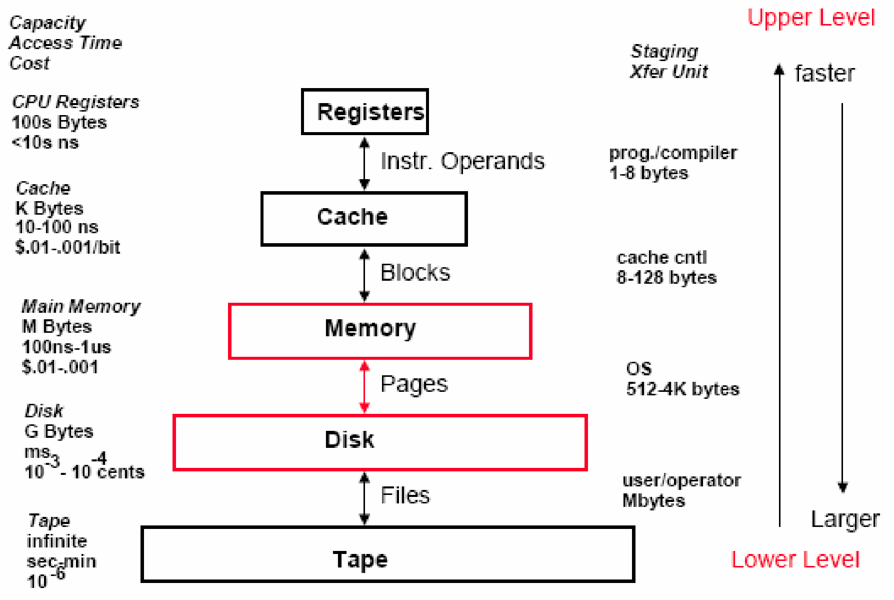
Virtual Memory

Lecture 18

Virtual Memory – Recall

- Virtual and physical address are used to describe address in virtual and physical address space
  - Virtual address space is divided in pages – hard disk
    - Virtual page
  - Some of the pages are copied into memory – **page frame**
    - Physical page
  - Main memory and virtual memory are divided into equal sized pages.
- A process page can be swapped in and out of memory
  - occupying different regions of main memory
- When OS supports virtual memory
  - it is not required for a process to have all its pages loaded in main memory at the time it executes

