15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 2

ARITHMETIC & LOGIC INSTRUCTIONS AND PROGRAMS

Manoj Prasanna K.
Dept. of CSE, VCET.

UNSIGNED ADDITION & SUBTRACTION

- Unsigned numbers are defined as data in which all the bits are used to represent data.
  - Operand can be between 00 and FFH (0 to 255 decimal) for 8-bit data
  - Between 0000 and FFFFH (0 to 65535 decimal) for 16-bit data

Addition of Unsigned Numbers

- Instructions ADD and ADC
  - ZF, SF, AF, CF, or PF bits of the flag register
  - OF flag is used only in signed number operations

Show how the flag register is affected by

```
MOV   AL,0F5H
ADD   AL,0BH
```

Solution:

```
      F5H        1111 0101
+     0BH      + 0000 1011
     100H        0000 0000
```

After the addition, the AL register (destination) contains 00 and the flags are as follows:
CF = 1, since there is a carry out from D7
SF = 0, the status of D7 of the result
PF = 1, the number of 1s is zero (zero is an even number)
AF = 1, there is a carry from D3 to D4
ZF = 1, the result of the action is zero (for the 8 bits)

CASE1: Addition of Individual Byte & Word Data

Write a program to calculate the total sum of 5 bytes of data. Each byte represents the daily wages of a worker. This person does not make more than $255 (FFH) a day. The decimal data is as follows: 125, 235, 197, 91, and 48.

```
TITLE       PROG3-1A (EXE) ADDING 5 BYTES
PAGE        60,132
.MODEL SMALL
.STACK 64
;--------------------------------
            .DATA
COUNT       EQU    05
DATA        DB          125,235,197,91,48
            ORG    0008H
SUM         DW     ?
;--------------------------------
      .CODE
MAIN PROC   FAR
     MOV    AX,@DATA
     MOV    DS,AX
     MOV    CX,COUNT            ;CX is the loop counter
     MOV    SI,OFFSET DATA      ;SI is the data pointer
     MOV    AX,00               ;AX will hold the sum
BACK:ADD    AL,[SI]             ;add the next byte to AL
     JNC    OVER          ;if no carry, continue
     INC    AH            ;else accumulate carry in AH
OVER:INC    SI            ;increment data pointer
     DEC    CX            ;decrement loop counter
     JNZ    BACK          ;if not finished, go add next byte
     MOV    SUM,AX        ;store sum
     MOV    AH,4CH
     INT    21H           ;go back to OS
MAIN ENDP
     END    MAIN
```

- due to pipelining it is strongly recommended that the following lines of the program be replaced

Replace these lines	With these lines
`BACK: ADD AL,[SI]`	`BACK: ADD AL,[SI]`
`JNC OVER`	`ADC AH,00 ;add 1 to AH if CF=1`
`INC AH`	`INC SI`
`OVER: INC SI`	

- The instruction "*JNC OVER*" has to empty the queue of pipelined instructions and fetch the instructions from the OVER target every time the carry is zero (CF = 0).
- Hence, the "*ADC AH, 00*" instruction is much more efficient.

Write a program to calculate the total sum of five words of data. Each data value represents the yearly wages of a worker. This person does not make more than $65,555 (FFFFH) a year. The decimal data is as follows: 27345, 28521, 29533, 30105, and 32375.

```
TITLE       PROG3-1B (EXE) ADDING 5 WORDS
PAGE        60,132
.MODEL SMALL
.STACK 64
;-----------------
        .DATA
COUNT   EQU       05
DATA    DW        27345,28521,29533,30105,32375
        ORG       0010H
SUM     DW        2 DUP(?)
;-----------------
      .CODE
MAIN PROC  FAR
     MOV   AX,@DATA
     MOV   DS,AX
     MOV   CX,COUNT            ;CX is the loop counter
     MOV   SI,OFFSET DATA      ;SI is the data pointer
     MOV   AX,00               ;AX will hold the sum
     MOV   BX,AX               ;BX will hold the carries
BACK:ADD   AX,[ SI]            ;add the next word to AX
     ADC   BX,0          ;add carry to BX
     INC   SI            ;increment data pointer twice
     INC   SI            ;to point to next word
     DEC   CX            ;decrement loop counter
     JNZ   BACK          ;if not finished, continue adding
     MOV   SUM,AX              ;store the sum
     MOV   SUM+2,BX      ;store the carries
     MOV   AH,4CH
     INT   21H           ;go back to OS
MAIN ENDP
     END   MAIN
```

CASE2: Addition of Multiword Numbers

```
TITLE       PROG3-2 (EXE) MULTIWORD ADDITION
PAGE        60,132
.MODEL SMALL
.STACK 64
;-----------------------------------
      .DATA
DATA1 DQ    548FB9963CE7H
      ORG   0010H
DATA2 DQ    3FCD4FA23B8DH
      ORG   0020H
DATA3 DQ    ?
;-----------------------------------
      .CODE
MAIN PROC   FAR
     MOV    AX,@DATA
     MOV    DS,AX
     CLC                        ;clear carry before first addition
     MOV    SI,OFFSET DATA1     ;SI is pointer for operand1
     MOV    DI,OFFSET DATA2     ;DI is pointer for operand2
     MOV    BX,OFFSET DATA3     ;BX is pointer for the sum
     MOV    CX,04               ;CX is the loop counter
BACK:MOV    AX,[SI]                  ;move the first operand to AX
     ADC    AX,[DI]                  ;add the second operand to AX
     MOV    [BX],AX                  ;store the sum
     INC    SI                       ;point to next word of operand1
     INC    SI
     INC    DI                  ;point to next word of operand2
     INC    DI
     INC    BX                  ;point to next word of sum
     INC    BX
     LOOP   BACK                ;if not finished, continue adding
     MOV    AH,4CH
     INT    21H                 ;go back to OS
MAIN ENDP
     END    MAIN
```

```
LOOP  xxxx  ;is equivalent to
```

```
DEC   CX
JNZ   xxxx
```

Subtraction of Unsigned Numbers

```
SUB  dest,source;dest = dest - source
```

- The x86 uses internal adder circuitry to perform the subtraction command.
- Hence, the 2's complement method is used by the microprocessor to perform the subtraction. The steps involved are –
  - Take the 2's complement of the subtrahend (source operand)
  - Add it to the minuend (destination operand)
  - Invert the carry.

Show the steps involved in the following:

```
MOV   AL,3FH      ;load AL=3FH
MOV   BH,23H      ;load BH=23H
SUB   AL,BH       ;subtract BH from AL. Place result in AL.
```

Solution:

```
 AL    3F          0011 1111        0011 1111
-BH   -23        - 0010 0011       + 1101 1101 (2's complement)
       1C                          1 0001 1100 CF=0 (step 3)
```

The flags would be set as follows: CF = 0, ZF = 0, AF = 0, PF = 0, and SF = 0.
The programmer must look at the carry flag (not the sign flag) to determine if the result is positive or negative.

- After the execution of SUB,
  - if CF = 0, the result is positive;
  - if CF = 1, the result is negative and the destination has the 2's complement of the result

» Normally, the result is left in 2's complement, but the NOT and INC instructions can be used to change it.

» The NOT instruction performs the 1's complement of the operand;

» The operand is incremented to get the 2's complement

Analyze the following program:

```
;from the data segment:
DATA1      DB     4CH
DATA2      DB     6EH
DATA3      DB     ?
;from the code segment:
           MOV    DH,DATA1      ;load DH with DATA1 value (4CH)
           SUB    DH,DATA2      ;subtract DATA2 (6E) from DH (4CH)
           JNC    NEXT          ;if CF=0 jump to NEXT target
           NOT    DH            ;if CF=1 then take 1's complement
           INC    DH            ;and increment to get 2's complement
NEXT:      MOV    DATA3,DH      ;save DH in DATA3
```

Solution:

Following the three steps for "SUB DH,DATA2":

```
 4C    0100 1100        0100 1100
-6E    0110 1110      + 1001 0010   (2's complement)
-22                     01101 1110 CF=1 (step 3)result is negative
```

SBB (subtract with borrow)

» If the carry flag is 0, SBB works like SUB. If the carry flag is 1, SBB subtracts 1 from the result

0625	62FA
- 0412	963B
0212	CCBF

Analyze the following program:

```
DATA_A      DD    62562FAH
DATA_B      DD    412963BH
RESULT      DD    ?
...         ...   ...
            MOV   AX,WORD PTR DATA_A       ;AX=62FA
            SUB   AX,WORD PTR DATA_B       ;SUB 963B from AX
            MOV   WORD PTR RESULT,AX       ;save the result
            MOV   AX,WORD PTR DATA_A +2    ;AX=0625
            SBB   AX,WORD PTR DATA_B +2    ;SUB 0412 with borrow
            MOV   WORD PTR RESULT+2,AX     ;save the result
```

Solution:

After the SUB, AX = 62FA – 963B = CCBF and the carry flag is set. Since CF = 1, when SBB is executed, AX = 625 – 412 – 1 = 212. Therefore, the value stored in RESULT is 0212CCBF.

» The PTR (pointer) data directive is used to specify the size of the operand when it differs from the defined size.

» In above Example; "*WORD PTR*" tells the assembler to use a word operand, even though the data is defined as a double word

REVIEW

1. Find CF, ZF, and AF for each of the following:

(a)
```
MOV BH, 3FH
ADD BH, 45H
ADD CX, DX
```

(b)
```
MOV DX, 4599H
MOV CX, 3458H
ADC AX, 00
```

(c)
```
MOV DX, 0FF01H
ADD BL, BH
ADC CX, 00
```

(d)
```
MOV CX, 0FFFFH
STC
ADD AH, 00
```

2. Write a program that calculates the total sum paid to a salesperson for eight months Monthly paychecks are as follows: $2300, $4300, $1200, $3700, $1298, $4323, $5673, $986.

3. Write a program that subtracts two multi-bytes (548FB9963CEH & 3FCD4FA23B8DH) and saves the result. Subtraction should be done a byte at a time.

