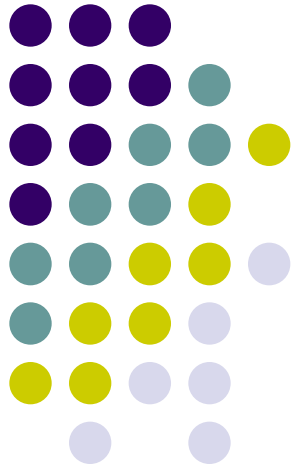
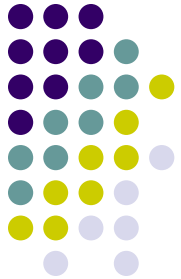
Chapter 8. Pipelining

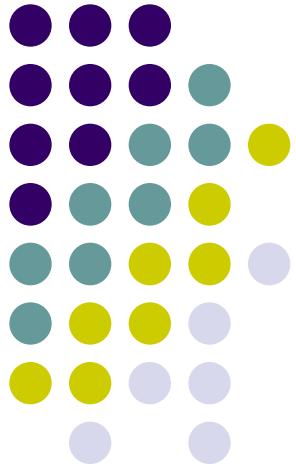


Overview

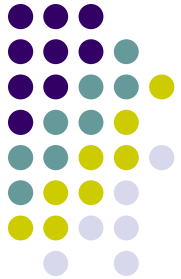


- Pipelining is widely used in modern processors.
- Pipelining improves system performance in terms of throughput.
- Pipelined organization requires sophisticated compilation techniques.

