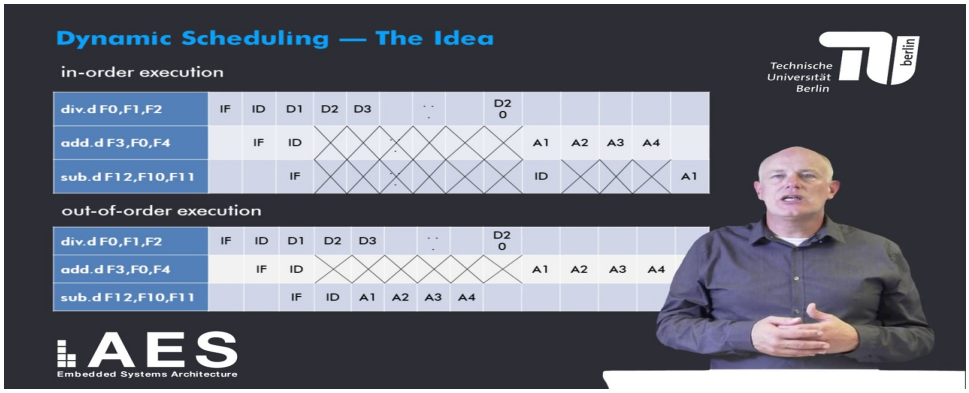
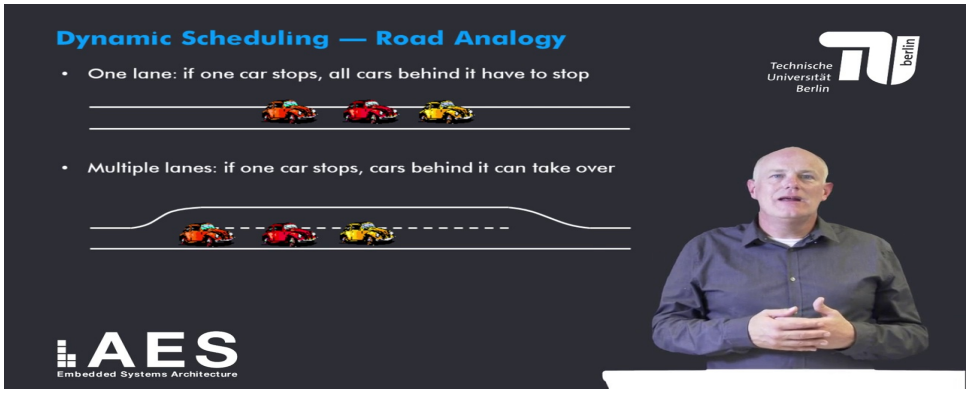
In this lesson I will describe Tomasulo algorithm and how it implements dynamic instruction scheduling.Before I describe the details of Tomasulo's algorithm, I will describe the basic idea behind dynamic scheduling. Here are three MIPS instructions that divide double and add double and subtract double.There is a data dependence between the divide and the ad double. Since the divide writes F0 and the ad reads F0AS indicated by the Red Arrow. The ad double stalls for 19 cycles in its decode stage because it is data dependent on the divide and because the ad double stalls in its decode state to subtract also needs to stall in its fetch tails.The main point of this example is that the subtract is stalled even though it does not depend on any previous instruction.

The basic idea of dynamic scheduling is to get rid of such stall cycles by allowing instructions to execute out of order.There are also some stall cycles caused by structural hazard. If there is only one floating point adder, but they are not the main focus of this lesson.

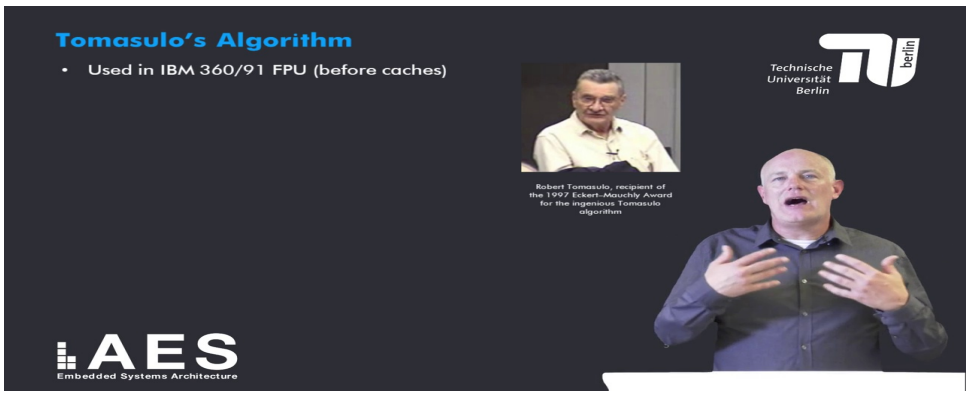


In an in order processor, it's like a road with one lane.

