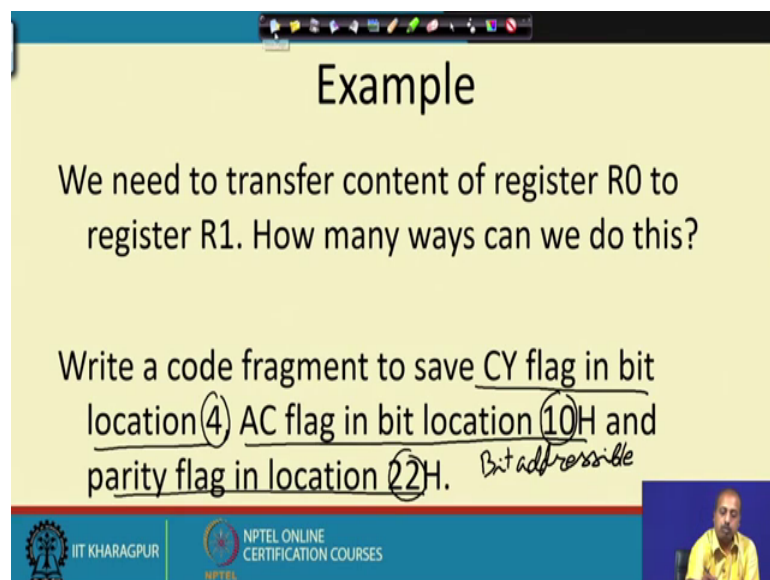


Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 37
8051 Programming Examples (Contd.)

The next example that we look into is to transfer the content of register R 0 to R 1. So, this is a simple thing that we need to do, but we have to say in how many ways we can do this thing what are the different assembly language programming techniques in 8051 that can be utilized for doing this simple operation that is transferring the content of register R 0 to register R 1.

(Refer Slide Time: 00:37)



Example

We need to transfer content of register R0 to register R1. How many ways can we do this?

Write a code fragment to save CY flag in bit location 4 AC flag in bit location 10H and parity flag in location 22H. *Bit addressable*

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

(Refer Slide Time: 00:44)

Current Bank registers ←
Irrespective of Bank selection ←

1. `MOV R1, R0` $\Rightarrow R1 \leftarrow R0$
2. `MOV @R0, R0` $\Rightarrow R1 \leftarrow R0$
3. `MOV R1, 0` $\Rightarrow R1 \leftarrow m[0] \Rightarrow R0$
4. `MOV 1, 0` $\Rightarrow m[R1] \leftarrow m[R0]$
5. `MOV R1, #1` $\Rightarrow R1 = 1$
`MOV @R1, 0` $\Rightarrow m[1] \leftarrow m[0]$

Now, if we want to do this then one the first method is to have the move instruction directly say move R 1 comma R 0. So, this will move the content of R 0 into the R 1 register; so, this is one possible technique of specifying. The second possible way of specifying this maybe we can write like say move 1 comma R 0. So, this will do the same thing like when we are say at like 1; so, this is taken as the direct memory address and as a result the R 1 register will get R 0.

The third possible way of doing it may be; just the opposite that is move R 1. So, this is registered, but this is a memory location; so, here also same thing R 1 gets the content of ram location 0, which is basically the R 0 register. Then another way of doing it is when we are specify both the registers as memory operands; like we simply write like move 1 comma 0.

So, when you say move 1 comma 0; so that is memory 1 will get the content of memory 0 which is essentially. So, memory 1 is nothing, but R 1 memory 0 is nothing, but R 0; so the same operation is done. Another way you can write it is a slightly tricky; we do it like this move R 1 comma the immediate value 1; the hash 1, the immediate 1. And then we say move at the rate R 1 comma 0; so, what we are doing? So, first in R 1 this statement we will make R 1 equal to 1 and in this statement, it will be moving the content of this memory location 0; to the memory location pointed content by R 1.

So, memory location content by R 1; so, that we will get the content of memory location 0, so effectively the R 1 register will get the value of R 0. So, out of all these instructions you see that this one is definite in the sense that; this is modified, this is irrespective of memory banks; so, this one is irrespective of memory banks.

