**Embedded Systems**

Dr.Santanu Chaudhury

Department of Electrical Engineering

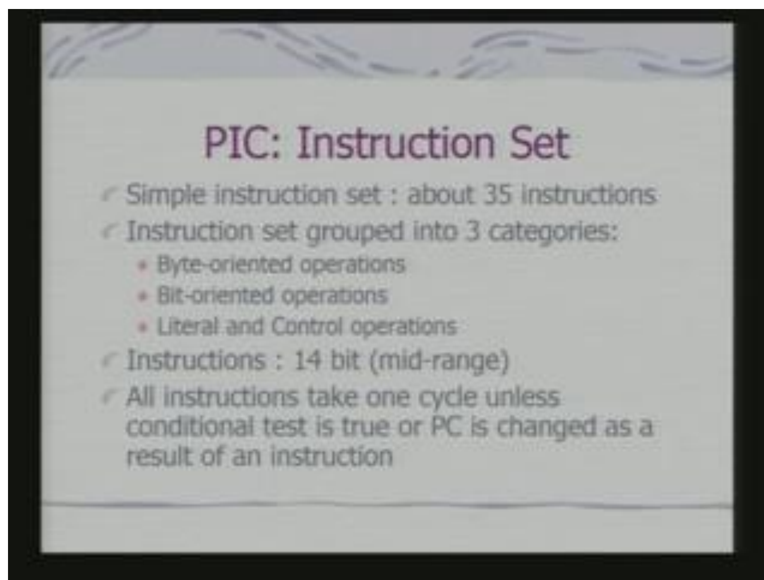
IIT Delhi

Lecture 3

PIC: Instruction Set

In the last class we have studied the architecture of PIC processor. Today, we shall look at the instruction set of the PIC family of processors. PIC has got very simple instruction set. Typically it has about 35 instructions. These instructions can be grouped into three categories: byte-oriented operations, bit-oriented operations, literal and control operations.

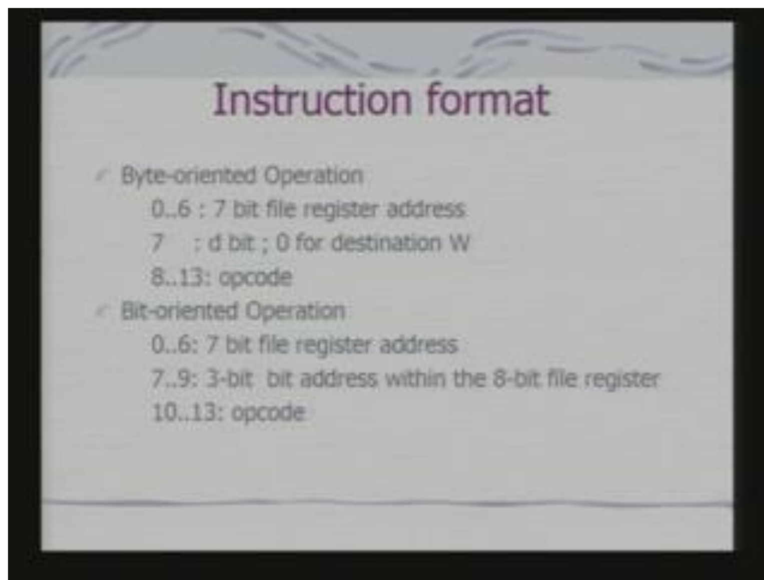
(Refer Slide Time 01:31 min)




For mid-range PIC processors, instructions are of typically 14 bit long. All instructions execute in one cycle unless some conditional test is true and the next instruction has to be skipped or the content of PC is explicitly changed because of the instruction. In those cases execution of each instruction takes about two cycles. The instruction format is

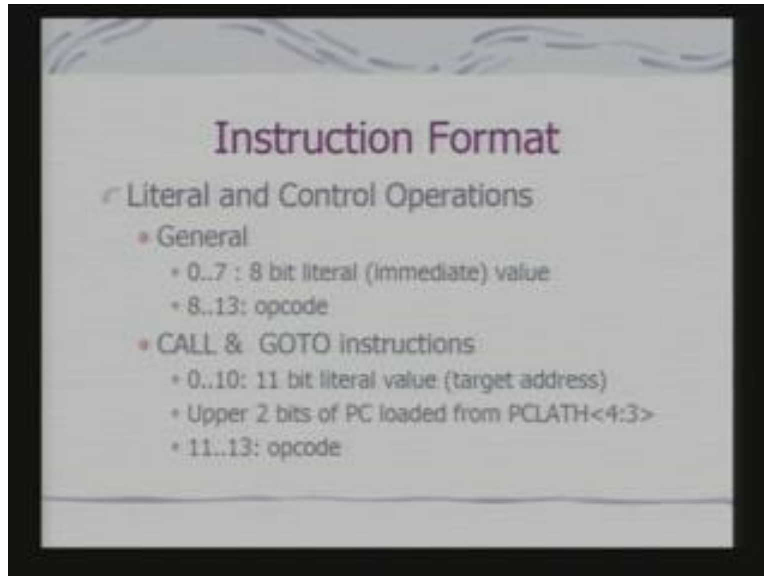
different for different kind of operations. We have already seen that we have got byte oriented instructions, bit oriented instructions and for these instructions, the structure looks something like this. In a byte oriented operation, you specify a 7 bit file register address which is typically the memory location.

(Refer Slide Time 02:31 min)



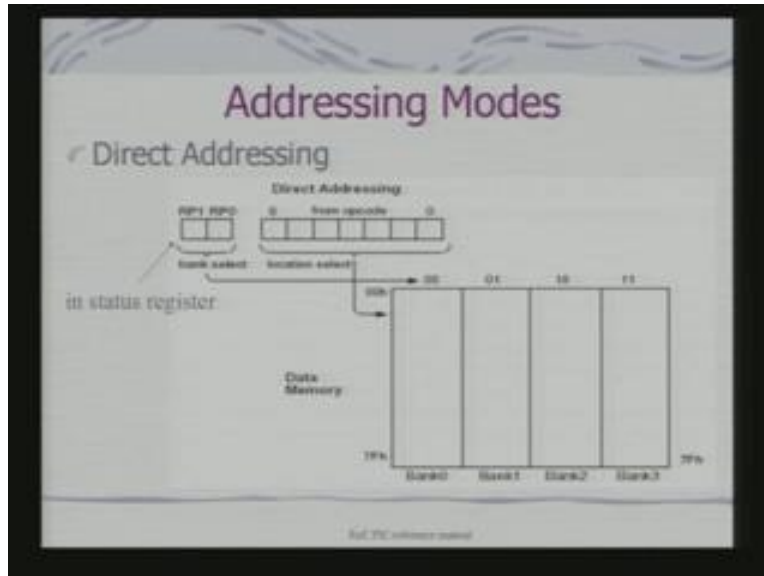

The 7 bit is a destination bit which is zero when we are using the working register W as a destination. Otherwise it is set to 1 when the register that you have specified in zero to 6th bit is a destination itself. 8 to 13 bits are storing the op-code that is the binary code for specific operation. For the bit-oriented operation, the interesting feature is using 3 bits you can actually specify exactly the bit number which is to be checked or modified in the file that you have specified in the address. Here also 10 to 13 bits are used to the opcode. In a literal and control operations the structure is slightly different. In this case, we use immediate value. For a general instruction, the 8 bit literal or the immediate value forms part of the instruction.

(Refer Slide Time 04:18 min)




In case of CALL and GOTO instructions, we have got 11 bit literal value as part of the instruction. Now, these literal values provide for immediate mode addressing and the data which is used in the literal value is part of your program memory itself. Interesting to note that your PC has got 13 bits and your CALL and GOTO instruction has got provision for specifying 11 bit target address value. So, the upper 2 bits come from PC latch high register, that bit number 3 and 4 is loaded for upper 2 bits. So, therefore when you are making a CALL or GOTO, the most significant 2 bits are always specified through PC latch register. This is a very interesting feature of PIC instruction set, particularly the instructions which control program flow.

