

**Indian Institute of Technology  
Kanpur  
NP-TEL  
National Programme  
On  
Technology Enhanced Learning  
Course Title  
Compiler Design  
Lecture-20**

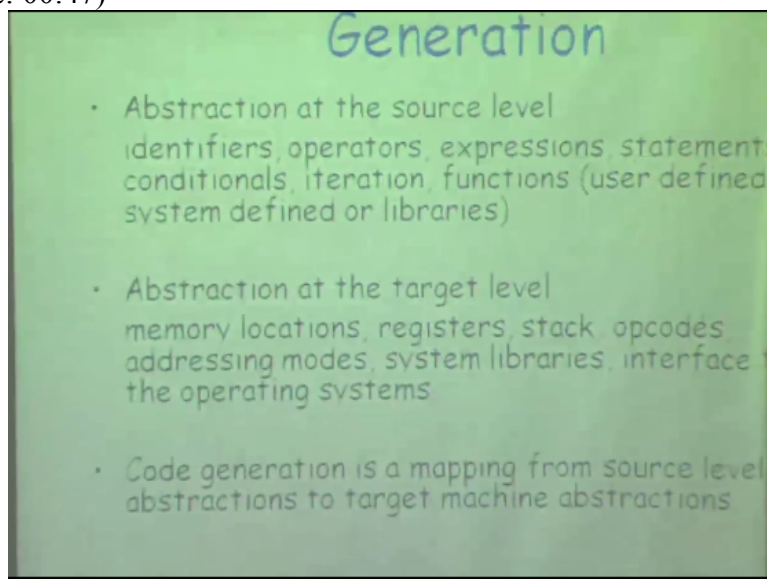
by...

**Prof. S.K. Aggarwal.**

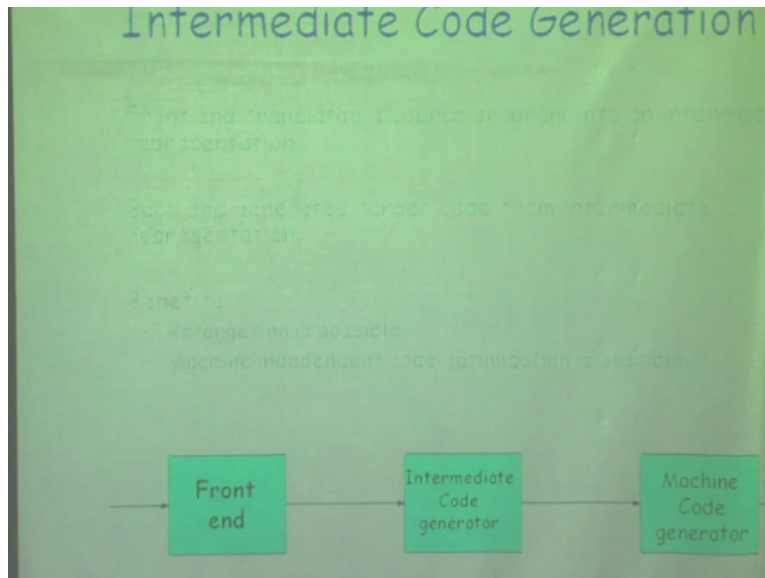
**Dept. of Computer Science and Engineering.**

So let us start over the discussion from the intermediate representation and we start looking at the end of the previous class was that how do we represent the intermediate representation and were looking at certain functionalities and certain properties of intermediate representation and the first functionality we looked at as there is an abstraction the source level and there is an abstraction at the target level and we want to make sure that these two abstractions match in the process of conversion .

(Refer Slide Time: 00:47)



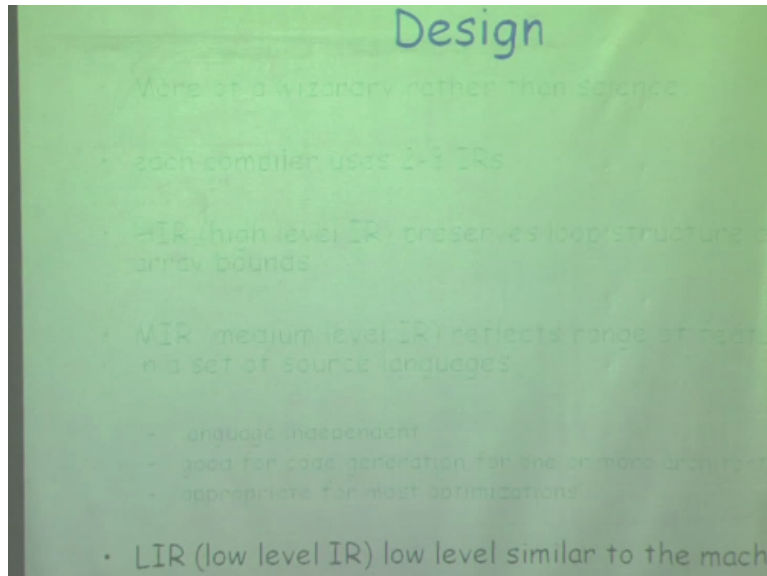
(Refer Slide Time: 01:05)



And the code generation is going to be that you have the frontend which you will not translate so what we should realize that we are talking about implementation so implementation is that I am doing type checking along with the code generation and then get the intermediate and then we convert it into the machine code and the backend is then will generate the machine code and benefits always we are that we are able to retarget the whole port and the machine independent code optimization is possible.

So we talk about both and the code optimization on IR we will also see how to write code optimization on IR and we will see how to write machine code specifications each specifications so that it target the final machine code and this is going to be the logical model that we have a front end and we have an intermediate code generator and the machine code generator and what we done so far is the front end and then we are going to talk about the intermediate code generation.

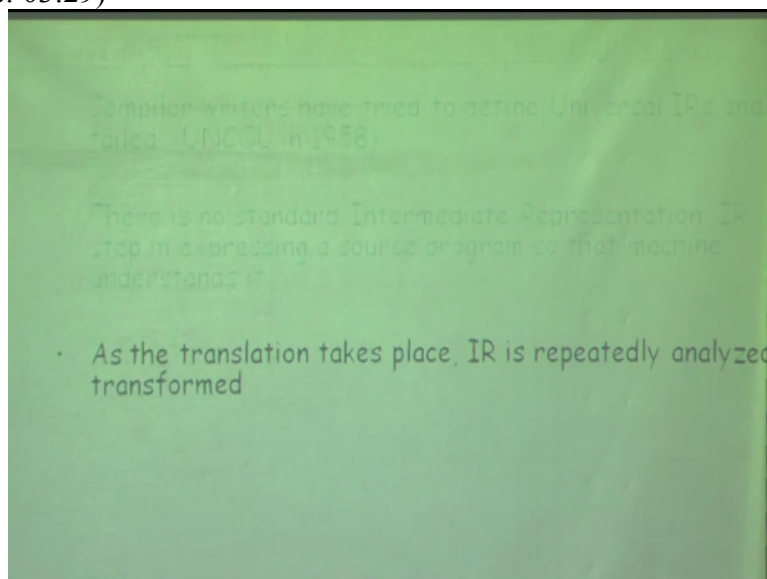
(Refer Slide Time: 02:05)



So this is going to be the overall structure of we will discuss here so we started discussing that how do we write an IR and turn down the IR does not seem to have a scientific basis but it has more of an engineering problem when we said that I want to design something and good designs can give you only the guidelines and so I will get a good IR so each compiler is going to use a set of representations and there are several intermediate representations .

