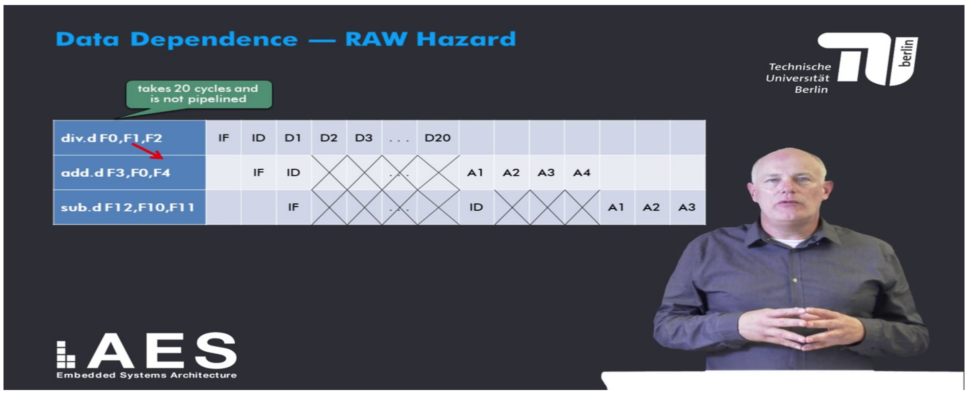


Data Dependence — RAW Hazard

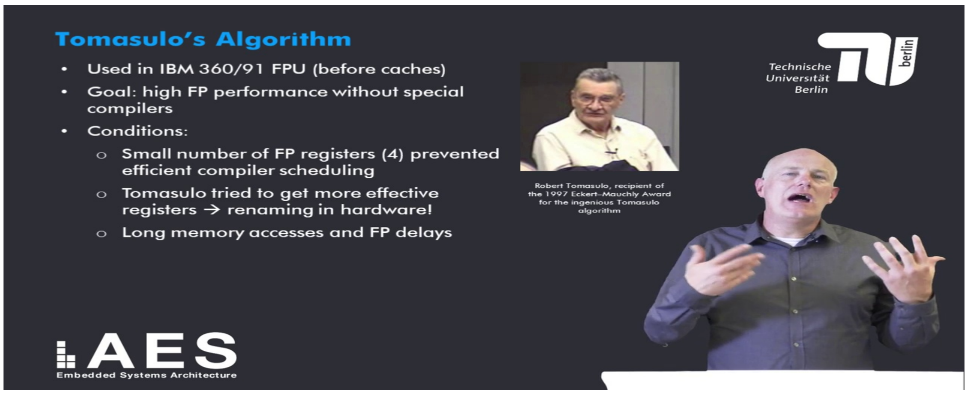
takes 20 cycles and is not pipelined

div.d F0,F1,F2	IF	ID	D1	D2	D3	. . .	D20							
add.d F3,F0,F4		IF	ID	✗	✗	✗ . . .	✗	A1	A2	A3	A4			
sub.d F12,F10,F11			IF	✗	✗	✗ . . .	✗	ID	✗	✗	✗	A1	A2	A3

Technische Universität Berlin

AES Embedded Systems Architecture

Welcome back to this course on I'll pant leg. 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.And the basic idea of dynamic scheduling is to get rid of these stall cycles by allowing instructions to execute out of order, as this pipeline diagram illustrates. In this execution the subtract does not stall, but it is executed while the ad double stalls.An analogy can be drawn between dynamic scheduling of instructions and roads and cars.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. 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.Tomasulo algorithm was implemented in the floating point unit of the IBM 360 91 processor. Even before caches were invented.



