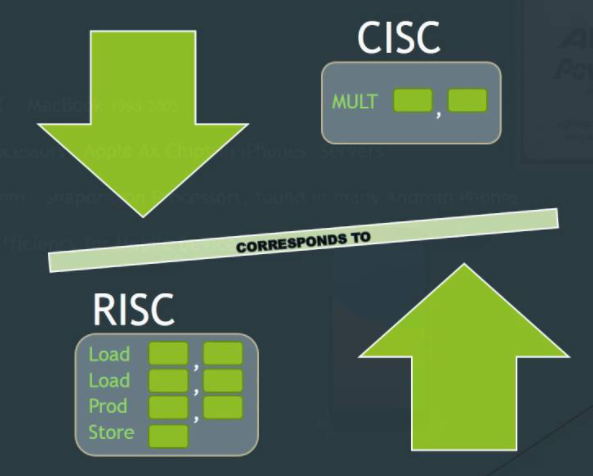
**Lecture 1**

CISC**:**  Complete Instruction set

RISC**:**  Reduced Instruction set

* All instruction in Risc take only 1 cycle
* Micro Controller is processor do only one thing
* Power consumption proportionate to Clock speed

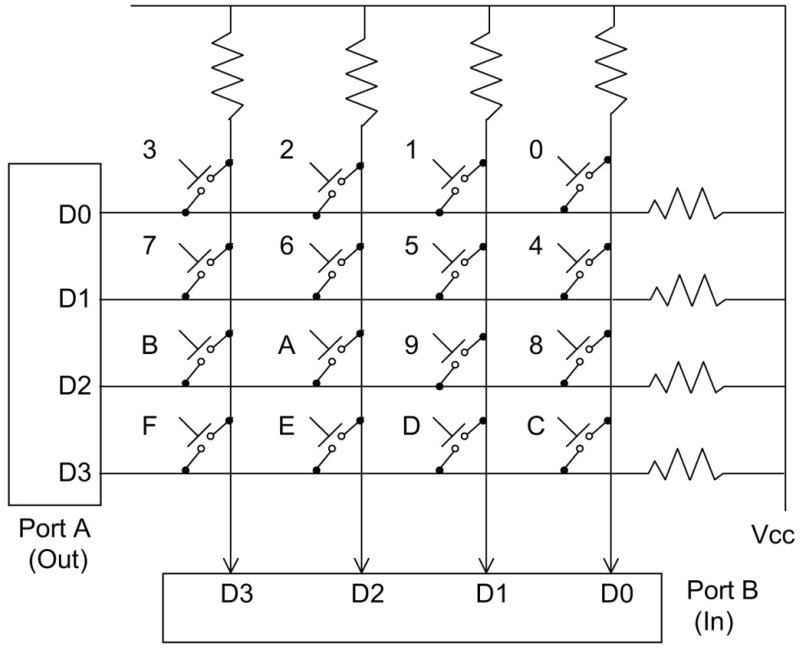
|  |  |
| --- | --- |
| CISC | RISC |
| FEW INSTRUCTIONS | MULTIPLE INSTRUCTIONS |
| LESS REGISTERS | MORE REGISTERS |
| MORE MICROPROGRAMMING | MORE COMPLEX COMPILERS |
| N CYCLE TIMES PER INSTRUCTION | ONE CYCLE TIME PER INSTRUCTION |
| HARDWARE FOCUSED | SOFTWARE FOCUSED |

\*Risc have more registers because we want to make it faster without increase clock Cycle

**KEYBOARD**

-Keyboards are organized in a matrix of rows and columns, and the CPU accesses rows & columns through ports.

-With two 8-bit ports, an 8 x 8 = A \* A matrix of keys can be connected to a microprocessor. A = Log2(n of keys)

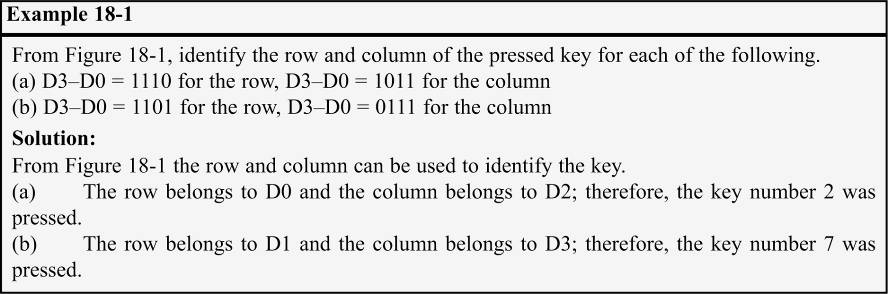
- If no key has been pressed, the input port will yield 1s for all columns, since they are all connected to high. (Vcc)

- at start all Port A is 0

- when key pressed we start (**Detection phase**) port B corresponding to clicked key become 0 due to short circuit Between it and Port A Corresponding

- as ex if we click 9 so B(D1) will become zero but that mot enough because if 1 ,5,D clicked same will happed

- here start (**Identification phase**) we try put 0 to each one Of Port A and see Port B so if Corresponding Port B is 1 so that mean its not right & if is 0 so that mean we are right and this is corresponding key.

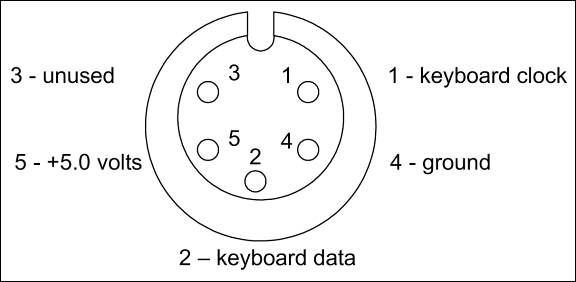
- so to do that we create Micro Controller to do that operation and send it to processor serial to save money because human is so slow compared to computer.

