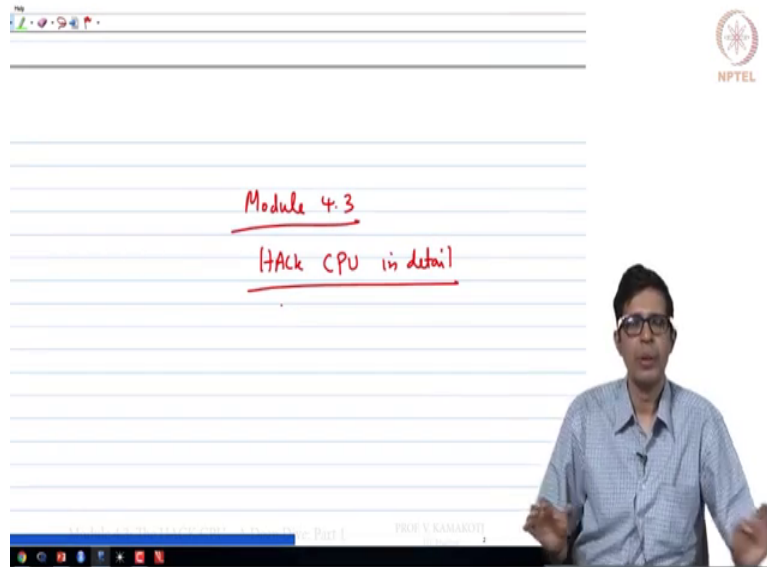


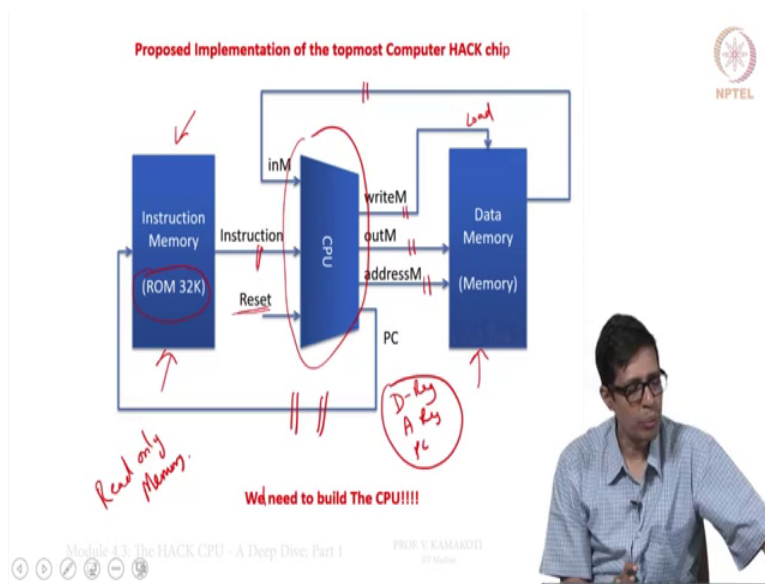
Foundations To Computer Systems Design
Professor V.Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 4.3
The HACK CPU- A Deep Dive: Part 1

(Refer Slide Time: 00:16)

A video frame showing a whiteboard with handwritten text in red ink. The text on the whiteboard reads "Module 4.3" and "HACK CPU in detail", both underlined. In the foreground, a man wearing glasses and a light blue shirt is speaking. The NPTEL logo is visible in the top right corner of the video frame. At the bottom of the video frame, there is a Windows taskbar and a footer that says "Part 1" and "PROF. V. KAMAKOTI".

So welcome to module 4.3 in this module we will discuss about the HACK CPU in great detail. We have seen what the HACK computer will look like, the computer controls three things the ROM what the instruction memory, instructions will be stored and then the CPU and then the data memory. In this particular module we will see the CPU in detail.