Basic Concepts



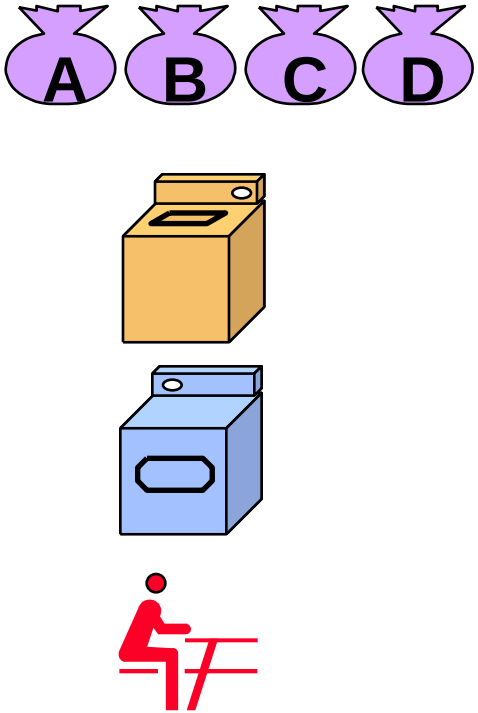
Making the Execution of Programs Faster



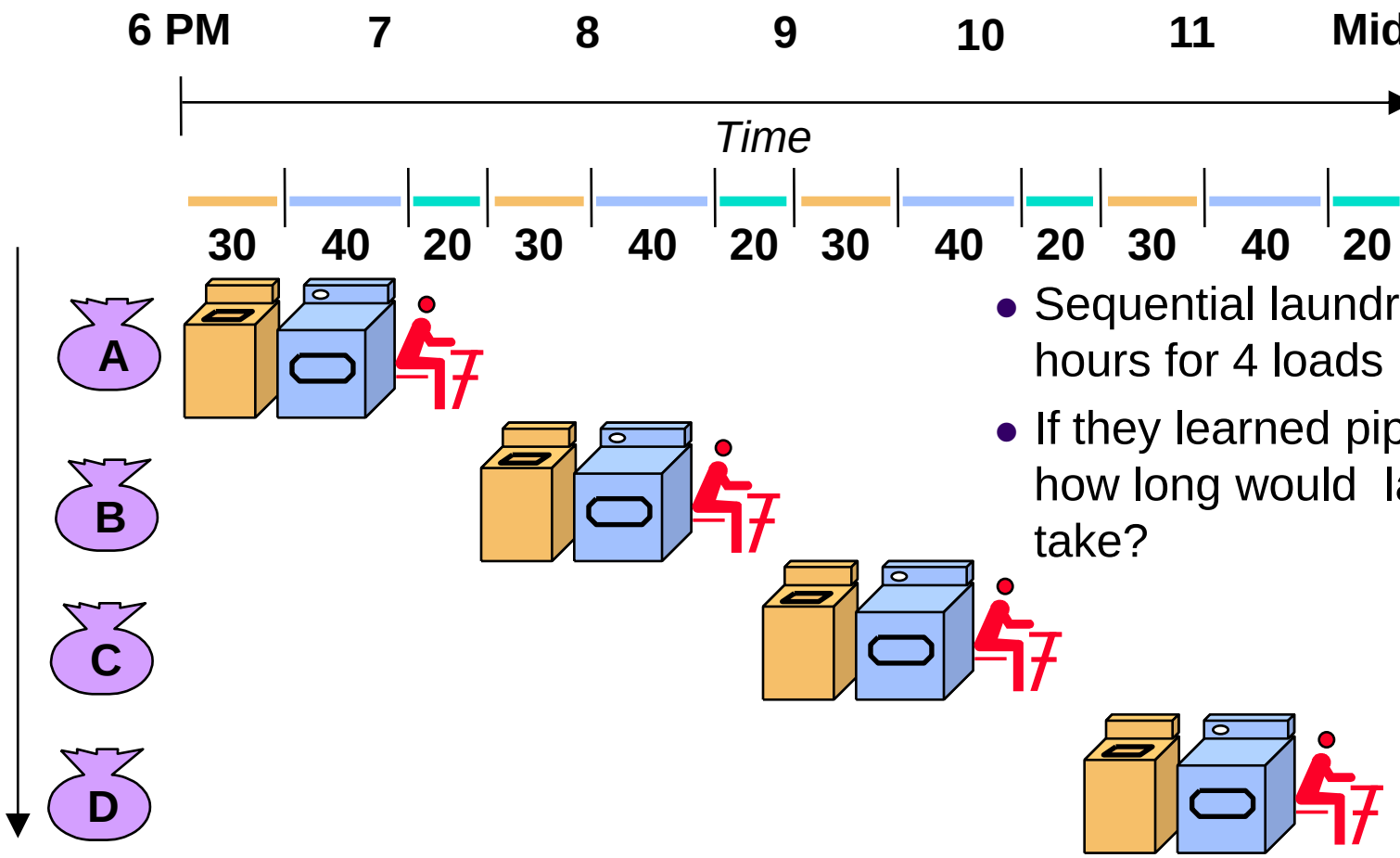
- Use faster circuit technology to build the processor and the main memory.
- Arrange the hardware so that more than one operation can be performed at the same time.
- In the latter way, the number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed.

