Chapter Four

Interrupts & I/O Interfacing

Discussion Points

- Interrupts & its processing
- I/O Interfacing & Address decoding
- ADC and DAC

Interrupts

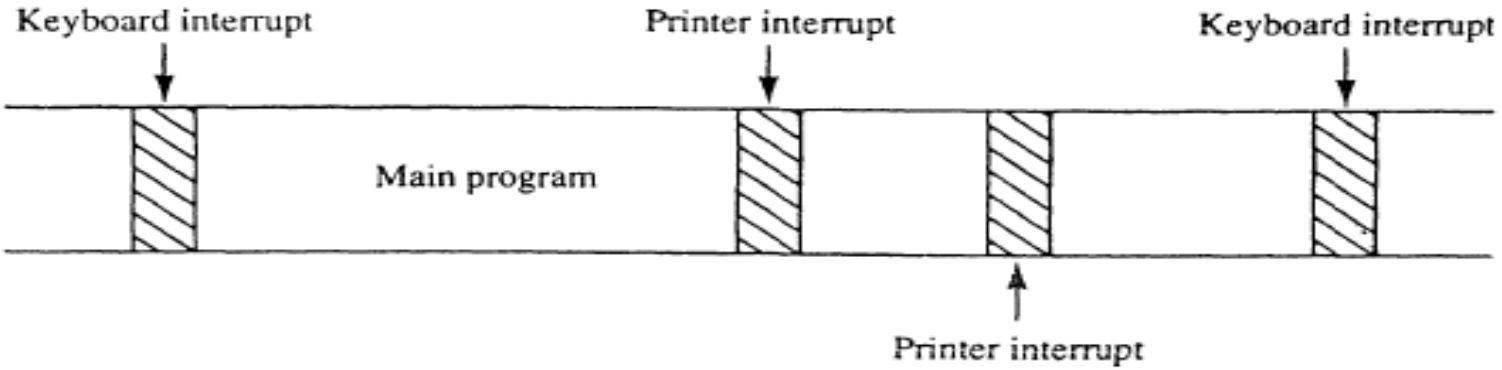
- Capability to suspend the execution of running program and execution of another program to fulfill specific requirement upon request
- The microprocessor stops executing its current program and calls a procedure (Interrupt Service routine) which “services” the interrupt
- After finishing the second program, automatically return to the first program and start execution from where it was left
- A special instruction --- IRET --- at the end of interrupt-service procedure returns execution to the interrupted main program

Purpose of Interrupts

- Interrupts are useful when interfacing I/O devices with low data-transfer rates, like a keyboard or a mouse, in which case polling the device wastes valuable processing time
- The peripheral interrupts the normal application execution, requesting to send or receive data.

For example:

- If a person using the keyboard typed one character per second, the Microprocessor waited an entire second between each key stroke for the person to type another key.
- This process is such a tremendous waste of time that designers have developed another process called interrupt processing.
- interrupt processing allows the microprocessor to execute other program while the keyboard operator is thinking about what key to type next.
- As soon as the key is pressed, the keyboard interrupt controller interrupts the microprocessor.



- Fig Above shows a time line that indicates a typist typing data on a keyboard, a printer removing data from the memory, and a program executing.
- The program is the main program that is interrupted for each keystroke and each character print.
- Note that the keyboard interrupt service routine, called by the keyboard interrupt, and the printer interrupt service procedure each take little time to execute.

Interrupt Types

- 8086 interrupts can be classified into two types:
  - **Internal (or) Software** Interrupts are triggered by a software instruction and operate similarly to a jump or branch instruction.
  - **External (or) Hardware** Interrupts are caused by peripheral devices by sending a signal to INTR pin

Hardware interrupts

- Hardware interrupts are generated by hardware devices when something unusual happens.
  - key-press or a mouse move.
- The two hardware interrupt pins **INTR** and **NMI** that request interrupts, and **INTA** to acknowledge the interrupt request
- The NMI input is often used for parity errors and system faults such as power failures.

INTR and INTA

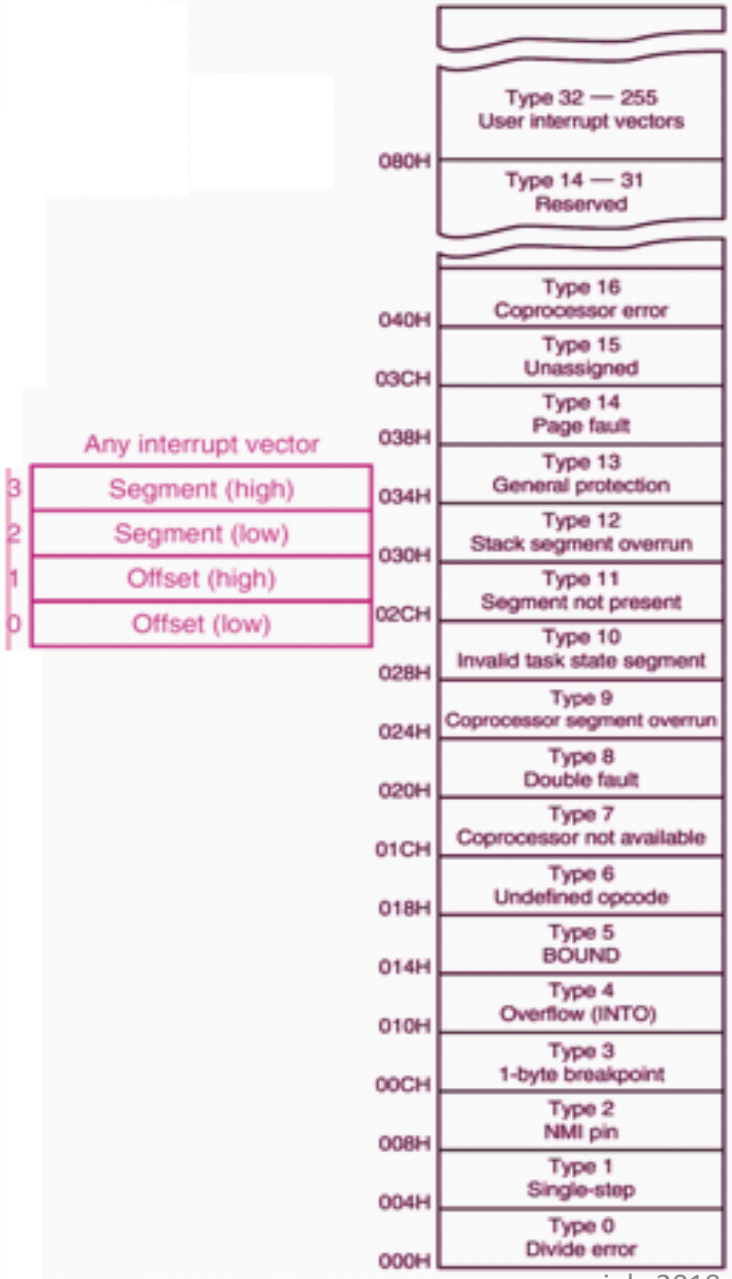
- The INTR pin is set by an external event and cleared inside the interrupt service procedure.
- This input is automatically disabled once it is accepted by the microprocessor and re-enabled by the IRET instruction at the end of the interrupt service procedure.
- The microprocessor responds to the INTR input by pulsing the INTA output in anticipation of receiving an interrupt vector type number on data bus connection.