Levels of Memory Hierarchy



Capacity
Access Time
Cost

CPU Registers
100s Bytes
<10s ns

Cache
K Bytes
10-100 ns
$.01-.001/bit

Main Memory
M Bytes
100ns-1us
$.01-.001

Disk
G Bytes
ms
10^{-3} - 10^{-4} cents

Tape
infinite
sec-min
10^{-6}

Registers
Instr. Operands
Cache
Blocks
Memory
Pages
Disk
Files
Tape

Staging
Xfer Unit

prog./compiler
1-8 bytes

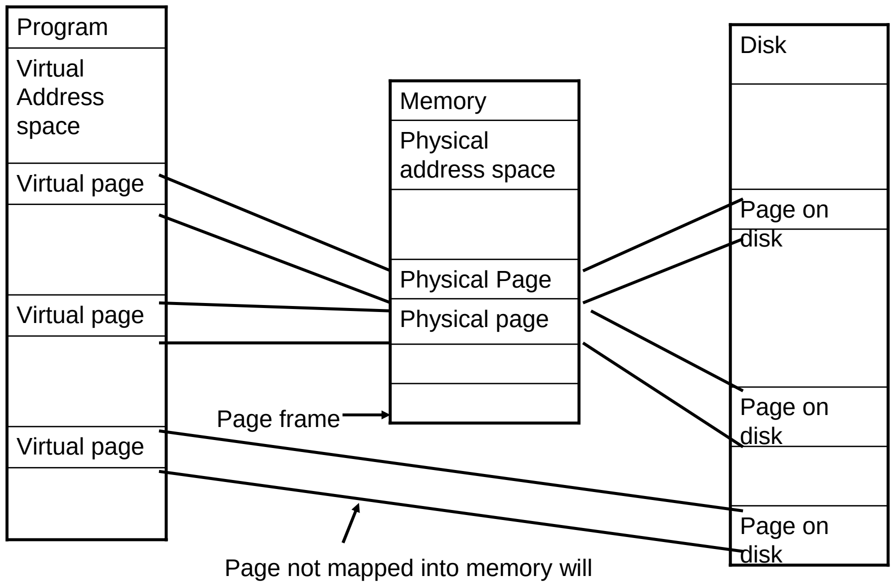
cache cntl
8-128 bytes

OS
512-4K bytes

user/operator
Mbytes

Upper Level
faster
Larger
Lower Level

Virtual memory



Program
Virtual Address space
Virtual page
Virtual page
Virtual page

Memory
Physical address space
Physical Page
Physical page
Page frame

Disk
Page on disk
Page on disk
Page on disk

Page not mapped into memory will be copied in memory when needed

Address Translation

- When a process generates a virtual address, the operating system translates it into a physical memory address.
- To accomplish this,
  - The virtual address is divided into two fields:
    - A *page* field, and an *offset* field.
  - The page field determines the page location of the address.
  - The offset indicates the location of the address within the page.
- The logical page number is translated into a physical page frame through a lookup in the page table.

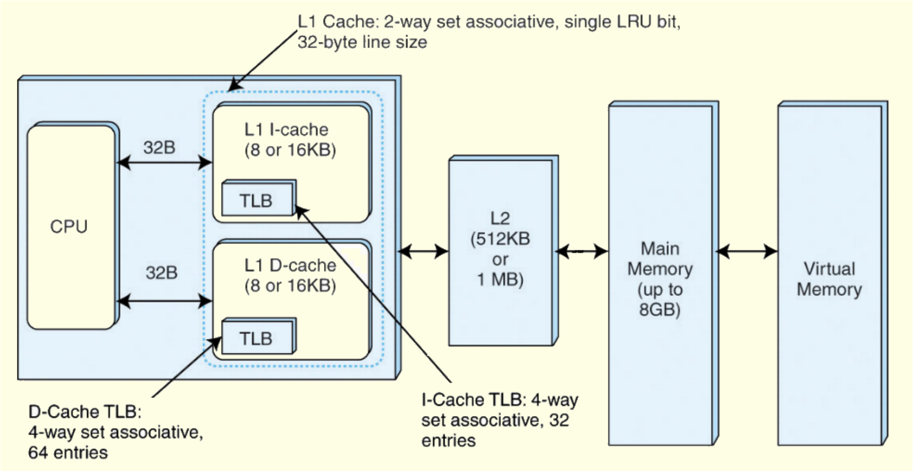
Page Fault

- If the valid bit is zero in the page table entry for the logical address, this means that the page is not in memory and must be fetched from disk.
  - This is a page fault.
  - If necessary, a page is evicted from memory and is replaced by the page retrieved from disk, and the valid bit is set to 1.
- If the valid bit is 1, the virtual page number is replaced by the physical frame number.
- The data is then accessed by adding the offset to the physical frame number.

Page Table

- Because page tables are read constantly, it makes sense to keep them in a special cache called a *translation look-aside buffer* (TLB).
  - TLBs are a special associative cache that stores the mapping of virtual pages to physical pages.

Translation look-aside buffer- TLB



L1 Cache: 2-way set associative, single LRU bit, 32-byte line size
CPU
32B
32B
L1 I-cache (8 or 16KB)
TLB
L1 D-cache (8 or 16KB)
TLB
L2 (512KB or 1 MB)
Main Memory (up to 8GB)
Virtual Memory
D-Cache TLB: 4-way set associative, 64 entries
I-Cache TLB: 4-way set associative, 32 entries

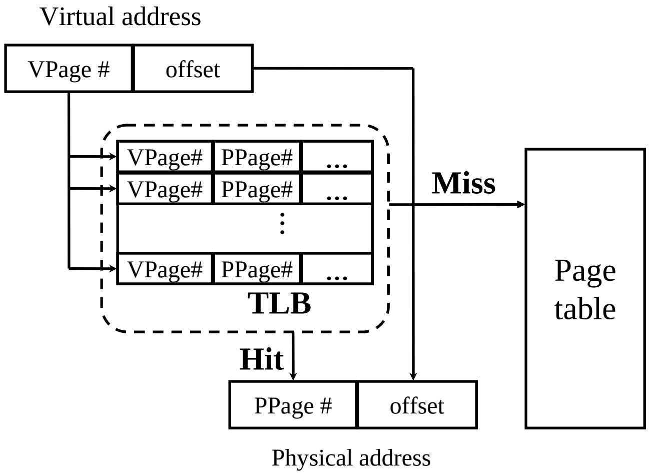
Translation look-aside buffer- TLB

- **TLB** is a CPU cache that memory management hardware uses to improve virtual address translation speed:
  - All current desktop and server processors use a TLB to map virtual and physical address spaces
  - TLB implemented as Content Addressable Memory (CAM).
    - The CAM search key is the VIRTUAL ADDRESS and the search result is a PHYSICAL ADDRESS

