

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

Lecture - 14
Dynamic Instruction Scheduling (Contd.)

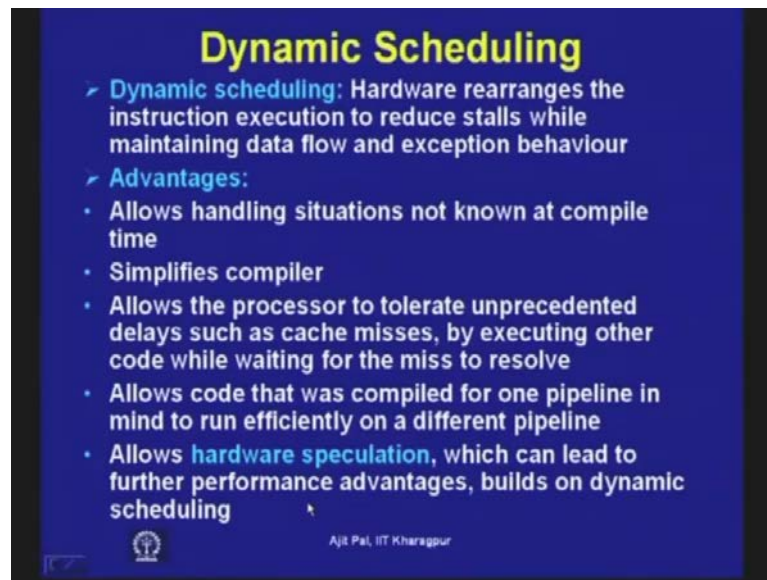
Hello and welcome to the today's lecture. We shall continue our discussion on dynamic instruction scheduling. In the last lecture, we have discussed about one technique known as scoreboard.

(Refer Slide Time: 01:15)



And today we shall discuss about Tamsulo's algorithm which is an advanced version, and enhance version of dynamic instruction scheduling based on score boarding approach.

(Refer Slide Time: 01:29)



Dynamic Scheduling

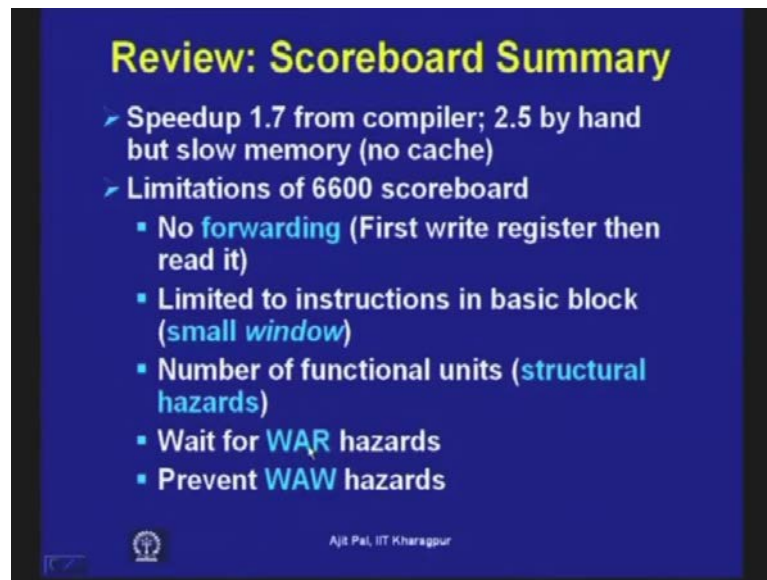
- **Dynamic scheduling:** Hardware rearranges the instruction execution to reduce stalls while maintaining data flow and exception behaviour
- **Advantages:**
 - Allows handling situations not known at compile time
 - Simplifies compiler
 - Allows the processor to tolerate unprecedented delays such as cache misses, by executing other code while waiting for the miss to resolve
 - Allows code that was compiled for one pipeline in mind to run efficiently on a different pipeline
 - Allows **hardware speculation**, which can lead to further performance advantages, builds on dynamic scheduling

Ajit Pal, IIT Kharagpur

As I have mentioned dynamic scheduling means the hardware rearranges instruction execution to reduce stalls while maintaining data flow and exception behavior. So, instead of compiler it is done by hardware so that was that is mention earlier and it is got a number of advantages it allows handling situations not known at compile time. And obviously its simplifies the compiler it is not necessary to have a optimizing compiler for a particular higher processor. And it allows the processor it tolerant unprecedented delays such as cache misses by executing other code while waiting for the miss to resolve. That means certain situation will happen at run time which cannot be predicted at compile time. And when those things happened at run time the hardware will take care of it and that is it.

That is one very important advantage as I have mentioned earlier and it allows code that was compiled for 1 pipeline in mind to run efficiently on different pipeline. So, it is not explicitly pipeline dependent, a particular code which has being compiled for a particular pipeline will run in another pipeline whenever we use this hardware based instruction scheduling instead of compiler based instruction scheduling. And also later on we shall discuss on advance technique like hardware speculation which can lead to further performance advantage advantages. And particularly it will increase the instruction level parallelism which is essential for superscalar and other multiple issue processors. And obviously this is based on dynamic instruction scheduling.

(Refer Slide Time: 03:36)



Review: Scoreboard Summary

- Speedup 1.7 from compiler; 2.5 by hand but slow memory (no cache)
- Limitations of 6600 scoreboard
 - No forwarding (First write register then read it)
 - Limited to instructions in basic block (small window)
 - Number of functional units (structural hazards)
 - Wait for WAR hazards
 - Prevent WAW hazards

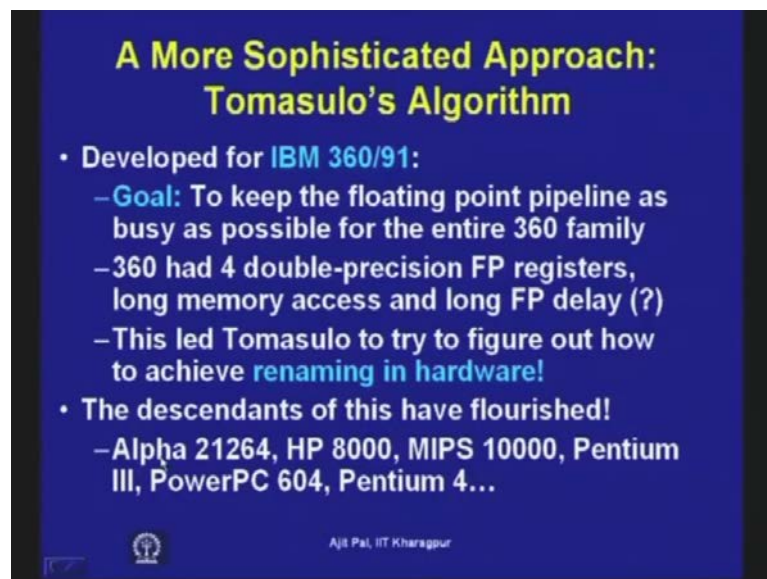
Ajit Pai, IIT Kharagpur

Now, let me very quickly review about this scoreboard approach which we have discussed in last lecture. We have seen that it gives you speedup of 1.7 for compiler generated code and 2.5 speedup by hand. But slow memory so by hand whenever you do the coding then you get better speedup. However at that time there was no cache memory so there was cache slow memory was used. And it has got a number of limitations, number first limitation that arise is that it does there is no forwarding. As I mention in case of score boarding data is taken from actual register. So, until the write back take place in the actual register you cannot read the data. So, forwarding scheme which enhances the performance by reading data from pipeline register that is not allows in soccer boarding so that say very strong limitation.

Second is limited to instruction in basic block, as you have already seen the basic block size is very limited may be 6 8 10 that is the number of instruction that you get in the basic block. That means without doing lupa rolling without doing other techniques to improve instruction level parallelism, the parallelism available within the basic block is very restricted and as a consequence the, you do not get much benefit. And then number of functional units is also restricted and it decides the structural hazards. That means the, whenever you have got some even you have got instruction level parallelism; you cannot really, proceed because of structural hazard that is restricted by the number of functional units.

And in case of write after read hazards, we have seen since it is based on reading from the register. It leads to stall that means it depends the execution of that instruction has to be has to wait till writing technique take place severely. Similarly, it prevents write after write hazard, but again it does it by not executing instruction which are in the later in the later part of the program code that means it follows the program order. So, by doing that it prevents WAW hazards although issue should takes place, but the write back that means the completion is avoided completion is prevented.

(Refer Slide Time: 06:38)



**A More Sophisticated Approach:
Tomasulo's Algorithm**

- Developed for **IBM 360/91**:
 - **Goal**: To keep the floating point pipeline as busy as possible for the entire 360 family
 - 360 had 4 double-precision FP registers, long memory access and long FP delay (?)
 - This led Tomasulo to try to figure out how to achieve **renaming in hardware**!
- The descendants of this have flourished!
 - Alpha 21264, HP 8000, MIPS 10000, Pentium III, PowerPC 604, Pentium 4...

Ajit Pal, IIT Kharagpur

So, these are the limitations of scoreboard approach and some of them have been overcome by using the Tomasulo's approach; so it is most sophisticated approach and it was developed for IBM 360 by 91. The basic goal was to keep the floating point pipeline as busy as possible for the entire 360 family. So, 360 family means there were many variations a series of process computers were developed. So, it was meant for the entire series not that at this the enhance versioned. The more advance version for which you develop good compiler and the the lower versions you leave adjective instead of that the score boarding this particular approach will allow simple compiler. And the scoreboard will be used for the entire series of 360 family that was the basic goal particular for keeping the floating point pipeline in mind.