We started looking at so we have high level intermediate representation and we can also talk about medium level intermediate representation you will each representation closer to machine but not really will be the machine concrete machines and so that name is going to be a low-level representation which is very similar to machines so again going back to the introduction.

(Refer Slide Time: 03:29)

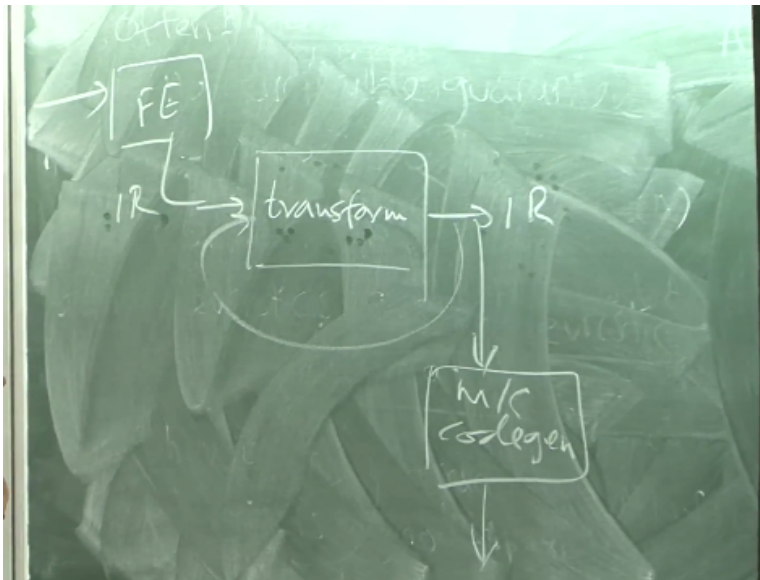


So at least what we say in that point of time there was a move very early in the history of compiler and thought that there should be some kind of intermediate representation and people designed them languages where you could take a set of languages and translate that into a set of machines and it was not possible to have some kind of IR which will be able to represent all the programming languages and generate code but now let us start looking at therefore what the property.

And so the first that happens is that so I want to continue IR so which is going to be some kind of intermediate code generation and I have to optimize somewhere and this optimize starts transformations and possibly and then the machine code generator is going to generate code from this and what we want is so there are several categories of user and the compiler or the people who write the compiler so what is that the man user do man user is not concerned with the IR okay.

They are not bothered about what goes inside the compiler for them the compiler is just a back bone so they are only looking at the two things so I should do a fast compilation and I should be able to correct it beyond that it does not matter what kind of representation you have so as far as the user compiler are concerned you are only looking at something which will be correct and fast beyond that it does not matter but the compiler are these the only things I am looking and do I need something which will be flexible between helping as many as possible it also helped me in doing transformations it will be easy to transform .

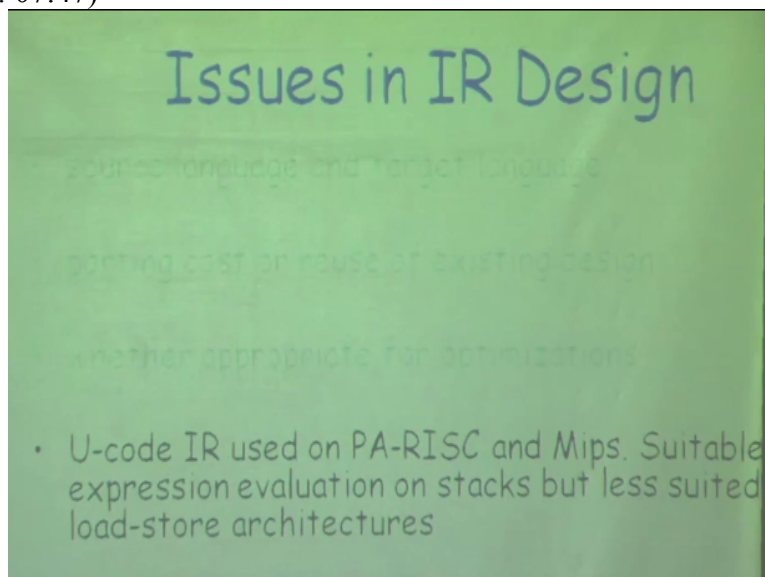
So kind of things as the compiler are writers want optimizations to be simple to be written and easy to understand and so what may happen is suppose at this point of time let us not worry about what an optimization is but what I am looking at is some kind of transformer .  
(Refer Slide Time: 06:22)



So if I have this transform which is looking at IR and this is giving me and this loop can this several times so what I want here is that continuously I will keep on changing my intermediate representation therefore on something which will be available to this kind of transformation so that is one thing we have this intermediate representation which is very difficult to understand then this phase becomes very complex.

So I want to make sure that this phase does not become too complex and therefore the representation I use is also simple and straightforward but that has an implication it should not happen that it becomes so simple that I am not able to represent my language and machines so IR should be simple it should not be too complex it should be lightweight it but it should also allow us and to express optimization and transformation .

(Refer Slide Time: 07:47)



So continuing on the issues in IR design obviously IR got is going to be impacted by the source so both source and target language are going to impact it that another thing that is going to happen is that the cost of porting that means time and when I plug in my machine podium detector and this is my input and then I want to really do this transformation and what I am looking at is what is the porting cost another is that reuse of this design .

We imagine a situation where I have an IR and I say that instead of that you have to redesign the whole IR means that my transformer will change and my front end will change front end which is generating this information that means to begin with this was not something which was good because it is exactly so many things.

So I want to make sure that all these issues are addressed in IR design and also whether this is appropriate for these transformations that remains a very important issue for us so this is the summary so here is a small example this example says is not may not be usable in the current machines but historically what is exposing is that this IR designed and how so there was at some point of time.

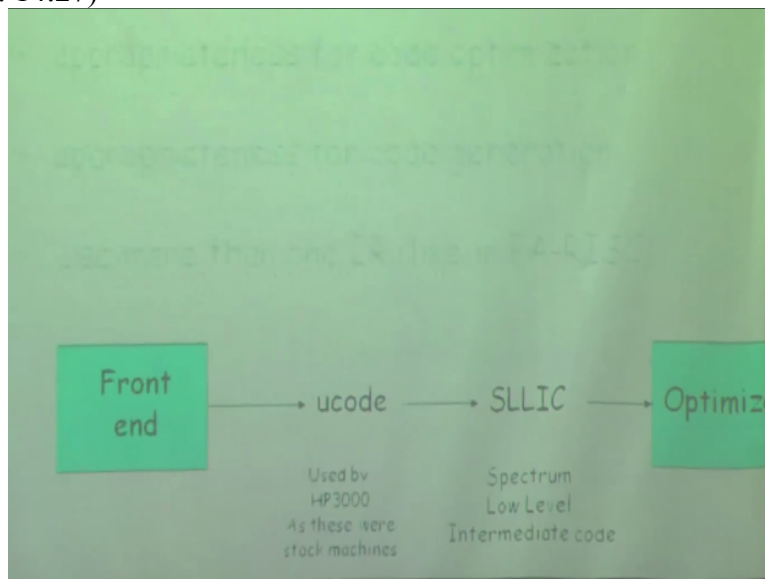
It was an IR code called U code and I will give you examples of many IRs so U code was based on PA-RISC so this is PA-RISC was an architecture which was used by HP and then it will also be this form may be produced by these are available and this stack machine was the first one used for compilation.

