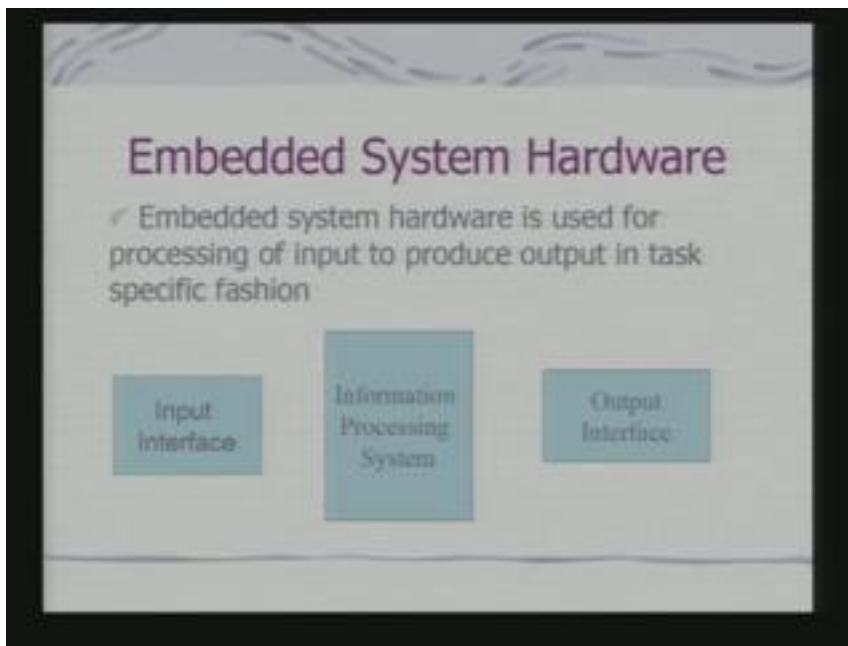


**Embedded Systems**  
Dr. Santanu Chaudhury  
Department of Electrical Engineering  
IIT Delhi  
Lecture 2  
Embedded Hardware

In the last class we had an overview of embedded systems, today we shall look at embedded hardware and in course of next few lectures we shall explore hardware of embedded systems in more detail. Embedded system hardware's basic task is to receive input process it and provide output.

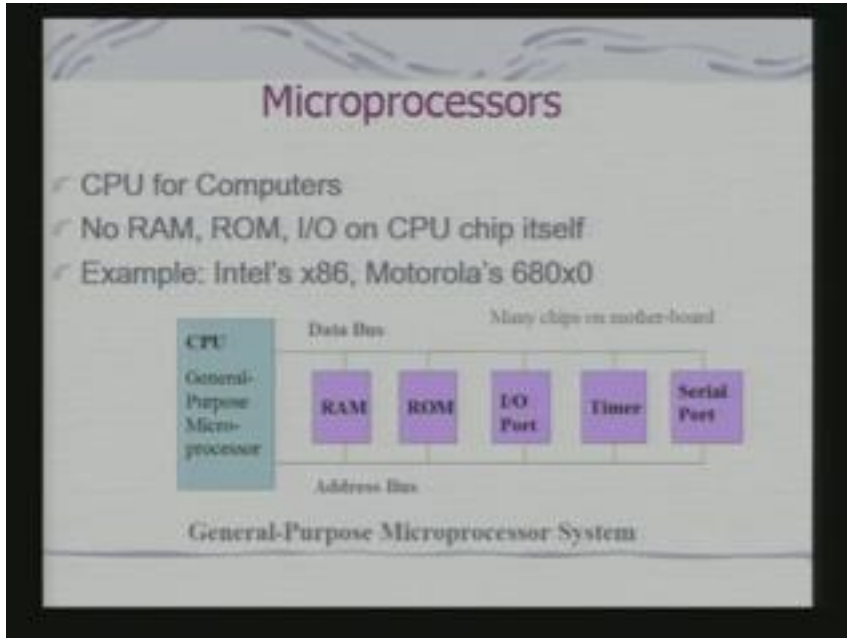
(Refer Slide Time 02:03 min)



So the basic hardware is built to meet the requirement of the information processing system of the embedded appliance. The information processing system basically it would consist of a processor and the other peripherals to maintain and manage input and output interfaces. Basically the processors are micro processors and micro-controllers, but in comparison to general purpose computing platforms. Here, the considerations are energy efficiency and I would like to have instruction sets which can provide high code density. Obviously the energy efficiency leads to enhanced battery life and less power consumption; high code density will lead to less requirement of program memory. We are

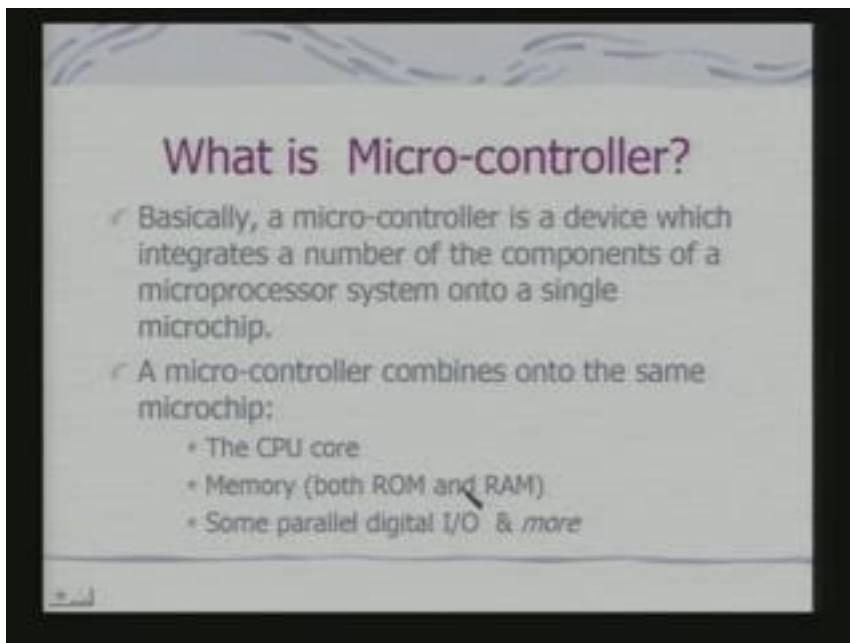
all familiar with microprocessors and microprocessors like Intel Pentium, is used as a CPU for the computers that we use.

(Refer Slide Time 03:05 min)



Typically a microprocessor does not have any RAM, ROM, I/O on the chip itself. The family of 8086 processors of Intel, of which Intel Pentium is an example, which we are using today in the PC's as well as Motorola 68000 series. These are all examples of this kind of general purpose microprocessors. And a typical computer is built around this microprocessor having these as architecture. And this is an example of a general purpose microprocessor system. Then what is a micro-controller? Basically a micro-controller is a device which integrates a number of the components of a microprocessor system onto a single microchip.

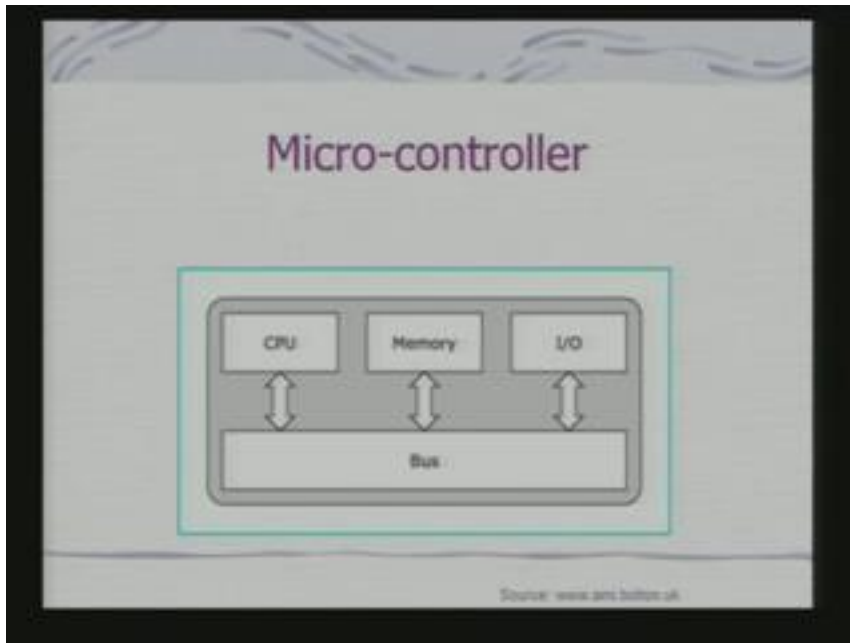
(Refer Slide Time 03:36 min)



A micro-controller on its core must have of the CPU memory and some parallel digital I/O and many other peripherals as well. So, let us look at a basic block diagram of a micro-controller. What we have got, we have got the CPU, we have got memory and we have got I/O all connected via bus. And all these things are integrated on to the same

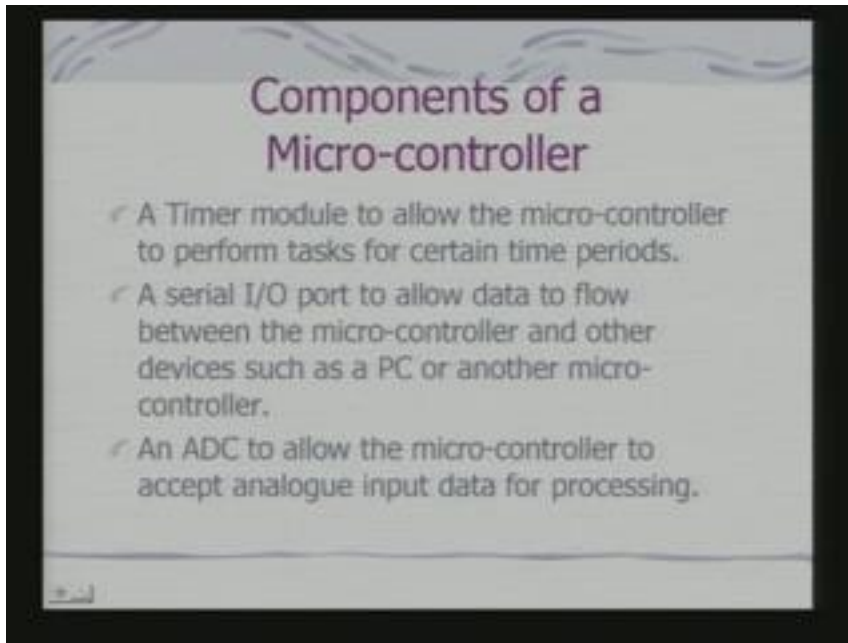
silicon real estate. So, they are the same chip itself. There are other components of micro-controller as well. Typically a timer module allows micro-controller to perform tasks for certain time periods.

