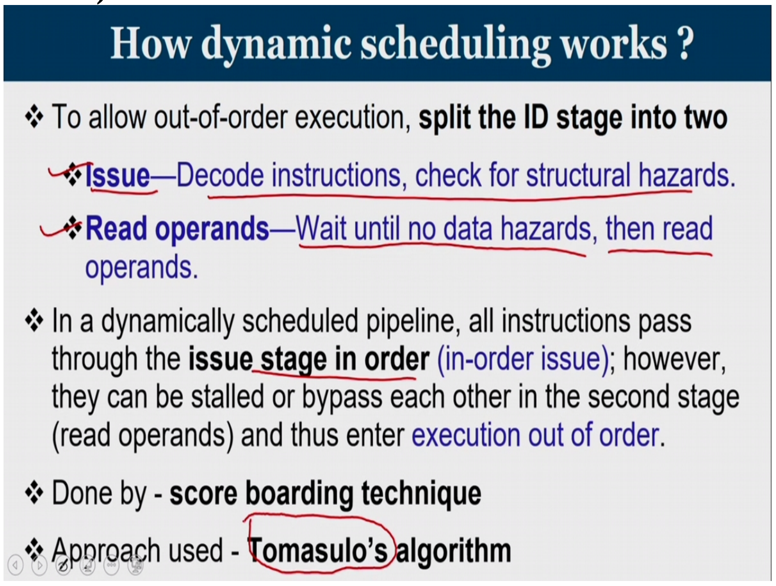
Advanced Computer Architecture

Dr. John Jose, Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati, Assam

Lecture 9
Dynamic Scheduling with Tomasulo's Algorithm

Welcome to lecture number nine. In today's lecture, we are focusing on dynamic scheduling in instruction pipeline and specifically our target is to understand what is Tomasulo's algorithm. So, we will have a first to recap on the concepts that we learned in the last lecture, wherein dynamic scheduling was introduced, this will help you to appreciate Tomasulo's algorithm in a better way.

(Refer Slide time: 00:55)



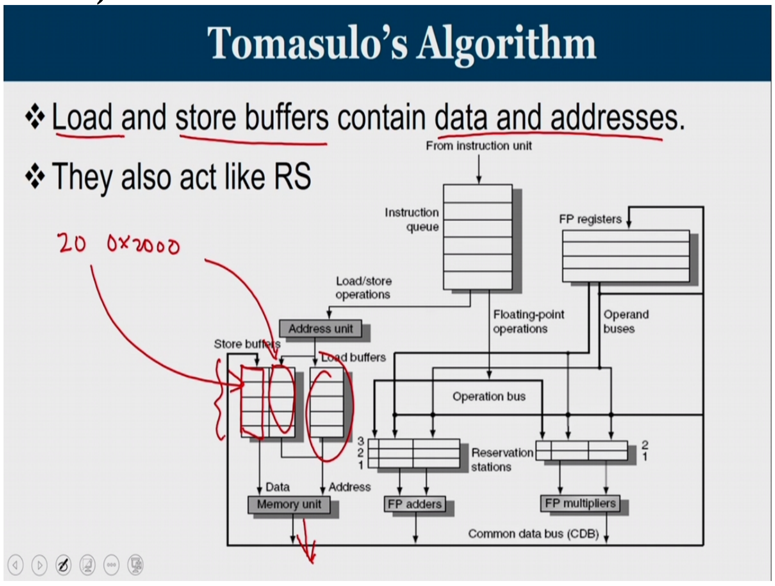
How dynamic scheduling works ?

- To allow out-of-order execution, **split the ID stage into two**
  - **Issue**—Decode instructions, check for structural hazards.
  - **Read operands**—Wait until no data hazards, then read operands.
- In a dynamically scheduled pipeline, all instructions pass through the **issue stage in order** (in-order issue); however, they can be stalled or bypass each other in the second stage (read operands) and thus enter execution out of order.
- Done by - **score boarding technique**
- Approach used - **Tomasulo's algorithm**

So, how basically dynamic scheduling is works this is what we have seen earlier. To allow out-of-order execution, what we do is we are splitting the ID stage in 2. 1 is the issue stage where in, a decoding of instructions and checking for structural hazard happens. So, at the end of the issue stage your functional unit is ready, and you are been assigned to these functional units and then, while waiting for entering into the functional unit, the read operands is performed where you wait until there is a data hazard.

So, as long as the issue is done, then you will reach the reservation station associated with the functional unit and assignment operand is available to perform execution. So, in a dynamically scheduled pipeline, all instructions pass through issue stage in order However, they can be stalled or bypass each other in the second stage, and thus entering execution out of order fashion. And today, our focus is on to understand Tomasulo's algorithm, and what is the basis of dynamic scheduling.

(Refer Slide time: 01:55)



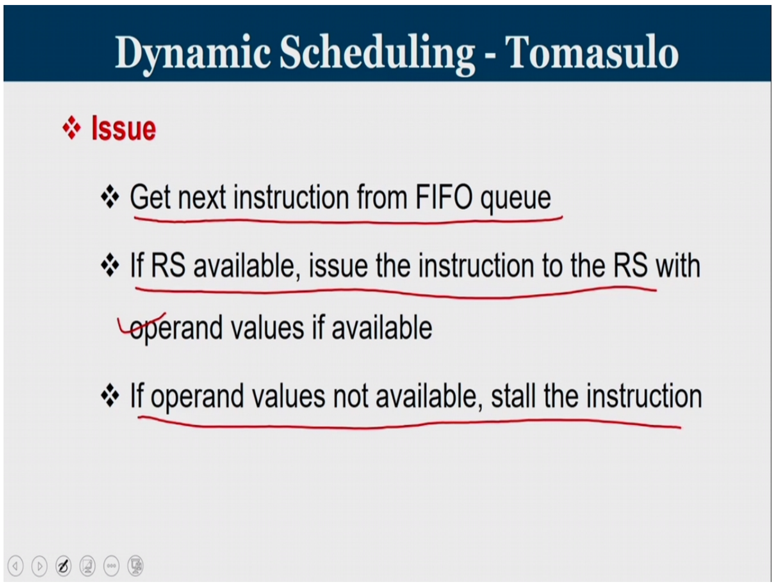
Tomasulo's Algorithm

- Load and store buffers contain data and addresses.
- They also act like RS

So, Tomasulo's algorithm you have your load and store buffers contain data as well as an address. So, you are store buffers, this is the place where you store the address and this is the place where we are going to store the corresponding data to be stored. So, for example, let us say I wanted to store a value to 20 into the address at say hexadecimal 2000. So, this 2000 will be stored in the address portion and the 20 will be stored in the data portion and it is going to be a queue as on when you are going to access memory using this address.

We are going to work on it. Similarly, for loading, the value will be taken from memory. So, the by-product of node will come only after from memory. So, we need only the address components. So, here also that the number of entries as far as this memory unit is concerned can be considered as a reservation station.

(Refer Slide time: 02:52)

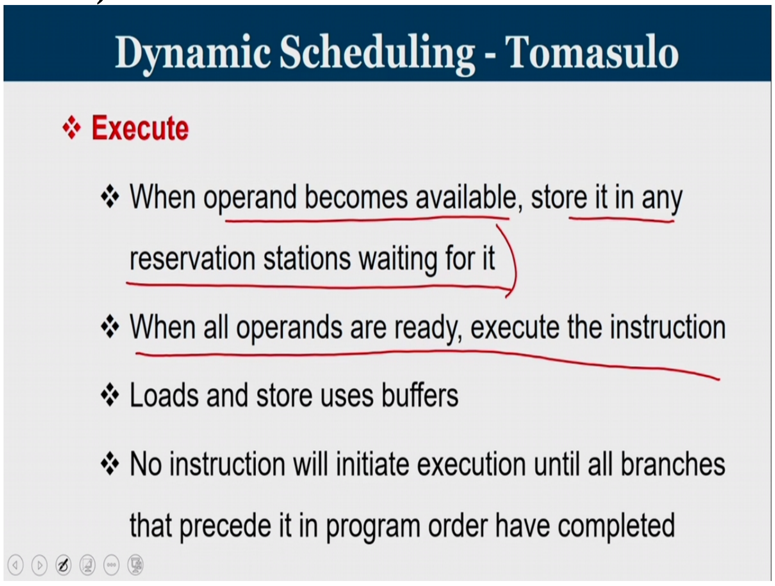


Dynamic Scheduling - Tomasulo

- Issue
  - Get next instruction from FIFO queue
  - If RS available, issue the instruction to the RS with operand values if available
  - If operand values not available, stall the instruction

Now, in dynamics scheduling, we have basically issue is the very first stage what we will do is get the next instruction from the FIFO queue that is the instruction that you got after fetching if the reservation session is available issue the instruction to the reservation station, if the operand value is there, if the operand value is not there, then we are going to wait in the reservation station.

