CHAPTER-1

INTRODUCTION TO MICROPROCESSOR

TECHNICAL AND VOCATIONAL TRAINING INSTITUTE
Department of Electrical electronics technology

Fundamentals of microprocessors/microcontroller

EETe 3032

By Zemenu T. /2022

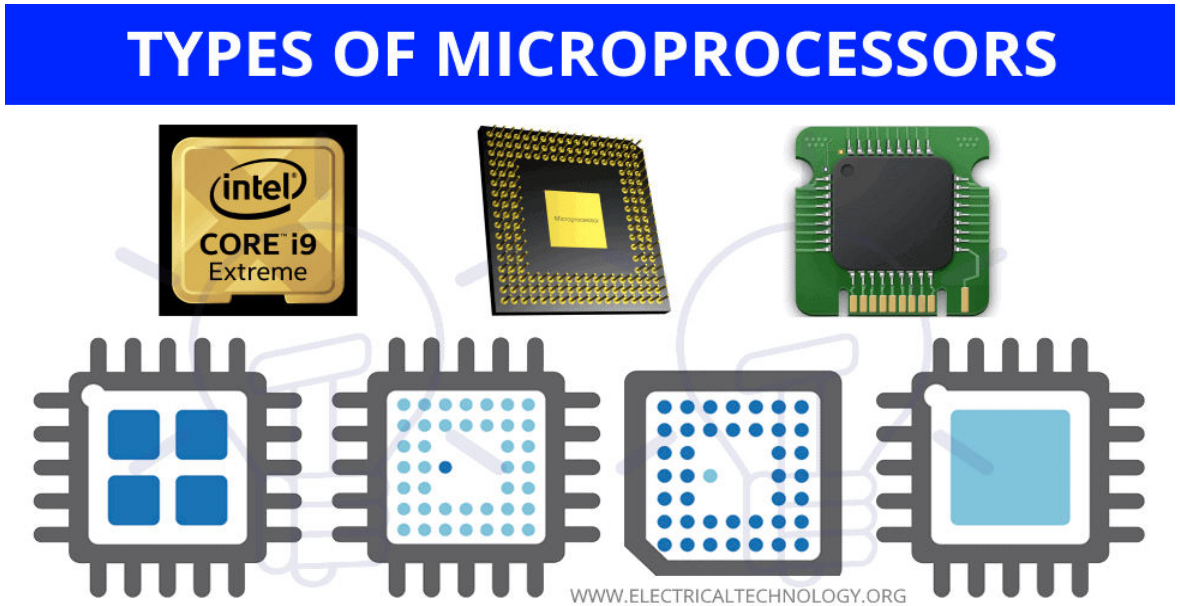
Introduction of Microprocessor

A Microprocessor is an important part of a computer architecture without which you will not be able to perform anything on your computer. It is a programmable device that takes in input performs some arithmetic and logical operations over it and produces the desired output. In simple words, a Microprocessor is a digital device on a chip that can fetch instructions from memory, decode and execute them and give results.

Microprocessor is the brain of computer, which does all the work. It is a computer processor that incorporates all the functions of CPU (Central Processing Unit) on a single IC (Integrated Circuit) or at the most a few ICs. Microprocessors were first introduced in early 1970s. 4004 was the first general purpose microprocessor used by Intel in building personal computers. Arrival of low cost general purpose microprocessors has been instrumental in development of modern society the way it has.

What is a Microprocessor?

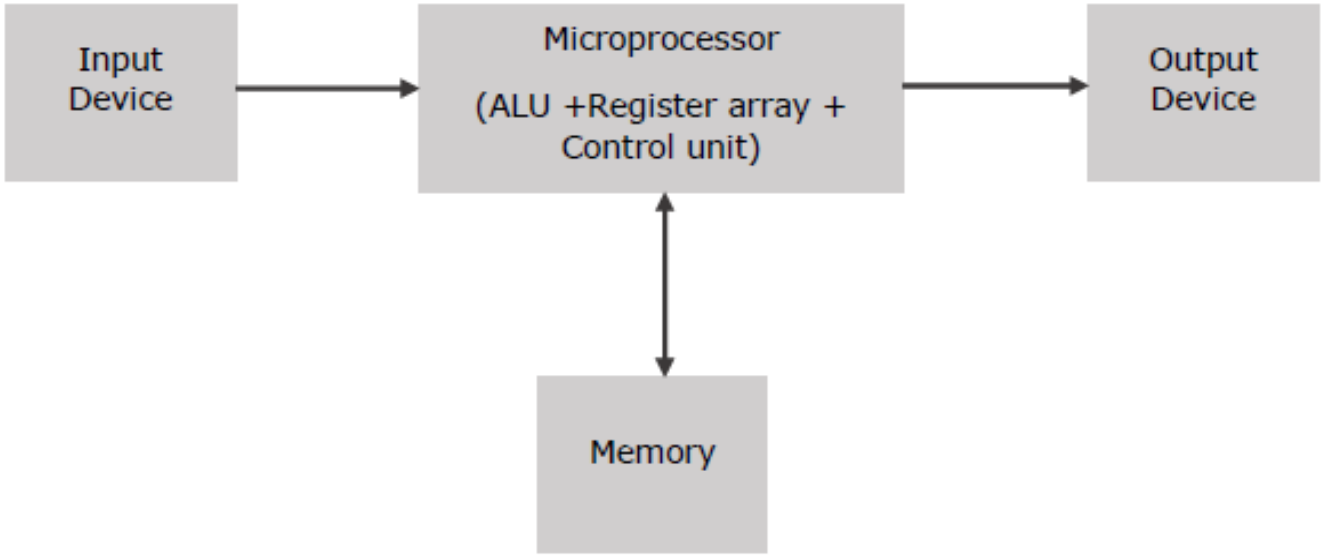
A microprocessor is a central processing unit or the brain of a computer inside a single Integrated circuit (IC). It is made up of millions of semiconductor transistors, diodes & resistors and it is responsible for any arithmetic or logical operation. It is a digital device capable of processing any binary data given to it.
A general-purpose microprocessor consists of **ALU** (arithmetic logic unit), **control unit** and **register array**. The ALU's purpose is to apply any logical or arithmetic operation on the data provided by the computer memory. The microprocessor fetches the instruction from memory and executes it sequentially. When the instruction is complete, it sends the resultant data through the output in a binary digital form.



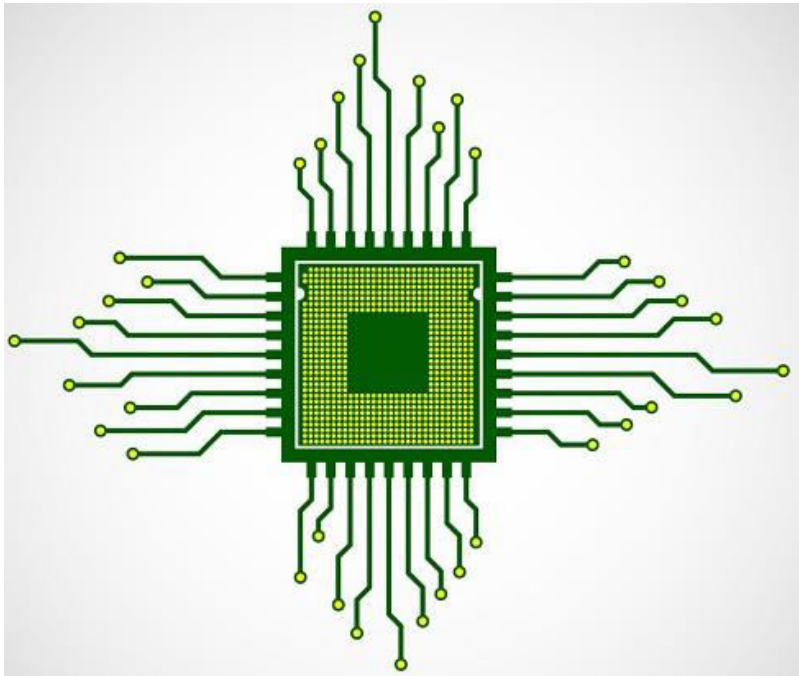
Basics of Microprocessor

A Microprocessor takes a bunch of instructions in machine language and executes them, telling the processor what it has to do. Microprocessor performs three basic things while executing the instruction:

1. It performs some basic operations like addition, subtraction, multiplication, division, and some logical operations using its Arithmetic and Logical Unit (ALU). New Microprocessors also perform operations on floating-point numbers also.
2. Data in microprocessors can move from one location to another.
3. It has a Program Counter (PC) register that stores the address of the next instruction based on the value of the PC, Microprocessor jumps from one location to another and takes decisions.



Block Diagram of a Basic Microcomputer



Evolution of Microprocessors

Transistor was invented in 1948 (23 December 1947 in Bell lab). IC was invented in 1958 (Fair Child Semiconductors) By Texas Instruments J Kilby. The first microprocessor was invented by INTEL(INTegrated Electronics).

Name	Year of Invention	Clock speed	Number of transistors	Inst. per sec
Size of the microprocessor – 4 bit				
INTEL 4004/4040	1971 by Ted Hoff and Stanley Mazor	740 kHz	2300	60,000
Size of the microprocessor – 8 bit				
8008	1972	500 kHz	3500	50,000
8080	1974	2 MHz	6000	10 times faster than 8008
8085	1976 (16-bit address bus)	3 MHz	6500	769230
Size of the microprocessor – 16 bit				
8086	1978 (multiply and divide instruction, 16-bit data bus and 20-bit address bus)	4.77 MHz, 8 MHz, 10 MHz	29000	2.5 Million
8088	1979 (cheaper version of 8086 and 8-bit external bus)			2.5 Million
80186/80188	1982 (80188 cheaper version of 80186, and additional components like interrupt controller, clock generator, local bus controller, counters)	6 MHz		
80286	1982 (data bus 16bit and address bus 24 bit)	8 MHz	134000	4 Million

Evolution of Microprocessors (Contd.)

Size of the microprocessor – 32 bit				
INTEL 80386	1986 (other versions 80386DX, 80386SX, 80386SL , and data bus 32-bit address bus 32 bit)	16 MHz – 33 MHz	275000	
INTEL 80486	1986 (other versions 80486DX, 80486SX, 80486DX2, 80486DX4)	16 MHz – 100 MHz	1.2 Million transistors	8 KB of cache memory
PENTIUM	1993	66 MHz		Cache memory 8 bit for instructions 8 bit for data
Size of the microprocessor – 64 bit				
INTEL core 2	2006 (other versions core2 duo, core2 quad, core2 extreme)	1.2 GHz to 3 GHz	291 Million transistors	64 KB of L1 cache per core 4 MB of L2 cache
i3, i5, i7	2007, 2009, 2010	2.2GHz – 3.3GHz, 2.4GHz – 3.6GHz, 2.93GHz – 3.33GHz		

Microprocessors Characteristics

Microprocessors are multipurpose devices that can be designed for generic or specialized functions. The microprocessors of laptops and smart phones are general purpose whereas ones designed for graphical processing or machine vision are specialized ones. There are some characteristics that are common to all microprocessors.

These are the most important defining characteristics of a microprocessor −

- Clock speed
- Instruction set
- Word size

Clock Speed

Every microprocessor has an **internal clock** that regulates the speed at which it executes instructions and also synchronizes it with other components. The speed at which the microprocessor executes instructions is called **clock speed**. Clock speeds are measured in MHz or GHz where 1 MHz means 1 million cycles per second whereas 1 GHz equals to 1 billion cycles per second. Here cycle refers to single electric signal cycle.

Currently microprocessors have clock speed in the range of 3 GHz, which is maximum that current technology can attain. Speeds more than this generate enough heat to damage the chip itself. To overcome this, manufacturers are using multiple processors working in parallel on a chip.

Word Size

Number of bits that can be processed by a processor in a single instruction is called its **word size**. Word size determines the amount of RAM that can be accessed at one go and total number of pins on the microprocessor. Total number of input and output pins in turn determines the architecture of the microprocessor.

First commercial microprocessor Intel 4004 was a 4-bit processor. It had 4 input pins and 4 output pins. Number of output pins is always equal to the number of input pins. Currently most microprocessors use 32-bit or 64-bit architecture.

Instruction Set

A command given to a digital machine to perform an operation on a piece of data is called an **instruction**. Basic set of machine level instructions that a microprocessor is designed to execute is called its **instruction set**. These instructions do carry out these types of operations −

- Data transfer
- Arithmetic operations
- Logical operations
- Control flow
- Input/output and machine control

Microprocessor Components

Compared to the first microprocessors, today's processors are very small but still they have these basic parts right from the first model −

- CPU
- Bus
- Memory

CPU

CPU is fabricated as a very large scale integrated circuit (VLSI) and has these parts −

- **Instruction register** − It holds the instruction to be executed.
- **Decoder** − It decodes (converts to machine level language) the instruction and sends to the ALU (Arithmetic Logic Unit).
- **ALU** − It has necessary circuits to perform arithmetic, logical, memory, register and program sequencing operations.
- **Register** − It holds intermediate results obtained during program processing. Registers are used for holding such results rather than RAM because accessing registers is almost 10 times faster than accessing RAM.

BUS

Connection lines used to connect the internal parts of the microprocessor chip is called bus. There are three types of buses in a microprocessor −

•**Data Bus** − Lines that carry data to and from memory are called data bus. It is a bidirectional bus with width equal to word length of the microprocessor.

•**Address Bus** − It is a unidirectional responsible for carrying address of a memory location or I/O port from CPU to memory or I/O port.

•**Control Bus** − Lines that carry control signals like **clock signals, interrupt signal** or **ready signal** are called control bus. They are bidirectional. Signal that denotes that a device is ready for processing is called **ready signal**. Signal that indicates to a device to interrupt its process is called an **interrupt signal**.

Memory

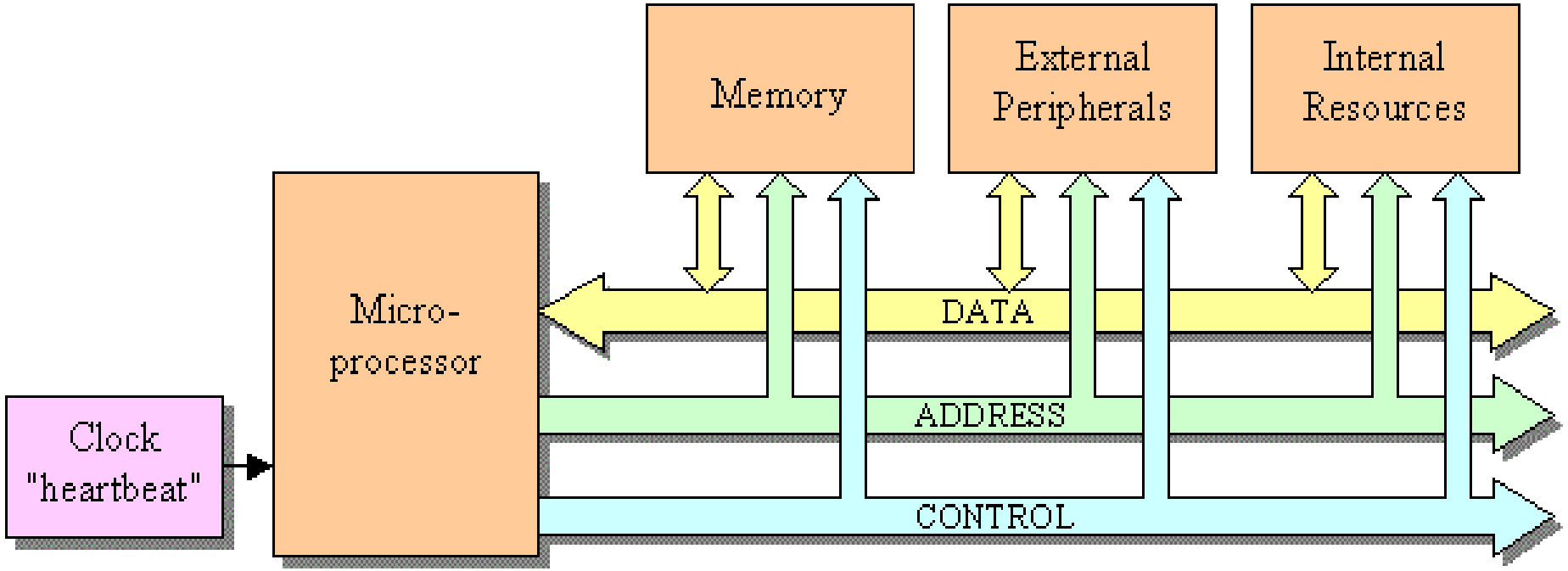
Microprocessor has two types of memory

•**RAM** − Random Access Memory is volatile memory that gets erased when power is switched off. All data and instructions are stored in RAM.

•**ROM** − Read Only Memory is non-volatile memory whose data remains intact even after power is switched off. Microprocessor can read from it any time it wants but cannot write to it. It is preprogrammed with most essential data like booting sequence by the manufacturer.

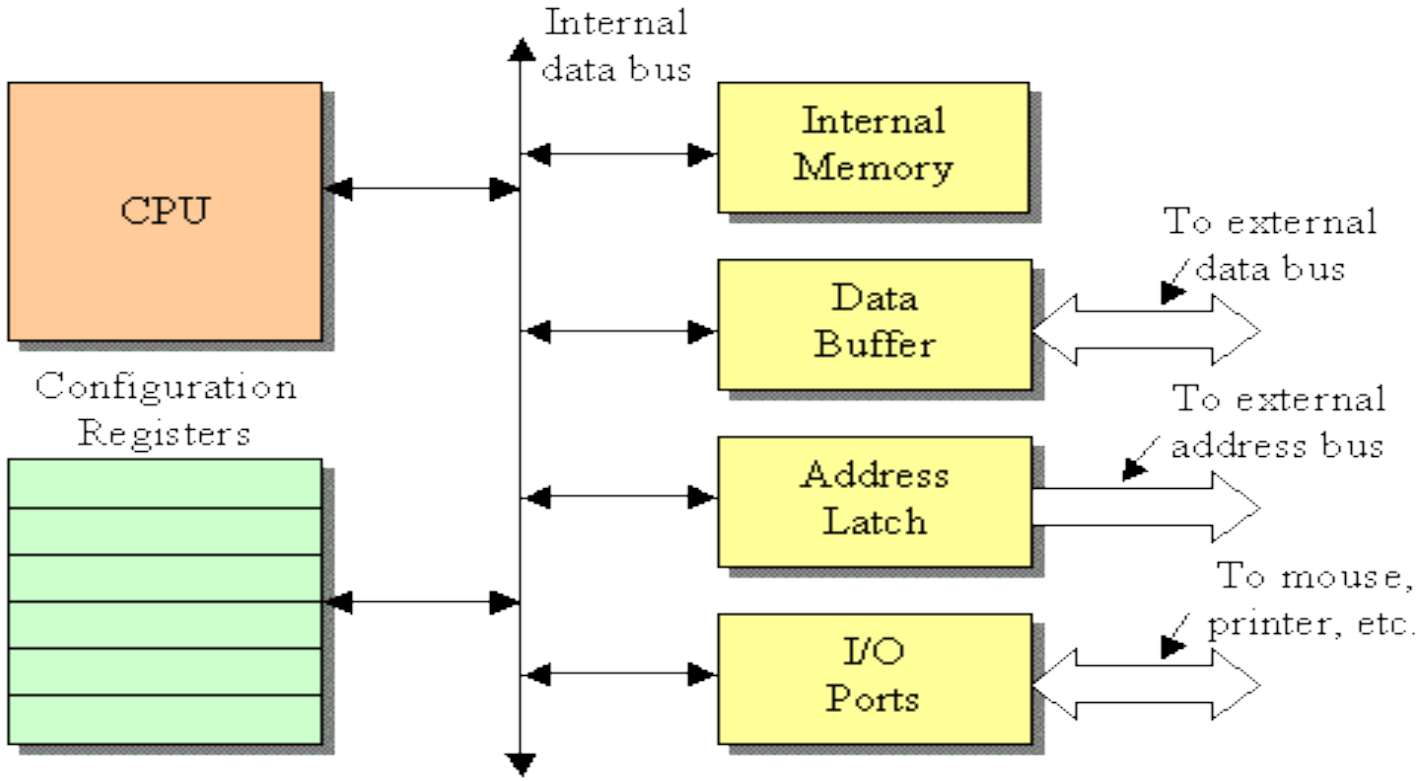
Microprocessor Architecture

Below is the generic block diagram of a microprocessor system. It consists of the bus. This bus includes the data that can be passed bidirectional between the microprocessor and its external resources such as memory, address lines that define the chip and the memory location within the chip of the external resources, and the control lines orchestrating the transfers.



Microprocessor Architecture (Contd.)

The internals of the microprocessor are essentially a microcosm of the microprocessor system shown above. As shown in the figure below, it too has a data bus that passes data between the CPU, the internal memory, the data buffer, the address latch, I/O ports, and the configuration registers. The difference is that this bus is contained entirely within the processor.



Microprocessor Architecture (Contd.)

•Internal memory is small, but extremely quick memory. It's used for any internal computations that need to be done fast without the added overhead of writing to external memory.

•**Data buffer** allows for data to be passed to the external data bus which then goes onto devices such as memory. The data buffer takes care of data direction (is the data coming in or going out) and also provides signal conditioning for the world outside the chip.

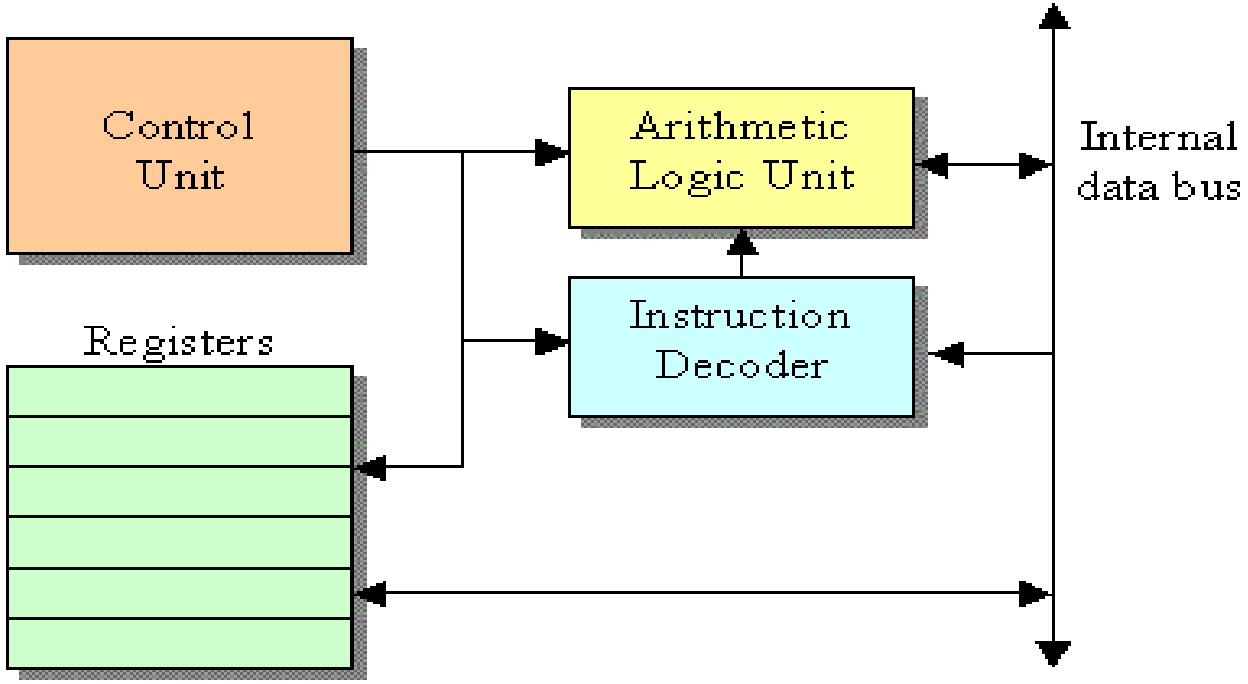
•**Address latch** maintains the address for use by the external devices such as memory.

•**I/O ports** are specialized ports used by the processor. It differs from memory in that the port has a port number or identification that is used rather than having to use a memory address and pass data back and forth across the bus.

•The configuration registers maintains the current configuration of the microprocessor. For example, if the microprocessor has a serial port (COM port), configuration data might include baud rate, buffer status, idle status, etc.

Microprocessor Architecture (Contd.)

The CPU is then a microcosm of the microprocessor block diagram.



•Control unit is the brains of the CPU. It tells all the other devices what they do and when they should do it.

•Arithmetic Logic Unit (ALU) performs all of the arithmetic and logic functions.

•All instructions are stored as binary values. The instruction decoder reads that value and tells the ALU which computational circuits to energize in order to perform the function.

•The registers are used for all calculations and for maintaining addressing information.

Evaluation of Microprocessor

The first microprocessor introduced in 1971 was a 4-bit microprocessor with 4m 5KB memory and had a set of 45 instructions. In the past 5 decades microprocessor speed has doubled every two years, as predicted by Gordon Moore, Intel co-founder. Current microprocessors can access 64 GB memory. Depending on width of data microprocessors can process, they are of these categories

- 8-bit
- 16-bit
- 32-bit
- 64-bit

Size of instruction set is another important consideration while categorizing microprocessors. Initially, microprocessors had very small instructions sets because complex hardware was expensive as well as difficult to build.

As technology developed to overcome these issues, more and more complex instructions were added to increase functionality of the microprocessor. However, soon it was realized that having large instruction sets was counterproductive as many instructions that were rarely used sat idle on precious memory space. So the old school of thought that supported smaller instruction sets gained popularity.

Application-specific integrated circuit

These processors are application-specific like personal digital assistant computers. They are designed according to proper specifications.

Digital signal multiprocessor

These processors are used to convert signals like analog to digital or digital to analog. The chips of these processors are used in many devices such as RADAR SONAR home theatres etc.

Advantages of the microprocessor

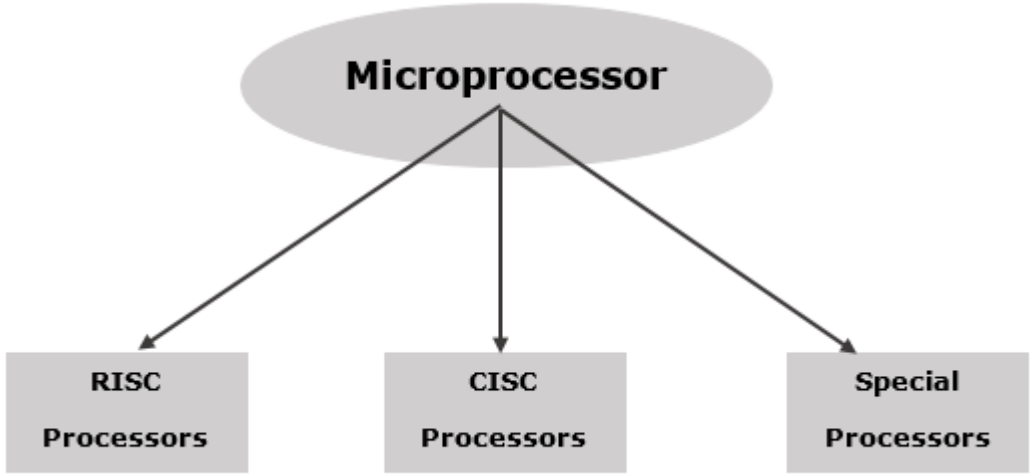
- High processing speed
- Compact size
- Easy maintenance
- Can perform complex mathematics
- Flexible
- Can be improved according to a requirement

Disadvantages of microprocessors

- Overheating occurs due to overuse
- Performance depends on the size of the data
- Large board size than microcontrollers

Microprocessor - Classification

A microprocessor can be classified into three categories



1. RISC Processor

RISC stands for **Reduced Instruction Set Computer**. It is designed to reduce the execution time by simplifying the instruction set of the computer. Using RISC processors, each instruction requires only one clock cycle to execute results in uniform execution time. This reduces the efficiency as there are more lines of code, hence more RAM is needed to store the instructions. The compiler also has to work more to convert high-level language instructions into machine code.

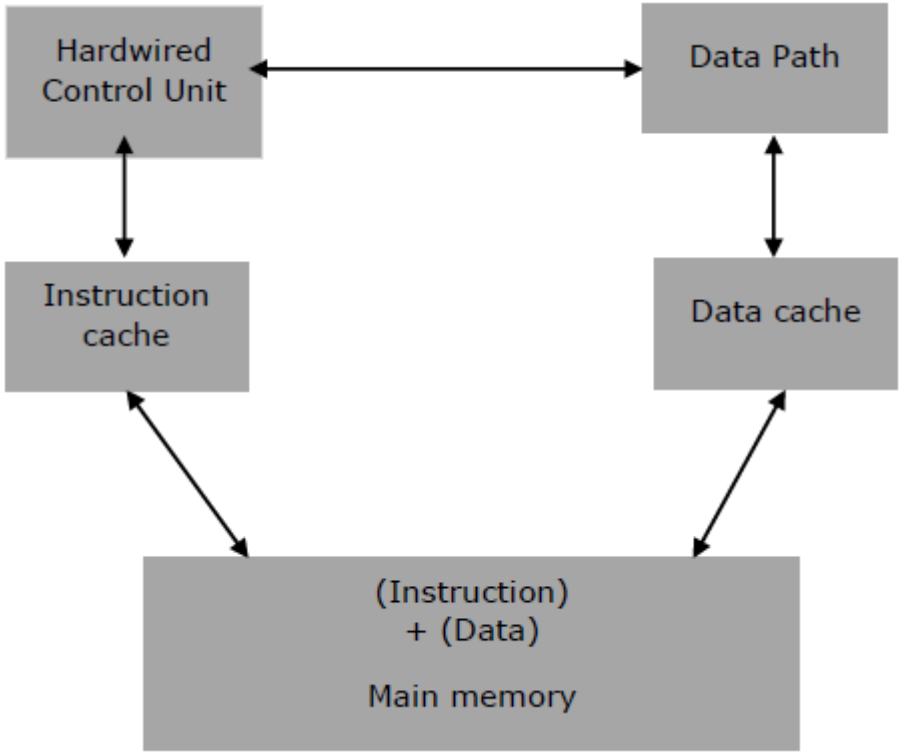
Some of the RISC processors are −

- Power PC: 601, 604, 615, 620
- DEC Alpha: 210642, 211066, 21068, 21164
- MIPS: TS (R10000) RISC Processor

PA-RISC: HP 7100LC

Architecture of RISC

RISC microprocessor architecture uses highly-optimized set of instructions. It is used in portable devices like Apple iPod due to its power efficiency.



Characteristics of RISC

The major characteristics of a RISC processor are as follows

- It consists of simple instructions.
- It supports various data-type formats.
- It utilizes simple addressing modes and fixed length instructions for pipelining.
- It supports register to use in any context.
- One cycle execution time.
- "LOAD" and "STORE" instructions are used to access the memory location.
- It consists of larger number of registers.
- It consists of less number of transistors.

CISC Processor

CISC stands for **Complex Instruction Set Computer**. It is designed to minimize the number of instructions per program, ignoring the number of cycles per instruction. The emphasis is on building complex instructions directly into the hardware.

The compiler has to do very little work to translate a high-level language into assembly level language/machine code because the length of the code is relatively short, so very little RAM is required to store the instructions.

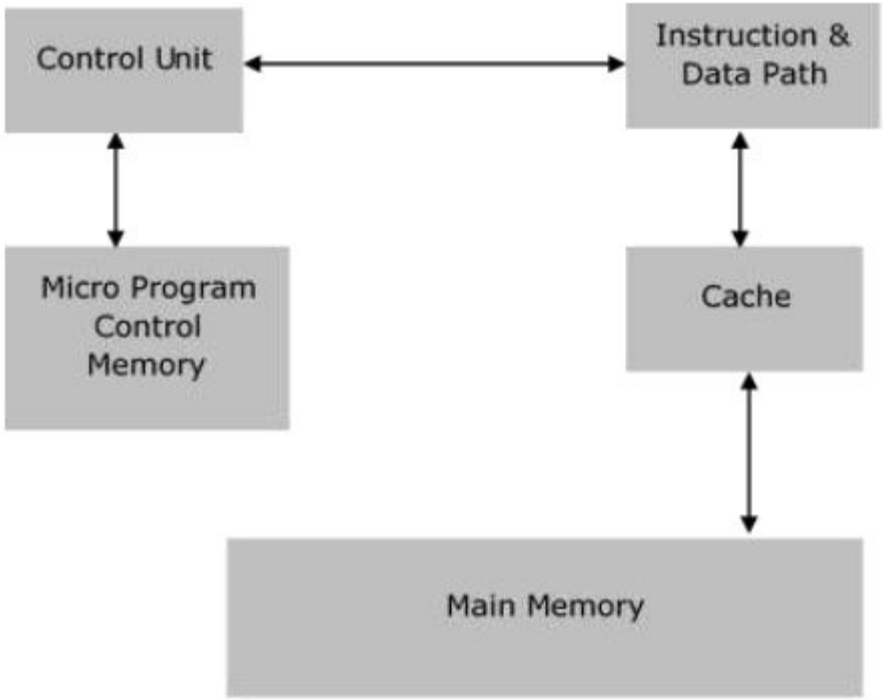
Some of the CISC Processors are −

- IBM 370/168
- VAX 11/780
- Intel 80486

Architecture of CISC

Its architecture is designed to decrease the memory cost because more storage is needed in larger programs resulting in higher memory cost. To resolve this, the number of instructions per program can be reduced by embedding the number of operations in a single instruction.

Architecture of CISC (Contd.)



Characteristics of CISC

- Variety of addressing modes.
- Larger number of instructions.
- Variable length of instruction formats.
- Several cycles may be required to execute one instruction.
- Instruction-decoding logic is complex.
- One instruction is required to support multiple addressing modes.

Special Processors

These are the processors which are designed for some special purposes. Few of the special processors are discussed below

Coprocessor

A coprocessor is a specially designed microprocessor, which can handle its particular function many times faster than the ordinary microprocessor.

For example − Math Coprocessor.

Some Intel math-coprocessors are −

- 8087-used with 8086
- 80287-used with 80286
- 80387-used with 80386

Input/ Output Processor

It is a specially designed microprocessor having a local memory of its own, which is used to control I/O devices with minimum CPU involvement.

For example

- DMA (direct Memory Access) controller
- Keyboard/mouse controller
- Graphic display controller
- SCSI port controller

Transputer (Transistor Computer)

A transputer is a specially designed microprocessor with its own local memory and having links to connect one transputer to another transputer for inter-processor communications. It was first designed in 1980 by nmos and is targeted to the utilization of VLSI technology.

A transputer can be used as a single processor system or can be connected to external links, which reduces the construction cost and increases the performance.

For example − 16-bit T212, 32-bit T425, the floating point (T800, T805 & T9000) processors.

DSP (Digital Signal Processor)

This processor is specially designed to process the analog signals into a digital form. This is done by sampling the voltage level at regular time intervals and converting the voltage at that instant into a digital form. This process is performed by a circuit called an analogue to digital converter, A to D converter or ADC.

A DSP contains the following components −

- **Program Memory** − It stores the programs that DSP will use to process data.
- **Data Memory** − It stores the information to be processed.
- **Compute Engine** − It performs the mathematical processing, accessing the program from the program memory and the data from the data memory.
- **Input/ Output** − It connects to the outside world.

Its applications are −

- Sound and music synthesis
- Audio and video compression
- Video signal processing
- 2D and 3d graphics acceleration.

For example − Texas Instrument's TMS 320 series, e.g., TMS 320C40, TMS320C50

8085 - Microprocessor

8085 is pronounced as "eighty-eighty-five" microprocessor. **It is an 8-bit microprocessor** designed by Intel in 1977 using NMOS technology.

It has the following configuration −

- 8-bit data bus
- 16-bit address bus, which can address up to 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor – Functional Units

8085 consists of the following functional units −

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops −

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Instruction register and decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits −

•Control Signals: READY, RD′, WR′, ALE

•Status Signals: S0, S1, IO/M′

•DMA Signals: HOLD, HLDA

•RESET Signals: RESET IN, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

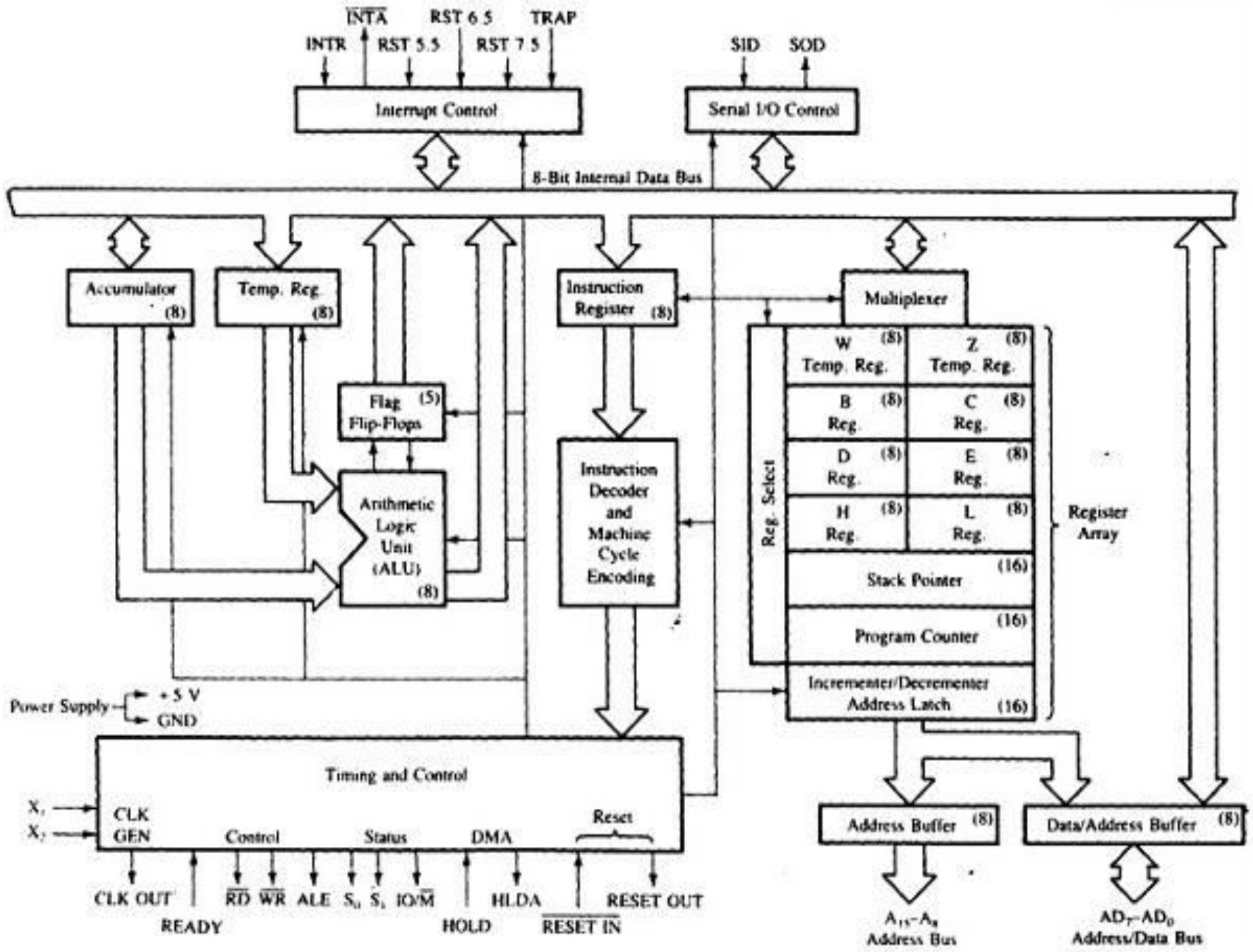
Address buffer and address-data buffer

The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

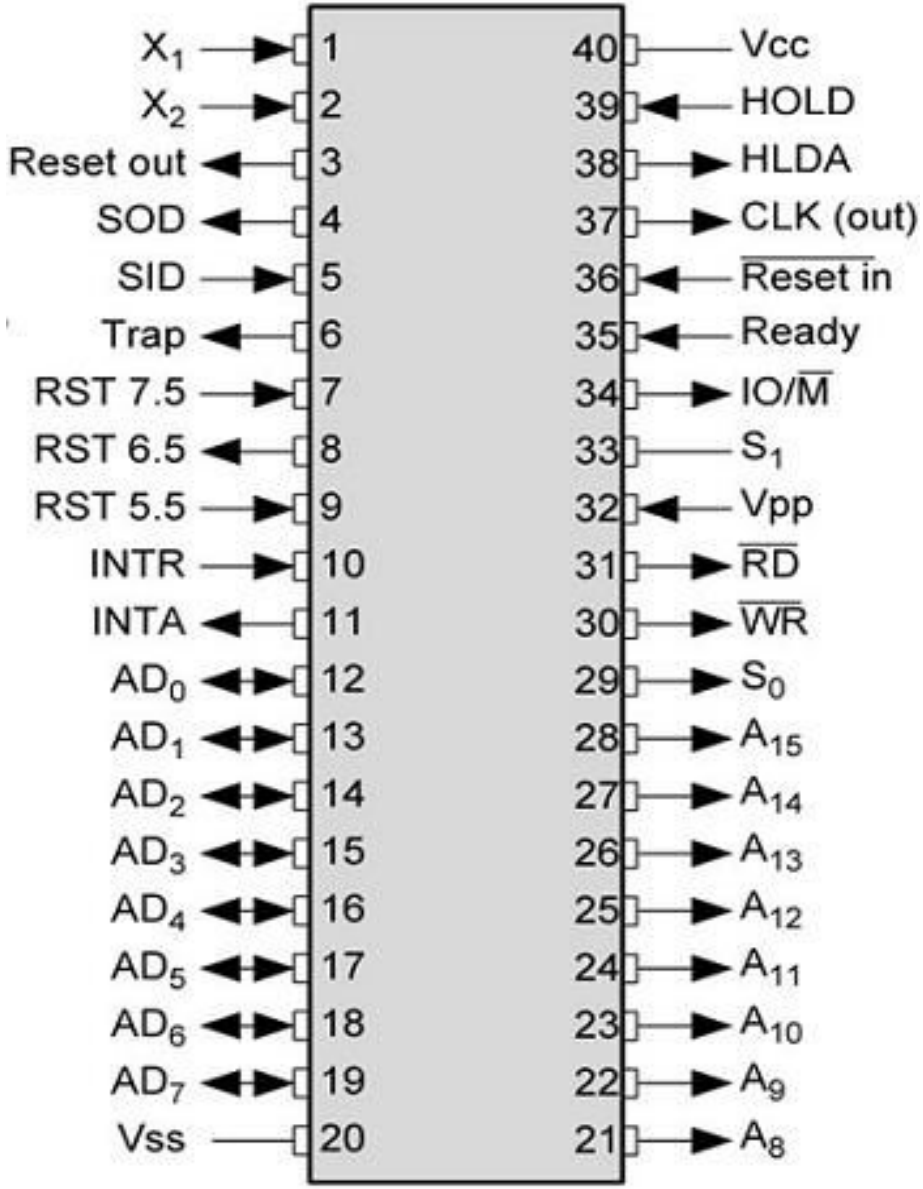
Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

8085 Architecture



Microprocessor - 8085 Pin Configuration



The pins of a 8085 microprocessor can be classified into seven groups

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signal and **3 status signals.**

Three control signals are RD, WR & ALE.

- **RD** − This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** − This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE** − It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.

Power supply

There are 2 power supply signals − VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

Clock signals

There are 3 clock signals, i.e. X1, X2, CLK OUT.

•**X1, X2** − A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.

•**CLK OUT** − This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.

- **INTA** − It is an interrupt acknowledgment signal.
- **RESET IN** − This signal is used to reset the microprocessor by setting the program counter to zero.
- **RESET OUT** − This signal is used to reset all the connected devices when the microprocessor is reset.
- **READY** − This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- **HOLD** − This signal indicates that another master is requesting the use of the address and data buses.
- **HLDA (HOLD Acknowledge)** − It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

Serial I/O signals

There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

- **SOD** (Serial output data line) − The output SOD is set/reset as specified by the SIM instruction.
- **SID** (Serial input data line) − The data on this line is loaded into accumulator whenever a RIM instruction is executed

8086 - Microprocessor

8086 Microprocessor is an enhanced version of 8085 Microprocessor that was designed by Intel in 1976. **It is a 16-bit Microprocessor** having 20 address lines and16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily.

It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

Features of 8086

The most prominent features of a 8086 microprocessor are as follows −

- It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.
- It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
- It is available in 3 versions based on the frequency of operation −
  - 8086 → 5MHz
  - 8086-2 → 8MHz
  - (c)8086-1 → 10 MHz
- It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.
- Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue.
- Execute stage executes these instructions.
- It has 256 vectored interrupts.

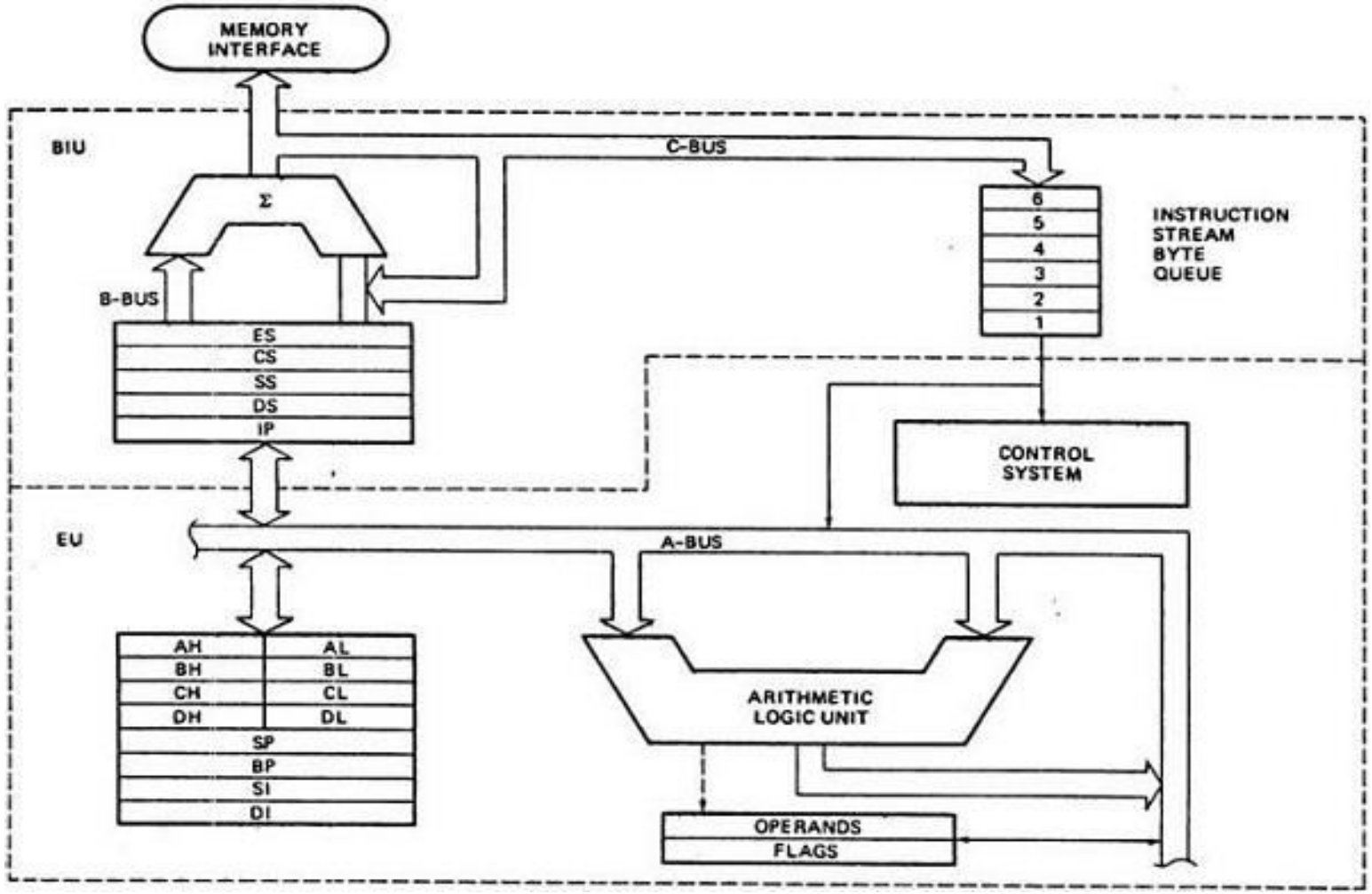
It consists of 29,000 transistors.

Comparison between 8085 & 8086 Microprocessor

- **Size** − 8085 is 8-bit microprocessor, whereas 8086 is 16-bit microprocessor.
- **Address Bus** − 8085 has 16-bit address bus while 8086 has 20-bit address bus.
- **Memory** − 8085 can access up to 64Kb, whereas 8086 can access up to 1 Mb of memory.
- **Instruction** − 8085 doesn’t have an instruction queue, whereas 8086 has an instruction queue.
- **Pipelining** − 8085 doesn’t support a pipelined architecture while 8086 supports a pipelined architecture.
- **I/O** − 8085 can address 2^8 = 256 I/O's, whereas 8086 can access 2^16 = 65,536 I/O's.
- **Cost** − The cost of 8085 is low whereas that of 8086 is high.

Architecture of 8086

The following diagram depicts the architecture of a 8086 Microprocessor



8086 Microprocessor Functional Units

8086 Microprocessor is divided into two functional units, i.e., **EU** (Execution Unit) and **BIU** (Bus Interface Unit).

EU (Execution Unit)

Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions. Its function is to control operations on data using the instruction decoder & ALU. EU has no direct connection with system buses as shown in the above figure, it performs operations over data through BIU.

Let us now discuss the functional parts of 8086 microprocessors.

