18-640 Foundations of Computer Architecture

Lecture 3:
"From Pipelined to Superscalar Processors"

John Paul Shen
September 2, 2014

- Required Reading Assignment:
  - **Chapters 3 and 4 of Shen and Lipasti (SnL).**
- Recommended Reading Assignment:
  - "*The Interaction of Architecture and Operating System Design*," by Anderson, Levy, Bershad, and Lazowska, ASPLOS 1991.

Electrical & Computer ENGINEERING
Carnegie Mellon University

18-640 Foundations of Computer Architecture

Lecture 3:
"From Pipelined to Superscalar Processors"

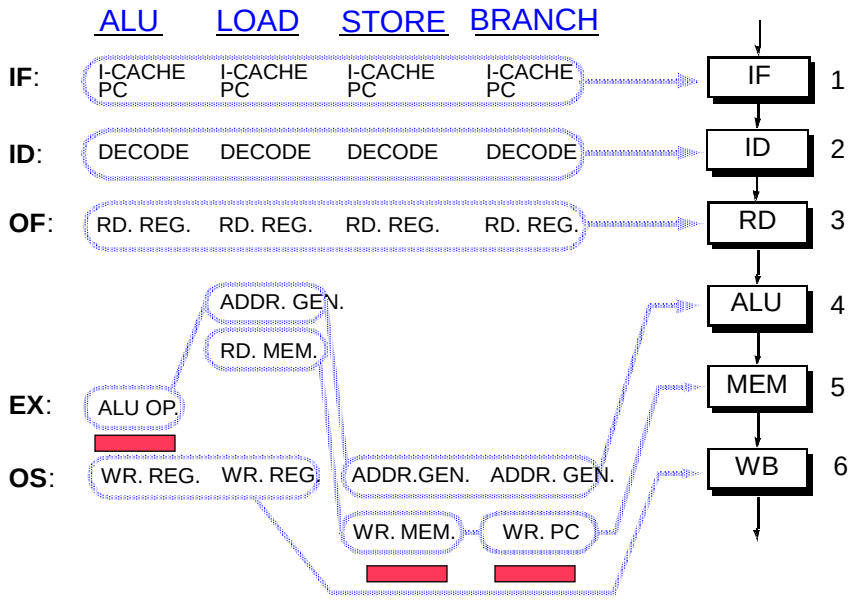
A. Limitations of Scalar Pipelined Processors
   - IBM RISC Experience
B. Modern Superscalar Processor Organization
   - Parallel, Diversified, and Dynamic Pipelines
C. Limits on Instruction Level Parallelism (ILP)
   - Motivation for Superscalar Processors
D. Computer Architecture & Operating Systems

Electrical & Computer ENGINEERING
Carnegie Mellon University

Review from Lecture 2

Unifying Instruction Types in a Pipelined Processor

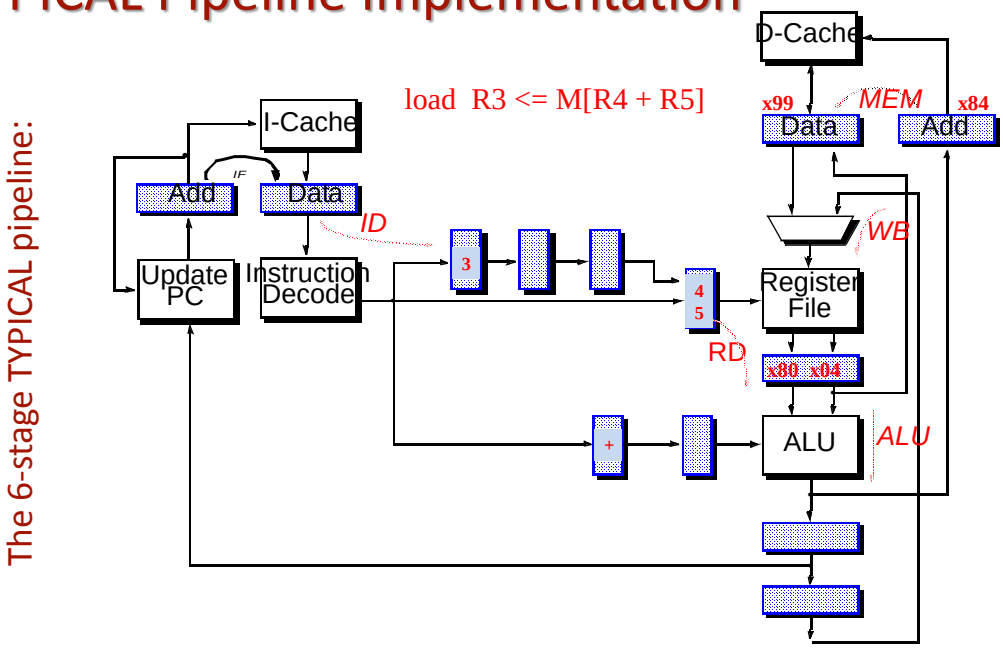
The 6-stage TYPICAL pipeline:



Review from Lecture 2

TYPICAL Pipeline Implementation

The 6-stage TYPICAL pipeline:



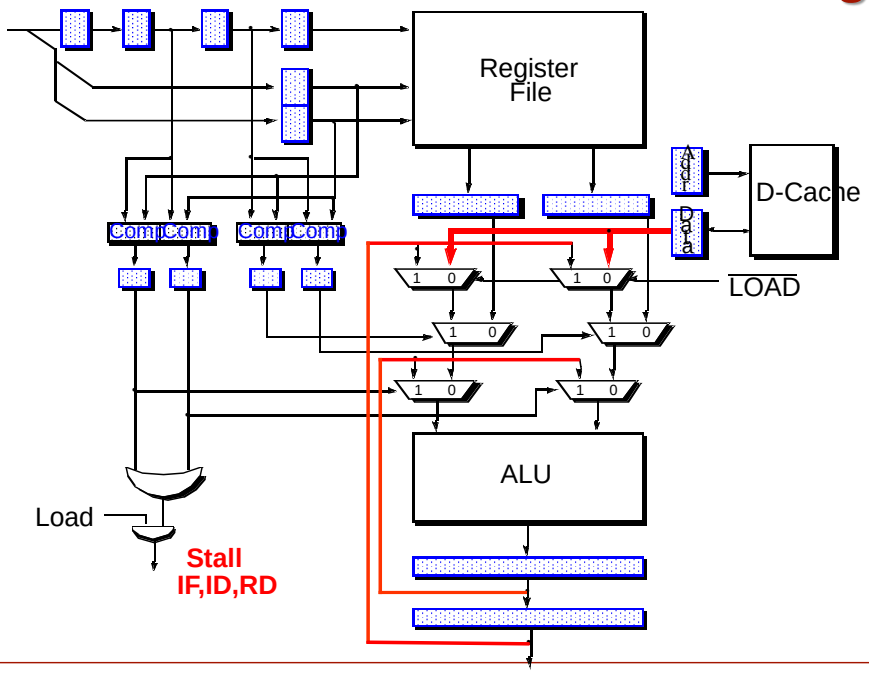
Review from Lecture 2

Inter-Instruction Hazards

Pipe Stage	ALU Inst.	Load inst.	Store inst.	Branch inst.
1. IF	I-cache PC<PC+4	I-cache PC<PC+4	I-cache PC<PC+4	I-cache PC<PC+4
2. ID	decode	decode	decode	decode
3. RD	read reg.	read reg.	read reg.	read reg.
4. ALU	ALU op.	addr. gen.	addr. gen.	addr. gen. cond. gen.
5. MEM	-----	read mem.	write mem.	PC<-br. addr.
6. WB	write reg.	write reg.	-----	-----

