Syllabus: Microprocessors And Microcontrollers - 15CS44:

Module 3: 8255 I/O programming: I/O addresses MAP of x86 PC's, programming and interfacing the 8255.
Text book: Muhammad Ali **Mazidi**, Janice GillispieMazidi, Danny Causey, The x86 PC Assembly Language Design and Interfacing, 5th Edition, Pearson, 2013**. Ch 11: 11.1 to 11.4**

8086 I/O Interfacing & Programming - Part I

- I/O ports or input/output ports are the devices through which the microprocessor communicates with other devices or external data sources/ destinations.
- Input activity, enables the microprocessor to read data from external devices, viz., keyboard, joysticks, mouse etc. These devices are known as input devices as they feed data into a µP system.
- Output activity transfers data from the µP to the external devices, say CRT display, 7-segment displays, printer, etc. These devices that accept the data from a µP system are called output devices.

- All x86 processors from 8086 to Pentium can access external I/O devices (called *ports – 8/16 bit wide*) through I/O instructions – **IN *dst, source* & OUT *dst,src***
- Access data through AL (8bit) or AX (16 bit)
- I/O space in addition to memory space -- $IO/\overline{M}$
- Memory can contain both data & opcodes (program)
- I/O ports contain only data

Direct Addressing Mode	**IN AL, 8BIT_PORT#** **OUT 8BIT_PORT#, AL**	**IN AX, 8BIT_PORT#** **OUT 8BIT_PORT#, AX**
Indirect Addressing Mode	MOV DX, 16BIT_PORT# IN AL, DX OUT DX, AL	MOV DX, 16BIT_PORT# IN AX, DX OUT DX, AX

Example: Write an ALP to toggle the bits of port address 300H continuously.

```
        MOV DX, 300H
again:  MOV AL, 55H
        OUT DX, AL
        MOV AL, 0AAH
        OUT DX, AL
        JMP BACK
```

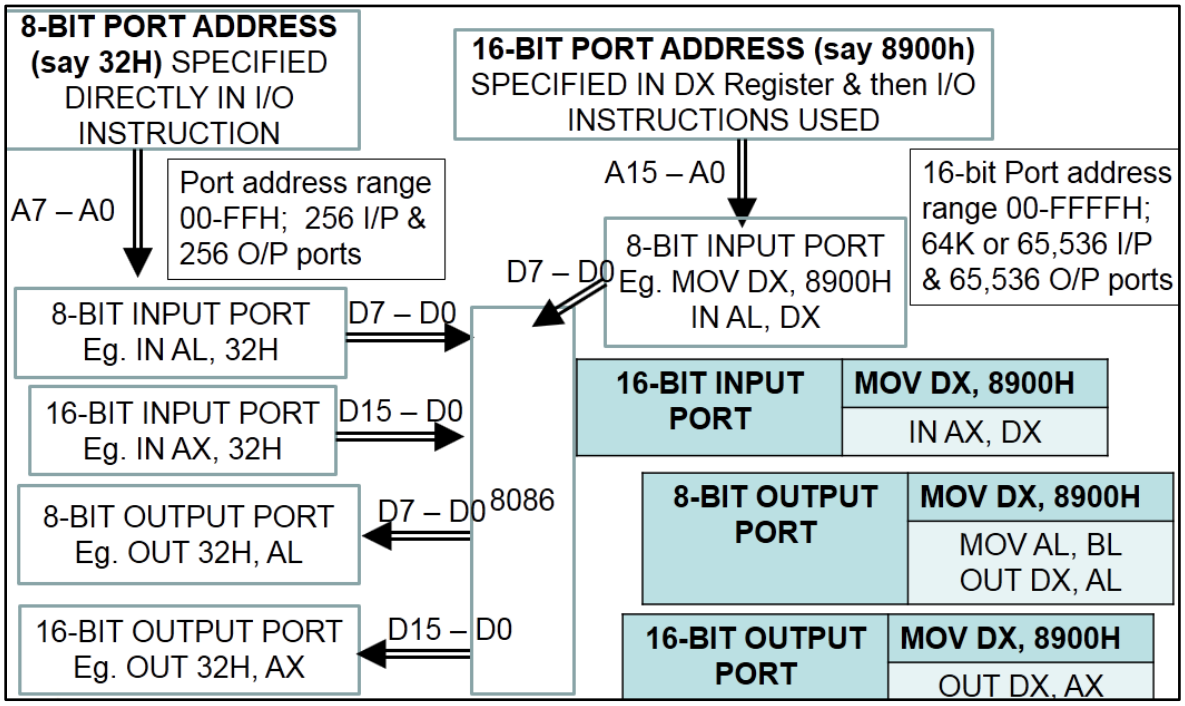
The patterns 55 => 0101 0101 and AA=> 1010 1010 toggle the pins.

Example : Write an ALP to bring in 8-bit data from port address 302H; add contents of BL to it and send the 16-bit sum to port address 45H.

```
        MOV DX, 302H   ;16-bit port address – use DX
        IN AL, DX      ;8-bit data into AL
        MOV AH,00      ;initialize AH to hold carry
        ADD AL, BL
        JNC SKIP       ;skip if no carry
        INC AH
SKIP:   OUT 45H, AX
```

; The last OUT instruction makes 16-bit sum (data) in AX –AHAL to be sent to port with 8-bit address – 45H using direct addressing (no DX)

Overview of 8088 Input/Output Instructions & how to use the I/O Instructions are shown in Fig. 1 below.



