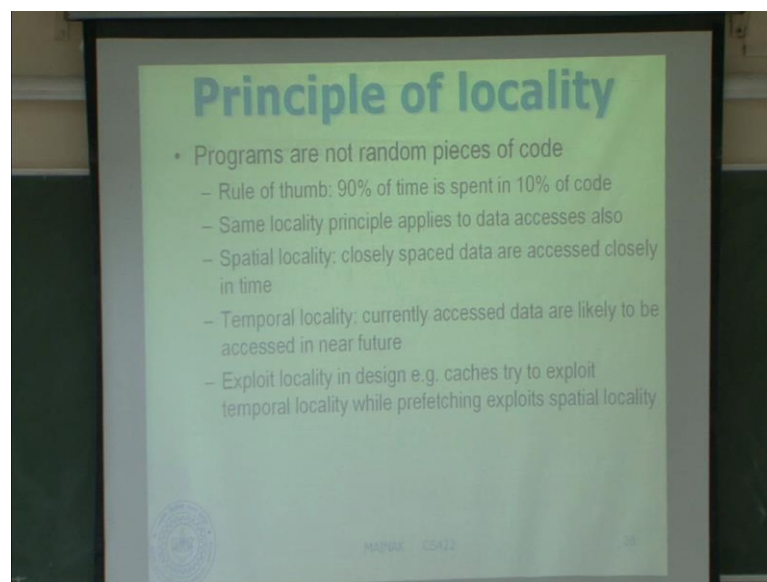


**Computer Architecture**  
**Prof. Mainak Chaudhuri**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kanpur**

**Lecture - 2**  
**CPI equation, Research practices, Instruction set architecture**

We talked about MIPS instructions that actually has fuse compare jump instructor. So, quickly to recap what we were discussing. We talked about performance measurement, metrics, benchmark applications, and the little bit on the performance comparison. We talked about arithmetic mean, geometric mean, and harmonic mean. And talked about Amdahl's law, and CPI equation (( )) numerical put as well.

(Refer Slide Time: 00:50)



We finish here talking about principle of locality last time. So, this is one of the things that will visit over and over, in the contexts of not just caches, but in many other contexts; the principle of locality. So, here do not think that this data always refers to your memory data, it can be any data. So, for example, if you are looking at sequence of data values, any data values, locality principles may still apply there. So, we look at such things also in it.

(Refer Slide Time: 01:37)



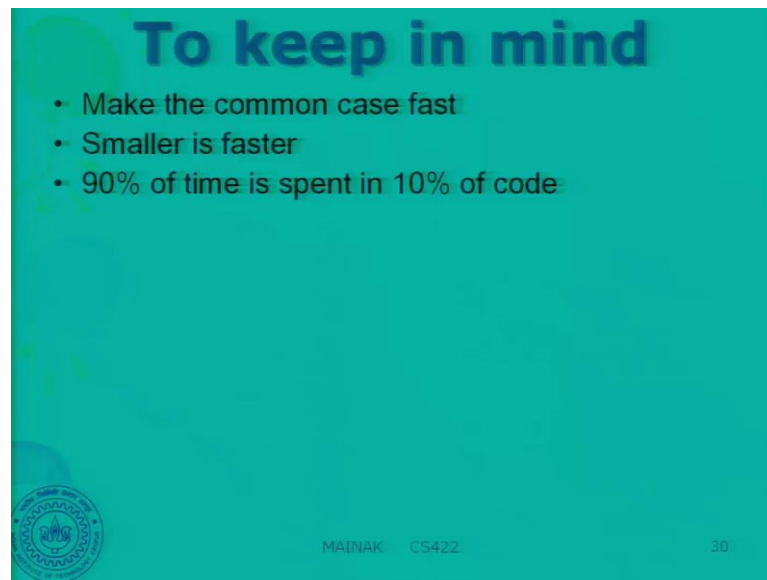
So, the other principle that will be applying, or at least we will try to apply at many places is, (( )) parallelism, and it is pretty much mantra of today's computer system, and you will see parallelism at different levels, in these machines. So, here are some examples; the simple one; for example, you may want to have disks to improve your i o throughput. You may want to have some more memory banks, to support parallel data access. You may want to process multiple instructions in parallel, and digital circuits are inherently parallel systems, because individual bits, get operated on in parallel. You may want to have more ALUs to carry out parallel additions, and finally, speculation is the ultimate solution for extracting parallelism. So, here the main idea is to do multiple possible operations in parallel, without actually knowing which one is the correct one; whereas, the correctness detection proceeds in parallel. So, essentially what it says is that; suppose you have this piece of a code, some condition check here.

(Refer Slide Time: 02:42)



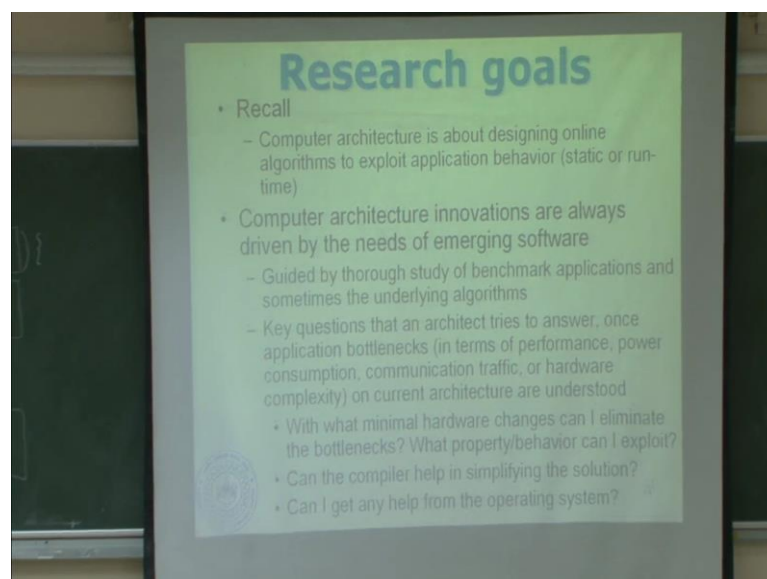
So, here there are three parts of the computation; one is computing this condition. This is the IF block, and this is the ELSE block. So, there are three pieces of computation here, what I can do is, I can run all these three pieces in parallel. Eventually this competition will finish, and I will cancel one of these two. So, effectively what have done is that, I have split up the whole execution. So, that exactly what is trying to saying. I am essentially speculating, about which one is going to be actually correct, so I will run both of them, and essentially cancel the one that is not. If you are smart, then you will actually figure out, which one is going to have higher probability of being correct. So, then I can suppress the other one right there. I can say that well, I know that, with very high chance, this is going to be the correct path of execution. I will not even try to execute that, so will learn about these things as well. So, keep this in mind; parallelism is pretty much everywhere, and we will look at even more complicated varies of exploiting parallelism, inside the processor.