UNSIGNED MULTIPLICATION & DIVISION

Multiplication of Unsigned Numbers

- In discussing multiplication, the following cases will be examined:
  - (1) byte times byte
  - (2) word times word
  - (3) byte times word

8-bit * 8-bit	AL * BL	16-bit * 16-bit	AX * BX
16-bit	AX	32-bit	DX AX

byte x byte

```
RESULT   DW    ?          ;result is defined in the data segment
         ...
         MOV   AL,25H     ;a byte is moved to AL
         MOV   BL,65H     ;immediate data must be in a register
         MUL   BL         ;AL = 25 x 65H
         MOV   RESULT,AX  ;the result is saved
```

```
;from the data segment:
DATA1      DB      25H
DATA2      DB      65H
RESULT     DW      ?
;from the code segment:
      MOV   AL,DATA1
      MOV   BL,DATA2
      MUL   BL                  ;register addressing mode
      MOV   RESULT,AX
```

or

```
      MOV   AL,DATA1
      MUL   DATA2               ;direct addressing mode
      MOV   RESULT,AX
```

or

```
      MOV   AL,DATA1
      MOV   SI,OFFSET DATA2
      MUL   BYTE PTR [SI]       ;register indirect addressing mode
      MOV   RESULT,AX
```

word x word

```
DATA3       DW    2378H
DATA4       DW    2F79H
RESULT1     DW    2 DUP(?)
...         ....
            MOV   AX,DATA3      ;load first operand into AX
            MUL   DATA4         ;multiply it by the second operand
            MOV   RESULT1,AX    ;store the lower word result
            MOV   RESULT1+2,DX  ;store the higher word result
```

word x byte

```
;from the data segment:
DATA5       DB    6BH
DATA6       DW    12C3H
RESULT3     DW    2 DUP(?)
;from the code segment:
            MOV   AL,DATA5      ;AL holds byte operand
            SUB   AH,AH         ;AH must be cleared
            MUL   DATA6         ;byte in AL mult. by word operand
            MOV   BX,OFFSET RESULT3  ;BX points to product
            MOV   [BX],AX            ;AX holds lower word
            MOV   [BX]+2,DX          ;DX holds higher word
```

Summary

Multiplication	Operand 1	Operand 2	Result
byte × byte	AL	register or memory	AX
word × word	AX	register or memory	DX AX
word × byte	AL = byte, AH = 0	register or memory	DX AX

Division of Unsigned Numbers

» In the division of unsigned numbers, the following cases are discussed: (a) Byte over byte (b) Word over word

(c) Word over byte (d) Double-word over word

8-bit	AL	Q: AL
8-bit	BL	R: AH

16-bit	AX	Q: AX
16-bit	BX	R: DX

16-bit	AX	Q: AL
8-bit	BL	R: AH

32-bit	DA AX	Q: AX
16-bit	BX	R: DX

» In divide – CPU cannot perform the division

» In these cases an ***interrupt*** (*exception*)

1. If the denominator is zero (dividing any number by 00)
2. If the quotient is too large for the assigned register.

byte/byte

```
QOUT1       DB    ?
REMAIN1     DB    ?

;using immediate addressing mode will give an error
            MOV   AL,DATA7          ;move data into AL
            SUB   AH,AH             ;clear AH
            DIV   10                ;immed. mode not allowed!!
;allowable modes include:
;using direct mode
            MOV   AL,DATA7          ;AL holds numerator
            SUB   AH,AH             ;AH must be cleared
            DIV   DATA8             ;divide AX by DATA8
            MOV   QOUT1,AL          ;quotient = AL = 09
            MOV   REMAIN1,AH        ;remainder = AH = 05
;using register addressing mode
            MOV   AL,DATA7          ;AL holds numerator
            SUB   AH,AH             ;AH must be cleared
            MOV   BH,DATA8          ;move denom. to register
            DIV   BH                ;divide AX by BH
            MOV   QOUT1,AL          ;quotient = AL = 09
            MOV   REMAIN1,AH        ;remainder = AH = 05
;using register indirect addressing mode
            MOV   AL,DATA7          ;AL holds numerator
            SUB   AH,AH             ;AH must be cleared
            MOV   BX,OFFSET DATA8   ;BX holds offset of DATA8
            DIV   BYTE PTR [BX]     ;divide AX by DATA8
            MOV   QOUT2,AX
            MOV   REMAIND2,DX
```

word/word

2742

0064

```
MOV  AX,10050      ;AX holds numerator
SUB  DX,DX         ;DX must be cleared
MOV  BX,100        ;BX used for denominator
DIV  BX
MOV  QOUT2,AX      ;quotient = AX = 64H = 100
MOV  REMAIND2,DX   ;remainder = DX = 32H = 50
```

word/byte

0807 Q (AL) = 14

64 R (AH) = 37

```
MOV  AX,2055       ;AX holds numerator
MOV  CL,100        ;CL used for denominator
DIV  CL
MOV  QUO,AL        ;AL holds quotient
MOV  REMI,AH       ;AH holds remainder
```

double-word/word

```
;from the data segment:
DATA1       DD    105432
DATA2       DW    10000
QUOT        DW    ?
REMAIN      DW    ?
;from the code segment:
            MOV   AX,WORD PTR DATA1        ;AX holds lower word
            MOV   DX,WORD PTR DATA1+2;DX higher word of numerator
            DIV   DATA2
            MOV   QUOT,AX                  ;AX holds quotient
            MOV   REMAIN,DX                ;DX holds remainder
```

0001 9BD8 Q (AX) = 000A

2710 R (DX) = 1538

Summary

Division	Numerator	Denominator	Quotient	Rem.
byte/byte	AL = byte, AH = 0	register or memory	AL[1]	AH
word/word	AX = word, DX = 0	register or memory	AX[2]	DX
word/byte	AX = word	register or memory	AL[1]	AH
doubleword/word	DXAX = doubleword	register or memory	AX[2]	DX

REVIEW

1. If byte1 = 230, byte2 = 100, word1 = 9998, word2 = 300, and double-word = 100000; write the result of a program that performs the following:

   (a) byte1 x byte2 (b) byte1 x word1

   (c) word1 x word2 (d) byte1 / byte 2

   (e) word1 / word2 (f) double-word / byte1

LOGIC INSTRUCTIONS

» The logic instructions AND, OR, XOR, SHIFT, and COMPARE are discussed with examples

AND

Show the results of the following:

```
MOV   BL,35H
AND   BL,0FH        ;AND BL with 0FH. Place the result in BL.
```

Solution:

```
35H   0 0 1 1 0 1 0 1
0FH   0 0 0 0 1 1 1 1
05H   0 0 0 0 0 1 0 1
```

Flag settings will be: SF = 0, ZF = 0, PF = 1, CF = OF = 0.

Inputs		Output
A	**B**	**A AND B**
0	0	0
0	1	0
1	0	0
1	1	1

» AND can be used to mask certain bits of the operand. The task of clearing a bit in a binary number is called **masking**.

```
  xxxx xxxx   Unknown number
• 0000 1111   Mask
  ----------
  0000 xxxx   Result
```

```
        AND   DH,DH
        JZ    XXXX
        ...
XXXX:   ...
```

OR

```
OR    destination,source
```

Inputs		Output
A	**B**	**A OR B**
0	0	0
0	1	1
1	0	1
1	1	1

Show the results of the following:

```
MOV  AX,0504      ;AX = 0504
OR   AX,0DA68H    ;AX = DF6C
```

Solution:

```
0504H  0000 0101 0000 0100
DA68H  1101 1010 0110 1000
DF6C   1101 1111 0110 1100
```

Flags will be: SF = 1 , ZF = 0, PF = 1, CF = OF = 0.

Notice that parity is checked for the lower 8 bits only.

» The OR instruction can be used to test for a zero operand. For example, "*OR BL, 0*"will OR the register BL with 0 and make ZF = 1, if BL is zero. "*OR BL, BL*" will achieve the same result.

» OR can also be used to set certain bits of an operand to 1

```
  x x x x  x x x x   Unknown number
+ 0 0 0 0  1 1 1 1   Mask
-----------------
  x x x x  1 1 1 1   Result
```

XOR

```
XOR  dest,src
```

Show the results of the following:

```
MOV    DH,54H
XOR    DH,78H
```

Solution:

```
54H   0 1 0 1 0 1 0 0
78H   0 1 1 1 1 0 0 0
2C    0 0 1 0 1 1 0 0
```

Flag settings will be: SF = 0, ZF = 0, PF = 0, CF = OF = 0.

Inputs		Output
A	**B**	**A XOR B**
0	0	1
0	1	0
1	0	0
1	1	1

The XOR instruction can be used to clear the contents of a register by XORing it with itself. Show how "XOR AH,AH" clears AH, assuming that AH = 45H.

Solution:

```
45H   01000101
45H   01000101
00    00000000
```

Flag settings will be: SF = 0, ZF = 1, PF =1 , CF = OF = 0.

- XOR can be used to see if two registers have the same value. "*XOR BX, CX*" will make ZF = 1, if both registers have the same value, and if they do, the result (0000) is saved in BX, the destination

- XOR can also be used to toggle (invert/compliment) bits of an operand. For example, to toggle bit 2 of register AL

```
  x x x x  x x x x   Unknown number
⊕ 0 0 0 0  1 1 1 1   Mask
  ---------------
  x x x x  x̄ x̄ x̄ x̄   Result
```

```
XOR  AL,04H        ;XOR  AL with 0000 0100
```

- This would cause bit 2 of AL to change to the opposite value; all other bits would remain unchanged.

SHIFT

- Shift instructions shift the contents of a register or memory location right or left.
- The number of times (or bits) that the operand is shifted can be specified directly if it is once only, or through the CL register if it is more than once.
- There are two kinds of shifts:
  - Logical – for unsigned operands
  - Arithmetic – signed operands.

SHR

Eg:

SHR BH, CL

Shift right

	Before	*After*
BH	0100 0100	0001 0001
CL	02H	
Cy	1	0

Show the result of SHR in the following:

```
MOV    AL, 9AH
MOV    CL, 3    ;set number of times to shift
SHR    AL, CL
```

Solution:

```
9AH =      10011010
           01001101     CF = 0 (shifted once)
           00100110     CF = 1 (shifted twice)
           00010011     CF = 0 (shifted three times)
```

After shifting right three times, AL = 13H and CF = 0.

- If the operand is to be shifted once only, this is specified in the SHR instruction itself rather than placing 1 in the CL. This saves coding of one instruction

```
MOV   BX,0FFFFH   ;BX=FFFFH
SHR   BX,1        ;shift right BX once only
```

- After the above shift, BX = 7FFFH and CF = 1

Show the results of SHR in the following:

```
;from the data segment:
DATA1       DW    7777H
;from the code segment:
TIMES       EQU   4
            MOV   CL,TIMES     ;CL=04
            SHR   DATA1,CL     ;shift DATA1 CL times
```

Solution:

After the four shifts, the word at memory location DATA1 will contain 0777. The four LSBs are lost through the carry, one by one, and 0s fill the four MSBs.

SHL

Eg:
SHL BH, CL

Shift left without Cy

	Before	*After*
BH	0010 0010	1000 1000
CL	02H	
Cy	1	0

Show the effects of SHL in the following:

```
MOV    DH,6
MOV    CL,4
SHL    DH,CL
```

Solution:

```
                 00000110
CF=0             00001100      (shifted left once)
CF=0             00011000
CF=0             00110000
CF=0             01100000      (shifted four times)
```

After the four shifts left, the DH register has 60H and CF = 0.