Summary :

- In x86 system with 8-bit address bus A7-A0 for port addresses; the maximum number of input ports = 256 (00 - FF) & the maximum number of output ports = 256
- x86 can have maximum of 2^{16} = 64K = 65,536 I/O ports
- Instruction OUT 24H, AL – sends the data in AL register to an output port at 24H I/O address
- ALP to accept data from port 300H & send it out to port 304H

```
MOV DX, 300H
IN AL, DX
MOV DX, 304H
```

OUT DX, AL

- Assembly language instructions to place status of port 60H in CH:

IN AL, 60H ; MOV CH, AL

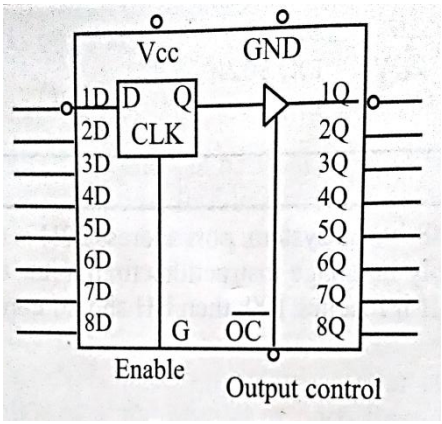
I/O ADDRESS DECODING

- The control signals IOR & IOW are used along with the decoder
- A0-A7 decoded for 8-bit address
- A0-A15 decoded for 16- bit address

Output port design using 74LS373

- Data sent out by the CPU via the data bus must be latched (memory devices have an internal latch)
- Latching system using 74LS373 designed for output devices
- OC pin must be grounded ;
- AND the address decoder output and IOW control signal to enable latching action.

Table: 74LS373 D LATCHFor output port



Function Table

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

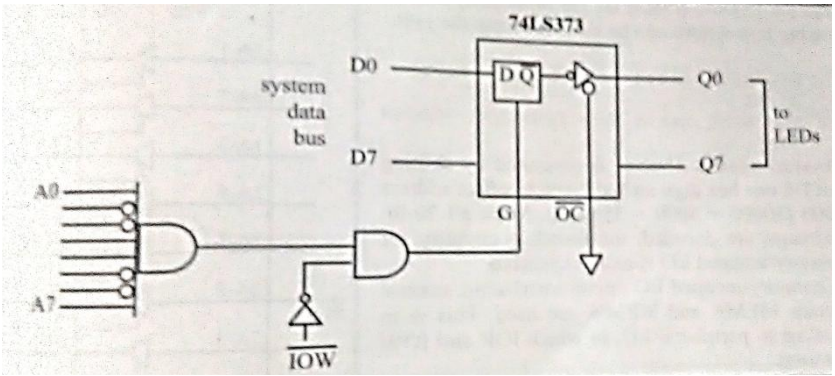


Fig: Design for 'OUT 99H, AL' with A7 – A0 : 1001 1001; IOW =0 for G

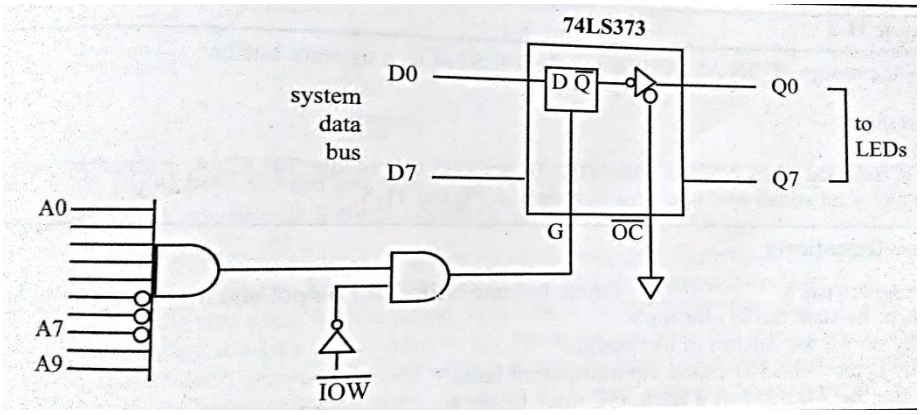


Fig: Design for output port address of 31F H

Input port design using 74LS244

- Data coming into the CPU via the data bus must come in through a THREE – STATE BUFFER (memory devices have an internal tri-state buffer)
- 74LS244 is designed for input devices for **buffer**ing as well as providing **high driving capability** for unidirectional buses
- 1G & 2G pins each control 4-bits. For 8-bits both of them must be activated;
- AND the address decoder output and IOR control signal to enable the tri – state input.
- 74LS245 – used for bidirectional buses

