Chapter 1 Notes

1.1 Introduction to Embedded Systems

Embedded Systems are products designed with computational intelligence and interfaces to the physical environment to perform specific functions. These products are designed to meet a set of specified requirements derived to provide solutions to real world problems. The students of engineering irrespective of their branch specialization are presently devoting a significant amount of time to the pursuit of this study. A basic knowledge of embedded systems will motivate engineers towards the innovation of intelligent products that are hard to duplicate. This subject is spurning young engineers to create elegant solutions to complex problems faced in the real world.

The computational intelligence in embedded systems refers to the use of microprocessors, microcontrollers or programmable FPGA devices that form the heart of embedded systems. Embedded engineers learn the art of interconnecting these intelligent devices along with discrete devices and sensors to form the hardware part of the embedded system. The intelligence for the embedded system is the software program that is loaded into the memory of the microprocessor peripherals. Embedded system engineers need to have a sound understanding of the hardware circuit design and programming fundamentals to create such a product. A hand held cell phone is a typical example of an embedded system that has a central microcontroller and peripheral devices such as a keypad, display and audio speakers.

Embedded systems have the following characteristics that differentiate them from other systems.
• **Specific Function:** A single specific function is performed by the embedded system in a repetitive manner. For example, a digital camera is an embedded system that is designed to take pictures. Even though the camera can be upgraded with newer software, the application for the digital camera will be to perform the specific function of taking pictures.
• **Constrained System:** The design for embedded systems will need to meet stringent requirements that make the application commercially viable. These applications need to be designed keeping in mind the performance, size and commercial aspects of the product.
• **Real time systems:** Embedded systems are also called real-time interactive systems since they need to perform in an environment within specific time constraints. For example an air-bag system in an automobile needs to be deployed within a few milliseconds of a head-on collision in order to save the occupants of the vehicle. An unexpected delay in the system behavior during such a real-time event is not acceptable.
• **Reliability:** The most important characteristic of an embedded system is its reliability. The dedicated programs running in these systems make them perform reliably in medical applications such as a pacemaker that is implanted next to the heart to provide electrical stimulations.
• **Complex Algorithms:** Sophisticated calculations are performed such as digital filtering of noisy signals, mathematic calculations such as interpolation and integration in short time intervals.

Simple Embedded Systems
The majority of the embedded systems designed are used for specific purposes where the user need not be trained extensively to understand the system. An example for a simple embedded system is a process indicator which displays temperature in degree Celsius or degree Fahrenheit. The features of this system are described below.
1. LCD indicator that displays the temperature.
2. ON-OFF switch to turn-on or turn-off the indicator.
3. One slide-switch to select the units to be either in degree Celsius or degree Fahrenheit.
4. Battery compartment to fi t a standard size battery.

A simple embedded system could have one microcontroller, a few buttons and a display and is specifically used to perform simple functions. Here are a few examples of such simple systems.
Digital clock – Has a microcontroller with a digital seven-segment display that has a few buttons to change the time.
Microwave oven – The microcontroller accepts keypad inputs and controls the microwave source and a motor. A display is included to provide a visual status indication.
Calculator – A few keys interfaced to a microcontroller allows arithmetic calculations to be performed and displayed in an LCD Simple embedded systems are usually designed by small teams or in some cases by a single individual. The following steps are required to design a simple embedded system.

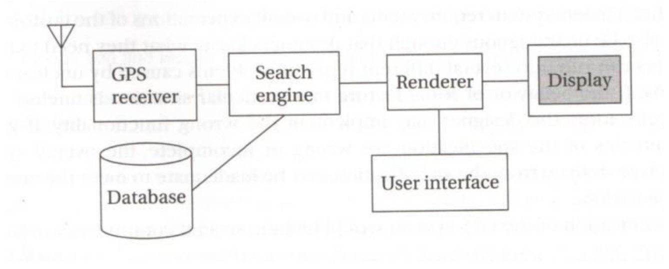
- Describe the set of requirements for the application
- Design a schematic diagram and printed circuit board that usually includes a microcontroller or microprocessor.
- Design and develop the software program that needs to reside in the memory of the microcontroller.
- Assemble the hardware, program the microcontroller and test the system.

The mechanical package that hosts the embedded system is usually not designed by the embedded system engineer. In some cases like the process indicator, all the mechanical components are purchased and the electronic circuit board is fitted into the enclosure to complete the embedded system.

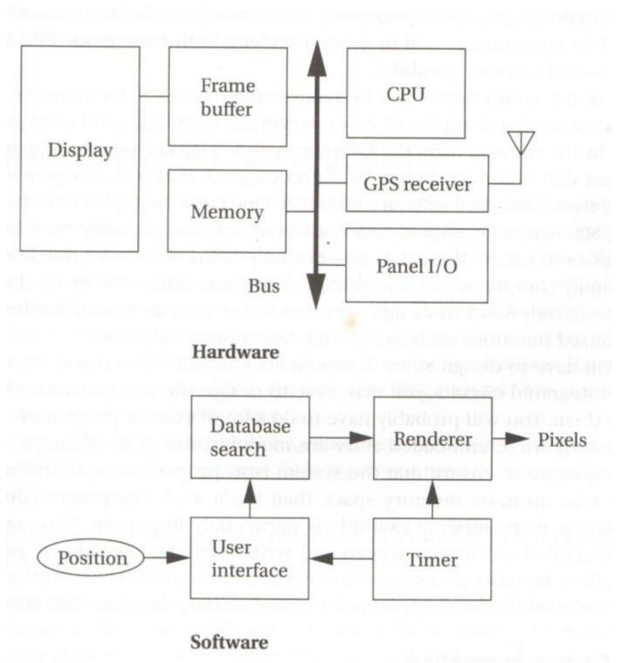
Complex Embedded Systems
There are quite a few applications where the requirements of the embedded system would itself run into several pages. Moreover the interconnectivity between the various functional blocks may change in real-time. Such systems may be interconnected by sub-systems comprised of simple embedded systems. A typical embedded system in an automotive application is a classic example of a complex embedded system.
The modern car has a few of the following sub-systems with each one having a microcontroller. Some of the modern cars have more than fifty embedded sub-systems working in unison, communicating among one another. Here is a partial list of embedded systems in a modern car with a brief description of its functionality.
a.)Air-conditioning system: The temperature of the car needs to be maintained at a set temperature irrespective of the temperature outside the car.
b.)Audio entertainment: CD changer, MP3 format digital audio player **(Multi-rate).** Audio and video portions stream at different rates and need to be synchronized.
c.)Cruise control: Maintain the speed of the car irrespective of the topography of the road with safety features
d.)Electronic Power steering: The steering control inputs are translated to wheel actuator output using an electric motor powered hydraulic system.
e.)Air-Bag control: In the event of a head-on collision, an air-bag is deployed in a fraction of a second to protect the occupants of the car by cushioning their impact.
f.)Ant-lock braking system: When the brakes are applied to the car on a wet road surface, the wheel lock conditions are detected in advance, and the braking pressure is adjusted to prevent the locking condition.
g.)The engine control unit called the ECU usually has a powerful microprocessor based system that has the ability to communicate with each of the subsystems and perform the following functions.
Control each subsystem by issuing commands over a communication network interface (CAN network)
Monitor the status of each subsystem by reading the system parameters and storing the data in memory.

