# UNIT 2

# EMBEDDED C PROGRAMMING

# MEMORY INTERFACING

# MEMORY INTERFACING

- Interfacing and timing diagrams for memory interfacing
- Memory address Decoding
- Interfacing Examples
- Accessing External data memory in 8051C

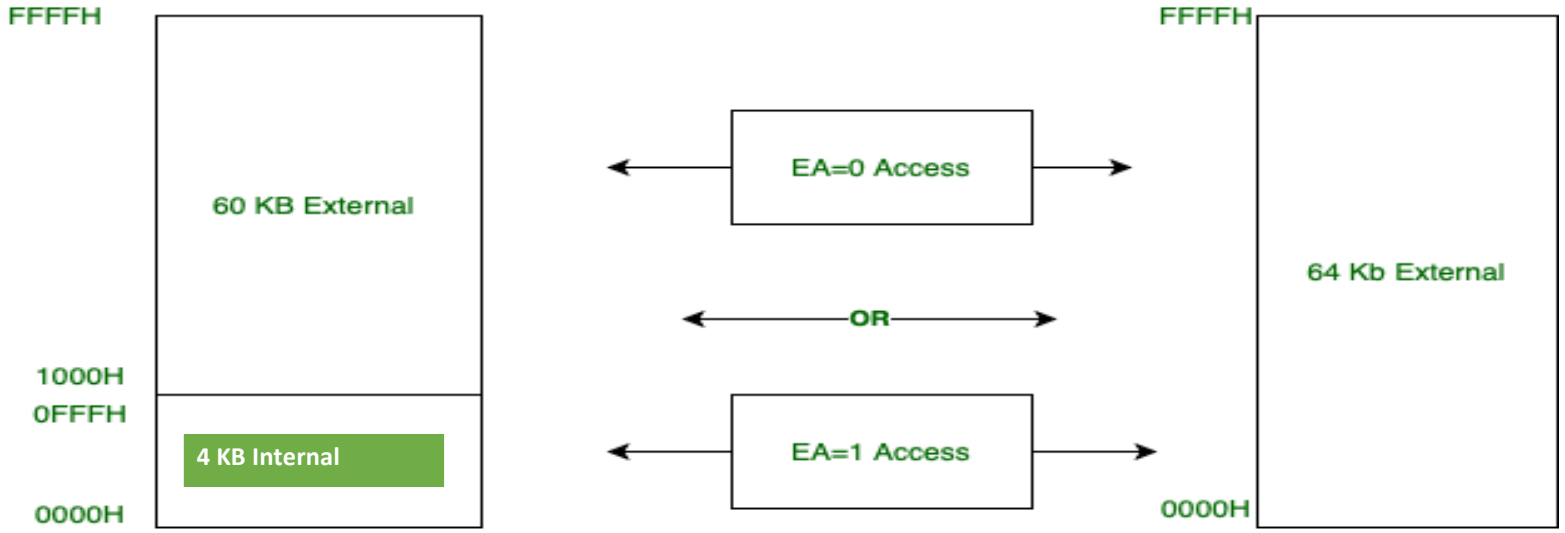
# Interfacing and timing diagrams for memory interfacing

- 8051 has internal data and code memory-limited memory capacity
- Memory capacity may not be sufficient for some appln
- In this cases-connect external ROM/EPROM and RAM to 8051 –increase memory capacity
- ROM-program memory  RAM-Data memory

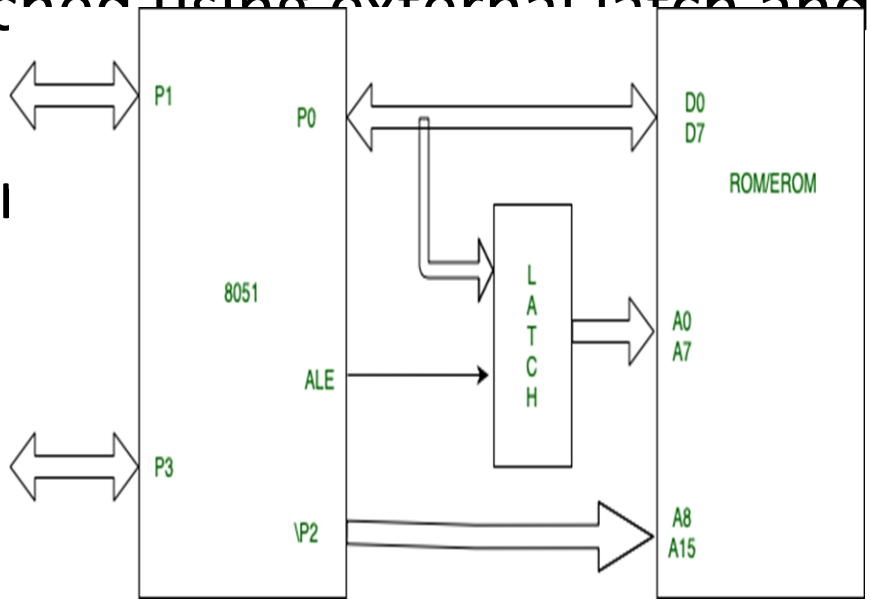
1. External Program Memory
2. External Data Memory
3. Important points to remember in accessing External memory

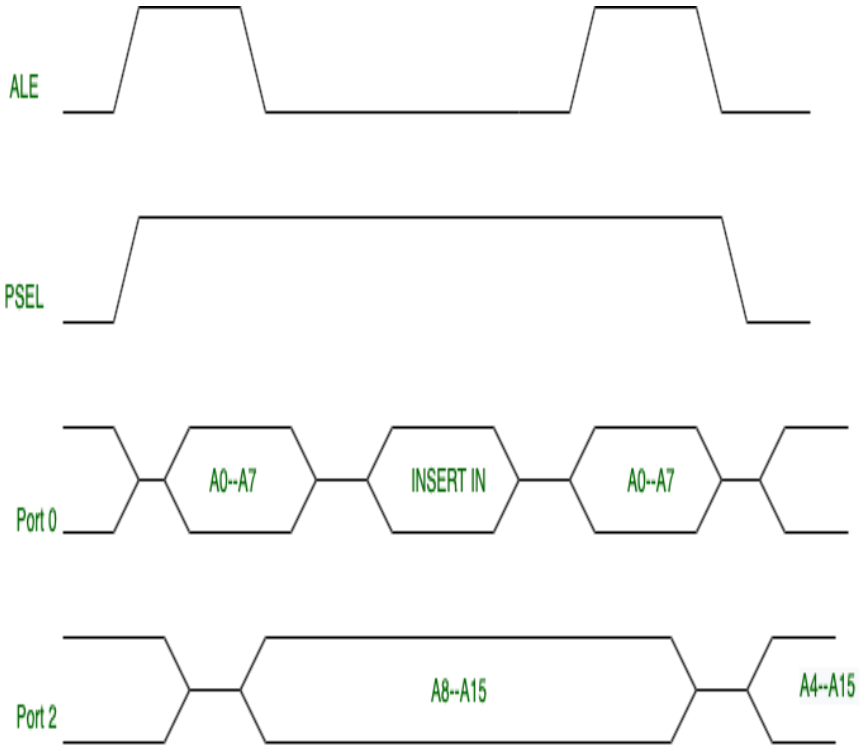
# External Program Memory

- $\overline{EA}$ pin connected to VCC-Program fetches the address 0000H-0FFFH- directed to the internal ROM ,1000H to FFFFH to external ROM/EPROM
- $\overline{EA}$ pin- gnd- all address fetched by program are directed to external ROM/EPROM.
- $\overline{PSEN}$- used to activate the output enable signal of external ROM/EPROM