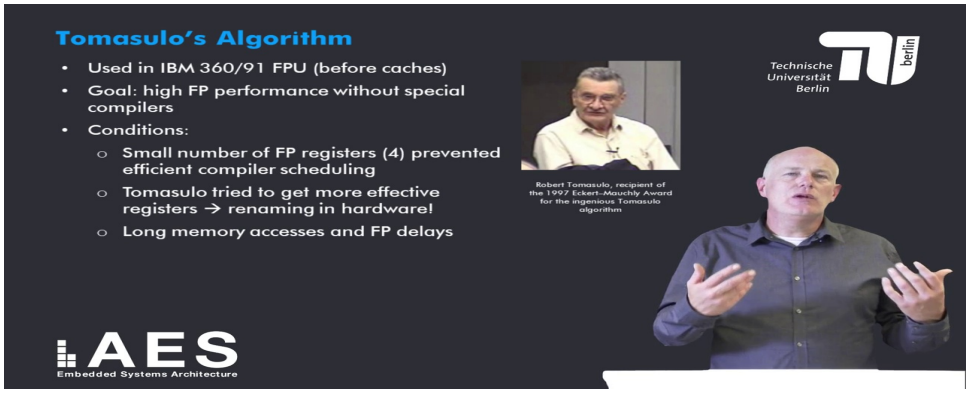


If one car stops all cars behind it have to stop also.An out of order processor, on the other hand is like a road with temporarily multiple lanes. If one car stops the cars behind, it can take over.I will now describe Tomasulo's algorithm. This is a picture of Robert Tomasulo, the man who invented the algorithm in 1997.

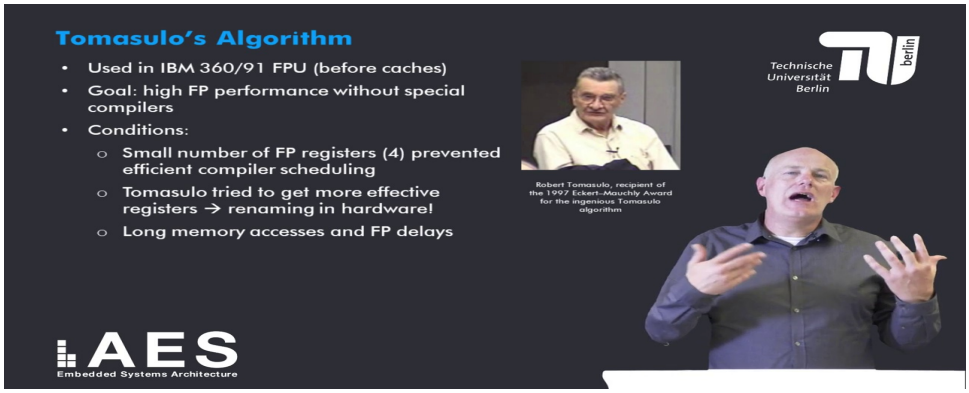
He received the Eccard Mauchly Award, which is an important award in computer science for its ingenious algorithm. If you have an algorithm named after you, you made it in this field. Unfortunately, he passed away in 2008.



The compiler developed for the entire 360 family of processes.The three 6091 processor had a number of limitations which made which made it difficult to achieve high floating point performance. First, it had only four double precision floating point registers, which limited the effectiveness of compiler scheduling. This motivated Thomas UNU to find a way to get more effective registers, and he did that by implementing register renaming in hardware.