• Communicate this data to a visual display called the cluster that shows the various critical parameters that the driver needs to know such as the vehicle speed, gasoline level and engine temperature.
• Optimize the functioning of the entire automobile to provide the best efficiency and performance

Hardware and Software Components: The embedded architecture of a moving map design is shown below. The architecture describes the system requirements of searching into a topographical database and render the map to a display while the system itself is moving since geographical position data is also received. User is also allowed to interact with the map system.



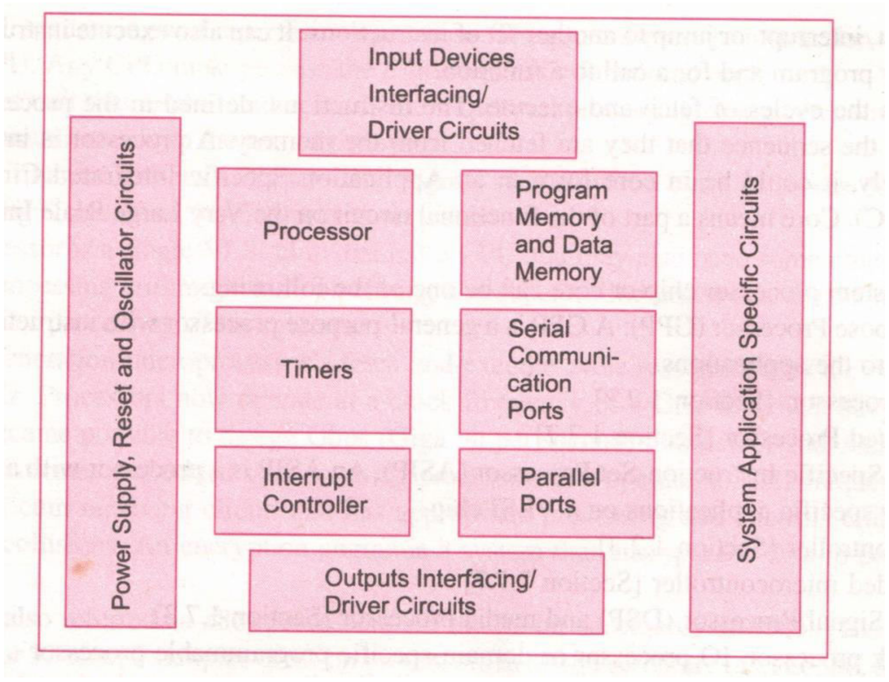
Hardware and software components can now be separated once we understand the system architecture. Block diagrams of typical software and hardware architecture are shown below.



Hardware: CPU with memory and I/O devices, different types of memory – frame buffer for map image and memory for CPU program, data.
Software: User interface, search for database to get geographical information, rendering algorithms to display the data, position retrieval from GPS device, timer functions to perform the operations synchronized to time.

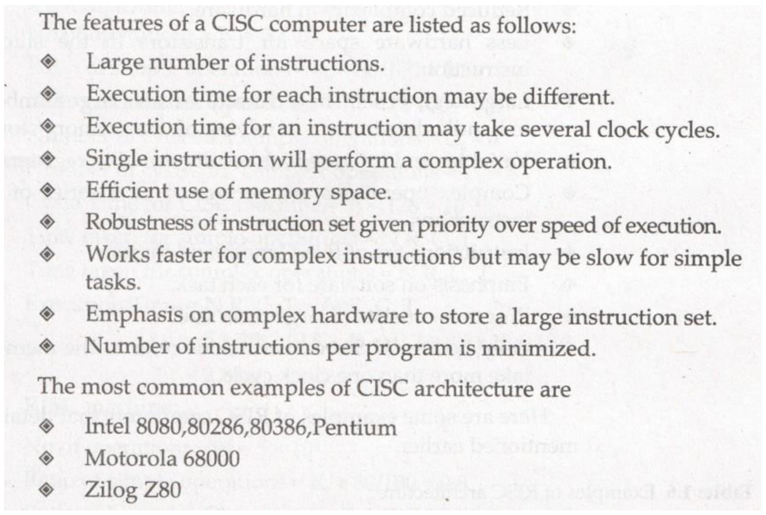
1.2 **Hardware Components of Embedded System**

Embedded System Hardware


Input Devices Interfacing/ Driver Circuits
Power Supply, Reset and Oscillator Circuits
Processor
Program Memory and Data Memory
Timers
Serial Communi-cation Ports
Interrupt Controller
Parallel Ports
Outputs Interfacing/ Driver Circuits
System Application Specific Circuits

Processor architectures: Complex Instruction Set architecture and Reduced Instruction Set Architecture

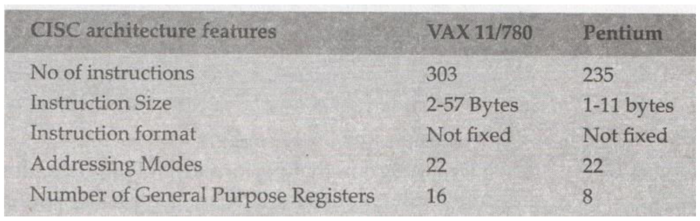
CISC Architecture – Complex Instruction Set


The features of a CISC computer are listed as follows:

- Large number of instructions.
- Execution time for each instruction may be different.
- Execution time for an instruction may take several clock cycles.
- Single instruction will perform a complex operation.
- Efficient use of memory space.
- Robustness of instruction set given priority over speed of execution.
- Works faster for complex instructions but may be slow for simple tasks.
- Emphasis on complex hardware to store a large instruction set.
- Number of instructions per program is minimized.