Can also be coded as

```
MOV    DH,6
SHL    DH,1
SHL    DH,1
SHL    DH,1
SHL    DH,1
```

COMPARE of Unsigned Numbers

```
CMP   destination,source   ;compare dest and src
```

Compare Operands	CF	ZF	Remark
destination > source	0	0	destination – source; results CF = 0 & ZF = 0
destination = source	0	1	destination – source; results CF = 0 & ZF = 1
destination < source	1	0	destination – source; results CF = 1 & ZF = 0

```
DATA1 DW    235FH
      ...
      MOV   AX,0CCCCH
      CMP   AX,DATA1     ;compare CCCC with 235F
      JNC   OVER         ;jump if CF=0
      SUB   AX,AX
OVER: INC   DATA1
```

```
        MOV   BX,7888H
        MOV   CX,9FFFH
        CMP   BX,CX        ;compare 7888 with 9FFF
        JNC   NEXT
        ADD   BX,4000H
NEXT:   ADD   CX,250H
```

```
TEMP  DB   ?
...
      MOV   AL,TEMP     ;move the TEMP variable into AL
      CMP   AL,99       ;compare AL with 99
      JZ    HOT_HOT     ;if ZF=1 (TEMP = 99) jump to HOT_HOT
      INC   BX          ;otherwise (ZF=0) increment BX
...
HOT_HOT: HLT            ;halt the system
```

Assume that there is a class of five people with the following grades: 69, 87, 96, 45, and 75. Find the highest grade.

```
TITLE       PROG3-3 (EXE) CMP EXAMPLE
PAGE        60,132
.MODEL SMALL
.STACK 64
;------------------
            .DATA
GRADES      DB    69,87,96,45,75
            ORG   0008
HIGHEST     DB    ?
;------------------
            .CODE
MAIN        PROC  FAR
            MOV   AX,@DATA
            MOV   DS,AX
            MOV   CX,5                ;set up loop counter
            MOV   BX,OFFSET GRADES    ;BX points to GRADE data
            SUB   AL,AL         ;AL holds highest grade found so far
AGAIN:      CMP   AL,[BX]             ;compare next grade to highest
            JA    NEXT                ;jump if AL still highest
            MOV   AL,[BX]             ;else AL holds new highest
NEXT:       INC   BX                  ;point to next grade
            LOOP  AGAIN               ;continue search
            MOV   HIGHEST,AL          ;store highest grade
            MOV   AH,4CH
            INT   21H                 ;go back to OS
MAIN        ENDP
            END   MAIN
```

```
Count = 5
Highest = 0

REPEAT
   IF (Next > Highest)
   THEN
         Highest = Next
   ENDIF
   Decrement Count
UNTIL Count = 0

Store Highest
```

NOTE:

» There is a relationship between the pattem of lowercase and uppercase letters, as shown below for *A* and *a*

A	*0100 0001*	*41H*
a	*0110 0001*	*61H*

» The only bit that changes is d5. To change from lowercase to uppercase , d5 must be masked

» Note that small and capital letters in ASCII have the following values

Letter	Hex	Binary	Letter	Hex	Binary
A	41	0100 0001	a	61	0110 0001
B	42	0100 0010	b	62	0110 0010
C	43	0100 0011	c	63	0110 0011
...	...	...	...	...	...
Y	59	0101 1001	y	79	0111 1001
Z	5A	0101 1010	z	7A	0111 1010

```
TITLE       PROG3-4 (EXE) LOWERCASE TO UPPERCASE CONVERSION
 PAGE       60,132
 .MODEL SMALL
 .STACK 64
 ;----------------
            .DATA
 DATA1      DB     'mY NAME is jOe'
            ORG    0020H
 DATA2      DB     14 DUP(?)
 ;----------------
      .CODE
 MAIN PROC  FAR
      MOV   AX,@DATA
      MOV   DS,AX
      MOV   SI,OFFSET DATA1    ;SI points to original data
      MOV   BX,OFFSET DATA2    ;BX points to uppercase data
      MOV   CX,14              ;CX is loop counter
 BACK:MOV   AL,[SI]            ;get next character
      CMP   AL,61H             ;if less than 'a'
      JB    OVER               ;then no need to convert
      CMP   AL,7AH             ;if greater than 'z'
      JA    OVER               ;then no need to convert
      AND   AL,11011111B       ;mask d5 to convert to uppercase
 OVER:MOV   [BX],AL            ;store uppercase character
      INC   SI                 ;increment pointer to original
      INC   BX                 ;increment pointer to uppercase data
      LOOP  BACK               ;continue looping if CX > 0
      MOV   AH,4CH
      INT   21H                ;go back to OS
 MAIN ENDP
      END   MAIN
```

REVIEW

1. If AX = F000H, BX = 3456H, and DX = E390H; perform following operations and indicate the result:

(a) AND DX, AX

(b) OR DH, BL

(c) XOR AL, 76H

(d) AND DX, DX

(e) XOR AX, AX

(f) OR BX, DX

(g) AND AH, 0FFH

(h) OR ax, 9999H

(i) XOR DX, 0EEEEH

(j) XOR BX, BX

(k) MOV CL, 04
SHL AL, CL

(l) SHR DX, 1

(m) MOV CL, 3
SHR DL, CL

(n) MOV CL, 5
SHL BX, CL

(o) MOV CL, 6
SHL DX, CL

2. Indicate the status of ZF and CF after CMP is executed for the following:

(a) MOV BX, 2500
CMP BX, 1400

(b) MOV AL, 0FFH
CMP AL, 6FH

(c) MOV DL, 34
CMP DL, 88

(d) SUB AX, AX
CMP AX, 0000
CMP CX, 0FFFFH

(e) XOR DX, DX
CMP DX, 0FFFFH

(f) SUB CX, CX
DEC CX

(g) MOV BX, 2378H
MOV DX, 4000h
CMP DX, BX

(h) MOV AL, 0AAH
AND AL, 55H
CMP AL, 00

3. Indicate whether or not the jump happens in each case:

(a)
```
MOV CL, 5
SUB AL, AL
SHL AL, CL
JNC TARGET
JC TARGET
SHR DL, CL
JNC TARGET
```

(b)
```
MOV BH, 65H
MOV AL, 48H
OR AL, BH
SHL AL, 1
AND CL, 0FH
```

(c)
```
MOV AH, 55H
SUB DL, DL
OR DL, AH
MOV AL, AH
```

BCD & ASCII CONVERSION

BCD

- » The binary representation of 0 to 9 is called BCD
- » (1) unpacked BCD:95 = 0000 1001 0000 0101
- » (2) packed BCD: 95 = 1001 0101

ASCII

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

ASCII to Unpacked BCD Conversion

```
ASC       DB        '9562481273'
          ORG       0010H
UNPACK    DB        10 DUP(?)
```

```
        MOV   CX,5
        MOV   BX,OFFSET ASC      ;BX points to ASCII data
        MOV   DI,OFFSET UNPACK   ;DI points to unpacked BCD data
AGAIN:  MOV   AX,[BX]            ;move next 2 ASCII numbers to AX
        AND   AX,0F0FH           ;remove ASCII 3s
        MOV   [DI],AX            ;store unpacked BCD
        ADD   DI,2               ;point to next unpacked BCD data
        ADD   BX,2               ;point to next ASCII data
        LOOP  AGAIN
```

Data is accessed in word-sized chunks

PTR directive is used to access the data

```
        MOV   CX,5                ;CX is loop counter
        MOV   BX,OFFSET ASC       ;BX points to ASCII data
        MOV   DI,OFFSET UNPACK    ;DI points to unpacked BCD data
AGAIN:  MOV   AX,WORD PTR [BX]    ;move next 2 ASCII numbers to AX
        AND   AX,0F0FH            ;remove ASCII 3s
        MOV   WORD PTR [DI],AX    ;store unpacked BCD
        ADD   DI,2                ;point to next unpacked BCD data
        ADD   BX,2                ;point to next ASCII data
        LOOP  AGAIN
```

```
        MOV   CX,10              ;load the counter
        SUB   BX,BX              ;clear BX
AGAIN:  MOV   AL,ASC[BX]         ;move to AL content of mem [BX+ASC]
        AND   AL,0FH             ;mask the upper nibble
        MOV   UNPACK[BX],AL      ;move to mem [BX+UNPACK] the AL
        INC   BX                 ;point to next byte
        LOOP  AGAIN              ;loop until it is finished
```

based addressing mode (BX+ASC) is used as pointer

ASCII to Packed BCD Conversion

- First converted to unpacked BCD (to get rid of the 3) and then combined to make packed BCD
  - For 9 and 5 the keyboard gives 39 and 35, respectively. The goal is to produce 95H or"1001 0101"

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 or 47H

```
                ORG     0010H
VAL_ASC         DB      '47'
VAL_BCD         DB      ?
;reminder:      DB will put 34 in 0010H location and 37 in 0011H
                MOV     AX,WORD PTR VAL_ASC     ;AH=37,AL=34
                AND     AX,0F0FH            ;mask 3 to get unpacked BCD
                XCHG    AH,AL               ;swap AH and AL.
                MOV     CL,4                ;CL=04 to shift 4 times
                SHL     AH,CL               ;shift left AH to get AH=40H
                OR      AL,AH               ;OR them to get packed BCD
                MOV     VAL_BCD,AL          ;save the result
```

Packed BCD to ASCII Conversion

» First converted to unpacked and then the unpacked BCD is tagged with 011 0000 (30H)

Packed BCD	Unpacked BCD	ASCII
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 & 0000 1001	011 0010 & 011 1001

```
VAL1_BCD    DB    29H
VAL3-ASC    DW    ?
            ...
            MOV   AL,VAL1_BCD
            MOV   AH,AL          ;copy AL to AH. now AH=29,AL=29H
            AND   AX,0F00FH      ;mask 9 from AH and 2 from AL
            MOV   CL,4           ;CL=04 for shift
            SHR   AH,CL          ;shift right AH to get unpacked BCD
            OR    AX,3030H       ;combine with 30 to get ASCII
            XCHG  AH,AL          ;swap for ASCII storage convention
            MOV   VAL3_ASC,AX    ;store the ASCII
```

BCD Addition & Conversion

- In BCD addition, after adding packed BCD numbers, the result is no longer BCD

```
MOV AL,17H
ADD AL,28H
```

Adding them gives 0011 1111B (3FH), which is not BCD!