(Refer Slide Time 05:49 min)

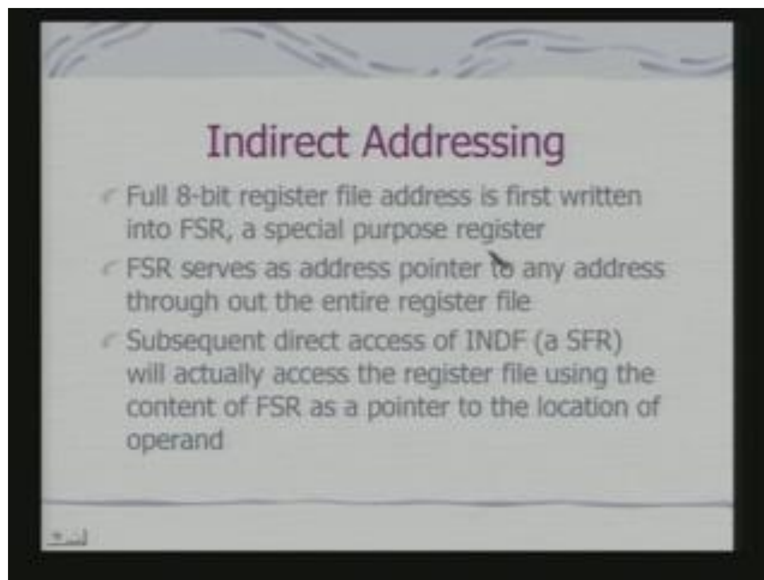



Now, let us look at addressing mode. What is addressing mode? Use addressing mode for specifying operands in an instruction. PIC supports direct addressing, indirect addressing as well as immediate mode addressing. We have already seen the structure of instructions which support immediate mode addressing. Now, let us see how direct addressing is supported in PIC.

In the PIC these two bits are RP1 and RP0 which are part of status register were used for selecting what is called bank. And in the bank the address is specified by this 8 bits which is part of the instructions. These bits which we have from the instructions, actually makes the choice of the location in a typical bank. So, these two bits are used for selecting the bank and the offset in the bank is specified by these bits which are part of the instructions. So, therefore in each of these banks I can select a location by specifying this 7 bit address in the instruction itself. In indirect addressing, PIC uses a slightly more complex structure. The full 8 bit register address is first written into FSR which is a special purpose register and FSR serves as address pointer to any address through out the entire register file. And the subsequent direct access of another special function register INDF will actually access the register file using the content of FSR as a pointer to the location of the operand. So, in this case what it implies is that, you first load the FSR

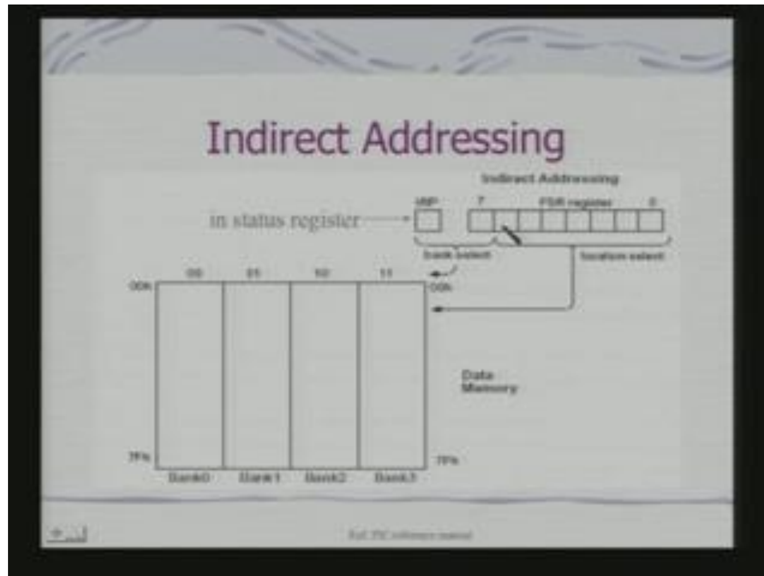
register with the value, now this value you need to load in the FSR register through an instruction; the bank select is via IRP bit which is part of the status register and the 7 bit in the FSR register. The remaining 7 bits, the 7 bits is used for selecting the offset in the bank.

(Refer Slide Time 07:05 min)




Now, once you have loaded FSR register with this value, what you need to do is, you need to access INDF register. Once you access INDF register. PIC actually uses the address in the FSR register to access operand in the memory.

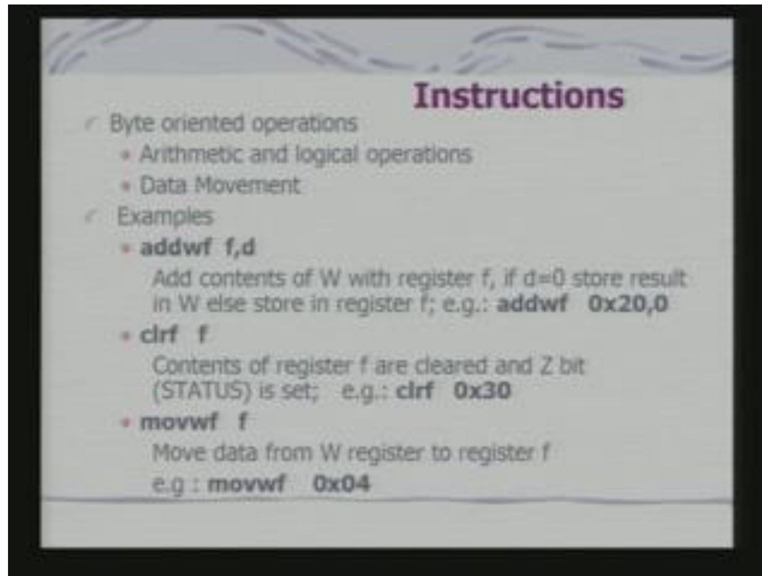
(Refer Slide Time 08:21 min)




So, actually you can see that in indirect addressing in the PIC actually requires use of not only one instruction, the two instructions are specifying the address.

There are a variety of instructions in PIC, although I have already told you, this number of instructions is much smaller compared to that of a typical CISC processor. If you are family of 8086 or 8085, we will find that compared to 8086 the number of instructions here is much less. And what is more interesting to note is the addressing modes. The number of addressing modes that are supported is also much less compare to any CISC processor.

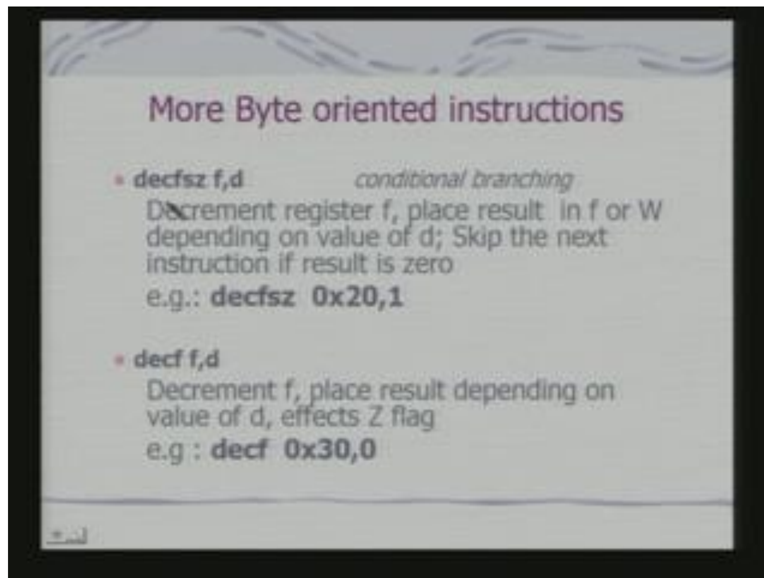
(Refer Slide Time 09:06 min)




Let us first look at byte oriented operations. This byte oriented operations are primarily targeted for arithmetic and logical operations as well as for data movement. We have shown some examples here. These instructions addwf f,d is an instruction for addition. When it executes what happens? The contents of register W, okay is added with register f and if d equal to zero the result is stored in W else, the result is stored in the register f. In fact, d is indicating your destination bit which we have discussed in instruction format. So, actually when will be using this, you will be writing it in this form. So, this actually represents a memory location in the bank and this memory is part of the PIC chip itself. It is in the register bank, that location you are specifying and then you are telling this is zero that means the result will be stored in W itself. So, you add the content of this location with the content of the working register W and store the result back in W. This is an instruction in for clearing the content. So, contents of register f are cleared and Z bit, Z bit is also part of the status register, is set. And here also you will be specifying exactly a location. Similarly, this is an example of a data movwf instruction. Here, you are moving data from W register to register f. Now, interestingly when you have to select the bank, the bank select operations you have to do trail to execution of this instruction. That means first you select the bank and once you select the bank then only you execute the

instructions so that you, correct location is accessed. Let us look at some more byte-oriented operations. Now, this is a pretty interesting instruction. This is an example of a conditional branching instruction.

