CPEN 411: Computer Architecture

Slide Set #2:
Fundamentals of Computer Design

Instructor: Mieszko Lis

Original Slides: Tor Aamodt

- “Chapter 1: How to Make a Lot of Money”

- “Chapter 1: How to Make a Lot of Money”
- “The major thing is avoiding the big mistake. It’s like downhill ski racing: Can you stay right on that edge beside disaster?”

- “Chapter 1: How to Make a Lot of Money”
- “The major thing is avoiding the big mistake. It’s like downhill ski racing: Can you stay right on that edge beside disaster?”
- [The Soul of a New Machine, Tracy Kidder - 1981]

AMD vs. Intel

	2Q 2006			2Q 2007			
	GM	~P/E	~SP	GM	~P/E	~SP	ΔSP
• Intel	52%	17	$18	47%	26	$26	44%
• AMD	57%	26	$24	33%	(loss)	$13	(50%)
ΔGM	5% (AMD higher)			14% (Intel higher)			

Introduction to Slide Set #2

In this set of slides we learn fundamental principles of computing that have guided the development of computer architecture for the past 30 or so years. The most important of these is the quantitative approach that gives the textbook its subtitle.

Before discussing these fundamentals, we briefly look at the interesting history of computing machines.

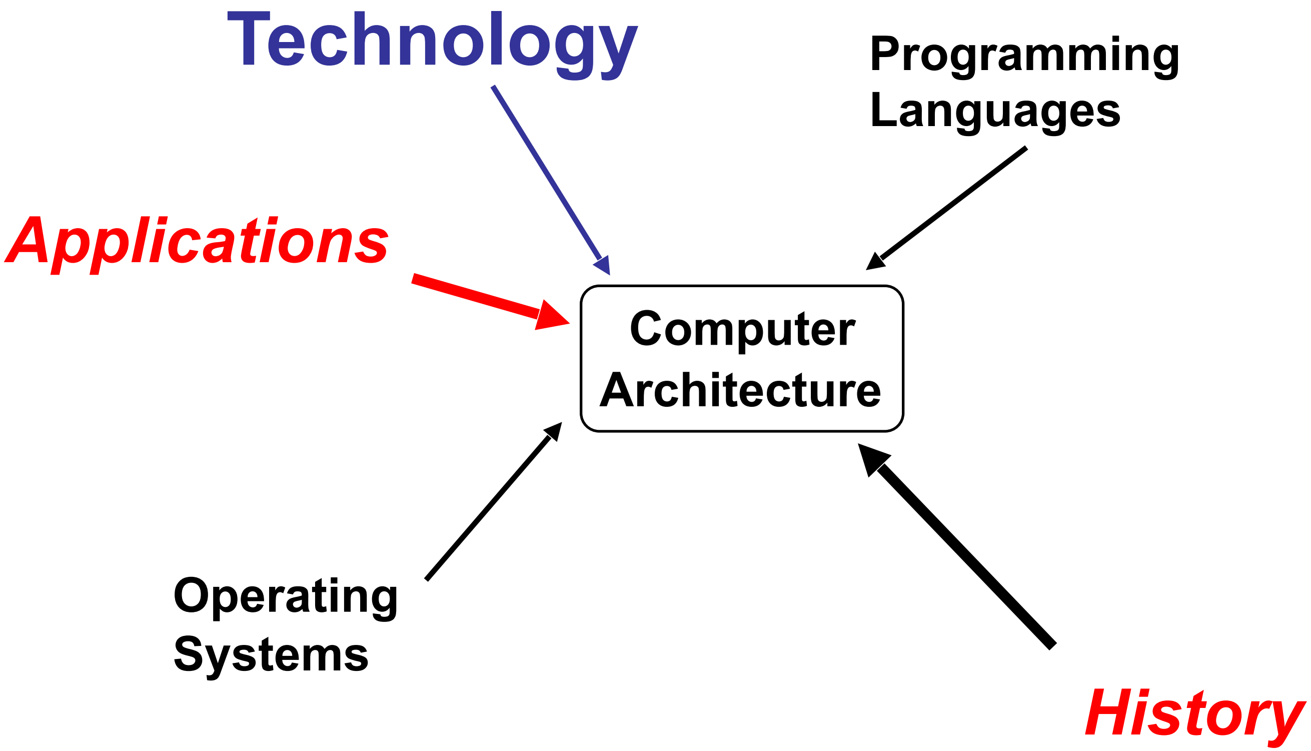
In this slide set we will also learn a bit about the MIPS64 instruction set architecture so you can get started with assignment #1.

Learning Objectives

After finishing this slide set should be able to…

1. Summarize a few historical changes in computing
2. Describe high level goals of a computer architect
3. Determine what a simple MIPS64 program computes
4. List instruction processing steps in the von Neumann architecture
5. Describe how technology trends impact latency and bandwidth
6. Analyze how active and static power change as voltage and clock frequency change.
7. Define cost, price and explain their trends
8. Demonstrate how to measure and report performance
9. Define several quantitative principles and apply them
10. Explain some common pitfalls and fallacies

Factors Affecting Development of Computer Architecture



Technology

Programming Languages

Applications

Computer Architecture

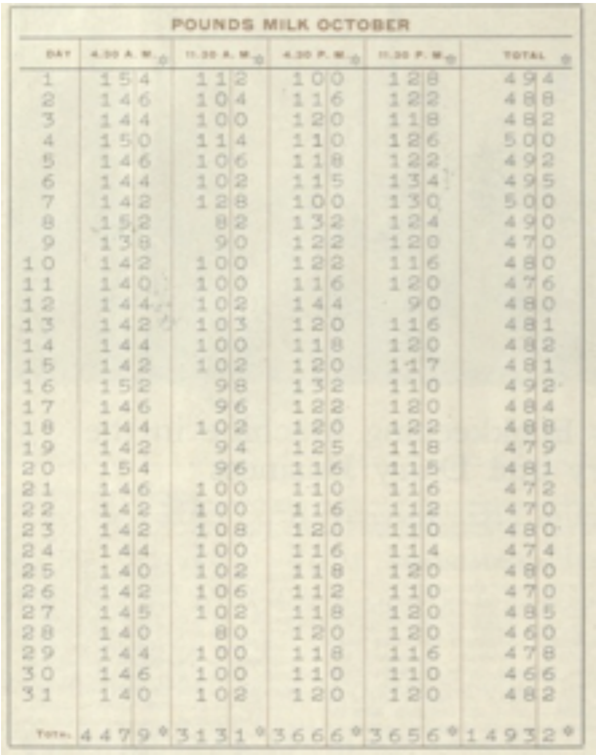
Operating Systems

History

Why look at computer history?

- Hard to imagine world without computers
- Perhaps easiest way to get sense of current impact (and potential change in future) is to look back and consider what computers were like in the past

State of the Art in Business Computing …85 years ago



DAY	4.30 A.M.	11.30 A.M.	4.30 P.M.	11.30 P.M.	TOTAL
1	154	112	100	128	494
2	146	104	116	122	488
3	144	100	120	118	482
4	150	114	110	126	500
5	146	106	118	122	492
6	144	102	115	134	495
7	142	128	100	130	500
8	152	82	132	124	490
9	138	90	122	120	470
10	142	100	122	116	480
11	140	100	116	120	476
12	144	102	144	90	480
13	142	103	120	116	481
14	144	100	118	120	482
15	142	102	120	117	481
16	152	98	132	110	492
17	146	96	122	120	484
18	144	102	120	122	488
19	142	94	125	118	479
20	154	96	116	115	481
21	146	100	110	116	472
22	142	100	116	112	470
23	142	108	120	110	480
24	144	100	116	114	474
25	140	102	118	120	480
26	142	106	112	110	470
27	145	102	118	120	485
28	140	80	120	120	460
29	144	100	118	116	478
30	146	100	110	110	466
31	140	102	120	120	482
TOTAL	4479	3131	3666	3656	14932

- Burroughs adding machines (# in use grows from 286 in 1895 to 750,000 in 1925)
- Addition and subtraction only

Schickard Calculator (1623)

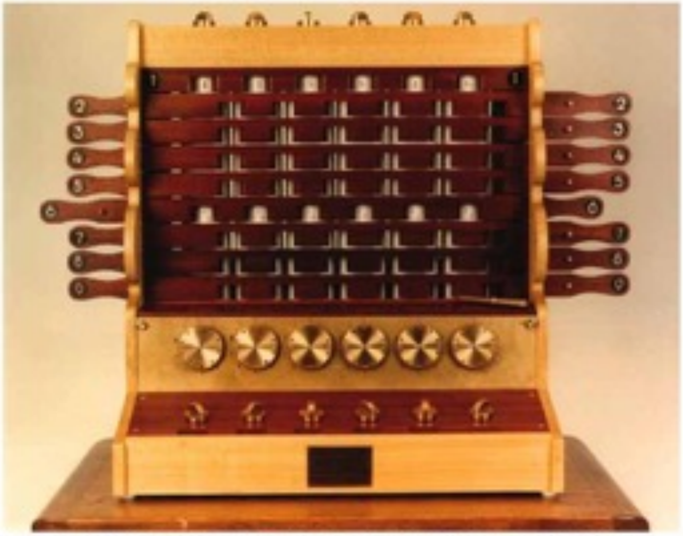
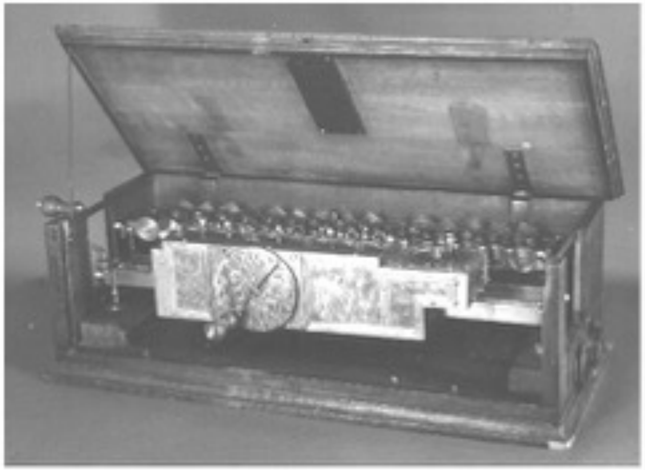


Figure 1. The Wilhelm Schickard calculator of 1623–1624, showing a front view from an elevated position. The middle section is the adding machine with the result windows. The adding machine's base contains dials, which are used to enter the partial products that are taken from the multiplication unit above. Notice the stylus on the right-hand side, which is used to operate the calculator. The calculator's pedestal shows the memory unit's dials. (Photo courtesy of IBM Germany.)

- Described in 1623-24 letters from Wihelm Schickard to Johannes Keplar (astronomer)
- Performed Multiplication, Addition, Subtraction
- Lost in fire while being constructed.
- Schickard dies of plague in 30 Years War and invention is forgotten until recently (letters discovered in 1957)

Leibnez's Calculator (1673)

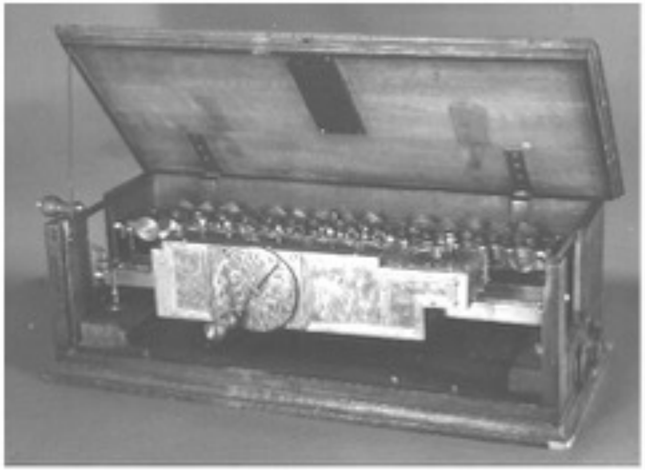


- Addition, Subtraction, Multiplication and Division
- Predecessor of mechanical "Desk Calculator" used in WW II
- Leibnez: "it is unworthy of excellent men [and women] to lose hours like slaves in the labor of calculation which could safely be relegated to anyone else if machines were used."
- Leibnez also helped initiate development of symbolic logic (as well as developing calculus)

Leibnez's C

Does one need a hardware multiplication mechanism to compute N^2+N+41?

A: Yes
B: No
C: Not sure

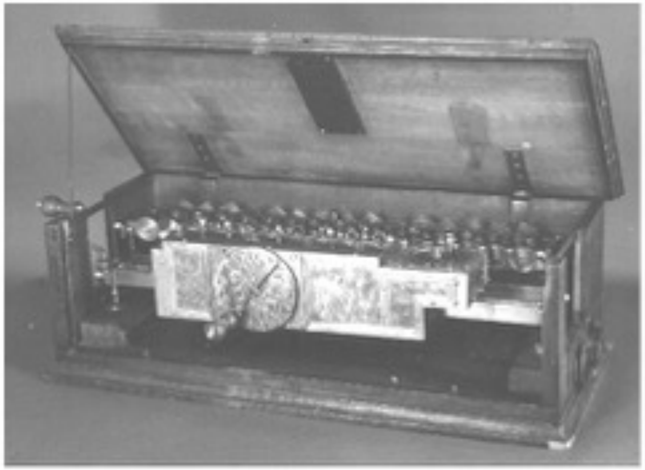


- Predecessor of mechanical "Desk Calculator" used in WW II
- Leibnez: "it is unworthy of excellent men [and women] to lose hours like slaves in the labor of calculation which could safely be relegated to anyone else if machines were used."
- Leibnez also helped initiate development of symbolic logic (as well as developing calculus)

Leibnez's C

Does one need a hardware multiplication mechanism to compute N^2+N+41?

A: Yes
B: No ✔
C: Not sure



- Predecessor of mechanical "Desk Calculator" used in WW II
- Leibnez: "it is unworthy of excellent men [and women] to lose hours like slaves in the labor of calculation which could safely be relegated to anyone else if machines were used."
- Leibnez also helped initiate development of symbolic logic (as well as developing calculus)

Babbage's Difference Engine (1822)

- Babbage was a Cambridge Professor (held same position as Newton and Hawking)
- Early instance of a scientist asking for government research funding. Funding "cut" by British Government after lack of progress and harsh comments from astronomer. Later completed in Sweden in 1850's by Pehr Georg Scheutz with funding from Swedish Government.
- Computed polynomials up to 6th degree
- 44 calculations per minute (not much faster than a human).
- Difference engine idea proposed by J.H. Muller in 1786

Babbage's Difference Engine (1822)

Example:

N	N^2+N+41	D_1	D_2
0	41		
1	43	2	
2	47	4	2
3	53	6	2
4	61	8	2

- Babbage was a Cambridge Professor (held same position as Newton and Hawking)
- Early instance of a scientist asking for government research funding. Funding "cut" by British Government after lack of progress and harsh comments from astronomer. Later completed in Sweden in 1850's by Pehr Georg Scheutz with funding from Swedish Government.
- Computed polynomials up to 6th degree
- 44 calculations per minute (not much faster than a human).
- Difference engine idea proposed by J.H. Muller in 1786

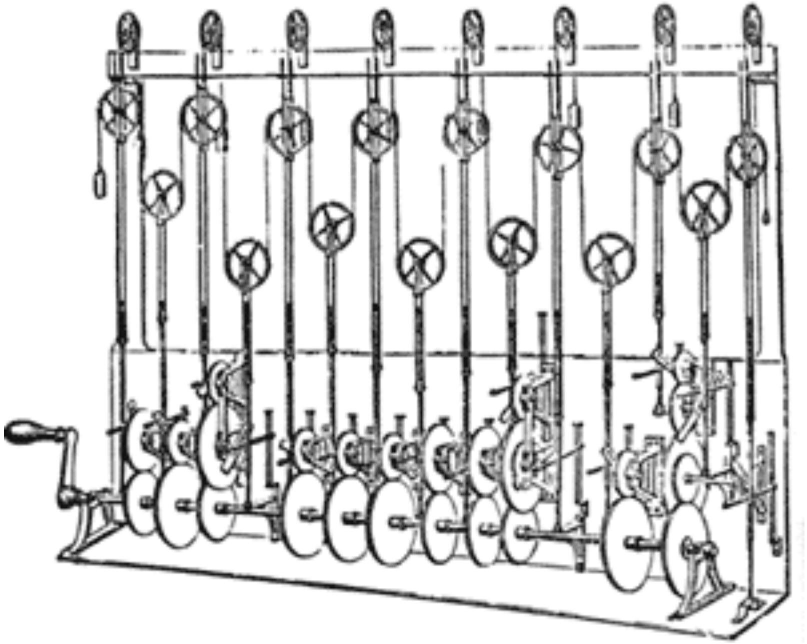
Babbage's Analytical Engine (1833-1871)

- Conceived in 1833 during break in development of the difference engine.
- Only partially constructed due to lack of financial backing (Babbage worked on design until his death in 1871).
- Inspired by Jacquard Loom (invented 1805); uses punch cards to program.
- Fully programmable, mechanical computing device. Included branch instructions, microprogram control, limited pipelining.
- Design called for two parts:
  1. "Store" (1000 x 50 decimal digits of memory)
  2. "Mill" (ALU)
- Multiply takes ~2 minutes; add ~3 seconds.

Ancient History?

- Only 14 years after Babbage’s death first “skyscrapper” built (Home Insurance Building in Chicago in 1885)

Analog Computers



- Translate mathematical problem into a physical system that matches the mathematics.
- Advantage: Much faster than mechanical digital computer/ calculator
- Disadvantage: Poor accuracy (tolerable for early physics research, but not for astronomy)

Earliest Electronic Computers

1946 ENIAC

- Univ. of Pennsylvania
- 18,000 vacuum tubes
- 30 tons, 80’ x 8.5’
- 5,000 operations per second

1949 EDSAC

- Cambridge University
- 714 operations per second
- Stored-program computer
- Uses subroutines

Earliest Electronic Computers

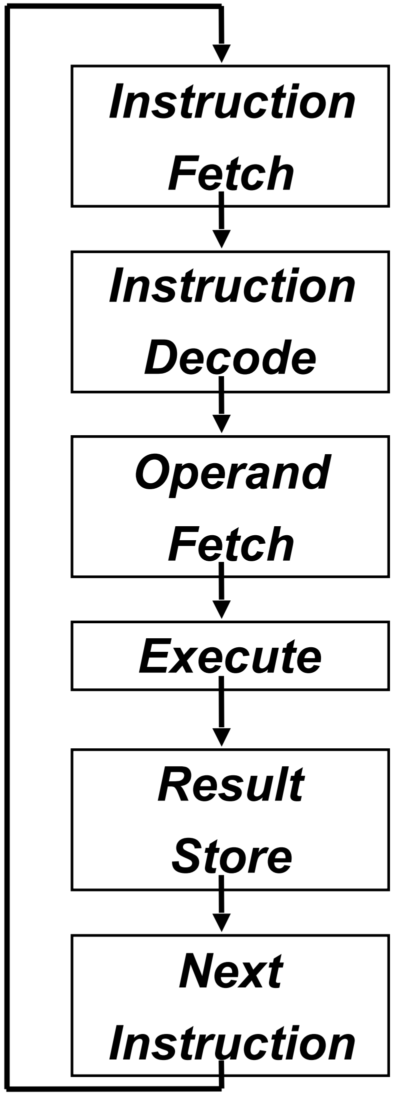
1946 ENIAC

- Univ. of Pennsylvania
- 18,000 vacuum tubes
- 30 tons, 80’ x 8.5’
- 5,000 operations per second

1949 EDSAC

- Cambridge University
- 714 operations per second
- Stored-program computer
- Uses subroutines

Fundamental Execution Cycle



Obtain instruction from program storage

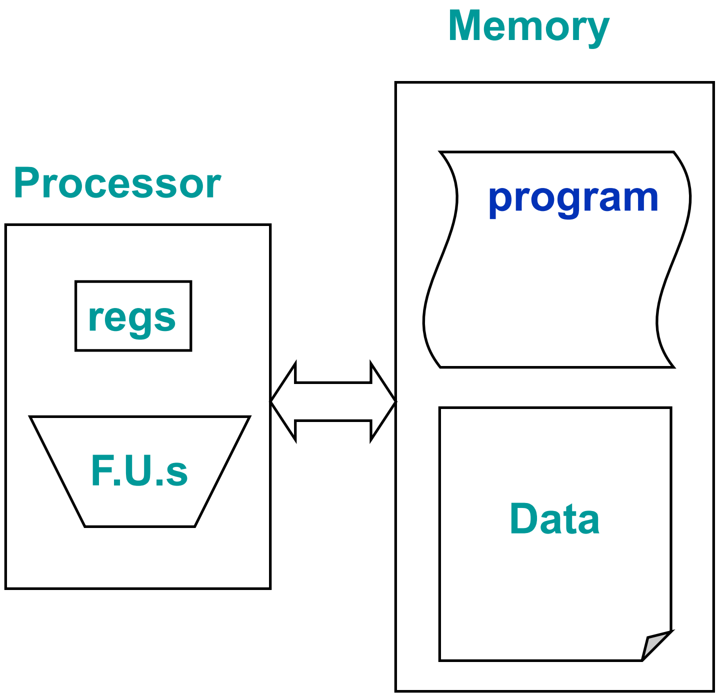
Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use

Determine successor instruction



von Neuman Architecture

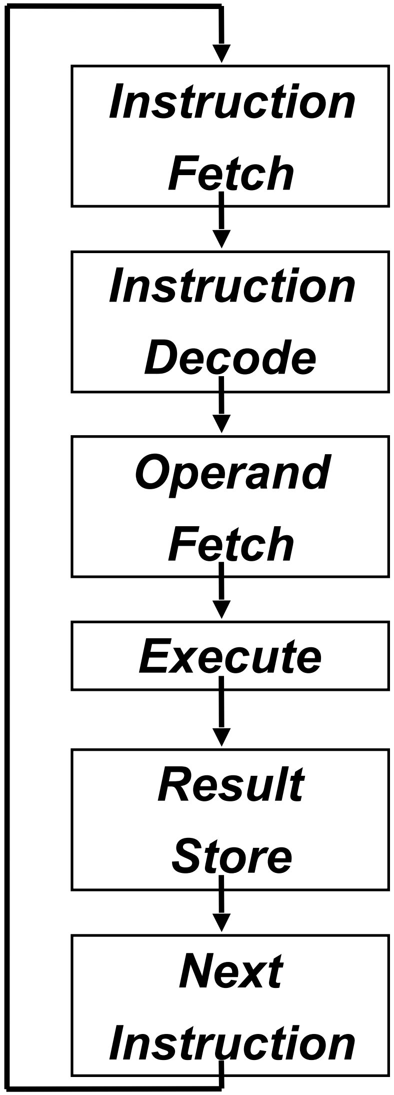
(stored program computer)

This is “sim_main()” in your Programming Assignment #1:

Fundamental E

Does “instruction fetch” of next instruction depend upon result of last instruction? (Pick “best” answer.)