TLB- Search

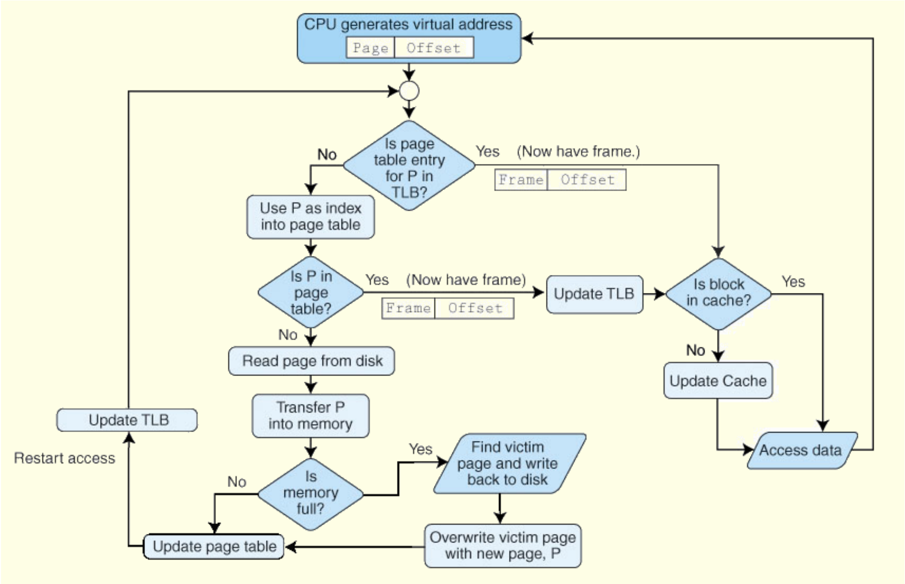
- CAM Searching
  - If the requested address is present in the TLB:
    - The CAM retrieved physical address to access memory
      - This is called a TLB hit.
  - If the requested address is not in the TLB, it is a miss:
    - The translation proceeds by looking up the page table in a process called a *page walk*
      - The page walk is an expensive process, as it involves reading the contents of multiple memory locations and using them to compute the physical address
  - After the physical address is determined by the page walk, the virtual address to physical address mapping is entered into the TLB.

Translation Lookaside Buffer (TLB)



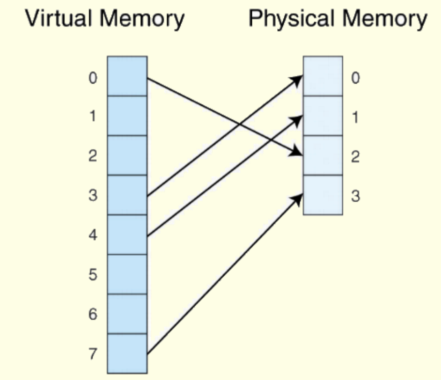
Note that each TLB entry must include the virtual page # as well as the corresponding PP#