(Refer Slide time: 03:16)

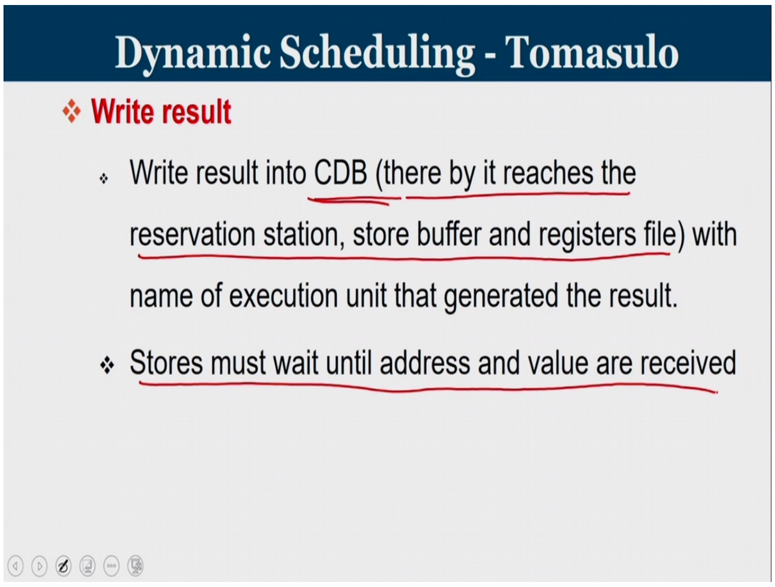


Dynamic Scheduling - Tomasulo

- Execute
  - When operand becomes available, store it in any reservation stations waiting for it
  - When all operands are ready, execute the instruction
  - Loads and store uses buffers
  - No instruction will initiate execution until all branches that precede it in program order have completed

The coming to the execute stage when all the operands are available, you store it on the reservation station and then you wait for it when all operands are ready you are going to execute. Loads and store also use buffers and no instruction will initiate execution until all branches that preceded the program model have completed. So, we do not have a mechanism to address the control hazard at this point and coming to write result.

(Refer Slide time: 03:41)



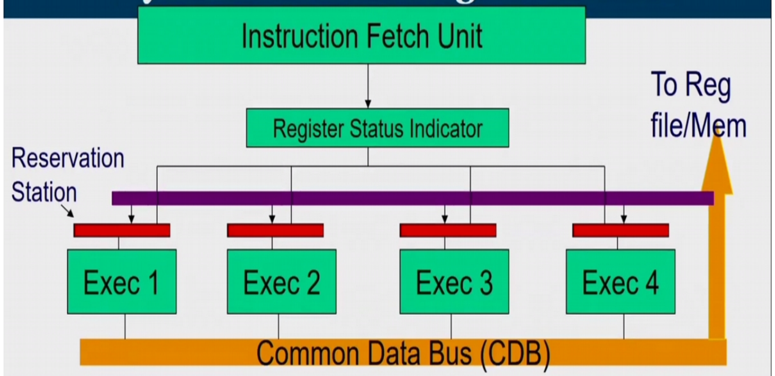
Dynamic Scheduling - Tomasulo

- Write result
  - Write result into CDB (there by it reaches the reservation station, store buffer and registers file) with name of execution unit that generated the result.
  - Stores must wait until address and value are received

As mentioned, we are going to have a common data bus to which the output of all the functional unit is being connected. So, write the result in the common data bus there by it reaches the reservation stations store buffer and the register file with the name of execution unit that generated the result and for all the stores they will wait until the address and the value are been properly received.

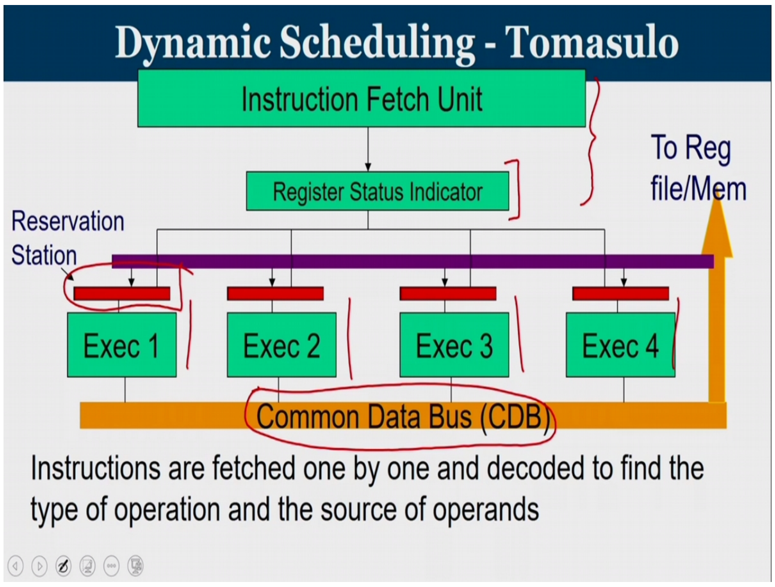
(Refer Slide time: 04:04)

Dynamic Scheduling - Tomasulo



Now, for the rest of Tomasulo’s algorithm, we will first try to see how the functional unit looks like. So, this slide will help you to understand Tomasulo’s algorithm in a better abstraction.

(Refer Slide time: 04:19)



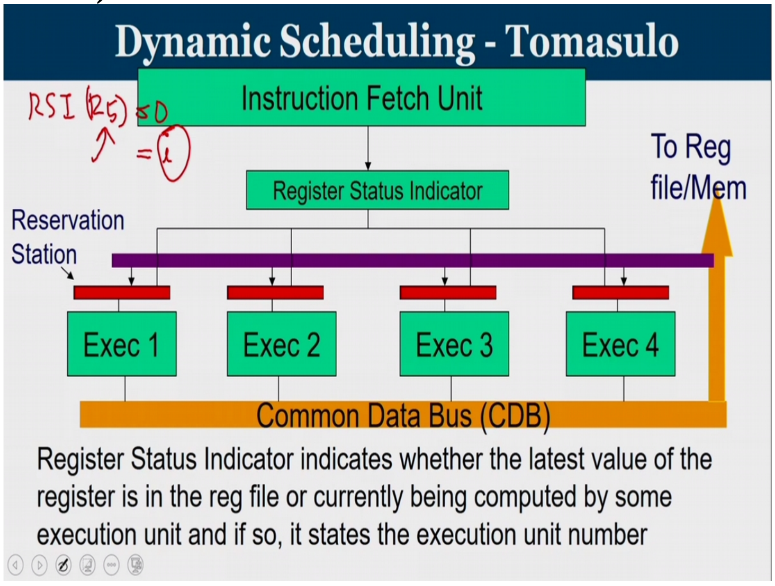
Dynamic Scheduling - Tomasulo

Instructions are fetched one by one and decoded to find the type of operation and the source of operands

You have an instruction fetch unit from where instructions are been brought. And that has been kept in memory and you have a unit called register status indicator, the purpose of the register status indicator is to mention about what is the status of a register value. For example, at this particular point, the content of register whether it is really available in the register file or whether it will be produced by some execution unit and then we have various execution units.

And as mentioned all execution units are connected to a common data bus. And the inputs out of execution unit we are having the reservation stations which basically queue to enter into these execution units. Now instructions are fetched one by one and decoded to find the type of operation and the source of operands.