A: Yes
B: No
C: Maybe
D: Not sure



Instruction Fetch

Instruction Decode

Operand Fetch

Execute

Result Store

Next Instruction

Obtain instruction fro program storage

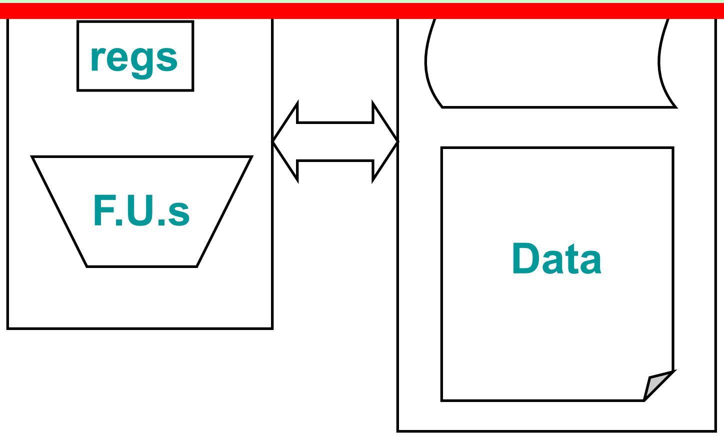
Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use

Determine successor instruction



von Neuman Architecture

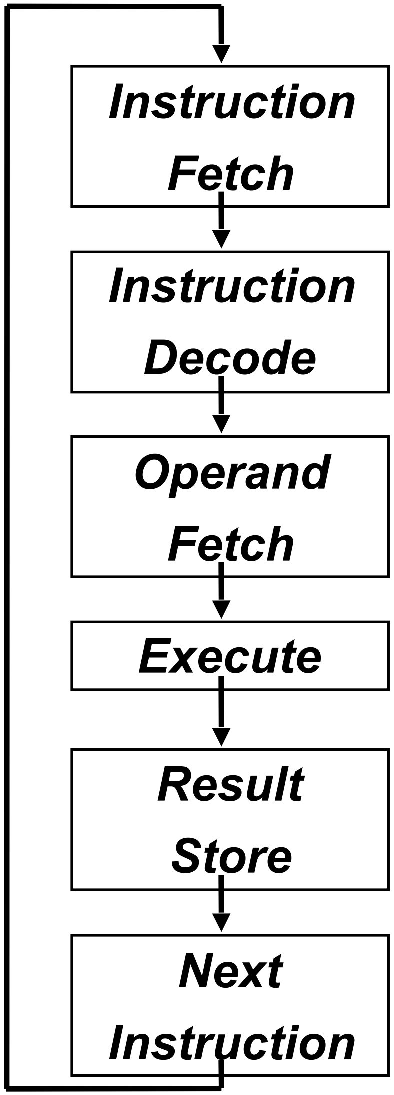
(stored program computer)

This is “sim_main()” in your Programming Assignment #1:

Fundamental E

Does “instruction fetch” of next instruction depend upon result of last instruction? (Pick “best” answer.)

A: Yes
B: No
C: Maybe
D: Not sure



Instruction Fetch

Instruction Decode

Operand Fetch

Execute

Result Store

Next Instruction

Obtain instruction fro program storage

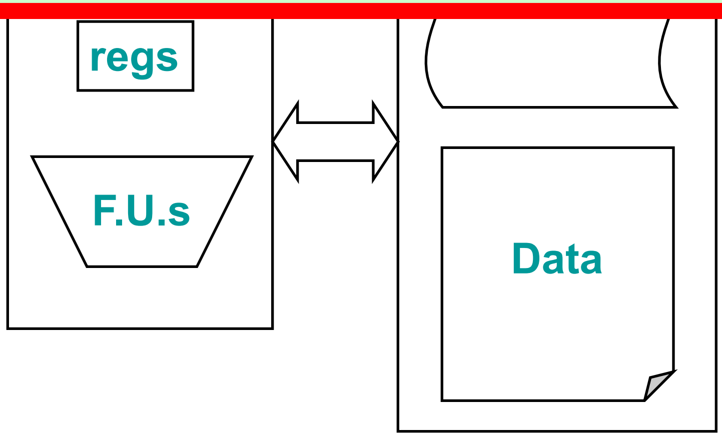
Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use

Determine successor instruction



von Neuman Architecture

(stored program computer)

This is “sim_main()” in your Programming Assignment #1:

Key Enablers

- Recognition that computers process information, not just numbers
- Decreasing cost of components used to build computers

1951 UNIVAC-1
- Remington-Rand
- First commercial computer @ $1,000,000 each
- Sold 48 systems
- 1,905 operations per second

1952 IBM 701
- First IBM computer
- Sold 19 systems

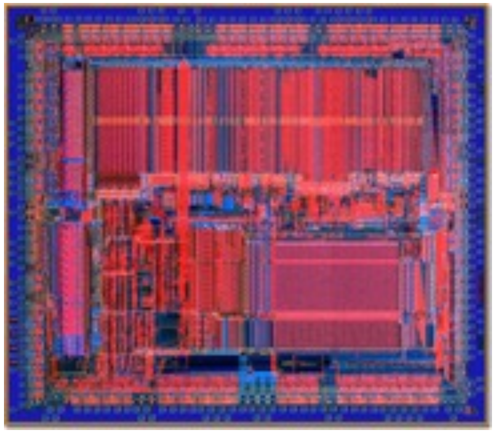
1964 IBM System/360
- First computer family, all use same instructions

1965 DEC PDP-8
- First minicomputer
- Spawns MULTICS, UNIX, C

Busicom* Calculator

1971 Intel 4004
- First microprocessor (used in calculator)
- 2,300 transistors

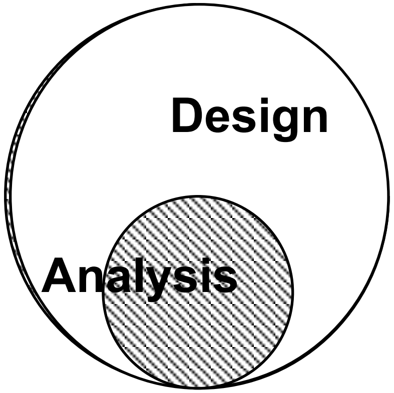
- 1977 VAX 11/780
  - Designed to increase address space from PDP-11 (VAX = Virtual Address eXtension)
  - Clock frequency 5 MHz. Surprising result [published in 1984]: Measured to run 0.5 million instructions per second when everyone thought machine performed 1 million instructions per second. One of first instances of quantitative analysis of computer design.



- 1985 MIPS R2000
  - Quantitative approach to computer design.
  - Focus on compiler effectiveness
  - Performance begins improving at ~50% per year.

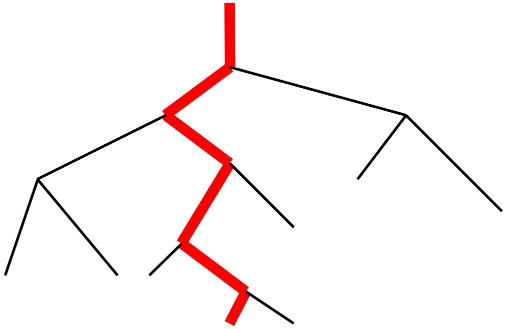
Measurement and Evaluation

Architecture design is an iterative process
-- searching the space of possible designs
-- at all levels of computer systems

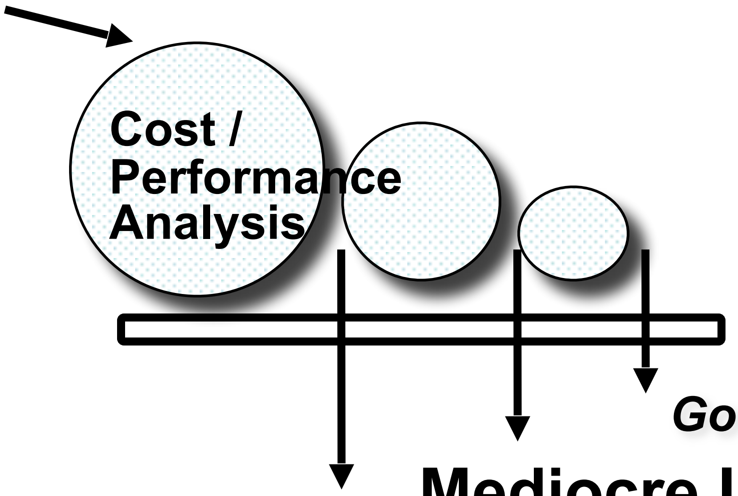


Design

Analysis



Creativity



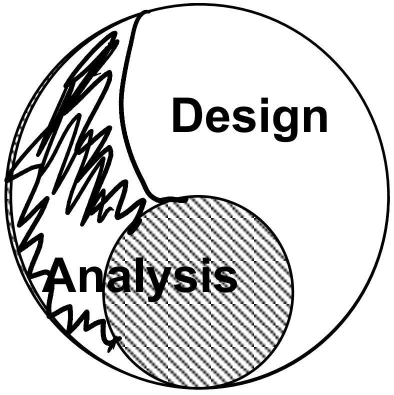
Cost / Performance Analysis

Good Ideas

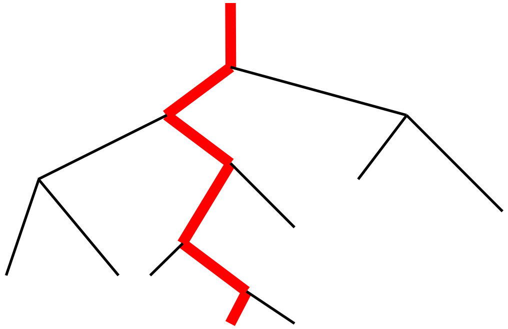
Mediocre Ideas

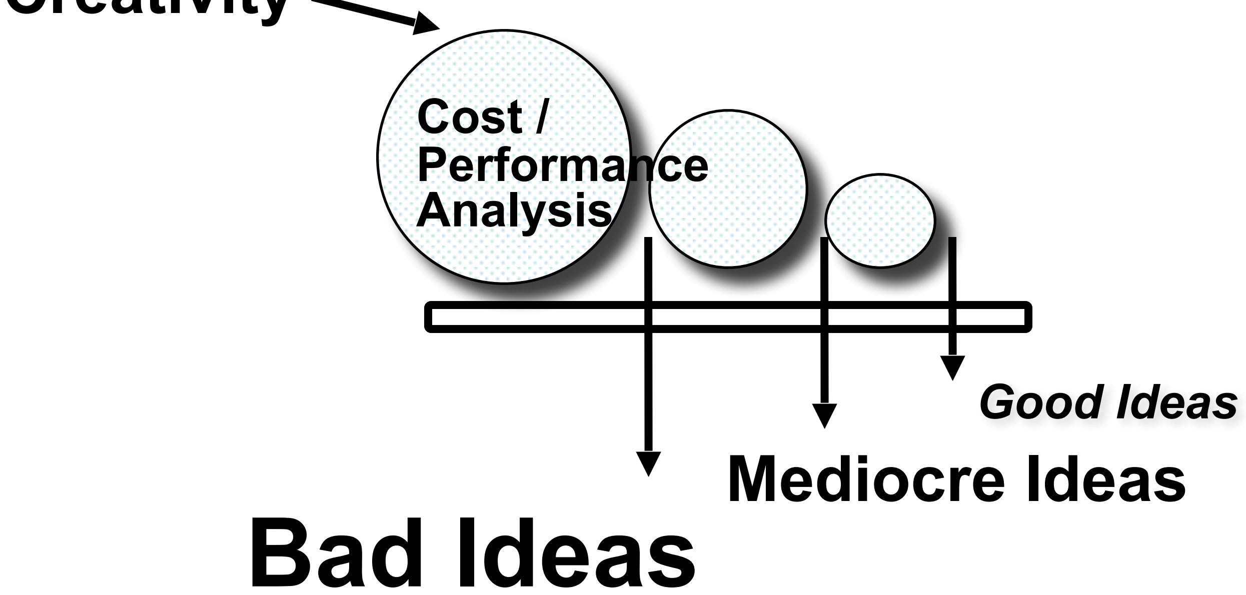
Bad Ideas

Measurement and Evaluation

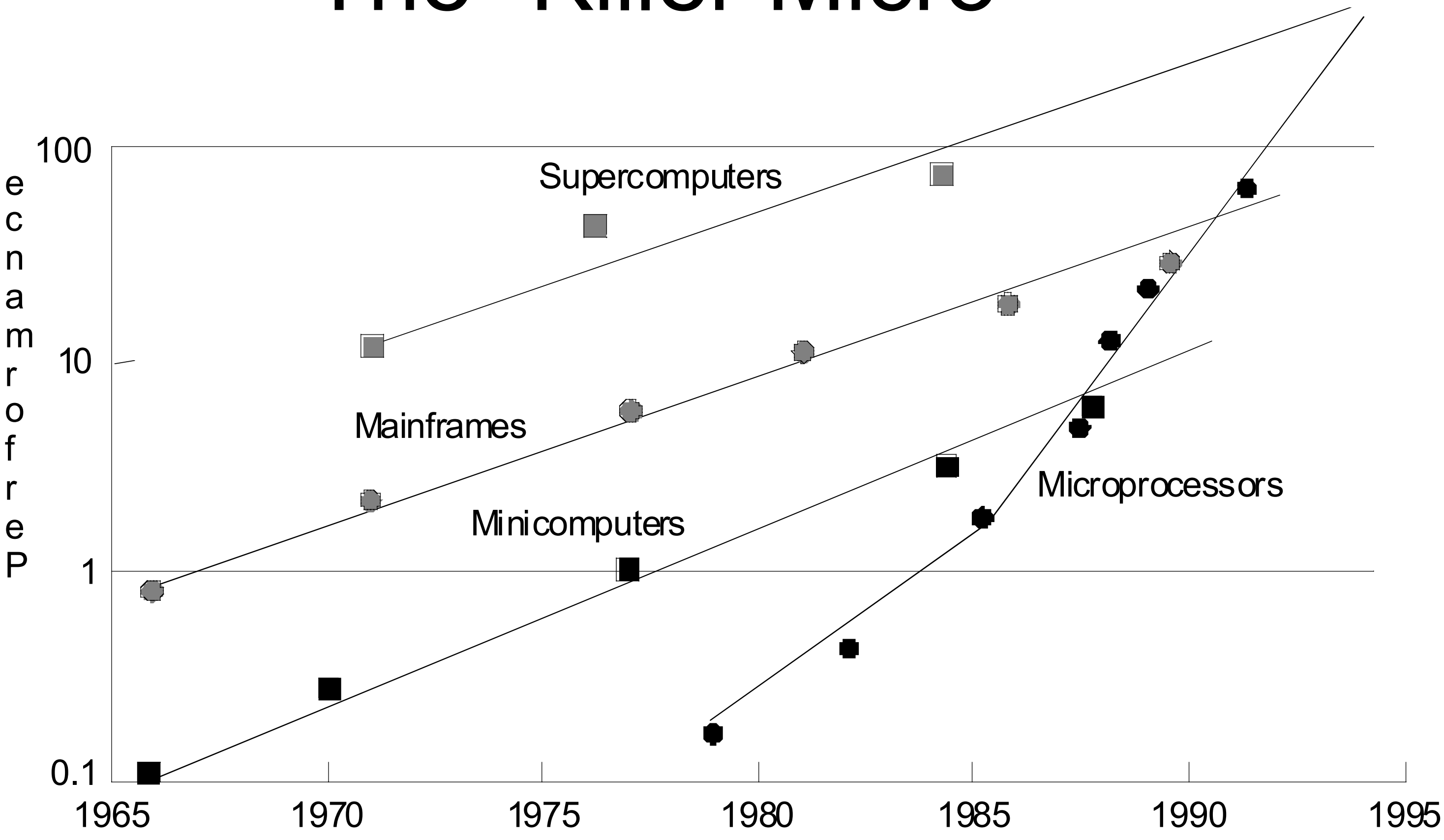


Architecture design is an iterative process
-- searching the space of possible designs
-- at all levels of computer systems





The “Killer Micro”



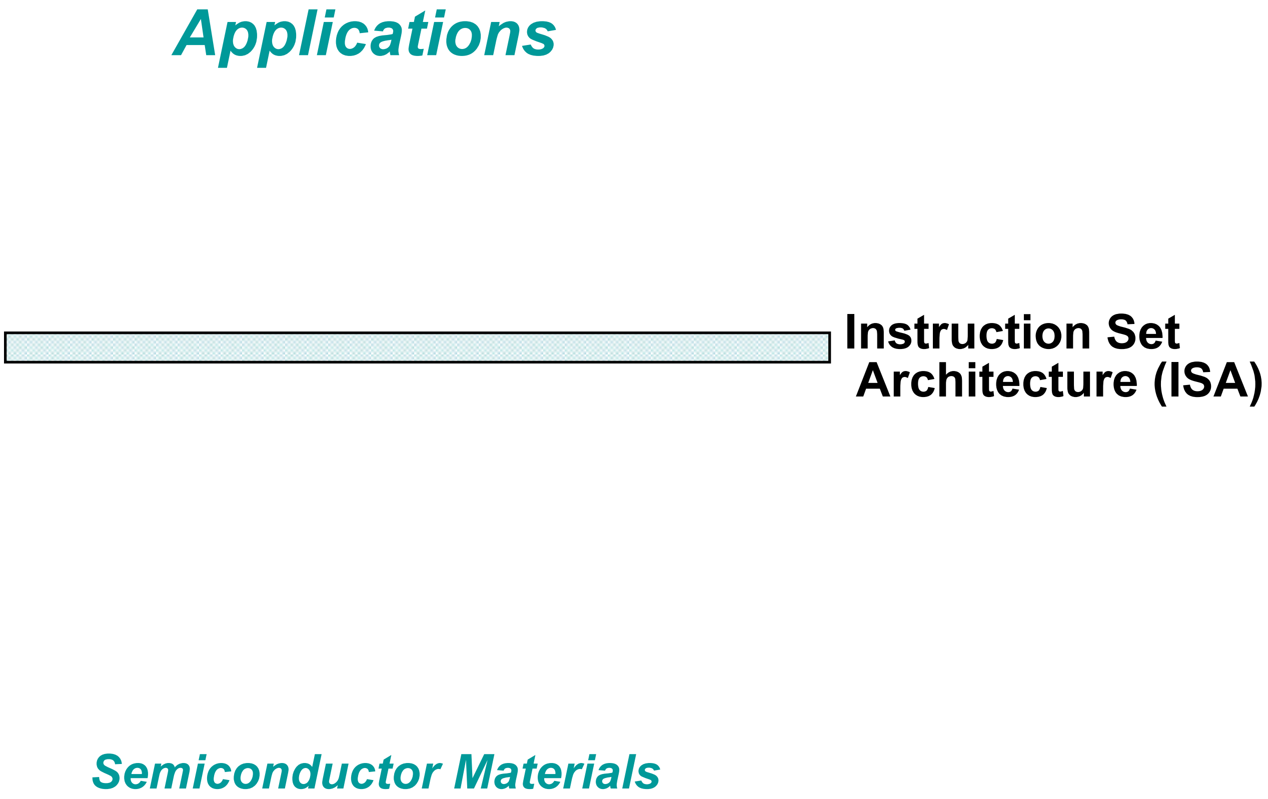
- Microprocessors “killed off” more specialized forms of computers due to cost advantages.
- Instance of “Disruptive Technology”

