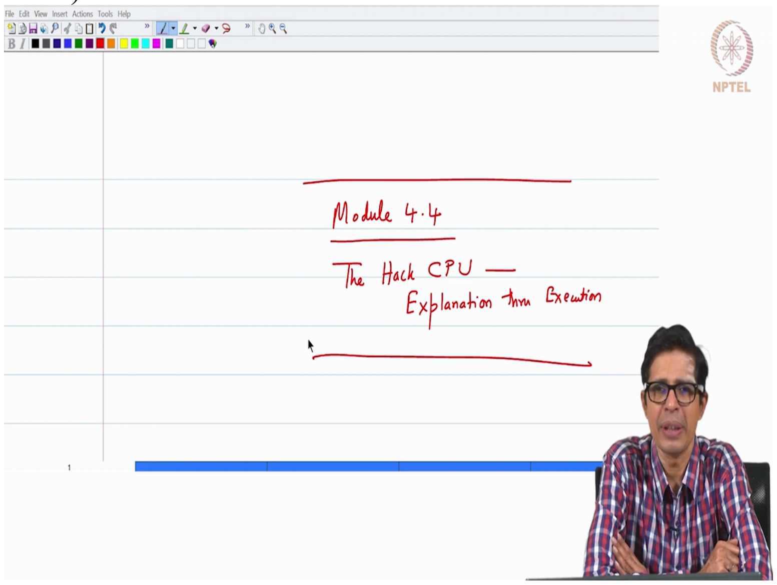
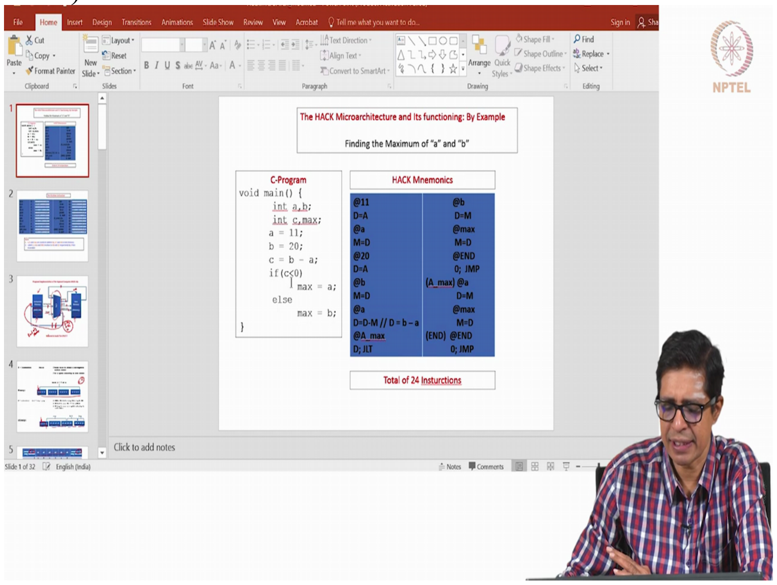
Foundations to Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Module 4.4
The HACK CPU - A Deep Dive- Part 2

(Refer Slide Time: 0:20)


Module 4.4
The Hack CPU — Explanation thru Execution

So welcome to model 4.4. And in this module, we will basically explain the Hack CPU and Jack the explanation will be through executing a program.

(Refer Slide Time: 0:32)


The HACK Microarchitecture and Its functioning: By Example

Finding the Maximum of "a" and "b"

C-Program

```
void main() {
        int a,b;
        int c,max;
       a = 11;
       b = 20;
       c = b - a;
       if(c<0)
            max = a;
        else
            max = b;
}
```

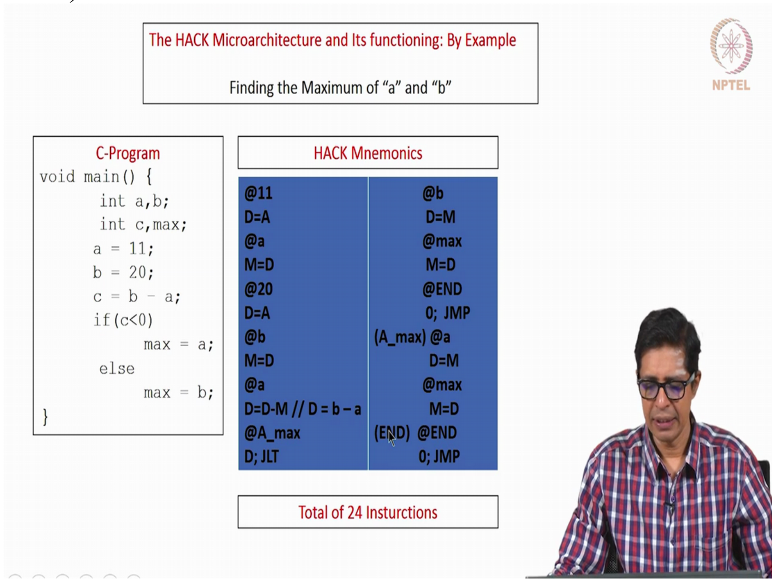
HACK Mnemonics

```
@11                    @b
D=A                    D=M
@a                     @max
M=D                    M=D
@20                    @END
D=A                    0; JMP
@b             (A_max) @a
M=D                    D=M
@a                     @max
D=D-M // D = b – a     M=D
@A_max           (END) @END
D; JLT                 0; JMP
```

Total of 24 Insturctions

So let us see a very simple program which actually finds the maximum of A and B. The way it finds it out is that it subtracts A from B and if the result is less than zero, then the maximum is A, else the maximum is B as you see in this program.

(Refer Slide Time: 0:50)



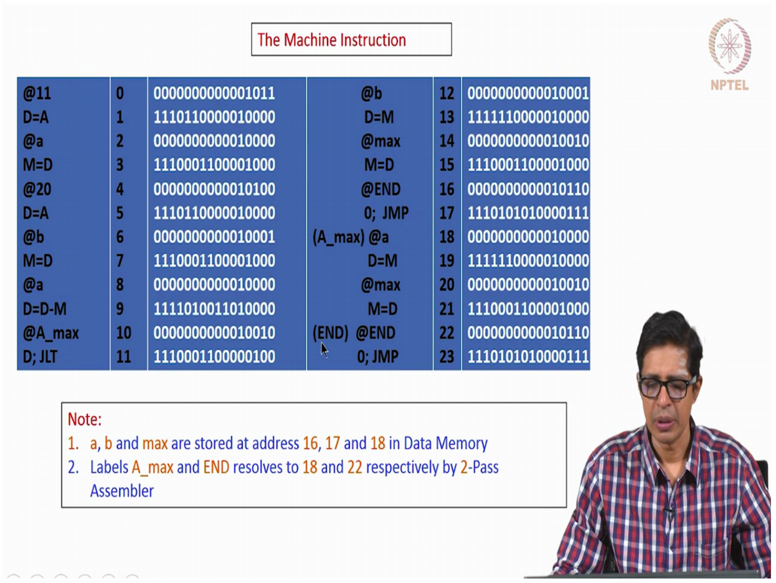
The HACK Microarchitecture and Its functioning: By Example

Finding the Maximum of "a" and "b"

C-Program
```
void main() {
        int a,b;
        int c,max;
       a = 11;
       b = 20;
       c = b - a;
       if(c<0)
              max = a;
        else
              max = b;
}
```

HACK Mnemonics
```
@11                    @b
D=A                    D=M
@a                     @max
M=D                    M=D
@20                    @END
D=A                    0; JMP
@b             (A_max) @a
M=D                    D=M
@a                     @max
D=D-M // D = b – a     M=D
@A_max           (END) @END
D; JLT                 0; JMP
```

Total of 24 Insturctions



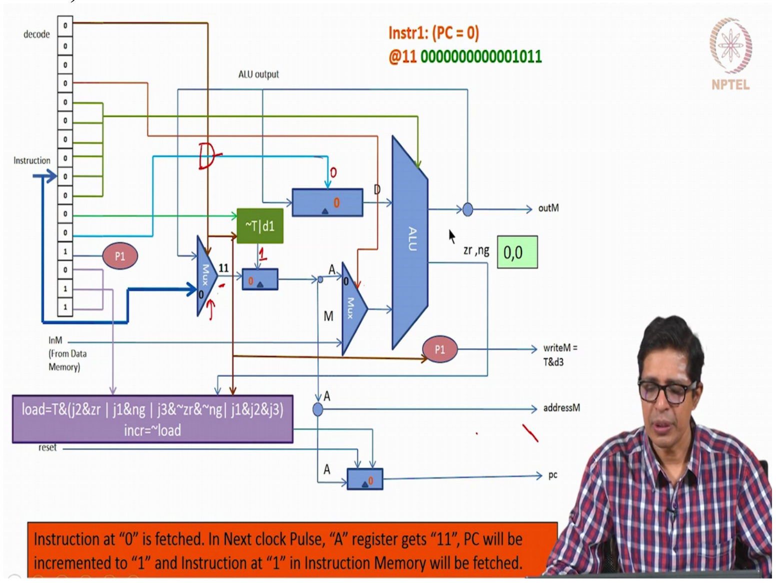
The Machine Instruction

@11	0	0000000000001011	@b	12	0000000000010001
D=A	1	1110110000010000	D=M	13	1111110000010000
@a	2	0000000000010000	@max	14	0000000000010010
M=D	3	1110001100001000	M=D	15	1110001100001000
@20	4	0000000000010100	@END	16	0000000000010110
D=A	5	1110110000010000	0; JMP	17	1110101010000111
@b	6	0000000000010001	(A_max) @a	18	0000000000010000
M=D	7	1110001100001000	D=M	19	1111110000010000
@a	8	0000000000010000	@max	20	0000000000010010
D=D-M	9	1111010011010000	M=D	21	1110001100001000
@A_max	10	0000000000010010	(END) @END	22	0000000000010110
D; JLT	11	1110001100000100	0; JMP	23	1110101010000111

Note:
1. a, b and max are stored at address 16, 17 and 18 in Data Memory
2. Labels A_max and END resolves to 18 and 22 respectively by 2-Pass Assembler

Now the corresponding code for this hack code is this and the way this gets, basically the 24 instructions and the way this gets Jack stored in the memory, so the memory is 16 bits, so this is our 16-bit instruction. So from 0 to 24, 0 to 23, the 24 instructions get stored. And let us also assume that A, B and Max are stored in the data memory at 16, 17 and 18 locations. And in instruction memory, the instructions are stored from 0 to 23.

(Refer Slide Time: 1:32)



Instruction at "0" is fetched. In Next clock Pulse, "A" register gets "11", PC will be incremented to "1" and Instruction at "1" in Instruction Memory will be fetched.

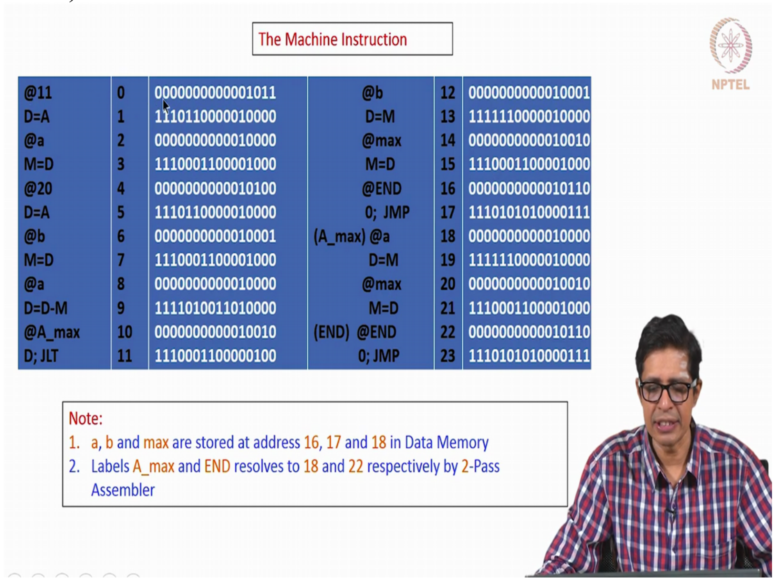
Now this is the setup Jack. Now this is the architecture that we have explained in the previous model. Now this architecture basically has a D register, A register, PC and memory. So the context of a architecture are the storage elements. The state of an architecture is all the storage elements. We have seen this in the earlier class, we have explained this in the earlier class. Now what is it that we are storing between instructions? We are storing the A register, the D register, the program counter and whatever is there in the memory.