Review from Lecture 2

Implementation of ALU and Load Forwarding Paths



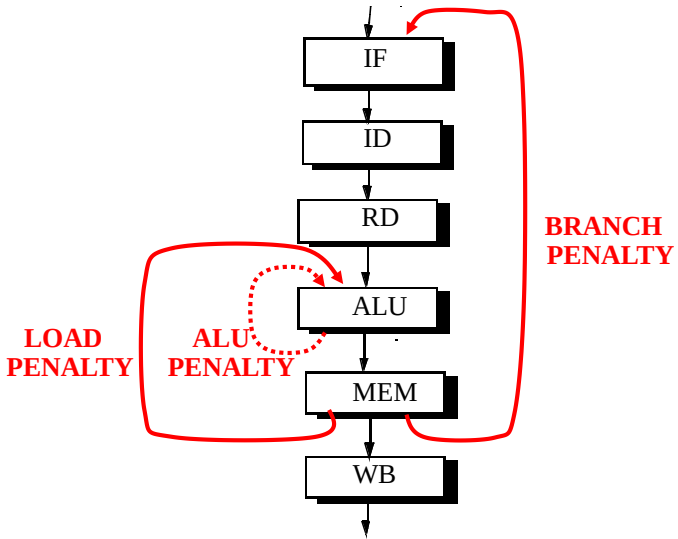
Review from Lecture 2

Penalties with Forwarding Paths

Leading $Inst_i$	ALU	Load	Branch
Trailing $Inst_j$	ALU, L/S, Br.	ALU, L/S, Br.	ALU, L/S, Br.
Hazard register	Int. Reg. (Ri)	Int. Reg. (Ri)	PC
Register WRITE stage (i)	WB (stage 6)	WB (stage 6)	MEM (stage 5)
Register READ stage (j)	RD (stage 3)	RD (stage 3)	IF (stage 1)
Forward from outputs of:	ALU,MEM,WB	MEM,WB	MEM
Forward to input of:	ALU	ALU	IF
RAW distance or penalty:	0 cycles	1 cycles	4 cycles

Review from Lecture 2

3 Major Penalty Loops of (Scalar) Pipelining



Performance Objective: Reduce CPI as close to 1 as possible.

IBM RISC Experience [Agerwala and Cocke 1987]

- Internal IBM study: Limits of a scalar pipeline?
- Memory Bandwidth
  - Fetch 1 instr/cycle from I-cache
  - 40% of instructions are load/store (D-cache)
- Code characteristics (dynamic)
  - Loads – 25%
  - Stores 15%
  - ALU/RR – 40%
  - Branches – 20%
    - 1/3 unconditional (always taken
    - 1/3 conditional taken, 1/3 conditional not taken

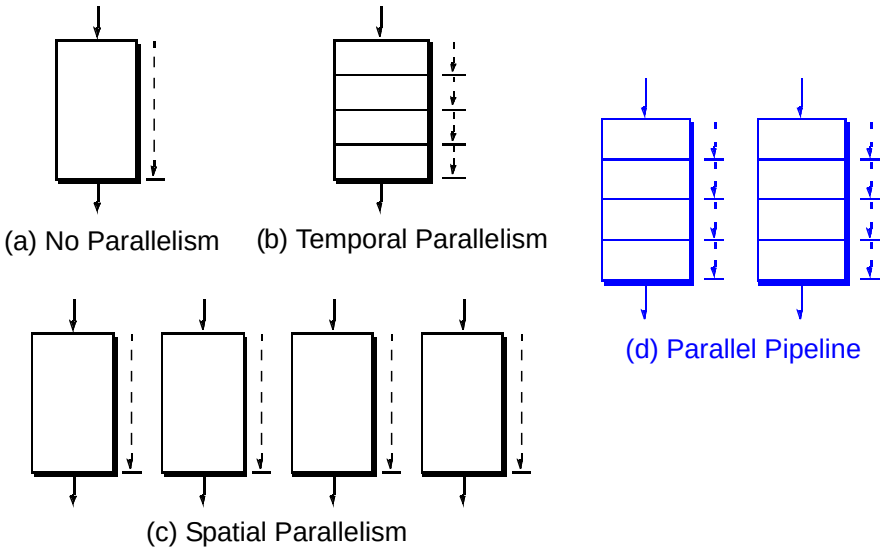
A. Limitations of Scalar Pipelined Processors

- IBM RISC Experience
  - Control and data dependences add 15%
  - Best case CPI of 1.15, IPC of 0.87
  - Deeper pipelines (higher frequency) magnify dependence penalties
- This analysis assumes 100% cache hit rates
  - Hit rates approach 100% for some programs
  - Many important programs have much worse hit rates

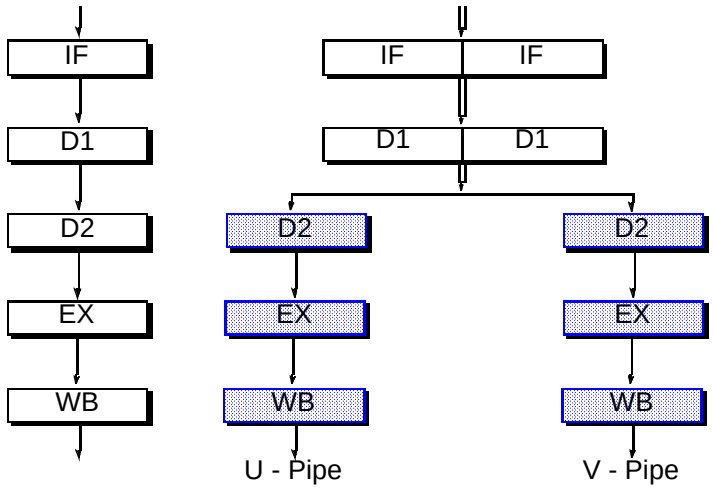
Address Limitations of Scalar Pipelined Processors

- Upper Bound on Scalar Pipeline Throughput
  Limited by IPC = 1
  ➔ Parallel Pipelines
- Inefficient Unification Into Single Pipeline
  Long latency for each instruction
  Hazards and associated stalls
  ➔ Diversified Pipelines
- Performance Lost Due to In-order Pipeline
  Unnecessary stalls
  ➔ Dynamic Pipelines