(Refer Slide Time: 00:47)



Look at this particular this is the top most view of the computer the HACK computer, you see an instruction memory here, the instruction memory is the read only memory so this is a built-in chip that will that is available for us. We have not build a read only memory it is already available to us and we can use that.

What is the difference between the data memory and the instruction memory now? You don't write into instructions you basically only read instructions so the CPU will not write the into a instruction memory it basically reads the instruction memory and executes the instruction. So that is why instruction memory will be a read only memory and this memory can if of size 32 kilobytes and then there is a data memory which is read write memory so we can read and write into data and so this is the CPU which we are going to discuss in this module.

The CPU there is a reset which is given externally to the CPU the reset will go and reset the program counter the D register and the A register there are three storage elements as we have mentioned so far there is a D register, A register and the program counter and this three are basically reset when you give the reset pulse and the CPU will take the instruction one by one from the instruction memory, which instruction it will take? That instruction which is pointed out by the program counter, that is the instruction it will take.

So that please note that there is indeed a need from the CPU to the instruction memory which is through the program counter. Once the program counter is sent to the instruction memory it will

send the instruction the corresponding instruction, the CPU has interpret that instruction, execute that instruction as a part of that execution it can read from the data memory if the source operands are memory then it an read from the data memory in which case if this the address and it basically collects the data or it can also write into the data memory in which case it can write it to the data memory in which case it has to give the address it has to give what data to write and it also have to enable the memory the load of the memory to see that in this particular address the corresponding data is written.

So all this three (two) activities the CPU can do the data memory. So this is the overall organization. Now what we will understand in this module is an instruction comes a 16 bit instruction basically flows from the read only memory to the CPU and how does the CPU basically executes it. Now very quickly to recap what are those 16 bits in that instruction?

(Refer Slide Time: 4:01)

A - instruction: @value //Where value is either a non-negative decimal number //or a symbol referring to such number.

Binary: 0 VVV VVVV VVVV VVVV

value (v = 0 or 1)

C - instruction: dest = comp ; jump // Either the dest or jump files may be 000 // If dest is zero, the "=" is omitted; // If jump is zero, or a symbol referring to such number.

Binary: 1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3

Module 4.3: The HACK CPU - Deep Dive: part 1

NPTEL

The 16 bits the instructions are of two types again, A instruction and C instruction. Now if the first bit (the) so we call it 0th bit to the 15th bit, so if the first bit is zero then it is an A instruction if the first three bits are 1 1 1 or for a addition of the first bit is 1 for sure then it is a C instruction.

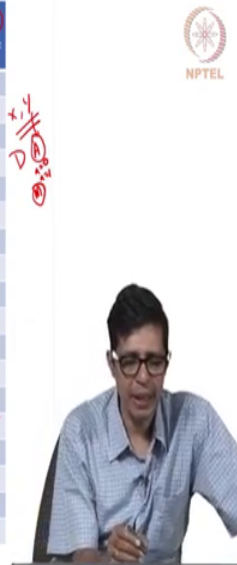
If the first bit is zero it is an A instruction if the first bit is one then it is a C instruction right. If for A instruction the remaining 15 bits is value and this value will be basically stored into the A register. So if I have an A instruction this values will be stored into the A register and similarly if

I have a c instruction then the entire thing happens then there are this we have to interpret this 13 bits, the first there is an a bit that will tell you whether the source operand is from the A register the source operand to the ALU one of the source operand so the A register has two source operand, the ALU has two operands, source operands x and y this A register the a bit small a bit will tell you whether one of the source operand is a A register or from the memory.

So if a is zero then it is from the A register if a is one then it is from memory right and the other operand to the ALU is always the D register ok so that is basically done here, then the C1 to C6 will tell you what sort of computation the ALU has to do on the two inputs that it is receiving and that we have seen. Then d1, d2, d3 will tell you the destination and j1, j2 and j3 if they are set if they are non-zero will tell you if there is it is a jump instruction or not. So essentially this is how the entire C instruction is (into)interpreted.