Example

- In the 82C55 keyboard Interrupt whenever a key is typed, the INTR output becomes a logic 1, requesting an interrupt through the INTR pin on the microprocessor.
- The INTR pin remains high until the ASCII data are read from port A.
- In other words, every time a key is typed, the 82C55 requests a type 40H interrupt through the INTR pin.

Interrupt service routine (ISR)

- 8086 get to the Interrupt Service Routine by
  - Loading its CS and IP registers with the address of ISR.
- 8086 get the address of Interrupt Service Routine (ISR) from the IVT (Interrupt Vector Table)
  - It goes to specified memory location of the IVT to fetch four consecutive bytes
    - Higher two bytes to be used as CS
    - Lower two bytes to be used as IP
  - The first 1Kbytes of memory, from 00000 to 003FF, is set aside as a IVT for storing the starting addresses of up to 256 interrupt service routines
- The starting address of an ISR is often called **interrupt vector** or **interrupt pointer**.
- Table is referred to as **Interrupt-vector table** or **Interrupt-pointer table.**



- The first five interrupt vectors are identical in all intel processors
- Intel reserves the first 32 interrupt vectors
- The last 224 vectors are user-available
- Each is four bytes long in real mode and contains the starting address of the interrupt service procedure.
- The first two bytes contain the offset address
- The last two contain the segment address

Priorities of 8086 Interrupts

Interrupt type	Priority
Divide Error, INT n, INT0	Highest
NMI	
INTR	
Single Step	lowest

Interrupt Flag Bits

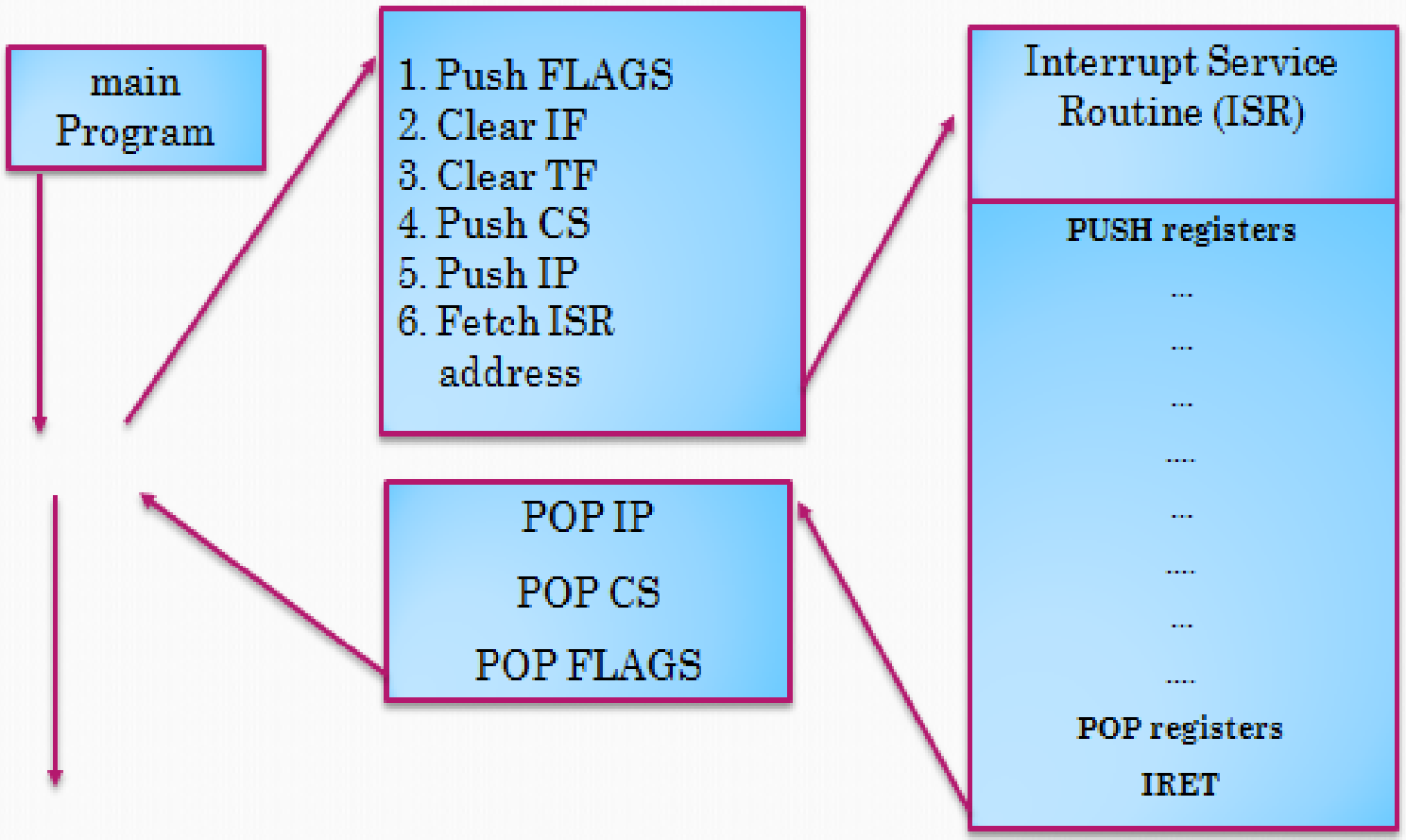
- The interrupt flag (IF) and the trap flag (TF) are both cleared after the contents of the flag register are stacked during an interrupt.
  - when IF is set, it allows the INTR pin to cause an interrupt
  - when IF is cleared, it prevents the INTR pin from causing an interrupt

Interrupt process

If one or more of these interrupt conditions are present, the following sequence of events occurs:

- The contents of the flag register are pushed onto the stack.
- Both the interrupt (IF) and trap (TF) flags are cleared. This disables the INTR pin and also the trap or single-step feature.
- The contents of the code segment register (CS) are pushed onto the stack.
- The contents of the instruction pointer (IP) are pushed onto the stack.
- The interrupt vector contents are fetched and placed into both IP and CS so that the next instruction executes at the interrupt service procedure addressed by the vector.

INTERRUPT PROCESS



I/O Interfacing and Address decoding

- There are two different methods of interfacing

1. Isolated I/O:

  - uses the IN, OUT, INS and OUTS instructions to transfer data to/from I/O device.
  - The I/O address space is separated from memory address space.
  - Memory or I/O operations are indicated by M/IO control signal

