Topic V2D

Instruction-Level Parallelism

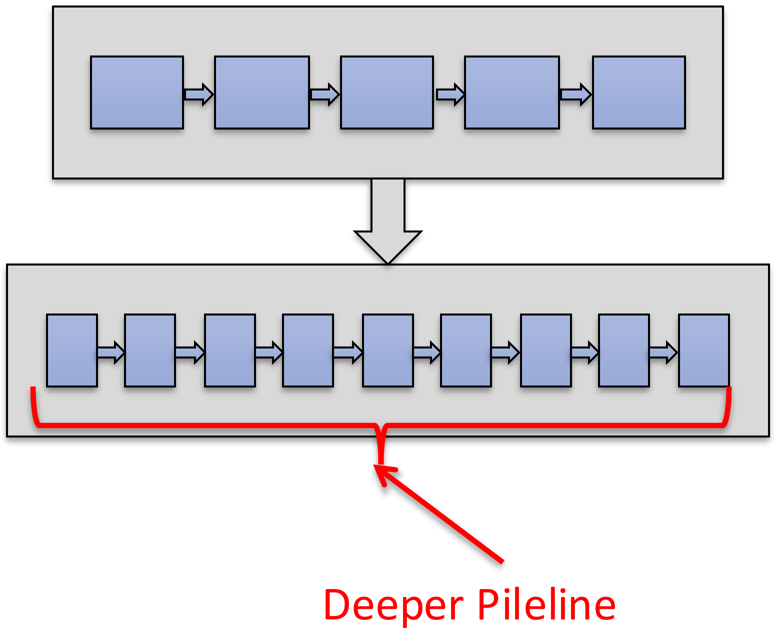
Reading: (Section 4.10)

Instruction-Level Parallelism (ILP)

Pipelining: executing multiple instructions in parallel

To increase ILP

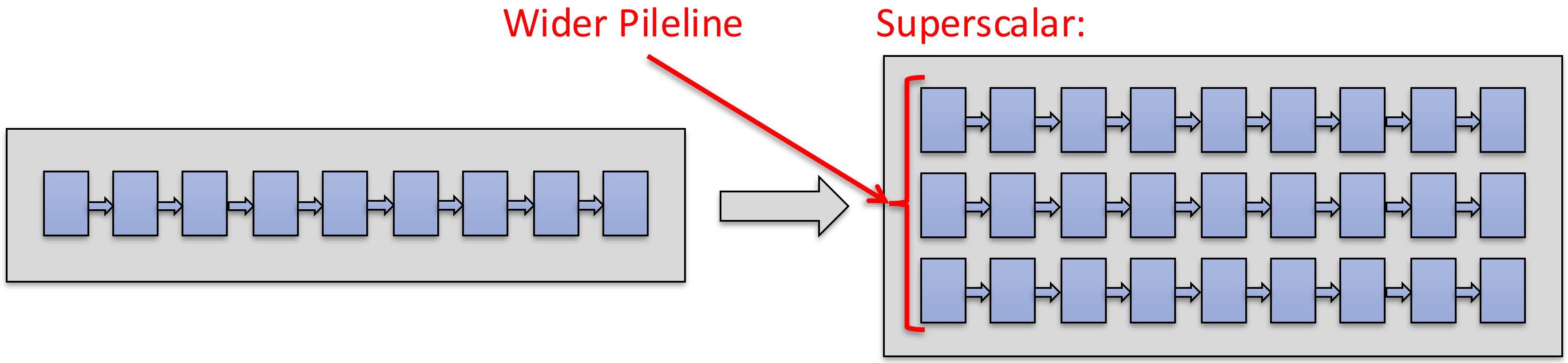
- Deeper pipeline
  - Less work per stage ⇒ shorter clock cycle