-EX

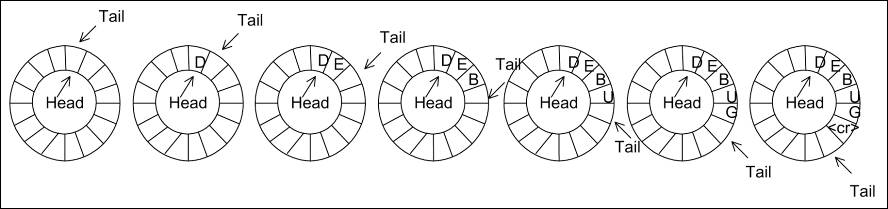
* IBM PC/AT keyboards use the following data frame to send scan code serially to the motherboard.
  + For each scan code, a total of 11 bits are transferred.
  + One start bit (always 0)
  + 8 bits for scan code
    - 7 bit for data
    - 1 bit to detect if it pressed(clicked = 0) or Release(Break = 1)
  + Odd parity bit
  + One stop bit (always 1)
* So each button is sent twice once with clicked & once for Break
* IRQ : interrupt request
  + It’s the priority according to it processor doe which tasks first or right now
  + Higher is zero 0 --> higher

-So all processor will be like

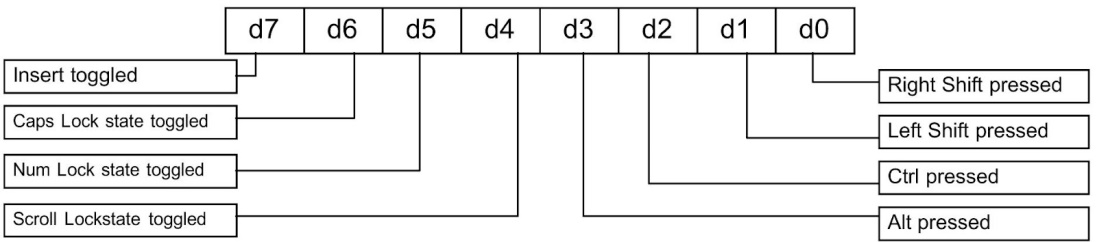
* The keyboard microcontroller scans the keyboard matrix continuously.
* When a key is pressed (a make)…
  + It is identified and its scan code is sent serially (Scan & ASCII Code) to the motherboard through the keyboard cable.
* The circuitry on the motherboard…
  + Receives the serial bits.
  + Gets rid of the frame bits
  + Makes one byte (scan code) with its serial-in-parallel-out shift register
  + Presents this 8-bit scan code to port A of 8255 at I/O addresses 60H.
  + Activates IRQ1.
* Since IRQ1 is set to INT 09, its interrupt service routine (ISR) residing in BIOS ROM is invoked.
  + ISR of INT 09 reads the scan code from port 60H.
  + ISR of INT 09 tests the scan code to see if it is the Right or Left Shift, Alt, Ctrl keys, etc.
    - If so, the appropriate bit of the keyboard status bytes in BIOS 0040:0017H and 0018H are set.
      * It will not write the scan code to the keyboard buffer.
* Before returning from INT 09, ISR will issue EOI(استني) to unmask IRQ1, followed by the IRET instruction.
  + This allows IRQ1 activation to be responded to again
* When the key is released (a break), the keyboard generates the second scan code by adding 80H to it and sends it to the motherboard.
* ISR of INT 09 checks the scan code to see if there is 80H difference between this code and the old one.
  + If D7 is high, it is interpreted as meaning the key has been released &the system ignores the 2nd scan code.
  + If the key is held down more than 0.5 seconds, it is interpreted as a new key and INT 09 will write it into  
    the keyboard buffer (32 bit which mean 16 char only in buffer) next to the preceding one. Commonly referred to as typematic in IBM literature, which means repeating the same key.



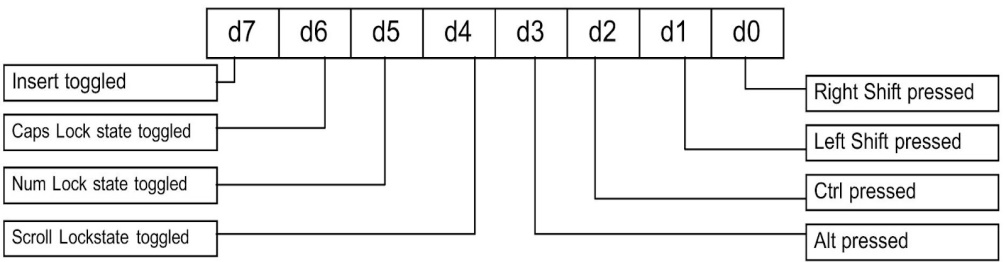
* As INT 16H reads a character from the keyboard buffer, it advances the head pointer, which is held by memory locations 41AH and 41BH.
  + As INT 09 inserts the character into the keyboard buffer, it advances the tail.
  + As INT 16H reads the character from the keyboard buffer it advances the head



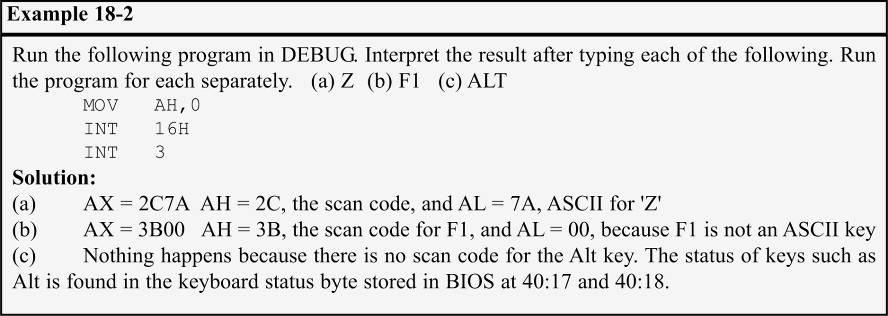
* **keyboard buffer**
* 32 bytes (16 words) of BIOS data memory is set aside, at addresses 40:001EH - 40:003DH.
* Each two consecutive locations are used for a single character.
  + One for the scan code, the other for the ASCII code (if any) of the character.
* When they come to the end of the keyboard buffer, they both wrap around, creating a ring of 16 words where the head is continuously chasing the tail.
* If the buffer is empty, head address equals tail address.
* **Status 2 Byte**



