

Chapters 6

Flow Control In Assembly

Embedded Systems with ARM Cortex-M

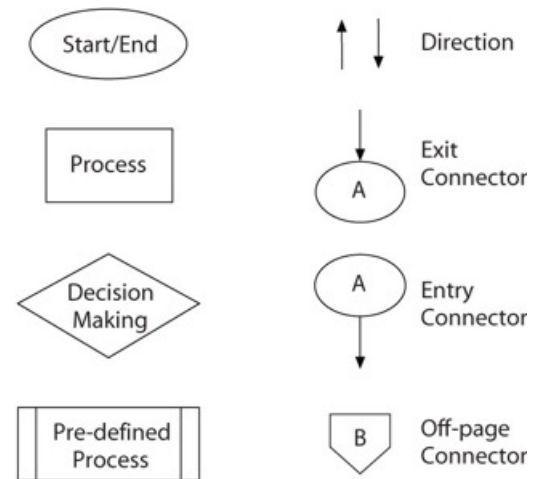
Updated: Monday, February 19, 2018

Overview: Flow Control

- Basics of Flowcharting
- **If-then-else**
- **While loop**
- **For loop**
-

Flowcharting

- Flowchart
 - A graphical representation of processes (tasks) to be performed and the sequence to be followed in solving computational problem

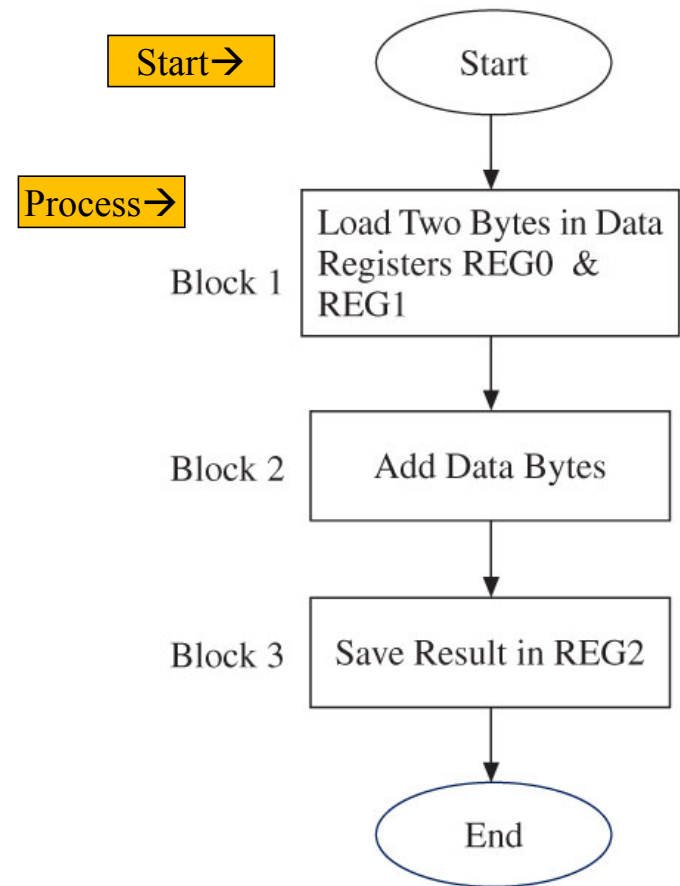


Example

- Write instructions to load two bytes (37H and 92H) in data registers REG0 and REG1. Add the bytes and store the sum in REG2.
- Steps
 - Load the two bytes in data registers REG0 and REG1.
 - Add the bytes.
 - Save the sum in data register REG2.