(Refer Slide time: 05:17)



Dynamic Scheduling - Tomasulo

Register Status Indicator indicates whether the latest value of the register is in the reg file or currently being computed by some execution unit and if so, it states the execution unit number

We have the register status indicator that indicates whether the latest value of a register is in the register file or currently being computed by some execution unit. If so, it states the execution unit number. So, if the registered status indicator of a register let us say R5 if that value equal to 0 means go to the register file R5 that will contain the latest value of registers. If this value is any number, let us say ‘I’ which is not 0, the meaning is at this particular point, the latest value of the register R5 is not available in the register file. Rather it will be a value that is going to be produced by your functional unit whose number is ‘I’.

So, register status indicator is a very important component or a facilitator which will produce you the latest value associated with the variable.

If RSI = 0

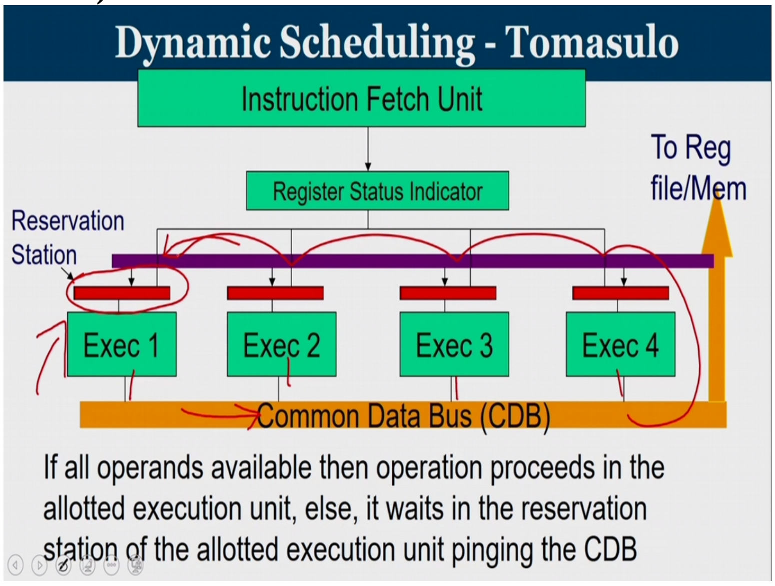
means you are value is up to date go to the register file take the value and that is the latest one. If the RSI value is not equal to 0, then it is actually mentioning a functional unit number

if RSI = 10

10th functional unit to produce a value and that is to be written into this register.

So, there is no point in going into register and taking the value whatever be the value that is produced by the 10th functional unit take it and use. So, it is actually a reflection of what is the latest value associated with a register.

(Refer Slide time: 06:54)



Dynamic Scheduling - Tomasulo

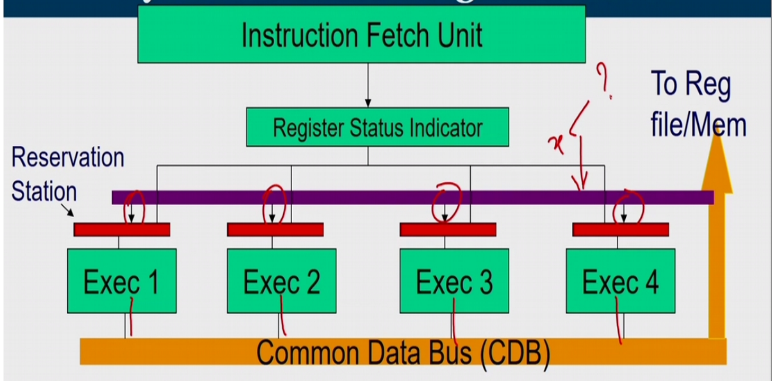
If all operands available then operation proceeds in the allotted execution unit, else, it waits in the reservation station of the allotted execution unit pinging the CDB

Now, if all operands are available, then operation proceeds in the allotted execution unit. So, this is the place where you generally wait. So, you have your operation details and the operands are available there if operands are available, then it is ready for execution as and when the execution unit is free, you are going to enter. If operands are not available, the only possible way by which you get the operands is through the common data bus that has been connected. And once the data is ready, you are going to update your reservation station there by entering into execution. Now, how will you get these values you can see that all execution units are connected to the common data bus. So, when we can say that the operand is not ready, when the value is still under process in some of the functional units. So, as on when this functional unit completes the value is return into the common data bus, then it is going to update your reservation stations.

So, if all operands are available, then operation proceeds in the allotted execution unit else it waits in the reservation station of the allotted execution unit bringing the value of the CDB.

(Refer Slide time: 08:00)

Dynamic Scheduling - Tomasulo

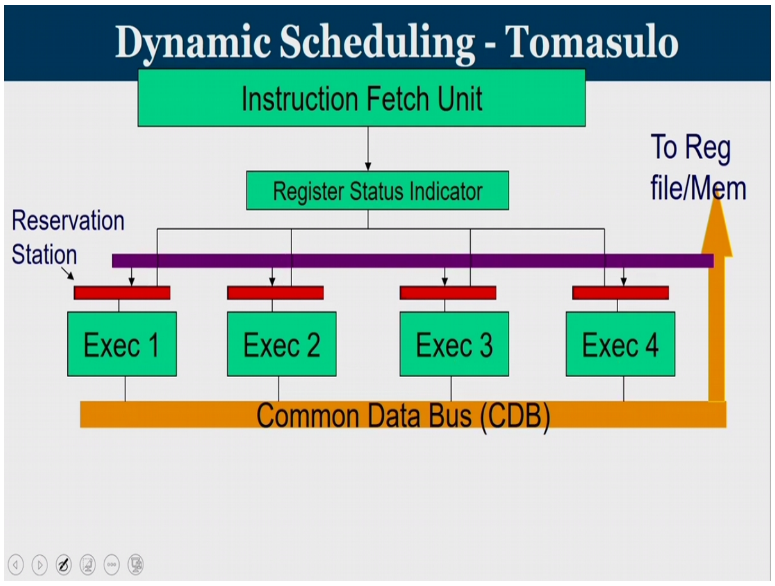


Every Execution unit writes the result along with the unit number on to the CDB which is forwarded to all reservation stations, Reg-file and Memory

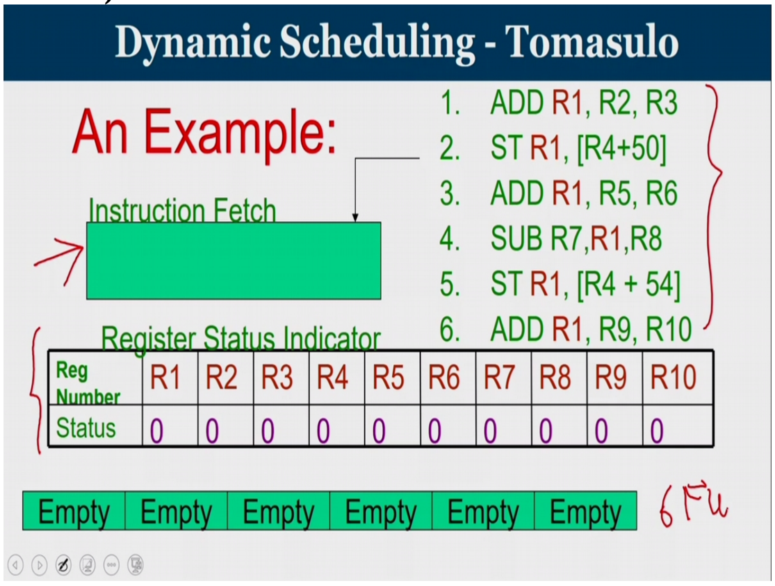
Every execution units write the result along with the unit number on to CDB, which is forward to all the reservation stations register file and memory. Now, the concept we have to understand is you have a common data bus to which many execution units are being directly connected. Now, if somebody watches here, let us say a value x is available in the CDB, the execution units which are waiting for some value, how will they know that this x is produced by which functional unit.

