15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 3

SIGNED NUMBERS & STRINGS

Mahesh Prasanna K.

Dept. of CSE, VCET.

SIGNED NUMBER ARTIHMETIC OPERATIONS

- The numbers used could be *positive* or *negative*
- Computers must be able to accommodate such numbers
  - The most significant bit (MSB) is set aside for the sign (+ or –)
  - The rest of the bits are used for the magnitude

D7	D6	D5	D4	D3	D2	D1	D0
sign	magnitude						

- For *negative numbers* D7 is 1, but the magnitude is represented in 2's complement

Show how the computer would represent –5.

Solution:

```
1. 0000 0101    5 in 8-bit binary
2. 1111 1010    invert each bit
3. 1111 1011    add 1 (hex = FBH)
```

This is the signed number representation in 2's complement for –5.

Decimal	Binary	Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
...	....	..
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
...	... ...	...
+127	0111 1111	7F

Show –34H as it is represented internally.

Solution:

```
1. 0011 0100
2. 1100 1011
3. 1100 1100    (which is CCH)
```

Show the representation for -128_{10}.

Solution:

```
1.  1000 0000
2.  0111 1111
3.  1000 0000  Notice that this is not negative zero (–0).
```

Word-sized Signed Numbers

Decimal	Binary	Hex
-32,768	1000 0000 0000 0000	8000
-32,767	1000 0000 0000 0001	8001
-32,766	1000 0000 0000 0010	8002
...	...	...
...	...	...
-2	1111 1111 1111 1110	FFFE
-1	1111 1111 1111 1111	FFFF
0	0000 0000 0000 0000	0000
+1	0000 0000 0000 0001	0001
+2	0000 0000 0000 0010	0002
...	...	...
...	...	...
+32,766	0111 1111 1111 1110	7FFE
+32,767	0111 1111 1111 1111	7FFF

Overflow Problem in Signed Number Operations

» What is an overflow?

Look at the following code and data segments:

```
DATA1      DB     +96
DATA2      DB     +70
           ...    ....
           MOV    AL,DATA1     ;AL=0110 0000 (AL=60H)
           MOV    BL,DATA2     ;BL=0100 0110 (BL=46H)
           ADD    AL,BL        ;AL=1010 0110 (AL=A6H= 90 invalid!)

+ 96 0110 0000
+ 70 0100 0110
+166 1010 0110 According to the CPU, this is 90, which is wrong. (OF = 1, SF = 1, CF = 0)
```

» The *designers of the CPU created the overflow flag specifically for the purpose of informing the programmer that the result of the signed number operation is erroneous*.

When the Overflow Flag is Set in 8-bit Operation?

» There is a carry from D6 to D7, but no carry out of D7 (CF = 0)

» There is a carry from D7 out (CF = 1), but no carry from D6 to D7

Observe the results of the following:

```
MOV   DL,-128     ;DL=1000 0000  (DL=80H)
MOV   CH,-2       ;CH=1111 1110  (CH=FEH)
ADD   DL,CH       ;DL=0111 1110  (DL=7EH=+126 invalid!)

  -128   1000 0000
+   -2   1111 1110
  -130   0111 1110 OF=1, SF=0 (positive), CF=1
```

According to the CPU, the result is +126, which is wrong. The error is indicated by the fact that OF = 1.

Observe the results of the following:

```
MOV   AL,-2       ;AL=1111 1110  (AL=FEH)
MOV   CL,-5       ;CL=1111 1011  (CL=FBH)
ADD   CL,AL       ;CL=1111 1001  (CL=F9H=7 which is correct)

  -2  1111 1110
+ -5  1111 1011
  -7  1111 1001
```

OF = 0, CF = 0 , and SF = 1 (negative); the result is correct since OF = 0.

Observe the results of the following:

```
MOV   DH,+7       ;DH=0000 0111     (DH=07H)
MOV   BH,+18      ;BH=0001 0010     (BH=12H)
ADD   BH,DH       ;BH=0001 1001     (BH=19H=+25, correct)

  +7      0000 0111
+ +18     0001 0010
  +25     0001 1001
```

OF = 0, CF = 0, and SF = 0 (positive).

When the Overflow Flag is Set in 16-bit Operation?

- There is a carry from D14 to D15, but no carry out of D15 (CF = 0)
- There is a carry from D15 out (CF = 1), but no carry from D14 to D15

Observe the results in the following:

```
MOV   AX,6E2FH    ; 28,207
MOV   CX,13D4H    ;+ 5,076
ADD   AX,CX       ;= 33,283 is the expected answer

6E2F          0110 1110 0010 1111
+13D4         0001 0011 1101 0100
8203          1000 0010 0000 0011  = - 32,253 incorrect!
              OF = 1, CF = 0, SF = 1
```

Observe the results in the following:

```
MOV   DX,542FH    ; 21,551
MOV   BX,12E0H    ; +4,832
ADD   DX,BX       ;=26,383

543F          0101 0100 0010 1111
+12E0         0001 0010 1110 0000
670F          0110 0111 0000 1111 = 26,383 (correct answer); OF = 0, CF = 0, SF = 0
```

Avoiding Erroneous Result in Signed Operations

» Sign extension copies;

» the sign bit (D7) of the lower byte of a register into the upper bits of the register – **CBW**

» the sign bit of a 16-bit register into another register – **CWD**

```
MOV   AL,+96      ;AL=0110 0000
CBW               ;now AH=0000 0000 and AL=0110 0000
```

or:

```
MOV   AL,-2       ;AL=1111 1110
CBW               ;AH=1111 1111 and AL=1111 1110
```

example:

```
MOV    AX,+260      ;AX=0000 0001 0000 0100 or AX=0104H
CWD                 ;DX=0000H and AX=0104H
```

example:

```
MOV    AX,-32766    ;AX=1000 0000 0000 0010B or AX=8002H
CWD                 ;DX=FFFF and AX=8002
```

Rewrite Example 6-4 to provide for handling the overflow problem.

Solution:

```
DATA1     DB     +96
DATA2     DB     +70
RESULT    DW     ?
          ......
          SUB    AH,AH        ;AH=0
          MOV    AL,DATA1     ;GET OPERAND 1
          MOV    BL,DATA2     ;GET OPERAND 2
          ADD    AL,BL        ;ADD THEM
          JNO    OVER         ;IF OF=0 THEN GO TO OVER
          MOV    AL,DATA2     ;OTHERWISE GET OPERAND 2 TO
          CBW                 ;SIGN EXTEND IT
          MOV    BX,AX        ;SAVE IT IN BX
          MOV    AL,DATA1     ;GET BACK OPERAND 1 TO
          CBW                 ;SIGN EXTEND IT
          ADD    AX,BX        ;ADD THEM AND
OVER:     MOV    RESULT,AX    ;SAVE IT
```

S	AH	AL		
0	000 0000	0110 0000	+96	after sign extension
0	000 0000	0100 0110	+70	after sign extension
0	000 0000	1010 0110	+166	

IDIV

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

Notes:
1. Divide error interrupt if −127 > AL > +127.
2. Divide error interrupt if −32,767 > AL > +32,767.

Eg1:

IDIV CH

		Before	After	
F0H = -10H	CH	F0H		EE = -12H
	AL	25H	EEH	Quotient
	AH	01H	05H	Remainder

Eg2:

IDIV BL

		Before	After	
F0H = -3H	BL	FDH		FB = -5H
	AL	10H	EBH	Quotient
	AH	00H	01H	Remainder

```
TITLE       PROG 6-1          FIND THE AVERAGE TEMPERATURE
PAGE        60,132
            .MODEL STMALL
            .STACK 64
;-----------------------
            .DATA
SIGN_DAT    DB +13,-10,+19,+14,-18,-9,+12,-19,+16
            ORG 0010H
AVERAGE     DW ?
REMAINDER   DW ?
;-----------------------
      .CODE
MAIN PROC   FAR
      MOV   AX,@DATA
      MOV   DS,AX
      MOV   CX,9                      ;LOAD COUNTER
      SUB   BX,BX                     ;CLEAR BX, USED AS ACCUMULATOR
      MOV   SI,OFFSET SIGN_DAT        ;SET UP POINTER
BACK:MOV    AL,[SI]                   ;MOVE BYTE INTO AL
      CBW                             ;SIGN EXTEND INTO AX
      ADD   BX,AX                     ;ADD TO BX
      INC   SI                        ;INCREMENT POINTER
      LOOP  BACK                      ;LOOP IF NOT FINISHED
      MOV   AL,9                      ;MOVE COUNT TO AL
      CBW                             ;SIGN EXTEND INTO AX
      MOV   CX,AX                     ;SAVE DENOMINATOR IN CX
      MOV   AX,BX                     ;MOVE SUM TO AX
      CWD                             ;SIGN EXTEND THE SUM
      IDIV  CX                        ;FIND THE AVERAGE
      MOV   AVERAGE,AX                ;STORE THE AVERAGE (QUOTIENT)
      MOV   REMAINDER,DX              ;STORE THE REMAINDER
      MOV   AH,4CH
      INT   21H                       ;GO BACK TO DOS
MAIN ENDP
      END   MAIN
```

IMUL

Multiplication	Operand 1	Operand 2	Result
byte × byte	AL	register or memory	AX[1]
word × word	AX	register or memory	DX AX[2]
word × byte	AL = byte CBW	register or memory	DX AX[2]

Notes:

1. CF = 1 and OF = 1 if AH has part of the result, but if the result is not large enough to need the AH, the sign bit is copied to the unused bits and the CPU makes CF = 0 and OF = 0 to indicate that.
2. CF = 1 and OF = 1 if DX has part of the result, but if the result is not large enough to need the DX, the sign bit is copied to the unused bits and the CPU makes CF = 0 and OF = 0 to indicate that. One can use the J condition to find out which of the conditions above has occurred. The rest of the flags are undefined.