Memory Mapped I/O

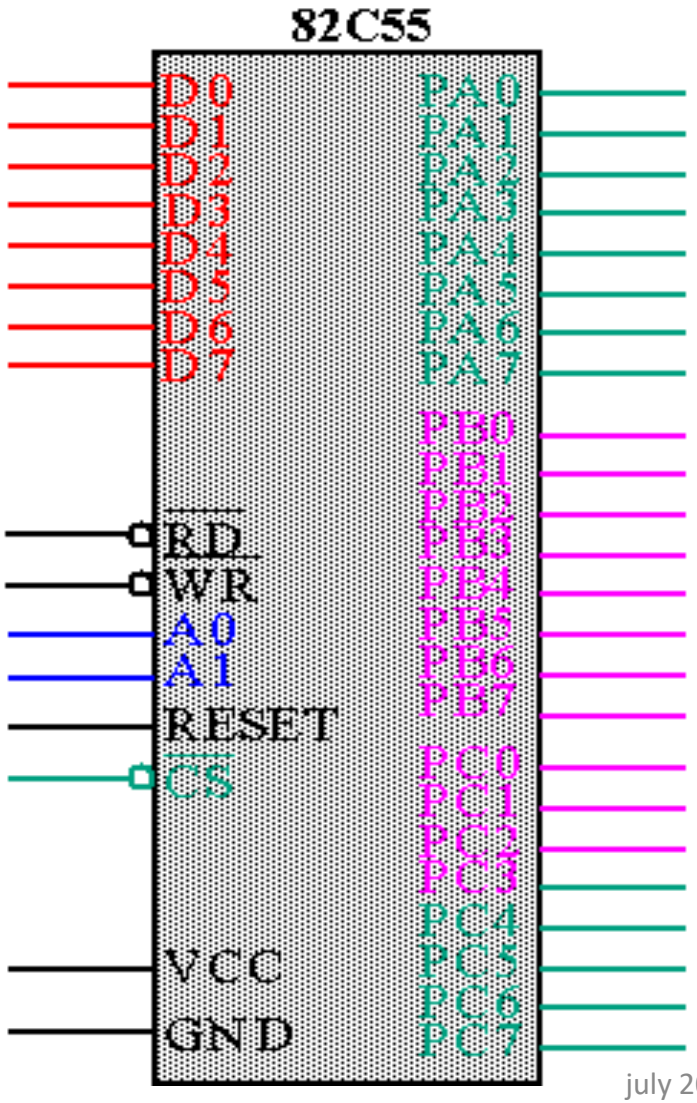
- There is only one single address space for both memory and I/O.
- Memory-mapped I/O does not use the IN, OUT, INS and OUTS instructions.
- It uses any instruction that transfers data between the microprocessor and memory
- Advantage: simple to implement in a program.
- Disadvantage: it takes up a portion of the system memory and reduces the amount of memory available to applications.
  - This is the reason why if you install 4GB of RAM on a PC, you cannot get the full 4GB

PROGRAMMABLE PERIPHERAL INTERFACE-8255

Features:

- It is used to interface to the keyboard and a parallel printer port in PCs
- PPI has 24 pins for I/O that are programmable in groups of 12 pins and has three distinct modes of operation.
- It has 24 I/O programmable pins like PA,PB,PC (3x8 pins).
- Suitable for all parallel port configurations

Pin layout



Group A

Port A (PA7-PA0) and upper half of port C (PC7 - PC4)

Group B

Port B (PB7-PB0) and lower half of port C (PC3 - PC0)

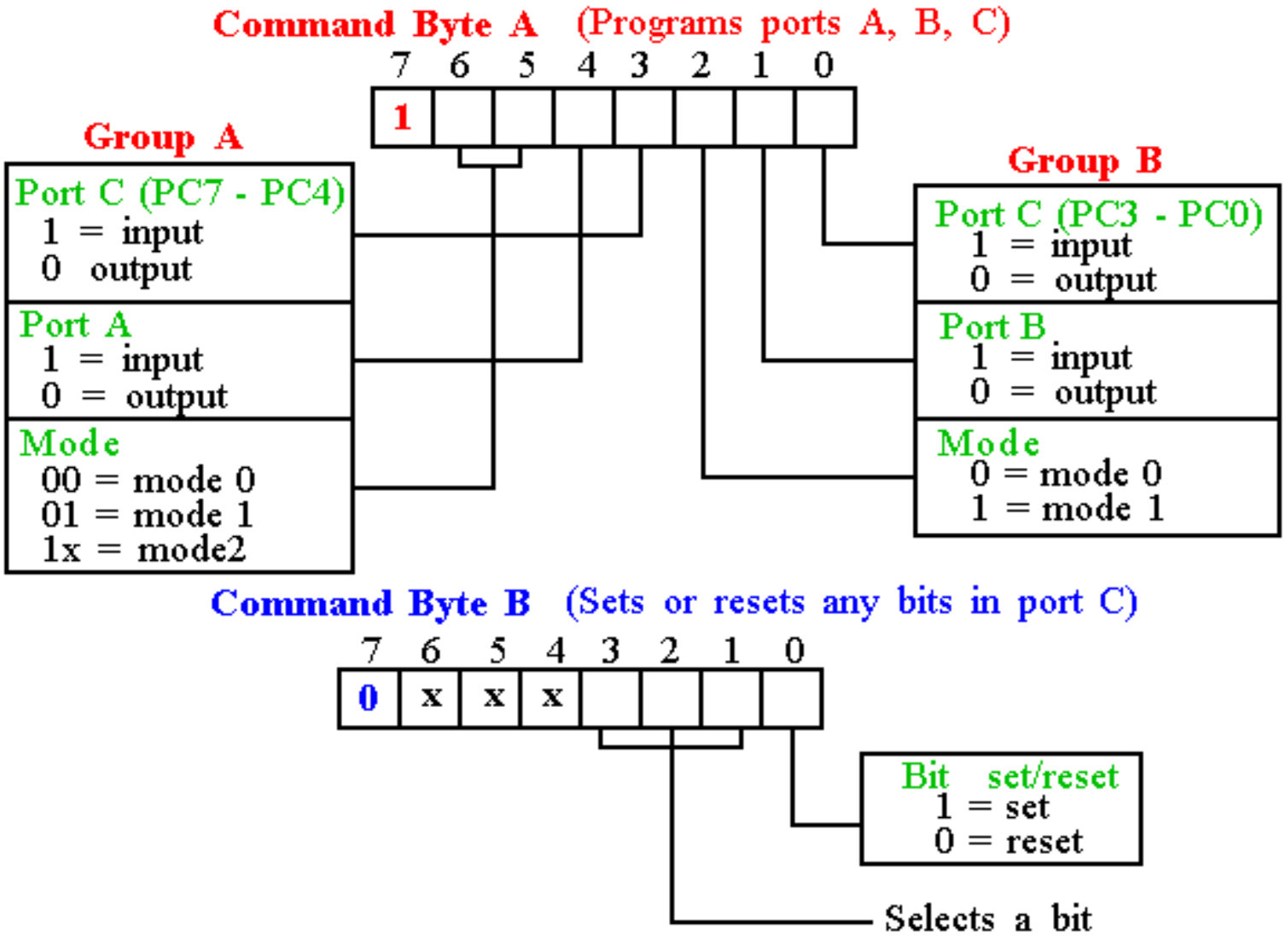
I/O Port Assignments

A_1	A_0	Function
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Command Register

Pin Functions

- Data bus(D_0-D_7):These are 8-bit bi-directional buses, connected to 8088 data bus for transferring data.
- CS: This is Active Low signal. When it is low, then data is transfer from 8088.
- Read: This is Active Low signal, when it is Low read operation will start.
- Write: This is Active Low signal, when it is Low Write operation will start.
- Address (A_0-A_1):This is used to select the ports.
- RESET: This is used to reset the device.
- PA_0-PA_7:It is the 8-bit bi-directional I/O pins used to send or receive data to peripherals
- PB_0-PB_7:Similar to PA
- PC_0-PC_7:This is also 8-bit bidirectional I/O pins.