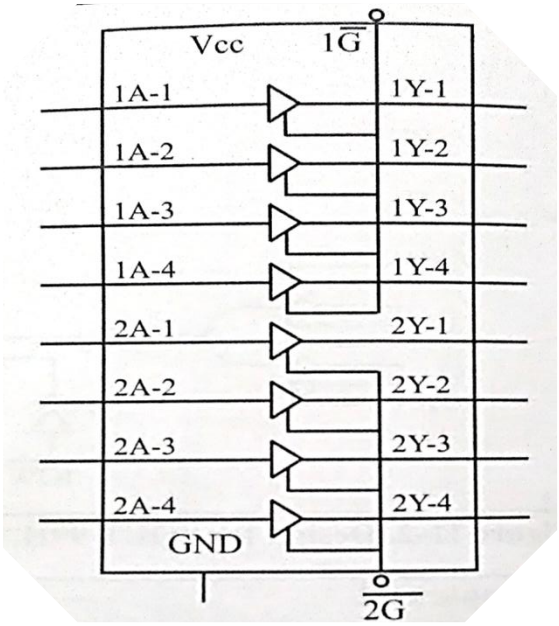


Fig: 74LS244 OCTAL BUFFER

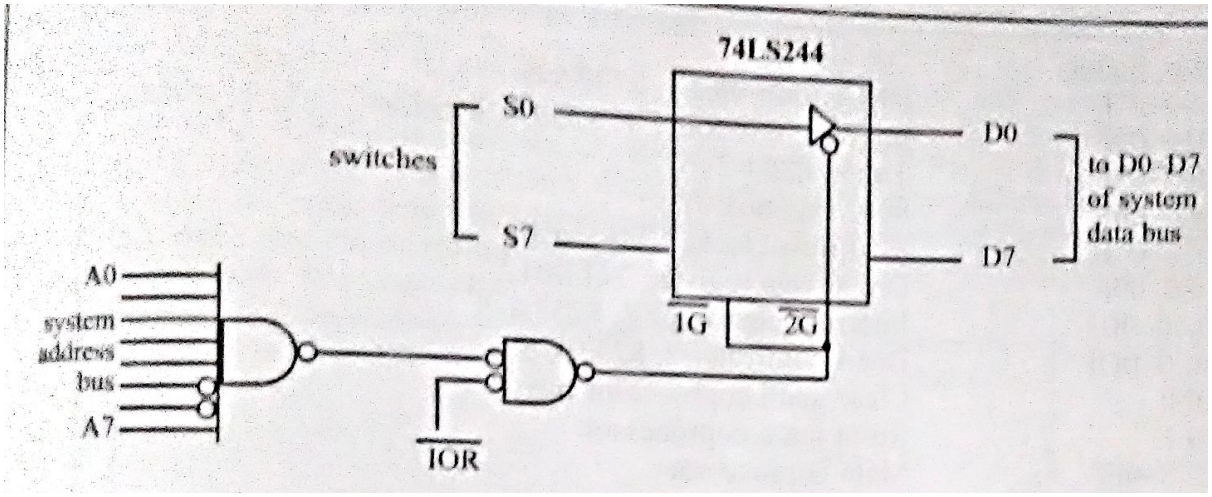


Fig: Input Port Design for 'IN AL, 9FH' 9F => A7 to A0 = 1001 1111

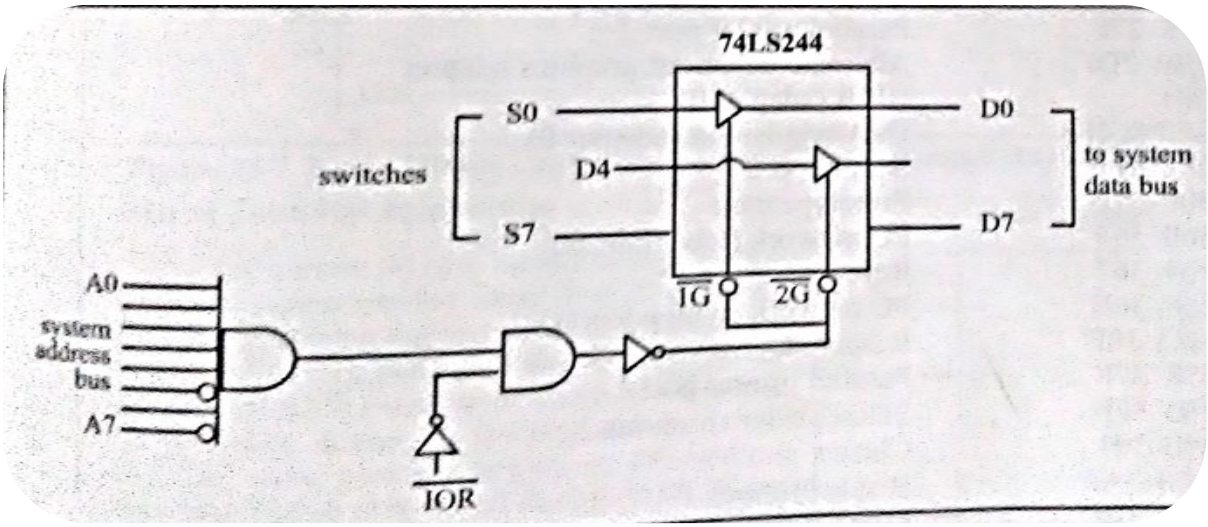


Fig: Input Port Design for 'IN AL, 5FH' => A7 to A0 at input to NAND gate is 5F – 0101 1111

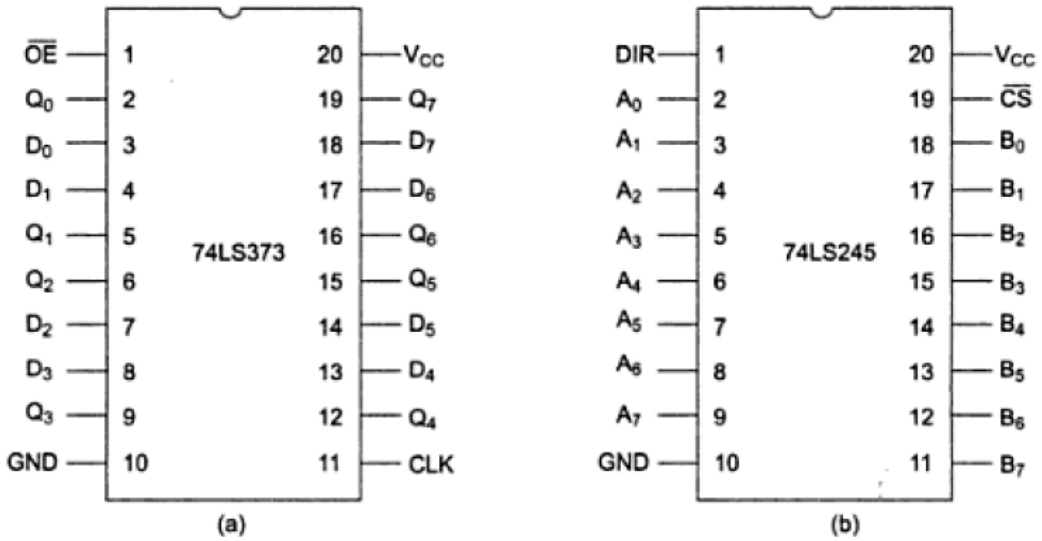


Fig. a) LATCH – (O/P PORT) b) BUFFER – (I/P PORT)

Memory Mapped I/O

- Communicating with I/O devices using IN & OUT instructions is known as *Peripheral I/O; Isolated I/O; I/O Mapped I/O*
- In RISC processors; no IN & OUT instructions. – use memory mapped I/O : wherein a memory location is assigned to be an input or output port
- Differences between memory mapped I/O & Peripheral I/O are shown in Table below.