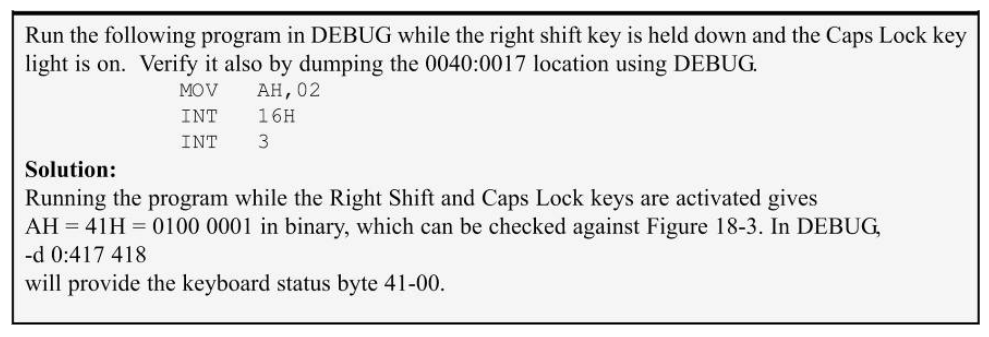
- Location 0040:0017H holds the shift status byte

- Location 0040:0018H holds the second status byte

* INT 16H reads a character from the keyboard
* INT 16H, AH = 0 (read a character)
  + checks the keyboard buffer for a character.
  + If a character is available, it returns its scan code in AH and its ASCII code in AL.
  + If no character is available in the buffer, it waits for a key press and returns it.
  + For characters with no ASCII code, it provides the scan code in AH and AL = 0. Such as F1–F10.

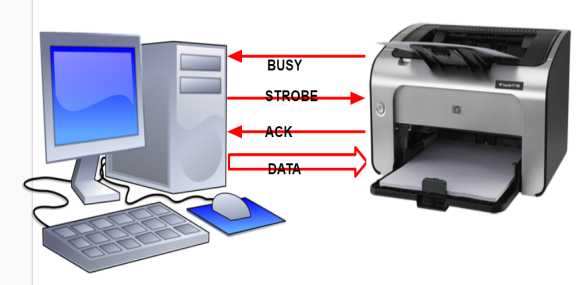


* INT 16H, AH = 01 (find if a character is available) - checks the keyboard buffer for a character.
  + If a character is available, it returns its scan code in AH, its ASCII code in AL, and sets ZF = 0.
  + If no character is available in the buffer, it does not wait for a key press, and simply makes ZF = 1.
* INT 16H, AH = 02 (return current keyboard status byte) - provides keyboard status in register AL.
  + The keyboard status byte (also referred to as the keyboard flag byte) is located in the BIOS data area memory location 0040:0017H.

****

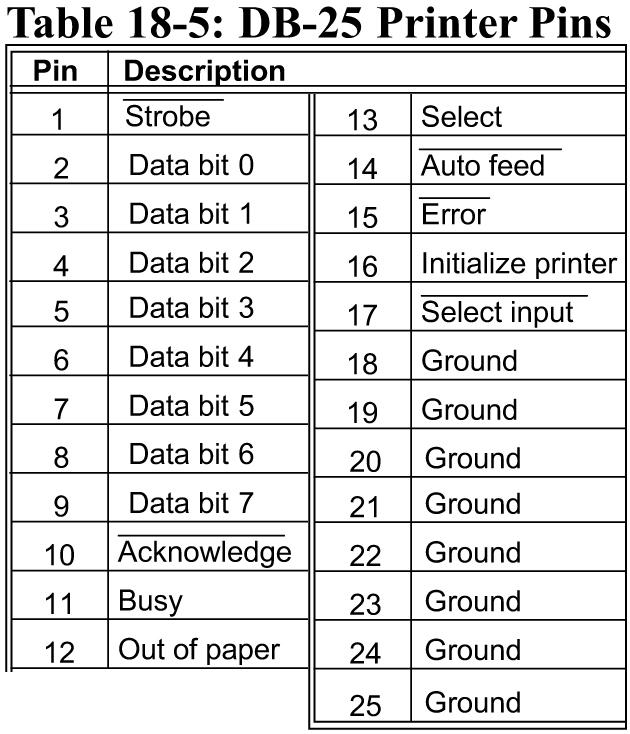
**Lecture 2**

Why parallel ?

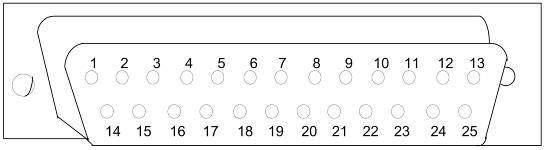
* Some devices are parallel by nature (easy to sent && some program need no long ways)
* Easier to program (No need for framing)
* Error handling (Due to short Distance)

