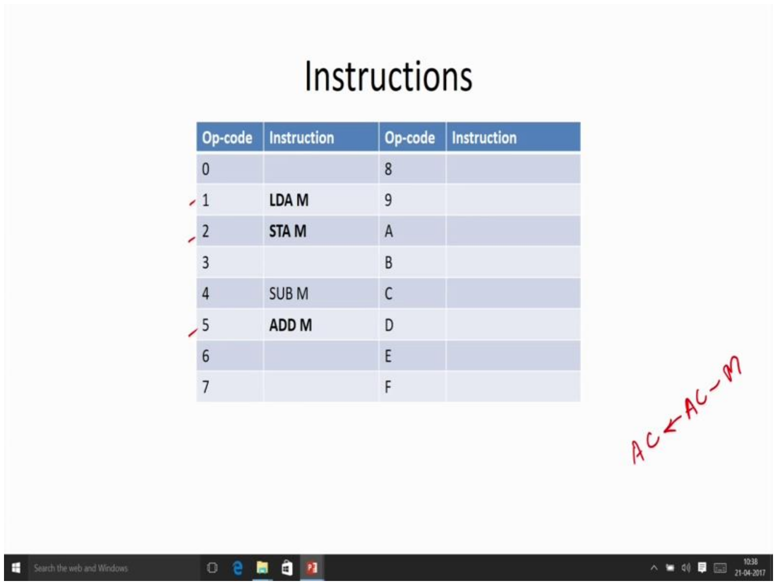
these are the 3 Instruction we have designed and the code assigned is your 1, 2 and 5. Other codes are now still available to me.

So, now we can design some more Instructions. So now, we are saying that already we have designed

(Refer Slide Time: 36:41).



Instructions

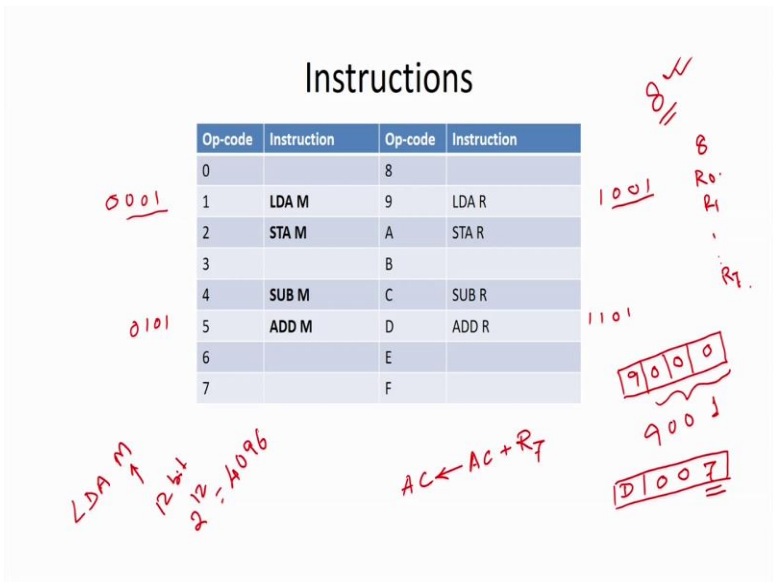
Op-code	Instruction	Op-code	Instruction
0		8	
1	LDA M	9	
2	STA M	A	
3		B	
4	SUB M	C	
5	ADD M	D	
6		E	
7		F	

$AC \leftarrow AC - M$

these 3 Instruction. Now, we are saying that we are designing one more Instruction call SUB M. So, it means subtraction. So, what is this Instruction this is basically nothing but Accumulator is equal to Accumulator minus contents of the Memory. Ok we are going to give the Instruction format is same whatever we are going to design for all the Instruction it is going to follow this particular pattern.

Ok So, now along with this 3, I am going to use one more code this is code 4 which your subtraction code. Like that we can now add more and more Instruction now along with that now again I am going to give designing 4 more Instruction. This is a similar Instruction load store sub m, but my reference is different initially we are talking about the memories now we are talking about the Registers.

(Refer Slide Time: 37:34)



Instructions

Op-code	Instruction	Op-code	Instruction
0		8	
1	LDA M	9	LDA R
2	STA M	A	STA R
3		B	
4	SUB M	C	SUB R
5	ADD M	D	ADD R
6		E	
7		F	

8
0001
1001
8
R0
R1
R7
0101
1101
9000
9001
D007
LDA M
$2^{12} = 4096$
$AC \leftarrow AC + R_7$

So, now what will happen whatever operation we are having say here I am going to say that this is again say load operation this is your 1 is your 0001, what is your 9, 1001; that means, for load I am returning this particular 0001, but with the most significant bit 0 is going to say that it is a Memory reference and 1 will indicate that it is a Register reference. So, when it is 9 then I can give the reference of the Memory. So, in that particular case generally already I have mentioned that number of Registers is limited very less number Registers. So, if I am having say only 8 Registers that we are going to use say $R0$, $R1$ to R7 say these are the Registers.

Then what will happen in that particular case we don't require the 12 bits. So, these are basically don't care, but I can keep all those things is 0 and along with that here I am going to give the Register number. So, if I say that this is your 9000 means it is going to refer to this particular Register $R0$. The value of the Register $R0$ will be loaded to the Accumulator.

So, similarly if my opcode is your 9001 it is going to say that take the value of the Register $R1$ and bring it to the Accumulator. So, if you see the Organization, we are saying that we are having some general purpose Register now we are having $R0$, $R1$ like that we are saying 8 different Register. So, contents of this Register can be taken out from this Register and put it to the Accumulator with the help of this Instruction 9000. So, it is referring to the Register 0 and what is the opcode says take his information and bring it to this Accumulator.

So, like that we can have the effect of those particular Instruction. Now, if I am going to write say this is your 5 is your 0101 and what is your this things when I am going to refer it this is your 1101. Ok So, now if my Instruction is like that this is your opcode is your D and say it is your 007. Now, what it basically does it is an addition of R.