- The result above should have been 17+ 28 = 45 (0100 0101)
- To correct this problem, the programmer must add 6 (0110) to the low digit: 3F + 06 = 45H
- Another Example:
  - 52H + 87H = D9H
  - 6 must be added to the upper digit: D9H + 60H = 139H

DAA (*decimal adjust for addition*)

» DAA will add 6 to the lower nibble or higher nibble if needed; otherwise, it will leave the result alone

```
DATA1   DB  47H
DATA2   DB  25H
DATA3   DB?
        MOV AL,DATA1      ;AL holds first BCD operand
        MOV BL,DATA2      ;BL holds second BCD operand
        ADD AL,BL         ;BCD addition
        DAA               ;adjust for BCD addition
        MOV DATA3,AL      ;store result in correct BCD form
```

» If after an ADD/ADC instruction the lower nibble (4 bits) is greater than 9, or if AF = 1, add 0110 to the lower 4 bits

» If the upper nibble is greater than 9, or if CF = 1, add 0110 to the upper nibble

Examples

```
Eg1:                    ; AL = 0011 1001 = 39 BCD
                        ; CL = 0001 0010 = 12 BCD
        ADD AL, CL      ; AL = 0100 1011 = 4BH
        DAA             ; Since 1011 > 9; Add correction factor 06.
                        ; AL = 0101 0001 = 51 BCD

Eg2:                    ; AL = 1001 0110 = 96 BCD
                        ; BL = 0000 0111 = 07 BCD
        ADD AL, BL      ; AL = 1001 1101 = 9DH
        DAA             ; Since 1101 > 9; Add correction factor 06
                        ; AL = 1010 0011 = A3H
                        ; Since 1010 > 9; Add correction factor 60
                        ; AL = 0000 0011 = 03 BCD. The result is 103.
```

1: Add decimal numbers 22 and 18.

```
MOV AL, 22H        ; (AL)= 22H
ADD AL, 18H        ; (AL) = 3AH Illegal, incorrect answer!
DAA                ; (AL) = 40H   Just treat it as decimal with Cy = 0

  3AH              In this case, DAA same as ADD AL, 06H
+06H               When LS hex digit in AL is >9, add 6 to it
=40H
```

2: Add decimal numbers 93 and 34.

```
MOV AL, 93H        ; (AL)= 93H
ADD AL, 34H        ; (AL) = C7H, Cy=0   Illegal & Incorrect!
DAA                ; (AL) = 27H   Just treat it as decimal with Cy = 1

  C7H              In this case, DAA same as ADD AL, 60H
+60H               When MS hex digit in AL is >9, add 6 to it
=27H
```

3: Add decimal numbers 93 and 84.

```
MOV AL, 93H        ; (AL)= 93H
ADD AL, 84H        ; (AL) = 17H, Cy = 1   Incorrect answer!

DAA                ; (AL) = 77H   Just treat it as decimal with Cy = 1 (carry generated?)

  17H              In this case, DAA same as ADD AL, 60H
+60H               When Cy = 1, add 6 to MS hex digit of AL and treat
=77H               Carry as 1 even though not generated in this addition
```

4: Add decimal numbers 65 and 57.

```
MOV AL, 65H        ; (AL)= 65H
ADD AL, 57H        ; (AL) = BCH
DAA                ; (AL) = 22H   Just treat it as decimal with Cy = 1

 BCH               In this case, DAA same as ADD AL, 66H
+66H
=22H  Cy = 1
```

5: Add decimal numbers 99 and 28.

```
MOV AL, 99H        ; (AL)= 99H
ADD AL, 28H        ; (AL) = C1H, Ac = 1
DAA                ; (AL) = 27H   Just treat it as decimal with Cy = 1

  C1H              In this case, DAA same as ADD AL, 66H
+66H               6 added to LS hex digit of AL, as Ac = 1
=27H  Cy = 1       6 added to MS hex digit of AL, as it is >9
```

6: Add decimal numbers 36 and 42.

```
MOV AL, 36H        ; (AL)= 36H
ADD AL, 42H        ; (AL) = 78H
DAA                ; (AL) = 78H   Just treat it as decimal with Cy = 0

  78H
+00H               In this case, DAA same as ADD AL, 00H
=78H
```

Hex		BCD	
29		0010 1001	
+ 18	+	0001 1000	
41		0100 0001	Because AF = 1,
+ 6	+	0110	DAA adds 6 to lower nibble.
47		0100 0111	The final result is BCD.

Hex		BCD	
53		0010 0011	
+ 75	+	0111 0101	
D8		1101 1000	Because the upper nibble is greater than 9,
+ 6	+	0110	DAA adds 6 to upper nibble.
128		0010 1000	The final result is BCD.

Two sets of ASCII data have come in from the keyboard. Write and run a program to:
1. Convert from ASCII to packed BCD.
2. Add the multibyte packed BCD and save it.
3. Convert the packed BCD result to ASCII.

```
TITLE       PROG3-6 (EXE) ASCII TO BCD CONVERSION AND ADDITION
PAGE        60,132
.MODE SMALL
.STACK 64
;---------------------
            .DATA
DATA1_ASC   DB      `0649147816'
            ORG     0010H
DATA2_ASC   DB      `0072687188'
            ORG     0020H
DATA3_BCD   DB      5 DUP (?)
            ORG     0028H
DATA4_BCD   DB      5 DUP (?)
            ORG     0030H
DATA5_ADD   DB      5 DUP (?)
            ORG     0040H
DATA6_ASC   DB      10 DUP (?)
;-----------------------
```

```
            .CODE
MAIN PROC   FAR
     MOV    AX,@DATA
     MOV    DS,AX
     MOV    BX,OFFSET DATA1_ASC      ;BX points to first ASCII data
     MOV    DI,OFFSET DATA3_BCD      ;DI points to first BCD data
     MOV    CX,10                    ;CX holds number bytes to convert
     CALL   CONV_BCD                 ;convert ASCII to BCD
     MOV    BX,OFFSET DATA2_ASC      ;BX points to second ASCII data
     MOV    DI,OFFSET DATA4_BCD      ;DI points to second BCD data
     MOV    CX,10                    ;CX holds number bytes to convert
     CALL   CONV_BCD                 ;convert ASCII to BCD
     CALL   BCD_ADD                  ;add the BCD operands
     MOV    SI,OFFSET DATA5_ADD      ;SI points to BCD result
     MOV    DI,OFFSET DATA6_ASC      ;DI points to ASCII result
     MOV    CX,05                    ;CX holds count for convert
     CALL   CONV_ASC                 ;convert result to ASCII
     MOV    AH,4CH
     INT    21H                      ;go back to OS
MAIN ENDP
;------------------------
;THIS SUBROUTINE CONVERTS ASCII TO PACKED BCD
CONV_BCD PROC
AGAIN:      MOV   AX,[BX]      ;BX=pointer for ASCII data
     XCHG   AH,AL
     AND    AX,0F0FH     ;mask ASCII 3s
     PUSH   CX           ;save the counter
     MOV    CL,4         ;shift AH left 4 bits
     SHL    AH,CL        ;to get ready for packing
     OR     AL,AH        ;combine to make packed BCD
     MOV    [DI],AL            ;DI=pointer for BCD data
     ADD    BX,2         ;point to next 2 ASCII bytes
     INC    DI           ;point to next BCD data
     POP    CX           ;restore loop counter
     LOOP   AGAIN
     RET
CONV_BCD ENDP
;---------------
```

```
;THIS SUBROUTINE ADDS TWO MULTIBYTE PACKED BCD OPERANDS
BCD_ADD PROC
      MOV  BX,OFFSET DATA3_BCD     ;BX=pointer for operand 1
      MOV  DI,OFFSET DATA4_BCD     ;DI=pointer for operand 2
      MOV  SI,OFFSET DATA5_ADD     ;SI=pointer for sum
      MOV  CX,05
      CLC
BACK: MOV  AL,[BX]+4    ;get next byte of operand 1
      ADC  AL,[DI]+4    ;add next byte of operand 2
      DAA               ;correct for BCD addition
      MOV  [SI]+4,AL    ;save sum
      DEC  BX           ;point to next byte of operand 1
      DEC  DI           ;point to next byte of operand 2
      DEC  SI           ;point to next byte of sum
      LOOP       BACK
      RET
BCD_ADD ENDP
;----------------
;THIS SUBROUTINE CONVERTS FROM PACKED BCD TO ASCII
CONV_ASC PROC
AGAIN2: MOV AL,[SI]     ;SI=pointer for BCD data
     MOV   AH,AL        ;duplicate to unpack
     AND   AX,0F00FH    ;unpack
     PUSH  CX           ;save counter
     MOV   CL,04        ;shift right 4 bits to unpack
     SHR   AH,CL        ;the upper nibble
     OR    AX,3030H     ;make it ASCII
     XCHG  AH,AL        ;swap for ASCII storage convention
     MOV   [DI],AX            ;store ASCII data
     INC   SI           ;point to next BCD data
     ADD   DI,2         ;point to next ASCII data
     POP   CX           ;restore loop counter
     LOOP  AGAIN2
     RET
CONV_ASC ENDP
     END   MAIN
```

BCD Subtraction & Conversion

» The problem associated with the addition of packed BCD numbers also shows up in subtraction

DAS

» If after a SUB or SBB instruction the lower nibble is greater than 9, or if AF = 1, subtract 0110 from the lower 4 bits.

» If the upper nibble is greater than 9, or CF = 1, subtract 0110 from the upper nibble.

```
BUDGET      DT    87965141012
EXPENSES    DT    31610640392
BALANCE     DT    ?            ;balance = budget - expenses

      MOV   CX,10              ;counter=10
      MOV   BX,00              ;pointer=0
      CLC                      ;clear carry for the 1st iteration
      BACK: MOVAL,BYTE PTR BUDGET[BX]  ;get a byte of the BUDGET
      SBB   AL,BYTE PTR EXPENSES[BX]   ;subtract a byte from it
      DAS                           ;correct the result for BCD
      MOV   BYTE PTR BALANCE[BX],AL ;save it in BALANCE
      INC   BX                      ;increment for the next byte
      LOOP  BACK                    ;continue until CX=0
```

```
Eg1:                    ; AL = 0011 0010 = 32 BCD
                        ; CL = 0001 0111 = 17 BCD
        SUB AL, CL      ; AL = 0001 1011 = 1BH
        DAS             ; Subtract 06, since 1011 > 9.
                        ; AL = 0001 0101 = 15 BCD

Eg2:                    ; AL = 0010 0011 = 23 BCD
                        ; CL = 0101 1000 =58 BCD
        SUB AL, CL      ; AL = 1100 1011 = CBH
        DAS             ; Subtract 66, since 1100 >9 & 1011 > 9.
                        ; AL = 0110 0101 = 65 BCD, CF = 1.
                        ; Since CF = 1, answer is – 65.
```

1: Subtract decimal numbers 45 and 38.

```
MOV AL, 45H        ; (AL)= 45H
SUB AL, 38H        ; (AL) = 0DH   Illegal, incorrect answer!
 DAS               ; (AL) = 07H   Just treat it as decimal  with Cy = 0

  0DH              In this case, DAS same as SUB AL, 06H
-06H               When LS hex digit in AL is >9, subtract 6
=07H
```

2: Subtract decimal numbers 63 and 88.

```
MOV AL, 63H      ; (AL)= 63H
SUB AL, 88H      ; (AL) = DBH, Cy=1   Illegal & Incorrect!
DAS              ; (AL) = 75H   Just treat it as decimal with Cy = 1 (carry generated?)

 DBH             In this case, DAS same as SUB AL, 66H
-66H             When Cy = 1, it means result is negative
=75H             Result is 75, which is 10's complement of 25
                 Treat Cy as 1 as Cy was generated in the previous subtraction itself!
```

3: Subtract decimal numbers 45 and 52.

```
MOV AL, 45H      ; (AL)= 45H
SUB AL, 52H      ; (AL) = F3H, Cy = 1   Incorrect answer!
DAS              ; (AL) = 93H   Just treat it as decimal with Cy = 1 (carry generated?)

  F3H            In this case, DAS same as SUB AL, 60H
-60H             When Cy = 1, it means result is negative
=93H             Result is 93, which is 10's complement of 07
```

4: Subtract decimal numbers 50 and 19.

```
MOV AL, 50H        ; (AL)= 50H
SUB AL, 19H        ; (AL) = 37H, Ac = 1
DAS                ; (AL) = 31H   Just treat it as decimal with Cy =0

  37H              In this case, DAS same as SUB AL, 06H
-06H               06H is subtracted from AL as Ac = 1
=31H
```

5: Subtract decimal numbers 99 and 88.

```
MOV AL, 99H        ; (AL)= 99H
SUB AL, 88H        ; (AL) = 11H
DAS                ; (AL) = 11H   Just treat it as decimal with Cy = 0

  11H              In this case, DAS same as SUB AL, 00H
-00H
=11H
```

6: Subtract decimal numbers 14 and 92.

```
MOV AL, 14H        ; (AL)= 14H
SUB AL, 92H        ; (AL) = 82H, Cy = 1
DAS                ; (AL) = 22H   Just treat it as decimal with Cy = 1

  82H              In this case, DAS same as SUB AL, 60H
-60H               60H is subtracted from AL as Cy = 1
=22H               22 is 10's complement of 78
```

REVIEW

1. Find the value of AL after the following code is executed:

```
MOV AL, 29H
ADD AL, 18H
DAA
```

2. Using DT directive, write a program to subtract two 10-byte BCD numbers
3. Using DT directive, write a program to add two 10-byte BCD numbers
4. Write a program to count up (00 to 99)
5. Write a program to count down (99 to 00)

6. An instructor has the following grading policy: “Curving of grades is achieved by adding to every grade the difference between 99 and the highest grade in the class”. If the following are the grades of the class, write a program to calculate the grades after they have been curved: 81, 65, 77, 82, 73, 55, 88, 78, 51, 91, 86, 76.