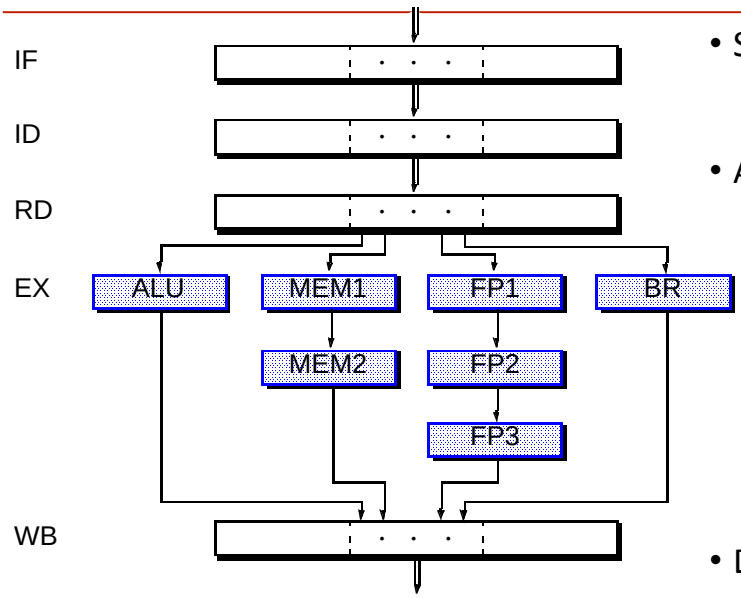
Parallel Pipelines



Intel Pentium Parallel Pipeline

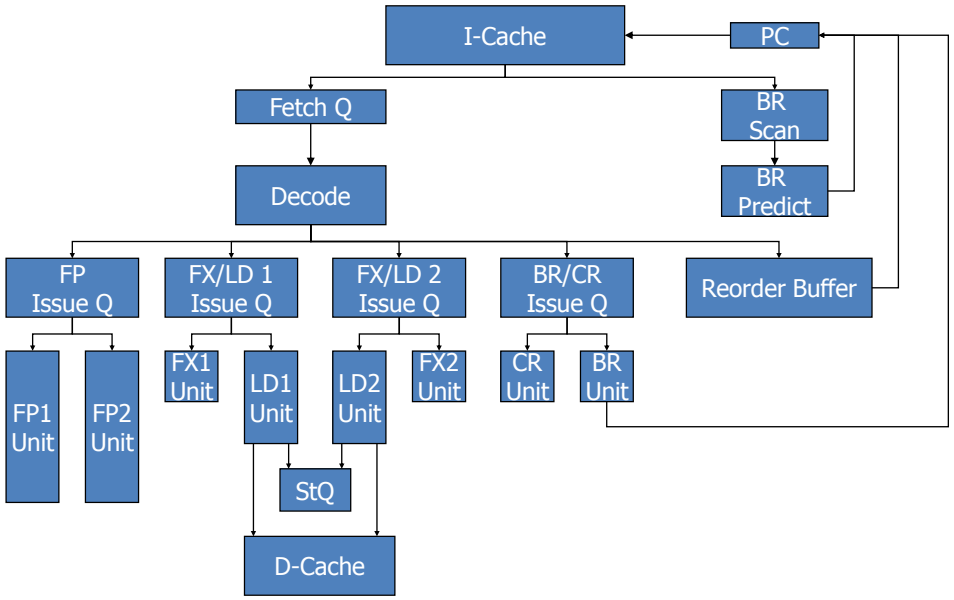


Diversified Pipelines

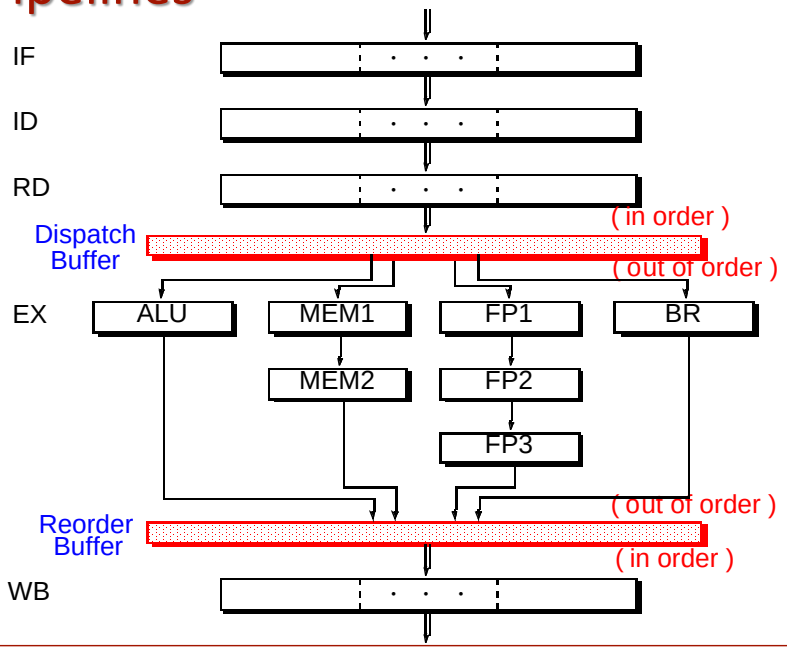


- Separate execution pipelines
  - Integer simple, memory, FP, …
- Advantages:
  - Reduce instruction latency
    - Each instruction goes to WB asap
    - Eliminate need for forwarding paths
  - Eliminate some unnecessary stalls
    - E.g. slow FP instruction does not block independent integer instructions
- Disadvantages ??

