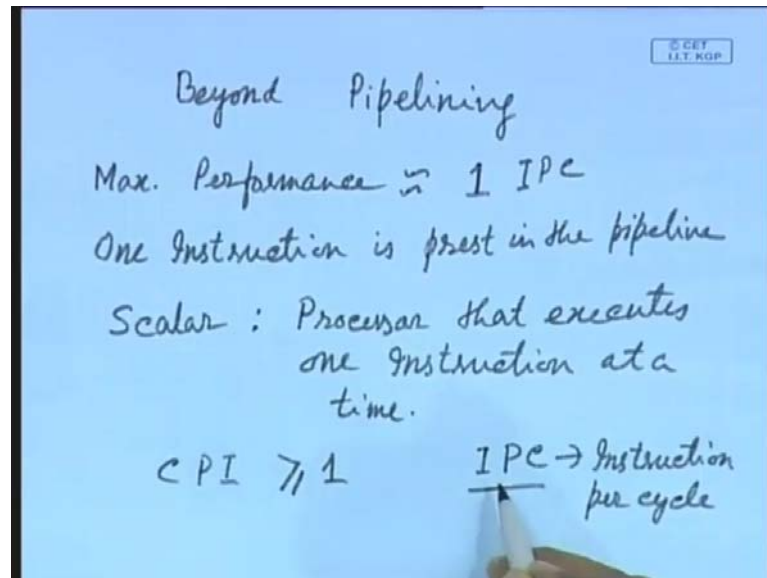


**High Performance Computer Architecture**  
**Prof. Ajit Pal**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

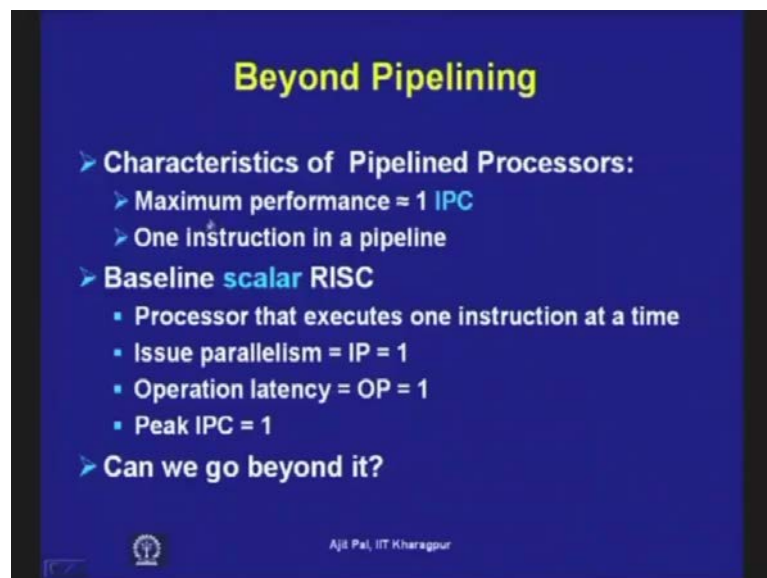
**Lecture - 11**  
**In Quest of Higher ILP**

Hello, and welcome to today's lecture on in quest of higher ILP; ILP stands for instruction level parallelism. In the last couple of lectures, we have discussed how pipelining can be implemented and how instruction level parallelism is exploited in implementing pipeline.

(Refer Slide Time: 01:25)



(Refer Slide Time: 01:37)



So, today we would like to go Beyond pipeline, we have already seen the characteristics of pipelined processors, in case of pipelined processors, we have seen maximum performance that you can achieve is close to one instruction per cycle. That means even if there is no stall, no hazard then we cannot achieve beyond this, you can this is the upper limit of instruction level parallelism that you can achieve. Second thing is one instruction is present in the pipeline at a time. That means although we shall be executing different instructions I mean different parts of the instruction in a overlapped manner. But at a particular instant of time if you look at we will find that only one instruction is

in the pipeline. And we have discussed about the baseline scalar RISC here a term scalar is introduced. So, this scalar is introduced to specify that you have got processor that executes one instruction at a time.

And later on, we shall discuss about super scalar where you will see more than one instruction can be executed at a time. So, the main characteristics of these baseline scalar RISC processors are issue parallelism is only one. That means at a time only one instruction is issued operational latency is equal to 1 that was assumed that the latency is 1. And peak instruction per cycle you see earlier we were using a term called CPI cycle per instruction that was obviously was greater than 1 in case of your greater than equal to 1 in case of pipeline. But now we are trying to introduce another term which is the just the reverse of CPI here instruction per cycle why you are doing this the instruction per cycle is a that number will be more than 1. So, since we want CPI less than 1, we want instead of CPI we shall using the terminology ICP which will be more than 1 in the processors that we shall be discussing after this.

So, in up to the pipeline processor, we were discussing CPI because the number of instructions cycle per instruction that was more than 1. So, it is just a terminology CPI and IPC they represent the same thing one is the inverse of the other that is all. Now, the question is can we go beyond it so pipeline processors we have explored in details. And we have seen, what are the different types of hazards? Structural hazard, data hazard and control hazard and how these hazards can be overcome by using suitable techniques. We have used loop unrolling; we have used software pipeline to overcome data hazard. We used enough additional resources to overcome structural hazard. Of course, we have not discussed in detail about controlled hazard that we shall discuss subsequently. Now, how can we go beyond pipeline?

(Refer Slide Time: 06:23)

**Limitations of Scalar Pipelines**

- Maximum throughput bounded by one instruction per cycle.
- Inefficient unification of instructions into one pipeline:
  - ALU, MEM stages very diverse e.g.: FP
- Rigid nature of in-order pipeline:
  - If a leading instruction is stalled every subsequent instruction is stalled

Ajit Pal, IIT Kharagpur

And these limitations, we have highlighted several times. So, in case of scalar pipelines maximum throughput is bounded by one instruction per cycle. Inefficient unification of instructions into one pipeline, you try to understand this particular point. The instructions as you know can be categorized into different types, we know that the instructions are number one is data transfer.

(Refer Slide Time: 06:57)

© CEE  
IIT KGP

Data transfer  $\Rightarrow$  Load / Store  
Data manipulation  $\Rightarrow$  ALU Operation  
Control instruction (transfer) using registers  
Status Manipulation

--	--	--	--	--

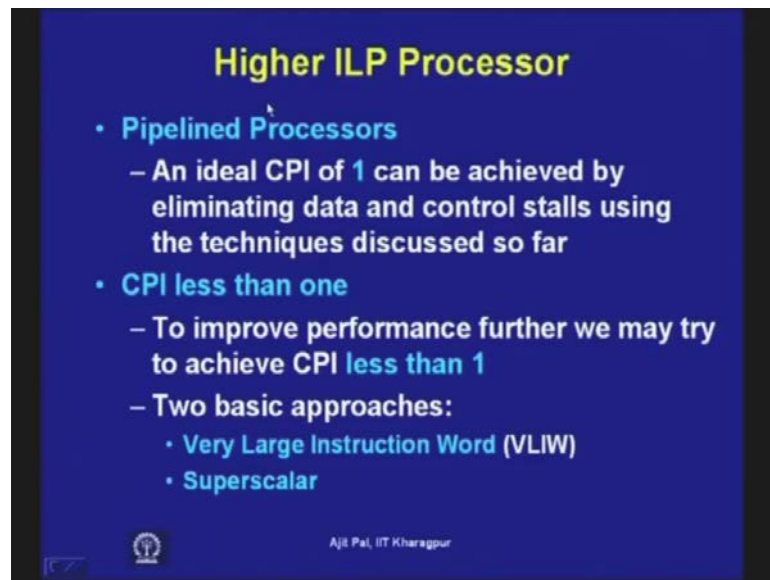
Then data manipulation and there are several control instructions or these are also known as control transfer. And there are few instructions which are known as status

manipulation instructions. In spite of the fact, we know that these instructions do not take same time for execution. For example, data transfer instructions in RISC those are load and store. So, these instructions will involve memory, in spite of the fact the instructions involve memory will take longer time; we group them with data manipulation instructions which are where the ALU operations are performed using registers obviously to get operands from the register takes smaller time. And they will take smaller lesser time to execute, but in our pipeline whenever we considered, we consider a single pipeline having different stages instruction phases, instruction decode, memory, execution memory and write back.

So, using the same pipeline we were trying to execute different types of instructions; obviously, it is little not only little unrealistic in fact is it can be really implemented in this phase. So, this inefficient unification of instruction into one pipeline ALU operation memory stages with diverse which are very diverse. Then floating point operations particularly, we see whenever we try to implement floating point operations in a processor they are quite complex. And they will involve several cycles and obviously, if we try unify them with the help of a single pipeline system. Then it is not feasible, but we assumed some assumptions were made with considered that they will take each stage will take only one cycle.

