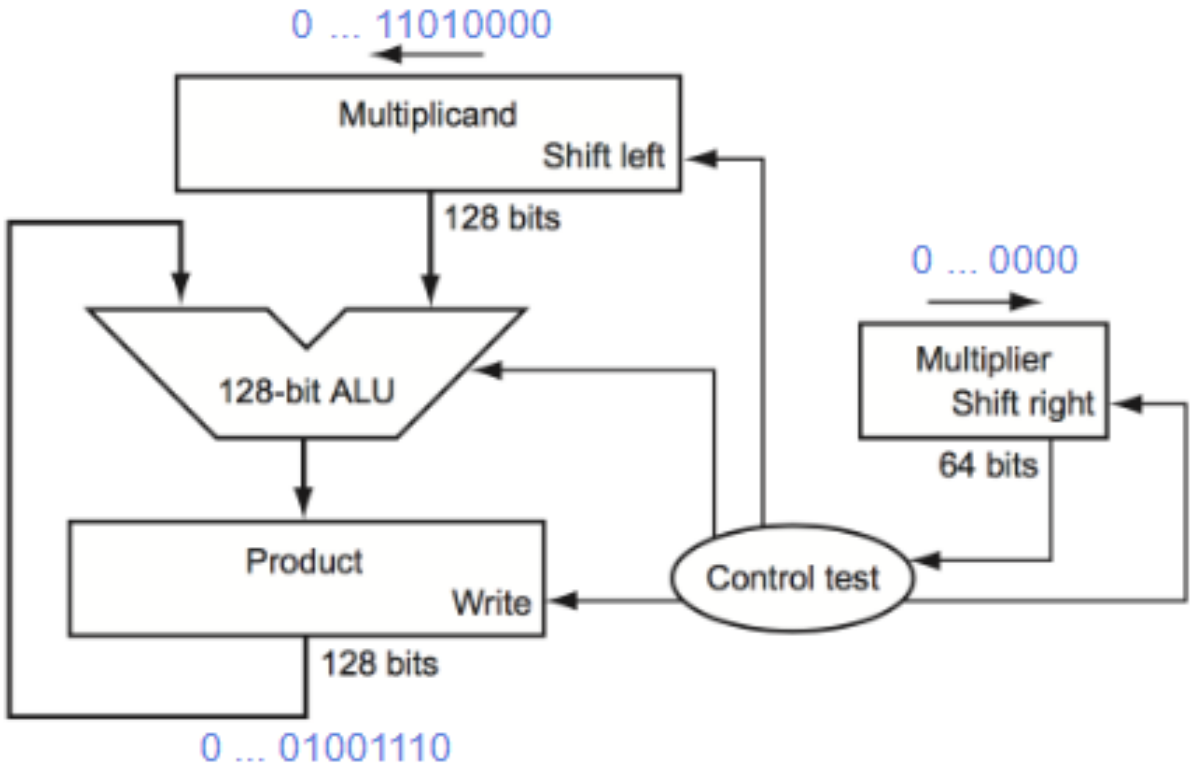
Using Algorith

- 1 1 0 1 (Multiplicand)
- x 0 1 1 0 (Multiplier)
- ----------

1 0 0 1 1 1 0 (Product)



Hardware for multiplication



Multiply Instructions

Mnemonic	Meaning	Register Operations
MLA	Multiply and Accumulate	Rd = (Rm*Rs)+Rn
MUL	Multiply	Rd = (Rm*Rs)
UMLAL	Multiply and Accumulate Long (unsigned)	[RdHi,RdLo] = [RdHi,RdLo]+ (Rm*Rs)
UMULL	Multiply long (unsigned)	[RdHi,RdLo] = (Rm*Rs)
SMLAL	Multiply and Accumulate Long (signed)	[RdHi,RdLo] = [RdHi,RdLo]+ (Rm*Rs)
SMULL	Multiply long (signed)	[RdHi,RdLo] = (Rm*Rs)