So, in this case, whenever a functional unit produces a value, it has to mention the functional unit name also that is very important. Is the functional unit name is not mentioned then the reservation station entries may not be able to distinguish between which functional unit produced this value.

(Refer Slide time: 08:58)



(Refer Slide time: 08:59)


Dynamic Scheduling - Tomasulo

An Example:

1. ADD R1, R2, R3
2. ST R1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7,R1,R8
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	0	0	0	0	0	0	0	0	0	0

Empty	Empty	Empty	Empty	Empty	Empty

6 FU

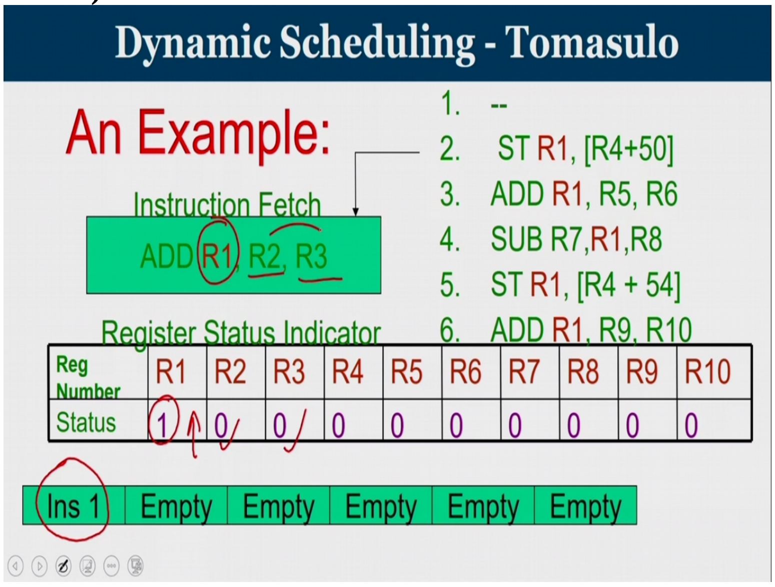
Now, let us take a simple example to understand how these register status indicator values are being operated on. Consider a scenario where you have a program to execute the program is already given, there are 6 instructions in the program. Now you have 10 registers from R1 to R10, and register status indicator is already shown here. So, this is the RSI value of this 10 registers. This box will tell you what is the current instruction that we are operating on and we have total 6 functional units.

These are the 6 functional units, and these functional units are going to operate on. For the time being let us assume all these functional units all these 6 functional units are identical. So whatever be the operation, it can be assigned to any functional unit. Now see, what is the meaning of registered status indicator at this point, the RSI value RSI stands for registered status indicator of all the 10 registers are 0.

If RSI = 0,

the meaning is for all these registers, the updated value is available in the register file itself. So go and fetch from the register file that is your operand. Having said this, let us move into the example.

(Refer slide time: 10:20)



Dynamic Scheduling - Tomasulo

An Example:

1. --
2. ST R1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7,R1,R8
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

ADD R1, R2, R3

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	1	0	0	0	0	0	0	0	0	0

Ins 1	Empty	Empty	Empty	Empty	Empty

So, consider the first instruction

add R1, R2, R3

all instructions of the format the first operand is going to be the destination. So, R2 and R3 are been added and you are going to write the result into R1. Now, in this case, you have to see that what is the value of R2. Value of R2 was 0, what is the value of R3. R3 also is 0. That means both the operand values are available inside the register file. So go to register file take the value of R2 and R3 and you are waiting in the reservation station of functional unit one.

Now in the functional unit 1 you can see that since both operands are R2 and R3 value are already available, it will start execution its instruction at this point, the R1 value is to be updated by functional unit 1 that is why

RSI value of R1 = 1

So, if anybody who wants the value of R1 they should not go to register file they should look into the value that is produced by execution unit number 1. This 1 indicate is the name of the functional unit that will produce a value to be updated to R1. That is the role of this.

So with this, our first instruction over. The change that is going to happen since the source operand of the instruction R2 and

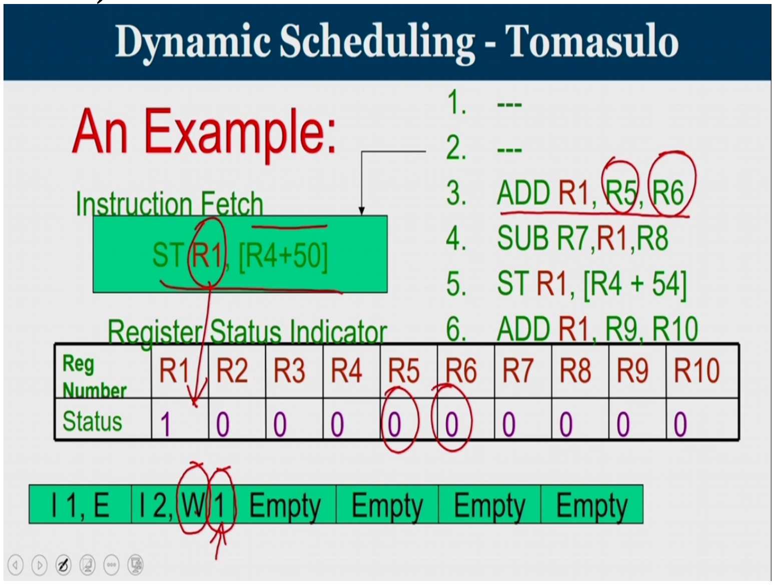
R3 RSI = 0

operands are available, perform the operation in the execution unit 1 and update the

RSI =1

because 1 will produce the value to be returned to R1.

(Refer Slide time: 11:50)



Dynamic Scheduling - Tomasulo

An Example:

1. ---
2. ---
3. ADD R1, R5, R6
4. SUB R7,R1,R8
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

ST R1, [R4+50]

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	1	0	0	0	0	0	0	0	0	0

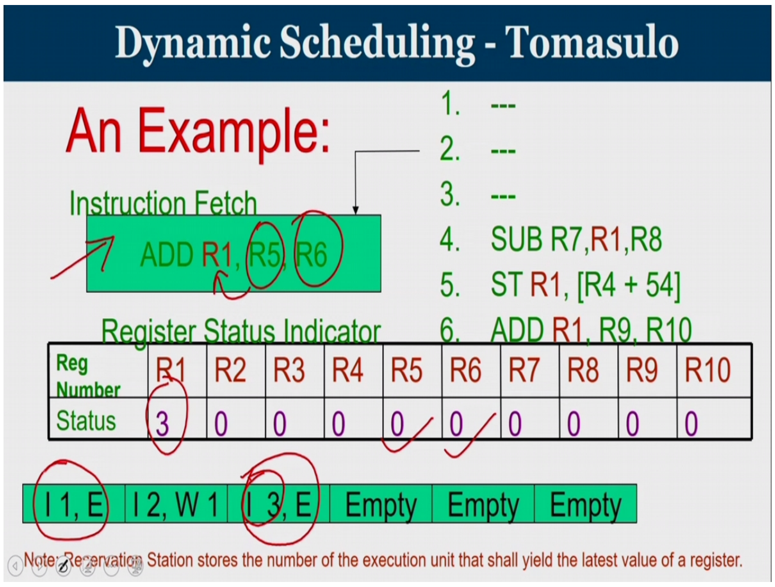
I 1, E	I 2, W 1	Empty	Empty	Empty	Empty

Moving on to the next instruction, here I am going to perform a store where the source operand is R1. So, the content of R1 has to be returned to a memory location whose address is computed by R4 and 50. Now we will to see that at this point value of R1 is not available. So, this instruction will be forwarded to the second functional unit where in you are in the waiting stage and you are waiting for the result that is produced by 1.

Because of this point, RSI value of R1 which is operand 1. Meaning by value it is not available, it will be produced by some other functional unit. So it is a name of that functional unit that is being written over here. So, second instruction cannot execute it is waiting for a result produced by the first functional unit. Now, let us try to see what is the third instruction, if you look at the third instruction, it is R5 and R6 these are the 2 operands.

These operands if you look at the RSI value, the operands are available. I can perform the operation and I will write the result into R1.

