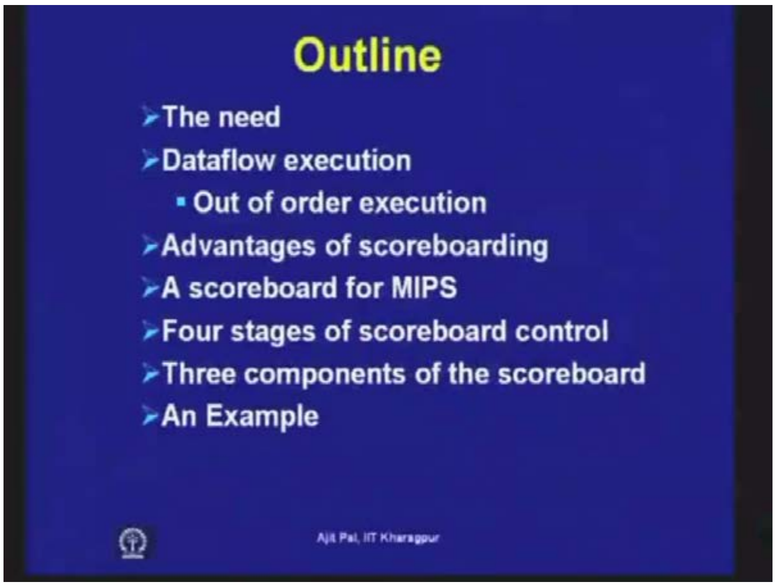
High Performance Computer Architecture

Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 13
Dynamic Instruction Scheduling

Hello, and welcome to today's lecture on dynamic instruction scheduling. Earlier we have discussed in detail about the static instruction scheduling which is done with the help of compiler and we have discussed its limitation. And today we shall start our discussion on dynamic instruction scheduling, and we shall see what are the advantages and disadvantages, and here is the outline of today's lecture.

(Refer Slide Time: 01:26)



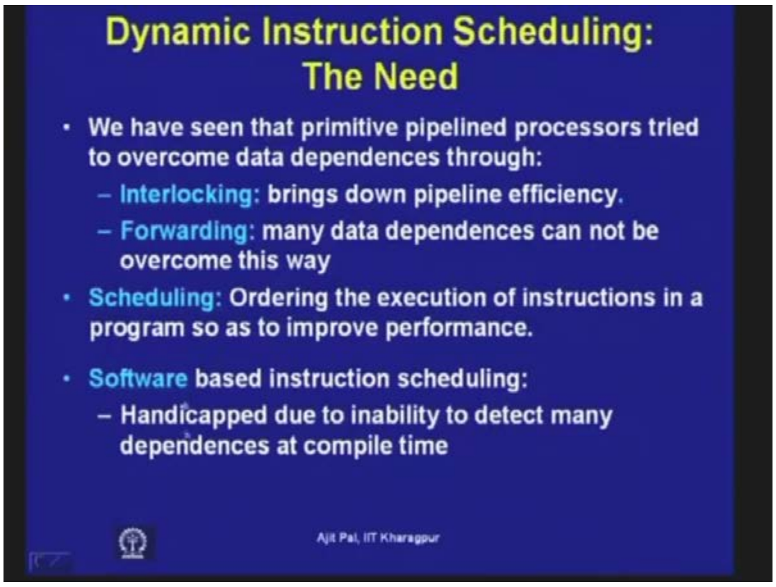
Outline

- The need
- Dataflow execution
  - Out of order execution
- Advantages of scoreboarding
- A scoreboard for MIPS
- Four stages of scoreboard control
- Three components of the scoreboard
- An Example

Ajit Pal, IIT Kharagpur

First I shall discuss about the need of dynamic instruction scheduling why it is necessary? And we shall see it is a kind of data flow execution and which allows you out of order execution. And particularly in this lecture I shall discuss about a technique known as scoreboarding which was developed for CDC 6600. And of course, for CDC 6600 the scoreboard will be quite complicated to discuss in a classroom. So, a simplified version that is for M I P S processor which we have introduced earlier has been considered. And we shall discuss the scoreboard for M I P S which is simplified, but it will definitely highlight the important characteristics of the scoreboarding. Because, both M I P S as well as CDC 6600 are based on load store architecture and we shall see the 4 stages of scoreboard control and 3 components of scoreboard. And illustrate the scoreboard operation with the help of an example

(Refer Slide Time: 02:57)



Dynamic Instruction Scheduling: The Need

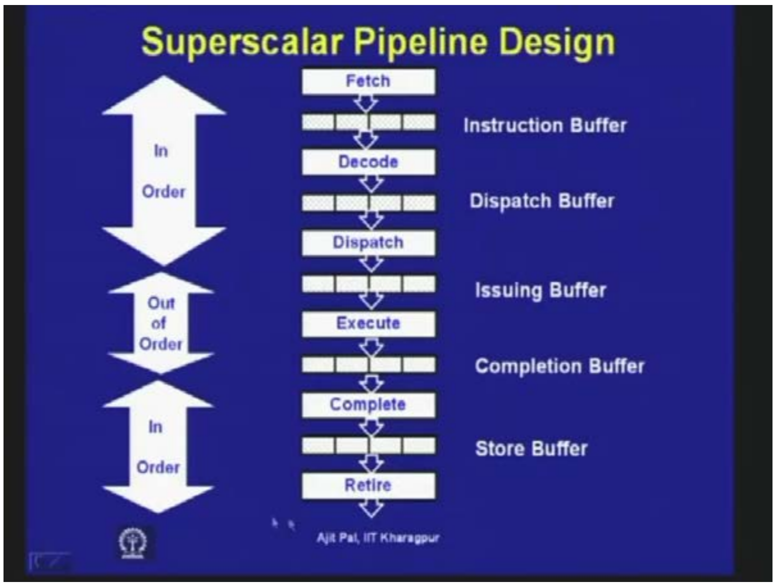
- We have seen that primitive pipelined processors tried to overcome data dependences through:
  - Interlocking: brings down pipeline efficiency.
  - Forwarding: many data dependences can not be overcome this way
- Scheduling: Ordering the execution of instructions in a program so as to improve performance.
- Software based instruction scheduling:
  - Handicapped due to inability to detect many dependences at compile time

Ajit Pal, IIT Kharagpur

Again this is the kind of recap we have seen that primitive pipelined processors tries to overcome data dependences through interlocking. That means whenever there is a hazard it stalls the processor which brings down pipeline efficiency. And we have also discussed a an approach which is known as forwarding which is a hardware based approach where we have seen operands are read form not from the actual registers. But from the pipelined registers and with the help of that the data dependences stalls due to data dependences are minimised or overcome. And we have also discussed scheduling of instructions, software scheduling of instructions software scheduling ordering the execution of instructions in a programme.

So, as to improve the performance we have seen how the data dependences can be overcome or reduced by instruction scheduling. And particularly software based instruction scheduling we have already discussed which can be done with the help of a compiler. And I have mentioned that this software based instruction scheduling is handicapped due to inability to detect much dependence at compile time. Because since it is trying to detect dependences at compile time which will not arise I mean which cannot be detected what will happen at runtime? And as a consequence its usefulness is very limited or restricted.

(Refer Slide Time: 04:36)



Superscalar Pipeline Design

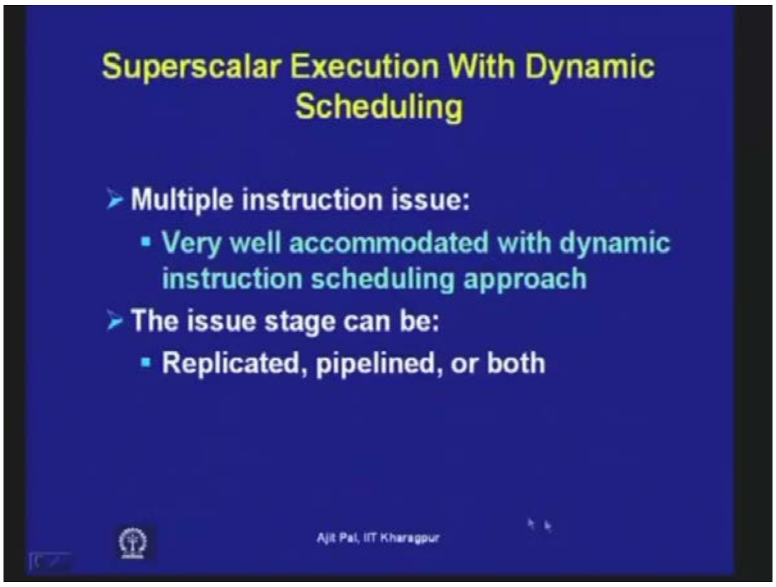
And particularly in the context of Superscalar architecture we shall discuss about this dynamic instruction scheduling the need is arising. Because of you can see the various stages of superscalar processor it will have fetch stage, decode stage, dispatch stage, execute stage, complete stage and retire stage. And as you can see the first part fetch, decode and dispatch up to this it is in order by that in order I mean the order in which instructions are appearing in the programme in the same order they will be read from the programme. And it will be dispatched and then it will be registered known as issuing buffer.

So, it is a multiple entry register and here there is a possibility of out of order issue. That means the order in which instructions appear in a programme that can be changed. And instructions which appear in the programme order at a later point later than another instruction which may be issued for execution earlier. So, it will lead to out of order issue and after the execution will be done with the help of different a number of functional units. Since it is superscalar processor we shall be having multiple functional units and those multiple functional units will have different amounts of latencies.

The latencies of different functional units will not be same the time needed for fixed point addition cannot be same as that of floating multiplication or floating point addition. And as a consequence you will see the execution outputs will be generated out of order. So, out of order issue will lead to out of order execution and what will be done they will be stored in a buffer called completion buffer. And completion buffer will allow you to again produce the result in a in a kind of in order fashions. So, that the results ultimately which is stored in the register or the status of the programme is changed the status of the programme is changed.

In such a way that it will appear as if the instruction execution has taken place in order. So, that is done with the help of you know after the completion of instructions they are stored in a store buffer. And then they are retired that means return into the registers the way the instructions have appeared in the programme order. So, this is how the Superscalar pipeline design will take place and these various functionalities that I have mentioned will be implemented with the help of hardware.

(Refer Slide Time: 07:32)



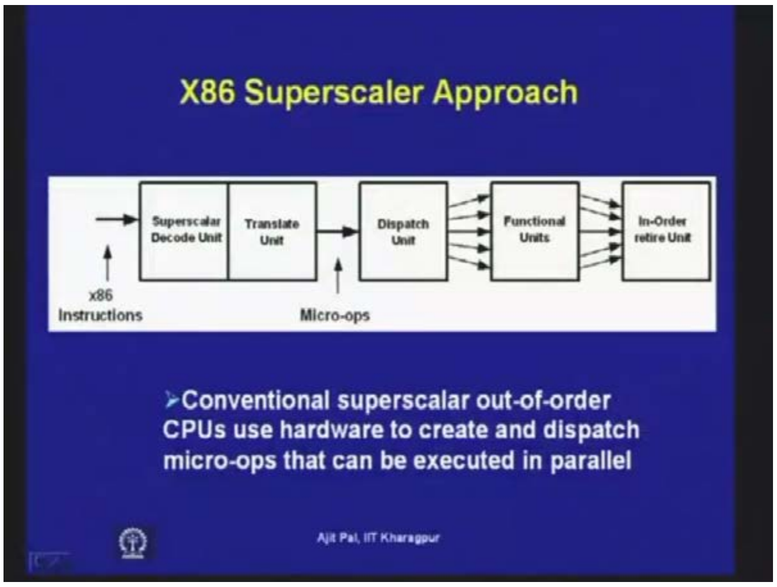
Superscalar Execution With Dynamic Scheduling

- Multiple instruction issue:
  - Very well accommodated with dynamic instruction scheduling approach
- The issue stage can be:
  - Replicated, pipelined, or both

Ajit Pal, IIT Kharagpur

So, particularly this dynamic instruction scheduling is very important in the context of multiple instruction issue which is done is superscalar processor. And the issue stage can be Replicated, pipelined or can be both. Here how it can be done for a C I S C processor is illustrated? As I have already told the techniques that I have been we are discussing is applicable to R I S C processors, having load store architecture.

(Refer Slide Time: 08:27)



X86 Superscaler Approach

Superscalar Decode Unit
Translate Unit
Dispatch Unit
Functional Units
In-Order retire Unit
x86 Instructions
Micro-ops

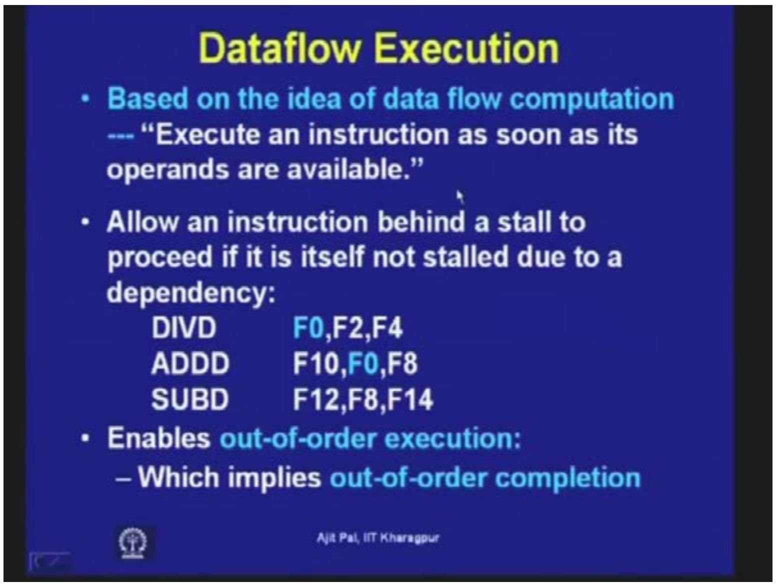
➢Conventional superscalar out-of-order CPUs use hardware to create and dispatch micro-ops that can be executed in parallel

Ajit Pal, IIT Kharagpur

However this approach can also be used to C I S C processor with some modification in the hardware, how is it done is... Shown in this particular diagram, here you have got x 86 instructions then you have got a superscalar decode unit and a superscalar translate unit. So, that instruction decodes stage has been divided into 2 components and then the complex instructions had decomposed into simple R I S C like micro-operations. That means the single x 86 instructions will be decomposed into more than 1 simple R I S C like micro-operations. Those micro-operations are sent to the dispatch unit. So, you can see here we are not exactly executing the instructions of the complex instructions as it appears in x 86.