Tomasulo's Algorithm

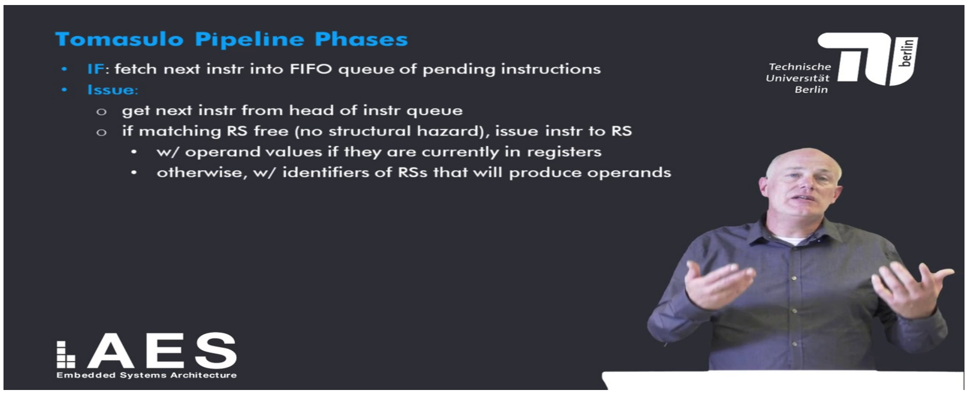
- Used in IBM 360/91 FPU (before caches)
- Goal: high FP performance without special compilers
- Conditions:
  - Small number of FP registers (4) prevented efficient compiler scheduling
  - Tomasulo tried to get more effective registers → renaming in hardware!
  - Long memory accesses and FP delays

Robert Tomasulo, recipient of the 1997 Eckert-Mauchly Award for the ingenious Tomasulo algorithm

Technische Universität Berlin

AES Embedded Systems Architecture

The IBM 360, ninety one, is a prehistoric processor available already in 1966. So you might ask the question, why should we study a 1966 processor. Well, the answer to that question is very simple. All modern desktop and server processors are based on Tomasulo algorithm. The Intel Core processors, the IBM power processes, etc. To understand their architecture and organization we need to study this algorithm.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.



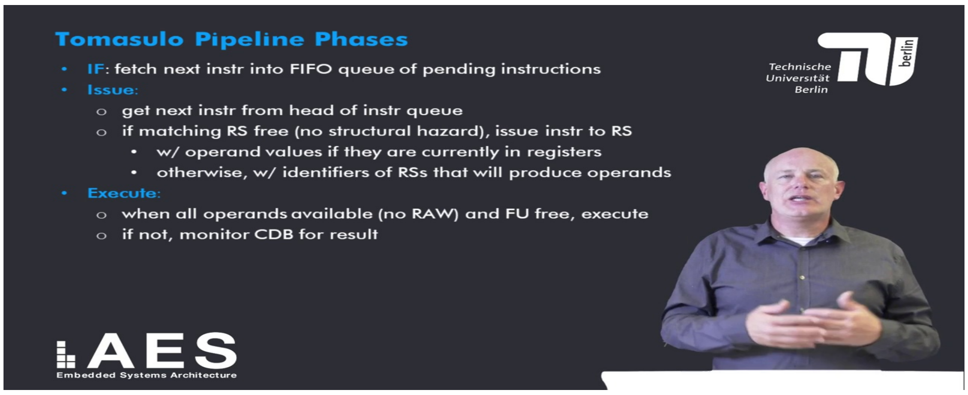
Tomasulo Pipeline Phases

- IF: fetch next instr into FIFO queue of pending instructions
- Issue:
  - get next instr from head of instr queue
  - if matching RS free (no structural hazard), issue instr to RS
    - w/ operand values if they are currently in registers
    - otherwise, w/ identifiers of RSs that will produce operands

