Lecture 05

Dependences and Pipelining

이재진
서울대학교 데이터사이언스대학원
서울대학교 공과대학 컴퓨터공학부
http://aces.snu.ac.kr/~jlee

Processor, CPU, and Core

- Poorly defined terms in these days
    - A processor or CPU typically refers to the physical chip (package)
- Core
    - A hardware unit that fetches instructions and executes them
    - Contains hardware components involved in executing instructions
        - ALU, FPU, (private) L1/L2 caches, etc.
- Processor or CPU
    - The combination of one or more cores with shared resources between cores and some supporting hardware
    - A processor sometimes refers to a processor that contains a single core or the core itself depending on the context

Uncore

- A term used by Intel
- Hardware components that are not in the core
    - QPI controllers, last level caches (e.g., L3 cache), on-chip memory controllers, etc.

Techniques to Improve a Single Core

- As VLSI technology improves, more room becomes available on a chip
- Two major techniques
  - Instruction pipelines
  - On-chip caches

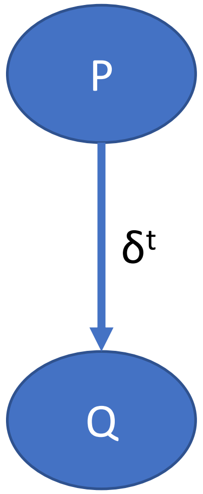
Dependences

- An ordering relationship between two computations
  - Any ordering of execution that obeys all dependences will produce the same result as that of the original program
- Data dependences
  - Flow (true) dependence
  - Anti dependence
  - Output dependence
  - Input dependence (not really a dependence)
- Control dependences

Flow Dependence

- True dependence
- Instruction P writes a memory location that instruction Q later reads (read after write)

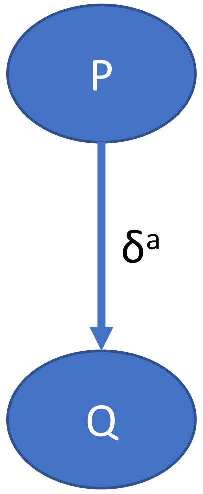
```
P:  a = …;
. . .
Q:  … = a;
```



Anti Dependence

- False dependence (can be removed)
- Instruction P reads a memory location that instruction Q later writes (write after read)

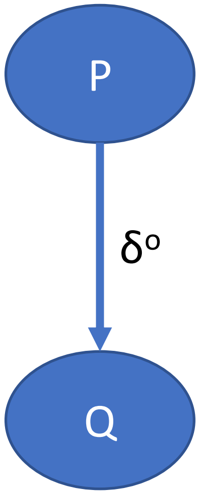
```
P:  … = a;
...
Q:  a = …;
```



Output Dependence

- False dependence (can be removed)
- Instruction P writes a memory location that instruction Q later writes (write after write)

```
P: a = …;
...
Q: a = …;
```



Input Dependence

- Not really a dependence
    - For caches
- Instruction P reads a memory location that instruction Q later reads (read after read)

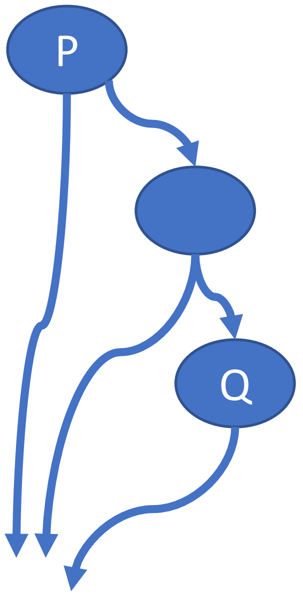
```
P: … = a;
...
Q: … = a;

//cache hit for the access to a
```