So, to the dispatch unit various micro-operations or R I S C like operations are being sent which can be issued to the multiple functional units and by the dispatch unit. So, this multiple functional unit will then execute the instructions and obviously here the order can be different. So, this is the in order retire unit which will ultimately produce result and store in the registers the way the instructions have appeared in the programme order. So, this is the in order retire unit so you can see how complex instructions I mean C I S C processors can be adopted to this approach I mean where you can use this R I S C like dynamic instruction scheduling.

(Refer Slide Time: 10:30)



Dataflow Execution

- Based on the idea of data flow computation --- "Execute an instruction as soon as its operands are available."
- Allow an instruction behind a stall to proceed if it is itself not stalled due to a dependency:

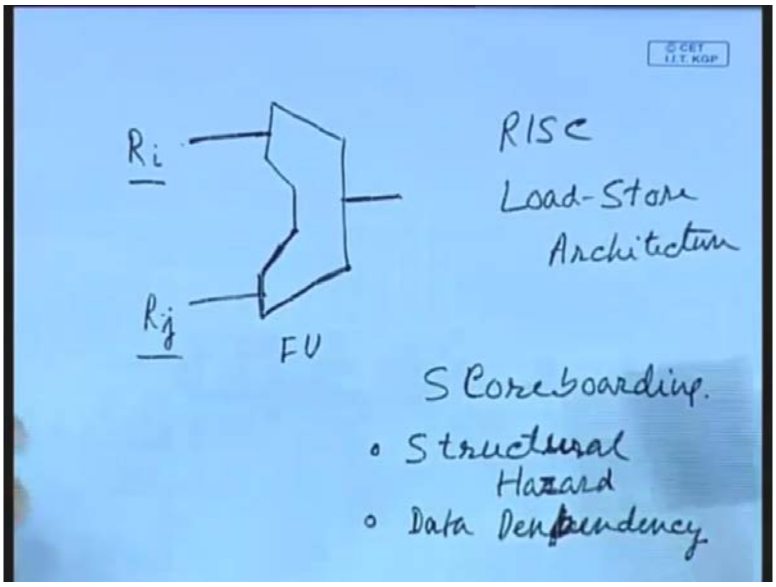
```
DIVD    F0,F2,F4
ADDD    F10,F0,F8
SUBD    F12,F8,F14
```

- Enables out-of-order execution:
  - Which implies out-of-order completion

Ajit Pal, IIT Kharagpur

Now, this dynamic instruction scheduling is based on a very simple idea that is known as data flow computation. What is the basic concept of data flow computation? Basic concept it execute an instruction as soon as its operands are available, you see you have got a functional unit.

(Refer Slide Time: 10:56)



This functional unit will require 2 operands possibly that will come from 2 registers R I and R j. Since, we are considering R I S C processor having load and store architecture load store architecture. So, the operands will be coming the source of the operands are 2 registers R I and R j. The basic idea in this dynamic instruction scheduling is as soon as these operands are available in these registers, execute it provide it to the functional unit or ALU and get it executed. So, this is the basic idea of data flow computation and you may have heard of data flow machine which was proposed by Professor Arvind.

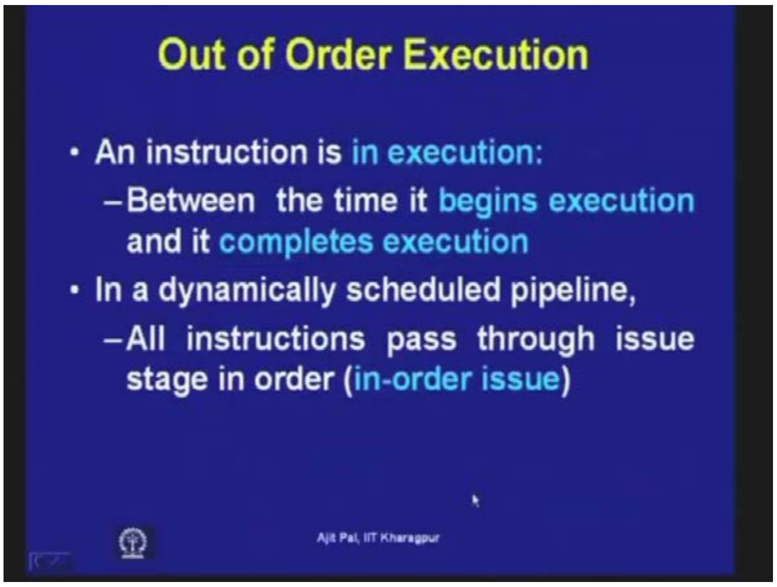
So, there also he used somewhat similar concept, but here it is done for dynamic instruction scheduling that means it is based on execute an instruction as soon as its operand are available. So, it is very easy to say this but it is hard to implement that means when the operands are available in the registers you have to identify the hardware has to identify. And as the instruction execution is progressing it has to closely monitored various boxes operations been performed by different functional unit to keep track of when the operands are I mean execution of some I mean operation is complete.

And it is going to write into a register and as soon as it is written into a register and if it is a source of register it knows that it is available in the register. And so if both the operands are available that execution can be started. So, it has to be done this way so and whenever do it this will allow an instruction behind a stall to proceed if it itself not stalled due to a dependency. That means we have already discussed about the data dependency in case of true data dependency what happens a particular instruction will produce a output which will be consumed by or used by subsequent instruction. So, in such a case obviously there is no alternative, but to stall due to data dependency. However various situations where instructions are not data dependent I mean there is no 2 data dependency.

For example, in this example you have got 3 instructions DIV D double F 0, F 2, F 4 ADD D double F 10, F 0, F 8. Obviously the first 2 instructions that mean ADD D has 2 data dependency on DIV D. So, because it is producing a result which will be available register F 0 which will be used by the subsequent instruction ADD D. However, if you look at it this SUB D has no true data dependency on the previous 2 instructions? That means the first instruction this third instruction is not dependent data dependent on the first 2. However as you can see it is the second instruction is reading from a register and third instruction is also reading from a register. So, although there is no dependency that means it this can what can be done in such a situation after DIV D it is possible to execute this instruction that SUB D.

So, in place of ADD D you can execute SUB D so that will lead to out of order execution and obviously this will lead to out of order completion. But this out of order execution and out of order completion will lead to other types of hazards that read after write hazard is because of true data dependence. But other type of hazards may arise even when there is no true data dependency. So, you have to overcome the other type other 2 types of hazards whenever you allow this out of order execution of completion.

(Refer Slide Time: 16:13)



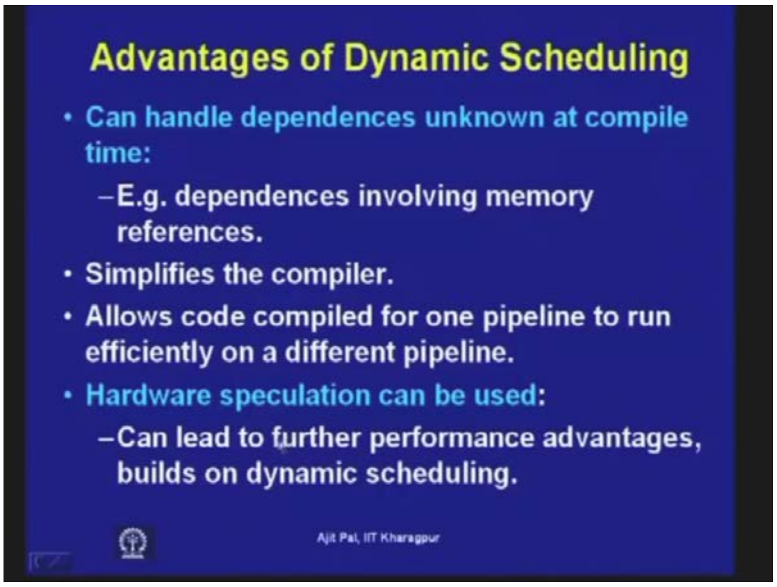
Out of Order Execution

- An instruction is in execution:
  - Between the time it begins execution and it completes execution
- In a dynamically scheduled pipeline,
  - All instructions pass through issue stage in order (in-order issue)

Ajit Pal, IIT Kharagpur

Now, here is some kind of convention and instruction is considered to be in execution between the time it begins execution and it completes execution. So, we shall say that instruction is in execution when it begins execution between the time it begins execution and it completes execution. So, in a dynamically scheduled pipeline all instructions pass through is to issue stage in order as I have already mentioned. So, it leads to in order issue, but it may lead to out of order execution as I have told you.

(Refer Slide Time: 16:54)



Advantages of Dynamic Scheduling

- Can handle dependences unknown at compile time:
  - E.g. dependences involving memory references.
- Simplifies the compiler.
- Allows code compiled for one pipeline to run efficiently on a different pipeline.
- Hardware speculation can be used:
  - Can lead to further performance advantages, builds on dynamic scheduling.

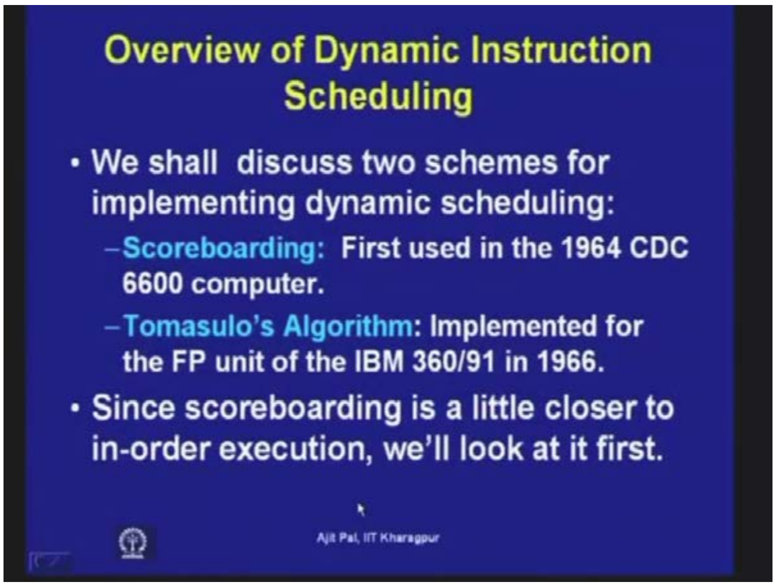
Ajit Pal, IIT Kharagpur

The advantage of dynamic scheduling is that can handle dependences unknown at compile time. I have already mentioned that dependences involving memory references which cannot be detected a compile time. But at runtime you know that particular value will be written into a register from a particular memory location. So, the effective address that is being generated can be same, but the instruction may look different. So, as a consequence this kind of dependences can be handled by dynamic scheduling.

And one another very important consequence is that compiler is simplified it leads to a simplifier compiler. And it also allows code compiled for one pipeline to run efficiently on a different pipeline. Now whenever go for static instruction scheduling that is instruction scheduling has to do instruction scheduling for a particular pipeline in mind. Now, if the pipeline is changed that code compatibility is lost this that programme cannot be executed in another processor having different pipeline.

But in this case whenever you go for dynamic instruction scheduling that problem does not arise because that instruction scheduling and all these things we are doing at runtime. So, if the pipeline is changed then hardware will automatically take care of it. And another approach which I shall discuss that hardware speculation which is used in modern processors can be used in dynamic instruction scheduling which cannot be done in static instruction scheduling. And this particularly this hardware speculation is used to improve the performance of the processor. And this can lead to further performance advantages builds on dynamic scheduling. That means later when I shall discuss hardware speculation we shall see that it cannot really it cannot be based on that static instruction scheduling. It has to be based on it builds on dynamic instruction scheduling.

(Refer Slide Time: 19:42)



Overview of Dynamic Instruction Scheduling

- We shall discuss two schemes for implementing dynamic scheduling:
  - Scoreboarding: First used in the 1964 CDC 6600 computer.
  - Tomasulo's Algorithm: Implemented for the FP unit of the IBM 360/91 in 1966.
- Since scoreboarding is a little closer to in-order execution, we'll look at it first.

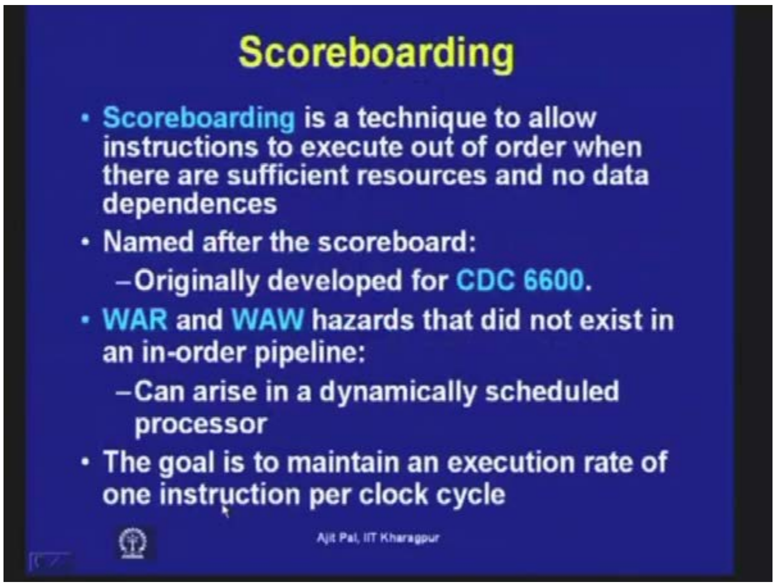
Ajit Pal, IIT Kharagpur

And there are 2 popular schemes which are available for dynamic instruction scheduling which have been which were developed for 2 different very popular processors. First one is known as score boarding so this score boarding was first used for back in 1964. So, in those days there was no concept of concept of software pipelining was not known. And obviously the instruction level parallelism was restricted only to the basic block. And in those days that cache memory and other things were also not present.

So, even those days they developed a technique known as score boarding and that was done for CDC 6600 computer. Of course, CDC 6600 computer has got large number of functional units 11 functional units. But later on we shall explain scoreboarding actually they the scoreboarding name has been taken from this processor CDC 6600. They gave the name scoreboard for this particular hardware based dynamic scheduling approach.

And later on for IBM 360 390 in 1966 another approach was developed by Tomasulo. Tomasulo was a scientist working in IBM and he developed that these approach this dynamic instruction scheduling approach for IBM 360 by 91. This IBM 360 91 was a very popular machine and this approach was developed particularly for improving the performance of floating point unit. And both of these approaches I shall discuss one after the other. But to start with let me focus on scoreboarding because it is little close to in order execution. And later on I shall discuss about this Tomasulo is approach which allows you out of order execution and out of order completion.