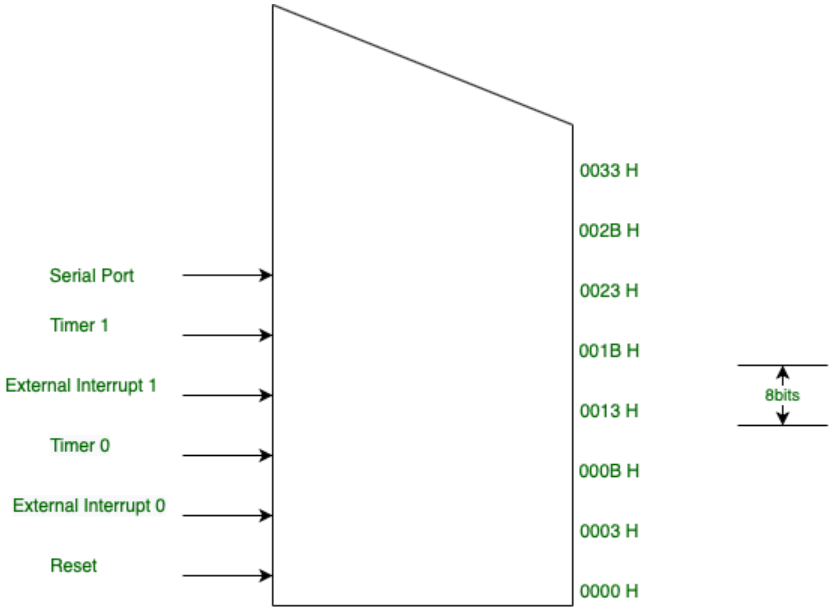


- Port 0-multiplexed address/bus
- It gives lower order 8 bit address in the initial T cycle and later it is used as a data bus
- 8 bit address is latched using external latch and ALE signal generated
- Port 2-higher order






timing waveform for external data memory write cycle



- If an interrupt is to be used, the operation routine for it must be in the same place as the interrupt.
- If the interrupt isn't used, the service location

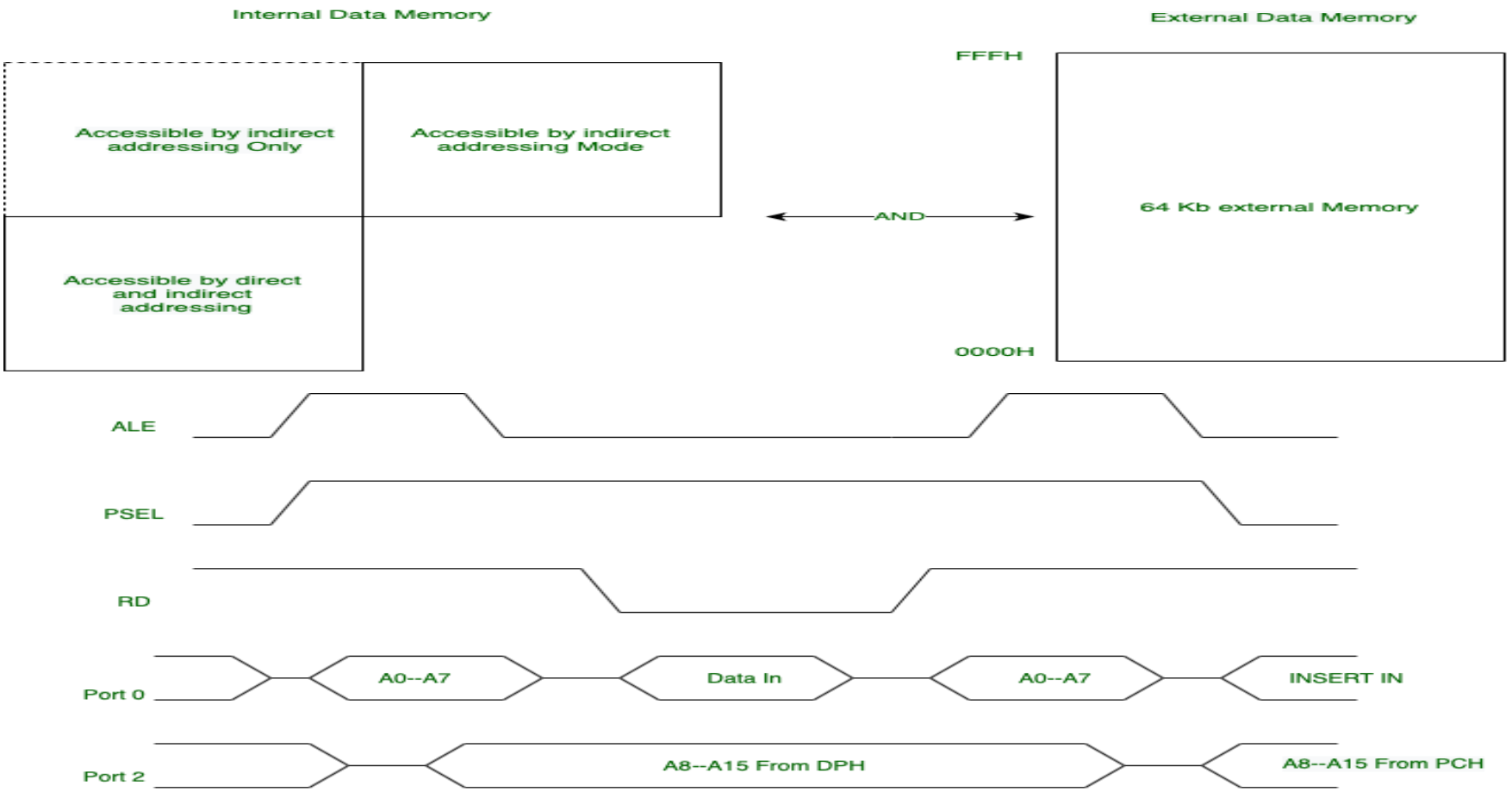
**Instructions to Access External ROM / Program Memory :**

This table is explaining the instructions to access external ROM/program memory.

| Mnemonic | Operation |
|---|---|
| MOVC A, @ A+DPTR | Copy the contents of the external ROM address formed by adding A and the DPTR, to A |
| MOVC A, @ A + PC | This operation will do copy This operation contents of the external ROM address formed by adding A and the PC, to A. |

# External Data Memory

- Up to 64 k-bytes of additional data memory can be addressed by the 8051.
- The external data memory is accessed using the "MOVX" instruction.
- The 8051's internal data memory is split into three sections:
- Lower 128 bytes, Upper 128 bytes, and SFRs.
- While they are physically distinct bodies, the upper addresses and SFRs share the same block of address space, 80H by FFH.
- The upper address space is only accessible via indirect addressing, and SFRs are only accessible via direct addressing, as seen in Higher address space, on the other hand, can be reached using either direct or indirect addressing




timing waveform for external data memory read cycle

**Instructions to Access External Data Memory :**

The Table explains the instruction to access external data memory.

| Mnemonic | Operation |
|---|---|
| MOVX A, @Rp | In this operation, it will copy the contents of the external address in Rp to A. |
| MOVX A. @DPTR | Copy the contents of the external address in DPTR to A. |
| MOVX @Rp. A | Copy data from A to the external address in Rp |
| MOVX DPTR, A | Copy data from A to the external address in DPTR. |

# Important Points

- In the Case of accessing external memory, All external data moves with external RAM or ROM involve the A register.
- While accessing external memory, R can address 256 bytes and DPTR can address 64 k-bytes
- MOV X instruction is used to access external RAM or I/O addresses.