(Refer Slide Time: 04:17)



So, here three things that you should keep it mind, throughout this course; make the common case fast, that is number one; means that do not spent your time in effort on things which are rare. Smaller is faster; that is a very important lesson. If we have small a cash it is going to be very fast, as the cash size increases, it will become slower and slower. Similar thing applies for logic also; small logic is fast, large logic is slow. 90 percent of time is spent in 10 percent, of course that essentially your code looks at.

(Refer Slide Time: 04:54)



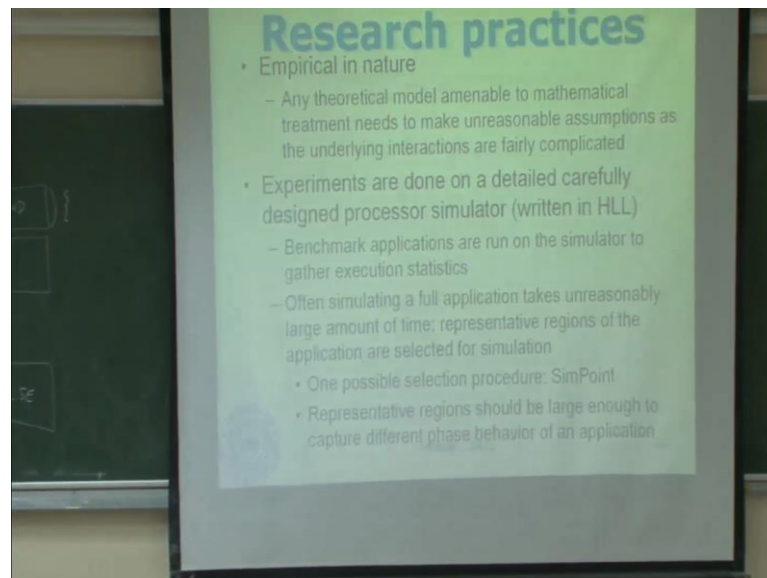
So, before I move on to the next topic, I just wanted to touch up on, little bit on the research side of this vertical pill. So, recall that we talked about this in the first lecture, that computer architecture is about designing online algorithms, they exploit application behavior, and that behavior maybe static or maybe run time. So, if we know that; an application is stackically heavy on memory, then you probably do something in your processor, that would optimize the memory behavior of the data, or there is maybe something that you do not know stackically, by just looking at the application you cannot say. Only when the application runs you discover something. So, ((refer time 05:38)) architecture should be able to adopt, when it sees that behavior, and that is why the online favor of the algorithm come in to picture. You have to react immediate, when you see certain things happening. So, computer architecture innovation are always driven by, the needs of emerging software, that is always a dragging force, but some new software comes up with some new problem, and that is what the architects study, and they propose solutions, so that the software spread up.

So, guided by thorough study of benchmark applications, and sometimes underlying algorithms. So, a computer architects spend a lot of time, understanding applications their behavior, profiling applications, and understanding what they do by understanding the, underline algorithms of the applications. Key questions that an architect tries to answer. Once the application bottlenecks, in terms of performance, power consumptions, and communication traffic or hardware complexity on current architecture are understood.

So, first of course, what you do is that, you take this application, study them on the current architecture; in terms of performance, power consumption, communication traffic, hardware complexity and s on. And there you ask the following three basic questions, with what minimal hardware changes can I eliminates the bottlenecks; that is the basic question. And this minimal part is very important; you have to be thrifty, when you propose something. You cannot just propose something gigantic, for solving something that would not be acceptable. And of course, what property and behavior, can I exploit, because we talked about this thing, that your solution usually exploits certain application behavior. So, what is that property and behavior that I can exploit to solve this bottleneck. Can the compiler help in simplifying the solution; of course, because not everything can be solved in purely hardware.

You will probably require in certain situations help from the software also. So, compiler has the advantages of seeing the whole program, which the processor cannot see. Processor can only see a certain window of execution; that is currently going on inside the processor. It cannot see what is coming in future. Then the compiler can see the whole code. Of course, it cannot see the whole data, but it at least knows what code is coming in future. Can I get any help from the operating system? The operating system schedules the jobs on the processes. So, can I get some hint about what is coming next, or can I get some behavioral hints from the operating system about the certain process. See usually these are the three questions, that pretty much encapsulate the basic research theme, only in computer architecture. Any question. So, how do you carry this out?

(Refer Slide Time: 08:36)



So, the research in this area, is empirical in nature. The reason is that, any theoretical model, amenable to mathematical treatments, needs to make unreasonable assumptions, as the underlying interactions are fairly complicated. So, coming up with an analytical model for an architecture is extremely difficult, without making certain unreasonable assumptions, which is why what you do is, experiments are done, on a detailed carefully designed processor simulator; usually written in high level language. Although you can go to a little lower than high level language, like very log, that is slightly lower than your `c++` that is called an `rtl` language, Ariston transfer language. So, what you do is, you the simulators, try to model the process are that you trying to design, as accurately as possible. So, it is a piece of software. This simulator is actually a piece of software, that

modules your hardware processor, and your benchmark applications of run, on the simulator, together executions statistics.

when you run the benchmark application on the simulator, and you pretty much can get to know whatever you want to know, because it is a piece of software design by you, it is under your control, you (( )) tell you something, you know you can tell that. Often simulating a full application, takes unreasonably large amount of time. You will probably not believe, but there are you know. Well actually pretty much almost all benchmark applications that are important, you will probably take months to run on any good simulator. So, if you if you have to, you know wait for so much of time, then of course, no innovation will get done in time. So, you need some other ways; of course, from scientific ways, of splitting up simulator. So, what usually people do, is that they find out representative regions of the application, and those are selected for simulation only. So, any suggestion how to pick representative regions of an application. So, it talks about something, it says simpoints. You do not have to know about that, I will tell you, any suggestion, can you think of something. I give you an application, some program and I ask you to tell me the representative regions of this program, how do you do that.