ALU

It handles all arithmetic and logical operations, like +, −, ×, /, OR, AND, NOT operations.

Flag Register

It is a 16-bit register that behaves like a flip-flop, i.e. it changes its status according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups − Conditional Flags and Control Flags.

Conditional Flags

It represents the result of the last arithmetic or logical instruction executed. Following is the list of conditional flags −

- **Carry flag** − This flag indicates an overflow condition for arithmetic operations.
- **Auxiliary flag** − When an operation is performed at ALU, it results in a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.
- **Parity flag** − This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1’s, then the Parity Flag is set. For odd number of 1’s, the Parity Flag is reset.
- **Zero flag** − This flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.
- **Sign flag** − This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.
- **Overflow flag** − This flag represents the result when the system capacity is exceeded.

Control Flags

Control flags controls the operations of the execution unit. Following is the list of control flags −

•**Trap flag** − It is used for single step control and allows the user to execute one instruction at a time for debugging. If it is set, then the program can be run in a single step mode.

•**Interrupt flag** − It is an interrupt enable/disable flag, i.e. used to allow/prohibit the interruption of a program. It is set to 1 for interrupt enabled condition and set to 0 for interrupt disabled condition.

•**Direction flag** − It is used in string operation. As the name suggests when it is set then string bytes are accessed from the higher memory address to the lower memory address and vice-a-versa.

General purpose register

There are 8 general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be used individually to store 8-bit data and can be used in pairs to store 16bit data. The valid register pairs are AH and AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively.

•**AX register** − It is also known as accumulator register. It is used to store operands for arithmetic operations.

•**BX register** − It is used as a base register. It is used to store the starting base address of the memory area within the data segment.

•**CX register** − It is referred to as counter. It is used in loop instruction to store the loop counter.

DX register − This register is used to hold I/O port address for I/O instruction.

Stack pointer register

It is a 16-bit register, which holds the address from the start of the segment to the memory location, where a word was most recently stored on the stack.

BIU (Bus Interface Unit)

BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory. EU has no direction connection with System Buses so this is possible with the BIU. EU and BIU are connected with the Internal Bus.

It has the following functional parts −

•**Instruction queue** − BIU contains the instruction queue. BIU gets upto 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed.

•Fetching the next instruction while the current instruction executes is called **pipelining**.

•**Segment register** − BIU has 4 segment buses, i.e. CS, DS, SS& ES. It holds the addresses of instructions and data in memory, which are used by the processor to access memory locations. It also contains 1 pointer register IP, which holds the address of the next instruction to executed by the EU.

- **CS** − It stands for **Code Segment**. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.
- **DS** − It stands for **Data Segment**. It consists of data used by the program andis accessed in the data segment by an offset address or the content of other register that holds the offset address.
- **SS** − It stands for **Stack Segment**. It handles memory to store data and addresses during execution.
- **ES** − It stands for **Extra Segment**. ES is additional data segment, which is used by the string to hold the extra destination data.

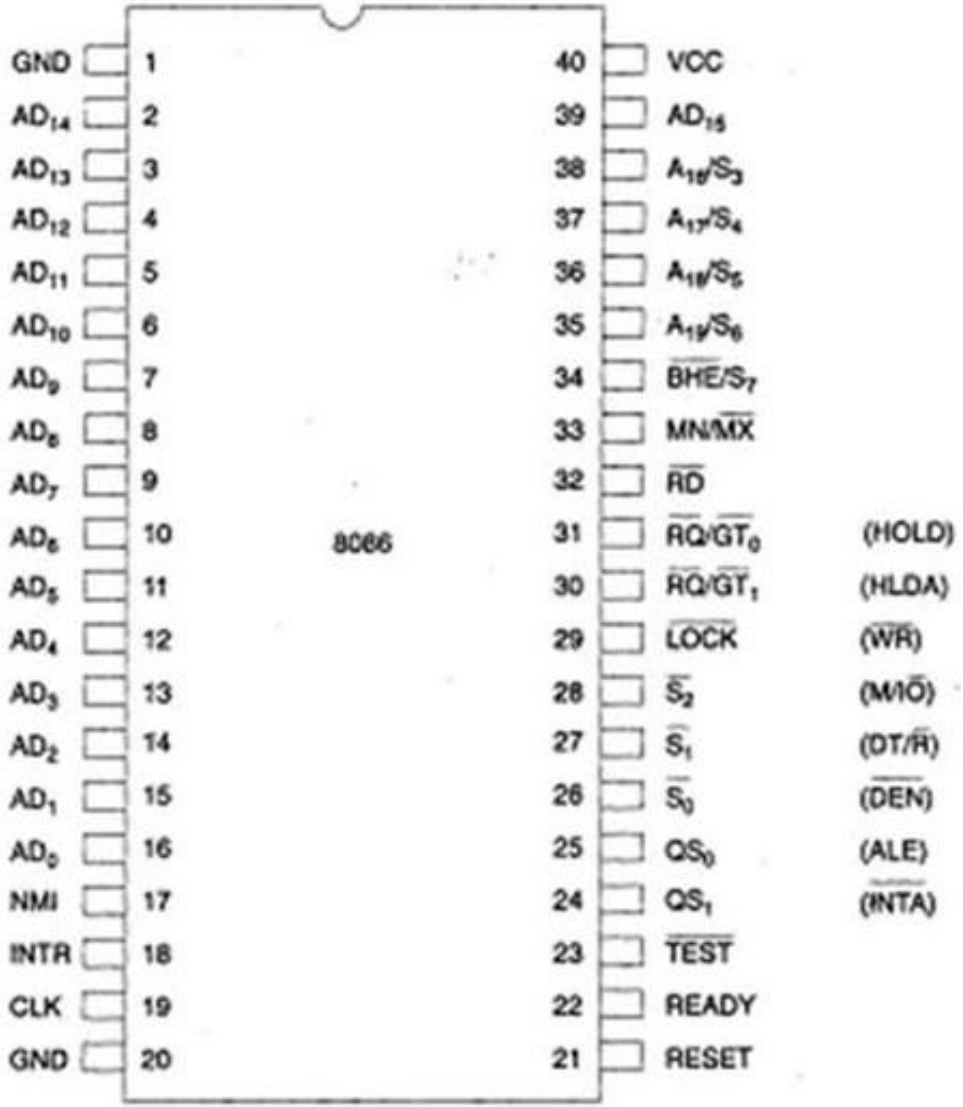
•**Instruction pointer** − It is a 16-bit register used to hold the address of the next instruction to be executed.

8086-Microprocessor Pin Configuration

8086 was the first 16-bit microprocessor available in 40-pin DIP (Dual Inline Package) chip.

8086 Pin Diagram

Here is the pin diagram of 8086 microprocessor



Power supply and frequency signals

It uses 5V DC supply at V_{CC} pin 40, and uses ground at V_{SS} pin 1 and 20 for its operation.

Clock signal

Clock signal is provided through Pin-19. It provides timing to the processor for operations. Its frequency is different for different versions, i.e. 5MHz, 8MHz and 10MHz.

Address/data bus

AD0-AD15. These are 16 address/data bus. AD0-AD7 carries low order byte data and AD8AD15 carries higher order byte data. During the first clock cycle, it carries 16-bit address and after that it carries 16-bit data.

Address/status bus

A16-A19/S3-S6. These are the 4 address/status buses. During the first clock cycle, it carries 4-bit address and later it carries status signals.

S7/BHE

BHE stands for Bus High Enable. It is available at pin 34 and used to indicate the transfer of data using data bus D8-D15. This signal is low during the first clock cycle, thereafter it is active.

Read(*RD)*

It is available at pin 32 and is used to read signal for Read operation.

Ready

It is available at pin 22. It is an acknowledgement signal from I/O devices that data is transferred. It is an active high signal. When it is high, it indicates that the device is ready to transfer data. When it is low, it indicates wait state.

RESET

It is available at pin 21 and is used to restart the execution. It causes the processor to immediately terminate its present activity. This signal is active high for the first 4 clock cycles to RESET the microprocessor.

INTR

It is available at pin 18. It is an interrupt request signal, which is sampled during the last clock cycle of each instruction to determine if the processor considered this as an interrupt or not.

NMI

It stands for non-maskable interrupt and is available at pin 17. It is an edge triggered input, which causes an interrupt request to the microprocessor.

TEST

This signal is like wait state and is available at pin 23. When this signal is high, then the processor has to wait for IDLE state, else the execution continues.

MN/*MX*

It stands for Minimum/Maximum and is available at pin 33. It indicates what mode the processor is to operate in; when it is high, it works in the minimum mode and vice-versa.

INTA

It is an interrupt acknowledgement signal and id available at pin 24. When the microprocessor receives this signal, it acknowledges the interrupt.

ALE

It stands for address enable latch and is available at pin 25. A positive pulse is generated each time the processor begins any operation. This signal indicates the availability of a valid address on the address/data lines.

DEN

It stands for Data Enable and is available at pin 26. It is used to enable Transreceiver 8286. The transreceiver is a device used to separate data from the address/data bus.

DT/R

It stands for Data Transmit/Receive signal and is available at pin 27. It decides the direction of data flow through the transreceiver. When it is high, data is transmitted out and vice-a-versa.

M/IO

This signal is used to distinguish between memory and I/O operations. When it is high, it indicates I/O operation and when it is low indicates the memory operation. It is available at pin 28.

WR

It stands for write signal and is available at pin 29. It is used to write the data into the memory or the output device depending on the status of M/IO signal.

HLDA

It stands for Hold Acknowledgement signal and is available at pin 30. This signal acknowledges the HOLD signal.

HOLD

This signal indicates to the processor that external devices are requesting to access the address/data buses. It is available at pin 31.

QS_1 and QS_0

These are queue status signals and are available at pin 24 and 25. These signals provide the status of instruction queue. Their conditions are shown in the following table

QS_0	QS_1	Status
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty the queue
1	1	Subsequent byte from the queue

S_0, S_1, S_2

These are the status signals that provide the status of operation, which is used by the Bus Controller 8288 to generate memory & I/O control signals. These are available at pin 26, 27, and 28. Following is the table showing their status −

S_2	S_1	S_0	Status
0	0	0	Interrupt acknowledgement
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

LOCK

When this signal is active, it indicates to the other processors not to ask the CPU to leave the system bus. It is activated using the LOCK prefix on any instruction and is available at pin 29.

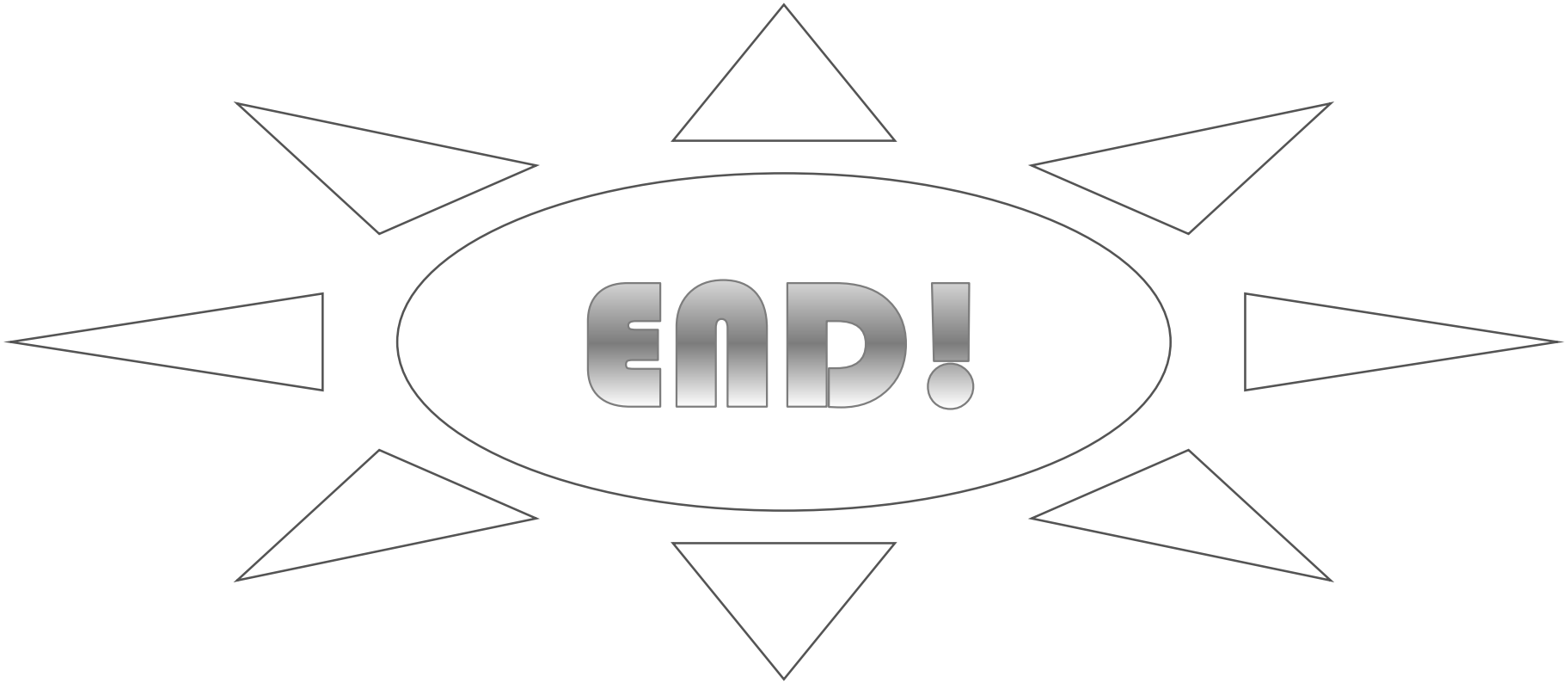
RQ/GT_1 and RQ/GT_0

These are the Request/Grant signals used by the other processors requesting the CPU to release the system bus. When the signal is received by CPU, then it sends acknowledgment. RQ/GT_0 has a higher priority than RQ/GT_1.

Microprocessor - 8086 Instruction Sets

The 8086 microprocessor supports 8 types of instructions −

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions


END!

CHAPTER-2A

INTRODUCTION TO MICROCONTROLLER

TECHNICAL AND VOCATIONAL TRAINING INSTITUTE
Department of Electrical electronics technology

Fundamentals of microprocessors/microcontroller

EETe 3032

By Zemenu T. /2022

Outlines

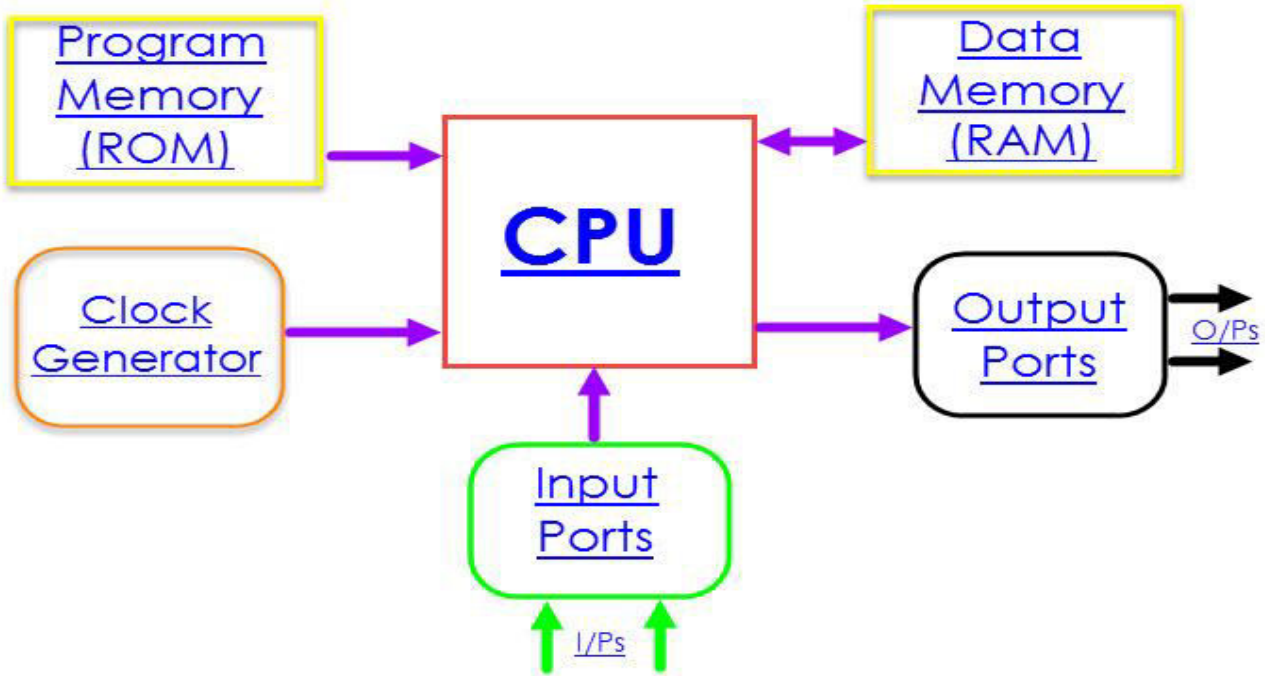
1. Introduction
2. A Brief History of Microprocessor and microcontroller
3. Microprocessors and Microcontroller comparison
4. Structure of a basic computer system,
5. The microcontroller architecture
6. CPU families used in microcontrollers,
7. Types of Memories
8. Memory organization, Registers and Clock concept

Introduction

Circumstances that we find ourselves in today in the field of microcontrollers had their beginnings in the development of technology of integrated circuits. This development has made it possible to store hundreds of thousands of transistors into one chip. That was a prerequisite for production of microprocessors, and the first computers were made by adding external peripherals such as memory, input-output lines, timers and other. Further increasing of the volume of the package resulted in creation of integrated circuits. These integrated circuits contained both **processor and peripherals**. That is how the first chip containing a **microcomputer**, or what would later be known as a **microcontroller** came about.

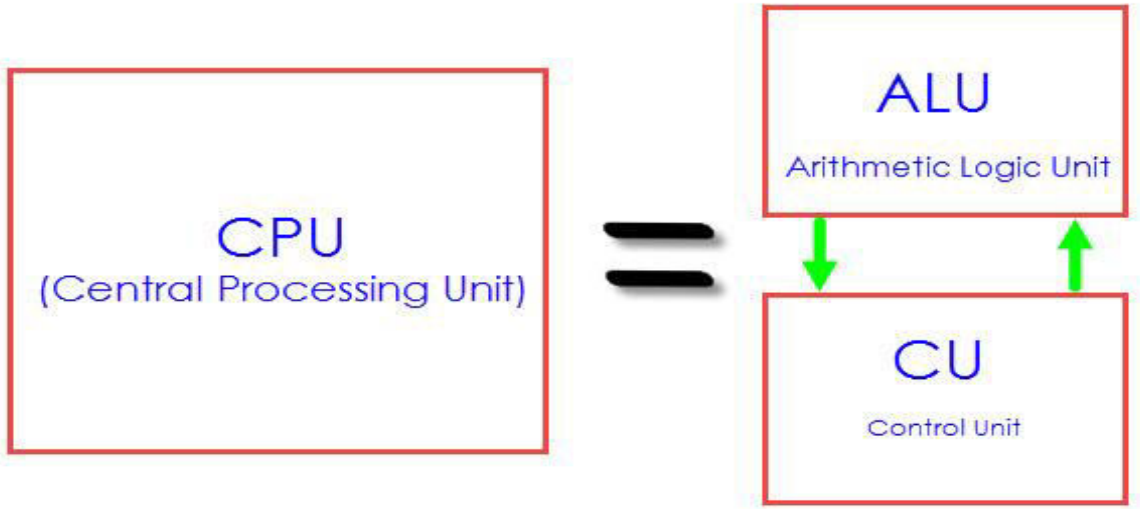
What is a Microcontroller?

A Microcontroller is a VLSI IC that contains a CPU (Processor) along with some other peripherals like Memory (RAM and ROM), I/O Ports, Timers/Counters, Communication Interface, ADC, etc. The following image shows the basic components of a Microcontroller. As all the components (and a few other components) are integrated on a single chip (Integrated Circuit – IC), a Microcontroller can be considered as a Microcomputer (or a Computer on – chip).



CPU (Central Processing Unit)

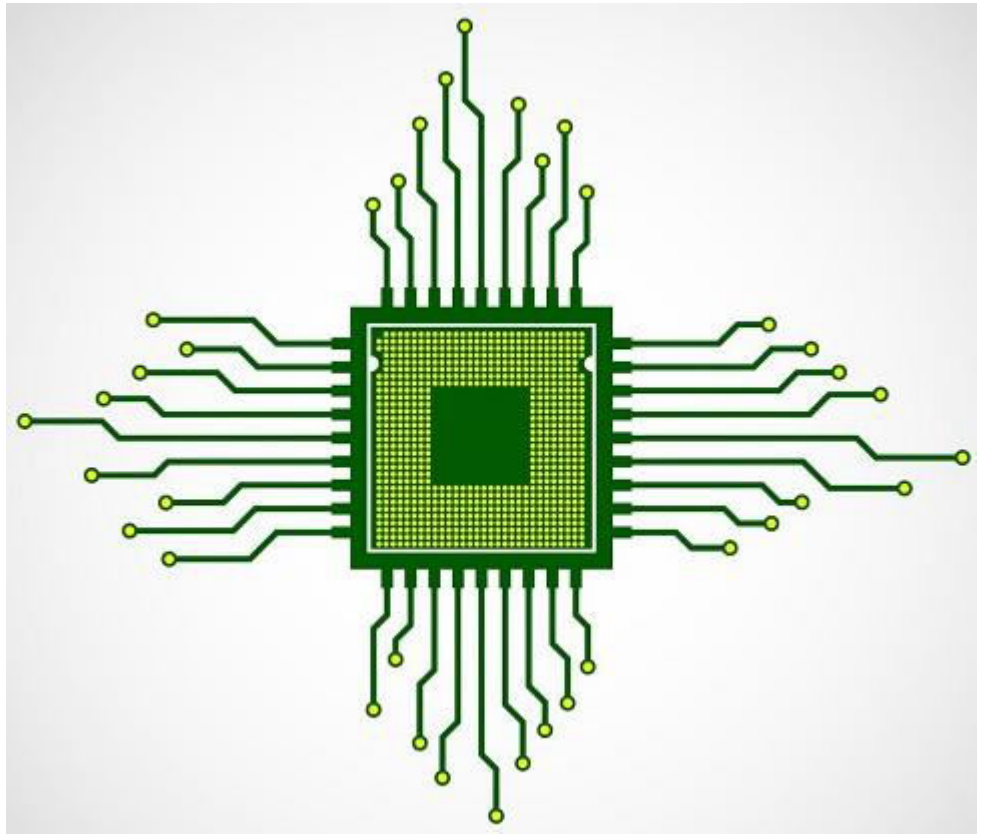
It is the heart of the Microcontroller that mainly comprises of an Arithmetic Logic Unit (ALU) and a Control Unit (CU) and other important components. The CPU is the primary device in communicating with peripheral devices like Memory, Input and Output.



ALU or Arithmetic Logic Unit, as the name suggests, performs the Arithmetical and Logical Operations. CU or Control Unit is responsible for timing of the communication process between the CPU and its peripherals

Microprocessor

Microprocessor is the brain of computer, which does all the work. It is a computer processor that incorporates all the functions of CPU (Central Processing Unit) on a single IC (Integrated Circuit) or at the most a few ICs. Microprocessors were first introduced in early 1970s. 4004 was the first general purpose microprocessor used by Intel in building personal computers. Arrival of low cost general purpose microprocessors has been instrumental in development of modern society the way it has.

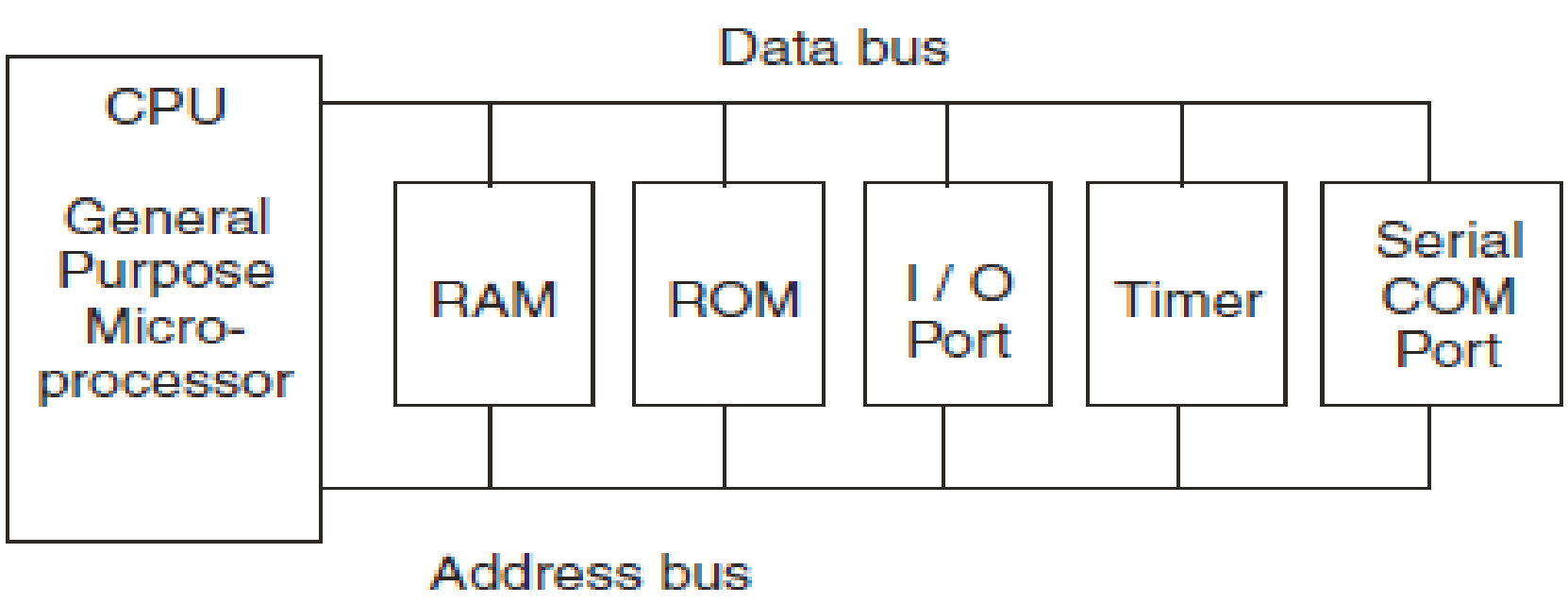


Microcontrollers Vs Microprocessors

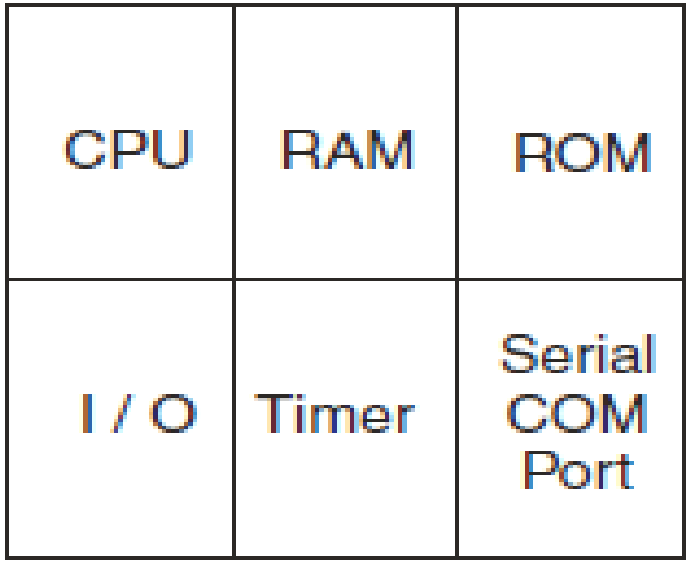
1. **A microprocessor** requires an external memory for program/data storage. Instruction execution requires movement of data from the external memory to the microprocessor or vice versa. Usually, microprocessors have good computing power and they have higher clock speed to facilitate faster computation.
2. **A microcontroller** has required on-chip memory with associated peripherals. A microcontroller can be thought of a microprocessor with inbuilt peripherals.

A microcontroller does not require much additional interfacing ICs for operation and it functions as a stand alone system. The operation of a microcontroller is multipurpose. Microcontrollers are also called **embedded** controllers. A microcontroller clock speed is limited only to a few tens of MHz Microcontrollers are numerous and many of them are application specific.

Microcontrollers Vs Microprocessors (Cont'd)



(a) General-purpose Microprocessor System



(b) Microcontroller

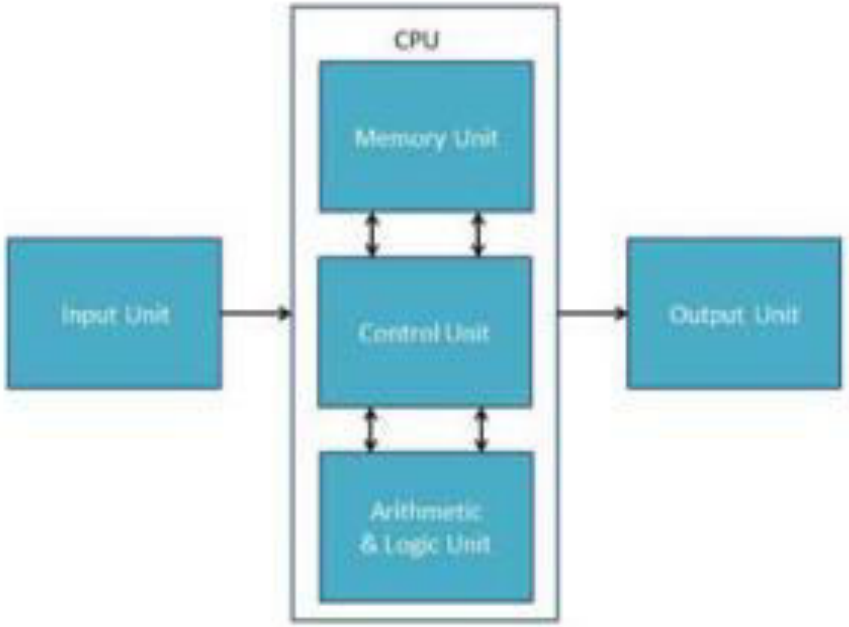
In order to make Microprocessor work or build a system around it, we need to interface the peripherals separately.

The following table highlights the differences between a microprocessor and a microcontroller −

Microprocessor	Microcontroller
Microprocessors are multitasking in nature. Can perform multiple tasks at a time. For example, on computer we can play music while writing text in text editor.	Single task oriented. For example, a washing machine is designed for washing clothes only.
RAM, ROM, I/O Ports, and Timers can be added externally and can vary in numbers.	RAM, ROM, I/O Ports, and Timers cannot be added externally. These components are to be embedded together on a chip and are fixed in numbers.
Designers can decide the number of memory or I/O ports needed.	Fixed number for memory or I/O makes a microcontroller ideal for a limited but specific task.
External support of external memory and I/O ports makes a microprocessor-based system heavier and costlier.	Microcontrollers are lightweight and cheaper than a microprocessor.
External devices require more space and their power consumption is higher.	A microcontroller-based system consumes less power and takes less space.

Basic Structure of Computer

A computer system is basically a machine that simplifies complicated tasks. It should maximize performance and reduce costs as well as power consumption. All types of computers follow a same basic logical structure and perform the following five basic operations for converting raw input data into information useful to their users.
Following diagram shows the basic structure of Computer:



Input Unit

This unit contains devices with the help of which we enter data into computer. This unit makes link between user and computer. The input devices translate the information into the form understandable by computer.

CPU (Central Processing Unit)

CPU is considered as the brain of the computer. CPU performs all types of data processing operations. It stores data, intermediate results and instructions(program). It controls the operation of all parts of computer.

CPU itself has following three components

- ALU(Arithmetic Logic Unit)
- Memory Unit
- Control Unit

Output Unit

Output unit consists of devices with the help of which we get the information from computer. This unit is a link between computer and users. Output devices translate the computer's output into the form understandable by users.

S.No.	Operation	Description
1	Take Input	The process of entering data and instructions into the computer system
2	Store Data	Saving data and instructions so that they are available for processing as and when required.
3	Processing Data	Performing arithmetic, and logical operations on data in order to convert them into useful information.
4	Output Information	The process of producing useful information or results for the user, such as a printed report or visual display.
5	Control the workflow	Directs the manner and sequence in which all of the above operations are performed.

Development/Classification of microcontrollers

Microcontrollers have gone through a silent evolution (invisible). The evolution can be rightly termed as silent as the impact or application of a microcontroller is not well known to a common user, although microcontroller technology has undergone significant change since early 1970's. Development of some popular microcontrollers is given as follows.

Development/Classification of microcontrollers (Contd.)

Intel 4004	4 bit (2300 PMOS trans, 108 kHz)	1971
Intel 8048	8 bit	1976
Intel 8031	8 bit (ROM-less)	.
Intel 8051	8 bit (Mask ROM)	1981
Microchip PIC16C64	8 bit	1985
Motorola 68HC11	8 bit (on chip ADC)	.
Intel 80C196	16 bit	1982
Atmel AT89C51	8 bit (Flash memory)	.
Microchip PIC 16F877	8 bit (Flash memory + ADC)	.

Basic functions of parts of microcontroller system

• **Input** – accepts coded information from human operators, from electromechanical devices (such as keyboards), or from other digital medium via digital communication lines.

•The information received is either stored in the ***memory*** or immediately used by the ***arithmetic and logic unit (ALU)*** to perform the desired operations.

•The results are sent back out through the ***output*** medium.

•All actions are coordinated through the ***control unit***.

Microcontroller basics

Information organization

- Categorized as either ***instructions*** or ***data***

-Instructions (or machine instructions) are explicit commands that

- Govern the transfer of information within a microcontroller as well as between the microcontroller and its I/O devices.
- Specify the arithmetic and logic operations to be performed.

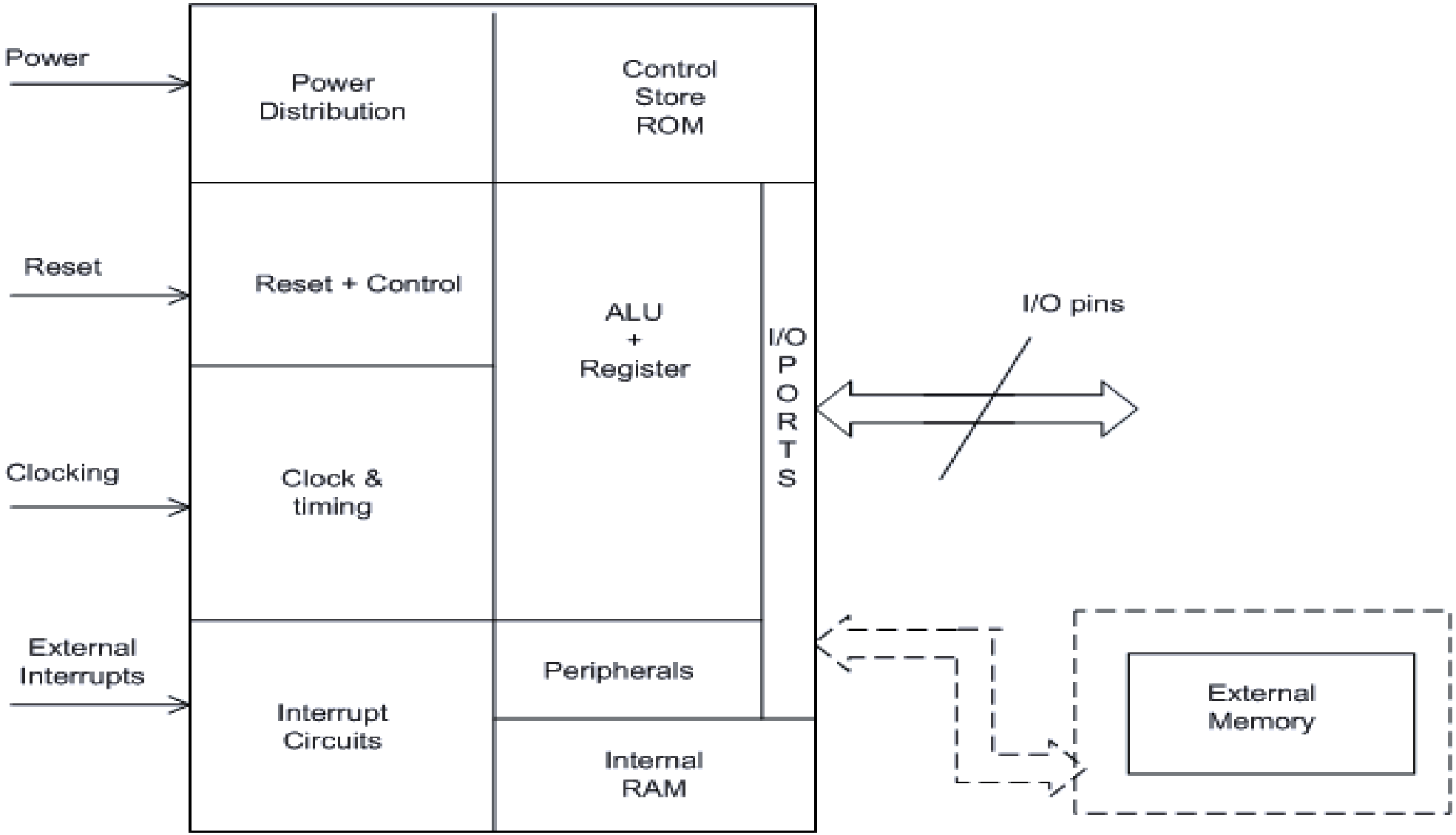
•Programs

- A list of instructions that performs a task is called a ***program***.
- Usually the program is stored in **memory**.
- The program fetches the instructions from memory, one after another, and performs the desired operations.
- The microcontroller is completely controlled by the stored program, except for possible interruption by an operator or by I/O devices connected to the machine.
- Data are numbers and encoded characters that are used as operands by the instructions.

Internal Structure of a Microcontroller

At times, a microcontroller can have external memory also (if there is no internal memory or external memory interface is required). Early microcontrollers were manufactured using bipolar or NMOS technologies. Most modern microcontrollers are manufactured with CMOS technology, which leads to reduction in size and power loss. Current drawn by the IC is also reduced considerably from 10mA to a few micro Amperes in sleep mode (for a microcontroller running typically at a clock speed of 20MHz).

Internal Structure of a Microcontroller (Cont'd)



Bus Structure

Bus - a group of parallel wires that transfer information from one part of the computer to another.

Control Bus

synchronizes the actions of all of the devices attached to the system bus.

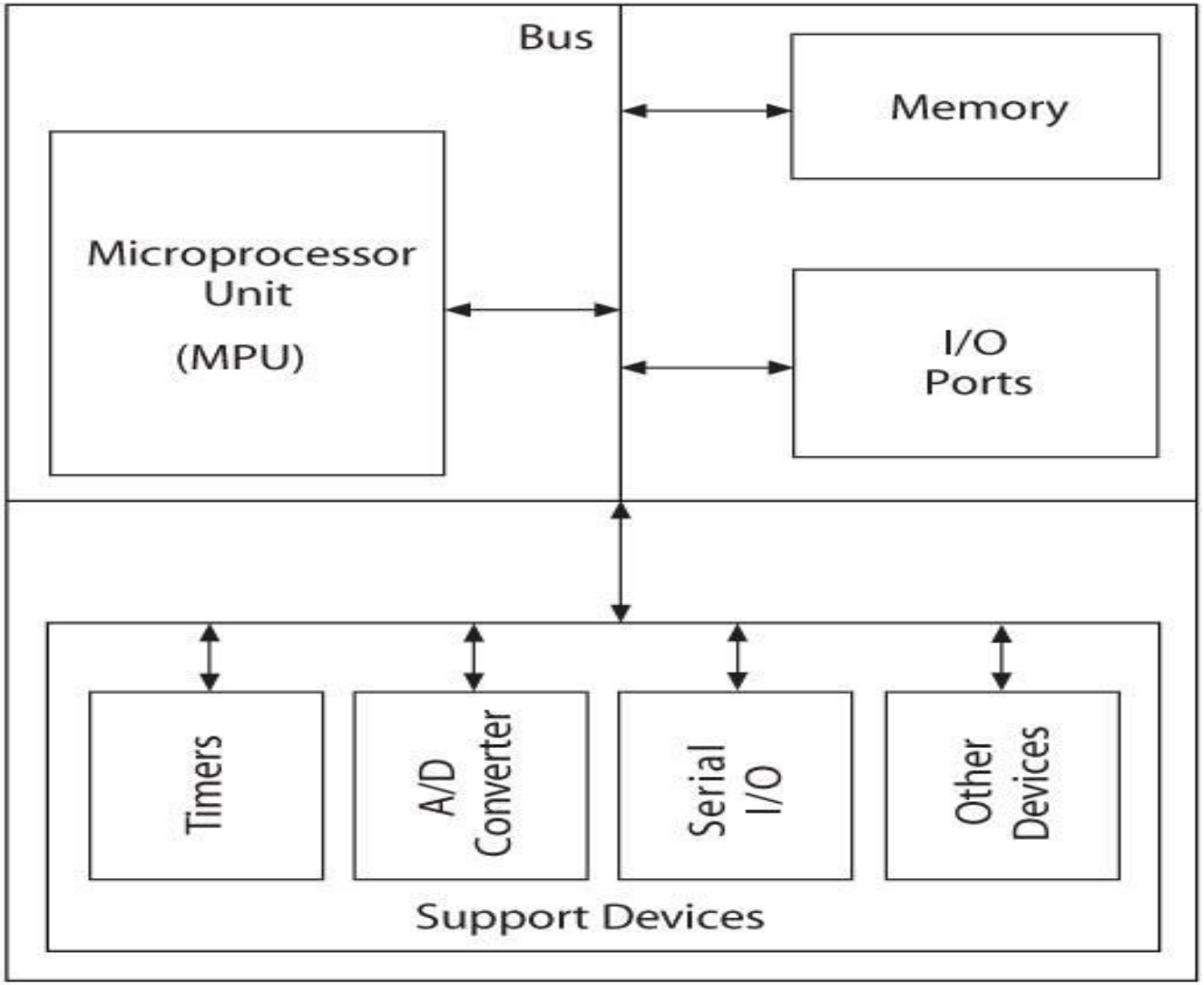
Address Bus

passes the addresses of instructions and data between the CPU and memory (or I/O).

Data Bus

transfers instructions and data between the CPU and memory (or I/O).

Block diagram of a microcontroller system



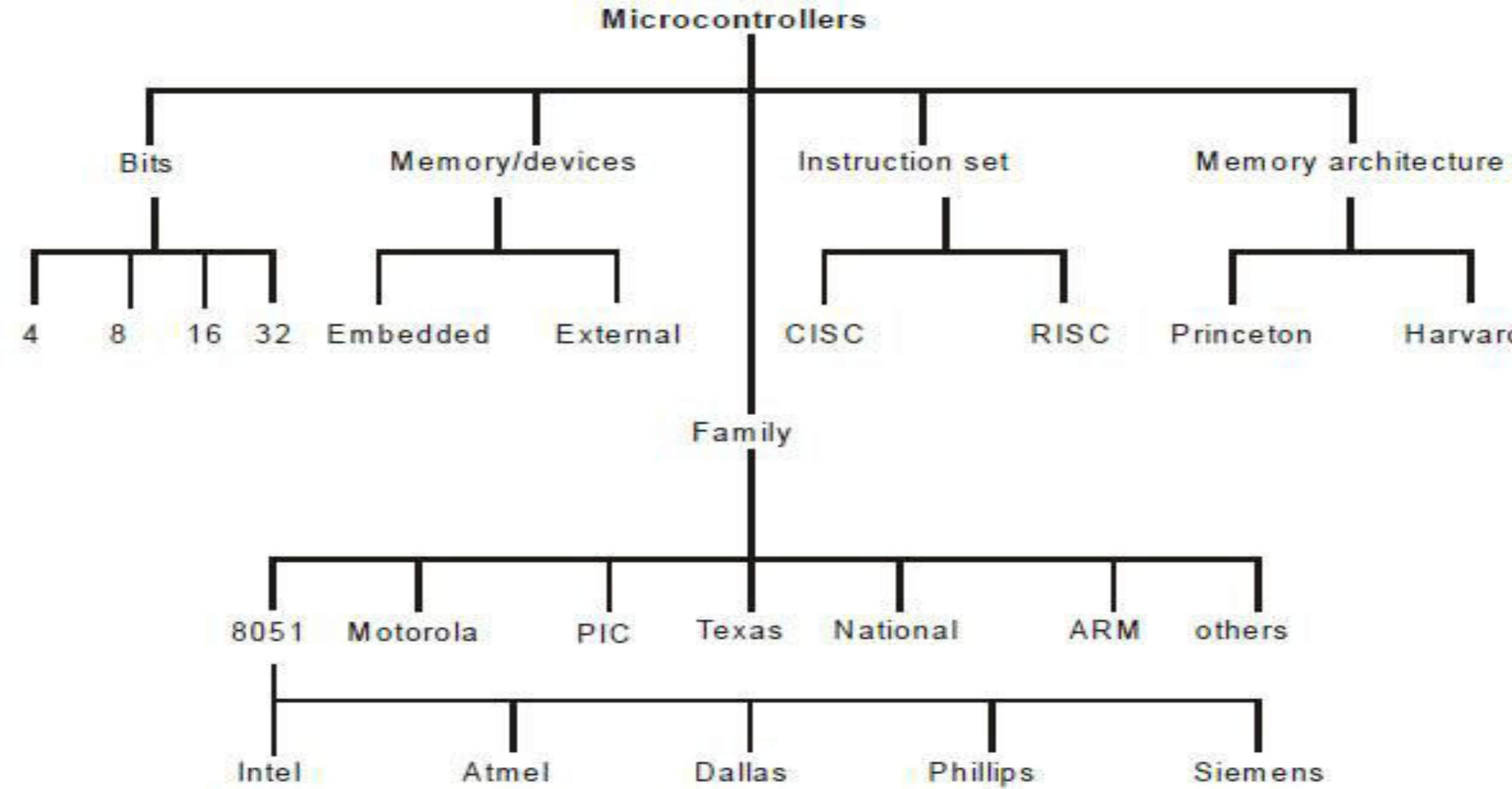


Figure 1.4 Types of microcontrollers

Types of Computer Architecture

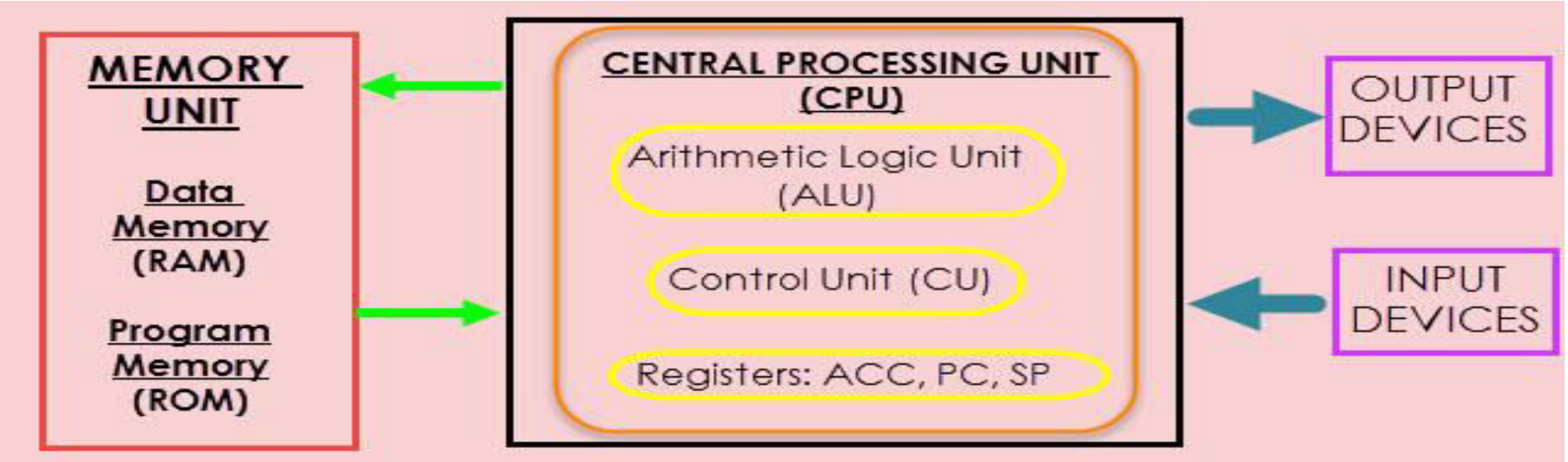
Basically, Microprocessors or Microcontrollers are classified based on the two types of Computer Architecture: Von Neumann Architecture and Harvard Architecture

Von Neumann Architecture

Von Neumann Architecture or Princeton Architecture is a Computer Architecture, where the Program i.e. the *Instructions* and the *Data* are stored in a **single memory**.