The most common examples of CISC architecture are

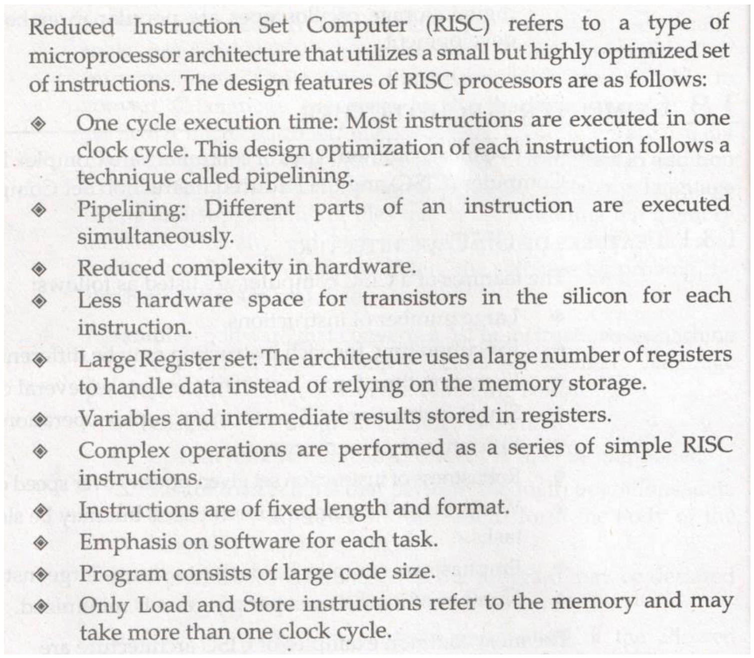
- Intel 8080,80286,80386,Pentium
- Motorola 68000
- Zilog Z80



CISC architecture features	VAX 11/780	Pentium
No of instructions	303	235
Instruction Size	2-57 Bytes	1-11 bytes
Instruction format	Not fixed	Not fixed
Addressing Modes	22	22
Number of General Purpose Registers	16	8

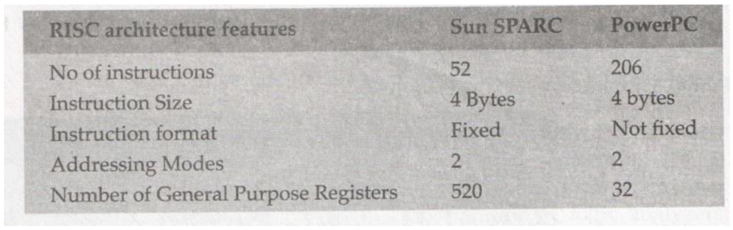
CISC Architecture Examples

RISC Architecture: Reduced Instruction Set Architecture



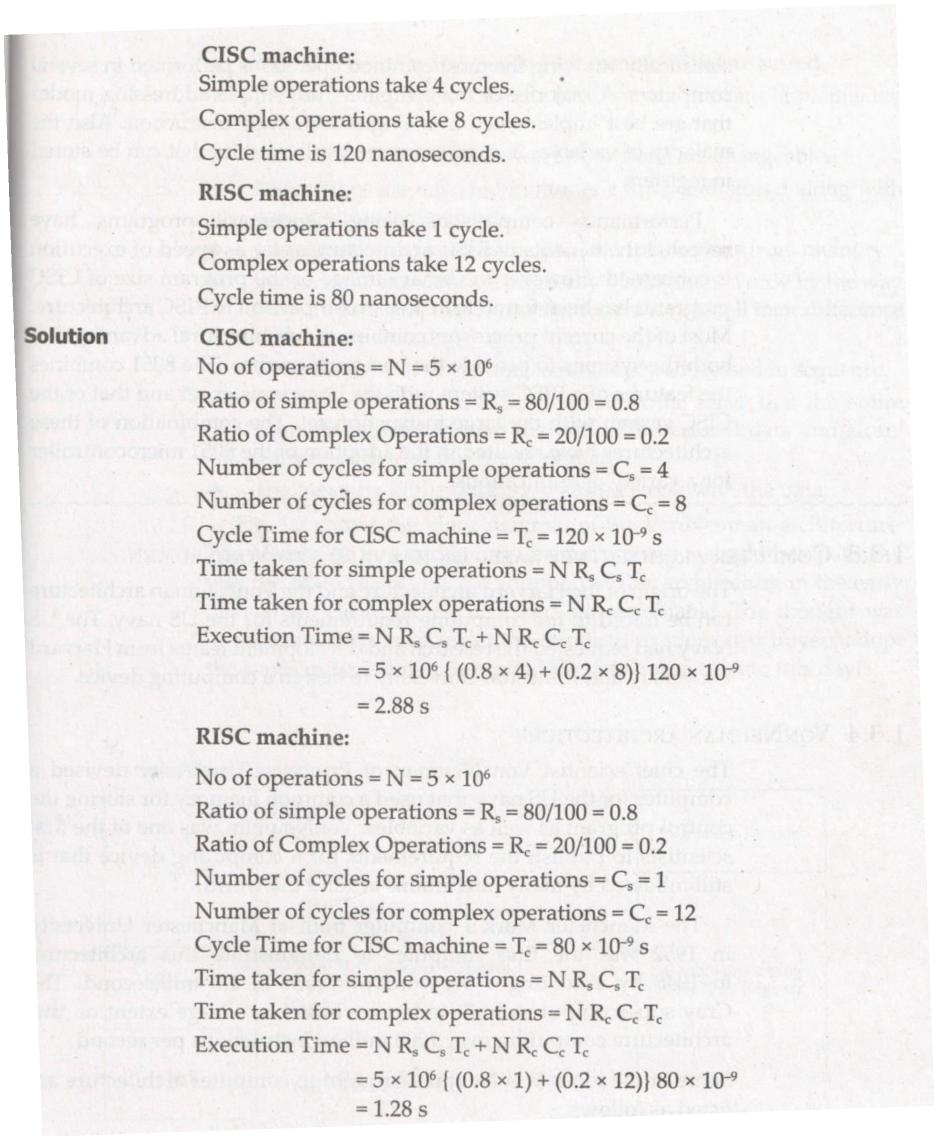
Reduced Instruction Set Computer (RISC) refers to a type of microprocessor architecture that utilizes a small but highly optimized set of instructions. The design features of RISC processors are as follows:

- One cycle execution time: Most instructions are executed in one clock cycle. This design optimization of each instruction follows a technique called pipelining.
- Pipelining: Different parts of an instruction are executed simultaneously.
- Reduced complexity in hardware.
- Less hardware space for transistors in the silicon for each instruction.
- Large Register set: The architecture uses a large number of registers to handle data instead of relying on the memory storage.
- Variables and intermediate results stored in registers.
- Complex operations are performed as a series of simple RISC instructions.
- Instructions are of fixed length and format.
- Emphasis on software for each task.
- Program consists of large code size.
- Only Load and Store instructions refer to the memory and may take more than one clock cycle.