Student: How do you thought that sorry try to look at... (( ))

But that region will take months to run on realistic data. 90 percent of a month is what.

Student: (( ))

But that would not be realistic anymore. If you run it all the small data, everything will fit in my memory, I will not be able to module page faults. If you run it even on smaller, even smaller data able to fit in my cash. I will not be able to move up model cash misses. So, you finished your simulation in two minutes, but it is useless.

Student: Sir, try to figure out the regions in the code which had maximum complexity.

How do you define complexity.

Student: Inside more complexity, and we can...

No how do define complexity.

Student: Complexity means that that region you take maximum amount time.

So, that is what he was mentioned you right.

Student: You do not have to run the code, you can just visibly.

you can do that, how do you do that, just by eying a code you can tell me which takes longer, which part takes longer.

Student: To some extent we can say.

You can how.

Student: Loops and some calls.

There were many of those actually, how do you know which one take longer. There may be 20 loops in your program. Which one is important loop.

Student: So, we can figure out, by saying that which has maximum complexity.

are you saying that the loop that is larger in size will be more complex.

Student: sometimes not always.

But I need a solution which is accurate all the time right. I cannot say that hey I am publishing this paper you know 10 percent of this papers results are correct, others may be wrong. Nobody is going to read my paper. Tell me something that is little more scientific.

Student: (( )) we can try that each path is covered, we can have each path is covered. So, that we have execution in the.

So, usually each benchmark application comes with multiple data inputs, and that is how the data inputs are actually designed. So, that it gives you more or less 100% coverage of the program, but so what. Each of those data inputs will take probably months to run. I do not know if you are understanding the question that I am asking. I give you a program I am asking you to give me a representative region of that execution, and a realistic data. So, you can think of an execution as what; a dynamic sequence of instructions, that is an execution. If we run for month, runs for minute whatever it is. So, it is a string of instructions, that are given to you. I want representative portions of this. I can run you know few such portions, and I tell you that well you know, whatever behavior you will



getting from these processor by running these portions, is representative of running the whole thing. Does this problem sound familiar to anybody; no. Similar regions in a sequence; no. Nobody ever has seen such problems. It is not common sub sects exactly. I am looking for similar regions.

Student: Equivalence classes.

But what are the equivalence classes how do I know.

Student: All you also find faces in a running.

Right exactly, I want to find faces in. So, he has used a new term. I do not know if you have heard of term. Faces are program regions, so that usually shows similar behavior. So, one face may come here, that this face may repeat somewhere at there. So, I want to pick up all the distinct faces of the execution, then I have pretty much covered everything, all possible behaviors.

Student: Regions you search for the single this becomes the same.

Yes it is substrates of this whole string yeah right.

Student: So, maximum equity substring or repeating substring.

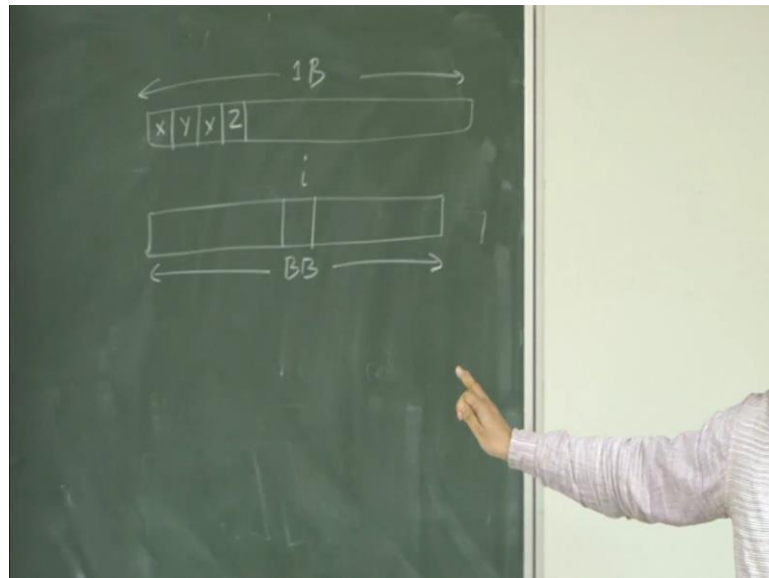
But I want all such substrings.

Student: All substrings, all repeating substrings.

All repeating substrings. So, that is pretty much what the problem is, so there is one tool available, which you can actually download, you can read about, you can go and search in Google; that is called simpont. So, I will tell you what it does. It is a very simple thing. Who does not know basic blocks; raise hands. So, basic blocks are regions of code that has one entry point, and exactly one exit point. So, basic block is essentially a straight line piece of code. You enter here, you exit here alright. So, what simpont does is, it takes this whole instruction sequence; dynamic instruction sequence. Let us suppose this, just for the sake of example; suppose this length is, 100 billion instructions there, this whole execution of the program. So, what it does is that, it asks you that, you know tell me what is a size of the face that you want. Suppose you tell it that I want one billion faces, one billion length, is over each face should be of length one billion instructions.

So, what it does is that, it chops it up into 100 different parts. So, 100 billion will give you, hundred one billion substrings. Now what it does is that, it looks at each of these one billion substrings, and encodes this one each substring, as a series of basic block factors.

