

Foundations To Computer Systems Design
Professor V.Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 4.2
The HACK Microarchitecture

(Refer Slide Time: 00:18)

The image shows a presentation slide with a white background and blue horizontal lines. The title 'Module 4.2 The HACK Microarchitecture' is written in red cursive. In the top right corner is the NPTEL logo. At the bottom right, there is a small video window showing a man with glasses and a blue shirt. At the bottom left, the text 'Module 4.2: The HACK Microarchitecture' is visible. At the bottom center, the text 'PROF. V. KAMAKOTI IIT Madras' is visible.

So welcome to module 4.2 The HACK Microarchitecture, the expectations of the HACK Microarchitecture has been set forth in the module 5.1 wherein we said that it had to do five operations for every instruction so the PC the program control point to some instruction in the instruction memory and it has to fetch it decode it filter the relevant data execute the instructions ((00:40)) then go the next instruction and by that time the PC will also get updated what should be the next instruction.

The next instruction can be the instruction with this follow unit in which case the PC is incremented or it can be some other instruction as in the case of a control transfer instruction like a conditional jump or an unconditional jump. In that case the PC will get the value the address of the next instruction from the A register. So to this level of detail we have finished of a module 5.1 and if you have any doubts please do raise it up in the discussion forum you can address this. So what we will do in this module 5.2 this we will may go so there are two classes of instruction only the A and C instructions and that has to be executed by the HACK Microarchitecture.

So how do we build this microarchitecture from the basic combinational and sequential circuits that we have build up as part of project 1, 2 and 3? So that is for we are going to take up, so this is a context of what we are doing.

(Refer Slide Time: 1:53)

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	O; JMP
@b	(A_max) @a
M=D	D=M
@a	@max
D=D-M // D = b - a	M=D
@A_max	@END
D; JLT	O; JMP

Total of 24 Instructions





Module 4.2: The HACK Microarchitecture
PROF. V. KAMAROTTI

So I will be using set of slides for this to explain. So we want to understand the HACK microarchitecture and its functioning and the best way to learn this is by an example right. So we will take a sequel we will take its equivalent mnemonic code right and then we will basically go and do the next step of how the mnemonic is getting translated into machine that we have seen as a part of our module 3 and once that machine languages available the zeros and ones are available how does the you know the architecture takes those zeros and ones and carry out (carries out) the necessary action, so that part we need to understand and for that we will be using set of slides.

Now (in the) simplest example I am going to take to explain you the entire HACK microarchitecture is used to is one which will find the maximum of two numbers a and b. We could receive program that we have on the side of this so the so there is a void main, int a or b, c is max I make a is 11 I make b equal to 20, I make c equal to b minus a if c is less than 0 then max is a else max is b right. So they are very simple program which will find the you know the maximum of a and b. Now how do we do this right, now we assume there are 24 instructions this will essentially get (transfer) transformed into 24 instructions and those are as follows.

First thing is the first (instruc) first translation would be at a equal to 11 so what we are doing here? We are basically these two instructions will tell you what you mean by this at a equal to 11. So at 11 D equal to A, so you just take D as such our microarchitecture has two registers D register and A register so the register D will basically take the value 11 right, now let us assume that a is stored somewhere so @a is the address of a, this is the address of a, M equal to D so now A becomes 11 so M is M of whatever is stored in the address register that will now become D, D is 11 this is what M equal to D means right.

So when I say @a the address of a is loaded into the capital A register and the moment I say M equal to D that M of A becomes 11 that means the variable A gets a value 11, A is equal to 11 is completed. Similarly @20 now D becomes 20, now @b this is the address of b now M equal to D that means b equal to 20 ok. So a equal to 11 is done by the first four instruction b equal to 20 is done by the second four instructions (right) D equal to 20 is done by the second four instructions. Now I have do C equal to b minus a right, c is a temporary variable for me so I will not I don't have a I don't want the need a location specific memory location for c.

So the way I am compiling it is @a again I go to the location a and I say D equal to D minus M, what is D here? D is storing the value of b here as such D is storing the value of b now again I say @a so in the capital A register the address of small a will be there now I say D equal to D minus M that means now D will be storing b minus a because M now is get the value of small a because address that is there in the capital A register is that of small a so M will get the value of small a. So now D equal to b minus a. Now I am putting a label @a max the label here is a max.

Now D, J less than if D is currently storing b minus a this JLT essentially says that this result is less than zero take a jump so if b minus a is less than zero that is c is less than zero I will go to a max right, otherwise I will stop doing the set of instructions, What are this instructions doing? First @b so D will now become b now @ max right so max is the memory location for max right so M equal to D so M equal to D is M gets the value b that means max is equal to b because the address now points to max so when I say M, M equal to b so max gets the value b.

Now @end, end is a label which is given here I zero colon (jump) this is an unconditional jump so I will go and jump to this end and then this is a infinite loop just to say that I have finished I just be looping here. So if your b minus a is a is greater than or equal to zero then your max is

equal to b that is what we are doing if c is greater than or equal to zero, c is b minus a whatever at zero then max is equal to b that is max is equal to a so this a max is anything at a D equal to M so D gets the value of a because at a less the address of a to the capital A register so that is M so I get the value of small a and so at max M equal to D.

