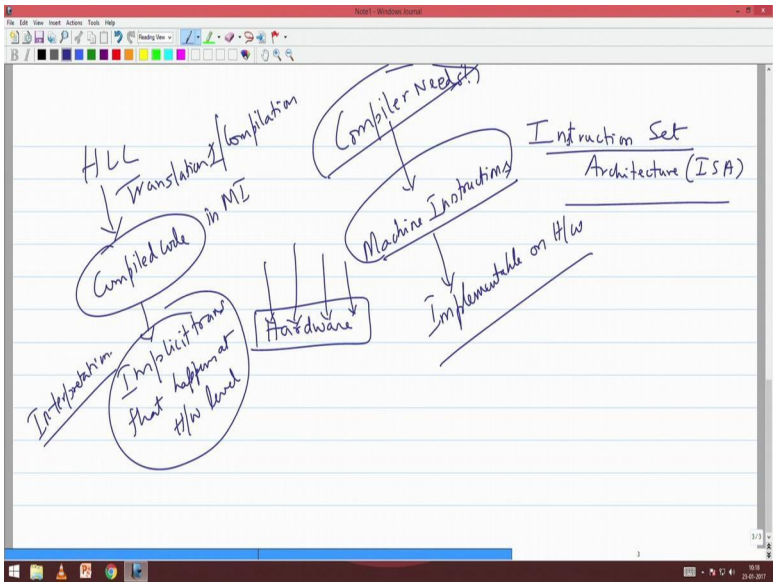
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 09
Instruction Set Architecture

(Refer Slide Time: 00:21)



So, what is an instruction set architecture. So, I have a hardware there are some instructions which this hardware can understand those instructions are called machine instructions these machine instructions have to meet 2 ends this should satisfy the need of the compilers and on the other hand it should also be implementable. So, the machine instruction rather the instructions that are understandable by the machine have 2 ends to meet one thing is it should be it should satisfy the needs of the compiler and at the same time it should be implementable on the hardware.

Now; that means, when this machine instructions are given to the hardware the hardware understands this what do you mean by implementable on hardware it should be understandable to the hardware in some sense what do you mean by understandable to the

hardware the hardware actually interprets these instructions in some form. So, implicitly there are 2 translations that are happening when you write a program. So, I have a high level language there is one level of translation I could call translation one which gives you a complied code in machine instructions. So, this is an explicit

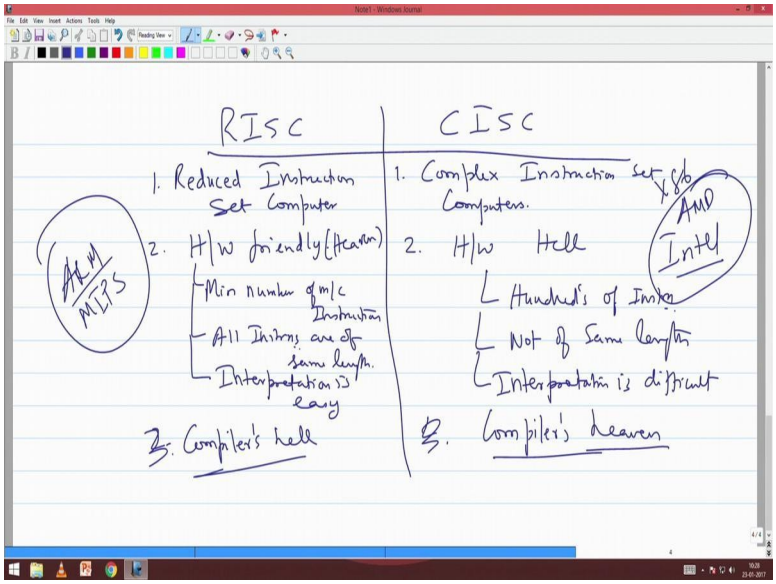
translation wherein you can see the executive double code encoded in the machine language right the language of the machine now these instructions are pushed one by one into the hardware. So, now, the hardware now interprets each of this instruction for execution. So, there is an implicit translation that happens at hardware level.

So, when we are looking at instructions there is a translation or what we call as compilation which is explicit and then there is what another translation which we call it as interpretation which is implicit right. So, in last course I have distinguished you between what do you mean by a compilation and an interpretation what is compilation I take a program compile it end to end convert it end to end translate it end to end and create another executable which is the in the form of the target language this is called compilation.