(Refer Slide Time: 21:59)



Scoreboarding

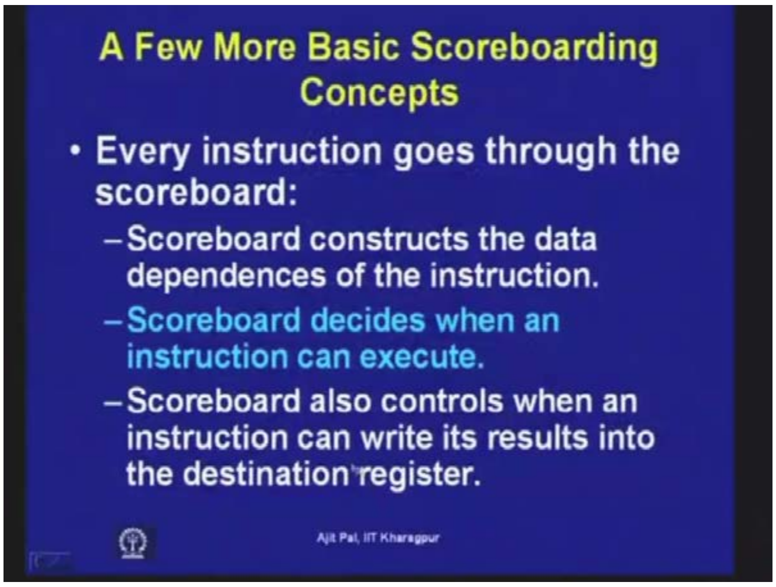
- Scoreboarding is a technique to allow instructions to execute out of order when there are sufficient resources and no data dependences
- Named after the scoreboard:
  - Originally developed for CDC 6600.
- WAR and WAW hazards that did not exist in an in-order pipeline:
  - Can arise in a dynamically scheduled processor
- The goal is to maintain an execution rate of one instruction per clock cycle

Ajit Pal, IIT Kharagpur

Now score boarding is a technique to allow instructions to execute out of order when there are sufficient resources and no data dependences. That means scoreboarding checks 2 things number 1 is structural hazard structural hazard, that means structural hazard is because of limited resources available in the processor. So, if resources are not available obviously then an instruction cannot be scheduled. So, structural hazard is overcome by looking at the resources available in the processor and second thing is that it also checks data dependency true data dependency. So, whenever there is true data dependency so whenever there is true data dependency. Then also instruction is solved however what it does in both the case if enough resources are not available or there is a data dependency.

If an instruction is waiting for result generated by already scheduled instruction then it will be also stalled. So, the way it resolves is by stalling and WAR and WAW hazards that did not exist in order pipeline can arise in dynamic scheduled processors as I have already mentioned. The goal is to maintain an execution rate of one instruction per cycle that was the basic objective of the scoreboarding. That means it will overcome WA R and WAW hazards which can arise in this approach. And of course, basic goal is to maintain execution rate of one instruction per cycle. So, in this particular case every instruction goes through a special hardware known as scoreboard and scoreboard constructs. The data dependences of the instructions and that means it maintains a kinds of database. And with the help of data base it maintains the data dependences.

(Refer Slide Time: 24:51)



A Few More Basic Scoreboarding Concepts

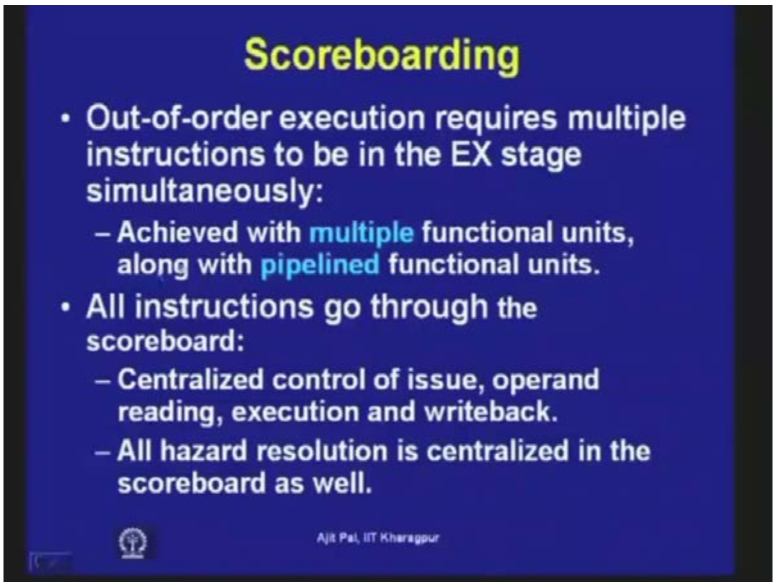
- Every instruction goes through the scoreboard:
  - Scoreboard constructs the data dependences of the instruction.
  - Scoreboard decides when an instruction can execute.
  - Scoreboard also controls when an instruction can write its results into the destination register.

Ajit Pal, IIT Kharagpur

And it can decide only when there are no data dependences it will allow and instruction to execute that means operands are available in the registers. So, another thing I should tell you that you know scoreboarding did not allow forwarding. That means we have already earlier discussed of concept of forwarding where the intermediate results are taken from the pipeline registers, before the results are written into the registers. But in this case you will see it is the results are taken from the register itself not from the pipeline registers. That means bypassing and forwarding technique is not used in the context of scoreboarding. And score boarding also controls when an instruction can write its results into the destination register.

That means whenever you the data has to be registered into a register it will do the writing by avoiding the WAW type of hazards which can arise whenever you go for dynamic instruction scheduling. And this out of order execution requests multiple instructions to be in the execution stage simultaneously achieved with multiple functional units along with pipeline functional units. So, here actually there is no distinction between multiple functional units or pipeline functional unit, both multiple and pipeline functional unit allows you know issue of I mean execution of more than 1 instruction simultaneously. So, logically they will give the same result as you have already seen so in this context.

(Refer Slide Time: 26:42)



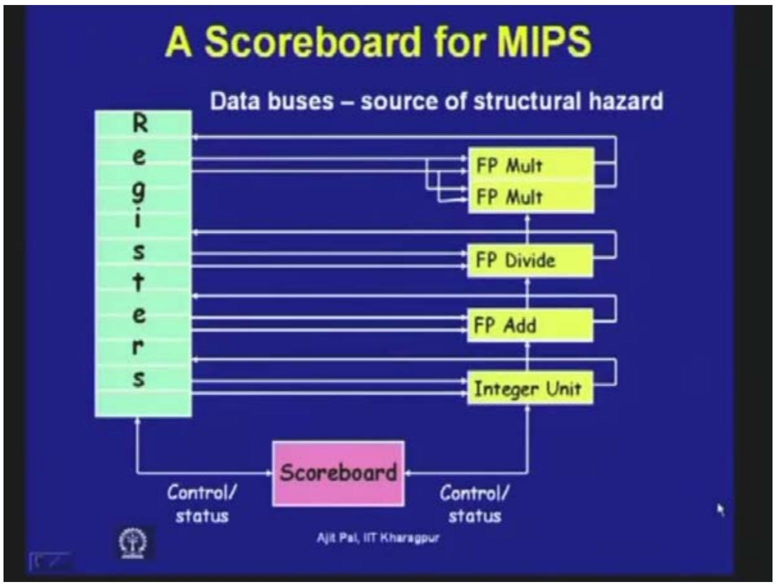
Scoreboarding

- Out-of-order execution requires multiple instructions to be in the EX stage simultaneously:
  - Achieved with multiple functional units, along with pipelined functional units.
- All instructions go through the scoreboard:
  - Centralized control of issue, operand reading, execution and writeback.
  - All hazard resolution is centralized in the scoreboard as well.

Ajit Pal, IIT Kharagpur

It does not really matter whether you are using multiple functional unit or functional units of pipeline. So, all instruction go through this scoreboard which is the centralised control of issue operand reading execution and write back. That means 4 operations like issue of instruction, reading of operands, and execution of instruction, and write back all these are controlled by hardware which is known as scoreboard. And all hazard detection is also centralised in the scoreboard.

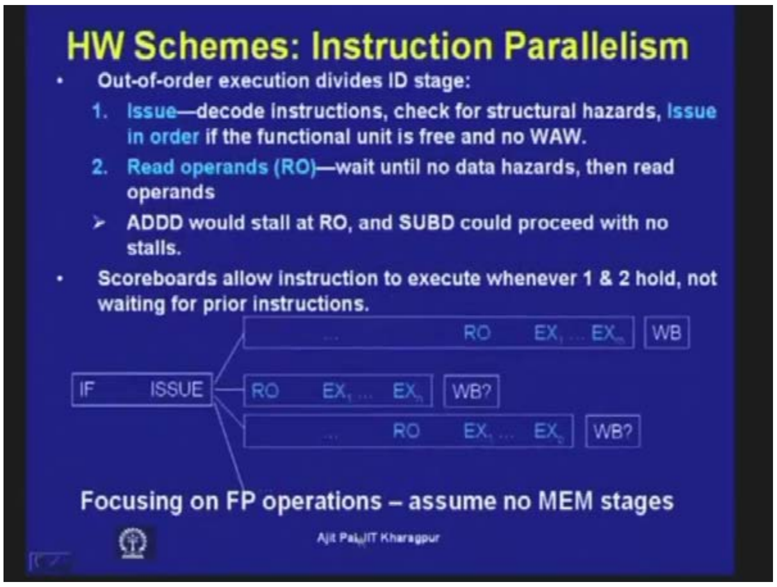
(Refer Slide Time: 27:16)



A Scoreboard for MIPS

And this is the hardware we are eagerly waiting for as you can see for the simplified M I P S scoreboard where you have got only 5 functional unit that CDC 6600 has got 11 functional units for which scoreboard was developed. But this M I P S scoreboard which has been which will be explained has got 5 functional units 2 floating point multiplier, 1 floating point divider floating, 1 floating point adder and 1 integer unit. And here is your register bank and you have got these buses various buses and various buses are available. And this is the scoreboard which controls different functional units and also controls the registers. That means control reading of status controlling the registers controlling the functional units all are done with the help of a centralised hardware, known as scoreboard.

(Refer Slide Time: 28:31)



HW Schemes: Instruction Parallelism

- Out-of-order execution divides ID stage:
  1. Issue—decode instructions, check for structural hazards, Issue in order if the functional unit is free and no WAW.
  2. Read operands (RO)—wait until no data hazards, then read operands
  - ADDD would stall at RO, and SUBD could proceed with no stalls.
- Scoreboards allow instruction to execute whenever 1 & 2 hold, not waiting for prior instructions.

Focusing on FP operations – assume no MEM stages

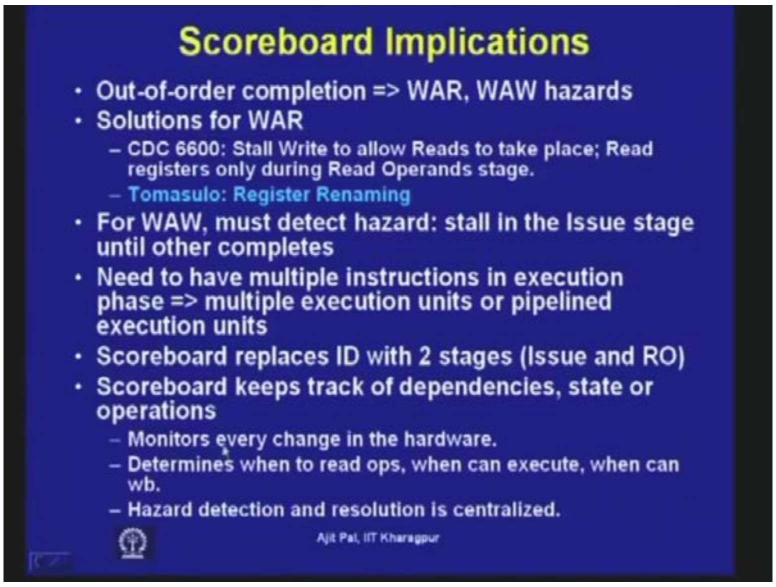
Ajit Pal, IIT Kharagpur

And to allow out of order execution they the id stages instruction decode stage has been divided into 2 parts. The first part is known as issue and this issue stage will decode instructions check for structural hazard. And it will issue in order if the functional unit is free and no write after write. That means if there is no write after write hazard and if the instructions can be I mean if the hardware is available issue will be performed. And the then read operands wait until no data hazards then read operands.

That means here the reading is also taking place with the help of I mean that means reading operation is also delayed I mean until the all the hazards are overcome. And ADD D would stall that read operands and SUB D could proceed with no stalls. As we have already seen that 3 instruction example that means that ADD D stall read operands because of data dependency. But SUB D could proceed with no stalls because there is no data dependency. So, scoreboard allows instruction to execute whenever that first conditions are hold and not waiting for any prior instructions.

So, it is allowing you out of order execution so you can see here it is instruction phase then issue is being done. And different functional units will take different time and after the read operations are performed execution is done. Then the write is performed at different instances of time. And that is controlled by with the help of the scoreboard by avoiding write after read type of hazards in both the cases whenever you perform write operation.

(Refer Slide Time: 30:45)



Scoreboard Implications

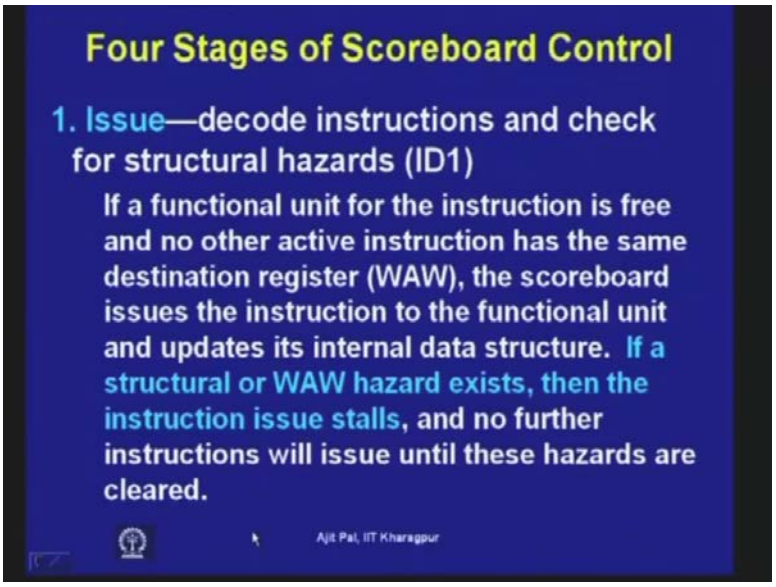
- Out-of-order completion => WAR, WAW hazards
- Solutions for WAR
  - CDC 6600: Stall Write to allow Reads to take place; Read registers only during Read Operands stage.
  - Tomasulo: Register Renaming
- For WAW, must detect hazard: stall in the Issue stage until other completes
- Need to have multiple instructions in execution phase => multiple execution units or pipelined execution units
- Scoreboard replaces ID with 2 stages (Issue and RO)
- Scoreboard keeps track of dependencies, state or operations
  - Monitors every change in the hardware.
  - Determines when to read ops, when can execute, when can wb.
  - Hazard detection and resolution is centralized.