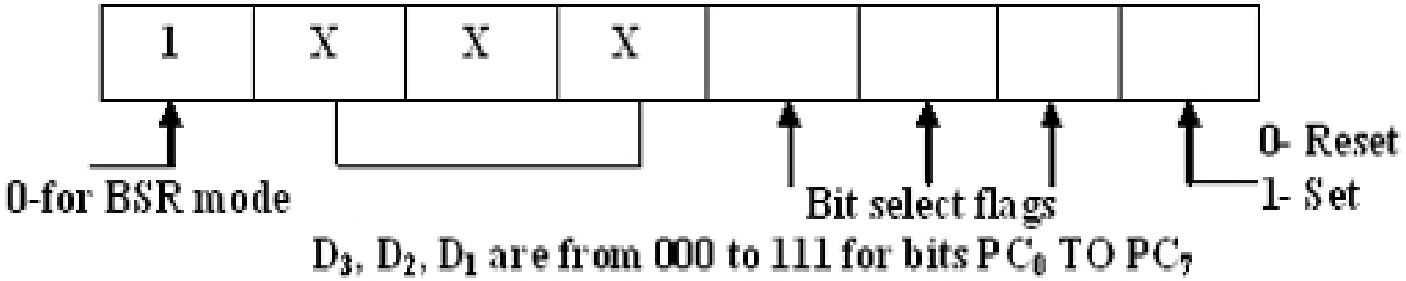
Programming 8255



Formats of control register

•The control word register has two formats. The first format is valid for I/O modes of operation, i.e. modes 0, mode 1 and mode 2 while the second format is valid for bit set/reset (BSR) mode of operation.

•All these modes can be selected by programming a register internal to 8255 known as CWR.

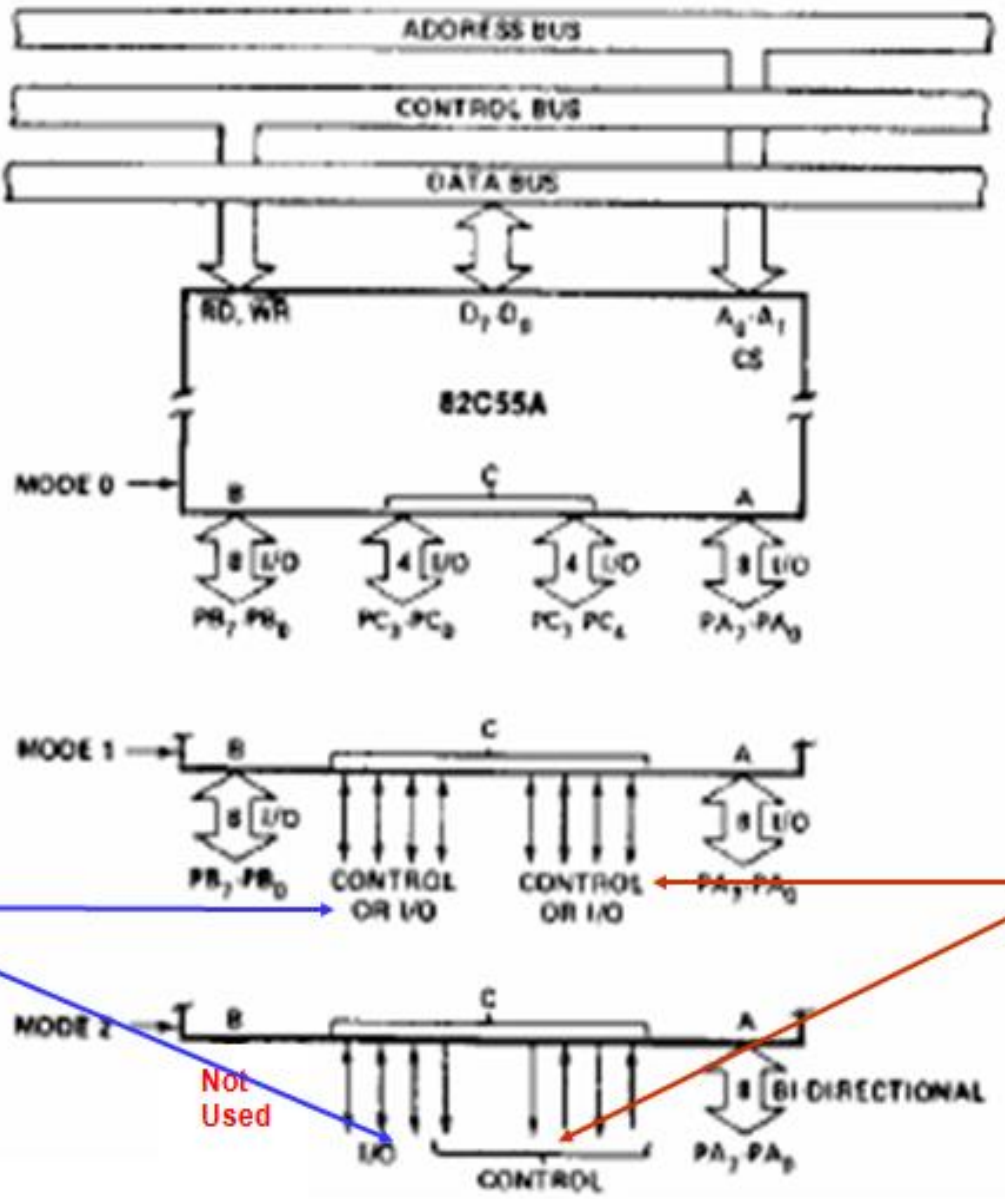


I/O Mode Control Word Register Format and
BSR Mode Control Word Register Format

Operating modes of 8255

- Mode 0 (for groups A & B)- the most commonly used mode: All 12 bits of the group are simple inputs or simple latched outputs
- Mode 1 (for groups A & B)- is used occasionally to provide handshaking to an I/O device and operate asynchronously with the device. Most Port C bits are dedicated for handshake functions for the operation
- Mode 2 (for group A only- Group B not used)- is a bidirectional mode for Port A only (Port B is not used). Port C provides handshaking signals.