Example

REG0	EQU	0x37
REG1	EQU	0x92
	MOV	R0,#REG0
	MOV	R1,#REG1
	ADD	R2, R1, R0
A_HERE	B	A_HERE



Steps in Writing and Executing Assembly Language Program

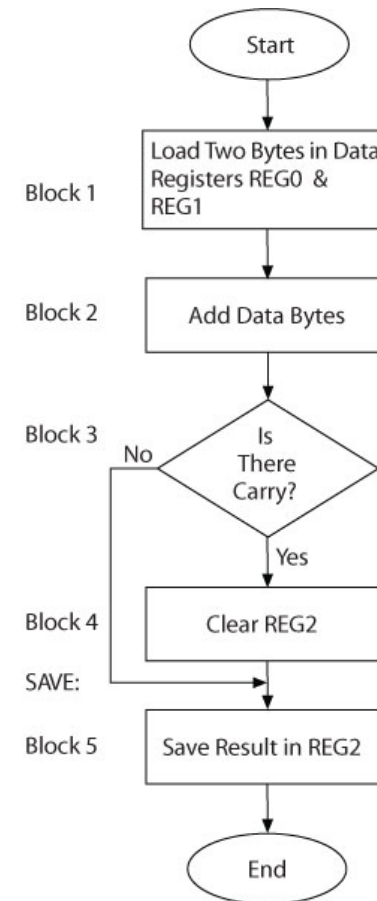
- Analyze the problem.
- Draw a flowchart.
- Convert the flowchart in mnemonics.
- Look up Hex code and assign memory addresses.
- Enter the Hex code into memory of a lab training board.
- Execute the program.
- Debug the program if necessary.

Illustrative Program: Addition With Carry Check

- Write instructions to load two bytes, Byte1 (F2H) and Byte2 (32H), in data registers REG0 and REG1 respectively and add the bytes.
- If the sum generates a carry, clear the data register REG2; otherwise, save the sum in REG2.

Illustrative Program: Addition With Carry Check

- Write instructions to load two bytes, Byte1 (F2H) and Byte2 (32H), in data registers REG0 and REG1 respectively and add the bytes.
- If the sum generates a carry, clear the data register REG2; otherwise, save the sum in REG2.



Condition Testing

Application PSR
(APSR)




- Most assembly instructions can be executed based on flags (N, Z, C, V)
- The condition of these flags is set in Application Program Status Register (APSR)
- These conditions depend on SIGNED or UNSIGNED nature of the inputs

Suffix	Description	Flags tested
EQ	Equal (Unsigned & Signed)	Z=1
NE	Not Equal (Unsigned & Signed)	Z=0
CS/HS	Unsigned Higher or Same	C=1
CC/LO	Unsigned Lower	C=0
MI	Minus (Negative)	N=1
PL	Plus (Positive or Zero)	N=0
VS	Overflow Set	V=1
VC	Overflow Clear	V=0
HI	Unsigned Higher	C=1 & Z=0
LS	Unsigned Lower or Same	C=0 or Z=1
GE	Signed Greater or Equal	N=V
LT	Signed Less Than	N!=V
GT	Signed Greater Than	Z=0 & N=V
LE	Signed Less than or Equal	Z=1 or N!=V
AL	Always	

Signed vs. Unsigned & Branch Conditions

Conditional codes applied to branch instructions

Compare	Signed	Unsigned
==	EQ	EQ
≠	NE	NE
>	GT	HI
≥	GE	HS
<	LT	LO
≤	LE	LS



Compare	Signed	Unsigned
==	BEQ	BEQ
!=	BNE	BNE
>	BGT	BHI
≥	BGE	BHS
<	BLT	BLO
<=	BLE	BLS

Note: Branches can be conditional or unconditional (B)

Comparison Instructions

Instruction	Operands	Brief description	Flags
CMP	Rn, Op2	Compare	N,Z,C,V
CMN	Rn, Op2	Compare Negative	N,Z,C,V
TEQ	Rn, Op2	Test Equivalence	N,Z,C
TST	Rn, Op2	Test	N,Z,C

➤ The only effect of the comparisons is to **update the condition flags**.

- No need to set S bit.
- No need to specify Rd.

➤ Operations are:

- **CMP** operand1 - operand2, but result not written
- **CMN** operand1 + operand2, but result not written
- **TST** operand1 & operand2, but result not written
- **TEQ** operand1 ^ operand2, but result not written

➤ Examples:

- **CMP** r0, r1
- **TST** r2, #5

Example: Which is Greater: 0xFFFFFFFF or 0x00000001?