(Refer Slide Time 12:09 min)



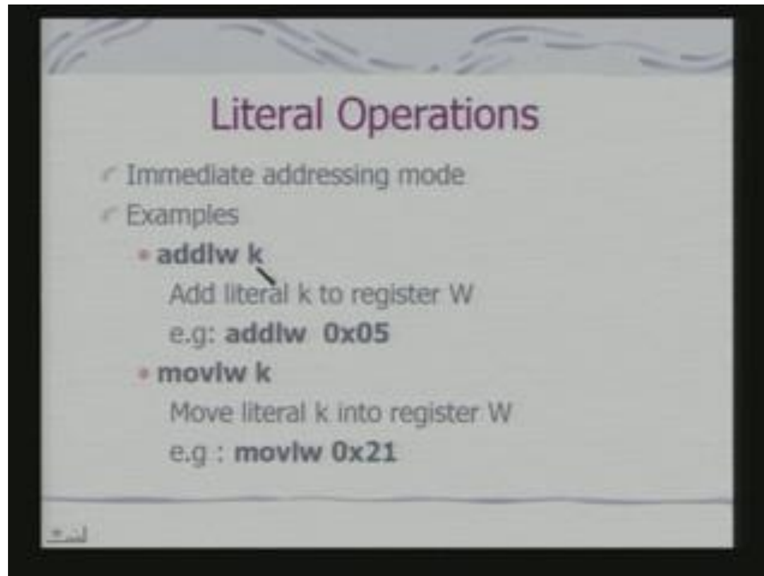

In the PIC, we have got GOTO instruction which is an unconditional branch. We have also got conditional branch. This is an example of that kind of instruction, but in case of conditional branch you skip only next instructions. So, it is not that you can make a conditional jump to any arbitrary location in memory. This is a very distinct feature of PIC and you should make note of it. Now, this instruction that we are looking at here, is interesting in various ways because these indicates and operation as well as branching. It is not just a simple branching instruction. In this case you decrement the register f; you place the result in f or w depending on the value of your destination bit d. And then you may skip the next instruction if the result is zero. Which result? After decrement the result that you are getting, if that result is zero then you skip the next instruction. So, this is just not conditional branching; this is to doing and arithmetic operations combined with conditional branching. This is a very interesting feature. Another interesting thing what happens is that, when the result is zero, the PIC has to skip the next instruction and if you

remember the way the PIC instructions are executed, I told you in the last class that while an instruction is being executed the next instruction is fetched. Now, in this case what happens, the next instruction which has been fetched cannot be executed.

So, that instruction has to be replaced by nope or no-operation. So, effectively this instruction would require now two cycle for execution that is when the result is zero, this instruction will actually require two cycle for execution and not just one cycle for execution. Along with it I am showing another example which is simple decrement. So, in this case you simply decrement the register and place the result depending on the value of d either in W or f and this would also affect the zero flag of the status register. Now, one interesting thing you should note with respective this previous instruction, that is a combination of an arithmetic operation and conditional branching into one instruction. What is the motivation for designing such instructions, because similar instruction you will find in other micro-controllers and processors targeted for embedded systems. This increases what we call code density that is instead of using two instructions, that is one instruction to decrement and then jump, you are using a single instruction. So, the number of memory locations that your code would occupy can decrease. If it decreases, what does what is that mean? You require less memory and obviously when you are working in an embedded system you try to work with less memory. And less memory would also imply less power consumption.

Let us look at now, literal operations.

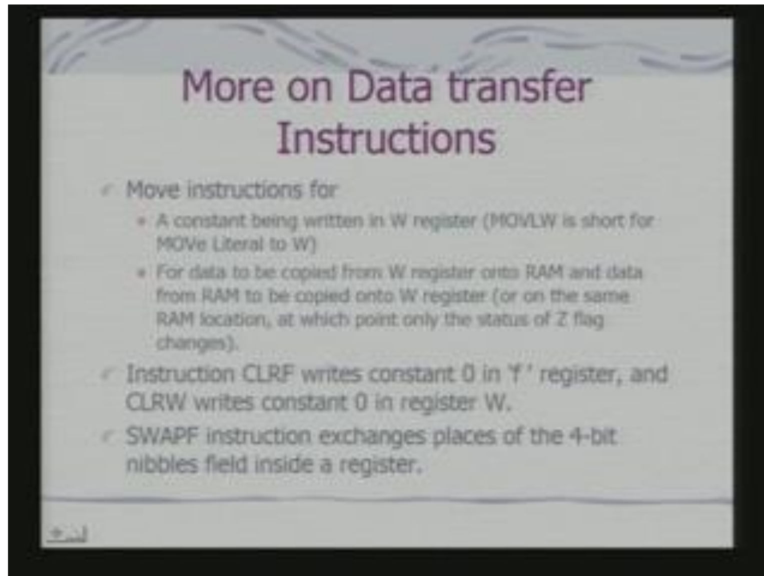
(Refer Slide Time 16:12 min)




Now, in the literal operations, we actually use immediate mode addressing. In this case, these are all direct operations that is where your data is part of instruction itself. So, this literal k which is part of your instruction, you are adding to the content of register W. So if you look at this example, in this example, this 5 is actually added with the content of W and the result would be store in W itself.

In this case, you are moving a particular value into the W register. So now, if you can summarize what we have seen in particular for byte oriented data transfer operations, then we have provision for writing on to the W register through immediate mode.

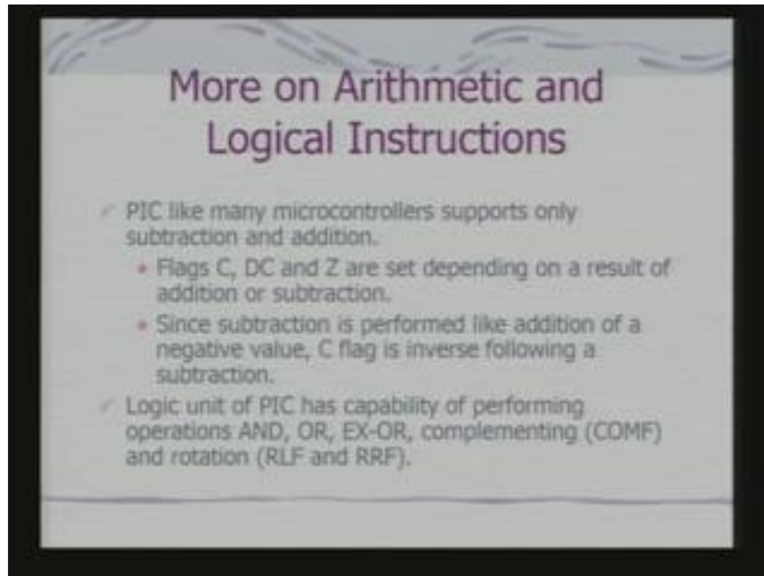
(Refer Slide Time 17:13 min)




That is the literals can be copied to the W register. In fact, we can also have arithmetic operations done with the immediate operands involving double register. Also, we can do data movements, in the sense that, data can be copied from W register on to RAM locations as well as vise versa. So, these are the basic data transfer operations that are supported in the PIC instruction set. We have also seen that there are instructions by which we can clear the content of the registers, that is we can make them all zero. And this is interesting because at times we require the register contents to be cleared for various kinds of initialization tasks. Initialization not only for arithmetic operations, for also doing hardware interfacing operations .The last instruction that we have, we are referring to here is SWAPF instruction. Now, SWAPF instruction, what does it do? It swaps the nibbles of a register that means it exchanges. And this exchange takes place in a single instruction. Typically, when we try to exchange value of two memory locations, we require a third and temporary location and this kind of exchange operation, involving more than one instruction can also lead to various kinds of problems if there is an interrupt occurring in between. Then the SWAPF may not take place correctly. So, if SWAPF can be provided as an atomic instruction, then we can guarantee the SWAPF operation is taking place without any disturbance and the SWAPF is correctly swapping