Ajit Pal, IIT Kharagpur

So, out of order completions which may lead to WAR and WAW type of hazards are overcome in CDC 6600 by stalling write stall write to allow read operations to take place. Read registers only during read operands stage that means only after read operands are complete then it is done. And later on we shall discuss about Tomasulo is algorithm where register renaming was done. Register renaming has not been done in this scoreboard and particularly for WAW type of hazard must be take hazard stall the issue stage until other completes.

So, need to have multiple execution in execution phase multiple execution in units or pipeline execution units they are same I mean so far as the functionality is concerned. So, the id stage is replaced by 2 stages I have already mentioned and scoreboard keeps track of dependences and state of operations. That means monitors every change in hardware I mean whether execution is completes and also determines when to read operands when can execute. And when can write back hazard detection and resolution is centralised as I have mentioned.

(Refer Slide Time: 32:13)



Four Stages of Scoreboard Control

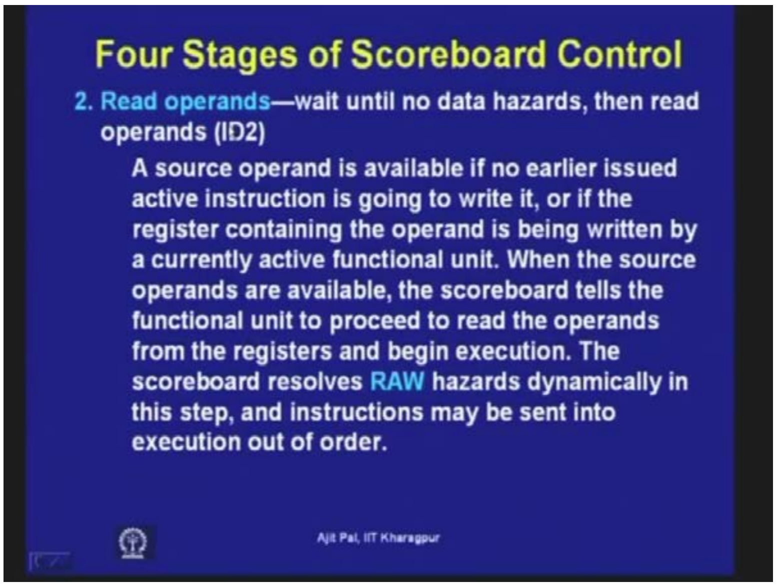
1. Issue—decode instructions and check for structural hazards (ID1)

   If a functional unit for the instruction is free and no other active instruction has the same destination register (WAW), the scoreboard issues the instruction to the functional unit and updates its internal data structure. If a structural or WAW hazard exists, then the instruction issue stalls, and no further instructions will issue until these hazards are cleared.

Ajit Pal, IIT Kharagpur

And it has got 4 stages of scoreboard control number 1 is issue it decode instructions and checks structural hazards as I have already told. If a functional unit for the instruction is free and no other active instructions has the same destination register. That means it keeps track of the already issued instructions. And if the already issued instructions has a destination register which is a source register of a particular instruction then it is stalled.So, this scoreboard issues the instruction to functional unit and updates the its internal data structure if a structural or WAW type of write after write hazard exists then the instruction issue stalls. As I told the solution for this scoreboard is essentially stalling and no further instruction will issue until these hazards are clear.

(Refer Slide Time: 33:16)



Four Stages of Scoreboard Control

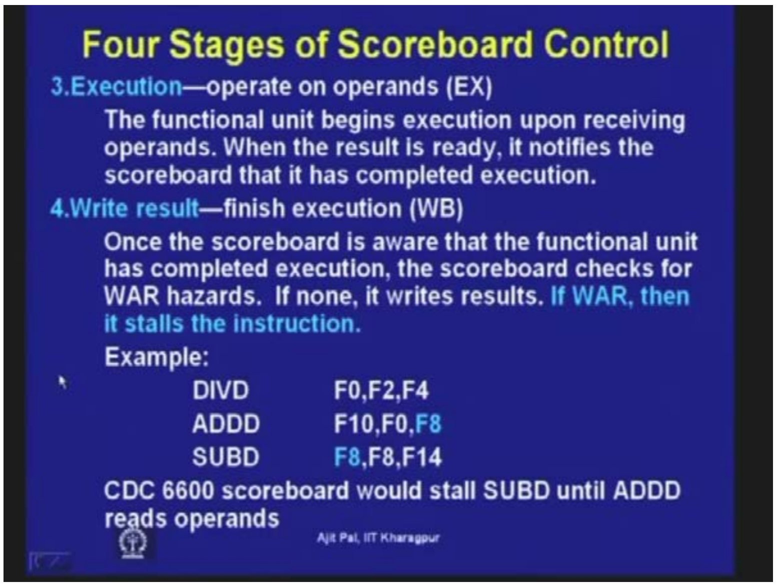
2. Read operands—wait until no data hazards, then read operands (ID2)

   A source operand is available if no earlier issued active instruction is going to write it, or if the register containing the operand is being written by a currently active functional unit. When the source operands are available, the scoreboard tells the functional unit to proceed to read the operands from the registers and begin execution. The scoreboard resolves RAW hazards dynamically in this step, and instructions may be sent into execution out of order.

Ajit Pal, IIT Kharagpur

Then read operands wait until no data hazards then read operands. So, reading of operands is done in the second stage of instruction decode. And it resolves read after write type of hazards dynamically as source operand is available. If no earlier issued active instruction is going to write it or if the register containing the operand is being written by currently active functional units. When the source operands are available the scoreboard tells the functional unit to proceed to read the operands from the register and begin execution. The scoreboard resolve read after write as I mentioned hazards dynamically in this step And instructions may be sent into execution out of order so this is how it is allowing an out of order execution if operands are available.

(Refer Slide Time: 34:12)



Four Stages of Scoreboard Control

3. Execution—operate on operands (EX)

The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard that it has completed execution.

4. Write result—finish execution (WB)

Once the scoreboard is aware that the functional unit has completed execution, the scoreboard checks for WAR hazards. If none, it writes results. If WAR, then it stalls the instruction.

Example:

```
DIVD    F0,F2,F4
ADDD    F10,F0,F8
SUBD    F8,F8,F14
```

CDC 6600 scoreboard would stall SUBD until ADDD reads operands

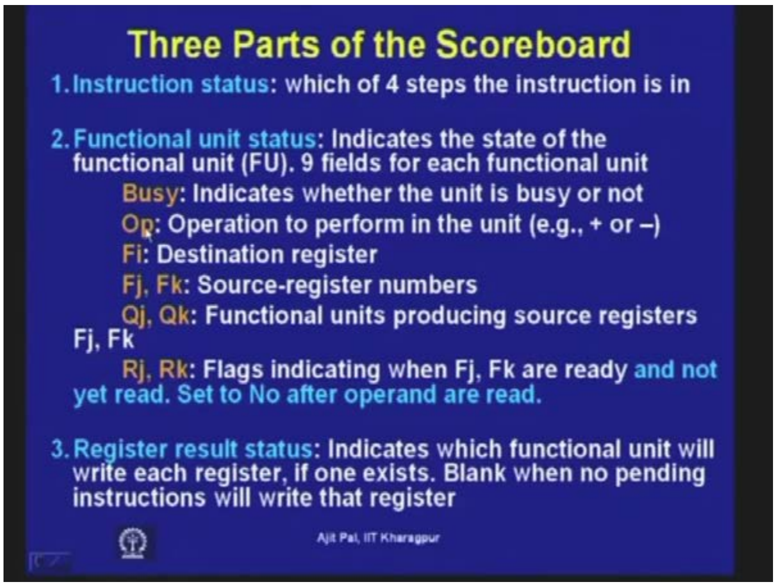
Ajit Pal, IIT Kharagpur

Then the third stage is execution stage and operate on operands the functional units begin execution upon receiving operands when the result is ready it notifies the scoreboard that it has completed execution. And finally come the write back stage it finishes the execution by writing results into the register appropriate registers. So, once the scoreboard is aware that the functional unit has completed execution the scoreboard checks for W A write after read hazard if none it writes into the register in results.

So, you can see dynamically it overcomes w write after read type of hazards and writing is done only when this type of hazard is not available if WAR is then stalls the instruction. So, in this particular example CDC 6600 scoreboard stall SUB D until ADD D read operands. That means you can see there is not 2 data dependency in this particular case it is reading an operand. And it is being used here and here it is writing that means it is a read after write type of operation.

So, this reading this sub d although there is no 2 data dependency. But there is a write after read type of dependency and that type of hazard can occur. So, that is being overcome by stalling these instructions SUB D instruction until reading operation is completed by the second instruction. So, this is done dynamically with the help at the write back stage so these are the 4 stages.

(Refer Slide Time: 36:07)



Three Parts of the Scoreboard

1. Instruction status: which of 4 steps the instruction is in
2. Functional unit status: Indicates the state of the functional unit (FU). 9 fields for each functional unit
   - Busy: Indicates whether the unit is busy or not
   - Op: Operation to perform in the unit (e.g., + or –)
   - Fi: Destination register
   - Fj, Fk: Source-register numbers
   - Qj, Qk: Functional units producing source registers Fj, Fk
   - Rj, Rk: Flags indicating when Fj, Fk are ready and not yet read. Set to No after operand are read.
3. Register result status: Indicates which functional unit will write each register, if one exists. Blank when no pending instructions will write that register

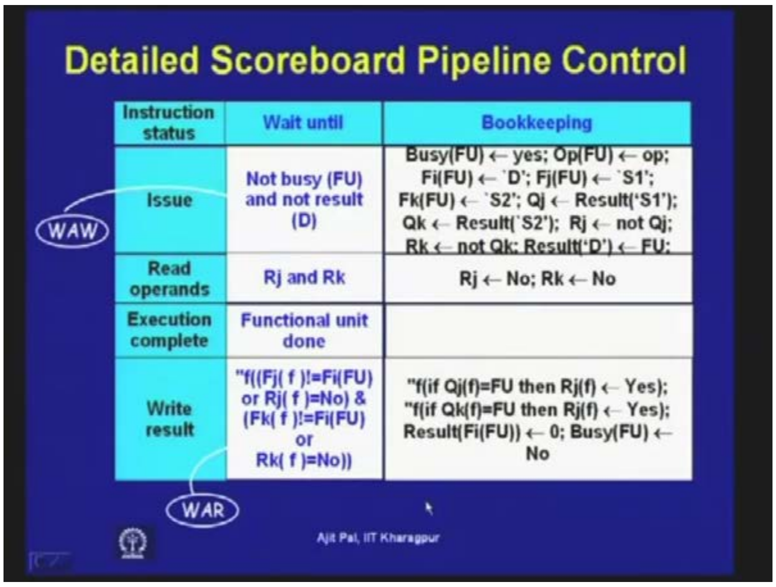
Ajit Pal, IIT Kharagpur

Now in addition to these stages it has got 3 three different parts to maintain the database. So, first of all instruction status so which of the 4 steps of the instruction is in instruction is in. So, for a, for the instructions which have been issued they can be in different stages instruction issue reading operands execution or write back. So, it keeps a it maintains a database about in which stage a particular instruction is in. And then functional unit status it indicates the state of the functional unit 9 fields for each functional unit. So, you can see database is quite completed so that different functional unit it has got there are 9 fields busy then operation to perform.

Operation can be addition subtraction multiplication divide and so on. Then F I the destination register for a particular functional unit. And F j and F k they are as source register numbers and Q j and Q k functional units producing the source register. So, you can see not only keeps track of the source register numbers, but which functional units will produce the result. And write into the registers that is also I mean maintained in this data base functional units status database. And R j and R k flags indicating when F j and F k are ready and not yet read that means the functional units may be, but the operands have not yet been read that is being maintained with the help of this R k and R j flag bits.

So, there are 7 flag bits busy O p, F I, F j, F k, Q j, Q k and R j R k we shall see how they are being used when instructions are in flight and then you have got register result status. So, there are 32 registers and from which functional units these registers are being written indicates which functional unit will write each register. If one exists black when no pending instructions will write into the registers that means the registers will be written by some functional unit. So, it is keeping track of which functional unit will write into which register. So, this is the database that is being maintained.

(Refer Slide Time: 39:00)



Detailed Scoreboard Pipeline Control

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result (D)	Busy(FU) ← yes; Op(FU) ← op; Fi(FU) ← 'D'; Fj(FU) ← 'S1'; Fk(FU) ← 'S2'; Qj ← Result('S1'); Qk ← Result('S2'); Rj ← not Qj; Rk ← not Qk; Result('D') ← FU;
Read operands	Rj and Rk	Rj ← No; Rk ← No
Execution complete	Functional unit done	
Write result	"f((Fj(f)!=Fi(FU) or Rj(f)=No) & (Fk(f)!=Fi(FU) or Rk(f)=No))	"f(if Qj(f)=FU then Rj(f) ← Yes); "f(if Qk(f)=FU then Rj(f) ← Yes); Result(Fi(FU)) ← 0; Busy(FU) ← No

WAW

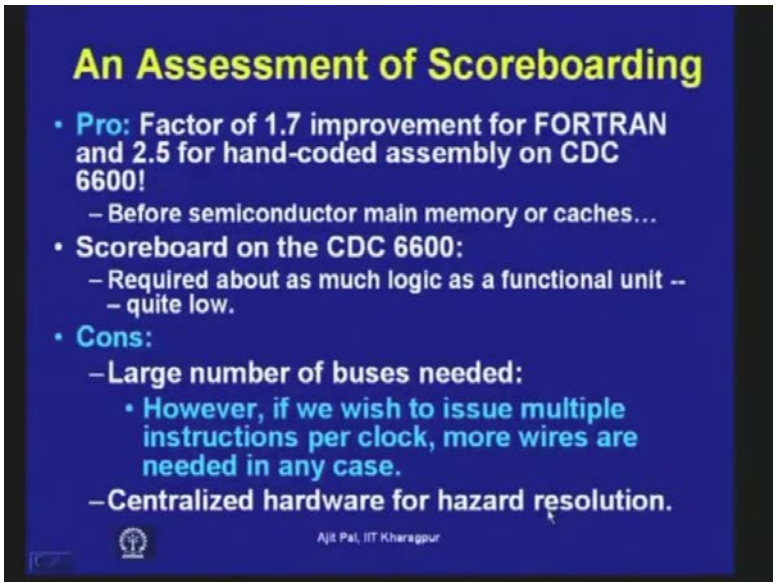
WAR

Ajit Pal, IIT Kharagpur

And you can see this is the detailed scoreboard control pipeline control these are the is a instruction status. And it will wait until functional units are not available and results are not available only when functional units are available results are available instructions are issued. And the various book keeping that is being done that means to maintain those you know that 7 flag bits that is being shown here. So, busy O p, F I, F j, F k, Q j, Q k, R j, R k and how they are how they are getting of the result. And doing the necessary book keeping it is read operands R j and R k no or yes that is being done wait until these are available. And execution complete that functional unit is whenever execution is complete a functional unit is released.