It's **software's responsibility** to tell computer how to interpret data:

- If written in C, declare the signed vs unsigned variable
- If written in Assembly, use signed vs unsigned branch instructions

```
signed int x, y ;  
x = -1;  
y = 1;  
if (x > y)  
    ...
```

```
MOVS r6, #0xFFFFFFFF  
MOVS r5, #0x00000001  
CMP  r5, r6  
BLE Then_Clause  
...
```

```
MOVS    R5, #1  
MOVS    R6, #0xFFFFFFFF    R6=-1  
CMP     R5,R6  
BLE     ELSE ;IF R5<=R6 (UNSIGNED)  
THEN    MOV  R7,#1  
B       ENDIF  
ELSE    MOV  R7,#0  
ENDIF
```

BLE: Branch if less than or equal, signed \leq

```
unsigned int x, y ;  
x = 4294967295;  
y = 1;  
if (x > y)  
    ...
```

```
MOVS r6, #0xFFFFFFFF  
MOVS r5, #0x00000001  
CMP  r5, r6  
BLS Then_Clause  
...
```

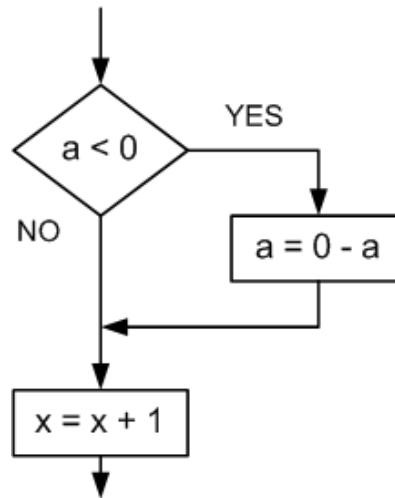
```
MOVS    R5, #1  
MOVS    R6, #0xFFFFFFFF    R6= 4294967295  
CMP     R5,R6  
BLS     Q_ELSE ;IF R5<=R6 (UNSIGNED)  
Q_THEN  MOV  R7,#1  
B       Q_ENDIF  
Q_ELSE  MOV  R7,#0  
Q_ENDIF
```

BLS: Branch if lower or same, unsigned \leq

If-then Statement

C Program

```
if (a < 0) {  
    a = 0 - a;  
}  
x = x + 1;
```



```
;CONDITIONAL BRANCH  
MOV    R1, #-5  
CMP    R1, #0  
BGE    MENDIF ;BRANCH IF R1>0 (SIGNED)  
RSB    R1, R1, #0  
MENDIF ADD    R2, R2, #1
```

Implementation 1:

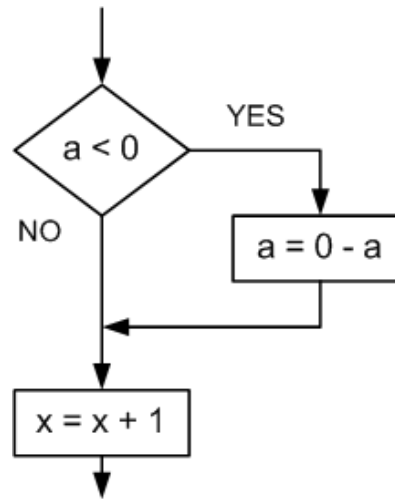
```
        ; r1 = a, r2 = x  
        CMP r1, #0          ; Compare a with 0  
        BGE endif          ; Go to endif if a ≥ 0  
then    RSB r1, r1, #0      ; a = - a  
endif   ADD r2, r2, #1      ; x = x + 1
```

If-then Statement

```
C Program
if (a < 0 ) {
    a = 0 - a;
}
x = x + 1;
```

Implementation 2:

Example of
conditional
Execution
RSB + LT



```
; r1 = a, r2 = x
CMP    r1, #0          ; Compare a with 0
RSBLT r1, r1, #0      ; a = 0 - a if a < 0
ADD    r2, r2, #1      ; x = x + 1
```

```
;;CONDITIONAL EXECUTION
MOV    R1, #5
CMP    R1, #0
RSBLT R1, R1, #0 ;EXECUTE RSB IF SIGNED R0 LESS THAN 0 (IF R1<=0)
ADD    R2, R2, #1
```