The memory is basically given by this outM Jack and controlled by this writeM. So if the writeM is one, then whatever goes on the outer will be that data will be stored in the address given by address. Always the address of the memory will be stored in the A register. So these are the things that we have seen okay. So the context of the process or the state of the microarchitecture is defined by this D, A, PC and the memory. So first when a reset comes, D will become zero, A will become zero, PC will become 0. So this is the state of the architecture before we send a reset.

Now when a reset comes, please note that Jack Jack this became zero, Jack Jack the D register also became zero, the PC is also reset to 0. The moment you remove reset, what will happen? So since the PC is pointing to 0, the instruction in the 0th address in the instruction memory will be fetched. So on a tick of the clock, the instruction will be fetched. So these, this is, these are the instructions, 16 bits of the instruction and then that instruction will be processed and in the tock of the clock, the clock goes tick-tock, right?

So the tick, the instruction will be fetched from the memory and in the tock of the clock, T-O-C-K, the result will be stored in the destination. So this is how the whole thing works. Now initially, this is a system reset. So we give a reset pulse and everything becomes zero. Now let us see what the first instruction is. Since the PC was zero, in the tick of the clock, the first instruction is fetched and what is the first instruction?

(Refer Slide Time: 4:12)

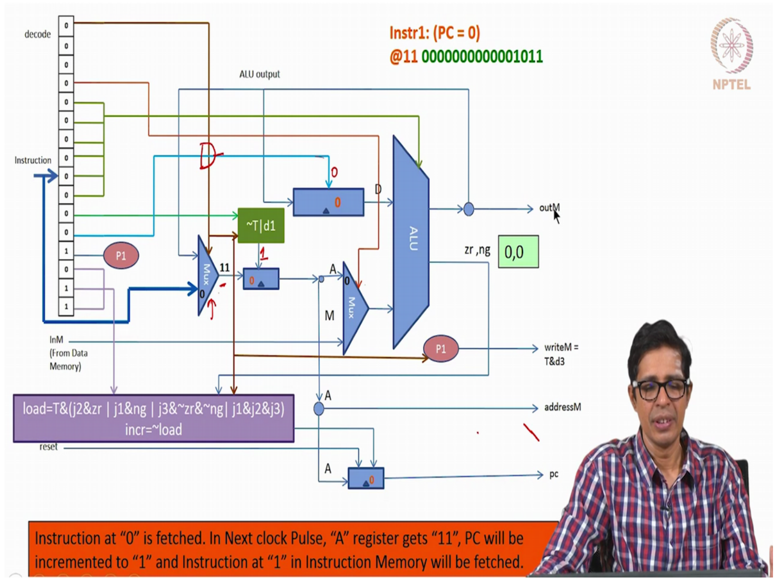


The Machine Instruction

@11	0	0000000000001011	@b	12	0000000000010001
D=A	1	1110110000010000	D=M	13	1111110000010000
@a	2	0000000000010000	@max	14	0000000000010010
M=D	3	1110001100001000	M=D	15	1110001100001000
@20	4	0000000000010100	@END	16	0000000000010110
D=A	5	1110110000010000	0; JMP	17	1110101010000111
@b	6	0000000000010001	(A_max) @a	18	0000000000010000
M=D	7	1110001100001000	D=M	19	1111110000010000
@a	8	0000000000010000	@max	20	0000000000010010
D=D-M	9	1111010011010000	M=D	21	1110001100001000
@A_max	10	0000000000010010	(END) @END	22	0000000000010110
D; JLT	11	1110001100000100	0; JMP	23	1110101010000111

Note:

1. a, b and max are stored at address 16, 17 and 18 in Data Memory
2. Labels A_max and END resolves to 18 and 22 respectively by 2-Pass Assembler


Instr1: (PC = 0)
@11 0000000000001011
decode
Instruction
ALU output
~T|d1
ALU
Mux
Mux
P1
P1
D
A
M
zr ,ng
0,0
outM
writeM = T&d3
addressM
pc
InM (From Data Memory)
load=T&(j2&zr | j1&ng | j3&~zr&~ng| j1&j2&j3)
incr=~load
reset
NPTEL
Instruction at "0" is fetched. In Next clock Pulse, "A" register gets "11", PC will be incremented to "1" and Instruction at "1" in Instruction Memory will be fetched.

We go here and see. The first instruction is at 11. That is 00001011. Now what is this instruction supposed to do? It is to load the A register with 11 and it should not do anything with anything else. So what are the other things? It should load the A register with 11, the D register should be untouched, memory should not be written and at the end of the instruction, that is on the tock your PC should be incremented to one. So two of the storage elements in this context will change. One is the A register which will now get upgraded to 11 and then the PC which needs to get incremented to one.

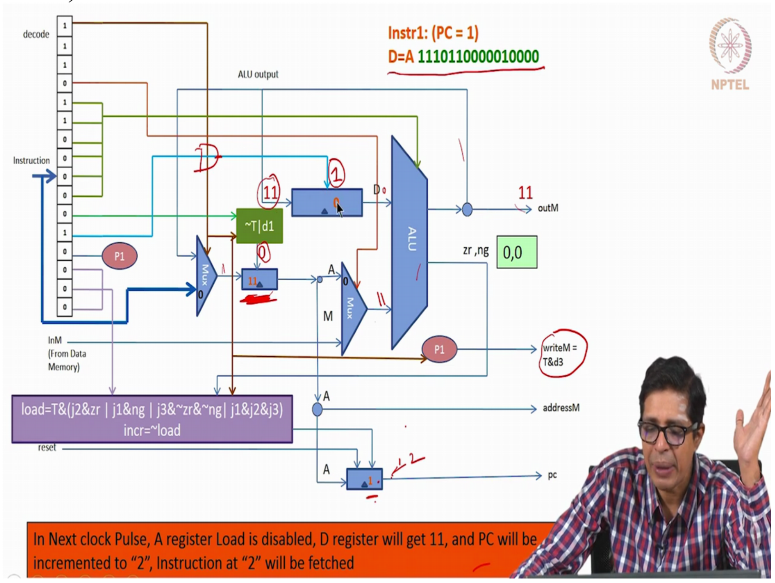
The D register will be unchanged and the very also will be unchanged. Now that we can see. Now what will happen? Now what is the so when the A register has to change on the tock of the clock, T-O-C-K of the clock, the load for the A register should be 1 and that essentially happens here. So this is the type bit. As you remember, this is the type of bit which distinguishes between an A instruction and a C instruction. This is an A instruction and now that, this is 1 because this is the T bit, the type bit and tilda T is one. So A load is one while D load is zero.

So nothing will happen to D. And the increment, load is zero for the PC, so the increment is one for the PC. And writeM is 0 to the memory, so nothing will get written into the memory. Right? writeM is zero for the memory. So nothing gets written into the memory. Right? So so this one will arrive here and also note that this mul particular multiplexer, it is this is to decide between the output of the ALU and the 15 bits that are coming from the instruction. So that will now decide since this is zero, this will decide that whatever is coming in the instruction has to go and wait at the input of the A register.

So the moment the tick comes, the A register's load has become 1, the value 11 is now available at the, just the edge of the A register. D register is untouched, writeM is zero. T is zero, so this is zero. Nothing will write into the memory. Increment is one for the PC and so in the tock, what will happen? Since A register's load is 1, 11 will get into the A register and since increment to the PC is 1, PC will get incremented to 1.

Since the load for the D register is zero, nothing will happen to the D register. And since writeM is zero, nothing will happen to the memory.

(Refer Slide Time: 7:24)



In Next clock Pulse, A register Load is disabled, D register will get 11, and PC will be incremented to "2", Instruction at "2" will be fetched

So at the end of this, A register will get 11 and your PC will now become 1. Now this PC is now again fed back into your memory. So now in the next tick, what will happen? The next instruction will be fetched from the memory. And what is the next instruction? D equal to A. So whatever is there on the A register should essentially move to the D register. That is what that needs to be done here and this is a C type instruction. Okay.

Now that means now note that let us go. So so since D needs to be updated, in the next tock… Tick, this instruction has come. In the tock, what should happen? Your D should get 11 and your

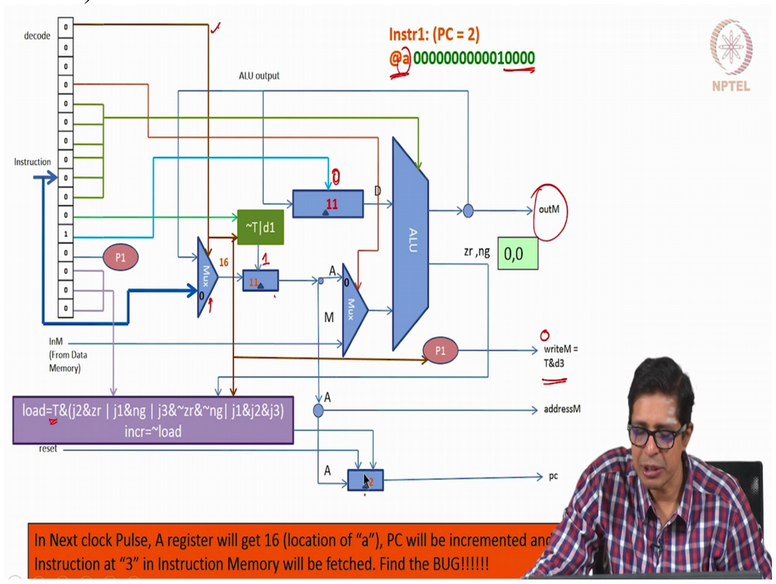
PC should now increment to 2. Nothing should happen to memory and nothing should happen to the A register. So the D register and the PC alone has to change here. Now you see, the bit here is 1. The load to the D register is one. So the moment I get this instruction, immediately you see that the load to this instruction is one and then Jack the input to the ALU is 110000 and the A bit is also zero.

Since the A bit is zero, then whatever is there in the A register will go into the ALU. If the Abit as you see here, if the a bit is one then whatever is there on the memory should go. So the small A bit decides which is the operand to the ALU. This we have explained. So now to the ALU, 11 has come in and the A is zero and then here, D bit is zero. Anyway, when we give 110000 with A bit is zero, the ALU will just transfer this A good output. So 11 has come out of this and then it is fed back and it is available in the edge of the D register.

It has not entered the D register. So what has happened? The moment the instruction entered this point, please note that the value in this D register has gone through this Mux and come out through the ALU and then it is now fed back and it is available in the mouth of the D register. And the D register's load is one because the destination bits D1, D2, D3 right, the destination bit has set to 1 and PC's increment is also one, load is zero. Now and importantly also note that the A register's load is zero. So nothing will happen to A register.

The writeM bit is also zero when D3 is zero as you see here. D1, D2, D3 right? The D3 is zero here. So since the D3 is zero, writeM is zero, so nothing will happen to the memory. So when that tock of the clock comes, the D register will now get updated by 11 and your PC now will increment to 2. Okay? Right? So that is what needs to happen. Nothing will happen to the memory, nothing will happen to the A register. That context will happen. So this is the status.

(Refer Slide Time: 11:13)

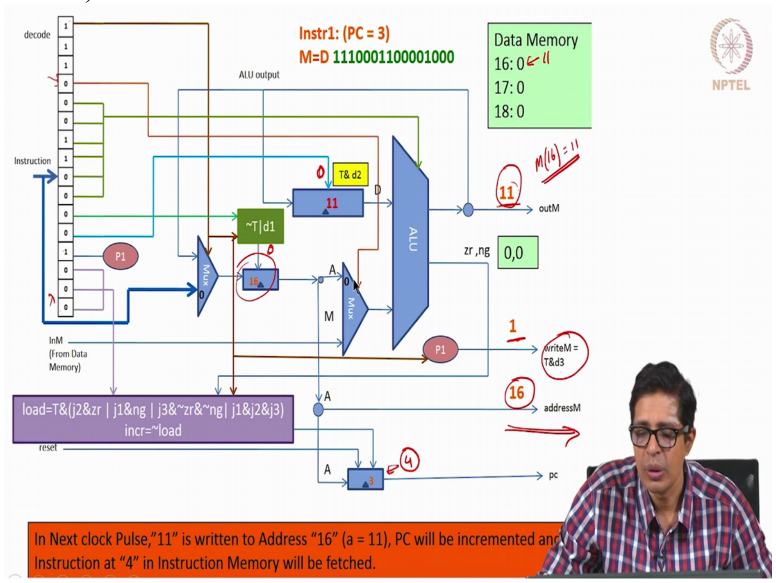

Instr1: (PC = 2)
@a 0000000000010000
decode
Instruction
ALU output
~T|d1
Mux
16
A
M
D
ALU
outM
zr ,ng
0,0
P1
writeM = T&d3
InM (From Data Memory)
load=T&(j2&zr | j1&ng | j3&~zr&~ng| j1&j2&j3)
incr=~load
reset
addressM
pc
NPTEL
In Next clock Pulse, A register will get 16 (location of "a"), PC will be incremented and Instruction at "3" in Instruction Memory will be fetched. Find the BUG!!!!!!