And so if it is not busy a functional unit is released then here is the write result how when the writing of result has to be delayed that is being mentioned here. And there are various conditions it will do the book keeping and wait until the results can be written in a proved register. And as a consequence what is being done particularly in the issue stage WAW type of hazards are overcome and in the write back stage WAR type of hazards are overcome. And of course, that the most common type the read after write that that hazards which are essentially representing 2 data dependency those hazards are overcome by stalling because if operands are not available. Then stalling has to be done if the functional units are not available that is your structural hazard then stalling has to be done so these are being done.

(Refer Slide Time: 41:12)



An Assessment of Scoreboarding

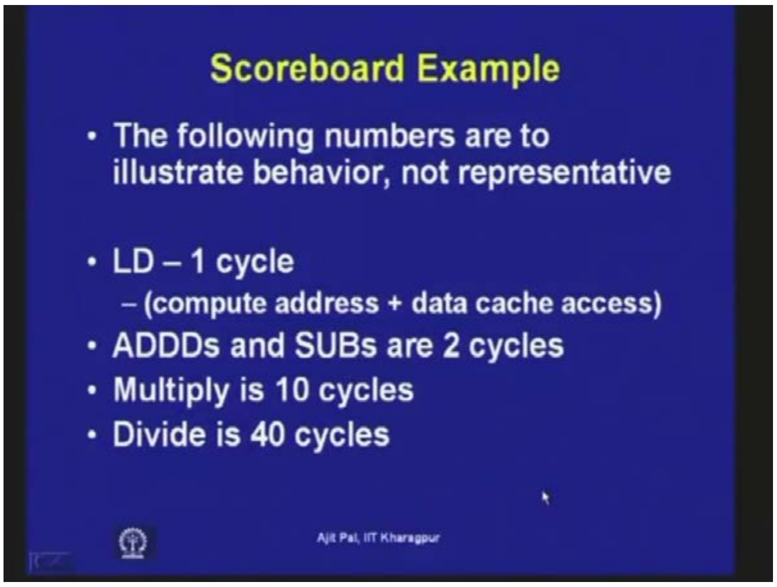
- Pro: Factor of 1.7 improvement for FORTRAN and 2.5 for hand-coded assembly on CDC 6600!
  - Before semiconductor main memory or caches…
- Scoreboard on the CDC 6600:
  - Required about as much logic as a functional unit -- quite low.
- Cons:
  - Large number of buses needed:
    - However, if we wish to issue multiple instructions per clock, more wires are needed in any case.
  - Centralized hardware for hazard resolution.

Ajit Pal, IIT Kharagpur

It has and for CDC 6600 there was improvement of 1.7 factors of 0.1 of 7 improvement for FORTAN and 2.5 for hand-coded assembly. And of course, this was done as I mentioned before main memory or cache memory I mean cache memory were available. And for CDC 6600 surprisingly the hardware was not complex only equivalent to a single functional unit. However 1 very disadvantage is that large number of buses needed we have seen even for 5 functional unit you have got a large number of buses, because you have to do parallel reading and writing.

So, number of buses is quite large however if we want to issue multiple instructions per clock more wires are needed. In any case so centralised hardware for hazard then another thing is the scoreboard effectively handles 2 data dependences minimises the number of stalls due to 2 data dependences. And anti-dependences and output dependences are also handled using stalls and we have seen which is done by the issue and write back stages.

(Refer Slide Time: 42:45)



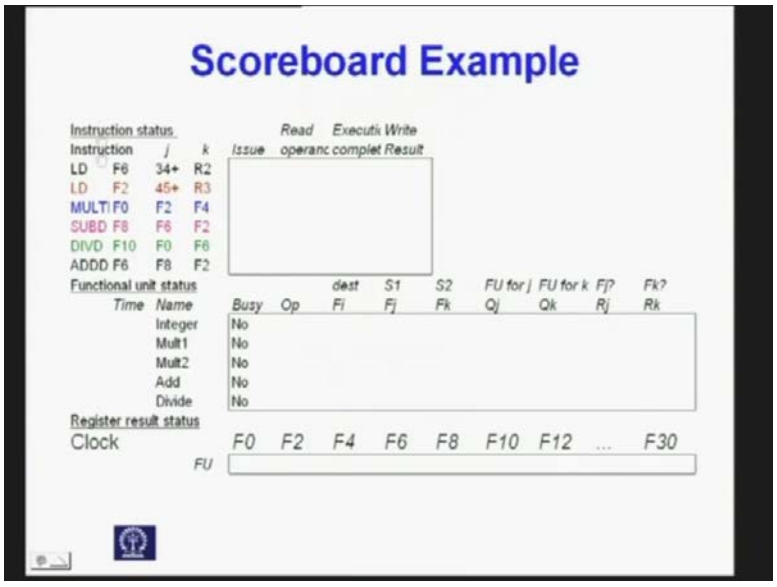
Scoreboard Example

- The following numbers are to illustrate behavior, not representative
- LD – 1 cycle
  - (compute address + data cache access)
- ADDDs and SUBs are 2 cycles
- Multiply is 10 cycles
- Divide is 40 cycles

Ajit Pal, IIT Kharagpur

Now let us consider illustrate the operation with the help of an example and to illustrate the example we shall consider that the load has a 1 cycle latency and ADD D and SUB B addition and subtraction has 2 cycles latency multiply has got 10 cycles latency. And divide has got forty cycles latency some realistic numbers have been taken just to illustrate the example.

(Refer Slide Time: 43:15)



Scoreboard Example

Instruction status

Instruction		j	k	Issue	Read operanc	Executi complet	Write Result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTI	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

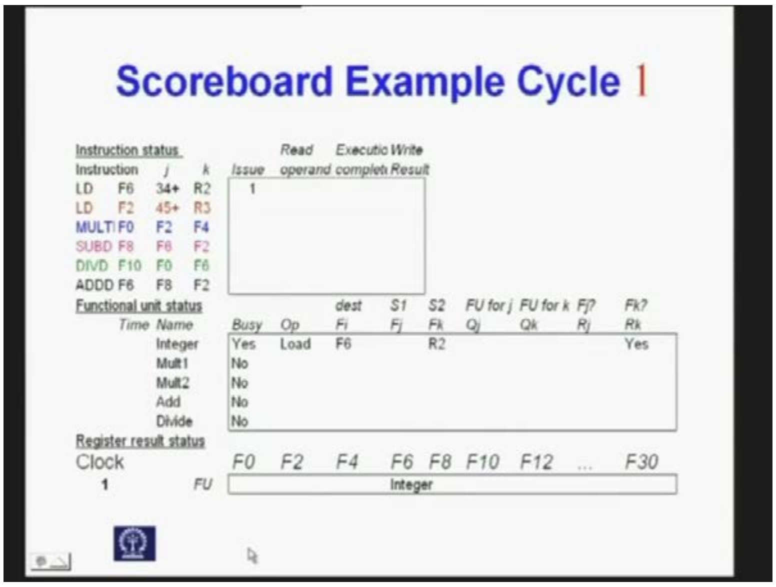
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
	FU									

Now here is the scoreboard, where you can see there are 3 stages an instruction status, functional unit status and register result status. And these are the instructions to be executed load there are 2 loads followed by multiplication, subtraction, division, addition. So, these are the instructions essentially straight line code instruction which is a basic block it cannot operate beyond basic block. So, these are the instructions to be executed and right now you the instruction status functional unit status register result status all are empty. Now, let us start execution with the help of scoreboard.

(Refer Slide Time: 44:03)



Scoreboard Example Cycle 1

Instruction status

Instruction	j	k		Issue	Read operand	Executio complete	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTI	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

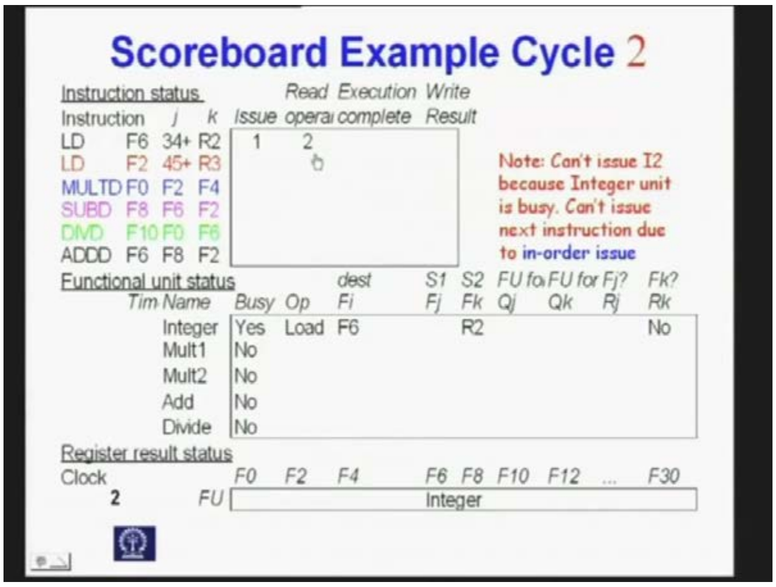
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU				Integer					

So, after first cycle the instruction is issued. So, here it shows it has been issued and functional unit status shows that this that integer unit has become busy. So, it is yes it is busy and instruction to be performed operation to be performed is load and destination register here is F 6. So, F 6 is written here and source register is R 2. So, you can see how the database is updated as you go to the cycle number 1 and here the functional unit that is busy that register result status here F 6. So, it will come from the integer unit so functional unit which will produce the result is shown here. So, that is after the first cycle.

(Refer Slide Time: 45:01)



Scoreboard Example Cycle 2

Instruction status

					Read	Execution	Write
Instruction		j	k	Issue	operai	complete	Result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Note: Can't issue I2 because Integer unit is busy. Can't issue next instruction due to in-order issue

Functional unit status

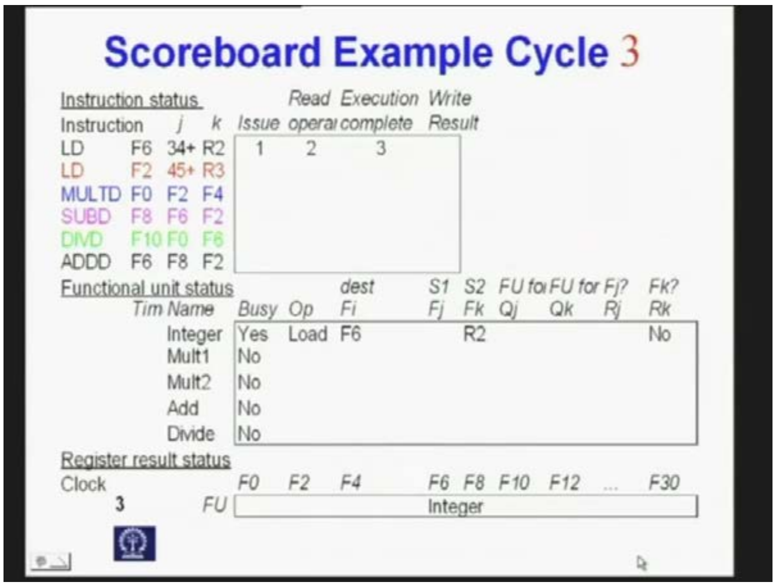
				dest	S1	S2	FU fo	FU for	Fj?	Fk?
Tim	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	FU				Integer					

After second cycle you can the operation this particular operation that second that operand read operands has taken place for in the second cycle. But cannot issue I 2 because integer unit is busy it has got only one integer unit. And so the load the second load instruction cannot be unfortunately issued, because here it is a lead to a kind of structural hazard. So, because of the structural hazard in a second cycle the second instruction cannot be issued. So, next instruction due to due to in order issue, so it has not anything much has changed except it has this particular status has changed.

(Refer Slide Time: 45:57)



Scoreboard Example Cycle 3

Instruction status

Instruction	j	k	Issue	Read operands	Execution complete	Write Result
LD F6	34+	R2	1	2	3	
LD F2	45+	R3				
MULTD F0	F2	F4				
SUBD F8	F6	F2				
DIVD F10	F0	F6				
ADDD F6	F8	F2				

Functional unit status

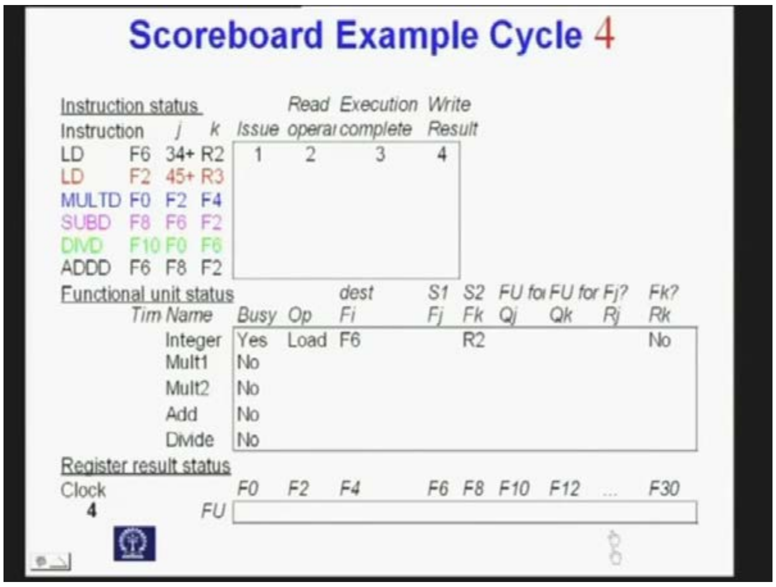
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU				Integer					

So, we go to the third cycle execution is completed and execution is completing, because it requires 2 instruction. And you can see how the status is being changed here I mean not much has been changed compared to previous thing, but only this has changed.