# Memory address Decoding

- Read/write memories-array of registers
- Registers-Unique address
- Size of the memory is NXM
- N-No of registers
- M-Word length

## RAM

- Memory chip has 11 address lines
- One chip select($\overline{CS}$) and 2 control lines
- Read ($\overline{RD}$)-enable output buffer
- Write($\overline{WR}$)-enable input buffer
- Internal decoder-decode the address lines

## EPROM

- 4K Registers(4096)
- 12 address lines
- One chip select , one read control signal

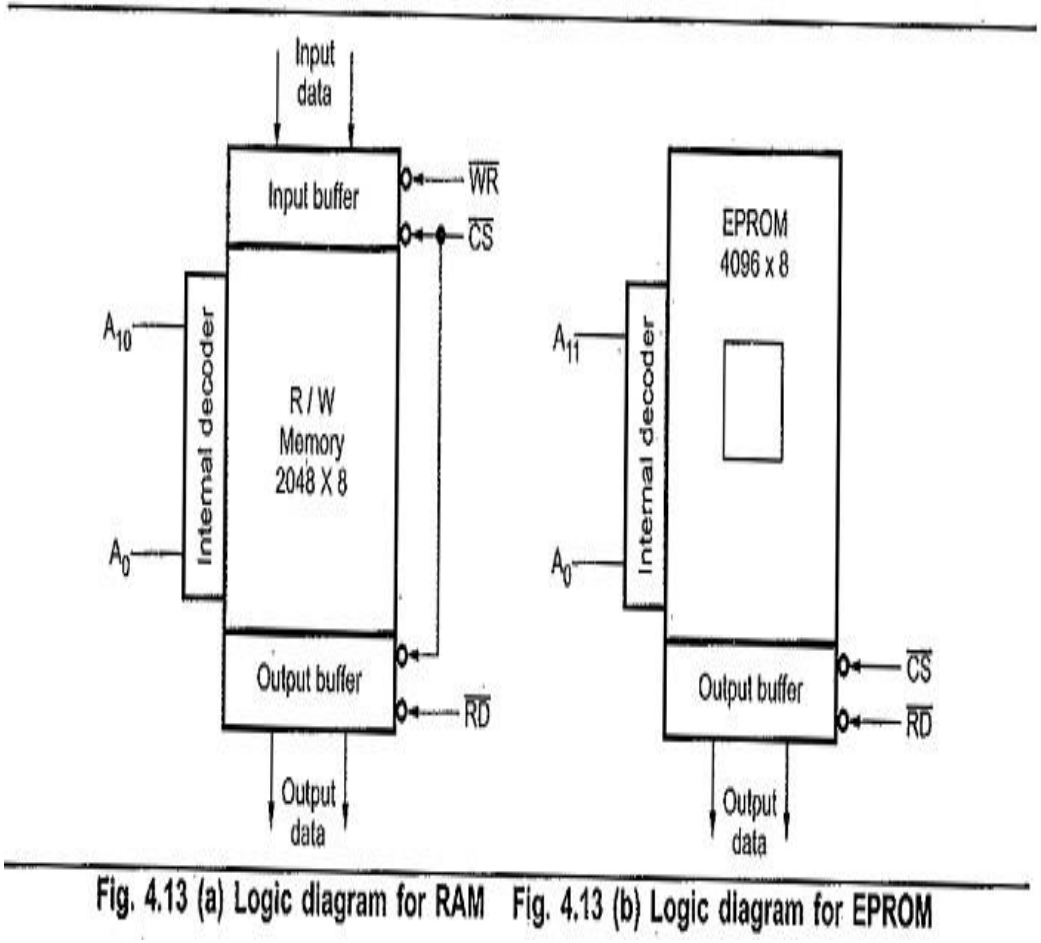



Fig. 4.13 (a) Logic diagram for RAM Fig. 4.13 (b) Logic diagram for EPROM

It requires

- To select the chip
- Identify the registers
- Enable the appropriate buffer

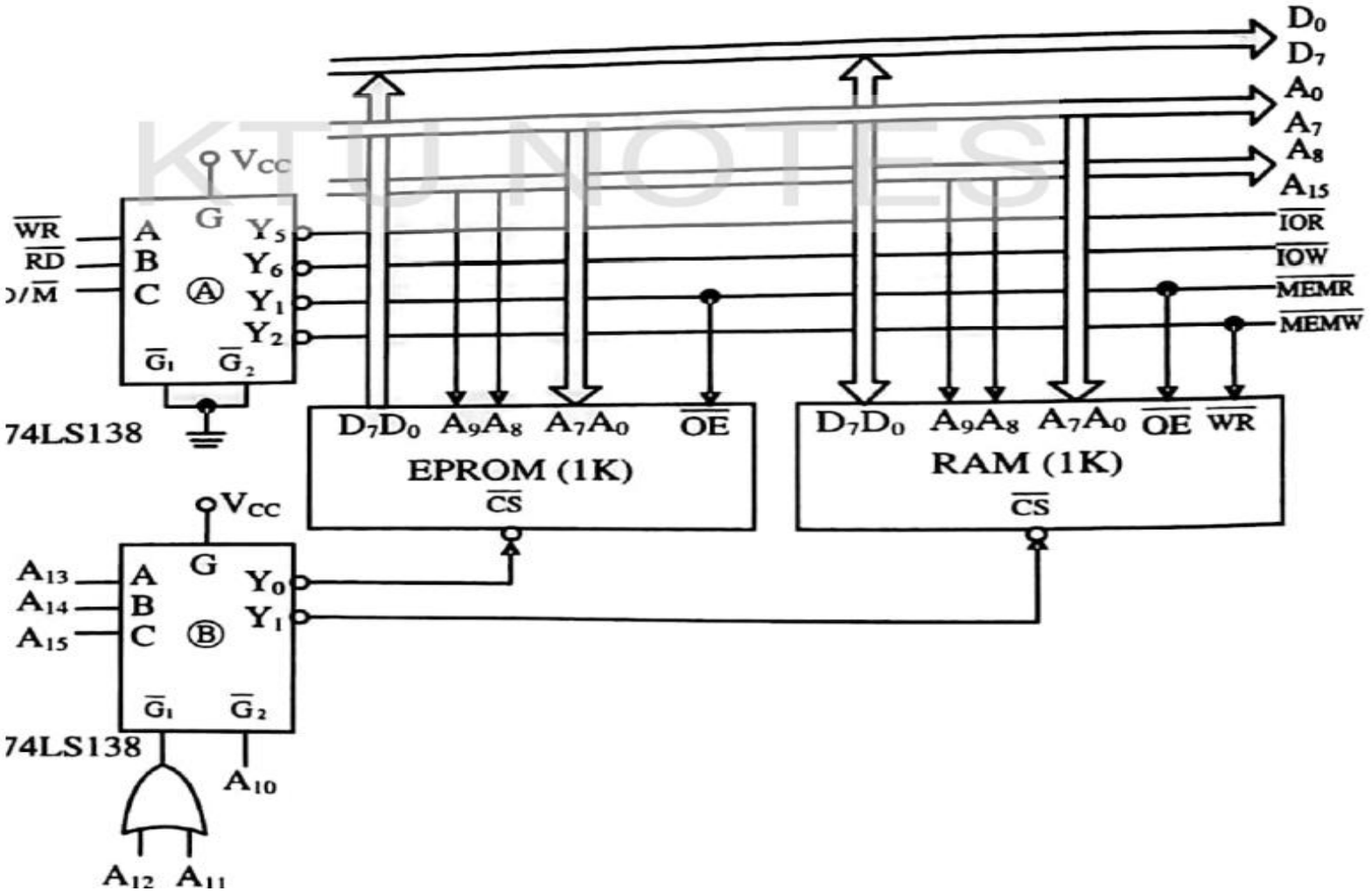
Address decoding techniques

- Absoluter/full decoding
- Linear/Partial Decoding

# Absolute Decoding

- All the higher address lines are decoded to select the memory chip, and the memory chip is chosen only for the logic levels defined in these high-order address lines and no other logic levels will select the chip.
- Used in large memory systems
- The decoding in which all available address lines are u lute d