(Refer Slide Time 04:17 min)



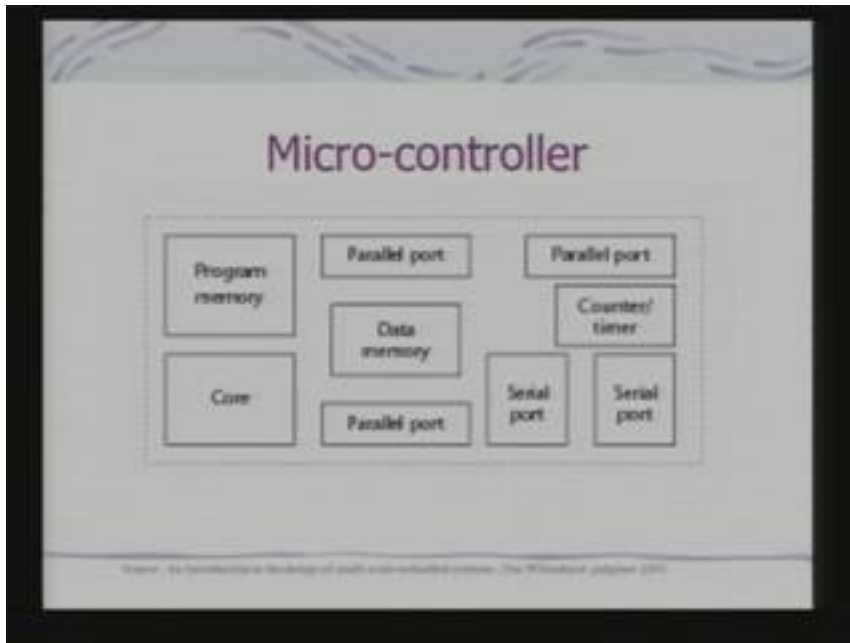
You know how the timer is typically used in a general purpose computing platform. A timer is programmed to work for a certain time period with respect to the system clock and after that typically a timer can generate an interrupt to the processor so that there can be switching from one task to another task. The similar function is performed by the timer on micro-controller used for embedded applications as well. We have on top of parallel I/O, serial I/O ports which can allow data flow between the micro-controller and devices which support serial interfaces. The serial ports that also used for interfacing with the PC during development environment and development life cycle of the embedded system. As well, when you need to use the embedded appliance with the PC to maybe read data or reprogram it the serial port is also used. Further, we have analogue to digital converters on micro-controllers to convert the external inputs which may come in the analogue form to the digital form for subsequent processing by the micro-controller.

(Refer Slide Time 04:44 min)



We also have in many cases DAC the digital to analogue converter to provide the output to the external world. So, now we have got a more detailed block diagram of a micro-controller. Here, you find you have got the core which is actually the processing core around which we have shown program memory where typically program is expected to be stored. I have shown data memory as a separate entity there are parallel ports; there are serial ports as well as counters and timers.

(Refer Slide Time 06:32 min)

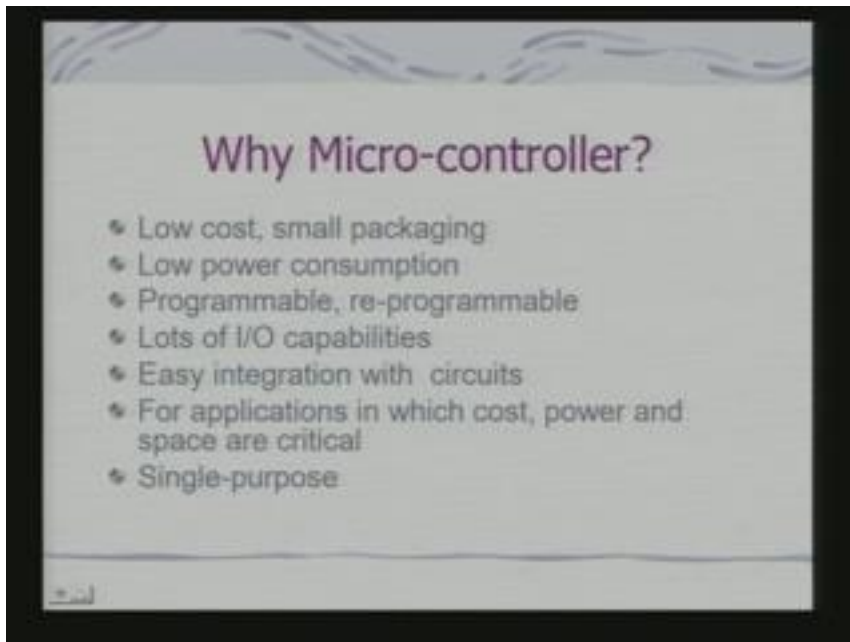


Now, why I have shown program memory and data memory separately? In many cases, my program memory can be actually a read only memory or an erasable PROM or even flash which is expected to be a persistent storage that means when an embedded system comes preprogrammed. The program port is stored in the program memory and which is not typically RAM. The data part of it is typically expected to be RAM, Random Access Memory and here it is not a persistent store. Otherwise also, you may find later on and we shall discuss later on; there are differences in all emaciation of program and data memory with respect to the processor core.

Parallel ports are primarily targeted for interfacing with the devices, serial port function I have already discussed, counters and timers to take care of different timing requirements for the embedded system. Now, the question is we have found the microprocessor we have used for general purpose computing and we had saying that microcontrollers are being used in many cases for special purpose embedded computing. The question is why microcontroller. Obviously the dominating factor in this case is low cost and the small foot print that it offers. Since embedded systems is suppose to perform specific tasks we

do not need facilities of general purpose computing on a micro-controller targeted for embedded systems. So, we can have multiple peripherals integrated together onto a single piece of silicon to get a chip which has got as small a foot print as well as less power consumption.

(Refer Slide Time 09:08 min)

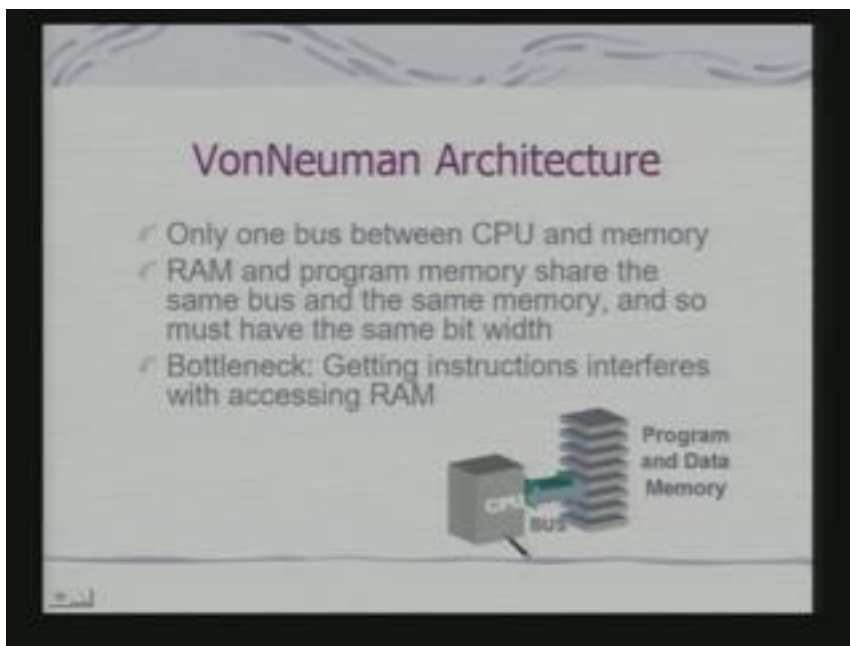


We will find that on micro-controllers, on the chip itself we have a number of I/O capabilities. Because I need to interface external devices which has sensors as well as actuators and it provides, you see, integration with external circuit and foremost as I already stated, since I am having single purpose usage I can manage with less sophistication on the processor hardware itself. That is why I have other peripherals integrated on to it, to get a complete system with a small foot print with small power consumption and that basically the reason why I go for micro-controller for embedded system design. Now, we shall review computer architecture before we go into details of this micro-controller architecture because that would provide us the proper background and perspective.



