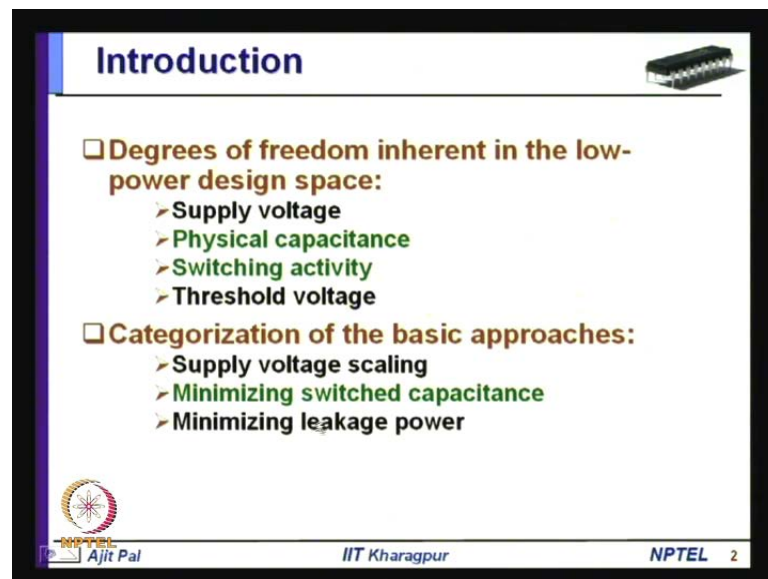


Low Power VLSI Circuits and Systems
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute Of Technology, Kharagpur

Lecture No. # 27
Minimizing Switched Capacitance – I

Hello and welcome to today's lecture on minimizing switched capacitance, in the last four lectures we have discussed various techniques of supply voltage scaling for minimizing power dissipation. We have discussed static voltage scaling multilevel voltage scaling and also you have discuss dynamic frequency and voltage scaling..

(Refer Slide Time: 00:55)



Introduction

- **Degrees of freedom inherent in the low-power design space:**
 - Supply voltage
 - Physical capacitance
 - Switching activity
 - Threshold voltage
- **Categorization of the basic approaches:**
 - Supply voltage scaling
 - Minimizing switched capacitance
 - Minimizing leakage power

NPTEL
Ajit Pal
IIT Kharagpur
NPTEL 2

Now, we shall switch to a new topic that is minimizing switched capacitance and as you know based on over study of sources of power dissipation we identified four degree's of freedom that is supply voltage which been capacitance switching activity and threshold voltage which can be used to reduce power dissipation in c mos circuits and based on based on the various techniques that we shall be discussing. They can be categorized into three types as I mention.

First one was supply voltage scaling second one is minimizing switched capacitance which I shall discuss which I mean, I shall start about discussing on this topic today and after that we shall discuss minimizing leakage power we have already seen.

(Refer Slide Time: 01:43)

Introduction

➤ Power dissipation occurs in the load capacitance as well at all the internal node capacitances

$$P_d = \alpha_0 C_L V_{dd}^2 f + \sum_{i=1}^k \alpha_i C_i V_i V_{dd} f$$

- ❑ The switching power dissipation can be minimized by minimizing capacitance and/or switching activity
- ❑ switching activity × Capacitance is known as switched capacitance

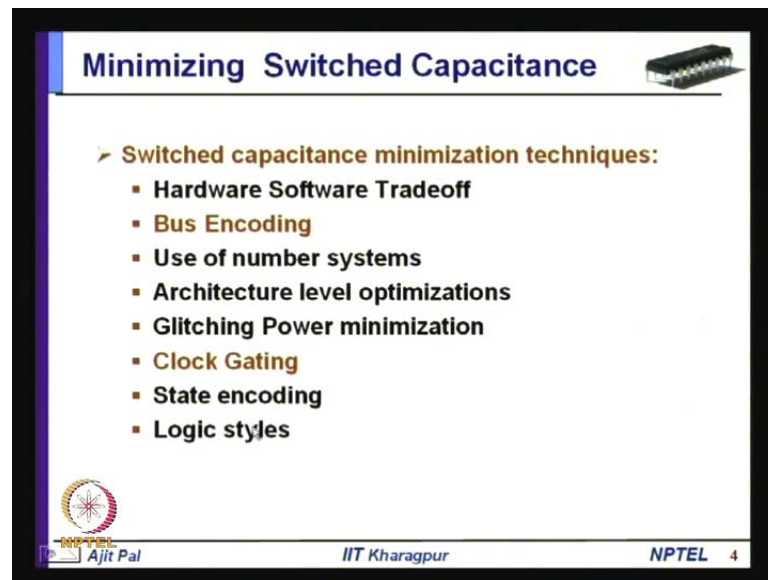
Ajit Pal
IIT Kharagpur
NPTEL 3

The switching power dissipation expression that is $\alpha_0 C_L V_{dd}^2 f$ plus summation of $\alpha_i C_i V_i V_{dd} f$ this is the switching power dissipation that take place in CMOS circuits and essential this power dissipation occurs because of charging and discharging of capacitances.

Now, the switching power dissipation can be minimized by minimizing capacitor capacitance and or switching activity of course, you can minimize by using supply voltage which have already discussed what apart from minimizing power dissipation by scaling down the supply voltage another technique is reduce the power dissipation by reducing the by switched capacitance by switched capacitance i mean which is which is a product of switching activity and capacitance.

So, essentially αC_L **switched capacitance** αC_L ; that means, αC_L the presents the switched capacitance either we shall minimize switching activity α or C or together.

(Refer Slide Time: 03:03)



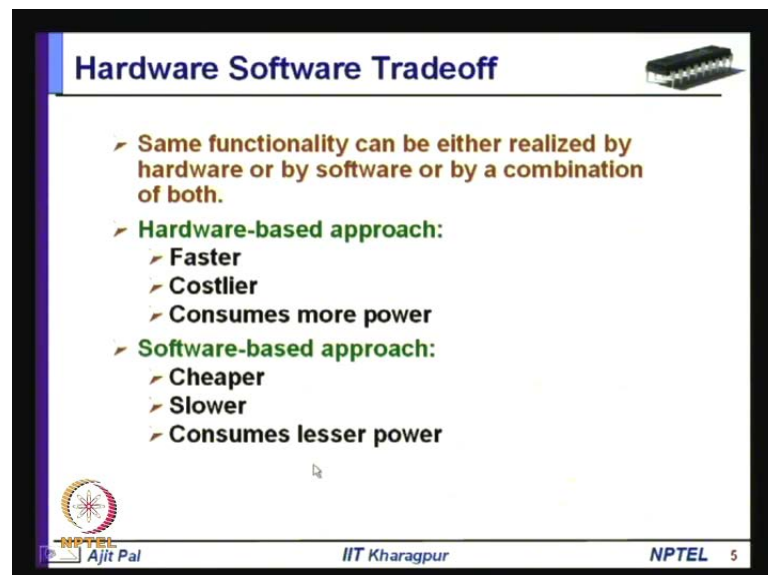
Minimizing Switched Capacitance

- **Switched capacitance minimization techniques:**
 - **Hardware Software Tradeoff**
 - **Bus Encoding**
 - **Use of number systems**
 - **Architecture level optimizations**
 - **Glitching Power minimization**
 - **Clock Gating**
 - **State encoding**
 - **Logic styles**

NPTEL Ajit Pal IIT Kharagpur NPTEL 4

So, I mean we shall discuss various techniques of minimizing switched capacitance you can see the here is a big list first one is hardware software tradeoff then bus encoding use of number systems and there are a various architecture level optimizations techniques then glitching power minimization techniques clock gating state encoding logic styles.

(Refer Slide Time: 03:28)



Hardware Software Tradeoff

- **Same functionality can be either realized by hardware or by software or by a combination of both.**
- **Hardware-based approach:**
 - **Faster**
 - **Costlier**
 - **Consumes more power**
- **Software-based approach:**
 - **Cheaper**
 - **Slower**
 - **Consumes lesser power**

NPTEL Ajit Pal IIT Kharagpur NPTEL 5

So, today we shall start about discussion on hardware software tradeoff.

What do you really mean by hardware software tradeoff whenever we start designing a system first we identify which part to be implemented by hardware and then that

hardware part of the design will follow the i mean flow of hardware design will take place; that means, the hardware design will flow be used to realize the hardware.

Similarly, we identify which functions we can realize by software and then there is a traditional software design flow the software design will take place using that flow and then of course, you have to do what is known as system integration; that means, you have to integrate hardware and.