Eg1:

IMUL CH

FEH = -02

	Before	After
CH	FEH	
AL	02H	FCH
AH	34H	FFH

FFFCH = -04

SAR (shift arithmetic right)

```
SAR    destination,count
```

Eg:

SAR BH, CL

Shift right

1100 0000 = -40H

1111 0000 = -10H

	Before	*After*
BH	1100 0000	1111 0000
CL	02H	
Cy	1	0

```
MOV   AL,-10      ;AL=-10=F6H=1111 0110
SAR   AL,1        ;AL is arithmetic shifted right once
                  ;AL=1111 1011=FDH=-5
```

Using DEBUG, evaluate the results of the following:

```
MOV   AX,-9
MOV   BL,2
IDIV  BL          ;divide -9 by 2 results in FCH
MOV   AX,-9
SAR   AX,1        ;divide -9 by 2 with arithmetic shift
                  ;results in FBH
```

Solution:

The DEBUG trace demonstrates that an IDIV of –9 by 2 gives FCH (– 4), whereas SAR –9 gives FBH (–5). This is because SAR rounds negative numbers down but IDIV rounds up.

SAL (shift arithmetic left)

Signed Number Comparison

```
CMP    dest,source
```

» In unsigned number comparisons, CF and ZF are checked for conditions of larger, equal, and smaller

» In signed number comparison, OF, ZF, and SF are checked

```
destination > source    OF=SF or ZF=0
destination = source    ZF=1
destination < source    OF=negation of SF
```

```
JG    Jump Greater            jump if OF=SF or ZF=0
JGE   Jump Greater or Equal   jump if OF=SF
JL    Jump Less               jump if OF=inverse of SF
JLE   Jump Less or Equal      jump if OF=inverse of SF or ZF=1
JE    Jump if Equal           jump of ZF = 1
```

```
TITLE       PROG6-2      ;FIND THE LOWEST TEMPERATURE
PAGE        60,132
;-----------------
            .MODEL SMALL
            .STACK 64
;-----------------
            .DATA
SIGN_DAT    DB    +13,-10,+19,+14,-18,-9,+12,-19,+16
            ORG   0010H
LOWEST      DB    ?
;-----------------
      .CODE
MAIN PROC   FAR
      MOV   AX,@DATA
      MOV   DS,AX
      MOV   CX,8                    ;LOAD COUNTER (NUMBER ITEMS - 1)
      MOV   SI,OFFSET SIGN_DAT      ;SET UP POINTER
      MOV   AL,[SI]             ;AL HOLDS LOWEST VALUE FOUND SO FAR
BACK:INC    SI                  ;INCREMENT POINTER
      CMP   AL,[SI]             ;COMPARE NEXT BYTE TO LOWEST
      JLE   SEARCH              ;IF AL IS LOWEST, CONTINUE SEARCH
      MOV   AL,[SI]             ;OTHERWISE SAVE NEW LOWEST
SEARCH:LOOP BACK                ;LOOP IF NOT FINISHED
      MOV   LOWEST,AL           ;SAVE LOWEST TEMPERATURE
      MOV   AH,4CH
      INT   21H                 ;GO BACK TO DOS
MAIN ENDP
      END   MAIN
```

REVIEW

1. Convert 16H to its 2's complement representation
2. Explain the difference between an overflow and a carry
3. Explain the purpose of the CBW and CWD instructions
4. Demonstrate the effect of CBW on AL = F6H
5. Demonstrate the effect of CWO on AX= 124CH
6. Explain the difference between the SHR and SAR instructions
7. For each of the following instructions, indicate the flag condition necessary for each jump to occur:

   (a) JLE (b) JG

8. Find the overflow flag for each case –

(a) (+15) + (-12) (b) (-123) + (-127)

(c) (+25H) + (+34) (d) (-127) + (+127)

(e) (+1000) + (-100)

9. Sign-extend the following –

(a) -122 (b) -999H (c) +17H

(d) +127 (e) -129

STRING & TABLE OPERATIONS

- Capable of performing operations on a series of operands located in consecutive memory locations
- Byte (B) and Word (W) Operands in String Instructions
- Use of SI and DI, DS and ES in String Instructions
- DF, the Direction Flag – CLD & STD

Summary of String Operations

Instruction	Mnemonic	Destination	Source	Prefix
Move string byte	MOVSB	ES: DI	DS: SI	REP
Move string word	MOVSW	ES: DI	DS: SI	REP
Store string byte	STOSB	ES: DI	AL	REP
Store string word	STOSW	ES: DI	AX	REP
Load string byte	LODSB	AL	DS: SI	None
Load string word	LODSW	AX	DS: SI	None
Compare string byte	CMPSB	ES: DI	DS: SI	REPE/REPNE
Compare string word	CMPSW	ES: DI	DS: SI	REPE/REPNE
Scan string byte	SCASB	ES: DI	AL	REPE/REPNE
Scan string word	SCASW	ES: DI	AX	REPE/REPNE

REP/REPZ/PERNZ Prefix

Instruction Code	Condition for Exit
REP	CX = 0
REPE/REPZ	CX = 0 or ZF = 0
REPNE/REPNZ	CX = 0 or ZF = 1

Using string instructions, write a program that transfers a block of 20 bytes of data.

Solution:

```
;in the data segment:
DATA1 DB        'ABCDEFGHIJKLMNOPQRST'
      ORG  30H
DATA2 DB        20 DUP (?)

;in the code segment:
      MOV   AX,@DATA
      MOV   DS,AX                 ;INITIALIZE THE DATA SEGMENT
      MOV   ES,AX                 ;INITIALIZE THE EXTRA SEGMENT
      CLD                         ;CLEAR DIRECTION FLAG FOR AUTOINCREMENT
      MOV   SI,OFFSET DATA1       ;LOAD THE SOURCE POINTER
      MOV   DI,OFFSET DATA2       ;LOAD THE DESTINATION POINTER
      MOV   CX,20                 ;LOAD THE COUNTER
      REP   MOVSB                 ;REPEAT UNTIL CX BECOMES ZERO
```

MOV CX, 10

REP MOVSB

Write a program that:
(1) Uses STOSB to store byte AAH in 100 memory locations.
(2) Uses LODS to test the contents of each location to see if AAH is there. If the test fails, the system should display the message "bad memory".

Solution:

Assuming that ES and DS have been assigned in the ASSUME directive, the following is from the code segment:

```
        ;PUT PATTERN AAAAH IN TO 50 WORD LOCATIONS
        MOV   AX,DTSEG            ;INITIALIZE
        MOV   DS,AX               ;DS REG
        MOV   ES,AX               ;AND ES REG
        CLD                       ;CLEAR DF FOR INCREMENT
        MOV   CX,50               ;LOAD THE COUNTER (50 WORDS)
        MOV   DI,OFFSET MEM_AREA  ;LOAD THE POINTER FOR DESTINATION
        MOV   AX,0AAAAH           ;LOAD THE PATTERN
        REP   STOSW               ;REPEAT UNTIL CX=0
        ;BRING IN THE PATTERN AND TEST IT ONE BY ONE
        MOV   SI,OFFSET MEM_AREA  ;LOAD THE POINTER FOR SOURCE
        MOV   CX,100              ;LOAD THE COUNT (COUNT 100 BYTES)
AGAIN:  LODSB                     ;LOAD INTO AL FROM DS:SI
        XOR   AL,AH               ;IS PATTERN THE SAME?
        JNZ   OVER                ;IF NOT THE SAME THEN EXIT
        LOOP  AGAIN               ;CONTINUE UNTIL CX=0
        JMP   EXIT                ;EXIT PROGRAM
OVER:   MOV   AH,09               ;{ DISPLAY
        MOV   DX, OFFSET MESSAGE  ;{ THE MESSAGE
        INT   21H                 ;{ ROUTINE
EXIT: ..
```

Assuming that there is a spelling of "Europe" in an electronic dictionary and a user types in "Euorope", write a program that compares these two and displays the following message, depending on the result:

1. If they are equal, display "The spelling is correct".
2. If they are not equal, display "Wrong spelling".

Solution:

```
DAT_DICT    DB    'Europe'
DAT_TYPED   DB    'Euorope'
MESSAGE1    DB    'The spelling is correct','$'
MESSAGE2    DB    'Wrong spelling','$'

;from the code segment:
            CLD                                ;DF=0 FOR INCREMENT
            MOV   SI,OFFSET DAT_DICT           ;SI=DATA1 OFFSET
            MOV   DI,OFFSET DAT_TYPED          ;DI=DATA2 OFFSET
            MOV   CX,06                        ;LOAD THE COUNTER
            REPE  CMPSB          ;REPEAT AS LONG AS EQUAL OR UNTIL CX=0
            JE    OVER                   ;IF ZF=1 THEN DISPLAY MESSAGE1
            MOV   DX,OFFSET MESSAGE2     ;IF ZF=0 THEN DISPLAY MESSAGE2
            JMP   DISPLAY
OVER:       MOV   DX,OFFSET MESSAGE1
DISPLAY:    MOV   AH,09
            INT   21H
```

Write a program that scans the name "Mr. Gones" and replaces the "G" with the letter "J", then displays the corrected name.