Traditional Pipeline Concept

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- “Folder” takes 20 minutes

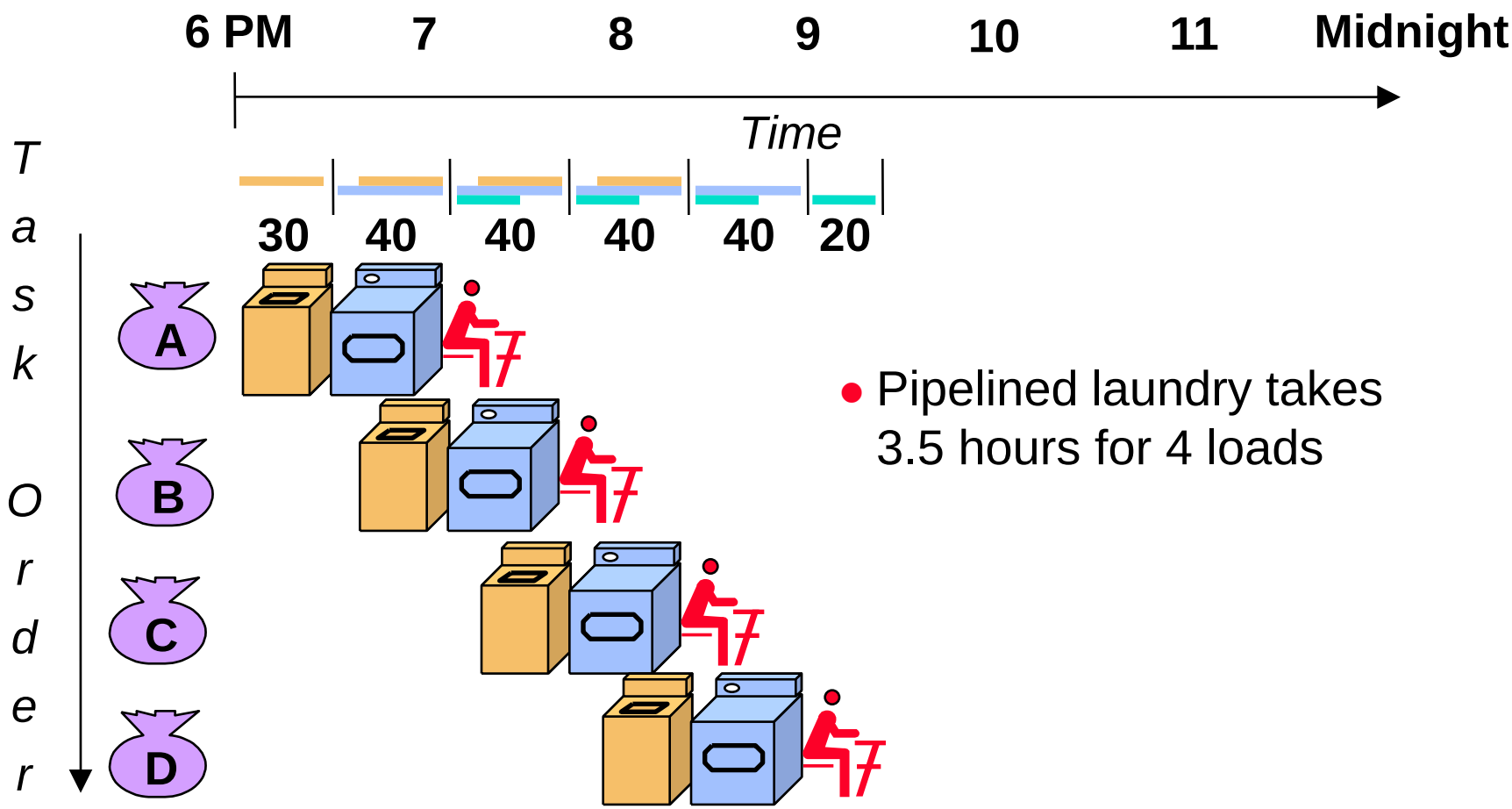


Traditional Pipeline Concept



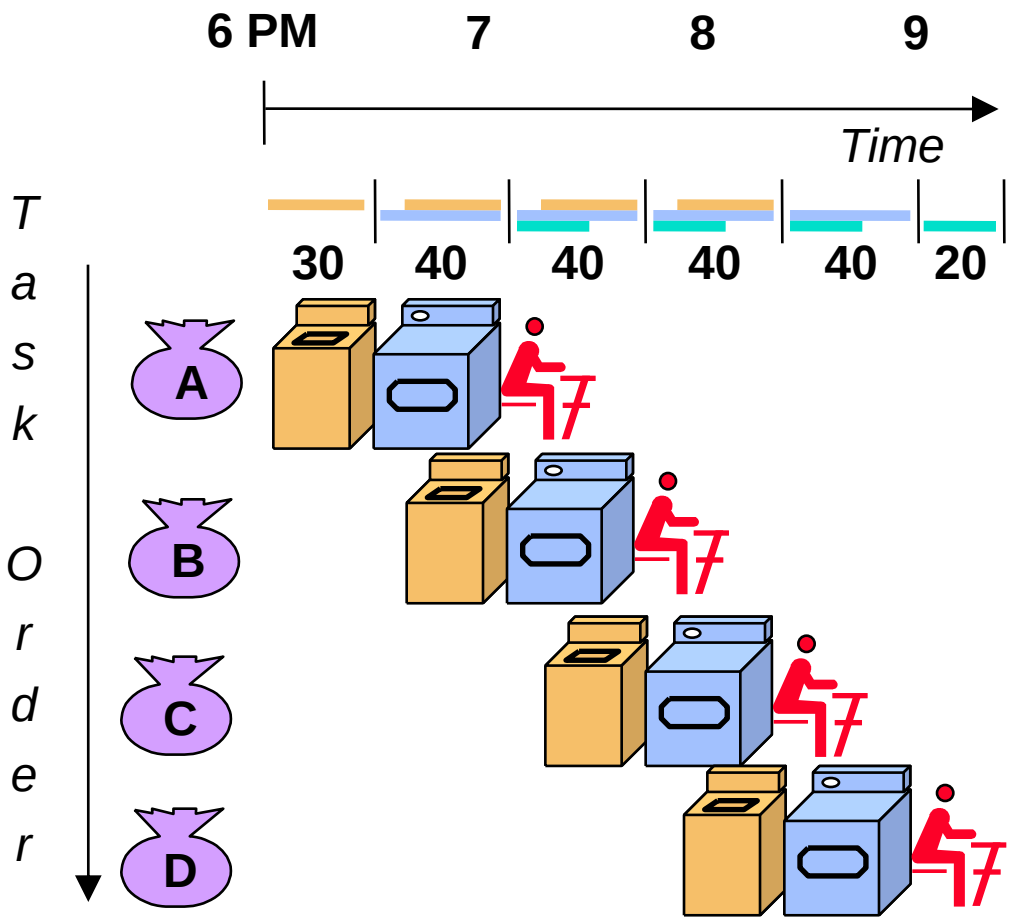
- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Traditional Pipeline Concept