So again this two instructions as you see will make M is equal to A. So when b minus a is less than zero max when b minus a is less than zero max is A otherwise max is b then it will come to this end and jump. So the secret that you see on the left hand side essentially gets translated to this HACK mnemonics right. I just re-did this for just for you to recap because we have done it as a part of module 3(9:25) that I am just doing this as a recap so that it reinforces your understanding of the mnemonics and its equivalent high level language programming concept.


(Refer Slide Time: 9:41)


The Machine Instruction

@11	0	000000000001011	@b	12	000000000010001
D=A	1	1110110000010000	D=M	13	1111110000010000
@a	2	000000000010000	@max	14	000000000010010
M=D	3	1110001100001000	M=D	15	1110001100001000
@20	4	0000000000010100	(END)	16	0000000000010110
D=A	5	1110110000010000	0; JMP	17	1110101010000111
@b	6	000000000010001	(A_max) @a	18	000000000010000
M=D	7	1110001100001000	D=M	19	1111110000010000
@a	8	000000000010000	@max	20	000000000010010
D=D-M	9	1111010011010000	M=D	21	1110001100001000
@A_max	10	000000000010010	(END) @END	22	0000000000010110
D; JLT	11	1110001100000100	0; JMP	23	1110101010000111

Note:

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





Module 4.2: The HACK Microarchitecture
PROF. V. KAMARAJI

Now what we do now, now this is what the corresponding machines instructions would be, so the mnemonic essentially becomes this machine instructions so the translation from the mnemonic to the machine instruction is done by the assembler ok, we will (9:58) we understood what assembler is but how the real assembler is constructed we will see in the subsequent modules. Now this are the instructions so there are 24 instructions so this those are (will) stored in the memory from 0 to 23. So this is the instruction memory as you see here this is the instruction memory so the instructions are stored from 0 to 23.

So at 11 is the a instruction so the first bit is 0 and the remaining bits if you see it gives you 1 0 1 1 which is 11, D equal to A this is the C instruction the first three bits are ones right and then we go back to module 3 to understand that the a bit should be zero, when the a bit is zero then basically this are the 1 1 0 0 0 this are the bits that are needed for the computation by the ALU. So this will correspond to a Z x, n x, Z y n y f o the functionality and n o, f and n o so this are six computation bits 0 1 0 is the destination bit and 0 1 0 essentially says that your destination is D if you go and look back and this is not a jump instruction that is the last three zeros, so the D equal to A essentially gets translated to this.

Now @a again this is same a instruction which is zero and let us assume that a is stored in address a, b and max are stored at address 16, 17 and 18 in the data memory, this 16, 17 and 18 in the instruction memory is occupied by this instructions 16, 17 and 18 in data memory is occupied by a, b and max ok. So when I say @a it means that a is stored at 16 so this is zero and we look at the remaining things this corresponds to the decimal value 16, 1 0 0 0 . Now I am saying M equal to D again this is a c instructions the first three bits are (your) this a basically decides that the source operand is D and 0 0 1 1 0 0 are the corresponding computation bits and then this three 0 0 1 essentially says that the destination is M and then this says it is not a jump last three zeros that it is not a jump instruction.

Now again at 20 this is an a instruction and this is when you see 1 0 1 0 0 0 this 16 plus 4, 20 this is a decimal binary equivalent of 20, D equal to a this is a C instruction this with zero bit essentially says that this is the operand a, 1 1 0 0 0 0 essentially gives you the computation bits 0 1 0 is the destination which says that destination should be D and then this is not a jump instruction. Now at b, note that B is stored at 17 so this is an a instruction so this is zero and the remaining thing correspond to 17. Now M equal to D, this is a C instruction again 1 1 1, this zero bit will tell you about the destination operand that is D, 0 0 1 1 0 0 is the computation bits 0 0 1 is basically the destination which says destination is N and then 0 0 0 is a jump.

You can also refer to the table that we have introduced in the module 3 and in the book which gives you a combination of a and the computation bits for a0 and c1, c2, c3 till c6 (())(14:22) then the corresponding operation that will happen and then there is also a table that we have introduced wherein your the destination bits depending on the three bits destination bits (where)

what is the corresponding, is it M or b what is the destination? And also there is a table which says the seven different possible jumps ok which corresponds to a last three bits.

So this we have already taught so just go back and have the table to your side so that this you can understand this much better right. Now next is @a so this is a instruction and a is stored at 16 so you get 16 here, D equal to D minus M so this is a C instruction this is a which says a is one and the computation bits are 0 1 0 0 1 1 then the answer will be D minus M and then you have the 0 1 1 0 0 (0 1 0) which basically says the your destination is D and this is not a jump. Now @a max ok this is something that we need to understand so what is a max? a max actually resolves to this location which is 18, right a max is a symbol which is associated with the address number 18 right that who will resolve that so essentially what this is A instruction so 0, this is 18 if you take this set of bits this is now 18, 1 0 0 1 0.