ROTATE INSTRUCTIONS

- The rotation instructions ROR, ROL and RCR, RCL allow a program to rotate an operand right or left
- Operand can be in a register or memory
- If the number of times an operand is to be rotated is more than 1, this is indicated by CL
- There are two types of rotations
  - Rotation of the bits of the operand
  - Rotation through the carry

ROR (rotate right)

Eg:

ROR BH, 1

Rotate right without Cy

	Before	*After*
BH	0100 0010	0010 0001
Cy	1	0

```
        MOV     AL,36H          ;AL=0011 0110
        ROR     AL,1            ;AL=0001 1011   CF=0
        ROR     AL,1            ;AL=1000 1101   CF=1
        ROR     AL,1            ;AL=1100 0110   CF=1
;or:
        MOV     AL,36H          ;AL=0011 0110
        MOV     CL,3            ;CL=3 number of times to rotate
        ROR     AL,CL           ;AL=1100 0110 CF=1
;the operand can be a word:
        MOV     BX,0C7E5H       ;BX=1100 0111 1110 0101
        MOV     CL,6            ;CL=6 number of times to rotate
        ROR     BX,CL           ;BX=1001 0111 0001 1111 CF=1
```

ROL (rotate left)

Eg:

ROL BH, CL

Rotate left without Cy

	Before	*After*
BH	0010 0010	1000 1000
CL	02H	
Cy	1	0

```
        MOV     BH,72H          ;BH=0111 0010
        ROL     BH,1            ;BH=1110 0100   CF=0
        ROL     BH,1            ;BH=1100 1001   CF=1
        ROL     BH,1            ;BH=1001 0011   CF=1
        ROL     BH,1            ;BH=0010 0111   CF=1
;or:
        MOV     BH,72H          ;BH=0111 0010
        MOV     CL,4            ;CL=4 number of times to rotate
        ROL     BH,CL           ;BH=0010 0111   CF=1

;The operand can be a word:
        MOV     DX,672AH        ;DX=0110 0111 0010 1010
        MOV     CL,3            ;CL=3 number of times to rotate
        ROL     DX,CL ;DX=0011 1001 0101 0011 CF=1
```

Write a program that finds the number of 1s in a byte.

```
;From the data segment:
DATA1      DB          97H
COUNT      DB          ?
;From the code segment:
           SUB   BL,BL       ;clear BL to keep the number of 1s
           MOV   DL,8        ;rotate total of 8 times
           MOV   AL,DATA1
AGAIN:     ROL   AL,1        ;rotate it once
           JNC   NEXT        ;check for 1
           INC   BL          ;if CF=1 then add one to count
NEXT:      DEC   DL          ;go through this 8 times
           JNZ   AGAIN       ;if not finished go back
           MOV   COUNT,BL    ;save the number of 1s
```

Write a program to count the number of 1s in a word. Provide the count in BCD.

```
DATAW1     DW          97F4H
COUNT2     DB          ?
           ...
           SUB   AL,AL       ;clear AL to keep the number of 1s in BCD
           MOV   DL,16       ;rotate total of 16 times
           MOV   BX,DATAW1   ;move the operand to BX
AGAIN:     ROL   BX,1        ;rotate it once
           JNC   NEXT        ;check for 1. If CF=0 then jump
           ADD   AL,1        ;if CF=1 then add one to count
           DAA               ;adjust the count for BCD
NEXT:      DEC   DL          ;go through this 16 times
           JNZ   AGAIN       ;if not finished go back
           MOV   COUNT2,AL   ;save the number of 1s in COUNT2
```

RCR (rotate right through carry)

Eg:

RCR BH, 1

Rotate right with Cy

	Before	*After*
BH	0100 0010	1010 0001
Cy	1	0

```
        CLC                     ;make CF=0
        MOV     AL,26H          ;AL=0010 0110
        RCR     AL,1            ;AL=0001 0011 CF=0
        RCR     AL,1            ;AL=0000 1001 CF=1
        RCR     AL,1            ;AL=1000 0100 CF=1
or:
        CLC                     ;make CF=0
        MOV     AL,26H          ;AL=0010 0110
        MOV     CL,3            ;CL=3 number of times to rotate
        RCR     AL,CL           ;AL=1000 0100 CF=1

;the operand can be a word
        STC                     ;make CF=1
        MOV     BX,37F1H        ;BX=0011 0111 1111 0001
        MOV     CL,5            ;CL=5 number of times to rotate
        RCR     BX,CL           ;BX=0001 1001 1011 1111 CF=0
```

RCL (rotate left through carry)

Eg:

RCL BH, CL

Rotate left with Cy

	Before	*After*
BH	0010 0010	1000 1010
CL	02H	
Cy	1	0

```
        STC                     ;make CF=1
        MOV     BL,15H          ;BL=0001 0101
        RCL     BL,1            ;0010 1011 CF=0
        RCL     BL,1            ;0101 0110 CF=0
or:
        STC                     ;make CF=1
        MOV     BL,15H          ;BL=0001 0101
        MOV     CL,2            ;CL=2 number of times for rotation
        RCL     BL,CL           ;BL=0101 0110 CF=0

;the operand can be a word:
        CLC                     ;make CF=0
        MOV     AX,191CH        ;AX=0001 1001 0001 1100
        MOV     CL,5            ;CL=5 number of times to rotate
        RCL     AX,CL           ;AX=0010 0011 1000 0001 CF=1
```

REVIEW

1. What is the value of BL after the following?

```
MOV   BL, 25H
MOV   CL, 4
ROR   BL, CL
```

2. What are the values of DX and CF after the following?

```
MOV   DX, 3FA2H
MOV   CL , 7
ROL   DX, CL
```

3. What is the value of BH after the following?

```
SUB   BH, BH
STC
RCR   BH, 1
STC
RCR   BH, 1
```

4. What is the value of BX after the following?

```
MOV   BX,  FFFFH
MOV   CL, 5
CLC
RCL    BX,CL
```

5. Why does "ROR BX , 4" give an error in the 8086? How would you change the code to make it work?

6. Write a program to find the number of zeros in a 16-bit word

7. Explain the difference between RCL and ROL instructions

15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 2 – QUIZ 1

ARITHMETIC & LOGIC INSTRUCTIONS AND PROGRAMS

Manjunath Prasanna K.
Dept. of CSE, VCET.

1. Instruction ADD DATA1, DATA2 is illegal (TRU/FALSE)
2. In unsigned multiplication of a byte in DATAl with a byte in AL, the product will be placed in register(s) _____
3. In unsigned multiplication of AX with BX, the product is placed in register(s) _____
4. In unsigned multiplication of CX with a byte in AL, the product is placed in register(s) _____
5. In unsigned division of a byte in AL by a byte in DH, the quotient will be placed in _____ and the remainder in _____

6. In unsigned division of a word in AX by a word in DATAl , the quotient will be placed in _____ and the remainder in _____
7. In unsigned division of a word in AX by a byte in DATA2, the quotient will be placed in _____ and the remainder in _____
8. In unsigned division of a double-word in DXAX by a word in CX, the quotient will be placed in _____ and the remainder in _____

9. To set all bits of an operand to 0, it should be ANDed with_____

10. To set all bits of an operand to 1 , it could be ORed with _____

11. XORing an operand with itself results in _____

12. The CMP instruction works by performing a(n) _____ operation on the operands and setting the flags accordingly

13. The CMP instruction does not alters the contents of its operands (TRUE/FALSE)

14. The DAA instruction must be used after the ADD instruction (TRUE/FALSE)
15. The DAS instruction must be used after the SUB instruction (TRUE/FALSE)

1. Instruction ADD DATA1, DATA2 is illegal (TRUE/FALSE)
2. In unsigned multiplication of a byte in DATAl with a byte in AL, the product will be placed in register(s) _____ (AX)
3. In unsigned multiplication of AX with BX, the product is placed in register(s) _____ (DX and AX)
4. In unsigned multiplication of CL with a byte in AL, the product is placed in register(s) _____ (AX)
5. In unsigned division of a byte in AL by a byte in DH, the quotient will be placed in _____ and the remainder in _____ (AL, AH)

6. In unsigned division of a word in AX by a word in DATAl , the quotient will be placed in _____ and the remainder in _____ (AX, DX)

7. In unsigned division of a word in AX by a byte in DATA2, the quotient will be placed in _____ and the remainder in _____ (AL, AH)

8. In unsigned division of a double-word in DXAX by a word in CX, the quotient will be placed in _____ and the remainder in _____ (AX, DX)

9. To set all bits of an operand to 0, it should be ANDed with_____ (all zeros)

10. To set all bits of an operand to 1 , it could be ORed with _____ (all ones)

11. XORing an operand with itself results in _____ (all zeros)

12. The CMP instruction works by performing a(n) _____ operation on the operands and setting the flags accordingly (SUB)

13. The CMP instruction does not alters the contents of its operands (TRUE/FALSE)

14. The DAA instruction must be used after the ADD instruction (TRUE/FALSE)
15. The DAS instruction must be used after the SUB instruction (TRUE/FALSE)

15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 2

INTERRUPTS IN x86 PC

Mahesh Prasanna K.

Dept. of CSE, VCET.

8088/86 INTERRUPTS

- An interrupt is an external event that informs the CPU that a device needs its service
  - In 8088/86, there are 256 interrupts: INT 00, INT 01, . . . , INT FF
- When an interrupt is executed, the microprocessor
  - automatically saves the flag register (FR)
  - the instruction pointer (IP)
  - the code segment register (CS) on the stack;
- and goes to a fixed memory location
- The memory locations to which an interrupt goes is four times the value of the interrupt number. For example, INT 03 will go to address 0000CH (4 * 3 = 12 = 0CH).