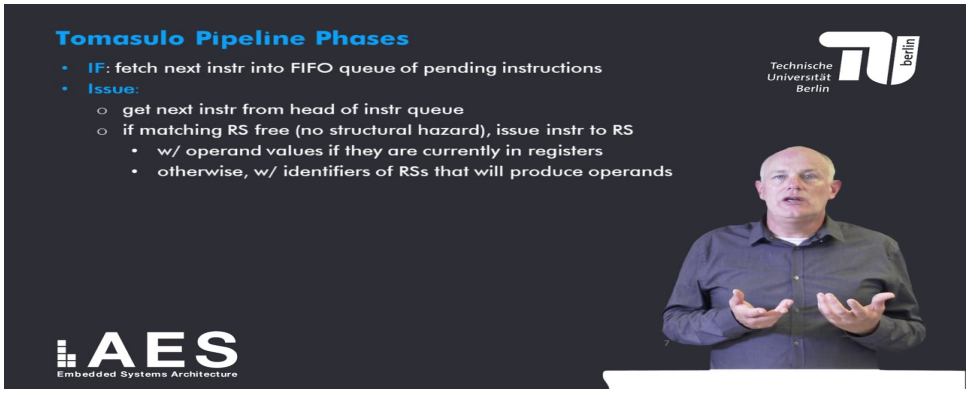


In addition, the IBM 360, ninety one had long memory accesses and long floating point delays.

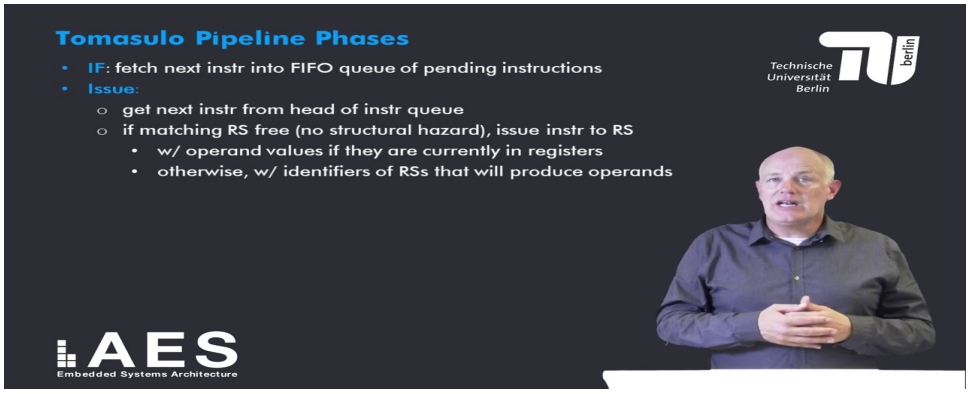


Now I describe some key hardware structures used in Tomasulo algorithm. Here's the same example as before. We want to allow the subtract double to proceed when the ad double is told.To be able to do so, we need to get the add double out of the way, so to speak. In other words, we need to buffer the add double somewhere and tomasulo's algorithm.Or broadcast on a bus called the common Data bus to all reservation stations.

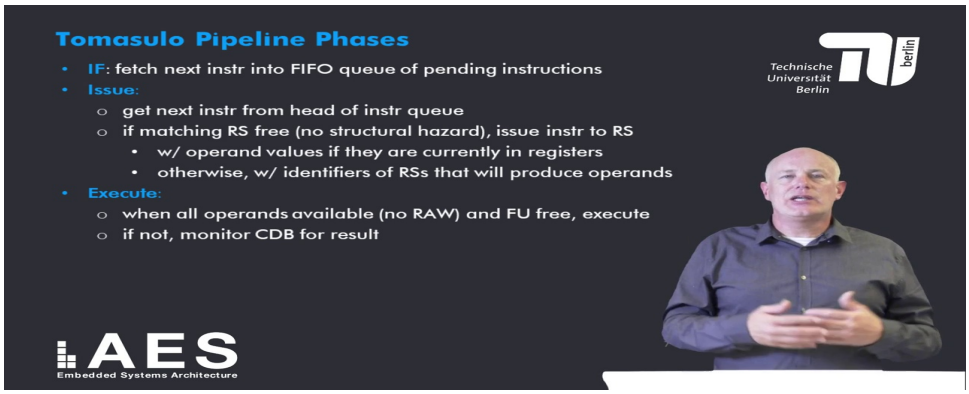
This brings me to the pipeline phases of Tomasulo algorithm. I purposely say phases rather than stages because pipeline phases may take several clock cycles, whereas the pipeline stage always takes a single cycle.The first phase is the instruction fetch phase. In this phase, the next instruction is fetched, but not in a single pipeline register, but into a 5 foot queue of pending instructions.The second stage is the issue phase. In this phase, the next instruction is retrieved from the instruction queue, and if there is a free matching reservation station, the instruction is placed in that reservation station.



