CS223 : Computer Architecture & Organization

Lecture 29 [19.04.2022]

Dynamic Scheduling to Explore ILP

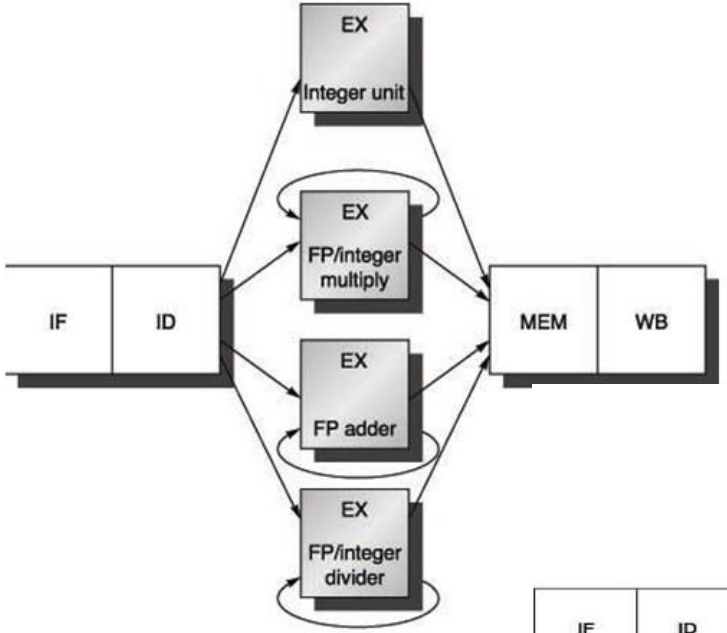
Dr. John Jose

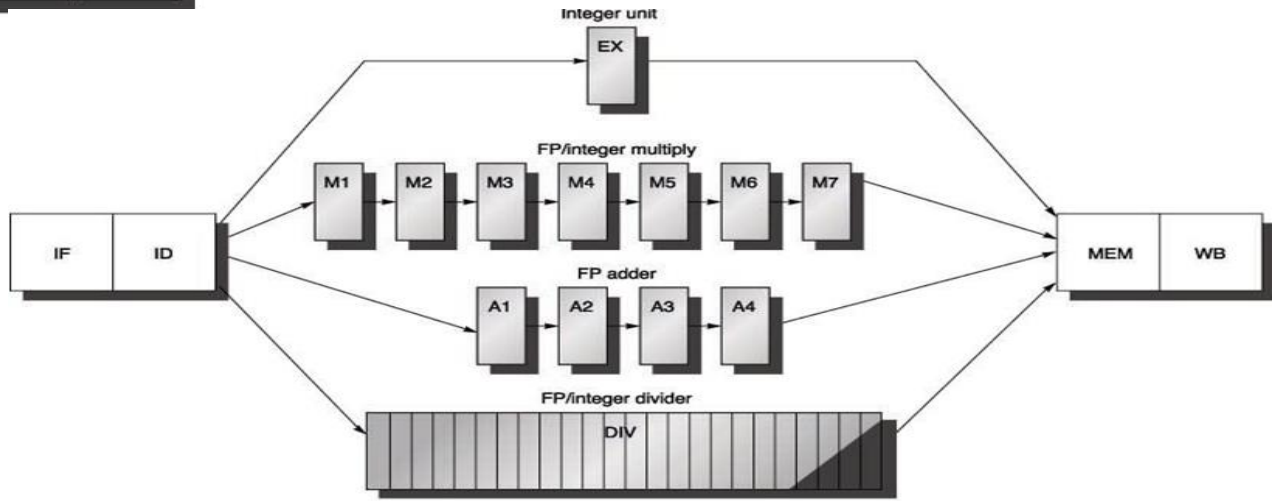
Associate Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

Multi-cycle Operations





Limitation of simple pipelining.

- Limitation of simple pipelining.
  - In-order instruction issue and execution.
  - Instructions are issued in program order.
  - If an instruction is stalled in the pipeline, no later instructions can proceed.
- If instruction j depends on a long-running instruction i, currently in execution in the pipeline, then all instructions after j must be stalled until i is finished and j can execute.

```
DIV.D     F0,F2,F4
ADD.D     F10,F0,F8
SUB.D     F12,F8,F14
```

How dynamic scheduling works ?