- Pipelined laundry takes 3.5 hours for 4 loads

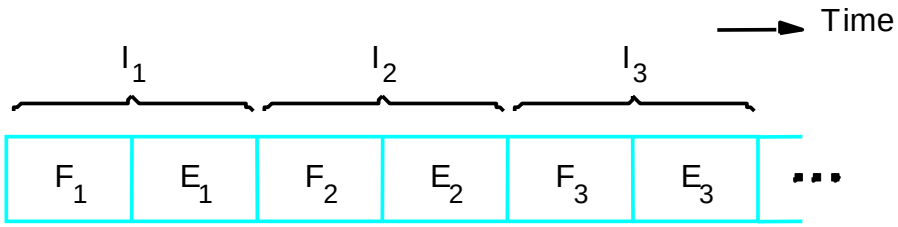
Traditional Pipeline Concept



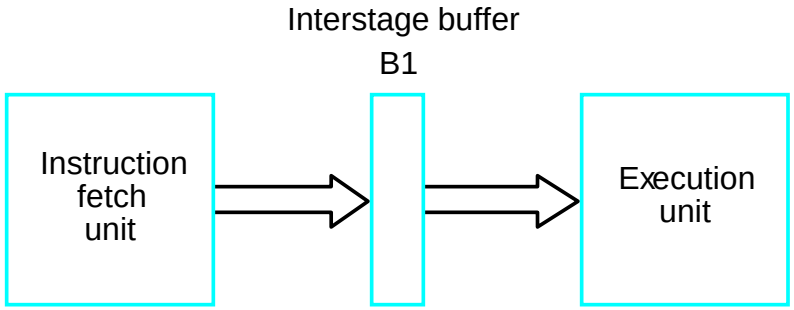
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup
- Stall for Dependences