Now D register has become 11, A register is 11 already, PC is now 2. Now this PC is fed back into the ROM and now what we are going to get is the 2nd instruction the 3rd instruction which is PC equal to 2, 012 that 3rd instruction from the ROM.

And what is it? It is again an at A. Now what is A? The assembler has said that A is 16, 1000. So since it is an A type instruction, this mux will take it directly from the instruction. So 16 will be available in the mouth of the A register. And what is this instruction supposed to do? It has to now update the D register by 16. It should not touch the D register, it should not touch the memory and it should increment the PC. Already, increment to the PC is one. Because load is zero, so the increment to the PC is one.

And Jack the 16 is available here and at the at the input of the A register and there is going to be Jack and also note that the load better to the capital a register is also one. So load bit to the one and the D register's load is zero. Right? So and since writeM bit is also zero, so nothing will happen to the memory. So when the tock of the clock comes, this 16 basically gets loaded into the A register, nothing happens to the D register, this PC gets incremented to 3 and nothing gets done into the memory.

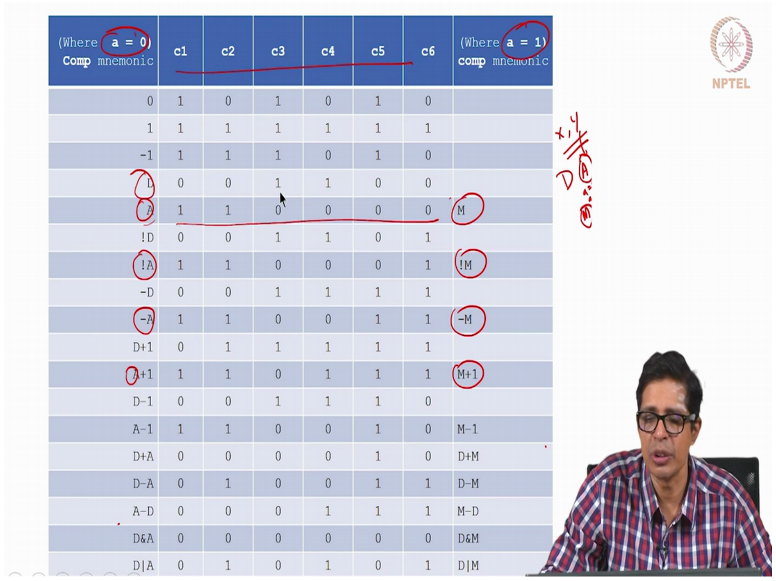
(Refer Slide Time: 12:55)



Instr1: (PC = 3)
M=D 1110001100001000

Data Memory
16: 0 ← 11
17: 0
18: 0

In Next clock Pulse,"11" is written to Address "16" (a = 11), PC will be incremented and Instruction at "4" in Instruction Memory will be fetched.

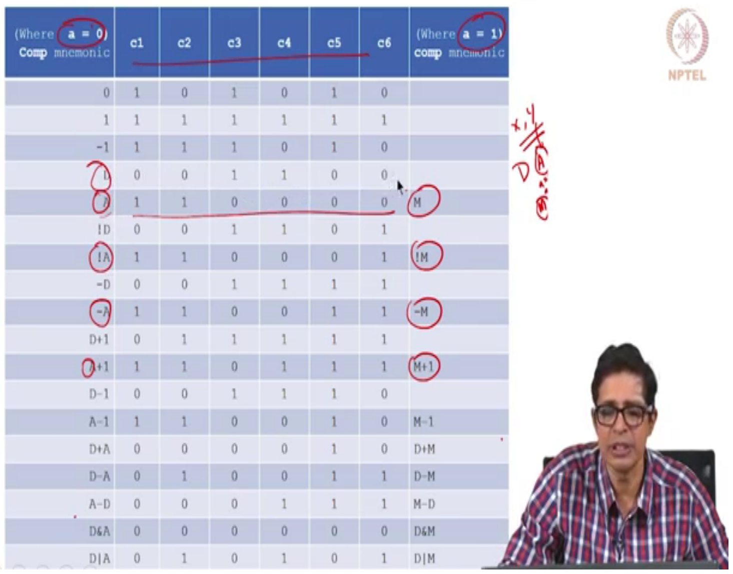


(Where a = 0) Comp mnemonic	c1	c2	c3	c4	c5	c6	(Where a = 1) comp mnemonic
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	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
D|A	0	1	0	1	0	1	D|M

So now we have 11, 16 and 3. Now this 3 goes in. What is this 3? What is there in the 3rd location in the ROM? M equal to D. What should M do? what should this M equal to D do? What is M? So in the address given in the A register, go and write the content of D, that is what this means. In the address given in the A register, that is what we specify by M. Go and write the content of D. So what is there in the A register? Before this instruction, 16 was there. This 16 show 6 show into the memory address 16, the content of D, what is it? 11. So in the memory, in the data memory where 16 currently zero is there, it should now be 11. So at the end of this instruction, the data memory location 16 should get the value 11.

And nothing else should change. Your A register should not change, your D register should not change, your memory 16th location in the memory should get 11 and your PC should increment to 4. Now note that when given this instruction, you can see that your load of the A is zero, load of the D is zero and Jack but writeM to the memory is one. Because your Tbit is one and your destination 3 is one. So T and D3 is one. Right? And your addressM is basically taken from the A register, that is 16 right? And what will the ALU do? 001100, this basically transfers the value D. The A bit is zero Jack so the value will be, the output will be D.

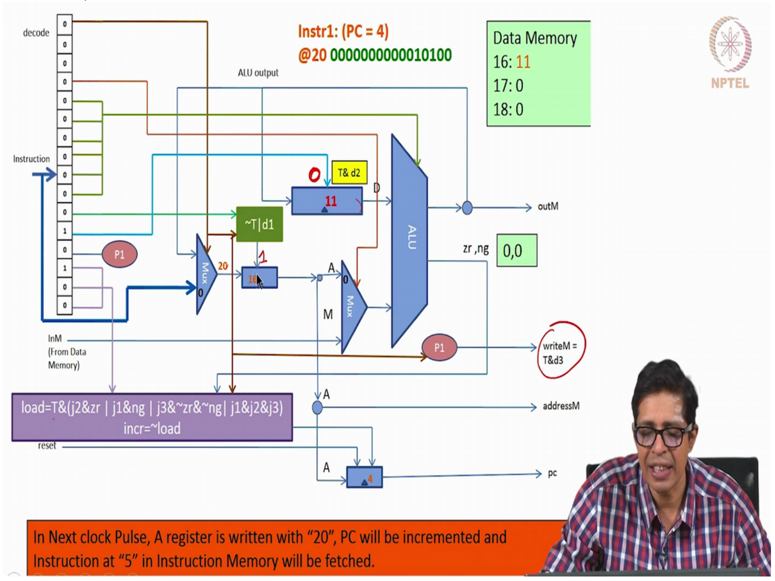
(Refer Slide Time: 14:59)



(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	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+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
D|A	0	1	0	1	0	1	D|M

So this you can see from this table. So I just want you to go back to this table and see. The output will be D. 001100 with A, small A bit zero, the output will be D. So the output is D. So 11 will be available here. Now when the tock of the clock comes, your PC gets incremented to 4 and your 16th location given by addressM gets the value 11 because your writeM is one. And since the load bit for load for the D register and A register are 0, they remain unchanged. Right? So this is what happens here.

(Refer Slide Time: 16:00)

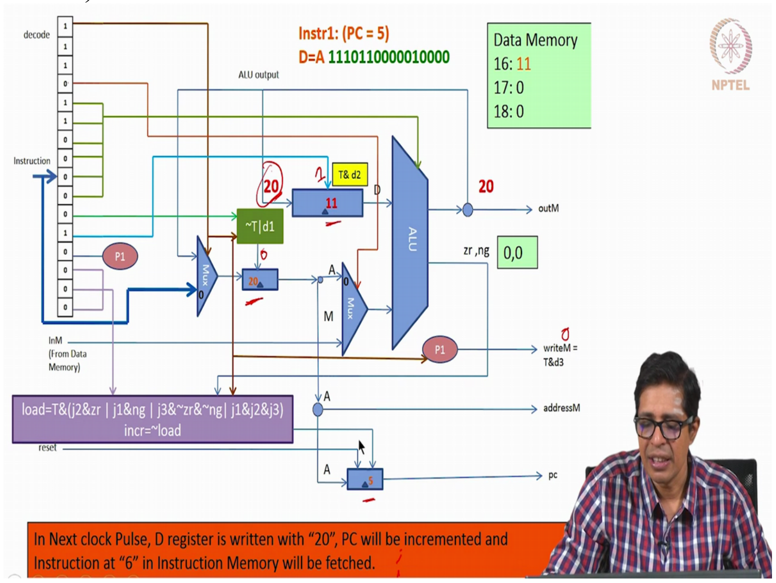


In Next clock Pulse, A register is written with "20", PC will be incremented and Instruction at "5" in Instruction Memory will be fetched.

So at the end of the story, please note that the 16th location in the memory got 11 and your PC got incremented to 4 while your A register and your D register remain same. Now this 4 goes, again the tick of the clock comes, so the instruction in the 4th location in the instruction memory is fetched which is nothing but at 20. So this is an A instruction. Now note that the D register is zero, load load for the D register is zero, writeM is zero. So nothing will happen to memory, nothing will happen to D register. What should happen? The A register should get the value 20 and note that increment to the PC is one.

So at the talk of the clock, what will happen? A register will get 20 while PC will get incremented to, nothing will happen to memory, nothing will happen to the D register. And that is what you see here. And again this mux, please note that because this is a A type instruction, you get a zero here. This Mux will push in whatever is there in the instruction to the A register so that will push in this 20 here. Right? So A register gets 20.

(Refer Slide Time: 17:18)



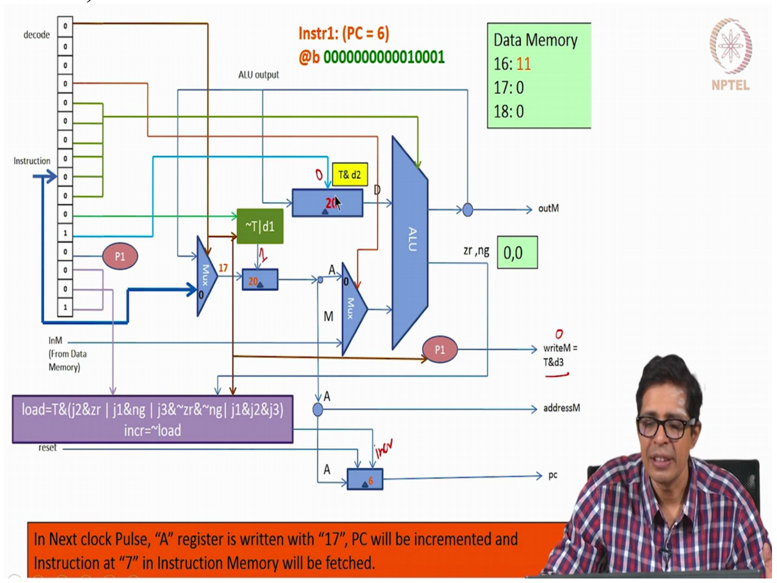
In Next clock Pulse, D register is written with "20", PC will be incremented and Instruction at "6" in Instruction Memory will be fetched.