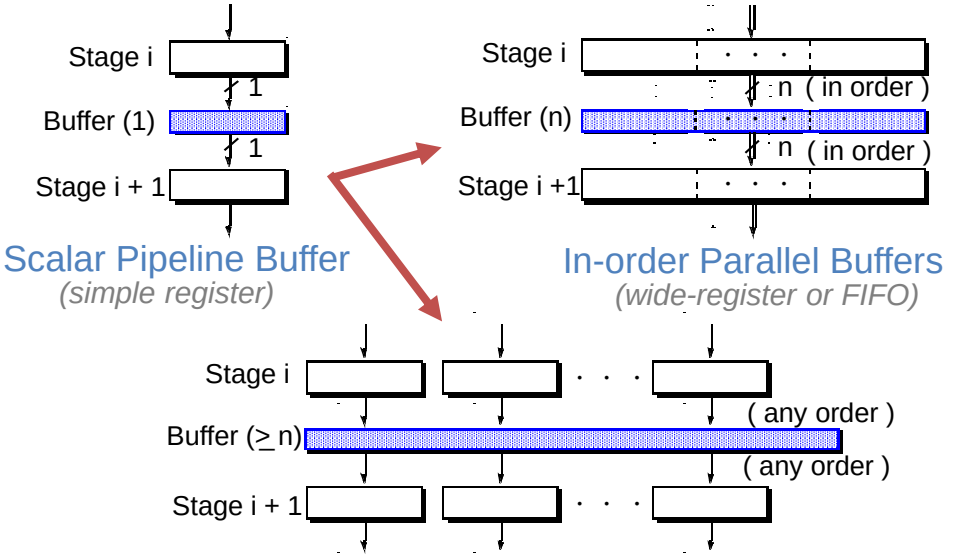
Power4 Diversified Pipelines



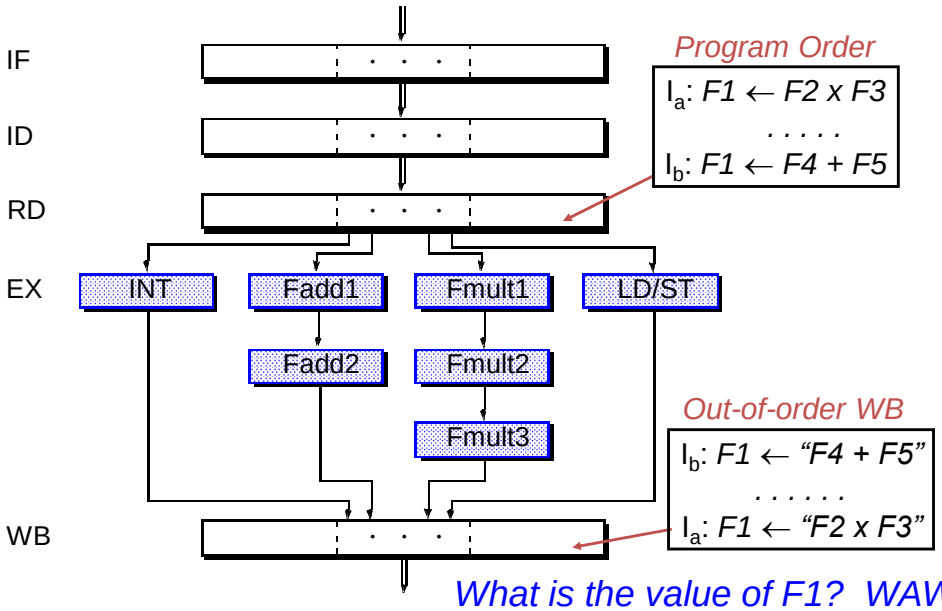
Dynamic Pipelines



Designs of Inter-stage Buffers

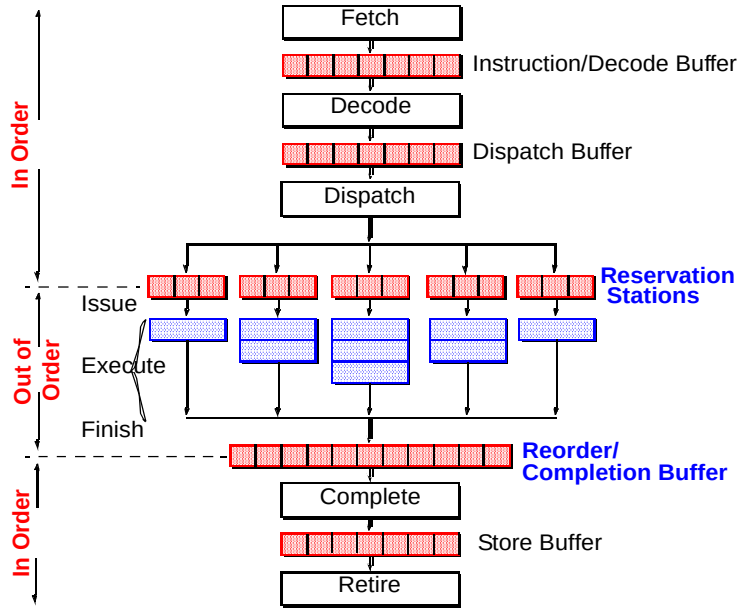


The Challenges of Out-of-Order

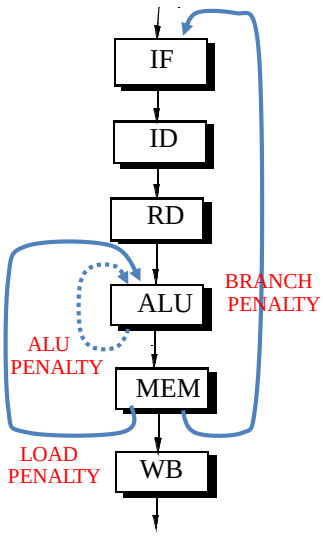


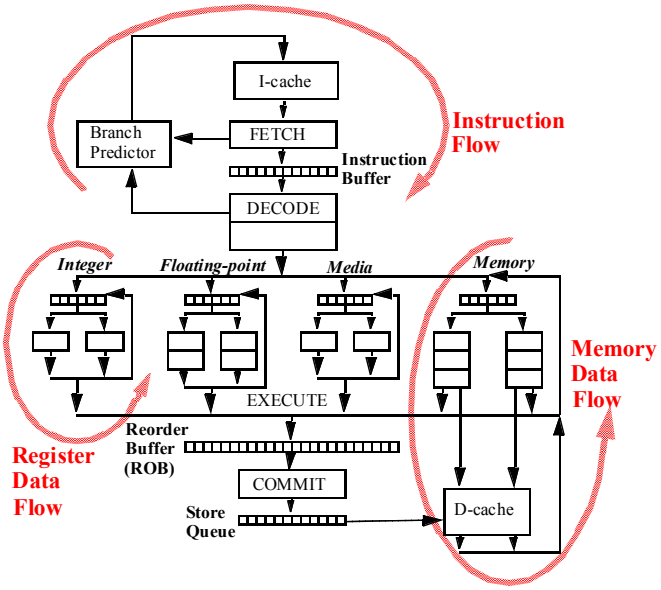
What is the value of F1? WAW!!!

B. Modern Superscalar Processor Organization

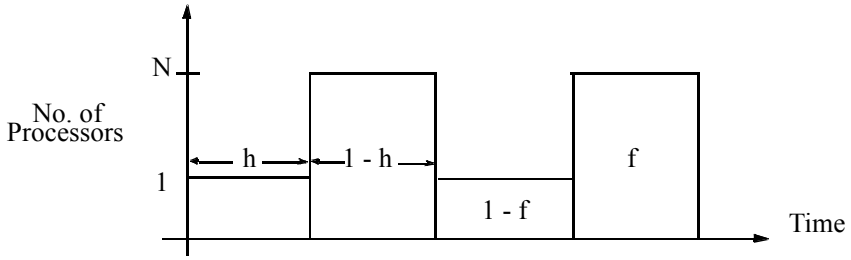


Impediments to Superscalar Performance





Amdahl's Law and Instruction Level Parallelism



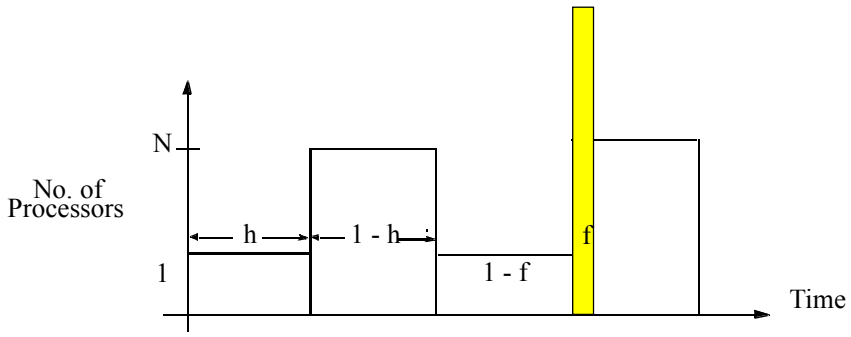
- h = fraction of time in serial code
- f = fraction that is vectorizable or parallelizable
- N = max speedup for f
- Overall speedup → →

$$Speedup = \frac{1}{(1-f) + \frac{f}{N}}$$

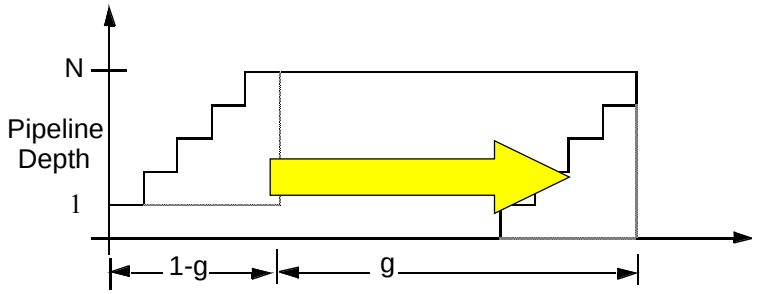
Revisit Amdahl's Law

- Sequential bottleneck
- Even if N is infinite
  - Performance limited by non-vectorizable portion (1-f)

$$\lim_{N \to \infty} \frac{1}{(1-f) + \frac{f}{N}} = \frac{1}{1-f}$$

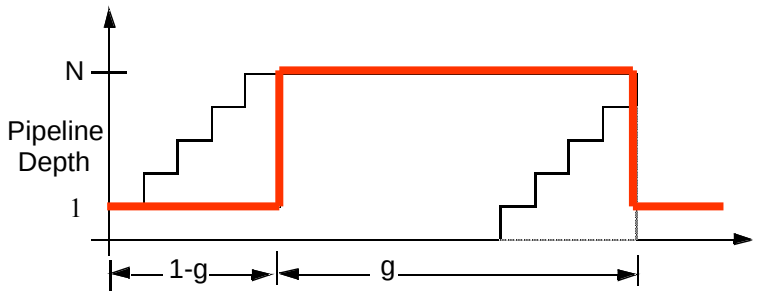


Pipelined Processor Performance Model



- g = fraction of time pipeline is filled
- 1-g = fraction of time pipeline is not filled (stalled)