All of us are familiar with classical VonNeuman Architecture. Here, what I have got; I have got CPU and I have got program in data memory put together and having a single bus connecting the tool. I provide address from the CPU to the memory to fetch instruction as well as to fetch data and we use the bus to read the data as well as write the data on to that memory.

(Refer Slide Time 10:22 min)



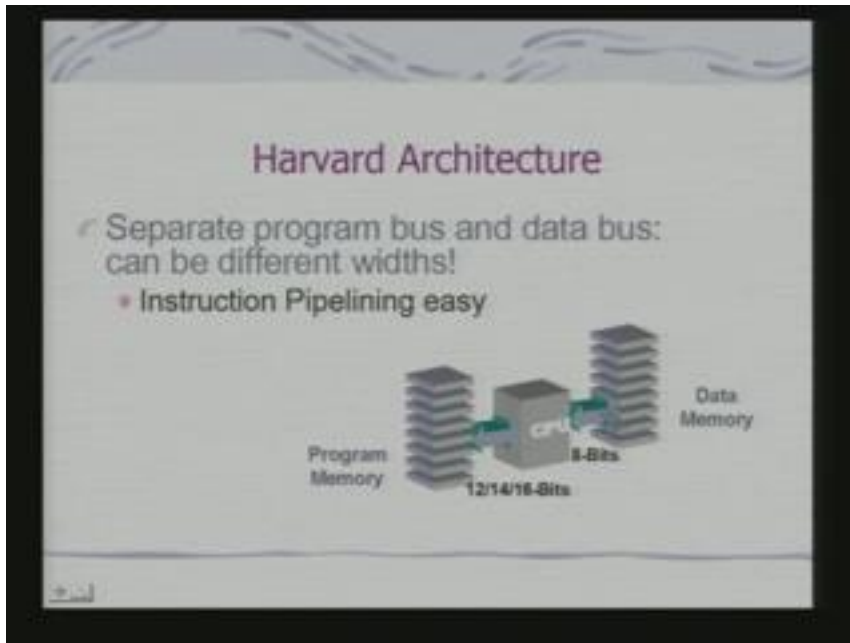
In fact, there is no difference between the data and instruction in VonNeuman architecture. Both program and data memory is connected to the CPU by the same

address and data bus. So what happens as a consequence, we can get a bottleneck because getting instructions may interfere accessing RAM for data.

So, I need to have distinct cycles on this bus forgetting instructions as well as getting or writing data. The other typical feature is that we have the same bus and hence the instruction and the data word is typically of the same bit width. Now with this background we shall look at a variant of this architecture; this is called Harvard architecture.

The fundamental difference in the Harvard architecture is that I have got a program memory different from that of the data memory. And I have shown here, two distinct buses connecting my CPU to the program memory and data memory. And interestingly since I have two distinct buses their bit widths can be different. They can be, as shown here an example, where the program memory can have 12, 14 or 16 bits while the data memory can be simply having the bus width of eight bits. What is the advantage of this, the advantage is just look at this structure. I can fetch an instruction from the program memory because program memory is typically expected to store the program. And while I am actually doing a processing, let us assume I have got some data in the CPU and I am doing some processing and then I need to store the result. And where shall I store the result? I shall store the result in data memory.

(Refer Slide Time 12:11 min)



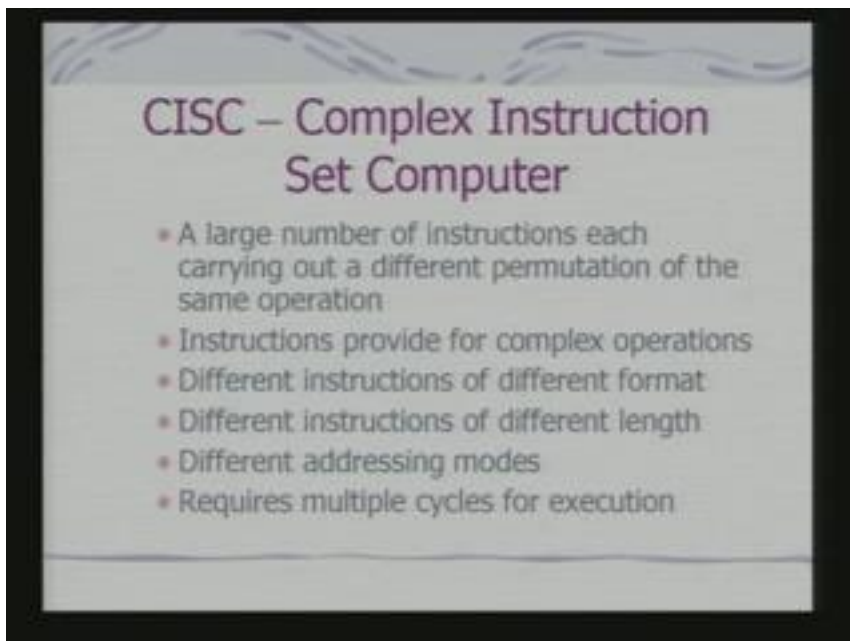
Now, while storing the result in the data memory I shall be using this bus and during that time period this bus is actually free. So, I can use this bus to fetch instruction. So, the instruction pipelining is facilitated by this Harvard architecture and this is the advantage of Harvard architecture over VonNeuman architecture

and you will find many of this modern micro-controllers are built around Harvard architecture. Next, we come to the concept of CISC.

CISC is the Complex Instruction Set Computer. Pentium processor which is used in your PC is an example of a CISC processor. What are the characteristics of CISC processors? In fact, your 8086 and other family of the processors, other processors of the same family, 68000 all satisfy the properties of CISC. What are the characteristics of CISC? A large number of instructions are typically found in the repertoire of the CISC processors and they actually perform various complex tasks. And these instructions are typically of different length and different format and they require in many cases different number of clock cycles to execute. Also, it supports such processors support a number of addressing modes. So, what we find that the basic philosophy which goes into the CISC design is the more complex operations being implemented in hardware and more elaborate way of accessing data as part of the instructions because that is addressing modes; addressing

modes gives you a variety of ways of accessing data in the memory. And you can even have operations involving data in the memory. These are the typical characteristics of a systems processor.

(Refer Slide Time 14:13 min)

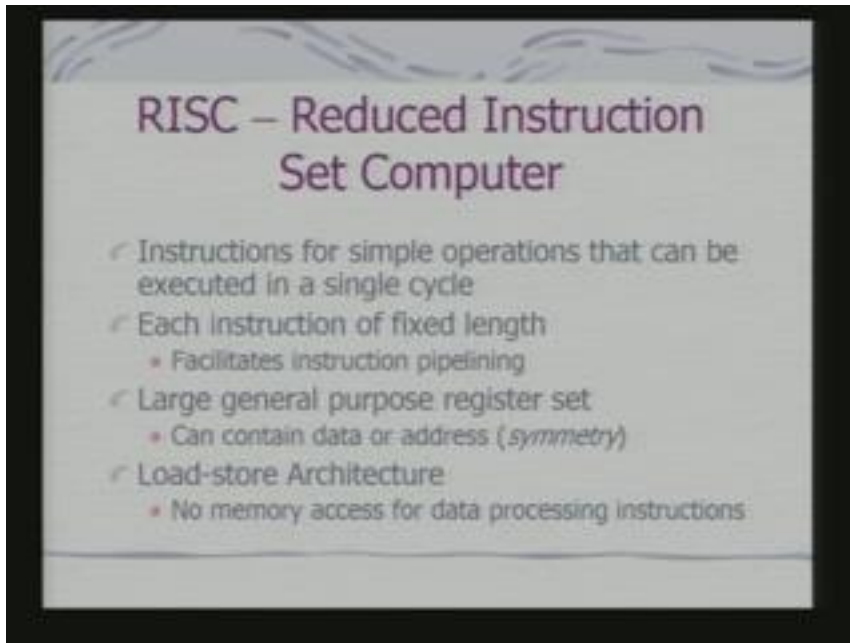


On the other hand, we have RISC of the reduce instruction set computer. As the name suggests, here the basic objective is to have less number of instructions and instructions are designed in such a way that they perform single operations that can be executed in a single cycle. Also, instructions are of fixed length. A fixed length instruction facilitates what we call instruction pipelining. Because I know that if I am fetching an instruction of a fixed word; so the next instruction that I fetch or the next byte that I fetch would

correspond to the next instruction and it will not correspond to byte or word of a previous instruction.