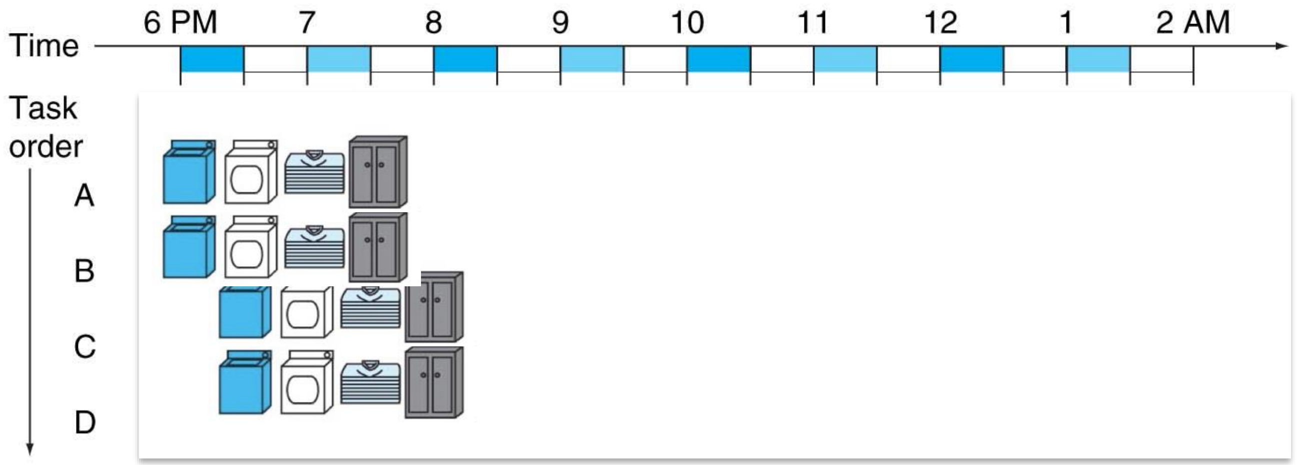
Multiple Issue

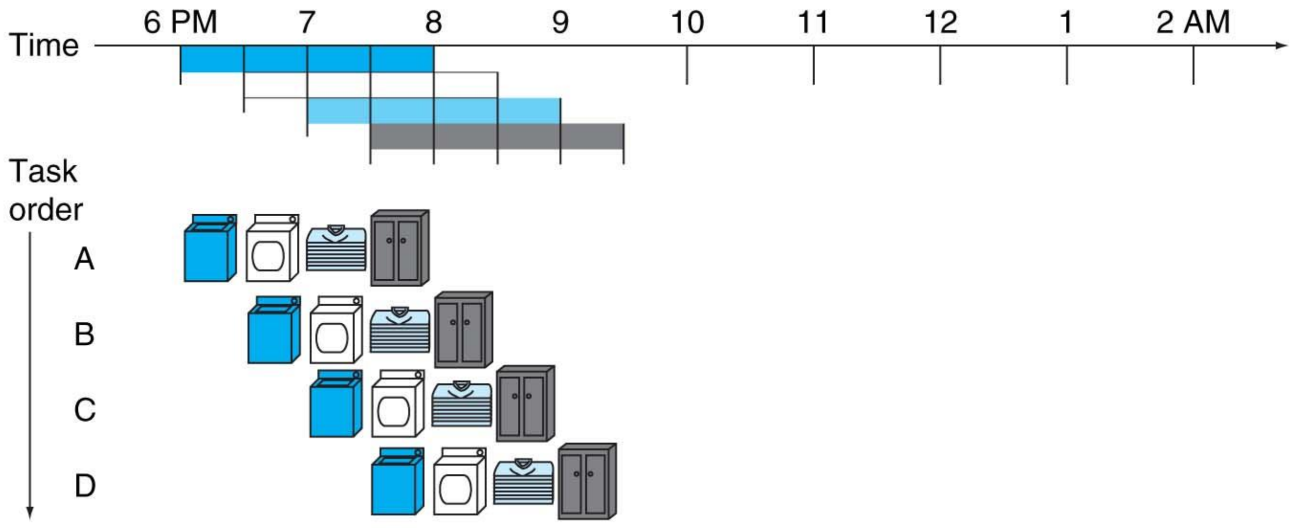
Replicate pipeline stages ⇒ multiple pipelines