Control Dependences

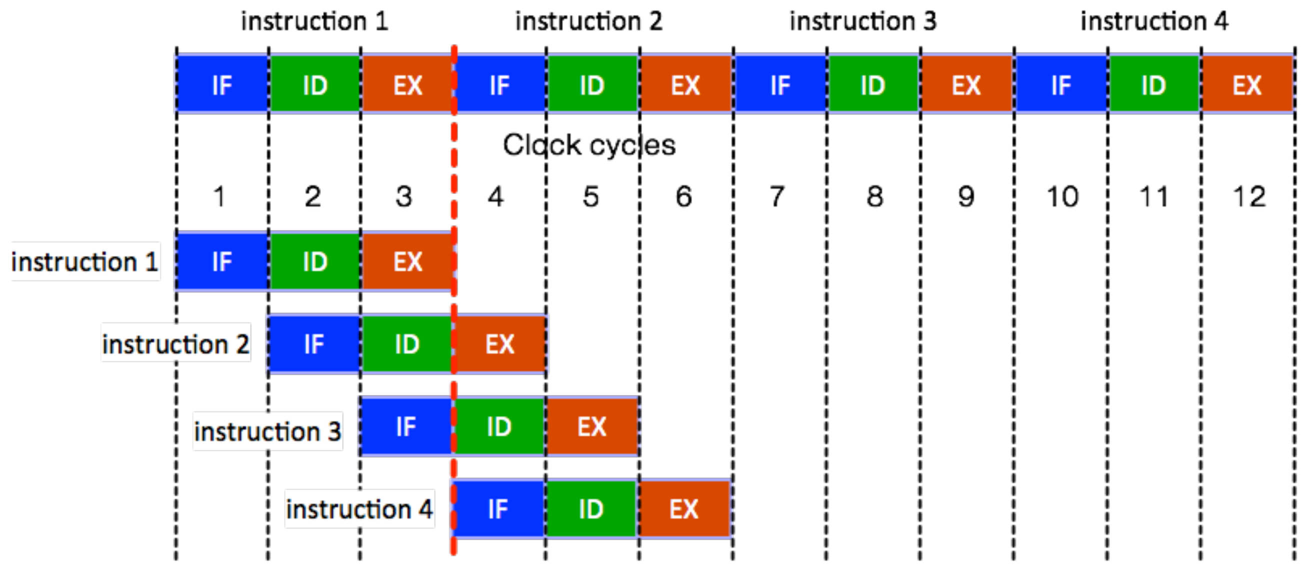
- An instruction Q has a control dependence on a preceding instruction P if the outcome of P determines whether Q should be executed or not
- An instruction Q is said to be control dependent on another statement P if and only if
  - There exists a path from P to Q such that every instruction I ≠ P within the path will be followed by Q in each possible path to the end of the program, and
  - P will not necessarily be followed by Q, i.e. there is an execution path from P to the end of the program that does not go through Q

```
P: if (x > 3)
Q:     a = …;
R: a = …;
```



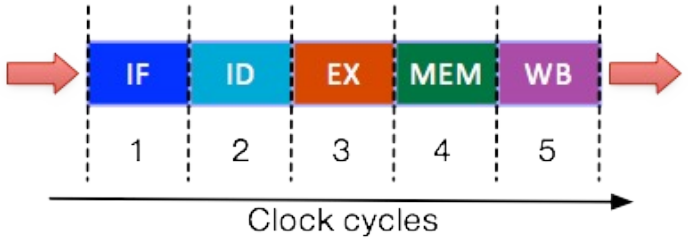
Instruction Pipeline

- Pipelining is a hardware technique that increases instruction throughput (e.g., the number of instructions executed in a CPU clock cycle)
- Assume
    - Each instruction takes 3 cycles to complete
    - Each pipeline stage takes a single cycle



Pipeline Hazards

- Typical five-stage pipeline
  - IF: instruction fetch
  - ID: instruction decode and register fetch
  - EX: execute
  - MEM: memory access
  - WB: register write back