(Refer Slide time: 13:00)



Dynamic Scheduling - Tomasulo

An Example:

1. ---
2. ---
3. ---
4. SUB R7,R1,R8
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

ADD R1, R5, R6

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	3	0	0	0	0	0	0	0	0	0

I 1, E	I 2, W 1	I 3, E	Empty	Empty	Empty

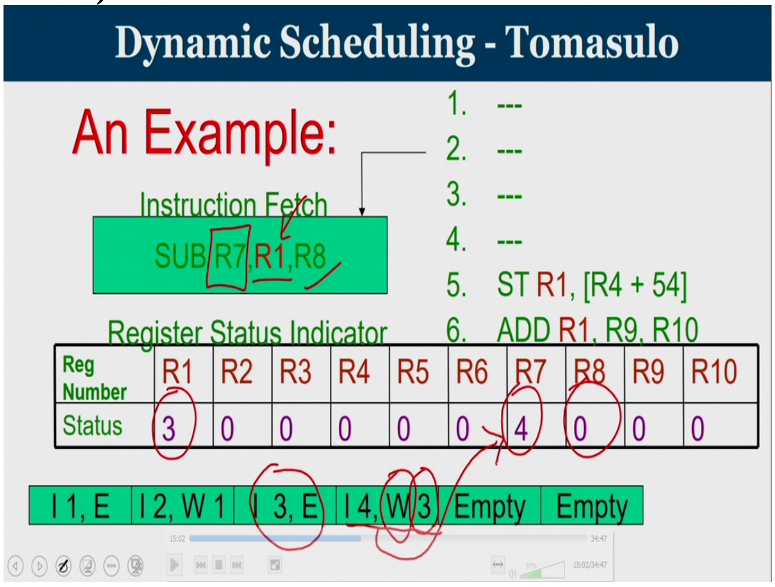
Note: Reservation Station stores the number of the execution unit that shall yield the latest value of a register.

So, looking at the third instruction R5, R6 are the source operands and RSI value tells that

$$RSI = 0$$

that means you can go into the register file get the data since operands are available, I will assign them into the third execution unit.

(Refer Slide time: 13:59)



Dynamic Scheduling - Tomasulo

An Example:

1. ---
2. ---
3. ---
4. ---
5. ST R1, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

SUB R7,R1,R8

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	3	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	Empty	Empty

So, the third execution unit will perform. This is the third execution unit. Your I 3, this is the I 3 is a third instruction is going to execute. Now at this point the result needs to be returned to R1. So R1, RSI value is made to three. So anybody who is going to look for value of R1 after this point, there should not go to register file, they should not wait for a value which has been produced by the first execution unit, it should only look for the value that is produced by the third execution unit.

Now, moving on to the fourth instruction, here R1 is your source operand. You can see that. The RSI value of R1 is 3. The RSI value of R1

RSI R1 = 3

means R1 value is not readily available, you have to wait for the result produced by the third execution unit. So, upon completion of this execution unit only you will get R1. Now coming to R8 RSI value

R8 RSI =0

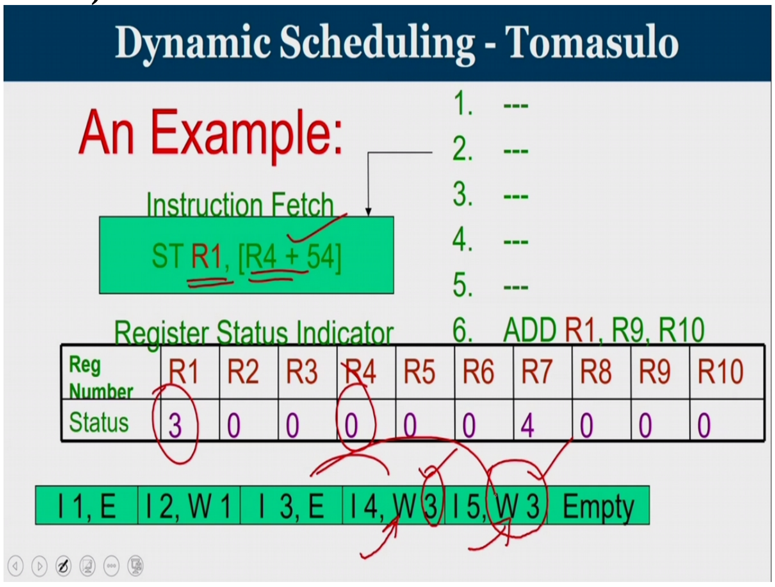
And so, 1 of the operand is available the second option by how to wait that is the reason that the fourth instruction is in the waiting stage in the fourth functional unit.

Waiting for a result that is produced by the third functional unit and the result is to be returned to R7. So, at from this point

(RSI value) R7 = 4

because the result of that is produced by the fourth to functional unit will go to R7. So, anybody wanted the value of R7 from this point, it has to wait in the result from the fourth functional unit.

(Refer slide time: 15:03)



Dynamic Scheduling - Tomasulo

An Example:

Instruction Fetch

ST R1, [R4 + 54]

1. ---
2. ---
3. ---
4. ---
5. ---
6. ADD R1, R9, R10

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	3	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	I 5, W 3	Empty	

Moving further, we are going to perform a store operation on R1 when you perform a store operation of R1 we know that the

R1 RSI = 3

So you cannot proceed. So it is going to wait for a result that is produced by the third functional unit. And R4 is available because

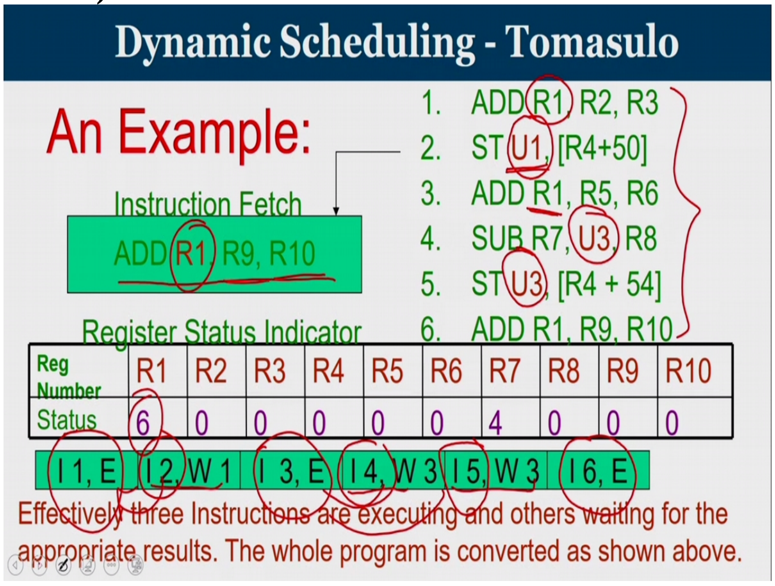
RSI value of R4 = 0

So, 1 of the operand is available you can compute the address, but you cannot perform the operation until the third unit produces the result.

Now, if you see there is 1 instruction which is waiting for a result produced by the third functional unit. Similarly, there is another instruction also that is going to wait for a result that is produced by the third functional unit. So there is a dependency between the third instruction to the fourth and to the fifth the both are waiting together. Now if you see the moment third produces result, both instructions 4 got its data.

And instruction 5 got data, both will execute parallellys in 2 of these functional units. Let us now move in to the last instruction.

(Refer Slide time: 16:07)



Dynamic Scheduling - Tomasulo

An Example:

1. ADD R1, R2, R3
2. ST U1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7, U3, R8
5. ST U3, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

ADD R1, R9, R10

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	6	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	I 5, W 3	I 6, E

Effectively three Instructions are executing and others waiting for the appropriate results. The whole program is converted as shown above.

The last instruction is

add R1, R9 and R10

So here if you see after this instruction R1, RSI valuable will become 6 and since it is R9 and R10. You are going to have your sixth instruction executing directly. So you have your third instruction executing you have your first instruction executing, and you have your sixth instruction executing. The peculiarity of this is in all these 3 instruction I 1, I 3 and I 6, the operands are available, since operands are available, it is not waiting for anything else as long when you get functional unit you start executing. Whereas your second, fourth and fifth instructions are waiting second is waiting for a value of R1. So second is waiting for R1, fourth and fifth also waiting for R1 but they are waiting for different versions of R1. I 2 wait for I 1 and I 4 and I 5 wait for I 3.

