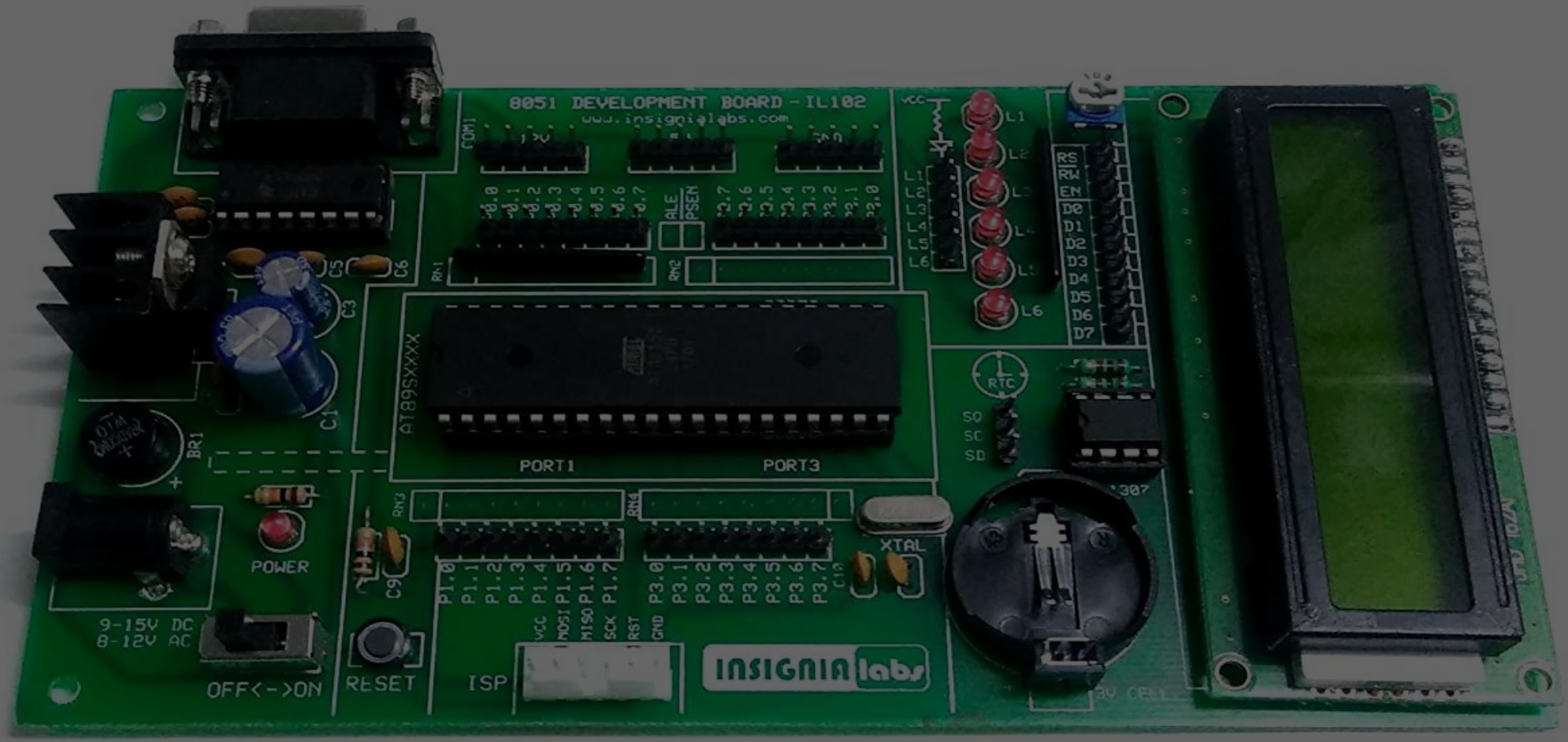


8051 MICROCONTROLLER

UNIT III SYLLABUS

- **The 8051 Microcontrollers:** Microcontrollers and Embedded processors, Overview of 8051 family. 8051 Microcontroller hardware, Input/output pins, Ports, and Circuits, External Memory.

- **8051 Programming in C:** Data Types and time delay in 8051 C, I/O Programming, Logic operations, Data conversion Programs.

8051(MCS-51) History

In 1981, Intel introduced The 8051(8-bit processor) also referred as System-on-chip.

- The CPU can work on only 8 bits of data at a time
- 128 bytes of RAM
- 4K bytes of on-chip ROM
- Two timers
- One serial port
- Four I/O ports, each 8 bits wide
- 6 interrupt sources

8051 Microcontroller family

- 8052 and 8031(ROM-less 8051)

Feature	8051	8052	8031
ROM (on-chip program space in bytes)	4K	8K	0K
RAM (bytes)	128	256	128
Timers	2	3	2
I/O pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	8	6

Various 8051 Microcontrollers

- 8751 microcontroller
- AT89C51 from Atmel Corporation
- DS89C4x0 from Dallas Semiconductor
- DS5000 from Dallas Semiconductor
- OTP (one-time-programmable) version of 8051
- 8051 family from Philips

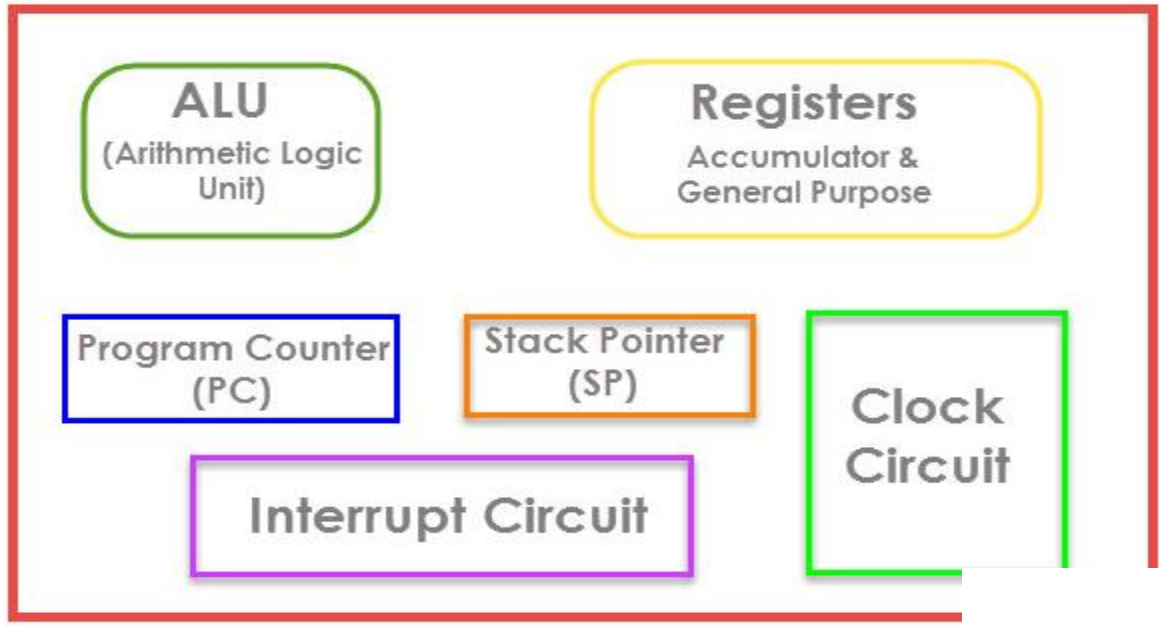
AT89C51

Table 1-6: Versions of 8051 From Atmel (All ROM Flash)

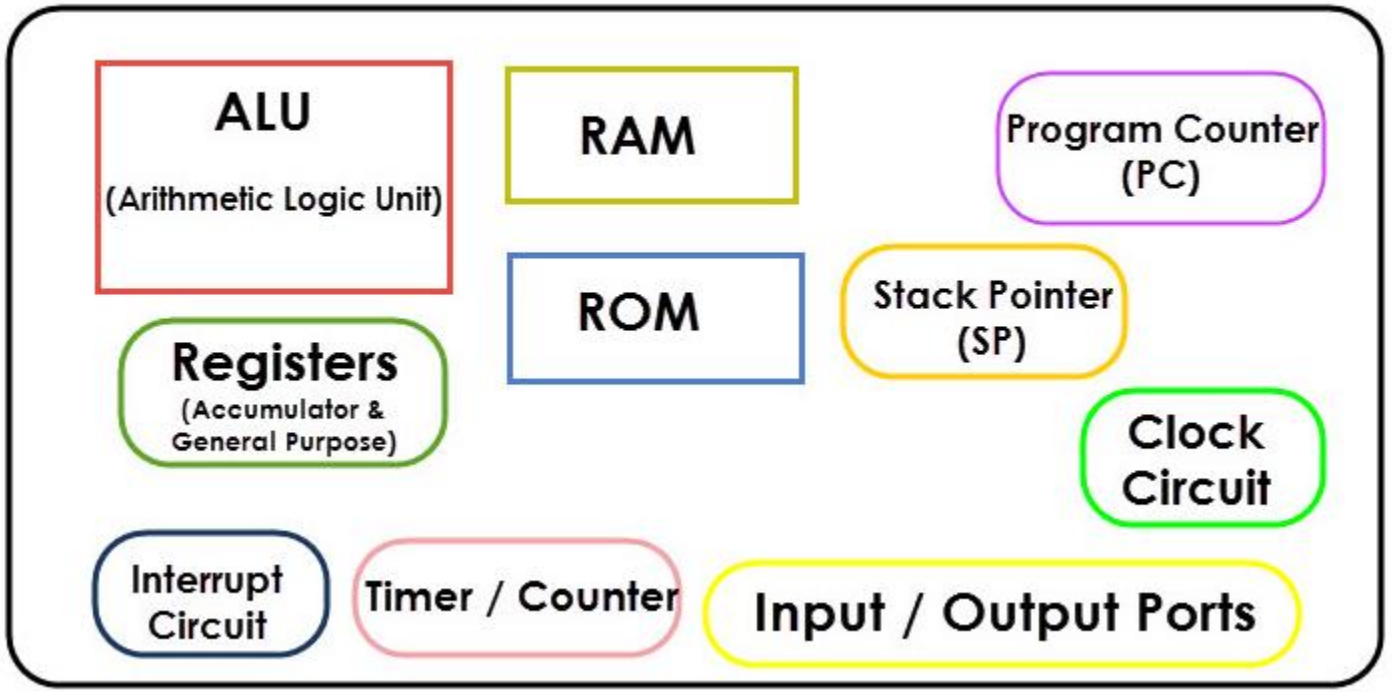
Part Number	ROM	RAM	I/O pins	Timer	Interrupt	V_{CC}	Packaging
AT89C51	4K	128	32	2	6	5V	40
AT89LV51	4K	128	32	2	6	3V	40
AT89C1051	1K	64	15	1	3	3V	20
AT89C2051	2K	128	15	2	6	3V	20
AT89C52	8K	128	32	3	8	5V	40
AT89LV52	8K	128	32	3	8	3V	40

Note: "C" in the part number indicates CMOS.