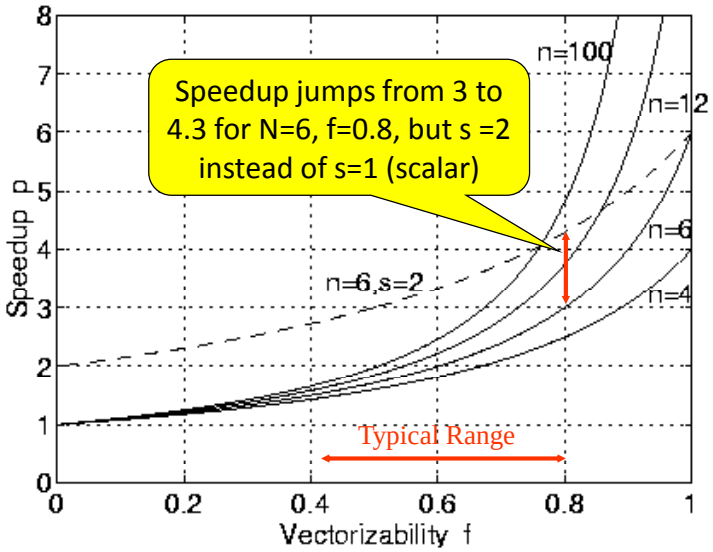
Pipelined Processor Performance Model



- "Tyranny of Amdahl's Law"
  - When g is even slightly below 100%, a big performance hit will result
  - Stalled cycles in the pipeline are the key adversary and must be minimized as much as possible
  - Can we somehow fill the pipeline bubbles (stalled cycles)?

[Tilak Agerwala and John Cocke, 1987]

Motivation for Superscalar Design



Superscalar Proposal

- Moderate the tyranny of Amdahl's Law
  - Ease the sequential bottleneck
  - More generally applicable
  - Robust (less sensitive to f)
  - Revised Amdahl's Law:

$$Speedup = \frac{1}{\frac{(1-f)}{S} + \frac{f}{N}}$$

The Ideas Behind Modern Superscalar Processors

- Superscalar or wide instruction issue
  Ideal IPC = n (CPI = 1/n)
- Diversified pipelines
  Different instructions go through different pipe stages
  Instructions go through needed stages only
- Out-of-order or data-flow execution
  Stall only on RAW hazards and structural hazards
- Speculation
  Overcome (some) RAW hazards through prediction

And it all relies on: **Instruction Level Parallelism (ILP)**
Independent instructions within sequential programs

C. Limits on Instruction Level Parallelism (ILP)

Weiss and Smith [1984]	1.58
Sohi and Vajapeyam [1987]	1.81
Tjaden and Flynn [1970]	1.86 (Flynn's bottleneck)
Tjaden and Flynn [1973]	1.96
Uht [1986]	2.00
Smith et al. [1989]	2.00
Jouppi and Wall [1988]	2.40
Johnson [1991]	2.50
Acosta et al. [1986]	2.79
Wedig [1982]	3.00
Butler et al. [1991]	5.8
Melvin and Patt [1991]	6
Wall [1991]	7 (Jouppi disagreed)
Kuck et al. [1972]	8
Riseman and Foster [1972]	51 (no control dependences)
Nicolau and Fisher [1984]	90 (Fisher's optimism)

Processor Performance

$$1/\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

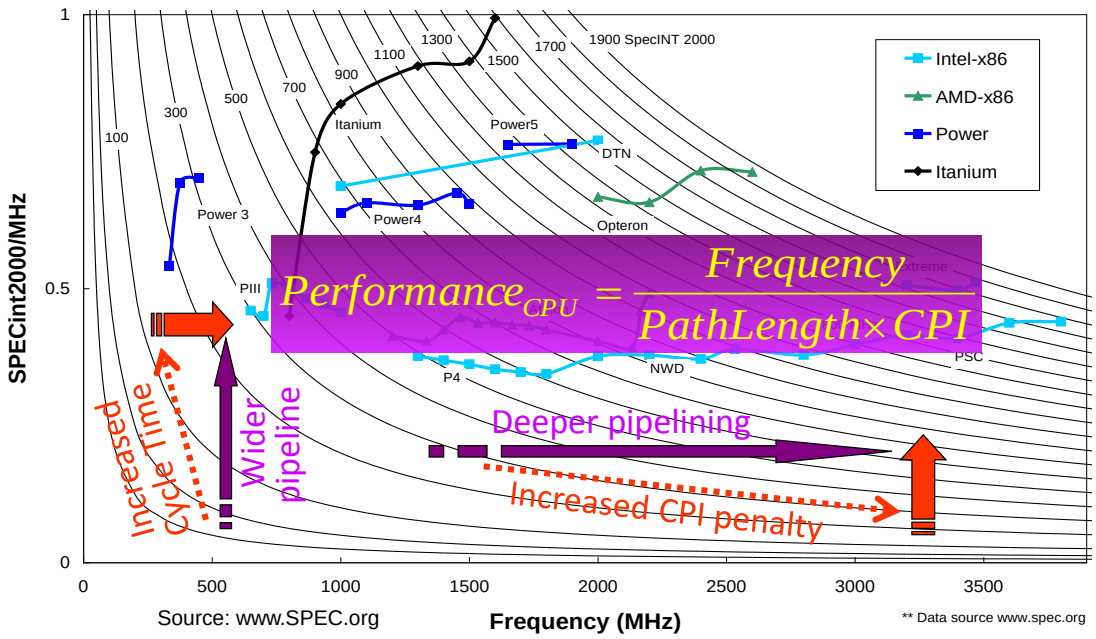
$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(path length) (CPI) (cycle time)

- In the 1980's (decade of **pipelining**):
  - CPI: 5.0 → 1.15
- In the 1990's (decade of **superscalar**):
  - CPI: 1.15 → 0.5 (best case)
- In the 2000's (decade of **multicore**):
  - Core CPI unchanged; chip CPI scales with #cores

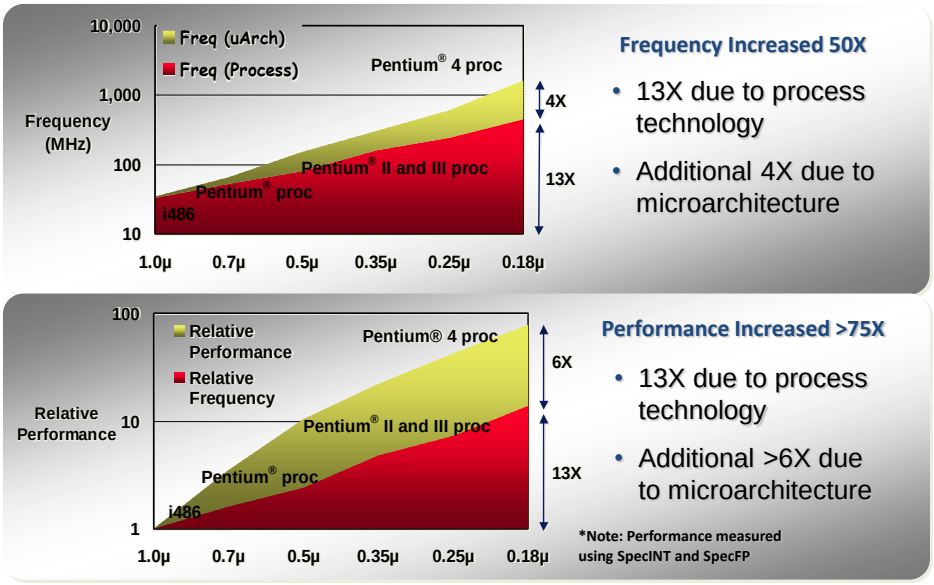
Frequency vs. Parallelism

[John DeVale & Bryan Black, 2005]



