

Y86 Pipelined Implementation

Overview

General Principles of Pipelining

- Goal
- Difficulties

Creating a Pipelined Y86 Processor

- Rearranging SEQ
- Inserting pipeline registers
- Problems with data and control hazards

Real-World Pipelines: Car Washes

Sequential



Parallel



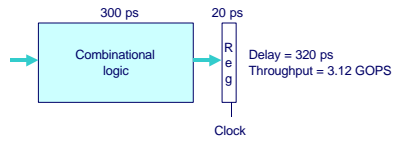
Pipelined



Idea

- Divide process into independent stages
- Move objects through stages in sequence
- At any given times, multiple objects being processed

Computational Example



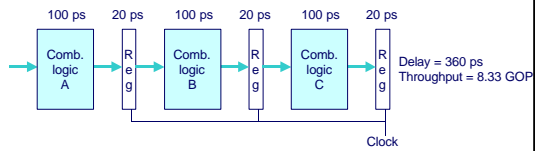
System

- Computation requires total of 300 picoseconds
- Additional 20 picoseconds to save result in register
- Can must have clock cycle of at least 320 ps

CompOrg : Y86 Pipelined

4

3-Way Pipelined Version



System

- Divide combinational logic into 3 blocks of 100 ps each
- Can begin new operation as soon as previous one passes through stage A.
 - Begin new operation every 120 ps
- Overall latency increases
 - 360 ps from start to finish

CompOrg : Y86 Pipelined

5

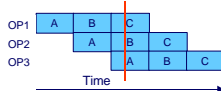
Pipeline Diagrams

Unpipelined



- Cannot start new operation until previous one completes

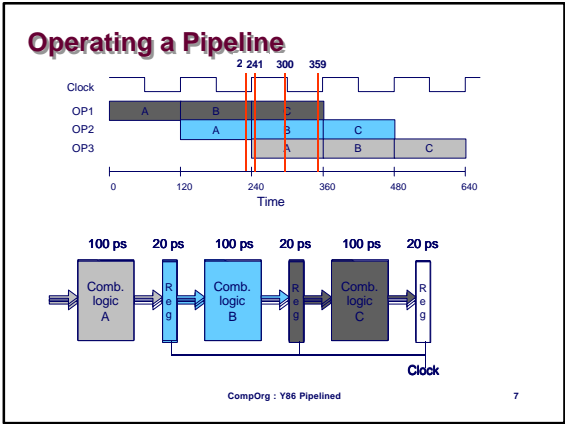
3-Way Pipelined

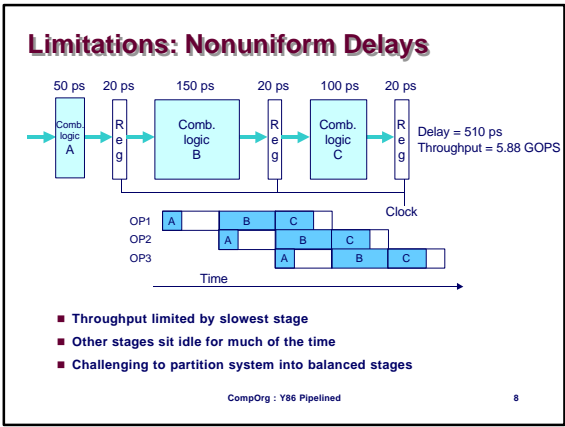


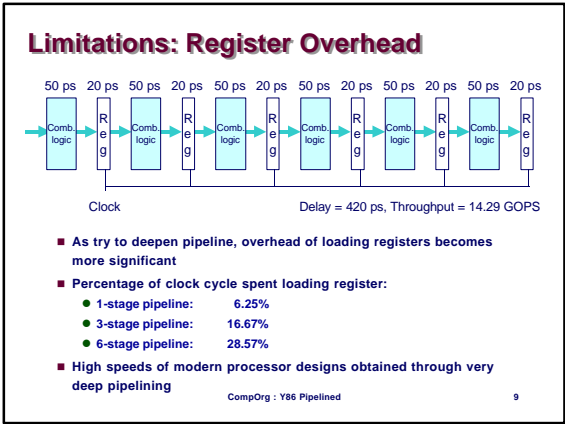
- Up to 3 operations in process simultaneously

CompOrg : Y86 Pipelined

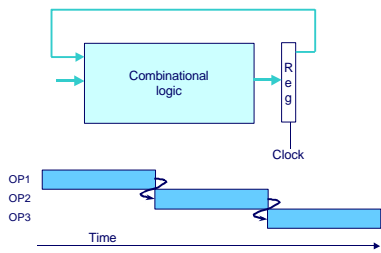
6







Data Dependencies



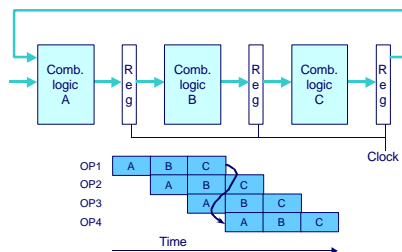
System

- Each operation depends on result from preceding one

CompOrg : Y86 Pipelined

10

Data Hazards



- Result does not feed back around in time for next operation
- Pipelining has changed behavior of system