8255 Modes



When O/P, Set or Reset Using Command Byte B

Not Used

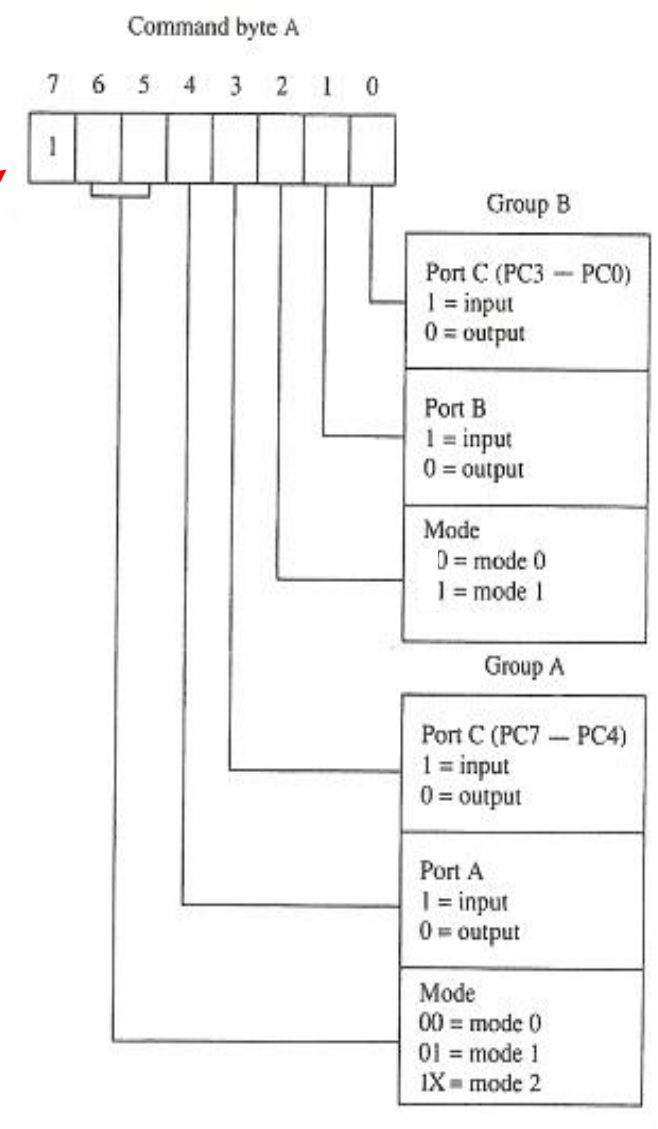
Control: Handshaking For the data port

Example: Programming the 82C55

- To program the command register of the 82C55 and select operation use command byte A
- For example, to program all the ports as outputs and in mode 0 (the most common mode) use:

```
MOV  AL,80H
MOV DX,COMMAND_PORT
OUT   DX,AL
```

Command port= C6H



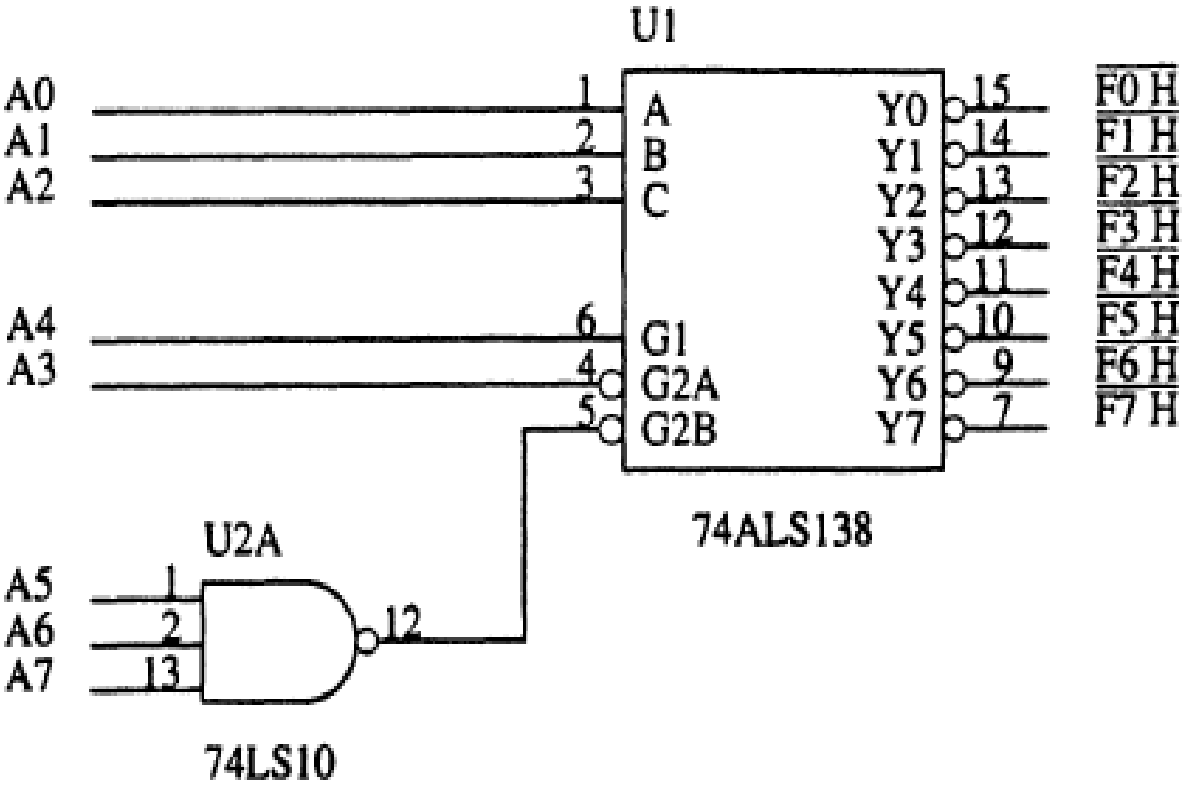
Address decoding of I/O

- I/O port address decoding is very similar to memory address decoding, especially for memory mapped I/O devices.
- The main difference between memory decoding and isolated I/O decoding is the number of address pins connected to the decoder.
- We decode A19-A0 for memory and A15-A0 for isolated I/O.
- Decoding 8-Bit I/O Addresses the fixed I/O instruction uses an 8-bit I/O port address that appears on AI5-AO as OOOOH-OOFFH.
- Also note that if the address is decoded as an 8-bit address, then we can never include I/O devices that use a 16-bit I/O address.

Example

- Consider the system that will only use I/O ports FOH-F7H for this decoder.
- This decoder is identical to a memory address decoder except we only connect address bits F0H-F7H to the inputs of the **decoder.**

