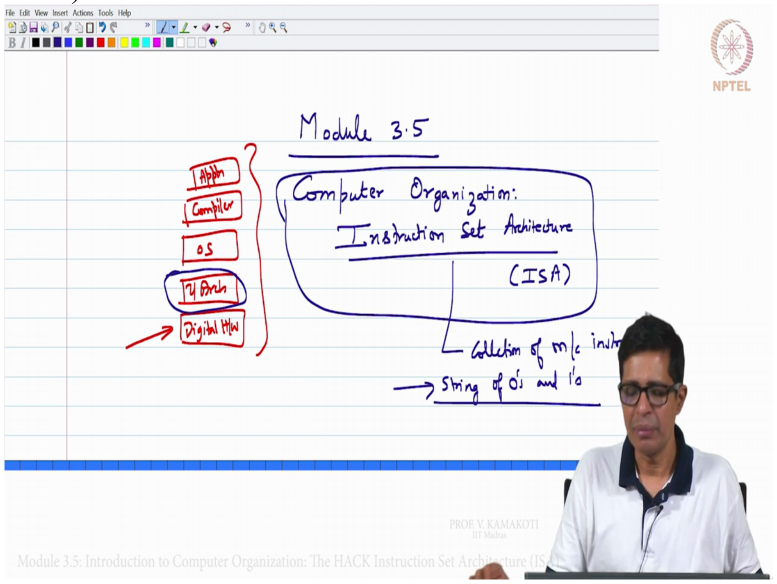
Foundations to Computer Systems Design

Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Module 3.5
Introduction to Computer Organisation: The HACK Instruction Set Architecture (ISA).

(Refer Slide Time: 0:17)



Welcome to module 3.5 and we now move on to the computer organisation. As I told in the introductory lecture which you have seen for this course, the entire system start is organised as, we have the digital hardware, then we have the microarchitecture, then we have operating systems, then compilers and then application programs. So the entire system can be viewed as these 5 layers, 1 on top of another, a stack of 5 layers. Now what we have done now is digital hardware.

So till module 3.4, we have done digital hardware, we have seen some sensible arithmetic circuits that could be formed. We started with a transistor and the D flip-flop as the basic element, we built quite a significant amount of, combinational circuits and a significant amount of sequential circuits. Now using these circuits, we are now going to assemble a system and that is what we call as the microarchitecture. So before building the microarchitecture, we should know what is it, what is it that we expect from the microarchitecture.

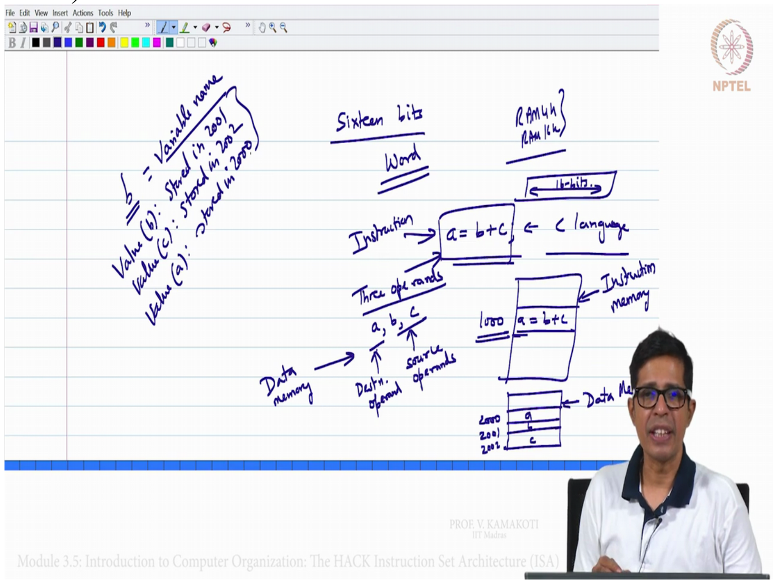
What are the expectations of the namely the compiler and the operating system on the microarchitecture? We need to understand that expectation. And that in my opinion, that is the matter of what we call as the subject called computer organisation. In computer organisation, basically we are going to talk about the interface of this compiler and OS with the microarchitecture. And that interface is basically moved by which the compiler and operating system can basically access the microarchitecture. How can the compiler and operative system access the microarchitecture?

They need to agree on some common mode by which the compiler and the operative system can talk to the microarchitecture. That common mode is a set of machine instructions. So we need to have a set of machine instructions using which the compiler and the operating system can talk to the microarchitecture. Those machine instructions are specified using what we call as an instruction set architecture. So instruction set architecture is nothing but a collection of machine instructions using which your application compiler and operating system can talk to the microarchitecture. Understanding the ISA is the major portion of the subject called computer organisation.

And so what we will do in this module and next module is to understand how we have formed the instruction set architecture. Now a machine instruction is nothing but a string of 0s and 1s. Please do understand that all these circuits that we have formed now are based on transistors and they all work by the principle they all work, they all give you sensible outputs based on whether these transistors are on or off. Right? If you take anything like sequential gates, like transmission gates, like your NAND gate, anything, NAND gate is a basic thing.

The D flip-flop is actually made out of transmission gates. All of these NAND gates and the transmission gates are built out of transistors at the transistors can recognise only 2 things, 0s and 1s, switch on or off. If it is P, if gate is 0, it is on, gate is 1, it is off and that is the way it [Inaudible 4:09] off that. So everything that we need to do, need to be converted to 0s and 1s. So if the OS wants to do something on the microarchitecture, it is to convert it to 0 and 1 and that is what is going to be executed on the microarchitecture. Now we will show essentially the instruction that is understandable by the microarchitecture need to be in 0 and 1 and that that is a tough instructions are called machine instructions. Okay?

(Refer Slide Time: 4:46)



Now machine instruction that we are going to use essentially will have 16-bits. We have already built up memory. So the instructions will always be in the memory. So the memories that we have built up, we built that RAM 4K, RAM 16 K, et cetera right? So whatever we have built so far, these memories can store in each location 16 bits. The 16 bits is called a word. So every instruction will be of size 16 bits and within that 16 bits, we will tell the machine what it is supposed to do. Right? Now before actually describing what those 16 bits will tell the machine, we will now just understand the model of computation.

Suppose I write, let us take A equal to B plus C. This is 1 command. This is in see C language. Right? Now where is this A equal B plus C? This command is basically, this command is an instruction which says add B and C and store it in A. Right? So this itself is an instruction. And it will be stored somewhere in the memory. So you have a memory, so all the instructions will be stored somewhere in the memory. So A equal to B plus C will be stored in see some 1000, address number 1000 in the memory. Now what this will tell? It will tell, add the 2 numbers and store it in some number. Right?

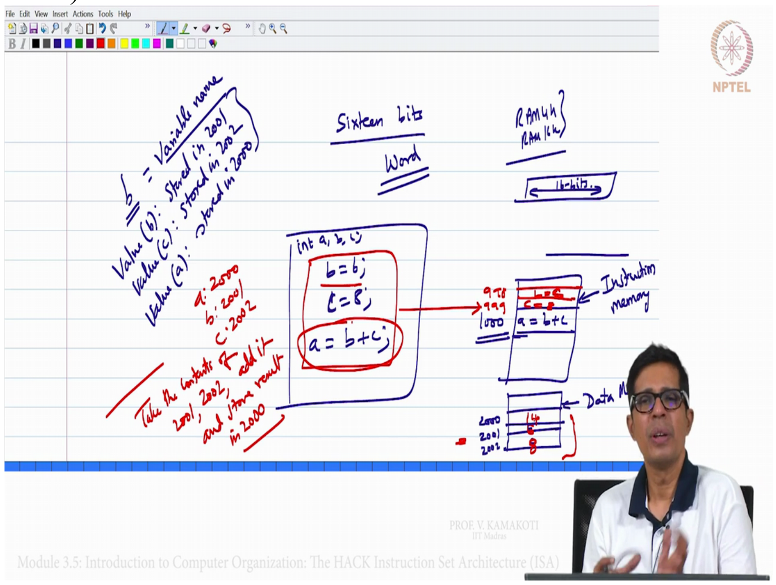
So add B and C and store it in A. That is what this particular thing will tell you. Now where is B? Where is C? And where is A? Right? Right? So this is the instruction and this instruction has 3 operands. The 3 operands are A, B and C. B and C are the source operands and A is the destination operand. B and C are the source operands and A is the destination operand.