And particular 360 has got very restricted number of registers. And we have seen score boarding and also the compiler based instruction scheduling requires large number of

registers. But unfortunately 360 has 4 double processing floating registers and long memory access, because in those and long floating point delay. The reason for that was in those days cache memory was not invented. So, in 360 there was no cache memory so reading data from memory used to take long time that is number 1, number 2 is floating point operations and other multiplication another operations used to take longer time.

So, this led to Tomasulo to try a try to figure out how to achieve renaming in hardware. So, this another innovative technique that has been used here it, you know we have already mention about the use of renaming which can help you to remove anti dependences and output dependences. But whenever you do the renaming; obviously, will require large number of registers. But we should do it with the help of software or with the help of compiler. But in a very innovative way it has been done by hardware rename renaming by hardware with the using in Tomasulo's approach we shall see how it has been done.

And because of these advantages you know some version of this Tomasulo's approach were used in more recent processors. So, although that was develop for 360 by 91, but subsequent processors more recent processors like take alpha 21264 HP 8000 MIPS run 1000 Pentium 3 power p c 604 Pentium 4 and so on. In all the modern processors Tomasulo's approach has been used in some way out. The other may not be the in the exactly the way it was develop for data I that 360 by 91. But some modifications modify version of it has been used I all the modern processors.

(Refer Slide Time: 10:11)

A Simple Example

➤ Consider the following example:

DIV.D	F0,F2,F4
ADD.D	F6,F0,F8
S.D	F6,0(R1)
SUB.D	F8,F10,F14
MUL.D	F6,F10, F8

- If SUB.D is scheduled before ADD.D, it leads to **WAR** hazard
- Similarly, if MUL.D is scheduled before ADD.D, it leads to **WAW** hazard
- **Register renaming** can be used to overcome this problem

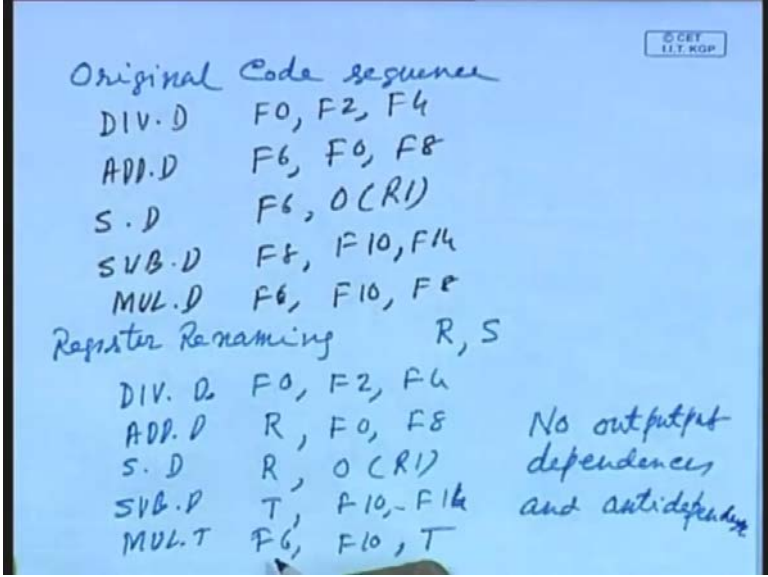
Ajit Pal, IIT Kharagpur

Let me give you a simple example before I proceed to discuss Tomasulo's approach. This is a simple 5 line code sequence; here you can see it has got a large number of dependences. For example, this it has got 3 3 2 dependences for example, divide D and ADD D has 2 dependences, because the, and which can read after write hazard. So, the operand that has been generated by the first instruction is used by the second instruction. Similarly, you have got another output dependences for example, between your divide D and SUB D here also there is that means these two has got output dependences. So, you are writing into the register here also is writing into the register.

So, you here you got output dependences then here you have got anti dependences you can see SUB D and multi multi D. So, here you have got output dependences so we find that and here is data dependences 2 data dependences also SUB D and MUL D. You are writing in a faith and then that is being used in a subsequences instruction. So, you find that apart from in several 3 data dependences that is present in this program code. It has got other dependences like output dependences and name dependences. Now, the and if you want to avoid the hazard say you may want to scheduled the SUB D instruction before ADD D, I mean by doing that. But it cannot be done, because it leads to WAW hazard. And similarly, multi multiplication double is scheduled before it ADD D it will lead to WAW hazard write after write hazard.

So, because of this instruction scheduling cannot be done, but register renaming can be used to overcome this problem. That means this w a are and WAW hazards can be overcome by using register renaming. Let me illustrate this with the help of this simple example here you have got.

(Refer Slide Time: 12:45)

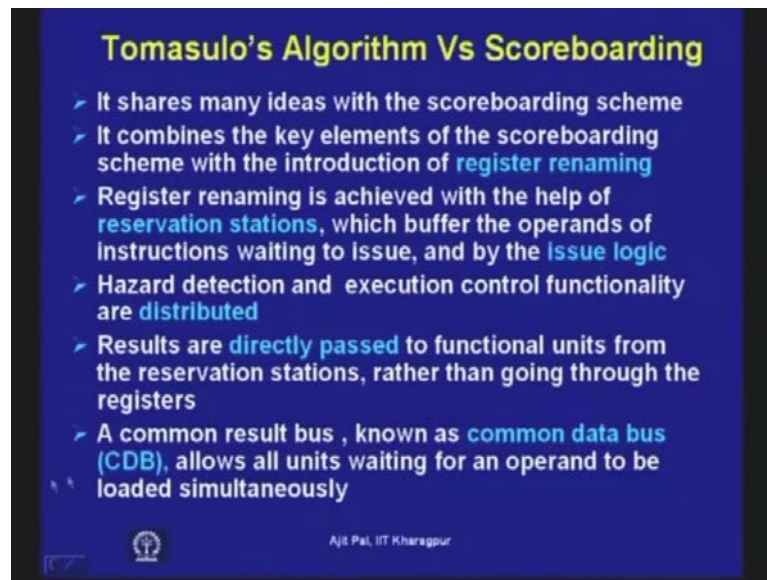


Original Code sequence		
DIV.D	F0, F2, F4	
ADD.D	F6, F0, F8	
S.D	F6, 0(R1)	
SUB.D	F8, F10, F14	
MUL.D	F6, F10, F8	
Register Renaming R, S		
DIV.D	F0, F2, F4	
ADD.D	R, F0, F8	No output dependencies and anti-dependencies
S.D	R, 0(R1)	
SUB.D	T, F10, F14	
MUL.D	F6, F10, T	

Divide D F 0 F 2 F 4 then you have got ADD D F 6 F 0 F 8 stored that double F 6 0 R 1 then SUB D F 8 F 10 F 14 and you have got mult multiplication double F 6 F 10 F 8. Now, let us do this is the original code sequence. Now, let us do register renaming let us assume we have got 2 temporary register R and S, R and S we have got 2 temporary register which we shall use in register renaming.

So, what we can do? we can do divide D F 0 F 2 F 4 then here ADD D here instead of writing in F 6 we write in say a temporary register R F 0 and F 8. And the next instruction S D also use R output R 1. And then what we do then SUB D here we used T F 10 F 16 F 14. And then MUL D here you have got a F 6 F 10 and F 8 has been replaced by T so here it will be T. So, here we find that there is no output dependencies and anti dependencies. So, you ofcourse, you have got 2 dependences 2 data dependences which will be present there. And so renaming register renaming can be used to overcome these the WAW and WAR are type hazard.

(Refer Slide Time: 15:42)



Now, this we shall do if we do it by software then you will require large number of registers how it is done by using Tomasulo's approach by hardware we shall see later. And let me compare Tomasulo's algorithm verses score boarding it shares many ideas with the score boarding scheme. It combines the key elements of the score boarding scheme with the introduction of register renaming.

So, this is the additional new features; that is the register renaming by hardware and register renaming is achieved with the help of some hardware known as reservation stations. So, in the reservation stations the operands are buffered. So, instead of writing into the register or any temporary register we used reservation stations where the operand values are buffered. And then the from the reservation stations it can be provided to the execution unit and so which buffered the operands of instructions waiting to issue and by the issue logic.

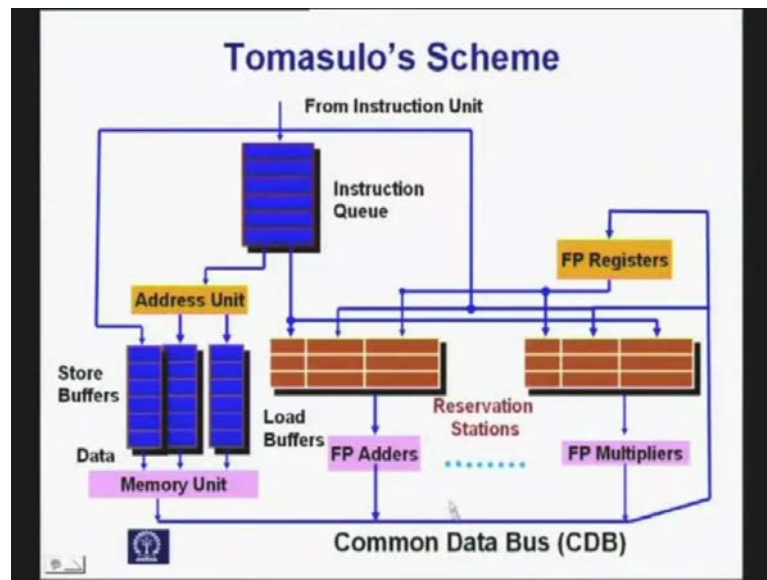
So, reservation unit stations are used corresponding to each functional unit and where the the operand values can be buffered without the need for additional registers in the processor and hardware detection and execution control functionality are distributed. We have seen in Tomasulo's approach there was a scoreboard. Scoreboard is used to overcome the to detect if there any if there is any hazard that means if there is any structural hazard then the instruction is not issued. And if there is any data hazard then it is not issued unless until the operands are available.