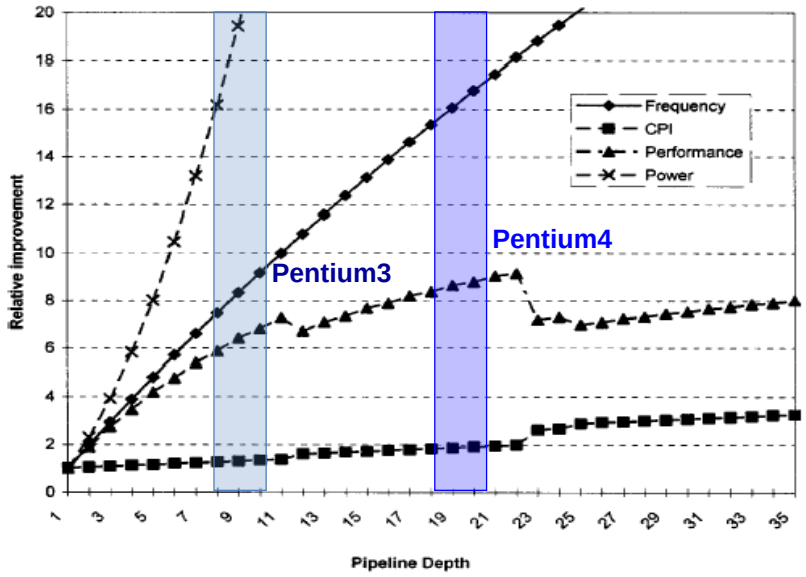
[Source: Intel Corporation]

Frequency and Performance Boost



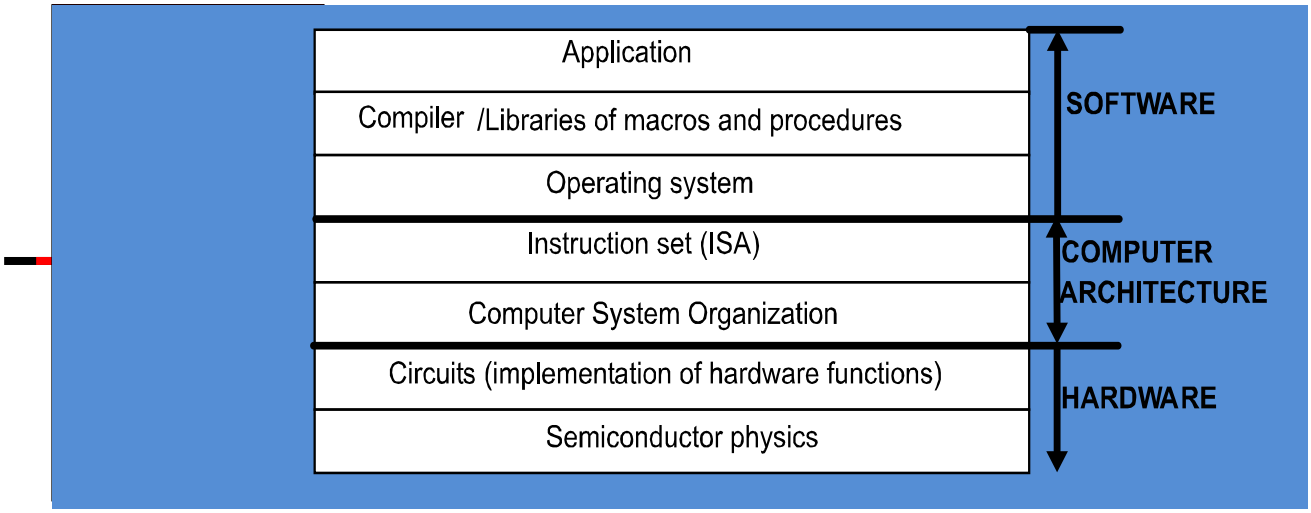
[Ed Grochowski, Intel, 1997]

Putting it All Together: Limits to Deeper Pipelines



[Hsien-Hsin Sean Lee, GIT 2007] [Xuxian Jiang, NCSU 2009]

D. Computer Architecture and Operating Systems



DSI = ISA = a contract between the program and the machine.

[Hsien-Hsin Sean Lee, 2007]

What is an Operating System?

- An intermediate program between a user of a computer and the computer hardware (to hide messy details)
- Goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient and efficient to use

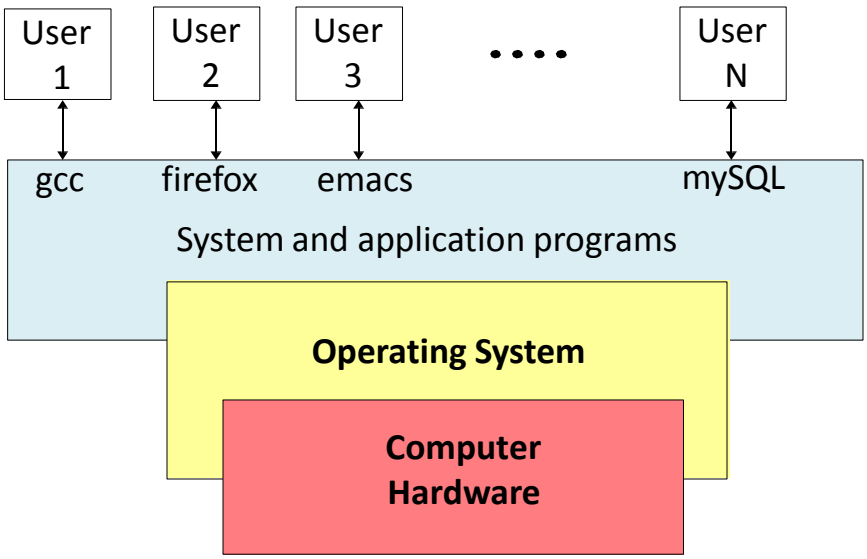
PwrPoint	Gem5	IE
Compiler	Editors	Shell
Operating System		
Instruction Set Architecture		
Microarchitecture		
Physical Devices		

System Program

Computer System Components

- Hardware
  - Provides basic computing resources (CPU, memory, I/O)
- Operating System
  - Controls and coordinates the use of the hardware among various application programs for various users
- Application Programs
  - Define the ways in which the system resources are used to solve the computing problems of users (e.g. database systems, 3D games, business applications)
- Users
  - People, machines, other computers

Abstract View of Computer System Components



Time-Sharing Computing Systems

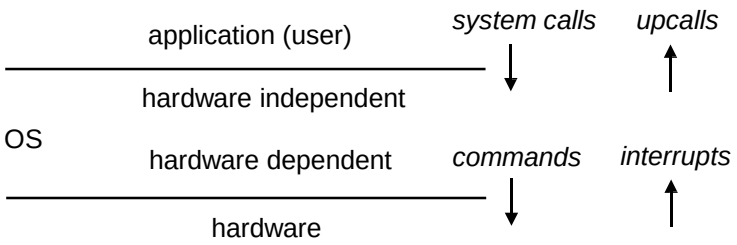
- CPU is multiplexed among several jobs that are kept in memory and on disk (The CPU is allocated to a job only if the job is in memory)
- A job swapped in and out of memory to the disk
- On-line communication between the user and the system is provided
  - When the OS finishes the execution of one command, it seeks the next "control statement" from the user's keyboard
- On-line system must be available for users to access data and code
- MIT MULTICS (MULtiplexed Information and Computing Services)
  - Ken Thompson went to Bell Labs and wrote one for a PDP-7
  - Brian Kernighan jokingly dubbed it UNICS (UNIplexed ..)
  - Later spelled to UNIX and moved to PDP-11/20
  - IEEE POSIX to standardize UNIX

Operating System Concepts

- Process Management
- Main Memory Management
- File Management
- I/O System Management
- Networking
- Protection System
- Command-Interpreter System

[Xuxian Jiang, NCSU 2009]

How Does an Operating System Work?