So A register got 20, this got 11 and then this is 5. So the 5th instruction is now being fetched. On the tick of the clock again, you get the 5th instruction which is nothing but D equal to A. So this 11 now needs to become 20. Now how does it happen? Again A register, the input to the A register, the load is zero here while please note that the load here which is T and D2, this is one right? This is one for the D register because D has to get updated. So D has to now become 20.

So the load for the D register is 1 here, PC is just incrementing, this is zero, writeM to the memory is zero. So nothing will happen to the memory. PC will get incremented but what has happened here? This A, this small A bit what you see here is 0 Jack Jack to this Mux. So that means whatever is the content of A will be pushed into the ALU. So this is 20 and then the type of competition is 110000 to the ALU. And what it will do? It is just if you just see back the table, the output will be A, that is 20 reaches here and this 20 is fed back.

So on the tick of the clock when this instruction comes, 20 is available in the mouth of the Jack D register and its load is 1, A register is zero, write to the memory is zero and this PC is in the increment mode. So at the tock of the clock, 20 will now move into the D register and this PC will increment to 6. Right? And that is what is happening. So PC has incremented to 6. So PC will now be fetched. PC now will be fed back to the ROM.

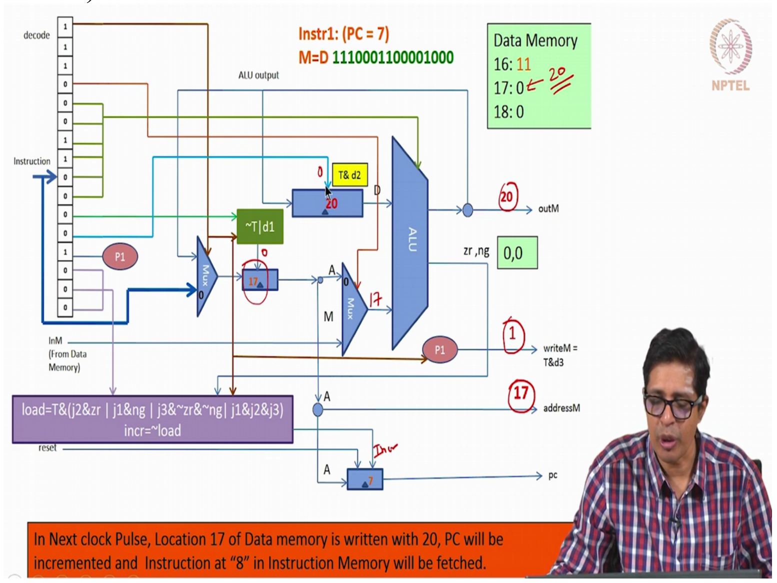
(Refer Slide Time: 19:08)



In Next clock Pulse, "A" register is written with "17", PC will be incremented and Instruction at "7" in Instruction Memory will be fetched.

In the tick of the clock again, you get the next instruction which is at B. At B is 17. So again, this is an A type instruction. So 17 comes 17 from the instruction is routed by this Mux and it is available in the mouth of the A register and the load to the A is 1, load to D is zero, write to the memory is zero, PC is an increment mode. So the in the tock of the clock, 17 will be loaded into the A register and PC will be incremented to 7. Since D register's load is zero, write to the memory is zero, nothing will happen to the memory or the D register.

(Refer Slide Time: 19:59)

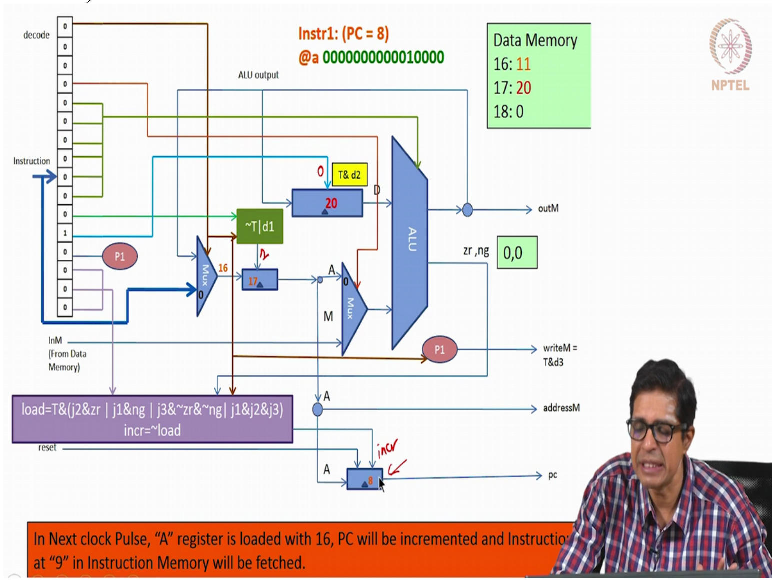


In Next clock Pulse, Location 17 of Data memory is written with 20, PC will be incremented and Instruction at "8" in Instruction Memory will be fetched.

Now this 7 is fed back to the ROM and the at the tick of the next clock, you get M equal to D. What will M equal to D mean? That in the location 17 that is given in the A register, store the content of the D register that is 20. So in the data memory 17, the value 20 will get stored. The value 20 needs to get stored in data location 17. So that is what will happen as a part of this instruction. Now again you see, A register's load is zero, D register's load is zero, writeM is one. Address bit is 17 and you have given 001100 to the ALU and what will the A and the A bit, small A bit is zero.

Now if you see from the table, the output will be D. So 20 comes here. Now this 20, now since the writeM is one, in the tock of the clock, the location 17 will be written with the value 20 and the PC will get incremented to 8. Okay. And nothing will happen to the A register or the D register because the load bit for them is zero.

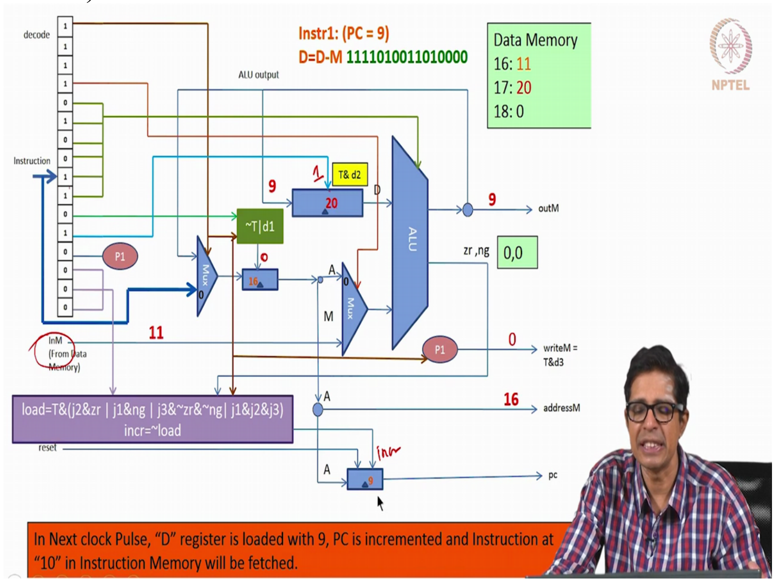
(Refer Slide Time: 21:26)



In Next clock Pulse, "A" register is loaded with 16, PC will be incremented and Instructio at "9" in Instruction Memory will be fetched.

Now the next in now this 8 is being fed back into the ROM. Again in the tick of the clock, you get PC you get this at A. at A is 60. So the small A is mapped onto 16. So this is 16. Now 16 should get, what will this instruction do? Again it will make that capital A register 16. So the 17t should be now replaced by 16 at the end of this instruction. So again you see that the Jack jack load bit for the A register is one, load bit for the D register is zero, writeM to the memory is zero Jack and PC is in increment mode. At the tock of the clock, your A register basically gets 16 and your PC will be incremented to 9.

(Refer Slide Time: 22:19)



In Next clock Pulse, "D" register is loaded with 9, PC is incremented and Instruction at "10" in Instruction Memory will be fetched.

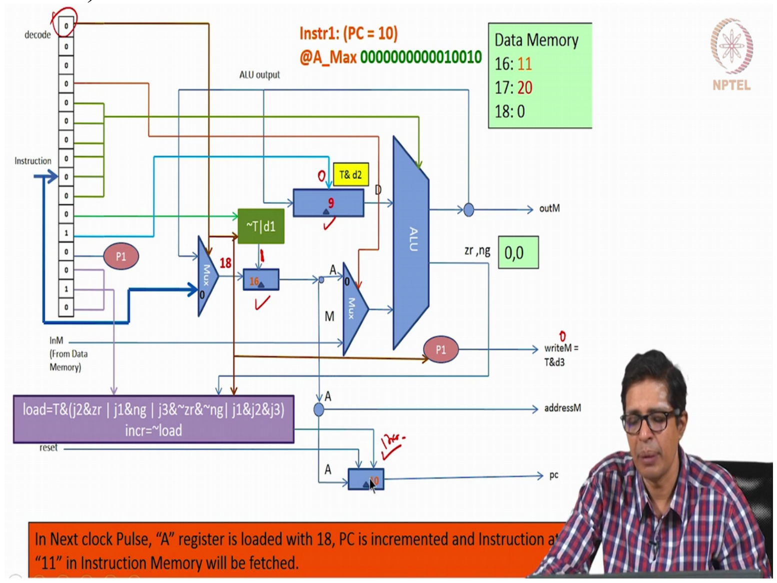
Now PC got incremented to 9 and your A register did get updated to 16. Now this PC is again fed again and the 9th instruction is fetched. So this is a nice instruction, D equal to D minus M. So what you should do? Your 20 is equal to sorry, your D is equal to 20 minus whatever is there in the memory, which memory? Whatever is given by the A register. So the D register should get its content 20 subtracted by the content of the A register, that is the 16th location, content of the 16th location.

Content of the 16th location is 11. So 20 minus 11 should become 9. So D should get the value 9. That is what should happen in this instruction. And the PC should become incremented to 10. So 2 things should happen at the tock of this, at the end of this instruction your D register should get 9 and your PC should become 10, incremented. A register should not change, nothing in the memory should change. First note that writeM is 0 but the address is 16.

Since the address is 16, that will be fed to the data memory and in the 16th location, what is there? 11 is there. That 11 will be available at inM as you see here. So 11 is available here. Now you are A, small A bit determines what goes to the Y input of the ALU and since that is 1, that 11 value goes to this ALU. Here 20 is always fed here and the next one is 010011 and that is the compute. So the answer, you can go back to the table and you can find that. D minus M is what it will compute.

So 20 minus 11 is computed, 9 is available here and it is better back and it is available in the mouth of the D register. So in the tock of the clock, nothing will happen to memory because writeM is zero, nothing will happen to the A register because it is load is zero but the D register since its load is one, T and D2, the T bit and the destination 2 bit, that is also one, the end of that is one, so the value 9 will get uploaded into the D register and your PC will get incremented to 10.

(Refer Slide Time: 25:00)

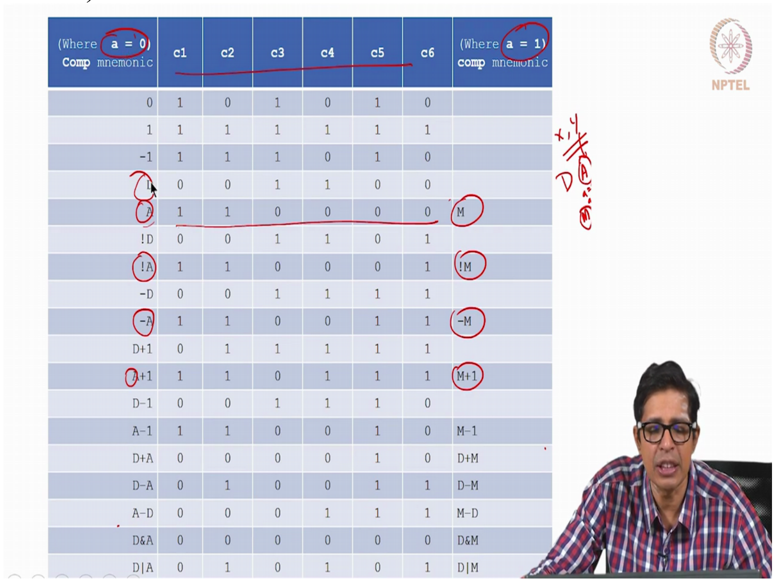


In Next clock Pulse, "A" register is loaded with 18, PC is incremented and Instruction at "11" in Instruction Memory will be fetched.