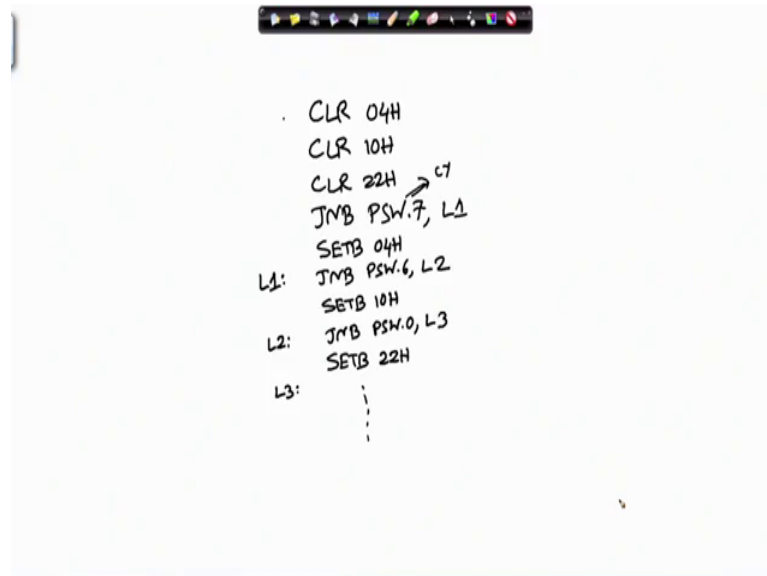
So, it will move it from; it will operate on the bank 0; irrespective of bank selection. So, it will be operating on the bank 0 registers; on the other hand other instructions that we have, so, they will depend on the bank selection. So, all the other instructions; so, they will be talking about moving with respect to the current bank, so they will always access the current bank registers.

So, this way if you are willing to access say registers of a particular bank. So, you can change the address and you will use this particular mode. So, if you are looking for a any arbitrary bank register movement; so you should use this instruction, otherwise you can have many other options. Next we will look into another program which is to write a code fragment to save the carry flag; it will save the carry flag in bit location 4, auxiliary carry in bit location 10 and parity in bit location 22 H.

So, this 4, 10 and this 22 H; so, these are all in the bit addressable memory; so, these are actually the beat numbers. So, you remember that we talked about this bit addressable memory and then if every beat has got an address. So, we are trying to save the carry flag, auxiliary carry flag and parity flag at 3 different locations.

So, how to do that? Is like this.

(Refer Slide Time: 04:50)



So, first we clear those locations first we say clear 0 4 H; so, this will clear the bit 0 4. We also clear the bit 1 0 H that 1 0 location will also cleared and clear 22 H. So, that will also clear the third bit that we have in our question. Now we will check the individual bits they carry auxiliary carry and parity and accordingly set the bits. So, for the carry part we can write like this say jump on naught not bit PSW dot 7 comma L 1.

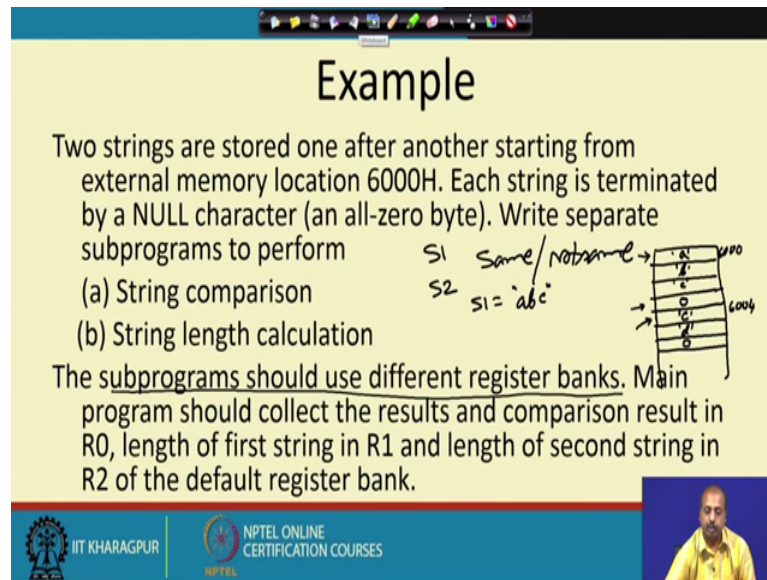
So, if this bit is not set; so, PSW dot 7 this corresponds to the carry bit. So, if you look into this beta addressable memory and this PSW dot 7 is the carry bit, then if it is not 0 in that case we need to set this location; this 0 4 location should be set. So, then we will be doing like set bit 0 4 H; so, it will be setting the bit address 0 4 H. Otherwise, we will come to the level L 1 and where we need to check whether the same thing with the auxiliary carry.