Solution:

```
;in the data segment:
DATA1      DB     'Mr. Gones','$'

;and in the code segment:

           MOV    AX,@DATA
           MOV    DS,AX
           MOV    ES,AX
           CLD                        ;DF=0 FOR INCREMENT
           MOV    DI,OFFSET DATA1     ;ES:DI=ARRAY OFFSET
           MOV    CX,09               ;LENGTH OF ARRAY
           MOV    AL,'G'              ;SCANNING FOR THE LETTER 'G'
           REPNE  SCASB               ;REPEAT THE SCANNING IF NOT EQUAL
;or
           JNE    OVER                ;UNTIL CX IS ZERO. JUMP IF Z=0
           DEC    DI                  ;DECREMENT TO POINT AT 'G'
           MOV    BYTE PTR [DI],'J'   ;REPLACE 'G' WITH 'J'
OVER:      MOV    AH,09               ;DISPLAY
           MOV    DX,OFFSET DATA1     ;THE
           INT    21H                 ;CORRECTED NAME
```

XLAT

- To access the elements of the table, 8088/86 microprocessors provide the XLAT (translate) instruction
- Assume a table for the values of x^2, where x is between 0 and 9

```
SQUR_TABLE  DB    0,1,4,9,16,25,36,49,64,81
```

- To access the square of any number from 0 to 9:
  - BX must have the offset address of the look-up table
  - Number whose square is required must be in the AL register
- After XLAT, the AL register will have the square of the number

```
MOV   BX,OFFSET SQUR_TABLE ;load the offset address of table
MOV   AL,05        ;AL=05 will retrieve 6th element
XLAT               ;pull the element out of table
                   ;and put in AL
```

```
SUB   AH,AH
MOV   SI,AX
MOV   AL,[BX+SI]
```

Code Conversion using XLAT

```
;data segment:
ASC_TABL    DB    '0','1','2','3','4','5','6','7','8'
            DB    '9','A','B','C','D','E','F'
HEX_VALU    DB    ?
ASC_VALU    DB    ?
;code segment:
            MOV   BX,OFFSET ASC_TABL        ;BX= TABLE OFFSET
            MOV   AL,HEX_VALU               ;AL=THE HEX DATA
            XLAT                      ;GET THE ASCII EQUIVALENT
            MOV   ASC_VALU,AL               ;MOVE IT TO MEMORY
```

REVIEW

1. In the following program segment, what condition will cause the REPNZ to fail?

```
MOV SI, OFFSET DATAl
MOV DI, OFFSET DATA2
MOV CX , LENGTH
REPNZ CMPSB
```

2. Write a program that transfers a block of 200 words of data
3. Write a program that scan the initials “IbM” and replaces the lowercase “b” with uppercase “B”
4. Write a program using look-up table and XLAT to retrieve the y value in the equation $y = x^2 + 2x + 5$ for x values of 0 to 9

15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 3 – QUIZ 1

SIGNED NUMBERS & STRINGS

Mahesh Prasanna K.

Dept. of CSE, VCET.

1. In an 8-bit operand, bit _____ is used for the sign bit; whereas in a 16-bit operand, bit _____ is used for the sign bit
2. The range of byte-sized operands is _____ to _____
3. The range of word-sized operands is _____ to _____
4. In string operations, register _____ is used to point to the source operand and register _____ is used to point to the destination operand

5. SI is used as an offset into the _____ segment, and DI is used as an offset into the _____
6. The _____ flag, bit _____ of the flag register, is used to tell the CPU whether to increment or decrement pointers in repeated string operations
7. STD instruction is used to _____

1. In an 8-bit operand, bit _____ is used for the sign bit; whereas in a 16-bit operand, bit _____ is used for the sign bit (D7, D15)
2. The range of byte-sized operands is _____ to _____ (–128 to +127)
3. The range of word-sized operands is _____ to _____ (–32,768 to +32,767)
4. In string operations, register _____ is used to point to the source operand and register _____ is used to point to the destination operand (SI, DI)

5. SI is used as an offset into the _____ segment, and DI is used as an offset into the _____ (Data, Extra)
6. The _____ flag, bit _____ of the flag register, is used to tell the CPU whether to increment or decrement pointers in repeated string operations (DF, 11 or D10)
7. STD instruction is used to _____ (set the DF)

15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 3

MEMORY & MEMORY INTERFACING

Mahesh Prasanna K.

Dept. of CSE, VCET.

SEMICONDUCTOR MEMORIES

- Semiconductor memories – used as primary storage for code and data – connected directly to the CPU – *primary memory*
- ***ROM*** **–** system software and permanent system data
- ***RAM*** **–** temporary data and application software
- Number of bits, semiconductor memory chip can store – *capacity*
- Organized into a number of locations – 1 bit, 4-bits, 8-bits, or even 16-bits
- 2^x locations; x-address pins
- Each location contains y bits; y-data pins

- 2^x x y bits – the *capacity*

Examples: 1] A given memory chip has 12 address pins and 8 data pins. Find the memory organization and the capacity.

Solution:

- ⇨ Memory chip has 12 address lines ↔ 2^{12} = 4,096 locations.
- ⇨ Memory chip has 8 data lines ↔ Each location hold 8 bits of data.
- ⇨ Thus, the memory organization is 4,096 x 8 = 4K x 8 = 32K bits capacity.

Examples: 2] A 512K memory chip has 8 data pins. Find the organization.

Solution:

- ⇨ The memory chip has 8 data lines ↔ Each location within the chip can hold 8 bits of data.
- ⇨ Given, the capacity of the memory chip = 512K.
- ⇨ Hence, the locations within the memory chip = 512K / 8 = 64K.
- ⇨ Since, 2^{16} = 64K; the memory chip has 16 address lines.
- ⇨ Hence, the memory organization is: 64K x 8 = 512K bits capacity.

MEMORY ADDRESS DECODING

- Consider a 32K x 8 capacity memory chip.
  - Chip has 15 (2^{15} = 32K) address lines and 8 data lines.
- To be interface to x86 microprocessor (which is having 20 address lines and 16 data lines)
  - Microprocessor sends out a 20-bit memory address whenever it reads or writes data
- Hence there is a mismatch that must be corrected
- The *decoder* corrects the mismatch by decoding the address pins that do not connect to the memory component

Simple Logic Gates as Address Decoder

A19				A0	
0000	1000	0000	0000	0000	= 08000H address of the first location
0000	1111	1111	1111	1111	= 0FFFFH address of the last location

A19				A0	
1001	0000	0000	0000	0000	= 90000H address of the first location
1001	1111	1111	1111	1111	= 9FFFFH address of the last location

Referring to above Fig., we see that the memory chip has 64K bytes of space. Show the calculation that verifies that address range 90000 to 9FFFFH is comprised of 64K bytes.

Solution:

To calculate the total number of bytes for a given memory address range, subtract the two addresses and add 1 to get the total bytes in hex. Then the hex number is converted to decimal and divided by 1024 to get K bytes.

```
 9FFFF        FFFF
-90000       +    1
 0FFFF        10000 hex = 65,536 decimal = 64K
```

Using the 74LA138 as Decoder

Block Diagram

Function Table

Inputs															
Enable		Select			Outputs										
G1	G2	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7			
X	H	X	X	X	H	H	H	H	H	H	H	H			
L	X	X	X	X	H	H	H	H	H	H	H	H			
H	L	L	L	L	L	H	H	H	H	H	H	H			
H	L	L	L	H	H	L	H	H	H	H	H	H			
H	L	L	H	L	H	H	L	H	H	H	H	H			
H	L	L	H	H	H	H	H	L	H	H	H	H			
H	L	H	L	L	H	H	H	H	L	H	H	H			
H	L	H	L	H	H	H	H	H	H	L	H	H			
H	L	H	H	L	H	H	H	H	H	H	L	H			
H	L	H	H	H	H	H	H	H	H	H	H	L			

Looking at the design in above Fig. , find the address range for (a) Y4, (b) Y2, and (c) Y7, and verify the block size controlled by each Y.

Solution:

(a) The address range for Y4 is calculated as follows.

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The above shows that the range for Y4 is F0000H to F3FFFH. In Figure 10-13, notice that A19, A18, and A17 must be 1 for the decoder to be activated. Y4 will be selected when A16 A15 A14 = 100 (4 in binary). The remaining A13–A0 will be 0 for the lowest address and 1 for the highest address.

(b) The address range for Y2 is E8000H to EBFFFH.

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(c) The address range for Y7 is FC000H to FFFFFH. Notice that FFFFF – FC000H = 3FFFH, which is equal to 16,383 in decimal. Adding 1 to it because of the 0 location, we have 16,384. 16,384/1024 = 16K, the block (chip) size.

A circuit that uses eight 2764 EPROMs for a 64K × 8 section of memory in an 8088 microprocessor -based system. The addresses selected in this circuit are F0000H–FFFFFH.

REVIEW

1. Find the organization and chip capacity for –

   (a) 11 address, 8 data SRAM (b) 13 address, 8 data ROM

   (c) 8 address, 4 data DRAM (d) 9 address, 1 data DRAM

2. Find the capacity and number of pins for –

   (a) 16K x 8 SRAM (b) 32K x 8 EPROM

   (c)1M x 1 DRAM (d) 256K x 4 DRAM

   (e) 64K x 8 EEPROM (f) 1M x 4 DRAM

3. Using NAND gates and inverters, design decoding circuitry for the address range 0C0000H – 0C0FFFH

4. Find the address range of the following memory design

5. Using 74LS138, design the memory decoding circuitry in which the memory block controlled by Y0 is in the range 00000H to 03FFFH

6. Using the 74LS138 and OR gates, design memory decoding circuitry in which the memory block controlled by Y0 is in the 80000H to 807FFH space.

7. Find the address range for the following design

DATA INTEGRITY IN RAM & ROM

- When storing data; *data integrity – ensuring that, the data retrieved is the same as the data stored*
- When transferring data; *data integrity – ensuring that, the data received is the same as the data transmitted*
  - *Checksum* method – ROM and
  - *Parity bit* – DRAM
  - Mass storage devices (hard disks) and for transferring data on the Internet, the *CRC* (*cyclic redundancy check*)

Checksum Byte

- ROM may be corrupted – during the current surge, or when the PC is turned on, or during operation
- Checksum will detect any corruption of the contents of ROM
- Uses a *checksum byte* – an extra byte that is tagged to the end of a series of bytes of data. To calculate –
  - Add the bytes together and drop the carries
  - Take the 2's complement of the total sum, and that is the checksum byte, which becomes the last byte of the stored information
- To perform the checksum operation, add all the bytes, including the checksum byte. The result must be zero. If it is not zero, one or more bytes of data have been changed (corrupted)

Assume that we have 4 bytes of hexadecimal data: 25H, 62H, 3FH, and 52H.
(a) Find the checksum byte.
(b) Perform the checksum operation to ensure data integrity.
(c) If the second byte 62H had been changed to 22H, show how checksum detects the error.

Solution:

(a) The checksum is calculated by first adding the bytes.

```
  25H
+ 62H
+ 3FH
+ 52H
1 18H
```

The sum is 118H, and dropping the carry, we get 18H. The checksum byte is the 2's complement of 18H, which is E8H.