So, with the based on that assumption we we have implemented pipelining. And another very important limitation of scalar pipelines were rigid nature of in order pipeline. You know we are one instructions is getting entered until that instruction is execution of that instruction is completed, we cannot issue another instruction. So, as a consequence what is happening at a if any instruction is stalled then subsequent, the subsequent instructions cannot be issued or executed. That means because of this in order nature of instruction issue which is done one instruction per cycle this limitation is coming. So, if a leading instruction is stalled every subsequent instruction is stalled. So, these are the limitations of scalar pipeline and obviously our objective of this lecture that is beyond pipelining would like to overcome these limitations.

(Refer Slide Time: 11:16)



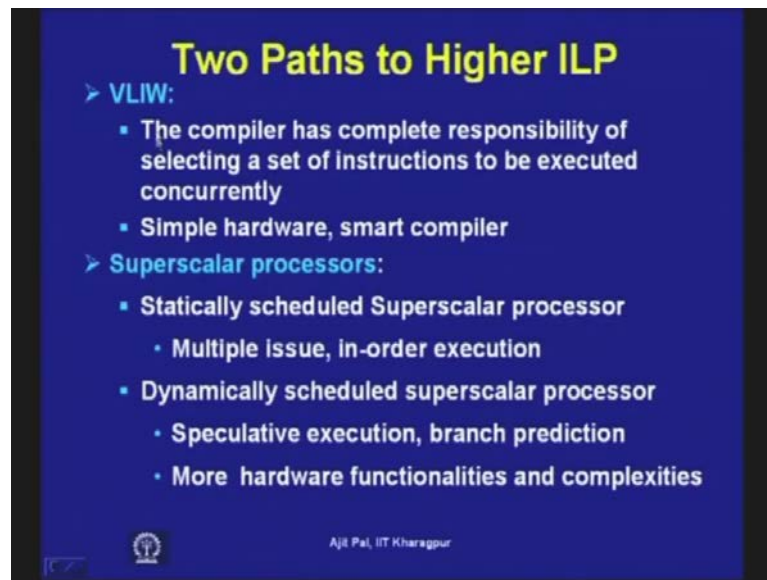
**Higher ILP Processor**

- **Pipelined Processors**
  - An ideal CPI of 1 can be achieved by eliminating data and control stalls using the techniques discussed so far
- **CPI less than one**
  - To improve performance further we may try to achieve CPI less than 1
  - Two basic approaches:
    - **Very Large Instruction Word (VLIW)**
    - **Superscalar**

Ajit Pal, IIT Kharagpur

So, here the main objective is to have higher ILP processor so an ideal CPI of 1 can be achieved by eliminating data and control hazards that we have seen. So, CPI less than 1 or IPC greater than 1, whatever way you will say we would like to improve the performance further and we may try to achieve CPI less than 1 or IPC greater than 1. And in this connection 2 basic approaches have emerged, first one is known as very large, large instruction word VLIW and second approach is known as superscalar. So, first approach that VLIW, we shall discuss today and superscalar processors we shall discuss later.

(Refer Slide Time: 12:12)

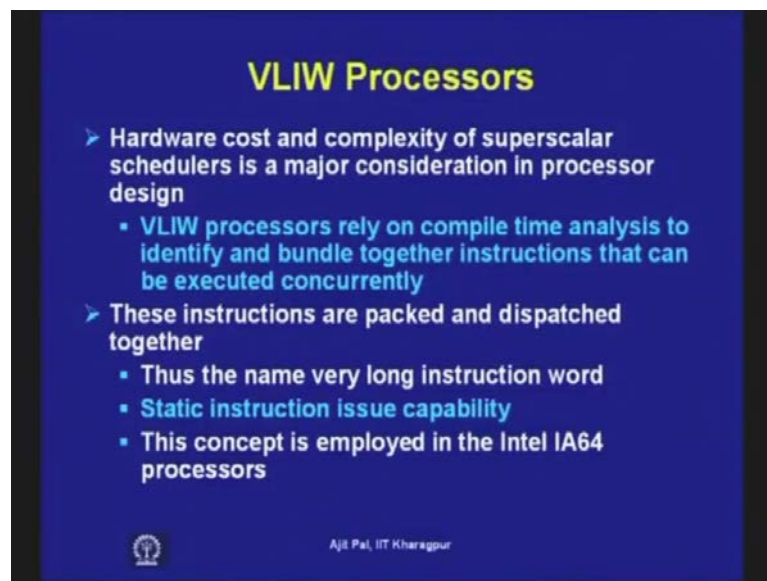


And just let me highlight the key differences between these 2 approaches before I discuss in more detailed about VLIW. So, VLIW in case of VLIW processor, the compiler has complete responsibility of selecting a set of instructions to be executed concurrently. Obviously, if we want CPI less than or IPC greater than 1, we have to concurrently execute more than 1 instruction question naturally arises who will identify those instructions who have which can be executed concurrently. And the primary requirement is that they should be independent so that can be that difference leads to 2 different approaches. The first one that VLIW; in this case the compiler is given the complete responsibility of selecting a set of instructions to be executed concurrently. And obvious consequence is that hardware will be simple.

But it will require a smart compiler that means the complexity is passed on to the software designer who are developing the compiler instead of passing on the complexity to the hardware designer who are implementing the processor. So, in this case the compiler will be complex the hardware will be simple in case of VLIW. On the other hand in case of superscalar architecture you will find that the instructions which can be issued or which can be executed concurrently is identified by the help a hardware not by software. And there are 2 basic approaches; one is known as statically scheduled superscalar processor. So, in this case multiple instructions are issued and then they are executed in order.

So, statically scheduled superscalar processors will perform multiple issue of instructions. And that will be done by hardware and in order execution of an instruction will take place. On the other hand the dynamically scheduled superscalar processors where sophisticated techniques like speculative execution branch prediction those things will be done. And in such cases you will find it will be, it will allow out of order execution. That means instructions will not only I mean they will be issued out of order and execution will take place out of order.

(Refer Slide Time: 15:55)



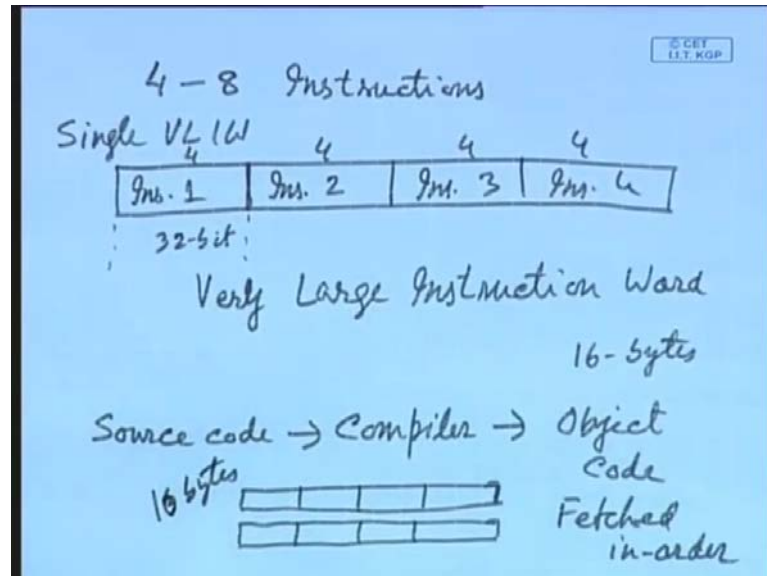
So, that will be done by by incorporating these techniques like speculative execution branch prediction and so on. So, in this particular case; obviously, in case of superscalar processor you will require more hardware functionalities and complexities. And we shall see the processor will be very complex they will consume lot of power and compared to VLIW processors. So, you may be asking that why VLIW processors are important the reason for that is hardware cost and complexity of super scalars is a major consideration in processor in design. Because hardware is very complex they will require lot of chip area they will that means the complexity will be very high that that and the power dissipation will be very high.

And to overcome that VLIW processors will rely on compile time analysis to identify and bundle together instructions that can be executed concurrently. So, the compiler will do analysis on the instructions and identify which instructions are independent. And then



it will bundle together instructions that can be executed concurrently and these instructions are packed and dispatched together.