Task of the Computer Designer

Applications

Semiconductor Materials

Task of the Computer Designer



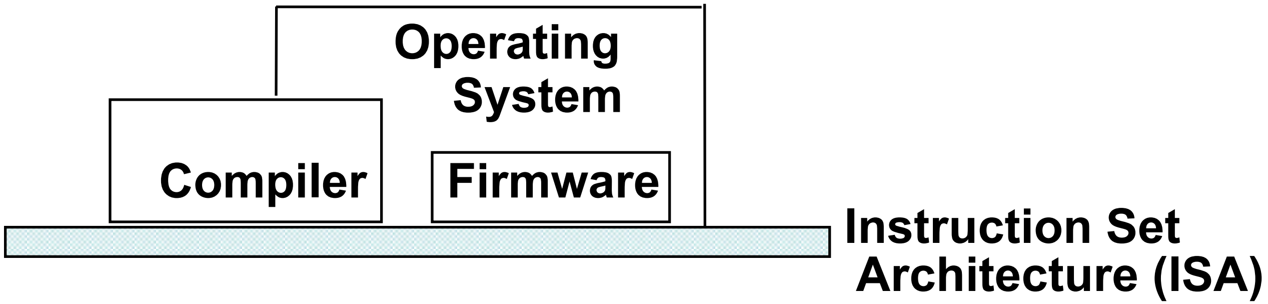
Applications

Instruction Set Architecture (ISA)

Semiconductor Materials

Task of the Computer Designer

Applications



Operating System

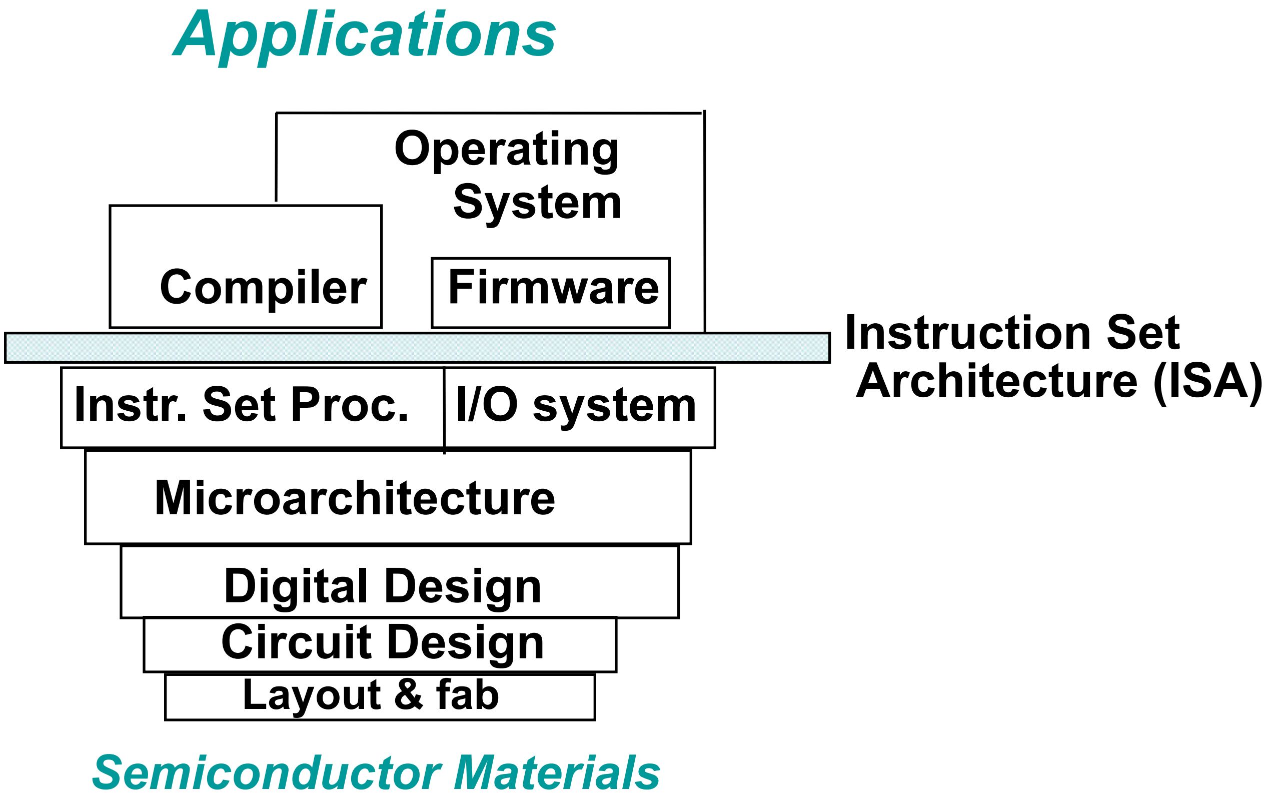
Compiler

Firmware

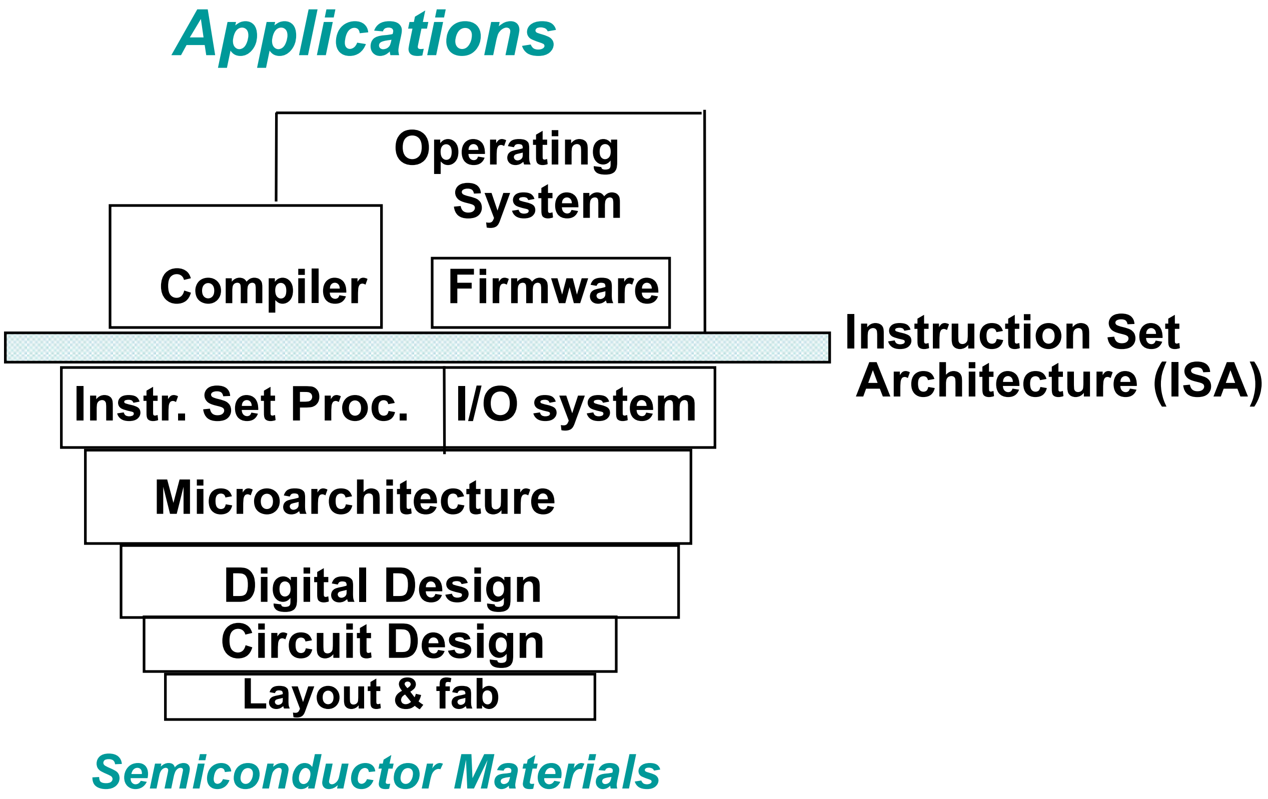
Instruction Set Architecture (ISA)

Semiconductor Materials

Task of the Computer Designer

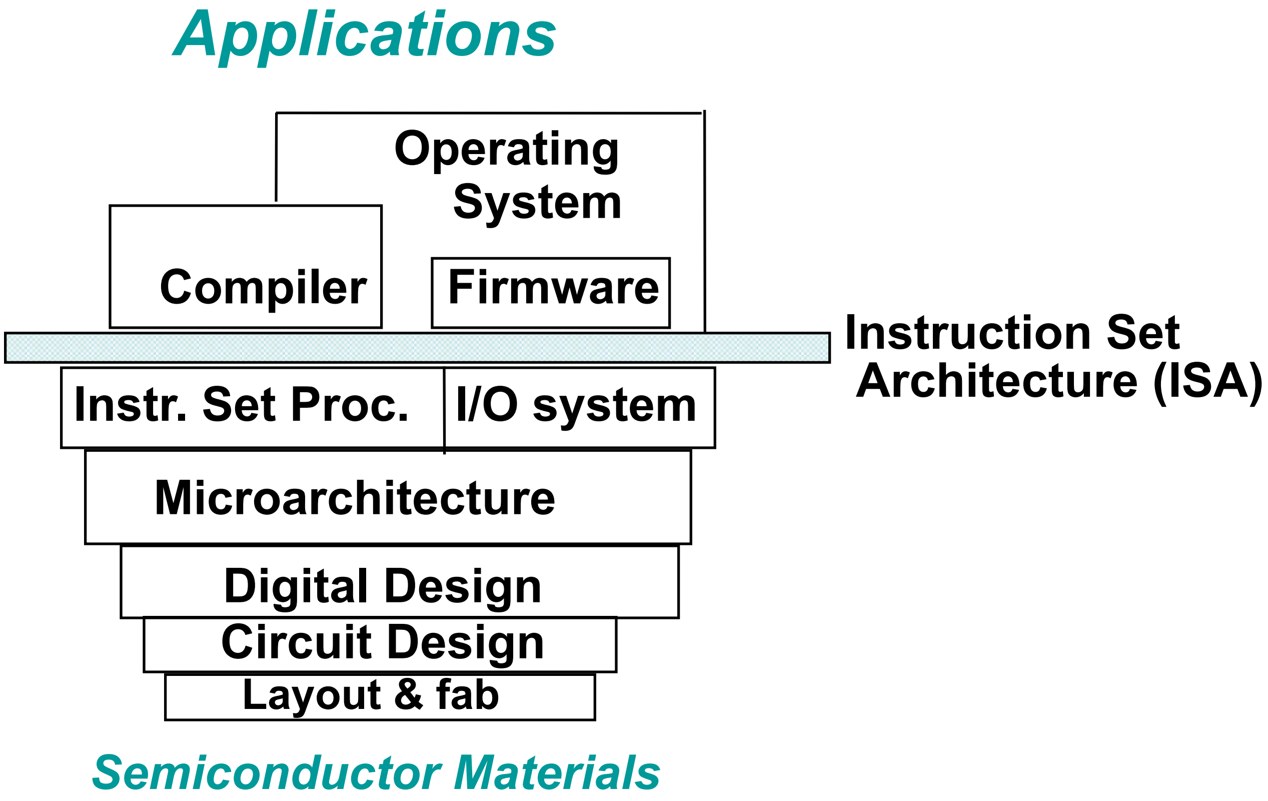


Task of the Computer Designer



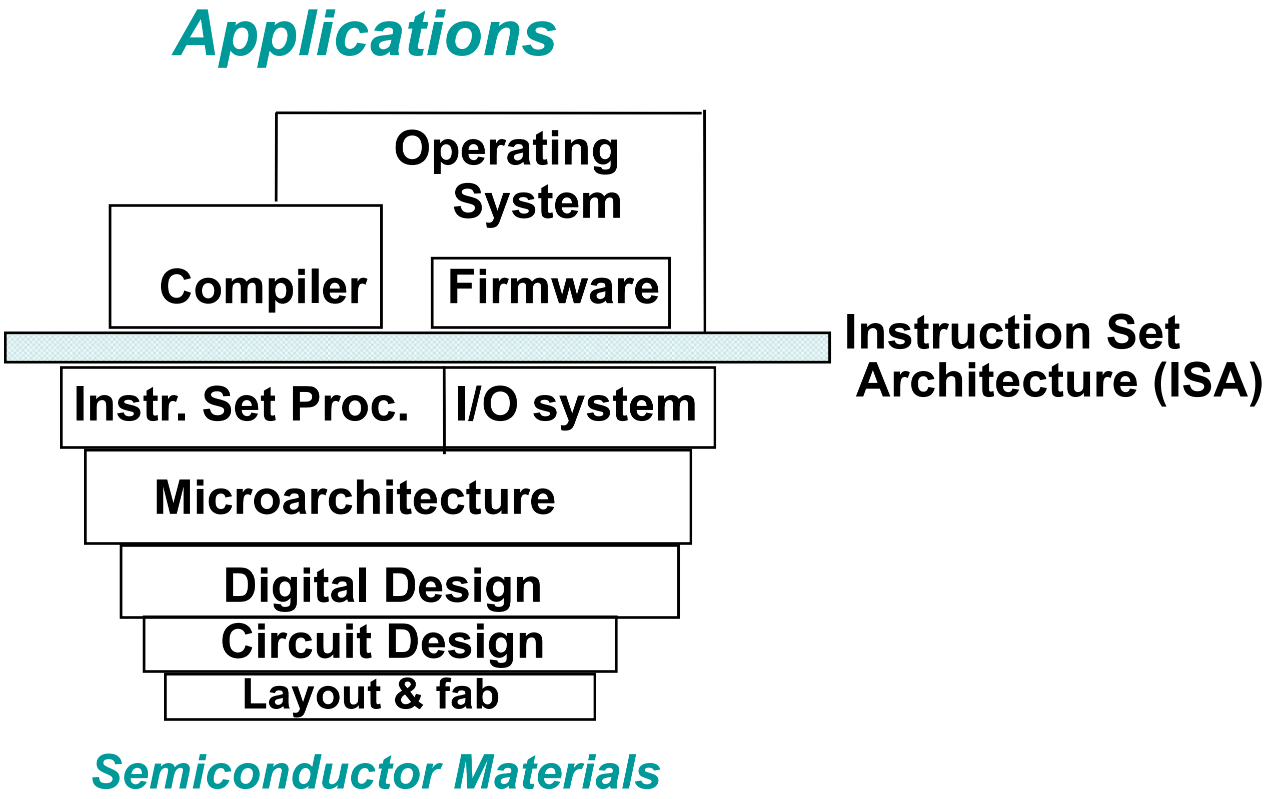
- Coordination of many *levels of abstraction*

Task of the Computer Designer



- Coordination of many *levels of abstraction*
- Under a rapidly changing set of forces

Task of the Computer Designer



- Coordination of many *levels of abstraction*
- Under a rapidly changing set of forces
- Design, Measurement, *and* Evaluation

Example ISA: MIPS64 [H&P B.9]

- We use MIPS64 when studying different microprocessor microarchitectures.
- Today: MIPS used in embedded microprocessors.
- In 1990's: MIPS used in some of the fastest computers.
- MIPS originally stood for "Microprocessor without Interlocking Pipeline Stages" (we'll learn about "interlocking in Slide Set 4 & 6)
- MIPS is a Reduced Instruction Set Computer (RISC) instruction set (we will learn more about this in Slide Set 3)

- MIPS64 architecture contains:
  32 integer registers R0… R31 (R0 always zero)
  32 floating-point registers F0… F31
  (+a few other registers)

Example MIPS64 ALU instructions

Notation (page B-36 in H&P)

Regs[Rn] Contents of register “n”
Mem[Addr] Contents of memory at location “Addr”
Concatenate bits
<- Means assign right hand side to location on left hand side
Superscript Replicate a field (0^{16} is a 16-bit field with all zeros)
Subscript Selection of bit (most significant bit = 0)

Example Instruction		Instruction Name	Meaning
DADDU	R1,R2,R3	Add unsigned	Regs[R1] <- Regs[R2]+Regs[R3]
DADDIU	R1,R2,#3	Add immediate unsigned	Regs[R1] <- Regs[R2]+3
LUI	R1,#42	Load upper immediate	Regs[R1] <- 0^{32}##42##0^{16}
DSLL	R1,R2,#5	Shift left logical	Regs[R1] <- Regs[R2] << 5

DSLT	R1,R2,R3	Set less than	if(Regs[R2] < Regs[R3])
			Regs[R1] <- 1 else Regs[R1] <- 0

Example MIPS64

Notation (page B-36 in H&P)

Regs[Rn]	Contents of register “n”
Mem[Addr]	Contents of memory at locati
##	Concatenate bits
<-	Means assign right hand side location on left hand side
Superscript	Replicate a field (0^{16} is a 16-b
Subscript	Selection of bit (most significant bit = 0)

What is the value of Regs[R4] at the end of the following sequence of instructions?

```
DADDIU      R1,R0,#2
LUI         R2,#4
DADDU       R3,R1,R2
DSLL        R4,R3,#1
```

A: 0x00000002

B: 0x00000004

C: 0x00040002

D: 0x00020001

E: 0x00080004

Example Instruction		Instruction Name	Meaning
DADDU	R1,R2,R3	Add unsigned	Regs[R1] <- Regs[R2]+Regs[R3]
DADDIU	R1,R2,#3	Add immediate unsigned	Regs[R1] <- Regs[R2]+3
LUI	R1,#42	Load upper immediate	Regs[R1] <- 0^{32}##42##0^{16}
DSLL	R1,R2,#5	Shift left logical	Regs[R1] <- Regs[R2] << 5

DSLT	R1,R2,R3	Set less than	if(Regs[R2] < Regs[R3])
			Regs[R1] <- 1 else Regs[R1] <- 0

Example MIPS64

What is the value of Regs[R4] at the end of the following sequence of instructions?

```
DADDIU      R1,R0,#2
LUI         R2,#4
DADDU       R3,R1,R2
DSLL        R4,R3,#1
```

A: 0x00000002
B: 0x00000004
C: 0x00040002
D: 0x00020001
E: 0x00080004

Notation (page B-36 in H&P)

Regs[Rn] Contents of register “n”
Mem[Addr] Contents of memory at locatio
Concatenate bits
<- Means assign right hand side
location on left hand side
Superscript Replicate a field (0^{16} is a 16-b
Subscript Selection of bit (most significant bit = 0)

Example Instruction		Instruction Name	Meaning
DADDU	R1,R2,R3	Add unsigned	Regs[R1] <- Regs[R2]+Regs[R3]
DADDIU	R1,R2,#3	Add immediate unsigned	Regs[R1] <- Regs[R2]+3
LUI	R1,#42	Load upper immediate	Regs[R1] <- 0^{32}##42##0^{16}
DSLL	R1,R2,#5	Shift left logical	Regs[R1] <- Regs[R2] << 5

DSLT	R1,R2,R3	Set less than	if(Regs[R2] < Regs[R3])
			Regs[R1] <- 1 else Regs[R1] <- 0

MIPS Load/Store instructions

Example Instruction		Instruction Name	Meaning
LD	R1,30(R2)	Load double word	Regs[R1] $<-_{64}$ Mem[30+Regs[R2]]
LW	R1,60(R2)	Load word	Regs[R1] $<-_{64}$ (Mem[60+Regs[R2]]$_{31}$)^{32} ## Mem[60+Regs[R2]]
SW	R3,500(R4)	Store word	Mem[500+Regs[R4]] $<-_{32}$ Regs[R3]
L.S	F0,50(R3)	Load FP single	Regs[R0] $<-_{64}$ Mem[50+Regs[R3]] ## 0^{32}
L.D	F0,50(R2)	Load FP double	Regs[F0] $<-_{64}$ Mem[50+Regs[R3]]

MIP

Assume Mem[100] contains 7, Mem[200] contains 2, Regs[R1] contains 80, Regs[R2] contains 204, and Regs[R3] contains 100, which of the following is true after the following code executes?

```
    LD      R1,20(R1)
    LD      R4,-4(R2)
    DADDU   R2,R4,R1
    SW      R2,200(R3)
```

A: Regs[R1] contains 100
B: Mem[300] contains 9
C: Regs[R2] contains 204
D: Mem[200] contains 9
E: Mem[100] contains 9

Example Instruction		Instruction Name	Meaning
LD	R1,30(R2)	Load double word	Regs[R1] <-$_{64}$ Mem[30+Regs[R2]]
LW	R1,60(R2)	Load word	Regs[R1] <-$_{64}$ (Mem[60+Regs[R2]]$_{31}$)^{32} ## Mem[60+Regs[R2]]
SW	R3,500(R4)	Store word	Mem[500+Regs[R4]] <-$_{32}$ Regs[R3]
L.S	F0,50(R3)	Load FP single	Regs[R0] <-$_{64}$ Mem[50+Regs[R3]] ## 0^{32}
L.D	F0,50(R2)	Load FP double	Regs[F0] <-$_{64}$ Mem[50+Regs[R3]]

MIP

Assume Mem[100] contains 7, Mem[200] contains 2, Regs[R1] contains 80, Regs[R2] contains 204, and Regs[R3] contains 100, which of the following is true after the following code executes?

```
    LD      R1,20(R1)
    LD      R4,-4(R2)
    DADDU   R2,R4,R1
    SW      R2,200(R3)
```

A: Regs[R1] contains 100
B: Mem[300] contains 9
C: Regs[R2] contains 204
D: Mem[200] contains 9
E: Mem[100] contains 9

Example Instruction		Instruction Name	Meaning
LD	R1,30(R2)	Load double word	Regs[R1] <-$_{64}$ Mem[30+Regs[R2]]
LW	R1,60(R2)	Load word	Regs[R1] <-$_{64}$ (Mem[60+Regs[R2]]$_{31}$)^{32} ## Mem[60+Regs[R2]]
SW	R3,500(R4)	Store word	Mem[500+Regs[R4]] <-$_{32}$ Regs[R3]
L.S	F0,50(R3)	Load FP single	Regs[R0] <-$_{64}$ Mem[50+Regs[R3]] ## 0^{32}
L.D	F0,50(R2)	Load FP double	Regs[F0] <-$_{64}$ Mem[50+Regs[R3]]

What is a branch?

