# ELE 655
# Microprocessor System Design

## Section 2 – Instruction Level Parallelism

## Class 3 – Dynamic Scheduling

# ILP
## Instruction level Parallelism

- Static Scheduling
  - Structural Hazards
  - Data Hazards
    - Data dependencies
    - Name dependencies
  - Control Dependencies
  - → Reorder instructions, replace names, recalculated fixed values

# ILP
## Instruction level Parallelism

- Dynamic Scheduling
  - Hardware re-orders instructions
    - Maintain data flow
    - Maintain exception behavior
  - Enables:
    - Portable code
    - Unknowns at compile time
    - Unexpected delays – eg. cache delays
  - HW cost is significant

# ILP

## Instruction level Parallelism

- Simple Pipeline
  - In order instruction issue and execution
    - Stalls stop all subsequent instructions
  - If multiple execution units are available – all are stalled
  - Check for structural and data hazards in the ID stage
    - Issue when all was clear

# ILP

## Instruction level Parallelism

- Issue Process
  - Break issue into 2 parts
    - Check for structural hazards
    - Check for data hazards (wait on data hazards to clear)
  - Assuming no structural hazard – start execution as soon as data is available
  - Out of Order Execution
  - Out of Order Completion

# ILP
## Instruction level Parallelism

- Out of Order Execution
  - Creates the possibility of WAR and WAW hazards which did not originally exist

```
DIV.D    F0,F2,F4
ADD.D    F6,F0,F8
SUB.D    F8,F10,F14
MUL.D    F6,F10,F8
```

Anti-dependence

  - All data values are available for SUB.D and no structural hazard
    - If we move it forward we create a WAR Hazard for ADD.D

# ILP
## Instruction level Parallelism

- Out of Order Completion
  - Creates issues with exception handling
  - Exactly the exceptions that would arise in normal program order must arise
  - Instructions completed out of order temporarily leave the processor in a different state than if completed in order
    - Some instruction complete early
    - Some instructions complete late
  - Any exception during this temporary period will leave the processor in a different state
    - Imprecise Exception

# ILP

## Instruction level Parallelism

- Out of Order Execution
  - Split the ID pipe into Issue and Read Operands
  - Issue
    - decode instructions
    - Check for structural hazards
  - Read Operands
    - Wait for all data hazards to clear
    - Provide operands to execute stage

# ILP

## Instruction level Parallelism

- Out of Order Execution
  - All instructions are issued in order
    - In-order issue
  - Can be stalled, or bypass each other while waiting for operands to be available
    - Out of order execution start

# ILP
## Instruction level Parallelism

- Tomasulo's Algorithm
  - Dynamic Scheduling
  - Tracks operand availability to limit RAW hazards
  - Uses register renaming to limit WAR and WAW hazards

# ILP
## Instruction level Parallelism

- Tomasulo's Algorithm
  - WAR and WAW hazards

```
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
```

Anti-dependencies: ADD.D–SUB.D, S.D-MUL.D
Output dependencies: ADD.D-MUL.D
WAR – F8 in Add.d
WAR – F6 in S.D
WAW – MUL.D finishes before ADD.D

# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm
  - WAR and WAW hazards – register renaming

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

WAR – F8 in Add.d --- T retains data dependency, clears ADD
WAR – F6 in S.D --- S separates S.D and MUL.D
WAW – MUL.D finishes before ADD.D --- S S not needed after S.D