Essentially, you will see that add 2 numbers and store it in the 3rd number and those 3 numbers are A, B and C. So B and C are the input to this operation and plus is the operation itself.

And the result of this operation is stored back in capital A. So there are 3 things that are involved. There are source operands, then there are destination operand and then there are, there is an operation itself. The operation in this case is plus, the source operations are B and C and the destination operand is A. Now where should this A, B and C be stored? Now somewhere this A, B and C are basically data. They will be stored in a data memory where A equal to B plus C is an instruction that will be stored in instruction memory while this is stored in data memory.

So A will be stored in some 2000, B will be stored in 2001 and C will be stored in 2002. Let us see. Right? So what do we mean by B will be stored in 2000? The value of B. B is nothing but a variable name. Right? The value of B will be stored in now 2001. The value of C similarly will be stored in 2002 and then the value of A ultimately when you are calculating will be stored in 2000. This is what the entire thing means, A equal to B plus C.

(Refer Slide Time: 9:55)



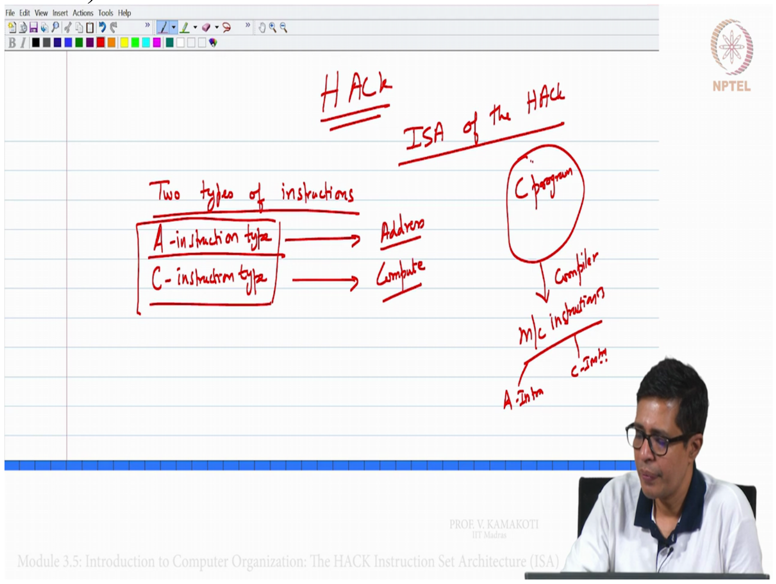
Now let us write a C program and a C program snippet and tell how this is going to happen. So I will say, A equal to 5, B equal to 6, sorry. B equal to 6, C equal to 8. A equal to B plus C. Right? Now when this program is compiled, the compiler will allocate some space for B. So it sees that B is a variable, C is a variable it also sees that A is a variable. Somewhere you are going to declare int A, B, C. Once the compiler sees A, B, C, they are data, they will be allotted space in the data memory as I am marking in the red colour here, okay? A, B, C.

So A is allotted to 2000, B is allotted to 2001, C is allotted to 2002. Right? Now this code B equal to 6, C equal to 8, these are all instructions. So these will be stored in the instruction memory, B equal to 6, C equal to 8. So this will be in 998, 999, et cetera. So these are all instructions. The moment I say B equal to 6, that means the location, where is A is stored at? 2000, B is stored at 2001 and C is stored at 2002. So when I say B equal to 6, that means the location 2001 will get the value 6 as you see here. I am marking here.

The location 2001 will store the value 6. The next instruction is C equal to 8. When C equal to 8, the location 2002 will get the value 8. Then the next instruction is A equal to B plus C. So what will happen? What is B? B is stored at 2001 and C is stored at 2002. So what does this A equal to B plus C mean? Take the contents of 2001, 2002, add it and store result in 2000. This is what this means. A equal to B plus C essentially means, take the contents of 2001, 2002, add it and store result in 2000. So it will take the content of 2001, it will take the content of 2002, add it and the result will be 14 and that will be stored in 2000.

So this is how a given program is viewed as 2, essentially 2 components. 1 is the instruction, 1 are the instructions that will be in the instruction memory and then these instructions will add on some data and those data will be in the data memory. So this is the difference between you know the instruction memory and data memory. Right? The instruction memory will have the instructions; the data memory will have the corresponding data on which this instruction is stacked. Right? So now when we take up program, a C higher-level program and want to translate it into a machine language, 0s and 1s then we need to understand these basic operations. Understand the first part.

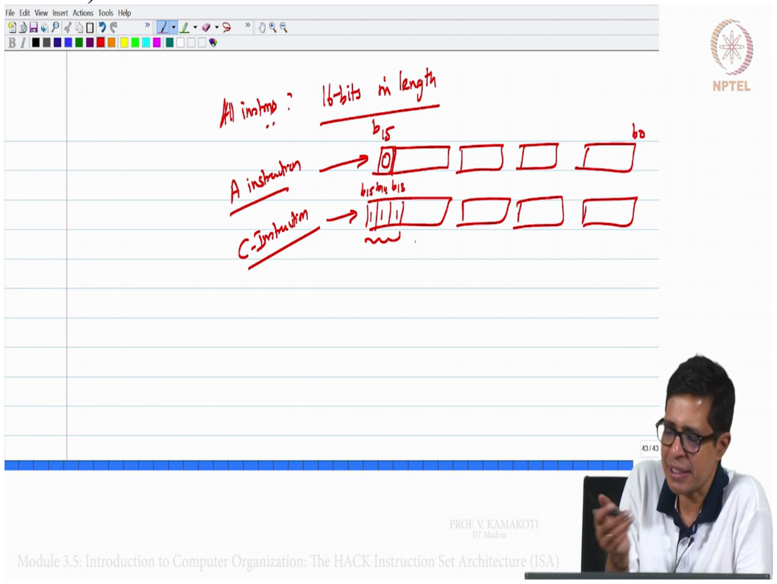
(Refer Slide Time: 13:50)



We will now describe how the entire the microarchitecture that we are going to builder, namely the hack, we call it as the hack, H-A-C-K. This is the microarchitecture that we will be building. What is the instruction set architecture, ISA of the hack? And that is what we will see now? Two types of instructions. 1 is called the A instruction, another is called the C instruction. A instruction type, C instruction type. Right? As the name or the abbreviation suggests, A is an, A stands for address, C stands for compute. Right? A is an address instruction. C is for the compute type. Right? So you have a C program. The C program you have compiled, right?

So the compiler will compile this and it will convert it into a machine instruction. That machine instruction, collection of machine instructions or sequence of machine instructions and those machine instructions will be either A instruction type or a C instruction type. These machine instructions will be of 2 types. Namely the A instruction or the C instruction. Okay?