Again this is another A instruction at A Max. A max is again 10010 which is nothing but 16 plus 2 18. But this is in the instruction memory. Please do not confuse it with the data memory. So at A max is 18. So 18 should get loaded into the D register. Again now you note that at the end of this instruction, so at the tick when this instruction comes, 18 Jack this zero this is a zero bit. So this particular Mux routes 18 to the mouth of the A register, writeM is zero, so nothing happens to memory.

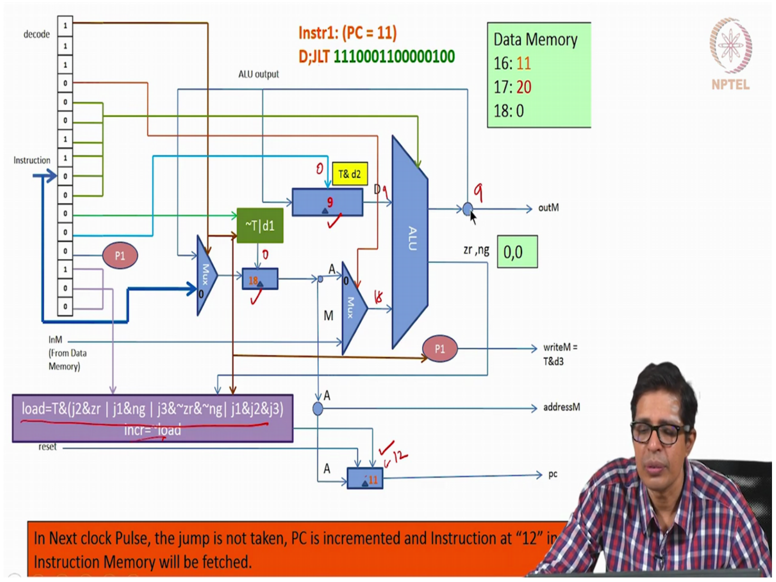
The load bit for the D register is zero, so nothing happens to the D register and your PC is increment Jack Jack increment input of the PC is one. So it should get incremented to 11. So at the tock of this clock, again your Jack PC will so your A register will get 18 and your PC will become 11.

(Refer Slide Time: 26:04)



(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	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+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
D|A	0	1	0	1	0	1	D|M

x, y
D
A
M
NPTEL



In Next clock Pulse, the jump is not taken, PC is incremented and Instruction at "12" in Instruction Memory will be fetched.

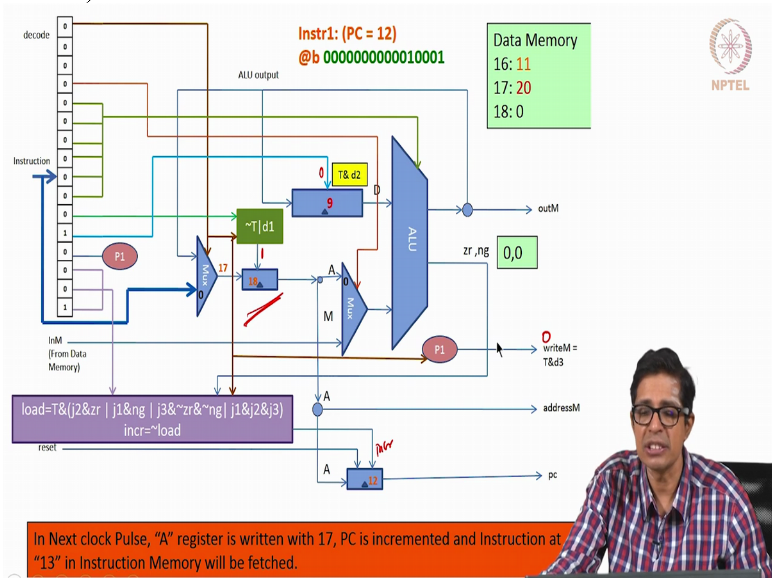
That is what has happened. Your A register has become 18 and your PC has become 11. Now this PC is now fed back into this. Now this is one instruction, D:JLT. If your D is less than zero, then jump right? If your D is less than zero, then jump. Jump where? Jump to the location 18. Right? If your D is less than zero, jump to the location 18. Right? Otherwise what you do? If your D is not less than zero, so just go to PC equal to 12. So what happens here? So let us see this. This is, this is a new instruction because the jump instruction.

Now your A load of A is zero, load of D is zero, D register is zero Jack and the what is the ALU supposed to compute? It is supposed to compute D and for that what you give? 001100, that is fed here, right? And if you just give 001100, Jack and the A bit is zero then the output of the memory is D. That we see again. I just want you to go back, 001100 and the small A bit is zero, please note that the output of the ALU is capital D. Right? So since 01100 is given to the ALU and so the output of this is 9.

Since it is 9, it is neither 0 and it is also Jack Jack the output is Jack Jack so NG is also zero, right? So because the output is greater than zero, NG says that NG becomes one if the output is not greater than zero. So output is not zero and it is also greater than zero. So so the ZR and NG Flags are 00. Now if you calculate the load, since is though the T bit is one, your ZR and NG flag, both are 0s right? Both are 0s and this is a conditional jump, that is J1 is zero, J1 is one, you will see that the load is zero because surely, since the D output is greater than zero, so this jumper will not happen. Correct?

So your load is zero, so increment is one. So the increment is one and so this will just do, it will just increment the PC. Nothing will happen to memory, nothing will happen to the Jack Jack A register or D register. But what, only the PC will get updated. So in any jump instruction, the PCS and surely has to get updated and the PC essentially gets updated here. If the jump is taken, then the PC should get the new value. If the jump is not taken, then it just it gets incremented. In this case, that this is a conditional jump but the jump is not taken. So PC just gets incremented to12.

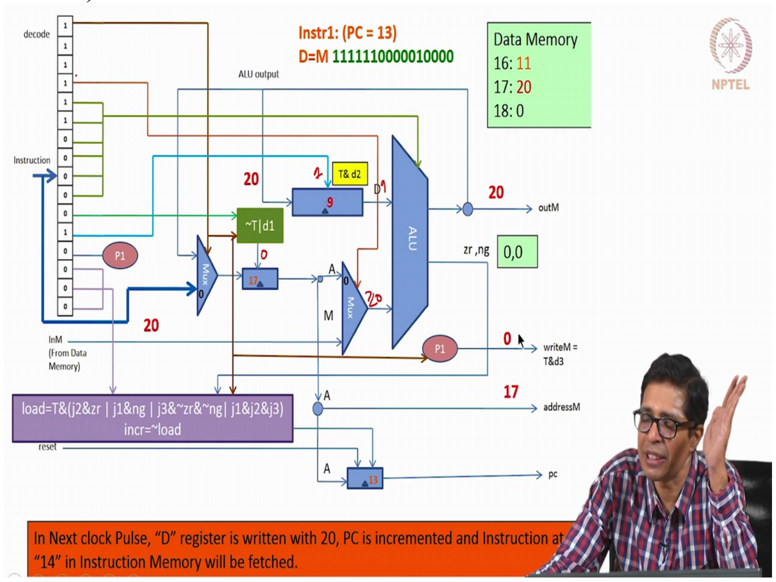
(Refer Slide Time: 29:36)



In Next clock Pulse, "A" register is written with 17, PC is incremented and Instruction at "13" in Instruction Memory will be fetched.

Now the next instruction is again at B. So at B is 17. So this is an A instruction. So this Mux basically routes 17 which is coming from the instruction to the mouth of the A register and the load to the A register is 1, load to the D register is zero, writeM is zero and the PCs in increment more. So at the tock of the clock, 17 gets loaded into this A register and your PC gets incremented to 13. The memory and the D registers are untouched.

(Refer Slide Time: 30:10)



In Next clock Pulse, "D" register is written with 20, PC is incremented and Instruction at "14" in Instruction Memory will be fetched.

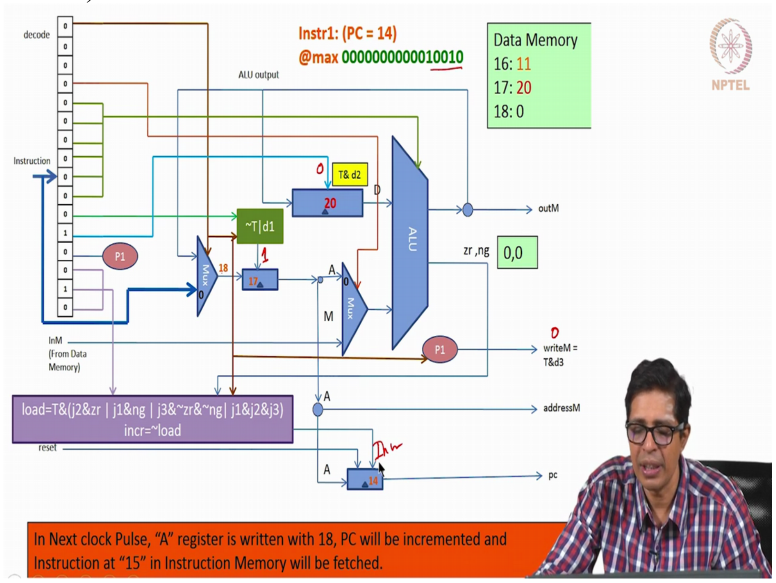
Now what is PC equal to 13? Now D should become M. The 13th instruction when it comes as a tick, what it says? D equal to M. So what is M? M is what is there in the 17th location in your data memory. What is there in the 17th location in the data memory? 20 is there in the 17th location in the data memory. And that should basically get updated into the D register. Okay? 20 should get updated into the D register. So what is there in the 17th location in the memory, that is the A register, content of the A register.

17th location in the memory should be loaded into the D register. So so essentially at the end of this instruction, the D register should get the value 20. So that is what would happen. So how does it happen? First note that the writeM is 0. So nothing happens to the memory. The addressM is 17 which comes from the A register. Jack Your addressM is 17. So this zero and 17 fed to the data memory essentially will read the content of this 17th location, so 20 comes in. And your A bit is one, small A bit is one, so this Mux distinguishes between you know routes between A and this M, so this will route 20 here.

And you have given 110000, so if you give 11000, whatever is there in this M, goes out, so 20 will be available at the outM and that will be fed back. So after the tick, when the instruction comes, the value 20 essentially comes to the mouth of the D register. A register is zero, the D register's load is 1. So T is one and D2, the destination to is also one. T and D2 is one. So at the tock of the clock, your D register will get 20, A register will be untouched because it is zero, it is load is zero. Your memory will be untouched because writeM is zero.

You have read from the memory, you have you are not writing into the memory. And your PC essentially will get incremented to 14. So that is what is going to happen.

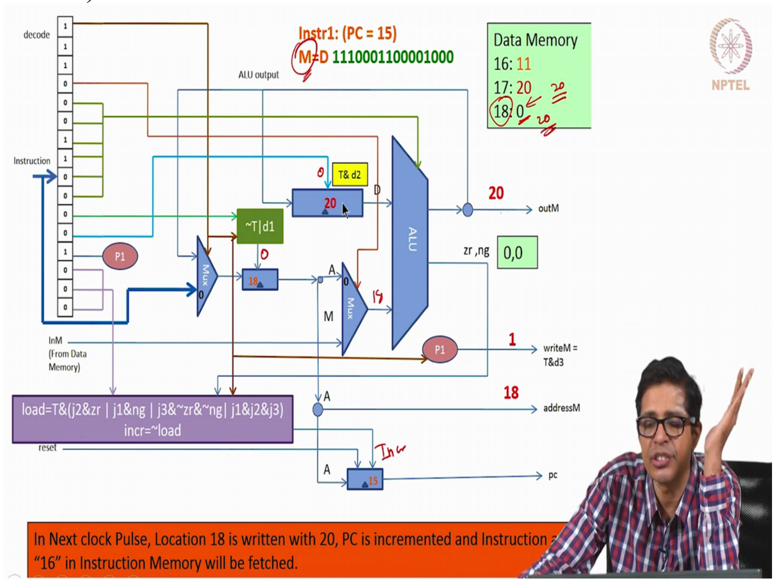
(Refer Slide Time: 32:47)



In Next clock Pulse, "A" register is written with 18, PC will be incremented and Instruction at "15" in Instruction Memory will be fetched.