Okay so what happened was that some kind of stack machines which were used and basically stack machines they do not have a register architecture when I am going to take the offensive push this stack and then operate on the stack we are in fact that the stack machine was the first machine which was used for porting compiler for multiple parser it was not generating machine code for any target architecture but they had both coming out for a stack machine and just try to interpreter for stack machine is not difficult.

So this is intermediate to presentation which is not so suitable but it is suitable for stack machine so both compilers they translate they take U code and before they can really operate it on their machine something else so HP decided that they want to translate it into a very low level representation and MIPS decided that they will take this port and then translated back into some MIR and then translate back it into the U code generator for them which is a better approach.

Okay and which approach is good the first and second one and why the first one is not good approach very few machines so we can write compilation for very few by this approach but this approach seems not to be flexible but the observation is correct but the compilation is wrong but

suppose I want to write compilations for few machines but this so by looking at this you cannot arrive at a conclusion which one is better and your machines are chained .  
(Refer Slide Time: 14:27)



And so the issue in a new IR design is going to be one question we have two addresses how much of it is machine dependent like you saw that in the case of HP this was completely machine dependent in case mill it was not so if you say that my architecture line is really does not matter okay then I also want to see that how expressive this is that means how many languages can I cover with my IR how many different languages can I cover.

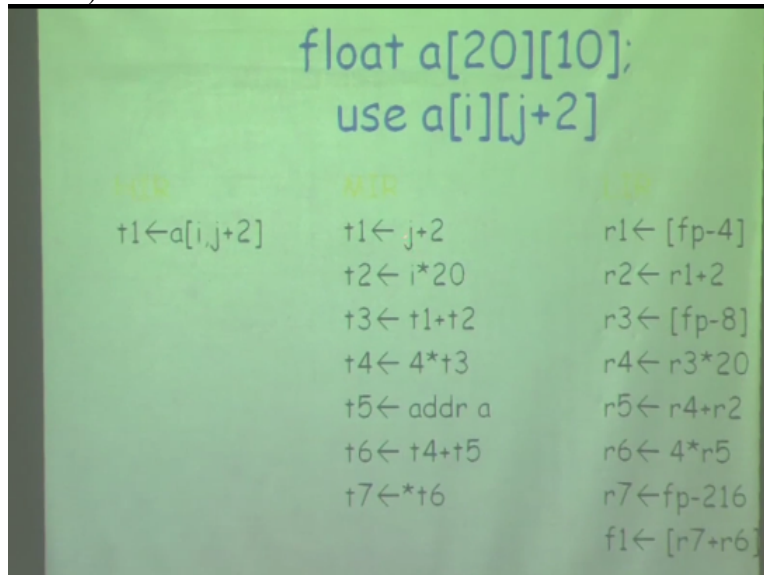
I suppose I say that my group is only language solution when I say that my solution for expressiveness there I am also making sure that this is appropriate for optimization and also obviously appropriate for code generations so you can see that it should be appropriate. I should be able to easily generate this I should be able to easily transform it and I should be able to easily convert this into a machine that is always something useful and therefore be used more than one IR like .

So in their front end they had this new code which was then converted into machine code and this what they were doing and this is my front end and in the front end they had u code and then some low-level intermediate representation and optimizing it was because it is used and so continuing on these discussions okay more than one IR for more than one optimization is possible now quite widely more than one IR for different optimizations who one IR is not used for optimization sleeping in the afternoon 9 o clock is to early Friday so suppose I am doing optimization for time .\

So I need more than one IR so I am not talking about optimization implementation will come later. So let me give you an example right there let us skip this and okay so here is an example so

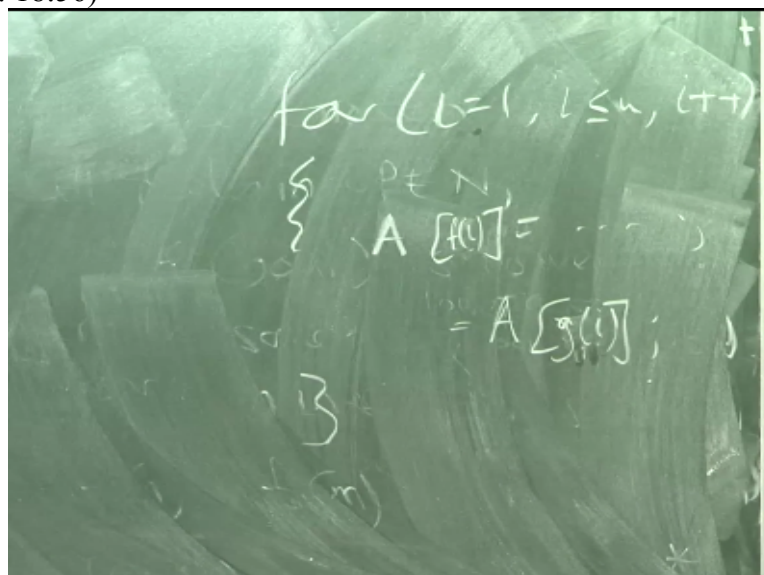
I have this instruction this declaration.

(Refer Slide Time: 17:59)



And I have an array everything and now I have an array is of the form it says t1 will assign a i j + 2 and I have an MIR which says that J + 2 will sign t1 and then I multiply that by 20 and then I add t1 and so I multiply by 20 and t1 and t2 then I multiply t 3 by 4 add some basic to this and write this kind code okay now what are the things I can do here which I cannot do here are there some transformations or some analysis it is possible would be of the first one and not on the second one .

(Refer Slide Time: 18:56)





So again let me write a small piece of code so suppose I write this piece of work say I is going from one end and then and the question I want to ask is for some value of  $i$  or let me write say some function of  $I$  and another function of  $I$  now want to ask a question which says that for what value of  $I$  in some iteration when you have  $F_i$  now suppose I use this intermediate representation or so which one is which representation will be better this or this to answer this question for what value of  $i$  in some iteration  $f_i$  will be equal to  $G$  first of all right .

So suppose I am doing optimizations where I need to do lot of analysis on the indices that I do not want to lose this information now what is this code for the same statement it is finding out the memory location where  $A[I][J + \mu]$  results in what it does is taking  $j + 2$  and it is assuming that dimension of  $I$  is 4 and is now counting number of rows and taking basically the offset from the base and this 4 is assuming that rows are being taken by each element so this gives me total offset from the base of  $a$  and that gives me the complete address and then I depress that.

Now suppose I want to do optimizations here which are at the level of statement and saying that on to then obviously this will be much better as compared to this so depending upon the optimizations I want to do and the kind of questions I want to ask my the presentation is going to be different so even when I am doing optimizations you may find the time to do some optimization so on one piece IR and there is yet another IR.

Which I am calling now low level and if you see here what it has done is it now says that  $J$  is some value and  $J$  must be stored somewhere and suppose this is stored at an offset of minus 4 from the frame pointer so this saying that I want value of  $J$  from this location and then I want to add 2 to this and  $I$  is stored at same point of minus 8 so I take this when you multiply that by 20 and so on right so this is very close to the machine.