Compound Boolean Expression

```
x > 20 && x < 25
x == 20 || x == 25
!(x == 20 || x == 25)
```

```
60 ;IF-THEN WITH COMPOUNN LOGICAL OR
61 ; IF (R0 <= 20 || R0>=25)--> R1=1
62 MOV R0, #-2
63 CMP R0, #20
64 BLE S_THEN
65 CMP R0, #25
66 BLT S_ENDIF
67 S_THEN MOV R1, #1
68 S_ENDIF
```

C Program	Assembly Program
// x is a signed integer if(x <= 20 x >= 25){ a = 1 }	; r0 = x CMP r0, #20 ; compare x and 20 BLE then ; go to then if x ≤ 20 CMP r0, #25 ; compare x and 25 BLT endif ; go to endif if x < 25 then MOV r1, #1 ; a = 1 endif

Compound Boolean Expression Alternative

```
x > 20 && x < 25  
x == 20 || x == 25  
!(x == 20 || x == 25)
```

C Program

```
// x is a signed integer  
if(x <= 20 || x >= 25){  
    a = 1  
}
```

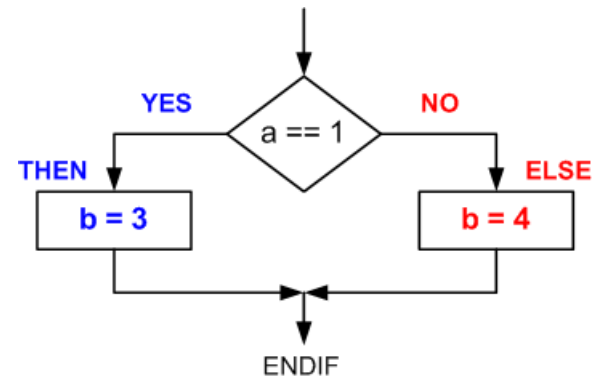
```
70 ;SAME AS ABOVE BUT SLOWER  
71 ;IF-THEN WITH COMPOUND LOGICAL OR  
72 ; IF (R0 <= 20 || R0 >= 25)--> R1=1  
73 MOV R1,#0  
74 MOV R0,#19  
75 CMP R0,#20  
76 MOVLE R1,#1  
77 CMP R0,#25  
78 MOVGE R1,#1
```

Example of
conditional
Execution
MOV + GE

If-then-else

C Program

```
if (a == 1)
    b = 3;
else
    b = 4;
```



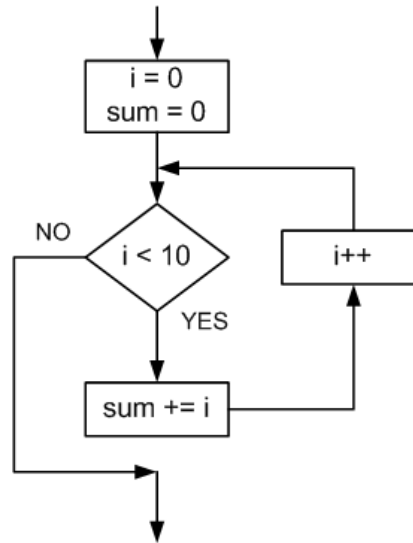
```
    ; r1 = a, r2 = b
    CMP r1, #1    ; compare a and 1
    BNE else     ; go to else if a ≠ 1
then MOV r2, #3   ; b = 3
    B endif      ; go to endif
else MOV r2, #4   ; b = 4
endif
```

```
;IF-THEN-ELSE STATEMENT
;IF R2==1 THEN R1=3 ELSE R1=4
MOV R1, #0
MOV R2, #-2
CMP R2, #1
MOVEQ R1, #3
MOVNE R1, #4
```

For Loop

C Program

```
int i;  
int sum = 0;  
for(i = 0; i < 10; i++){  
    sum += i;  
}
```



One Implementation is

```
MOV r0, #0 ; i  
MOV r1, #0 ; sum  
  
loop    CMP r0, #10  
        BGE endloop  
        ADD r1, r1, r0  
        ADD r0, r0, #1  
        B loop  
endloop
```

Question:
How can you implement
this code using
Conditional execution
(e.g., ADD)?