Yes the PC has incremented to 14, your D register, that gets 20. Now again at Max and we assume that the Max is in the 18th location. So again this is an A in A type instruction, so this Mux will route 18 which is coming in the instruction itself. As you see here, this is the instruction itself into the mouth of the A register and the load for the A register is one, load for the D register is zero, load to the memory is zero and increment Jack for the PC is in the increment mode. So at the tock of the clock, 18 gets loaded to the A register while the PC gets incremented to 15.

(Refer Slide Time: 33:31)

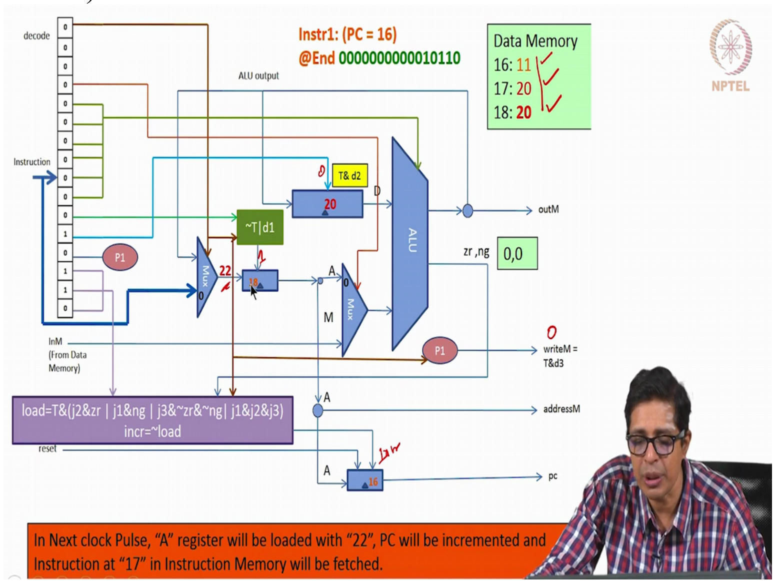


In Next clock Pulse, Location 18 is written with 20, PC is incremented and Instruction at "16" in Instruction Memory will be fetched.

Now M equal to D. So the 15th instruction is M is equal today. So what M equal to D means? The content of the D register namely 20 needs to be returned into the address 18. M means memory. Which address in the memory? Whatever is there in the capital A register, 18. So now you see, so this essentially happens here. So the A register's load is zero, your D registers load is zero, so nothing happens to your A or D register. Your writeM is one, so something happens, something is written into the memory.

What is written into the memory? To which address it is written to the memory? The 18th address. It is written into the address 18 and what is written? The output of the ALU is written and the PC will get incremented to 16. So that is what needs to we have to do. And now this is zeroand we have given 001100. So if I give 001100 to the ALU and the A bit is also zero, the output will be D. So 20 now goes to outM. So at the tock of the clock, nothing happens to the A register or the D register. But the memory, since writeM is one, the 18th location of the memory gets written with the value 20 and your PC gets incremented to 16.

(Refer Slide Time: 35:04)


Instr1: (PC = 16)
@End 0000000000010110

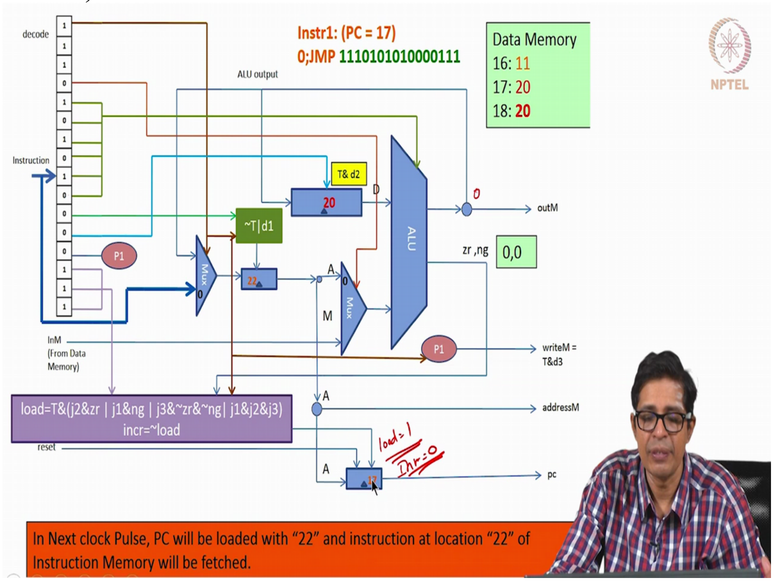
Data Memory
16: 11
17: 20
18: 20

load=T&(j2&zr | j1&ng | j3&~zr&~ng| j1&j2&j3)
incr=~load

In Next clock Pulse, "A" register will be loaded with "22", PC will be incremented and Instruction at "17" in Instruction Memory will be fetched.

So your 18th location got 20 and your PC did get increment to 16. So what is there in 16? Again it is an at instruction which is 10110. So 10110 is Jack Jack 22. So since it is again an A instruction, this Mux will route that 22 that is coming as a part of the instruction to the mouth of the A register. The A register's load is 1 your D register's load is zero, memory is zero and the PC is in increment mode. So nothing happens to the memory or the D register, your A gets updated to 22 while the PC gets Jack Jack incremented to 17.

(Refer Slide Time: 35:49)

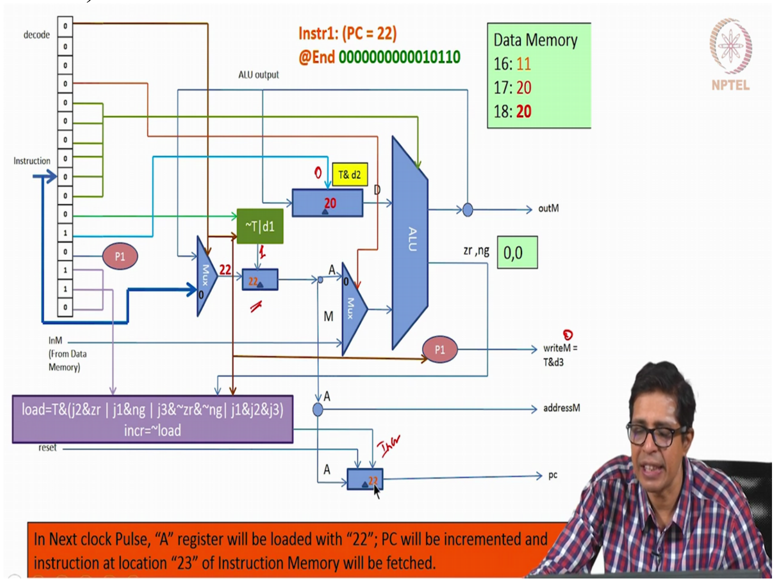


In Next clock Pulse, PC will be loaded with "22" and instruction at location "22" of Instruction Memory will be fetched.

Now in the tick of the next clock, the instruction and 17 is brought here and please note that this is going to be a unconditional jump, 0: jump Right? So what will happen? That in this case your A address, A register or the D register will not be touched, your memory also will not be touched but your PC instead of incrementing from 17 to 18, should now get updated with 22. PC is currently in 17. Now it should get updated by 22. That is whatever is there in the A register.

So now you note that the load is one. When the load is one, what will happen? The A register's content will get uploaded here. So the PC will now will become 22 while the Jack nothing will happen to the A register or the D register or the memory. PC alone instead of becoming 18, will now become 22. Please note that for the unconditional jump, your J1, J2, J3 are 1. So in this context, your Jack J1 and J2 and J3 are one and the T bit is also one. So your load actually becomes one.

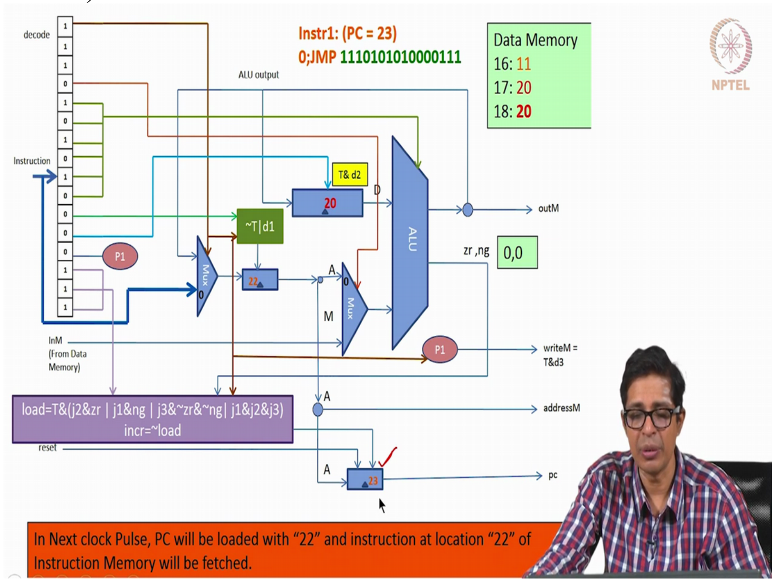
(Refer Slide Time: 37:09)



In Next clock Pulse, "A" register will be loaded with "22"; PC will be incremented and instruction at location "23" of Instruction Memory will be fetched.

And so in the next, you get 22. So again you fetch the instruction 22, instruction in the memory 22 Jack in the instruction memory 22 which is again an at instruction which loads 22 back into the A register right? So again this is an A…so this Mux will route that 22 that is coming as a part of the instruction here and the again the A bit is 1, load for the A bit is one, load for this D register is zero, the load for memory is zero and this is incremented. So what will happen? So this Jack A register will become 22 and the PC will get incremented to 23. Okay.

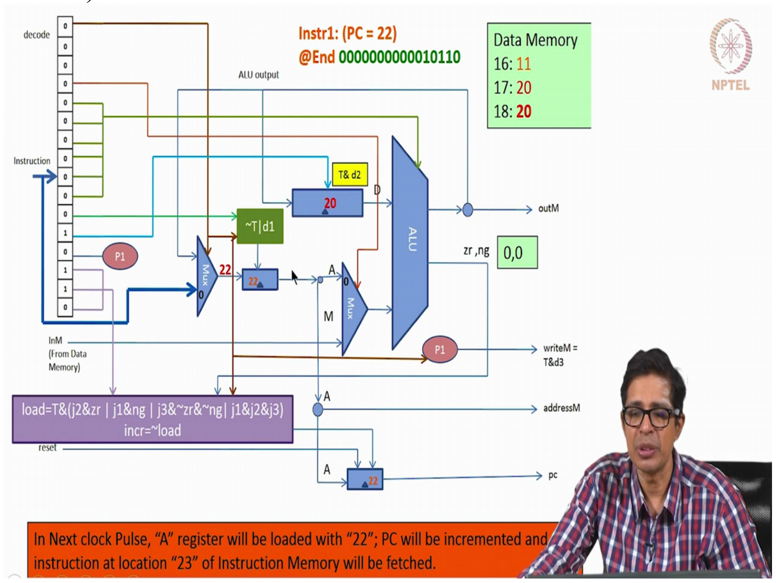
(Refer Slide Time: 37:51)



In Next clock Pulse, PC will be loaded with "22" and instruction at location "22" of Instruction Memory will be fetched.

PC got incremented to 23. And again, in 23, it is again 0: jump. 0: jump means this is again unconditional jump. So again the PC should instead of becoming 24, it should now become 22 and that happens. Note that the A register's load is zero, D register's load is zero, write to memory is zero but the load to the PC is one because your J1, J2, J3 R1 and your T instruction is one. So your Jack your PC now gets instead of getting incremented to Jack 24 it again gets back 22.