Use the Idea of Pipelining in a Computer

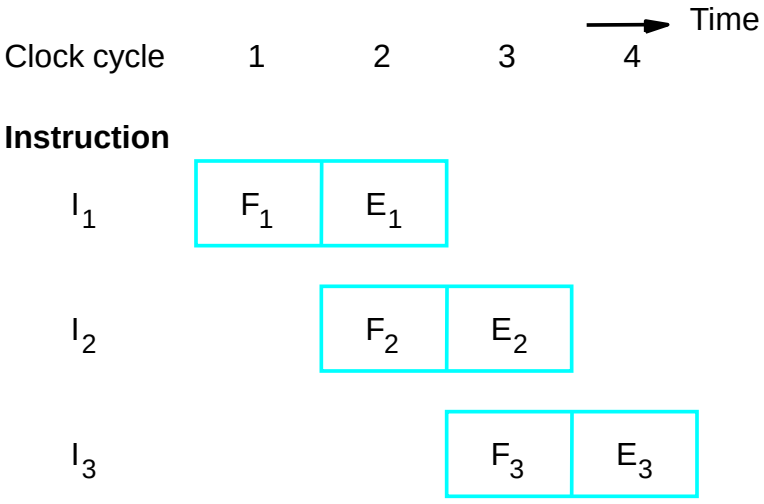
Fetch + Execution



(a) Sequential execution



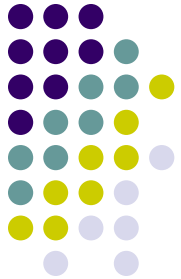
(b) Hardware organization



(c) Pipelined execution

Figure 8.1. Basic idea of instruction pipelining.

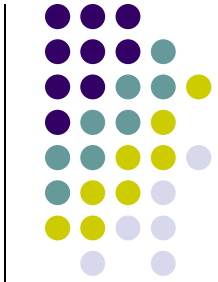
Use the Idea of Pipelining in a Computer



Fetch + Decode
+ Execution + Write

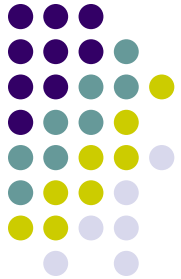
Textbook page: 457

Role of Cache Memory



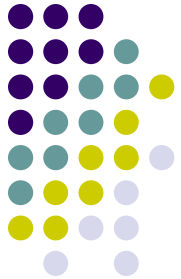
- Each pipeline stage is expected to complete in one clock cycle.
- The clock period should be long enough to let the slowest pipeline stage to complete.
- Faster stages can only wait for the slowest one to complete.
- Since main memory is very slow compared to the execution, if each instruction needs to be fetched from main memory, pipeline is almost useless.
- Fortunately, we have cache.

Pipeline Performance

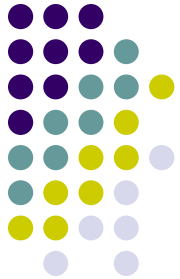


- The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages.
- However, this increase would be achieved only if all pipeline stages require the same time to complete, and there is no interruption throughout program execution.
- Unfortunately, this is not true.

Pipeline Performance

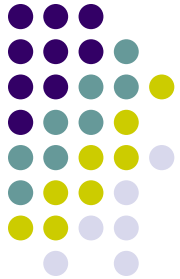


Pipeline Performance



- The previous pipeline is said to have been stalled for two clock cycles.
- Any condition that causes a pipeline to stall is called a hazard.
- Data hazard – any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. So some operation has to be delayed, and the pipeline stalls.
- Instruction (control) hazard – a delay in the availability of an instruction causes the pipeline to stall.
- Structural hazard – the situation when two instructions require the use of a given hardware resource at the same time.