Virtual Memory



Page Table

Example: How processes pages are located in memory



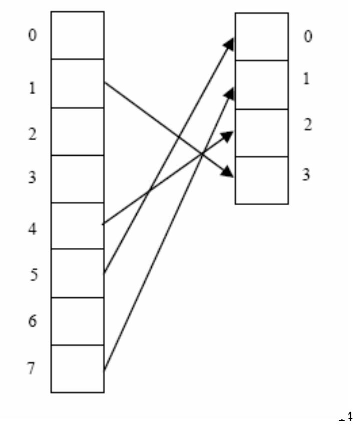
Page Table

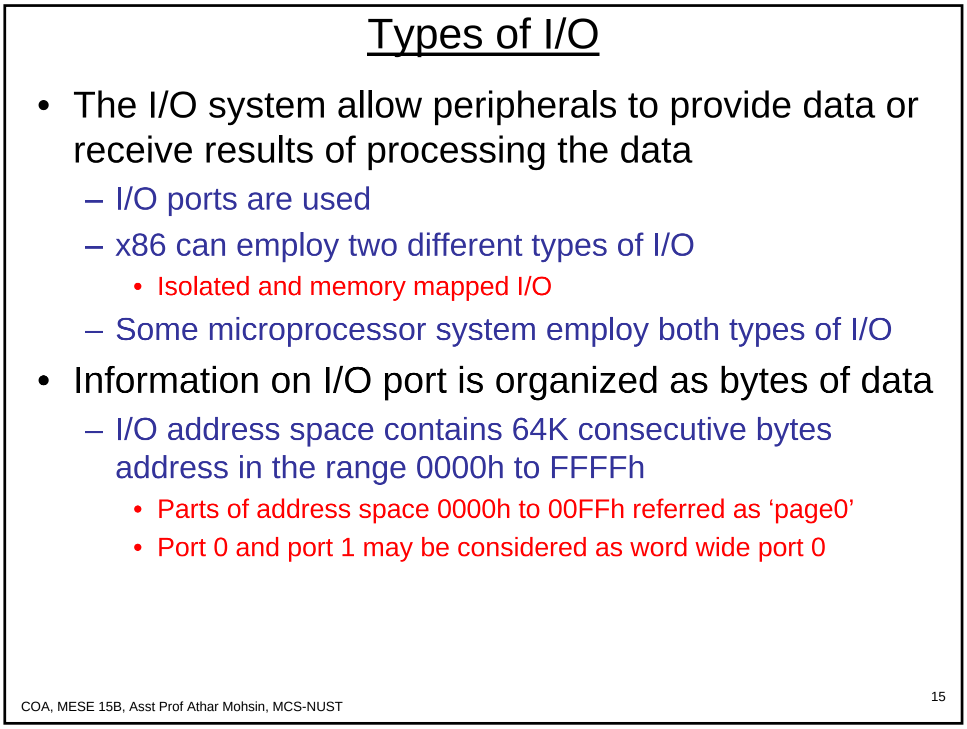
Page	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

Example

- Process page table showing the entries. Can you show where the process pages are located in memory?

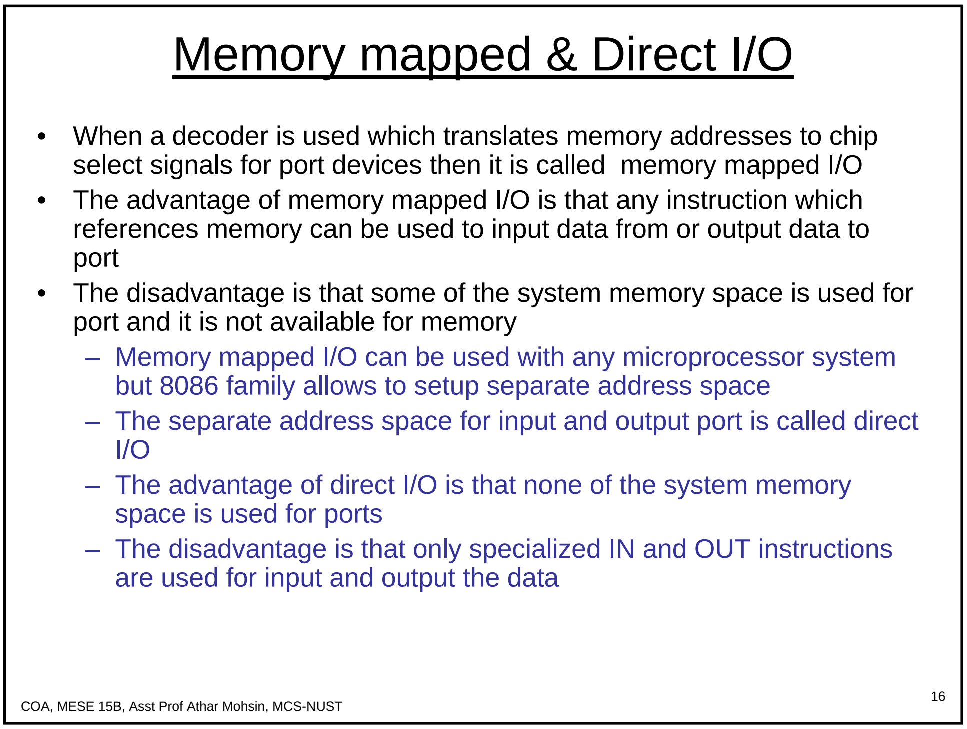
Frame	Valid Bit
--	0
3	1
--	0
--	0
2	1
0	1
--	0
1	1