In this phase, the next instruction is retrieved from the instruction queue, and if there is a free matching reservation station, the instruction is placed in that reservation station.

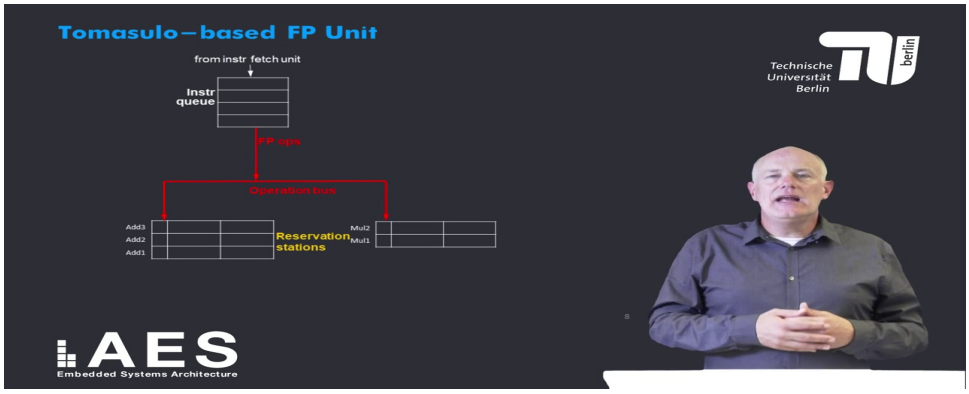


If there is no free matching reservation station, then we have a structural hazard in the execution has to store. Furthermore, if the values of the opera if the values of the operations of the instruction are currently available in register in register.Then the instruction is issued to the reservation station together with these values. If not, the operand values are currently being produced and the instruction issued together with identifiers of the reservation station that will produce these operands.The third phase in Tomasulo algorithm is the execute phase when all operands are available, meaning there is no raw hazard and a functional unit that can execute the instruction is available.

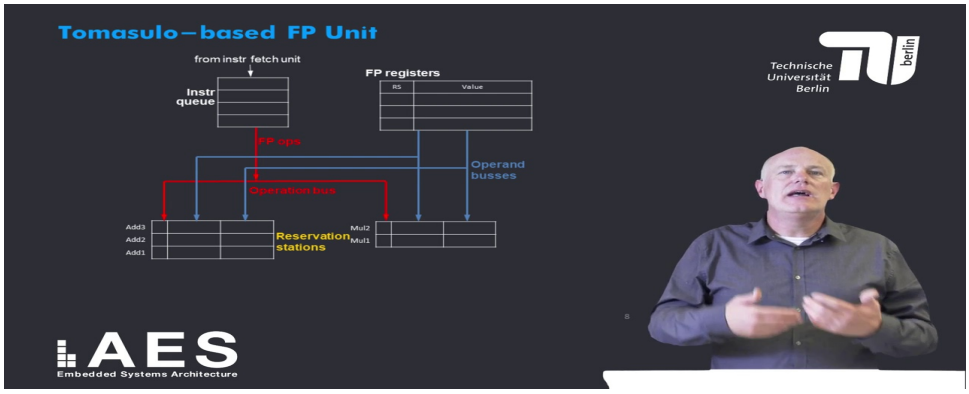


Then the instruction is executed if not, all operands are available, the reservation station must monitor the common data bus for the result to become available. First, the instruction is fetched by the instruction fetch unit into a FIFO queue of pending instructions.

Then in the second phase, the instructions at the head of the queue is issued to a matching reservation stations.