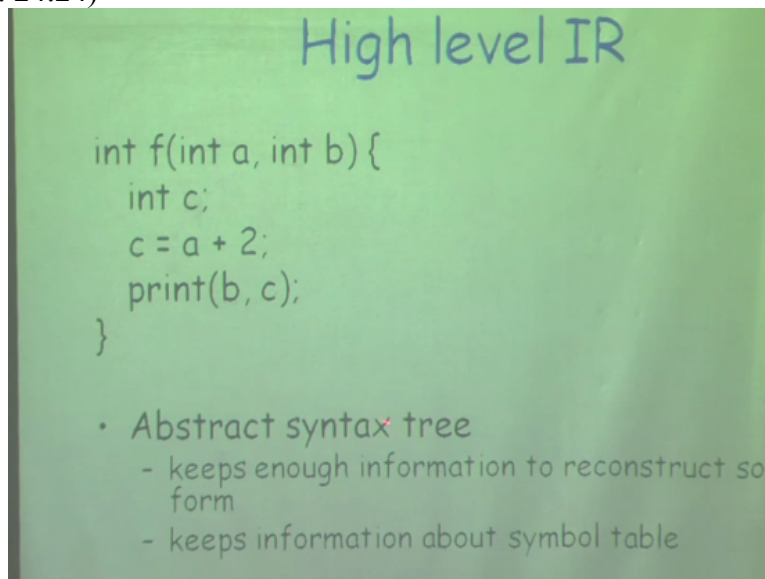
Now if I am doing only code optimization now you can see that this is actually exposed all the  $N$  this is remove so if I want to do now optimize you should set machine level then this is going to be very useful IR as compared to both this and this in this case you can see that addresses are not really available to me all I am looking at are certain so important point to remember is that there is nothing like some best IR and I need to have optimization sometimes so this represents our subscript by list of there is something missing there.

So HIR represents subscripts okay which is suitable for the kind of analysis which I just talked about which is dependence analysis and low-level IR is something which really making all that this is really exquisite and some kind of optimizations are very useful there so when we want to

do constant folding and code motion for the extension is very important and so on then this is the kind of IR which is going to be more important.

So we will talk about these optimizations but I do not want to have number of IRs so the way I look at the problem is that first I decide is that suitable IR that I want to do you want to fix a number of anything that IR orders so depending on the kind of things I am trying to earn you have many IRs what that means so I do not want to thank that - number one okay so this point already has been worked out .

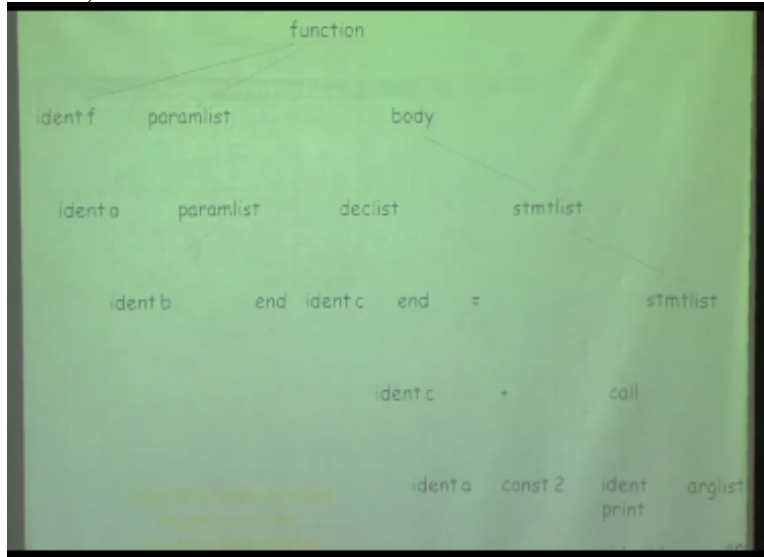
(Refer Slide Time: 24:24)



Okay so let us look at now various high level IR so let us take this kind of code end and this is something which you have frequently encountered so what I have is now a function f which is returning type integer it takes two arguments a and B both are type integer and then it has a local variable of type integer and then there is an expression that involves this argument and local variable is being a scientist value and that is just printing b and C now what are the various IRs we have seen abstract syntax tree is definitely one kind of representation and this is going to keep enough information.

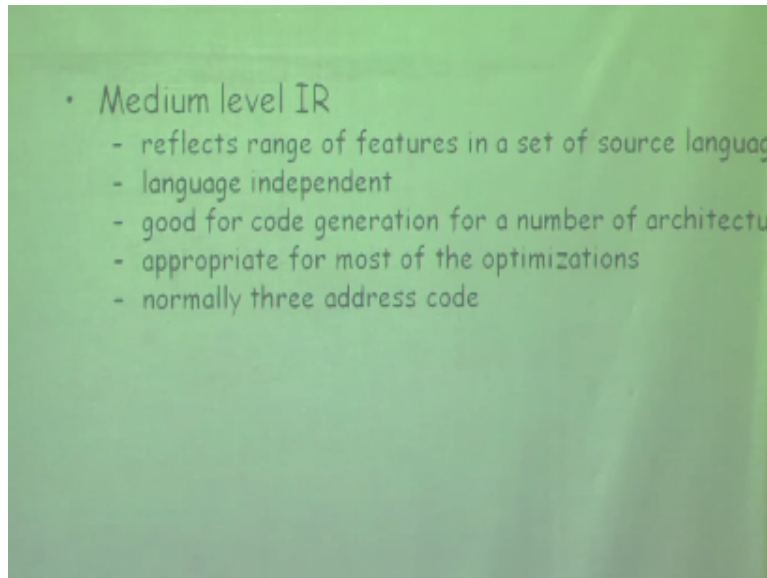
So that I can reconstruct the source okay so if I look at if I look at abstract syntax tree if I unparsed the abstract syntax tree I can get something which is very close to the source so many times when you are trying to write B assemblers than this kind of IR may be very good when you write de assemblers so if you are doing for example so some small stock solution you say that I want to convert my the C programs into parser I want to convert C++ program to C in so. That time you want to go all the way to something which is very low level high or and then type weakness at something it all be possible because and therefore I would like to keep my I R at

much higher level which is close to X X Index tree and all the information I mean like you put whatever is required for this symbol T so it may look something like this I will say that I have a function and the identifier is F here.  
 (Refer Slide Time: 25:52)

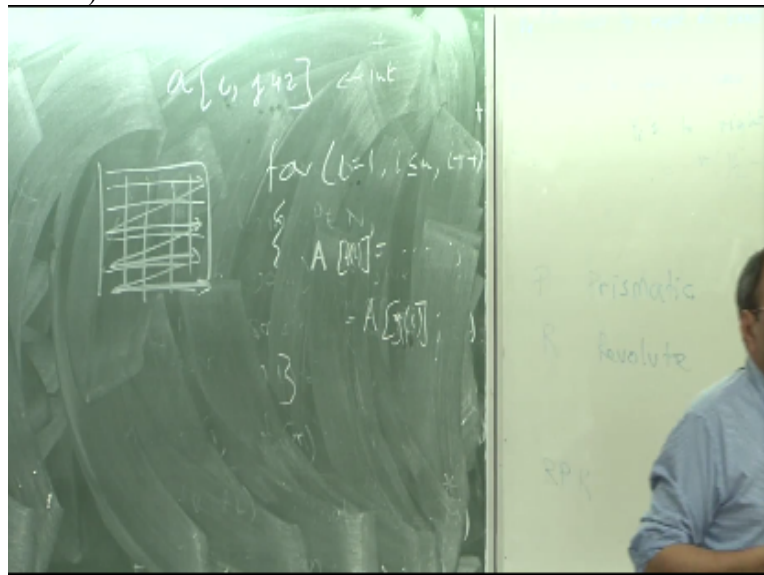


Then I have a list of parameter then I have a body and parameter list may be given again by a recursive definition it says that first argument is A and again the parameter placed another argument is B and end of it and then I have body and within body I have a declaration in statements in so you can see that in this hire okay I will be able to reconstruct almost which is something goes to source okay if I am taking to something like MIR or LIR and not fit to what source of.