Now, a very important step is identifying which part to be implemented by software which part to be implemented by software there is a tradeoff. So, the tradeoff is based on cost performance power dissipation and, so on.

So, in let me lasted with the help of an example and example is coming because same functionality can be either realized by hardware or by software or by combination of both; that means, you know you will find that a particular piece of function weather it is analog to digital conversion or compression or decompression or inscription or description many such functions you will encounter in imbedded systems or various other applications.

These functions can be implemented either by hardware fully by hardware or by software or a judicious combination of hardware and software question is how to implement them efficiently and; obviously, whenever you implement it by hardware it will be faster and unfortunately it will be costlier and more importantly it will consume more power.

On the other hand we know that whenever we go for software based approach. It is cheaper to implement because you do not require any additional hardware; however, a software based approach will be slower in in performance, but most important is it will consume much less power.

(Refer Slide Time: 05:28)

Example: 8-bit ADC

➤ **Approach-1:** This approach involves the use of a costly ADC chip along with a few lines of program code to read the ADC data

NPTEL Ajit Pal IIT Kharagpur NPTEL 6

Let me lasted with the help of a very simple example suppose you have to implement an eight bit analog to digital converter in the context of a microcontroller based application.

Where may be you have to sense temperature ambient temperature or temperature of your water bottle whatever it may be. So, you have to convert an analog voltage to a digital one this is a very common requirement in almost I mean many imbedded applications how you implement the analog to digital convertor.

(Refer Slide Time: 06:46)

①

- Costly
- Fast
- More Power

②

- Cheaper
- Slower
- Less Power

NPTEL

Your first approach is to implement it with the help of a your hardware; that means, what you can do you can use one a d c chip analog to digital converter then it can be interfaced to a micro controller. So, here is your micro controller and of course, microcontroller interfacing can be done through I the data bus this is your data bus. So, say zero to seven whatever it is a eight bit eight bit a d converter.

Then there will be two additional lines for hand shaking because the microcontroller will initiate send signal to start conversion and also there will be another line which is end of conversions these are available in the analog to digital converters which these signal will go to port lines of the microcontroller and then will interface this a d c and; obviously, there will be analog input here the analog to digital conversion can be done.

So, this is the first approach. So, in this particular approach we find that the a d c function is implemented by hardware and; obviously, you will require very few lines of software; that means, start conversion signal for sending the start conversion signal one few lines of code then another few lines of code for monitoring to see when the end of conversion signal is coming then another one or two line to read the date from the data bus.

So, the software part is very simple here, but you are using a costly hardware because this **this** a d c is usually costly and of course, it is fast and it consumes more power another implementation technique is you can realize a analog to digital converter function with the help of a digital analog converter you can interface one digital to analog converter then of course, you will require you will require some additional hardware that is your comparator.

So, this analog to digital analog convertor will receive input from the microcontroller and the output of the d a c which is an analog signal will be applied to one input of the comparator and another to another input you will be apply the analog signal analog input and this will going to one port line of the microcontroller and of course, usually a pull-up resistor is needed for interfacing into microcontroller because this is this comparator can be implemented by using say l m three nine three chip.

So, this is how you can interface digital to analog converter then you will be have applying the output of the d a c to a comparator and comparator output is fade to microcontroller.

Now, question arises how do you really implement the analog to digital converter operation the analog to digital conversion function that is your successive approximation algorithm can be realized in software you can realize right a program of course, the program will be little longer and by that you can implement that successive approximation algorithm and successive algorithm will perform analog to digital conversion by with the help of this digital to analog convertor and comparator.

So, in this particular case you see the digital to analog convertor is a much simplified i mean much simpler then analog to digital analog to digital convertor for example, if the cost of the a d c is rupees five hundred the cost of a d d d a c is rupees 50. So, order of magnitude differences in cost not only order of magnitude difference in cost order of magnitude difference in complexity. So, the power of consumption d a c is much less.

However here it is it is the operation will be slow. So, although it is cheaper to implement for example, total cost of implementation will not exceed may be two less than it will be less than two hundred rupees on the other hand the cost of a d c a d c itself here will be rupees five hundred. So, it will be much cheaper to implement; however, it will be and power consumption will be less, so less power consumption.

So, we find that with the help of this simple example we have seen hardware software tradeoff the analog to digital converter conversion operation can be either realized in hardware or it can be realize in software and depending on your requirement. If you require very fast conversion; obviously, the you have to go for this this particular choice; that means, you have to use this first technique where you will be used an analog to digital convert a chip.

However in numerous applications the conversion need not be very fast for example, when you are monitoring the temperature of a water bath or monitoring the temperature of geyser or monitoring the temperature of a air condition room in such cases the temperature does not change very quickly and; obviously, you have the option of using slow analog to digital conversion and in such a case you can use this second option and whenever you use this second option you have the benefit of a cheap implementation and consume the power consumption is less.

(Refer Slide Time: 13:04)

Example: 8-bit ADC

Approach-I: This approach involves the use of a costly ADC chip along with a few lines of program code to read the ADC data

Approach-II: In this approach the ADC is implemented by software with few inexpensive external components such as a digital-to-analog converter (DAC) and a comparator

NPTEL 6

So, this is very simple example by which i have illustrated the illustrated the software hardware tradeoff.

(Refer Slide Time: 13:09)

Superscalar Architecture

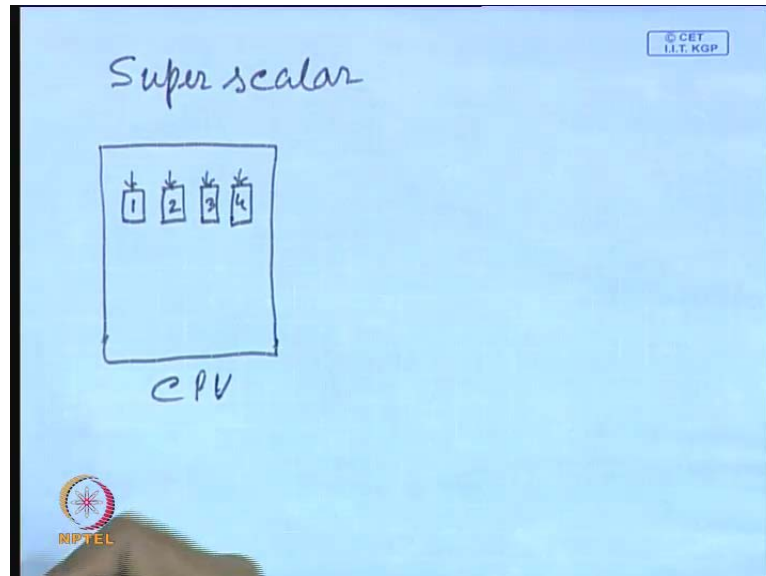
□ A superscalar architecture implements a form of parallelism called instruction-level parallelism within a single processor

NPTEL 7

Now, let me go to a more complex application more complex example, but very important example here we shall consider two different c p u architecture one of them is superscalar architecture all of you may not be familiar. So, i shall briefly explain what do you really mean by superscalar architecture all the Intel processors Pentium starting from

three eight six four eight six to Pentium all of them are superscalar processor what do you really mean by superscalar processors.

(Refer Slide Time: 13:59)



Superscalar as you know the performance can be improved performance of the processor can be improve by incorporating parallel. So, in the say if you go back to the days of eight zero eight five eight zero eight six inside the c p u there was a single functional element; that means, there was only one a l u arithmetic and logic unit, but now i mean with the advancement of technology it is possible to put more functionality in a single c p p u. So, what was done more functional more than one functional elements were provided inside a c p u.