Before connect with printer we send Stoped.io

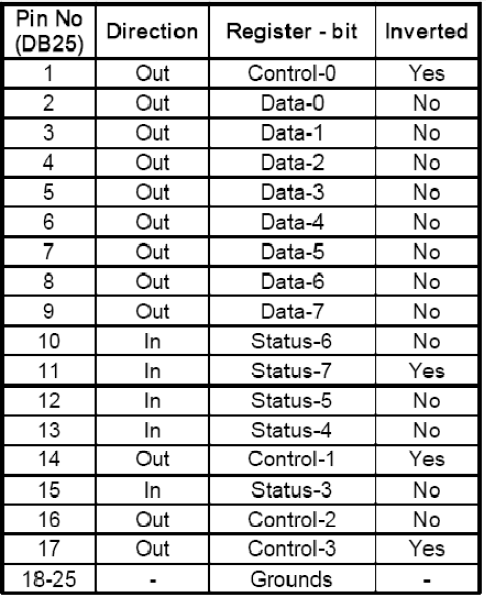
* Printer send ( busy ) so I can’t send any data
* Then wen Printer stop sending Busy Computer send Data & STORBE signal
  + STORBE : used to tell printer that this data aren’t garbage
* Printer received data then send ACK to confirm printer received Data

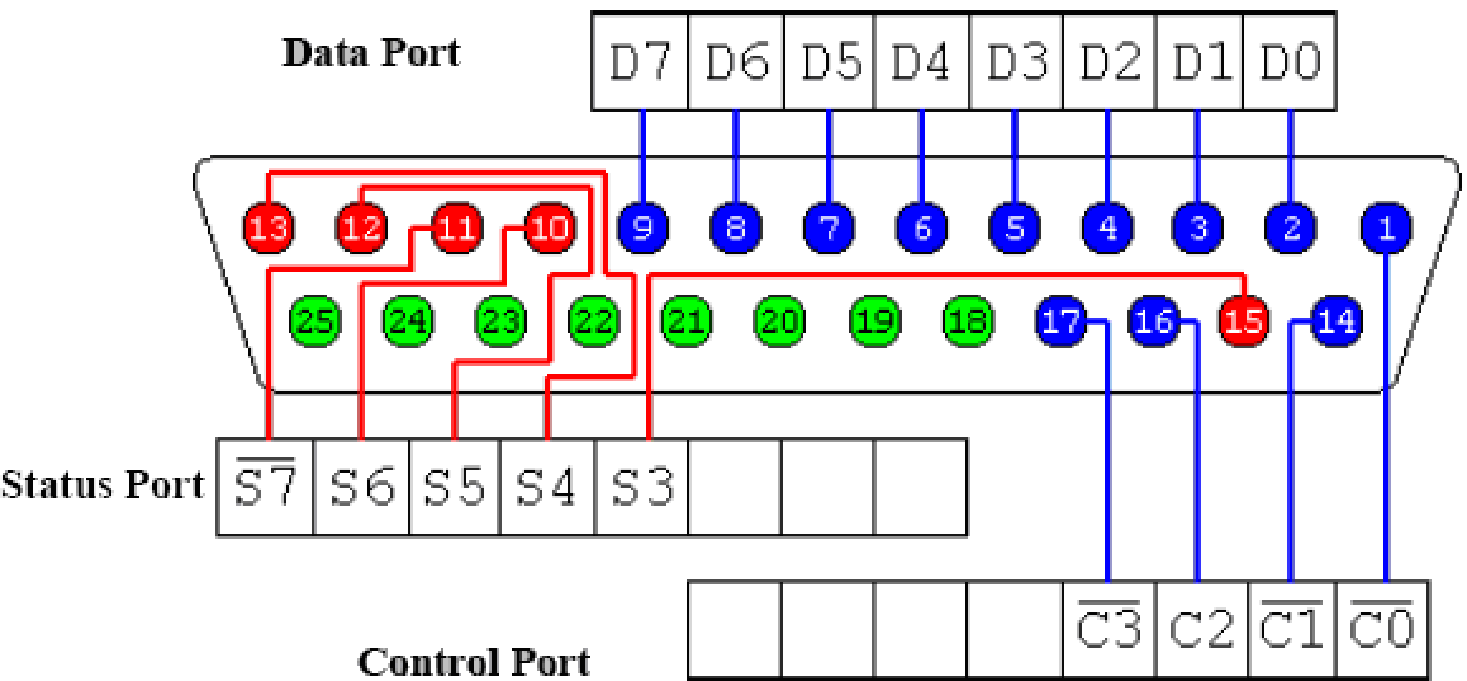


Parallel port Between Pc & Printer

* from 18 to 25 we use it as Ground because we don’t want to

over load one ground for stability of signal

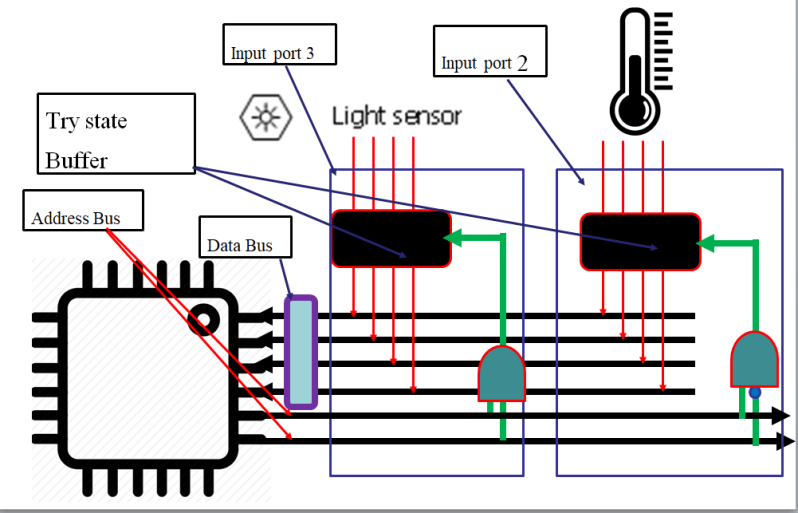
* Control port is out pins because PC can control printer with them
* status port used for know the status of printer
* Data port can be used as in or out to send or receive data from printer
* **MOV DX,378H (387H is the address of parallel port)**
* **OUT DX,00011101b**