Types of I/O

- The I/O system allow peripherals to provide data or receive results of processing the data
  - I/O ports are used
  - x86 can employ two different types of I/O
    - Isolated and memory mapped I/O
  - Some microprocessor system employ both types of I/O
- Information on I/O port is organized as bytes of data
  - I/O address space contains 64K consecutive bytes address in the range 0000h to FFFFh
    - Parts of address space 0000h to 00FFh referred as ‘page0’
    - Port 0 and port 1 may be considered as word wide port 0



Memory mapped & Direct I/O

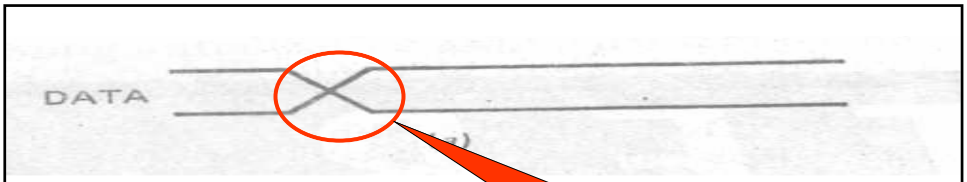
- When a decoder is used which translates memory addresses to chip select signals for port devices then it is called memory mapped I/O
- The advantage of memory mapped I/O is that any instruction which references memory can be used to input data from or output data to port
- The disadvantage is that some of the system memory space is used for port and it is not available for memory
  - Memory mapped I/O can be used with any microprocessor system but 8086 family allows to setup separate address space
  - The separate address space for input and output port is called direct I/O
  - The advantage of direct I/O is that none of the system memory space is used for ports
  - The disadvantage is that only specialized IN and OUT instructions are used for input and output the data

Direct I/O

- Whenever 8086 executes an IN and OUT instruction to access a port,
  - none of the segment registers produce the physical address,
  - the port address are directly sent out by 8086 through AD0 to AD15 and 0’s are output on line A16 to A19
- In 8086 which uses direct I/O
  - The M/IO is used to enable a memory decoder or a port decoder
    - M/IO high enable ROM and RAM decoder
    - M/IO low is used for port decoder
- Since 8086 outputs up to a 16 bit address for direct I/O operation so it can address any one of 2^{16} input ports and output any one of 2^{16} output ports

Parallel Data Transfer - Methods

- Simple Input and Output
  - To get the digital data from a simple switch into microprocessor
    - Connect the switch to an input port line and read the port
  - To output data to a simple display device (LED)
    - Connect the LED on an output port pin and output the required logic level to turn on the light

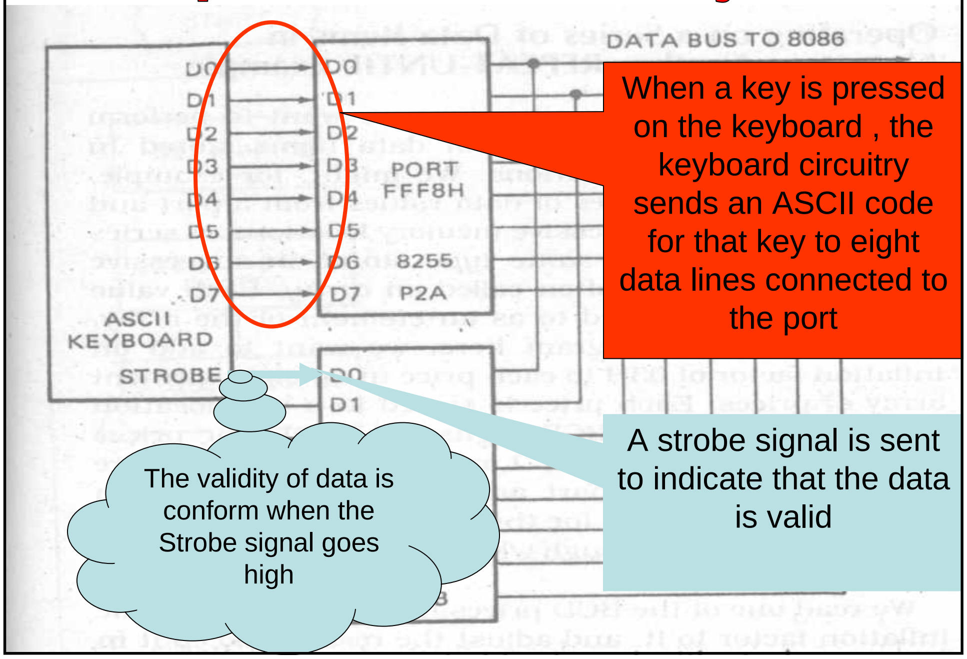


Crossed line represents the time at which the new data byte becomes valid on the output lines of the port