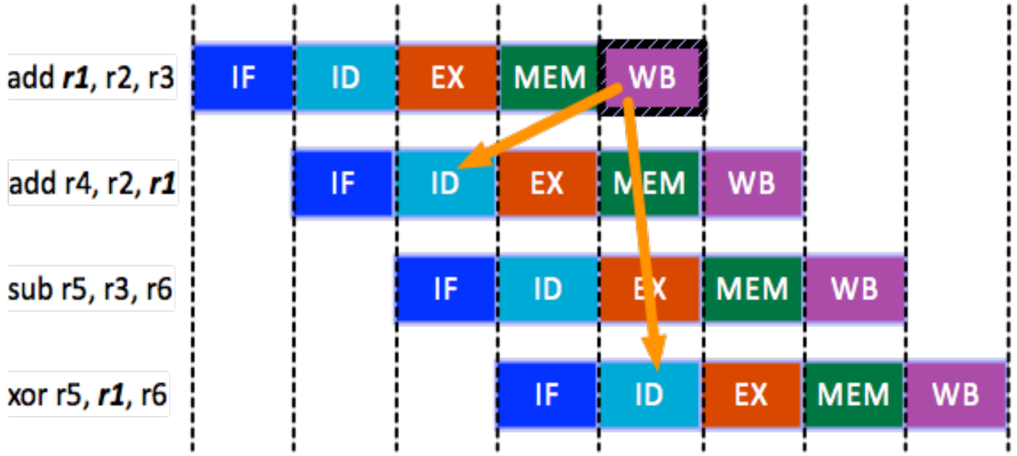
Pipeline Hazards (cont'd)

- Occur when the next instruction does not execute in the following clock cycle

- Data hazards
- Structural hazards
- Control hazards

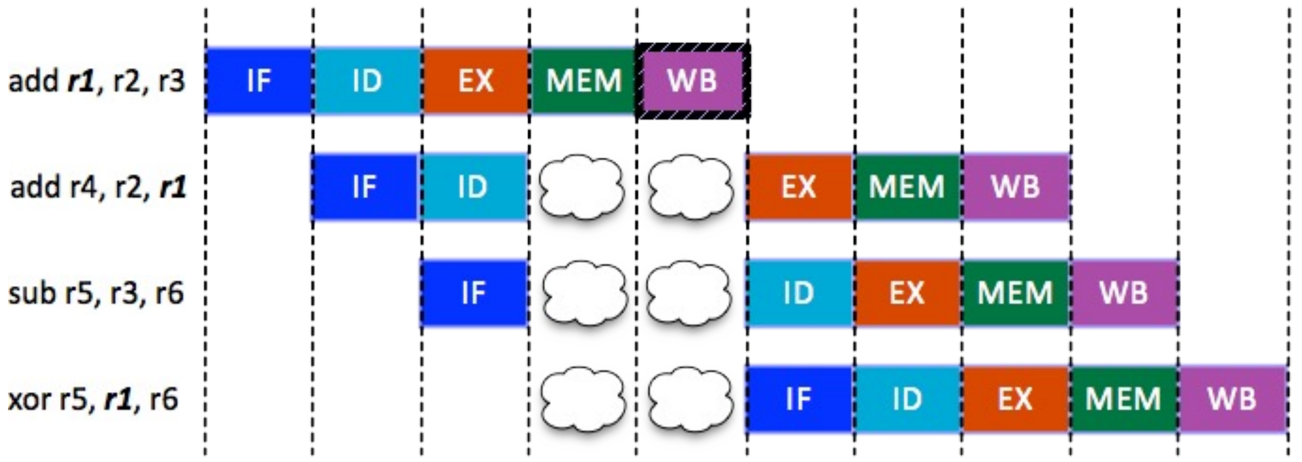
Data Hazards

- Occur when a result is needed in the pipeline before it is available



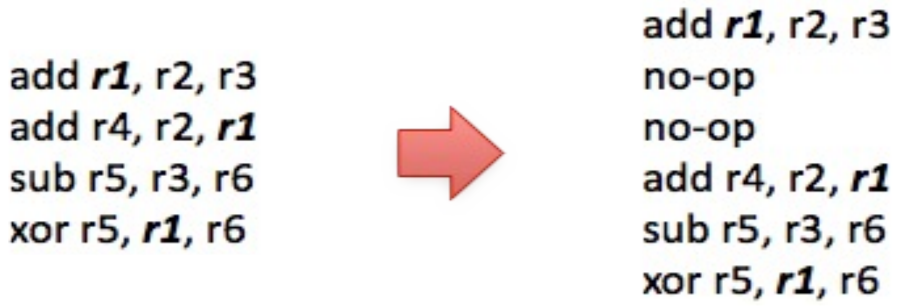
Resolving Data Hazards

- Stalling the pipeline
    - A bubble represents a delay in execution of an instruction in the pipeline to resolve a hazard