Right, but then take a program take every instruction execute then take the next instruction execute right. So, this is actually called as interpretation. So, the difference between a translation a compiled program and an interpreted program is that this is one interesting example if I take the hardware the first level of translation which takes the higher level language program and converts into a machine language that is compilation now this machine language program is taken is put to the system hardware which takes instruction by instruction understands it and executes it somehow we should know that this is add this is multiply. So, there is something some semantics associated with the instruction which is understood by the hardware and that is basically interpretation.

So, when I look at you know. So, when I am trying to develop an instruction set architecture I should look now at 2 aspects one thing is there is a great compiler need compiler needs so many things. So, I need to have those features which will be supportant, but at the same time those features cannot be. So, fancy that it is not implementable on the hardware. So, I have to strike a balance between this and that balance is what gives you gives rise to a bigger classification of computing systems in general or based on what how biased I am to the hardware versus how biased I am to the compiler.

(Refer Slide Time: 05:07)



So, I can be right. So, I can be very compiler as an instruction set architecture I can be very friendly at the same time I can be literally hardware I can make the hardware mad I could have. So, much complexity on the other hand I could be. So, much hardware friendly while the while I will be very unfriendly to the compilers.

So, depending on how close the instruction set architecture or how friendly code un-code the instruction set architecture is to the compiler versus the hardware there is a classification of computing system based on that and that is what we mean by RISC versus CISC, RISC actually stands for reduced instruction set computers this actually CISC actually stands for complex instruction set computers. So, what do you mean by reduced instruction set computer you have already done in your third semester course what you did there is a RISC machine. So, the small hack hardware that you made the small processor toy processor that you made was a RISC machine in sense what you mean by that is that this RISC is more hardware friendly hardware friendly in the sense that it has minimum number of instructions right and then all instructions are of same length.

So, every instruction is say four bytes right say in this case minimum number of machine instructions. So, I may not have any fancy machine instruction. So, if I have a fancy compiler requirement right then I need to realize that compiler requirement using

these so called primitive instructions. So, it makes the compilers job extremely complex

because I have very primitive instructions and say I have to support say suppose in hack the hardware that you generated in your third semester suppose I want to put say protection.

I want to introduce segmentation it will kill you, but if I do not have segmentation we will see in the afternoons tomorrows afternoon class we cannot do even compilation. So, we are trying to do something that a lot of things that we keep in mind are going to be now complex. So, we for example, if I say that every instruction should be just 32 bytes I cannot even initialize register of the 32 bit length with an immediate value because I need 32 bit immediate value 32 bit immediate value itself will consume four bytes. So, how will I have an instruction that will initialize a register with a 32 byte value it is going to be extremely complex. So, we have to do a lot of round about ways by which I can go and initialize a register 32 bit register with a value right.

So, these are all very very difficult things that come up. So, if I have all instructions are of same length then even basic initialization of constants into registers like I go and say a equal to 515 and a is a 32 bit integer I cannot assign 515 to a as an assembly language because if I want to put 32 bits immediately four bytes are consumed each instruction can be only 4 bytes correct. So, there are very different tricks that I need to follow to basically get that value of you know some 500 - 510 or something into that value a it can become multiple instructions etcetera etcetera etcetera. So, I will talk about the armed instructions set architecture from the risk point of view while you will be writing a small mini kernel in CISC. So, I will not bother to teach about CISC you learn it through your assembly language, but about RISC we will talk a little bit more in subsequent classes also.

So, in essence RISC is hardware friendly it has minimum number of machine instructions, it has all instructions are of same length by this what happens you are the second level of translation that that I told you namely that interpretation is easy the con of this RISC is that it is compiler compilers hell it will create mash for the complier especially when I want to support fancy things this becomes extremely complex for you to code translate on the other hand this is a complex instruction set computers where this is completely hardware's hell, this is hardware's heaven while this is hardware's hell in the sense that it will have thousand hundreds of instructions the instruction set manual of

Intel; Intel is an example of this Intel AMD especially the X86 right is an example of this while on the other hand here arm MIPS these are all RISC machines.

We will have hundreds of instruction the manual is around 505, just a instruction set alone is 570 or 580 pages which basically will describe this instruction. So, this is one thing and they will not be not of same length if it is not of same length. So, I have a variable I associate with a register EAX or whatever and I say I want to initialize with any constant I can do it. So, the compiler need not actually spend lot of time to even to initialize a particular register with a particular constant are you able to get this. So, if the instructions are not of same length then it becomes much more easy, but then what is the interpretation is going to be extremely difficult, but surely this is compilers heaven why is the interpretation going to be difficult why is the interpretation going to be difficult.

Student: Because of the variable length description.