So, effectively, 3 instructions are executing and others are waiting for appropriate result. So, if you look at the whole program can be converted like this. You have your result to be produced in R1. Now, that result, which will be produced by unit number 1 is going to be used here also you are going to produce the result R1. But unit number 3 is whatever is the result produced by you unit number 3 act as an option for fourth instruction as well as the fifth instruction.
(Refer Slide time: 17:45)

So operand forwarding has automatically happened. So, what happened is this is operand forwarding. This is also operand forwarding. You are going to forward the result through some functional unit and register renaming also happened. So if you look at you can see that there is an R1 where I am going to write the result. So all going to write to different time to the same register R1.

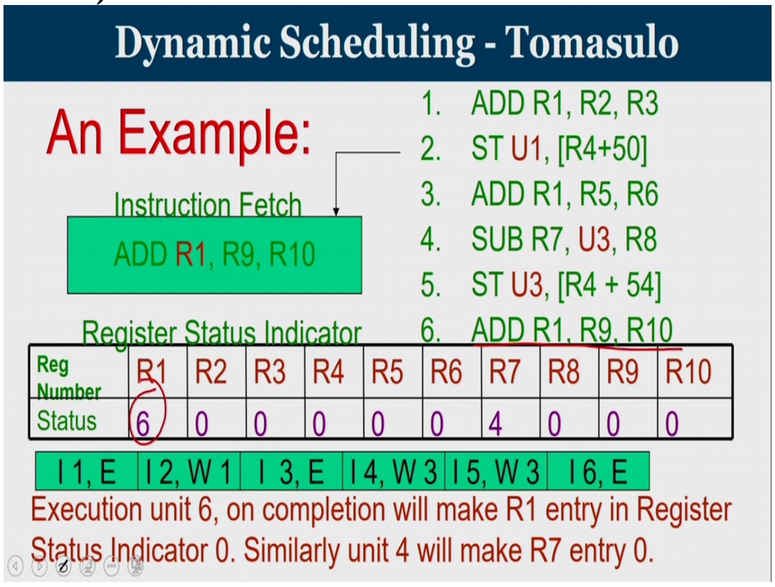
But this WAW hazard is not happening because instruction those who want 1 version of R1. So only instruction 2 want this R1. So, to 2 I am not telling you should go to R1 you are telling the functional unit. Whereas 4 and 5, want this result of R1 they are not given directly access to R1 rather they are been asked to read the value that has been produced by U3 and after instruction 6, they will look into the result produced by the sixth functional unit.

So, if you look into the example that has been discussed right now. It is very simple. You have a sequence of instruction you take an instruction, see what is the operation to be done, if the functional unit is free, you are going to assign them into the appropriate reservation station. The operand is there how will you know whether a operand is there look at register status indicator of the value is 0.

Go and fetch the operand value from the register file and keep the operand ready and wait in the reservation station associated with the functional unit. If operand is not available which you will come to know by looking at RSI value, so you are waiting in the reservation station pinging the common data bus. So as on when the functional unit produces a result the reservation station which are continuously pinging the CDB will grab the result get the operand and then start executing.

So, reservation station and register status indicator are playing a very crucial role in handling operand forwarding in handling register renaming associated with the instruction pipeline.

(Refer Slide time: 19:55)



Dynamic Scheduling - Tomasulo

An Example:

1. ADD R1, R2, R3
2. ST U1, [R4+50]
3. ADD R1, R5, R6
4. SUB R7, U3, R8
5. ST U3, [R4 + 54]
6. ADD R1, R9, R10

Instruction Fetch

ADD R1, R9, R10

Register Status Indicator

Reg Number	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
Status	6	0	0	0	0	0	4	0	0	0

I 1, E	I 2, W 1	I 3, E	I 4, W 3	I 5, W 3	I 6, E

Execution unit 6, on completion will make R1 entry in Register Status Indicator 0. Similarly unit 4 will make R7 entry 0.

Now, the execution unit 6 on completion will make

R1 =0 in the register status indicator

So as on when the sixth instruction completes the operation it writes into R1 and then make

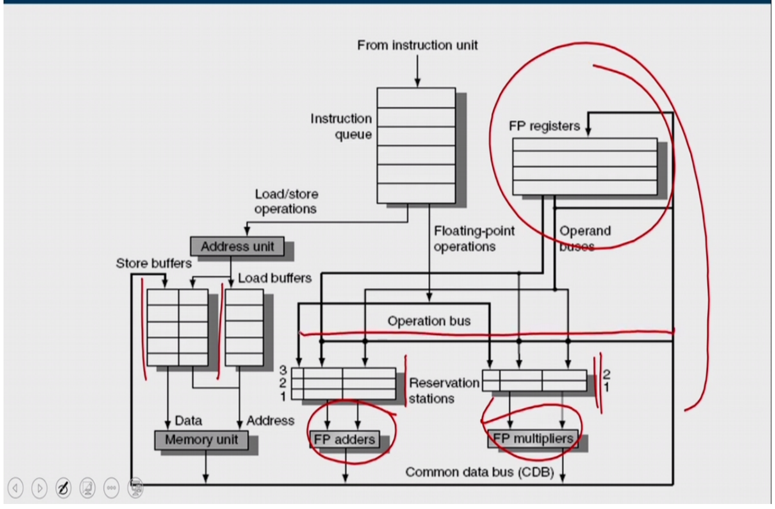
RSI = 0

Similarly unit 4 will make the

R7 = 0

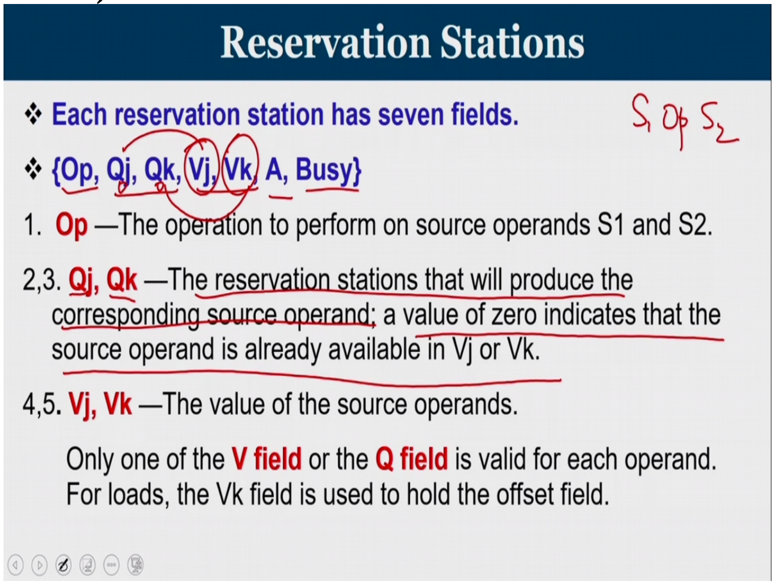
(Refer Slide time: 20:13)

Tomasulo's Algorithm



So, this is how the whole structure looks like we have had a discussion we have reservation station. This is the reservation station for adders, you have a reservation station for multipliers. So every functional unit has its own reservation station. This queue is acting as the reservation station for memory unit for load and store. We call it us load buffer and store buffer and CDB updates is floating point registers or whatever is your register file. At the same time CDB is connected to the input of the reservation stations also.

(Refer Slide time: 20:45)



Reservation Stations

- Each reservation station has seven fields.
- {Op, Qj, Qk, Vj, Vk, A, Busy}

S_1 Op S_2

1. **Op** —The operation to perform on source operands S1 and S2.

2,3. **Qj, Qk** —The reservation stations that will produce the corresponding source operand; a value of zero indicates that the source operand is already available in Vj or Vk.

4,5. **Vj, Vk** —The value of the source operands.

Only one of the **V field** or the **Q field** is valid for each operand. For loads, the Vk field is used to hold the offset field.