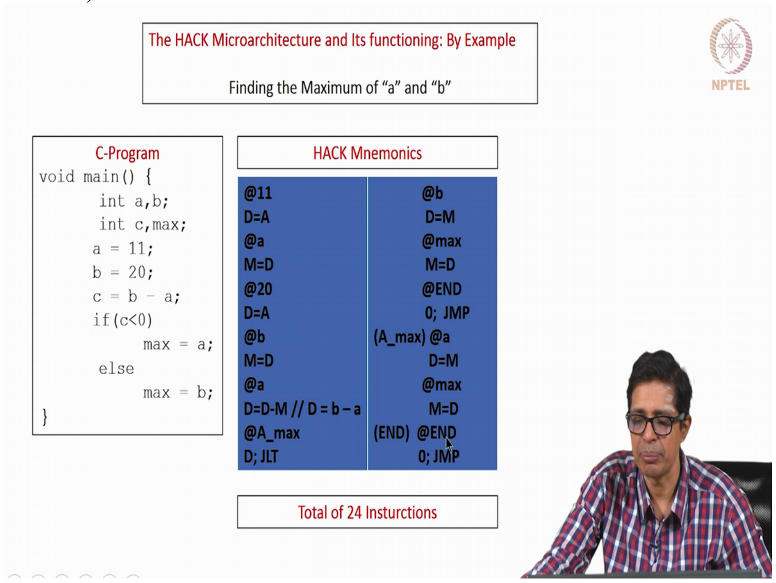
(Refer Slide Time: 38:32)



In Next clock Pulse, "A" register will be loaded with "22"; PC will be incremented and instruction at location "23" of Instruction Memory will be fetched.

So again PC becomes 22. It is again the same at End instruction. So this is an infinite loop that we had created at the end of this as you see here. So again PC equal to 23, PC equal to 22. So this repeats. So this is an infinite loop that we have just done as a part of this program. Okay.

(Refer Slide Time: 38:52)



The HACK Microarchitecture and Its functioning: By Example

Finding the Maximum of "a" and "b"

C-Program

```
void main() {
        int a,b;
        int c,max;
       a = 11;
       b = 20;
       c = b - a;
       if(c<0)
              max = a;
        else
              max = b;
}
```

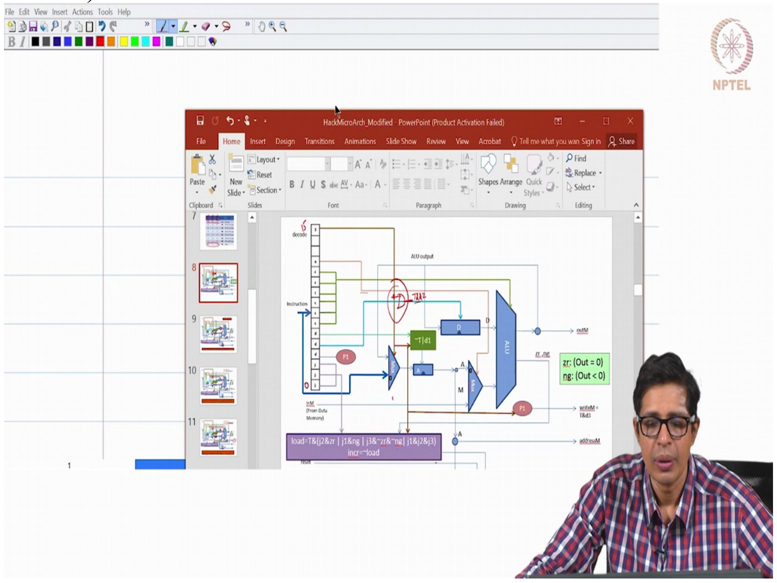
HACK Mnemonics

```
@11                  @b
D=A                  D=M
@a                   @max
M=D                  M=D
@20                  @END
D=A                  0; JMP
@b              (A_max) @a
M=D                  D=M
@a                   @max
D=D-M // D = b – a   M=D
@A_max          (END) @END
D; JLT               0; JMP
```

Total of 24 Insturctions

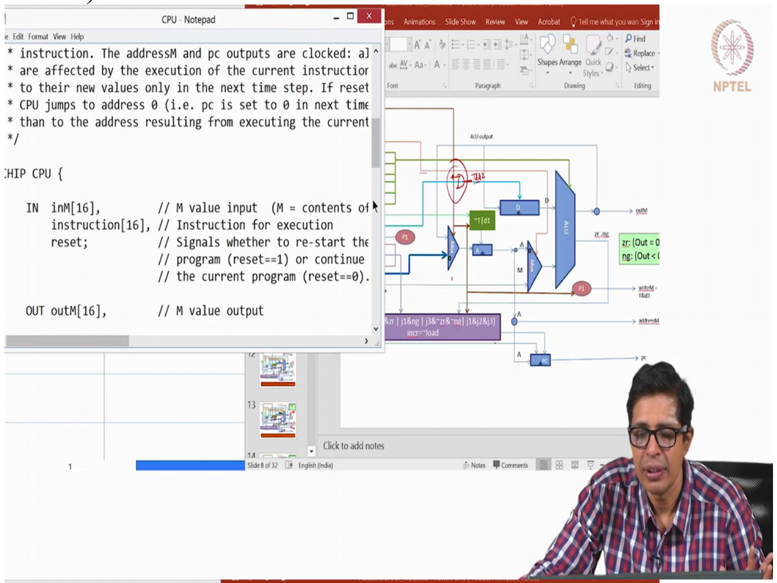
So we have done this, at End, 0: jump. This is between 22 and 23. So what we have done in this particular module so far is that we have explained the architecture by making one complete simulation of a simple program. So this will give you more clarity of how the architecture needs to be built.

(Refer Slide Time: 39:30)

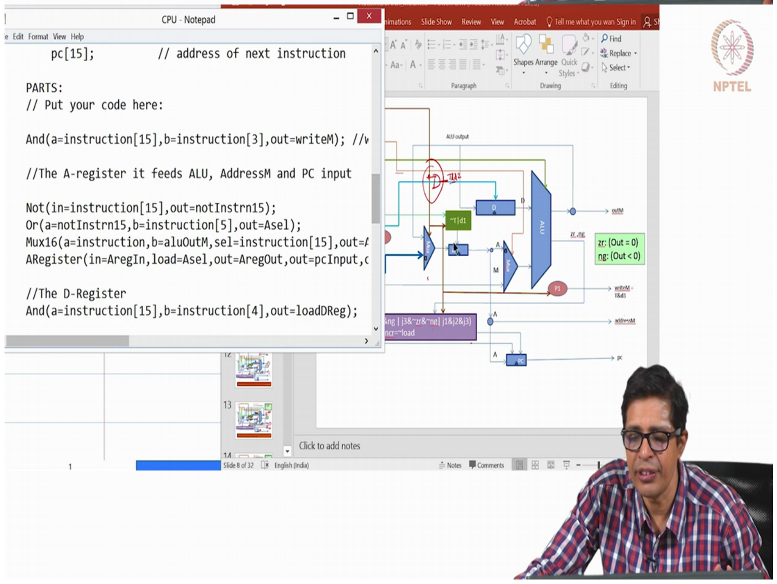


Now what we will do is that we basically look at this architecture, now we will just see what how the program is written.

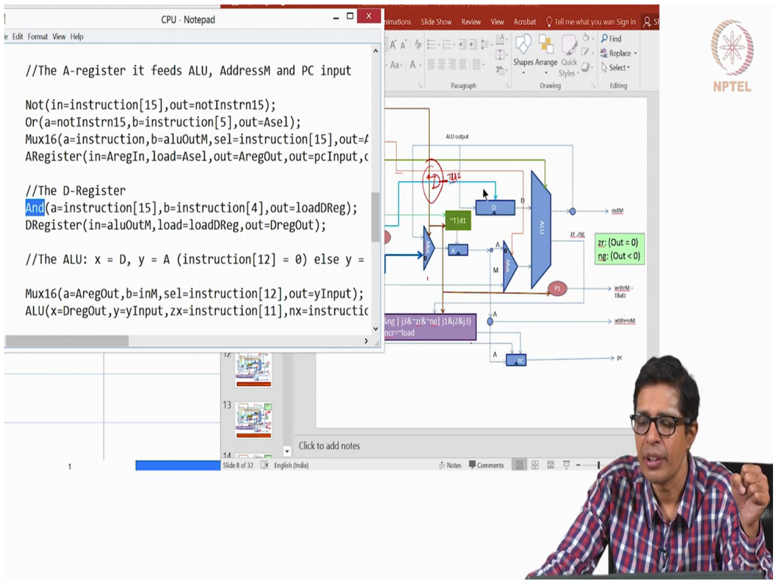
(Refer Slide Time: 40:02)



```
 * instruction. The addressM and pc outputs are clocked: al
 * are affected by the execution of the current instructior
 * to their new values only in the next time step. If reset
 * CPU jumps to address 0 (i.e. pc is set to 0 in next time
 * than to the address resulting from executing the current
 */

HIP CPU {

    IN  inM[16],         // M value input  (M = contents of
        instruction[16], // Instruction for execution
        reset;           // Signals whether to re-start the
                         // program (reset==1) or continue
                         // the current program (reset==0).

    OUT outM[16],        // M value output
```

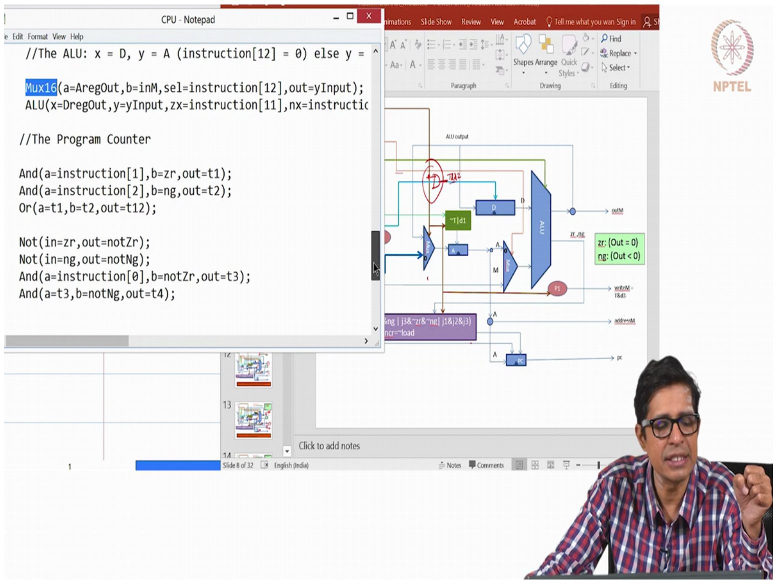


```
        pc[15];          // address of next instruction

    PARTS:
    // Put your code here:

    And(a=instruction[15],b=instruction[3],out=writeM); //w

    //The A-register it feeds ALU, AddressM and PC input

    Not(in=instruction[15],out=notInstrn15);
    Or(a=notInstrn15,b=instruction[5],out=Asel);
    Mux16(a=instruction,b=aluOutM,sel=instruction[15],out=A
    ARegister(in=AregIn,load=Asel,out=AregOut,out=pcInput,c

    //The D-Register
    And(a=instruction[15],b=instruction[4],out=loadDReg);
```



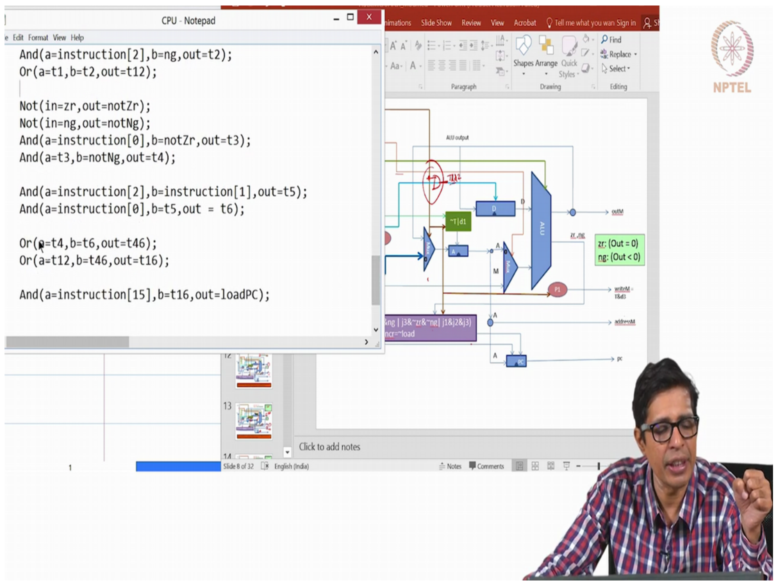
CPU - Notepad

```
//The A-register it feeds ALU, AddressM and PC input

Not(in=instruction[15],out=notInstrn15);
Or(a=notInstrn15,b=instruction[5],out=Asel);
Mux16(a=instruction,b=aluOutM,sel=instruction[15],out=A
ARegister(in=AregIn,load=Asel,out=AregOut,out=pcInput,c

//The D-Register
And(a=instruction[15],b=instruction[4],out=loadDReg);
DRegister(in=aluOutM,load=loadDReg,out=DregOut);

//The ALU: x = D, y = A (instruction[12] = 0) else y =

Mux16(a=AregOut,b=inM,sel=instruction[12],out=yInput);
ALU(x=DregOut,y=yInput,zx=instruction[11],nx=instructic
```