<table>
<tr><th colspan="2">Memory Mapped I/O</th><th>Peripheral I/O</th></tr>
<tr><td colspan="2">Use instructions that access memory locations:
MOV AL,[2000] – for i/p port
MOV [2010], AL – for accessing o/p port with memory address 2010</td><td>Use IN & OUT instructions
IN AL, 30H
OUT DX, AL</td></tr>
<tr><td colspan="2">Entire 20-bit address A19 –A0 is to be decoded. Requires DS to be loaded before</td><td>Decode only 16/8 bit – A15- A0</td></tr>
<tr><td colspan="2">Say if physical address is 35000H for i/p port</td><td rowspan="2">MOV DX, 3500H
IN AL,DX</td></tr>
<tr><td>MOV AX, 3000H
MOV DS,AX
MOV AL, [5000H]</td><td>Decoding circuitry is expensive as 20 bit addresses are decoded</td></tr>
<tr><td colspan="2">MEMR , MEMW – Control signals used</td><td>IOR, IOW</td></tr>
<tr><td colspan="2">Number of ports can go upto as high as 2^{20} = 1M ports</td><td>Limited to 2^{16} – 65,536 input & 65,536 output ports</td></tr>
<tr><td colspan="2">Disadv: eats into/ uses memory space; leads to memory space fragmentation</td><td></td></tr>
</table>

I/O Address Map of x86 PCs

- Assignment of different port addresses to various peripherals such as LPT, COM port, etc
- List of designated I/O port addresses is the I/O MAP

HEX RANGE	DEVICE
000 – 01F	DMA Controller I, 8237A -5
000 – 03F	Interrupt Controller I, 8259A, Master
040 – 05F	Timer, 8254 – 2
060 – 06F	8042 (Keyboard)
378 – 37F	Parallel Printer Port 1
3F8 – 3FF	Serial Port 1
1390 – 1393	Cluster (adapter 3)
E2E1	GPIB (adapter 7)

Absolute Decoding

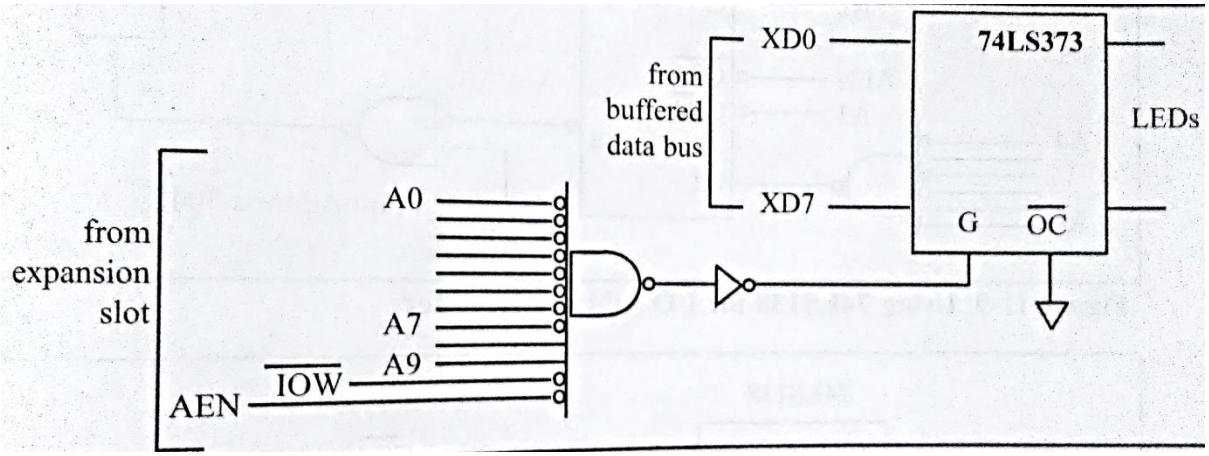
- All the address lines are decoded

Linear Select Address Decoding

- Only selected address lines are decoded
- Is cheaper, less input; fewer decoding gates used
- Disadv : Aliases – same port with multiple addresses. Document the port addresses in the I/O map thoroughly. Large gap in the I/O address map of the x86 PC, due to address aliases.
- Prototype addresses 300 – 31FH in x86 PC
- Set aside for prototype cards to be plugged in to the expansion slot.
- Can be data acquisition boards – used to monitor analog signals such as temperature, pressure, etc
- 62-pin section of the ISA expansion slot uses the following signals:
    - IOR & IOW – both active low
    - AEN signal = 0 when CPU is using the bus
    - A0 - A9 for address decoding

I method : Use of logic gates as address decoders

- Fig below shows latch connected to port address 300H of an x86 PC via an ISA expansion slot.
- A0- A9, IOW, AEN =0 used



II method : Use of 74LS138 as address decoders

74LS138 is a group of NAND gates in a single chip. Each Y can control a single device. Much more efficient than simple logic gates as shown under memory interfacing section.