(Refer Slide Time: 15:49)

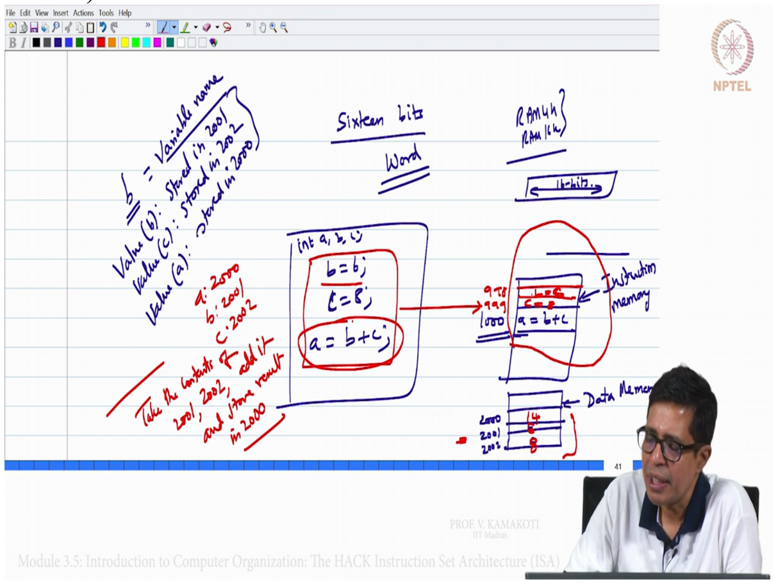

All instrns : 16-bits in length
b15
b0
A instruction
0
b15 b14 b13
C-Instruction
1 1 1
PROF. V. KAMAKOTI
IIT Madras
Module 3.5: Introduction to Computer Organization: The HACK Instruction Set Architecture (ISA)

So now we will describe the A instruction and the C instruction and then map it onto how this high-level language program can translate to that. Okay? Now these instructions as mentioned earlier will be 16 bits in length. So all instructions are 16 bits in length. All instructions are 16 bits in them. So A instructions and C instructions will be 16 bits in length. So we can split that 16 bits as 4 stems of 4-4 bits each. The A instruction will always start with 0 as the most significant bit. So this is bitter 0 to bit 15. The most significant bit is 0.

This means the instruction is a A instruction. If that first 3 significant, first 3 if the first 3 bits are 1, 1 and 1, then this points to a C instruction. This is how we have organised this entire thing. Right? So ultimately you will have a program in machine language which the microarchitecture has to take it and execute it. That machine language instructions will be of 2 types. The A instruction and C instruction. By just looking at the instruction, by looking at the bits in the instruction, 1 can decide whether it is an A instruction or C instruction.

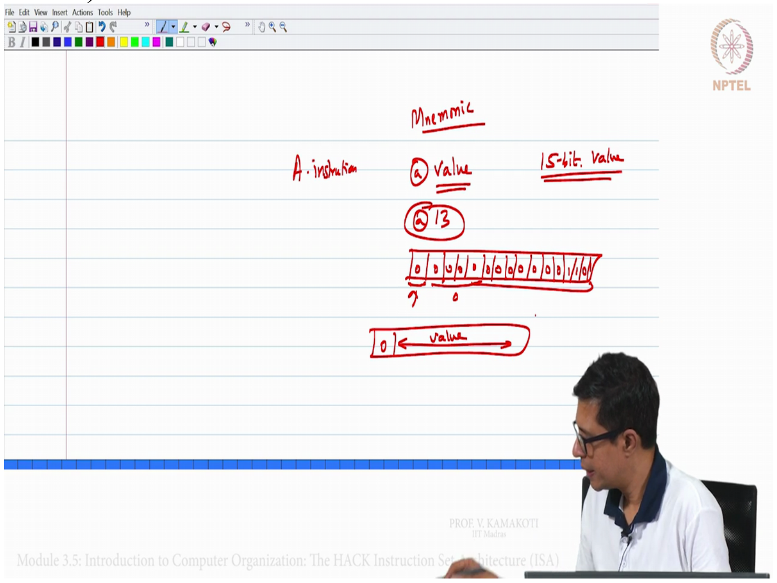
If the 15th bit is 0, then it is an A instruction. If the first 3 most significant bits namely B 15, B 14 and B 13 are 111, then it immediately becomes a C instruction. Right? So this is how we distinguish between the A instruction and C instruction.

(Refer Slide Time: 18:05)


b = Variable name
Value (b): Stored in 2001
Value (c): Stored in 2002
Value (a): Stored in 2000
a: 2000
b: 2001
c: 2002
Take the contents of 2001, 2002, add it and store result in 2000
Sixteen bits
Word
RAM4k? RAM16k
16-bits
int a, b, c;
b=6;
c=8;
a = b+c;
998
999
1000
a = b+c
Instruction memory
Data Memory
2000
2001
2002
14
6
8
PROF. V. KAMAKOTI
IIT Madras
Module 3.5: Introduction to Computer Organization: The HACK Instruction Set Architecture (ISA)

So when the microarchitecture reads instruction, the instruction we have shown in the previous part, so it takes 998, then 999, then the instruction at 1000 as you see here, I am marking it here. So when I see from 998, so each instruction, then it will find out whether it is an A or C by looking at the bit pattern. Right?

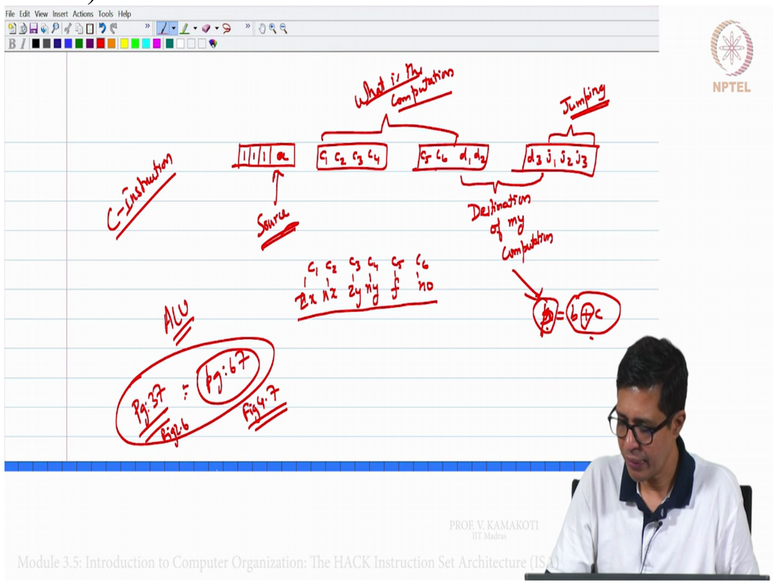
(Refer Slide Time: 18:34)



So now let us go to what will be an A instruction. A instruction is, so all these machine instructions for us to make it readable, we will have something called a mnemonic. Instead of writing it as 0s and 1s, we like to understand it in a more human readable form. So we will call it as mnemonic. So the A instruction, we will say, we will mention it as at value. This value should be a 15 bit value. Right? This value is a maximum it should be 15 bit. So when I say at 100, so let us say at 16, make it a little more, at 13.

So this instruction will become 0 because the A, the moment I see at, it is A instruction, 0 and then so this is it. So 0 is the first bit. This is 0, this is 13, right? 1101 is 13. So at value will be stored at, the first will be 0 and the value which is 15 bit, maximum 15 bit in length, that will be stored as a part. So this is the A instruction, a single instruction. So if I give 0, if I say at 12, at 13, this represents an A instruction where the first bit is 0, the remaining 15 bits will store this 15 bit value. Right? Right.

(Refer Slide Time: 20:48)



Now we will see a C instruction. C instruction is a little more thing. C instruction will be of this type. This will be 111 to start with. We have a bit called A, then we have C1, C2, these 4 bits, we name it as C1, C2, C3, C4 and C5, C6. Then we have D1, D2, D3, J1, J2, J3. Right? Okay. So this is how the 16-bits of a C instruction is interpreted. Right? Now we will just come to what is A a little later. C1 C2, C3, C4, C5, C6 will tell you what I should do, what is the computation. D1, D2, D3 will tell me what is the destination of my computation, where should I store the result of my computation? J1, J2, J3 will talk about jumping.