This is something it is may be useful to representation for some reason today volunteer is misbehaving so even right side is getting cut and these colors are not coming out properly I will change it okay in this case what happens is that all the information about IUD participants read this symbol okay and where ever you see if I have basically all this is saying is that I just want to have here a link to the symbol table and get all the information and so it will the part of symbol.  
 (Refer Slide Time: 26:57)



Medium level IR is going to then have features of source language okay so again what you remember what we saw in the IR I was trying to translate this particular expression. (Refer Slide Time: 27:10)



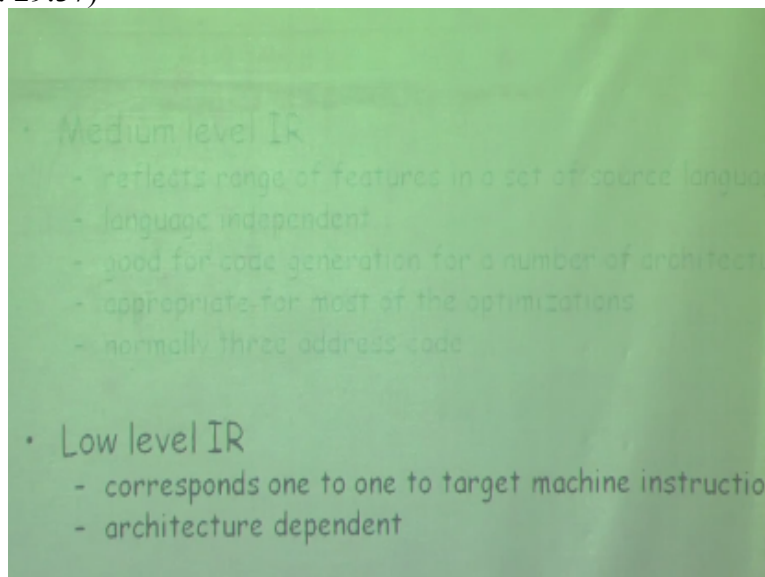
That I really went for not actual addresses of I and A but some symbolic name for IUD and I also recognize the fact that this is of I may be in the float and I know that there was the additional thing I need and what was that the goods ambition arrive and wrap down to single ambition okay now what is the order in which I am going to store so suppose this is I have that has certain rows and columns then it knows that how do i map this good dimension into single dimension . So what was the mapping I have used here row order mapping or row major mapping so I was going in this order okay so this is something I recognized that my language suppose this i also recognized the fact that each cell was taking four parts and depending upon the language it is

possible that instead of row major I may have column major mapping for example if you have a language like Fortran he has column major okay.

Again it also had information while that floating point was taking four bytes but on some other machine floating points would have been taken eight bytes so much higher had this information but it did not deal with the machine information that something about the implementation okay which a HIR never that okay so if i go back to this it has no information whatsoever about the machine this is something called abstract syntax.

So you can see that multiple I are surviving multiple information and they are exposing different information at different places so normally what we have at MIR is what we know is three address book now this is something which is good for optimization so large number of optimizations have written for this IR again see that and this is also good for code generation for a large number of architects so typically what is three address code and this is something we will be working with working with fourth generation so on so .

Three address code says that every statement will have a four sphere this it means I cannot use code and the argument and right hand side of the statement will have the born and only operator. (Refer Slide Time: 29:37)

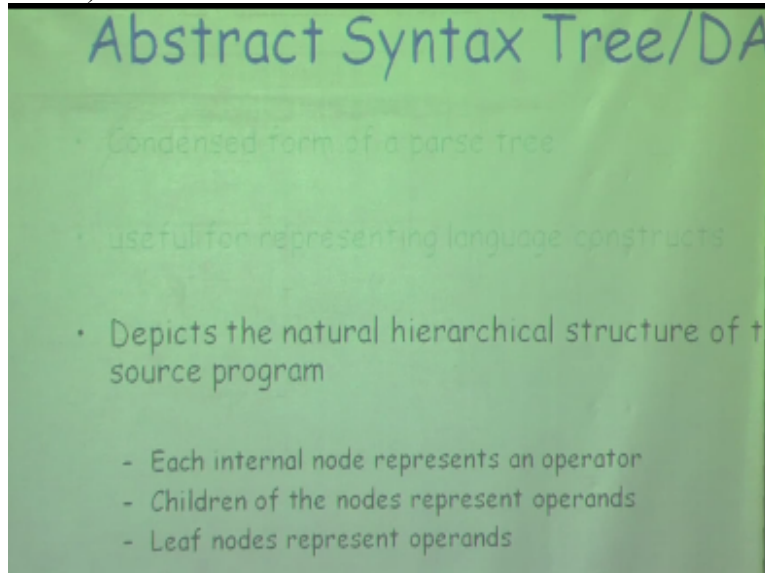


And low level IR is will correspond so we look at examples of this and low level IR have is going to correspond to more than one target architecture multi level IR that sometimes I want to mix all these there is nothing sacrosanct about saying what my ions whether it should be HIR or MIR okay when there is no reason to say that I will keep rule in the stack higher in HIR level but all other statement will get constant into MIR.

That is also positive so I will have an hybrid of all these so important thing and important take away from this is first you try to define what your functionality is and then you decide what is

the higher rather than saying that I will have an higher then I will say that let me now streak my functionality so that it is particular hire these two things cannot do in higher so an abstract syntax tree is one hire and this is sentence form of parse tree.

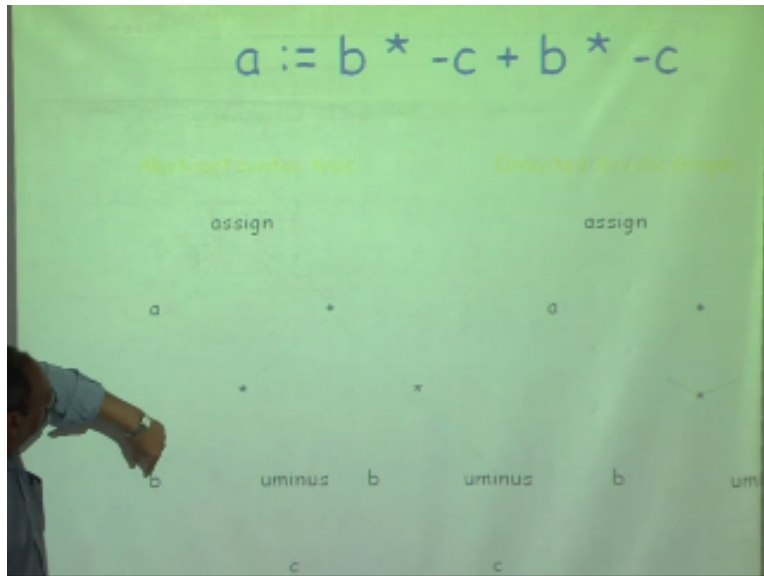
(Refer Slide Time: 30:34)