Pipeline Performance



Instruction
hazard

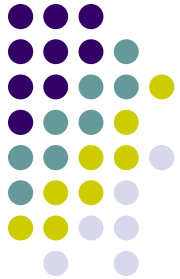
Idle periods –
stalls (bubbles)

Pipeline Performance

Structural hazard

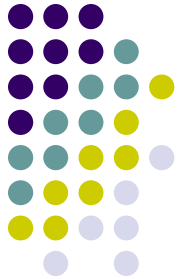
Load X(R1), R2

Pipeline Performance



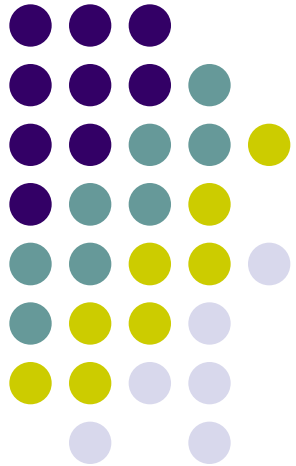
- Again, pipelining does not result in individual instructions being executed faster; rather, it is the throughput that increases.
- Throughput is measured by the rate at which instruction execution is completed.
- Pipeline stall causes degradation in pipeline performance.
- We need to identify all hazards that may cause the pipeline to stall and to find ways to minimize their impact.

Quiz

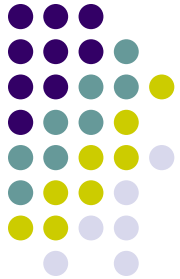


- Four instructions, the I2 takes two clock cycles for execution. Pls draw the figure for 4-stage pipeline, and figure out the total cycles needed for the four instructions to complete.

Data Hazards



Data Hazards



- We must ensure that the results obtained when instructions are executed in a pipelined processor are identical to those obtained when the same instructions are executed sequentially.
- Hazard occurs

  $A \leftarrow 3 + A$

  $B \leftarrow 4 \times A$
- No hazard

  $A \leftarrow 5 \times C$

  $B \leftarrow 20 + C$
- When two operations depend on each other, they must be executed sequentially in the correct order.
- Another example:

  Mul R2, R3, R4

  Add R5, R4, R6

Data Hazards

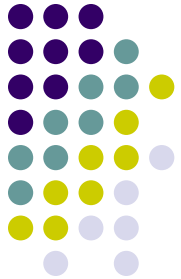
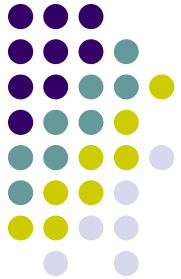
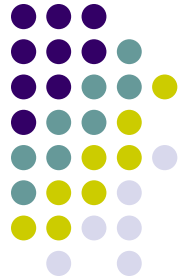


Figure 8.6. Pipeline stalled by data dependency between D_2 and W_1.

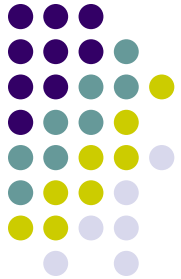
Operand Forwarding



- Instead of from the register file, the second instruction can get data directly from the output of ALU after the previous instruction is completed.
- A special arrangement needs to be made to “forward” the output of ALU to the input of ALU.



Handling Data Hazards in Software

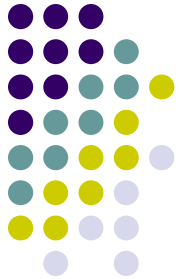


- Let the compiler detect and handle the hazard:

```
I1: Mul  R2, R3, R4
    NOP
    NOP
I2: Add  R5, R4, R6
```

- The compiler can reorder the instructions to perform some useful work during the NOP slots.

Side Effects



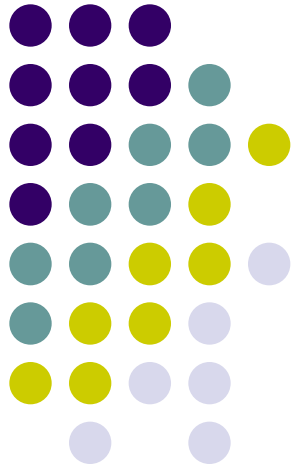
- The previous example is explicit and easily detected.
- Sometimes an instruction changes the contents of a register other than the one named as the destination.
- When a location other than one explicitly named in an instruction as a destination operand is affected, the instruction is said to have a side effect. (Example?)
- Example: conditional code flags:

  Add R1, R3

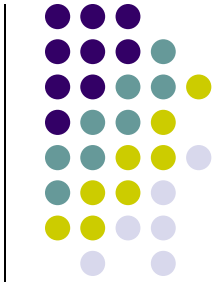
  AddWithCarry R2, R4

- Instructions designed for execution on pipelined hardware should have few side effects.

