
Chapter 7.1 Programs with Loops

Objectives of this Chapter

Having studied this chapter you will be able to:

- Describe the different types of program loop structure
- Understand the 80286 assembly language conditional and unconditional jump instructions
- Understand the mechanism of relative addressing
- Determine relative addressing displacements using complementary arithmetic
- Understand the function and operation of the following 80286 Status Register Bits:
 - Carry Flag
 - Zero Flag
- Use the 80286 assembly language conditional and unconditional jump instructions in programs which make decisions
- Use the 80286 assembly language conditional and unconditional jump instructions in programs which repeat an action a given number of times
- Explain the action of the Increment and Decrement instructions

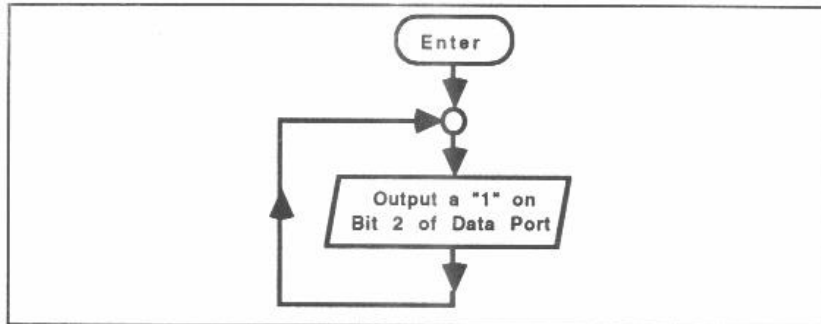
Introduction

Often it will be necessary to use a loop to **repeat** a section of a program a number of times.

There are three main types of program loop:

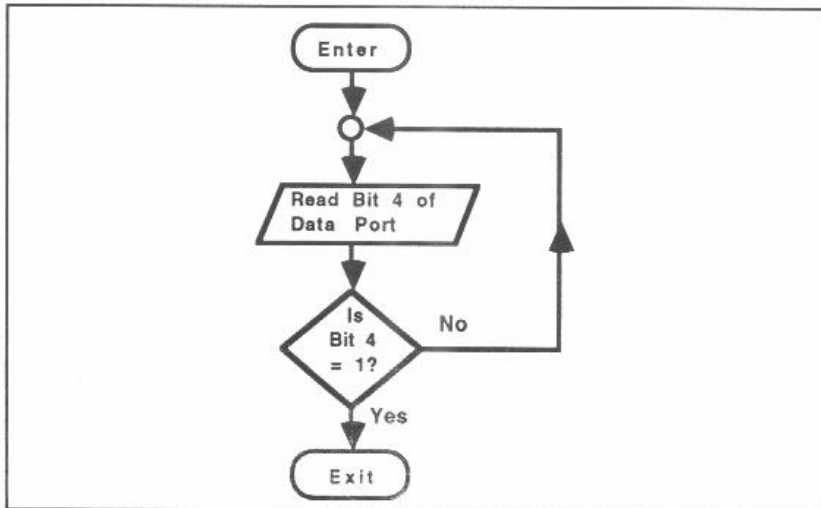
1 Repeating a program section indefinitely

For example: Output a "1" on bit 2 of a data port indefinitely.



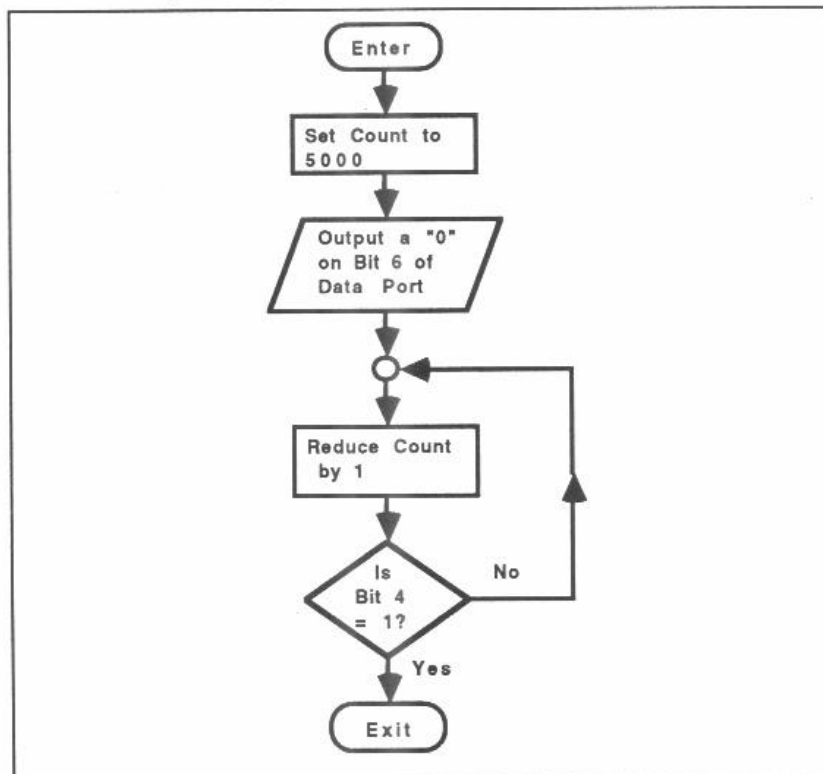
2 Repeating a program section until some predetermined condition becomes true.

For example: Waiting for a "1" at input bit 4 of a data port.



3 Repeating a program section for a predetermined number of passes.

For example: Output a "0" on bit 6 of a data port for the time it takes to repeat a loop 5000 times.



If, in this example, each pass through the loop were to take $1\mu\text{s}$, a "0" would be output on bit 6 of the data port for 5ms.

In order to write assembly language programs with loops, it will be necessary to use JUMP instructions. These can be **conditional** or **unconditional**.

Unconditional JUMP Instruction

This instruction **always** causes program execution to be continued from some point **other than** the next location in sequence by changing the Program Counter value.

Exercise 9

You have already seen the program shown below in Chapter 5.1. Use Merlin to generate this source program, assemble the object code and then transfer to PAT.

```

                ORG 0200H                ;Defines the start address for
                ;object code as 0080:0200H
VAL1 EQU 2222H  ;Defines "VAL1" as 2222H
VAL2 EQU 3333H  ;Defines "VAL2" as 3333H
MEM1 EQU 2000H  ;Defines "MEM1" as 2000H

BEGIN: MOV AX, VAL1                    ;Moves the value defined by the
                                        ;label "VAL1" into the AX
                                        ;Register
                JMP NEXT                ;Jumps forward to the label "NEXT"

LAST:  MOV DS:MEM1, AX                 ;Saves result in location "MEM1"
        MOV BX, 0000H                 ;Returns to PAT System
        MOV AH, 04H
        INT 28H

NEXT:  ADD AX, VAL2                    ;Adds the value defined by the
                                        ;label "VAL2" to the AX
                                        ;Register
        JMP LAST                       ;Jumps backward to the label "LAST"
    
```

The arrows are just to help you understand the operation of this program and should not be included in your source program.