Simple Strobe I/O

- In applications when valid data is present on an external device only at a certain time
  - The data must be read in at that time
- System interface with the microcomputer output data on parallel signal lines and then output a separate signal to indicate the presence of valid data on the parallel line, that data ready signal is often called a "Strobe"

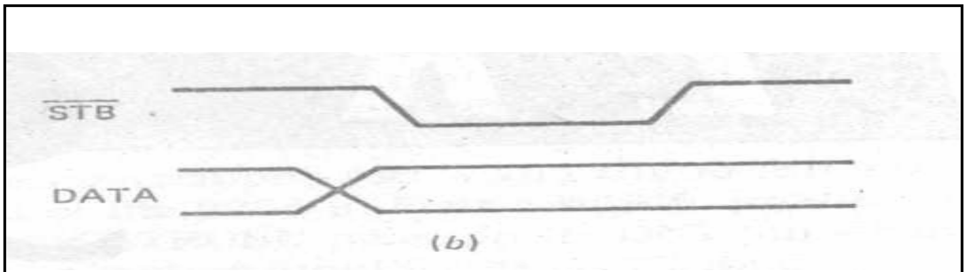
Example of Strobe data system



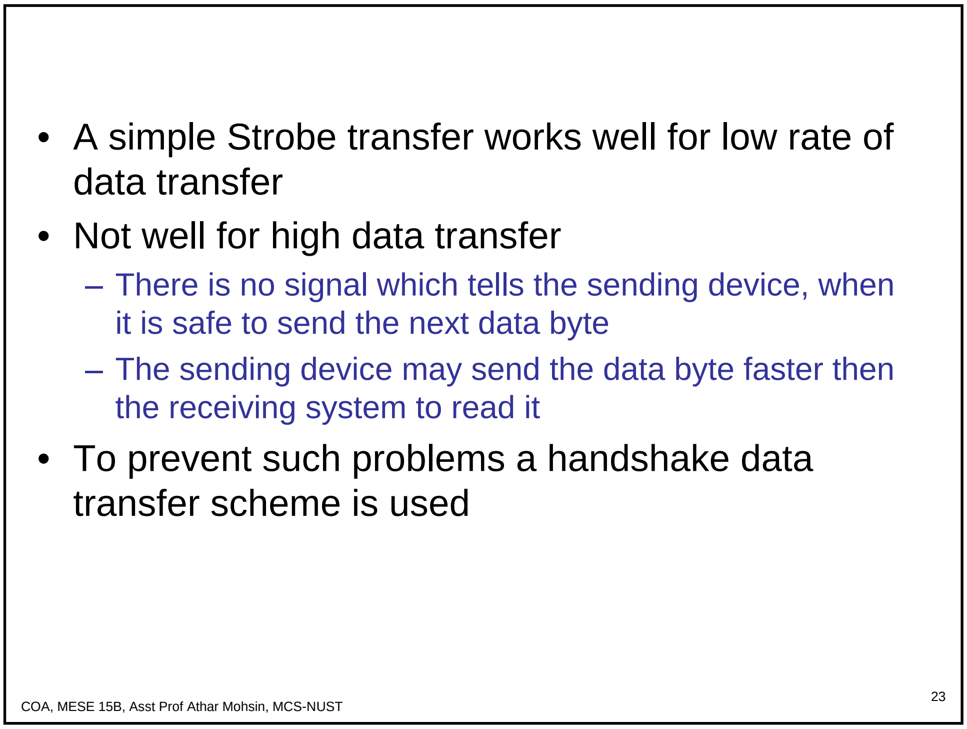
When a key is pressed on the keyboard , the keyboard circuitry sends an ASCII code for that key to eight data lines connected to the port