We will just come to that a little later, on jumping. So D1, D2, D3 will tell me the destination of my computation, C1 C2, C3, C4, C5, C6 will tell me what should I complete. This A bit will tell me something about the source. We already saw right? A equal to B plus C, we said B and C are source operation, A is the destination operation. So these 3 bits, D1, D2, D3, so we are seeing from here, this is the most significant bits. So this D1, D2, D3 will basically tell me about say let us say some P equal to B plus C, this will tell about P, the destination.

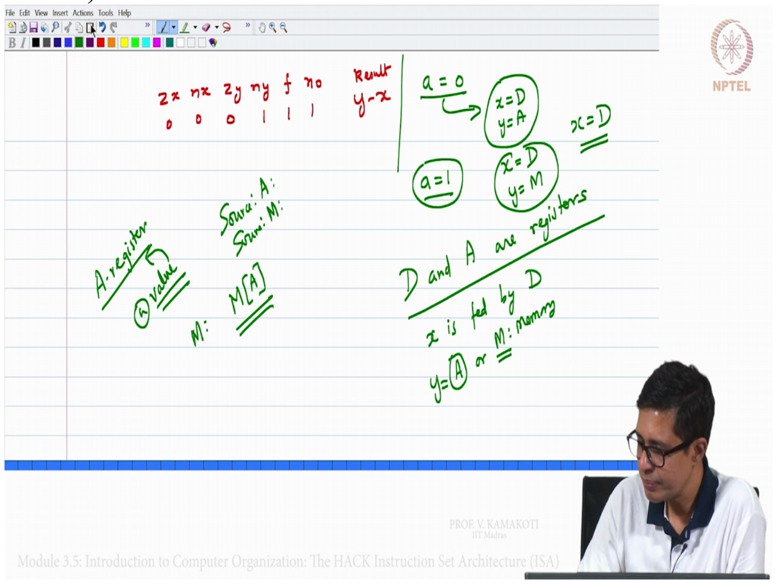
This C1 to C6 will tell me about plus, what is the type of operation? What should I do? And this, 1 single bit A will tell me about where can I find this B and C, it is about the source. Right? Jumps will just come a little later. So this is how the C instruction is organised. Right? So the A instruction basically will be, it will have 0 and then whatever is the remaining 15 bits will store some constant value, while the C instruction will basically give you all these things. Now before proceeding further, let us understand the most crucial part, the 6 bits.

When you design the ALU, right? sometimes before this C1 C2, C3, C4, C5, C6, you will remember that ALU had 6 control values. Namely ZX, NX, ZY, NY, F, NO. And for different combinations of this, ZX, NX, ZY, NY, F, NO, we were doing different types of computations. This is actually available in page 37 of your book. Right? Now the C1 C2, C3, C4, C5, C6 corresponds to that computation. Now this table is available in page 67 of the book. So you can look at page 37 of the book and page 67 of the book and you will see a one-to-one correspondence between C1 ZX, C2 NX, C3 ZY, C4 NY, C5 F and C6 NO. Right?

By assigning different values 0s and 1s to this, we were achieving different types of results of X and Y. The same thing is going to happen with this. Right? So there is a one-to-one correspondence between the table in page 37 which is figure 2.6 of your book and figure 4.7. Why this understanding is important? Because this is exactly how your the microarchitecture that you are developing will interface with the ISA. Right? With interface with the instructions. How is the machine instruction basically interpreted by your microarchitecture?

You are getting some 0s and 1s. How will the architecture basically understand what should it would do with the 0s and 1s? So these are all the small small steps by which you understand how the machine interprets your bits that it is receiving. So the C1 to C6 essentially maps onto ZX, NX, ZY, NY, F, NO.

(Refer Slide Time: 26:55)



So let us take 1 example and I will tell you, so let us take say for example, if you take to figure 2.6, if you see, this is what will happen. ZX, NX, ZY, NY, F, NO. Let us take 1 example which is 000111. 000111. Right? This will do Y minus X where X and Y are inputs. The source operand is basically dictated by this A bit. Right? This A bit, this is the 4th most significant bit, determines what is the A operand. Now in this case, we will make a distinction, if A equal to 0, then X is equal to D, Y equal to A. If A equal to 1, X is equal to D, Y is equal to M. Right? So if A equal to 1, X is equal to D, Y equal to M, A equal to 0, X equal to D, Y equal to A. Right?

So the A bit will determine what is X and what is Y. Always X in our case will be D and always Y, Y will be either A or M depending on whether A equal to 0 or A equal to 1. Right? So what is this D and what is this A? D and A are registers. D and A are basically registers. Right? So they are they are registers which are feeding this X and Y. Always X is fed by D. As I say X, that is the output of the D register. We have already seen what is a register in our previous project, right? So X is fed by D and Y can be fed either by A or M. M is nothing but memory.

We will now see what is M but A is a register. Right? So D and A are registers, M is actually memory. We will see what is memory a little later. But depending on the value of A, the value of Y, the who is feeding Y will change but X is always D. Right? Okay. Now we will basically say what is the A register. When I say at some value, this value basically gets stored in the A register.

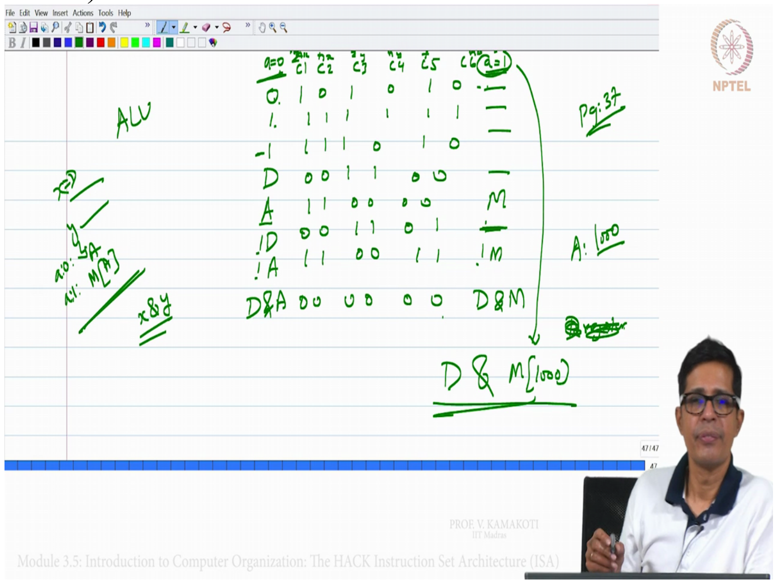
This value basically gets stored in the A register. I can use that value as that value itself or I can use that value to address a memory location.

For example, when I say M, it is nothing but when I say memory, means I should give the address to which I should access that memory. This is nothing but M of A. M of the content of your A register. Okay? So to sum up, now there are 2 registers. So this is the ALU. The ALU that we have designed had 6 control bits right? ZX, NX, the same 6 control bits are going to be so for me to specify what the ALU should do, my instruction will have those 6 control bits in the sequence namely C1 to C6. Now the source operand, basically there is the source to the ALU, we had 2 sources X and Y which are 16-bit numbers.

The X will always be fed by a register called the D register, Y will be fed by two fellows, either it can be A or it can be M. A is a register again, a 16-bit register, M is nothing but a memory location whose address is stored in the A register. So if I am, if my source operand is A register, then it is coming directly from the D register. If my source operand is M register, that means it is the location in the memory whose address is stored in the A register. Okay? So this is how we have to interpret this.

So this bit A will basically tell me whether my other operand is coming from a A register or it is coming from the memory. And if it is coming from the memory, which address in the memory? That address is stored in the A register. If it is coming from the A register, you just read it from the A register. If it is coming from the memory, then the address, which address it is coming from? It is the address that is stored in the A register. Right? So this is how this particular C instruction is interpreted.