Here also here it is done by in distributed manner instead of using a single scoreboard it uses several reservation stations. And these reservation stations does the hazard detection and execution control functionality functions in a distributed manner. And results are directly passed to functional units from the reservation stations rather than going through the registers. We have seen in score boarding the results were first written into the registers. And the execution units were reading them from the registers. But here what is being done the reservation stations are holding the values and which are directly passed to the execution units so rather than going through the registers.

So, these are the main differences and a another very interesting thing is it uses it uses a bus a common bus, a common result bus known as common data bus CDB, allows all units waiting for an operand to be loaded simultaneously so what can happen. A particular functional unit may produce a result and which can be used by several other functional units. So, that means the result produced by 1 instruction may be used by several other instructions.

So, in this particular case there is a common data bus on which a particular execution unit will produced the result then since this result is available on a common data bus. So, all the execution units which are waiting for the operands will get them simultaneously and there execution can be started. So, this another difference between Tomasulo's algorithm and score boarding. So, we find that these are the key differences between score boarding and Tomasulo's algorithm.

(Refer Slide Time: 19:59)



And here is the structure of the hardware that you require to implement the Tomasulo's approach. And you can see here the instructions are fetched by instruction fetch unit and they are stored in a instruction queue. So, instructions are fetched and the instruction unit will put them in this instruction queue. And from the instruction queue the instructions can be issued to multiple functional units. For example, here you have got memory unit here, you have got several floating point adders. And you have got several floating point multipliers and these functional units floating point adders and floating point multipliers are getting their inputs not from the registers but from the reservation stations.

So, reservation stations are holding not only info I shall tell about the data's structure that will be used by reservation stations. And you will see the values are stored in this reservation stations. And they can be directly provided to the functional units whenever both the operands are available and whenever the operands are available the execution is started.

Similarly of course, the load and store is done it perform in a little different way. The load operation can be perform as soon as the address is available. So, address unit will initially stores the value that is up taken from the instructions. And then it computes the actual address and that address is stored that in the address unit. And this address the in

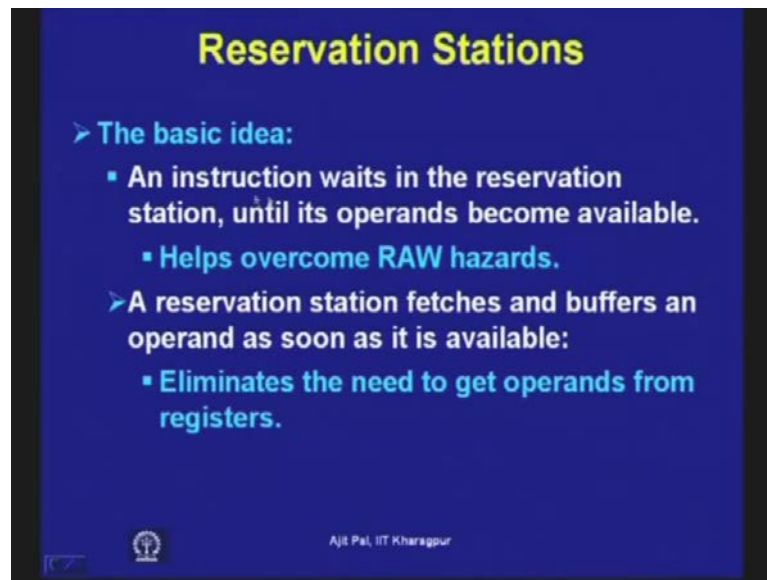
the store buffers the addresses are available and as soon as the addresses are available the memory unit is I mean they sense the address to memory unit.

And it reads the data from the memory and that data is available on this common data bus as you can see as the data is available from the common data bus. It goes to not only to the registers where the data has to be written, but it also goes to all the reservation stations. So, it goes to all the reservation stations which are waiting for the data. And they will write it into the registers. Similarly, whenever any operation is performed by adder or a multiplier those outputs are provided on the common data bus. And from the common data bus it is written into not only written into registers as well as it goes to the reservation stations where they are buffered if you subsequent structure needs them.

So, this is how it goes on and you can see the store operation is concerned the you will require not only the address. So, you will require not only the address but you will require data. So, you can see that it will store that address as well as the data that will be available as you can see through this common data path. And as soon as the address and data are available it is put into this buffer. And when both address and data are available it can be stored in the appropriate memory with the help of this memory unit that is the store and load differs. But so far as the execution is concerned you see in the load. And store also is considered just like any other instruction. I mean ALU based instructions performing addition multiplication and other things.

So, it does not have to treat load and store differently, but you have the data is coming not from reservation stations. But the address and data are coming from separate buffers, you have got buffer for address and also buffer for data from where the memory unit will get information about address as well as address and data in case of store. And accordingly writing will take place as soon as both the values are available. So, this is the structure that you required for implementing Tomasulo's scheme.

(Refer Slide Time: 24:43)



Reservation Stations

- The basic idea:
 - An instruction waits in the reservation station, until its operands become available.
 - Helps overcome RAW hazards.
- A reservation station fetches and buffers an operand as soon as it is available:
 - Eliminates the need to get operands from registers.

Ajit Pal, IIT Kharagpur

Now, let me highlight some of the important features of different functional units. In reservation stations, the single entry buffer at the head of each functional unit has been replaced by a multiple entry buffer. We have seen in the case of scoreboarding that there were multiple entry buffers, and here you have a simple single entry buffer as you can see, which are stored in the reservation stations.


A common data bus connects the output of the functional units to the reservation stations as well as registers. That means parallel writing will take place in the register as well as in the buffer of a reservation station from the common data bus. And wherever it is required, it will be channelized to the appropriate reservation stations where it is waiting for the data. Obviously, to do that, it has to maintain a suitable data structure, which I shall discuss a little later.

Then it has got register tags. A tag corresponds to the reservation station entry number for the instruction producing the results, that means which particular reservation station will produce a result. And that will be used by subsequent instructions, and that it is maintained with the help of register tags. So, the basic idea is an instruction waits in the reservation station until its operands become available. So, its basic philosophy is the same as that of a scoreboarding-like data flow machine. So, as soon as data is available, execute it so it will definitely lead to out of order execution and obviously out of order completion. And

whenever since the operands it waits for the operands to become available it helps to overcome read after write hazard.

So, read after write hazard is overcome by following this approach because only when operands are available then execution is started. So, read after write hazard is overcome by this and a reservation station fetches and buffers an operand. As soon as it is available we have seen that the output from different functional units are directly going to the reservation stations. And it eliminates the need to get operands from registers so which was done in case of scoreboard.

(Refer Slide Time: 27:40)



Tomasulo's Algorithm

- Control & buffers distributed with Function Units (FU)
 - In the form of "reservation stations" associated with every function unit.
 - Store operands for issued but pending instructions.
- Registers in instructions replaced by values and others with pointers to reservation stations (RS):
 - Achieves register renaming.
 - Avoids WAR, WAW hazards without stalling.
 - Many more reservation stations than registers (why?), so can do optimizations that compilers can't.

Ajit Pal, IIT Kharagpur

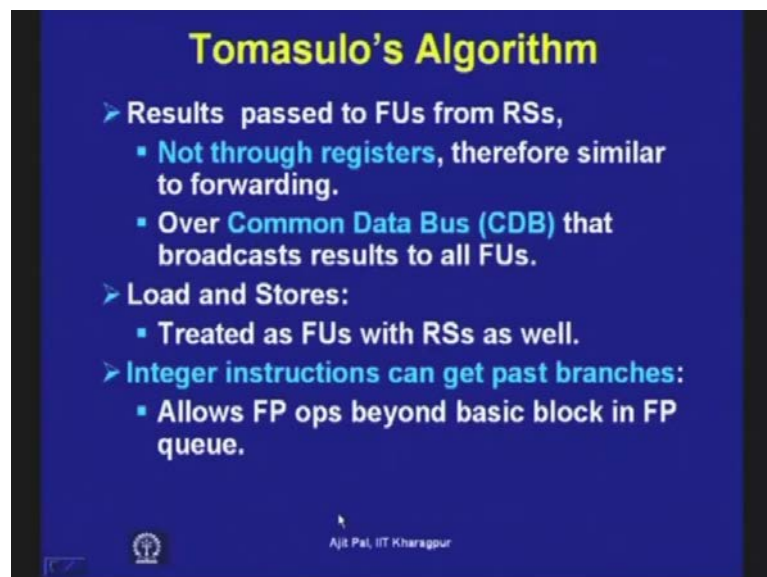
So, instead of having a centralized control and buffer, the control and buffer distributed with functional units in the form of reservation stations where reservation stations can be consider a kind of which are part of the functional units. That means a function each functional unit is associated with the reservation station and where the operand value can be stored. As soon as operands are available they are transferred to the reservation stations and store operands for issued, but pending instruction and store operands for issued. But pending instruction that means as soon as an instruction issue take place automatically a functional unit.

And reservation stations gets allocated and they keep track of it. And since a reservation station along with functional unit is allocated as an instruction issued they will automatically keep track of when the outputs are generated by are by instructions which

will produce the operands needed by those instructions. So, register in instructions in instruction replaced by values and others will with pointers to reservation stations. So, as I have already mentioned we are not using ah registers, but the values actually obtained by doing a computation are stored in the reservation stations and obviously you have to keep pointers. So, that the values can be appropriately used and this is achieved this achieves register renaming as I have told and it avoids WAR and w a type of hazards without stalling.