(Refer Slide Time: 17:17)



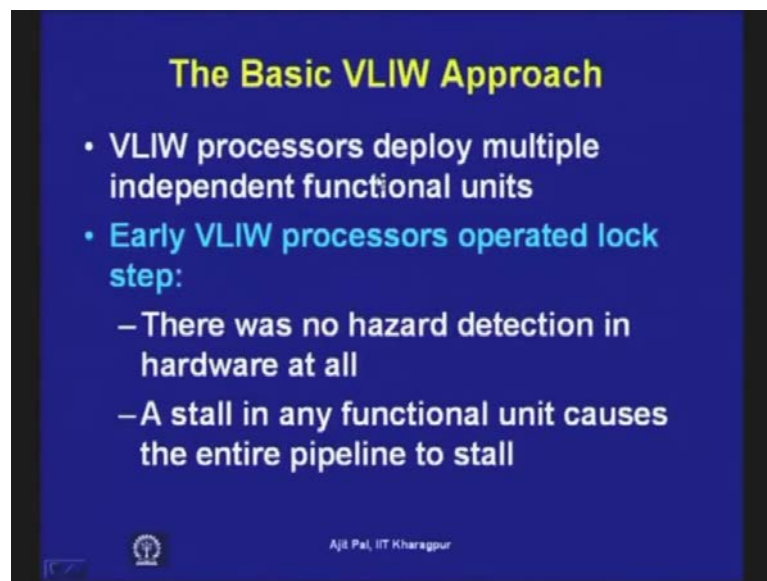
So, what you are doing you are identifying several instruction may be say 4 to 8, 4 to 8 instructions which are identified by the compiler then you are packing them into a single instruction. So, instruction 1; instruction 2; instruction 3 and instruction 4 so these 4 instructions are put together into a single VLIW instruction. And as a consequence the size of the instruction is long, because a single instruction I mean normally in pipeline processors you will find that instruction with these only this much. But since you have packed or bundled several instruction in a single instruction in a VLIW processor the instruction with these longer And that is a reason why this is known as very large instruction word. So, the name has been derived from this, because the length of an instruction is longer so for example, if it 32 bit. So, if you pack 4 instructions; you will require that means it is 4 word 4 into 4 16 bytes. So, 4 byte 4 byte 4 byte 4 byte so you may require 16 bytes for a single instruction and these instructions are stored in the memory.

So, after the compiler you know source code is applied is applied to the compiler and compiler will produce that object code. And in case of superscalar processor, the object code will consist of instructions. And these VLIW instructions that means each instruction is of 4, I mean will comprise of 4 instructions next instruction will also

comprise of 4 instructions like that in this way they will be stored in the memory. Then they can be fetched in order and executed in order. So, after the instructions are bundled together to form very large instruction word they are stored in the memory cache memory or main memory whatever it may be. Then they are fetched one after the other the way simple instructions are fetched and executed in a pipeline processor.

So, after the compilation is done you are your that if you look at the source code; you will find the instructions are like this which are large instruction words. So, may be each of 16 words 16 bytes and static instruction issue, capability static instruction issue means since it is done at compiled time. We are calling it static instruction issue, because as I have explained at compiled time the compiler is analysing. And then they are statically they are generated then they are stored in the memory. So, at run time there is no change so at run time they are fetched one after the other and then they are executed in the same order. So, in fact this concept has been employed in several commercial processors including Intel I a 64 processors this VLIW architecture.

(Refer Slide Time: 21:51)



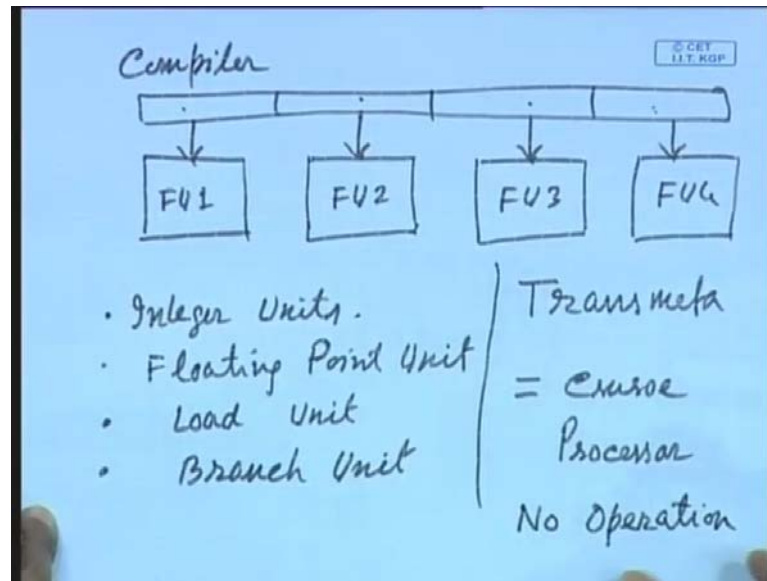
**The Basic VLIW Approach**

- VLIW processors deploy multiple independent functional units
- Early VLIW processors operated lock step:
  - There was no hazard detection in hardware at all
  - A stall in any functional unit causes the entire pipeline to stall

Ajit Pal, IIT Kharagpur

And as I have already mentioned VLIW processors deploy multiple independent functional units so what do you really mean you have got several.

(Refer Slide Time: 22:14)

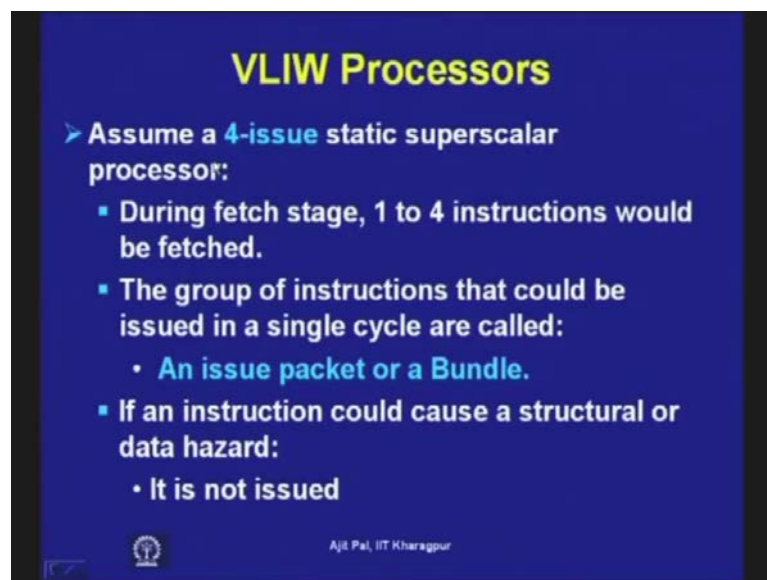


You will require multiple function units say if you are issuing 4 instructions you will require 4 functional units in a single CPU. And these functional units will be fed by a single VLIW instruction as we have already mentioned it consists of 4 operations or 4 instructions. So, each field will feed to this functional unit 1; functional unit 2; functional unit 3; functional unit 4. So, different functional units will be fed by instruction 1; instruction 2; instruction 3 and instruction 4. These functional units obviously need not be identical and in fact they are not identical. For example, some of them can be integer units some of them can floating point units some of them can be load units that means which performs only loading which will perform I mean load and store operation so specialised and fourth type can be branch unit.

So, that means these functional units are not identical they are capable of performing different types of operations. For example, integer units will be able to perform addition, subtraction, multiplication, division of integers or various logical operations. Similarly, floating point units will perform various floating point operations floating point I mean addition, subtraction, multiplication, division, the load store unit will be responsible for storing register values into the memory or loading memory contents into registers. So, their job is specialised so this is this type of multiple functional units actively present in superscalar processors and early VLIW processors operated lock step. That means there were no hazard detection hardware at all.

So, it was assumed since the compiler has done the job they have already identified that there will be no they are independent and that is how they have been issued. So, there was no hazard detection that was necessary that is known as lock step execution. That means one say a bundle of instructions are executed then next bundle of instructions are fetched and executed that is how it proceeded. So, because of some reason if a bundle gets delayed for example, may be because of say cache miss or in such a case it will be delayed. So, load store unit will get delayed instead of loading from the cache memory have to load it from the main memory since such a case it will be delayed. So, a stall in any functional unit caused the entire pipeline to stall so that is how it is implemented.