Block Diagram of Microprocessor

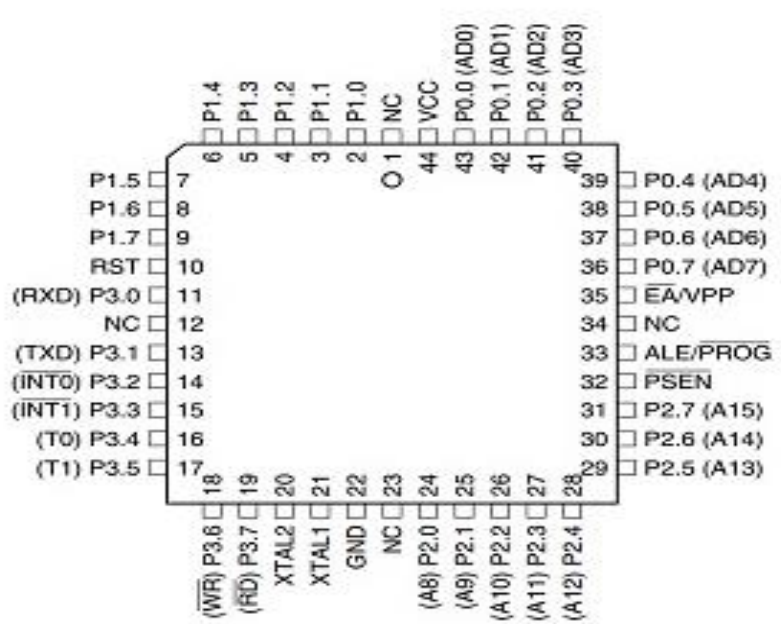


Block Diagram of Microcontroller

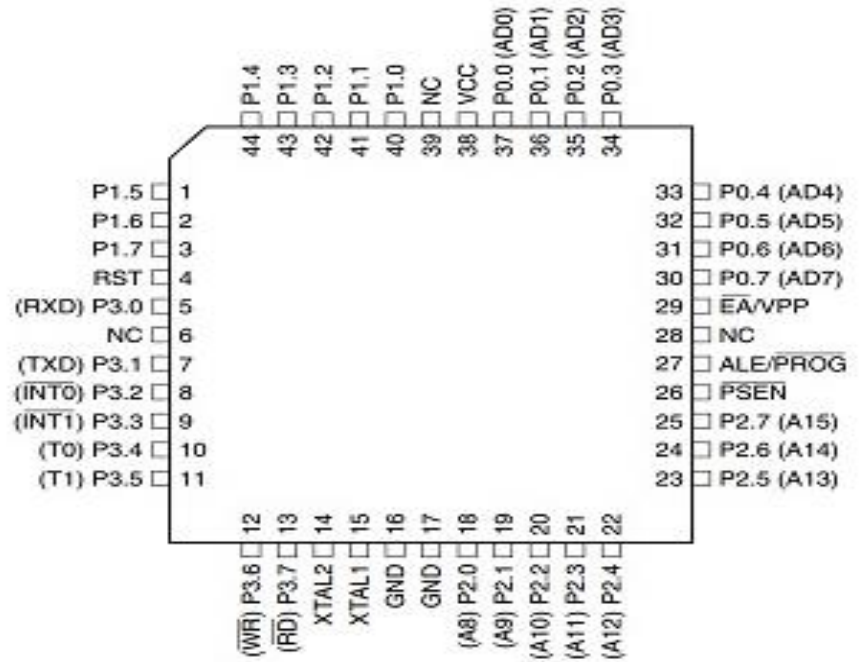


8051 PIN DESCRIPTION

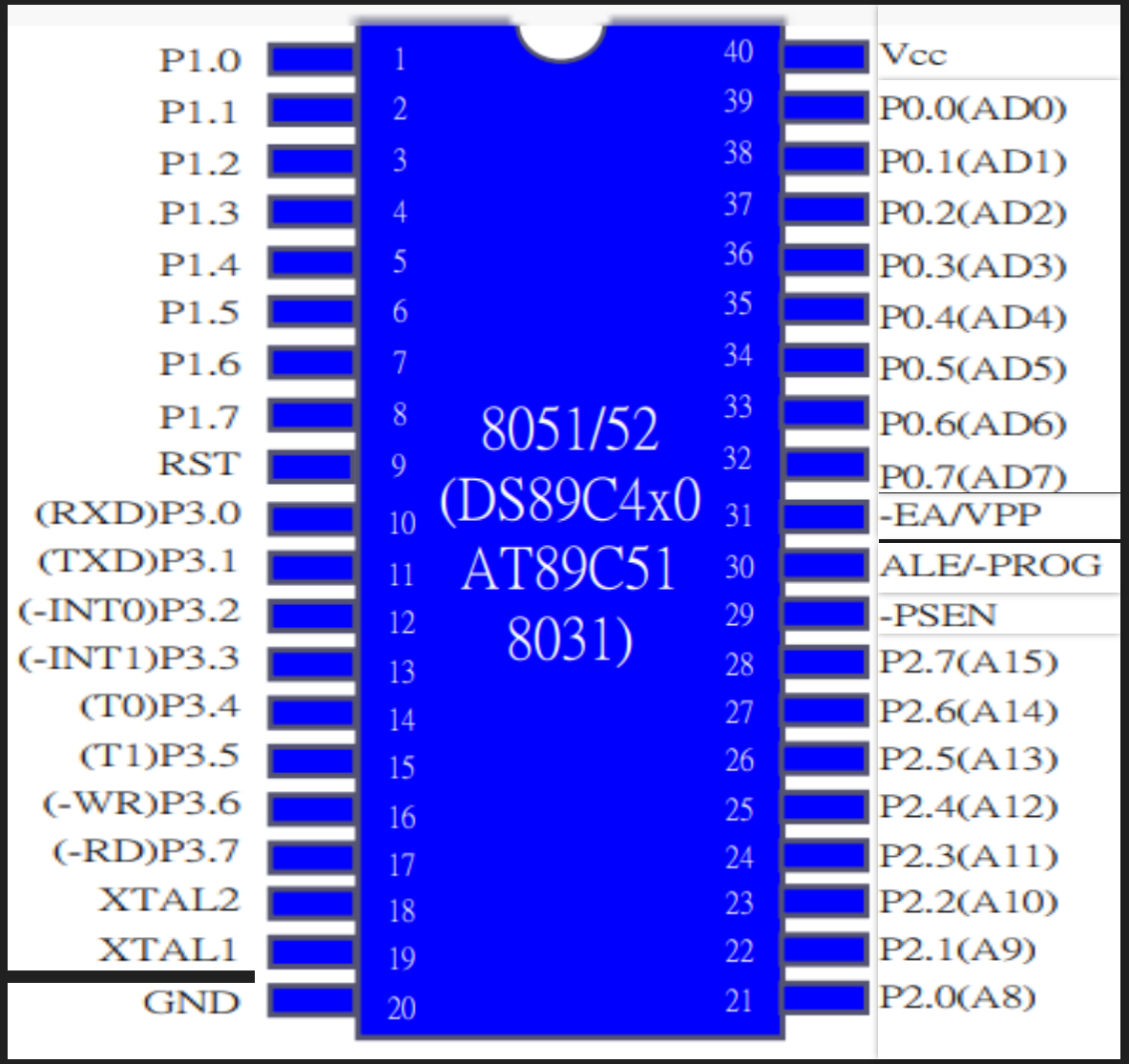
- 8051 family members (e.g, 8751, 89C51, 89C52, DS89C4x0)
- Have 40 pins dedicated for various functions such as I/O, -RD, -WR, address, data, and interrupts
- Some companies provide a 20-pin version of the 8051 with a reduced number of I/O ports for less demanding applications



44-Lead PLCC

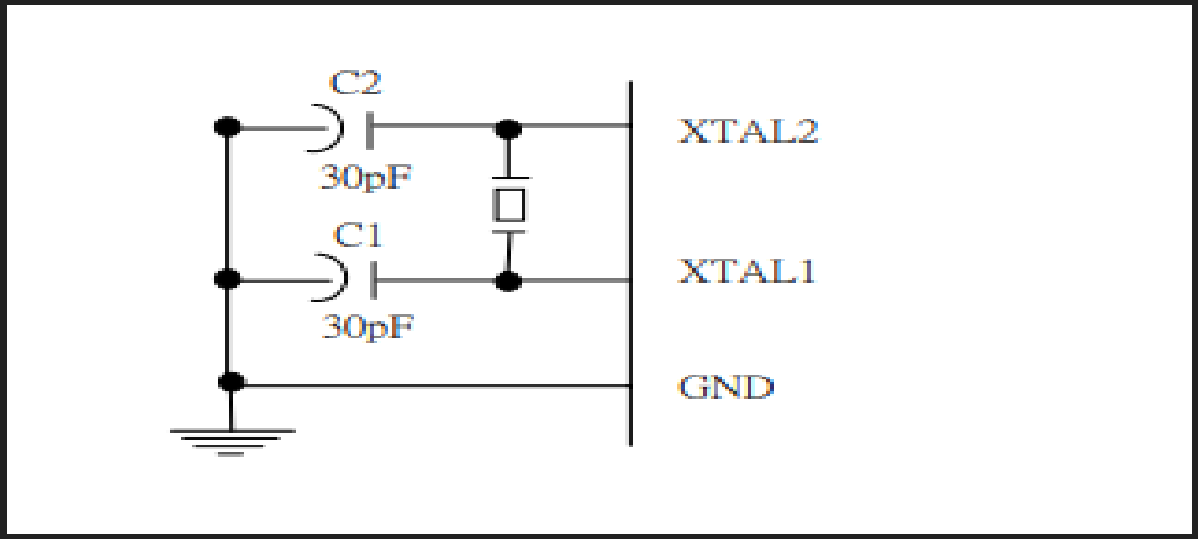


44-Lead TQFP


P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
(RXD)P3.0 10
(TXD)P3.1 11
(-INT0)P3.2 12
(-INT1)P3.3 13
(T0)P3.4 14
(T1)P3.5 15
(-WR)P3.6 16
(-RD)P3.7 17
XTAL2 18
XTAL1 19
GND 20
8051/52
(DS89C4x0
AT89C51
8031)
40 Vcc
39 P0.0(AD0)
38 P0.1(AD1)
37 P0.2(AD2)
36 P0.3(AD3)
35 P0.4(AD4)
34 P0.5(AD5)
33 P0.6(AD6)
32 P0.7(AD7)
31 -EA/VPP
30 ALE/-PROG
29 -PSEN
28 P2.7(A15)
27 P2.6(A14)
26 P2.5(A13)
25 P2.4(A12)
24 P2.3(A11)
23 P2.2(A10)
22 P2.1(A9)
21 P2.0(A8)

XTAL1 and XTAL2

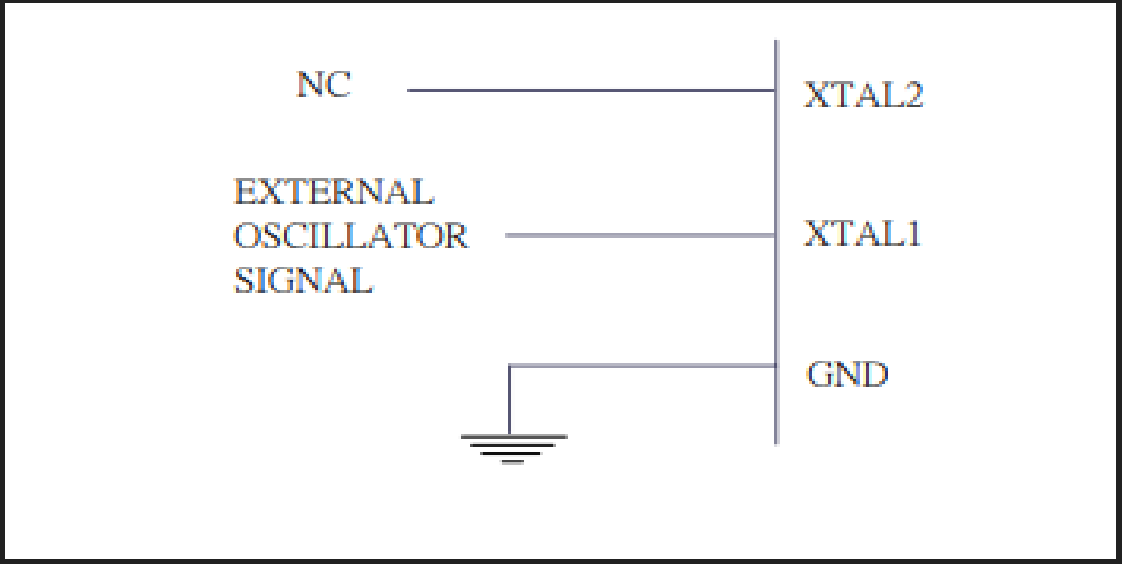
- The 8051 has an on-chip oscillator but requires an external clock to run it
- A quartz crystal oscillator is connected to inputs XTAL1 (pin19) and XTAL2 (pin18) The quartz crystal oscillator also needs two capacitors of 30 pF value



- If you use a frequency source other than a crystal oscillator, such as a TTL oscillator

It will be connected to XTAL1

XTAL2 is left unconnected



- The speed of 8051 refers to the maximum oscillator frequency connected to XTAL

Eg: A 12-MHz chip must be connected to a crystal with 12 MHz frequency or less

We can observe the frequency on the XTAL2 pin using the oscilloscope

RST

RESET pin is an input and is active high (normally low)

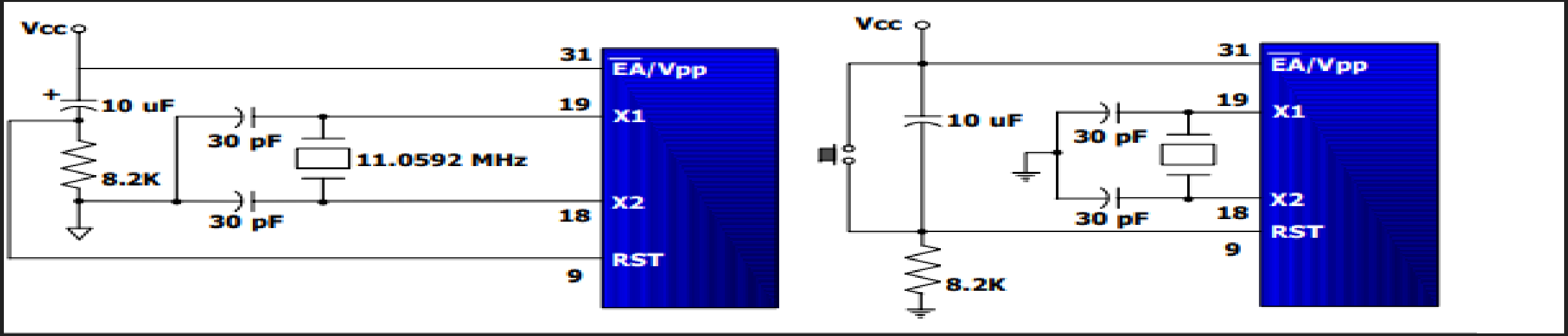
- Upon applying a high pulse to this pin, the microcontroller will reset and terminate all activities. This is often referred to as a power-on reset
- Activating a power-on reset will cause all values in the registers to be lost