(Refer Slide Time: 6:13)

(Where $a = 0$) Comp mnemonic	c1	c2	c3	c4	c5	c6	(Where $a = 1$) comp mnemonic
	0	1	0	1	0	1	0
	1	1	1	1	1	1	1
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
β	1	1	0	0	0	0	M
D	0	0	1	1	0	1	
βA	1	1	0	0	0	1	M
-D	0	0	1	1	1	1	
βA	1	1	0	0	1	1	M
D+1	0	1	1	1	1	1	
$\beta +1$	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M



Now let us see how the architecture will look like? So we will just have this table this is already the same table that we have seen in the thing, so when a equal to zero please note that the whole thing is ALU thus x, y as inputs right, the x is always D, y will be A when small a is zero will be M when the small a is one ok. So the y will be between A and M that is what we are saying that A equal to zero you see all A's here instead of y and right and while A is one you will see instead of A is you see M ok. So this is the way we interpret and this are the 6 bits.

(Refer Slide Time: 7:15)



d1	d2	d3	Mnemonic	Destination (where to store the computed value)
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory [A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory [A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory [A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory [A] and D register



Module 4.3: The HACK CPU - A Deep Dive: Part 1

PROF. V. KAMAROTTI
IIT Madras



Now the destination please note that if d1, d2, d3 are all zeros then there is no destination then perhaps this is going to be the jump instruction but when you have (d1) d3 bit is 1 that means the destination is memory, d2 bit is 1 then the destination is D register, d1 bit is 1 then the destination is A register. So you see 0 0 1 the destination is just memory 0 1 0 destination is D, 0 1 1 the destination is both the memory and the D register and when I say M it is nothing but which memory location? The address of that memory location is stored in the A register that we have seen in the earlier modules right. So this is how the D register the D bits d1, d2, d3 are basically interpreted.

(Refer Slide Time: 8:15)

j1 (out < 0)	j2 (out = 0)	j3 (out > 0)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If out > 0 jump
0	1	0	JEQ	If out = 0 jump
0	1	1	JGE	If out ≥ 0 jump
1	0	0	JLT	If out < 0 jump
1	0	1	JNE	If out ≠ 0 jump
1	1	0	JLE	If out ≤ 0 jump
1	1	1	JMP	Jump

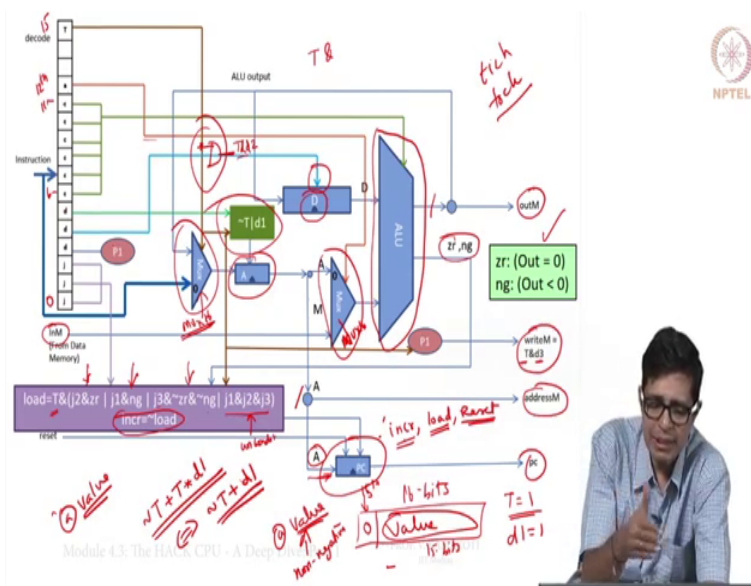
Module 4.3: The HACK CPU - A Deep Dive: Part 1

PROF. V. KAMAROTT
IIT Madras



Similarly the j1 again 0 0 0 means no jump, 1 1 1 means unconditional jump but remaining thing j3 stands for out greater than zero, j2 stand for out equal to zero and j1 stand for out less than zero. So this is basically given here and so this is so when the CPU sees non-zero value on the j bits then it is a jump instruction right so that is how the CPU has to interpret the instruction. So very quickly we have seen how the instructions are going to be organized the each instruction what each bit means.