(Refer Slide Time: 25:52)



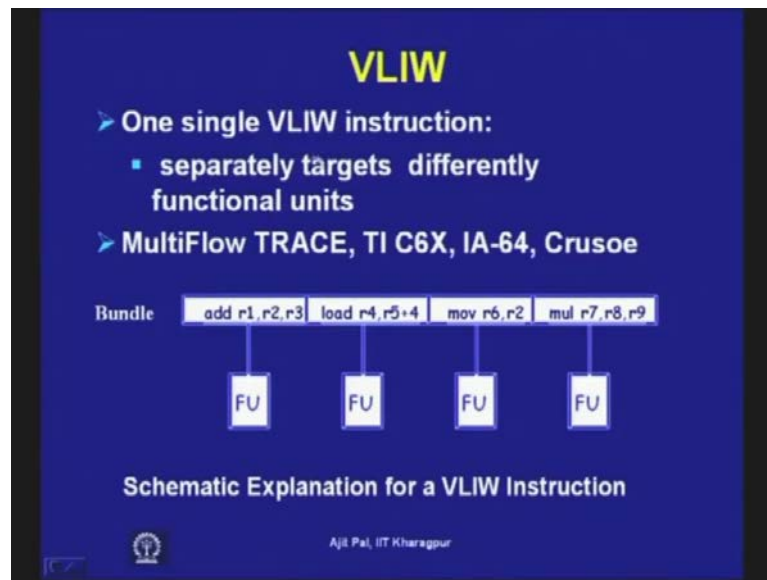
**VLIW Processors**

- Assume a 4-issue static superscalar processor:
  - During fetch stage, 1 to 4 instructions would be fetched.
  - The group of instructions that could be issued in a single cycle are called:
    - An issue packet or a Bundle.
  - If an instruction could cause a structural or data hazard:
    - It is not issued

Ajit Pal, IIT Kharagpur

So, let us now consider a 4 issue static superscalar processor during fetch superscalar means here we are trying to tell that you have got multiple functional units, and during fetch stage 1 to 4 instructions would be fetched. And the group of instructions that would be issued in a single cycle are called an issue packet or a bundle. So, if an instruction could cause a structural or data hazard it is not issued that means here the compiler does the analysis. And finds out whether a particular instruction would cause structural or data hazard structural hazard means there is no enough resources. So, if enough resources are not available then that instruction is not issued of if it identifies that there is data hazard. So, there is data dependency among instructions they are also not issued. That means the instructions have to be completely independent only then they are issued with the help of this v l l d processors.

(Refer Slide Time: 27:03)

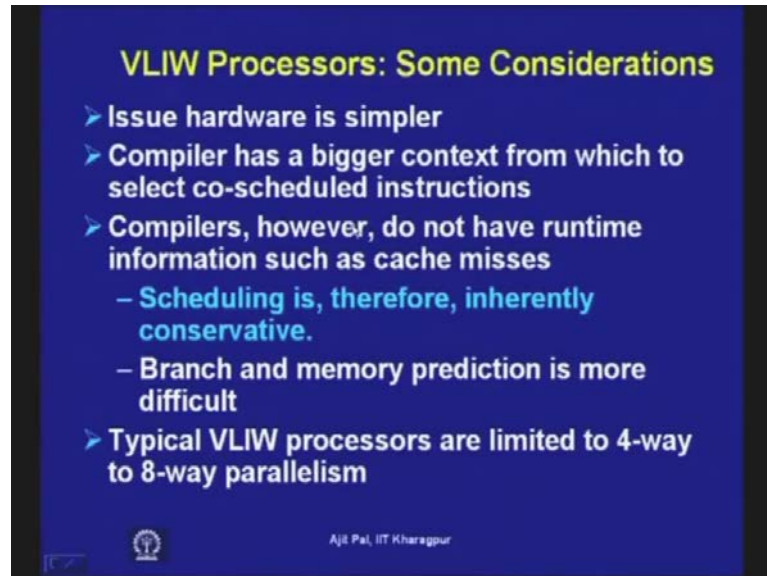


So, here for example, one single VLIW instruction these are separated separately targets different functional units. Obviously, they have to be executed concurrently and each field should target different functional units as I have already told. For example, so here some practical or commercial processors which are based on these VLIW approach are multiflow, TRACE, Texas instruments, C6X, titanium IA 64 by Intel then Crusoe processor by transmeta. So, these processors these are commercial processors which were based on VLIW approach so and a bundle in these cases in all these cases the processor the compiler will issue a bundle. I mean the processor will issue a bundle issue a bundle means. It will fetch an instruction and that instruction comes comprises few 4 field and they are fetched together and the 4 operations will be issued. So, the bundle is issued. So, you can see you have multiple functional units so this is a add r 1, r comma 2 comma r 3. So, this will be issued to one functional units so this is a obviously; this is a this is a integer functional unit.

So, it is it is performing addition of integers similarly; this is a load store unit to which this instruction is issued load r r 4 comma r 5 plus 4. Similarly, mov r 6 comma r 4 this; this is again a data transfer instruction, but between registers. So, this will be also be performed by functional units then multiplication another integer unit which is used which will perform the multiplication. It will take 2 operants from registers and perform multiplication and store the result in a register. So, you can see this is the schematic

explanation for a VLIW instruction which is the generalised you know generalised picture but.

(Refer Slide Time: 29:37)

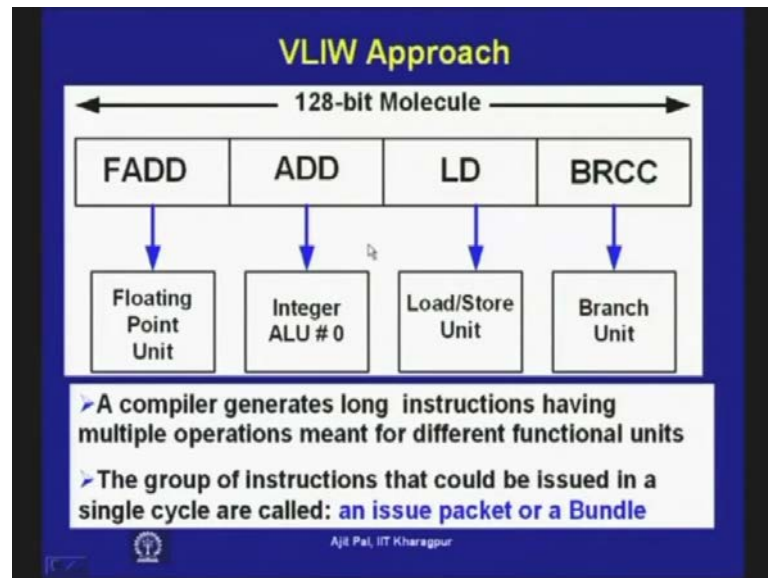


For different processors I mean there will be some differences I shall consider in detail one particular type. So, these I have already mentioned in the issue hardware is simpler, because here issue hardware is not doing analysis on the instructions, because that analysis has been performed by the compiler. So, as a consequence the issue hardware is simpler compiler has a bigger context from which to select co-scheduled instructions. And compilers however, do not have runtime information such as cache misses. I have already explained that at compiled time many thing which can happen at runtime is not known. That means at runtime cache miss is a phenomenon which will happen only at runtime which cannot be predicted at compiled time.

That means whether cache miss will take plus by looking at the set of instructions one cannot say that a cache miss will occur. So, scheduling is therefore, inherently conservative, because that means the instruction level parallelism that it can exploit is that scope is limited. For example, branch and memory prediction is more difficult. So, whether a branch will be taking place or not and and memory prediction that means the effective address value whether they are same or not that identification is also difficult that can be identified only at runtime. And as a consequence difficult VLIW processors

are limited to 4 way or eighth way parallelism as I have already told that the number of instructions that can be issued in parallel is restricted to 4 to 8 in case VLIW processors.

(Refer Slide Time: 31:38)



So, this is in the context of that transmeta Crusoe processors I have already mentioned transmeta that is a name of a company they introduced their processor named as Crusoe. crusoe processor that in for that Crusoe processor. Then Crusoe processor that is a VLIW that used a VLIW architecture and there what they said that an instruction of VLIW processor is called molecule and each field is called atom. So, as per their terminology transmeta terminology, so each molecule consists of 4 atoms. So, you have you have got different functional units floating point unit, integer unit, load store unit and branch unit and in 4 fields instructions of those types are packed and then they are issued simultaneously. So, a compiler generates long instructions having multiple operations meant for different functional units and the group of instructions that can be issued in a single cycle are called an issue packet or a bundle.