Instruction Hazards

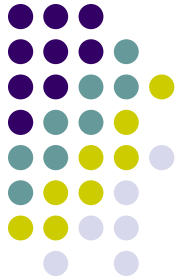


Overview

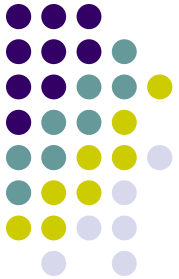


- Whenever the stream of instructions supplied by the instruction fetch unit is interrupted, the pipeline stalls.
- Cache miss
- Branch

Unconditional Branches



Branch Timing



- Branch penalty

- Reducing the penalty

Instruction Queue and Prefetching

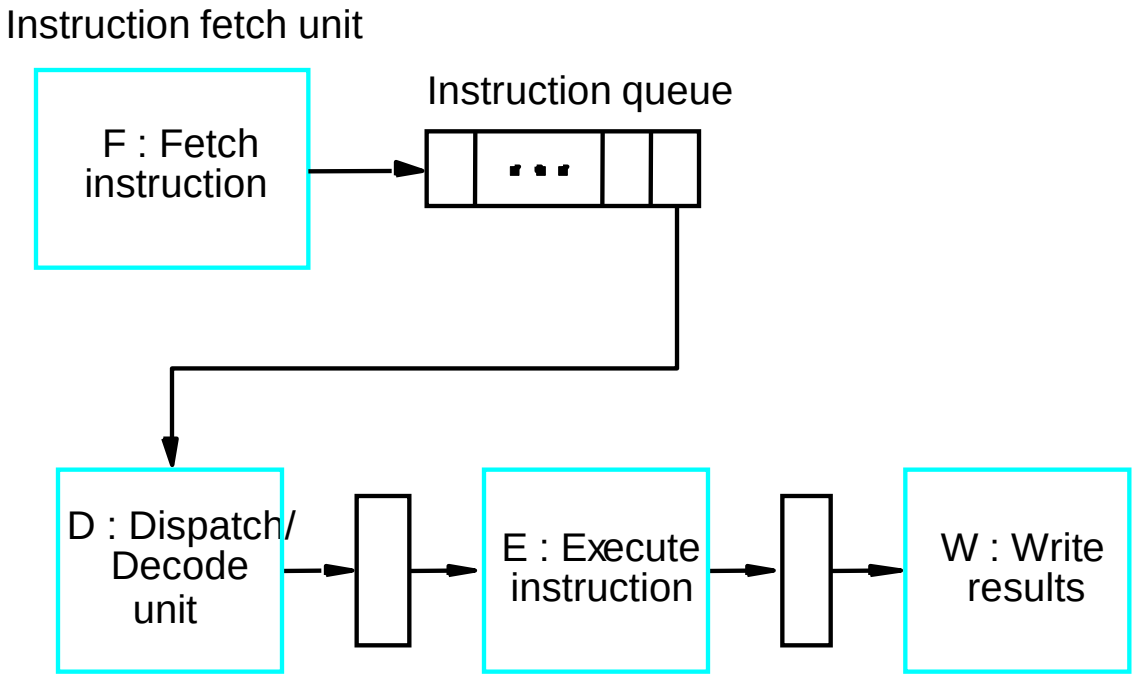
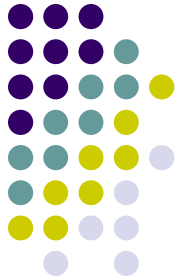


Figure 8.10. Use of an instruction queue in the hardware organization of Figure 8.2*b*.

Conditional Braches



- A conditional branch instruction introduces the added hazard caused by the dependency of the branch condition on the result of a preceding instruction.
- The decision to branch cannot be made until the execution of that instruction has been completed.
- Branch instructions represent about 20% of the dynamic instruction count of most programs.