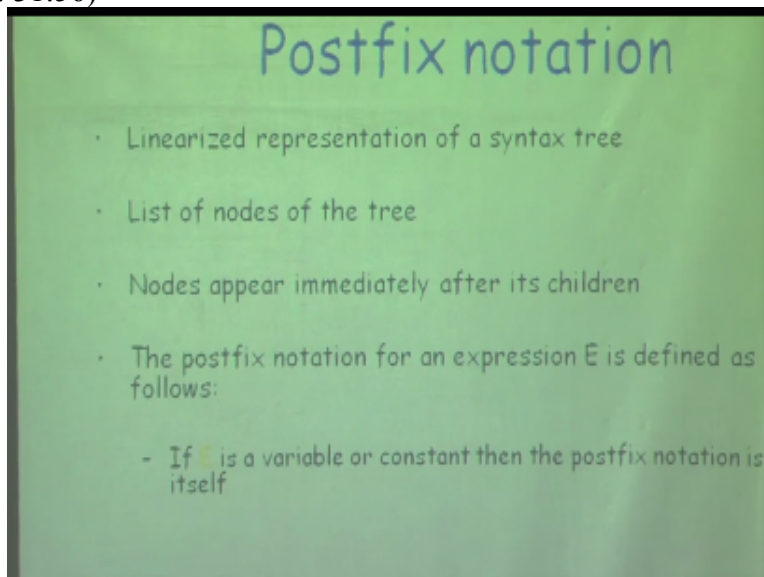
And useful for presenting language construct and this also gives you a very natural order of hierarchical if you remember the function free I showed you and it is very clearly and which offer your arguments are referring and in which order and statements are referring so it has a nice structure okay so the internal notes correspond to the operator assign to the children notes are going to be operated so on so the leaf notes are going obviously with the operator.

So this is one kind of hire and gang is again more compressed form of aspect syntax have been seen out the stack so there is an aspect syntax tree this expression tag for the same expression okay.

(Refer Slide Time: 31:18)



So what you have to remember here is that I have an X X syntax 3 here we have the root is at the sign which is this operator and the left hand side is the left hand side of this and right hand side is the expression because on into this which says it is an addition of two terms and each term is B multiplication of minus and similarly this part and when it comes to this part next what we have is like we know that since this part of the tree has been replicated here rather than replicating it I have one tree and then I am just having the as I swing from here.  
 (Refer Slide Time: 31:56)



So this is one kind of intermediate representation then another common representation is what we know as post exactly what happens is people envision define this in post order traversal of the syntax tree that will give the postal now I am posting may be useful what are the situation of postal this kind of postfix may be used stack based machine so if you have stack based

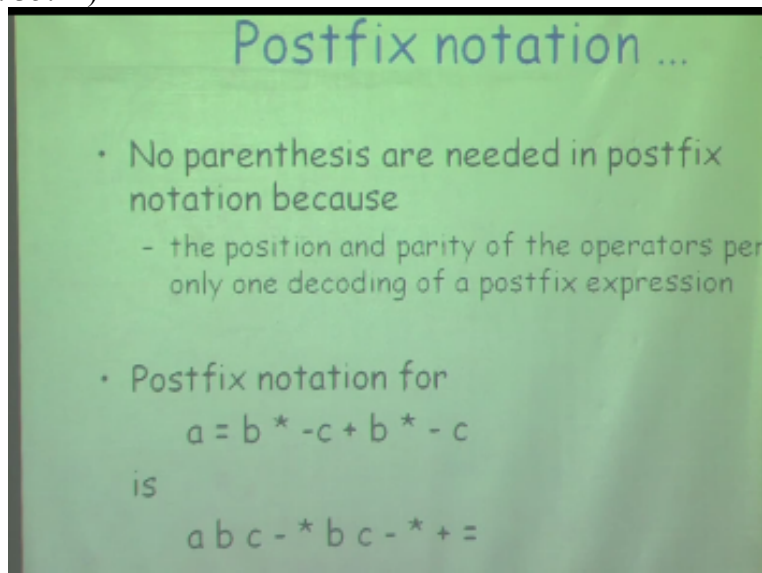
evaluation okay then your having the postfix mix normal sense I mean there is no point in having a tree.

Because once you convert it to postfix all we are saying is that all your operators will be on the stack and whenever you get no friend your operators will be on the top of stack of them oh do the operation so that postfix is going to be very useful so you could have just the list of nodes of the tree and nodes are going to appear immediately after it is so if I now say that I want to convert in expression E into postfix how will I do that yeah.

If it I s just available then what is the postfix notation the variable itself but suppose I say that I have E which is of the form even of into what will be post fix of this E1 and E2 is it correct yes no recursive definitions that even also has to be converted into its own clothes how do you know that E1not converted this so if I write something like this is this the correct transcription is so then it has to be even if I am in Ukraine where even tiny spare part even a small things you have to remember do not say that you never know what you are going to need to work these are the special so it is an expression right okay.

So these are the expression of this form it says even E1 and E2 then post pictures this is going to be even prime even prime check post of economy 2 and this is even time you talk and if even is an expression of foam which is graduated expression and specifics of this of this form if I say what will be four sticks of this white ones diets back inside just to make sure that I am a client right posting to seeing in the brackets so what will co fix of the brackets especially this were close fix of it right so that is going to give me just okay.

(Refer Slide Time: 35:41)

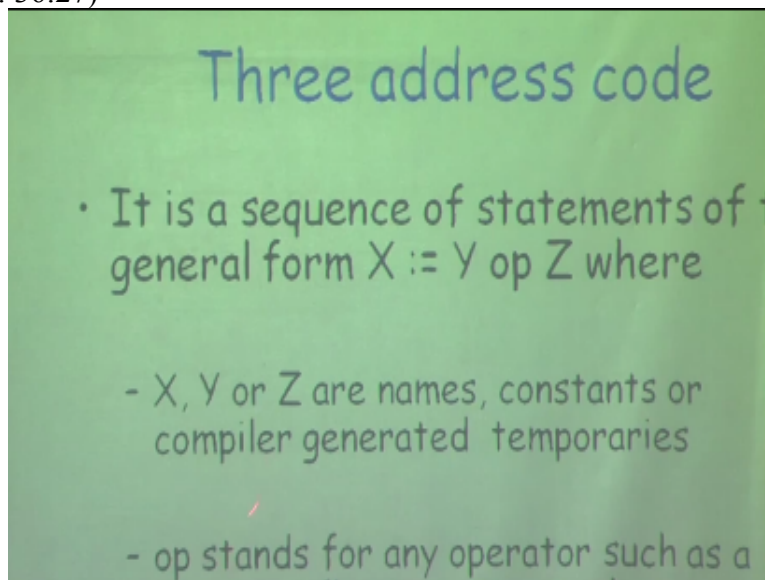


Postfix notation ...