Resolving Data Hazards (cont'd)

- Software
    - Insert independent instructions (or no-ops)



```
add r1, r2, r3
add r4, r2, r1
sub r5, r3, r6
xor r5, r1, r6
```

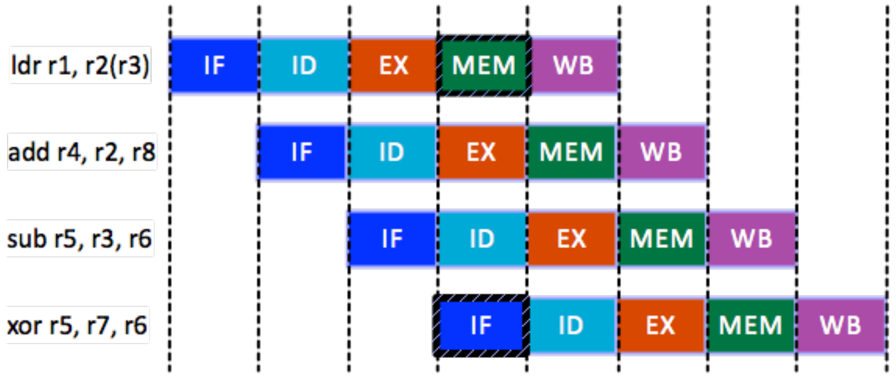
```
add r1, r2, r3
no-op
no-op
add r4, r2, r1
sub r5, r3, r6
xor r5, r1, r6
```

Resolving Data Hazards (cont'd)

- Hardware
    - Stalling the pipeline (i.e., insert bubbles)
    - Transparent register file
        - Resolves data hazard at the WB stage
        - If the register file is asked to read and write the same register in the same cycle, the register file allows the written data forwarded to the stage in which the data is needed
            - First half cycle: the register is written with the data
            - Send half cycle: the register is read into the pipeline (in the ID stage)
    - Data forwarding
        - Feeding output data into a previous stage of the pipeline
        - Resolves data hazard at the EX and MEM stages

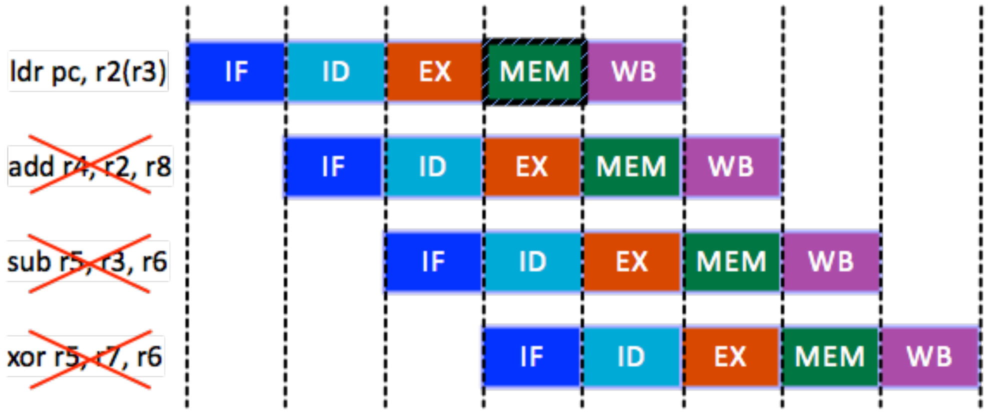
Structural Hazards

- Occur when a part of the processor's hardware is needed by two or more instructions at the same time
- Due to resource conflicts
  - A single memory unit that is accessed both in the IF stage and the MEM stage
- Resolving structural hazards
  - Stalling the pipeline



Control Hazards

- Occur because of branches
  - The pipeline does not know the branch target until the instruction reaches the MEM stage
    - Assume PC is set in the MEM stage
  - The pipeline continues fetching instructions sequentially
    - These fetched instructions cannot be allowed to execute because the programmer has diverted control