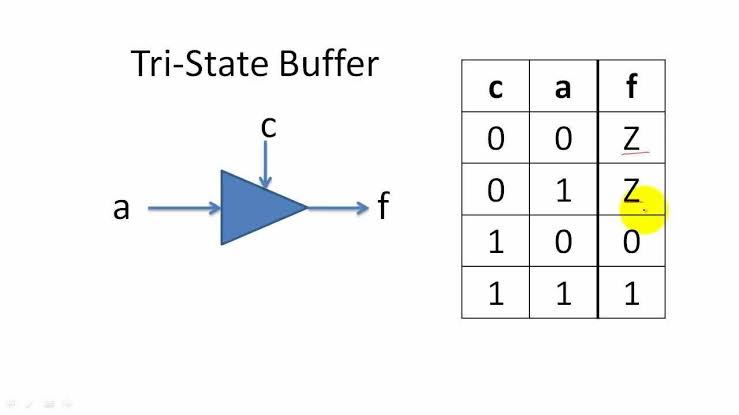
* The **data lines** carry data, sent by the PC to the printer.
* The **printer status signals** indicate the printer status  
  at any given time.
* **Printer control signals** tell the printer what to do.
* **Ground signals** provide an individual ground return  
  line for each data line, and certain control & status lines.
* **PE** (pin 12) - used by the printer to indicate it is out of paper.
* **BUSY** (pin 11) - is *high* if the printer is not ready to accept a new character.
  + This pin is *high* when the printer is off line or when it is printing & can’t accept any data.
  + As long as this pin is high, the PC will not transfer data to the printer.
* **ACKNLG** (pin 10) - used to acknowledge printer receipt of data & that it can accept a new character.
* **ERROR** (pin 15) - normally high output, activated (*goes low*) in conditions such as out of paper, off line state, or jammed print head, in which the printer cannot print.
* **SLCT** (pin 13) is *active-high*, from printer to PC when the printer is on & online.
* **STROBE** (pin 1) and **ACKNLG** are the most widely used signals among control and status pins.
  + When the PC presents a character to the data pins of the printer, it activates the STROBE pin of the printer. Telling it that there is a byte sitting at the data pins.
* **INIT** (pin 16) - an input to the printer, normally *high*.
  + When it is activated (*active-low*) it resets the printer.

I/O Programming

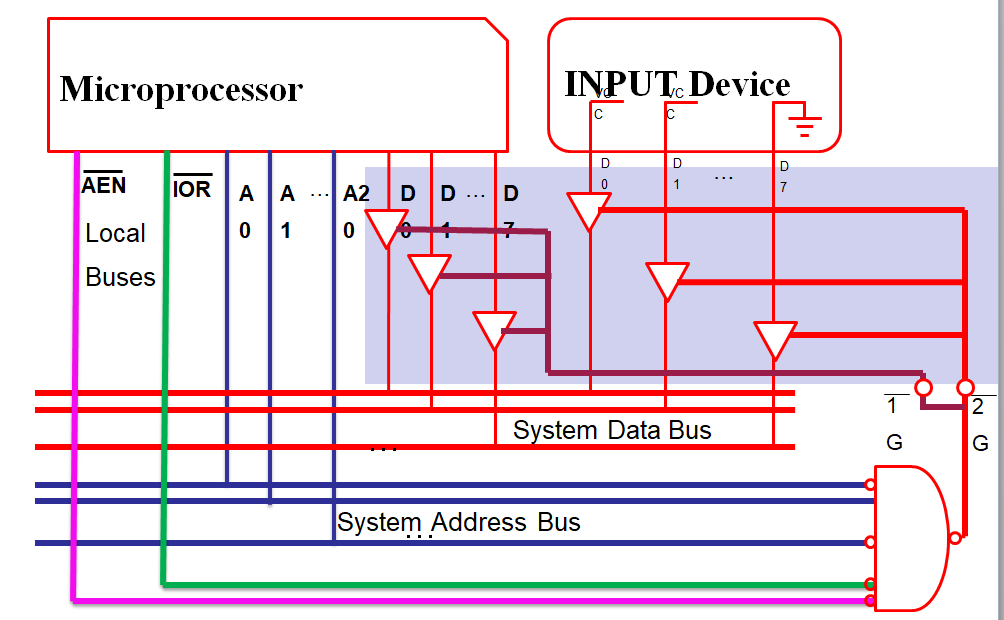
Why I need port ?

* Because connecting many sensors on pc will confuse processor and distribute data So
* We need to add extra controlling part to Connect/disconnect(Z not same as 0) sensors (Tri state Buffer)
* We could use extra enable/disable pin (Address Bus)
*  Input port x detected by the signal on address bus that active it