(Refer Slide Time: 33:06)

### VLIW: Example

- A simple example:
  - Original code:  
for (i = 1000; i > 0; i = i-1)  
x[i] = x[i] + s
  - MIPS code:  
loop: L.D            F0,0(R1)  
      ADD.D        F4,F0,F2  
      S.D           F4,0(R1)  
      DADDUI       R1,R1,#-8  
      BNE           R1,R2,loop

➤ VLIW Processor

➤ Five operations: One integer, two FP, two memory references, each requiring 16-32 bits field

Ajit Pal, IIT Kharagpur

So, let me illustrate this VLIW execution with the help of the same example that we have discussed in the context of pipelining. So, same programme high level language programme where you are adding a scalar value s 2, different elements of array and the result is stored in the in the memory. So, and this is the corresponding mips code for that particular programme which I have explained earlier. Now, let us see how we can, how we can run this or how this particular programme can be run on a VLIW processor. So, here VLIW processor is assumed to perform 5 operations; 1 integer, 2 floating point operations, 2 memory references each requiring 16 to 32 bits field.

(Refer Slide Time: 34:19)

### VLIW: Example

Memory reference 1	Memory reference 2	FP operation 1	FP operation 2	Integer operation/branch
L.D F0,0(R1)	L.D F6,-8(R1)			
L.D F10,-16(R1)	L.D F14,-24(R1)			
L.D F18,-32(R1)	L.D F22,-40(R1)	ADD.D F4,F0,F2	ADD.D F8,F6,F2	
L.D F26,-48(R1)		ADD.D F12,F10,F2	ADD.D F16,F14,F2	
		ADD.D F20,F18,F2	ADD.D F24,F22,F2	
S.D F4,0(R1)	S.D F8,-8(R1)	ADD.D F28,F26,F2		
S.D F12,-16(R1)	S.D F16,-24(R1)			DADDUI R1,R1,#-56
S.D F20,-32(R1)	S.D F24,-40(R1)			
S.D F28,-48(R1)				BNE R1,R2,loop

➤ Unrolled 7 times to avoid delays

➤ 7 results in 9 clocks, or 1.29 clocks per iteration

➤ 23 ops in 9 clock, average 2.5 ops per clock,

➤ 60% FUs are used

➤ Note: Need more registers in VLIW

Ajit Pal, IIT Kharagpur

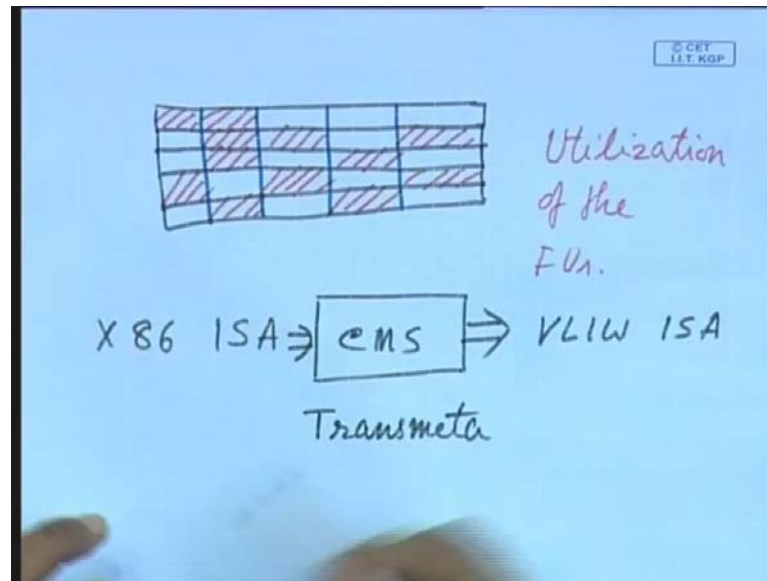


And here the same instructions are getting executed. So, here what has been done the loop has been unrolled this particular role has been unrolled 7 times to avoid delays. Because you require more instruction level parallelism to feed the different functional units and that is a reason why you will have to unroll many times. And that unrolling has been done 7 times to avoid delays and 7 results in 9 clocks we can see or one point here what is being done each line corresponds to a single VLIW instruction.

So, here you have got 1 2 3 4 5 6 7 8 9. So, that means you will require 9 clock cycles whenever you execute it with the help of a VLIW processor. So, this whenever you do that it results in 9 clocks or 1.2 clocks per iteration. So, 1.29 per iteration and you are performing 23 23 operations in 9 clocks average of 2.5 operations per clock. So, that means in terms of your simple pipeline what will you say here IPC is 2.5 instruction per cycle is 2.5, because you are able to perform two point operations per clock.

But one point you should notice that the instructions is the instructions are having 1 2 3 4 5 fields, out of these five fields in many for if you look at different instructions you will find the different fields are empty. What does it mean? That means that that the compiler has failed to identify independent instructions which can be executed concurrently. So, it does not mean that you have got 4 fields or 5 fields that does not mean that all the fields can be filled up by the compiler. Compiler may not be able to fill up these fields and as a consequence some of the fields will remain empty what do you mean by they will remain empty you have to fill up those fields with the help of an instruction known as no operation or no op that means nothing is being performed. So, if you look at the different instructions so let us assume that it has got 5 fields.

(Refer Slide Time: 37:26)



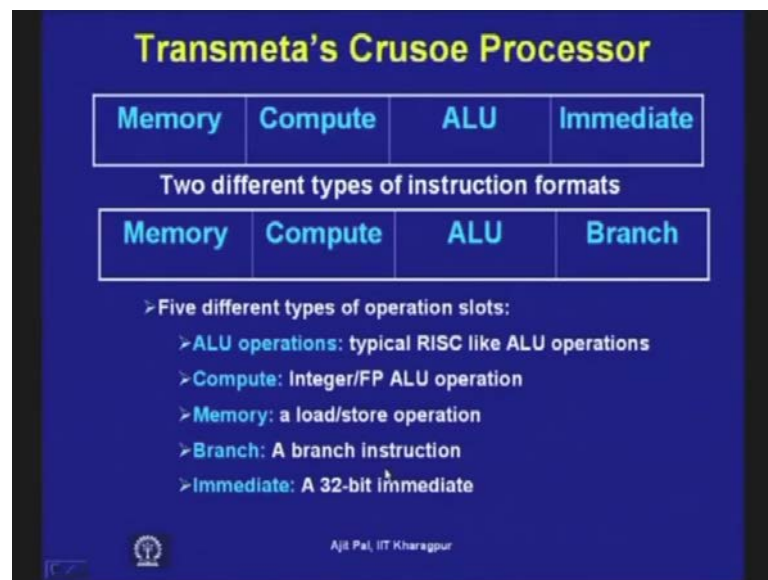
These are the subsequent instructions now one field, second field, third field, fourth and fifth field. So, what can happen if you look at the different fields? You may find that these two fields have been filled up other, three has remained unutilised or these three fields have been filled up two has remained unutilised or these three fields have been filled up. These two fields have been filled up and or it may be like this field has been filled up; this field has been filled up; this filled has been filled up and this way some of the fields may remain unfilled.

So, in such a case what will happen that utilisation of the functional units will not be 100 percent that means the utilisation of the functional unit will be less, because some of the fields have remained unutilised. So, that is the situation here also. So, only 60 per cent of the functional units are used. So, if you look at you will find that 3 fields have remained unutilised in the first instruction, 3 fields have remained unutilised in the second instruction. One field has remained unutilised in a third instruction 2 fields have remained unutilised in the fourth instruction and like that.

So, and another point you must also notice you will require more registers in VLIW that is the characteristic, because whenever you do loop unrolling you need more registers to avoid constraints. And whenever you go for VLIW you require more instruction level parallelism. So, the loop unrolling required for achieving higher ILP is more we have seen in the context of simple pipelining the unrolling of 3 times or 4 times was enough.

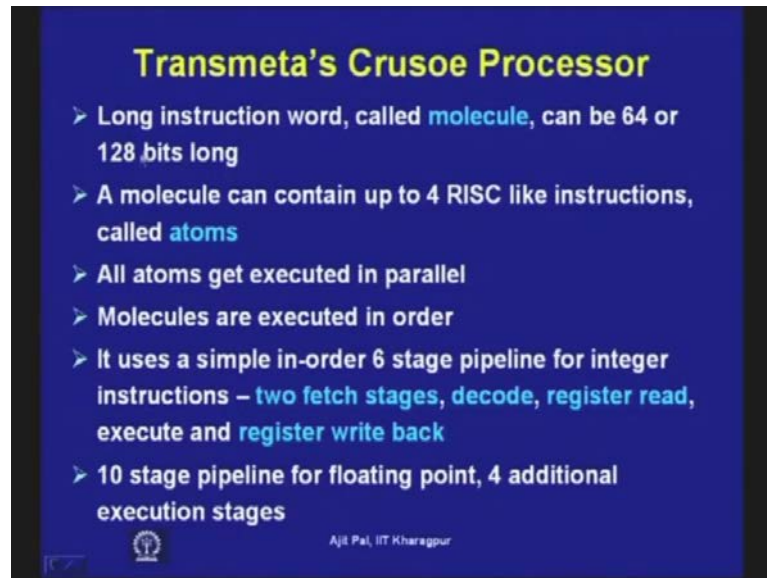
And that was built to provide you enough instruction level parallelism and without any stall. But whenever you are going for VLIW processor; you will require more unrolling to have higher ILP, and as a consequence you will require more registers in the VLIW. That means the VLIW processors should be provided with more number of registers and here as you can see you have utilised all the, I mean 32 different registers floating point and fixed point registers. Most of the since the operations are floating point operations all the floating point registers the 32 floating point registers have been used effectively and in spite of that your functional utilisation is only 60 per cent So, this limitation you have to accept whenever you go for VLIW architecture.