Division Instructions

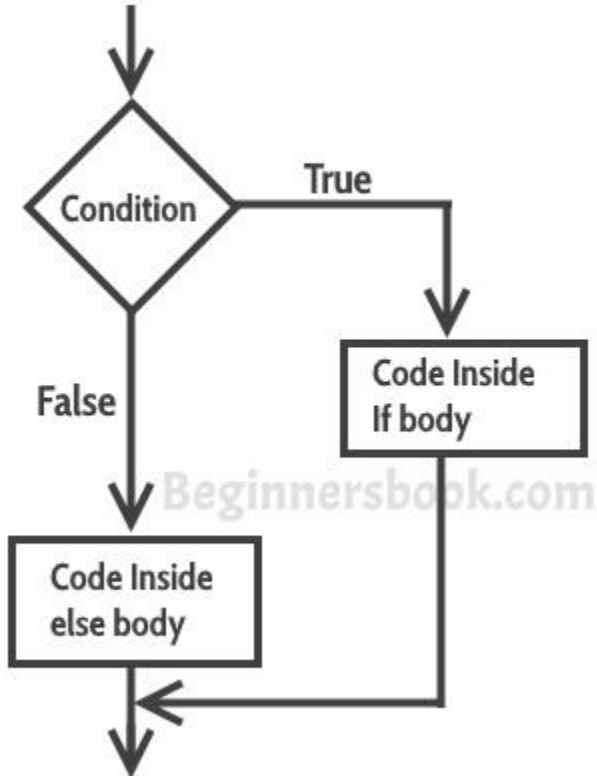
Mnemonic	Meaning	Register Operations
UDIV	Unsigned division	UDIV R0,R1,R2 => R0=R1/R2(Unsigned)
SDIV	Signed division	SDIV R0,R1,R2 => R0=R1/R2 (Signed)

Branching

- Decision making:

- CBZ register, LabelName
  - CBZ R0, goto1

- CBNZ register, LabelName
  - CBNZ R0, goto1



Example

```
        MOV R0,#0
        CBZ R0,goto1     ; Conditional Branch  Zero/CBNZ for non zero
        ADD R0,R0,#1
        B stop
goto1
         ADD R0,R0,#2
stop  B stop
```

Example

```
if(i==j)
   a=a+1
else
  a=a-1
```

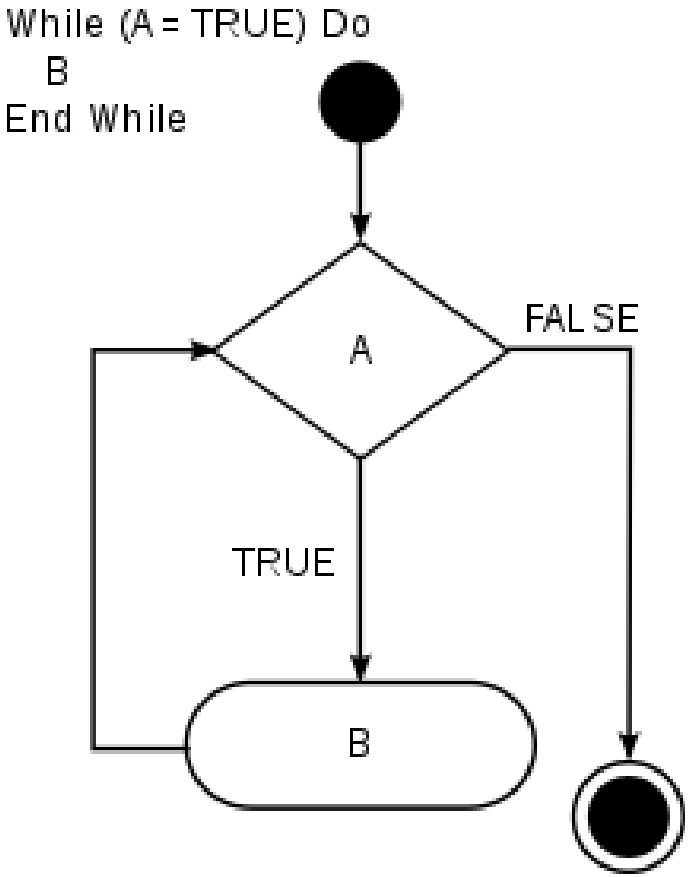
```
        MOV R4,8
        MOV R1,5
        MOV R2,5
        SUB R3,R1,R2
        CBNZ R3, else1
        ADD R4,R4,#1
        B stop
else1
        SUB R4,R4,#1
stop
        END
```

In Class Program conditional Branching

- Write an ARM assembly program to accept a value from a variable ‘a’ . If value in ‘a’ is equal to 10 , Add 1 to it else add 2 to it.