So, it does not allowed to be I mean stall for the WAR and WAW type of hazards So, it does register renaming without it need of registers that is a key advantage. However it will require may more reservation stations than registers, because each functional units is associated with a register reservation station. And each reservation station must have the necessary registers to store the operands. So, can do optimizations that compilers cannot that means the compile what the compiler cannot do, can be done by the hardware with the help of these reservation stations. Because it can at run time it obtains the, that values which are stored in the reservation stations and which cannot be predicted or obtain at compile time.

(Refer Slide Time: 30:39)



Tomasulo's Algorithm

- Results passed to FUs from RSs,
 - Not through registers, therefore similar to forwarding.
 - Over Common Data Bus (CDB) that broadcasts results to all FUs.
- Load and Stores:
 - Treated as FUs with RSs as well.
- Integer instructions can get past branches:
 - Allows FP ops beyond basic block in FP queue.

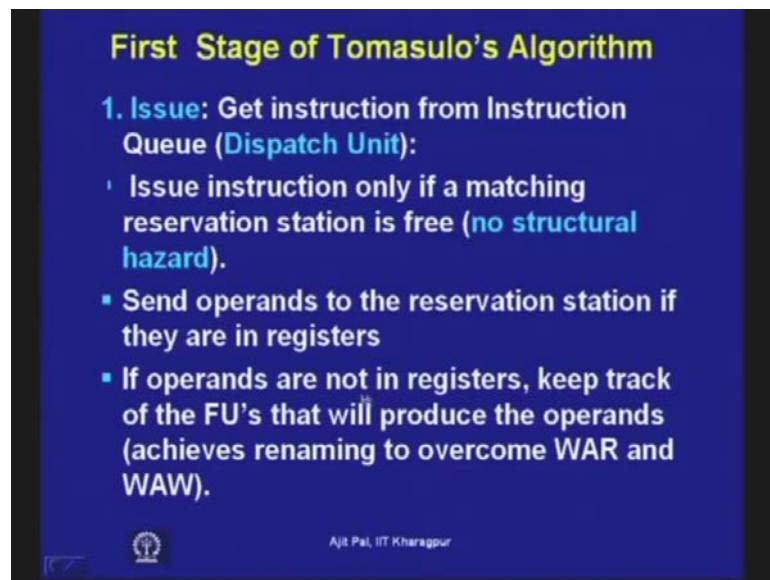
Ajit Pal, IIT Kharagpur

So, results passed to functional units from reservation stations, this is another thing I have already repeated it not through the registers they are similar to forwarding. So, without doing forwarding hardware forwarding is also achieved, because you know

before the values are written into the registers these are available in the form of reservation stations as soon as they are generated by functional units. And as a consequence it is serving purpose of forwarding without the forwarding hardware. So, over common data bus that broadcasts results to all functional units. So, as I have already seen the structure of the hardware where the through the common data bus. The results are broadcasted and which goes to all the registers which need them.

And as I have already mentioned load and stores are treated as functional units with reservation stations as well I have explained that. And then integer instructions can get past branches this one I shall discuss in the subsequent lecture you we have been seen in case of score boarding the it is restricted that instruction scheduling is restricted to simple block I mean it cannot go beyond the branches. But with the help of this Tomasulo's algorithm we shall see it can go beyond basic block. And as a consequence that means the basic block size that means the number of instructions that can be that is available in instruction window for I mean carries larger than score boarding.

(Refer Slide Time: 32:36)



First Stage of Tomasulo's Algorithm

- 1. Issue:** Get instruction from Instruction Queue (**Dispatch Unit**):
 - Issue instruction only if a matching reservation station is free (**no structural hazard**).
 - Send operands to the reservation station if they are in registers
 - If operands are not in registers, keep track of the FU's that will produce the operands (achieves renaming to overcome WAR and WAW).

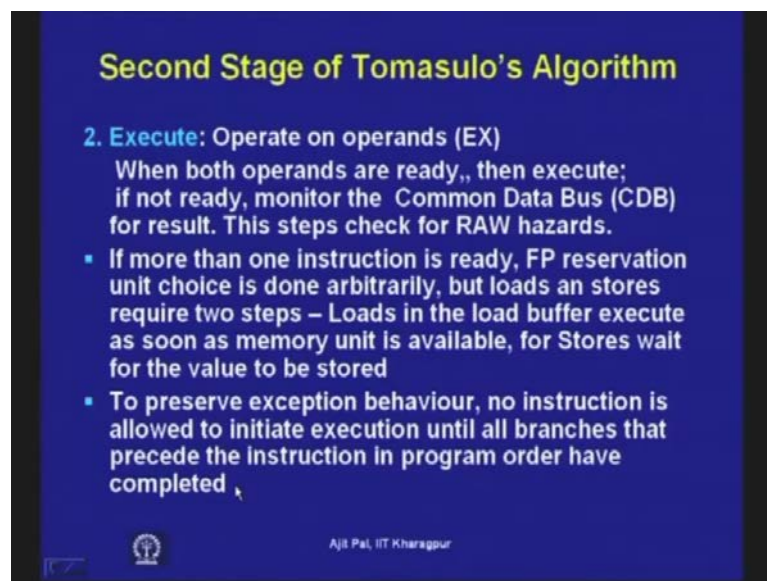
Ajit Pal, IIT Kharagpur

So, let us see the different stages of Tomasulo's algorithm it has got three stages. First stage is known as issue stage so it gets instruction from instruction queue I have already seen. So, it is serving the function of dispatch unit and it issue instruction only if matching reservation station is free. So, here the way the structural hazard is overcome

this that means it must have the reservation stations available for the corresponding function. So, you have got multiple functional units But you know functional units may be busy and as a consequence whenever a particular instruction is a reject then it may not be issued because there will be structural hazard. So, instruction issue is only if matching reservation station is available and along with the functional units.

So, second operand to the reservation station if they are in registers send operands to the reservation stations if they are in the registers. So, if they are in registers they are send to the operands a reservation stations they can be But most of this situation most of the cases they take it directly from the output of the functional unit. So, if operands are not registers keep track of the functional units that will produce the operands as you have seen and this achieves renaming to overcome WAR and WAW type of hazards.

(Refer Slide Time: 34:15)



Second Stage of Tomasulo's Algorithm

2. Execute: Operate on operands (EX)
When both operands are ready, then execute;
if not ready, monitor the Common Data Bus (CDB) for result. This step checks for RAW hazards.

- If more than one instruction is ready, FP reservation unit choice is done arbitrarily, but loads and stores require two steps – Loads in the load buffer execute as soon as memory unit is available, for Stores wait for the value to be stored
- To preserve exception behaviour, no instruction is allowed to initiate execution until all branches that precede the instruction in program order have completed

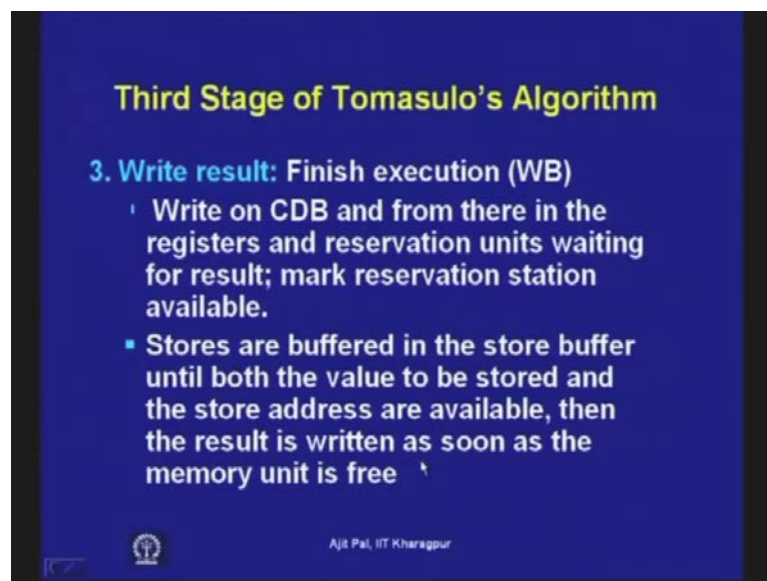
Ajit Pal, IIT Kharagpur

And the second stage is the execution stage which operate on the operands and when both operands are ready then execute if not ready monitor the common data bus for result. So, as soon as the result operands are available at the output of the common data bus and it will do the reading. And then execution can be started so this step checks for read after write type of hazards. So, if more than one instructions is ready that means what can happen several instructions more than one instructions may get ready that means may have operands available. And they can they are in dependent they can be

issued simultaneously in such a case what can be done in that in such a case instruction issue is done arbitrarily.

So, floating a FP reservation unit choice is done arbitrarily, but loads and loads and stores are 2 steps as I have already seen discuss loads in the load buffer execute as soon as memory unit is available for stores it has to wait for value as well as for address. So, to preserve exception behavior no instruction is allowed to initiate execution until all branches that precede the instruction in program order have completed. So, exception behavior has to be also maintain by this approach so that means what is being done the instruction is allowed to precede. Precede means it is allowed to execute out of order, but it is not allowed to write into the registers until you know that all branches that precede the instruction in the program order have been completed. That means it maintains the program order particular whenever there are branches in between that is the reason why it can go beyond the branches.

(Refer Slide Time: 36:24)



And the third stage of the Tomasulo's algorithm is write result so finish execution so it is write back. So, write back on CDB and from there in the registers and reservation units waiting for result. So, mark reservation stations available that means here we have seen whenever writing take place. Writing is done by floating the data on the common data bus. And accordingly it goes to appropriate register as well as to the reservation stations who are waiting for the data stores are buffered in the store buffer until both the value to

be stored. And the store address are available then the result is written as soon as memory unit is free. So, in the this write back stage also the that store operation is performed, because it trick were 2 steps because address is required and value to be stored is also required.