(Refer Slide Time: 9:04)



Now we will quickly see how the CPU is basically built now this is the building of the CPU nothings carry about it let us go register by register. First thing is let us take the ALU, what are inputs to the ALU? If you go back and see the ALU has the six computation bits which tell you what we need to compute ok. The six computation bits are necessary so that six computation bits basically comes in from the c1, c2 so this is the their instruction this is the 15th bit of the instruction this is the 0th bit of the instruction.

So your 15, 14, 13, 12, 11, 10, 9, 8, 7, 6 where 6 to 6, 7, 8, 9, 10, 11 so 6 to 11th bit is basically given as input to the ALU just follow this green bar green line and which is given as the compute bit to the ALU right. Now what will the ALU basically get us an input always it is going to get x is always D, so this is the D register and x is always D so this gets the D input right and x is the D register and the other input is either the output of the memory or the A register depending upon the a bit, a bit is the 12th bit here so if you say 0 to 12th instruction bit follow this redline as I am marking here, when the a bit is 0 then the output of the A register goes to the ALU.

When the A bit is 1 the in M that is coming from the memory right so if you just see go back here the in M is coming from the data memory that in M is basically given as input so the y input for the ALU is depending upon this small a bit is a 12th bit in your instruction it can be the output of the A register or the output of the data memory right. So this basically comes so this is so this mux as you see is the a 16 bit mux or mux-16 wherein 16 lines come here, 16 lines come from

the data memory, 16 lines come from the A register and this mux depending upon the small a bit will route one of the 16th ALU right and the ALU will output there is an output of the ALU and output of the ALU always goes as out M.

It goes to the memory and its also feed the D register, it should also feed the D register, it should also the output of the memory output of the ALU offcourse the two outputs let us see this the Z r and n g which basically talks about the zero and the not greater than flag right. So that is here but in addition to that the output of the ALU the 16 bit output (can) will be fed to the D register and will also be fed back to the A register right, because the destination for an ALU the destination for an ALU can be the memory, the D register and the A register that we have seen here.

So the destination for the ALU can be the A register as you see here, the D register or the memory. So the output of this binary basically goes output of the ALU goes to the memory also comes back to the D register and also comes back to the A register right. So what should go into the A register? Now this we have just seen the ALU. Now what should go into which one of this register? What should go into the A register? If it is a instruction that means your t bit is zero right then your t bit see this green line if your t bit is zero then the 15 bits that are coming from the instruction should go into the A register if the t bit is zero then the 15 bits so how does your a instruction look zero and then this is the value.

This entire value should go into the A register, so this should basically go into the A register right when your t (instruction) t is one that means it is a c instruction then the a instruction the A register will be update that A register will be updated only if it is a destination right. So that means when will a register be a destination? The A register will be a destination when the d1 bit is set to 1. When the d1 bit is set to 1 you're a register would be a destination. So this is how this will happen. So your A register will be load at when your t bit is zero or when the t bit is zero it will be loaded and it will be loaded when your t bit is zero or your t bit is one and your d1 bit is one right.

So because when the d1 is 1 then A becomes a destination the results should basically become the output of the ALU should be loaded by (())(15:53) so we have the load for the A register will be tilde of t bit that is the (())(16:00) bit that is the 15th bit plus or the t bit is 1 and d1 so when you optimize this logic this is equivalent to saying tilde t plus d1 ok so this is tilde T or d1 right.

So tilde T or d1 would be the condition that the A register will be loaded at the end of this instruction right as a given instruction.

