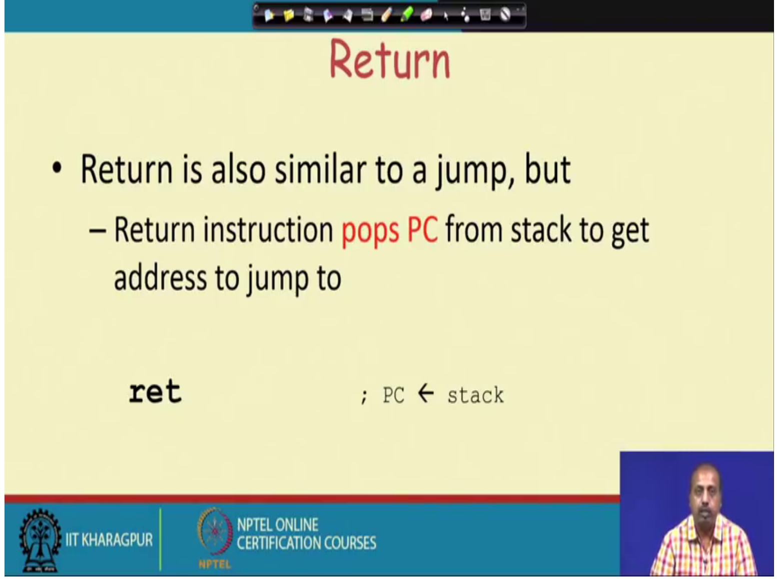
Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

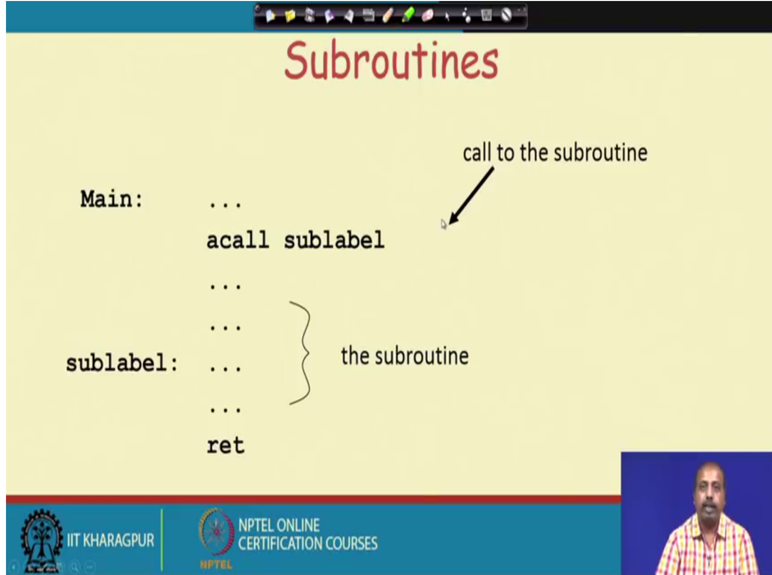
Lecture - 30
8051 Microcontroller (Contd.)

(Refer Slide Time: 00:21)



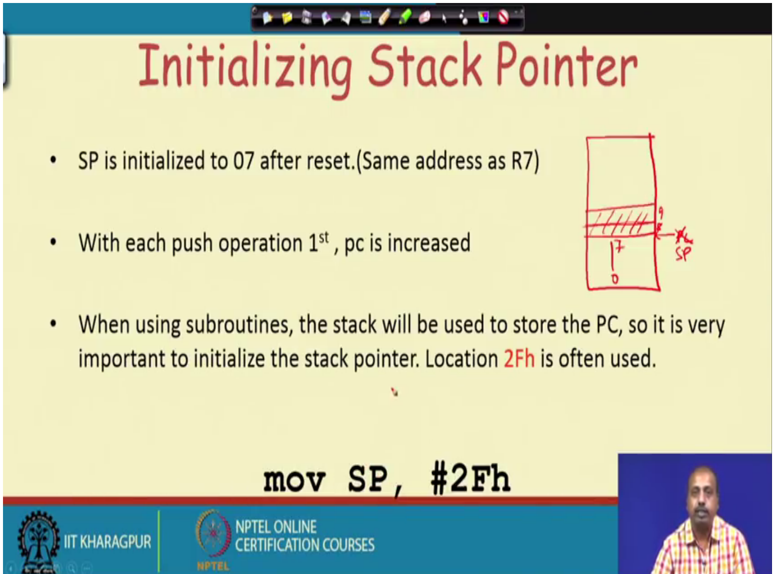
Then the return instructions so it is similar to a jump instruction, but it will pop out the content of the program counter from the stack and then the program counter will be loaded with that value. So, ret is loading the stack top, it is getting the content from the stack top and putting it into the program counter, so that the program counter gets back the point to which it should return and execution continues that way.