| Memory ICs | $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ | $A_{11}$ | $A_{10}$ | $A_9$ | $A_8$ | $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | Address |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Starting address of EPROM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0000H |
| End address of EPROM | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 03FFH |
| Starting address of RAM | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2000H |
| End address of RAM | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 23FFH |

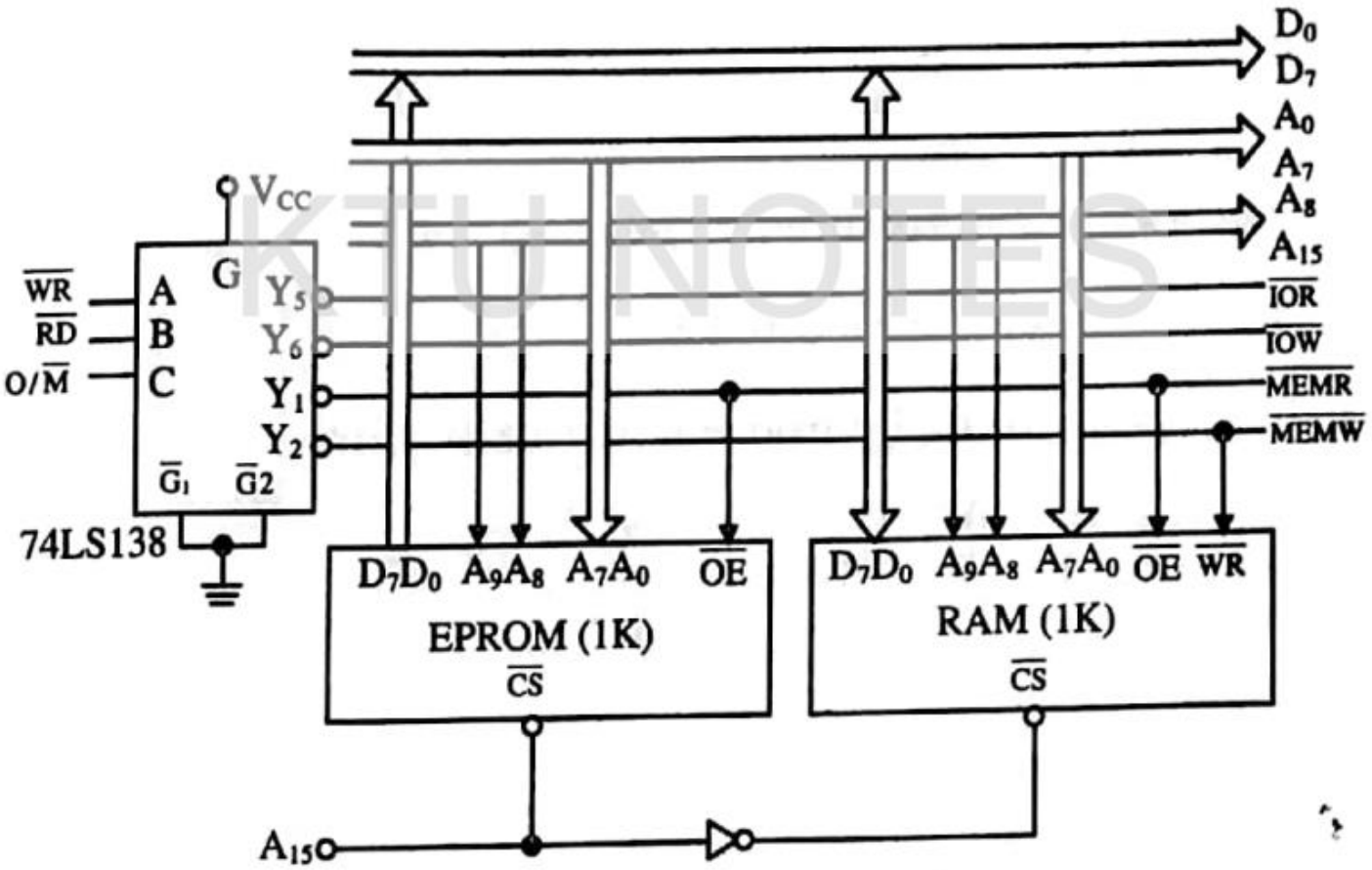

D0
D7
A0
A7
A8
A15
IOR
IOW
MEMR
MEMW
Vcc
G
WR
RD
IO/M
A
B
C
Y5
Y6
Y1
Y2
G1
G2
74LS138
D7D0 A9A8 A7A0 OE
EPROM (1K)
CS
D7D0 A9A8 A7A0 OE WR
RAM (1K)
CS
Vcc
A13
A14
A15
A
B
C
G
Y0
Y1
G1
G2
74LS138
A10
A12
A11

KTU NOTES

# Linear Decoding

- The decoding in which all available address lines are not used for decoding to select a location is called partial decoding technique
- In small systems, the hardware for decoding circuit can be eliminated by using individual high order address lines to select memory chips. This is the partial decoding technique.
- The cost of the decoding circuit can be reduced with this tec
- Dra

| Memory ICs | $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ | $A_{11}$ | $A_{10}$ | $A_9$ | $A_8$ | $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | Address |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Starting address of EPROM | 0 | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0000H |
| End address of EPROM | 0 | x | x | x | x | x | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 03FFH |
| Starting address of RAM | 1 | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8000H |
| End address of RAM | 1 | x | x | x | x | x | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 83FFH |


D0
D7
A0
A7
A8
A15
IOR
IOW
MEMR
MEMW
Vcc
G
WR
RD
O/M
A
B
C
Y5
Y6
Y1
Y2
G1
G2
74LS138
D7D0 A9A8 A7A0 OE
EPROM (1K)
CS
D7D0 A9A8 A7A0 OE WR
RAM (1K)
CS
A15

KTU NOTES

| Full Address Decoding | Partial Address decoding |
|---|---|
| 1. All higher address lines are decoded to select the memory or I/O device. | 1. Few higher address lines are decoded to select the memory or I/O device. |
| 2. More hardware is required to design decoding logic. | 2. Hardware required to design decoding logic is less and sometimes it can be eliminated. |
| 3. Higher cost for decoding circuit. | 3. Less cost for decoding circuit. |
| 4. No Multiple addresses. | 4. It has a advantage of multiple addresses. |
| 5. Used in large systems | 5. Used in small systems |