RESET value of some 8051 registers

we must place the first line of source code in ROM location 0

Register	Reset Value
PC	0000
DPTR	0000
ACC	00
PSW	00
SP	07
B	00
P0-P3	FF

- In order for the RESET input to be effective, it must have a minimum duration of 2 machine cycles
- In other words, the high pulse must be high for a minimum of 2 machine cycles before it is allowed to go low



EA

- EA, "external access", is an input pin and must be connected to Vcc or GND
- The 8051 family members all come with on-chip ROM to store programs

EA pin is connected to Vcc

- The 8031 and 8032 family members do no have on-chip ROM, so code is stored on an external ROM and is fetched by 8031/32

EA pin must be connected to GND to indicate that the code is stored externally

PSEN

used mainly in 8031-based systems

- PSEN, "program store enable", is an output pin

This pin is connected to the OE pin of the ROM

ALE

- ALE, "address latch enable", is an output pin and is active high
- Port O provides both address and data The 8031 multiplexes address and data through port 0 to save pins
- ALE pin is used for demultiplexing the address and data by connecting to the G pin of the 74LS373 chip

I/O Port pins

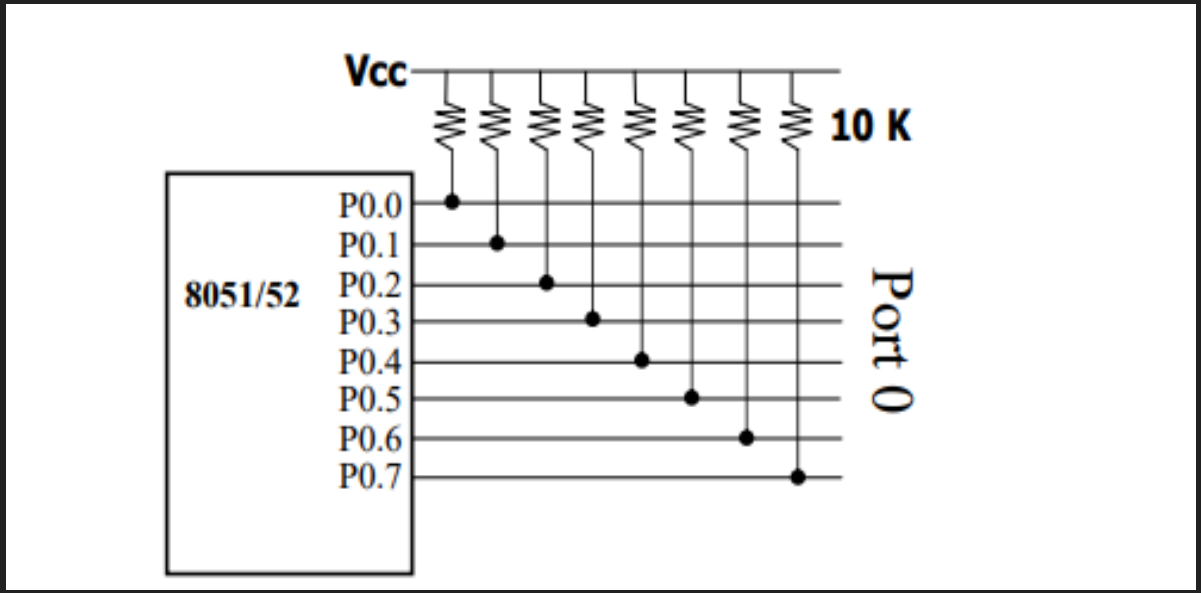
- The four 8-bit I/O ports P0, P1, P2 and P3 each uses 8 pins
- All the ports upon RESET are configured as output, ready to be used as input ports

P0

Port O is also designated as ADO-AD7, allowing it to be used for both address and data

- When connecting an 8051/31 to an external memory, port 0 provides both address and data
- The 8051 multiplexes address and data through port 0 to save pins
- ALE indicates if PO has address or data
- When ALE=0, it provides data D0-D7
- When ALE=1, it has address A0-A7

- It can be used for input or output, each pin must be connected externally to a 10K ohm pull-up resistor
- This is due to the fact that PO is an open drain, unlike P1, P2, and P3
- Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips



P1 and P2

In 8051-based systems with no external memory connection

Both P1 and P2 are used as simple I/O

- In 8031/51-based systems with external memory connections

Port 2 must be used along with PO to provide the 16-bit address for the external memory

- PO provides the lower 8 bits via A0 - A7
- P2 is used for the upper 8 bits of the 16-bit address, designated as A8 - A15, and it cannot be used for I/O

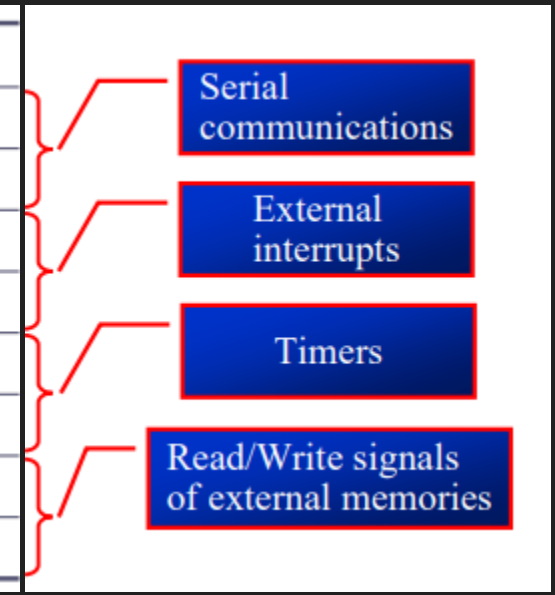
PORT 3

- Port 3 can be used as input or output

Port 3 does not need any pull-up resistors

- Port 3 has the additional function of providing some extremely important signals

P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17



Machine cycle and crystal frequency

- CPU executing an instruction takes a certain number of clock cycles

These are referred as to as machine cycles

- The length of machine cycle depends on the frequency of the crystal oscillator connected to 8051
- In original 8051, one machine cycle lasts 12 oscillator periods

- 8051 uses one or more machine cycles to execute an instruction.
- The period od machine cycle varies among the different versions of 8051 from 12 clocks in AT89C51 to 1 clock in DS89C4x0 chip.
- The frequency of the crystal oscillator connected to X-X2 pins dictates the speed of clock used in the machine cycle

INTEL HEX FILE

- Intel hex file is a widely used file format

Designed to standardize the loading of executable machine codes into a ROM chip

- Loaders that come with every ROM burner (programmer) support the Intel hex file format
- In many newer Windows-based assemblers the Intel hex file is produced automatically (by selecting the right setting)
- In DOS-based PC you need a utility called OH (object-to-hex) to produce that

In the DOS environment

- The object file is fed into the linker program to produce the abs file

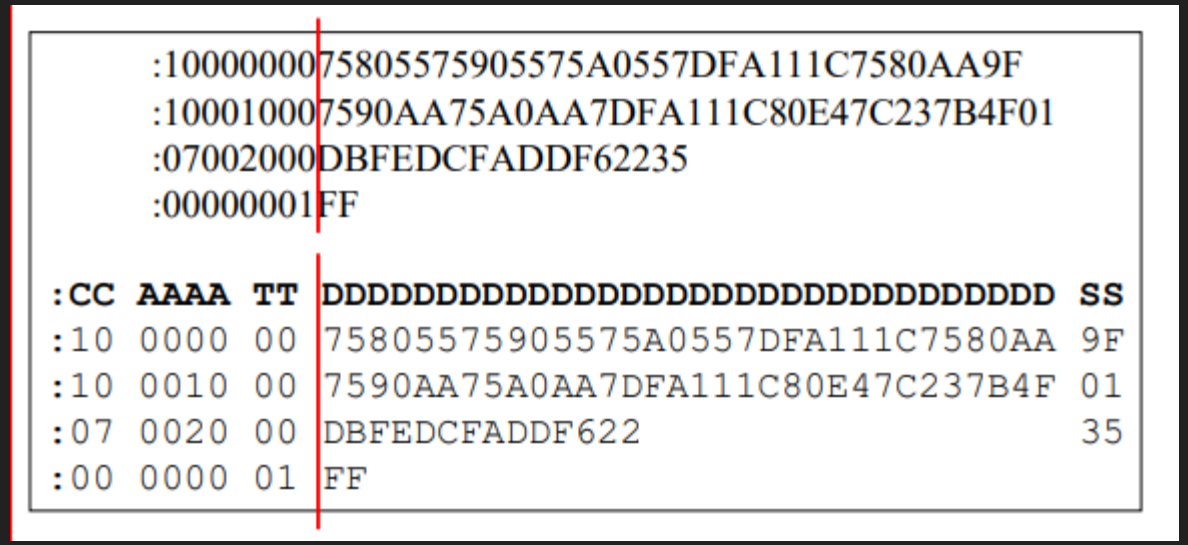
The abs file is used by systems that have a monitor program

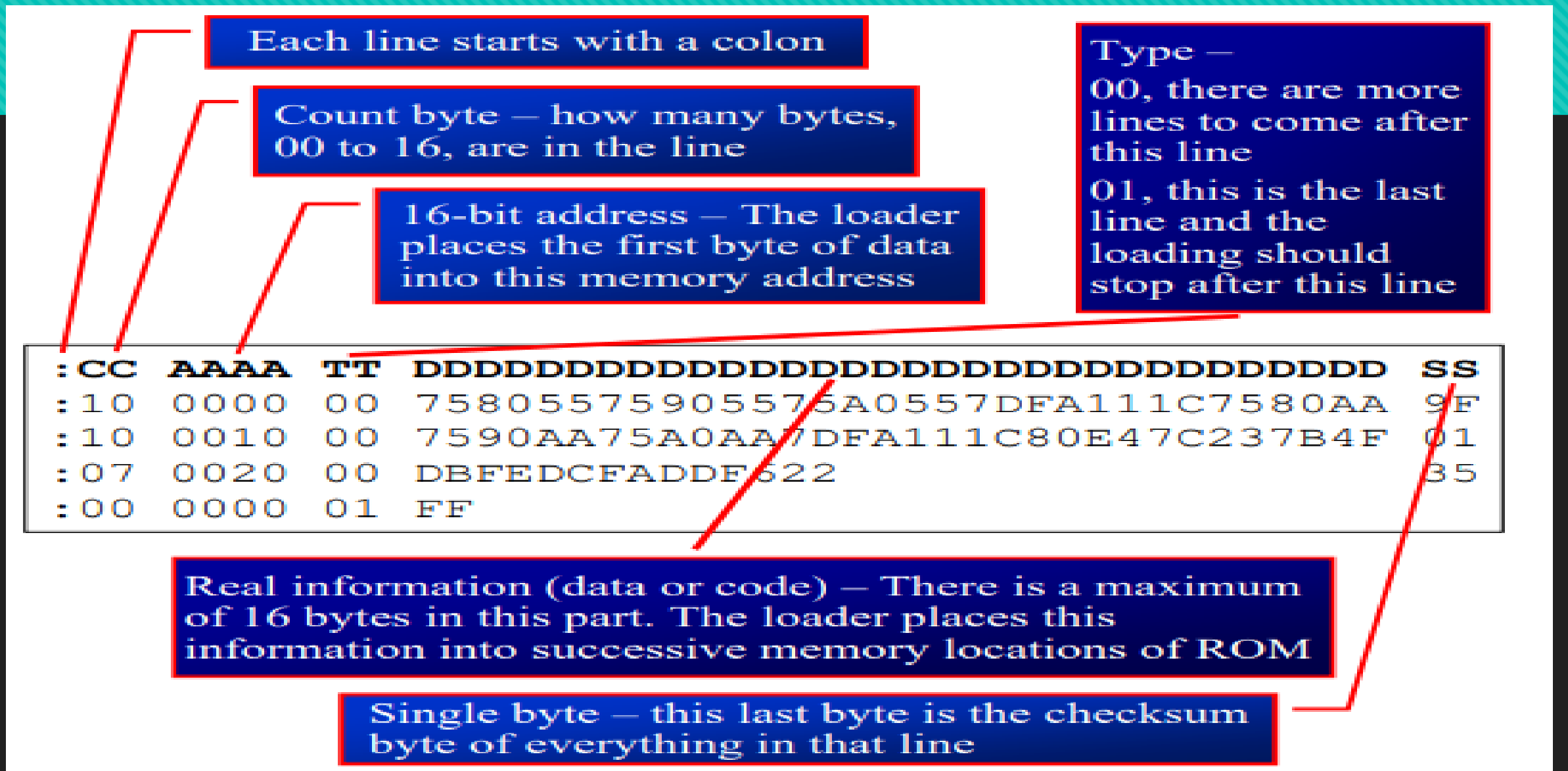
- Then the abs file is fed into the OH utility to create the Intel hex file

The hex file is used only by the loader of an EPROM programmer to load it into the ROM chip

The hex file provides the following:

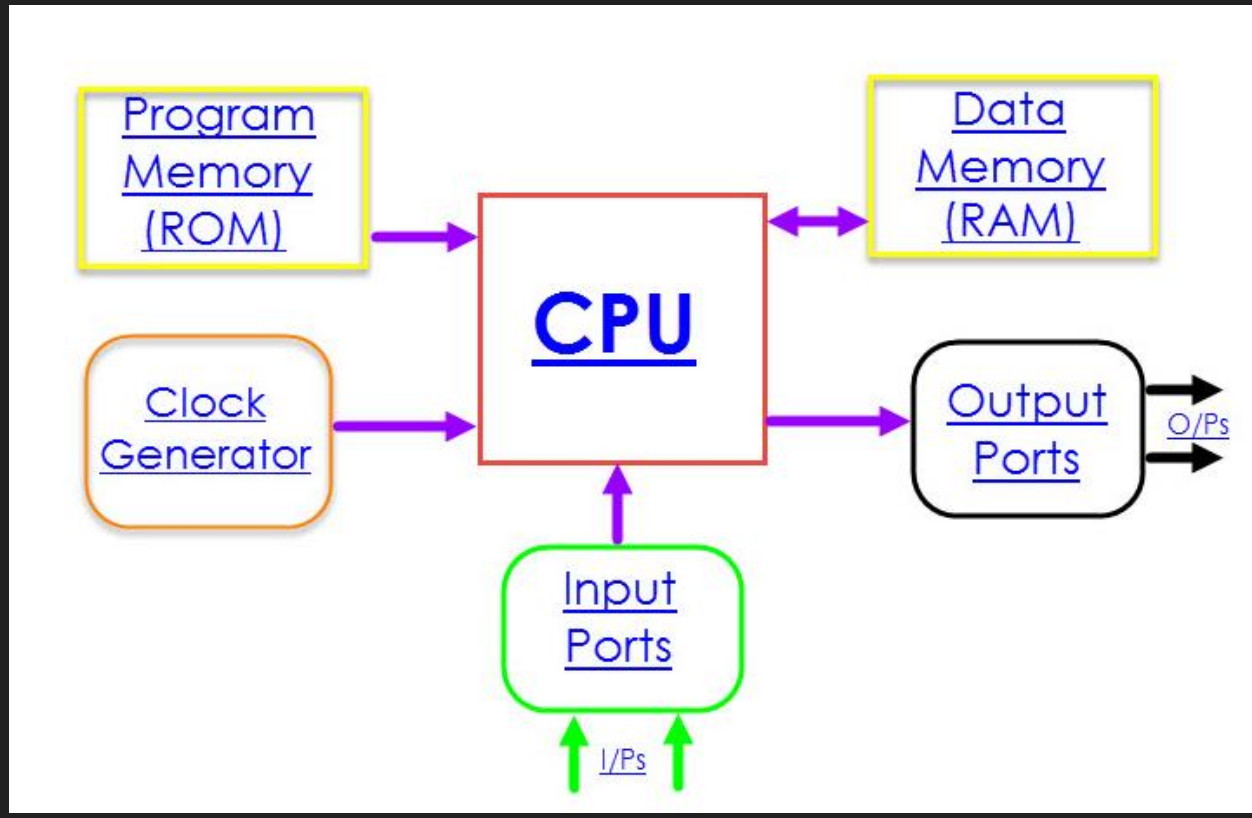
- The number of bytes of information to be loaded
- The information itself
- The starting address where the information must be placed




Each line starts with a colon
Count byte — how many bytes, 00 to 16, are in the line
16-bit address — The loader places the first byte of data into this memory address
Type —
00, there are more lines to come after this line
01, this is the last line and the loading should stop after this line
:CC AAAA TT DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD SS
:10 0000 00 7580557590557 5A0557DFA111C7580AA 9F
:10 0010 00 7590AA75A0AA7DFA111C80E47C237B4F 01
:07 0020 00 DBFEDCFADDF622 35
:00 0000 01 FF
Real information (data or code) — There is a maximum of 16 bytes in this part. The loader places this information into successive memory locations of ROM
Single byte — this last byte is the checksum byte of everything in that line

8051 Microcontroller Memory Organization

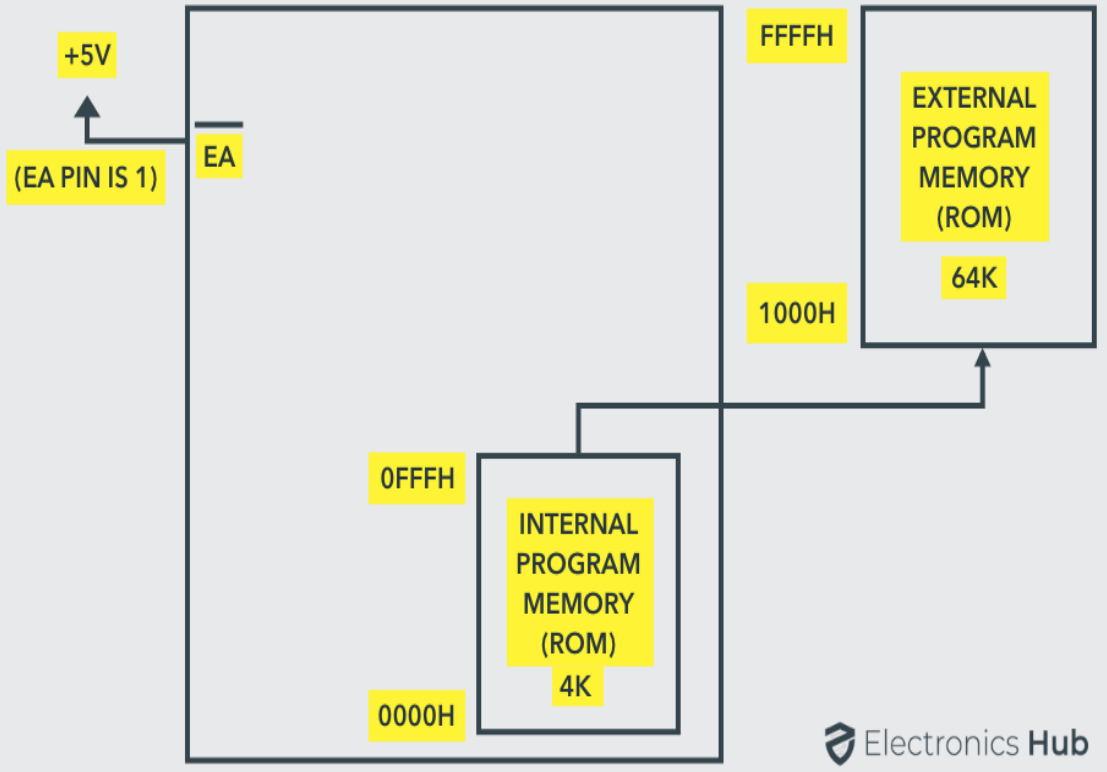
- The 8051 Microcontroller Memory is separated in Program Memory (ROM) and Data Memory (RAM). The Program Memory of the 8051 Microcontroller is used for storing the program to be executed i.e., instructions.
- The Data Memory on the other hand, is used for storing temporary variable data and intermediate results.
- 8051 Microcontroller has both Internal ROM and Internal RAM. If the internal memory is inadequate, you can add external memory using suitable circuits.



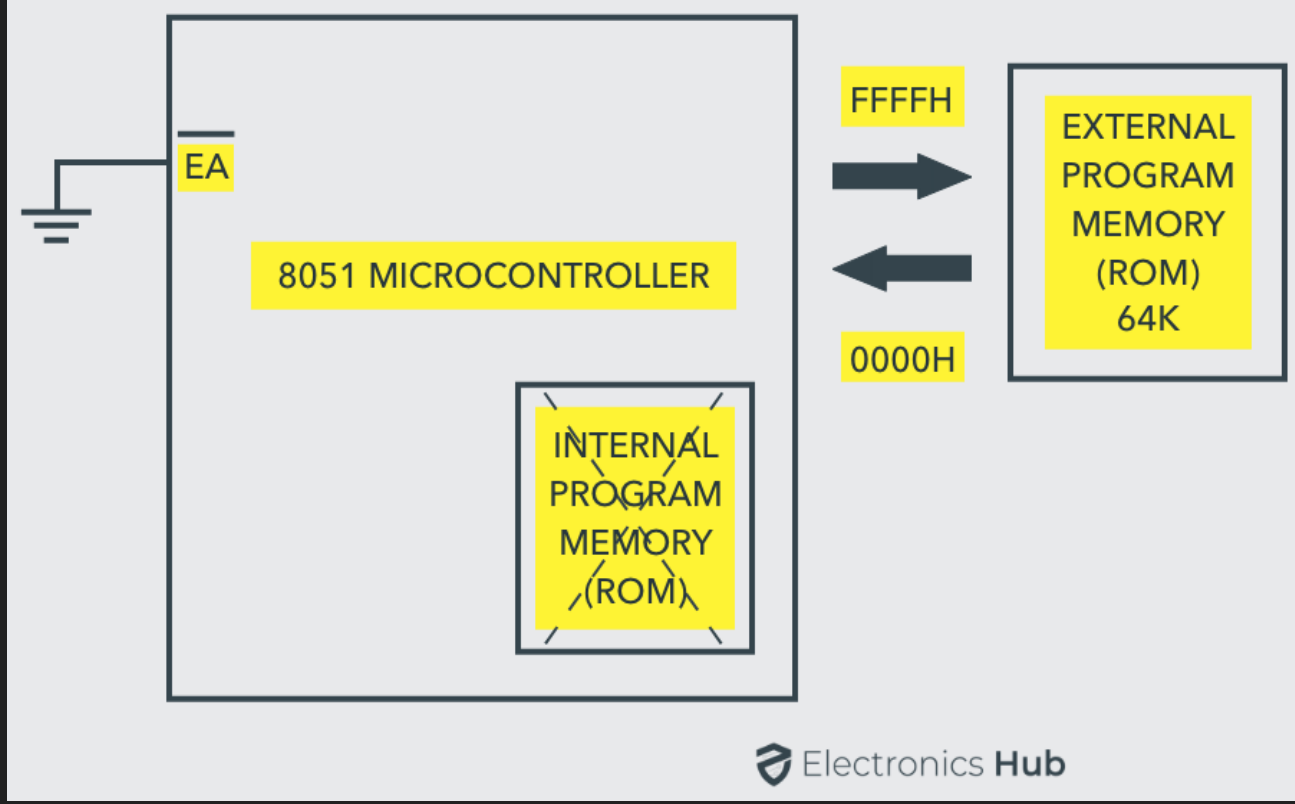
Program Memory (ROM)

- The original 8051 Microcontroller by Intel has 4KB of internal ROM.
- Some variants of 8051 like the 8031 and 8032 series doesn't have any internal ROM (Program Memory) and must be interfaced with external Program Memory with instructions loaded in it.
- In case of 4KB of Internal ROM, the address space is 0000H to 0FFFH. If the address space i.e., the program addresses exceed this value, then the CPU will automatically fetch the code from the external Program Memory.
- For this, the External Access Pin (EA Pin) must be pulled HIGH i.e., when the EA Pin is high, the CPU first fetches instructions from the Internal Program Memory in the address range of 0000H to 0FFFFH and if the memory addresses exceed the limit, then the instructions are fetched from the external ROM in the address range of 1000H to FFFFH.

Using Both Internal And External Program Memory With 8051

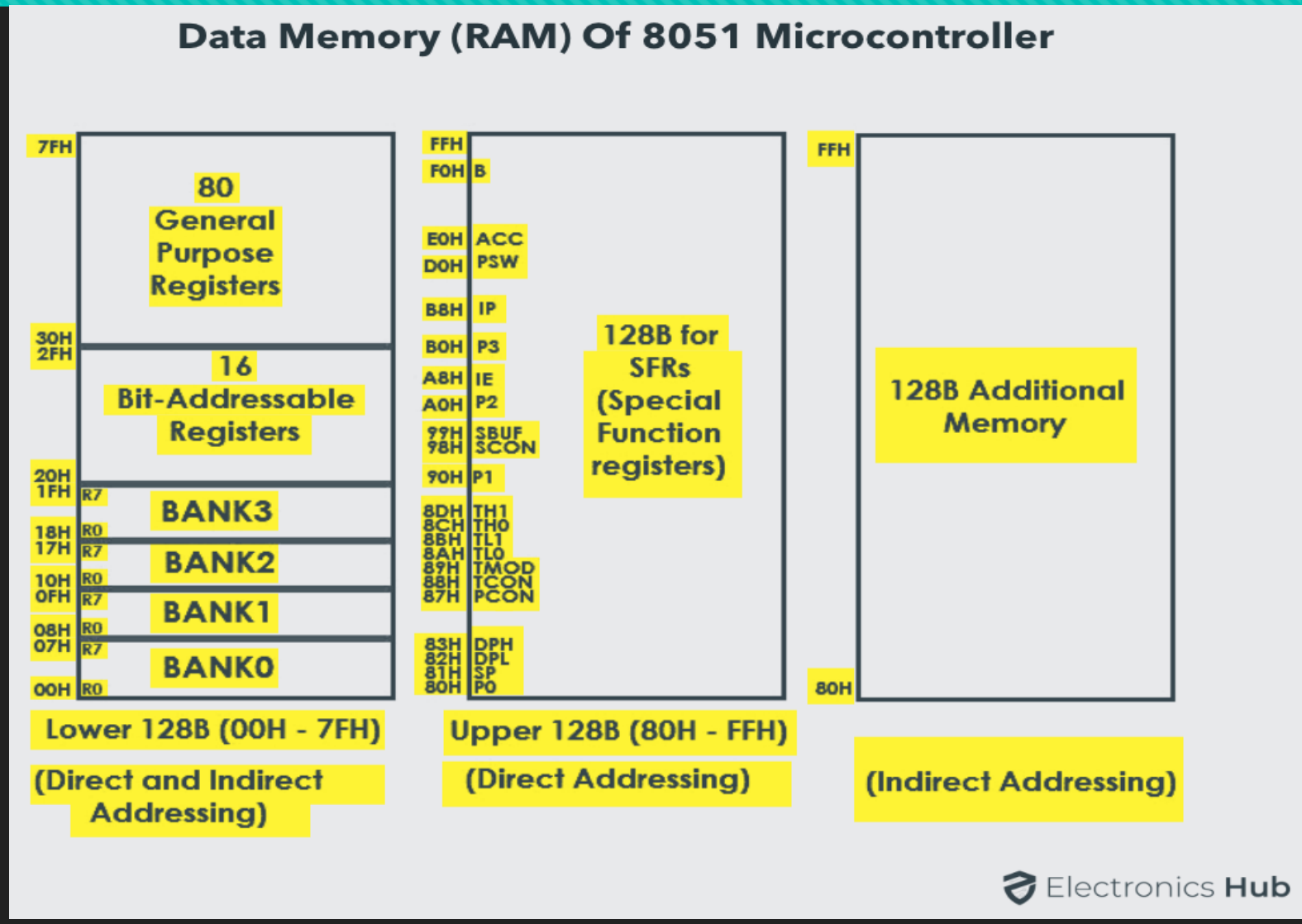


Using Only External Program Memory With 8051



Data Memory (RAM)