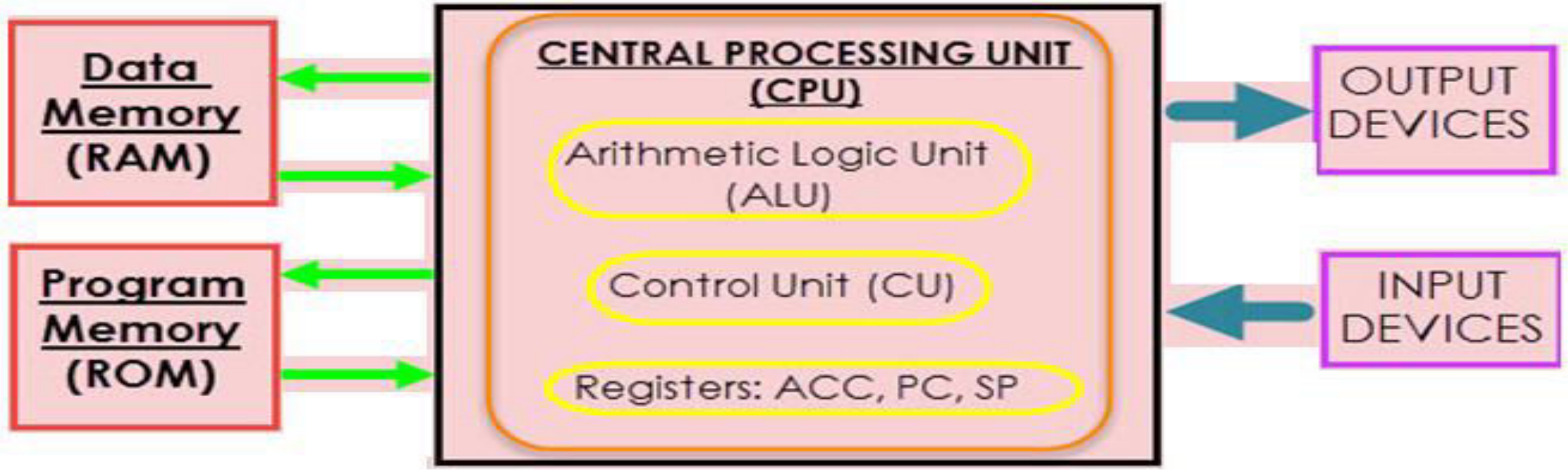
Since the Instruction Memory and the Data Memory are the same, the Processor or CPU cannot access both Instructions and Data at the same time as they use a single bus.

This type of architecture has severe limitations to the performance of the system as it creates a bottleneck while accessing the memory.



Harvard Architecture

Harvard Architecture, in contrast to Von Neumann Architecture, uses **separate memory** for Instruction (Program) and Data. Since the Instruction Memory and Data Memory are separate in a Harvard Architecture, their signal paths i.e. buses are also different and hence, the CPU can access both Instructions and Data at the same time. Almost all Microcontrollers, including 8051 Microcontroller implement **Harvard Architecture**.



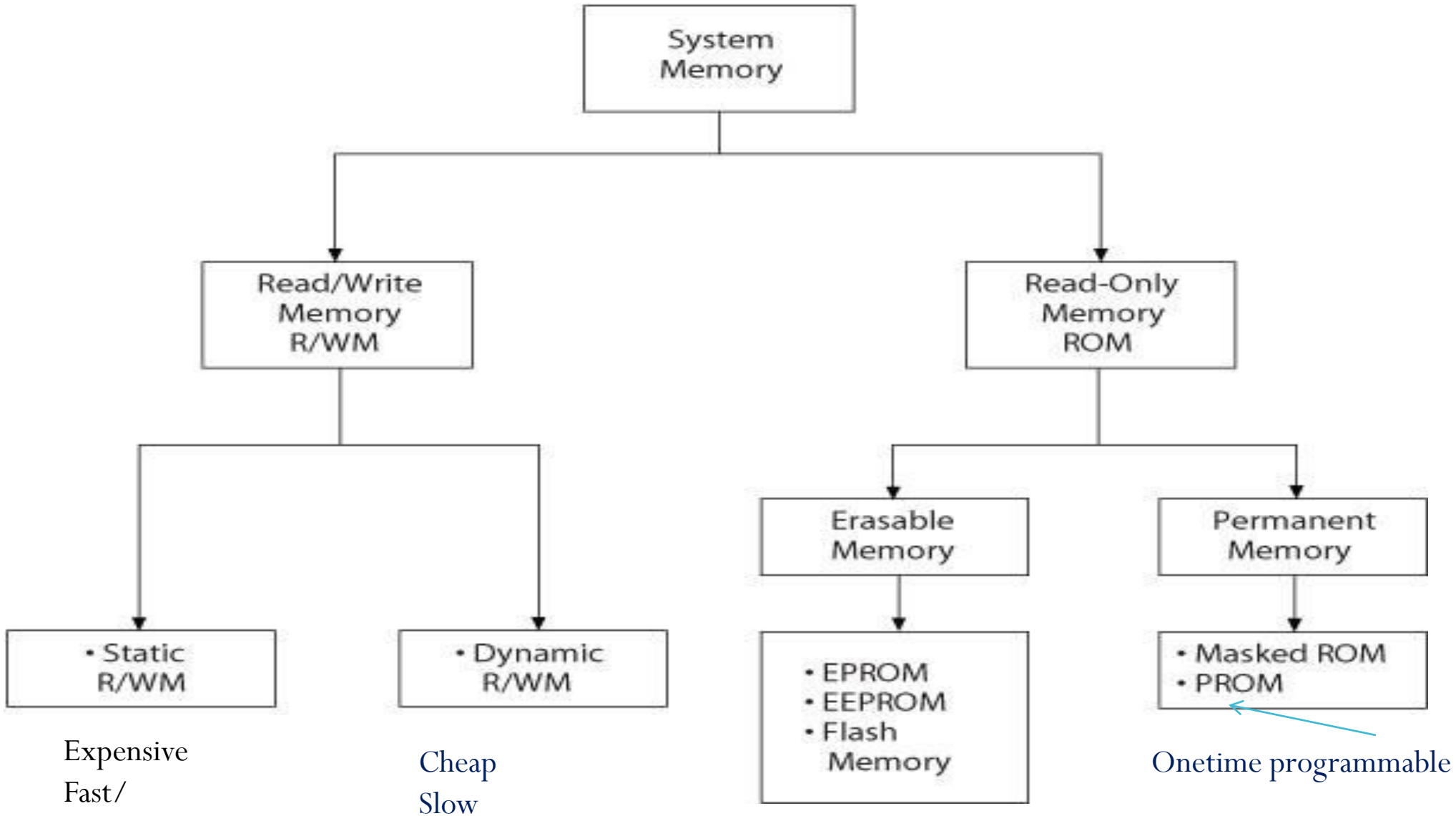
CISC and RISC

CISC is a Complex Instruction Set Computer. It is a computer that can address a large number of instructions. In the early 1980s, computer designers recommended that computers should use fewer instructions with simple constructs so that they can be executed much faster within the CPU without having to use memory. Such computers are classified as Reduced Instruction Set Computer or RISC.

CISC vs RISC

CISC	RISC
Larger set of instructions. Easy to program	Smaller set of Instructions. Difficult to program.
Simpler design of compiler, considering larger set of instructions.	Complex design of compiler.
Many addressing modes causing complex instruction formats.	Few addressing modes, fix instruction format.
Higher clock cycles per second.	Low clock cycle per second.

Memory Classification



Expensive
Fast/

Cheap
Slow

Electronically Erasable
PROM

Onetime programmable

Memory Classification (Cont'd)

In a microcontroller, two types of memory are found. They are, **program memory** and **data memory** respectively. Program memory is also known as 'control store' and 'firm ware'. It is **non-volatile** i.e, the memory content is not lost when the power goes off. Non-volatile memory is also called Read Only Memory (ROM). There are various types of ROM.

Program Memory Types

1. **Mask ROM:** Some microcontrollers with ROM are programmed while they are still in the factory. This ROM is called Mask ROM. Since the microcontrollers with Mask ROM are used for specific application, there is no need to reprogram them.

2. Reprogrammable program memory (or) Erasable PROM (EPROM): Microcontrollers with EPROM were introduced in late 1970's. These devices are electrically programmable but are erased with UV radiation. Usually, these versions of micro controllers are expensive.

Program Memory Types (Cont'd)

3. **OTP EPROM:** One time programmable (OTP) EPROM based microcontrollers do not have any glass window for UV erasing. These can be programmed only once. This type of packaging results in microcontroller that have the cost 10% of the microcontrollers with UV erase facility (i.e., 1/10th cost).

4. **EEPROM**: (Electrically Erasable Programmable ROM): This is similar to EPROM but the float charge can be removed electrically.

5. **FLASH (EEPROM Memory)**: FLASH memory was introduced by INTEL in late 1980's. This memory is similar to EEPROM but the cells in a FLASH memory are bussed so that they can be erased in a few clock cycles. Hence the reprogramming is faster.

Organization of Data Memory

DATA Memory

Data memory can be classified into the following categories

- Bits
- Registers
- Variable RAM
- Program counter stack

Microcontroller can have ability to perform manipulation of individual bits in certain registers(bit manipulation). This is a unique feature of a microcontroller, not available in a microprocessor.

Eight bits make **a byte**. Memory bytes are known as file registers.

Registers are some special RAM locations that can be accessed by the processor very easily.

Organization of Data Memory (Contd.)

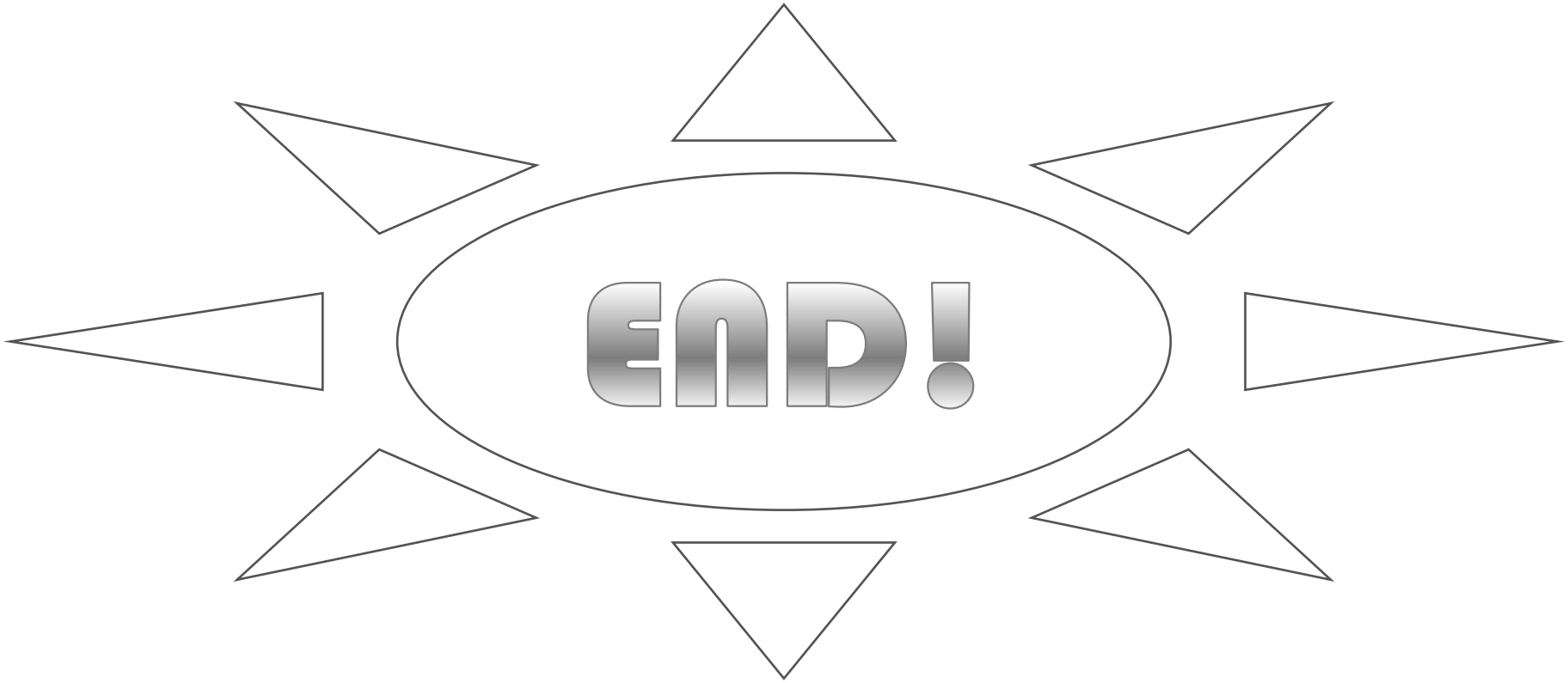
I/O Registers:

In addition to the Data memory, some special purpose registers are required that are used in input/output and control operations. These registers are called **I/O registers**. These are important for microcontroller peripheral interface and control applications.

Organization of Data Memory (Contd.)

Hardware interface registers (I/O Space)

As we already know a microcontroller has some embedded peripherals and I/O devices. The data transfer to these devices takes place through I/O registers. In a microprocessor, input /output (I/O) devices are externally interfaced and are mapped either to memory address (memory mapped I/O) or a separate I/O address space (I/O mapped I/O).


END!

CHAPTER-2B

INTRODUCTION TO MICROCONTROLLER

TECHNICAL AND VOCATIONAL TRAINING INSTITUTE
Department of Electrical electronics technology

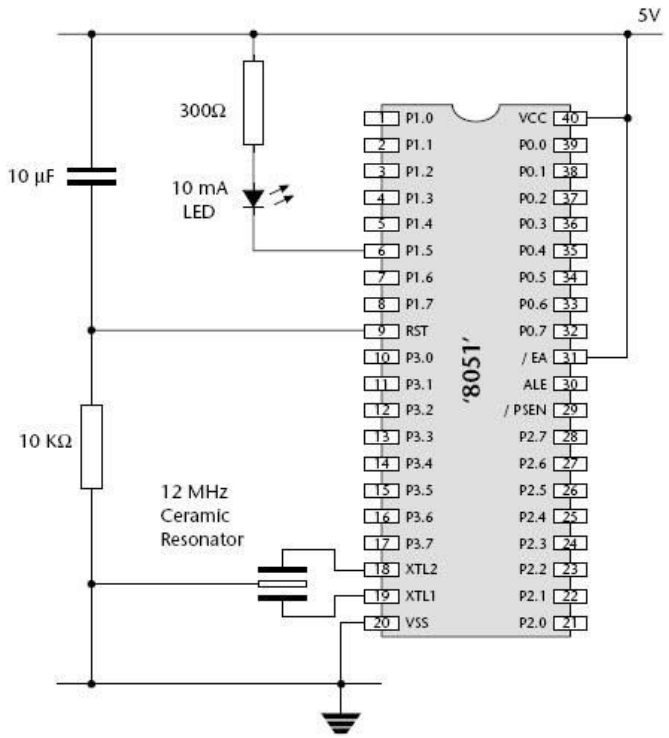
Fundamentals of microprocessors/microcontroller

EETe 3032

By Zemenu T. /2022

Introduction to the 8051 Microcontroller

The 8051 microcontroller is a basic microcontroller, it is first introduced by the 'Intel Corporation' since 1970. It is developed by the 8086 processor architecture. The 8051 is a family of the microcontroller, which has been developed by different manufactures such as Philips, atmel, dalls, and so on. The 8051 microcontroller has been used in lots of embedded products from small children's toys to large automotive systems.



A brief history of the 8051

Originally, 8051 Microcontrollers were developed using N-MOS Technology but the use of battery powered devices and their low power consumption lead to usage of CMOS Technology (which is famous for its low power consumption).

Even though Intel developed 8051 Microcontrollers, more than 20 semiconductor manufacturers are still producing 8051 compatible microcontrollers i.e. processors based on MSC-51 Architecture.

Some of the 8051 Microcontrollers produced by different manufacturers are: Atmel (AT89C51, AT89S51), Phillips (S87C654), STC Micro (STC89C52), Infineon (SAB- C515, XC800), Siemens (SAB-C501), Silicon Labs (C8051), NXP (NXP700, NXP900), etc.

Majority of the modern 8051 Microcontrollers are **Silicon IP Cores** (Intellectual Property Cores) but discrete 8051 Microcontroller IC's are also available. Because of their low power consumption, smaller size and simple architecture, 8051 IP Cores are used in FPGAs (Field Programmable Gate Array) and SoCs (System on Chip) instead of Advanced ARM Architecture based MCUs.

A brief history of the 8051 (Cont'd)

In 1981, Intel Corporation introduced an 8-bit microcontroller called the 8051. This microcontroller had 128 bytes of RAM, 4K bytes of on-chip ROM, two timers, one serial port, and four ports (each 8-bits wide) all on a single chip. At the time it was also referred to as a "**system on a chip**." **The 8051 is an 8-bit processor**, meaning that the CPU can work on only 8 bits of data at a time. Data larger than 8 bits has to be broken into 8-bit pieces to be processed by the CPU. The 8051 has a total of four I/O ports, each 8 bits wide. The 8051 can have a maximum of 64K bytes

Some of the microcontrollers of 8051 family are given as follows:

DEVICE	ON-CHIP DATA MEMORY (bytes)	ON-CHIP PROGRAM MEMORY (bytes)	16-BIT TIMER/COUNTER	NO. OF VECTORED INTERUPTS	FULL DUPLEX I/O
8031	128	None	2	5	1
8032	256	none	2	6	1
8051	128	4k ROM	2	5	1
8052	256	8k ROM	3	6	1
8751	128	4k EPROM	2	5	1
8752	256	8k EPROM	3	6	1
AT89C51	128	4k Flash Memory	2	5	1
AT89C52	256	8k Flash memory	3	6	1

8051 Microcontroller family

The microcontroller usually referred as 8051 has many variants from different manufacturers

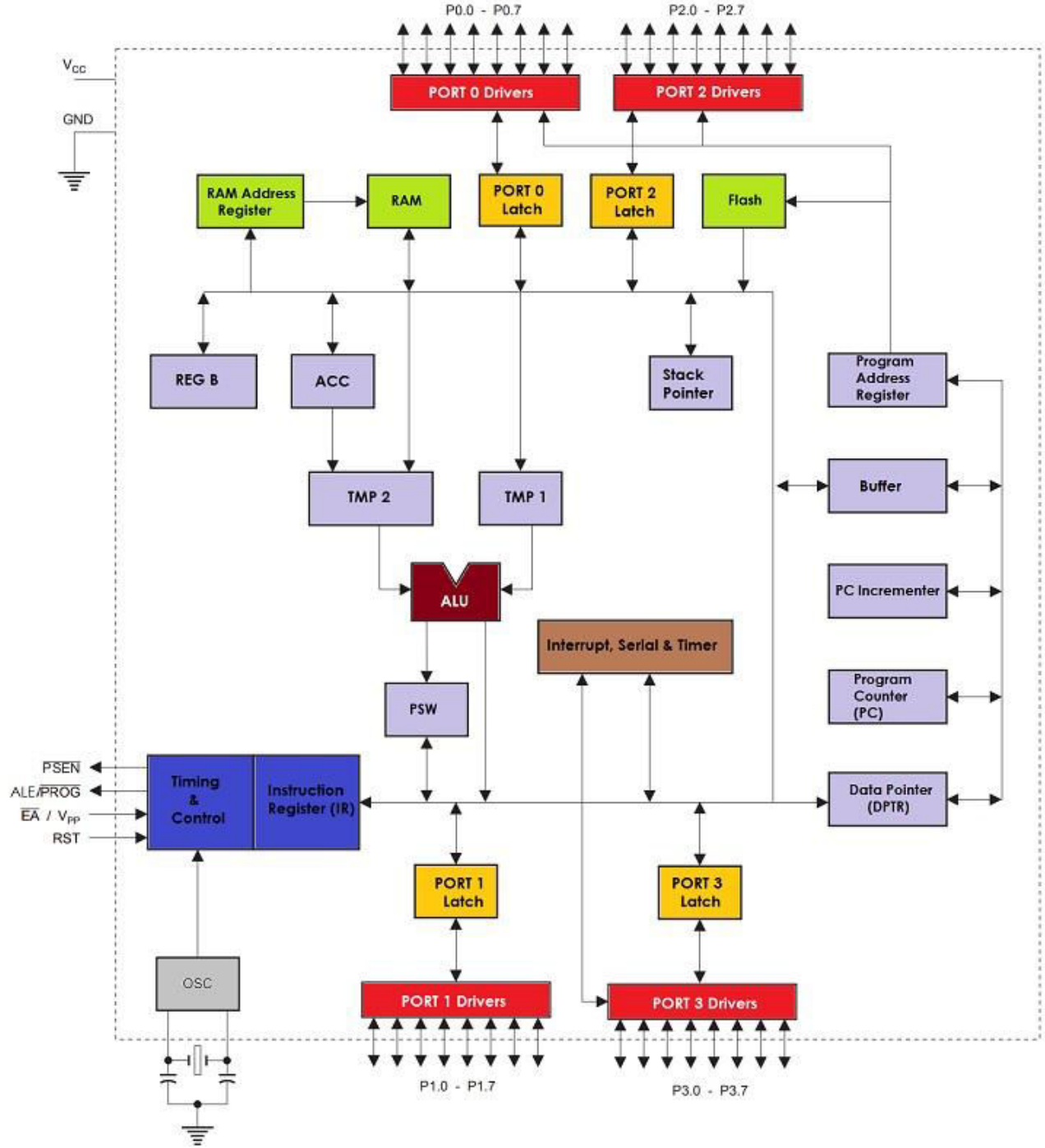
The generic part numbering scheme is as follows:

- 8xxx: NMOS logic
- 8xCxx: CMOS logic
- 803x: No internal program memory
- 805x: Factory programmed internal ROM program memory
- 87xx: Internal user programmable EPROM program memory
- 89xx: Internal flash EPROM program memory
- 8xx1: 4 kilobyte internal program memory, 128 byte internal RAM
- 8xx2: 8 kilobyte internal program memory, 256 byte internal RAM

8051 Architecture

8051 is an 8 – bit Microcontroller i.e. the data bus of the 8051 Microcontroller (both internal and external) is 8 – bit wide. It is a CISC based Microcontroller with Harvard Architecture (separate program and data memory).

Since the basic layout of a microcontroller includes a CPU, ROM, RAM, etc. the 8051 microcontroller also has a similar layout. Irrespective of the manufacturer, the internal hardware design i.e. the 8051 Microcontroller Architecture remains more or less the same. The following image shows a brief layout of a typical 8051 Microcontroller.


P0.0 - P0.7
P2.0 - P2.7
V_{CC}
GND
PORT 0 Drivers
PORT 2 Drivers
RAM Address Register
RAM
PORT 0 Latch
PORT 2 Latch
Flash
REG B
ACC
Stack Pointer
Program Address Register
TMP 2
TMP 1
Buffer
ALU
PC Incrementer
Interrupt, Serial & Timer
Program Counter (PC)
PSW
PSEN
ALE/PROG
EA / V_{PP}
RST
Timing & Control
Instruction Register (IR)
Data Pointer (DPTR)
PORT 1 Latch
PORT 3 Latch
OSC
PORT 1 Drivers
PORT 3 Drivers
P1.0 - P1.7
P3.0 - P3.7

8051 Architecture (Contd.)

The basic architecture of 8051 is shown slide no.8 consists of:

•**On-chip RAM**: Random access memory of 128 byte is used for data storage in 8051. RAM as a non-volatile memory consists of register banks, stacks for temporary data storage and some special function registers.

•**On-chip ROM**: 8051 consists of 4KB ROM for program storage. ROM as a volatile memory helps in permanent data storage.

•**Timers and Counters**: Timer helps in providing delay between the events. In 8051, there are two timer pins T0, T1. If these pins are used in the counter mode, we can count the external pulses. In T0, it is possible to store 16 bit data. This is done by storing the lower 8 bit in TL0 and the upper 8 bit in TH0. Similarly, we can store 16 bit data in T1 also. TMOD and TCON helps in the timer operation.

8051 Architecture (Contd.)

•**Serial Port**: In order, to perform the serial communication, TXD and RXD pins are used. TXD pin is used for transmitting the serial data and the RXD pin is used for the transmission of the data. SCON register is used to control the operation of the serial communication.

•**Input and Output Ports**: P0, P1. P2. P3 form the four ports of 8051 microcontroller. Each of the port is 8 bit wide. Port **P0** is used as a Lower Order Address bus. Port **P2** can be used as I/O port and higher order bus A8 to A15. Port **P3** can be used as I/O pin and each pin of port 3 has special functions.

•**Oscillator** – This is used to provide clock to the 8051 microcontroller. The crystal frequency can vary from 4MHz to 30MHz.

•**Interrupts** - Interrupts are requests which are used to handle special events or routines known as Interrupt Service Routines. INT0 and INT1 are the basic interrupt pins used in 8051.

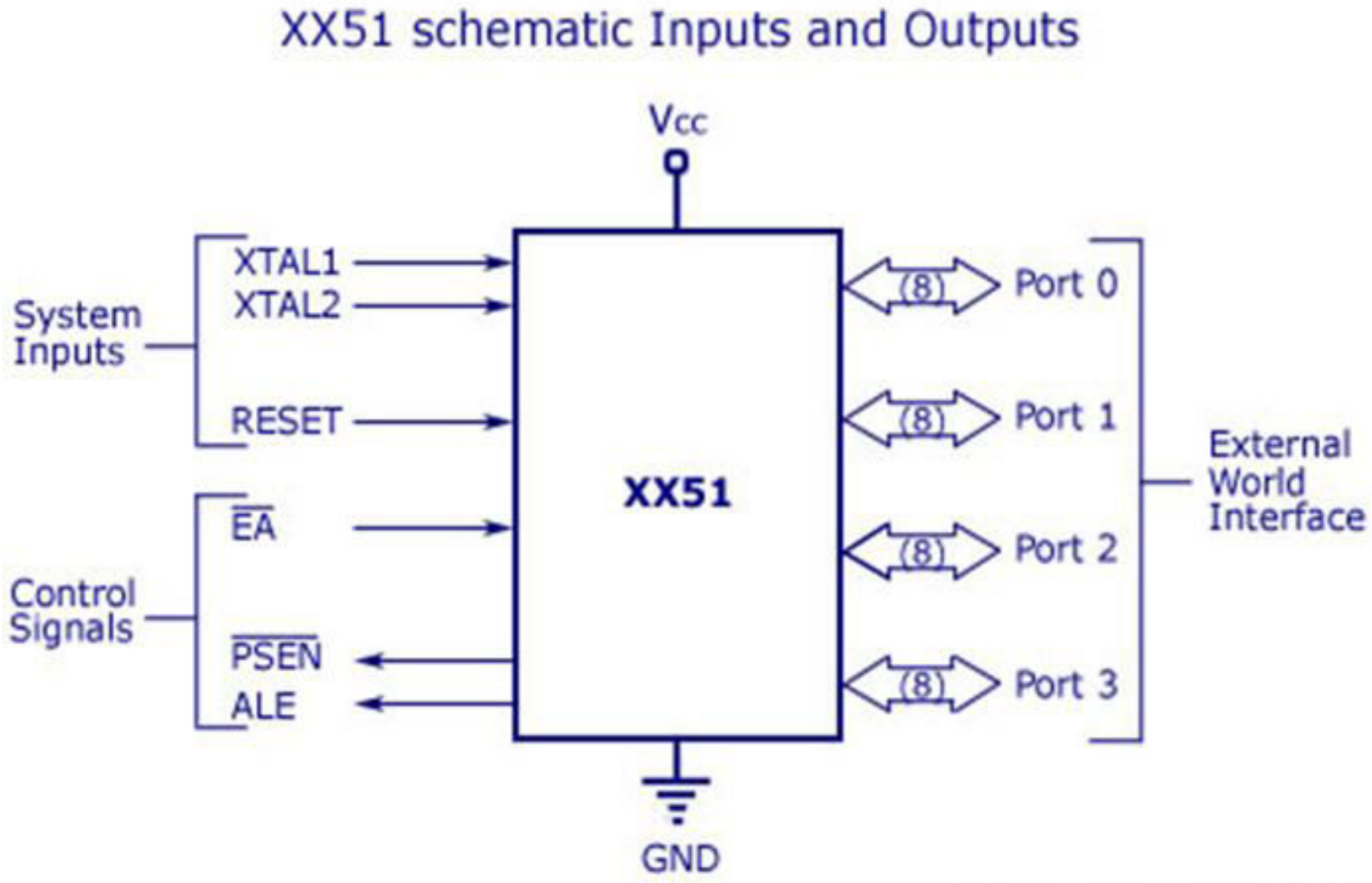
8051 Architecture (Contd.)

- **Arithmetic Logic Unit** - This unit is used for arithmetic calculations.
- **Accumulator** (A register) – This register is used for arithmetic operations.
- **B register** – This is an 8bit register that is bit addressable and is used for two instructions only like MUL AB and DIV AB.
- **Program Counter** – is a 16 bit register that helps to access address from 0000H to FFFFH. Program Counter is used to address the next instruction to be executed from the ROM.
- **Flag Bits and PSW register** – The flag bits are used to indicate the arithmetic condition of the ACC. Program Status Word (PSW) is the flag resister in 8051. This register consists of four flags like Carry, Auxiliary Carry Flag, Register Select 1, Register Select 0, Parity Flag, Overflow flag.

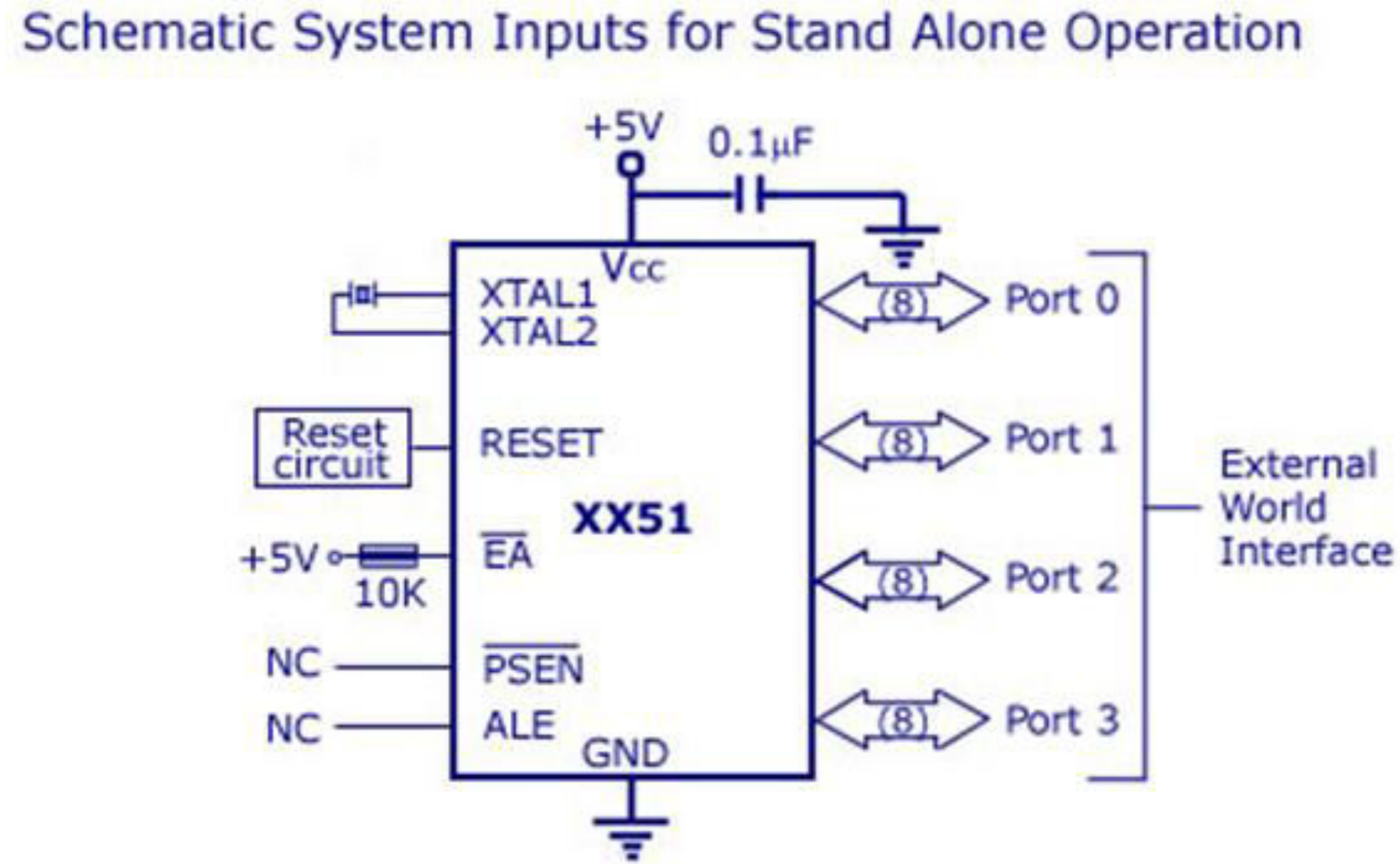
Basics of 8051 Microcontroller

- The most famous type of microcontroller is 8051
- Architecture
  - **3 system inputs**
    - Vcc power supply and ground
    - Clock inputs (XTAL 1 and XTAL 2)
    - Reset
  - **3 control signals**
    - EA (External Access)
    - PSEN (program store enable)
    - ALE (Address latch enable)
  - **4 ports**
    - Port 0, port 1, port 2 and port 3
    - Each 8 bit is programmable for I/O
    - Used to connect ADC, DAC, LED, 7 segment display etc

Basics of 8051 Microcontroller (cont'd)



Connection for standalone operation



8051 Microcontroller Basics

The microprocessor performs primarily four operations:

I. **Memory Read**: Reads data (or instruction) from memory.

II. **Memory Write**: Writes data (or instruction) into memory.

III. **I/O Read**: Accepts data from input device.

IV. **I/O Write**: Sends data to output device.

The 8051 processor performs these functions using address bus, data bus and control bus as shown in Fig. below.

8051 Bus structure

The block diagram of the 8051 Microcontroller Architecture shows that 8051 Microcontroller consists of a CPU, RAM (SFRs and Data Memory), Flash (EEPROM), I/O Ports and control logic for communication between the peripherals.

All these different peripherals inside the 8051 Microcontroller will communicate with each other via the 8 – bit Data Bus, also known as the internal data bus.

8051 Bus structure (Contd.)

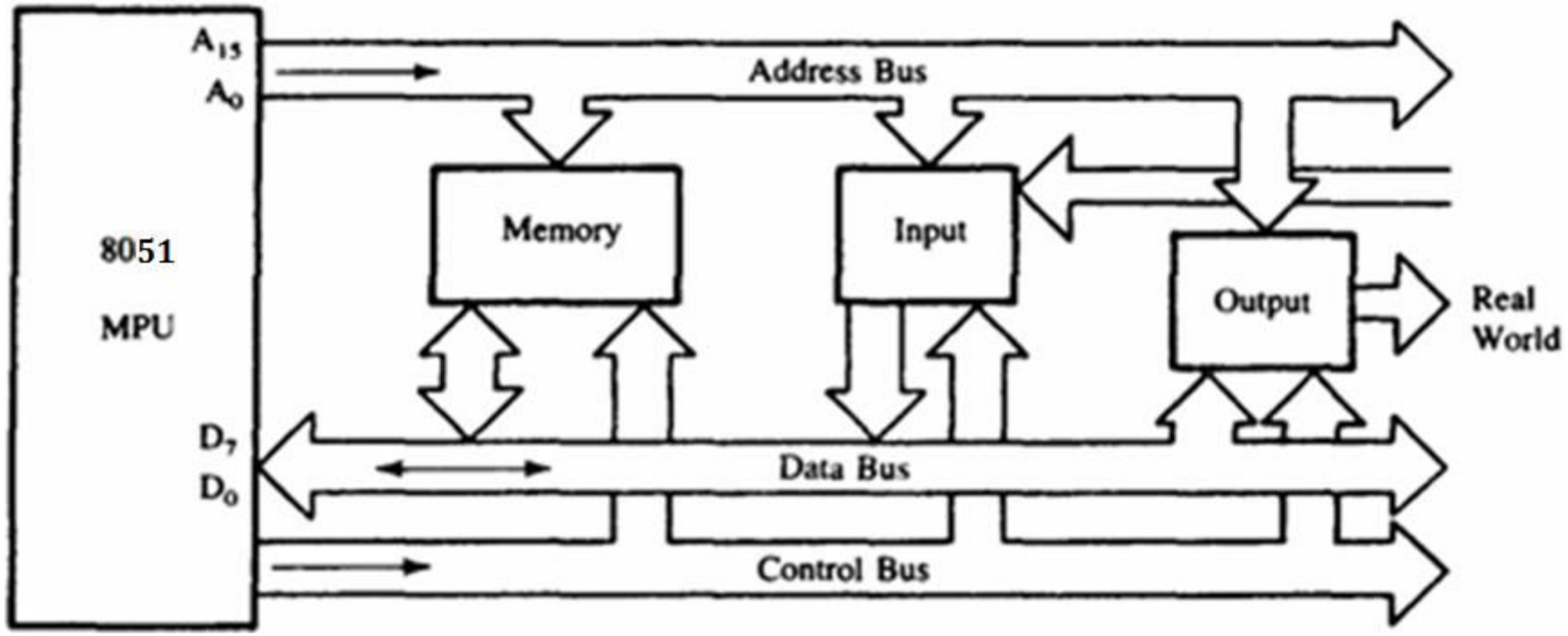
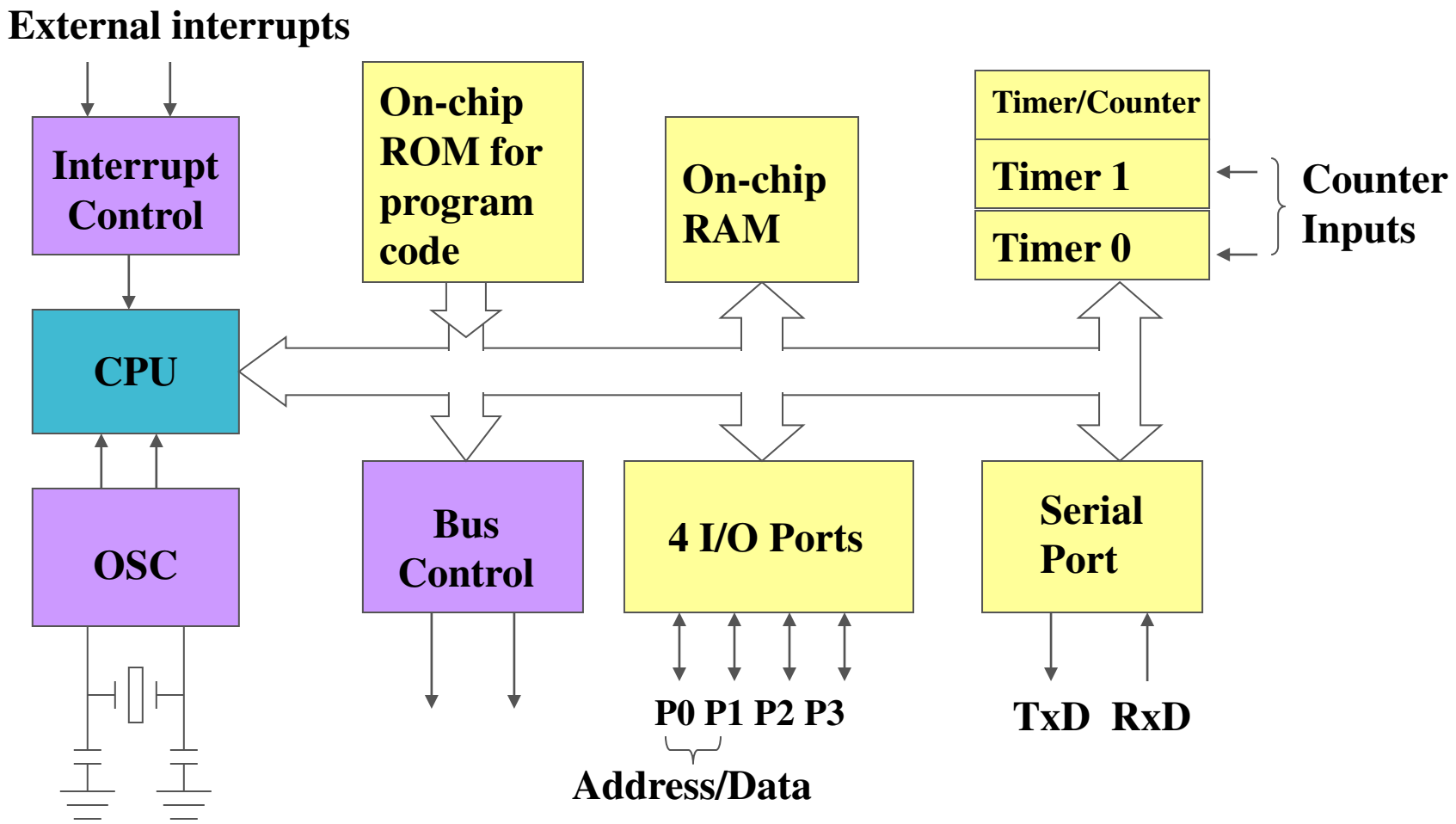


Fig. The 8051 bus structure

8051 employs **Harvard architecture**. It has some peripherals such as 32 bit digital I/O, Timers and Serial I/O.

Inside the 8051 Microcontroller Block Diagram



The 8051 is the original member of the 8051 family. Intel refers to it as MCS-51.

8051 PIN DESCRIPTION

- **Pin configuration**
  - 8051 MCU has 40 pins
  - Is the functional assignment of pins of a microcontroller
  - Some pins are dedicated pins, i.e. used for one purpose and others are multiplexed (used for multiple functions)
  - It can be used as a 4 i/o port standalone processor without external memory or can be used with 16 bit address line for external memory and 8 bit data line for external data access

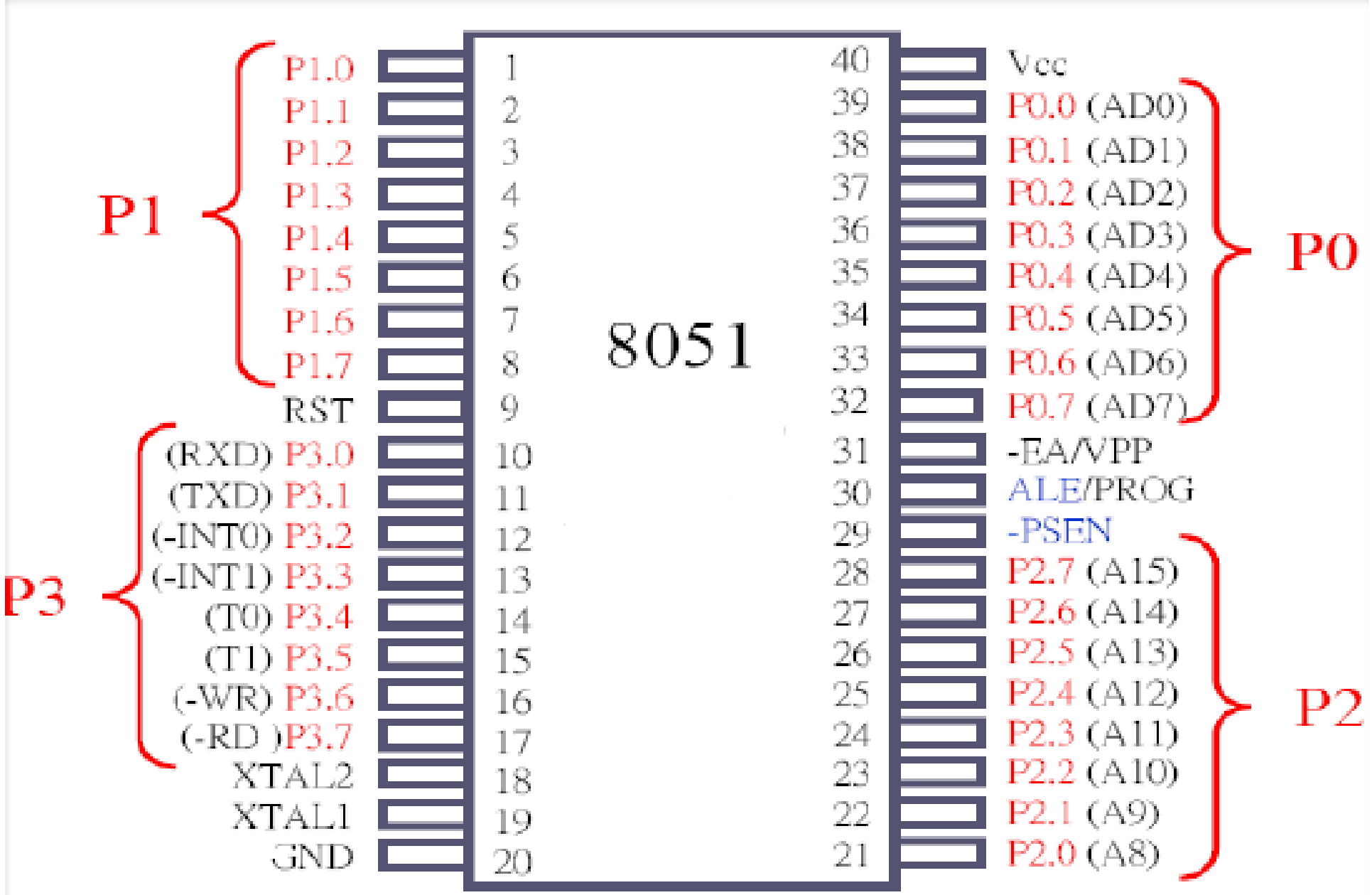
8051 PIN DESCRIPTION (Contd.)

The logic pin layout and signal groups of the 8051 microcontroller is shown in Fig. below. All the **signals** are classified into six groups:

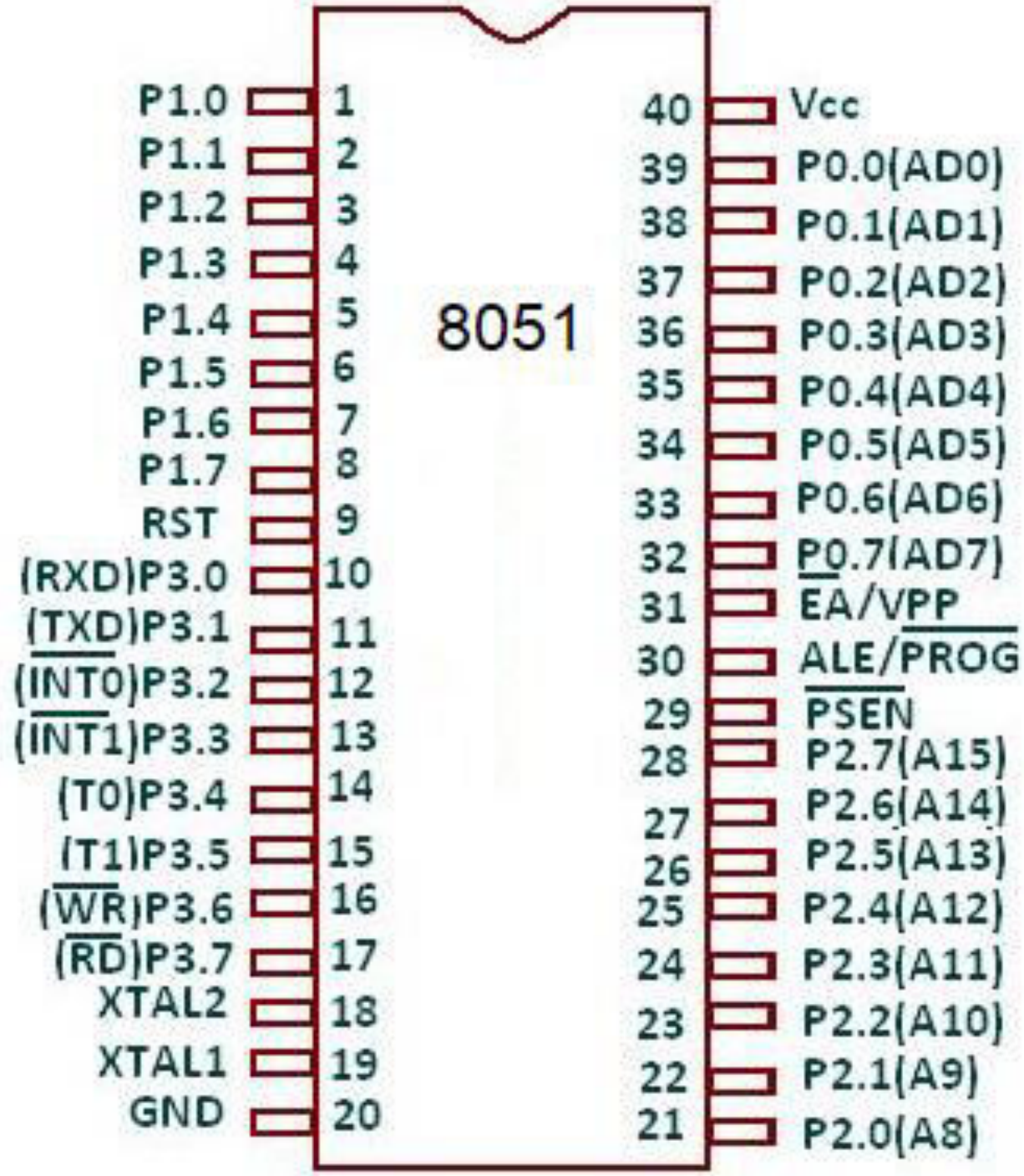
- Address bus
- Data bus
- Control & status signals
- Power supply and frequency signals
- Externally initiated signals
- Serial I/O signals

The following image shows the 8051 Microcontroller Pin Diagram with respect to a 40 – pin Dual In-line Package (DIP).