(Refer Slide Time: 32:48)



ALU
Pg: 37
A: 1000
D & M[1000]
D&M
x&y
PROF. V. KAMAKOTI
IIT Madras
Module 3.5: Introduction to Computer Organization: The HACK Instruction Set Architecture (ISA)

Now we will just see, now we will we will go through this so you have already done the ALU, so we will basically go through this particular thing. So we have C1 C2, C3, C4, C5, C6 and we basically have 101010, 111111, 111010, 001100, 110000, 001101 and then 110001. So we will take these 7 examples. Totally there are 18 such cases. So as we see there we used to get a 0. The output of this operation would be 0, 1 minus 1, C1 corresponds to NX sorry ZX, ZY, NY, F, and NO. So just also refer the page 37 table. So we will also see the one-to-one correspondence.

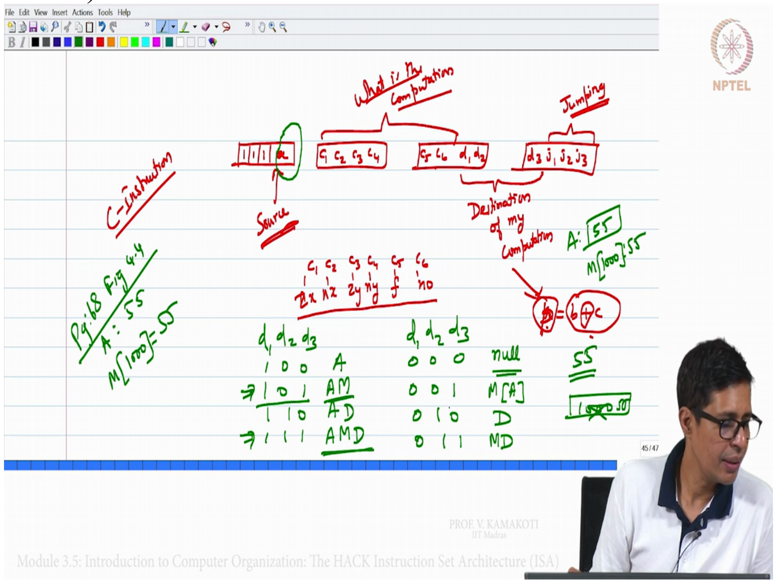
So you actually get for this combination, you get 0, 1, minus 1. Then you actually get X, you get Y, then you get not X, not Y. This is what you get as per that table. Now when A equal to 0 right? Now we have to interpret it in this context of A. So when A equal to 0, you get 0 as the output of the ALU, 1 as an output minus 1 as an output. X will be always D. So this is going to be D. Y will be when A equal to 0, it will be A. This will be not of D, this will be not of A.

When A equal to 1, this will be, this is do not care, we will not get this combination because nothing do with this. So this will be blank. This will be blank. Now this will also, D will also be blank. Instead of A, when D equal to 1, you will get M. What is this M? M the there is some value stored in the A register, say 1000. M is the content of that 1000th memory location. Similarly, here you will get same thing. So here you will get not M. So whenever you see a A here, you will see the corresponding M there.

So when A equal to 0, this is what is happening, when A equal to 1, this is what you will get. This is the source operands. So let us go and say 1 more case like 000000, that is actually in the table, in page 37, you will say X and Y. So this will be D and A and here it will be D and M. Right? So the moment I see 000000 in the C bits of your C instruction, go and take and I also say A equal to 0, what will A do? I will go and say, I will take the content of A register and the content of D register and add them. Right?

And store it in some destination. Destination will come a little later. But if I see 000000 and I also see that A equal to 1 there, then what I do? I take the content of the D register and the content of, suppose A is storing say 1000, I take the content of M of 1000 and the content of D register and add them. That is what I do when I see A equal to 1. Right? D. So in the ALU, when I have X and Y, depending on this A bit, this is always D. If A is 0, this is going to be A and if A is 1, this is going to be M of A. So you get an out of this ALU.

(Refer Slide Time: 37:51)



Where should it get stored? So this, this is there are 3 bits for this and these destinations, so we could have a different destinations. Right? Because we have 3 bits and that is given in page 68, figure 4.4 of the book. Right? Now we will see that table very quickly here. D1, D2, D3, 000, this is null. This type of a thing will never come. The destination cannot be null. Right? 001 means memory. M of A. At that, there will be an address A. So go to that, 001 go to that memory location given by that address.

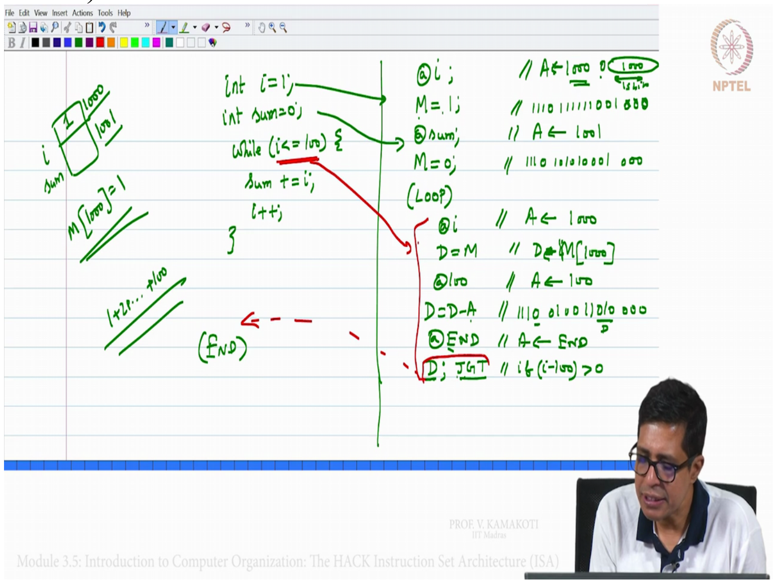
So A register has some value, go to that location pointed out by the A register and that is where you store the result. 010 means store the result in the D register. 011 means store the result both in the memory and the D register. Right? So MD means store both, so I can store it into locations, M and D. Similarly, the other combinations, I could have 100 means store it in A, 101 store it in A and memory and 110 store it in A and D and 111 store it in A, memory and D. Okay? Right? So these are all the different combinations. When I say memory, the address for that memory is always in the A register.

So in that way, these 2 combinations are slightly tricky in the sense that first you take so the answer is some 55 which I want to store in some destination, the address A has 1000. If my destination operand is 101, that means first I have to read this 1000, I have to store the answer in both A and M of 1000. So I have to read 1001st, then keep it here, so M of 1000 and store 55 in both these locations. If I first go and store 55 here, then that demo of A then I will be going and writing that 55 in location 55 also.

Do you understand? So the main thing is M means M of A. So when I say A, M, that means first I read the value of A, find the location where I need to store and then go and update it. The moment I update A and then start accessing memory pointed to by A, then it does not make any sense. So first I should do M of A, so in the A if 1000 is store, first I have to find out M 1000, first I have to read out 1000. So I will know I have to write it in M 1000. Now go and write into A and M 1000, 55. This is what I need to do.

The same care has to also be taken when I am writing into A, M and D. Right? Normally these 2 combinations rarely come with the other things. So this basically tells you why should they write back the result. Okay? Right? And the last 1 is about jump instructions.

(Refer Slide Time: 41:58)



```
int i=1;
int sum=0;
While (i<=100) {
    sum += i;
    i++;
}
(END)
```

```
@i;       // A←1000 ? 1000 15 bits
M=1;      // 1110 111111 001 000
@sum;     // A ← 1001
M=0;      // 1110 101010 001 000
(LOOP)
@i        // A← 1000
D=M       // D←M[1000]
@100      // A← 100
D=D-A     // 1110 010011 010 000
@END      // A← END
D; JGT    // if (i-100) > 0
```

i
sum
1
1000
1001
M[1000]=1
1+2+... +100
File Edit View Insert Actions Tools Help
PROF. V. KAMAKOTI
IIT Madras
Module 3.5: Introduction to Computer Organization: The HACK Instruction Set Architecture (ISA)

So let us see this particular program. Say I want, I have int I equal to 1, I will write a C program. Int sum equal to 0. While I is less than or equal to 100, sum plus equal to i, i++, end. So this is going to sum 1 plus 2 plus 3 till 100. Right? So this code, how will I translate it to machine language as A instructions and C instructions? Right? Wherever i is mapped, so i is basically an address right where i is stored. So i any variable will have 2 attributes, 1 is the address and another is the value of that.