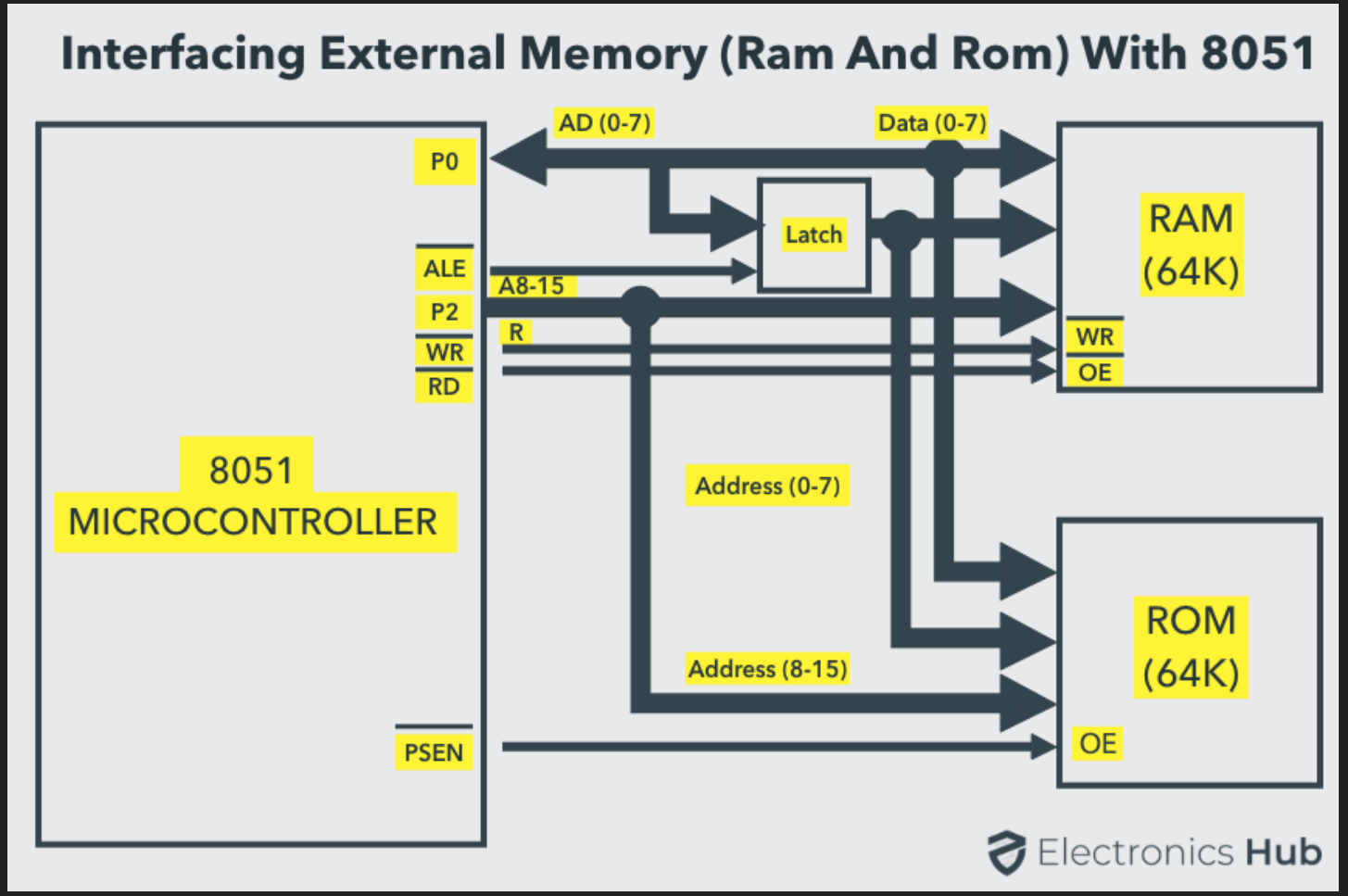
- Original Intel's 8051 Microcontroller had 128B of internal RAM.
- In the first 128B of RAM (from 00H to 7FH), the first 32B i.e., memory from addresses 00H to 1FH consists of 32 Working Registers that are organized as four banks with 8 Registers in each Bank.



Name of the Register	*Function*	*Internal RAM Address (HEX)*
ACC	Accumulator	E0H
B	B Register (for Arithmetic)	F0H
DPH	Addressing External Memory	83H
DPL	Addressing External Memory	82H
IE	Interrupt Enable Control	A8H
IP	Interrupt Priority	B8H
P0	PORT 0 Latch	80H
P1	PORT 1 Latch	90H
P2	PORT 2 Latch	A0H
P3	PORT 3 Latch	B0H
PCON	Power Control	87H
PSW	Program Status Word	D0H
SCON	Serial Port Control	98H
SBUF	Serial Port Data Buffer	99H
SP	Stack Pointer	81H
TMOD	Timer / Counter Mode Control	89H
TCON	Timer / Counter Control	88H
TL0	Timer 0 LOW Byte	8AH
TH0	Timer 0 HIGH Byte	8CH
TL1	Timer 1 LOW Byte	8BH
TH1	Timer 1 HIGH Byte	8DH

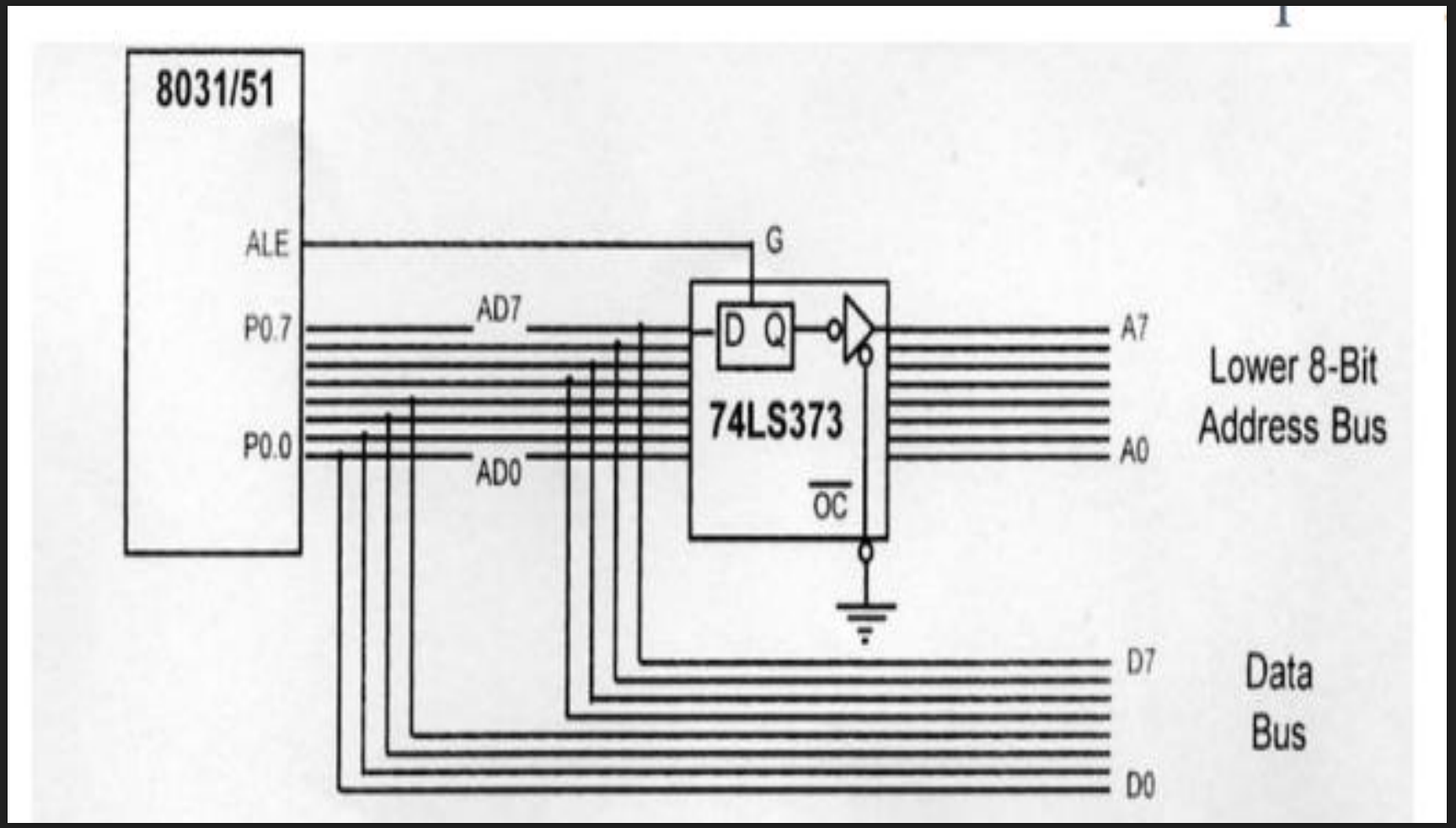
Interfacing External Memory with 8051 Microcontroller

- The reason for interfacing external Program Memory or ROM is that complex programs written in high – level languages often tend to be larger and occupy more memory.
- Another important reason is that chips like 8031 or 8032, which doesn't have any internal ROM, have to be interfaced with external ROM.

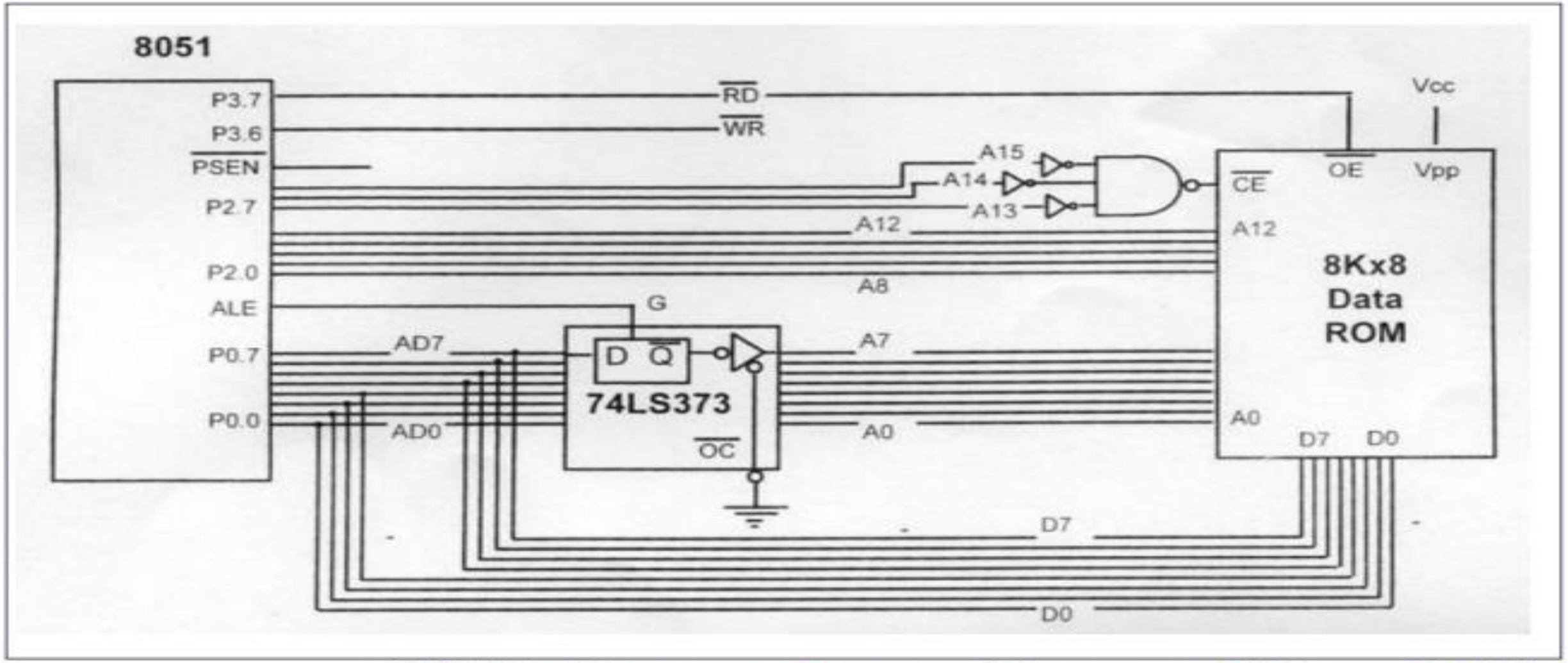


- For 8751/89C51/DS5000-based system, we connected the EA pin to Vcc to indicate that the program code is stored in the microcontroller's on-chip ROM
- To indicate that the program code is stored in external ROM, this pin must be connected to GND
- In the 8031/51, port 0 and port 2 provide the 16-bit address to access external memory
- P0 provides the lower 8 bit address A0 – A7, and P2 provides the upper 8 bit address A8 – A15
- P0 is also used to provide the 8-bit data bus D0 – D7
- P0.0 – P0.7 are used for both the address and data paths