Pin Diagram-8051



40 – pin DIP


8051
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
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)

8051 PIN DESCRIPTION

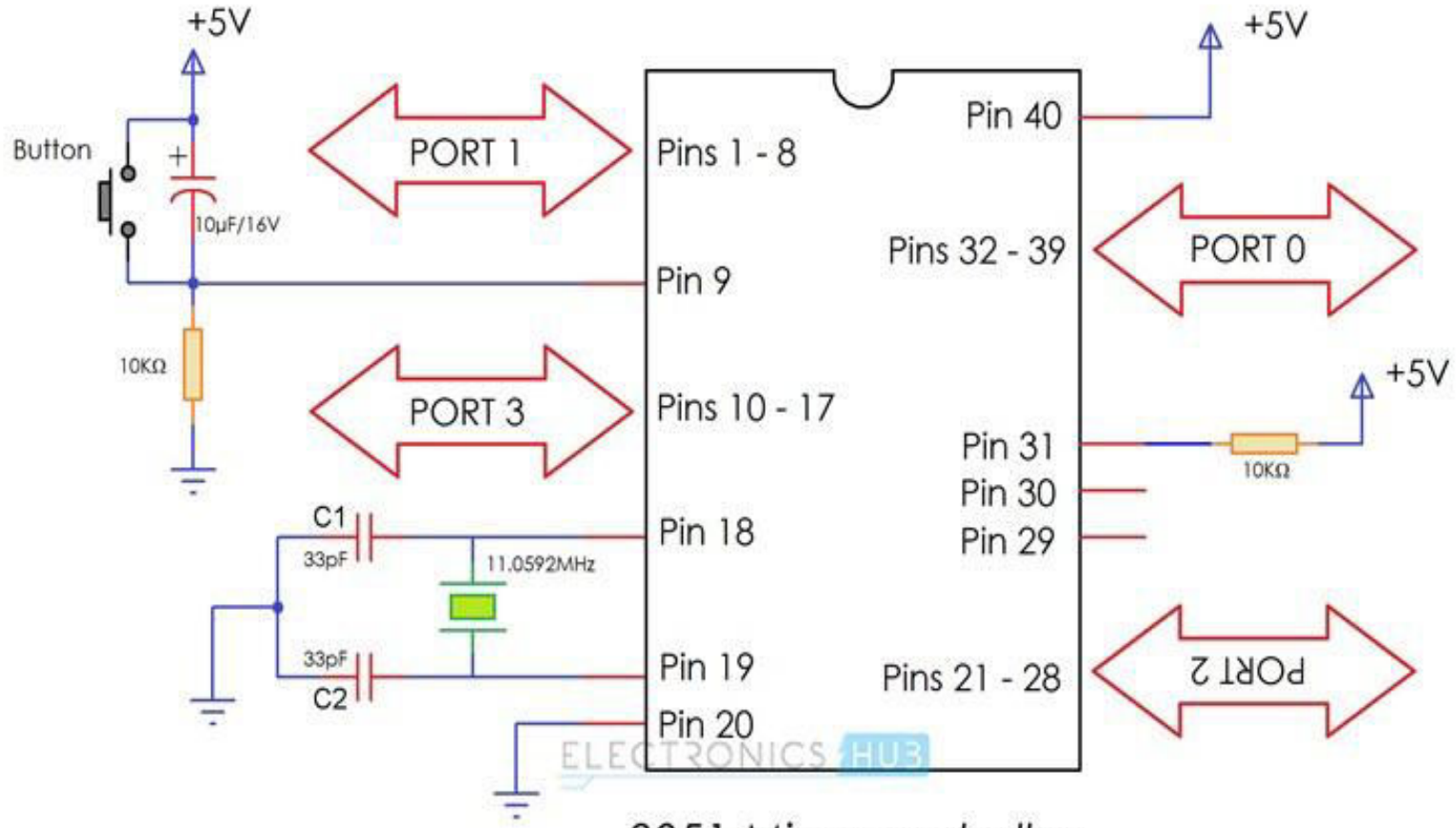
Pins 1-8	**PORT 1**. Each of these pins can be configured as an input or an output.
Pin 9	**RESET**. A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.
Pins10-17	**PORT 3**. Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions
Pin 10	**RXD.** Serial asynchronous communication input or Serial synchronous communication output.
Pin 11	**TXD.** Serial asynchronous communication output or Serial synchronous communication clock output.
Pin 12	**INT0.**External Interrupt 0 input
Pin 13	**INT1.** External Interrupt 1 input
Pin 14	**T0.** Counter 0 clock input
Pin 15	**T1.** Counter 1 clock input
Pin 16	**WR**. Write to external (additional) RAM
Pin 17	**RD.** Read from external RAM
Pin 18, 19	**XTAL2, XTAL1.** Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins.
Pin 20	**GND**. Ground.

8051 PIN DESCRIPTION (Contd.)

Pin 21-28	**Port 2**. If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.
Pin 29	**PSEN.** If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.
Pin 30	**ALE.** Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external latch latches the state of P0 and uses it as a memory chip address. Immediately after that, the ALE pin is returned its previous logic state and P0 is now used as a Data Bus.
Pin 31	**EA**. By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).
Pin 32-39	**PORT 0**. Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).
Pin 40	**VCC**. +5V power supply.

8051 Microcontroller Basic Circuit

We have seen the 8051 Microcontroller Pin Diagram and corresponding Pin Description, we will proceed to the basic circuit or schematic of the 8051 Microcontroller. The following image shows the basic circuit of the 8051 Microcontroller.



8051 Microcontroller

8051 Microcontroller Basic Circuit (Cont'd)

This basic circuit of 8051 microcontroller is the minimal interface required for it to work. The basic circuit includes a Reset Circuit, the oscillator circuit and power supply.

First is the power supply. Pins 40 and 20 (VCC and GND) of the 8051 Microcontroller are connected to +5V and GND respectively.

Next is the Reset Circuit. A logic HIGH (+5V) on Reset Pin for a minimum of two machine cycles (24 clock cycles) will reset the 8051 Microcontroller. The **reset circuit** of the 8051 Microcontroller consists of a capacitor, a resistor and a push button and this type of reset circuit provides a Manual Reset Option. If you remove the push button, then the reset circuit becomes a Power-On Reset Circuit.

The next part of the basic circuit of the 8051 Microcontroller is the Oscillator Circuit or the Clock Circuit. A Quartz Crystal Oscillator is connected across XTAL1 and XTAL2 pins i.e. Pins 19 and 18. The capacitors C1 and C2 can be selected in the range of 20pF to 40pF.

The basic setup of microcontroller

VCC and GND: - The pin 40 is Vcc i.e. it is given 5V and pin 20 is GND i.e. it is given 0V (supplied from power source) for powering up the microcontroller.

CONNECTING THE CRYSTAL (XTAL1- XTAL2): The pin numbers 18 - 19 are used to connect the crystal. The frequency of this crystal determines the machine cycle. With 8051 family we can use a crystal having frequency from 3 to 24 MHz.

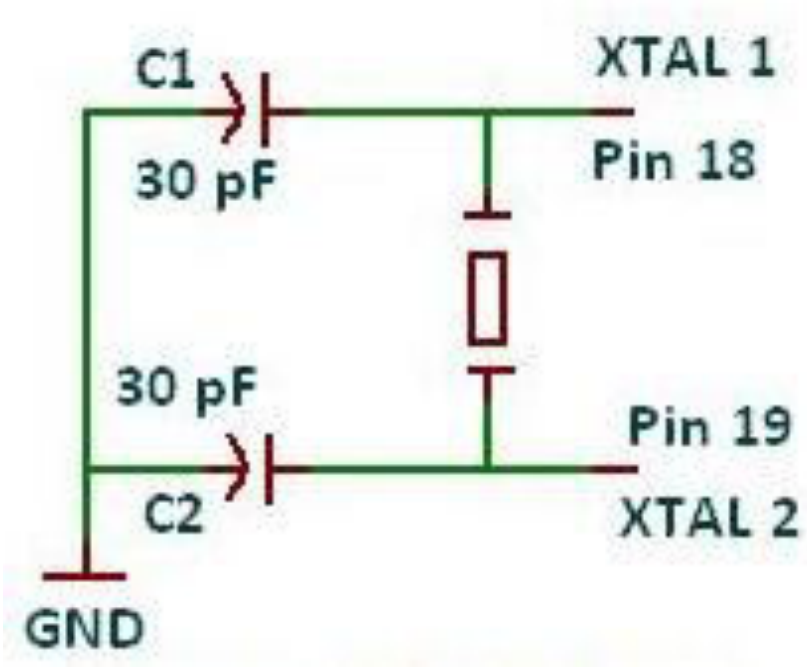


Fig. Oscillator and Timing Circuitry

The basic setup of microcontroller (Contd.)

RESET PIN CONNECTION (RST): The reset pin i.e. pin number 9 is used to reset the program just like we restart a computer, the program starts executing from the very beginning. When reset is used the program counter is set back to zero and the values in all other registers are lost.

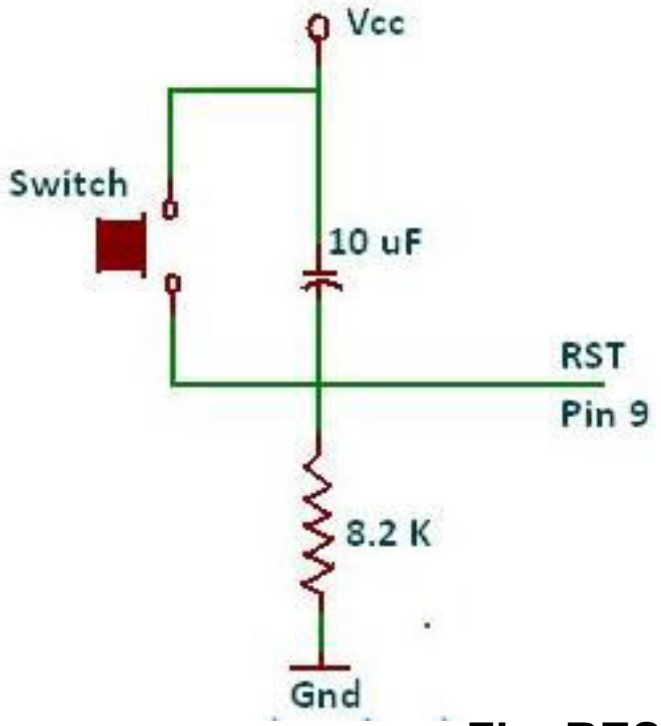


Fig. RESET PIN CONNECTION (RST)

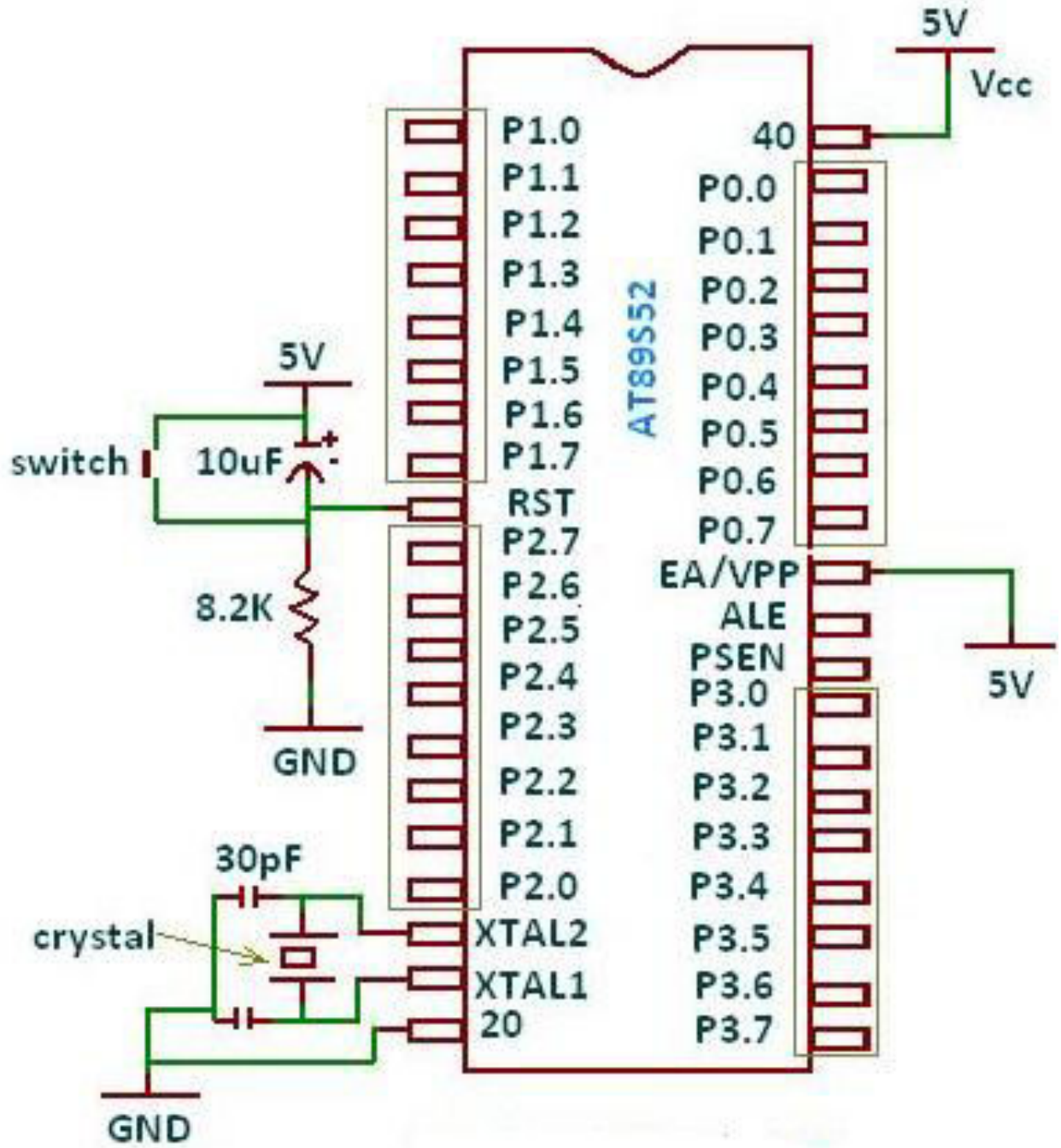
The basic setup of microcontroller (Contd.)

EA/VPP: The EA pin no. 31 is known as **external access**. The 8051 microcontrollers have on chip ROM so the EA/VPP is set to one (or Vcc). In case there is no on- chip ROM, like in case of 8031 we set this pin equal to zero. This pin cannot be left unconnected.

PSEN: This is an output pin. PSEN stands for "**program store enable**". In 8051 we leave this pin unconnected until otherwise mentioned.

ALE/PROG: ALE refers to address latch enable. In 8051 we leave this pin unconnected until otherwise mentioned.

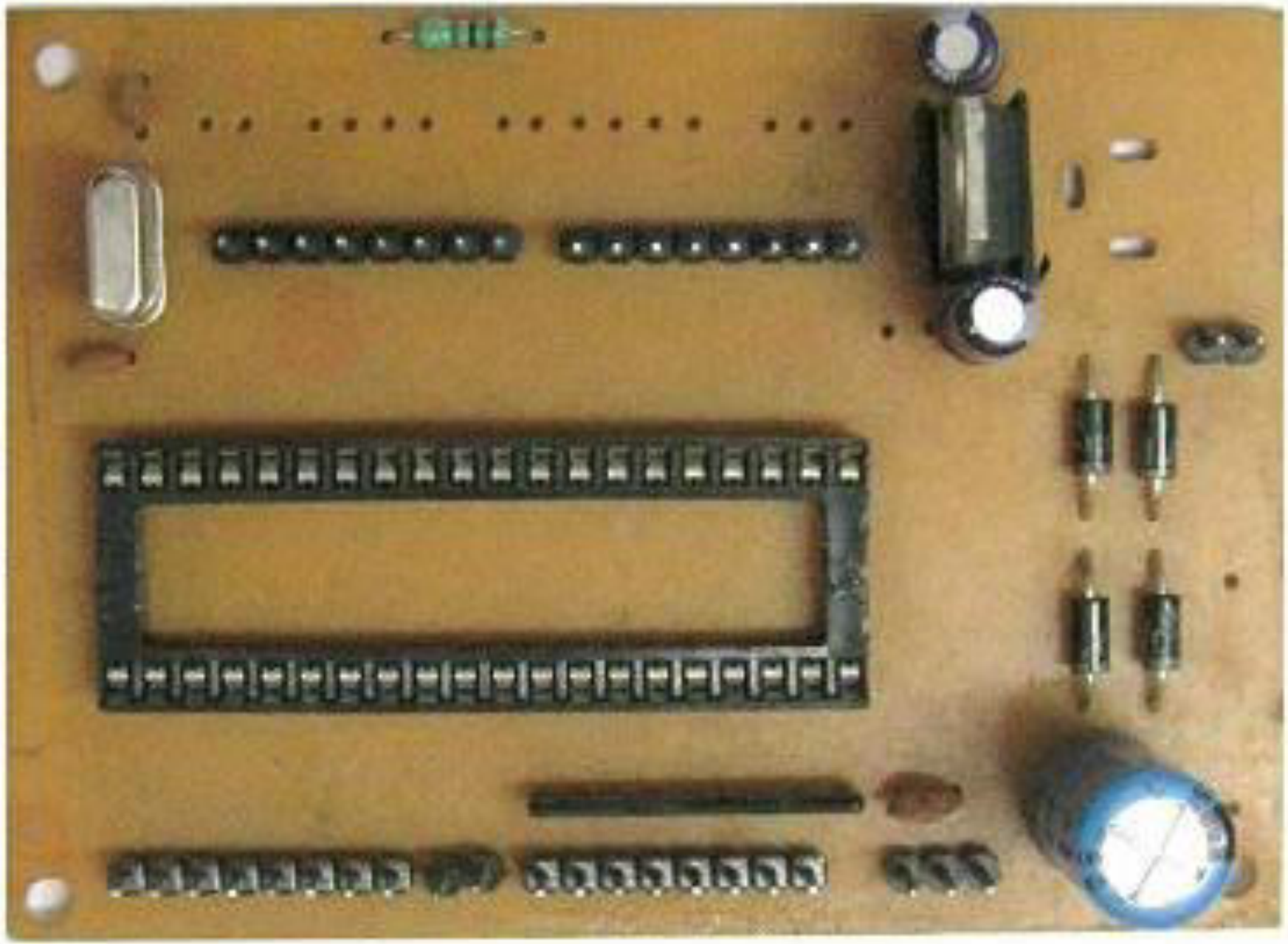
The basic setup of microcontroller



The basic setup of microcontroller (Contd.)

This is the basic setup of microcontroller that you need to have in order to perform an 8051 activity. The pins ALE/ PROG, EA/ VPP and PSEN may have some modifications.

The basic setup of microcontroller (Contd.)



The basic setup of microcontroller (Contd.)

This is a module that has been prepared by us. You can see that a 40 pin base has been mounted on the PCB (printed circuit board) instead of directly soldering the microcontroller to PCB which maybe damaged while soldering.

8051 Microcontroller Memory Organization

Hence, it is clear that the memory is an important part of the 8051 Microcontroller Architecture. So, it is important for us to understand the 8051 Microcontroller Memory Organization i.e. how memory is organized, how the processor accesses each memory and how to interface external memory with 8051 Microcontroller. 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.

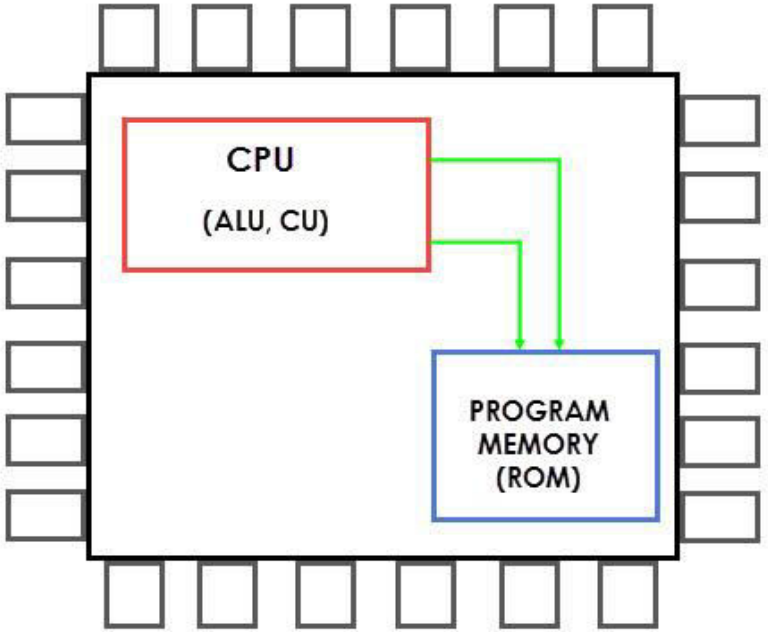
8051 Microcontroller has both Internal ROM and Internal RAM. If the internal memory is inadequate, you can add external memory using suitable circuits.

Program Memory (ROM) of 8051 Microcontroller

In 8051 Microcontroller, the code or instructions to be executed are stored in the Program Memory, which is also called as the ROM of the Microcontroller. The original 8051 Microcontroller by Intel has 4KB of internal ROM.

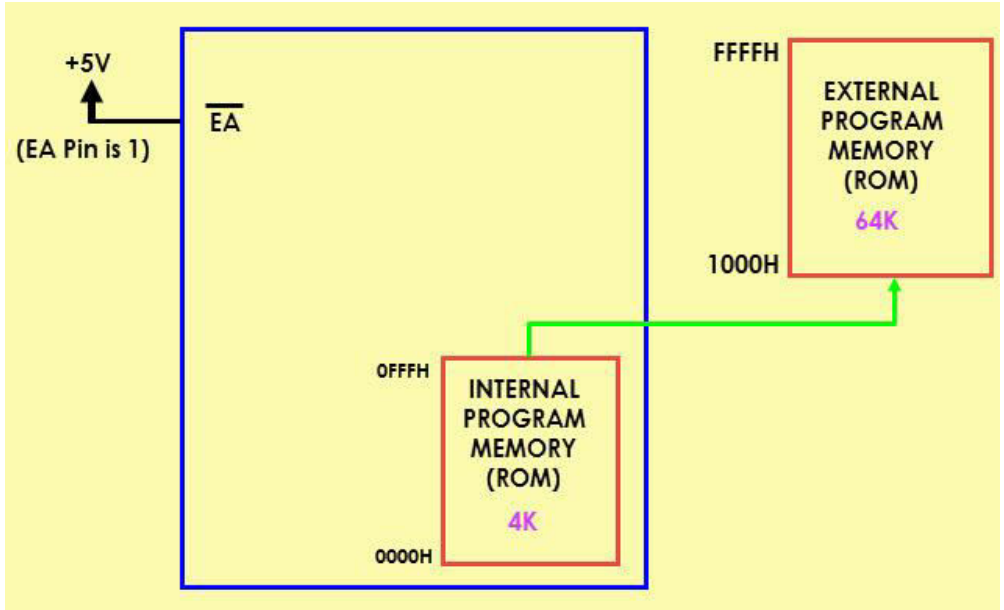
Some variants of 8051 like the 8031 and 8032 series doesn't have any internal ROM (Program Memory) and must be interfaced with external Program Memory with instructions loaded in it.

Almost all modern 8051 Microcontrollers, like 8052 Series, have 8KB of Internal Program Memory (ROM) in the form of Flash Memory (ROM) and provide the option of reprogramming the memory.



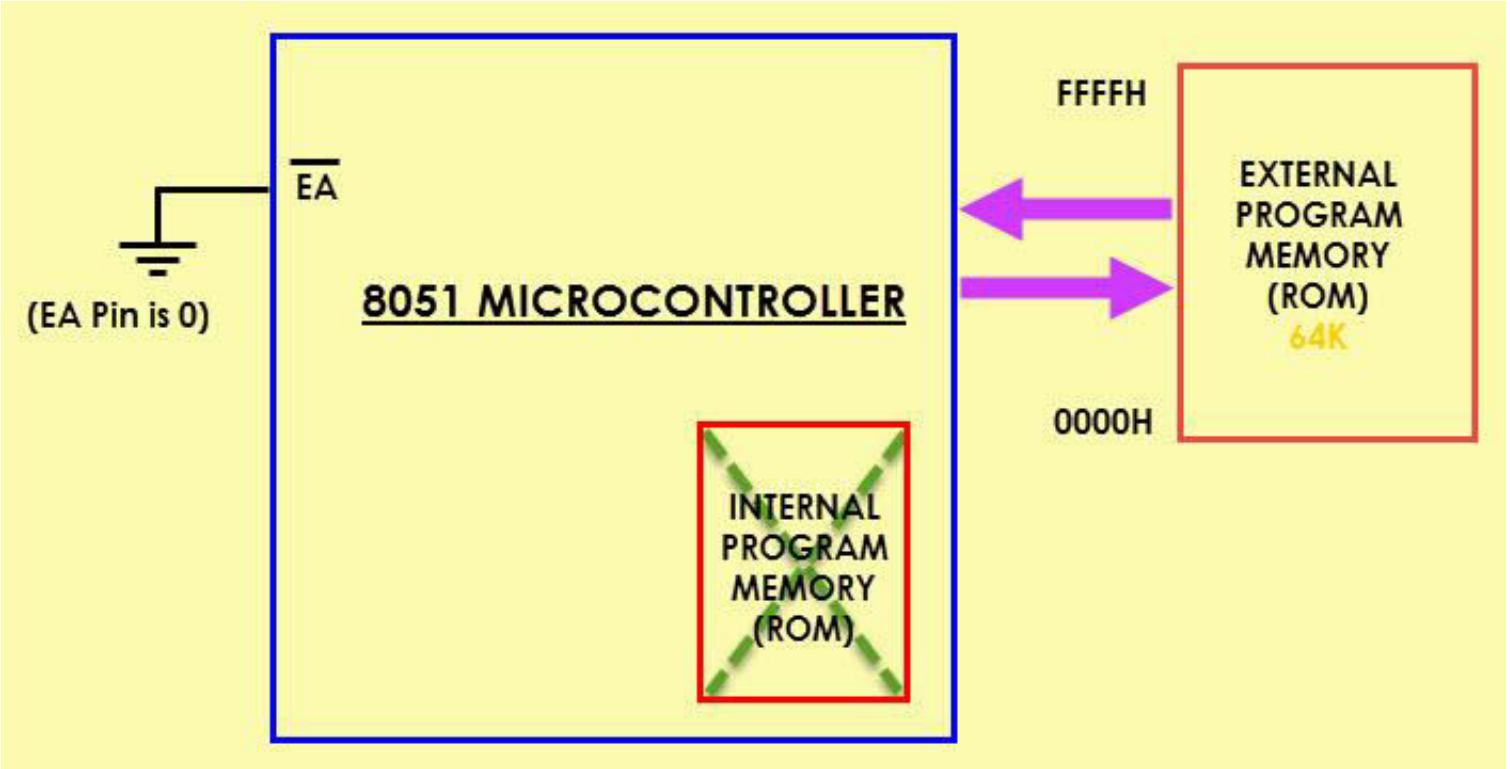
Program Memory (ROM) of 8051 Microcontroller(Cont'd)

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.

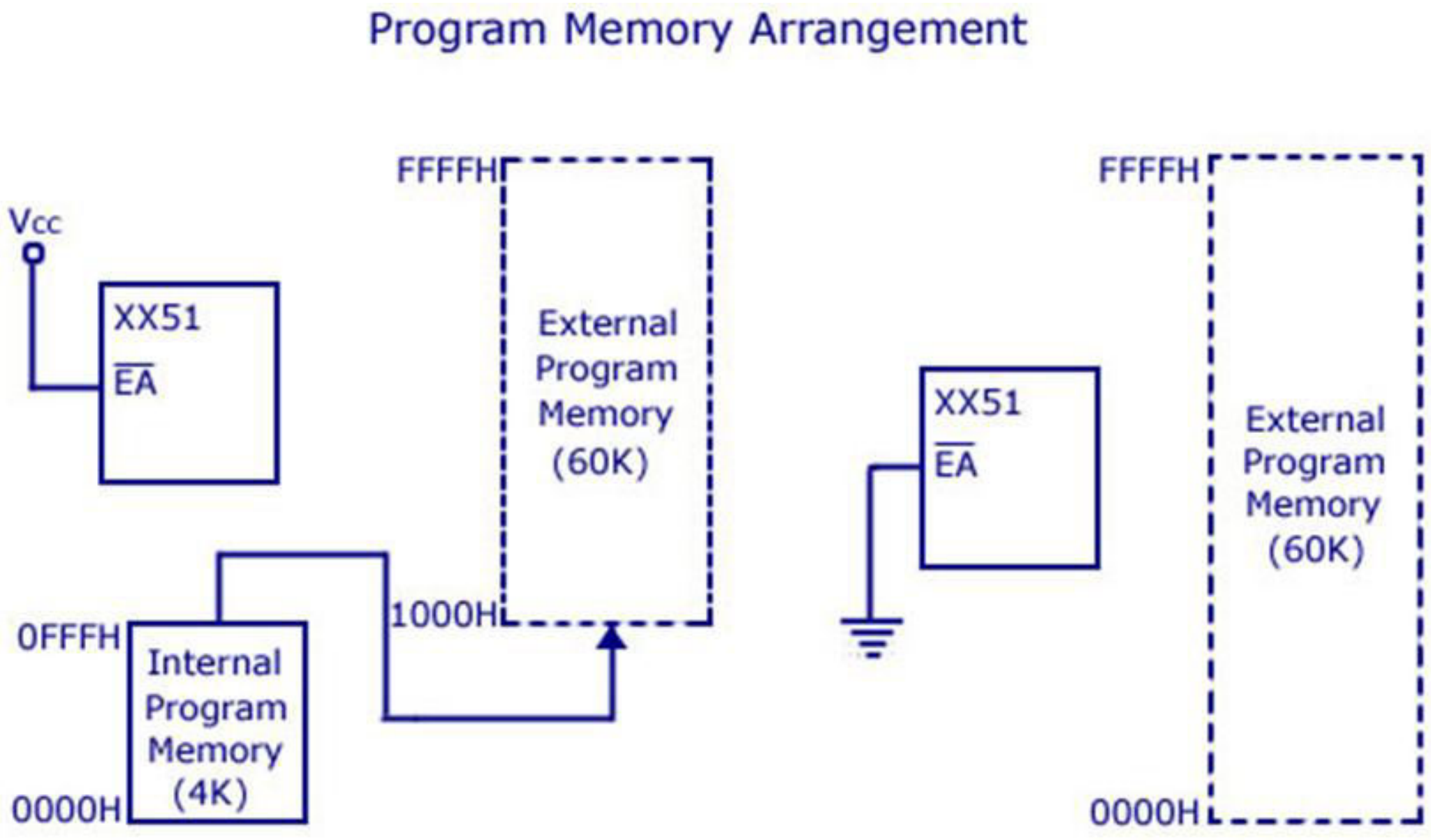


Program Memory (ROM) of 8051 Microcontroller(Cont'd)

There is another way to fetch the instructions: ignore the Internal ROM and fetch all the instructions only from the External Program Memory (External ROM). For this scenario, the EA Pin must be connected to GND. In this case, the memory addresses of the external ROM will be from 0000H to FFFFH.



Program Memory (ROM) of 8051 Microcontroller(Cont'd)



Data Memory (RAM) of 8051 Microcontroller

The Data Memory or RAM of the 8051 Microcontroller stores temporary data and intermediate results that are generated and used during the normal operation of the microcontroller. Original Intel's 8051 Microcontroller had 128B of internal RAM.

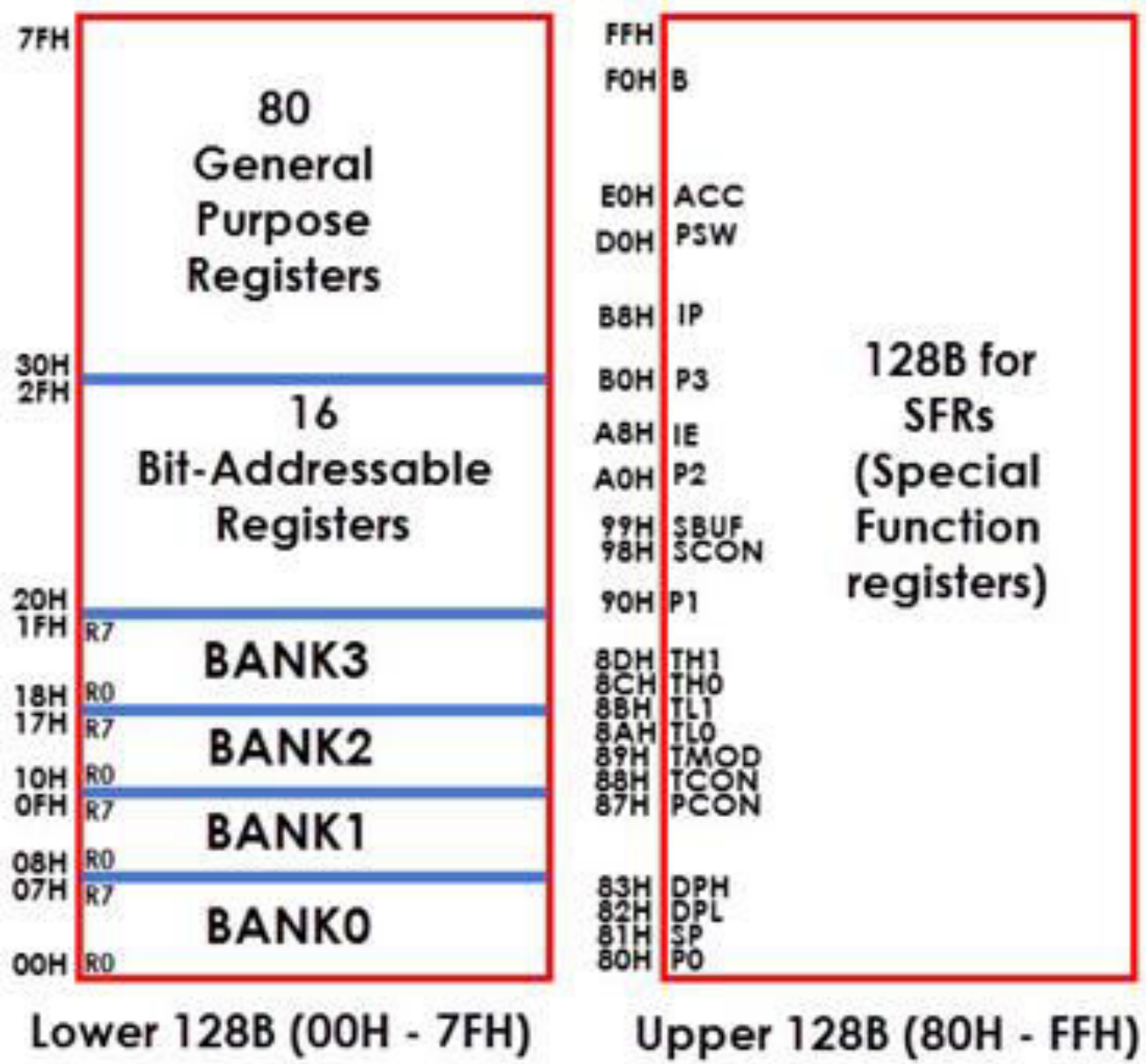
But almost all modern variants of 8051 Microcontroller have 256B of RAM. In this 256B, the first 128B i.e. memory addresses from 00H to 7FH is divided in to Working Registers (organized as Register Banks), Bit – Addressable Area and General Purpose RAM (also known as Scratchpad area).

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.

Data Memory (RAM) of 8051 Microcontroller

- **Data memory**
  - This is the area used to store data to be used in a program
  - Has 128byte internal memory and is capable of interfacing external memory
  - The 128 byte internal memory is divided into three parts
    - Register banks (bank #0,#1, #2, #3)
    - Bit addressable area (used to store bit values such as status of motors, LEDS etc)
    - General purpose storage area
  - Special function registers (80-FF0) is used for
    - Instruction register
    - Data pointer
    - Stack pointer
    - PSW, timers, IO ports, ACC and B registers

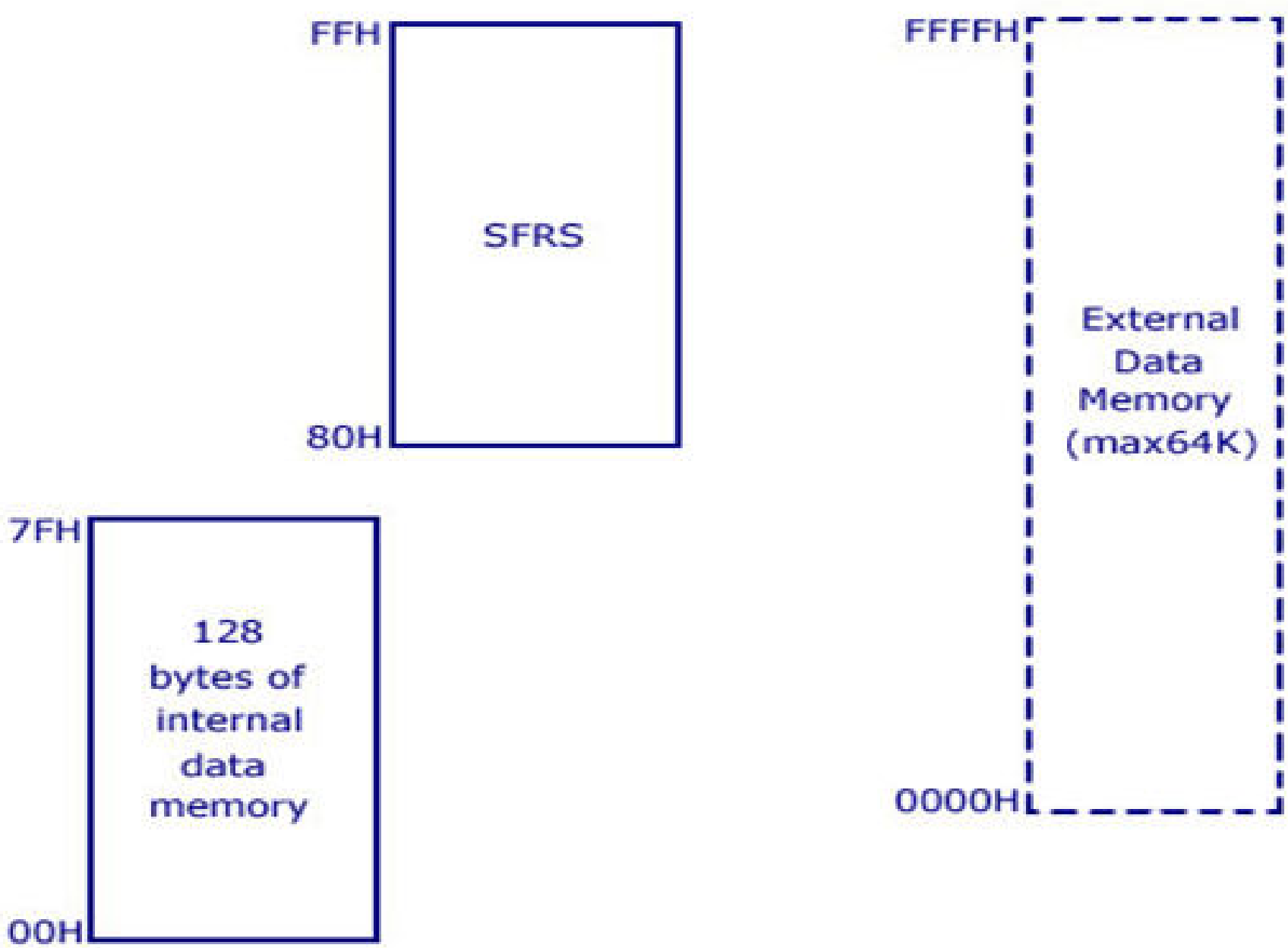
Data Memory (RAM) of 8051 Microcontroller(cont'd)



Data Memory (RAM) of 8051 Microcontroller(cont'd)

The lower 32 bytes are divided into 4 separate banks. **Each register bank has 8 registers of one byte each**. A register bank is selected depending upon two bank select bits in the PSW (Program status word) register. Next 16 bytes are bit addressable. In total, 128bits (16X8) are available in bit addressable area. Each bit can be accessed and modified by suitable instructions. The bit addresses are from 00H (LSB of the first byte in 20H) to 7FH (MSB of the last byte in 2FH). Remaining 80bytes of RAM are available for general purpose.

Data Memory (RAM) of 8051 Microcontroller(cont'd)



Data Memory (RAM) of 8051 Microcontroller(cont'd)

The 4 banks are named as Bank0, Bank1, Bank2 and Bank3. Each Bank consists of 8 registers named as R0 – R7. Each Register can be addressed in two ways: either by name or by address. To address the register by name, first the corresponding Bank must be selected. In order to select the bank, we have to use the RS0 and RS1 bits of the Program Status Word (PSW) Register (RS0 and RS1 are 3^{rd} and 4^{th} bits in the PSW Register). When addressing the Register using its address i.e. 12H for example, the corresponding Bank may or may not be selected. (12H corresponds to R2 in Bank2).

Data Memory (RAM) of 8051 Microcontroller(cont'd)

The next 16B of the RAM i.e. from 20H to 2FH are Bit – Addressable memory locations. There are totally 128 bits that can be addressed individually using 00H to 7FH or the entire byte can be addressed as 20H to 2FH. For example 32H is the bit 2 of the internal RAM location 26H.

The final 80B of the internal RAM i.e. addresses from 30H to 7FH, is the general purpose RAM area which are byte addressable. These lower 128B of RAM can be addressed directly or indirectly. The upper 128B of the RAM i.e. memory addresses from 80H to FFH is allocated for *Special Function Registers* (SFRs). SFRs control specific functions of the 8051 Microcontroller. Some of the SFRs are I/O Port Registers (P0, P1, P2 and P3), PSW (Program Status Word), A (Accumulator), IE (Interrupt Enable), PCON (Power Control), etc.

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

SFRs Memory addresses are only direct addressable. Even though some of the addresses between 80H and FFH are not assigned to any SFR, they cannot be used as additional RAM area.

In some microcontrollers, there is an additional 128B of RAM, which share the memory address with SFRs i.e. 80H to FFH. But, this additional RAM block is only accessed by indirect addressing.

Interfacing External Memory with 8051 Microcontroller

We have seen that a typical 8051 Microcontroller has 4KB of ROM and 128B of RAM (most modern 8051 Microcontroller variants have 8K ROM and 256B of RAM).

The designer of an 8051 Microcontroller based system is not limited to the internal RAM and ROM present in the 8051 Microcontroller. There is a provision of connecting both external RAM and ROM i.e. Data Memory and Program.

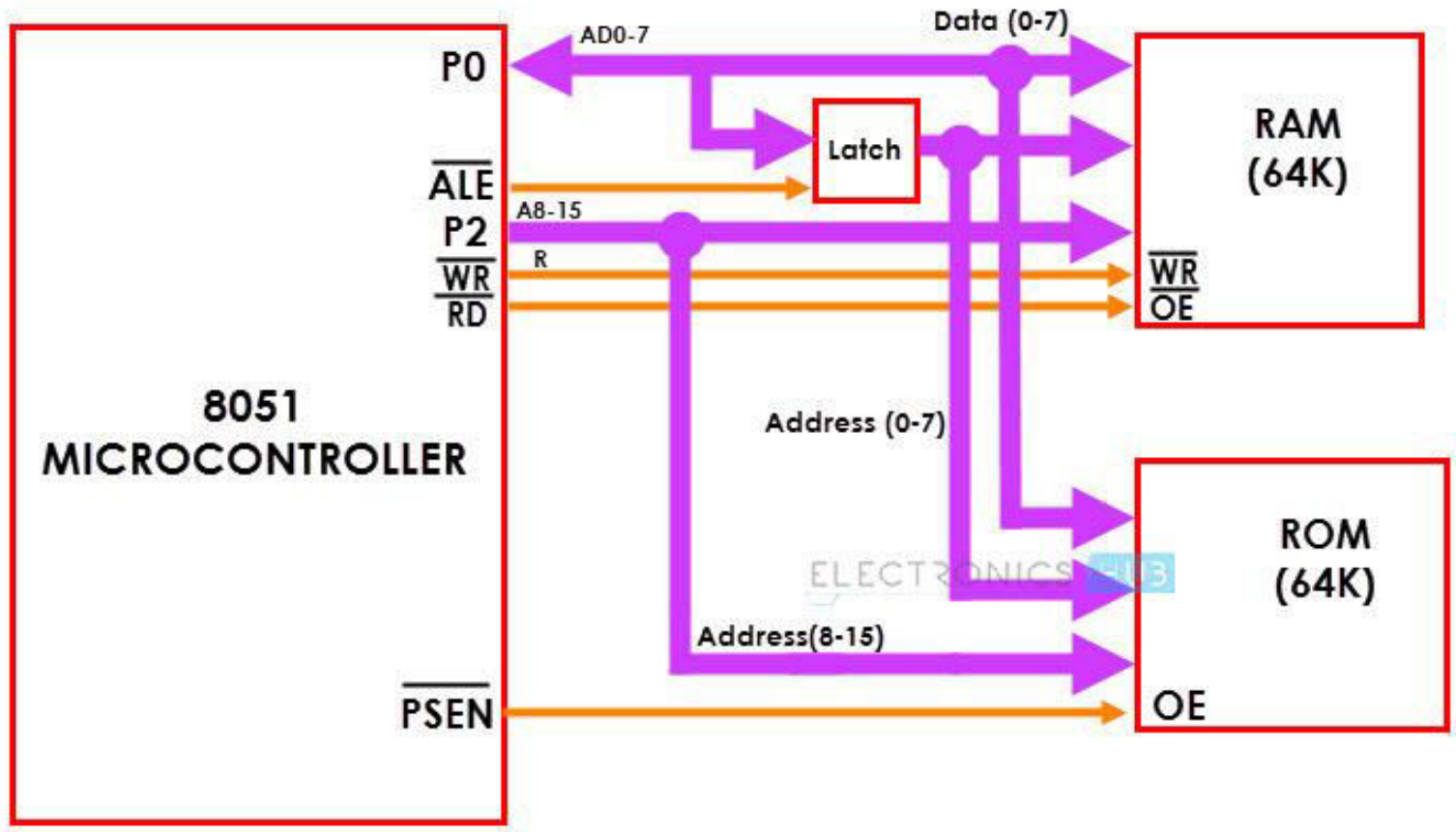
The reason for interfacing external Program Memory or ROM is that complex programs written in high – level languages often tend to be larger and occupy more memory.

Another important reason is that chips like 8031 or 8032, which doesn't have any internal ROM, have to be interfaced with external ROM.

A maximum of 64KB of Program Memory (ROM) and Data Memory (RAM) each can be interface with the 8051 Microcontroller.

The following image shows the block diagram of interfacing 64KB of External RAM and 64KB of External ROM with the 8051 Microcontroller.

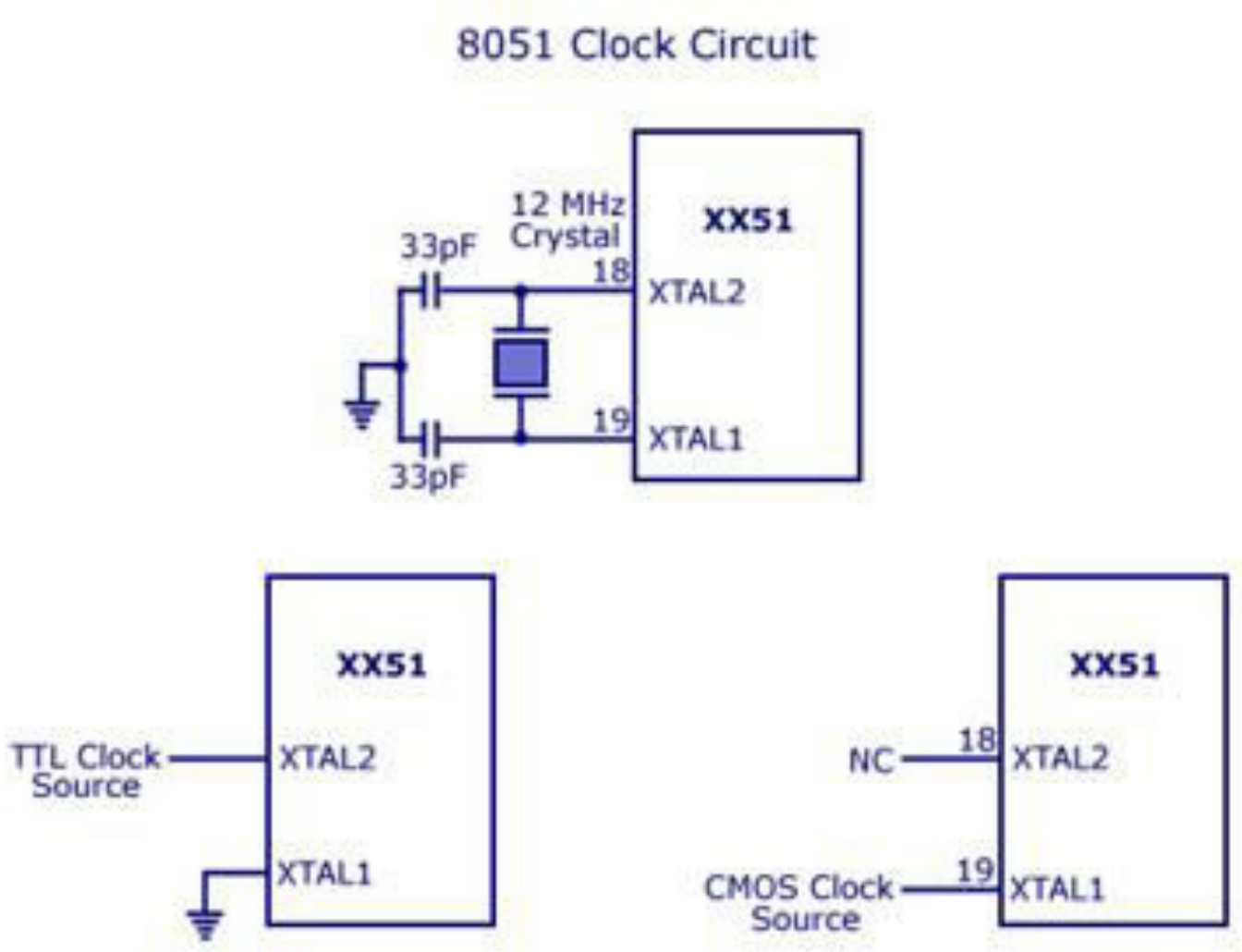
Interfacing External Memory with 8051 Microcontroller



In this fig. we have seen the 8051 Microcontroller Memory Organization, Internal ROM and RAM and how to interface external ROM and RAM with 8051 Microcontroller.