(Refer Slide Time: 18:30)



Let us suppose this is my one billion instruction substring. If I take this instruction, the first one, it belongs to some basic block in the program. So, let us suppose it belongs to basic block x. The next instruction probably belongs to basic block y. This one belongs to x again. This may belong to some other z and so and so. So, if you scan through this, what you will get. For each basic block, you are going to get a count; for example, here I have shown a count of two for basic block x. So, at the end what you can do is, you can answer the following question that, in this one billion instruction string, how many times did I visit basic block x, that should be a number. So, it is the vector from this. The length of the vector, is equal to the number of basic blocks that you have in your program, and the entry i tells you, number of times you visited basic block i in this instruction sequence.

So, let us suppose I have any basic blocks in my program, so this is an n dimensional vector. So, i will be going to get hundred such n dimensional vectors, for my 100 billion instruction sequence. So, now, it boils down to cluster in these 100 vectors, in a n dimensional space. And what i will do is that, once I had clustered my cluster will have

similar vectors, and I will pick thus the vector closest to the center of each cluster, and that would be mind representative one billion instruction basis, is it clear. You may not know what how you do clustering, but clustering essentially clubs together the similar vectors. If you just want to run one representative region of one billion instructions, what can you do, can anybody suggest. Just one I want to run, because I can. Suppose somebody tells you that my simulator is so slow that I can only run one billion instructions, not more than that, what can I do.

I have to pick one of these hundred. Which one should I pick.

Student: Most frequent one.

Most frequent one. What do you mean by most frequent, these are all distinct. I have 100 million instruction sequences; I have chopped it up into hundred parts.

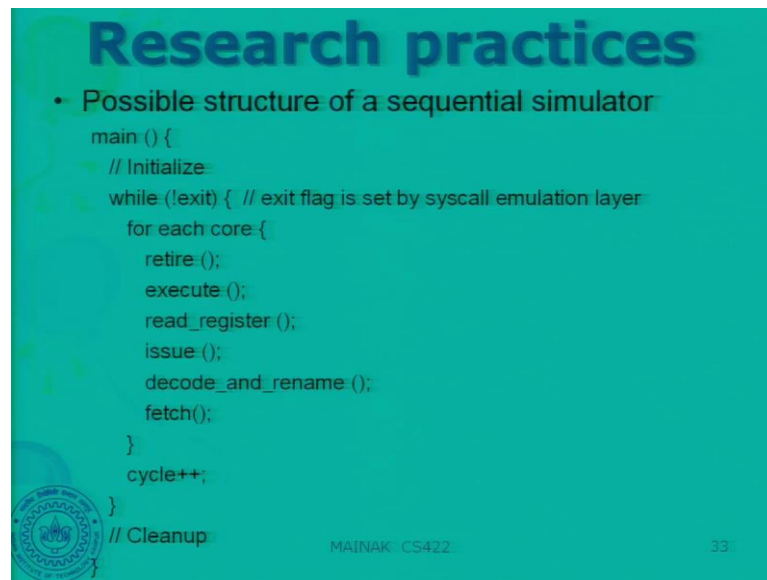
Student: (( ))

The one that is closest to the center of these vectors.

Student: what is one billion, some instruction set of one billion model the (( ))

So, yes here the assumption is that, you are what you execute. So, the piece of code that you are executing, should manifest your behavior; that is what the assumption is here, and it actually follows, pretty accurate comes out in most cases, that the set of instructions that you are executing, roughly tells you your trans prediction behavior, your memory behavior and many other things. It is a very simple algorithm, you can of course improve it, significant impact, but to sought with is a good point. So, representative regions. So, one thing that we assumed here is the parameter; one billion, which simpoint will ask you, that tell me how bigger face you want. So, here you should be careful. These regions should be large enough, to capture different face behavior of an application. So, it should not be too small, and if it is too large then of course, you lose the whole benefit of doing this. You will actually now approaching pretty much similarity in the whole application; any question. So, if you want to read up on this, you can just go and search in Google, simpoint.

(Refer Slide Time: 23:03)



## Research practices

- Possible structure of a sequential simulator

```
main () {  
    // Initialize  
    while (!exit) { // exit flag is set by syscall emulation layer  
        for each core {  
            retire ();  
            execute ();  
            read_register ();  
            issue ();  
            decode_and_rename ();  
            fetch ();  
        }  
        cycle++;  
    }  
    // Cleanup  
}
```

MAINAK CS422 33

So, how does a simulator look like? So, here is a possible structure of a sequential simulator. So, these are the main functions of a simulator, it would first initialize launch of stacks in your simulator. By the way this is over simplified. So, usually a simulator is a several hundreds of thousands of lines of code. It cannot fit in a slide, out of question; that is just a skeleton. So, what you do is, then you run a while loop, which says while not exit, or at the exit flag is usually said by Syscall emulation layer, when your program exists. When program make exit call, this exit flag will be said at the time, until then you keep on running, then what you do. For each code so, I am assuming that you have a multi processor. So, for each co processor code, you would go and do something, you need cycle, what do you do. So, you will first. So, this is a pipe line of a processor, which we have not talked about, we will soon get there. It first retires that, retirement means completion of an instruction.

You would complete any instruction that is pending, in this cycle. Then you would go and execute an instruction, that should be executed in this cycle. You will find out the instruction that are waiting to read registers, you read those registers. You would issue the instructions that are waiting to be issued in this cycle. You decode and rename instructions, will talk about this particular face of the pipe line soon. Decode should be easy to understand, what is rename will talk about. And you will fetch any instruction that should be faced in this cycle. These are distinct pipe line face, and then you do this for each code, and then you the cycle count; that is one cycle simulator, and when the

exit is set, you have essentially stimulated your processor pipe line for this program and you exit. So, we will talk about these pipeline stages very soon. currently just think about these as, some faces of execution, and instructions start such fetch, and leaves the pipeline here.

Student: ups and down (( )) decode issue

Can somebody, are you was actually trying to avoid these question right now, but since he has asked. So, what is pointing out that, an instruction really starts here. It is fetch, and then decoded, then issued, then read register operands, executes, and then completes. Why am I putting it in the opposite order. What would have happened if I put it in this order, in the other order.