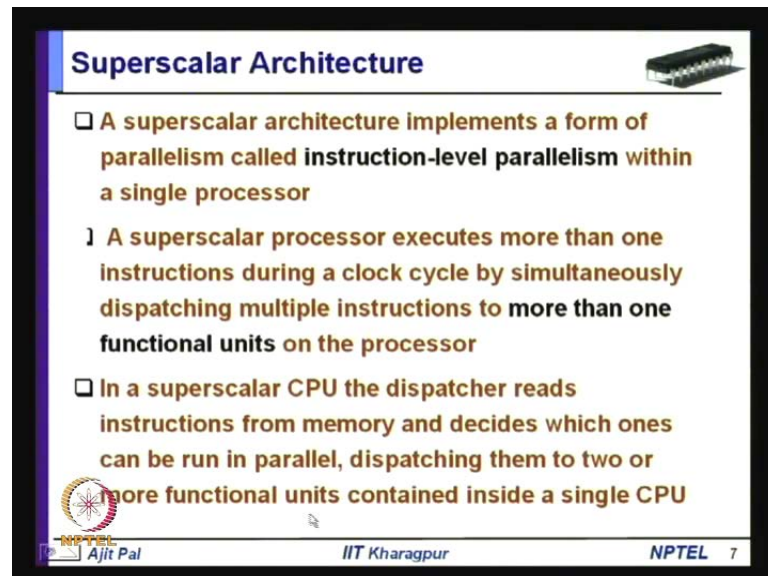
So, normally say this is a c p u inside the c p u you can have several functional unit one can perform floating point operation one can perform fixed point operation one can perform load and branch operation like that. So, you can have several functional unit within a single processor essentially to increase the through-put of the processor that is the basic idea behind superscalar architecture.

So, a superscalar architecture implements a form of parallelism called instruction level parallelism. So, here what you are doing you are having a number of functional unit say in this case I have shown four functional units what is being done four different operations can be executed in parallel; that means, you will be identifying which instructions can be executed in parallel. So, it exploits instruction level parallelism and

by that all the four functional units can be kept busy by issuing i mean up-codes for these four functional unit how it can be done I shall explain.


Essentially within a single processor you are having multiple functional units.


(Refer Slide Time: 16:03)



Superscalar Architecture

- ❑ A superscalar architecture implements a form of parallelism called **instruction-level parallelism within a single processor**
- 】 A superscalar processor executes more than one instructions during a clock cycle by simultaneously dispatching multiple instructions to more than one functional units on the processor
- ❑ In a superscalar CPU the dispatcher reads instructions from memory and decides which ones can be run in parallel, dispatching them to two or more functional units contained inside a single CPU

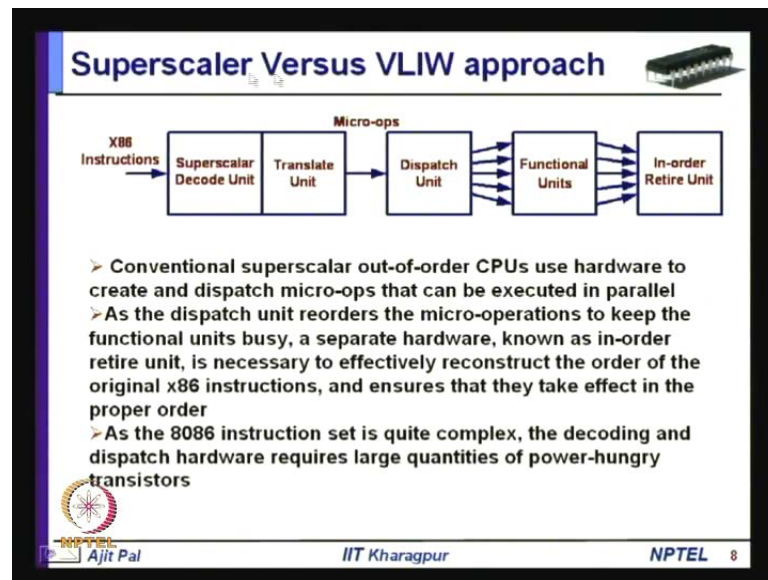


 NPTEL
Ajit Pal IIT Kharagpur NPTEL 7

And where parallelism is exploited using the instruction level parallelism and superscalar-superscalar processor executes more than one instruction during a single clock cycle simultaneously dispatching multiple instruction to more than one functional units on the processors.

As I have shown there are multiple functional units and in a superscalar c p u the dispatcher reads instruction from the memory and decides which ones can be run parallel. So, there is a special case of hardware which reads the instructions from the memory and then identifies which operations or instructions can be executed in parallel and this is all done by hardware.

(Refer Slide Time: 16:45)



So, this is the schismatic background how the Intel's series of processor implement superscalar operation you can see these are the x eighty-six instructions which are complex in nature as you know those Intel user processors are based on cisp **cisp** stand for complex instruction said compute processor or computer. So, these complex instructions are received by superscalar decode unit. So, superscalar decode unit will do the decoding and decomposes a single complex instructions into multiple simple risk like operation.

So, that is been done by the decoding and translation unit. So, those complex instructions are translated into several micro-operations which are typically simple operations risk operation which are perform by processors reduce instruction in computer and then they are send to a another piece of hardware known as dispatch unit. So, dispatch unit identifies which operations or micro-operations can be executed can be performed in parallel and they are send to these multiple functional units and after the operations are performed by the multiple functional units they are send another piece of hardware known as in-order retire unit.

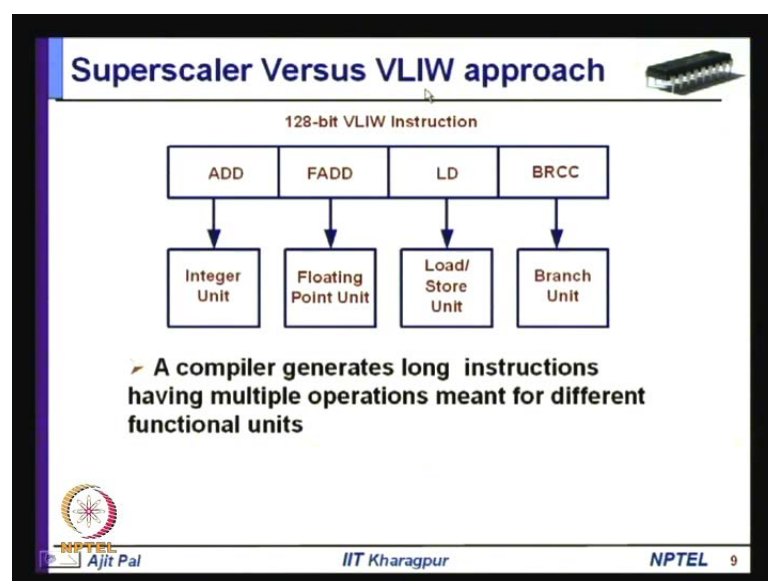
So, what is happening here you are fetching instructions sequentially one after the other then you are doing decoding and sending it to the dispatch unit, but depending on the parallelism possible here there they may be executed with the help of functional units out of order; that means, in-order fetching i mean those instructions which have been fetched

in-order may be executed out of order in this part of hardware. So, you required a another hardware which will ultimately re-order i mean those instructions to get in-order retire unit.

So, you see you require power hungry de-coder unit translation unit dispatch unit and then in-order retire unit. So, these conventional superscalar out of order c p u use hardware to create and dispatch micro-operations that can be executed in parallel and as the dispatch unit reorders the micro-operations to keep the functional units busy a separate hardware known as in-order retire unit as i have shown here is necessary to effectively reconstruct the order of the original x eighty-six instructions and ensure that they take effect in the proper order.

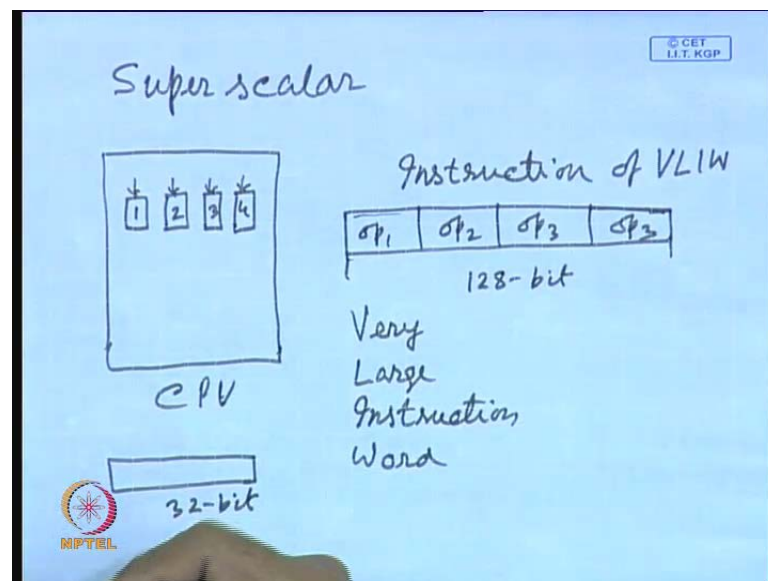
That means in this case we cannot be really disturbed the order. So, as the x eighty-six instruction set is quite complex as i have told the decoding and dispatch hardware requires large quantity quantities of power hungry transistors; that means, the functional units; obviously, will occupy some area, but this part of the hardware which also which will also require a very large portion of the processor; that means, a large part of the silicon real state will be occupied by these power hungry component as a consequence we have seen the Pentium cheap has reach the power dissipation of the hundred watt which we have already.