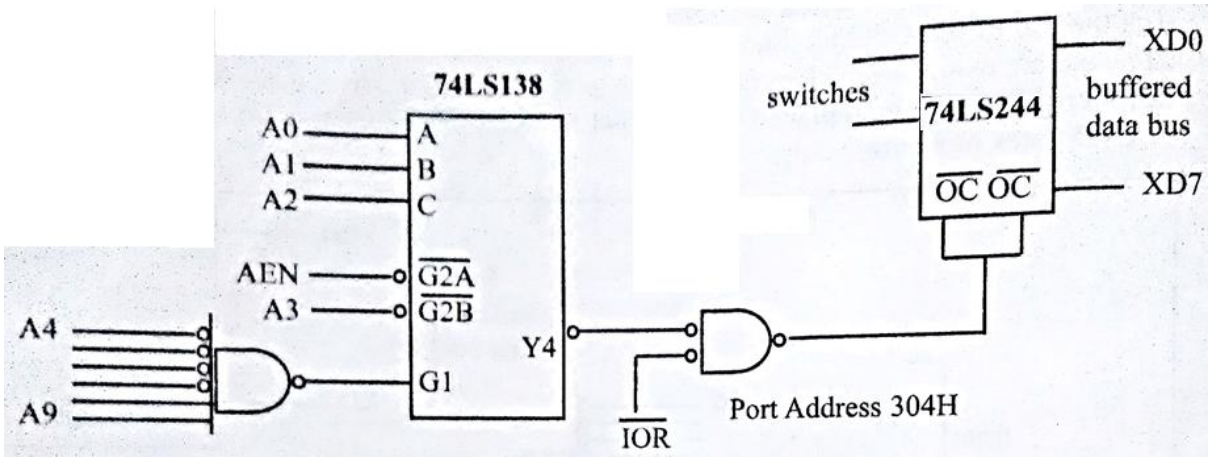


Fig: Address decoding for an **input port** located at 304H

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
NAND GATE INPUTS – HIGH TO ENABLE						**G2 =0**	**CBA = 100 for Y4**		
1	**1**	**0**	**0**	**0**	**0**	0	1	0	0
3		**0**				**4**			

8255 I/O Programming

Introduction

To have more pins for I/O, the 8086 is interfaced to 8255 - a PPI (Parallel Peripheral Interface) – pin diagram is shown in Fig.1. 8255 provides 3ports- Port A, B and C-each of 8 pins-totally 24 I/O pins. Each port can be dynamically changed as input or output in contrast to hardwired 74LS244 or 74LS373. But still the flexibility is limited. In 8255, the complete port A should be configured as either input port or output port. Similarly port B and port C can be configured. A little flexibility is available with port C, which is divided into 2 halves - upper and lower, ie., PCu and PC_L which can be individually configured as either input or output.Also using the BSR (Bit-Set and Reset Mode), individual port C pins can be set or reset irrespective of the other pins (but this is only output).

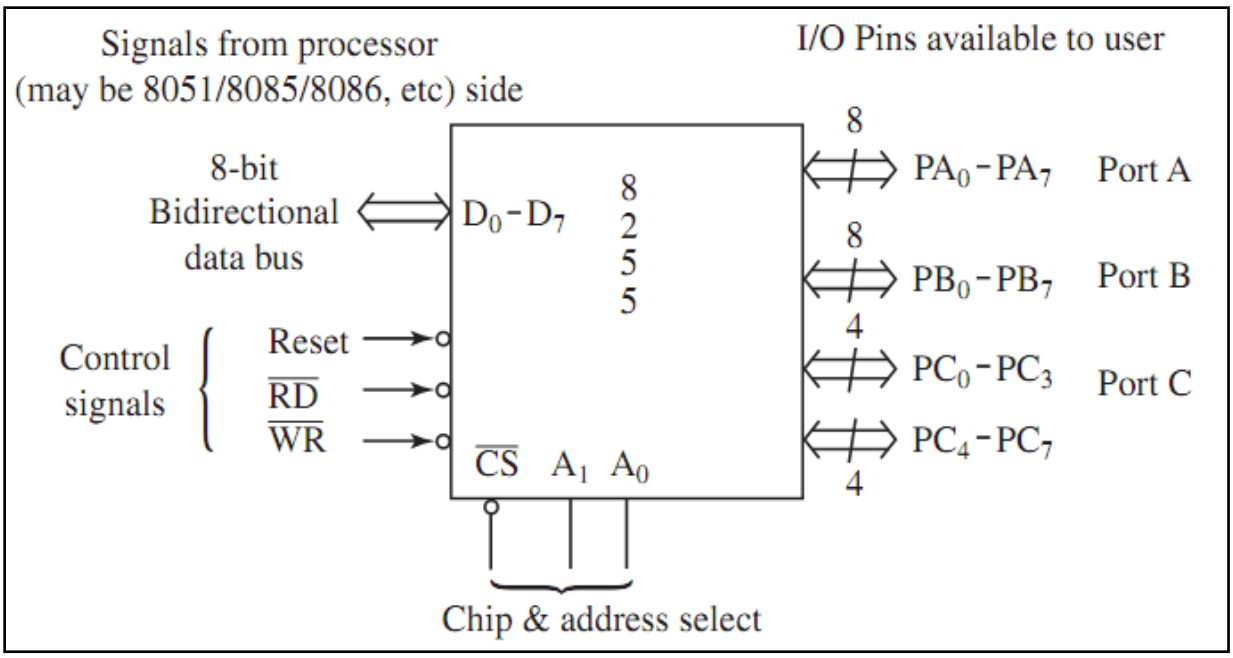


Fig.1 8255 Functional Diagram

Features of 8255

1. It is a 40-pin IC
2. It has 24-I/O pins grouped as port A, B, C
3. Port A can be programmed as either input or output port, with or without handshake signals, and also it can be programmed as bi-directional port.
4. Port B can be programmed as input or output port, and with or without handshake signals.
5. Port C is grouped in two 4 bit ports: PC4-PC7 (Port C upper) and PC3-PC0 (Port C lower)-each can be programmed as input or output port.
6. Port C lines can be individually set or reset to generate control signals for controlling external I/O devices.