Student: While retiring and creating space for the instructions that (( )) fill up the gap (( )) the pipeline and then shifting and all by all (( )) then is already when some instruction that is blocking that is still (( )) fetch (( ))

you talking about free of this is space. So, you are assuming there is some space allocated for the instruction to be fetched.

Student: maybe some resource.

Some resource.

Student: (( ))

I could oversize that, but there is a more important reason why I do not do it. If I put a fetch that side down, this whole thing, what will happen, I fetch an instruction, and when I invoke decode, I will decode that instruction, in the same cycle, but that is wrong. I should not be decoding that in this cycle, should be decoded in the mix cycle. By invoking the pipe line from the end, I actually avoid that problem. I complete the instruction that have to be completed in this cycle. Then I find out what is to be executed in this cycle, I execute them on so and so, and at the end I fetch. It makes sure that an instruction, will not be faced in the cycle and will also get completed in the same cycle; that is how if the pipeline works. We will come back to the structure again later. And there are parallel simulators also. So, here what happens is that, each pipe line stage; these ones, actually run on a different thread of execution, so that is how you get the

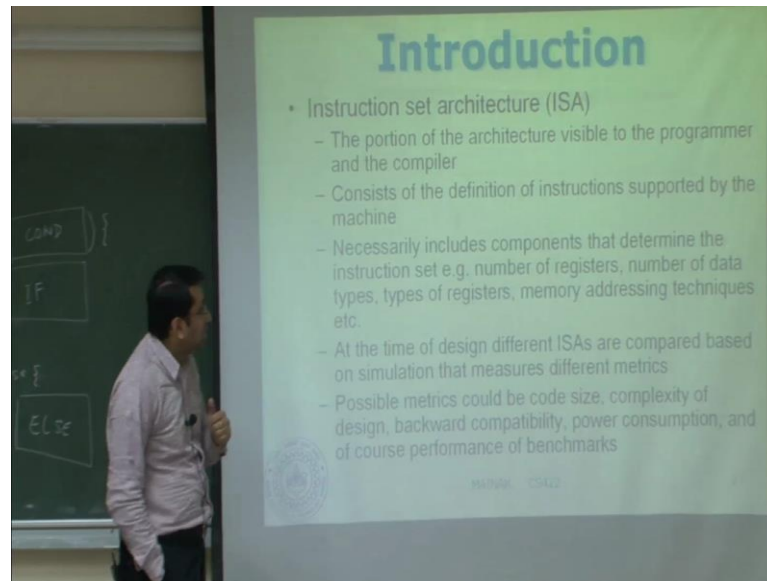
parallelism. So, for each code, each pipeline stage, will be assigned to a different processor, and they will simulate that. So, a clock signal will synchronize the threads etcetera. I am going to skip over these syncs, because there are details which will probably, not even follow right now. We will come back to this later. So, will start something a new. Any question before that, on a material the first lecture. See you should be reading the text, the first chapter, that has more than what we have discussed in the class. It talks about industry trends and all these things, which we have skipped over in the interest of time. So, this one, is again from your book, is the first appendix, and it is in the appendix, because the assumption is that you know this things already.

(Refer Slide Time: 23:48)



So instruction that architecture, is the portion of the architecture, visible to the programmer and the compiler, and it consists of the definition of the instructions, supported by the machine.

(Refer Slide Time: 28:46)



So, whenever you talk about a processor reference manual, this is what usually that manual contains; introductions they targeted. And necessarily includes components that determine the introduction said that is number registers, number of data types, types of registers, memory techniques. At the time of design different I S A s are compared, based on simulation, that measures different metrics. So, at every stage of the design, you will find that a simulator is used. So, that is integral part of design cycle, and almost 50 to 60 percent of your cycle expand on simulating, before you send a design to the factory, because you have to make sure that your design make sense, we could send it to factory. The manufacturing is a very expensive matter. You cannot just send something a design to a factory for fun. Is that I made a mistake that you know, that cost your company a billion dollars actually, so it cannot be done. you spent lot time simulating, to make sure that what your proposing make sense, what you are proposing is correct, what you are proposing, probably brings performance.

So, at this stage we are talking about instructions at architectures, and of course, we will see that there are many options. We have already discussed a couple of options in the last lecture, regarding branches option. So, you have to figure out which way to go, and the only way to figure that out, is to use simulation. You take your bench mark applications, compile them for multiple different I S A's. Run each of the executable on the simulator, and find out which one runs fast. It is not just running faster, it also depends on how complicated your design is. So, possible metrics could be code size, when you compile a

program for particular I S A, how big is my binary; that is a very important thing. complexity of the design; that is a if I have a have an instructions at architecture, that is very compact, but each instruction is overly complicated, so it gives you extremely compact binary, very small, but my design is lousy, gigantic, and complicated, not done. Backward compatibility, that is very important for business. If you are in industry, you are designing a new instructions at architecture, it is important that users, that are using your machines previous versions, should be a able to continue to run those binaries on the new machine, so that is backward compatibility. Power consumptions; how much electric bill, my new I A S would actually make my user pay, that also another important point, and of course, performance of the bench marks.

So, there are various things that you have to balance, when choosing an instructions and architecture. And this is a very important decision, because remember that this is the only thing that will ultimately get exposed, with a compiler designer and programmer. So, if you make a gross mistake here, or how the smarter design you had inside; nothing matter actually. This is very important architecture. Like recall that you know stream copy instruction, that you discuss last time. It would require a very complicated hard ware. Whereas, a offering just one instruction for bite copy, and allowing the programmer we actually orchestrate the stream copy, by running a loop with bite copy instruction, that would of course, explode your code size a little bit, but your design is going to be much simpler.