- Start multiple instructions per clock cycle
- CPI < 1, so use Instructions Per Cycle (IPC)
    - But dependencies limit ILP in practice



Multiple Issue





Multiple Issue

Static multiple issue - Very Long Instruction Word (VLIW)

- Compiler groups instructions to be issued together
- Packages them into “issue slots”
- Compiler detects and avoids hazards

Dynamic multiple issue

- CPU examines instruction stream and chooses instructions to issue each cycle
- Compiler can help by reordering instructions
- CPU resolves hazards using advanced techniques at runtime

Static Multiple Issue

- Compiler groups instructions into “issue packets”
  - Group of instructions that can be issued on a single cycle
  - Determined by pipeline resources required
- Think of an issue packet as a very long instruction
  - Specifies multiple concurrent operations
    - ⇒ Very Long Instruction Word (VLIW)

Dynamic Pipeline Scheduling

- Allow the CPU to execute instructions out of order to avoid stalls
  - But commit result to registers in order
- Example

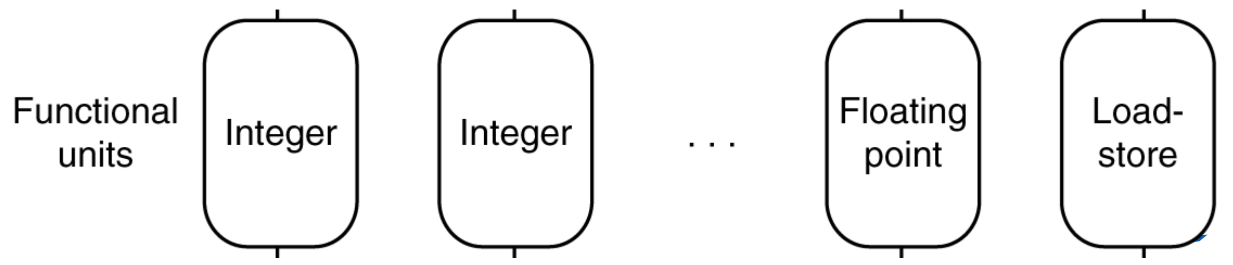
```
ld   x31, 0(x21)
add  x1, x31, x2
sub  x23, x23, x3
andi x5, x23, 20
```

  - Can start `sub` while `add` is waiting for `ld`

Dynamically Scheduled CPU

Preserves dependencies

Hold pending operands



Results sent to any waiting reservation stations

Rec
r g

y
operands for issued instructions

Register Renaming

- Reservation stations and reorder buffer provide register renaming
- When issuing instruction to a reservation station:
  - If operand is available in register file or reorder buffer
    - Copy to reservation station
  - If operand is not yet available
    - It will be provided to the reservation station by a functional unit

Why Do Dynamic Scheduling?

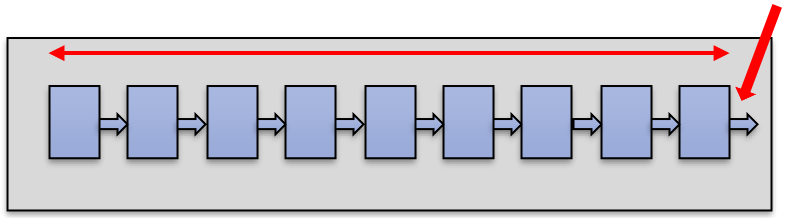
- Why not just let the compiler schedule code?
- Not all stalls are predicable
  - e.g., cache misses
- Can’t always schedule around branches
  - Branch outcome is dynamically determined
- Different implementations of an ISA have different latencies and hazards

Does Multiple Issue Work?