A strobe signal is sent to indicate that the data is valid

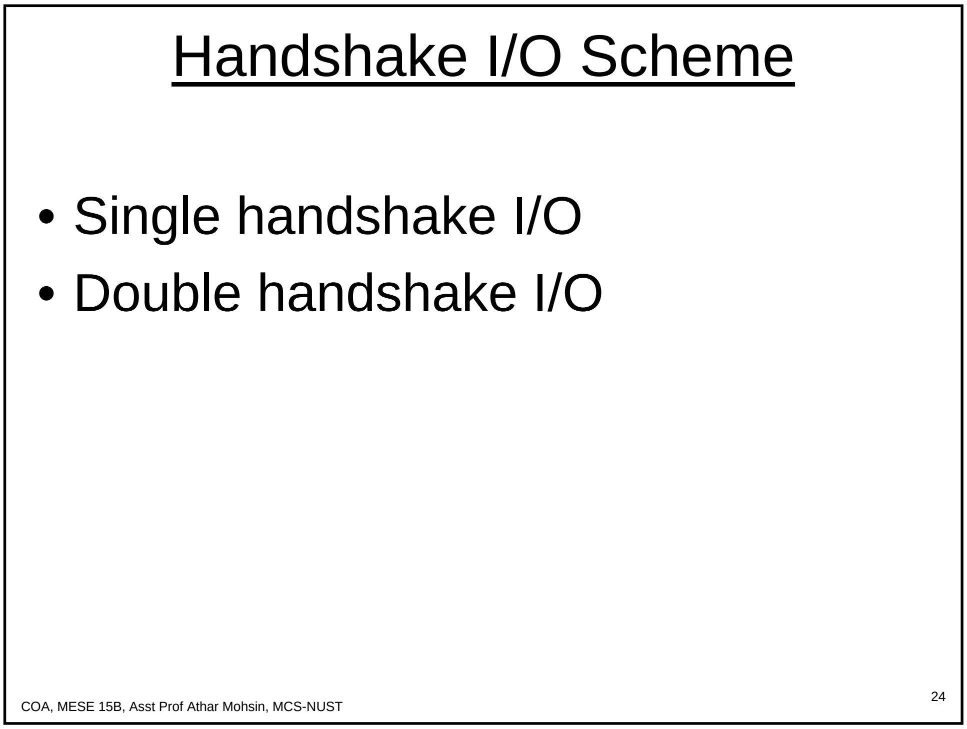
The validity of data is conform when the Strobe signal goes high



The sending device (keyboard) output parallel data on the data line and then sends out the $\overline{\text{STB}}$ signal to confirm the validity of the data



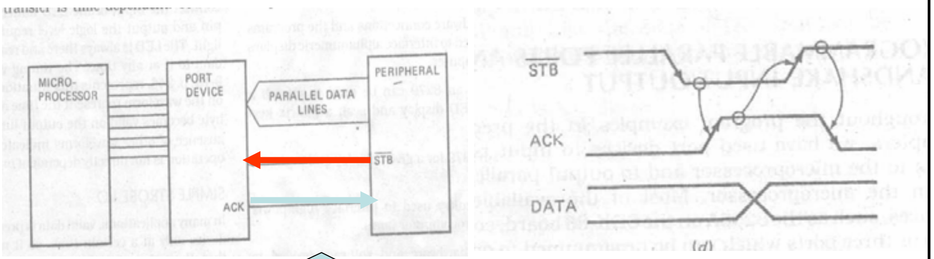
- A simple Strobe transfer works well for low rate of data transfer
- Not well for high data transfer
  - There is no signal which tells the sending device, when it is safe to send the next data byte
  - The sending device may send the data byte faster then the receiving system to read it
- To prevent such problems a handshake data transfer scheme is used