C code:		MIPS64:
		; x => R1, p => R2, q => R3
x = 0;		DADDI R1,R0,#0
if(p!=NULL)	↔	BEQZ R2,target
x = *p;		LD R1,0(R2)
*q = x;		target: SD R1,0(R3)

MIPS Control flow instructions

Terminology we will use:
Jump = unconditional change of control flow
Branch = conditional change of control flow

Example Instruction		Instruction Name	Meaning
J	name	Jump	$PC_{36..63}$ <-name
JAL	name	Jump and link	Regs[R31] <- PC + 4; $PC_{36...63}$<-name
JALR	R2	Jump and link register	Regs[R31] <- PC + 4; PC <- Regs[R2]
JR	R3	Jump register	PC <- Regs[R3]
BEQZ	R4,name	Branch equal zero	if(Regs[R4] == 0) PC <- name
BNE	R3,R4,name	Branch not equal	if(Regs[R3]!=Regs[R4]) PC <- name
MOVZ	R1,R2,R3	Conditional move if zero	if(Regs[R3]==0) Regs[R1] <- Regs[R2]

MIPS

Terminology we w
Jump = unconditio
Branch = conditior

What is the value of Regs[R2] after the following code executes?

```
        DADDIU   R1,R0,#1
        BEQZ     R1,LABEL
        DADDIU   R1,R0,#2
LABEL:  DADDU    R2,R1,R1
```

A: 1
B: 2
C: 3
D: 4
E: 6

Example Instruction	Instruction Name	Meaning
J name	Jump	$PC_{36..63}$ <-name
JAL name	Jump and link	Regs[R31] <- PC + 4; $PC_{36...63}$<-name
JALR R2	Jump and link register	Regs[R31] <- PC + 4; PC <- Regs[R2]
JR R3	Jump register	PC <- Regs[R3]
BEQZ R4,name	Branch equal zero	if(Regs[R4] == 0) PC <- name
BNE R3,R4,name	Branch not equal	if(Regs[R3]!=Regs[R4]) PC <- name
MOVZ R1,R2,R3	Conditional move if zero	if(Regs[R3]==0) Regs[R1] <- Regs[R2]

MIPS

What is the value of Regs[R2] after the following code executes?

```
        DADDIU   R1,R0,#1
        BEQZ     R1,LABEL
        DADDIU   R1,R0,#2
LABEL:  DADDU    R2,R1,R1
```

A: 1
B: 2
C: 3
D: 4
E: 6

Terminology we w
Jump = unconditio
Branch = conditio

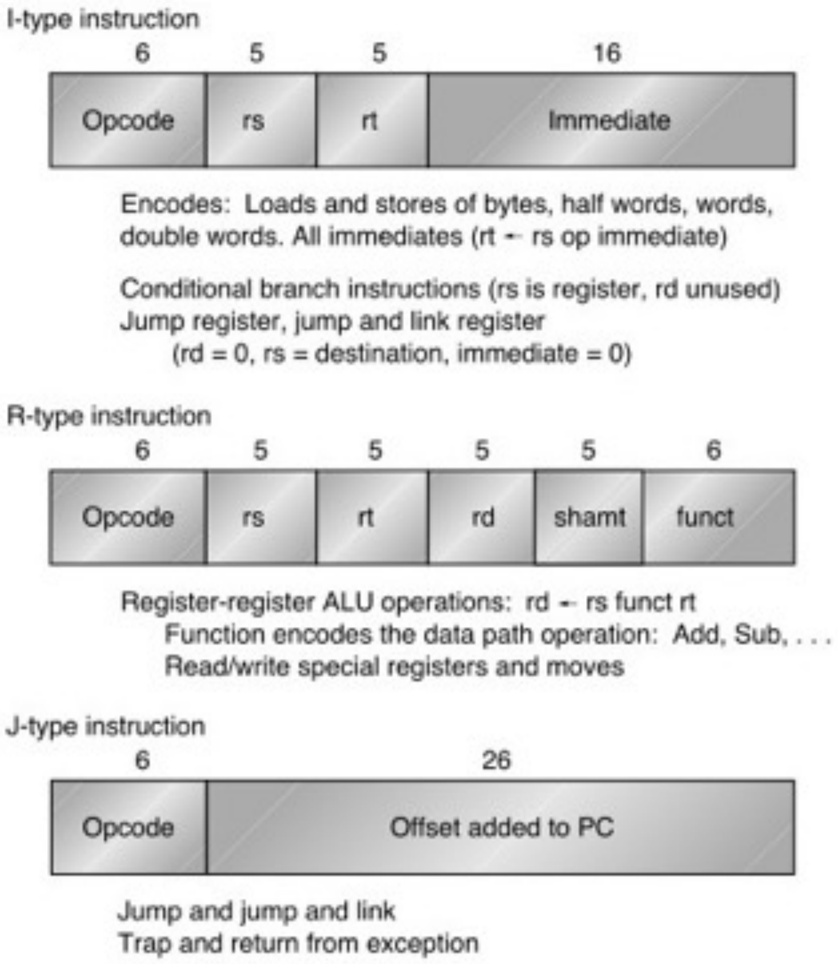
Example Instruction		Instruction Name	Meaning
J	name	Jump	PC$_{36..63}$ <-name
JAL	name	Jump and link	Regs[R31] <- PC + 4; PC$_{36...63}$<-name
JALR	R2	Jump and link register	Regs[R31] <- PC + 4; PC <- Regs[R2]
JR	R3	Jump register	PC <- Regs[R3]
BEQZ	R4,name	Branch equal zero	if(Regs[R4] == 0) PC <- name
BNE	R3,R4,name	Branch not equal	if(Regs[R3]!=Regs[R4]) PC <- name
MOVZ	R1,R2,R3	Conditional move if zero	if(Regs[R3]==0) Regs[R1] <- Regs[R2]

MIPS Instruction Encoding

Need some way to represent (or “encode”) an instruction as 1’s and 0’s to communicate with computer.

- All instructions 32 bits wide
- All instructions perform simple operations
- Only three instruction formats for all instructions

- Opcode specifies what operation to perform.
- “rs”, “rt”, “rd” fields indicate registers to read and/or write.



I-type instruction

6	5	5	16
Opcode	rs	rt	Immediate

Encodes: Loads and stores of bytes, half words, words, double words. All immediates (rt ← rs op immediate)

Conditional branch instructions (rs is register, rd unused)
Jump register, jump and link register
(rd = 0, rs = destination, immediate = 0)

R-type instruction

6	5	5	5	5	6
Opcode	rs	rt	rd	shamt	funct

Register-register ALU operations: rd ← rs funct rt
Function encodes the data path operation: Add, Sub, . . .
Read/write special registers and moves

J-type instruction

6	26
Opcode	Offset added to PC

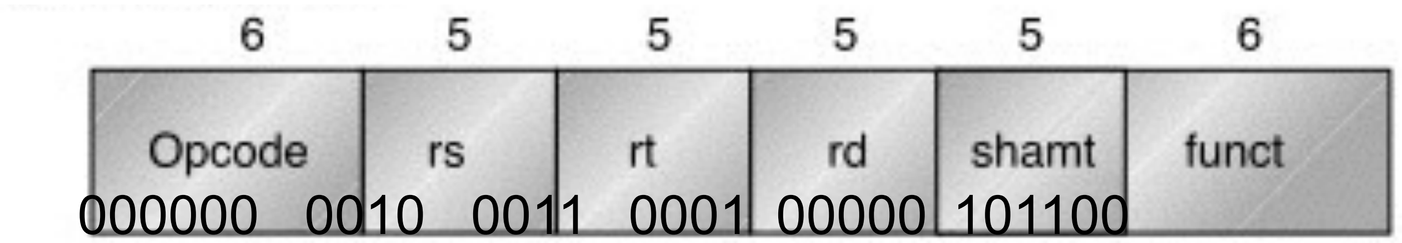
Jump and jump and link
Trap and return from exception

Example of Encoding MIPS64

The following assembly code

DADD R1,R2,R3

Is translated into:



6	5	5	5	5	6
Opcode	rs	rt	rd	shamt	funct
000000	0010	0011	0001	00000	101100

Technology Trends

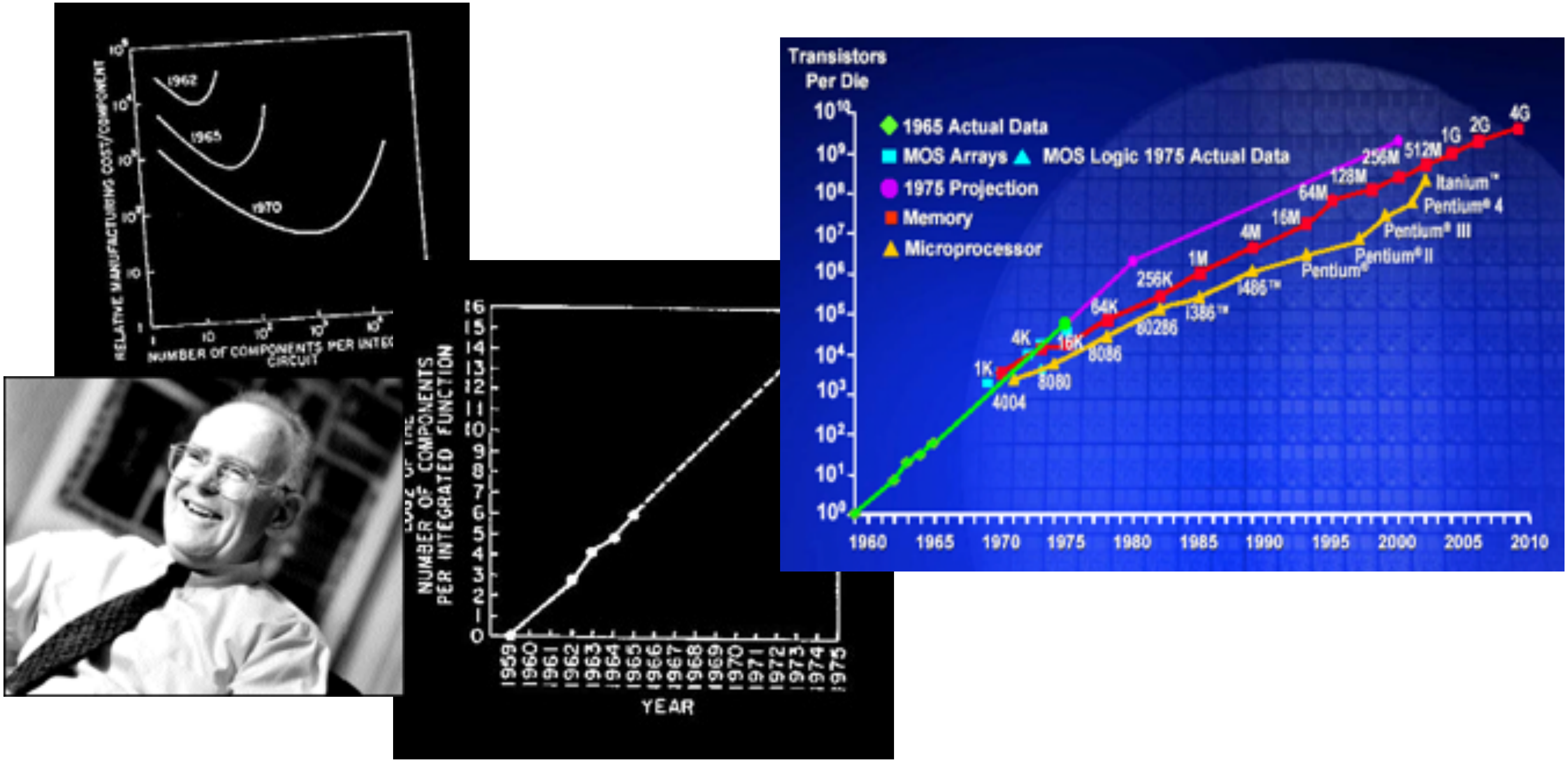
- In this context, “technology” refers to how computing elements are implemented. For example, is a circuit implemented from vacuum tubes, 1 um transistors, or 45 nm transistors.

- The technology used to implement a computer places constraints upon the design. As the implementation technology (e.g., transistors or circuit techniques) changes, some properties (e.g., switch time) might improve faster than others (e.g., threshold voltage) resulting an architecture design that would be a poor choice for an earlier technology become a good choice for a later technology.

- “transistors are [now] cheaper than printed
- characters in the Sunday New York Times.”

- [Gordon Moore, “Computer History Museum Presents: The 40th Anniversary of Moore's Law”,
- Sept. 29, 2005]

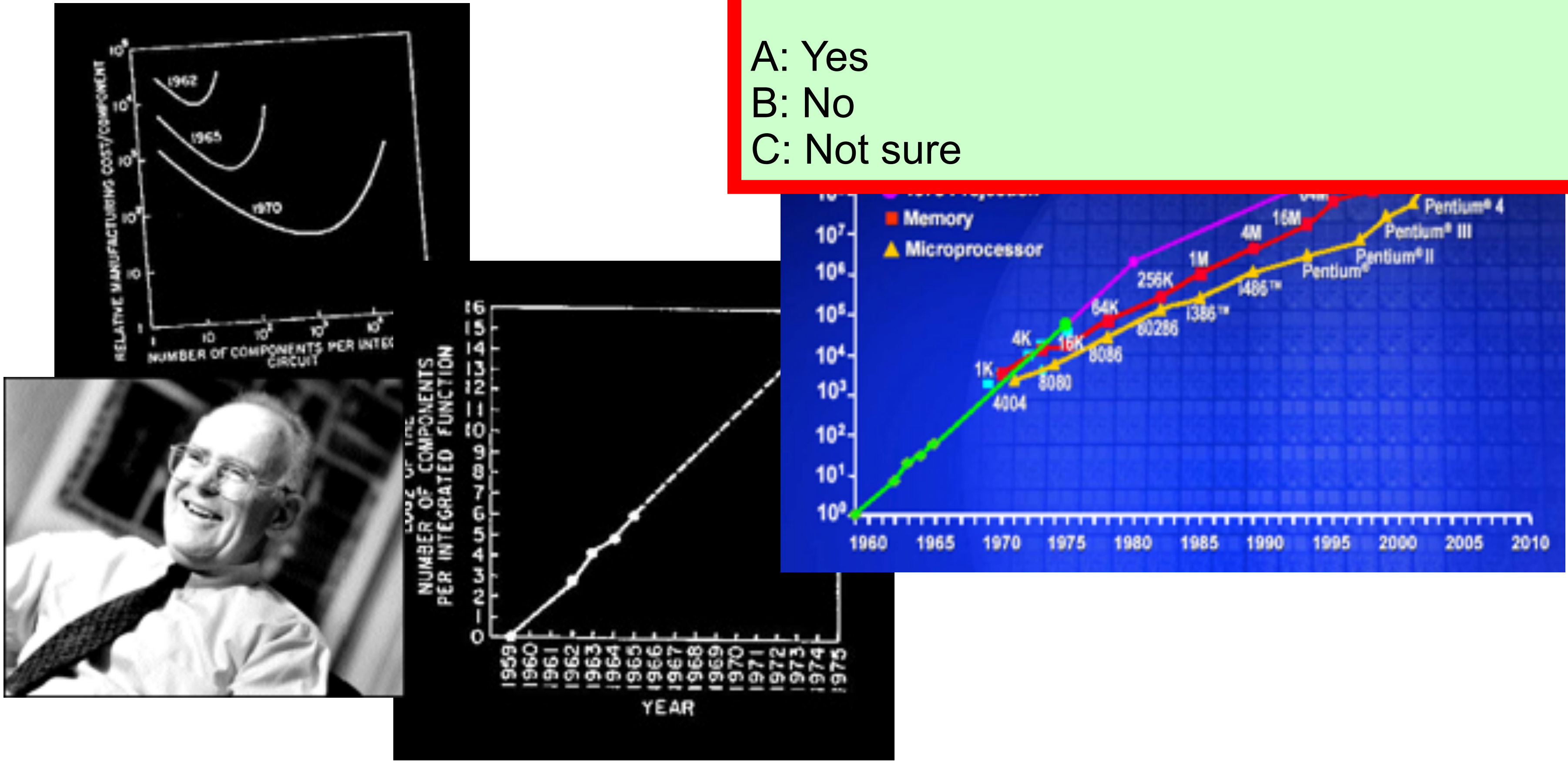
- <u>Let’s check (Sept 3, 2006)</u>
- Newegg.com: Celeron D 310 -> $43 / ~ 120 million transistors
- = $0.358 / million transistors
- Sunday Nytimes: $3.5 / [(5000 word/page) x (7 char/word) x (273 pages)]
- = $0.366 / million characters



- “Cramming More Components onto Integrated Circuits”
  - Gordon Moore, Electronics, 1965
- # on transistors on cost-effective integrated circuit double every N months (12 ≤ N ≤ 24)

If number of transistors doubles is it automatically true the time to run a program is cut in half?

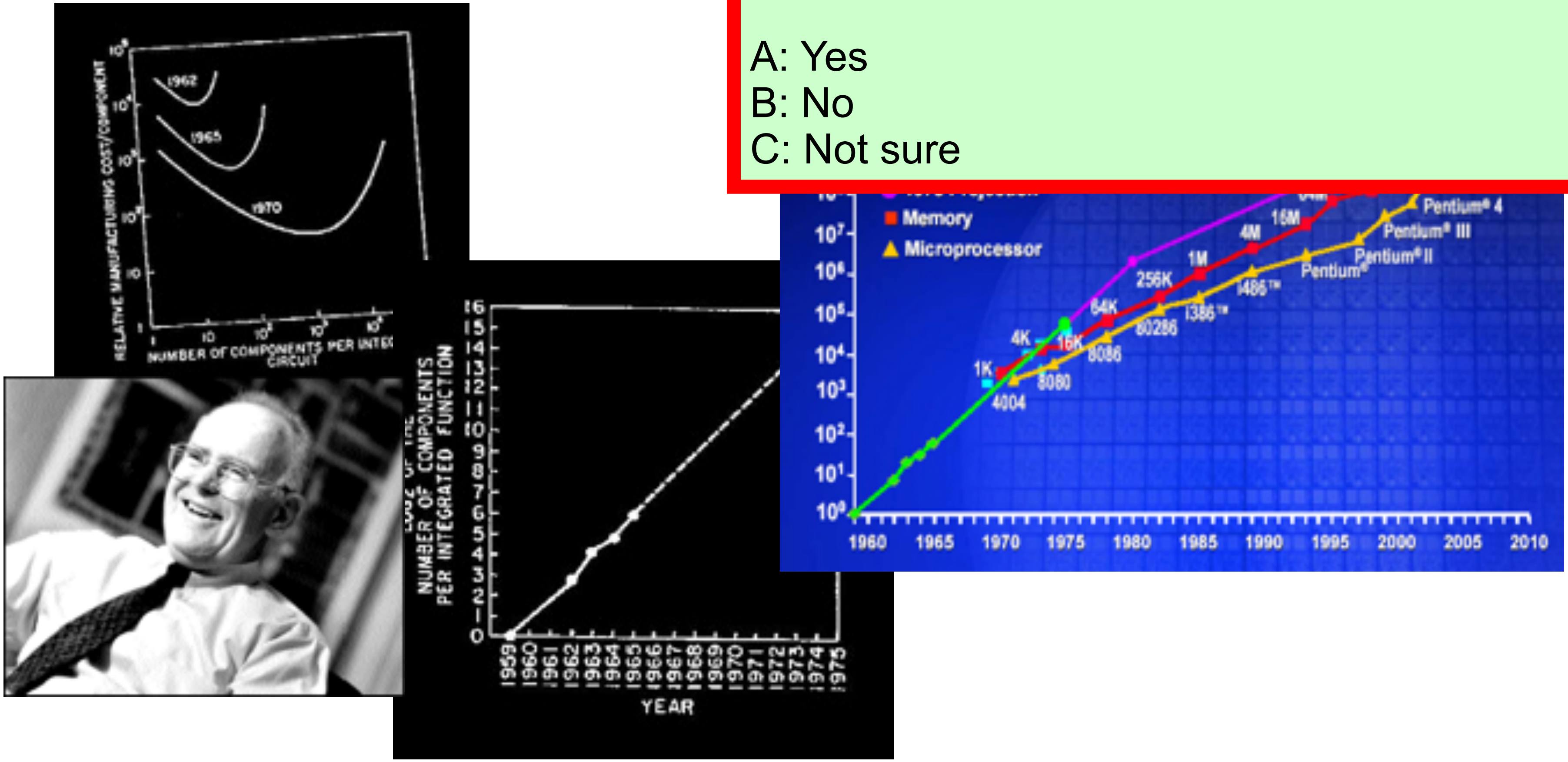
A: Yes
B: No
C: Not sure



- “Cramming More Components onto Integrated Circuits”
  - Gordon Moore, Electronics, 1965
- # on transistors on cost-effective integrated circuit double every N months (12 ≤ N ≤ 24)

If number of transistors doubles is it automatically true the time to run a program is cut in half?