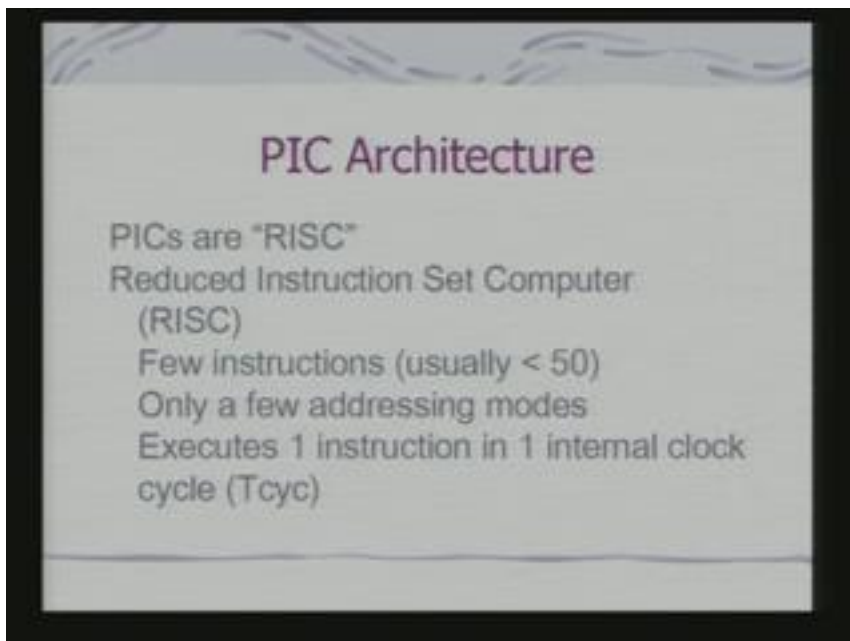
So, I can design fixed cycles of instructions to fetch words of fixed length which correspond to a single instruction for a RISC computer. It has generally a large number of registers and these registers can contain data and they can also contain addresses. And what we say, that is registers are symmetric because they can be used for address as well as data manipulation in a similar way. And your arithmetic operations, in fact, arithmetic as well as logical operations for RISC processors always involve registers. So, the operations are always involving registers and not involving memory locations directly. And once you have performed the operation, the data is stored to the memory. So, you have got stored instructions and before performing an operation, data is loaded from memory so you have got load instructions. So, that is why typically RISC architectures are also referred to as load store architecture. Yeah... you can have pipelining in CISC architecture as well okay, but the pipeline design for a CISC architecture becomes more complex than that of a RISC architecture.

(Refer Slide Time 15:20 min)



Now, we shall come to architecture of a particular microcontroller and that is PIC and PIC is one of the leading architectures for low end applications. What do we mean by low end applications? Typically the applications which require your 8 bit and 16 bit processors or even the applications which require four bit processors. So, these are typically low end applications and PIC is one of the leading architectures for low end applications. In fact, the basic objective with which PIC microcontrollers were designed were embedded control. PIC's are RISC processors; reduced instruction and processors. It uses fewer instructions, few addressing modes and typically executes one instruction in one internal clock cycle. Now, why PIC was designed as a RISC processor?

(Refer Slide Time 18:57 min)



Obviously the reason is that the moment I have got simple set of instructions and if I can execute as single set of instruction in one cycle, I can actually execute a large number of operations and I can have easy implementation of pipelining without having a complex hardware. Less complex hardware would essentially mean less consumption of silicon area as well as less consumption of power.

Let us try to look at the family; in fact it is a family of processors which have got variety of configurations.

(Refer Slide Time 19:49 min)



## The PIC Family: Packages

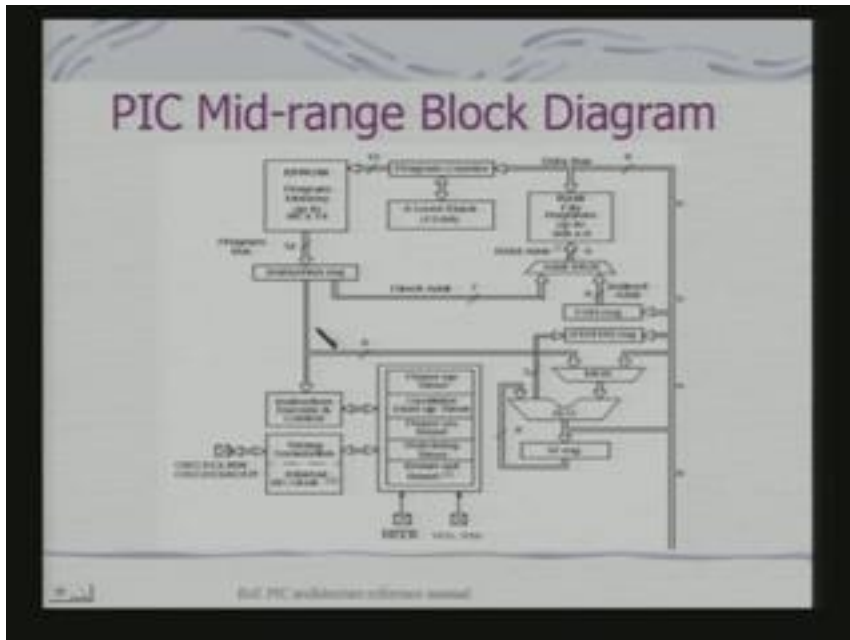
PICs come in a huge variety of packages:

Examples:

8 pin	:	12C50x (12bit) and 12C67x (14bit)
18pin	:	16C5X (12bit), 16Cxxx (14bit)
28pin	:	16C5X (12bit), 16Cxxx (14bit)
40pin	:	16Cxxx (14bit), 17C4x (16bit)
44 - 68pin	:	16Cxxx (14bit), 17C4x / 17Cxxx (16bit)

Here, you will find, we have shown some examples and with different numbers because PIC is not just a single microcontroller but a family of microcontrollers and this 12 bit, 14 bit, 16 bit, these are different word lengths that these processors support. Let us look at this block diagram and this is the block diagram of what we call a mid range PIC processor.

(Refer Slide Time 20:19 min)



I have showing here a part of the architecture and not the complete architecture, because a complete architecture will have other components as well. Here, let us see, this is my program memory; this program memory is obviously distinct from that of my data memory. The data memory is located here so, this is an example of Harvard architecture. I have got this program counter and program counter is obviously generating addresses for my program memory for execution of the instruction. The interesting feature is, it has got a hardware stack. Why hardware stack is there? Because when there is a call or there

is an interrupt the program counter value has to be saved to enable return to that same location. We know what for the stack is used in call in and interrupts for general purpose microprocessors.

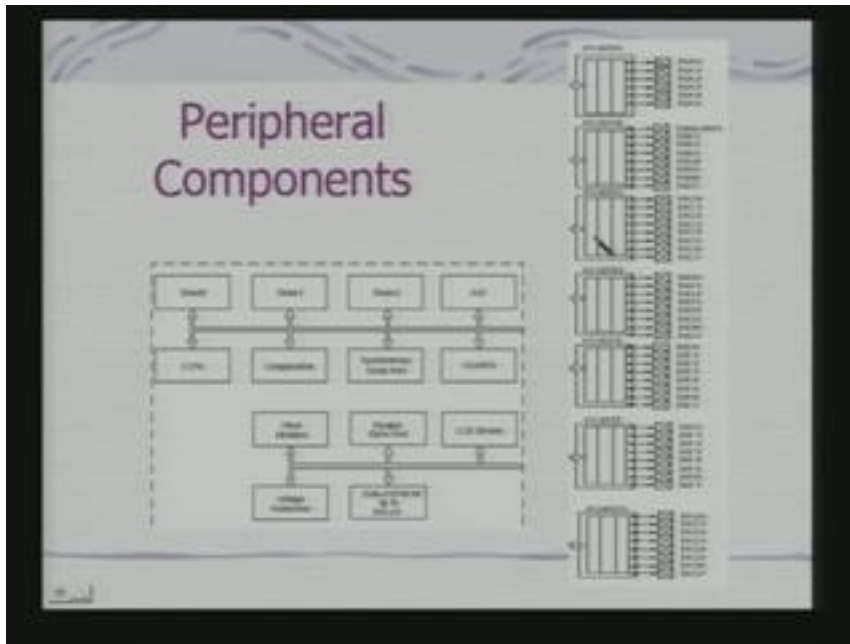
Here, the stack is implemented in hardware and if you see here, what is very interesting in this stack area is not part of program memory or that of data memory, but typically 8086 or such processors stack is part of your main memory itself. This is your instruction register; now this instruction registers basic job is to take the instruction, the instruction is correspond to what? The instruction whose address is provided by the program counter, this instruction is decoded by this block to generate what is called the control and the timing signals for the internal block of the PIC processor. Here, there is this oscillators or the clock and this is the timing generation circuit with an internal RC clock. This provides a basic clock for the processors to operate. And these are the set of timer blocks that we have already talked about. We have got power of timer, then power on reset, watch talk timer, **thrown** out timer and so on so forth.

We shall discuss them later on; the only thing that I shall touch upon is this watch talk timer, because watch talk timer is a very important component of micro- controllers which are intended for embedded applications. What is watch talk timer? Say, you have written a program and you are running it on a PC and it has gone into an infinite loop. When you realize that, what do you do? You try to stop that program and for stopping that program you actually initiate and interrupt or a signal to the processor. Now, if such an error condition happens in an embedded application, you are not there to send in an interrupt or act like a control C from the keyboard or anything of that sort. It has to take a self correcting action because an embedded system is supposed to work without human intervention and supposed to work for all times without human intervention.

So, typically what watch talk timer is doing is, it typically gets loaded with a particular timing parameter that is a particular count. And it starts down counting with respect to the clock and what is expected is the system would reset watch talk timer at regular intervals. If it does not reset and the count terminates, it should generate and interrupt. Why it is not resetting? Most probably your software has forced the micro-controller to going to some infinite loop or error condition so that the system is not in a position to reset the counter value. This is a generic concept of watch talk timer in an embedded system. So, when the

timer finishes its count and it has not been reset it will generate an interrupt and that interrupt will enable the system to go back to one of the known predefined states, okay.