Handshake I/O Scheme

- Single handshake I/O
- Double handshake I/O

Single Handshake I/O

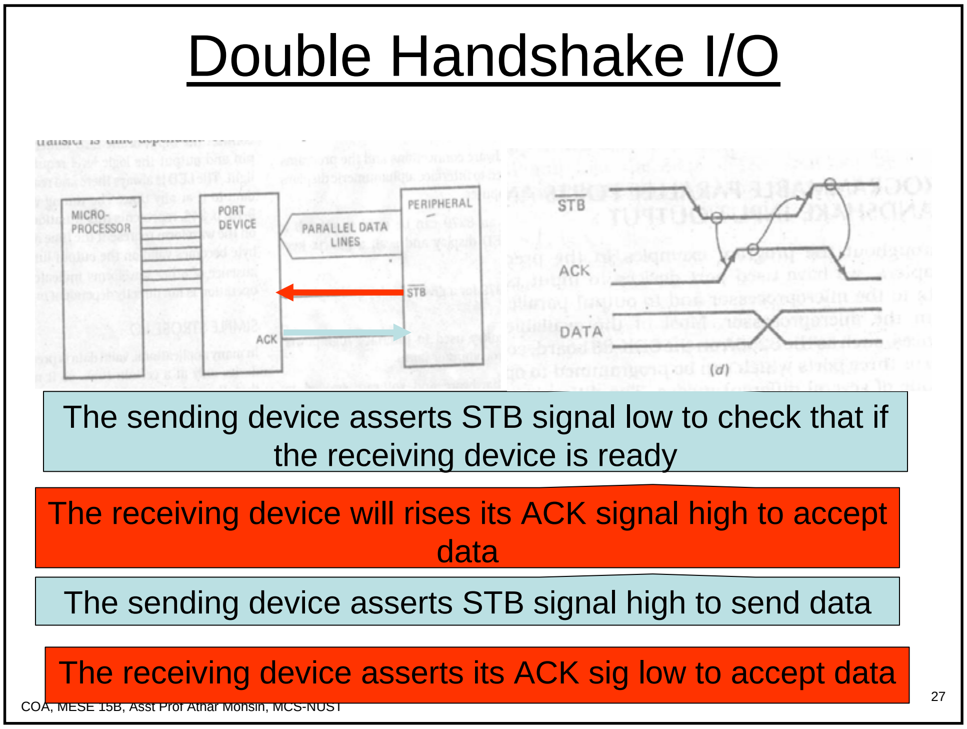


The peripheral sends an parallel data and sends an STB sig while microprocessor send the ACK signal to peripheral

The operation is refer to as a handshake or strobed I/O

Double handshake data transfer

- When more coordination is required between sending and receiving system



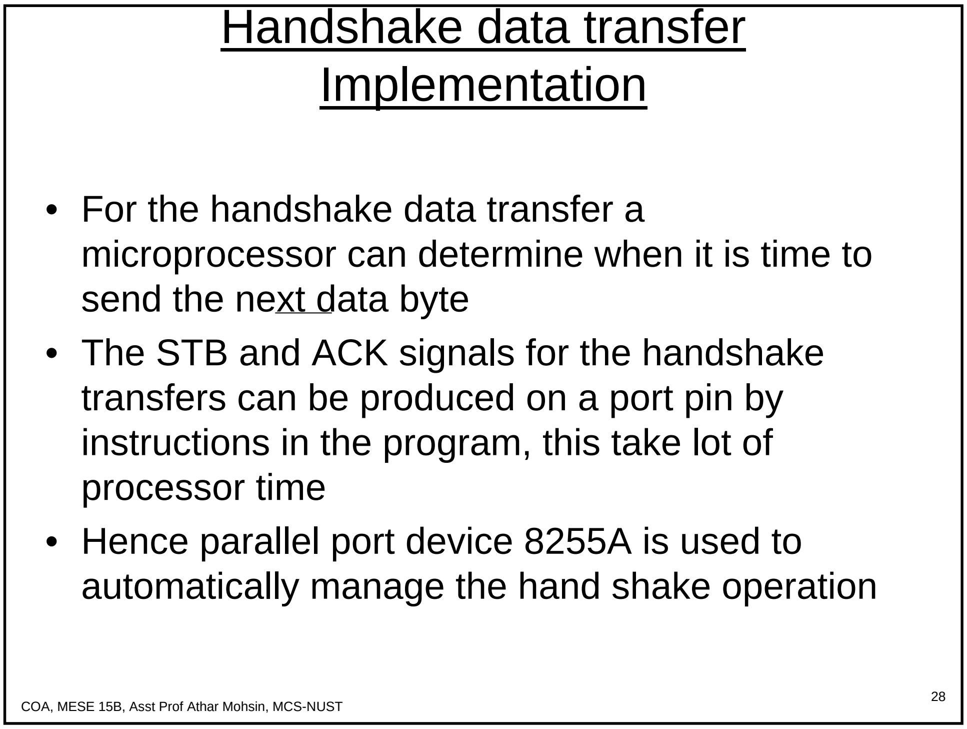
Double Handshake I/O

The sending device asserts STB signal low to check that if the receiving device is ready

The receiving device will rises its ACK signal high to accept data

The sending device asserts STB signal high to send data

The receiving device asserts its ACK sig low to accept data



Handshake data transfer Implementation

- For the handshake data transfer a microprocessor can determine when it is time to send the next data byte
- The STB and ACK signals for the handshake transfers can be produced on a port pin by instructions in the program, this take lot of processor time
- Hence parallel port device 8255A is used to automatically manage the hand shake operation