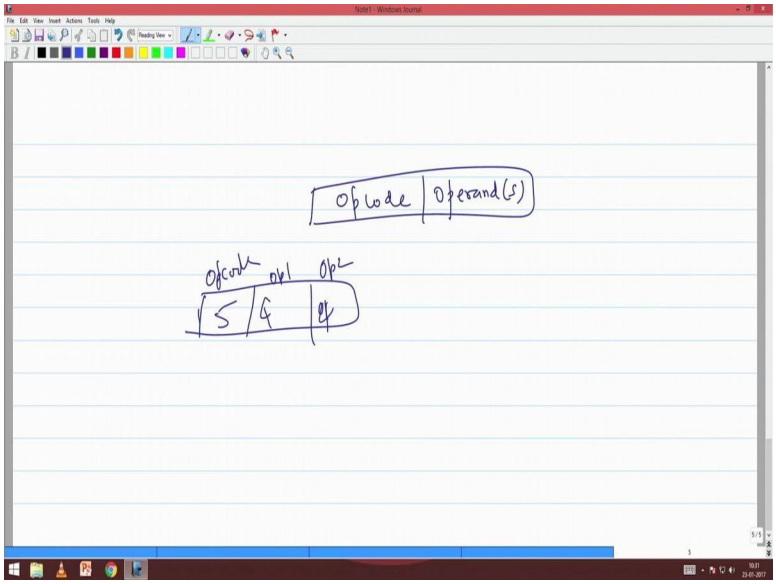
Why?

Student: Because we initially we do not know of how much length the instruction is going to come. So, before that we have to find out like or based on the length we have to find out what kind of instruction it is and then we have to assemble it.

See I do not know. So, when I set something from the memory I fetch it in blocks. So, so when I fetch something say I can have one bit instructions in X86 hen instructions can run to several bytes. So, I can have several instructions in one fetch or not even half an instruction in a single fetch and an instruction can start one path and end somewhere else in some other next fetch it can spun across 2 fetches and so if I am going to have 1 byte instruction to say 100 as 20 byte instructions or 10 byte instructions right suppose I am going to have these variety of things now your decoding itself becomes complex I have to see the first byte then the second byte then the third byte etcetera.

Right, but on the other hand if I have a fixed length an instruction basically has what do you what do you what are the things inside an instruction what are the things inside an instruction.

(Refer Slide Time: 14:52)



What will an instruction carry what do those bits represent it represents 2 things right it represents op code and operands right add something add is an op code on what should I add becomes the operand. So, every instruction will have op code and it will have a also have operands

Right in a fixed length normally what happens is we will reserve op say 5-5 bits for op code four bits for operand one four bits for operand 2 and so on let us say like this. So, when an instruction comes I can go and find out what the op code is and concurrently I can also go and find out where what the operands are right if you carefully look at what you have done in hack that is what you have done right in an instruction of 32 bit length we know exactly how to interpret it. So, we know exactly where the operand one is where the operand 2 which are the bytes the bits which are going to specify the operands that we will know very clearly in a fixed length instruction.

So, when I get 32 bit I can fix first 5 bits are for op code the remaining things like this they are organized. So, when the hardware see hardware is having several transistors 1; 1 part of the hardware starts interpreting what the instruction is concurrently the other set of hardware can start fetching the operands understand and start fetching the operands right, but in a variable length instruction I do not know where this types of instruction are going to be there. So, in a variable instruction when I want to process a decode a variable instruction

I go byte by byte and then find out what where the op code ends where the

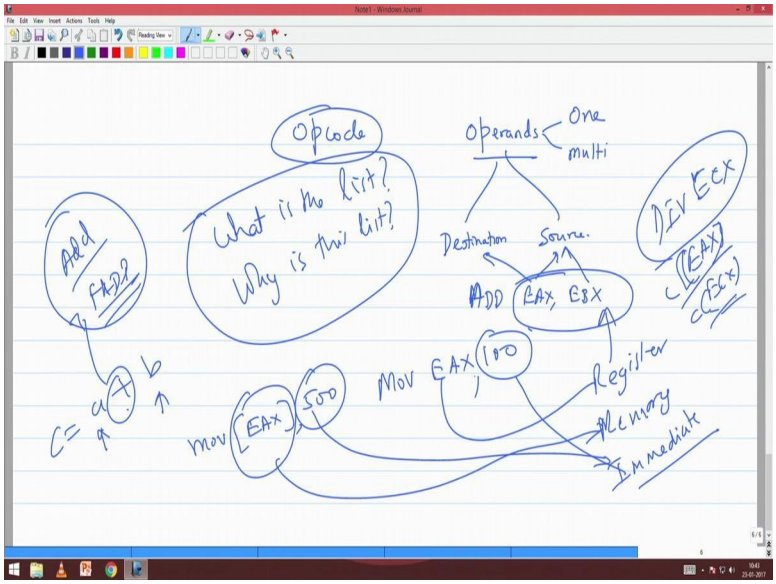
operands start etcetera, but in a fixed length instruction I need not go byte by byte I have more concurrency in understanding what do you mean by you know fetching the operands fetching this I am trying to understand what that instruction is.