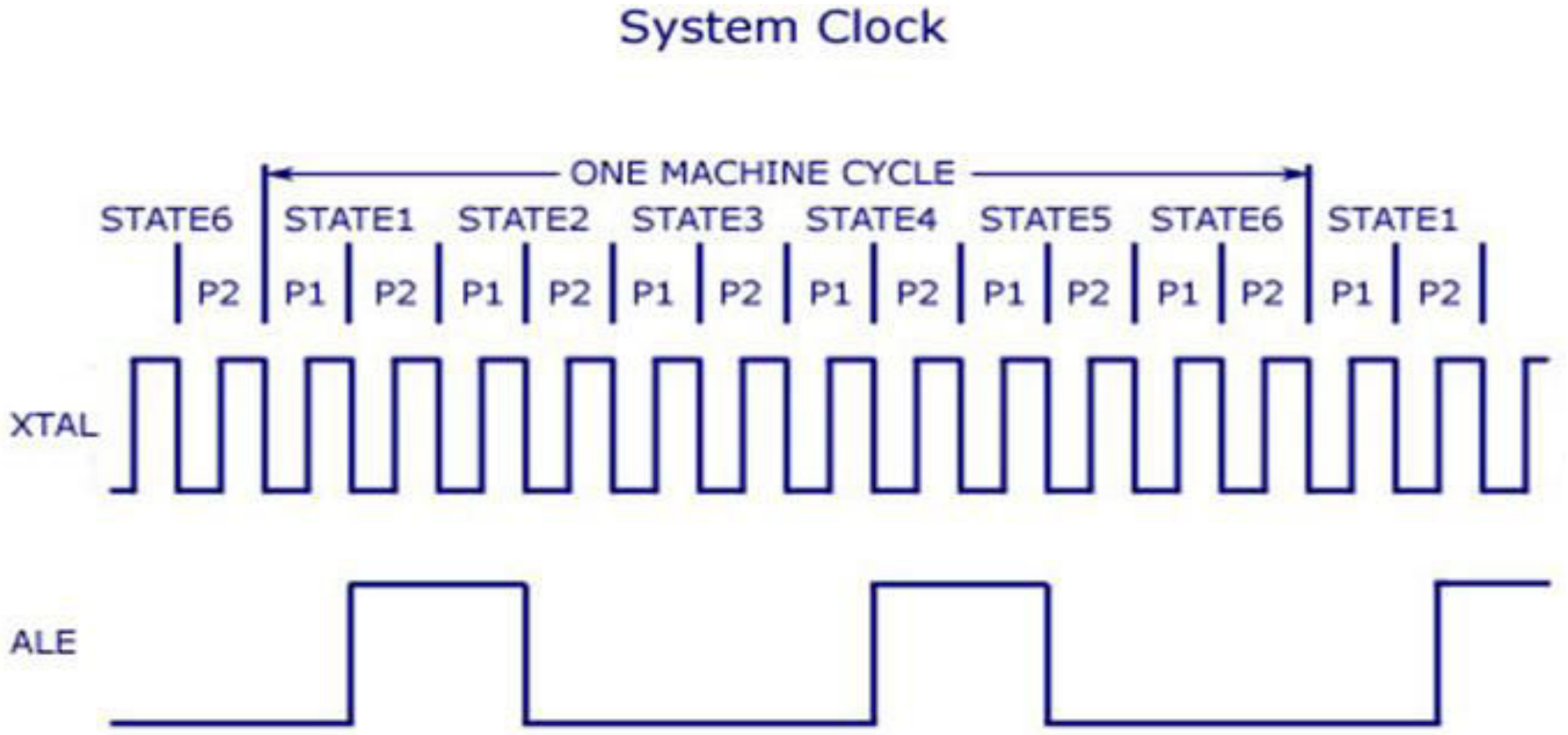
Clock connection

- Clock connection
  - For 8051 microcontroller operation, one of the external inputs needed is the clock.
  - There are three variations for this



Clock connection

- **A machine cycle** in 8051 contains six states or 12 clock cycles

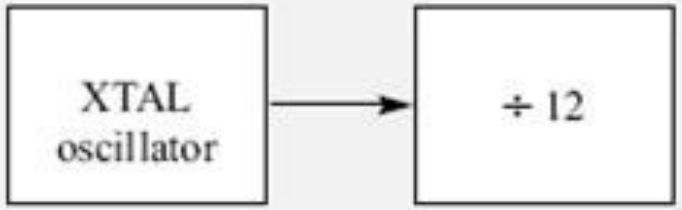


Example

Find the timer's clock frequency and its period for various 8051-based systems, with the following crystal frequencies.

(a) 12 MHz
(b) 16 MHz
(c) 11.0592 MHz

Solution:



(a) 1/12 × 12 MHz = 1 MHz and T = 1/1 MHz = 1 µs

(b) 1/12 × 16 MHz = 1.333 MHz and T = 1/1.333 MHz = .75 µs

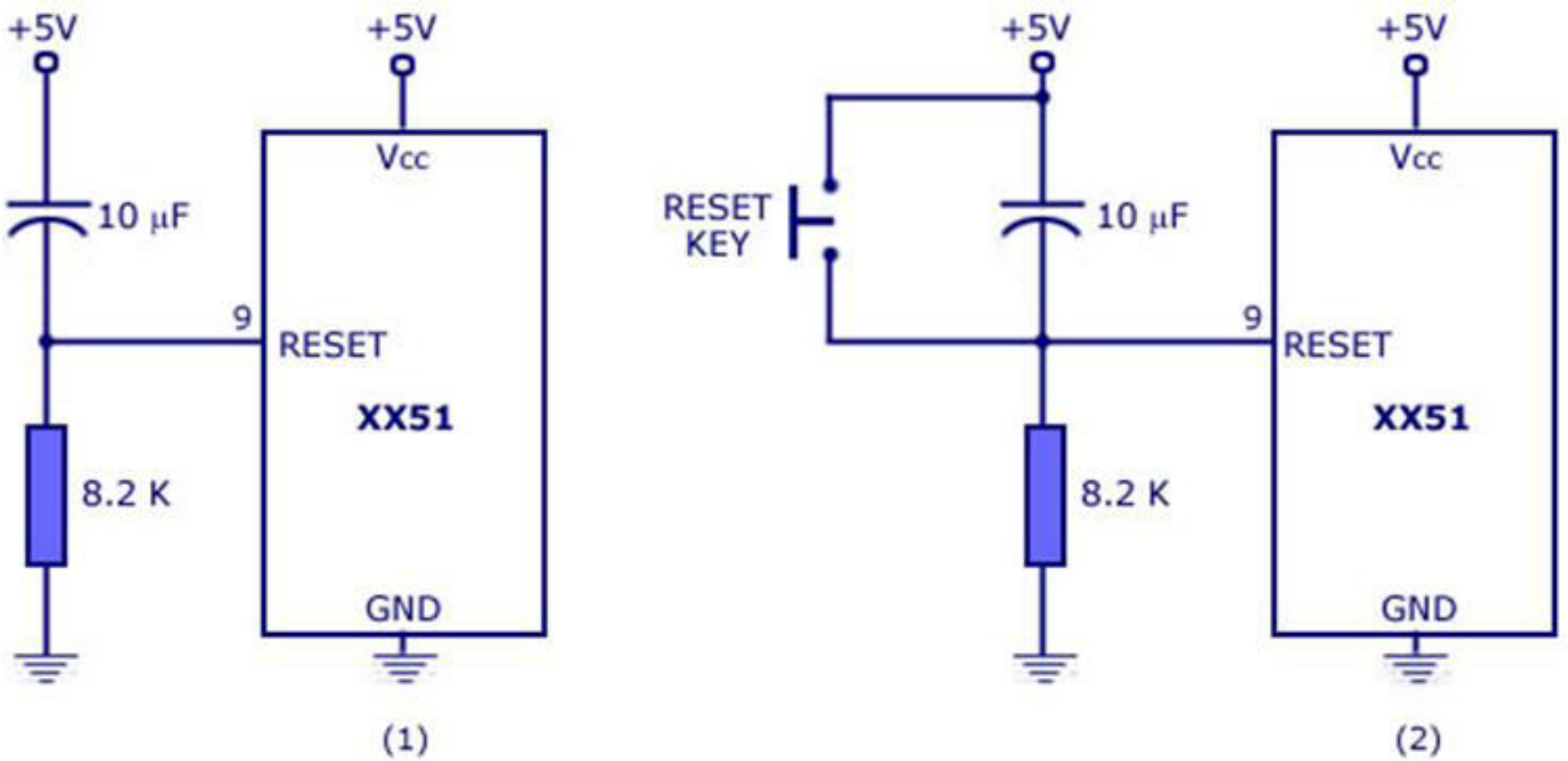
(c) 1/12 × 11.0592 MHz = 921.6 kHz;
T = 1/921.6 kHz = 1.085 µs

NOTE THAT 8051 TIMERS USE 1/12 OF XTAL FREQUENCY, REGARDLESS OF MACHINE CYCLE TIME.

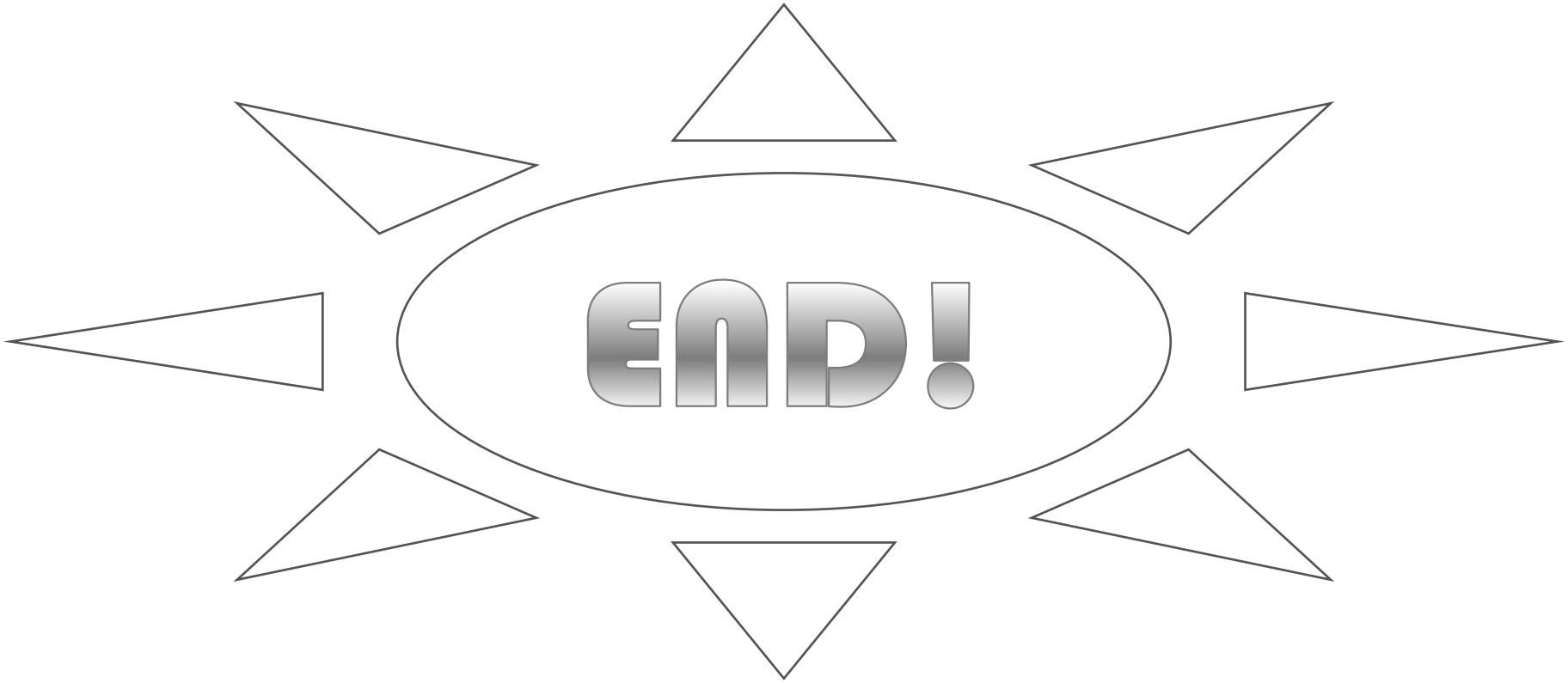
Reset operation

- **Reset operation**
  - The other external input in 8051 is the reset input
  - It is activated by making pin 9 high
  - When reset occurs
    - Program counter is cleared
    - Register bank 0 is selected
    - Stack pointer is initialized to 07H
    - All ports are written with FFh

Reset operation (Contd.)



(1) Power-on Reset Circuit and (2) With Manual Reset Option


END!

CHAPTER-3

C PROGRAMMING FOR MICROCONTROLLER

8051
MICROCONTROLLER

TECHNICAL AND VOCATIONAL TRAINING INSTITUTE
Department of Electrical electronics technology

Fundamentals of microprocessors/microcontroller

EETe 3032

By Zemenu T. /2022

Outlines

1. Introduction to programming
2. Basic C Programming
3. Variables and constants, Mathematical & logic operations, Conditions and loops

Introduction to programming

All the controllers can be programmed with the help of the programming languages.

What language should you use? Three languages are common with the 8051 family i.e C, BASIC, and assembly. These programming languages can be of a low level language or a high level language depends on the flexibility of the person working with and the size of the project he is going to work with. The low level language is the Assembly language and examples of high level languages are Embedded C, Embedded C++. Now a day the trends has been to switch to **C**. Most developers now rely on the C language. Along with benefits of high-level languages like C, the elaborate software tools of today represent a dramatic shift from the difficult, detail oriented programming techniques necessary for the first microcontroller. In this course all the programming examples use C language.

What is a Programming Language?

Programming in the sense of Microcontrollers (or any computer) means writing a sequence of **instructions** that are executed by the processor in a particular order to perform a predefined task. Programming also involves debugging and troubleshooting of instructions and instruction sequence to make sure that the desired task is performed.

Like any language, Programming Languages have certain words, grammar and rules. There are two types or levels of Programming Languages for 8051 Microcontroller. These levels are based on how closely the statements in the language resemble the operations or tasks performed by the Microcontroller.

The two levels of Programming Languages are:

- Low level (Assembly) Language
- High-level Language

There are two language that you can use to program the 8051

1. Assembly language

It is basically a low level programming language for computers and is machine dependant which in simple words means that the code you write for one machine will not be compatible with some other machine. It is composed of words-like instructions called MNEMONICS. Assembly language is very specific and a bit complicated. Being a low level language, it is not easily understandable at first sight but being very specific, it gives you total control over what you do. You can even determine the total machine cycles of the whole cod you have written. You can call each register by name store/manipulate values in it or you can generate exact time delays by using simple statements.

Assemblers

An assembler is a program that takes basic computer instructions (called as assembly language) and converts them into a pattern of bits that the computer's processor can use to perform its basic operations. An assembler creates object code by translating assembly instruction mnemonics into opcodes, resolving symbolic names to memory locations. Assembly language uses a mnemonic to represent each low-level machine operation (opcode).

2. C language

C language is not so specific and gives you freedom free hand forget about the names specific to some device. You can start writing code in your natural way and the compiler will take care of the internal operations for you. This makes things a lot more easier as composed to assembler language. A program called **Compiler** will convert the Programs written in High-level languages to Machine Code.

Examples

ASSEMBLY LANGUAGE CODE

code.asm

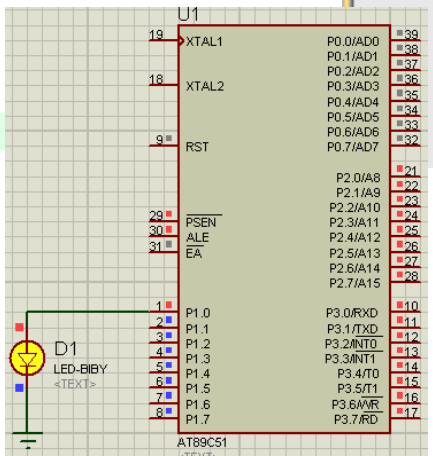
```
org 00H

;MAIN PROGRAM

toggle:  MOV P1, #01H      ; move 00000001 to PORT1
         CALL delay        ; execute dalay
         MOV A, P1         ; move PORT1 value to accumulator
         CPL A             ; complement PORT1 value
         MOV P1, A         ; move 11111110 to PORT1
         CALL delay        ; execute delay

         sjmp toggle


;DELAY SUB-ROUTINE

delay:   MOV R5, #10       ; load register R5 with 10
third:   MOV R6, #200      ; load register R6 with 200
second:  MOV R7, #200      ; load register R7 with 200

         DJNZ R7, $        ; decrement R7 till it is zero
         DJNZ R6, second   ; decrement R6 till it is zero
         DJNZ R5, third    ; decrement R5 till it is zero

         ret               ; go back to main program
END
```

C LANGUAGE CODE

c_code.c

```
#include <REGX51.H>

void delay( unsigned int );

void main()
{
  while(1)
  {
    P1 = 0x01;
    delay(1000);

    P1 = ~P1;
    delay(1000);
  }
}

//DELAY FUNCTION ( mS )
void delay( unsigned int time )
{
    unsigned int i;
    unsigned int j;

    for( i=0; i<time; i++ )
    for( j=0; j<1275; j++ );
}
```

PROTEUS SIMULATION DESIGN



Why program the 8051 in C?

Compilers produce hex files that we download into the ROM of the microcontroller. The size of the hex file produced by the compiler is one of the main concerns of microcontroller programmers, for two reasons:

1. Microcontrollers have limited on-chip ROM.
2. The code space for the 8051 is limited to 64K bytes.

How does the choice of programming language affect the compiled program size? While Assembly language produces a hex file that is much smaller than C, programming in Assembly language is tedious and time consuming. C programming, on the other hand, is less time consuming and much easier to write, but the hex file size produced is much larger than if we used Assembly language. The following are some of the major reasons for writing programs in C instead of Assembly:

1. It is easier and less time consuming to write in C than Assembly.
2. C is easier to modify and update.
3. You can use code available in function libraries.
4. C code is portable to other microcontrollers with little or no modification.

The study of C programming for the 8051 is the main topic of this chapter.

Factors for Selecting the Programming Language

The following are few factors that are to be considered while selecting the Programming Language for the development of Embedded Systems.

- **Size**: The memory that the program occupies is very important as Embedded Processors like Microcontrollers have a very limited amount of ROM.
- **Speed**: The programs must be very fast i.e. they must run as fast as possible. The hardware should not be slowed down due to a slow running software.
- **Portability**: The same program can be compiled for different processors.
- **Ease of Implementation**
- **Ease of Maintenance**
- **Readability**

Earlier Embedded Systems were developed mainly using Assembly Language. Even though Assembly Language is closest to the actual machine code instructions, the lack of portability and high amount of resources spent on developing the code, made the Assembly Language difficult to work with.

There are other high-level programming languages that offered the above mentioned features but none were close to C Programming Language.

Embedded System Design Steps

- An embedded system contains **hardware** and **software** system
- **Design of an embedded system means**
  - Specifying hardware required
  - Specifying sensors and actuators to be used
  - System integration and Writing the code required
  - Testing the system performance
- **Specifying the hardware required**
  - This needs the understanding of the objective to be achieved or what the embedded system has to do
  - **Hardware required means**
    - The processor to be used
    - Required interface (external memory, I/O ports etc)
    - Power supply and clock
    - Reset circuits for the processor
  - In the selection, the **cost and performance** should be optimized

Embedded System Design (Contd.)

- **Specify sensors and actuators required**
  - In design of embedded system, there will always be a control to be performed
  - Hence the external environment to be sensed and the output signal required needs to be identified
  - This will determine the sensors and actuators to be used
    - **Sensors** can have digital or analog output
    - **Actuators** can be driven by analog or digital signal
    - Required interface like ADC or DAC need to identified

Embedded System Design (Contd.)

- **System integration and code writing**
  - **System integration is assembling the hardware parts**
    - Connecting the microcontroller with the system inputs
    - Connecting external ports and memory
    - Connecting sensors and actuators required
  - **Code writing**
    - Program for an embedded system can be written in assembly language or HLP
    - Assembly language programming needs **assembler** while HLP needs **compiler**
    - The best practice is to use **HLP languages** and emulators to write the program and test its performance before downloading it to the processor

Embedded System Design (Contd.)

- **System integration and code writing**
  - **Code writing**
    - There are various types of software used for developing embedded systems
    - Some of them even are used to test the hardware integration also (protues 8)
    - The most widely used ones are
      - Code composer studio (CSS)
      - **Kiel µvision**
      - EDSIM51
      - Xilinx etc

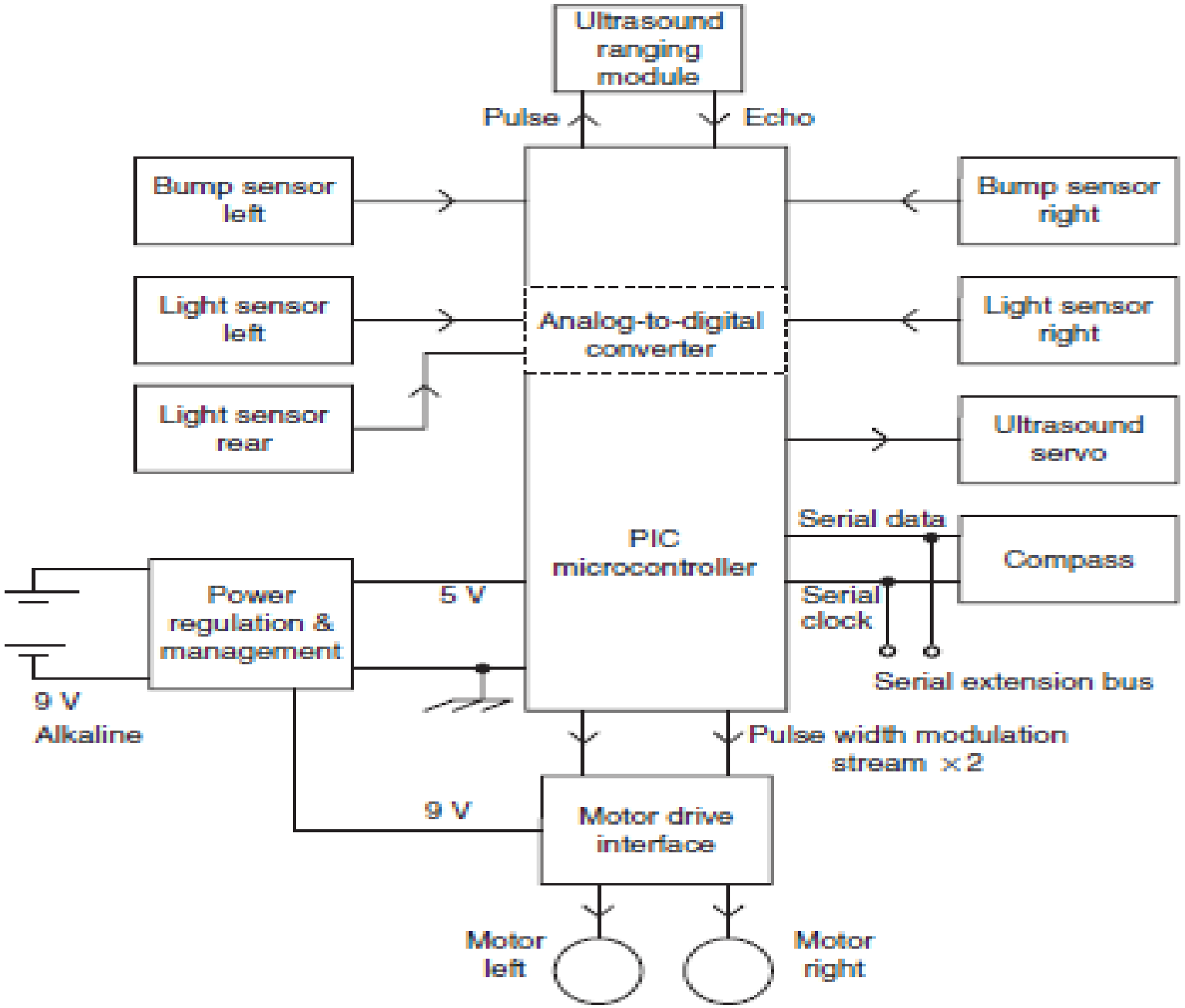
Embedded System Design (Contd.)

- **The code writing/ firm ware development/ consists of**
  - **Writing** the source code using editor
  - **Translating** the code using assembler or compiler
  - **Debugging** for error
  - **Simulating** the system (microcontroller with peripherals , emulator, starter kits, mother board and daughter cards)
  - **Downloading** the code to the target device or burning the EEPROM

Embedded System Design example

- **Autonomous guided vehicle**
  - The vehicle is to navigate through an environment where there are obstacles
  - The vehicle has to avoid obstacles
    - Two bump detectors
    - Ultrasound detector
    - Two motor actuators
    - Encoder to calculate distance
    - Motor drive circuit
  - Microcontroller reads the environment and droves the actuator using a program

Embedded System Design example



The Basics Of Microcontroller Programming

A microcontroller does not know what to do by itself. It's your job to tell it what you want it to do.

So, you need to:

•write program code on your computer

•compile the code with a compiler for the microcontroller you are using

upload the compiled version of your program to your microcontroller

Write your program code

The first step is to write your program code. This is usually done is C. But some compilers support other languages as well. Find out what other people who are using the same microcontroller are doing.

Compile your code for your microcontroller

Before you can upload your program to your microcontroller, you need to compile it. This means converting the code from human-readable code to machine-readable code.

Use a compiler that supports your microcontroller and compile your code into machine-code for your chip. A popular compiler for Atmel AVR microcontrollers is avr-gcc.

After compilation, you will have one or more files containing machine code. Then you need to upload these files to your microcontroller.

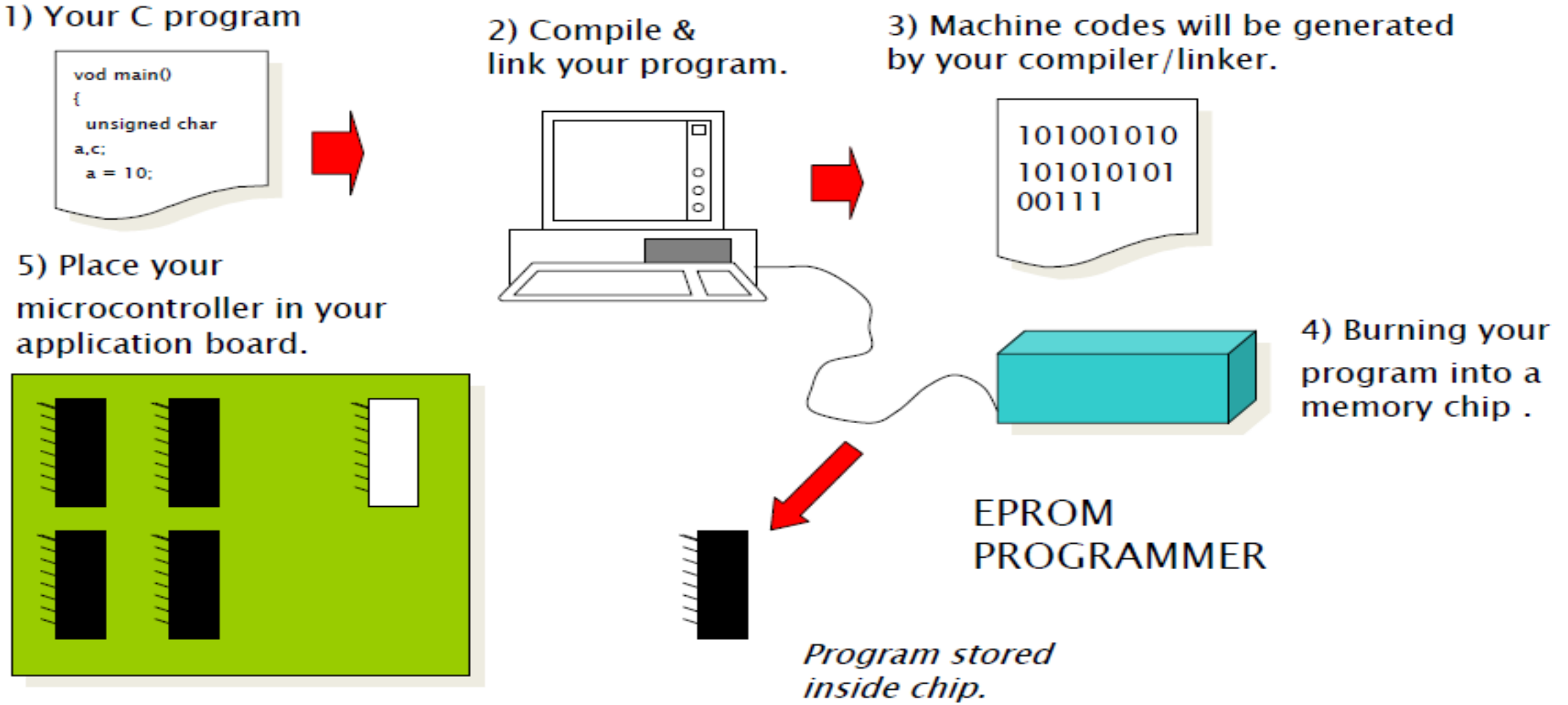
Upload the compiled file(s) to your microcontroller

Usually, it's one program file and a file for EEPROM and/or flash that you need to upload.

You need a physical connection from your computer to your microcontroller. Either you can use a dedicated programmer (such as the AVRISP for AVR microcontrollers), or if you have a USB programmable chip you can program it with a USB cable (my preferred method).

And you need a program for uploading the file(s). For 8051 chips, you can use **flash magic**

THE DEVELOPMENT PROCESS – *The Compiling Process*



Assembly programming

- Program is a list of instructions combined together to do a specific task
- Assembly programming using mnemonics
  - Has four columns
    - **Label** : name given to a program line
    - **Mnemonic**: specific code for the task to be performed
    - **Operand** : data needed to complete the instruction
    - **Comment**: clarification

C Programming

- Developed in Bell laboratories in 1970s
- Of higher level languages, C is the closest to assembly languages
  - bit manipulation instructions
  - pointers (indirect addressing)
- Most microcontrollers have available C compilers
- Writing in C simplifies code development for large projects.
- A **c program** contains
  - Comment
  - Declaration
  - Main directive
  - Program statements

C Programming (Contd.)

- Like most high level languages, C is a modular programming language (but NOT an object oriented language)
- Each task can be encapsulated as a function.
- Entire program is encapsulated in “main” function

C Programming

- **Main function**

```
void main()
{
    //Your one time initialization code will come here
while (1)
{
        //while 1 loop
        //This loop will have all your application code
        //which will run infinitely
}
}
```

Basics of Embedded C Program

Now that we have seen a little bit about Embedded Systems and Programming Languages, we will dive in to the basics of Embedded C Program. We will start with two of the basic features of the Embedded C Program: **Keywords and Data types.**

Keywords in Embedded C

A Keyword is a special word with a special meaning to the compiler (a C Compiler for example, is a software that is used to convert program written in C to Machine Code). Are reserved words which are recognized by the compiler. For example, if we take the Keil's Cx51 Compiler (a popular C Compiler for 8051 based Microcontrollers) the following are some of the keywords:

- ♦bit
- ♦sbit
- ♦ sfr
- ♦small
- ♦large

These are few of the many keywords associated with the Cx51 C Compiler along with the standard C Keywords.

Data Types in Embedded C

Data Types in C Programming Language (or any programming language for that matter) help us declaring variables in the program. There are many data types in C Programming Language like signed int, unsigned int, signed char, unsigned char, float, double, etc. In addition to these there few more data types in Embedded C.

The following are the extra data types in Embedded C associated with the Keil's Cx51 Compiler.

- ♦ bit
- ♦ sbit
- ♦ sfr
- ♦ sfr16

Variables in C

- All variables must be declared at top of program, before the first statement.
- Declaration includes *type* and list of variables.
  Example:
```
void main (void) {
    int var, tmp;
```
- Types:
  - **int** (16-bits in our compiler)
  - **char** (8-bits)
  - **short** (16-bits)
  - **long** (32-bits)
  - **sbit** (1-bit)
  - others that we will discuss later

Data types in C

Table A6.1 C keywords associated with data type and structure definition

Word	Summary meaning	Word	Summary meaning
char	A single character, usually 8-bit	**signed**	A qualifier applied to **char** or **int** (default for **char** and **int** is signed)
const	Data that will not be modified	**sizeof**	Returns the size in bytes of a specified item, which may be variable, expression or array
double	A 'double precision' floating-point number	**struct**	Allows definition of a data structure
enum	Defines variables that can only take certain integer values	**typedef**	Creates new name for existing data type
float	A 'single precision' floating-point number	**union**	A memory block shared by two or more variables, of any data type
int	An integer value	**unsigned**	A qualifier applied to **char** or **int** (default for **char** and **int** is signed)
long	An extended integer value; if used alone, integer is implied	**void**	No value or type
short	A short integer value; if used alone, integer is implied	**volatile**	A variable which can be changed by factors other than the program code

The following table shows some of the data types in Cx51 Compiler along with their ranges.

Data Type	*Bits (Bytes)*	*Range*
bit	1	0 or 1 (bit addressable part of RAM)
signed int	16 (2)	-32768 to +32767
unsigned int	16 (2)	0 to 65535
signed char	8 (1)	-128 to +127
unsigned	8 (1)	0 to 255
sbit	1	0 or 1 (bit addressable part of RAM)
sfr	8 (1)	RAM Addresses (80h to FFh)
sfr16	16 (2)	0 to 65535

Statements in c

- The other types of statements are **arithmetic or logic operations**
- Operations and symbols in c
  - Arithmetic: **+, -, *, /**
  - Relational comparisons: **>, >=, <, <=**
  - Equality comparisons: **==, !=**
  - Logical operators: **&&** (and), **||** (or)
  - Increment and decrement: **++, --**

Example:

```
if (x != y) && (c == b)
        {
        a=c + d*b;
        a++;
        }
```

Reserved words for program flow

Table A6.2 C keywords associated with program flow

Word	Summary meaning	Word	Summary meaning
break	Causes exit from a loop	**for**	Defines a repeated loop – loop is executed as long as condition associated with **for** remains true
case	Identifies options for selection within a **switch** expression	**goto**	Program execution moves to labelled statement
continue	Allows a program to skip to the end of a **for**, **while** or **do** statement	**if**	Starts conditional statement; if condition is true, associated statement is executed
default	Identifies default option in a **switch** expression, if no matches found	**return**	Returns program execution to calling routine, causing also return of any data value specified by function
do	Used with **while** to create loop, in which statement following **do** is repeated as long as **while** condition is true	**switch**	Used with **case** to allow selection of a number of alternatives; **switch** has an associated expression which is tested against a number of **case** options
else	Used with **if**, and precedes alternative statement used if **if** condition is not true	**while**	Defines a repeated loop – loop is executed as long as condition associated with **while** remains true

Data storage class

Table A6.3 C keywords associated with data storage class

Word	Summary meaning	Word	Summary meaning
auto	Variable exists only within block within which it is defined. This is the default class	**register**	Variable to be stored in a CPU register; thus, address operator (&) has no effect
extern	Declares data defined elsewhere	**static**	Declares variable which exists throughout program execution; the location of its declaration affects in what part of the program it can be referenced

Pre-processor directives

Table A6.6 Example preprocessor directives

Directive	Summary description	Directive	Summary description
#if	Used for conditionally compiling code, based on evaluation of associated expression. Must be terminated by **#endif**	**#define**	Defines string constants which are used in source code and are substituted before code line is compiled
#ifdef	Similar to **#if**, but tests if specified symbol has been defined. Terminated by **#endif**	**#error**	Generates user-defined error message
#ifndef	Identical to **#ifdef**, but tests if specified symbol has not been defined	**#include**	Include at this point the full text from file specified, which may contain unlimited C code, and is then compiled with the source program
#else	Used with **#if** to provide alternative section for compilation	**#line**	Allows user to specify line number within code
#elif	Used within an **#if** section to test a new expression	**#pragma**	Allows further directive-like information to be sent to the preprocessor, generally compiler specific
#endif	Terminates an **#if** section.	**#undef**	Reverses the action of **#define**, on string constant specified

Loop statements

- While loop:

while (condition) { statements }

while condition is true, execute statements

if there is only one statement, we can lose the {}

Example: while (1) ; // loop forever

Decision statements

- **If**

```
if (condition1)
    {statements1}
  else if (condition2)
    {statements2}
  …
  else
    {statements}
```

Arrays in C

- Useful for storing data

type arr_name[dimension]

char temp_array[256]

Array elements are stored in adjacent locations in memory.

temp_array[0]
temp_array[1]
temp_array[2]
temp_array[3]
...
temp_array[253]
temp_array[254]
temp_array[255]

Keil uvision

•Keil is an integrated development environment for a wide range of microcontrollers including 8051, 251, ARM7, and C16x/ST10 microcontrollers.

•This software includes compiler, assembler, linker, debugger, simulator etc.

This software is also easy to use and learn.

The software for 8051 is used by professional embedded system developers and beginners both.

Depending on the microcontroller, you are using you can go for the right software.

Embedded C Programming with Keil

The embedded C is the most popular programming language in the software field for developing electronic gadgets. Each processor is associated with embedded software. Embedded C programming plays a major role in performing specific functions by the processor. In our day-to-day life we frequently use many electronic devices such as washing machine, mobile phone, digital camera and so on will work based on microcontrollers that are programmed by embedded C. The C code written is more reliable, portable, and scalable; and in fact, much easier to understand. The first and foremost tool is the embedded software that decides operation of an embedded system. Embedded C programming language is most frequently used for programming the microcontrollers.

Steps For Embedded C

Step1: Comments

Step2: Preprocessor directives

Step3: Port configuration

Step4: Global variables

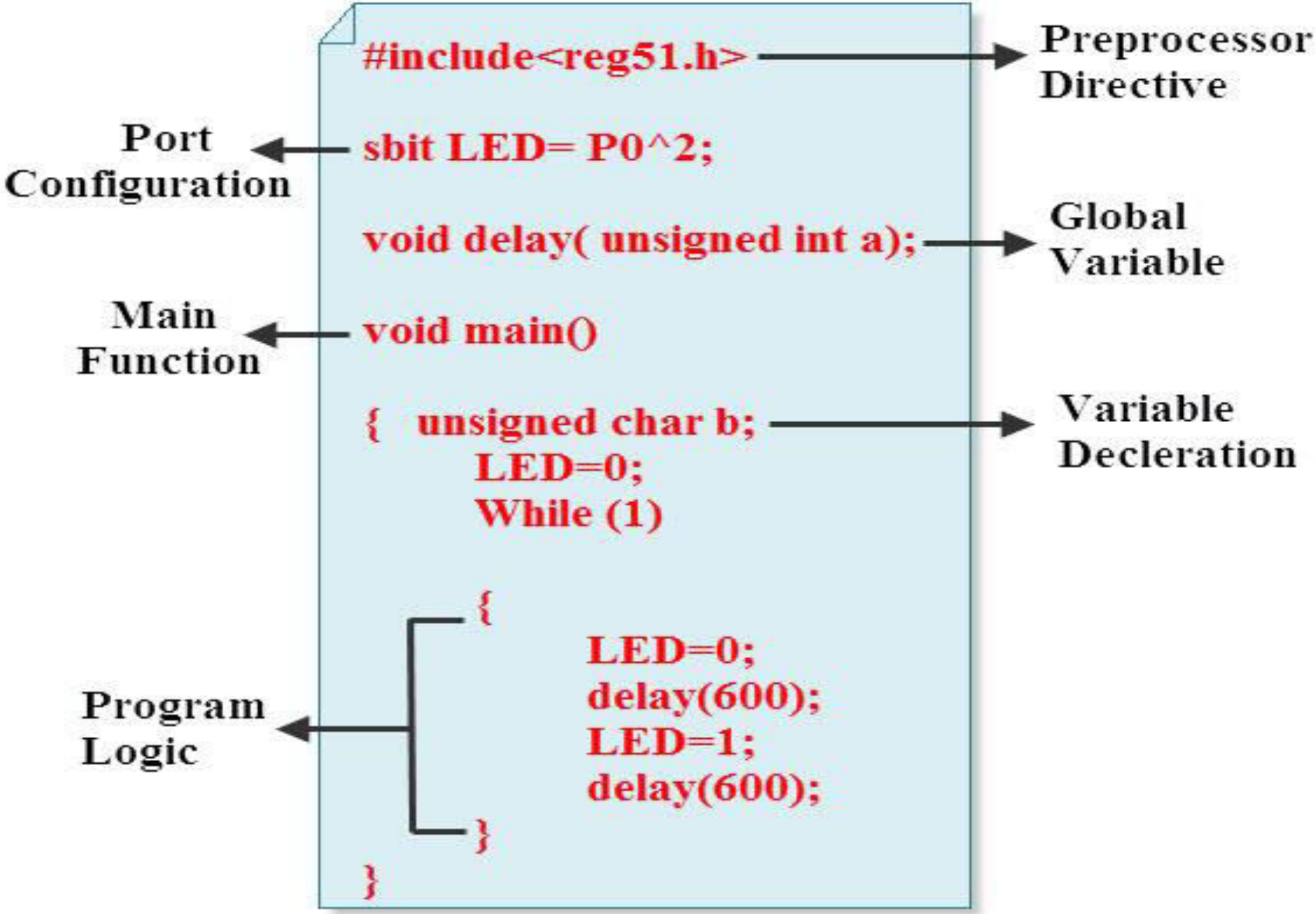
Step5: main() function or core function

{

Step6: variable declaration

Step7: Program Logic

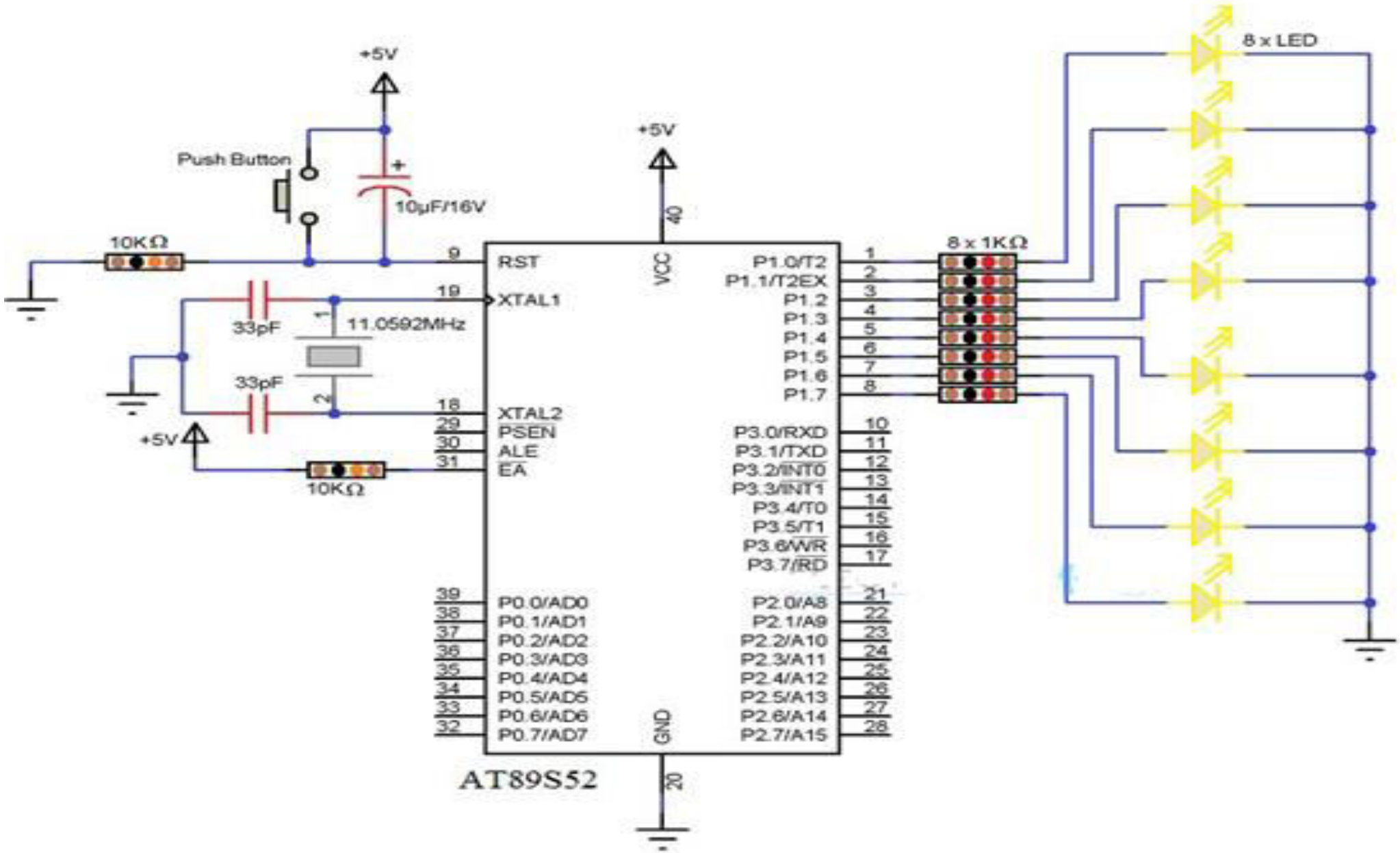
}


#include<reg51.h>
Preprocessor Directive
Port Configuration
sbit LED= P0^2;
void delay(unsigned int a);
Global Variable
Main Function
void main()
{ unsigned char b;
Variable Decleration
LED=0;
While (1)
{
LED=0;
delay(600);
LED=1;
delay(600);
Program Logic
}
}

A sample C program

Lets write a simple code to Blink LED on Port1.

```
#include <reg51.h>
void delay(void);

void main(void)
{
    while(1)
    {
        P1 = 0xFF; // Turn ON all LED's connected to Port1
        delay();
        P1 = 0x00; // Turn OFF all LED's connected to Port1
        delay();
    }
}
void delay(void)
{
    int i,j;
    for(i=0;i<0xff;i++)
        for(j=0;j<0xff;j++);
}
```


+5V
Push Button
10µF/16V
10KΩ
9
RST
19
XTAL1
33pF
11.0592MHz
33pF
18
XTAL2
29
PSEN
30
ALE
31
EA
+5V
10KΩ
+5V
40
VCC
GND
20
AT89S52
P1.0/T2
P1.1/T2EX
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
8 x 1KΩ
8 x LED

Introduction to 8051 Microcontroller Instruction Set

Writing a Program for any Microcontroller consists of giving commands to the Microcontroller in a particular order in which they must be executed in order to perform a specific task. The commands to the Microcontroller are known as a Microcontroller's **Instruction Set.**

Just as our sentences are made of words, a Microcontroller's (for that matter, any computer) program is made of Instructions. Instructions written in a program tell the Microcontroller which operation to carry out.

An Instruction Set is unique to a family of computers. This lesson introduces the 8051 Microcontroller Instruction Set also called as the MCS-51 Instruction Set.

As the 8051 family of Microcontrollers are 8-bit processors, the 8051 Microcontroller Instruction Set is optimized for 8-bit control applications. As a typical 8-bit processor, the 8051 Microcontroller instructions have **8-bit** Opcodes. As a result, the 8051 Microcontroller instruction set can have up to $2^8 = 256$ Instructions.