(Refer Slide Time: 00:40)



So, when you are writing a subroutine, so we are doing it like this. So, suppose this is the main program. So, after some time we are calling a subroutine called sublabel. So, this is a call sublabel. So, this is the call to the subroutine. So, this is the subroutine body. So, it should be there is a shift, so this bracket should be in this region, so that is the subroutine will be there. So, after this sublabel is over, there is a ret instructions. So, ret instruction will take it back to this that instruction just after the call. So, it will take it to this point.

(Refer Slide Time: 01:15)



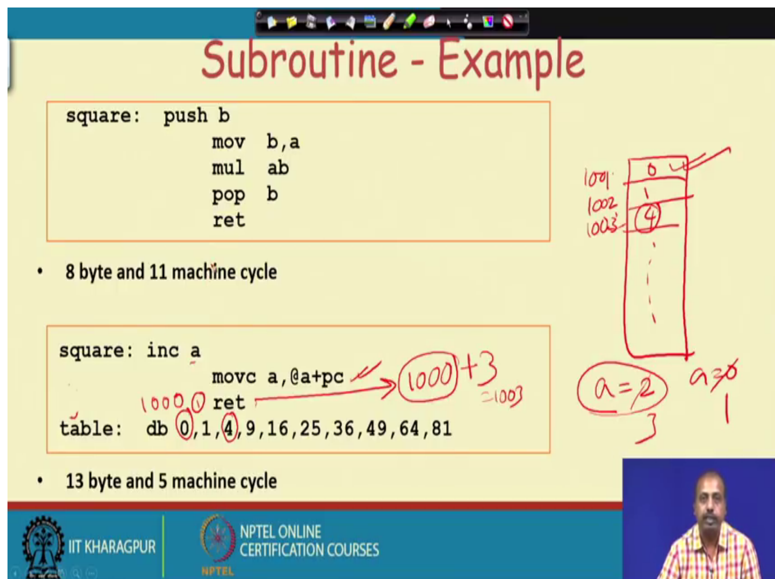
But for doing all these operations, for particularly when you are using subroutines in a program, we have to be careful that the stack pointer is initialized to proper value. Otherwise, this return address will be when the processor will try to save the return address, it may inadvertently modify some of the RAM locations which are useful for the program. So, by default these stack pointer is initialized to 07, after the reset that is same address as the register 7 of bank 0.

Now, with each push operation first, pc is increased. So, when you are first doing the push operation, the program counter will be implemented. So, if you look into this memory, so we remember that there is a register bank 0, which is in the range from 0 to 7, and program counter is also initialized to 7 fine sorry not program counter the stack pointer is initialized to 7. Now, if you are doing a call. So, first this return value has to be saved onto the stack, so the return value will be saved in these two location that is location number 8 and location number 9; in these two locations the program counter value will be saved.

Now, if your program is using something meaningful, so it is doing something meaningful with these locations then those contents will be lost. So, it is better that we initialize this program this stack pointer to some proper values. So, when using subroutines, the stack will be used to store the program counter. So, it is important to initialize the stack pointer. So, in many cases, we will use this to 2Fh as the stack pointer value because after that normally we do not have this register banks and all that.

So, it is advisable that we put it around 2F here, but in your this is just an advice. So, if you find that your program is using even this location then you have to find out a suitable chunk of memory that can be used as stack, and you have to initialize the stack pointer that way. And initializing stack pointer is very simple move sp comma hash 2Fh in this particular case.

(Refer Slide Time: 03:36)



So, next we will see an example of this subroutine. So, this suppose we are writing a subroutine square. So, the square root in what it does is uses it actually squares the number stored in the accumulator. And for that purpose, it will use that multiplication instruction and mul ab. So, it will multiply a by b. Now, since b registered is being used by the subroutine, so it is advisable that we first save the b registered because the main program from where you are calling this square subroutine maybe using these b register also.

So, we first save this b register, then we use that b register to get a copy of a and then we multiply by this mul ab instruction. And at the end the multiplication result is available in the a register. So, it is assumed that since the multiplication result can be contained in 8bits the numbers are small enough, so that the result is contained in 8bit. So, we can ignore the content of b register now. So, we use the pop b instruction, so that this previous value of b is restored and then it will use the ret instruction to return. So, this is an 8 byte in stack program.

So, this is push b is 1 byte, 2 byte, 3 byte, 4 byte. And this I think some instruction has got two bytes overall it is an 8 byte program and execution requires eleven machine cycles. So, for the details of how this bytes and machine cycles are coming as we have noted in 8085 also, so you have to consult the user manual to know the sizes of individual instructions and the number of machine cycles they take.