So, my decoding becomes extremely fast and decoding and the corresponding actions become extremely fast in the case of RISC instructions because of fixed length while the same decoding and other things become extremely complex in the case of CISC because I do not know where I should look for the operands. So, the decoding essentially becomes almost sequential in the case of CISC in sequential in the sense I have 8 bytes of instruction I do not know whether 8 bytes will finish off will it have half the instruction or it is having more than one instruction or its exactly the instruction. So, I see start looking at byte one based on that I start looking at bytes 2 three etcetera. So, it can become almost sequential for me to decode an instruction. So, that is that is the basic difficulty.

So, RISC; so my interpretational RISC level is going to be extremely easy while CISC level is going to be complex, but on the other hand from the compilers prospective RISC is very very difficult in the sense it; it takes lot more effort for the compiler to generate a RISC based you know machine code while on the CISC it becomes much more easier right. So, so this is this is this is a classification of computers today arm is a RISC processor RISC V is a RISC processor while on the other hand Intel X86 is CISC processor. So, based on how the instruction set architecture is designed based on the length of the instruction sets there is now a distinction between 2 classes of processors which comes as CISC and RISC able to follow; yes or no.

So, let us go to the next step one thing is. So, as I told you RISC or CISC the instruction has 2 parts one is the op code another is the operands.

(Refer Slide Time: 19:06)



So, I could have one operand instruction I could have multi operand instructions operands are of 2 types one a destination operand and a source operand suppose I say add EAX comma EBX in MASM; EBX and EAX both are source operand and EAX is the destination operand. So, I could have an operand that is both source and destination now there could be there could be one operand or multi operands in this case there are 2 operands here right and in addition the op code right is the decision of what the op code is right it should be should also be. So, that is that is one very important. So, when I am coming out with an instruction set architecture.

What should be the op code what is the minimum op code that is necessary somebody has to come out with. So, what is the list why is this list right this we need to come out with a very clear justification for that similarly on the operand side there could be multiple types of operand. So, this is basically a register operand I could have memory operands I could have immediate operands right memory operands are those I want to add a register value with something in the memory I want to move something from the memory into a register I want to move something from a register into a memory right. So, every operand can be a register operand it can be a immediate value that is for example, move EAX comma 100, this 100 is a immediate operand EAX is a register operand I can say move EAX comma 500, this is a memory operand the address that of the that memory which I am trying to

access is the value of EAX and 500 is the immediate operand.

So, I could have instructions with one or multiple operands we already saw last time right div some ECX right we saw last time right; this basically gives EAX content of EAX by content of content of EAX by content of ECX correct. So, I could have registered operands; operands can be one or many. So, this is a one operand instruction, but actually there are implicit operands inside this I could have multiple operands and this operands can be of three types it can be a register operand it can be a memory operand it can also be a immediate operand. So, in the in the assembly language lab class we will see more of this instructions and operating operation modes. So, just for clever understanding now we need to understand; what are operands?

On the other side how do we how do we; so how do we fix these operands size of these operands. So, as you have seen in the Intel manual you would see 8 bit operand you see 16 bit operand; 32 bit operands etc. So, how do you fix the size of these operands?

Student: The op code

Op code will not tell you the size re. So, how do you fix the size of the operands?

Student: Data types.

Database;

Data types; so in a programming language there could be multiple data types for example, character is 8 byte 8 bits. So, 1 byte so, I will have one byte operands then we have we have lot of times where we need we use you know short variables right. So, we have that can also be 8 bits or 16 bits then I have long variable I have long-long variables. So, I can go up to 64 bits or 128 bits today. So, so depending on the size of the operands which are compiler wants; who supports data types who is responsible for understanding and you know and interpreting data types in your. So, I have operating I have hardware I have micro architecture then I have operating system then compiler etcetera who is responsible for interpreting.

The compiler is responsible for interpreting what is data type is right. So, if I have a 64 bit compiler then a normal int in c would be 64 bits there versus the same int for a 32 bit compiler would be a four byte. So, I may I can have inconsistencies in the in the way I am I am looking at int declaration int in c depending on which compiler I am using right.