Conditional Execution Examples for ADD

Add instruction	Condition	Flag tested
ADDEQ r3, r2, r1	Add if EQual	Add if Z = 1
ADDNE r3, r2, r1	Add if Not Equal	Add if Z = 0
ADDHS r3, r2, r1	Add if Unsigned Higher or Same	Add if C = 1
ADDLO r3, r2, r1	Add if Unsigned LOwer	Add if C = 0
ADDMI r3, r2, r1	Add if Minus (Negative)	Add if N = 1
ADDPL r3, r2, r1	Add if PLus (Positive or Zero)	Add if N = 0
ADDVS r3, r2, r1	Add if oVerflow Set	Add if V = 1
ADDVC r3, r2, r1	Add if oVerflow Clear	Add if V = 0
ADDHI r3, r2, r1	Add if Unsigned HIgher	Add if C = 1 & Z = 0
ADDLS r3, r2, r1	Add if Unsigned Lower or Same	Add if C = 0 or Z = 1
ADDGE r3, r2, r1	Add if Signed Greater or Equal	Add if N = V
ADDLT r3, r2, r1	Add if Signed Less Than	Add if N != V
ADDGT r3, r2, r1	Add if Signed Greater Than	Add if Z = 0 & N = V
ADDLE r3, r2, r1	Add if Signed Less than or Equal	Add if Z = 1 or N = !V

Examples of Conditional Execution

```
if (a <= 0)
    y = -1;
else
    y = 1;
```



a → r0
y → r1

```
CMP    r0, #0
MOVLE  r1, #-1
MOVGT  r1, #1
```

LE: Signed Less than or Equal
GT: Signed Greater Than

```
if (a==1 || a==7 || a==11)
    y = 1;
else
    y = -1;
```



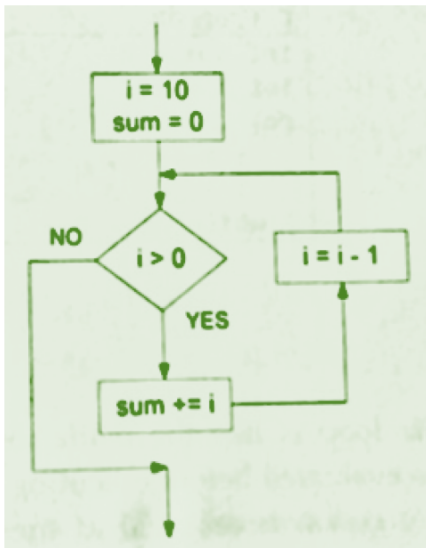
```
CMP    r0, #1
CMPNE  r0, #7
CMPNE  r0, #11
MOVEQ  r1, #1
MOVNE  r1, #-1
```