(Refer Slide Time: 32:42)

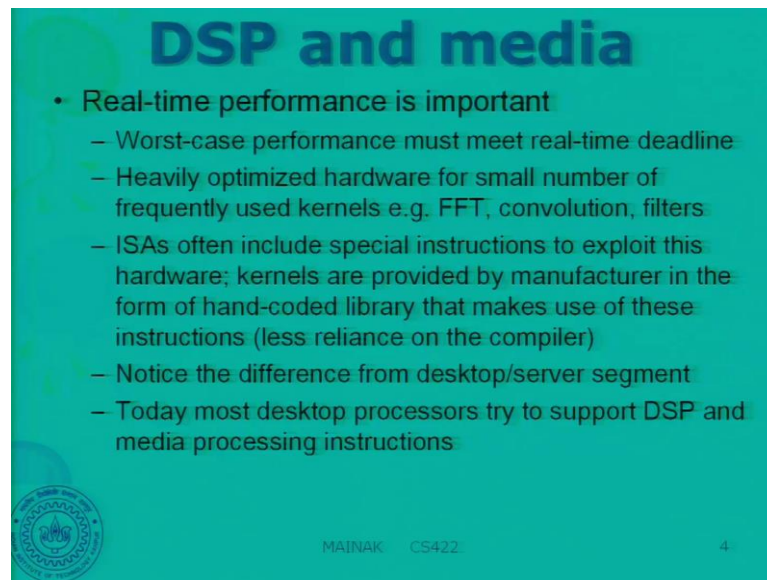




So, just to remind you, we have to keep in mind that there are three market sectors, which we have to (( )). Since the desktop segment performance is very important. So, that is all that a user actually cares about in the desktop, both integer and floating point performance. Power consumption is equally important, I install a desktop in my home, my electric bill cannot be doubled, then of course, that is a no goal. What size is of little or no importance. I can buy, you know d m cards and I can install. Code size is not of that much of important. Then also the reason is that, we do not really run very complicated codes on a desktop as such; that is why the code size is not that large. Backward compatibility is very important for success, because I might have bought certain software, for my previous computer. I get a new computer; that software should continue to run. I do not have to get new licenses; that is very costly matter. In the server market, databases and web services interior performance is much more important. In fact, there will be very little floating point computation in such applications. For supercomputing, on the other hand, floating point performance is more important.

In the embedded market, cost power code size all of these are very important, because here we are talking about things like handbag, your smart phones and all, but your memory on board is very small. So, you cannot have an ISA's that compiles in to a large bind, that is not. You need to have smart compact binaries. You need to have low power, so that batteries may last long. And of course, the design should be low cost. Floating point performance is less important, depending on the of course, application. If you are running, for example, various image processing applications on your handle, of course, floating point performance will be important in those cases. So, keep this in mind when designing your instructions at architecture, because when you design a processor usually what happens to day is that; you would be keeping your instructions at more or less unchanged, and you will be catering to all these three markets. So you have to common ground for all of these.

(Refer Slide Time: 35:14)



## DSP and media

- Real-time performance is important
  - Worst-case performance must meet real-time deadline
  - Heavily optimized hardware for small number of frequently used kernels e.g. FFT, convolution, filters
  - ISAs often include special instructions to exploit this hardware; kernels are provided by manufacturer in the form of hand-coded library that makes use of these instructions (less reliance on the compiler)
  - Notice the difference from desktop/server segment
  - Today most desktop processors try to support DSP and media processing instructions

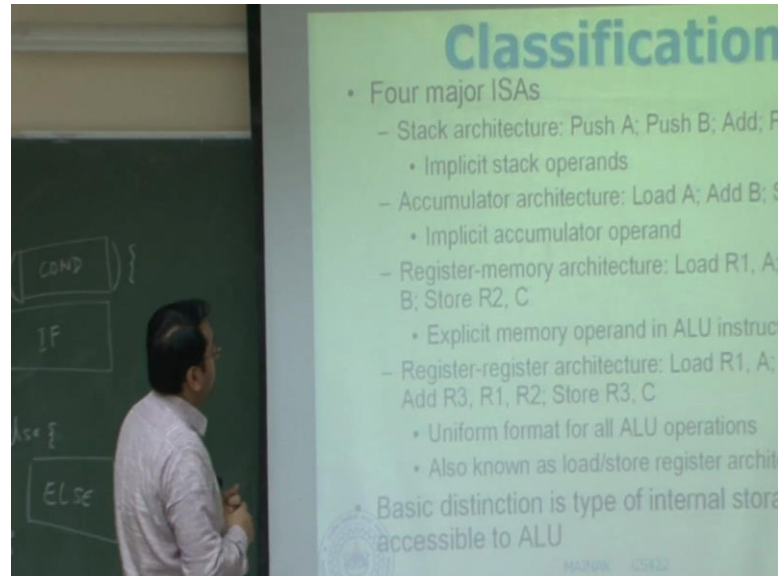
MAINAK CS422 4

And of course, there is a digital signal processors and the media applications, where real time performance is important, worst case performance must meet real time deadline for example, if you are running a codec, it should be real time. It should not take longer; otherwise the user will be denied. I am heavily optimizing hardware for small number of frequent use kernels; like transform, convolution filters, they are occasionally found in such filters. Instructions and architectures, often include special instructions to exploit the hardware. For example, you may often find an instruction for invoking them effectively. So, just one instruction would actually do and effective for it. Kernels are provided by manufacturer in the form of hand coded library that makes use of these instructions. So, here there is less reliance on compilers, but it is improving. The simple reason is that given a piece of code, compiler will have very hard time figuring on that this is FFT that is very difficult actually.

Unstructured piece of code that is somebody has written how does the compiler figure out that is in effective. So, that is why you know the vendors already provide, the libraries for you. You just have to invoke that library function which is actually already been compiled to use this particular instruction to effect the module. So, this is the big deference, from you disturb server segment, where we delight very much on the compiler, which seldom do hand coded assembly program, and today most desktop processors support D S P and media processing instructions. In fact, this has become an integral part of your processors today, and you must be knowing about your M M X and

all these things, multimedia extension of Intel processors. So, they all have these instructions.