- No parenthesis are needed in postfix notation because
  - the position and parity of the operators per only one decoding of a postfix expression
- Postfix notation for
$$a = b * -c + b * -c$$
is
$$a b c - * b c - * + =$$



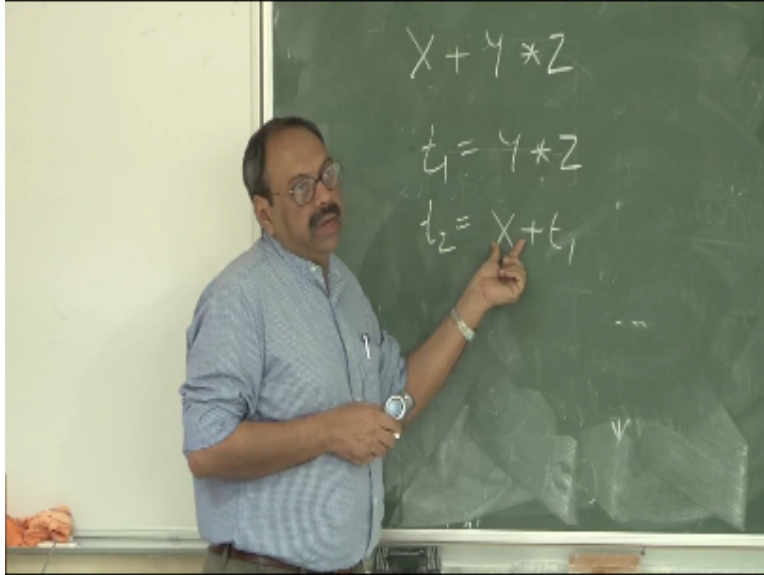
So continue on post fix notation you do not require any parentheses because postfix notation excesses will capture all associated and postfix notation therefore for this kind of expression is going to be where I will say that even assignment is an operation and then you can see that this becomes very suitable for machinery already blue air is that I am going to push A I am going to push B then I am going to push T and then I will say now I have unary negation in therefore of this parse tree value and negate the value and then push the right one this time and then I will say star, star now say take B and minus C and multiplication N so by the time I finish this well.  
 (Refer Slide Time: 36:27)



What is three address code three address code is this is going to be something which we are going to use for all the example we have because this is something which is close to most of the machine architecture and three address code is where all five instructions are going to be of the form  $X := Y \text{ op } Z$  which is the most general case everything else will be a subset of this and what that means is that I have at most three addresses on the right hand side I have so XY and Z are some names so these need not be variables these are either variables.

Which are return in the program or this could be internally generated by the compiler there and they could also be constants and off stands for any operator in this case this is a binary operator per suppose I have a unity operator the transcription I will be writing I just X is 5 minus 1 and if I low operating that I will just say if I just have a coffee instruction.  
 So I can have three address port of this form there is only one operator on the right hand side and if I now say that I want to generate free address code for an expression which is X plus y star Z this what kind of expression I will have on this.

(Refer Slide Time: 37:55)

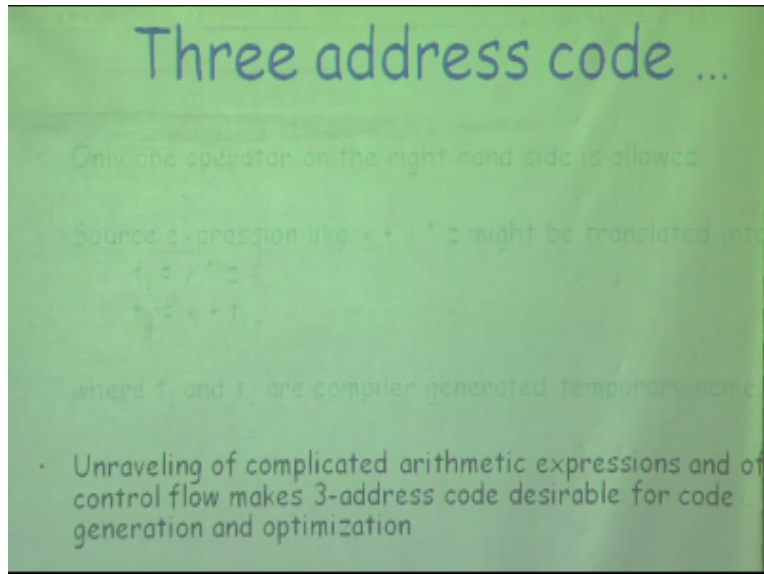


So if I want to say  $X$  or the three address code I am going to write for this  $T$  is equal to  $Y * Z$   $R$  is equal to  $X$  plus  $T$  so normally what will go is that will use only  $T$  and subscribe with different numbers so these are all temporarily which have been generated internally by the compiler user variables and therefore I may use something which is little more informative but clearly what you have pointed out is correct that is for the set  $T$  one is by stars and  $T_2$  is  $X$  plus  $T_1$  and so okay.

So I will write this kind of code yeah and  $T_1$  and  $T_2$  are some finite element that really makes no immediately at this point of time what should bring advantage is that when I say that I am now trying to convert an expression like this into a three address sport okay you can really see that what I am doing here at machine level so this is not really machine code but at machine level what is it I can doing what kind of code I am enter generating for  $T_1$  being assigned by star  $Z$  machine level so I say load  $Y$  load  $Z$  multiply the two men stole value in register  $R_1$  okay. And then I say load  $X$  and add  $R_1$  and whatever location I loaded  $X$  and store that into  $r_2$  now also what I could have done if you from the point of view of optimization if I find that none of them are in registers then I say load  $Y$  into  $R_1$  loads  $Z$  into  $R_2$  multiply the two and store result into  $R_1$  and then  $R_1$  already has  $T_1$  that I will say load  $X$  into  $R_2$  and the two and store is up into  $R_1$  so two the list of able to do all the communication okay.

So if I can remember all this information this, this will get exposed when we really do machine code generation from the three of this code that how are we to manage my registers okay so three address code you can see is something which is very close to most of the machine instructions kind of float to the architectural which has finished okay.

(Refer Slide Time: 40:25)



So this also takes all the complex expressions like so I will not call it very complex straight forward expression but you can see that even if you have complex and expression okay then I am going to almost take this and convert that into a sequence of three address codes so if you recall I took this and convert this into sequence of sphere so what did I do I took first index the second one and the two to that row bit multiple that to the first one and found out the off set of IED plus 2 then you multiply the whole thing by four the base address of a added it that, that really gave me the full computation of A, A plus right.

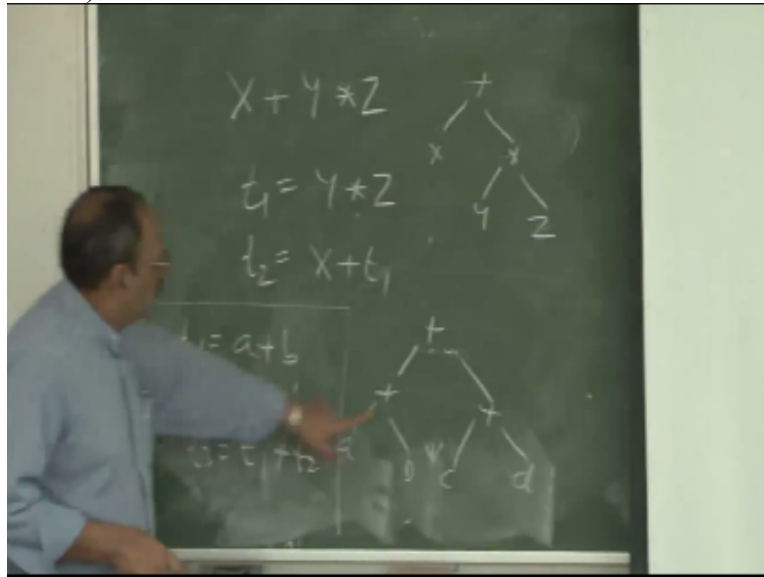
So this way I can expose all the address information and the use of name immediately allows it to be easily really so for example if I had so take an abstract syntax tree for this expect syntax tree for this will look something like this right now suppose I am hearing this okay I want to reorder this here is one computation here is another computation okay now suppose there was a situation where I say that I want to switch order of computation here so take a situation like this where I say my expect syntax tree is and now I say that normally I am going to go from left to right.

And therefore I say that this is going to be evaluate first then I will evaluate this and then I let the two now suppose I did not have any side effects in evaluation of ABC here are straight forward I just loading or unloading this information does it matter in which order I compute this whether I can do this or I compute this no but now suppose I want to capture this information here and say that now instead of this computation do this computation what will I have to do on the tree.