So, we can do the next check like JNB; PSW dot 6 comma L 2 and if the auxiliary carry bit is set; in that case we need to set bit location 10. So, we write like set bit 10 H and then in L 2; we do the last check where it is JNB; that is the parity bit. So, that is by JNB PSW dot 0 comma L 3 and if it is not set; in that case I have to say if that PSW 0 bit is 1; in that case I have to set the bit 22.

So, this is that SETB 22 H and this is our L 3; so from this point it will continue. So, this way we can write a program fragment for checking and setting the individual bits like; so, getting it is doing the copy of those bits to the corresponding locations. So, this

checking the bit and if the bit is 1, then it is setting the bit; if the carry bit is 1, then the corresponding memory location that is mentioned in the problem that is set to 1; so, this way we can do this thing.

(Refer Slide Time: 07:50)



Example

Two strings are stored one after another starting from external memory location 6000H. Each string is terminated by a NULL character (an all-zero byte). Write separate subprograms to perform

- (a) String comparison
- (b) String length calculation

The subprograms should use different register banks. Main program should collect the results and comparison result in R0, length of first string in R1 and length of second string in R2 of the default register bank.

Handwritten notes on the slide:
S1 Same/not same
S2 S1 = 'abc'

Diagram of memory locations:
6000H: 00
6001H: 00
6002H: 00
6003H: 00
6004H: 00

Next, we will look into another program; so, this program is slightly complex. So, here what we try to do is that it says; it is a string manipulation program and there are 2 strings stored one after another, starting from external memory location 6000 H and each string is terminated by a null character that is a zero byte. Then we have to write two separate sub programs for doing this thing; one sub program we will do string comparison.

So, suppose the names of the strings are say S 1 and S 2; so if the S 1 string and S 2 string; so, it will be compared. So, it will just compare whether the strings are same or not; same, not same; it is just doing this type of comparison; it is not that it is finding greater than, less than relationship; it just takes for the equality the strings are exactly same and the strings are different.

So, if their strings are different in that case it will put some value into some register mentioning that the program should do this thing. The main program should collect the results and the company that this character is a comparison result in R 0; length of string first thing in R 1 and length of second string in R 2 of the default register bank. So, what it says is that this comparison result should be in R 0. So, if the strings are same then

maybe the answer will be yes and if the strings are not same answer will be no. So, it can be identified by putting a 1 or 0 in the R 0 register; then for the length of the strings; can be stored in R 1 and R 2.

So, the problem also says that if should the sub programs should use different register banks. So, compared to the original register bank whatever the moving program we will use, this program the sub programs they should use the different register bank. Now for doing that we have to change that PSW bank selection bits; register bank selection bits. And accordingly; it will be using different banks; so, the strings are null terminated that is if I have got say S 1 equal to say the string abc, then from location 6000 onwards; the string is stored; so, if this is my memory; so from 6000 onwards; so this is 6000 say.

So, here I have got the character a; then the character b, then the character c. So, these are the; the corresponding ascii codes are stored there and then we have got the null character 0. So, this is 0 and then after that S 2 starts; so, it is not to the string lengths are not mentioned, so you cannot say the other point that directly.

So, in this particular case the next string will start at 6004, but it is that the length is not specified; so, you cannot set it directly. So, from the program we have to locate for this industry and accordingly we have to set the beginning of the next string and once we are done; we have a set one pointer; or somehow we are attached one pointer here and one pointer here; then we can start doing the comparison.

So, the next string may be say c d; so, it is c, this is d and this character should be 0; that is the end of the string character is there. So, this way the strings are stored; so, we need to write down the program which we will do this string comparison and length calculation. So, see how to do this thing; so first what we do? First we change that register bank.