INT Number	Physical Address	Logical Address
INT 00	00000	0000 – 0000
INT 01	00004	0000 – 0004
INT 02	00008	0000 – 0008
INT 03	0000C	0000 – 000C
INT 04	00010	0000 – 0010
INT 05	00014	0000 – 0014
. . .	. . .	. . .
INT FF	003FC	0000 – 03FC

» The lowest 1024 bytes (256 x 4 = 1024) of memory space are set aside for the interrupt vector table

Interrupt Service Routine (ISR)

- For every interrupt there must be a program associated with it. This program is commonly referred to as an *interrupt service routine* (*ISR*) or *interrupt handler*
- For every interrupt there are allocated four bytes of memory in the interrupt vector table.
- These four memory locations provide the addresses of the interrupt service routine for which the interrupt was invoked.
- When an interrupt is invoked, it is asked to run a program to perform a certain service.

Find the physical and logical addresses in the interrupt vector table associated with:
(a) INT 12H (b) INT 8

Solution:

(a) The physical addresses for INT 12H are 00048H–0004BH since (4 × 12H = 48H). That means that the physical memory locations 48H, 49H, 4AH, and 4BH are set aside for the CS and IP of the ISR belonging to INT 12H. The logical address is 0000:0048H–0000:004BH.

(b) For INT 8, we have 8 × 4 = 32 = 20H; therefore, memory addresses 00020H, 00021H, 00022H, and 00023H in the interrupt vector table hold the CS:IP of the INT 8 ISR. The logical address is 0000:0020H–0000:0023H.

Difference between INT and CALL Instructions

CALL Instruction	INT instruction
1. A CALL FAR instruction can jump to any location within the 1M byte address range of the 8088/86 CPU.	1. INT nn goes to a fixed memory location in the interrupt vector table to get the address of the interrupt service routine.
2. A CALL FAR instruction is used by the programmer in the sequence of instructions in the program.	2. An externally activated hardware interrupt can come-in at any time, requesting the attention of the CPU.
3. A CALL FAR instruction cannot be masked (disabled).	3. INT nn belonging to externally activated hardware interrupts can be masked.
4. A CALL FAR instruction automatically saves only CS: IP of the next instruction on the stack.	4. INT nn saves FR (flag register) in addition to CS: IP of the next instruction.
5. At the end of the subroutine that has been called by the CALL FAR instruction, the RETF (return FAR) is the last instruction. RETF pops CS and IP off the stack.	5. The last instruction in the interrupt service routine (ISR) for INT nn is the instruction IRET (interrupt return). IRET pops off the FR (flag register) in addition to CS and IP.

Processing Interrupts

Categories of Interrupts – Hardware Interrupts

- 3 pins in the x86 – associated with hardware interrupts:
  - INTR – maskable – CLI/STI
  - NMI – non-maskable – INT 02
  - INTA

Categories of Interrupts – Software Interrupts

- Execution of *INT nn* – invoked from software, not from hardware
  - INT 10H – video interrupts
  - INT 21H – DOS functions
  - Predefined Functions –
    - "*INT 00*" (divide error)
    - "*INT 01*" (single step)
    - "*INT 03*" (breakpoint)
    - "*INT 04*" (signed number overflow)
- "*INT 05*" to "*INT FF*" can be used to implement either software or hardware interrupts

INT 00 (divide error)

- Conditional or exception interrupt – attempt to divide a number by zero – responsible for displaying the message "DIVIDE ERROR" on the screen

```
MOV   AL,92        ;AL=92
SUB   CL,CL        ;CL=0
DIV   CL           ;92/0=undefined result
```

- Also invoked if the quotient is too large to fit

```
MOV   AX,0FFFFH    ;AX=FFFFH
MOV   BL,2         ;BL=2
DIV   BL           ;65535/2 = 32767 larger than 255
                   ;maximum capacity of AL
```

INT 01 (single step)

- To examine the contents of the CPU's registers and system memory – executing the program one instruction at a time and then inspecting registers and memory – single step
- The trap flag (TF) (D8 of the flag register), must be set to 1

```
PUSHF
POP     AX
AND     AX,1111111011111111B
PUSH    AX
POPF
```

- To make TF = 1, one simply uses the OR instruction in place of the AND instruction

INT 02 (non-maskable interrupt)

- An active-high input pin in x86 microprocessor, NMI
- NMI input is often used for major system faults, such as power failures
- Microprocessor stores all of the internal registers in a battery-backed-up memory or an EEPROM

INT 03 (breakpoint)

- To allow implementation of breakpoints
- A breakpoint is used to examine the CPU and memory after the execution of a group of instructions
- A 1-byte instruction

INT 04 (signed number overflow)

- Invoked by signed number overflow condition
- if OF = 1 – INTO – INT 04
- If OF = 0 – NOP

```
MOV   AL,DATA1
MOV   BL,DATA2
ADD   AL,BL;add BL to AL
INTO
```

	+ 64	0100 0000	
+	+ 64	0100 0000	
	+128	1000 0000	OF=1 and the result is not +128

x86 PC & INTERRUPT ASSIGNMENT

- Of the 256 possible interrupts in the x86 –
  - some are used by the PC peripheral hardware (BIOS)
  - some are used by the Microsoft operating system
  - the rest are available for programmers of software applications

For a given ISR, the logical address is F000:FF53. Verify that the physical address is FFF53H.

Solution:

Since the logical address is F000:FF53, this means that CS = F000H and IP = FF53H. Shifting left the segment register one hex digit and adding it to the offset gives the physical address FFF53H.

REVIEW

1. Give the logical and physical addresses in the interrupt vector table associated with each of the following interrupts:

   (a) INT 5 (b) INT 21H

2. Calculate and compare the number of bytes of stack memory used by each of the following:

   (b) CALL FAR (b) Interrupt activation

3. What is the difference between RETF and IRET in terms of stack activity?

4. Explain how to set TF & IF to high.

15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 2

INT 21H & INT 10H PROGRAMMING

Mahesh Prasanna K.

Dept. of CSE, VCET.

BIOS INT 10H PROGRAMMING

- Subroutines, that are burned into the ROM BIOS of the x86-based IBM PC
- Used to communicate with the computer's screen video
- The manipulation of screen text or graphics can be done
  - Changing the color of characters or the background color,
  - Clearing the screen
  - Changing the location of the cursor
- Chosen by putting a specific value in register AH

Monitor Screen in Text Mode

» 80 columns and 25 rows in normal text mode

INT 10H Function 06H: Clearing the Screen (*scroll window up*)

» To clear the screen before displaying data; the following registers must contain certain values before INT 10H is called

```
MOV   AH,06      ;AH=06 to select scroll function
MOV   AL,00      ;AL-00 the entire page
MOV   BH,07      ;BH=07 for normal attribute
MOV   CH,00      ;CH=00 row value of start point
MOV   CL,00      ;CL=00 column value of start point
MOV   DH,24      ;DH=24 row value of ending point
MOV   DL,79      ;DL=79 column value of ending point
INT   10H        ;invoke the interrupt
```

```
MOV   AX,0600H   ;scroll entire screen
MOV   BH,07      ;normal attribute
MOV   CX,0000    ;start at 00,00
MOV   DX,184FH   ;end at 24,79 (hex = 18,4F)
INT   10H        ;invoke the interrupt
```

INT 10H Function 02H: Setting the Cursor to a Specific Location

Write the code to set the cursor position to row = 15 = 0FH and column = 25 = 19H.

Solution:

```
MOV   AH,02        ;set cursor option
MOV   BH,00        ;page 0
MOV   DL,25        ;column position
MOV   DH,15        ;row position
INT   10H          ;invoke interrupt 10H
```

Write a program that (1) clears the screen and (2) sets the cursor at the center of the screen.

Solution:

The center of the screen is the point at which the middle row and middle column meet. Row 12 is at the middle of rows 0 to 24 and column 39 (or 40) is at the middle of columns 0 to 79. By setting row = DH = 12 and column = DL = 39, the cursor is set to the screen center.

```
;clearing the screen
      MOV   AX,0600H     ;scroll the entire page
      MOV   BH,07        ;normal attribute
      MOV   CX,0000      ;row and column of top left
      MOV   DX,184FH     ;row and column of bottom right
      INT   10H          ;invoke the video BIOS service

;setting the cursor to the center of screen
      MOV   AH,02        ;set cursor option
      MOV   BH,00        ;page 0
      MOV   DL,39        ;center column position
      MOV   DH,12        ;center row position
      INT   10H          ;invoke interrupt 10H
```

INT 10H Function 03H: Get Current Cursor Position

```
MOV   AH,03      ;option 03 of BIOS INT 10H
MOV   BH,00      ;page 00
INT   10H        ;interrupt 10H routine
```

- After execution of the program above,
  - Registers DH will have the current row position
  - Resister DL will have the current column positions
  - Register CX provides information about the shape of the cursor

Attribute Byte in Monochrome Monitors

» Attribute provides information to the video circuitry, such as color and intensity of the character (foreground) and the background

Binary	Hex	Result
0000 0000	00	white on white (no display)
0000 0111	07	white on black normal
0000 1111	0F	white on black highlight
1000 0111	87	white on black blinking
0111 0111	77	black on black (no display)
0111 0000	70	black on white
1111 0000	F0	black on white blinking

Write a program using INT 10H to:

(a) Change the video mode.

(b) Display the letter "D" in 200H locations with attributes black on white blinking (blinking letters "D" are black and the screen background is white).

(c) Then use DEBUG to run and verify the program.

Solution:

(a) INT 10H function AH = 00 is used with AL = video mode to change the video mode. Use AL = 03.

```
MOV   AH,00       ;SET MODE OPTION
MOV   AL,03       ;CHANGE THE VIDEO MODE
INT   10H         ;MODE OF 80X25 FOR ANY COLOR MONITOR
```

(b) With INT 10H function AH = 09, one can display a character a certain number of times with specific attributes.

```
MOV   AH,09       ;DISPLAY OPTION
MOV   BH,00       ;PAGE 0
MOV   AL,44H      ;THE ASCII FOR LETTER "D"
MOV   CX,200H     ;REPEAT IT 200H TIMES
MOV   BL,0F0H     ;BLACK ON WHITE BLINKING
INT   10H
```

(c) Reminder: DEBUG assumes that all the numbers are in hex.

```
C>debug
-A
1131:0100 MOV AH,00
1131:0102 MOV AL,03    ;CHANGE THE VIDEO MODE
1131:0104 INT 10
1131:0106 MOV AH,09
1131:0108 MOV BH,00
1131:010A MOV AL,44
1131:010C MOV CX,200
1131:010F MOV BL,F0
1131:0111 INT 10
1131:0113 INT 3
1131:0114
-
```

Now see the result by typing in the command -G Make sure that IP = 100 before running it. As an exercise, change the BL register to other attribute values given earlier. For example, BL = 07 white on black, or BL = 87H white on black blinking.

Attribute Byte in CGA Text Mode

D7	D6	D5	D4	D3	D2	D1	D0
B	R	G	B	I	R	G	B
	background				foreground		

B = blinking
I = foreground intensity
Blinking and intensity apply to foreground only.