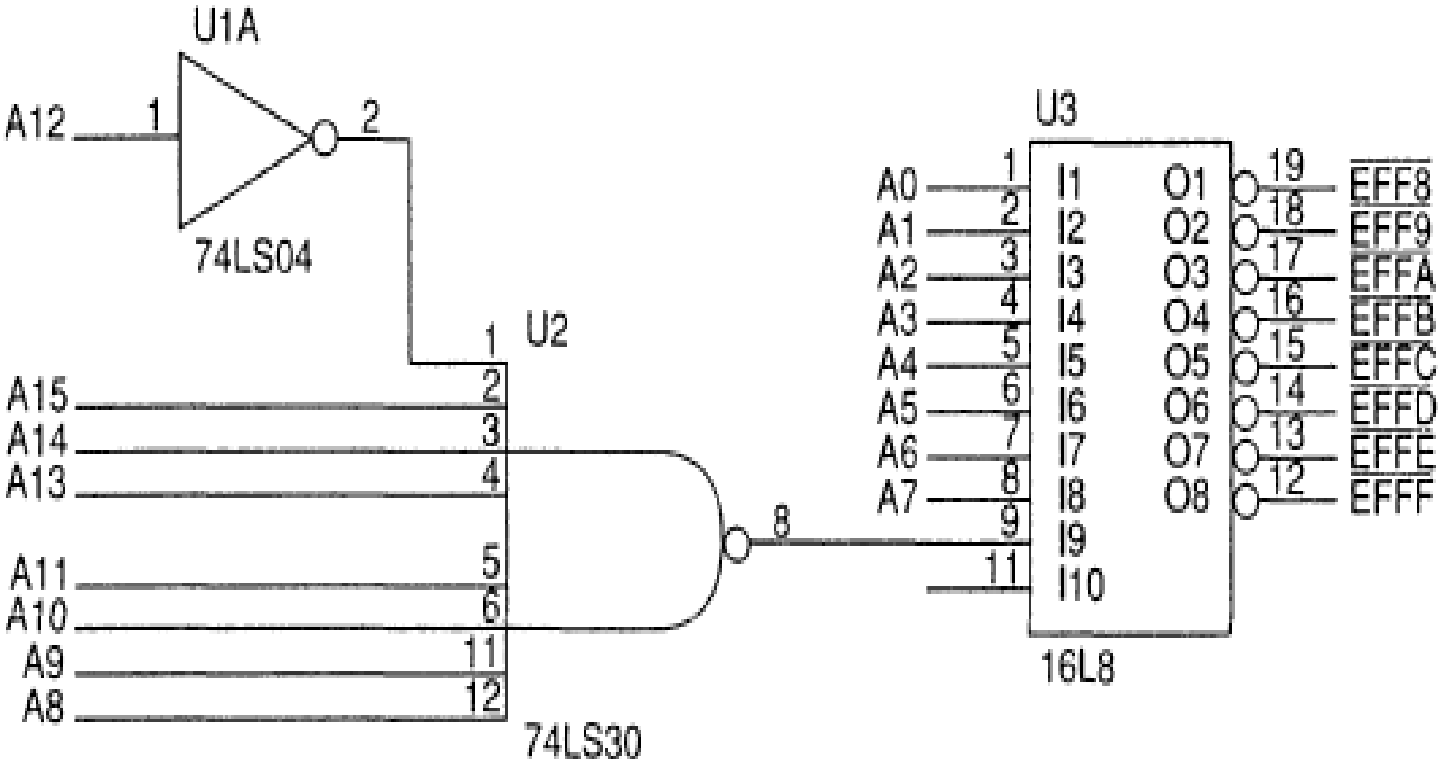
Address decoding example



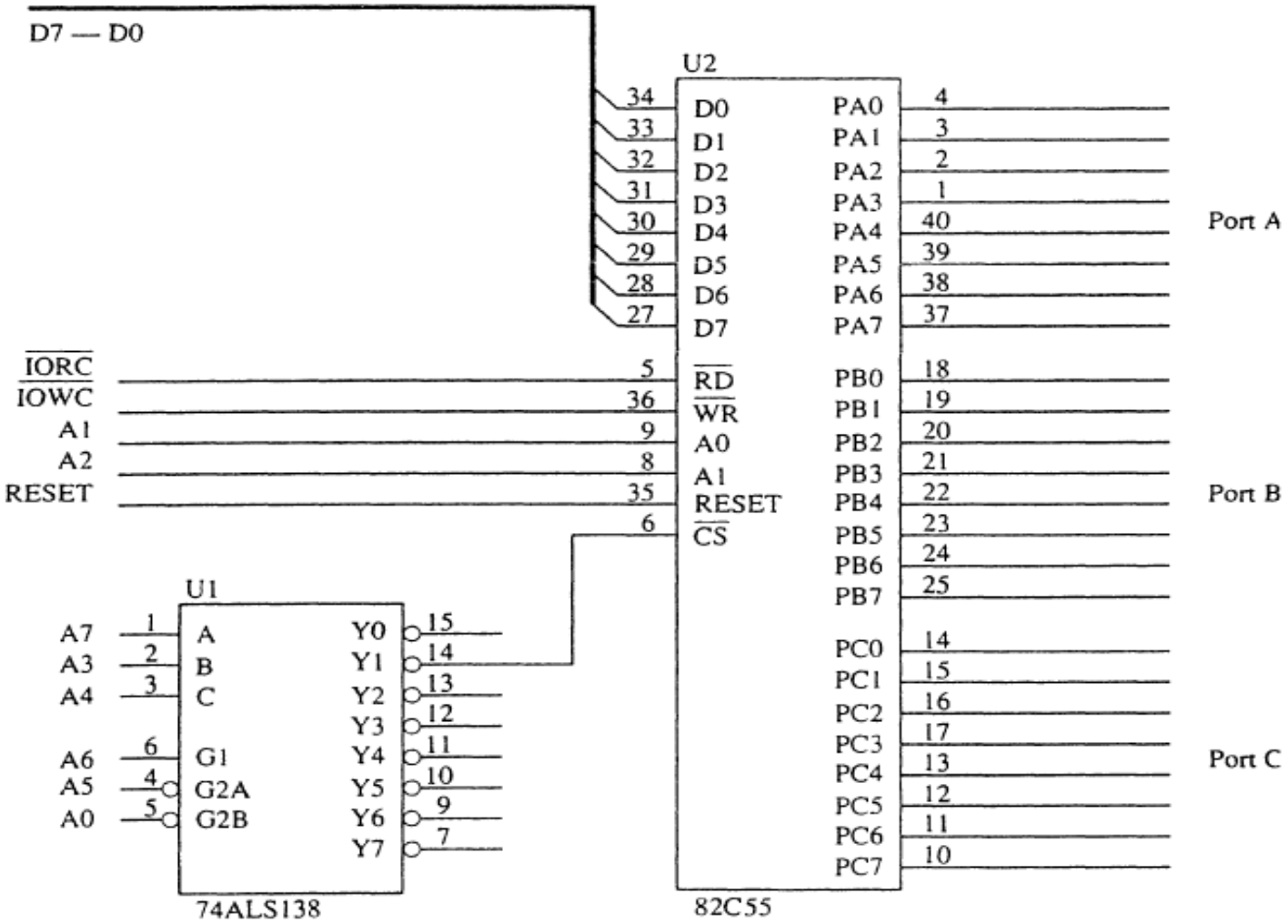
16 bit address decoder

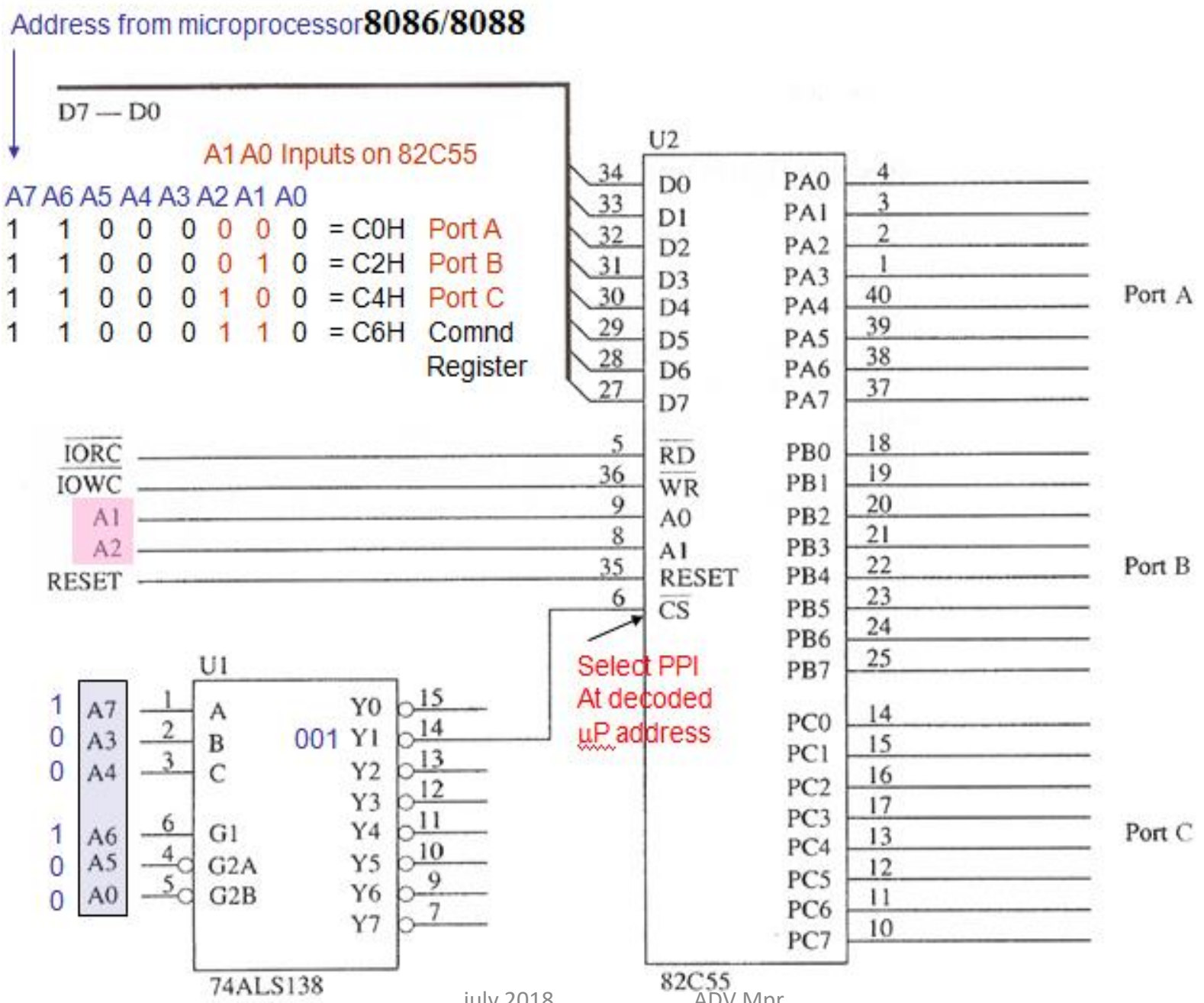
- We also decode 16-bit I/O addresses, especially in a personal computer system.
- The main difference between decoding an 8-bit I/o address and a 16-bit I/O address is that eight additional address lines (A15-A8) must be decoded.
- Consider to decode I/o ports EFF8H-EFFFH