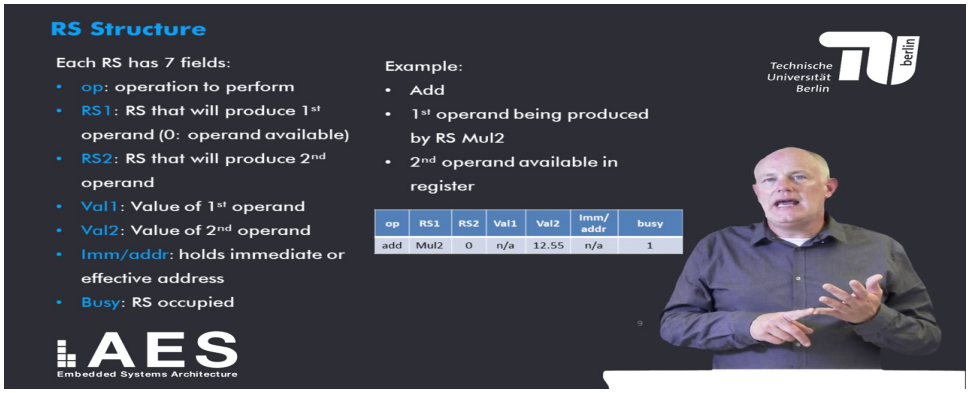


Here I have drawn in total 5 reservation stations, 3 floating point addition register reservation stations that can hold floating point add and subtract operations, and three floating point multiply reservation stations that can hold multiply instructions.As explained before, the instruction is issued together with its operand values if they are available in registers. Otherwise they are issued together with identifiers of the reservation stations that will produce these operant values in the next phase.

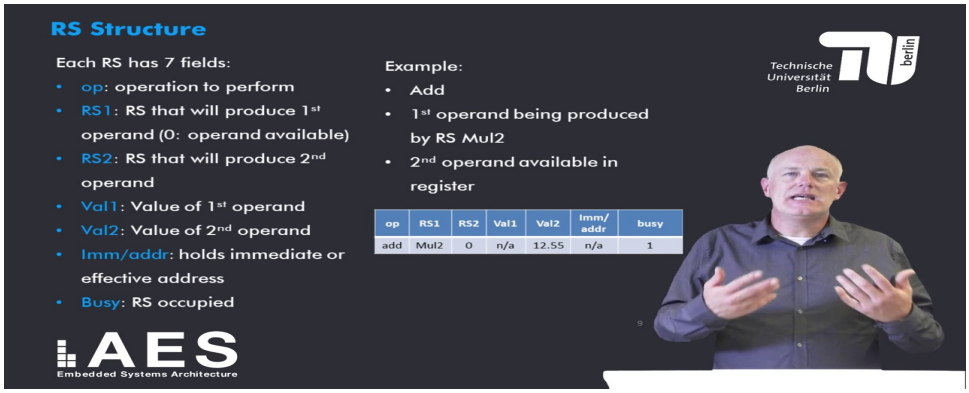


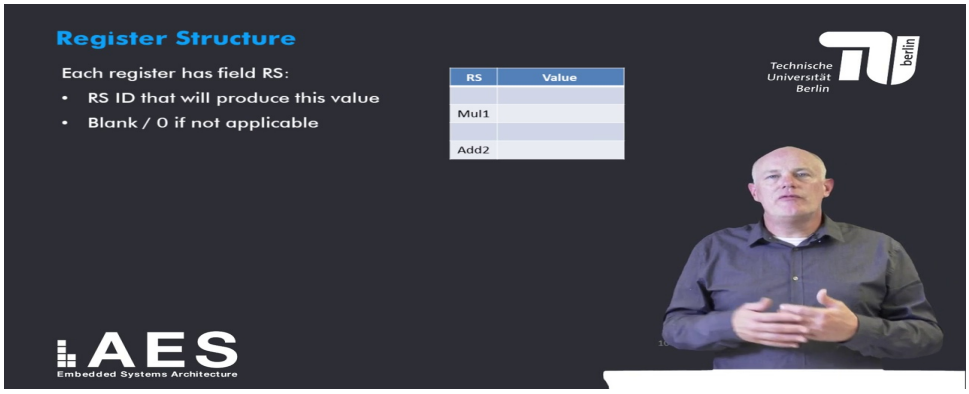
The first field is the operation to perform, such as add or subtract.The second field, or as one contains the identifier of the reservation station that will produce the first operand value. A value of 0 indicates that the source operand is already available.Similarly, the RS2 field contains the identifier of the Rs that will produce a second operant value.The next two fields are Val, one and Val 2. The value of the 1st and of the second operand. Note that either the errors field or the value field is valid, but not both. The six field contains an immediate or address. If the instruction has an immediate operand, it is stored here and after address calculation loads and store store the effective address in this field. An example to make this more concrete, suppose the instruction is an ad. The first operand is being produced by mall two and the 2nd operand value is available in the register.

Then the content of the reservation station is as shown here.