I have transformation in the transformation on tree can be very complex depending upon the kind of operator as opposed to that imagine that I have a three address code how the three address code look so three address course will look something like this T1 is a sign A plus B and T2 is

sign C plus B and then I may say T3 sign T1 plus T2 and now want to do the same transformation what would I do I will just reorder that I say just take it here this structural transformation is going to be lot easier on three address code as prepare to aspect syntax trees okay.

(Refer Slide Time: 43:15)



So use of names and intermediate values this allows three address code could be easily rearrange basically if you see T1 and T2 here T1 and T2 really correspond to notes of these subjects okay so rather than using this kind of tree if I start using names explicitly then this rearrangement becomes easy point because all these are going to be required at the time of coordination and cooperation of these so three of this code is nothing but if I just linearized this and say that whenever I am line arising for every node every sub tree I introduce a temporary name than under.

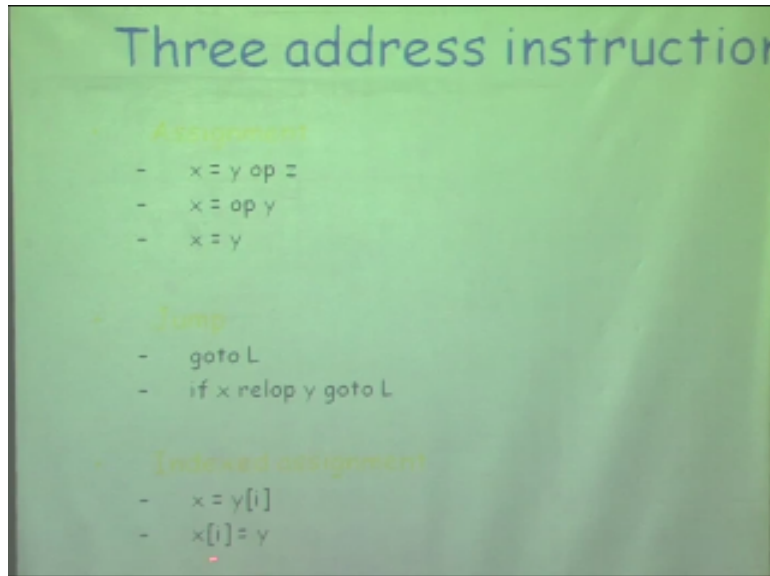
So in fact another way to write three address code may be that I just linearize it and I instead of using sensory names I start using many apoptosis so I can say that I am going to now put holes into this forms you to say that I will have plus A B and this instruction is going to be some address A1 and then I can also write something like saying that I have plus C and B and this is address A2 and then I say that address A3 is nothing but plus A1 and A2 so this is another notation which is commonly used and therefore what will happen is that these are nothing but really point to say.

So instead of using temperate names I can also use addresses so various kind of address are going to be mistake okay so three address code is nothing but some kind of linearize representation of at sec syntax tree where for each sub node I have put an expressive name or

expressive address rather than just keeping some kind of pre stitch so continuing of on this we have this instructions I may have a set of assignments so assignments will look something like X being assigned by your Z or actually be a sign of Y or X may be assigned point. So this really captures all possible combinations of binary operators unary operators straight away operators right and I can also have jumps which are either unconditional jump which says jump well what is address here you have an address to go to L then we said that I can have an address here which is X and Y and Z what an address here go to L later they had address this when I say go to L, L is a label you can address yes no if you have return assembly language programme when you say jump label what, what does it mean is that change your PC value for whatever is the current PC value to the new PC value near the instruction with label added scope. (Refer Slide Time: 46:27)

|   |   |
|---|---|
| <ul style="list-style-type: none"> <li>• Assignment           <ul style="list-style-type: none"> <li>- <math>x = y \text{ op } z</math></li> <li>- <math>x = \text{op } y</math></li> <li>- <math>x = y</math></li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>• Function           <ul style="list-style-type: none"> <li>- param x</li> <li>- call p, n</li> <li>- return y</li> </ul> </li> </ul>                                      |
| <ul style="list-style-type: none"> <li>• Jump           <ul style="list-style-type: none"> <li>- goto L</li> <li>- if x relop y goto L</li> </ul> </li> </ul>   | <ul style="list-style-type: none"> <li>• Pointer           <ul style="list-style-type: none"> <li>- <math>x = \&amp;y</math></li> <li>- <math>x = *y</math></li> <li>- <math>*x = y</math></li> </ul> </li> </ul> |
| <ul style="list-style-type: none"> <li>• Indexed assignment</li> </ul>  |   |

So L is the address here and this says if X L of Y and this is some relational operation so X and Y are not two operators and jump well so there is a conditional jump okay so this is that if this is true then jump to this if it is false and what I do can have a statement which says if X of X well of y equal to L1 else that kind of instruction so that will be four addresses I use a fall to mechanism therefore that if this fails I will just fall to an existence right. (Refer Slide Time: 47:05)



And similarly I may have for indexing so I may say X is sine Y I and X I may be fine right then I can write XD sign Y I plus okay so what are the three addresses here in this XY and I X and Y are the base address of X is the address of X Y is the base address and I is the offset these are the three addresses I have okay and what is the operator on the right hand side we have an operator here so operator here is I am just the base address plus offset that is the operator okay.

So similarly this is our three addresses but operator is on similarly I can have for argument passing I may have saying that I can have a parameter and then I can have a list of parameters and then I say now jump to procedure P with n parameters and this is return control to from were ever I take or return control or I can have pointers I can say that I can be reference point or I can take a variable thing it is called first and so on so you can see that this kind of intermediate representations I can generate code for intermediate for a large class of languages looks fairly small and straightforward.

But you can see that this itself can be converted into something which is closer to machine code for large pass of language what we need to do now is we need to start from program and program is going to have several parts of program is going to have declarations trying to process those declarations input information in simple tables and so you have loops and is going to have conditionals and is going to have state bank code I need to convert all that into three address code and that is going to have functions and procedures.

I need to convert all my functions and procedures into a sequence of instructions like this plus I stack and E so what we are going to do in the next classes we will see that how do I start taking parts of the code or parts of my program and start converting that either symbol table or code operating okay so let us start with the next class let us take a break here today.

**Acknowledgment**  
**Ministry of Human Resources & Development**

Prof. Phalguni Gupta

**Co-ordinator, NPTEL IIT Kanpur**

Satyaki Roy

**Co Co-ordinator, NPTEL IIT Kanpur**

**Camera**

Ram Chandra

Dilip Tripathi

Padam Shukla

Manoj Shrivastava

Sanjay Mishra

**Editing**

Ashish Singh

Badal Pradhan

Tapobrata Das

Shuubham Rawat

Shikha Gupta

Pradeep Kumar

K.K Mishra

Jai Singh

Sweety Kanaujia

Aradhana Singh

Sweta

Preeti Sachan

Ashutosh Gairola

Dilip Katiyar

Ashutosh Kumar

**Light& Sound**

Sharwan

Hari Ram

**Production Crew**

Bhadra Rao

Puneet Kumar Bajpai

Priyanka Singh

**Office**

Lalty Dutta

Ajay Kanaujia

Shivendra Kumar Tiwari

Saurabh Shukla

**Direction**

Sanjay Pal

**Production Manager**

Bharat Lals

**an IIT Kanpur Production**

**@Copyright reserved**