Now, another way of doing this multiplication this square operation is by another this is the another program square. So, it increments a then movc a comma at the ret a plus pc. So, this will be. So, this table is located just after the program. So, you see that program counter at this point program counter has got a value which is pointing to the at the ret instruction. Now, if we are having the value 0, then implement a will have the value 1, so this is a value 0. So, if the value was 0, the return valve is also 0. If a value was 1, then after increment, so it will become 2. So, these the program counter value at this point suppose at this point the program counter value is say 1000.

Then what happens is that since this ret is a 2-byte instruction, so this table will be stored from location 1000 sorry this ret is a is 1 byte instruction. So, this table will be stored from memory location 1000, so memory location 1000 will get the value 0; then 1001 will get the value of 1, then 1002 will get the value 4, so that way it will continue fine.

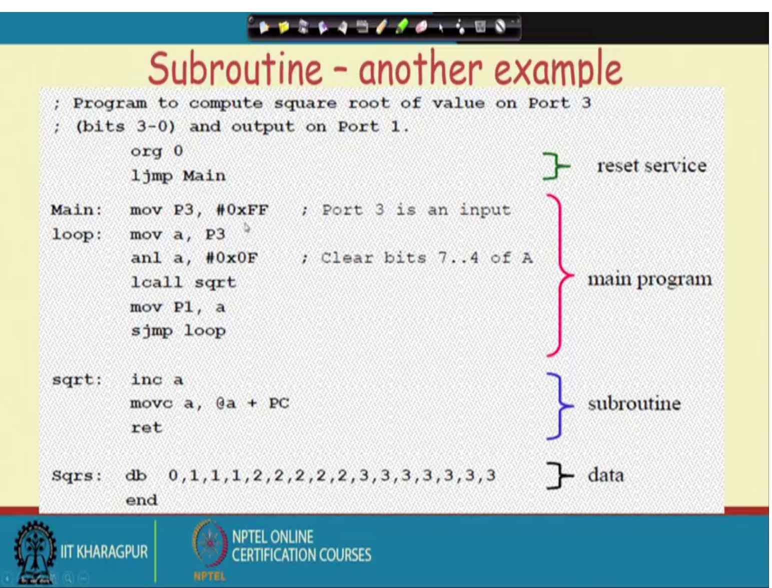
Now, suppose the value that we want to square suppose a is equal to say 2, so we want to get the value 4. So, when this instruction has been fetched, so program counter value is equal to 1000 because that is the address of this ret instruction. So, with that 1000, we will be adding what so we will be adding the value implement a. So, a has become 3. So, it will become 1003. So, 1003, so 1000 one two three so that is wrong actually this ret instruction is 2 byte. So, ret instruction is two byte. So, this is actually to 1002.

No, sorry, sorry, sorry, sorry absolutely sorry. So, this is ret instruction is one byte. So, this is this is going to be this address is going to be 1000, this address is 1000. So, this is 1 byte instruction. So, my table will start at 1001, my table will start at 1001. So, this is 1001, this is 1002, this is 1003 fine. Now, what happens is that a equal to 2, so a equal to 2, so increment a will make a equal to 3, then this is a plus pc the 1000 plus 3 - 1003. So, it will access the location 1003 and a 1003 you see we have got the value four. So, it gets the square value 4.

On the other hand, if the value suppose a is equal to 0, in that case after increment a, a will become equal to 1, now it will be 1000 plus 1. So, it will be accessing this location 1001, which will get this 0. So, say this program also will be computing the same square, but it is faster, the program is faster you see that this required five machine cycles compared to eleven machine cycles here, because the multiplication instruction takes a number of machine cycles. So, this way we can use the this program for getting the

multiplication this squaring done faster. Of course, you can say that this is limited because we are just going up to the value 9, so 0 to 9 only those squares can be computed by this program not beyond that.

(Refer Slide Time: 09:09)



So, this is another example. So, what it does is that it will complete the square root of value on port 3 and output the value on port 1. So, it will compute the square root. So, how will it do it. So, this is the main program. So, p 3 first of all we have to get the value from port 3. So, it says that the value available on port 3. Now, since we want to read the content of port 3, port 3 has to be configured as an input port. So, we have to output all one on to port 3 that we have seen in the port discussion. So, this initializes port 3 bits to be input. And the next instruction it will read the content of this p 3 registered into a.

Now, we are clearing the bits seven to four by ending with 0 x 0 f. So, you see that this 0 will turn off the bits 4 to 7 only 0, 1, 2, 3 there they will be there. Now, we are calling this square root program. So, this square root program. So, this will be implemented it will implement a and then this it will be movc a comma a at the ret pc. So, this way it computes the value depending upon the number so up to 9, so it has got the corresponding digit that should come so if you are taking that lower ordered byte.

And then after that it is outputting that number to a onto the port p 1, and then it is sjmp l 1, so sjmp loop. So, it will again come here get the next number and it will do continue with the program. So, correctness of the program, you can verify, but the operations are

like that. So, this is typically to this is just to show you how the subroutines can be written. So, this is the way we can write down the subroutine.