RISC architecture features	Sun SPARC	PowerPC
No of instructions	52	206
Instruction Size	4 Bytes	4 bytes
Instruction format	Fixed	Not fixed
Addressing Modes	2	2
Number of General Purpose Registers	520	32

Example: Consider a program that performs 5 million operations: 80% of which are simple operations and 20% are complex operations. Compare the execution time of the program given the following data.



CISC machine:
Simple operations take 4 cycles.
Complex operations take 8 cycles.
Cycle time is 120 nanoseconds.

RISC machine:
Simple operations take 1 cycle.
Complex operations take 12 cycles.
Cycle time is 80 nanoseconds.

Solution **CISC machine:**
No of operations = N = 5×10^6
Ratio of simple operations = $R_s = 80/100 = 0.8$
Ratio of Complex Operations = $R_c = 20/100 = 0.2$
Number of cycles for simple operations = $C_s = 4$
Number of cycles for complex operations = $C_c = 8$
Cycle Time for CISC machine = $T_c = 120 \times 10^{-9}$ s
Time taken for simple operations = $N\,R_s\,C_s\,T_c$
Time taken for complex operations = $N\,R_c\,C_c\,T_c$
Execution Time = $N\,R_s\,C_s\,T_c + N\,R_c\,C_c\,T_c$
$= 5 \times 10^6 \{(0.8 \times 4) + (0.2 \times 8)\}\, 120 \times 10^{-9}$
$= 2.88$ s

RISC machine:
No of operations = N = 5×10^6
Ratio of simple operations = $R_s = 80/100 = 0.8$
Ratio of Complex Operations = $R_c = 20/100 = 0.2$
Number of cycles for simple operations = $C_s = 1$
Number of cycles for complex operations = $C_c = 12$
Cycle Time for CISC machine = $T_c = 80 \times 10^{-9}$ s
Time taken for simple operations = $N\,R_s\,C_s\,T_c$
Time taken for complex operations = $N\,R_c\,C_c\,T_c$
Execution Time = $N\,R_s\,C_s\,T_c + N\,R_c\,C_c\,T_c$
$= 5 \times 10^6 \{(0.8 \times 1) + (0.2 \times 12)\}\, 80 \times 10^{-9}$
$= 1.28$ s

RISC architecture has been developed to improve the CPU power with a simpler instruction set. The architecture has been developed after statistically studying the most common operations performed on several computers. A majority of the computers use simple addressing modes that are best implemented in a single clock cycle instruction. Also the majority of the variables are local variables that can be stored in registers. The speed of RISC architectures are fast but the binary image of the program memory for CISC architectures are smaller. Processors are designed with a combination of the same these days.

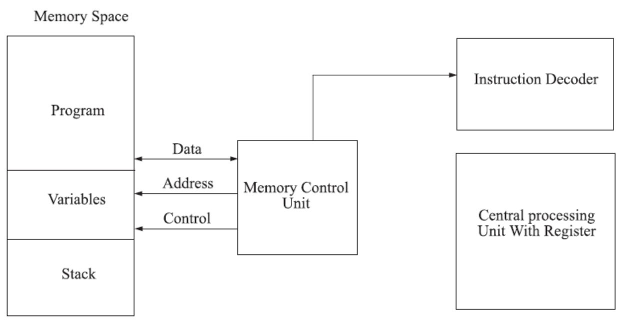
VonNeuman architecture

The chief scientist von Neuman of Princeton University devised a computer for the US navy that used a common memory for storing the control program as well as variables. VonNeuman was one of the first scientists to publish the requirements for a computing device that is still followed by many institutions around the world. The Mancheter Mark 1 computer built at Manchester University in 1952 was the first computer to demonstrate this architecture in 1948 executing a 96 word program in 1.2millisecond. The Cray supercomputers built today are based to a large extent on this architecture computing over 1000 million instructions per second. Some of the ideas von Neuman brought to computer architecture are listed as follows.

- There are four main parts for the computing device: Arithmetic logic unit, memory unit, control unit and Input/Output units.
- Data and intermediate results of operation need to be stored in memory.
- Instructions or orders needed for computation need to be stored.
- In special machines, computational procedures should be implemented in hardware.
- In general purpose machines, instructions could be changeable.
- Instructions should be encoded in numeric form and stored along with data in the same memory.
- The control unit would execute the encoded instructions from memory.
- Instructions and data are implicitly distinguished only by the way they are used and there is no significant way to tell their difference in the same stored memory.
- The memory is a single unit continuously addressed in sequence.
- All the memory is one-dimensional in the sense that the entire program will try to map the multi-dimensional data and store/retrieve data with this memory map.
- The meaning of the data is not stored along with the data.

Fig. below shows the block diagram of the vonNeuman architecture. The simplicity in accessing the memory using a single bus pathway was the main reason for the adoption of this architecture in the early days when electronic circuits were not as reliable.

The memory block shows the memory occupied by the program, variables and the program counter stack. Access to the program counter stack by the main program is an useful feature that is used to develop real-time operating systems.



Harvard Architecture

Harvard architecture refers to computers with physically separate memory for program storage and data variable storage. Some of the significant features of the Harvard architecture are listed as follows.

- The central processing unit can read and instruction and data at the same time.