16 bit address decoding



I/O Interfacing using 8255



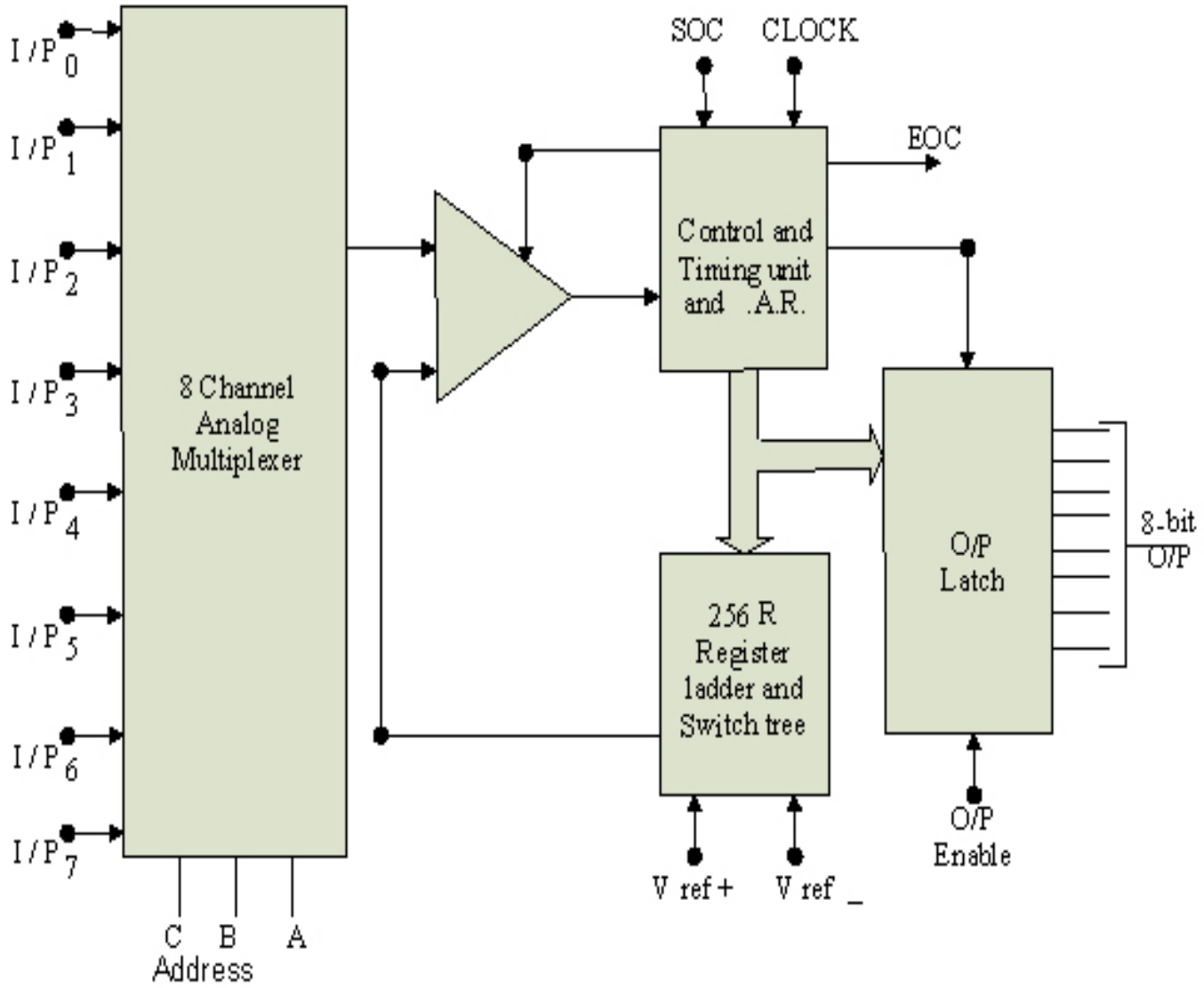

Address from microprocessor 8086/8088
D7 — D0
A1 A0 Inputs on 82C55
A7 A6 A5 A4 A3 A2 A1 A0
1 1 0 0 0 0 0 0 = C0H Port A
1 1 0 0 0 0 1 0 = C2H Port B
1 1 0 0 0 1 0 0 = C4H Port C
1 1 0 0 0 1 1 0 = C6H Comnd Register
U2
D0
D1
D2
D3
D4
D5
D6
D7
RD
WR
A0
A1
RESET
CS
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
82C55
Port A
Port B
Port C
IORC
IOWC
A1
A2
RESET
Select PPI
At decoded
uP address
U1
A7
A3
A4
A6
A5
A0
A
B
C
G1
G2A
G2B
001
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
74ALS138