(b) Adding the series of bytes including the checksum byte must result in zero. This indicates that all the bytes are unchanged and no byte is corrupted.

```
  25H
+ 62H
+ 3FH
+ 52H
+ E8H
2 00H  (dropping the carry)
```

(c) Adding the series of bytes including the checksum byte shows that the result is not zero, which indicates that one or more bytes have been corrupted.

```
  25H
+ 22H
+ 3FH
+ 52H
+ E8H
1 C0H  dropping the carry, we get C0H.
```

Assuming that the last byte of the following data is the checksum byte, show whether the data has been corrupted or not: 28H, C4H, BFH, 9EH, 87H, 65H, 83H, 50H, A7H, and 51H.

Solution:

The sum of the bytes plus the checksum byte must be zero; otherwise, the data is corrupted

28H + C4H + BFH + 9EH + 87H + 65H + 83H + 50H + A7H + 51H = 500H

By dropping the accumulated carries (the 5), we get 00. The data is not corrupted. See Figure 10-17 for a program that performs this verification.

Checksum Program

- When the PC is turned on – BIOS test the system ROM
- The code for such a test is stored in the BIOS ROM

```
                ;-----------------------------------
                ;     ROS CHECKSUM SUBROUTINE       :
                ;-----------------------------------
EC4C            ROS_CHECKSUM PROC NEAR   ;NEXT_ROS_MODULE
EC4C B90020                  MOV  CX,8192 ;NUMBER OF BYTES TO ADD
EC4F            ROS_CHECKSUM_CNT:  ;ENTRY PT. FOR OPTIONAL ROS TEST
EC4F 32C0                   XOR   AL,AL
EC51            C26:
EC51 0207                   ADD   AL,DS:[BX]
EC53 43                     INC   BX     ;POINT TO NEXT BYTE
EC54 E2FB                   LOOP  C26    ;ADD ALL BYTES IN ROS MODULE
EC56 0AC0                   OR    AL,AL ; SUM = 0?
EC58 C3                     RET
                ROS_CHECKSUM ENDP
```

Use of Parity Bit in DRAM Error Detection:

- System boards or memory modules are populated with DRAM chips of various organizations, depending on the time they were designed and the availability of a given chip at a reasonable cost
- The memory technology is changing so fast that DRAM chips on the boards have a different look every year or two. While early PCs used 64K DRAMs, current PCs commonly use 1G chips
- To understand the use of a parity bit in detecting data storage errors, we use some simple examples from the early PCs to clarify some very important design concepts

DRAM Memory Chips:

- The arrangement of DRAM chips on the system or memory module board is often referred to as a *memory bank*
  - For example, the 64K bytes of DRAM can be arranged as one bank of 8 IC chips of 64K x 1 organization, or 4 bank of 16K x 1 organization
- The first IBM PC introduced in 1981, used memory chip of l6K x l organization
- The following Figure shows the memory banks for 640K bytes of RAM using 256K and 1M DRAM chips
  - Notice the use of an extra bit for every byte of data to store the parity bit
  - With the extra parity bit every bank requires an extra chip of x 1 organization for parity check

d7 ... d4
d3 ... d0
Parity
Bank 3: 64K × 9
64K × 4
64K × 4
64K × 1
Bank 2: 64K × 9
64K × 4
64K × 4
64K × 1
Bank 1: 256K × 9
256K × 4
256K × 4
256K × 1
Bank 0: 256K × 9
256K × 4
256K × 4
256K × 1
Note:
64K × 4
is a single 256K-bit chip
256K × 4
is a single 1M-bit chip

» DRAM design and parity bit circuitry for a bank of DRAM

Parity Bit Generator and Checker in IBM PC:

- There are two types of errors that can occur in DRAM chips:
- *Hard error* – some bits or an entire row of memory cell inside the memory chip get stuck to high or low permanently, thereafter always producing 1 or 0 regardless of what you write into the cell(s)
- *Soft error* – a single bit is changed from 1 to 0 or from 0 to 1 due to current surge or certain kinds of particle radiation in the air. Parity is used to detect soft errors

- Including a parity bit to ensure data integrity in RAM
  - the most widely used method
  - the simplest and cheapest
- This method can only indicate if there is a difference between the data that was written to memory and the data that was read.
- It cannot correct the error as is the case with some high-performance computers.
  - In those computers and some of the x86-based servers, the **EDC** (error detection and correction) method is used to detect and correct the error bit.
- The early IBM PC and compatibles use the 74S280 parity bit generator and checker to implement the concept of the parity bit.

74LS280 Parity Bit Generator & Checker:

» The 74S280 chip has 9 inputs and 2 outputs. Depending on whether an even or odd number of ones appear in the input, the even or odd output is activated

Case	Inputs		Outputs	
	A – H	I	Even	ODD
1	Even	0	1	0
2	Even	1	0	1
3	Odd	0	0	1
4	Odd	1	1	0

» As can be seen from Table, if all 9 inputs have an even number of 1 bits, the even output goes high (as in cases 1 and 4). If the 9 inputs have an odd number of high bits, the odd output goes high (as in cases 2 and 3).

- The way the IBM PC uses this chip is as follows:
- In above Figure (Slide 54)), inputs A – H are connected to the data bus. I input is used as a parity bit to check the correctness of the byte of data read from memory
- When a byte of information is written to a given memory location in DRAM, the even-parity bit is generated and saved on the ninth DRAM chip as a parity bit with use of control signal. This is done by activating the tri-state buffer using . At this point, I of the 74S280 is equal to zero, since high
- When a byte of data is read from the same location, the parity bit is gated into the I input of the 74S280 through . This time the odd output is taken out and fed into a 74LS74. If there is a difference between the data written and the data read, the Q output (called PCK, parity bit check) of the 74LS74 is activated and Q activates NMI, indicating that there is a parity bit error, meaning that the data read is not the same as the data written. Consequently, it will display a parity bit error message

- For example, if the byte of data written to a location has an even number of ls, A to H has an even number of ls, and I is zero, then the even-parity output of 74S280 becomes 1 and is saved on parity bit DRAM. This is case 1 shown in the above Table.
- If the same byte of data is read and there is an even number of ls (the byte is unchanged), I from the ninth bit DRAM, which is 1, is input to the 74S280, even becomes low, and odd becomes high, which is case 2 in the above Table.
- This high from the odd output will be inverted and fed to the 74LS74, making Q low. This means that is high thereby indicating that the written byte is the same as the byte read and there is no errors occurred.
- If the number of 1s in the byte has changed from even to odd and the 1 from the saved parity DRAM makes the number of inputs even (case 4 above), the odd output becomes low, which is inverted and passed to the 74LS74 D flip-flop. This makes Q = 1 and = 0, which signals the NMI to display a parity bit error message on the screen.

REVIEW

1. find the checksum byte for the following bytes: 24H, 76H, F5H, 98H, 89H, 7AH, 61H, C2H
2. Show a simple program in Assembly language to find the checksum byte of the 8 bytes of information (given in Qno. 1). Assume that SI equals the offset address of the data
3. In a given PC, we have only 512K of memory on the motherboard. Show possible configurations and the number of chip used to add memory up to the maximum allowed by the limits of conventional memory if we have each of the following. Include the parity bit in your configuration and count

(a) 64K x 1 (b) 64K x 4 and 64K x 1

REVIEW

4. Find the checksum byte for the following bytes. 34H, 54H, 7FH, llH, E6H, 99H

5. For each of the following sets of data (the last byte is the checksum byte) verify if the data is corrupted.

   (a) 29H, lCH, 16H, 38H, and 6DH

   (b) 29H, lCH, 16H, 30H, and 6DH

6. A given PC need only 320K byte to reach the maxim um allowed conventional memory. Show the memory configuration using 256K x 1 and 64K x 1 memory chips. How many chips are needed? (Include the parity bit)

16 – BIT MEMORY INTERFACING

» In a 16-bit CPU such as the 80286, memory locations 00000-FFFFFH are designated as odd and even bytes

BHE	A0	Memory Selection	
0	0	Even Word	D0 – D15
0	1	Odd Byte	D8 – D15
1	0	Even Byte	D0 – D7
1	1	None	-

» 640K Bytes of DRAM with ODD & EVEN Banks Designation

» Use of A0 and BHE as bank selectors

Memory Cycle Time & Waite States:

» To access an external device – CPU provides fixed amount of time – bus cycle

» **MC** (memory cycle) time – memory read cycle time – 8088/86 (4 clocks) and from 286 to Pentium (only 2 clocks)

» If memory is slow and its access time does not match the MC time of the CPU, extra time can be requested from the CPU to extend the read cycle time. This extra time is called a wait state (WS)

Simplified 8086/8088 read bus cycle

- Factor in slowing down the CPU – Memory access time and the delay associated with signals going through the data and address path
- Delay associated with reading data stored in memory has the following two components:
  - The time taken for address signals to go from CPU pins to memory pins, (going through decoders and buffers (e.g., 74LS245)); plus the time taken for the data to travel from memory to CPU, is referred to as a ***path delay***
  - The *memory access time* to get the data out of the memory chip.
    - This is the larger (80% of the read cycle time) of the two components.
- The total sum of these two (path delay + memory access time) must equal the memory read cycle time provided by the CPU.

Calculate the memory cycle time of a 20-MHz 8386 system with
(a) 0 WS,
(b) 1 WS, and
(c) 2 WS.
Assume that the bus speed is the same as the processor speed.

Solution:

1/20 MHz = 50 ns is the processor clock period. Since the 386 bus cycle time of zero wait states is 2 clocks, we have:

	80386 20 MHz
Memory cycle time with 0 WS	$2 \times 50 = 100$ ns
Memory cycle time with 1 WS	$100 + 50 = 150$ ns
Memory cycle time with 2 WS	$100 + 50 + 500 = 200$ ns

It is preferred that all bus activities be completed with 0 WS. However, if the read and write operations cannot be completed with 0 WS, we request an extension of the bus cycle time. This extension is in the form of an integer number of WS. That is, we can have 1, 2, 3, and so on WS, but not 1.25 WS.

A 20-MHz 80386-based system is using ROM of 150 ns speed. Calculate the number of wait states needed if the path delay is 25 ns.