So when I say at i, then immediately the address to which I is mapped, namely 1000 will be loaded into the A register. This is what we will happen. Right? Now I will just say now M equal to 1. Right? So what is M? M is memory. Memory, the moment I say M, which memory? Memory whose address is stored in A. So M of 1000 now becomes 1one. That is what this means. So i will essentially this means I am setting i equal to 1. Okay? Right? Now, please understand, this is an A instruction. This is a C instruction in which in which the C bits are 111111 and the A is 0.

That means, the answer will become 1 and your destination has to be M. If the destination has to be M which is 001, this is your destination. So the C bit will be as seen here, the C bits will be 1111111. So if you go and see, look at this instruction here, your C bits will be 111111 and your D bits will be 001 which will correspond to your M and the A bit will be 0. This immediately says that computing the value 1 by the ALU and store it in M of A where A is the address register. Okay? Right?

So that is how M equal to 1 will be translated. So corresponds to the machine instructions where this corresponds to a C instruction where the first 3 bits are 111, the A bit is 0, then all the 6, additional bits are 111, then the the destination bit is 001 and of course there is no jump here that can be [Inaudible 45:29]. Or normally this should be 000. No jump. Right? So this is the machine 16 bit C instruction corresponding to this. This at i is nothing but 0, that is the A bit. This is an A instruction, so the first bit will be 0 and the binary equivalent of 1000 will be stored here in the 15 bits.

So this is an A instruction, this is a C instruction. Right? Now this is how the first statement gets translated. Now we have in some equal to 0, again we say sum some, now this is an A instruction in which the A register will be loaded with 1001 because sum is mapped on 2000. Now I can say M equal to 0 right? So this will again become a C instruction where again 0 means what? So let us see what is 0. 0 means 101010. That will be this. And A is 0. So A is 0 and 101010 and then the destination is again M and no jump.

So this will be the C instruction corresponding to this M equal to 0. So these 2 are done. Now I am doing while i less than or equal to 100, so 1 of the thing is that we can introduce labels here. I look here, so when I put anything within parenthesis, it eventually means it is a labourer. What we do? First at i first I go and compare whether i is less than or equal to 100. How do I do? First I access i, I store i in the D register. Right? So D equal to i. I store i in the D register. At i, that means your A register will now store 1000 and D equal to M of A means the content of the 1000 will be loaded into D, that is the value of i will be loaded into D. So initially, the value of i is 1 right? So 1 will be loaded now I will just say at 100, that means in the D register, 100 will be loaded.

I will say D equal to D minus A. Right? D equal to D minus A. Now this is a C instruction. This is very interesting. So I am doing X minus Y. So 111 and the Y is A. I want A, I do not want capital M. So the A bit is 0. That means, for the ALU, I want X minus Y as an output from the ALU. So and that Y should be A. So if I put A equal to, that A bit the 3rd bit as we see here, this bit when I make it 0, then it is A. If I make it 1, then it is M. So I make it 0 here. Right? And then what do I do? I want X minus i, right?

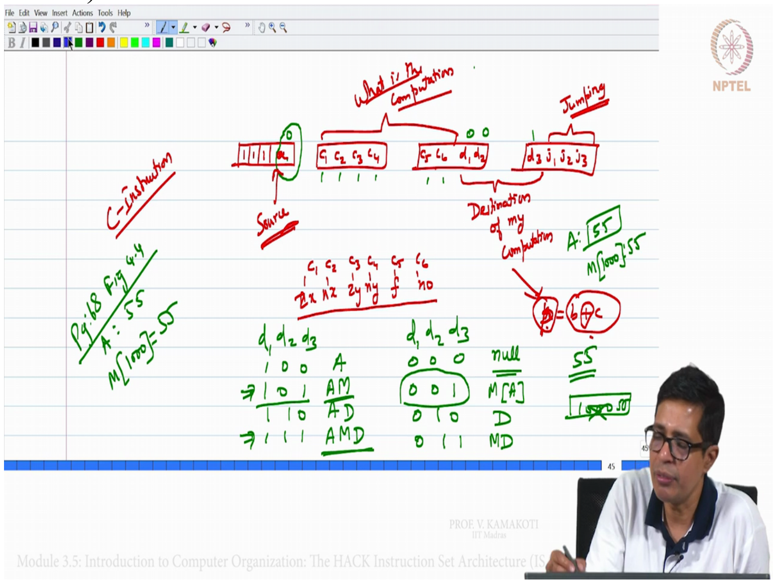
So X minus Y when you see in your either page 67 or page 37, your X minus i is 010011. 010011, this is the ZX, Annex or your C1 C2, C3. So this will give me D minus A. And where you want to store that? You want to store it in D. So the destination is 010. This is the destination part, it will go to D and that is no jump. So D equal to D minus A. Right? Now what we do is very interesting.

At END. END is a label which I am going to put somewhere down here. This will also be mapped onto some memory location. So when this is loaded in memory, this at End will be some memory location. So at End, so your A will get the label of End. So this, suppose I start with 1000, this will be someone 1500, so 1500 will be loaded here. Right? And I know say D; JGT. So what we are saying is by this, this is a machine and structure. If the content of D, what is content of D? D today the content, I have made D equal to D minus A.

So the content of D is i minus 100. Right? A is 100 here. So when I say D equal to D minus A, I have actually computed i minus 100. So D basically has i minus 100. If D is greater than 0, if i minus 100 is greater than 0, jump to end. JGT means the D register that you see here, is it greater than 0? JGT means jump and greater than 0. If i minus 100 is greater than 0, jump to some End. So we will just continue here. If right? That means from here, the control will go to this end. From here, the control will go to this end if your D is greater than 0.

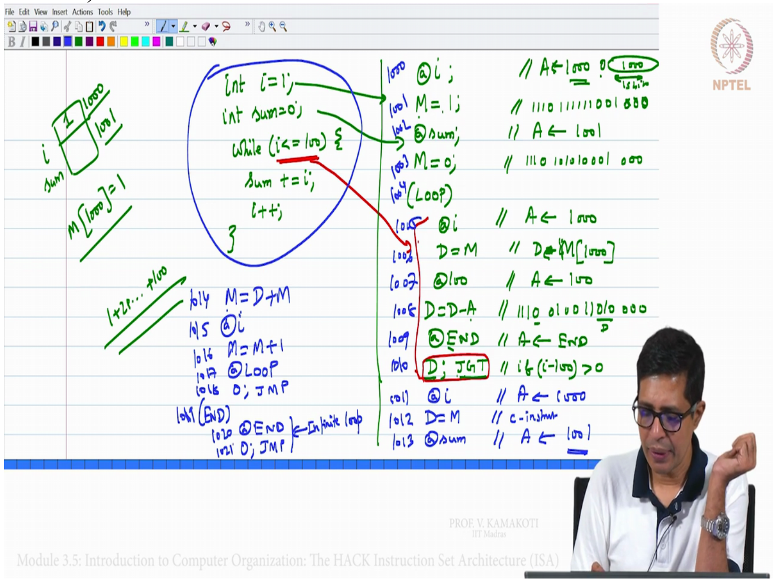
If the D is less than 0, that means this while loop is yet to finish, I have to do something more. So this instruction, that is a tough instructions that I have put here basically checks the condition is y i less than or equal to 100. Right? And this is how this is. So how will this D; JGT is going to get translated, we will see a little later but this will also be an A instruction.