(Refer Slide Time: 37:32)

Three Parts of the Tomasulo's Algorithm

1. **Instruction status:** which of 3 steps the instruction is in
2. **Reservation stations:** Each reservation station has six fields
Op: Operation to perform on source operands S1 and S2
Qj, Qk: Reservation stations producing source operands
Vj, Vk: The value of the source operands
Busy: Indicates that this reservation station and its accompanying functional unit are occupied
3. **Register status:** The register file and store buffer each have a field, **Qi**: If the value is blank, no currently active instruction is computing a result destined for this register
The load and store buffers each has a field, **A**, which holds the result of the effective address once the first step of execution has been completed

Ajit Pai, IIT Kharagpur

Now, there are 3 parts number 1 is instruction status that means instruction can be one of the 3 stages issue execute write back. So, that that is maintain that is instruction status is maintain and which of the 3 steps the instruction is in that is being maintained by suitable data structure. Then reservation stations each reservation station has 6 fields and these are the 6 fields. Number 1 is operation to perform on source operand S 1 and S 2 then Q j and Q k reservation stations producing the source operand, you can see the a reservation station keeping track which other reservation stations are producing result producing operands. And as a consequence as soon as they are available they can be as transferred to the proper registers available in the reservation station so Q j and Q k field keeps track of those reservation stations.

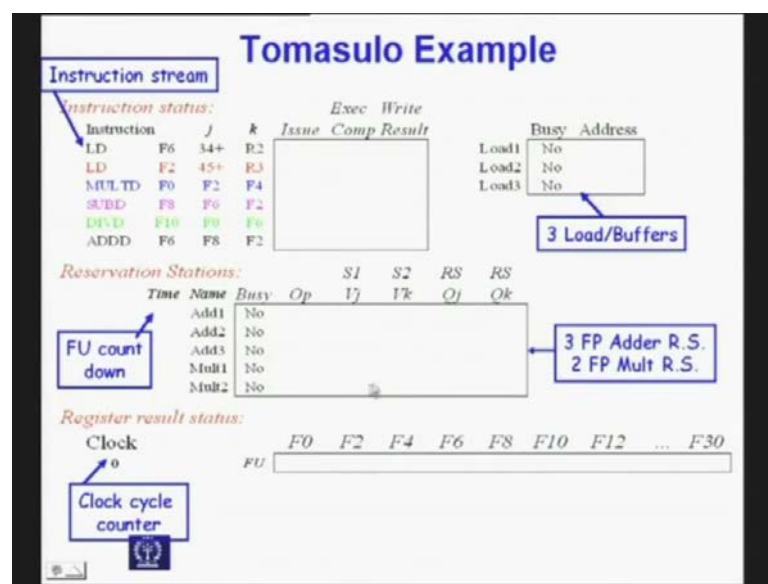
Then V j and V k the value of the source operands so value of the source operands as I mentioned the value generated by functional units or by store I mean load. These are all generated with the help of with the help of are stored in this registers V j and V k. The value of the store operands are stored. And this V j indicates that this reservation station and it is a accompany functional units are occupied. So, this will help you to overcome

structural hazard by looking at the busy flat bit. And third type of form third functional unit that is third part is register status the register file. And store buffer each has a have a field Q I. So, if Q I if the value is blank no currently active instruction is a computing a result destined for this register.

That means here it pentanes about the register status the up whenever a particular instruction will produce a result that will be stored in a register. Now, if the if this field is blank that means all know already issued instructions are producing in result that will be stored in that register so it implies that. So, these are the different fields that are the that is to be maintain to maintain this status of the processor and the load and store buffers each has a field a which holds the result of the effective address once the first step of if execution has been completed.

So, the load and store buffers initially it actually takes the instruction from the instruction register I mean as the instruction is fetched then it calculates the effective address. So, effective address is calculated in the second step. And that is being stored in the in the load and store buffer that means initially it is the it is done in 2 steps. First; the, you know that whatever is available in the instruction and by using that the effective address this is calculated by adding the value of the program counter with the value that is available in the instruction. And then that effective address is stored in the load and store buffer.

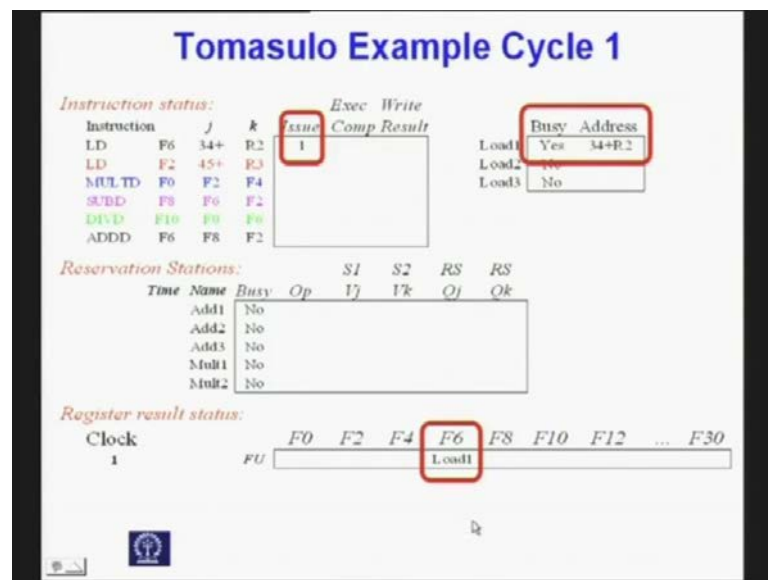
(Refer Slide Time: 41:27)



So, let me illustrate the Tomasulo's algorithm with the help of an example. And here is the, here you have got this is the it is maintaining the clock cycle counter clock 0. We shall start with clock 0 then you have got 3 load and store buffers there load you have got 3 load units then this is the instruction stream. That means this is the window on which the Tomasulo's algorithm will be working this is the instruction window. These are the instructions to be executed and here the functional unit countdown information is stored.

Different functional unit will take different time for example, we shall see multiplication will require 10 clock cycles division will require still longer number of clock cycles addition I mean for fixed point will require 2 clock cycles so that information stored here and time information's. So, initially it may have the value 2 or 10 and then it will be decremented. And these are the reservation stations corresponding to different functional units. We have got 5 functional units 3 floating point adder reservation stations are available here and 2 floating point multiplier divider which are also available in this particular. As we have seen this Tomasulo's algorithm was primarily developed to enhance the performance of the floating point unit. So, that is why you have got information about only the floating point units.

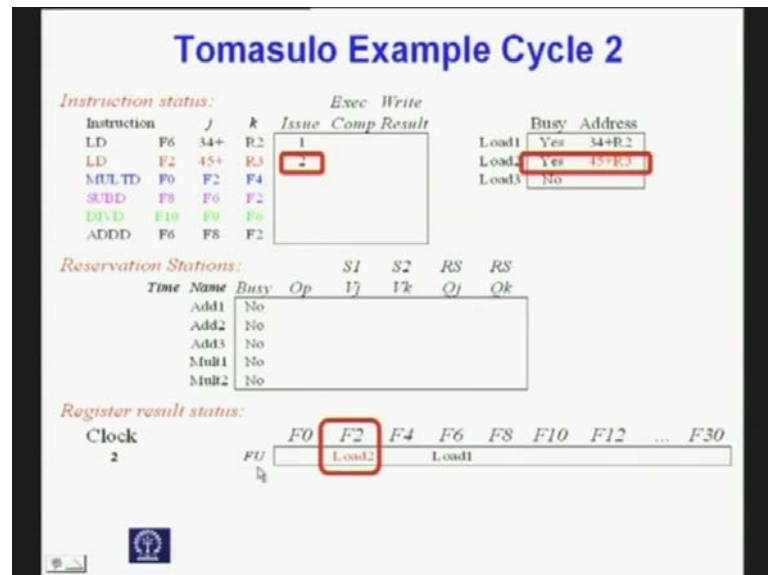
(Refer Slide Time: 43:15)



Now, we have started the clock this is the clock cycle 1 and instruction 1 has been issued. And accordingly it is a load instruction that load 1 unit has become busy. And the fetched the address be calculated is 34 plus the content of R 2. So, that will be I mean actually

the register R 2 will have the value. And with that you have to ADD 34 to get the effective address.

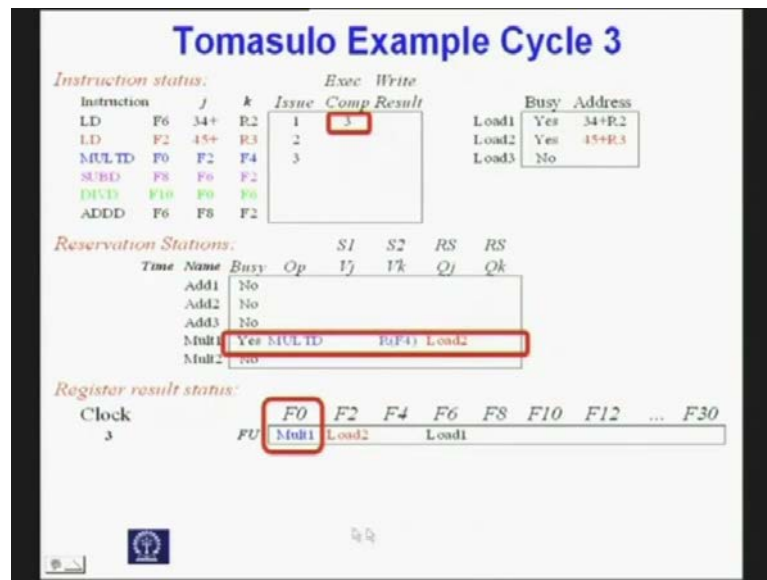
(Refer Slide Time: 43:47)



And then in a next cycle the instruction 2 is issued which also is a load instruction fortunately we have got another load instruction I mean load unit which will perform that load operation. And the; it gets busy and the corresponding effective address will be calculated from here so and you can see the functional units where who on the register on which that F 2 and F 6 where the value will be loaded by different functional units. And load 1 functional unit will load in floating point register 6 load 2 will load in floating point register 2 these are also maintain.

