

## 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	





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 add double. Since the divide writes F0 and the add reads F0 as indicated by the Red Arrow. If there is only one floating point adder, but they are not the main focus of this lesson. So if instructions are executed in order, there will be many stall cycles as illustrated in this pipeline diagram. 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 add double stalls. An analogy can be drawn between dynamic scheduling of instructions and roads and cars.


## Dynamic Scheduling — Road Analogy


- One lane: if one car stops, all cars behind it have to stop






- Multiple lanes: if one car stops, 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 Eckert-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'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



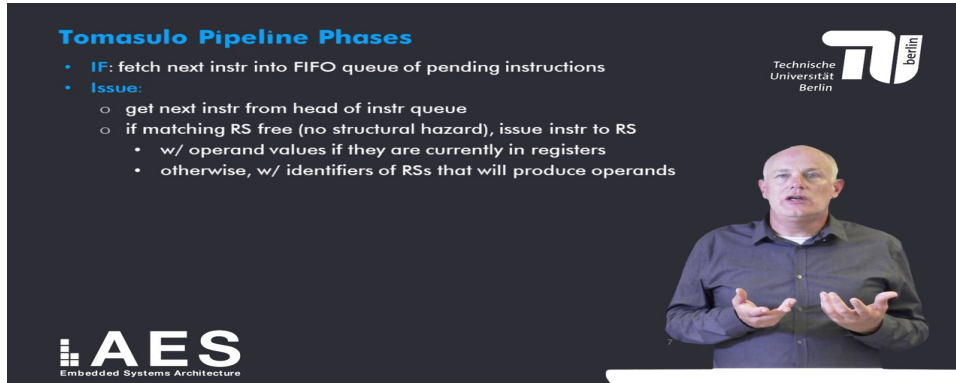


Robert Tomasulo, recipient of the 1997 Eckert-Mauchly Award for the ingenious Tomasulo algorithm

Here's the same example as before. We want to allow the subtract double to proceed when the add double is stalled. 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. These buffers are called reservation stations. 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. 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.



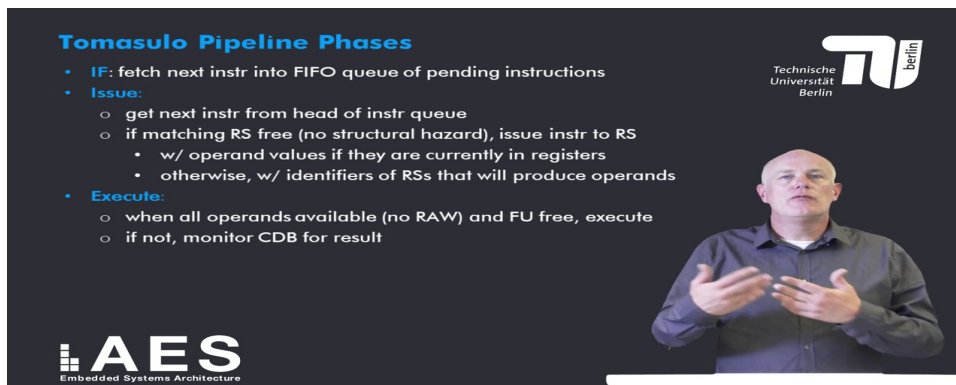
**Tomasulo Pipeline Phases**

- **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

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 operands 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.



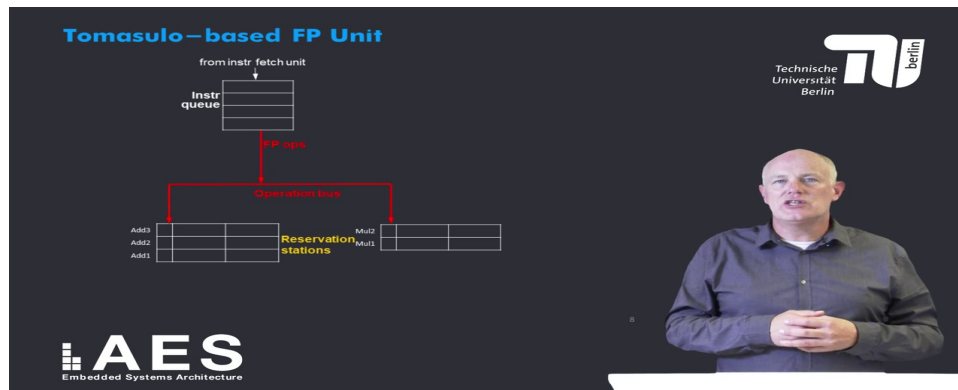
**Tomasulo Pipeline Phases**

- **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

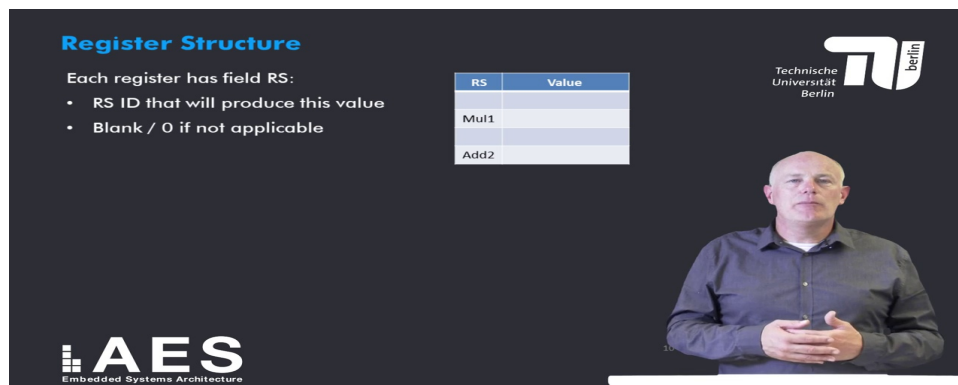
Technische Universität Berlin

**AES**  
Embedded Systems Architecture

The 4th and final phase is the write back 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 marked 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 station. 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 operand values in the next phase. 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 operand 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. An example to make this more concrete, suppose the instruction is an add. The first operand is being produced by mul2 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 add or as one is mul2 since mul2 produces the first operand or is 2 is 0, indicating that the 2nd operand is available, value one is not applicable since it is being produced by mul2 and value 2 is for example 12. The immediate address field is also not applicable, and the basic bit is set to indicate that the reservation station is occupied. 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 Mul1 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. As explained before, a floating point instruction is issued if a matching reservation station is available.



### Issue FP Instr

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

Technische Universität Berlin

**AES**  
Embedded Systems Architecture

### Write Result

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

Technische Universität Berlin

**AES**  
Embedded Systems Architecture

Add to that has produced it. The result is broadcasted on the common data bus to all awaiting reservation stations and the register file. And finally, the first operand of the instruction in Rs in reservation station mode two is now available and equals 2.

### Write Result

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

Technische Universität Berlin

**AES**  
Embedded Systems Architecture

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.

### Thank You For Your Attention

Prof. Dr. Ben Juurlink  
Embedded Systems Architecture  
Institute of Computer Engineering and Micro-Electronics  
School of Electrical Engineering and Computer Science  
TU Berlin

Technische Universität Berlin

**AES**  
Embedded Systems Architecture