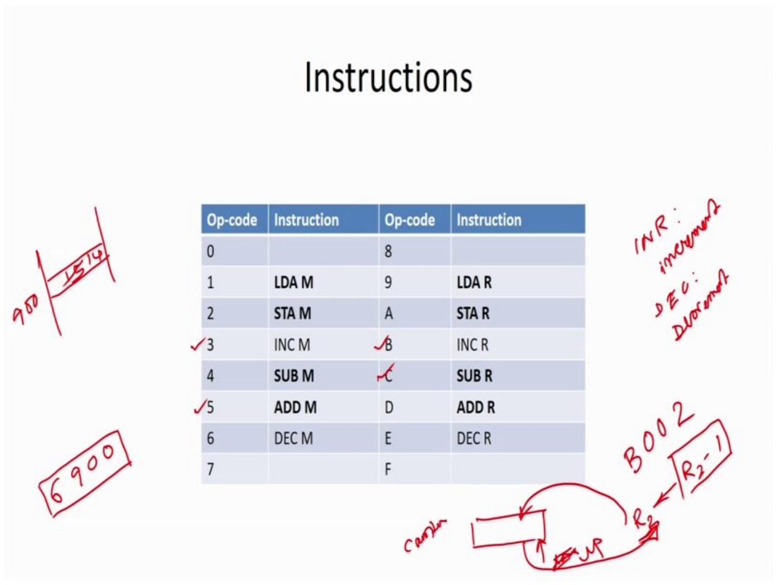
So, this Instruction is basically nothing but Accumulator is equal to Accumulator + R7, we are referring this particular Register 7 or R7 . So, the effect of this Instruction is like that. So, similarly we can now interpret other also. So, we are having similar Instruction one is referring to the Memory location and second one is referring to the Register.

Ok So, in this particular case you just see that how many different kind of combination we are going to have? We are going to have only 8 different combination. Because, that Registers values can go from 0 to 7 totally. So, for all those Instruction we are going to get 8 different variation maybe load $R0$, load $R1$ like that.

But what will happen in this particular case when I am going to talk about LDA M. So, M is going to take all the 12 bit address. So, this is your 2^twelve; that means, 4096 Memory space so; that means, we are going to have 4096 variation of this particular LDA M. Because, we can get or we can use any Memory location to take our information, but for LDA R we are going to get 8 variation only because we are having 8 general purpose Register.

So, similarly LDA R, STA R, SUB R, ADD R now you just see that 4 + 4 we have designed 8 Instruction still we can design 8 more Instruction. Now, in that particular case

(Refer Slide Time: 41:41)



Instructions

Op-code	Instruction	Op-code	Instruction
0		8	
1	LDA M	9	LDA R
2	STA M	A	STA R
✓3	INC M	✓B	INC R
4	SUB M	✓C	SUB R
✓5	ADD M	D	ADD R
6	DEC M	E	DEC R
7		F	

900
1514
INR: increment
DEC: Decrement
6900
B002
R_2-1
R_2
counter
ALU

We are designing some more Instruction over here. So, this is INR is basically in increment and DEC is your decrement. So, what will happen we are having one increment operation and one decrement operation we can increment the value of my Register or Memory or we can decrement the value of my Memory or Register.

So, in that particular case say if I am going to say 6900. So, in that particular case what will happen 6 is my decrement. So, whatever value we have in the Memory location 900 say if this is my Memory and in my Memory location 900 say we are having say 15 then what will happen it will decrement of value of this particular Memory location and after execution we are going to get 14.

Similarly, if I am going to have say B002 now what will happen it is basically nothing but we are having the Register $R2$,the value of Register $R2$ is nothing but $R2$ - 1. So, this is the way we can look into it now when we are using increment and decrement operation in processor general we are not going to use the ALU for such type of Organization. Because, here we are having some values in the Accumulator.

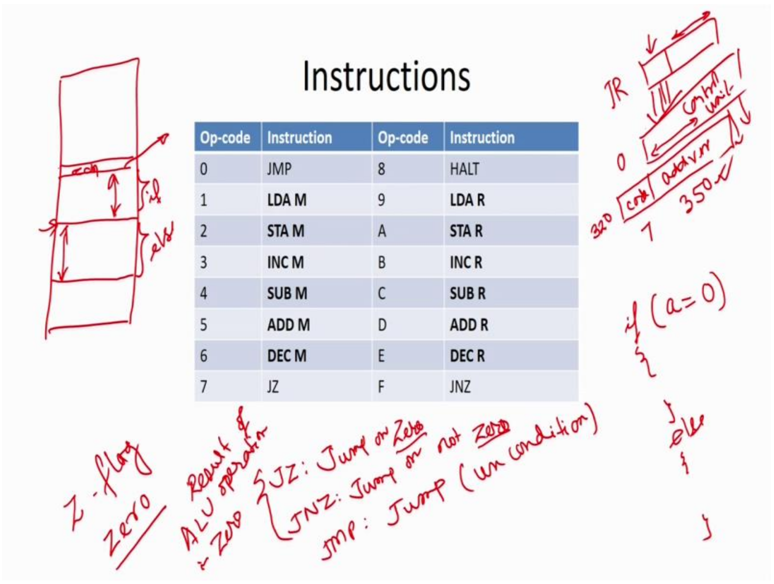
If I am going to use Accumulator for decrementing then we have to bring it from the Register to the Accumulator. So, it will get disturbed that means, temporary we have to store the value of Accumulator. So, instead of doing it what generally we used to do we are going to put special circuit over here which is going to increment it or decrement it. One simple way we can think about that how to implement it maybe we can use a counter which is an up down counter because already we have discussed up down counter. So, we put the value once we going to execute the increment Instruction what we are going to do we are going to count up and when we are going to implement a decrement operation we are going to count down.