- **Rearrange execution order of instructions to reduce stalls while maintaining data flow**
- Separate the issue process into two parts:
  - checking for any structural hazards.
  - waiting for the absence of a data hazard.

```
DIV.D F0,F2,F4
ADD.D F10,F0,F8
SUB.D F12,F8,F14
```

- Use in-order instruction issue but we want an instruction to begin execution as soon as its data operands are available.
- out-of-order execution → out-of-order completion.
- OOO execution introduces WAR and WAW hazards

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**

Register Renaming

- WAR and WAW hazards – solved by register renaming

```
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
```

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

name dependence with F6

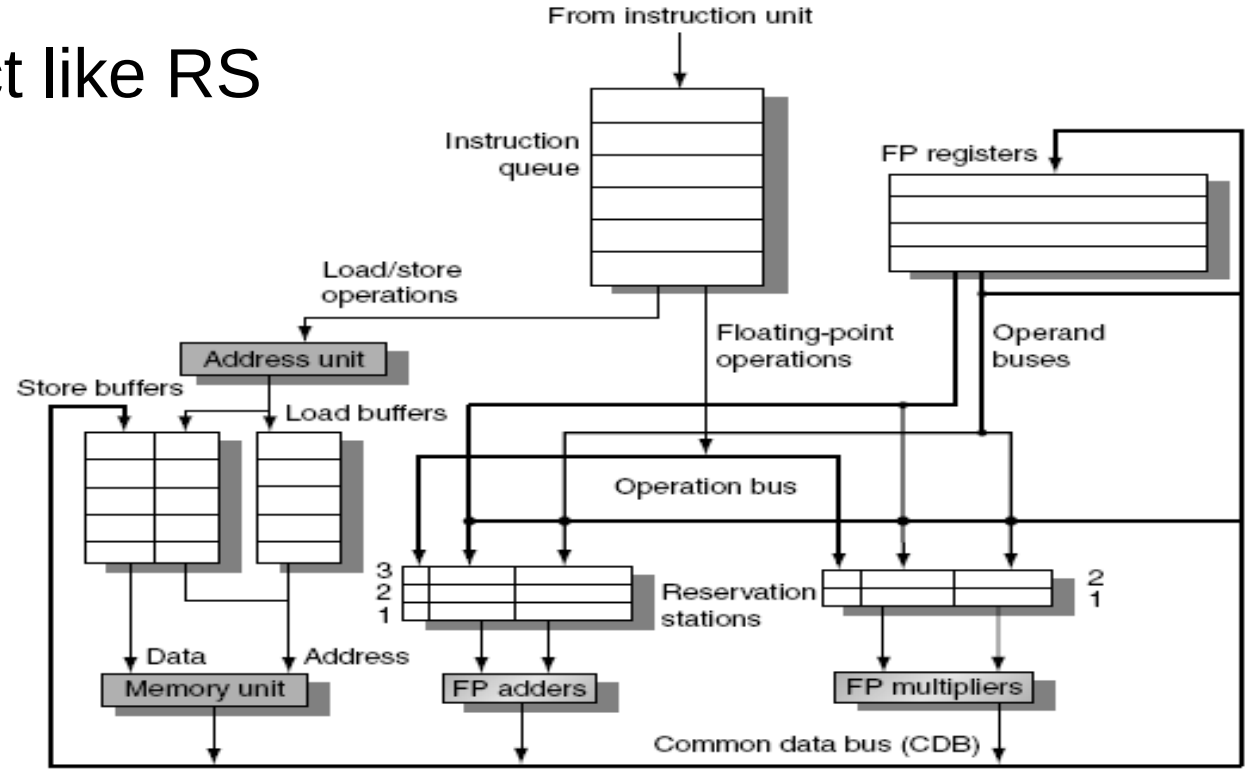
RAW hazard on T

Register Renaming

- Register renaming is done by reservation stations (RS)
- Each RS Contains:
  - The instruction (operation to be done)
  - Buffered operand values (when available)
  - Reservation station number of instruction providing the operand values
- RS fetches and buffers an operand as soon as it becomes available (not necessarily involving register file)
- Pending instructions designate the RS that will provide input
- Result values broadcast on common data bus (CDB)

Tomasulo's Algorithm

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



Reservation Stations

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

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.

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.

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.

