Chapter #16
Instruction-Level Parallelism and Superscalar Processors

Instruction-level Parallelism

- Degree to which instructions of program can be executed in parallel

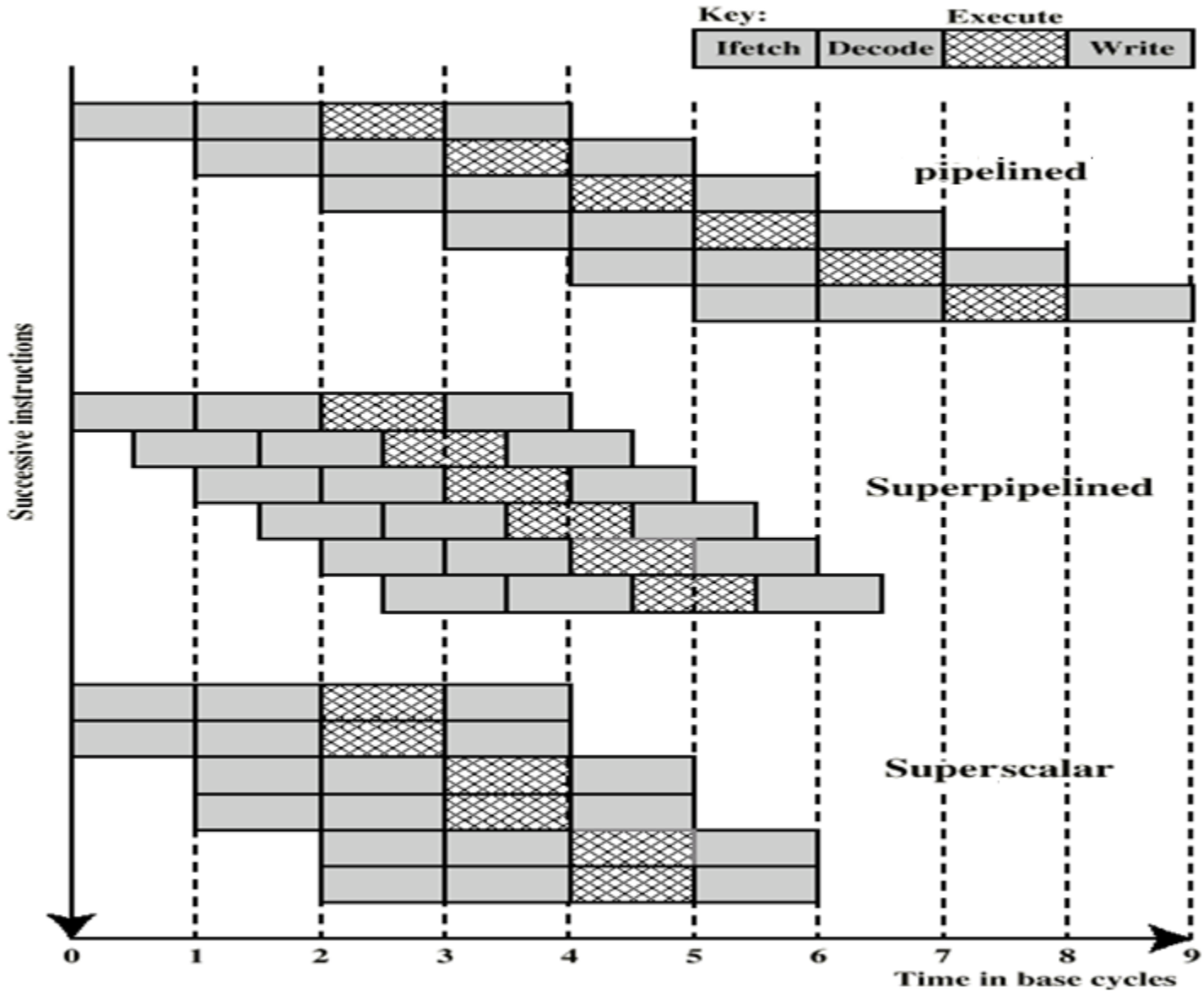
- Methods:
  - —Pipelining
  - —Superscalar
  - —Superpipelined

- Issues to resolve:
  - —Branches
  - —Dependencies

Superscalar vs. Superpipelined

- Superscalar:
  - —Fetches and executes multiple instructions independently and simultaneously in different pipelines via multiple functional units
- Superpipelined:
  - —Allows overlapping of instructions beyond 1 external clock cycle using doubled internal clock cycle & dividing units into sections

Superscalar & Superpipelined Examples



Dependencies

- Read-after-write (RAW, *true dependency*)
  - —Second instruction cannot read operand until first instruction writes result
- Write-after-read (WAR, *antidependency*)
  - —Second instruction must not write result before first instruction reads operand
- Write-after write (WAW, *output dependency*)
  - —Second instruction must not write result before first instruction writes result to same destination

RAW Dependency

- Example:

```
r2:=r0+r1;
r3:=r2-r3;
```

- Ability:
  - —Fetch and decode second instruction in parallel with first
- Inability:
  - —Execute second instruction before first instruction writes output
- Solution:
  - —Out-of-order completion of instructions

WAR Dependency

- Example:

```
r3:=r2-r3;
r2:=r0+r1;
```

- Ability:
  - —Fetch and decode second instruction in parallel with first
- Inability:
  - —Execute second instruction before first instruction reads input, creating false dependency (antidependency)
- Solution:
  - —Register renaming

WAW Dependency

- Example:

```
r3:=r1-r3;
r2:=r3+r1;
r3:=r0+r1;
```

- Ability:
  - —Fetch and decode third instruction in parallel with first
- Inability:
  - —Execute third instruction before first instruction writes output, creating false dependency (output dependency)
- Solution:
  - —Register renaming

Register Renaming

- Motivation:
  - —WAR and WAW occur because register contents may not reflect the correct ordering from the program
  - —May result in a pipeline stall
  - —Registers allocated dynamically
- Approach:
  - —Rename register to either unused or scratch register to remove non-true dependency

In-Order Completion

- Complete instructions in the order they occur

- Adv:
  - —Simple approach
  - —No extra hardware necessary

- Disadv:
  - —Instructions must stall if necessary
  - —Reduces optimal performance

Out-of-Order Completion

- Complete instructions out of the order they originally occur

- Adv:
  - —Can reduce stalled stages in pipeline
  - —Can achieve more optimal performance

- Disadv:
  - —More complex to manage
  - —Need to store resource usage