Loops

```
Loop
    CBNZ X0, Exit
    # Loop body
    B Loop
Exit
```



Example Loops

```
while(a<5)
{
 a=a+1
}
```

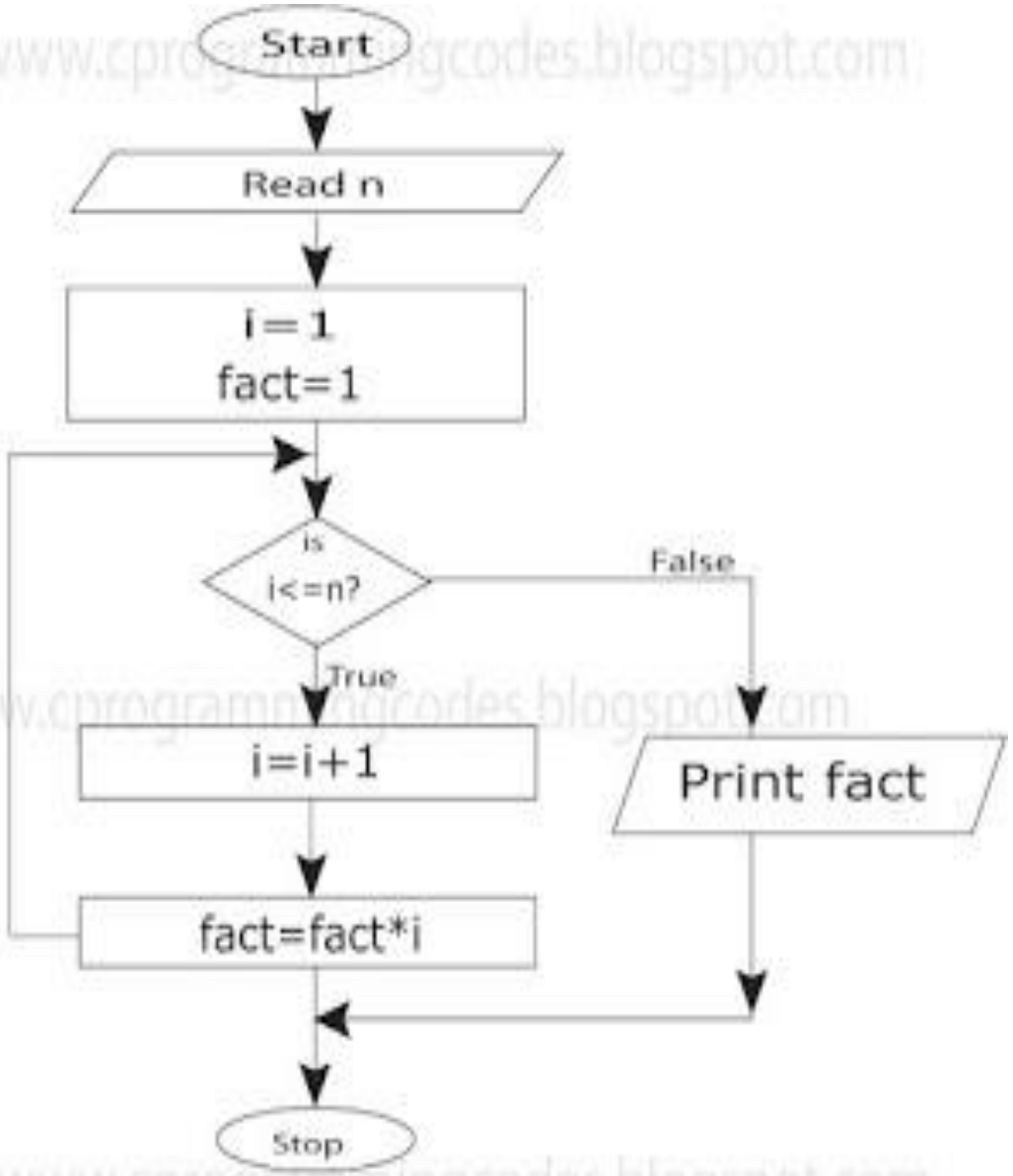
```
        LDR R4,=a
        LDR R1,[R4]
loop1
        SUB R2,R1,#5
        CBZ R2,exit
        ADD R1,R1,#1
        B loop1
exit
stop    B stop

a DCD 1
    END
```