(Refer Slide Time 28:39 min)

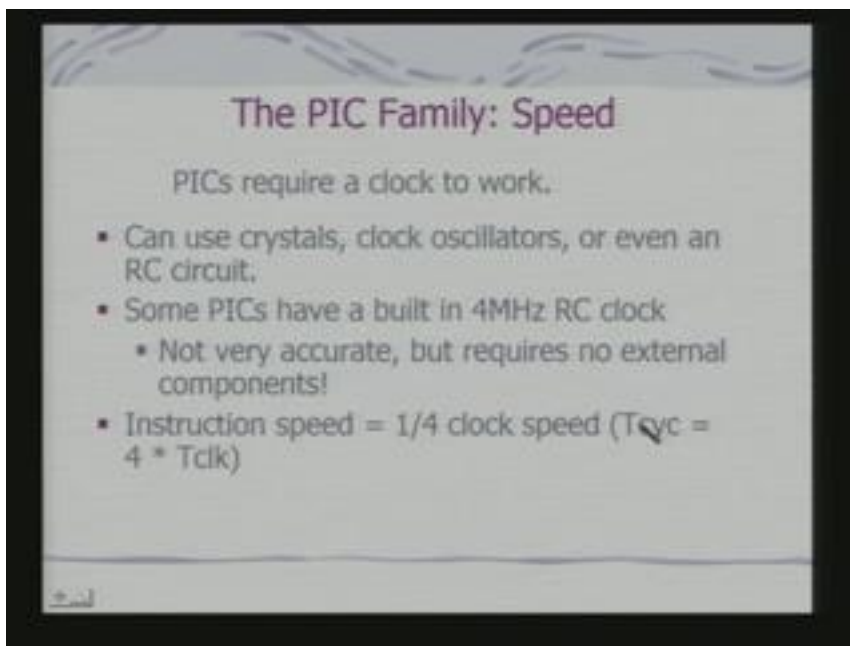


So, this watch talk timer is a very important component. This part is primarily the ALU part for the doing arithmetic computations and what is interesting that you will find that this RAM area is also being referred to as register file, okay. In fact, all these RAM locations can be actually considered as registers and as we, I have already stated a RISC processor will have a large register bank, because it will use a registers for majority of its operations. Here, you will find that the same condition is being satisfied for the PIC processor. This instruction register typically will have the instruction and it will refer to the registers for operation; so that is why you find a bus connecting these a part of your instruction register to the address multiplexer through which the register gets selected for any kind of arithmetic operation, okay.

So, what we will find that from these register, the data would flow on to the data bus and then it will go in to this multiplexer so that it can be used in ALU for various operations. Now, there are two kinds of bus structure you have found here one is, here the, I have got the direct address which goes in through address multiplexer to select the registers. And here you will find the directly the data eight bit data has part of the instruction is coming to the multiplexer by which it is fed to the ALU. Now, why is this bus there? To enable to have immediate mode addressing, that is when the data is part of the instruction. And there is also a provision for indirect addressing. Now, how does the indirect addressing take place? Here, if you look into it conceptually, it is that I select a register, okay and from that register I generate I come to what is called a FSR register and from FSR register I generate the indirect address. It is a slightly complex process; we shall come to exactly the mechanism which is used for indirect addressing when we discuss the instruction set of the processor.

So, at least from this architecture, what is straight forward is, I support immediate addressing, direct addressing, indirect addressing. And primarily my operations are using this RAM and this RAM is nothing, but also the same register bank that of a RISC processor. Now, we come to other parts of this same architecture this are all integrated on to the same chip along with the previous blocks that I have shown it. These are nothing but a set of ports; through this port actually the external devices are to be connected. And these blocks contains again support for variety of peripheral devices. I am just referring to two examples; one is LCD drivers because typically if it is a standalone device it would require an LCD display. So, on the micro-controller itself, we have the integrated form of LCD drivers so that I can directly have the LCD display interfaced with the micro- controller with minimum of additional circuitry. Similarly, I have got the other ports, okay. You will find synchronous serial ports and as well as **US sorts**, this is for the purpose of having serial ports implemented for communication, maybe with PC and other devices.

(Refer Slide Time 30:55 min)



The PIC Family: Speed

PICs require a clock to work.

- Can use crystals, clock oscillators, or even an RC circuit.
- Some PICs have a built in 4MHz RC clock
  - Not very accurate, but requires no external components!
- Instruction speed = 1/4 clock speed ( $T_{\text{yc}} = 4 * T_{\text{clk}}$ )

There are a number of other timers as well. In fact I have shown you here the three timers; these timers are different from that of watch talk timer. Watch talk timers is the

special purpose timer. These timers are application specific timers that mean you are expected to program these timers depending on your application requirements. There is an analogue to digital converter block to interface or receive an analogue input. Now, obviously I have seen that, I have got a clock generated block and you know that all processors require clocks to work with. So, PIC's also require a clock and typically it use crystals or clock oscillators or even an RC circuit.

And instruction speed is typically what we called one fourth of clock speed.

So, this is, that is I talk about T<sub>cy</sub> cycle which is four times a T<sub>clk</sub>. So, these are the typical parameters that we refer to in a PIC architecture. So, where from this factor four comes in? Let us try to understand that. Before that let us look at some examples of clock frequency. These are some family of the processors and these are the clock frequencies with which these processors operate. Here, we will see that this is my oscillator clock, the basic clock which is coming in. See, if I say a 4 megahertz processor, a 4 megahertz clock input that means essentially I have a clock whose frequency is 4 megahertz.

(Refer Slide Time 31:22 min)

Clock Frequency	
▪ Examples	
12C50x	4MHz
12C67x	10MHz
16Cxxx	20MHz
17C4x / 17C7xxx	33MHz
18Cxxx	40MHz

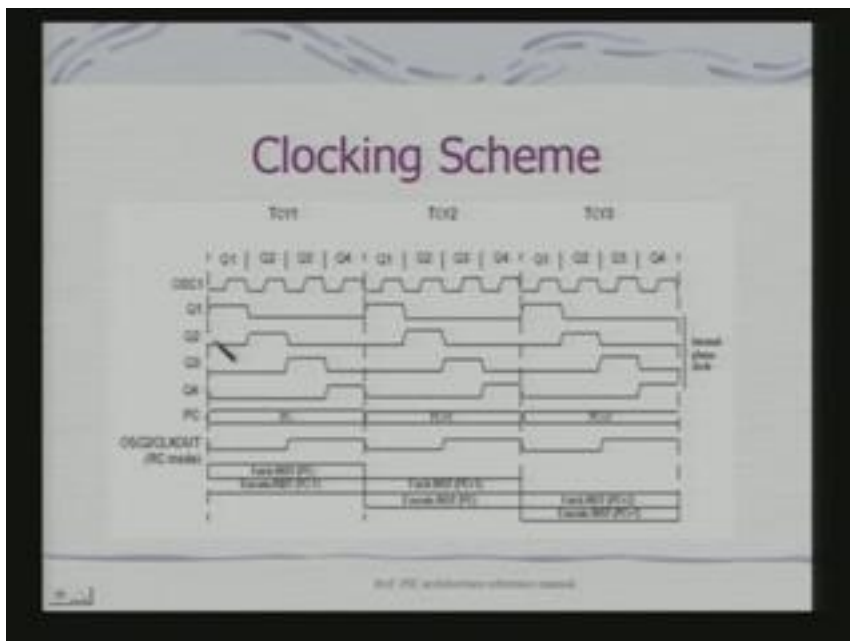
This clock is actually internally divided or split up into four **quadrature** clocks.

So, I am showing you Q 1, Q 2, Q 3 and Q 4. The whole instruction execution is managed by this four **quadrature** clocks. So, what you find is, in these, just look at here, so you have got your first Tcyc where you actually fetch an instruction. Let us look at start with this in this I fetch an instruction. Now, once I fetch the instruction, so I have shown it within the bracket cyc PC.

That is, I am fetching which instruction fetching the instruction which is being pointed to by the program counter. In the next clock cycle, what I am doing? I am actually executing this instruction that I fetched. The execution of this instruction would essentially mean what? A data movement in terms of bus demand; data movement from your register file, that moment would take place on your data bus.



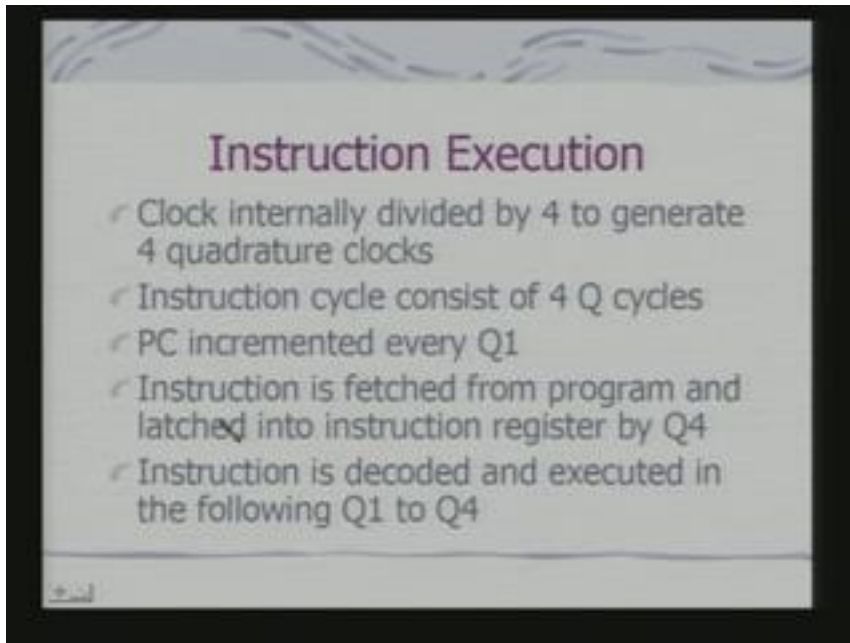
(Refer Slide Time 32:03 min)