Tri-state Buffer

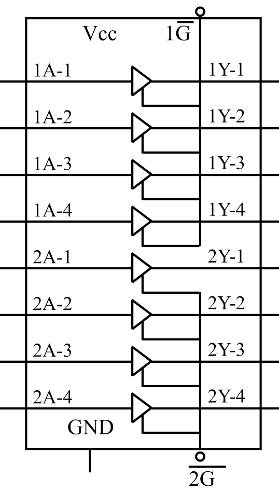


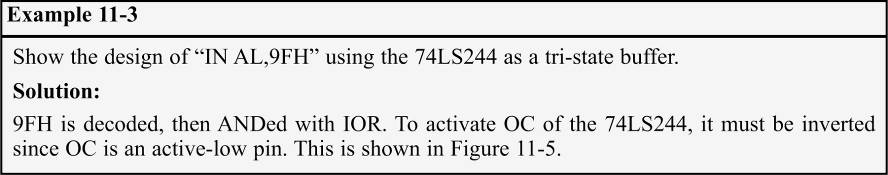
**Getting processor in img**

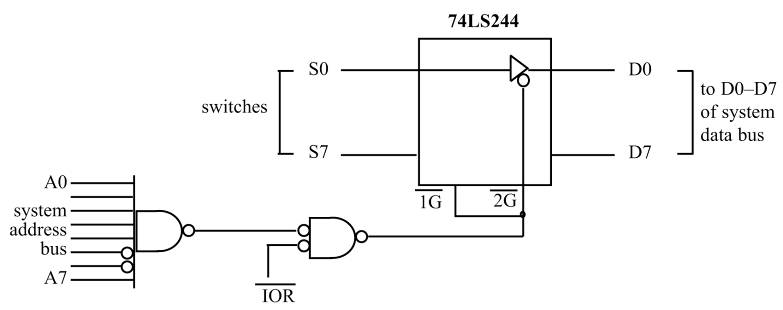
* we connect device with Tri-state Buffer to avoid overwriting
* IOR\* used to detect if I send this address to memory or I/O device (its inverted)
* AEN\* (Address enable) used to identify that processor used Buses (inverted)
* Processor also need Tri-state Buffer to detect if he want this data or other device to avoid garbage on databuses

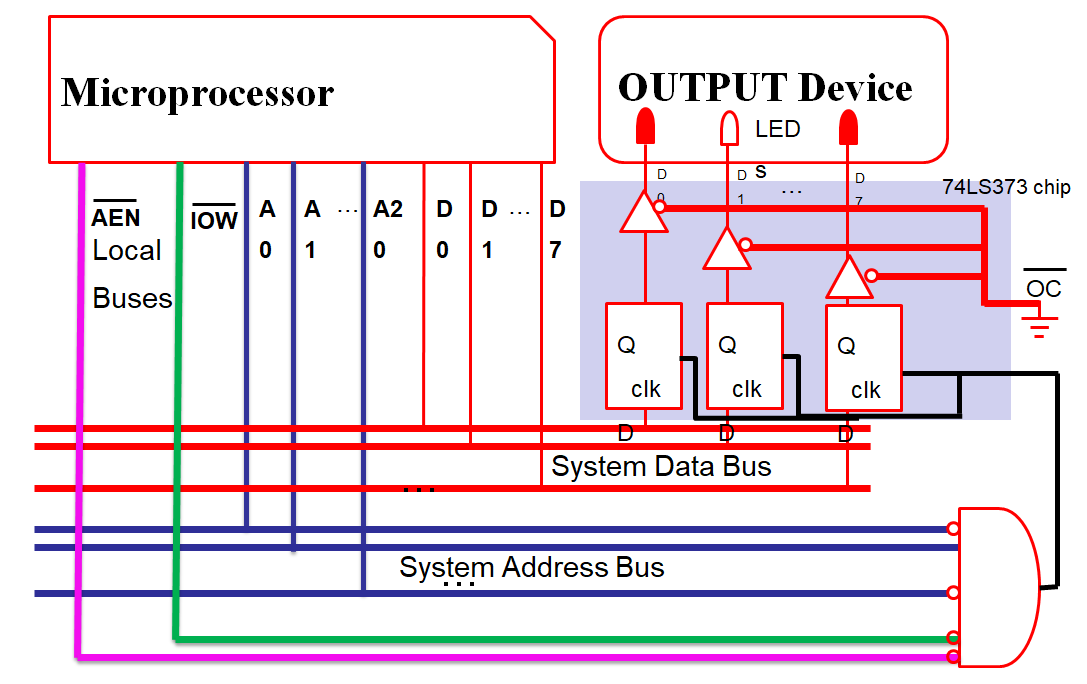
For activation of moicroproccesor and Input device

* Data from a data bus, must come in through a three-state buffer—referred to as *tri-stated*.
  + Simple input ports we
* Since **1G** & **2G** each control only 4 bits of 74LS244, they *both* must be activated for 8-bit input.

ex



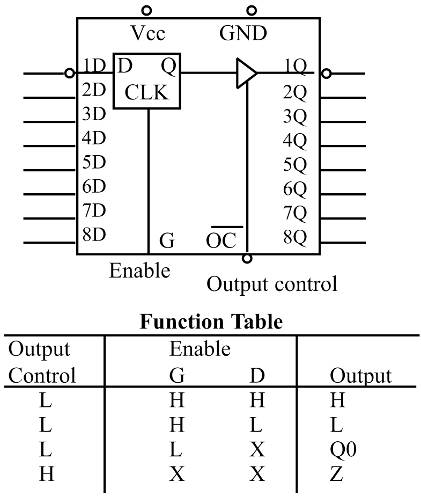


** For Output Device (*Note: IOR is now IOW*)**

* same sa input device but with some different
  + when I but signal on device I want it to keep this signal till change not stay connected all time
  + it should be always ON. Output devices doesn’t change the line status, so I can put 0 to tri-state (intverted to save power of let it vcc)
  + so to save data in device we use latches its clock connected to address bus so when data send to this device it will work and take input to save it

74LS373 chip

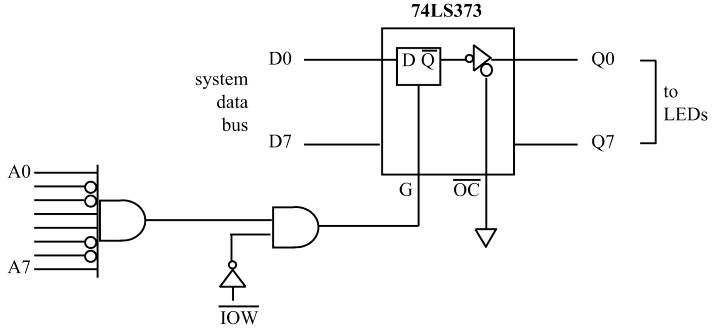
* 74LS373 can be used as a latching system for simple I/O ports.