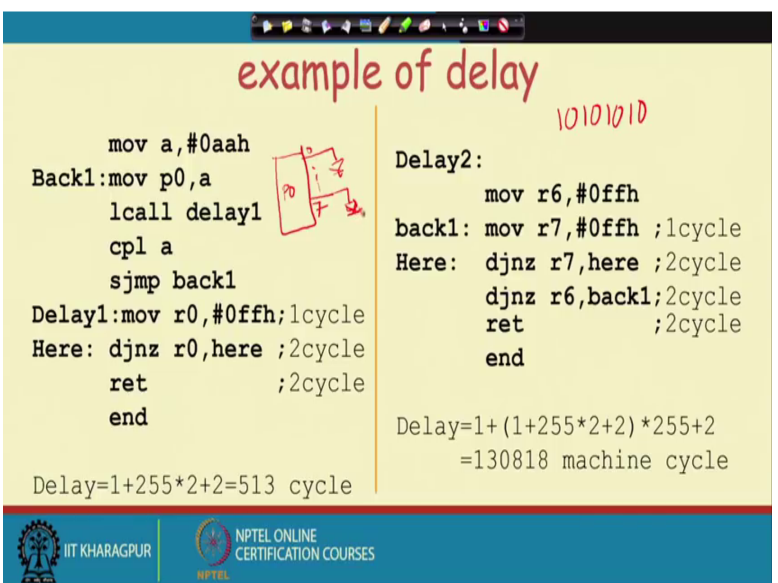
(Refer Slide Time: 11:00)

Why Subroutines?

- Subroutines allow us to have "structured" assembly language programs.
- This is useful for breaking a large design into manageable parts.
- It saves code space when subroutines can be called many times in the same program.

So, why should we use subroutine. So, they will allow us to have structured assembly language program. So, we can divide the whole job into a set of subroutines and that is useful for making the program manageable, and it saves code space because subroutines may be called several times in the say in the program for doing up. So, we do not need to have so much of space, for every time if you want to repeat the same piece of code then the space requirement of the program will be high.

(Refer Slide Time: 11:34)



So, an example of say delay routine, so this is how is it being done. So, this is move a comma 0, so 0 aah is moved onto a registered. Then we are outputting this a register value onto this p 0. So, this p 0 is getting the value. So, in actually this program, so this is what this program does is, so if you look into the pattern is a, so a is 10, so it is 10101010. So, ideally this program is something like this that we have got this port p 0 from which we have got this led is connected. So, we have got this type of led is connected from bit number zero through bit number seven.

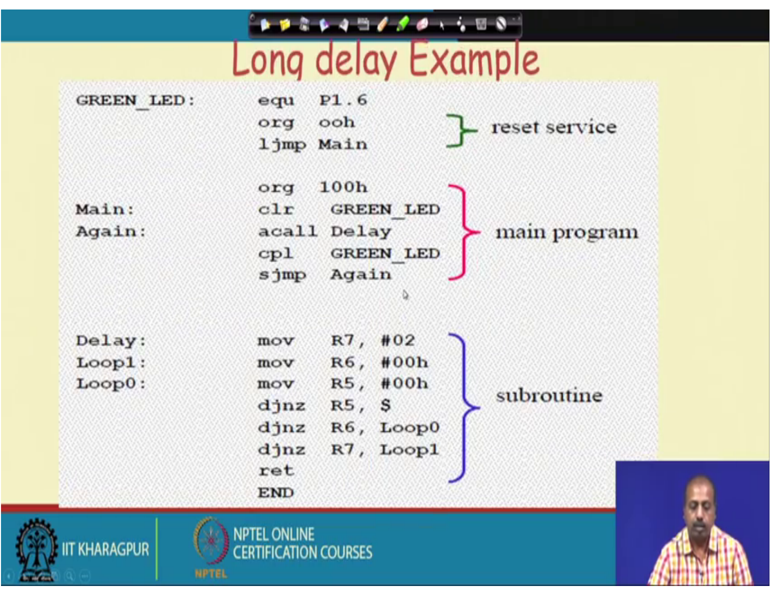
And then this led's, so they are actually. So, you want to blink the alternate led's, so that is why that is what this program is doing. So, what it do what is he doing first it is taking the value 000 into aah register and outputting the value to port 0, so alternate leds are turned on then we have to put a delay there. So, this routine is the l call delay one. So, this will call this delay routine then it will complement this a register and then it will go back to this. So, when its complements then the led's, which were off previously will be on now, and those which were on previously will be off now. So, it will be doing like this.