Binary	Hex	Color effect
0000 0000	00	Black on black
0000 0001	01	Blue on black
0001 0010	12	Green on blue
0001 0100	14	Red on blue
0001 1111	1F	High-intensity white on blue

Write a program that puts 20H (ASCII space) on the entire screen. Use high-intensity white on a blue background attribute for any characters to be displayed.

Solution:

```
MOV   AH,00        ;SET MODE OPTION
MOV   AL,03        ;CGA COLOR TEXT MODE OF 80 × 25
INT   10H
MOV   AH,09        ;DISPLAY OPTION
MOV   BH,00        ;PAGE 0
MOV   AL,20H       ;ASCII FOR SPACE
MOV   CX,800H      ;REPEAT IT 800H TIMES
MOV   BL,1FH       ;HIGH-INTENSITY WHITE ON BLUE
INT   10H
```

Graphics: Pixel Resolution & Color

- In the text mode, the screen - a matrix of rows and columns of characters.
- In graphics mode, the screen - a matrix of horizontal and vertical pixels.
- Number of pixels depends on
  - monitor resolution
  - Type of the video board used
- There are two facts associated with every pixel on the screen:
  - The location of the pixel
  - Its attributes, color, and intensity
- These two facts must be stored in the video RAM.

- More pixels and colors – large memory is needed to store
- The CGA can have a maximum of 16K bytes of video memory.
- 16K bytes of memory can be used in three different ways:
  - Text mode of 80 x 25 characters: Use AL = 03 for mode selection in INT 10H option AH = 00 – 16 colors are supported.
  - Graphics mode of resolution 320 x 200 (medium resolution): Use AL = 04 – 4 colors are supported.
  - Graphics mode of resolution 640 x 200 (high resolution): Use AL = 06 – only 1 color (black and white) is supported.
- Hence, with a fixed amount of video RAM, the number of supported colors decreases as the resolution increases.

I	R	G	B	Color		I	R	G	B	Color
0	0	0	0	Black		1	0	0	0	Gray
0	0	0	1	Blue		1	0	0	1	Light Blue
0	0	1	0	Green		1	0	1	0	Light Green
0	0	1	1	Cyan		1	0	1	1	Light Cyan
0	1	0	0	Red		1	1	0	0	Light Red
0	1	0	1	Magenta		1	1	0	1	Light Magenta
0	1	1	0	Brown		1	1	1	0	Yellow
0	1	1	1	White		1	1	1	1	High Intensity White

INT 10H & Pixel Programming

Write a program to: (a) clear the screen, (b) set the mode to CGA of 640 × 200 resolution, and (c) draw a horizontal line starting at column = 100, row = 50, and ending at column 200, row 50.

Solution:

```
      MOV   AX,0600H    ;SCROLL THE SCREEN
      MOV   BH,07       ;NORMAL ATTRIBUTE
      MOV   CX,0000     ;FROM ROW=00,COLUMN=00
      MOV   DX,184FH    ;TO ROW=18H,COLUMN=4FH
      INT   10H         ;INVOKE INTERRUPT TO CLEAR SCREEN
      MOV   AH,00       ;SET MODE
      MOV   AL,06       ;MODE = 06 (CGA HIGH RESOLUTION)
      INT   10H         ;INVOKE INTERRUPT TO CHANGE MODE
      MOV   CX,100      ;START LINE AT COLUMN =100 AND
      MOV   DX,50       ;ROW = 50
BACK: MOV   AH,0CH      ;AH=0CH TO DRAW A LINE
      MOV   AL,01       ;PIXELS = WHITE
      INT   10H         ;INVOKE INTERRUPT TO DRAW LINE
      INC   CX          ;INCREMENT HORIZONTAL POSITION
      CMP   CX,200      ;DRAW LINE UNTIL COLUMN = 200
      JNZ   BACK
```

REVIEW

1. Write a program that:

   (a) Clears the screen

   (b) Puts the cursor position at row = 5 and column = 12

   (c) Displays the string “DEPT.OF CSE, VCET”

2. what is the function of the following PROGRAM?

```
MOV AH, 02
MOV BH, 00
MOV DL,  20H
MOV DH, 10H
INT 10H
```

3. The following program is meant to set the cursor at position row = 14 and column = 20. Fix the error:

```
MOV AH, 02
MOV BH, 00
MOV DH, 14H
MOV DL, 20H
INT 10H
```

4. The following program is meant to clear the screen. Fix the error:

```
MOV AX, 0600H

MOV BH, 07

MOV CX,  0000

MOV DX, 184F

INT 10H
```

DOS INTERRUPT 21H

- INT21H is provided by DOS, which is BIOS-ROM based.
- When the OS is loaded into the computer, INT 21H can be invoked to perform some extremely useful functions.
- These functions are commonly referred to as *DOS INT 21H* function calls.

INT 21H Option 09: Outputting a String of Data to the Monitor

- INT 21H Option 09 – to send a set of ASCII data to the monitor
  - AH = 09
  - DX = the offset address of the ASCII data to be displayed
- INT 21H option 09 will display the ASCII data string pointed at by DX until it encounters the dollar sign "$“

```
DATA_ASC    DB    'The earth is but one country','$'

      MOV   AH,09          ;option 09 to display string of data
      MOV   DX,OFFSET DATA_ASC      ;DX= offset address of data
      INT   21H                     ;invoke the interrupt
```

- In the absence of a dollar sign?

INT 21H Option 02: Outputting a Single Character to the Monitor

- INT 21H Option 02 – to output a single character to the monitor
  - 02 is put in AH
- DL should be loaded with the character to be displayed, and then INT 21H is invoked.

```
MOV   AH,02      ;option 02 displays one character
MOV   DL,'J'     ;DL holds the character to be displayed
INT   21H        ;invoke the interrupt
```

INT 21H Option 01: Inputting a Single Character, with Echo

- INT 21H Option 01 – waits until a character is input from the keyboard, and then echoes it to the monitor
- After the interrupt, the input character (ASCII value) will be available in AL.

```
MOV   AH,01 ;option 01 inputs one character
INT   21H   ;after the interrupt, AL = input character (ASCII)
```

» The following Program: (a) clears the screen (b) sets the cursor to the center of the screen, and (c) starting at that point of the screen, displays the message "This is a test of the display routine"

```
TITLE       PROG4-1 SIMPLE DISPLAY PROGRAM
PAGE        60,132
            .MODEL SMALL
            .STACK       64
;-----------------------------
            .DATA
MESSAGE     DB    'This is a test of the display routine','$'
;-----------------------------
      .CODE
MAIN  PROC  FAR
      MOV   AX,@DATA
      MOV   DS,AX
      CALL  CLEAR              ;CLEAR THE SCREEN
      CALL  CURSOR             ;SET CURSOR POSITION
      CALL  DISPLAY            ;DISPLAY MESSAGE
      MOV   AH,4CH
      INT   21H                ;GO BACK TO DOS
MAIN  ENDP
;-----------------------------
```

```
;
;THIS SUBROUTINE CLEARS THE SCREEN
CLEAR PROC
     MOV   AX,0600H          ;SCROLL SCREEN FUNCTION
     MOV   BH,07             ;NORMAL ATTRIBUTE
     MOV   CX,0000           ;SCROLL FROM ROW=00,COL=00
     MOV   DX,184FH          ;TO ROW=18H,COL=4FH
     INT   10H               ;INVOKE INTERRUPT TO CLEAR SCREEN
     RET
CLEAR ENDP
;-----------------------------
;THIS SUBROUTINE SETS THE CURSOR AT THE CENTER OF THE SCREEN
CURSOR PROC
     MOV   AH,02             ;SET CURSOR FUNCTION
     MOV   BH,00             ;PAGE 00
     MOV   DH,12             ;CENTER ROW
     MOV   DL,39             ;CENTER COLUMN
     INT   10H               ;INVOKE INTERRUPT TO SET CURSOR POSITION
     RET
CURSOR ENDP
;-----------------------------
;THIS SUBROUTINE DISPLAYS A STRING ON THE SCREEN
DISPLAY  PROC
     MOV   AH,09             ;DISPLAY FUNCTION
     MOV   DX,OFFSET MESSAGE ;DX POINTS TO OUTPUT BUFFER
     INT   21H               ;INVOKE INTERRUPT TO DISPLAY STRING
     RET
DISPLAY  ENDP
     END   MAIN
```

INT 21H Option 0A: Inputting a String of Data from the Keyboard

- INT 21H Option 0A – get data from the keyboard, & store it in a predefined area of memory (buffer area) in the data segment, & echoes it on the screen as it is typed in
  - AH = 0AH and DX = offset address at which the string of data is stored
- Following program will accept up to six characters from the keyboard, including the return (carriage return) key. Six locations were reserved for the buffer and filled with FFH

```
ORG   0010H
DATA1 DB    6,?,6 DUP (FF);0010H=06, 0012H to 0017H = FF

      MOV   AH,0AH            ;string input option of INT 21H
      MOV   DX,OFFSET DATA1   ;load the offset address of buffer
      INT   21H               ;invoke interrupt 21H
```

» The memory contents of offset 0010H

0010	0011	0012	0013	0014	0015	0016	0017
06	00	FF	FF	FF	FF	FF	FF

» When the data comes in, the IBM PC will not exit the INT 21H routine until it encounters the return key

» Assuming the data that was entered was "USA" <RETURN>, the contents of memory locations would look like this:

0010	0011	0012	0013	0014	0015	0016	0017
06	03	55	53	41	0D	FF	FF
USACR							

0010H = 06	DOS requires the size of the buffer in the first location.
0011H = 03	The keyboard was activated three times (excluding the RETURN key) to key in the letters U, S, and A.
0012H = 55H	This is the ASCII hex value for letter U.
0013H = 53H	This is the ASCII hex value for letter S.
0014H = 41H	This is the ASCII hex value for letter A.
0015H = 0DH	This is the ASCII hex value for CR (carriage return).

» Inputting more than buffer size?