So that means, value of $R1$ will be placed to counter. So, basically what we are going to do, when we are going to have this thing say decrement your $R2$ then what will happen we may use a counter over here. So, value of $R2$ will be loaded over here then we are going to count down operation then oh sorry B is your increment. So, we are going to count up. Then value of counter will be incremented by one and going to store it back to Register $R0$.

So, like that now we say out of 16 we have now consumed 12 operations. Still 4 are remaining left till now we have designed this particular 12 Instruction still 4 more code are remaining left

0, 8, 7 and F. So, that means we can design 4 more Instruction now just see what we are designing.

(Refer Slide Time: 44:52)



Instructions

Op-code	Instruction	Op-code	Instruction
0	JMP	8	HALT
1	LDA M	9	LDA R
2	STA M	A	STA R
3	INC M	B	INC R
4	SUB M	C	SUB R
5	ADD M	D	ADD R
6	DEC M	E	DEC R
7	JZ	F	JNZ

if
else
JR
control unit
0
code
addr
320
350
7
if (a=0)
{
}
else
{
}
Z-flag
zero
Result of ALU operation is zero
JZ: Jump on Zero
JNZ: Jump on not Zero
JMP: Jump (un condition)

Now, here I am saying that opcode 0 is your JMP. It is a jump Instruction. It is a halt Instruction; that means, it is going to say that halting the program or stopping the program execution that means at the end of the program we have to give this HLT Instructions just to say that now stop execution need not fetch any more information.

Similarly 7 is given as your JZ and F is your JNZ. This is JZ is your jump on 0 and JNZ is your jump on not 0 and this is JMP is your jump. So, these are the control Instruction we are defining this jump is your unconditional jump without any condition we are going from one Memory location to another Memory location, but jump Z and jump not Z and Z jump 0 and not 0 these are conditional jump Instruction. It depends on some condition and we are going to jump to some other Memory location. So, basically what will happen we are talking about the 0, now how we are going to take a decision you know that we are having a Z flag which is basically zero flag. Now when we are going to set this particular 0 flag? When the ALU operation is 0.

Ok now, you just see that in your high level language sometimes you write if a = 0 do something otherwise do something else. So, if we are having such type of program. So, in Memory what will happen we are storing our problem like that we have said this is the conditional Instruction and this is one part and this is the other part.

So, I can say this is if part this is else part. In that particular case what will happen first we are going to check this particular condition depending on the condition if it is true we are going to execute this particular set of Instruction and if it is false then we are going to execute the other set of Instruction. So it is making a say choice.

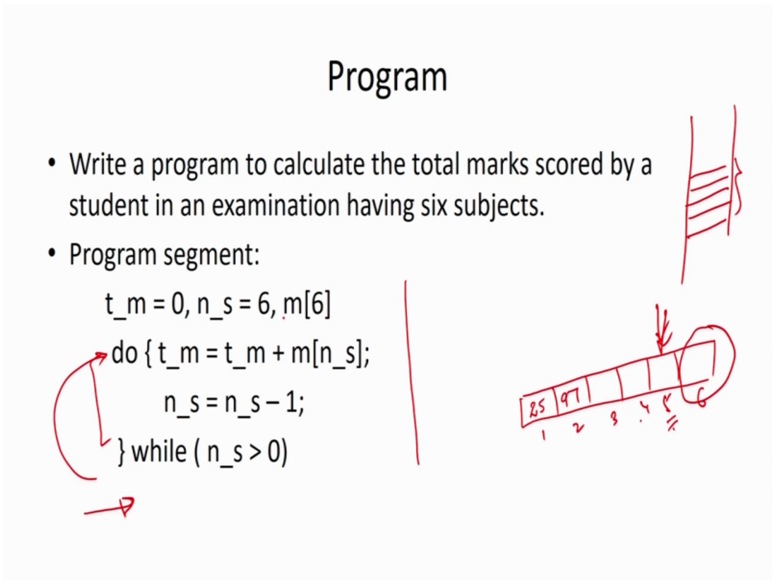
Now, how we are going to making a choice depending on the condition. So, such type of condition can be implemented with the help of this conditional jump Instruction. So, one of the things is we are talking about jump 0 so; that means, we are having this particular zero flag. Zero flag will be set to one if the result of ALU operation is 0 result of ok. So, we perform an ALU operation if the result of ALU operation is coming as 0 then it is going to take a decision and it may take come to this particular Memory location ok. We have to specify the Memory location along with Instruction. We know the format. Basically this is your opcode and this is address.

So what will happen, here I am going to say that opcode is 7 and say address is 350. So, in that particular case when I am going to execute this thing if this condition is true we are going to jump to the Memory location 350 and say here it is a now we are in say 320. So, next Instruction should be we need to fetch from 321. But when this condition satisfied then we are not fetching it from 321, but we are going to fetching it from 350.

Similarly, that opcode 0 it is unconditional jump. So, whatever address we provide it will simply go to that particular Memory location then how we are going to get it; that means, we are going to load the program counter with this particular of value then only we will be knowing that next Instruction you have to fetch from that particular Memory location and it is HLT it is going to say that stop the execution of the program. So, we need not to fetch any more information from the memory ok. Now you just see that we are having 16 different operation with and we are assigning this particular 16 different operation to those particular code 16 code.

Now, when we fetch it then what will happen it will come to the IR and in IR this is the basically code and this is the other 12 bits is different. These code what this 4 bit we are going to give it to the control unit and according to the nature of this particular Instruction, now control unit will generate the appropriate control signal and this will go to the appropriate component inside the processor. So, this is the way we can see how we are going to execute this particular program.

(Refer Slide Time: 50:00)



Program

- Write a program to calculate the total marks scored by a student in an examination having six subjects.
- Program segment:

```
t_m = 0, n_s = 6, m[6]
do { t_m = t_m + m[n_s];
        n_s = n_s – 1;
} while ( n_s > 0)
```

25
97
1
2
3
4
5
6

Now, you just see that taking a small example just is I am saying that I have to write a program to calculate the total marks scored by a student in an examination having 6 subjects. So, say in your semester you are having 6 subject you scored say marks in the 6 subject and finally, you can get the add them together and find out what is the total marks that you have scored.