Similarly what will the A register be loaded with? When the t bit is zero the 15 bits from the instruction would be loaded, when the t bit is one and your d1 is 1 then the output of the ALU would be loaded so essentially here again we have a mux-16, so I have a mux-16 here which will take the entire instructions so we take the 16 bits of the instruction anyway because at some value this value should always be non-negative it cannot be a negative value.

So the non-negative value you are going to put it as 15 bits so if I put a zero in front of that 15 bits till it will become a non-negative value the value will not change and it will be a same but would one bit extended to zero. So though when I say though when I say (())(17:37) value that 15 bits have to go to A register that 16th bit the most significant bit will always be zero anyway, so and the a instruction the 16th bit the 15th bit that is 0 to 15 this more significant bit is anyway zero so we can take the entire instruction and put it here.

So and also the output of the ALU is also wroted here and (())(18:08) ok so what will (happen) so now this is given similarly if D for the D register so when will the d be loaded at the end of the instruction when d is a destination and how do you know that d is a destination? D will be a destination when your t bit is one and your d2 is one, it should be a two bit and d2 should be one. So essentially we do a at this point you put an AND gate this is the t bit and this is the (d2) this is d1 this is the blue line is the d2 so this will give me t and d2 and that is what will be the load condition for the D register and what it will be loaded?

The output of the ALU right and when will the memory be loaded? The memory will be loaded when the t bit is one and your d3, so d3 basically says that the (())(19:27) memory so your d3 will be loaded and always the address to which the memory should be loaded is the A register so this address M is basically going to the A register ok right, so this is about the different registers. Now the last thing that we need to see here to understand this entire architecture would be this PC the program counter. So what will the program counter do? When reset the program counter will become zero so the reset is given to the reset of the program counter right and otherwise the program counter will be basically there are other things it can be incremented, it can be loaded or it can be reset.

When the reset comes the reset if the program counter would be reset that is the highest probability otherwise when a jump comes this will be loaded, when a jump comes this gets loaded and if it is not reset and it is not a load then if it is a normal instruction then it is getting incremented. So increment is always not of load, reset is highest in the probability so irrespective of what is increment or load when a reset the program counter will reset for sure. Now if it is not reset that we have seen when the PC has been designed so you can go back to that module and revise it (21:13) right otherwise now your PC has to get loaded.

When it will get loaded? When it is a jump instruction if you are not loading it then it is a non-jump instruction then it automatically gets incremented so your increment will be not (21:35) right, so you now what is load we will see but once you design that load the increment will be not of your. Now what is load? Load is you have a t's it should be c type instruction and right and then please note that if it is a un-conditional jump then j1, j2 and j3 this stands for un-conditional jump.

If j1 and j2 and j3 are true or it is a conditional jump that is it is out greater than zero it is a conditional jump so j3 is set out is greater than zero and so we just say that the jump should take off when out is greater than zero and really out is greater than zero. How will I find out whether out is greater than zero? Your Z r should be non-zero that means output is not zero and output is also not greater than it is output is also greater than zero tilde n g means out is greater than zero, so you just see here tilde is a Z r means it is not zero and out is greater than zero that means out is greater than zero.

So j3 and tilde Z r and tilde n g ok that will so if j3 bit is set that means you go and so that take a jump if your out is greater than zero and so you now test whether out is greater than zero using Z r and n g flag and j3 is set and that means you have to take the jump. Similarly j1 essentially means out is less than zero right if j1 is set here and n g is really then, then we have to take the jump and similarly j2 and Z r, j2 is out is equal to zero and (23:50) zero because this is the way I set, j2 says that you take the branch when output equal to zero and Z r says that output is indeed equal to zero.

So this four conditions capture out equal to zero, out less than zero, out greater than zero and jump, unconditional jump and all this things provided your instruction is a c type instruction, in a

A type instruction when this can be this are only values, this cannot be interpreted as j d bits, this j see a bit interpretation comes only if it is a c type instruction so that is a c type so that is ((0)) (24:31) and if this is true that means the jump has to be so this is the jump instruction the jump has to be taken and the jump has to be taken to which address? That address will always be the A register that is what you say.