(Refer Slide Time: 35:12)



So, will start with classification the instructions at the architectures; stack architecture a essentially is an architecture that has arithmetic operations, and the other two operations to operate in the stack that is push and pump. So, if you look at the sequence of instructions here; push A push B add and pop C. what it does is that. It first pushes the essentially, I want to do an add operation, I want to add A and B so the result with C; that gets compiled into directly. So, I push my first operand, I push my second operand, and then I do add, and the add instruction has to implicit arguments, what are they. The top two elements of this of the stack, and the result goes on the top two stack, then I pop it, goes back to. Is that clear to everybody, the stack architecture. It will only have push pop and arithmetic operation with these basic arguments.

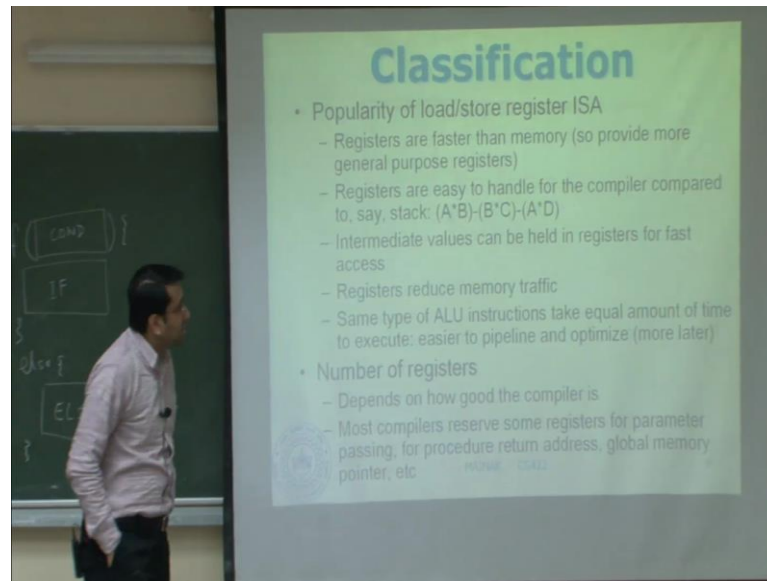
The next one is, accumulated architecture, where, what you do is, you have implicit accumulated operands. So, the same operation  $C = A + B$  gets compiling into that. So, you first do load A so that brings your operand A into an accumulator. Then you do add B. well one operand comes from memory, where these stored. Other operand comes from accumulator which is A, and the result remains in the accumulator. So, that when you find say store C, the operand is picked from that accumulator and send it send to C. So, here the operand was always the top of the stack. Here the operand is always

the accumulator; one of the implicit operands. The third one is the register memory architecture, that your operations can either take register or a memory operand or both. So, R1 A, you load the value of variable a into register R1. So, this is a memory address, this is the register.

Then you say add R2 R1, B. So, always we ride the destination first, and the two sources next. So, it is essentially adding R1 with the memory operand B, and brings the result in R2. So, this is a combined register memory operation. Text one register argument, text one memory argument, and push the result in the register. And then finally, I say store R2,C which picks up the source from R2, push the result back to a memory location C. So, here we have explicit memory operand in a l u instruction. And finally, register architecture, where the only difference from this with this is that; no ALU operation can be done on memory, in this particular architecture, register architecture. Otherwise this is exactly same as this. So, here you also have load R1,A, then load R2, B, and then you had add R3, R1 on R2. So, this add operation here cannot take any memory operand, it all has to be registered, and then you say store R3, C. the result is always stored in register, and then you have to pick it up from there in the store operation.

So, here you have a uniform format for all ALU operations, also known as load store register architecture. Basic distinction is type of internal storage, accessible to the ALU. If you look at all these four, this is really the basic thing that requires. What does the ALU access, when you do an operation. Here it accesses the stack top, the top two or the top most. Here it accesses the accumulator. Here it accesses the register and the memory, and here it access only; that is what the name come from. What type of storage the ALU has added. Any question on this classification. So, today your x 86 ISA belongs to this category, register of memory. It has a lot of register memory operations. we will look at something called risk architecture, that only supports this register, and to contrast with that we talk about sysc also, which is your x 86. So, we will come to that soon, that description.

(Refer Slide Time: 42:17)

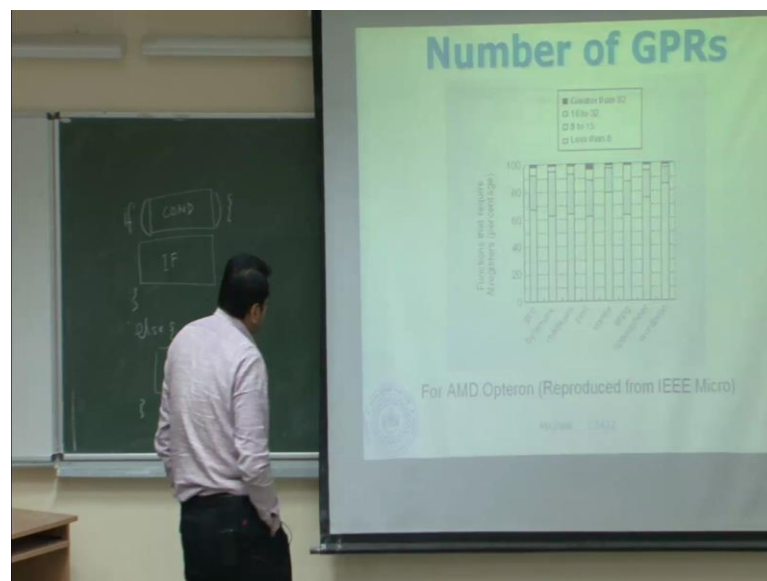


The load store register ISA of the register register ISA, has become very popular, and the simple reason is that, registers are faster than memory. So, you can do very fast ALU operations. As the result there is a push to provide more general proper registers. So, I can do more ALU operations on registers. Registers are easy to handle for the compiler, compare to sales stack for example. So, you should go back home and try to generate stack code of this particular calculation, try to see what kind of code you catch. intermediate values can be held in registers from fast access, that is the big advantage actually, where has been stack architecture, you have to keep on manipulating the top of the stack, to make sure that you intermediate values are not over.