Control Signals in 8255:

RD & WR: active low control input signals – connected to IOR & IOW – I/O (peripheral) mapped I/O MEMR & MEMW - memory mapped IO

RESET: active high input to 8255; Clears the control register. All ports initialized to input

A0, A1, CS: CS selects entire chip**A0, A1**- used to select specific port as shown in Fig. 2

$\overline{CS}$	A1	A0	Port
0	0	0	A
0	0	1	B
0	1	0	C
0	1	1	Control Register

Fig. 2 Selection of specific ports & CWR

Control Register in 8255

Control register is an 8 bit register, and its content is called control word.Control register controls the over all operations of the 8255A.Control register is divided into two blocks-Group A control and Group B control. Group A control, controls the port A and upper port C, and group B control, controls the port B and lower port C. Fig. 3 shows the functions of control word, and control register must be programmed to select the operations of Port A, B and C. The 8255 operates in BSR mode or I/O mode. If D7 = 0, port C operates in the bit set/reset (BSR) mode.

Fig. 4 shows the control word format in BSR mode.

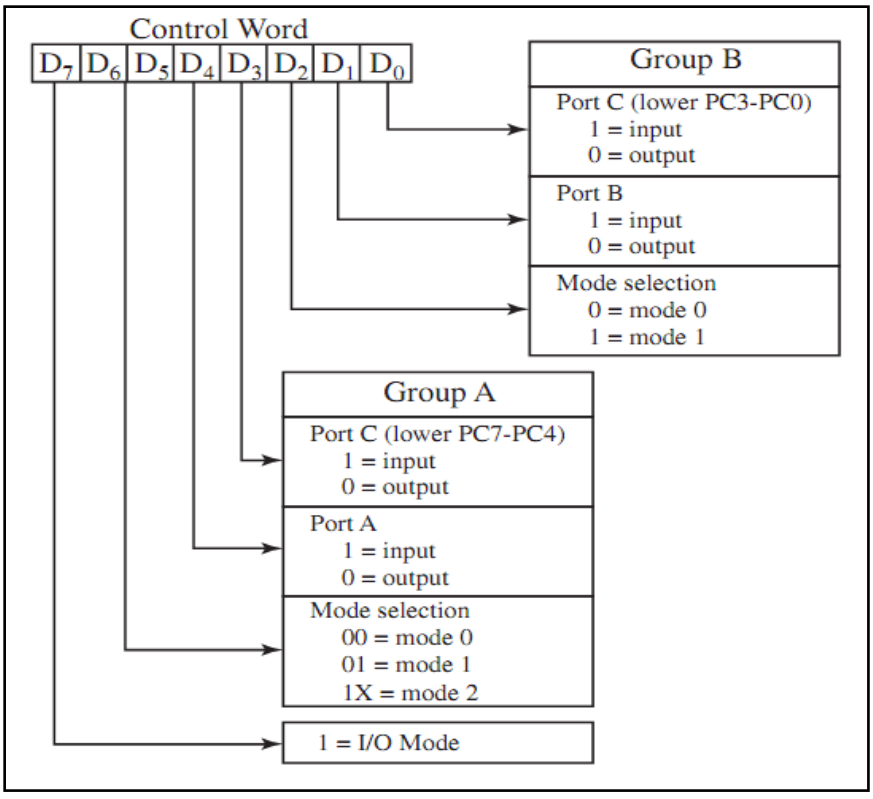


Fig. 3 Control word format of 8255A in I/O mode

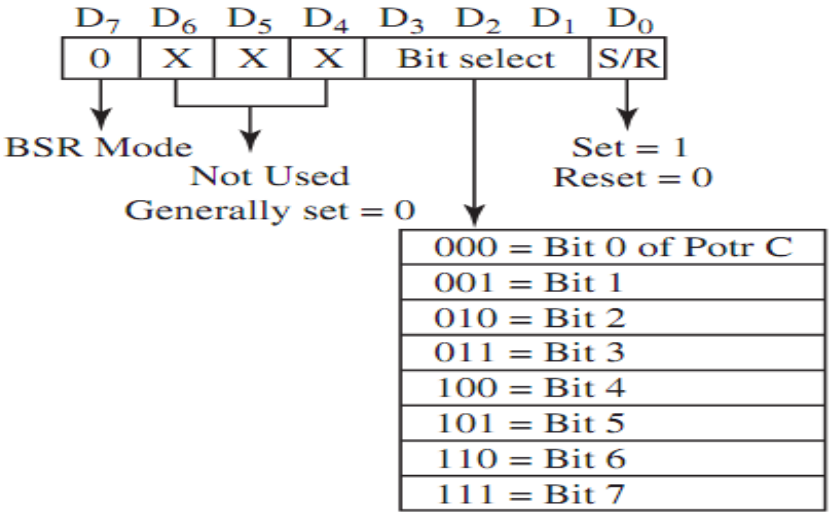


Fig. 4 Control word format of 8255A in BSR mode

Interfacing 8255 to 8086 Processor is as shown in Fig. 5 below. The interfacing circuit consists of an address decoding circuitry along with A1A0 for selecting the THREE ports and Control word Register.