A: Yes
B: No
C: Not sure



- "Cramming More Components onto Integrated Circuits"
  - Gordon Moore, Electronics, 1965
- # on transistors on cost-effective integrated circuit double every N months (12 ≤ N ≤ 24)

Tracking Technology Performance Trends

- Examine 4 components of computing systems
  - Disks,
  - Memory,
  - Network,
  - Processors
- Compare ~1980 Archaic (Nostalgic) vs. ~2000 Modern (Newfangled)
  - Performance Milestones in each technology
- Compare for Bandwidth vs. Latency improvements in performance over time
- Bandwidth: number of events per unit time
  - E.g., M bits / second over network, M bytes / second from disk
  - Often also called “throughput” (tasks per unit time)
- Latency: elapsed time for a single event
  - E.g., one-way network delay in microseconds, average disk access time in milliseconds
  - Execution time for a single task

What is the fastest way to get 400 people from New York to London?

A: One Concorde in air at a time

B: One 747 in air at a time

-
  - Disks,
  - Memory,
  - Network,
  - Processors
- Compare ~1980 Archaic (Nostalgic) vs. ~2000 Modern (Newfangled)
  - Performance Milestones in each technology
- Compare for Bandwidth vs. Latency improvements in performance over time
- Bandwidth: number of events per unit time
  - E.g., M bits / second over network, M bytes / second from disk
  - Often also called “throughput” (tasks per unit time)
- Latency: elapsed time for a single event
  - E.g., one-way network delay in microseconds, average disk access time in milliseconds
  - Execution time for a single task

What is the fastest way to get 400 people from New York to London?

A: One Concorde in air at a time

B: One 747 in air at a time

- Disks,
- Memory,
- Network,
- Processors

- Compare ~1980 Archaic (Nostalgic) vs. ~2000 Modern (Newfangled)
  - Performance Milestones in each technology
- Compare for Bandwidth vs. Latency improvements in performance over time
- Bandwidth: number of events per unit time
  - E.g., M bits / second over network, M bytes / second from disk
  - Often also called “throughput” (tasks per unit time)
- Latency: elapsed time for a single event
  - E.g., one-way network delay in microseconds, average disk access time in milliseconds
  - Execution time for a single task

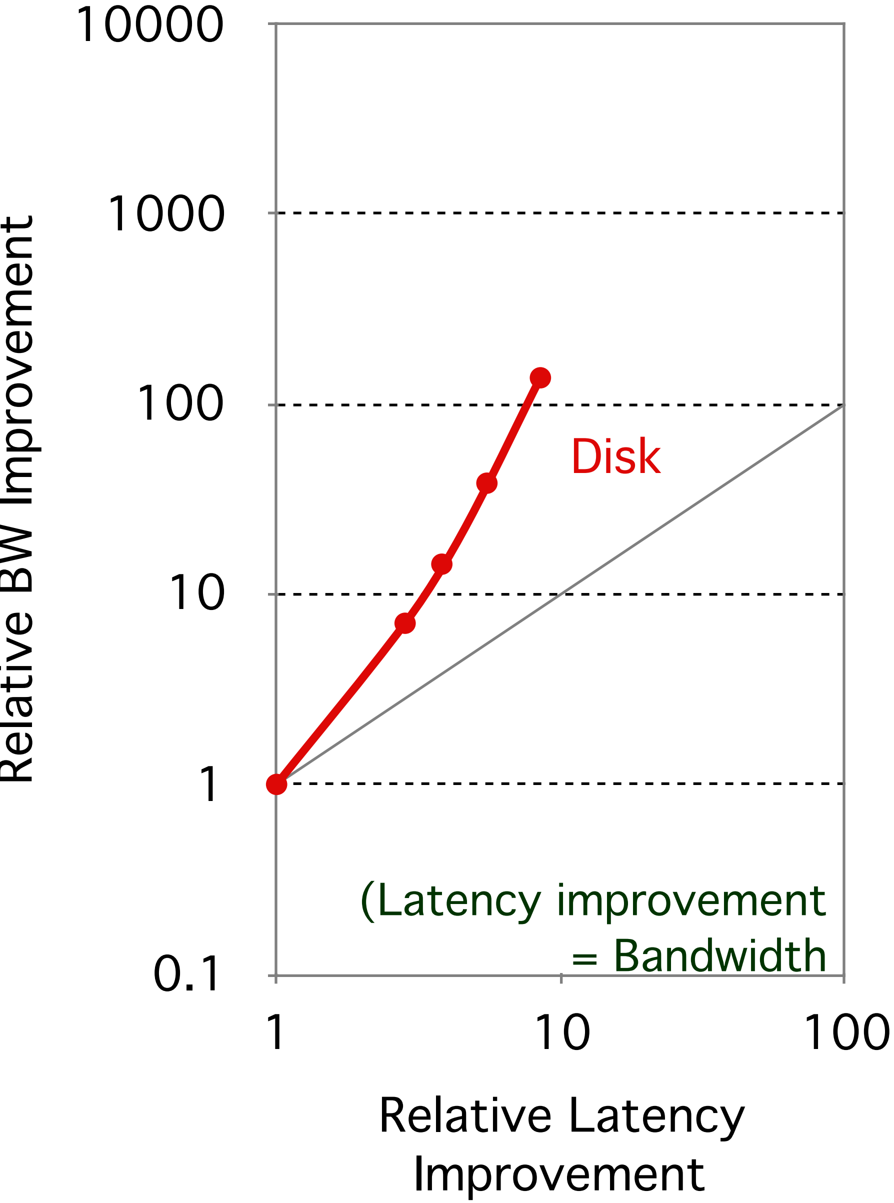
Tracking Technology Performance Trends

- Examine 4 components of computing systems
  - Disks,
  - Memory,
  - Network,
  - Processors
- Compare ~1980 Archaic (Nostalgic) vs. ~2000 Modern (Newfangled)
  - Performance Milestones in each technology
- Compare for Bandwidth vs. Latency improvements in performance over time
- Bandwidth: number of events per unit time
  - E.g., M bits / second over network, M bytes / second from disk
  - Often also called “throughput” (tasks per unit time)
- Latency: elapsed time for a single event
  - E.g., one-way network delay in microseconds, average disk access time in milliseconds
  - Execution time for a single task

Disks: Archaic(Nostalgic) v. Modern (Newfangled)

• CDC Wren I, 1983	• Seagate 373453, 2003	
• 3600 RPM	• 15000 RPM	(4X)
• 0.03 GBytes capacity	• 73.4 GBytes	(2500X)
• Tracks/Inch: 800	• Tracks/Inch: 64000	(80X)
• Bits/Inch: 9550	• Bits/Inch: 533,000	(60X)
• Three 5.25” platters	• Four 2.5” platters (in 3.5” form factor)	
• Bandwidth: 0.6 MBytes/sec	• Bandwidth: 86 MBytes/sec	(140X)
• Latency: 48.3 ms	• Latency: 5.7 ms	(8X)
• Cache: none	• Cache: 8 MBytes	

Latency Lags Bandwidth (for last ~20 years)



- Performance Milestones

- Disk: 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)

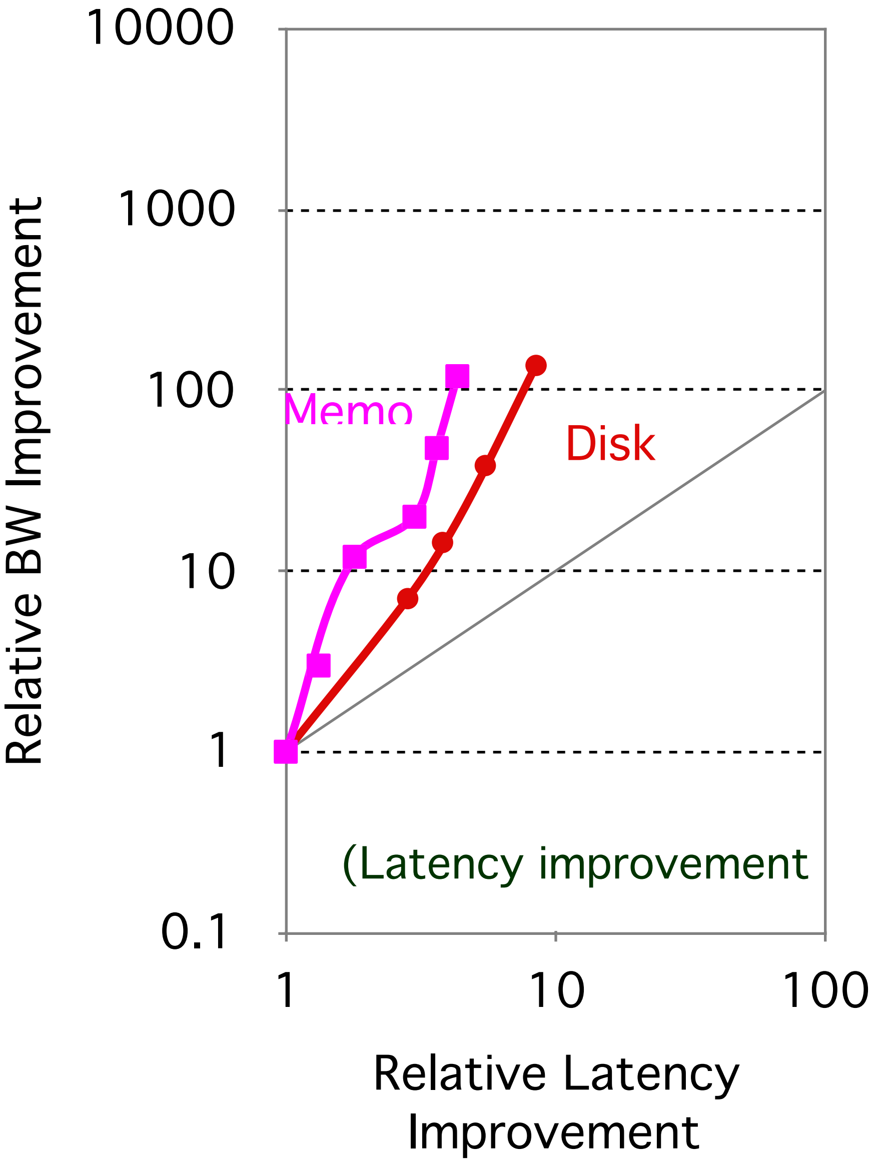
(latency = simple operation w/o contention
BW = best-case)

Memory: Archaic (Nostalgic) v. Modern (Newfangled)

- 1980 DRAM (asynchronous)
- 0.06 Mbits/chip
- 64,000 xtors, 35 mm^2
- 16-bit data bus per module, 16 pins/chip
- 13 Mbytes/sec
- Latency: 225 ns
- (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM
- 256.00 Mbits/chip (4000X)
- 256,000,000 xtors, 204 mm^2
- 64-bit data bus per DIMM, 66 pins/chip (4X)
- 1600 Mbytes/sec (120X)
- Latency: 52 ns (4X)
- Block transfers (page mode)

Latency Lags Bandwidth (last ~20 years)



- Performance Milestones

- Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM (4x,120x)
- Disk: 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)

(latency = simple operation w/o contention
BW = best-case)

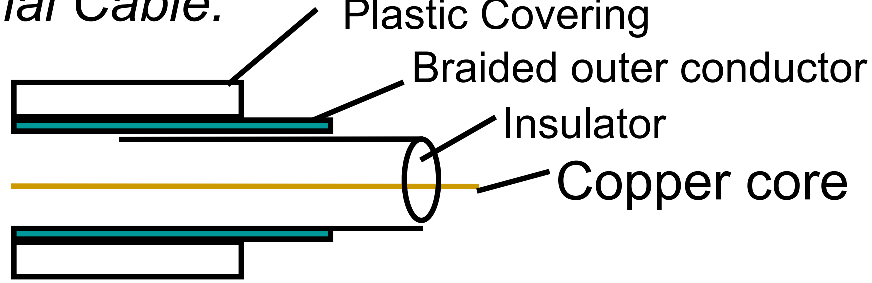
LANs: Archaic (Nostalgic) vs. Modern (Newfangled)

- Ethernet 802.3
- Year of Standard: 1978
- 10 Mbits/s
  link speed
- Latency: 3000 μsec
- Shared media
- Coaxial cable

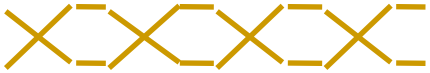
- Ethernet 802.3ae
- Year of Standard: 2003
- 10,000 Mbits/s
  (1000X)
  link speed
- Latency: 190 μsec (15X)
- Switched media
- Category 5 copper wire

"Cat 5" is 4 twisted pairs in bundle

Coaxial Cable:

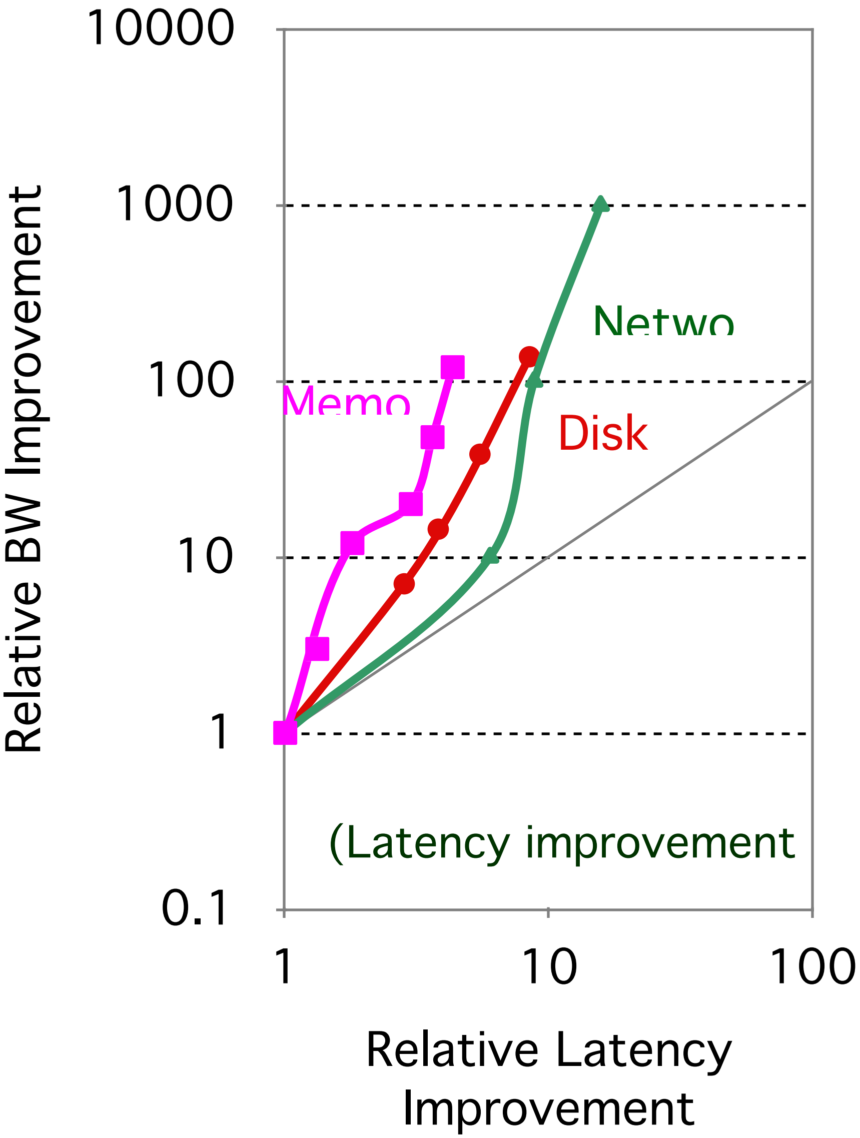


Twisted Pair:



Copper, 1mm thick,
twisted to avoid antenna effect

Latency Lags Bandwidth (last ~20 years)



- Performance Milestones
- Ethernet: 10Mb, 100Mb, 1000Mb, 10000 Mb/s (16x,1000x)
- Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM (4x,120x)
- Disk: 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)

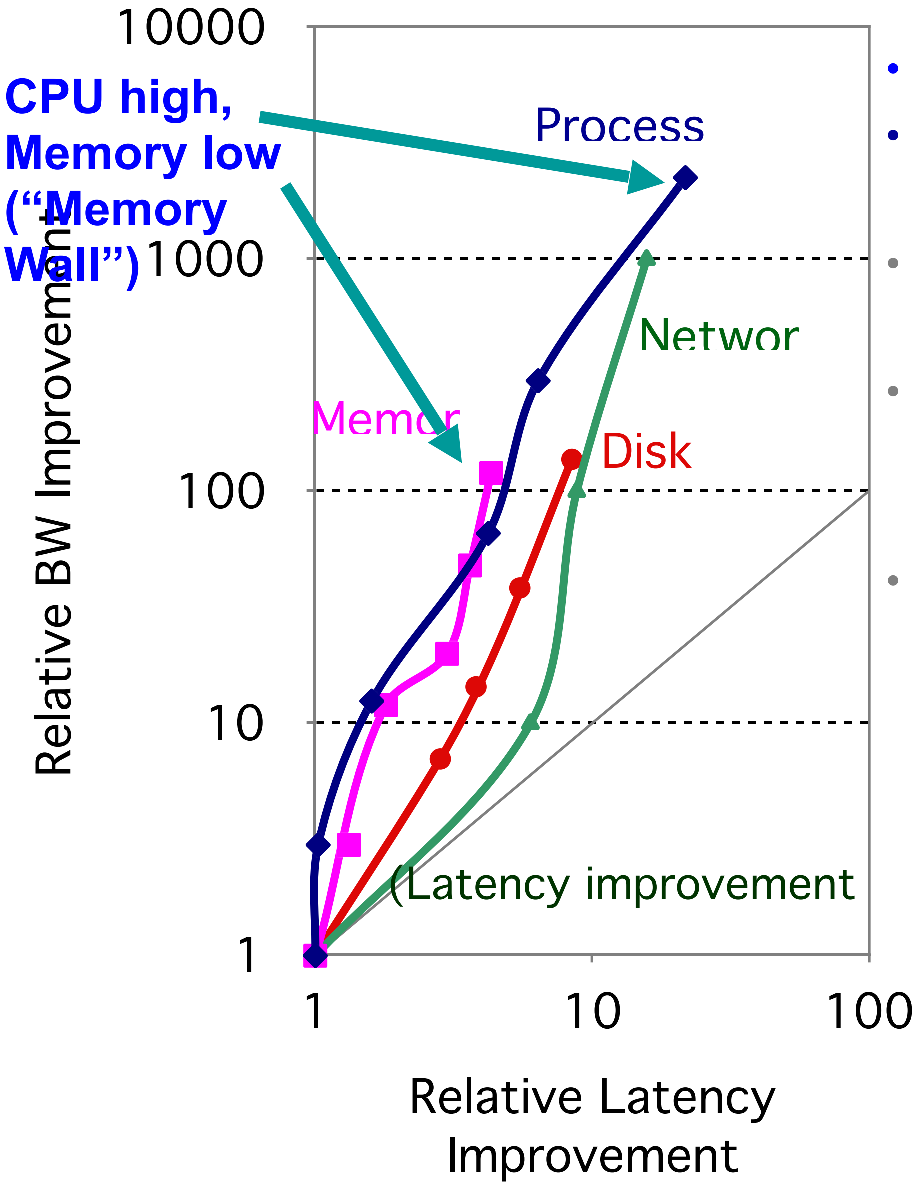
(latency = simple operation w/o contention
BW = best-case)

CPUs: Archaic (Nostalgic) vs. Modern (Newfangled)

- 1982 Intel 80286
- 12.5 MHz
- 2 MIPS (peak)
- Latency 320 ns
- 134,000 xtors, 47 mm^2
- 16-bit data bus, 68 pins
- Microcode interpreter, separate FPU chip
- (no caches)

- 2001 Intel Pentium 4
- 1500 MHz (120X)
- 4500 MIPS (peak) (2250X)
- Latency 15 ns (20X)
- 42,000,000 xtors, 217 mm^2
- 64-bit data bus, 423 pins
- 3-way superscalar, Dynamic translate to RISC, Superpipelined (22 stage), Out-of-Order execution
- On-chip 8KB Data caches, 96KB Instr. Trace cache, 256KB L2 cache

Latency Lags Bandwidth (last ~20 years)



- Performance Milestones
- Processor: ‘286, ‘386, ‘486, Pentium, Pentium Pro, Pentium 4 (21x,2250x)
- Ethernet: 10Mb, 100Mb, 1000Mb, 10000 Mb/s (16x,1000x)
- Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM (4x,120x)
- Disk : 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)

Rule of Thumb for Latency Lagging BW

- In the time that bandwidth doubles, latency improves by no more than a factor of 1.2 to 1.4 (and capacity improves faster than bandwidth)
- Stated alternatively:
  Bandwidth improves by more than the square of the improvement in Latency

Power (1 / 2)

- For CMOS chips, traditional dominant energy consumption has been in switching transistors, called *dynamic power*

$$Power_{dynamic} = 1/2 \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched$$

- For mobile devices, energy better metric

$$Energy_{dynamic} = CapacitiveLoad \times Voltage^2$$

- For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy (but, slowing clock rate may allow voltage to decrease)
- Capacitive load a function of number of transistors connected to output and technology, which determines capacitance of wires and transistors
- Dropping voltage helps both (so went from 5V to 1V)
- To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)

Example of quantifying power

- Suppose 15% reduction in frequency (reducing performance by roughly the same amount) allows us to safely reduce voltage by 15%. What is impact on dynamic power?