Technische Universität Berlin

AES Embedded Systems Architecture

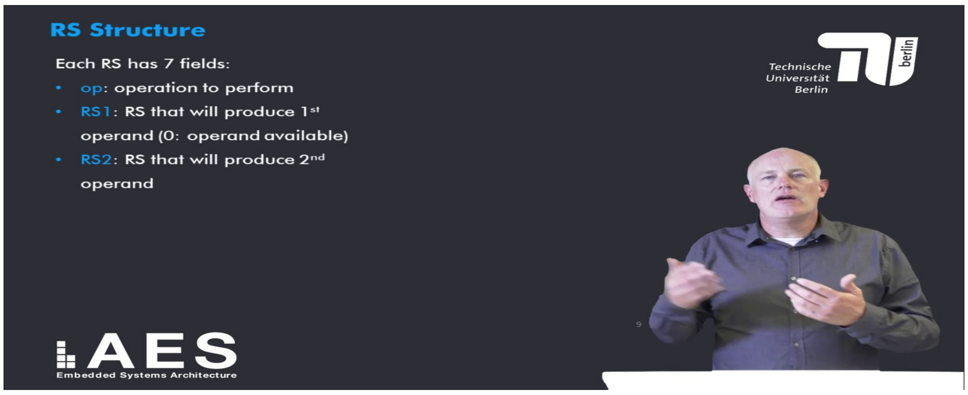
The third phase in Tomasulo algorithm is the execute phase when all operands are available, meaning there is no raw hazard and a functional unit that can execute the instruction is available.



Tomasulo Pipeline Phases

- IF: fetch next instr into FIFO queue of pending instructions
- Issue:
  - get next instr from head of instr queue
  - if matching RS free (no structural hazard), issue instr to RS
    - w/ operand values if they are currently in registers
    - otherwise, w/ identifiers of RSs that will produce operands
- Execute:
  - when all operands available (no RAW) and FU free, execute
  - if not, monitor CDB for result

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.The 4th and final phase is the right result phase. In this phase, the functional unit writes its result onto the common data bus to all reservation stations waiting for them and to the register file. In addition, the reservation station is mark free so that it can be reused for another instruction. 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 execute phase the instruction is executed. In this example, the reservation stations add one to add three are connected to one or several floating point errors, and the reservation stations. No one.And mold two are connected to one or more floating point multiplied multipliers.Finally, in the right result phase, the result is broadcast on the common data bus and written to every waiting reservation station as well as the register file.In order to keep track of the state of every instruction, each reservation station consists of seven fields.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.

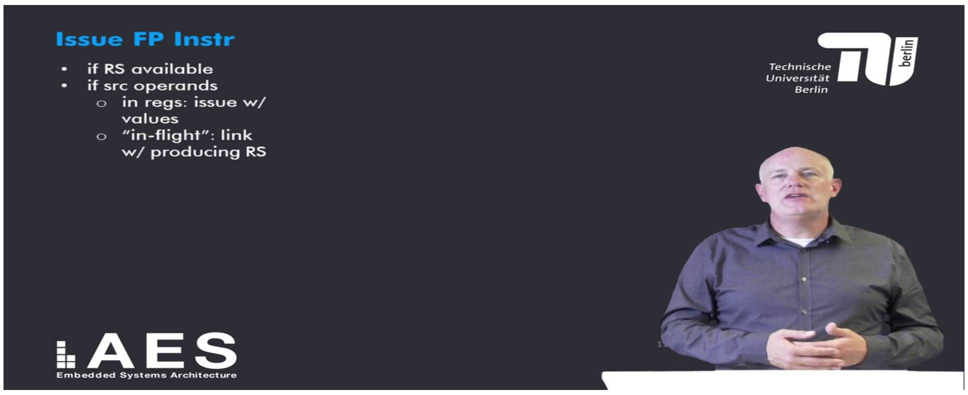


RS Structure

Each RS has 7 fields:

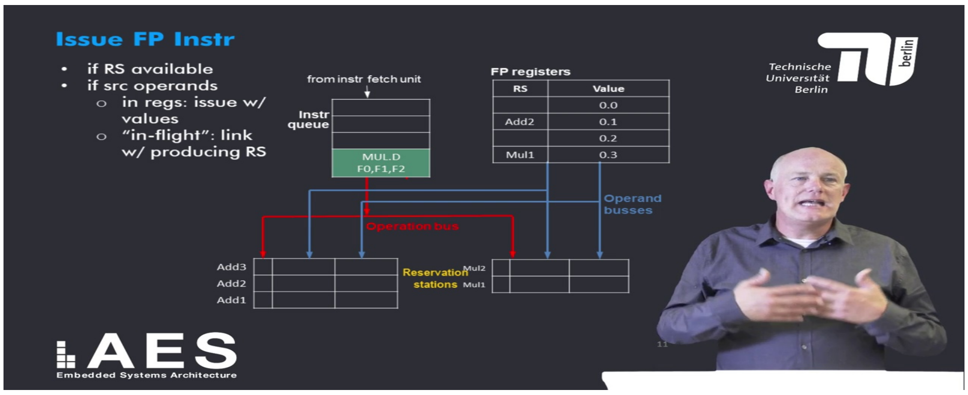
- op: operation to perform
- RS1: RS that will produce 1st operand (0: operand available)
- RS2: RS that will produce 2nd operand