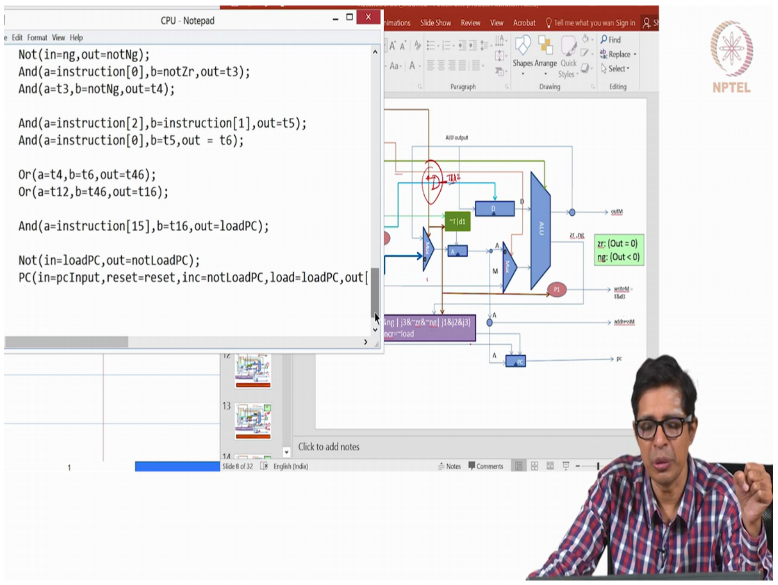


CPU - Notepad

```
 //The ALU: x = D, y = A (instruction[12] = 0) else y =

 Mux16(a=AregOut,b=inM,sel=instruction[12],out=yInput);
 ALU(x=DregOut,y=yInput,zx=instruction[11],nx=instructic

//The Program Counter

And(a=instruction[1],b=zr,out=t1);
And(a=instruction[2],b=ng,out=t2);
Or(a=t1,b=t2,out=t12);

Not(in=zr,out=notZr);
Not(in=ng,out=notNg);
And(a=instruction[0],b=notZr,out=t3);
And(a=t3,b=notNg,out=t4);
```



```
And(a=instruction[2],b=ng,out=t2);
Or(a=t1,b=t2,out=t12);

Not(in=zr,out=notZr);
Not(in=ng,out=notNg);
And(a=instruction[0],b=notZr,out=t3);
And(a=t3,b=notNg,out=t4);

And(a=instruction[2],b=instruction[1],out=t5);
And(a=instruction[0],b=t5,out = t6);

Or(a=t4,b=t6,out=t46);
Or(a=t12,b=t46,out=t16);

And(a=instruction[15],b=t16,out=loadPC);
```



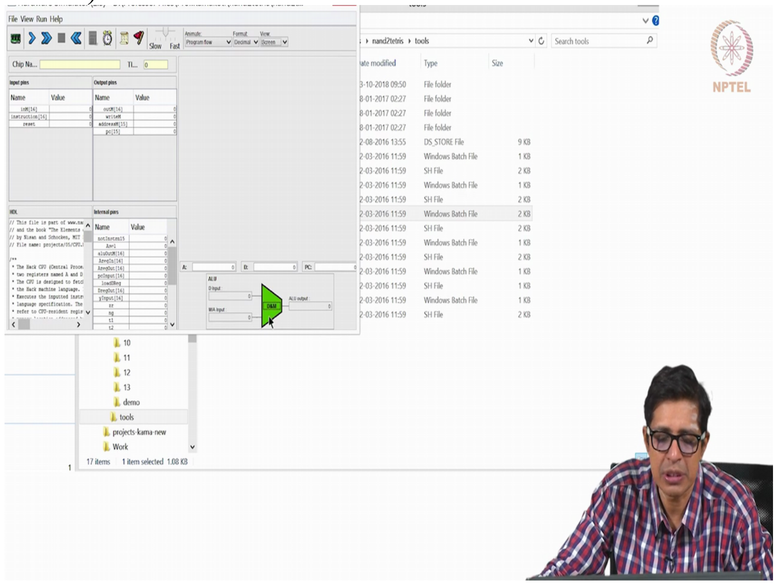
```
Not(in=ng,out=notNg);
And(a=instruction[0],b=notZr,out=t3);
And(a=t3,b=notNg,out=t4);

And(a=instruction[2],b=instruction[1],out=t5);
And(a=instruction[0],b=t5,out = t6);

Or(a=t4,b=t6,out=t46);
Or(a=t12,b=t46,out=t16);

And(a=instruction[15],b=t16,out=loadPC);

Not(in=loadPC,out=notLoadPC);
PC(in=pcInput,reset=reset,inc=notLoadPC,load=loadPC,out[
```

So go to the project, go to 05, let us see the CPU and let us have this side-by-side. Now you see that as you see here, Jack so the CPU basically has a 16 bit instruction that is given here as we are seeing. And then a 16 bit memory, this is coming from the memory inM and a reset. Reset comes from outside as you see here. And then what goes out of the CPU is something, there is an outM which goes which is 16-bit. There is a writeM which goes to the memory, the data memory whether it should be written or not, there is an address to the data memory and the PC that is going back to the ROM.

So these are all the outputs. So this is how we have created this and please note that now this Mux 16 that we are seeing here, of course corresponds to this Mux as you see here and this A register can be you can use the register that you have done as a part of your previous assignment project. But you can also use this A register which is a built in chip because then you get that A register in the simulation. So you get a figure of the A register, so you can actually monitor what is happening inside A register separately.

You can just use register or you can use A register, both will work. But if you put this A register, on the screen you will see a separate block for A. You can keep monitoring it. So this is… And similarly the D register here and Jack so this this AND gate is basically for this the load for the D register and this OR and NOT is for the load for this A register whatever you see here. And this Mux 16 that you see is, this Mux 16 which is feeding to the ALU, then you instantiate the ALU which you have done earlier and all these Jack 7 OR things that you see here, this these instructions are for this load and increment logic. That is for the PC, right. And you also instantiate this whole thing, 7 or 8 this thing and then you also instantiate this PC here, this is for the PC. So this is how you create the and this is all that you need to do to basically generate this CPU.

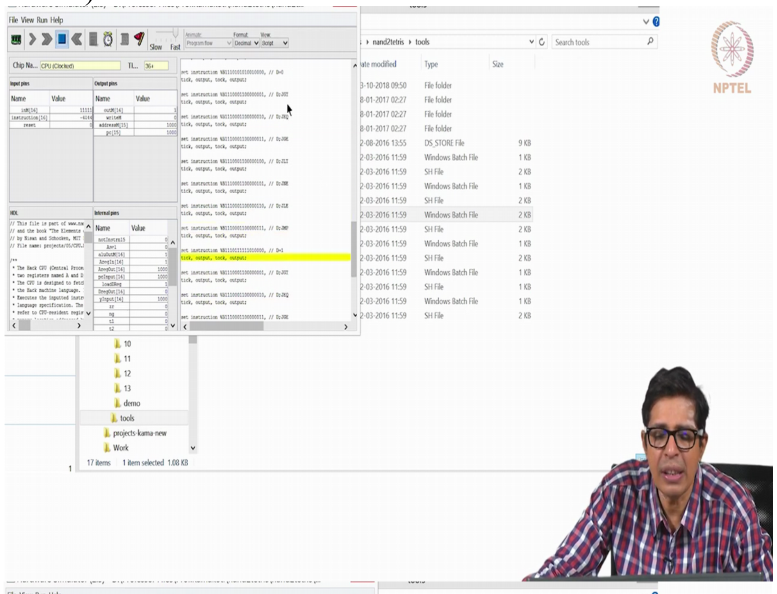
(Refer Slide Time: 42:53)



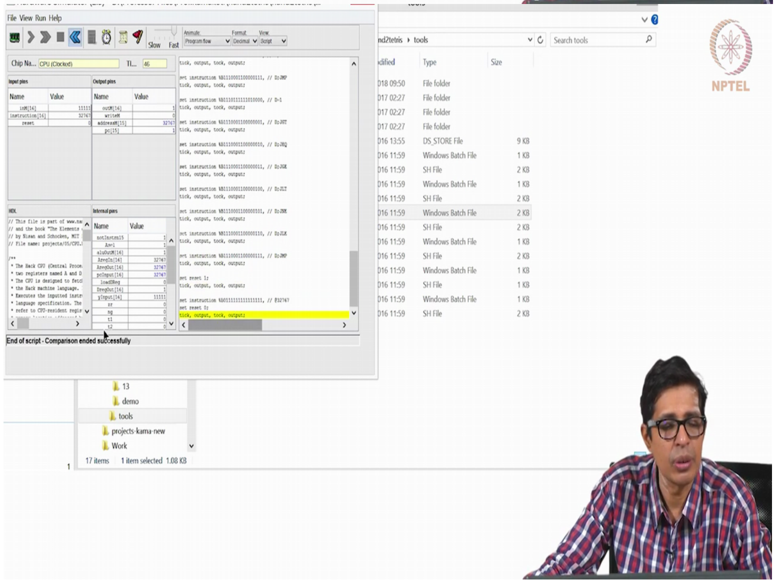
Now very quickly, we will we will go and execute this CPU. Let us go and take the hardware simulator. So I can basically take this CPU. Now, I am loading the CPU, I am loading the script.

So there is a CPU here. There are 2 scripts, CPU and CPU external. So you need to load both the scripts and see how it is working. So since I used that capital A register and the capital D register, if you see the screen here, you will see one separate entry for the A register and the separate entry for the D register. Of course PC is there and this ALU of course is also here. Okay. So we can use this. Right?

(Refer Slide Time: 43:48)



```
set instruction %B1110101010010000, // D=0
tick, output, tock, output;

set instruction %B1110001100000001, // D;JGT
tick, output, tock, output;

set instruction %B1110001100000010, // D;JEQ
tick, output, tock, output;

set instruction %B1110001100000011, // D;JGE
tick, output, tock, output;

set instruction %B1110001100000100, // D;JLT
tick, output, tock, output;

set instruction %B1110001100000101, // D;JNE
tick, output, tock, output;

set instruction %B1110001100000110, // D;JLE
tick, output, tock, output;

set instruction %B1110001100000111, // D;JMP
tick, output, tock, output;

set instruction %B1110111111010000, // D=1
tick, output, tock, output;

set instruction %B1110001100000001, // D;JGT
tick, output, tock, output;

set instruction %B1110001100000010, // D;JEQ
tick, output, tock, output;

set instruction %B1110001100000011, // D;JGE
```



```
tick, output, tock, output;

set instruction %B1110001100000111, // D;JMP
tick, output, tock, output;

set instruction %B1110111111010000, // D=1
tick, output, tock, output;

set instruction %B1110001100000001, // D;JGT
tick, output, tock, output;

set instruction %B1110001100000010, // D;JEQ
tick, output, tock, output;

set instruction %B1110001100000011, // D;JGE
tick, output, tock, output;

set instruction %B1110001100000100, // D;JLT
tick, output, tock, output;

set instruction %B1110001100000101, // D;JNE
tick, output, tock, output;

set instruction %B1110001100000110, // D;JLE
tick, output, tock, output;

set instruction %B1110001100000111, // D;JMP
tick, output, tock, output;

set reset 1;
tick, output, tock, output;

set instruction %B0111111111111111, // @32767
set reset 0;
tick, output, tock, output;
```

End of script - Comparison ended successfully

So we will just see the script here. So we will just run the script. We need to run the script. So all the instructions are tested in the script and then you see and it has come back successfully. Similarly, you load the next script which is CPU external, load the script and again you run the script, this script also and you see all these jumps are being tested one after another and this also ends as successful. So at the end of this, you have done the hack CPU correctly. Now we will go to the next module where we will be talking about the data memory.