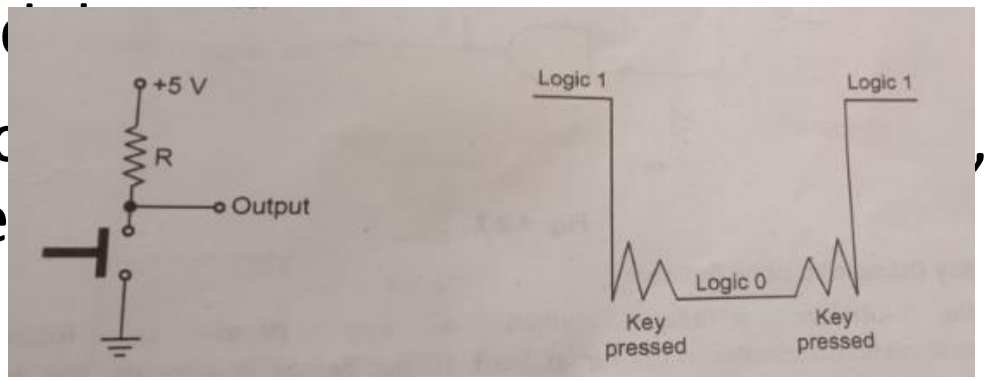
# I/O INTERFACING

# I/0 Interfacing

- Keyboard Interfacing
  1. Key Debounce using Hardware
  2. Key Debounce using Software
  3. Simple keyboard Interface
  4. Matrix keyboard interface
- LED interfacing
  1. Interfacing LED Displays
     - ✓ Static Display
     - ✓ Multiplexed Display
- Multiplexed 7 segment Display interfacing
- LCD interfacing

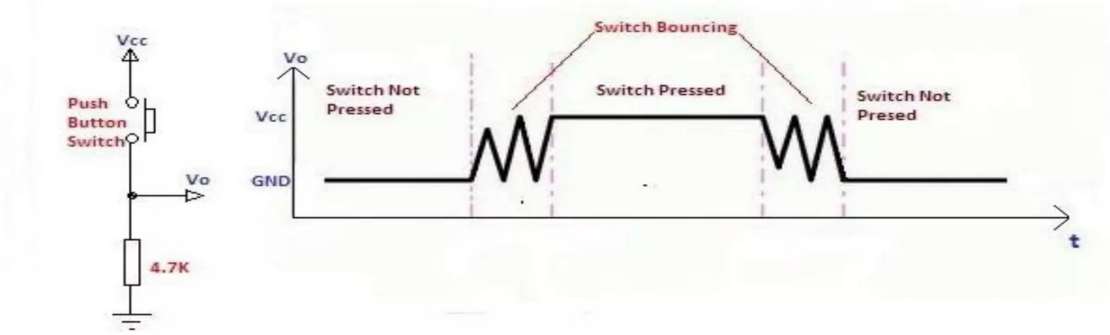
# Keyboard Interfacing

- Push button keys are used
- Reading taken during bouncing period may be faulty
- Mp/mc must wait until the key reach to a steady state-key
- Key bounc , key debounce

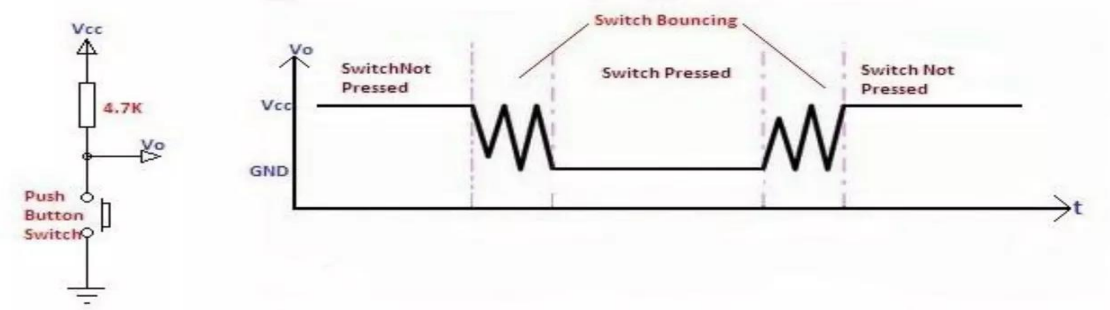


# BOUNCING & DEBOUNCING

Bouncing is the tendency of any two metal contacts in an electronic device to generate multiple signals as the contacts close or open; Debouncing is any kind of hardware device or software that ensures that only a single signal will be acted upon for a single opening or closing of a contact.




*Switch Bouncing in Pull Down Connection*



# Key Debounce using hardware

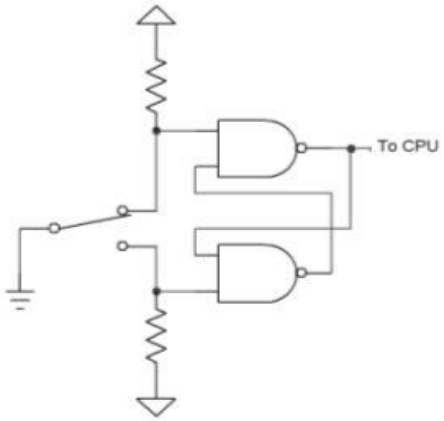
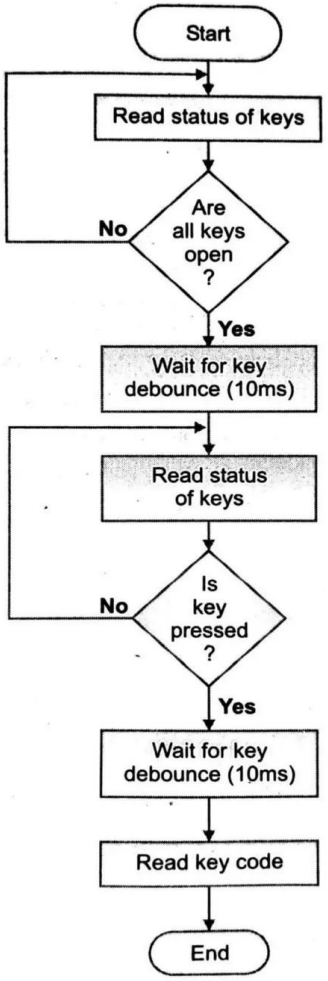



*Figure 1: The SR debouncer*

| Key Position | a | b | Y | c | d | Ȳ |
|---|---|---|---|---|---|---|
| A | 0 | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 1 | 0 | 0 | 0 | 1 |
| A and B | 1 | Ȳ | NC | Y | 1 | NC |