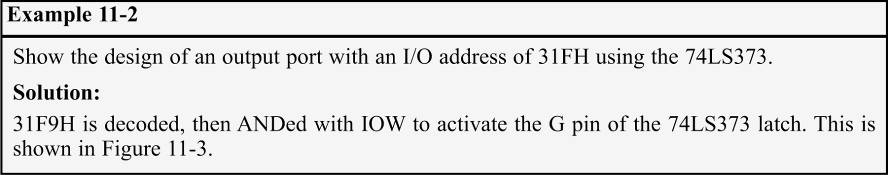


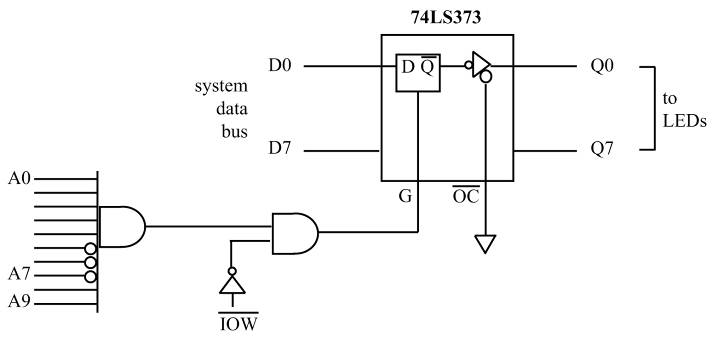
* + Pin **OC** must be grounded.

EX

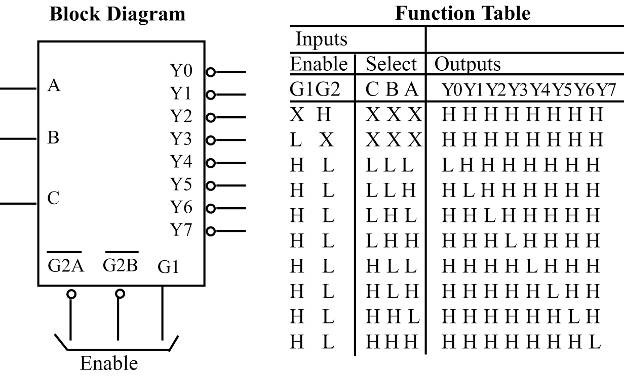
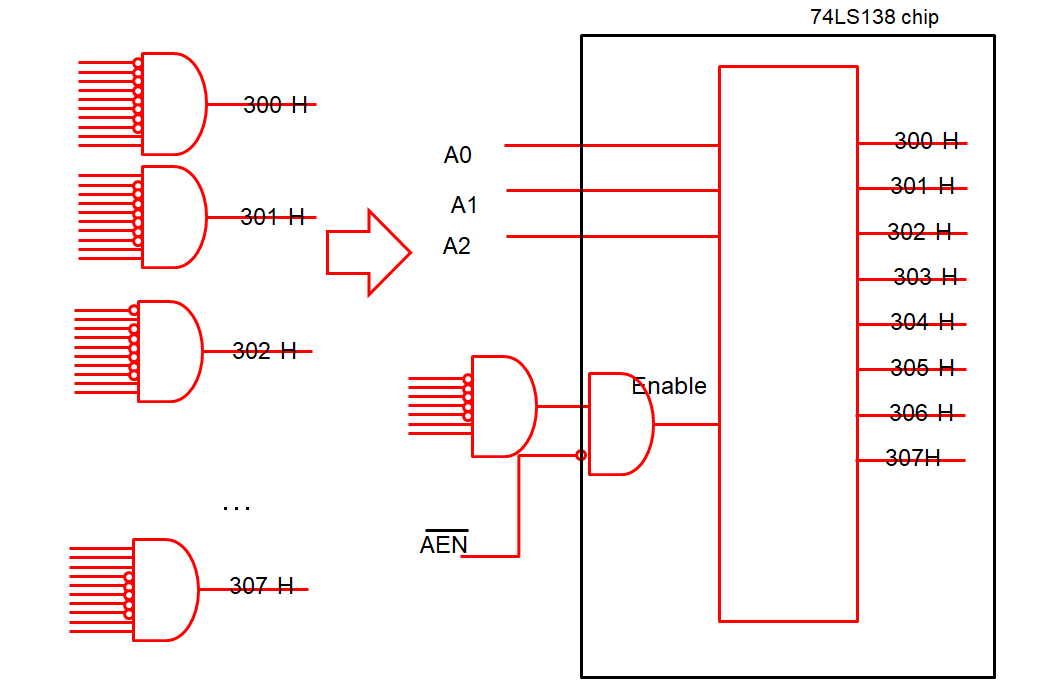
Design a decoding circuit for “OUT 99H, AL”.

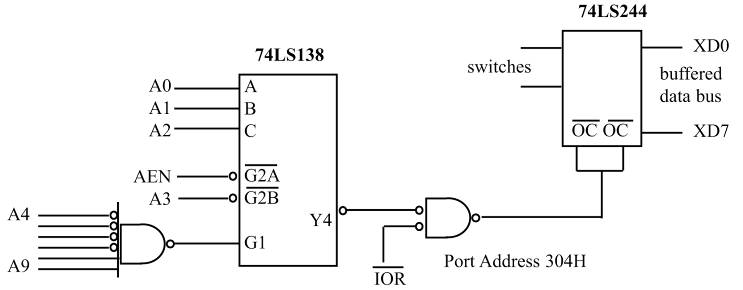






So should we make decoder for each device ?!! 74LS138 chip





Lets have a big look now

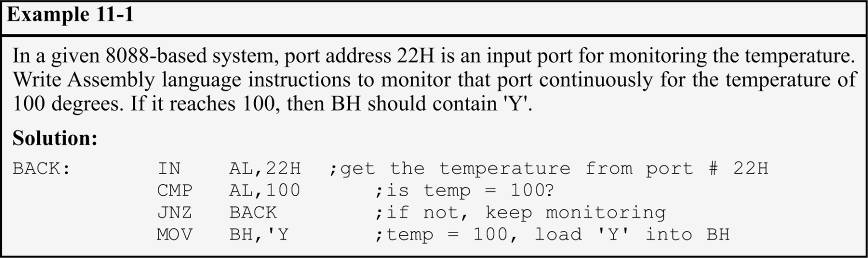
* 74LS138 showing I/O address decoding for an input port located at address **304H.**
  + Each Y output controls a single I/O device.
  + **Y4** output, together with the signal at **IOR**, controls
* With 20 address bus we can have about 106 with ranges