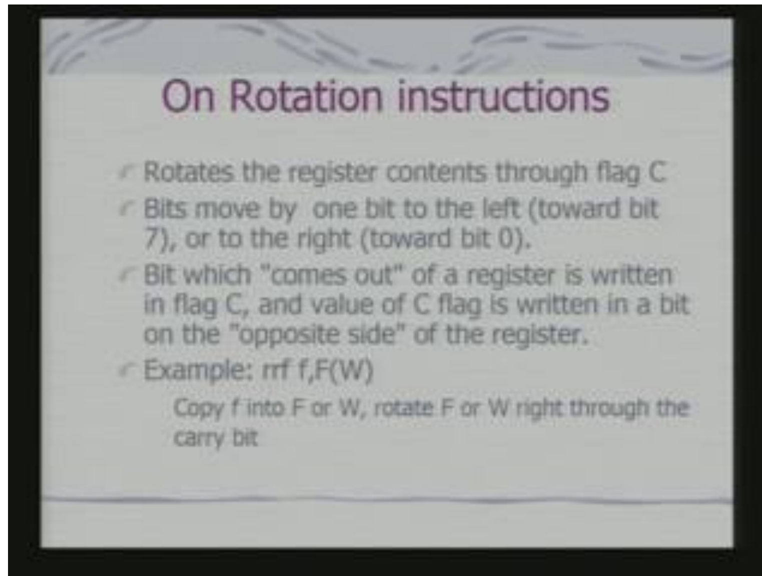
the data values. In case of PIC, we have got SWAPF operations for nibbles and it can be operated in various interesting ways. We can SWAPF the 4 bits, okay; lower four bits which have four bits in the W register and we can also store that content into another register as well. Now, obviously why is PIC supporting SWAPF instruction? SWAPF instruction is required for what we call various kinds of synchronizing construct implementations when we have concurrent processing supported on in an embedded system. These concurrent threads are what, if in from the external world, I need to support multiple operations. Take for example; when we have discussed in introductory course also, I have got a temperature controller which is sensing the temperature. This temperature controller can also be modified by the user to set a value and it also is recording the real time clock. Now, it is doing therefore what, free tasks- these are three concurrent tasks which are to be supported on the embedded system. And if these three tasks are accessing a common memory location or executing a common code, we have to make sure that only one of the tasks is accessing that common memory location or executing the common code. In order to ensure exclusive access, we need to implement various kinds of lock variables. In fact this SWAPF instruction supports or facilitates implementation of this kind of locking scheme- what we call formally as synchronizing primitives.

(Refer Slide Time 21:31 min)



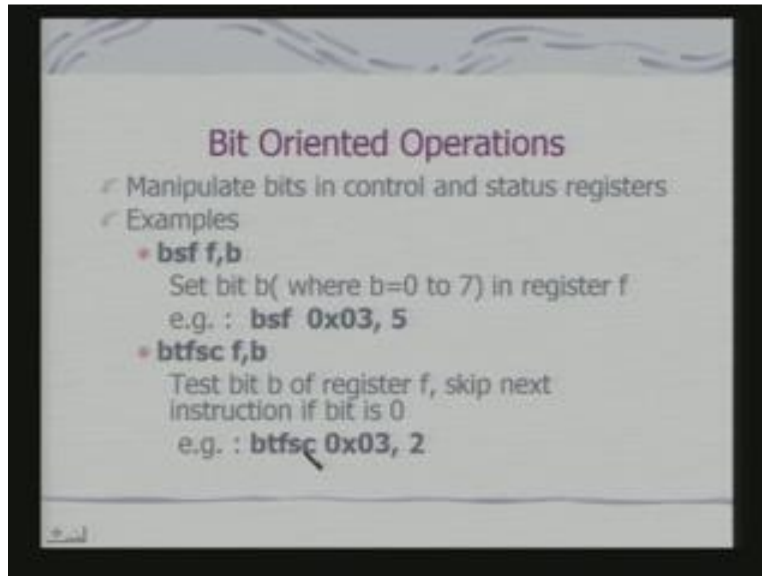

Now, let us look at more information. Let us look at and understand more information about arithmetic and logical instructions. PIC does not support complex arithmetic operations; it supports simple addition and subtraction because PIC in its basic form is not really targeted for such signal processing applications or number branching applications. It is typically targeted for simple control or instrumentation applications. In that case, addition and subtraction operations actually can meet majority of requirements. But I am not telling that multiplication is not required; multiplication and division is also required but then that has to be implemented with the help of these addition and subtraction instructions. And this flags C, DC and Z are affected by these arithmetic instructions. Since subtraction is performed like addition of a negative value, that the C flag behaves like a borrow flag when actually subtraction operation is executed. The logic unit of PIC has a capability of performing operations like AND OR XOR complementing as well as that of rotation; just like any other standard microprocessor or micro-controller. We shall look at rotation instructions; now typically rotation instruction rotates the register content through the flag C. So, carry flag is involved when we are doing the rotations.

(Refer Slide Time 23:17 min)



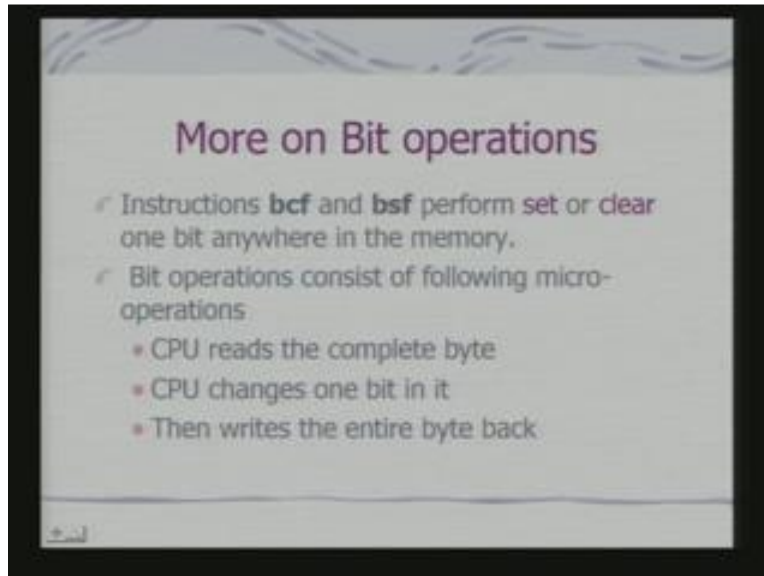

So, bits move by one bit to the left or to the right depending on whether you are doing a rotation left or rotation right instruction. The bit which comes out of a register is written into flag C and the value of C flag is written in a bit on the opposite side of the register, of the target register. That means that original content of C now becomes what, the least significant bit if you are looking in that way, either least significant bit or the most significant bit of the target register. So, this is an example of a rotation instruction that we have shown here. So, copy in this case, you are coping f into F or W and rotate F or W right through the carry bit. Now, F or W is F is register file itself or W is the working register. So, this is how we implement the rotation, rotation instructions. Now, let us look at bit-oriented operations. Now, byte-oriented operations obviously is targeted for primarily data moment as well as arithmetic operations. But, when we look at bit-oriented operations you should understand that why at all this bit-oriented operations have been implemented. Because you can say that, fine if I can do ,if I can select the correct combination, okay of logical operations, I can access and modify the bits. But that becomes again what, a combination of multiple instructions. Now, if I need to do lots of bit manipulations and these bit manipulations are essential when we have hardware flags are hardware interfaces to be manipulated and controlled

(Refer Slide Time 24:18 min)