Use of Carriage Return and Line Feed

```
;Program 4-2 performs the following: (1) clears the screen, (2) sets
;the cursor at the beginning of the third line from the top of the
;screen, (3) accepts the message "IBM perSonal COmputer" from the
;keyboard, (4) converts lowercase letters of the message to uppercase,
;(5) displays the converted results on the next line.

TITLE       PROG4-2
PAGE        60,132
            .MODEL SMALL
            .STACK      64

;-----------------------------
            .DATA
BUFFER      DB    22,?,22 DUP (?)          ;BUFFER FOR KEYED-IN DATA
            ORG   18H
DATAREA     DB    CR,LF,22 DUP (?),'$'     ;DATA HERE AFTER CONVERSION
;DTSEG      ENDS
CR    EQU 0DH
LF    EQU 0AH

;-----------------------------
      .CODE
MAIN PROC   FAR
      MOV   AX,@DATA
      MOV   DS,AX
      CALL  CLEAR              ;CLEAR THE SCREEN
      CALL  CURSOR             ;SET CURSOR POSITION
      CALL  GETDATA            ;INPUT A STRING INTO BUFFER
      CALL  CONVERT            ;CONVERT STRING TO UPPERCASE
      CALL  DISPLAY            ;DISPLAY STRING DATAREA
      MOV   AH,4CH
      INT   21H          ;GO BACK TO DOS
MAIN ENDP

;-----------------------------
```

```
;THIS SUBROUTINE CLEARS THE SCREEN
CLEAR PROC
     MOV   AX,0600H          ;SCROLL SCREEN FUNCTION
     MOV   BH,07             ;NORMAL ATTRIBUTE
     MOV   CX,0000                ;SCROLL FROM ROW=00,COL=00
     MOV   DX,184FH          ;TO ROW=18H,4FH
     INT   10H               ;INVOKE INTERRUPT TO CLEAR SCREEN
     RET
CLEARENDP
;THIS SUBROUTINE SETS THE CURSOR TO THE BEGINNING OF THE 3RD LINE
CURSOR PROC
     MOV   AH,02             ;SET CURSOR FUNCTION
     MOV   BH,00             ;PAGE 0
     MOV   DL,01             ;COLUMN 1
     MOV   DH,03             ;ROW 3
     INT   10H               ;INVOKE INTERRUPT TO SET CURSOR
     RET
CURSOR ENDP

;-----------------------------
;THIS SUBROUTINE DISPLAYS A STRING ON THE SCREEN
DISPLAY PROC
     MOV   AH,09             ;DISPLAY STRING FUNCTION
     MOV   DX,OFFSET DATAREA ;DX POINTS TO BUFFER
     INT   21H               ;INVOKE INTERRUPT TO DISPLAY STRING
     RET
DISPLAY    ENDP
```

```
;-----------------------------
;THIS SUBROUTINE PUTS DATA FROM THE KEYBOARD INTO A BUFFER
GETDATA PROC
     MOV   AH,0AH              ;INPUT STRING FUNCTION
     MOV   DX,OFFSET BUFFER    ;DX POINTS TO BUFFER
     INT   21H                 ;INVOKE INTERRUPT TO INPUT STRING
     RET
GETDATA ENDP
;-----------------------------
;THIS SUBROUTINE CONVERTS ANY SMALL LETTER TO ITS CAPITAL
CONVERT PROC
     MOV   BX,OFFSET BUFFER
     MOV   CL,[BX]+1                 ;GET THE CHAR COUNT
     SUB   CH,CH                     ;CX = TOTAL CHARACTER COUNT
     MOV   DI,CX                     ;INDEXING INTO BUFFER
     MOV   BYTE PTR[BX+DI]+2,20H     ;REPLACE CR WITH SPACE
     MOV   SI,OFFSET DATAREA+2       ;STRING ADDRESS
AGAIN: MOV AL,[BX]+2                 ;GET THE KEYED-IN DATA
     CMP   AL,61H                    ;CHECK FOR 'a'
     JB    NEXT                      ;IF BELOW, GO TO NEXT
     CMP   AL,7AH                    ;CHECK FOR 'z'
     JA    NEXT                      ;IF ABOVE GO TO NEXT
     AND   AL,11011111B              ;CONVERT TO CAPITAL
NEXT: MOV [SI],AL                    ;PLACE IN DATA AREA
     INC   SI                        ;INCREMENT POINTERS
     INC   BX
     LOOP  AGAIN                     ;LOOP IF COUNTER NOT ZERO
     RET
CONVERT ENDP
     END   MAIN
```

```
TITLE       PROG4-3     READS IN LAST NAME AND DISPLAYS LENGTH
PAGE        60,132
            .MODEL SMALL
            .STACK 64 (?)
;----------------------------
            .DATA
MESSAGE1    DB    'What is your last name?','$'
            ORG   20H
BUFFER1     DB    9,?,9 DUP (0)
            ORG   30H
MESSAGE2    DB    CR,LF,'The number of letters in your name is: ','$'
ROW         EQU 08
COLUMN      EQU 05
CR          EQU 0DH     ;EQUATE CR WITH ASCII CODE FOR CARRIAGE RETURN
LF          EQU 0AH     ;EQUATE LF WITH ASCII CODE FOR LINE FEED
;----------------------------
            .CODE
MAIN        PROC  FAR
     MOV    AX,@DATA
     MOV    DS,AX
     CALL   CLEAR
     CALL   CURSOR
     MOV    AH,09              ;DISPLAY THE PROMPT
     MOV    DX,OFFSET MESSAGE1
     INT    21H
     MOV    AH,0AH             ;GET LAST NAME FROM KEYBOARD
     MOV    DX,OFFSET BUFFER1
     INT    21H
     MOV    BX,OFFSET BUFFER1 ;FIND OUT NUMBER OF LETTERS IN NAME
     MOV    CL,[BX+1]          ;GET NUMBER OF LETTERS
     OR     CL,30H             ;MAKE IT ASCII
     MOV    MESSAGE2+40,CL     ;PLACE AT END OF STRING
     MOV    AH,09              ;DISPLAY SECOND MESSAGE
     MOV    DX,OFFSET MESSAGE2
     INT    21H
     MOV    AH,4CH
     INT    21H          ;GO BACK TO DOS
MAIN ENDP
;----------------------------
```

```
CLEAR PROC                          ;CLEAR THE SCREEN
     MOV   AX,0600H
     MOV   BH,07
     MOV   CX,0000
     MOV   DX,184FH
     INT   10H
     RET
CLEAR ENDP
;-----------------------------
CURSOR PROC                              ;SET CURSOR POSITION
     MOV   AH,02
     MOV   BH,00
     MOV   DL,COLUMN
     MOV   DH,ROW
     INT   10H
     RET
CURSOR ENDP
     END   MAIN
```

Write a program to perform the following: (1) clear the screen, (2) set the cursor at row 5 and column 1 of the screen, (3) prompt "There is a message for you from Mr. Jones. To read it enter Y". If the user enters 'Y' or 'y' then the message "Hi! I must leave town tomorrow, therefore I will not be able to see you" will appear on the screen. If the user enters any other key, then the prompt "No more messages for you" should appear on the next line.

```
TITLE PROGRAM 4-4
PAGE 60,132
     .MODEL SMALL
     .STACK 64
;----------------------------
          .DATA
PROMPT1   DB    'There is a message for you from Mr. Jones. '
          DB    'To read it enter Y','$'
MESSAGE   DB    CR,LF,'Hi! I must leave town tomorrow, '
          DB    'therefore I will not be able to see you','$'
PROMPT2   DB    CR,LF,'No more messages for you','$'
;DTSEG    ENDS
CR        EQU 0DH
LF        EQU 0AH
;----------------------------
```

```
      .CODE
MAIN PROC  FAR
      MOV   AX,@DATA
      MOV   DS,AX
      CALL  CLEAR        ;CLEAR THE SCREEN
      CALL  CURSOR       ;SET CURSOR POSITION
      MOV   AH,09        ;DISPLAY THE PROMPT
      MOV   DX,OFFSET PROMPT1
      INT   21H
      MOV   AH,07        ;GET ONE CHAR, NO ECHO
      INT   21H
      CMP   AL,'Y'       ;IF 'Y', CONTINUE
      JZ    OVER
      CMP   AL,'y'
      JZ    OVER
      MOV   AH,09        ;DISPLAY SECOND PROMPT IF NOT Y
      MOV   DX,OFFSET PROMPT2
      INT   21H
      JMP   EXIT
OVER:MOV    AH,09        ;DISPLAY THE MESSAGE
      MOV   DX,OFFSET MESSAGE
      INT   21H
EXIT:MOV    AH,4CH
      INT   21H          ;GO BACK TO DOS
MAIN        ENDP
;-----------------------------
CLEAR PROC               ;CLEARS THE SCREEN
      MOV   AX,0600H
      MOV   BH,07
      MOV   CX,0000
      MOV   DX,184FH
      INT   10H
      RET
CLEAR       ENDP
;-----------------------------
CURSOR PROC                    ;SET CURSOR POSITION
      MOV   AH,02
      MOV   BH,00
      MOV   DL,05        ;COLUMN 5
      MOV   DH,08        ;ROW 8
      INT   10H
      RET
CURSOR ENDP
      END   MAIN
```

INT 21H Option 07: Keyboard Input without Echo

- INT 21H Option 07 – requires the user to enter a single character but that character is not displayed (or echoed) on the screen
- After execution of the interrupt, the PC waits until a single character is entered and provides the character in AL

```
MOV   AH,07 ;keyboard input without echo
INT   21H
```

Using LABEL Directive to Define a String Buffer

- The LABEL directive can be used in the data segment to assign multiple names to data.
- When used in the data segment it looks like this:

```
name   LABEL attribute
```

- The attribute can be BYTE, WORD, DWORD, FWORD, QWORD, or TBYTE

```
JOE    LABEL BYTE
TOM    DB    20 DUP(0)
```

REVIEW

1. Write a program that:

   (a) Clears the screen

   (b) Puts the cursor position at row = 15 and column = 20

   (c) Displays the prompt “What is your name?”

   (d) Gets a response from the keyboard and displays it at row = 18 and column = 20

15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 2 – QUIZ 2

INTERRUPTS IN x86 PC

Mahesh Prasanna K.

Dept. of CSE, VCET.

1. When any interrupt (software or hardware) is activated, the CPU jumps to a fixed and unique address (TRUE/FALSE)
2. There are _____ bytes of memory in the interrupt vector table for each "INT nn”
3. The beginning address of Interrupt Vector Table is _____ and the ending address is _____
4. The program associated with an interrupt is referred to as _____
5. The divide error exception handling is assigned to _____ interrupt

6. When an interrupt through INTR is executed, IF = 0 and TF = 0
7. CLI blocks only the INTR, not the NMI (TRUE/FLASE)
8. INTO is executed only of the OF is high (TRUE/FALSE)
9. The monitor in text mode has _____ rows and columns _____. The top left position is _____, _____ and the bottom right position is _____, _____
10. In text mode, there is one attribute byte associated with each _____ on the screen

11. The attribute byte to display background green, foreground white blinking is _____

12. INT _____ function calls reside in ROM BIOS, whereas INT _____ function calls are provided by DOS

1. When any interrupt (software or hardware) is activated, the CPU jumps to a fixed and unique address (TRUE/FALSE)
2. There are _____ bytes of memory in the interrupt vector table for each "INT nn“ (4)
3. The beginning address of Interrupt Vector Table is _____ and the ending address is _____ (00000H, 003FFH)
4. The program associated with an interrupt is referred to as _____ (Interrupt Service Routine or Interrupt handler)
5. The divide error exception handling is assigned to _____ interrupt (INT 0)

6. When an interrupt through INTR is executed, IF = 0 and TF = 0 (TRUE/FALSE)
7. CLI blocks only the INTR, not the NMI (TRUE/FLASE)
8. INTO is executed only of the OF is high (TRUE/FALSE)
9. The monitor in text mode has _____ rows and columns _____. The top left position is _____, _____ and the bottom right position is _____, _____ (25, 80, 00, 00, 24, 79)
10. In text mode, there is one attribute byte associated with each _____ on the screen (character)

11. The attribute byte to display background green, foreground white blinking is _____ (1010 0111)

12. INT _____ function calls reside in ROM BIOS, whereas INT _____ function calls are provided by DOS (10H, 21H)