Solution:

If ROM access time is 150 ns and the path delay is 25 ns, every time the 80386 accesses ROM it must spend a total of 175 ns to get data into the CPU. A 20-MHz CPU with zero WS provides only 100 ns (2×50 ns = 100 ns) for the memory read cycle time. To match the CPU bus speed with this ROM we must insert 2 wait states. This makes the cycle time 200 ns ($100 + 50 + 50 = 200$ ns). Notice that we cannot ask for 1.5 WS since the number of WS must be an integer. That would be like going to the store and wanting to buy half an apple. You must get one or more complete WS or none at all.

Accessing EVEN & ODD Words:

- Intel defines 16-bit data as a word. The address of a word can start at an even or an odd number
  - For example, in the instruction "*MOV AX, [2000]*" the address of the word being fetched into AX starts at an even address
  - In the case of "*MOV AX, [2007]*" the address starts at an odd address
- In systems with a 16-bit data bus, accessing a word from an odd addressed location can be slower

» Accessing a word in 8-bit system – treated like accessing two bytes regardless of whether the address is odd or even

» Since accessing a byte takes one memory cycle, accessing any word will take 2 memory cycles.

- In the 16- bit system, accessing a word with an even address takes one memory cycle
  - This is because; one byte is carried on D0-D7 and the other on D8-Dl5 in the same memory cycle
- But, accessing a word with an odd address requires two memory cycles
  - For example, see how accessing the word in the instruction "*MOV AX, [F617]*" works as shown in following Fig.

```
                EVEN
VALUE1          DW      ?
```

Bus Bandwidth:

- The main advantage of the 16-bit data bus is; doubling of the rate of transfer of information between the CPU and the outside world.
  - The rate of data transfer is generally called *bus bandwidth*.
  - *Bus bandwidth* is a measure of how fast buses transfer information between the CPU and memory or peripherals. The wider the data bus, the higher the bus bandwidth.
- Advantage of the wider external data bus comes at the cost of increasing the size of the printed circuit board.
- Bus bandwidth is measured in MB (megabytes) per second and is calculated as follows: ***bus bandwidth = (1/bus cycle time) x bus width in bytes***
- In the above formula, bus cycle time can be either memory or I/O cycle time.

Calculate memory bus bandwidth for the following microprocessors if the bus speed is 20 MHz.

(a) 286 with 0 WS and 1 WS (16-bit data bus)
(b) 386 with 0 WS and 1 WS (32-bit data bus)

Solution:

The memory cycle time for both the 286 and 386 is 2 clocks, with zero wait states. With the 20 MHz bus speed we have a bus clock of 1/20 MHz = 50 ns.

(a) Bus bandwidth = (1/(2 × 50 ns)) × 2 bytes = 20M bytes/second (MB/s)
With 1 wait state, the memory cycle becomes 3 clock cycles
3 × 50 = 150 ns and the memory bus bandwidth is = (1/150 ns) × 2 bytes = 13.3 MB/S

(b) Bus bandwidth = (1/(2 × 50 ns)) × 4 bytes = 40 MB/s
With 1 wait state, the memory cycle becomes 3 clock cycles
3 × 50 = 150 ns and the memory bus bandwidth is = (1/150 ns) × 4 bytes = 26.6 MB/S

From the above it can be seen that the two factors influencing bus bandwidth are:

1. The read/write cycle time of the CPU
2. The width of the data bus

Notice in this example that the bus speed of the 286/386 was given as 20 MHz. That means that the CPU can access memory on the board at this speed. If this 286/386 is used on a PC board with an ISA expansion slot, it must slow down to 8 MHz when communicating with the ISA bus since the maximum bus speed for the ISA bus is 8 MHz. This is done by the chipset circuitry.

- There are two ways to increase the bus bandwidth:
  - Use a wider data bus
  - Shorten the bus cycle time

- While the data bus width has increased from 16-bit in the 80286 to 64-bit in the Pentium, the bus cycle time is reaching a maximum of 133 MHz.

REVIEW

1. How many memory chips are needed is we use 256K x 8, 256K x 1, and 64K x 1 memory chips for conventional memory of the 80286 PC/compatible?
2. What is the unit of bus bandwidth? Give the variables that affect the bus bandwidth.
3. Calculate the memory bus bandwidth for –
   a) 80286 of 10MHz and 0 WS
   b) 80286 of 16MHz and 0 WS

15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 3 – QUIZ 2

MEMORY & MEMORY INTERFACING

Mahesh Prasanna K.

Dept. of CSE, VCET.

1. The speed of semiconductor memory is in the range of _____
2. The MEMR signal from the CPU is connected to the _____ pin of the ROM chip
3. The MEMW signal from the CPU is connected to the _____ pin of the RAM chip
4. The CS pin of the memory chip is normally an _____ signal
5. The 74LS138 has total of _____ outputs
6. The Y output of the 74138 is _____

7. The 74S280 is both a parity generator and a checker (TRUE/FALSE)
8. To detect corruption of information stored in RAM and ROM memories, system designers use the _____ method for RAM and the _____ method for ROM
9. If A0 = 0 and BHE = 0, a byte is being transferred on the D0-D15 data bus (TRUE/FALSE)
10. If A0 = 0 and BHE = 1, a byte is being transferred on the D0-D7 data bus (TRUE/FALSE)

11. If A0 = 1 and BHE = 0, a byte is being transferred on the D8-D15 data bus (TRUE/FALSE)

12. In the instruction "*MOY AX, [2000]*", the transferring of data into the accumulator takes _____ memory cycles for the 8088 and _____ for the 80286

13. A 16-MHz 286 has a memory cycle time of _____ ns if it is used with a zero wait state

14. One way to increase the bus bandwidth is to widen the data bus (TRUE/FALSE)

1. The speed of semiconductor memory is in the range of _____ (nanoseconds)
2. The MEMR signal from the CPU is connected to the _____ pin of the ROM chip (OE)
3. The MEMW signal from the CPU is connected to the _____ pin of the RAM chip (WE)
4. The CS pin of the memory chip is normally an _____ signal (active-low, active-high)
5. The 74LS138 has total of _____ outputs (8)
6. The Y output of the 74138 is _____ (active low, active high)

7. The 74S280 is both a parity generator and a checker (TRUE/FALSE)
8. To detect corruption of information stored in RAM and ROM memories, system designers use the _____ method for RAM and the _____ method for ROM (parity bit generation/checker, checksum)
9. If A0 = 0 and BHE = 0, a byte is being transferred on the D0-D15 data bus (TRUE/FALSE)
10. If A0 = 0 and BHE = 1, a byte is being transferred on the D0-D7 data bus (TRUE/FALSE)

11. If A0 = 1 and BHE = 0, a byte is being transferred on the D8-D15 data bus (TRUE/FALSE)

12. In the instruction "*MOY AX, [2000]*", the transferring of data into the accumulator takes _____ memory cycles for the 8088 and _____ for the 80286 (2, 1)

13. A 16-MHz 286 has a memory cycle time of _____ ns if it is used with a zero wait state (125)

14. One way to increase the bus bandwidth is to widen the data bus (TRUE/FALSE)

15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 3

8255
I/O PROGRAMMING

Mahesh Prasanna K.
Dept. of CSE, VCET.

8088 INPUT/OUTPUT INSTRUCTIONS

- All x86 microprocessors can access external devices called ports
  - This is done using I/O instructions
- The x86 CPU has I/O space in addition to memory space
  - While memory can contain Opcode and data, I/O ports contain data only
- There are two instructions for this purpose: OUT and IN
  - These instructions can send data from the accumulator (AL or AX) to ports or bring data from ports into the accumulator
- In accessing ports, we can use an 8-bit or 16-bit data port

8-Bit Ports:

- The 8-bit I/O operation of the 8088 is applicable to all x86 CPUs from the 8088 to the Pentium
- The 8-bit port uses the D0-D7 data bus to communicate with I/O devices
- In 8-bit port programming,
  - register AL is used as the source of data, when using the OUT instruction; and as the destination, for the IN instruction
- Hence, to input or output data from any other registers, the data must first be moved to the AL register

» Instructions OUT and IN have the following formats:

	Inputting Data		Outputting Data	
Format:	IN	dest,source	OUT	dest,source
(1)	IN	AL,port#	OUT	port#,AL
(2)	MOV	DX,port#	MOV	DX,port#
	IN	AL,DX	OUT	DX,AL

» port# is the address of the port and can be from –

» 00 to FFH, allowing up to 256 input and 256 output ports – 8-bit port address is carried on address bus A0-A7

» 0000 to FFFFH, allowing up to 65,536 input and 65,536 output ports – 16-bit port address is carried on the address bus A0-A15 – dynamic compilations

- I/O instructions are widely used in programming peripheral devices such as printers, hard disks, and keyboards
- The port address can be either 8-bit or 16-bit
- For an 8-bit port address, we can use the immediate addressing mode

```
MOV   AL,36H      ;AL=36H
OUT   43H,AL      ;send value 36H to port address 43H
```

- The 8-bit address used in immediate addressing mode limits the number of ports to 256 for input plus 256 for output
- To have a larger number of ports we must use the 16-bit port address instruction

» To use the 16-bit port address, *register indirect addressing mode* must be used. The register used for this purpose is DX

```
BACK: MOV   DX,300H      ;DX = port address 300H
      MOV   AL,55H
      OUT   DX,AL        ;toggle the bits
      MOV   AL,0AAH
      OUT   DX,AL        ;toggle the bits
      JMP   BACK
```

» We can only use register DX for 16-bit I/O addresses; no other register can be used for this purpose.

» Also, notice the use of register AL for 8-bit data:

```
MOV DX,378H        ;DX=378 the port address
MOV AL,BL          ;load data into accumulator
OUT DX,AL          ;write contents of AL to port
                   ;whose address is in DX
```

- Just like the OUT instruction, the IN instruction uses the DX register to hold the address and AL to hold the arrived 8-bit data
  - In other words, DX holds the 16-bit port address while AL receives the 8-bit data brought in from an external port
  - The following program gets data from port address 300H and sends it to port address 302H

```
MOV   DX,300H     ;load port address
IN    AL,DX       ;bring in data
MOV   DX,302H
OUT   DX,AL       ;send it out
```

In a given 8088-based system, port address 22H is an input port for monitoring the temperature. Write Assembly language instructions to monitor that port continuously for the temperature of 100 degrees. If it reaches 100, then BH should contain 'Y'.