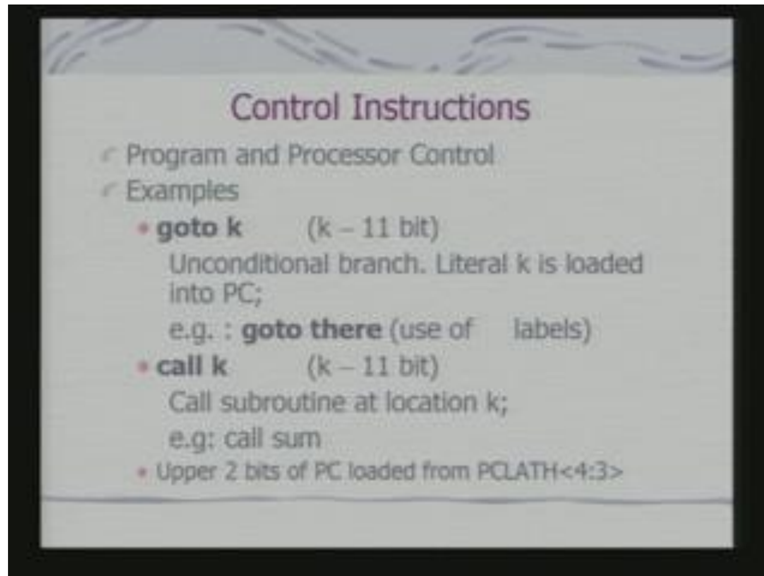

As a micro-controller targeted for embedded applications, it has got number of peripherals on chip; various control bits of this peripherals have to be manipulate to software's. I/O operations are needed to be controlled through software's. These operations, in most of the cases are bit oriented operations and that is why you will find a pretty powerful set of bit oriented instructions supported in PIC microprocessor, PIC processor. So, let us take some examples. This example is a very simple case where you are setting a particular bit in a register f. So, here if you take explicit example, this is the register that we have referring to and we are referring to which bit, we are referring to fifth bit. And what we are doing? We are explicitly setting the corresponding bit, okay. Similarly, we can do this; this is a kind of again an example of a branching instruction, okay. And this branching instruction is on the basis of testing of particular bit of a register f. So, what we say that we skip the next instruction if the bit b is zero in the target register. So, this is the target register and we are actually checking for the bit two, this is again a conditional branch instruction. So, conditional branch instruction means you will be skipping the next instruction only.

(Refer Slide Time 27:34 min)



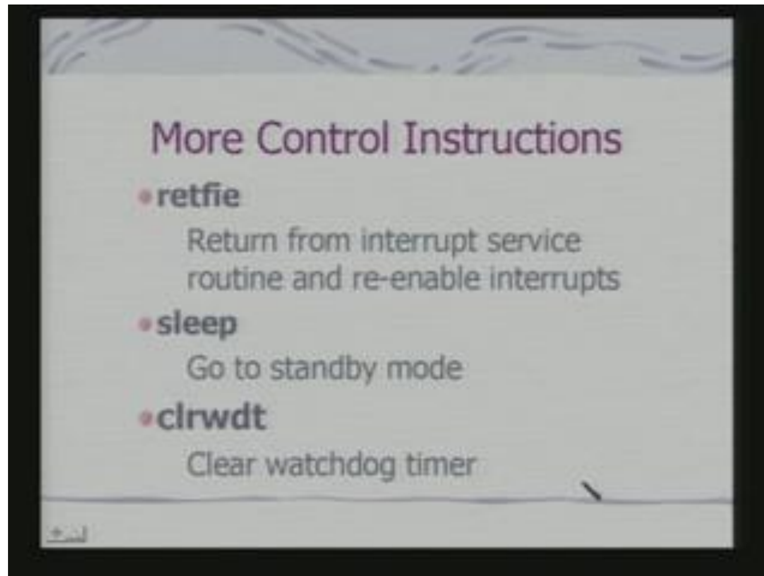

So, instructions we have seen therefore the bcf and bsf perform set or clear one bit anywhere in memory. And bit operations consist of following micro operations. CPU reads the complete byte, CPU changes one bit in it and then writes the entire byte back. So, it is just try to look at this, because this has got various, various interesting significant attached to it. This is read, modify, write and this read, modify, write is taking place in an atomic fashion. This read, modify, write is taking place again in an atomic fashion. Obviously the issue that comes up is that when you are reading a data from in a external port, you are reading a data and you are modifying it. And this modification requires definitely an infinite time period and during that period, if there is a change in the input; because you are reading the data from the port input P from the external source what happens. So, we shall discuss this problem when we actually discuss the peripherals, that peripherals that is digital I/O ports which has supported on PIC. But this read, modify, write is again an atomic operation; the advantage of the atomic operation is, this cannot be interacted by an external interrupt. The control instructions are primarily programmed and processor controlled. In this case GOTO, GOTO is effectively unconditional branch. You can jump from any location to any other location. And in this case we have got an 11 bit and the remaining 2 bits is coming from the PCLATH high

(Refer Slide Time 29:07 min)




And same thing is true for subroutine may upper 2 bits of PC is loaded from PCLATH and these are the bits corresponding to the PCLATH high register. There are some more interesting instructions control instructions in PIC. This is an instruction retfie or return from interrupt with interrupt enable. This happens when you, this is typically executed when you return from an interrupt service routine.

(Refer Slide Time 29:37 min)

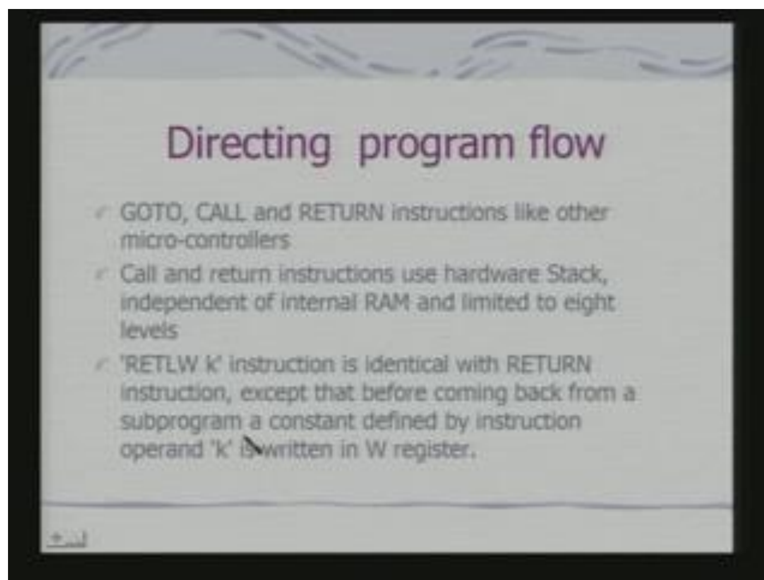



Now, what is the difference between returning from an interrupt service routine and returning from subroutine? Effectively nothing, because in both the cases you need to get the return address from the stack. In case of PIC, you have got hardware stack. So, the return address will be stored in the hardware stack and you have to get the return address from the hardware stack. But in this case, this instruction typically re-enables the interrupts. What does that mean? It means that when an interrupt occurs and you are servicing in the interrupt the interrupt can be disabled. So that the current interrupt service routine is not further interrupted. Now, this is also important for PIC. Why? How this interrupt that is your coupling of the interrupt services, interrupt service routine to be managed?

Why this important for PIC? Because PIC can support only 8, that is on the mid-range processors we have got the stack of fixed size, hardware stack of fixed size. So, I cannot permit any arbitrary nesting of interrupts and subroutine calls, okay. So, I need to enable or disable interrupts whenever I am servicing the interrupts and when we return from an interrupt we should enable the interrupts. So, these retfie instructions typically enable interrupts. Then there is a sleep mode of PIC and this is a power saving mode. That

means when no useful operation is being done, the PIC processor should be put to sleep. So, there is a specific instruction by which the PIC processor can be put to standby mode. The other instruction is relation to watch watchdog timer. You can clear the watchdog timer because I have told you that if watchdog timer finishes its counting, okay or at a after a particular time period and watchdog timer is supposed to interrupt the system. It may reset the system, ideally it should reset the system and I have already discussed why watchdog timer is required. So, if you do not want the watchdog timer to really reset the system, you would like it to be cleared periodically or reprogrammed periodically. So, this cleared watchdog timer is an instruction for clearing the content of the watchdog timer. And if you see how this program flow control instruction or GOTO, CALL and RETURN; see CALL and RETURN instructions obviously using hardware stack and which is independent of the internal RAM and it is limited to 8 levels.