(Refer Slide Time: 20:34)



Now, what is the other alternative the other alternative is known as v l i w v l i w stands for very large instruction word. So, in this very large instruction word approach what is being done by hardware in superscalar architecture is perform by software. So, in this particular case what is being done the instructions are fetched form the memory and then there is a kind of complier that complier will do the necessary decoding and then after decoding it will decompose complex instructions into micro-operations then it will put several micro-operations into a single instruction like this.

(Refer Slide Time: 21:21)

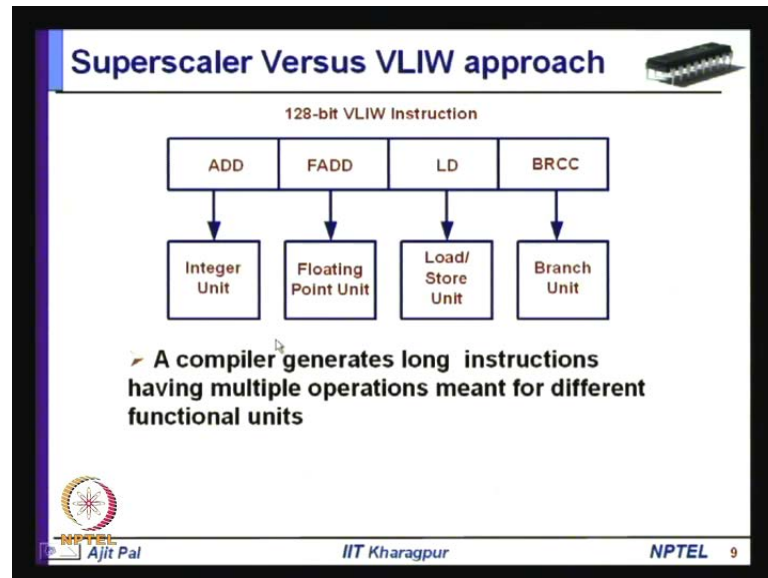


Say a single instruction may have say four or five fields and in each field will correspond to one simple micro-operation say is one operation this is another operation this is another operation this is another operation. So, this is a single instruction single instruction of a v l i w which stand for very large instruction very large instruction.

So, this why it is very large because you can see this instruction length can be much larger compare-**compare** to a superscalar architecture a an instruction of a superscalar architecture can be \ thirty-two bit on the other hand this instruction the width of the instruction can be one twenty-eight bit that is why it is called very large instruction ward a single instruction ward because it is it comprises four different operations and four different operations are here and that is the reason why the width of these instructions will be longer than the than that of superscalar architecture and what is the benefit that we are getting here.

These after that you know after these superscalar **sorry** v l i w compiler has performed conversion of the code into a v l i like instructions then they can be stored in the memory and executed in-order.

(Refer Slide Time: 23:13)

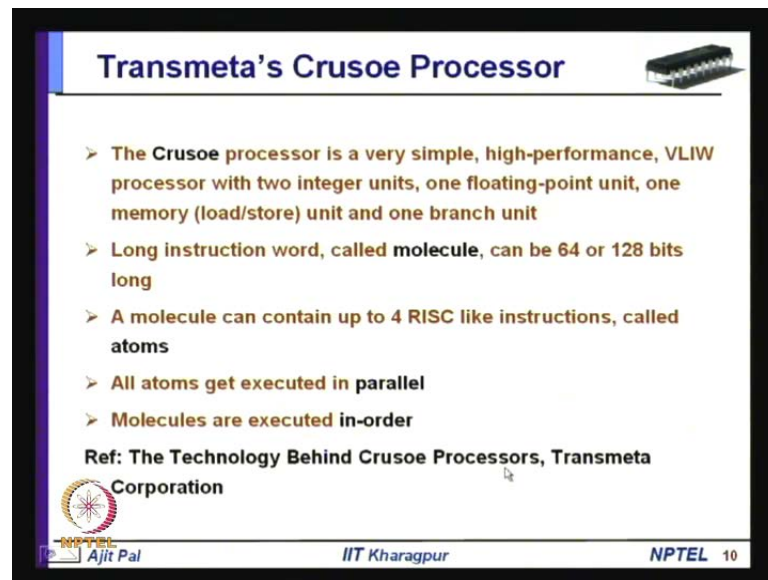


So, in this case a compiler generates long instructions having multiple operations meant for different functional units.

So, you see each of these operations each of these instructions will be fetched one after the other and then they will be executed serially and you can see the operations corresponding to this add is going to this integer unit this floating point add operation up code is going to the floating point unit load operation code is going to the load store unit and the b r c c this code branch-**branch** conditional code is going to do going to the branch unit.

So, in this particular case the c p u has got four functional units integer unit floating point unit load store unit and branch unit and which are fade parallel by the by the by the codes available in a single instructions. So, here it here is single one twenty-eight bit v l i w instruction is shown and there these are this is fade to those operation. So, in this case what will happen the implementation will be far simpler then superscalar architecture?

(Refer Slide Time: 24:32)



Transmeta's Crusoe Processor

- The Crusoe processor is a very simple, high-performance, VLIW processor with two integer units, one floating-point unit, one memory (load/store) unit and one branch unit
- Long instruction word, called molecule, can be 64 or 128 bits long
- A molecule can contain up to 4 RISC like instructions, called atoms
- All atoms get executed in parallel
- Molecules are executed in-order

Ref: The Technology Behind Crusoe Processors, Transmeta Corporation

NPTEL Ajit Pal IIT Kharagpur NPTEL 10

One real i mean real life example is a Crusoe processor there is a there is a manufacturer transmeta. So, transmeta has developed a processor known as Crusoe. So, this the Crusoe processor is a very simple high performance v l i w processor with two integer units one floating point unit one memory unit and one branch unit. So, you can see two plus three plus one load to four five unit; that means, a Crusoe processor has got five functional units inside the c p u inside the single c p u.

But the way it a execute instructions is completely different form that of superscalar architecture how it is done you see the long instruction word they call it in the transmit's Jorgen it is called molecule. So, the long instruction ward called molecule can be sixty-four or one twenty-eight bit. I have already shown the example of one twenty-eight bit long instruction and those are and those four a molecule can contain up to four risk like instructions called atoms.

So, in the Jorgen of transmitter you know each risk like a instructions is called atoms and then four atoms form a molecule and these all these atoms get executed in parallel as you have seen because each of them is a fade to a fade to different functional units. So, they can be executed in parallel and the molecules are executed in-order; that means, the these v l i w instructions are stored in the memory and they are fetched one after the other executed in-order. So, there is no you know that to convert out of order to in-order with the help of the retire unit in this particular case.


So, this you will get details of this from transmitter sites and the particular the part of this what I am discussing today is based on a based on a particular paper the technology behind Crusoe processors and this you can get from traumata's side.

(Refer Slide Time: 26:59)

Area Comparison

	Mobile PII	Mobile PII	Mobile PIII	TM3120	TM5400
Process	0.25m	0.25m	0.18m	0.22m	0.18m
On-chip L1 Cache	32KB	32KB	32KB	96KB	128KB
On-chip L2 Cache	0	256KB	256KB	0	256KB
Die Size	130 sq.mm.	180 sq.mm.	106 sq.mm.	77 sq.mm.	73 sq.mm.

➤ It is evident from the table that Crusoe processor using the same technology and comparable performance level require about 50% chip area


Ajit Pal
IIT Kharagpur
NPTEL 11

And now let us see what is the impact of these v l i w implementation. So, here we have made comparison of several processors some of them are Pentium which are in black color this mobile Pentium two mobile Pentium two mobile Pentium three these are all Pentium processors and you can see the process technology that have been that have been used are given here point two five micron for mobile Pentium two mobile Pentium three point two five micron and mobile Pentium three point one eight micron and then the on-chip cache memory they have got is thirty-two kilobyte and the on chip ready to cache is to have two 50-six kilobyte and die size is the one thirty square millimeter the first one eighty square millimeter for the second one and one zero six square millimeter for the third one.