So, in some high level language you can write such type of program code what is that now we are having we need some placeholder. So we are defining one t_m, this is going to say what is the total mark initially this value is 0, n_s number of subject now we are talking about the 6 subject and we are having marks of 6 different subject. So, we are saying $m[6]$. So, in high level you know that we can define array.

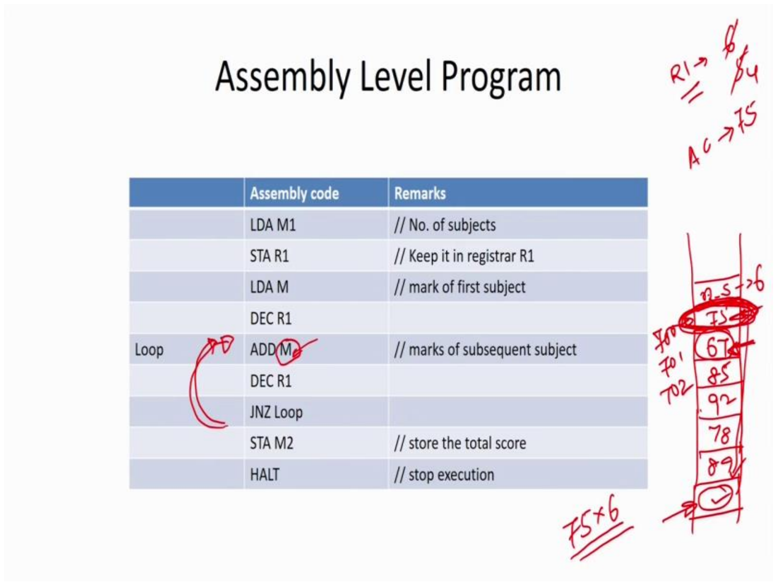
So, we are defining arrays. So, we are having six elements ok 1, 2, 3, 4, 5, 6. So, here I am having some marks like that 25, 97 something like that. So, this marks whenever defining arrays so, this marks will be stored in our Memory in 6 consecutive Memory location.

Now, what we are doing now do what I am going to do t_m = t_m + m ns. So, if this is 1, 2, 3, 4, 5, 6 location of this thing first I am going to add the contents these things to total marks. So, total marks will become 0 plus something then we are decrementing n_s by 1; n_s - 1. So, now, ns becomes 5. So, I am going to perform this particular operation while ns is greater than 0 after that I am going to check now it is 5 not it is still greater than 0.

So, we will go back and we add the next number, we will decrement it like that when it becomes we come out from the loop and it will go to the next Instruction. So, I am getting after coming out from this loop what we are getting we are adding all those numbers and the result is available in the t_m, $t_m = t_m +$ this array element.

And this array will be stored in consecutive Memory location in my main Memory. Now this is a high level language now when we are going to write a program. So, in low level

(Refer Slide Time: 52:17)



Assembly Level Program

	Assembly code	Remarks
	LDA M1	// No. of subjects
	STA R1	// Keep it in registrar R1
	LDA M	// mark of first subject
	DEC R1	
Loop	ADD M	// marks of subsequent subject
	DEC R1	
	JNZ Loop	
	STA M2	// store the total score
	HALT	// stop execution

R1→
AC→75
700
701
702
75
67
85
92
78
89
75+6

Or assembly level, I can say like that. So, this is basically related to this particular example. Here we are saying what first I am writing load Accumulator M1.

So, basically what I am thinking say this is the Memory and we are having some Memory location in this Memory location we are storing the number of subject then I am having say 6 marks in some consecutive Memory location ok, this is say 75, 67, 85, 92, 78, 89. So, this is the number of subjects say here basically we are storing 6.

So, initially first I am bringing it this particular number 6 then store Accumulator $R1$. We are storing this particular thing in Register $R1$. So, now in $R1$ what we are having 6 this is the number of subject then load Accumulator Memory. So, I am now going to give the address of this particular Memory location in this particular Instruction and we are getting this is now what we are having this information is in Accumulator now. So, in Accumulator we are having 75.

Already we have taken care of the first number. So, we are decrementing $R1$ now value of $R1$ become now 5 because already have taken care one number then what you are doing ADD M what ADD M will do it will take the contents of this particular Memory location 67 adding with the Accumulator and store the result in a Accumulator. So, we have taken care of the second number. So, we are decrementing $R1$.

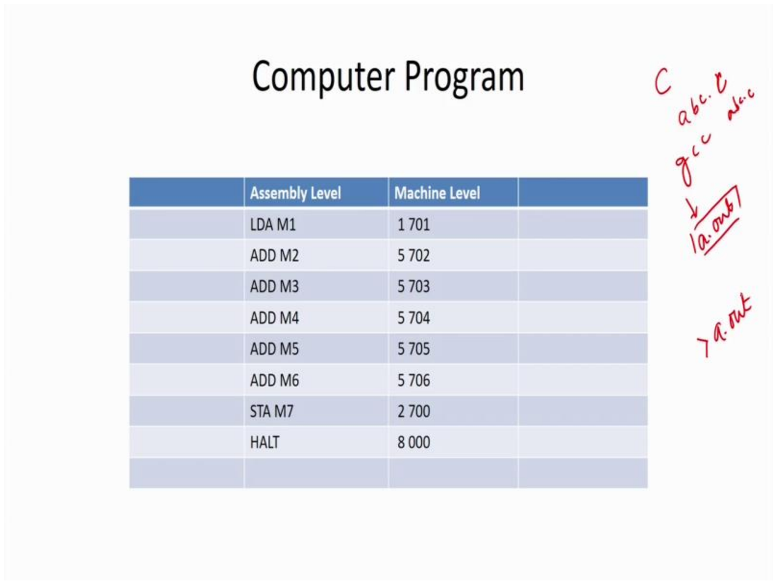
So, now $R1$ will become 4 we will say sum not equal to 0 then go back to this particular Memory location and fetch this operation. So, it will be in this particular loop till we are going to exhaust this particular 6 number once it is over then say store Accumulator and I can say that the result we are going to store in some Memory location and after getting the result in the Memory location now we halt it. So, this is the way we are going to do it.