(Refer Slide Time: 46:25)



Scoreboard Example Cycle 4

Instruction status

Instruction	j	k	Issue	Read operands	Execution complete	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3				
MULTD F0	F2	F4				
SUBD F8	F6	F2				
DIVD F10	F0	F6				
ADDD F6	F8	F2				

Functional unit status

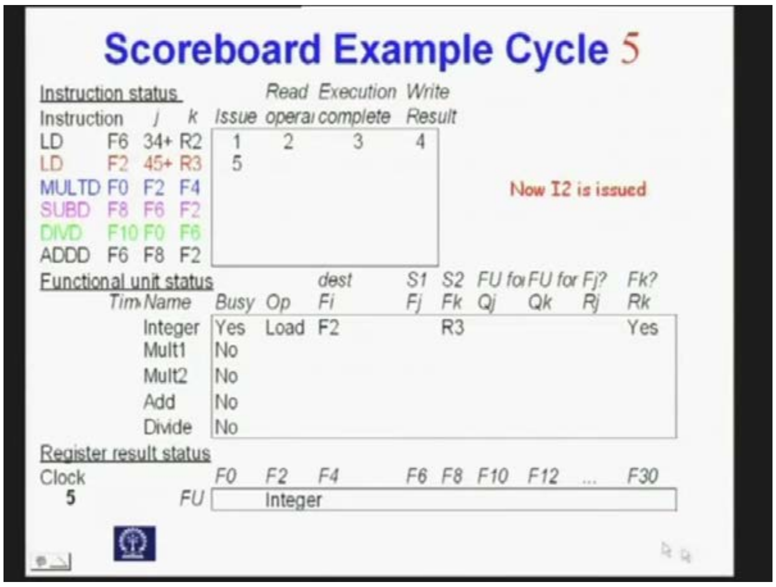
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F6		R2				No
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU									

And it has gone to the write back stage it will write result into F 6 and after the writing of result. You can see here that that functional unit is no longer written here, because now the functional unit is released after it has completed that functional unit is released and after it has completed the writing of result into the appropriate register R 6. So, here it is not also shown now it wills we shall go to the fifth cycle. As we go to the.

(Refer Slide Time: 47:00)



Scoreboard Example Cycle 5

Instruction status

Instruction	j	k	Issue	Read operai	Execution complete	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5			
MULTD F0	F2	F4				
SUBD F8	F6	F2				
DIVD F10	F0	F6				
ADDD F6	F8	F2				

Now I2 is issued

Functional unit status

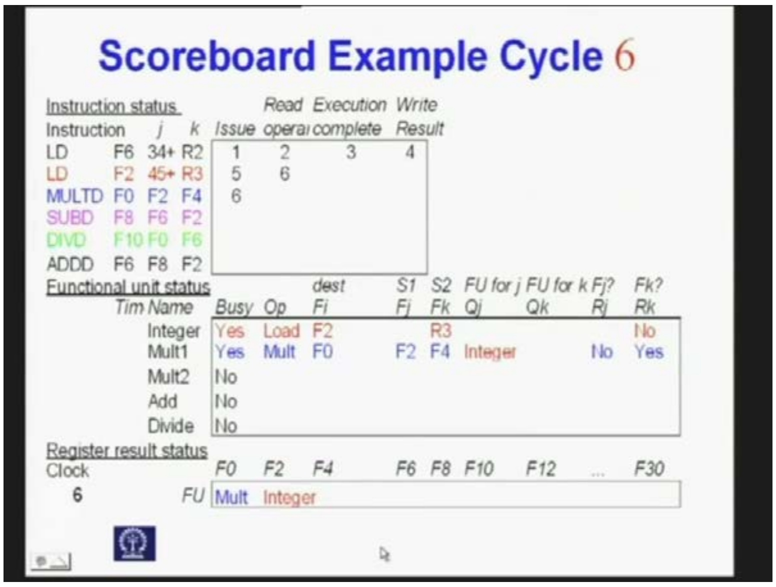
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU foi Qj	FU for Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU		Integer							

Fifth cycle the second load instruction is issued and second load instruction is issued and you can see here this F 2 is the register in which result will be written. So, functional unit is integer functional unit and the F 2 register will be written by this functional units and integer unit is again busy it is performing. This load operation and destination register is F 2 and source register is R 3 and this R k is yes as it is shown here. And now, let us go to the fifth cycle.

(Refer Slide Time: 47:45)



Scoreboard Example Cycle 6

Instruction status

Instruction	j	k		Issue	Read operai	Execution complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULTD	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status

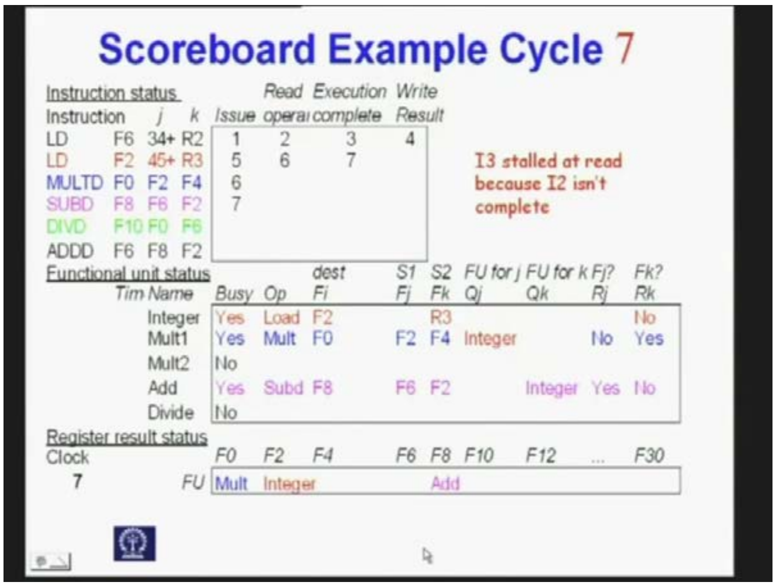
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult	Integer							

Sixth cycle now; here you see the third instruction has been issued. Because there is no true data dependency and functional unit required is different. So, here it requires a multiplier so since the multiplier is become busy now and various components like destination register source register. And various other things are filled up appropriately in this data base and this result register status is also properly mentioned. So, it has gone to the sixth cycle.

(Refer Slide Time: 48:28)



Scoreboard Example Cycle 7

Instruction status

Instruction	j	k	Issue	Read operai	Execution complete	Write Result
LD	F6	34+ R2	1	2	3	4
LD	F2	45+ R3	5	6	7	
MULTD	F0	F2 F4	6			
SUBD	F8	F6 F2	7			
DIVD	F10	F0 F6				
ADDD	F6	F8 F2				

I3 stalled at read because I2 isn't complete

Functional unit status

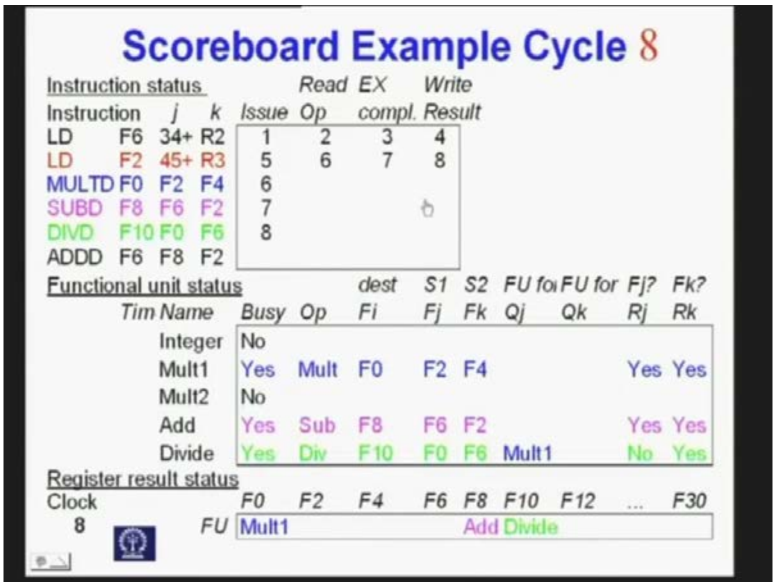
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	Yes	Load	F2		R3				No
	Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
	Mult2	No								
	Add	Yes	Subd	F8	F6	F2		Integer	Yes	No
	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult	Integer			Add				

Now, it will go to the seventh cycle in the seventh cycle it will proceed to I mean sequentially operand read it has completed execution it is it will complete. And it will now issue the forth instruction, because we have got one divide so that subtraction so we that integer this is available now. So, this particular functional unit is getting busy and corresponding fields of those 7 fields are being filled up appropriately. Operation to be performed destination register source register and so on and you can see various register. I mean which will be written by different functional units are maintained by this register result status. So, this is a clock cycle 7.

(Refer Slide Time: 49:31)



Scoreboard Example Cycle 8

Instruction status

Instruction	j	k	Issue	Read Op	EX compl.	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULTD F0	F2	F4	6			
SUBD F8	F6	F2	7			
DIVD F10	F0	F6	8			
ADDD F6	F8	F2				

Functional unit status

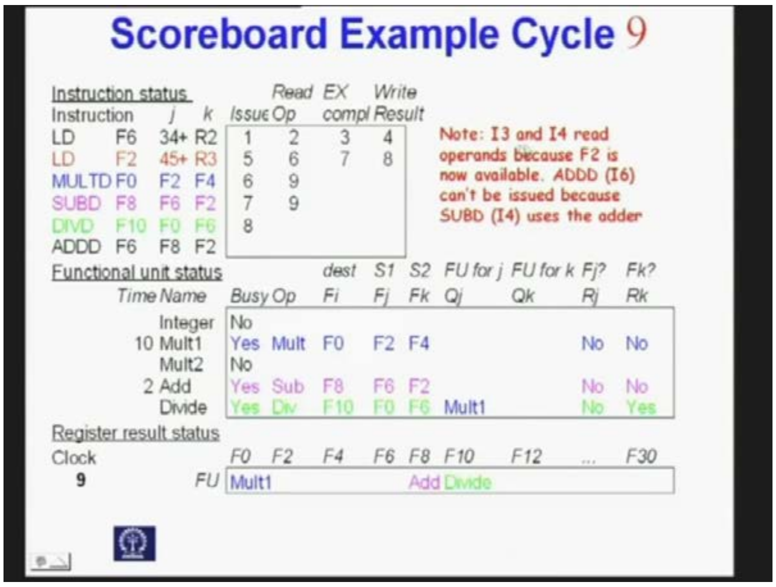
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU foi Qj	FU for Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	FU	Mult1				Add	Divide			

Now, we go to the clock cycle 8 as we go to the clock cycle 8 the integer unit will complete its execution it will write the result into the register F 2. So, you can see here there is no longer there is a mention about the integer unit and integer unit has become free. So, here it is not yes it is no longer busy so integer unit is now free. But it has already issued the remaining 3 instructions they will be in different stages of completion. Because they are we have seen that multiplication takes longer time. Now let us see although the instruction multiplication double instruction was it should, but F 2 and F 4 you see F it was waiting for the result for to be written by the second instruction. So, it did not read the operands until this writing was complete so in the next cycle if we go you see it will read the operand.

(Refer Slide Time: 50:33)



Scoreboard Example Cycle 9

Instruction status

Instruction	j	k		Issue	Read Op	EX compl	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Note: I3 and I4 read operands because F2 is now available. ADDD (I6) can't be issued because SUBD (I4) uses the adder

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

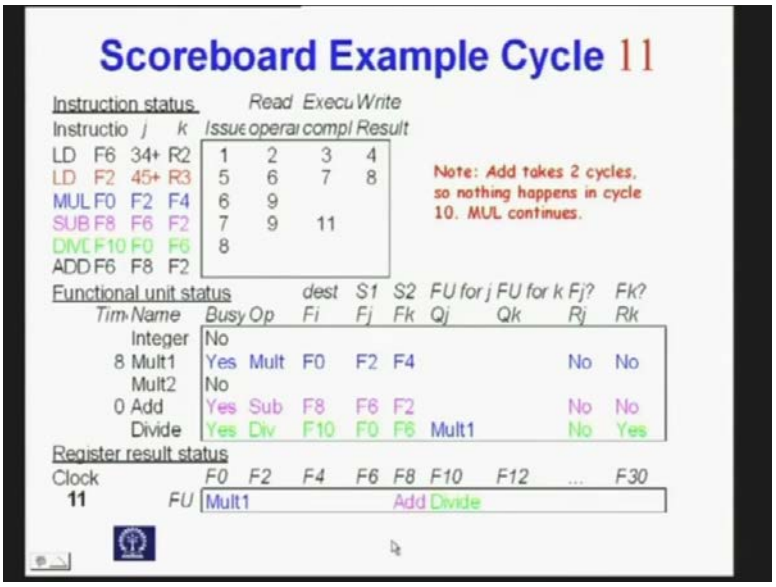
Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	FU	Mult1				Add	Divide			

So, it has read the operand because now I 3 and I 4 read the operands because the F 2 is now available. So, earlier F 2 was not available till the 8 cycle, but in the 8 cycle the writing the write back operation has been completed. And it is the operands are now available and you can see both these instructions where F 2 is the source operand or now reading their operands.

So, they will go to the second stage of the pipeline that is you are read operands. And this SUB D both of them will do that and here accordingly we can see the multiplication and division these are in progress. And ADD is not yet released that this particular so these 3 are in execution. Now and accordingly where the operands will be written are mentioned multiplier will write on into F 0 adder will write into F 8 and divide will write into F 10.

(Refer Slide Time: 51:46)



Scoreboard Example Cycle 11

Instruction status

Instructio	j	k	Issue	Read operai	Execu compl	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MUL F0	F2	F4	6	9		
SUB F8	F6	F2	7	9	11	
DIVD F10	F0	F6	8			
ADD F6	F8	F2				

Note: Add takes 2 cycles, so nothing happens in cycle 10. MUL continues.