**Absolute vs. linear address decoding.**

|  |  |
| --- | --- |
| **Absolute address decoding** | **Linear address decoding** |
|  |  |
| Motorola | Intel |
| expensive | cheaper |
| all address lines are decoded | Some address only decoded |
| No aliases, can support 106 device | creates aliases, the same port with multiple addresses. |
| NO gaps | Cause gaps |

|  |  |
| --- | --- |
| **Memory-mapped IO** | **Port-mapped IO / isolated I/O** |
| Motorola | Intel |
| Absolute address decoding | Linear address decoding |
|  | **MEMR and MEMW** |
|  |  |
| Same address bus to address memory and I/O devices | Different address spaces for memory and I/O devices |
| Access to the I/O devices using regular instructions | Uses a special class of CPU instructions to access I/O devices |
| Most widely used I/O method | x86 Intel microprocessors - IN and OUT instructions |

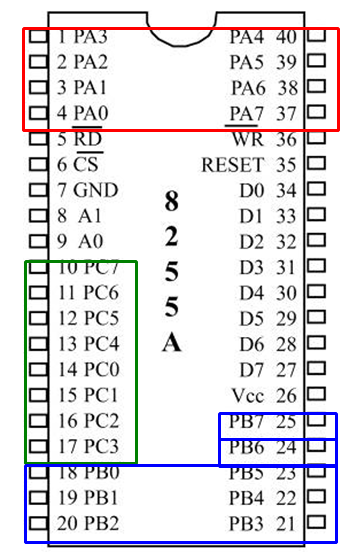
Ex

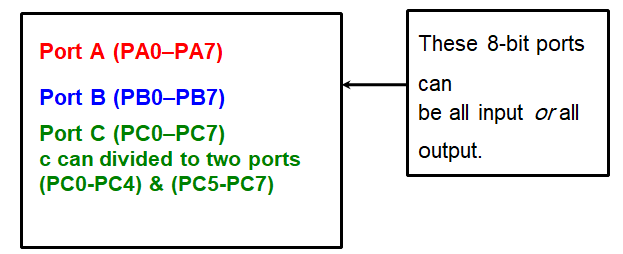


**port 61H and time delay generation**

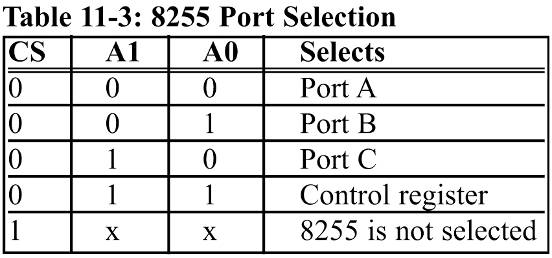
* Port 61H, a widely used port, can be used to generate a time delay.
* I/O port 61H has eight bits (D0–D7), of which D4 is of particular interest.
  + In all 286 & higher PCs, D4 of port 61H changes its state, indefinitely every 15.085 microseconds (ms).
    - Low for 15.085 ms.
    - High for the same amount of time.
    - Low again.

**PROGRAMMING & INTERFACING THE 8255**

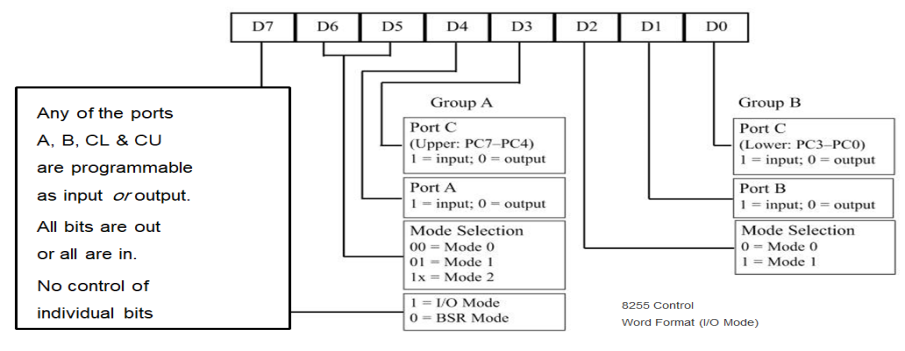
* The 8255 is a widely used
  + Data come across D0-D7
  + It has three separately accessible programmed ports, A, B & C.
  + Each port can be programmed to be input or output.
  + Ports can also be changed dynamically.



* **RD** and **WR** - *active-low* 8255 control signal inputs.
  + If the 8255 is using peripheral I/O, **IOR** & **IOW** of the system bus are connected to these two pins.
  + If memory-mapped I/O, **MEMR** & **MEMW** activate them.
* **RESET** - an active-high signal input into the 8255, used to clear the control register.
  + All ports are initialized as input ports.
* **A0**, **A1**, and **CS**
  + **CS** (chip select) selects the entire chip.
  + Address pins **A0** and **A1** select the *specific port* within the 8255.



* 8255 ports can be programmed in various modes.
  + The *simple I/O mode*, Mode 0, is most widely used.

**AT D7 = 0 (BSR Mode)**