Now, this program cannot be written for the processor that we are discussing and for the Instruction set we are having because here you just see that here I am talking about that add m we are talking one Memory location when you go back in the next time we have to go to the next Memory location, but here in this Instruction set we do not have any provision to manipulate this particular address Memory. So, that’s why this program is cannot be executed in this particular processor if we are going to write this particular program what will happen is going to add this particular 75, 6 times because we do not have any provision to sense manipulate this particular address M we do not have any Instruction.

So, address manipulation is not available in this particular Instruction set. So, if we are going to execute this particular program in this particular processor we are going to add up 75 in 6 times. So, result will be your 75 into 6 because we do not have an Instruction to manipulate this particular address M because if I am storing it in say 700 second one I have to take it from 701 then 702; that means, we need to increase this particular M after adding that particular number.

But in my Instruction set I do not have any such provision whatever we are doing incrementing and decrementing it is going to increment and decrement the contents of the Memory location it is not going to manipulate the address. So, we need some more Instruction to manipulate the address also. So, we will see in our subsequent lecture, but in this processor we cannot design it because we are having 16 code and we have exhaust all the 16 code. So, we need more bits to design more number of Instruction.

(Refer Slide Time: 56:20)



Computer Program

	Assembly Level	Machine Level	
	LDA M1	1 701	
	ADD M2	5 702	
	ADD M3	5 703	
	ADD M4	5 704	
	ADD M5	5 705	
	ADD M6	5 706	
	STA M7	2 700	
	HALT	8 000	

C
abc.c
gcc abc.c
↓
a.out
> a.out

So, one simple program I can write for this particular machine. So, load Accumulator M1. So, first number I am getting it in Memory location the way I am saying that 700 so; that means, load it from this particular Memory location an M2 take the number from the second location add it and store it in Memory location like that ADD M3, ADD M4, ADD M5, ADD M6 then store the result in M7 and halt. So, we are going to write individual Instruction for each and every number.

So, for 6 of this it is fine, but if I say that going to add thousand numbers then we cannot write thousand Instructions somehow we have to use such type of loop only. So, for that we need Instruction to manipulating address also now finally, I can say that now what we are saying that this numbers we are storing in Memory location 701, 702 like that.

So, in assembly level we are writing it then in machine level we are going to have this particular effect ok. So, now, you just see that we are talking about the assembly level, machine level and like that we can have high level also. So, here I can write high level program the way I am writing over here. So, this is a high level program. So, the same problems I can say this is the assembly level and this is a machine level.

Now, when I am going to write a high level program now how I am going to execute it. So, somehow I have to convert it to the machine level program. So, for that we are having a software which is known as your compiler. So, compiler is there. So, we are going to compile it and after compilation we are going to get such type of machine level code and this machine level code will be executed in program.

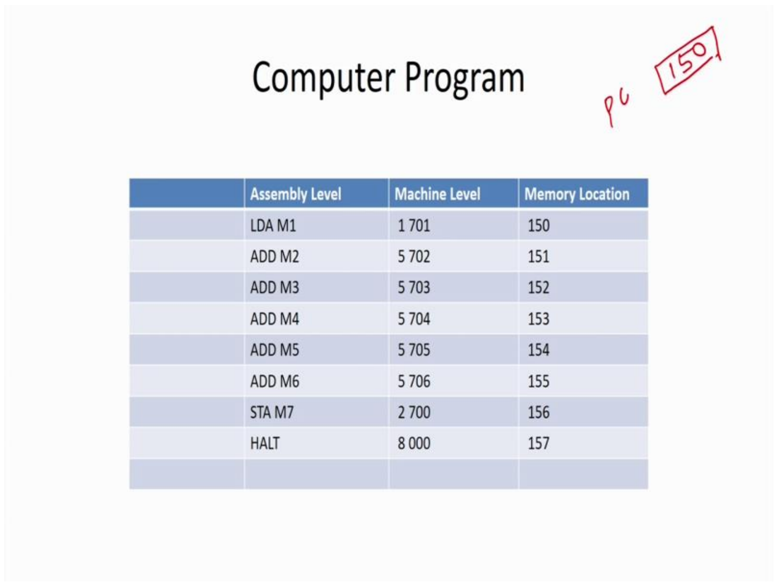
So, if you are accustomed with your C program. So, if you have written any C program and say if you are working with unix system or Linux system generally you use some compiler gcc and like that. Say if you are writing a program called say abc.c then what will happen generally you compile something like that gcc abc.c and it is going to give me a.out. So, this is the executable file and this executable file is not having such type of machine level code only.

Now, we can execute this particular code. So, high level language can be converted to the machine level code similarly we are having an assembler also we can write the code in the assembly level and with the help of assembler we can convert it to the machine level code finally, we need this particular machine level code to execute in the computer.

So, assembler is going to convert assembly code to the machine code and compiler is going to convert the high level language to the machine code of a particular machine. Along with that we are having another term which is known as your interpreter. So, again interpreter is used for high level language. So, in case of compiler we are going to compile the whole program and going to get an executable file and going to execute this particular file, but in case of interpreter it is going to interpret the Instruction by Instruction and it is going to first interpret the Instruction first then going to execute it then it will interpret the Instruction 2 and going to execute it.

So, if I am writing a high level program it is going to interpret the Instruction by Instruction and execute it one by one. Now finally, we are going to get this particular machine level code now we have to executive it now one how we are going to execute it generally we give the command like that a.out what will happen when I am giving the command a.out it will load this particular program to main Memory.