In class program

- *Write a program using ARM instructions to find Factorial of a number .*

- *Write the program on paper and then test it using Keil.*



- Factorial Program

```
;factorial
        LDR R4,=a
        LDR R1,[R4]
        MOV R3,#1
loop1
         SUB R2,R1,#1
        CBZ R2,exit
        MUL R3,R3,R1
        SUB R1,R1,#1
        B loop1
exit
stop   B stop

a DCD 3
     END
```

Conditions using Branching

	Signed		Unsigned	
Comparison	Instruction	Flag Test	Instruction	Flag Test
$=$	BEQ	$Z = 1$	BEQ	$Z = 1$
$\neq$	BNE	$Z = 0$	BNE	$Z = 0$
$<$	BLT	$N! = V$	BLO	$C = 0$
$\leq$	BLE	$\sim(Z = 0 \,\&\, N = V)$	BLS	$\sim(Z = 0 \,\&\, C = 1)$
$>$	BGT	$(Z = 0 \,\&\, N = V)$	BHI	$(Z = 0 \,\&\, C = 1)$
$\geq$	BGE	$N = V$	BHS	$C = 1$

Conditional Branc

```
        MOVS R0,#1
        BEQ  goto1   ; Z = 1
        ADD R0,R0,#1
stop            B  stop
goto1
        ADD R0,R0,#2
        B stop
```

Suffix	Description	Flags
EQ	Equal / equals zero	Z
NE	Not equal	!Z
CS / HS	Carry set / unsigned higher or same	C
CC / LO	Carry clear / unsigned lower	!C
MI	Minus / negative	N
PL	Plus / positive or zero	!N
VS	Overflow	V
VC	No overflow	!V
HI	Unsigned higher	C and !Z
LS	Unsigned lower or same	!C or Z
GE	Signed greater than or equal	N == V
LT	Signed less than	N != V
GT	Signed greater than	!Z and (N == V)
LE	Signed less than or equal	Z or (N != V)
AL	Always (default)	any

Compare Instructions

Mnemonic	Meaning	Register Operations
CMN	Compare Negated	Flags Set -> Rn+Operand 2
CMP	Compare	Flags Set -> Rn-Operand2
TEQ	Test for quality-two 32 bit values	Flags Set -> Rn ^ Operand2

Example

- **Compare two given strings ?**
- LDR R4,=St1
- LDR R0,[R4]
- LDR R4,=St2
- LDR R1,[R4]
- **TEQ R0,R1 ; Z=0**

- St1 DCB “one”
- St2 DCB “test”

LDR R4,=St1

LDR R0,[R4]

LDR R4,=St2

LDR R1,[R4]

TEQ R0,R1 ; Z=1

St1 DCB “one”
St2 DCB “one”

Q

Convert the algorithm by **conditional execution using branches**

```
while (a != b)
    {
      if (a > b)
          a = a - b;
      else
          b = b - a;
    }
```

Q Solution

```
;Conditional Branching

            MOV r0,#3
            MOV r1,#4
gcd         CMP     r0, r1
            BEQ     end
            BLT     less
            SUB     r0, r0, r1
            B       gcd
less
            SUB     r1, r1, r0
            B       gcd
end
```

Modify the code so that we can reduce the number of instructions ?

Modified Program Solution

```
;Modified Conditional Branch
                MOV r0,#3
                MOV r1,#4
gcd             CMP r0, r1
                SUBGT r0, r0, r1
                SUBLT r1, r1, r0
                BNE gcd
exit            B  exit
                    END
```