So, my program bus is free so I can fetch an instruction. So, what I am fetching? I am fetching the instruction at PC plus one. So, similarly I have got, I have shown here, when I am fetching the instruction point it to by PC I am executing in instruction point it to by PC minus one. And this is how the instruction is execution takes place and what you really realize is that in one cycle one instruction gets executed. Is this clear? So, clock internal is divided by 4 to generate 4 quadrature clocks. Instruction cycle consist of 4 Q cycles. PC is incremented every Q 1; instruction is fetched from and latched into

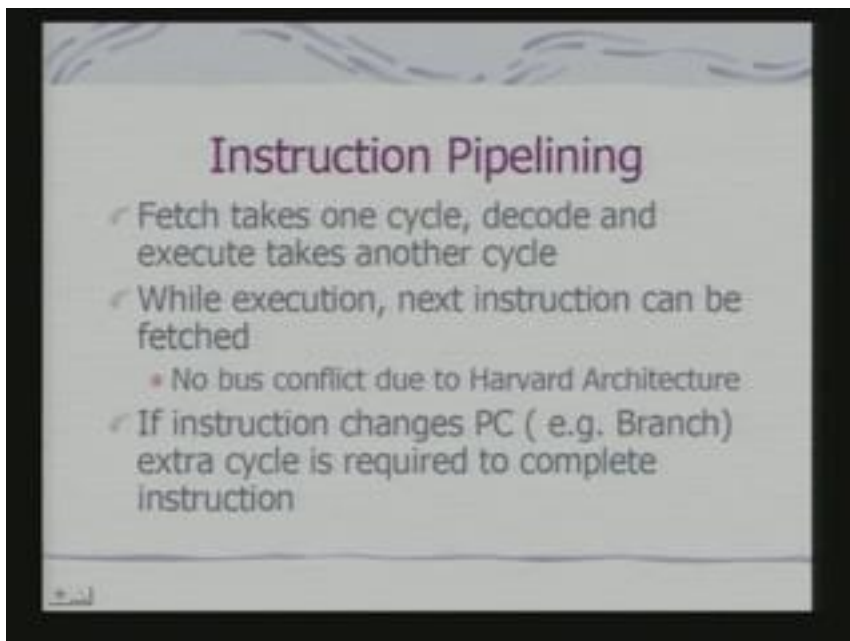
instruction register Q 4 and instruction is decoded and executed in the following Q 1 to Q 4.

(Refer Slide Time 34:04 min)



So, if I go to the previous slide you will find exactly that what is happening. So, in this Q 1, this PC is incremented and between this Q 1 to Q 4; Q 4 in the next cycle instruction getting executed. This is typically how the instruction execution takes place in PIC. So, obviously I can have instruction pipeline. Okay, you can ask your question. We have gone for four. This is an example of architecture by four. This is how there was a demand of the internal design because this processor I said, it is a low end application. So, it is not required to support an extensive multistage pipelining. So, when you go to complex processors you will find more complex multistage pipeline support. Here, it is just the support such that I have an overlap of an instruction execution and instruction fetch. Now, you find that in this case of an instruction pipelining, fetch takes one cycle, decode and execute takes another cycle.

(Refer Slide Time 35:26 min)



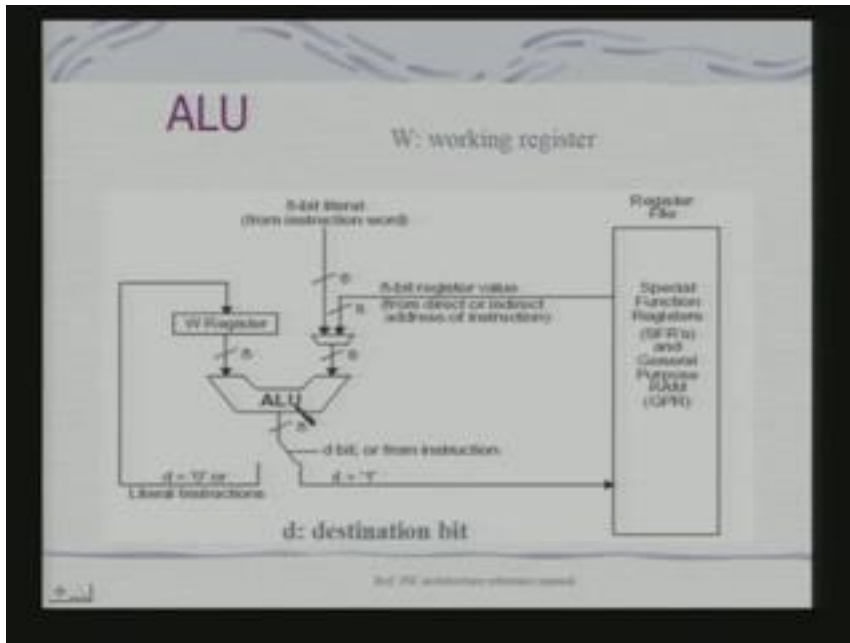
So effectively I have got one instruction execution and one fetch overlapped. They can be more complex pipelining when we look at more complex processors. You will have more complex architectures coming in place

and there are no bus conflicts because we have got essentially Harvard architecture. Now, the last point that I am showing in this slide is very important. If instruction changes PC that is, when there is a branching extra cycle is required to complete instruction. In fact this is a basic problem with all pipelining that is whenever there is a branching obviously

my pipeline efficiency goes down, okay. So, here I would require more than one cycle to complete the instruction.

Now, let us look at ALU in more details the ALU part of the architecture that I had already seen.

(Refer Slide Time 37:14 min)



Now, in case of an ALU, in fact, you remember we had there was a register called W register. In fact W register placed a key role in all arithmetic operations.

This W register is called working register; in fact it is very similar in concept to that of an accumulator that you have in 8085 or 8086. You have your operands either from instruction work when you send immediate or it can come from a register and that is one of the operands that is used in an arithmetic operations. The other operand universally comes from the W register. Now, the output of the operation goes either to W register or to any one of the registers in your register file depending on what is called a destination bit in the instruction. Destination bit when it is zero, it is typically part of the W registers so the data is stored in the W register. So, that it can be used in the next instruction without being fetched from the register bank. If it is one it is stored in the register file.

The status register is just like your flag register in your 8085. You can have ,see, what we have is that when an operation takes place, the result which goes out from ALU has to be stored in some register; which register it is to be stored depends on the D bit. There is one particular bit in the instruction word which is called destination bit depending on value of that bit. If it is zero then that data is stored in the W, if it is one it goes to one of the registers- general purpose registers. Along with this register there is a status register

which place a key role just like a flag register because flag register stores what, the status of a computation. So, in this case status register plays that role, but status register along with arithmetic status stores other information as well.

(Refer Slide Time 37:55 min)

**Status Register**

Status register contains

- Arithmetic status of ALU operation
- RESET status
- Memory bank select bits

R/W0	R/W0	R/W0	R/1	R/1	R/Wa	R/Wa	R/Wa
IRP	RP1	RP0	TO	PD	Z	DC	C
M7							M0

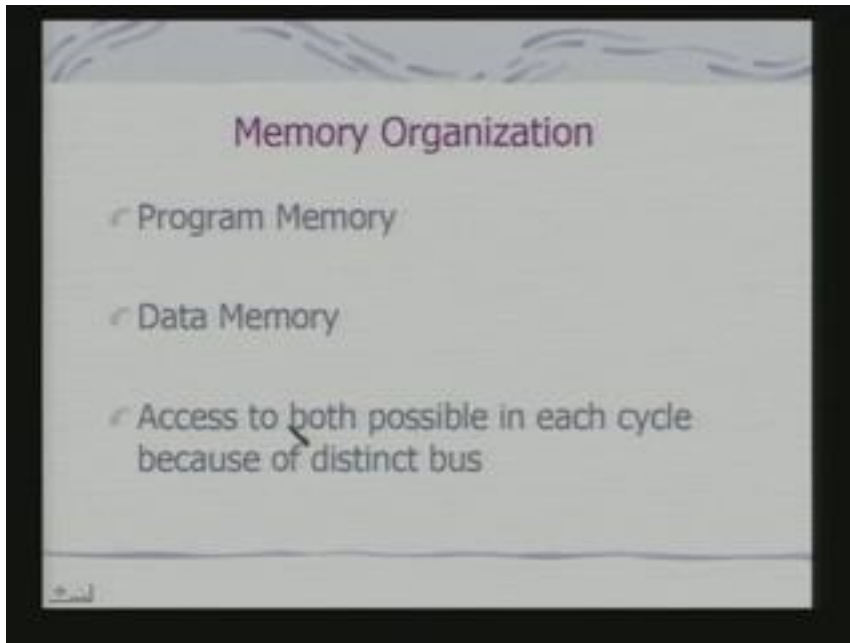
TO: time out; PD: power Down; IRP, RP1, RP0: bank select; Z: zero, DC: digit carry (BCD); C: carry

So, what we say the status register in a PIC stores arithmetic status, RESET status and memory bank select bits. What are memory bank select and what are their options and what are the functions? We shall come when you look at the memory organization. Particular interest for ask since we are considering ALU operation at these bits: Z DC and C. What are their functions? Z is a zero bit, see you know what a zero bit is because when reset of an operation is zero this flag is set. What is DC? This is called a digit carry bit and this is a carry bit. What is a difference between the two? When there is a, if I am

using an 8 bit arithmetic operation, when there is a carry from the fourth bit I get digit carry. Why that carry is important; if I am looking at BCD operations. In a BCD operation I need to keep track of the carry from the fourth bit and that is why I have got a special status register to indicate that. These two bits are used for time out indication as well as power down mode and these we shall discuss when we consider the power management point or power management aspect of these micro-controllers. These bits are used for memory bank select. The memory in PIC is obviously organized into two areas: program memory area as well as the data memory area.