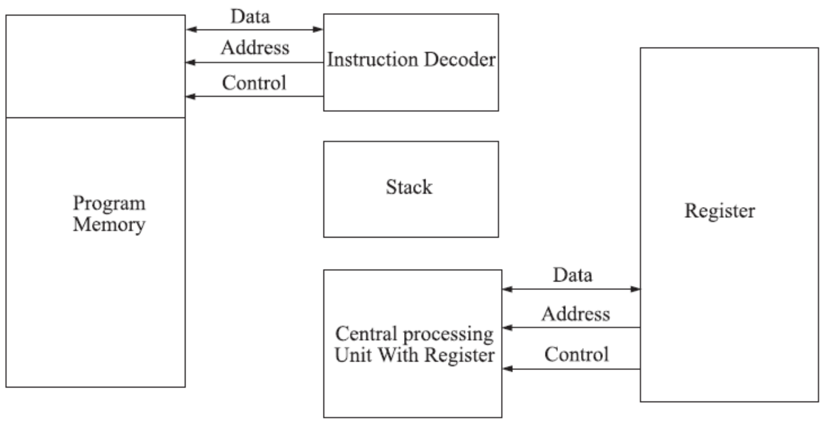
- The next instruction can be fetched from memory while the present instruction is being executed.
- Electronic hardware circuitry is more complex.
- Used in specialized DSPs for audio and video applications.
- Along with RISC architecture they contain small program and data memory

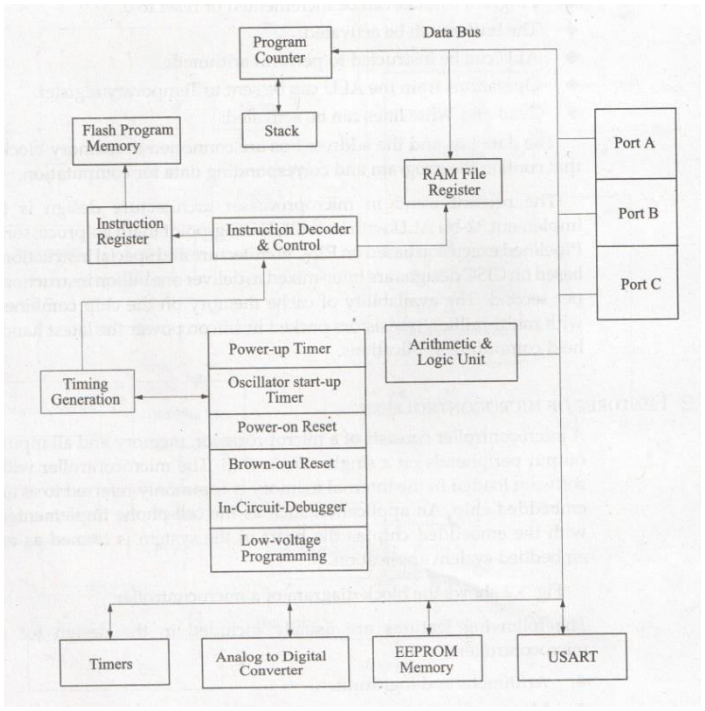
Some of the examples of microcontrollers that are implemented using the Harvard architecture are as follows:

- Blackfin processors from Analog Devices
- PIC microcontrollers from Microchip Technology

In order to implement efficient CPU designs, a combination of Harvard and von Neuman architecture is employed. The time required for memory access is the main limitation to be overcome in processor designs. A small amount of fast but expensive memory called cache is implemented in new designs. The cache memory is located on the processor chip and is segmented into instruction cache and data cache. Harvard architecture is employed to access the cache memory. The cache memory being of limited size, the main memory is accessed using von Neuman architecture. The main memory is however not segmented.

Fig. below shows the block diagram of the Harvard architecture with the multiple pathways that allows an instruction to pre-fetch the next instruction while performing the execution of the present instruction. For execution of time-critical systems, this architecture has the obvious advantages and hence is widely adopted in present-day microcontrollers.





Block Diagram of a Microcontroller

Memory Types: OTPROM, EPROM, EEPROM, RAM (SRAM,DRAM), Flash memory

<u>Memory organization in a typical microcontroller (8051)</u>

The 8051 devices have separate address spaces for program and data memory. This allows the 8-bit CPU to generate 8-bit addresses for accessing data quickly. The 8051 architecture allows program memory to be 64Kbytes long. In the basic version of the 80C51, program memory inside the chip is 4Kbytes long. In this version of the chip, the internal memory is implemented in read-only-memory or ROM. The version of the chip that does not have internal memory is called the ROMless version. In this case, all the program memory is implemented outside the chip according to the requirement of the application. This memory is usually implemented using EPROM (Electrically Programmable Read Only memory).
The 8051 device has control pins such as PSEN (Program Store Enable) and EA (External access) that determine whether external memory is accessed or internal memory is accessed. If the control pin EA is connected to VCC (usually 5V), then the program memory accessed by the chip is internal memory. The range of memory for internal on-chip ROM is from 0000 to 0FFFh. Program memory beyond this range from 1000h to FFFFh is however directed to external memory. If the EA pin is connected to VSS (usually 0V or Ground), then the 8051 can access external memory starting from address 0000Hex.

Various combinations of memory types may be used. It is possible to have 4K of code memory on-chip and the remaining memory off-chip in an EPROM. When the program is stored on-chip the program memory size is usually limited to 2K, 4K or 8K based on the version of the chip that is being used. Each version offers specific capabilities and one of the distinguishing factors from chip to chip is how much space the ROM or EPROM occupies.

Data memory occupies a separate memory outside the program memory area. Data memory is implemented using Static Random Access Memory (SRAM) memory. In the 80C51, the lowest 128 bytes of data memory are on-chip. The upper 128 bytes are reserved for special function registers that control the various functions of the 8051.

External RAM

This refers to Random Access Memory found off-chip. This memory has slower access time than on-chip RAM. While internal RAM is limited to 128 bytes, the 8051 can access 64K external RAM.

- P0 and P2 are used as input/output ports.
- P0 and P2 lines are used for address/ data bus activity.

External data access uses control signals such as RD, WR. Since the address information and data information are multiplexed using the P0 and P2 lines, a means to separate the address information and data information is required. The address latch enable (ALE) signal is a short pulse that is active when the address information is present on the 16 lines that comprises P0 and P2. The ALE signal is used as a control signal to a latch that has the multiplexed address/data lines as its input. The output of the latch is the address information that will result when the ALE pulse causes the address information to be transmitted to the output. The address information is thus held at the output of the latch until a complete instruction cycle is completed. The RD and WR lines generated by the microcontroller are used for output enable (OE) and write enable (WE) control for the external device.