(Refer Slide Time: 11:53)

```
COMPARE:
MOV DPTR, R3
MOV DPL, R4
MOV R5, R3 } Copy R3, R4
MOV R6, R4 }
MOVX A, @DPTR
CJNE A, #0, L1
SJMP L2 -
L1: INC DPTR
MOV R3, DPH
MOV R4, DPL
MOV DPH, R5
MOV DPL, R6
MOV A, @DPTR
MOV B, A
INC DPTR
MOV R5, DPH
MOV R6, DPL

SETB PSW.3 -> Reg Bank 1
MOV R3, #60H
MOV R4, #00H
ACALL COMPARE
CLR PSW.3
MOV R0, 08
SETB PSW.3
MOV R3, #60H
MOV R4, #00H
ACALL LENGTH
CLR PSW.3
MOV R1, 09
MOV R2, 10

LENGTH:
MOV DPTR, R3
MOV DPL, R4
MOVX A, @DPTR
INC DPTR
MOV R3, DPH
MOV R4, DPL
CJNE A, B, L4
ADD A, B
CJNE A, #0, L2
L4: MOV R0, #1
SJMP L5
L5: RET
```

Suppose we do like say set bit PSW dot 3. So, it changes one of it sets; one of those registers bank bits and it is assumed that by this the register bank is modified. So, of course, there is an assumption that PSW dot 3; this bit was 0 previously. So, the main program was using the register bank 00 and so that assumption is there, but otherwise; so, you can manipulate this part here so, that the appropriate register bank is selected.

So, we have to forget there is this 6000 address. So, we save in the register R 3; the number 60 H and in register R 4, we save the number 00 H that will constitute. So, these values will be needed to be moved to this DPTR register for doing the comparison and all, getting the values from external memory. So, DPTR will be used; so we do that later.

So, then after that; we give a call to ACALL compare; so, this ACALL compare statement; so, it will give a call to the sub program compare. So will see that compare program; after that suppose this compare program, so it has put in R 0 some value that is the result of this comparison; so, then we clear PSW dot 3, so, that we go back to the previous register bank and then move; R 0 comma 0 8. So, move R 0 comma 0 8 because the result should be in R 0 and by setting this PSW 3 bit 1.

So, it is expected that it is going to register bank 1; the basic program is working with register bank 0 and if it is the sub programs, they will be working at register bank 1. Register bank 1; starts with address 8, so this compare routine, so it has set that return value in R 0, but that is in the; actually that register bank 1s; R 0 and; so, that as a result

that is the memory location 0 8; so after creating this bit. So, if you make this move R 0 comma 0 8; so, from memory location 0 8, the content will come to the register 0 of the current bank.

So, that is actually; if the current bank is 0 then it is the actually the memory location 0; so, this is done. So, this is end of this; so, after compare the result is stored in R 0 that part will be that what we will be finished by this, after this I have to compute the length of the string. So, again I do the same thing; first of all I change that register bank to 1. So, SETB; PSW dot 3 setting there; now we again set these R 3 and R 4 registers to contain the lengths of the start address of the 2 strings; R 3 and R 4; R 4 00H and R 3 60H. And then we call a subroutine ACALL length; this length routine, so it will return the length of the 2 strings in the registers R 1 and R 2 of the bank 1.

So, from there I need to get them back onto R 1 and R 2 of the current register bank. So, for that purpose; what I do? Clear this PSW dot 3 bit. So, you clear this PSW dot 3; so, that we come back to the previous register bank 0 and then write like say move R 1 comma 09; so that from this register R 1 of bank 1; the value will come to the register 1 of; the current bank or that the original bank; bank 0. So, that is the length of first string and then a move R 2 comma 10; so, this will take the length of the second string available at register 2 of bank 1 to register 2 of bank 0; so, that is the end of the program.

Now I have to write down the routine for this comparison and this length; so, these 2 routines have to be written. So, first we try to write the routine for compare; so, the compare routine will be something like this. First we have to get; so, R 3 and R 4 they are actually pointing to the memory location 6000. So, if this is your memory location; memory and this is the address 6000 then that R 3, R 4 pair is pointing to this one. So, these R 3, R 4 values they should be moved to DPL and DPH registers so that I can use that DPTR for accessing individual memory bit; individual memory locations.