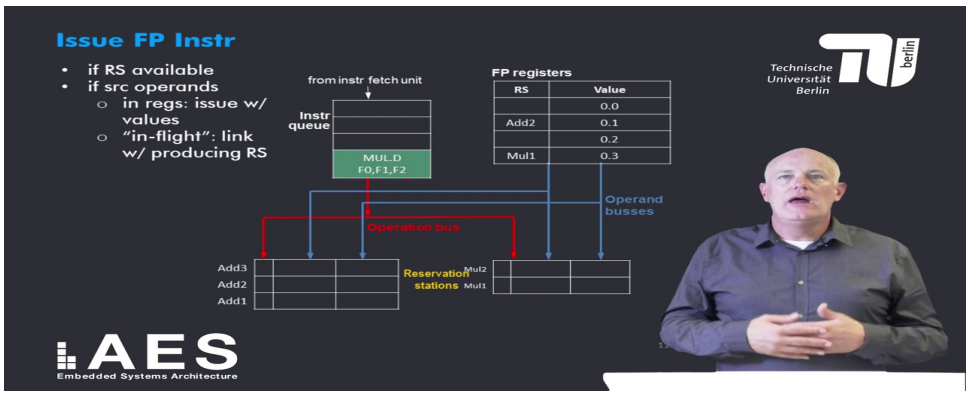


The operation is an ad or as one is mill two since middle 2 produces the first operand or is 2 is 0, indicating that the 2nd.Operant is available, value one is not applicable since it is being produced by mold two and value 2 is for example 12.