The section on CPU timing lays the foundation for the sequence of operations and the associated timing for an external memory access.

On-Chip Memory

The 8051 includes on-chip memory of two types: Internal SRAM and Special Function Register (SFR) memory. There are three spaces of memory partitioned as described below.

- Lower 128 bytes- These memory spaces are addressed by direct or indirect addressing methods.
- Upper 128 bytes- These memory spaces are accessible by indirect addressing modes only (Available only in some devices)
- SFR space- These memory spaces are accessible by direct addressing modes only.

Hence there are 384 bytes of internal memory that can be addressed by the 8051 architecture.

<u>**Register Banks**</u>

The 128 bytes of internal Ram are subdivided into banks as shown on the memory map.

- Addresses (00h - 07h) are "Register Bank 0".
- Addresses (08h – 0Fh) are "Register Bank 1".
- Addresses (10h - 17h) are "Register Bank 2".
- Addresses (18h – 1Fh) are "Register Bank 3".

These register banks contain registers R0 to R7. A program may choose any one of the four register banks for normal operation. The register banks are selected by the RS0 and RS1 bit in the PSW register. User software can write to these bits to select the required register bank.
The 8051 microcontroller uses registers in many of its instructions. These "R" registers are numbered from 0 through 7 (R0, R1, R2, R3, R4, R5, R6, and R7). These registers are generally used to assist in manipulating values and moving data from one memory location to another.

When the microcontroller is first powered up, register bank 0 is used by default. Registers R0-R7 is mapped to addresses 00h through 07h. However, your program may instruct the 8051 to use one of the alternate register banks; i.e., register banks 1, 2, or 3. In this case, R3 will no longer be the same as Internal RAM address 03h. For example, if your program instructs the 8051 to use register bank 3, register R3 will now be synonymous with Internal RAM address 1Bh. Register banks really reside in the first 32 bytes of Internal RAM. If only the first register bank (i.e. bank 0) is used, the Internal RAM locations 08h through 1Fh may be used without caution. But if register banks 1, 2, or 3 are used great care must be exercised in using addresses below 20h since there is a possibility that the "R" registers may be overwritten.

Bit Memory

The program may also use memory locations one bit at a time. The bit memory resides from addresses 20h through 2Fh. The 8051gives the user the ability to access a number of bit variables. These variables may take the value of either 1 or 0. There are 128 bit variables available to the user, numbered 00h through 7Fh. The user may make use of these variables with commands such as SETB and CLR.

- Bit memory is part of internal RAM.
- The 128 bit variables occupy the 16 bytes of Internal RAM from 20h through 2Fh.
- Writing FFh to internal RAM address 20h implies setting the bits 00h to 07h to 1.
- Avoid using 20h to 2Fh as byte variables if your program needs to use bit variables.
- Bit variables 00h through 7Fh are meant for user-defined programs.
- Bit variables 80h and above are used to access certain SFRs.

Stack memory

The stack refers to an area of internal RAM memory that is used to store and retrieve data on a temporary basis. The 8051 also uses this area to store return addresses when jumping to user subroutines or interrupt service routines**.** The 8051 has an 8-bit register called the stack pointer that is used to point to the top of the stack of data. On power up, the Stack Pointer (SP) is initialized to 07h. This means that the stack will start at address 07h and expand upwards. If you will be using the alternate register banks (banks 1, 2 or 3) you must initialize the stack pointer to an address above the highest register bank you will be using, otherwise the stack will overwrite your alternate register banks. Similarly, if you will be using bit variables it is usually a good idea to initialize the stack pointer to some value greater than 2Fh to guarantee that your bit variables are protected from the stack. It is normal practice to locate the Stack pointer SP at an address of 30h in most assembly language programs.

The stack pointer SP holds the RAM address of the last byte of data stored in the stack. In order to store data, stack instructions result in first incrementing the stack pointer before moving the data to the stack. Similarly when data is retrieved from the stack, the data is first moved out and then the stack pointer is decreased. Thus the SP register always points to the top of the stack heap.

It is left to the programmer to use the stack memory with care and not allow the stack to overwrite user variables.

User variable RAM

- Addresses 30h-7Fh may be used by user variables.
- This RAM memory is used for fast arithmetic computations.
- This memory is also used for stack memory storage area.

The stack memory is temporary storage area for variables that need to be stored when the microcontroller executes several functions in parallel but not necessarily in sequence.

The stack memory space in combination with the user variable memory space is limited to 80 bytes.

Special Function Register (SFR) Memory

Special Function Registers (SFRs) are areas of memory that control specific functionality of the 8051 processor. For example, four SFRs permit access to the 8051's 32 input/ output lines. Another SFR allows a program to read or write to the 8051's serial port. Other SFRs allow the user to set the serial baud rate, control and access timers, and configure the 8051's interrupt system.

When programming, SFRs have the appearance of being internal memory. When accessing SFR memory, any instruction that accesses an address of 00h through 7Fh refers to an Internal RAM memory address; any instruction with an address of 80h through FFh refers to an SFR control register.

Table below lists the symbols, names and addresses of the 8051 SFRs. Some of the SFRs are available on an enhanced version of the chip called the 8052.

Symbol	Name	Address (Hex)	Remarks
ACC	Accumulator	0E0	Bit addressable
B	B Register	0F0	Bit addressable
PSW	Program Status Word	0D0	Bit addressable
SP	Stack Pointer	81	
DPTR	Data Pointer 2 Bytes		
DPL	Low Byte	82	
DPH	High Byte	83	
P0	Port 0	80	Bit addressable
P1	Port 1	90	Bit addressable
P2	Port 2	0A0	Bit addressable
P3	Port 3	0B0	Bit addressable
IP	Interrupt Priority Control	0B8	Bit addressable
IE	Interrupt Enable Control	0A8	Bit addressable
TMOD	Timer/Counter Mode Control	89	
TCON	Timer/ Counter Control	88	Bit addressable
T2CON	Timer/ Counter 2 Control	0C8	Bit addressable, 8052 only

TH0	Timer/ Counter 0 High Byte	8C	
TL0	Timer/ Counter 0 Low Byte	8A	
TH1	Timer/ Counter 1 High Byte	8D	
TL1	Timer/ Counter 1 Low Byte	8B	
TH2	Timer/ Counter 2 High Byte	0CD	8052 only
TL2	Timer/Counter 2 Low Byte	0CC	8052 only
RCAP2H	T/C 2 Capture Reg. High Byte	0CB	8052 only
RCAP2L	T/C 2 Capture Reg. Low Byte	0CA	8052 only
SCON	Serial Control	98	Bit addressable
SBUF	Serial Data Buffer	99	
PCON	Power Control	87	

Description of Special Function Registers

The 8051 architecture allocated the addresses 80h to FFh for SFRs. In 80C51 in particular, there are only 21 SFRs out of the possible 128 possible addresses. The remaining addresses are reserved and writing to them or reading from them will produce unpredictable results.