So, we move DPH R 3; so that six 0 comes to DPH and move DPL R 4. So, we have got this 6000 in the DPTR register and we will also set this R 3 and R 4 in 2 registers; R 5 and R 6 for; otherwise they may be required. So, move R 5 comma R 3 and move R 6 comma R 4. So, these two statements; this is just to keep a copy of these R 3; R 4 pair because this pair will be modified as we proceed.

Then we have to get; now what the next job that I have to do is the; say I have got the string like this a b c etcetera the last character of the first string is 0 and after that the second string will start from this point onward. So, first job is to find the location of the second string; from which point it will start, I have to find that.

So, for that I need to search out this 0; where is this 0 that has to be searched out; so, that will be the; so, we will be doing that. So, what we do? We do it like this, we get use MOVX; A comma at the rate DPTR. So, as a result the first location content will be coming to the A register and then we do a CJNE compare jump not equal A comma hash 0 comma some level L 1. So, if the value is non zero; the value that I have got here is non zero; then that is not the end of the first string.

So, we are looking for this end of string; so, the character that we have now is not the end of the first string. So, we jump to level L 1; compare jump not equal to L 1 then we put here one, but if they are same; if the 2 characters, if we are accessing say this particular location then what will happen? This A register content will become equal to 0 and by that time our pointer has advanced and we have come to the location which is the end of the first string.

So, that way; so, if they are same then we jump to the level L 2; that means, which it actually we will see that it corresponds to the position where we have come to the end of first string. Then in L 1; so, in L 1 what I have to do is? So, the character is not the end of string character. So, I need to increment the DPTR value; so, INC; DPTR, the DPTR value is increment. Then this DPTR is saved again in R 3 and R 4; so, move R 3 comma DPH and move R 4 comma DPL.

So, DPH and DPL values are saved in R 3 and R 4; then what we do? Then we have to go for next one; so, this R 3 and R 4 we have set the value. Now this R 5 and R 6, we had the previous value of this DPTR; so, this L 2 will come here. So, L 2 is somewhere here; so this is our L 2, in L 2 that means we have come to the second string.

So, then I need to start doing the comparison; so this way, so see from this point SJMP L 2; so when this is matching, so we are coming to this point. So, from this point onwards; so if they are same; this is the end of the first string, so I can start doing the comparison. So, how do I do it? Like I first get this; so, it is the DPTR has come to the end. So, DPTR has to be taken back to the first position; so, we first to move DPH comma R 5 and move

DPL comma R 6. So, that way in R 5, R 6 pair we had stored the previous DPH; DPL value; so from R 5 and R 6 those values are moved.

Then we do like MOV A comma at the rate DPTR; so from DPTR, so we are getting the next value. So, we have we have come back to this point; so we have got that character again here. Then you set this character; that we have read from the first string into the B register. So, MOV B comma A; this will save this value that we have read into the B; register then we increment DPTR and save DPTR. So, whenever we increment; so, we set the DPTR in R 5; R 6 pair, so that we can load it later. So, we move this is DPH and move R 6 comma DPL.

So, that is set there now; so, after this is done; so, we have set this DPL, DPH; now I have to get this the first character of the next string and those DPH, DPL values are stored in this R 3 and R 4 registers in these 2 statements. So, those DPH, DPL values will be accessed; so, I am writing from this point here. So, move DPH comma R 3, move DPL comma R 4; then move X A comma at the rate DPTR; then INC; so, we will increment. So, that will go to the next character; so, whenever you are access the; so, you have read the first character from the second string. So, the pointer should advance; so this is pointer in advanced and the values are saved in R 3, R 4 again.

So, move this R 3 comma DPH and move R 4 comma DPL; they are saved there and then what we do? We do CJNE, compare jump or not equal. So, now, this A and B; so, those 2 register values will be compared and it will be going to L 4. Now if they are not same then I am going to L 4, if they are same then another condition I need to check maybe this character is A; so, that is also a match. So for the entire string matched the last character that will have; so, that will also have a 0 there. So, this 0 should match with this 0; so, these two 0's should match; so, for doing that what I do?