Functional unit status

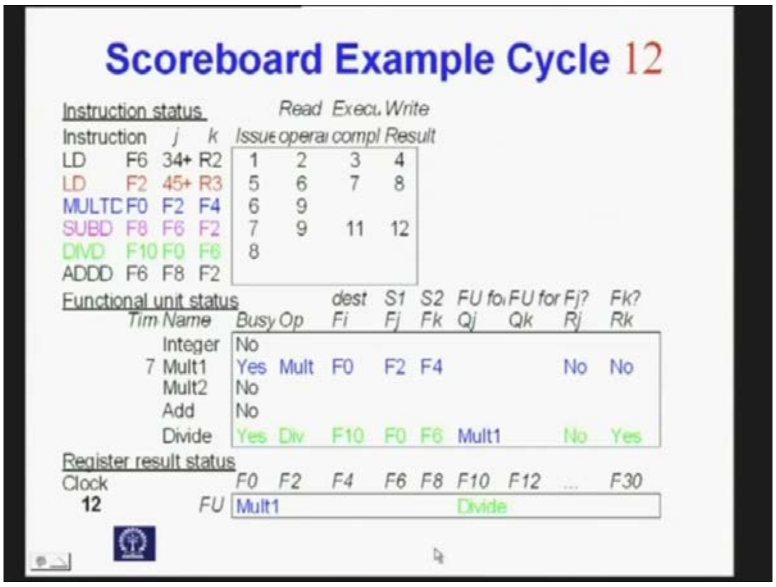
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	Mult1				Add	Divide			

So, we shall go to now cycle 11, in 11 it completed the operation and writing of the result takes place in F 8. And this other things because this one multiplier and divide will take more number of cycles so this multiplication will continue.

(Refer Slide Time: 52:15)



Scoreboard Example Cycle 12

Instruction status

Instruction	j	k	Issue	Read operai	Execu compl	Write Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULTD F0	F2	F4	6	9		
SUBD F8	F6	F2	7	9	11	12
DIVD F10	F0	F6	8			
ADDD F6	F8	F2				

Functional unit status

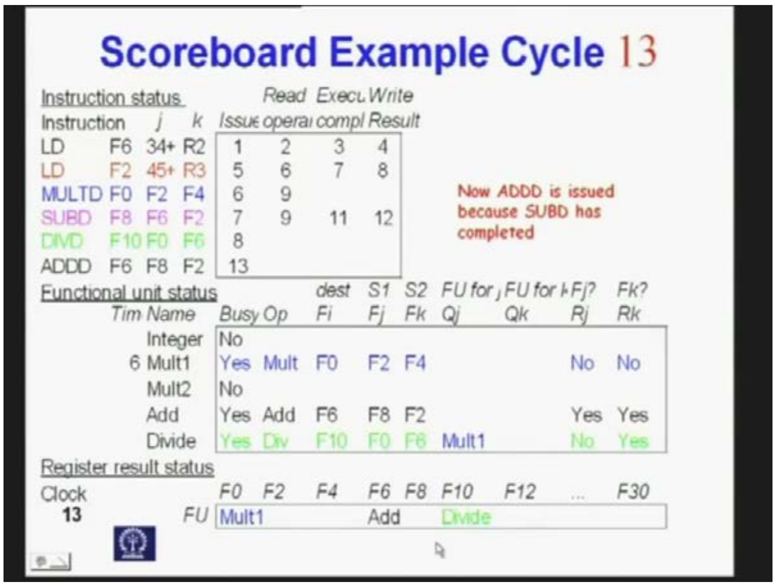
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
12	FU	Mult1					Divide			

So, we shall go to the 12 we have skipped few cycles and only in the twelfth cycle this particular instruction this particular instruction execution is complete. So, it has read its operand in ninth cycle execution it will take 2 cycles. So, in twelfth cycle the result is written into register F 8. And then you will see the can be issued that that this instruction can be issued in the next cycle.

(Refer Slide Time: 52:52)



Scoreboard Example Cycle 13

Instruction status

Instruction	j	k		Issue	Read operands	Execution complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Now ADDD is issued because SUBD has completed

Functional unit status

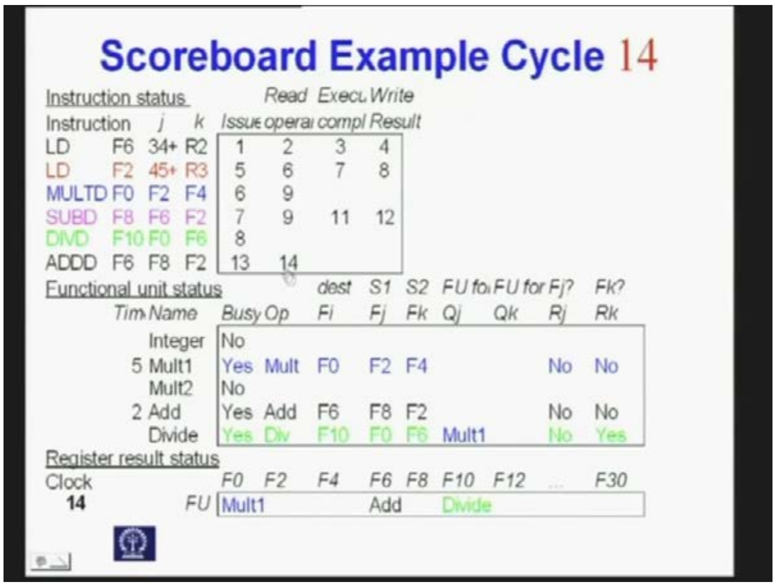
				dest	S1	S2	FU for j	FU for k	Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	Mult1			Add		Divide			

So, this instruction has been issued it the next cycle because operands are now available. So, all the instructions have been issued now and they will be in different stages of execution. The first instruction and this instruction execution have not been completed. So, you see that out of order execution has taken place out of order completion has also taken place, but writing of results has been done very I mean carefully such that hazards are overcome.

(Refer Slide Time: 53:27)



Scoreboard Example Cycle 14

Instruction status

				Read	Execu	Write
Instruction	j	k	Issue	operai	compl	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional unit status

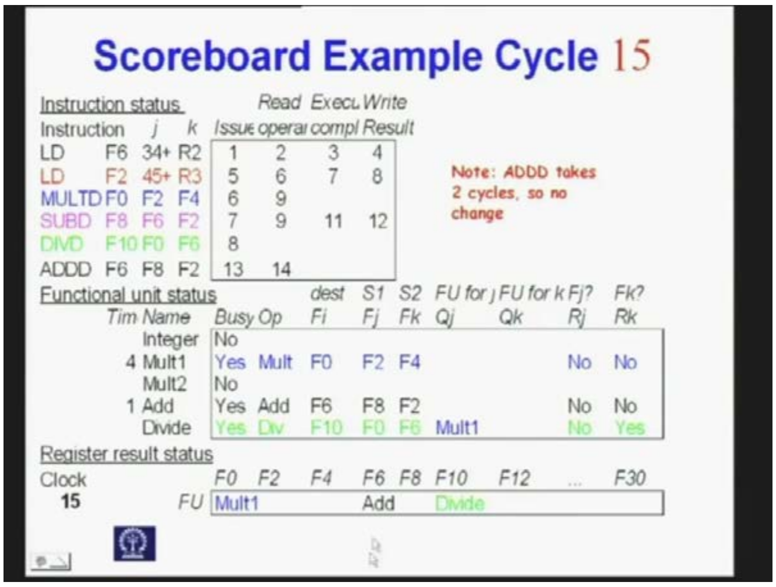
				dest	S1	S2	FU for j	FU for k	Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
5	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
2	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	FU	Mult1			Add		Divide			

So, this is the 14 cycle now the operands are available it will read the operands.

(Refer Slide Time: 53:33)



Scoreboard Example Cycle 15

Instruction status

				Read	Execu	Write
Instruction	j	k	Issue	operai	compl	Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Note: ADDD takes 2 cycles, so no change

Functional unit status

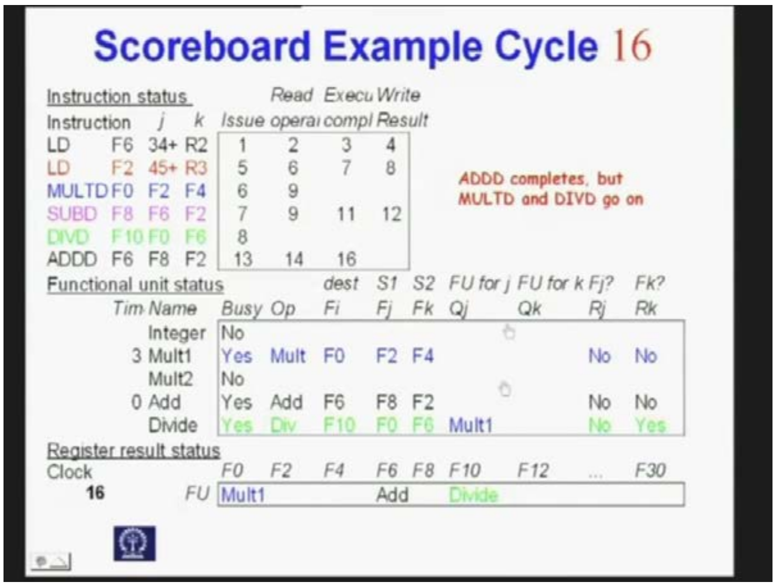
				dest	S1	S2	FU for j	FU for k	Fj?	Fk?
Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
4	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
1	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
15	FU	Mult1			Add		Divide			

And in the fifteenth cycle ADD D takes 2 cycles so no change.

(Refer Slide Time: 53:40)



Scoreboard Example Cycle 16

Instruction status	j	k	Issue	Read operai	Execu compl	Write Result
LD	F6	34+ R2	1	2	3	4
LD	F2	45+ R3	5	6	7	8
MULTD	F0	F2 F4	6	9		
SUBD	F8	F6 F2	7	9	11	12
DIVD	F10	F0 F6	8			
ADDD	F6	F8 F2	13	14	16	

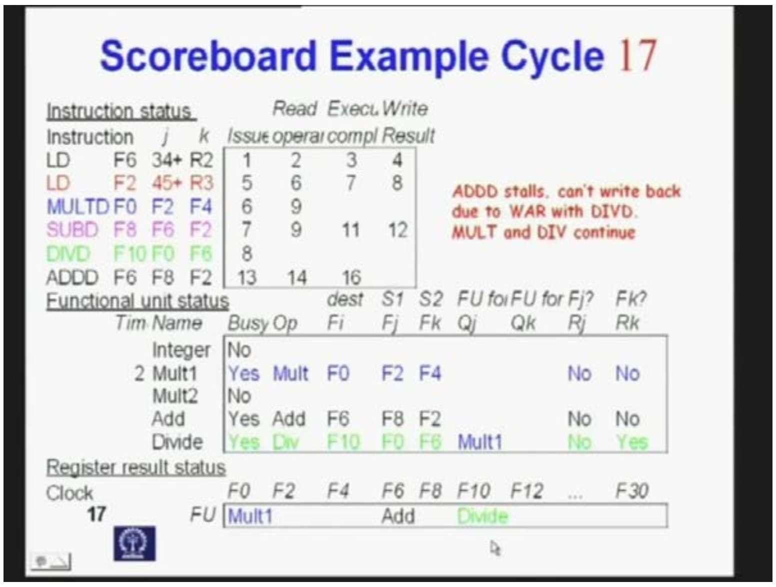
ADDD completes, but MULTD and DIVD go on

Functional unit status				dest	S1	S2	FU for j	FU for k	Fj?	Fk?
Tim	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
3	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
0	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock 16	FU	Mult1			Add		Divide			

And we shall be ADD D completes and but multiplication and divide will go on.

(Refer Slide Time: 53:47)



Scoreboard Example Cycle 17

Instruction status	j	k	Issue	Read operai	Execu compl	Write Result
LD	F6	34+ R2	1	2	3	4
LD	F2	45+ R3	5	6	7	8
MULTD	F0	F2 F4	6	9		
SUBD	F8	F6 F2	7	9	11	12
DIVD	F10	F0 F6	8			
ADDD	F6	F8 F2	13	14	16	

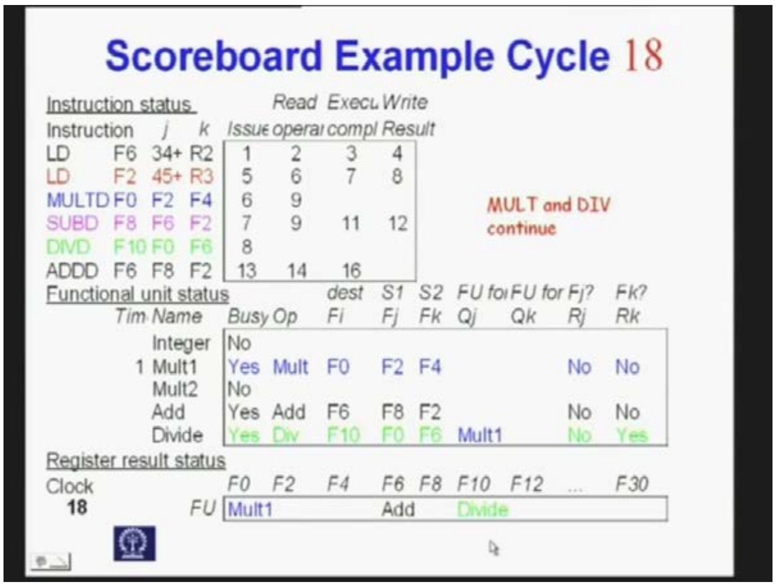
ADDD stalls, can't write back due to WAR with DIVD. MULT and DIV continue

Functional unit status				dest	S1	S2	FU fo	FU for	Fj?	Fk?
Tim	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status		F0	F2	F4	F6	F8	F10	F12	...	F30
Clock 17	FU	Mult1			Add		Divide			

It will take several cycles.

(Refer Slide Time: 53:49)



Scoreboard Example Cycle 18

Instruction status

Instruction	j	k	Read Issue	Execu operai	Write compl	Result
LD	F6	34+ R2	1	2	3	4
LD	F2	45+ R3	5	6	7	8
MULTD F0	F2	F4	6	9		
SUBD	F8 F6	F2	7	9	11	12
DIVD	F10 F0	F6	8			
ADDD	F6 F8	F2	13	14	16	

MULT and DIV continue

Functional unit status

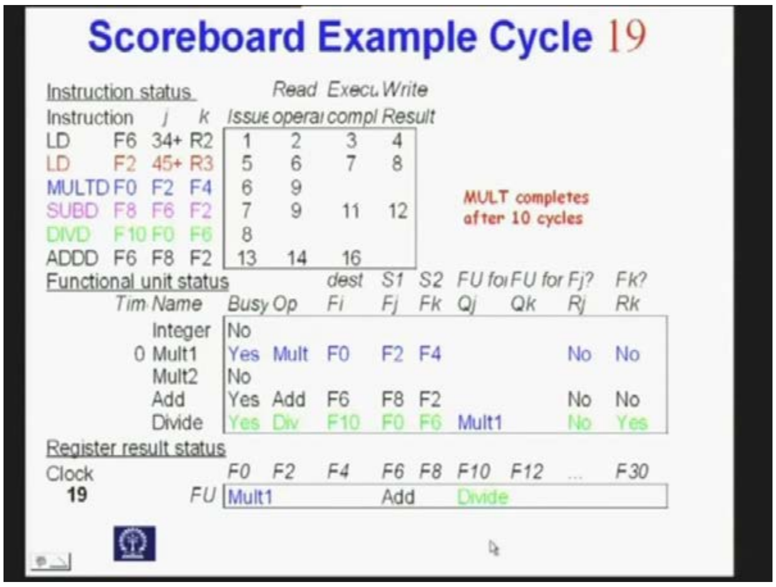
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU foi Qj	FU for Qk	Fj? Rj	Fk? Rk
	Integer	No								
1	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
18	FU	Mult1			Add		Divide			

It will continue.