- Receives requests from the application: system calls
- Satisfies the requests: may issue commands to hardware
- Handles hardware interrupts: may upcall the application

- Abstraction
  - Process, memory, I/O, file, socket, …
- Tradeoff
  - Separation between mechanisms and policies

Abstraction I: Process

A process is a system abstraction:
illusion of being the only job in the system

user:	*application*
OS:	*process*
hardware:	*computer*

- Mechanism:
  - Creation, destruction, suspension, context switch, signalling, IPC, etc.
- Policy:
  - How to share system resources between multiple processes?

Abstraction II: Thread

A thread is a processor abstraction: illusion of having 1 processor per execution context

application:	*execution context*
OS:	*thread*
hardware:	*processor*

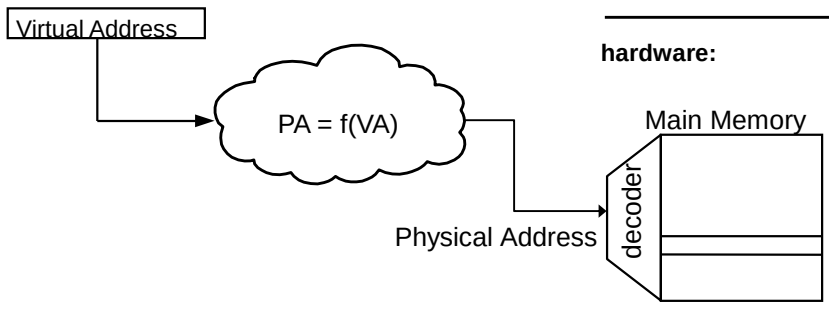
Process vs. Thread: Process is the unit of resource ownership, while Thread is the unit of instruction execution.

- Mechanism:
  - Creation, destruction, suspension, context switch, signalling, synchronization, etc.
- Policy:
  - How to share the CPU between threads from different processes?
  - How to share the CPU between threads from the same process?

Abstraction III: Virtual Memory

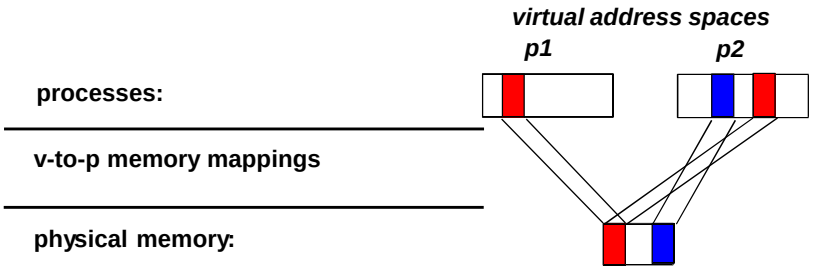
Virtual memory is a memory abstraction: illusion of large contiguous memory, often more memory than physically available

application:	***address space***
OS:	***virtual memory***
hardware:	***physical memory***



Virtual Memory Mechanism and Policy

- Mechanism:
  - Virtual-to-physical memory mapping, page-fault, etc.



- Policy:
  - How to multiplex a virtual memory that is larger than the physical memory onto what is available?
  - How to share physical memory between multiple processes?

Abstraction IV: File System

A file system is a storage abstraction:
illusion of structured storage space

application/user:	*copy file1 file2*
OS:	*files, directories*
hardware:	*disk*

- Mechanism:
  - File creation, deletion, read, write, file-block- to-disk-block mapping, file buffer cache, etc.
- Policy:
  - Sharing vs. protection?
  - Which block to allocate for new data?
  - File buffer cache management?

Abstraction V: Messaging

Message passing is a communication abstraction:
illusion of reliable (sometimes ordered) transport

application:	***sockets***
OS:	***TCP/IP protocols***
hardware:	***network interface***

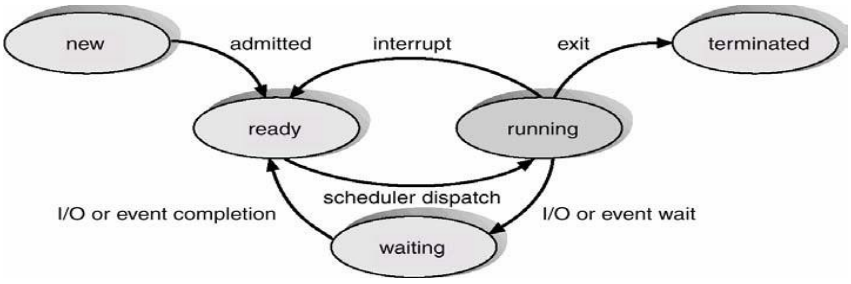
- Mechanism:
  - Send, receive, buffering, retransmission, etc.
- Policy:
  - Congestion control and routing
  - Multiplexing multiple connections onto a single NIC

Process Management

- A *process* is a program in execution
- A process contains
  - Address space (e.g. read-only code, global data, heap, stack, etc)
  - PC, $sp
  - Opened file handles
- A process needs certain resources, including CPU time, memory, files, and I/O devices
- The OS is responsible for the following activities for process management
  - Process creation and deletion
  - Process suspension and resumption
  - Provision of mechanisms for:
    - Process synchronization
    - Process communication

Process State

- As a process executes, it changes *state:*
  - New: The process is being created
  - Ready: The process is waiting to be assigned to a processor
  - Running: Instructions are being executed
  - Waiting: The process is waiting for some event (e.g. I/O) to occur
  - Terminated: The process has finished execution

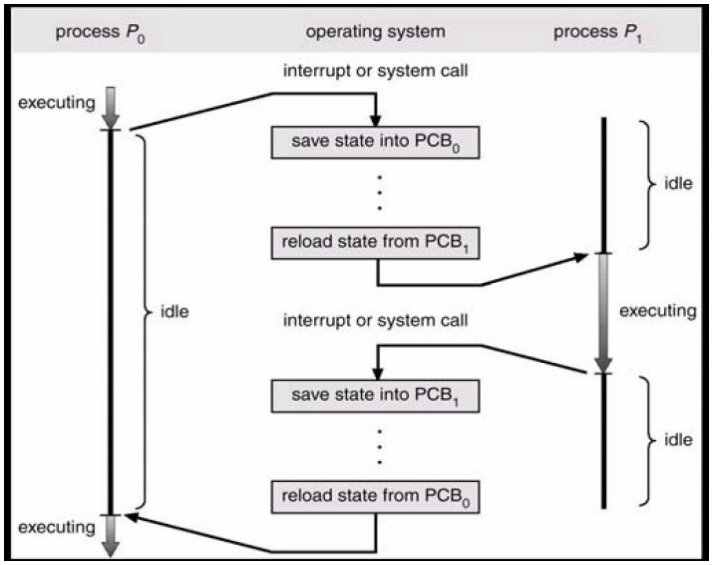


Process Control Block (PCB)