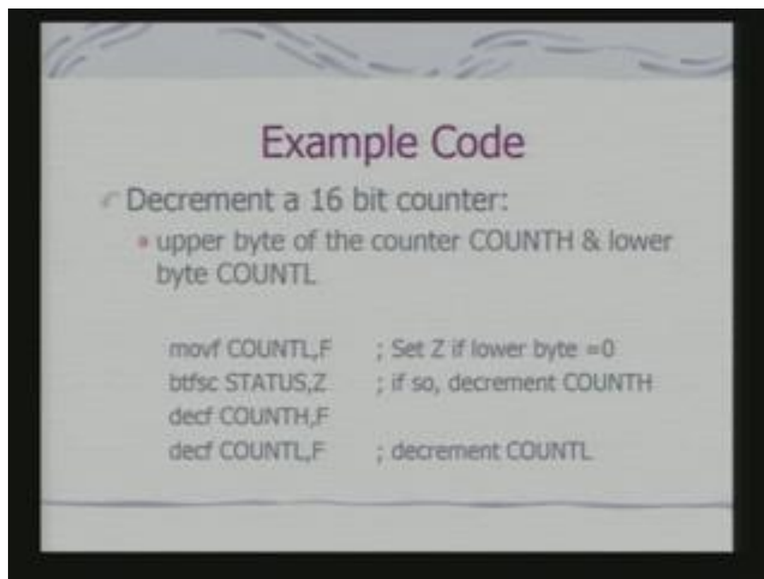
(Refer Slide Time 33:40 min)




So, even if you are considering a scenario, a kind of, kind of hypothetical scenario where you assume there are no external interrupts, then also you can not have arbitrary nesting of subroutine calls because your maximum depth is fixed. There is also a very interesting instruction. Another interesting instruction, this instruction is again a return instruction

except that before coming back from a subprogram a constant defined by instruction operand K is written in the W register. This is again what, this is an interesting form because this is a particular value which is written on to the W register when you are returning from a subroutine call. So, this is your returning a particular value in the subroutine call and using a single instruction. Again, you will find that this instruction enables to you returning a value in W register and this would lead to very interesting tricks that you can play with your code and we shall see such examples now. Let us start looking at some of these example codes; because what you have learnt so far, you have learned instruction. Using these instructions you have to write small codes so that the processor can do useful operations.

(Refer Slide Time 34:56 min)

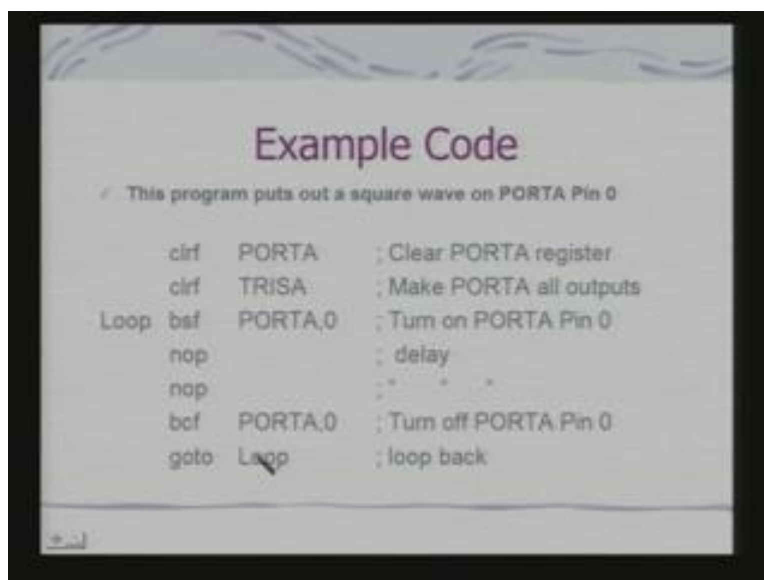



This is a very simple example, that of a 16 bit counter. So, obviously if I have to implement a 16 bit counter what I need to do? I need to use two memory locations and not one because effectively each memory location is of length 8 bits, okay. And in this case, what I am doing? I am using two locations, one I am calling COUNTH and another I am calling COUNDL. And I am decrementing and I am decrementing COUNTH,

COUNTL and here I am checking that if COUNTL is zero, okay and what we are checking for; this we are checking that depending on the condition, we shall be whether decrement COUNTL or we shall be decrementing COUNTH; fine. Now, the interesting feature is, if an interrupt occurs, try to understand this, if an interrupt occurs while this code is in execution, what can happen? Your decrement may not take place correctly. Because your counter is not getting decremented by a atomic operation, is this clear? Because there are 2 bytes and they have to be decremented.

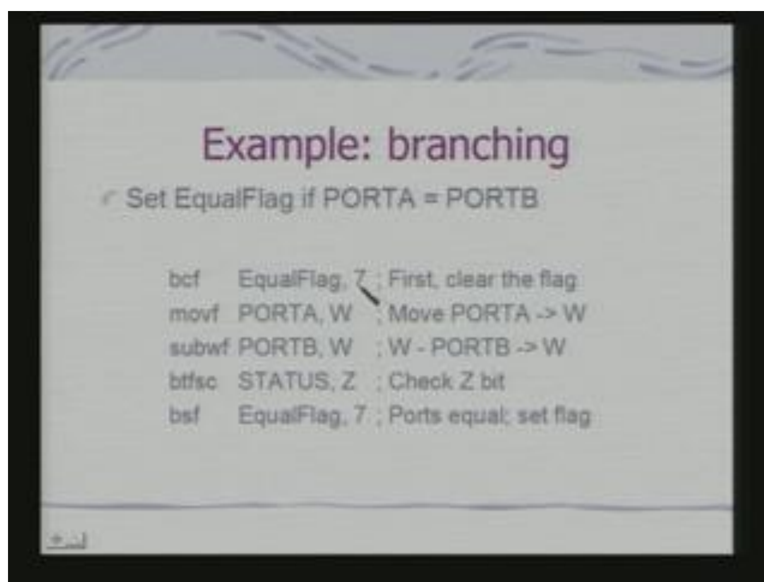
So, what you need to do is that, for the purpose of atomic, if you want to ensure atomic decrement operation on this counter, you need to protect this code. That means you need to disable the interrupt from occurs. Let us look at another example. This example is for producing a square wave on one of these ports because I told you that PIC microcontroller has got one chip port, if you remember the architecture, we have shown ports. And we shall discuss how exactly these ports are to be used when we discuss its peripherals. But this is a small code example which actually interfaces with the port. Now, typically each port has got associated with a register and you clear the content of the register because if you want to write something on to the port you would ideally like the content to be first clear.

(Refer Slide Time 36:43 min)



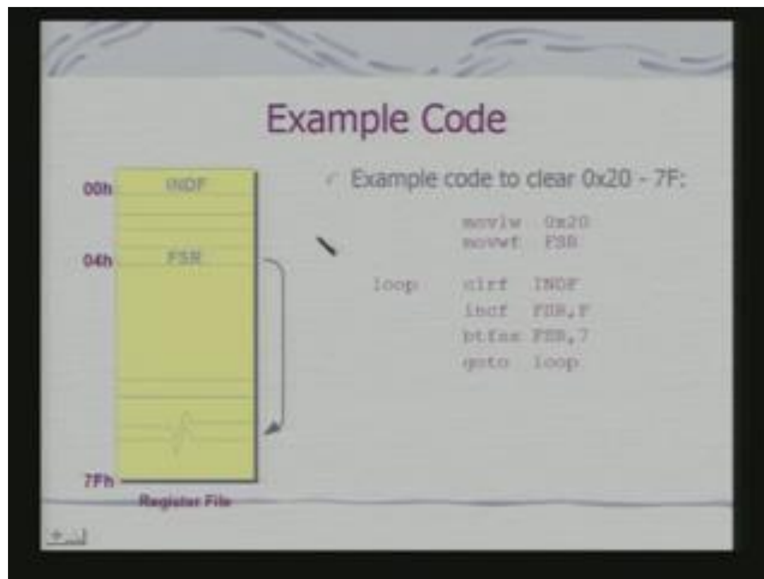
The next thing is, these ports can be I/O ports, okay that means each pin can be configured as input pin or as output pin. Now, if we are doing that, then I need to pre-programme that port and each pin of that port. So in this case, we use a register called TRISA and for each port, since we are talking about PORTA we have got a TRISA register and we have clearing TRISA register making all pins of PORTA as output port. Now, you can see here again, a usage of clear instruction for doing hardware programming .This is an example of usage of clear instruction for doing hardware programming. We are clearing off the TRISA register to make each bit of PORTA as output port. Then next is very straight forward. We are just setting a particular pin zero, okay and then what we are doing? We are going into nop, a kind, a delay; then we are clearing that bit and then we are going back and looping it in through an infinite loop. So, I should get a square wave. And depending on the delay, I have simply used nop; I can add on nop here as well and depending on the delay I can vary the period, I can also vary the duty day issue of this square wave. This is an example of branching and obviously you know that when we are using branching, when we are using branching, the most interesting thing is that I shall be using a typically this skip instruction, okay.