8051 Microcontroller Instruction set

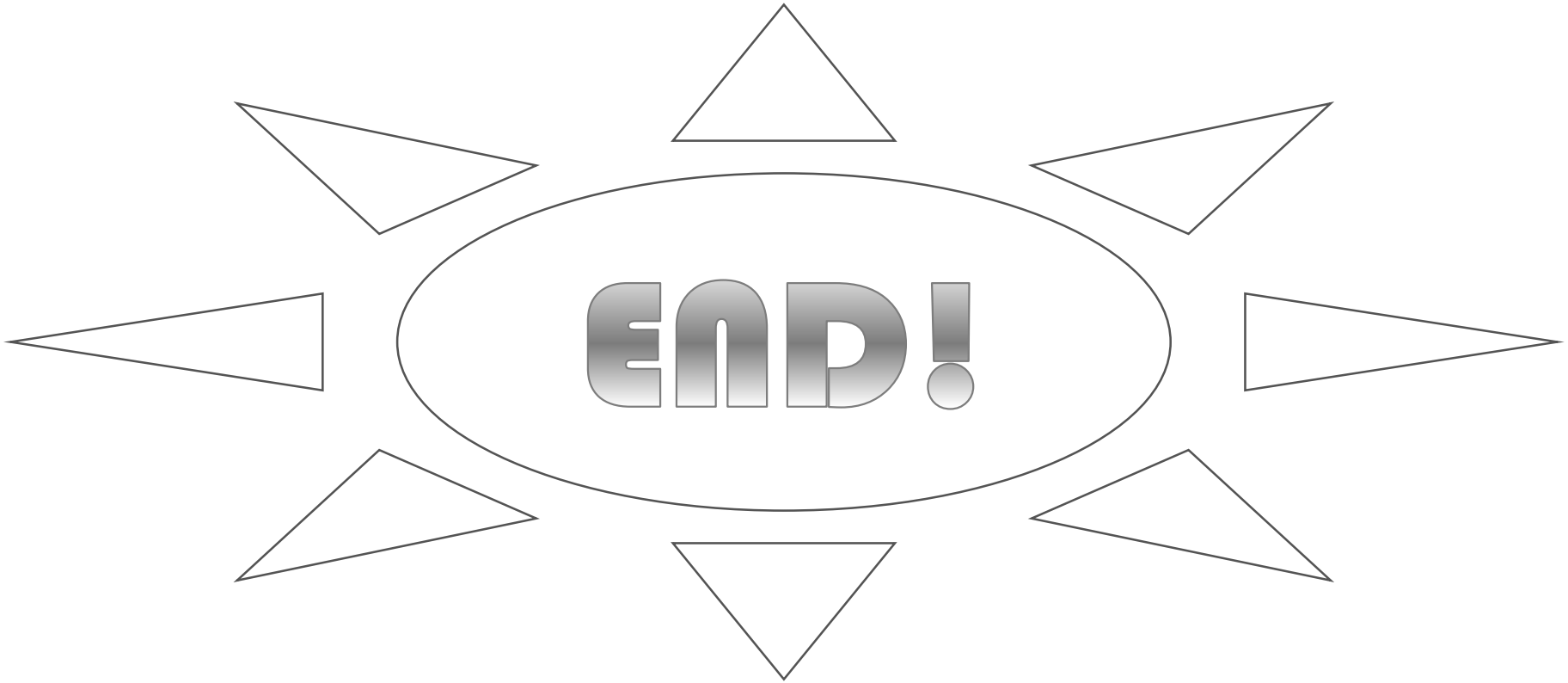
- **Instruction set**
  - Has more than 110 instructions
  - Can be grouped into 5 categories
  - These are
    - Arithmetic instructions
    - Logical instructions
    - Data transfer instructions
    - Bit manipulation instructions
    - Program branching and control

8051 INSTRUCTION SET (CONTD.)

Arithmetic	logical	Data transfer	Bit manipulation	Program control
ADD, ADDC, SUBB, INC, DEC, MUL, DIV, DA	ANL,ORL, XRL,CLR,CPL,RL,RLC,RR,RRC,SWAP	MOV, MOVX, MOVC, PUSH, POP, XCH, XCHD	CLR, SETB, CPL, ANL, ORL, MOV, JC, JNC, JB, JNB, JBC	ACALL, LCALL, RET, RETI, AJMP, LJMP, SJMP, JMP, JZ, JNZ, CJNE, DJNC, NOP

ADDRESSING MODES

- Immediate addressing
  - MOV A, #09h
  - MOV 52H, #3Ah
- Direct addressing
  - MOV A,03h
  - MOV 24h, A
  - MOV 7Ch, 0Fh
  - PUSH 49h
  - POP 22h
- Register addressing
  - MOV A, R3
  - MOV R1,A
- Indirect addressing
  - Can be performed in four different ways
    - Using R0 and R1 of each bank
    - Using DPTR and Acc
    - Using PC and ACC
    - Using XCHD instruction


END!

CHAPTER-4A

INPUT-OUTPUT INTERFACING

TECHNICAL AND VOCATIONAL TRAINING INSTITUTE
Department of Electrical electronics technology

Fundamentals of microprocessors and microcontroller
EETe 3032

By Zemenu T. /2022

Outlines

1.Basic Input/output Operations

2. I/O port detailed structure

3.Simple Interrupt, timers and counter

4. Identifying and appreciating I/O Ports

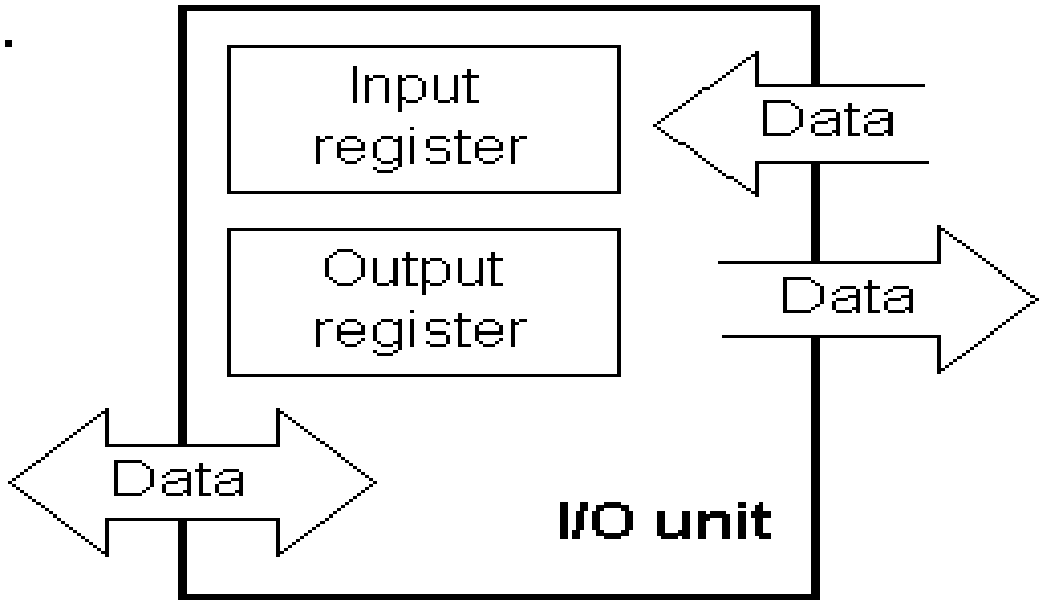
5.Interfacing the input/output

6. DAC and ADC for interfacing purpose

7.Modeling with Proteus Software

Port structure and operation

Those locations we have just added are called "**ports**". There are several types of ports: input, output or bidirectional ports. When working with ports, first of all it is necessary to choose which port we need to work with, and then to send data to, or take it from the port.



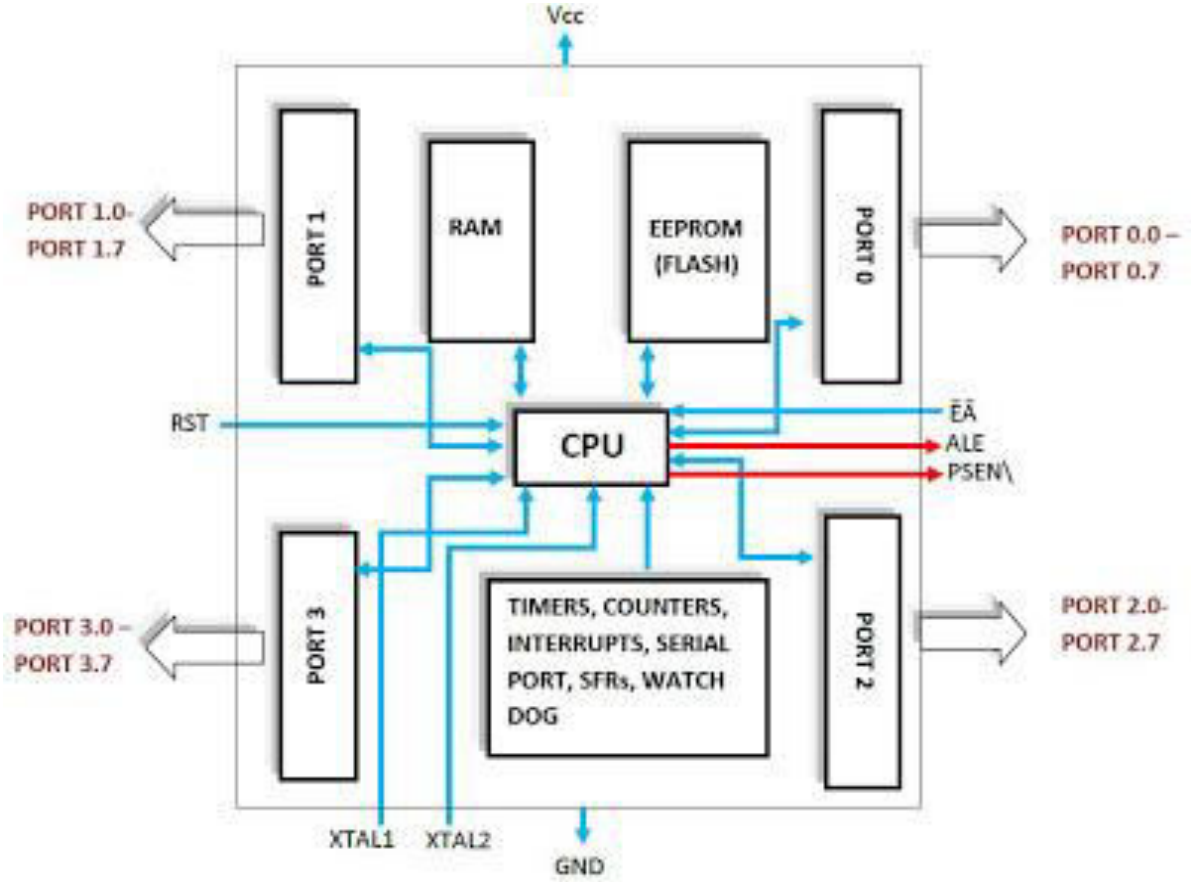
Example of a simplified input-output unit that provides communication with external world

When working with it the port acts like a **memory location**. Something is simply being **written into or read from it**, and it could be noticed on the pins of the microcontroller. All four ports in the 8051 are **bidirectional**. Each consists of a **latch** (Special Function Registers P0 through P3), an **output driver**, and an **input buffer**. The output drivers of port 0 and 2, and the input buffers of port 0, are used in accesses to external memory.

8051 INPUT OUTPUT PORTS

The **AT89C51** from ATMEL is the most commonly used variation of the 8051 microcontroller because of the built-in Flash which makes programming / burning very easy. The 8051 has a total of 4 ports for input / output operations which means you can transfer data in or out of the microcontroller through these ports. Let's have a brief introduction to these ports.

8051 INPUT OUTPUT PORTS (Contd)



This is a basic block diagram of the internal architecture including the 4 ports. They are named as **P0, P1, P2, and P3**. Later on we will see that we can refer to them by using these names while programming for the 8051. Each port is 8-bit wide thus capable of handling 8-bit data at a time.

DUAL NATURE

Some of the ports have dual nature / purpose which means that they can perform some secondary function designated to them. If you are using them for data transfer then you simply forget about the dual function as you are concerned only with the data that appears on the port as a whole (i.e. 1 byte) but if you are using **interrupts, timers** (to be discussed later) then you have to consider the dual nature of each pin accordingly. Here is the description of each pin

PORT 0	
PIN	**DUAL FUNCTION**
0.0	AD0
0.1	AD1
0.2	AD2
0.3	AD3
0.4	AD4
0.5	AD5
0.6	AD6
0.7	AD7

PORT 1	
PIN	**DUAL FUNCTION**
1.0	None
1.1	None
1.2	None
1.3	None
1.4	None
1.5	None
1.6	None
1.7	None

PORT 2	
PIN	**DUAL FUNCTION**
2.0	A8
2.1	A9
2.2	A10
2.3	A11
2.4	A12
2.5	A13
2.6	A14
2.7	A15

PORT 3	
PIN	**DUAL FUNCTION**
3.0	RxD
3.1	TxD
3.2	INT0
3.3	INT1
3.4	T0
3.5	T1
3.6	WR
3.7	RD

I/O ports and circuits

PORT 0 Pin Structure

- Port0 occupies a total of 8 pins (pins 32-39).
- It can be used for input or output.
- To use the pins of port 0 as both input and output ports, each pin must be connected externally to 10k-ohm pull up resistor.
- This is due to fact that p0 is an open drain with external pull up resistors connected upon the reset. Port-0 Pin Structure is shown in fig below.

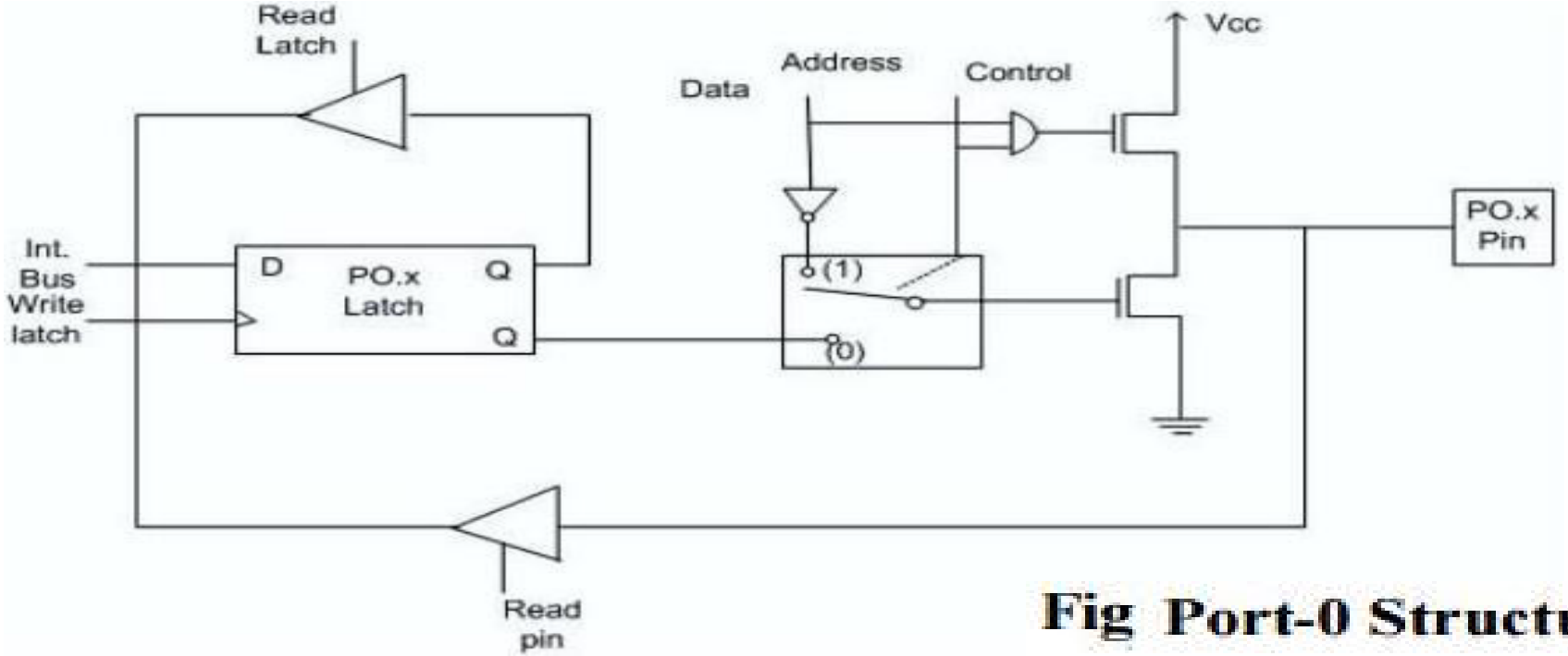
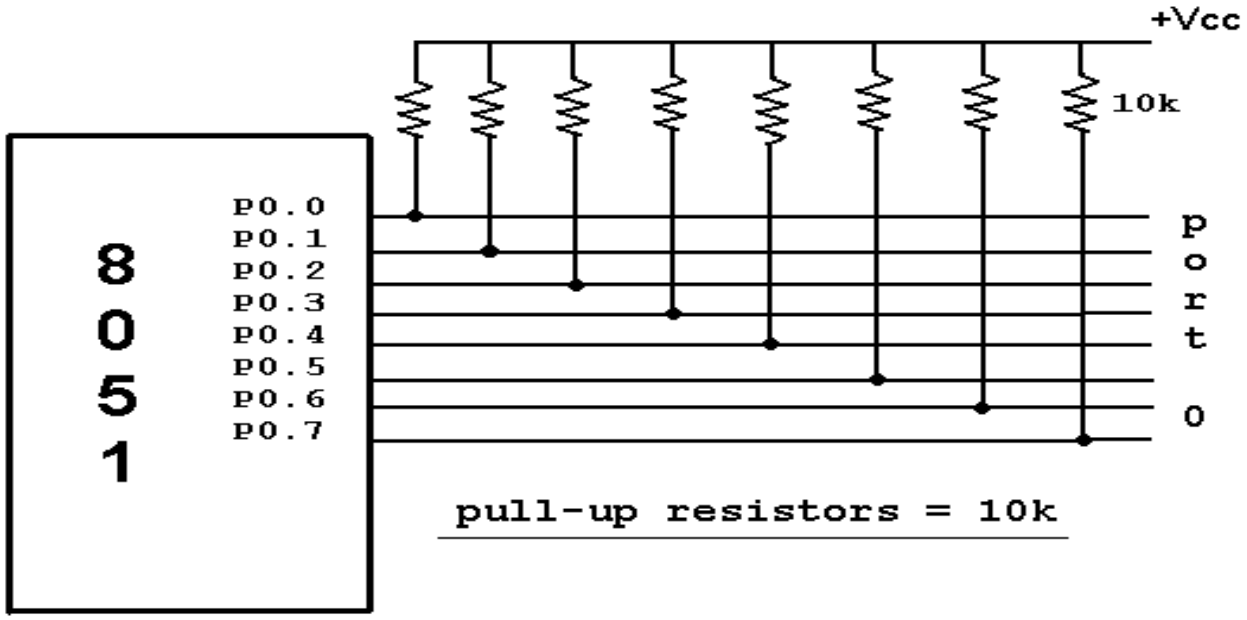


Fig Port-0 Structure

PORT 0 Pin Structure (Contd.)

Port-0 can be configured as a normal bidirectional **I/O** port or it can be used for **address/data interfacing** for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port.



Port0 with pull up resistor

Dual role of port 0

- Port0 is also designated as AD0-AD7, allowing it to be used for both address and data .
- When connecting 8051 to an external memory, port0 provides both address and data
- The 8051 multiplexes address and data through port0 to save pins.
- When ALE=0, it provides data D0-D7 but when ALE=1 it has address A0-A7.

PORT 1 Pin Structure

Port-1 has 8 pins (P1.0 - P1.7) .The structure of a port-1 pin is shown in fig. below

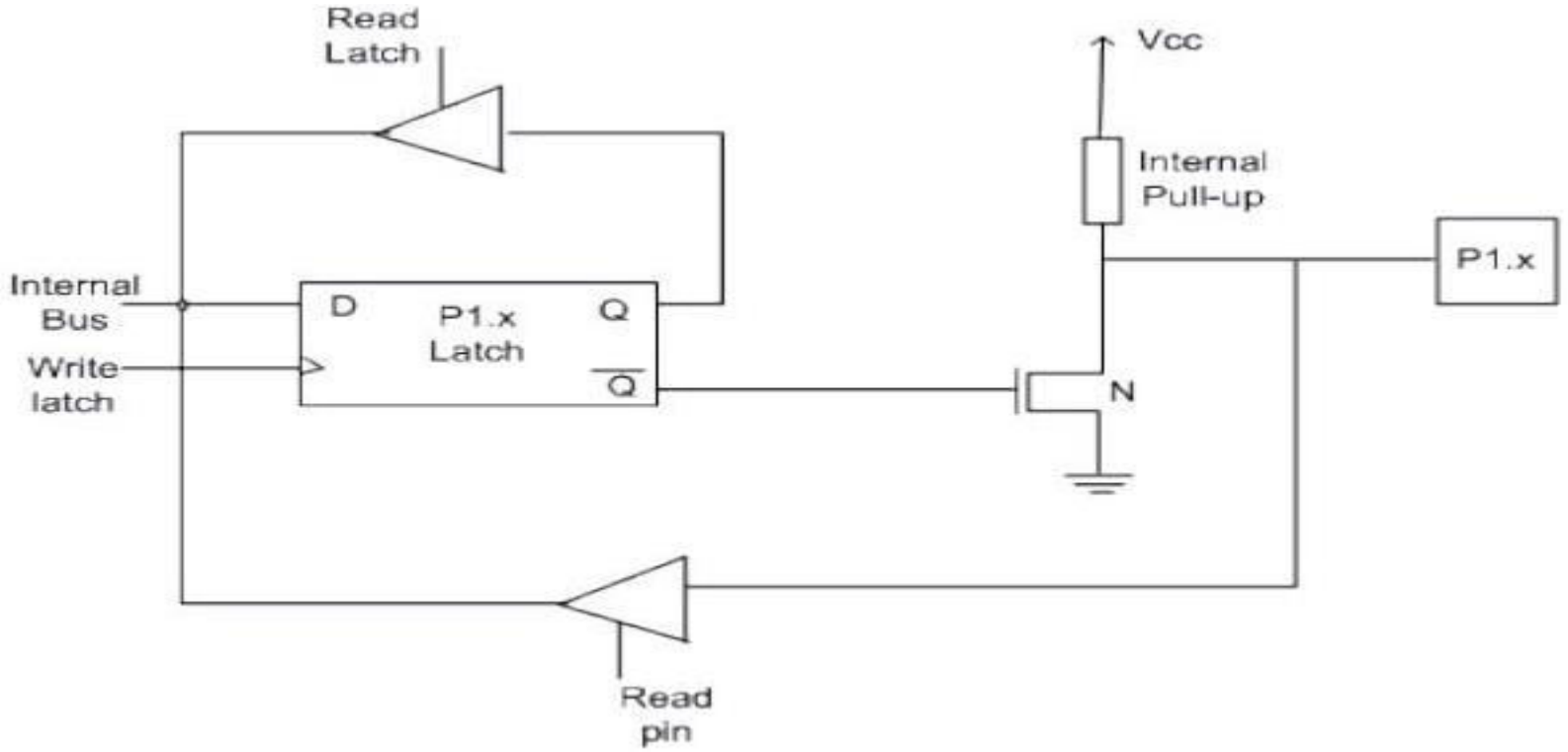


Fig Port 1 Structure

PORT 1 Pin Structure (Contd.)

Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing. When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input port, '1' has to be written to the latch. In this input mode **when '1'** is written to the pin by the external device then it read fine. But **when '0'** is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

PORT 2 Pin Structure

Port-2 has 8-pins (P2.0-P2.7). The structure of a port-2 pin is shown in fig below.

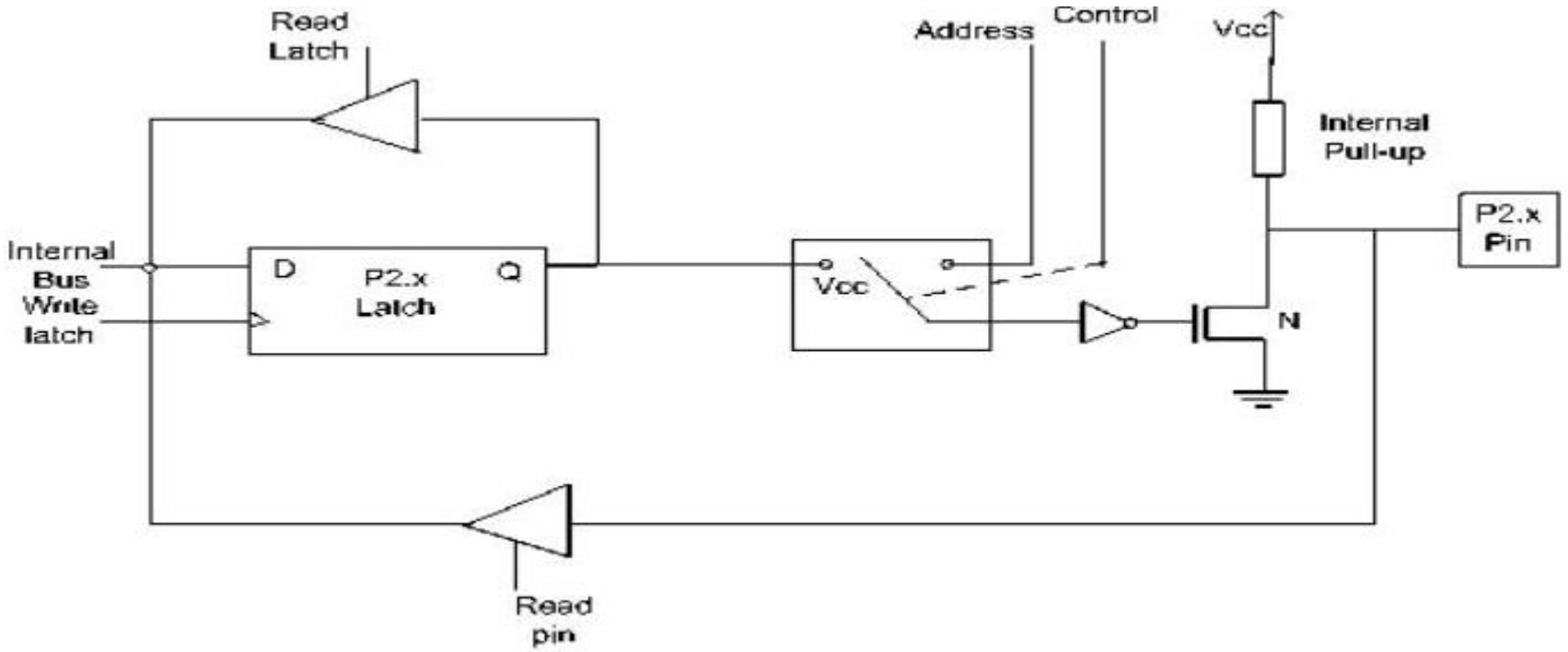


Fig Port 2 Structure

Dual role of port2

Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1.

Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.

•Port2 must be used along with p0 to provide the 16 bit address for the external memory.

•Port2 is designated as **A8-A15**,indicating its dual function

PORT 3 Pin Structure

Port-3 has 8 pin (P3.0-P3.7) . Port-3 pins have alternate functions. The structure of a port-3 pin is shown in fig below.

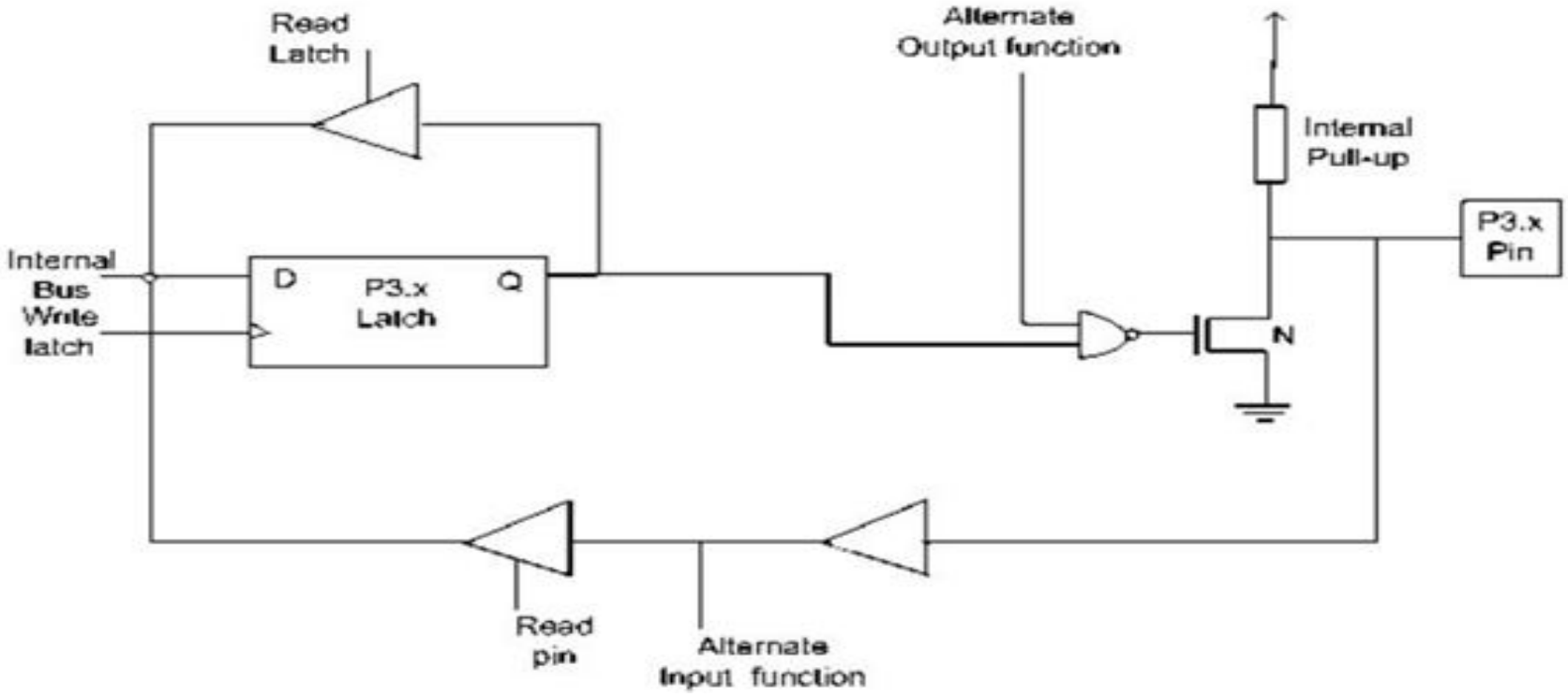


Fig Port 3 Structure

PORT 3 Pin Structure (Contd.)

Each pin of Port-3 can be individually programmed for I/O operation or for alternate function. The alternate function can be activated only if the corresponding latch has been written to '1'. To use the port as input port, '1' should be written to the latch. This port also has internal pull-up and limited current driving capability.

Alternate functions of Port-3 pins are –

BIT	NAME	BIT ADDRESS	ALTERNATE FUNCTION
P3.0	RXD	B0H	Receive data for serial port
P3.1	TXD	B1H	Transmit data for serial port
P3.2	$\overline{\text{INT0}}$	B2H	External interrupt 0
P3.3	$\overline{\text{INT1}}$	B3H	External interrupt 1
P3.4	T0	B4H	Timer/counter 0 external input
P3.5	T1	B5H	Timer/counter 1 external input
P3.6	$\overline{\text{WR}}$	B6H	External data memory write strobe
P3.7	$\overline{\text{RD}}$	B7H	External data memory read strobe

INTERRUPT

An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service. Every interrupt has a program associated with it called the ISR, or interrupt service routine. The 8051 has 6 interrupts, 5 of which are user-accessible. The interrupts- are for reset: two for the timers, two for external hardware interrupts, and a serial communication interrupt. The 8052 has an additional interrupt for Timer 2.

The 8051 can be programmed to enable or disable an interrupt, and the interrupt priority can be altered.

INTERRUPT(Cont'd)

An interrupt is an external or internal event to get the CPU's attention. Once the controller detects the interrupt, it suspends the current job and executes a **special service routine** known as Interrupt Service Routine(ISR). In writing applications for a typical computer, the operating system provides system calls for setting a function, declared in the standard manner, as the handler for an interrupt. However, in writing code for an 8051 without an operating system, such a system would note be possible using solely C code. To eliminate this problem, the keil compiler implements a function extension that explicitly declares a function as an interrupt handler. The **extension is interrupt**, and it must be followed by an integer specifying which interrupt the handler is for.

INTERRUPT (Cont'd)

For example:

/*This is a function that will be called whenever a serial*/

/*interrupt occurs. Note that before this will work, interrupts*/

/*must be enabled.*/

```
Void serial_int (void) interrupt
{
   ....
}
```

In the example, a function called **serial_int** is set as the handler for interrupt, which is the serial port interrupt.

INTERRUPT (Cont'd)

The **five standard** interrupts for the 8051 are shown below
:

Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins.

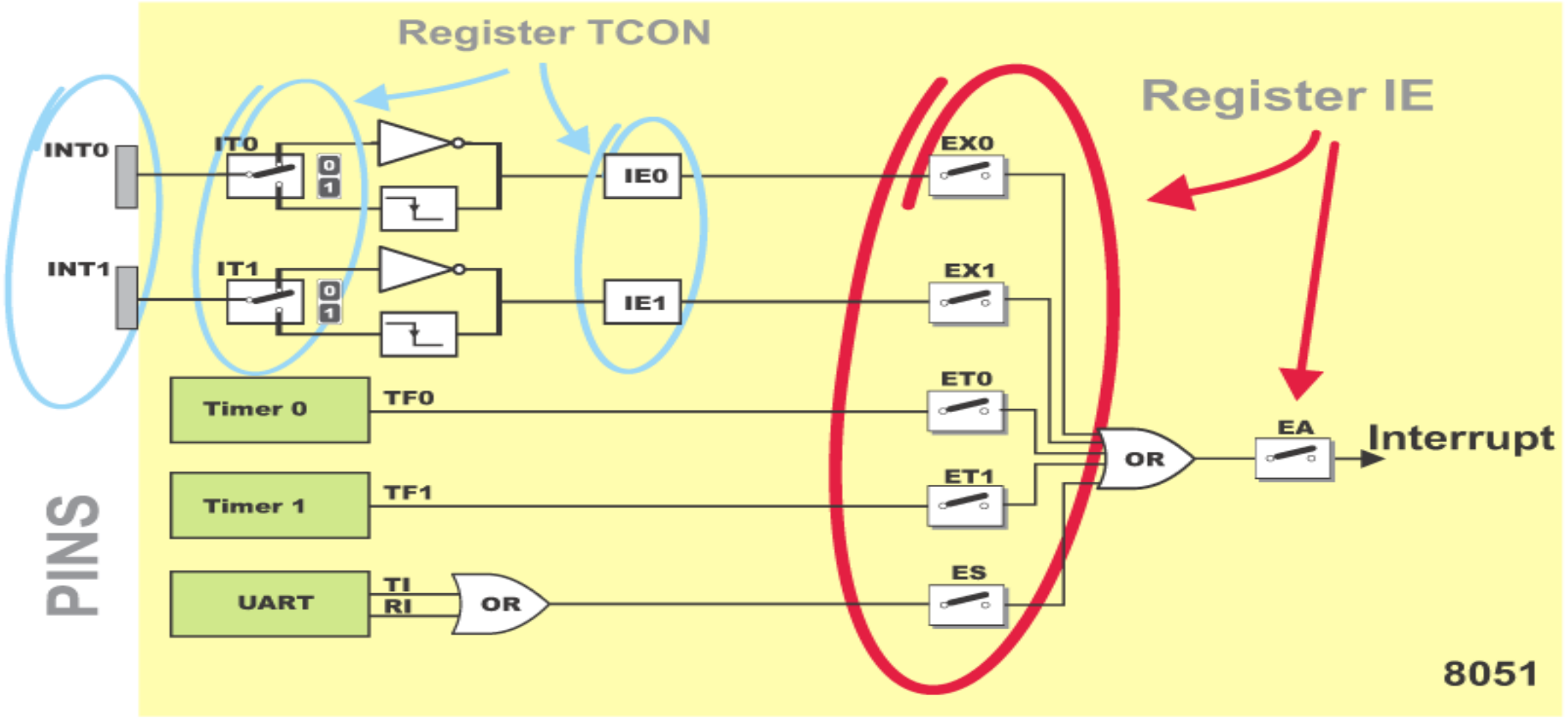
Interrupt	Flag	Vector Address
Reset	_	0000
External 0	IE0	0003h
Timer 0	TF0	000Bh
External 1	IE1	0013h
Timer 1	TF1	001Bh
Serial	RI/TI	0023h

INTERRUPT (Cont'd)

There are five interrupt sources for the 8051, which means that they can recognize **5 different events** that can interrupt regular program execution. Each interrupt can be enabled or disabled by setting bits of the IE register. Likewise, the whole interrupt system can be disabled by clearing the **EA** bit of the same register. Refer to figure below.

Now, it is necessary to explain a few details referring to external interrupts - INT0 and INT1. If the IT0 and IT1 bits of the TCON register are set, an interrupt will be generated on high to low transition, i.e. on the falling pulse edge (only in that moment). If these bits are cleared, an interrupt will be continuously executed as far as the pins are held low.

INTERRUPT (Cont'd)



IE Register (Interrupt Enable)

	0	X	0	0	0	0	0	0	Value after Reset
IE	EA		ET2	ES	ET1	EX1	ET0	EX0	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

INTERRUPT (Cont'd)

Bit	Symbol	Function
IE.7	EA	Disables **all interrupts**. If EA=0, no interrupt will be acknowledged. If EA=1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
IE.6	-	Reserved
IE.5	-	Reserved
IE.4	ES	Enables or disables the **serial port** interrupt. If ES=0, the serial port interrupt is disabled.
IE.3	ET1	Enables or disables the **Timer 1** overflow interrupt . If ET1=0, the Timer 1 interrupt is disabled.
IE.2	EX1	Enables or disables **External interrupt 1**. If EX1=0, External interrupt 1 is disabled.
IE.1	ET0	Enables or disables the **Timer 0** overflow interrupt . If ET0=0, the Timer 0 interrupt is disabled.
IE.0	EX0	Enables or disables **External interrupt 0**. If EX0=0, External interrupt 0 is disabled.

INTERRUPT (Cont'd)

The External Interrupts INT0 and INT1 can each be either level-activated or transition - activated , depending on bits IT0 and IT1 in Register TCON. The flags that actually generate these interrupts are bits IE0 and IE1, in TCON, When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to only if the interrupt was transition-activated. If the interrupt was level-activated, then the external requesting source is what controls the request flag, rather than the on-chip hardware.

The **Timer 0 and Timer 1 Interrupts** are generated by TF0 TF1, which are set by a rollover in their respective Timer/Counter registers (except see Timer 0 in mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

INTERRUPT (Cont'd)

The **serial port interrupt** is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt, and the bit will have to be cleared in software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be canceled in software. Each of these interrupt sources can be individually enabled by setting or clearing a bit in Special Function Register IE. IE also contains a **global disable bit**, EA, which disables all interrupts at once.

INTERRUPT (Cont'd)

In the 8051, interrupts have two possible priorities: high and low. If, during the processing of an interrupt, another interrupt of the same priority occurs, the processor will continue processing the first interrupt. The second interrupt will only be processed after the first has finished. However, if an interrupt of a higher priority arrives, the first (low priority) interrupt will it self be interrupted, and not resume until the higher priority interrupt has finished. Because of this, all interrupts of the same priority may use the same register bank. The using extension should be used when quick execution time is of high importance, or when other functions are called from the interrupt handler, as it would otherwise push all of the registers on to the stack prior to calling the function, incurring more time penalties.

8051 timer/counter

The 8051 has 2 timers/counters: timer/counter 0 and timer/counter 1. They can be used as

1. **Timer**
   - The **timer** is used as a time delay generator.
   - The clock source is the **internal** crystal frequency of the 8051.
2. **counter**.
   - Which counts number of events.
   - An "event" is any external stimulus that provides a 1-to-0 transition to a pin on the 8051 IC.
   - These clock pulses can represent the number of people passing through an entrance, or the number of wheel rotations, or any other event that can be converted to pulses.

8051 timer/counter (Cont'd)

The 8051 has two 16-bit Timer/Counters: Timer 0 and Timer 1. Both can be configured to operate either as timers or event counters.

- **In the "Timer" function**, the register is incremented every machine cycle. Thus, one can think of it as counting machine cycles. Since a **machine cycle** consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.
- **In the counter function**, the register is incremented in response to a **1-to-0** transition at its corresponding external input pin, (T0 or T1). It requires **2 machine cycles** to detect a high to low transition. Hence maximum count rate is 1/24 th of oscillator frequency.

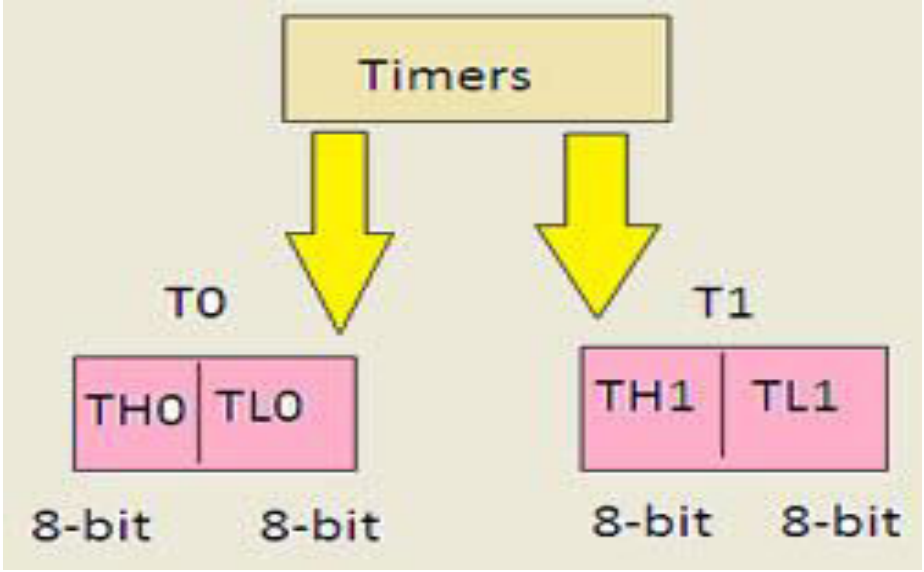
Difference between a Timer and a Counter

The points that differentiate a timer from a counter are as follows –

Timer	Counter
The register incremented for every machine cycle.	The register is incremented considering 1 to 0 transition at its corresponding to an external input pin (T0, T1).
Maximum count rate is 1/12 of the oscillator frequency.	Maximum count rate is 1/24 of the oscillator frequency.
A timer uses the frequency of the internal clock, and generates delay.	A counter uses an external signal to count pulses.

8051 timer/counter (Cont'd)

There are two 16-bit timers and counters in 8051 microcontroller: timer 0 and timer 1. Both timers consist of 16-bit register in which the lower byte is stored in TL and the higher byte is stored in TH. Timer can be used as a counter as well as for timing operation that depends on the source of clock pulses to counters.



8051 timer/counter (Cont'd)

Counters and Timers in 8051 microcontroller contain two special function registers: TMOD (Timer Mode Register) and TCON (Timer Control Register), which are used for activating and configuring timers and counters.

Timer Mode Control (TMOD): TMOD is an 8-bit register used for selecting timer or counter and mode of timers. Lower 4-bits are used for control operation of timer 0 or counter 0, and remaining 4-bits are used for control operation of timer1 or counter1.This register is present in SFR register, the address for SFR register is 89th.

8051 timer/counter (Cont'd)

Timer Mode Control (TMOD)

Gate: If the gate bit is set to '0', then we can start and stop the "software" timer in the same way. If the gate is set to '1', then we can perform hardware timer.

C/T: If the C/T bit is '1', then it is acting as a counter mode, and similarly when set C/T bit is '0'; it is acting as a timer mode. **Mode select bits:** The M1 and M0 are mode select bits, which are used to select the timer operations. There are four modes to operate the timers.

Different Modes of Timers

Mode 0: This is a 13-bit mode that means the timer operation completes with “8192” pulses.

Mode 1: This is a16-bit mode, which means the timer operation completes with maximum clock pulses that “65535”.

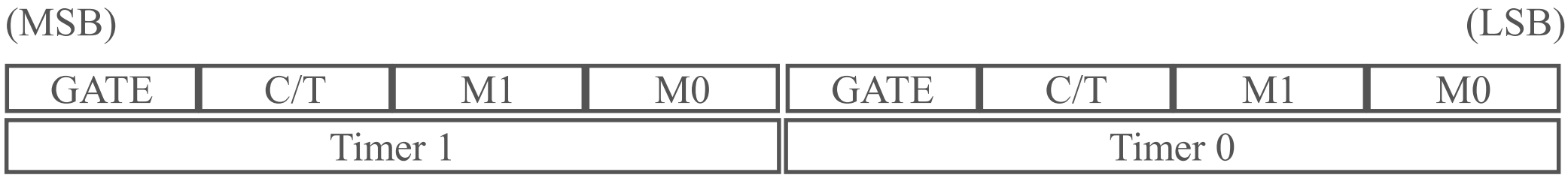
Mode 2: This mode is an 8-bit auto reload mode, which means the timer operation completes with only “256” clock pulses.

Mode 3: This mode is a split-timer mode, which means the loading values in T0 and automatically starts the T1.

M0	M1	Mode	Timer Pulses
0	0	M0	13-bit-2^13-8192
0	1	M1	16-bit-2^16-65535 pulses
1	0	M2	8-bit-autoreload mode-2^8= 256 pulses
1	1	M3	Split mode(load the values in T0 automatically start the T1

Mode Selection Bits

TIMER MODE REGISTER (TMOD)



GATE:-This is an OR Gate enabled bit which controls the effect of INT1/0 on START/STOP of Timer. It is set to one ('1') by the program to enable the interrupt to start/stop the timer. If TR1/0 in TCON is set and signal on INT1/0 pin is high then the timer starts counting using either internal clock (timer mode) or external pulses (counter mode).

C/T:- This bit in the TMOD register is used to decide whether the timer is used as a delay generation or an event counter.

- C/T = 0 : timer
- C/T = 1 : counter

M1 - Mode bit 1

M0 - Mode bit 0

TIMER MODE REGISTER (TMOD) (Cont'd)

GATE

- Every timer has a mean of starting and stopping.
  - GATE=0
    - **Internal** control
    - The start and stop of the timer are controlled by way of **software.**
    - Set/clear the TR for start/stop timer.
  - GATE=1
    - **External** control
    - The hardware way of starting and stopping the timer by **software** and **an external source**.
    - Timer/counter is enabled only while the INT pin is high and the TR control pin is **set** (TR).

TIMER MODE REGISTER (TMOD) (Cont'd)

C/T (CLOCK / TIMER)

This bit in the TMOD register is used to decide whether a timer is used as a **delay generator** or an **event manager**. If C/T = 0, it is used as a timer for timer delay generation. The clock source to create the time delay is the crystal frequency of the 8051. If C/T = 0, the crystal frequency attached to the 8051 also decides the speed at which the 8051 timer ticks at a regular interval.

Timer frequency is always 1/12th of the frequency of the crystal attached to the 8051. Although various 8051 based systems have an XTAL frequency of 10 MHz to 40 MHz, we normally work with the XTAL frequency of 11.0592 MHz. It is because the baud rate for serial communication of the 8051.XTAL = 11.0592 allows the 8051 system to communicate with the PC with no errors.

TIMER MODE REGISTER (TMOD) (Cont'd)

- M0 and M1 select the timer mode for timers 0 & 1.

M1	M0	Mode	Operating Mode
0	0	**0**	**13-bit timer** mode 8-bit THx + 5-bit TLx (x= 0 or 1)
0	1	**1**	**16-bit timer** mode 8-bit THx + 8-bit TLx
1	0	**2**	**8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded into TLx each time it overflows.
1	1	**3**	Split timer mode

TIMER MODE 0

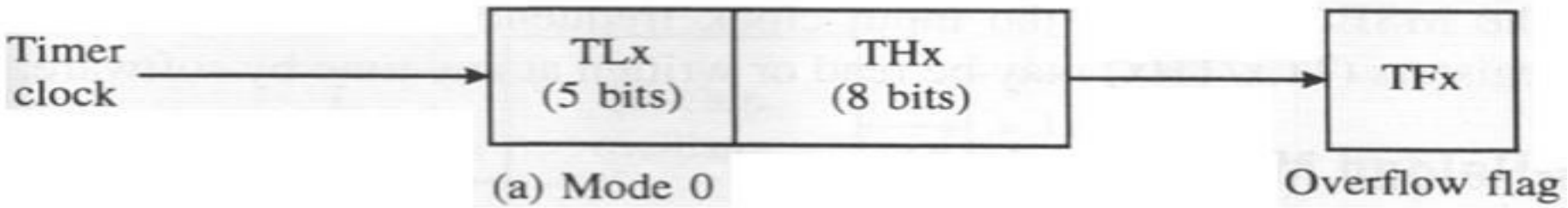
- Mode 0 is exactly like mode 1 except that it is a **13-bit** timer instead of 16-bit.
  - 8-bit TH0
  - 5-bit TL0
- The counter can hold values between 0000 to 1FFF in TH0-TL0.
  - 2^{13}-1= 2000H-1=1FFFH
- We set the initial values TH0-TL0 to count up.
- When the timer reaches its maximum of 1FFFH, it rolls over to 0000, and TF0 is raised.

TIMER MODE 0 (CONT'D)

•This mode configures timer 0 as a 13-bit timer which consists of all 8 bits of TH0 and the lower 5 bits of TL0.

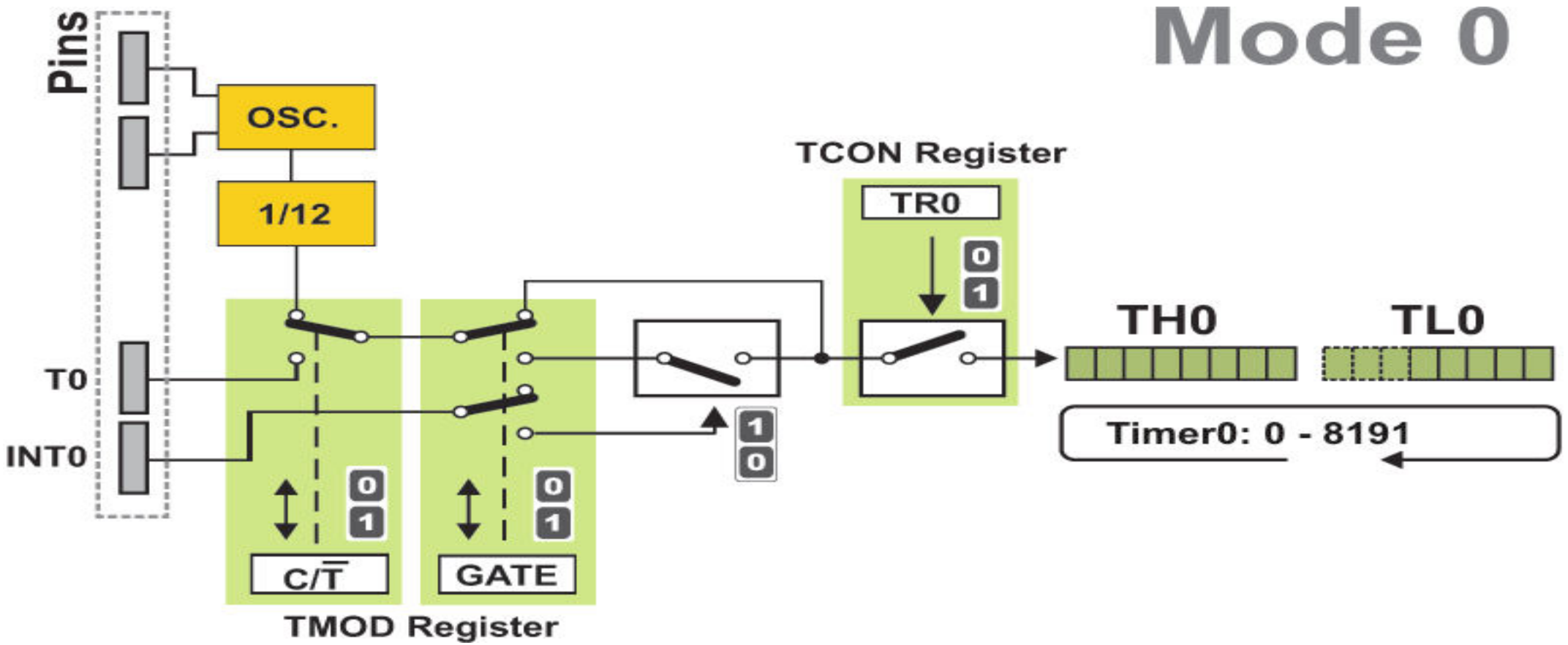
As a result, the Timer 0 uses only 13 of 16 bits

•The timer high-byte (THx) is cascaded with the five least-significant bits of the timer low-byte (TLx) to form a 13-bit timer.