(Refer Slide Time: 41:17)



Now, I was talking about one processor that is transmeta's Crusoe processor that is transmeta's Crusoe processor that is say commercial processor it allows two different types of instruction formats memory compute ALU and immediate, immediate means immediate data field or memory compute ALU branch. So, we find that using these 4 fields you can have 5 different types of operation slots ALU operations typical RISC like ALU operation. Then compute integer or floating point ALU operation where these 2 fields compute field then memory field a load store operation. The first field and branch a branch instruction this is here is a branch and immediate a 32 bit immediate data that can be provided. So, you find 32 32 32 32 that means 32 into 4 that is the, that is a size of the instruction in case of this transmeta's Crusoe processor.

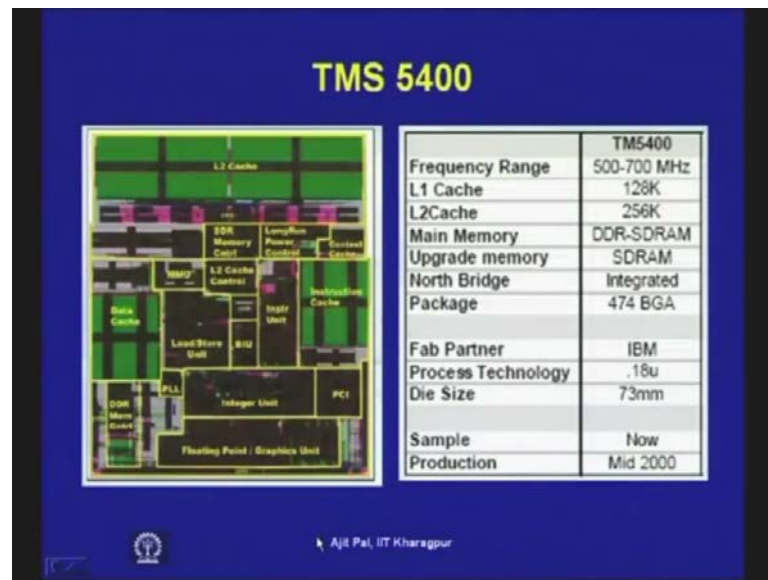
(Refer Slide Time: 42:31)



And I have already told that long instruction word called molecule can be of 64 or one hundred and twenty eight bit long a molecule can contain up to 4 RISC like instructions called atoms because here you are forming a bundle with 4 instructions all atoms get executed in parallel molecules are executed in order. So, these are the typical characteristics of VLIW processors and the same thing is followed in transmeta's Crusoe processor. And it uses a simple in order 6 stage pipeline for integer. So, we find that not only you have got multiple functional units the functional units itself can be pipeline. So, that is additional thing additional complexity that is being incorporated.

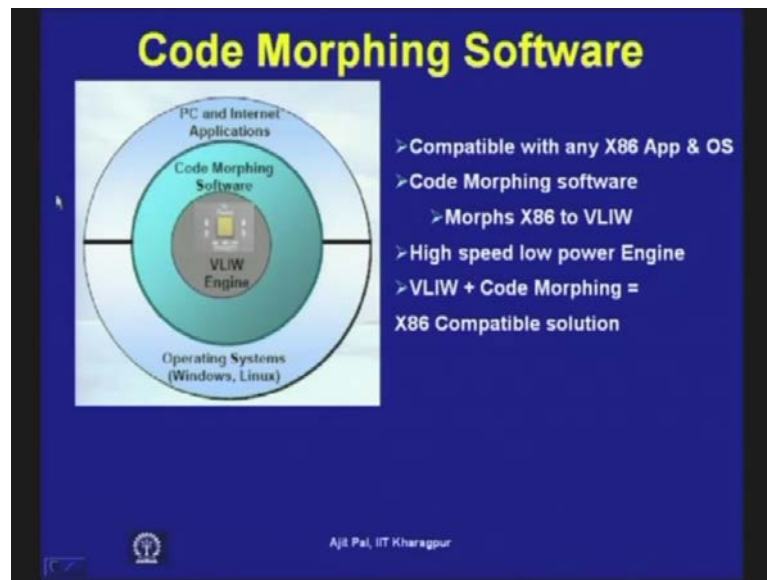
So, the functional units will be pipelined so 6 stage pipeline for integer instructions to fetch stage decode register read register write back. So, these are the pipeline stages that is present in for the integer instructions similarly 10 stage pipeline for floating point and 4 additional execution stages. So, we require 10 plus 4 4 stages for floating point.

(Refer Slide Time: 44:19)



So, we find that different instructions will take different time to execute they will not take same time so that is present in transmeta that is done is, Transmeta's Crusoe processor and that is the implementation of transmeta's Crusoe processor. This is the VLSI chip layout it can operate in the range of 500 to 700 megahertz L 1 cache is 128 kilobyte, L 2 cache is 256 kilobyte, main memory is DDR SD RAM typ. It can upgrade to SD RAM type north bridge is integrated package is this is the package in which is the 474 BGA and Fab partner IBM anyway. And process technology is 0.1 these are the details of implementation die size is 73 millimetre.

(Refer Slide Time: 45:08)

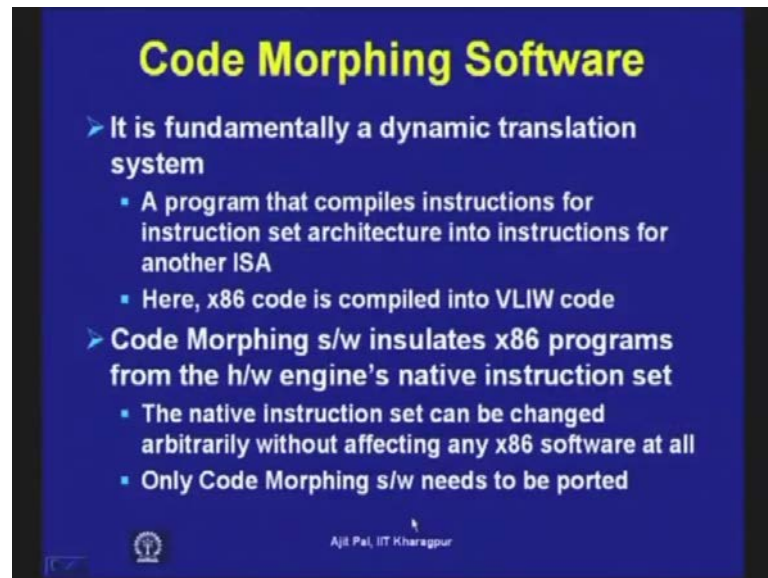


And that the production started in mid 2000 and this is the basic approach that is being followed in transmeta's Crusoe processor. They have used a Nobel concept known as code morphing software. What is this code morphing software? This code morphing software; what it does? It translates from one instruction set architecture to another instruction set architecture. So, for example, the instructions that it will fetch will correspond to X86 ISA then code morphing software CMS will convert it transform it into VLIW instruction set architecture. So, this is a special software code morphing software which is provided by transmeta to translate from one instruction set architecture to another instruction set architecture. Why it is it being done? This is being done, because they found that most of the application softwares are available for you know X86 processors that is Pentium and other processors.

So, the application software will correspond to X86 or Pentium like processors. Then they are converted into instruction set architecture of VLIW and that can run in the VLIW processor that is you at the centre you have got this VLIW engine. That means outer layer correspond to PC and other internet applications operating system windows Linux and so on. And in between the VLIW engine and the outer layer application software, system software you have got this code morphing software which is acting as a interface between the two. So, which morphs X86 to VLIW and it is a high speed low power engine VLIW plus code morphing is equivalent to X86 compatible solution. That

means the programme which has been developed for Pentium can be run on this particular processor.