CompOrg : Y86 Pipelined

11

Data Dependencies in Processors

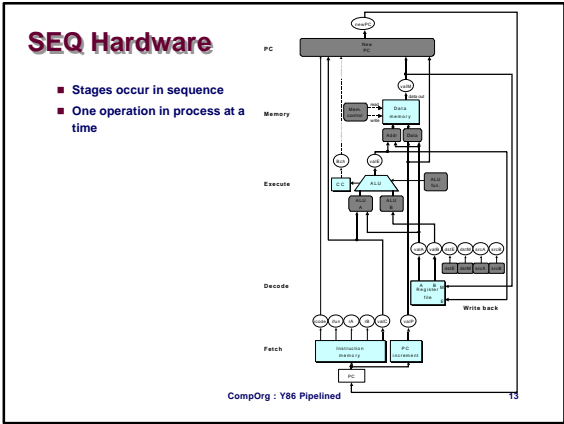
```

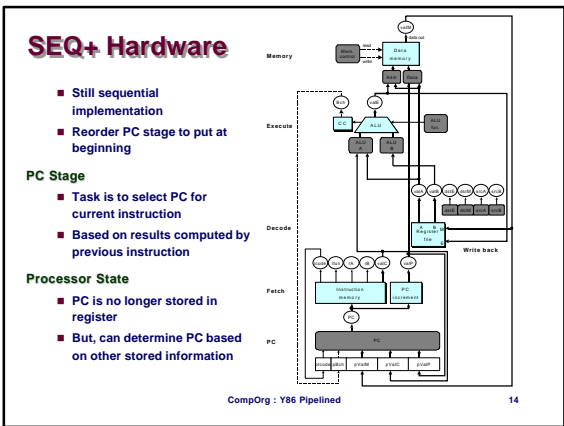
1  irmovl %50, %eax
2  addl (%eax), %ebx
3  mrmovl 100(%ebx), %edx
    
```

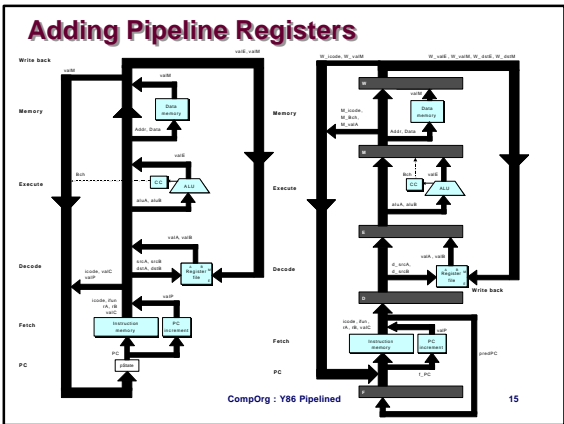
- Result from one instruction used as operand for another
 - Read-after-write (RAW) dependency
- Very common in actual programs
- Must make sure our pipeline handles these properly
 - Get correct results
 - Minimize performance impact

CompOrg : Y86 Pipelined

12



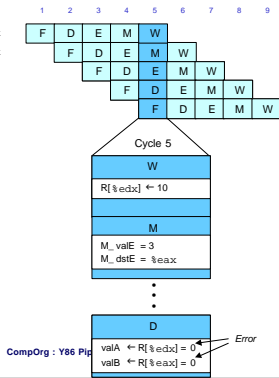




Data Dependencies: 1 Nop

demo-h1.ya

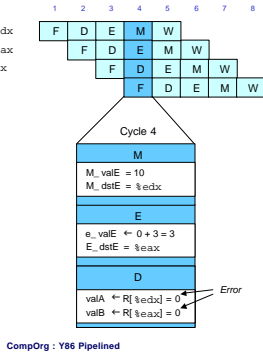
```
0x000: irmovl $10,%edx
0x006: irmovl $3,%eax
0x00c: nop
0x00d: addl %edx,%eax
0x00f: halt
```



Data Dependencies: No Nop

demo-h0.ya

```
0x000: irmovl $10,%edx
0x006: irmovl $3,%eax
0x00c: addl %edx,%eax
0x00e: halt
```



Branch Misprediction Example

demo-j.ya

```
0x000: xorl %eax,%eax
0x002: jne t # Not taken
0x007: irmovl $1,%eax # Fall through
0x00d: nop
0x00e: nop
0x00f: nop
0x010: halt
0x011: t: irmovl $3,%edx # Target (Should not execute)
0x017: irmovl $4,%ecx # Should not execute
0x01d: irmovl $5,%edx # Should not execute
```

- Should only execute first 8 instructions

CompOrg : Y86 Pipelined

27

Pipeline Summary

Concept

- Break instruction execution into 5 stages
- Run instructions through in pipelined mode

Limitations

- Can't handle dependencies between instructions when instructions follow too closely
- Data dependencies
 - One instruction writes register, later one reads it
- Control dependency
 - Instruction sets PC in way that pipeline did not predict correctly
 - Mispredicted branch and return

Fixing the Pipeline

- We'll do that next time