Example of quantifying power

Power reduces by:

- Suppose 15% reduction in frequency (reducing performance by roughly the same amount) allows us to safely reduce voltage by 15%. What is impact on dynamic power?

Example of quantifying power

- Suppose 15% reduction in frequency (reducing performance by roughly the same amount) allows us to safely reduce voltage by 15%. What is impact on dynamic power?

Power reduces by:

A: ~15%
B: ~20%
C: ~30%
D: ~40%
E: not sure

Example of quantifying power

- Suppose 15% reduction in frequency (reducing performance by roughly the same amount) allows us to safely reduce voltage by 15%. What is impact on dynamic power?

Power reduces by:

A: ~15%
B: ~20%
C: ~30%
D: ~40% ✔
E: not sure

Example of quantifying power

- Suppose 15% reduction in frequency (reducing performance by roughly the same amount) allows us to safely reduce voltage by 15%. What is impact on dynamic power?

Power reduces by:

A: ~15%
B: ~20%
C: ~30%
D: ~40% ✓
E: not sure

$$
\begin{aligned}
Power_{dynamic} &= 1/2 \times CapacitiveLoad \times Voltage^{2} \times FrequencySwitched \\
&= 1/2 \times .85 \times CapacitiveLoad \times (.85 \times Voltage)^{2} \times FrequencySwitched \\
&= (.85)^{3} \times 1/2 \times CapacitiveLoad \times Voltage^{2} \times FrequencySwitched \\
&= (.85)^{3} \times OldPower_{dynamic} \\
&\approx 0.6 \times OldPower_{dynamic}
\end{aligned}
$$

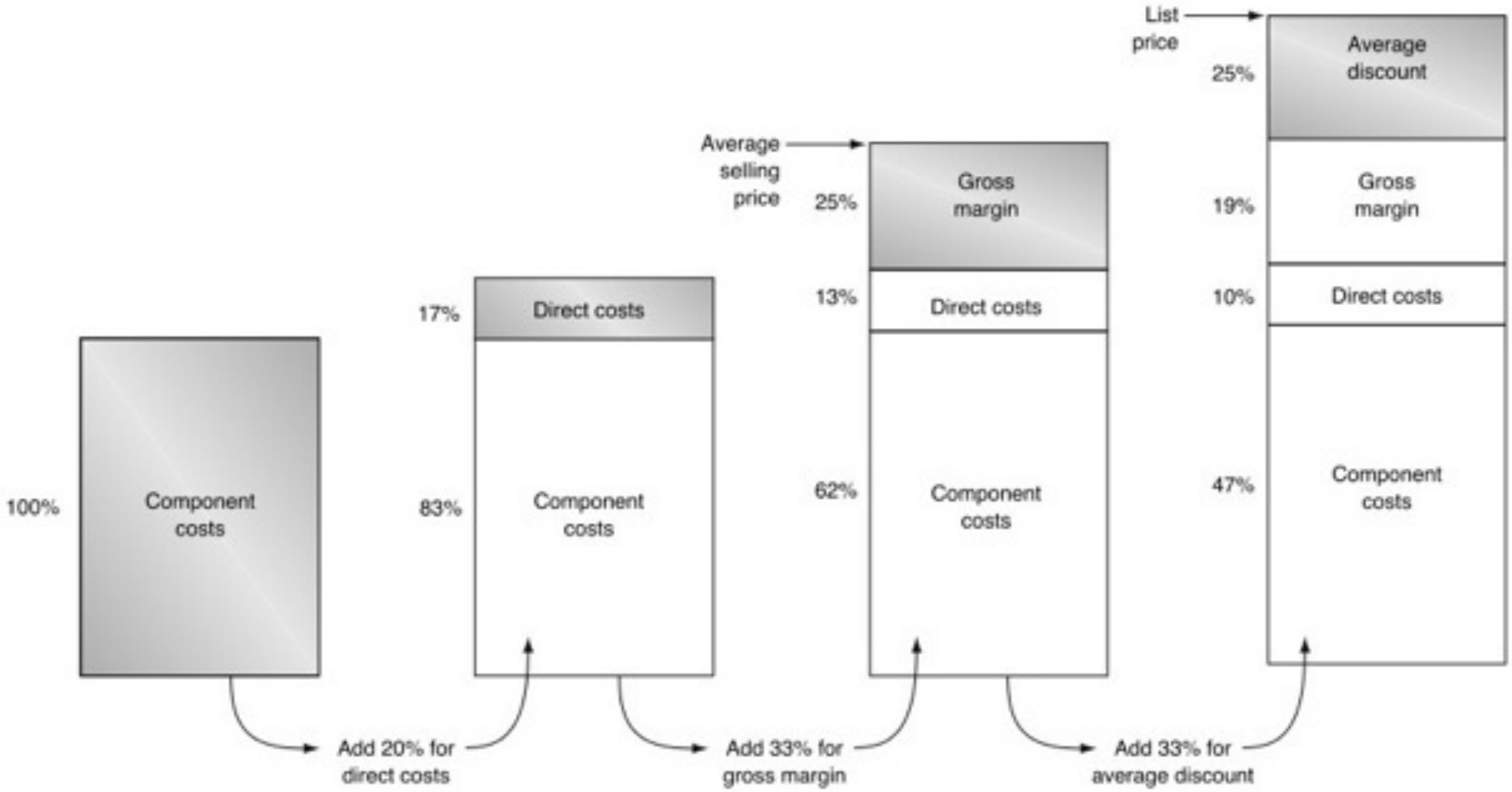
Power (2 / 2)

- Because leakage current flows even when a transistor is off, now *static power* important too

$$\text{Power}_{\text{static}} = \text{Current}_{\text{static}} \times \text{Voltage}$$

- Leakage current increases in processors with smaller transistor sizes
- Increasing the number of transistors increases power even if they are turned off
- In recent years, goal for leakage is 25% of total power consumption; high performance designs at 40%
- Very low power systems even gate voltage to inactive modules to control loss due to leakage

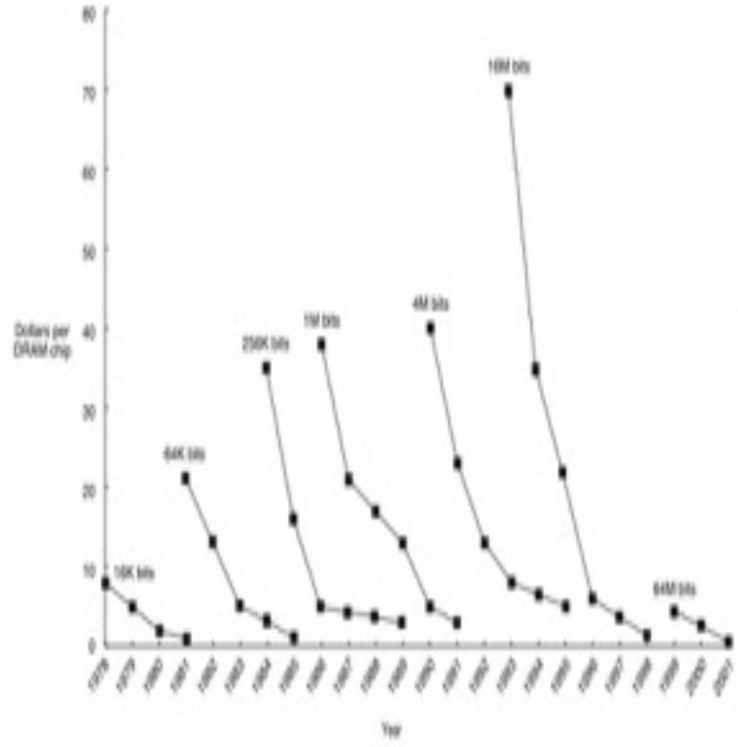
Cost versus Price

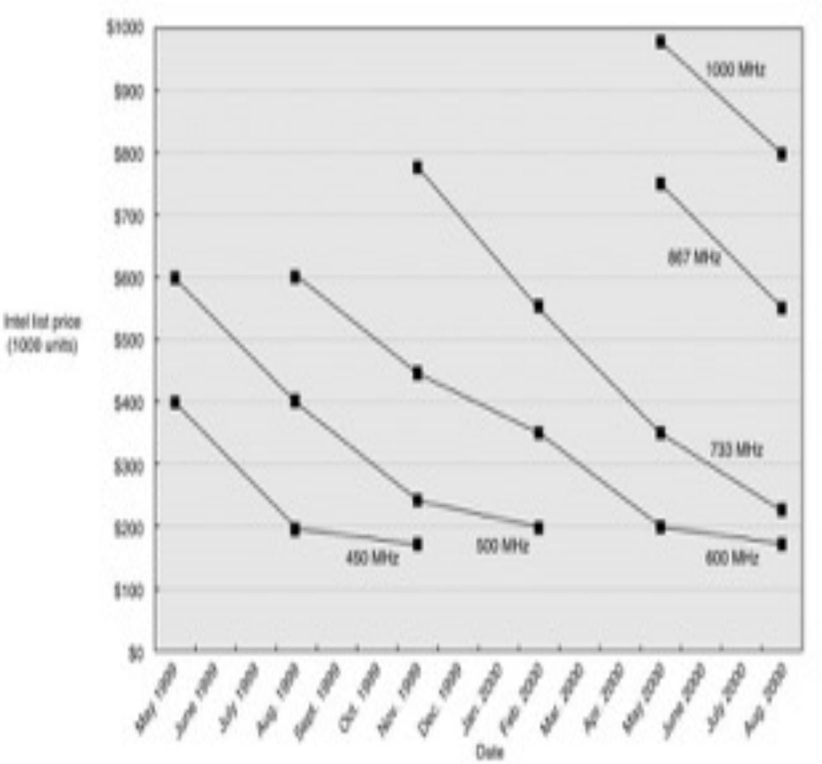


	2Q 2006			2Q 2007			
	GM	~P/E	~SP	GM	~P/E	~SP	ΔSP
Intel	52%	17	$18	47%	26	$26	44%
AMD	57%	26	$24	33%	(loss)	$13	(50%)
ΔGM	5%	(AMD higher)		14% (Intel higher)			

Cost, Price and Their Trends…
Impact of Time, Volume, Commodification

Price





Time

Cost, Price and Their Trends…
Impact of Time, Volume, Commodification

- Why is cost important?
  - tradeoffs in design.
- Why does cost change?
  - "learning curve" (improvement in yield)
- How does volume impact cost?
  - reduces time needed to get down the learning curve (proportional to number of chips produced)
  - increased purchasing and manufacturing efficiency
  - amortization of development cost (lower price)
- How does commodification impact price?
  - reduces price due to competition

Silicon: What chips are made of

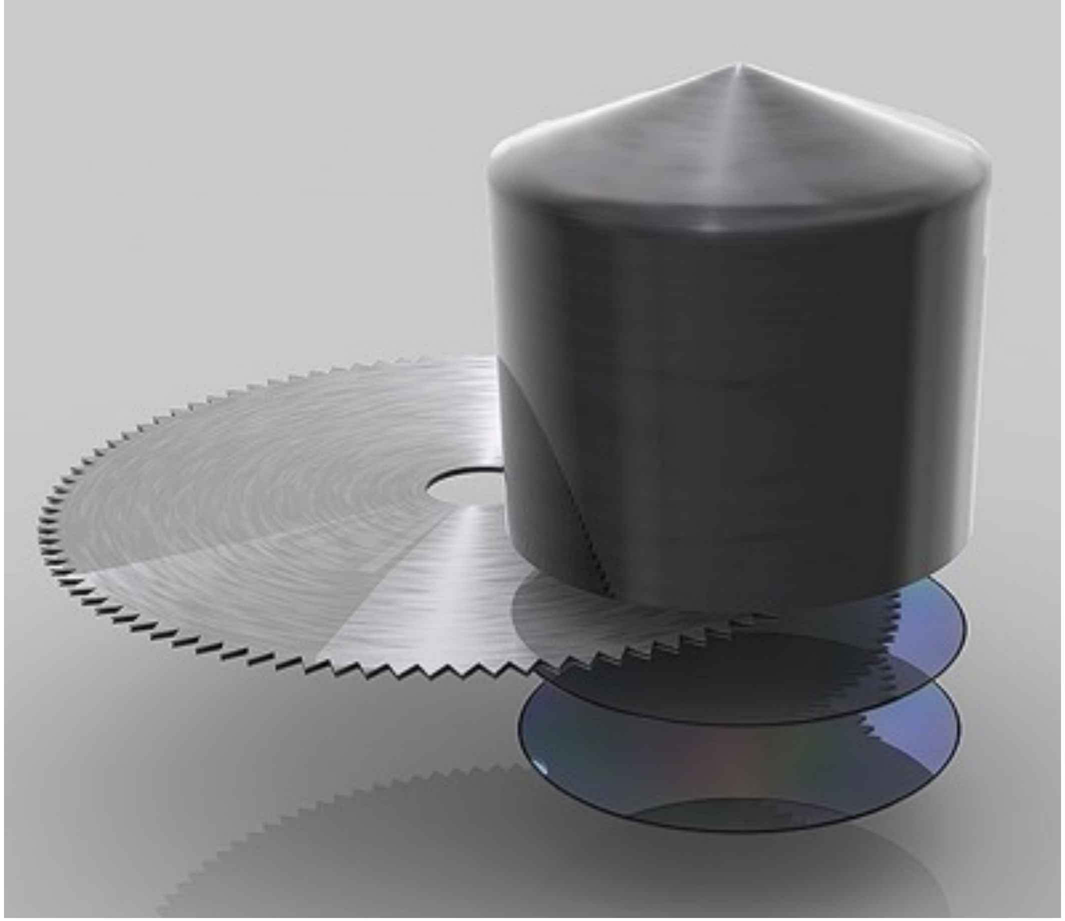


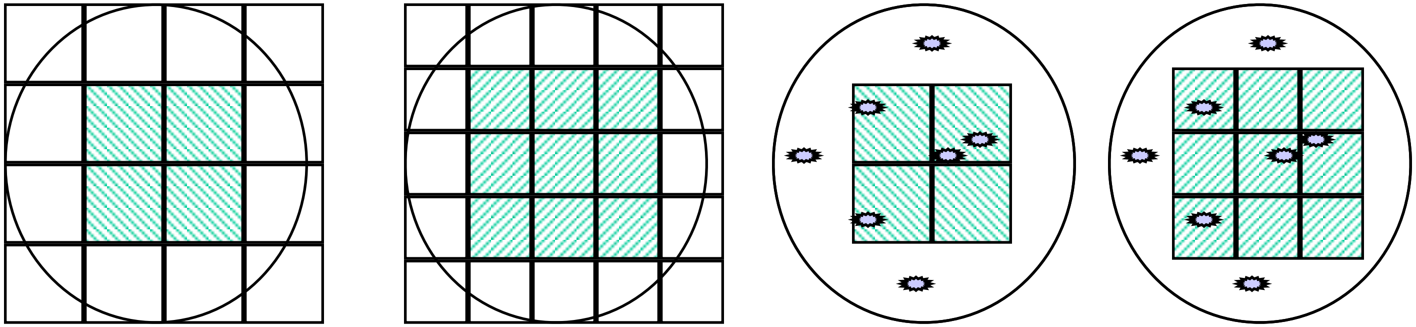
Image Courtesy of apcmag.com

Integrated Circuits Costs

$$\text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}$$

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi\ (\text{Wafer\_diam}/2)^2}{\text{Die\_Area}} - \frac{\pi \times \text{Wafer\_diam}}{\sqrt{2 \cdot \text{Die\_Area}}} - \text{Test\_Die}$$



$$\text{Die Yield} = \text{Wafer\_yield} \times \left(1 + \frac{\text{Defect\_Density} \times \text{Die\_area}}{\alpha}\right)^{-\alpha}$$

Real World Examples

Chip	Metal layers	Line width	Wafer cost	Defect /cm^2	Area mm^2	Dies/ wafer	Yield	Die Cost
386DX	2	0.90	$900	1.0	43	360	71%	$4
486DX2	3	0.80	$1200	1.0	81	181	54%	$12
PowerPC 601	4	0.80	$1700	1.3	121	115	28%	$53
HP PA 7100	3	0.80	$1300	1.0	196	66	27%	$73
DEC Alpha	3	0.70	$1500	1.2	234	53	19%	$149
SuperSPARC	3	0.70	$1700	1.6	256	48	13%	$272
Pentium	3	0.80	$1500	1.5	296	40	9%	$417

– From "Estimating IC Manufacturing Costs," by Linley Gwennap, *Microprocessor Report*, August 2, 1993, p. 15

Example Question

$$\text{Testing cost} = \frac{\text{Cost of testing per hour} \times \text{Average die test time}}{\text{Die yield}}$$

- Itanium…

- Alpha	= 4
- Die area	= 300 mm^2
- Wafer size	= 200 mm (diameter)
- Wafer yield	= 0.95
- Pins	= 418
- Technology	= CMOS, 0.18 um, 6M
- Est. Wafer Cost	= $4900
- Package	= $20 each
- Avg Testing Time	= 30 sec
- Cost of testing	= $400/hr
- Final test yield	= 1.0

Example Question, cont'd

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

Example Question, cont'd

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

Dies per wafer = 104 - 25 = 79

Example Question, cont'd

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

Dies per wafer	= 104 - 25	= 79
Good dies per wafer (0.3/cm^2)	= 79*(0.4219)	= 33

Example Question, cont'd

- **Determine cost if defect density = $0.3/cm^2$ vs. $1.0/cm^2$**

Dies per wafer	= 104 - 25	= 79
Good dies per wafer ($0.3/cm^2$)	= 79*(0.4219)	= 33
($1.0/cm^2$)	= 79*(0.1013)	= 8

Example Question, cont'd

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

Dies per wafer	= 104 - 25	= 79
Good dies per wafer (0.3/cm^2)	= 79*(0.4219)	= 33
(1.0/cm^2)	= 79*(0.1013)	= 8
Die Cost (0.3/cm^2)	= $4900/33	= $148.48

Example Question, cont'd

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

Dies per wafer		= 104 - 25	= 79
Good dies per wafer	(0.3/cm^2)	= 79*(0.4219)	= 33
	(1.0/cm^2)	= 79*(0.1013)	= 8
Die Cost	(0.3/cm^2)	= $4900/33	= $148.48
	(1.0/cm^2)	= $4900/8	= $612.50

Example Question, cont'd

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

Dies per wafer		= 104 - 25	= 79
Good dies per wafer	(0.3/cm^2)	= 79*(0.4219)	= 33
	(1.0/cm^2)	= 79*(0.1013)	= 8
Die Cost	(0.3/cm^2)	= \$4900/33	= \$148.48
	(1.0/cm^2)	= \$4900/8	= \$612.50
Testing Cost	(0.3/cm^2)	= \$400*(1/120)/0.4219	= \$7.90

Example Question, cont’d

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

Dies per wafer		= 104 - 25	= 79
Good dies per wafer	(0.3/cm^2)	= 79*(0.4219)	= 33
	(1.0/cm^2)	= 79*(0.1013)	= 8
Die Cost	(0.3/cm^2)	= \$4900/33	= \$148.48
	(1.0/cm^2)	= \$4900/8	= \$612.50
Testing Cost	(0.3/cm^2)	= \$400*(1/120)/0.4219	= \$7.90
	(1.0/cm^2)	= \$400*(1/120)/0.1013	= \$32.91