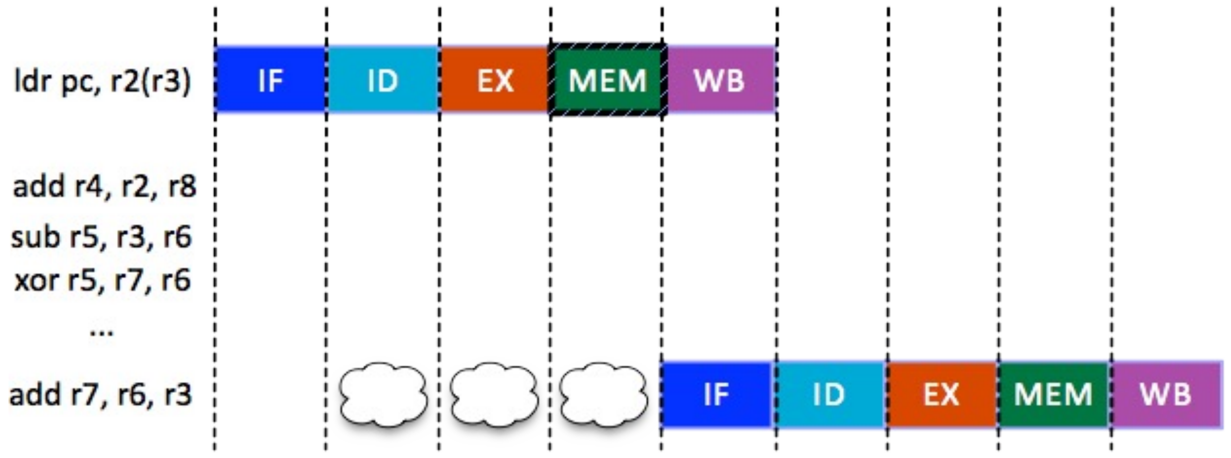


Resolving Control Hazards

- Software
  - Delayed branch
    - Perform instruction scheduling into branch delay slots with instructions before the branch instructions
      - From the target address (when it is known to be taken)
      - From fall through (when it is known to be not-taken)
- Programs written for a pipelined processor deliberately avoid branching to avoid performance degradation

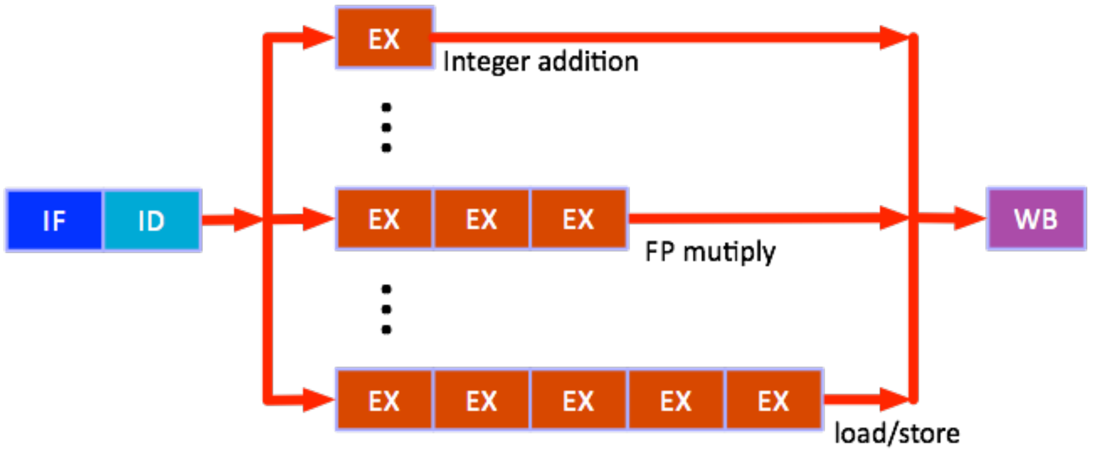
Resolving Control Hazards (cont'd)

- Hardware
  - Stalling the pipeline until the branch target is known
  - Branch predictors (complicated)