a → r0
y → r1

NE: Not Equal
EQ: Equal

While Loop

- Run a loop until a condition occurs



C Program

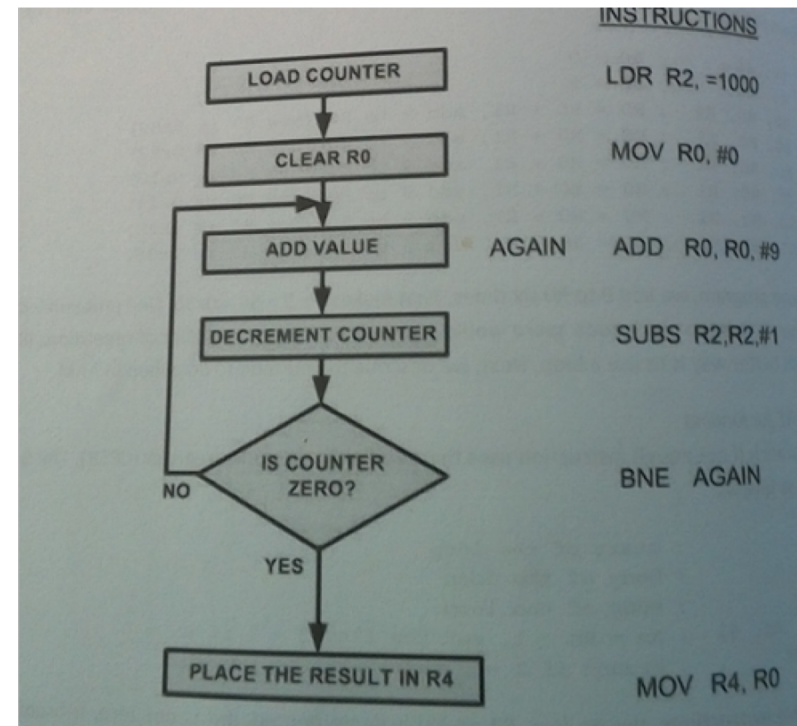
```
int i = 10;
int sum = 0;

while( i > 0 ){
    sum += i;
    i--;
}
```

```
15 ;IMPEMENTING WHILE-LOOP
16 ;WHILE i>0 --> SUM = SUM+1, i=i-1
17 MOV R0,#5 ;R0=1
18 MOV R1,#0 ;SUM
19 B_LOOP CMP R0,#0
20 BLE B_ENDLOOP
21 ADD R1,R1,R0
22 SUB R0,R0,#1
23 B B_LOOP
24 B_ENDLOOP
```

Write a program to clear R0 and add 9 to R0 1000 times. Place the sum in R4. Use BNE.

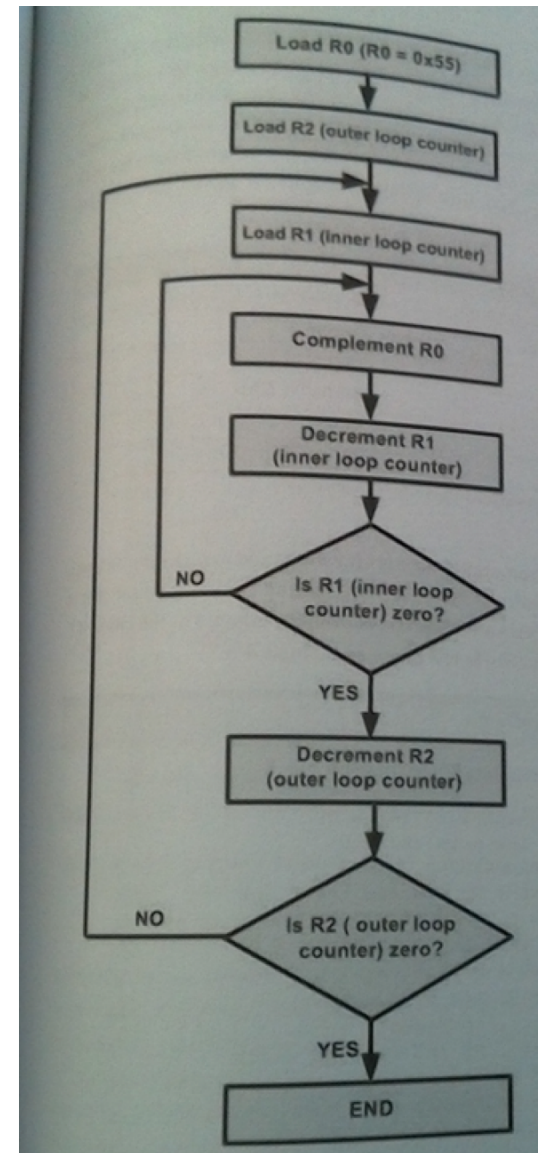
```
1 ;Write a program to clear R0 and add 9 to R0 1000 times.  
2 ;Place the sum in R4. Use BNE.  
3 LDR R2, =1000  
4 MOV R0, #0  
5 AGAIN ADD R0, R0, #9  
6 SUBS R2, R2, #1  
7 BNE AGAIN  
8 MOV R4, R0  
9 HERE B HERE  
10 END
```



Write a program to load r0 with value 0x55 and complement (negate) it 16 Billion times!