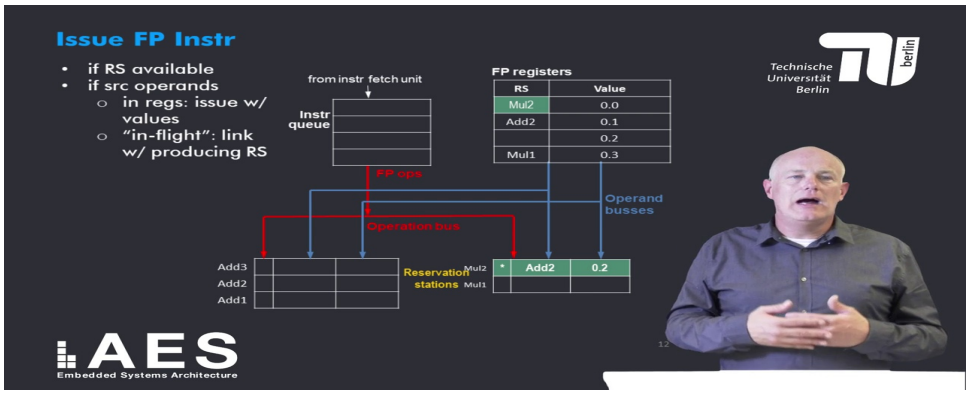


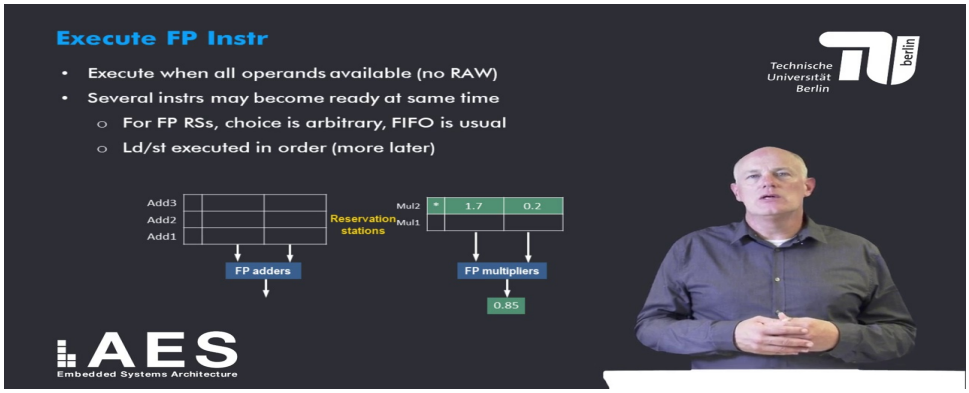


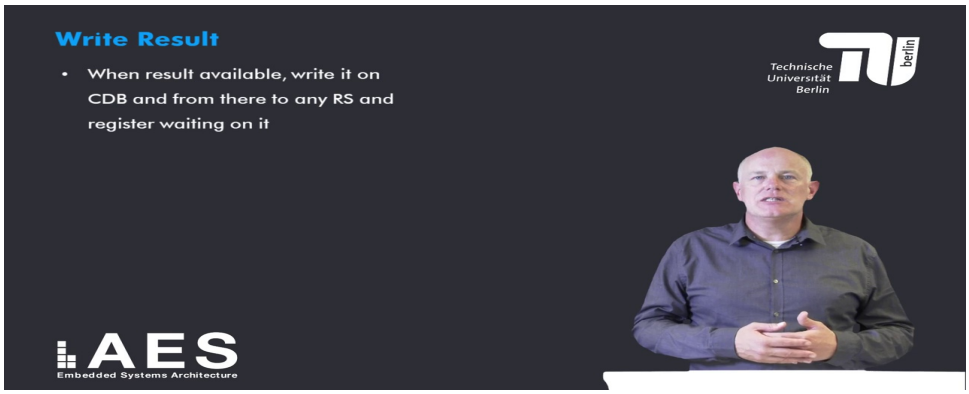
It is issued together with the operand value. On the other hand, if the source operand is currently in flight being produced, then the instruction is linked with the reservation station that will produce this apparent value.Suppose now that the instruction multiply double F0F1F2 is at the head of the instruction queue and should be issued as highlighted here in green, the register file shows that register F2 is available and contains the value of 0.2 register F1 on the other hand is being produced by reservation station ADD 2.

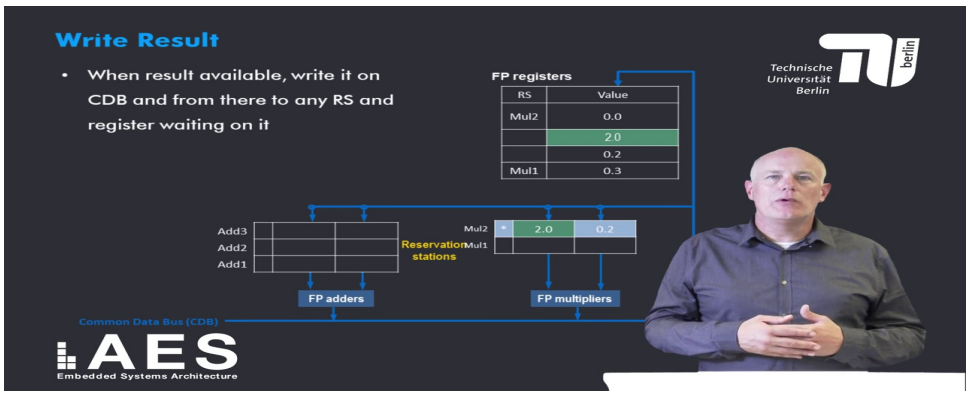


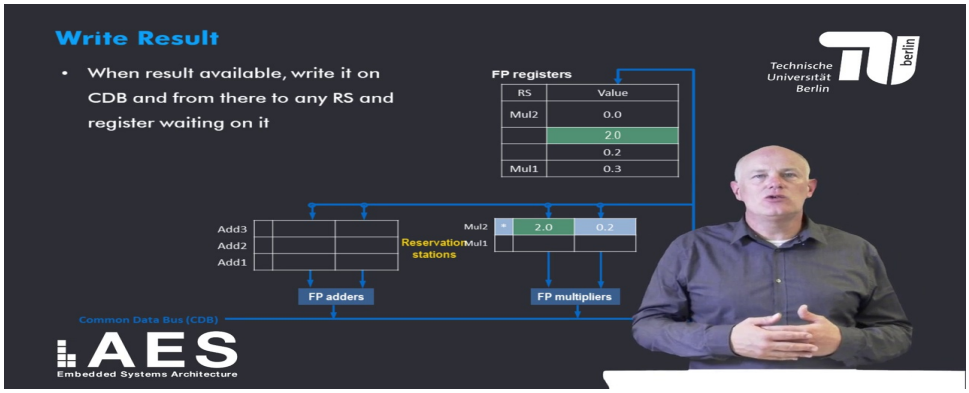
The multiply instruction will issue 2 reservation station mode 2 for example, highlighted in green.











Thanks for watching in the next lesson. I will show how several consecutive instructions are dynamically scheduled using TOMASULO'S algorithm TOMASULO'S algorithm. Hoping that you will fully understand it and never forget about it. Hope to see you back.