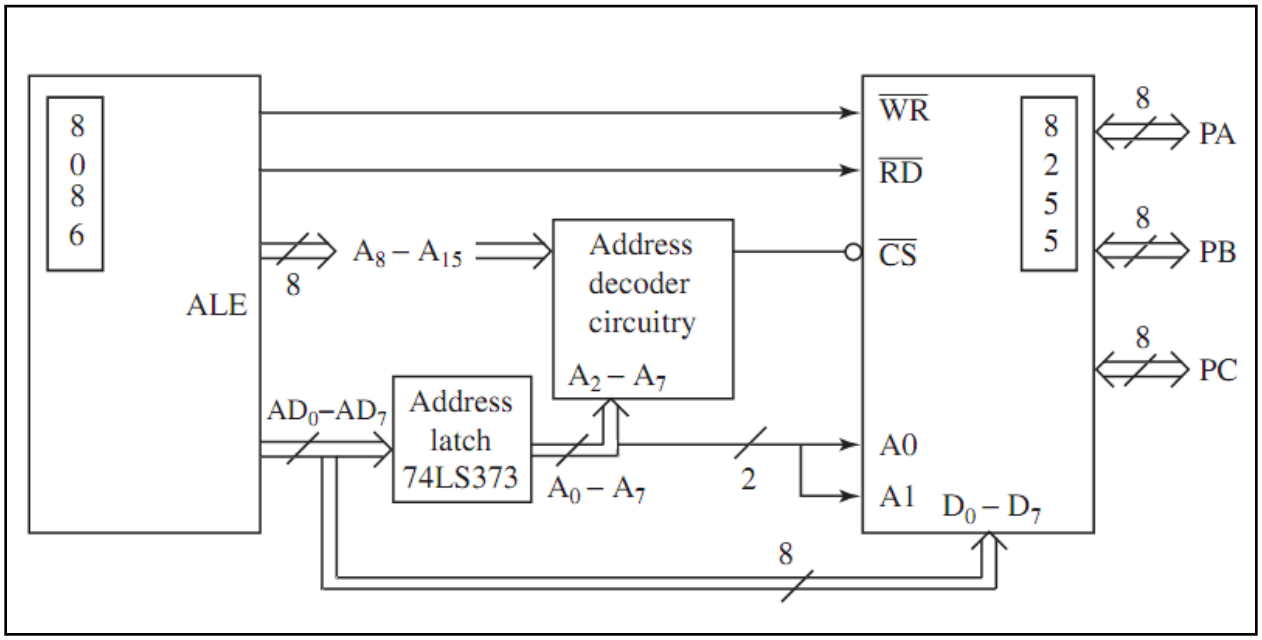


Fig. 5 8255 Interfacing to 8086

Example 1: Develop a solution to Interface 8255 with 8086 µP for a base address of B800H.

Solution: A base address of B800H implies that PORT A address is B800H, PB is at B801H; PC is at B802H and CWR is at B803H.

The address decoding circuit using NAND gates is shown in Fig. 6 & the address map is shown in Table 1

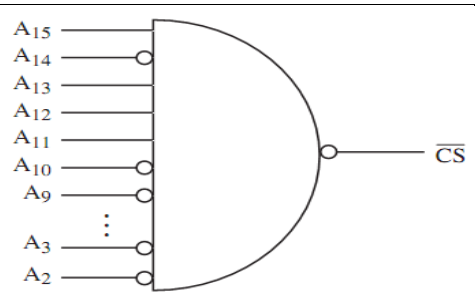


Fig. 6 Decoding Circuit for Example 1

Table 1 Address Map for Example1

Address (in Hex)	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Selected
B800H	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	Port A
B801H	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	Port B
B802H	1	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	Port C
B803H	1	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	CWR

Example 2: Find the control word if PA = OUT;PB = IN; PC0-PC3 = IN; PC4 – PC7 = OUT

Solution: CONTROL WORD = 83 H as developed in Table 2.

Table 2: Control Word Assignment for Example 2

D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
I/O MODE	PORT A			Port C UpperO/P	Port B MODE0	PortB I/P	Port C Lower I/P
	MODE 0		OUT				
1	0	0	0	0	0	1	1
8				**3**			

Example 3: Program the 8255 to get data from Port A and send it to Port B. In addition, data from PCL is sent to PCU. Use port addresses 300H to 303h for 8255 chip

Answer:CW : Control Word = 83h (worked out in example 2)

Port addresses: PA- 300H, PB = 301H; PC = 302H & CWR = 303H.

The Program is developed below.

```
PA EQU 300H
PB EQU 301H
PC EQU 302H
CWR EQU 303H
CW EQU 83H

;initialize CW
MOV DX, CWR
MOV AL, CW
OUT DX, AL

; get data from PA
MOV DX, PA
IN AL, DX
```

```
;send it to PB
MOV DX,PB
OUT DX, AL

; get data from PCL
MOV DX, PC
IN AL, DX

;lower nibble -PCL
AND AL,0FH

;rotate left to upper nibble position
ROL AL,1
ROL AL,1
ROL AL,1
ROL AL,1
OUT DX,AL
```

Example 4: 8255 shown in Fig. 7 is configured as:PA –I/P; PB & PC as O/P. Find control word, find port addresses of PA, PB, PC & CWR. Also Program the ports to input data from PA & send it to PB & PC.

Solution: The control word is developed in Table 3a & the I/O addresses of the port computed from Table 3b.

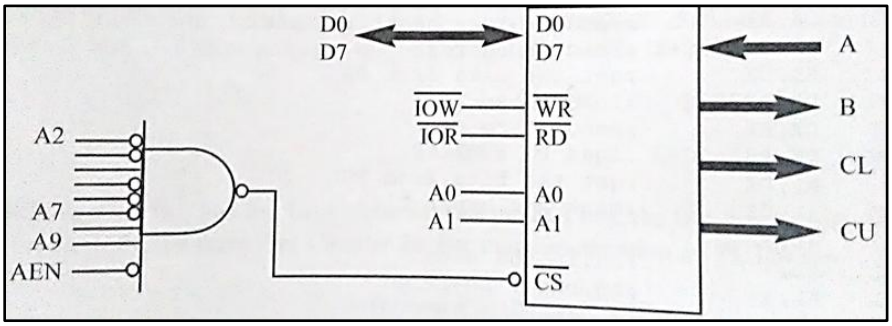


Fig. 7 Figure for example 4

Table 3b Computation of Port addresses from Fig. 7

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	PA	310
								Port A		PB	311
1	1	0	0	0	1	0	0	0	0	PC	312
3		1				0				CWR	313

Table 3a Computation of Control Word for Example 4: PA –I/P; PB & PC as O/P

D7	D6	D5	D4	D3	D2	D1	D0
I/O MODE	PORT A			Port C Upper O/P	Port B MODE 0	Port B O/P	Port C Lower O/P
	MODE 0		i/p				
1	0	0	1	0	0	0	0
9				0			

Program the ports to input data from PA & send it to PB & PC

```
PA EQU 310H
PB EQU 311H
PC EQU 312H
CWR EQU 313H
CW EQU 90H
;initialize CW
MOV DX, CWR
MOV AL, CW
OUT DX, AL
; get data from PA
MOV DX, PA
IN AL, DX
;send it to PB & PC
MOV DX,PB
OUT DX, AL
MOV DX,PC
OUT DX, AL
```

Example 5: Write a program to toggle all the bits of PA continuously. Use INT 16H to exit if there is a key press.