Registers reduce memory traffic, same time of ALU instructions, take equal amount of time to execute, so it is easier to pipeline and optimize for the compilers, and we will come back to that later. So, that is why compiler designers just love register in architecture for this unit. And the simple, there is thumb rule for compiler designer is that, do not give too many options to your compiler designer, as compiler will get confused about which one to pick. So, here it is easy, everything is same; all ALU instructions will take equal amount of time. So, the compiler does not have to debate about, I can do this particular operation in two ways, which one to pick. I will pick anyone it is same, it is a same time. So, in this architecture one important thing, to figure out is how many registers to support in architecture, and it depends very much on how good the compiler is. So, a smart compiler will probably be happy with small number of

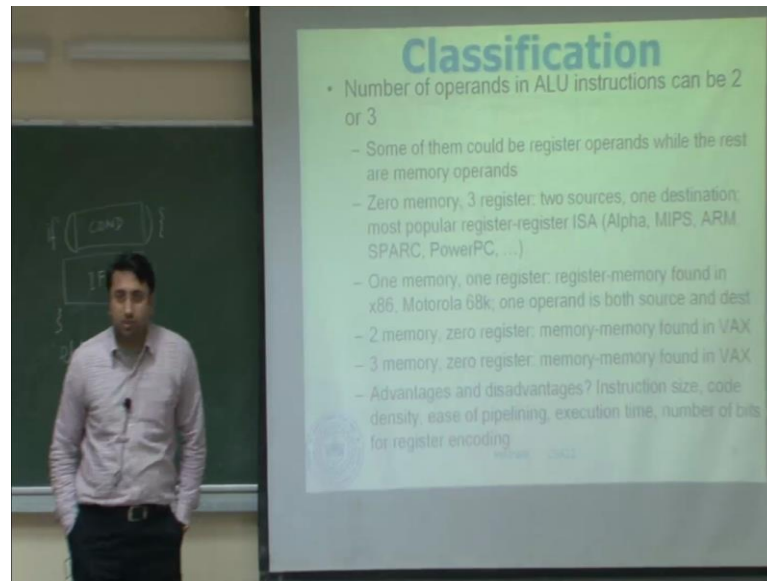
registers, because it can allocate the registers in a smart way. Whereas, a lousy compiler will keep on complaining, you give it hundred registers I need more. So, say it is a very tightly couple of thing, where you sit it down with the compiler designer team, and decide how many registers actually needed. Most compilers resolve some registers for parameters passing to functions, for proceed your return address, global memory pointer etcetera. So, will we elaborate all these when we talk about MIPS.

(Refer Slide Time: 44:50)



So, here is some old data, very old actually, dates back to a 2004-05 time when A M D Opteron most coming out. So, A M D Opteron midder change to their number of registers compare to their previous processor. So, there to figure out how much we increase. So, here I have a several benchmark application, these are from old benchmark. y access shows a functions that require in registers, it is a percentage of that. So, for example, in g c c, if you look at this portion of this bar, it is says that about 70 percent of functions, are happy with less than (( )) that is what it means. So, you can see that you seldom require more than 32 registers, very few actually, and 16 to 32 is also very small. So, this is the kind of data that you normally require you, decide how many registers to support in a machine. So, essentially what they did was that, they simulated a processor with different number of registers, and they ran these application and they generated this data. Now it becomes very clear what you should do. You can do a cause benefit analysis, it goes up. So, in most cases I will pretty much happy to have 15 registers. 15 registers have very good coverage.

(Refer Slide Time: 46:36)



So, continuing with the classification; number of operands in the ALU instructions, that is also important. It can be two or three. So, some of them could be register operands so the rests are memory operands. So, this is not just about your register architecture, register memory architecture is also included here. So, if you have zero memory operand and three register operand, so in that case you would be having two sources and one destination. This is the most popular register register ISA's. So, here I am again classifying your ISA based on number of ALU operands, and the types them. The first classification is that you have zero memory operands, three register operands, out of which towards source one is destination. So, this is a most popular register register ISA, found in alfa MIPS ARM SPARC power pc.

So, you might be surprised to see that x 86 is not there in this list, because it is actually not a register register ISA. One memory, one register; register memory found in x 86 motorola 68 k. And one of these two operands is both source and destinations, so it is a share to source destination. Two memory, zero register; these are memory memory operands found in backs, old machine. Three memory zero register, memory memory found in VAX, that is again old. Advantages and disadvantages, so these have the implication on your instruction size; code density, ease of pipelining, execution time, number of bits for register encoding. Let us go one by one sequence I have chopped it up in to hundred parts. .

The one that is closest to the center of this (( )). Remember that when you are designing an instruction, if you have a memory operand in instruction, there has to be way to encode the address of the memory operands in the instruction. That requires lot of bits; for example, if you want to address four Giga bite of memory, you require 32 bits for that, just for that. So, clearly your instructions that require memory operands, will look large, that will require more space than your register register operands. So, that that is what determines to the instruction size, size of one instruction.

And if you have large instructions, to do the same operation, you will have larger code size also; that is obvious, we have the same instruction count, and then you have memory operands inside the instruction, pipelining becomes difficult, so we will talk about this soon. And your impact your execution time, whereas if you have register register instructions. Of course, the number of bits required to mention, specify a register in the instruction, depends on how many registers you have. For example, you have 32 registers, you require five bits, specify a register. So, that is the number of bits for the register encoding, that essentially log of numbers registers is required. It is much smaller than your memory addresses. So, we will continue from here.

Student: Rate of between the number of instructions required in the register register I S A.

Exactly. So, register memory I S A would require less number of instructions, because it would actually be encapsulating a ALU operation, and the store operation, or a operation and the ALU operation required. So, yes that is right. So, your code density will get affected by that definitely. You will probably have smaller code, but there is trade out. Per instruction you would require larger space, but you may have smaller number of instructions, it is not getting cleared which way you will cover.

Student: How can we (( )) ISA

ISA is the starting point from there you are starting designing your processor; yes, but it is usually a very tightly coupled that you know.