(Refer Slide Time: 59:57)



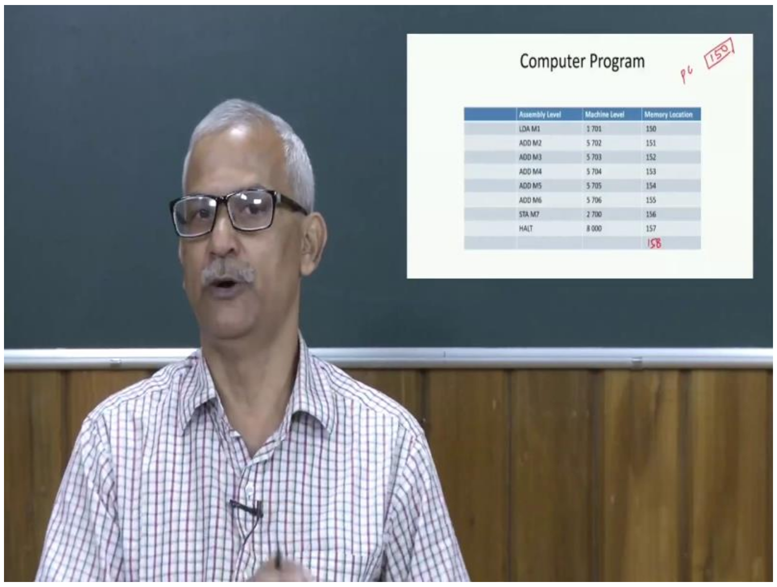
Computer Program

pc $\boxed{150}$

	Assembly Level	Machine Level	Memory Location
	LDA M1	1 701	150
	ADD M2	5 702	151
	ADD M3	5 703	152
	ADD M4	5 704	153
	ADD M5	5 705	154
	ADD M6	5 706	155
	STA M7	2 700	156
	HALT	8 000	157

So, you just see that we are loading this particular program say from main Memory location 150 to 157. So, we are loading it and Memory location; that means, in Memory location hundred fifty I am having 1701, in 151 I am having 5702 like that 157 we are having eight thousand 8000 the other bits are immaterial because it is the halt function. So, we are loading it to the Memory location and when we load it to the Memory location then we are having the program counter that program counter will be load with the value 150, because we must know the address of an Instruction since this program is loaded from Memory location 150 to 157. So, program counter will be loaded with this particular value 150 now we are going to fetch the Instruction from 150 we will execute it, after execution of first Instruction next Instruction we are going to fetch it from the 151 will execute it like that when we fetch the Instruction from 157 we halt the program we stop the program execution; that means, we are not going to fetch anymore Instruction from 158 memory location, it stops. So this is the way we are going to execute our program in our computer.

(Refer Slide Time: 61:04)

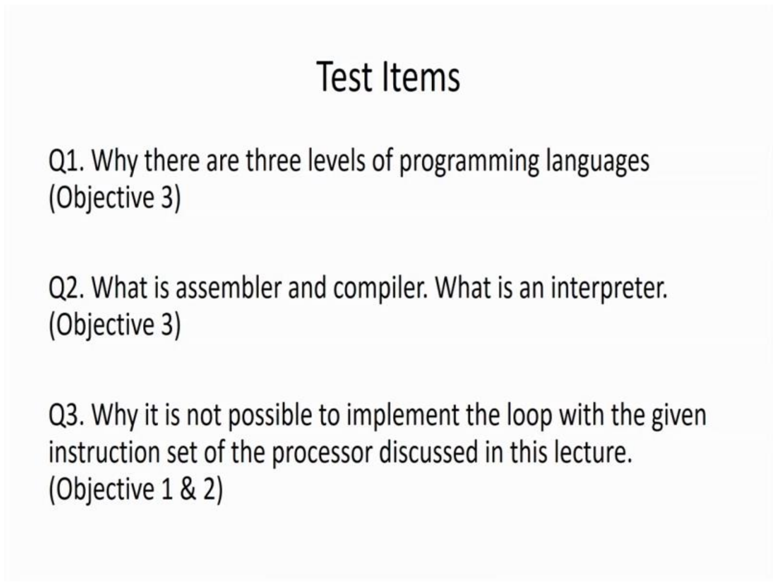


Computer Program

pc $\boxed{150}$

	Assembly Level	Machine Level	Memory Location
	LDA M1	1 701	150
	ADD M2	5 702	151
	ADD M3	5 703	152
	ADD M4	5 704	153
	ADD M5	5 705	154
	ADD M6	5 706	155
	STA M7	2 700	156
	HALT	8 000	157
			158

(Refer Slide Time: 61:13)



Test Items

Q1. Why there are three levels of programming languages (Objective 3)

Q2. What is assembler and compiler. What is an interpreter. (Objective 3)

Q3. Why it is not possible to implement the loop with the given instruction set of the processor discussed in this lecture. (Objective 1 & 2)

Now say I feel that now we are having an idea how a program is executed in a computer whether we are writing it in a high level language or assembly level language or machine level language.

So, now just I am giving some very simple test item here I am saying that first question is why there are 3 levels of programming language? Now I am talking about the machine level, assembly level and compiler. You just see if I am going to give it a processor and just saying that you just program it then what will happen you have to write everything in machine level;

that means, you must remember the code of each and every Instruction which is not possible always we have to take a reference.

But instead of remembering Instruction or numbers slightly it is easy to remember some code like that add, subtract, multiply, load, store. So, for that what will happens we are giving a code to each and every Instruction which is known as your mnemonic codes now once we have the mnemonic codes now we can use this code to write a program.