Enter the Terminal Mode by pressing Alt-T and disassemble from location 0080:0200H. (Use the command "D 0200").

You will find that the screen shows:

```
PAT: D 0200
0080:0200 B82222      MOV  AX,2222
0080:0203 E90C00      JMP  0212
0080:0206 3E          DS:
0080:0207 A30020      MOV  [2000],AX
0080:020A BB0000      MOV  BX,0000
0080:020D B80400      MOV  AX,0004
0080:0210 CD28          INT  28
0080:0212 053333      ADD  AX,3333
0080:0215 E9EEFF      JMP  0206
PAT:
```

Examine the machine code for the second instruction:

```
0080:0203 E90C00 JMP 0212
```

Notice that the Assembly Language mnemonics for the operand (0212_H) seems to be inconsistent with the machine code (0C00_H). This is because the destination for the JUMP is **not** defined as an **address**. The value 0C00_H represents the number of bytes which must be JUMPed. This is called a **displacement** (or offset). The displacement is expressed low byte first so here it is actually 000C_H (ie 12₁₀). If you count forward 12 bytes you will find the destination for the JUMP - location 0080:0212_H. This way of specifying the destination for a JUMP is called **Relative Addressing**.

Now examine the machine code for the last instruction:

```
0080:0215 E9EEFF JMP 0206
```

This instruction must cause a JUMP **backwards** of 18₁₀ bytes. Now, the displacement here is FFEE_H so this must represent "-18₁₀" (ie -12_H).

Microprocessors are unable to interpret a minus sign. Instead they use **Complementary Arithmetic**.

Complementary Arithmetic

For any binary number there are **two** possible **complements**:

1's Complement: Found by simply **inverting** each bit.

For example: The 1's Complement of 1101_2 is 0010_2 , so -1101_2 is 0010_2 in 1's complement notation.

2's Complement: Found by adding 1 to the 1's complement.

For example: The 1's Complement of 1101_2 is 0010_2 ; the 2's complement of 1101_2 is $0010_2 + 1_2 = 0011_2$ so -1101_2 is 0011_2 in 2's complement notation.

Consider the value 0110_2 :

The 1's complement of 0110_2 is 1001_2 . Taking the 1's complement again gives 0110_2 .

Similarly, the 2's complement of 0110_2 is $1001_2 + 1_2 = 1010_2$

Taking the 2's complement again gives $0101_2 + 1_2 = 0110_2$.

So a complementary number may be converted back to an ordinary number by simply taking the complement again.

Returning to our instruction:

```
0080:0215 E9EEFF JMP 0206
```

$FFEE_H$ is $1111\ 1111\ 1110\ 1110_2$. The 1's complement is $0000\ 0000\ 0001\ 0001_2$.

The 2's complement is $0000\ 0000\ 0001\ 0001_2 + 1_2 = 0000\ 0000\ 0001\ 0010_2$ which is 12_H .

So the JUMP will be 12_H (ie 18_{10}) bytes **backward**.

You will not need to calculate displacements, since the D2000 80286 Cross Assembler will calculate them for you, using label references.

Range of Relative Addressing

The displacement for 80286 JUMP instructions is 16 bits in length.

The largest **positive** (ie forward) displacement will be 7FFF_H (0111 1111 1111 1111₂) which is 32767₁₀ bytes.

The largest **negative** (ie backward) will be 8000_H (1000 0000 0000 0000₂).

$$\begin{aligned} 8000_{\text{H}} &= + 1000\ 0000\ 0000\ 0000_2 \\ &= - 0111\ 1111\ 1111\ 1111_2 \text{ (1's complement)} \\ &= - 1000\ 0000\ 0000\ 0000_2 \text{ (2's complement)} \\ &= - 8000_{\text{H}} \\ &= - 32768_{10} \end{aligned}$$

So a backward JUMP can cover up to 32768₁₀ locations.

Conditional JUMP Instructions

A **conditional** JUMP is only taken if some predetermined condition is true. Otherwise the next instruction in sequence is executed. These instructions allow the microprocessor to take **decisions**. The conditions which are tested are the states of various bits within the **Status Register** (or Flag Register).

80286 Status Register

Each bit (or "flag") within the Status Register is a single flip-flop which can store a 0 or a 1. These flags indicate the nature of the result of the last operation. Many instructions will affect various flags. The 80286 has a number of flags but we shall only consider two for the time being: the Carry Bit and the Zero Bit:

Carry Bit

This flag is **set** (ie = 1) if the result of the last arithmetic operation is greater than the destination can hold. For example: If 50E8 278B_H is added to 814B E4CC_H the result is D234 0C57_H and there is no carry out:

$$\begin{array}{r} 50\text{E8 } 278\text{B}_{\text{H}} + \\ 814\text{B } \text{E4CC}_{\text{H}} \\ \hline \text{D234 } 0\text{C57}_{\text{H}} \end{array}$$

However, if 50E8 278B_H is added to B246 A9E7_H the result is 1 032E D172_H. Thus a carry out is generated:

$$\begin{array}{r} 50E8\ 278B_H + \\ B246\ A9E7_H \\ \hline 1\ 032E\ D172_H \end{array}$$

The carry flag is also used as a "borrow" flag when performing subtraction.

Zero Bit

This flag is **set** (ie = 1) if the result of the last operation was **zero**. For example, if the microprocessor subtracts 1234_H from 1234_H the result is 00_H and the zero flag is **set** (Z=1). If 1234_H is added to 1234_H the result is 2468_H which is non-zero so the zero flag is **cleared** (Z=0).

The action of these flags can be summarized thus:

Result Equal to Zero	(Z=1)
Result Not Equal to Zero	(Z=0)
Carry Bit Set	(C=1)
Carry Bit Cleared	(C=0)

Now, refer to the 80286 Programmer's Reference Manual for some of the instructions you have met so far. Note how the Zero and Carry Flags are affected by each instruction:

Instruction	Zero Flag	Carry Flag
MOVE	Not affected.	Not affected.
ADD	Set if result is zero, otherwise cleared.	Set if a Carry is generated, otherwise cleared.
SUB	Set if result is zero, otherwise cleared.	Set if a Borrow is generated, otherwise cleared.
JMP	Not affected.	Not affected.

Each of the Conditional Jump Instructions will test for different flags set (=1) or clear (=0). If the condition is **true**, then the displacement is added to the Program Counter and execution continues from a **point other than the next instruction**. However, if the condition is **not true** execution will continue with the **next instruction in memory**.

Some of the Conditional Jump Instructions which test the Carry and Zero Flags are:

JZ	Jump if Result Equal to Zero	(Z=1)
JNZ	Jump if Result Not Equal to Zero	(Z=0)
JC	Jump if Carry Bit Set	(C=1)
JNC	Jump if Carry Bit Cleared	(C=0)

Examples:

```
JZ 0140H ;Jumps to location 0140H in the
          ;Current Segment if the Zero
          ;Flag is set (ie if
          ;Zero Flag = 1)

JNZ KBDSCL ;Jumps to location specified
          ;by the label KBDSCL if the
          ;Zero Flag is clear (ie if
          ;Zero Flag = 0)

JC NXTWD ;Jumps to location specified
          ;by the label NXTWD if the
          ;Carry Flag is set (ie if
          ;Carry Flag = 1)

JNC MINVAL ;Jumps to location specified
          ;by the label MINVAL if the
          ;Carry Flag is clear (ie if
          ;Carry Flag = 0)
```

There are a number of other flags which may be tested but we shall only consider the carry and zero flags for the time being.