ADC Interfacing

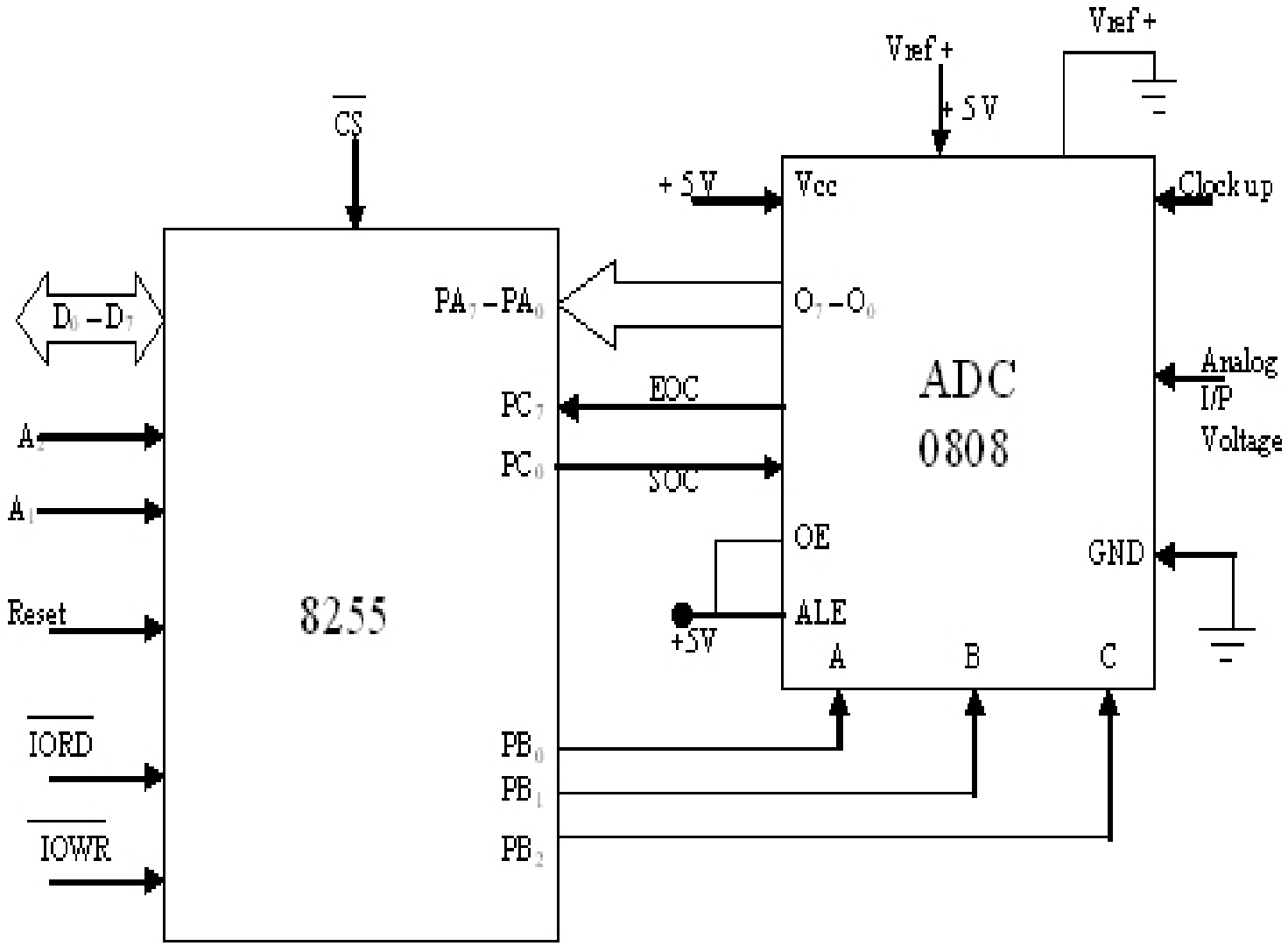
- In most of the cases, the 8255 is used for interfacing the analog to digital converters with microprocessor.
- The microprocessor sends an initializing signal to the ADC to start the analog to digital data conversation process.
- After conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC.
- These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.

ADC 0808 and 0809

- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters.
- The conversion delay is 100µs at a clock frequency of 640 KHz, which is quite low as compared to other converters.
- These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines A, B & C.
- Using these address inputs, multichannel data acquisition system can be designed using a single ADC.



Block Diagram of ADC 0808 / 0809



Interfacing 0808 with 8086

DAC Interfacing

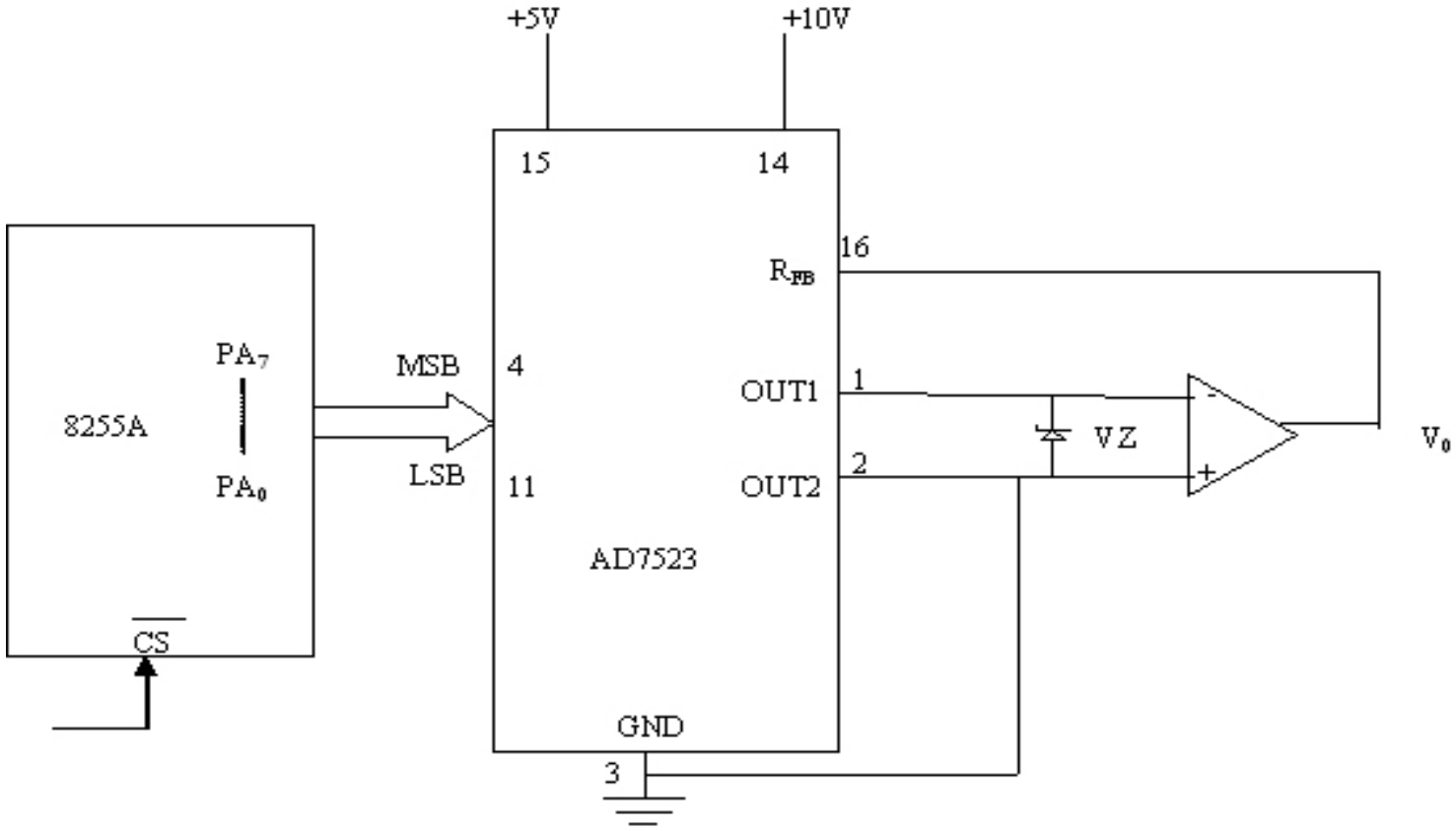


Fig: Interfacing of AD7523