(a) Mode 0

Timer 0 in mode 0 (13-bit timer) Cont..

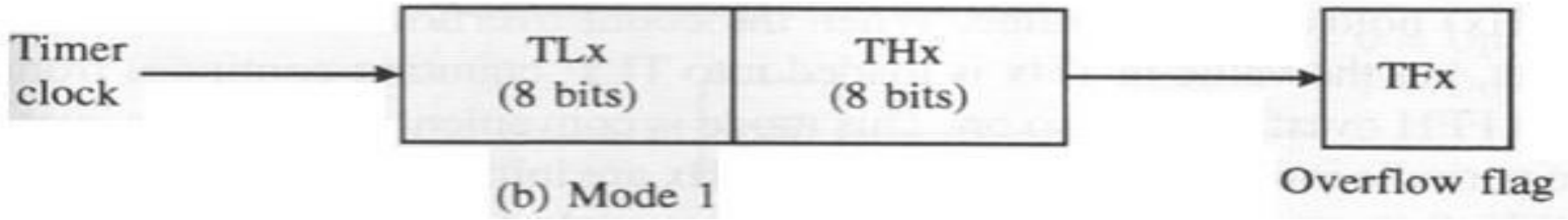


TIMER MODE 1

- In following, we all use timer 0 as an example.
- **16-bit** timer (TH0 and TL0)
- TH0-TL0 is incremented continuously when TR0 is set to 1. And the 8051 stops to increment TH0-TL0 when TR0 is cleared.
- The timer works with the internal system clock. In other words, the timer counts up each machine cycle.
- When the timer (TH0-TL0) reaches its maximum of FFFFH, it rolls over to 0000, and TF0 is raised.

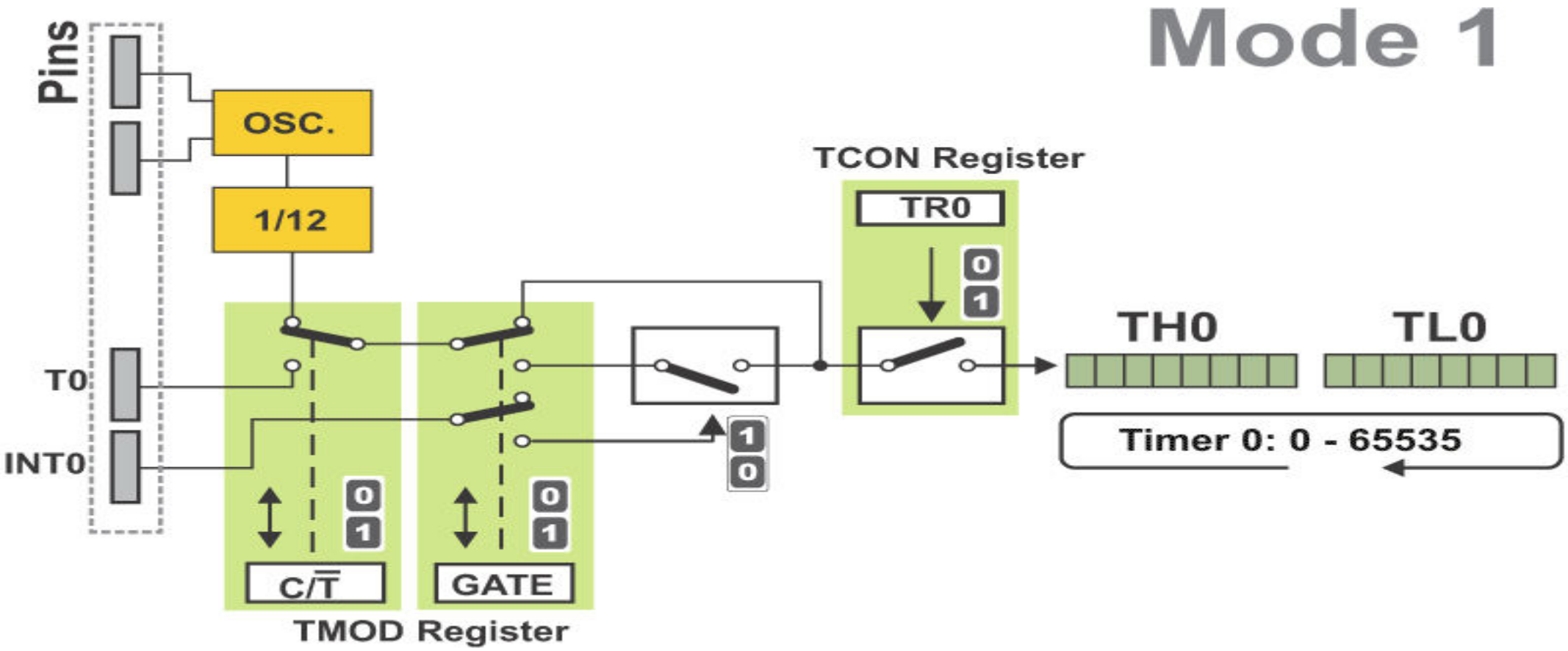
Timer Mode 1 (Cont'd)

Mode 1 configures timer 0 as a 16-bit timer. It uses all the bits of both registers TH0 and TL0. This is one of the most commonly used modes. Timer operates in the same way as in mode 0, with difference that the registers count up to 65,536 as allowable by the 16 bits.



(b) Mode 1

Timer mode 1 (16-bit timer) Cont..



Timer Mode 2

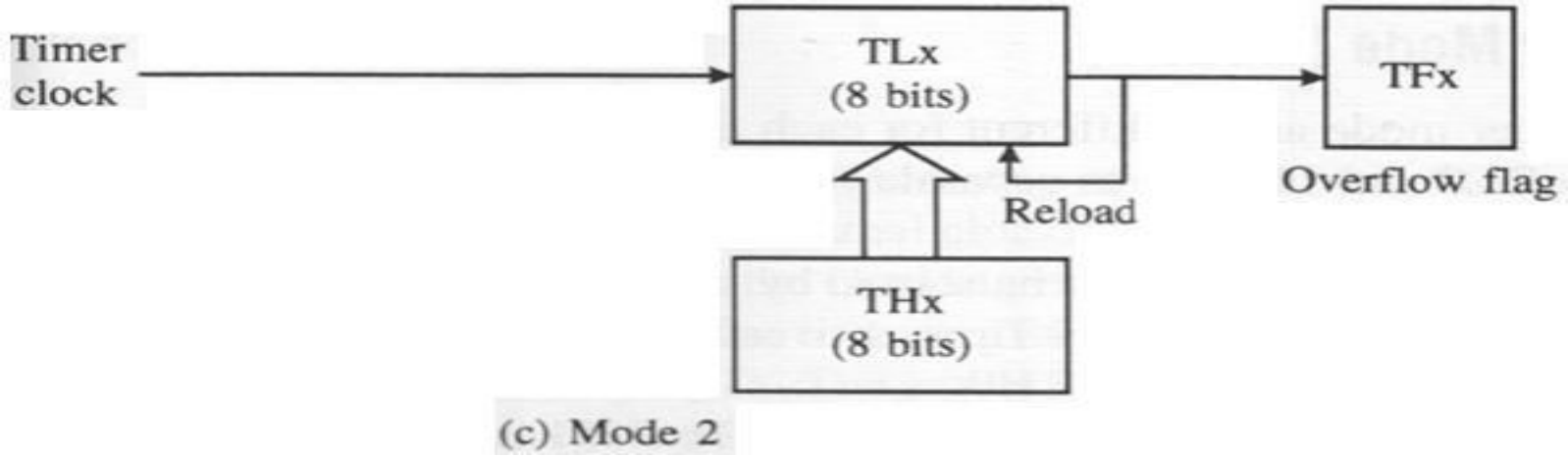
Actually, timer 0 uses only one 8-bit register for counting and never counts from 0, but from an arbitrary value (0-255) stored in another (TH0) register.

In fact, only the TL0 register operates as a timer, while another (TH0) register stores the value from which the counting starts

When the TL0 register is loaded, instead of being cleared, the contents of TH0 will be reloaded to it.

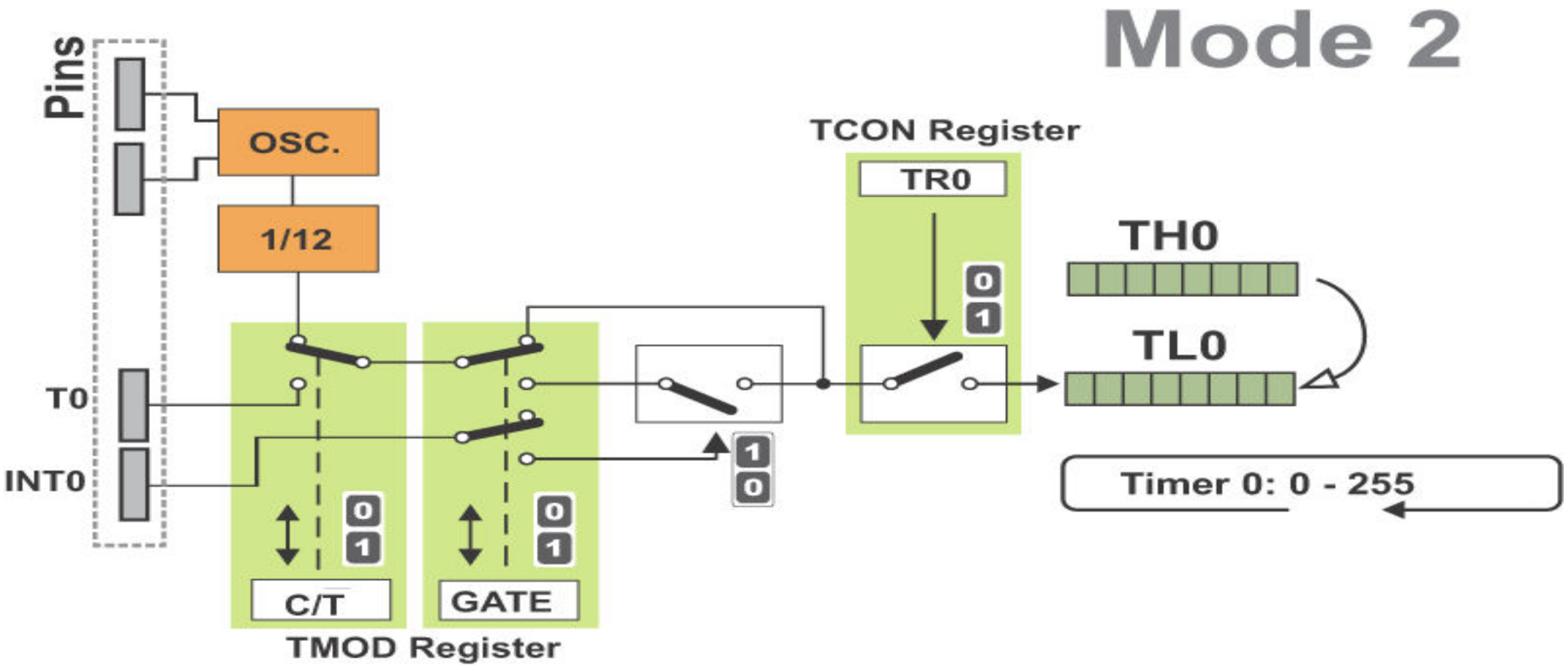
This is a 8 bit counter/timer operation. Counting is performed in TLX while THX stores a constant value. In this mode when the timer overflows i.e. TLX becomes FFH, it is fed with the value stored in THX. For example if we load THX with 50H then the timer in mode 2 will count from 50H to FFH. After that 50H is again reloaded. This mode is useful in applications like fixed time sampling.

8-bit Auto-reload Mode (Mode 2)



- 8-bit auto-reload mode.
  - It allows only values of 00 to FFH to be loaded into TH0.
- The timer low-byte (TLx) operates as an 8-bit timer while the timer high-byte (THx) holds a reload value.
- When the count overflows from FFH, not only is the timer flag set, but the value in THx is loaded into TLx.

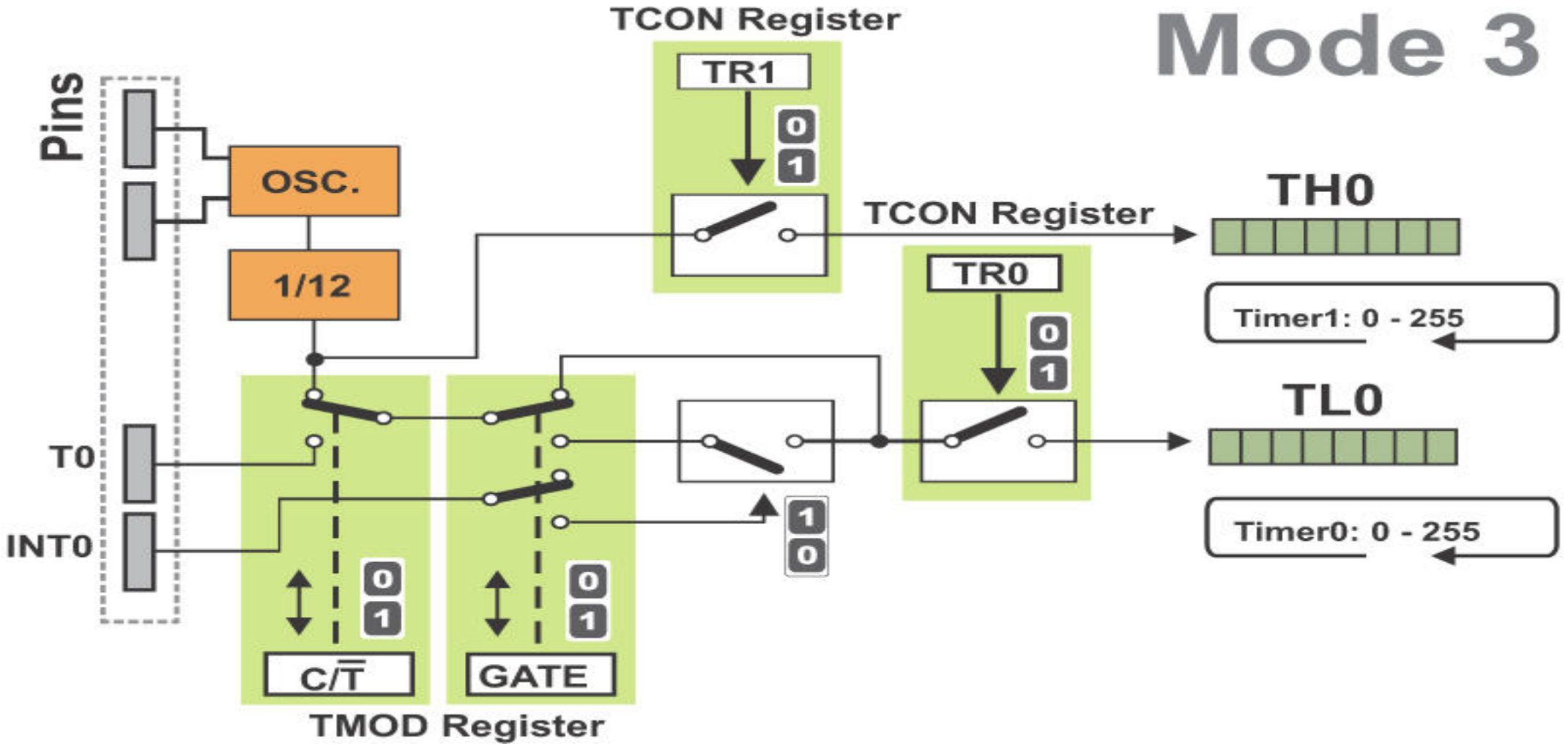
Timer 0 mode 2 (Auto-Reload Timer)



Timer Mode 3

Mode 3 configures timer 0 so that registers TL0 and TH0 operate as separate 8-bit timers. In other words, the 16-bit timer consisting of two registers TH0 and TL0 is split into two independent 8-bit timers. This mode is provided for applications requiring an additional 8-bit timer or counter. The TL0 timer turns into timer 0, while the TH0 timer turns into timer 1. All the control bits of 16-bit Timer 1 (consisting of the TH1 and TL1 register), now control the 8-bit Timer 1. Even though the 16-bit Timer 1 can still be configured to operate in any of modes (mode 1, 2 or 3), it is no longer possible to disable it as there is no control bit to do it. Thus, its operation is restricted when timer 0 is in mode 3.

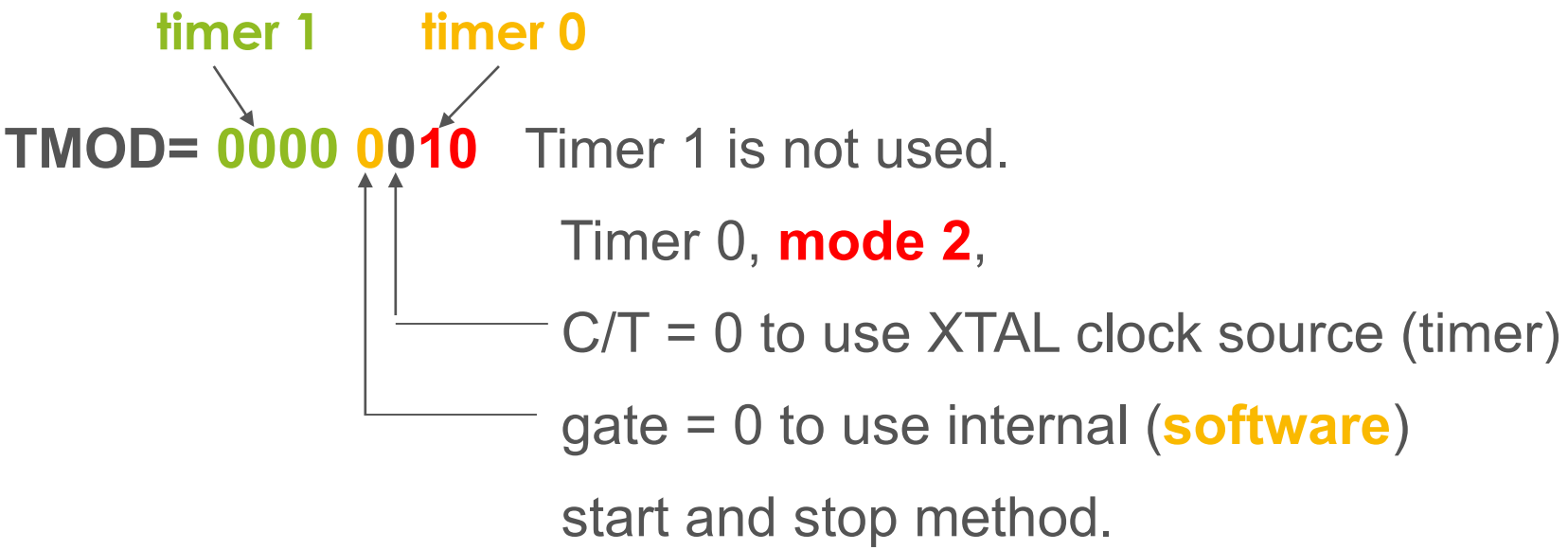
Timer 0 in Mode 3 (Split Timer)



Example

Find the value for TMOD if we want to program timer 0 in mode 2, use 8051 XTAL for the clock source, and use instructions to start and stop the timer.

Solution:



TIMER CONTROL REGISTER (TCON)

- Timer Control Register (TCON): TCON is another register used to control operations of counter and timers in microcontrollers. It is an 8-bit register where in four upper bits are responsible for timers and counters and lower bits are responsible for interrupts
- Contains status and control bits for Timer 0 and Timer 1
- The upper four bits in TCON (TCON.4-TCON.7) are used to turn the **timers** on and off (TR0, TR1), or to signal a timer overflow (TF0, TF1).
- The lower four bits in TCON (TCON.0-TCON.3) are used to detect and initiate external **interrupts**.

TIMER CONTROL REGISTER (TCON) (Cont'd)

The various bits of TCON are as follows.

- **TF1 [Timer1 over flow flag]** :- Set by hardware when timer 1 counters over flows, cleared by hardware as the processor vectors to the service routine.
- **TR1 [Timer1 run control bit]** :- Set/cleared by software to turn timer1 counter1 on/off.
- **TF0 [Timer0 over flow flag]** :- Set by hardware when timer /counters over flows, cleared by hardware as the processor vectors to the service routine.

TIMER CONTROL REGISTER (TCON) (Cont'd)

- ***TR0 [Timer0 run control bit] :-*** *Set/cleared by* software to turn timer/counter on/off.
- ***IE1 [External interrupt1 edge flag] :-*** *Set by* CPU when the external interrupt edge is detected, cleared by the CPU when the interrupt is processed.
- **IE0** : Interrupt0 edge flag. (Similar to IE1)
- ***IT1 [Interrupt 1 type control bit] :-*** Set/cleared by software to specify falling edge/low level triggered external interrupt.
- **IT0** : Interrupt0 type control bit. (Similar to IT1)

TIMER CONTROL REGISTER (TCON) (Cont'd)

TCON (timer control) register summary

BIT	SYMBOL	BIT ADDRESS	DESCRIPTION
TCON.7	TF1	8FH	Timer 1 overflow flag. Set by hardware upon overflow; cleared by software, or by hardware when processor vectors to interrupt service routine
TCON.6	TR1	8EH	Timer 1 run-control bit. Set/cleared by software to turn timer on/off
TCON.5	TF0	8DH	Timer 0 overflow flag
TCON.4	TR0	8CH	Timer 0 run-control bit
TCON.3	IE1	8BH	External interrupt 1 edge flag. Set by hardware when a falling edge is detected on INT1; cleared by software, or by hardware when CPU vectors to interrupt service routine
TCON.2	IT1	8AH	External interrupt 1 type flag. Set/cleared by software for falling edge/low-level activated external interrupt
TCON.1	IE0	89H	External interrupt 0 edge flag
TCON.0	IT0	88H	External interrupt 0 type flag

TIMER CONTROL REGISTER (TCON) (Cont'd)

- Timer control register: **TMOD**
  - Upper nibble for timer/counter, lower nibble for interrupts
- **TR** (run control bit)
  - TR0 for Timer/counter 0; TR1 for Timer/counter 1.
  - TR is set by programmer to turn timer/counter on/off.
    - TR=0: off (stop)
    - TR=1: on (start)

(MSB) (LSB)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer 1		Timer0		for Interrupt			

TIMER CONTROL REGISTER (TCON) (Cont'd)

- **TF** (timer flag, control flag)
  - TF0 for timer/counter 0; TF1 for timer/counter 1.
  - TF is like a carry. Originally, TF=0. When TH-TL roll over to 0000 from FFFFH, the TF is set to 1.
    - TF=0 : not reach
    - TF=1: reach
    - If we enable interrupt, TF=1 will trigger ISR.

(MSB) (LSB)

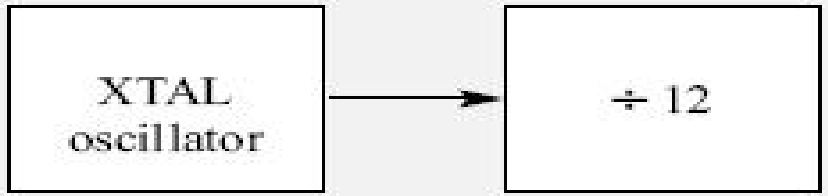
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer 1		Timer0		for Interrupt			

Example 9-2

Find the timer's clock frequency and its period for various 8051-based systems, with the following crystal frequencies.

(a) 12 MHz
(b) 16 MHz
(c) 11.0592 MHz

Solution:



(a) 1/12 × 12 MHz = 1 MHz and T = 1/1 MHz = 1 µs

(b) 1/12 × 16 MHz = 1.333 MHz and T = 1/1.333 MHz = .75 µs

(c) 1/12 × 11.0592 MHz = 921.6 kHz;
T = 1/921.6 kHz = 1.085 µs

NOTE THAT 8051 TIMERS USE 1/12 OF XTAL FREQUENCY, REGARDLESS OF MACHINE CYCLE TIME.

COUNTER

- These timers can also be used as counters **counting events** happening outside the 8051.
- When the timer is used as a counter, it is a pulse outside of the 8051 that increments the TH, TL.
- When C/T=1, the counter counts up as pulses are fed from
  - T0: timer 0 input (Pin 14, P3.4)
  - T1: timer 1 input (Pin 15, P3.5)

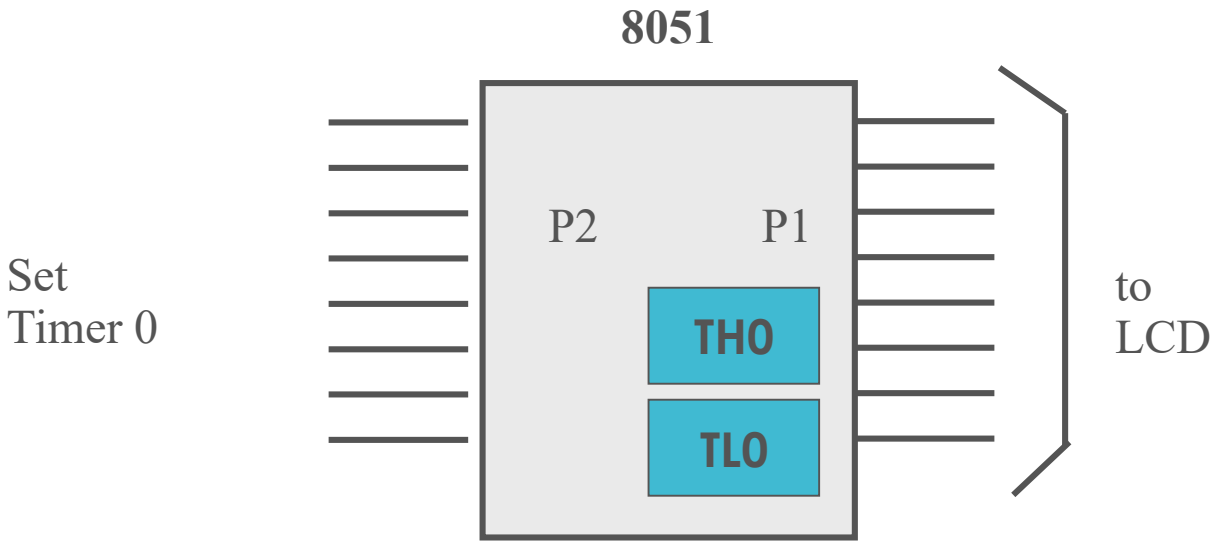
EVENT COUNTING

- If C/T= 1, the timer is clocked from an external source.
- In most applications, this external source supplies the timer with a pulse upon the occurrence of an "event" - the timer is event counting.
- The number of events is determined in software by reading the timer registers TLx/THx
- Port 3 bit 4 (P3.4) serves as the external clocking input for Timer 0 and is known as "T0" in this context. P3.5, or "T1," is the clocking input for Timer 1.

Timer/Counter in 8051 (Summery)

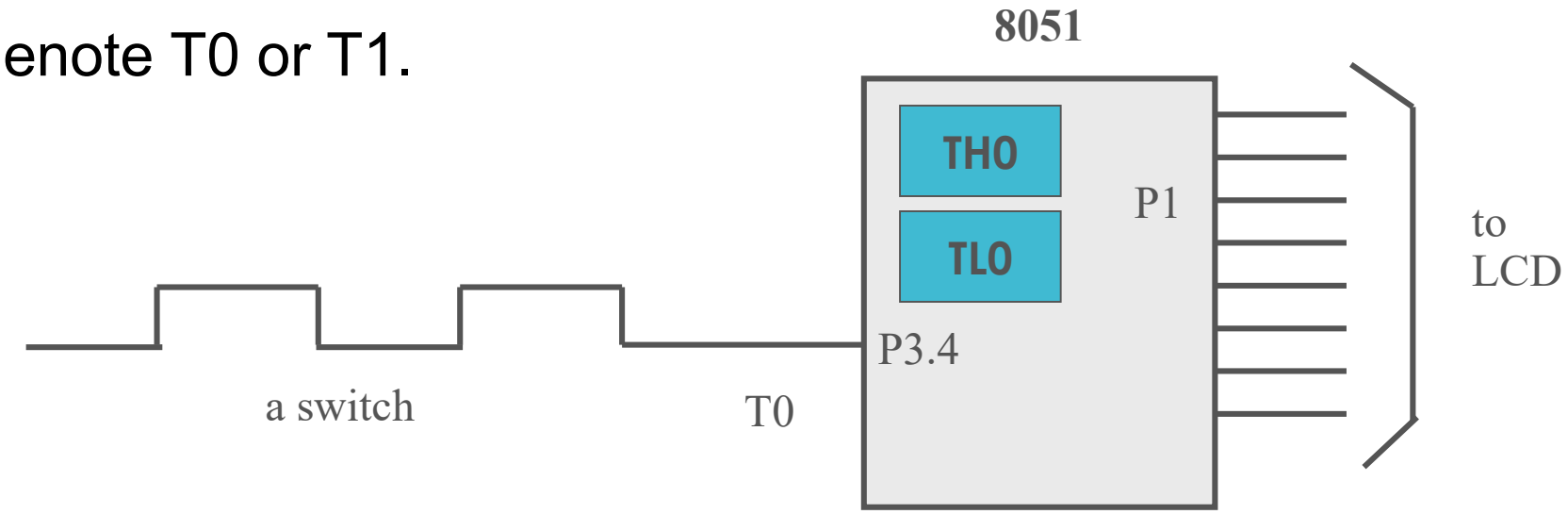
TIMER

- Set the initial value of registers
- Start the timer and then the 8051 counts up.
- Input from internal system clock (machine cycle)
- When the registers equal to 0 and the 8051 sets a bit to denote time out



COUNTER

- Count the number of events
  - Show the number of events on registers
  - External input from T0 input pin (P3.4) for Counter 0
  - External input from T1 input pin (P3.5) for Counter 1
  - **External input** from Tx input pin.
  - We use Tx to denote T0 or T1.



CHAPTER-4B

INPUT-OUTPUT INTERFACING

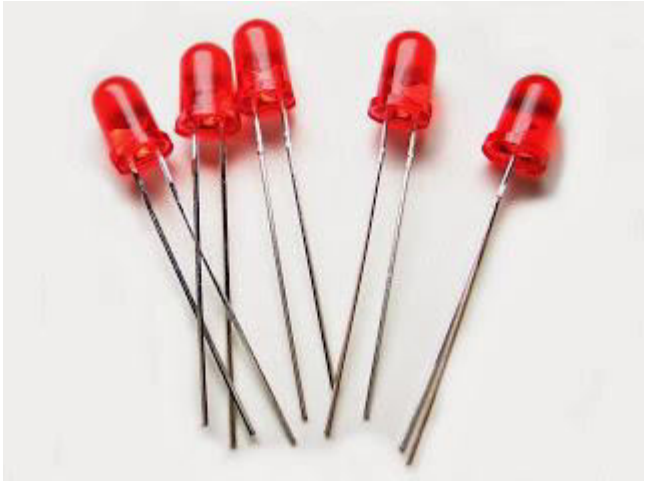
TECHNICAL AND VOCATIONAL TRAINING INSTITUTE
Department of Electrical electronics technology

Fundamentals of microprocessors and microcontroller
EETe 3032

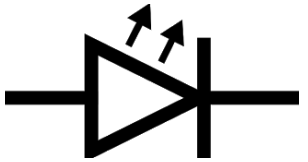
By Zemenu T. /2022

INTERFACING LED WITH 8051

LED (Light Emitting Diodes) are simple lights that you can turn **ON/OFF**. They glow when they are ON and they don't when they are not. They look somewhat familiar to the picture below



Standard RED LEDs

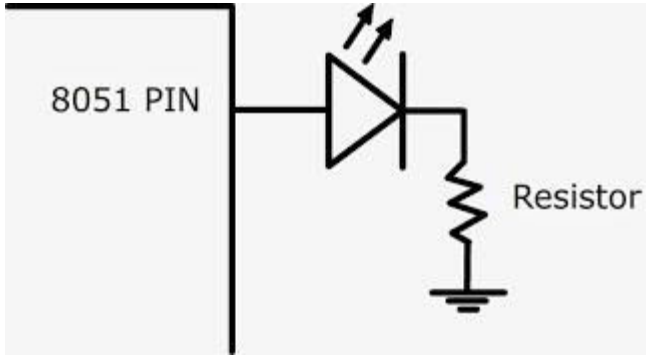


LED Symbol

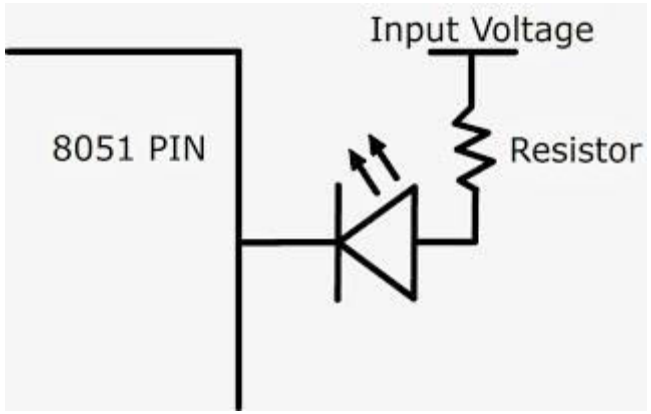
The bigger leg represents the POSITIVE terminal while the shorter one is definitely NEGATIVE. So you take care of the legs when you use them (high voltage goes to POSITIVE while NEGATIVE is connected to GROUND of the power supply). In our diagrams, we are used to representing it with the symbol as shown so don't get confused. The left terminal is POSITIVE while the right end is NEGATIVE terminal.

INTERFACING LED WITH 8051 (Contd.)

There are two genuine methods of interfacing (in simple words, connecting) LED to your 8051. Both will work just fine but have opposite programming techniques.



LED INTERFACE 1



LED INTERFACE 2

The resistor is important in **interface 2** to limit the flowing current and avoid damaging the LED and/or MCU.

•Interface 1 will glow LED ONLY if the PIN value is HIGH as current flows towards ground.

•Interface 2 will glow LED ONLY if the PIN value is LOW as current flows towards PIN due to its lower potential.

INTERFACING switch WITH 8051

Switches come in many different shapes and sizes. The main idea or function is to either enable or disable something by simply turning it on/off. Thus it acts as an input device that users can use to control certain parts of the system. Below are the most common switches that you will come across

Push Button

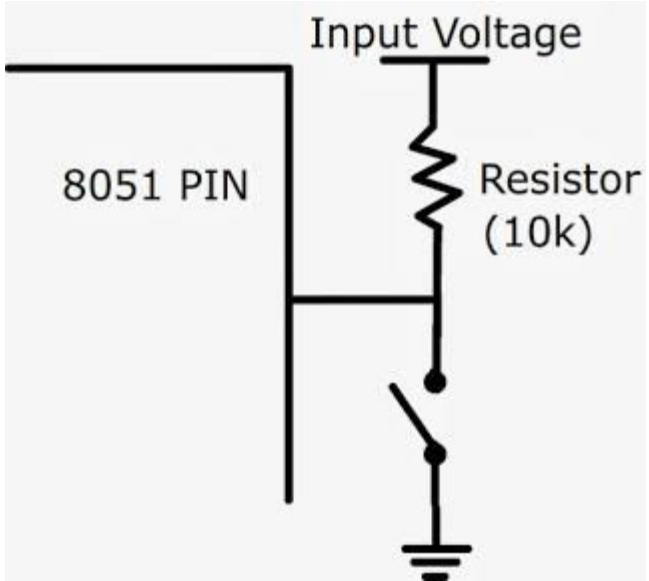
DIP (Dual in-line Package) Switch

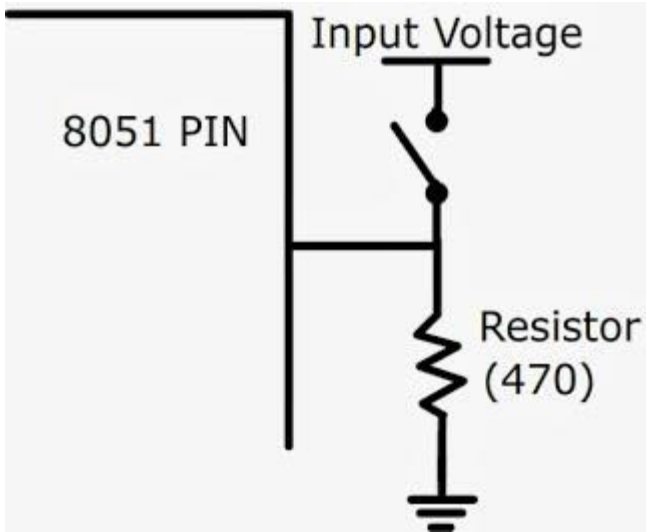
ON-OFF Switch

In circuit diagrams, a switch is generally represented by

INTERFACING switch WITH 8051 (Contd.)

After examining the Switch Symbol, I think it is quite clear how a switch actually works. It **closes** (connects) or **opens** (disconnects) a circuit. The **Push Button** behaves in the same fashion as it closes the circuit as long as you keep holding it in pressed state. You can connect a switch to 8051 MCU using different configurations but the basic functionality remains the same. Always remember this cool fact that the user is never concerned about the actual implementation. He/she just wants something to turn ON when the switch is ON and vice versa OFF. As for you, you can handle it in many different ways. For example, you can connect a switch with 8051 MCU in the following ways.



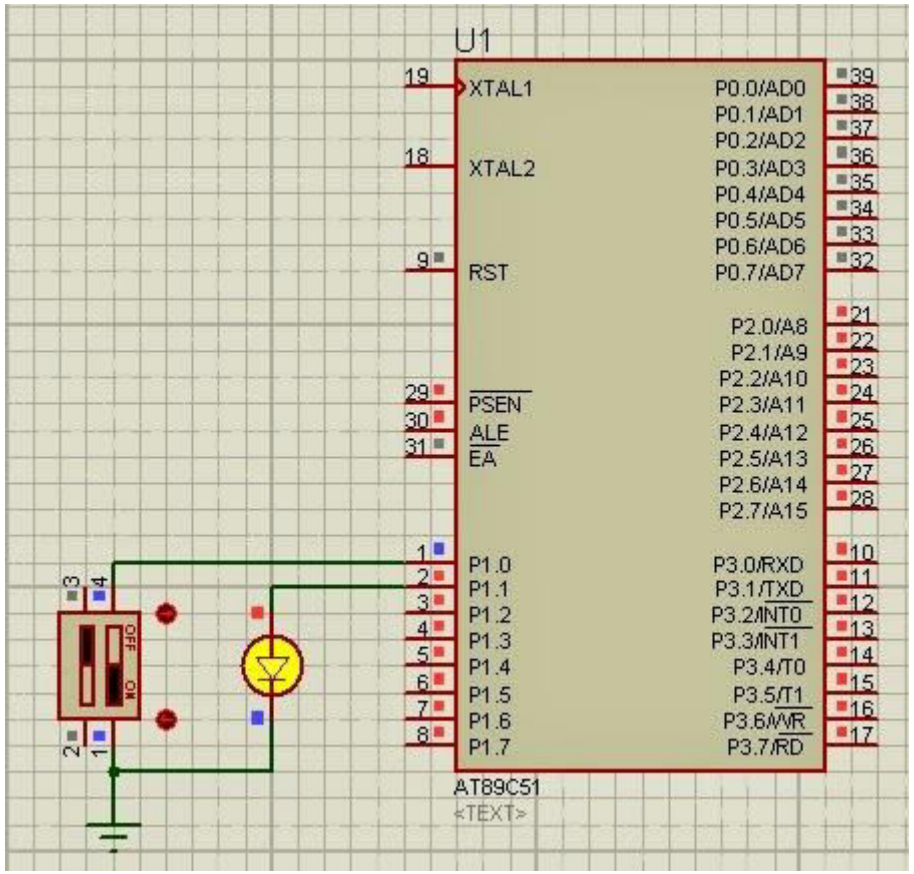


Switch Interface CODE

code.c* | REGX51.H

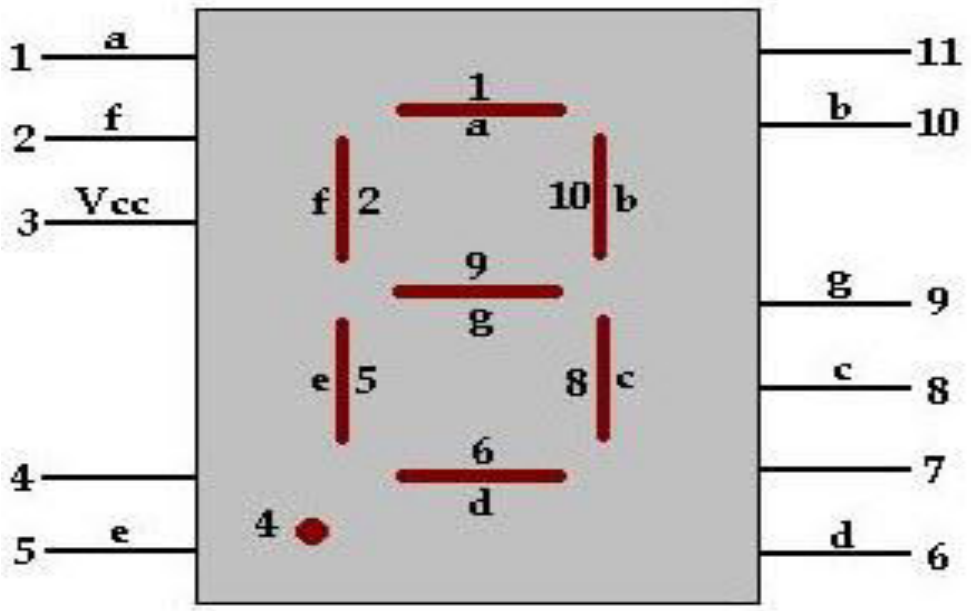
```
1  #include <REGX51.H>
2
3  #define sw P1_0
4  #define led P1_1
5
6  void main()
7  {
8    sw = 1;
9    led = 0;
10
11   while(1)
12   {
13     if(!sw)
14       led = 1;
15
16     else
17       led = 0;
18   }
19 }
```

Proteus simulation

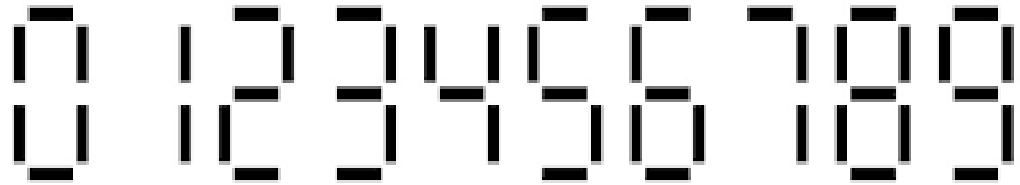


The seven Segment

A Seven segment display is the most basic electronic display. It consists of eight LEDs which are associated in a sequence manner so as to display digits from **0 to 9** when proper combinations of LEDs are switched on. A 7-segment display uses seven LEDs to display digits from 0 to 9 and the 8th LED is used for dot. A typical seven segment looks likes as shown in figure below.



7-Segment Display



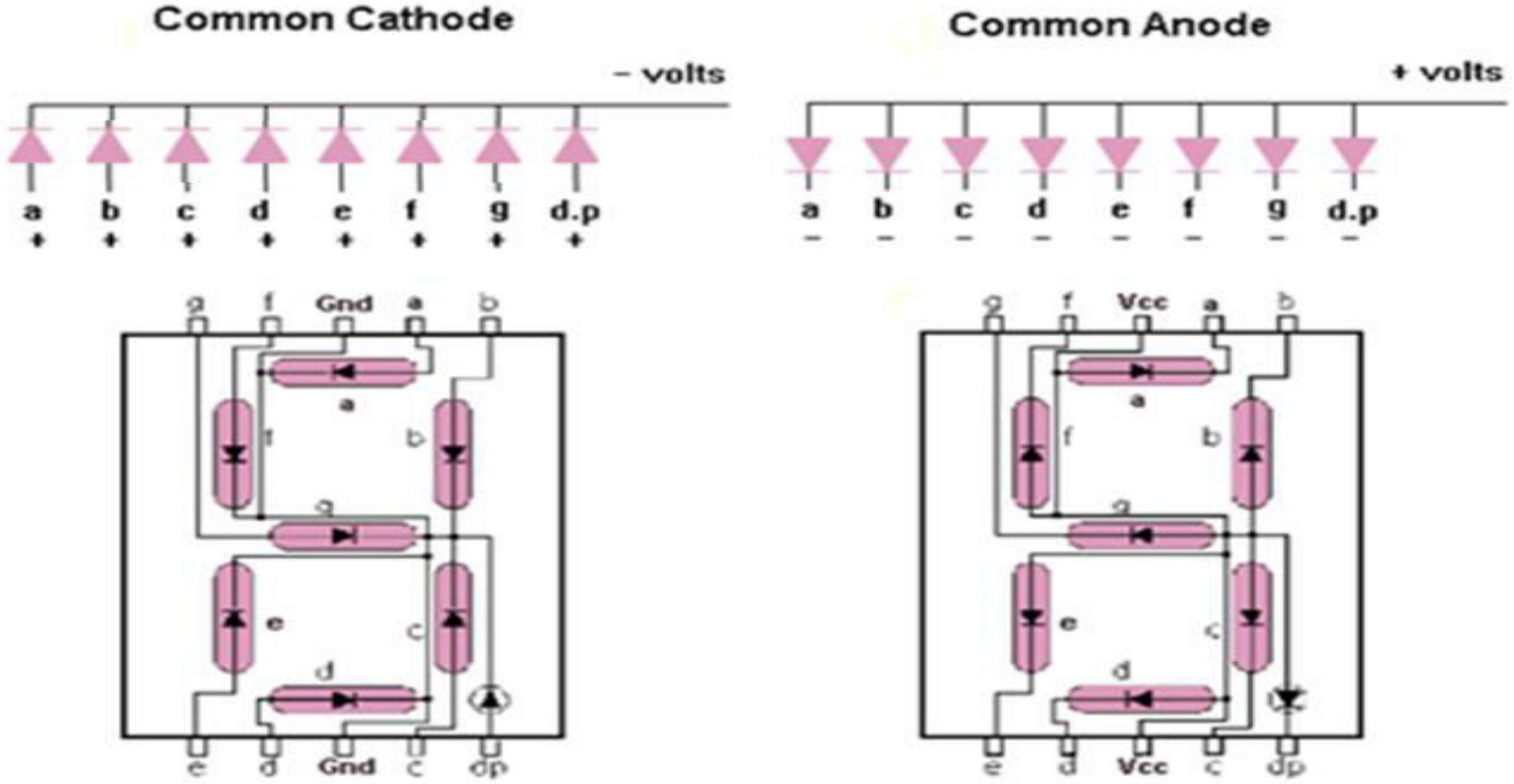
SSD Output (Numbers)

The seven Segment (Contd.)

The 7-segment displays are used in a number of systems to display the numeric information. They can display one digit at a time. Thus the number of segments used depends on the number of digits to display. Here the digits 0 to 9 are displayed continuously at a predefined time delay.

The 7-segment displays are available in two configurations which are **common anode** and **common cathode**. Here common anode configuration is used because output current of the microcontroller is not sufficient enough to drive the LEDs. The 7-segment display works on negative logic, we have to provide logic 0 to the corresponding pin to make on LED glow.

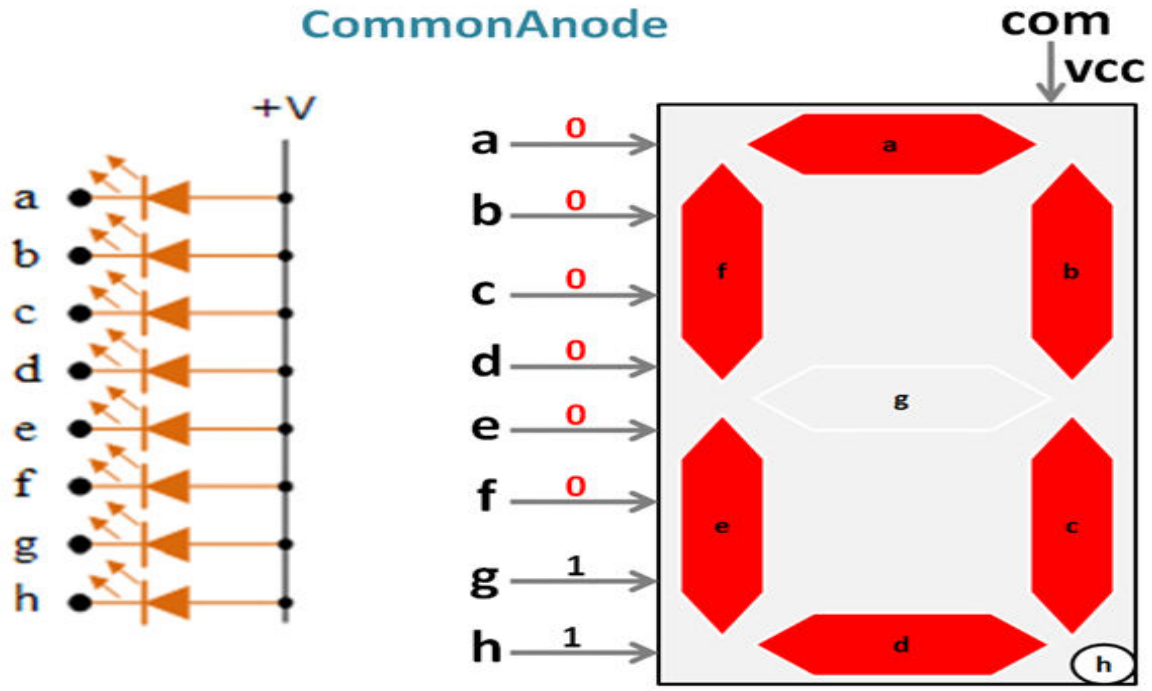
The seven Segment (Contd.)