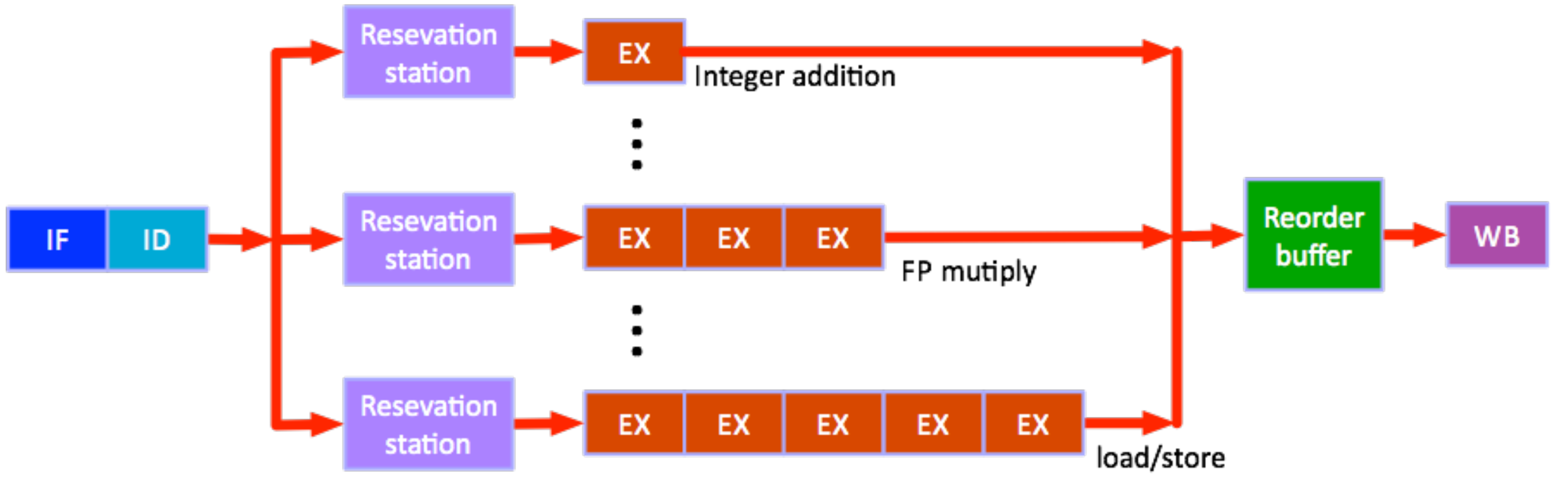
In-order Execution

- Steps
  - Fetch and decode the next instruction
  - If its operands are available, the instruction is dispatched to the appropriate functional unit
  - Otherwise, the processor stalls until they are available
  - The result is written back to register file



Out of Order Execution (OoO)

- Dynamic instruction scheduling by hardware
- Steps
    - Fetch and decode the next instruction
    - Issue it to the appropriate reservations station
    - It waits in the reservation station until its operands are available
    - The instruction is dispatched to the appropriate functional unit and executes
    - The execution completes, and the result is queued in the reorder buffer in the commit unit
    - The commit unit write results to registers and memory in program fetch order



Tomasulo's Algorithm

- IBM 360/91
- Single issue
- In-order issue
- Out-of-order dispatch
- Out-of-order execution
- In-order commit

Issuing and Dispatching an Instruction

- Once the instruction has passed the ID stage, we say that the instruction is issued

- When all operands are available, we say that the instruction is dispatched from a reservation station to the execution (functional) unit

Retirement (Graduation)

- An instruction retires when the reorder buffer slot of an instruction is freed either
  - because the instruction commits (the result is made permanent) or
  - because the instruction is removed (without making permanent changes)

Superscalar Processors

- Dynamically issue multiple instructions in each clock cycle
    - A typical superscalar processor fetches and decodes several instructions at a time
    - In-order or out-of-order issue
- Exploit he potential of ILP (Instruction Level Parallelism)
    - How many instructions can be executed simultaneously?
    - Limited amount of ILP in an application

VLIW Processors

- Use a long instruction word that contains a fixed number of instructions that are fetched, decoded, issued, and executed synchronously

- Static instruction scheduling by a compiler