(Refer Slide Time: 52:39)



As I told you earlier, as we saw earlier in the C instruction, there are 3 bits that are reserved basically for as you see here, for the jump and that is what will be used for these jump instructions. Okay?

(Refer Slide Time: 52:56)



```
int i=1;
int sum=0;
While (i<=100) {
    sum +=i;
    i++;
}
```

```
1000 @i;          // A←1000 : 1000 (15 bits)
1001 M=1;         // 1110111111001000
1002 @sum;        // A←1001
1003 M=0;         // 1110101010001000
1004 (LOOP)
1005   @i         // A←1000
1006   D=M        // D←M[1000]
1007   @100       // A←100
1008 D=D-A        // 1110010011010000
1009 @END         // A←END
1010 D; JGT       // if (i-100)>0
1011 @i           // A←1000
1012 D=M          // c-instruction
1013 @sum         // A←1001
1014 M=D+M
1015 @i
1016 M=M+1
1017 @LOOP
1018 0; JMP
1019 (END)
1020 @END   } ← Infinite loop
1021 0; JMP }
```

i
sum
1
1000
1001
M[1000]=1
1+2+...+100
PROF. V. KAMAKOTI
IIT Madras
Module 3.5: Introduction to Computer Organization: The HACK Instruction Set Architecture (ISA)

Now this is done. Now now after that, we have to do this sum equal to sum plus i and i plus plus. So how do we do that? First we take again we bring in at i right? Right? And T equal to M. at i is now an A instruction, D equal to M is again another C instruction. Okay? So at i now A will become 1000 and D will become M of that 1000. So whatever is the value of i and then we will now bring sum, so again A now at sum, A will now become 1001 right? And then A will now become 1001, we will continue the program this side. Then we say M equal to D plus M. So so M M is M is pointing to some number.

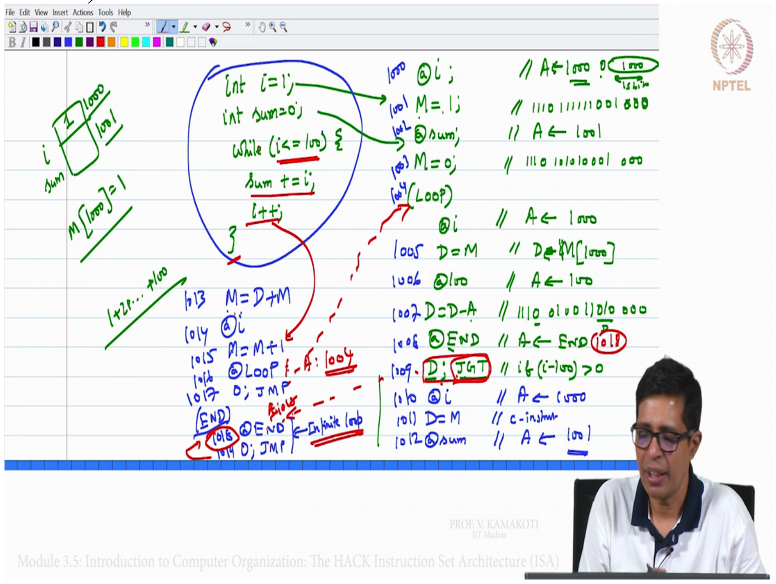
Because your A register has 1001, your M is nothing but M of A. M of 1001, that is sum. So your M is now pointing to sum here because D register has pointed to sum. So sum is equal to D. What is D? I I plus sum. sum equal to I plus sum comes here, right? Now again we see at i now M will point to i. So M is equal to M plus 1. This is incremental. Then at loop and I say at loop and we say 0,; jump. So you go to jump and here we see at M and 0,; jump. We will just do an infinite loop here which is equivalent to halting. Okay?

So this is the entire program this is how this C program can be compiled into our A C instruction machine language. So before winding up, I will again go through this in in a great in in some more understanding at a little more abstract level. When I say M, it points to the address it it points to the content of the address in the A register. So if I see at i and immediately say M, it means the value of the variable i. At i is the address where i is stored. If I say M, M is at i is the address of where i is stored. The moment I execute at i that address where i is stored is stored in the capital a register.

The moment I say M, it essentially means I am referring to the value of i. Right? And if I say M on the right-hand side, essentially it means the value of i. If I say M on the left-hand side of the question, writing with the address of i. So if I so I say at i and I say M is equal to D plus M, that means I am adding to the content of D, the value of i and storing back that result back in i right? You got it? So if I say at i, the A will get that i value. Now I say M equal to D plus M. Now let us say this is 1005 location, i is at 1005, for just our understanding. 1005 has 6, has storing 6. No D register has some 7.

Now when I say at i, the A register will now store 1005. When I say M equal to D plus M, 6 this 6 will be brought from 1005, added to 7, that value 13 will be stored back in this 1005. Sober 1005 will now become 13. So this is what this at I essentially means. With that as a background let us quickly at this program. At i M equal to 1. Essentially it will make i equal to 1. At sum M equal to 0, it will make sum equal to 0. Now we are introducing a symbol loop. So if this starts at 1000, right? This will be 1001, 1002, 1003, your loop will become 1004. Right? So this instruction will be 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 14, 15, 16, 17, 18, your 1000, N 1019, 1020, 1019, 1020, 1021 okay?

(Refer Slide Time: 58:58)



Or to be so let me just number, so this is 1004, so this loop corresponds to this instruction but 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, this will be 1018, 1019. So this, these are all basically 2 and End or called the symbolS which we will be using here. So what will happen is at i M equal to 1 will make i equal to 1, at sum M equal to 0 will make sum equal to 0. Now we will start checking i is less than or equal to 100 and that is the start of the loop. So we just give the a symbol loop which is mapped on 2004 and this the assembler will do.

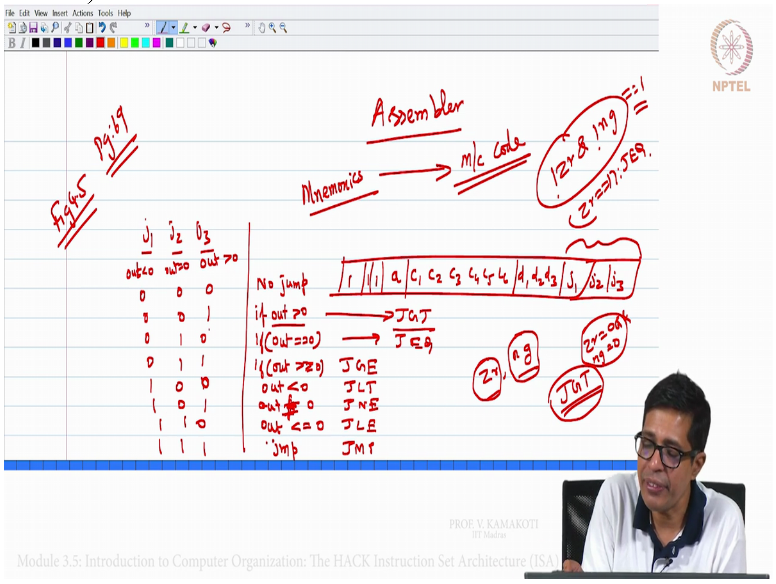
So at i again will give i. D equal to M, so D will get the value of i, initially it will be 1. At 100 so the value 100 will be stored in A. I will do D equal to D minus A. That essentially I am doing i minus 100 because D is storing i and A is a storing 100. Now load A with end. A with end means A is now mapped onto please note here 1018. End is now mapped onto 1018. So when I say at end, A will get 1018. And now I see D;JGT means D if the D register has a value greater than 0 jump to the address stored in the A register. What is that A register storing? 1018. So if D is greater, if the content of D is greater than 0, then you jump to 1018.