And right now as you can see none of the reservation stations are busy at this moment. Because only load and store instructions have been issued load instructions have been issued no arithmetic instructions have been issued so far. Now, suppose there are a several load instructions say of 3 or 4 or five 4 or 5 in such a case only 3 load instructions can be issued remaining instructions cannot be issued because of structural hazard. So, but in this particular code sequence you have got only 2 load instructions so there is no problem.

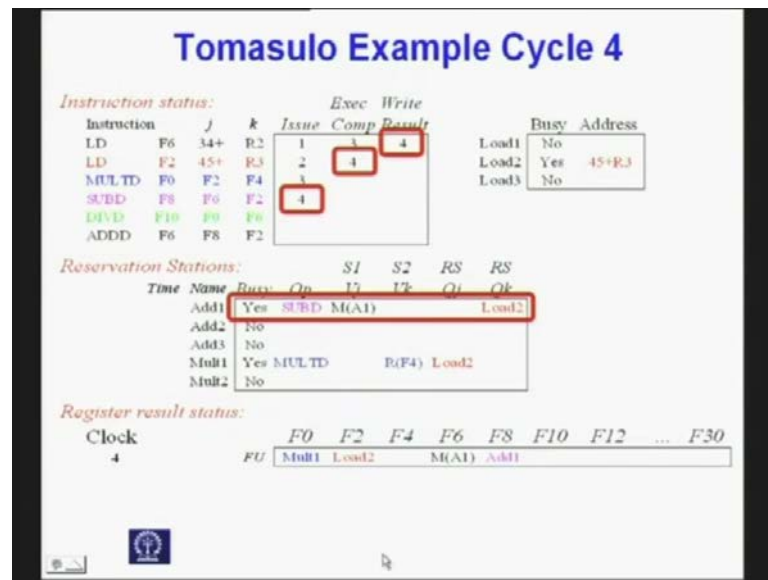
(Refer Slide Time: 45:13)



Now, the execution of this load operation will be performed that means it will load the value into the register from the by reading it from the memory. So, execution will start proceed it will take 2 clock cycles and it will that. And then that multiplication double instruction has also been issued in the third clock cycle. And here F 0 is the destination register that is been reflected here and corresponding functional unit gets busy as you can see the, this multiplication multiplier 1 is now busy and it will 2 multi operations to be performed is mention here the address is mention here and it will that load 2.

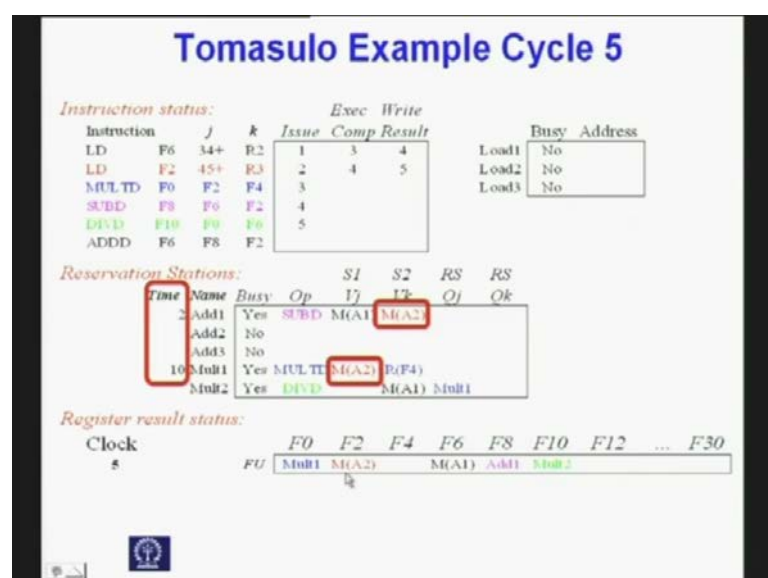
The second operand will come from load 2. So, you see here is maintain mentioning from which functional units the data will be available. And it will directly load the value in this register. So, register names are removed renamed by the reservation stations So, you can see registers names are not there load 1 is completing. So, one is completing and it will be loaded into the register 6.

(Refer Slide Time: 46:42)



So, it completes it and you can see the, that writing has taken place into the register F 6 and the forth instruction has also been issued in the forth cycle. And the load instruction is in the execution stage here it is maintain and accordingly the effective address is calculated by the here. So, that it can do the execution and you can see the functional units are maintaining the destination register rather from which functional unit the operands should be available that is being maintain here. So, load 2 is completing now and it will be writing into the float F 2 register F 2 register here load 2 will write it into that register.

(Refer Slide Time: 47:43)



So, MA2 you can see it writes from its reads from the memory and writes it into writes it will write into the register. But you can see here the not only it is writing into the register, it is also writing into the reservation stations. So, MA2 MA2, you can see those values are written into the register that means that F 2 that is needed here F 2 that is needed here. These are directly written into the registers because V j and V k are holding the values. So, these values are directly after reading from the memory they are written into these registers in V j and V k.

So, operand values are available for this ADD 1 so ADD 1 that means this particular instruction can be in so you can be started. And you can see the time required to execute this instruction is 2 cycles. So, 2 is being mention and it will be decremented as we shall go to the next cycles similarly, see the multiplier that multiplication operation floating point multiplier will require 2 clock cycles and it has got both the operands. Now, both the operands are available here also both the operands are available so they will start execution in a next clock cycle. So, timer starts down for ADD 1 and multiplier 1 as I have said.

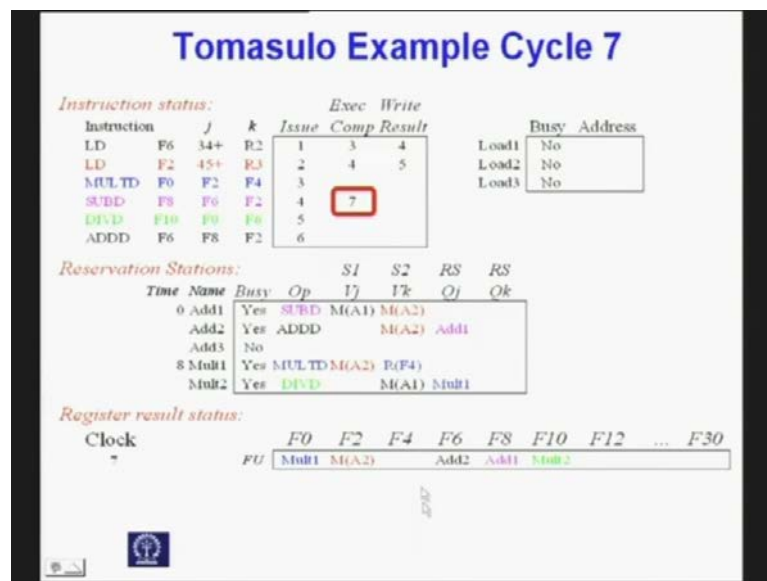
(Refer Slide Time: 49:12)

Tomasulo Example Cycle 6									
Instruction status:				Exec Write				Busy Address	
Instruction	J	k	Issue	Comp	Result			Load1	Load2
LD F6 34+	R2		1	3	4			No	
LD F2 45+	R3		2	4	5			No	
MULD F0 F2 F4			3					No	
SUBD F8 F6 F2			4						
DIVD F10 F8 F6			5						
ADD F6 F8 F2			6						
Reservation Stations:									
Time	Name	Busy	Op	Ij	Ik	Qj	Qk		
1	Add1	Yes	SUBD	M(A1)	M(A2)				
2	Add2	Yes	ADD	M(A2)	Add1				
3	Add3	No							
9	Multi1	Yes	MULD	M(A2)	R(F4)				
10	Multi2	Yes	DIVD	M(A1)	Multi1				
Register result status:									
Clock		F0	F2	F4	F6	F8	F10	F12	... F30
6	FU	Multi1	M(A2)	Add2	Add1	Multi2			

So, in a next clock cycle 2 has become 1 and 10 has become 9 and execution has started for that for those 2 instructions. And all these instructions have now been issued, you can see as we have reached the clock cycle 6. All the instruction cycles have been issued because we have the necessary functional units and reservation stations.. So, since there

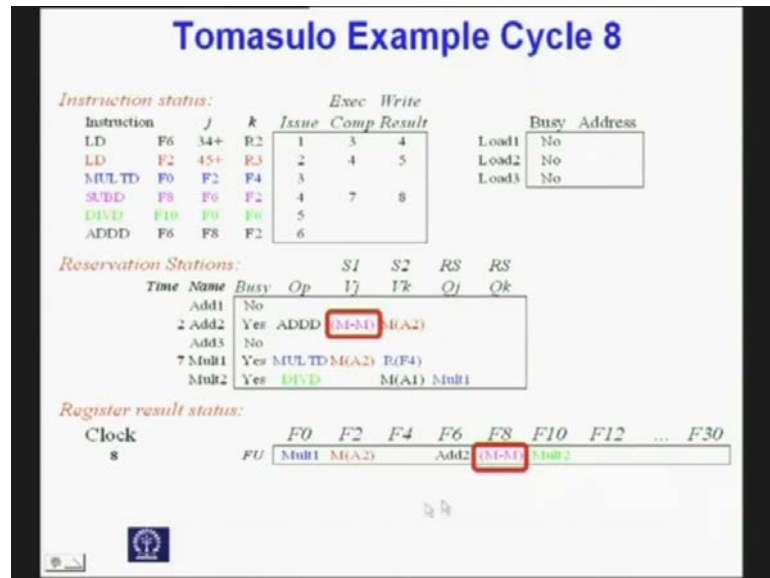
is no structural hazard the, all these instructions have been issued. But you can see different instructions and in are in different stages the instruction 1 and 2 have been completed. And now the remaining 4 instructions have been issued, but the execution is proceeding for instruction 1 I mean that that instruction 4 and multiplication that is your instruction 3. So, issue ADD D here despite name dependency. So, there is a name dependency, but in spite of that it has been issued.