Now, how this delay routine is working. So, r 0 registered is getting the value ffh that is 255, and then we are decrementing jump on nonzero r 0 here. So, r 0 will be decremented and it will be jump it will be looping at this point till r 0 becomes 0 for 255 cycles this will happen, and then it will return. So, this puts a small delay into this

display, so that it is visible to the human eye. And the total delay that is produced you can find out this mov r 0 0 ffh this (Refer Time: 13:41) one cycle then this instruction djnz r 0 here, so it takes to cycles and that is repeated 255 times and so the plus the last return that takes two cycles to total all together it takes 513 cycles.

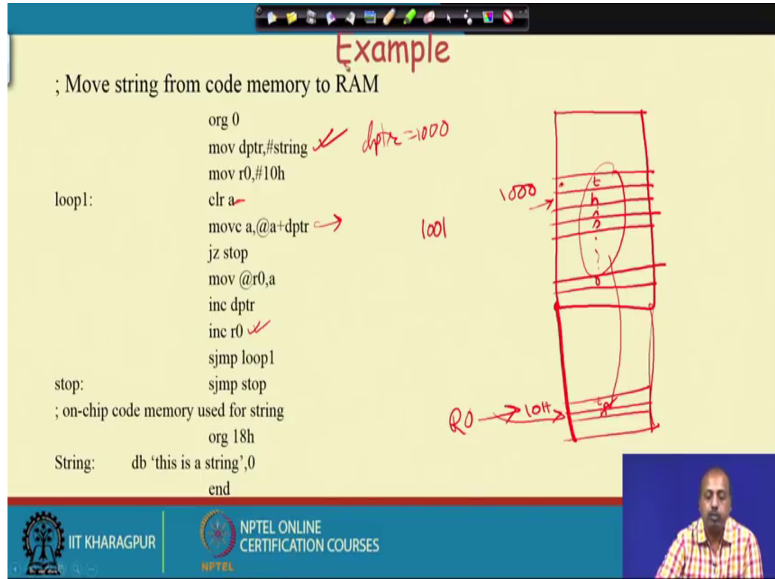
On the other hand, if you want a larger delay. So, then we can have another delay routine. So, here we have got r 6 0 ffh then r 7 0 ffh then djnz r 7 here. So, this is doing it here. Once it is over, so it we are doing a djnz r 6 to back one. So, this is putting two delays r 2 r register pair r 6 and r 7. And total number of cycles, you can compute in a similar fashion, so it is 130818 machine cycles. So, once you know the crystal frequency, so you can find out what is the actual delay that is produced by this routine. So, this one has got much higher delay.

(Refer Slide Time: 14:39)



Next, so this is the long delay example. So, we have got this green led then sjmp again we can just go through this program, and see that how this long delay is being used.

(Refer Slide Time: 14:49)



Then we take another example where we can move a string from code memory to ram. So, this dptr hash string. So, dptr will get a number from where this string is stored. So, this is the string. So, this string has got an address. So, when you say hash string, the corresponding address gets loaded in corresponding address will be coming here and dptr will get that address. And r 0 we are loading how many characters are there. So, this is actually 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 characters. So, there are 16 characters. So, it is 01h and it is terminated with a null character here. So, you are saying that there are 6 characters

Then we clear the a register then move ca comma at the ret a plus dptr. So, dptr was pointing to this character t. So, in terms of memory you can say that is if this is the memory and suppose from memory location 1000, we have got these characters stored. So, this is the character t, this is the character h, this is the character i, s, so it goes like this and it is ended with 0. So, this is the string. So, when you execute say this instruction then dptr gets the value 1000. So, dptr peter becomes equal to 1000.

Now, if you look into say this instruction, what it is saying is it is a plus dptr. So, a value is 0 at this point a plus dptr, so it will be accessing this location, so that character will be moved to a. So, a will get the character t and jump on 0 to stop. So, if you have got is zero then it will be ending, so that is over. Otherwise, it implements r 0, so, it gets the content of a register copied onto the location pointed to by r 0. So, r 0 setting is sorry r 0

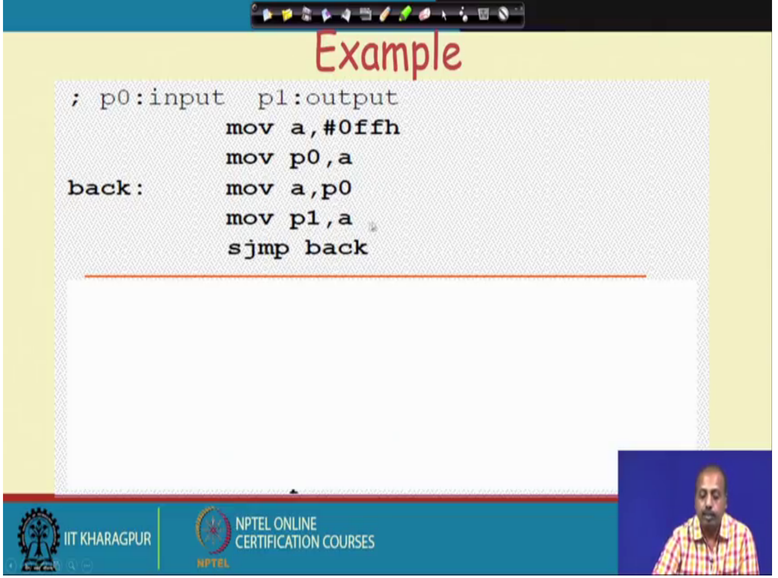
is 10 H, so it is assumed that this r 0 will be pointing to the location where you really want to move the this is the 10 H address. So, r 0 is pointing to this meaning that I want to move this string from this location to this location copy it into the ram.