This control will come to this. Right? That means this while loop will end. This is the part. Otherwise what you do at i, D equal to M. Again D becomes the value of i, at sum M equal to D plus M. So D is i sum equal to i plus sum. That is what we are doing here. Because at sum and I say M, M will be sum. Again at i,so and M equal to M plus 1 i equal to i plus 1. This we have done. Then at loop. At loop is 1004. So here, your A will get 1004. So 1017 essentially says do not check anything. If I put 0 here, do not check everything, just jump to the address in A register which is 1004. So the control now goes back to this and again you start checking for while, while i is less than or equal to 100.

So you are seeing and now become here. After you finish, i loops out 100 times, basically this 1009 whatever condition in 1009 will get satisfied and the jump will get taken here. And when you come here, attain this 1018, so A will get 118 and we go to 1019 and see it will jump to 1018 again. So this will just keep rotating as an infinite loop between 1018, 1019 and that will be the sort of an halt condition. So this is the interpretation of how your C program essentially gets translated into a set of A and C instructions, right? A and C instruction and basically it getting executed.

So we have not written the entire set of A and C instructions in 0s and 1s because it is not human readable. So we are writing it using some at symbol and M and etc. So but then there will be someone who will take these symbols, this sort of pseudo-English command and converting it into 0s and 1s and that program is basically called and assembler.

(Refer Slide Time: 63:31)



The assembler will basically take your mnemonics as you see here and converted into the machine code or machine language, the 0s and 1s. Right? So the last thing that we need to see before we go into more of the project is the jump instruction. So as we see in the C instruction, how is the C instruction constructed? We have 111 and then we have the A bit, then we have C1 C2, C3, C4, C5, C6, then we have D1, D2, D3, then we have J1, J2, and J3. These 3 bits will tell us something, right? So the J1, J2, J3 bits are there. So we will see figure 4.5 of page 69 of the book and that will basically give you this table for J1, J2, and J3.

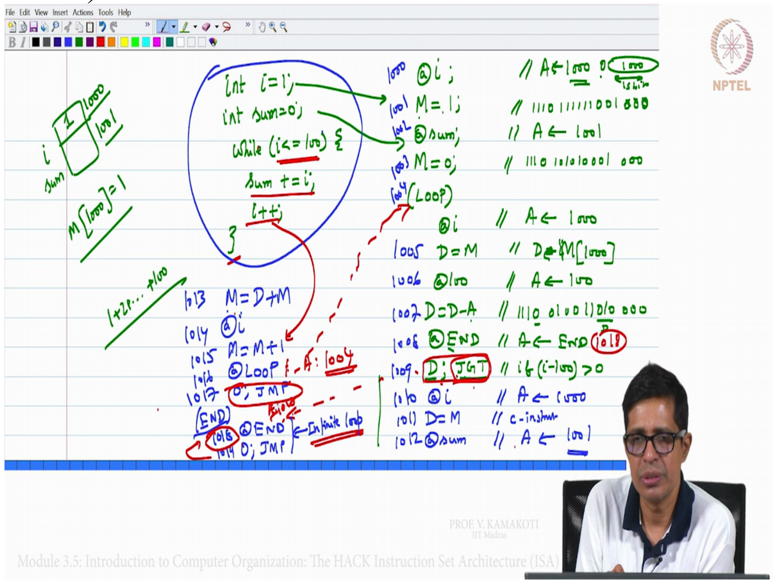
J1, this bit J1 stands for outer less than 0, this will stand for out equal to 0 and this will stand for out greater than 0. Okay? So 000, 001, 010, 011, 100, 101, 110, 111. Okay. This 000 essentially means no jump. This means if ALU output is greater than 0, jump. This, if ALU output is equal to 0, jump. If out is greater than or equal to 0, this is JG, that points to these 3 bits, 111. So this will be J greater than, J equal to, this is J greater than or equal to, JGE. J less than, J not equal to, J less than or equal to and normal jump. This is the thing.

So if the ALU output is depending on the ALU output, we take this. So how do we know this ALU output? If you carefully look at ALU, there were 2 parts. The ZR part and the NG part. So the ZR part told whether the ALU current output is 0, the NG part basically told whether the ALU current output is negative. Right? So depending on that combination of ZR and NG, we can decide whether to which which, whether to take the jump or not. Suppose I say jump when greater than and if greater than, jump when greater than 0 and you then it will take when ZR equal to 0 and NG equal to 0, this jump will happen.

When ZR equal to 0 mixed output is not 0 and NG equal to 0 means output is not negative. Obviously the output is greater than 0. So JGT can be realised as when ZR equal to so JGT is nothing but not of ZR and not of NG. If this condition, if this is equal to 1, then you take that jump, JGT will happen. Similarly, JEQ will happen when ZR equal to 1, then JEQ will happen, that is the jump will take less. Otherwise no jump will take place. So this is how we can interpret.

So by using the ZR and NG Flag, these jumps are basically executed. Okay? So the output of the ALU is basically utilised.

(Refer Slide Time: 68:20)


File Edit View Insert Actions Tools Help
i
sum
1
1000
1001
M[1000]=1
1+2+... +100
Infinite loop
A: 1004
PROF. V. KAMAKOTI
IIT Madras
Module 3.5: Introduction to Computer Organization: The HACK Instruction Set Architecture (ISA)

```
int i=1;
int sum=0;
While (i<=100) {
   sum +=i;
   i++;
}
```

```
1000 @i;        // A←1000
1001 M=1;       // 1110 111111001 000
1002 @sum;      // A ← 1001
1003 M=0;       // 1110 101010001 000
1004 (LOOP)
     @i         // A← 1000
1005 D=M        // D←M[1000]
1006 @100       // A← 100
1007 D=D-A      // 1110 010011010 000
1008 @END       // A← END 1018
1009 D; JGT     // if (i-100) >0
1010 @i         // A← 1000
1011 D=M        // c-instruction
1012 @sum       // A ← 1001
1013 M=D+M
1014 @i
1015 M=M+1
1016 @LOOP
1017 0; JMP
(END)
1018 @END
1019 0; JMP
```

So similarly here. See this JGT, now we are taking the D is the computation. So this is the output. If D is going to be greater than this, then we take it. Otherwise we do not do anything. So so I had to calculate D. How do you calculate D here? D is X, right? So how do you calculate D? So we go back to here. D. So in the C instruction so so I this is a C instruction 111, A is 0 and I put this computation with 001100. So this is the the A bit and this is C1 to C5. Okay? Right? Then the destination is not necessary here.

The D1, D2, D3 can be anything. I have jump. Jump is JGT. So JGT is nothing but 001. So this means that take D, just compute D in the ALU. Right? Compute D in the ALU and now you go and check whether the answer the content of that output is either greater than 0 and if it is greater than 0, take a jump. So that is what this basic thing means. So the jump will also be modelled as these type of C instructions. Wherein there will be no destination involved but there will be conditions involved and the computations here can be any of those 18 computations. 18 into 2, 36 computations depending upon A is 0 or one.

Different things that you do, that is part of your figure 4.3 in page 67. So all those computations can be done and output of that computation depending on that, I can do 8 varieties of jump, whether it is greater than, greater than or equal to, less than, et cetera. 1 interesting thing is this unconditional jump where no condition is there. If I put 111, then I do not care about all the other things, I just do a jump to the A address and that type of jump is extremely necessary because of some of the things that we have seen here like 0: jump where I need to jump without taking any condition I am going to jump at the end of this value.

Right? So in essence, this is the basic introduction to the machine language which comprises 2 instructions and then there is an assembler which is basically going to take your mnemonics. As you see here, it is is going to take your mnemonics language and convert it into machine code and that small bit of software is called the assembler. Okay? So in the next module, we will basically see the working of an assembler, we will code some programs and we will see the working of an assembler as we proceed in the next model. Thank you.