(Refer Slide Time: 50:22)



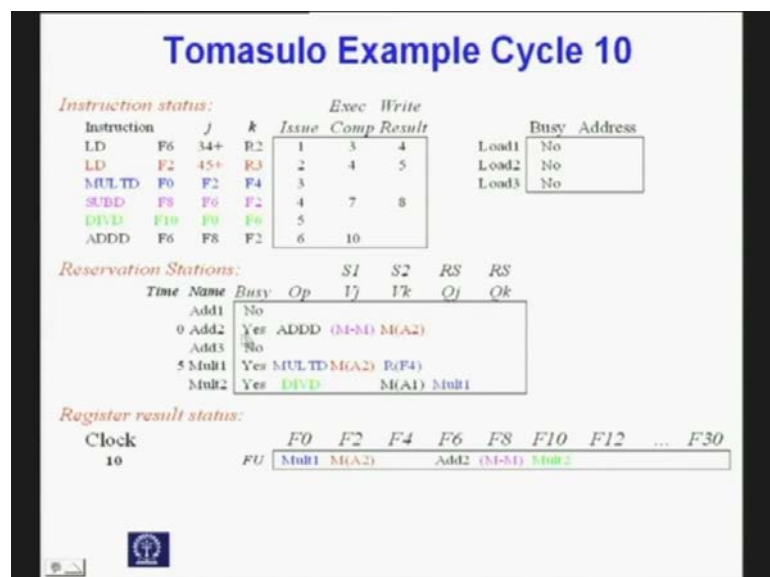
And now this instruction execution has been completed and as it is shown here that timer value has become 0. And operands are available it will write in the appropriate register that is your F 8 it will write in a F 8. So, ADD 1 adder 1 will write in F 8 that is begin and shown here and as we go to the so this instruction is completing.

(Refer Slide Time: 50:54)



Now, you can see we have reached the clock cycle 8 and the first second and forth instructions have been completed. So, you see not only the instructions issue was out of order in order issue to place. But out of order execution have been done and out order out of order completion has taken place. So, out of order completion has also taken place for instruction 4 and instruction. Now, the other instructions will proceed for example, this the, this second instruction that is your this ADD D that means 6 instruction. Now, we will has got operands it will be now be in the execution stage. And it will require 2 clock cycles and multiplication will require seven clock cycles.

(Refer Slide Time: 51:48)



Here this is instruction execution is complete as you can see it is the time is 0. So, execution has been completed by getting the operands from the registers V j and V k. So, V j and V k have provided the operand values and the, this operation has been performed. And it will now be written into the F 6 register in the next cycle.

(Refer Slide Time: 52:16)

Tomasulo Example Cycle 11

Instruction status:

Instruction	J	k	Issue	Comp	Result	Busy	Address
LD F6 34+ R2			1	3	4	Load1	No
LD F2 45+ R3			2	4	5	Load2	No
MUL.TD F0 F2 F4			3			Load3	No
SUBD F8 F6 F2			4	7	8		
DIVD F10 F0 F6			5				
ADDD F6 F8 F2			6	10	11		

Reservation Stations:

Time	Name	Busy	Op	Ij	Ik	Qj	Qk
Add1		No					
Add2		No					
Add3		No					
4 Mult1		Yes	MUL.TD	M(A2)	R(F4)		
Mult2		Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
11	Mult1	M(A2)		(31-31+5)	M(A1)	Mult2			

So, in the next cycle the operand the writing has been taken place and the now only instruction that is been I mean that has been that is in execution is multiplication. division has not yet started, because it has to it has to get the both the operands only then division will be started. So, one operand is yet to come here as you can see the V j is not yet available so it is waiting for the operand.

(Refer Slide Time: 52:46)

Tomasulo Example Cycle 12

Instruction status:

Instruction	J	k	Issue	Comp	Result	Busy	Address
LD F6 34+	R2		1	3	4	Load1	No
LD F2 45+	R3		2	4	5	Load2	No
MULTD F0 F2 F4	F4		3			Load3	No
SUBD F8 F6 F2	F2		4	7	8		
DIVD F10 F0 F6	F6		5				
ADDD F6 F8 F2	F2		6	10	11		

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
				Vj	Vk	Qj	Qk
Add1	No						
Add2	No						
Add3	No						
3 Mult1	Yes	MULTD	M(A2)	R(F4)			
Mult2	Yes	DIVD		M(A1)	Mult1		

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30	
12	FU Mult1 M(A2) (M1-M1) (M1-M1) Mult2									

And writing has taken place multiplication will continue.

(Refer Slide Time: 52:51)

Tomasulo Example Cycle 13

Instruction status:

Instruction	J	k	Issue	Comp	Result	Busy	Address
LD F6 34+	R2		1	3	4	Load1	No
LD F2 45+	R3		2	4	5	Load2	No
MULTD F0 F2 F4	F4		3			Load3	No
SUBD F8 F6 F2	F2		4	7	8		
DIVD F10 F0 F6	F6		5				
ADDD F6 F8 F2	F2		6	10	11		

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
				Vj	Vk	Qj	Qk
Add1	No						
Add2	No						
Add3	No						
2 Mult1	Yes	MULTD	M(A2)	R(F4)			
Mult2	Yes	DIVD		M(A1)	Mult1		

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30	
13	FU Mult1 M(A2) (M1-M1) (M1-M1) Mult2									

Multiplication will continue.

(Refer Slide Time: 52:52)

Tomasulo Example Cycle 14

Instruction status:

Instruction	J	k	Issue	Comp	Result	Busy	Address
LD	F6	34+	R2	1	3	4	
LD	F2	45+	R3	2	4	5	
MULD	F0	F2	F4	3			
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULD M(A2)	R(F4)			
	Mult2	Yes	DIVD	M(A1)	Mult1		

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
14									

(Refer Slide Time: 52:53)

Tomasulo Example Cycle 15

Instruction status:

Instruction	J	k	Issue	Comp	Result	Busy	Address
LD	F6	34+	R2	1	3	4	
LD	F2	45+	R3	2	4	5	
MULD	F0	F2	F4	3	15		
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULD M(A2)	R(F4)			
	Mult2	Yes	DIVD	M(A1)	Mult1		

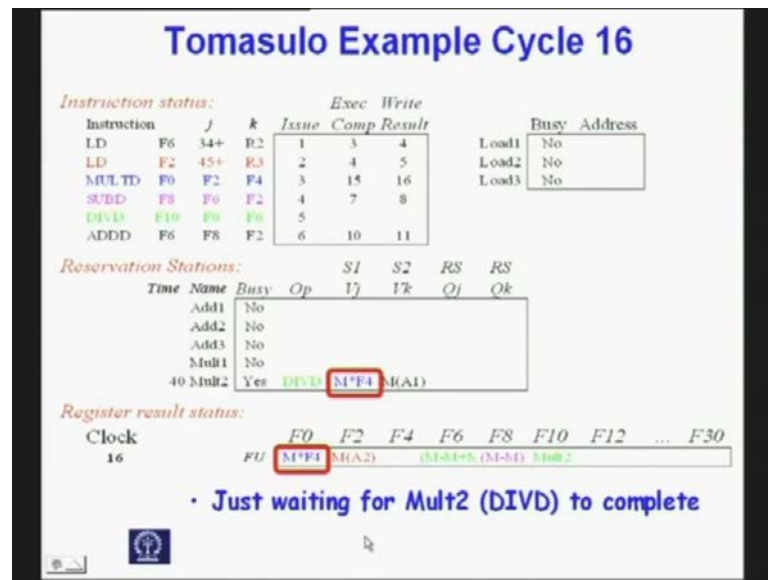
Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
15									

• Mult1 (MULD) completing; what is waiting for it?

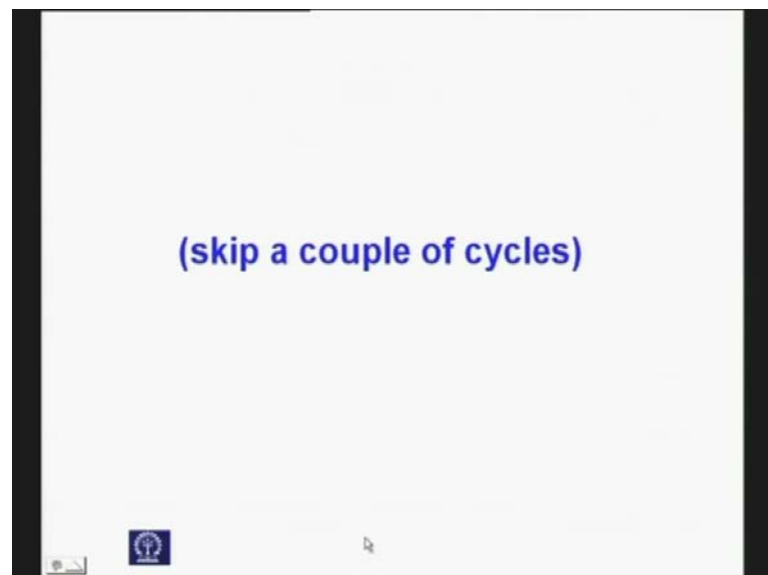
And multiplication operation is now completing.