So, this is move r at the rate r 0 comma a, so this t character was there in a register, so that will come to this location 10. Then implement dptr. So, dptr value will be incremented then r increment r 0. So, r 0 will also incremented. Then sjmp loop one, so it will come back here. So, it will again clear smp loop one. So, it will be it should not be here actually yeah clear a is yeah, it should come here. So, now, with this dptr is now dptr value is now 1001.

So, with the dptr, so this again that a will be added, so that way it will become it will become 1001. So, it will be pointing to the next location. So, that way it will be getting the next character into the a register and that character will be moving to the next location pointed to be r 0. So, the here we have incremented r 0. So, r 0 is now pointing to the next location. So, the h character will be copied here. So, t character was copied there h character will be copied here.

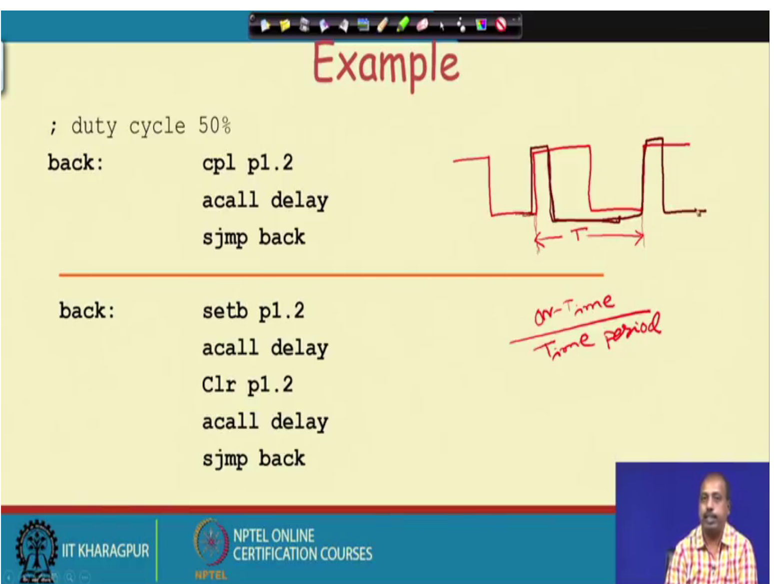
So, this way it goes on. So, finally, so it will be jump on zero to stop when it reaches this zero character. So, it will be jumping to this stop and here at this point the program in an infinite loop. So, it is continually sjpm stops. So, it is in an infinite loop at this point. So, we can use this type of program for moving string from code memory to ram.

(Refer Slide Time: 19:05)



Next, we will look into another program. So, this is outputting it is setting this port zero into input mode reading the value from port zero into a register, and then it is putting the value that is read in a into the p 1 register. So, this is basically copying the value coming in p 0 port to the p 1 port. So, from p 0 port the value is copied into p 1 port. So, you cannot directly transfer between p 0 and p 1, if you do it via a register.

(Refer Slide Time: 19:34)



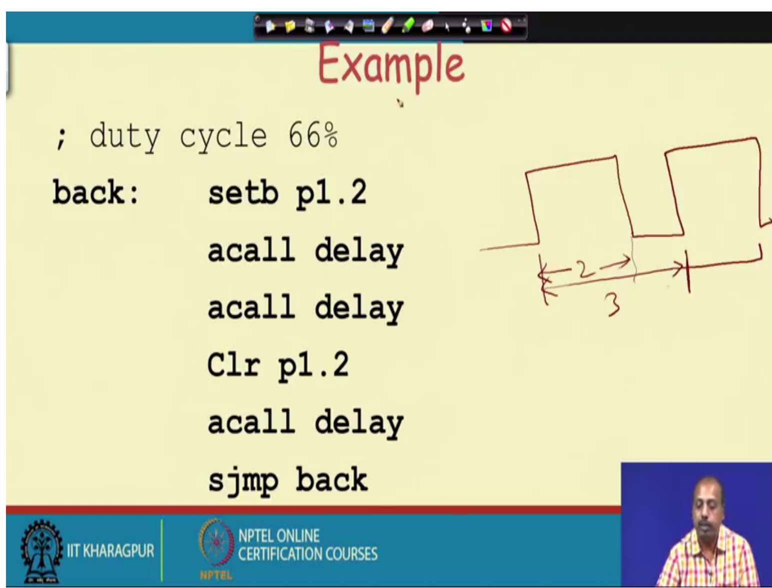
Next, we talk about a program that produces a square wave of duty cycle 50 percent. So, if I have got a square wave, so it has got square wave is like this. Now, this on period to off period, so this is if I say that this is the start time of a period then this is the corresponding n time of a period. So, this is the time period t that we have. Now, in this time period, so duty cycle means the on time the on time divided by time period. So, if I say 50 percent duty cycle that means, for half of the time, the signal high and half of the time signal is low.