So, now let us see what is the content of your reservation station. Each reservation station has 7 fields. We represented it as operation Q j and Q k that is 1couple, V j and v k that is another couple, A for address field and there is a busy field. Now the Op tells what is operation to be performed on 2 source operand S1 and S2. So if the operation is going to be S1 Op S2. The Op field will tell you what is operation let us say it is add operation, logical shift operation like that.

Now, the second and third field is Q j and Q k. If the operand value is not available in the register, because your registered status indicator is reflecting like that, then you will get the operand from the output of some functional unit. So Q j and Q k indicates the functional unit that will produce the result the reservation station that will produce the corresponding source operand. The value of 0 indicates that the source operand is already available in V j or V k.

So, this Q j and V j are coupled. Q k and V k are coupled. If value of

$$Q\,j = 0$$

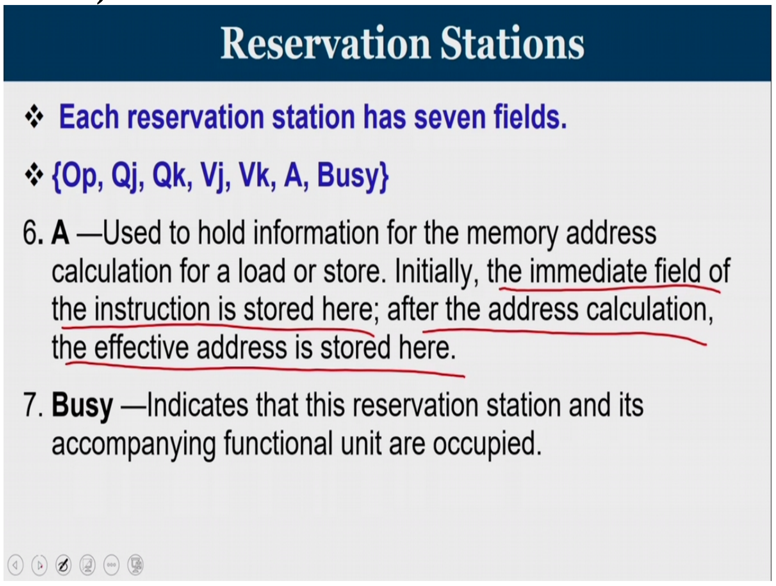
that means V j is your operand value of

Q k = 0

then V k is the operand. So V j and V k are populated from taking the value directly from the register file. If Q j and Q k contains a nonzero value, then V j and V k is not at all used because Q j and Q k will tell you the functional units that will produce the result. So V j V k is a value of the source operand taken from the register file. Only one of V field or the Q field is valid for each operand.

So, if Q j is valid, then V j is not valid. Similarly, if Q k is valid, then V k is not valid and vice versa. And for loads V k field is used to hold the offset field.

(Refer Slide time: 22:52)



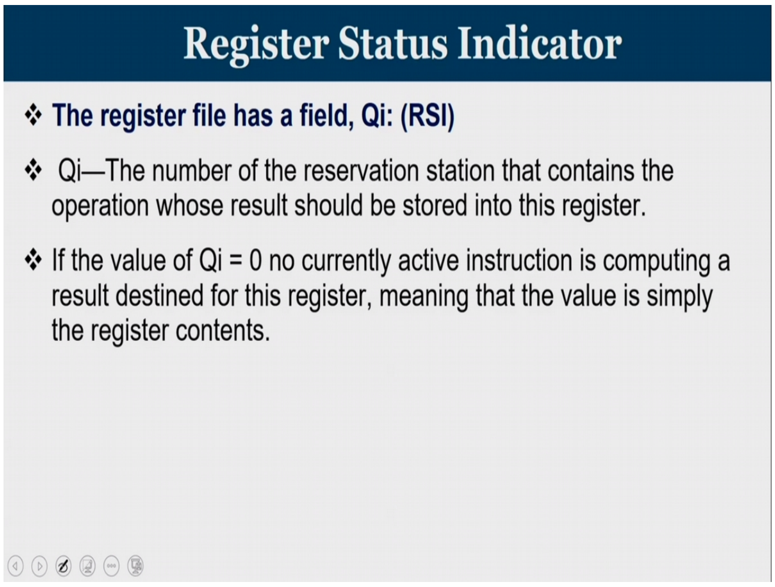
Reservation Stations

- Each reservation station has seven fields.
- {Op, Qj, Qk, Vj, Vk, A, Busy}

6. **A** —Used to hold information for the memory address calculation for a load or store. Initially, the immediate field of the instruction is stored here; after the address calculation, the effective address is stored here.
7. **Busy** —Indicates that this reservation station and its accompanying functional unit are occupied.

Then coming to A, A is used to hold information for memory address calculation for a load or a store. So, initially the immediate field of instruction is stored here and after the address calculation, the effective address is stored in this field. And busy indicates that there is a reservation station and its accompanying functional unit are currently occupied.

(Refer Slide time: 23:17)



Register Status Indicator

- The register file has a field, Qi: (RSI)
- Qi—The number of the reservation station that contains the operation whose result should be stored into this register.
- If the value of Qi = 0 no currently active instruction is computing a result destined for this register, meaning that the value is simply the register contents.

So the register file has a field known as Qi that is known as RSI. So, Qi is a number of the reservation station that contains the operation whose result should be stored into this register. If the value of

$$Qi = 0$$

not currently active instruction is computing result destined for this register, meaning that the value is simply the register content. So, if

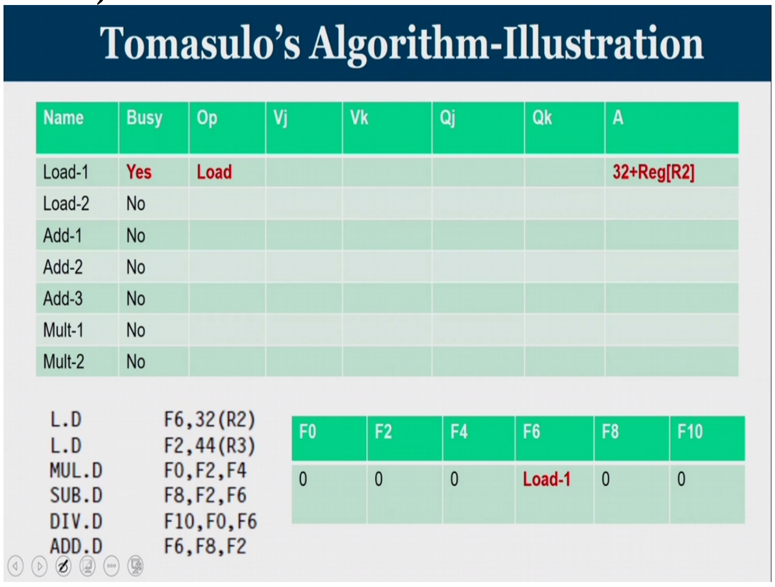
$$Qi = 0$$

then RI value is available in the register file. If

$$Qi = n$$

and then RI value will be updated by functional unit number n.