(Refer Slide Time 39:30 min)



Now, in this case we are declaring that a particular location that is this EqualFlag and it is, this is what we are doing, a particular bit in that register and clearing that bit, okay. So, I am calling that bit as a, maybe that bit in that register as, EqualFlag bit and including that. Then I am moving these two that is we are moving the data content of the PORTA to W content of PORTB is now subtracted from W and the result is getting stored in W itself. And then I am doing a branch instruction that is, this is basically checking the Z bit of the STATUS and depending on the Z bit of the STATUS we shall take an action. So, when we find that, when we look at these, that is Z bit is set because when they will be clear, okay that is when both PORTA and PORTB content and equal, then zero bit will be set. If the zero bit is set, what we shall do we shall now set EqualFlag. If the zero bit is clear, we shall skip this instruction. So, this is a simple case of a conditional code by which we can check where the content of the two ports are same amount; in fact any memory location for that matter can be checked in this flag.

(Refer Slide Time 41:01 min)




This is another example and this example uses actually indirect addressing. So, for most of the examples that we have seen, they were using direct addressing. Now, in this case I

would like to clear the location from 20 to 7F in a particular bank, okay. So, what we are doing, we are moving; this is a literal and immediate operand which we are moving into the W register and then what we are doing? We are moving the content to FSR. Now, we are clearing this INDF, okay and then what we are doing? We are incrementing FSR, F actually means the result will be set back and then we are going through this loop, okay. Now, the interesting feature you should note here is this operation. How actually the clearing is taking place. We want to clear the locations from this to this and how will this take place. It will take place because I am clearing INDF. INDF is a indirect addressing register that means, when I am trying to clear INDF it essentially mean that the register memory location of the register pointed to by FSR should get cleared. Is this clear? So, what we are doing, we are accessing this register INDF and as we are accessing the register INDF we are actually clearing the location pointed to by FSR.

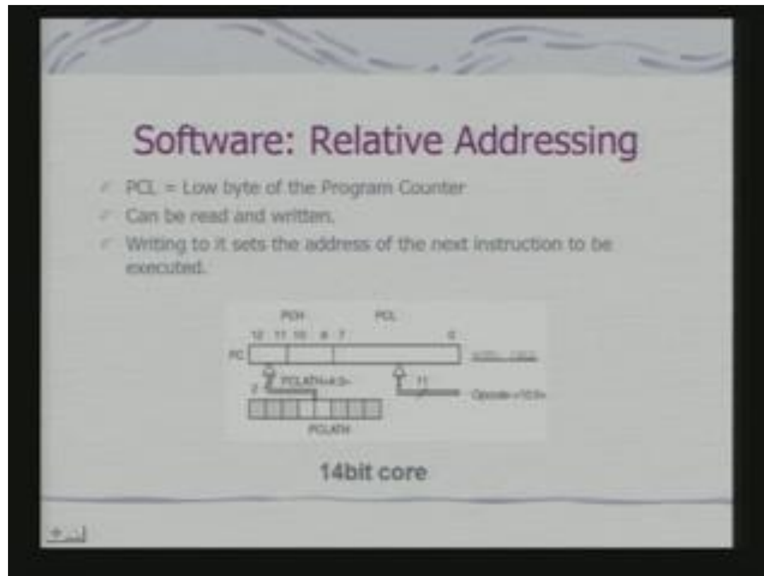
The initial location, FSR was pointing to what 20, okay; so that location will get clear and then what you are doing? You are incrementing FSR and you are going through that loop, okay. As you are going through the loop, what you are, what you are checking here? This is actually; this will do a skip when it is set; so this loop will not be executed when you have reached 7F. If you are trying to increment beyond 7F, you will skip otherwise you will be clearing the content from 20 to 7F location. This is an example of using indirect addressing and in fact what you have seen is the interestingly an application where really indirect addressing is required. Because here, you are changing content of the memory location in a loop. So, you cannot really use the address directly as part of the instruction, because this address has to be constantly changed; that each step we have addresses to be incremented. And as we increment, the address we have to go back and clear the content. So, this is a simple usage of indirect addressing and how it is exactly used in PIC.

Now, you have got the relative addressing and software relative addressing as well, because when you are actually doing any of these: GOTO and CALL; what is the significance of the fact that the two addresses, that the most significant bit here is coming from PC latch high register. It means all your GOTO and CALL is in a sense PC relative; the relative two current PC value, because current PC value the most significant bit is loaded on to PC latch high. So, whatever offset is coming, offset is an 11 bit offset which is part of the instruction. So, all your GOTO and CALL operations are PC relative. This

is the key significance of the fact that the most significant bits are coming from PC latch high register.

Let us look at this example and this is a very interesting example because it illustrates various features which you can use in PIC. Now, here we are using a table lookup and this code is primarily for that purpose. Initially when you are loading with the register W with 4 and then what we are doing? We are calling the subroutine table and table actually would be referred to by the 11 bit offset, okay.

(Refer Slide Time 45:04 min)

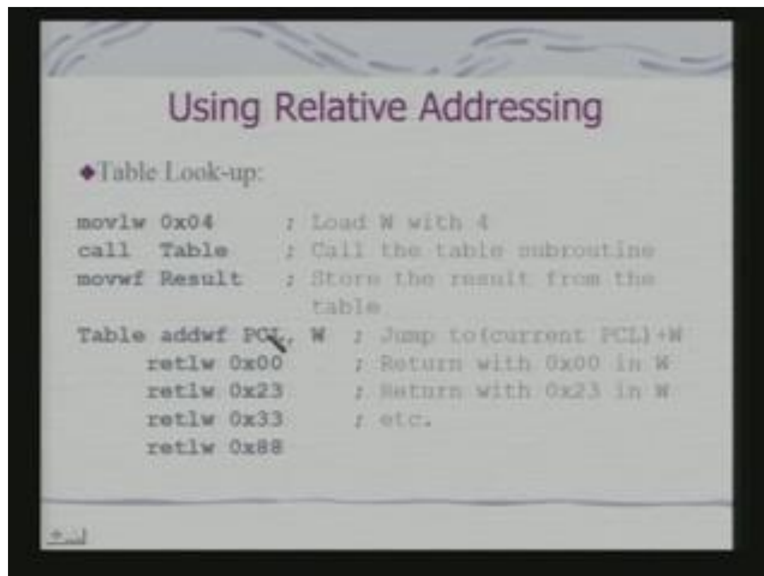



And then next instruction of a call what you are doing? You are moving the result that is you are moving the result; this result will be a result from the table so, you are moving into the result. Now, how you are getting that result; just look at the subroutine table. Now this table subroutine also uses other interesting instruction. These instruction addwf is routine instruction in the sense that it can be used with any register. But here, it is interesting because it is using PC- a program counter, okay. So, current PC low, a program counter part of the program counter as operand. Now, this feature is not typically supported in many instruction set, that is you really cannot use PC as an operand in a instruction. But PIC provides the support for this kind of instruction. So, what it is

doing now, that if you see here the, you will be loading that current content of W okay will be added to PCL and you will be jump to the current PCL plus W. So, the moment the offset is added the current location is this PCL plus W and this has been done through an arithmetic operation.