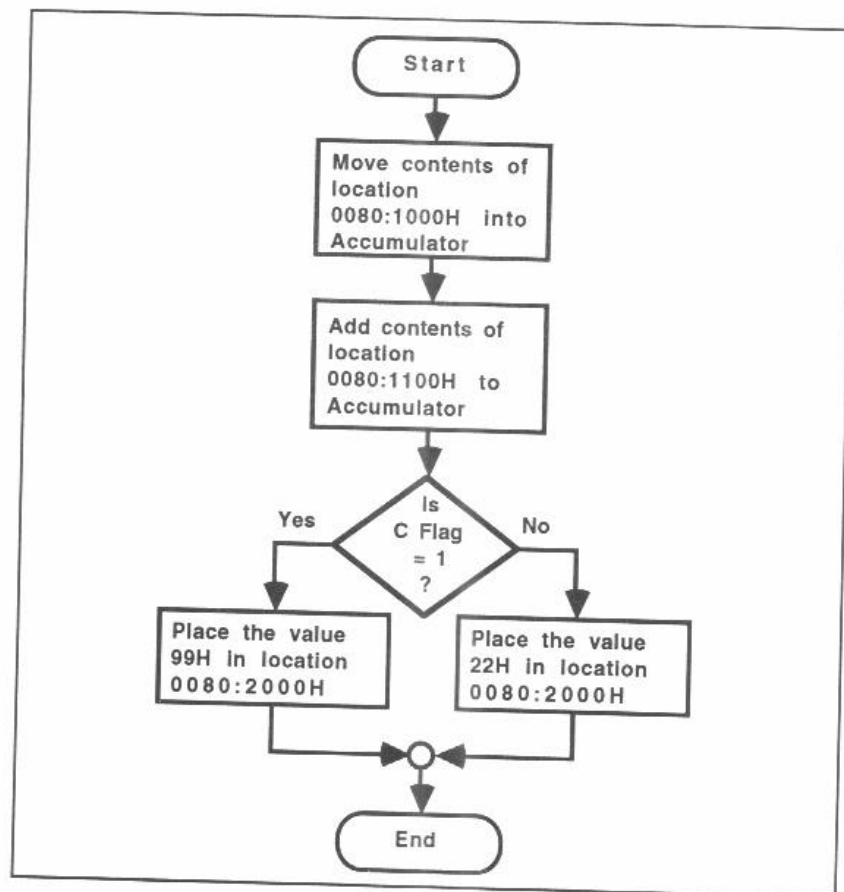
Exercise 10

Write a program which will add the words at locations 0080:1000H and 0080:1100H. If the result is greater than FFFF_H, the value 99_H should be placed in location 0080:2000H. If the result is **not** greater than FFFF_H, the value 22_H should be placed in location 0080:2000H.

Solution:

This problem requires the carry flag to be tested following the addition and then a **marker** value to be saved to indicate the **status** of the result.

Flowchart:



The assembly language program will be:

```
ORG 0100H                ;Defines the start address for
                          ;object code as 0080:0100H
MOV AX,DS:1000H          ;Moves the contents of location
                          ;0080:1000H into the
                          ;Accumulator
ADD AX,DS:1100H          ;Adds the contents of location
                          ;0080:1100H to the Accumulator
JC GREATR                ;If the Carry Flag is set, the
                          ;result must be greater than
                          ;FFFFH, so Jump to label GREATR
LESS: MOV DS:2000H,22H    ;The Carry Flag is not set, so
                          ;the result must be not
                          ;greater than FFFFH. Therefore
                          ;save the marker value 22H in
                          ;location 0080:2000H.
                          ;Task completed so Jump to the
                          ;end
JMP FINISH
GREATR: MOV DS:2000H,99H  ;Saves the marker value 99H in
                          ;location 0080:2000H
FINISH: MOV BX,0000H      ;Returns to PAT system
        MOV AH,04H
        INT 28H
```

Notice how the state of the carry flag determines which branch is taken and hence which marker value is saved in location 0080:2000H.

Use Merlin to create this source program and then assemble the object code program. Transfer to PAT and execute.

Experiment with known values in locations 0080:1000H and 0080:1100H to verify correct operation.

Practical Assignment

- 6 Write a program which will examine the contents of location 0080:1200H. If the contents are 0000H, location 0080:2100H should be loaded with 33H. If the contents are **non-zero** then 0080:2100H should be loaded with 55H.

Hint: The Move instruction will **not** affect the Zero Flag. An easy way to overcome this problem is to include an instruction which performs the logical OR of the value with 0000H. For example `OR AX, 0000H` will cause the Zero Flag to be set if the AX Register contains 0000H.

Loop Counters

So far programs have been decision-making rather than repeated loops. Consider the problem of repeating a section of a program a given number of times. These types of programs often use a register or memory location as a **loop counter**. The loop counter is **decremented** (decreased by 01H) on each pass through the loop and tested for zero. When the counter reaches zero the program exits from the loop and continues. This is a fundamental technique in assembly language programming.

Increment and Decrement Instructions

The Increment (INC) and Decrement (DEC) instructions allow the contents of a register or memory location to be increased or decreased by 1 respectively.

For example:

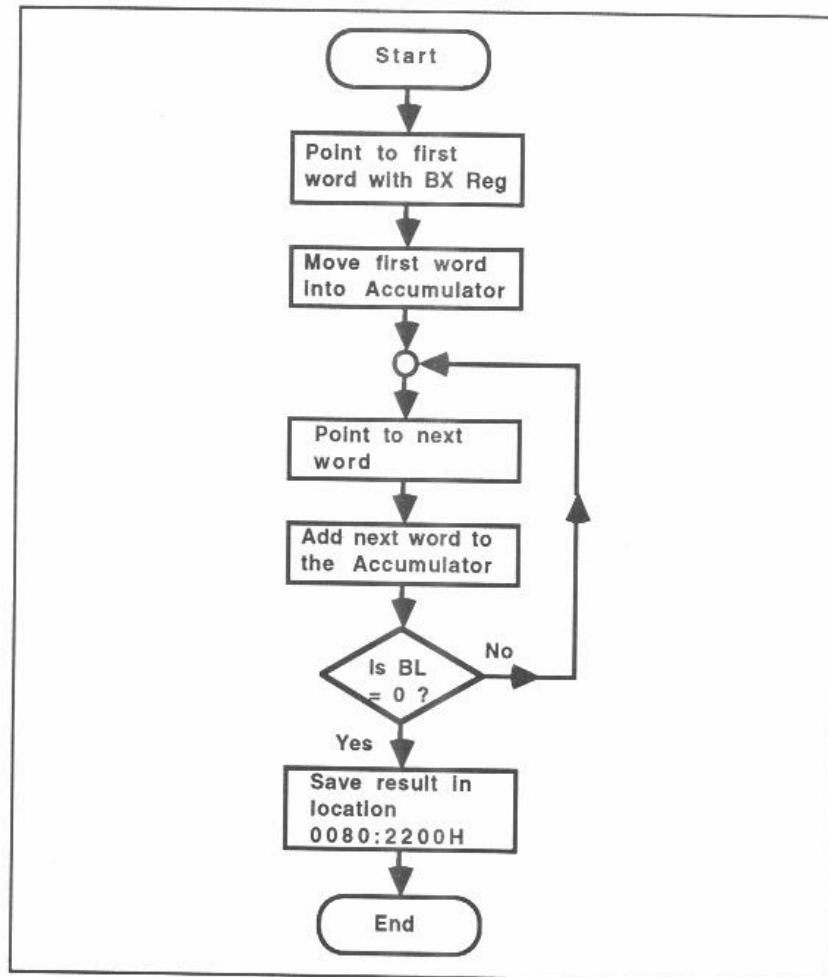
```
INC  AX                ;Increases the contents of the  
                        ;Accumulator by 1  
DEC  CX                ;Reduces the contents of the  
                        ;CX Register by 1
```