address/data multiplexing

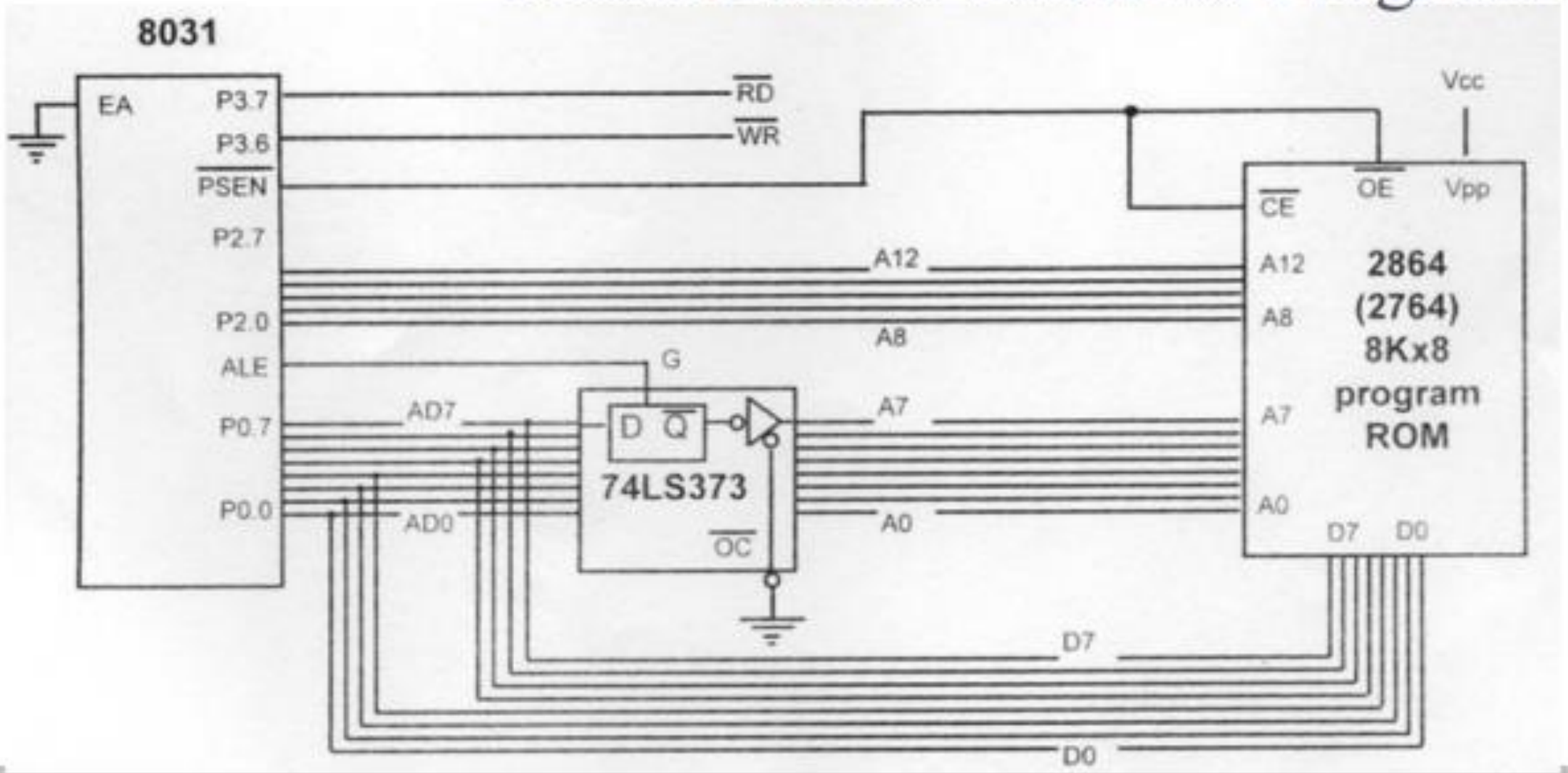

8031/51
ALE
P0.7
P0.0
G
AD7
AD0
D Q
74LS373
OC
A7
A0
Lower 8-Bit
Address Bus
D7
D0
Data
Bus