Who will resolve this for you? This will be resolved by the assembler, the assembler is responsible so the assembler will fist scan this it will be one pass, one pass is from end to end of the previous thing that we see. So this will do a one pass of this and find out that a max is going to be 18 and similarly end is going to be 22. So for all the symbols that we have used here namely a max and n it will find out what is the corresponding binary valued the it because this things basically point to addresses so it will find out which address it is mapping.

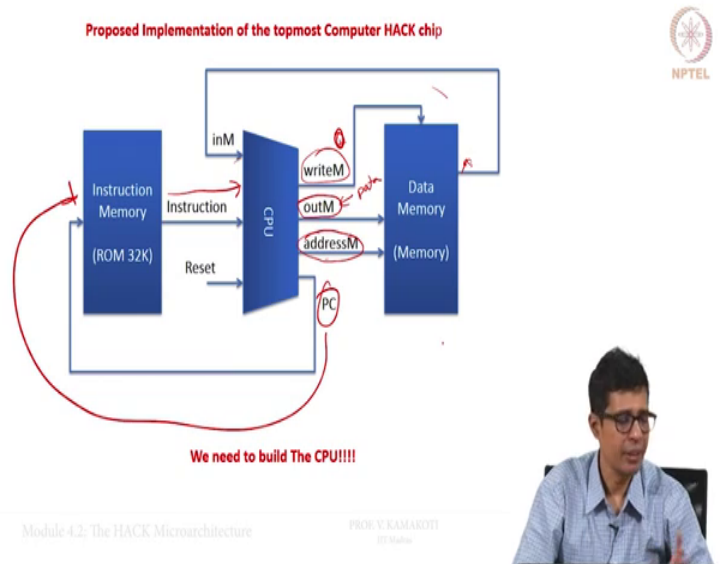
So at a max now it knows that this is you know 18 then D JLT so this is a c instruction so first 1 1 this is 0 essentially says the destination of source operand here, 0 0 1 1 0 0 essentially says that you are going to route it of the ALU will be D 0 bit is 0 and c bits are 0 0 1 1 0 0 the out of ALU will be b and there is no destination here there is a jump instruction and I the jumps are 1 0 0 which essentially says that I want to do a JLT jump when(())(17:33). Ok this is there is a jump table right seven different jumps and what are their encodings, so 1 0 0 corresponds to JLT, right.

Now again at b, D equal to M at max so we said that the max is at location 18 so this will come 18 n equal to D again we are at end, at end resolves to 22 where the two passes and so 22 will be 1 0 1 1 0 this is an A instruction and then zero calling jump and all this things ok. So essentially this is how your this is the interpretation what do mean went to this interpretation is because we need to have an understanding of how the machine instruction has to be interpreted by the architecture, so the interpretation that we have seen now once this machine code of 0's and 1's is

presented to the architecture the way we are interpreting in English the circuit has to interpret and carry out the necessary action and that is the essentials cucks of design of an architecture (right).

Now in the next module that is 4.3 we will go and see how the architecture basically take this bits and interprets it and how do they get the resolve the issues?

(Refer Slide Time: 19:13)



Before we conclude this is the proposed implementation of the to most computer HACK chip so the top most computer HACK chip will have an instruction memory will have a CPU and will have a data memory right. So it will have instruction memory it will have CPU it will have data memory right. Now and the CPU will have a program counter, internal to the CPU, a program counter will send its address the program counter will send its address to the ROM the instruction memory.

That instruction will come here, right, that instruction when its executing it will ask for read or write into memory this is the right terms so this will be used for the read or write and this is the address and if it a write operation the data will come in out M if it is a read operation and write operation data will come in out M and this write will become 1 and this will be the address. So if write is 1 out M is data then address M is the address so in this address the data will be readed. If write M is 0, then out M is not used it is don't care and address M so you read from this address back.

So the output of this data memory essentially comes back the output that is one output of the data memory which will come back to a CPU that is when read when the CPU reads some value from the memory that thing has to come from this part. So this is how the data memory and instruction memory (CPU) and the CPU works. The CPU essentially has a certain sequential component like a register and PC, two registers D and A and a PC this are the three sequential components inside the CPU and all the remaining things that we see all the remaining things that we see inside the CPU are combinational elements and we will be using the ALU will be using many-many things.

So this is how this the proportional implementation to some of this part there is a CPU, there is a PC that PC basically gives the address of the next instruction, based on that PC the instruction memory in a tick of a clock basically gives you this CPU address and in this CPU so we have write M, out M and address M, so you give a particular address and write M is zero and your then this data memory will (feed) read the particular address and give back the content of that address through in M which will come to the CPU.

When the CPU has to write something into the memory that will be the end of that operation it will make write M as 1 and it will put whatever it wants to write on the out M bus and the address in address M so the data memory essentially gets updated. So this is the proposed architecture of HACK and we will continue more on this as we proceed through the next steps, thank you.