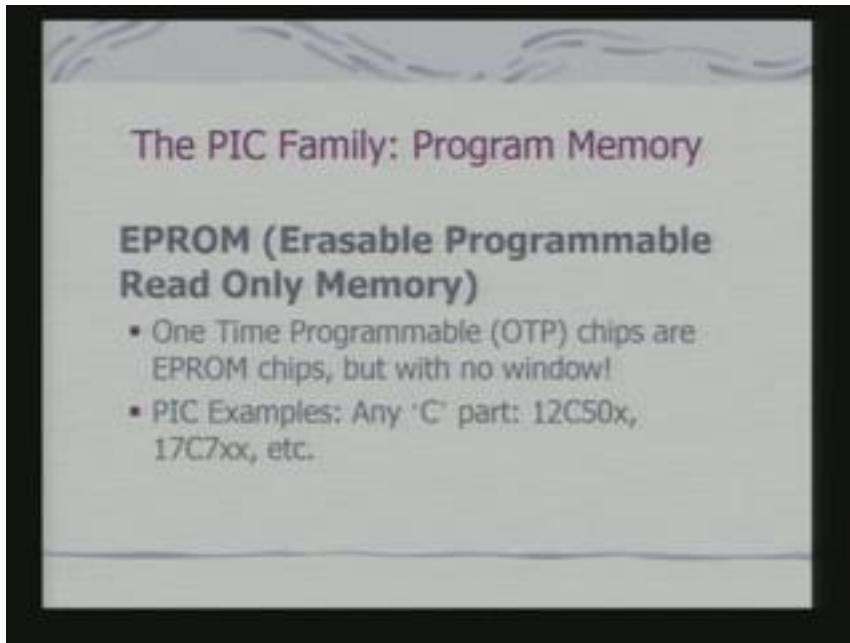
(Refer Slide Time 40:51 min)





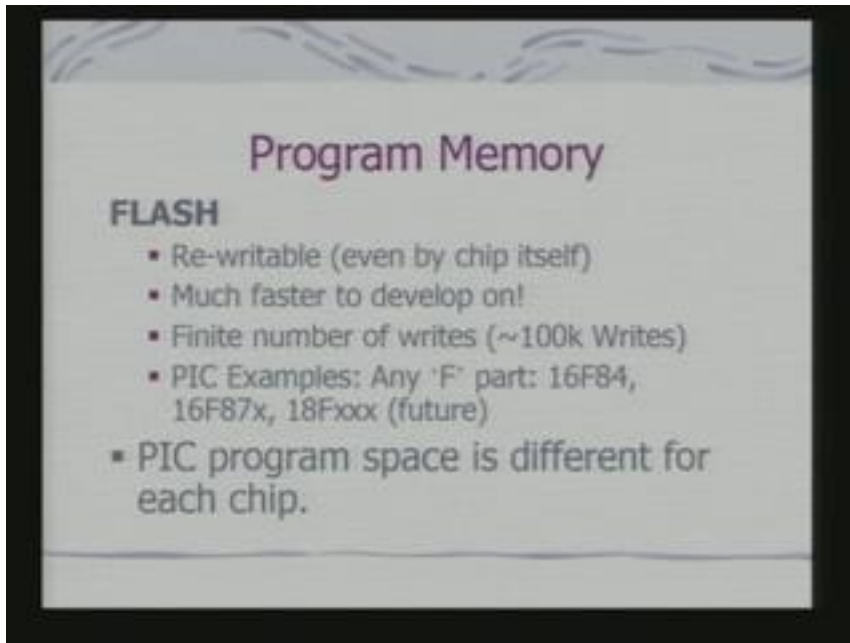
And accesses to both are possible in each cycle because I have got the distinct busses connecting them and that also you have already seen, facilities pipelining. In fact, PIC family has got different kinds of program memory. One possibility that we have seen already in the diagram are EPROM, Erasable Programmable Read only Memory.

(Refer Slide Time 41:31 min)



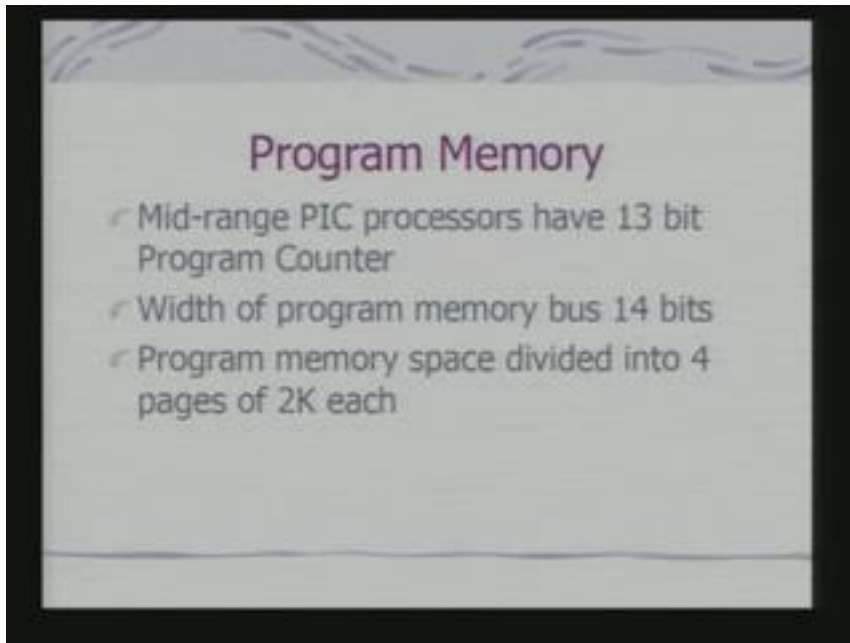
In fact, one time programmable chips are EPROM that means they have EPROM, but there are no windows. Why no windows? Because I can use window to erase the program through ultra violet light and if there are no windows then effectively I cannot erase that. So, I get one time programmable in fact they are something like a factory programmable chip. So, when we are doing a mass production of an embedded system the software development cost goes into NRE that is Non Reckoning cost and after that I simply write that program onto the EPROM and produce a chip and that program is not expected to be change ever after and these are particularly intended for very low cost applications. So, when you buy a PIC processor you should look for C, if there is C in its number that means it has got an EPROM as a programming. The other option is flash, you can have flash also in the program memory and when you have flash it is typically that PIC nomenclature has got F. So, flash is what, an electrically rewritable memory actually and it can be written even by chip.

(Refer Slide Time 42:39 min)



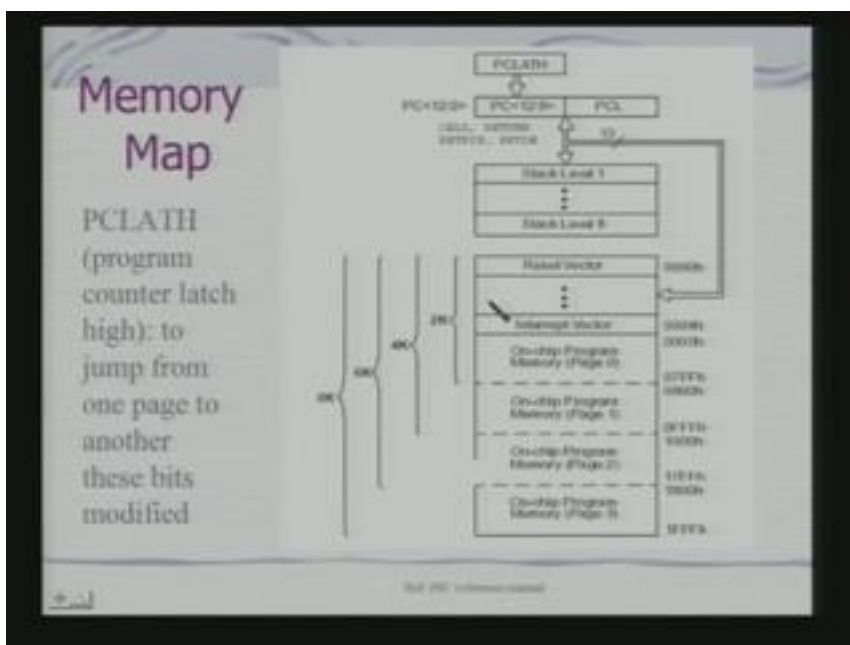
And much faster to develop on why because your software can be modified and modify on the fly. That means I can get or appliance which is got a flash memory I can update that program in the flash memory after even I have got the system and I have used it for a year. I get upgrade of the software and I put the upgrade of the software because I have got the system which is using a PIC processor with flash memory. This is the flexibility that use of flash memory in program area provides and in fact the program space for the PIC processors is different for each and in fact this is an important point to consider when you are selecting a particular processor from PIC family. Typically the program memory is thirteen bit, has got thirteen bit program counter.