(Refer Slide time: 23:53)



Tomasulo's Algorithm-Illustration

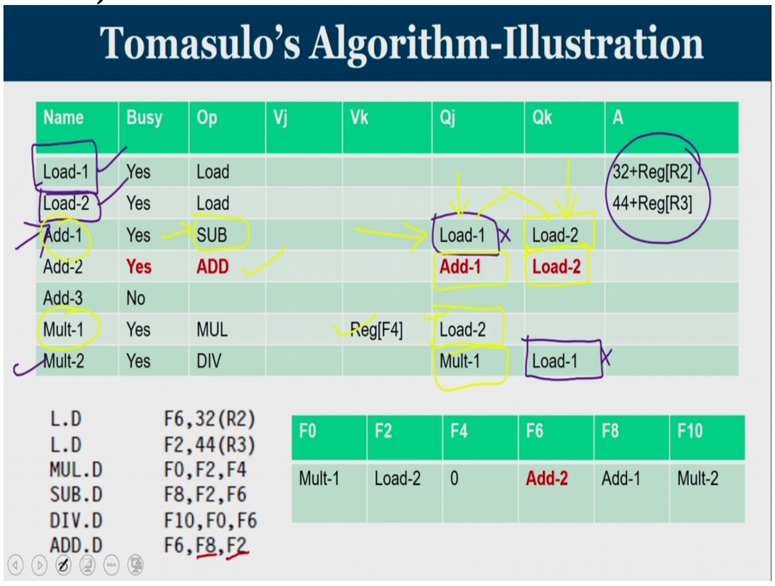
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	Yes	Load					32+Reg[R2]
Load-2	No						
Add-1	No						
Add-2	No						
Add-3	No						
Mult-1	No						
Mult-2	No						

```
L.D     F6,32(R2)
L.D     F2,44(R3)
MUL.D   F0,F2,F4
SUB.D   F8,F2,F6
DIV.D   F10,F0,F6
ADD.D   F6,F8,F2
```

F0	F2	F4	F6	F8	F10
0	0	0	Load-1	0	0

So, we understood the concept of how Tomasulo's algorithm works with the help of an example. Now the get a better clarity a set of instructions has been given and from these instructions, we will try to see how the reservation station values are being updated. So, consider the instruction that is given in the slide.

(Refer Slide time: 24:14)



Tomasulo's Algorithm-Illustration

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load-1	Yes	Load					32+Reg[R2]
Load-2	Yes	Load					44+Reg[R3]
Add-1	Yes	SUB			Load-1	Load-2	
Add-2	**Yes**	**ADD**			**Add-1**	**Load-2**	
Add-3	No						
Mult-1	Yes	MUL		Reg[F4]	Load-2		
Mult-2	Yes	DIV			Mult-1	Load-1	

```
L.D     F6,32(R2)
L.D     F2,44(R3)
MUL.D   F0,F2,F4
SUB.D   F8,F2,F6
DIV.D   F10,F0,F6
ADD.D   F6,F8,F2
```

F0	F2	F4	F6	F8	F10
Mult-1	Load-2	0	**Add-2**	Add-1	Mult-2

2 load instructions, let us say we have a multiplication instruction then sub then div and add and you can see this f indicate the name of the registers an R indicates name of integer register. So, let us look into the scenario where I have 2 load store unit load-1 and load-2. 2 adders add-1 and add-2 which can perform addition as well as subtraction. So, we have 3 adders, which can perform adding and subtraction operation and we have 2 multipliers which are capable of performing multiplication operation.

So, we have seen that in the reservation station associated with them, we have a operand field you have a V j V k field. If the value is there in V j and V k that means they are the real operands if Q j and Q k values been filled, then they are the values of the functional unit which will produce the result and A will produce the address. So, first is the load instruction. So

32 + R2 (32) + the register content of R2

that is there in the address.

And then once you do this instruction, the result has to be stored into F6. So, this is the register status indicator that you see on the bottom. So, we are using 5 registers. F0, F2, F4, F6, F8 sorry, we are using 6 registers F0, F2, F4, F6, F8 and F10. So, this indicates that at this point, the value of F6 will be produced by the load-1 unit. Load-1 is nothing but the name of this load unit.

Now, we move to the second instruction. So, when you go to second instruction, you can see that that is going to write the result to F2. So, after this point, F2 RSI value should contain load-2 the name of the functional unit that will produce and it is basically

$$44 + \text{register content of } R3$$

that is there in A field. So, for loads this V j V k Q j Q k fields are not used at this point. Now, we are going to the multiplication instruction, you know that one of the operand of multiplication instruction is F2 the first operand.

The second operand is F4 if you look at F2 the value F2 will be produced by load-2 where value of F4 is 0. So, the first operand F2 is not available, because RSI indicates that load-2 will produce the result. So Q j value will contain load-2 meaning to get the value of F2 you have to wait until load-2 produces the result, whereas F4 is an operand that is available. So, you go to register file the content of the register file is a value of V k.

So V j is blank so at that time I Q j. So, these 2 are coupled together. So, if Q j is nonzero, Don't look into the value of V j. So, here is Q k is 0. If Q k is 0, then they have to look into the value of V k. So, this is basically you have a value 0 here. So first operand is not available. So I wait for the result produced by load-2, second operand is available, that is where the value is kept in V k field.

So, after this load operation, your result is going to be written to F0 that is why from this point onwards the RSI value F0 indicates Mult-1. So, anybody who wanted a value F0 from this point onwards, they how to look into the output produced by Mult-1 do not go to the register file to access. So, if you access F0 from the register file, at this point instead of wrong value, you should not use it the latest value will be produced by Mult-1.

Moving further now, we are moving into the subtraction operation. So I can make use of my adder number 1, let us see what are the operands. F2 is one operand, F6 is another operand. F2 RSI status tells that load-2 will give you the latest value and F6 RSI value tells that load-1 will give you the latest value. So, this is a case where in both operands are not available. So you have to wait. So, in that case Q j will produce first and Q k will produce the second value.

Now, after the subtraction instruction, the result will be returned to F8. That is why F8 RSI value is given to adder number 1. So adder number 1will produce the result to F8. Now we move to the division operation which is been carried out in Mult-2 functional unit look at the operands. The operand is F0 and F6. F6 you know that load unit is going to produce the result. But as F0 it is a multiplier that is going to produce the result.

So, in this case also both Q j and Q k are been filled up with nonzero values means both operands have to wait for the output produced by the functional unit. So, the moment of F0 and F6 generates the values then this reservation entry has the full operand ready and it will move into the functional unit. And now we come to the last instruction that is the ADD instruction that you can see, first operand is F8 second operand is F2.

F8 knows that it is a result of Add – 1 and F2 you have to wait for the result of load-2. So, this is the way how all these instructions are go into work. Now, let us look into the reservation station and we can find that the load instruction which is a very first instruction, the address is ready so as on when you are able to access the memory unit it will produce the result. So, based upon that the load-1 and load-2 will produce some result at the moment.

Load-1 produces a result this operand will be available to the reservation station of adder number 1 and to the reservation station of multiplier number 2 for both of them are actually waiting for an output produced by load-1. So, the moment my load-1 the first load unit produces the result, these 2 reservation station, which are waiting for load-1 will get the operands. The moment load-2 complete then this field is available this field is available and this field.
So there are basically 3 instructions that will wait for the value that is produced by load-2 know if you look at this when load-1 and load-2 both completes, then this subtraction operation can proceed because both of its operands are now available. So, the reservation station of this Add-1 unit, where the subtraction is waiting, they are continuously pinging CDB, the moment the value is produced by load-1, it takes that value.

Similarly, the moment the value is produced by an order to it takes the value once you get both the value it start executing. Similarly, once it completes, then this is ready and this ADD also can continue further. Similarly, the moment multiplication operation is completed. How can multiplication complete? When load-2 completes, that will take one of the data, the second data is already available multiplier 1 will produce a result and when multiplier 1 will produce the result this multiplier 2 is going to make use of that.

So it is actually a cascading effect. So the whole dependency waiting for one data from other dependency relations, any instruction that is waiting for any other instruction where there is a data dependency, it is easily resolved. in this way operand forwarding register renaming handling of WAR and WAW hazard everything is taken care of in the Tomasulo’s algorithm. So, with this we come to the end of today's lecture, just a quick recap of what are the concepts that we learned today.

We understood what is dynamic scheduling, where you have issue is done in in-order and issuing is to the reservation station, there is no reservation session available then issue will not work. The once the issue is over then execution can be out-of-order depending on the availability of operands. While we issue instruction into the reservation station, if operands are available from the register file, take the value and keep the operand along with the instruction waiting in the reservation station.

If the operand is not available, you are waiting for some other previously executing instruction to complete then that execution unit number is mentioned. So the CDB where the output of functional units are being written. It is connected to all the reservation stations as on when the CDB update happens, the reservation station also get the updated value. In this way, Tomasulo’s algorithm gives you the platform by which dynamic scheduling is been done. To get a deeper understanding about this topic.

I request all the students to kindly refer the textbook Hennessy and Patterson and read it through multiple times. And if there is any doubt, feel free to ping us in the online forum. Thank you.