The BIG Picture

- Yes, but not as much as we'd like
- Programs have real dependencies that limit ILP
- Some dependencies are hard to eliminate
  - e.g., pointer aliasing
- Some parallelism is hard to expose
  - Limited window size during instruction issue
- Memory delays and limited bandwidth
  - Hard to keep pipelines full
- Speculation can help if done well

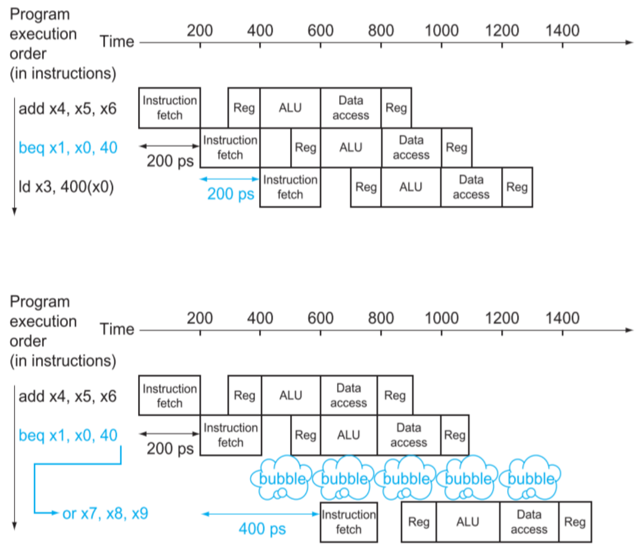
Pipelining and ILP



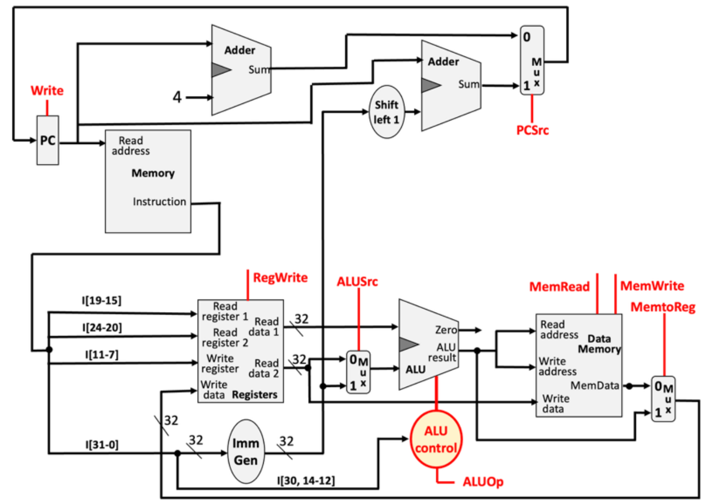
Pipelining improves instruction throughput using parallelism

- More instructions completed per second
- Latency for each instruction not reduced

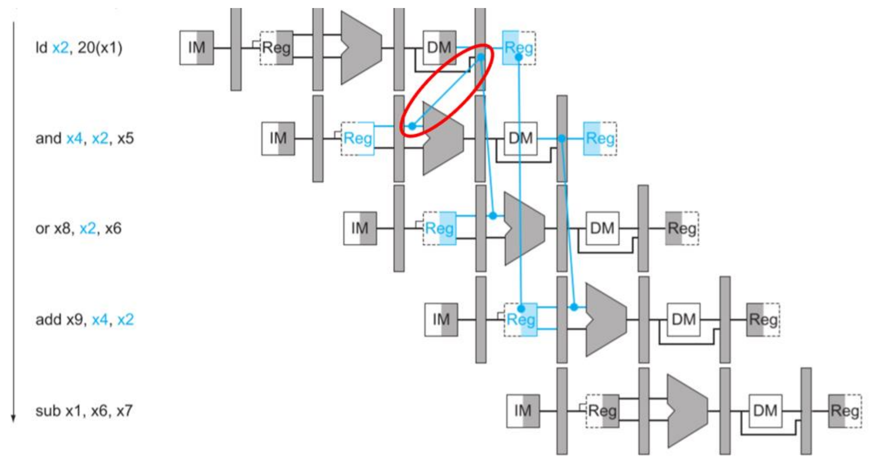
Hazards



control



structural



data

Pipelining and ILP

- Multiple issue and dynamic scheduling (ILP)
  - Dependencies limit achievable parallelism
  - Complexity leads to the power wall