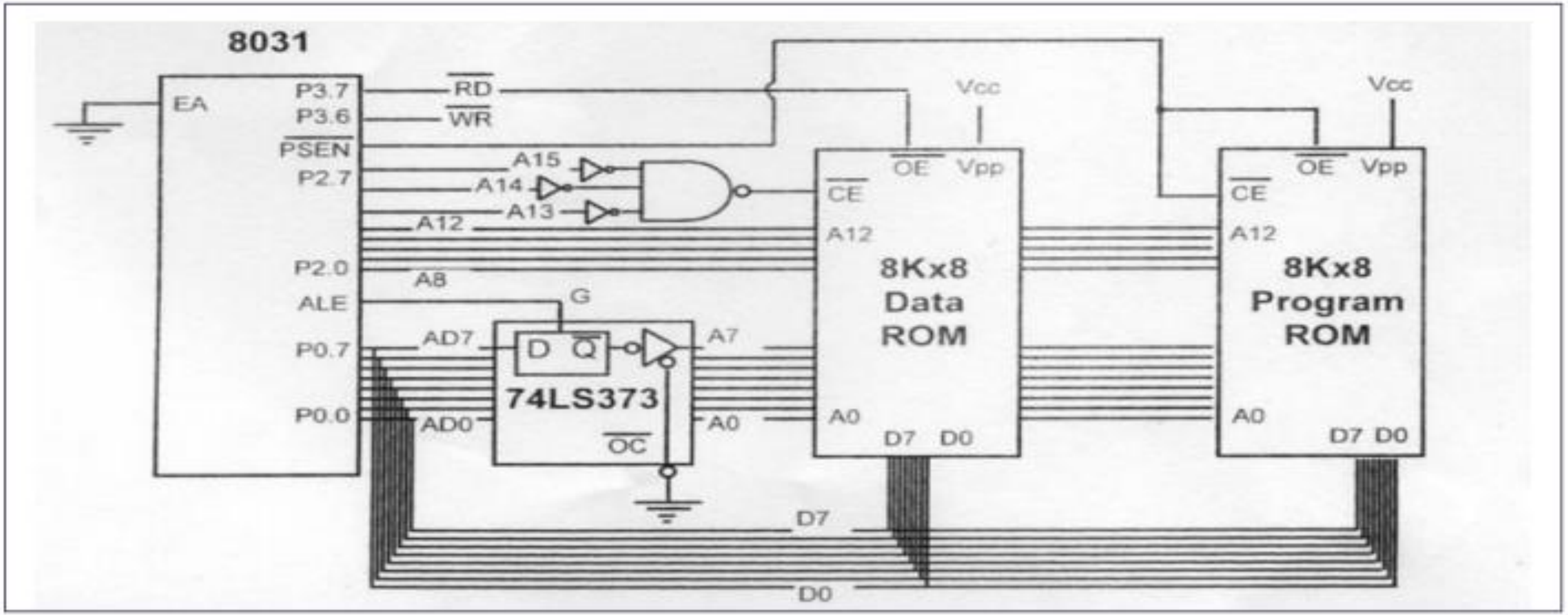
Interfacing External ROM



8051 Connection to External Data ROM

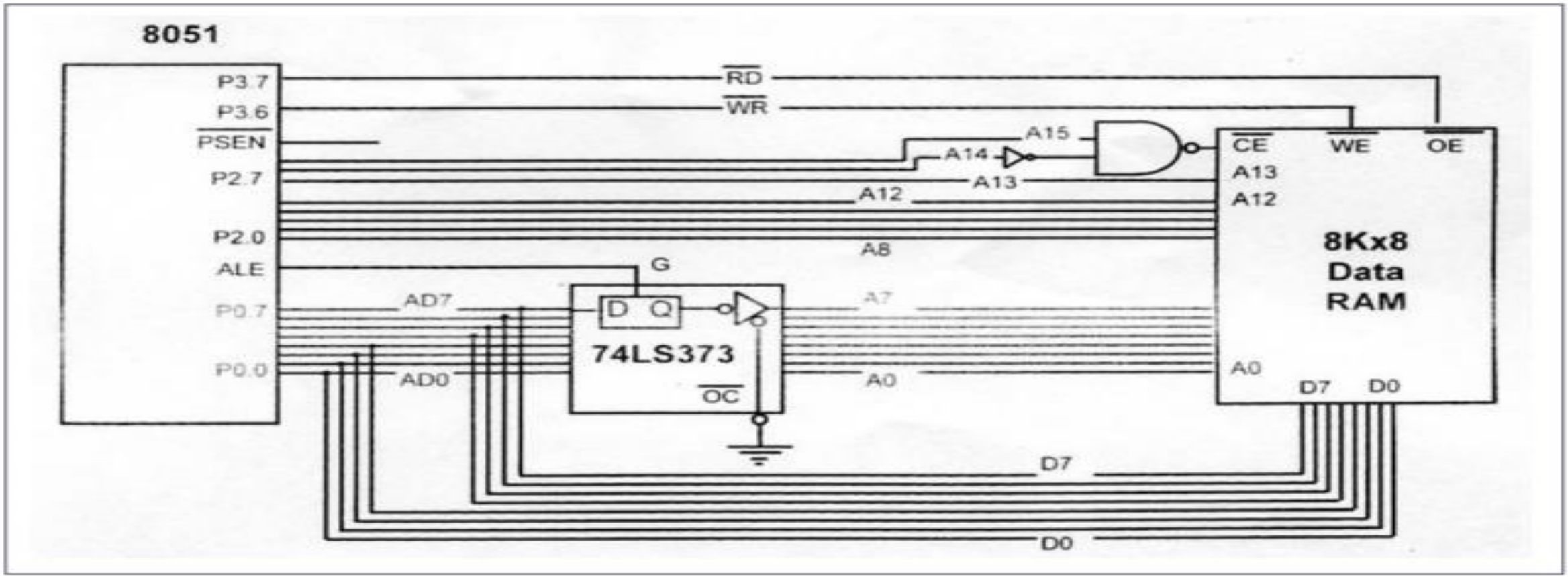
Connection to External Program ROM



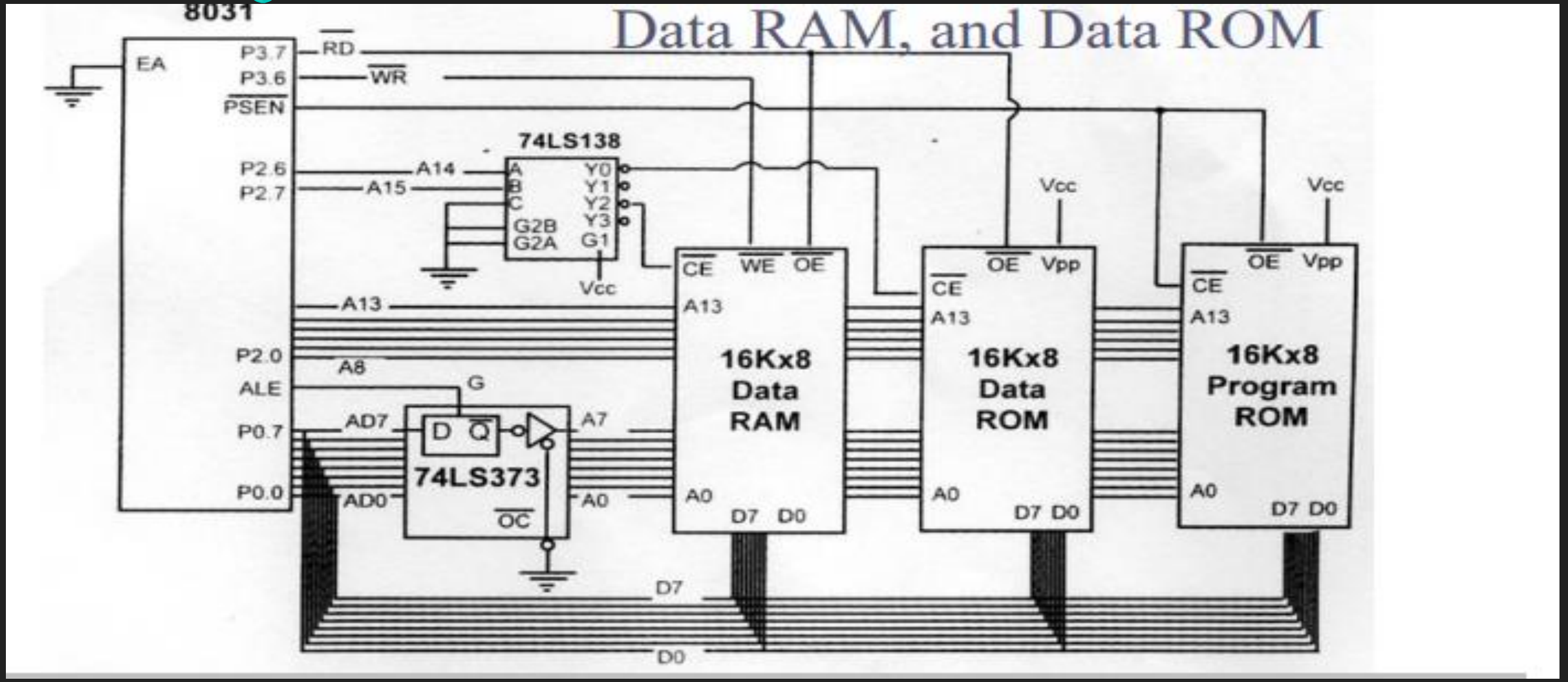


8031 Connection to External Data ROM and External Program ROM

Interfacing with External Data RAM



8051 Connection to External Data RAM


Data RAM, and Data ROM
8031
EA
P3.7
P3.6
PSEN
RD
WR
74LS138
P2.6
P2.7
A14
A15
A
B
C
G2B
G2A
Y0
Y1
Y2
Y3
G1
Vcc
A13
P2.0
A8
ALE
G
P0.7
AD7
D
Q
74LS373
A7
P0.0
AD0
OC
A0
CE
WE
OE
16Kx8
Data
RAM
D7
D0
Vpp
16Kx8
Data
ROM
16Kx8
Program
ROM

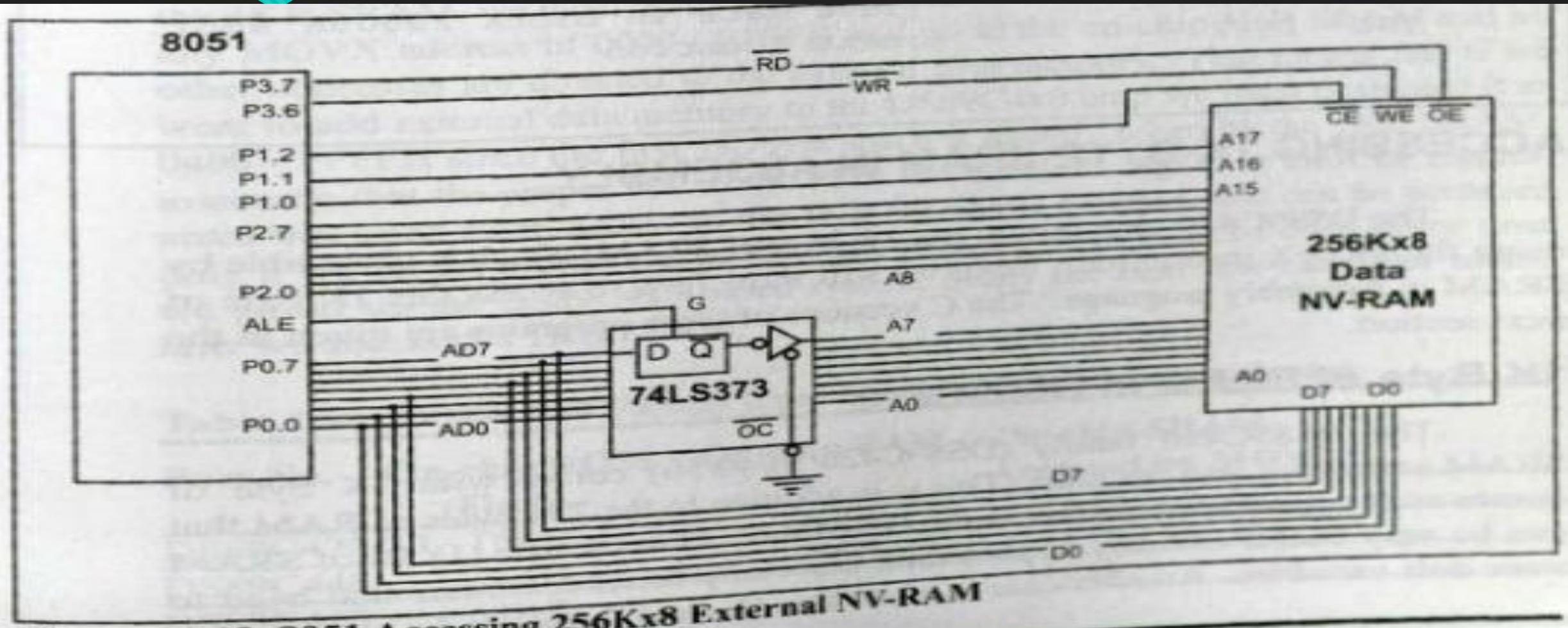


Figure 14-18. 8051 Accessing 256Kx8 External NV-RAM

Memory Capacity

- The number of bits that a semiconductor memory chip can store is called chip capacity
- This must be distinguished from the storage capacity of computer system

Memory Organization

- The number of locations within a memory IC depends on the address pins
- The number of bits that each location can hold is always equal to the number of data pins
- A memory chip contain 2x location, where x is the number of address pins
- Each location contains y bits, where y is the number of data pins on the chip
- The entire chip will contain $2^x \times$ y bits

- To access the data, the address is presented to the address pins, the READ pin is activated, and after a certain amount of time has elapsed, the data shows up at the data pins
- The shorter this elapsed time, the better, and consequently, the more expensive the memory chip.
- The speed of the memory chip is commonly referred to as its access time

MEMORY ADDRESS DECODING

The CPU provides the address of the data desired, but it is the job of the decoding circuitry to locate the selected memory block

- Memory chips have one or more pins called CS (chip select), which must be activated for the memory's contents to be accessed
- Sometimes the chip select is also referred to as chip enable (CE)

In connecting a memory chip to the CPU, note the following points

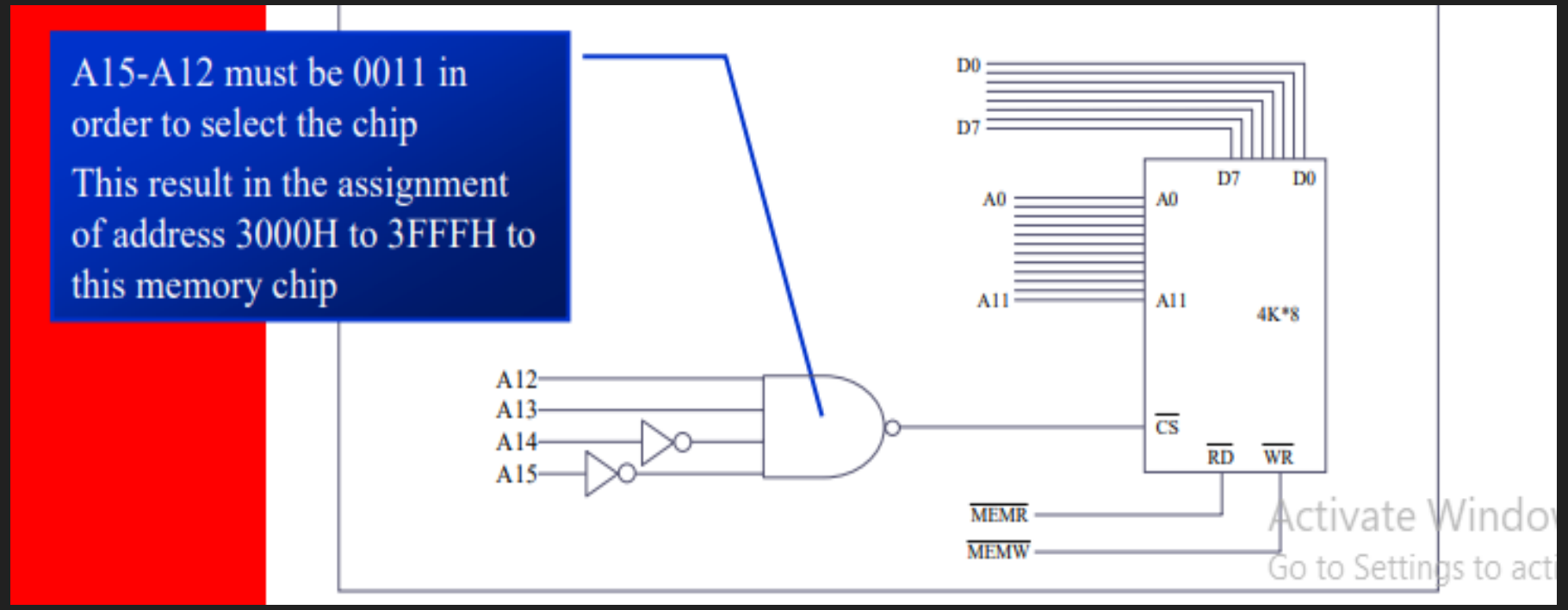
- The data bus of the CPU is connected directly to the data pins of the memory chip
- Control signals RD (read) and WR (memory write) from the CPU are connected to the OE (output enable) and WE (write enable) pins of the memory chip
- In the case of the address buses, while the lower bits of the address from the CPU go directly to the memory chip address pins, the upper ones are used to activate the CS pin of the memory chip

Normally memories are divided into blocks and the output of the decoder selects a given memory block

- Using simple logic gates
- Using the 74LS138
- Using programmable logics

The simplest way of decoding circuitry is the use of NAND or other gates

- The fact that the output of a NAND gate is active low, and that the CS pin is also active low makes them a perfect match



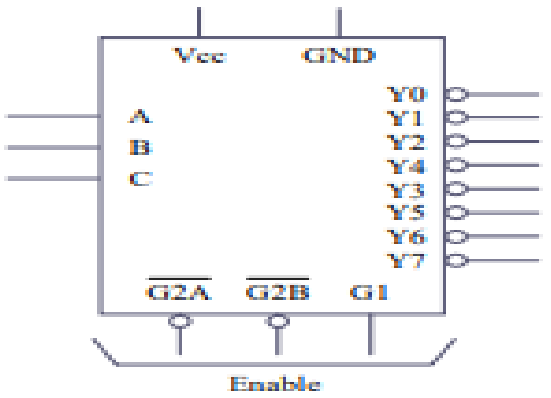
Using 74LS138 3-8 Decoder

- This is one of the most widely used address decoders
- The 3 inputs A, B, and C generate 8 active low outputs Y0 – Y7

Each Y output is connected to CS of a memory chip, allowing control of 8 memory blocks by a single 74LS138

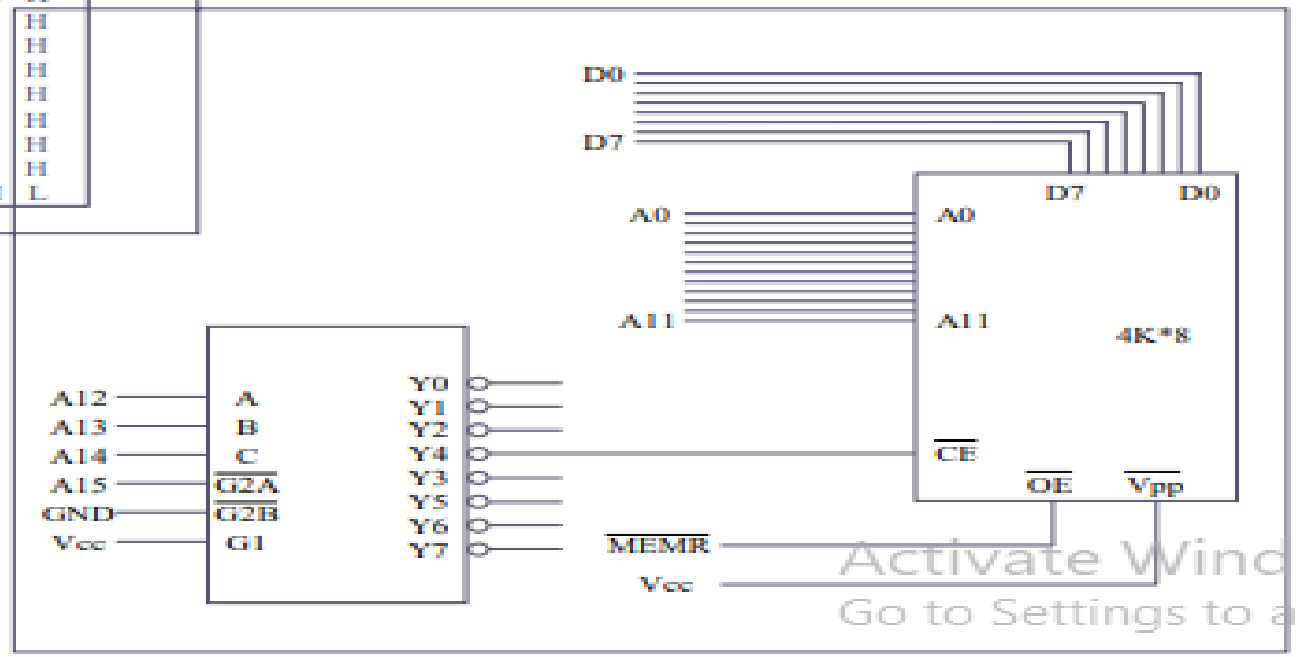
- In the 74LS138, where A, B, and C select which output is activated, there are three additional inputs, G2A, G2B, and G1
- G2A and G2B are both active low, and G1 is active high
- If any one of the inputs G1, G2A, or G2B is not connected to an address signal, they must be activated permanently either by Vcc or ground, depending on the activation level