Exercise 11

Write a program which will add the words at all locations between 0080:1300H and 0080:130EH. The result must be saved in location 0080:2200H.

Solution:

The BX Register can be used as a convenient loop counter:



A source program for this is shown below:

```
ORG 0300H ;Defines the start address for
           ;object code as 0080:0300H
MOV BX,130EH ;Uses BX Register to point to
             ;first word
MOV AX,WORD PTR [BX] ;Moves the first word into the
NEXT: DEC BL ;Reduces the low byte of the
       DEC BL ;Count by 1H
       ;Reduces the low byte of the
       ;Count by 1H again to point to
       ;next word
ADD AX,WORD PTR [BX] ;Adds the next word to the
                     ;Accumulator
OR BL,00H ;ORs BL Register with 00H to
          ;condition Flags
JNZ NEXT ;If Zero Flag is not set, Jump
         ;back to label NEXT to point to
         ;next word
MOV DS:2200H,AX ;Saves the result in the memory
               ;location 0080:2200H
MOV BX,0000H ;Returns to PAT system
MOV AH,04H
INT 28H
```

Create this source program and assemble the object code program. Transfer the object program to PAT and place known values in locations 0080:1300H to 0080:130FH.

Run the program and verify correct operation by examination of location 0080:2200H. Remember that words are stored **low** byte first.

Practical Assignments

- 7 Write a program which will fill each byte between locations 0080:1400_H and 0080:14FF_H with the value 77_H.
- 8 Write a program which will load each byte between 0080:1500_H and 0080:15FF_H according to the table below:

Location	Contents
0080:1500 _H	00 _H
0080:1501 _H	01 _H
0080:1502 _H	02 _H
0080:1503 _H	03 _H
0080:1504 _H	04 _H
and so on up to:	
0080:15FD _H	FD _H
0080:15FE _H	FE _H
0080:15FF _H	FF _H

Notes:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Student Assessment 7

1. The type of JUMP which is **always** taken is called a
 - a Conditional Jump
 - b Direct Jump
 - c Indirect Jump
 - d Unconditional Jump
2. If the instruction at location 0080:2418₁₁ is JZ 2432H, the displacement added to the Program Counter is:
 - a 16₁₁
 - b 17_H
 - c 18₁₁
 - d 19_H
3. The largest **positive** 16-bit offset for relative addressing is:
 - a 32765₁₀
 - b 32766₁₀
 - c 32767₁₀
 - d 32768₁₀
4. The Carry Flag is set when the result of the last 16 bit operation is:
 - a zero
 - b non-zero
 - c less than 16 bits
 - d greater than 16 bits
5. The CX Register contains the value 1234_H. After the instruction INC CX, the CX Register will contain:
 - a 1233₁₁
 - b 1234_H
 - c 1235₁₁
 - d 1236_H

Chapter 7.2 Further Programs with Loops

Objectives of this Chapter

Having studied this chapter you will be able to:

- Describe the fundamental operation of the following 80286 LOOP instructions:
 LOOP
 LOOPZ
 LOOPNZ
- Describe the operation of the 80286 COMPARE instruction
- Explain how the COMPARE instruction affects the Zero and Carry flags.
- Write programs using the LOOP and COMPARE instructions.

Introduction

In the previous chapter you saw probably the most common loop structure. This is where a "count" is increased or decreased and then tested. A Jump may then occur, depending upon the result. The 80286 has a set of special instructions which **combine** these two operations. These are the LOOP instructions.

Loop Instructions

These instructions are another type of conditional Jump. All Loop instructions use the CX Register as a Loop Counter. The Loop Counter is decremented and its contents examined each time the instruction is executed.

Different types of LOOP instruction will take different actions thus:

LOOP

Decrement the CX Register and then Jump to destination **until** CX = 0000_{1H}.

LOOPZ

Decrement the CX Register and then Jump to destination **until** CX = 0000_{1H} and Zero Flag is **Set**.

LOOPNZ

Decrement the CX Register and then Jump to destination **until** CX = 0000_{1H} and Zero Flag is **Clear**.

None of the Loop instructions will themselves affect any flags.

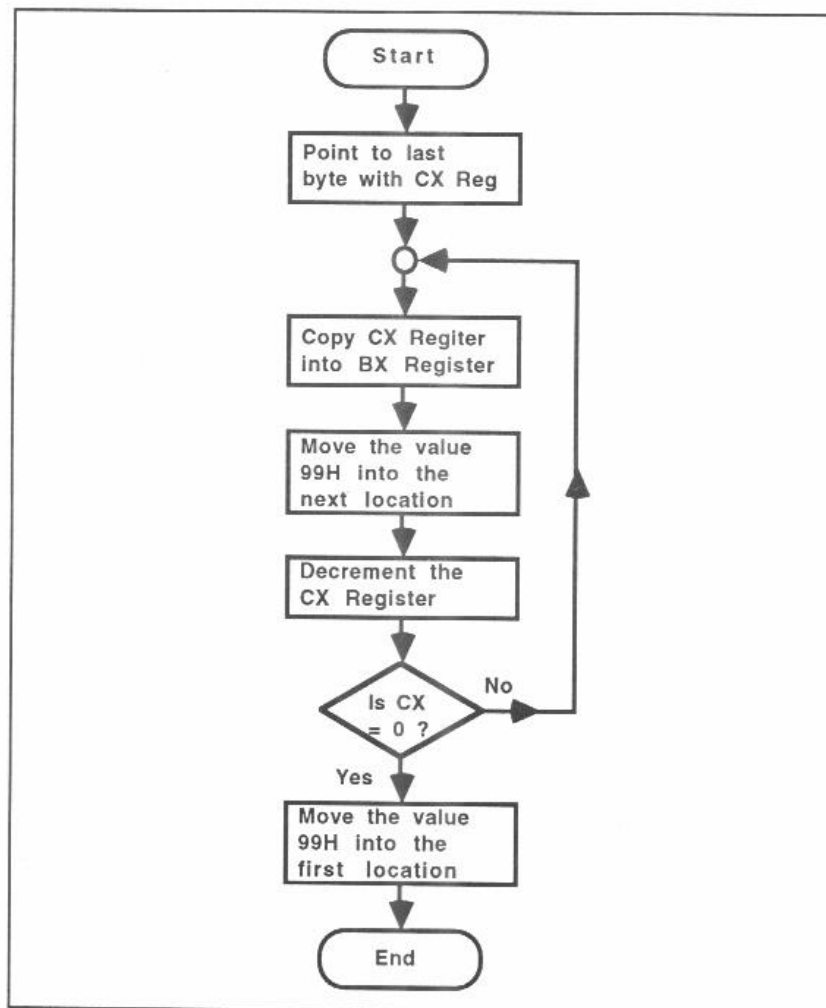
Loop instructions can be used in place of the Decrement/Jump if Non-Zero method introduced in the last chapter.