Solution:

```
BACK:   IN    AL,22H   ;get the temperature from port # 22H
        CMP   AL,100      ;is temp = 100?
        JNZ   BACK        ;if not, keep monitoring
        MOV   BH,'Y       ;temp = 100, load 'Y' into BH
```

REVIEW

1. In the x86 system, if we use only the 8-bit address bus A0-A7 for port addresses, what is the maximum number of (a) input, and (b) output ports?
2. The x86 can have a maximum of how many I/O ports?
3. What does the instruction "*OUT 24H, AL*" do?
4. Write Assembly language instructions to accept data input from port 300H and send it out to port 304H.
5. Write Assembly language instructions to place the status of port 60H in CH.

I/O ADDRESS DECODING & DESIGN

- The decoding of I/O ports is done by using TTL logic gates 74LS373 and 74LS244
- The following are the steps:
  - The control signals (IOR and IOW) are used along with the decoders
  - For an 8-bit port address, A0-A7 is decoded
  - If the port address is 16-bit (using DX), A0-A15 is decoded

Using 74LA373 in an Output Port Design:

» In every computer, whenever data is sent out by the CPU via the data bus, the data must be latched by the receiving device

» While memories have an internal latch to grab the data, a latching system must be designed for simple I/O ports

Function Table

Output Control	Enable G	Enable D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

» For an output latch, it is common to AND the output of the address decoder with the control signal IOW to provide the latching action as shown in Figure **OUT 99H, AL**

Show the design of an output port with an I/O address of 31FH using the 74LS373.

Solution:

31F9H is decoded, then ANDed with IOW to activate the G pin of the 74LS373 latch. This is shown in Figure below.

IN Port Design Using the 74LA244:

- When the data is coming in by way of a data bus, it must come in through a three-state buffer
- This is referred to as *tri-stated*

- Since 1G and 2G each control only 4 bits of 74LS244, both must be activated for 8 bits input
  - The following Fig shows the use of 74LS244 as an entry port to the system data bus
  - In the following Figures, the address decoder and IOR control signal together activate the tri-state input

Show the design of "IN AL,9FH" using the 74LS244 as a tri-state buffer.

Solution:

9FH is decoded, then ANDed with IOR. To activate OC of the 74LS244, it must be inverted since OC is an active-low pin. This is shown in Figure below.

Memory-Mapped I/O:

- Communicating with the I/O devices using IN and OUT instructions is referred to as *peripheral I/O* / *isolated I/O*
- Some new RISC processors do not have IN and OUT instructions; they use *memory-mapped I/O*
  - In memory-mapped I/O, a memory location is assigned to be an input and output port

The memory and I/O maps for the 8086/8088 microprocessors. (a) Isolated I/O (b) Memory-mapped I/O

Isolated (Peripheral) I/O	Memory-Mapped I/O
1. The IN and OUT instructions transfer data between the microprocessors accumulator or memory and the I/O device.	1. Instructions that access memory locations are used instead of IN and OUT instructions: *MOV AL, [2000]* will access the input port & *MOV [2000], AL* will access the output port.

2. Only A0-A15 are decoded; Hence, DS initialization is not required; decoding circuitry may be less expensive.	2. Entire 20-bit address, A0-A19, must be decoded (decoding circuitry is expensive); Hence DS must be loaded before accessing memory-mapped I/O:
	`MOV AX,3000H ;load the segment value`
	`MOV DS,AX`
	`MOV AL,[5000] ;get a byte from loc. 35000H`

3. IOR and IOW control signals are used.	3. MEMR and MEMW control signals are used.
4. Limited only to 65,536 input ports and 65,536 output ports.	4. The number of ports can be as high as 2^{20} (1,048,576).
5. Data should be moved to accumulator for any kind of operations.	5. Arithmetic and logic operations can be performed directly, without moving data to accumulator.
6. The user can expand the memory to its full size without using any memory space for I/O devices.	6. Uses memory address space, which could lead to memory space fragmentation.

REVIEW

1. Draw a logical design for "OUT 16H, AL" using AND and inverter gates in addition to a 74LS373
2. Draw a logical design for "IN AL,81H" using AND and inverter gates in addition to a 74LS244
3. Show one implementation of Problem 17 using NAND and inverter gates. Use as many as you need
4. Show one implementation of Problem 18 'using NAND and inverter gates. Use as many as you need

REVIEW

5. Show the logical design of "MOV [0100], AL" for memory-mapped I/O using AND and inverter gates and a 74LS373 latch. Assume that DS = B800H
6. Differentiate, memory-mapped I/O and isolated I/O

I/O ADDRESS MAP OF x86 PCs

» Any system that needs to be compatible with the x86 IBM PC must follow the I/O map of the following Table:

Hex Range	Device
000–01F	DMA controller 1, 8237A-5
020–03F	Interrupt controller 1, 8259A, Master
040–05F	Timer, 8254-2
060–06F	8042 (keyboard)
070–07F	Real-time clock, NMI mask
080–09F	DMA page register, 74LS612
0A0–0BF	Interrupt controller 2, 8237A-5
0C0–0DF	DMA controller 2, 8237A-5
0F0	Clear math coprocessor busy
0F1	Reset math coprocessor
0F8–0FF	Math coprocessor
1F0–1F8	Fixed disk
200–207	Game I/O
20C–20D	Reserved
21F	Reserved
278–27F	Parallel printer port 2
2B0–2DF	Alternate enhanced graphics adapter
2E1	GPIB (adapter 0)
2E2 & 2E3	Data acquisition (adapter 0)
2F8–2FF	Serial port 2
300–31F	Prototype card
360–363	PC network (low address)
364–367	Reserved
368–36B	PC network (high address)
36C–36F	Reserved
378–37F	Parallel printer port 1
380–38F	SDLC, bisynchronous 2
390–393	Cluster
3A0–3AF	Bisynchronous 1
3B0–3BF	Monochrome display and printer adapter
3C0–3CF	Enhanced graphics adapter
3D0–3DF	Color/graphics monitor adapter
3F0–3F7	Disk controller
3F8–3FF	Serial port 1
6E2 & 6E3	Data acquisition (adapter 1)
790–793	Cluster (adapter 1)
AE2 & AE3	Data acquisition (adapter 2)
B90–B93	Cluster (adapter 2)
EE2 & EE3	Data acquisition (adapter 3)
1390–1393	Cluster (adapter 3)
22E1	GPIB (adapter 1)
2390–2393	Cluster (adapter 4)
42E1	GPIB (adapter 2)
62E1	GPIB (adapter 3)
82E1	GPIB (adapter 4)
A2E1	GPIB (adapter 5)
C2E1	GPIB (adapter 6)
E2E1	GPIB (adapter 7)

Absolute vs. Linear Select Address Decoding:

- In decoding addresses, either all the address lines or a selected number of them are decoded –
  - If all the address lines are decoded, it is called *absolute decoding*
  - If only selected address pins are used for decoding, it is called *linear select decoding* –
    - This is cheaper due to the less number of input and the fewer the gates needed for decoding
    - The disadvantage is that it creates what are called *aliases*, the same port with multiple addresses
    - Hence, port address documentation is necessary

Portable Addresses 300 – 31FH in x86 PC:

- In the x86 PC, the address range 300H – 31FH is set aside for prototype cards to be plugged into the expansion slot
  - These prototype cards can be data acquisition boards used to monitor analog signals such as temperature, pressure, and so on
- Interface cards using the prototype address space use the following signals on the 62-pin section of the ISA expansion slot:
  - IOR and IOW. Both are active low
  - AEN signal: AEN = 0 when the CPU is using the bus
  - A0-A9 for address decoding

Use of Simple Logic Gates as Address Decoders:

- The following Fig shows the circuit design for a 74LS373 latch connected to port address 300H of an x86 PC via an ISA expansion slot
  - Notice the use of signals A0-A9 and AEN
  - AEN is low when the x86 microprocessor is in control of the buses.

Use of 74LS138 as Decoder:

Function Table

Inputs			Outputs
Enable		Select	
G1	G2	C B A	Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7
X	H	X X X	H H H H H H H H
L	X	X X X	H H H H H H H H
H	L	L L L	L H H H H H H H
H	L	L L H	H L H H H H H H
H	L	L H L	H H L H H H H H
H	L	L H H	H H H L H H H H
H	L	H L L	H H H H L H H H
H	L	H L H	H H H H H L H H
H	L	H H L	H H H H H H L H
H	L	H H H	H H H H H H H L

Using of 74LS138 for I/O Address Decoding

IBM PC I/O Address Decoder:

G1	G2A	G2B	C B A								
AEN	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
0	0	0	0	0	0	0	0	0	0	0	00 Lowest port address
0	0	0	1	1	1	1	1	1	1	1	FF Highest port address

Port 61H & Time Delay Generation:

- To maintain compatibility with the IBM PC & run operating systems, the assignment of I/O port addresses must follow the standard
- Port 61H is used to generate a time delay which will work in any PC with any type of processor from the 286 to the Pentium
- I/O port 61H has eight bits (D0-D7)
- Bit D4 of port 61H changes its state every 15.085 µs
- Toggling of bit D4 goes on indefinitely as long as the PC is on

» The following program shows how to use port 61H to generate a delay of 1/2 second

» In this program all the bits of port 310H are toggled with a 1/2 second delay in between

```
;TOGGLING ALL BITS OF PORT 310H EVERY 0.5 SEC
            MOV   DX,310H
HERE:       MOV   AL,55H        ;toggle all bits
            OUT   DX,AL
            MOV   CX,33144      ;delay=33144x15.085 us=0.5 sec
            CALL  TDELAY
            MOV   AL,0AAH
            OUT   DX,AL
            MOV   CX,33144
            CALL  TDELAY
            JMP   HERE

;CX=COUNT OF 15.085 MICROSEC
TDELAY      PROC  NEAR
            PUSH  AX            ;save AX
W1:         IN    AL,61H
            AND   AL,00010000B
            CMP   AL,AH
            JE    W1            ;wait for 15.085 usec
            MOV   AH,AL
            LOOP  W1            ;another 15.085 usec
            POP   AX            ;restore AX
            RET
TDELAY      ENDP
```