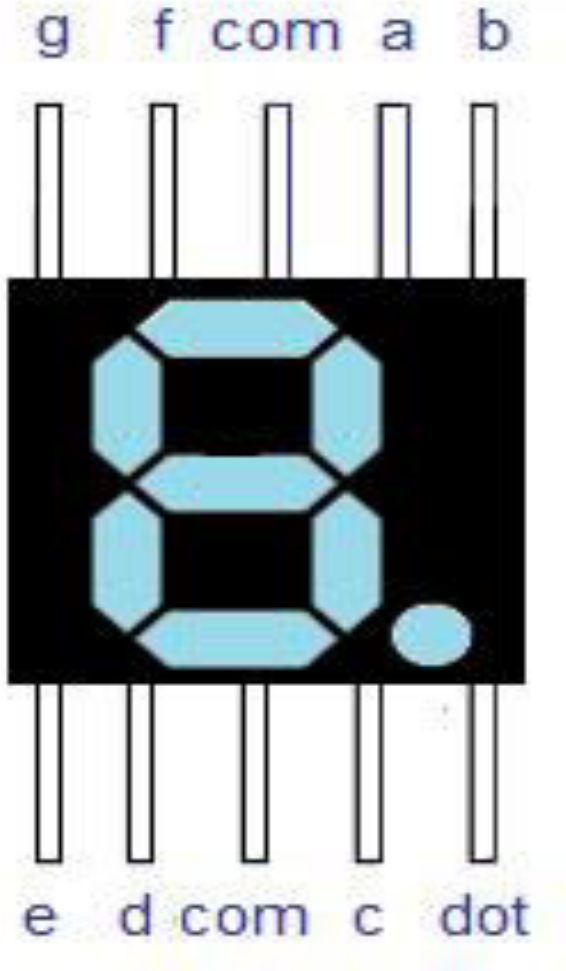
7-Segment Display Configurations

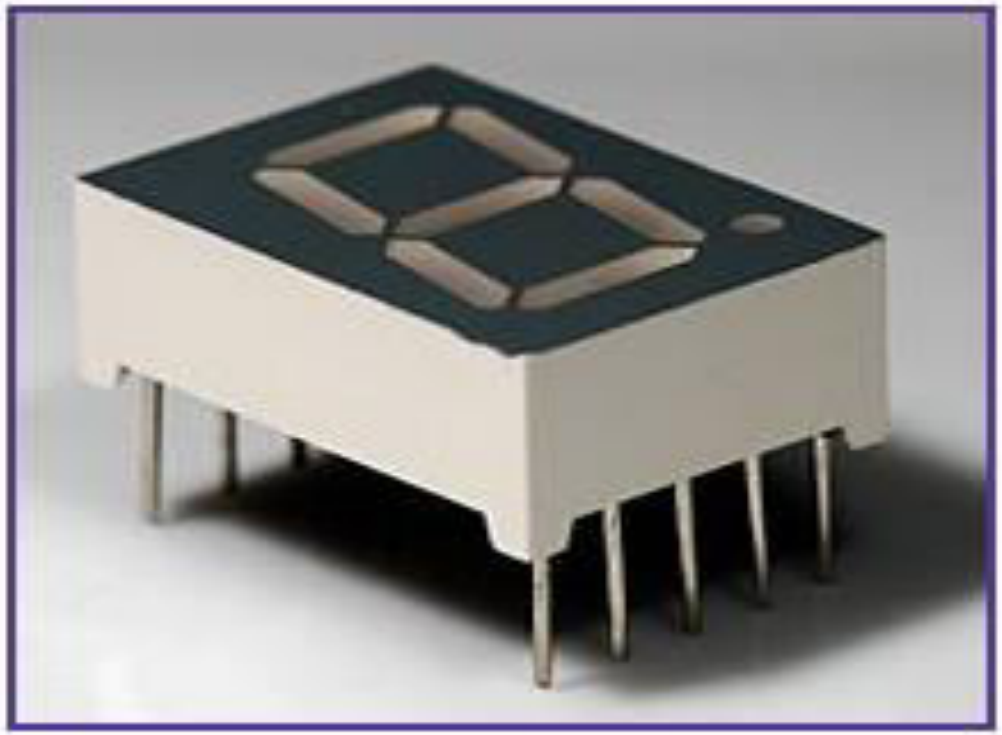
The seven Segment (Contd.)

- **Common Cathode**: In this type of segments all the cathode terminals are made common and tied to GND. Thus the segments **a** to **g** needs a logic High signal(**5**v) in order to glow.
- **Common Anode**: In this type of segments all the anodes terminals are made common and tied to VCC(5v). Thus the segments **a** to **g** needs a logic LOW signal(GND) in order to glow.



The seven Segment (Contd.)





The following table shows the hex values used to display the different digits.

Common Cathode

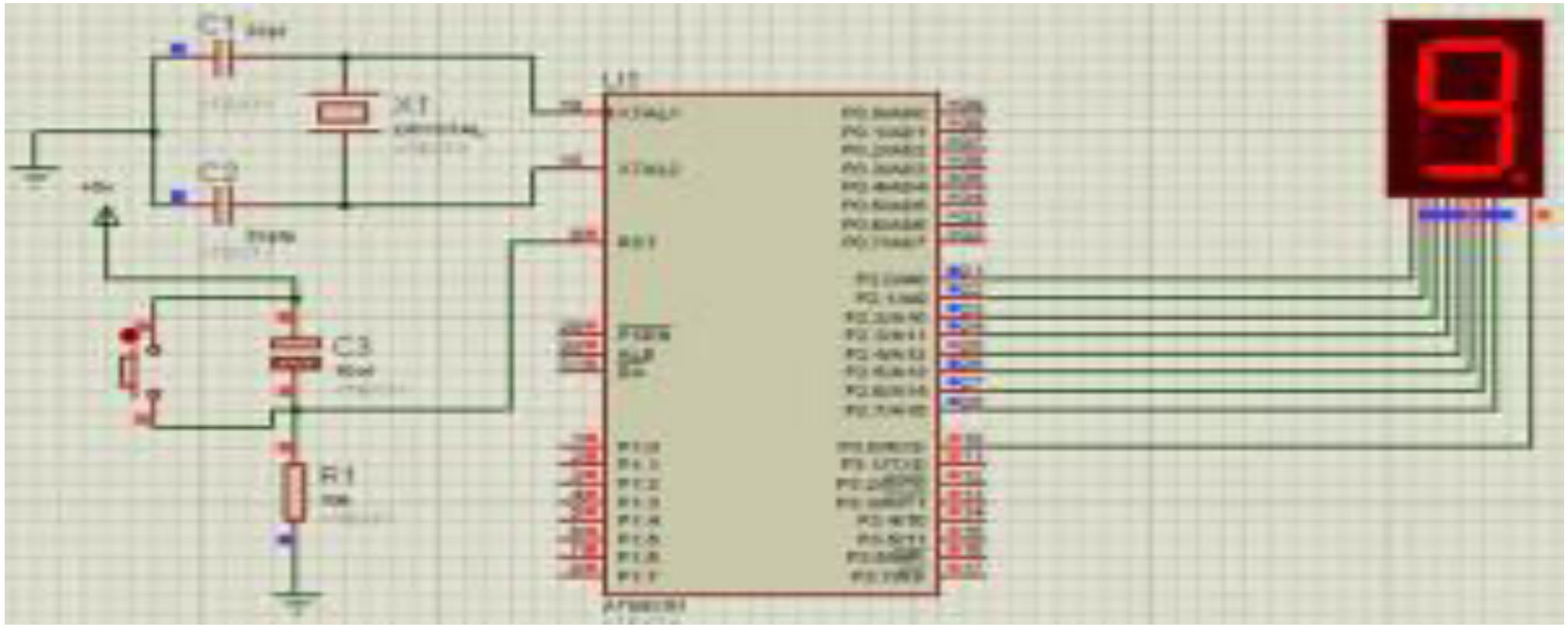
Digit	a	b	c	d	e	f	g	Hex Code
0	1	1	1	1	1	1	0	0x3F
1	0	1	1	0	0	0	0	0x06
2	1	1	0	1	1	0	1	0x5B
3	1	1	1	1	0	0	1	0x4F
4	0	1	1	0	0	1	1	0x66
5	1	0	1	1	0	1	1	0x6D
6	1	0	1	1	1	1	1	0x7D
7	1	1	1	0	0	0	0	0x07
8	1	1	1	1	1	1	1	0x7F
9	1	1	1	1	0	1	1	0x6F

The following table shows the hex values used to display the different digits.

Common Anode

Digit	a	b	c	d	e	f	g	Hex Code
0	0	0	0	0	0	0	1	0x40
1	1	0	0	1	1	1	1	0x79
2	0	0	1	0	0	1	0	0x24
3	0	0	0	0	1	1	0	0x30
4	1	0	0	1	1	0	0	0x19
5	0	1	0	0	1	0	0	0x12
6	0	1	0	0	0	0	0	0x02
7	0	0	0	1	1	1	1	0x78
8	0	0	0	0	0	0	0	0x00
9	0	0	0	0	1	0	0	0x10

Circuit Diagram



7-Segment common Anode Display interfacing

Source Code:

```
#include<reg51.h>
sbit a= P3^0;
void main()
{
unsigned char n[10]= {0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10};
unsigned int i,j;
a=1;
while(1)
{
for(i=0;i<10;i++)
{
P2=n[i];
for(j=0;j<60000;j++);
}
}
}
```

BCD to Seven segment Decoder

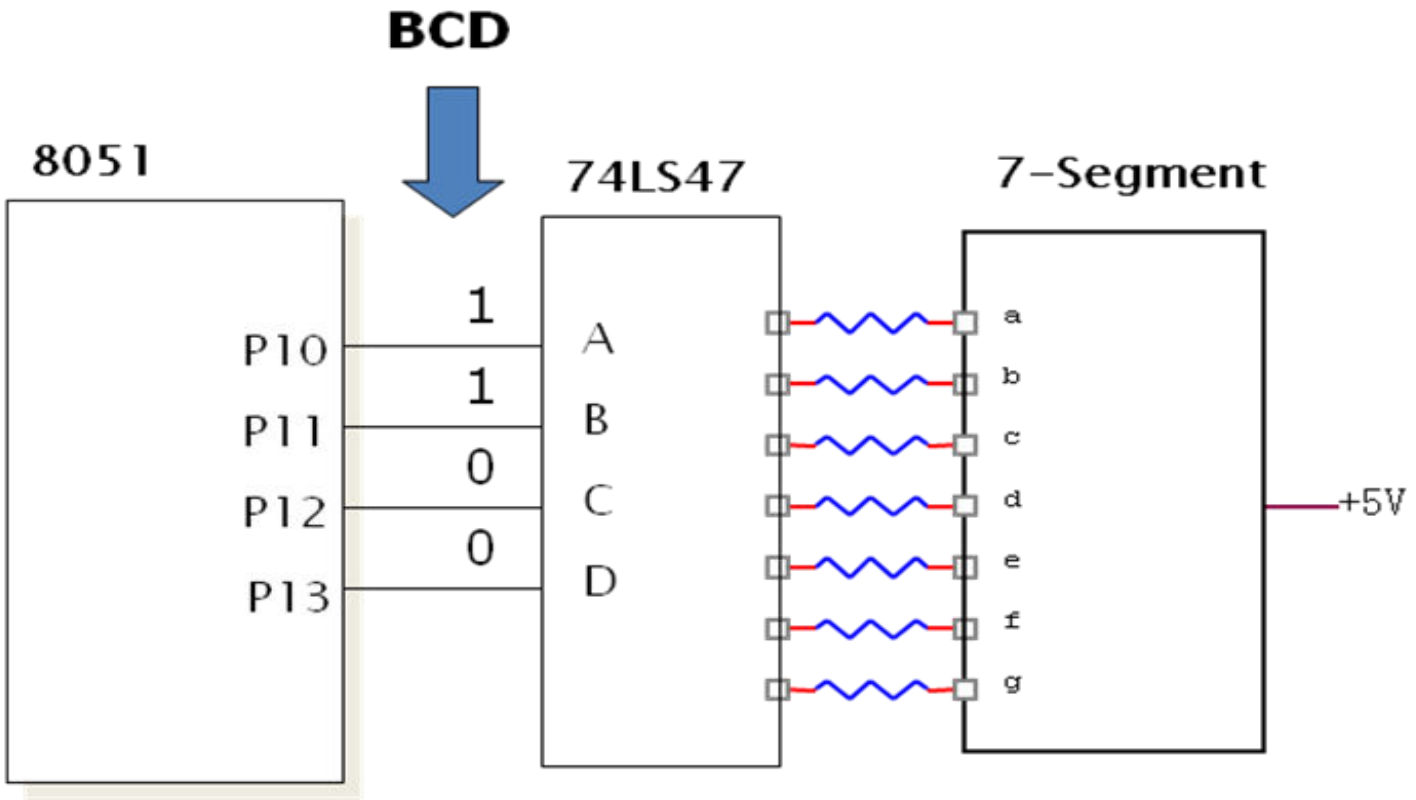
For interfacing it with 8051 MCU, you can use **74LS47 IC.** It is basically **BCD to 7-segment Decoder/Driver** which is specially designed for driving an SSD. It's function is such that it gives out the corresponding sequence for SSD when the MCU sends a number 0-9. You can find more details about the IC in it's datasheet Here is the truth table for common cathode configuration

s	BCD	G	F	E	D	C	B	A
0	0000	0	1	1	1	1	1	1
1	0001	0	0	0	0	1	1	0
2	0010	1	0	1	1	0	1	1
3	0011	1	0	0	1	1	1	1
4	0100	1	1	0	0	1	1	0
5	0101	1	1	0	1	1	0	1
6	0110	1	1	1	1	1	0	1
7	0111	0	0	0	0	1	1	1
8	1000	1	1	1	1	1	1	1
9	1001	1	1	0	1	1	1	1

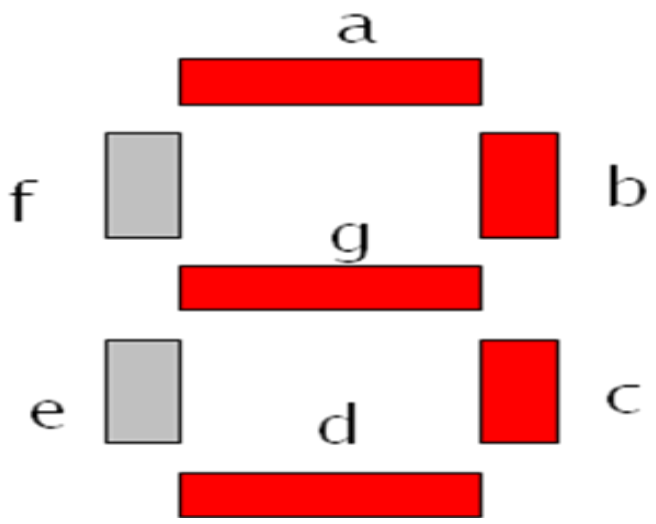
BCD to 7-SEGMENT Decoder Truth Table

	P17	P16	P15	P14	P13	P12	P11	P10
0x3 =	0	0	0	0	0	0	1	1

```
void main()
{
   P1 = 0x3;
}
```



The number ‘3’ will be displayed on the digits



8051 KEYPAD INTERFACE

Keypad is an **input device** and don't get confused by the name because it is nothing but a **keyboard**. So now we are definitely talking of many buttons which can serve as a keyboard for the user to input some data to our microcontroller. However, it can be of different shapes and sizes. For instance

1	2	3
4	5	6
7	8	9

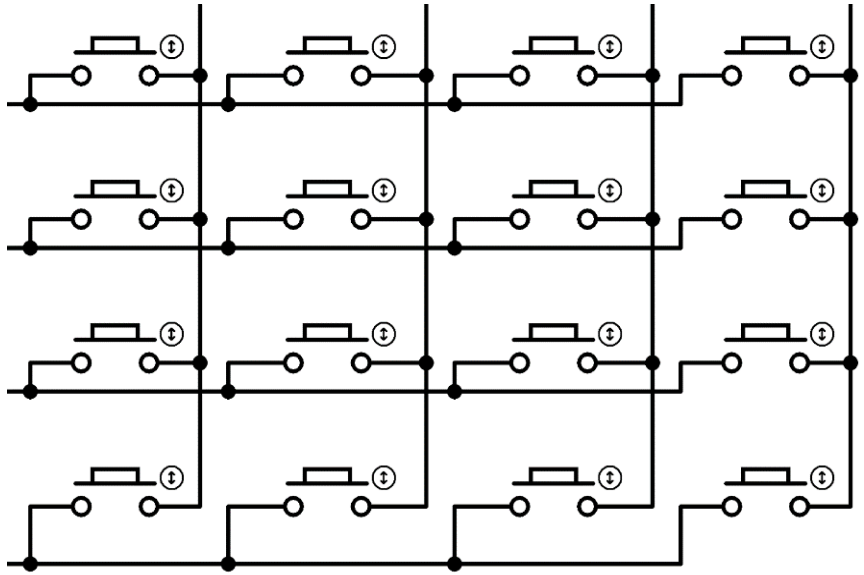
3x3 NUMERIC KEYPAD

1	2	3	F
4	5	6	E
7	8	9	D
A	0	B	C

4X4 HEX KEYPAD

These are just two cases. We can have it in many other shapes. The key point here is to understand the story behind it and only then will you be able to tackle any type of keypad for interfacing. "Internally, each button of a keypad is nothing but a PUSH button (switch)"

8051 KEYPAD INTERFACE (Contd.)



KEYPAD INTERNAL STRUCTURE

	C1	C2	C3	C4
R1	1	2	3	F
R2	4	5	6	E
R3	7	8	9	D
R4	A	0	B	C

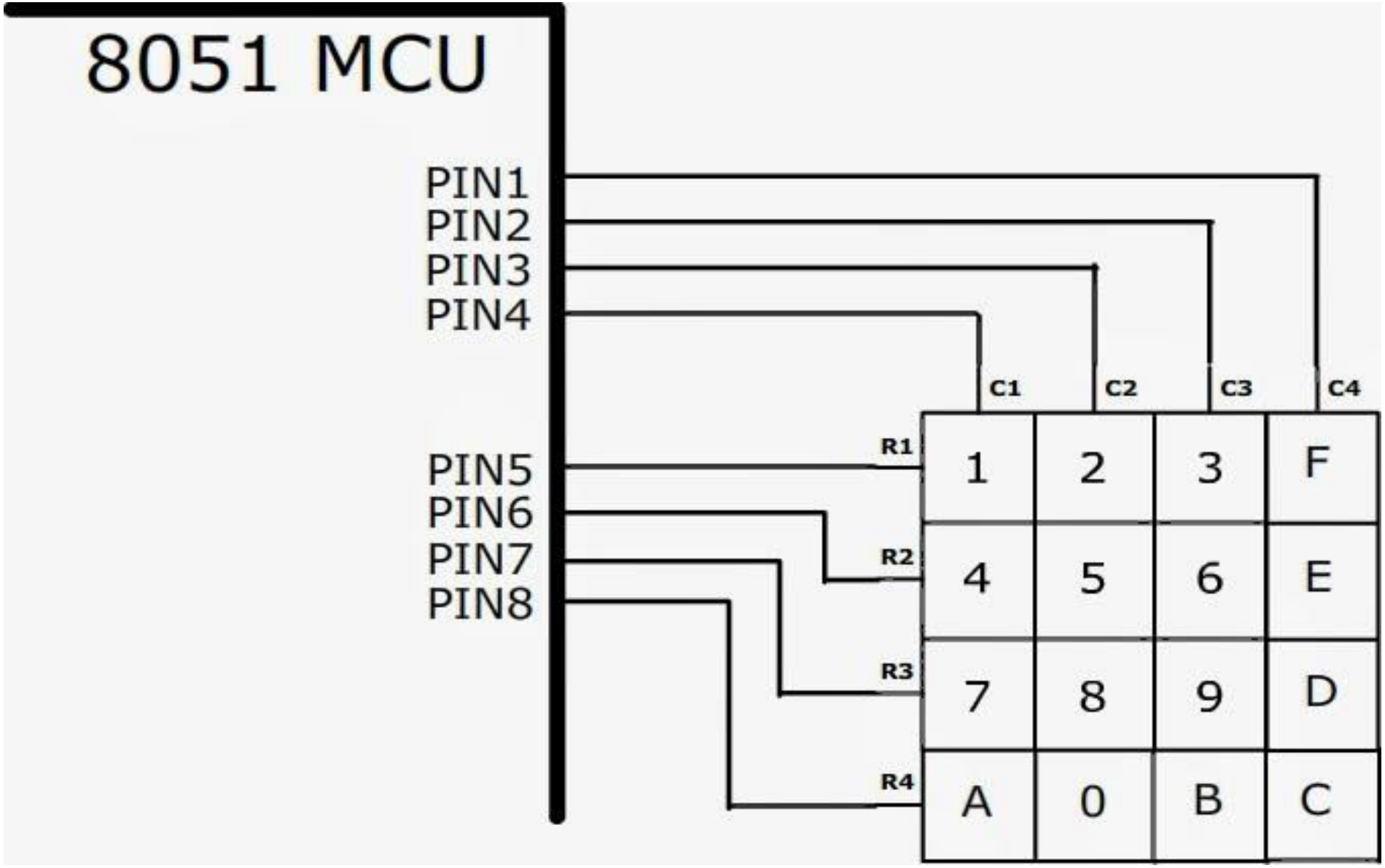
4x4 HEX KEYPAD (PRACTICAL)

As you can see, it has four columns and four rows (thus 4x4). In theory, pressing a particular button will simply connect the corresponding **ROW - COLUMN** lines and that's it. That is exactly how a keypad actually works. A button doesn't know what it represents (1,2,3 e.t.c.). Similarly a keypad has no idea what each of its buttons represent because it's just a simple circuitry for many buttons at one place.

8051 KEYPAD INTERFACE (Contd.)

If we say 4x4, then it will definitely have four columns and four rows (as shown). It is clear now, from the internal structure information, that if you press button '**5**' then **R2** and **C2** will be shorted (connected). But then at the same time the most important question comes to mind. Who exactly tells the microcontroller that it was button '5'? Because we know now that it was just a PUSH button and we also know that it won't shout '5' when we press it. So where does the magic happen? The answer is as simple as it can be: **Program Code**. It is the code itself that will monitor everything. It has to **detect** a button press and **interpret** the result. So interfacing a keypad to a microcontroller unit is purely a programmer's kind of thing because everything related to keypad is handled by the code. It's like creating something out of nothing and that's the beauty of it.

8051 KEYPAD INTERFACE



8051 LCD INTERFACE

LEDs (lights), interfaced with 8051 microcontroller, can be used to display binary numbers or on/off states while Seven Segment Displays (SSD) can display digits but that's not enough. What if you wanted to display a proper message consisting of **numbers, letters, characters, symbols** e.t.c.? That's where **LCD Screens** jump in. LCDs make it convenient to display anything to the user. It may sound tricky to interface an LCD with a microcontroller, it's simpler than it looks and this short tutorial will make sure that you get the hold of it.

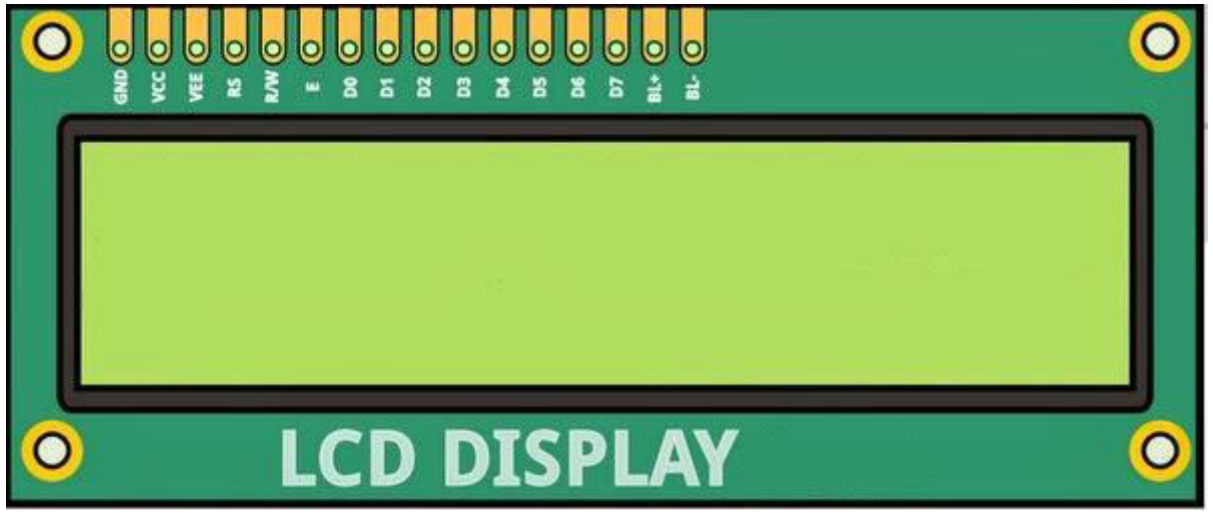
LCD

Stands for Liquid Crystal Display and is definitely an output device. It comes in many popular shapes and sizes but here is the one that you will most probably face at the beginner's level.

COMMON LCD

8051 LCD INTERFACE (Contd.)

Let's focus on this one. It is basically called **16x2 LCD** because it has sixteen **columns** and two **rows** thus it can display **thirty-two** (32) characters at a time (sixteen characters in each row). There are sixteen hardware pins as shown.



LCD HARDWARE PINS

Pin Description

Number	Pin	Function
1	GND	Ground
2	Vcc	Power Source (+5V)
3	Vee	Contrast Control
4	RS	**Register Select** 0=Command Register, 1=Data Register
5	R/W	**Read/Write** 0=Write, 1=Read
6	E	Enable
7-14	D0 – D7	8-bit data pins
15	BL+	Backlight Control
16	BL-	Ground

8051 LCD INTERFACE (Contd.)

LCD comes with two important registers that we can manipulate.

- **Control Register**

Used for **instructing** LCD on what to do next. It's like talking to your LCD using this register.

- **Data Register**

Used for **displaying data** on LCD.

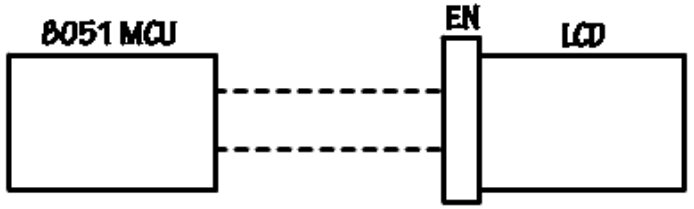
You have to use **Control Register** for sending commands to LCD and when it's ready, only then you can use **Data Register** to display your data on the LCD. In other words, first you have to talk to your LCD (by sending commands) about what you want it to do and then send all the useful data for display.

8051 LCD INTERFACE (Contd.)

The **RS** (Register Select) control pin is used to select either command or data register as described in the pin configuration table. Once selected, all data sent on the 8-bit data lines will be latched to that register. The **R/W** (Read/Write) control pin is used to determine the flow of data. You have to select

- **Write Mode** when you're sending something **to** the LCD (data or command)
- **Read Mode** when you're reading **from** the LCD

The **E** (Enable) control pin acts as a guard to allow exchange of data. This pin is very important in this whole process and must be handled precisely. The following figure explains its function.



SENDING DATA TO LCD

8051 LCD INTERFACE (Contd.)

COMMANDS

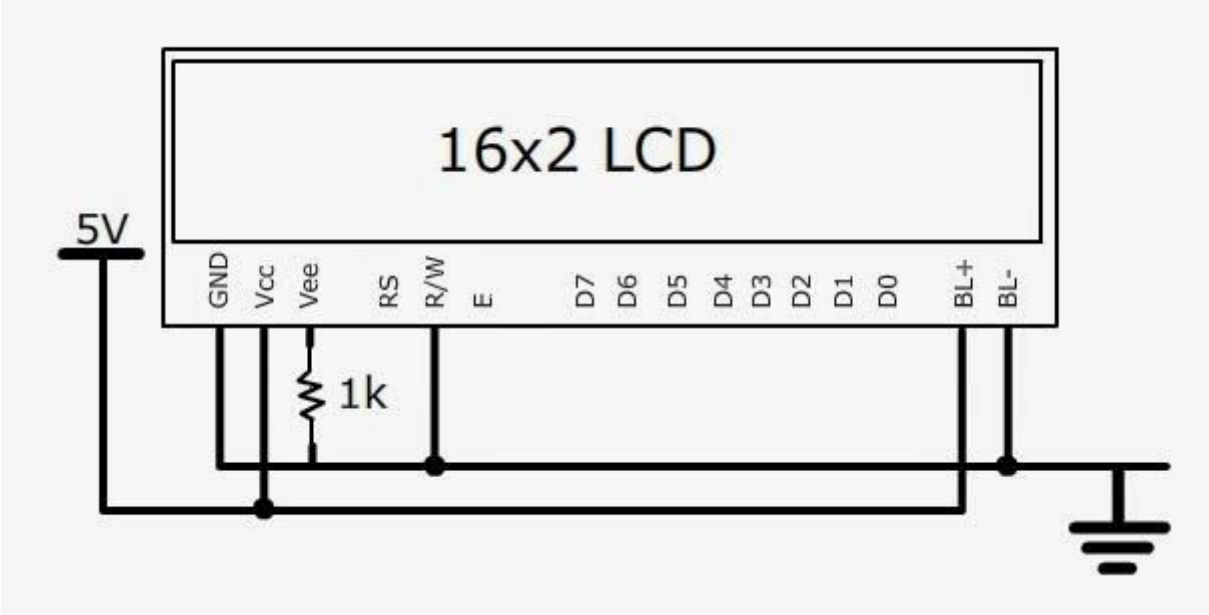
LCDs have some useful handful of commands that you can use for different operations. You can check a comprehensive list of commands here. We are going to concentrate only on the commands required to initialize an LCD because they are fairly sufficient to get things done.

LCD uses following command codes:

Hex Code	Command to LCD Instruction Register
1	Clear screen display
2	Return home
4	Decrement cursor
6	increment cursor
E	Display ON, cursor ON
80	Force the cursor to the beginning of the 1st line
C0	Force the cursor to the beginning of the 2nd line
38	Use 2 lines and 5*7 matrics

HARDWARE CONNECTIONS

Here is my optimized version of LCD hardware Connections. It takes the minimum possible time and works most of the time.

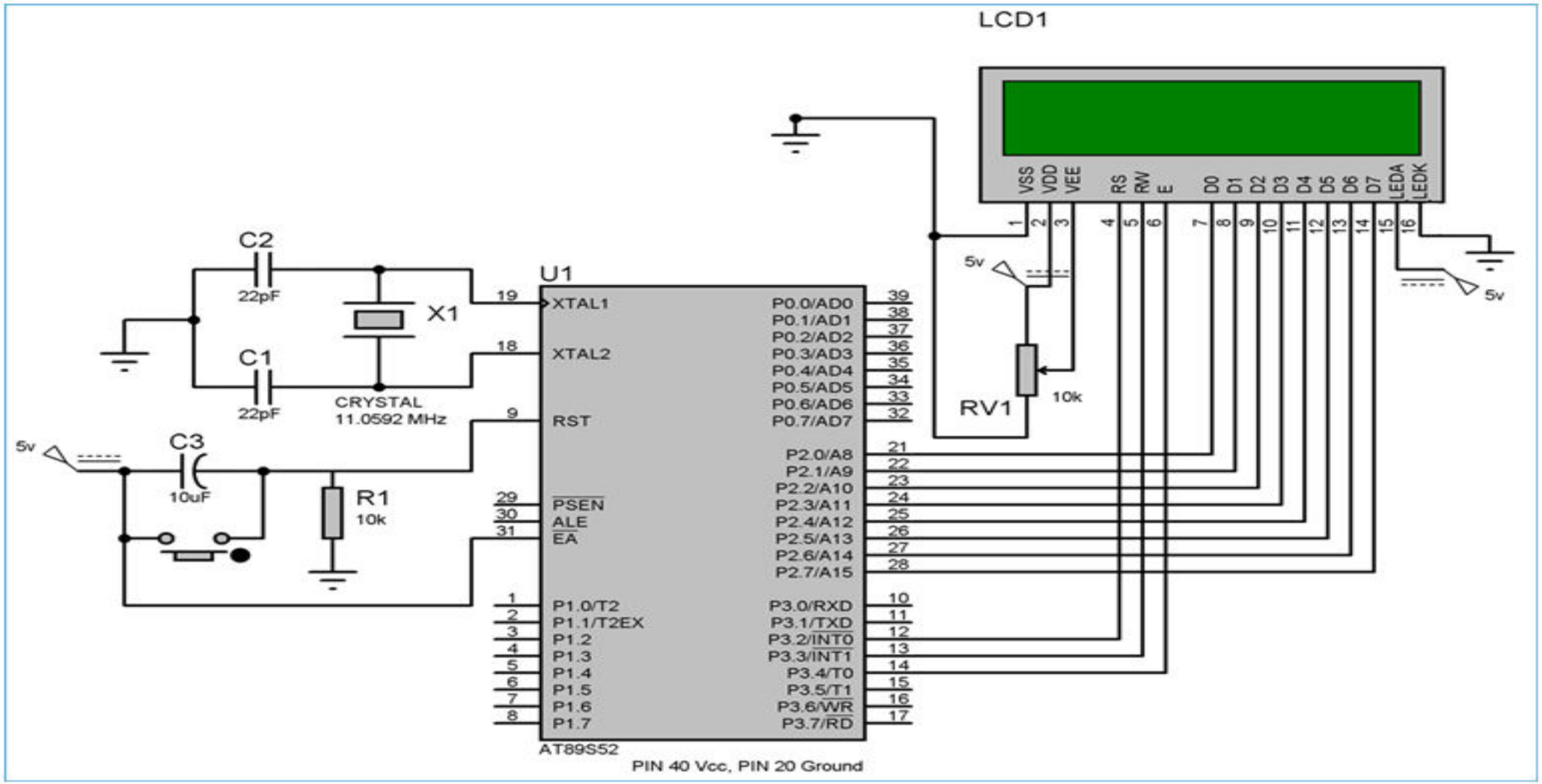


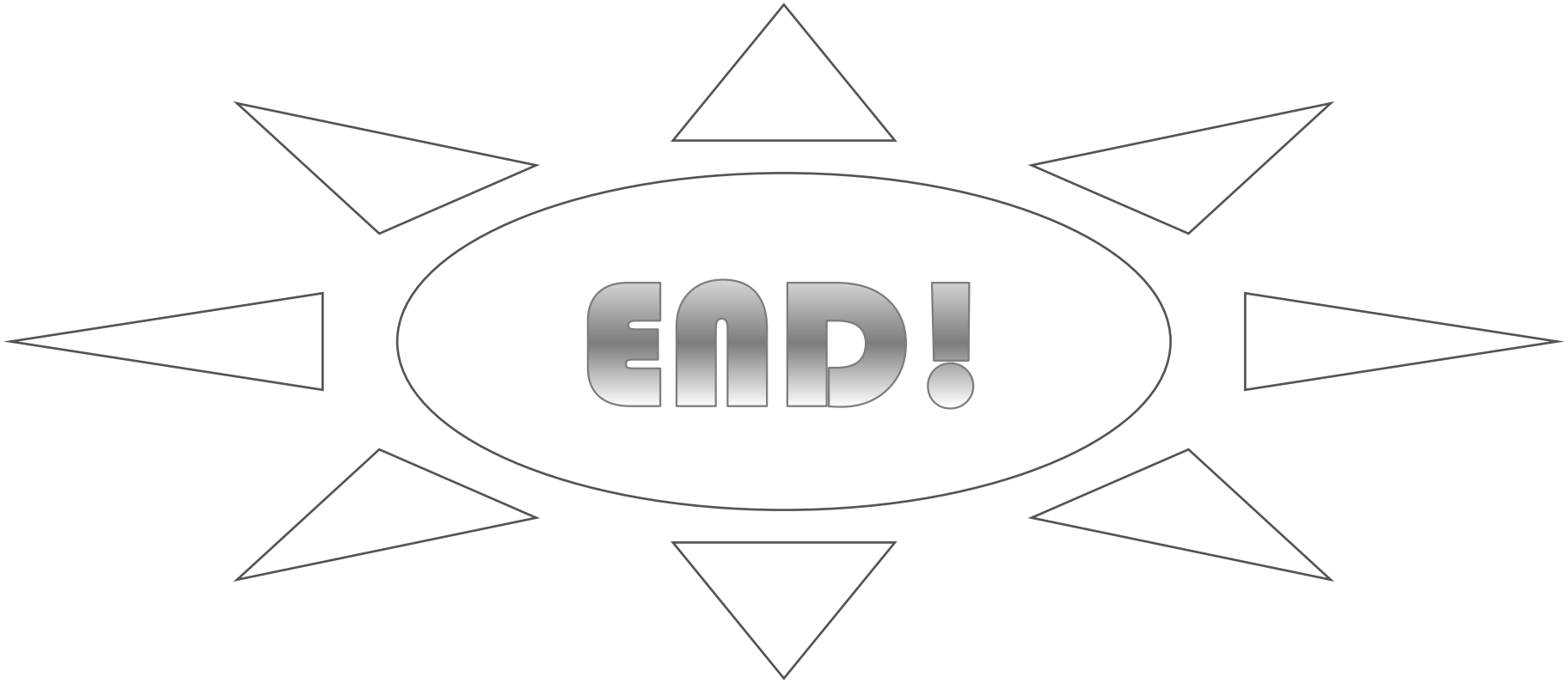
However it is assumed that we are

- Always writing to LCD (R/W = 0)
- Constant Contrast (through 1K ohm resistor)
- Full Brightness (no resistor in the BL connections)

The rest of the pins are connected to microcontroller. For instance, connect **potentiometers** to **Vee** or **BL** controls to control the Contrast or Brightness yourself.

Interfacing 16x2 LCD with 8051




END!

CHAPTER-4

APPLICATION OF MICROCONTROLLER

TECHNICAL AND VOCATIONAL TRAINING INSTITUTE
Department of Electrical electronics technology

Fundamentals of microprocessors and microcontroller
EETe 3032

By Zemenu T. /2022

Application of microcontroller

The hardware and software design and configuration scale of the microcontroller application system is based on the functional requirements of the application system, and has the best performance price ratio. After the use of microcontrollers, many hardware circuits can be implemented by software (called softening of hardware), which can greatly reduce the hardware structure of the system. This can reduce the cost, on the other hand, it also greatly improves the reliability of the system. The application system of micro controller has the characteristics of small size, low power consumption, strong function and high reliability. The micro controller has been widely used in various fields, such as household appliances, office equipment, measurement and control system, intelligent instrument, communication system and mechanical and electrical industry.

Application of microcontroller (Contd.)

With the advent of cheap microcontroller, its application will be more and more extensive. Our world is filled with electronic products that entertain us, make our lives easier, allow us to communicate with each other, help us prevent diseases, and much more. More often the heart of all these electronic products is a ***Microcontroller***. This awesome device is the brain that controls all actions and takes care of computations within an electronic device. Because of its endless capabilities, there are many uses of microcontroller and it is not confined to one singular application. A microcontroller can be found in applications such as consumer electronics, medical devices, the automotive industry, aviation, marine, and much more.

Application of microcontroller (Contd.)

Mechatronics products: The combination of micro controller and traditional mechanical products makes the structure of traditional mechanical products simplified, intelligent control, and the human-machine interface is more friendly, forming a new generation of mechanical and electrical integration products. Such as micro controller controlled knitting machine, CNC machine tool and so on.

Measurement and control system: A variety of industrial control systems, adaptive control systems, data acquisition systems, such as automatic control of electroplating production line, are made up of microcontrollers.

Intelligent sensor: The combination of the micro controller and the sensor constitutes the intelligent sensor, which can easily realize the nonlinear correction.

Consumer Electronics

Consumer electronics is a term that defines electronic equipment that is used on a daily basis (mostly in one's own home). Below are some example of consumer electronics that would have a microcontroller;

Radio

Microwave

Television

Heater/ Fan

Calculator

MP3 Players

Christmas lights

3D Printers

Medical

The ***Medical*** field has benefited immensely from using microcontrollers. As you can imagine, medical practices before the advent of the microcontroller would have been quite primitive. While us humans have amazing abilities, and intelligence, there is a limit to what we can detect within a human body.

Microcontrollers, along with other technology enable us to be able to detect a disease with a human body before it is too late.

Medical (Contd.)

These devices include;
•X-ray
•Magnetic Resonance Imaging (MRI)
•Electrocardiogram (ECG)
•Computed Tomography (CT scan)
•Sonography (Ultrasound imaging)
•Heart rate monitors
•Blood pressure monitors
As well as detection, microcontrollers are used in medical devices to help perform the functions of organs that might have deteriorated with time, are not working right, or might have been damaged. Devices such as;
•Dialysis machines (performs function of liver)
•Pacemakers (used to control abnormal heart rhythm)
•Hearing aids

Entertainment

In the age before electronics and microcontrollers you might have been limited to a few activities to help pass your time. Things like reading a book, going for a walk, playing the guitar, playing hide and seek, etc (Not that any of these are bad). Nowadays, we are all spoiled with a plethora of devices to entertain ourselves. Even before the microcontroller, there were electronic devices which could help entertain and pass your time. But, these devices grew not only in quantity, but complexity and quality as well with the help of the microcontroller. Your old transistor radio might have been limited to a couple of stations, with subpar sound quality. But, now your digital radio will have the capability to access all stations on the FM and AM bandwidth with superior sound quality, while being able to play CDs as well.

Entertainment (Contd.)

Below are some examples of electronic devices to help entertain you in one way or another;

- Television
- Computer/Laptop
- Projector
- Ipad
- Digital radio
- Playstation
- Smartphone
- etc

Instrumentation and process control

Instrumentation and Process control is a field of Engineering that deals with control theory. It is used to manufacture equipment to help design, monitor, and control many different processes across many different industries. These processes ensure that the overall system is productive, efficient, and repeatable. The overall system consists of software and hardware which includes, sensors, actuators (mechanical and electrical), and of course a microcontroller. Below are industries that use implement instrumentation and process control;

Oil and Gas
Petrochemical
Wastewater treatment
Fabric
Food and Beverage

Communication

Communication is a necessity in everyday life. We use communication for many different purposes; *to share information*, *make a statement, ask questions, express desires, keep in touch with loved ones, express emotions* and much more. But, most of the time you might not be in the same room with the person you might want to communicate with. They might be in another household, city, or country. Before you might have had to write them a letter, post it and wait a couple weeks for a reply. Nowadays, microcontrollers are used in many of these electronic communication devices to allow us to communicate with loved ones, colleagues, strangers, instantaneously, even if they are in another country.

Communication (Contd.)

Devices such as;

- Smartphones
  - Calling
  - Video calling
  - Texting
  - Email
- Computers and Laptops
  - Video calling
  - Email
  - Blogging
- Telephones
- Fax machines

Office

There are a myriad of electronic equipment which have microcontrollers to help you complete tasks in a more efficient manner. Each different device has a specific ability to complete a specific task. Below are some of the most common devices that have microcontrollers which you would find in an office setting;

- Scanners
- Printers
- Laminators
- Calculators
- Projectors
- Binding machines
- Automatic staplers (in case you have a lot of stapling to do)

Kitchen

Imagine having to wake up, go outside, collect firewood, start a fire, and then prepare your breakfast. Also, imagine having to repeat the process for lunch and dinner. This would get annoying quite fast. Luckily, people realised this early on and created devices that are much more efficient at preparing and making food and drinks in the kitchen.

These days our kitchens are filled with appliances that make many cooking tasks much more efficient as well as tasks that we might not be able to accomplish, like preserving food.

Common kitchen appliances include;

- Microwaves
- Ovens
- Toasters
- Refrigerators
- Coffee machines
- Blenders

Housekeeping

There are many mundane tasks that we need to carry out around the house. Tasks like washing the clothes, drying the clothes, washing the dishes, cleaning the floor, cleaning the toilet, and the list goes on. Without sounding like a broken record, we are blessed these days with electronic devices that help make these mundane tasks, well, less mundane!

Devices such as;

- Washing machines
- Dryers
- Dishwashers

Vacuum cleaners

Safety

Unfortunately, the world is filled with many dangers. But, we cannot live our lives scared all the time of every danger that exists. This is not what living should be. However, having systems in place to ensure that we are aware of dangers, or when a danger is imminent, is the best way to avoid injury or death. Since we are limited by our sensing abilities, we can harness the power of the microcontroller to help us stay one step ahead of the game. Technologies such as;

- Fire alarms
- Smoke alarms
- Security systems
    - PIR sensors
- Gas monitoring systems

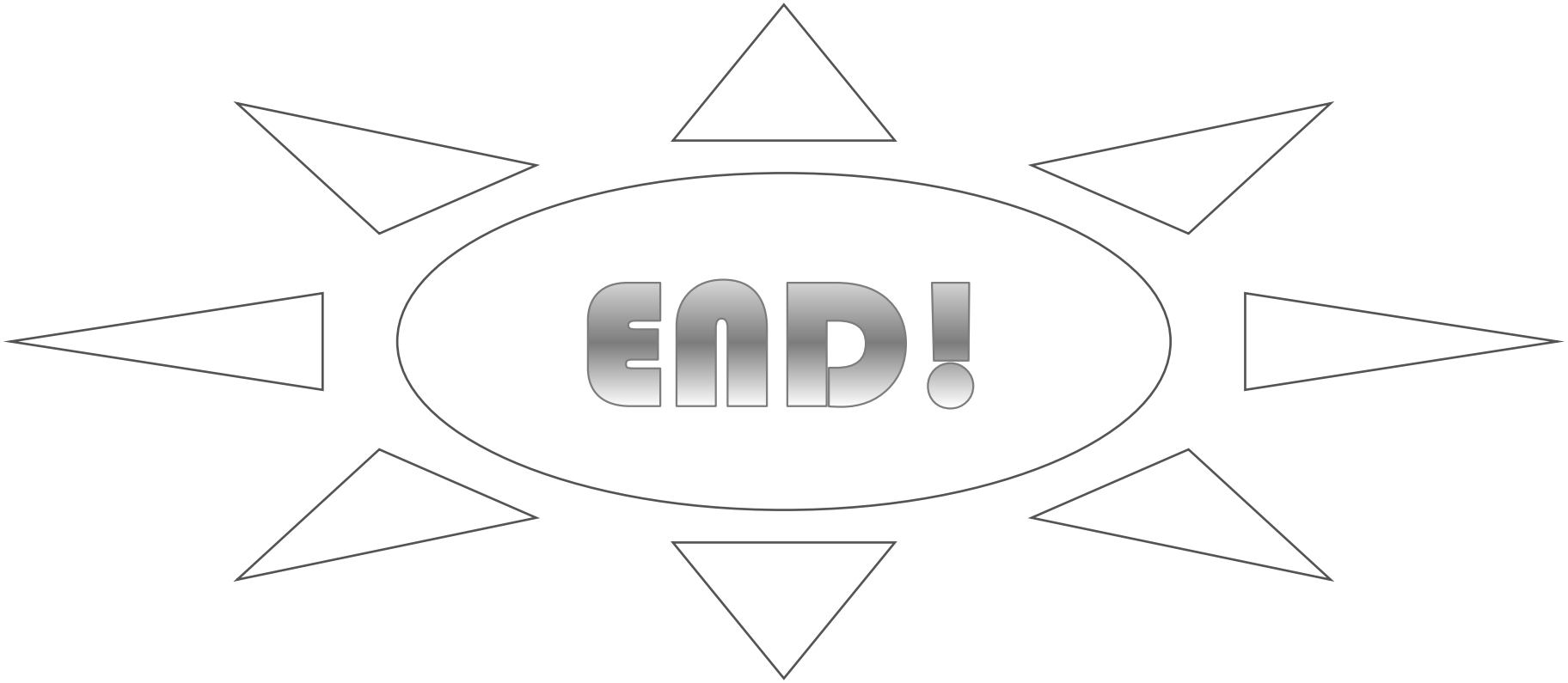
Transportation

Whether you need to go to your local grocery store, across the ocean to another country, and now even space, you can do so thanks to the advancement in ***Transportation*** Microcontrollers are used extensively within sub-systems of cars, motorbikes, boats, ships, trains, aero planes, spaceships, for a variety of different functions (safety, guidance, monitoring, etc). Below are some subsystems for the different transportations, that use microcontrollers;

- Guidance
  - GPS
- Safety
  - Airbags
  - Automatic braking system (ABS)

Transportation (Contd.)

- Monitoring
  - Temperature levels
  - Oil levels
  - Speed
  - Distance
  - Acceleration
- Entertainment
  - Radios
  - CD players
  - Televisions
- Communication
  - Bluetooth
  - Wi-Fi
  - Radio


END!