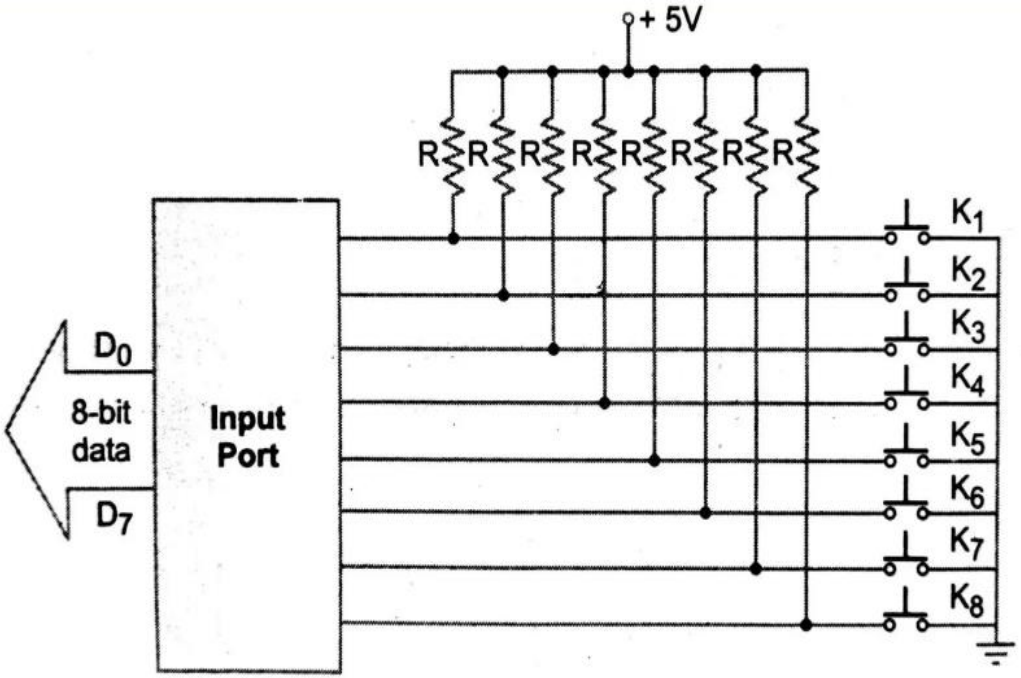



Flowchart of Key Input with Debounce

# KEY DEBOUNCE USING SOFTWARE

In the software technique, when a key press is found, the microprocessor waits for atleast 10ms before it accepts the key as an input.

# Simple Keyboard Interface




Simple Keyboard Interface

8 Keys are individually connected to specific pins of port 1
Each port pin-status of key connected to that pin
When port pin is logic 1- key open-otherwise key is closed

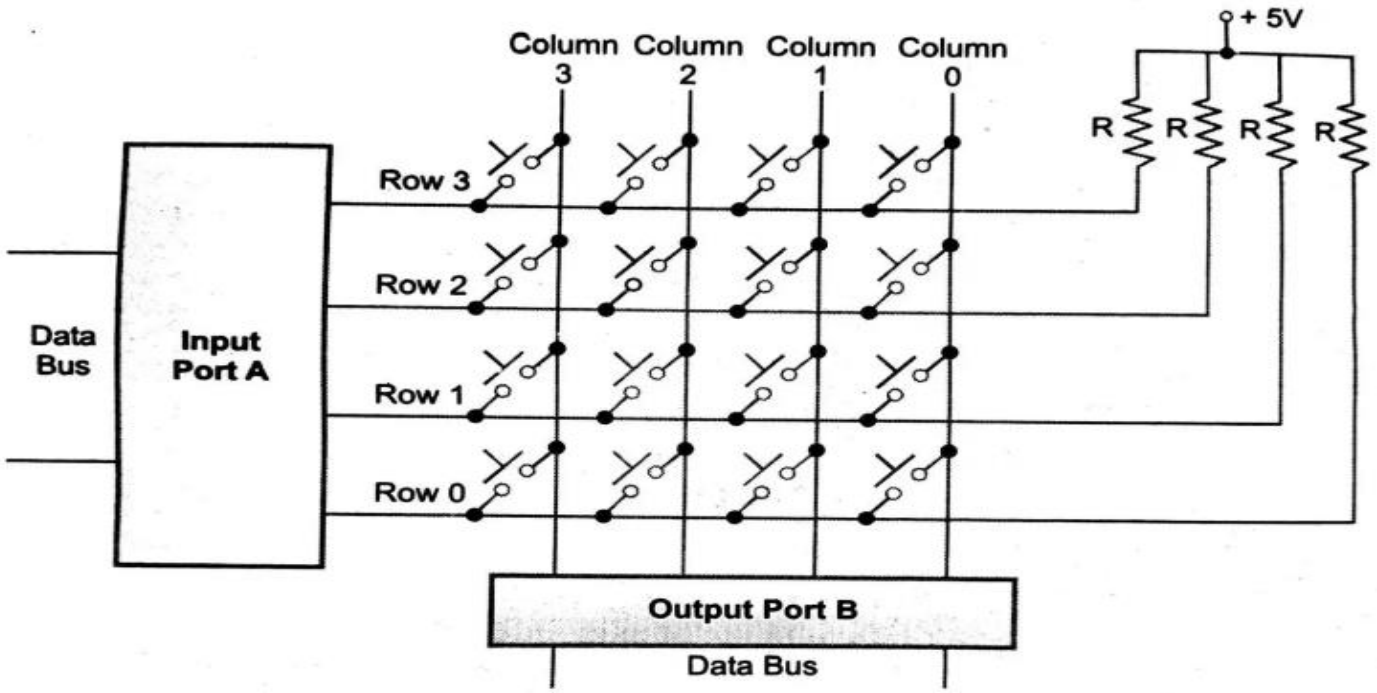
# Program

```
START   :   IN IN_PORT                ;  Read key status
            CPI FFH                   ;  check if keys are open
            JNZ START                 ;  if no, go to start otherwise
                                      ;  continue
            CALL DEBOUNCE_DELAY       ;  call debounce delay
AGAIN   :   IN IN_PORT                ;  Read key status
            CPI  FFH                  ;  check if any key is pressed
            JZ   AGAIN                ;  if no, goto AGAIN : otherwise
                                      ;  continue
            CALL DEBOUNCE_DELAY       ;  call debounce delay
            IN IN_PORT                ;  Get key code
            RET                       ;  Return from subroutine
```

# Key Code

| Key | Keycode | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
| $K_1$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| $K_2$ | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| $K_3$ | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| $K_4$ | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| $K_5$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| $K_6$ | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| $K_7$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| $K_8$ | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

# MATRIX KEYBOARD INTERFACE

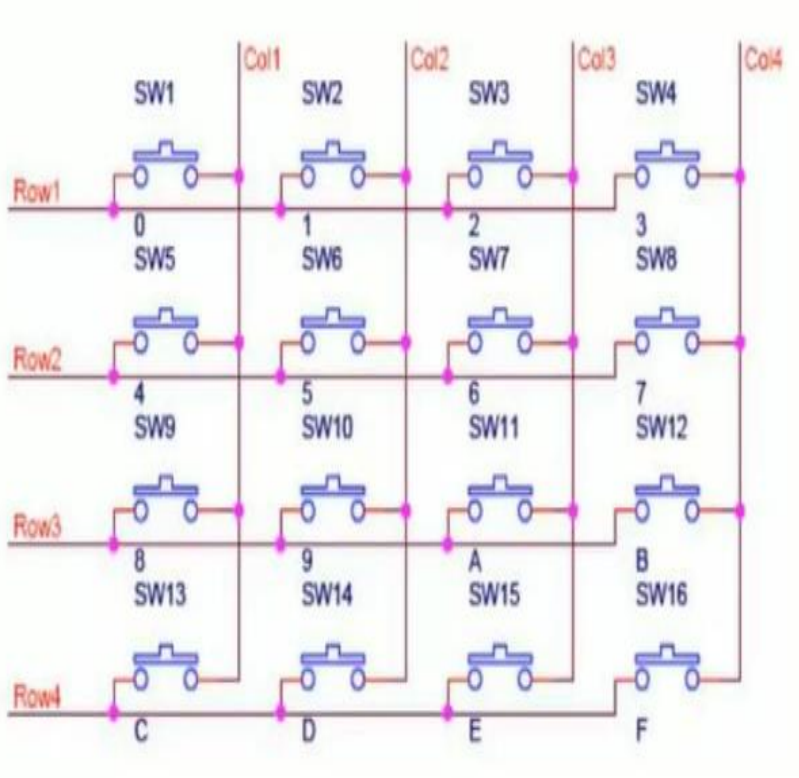



Matrix Keyboard Connections

# Introduction

- Keyboards are organized in a matrix of rows and columns
- The CPU accesses both rows and columns through ports
- When a key is pressed, a row and a column make a conta ct
- Otherwise, there is no connection between rows and columns