So if it is; they are not equal I am jumping to L 4, otherwise I am doing an ADD of A B. So, A and B are added and then we check whether the result is 0 or not. So, CJNE; A hash 0 comma L 5 and then move; so, if they are equal then I will come to this point move R 0 comma hash 1 and we move R 0 comma hash 1 and return from there. And this L 4 is actually this point move R 0 comma hash 0. So, this is the point L 4 and this is CJNE; their characters are same, but they are not 0. So, this is this is L 5; so, this should better be L 2 that will do it correct.

Because if they are not same; I am jumping from here to here, so this level should be L 2. So, this one should better make it L 2 then the program will be correct; because now I have got the next character. So, again I am going to compare the next character; so, what has happened is the first 2 characters have matched. So, if it is coming from here and the first 2 characters that matched; so now, I am getting the next character.

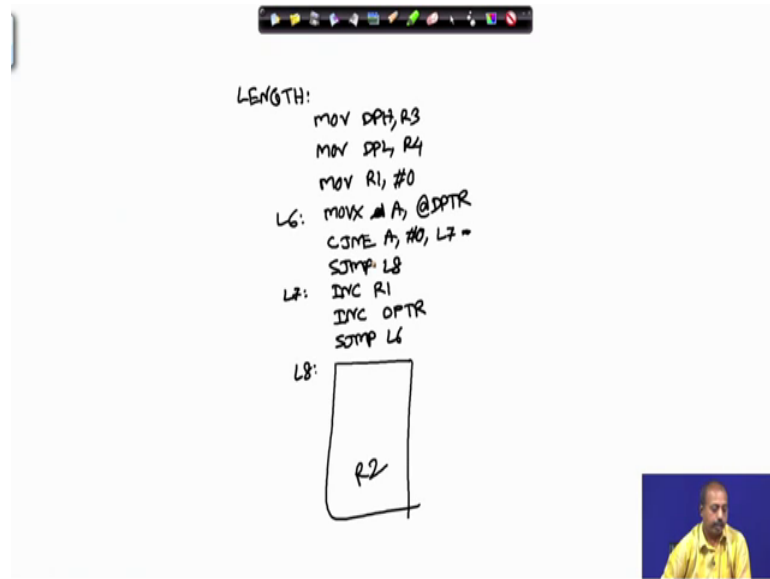
So, these MOV X will get me the next character of first string, these MOV X will get me the next character of the second string. And then I am doing a comparison of them in this CJNE instruction. So in the CJNE instructions; so, I do a comparison and if they are not if the characters are same. It will come here and it will come to this; if the characters are not same then it will come to L 4.

So, this R 0 becoming 1 means the number is; the other strings are not same, if R 0 becomes 1 then the strings are not same. If the strings are same, if the characters are same then I am taking whether I have reached the end of the string or not. So that way by doing this program; so, I can do a substring check part. Now if you want to do; so that is there, so that is setting 0 and this L 4 is so, CJNE hash 0 say if it is 0; then it is coming to; so it is otherwise it is not same.

So, in that case it will be written; if the characters are not same then it will be written in 0. So, how to do that? CJNE 0, L 2; so this can be. So, after this I should have something like this; MOV R 0 comma hash 0 if the strings are not same. So, if the characters are same then it is a adding here and then I am taking end of; they are not same. If they are not same, this with I am sorry yeah; so, there should be one SJMP here. So, I am L 5 and in L 5; I should have that RET; the return path. So, that will do this thing correctly.

Now, regarding the length part; so, the length calculation part; so, you can do it like this.

(Refer Slide Time: 31:44)



The length part is simple; so, we can do like move DPH comma R 3 comma 3; we get DPH; MOV DPL comma R 4; from R 4 we get DPL. Then MOV R 1 comma hash 0 and then we say MOV X at the A comma at the rate DPTR; then compare jump one not equal this A hash 0 say L 7. So, it is comparing whether we have reach the end of the string or not; SJMP L 8 in L 7 we implement R 1; then Inc DPTR will increment the DPTR register.

And then, we check whether then SJMP L 6; well L 6 is this point. So, this way I can check the; I can get the length of the first string. So, till it is coming to this 0; so, compare jump 1 or not equal A hash 0 L 7. So, it will be incrementing and this is the point L 8; so, this way I can get the length of the first string calculated in R 1 register. So, you can repeat this procedure; the similar ports starting from L 8 for the length of the second string in register R 2. So, which you can just replace this R 1 by R 2; you can get the next strings length is calculated.