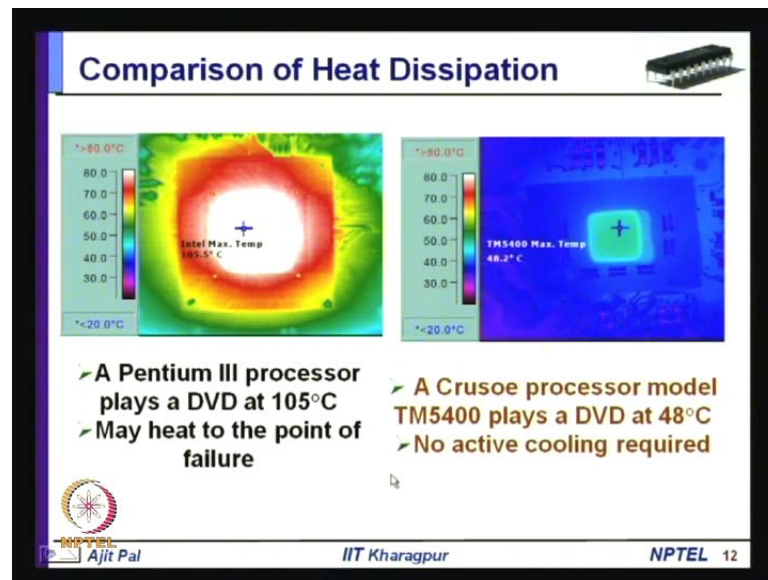
On the other hand transmeta has developed two different types of codes initially to us t m transmeta three one two zero implemented by using point two micron technology and here you can see they have used more on-chip l two cache, because you know the power hungry component which are present in those superscalar architecture is not present in transmitter processors as I consequence they are they can afford to provide more cache

memory. So, this is the thirty-six to thirty-six kilobyte of cache memory is available in t m three one two zero of course, it has no l two un-chip l two cache memory.

But the second generation processor t m five four zero-zero implemented by using point one eight micron technology has got one twenty-eight kilobyte on-chip l one cache and two 50-six kilobyte on-chip l two cache and in spite of using larger cache memory in transmitter processor you can see the die size is much smaller. So, the first generation processor it is seven-seven square millimeter on the other hand in the way for the second generation processor transmitter processor the die size is only seventy-three square millimeter.

So, we find that it is evident from the table that Crusoe processors using the same technology point one eight point one eight micron the comparable performance level and using comparable performance level requires about 50 percent chip area. So, chip area is much smaller; that means, to that same comparable performance with the same comparable performance the chip area requirement is 50 percent.

(Refer Slide Time: 29:53)



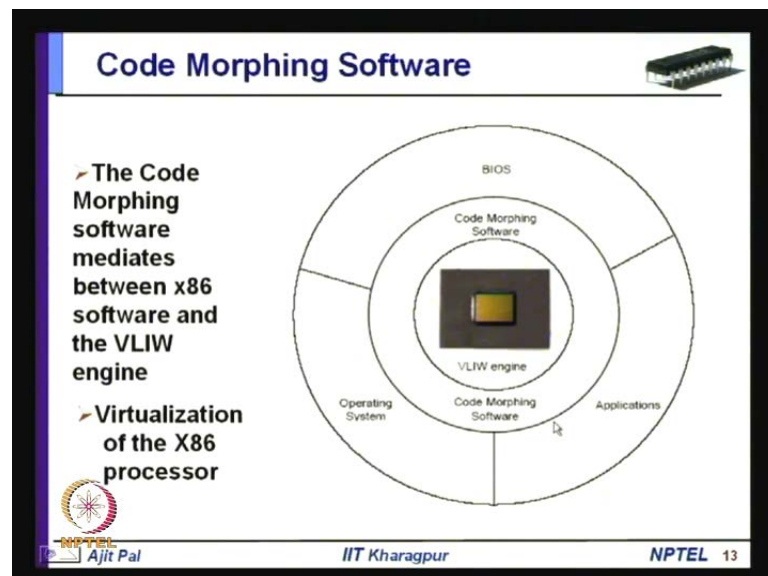
Now, another very interesting demonstration is that you know the test of the fooding is in eating you have to run some application to demonstrate the efficacy of the technique. So, here you know a Pentium three a processor playing a d v d that digital digital video it is executing and the temperature is raising to one zero five degree five point five degree centigrade. So, within the course here the temperature profile within the processor is

shown you can see in the central part the temperature is raising to one zero five point five degree centigrade and this kind of heat this kind of temperature may lead to you know may lead to the point of failure.

That means the Pentium processor may fail whenever the temperature is rising to this level now same applications whenever it is run on Crusoe processor it is very cool. So, you can see the transmit a processor the temperature is raising only to forty-eight degree centigrade. So, you do not really require any active cooling. So, in the in the Pentium processors you require active cooling you require **OO** a very big head sink a fan on the top of the head sink like that on the other hand in case of your transmitter processor no active cooling is required. So, normal ambient temperature is sufficient to cool it down because the temperature is not much.

So, this demonstrates you know the efficacy is approach proposed by transmitters essentially it is a it is a tradeoff between hardware and software the hardware which is implemented in superscalar architecture to perform instruction decoding instruction you know dispatching then reordering are being perform by piece of software in transmitter processor by using v l i w architecture.

(Refer Slide Time: 31:52)



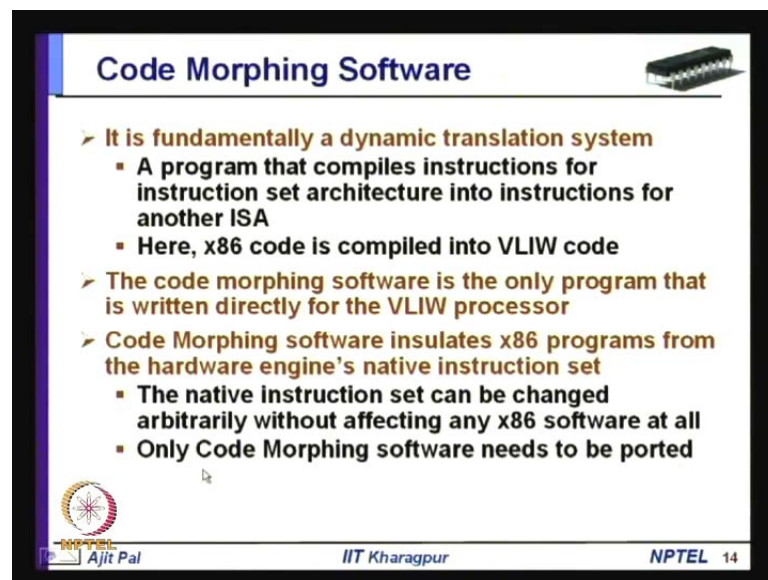
Now, the most important component in this transmitter processor is a piece of software known as code morphing software what it really it does the code morphing software

mediates between x eighty-six software and v l i w engine. So, essentially what it is doing is it is doing a kind of virtualization of the x eighty-six processor.

So, to the it is nothing, but a x eighty-six processor the users have written their program x eighty-six code, but the piece of there is a piece of software known as code morphing software as it is shown it is insulating the v l i w engine from the outside ward. So, here you can see this you know the operating system bios and all the applications that you encounter in in any in your traditional Pentium based system those are all written in x eighty-six; that means, the Intel series of for Intel series of processor.


So, you can have you know traditional operating system and also bios that is required there and various applications that you use word processing and various other things and then these soft these those codes are translated to v l i w code with the help of this code morphing software. So, code morphing software is essentially doing it is fundamentally a dynamic translation system a program that compiles instructions for instructions set architecture for into instructions for another instructions set architecture.

(Refer Slide Time: 33:30)



Code Morphing Software

- **It is fundamentally a dynamic translation system**
 - A program that compiles instructions for instruction set architecture into instructions for another ISA
 - Here, x86 code is compiled into VLIW code
- **The code morphing software is the only program that is written directly for the VLIW processor**
- **Code Morphing software insulates x86 programs from the hardware engine's native instruction set**
 - The native instruction set can be changed arbitrarily without affecting any x86 software at all
 - Only Code Morphing software needs to be ported

 NPTEL
Ajit Pal IIT Kharagpur NPTEL 14

So, what it is doing the code morphing software is translating code sub one instructions sets that is your x eighty-six to another instructions set architecture that is your transmitter's v l i w instruction set architecture. So, these called code morphing software is a only program that is written directly for the v l i w processor and as i mention code

morphing software insulates x eighty-six programs from the hardware engine's native instruction set.