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

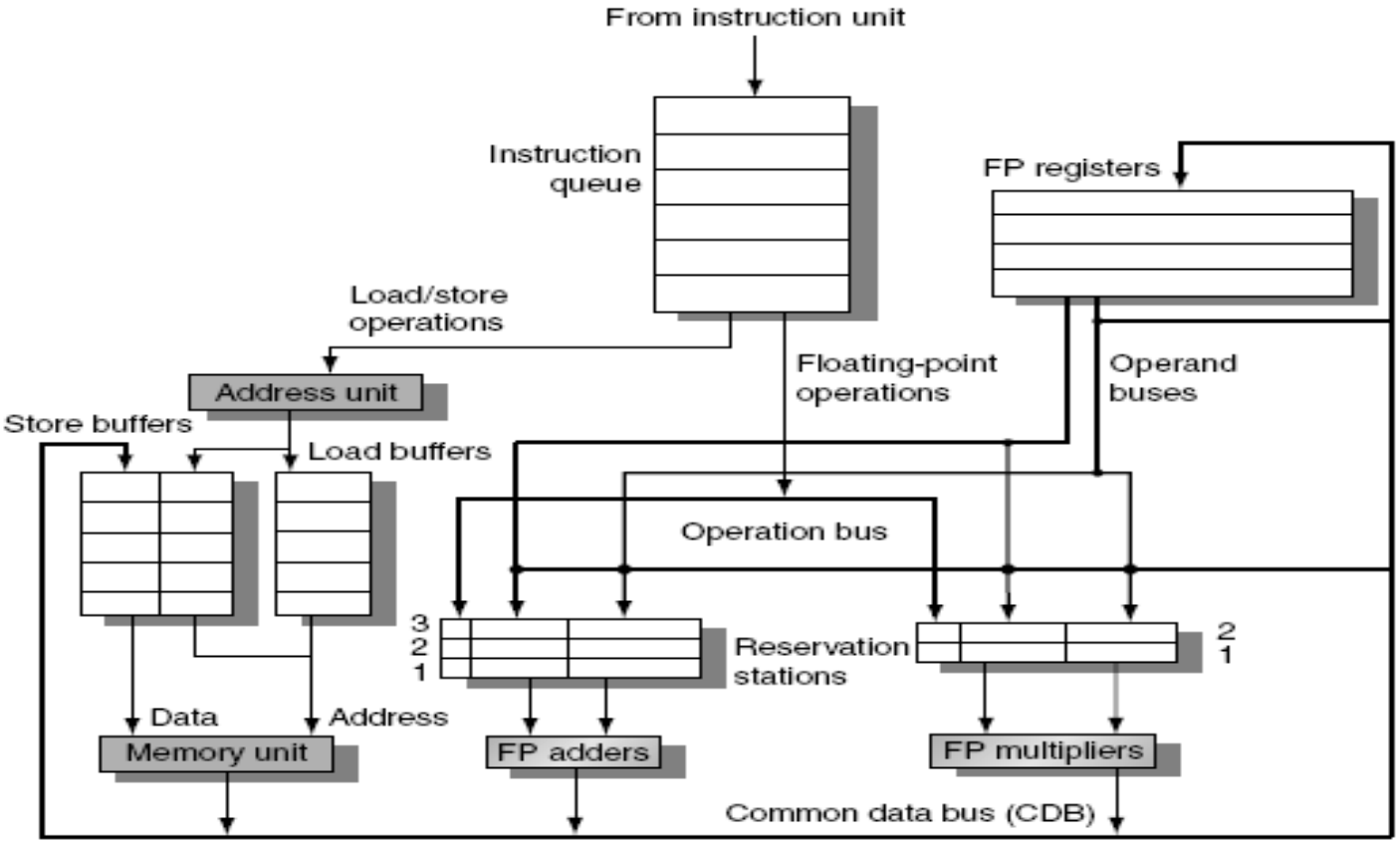
Dynamic Scheduling - Tomasulo

- **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

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

Tomasulo's Algorithm



Reference

- **Computer Architecture-A Quantitative Approach** (5th edition), John L. Hennessy, David A. Patterson, Morgan Kaufman.
- Chapter 3: **Instruction Level Parallelism and its Exploitation**
  - Section 3.4: **Overcoming Data Hazards with Dynamic Scheduling**
  - Section 3.5: **Dynamic Scheduling: Examples and the Algorithm**
- **NPTEL Video Links:** https://tinyurl.com/ybf48hef
  https://tinyurl.com/ybxgg83u

johnjose@iitg.ac.in
http://www.iitg.ac.in/johnjose/