Cache memory
The microcontroller uses a block of memory located between the main program memory and the core. This memory called cache memory is used to speed the instruction execution timing reducing the time for fetching from the main memory. Cache memory units can either contain data units and instruction units in a single block or be featured separately with independent bus access. The cache memory increases the overall performance by speeding up the instruction cycle. The cache memory usage does not guarantee predictable instruction cycle time but guarantees faster cycle time. The instruction cycle time may vary based on the cache usage.

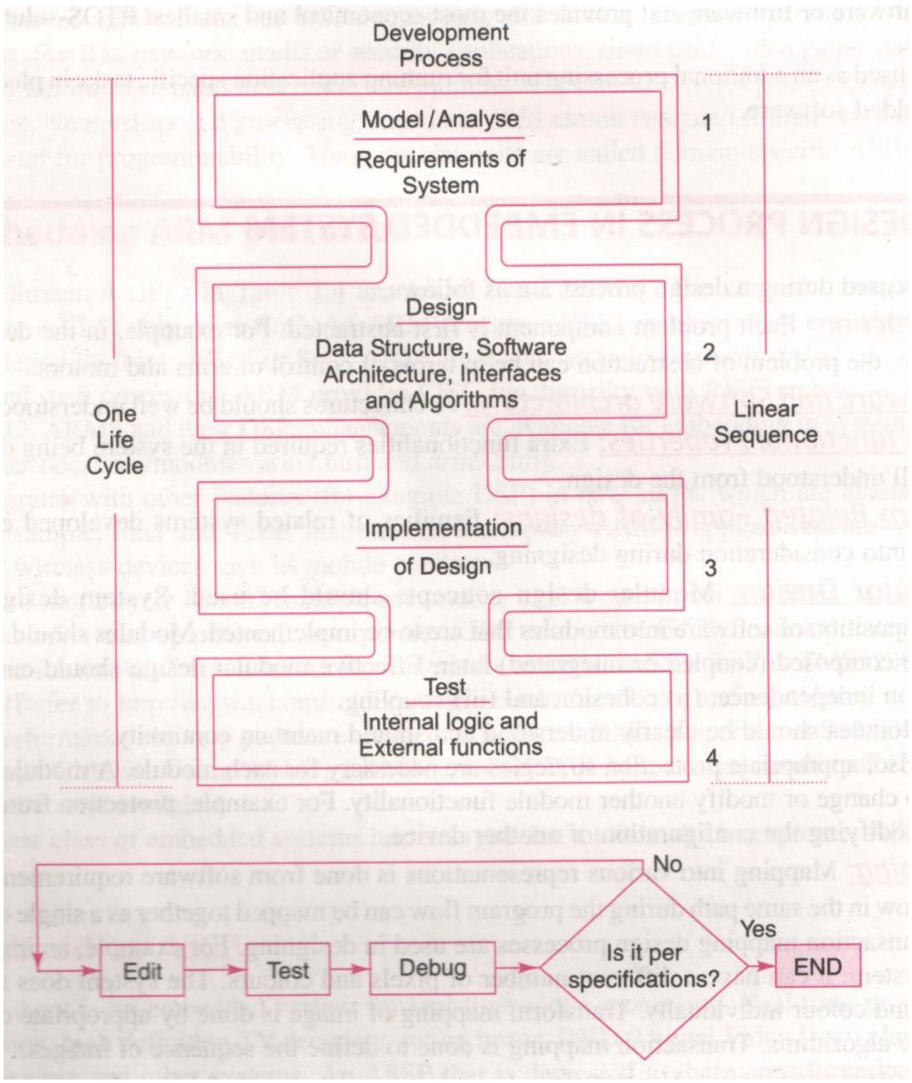
Interrupts
Interrupts are branches in code execution triggered by hardware events or software events. There are two external event interrupt sources featured by pins INTO and INT1 in 8051 controllers. Once configured in software, a high to low transition in one of these pins will cause the program execution to branch to a specific location called the interrupt vector address. The program counter will point to this interrupt vector address and execute what is called an interrupt subroutine. After execution of the interrupt subroutine, the Program counter points to the address from where it branched off before the interrupt occurred and continues program execution.

Peripherals
The 8051 controller communicates with the outside world through peripherals that perform input/output functions. A few of the peripherals found in 8051 controllers are listed below.

- Input/output ports
- Timer/Counter
- Analog to Digital Converter
- Serial Port
- Interrupt controller

1.3 **Software Components of Embedded Systems**

Activities for Software Design Cycle in Embedded Systems Development



RTOS Basics

A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time application process data as it comes in, typically without buffering delays. Key factors in an RTOS are minimal interrupt latency and minimal thread switching latency. An RTOS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

For embedded devices, the *general* rule is that an RTOS is used when the application needs to do more than a few simple actions. An RTOS allows an application to be structured in a manner that scales as more application/system features are added (e.g. communication stacks, power management, etc.).

A RTOS has the following goals

- Small latency: Minimum delay time from one task to another.
- Determinism: Time taken for execution of a task or thread needs to be predictable so that deadlines are known.
- Structured Software: Software tasks can be organized, added or deleted based on the system requirement.
- Scalability: RTOS must be able to scale from a simple application to a complex one with stacks, drivers, file systems.
- Offload development: An RTOS manages many aspects of the system which allows a developer to focus on their application. For example an RTOS, along with scheduling, generally handles power management, interrupt table management, memory management, exception handling.