Exercise 12

Write a program which will fill all of the locations between 0080:0500H and 0080:05FFH with the byte 99H.

Solution:

Flowchart:



The assembly language program will be:

```
ORG 0100H ;Defines the start address for
;object code as 0080:0100H
MEMORY EQU 0500H ;Defines "MEMORY" as 0500H
MOV CX,00FFH ;Points to last byte with the
;CX Register
NEXT: MOV BX,CX ;Copies the CX Register into
;the BX Register to point to
;the next byte
MOV BYTE PTR MEMORY[BX],99H ;Moves the value 99H into the
;next location
LOOP NEXT ;Decrement the CX Register. If
;CX is non zero, jump back to
;label "NEXT"
MOV BX,CX ;Points to first location
MOV BYTE PTR MEMORY[BX],99H ;Moves the value 99H into the
;last location
MOV BX,0000H ;Returns to PAT system
MOV AH,04H
INT 28H
```

Once you have generated an object program, transfer to PAT and verify for correct operation by examining locations 0080:0500H to 0080:05FFH before and after execution.

Practical Assignment

- 9 A series of non-zero bytes is stored from locations 0080:1000H onward. This series is terminated with the byte 00H. Write a program which will copy this series to locations 0080:2000H onward.

Up to now we have concentrated upon detecting if the contents of a register are **zero** or over **FFFFH**. It is also possible to detect **any given value**. This is achieved by means of the COMPARE instruction.

The Compare Instruction

Consider the problem of testing the Accumulator to see if it contains the word 3210_H. Subtracting 3210_H from the Accumulator would cause the zero flag to be set if the accumulator had contained 3210_H.

This could be achieved as follows:

```
      SUB AX,3210H           ;Subtracts the value 3210H from
                           ;the Accumulator
      JZ  PASS              ;If the Zero Flag is set, Jump
                           ;to the label "PASS"
FAIL:  MOV DX,0001H        ;Zero flag is not set so place
                           ;marker value 0001H in DX
                           ;Register
      JMP START            ;Returns to start
PASS:  MOV DX,FFFEH        ;Zero flag is not set so place
                           ;marker value FFFEH in DX
                           ;Register
```

The DX Register will be loaded with either FFFE_H or 0001_H, to indicate an Accumulator value of 3210_H or non-3210_H respectively.

The difficulty with this technique is that it destroys the contents of the Accumulator. Now, since this is a very common problem in assembly language programming, the 80286 provides a special instruction which operates like subtraction but does **not** destroy the register contents. This is the COMPARE instruction.

When a COMPARE is executed, the result of the subtraction is **lost** but the status flags are conditioned to reflect the **type** of result (eg zero flag set/clear, etc).

The 80286 COMPARE instruction (CMP) subtracts the contents of the source from the destination and conditions flags depending upon the result. The contents of the source and destination are **unaffected** by this instruction.

For example:

```
CMP AL, 37H
```

This instruction subtracts the value 37_H from the Accumulator but does **not** place the result in the Accumulator.

If the value 37_H is **equal** to the contents of the Accumulator:

Zero Flag = 1 **Carry Flag= 0**

If the value 37_H is **greater than** the contents of the Accumulator:

Zero Flag = 0 **Carry Flag= 0**

If the value 37_H is **less than** the contents of the Accumulator:

Zero Flag = 0 **Carry Flag= 1**

So the CMP instruction can be used in programs to discover whether a register or memory location holds a **specific**, non-zero value. This allows programs to be produced which make the following decisions:

- Is A = B ?
- Is A > B ?
- Is A < B ?

Notes:

.....

.....

.....

.....

.....

.....

.....

.....

.....

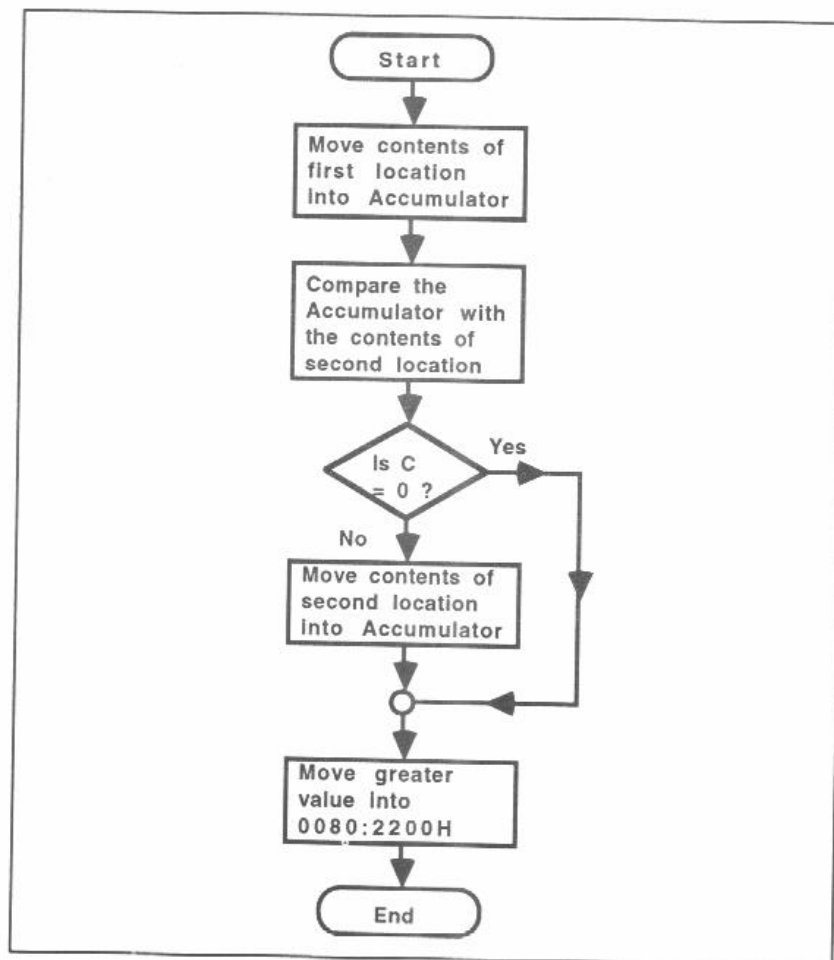
.....