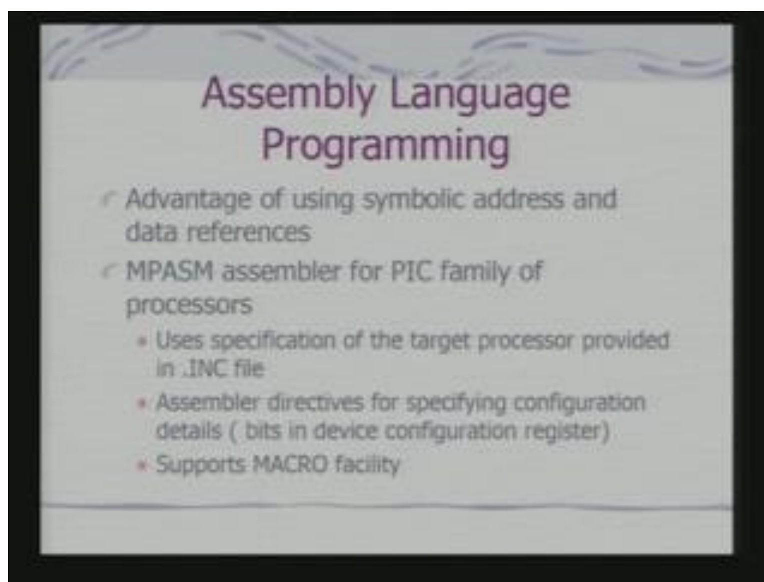
We have not executed any branch instruction at all, this is very-very interesting. We have not executed any GOTO or conditional branching just done an addition operation. And then what is happening? From here we are doing returning, returning with the value loaded in W. And this is really the table lookup operation because actually I am calling this subroutine. If you, if you try to analyze it, I am calling this subroutine with an argument, okay. This argument is being specified here, okay; this argument is used, is being used for indexing on to the table. How the indexing operation is taking place, by adding these value to the PCL. Once you add that value to that PCL the next instruction which is to be executed will be at an appropriate offset, that means it is to be indexed on to that table

(Refer Slide Time 49:22 min)



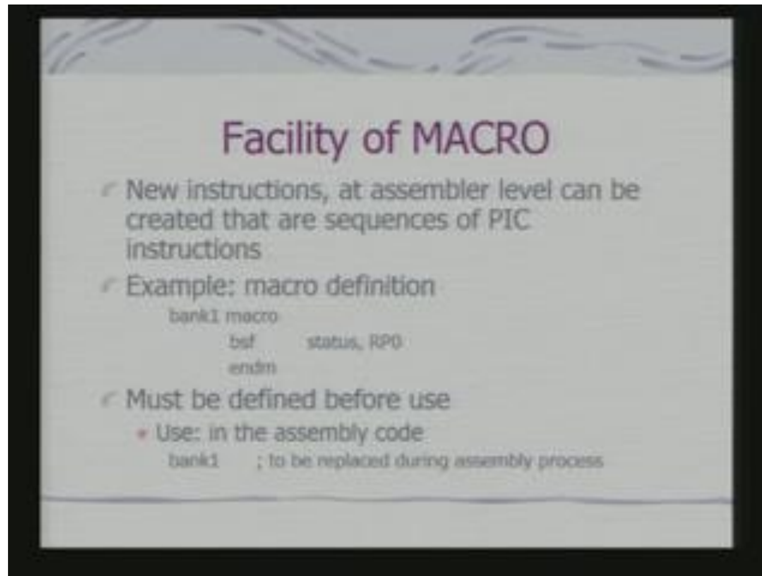
What is the instruction from the table? The table is presented as a return lw instruction. So, what does that mean? In this case you return, return by loading a particular value onto W. So, complete table lookup operation has been implemented and implemented pretty efficiently with pretty high code intensity because here what is happening is that I am adding this content of W to PCL to index on to the table. And which content I am looking on to is being specified by this argument. See, if I change this value, I shall return from the subroutine using some other executing, some other return lw instruction. So, I shall return with a different byte and that is why this is implementing a table lookup operation. So, you have seen more or less variety of instructions and we have also used some of these assembler directives. So, in an assembly language programming you use not only the mnemonics because our discussions have been so far in terms of the mnemonics for the instructions of PIC. And this mnemonics gets translated to its binary code by the assembler, but assembler also supports symbolic references to addresses and data which facilities your programming. But the assembler for PIC has to handle some other very interesting and important task, because PIC is not just a single processor but the family of processors and for that family there will be variations in number of configurationally issues.

(Refer Slide Time 50:38 min)



Variations in the stack size, variations in instruction length, variations and the availability of different peripherals on chip. So, if you have to use an assembler for a family, then that assembler requires to be designed differently. So, in this case you will find for a PIC family, while you are using your code for assembling, you also specify the specification of the target processor. And various information have been provided to this MPASM assembler to what we called .INC file. In fact, if you are using and 8086 assembler or 8085 assembler your processor configuration is pre specified; it is part of the assembler itself. But here, you are using assembler for a variety of processors. So, you have to specify what is your target processor by selecting an appropriate .INC file. This is the very important exception here. Also there is a device configuration register; the device configuration register programs your processors in variety of ways. And these configuration information is also to be provided and in fact these two feature are nothing exceptional for PIC; because you will find when you are using this assemblers for a family of processors at various target applications, you need to provide information about the configuration of the processor and specification of the target processor for assembling or translation of this code. This is true even when we shall look at compliers. This assembler also supports what we call MACRO facility, why. Because, using, you have got 35 instructions; it is very simple. And you might be using a combination of these instructions repeatedly in your code. So, to facilitate programming, what you do? you define MACRO.

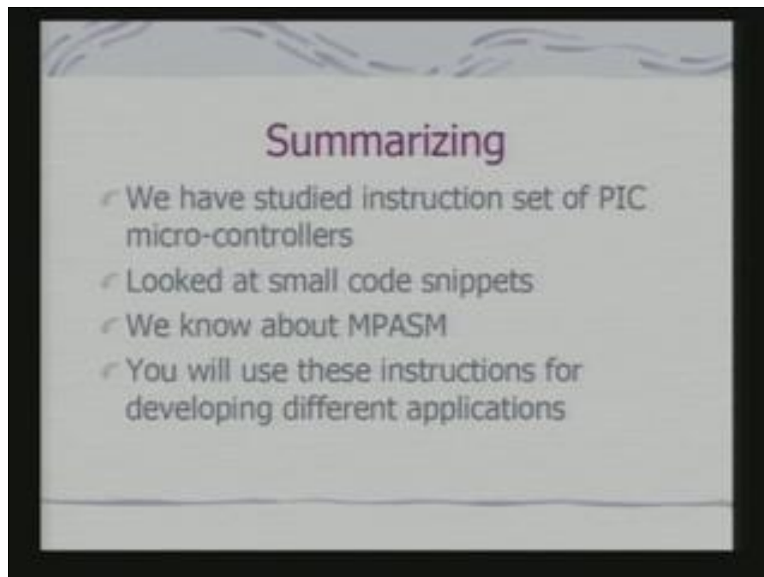
(Refer Slide Time 52:40 min)




So, MACRO is what; just a collection of assembly language instructions which can be called differently with different name by appropriately defining it in your assembly language program. So, this is an example where I have defined a macro, okay. So, I am calling this macro bank1 and this bank one macro, what is the function of this bank1 macro? It simply sets the appropriate bit in the status register so that you can select your bank1. So, in your code what you will be using? You will be simply using bank1; so bank1 would appear like a new instruction. The assembler would expand this instruction that means this will be replaced by the actual instruction of the PIC. Obviously when you use this macro, the length of the code increases but what is the advantage? You are not using subroutine. So, try to please be clear about the difference between subroutine and macro. The subroutine obviously implies an overhead. Why overhead, that is when you are making a call, you are putting PC onto the hardware stack and the next instruction that you have fetched, okay is not what is going to be executed, okay. So, it will be executed depending on the literal value which is stored in the CALL, okay. So, the basic problem you should understand in this case is what; that the overhead which is involved because of this CALL instruction which would now require effectively two cycles instead of one cycle. So, all this overheads should be avoided if you use macro. So, what we have

done today therefore we have studied instructions set PIC micro-controllers. You have look at small codes snippets and we also know about MPASM and how it is different from assemblers and hardware processors. And you will use these instructions for different use for developing different applications which we intent to do using PIC as a micro-controller.

(Refer Slide Time 54:41 min)




And we shall also see examples of these instructions, how they are used in conjunction with different one chip peripherals that the PIC micro-controller offers. So, if you have any question we can discuss them now. See, when we, when we do an addition, so the question what you are telling is that if I add something to PC latch what happens to the carry flag; is that the question? When we add something to the PC low, the PC low, what happens to the overflow? Typically, when you use that, the target is that you should use it in such a way that it points to an appropriate location because overflow is not really useful in any way and this increment, I have shown you an example, this increment is to be used for specific purpose like table lookup. See, if you look into it, the question is when we are executing the addwf plus that is content of PCL being added with W what happens to the next instruction. Obviously the next instruction that you are fetched will

not be executed. Just like any other branch instruction have to push in nop and these effective cycles required for execution of this instruction would be true.

See, let us try to, the question is whether we explicitly push nop; the programmer does not explicitly posh nop in case of any of these branching instructions.

What we are telling as nop, nop is automatically gets push in the pipeline, in the sense that the execution control logic is such that next instruction will not be executed and instruction will be fetched from the new location and then only it could be executed. So, in the program explicitly you do not insert nop. I think here we finish and we shall discuss the PIC peripherals in the next class.