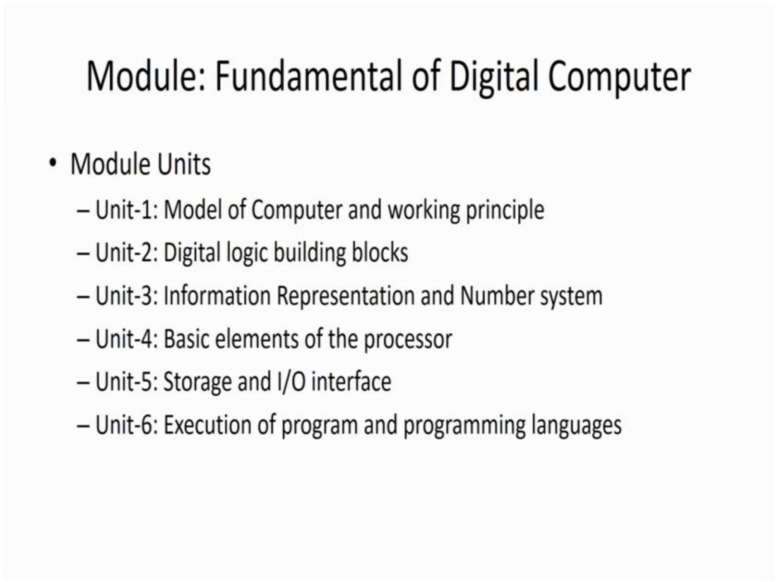
So, if we write our program with the help of mnemonics code then we are going to say this is the assembly level code, after writing in the assembly level code with the help of assembler we are going to convert it to the machine code and that machine code will be executed, but again remembering the assembly code on mnemonics is not that easy because now it is I am talking about the 16 Instruction. But if I am going to design a processor where the size of the Instruction is an 8 bit that opcode then I can design 256 different Instructions. So, remembering the different Instruction mnemonics is difficult. So, for that we are coming up with the high level language you write your program the way you think, but we have to follow the syntax of a particular programming language.

So, when we are going to write C program we must follow the syntax of the C program, once we write it then we can compile it to the machine level. So, this is the requirement of the high level language because it is difficult to code in machine level, it is against slightly difficult to code in assembly level also because we have to remember many more things. So, that’s why you write in high level then convert it to the machine code.

Question 2 what is assembler and compiler and what is an interpreter. So, already I have mentioned it what we use to do with assembler and compiler and what is an interpreter. Question 3 why it is not possible to implement the loop with the given Instruction set of the processor discussed in this lecture.

So, we are just saying that for a small processor Instruction is 4 bit, already I have explain why we cannot implement the loop. So, for implementing the loop we have to have some Instruction to manipulate the address given if we are going to take the information from our Memory. So, this is required so that's why here it is not possible.

(Refer Slide Time: 64:09)



Module: Fundamental of Digital Computer

- Module Units
  - Unit-1: Model of Computer and working principle
  - Unit-2: Digital logic building blocks
  - Unit-3: Information Representation and Number system
  - Unit-4: Basic elements of the processor
  - Unit-5: Storage and I/O interface
  - Unit-6: Execution of program and programming languages

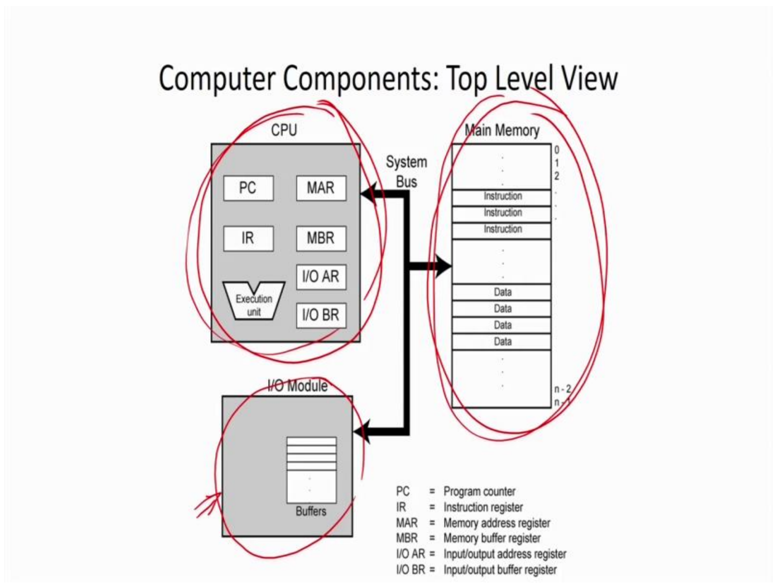
Now, this unit is the last unit of this particular module fundamental of digital computers so that means, we are coming to end of this particular module. So, in this particular module we have divided this module into 6 unit. So, in first unit we are talking about model of computers and working principle. So, we have discussed about the how we are going to model a computer and how the computer works.

Unit 2 is about digital logic building blocks because we are going to construct a digital computer. So, we are going to take help of many more digital building blocks. So, just we are giving a brief ideas about those particular digital building blocks and we are going to use those building blocks. So, we are not discussing anything about the design issues of those particular digital building blocks in knowledge level we have discussed it.

Unit 3 basically we see information representation and number system ok. So, how to represent numbers integer and real we have discussed, how to represent other information like that how we are going to write your name, store your name in a computer all those things we have discussed in unit 3.

Unit 4 is basic element of the processor. So, we have discussed about what are the basic elements that we have in the processor in the top level then unit 5 we have discussed about storage and I/O interfaces because processor works on von Neumann stored program principle somehow we have to give the information to the processor. So, that's why we have to interface or connect storage unit and as well as I/O unit and unit 6 that last unit that today I have covered we have discussed about how we are going to execute a program and what are the different levels of programming languages.