74LS138 Decoder



Function Table

Inputs			Outputs							
Enable		Select								
G1	G2	C B A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X X X	H	H	H	H	H	H	H	H
L	X	X X X	H	H	H	H	H	H	H	H
H	L	L L L	L	H	H	H	H	H	H	H
H	L	L L H	H	L	H	H	H	H	H	H
H	L	L H L	H	H	L	H	H	H	H	H
H	L	L H H	H	H	H	L	H	H	H	H
H	L	H L L	H	H	H	H	L	H	H	H
H	L	H L H	H	H	H	H	H	L	H	H
H	L	H H L	H	H	H	H	H	H	L	H
H	L	H H H	H	H	H	H	H	H	H	L



Using Programmable Logic

- Other widely used decoders are programmable logic chips such as PAL and GAL chips
- One disadvantage of these chips is that one must have access to a PAL/GAL software and burner, whereas the 74LS138 needs neither of these
- The advantage of these chips is that they are much more versatile since they can be programmed for any combination of address ranges

8051 Programming in C

Why program the 8051 in C?

Compilers produce hex files that is downloaded to ROM of microcontroller

- Easy and less time consuming
- Modification
- Function libraries
- Portable

Data types

- A good understanding of C data types for 8051 can help programmers to create smaller hex files
- Unsigned char
- Signed char
- Unsigned int
- Signed int
- Sbit (single bit)
- Bit and sfr

Unsigned Char

- The character data type is the most natural choice
- 8051 is an 8-bit microcontroller
- Unsigned char is an 8-bit data type in the range of 0 – 255 (00 – FFH)
- One of the most widely used data types for the 8051
  - Counter value
  - ASCII characters
- C compilers use the signed char as the default if we do not put the keyword unsigned

Write an 8051 C program to send values 00 – FF to port P1.

Solution:

```
#include <reg51.h>
void main(void)
   {
      unsigned char z;
      for (z=0;z<=255;z++)
         P1=z;
   }
```

1. Pay careful attention to the size of the data
2. Try to use unsigned *char* instead of *int* if possible

signed char

The signed char is an 8-bit data type

- Use the MSB D7 to represent – or +
- Give us values from –128 to +127
- We should stick with the unsigned char unless the data needs to be represented as signed number
- Eg: Temperature

Write an 8051 C program to send values of –4 to +4 to port P1.

Solution:

```
//Singed numbers
#include <reg51.h>
void main(void)
   {
      char mynum[]={+1,-1,+2,-2,+3,-3,+4,-4};
      unsigned char z;
      for (z=0;z<=8;z++)
         P1=mynum[z];
   }
```

Unsigned int

The unsigned int is a 16-bit data type

- Takes a value in the range of 0 to 65535 (0000 – FFFFH)
- Define 16-bit variables such as memory addresses
- Set counter values of more than 256
- Since registers and memory accesses are in 8-bit chunks, the misuse of int variables will result in a larger hex file

Signed int

Signed int is a 16-bit data type

- Use the MSB D15 to represent – or +
- We have 15 bits for the magnitude of the number from –32768 to +32767

Sbit

- To access single-bit addressable register
- To access to single bits of SFR registers
- To access individual bits of Port

Write an 8051 C program to toggle bit D0 of the port P1 (P1.0) 50,000 times.

Solution:

sbit keyword allows access to the single bits of the SFR registers

```
#include <reg51.h>
sbit MYBIT=P1^0;

void main(void)
  {
    unsigned int z;
    for (z=0;z<=50000;z++)
       {
         MYBIT=0;
         MYBIT=1;
       }
  }
```

Bit and sfr

- The bit data type allows access to single bits of bit-addressable memory spaces 20 – 2FH
- To access the byte-size SFR registers, we use the sfr data type

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
(signed) char	8-bit	-128 to +127
unsigned int	16-bit	0 to 65535
(signed) int	16-bit	-32768 to +32767
sbit	1-bit	SFR bit-addressable only
bit	1-bit	RAM bit-addressable only
sfr	8-bit	RAM addresses 80 – FFH only

TIME DELAY

There are two way s to create a time delay in 8051 C

- Using the 8051 timer
- Using a simple for loop

three factors that can affect the accuracy of the delay

- The 8051 design –
  - The number of machine cycle
  - The number of clock periods per machine cycle
- The crystal frequency connected to the X1 – X2 input pins
- Compiler choice – C compiler converts the C statements and functions to Assembly language instructions – Different compilers produce different code

Write an 8051 C program to toggle bits of P1 continuously forever with some delay.

Solution:

```
//Toggle P1 forever with some delay in between
//"on" and "off"
#include <reg51.h>
void main(void)
   {
     unsigned int x;
     for (;;)                    //repeat forever
        {
          p1=0x55;
          for (x=0;x<40000;x++); //delay size
                                 //unknown
          p1=0xAA;
          for (x=0;x<40000;x++);
        }
   }
```

We must use the oscilloscope to measure the exact duration

Write an 8051 C program to toggle bits of P1 ports continuously with a 250 ms.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
   {
      while (1)                          //repeat forever
         {
            p1=0x55;
            MSDelay(250);
            p1=0xAA;
            MSDelay(250);
         }
   }

void MSDelay(unsigned int itime)
   {
      unsigned int i,j;
      for (i=0;i<itime;i++)
         for (j=0;j<1275;j++);
   }
```

I/O Programming in 8051 C

Byte Size I/O

LEDs are connected to bits P1 and P2. Write an 8051 C program that shows the count from 0 to FFH (0000 0000 to 1111 1111 in binary) on the LEDs.

Solution:

```
#include <reg51.h>
#defind LED P2;

void main(void)
   {
      P1=00;                //clear P1
      LED=0;                //clear P2
      for (;;)              //repeat forever
         {
            P1++;           //increment P1
            LED++;          //increment P2
         }
   }
```

Ports P0 – P3 are byte-accessable and we use the P0 – P3 labels as defined in the 8051/52 header file.

Write an 8051 C program to get a byte of data form P1, wait 1/2 second, and then send it to P2.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);

void main(void)
   {
      unsigned char mybyte;
      P1=0xFF;                    //make P1 input port
      while (1)
          {
            mybyte=P1;            //get a byte from P1
            MSDelay(500);
            P2=mybyte;            //send it to P2
          }
   }
```

Write an 8051 C program to get a byte of data form P0. If it is less than 100, send it to P1; otherwise, send it to P2.

Solution:

```
#include <reg51.h>

void main(void)
   {
      unsigned char mybyte;
      P0=0xFF;                    //make P0 input port
      while (1)
          {
            mybyte=P0;            //get a byte from P0
            if (mybyte<100)
                P1=mybyte;        //send it to P1
            else
                P2=mybyte;        //send it to P2
          }
   }
```

- Bit-addressable I/O

Write an 8051 C program to toggle only bit P2.4 continuously without disturbing the rest of the bits of P2.

Solution:

```
//Toggling an individual bit
#include <reg51.h>
sbit mybit=P2^4;

void main(void)
   {
      while (1)
          {
             mybit=1;                    //turn on P2.4
             mybit=0;                    //turn off P2.4
          }
   }
```

Ports P0 – P3 are bit-addressable and we use *sbit* data type to access a single bit of P0 - P3

Use the Px^y format, where x is the port 0, 1, 2, or 3 and y is the bit 0 – 7 of that port

Write an 8051 C program to monitor bit P1.5. If it is high, send 55H to P0; otherwise, send AAH to P2.

Solution:

```
#include <reg51.h>
sbit mybit=P1^5;

void main(void)
   {
      mybit=1;                    //make mybit an input
      while (1)
          {
             if (mybit==1)
                 P0=0x55;
             else
                 P2=0xAA;
          }
   }
```

A door sensor is connected to the P1.1 pin, and a buzzer is connected to P1.7. Write an 8051 C program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
sbit Dsensor=P1^1;
sbit Buzzer=P1^7;

void main(void)
   {
      Dsensor=1;                    //make P1.1 an input
      while (1)
          {
             while (Dsensor==1)//while it opens
                {
                   Buzzer=0;
                   MSDelay(200);
                   Buzzer=1;
                   MSDelay(200);
                }
          }
   }
```

The data pins of an LCD are connected to P1. The information is latched into the LCD whenever its Enable pin goes from high to low. Write an 8051 C program to send "The Earth is but One Country" to this LCD.

Solution:

```
#include <reg51.h>
#define LCDData P1    //LCDData declaration
sbit En=P2^0;         //the enable pin

void main(void)
   {
      unsigned char message[]
                    ="The Earth is but One Country";
      unsigned char z;
      for (z=0;z<28;z++) //send 28 characters
          {
             LCDData=message[z];
             En=1;      //a high-
             En=0;      //-to-low pulse to latch data
          }
   }
```

Write an 8051 C program to turn bit P1.5 on and off 50,000 times.

Solution:

We can access a single bit of any SFR if we specify the bit address

```
sbit MYBIT=0x95;

void main(void)
  {
    unsigned int z;
    for (z=0;z<50000;z++)
       {
         MYBIT=1;
         MYBIT=0;
       }
  }
```

Write an 8051 C program to get the status of bit P1.0, save it, and send it to P2.7 continuously.

Solution:

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;              //use bit to declare
                         //bit- addressable memory

void main(void)
  {
    while (1)
       {
         membit=inbit;     //get a bit from P1.0
         outbit=membit;    //send it to P2.7
       }
  }
```

We use bit data type to access data in a bit-addressable section of the data RAM space 20 – 2FH

LOGIC OPERATIONS

- **Logical operators**
  - AND (&&), OR (||), and NOT (!)
- **Bit-wise operators**
  - AND (&), OR (|), EX-OR (^), Inverter (~), Shift Right (>>), and Shift Left (<<)
    - These operators are widely used in software engineering for embedded systems and control

Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

Write an 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay. Using the inverting and Ex-OR operators, respectively.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);

void main(void)
   {
     P0=0x55;
     P2=0x55;
     while (1)
         {
            P0=~P0;
            P2=P2^0xFF;
            MSDelay(250);
         }
   }
```

Write an 8051 C program to get bit P1.0 and send it to P2.7 after inverting it.

Solution:

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;

void main(void)
  {
    while (1)
       {
         membit=inbit;    //get a bit from P1.0
         outbit=~membit;  //invert it and send
                           //it to P2.7
       }
  }
```

Packed BCD to ASCII Conversion

Write an 8051 C program to convert packed BCD 0x29 to ASCII and display the bytes on P1 and P2.

Solution:

```
#include <reg51.h>

void main(void)
   {
     unsigned char x,y,z;
     unsigned char mybyte=0x29;
     x=mybyte&0x0F;
     P1=x|0x30;
     y=mybyte&0xF0;
     y=y>>4;
     P2=y|0x30;
   }
```

ASCII to Packed BCD Conversion

Write an 8051 C program to convert ASCII digits of '4' and '7' to packed BCD and display them on P1.

Solution:

```
#include <reg51.h>

void main(void)
   {
     unsigned char bcdbyte;
     unsigned char w='4';
     unsigned char z='7';
     w=w&0x0F;
     w=w<<4;
     z=z&0x0F;
     bcdbyte=w|z;
     P1=bcdbyte;
   }
```

Write an 8051 C program to calculate the checksum byte for the data 25H, 62H, 3FH, and 52H.

Solution:

```
#include <reg51.h>

void main(void)
   {
     unsigned char mydata[]={0x25,0x62,0x3F,0x52};
     unsigned char sum=0;
     unsigned char x;
     unsigned char chksumbyte;
     for (x=0;x<4;x++)
        {
          P2=mydata[x];
          sum=sum+mydata[x];
          P1=sum;
        }
     chksumbyte=~sum+1;
     P1=chksumbyte;
   }
```

Checksum Byte in ROM

Write an 8051 C program to perform the checksum operation to ensure data integrity. If data is good, send ASCII character 'G' to P0. Otherwise send 'B' to P0.

Solution:

```
#include <reg51.h>

void main(void)
   {
      unsigned char mydata[]
                          ={0x25,0x62,0x3F,0x52,0xE8};
      unsigned char shksum=0;
      unsigned char x;
      for (x=0;x<5;x++)
         chksum=chksum+mydata[x];
      if (chksum==0)
         P0='G';
      else
         P0='B';
}
```