Solution: control word = 80H for all ports as output ports. Assume PA address is 300H; PB – 301H; PC-302H; CWR – 303H. Sending patterns 55h & AAh alternately on the port A, toggles the bits. Delay program is to insert delay of 0.25s in between toggling the pins.

```
;initialize CW
   MOV DX, 303H
   MOV AL, 80H
   OUT DX, AL
Again:
   MOV DX,300H
   MOV AL, 55H
   OUT DX,AL
   CALL DELAY
   MOV AL, 0AAH
   OUT DX, AL
   CALL DELAY
```

```
;check for key press
MOV AH, 01
INT 16H
JZ AGAIN

; exit from pgm
MOV AH, 4CH
INT 21H
```

```
DELAY PROC NEAR
        PUSH AX
;INT 61H waits for 15.085µS
;0.25s = 16592 x 15.085µS
MOV CX, 16592
W1:   IN AL,61H
        AND AL, 00010000B
        CMP AL, AH
        JE W1
        MOV AH, AL
        POP AX
        RET
DELAY  ENDP
```

I/O Programming in Visual C / C++

Microsoft Visual C++ is an Object oriented language. It has many classes & objects to makeProgramming easier & more efficient. Disadv: there is no class or object to directly access I/O ports in the full Windows version of Visual C++. This is done by Microsoft – to ensure x86 programming is under full control of the Operating System and to Preclude any hacking into the system hardware. It applies to Windows NT, 2000, XP & higher versions. i.e., the System instructions INT 21H & I/O operations are not applicable in Windows XP & subsequent versions.

To access I/O & hardware features in XPenvironment, one has to use the Windows Platform SDK provided by Microsoft.In Windows 9x (95 & 98) environment,direct I/O addressing is available, whileINT 21H & other system interrupt instructions are blocked.

To access I/O directly in Windows 9x, Visual C++ must be programmed in Console mode with a different instruction syntax : has _ underscore
ALP has a distinction between 8 & 16 bit addresses; whereas in Visual C++ no such distinction. Port# - any value between 0000 to FFFFH.

Table 4: I/O operations in Microsoft Visual C++ (for Windows 98)

X86 Assembly	**Visual C++**
OUT port#, AL	_outp(port#,byte)
OUT DX, AL	
IN AL,port#	_inp(port#)
IN AL, DX	

Example 6: Write a Visual C++ program for Windows 98 to toggle all bits of PA & PB of the 8255 chip. Use the kbhit function to exit if there is a key press

```
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
#include<iomanip.h>
#include<windows.h>
void main()
{
  cout<<setiosflags(ios::unitbuf);
//When the unitbuf flag is set, the associated buffer is flushed after each insertion operation
cout<<"Pgm for toggling PA,PB"
_outp(0x303,0x80); //80 to CWR
do
{
_outp(0x300,0x55); //PA=55
_outp(0x301, 0x55);   //PB=55
_sleep(500);    //500ms delay
_outp(0x300,0xAA);  //toggle bits
_outp(0x301, 0xAA);
_sleep(500);
}
while(!kbhit());
}
```

Example 7: Write a Visual C++ program for Windows 98 to get a byte of data from PA & send it to both PB &PC of the 8255 chip in PC trainer

To clear the screen in Visual C++, utilize the code: *system*("*CLS*"); The standard library header file <stdlib.h> is needed

```
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
// std::cout, std::hex, std::endl
#include<iomanip.h>
// std::setiosflags
#include<windows.h>
void main()
{   unsigned char mybyte;
        //clear screen buffer
cout<<setiosflags(ios::unitbuf);
System("CLS")
_outp(0x303, 0x90); //PA –i/p; PB,PC-o/p
_sleep(5);
mybyte= _inp(0x300);
_outp(0x301, mybyte);
_sleep(5);
_outp(0x302, mybyte);
_sleep(5);
cout<<mybyte;   //send to PC screen also
cout<<"\n\n";
}
```

I/O Programming in Linux C/C++ : Linux is a popular OS for x86 PC.

To compile I/O programs:
To compile with a keypress loop,
link library ncurses as: >gcc –Incurses toggle.c –o toggle
To run the program: we must 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

X86 ASSEMBLY	LINUX C/C++
OUT port#, AL	outb(byte, port#)
OUT DX, AL	
IN AL, port#	inb(port#)
IN AL,DX	

Example 8: Write a C/C++ program for a PC with Linux OS to toggle all bits on PA & PB. 500ms delay. Key press to exit.

```
//main toggle loop
do
{ //display status on screen
printf("0x55\n\r");
refresh(); //update console
outb(0x55, 0x300);
outb(0x55, 0x301);
usleep(delay);  //500ms=5e5µS
printf("0xaa\n\r"); refresh();
outb(0xaa, 0x300);
outb(0xaa, 0x301);
usleep(delay); //500ms
n= getch();
//if no key press in 1ms, n=0 due to halfdelay()
}
while(n<=0);
//test for key press. If key press, exit program
endwin();
//close program console for ncurses
return 0; //exit program
}
```

Example 9: Write a C/C++ program for a PC with Linux OS to get abyte from PA & send it to PB & PC.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/io.h>
#include<ncurses.h>
int main() {
int n=0;
int i=0;
ioperm(0x300,4,0x300);
outb(0x90,0x303); //CW
initscr(); cbreak(); noecho();
halfdelay(1);
do { i = inb(0x300);
usleep(1e5); //100ms
outb(i,0x301);
outb(i,0x302);
n=getch();
} while(n<=0);
endwin();
return(0);
}
```