- We need to use a nested loop – note 2^{23} is 4 billion)

```
1      ;Write a program to load r0 with value 0x55
2      ;and complement (negate) it 16 Billion times!
3      MOV    R0, #0X55
4      MOV    R2, #16 ;16 X 1 BILLION
5  OUTER_LOOP LDR    R1, =1000000000
6  INNER_LOOP EOR    R0, R0, #0xFF
7          SUBS    R1, R1, #1
8          BNE     INNER_LOOP
9          SUBS    R2, R2, #1
10         BNE     OUTER_LOOP
11  HERE      B      HERE
12         END
```



Write a program to add 0x99999999 together TEN times.

- We read the result in registers R1 and R0
- Place the value in R2
- Place the counter in R3

```
1      ;Write a program to add 0x99999999 together TEN times.
2      MOV    R1,#0
3      MOV    R0,#0
4      LDR    R2,=0X99999999
5      MOV    R3,#10
6 LOOP  ADDS   R0,R0,R2
7      BCC    NEXT
8      ADD    R1,R1,#1
9 NEXT  SUBS   R3,R3,#1
10     BNE    LOOP
11 HERE  B     HERE
12     END
```


Assuming we have
5 values: 69, 87,
96, 45, and 75.
Find the highest
number and place
it in R1.

```

6  ;*****
7  ; Constants & Register Assignments
8  COUNT      RN      R0
9  MAX        RN      R1
10 POINTER    RN      R2
11 NEXT       RN      R3
12
13 ;*****
14 ; Program Section
15 ;*****
16 AREA        main, CODE, READONLY
17 THUMB
18 EXPORT      __main
19 ALIGN
20 ENTRY
21
22 __main PROC
23 MOV         COUNT, #5
24 MOV         MAX, #0
25 LDR         POINTER, =MYDATA ; Memory loc. of MyData -> Pointer
26 AGAIN LDR    NEXT, [POINTER] ; place the value in that Mem. Location in NEXT
27
28 CMP         MAX, NEXT ; compare the new data and old MAX
29 BHS         CTNU ;Branch if old MAX >= NEXT (new data)-unsigned
30 MOV         MAX, NEXT ;else replace the old_MAX with the new value
31 CTNU ADD     POINTER, POINTER, #4 ;Go to the next word (move by 4 bytes)
32 SUBS        COUNT, COUNT, #1 ;Decrement the counter
33 BNE         AGAIN ;if NOT EQUAL to zero then branch - branch until Z=0
34
35 HERE B       HERE
36 ENDP
37
38 ;-----DATA AREA in CODE starting 0x260
39 AREA        SOME_GRADES, DATA, READONLY
40 ALIGN
41 ;Reserve one word per input value
42 MYDATA DCD   0x65, 0x67, 0x63, 0x61, 0x34
43
44
45 END ;end of the program

```

Suffix	Description	Flags tested
EQ	Equal (Unsigned & Signed)	Z=1
NE	Not Equal (Unsigned & Signed)	Z=0
CS/HS	Unsigned Higher or Same	C=1
CC/LO	Unsigned Lower	C=0
MI	Minus (Negative)	N=1
PL	Plus (Positive or Zero)	N=0
VS	Overflow Set	V=1
VC	Overflow Clear	V=0
HI	Unsigned Higher	C=1 & Z=0
LS	Unsigned Lower or Same	C=0 or Z=1
GE	Signed Greater or Equal	N=V
LT	Signed Less Than	N!=V
GT	Signed Greater Than	Z=0 & N=V
LE	Signed Less than or Equal	Z=1 or N!=V
AL	Always	

Compare	Signed	Unsigned
==	EQ	EQ
≠	NE	NE
>	GT	HI
≥	GE	HS
<	LT	LO
≤	LE	LS

Instruction	Operands	Brief description	Flags
CMP	Rn, Op2	Compare	N,Z,C,V
CMN	Rn, Op2	Compare Negative	N,Z,C,V
TEQ	Rn, Op2	Test Equivalence	N,Z,C
TST	Rn, Op2	Test	N,Z,C

- **CMP** operand1 - operand2, but result not written
- **CMN** operand1 + operand2, but result not written
- **TST** operand1 & operand2, but result not written
- **TEQ** operand1 ^ operand2, but result not written