Exercise 13

Write a program which will inspect the data words at locations 0080:2000_H and 0080:2002_H. The greater of these two words should then be saved in location 0080:2004_H.

Solution:

Flowchart:



Source Program:

```

ORG 0300H                ;Defines the start address for
                        ;object code as 0080:0300H
MOV AX,DS:2000H          ;Moves the contents of first
                        ;location into the Accumulator
CMP AX,DS:2100H          ;Compares the second location
                        ;with the first
JNC LESS                 ;If the carry flag is clear
                        ;then First is greater than
                        ;Second so Jump to label "LESS"
MOV AX,DS:2100H          ;Carry flag is set so second
                        ;value is the greater.
                        ;Therefore place greater value
                        ;in the Accumulator
LESS: MOV DS:2200H,AX    ;Saves the greater value in
                        ;location 0080:2200H
MOV BX,0000H             ;Returns to PAT system
MOV AH,04H
INT 28H
    
```

Notice that at the JNC instruction, the contents of location 0080:2000H are already in Data Register 0. If this is the greater, it can simply be saved in location 0080:2200H. Use this source program to produce an object program and execute using the PAT to verify correct operation.

Notes:

.....

.....

.....

.....

.....

.....

.....

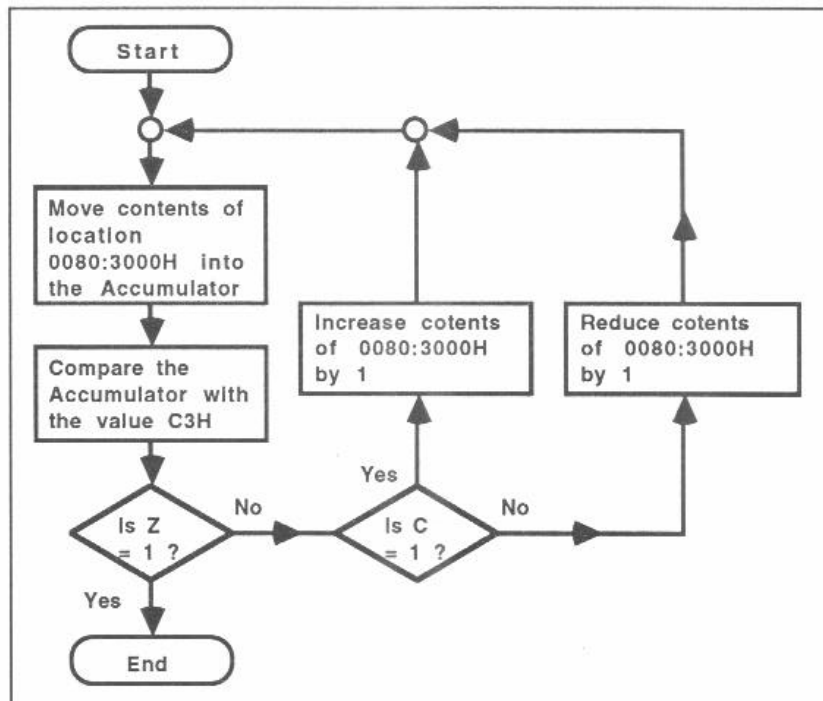
.....

Exercise 14

Memory location 0080:3000H may contain any byte. Write a program which will increase or decrease the contents of this location, in steps of 1, until the byte stored becomes C3H.

Solution:

Flowchart:



The source program will be:

```

                ORG 0400H                ;Defines the start address for
CHECK:          MOV AL, BYTE PTR DS:3000H ;object code as 0080:0400H
                CMP AL, 0C3H            ;Moves contents of location
                JZ  EQUAL                ;into Accumulator
                JC  LESS                 ;Compares Accumulator with the
                JZ  EQUAL                ;value C3H
                JC  LESS                 ;If the Zero Flag is set, byte
                JZ  EQUAL                ;equals C3H so Jump to end
                JC  LESS                 ;If the Carry Flag is set, byte
                JZ  EQUAL                ;is less than C3H, so Jump to
                JC  LESS                 ;label "LESS"
GREATR:         DEC BYTE PTR DS:3000H    ;Carry Flag is clear so reduce
                JMP CHECK                ;byte by 1
                JMP CHECK                ;Jumps to label "CHECK" to test
LESS:           INC BYTE PTR DS:3000H    ;byte again
                JMP CHECK                ;Increases byte by 1
                JMP CHECK                ;Jumps to label "CHECK" to test
EQUAL:          MOV BX, 0000H            ;byte again
                MOV AH, 04H              ;Byte is now C3H so returns to
                INT 28H                   ;PAT system

```

Once you have generated an object program, transfer to PAT and verify correct operation by loading location 0080:0500H with suitable values and checking after execution.

Practical Assignments

- 10 Write a program which will examine the data byte at location 0080:2300H. Location 0080:2400H should then be loaded with a marker value according to the byte found thus:
If the byte in location 0080:2300H is:
less than 7FH, load location 0080:2400H with 01H
equal to 7FH, load location 0080:2400H with AAH
greater than 7FH, load location 0080:2400H with FEH
- 11 Write a program which will inspect the words at locations 0080:2500H, 0080:2600H and 0080:2700H. The largest of these should be saved in location 0080:2800H.

Student Assessment 8

1. The Register used by Loop instructions as the loop counter is the:
 - a AX Register
 - b BX Register
 - c CX Register
 - d DX Register
2. The Loop instruction which will cause a Jump to the destination until the count is zero and the Zero Flag is clear is:
 - a LOOP
 - b LOOPNZ
 - c LOOPS
 - d LOOPZ
3. The instruction which will subtract the contents of a source from a destination and set or clear flags accordingly, without affecting the source or destination is:
 - a Compare
 - b Loop
 - c Subtract
 - d Test
4. Following a COMPARE instruction, both the Zero and Carry Flags are **clear** (ie: = 0). This indicates that:
 - a the source contains zero
 - b the source and destination are equal
 - c the source smaller than the destination
 - d the source is greater than the destination
5. If the destination is **smaller** than the source for a COMPARE instruction, the Zero and Carry Flags will be:
 - a Z = 0 C = 0
 - b Z = 0 C = 1
 - c Z = 1 C = 0
 - d Z = 1 C = 1

Chapter 7.3 Test and Logic Instructions

Objectives of this Chapter

Having studied this chapter you will be able to:

- Describe the function of the basic Logical Operators:
AND
OR
Exclusive OR
- Apply the basic Logical Operators to binary data
- Write programs which use the 80286 AND, OR and XOR instructions to test bits within a register or memory location
- Describe the operation of the 80286 TEST Instruction
- Use the 80286 TEST instruction to test a number of bits within a register or memory location
- Describe the principles and applications of Shift and Rotate Instructions
- Write programs which use the 80286 SHIFT and ROTATE instructions to manipulate binary data

Introduction

You have learned in the previous chapter how to detect if contents of a location or register are equal to any given **value**. In this chapter you will learn how to test **individual bits** within a memory location or register. The bit test instructions can be useful in Input/Output Programming, which you will encounter in the next Chapter.