REVIEW

1. Show the circuit connection to the PC bus for the following instructions. Use simple logic gates 74LS373 and 74LS244:

   (a) OUT 309H, AL (b) IN AL, 30CH

2. Repeat the above problem using a 74LS138 for the decoder

PROGRAMMING & INTERFACING THE 8255

- The 8255 is –
  - A widely used 40-pin DIP I/O chip
  - Having three separately accessible ports, A, B, and C, which can be programmed to be input or output port, hence the name PPI (*programmable peripheral interface*)
  - They can also be changed dynamically, in contrast to the 74LS244 and 74LS373, which are hard-wired

» Port A (PA0-PA7)

Port B (PB0-PB7)

Port C (PC0-PC7) – can be programmed all as input or all as output

» Port C can also be split into two parts; CU (upper bits PC4-PC7) and CL (lower bits PC0-PC3)

» Any bit of Port C can be programmed individually

CS	A1	A0	Selects
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	x	x	8255 is not selected

Mode Selection of 8255A:

(a) Find the control word if PA = out, PB = in, PC0–PC3 = in, and PC4–PC7 = out.
(b) Program the 8255 to get data from port A and send it to port B. In addition, data from PCL is sent out to the PCU.
Use port addresses of 300H–303H for the 8255 chip.

Solution:

(a) From Figure 11-12 we get the control word of 1000 0011 in binary or 83H.

(b) The code is as follows:

```
B8255C EQU  300H   ;Base address of 8255 chip
CNTL   EQU  83H    ;PA=out,PB=in,PCL=in,PCU=out
MOV    DX,B8255C+3;load control reg. address
                   ;(300H + 3 = 303H)
MOV    AL,CNTL     ;load control byte
OUT    DX,AL       ;send it to control register
MOV    DX,B8255C+1 ;load PB address
IN     AL,DX       ;get the data from PB
MOV    DX,B8255C   ;load PA address
OUT    DX,AL       ;send it to PA
MOV    DX,B8255C+2 ;load PC address
IN     AL,DX       ;get the bits from PCL
AND    AL,0FH      ;mask the upper bits
ROL    AL,1
ROL    AL,1        ;shift the bits
ROL    AL,1        ;to upper position
ROL    AL,1
OUT    DX,AL       ;send it to PCU
```

The 8255 shown in Figure 11-13 is configured as follows: port A as input, B as output, and all the bits of port C as output.
(a) Find the port addresses assigned to A, B, C, and the control register.
(b) Find the control byte (word) for this configuration.
(c) Program the ports to input data from port A and send it to both ports B and C.

Solution:
(a) The port addresses are as follows:

$\overline{CS}$	A1	A0	Address	Port
11 0001 00	0	0	310H	Port A
11 0001 00	0	1	311H	Port B
11 0001 00	1	0	312H	Port C
11 0001 00	1	1	313H	Control register

(b) The control word is 90H, or 1001 0000.
(c) One version of the program is as follows:

```
MOV   AL,90H      ;control byte PA=in, PB=out, PC=out
MOV   DX,313H     ;load control reg address
OUT   DX,AL       ;send it to control register
MOV   DX,310H     ;load PA address
IN    AL,DX       ;get the data from PA
MOV   DX,311H     ;load PB address
OUT   DX,AL       ;send it to PB
MOV   DX,312H     ;load PC address
OUT   DX,AL       ;and to PC
```

Using the EQU directive one can rewrite the above program as follows:

```
CNTLBYTE   EQU   90H    ;PA=in, PB=out, PC=out
PORTA      EQU   310H
PORTB      EQU   311H
PORTC      EQU   312H
CNTLREG    EQU   313H
           ....
           ...
           MOV   AL,CNTLBYTE
           MOV   DX,CNTLREG
           OUT   DX,AL
           MOV   DX,PORTA
           IN    AL,DX
           ;and so on.
```

Show the address decoding where port A of the 8255 has an I/O address of 300H, then write a program to toggle all bits of PA continuously with a 1/4 second delay. Use INT 16H to exit if there is a keypress.

Solution:
The address decoding for the 8255 is shown in Figure 11-14. The control word for all ports as output is 80H. The program below will toggle all bits of PA indefinitely with a delay in between. To prevent locking up the system, we press any key to exit to DOS.

```
            MOV   DX,303H       ;CONTROL REG ADDRESS
            MOV   AL,80H        ;ALL PORTS AS OUTPUT
            OUT   DX,AL
AGAIN:      MOV   DX,300H
            MOV   AL,55H
            OUT   DX,AL
            CALL  QSDELAY       ;1/4 SEC DELAY
            MOV   AL,0AAH       ;TOGGLE BIT
            OUT   DX,AL
            CALL  QSDELAY
            MOV   AH,01
            INT   16H           ;CHECK KEYPRESS
            JZ    AGAIN         ;PRESS ANY KEY TO EXIT
            MOV   AH,4CH
            INT   21H           ;EXIT

QSDELAY     PROC  NEAR
            MOV   CX,16572      ;16,572x15.085 usec=1/4 sec
            PUSH  AX
W1:         IN    AL,61H
            AND   AL,00010000B
            CMP   AL,AH
            JE    W1
            MOV   AH,AL
            LOOP  W1
            POP   AX
            RET
QSDELAY     ENDP
```

Notice the use of INT 16H option AH = 01 where the keypress is checked. If there is no keypress, it will continue. We must do that to avoid locking up the x86 PC.

Buffering 300 – 31FH Address Range:

- When accessing the system bus via the expansion slot; we must make sure that the plug-in card does not interfere with the working of system buses on the motherboard
- To do that we isolate (buffer) a range of I/O addresses using the 74LS245 chip
- In buffering, the data bus is accessed only for a specific address range, and access by any address beyond the range is blocked
- The following Fig shows how the I/O address range 300H-31FH is buffered with the use of the 74LS245

- The following Fig shows another example of 8255 interfacing using the 74LS138 decoder.
- As shown in the Fig., Y0 and Y1 are used for the 8255 and 8253, respectively
- The Table shows the 74LS 138 address assignment

Selector	Address	Assignment
Y0	300–303	Used by 8255
Y1	304–307	Used by 8253
Y2	308–30B	Available
Y3	30C–30F	Available
Y4	310–313	Available

- The following Fig shows the circuit for buffering all the buses
- The 74LS244 is used to boost the address and control signals

- The following shows a test program to toggle the PA and PB bits
- Notice that in order to avoid locking up the system, INT 16H is used to exit upon pressing any key

Write a program to toggle all bits of PA and PB of the 8255 chip on the PC Trainer. Put a 1/2 second delay in between "on" and "off" states. Use INT 16H to exit if there is a keypress.

Solution:
The program below toggles all bits of PA and PB indefinitely. Pressing any key exits the program.

```
            MOV     DX,303H     ;CONTROL REG ADDRESS
            MOV     AL,80H          ;ALL PORTS AS OUTPUT
            OUT     DX,AL
AGAIN:          MOV   DX,300H     ;PA ADDRESS
            MOV     AL,55H
            OUT     DX,AL
            INC     DX          ;PB ADDRESS
            OUT     DX,AL
            CALL    HSDELAY     ;1/2 SEC DELAY
            MOV     DX,300H     ;PA ADDRESS
            MOV     AL,0AAH
            OUT     DX,AL
            INC     DX          ;PB ADDRESS
            OUT     DX,AL
            CALL    HSDELAY     ;1/2 SEC DELAY
            MOV     AH,01
            INT     16H         ;CHECK KEYPRESS
            JZ      AGAIN           ;PRESS ANY KEY TO EXIT
            MOV     AH,4CH      ;
            INT     21H         ;EXIT

HSDELAY     PROC    NEAR
            MOV     CX,33144    ;33144x15.085 usec=1/2 sec
                    PUSH  AX
W1:         IN      AL,61H
            AND     AL,00010000B
            CMP     AL,AH
            JE      W1
            MOV     AH,AL
            LOOP    W1
            POP     AX
            RET
HSDELAY     ENDP
```

Notice the use of INT 16H option AH = 01 where the keypress is checked. If there is no keypress, it will continue.

Visual C/C++ I/O Programming:

- Visual C++ is an object-oriented language – it comes with many classes and objects to make programming easier and more efficient
- But, there is no object or class for directly accessing I/O ports in the full Windows version of Visual C++
  - The reason is that Microsoft wants to make sure the x86 system programming is under full control of the operating system
  - This prevents any hacking into the system hardware
  - This applies to Windows NT, 2000, XP, and higher
- Hence, none of the system INT instructions such as INT 21H and I/O operations are applicable in Windows XP and its subsequent versions
- To access the I/O and other hardware features of the x86 PC in the XP environment you must use the Windows Platform SDK provided by Microsoft

- The situation is different in the Windows 9x (95 and 98) environment
- While INT 21H and other system interrupt instructions are blocked in Windows 9x, direct I/O addressing is available
- To access I/O directly in Windows 9x, you must program Visual C++ in console mode
- The instruction syntax for I/O operations is shown in the following Table
- Notice the use of the underscore character in both the _outp and _inp instructions
- The x86 Assembly language makes a distinction between the 8-bit and 16-bit I/O addresses by using the DX register

x86 Assembly	**Visual C++**
OUT port#, AL	_outp (port#, byte)
OUT DX, AL	_outp (port#, byte)
IN AL, port#	_inp (port#)
IN AL, DX	_inp (port#)

- There is no such distinction in C programming.
- For the instruction "*outp (port#, byte)*" the port# can take any address value between 0000 and FFFFH

Write a Visual C++ program for Windows 98 to toggle all bits of PA and PB of the 8255 chip. Use the kbhit function to exit if there is a keypress.