You can have some other type of wave like say at this point the signal is high and maybe it comes down at this time it comes down, and then it remains low for this much of time say the wave is like this. So, the wave is like this. So, here the duty cycle is much less, because the wave is high for much less amount of time where the time period is same. So, this way we can have different duty cycle. So, if you are trying to generate a wave of duty cycle 50 percent, then we can do it like this. So, we want to generate on port ones

bit number 2. So, compliment port ones bit number 2 a call delay. So, a call delay will be produced some delay there then sjmp back. So, it will be coming back here.

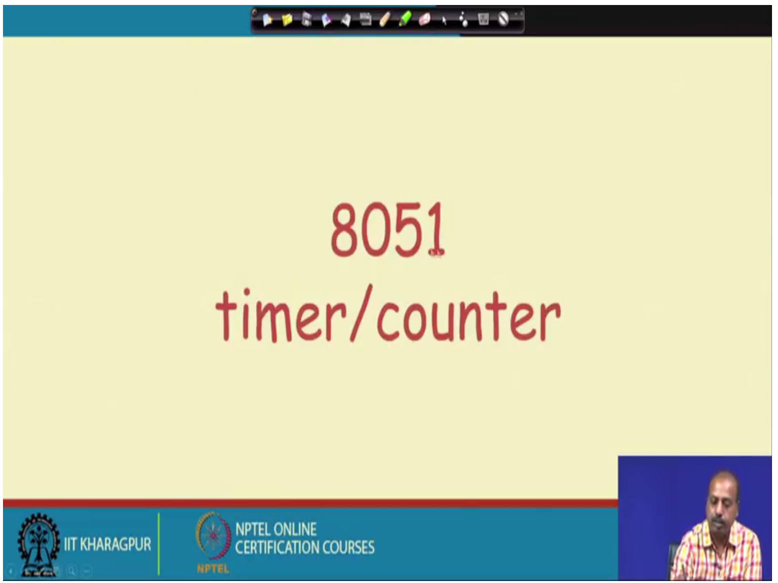
Or we can do it like this we can do it explicit set and this set. So, we can set bit 1.2 then put a delay then clear 1.2 then again call delay and then sjmp back. So, here the difference is that in the first case, it depends whatever be the random value the port has so it will be complimenting that value. So, we are not very sure about the phase of the signal at the beginning, but in the second case the phase of the signal is initially high and then it goes to low, so that way that the difference between the two cases; otherwise they are same.

(Refer Slide Time: 22:04)



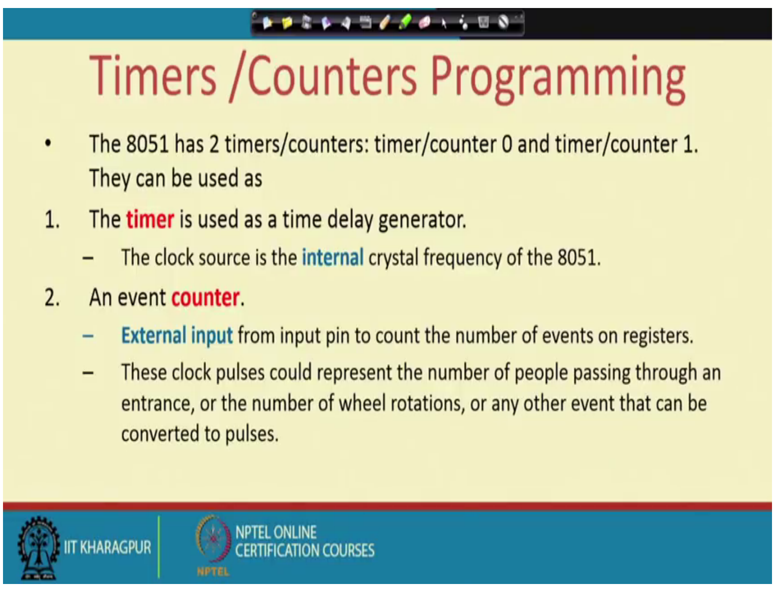
Another example suppose we want to make a duty cycle of 66 percent. So, if I put the clock be high for two times and two time units if the clock is high then it should be low for one time unit. In that case you can get a duty cycle of 66 percent because now this is the on-time is say if the total time period is say sorry.