Example Question, cont'd

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

Dies per wafer		= 104 - 25	= 79
Good dies per wafer	(0.3/cm^2)	= 79*(0.4219)	= 33
	(1.0/cm^2)	= 79*(0.1013)	= 8
Die Cost	(0.3/cm^2)	= \$4900/33	= \$148.48
	(1.0/cm^2)	= \$4900/8	= \$612.50
Testing Cost	(0.3/cm^2)	= \$400*(1/120)/0.4219	= \$7.90
	(1.0/cm^2)	= \$400*(1/120)/0.1013	= \$32.91
Total Cost	(0.3/cm^2)	= \$148.48+\$7.90+\$20	= \$176.38

Example Question, cont'd

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

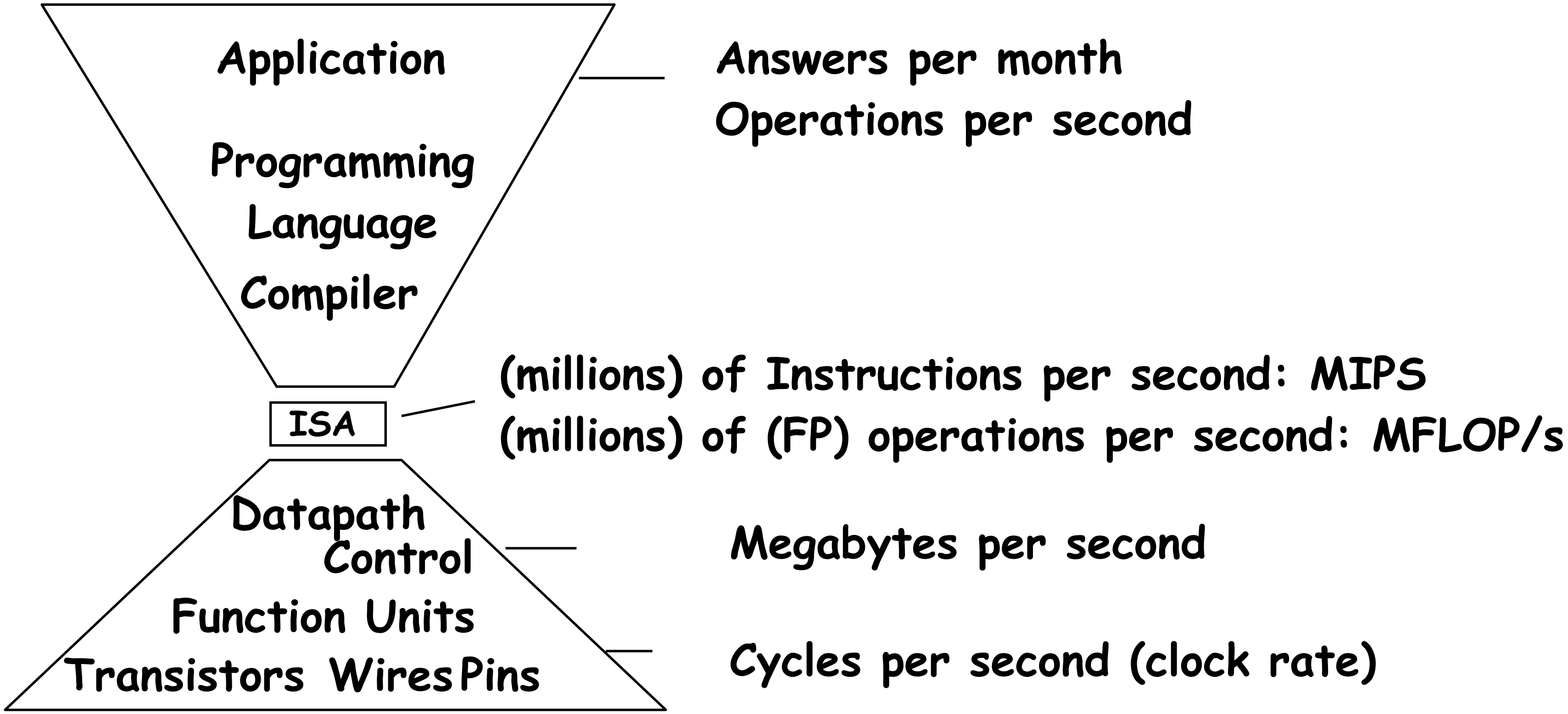
Dies per wafer		= 104 - 25	= 79
Good dies per wafer	(0.3/cm^2)	= 79*(0.4219)	= 33
	(1.0/cm^2)	= 79*(0.1013)	= 8
Die Cost	(0.3/cm^2)	= \$4900/33	= \$148.48
	(1.0/cm^2)	= \$4900/8	= \$612.50
Testing Cost	(0.3/cm^2)	= \$400*(1/120)/0.4219	= \$7.90
	(1.0/cm^2)	= \$400*(1/120)/0.1013	= \$32.91
Total Cost	(0.3/cm^2)	= \$148.48+\$7.90+\$20	= \$176.38
	(1.0/cm^2)	= \$612.50+\$32.91+\$20	= \$665.41

Example Question, cont'd

- **Determine cost if defect density = 0.3/cm^2 vs. 1.0/cm^2**

Dies per wafer		= 104 - 25	= 79
Good dies per wafer	(0.3/cm^2)	= 79*(0.4219)	= 33
	(1.0/cm^2)	= 79*(0.1013)	= 8
Die Cost	(0.3/cm^2)	= $4900/33	= $148.48
	(1.0/cm^2)	= $4900/8	= $612.50
Testing Cost	(0.3/cm^2)	= $400*(1/120)/0.4219	= $7.90
	(1.0/cm^2)	= $400*(1/120)/0.1013	= $32.91
Total Cost	(0.3/cm^2)	= $148.48+$7.90+$20	= $176.38
	(1.0/cm^2)	= $612.50+$32.91+$20	= $665.41

Metrics of Performance



Application

Programming Language

Compiler

ISA

Datapath

Control

Function Units

Transistors Wires Pins

Answers per month

Operations per second

(millions) of Instructions per second: MIPS

(millions) of (FP) operations per second: MFLOP/s

Megabytes per second

Cycles per second (clock rate)

Definitions

Definitions

- Performance is in units of things per sec

Definitions

- Performance is in units of things per sec
  - bigger is better

Definitions

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

Definitions

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

$$\text{Performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

Definitions

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

$$\text{Performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

" X is n times faster than Y" means

Definitions

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

$$\text{Performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

" X is n times faster than Y" means

$$n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(X)}$$

Definitions

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

$$\text{Performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

" X is n times faster than Y" means

$$n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(X)}$$

Another way of saying this: "**Speedup** of X compared to Y is n".

Alternative definitions…

- Performance = instructions / second
- Performance = FLOPS
- Performance = GHz

- Marketing numbers!

- Only consistent measure is total execution time.

Choosing Programs to Evaluate Performance

Choosing Programs to Evaluate Performance

- Real Applications

Choosing Programs to Evaluate Performance

- Real Applications
- Kernels
  - Small key piece from real program.
  - E.g., “Livermore Loops”, “Linpack”

Choosing Programs to Evaluate Performance

- <u>Real Applications</u>
- Kernels
  - Small key piece from real program.
  - E.g., “Livermore Loops”, “Linpack”
- Toy benchmarks
  - E.g., Sieve of Eratosthenes, Puzzle, Quicksort, …
  - Leave these in APSC 160/CPSC 260

Choosing Programs to Evaluate Performance

- Real Applications
- Kernels
  - Small key piece from real program.
  - E.g., “Livermore Loops”, “Linpack”
- Toy benchmarks
  - E.g., Sieve of Eratosthenes, Puzzle, Quicksort, …
  - Leave these in APSC 160/CPSC 260
- Synthetic benchmarks
  - Do not compute anything a user could want.
  - Whetstone, Dhrystone

Choosing Programs to Evaluate Performance

- <u>Real Applications</u>
- Kernels
  - Small key piece from real program.
  - E.g., “Livermore Loops”, “Linpack”
- Toy benchmarks
  - E.g., Sieve of Eratosthenes, Puzzle, Quicksort, …
  - Leave these in APSC 160/CPSC 260
- Synthetic benchmarks
  - Do not compute anything a user could want.
  - Whetstone, Dhrystone

- What are the advantages / disadvantages?

Benchmark Suites

Benchmark Suites

- What do you look for when buying a computer?

Benchmark Suites

- What do you look for when buying a computer?
- What programs do you run most often?

Benchmark Suites

- What do you look for when buying a computer?
- What programs do you run most often?

- Benchmark Suite := collection of applications used to measure performance of computer.

Benchmark Suites

- What do you look for when buying a computer?
- What programs do you run most often?

- Benchmark Suite := collection of applications used to measure performance of computer.

- Example: SPECint CPU2006 (www.spec.org)
  - 12 applications (gzip, gcc, perl, + other more exotic programs)

Benchmark Suites

- What do you look for when buying a computer?
- What programs do you run most often?

- Benchmark Suite := collection of applications used to measure performance of computer.

- Example: SPECint CPU2006 (www.spec.org)
  - 12 applications (gzip, gcc, perl, + other more exotic programs)
- Benchmarks change periodically as software gets more complex
  - e.g., SPEC2006, SPEC2000, SPEC95, SPEC92, SPEC89

Comparing and Summarizing Performance

	Computer A	Computer B	Computer C
Program P1	1 sec	10 sec	20 sec
Program P2	1000 sec	100 sec	20 sec

which computer is fastest?

Comparing and Summarizing Performance

	Computer A	Computer B	Computer C
Program P1	1 sec	10 sec	20 sec
Program P2	1000 sec	100 sec	20 sec

which computer is fastest?

- A is 10 times faster than B for program P1

Comparing and Summarizing Performance

	Computer A	Computer B	Computer C
Program P1	1 sec	10 sec	20 sec
Program P2	1000 sec	100 sec	20 sec

which computer is fastest?

- A is 10 times faster than B for program P1
- A is 20 times faster than C for program P1

Comparing and Summarizing Performance

	Computer A	Computer B	Computer C
Program P1	1 sec	10 sec	20 sec
Program P2	1000 sec	100 sec	20 sec

which computer is fastest?

- A is 10 times faster than B for program P1
- A is 20 times faster than C for program P1

- B is 10 times faster than A for program P2

Comparing and Summarizing Performance

	Computer A	Computer B	Computer C
Program P1	1 sec	10 sec	20 sec
Program P2	1000 sec	100 sec	20 sec

which computer is fastest?

- A is 10 times faster than B for program P1
- A is 20 times faster than C for program P1

- B is 10 times faster than A for program P2
- B is 2 times faster than C for program P1

Comparing and Summarizing Performance

	Computer A	Computer B	Computer C
Program P1	1 sec	10 sec	20 sec
Program P2	1000 sec	100 sec	20 sec

which computer is fastest?

- A is 10 times faster than B for program P1
- A is 20 times faster than C for program P1

- B is 10 times faster than A for program P2
- B is 2 times faster than C for program P1

- C is 50 times faster than A for program P2

Comparing and Summarizing Performance

	Computer A	Computer B	Computer C
Program P1	1 sec	10 sec	20 sec
Program P2	1000 sec	100 sec	20 sec

which computer is fastest?

- A is 10 times faster than B for program P1
- A is 20 times faster than C for program P1

- B is 10 times faster than A for program P2
- B is 2 times faster than C for program P1

- C is 50 times faster than A for program P2
- C is 5 times faster than B for program P2

Comparing and Summarizing Performance

	Computer A	Computer B	Computer C
Program P1	1 sec	10 sec	20 sec
Program P2	1000 sec	100 sec	20 sec

which computer is fastest?

- A is 10 times faster than B for program P1
- A is 20 times faster than C for program P1
- B is 10 times faster than A for program P2
- B is 2 times faster than C for program P1
- C is 50 times faster than A for program P2
- C is 5 times faster than B for program P2

Which computer would you consider "fastest overall" or "best"?

A: A
B: B
C: C
D: none is fastest
E: all are fastest

Comparing and Summarizing Performance

	Computer A	Computer B	Computer C
Program P1	1 sec	10 sec	20 sec
Program P2	1000 sec	100 sec	20 sec
Total Time	1001 sec	110 sec	40 sec

which computer is fastest?

- A is 10 times faster than B for program P1
- A is 20 times faster than C for program P1

- B is 10 times faster than A for program P2
- B is 2 times faster than C for program P1

- C is 50 times faster than A for program P2
- C is 5 times faster than B for program P2

Which computer would you consider "fastest overall" or "best"?

A: A
B: B
C: C
D: none is fastest
E: all are fastest

Comparing and Summarizing Performance

Comparing and Summarizing Performance

- Using total execution time:
  - B is 9.1 times faster than A
  - C is 25 times faster than A
  - C is 2.75 times faster than B

  => Summarize performance using average execution time:

$$\text{Average Execution Time} = \frac{1}{n}\sum_{i=1}^{n}\text{Time}_i$$

- **What if P1 and P2 are <u>not</u> run equal number of times?**

Comparing and Summarizing Performance

- Using total execution time:
  - B is 9.1 times faster than A
  - C is 25 times faster than A
  - C is 2.75 times faster than B

  => Summarize performance using average execution time:

$$\text{Average Execution Time} = \frac{1}{n}\sum_{i=1}^{n}\text{Time}_i$$

Avg(A) = 500.5

Avg(B) = 55

Avg(C) = 20

- **What if P1 and P2 are not run equal number of times?**

Comparing and Summarizing Performance

- Weighted Execution Time

$$\sum_{i=1}^{n} \text{Weight}_i \times \text{Time}_i$$

- Geometric Mean (used for SPEC 2000, SPEC 2006)

$$\text{Geometric Mean} = \sqrt[n]{\prod_{i=1}^{n} \text{Execution Time Ratio}_i}$$

$$\frac{\text{Geometric Mean}(X_1, X_2, \ldots., X_n)}{\text{Geometric Mean}(Y_1, Y_2, \ldots., Y_n)} = \text{Geometric Mean}\left(\frac{X_1}{Y_1}, \frac{X_2}{Y_2}, \ldots., \frac{X_n}{Y_n}\right)$$

Comparing and Summarizing Performance

- Weighted Execution Time

$$\sum_{i=1}^{n} \text{Weight}_i \times \text{Time}_i$$

- Geometric Mean (used for SPEC 2000, SPEC 2006)

$$\text{Geometric Mean} = \sqrt[n]{\prod_{i=1}^{n} \text{Execution Time Ratio}_i}$$

“Speedup” for benchmark “i”

$$\frac{\text{Geometric Mean}(X_1, X_2, \ldots, X_n)}{\text{Geometric Mean}(Y_1, Y_2, \ldots, Y_n)} = \text{Geometric Mean}\left(\frac{X_1}{Y_1}, \frac{X_2}{Y_2}, \ldots, \frac{X_n}{Y_n}\right)$$

Example: Weighted Execution Time

	Computers			Weightings		
	A	B	C	W(1)	W(2)	W(3)
P1	1	10	20	0.50	0.909	0.999
P2	1000	100	20	0.50	0.091	0.001
W(1)	500.5	55.00	20.00			
W(2)	91.91	18.19	20.00			
W(3)	2.00	10.09	20.00			

- Which computer (A,B, or C) is “fastest” depends upon weighting of program mix.

Geometric vs. Arithmetic Mean

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arith Mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geom Mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total Time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

Geometric vs. Arithmetic Mean

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arith Mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geom Mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total Time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

- Geometric mean => C fastest regardless of which machine we normalize to. Consistent regardless of “base machine”.

Geometric vs. Arithmetic Mean

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arith Mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geom Mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total Time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

- Geometric mean => C fastest regardless of which machine we normalize to. Consistent regardless of “base machine”.
- Drawback of Geom. Mean: Does not predict execution time.

Geometric vs. Arithmetic Mean

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arith Mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geom Mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total Time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

- Geometric mean => C fastest reg normalize to. <u>Consistent regardle</u>
- Drawback of Geom. Mean: Does

What are we comparing when we use "average performance"?

A: One program versus another program across a set of different computers

B: One computer versus another computer across a set of different programs.

C: Not sure

Geometric vs. Arithmetic Mean

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arith Mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geom Mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total Time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

- Geometric mean => C fastest regardless of which machine we normalize to. Consistent regardless of “base machine”.
- Drawback of Geom. Mean: Does not predict execution time.

Geometric vs. Arithmetic Mean

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arith Mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geom Mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total Time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

- Geometric mean => C fastest reg normalize to. <u>Consistent regardle</u>
- Drawback of Geom. Mean: Does

What are we comparing when we use "average performance"?

A: One program versus another program across a set of different computers

B: One computer versus another computer across a set of different programs. ✓

C: Not sure

Geometric vs. Arithmetic Mean

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.0	10.0	20.0	0.1	1.0	2.0	0.05	0.5	1.0
Program P2	1.0	0.1	0.02	10.0	1.0	0.2	50.0	5.0	1.0
Arith Mean	1.0	5.05	10.01	5.05	1.0	1.1	25.03	2.75	1.0
Geom Mean	1.0	1.0	0.63	1.0	1.0	0.63	1.58	1.58	1.0
Total Time	1.0	0.11	0.04	9.1	1.0	0.36	25.03	2.75	1.0

- Geometric mean => C fastest regardless of which machine we normalize to. Consistent regardless of “base machine”.
- Drawback of Geom. Mean: Does not predict execution time.

Comparing and Summarizing Performance

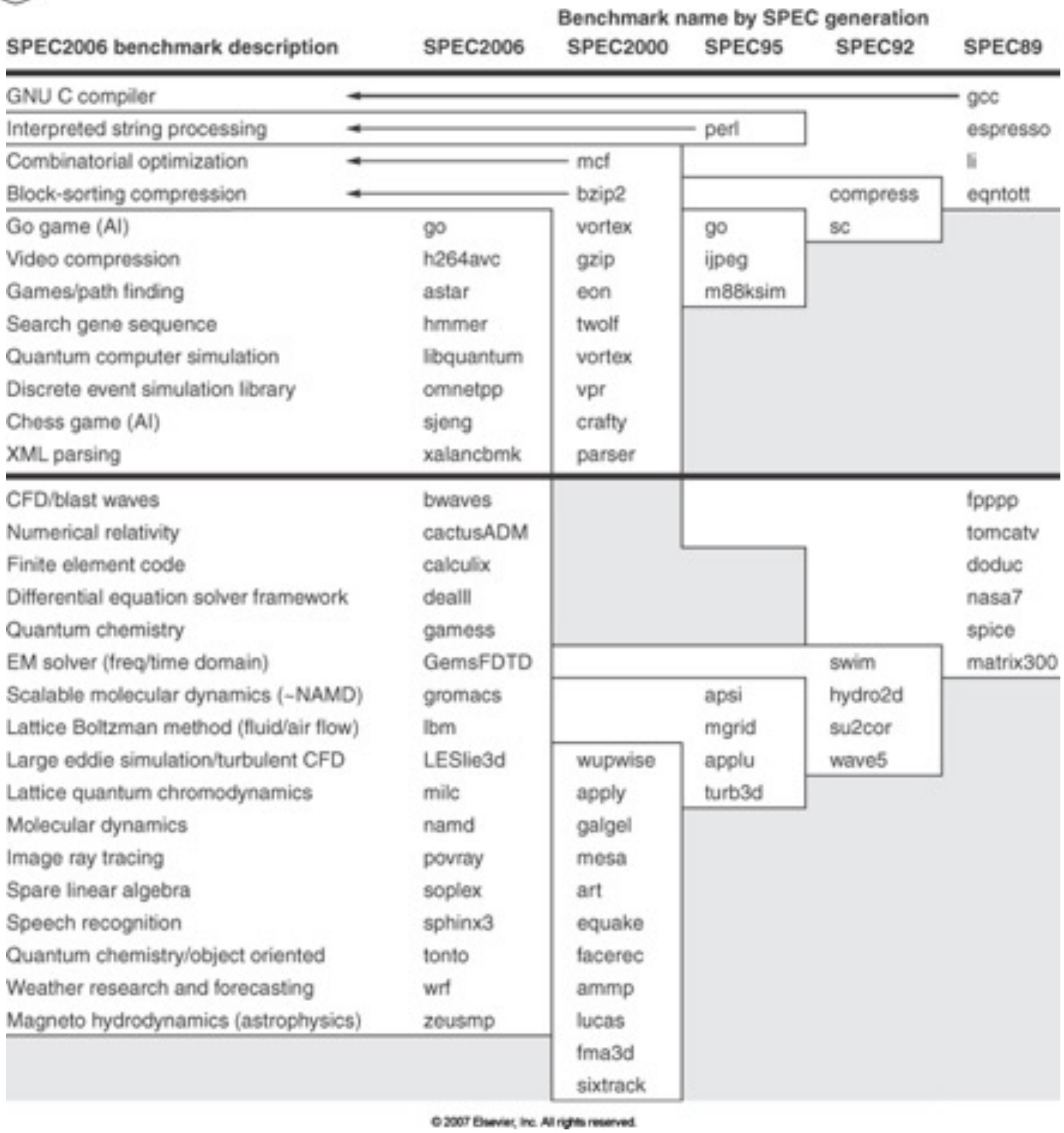
- Harmonic Mean very popular in Computer Architecture Research (HM of “rates” tracks execution time)

$$\text{Harmonic Mean} = \frac{n}{\sum \frac{1}{\text{Execution Time Ratio}_i}}$$

- Mathematical relationship:

  Harmonic Mean ≤ Geometric Mean ≤ Arithmetic Mean

SPEC Benchmark Evolution



SPEC2006 benchmark description	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
		Benchmark name by SPEC generation			
GNU C compiler					gcc
Interpreted string processing			perl		espresso
Combinatorial optimization		mcf			li
Block-sorting compression		bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzman method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

- Changes reflect changes in computer usage with time.
- Uses geometric mean speedup.