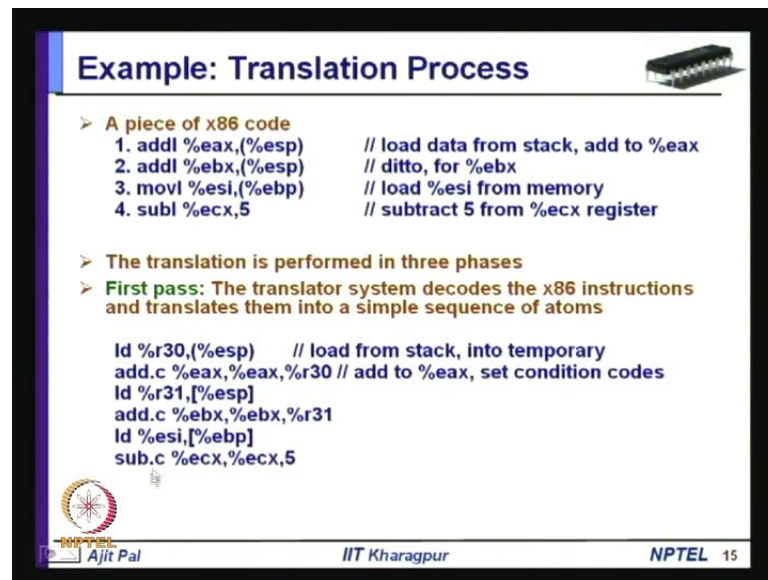
Now, you may be asking if v l i w is. So, advantageous why it has not been used earlier the concept of v l i w very large instruction word architecture is existing for a long time, but it was not popular the reason behind that is when the task of writing code for v l i w processor is given to a user it becomes very a difficult task; that means, user cannot handle the complexity of writing codes for v l i w architecture processors.

But in this particular case what is happening that problem is overcome by this by this code morphing software because the programmer or user is not is not you know is not bothered to write program in for the v l i w processor there writing their program in their traditional Intel processors and the code morphing software is doing the necessary job of translation and. So, that is that is a reason why earlier v l i w processors were not. So, popular.

Now, the native instruction can be changed arbitrary without affecting the x eighty-six software now here the question is suppose just like your Intel series of processors it has gone from three eighty-six to four eighty-six to Pentium and. So, on inductions set has been new instructions have been added it has been more and more complex what about this transmitters as Crusoe processor.

It can be also changed you can add more some important instructions in v l i w processors, but whenever you do such changes what modification in your system is required. So, whenever you do that only thing that has to be done is to change the code morphing software nothing else is required because in this particular system you can see the code morphing software is interfacing with the native v l i w engine and if this engine is changed all the change that is required is in this code morphing software and the outer layer software's which are written in x eighty-six need not be changed. So, the user is not it is transparent to the user you can say because the code morphing is provided by transmeta.

(Refer Slide Time: 37:02)



Example: Translation Process

- **A piece of x86 code**
 1. `addl %eax,(%esp)` // load data from stack, add to %eax
 2. `addl %ebx,(%esp)` // ditto, for %ebx
 3. `movl %esi,(%ebp)` // load %esi from memory
 4. `subl %ecx,5` // subtract 5 from %ecx register
- **The translation is performed in three phases**
- **First pass: The translator system decodes the x86 instructions and translates them into a simple sequence of atoms**

```
ld %r30,(%esp) // load from stack, into temporary
add.c %eax,%eax,%r30 // add to %eax, set condition codes
ld %r31,[%esp]
add.c %ebx,%ebx,%r31
ld %esi,[%ebp]
sub.c %ecx,%ecx,5
```

NPTEL Ajit Pal IIT Kharagpur NPTEL 15

Now, you may ask you may be curious to know how exactly this translation occurs. So, to illustrate that I am illustrating that with the help of the very simple example this is a piece of x86 code here you have got four instructions first one is load and data from stack add to `%eax`. So, you can see it is performing load operation and addition operation with the help of a single complex instruction then it is also the second instruction is also doing the same thing.

The third instruction is load from memory and fourth instruction is subtract from register. So, you can see these four instructions consist of a code of x86 code now the translation of this code into `llvm` code is performed in the three phases in the first pass that means the compiler that dynamic translation unit operates in three phases or three passes you can.

So, what are the three passes in the first pass the translator system decodes the x86 instructions and translates them into a simple sequence of atoms in case of superscalar architecture this is done by that decode and translate unit it is done by hardware. So, here you can see it is done by the software. So, these four instructions are now decomposed now converted into simpler sequence of operations and they called it atoms.

So, you can see load from stack one instruction add another instruction load another instruction add another instruction load another instruction subtract another instruction

these are the. So, this is done in the first pass. So, after this first pass is over in the second pass the translator performs typical compiler optimizations such as common sub expression elimination dead-code elimination and so on so.

(Refer Slide Time: 39:03)

Example: Translation Process

- In the **second pass**, the translator performs typical compiler optimizations such as common subexpression elimination, dead-code elimination, etc
- This helps in eliminating some of the atoms


```
ld %r30,[%esp]           // load from stack only once
add %eax,%eax,%r30      // reuse data loaded earlier
add %ebx,%ebx,%r30
ld %esi,[%ebp]
sub.c %ecx,%ecx,5       // only this last condition code
                        needed
```
- In the **final pass**, the scheduler reorders the atoms and groups them into individual molecules, which is somewhat similar to the function of the dispatch hardware
 1. ld %r30,[%esp]; sub.c %ecx,%ecx,5
 2. ld %esi,[%ebp]; add %eax,%eax,%r30; add %ebx,%ebx,%r30
- The program is executed in-order by the hardware and the molecules explicitly encode the instruction level parallelism

Ajit Pal
IIT Kharagpur
NPTEL 16

Whenever you are writing a optimizing compiler you have to do different types of optimizations and the optimizations like dead-code elimination some codes are generated by the compiler which are never executed. So, those codes can be removed and common sub expression eliminations same computation you have done maybe n times or forth time only one computation is sufficient those since can be removed with the help of a optimizing compiler and i have given two such examples there are several such optimization which are perform by optimizing compiler. So, in the second pass those optimization are done in this particular case essentially you have seen one instruction in the in the previous case you have seen one two three four five six codes six instructions were there which have been converted into i mean one two three four five one has been removed.

Because you know we have seen in the previous case you were loading to a temporary register thirty and you are loading the same data to another register thirty-one. So, which can be eliminated because you can reuse the value twice? So, you can reuse you are the loading into this temporary register then you are reusing it twice and. So, one instruction

can be removed. So, removed the dead-code which is not required and then load and subtracts. So, this is how you have performed some kind of optimizations.

Then you will do a third pass there is a final pass the scheduler reorders the atoms and groups them into individual molecules which is somewhat similar to the functions of dispatch hardware we have seen in a superscalar architecture after the complex codes have been converted into micro-operations they are sent to multiple functional unit with the help of the dispatcher. So, here again it is done by software.

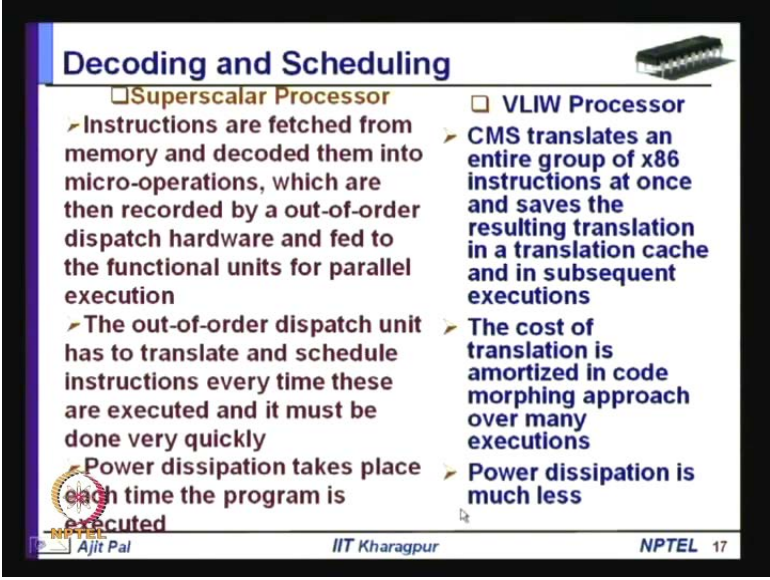
So, in a final pass the scheduler reorders the terms and groups them into individual molecules and which is somewhat similar to the function of dispatch hardware as I have told. So, you can see these five operations are now are now packed in a in few fewer number of v l i w instructions we can see we have it has been possible to pack two operations of course, you have to see which operations can be execute in parallel it may not be possible to execute all of them in parallel. So, whenever these whenever you put them in a single v l i w code then you have to see which operations can be executed in parallel.