(Refer Slide Time: 65:48)


Computer Components: Top Level View
CPU
PC
MAR
IR
MBR
I/O AR
Execution unit
I/O BR
System Bus
Main Memory
Instruction
Instruction
Instruction
Data
Data
Data
Data
0
1
2
n - 2
n - 1
I/O Module
Buffers
PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = Input/output address register
I/O BR = Input/output buffer register

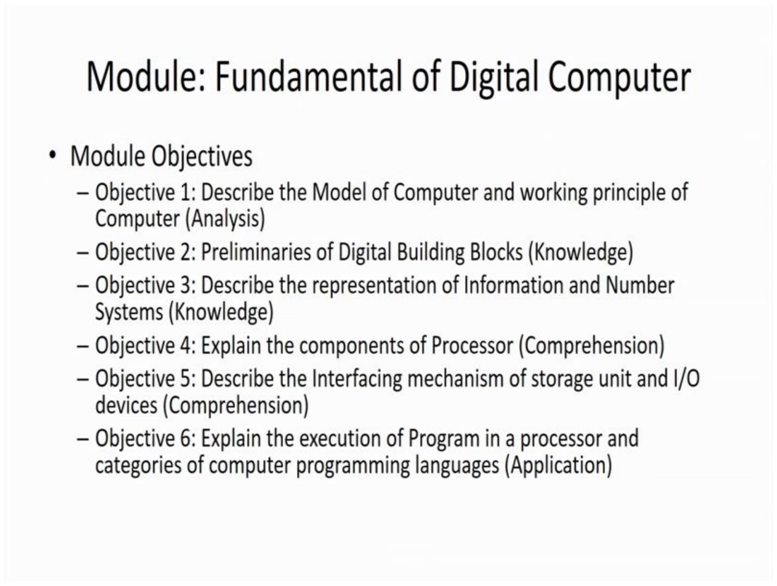
Now so, what idea till now we have, this is a top level view of our computer. So, main element is the processor it works on von Neumann stored program principle. So, we have the main Memory then processor is going to work with the contents available to the main Memory, but how we are going to bring the information to the main Memory for that we need input output device that will be connected through I/O modules and all those components are connected through this particular system bus. So, we are having these idea till now. So, in first module we have just have a very top level view of our computer. Now in subsequent module so we are going to discuss about the design of those particular processor. So, we are having modules to discuss about the design issues of this particular processor or CPU.

So, we are going to discuss in details in subsequent module how we are going to design a processor how we are going to design an Instruction for that particular processor how we are going to interpret those particular Instructions. So, all Organizational and architectural issues will be discussed with respect to the designing of this particular processor. We will have one module where we are going to discuss about the Memory module or Memory unit.

What are the components that we are having because here we have seen the top level only we have the data bus and we have the address bus, with the help of this thing we can fetch information from Memory and store information in Memory. So, what are the basic principle to design this particular Memory how we are going to construct the Memory we are going to design all those in this is another module on Memory model. We will have one module where we are going to discuss all the design issues of the I/O module ok. So, we will get another module and in this particular case whatever we are going to discuss over here we are going to discuss the basic things on the this is unique processor system we are having one processor and we are going to work with this particular processor only we are going to execute in this particular processor, but to enhancement of the performance we are having some advanced topics also advance feature also.

Whether something can be done in parallel or not whether it can be done in stages or not. So, some of the advanced features to enhance the performance of the computer will be discussed in one module here we can we will discuss only in the information purpose only in the knowledge level only.

(Refer Slide Time: 68:12)



Module: Fundamental of Digital Computer

- Module Objectives
  - Objective 1: Describe the Model of Computer and working principle of Computer (Analysis)
  - Objective 2: Preliminaries of Digital Building Blocks (Knowledge)
  - Objective 3: Describe the representation of Information and Number Systems (Knowledge)
  - Objective 4: Explain the components of Processor (Comprehension)
  - Objective 5: Describe the Interfacing mechanism of storage unit and I/O devices (Comprehension)
  - Objective 6: Explain the execution of Program in a processor and categories of computer programming languages (Application)

So when we are discussing about this particular module we have defined some objective in the module level also because in every unit we have define unit level objective and we have seen that after going through that particular unit we have achieved those particular objective.

Now, we have completed the complete module of fundamentals of digital computer let's see whether we could achieve those particular objective that we have already mentioned for this particular module. So, the objective first objective we have mentioned like that describe the model of computer and working principle of computer and we have defined it in the analysis level. I think we are now having some idea what is the model of computer and how it works.

So, I think we have achieved this particular objective. The objective type 2 we have defined it like that preliminaries of digital building blocks. So, it is in the logic level. So, what will happen in that particular case we are just giving the ideas of those particular building blocks and we are going to use those things?

So, objective 3 describe the representation of information and number system. So, in this particular objective I think we have made that particular objective also because now we are having some idea about the number system and how to represents those number in computer real numbers and integer along with that how to represent the other information also. Objective 4 explain the components of processor. So, it is in the comprehension level.

Now, at least now we are having an idea what are the components that we are having in processor and how it works. Objective 5 describe the interfacing mechanism of storage and I/O devices this is also in comprehension level. So, here now we are having that we have to connect I/O devices we have to connect storage and I have to take information from those things.

So, in comprehension level we are having some idea, but in subsequent module we are having to discuss all those issues in the design level like for the processor also in subsequent level we are going to discuss all those things in our design level and objective 6 explain the execution of program in a processor and categorise the computer programming language? This is in the application level or I can say that up to some extent we are going we have gone up to the analysis level also.

So, now we are pleased if we get a program we will be able to analyse that particular program and. Secondly, if we know the Instruction set of a processor I think you will be able to write a assembly code for that particular processor. So, it is in application and analysis level we have achieved and in this particular course we are not going to discuss anything about this particular objective.

In 6 we are not going to discuss anything about the programming languages like that all those things will be discussed in some other courses maybe that compiler is another course where we are going to discuss about the design of compiler. So, in this particular course we are not going to discuss anything about those particular programming languages and execution of the program. Now when we are completing this particular module now just look into some problems in the module level.

Now, what will happen in unit level problem you must know the concept of the particular unit then you will be able to solve the particular problem, but when I come to the module level