- A 4 X 4 matrix keypad is called so because,
  - ✓ it is organized in matrix structure
  - ✓ It has 4X4=16 switches (i.e. push buttons)
- It can be used to give multiple inputs
- Each input can have some significance.
  - ✓ If we use two ports of microcontroller, we can connect 8X8 keypad. This can be used to give 64 inputs, which would have been only 16, if we connect switches directly.
- A matrix keypad is
  - ✓ Easy to interface
  - ✓ Easy to procure




- 16 switches
- Each row & column has a switch, in between, strategically placed such that
    - Each switch has the capability, if presses, to short the particular row and column(form a path)
- The other paths would remain open

- Requires 2 ports
- Input port-rows-returned lines
- Output port-column-scan lines
- WKT all keys are open-row & column do not have any connection
- When key is pressed-corresponding work in row & column
- Output line of column low-corresponding row line low
- Data sent on the output port and input code received from the input port

Check 1

Whether any key is pressed or not

- Make all column lines 0 by sending low on all output lines. Activates all keys in the keyboard matrix
- Read the status of return lines, if the status of all lines is logic high, key is not pressed ; otherwise key is pressed

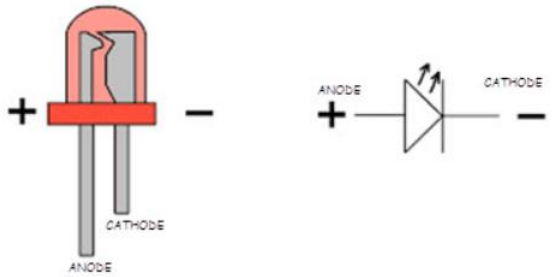
Check 2

- Activate keys from any one column & any 1 column line 0
- Read the status of return lines

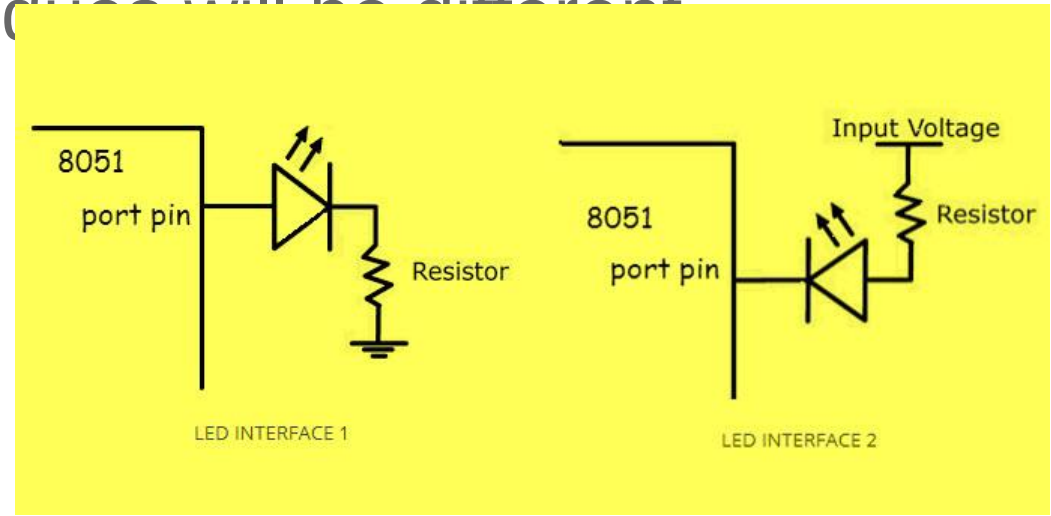
- Simple key board interface-one i/p line is required to interface one keys-number will increase with number of keys
- Large number keys-interface-not suitable
- To reduce the number of connection keys-arranged in matrix form
- 16 keys are arrange in 4 rows and columns
- Keys open-row and column-do not have any connection
- Key is pressed-it shorts corresponding one row and one column

# LED INTERFACING

- LED  is a semiconductor device used in many electronic devices, mostly used for signal transmission /power indication purposes.
- It is very cheaply and easily available in a variety of shape, color, and size. The LEDs are also used for design message display boards and traffic control signal lights etc
- It has two terminals positive and n



- There are two ways which we can interface LED to the Microcontroller 8051.
- But the connections and programming techniques will be different



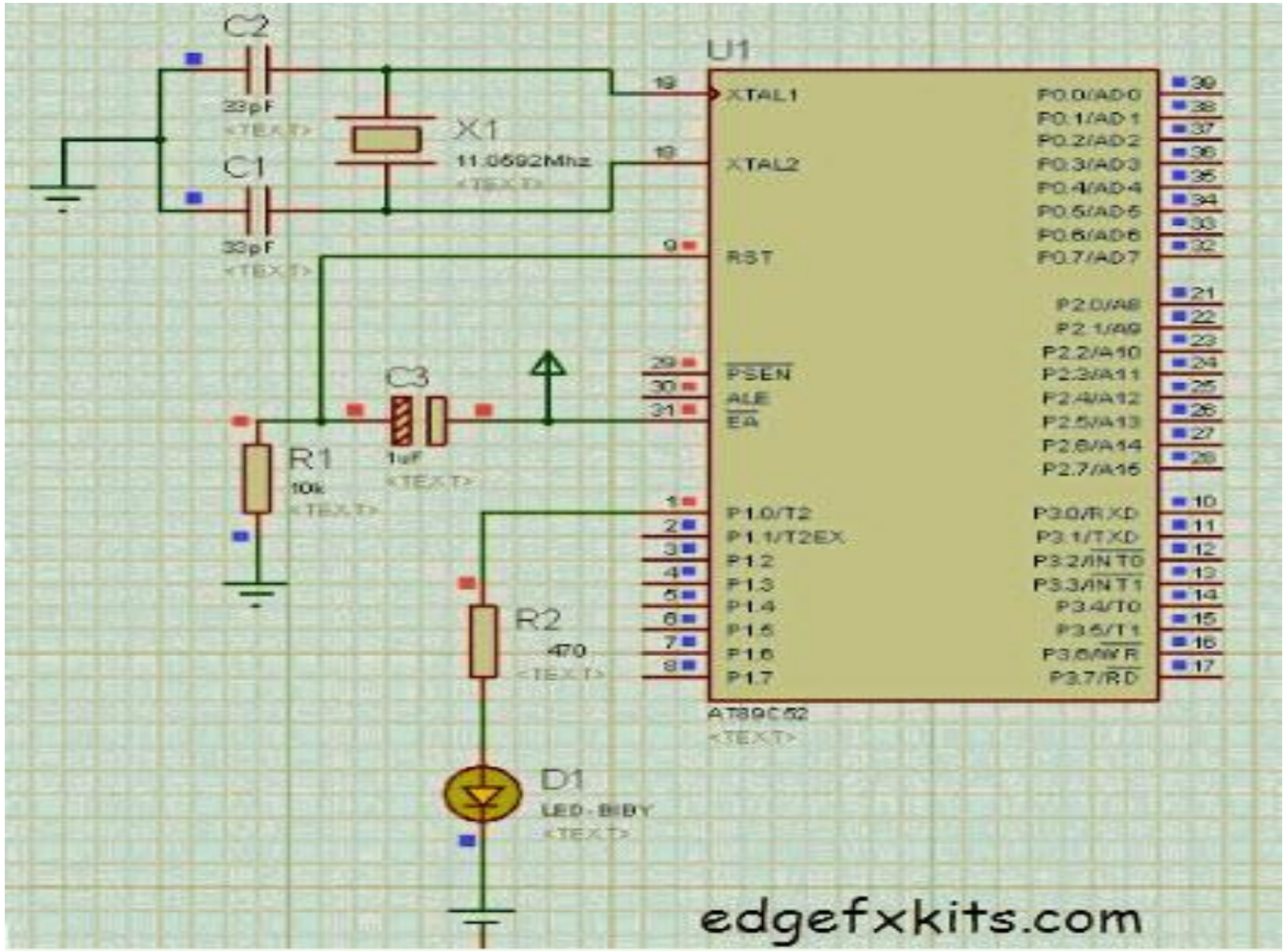
- Observe carefully the interface LED 2 is in forward biased because the input voltage of 5v connected to the positive terminal of the LED, So here the Microcontroller pin should be at LOW level. And vice versa with the interface 1 connections.
- The resistor is important in LED interfacing to limit the flowing current and avoid damaging the LED and/or MCU.
- Interface 1 will glow LED, only if the PIN value of the MC is HIGH as current flows towards the ground.
- Interface 2 will glow LED, only if the PIN value of the MC is LOW as current flows towards PIN due to its lower potential.