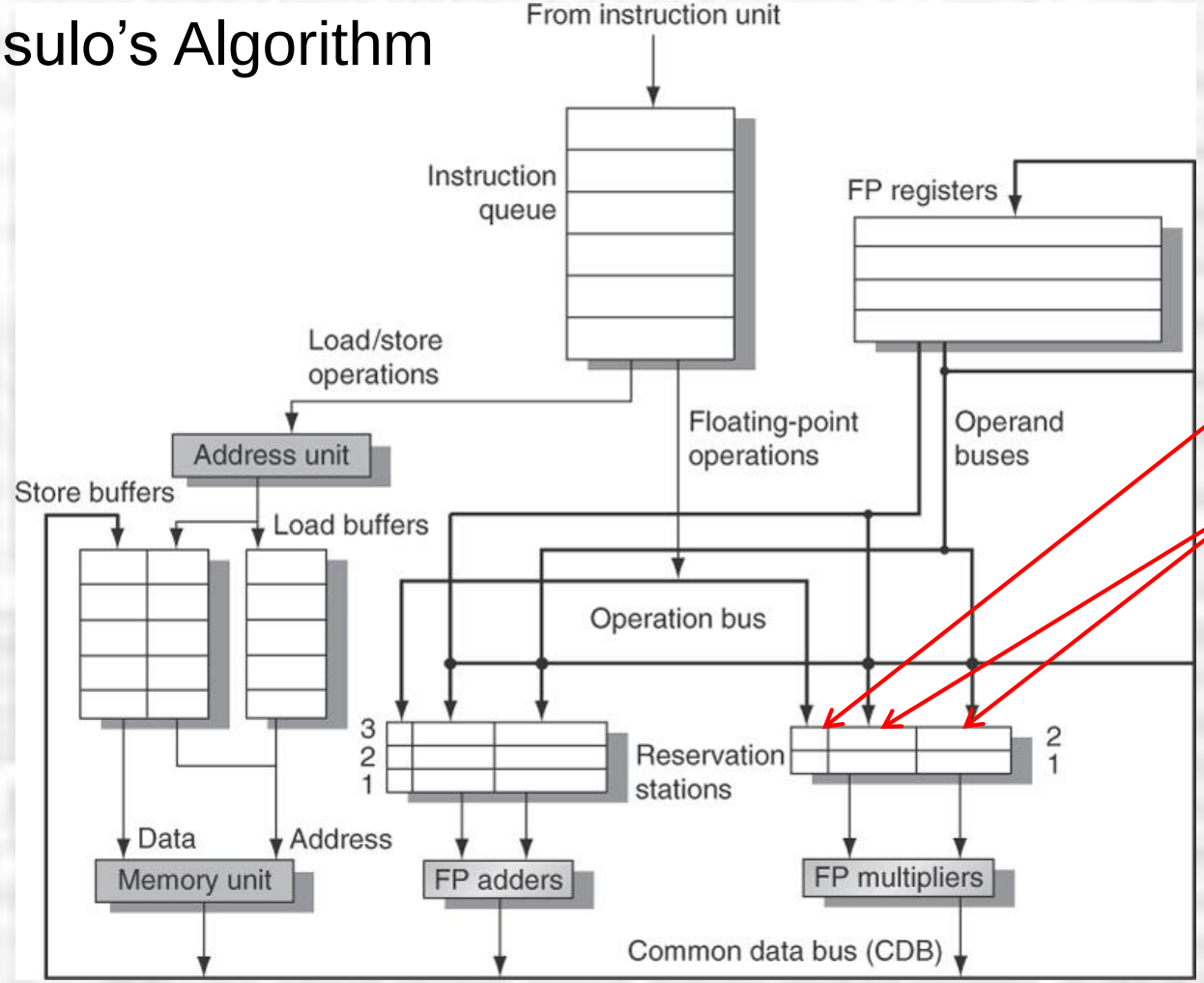
# ILP
## Instruction level Parallelism

- Tomasulo's Algorithm
  - Register Renaming done by Reservation Stations
  - Reservation Station
    - Buffers operands for each functional unit
    - Receives updates when available
    - Provides operands when ready
  - Shares a Common Data Bus with
    - Other functional units Reservation Stations
    - Functional unit outputs
    - Register File
    - Store Buffers

# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm



# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm
- Instruction Flow – Issue
  - Get next instruction from Queue
  - IF – the required execute unit has a free reservation station

    Issue the instruction (Dispatch)

    with either

    the value from the register if available

    or

    identify the functional unit that will create the value

    Else

    Stall – structural hazard

# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm
- Instruction Flow – Execute
  - Functional Units monitor the Reservation stations

    IF all operands are available in the RS – execute

    If an operand is not available – the RS monitors the Common Data Bus until the operand becomes available (completes execution elsewhere)

    Once available – load into reservation station

# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm
- Instruction Flow – Execute
  - Load/Stores are 2 step process
  - Calculate the effective address (waiting on the base register)
  - Place the effective address in the LD/ST buffers
  - Loads – execute when memory unit available
  - Stores – wait for the store data to be available
  - Loads/stores are kept in program order
    - via the effective address calculation

# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm
- Instruction Flow – Execute
  - Multiple instruction can be ready at the same time (same unit)
    - Some sort of prioritization scheme needed

# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm
- Instruction Flow – Write Result
  - When complete
    - Write result to the CDB
  - Any registers or reservation stations waiting for it will grab it from the CDB

# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm
  - Tags
    - Indicate the location from which the operand can be retrieved
      - Reservation station or load buffer
  - CDB + RS → replace our forwarding concept
  - 1 additional clock of latency is introduced
    - Getting from result → CDB → RS

# ILP
## Instruction level Parallelism

- Tomasulo's Algorithm

  - Reservation Station Fields

  - OP – operation to be performed by the functional unit
  - Qj,Qk – tag of reservation station producing the desired result
    a value of 0 indicated the result is already available
  - Vj,Vk – value of operand iff available
  - A – used in Load/Store – initially the immediate value is here
    the effective address is here when ready
  - Busy – this reservation station is in use

  - Register File Field
    - Qi – reservation station tag holding the value to store in the register
    - Note registers are loaded and stored via the load store buffers

# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm
  - Fig 3.7 and 3.8
  - Load – 1 cycle
  - Add – 2 cycles
  - Mult – 6 cycles
  - Divide – 12 cycles

# ILP

## Instruction level Parallelism

- Tomasulo's Algorithm
  - Fig 3.10
  - Adds ignored
  - Branches assumed tacked
  - Load – 1 cycle
  - Add – 2 cycles
  - Mult – 6 cycles
  - Divide – 12 cycles