RTOS Components

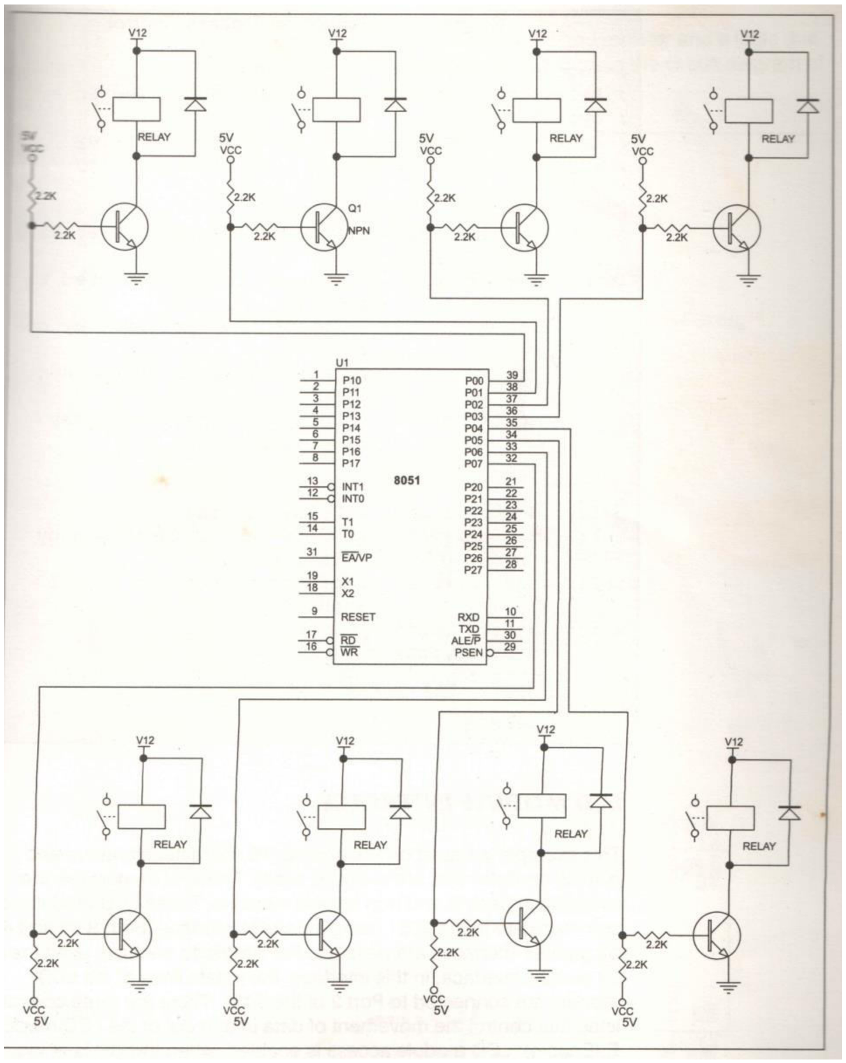
- Scheduler: Preemptive scheduler that guarantees the highest priority thread it running.
- Communication Mechanism: Semaphores, Message Queues, Queues, etc.
- Critical Region Mechanisms: Mutexes, Gates, Locks, etc.
- Timing Services: Clocks, Timers, etc.
- Power Management: For low power devices, power management is generally part of the RTOS since it knows the state of the device.
- Memory Management: Variable-size heaps, fixed-size heaps, etc.
- Peripheral Drivers: UART, SPI, I2C, etc.
- Protocol stacks: BLE, WiFi, etc.
- File System: FatFS, etc.
- Device Management: Exception Handling, Boot, etc.

SOC Design

A system on a chip (SoC) is an integrated circuit that integrates all components of a computer or other electronic system. They use a central processing unit (CPU), memory, input/output ports and secondary storage – all on a single substrate or microchip, the size of a coin. It may contain digital, analog, mixed-signal, and often radio frequency signal processing functions, depending on the application. As they are integrated on a single substrate, SoCs consume much less power and take up much less area than multi-chip designs with equivalent functionality. SoCs are common in the Smartphone applications. SOCs are common in embedded systems design and Internet of Things applications.

A typical SoC will typically integrate a CPU, graphics and memory interfaces, hard-disk and USB connectivity, wi-fi module, random-access and read-only memories and secondary storage on a single circuit die.

A simple example for Relay control



Assembly language example code

```
;Turn on relay connected to Port 0.0 ON and OFF with a short delay
        .org 0h    ;signifies start of assembly language routine
start:  mov P0,#00h
        acall delay
        mov P0,#01h ; Turn ON Relay1 connected to P0.0
        acall delay
        sjmp start

delay:  mov r0,#0FFh; Load the hex number FF to register r0
wait:   djnz  r0,wait
        ret
        .end  ; signifies end of assembly language routine
```

Review Questions (short)

1. Define an embedded system
2. Distinguish between microprocessor and microcontroller.
3. How is a DSP different from a microcontroller?
4. Give examples of microcontroller, microprocessor and DSP devices.
5. What is the purpose of watchdog timer?
6. What are the various types of memory used in embedded systems?
7. What is System On Chip?
8. What is meant by FPGA, PLA,CPLD?
9. Compare FPGA and microcontroller and explain the differences with an example.
10. Explain the functions of compiler, linker and loader in embedded systems.
11. Explain why RTOS is required in embedded systems.
12. What is meant by hard real time constraints? Explain with an example.
13. What is the difference between SRAM and DRAM?
14. Provide examples of simple embedded system and complex embedded systems.
15. How can power consumption in embedded systems be minimized?
16. What is cache memory?
17. Describe Harvard architecture with a block diagram.
18. Describe von Neumann architecture with a block diagram.

Review Questions (Detailed)

1. Describe the features of RISC and CISC architectures with examples.
2. Compare Harvard and von Neumann architecture explaining the features.
3. What are the various types of memory used in embedded systems?
4. Describe the various components and their functions used in the hardware design for an embedded system.
5. What are the various characteristics of embedded systems. Differentiate between simple and complex embedded systems.
6. Write short notes on ADC, Timers, DAC, I/O and interrupts in microcontrollers.
7. What are the software components tools used in developing embedded systems. Write short notes on the use of RTOS and tasks?
8. Describe the various steps involved in embedded system design
9. Describe the various steps involved in software design life cycle.
10. Explain with an example such as 8051 how Special function registers are implemented and describe a few example of how GPIO operations or Timer operations may use specific registers.
11. What is the purpose of RTOS. Describe a few features of how it assists in emdedded system design.
12. Write a short assembly language program to blink LEDs connected to Port P0 with a delay.
13. Write a short assembly language program to sequentially turn ON four relays connected to P1.0 to P1.3 with a delay one after another and keep them ON forever. Draw the circuit.