So, to support the needs of the compiler the hardware needs to support the need support the different issues that are raise during compiling and one such issue is the length of the operands and that the hardware has to help. So, that is how operands sizes are been fixed right. So, Intel as it supports 8 bit operand 16 bit 32 bit 64 bit now even larger able to are you able to follow what I am trying to say. So, this is how I fix operands.

Student: Sir what about languages that do not explicitly define a type.

Finally the compiler has to interpret it your syntax is different you have a syntax right. So, you go and say that within that scope you can make a scope based type analysis correct like if you when you are studying your popular principles of programming language you will be learning hopefully you will be learning lot more other things. So, the scope what do you do you mean by a scope of a variable suppose I am assigning say a variable c to an integer automatically I will go. So, it is in the scope of an integer. So, I will go and declare c as an integer this is how many programming languages do interpret what is right suppose I go and say c equal to a plus b very interesting case c equal to a plus b.

Now, if a is floating point and I have not declared b I can go and say b is also floating point because this addition right there are 2 adders one adder which is a integer adder another adder is a floating point adder which could have several integer adders in internally, but a floating point adder addition is different from a integer addition. So, here itself I have to see whether both are one of them is floating point then I will make this f plus as floating. So, in when I interpret this I will just add if a and b are integers, but if a if one of them is a floating point then I need FADD this is Intel instructions.

So, the compiler actually sees the different variables there and based on those variables types of those variables it will come out with a conclusion of whether it is a floating point or integer. So, lot of context dependent analysis happen in weakly type languages where the types are not explicitly stated nor the language insist that the types need to be stated there are a lot of program programming you know benefits there are a lot of benefits for the programmer if the language is weakly typed right it is very strongly typed it makes a verification easy, but for a programmer it may not give you that much amount of benefits. So, what are weakly typed and strongly typed languages you will

learn more in your course your principles of programming languages course, but as far as we are concerned there are some data types and that dictate the size of the operands.

Now, how do you go about op codes how do you design the other I need add I need subtract I do not need this. So, how do you come out is there some systematic way suppose you are asked to design a computer tomorrow like we did design hack how did with arrive at the different op codes no why.

Who will give you the op codes you are designing an a l u depending on the op codes correct please understand this the hardware is still not available to you, you are not talking about interface you are not taking about the instruction set architecture after that only you are going to build an hardware right. So, who will decide what are the instructions that should be inside instruct or what is the process that you would like to follow when suppose you are asked to build architecture of what would be the instructions inside that. So, how do we arrive at an instruction set architecture this is the question are you getting what I am trying to say we basically go through lot of experimentation on this; this decision of what instructions that should be put inside an op code comes out of lot of empirical analysis empirical means experimental analysis.

So, you take lot of examples from the domain. So, your system is expected to serve in different domains it might be that I need to open browser I may have to do some e transactions I may have to support some multimedia facilities. So, your system and I have I have to do lot of scientific computation. So, every system will have certain role for which it is designed for right and from there you take all the applications in that role and understand what are the common functionalities that are executed by those operations by those case studies and those functionalities the common functionalities eventually translate into op codes right I will do lot of additions. So, add should be one op code op code I will do lot of bitwise manipulation.

So, bitwise and bitwise or and bitwise not they should be they should be part of your instruction set architecture. So, like that we go and look at different-different applications for which the system has to cater to and then depending upon the need of the application you start introducing instructions which shall match the need for the application. So, the instruction set architecture at least the op code what are the op codes that need to go in this is completely dictated by the application or the application domain for which the

processor is intended for suppose I am trying to make a network processor and there I go and put some multimedia instruction right it will be useless right what sort of instructions I will put in a network processor I will try and put certain packet processing instructions because a network processor which is going to be sitting inside a router is going to get packet it is going to process that packet and send it across or make some decisions right. So, when I am looking at an op code for a network processor I will put certain things related to packetization or related to retransmission there rather than then putting some multimedia instructions there.

But when I am looking at a home pc or a processor for your I-pad or this thing we will start looking more on the media instructions right multimedia instructions etcetera. So, there is a there is enough amount of work that has that need to be done there are lot lot more work that need to be done in identifying what are the op codes and for each of those op code what are the types of operands that op code can support and so how do you codify those operands all this things form basis of how do you develop an instruction set architecture.

So, in tomorrow's afternoon class, tomorrow morning I will continue on this, but tomorrow's afternoon class we will start looking at the CISC architecture of X86 that we have very good case studies for CISC while for the RISC we will do something with respect to arm processor fine.

Thanks.