For example this load operation and this particular instruction these two can be executed in parallel, but not the other because unless load is done you cannot perform addition operation neither you can perform load operation. So, only these two codes can be executed with the help of the single v l i w instruction then the remaining three are perform are put in single v l i w instructions and they can be executed in parallel.

So, now this can be stored in the memory and then they can be fetched one after the other. So, the first this particular v l i w is instruction will be fetched and all these atoms will be executed together and then the second instruction will be fetched I mean order it will be done in same order then all these three atoms will be executed in parallel.

So, this is how the program execution take place in v l i w architecture. So, the program is executed in-order by the hardware and the molecules explicitly encode the instruction level parallelism; that means, the; that means, both superscalar architecture and v l i w architecture are exploiting instruction level parallelism, but there exploiting instruction level parallelism in two different way one is one is doing hard ware and another is doing by using software.

(Refer Slide Time: 43:44)



Decoding and Scheduling

Superscalar Processor

- Instructions are fetched from memory and decoded them into micro-operations, which are then recorded by a out-of-order dispatch hardware and fed to the functional units for parallel execution
- The out-of-order dispatch unit has to translate and schedule instructions every time these are executed and it must be done very quickly
- Power dissipation takes place each time the program is executed

VLIW Processor

- CMS translates an entire group of x86 instructions at once and saves the resulting translation in a translation cache and in subsequent executions
- The cost of translation is amortized in code morphing approach over many executions
- Power dissipation is much less

Ajit Pal IIT Kharagpur NPTEL 17

Now, here is a comparison between superscalar and v l i w processors. So, you can see what happens in case of superscalar architecture instructions are fetched from memory and decoded them to micro-operations and which are then recorded recorded by out-of-order re-order it will be reordered by out-of-order dispatch hardware and fed to the functional units for parallel execution.

And the out-of-order dispatch unit has to translate and schedule instructions every time these are executed and it must be done very quickly. So, you have seen whenever you are executing a program say you have to execute it 10 times this decoding dispatching reordering have to done 10 times every time you execute all these things have to be done 10 times and; obviously, 10 time power dissipation will take place and. So, power dissipation take place each time the programs is executed

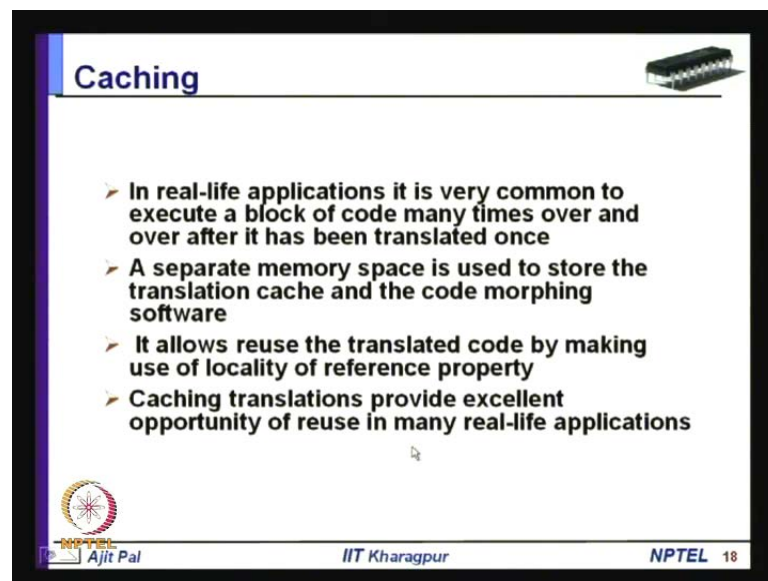
Now, what happens in case of v l i w processor in v l i w processor the code morphing software c m s translates an entire group of x eighty-six instructions at once and saves the resulting translation in a translation cache **cache** memory and in subsequent and use them in subsequent executions. So, what is happening in here you know in case of v l i w processor once the translation has been done by the code morphing software you can store them in cache memory you can executing them many times may be 10 times hundred times depending on the applications, but each times you do not have to do the

translation you will do the translation only once and execute them as many times as it is necessary by the application.

And what happens in this case the cost of translation is amortized in code morphing approach over many executions; that means, the; obviously, the translations operations done by the code morphing software has some overhead that overhead cost is amortized over a large number of execution. So, as a consequence what is happening power dissipation is much less as has been demonstrated by that video execution example?

So, this is the comparison and; obviously, apart from using the basic concept of a v l i w architecture doing translation by using software some more additional you know you'll require additional techniques to improve the efficiency of this particular processor.

(Refer Slide Time: 46:12)



Caching

- In real-life applications it is very common to execute a block of code many times over and over after it has been translated once
- A separate memory space is used to store the translation cache and the code morphing software
- It allows reuse the translated code by making use of locality of reference property
- Caching translations provide excellent opportunity of reuse in many real-life applications

Ajit Pal IIT Kharagpur NPTEL 18

Architecture number one is caching what is caching in real life applications it is very common to execute a block of code many times over and over after it has been translated once you do that in real-time real life that executable code you save in a memory.

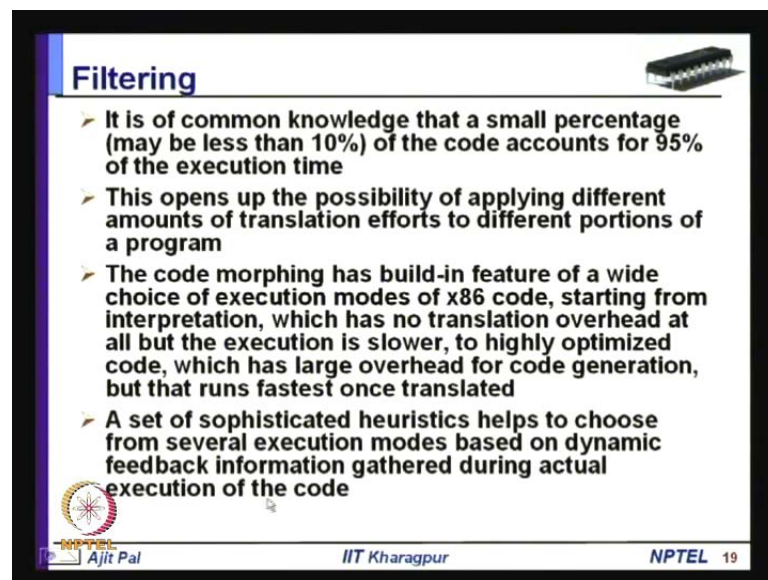
So, here also you will saving, but where you will be saving in a special type of memory known as translation code and it operates like a kind of cache memory you may have you may be familiar with cache memory what is the task of a cache memory you know to execute a program what is a primary requirement primary requirement is that it must be present in a main memory.

Now, instead of storing in main memory a part of the program is stored in the cache memory which part that part which is executed quite often we use a property known as locality of reference and by using a locality of reference a part of the program which is executed most of the time is stored in the cache memory and that is that is the concept used here.

In this case a separate memory space is used to store the translation cache and the code morphing software; that means, may be you have got say hundred different applications, but all of them you may not be executing quite often. So, which are executed quite often those codes you will be storing in the in this the particular cache memory and; obviously, you will be doing it by using the locality of reference property.


So, it allows reuse **reuse** the translated code by making the use of locality of reference property. So, caching translations provide excellent opportunity of reuse in many real-time real-life applications this is very quite common in real life applications apart from caching.

(Refer Slide Time: 48:34)



Filtering

- It is of common knowledge that a small percentage (may be less than 10%) of the code accounts for 95% of the execution time
- This opens up the possibility of applying different amounts of translation efforts to different portions of a program
- The code morphing has build-in feature of a wide choice of execution modes of x86 code, starting from interpretation, which has no translation overhead at all but the execution is slower, to highly optimized code, which has large overhead for code generation, but that runs fastest once translated
- A set of sophisticated heuristics helps to choose from several execution modes based on dynamic feedback information gathered during actual execution of the code

 NPTEL
Ajit Pal IIT Kharagpur NPTEL 19

Another technique which is also used in to improve the performances which is known as filtering what do you really mean by filtering it is common knowledge that a small percentage of the of the code accounts for ninety-five percent of the execution time.