Logical Operators

Logical instructions can be used to test **groups** of bits. This requires the use of **logical operators**. You will already be familiar with the ways in which some **arithmetic** operators can be applied to data (eg ADD, SUB, etc). It is also possible to apply **logical** operators to data (eg AND, OR, EXCLUSIVE-OR).

Logical AND

Recall	0 AND 0 = 0	0 . 0 = 0
	0 AND 1 = 0	0 . 1 = 0
	1 AND 0 = 0	1 . 0 = 0
	1 AND 1 = 1	1 . 1 = 1

To AND together two binary numbers, the AND operator is applied bit by bit.

For example:

$$\begin{array}{r}
 0110 \\
 0101 \\
 \hline
 0100
 \end{array}$$

$0 . 1 = 0$
 $1 . 0 = 0$
 $1 . 1 = 1$
 $0 . 0 = 0$

So $0110_2 \cdot 1101_2 = 0100_2$

Notice that any given bit in the result can only be 1 if **both** of the numbers have a 1 in that position. This property can be used to test for bits at 1 within a register or location.

For example: 1099_H ANDed with a mask 57F0_H:

```
1099H = 0001 0000 1001 10012
57F0H = 0101 0111 1111 00002
          0001 0000 1001 00002 = 1090H
```

This technique can be used to test a **number** of bits within a register or memory location.

The 80286 AND instruction can operate upon memory, register or immediate data.

For example:

```
AND AX,10FFH           ;ANDs the Accumulator with the
                        ;value 10FFH
AND BL,CL              ;ANDs the BL and CL Registers
AND DX,DS:2000H        ;ANDs the DX Register with the
                        ;contents of location 2000H
                        ;within the current segment
```

Notes:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

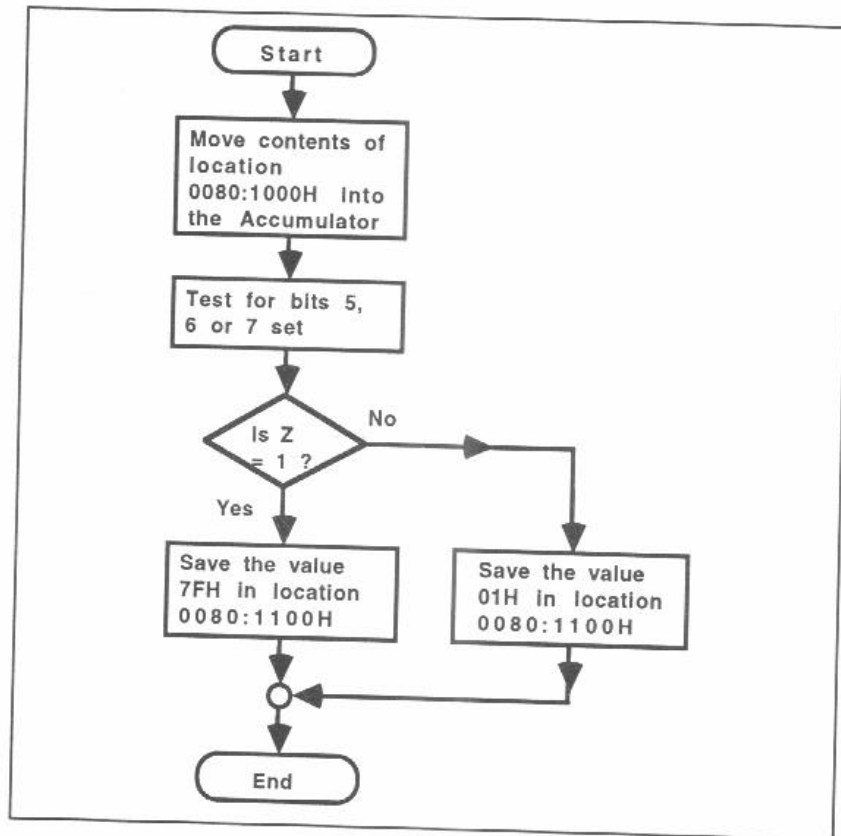
.....

Exercise 15

Write a program which will examine the byte at location 0080:1000H. A marker value of 10H should be saved in location 1100H if **any** of the bits 5, 6 and 7 in location 0080:1000H are **set**. Otherwise, a marker value of 7FH should be placed in location 0080:1100H.

Solution:

Flowchart:



Source Program:

The mask must enable bits 5,6 and 7 of the destination. The mask required is therefore 1110 0000₂ (ie:E0_H).

```
ORG 0100H                ;Defines the start address for
                          ;object code as 0080:0100H
MOV AL,BYTE PTR DS:1000H ;Moves the contents of location
                          ;0080:1000H into the
                          ;Accumulator
AND AL,0E0H              ;ANDs the Accumulator with the
                          ;mask E0H (1110 00002) to test
                          ;for bits 5, 6 or 7 set
JNZ TRUE                  ;If the Zero Flag is clear, at
                          ;least one of bits 5, 6 or 7 is
                          ;set so Jump to label "TRUE"
FALSE: MOV BYTE PTR DS:1100H,07FH ;The Zero Flag is set so none
                          ;of the bits 5, 6 or 7 are set.
                          ;Hence save marker for no bits
                          ;set.
JMP FINISH                ;Task completed, so Jump to the
                          ;end
TRUE:  MOV BYTE PTR DS:1100H,01H ;Saves the marker for one or
                          ;more of bits 5, 6 or 7 set
FINISH: MOV BX,0000H        ;Returns to PAT system
        MOV AH,04H
        INT 28H
```

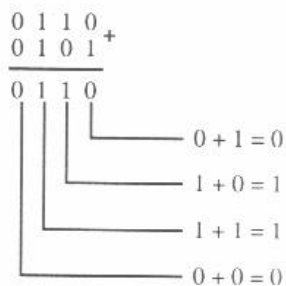
Use this source program to produce an object program. Run it on the PAT and ensure correct operation.

Logical OR and Exclusive OR

Recall	0 OR 0 = 0	0 + 0 = 0
	0 OR 1 = 1	0 + 1 = 1
	1 OR 0 = 1	1 + 0 = 1
	1 OR 1 = 1	1 + 1 = 1

To OR together two binary numbers, the OR operator is applied bit by bit.

For example:



So $0110_2 + 1101_2 = 0110_2$

Notice that any given bit in the result can only be 0 if **both** of the numbers have a 0 in that position.

80286 assembly language allows the OR and Exclusive-OR (XOR) operators to be applied to data in similar ways to the logical AND operator.

For example:

OR	AX, BX	; ORs the Accumulator with the ; BX Register
XOR	CL, BYTE PTR DS:2400H	; Exclusive ORs the CL Register ; with the byte at location ; 0080:2400H
OR	DX, C101H	; ORs the DX Register with the ; value C101H

We have already used the OR instruction to condition the Flags (eg OR AX, 0000H).