(Refer Slide Time: 22:26)



If the total time period is say.

(Refer Slide Time: 22:30)



If the total time period is say 3 and then out of that for two time unit, so we are remaining at for two time unit we are remaining at high; and for one time unit we are going to low. So, the signal is like this. For two time unit, it is high; then for one time unit, it is low. Two-time unit, it is high; one time unit is low; then again it is going high for two time units; then it is becoming low, so that we can have it. So, this is basically the two-third thing. So, if this is totally is say if this total is say 2, if this total is say 3,

then we can have say up to this much. So, if this is a 3, out of that, so this much is 2 in terms of sometime unit, so we get a duty cycle of 66 percent.

So, how do we do it? We set this port ones bit number two then call the delay routine then put two delays. If there is a one call to the delay routine, so if I assume that it produces a delay of one time unit then I am giving two calls. So, I will get two delays then clear that bit. So, it will become low and then I am putting a delay of one unit. So, this acall delay. So, it will produce a delay of one unit for that subroutine the delay routine. And then it will be sjmp back. So, you will come back this point. Now, it will again set bit to one point you know set bit of port 1.2 and accordingly this pattern will be generated, so that is one way by which we can generate delays and all that.

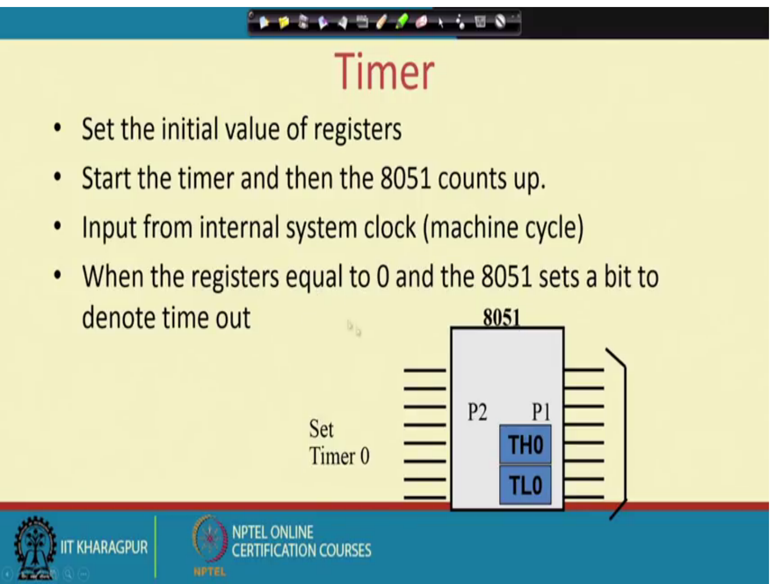
But many times what happens is that for embedded applications you need some precise delay. For example, if you are employing this 8051 microcontroller for designing one traffic light controlling system, so then the traffic signals are to be on for certain periods of time. So, it is not proportional, so some fixed amount so maybe 1 minute or 2 minute or 30 seconds something like that so, but that value should match with the exact clock. So, for those cases, it is not possible to have this accurate delays by means of this software routines that we are looking into so far. So, we have to use some sort of hardware called timers for that purpose.

So, 8051 has got built in timers they can be used either as timer or as counter and then we can produce some specific delays using those routines using those hardware. So, 8051 has got two timers counters a timer counter 0 and timer counter 1. And they can be used as a timer a for time delay generation. And in that case, the clock source is the internal crystal frequency of 8051. So, it will count time in terms of the internal crystal clock frequency. So, once we know that crystal frequency, so you can find out like how to how much value that timer should have. So, we will look into those calculations to produce some exact delays, so that is one type of utility.

Other possibilities you can also use it as an event counter like you say if there is a conveyor belt onto which items are passing, and you want to make a count like how much how many items are passed. So, for that purpose, so every event, so that can generate an external pulse, and that external pulse maybe counted by this timer counter

module. So, in that case we have to configure that the module as a counter and that pulse instead of coming from internal crystal. So, it will come from the this outside bit.

(Refer Slide Time: 26:39)



So, this is the generic structure like we like we have got this timer zero. So, timer zero it is. So, you have got this TL0 and TH0. So, this TL0 and. So, these are the two registers that will hold the time value. And we can set the initial value for the registers TH0 and TL0, then we can start the timer when 8051 counts up. So, if you start the timer and then so from that point onward.

So, once you start it, so it will start counting up. The input will come from the internal system clock, so system clock will give us the input. So, on that basis, so it will be counting it and when the register content will become 0, so it will set one output bit to denote that a timeout has occurred. So, based on that we can again take some decision to run some other piece of code or produce some external signal for that purpose.