(Refer Slide Time: 53:51)



Scoreboard Example Cycle 19

Instruction status

Instruction	j	k	Read Issue	Execu operai	Write compl	Result
LD	F6	34+ R2	1	2	3	4
LD	F2	45+ R3	5	6	7	8
MULTD F0	F2	F4	6	9		
SUBD	F8 F6	F2	7	9	11	12
DIVD	F10 F0	F6	8			
ADDD	F6 F8	F2	13	14	16	

MULT completes after 10 cycles

Functional unit status

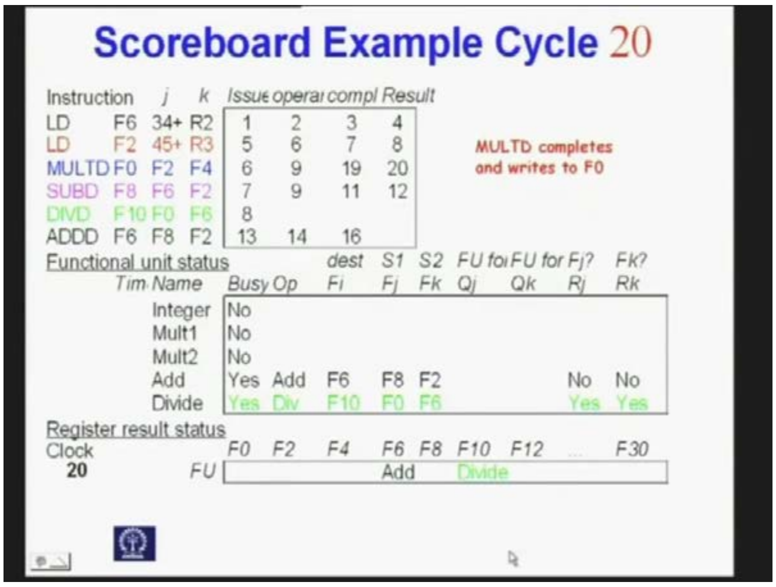
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU foi Qj	FU for Qk	Fj? Rj	Fk? Rk
	Integer	No								
0	Mult1	Yes	Mult	F0	F2	F4			No	No
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
19	FU	Mult1			Add		Divide			

Multiplication completes after 10 cycles.

(Refer Slide Time: 53:58)



Scoreboard Example Cycle 20

Instruction	j	k	Issue	operand	compl	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULTD F0	F2	F4	6	9	19	20
SUBD F8	F6	F2	7	9	11	12
DIVD F10	F0	F6	8			
ADDD F6	F8	F2	13	14	16	

MULTD completes and writes to F0

Functional unit status

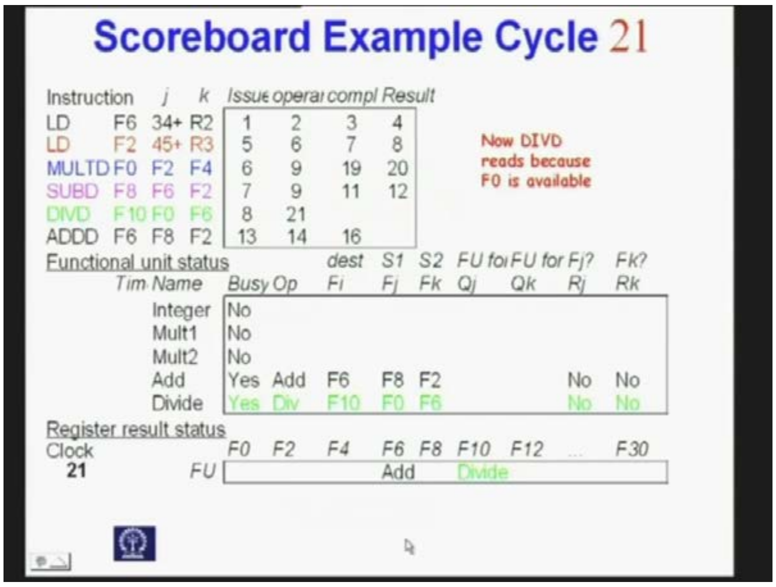
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
20	FU				Add		Divide			

So, in the 20th cycle multiplication is complete, but division will continue and accordingly these are corresponding databases are updated.

(Refer Slide Time: 54:10)



Scoreboard Example Cycle 21

Instruction	j	k	Issue	operand	compl	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULTD F0	F2	F4	6	9	19	20
SUBD F8	F6	F2	7	9	11	12
DIVD F10	F0	F6	8	21		
ADDD F6	F8	F2	13	14	16	

Now DIVD reads because F0 is available

Functional unit status

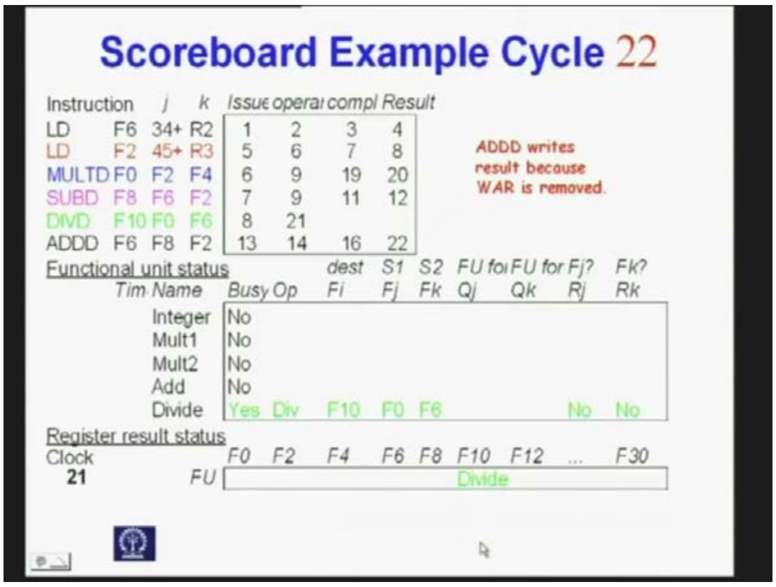
Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			No	No
	Divide	Yes	Div	F10	F0	F6			No	No

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
21	FU				Add		Divide			

So, scoreboard example after 21 only except divide and ADD D all execution are complete.

(Refer Slide Time: 54:20)



Scoreboard Example Cycle 22

Instruction	j	k	Issue	operands	compl	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULTD F0	F2	F4	6	9	19	20
SUBD F8	F6	F2	7	9	11	12
DIVD F10	F0	F6	8	21		
ADDD F6	F8	F2	13	14	16	22

ADDD writes result because WAR is removed.

Functional unit status

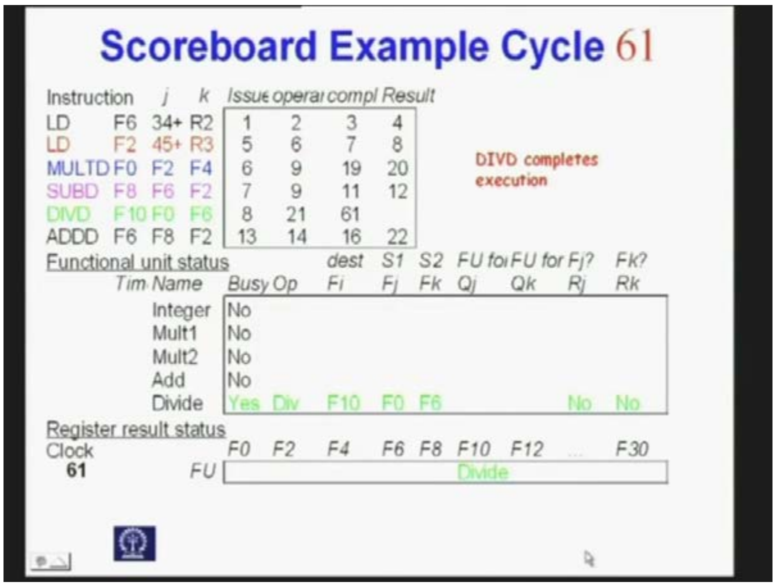
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for Qj	FU for Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6			No	No

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
21	FU						Divide			

And ADD D is also completing in cycle 22 and only divide is left out.

(Refer Slide Time: 54:27)



Scoreboard Example Cycle 61

Instruction	j	k	Issue	operands	compl	Result
LD F6	34+	R2	1	2	3	4
LD F2	45+	R3	5	6	7	8
MULTD F0	F2	F4	6	9	19	20
SUBD F8	F6	F2	7	9	11	12
DIVD F10	F0	F6	8	21	61	
ADDD F6	F8	F2	13	14	16	22

DIVD completes execution

Functional unit status

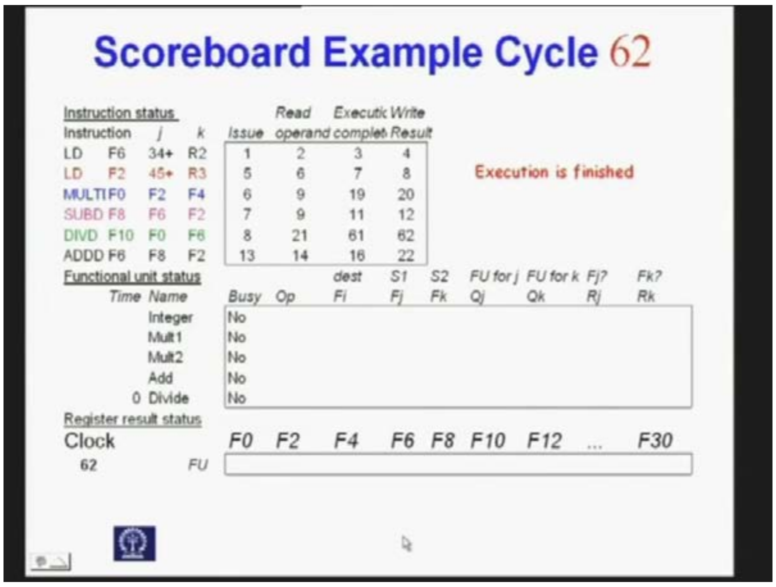
Tim	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for Qj	FU for Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6			No	No

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
61	FU						Divide			

So, we have skipped large number of cycles because 40 cycles are need by divide. So, accordingly we have skipped a number of cycles. Now, we shall go to the 60 first 61 cycles it is completing and now it will write the result.

(Refer Slide Time: 54:43)



Scoreboard Example Cycle 62

Instruction status

Instruction		j	k	Issue	Read operand	Executic complet	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTI	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

Execution is finished

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	No								

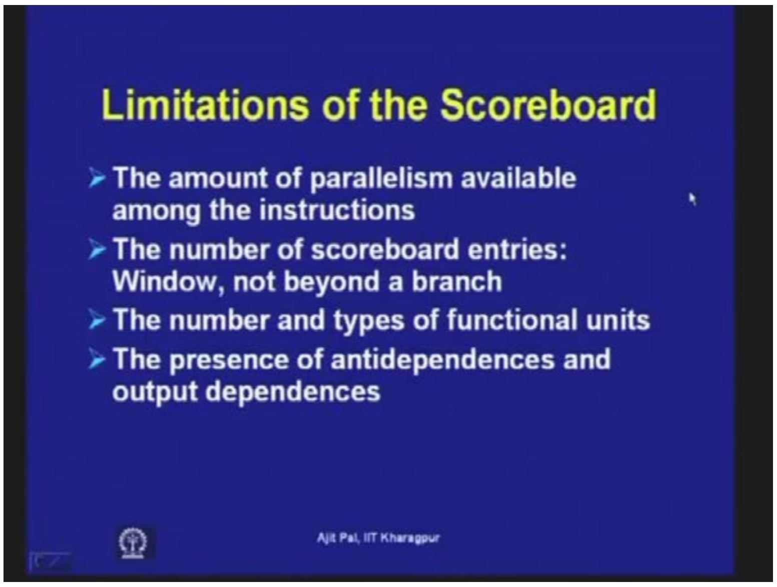
Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
62	FU									

Into the registers so all everything has been done execution is now finished. So, we have seen how the execution has been completed we have already discussed this. And we shall briefly mention about the limitation of scoreboard we have seen that amount of parallelism available among the instructions is very restricted for reason for that is it is restricting its window only to a basic block of a programme.

And we have seen that the within the basic block the instruction level parallelism is very much restricted. And as a consequence it cannot really give you very good result performance cannot be improved much. And second is the number of scoreboard entries uh that is window is not beyond branch. So, if the execution is completed beyond branch then that window of where the instructions which are considered by scoreboard is taken can be completed.

(Refer Slide Time: 55:55)



Limitations of the Scoreboard

- The amount of parallelism available among the instructions
- The number of scoreboard entries: Window, not beyond a branch
- The number and types of functional units
- The presence of antidependences and output dependences

Ajit Pal, IIT Kharagpur

But unfortunately the number of entries to the scoreboard is very limited because of the limited size of window the number of and types of functional units. So, the number and types of functional units has to be dependent 9 on the instructional level parallelism available. And the window size there is no fun in having a very large number of functional units, because that may overcome the structural hazard. But because of the other type of other 3 types of hazards that may be stalls and performance cannot be much.

So, the number and types of functional units are to be carefully chosen and the presence of anti-dependences. And output dependences we have already see the anti-dependences and output dependences are arising, Because of out of order execution and they are tackled by scoreboard with the help of by stalling the stalling the by introducing stalling cycles. And that is how the scoreboard is performing so in the next class or may be subsequently we shall discuss about that another dynamic instruction scheduling approach. That is Tomasulo is approach which was developed Doe IBM 360 and that is more sophisticated. And we shall see how it overcomes some of the limitations of scoreboard.

Thank you.