A useful property of the Exclusive OR function is that of **inverting** data.

For example:

$$\begin{array}{r} 0110 \\ 1111 \oplus \\ \hline 1001 \end{array}$$

So, exclusive ORing with 1111₂ gives the **logical inverse**. This can be useful in detecting 0's (rather than 1's). If the test word is first inverted, 1's in the inverse correspond to 0's in the actual test word. These can then easily be detected.

The Bit Test Instruction

The Bit Test instruction (TEST) is rather like a logical Compare. The contents of a Register or memory location are ANDed with a mask. However, neither the source nor destination are modified by this instruction; only the Flags are affected. So this instruction can be used in place of AND if the contents of Registers or memory locations are to be preserved.

Examples:

TEST AL, FIRST	;ANDs the Accumulator with the ;mask defined by label "FIRST". ;Contents of Accumulator are ;unaffected.
TEST BL, CL	;ANDs the Accumulator with the ;mask contained in the CL ;Register. Contents of BL and ;CL Registers are unaffected.
TEST DX, DS:1350H	;ANDs the DX Register with the ;mask contained in location ;1350 _H in the current segment. ;Contents of DX Register and ;memory location are ;unaffected.

Exercise 16

Repeat Exercise 15 , using a TEST instruction rather than an AND.

Solution:

The flowchart for this exercise will be the same as in the previous exercise (Page 130).

Source Program:

```
ORG 0200H ;Defines the start address for
;object code as 0080:0200H
MOV AL,BYTE PTR DS:1000H ;Moves the contents of location
;0080:1000H into the
;Accumulator
TEST AL,0E0H ;ANDs the Accumulator with the
;mask F0H (1110 00002) to test
;for bits 5, 6 or 7 set
JNZ TRUE ;If the Zero Flag is clear, at
;least one of bits 5, 6 or 7 is
;set so Jump to label "TRUE"
FALSE: MOV BYTE PTR DS:1100H,07FH ;The Zero Flag is set so none
;of the bits 5, 6 or 7 are set.
;Hence save marker for no bits
;set.
JMP FINISH ;Task completed, so Jump to the
;end
TRUE: MOV BYTE PTR DS:1100H,01H ;Saves the marker for one or
;more of bits 5, 6 or 7 set
FINISH: MOV BX,0000H ;Returns to PAT system
MOV AH,04H
INT 28H
```

Edit your source program for Exercise 15 to produce the source program above. Assemble and then run to verify correct operation.

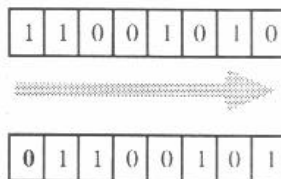
Practical Assignment

- 12 Write a program which will inspect the byte at location 0080:1200H. If any of bits 0, 1, 2 or 3 are **clear**, load 0080:1300H with F0H. Otherwise load 0080:1300H with 77H. .

Shift Instructions

A logical shift involves each bit within a register moving one place to the left or right (depending upon the direction of the shift) and usually a zero is shifted into the register and the bit at the other end is lost.

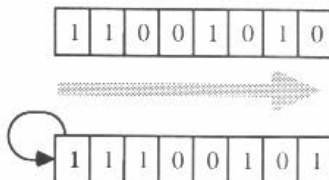
For example: a shift right of a register holding $1100\ 1010_2$:



Each bit moves one place to the **right**. A zero moves into the most significant bit position and the least significant bit is **lost**. So, the register will change to $0110\ 0101_2$. This type of shift is called a **Logical Shift**.

Notice that a Shift Left corresponds to **division** by 2 (in the same way that moving the decimal point left gives division by 10 in the denary system). Similarly, a Shift Right performs **multiplication** by 2.

The other type of shift is the **Arithmetic Shift**. For example, an arithmetic shift right of a register holding $1100\ 1010_2$:



Again, each bit moves one place to the right and the least significant bit is lost. However, unlike the logical shift, the most significant bit is **retained**. So, the register will change to $1110\ 0101_2$.

In 80286 Assembly Language both the target register or memory location and the number of shifts required is specified.

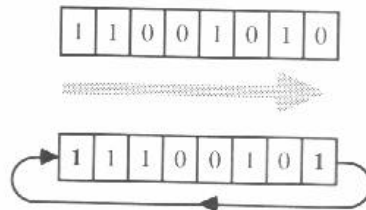
For example:

```
SAR AL,05H           ;Arithmetic Shift Right of the
                    ;Accumulator 5 times
SHL DS:2340H,08H    ;Logical Shift Left of memory
                    ;location 2340H, in the current
                    ;segment, 8 times
```

Rotate Instructions

These are similar to shift instructions, except that instead of one bit being lost and a zero shifting in, the last bit is shifted back in at the beginning.

For example: a register holding $1100\ 1010_2$:



Each bit moves one place to the right and the least significant bit moves to the most significant position.

As with the Shift instructions, the number of Rotates is specified as well as the target register or memory location.

For example:

```
ROL  BL,CL           ;Rotates the BL Register Left.  
                        ;The number of Rotates is found  
                        ;in the CL Register.  
ROR  AX,01H          ;Rotates the Accumulator Left  
                        ;once
```

Shift and Rotate instructions are used in the Conversion of Serial and Parallel Data; in Multiplication/Division and in Matching of Bit Patterns

Details of the 80286 Shift and Rotate instructions can be found in the 80286 Programmers' Reference Manual.

Practical Assignment

- 13 The word at location $0080:1400_H$ is initially $F800_H$. Write a program which will rotate this word **right** until bit 12 is **clear**.

Student Assessment 9

1. When the binary number $1001\ 1101_2$ is logically ANDed with the mask $1111\ 0000_2$, the result is:
 - a $0110\ 0000_2$
 - b $0110\ 0010_2$
 - c $1001\ 0000_2$
 - d $1001\ 1101_2$
2. The Accumulator contains the word 4500_H . After the instruction "XOR AX, 0FFH" is executed, the contents of the Accumulator become:
 - a 0000_H
 - b 4500_H
 - c $BAFF_H$
 - d $FFFF_H$
3. The Accumulator contains the word 4500_H . After the instruction "TEST AX, 0FFH" is executed, the contents of the Accumulator become:
 - a 0000_H
 - b 4500_H
 - c $BAFF_H$
 - d $FFFF_H$
4. A zero in bit 5 of the BL Register is to be detected. The correct program sequence is:

<ol style="list-style-type: none"><input type="checkbox"/> a OR BL, 00H TEST BL, 20H<input type="checkbox"/> b OR BL, 0FFH TEST BL, 20H	<ol style="list-style-type: none"><input type="checkbox"/> c XOR BL, 00H TEST BL, 20H<input type="checkbox"/> d XOR BL, 0FFH TEST BL, 20H
--	--
5. A Register contains the value $B7_H$. If this register is Rotated Left 3 times it will become:
 - a 16_H
 - b $B8_H$
 - c BD_H
 - d $F6_H$