(Refer Slide Time: 53:00)



And it will write the result in a next cycle so multiplication is also complete. Now, only instruction left is the divide D and divide D will require 40 clock cycles. So, it will continue, but fortunately both the operands are now available and it will continue to execute.

(Refer Slide Time: 53:24)



So, you have to you can skip a couple of cycles.

(Refer Slide Time: 53:27)

Tomasulo Example Cycle 55

Instruction status:

Instruction	J	k	Issue	Comp	Result	Busy	Address
LD	F6	34+	R2	1	3	4	
LD	F2	45+	R3	2	4	5	
MULTD	F0	F2	F4	3	15	16	
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
55									
	M*F4	M(A2)		(S1-S1)*5	(M-S1)	Mult2			

So, it will complete in and you have come to clock cycle 55. So, in clock cycle 55 we can see this division operation will require 1 more clock cycle to complete.

(Refer Slide Time: 53:39)

Tomasulo Example Cycle 56

Instruction status:

Instruction	J	k	Issue	Comp	Result	Busy	Address
LD	F6	34+	R2	1	3	4	
LD	F2	45+	R3	2	4	5	
MULTD	F0	F2	F4	3	15	16	
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5	56		
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	M(A1)		

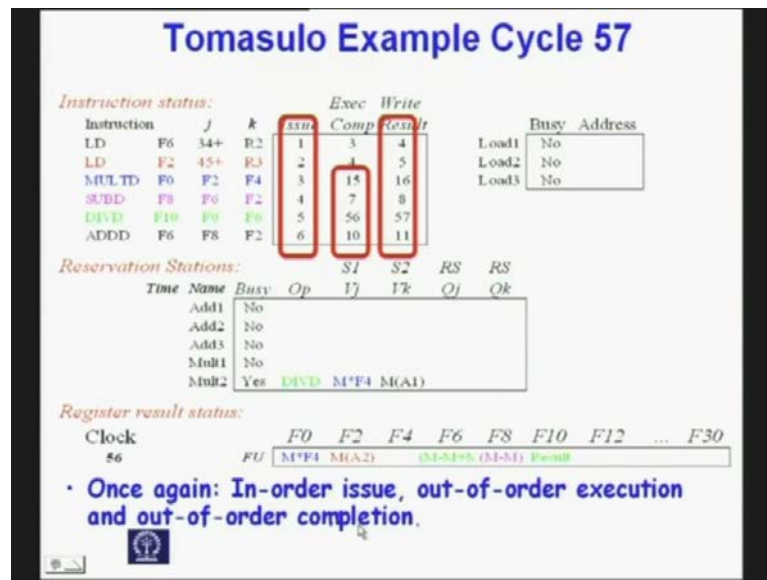
Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
56									
	M*F4	M(A2)		(S1-S1)*5	(M-S1)	Mult2			

• Mult2 (DIVD) is completing; what is waiting for it?

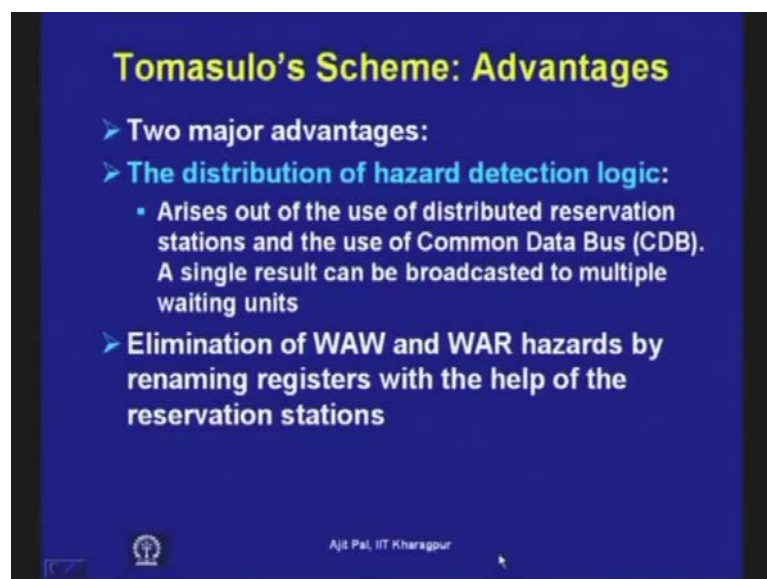
Another 1 clock cycle. So, execution is now complete and it will perform the write writing into the appropriate register that is an F 10 it will write multiplication 2 that hardware you will write.

(Refer Slide Time: 53:53)



It as F 10 So, you can see the execution of these all these instructions have been completed. And when have been issue the issue have taken place in order as you can 1 2 3 4 five 6, but execution has taken place out of order. And completion has also taken place out of order, but Tomasulo's algorithm has overcome the hazards that can arise because of you know that WAW and WAR type hazards. And we have seen how it is it has been done with the help of suitable hardware so in order issue out of execution and out of order completion.

(Refer Slide Time: 54:42)



So, these are the 2 major advantages the distribution of hazard detection logic you have seen arises out of the use of distributed reservation stations. And the use of common data bus a single result can be broadcasted to multiple functional units and elimination of WAW and WAR hazards by renaming registers with the help of the reservation stations. So, without registers renaming has been done with the help of reservation stations by writing the values into the registers.

(Refer Slide Time: 55:16)

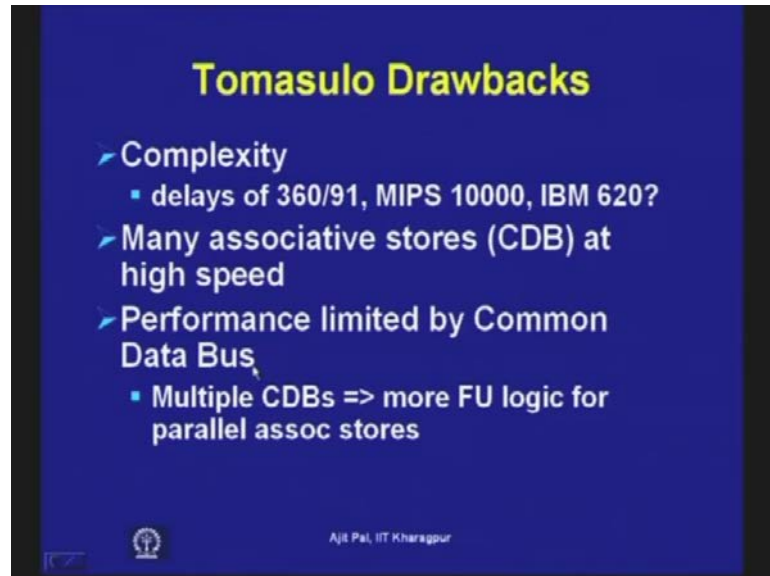
Tomasulo v. Scoreboard (IBM 360/91 v. CDC 6600)	
Pipelined Functional Units (6 load, 3 store, 3 +, 2 x/÷) window size: ≤ 14 instructions No issue on structural hazard WAR: renaming avoids WAW: renaming avoids Broadcast results from FU Control: reservation stations	Multiple Functional Units (1 load/store, 1 +, 2 x, 1 ÷) ≤ 5 instructions same stall completion stall completion Write/read registers central: scoreboard

And here is a comparison between Tomasulo's algorithm and scoreboard. So, the here in in CDC 6600 multiple functional units were used in IBM 360 91. Pipelined functional units have been used 6 load, 3 store, 3 adder, 2 multiplier divider and so on. And window size you can see is larger than the scoreboard approach here it is 6, 14 instructions in place of 5 instructions in case of CDC 6600. And there is on issue on structural hazard, because structural hazard is taken care of by the, because in order issue is taking place so that takes care of structural hazard.

WAR renaming is avoided WAW and WAR 2 hazards, but in case of CDC 6600 in scoreboard approach you have seen it leads to stall and here broadcast result from the functional units. But in case of CDC 6600, we have seen these were written into the registers. And then from the from the registers the reading was taking place and we have seen in case of for that Tomasulo's algorithm control is distributed with the help of

multiple reservation stations in case of CDC 6600 centralized control with the help of scoreboard.

(Refer Slide Time: 56:46)



Of course, in this world nothing is one sided there are some drawbacks number one is complexity, because of the complexity there is lot of delays of 360 91 MIPS 1000 and many associated stores. You require associative stores; you require large number of registers; you require in your hardware then performance limited by common data bus. As we know whenever the in large number of devices are connected to a bus the capacitances large and it becomes slow and as a consequence there is performance limitation because of the common data bus. So, you can go for multiple common data bus and more functional unit for parallely associated stores, but using a single data bus those are the limitations.

(Refer Slide Time: 57:43)

Tomasulo Loop Example

Loop: LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

- Assume Multiply takes 4 clocks
- Assume first load takes 8 clocks (cache miss?), second load takes 4 clocks (hit)
- To be clear, will show clocks for SUBI, BNEZ
- Reality, integer instructions ahead

Ajit Pai, IIT Kharagpur

Let us stop here today and subsequently we shall see how Tomasulo's algorithm can overcome I mean beyond the control loops. And before that we shall discuss about the control hazards in my subsequent lectures.

Thank you.