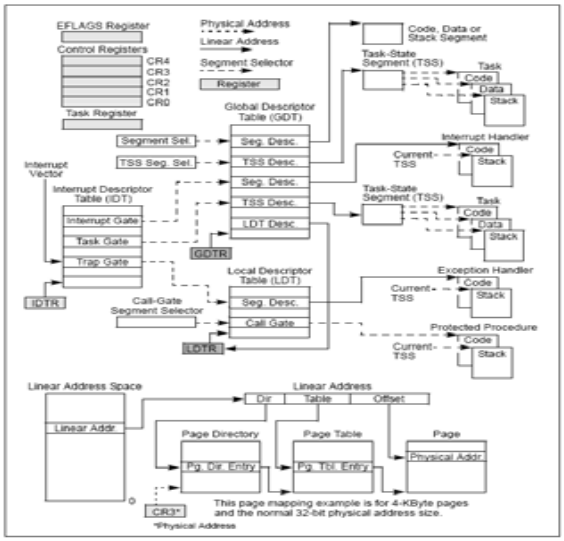
Information associated with each process:

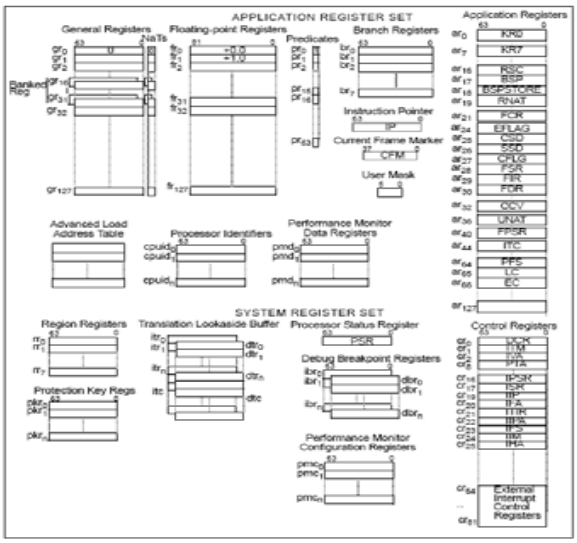
- Process state
- Program counter
- CPU registers (for context switch)
- CPU scheduling information (e.g. priority)
- Memory-management information (e.g. page table, segment table)
- Accounting information (PID, user time, constraint)
- I/O status information (list of I/O devices allocated, list of open files etc.)

CPU Switches from Process to Process



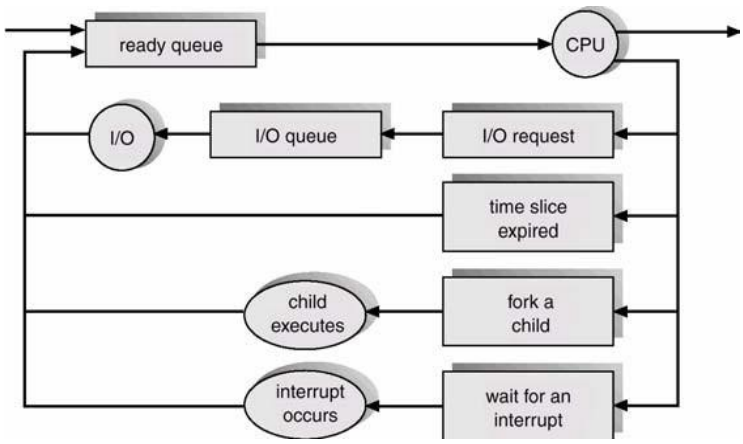
Examples of System State: Intel Core Duo & Itanium





Process Scheduling

- *Job queue* – set of all processes in the system
- *Ready queue* – set of all processes residing in main memory, ready and waiting to execute
- *Device queues* – set of processes waiting for an I/O device
- Process migration between the various queues



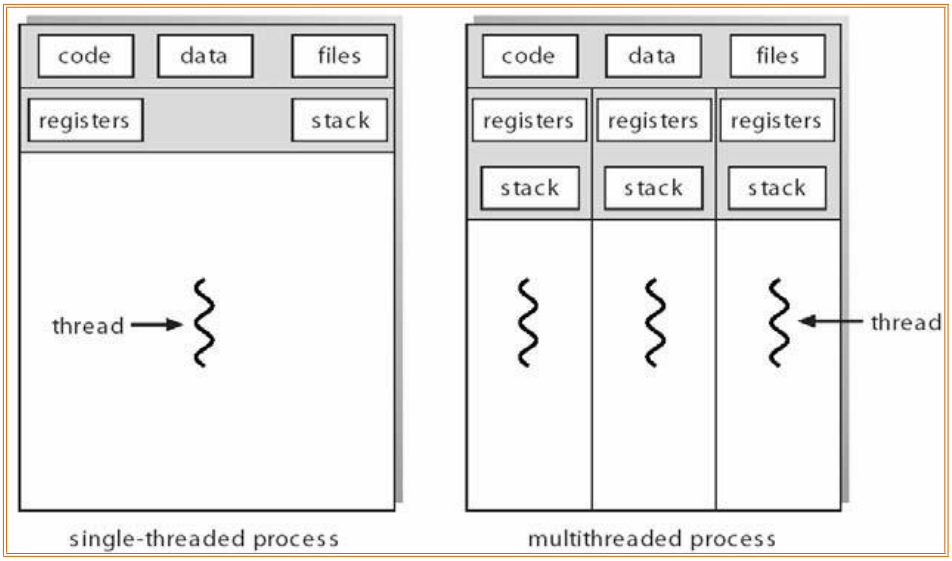
Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate
- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program

Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are de-allocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*

Single and Multi-Threaded Processes



User Threads vs. Kernel Threads

User Threads:

- Thread management done by user-level threads library
- Three primary thread libraries:
  - POSIX Pthreads
  - Java threads
  - Win32 threads

Kernel Threads:

- Supported by the Kernel
- Examples
  - Windows XP/2000
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Examples of Threads

- A web browser
  - One thread displays images
  - One thread retrieves data from network
- A word processor
  - One thread displays graphics
  - One thread reads keystrokes
  - One thread performs spell checking in the background
- A web server
  - One thread accepts requests
  - When a request comes in, separate thread is created to service
  - Many threads to support thousands of client requests
- RPC or RMI (Java)
  - One thread receives message
  - Message service uses another thread

Threads vs. Processes

Threads:

- A thread has no data segment or heap
- A thread cannot live on its own, it must live within a process
- There can be more than one thread in a process, the first thread calls main and has the process's stack
- Inexpensive creation
- Inexpensive context switching
- If a thread dies, its stack is reclaimed by the process

Processes:

- A process has code/data/heap and other segments
- There must be at least one thread in a process
- Threads within a process share code/data/heap, share I/O, but each has its own stack and registers
- Expense in creation
- Expensive context switching
- It a process dies, its resources are reclaimed and all threads die

18-640 Foundations of Computer Architecture

Lecture 4:
"Superscalar Implementation and Instruction Flow"

John Paul Shen
September 4, 2014

Next Time ...

- Required Reading Assignment:
  - **Chapter 4 of Shen and Lipasti (SnL).**
- Recommended Reference:

Electrical & Computer ENGINEERING