**Sourcing current:**

- Maximum current that 8051 port pin can be supply to drive an externally connected device
- Device-LED, buzzer, or TTL
- The anode of the LED is connected to the microcontroller. Hence the microcontroller provides the supplied voltage needed by the LED

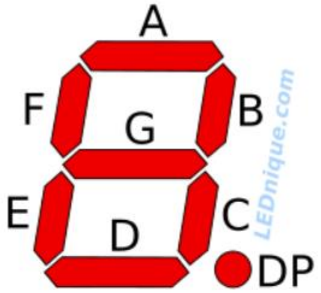
**Sinking Current:**

- Absorb through a device which is connected to an external supply. Pins 1,2,3-sink a maximum current-1.6mA
- Pins 0-3.2 mA
- The cathode of the LED, is connected to the microcontroller so as to provide it will with a logic zero.

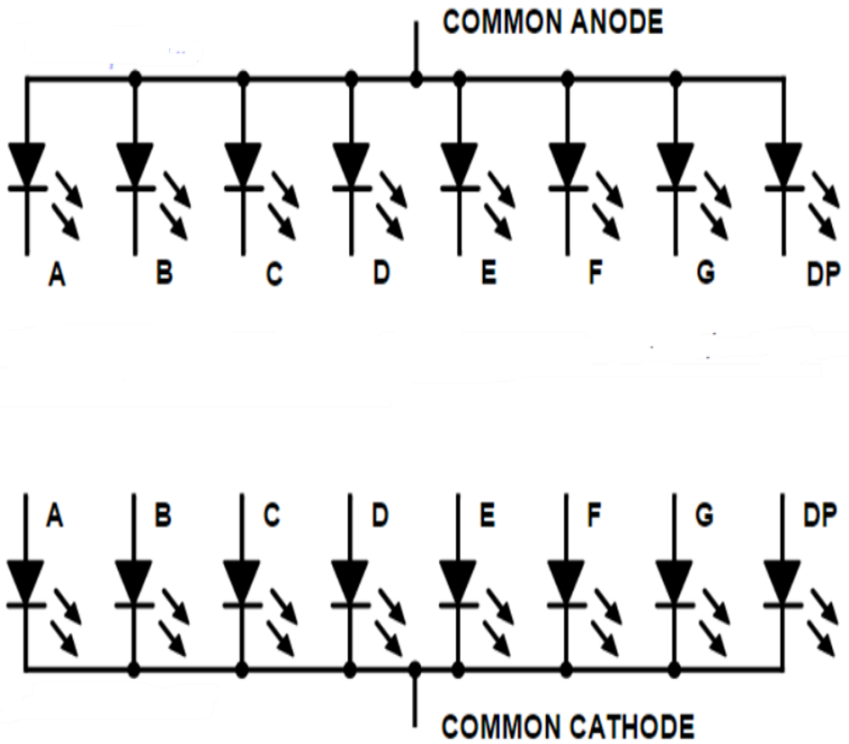

C2
33pF
<TEXT>
X1
11.0592Mhz
<TEXT>
C1
33pF
<TEXT>
U1
XTAL1
XTAL2
RST
PSEN
ALE
EA
C3
1uF
<TEXT>
R1
10k
<TEXT>
R2
470
<TEXT>
D1
LED-BIBY
<TEXT>
AT89C52
<TEXT>
edgefxkits.com

# Multiplexed 7 segment display interfacing

- 7 segment displays –numerical indicators
- No of LEDs are arranged in 7 segments
- Any number between 0 to 9 – lighting the appropriate segments
- Labelled-a to g. dot-h
- Forward bias-display digits-0 to 9
- Displays are of two types
  - Common anode
  - Common cathode



| Decimal | 7 Segment Code<br>a b c d e f g |
|---|---|
| 0 | 1 1 1 1 1 1 0 |
| 1 | 0 1 1 0 0 0 0 |
| 2 | 1 1 0 1 1 0 1 |
| 3 | 1 1 1 1 0 0 1 |
| 4 | 0 1 1 0 0 1 1 |
| 5 | 1 0 1 1 0 1 1 |
| 6 | 0 0 1 1 1 1 1 |
| 7 | 1 1 1 0 0 0 0 |
| 8 | 1 1 1 1 1 1 1 |
| 9 | 1 1 1 0 0 1 1 |



# PROGRAMMING IN EMBEDDED C

C consists of the following

- Preprocessor directives
- Main function
- Functions

Why C instead of ALP

- Lengthy and complex program in short time
- Easy to design the flow structure
- Facilitates data type declarations
- Software reusability is feasible

- Keywords in embedded C: special word with special meaning to the compiler.

  Bit Sbit sfr small Large

- Data types in Embedded C:

1. helps us declaring variables in the program
2. Different types: float, double, unsigned
3. Variable-addressable storage location to information to be used by the program. Each variable declare the to indicate the size and type of infn to be stored + to used to reference the infn

| C Language | Embedded C language |
| --- | --- |
| C is a general-purpose programming language, which can be used to design any type of desktop-based application.<br>It is a type of high-level language. | Embedded C is simply an extension of C language and it is used to develop micro-controller-based applications.<br>It is nothing but an extension of C. |
| C language is a hardware-independent language.<br>C compilers are OS-dependent. | Embedded C is a fully hardware-dependent language.<br>Embedded C is OS-independent. |
| For C language, the standard compilers can be used to compile and execute the program.<br>Popular Compiler to execute a C language program are:<br>GCC (GNU Compiler collection)<br>Borland turbo C,<br>Intel C++ | For Embedded C, specific compilers that are able to generate particular hardware/micro-controller-based output are used.<br>Popular Compiler to execute an Embedded C language program are:<br>Keil compiler<br>BiPOM ELECTRONIC<br>Green Hill Software |