Suppose you have written a big code and it has been observed that only 10 percent of the code is used executed ninety-five percent of the time. So, may be a loop that code which is part of the loop that will be executed many times other part of the code may not be executed as many times.

So, what can be done you can focus on that part of the code which is executed many time rather you can do more optimization you can you can devote more time for optimization on that part decode. So, you can selectively devote shorter or longer time for optimization to different parts of decode that is filtering.

So, this opens up the possibility of applying different amounts of translation affords that is you know effort for efforts for optimization to different portions of a program the code morphing software has built in feature of wide choice of execution modes of x eighty-six code starting from interpretation has no translation overhead at all, but the execution is slower.

To highly optimized code; that means, some part of the codes has to be highly optimized which has large overhead for code generation; that means, that code morphing software will deford long-time for to optimize it, but that runs fasted once translated because that will be executed many times. So, a set of sophisticated heuristics helps to choose from several execution modes based on dynamic feedback information gathered during actual execution of the code.

So, what is being done you know a particular program you running many time. So, whenever it is run once the code morphing software gatherers information which part of the decode is executed quite often which part of the decode is executed less often.

So, based on that you know the translation effort is devote I mean is done differently for different parts of the decode.

(Refer Slide Time: 51:14)

Prediction and Path Selection

➤ The code morphing software can gather feedback information about the x86 program with the help of an additional code present in the translator whose sole purpose is to collect information about block execution frequencies or branch history

➤ Depending on how often a piece of x86 code is executed or whether a conditional branch instruction is balanced (50% probability of taken) or biased in a particular direction, decision can be made about how much effort to put to optimize that code.

➤ In a conventional hardware only x86 implementation, it would be extremely difficult to make similar kind of decisions

The slide features a flowchart on the left with a decision diamond labeled 'Condition ?'. The 'Yes (taken)' path loops back to the start of the block above the diamond. The 'No' path leads to a block below the diamond. The slide also includes logos for NPTEL, Ajit Pal, IIT Kharagpur, and NPTEL 20.

So, this kind of filtering is done then another important thing is that is been done in prediction and path selection. So, the code morphing software can gather feedback information about the x eighty-six program with the help of additional code present in the translator whose sole purpose is to collect information about block execution frequencies or branch history.

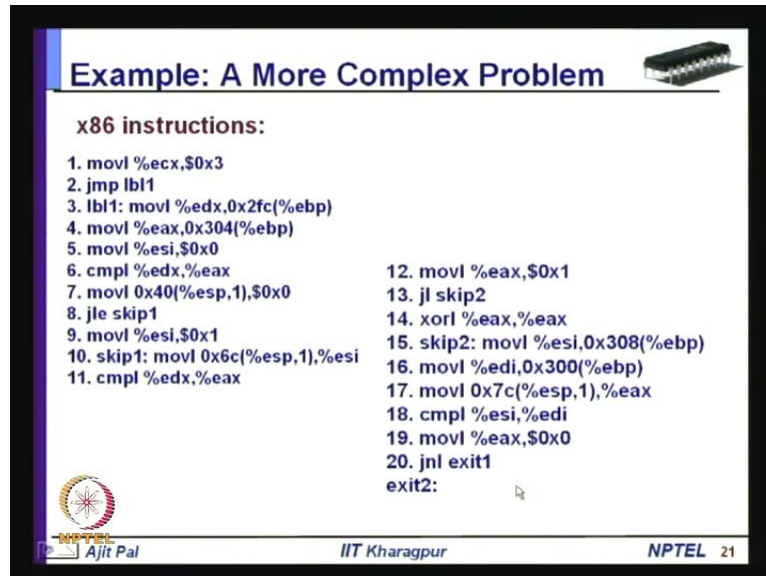
You know here what is been done a kind of branch prediction is done. So, branch prediction is done why for example, here there is a branch it may be taken in that case it will go in this direction and if the condition is not satisfied it will go back to this. So, if the if the taken probability is more then you will put more effort to optimize this part of the code.

On the other hand if the untaken probability is more you will put more effort to on to on this part. So, depending on how often a piece of x eighty-six code is executed or whether conditional branches instruction is balanced. So, if 50 percent probability then of course, there is no biased in any particular direction **direction**. So, a biased in a particular direction decision can be made about how much effort to put to optimize that code..

So, whether you will put more effort to this code or this code that is decided by the by this prediction and path selection. So, in a conventional hardware only x eighty-six implementation it would be extremely difficult to make similar kind of decisions; that means, whenever a you are in doing it hardware it is not really possible to devote more

time for optimization to a particular part of code less time for optimization another part of the code this is feasible only because you are doing by software.

(Refer Slide Time: 52:59)



Example: A More Complex Problem

x86 instructions:

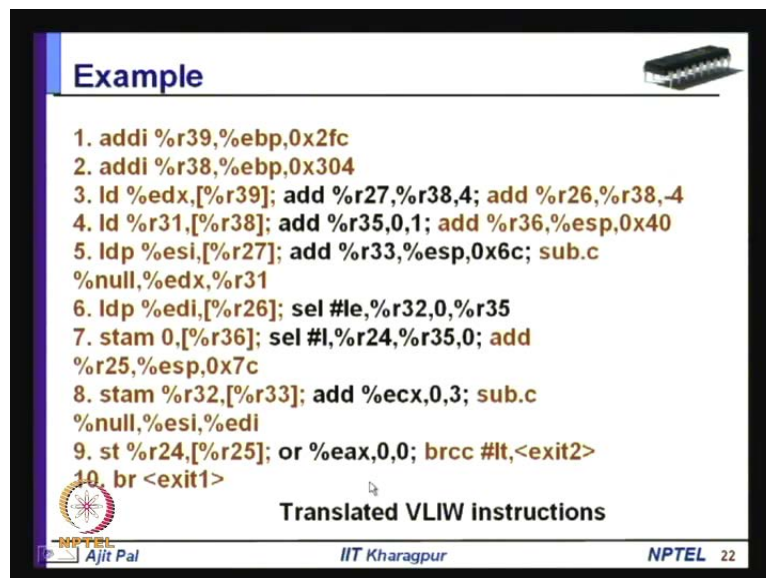
1. movl %ecx,\$0x3
2. jmp lbl1
3. lbl1: movl %edx,0x2fc(%ebp)
4. movl %eax,0x304(%ebp)
5. movl %esi,\$0x0
6. cmpl %edx,%eax
7. movl 0x40(%esp,1),\$0x0
8. jle skip1
9. movl %esi,\$0x1
10. skip1: movl 0x6c(%esp,1),%esi
11. cmpl %edx,%eax
12. movl %eax,\$0x1
13. jl skip2
14. xorl %eax,%eax
15. skip2: movl %esi,0x308(%ebp)
16. movl %edi,0x300(%ebp)
17. movl 0x7c(%esp,1),%eax
18. cmpl %esi,%edi
19. movl %eax,\$0x0
20. jnl exit1

exit2:

NPTEL Ajit Pal IIT Kharagpur NPTEL 21

Here is a more complex example. So, you can see these are all a complex x x 86.

(Refer Slide Time: 53:16)



Example

1. addi %r39,%ebp,0x2fc
2. addi %r38,%ebp,0x304
3. ld %edx,[%r39]; add %r27,%r38,4; add %r26,%r38,-4
4. ld %r31,[%r38]; add %r35,0,1; add %r36,%esp,0x40
5. ldp %esi,[%r27]; add %r33,%esp,0x6c; sub.c %null,%edx,%r31
6. ldp %edi,[%r26]; sel #le,%r32,0,%r35
7. stam 0,[%r36]; sel #l,%r24,%r35,0; add %r25,%esp,0x7c
8. stam %r32,[%r33]; add %ecx,0,3; sub.c %null,%esi,%edi
9. st %r24,[%r25]; or %eax,0,0; brcc #lt,<exit2>
10. br <exit1>

Translated VLIW instructions

NPTEL Ajit Pal IIT Kharagpur NPTEL 22

Instruction which is converted by the code morphing software into only 10 instructions of-course all the all the you know all if you look at a particular molecule you'll see that all the fields are not filled up all the atoms are not present that will happen. So, you

cannot expect to have complete parallelism or hundred percent efficiency, but in spite of that it will give better result.

So, with this we have come to the end of today's lecture. So, we have discussed a very important step known as hardware software tradeoff which can be used to reduce power dissipation particularly which will minimize the switched capacitance of a circuit thank you in the next lecture we shall discuss about other techniques.