(Refer Slide Time 44:27 min)



So, what is 13 bit essentially means, in fact it is a 8 K program memory effectively. And talking about mid-range PIC processors, there are other PIC processors range which has got a variant of this size. Width program memory bus is typically 14 bits; 14 bits that means this is the length of the instruction. And program memory spaces is divided into 4 pages of 2 K each and let us see how this organization is really managed. So, I have got the PC has got how many bits 13 bits, okay. So out of this thirteen bits, a part of it is map two, what is called program counter LATH high register, okay.

(Refer Slide Time 45:35 min)

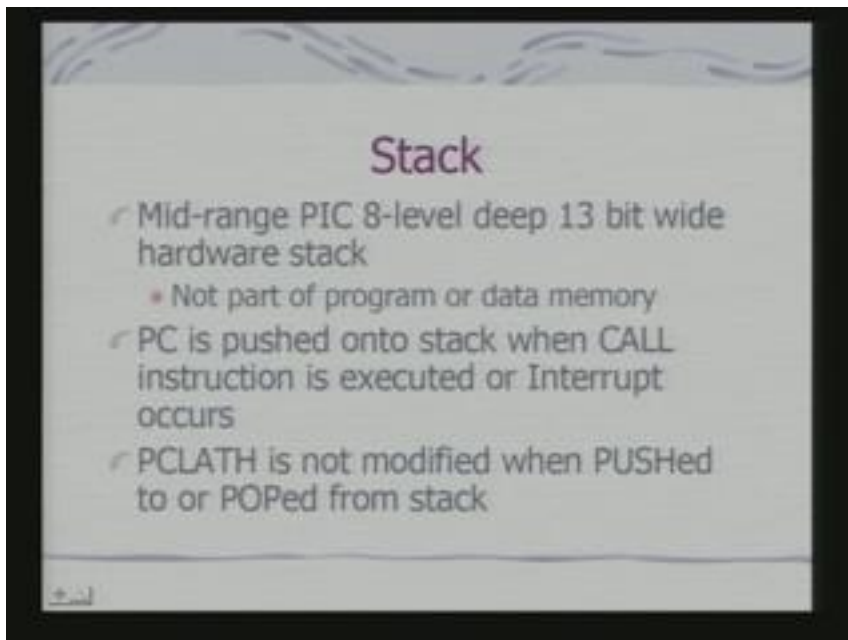


This part is mapped program counter latch high register and it is connected as for the architecture also you remember, it is connected to the hardware stack and to the 13 bit bus it is connected to the register bank as well. What I have shown is the individual pages which are of size 2 K; individual pages which are of size 2 K and the first page typically has got your reset vector, interrupt vector. Interrupt vectors are what, if an interrupt occurs where the jump has to take place.

So, the first page is typically allocated for those purposes and then subsequent pages are for one chip program memory, okay. So, your program is typically stored in these areas,

okay and what is interesting about is that this first part is separated out. Why this first part has been separated out? Because using this part I can actually do what, jump around at various locations with various kinds of constraints in post. The stack has got 8 level deep, typically mid-range PIC has got 8 level bits that means what it can stored 8 values-eight 13 bit values and obviously the PC is push on to this stack when there is a CALL or wherever the interrupt occurs.

(Refer Slide Time 46:43 min)



**Stack**

- ✓ Mid-range PIC 8-level deep 13 bit wide hardware stack
  - Not part of program or data memory
- ✓ PC is pushed onto stack when CALL instruction is executed or Interrupt occurs
- ✓ PCLATH is not modified when PUSHed to or POPed from stack

And typically PCLATH register, that PCLATH high register is not modified when PUSHed to or POPed from the stack. What does that mean? That means effectively you are confined to pages, okay. The banks that are organized into, you are confined to those pages. So, this gives a certain kind of restriction in moving around the value of PC. Then, we look at the PIC data memory, okay. Now, data memory is organized into two parts: one is called general purpose registers, the other part is basically special purpose registers or special function registers.

(Refer Slide Time 47:23 min)

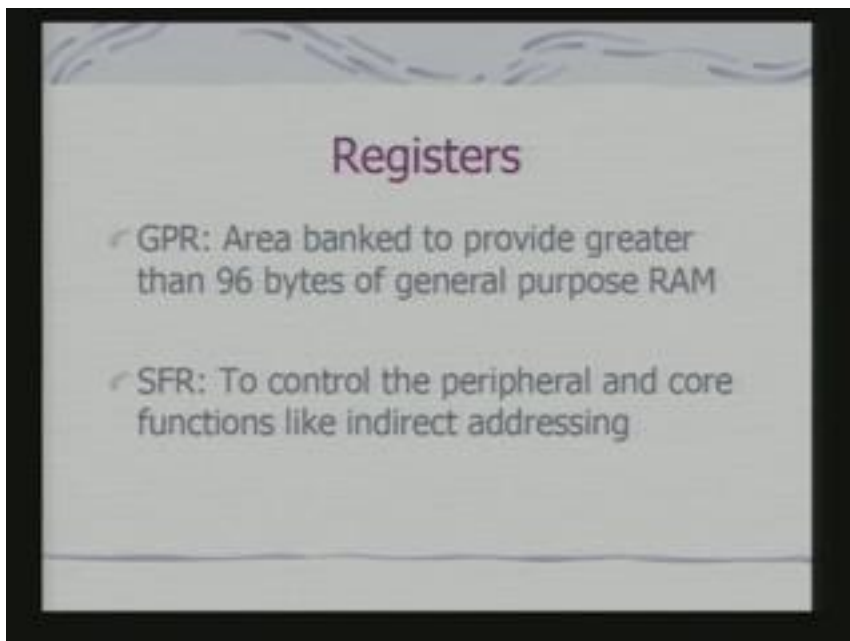
## The PIC Family: Data Memory

- PICs use general purpose "file registers" for RAM  
(each register is 8 bits for all PICs)
- ✓ Programs are stored in program space (not in data space), so low RAM space.
- ✓ Register File Memory Consist of 2 Components  
General Purpose Register (GPR) Files  
Special Purpose Register (SPR) files
- Memory organised into banks  
*16F877 has 4 banks of register*

Now, program since they are stored in your program memory. Typically you will find that the data memory is less because the data memory is typically is to be expected as a scratch pad memory for temporary calculations and other thing and for storing the input data, okay if it is available and for the major part of the data. Let us look at this example that, if I need to provide some background data say some offsets is to be is to be provided as part of the program. This offset is the permanent data and that offset value is expected to be stored where, in the program memory and not in the data memory. Data memory is typically to store the data which is getting generated during execution of the program and to store information which maybe coming from the external devices and to be used for subsequent calculation. Special purpose registers provide variety of core functions and here also the memory is organized into banks.



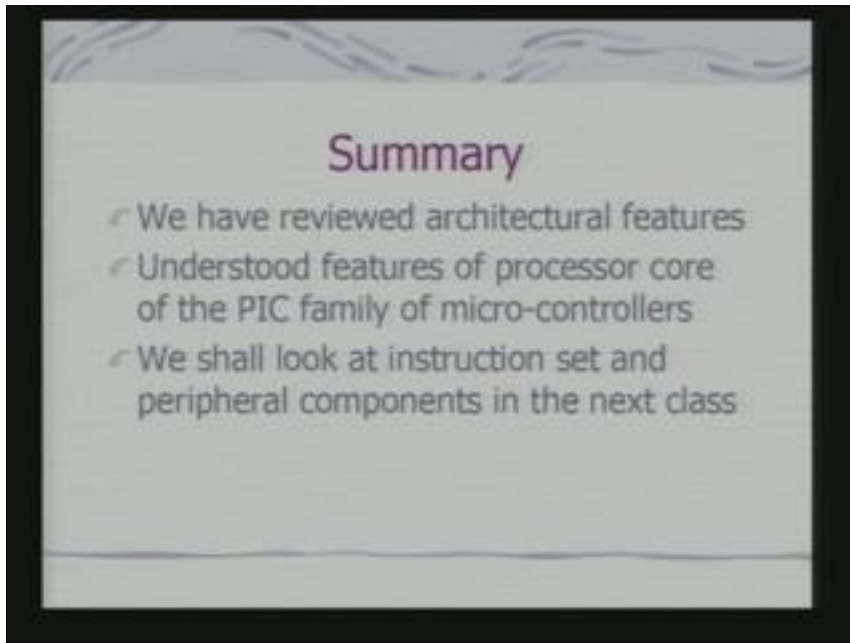
(Refer Slide Time 49:12 min)



If you remember we talked about the bits in the status registers, the bits in the status register, the first 3 bits that I had shown in the status registers, they are used for selection of memory banks. Typically, area that is in the general purpose registers, this bank are arranged so that it can provide greater than 96 bytes of general purpose RAM. And again typically I am talking about mid-range processors and again depending on which processor or which micro-controller you actually choose, this value could change and SFR or special purpose register SPR is used to control the peripheral and core functions

like indirect addressing. Peripheral control means what, that if I have to program the peripherals set some bits on the peripherals then we shall be using this special function registers.

(Refer Slide Time 49:57 min)



So, this more or less gives me the basic introduction to the core features of the processor, PIC processors. I am not talking about the complete PIC micro-controller obviously because I have not yet looked at the peripherals and the I/O ports. What I have looked that so far is the core processor, the CPU, its registers how the program memory is organized, how the data memory is organized and how instruction pipelining is implemented in PIC. We shall a look at instruction set and the peripherals components in the next class and subsequent lectures. If you have any questions now we can have them now. See, EPROM, Erasable Programmable Memory is not necessarily a onetime programmable memory, okay. I can erase and I can reprogram but typically if I have what we have on an EPROM, we have window by which we use ultra violet light to clear the data.