Module-3
CSEN 3104
Lecture 26
17/09/2019

Dr. Debranjan Sarkar

Superscalar Architecture

Data Dependences

- Show the example program and the dependence graph
- Register R1 is loaded by I1 and its content is used by I2, so we have flow dependence: I1 → I2
- The result in register R4 after executing I4 may affect the operand register R4 used by I3, we have anti-dependence between I3 and I4
- Since both I5 and I6 modify the register R6, and R6 supplies an operand for I6, we have both flow and output dependence between I5 and I6
- These are shown in the dependence graph
- To schedule instructions through one or more pipelines, these data dependences must not be violated. Otherwise erroneous results may be produced

Pipeline stalling

- Pipeline can be stalled due to
  - data dependences, or
  - by resource conflicts among instructions
    - already in the pipeline, or
    - about to enter the pipeline
- This problem may seriously lower pipeline utilization
- This problem exists in both scalar and superscalar processors
- It is more serious in a superscalar pipeline
- Proper scheduling avoids pipeline stalling

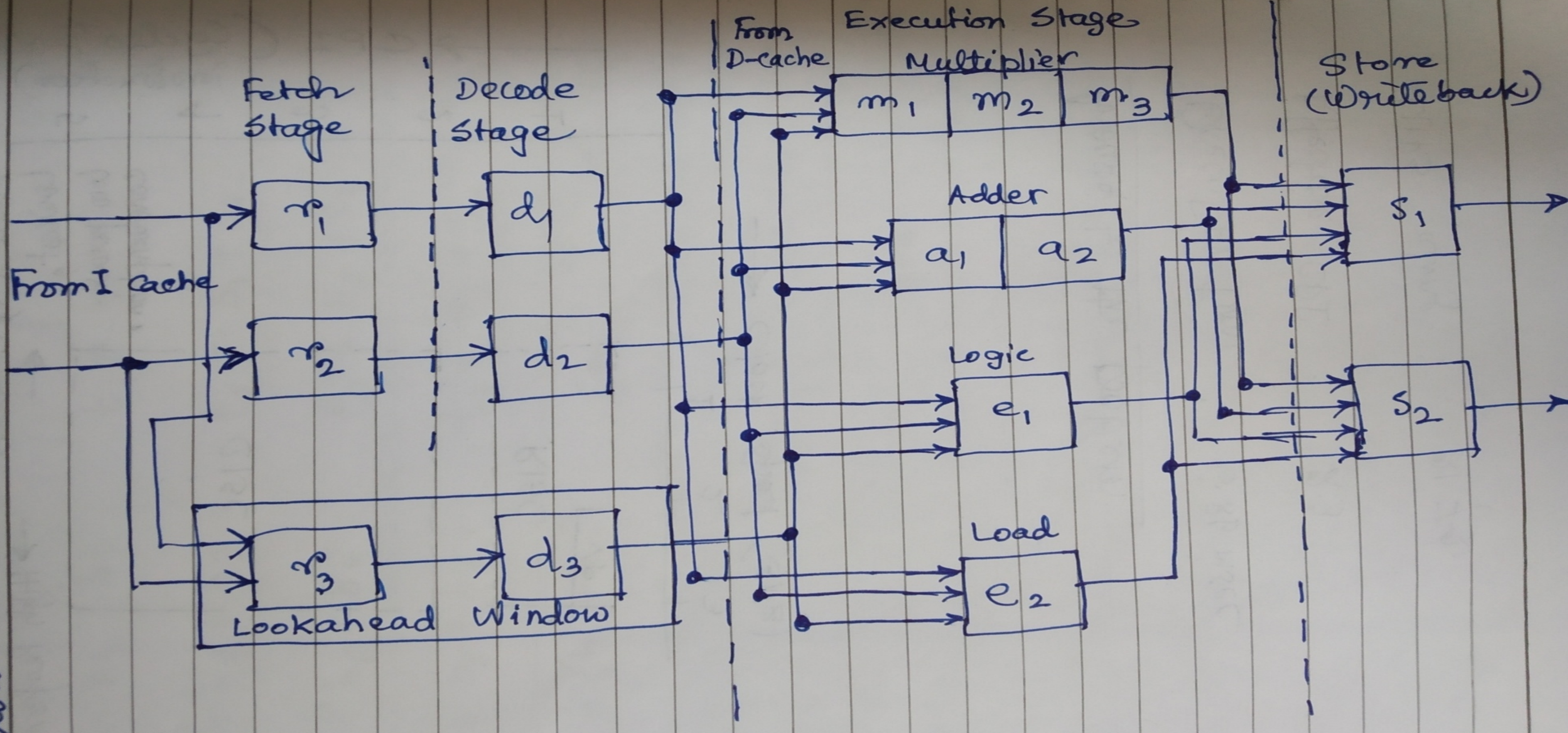
Pipeline Stalling

- Show figures to give examples
- Figure (a) shows the case of no data dependences
  - All pipeline stages are utilized without idling
- Figure (b) shows the case of flow dependences (I1 → I2)
  - I2 in the second pipeline must wait for 2 cycles, before entering the execution stages
  - This delay may also pass to the next instruction I4 entering the pipeline
- Figure (c) shows the effect of branching (instruction I2)
  - A delay slot of 4 cycles results from a branch taken by I2 at cycle 5
  - Therefore, both pipelines must be flushed before I3 and I4 can enter the pipelines from cycle 6
  - Here delayed branch and other amending actions are not taken

Pipeline Stalling

- Figure (d) shows the case of no resource conflicts
  - All pipeline stages are utilized without idling
- Figure (e) shows a combined problem of both resource conflict and data dependences (I1 and I2 use the same functional unit, and I2 → I4)
  - I2 must be scheduled 1 cycle behind because the two pipeline stages (e1 and e2) of the same functional unit must be used by I1 and I2 in an overlapped fashion
  - For the same reason, I3 is also delayed by 1 cycle
  - I4 is delayed by 2 cycles due to the flow dependence on I2

A dual-pipeline, superscalar processor with 4 functional units and a lookahead window.



Superscalar pipeline scheduling

- Instruction issue and completion policies are critical to superscalar processor performance
- Three scheduling policies:
  - In-order issue (Program order not violated. Easy to implement but may not yield the optimal performance)
    - In-order completion
    - Out-of-order completion
  - Out-of-order issue (Program order violated. Purpose is to improve performance)
    - Usually ends up with Out-of-order completion

Superscalar pipeline scheduling (In-order issue)

- Six instructions are issued in program order
- As I1→I2 (flow dependent), I2 has to wait one cycle to use the data loaded in by I1
- I3 is delayed by 1 cycle because the same adder a1 is being used by I2
- I6 has to wait for the result of I5 before it can enter the multiplier stages
- To have in-order completion, I5 has to wait for 2 cycles to come out of pipeline 1
- In total, 9 cycles are needed and 5 idles are observed
- If out-of-order completion is allowed, I5 is allowed to complete ahead of I3 and I4, which are totally independent of I5
- The total execution time does not improve but the pipeline utilization rate does
- Only 3 idle cycles are observed
- Show the figures of the scheduling policy In-order Issue

Thank you