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. 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.To track whether an operand value is available in a register or is currently being produced, each register also has an additional field Rs.This field contains the identifier of the reservation station that will produce a result that should be stored in this register. This field is blank or zero if no currently active instruction computes a result destined for this register.Here is an example vRS field of the 1st and 3rd register or Blank which means that they currently contain valid values. The second register is currently being produced by reservation station Mill One and the 4th Register is currently being produced by the reservation station at 2.I will now give an example of the issue phase of an instruction.



Issue FP Instr

- if RS available
- if src operands
  - in regs: issue w/ values
  - "in-flight": link w/ producing RS

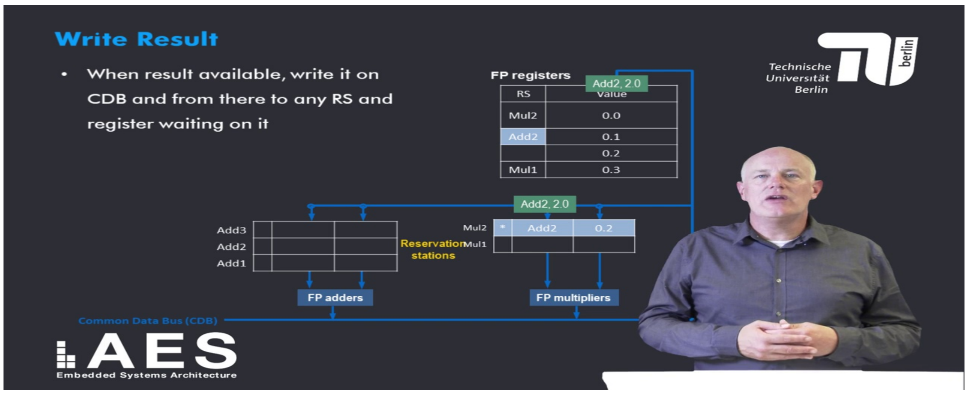


Issue FP Instr

- if RS available
- if src operands
  - in regs: issue w/ values
  - "in-flight": link w/ producing RS

RS	Value
	0.0
Add2	0.1
	0.2
Mul1	0.3

Add two and the 2nd operand is already available and equals 0.2.Notice also that because the multiple writes F0, the field of register F0 has been said to Mull two. Since this is the reservation station that will produce it.When all operands are available, the instruction in the reservation station can be executed. Notice that several instructions may become ready at the same time for floating point reservation station, which one is executed first is arbitrary, but first in first out. As usual, I will make sure that loads and stores are executed in order. However, I will explain later why.Here is a simple example of instruction execution. The multiply instruction is ready since both its operands are available, so it will execute on one of the floating point multipliers and produce the value 0.This picture illustrates this face. The instruction in reservation station ADD 2 highlighted in green is ready to execute. Furthermore, as highlighted in red reservation station mill two is waiting for this result and the result needs to be written to register F1.The subtracting instruction in reservation station at two has executed. It writes its result 2.0 onto the common data bus together with the identifier of the reservation station. Add to that has produced it. The result is broadcasted on the common data bus to all awaiting reservation stations and the register file.



Write Result

- When result available, write it on CDB and from there to any RS and register waiting on it

RS	Value
Mul2	0.0
Add2	0.1
	0.2
Mul1	0.3

And finally, the first operand of the instruction in Rs in reservation station mode two is now available and equals 2. Furthermore, register if one has been set to 2.0 and it's ours field has been cleared and we end up with this state.This completes this lesson. 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.