(Refer Slide Time: 47:57)



And this code morphing software is a it is a dynamically fundamentally a dynamic translation system a programme that compiles instructions for instruction set architecture instructions for instructions architecture into instructions for another instruction set architecture. So, as I mentioned X86 code is compiled into VLIW code. So, what is being done here code morphing software insulates X86 programme from hardware engine's native engine instruction set. So, what is the main benefit of this approach? The main benefits of this approach as we know with the advancement of technology, we keep on adding new generation of processors Pentium 1, Pentium 2, Pentium 3, Pentium 4 and so on.

So, like that this VLIW processor which was developed by transmeta back in 2000 they that will also keep on upgrading and their functionality their can be improved so as it is being done what has to be changed. So, as you change the VLIW engine what modification is required in your system? So that the users can user is not affected user interest is not affected only change that is required is the code morphing software that means as you change the VLIW engine. The code morphing software needs to be modified and that is being provided by transmeta even not affected. So, that approach is followed by transmeta so the native instruction set can be changed arbitrarily without



affecting any X86 software at all. And only code morphing software needs to be ported so that is the only modification that is required.

(Refer Slide Time: 50:08)

**Code Morphing Software**

- What are the keys to Code Morphing Software?
- Translates X86 compatible PC applications to VLIW hardware
- Learn and optimizes the application
- Provides a platform for future extensions

	Mobile PII	Mobile PII	Mobile PIII	TM3120	TM5400
Process	.25m	.25m shrink	.18m	.22m	.18m
On-chip L1 Cache	32KB	32KB	32KB	96KB	128KB
On-chip L2 Cache	0	256KB	256KB	0	256KB
Die Size	130mm <sup>2</sup>	180mm <sup>2</sup>	106mm <sup>2</sup>	77mm <sup>2</sup>	73mm <sup>2</sup>

Table 1. The Code Morphing software simplifies chip hardware.

Ajit Pal, IIT Kharagpur

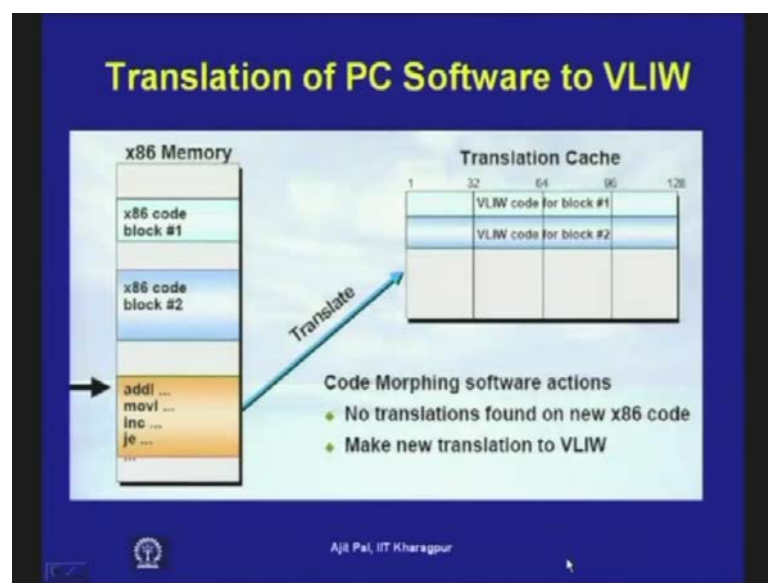
And this code morphing software this that learn and optimises the application that means this code morphing software, what it does from the first experience that code morphing software can be made more and more sophisticated to facilitate to provide give you more and more optimisation. And so that is that provides a platform for future extensions and here is some comparison about the that with Pentium and transmeta processors on the that second line, first line gives you different types of processors that is implemented mobile Pentium 2 mobile Pentium 2, mobile Pentium 3, transmeta TM 3120, TM 5400. So, that is latest and the process technology that is used where point to 5 micron. Then subsequently .18 micron and for transmeta TM 3120, .22 micron and for transmeta TM 5400 .18 micron and on chip cache was smaller for Pentium processors. But for transmeta the on chip cache size they could provide was larger, because they were able to provide more cache memory in the processor.

Because the processor was simple since the processor was simple they were requiring lesser chip area that means that real state silicone real state that were consumed by processor part was smaller. And as a consequence they were able to provide more cache memory and that is why for TM 3120 96 kilobyte and for TM 5400 128 kilobyte. And on chip L2 cache they were present for mobile Pentium was 256 kilobyte and for transmeta



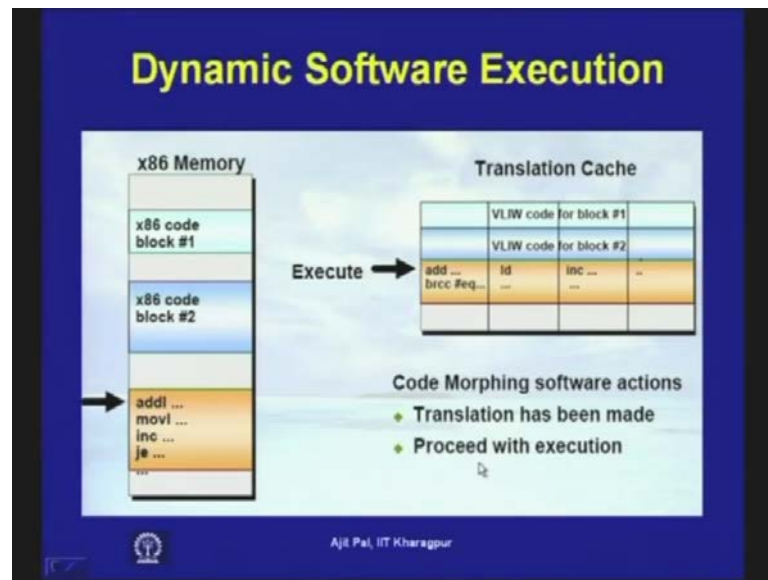
TM 3120. There was no on chip L2 cache, but for TM 5400 the on chip L2 cache that was provided was 256 kilobyte. But in spite of higher cache memory you can see the die size is smaller for transmeta processors. So, you find that that for mobile Pentium 2 130 millimetre square for mobile Pentium 2 another version 180 millimetre square or 106 millimetre square using smaller dimension .18 micron technology. But for transmeta it was around 77 of 73 millimetre square so with smaller die size this was possible. So, the particularly the code morphing software simplifies the chip hardware that is the conclusion from this.

(Refer Slide Time: 53:28)



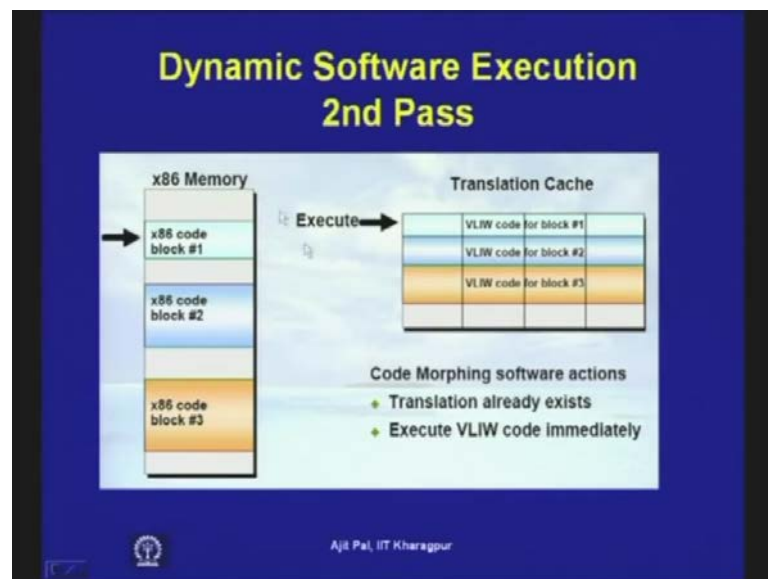
Now, what is the impact of this? We can see this is an example here you have got X86 memory after the translation, this is the VLIW code 4 fields are filled up and no translations found. So, there will be some there will be blanks.

(Refer Slide Time: 53:53)



That is quite natural.

(Refer Slide Time: 54:01)



So, this is the actually it is done in 2 steps that code morphing operation is done in 2 steps. First it finds out the, those it finds out the it divides into RISC like operations, and in the second step.