Solution:

```
//Tested by Dan Bent
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
#include<iomanip.h>
#include<windows.h>
void main()
  {
  cout<<setiosflags(ios::unitbuf);  // clear screen buffer
  cout<<"This program toggles the bits for Port A and Port B.";
  _outp(0x303,0x80);                //MAKE PA,PB of 8255 ALL OUTPUT
  do
    {
    _outp(0x300,0x55);              //SEND 55H TO PORT A
    _outp(0x301,0x55);              //SEND 55H TO PORT B
    _sleep(500);                    //DELAY of 500 msec.
    _outp(0x300,0xAA);              //NOW SEND AAH TO PA, and PB
    _outp(0x301,0xAA);
    _sleep(500);
    }
    while(!kbhit());
  }
```

Write a Visual C++ program for Windows 98 to get a byte of data from PA and send it to both PB and PC of the 8255 chip in PC Trainer.

Solution:

```
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
#include<iomanip.h>
#include<windows.h>
#include<process.h>
//Tested by Dan Bent
void main()
  {
      unsigned char mybyte;
       cout<<setiosflags(ios::unitbuf);// clear screen buffer
      system("CLS");
      _outp(0x303,0x90);     //PA=in, PB=out, PC=out
      _sleep(5);             //wait 5 milliseconds
      mybyte=_inp(0x300);    //get byte from PA
      _outp(0x301,mybyte);   //send to PB
      _sleep(5);
      _outp(0x302,mybyte);   //send to Port C
      _sleep(5);
      cout<<mybyte;          //send to PC screen also
      cout<<"\n\n";
  }
```

I/O Programming in Linux C/C++:

- Linux is a popular operating system for the x86 PC
- The following Table provides the C/C++ syntax for I/O programming in the Linux OS environment.

x86 Assembly	Linux C/C++
OUT port#, AL	outb (byte, port#)
OUT DX, AL	outb (byte, port#)
IN AL, port#	inb (port#)
IN AL, DX	inb (port#)

Compiling & Running Linux C/C++ Programs with I/O Functions:

- To compile the I/O programs, the following points must be noted
  - To compile with a keypress loop, you must link to library ncurses as follows:

```
> gcc -lncurses toggle.c -o toggle
```

- To run the program, you must either be root, or root must change permissions on executable for hardware port access

  Example: (as root or superuser)

```
> chown root toggle
> chmod 4750 toggle
```

- Now toggle can be executed by users other than root.

Write a C/C++ program for a PC with the Linux OS to toggle all bits of PA and PB of the 8255 chip on the PC Trainer. Put a 500 ms delay between the "on" and "off" states. Pressing any key should exit the program.

Solution:

```
//    This program demonstrates low level I/O
//    using C language on
//    Tested by Nathan Noe
#include <stdio.h>     // f
#include <unistd.h>    // f
#include <sys/io.h>    // f
#include <ncurses.h>   // f
```

```
int main ()
  {
  int n=0;                  // temp char variable
  int delay=5 e5;           // sleep delay variable

  ioperm(0x300,4,0x300);   // get port permission
  outb(0x80,0x303);         // send control word

  //----- begin ncurses setup ----------
  //--- (needed for console i/o) -------

  initscr();                // initialize screen for ncurses
  cbreak();                 // do not wait for carriage return
  noecho();                 // do not echo input character
  halfdelay(1);             // only wait for 1ms for input
                            // from keyboard
  //-----  end ncurses setup  ----------

  do                        // main toggle loop
    {
    printf("0x55 \n\r");    // display status to screen
    refresh();              // refresh() to update console
    outb(0x55,0x300);       // send 0x55 to PortA (01010101B)
    outb(0x55,0x301);       // send 0x55 to PortB (01010101B)
    usleep(delay);          // wait for 500ms (5 e5 microseconds)
    printf("0xAA \n\r");    // display status to screen
    refresh();              // refresh() to update console
    outb(0xaa,0x300);       // send 0xAA to PortA (10101010B)
    outb(0xaa,0x301);       // send 0xAA to PortB (10101010B)
    usleep(delay);          // wait for 500ms
                            // get input from keyboard
    n=getch();              // if no keypress in 1ms, n=0
                            // due to halfdelay()
   }
while(n<=0);                // test for keypress
                            // if keypress, exit program
  endwin();                 // close program console for ncurses
  return 0;                 // exit program
  }
```

Write a C/C++ program for a PC with the Linux OS to get a byte of data from port A and send it to both port B and port C of the 8255 in the PC Trainer.

Solution:

```
//    This program gets data from Port A and
//    sends a copy to both Port B and Port C.
//    Tested by: Nathan Noel -- 2/10/2002
//-----------------------------------------------
#include <stdio.h>
#include <unistd.h>
#include <sys/io.h>
#include <ncurses.h>

int main ()
  {
  int n=0;              // temp variable
  int i=0;              // temp variable

  ioperm(0x300,4,0x300);// get permission to use ports
  outb(0x90,0x303);     // send control word for
                  // PortA=input, PortB=output, PortC=output

  initscr();            // initialize screen for ncurses
  cbreak();             // do not wait for carriage return
  noecho();             // do not echo input character
  halfdelay(1);         // only wait for 1ms for input

  do                    // main toggle loop
    {
    i=inb(0x300);       // get data from PortA
    usleep(1e5);        // sleep for 100ms

    outb(i,0x301);      // send data to PortB
    outb(i,0x302);      // send data to PortC

    n=getch();          // get input from keyboard
                        // if no keypress in 1ms, n=0
    }while(n<=0);       // test for keypress
                        // if keypress, exit program

  endwin();             // close program window
  return (0);           // exit program
  }
```

REVIEW

1. Find addresses for all 8255 ports if A7-A2 = 111101 is used to activate CS.
2. Find the control word for an 8255 in mode 0 (simple I/O) if all the ports are configured as output ports.
3. Find the control word for an 8255 in mode 0 (simple I?O) if all the ports are configured as input ports.
4. Program an 8255 with the following specifications: All ports are output ports. Write 55H to the ports. After a delay, switch them all to AAH.
5. How are ports configured after the control register is loaded with 89H?

REVIEW

6. Show the decoding circuitry for the 8255 if we want port A to have address 68H. Use NAND and inverter gates.
7. If 91H is the control word, indicate which port is input and which is output.
8. Find the control word if PA is input, PB is input, and PC0-PC7 is output.
9. Write a program to monitor PA for a temperature of 100. if it is equal, it should be saved in register BL. Also, send AAH to port B and 55 to port C.

REVIEW

10. Write a program to get a byte of data from PA, convert it to ASCII bytes, and store them in registers CL, AH, and AL. For example, an input of FFH will show as 255 (Note: FF in binary becomes 323535 in ASCII)

15CS – 44

MICROPROCESSORS AND MICROCONTROLLERS

MODULE 3 – QUIZ 3

8255 I/O PROGRAMMING

Mahesh Prasanna K.

Dept. of CSE, VCET.

1. While memory contains both code and data, ports contain data only (TRUE/FALSE)
2. In the instruction "*OUT 99H, AL*", the port address is _____ bits
3. In the instruction "*OUT DX, AL*", the port address is _____ bits
4. In the instruction "*IN AL, 78H*", register AL is the destination (FTRUE/FALSE)
5. To access the maximum number of ports in the x86, we must decode address A0-A15 (TRUE/FALSE)

6. Designers use a _____ (latch, tri-state buffer) for output and a _____ for input (latch, tri-state buffer)
7. Designers use a _____ (IOW, IOR) for output and a _____ for input (IOW, IOR)
8. Segment register DS is used to generate a port's physical address (TRUE/FALSE)
9. In "OUT 65H, AL", only address pins A0-A7 are used by the 8088 to provide the address (TRUE/FALSE)
10. An input port is distinguished from an output port by the port address assigned to it (TRUE/FALSE)

11. An input port is distinguished from an output port by the IOR and IOW control signals (TRUE/FALSE)
12. Memory-mapped I/O uses control signals MEMR and MEMW (TRUE/FALSE)
13. In memory-mapped I/O, one can perform logical and arithmetic operations on the data without moving it into the accumulator first (TRUE/FALSE)
14. The I/O address range set aside for prototype cards is _____
15. In x86 PC, the _____ bit of port address ____ toggles every _____ microseconds

16. All three ports of 8255, A, B, and C can be programmed for simple I/O (TRUE/FALSE)

17. In simple I/O programming of port A of the 8255, we can use PA0-PA3 for output and PA4-PA7 for an input port. (TRUE/FALSE)

1. While memory contains both code and data, ports contain data only (TRUE/FALSE)
2. In the instruction "*OUT 99H, AL*", the port address is _____ bits (8)
3. In the instruction "*OUT DX, AL*", the port address is _____ bits (16)
4. In the instruction "*IN AL, 78H*", register AL is the destination (TRUE/FALSE)
5. To access the maximum number of ports in the x86, we must decode address A0-A15 (TRUE/FALSE)

6. Designers use a _____ (latch, tri-state buffer) for output and a _____ for input (latch, tri-state buffer)
7. Designers use a _____ (IOW, IOR) for output and a _____ for input (IOW, IOR)
8. Segment register DS is used to generate a port's physical address (TRUE/FALSE)
9. In "OUT 65H, AL", only address pins A0-A7 are used by the 8088 to provide the address (TRUE/FALSE)
10. An input port is distinguished from an output port by the port address assigned to it (TRUE/FALSE)

11. An input port is distinguished from an output port by the IOR and IOW control signals (TRUE/FALSE)
12. Memory-mapped I/O uses control signals MEMR and MEMW (TRUE/FALSE)
13. In memory-mapped I/O, one can perform logical and arithmetic operations on the data without moving it into the accumulator first (TRUE/FALSE)
14. The I/O address range set aside for prototype cards is _____ (300 – 31FH)
15. In x86 PC, the _____ bit of port address ____ toggles every _____ microseconds (D4, 61H, 15.085)

16. All three ports of 8255, A, B, and C can be programmed for simple I/O (TRUE/FALSE)

17. In simple I/O programming of port A of the 8255, we can use PA0-PA3 for output and PA4-PA7 for an input port. (TRUE/FALSE)