So if the load is true then the load value comes again from the A register as you see. So the PC when reset will come to zero when the load is true essentially means is a jump instruction so which instruction should have done? So that is always in the A register so that A will get loaded here and y is it just incremented so incrementing with tilde of your so this how the PC is basically done. So the entire so now we have constructed the ALU, ALU is available for us so you can use it, we have constructed mux 16 so connect the output of a and the instruction into mux 16 you instantiate so in the chips you can instantiate a mux 16 and you can instantiate two mux 16 to one AND gate, one OR gate as you see here I am just marking two mux 16 here and here one AND gate, one OR gate and one A register one D register and ofcourse this ALU and connect this corresponding instruction bits to each one of them and then ofcourse the PC and you have to realize this sort of big logic ok and ofcourse you have done the PC use the same PC, so it is now just that we need to realize one AND gate, one OR gate and ofcourse this slightly larger logic and ofcourse increment will be tilde load.

Once you realize all this logic then use the mux 16 use the two mux 16 you can instantiate and then the already design ALU and already design PC and D and A register and you can assemble the CPU and run the CPU. So this is how the CPU is basically built and one very interesting thing that we need to understand is this the instruction comes from the ROM memory as we have seen here the instruction comes from the ROM memory it feeds into the CPU and this is how the CPU interprets the memory and then it basically executes.

So when that instruction comes here so we just say tick that means the instruction has come here so then what will happen? So based on the instructions the based on this instructions this multiplexers will be start working and the corresponding inputs to the ALU is will be available here the corresponding inputs of ALU. ALU will compute and the corresponding output of the ALU will be available right. So when the instruction comes immediately your mux will start

working mux will be (0)(27:45) mux's that you have seen will get instantiated and so and then it will be ready.

So the output will be calculated that in on a tick when the instruction arrives then this whole thing is a combination logic it will come. Now the results of the ALU will be available out right so result of ALU will be available and if it is a jump should I take the jump, everything would be available, in the clock what will happen? The result of the clock will go tik-tok right in the tok what will happen? The result of the ALU depending on wherever the destination is whether it is D, A or M or whatever or a combination of all these things it will get stored in the corresponding stuff ok (0)(28:38) D register or A register or the memory it will go and stay.

At the same time if it is a jump instruction this whole thing would be ready and when in the tok, tok of the clock TOK of the clock the PC basically will get the new value, if it is a normal thing it will get incremented over the jump then it basically gets the value from the A register if the jump is to be taken, if it is a unconditional jump automatically A will come in, if it is a conditional jump then depending upon the results, status whether it is equal to zero or less than zero or depending on the Z r n g flag the conditional jump will take or not take.

So this is how the whole thing so one thing that you should understand is tik of the clock the instruction flows from the, the instruction pointed to by the PC please note that this PC, PC feeds to the ROM 32K so the PC basically switch back to this right so on the tik of the clock whatever (the) from the ROM the instruction gets into this the input of the CPU and then it does the computation and the tok of the clock the results are stored back in the D A, D A and memory depending upon the destination or in the case of jump the PC gets updated accordingly so that the next tik the next instruction flows from ROM.

So this is how the whole CPU basically works right, so what I will do now is basically I am just going to keep this slide up for a few I will just remove of this comments that I have made here so please note that this is D and D2 that is feeding this D register offcourse this is 0 to 15, so the entire HDL code that we are going to write as a part of your project 5 would be the very-very simple code ok so just a few lines offcourse you will take some amount of time to write this logic for this load ok and so that will be some set of lines but overall the entire project for the CPU just a CPU is just close to some 10 to 15 lines of code.

Just you have to instantiate all this things I mean understood this things, so is a very simple exercise ok so in the module 4.4 we will demonstrate the working of this entire CPU with a live case study, thank you.