(Refer Slide Time: 54:21)

## Code Morphing Software

**x86 (IA-32) Instruction Mix**

```
1. movl %eax,%edx
2. jmp 801
3. movl %edx,%ecx
4. movl %ecx,%edx
5. movl %edx,%ecx
6. movl %ecx,%edx
7. movl %edx,%ecx
8. movl %ecx,%edx
9. movl %edx,%ecx
10. movl %ecx,%edx
11. movl %edx,%ecx
12. movl %ecx,%edx
13. j 802
14. movl %ecx,%edx
15. movl %edx,%ecx
16. movl %ecx,%edx
17. movl %edx,%ecx
18. movl %ecx,%edx
19. movl %edx,%ecx
20. jmp 801
21. movl %ecx,%edx
```

**"Morphed" (128-bit) VLIW Instructions**

```
1. addl %r30,%r30,0x2fe
2. addl %r30,%r30,0x2fe
3. addl %r30,%r30,0x2fe
4. addl %r30,%r30,0x2fe
5. addl %r30,%r30,0x2fe
6. addl %r30,%r30,0x2fe
7. addl %r30,%r30,0x2fe
8. addl %r30,%r30,0x2fe
9. addl %r30,%r30,0x2fe
10. addl %r30,%r30,0x2fe
11. addl %r30,%r30,0x2fe
12. addl %r30,%r30,0x2fe
13. addl %r30,%r30,0x2fe
14. addl %r30,%r30,0x2fe
15. addl %r30,%r30,0x2fe
16. addl %r30,%r30,0x2fe
17. addl %r30,%r30,0x2fe
18. addl %r30,%r30,0x2fe
19. addl %r30,%r30,0x2fe
20. addl %r30,%r30,0x2fe
21. addl %r30,%r30,0x2fe
```

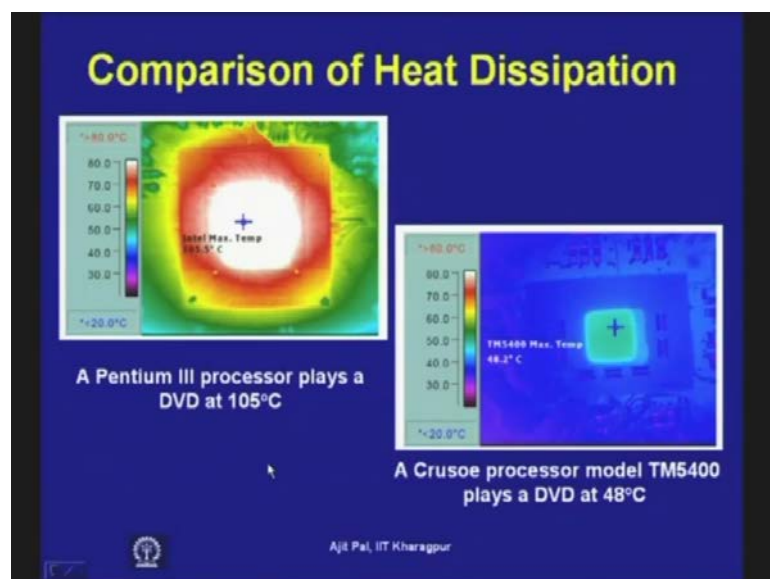
• A Smart Processor analyzes the programs you run

- Optimization begins as soon as programs are launched
- Code Morphing™ detects the top running blocks
- Uses techniques beyond out of order engine
- Benefits
  - Good Performance
  - Low Power (reduced number of executions)

Ajit Pal, IIT Kharagpur

It does the forming the, that translates into VLIW codes. So, here this is the output; this is the Intel X86 instructions and this is the morphed version of the VLIW instructions. So, this is how it is being done so.

(Refer Slide Time: 54:50)

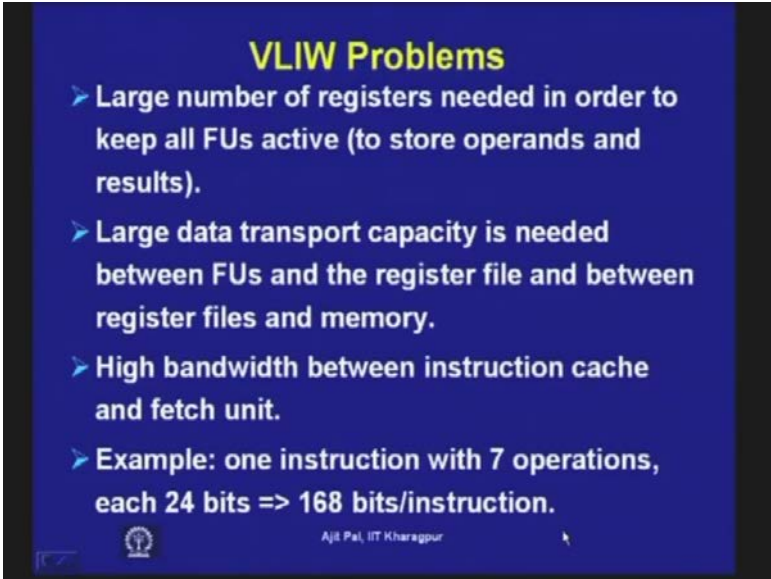


But the main benefit that you get from this VLIW processor is demonstrated by the help of this diagram. You can see here you are running the same programme one in case of Pentium 3 which is playing a DVD player. And you can see the temperature profile the Intel inside, Intel the temperature is reaching 105 centigrade. It is getting heated on the

other hand in case of Crusoe processor model we find that that is you TM s 5400 playing the same programme your temperature is only 48.2.

So, if you consider from the view point of low power now a days you know low power is becoming increasingly important particularly in portable applications most of the devices. Now, a day's you know laptop p d a's, cell phones and hot note in all these cases they are driven by battery since they are battery driven. It is very important to have lower power consumption and also you know that that temperature which is inside decode is decides the reliability of the processors for higher reliability lower temperature is also very important. It has been found that for every 10 degree rise in temperature reliability becomes half. So, in that context this transmeta's approach is quite good. So, we find that running the same programme it leads to smaller power consumption.

(Refer Slide Time: 56:37)



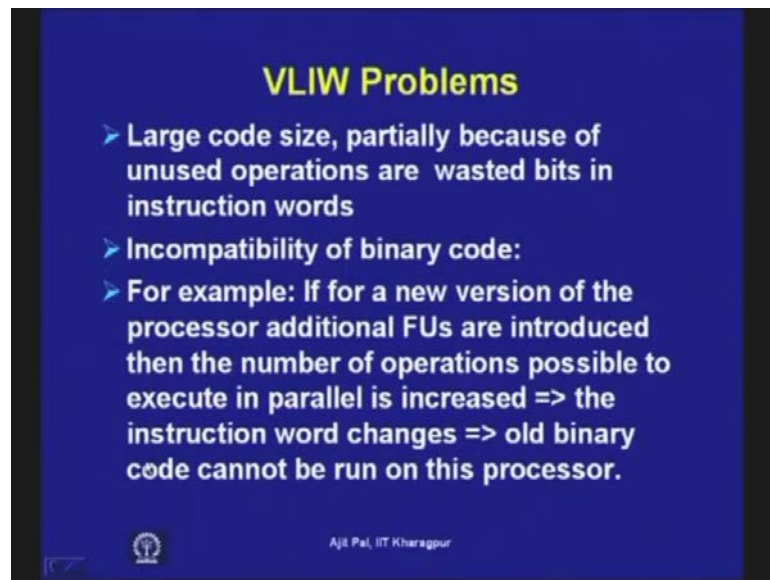
**VLIW Problems**

- Large number of registers needed in order to keep all FUs active (to store operands and results).
- Large data transport capacity is needed between FUs and the register file and between register files and memory.
- High bandwidth between instruction cache and fetch unit.
- Example: one instruction with 7 operations, each 24 bits => 168 bits/instruction.

Ajil Pal, IIT Kharagpur

So, here some of the VLIW problems is identified. Large number of registers needed in order to keep functional units active. Large data transport capacity is needed between functional units and the register file and between the register files and memory. And high bandwidth between instruction cache and fetch unit, because it has to be done in higher speed so one instruction with 7 operations each of 24 bits. So, number of that means the instructions that you have to fetch at this rate 128 bits per instructions that has to be fetched. So, these are the some of the limitations present in VLIW.

(Refer Slide Time: 57:26)



And which can be overcome another problem is large code size partially because of unused operations and wasted bits in instruction words. Incompatibility of binary code and if for a new version of the processor additional functional units are introduced then the number of operations possible to execute in parallel is increased, instruction word changes old binary code cannot run on this new processor. So, that means the, that backward compatibility cannot be easily mended in VLIW architecture. However, if you have got that approach that software like code morphing software then of course, there is no problem anyway these problems can be overcome in superscalar architecture in spite of larger chip area and larger power dissipation. And you will see that that superscalar architecture has been more accepted than VLIW because of the limitations that I have mentioned. So, with this let us stop here.

Thank you.