Controversy Over?

- Daniel Citron et al., “The Harmonic or Geometric Mean: Does it Really Matter?” ACM Computer Architecture News, vol.34, no. 4, Sept. 2006
- Found that rankings based upon harmonic mean are very close to those based upon geometric mean (for SPEC 2000).

- Recommend using harmonic mean of “speedup” if considering design alternatives when designing a microprocessor. “Outliers” less likely with real machines studied in above paper but very likely during early stages of microprocessor design (which would cause larger difference in geometric versus harmonic mean).

Quantitative Principles of Computer Design

“Make the Common Case Fast”

If you keep doing something over and over…
find a clever way to make it faster.

Original ketchup bottle (bottle on left) hard to get ketchup out of.

Store “upside down” so ketchup always ready to come out (bottle on right).

Quantitative Principl

Two engineers arguing. One says improve floating-point unit by 40x for division. Another says improve FP unit by 1.5x for all floating point operations. Have to pick one.

A: Speedup division by a factor of 40x
B: Speedup all FP operations by a
    factor of 1.5x
C: Not sure

“Make the Common C

If you keep doing some
find a clever way to make it faster.

Original ketchup bottle (bottle on left) hard to get ketchup out of.

Store “upside down” so ketchup always ready to come out (bottle on right).

Quantitative Principles of Computer Design

“Make the Common Case Fast”

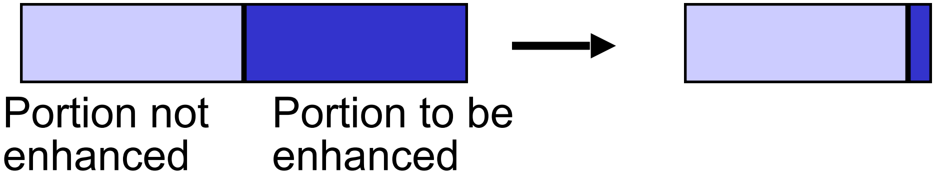
If you keep doing something over and over…
find a clever way to make it faster.

Original ketchup bottle (bottle on left) hard to get ketchup out of.

Store “upside down” so ketchup always ready to come out (bottle on right).

Very Important: **Amdahl's Law**

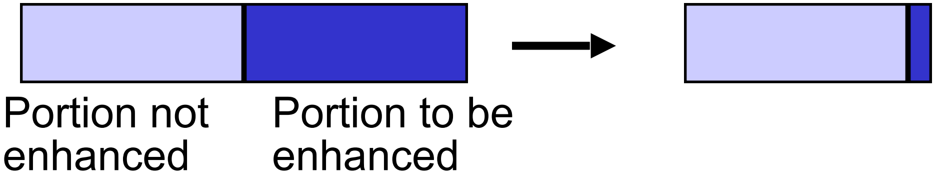
In 1967, Gene Amdahl examined question of whether it makes sense to develop parallel processors. Argued that it was very important to focus on a single processor (i.e., "core") since could never get rid of portion of code that could not be parallelized.



$$\mathbf{ExTime_{new}} = \mathbf{ExTime_{old}} \times \left[\left(1 - \mathbf{Fraction_{enhanced}}\right) + \frac{\mathbf{Fraction_{enhanced}}}{\mathbf{Speedup_{enhanced}}} \right]$$

Very Important: **Amdahl's Law**

In 1967, Gene Amdahl examined question of whether it makes sense to develop parallel processors. Argued that it was very important to focus on a single processor (i.e., "core") since could never get rid of portion of code that could not be parallelized.

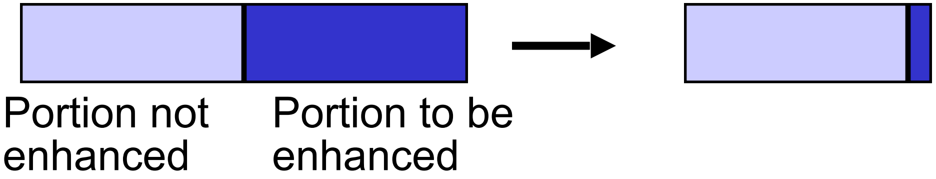


$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left[(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right]$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Very Important: **Amdahl's Law**

In 1967, Gene Amdahl examined question of whether it makes sense to develop parallel processors. Argued that it was very important to focus on a single processor (i.e., "core") since could never get rid of portion of code that could not be parallelized.



$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left[\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right]$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Best you could ever hope to do:

$$\text{Speedup}_{\text{maximum}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right)}$$

Example:
Floating Point (FP) Square Root (FPQSR)

- 20% of $ExTime_{old}$ due to FPSQR
- 50% of $ExTime_{old}$ due to all FP operations.
- Two alternatives:
  - Speedup FPSQR by a factor of 10
  - Speedup all FP operations by a factor of 1.6
- Which is better?

Example:
Floating Point (FP) Square Root (FPQSR)

- 20% of ExTime$_{old}$ due to FPSQR
- 50% of ExTime$_{old}$ due to all FP operations.
- Two alternatives:
  - Speedup FPSQR by a factor of 10
  - Speedup all FP operations by a factor of 1.6
- Which is better?

$$\text{Speedup}_{FPSQR} = \frac{1}{(1-0.2)+\frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{Speedup}_{FP} = \frac{1}{(1-0.5)+\frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

Example Question

- Three possible enhancements:
  - $\text{Speedup}_1 = 30$
  - $\text{Speedup}_2 = 20$
  - $\text{Speedup}_3 = 15$
- If E_1 and E_2 each usable 25% of time, what fraction must E_3 be used to achieve overall speedup of 10?

Example Question

- Three possible enhancements:
  - $\text{Speedup}_1 = 30$
  - $\text{Speedup}_2 = 20$
  - $\text{Speedup}_3 = 15$
- If E_1 and E_2 each usable 25% of time, what fraction must E_3 be used to achieve overall speedup of 10?

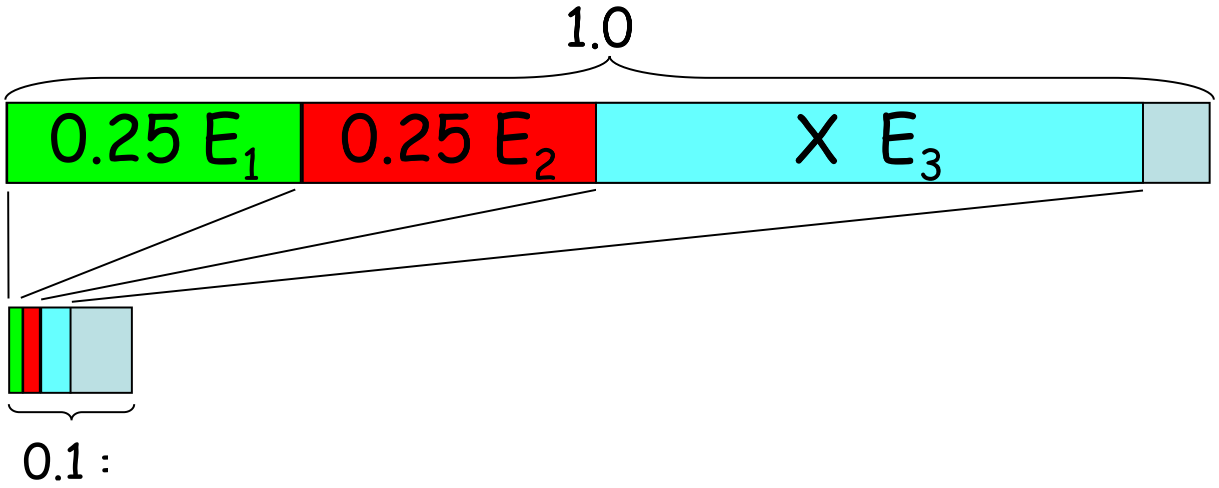
What fraction must E_3 be?

A: Figured out equation, need calculator

B: No clue how to start

Example Question

- Three possible enhancements:
  - $\text{Speedup}_1 = 30$
  - $\text{Speedup}_2 = 20$
  - $\text{Speedup}_3 = 15$
- If E_1 and E_2 each usable 25% of time, what fraction must E_3 be used to achieve overall speedup of 10?



What fraction must E_3 be?

A: Figured out equation, need calculator

B: No clue how to start

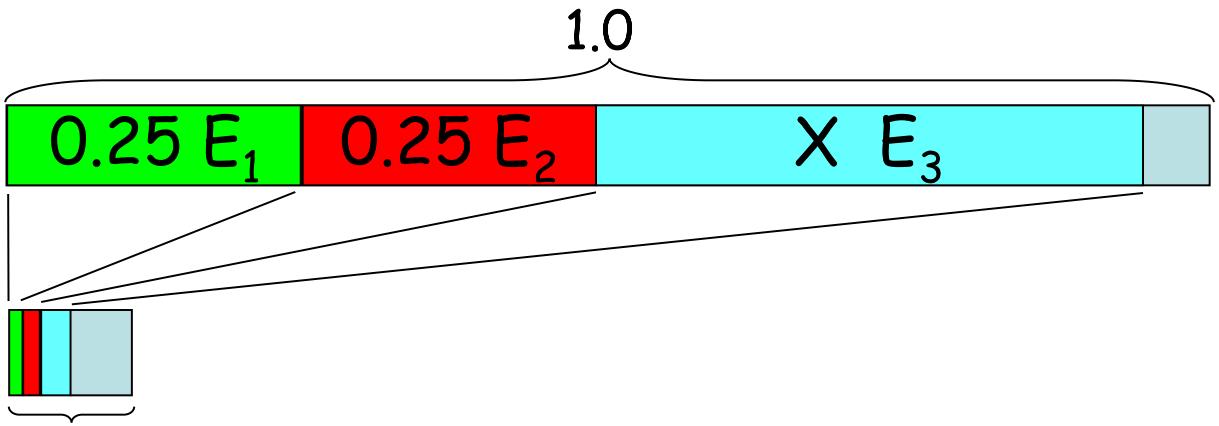
Example Question

What fraction must E_3 be?

A: Figured out equation, need calculator

B: No clue how to start

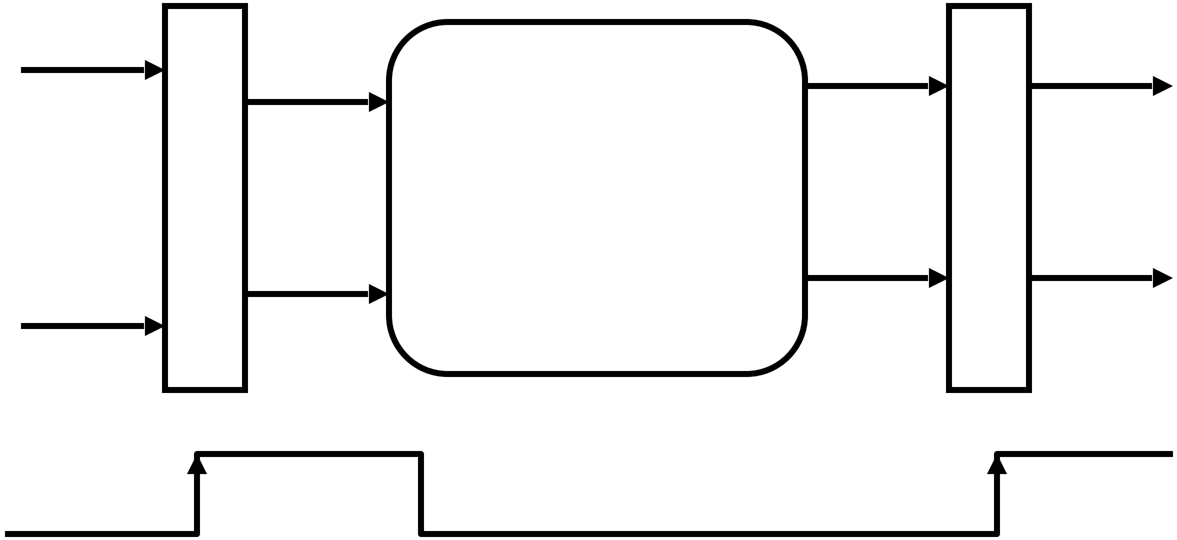
- Three possible enhancements:
  - $\text{Speedup}_1 = 30$
  - $\text{Speedup}_2 = 20$
  - $\text{Speedup}_3 = 15$
- If E_1 and E_2 each usable 25% of time, what fraction must E_3 be used to achieve overall speedup of 10?



$0.1 = 0.25*(1/30) + 0.25*(1/20) + X*(1/15) + (0.5-X)$

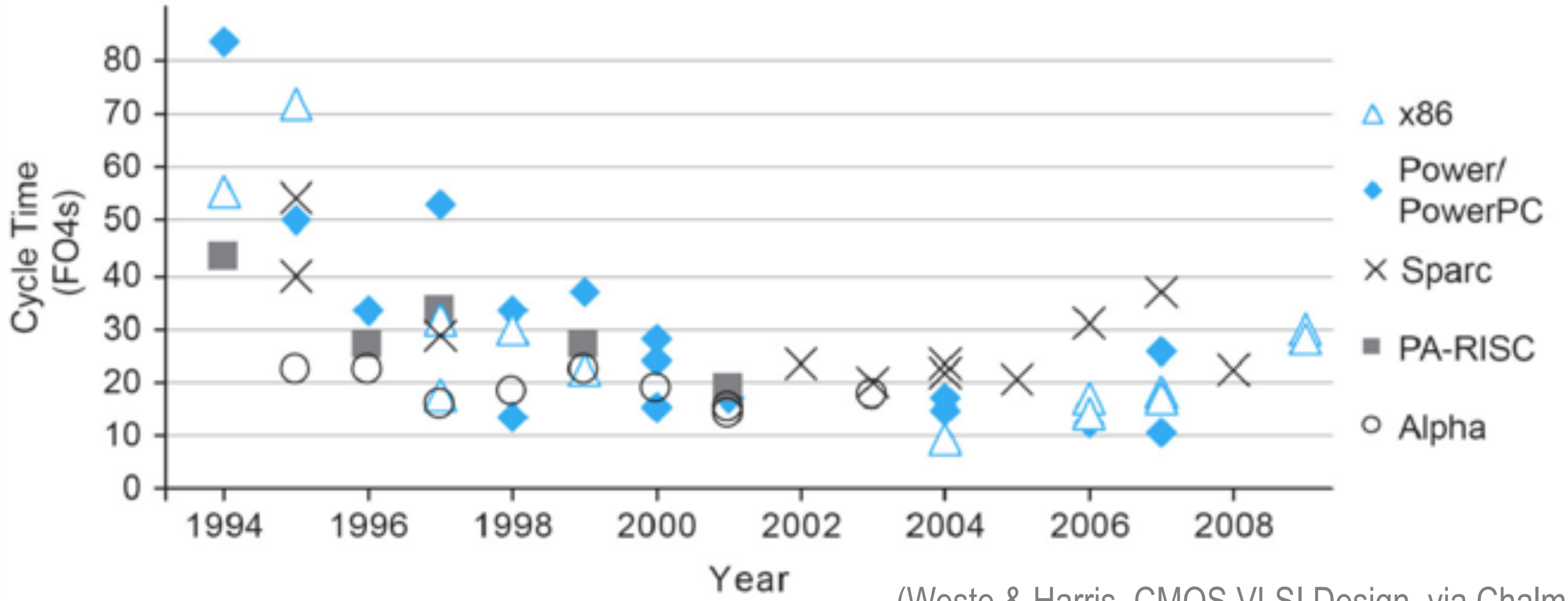
Solve to get: $X = 0.45$

What is a “Clock Cycle”?



- Old days: 10 levels of gates
- Today: determined by numerous time-of-flight issues + gate delays
  - clock propagation, wire lengths, drivers

What is a “Clock Cycle”?



(Weste & Harris, CMOS VLSI Design, via Chalmers)

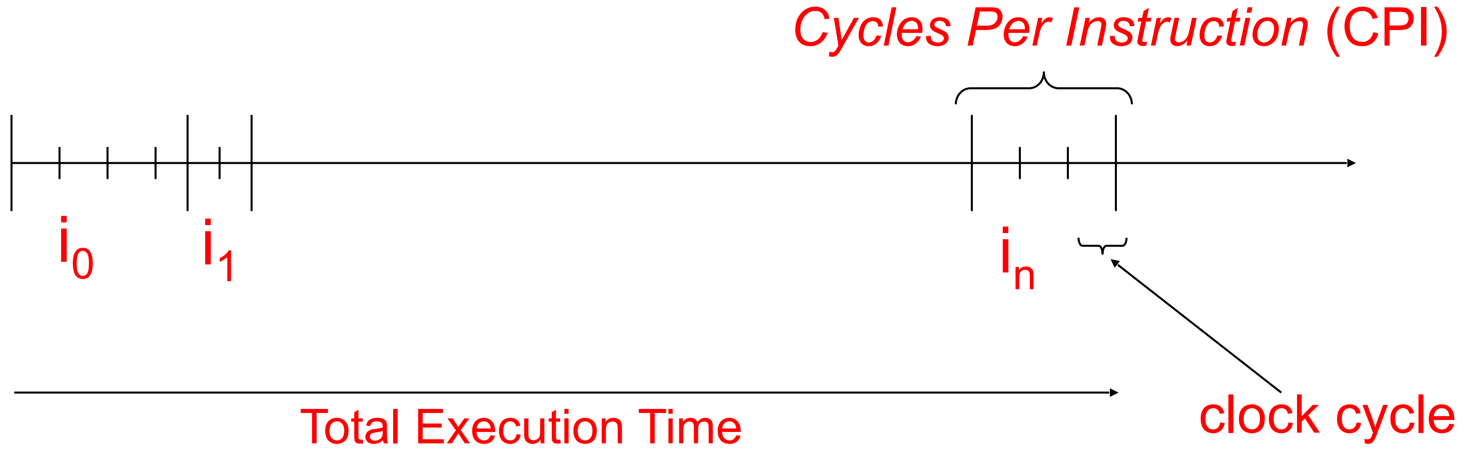
- FO4 (fan-out-4 delay)
- delay of an inverter
  - driven by inverter 4x smaller
  - drives inverter 4x bigger

Processor Performance Equation

(“Iron Law” of computer performance)

$$\frac{\text{CPU time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

$$\text{Execution Time} = \frac{1}{\text{Performance}} = (\text{Instr. Count}) \times (\text{CPI}) \times (\text{cycle time})$$



Computing Cycles Per Instruction

The CPI in the processor performance equation refers to the average cycles per instruction across all instructions executed by a program.

CPI = Cycles / Instruction Count
= (CPU Time * Clock Rate) / Instruction Count (1)

If different instructions take a different number of cycles, then we can also express "CPU Time" as:

$$\text{CPU Time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times I_j \qquad (2)$$

Where: I_j = Instruction count for instruction of type "j"
CPI_j = cycles per instruction for instruction of type j

Then, we can substitute (2) into (1) to obtain:

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times F_j \quad \text{where } F_j = \frac{I_j}{\text{Instruction Count}}$$

Here F_j is the normalized instruction frequency for instructions of type j

Example CPI Calculation

After graduating, at a small startup company you are asked to create a “soft core” processor that will be implemented on an FPGA to run specialized software.

You consider a particular way of optimizing the way one of the instructions is implemented. You implement your processor both “with” and “w/o” this optimization and measure:

- $\text{cycle\_time}_{\text{“with”}} = 1.05*\text{cycle\_time}_{\text{”w/o”}}$
- $\text{IC}_{\text{”with”}} = 0.99*\text{IC}_{\text{”w/o”}}$
- $\text{CPI}_{\text{”with”}} = 1.01*\text{CPI}_{\text{”w/o”}}$

Should you use the optimization in your processor?

Example CPI Calculation

After graduating, at a small startup company you are asked to create a “soft core” processor that will be implemented on an FPGA to run specialized software.

You consider a particular way of optimizing the way one of the instructions is implemented. You implement your processor both “with” and “w/o” this optimization and measure:

- $\text{cycle\_time}_{\text{“with”}} = 1.05*\text{cycle\_time}_{\text{”w/o”}}$
- $\text{IC}_{\text{”with”}} = 0.99*\text{IC}_{\text{”w/o”}}$
- $\text{CPI}_{\text{”with”}} = 1.01*\text{CPI}_{\text{”w/o”}}$

Should you use the optimization in your processor?

Speedup of processor with optimization compared to to processor without optimization is:

A: 1.05
B: 0.95
C: 1.10
D: 0.90
E: Not sure

Example CPI Calculation

After graduating, at a small startup company you are asked to create a “soft core” processor that will be implemented on an FPGA to run specialized software.

You consider a particular way of optimizing the way one of the instructions is implemented. You implement your processor both “with” and “w/o” this optimization and measure:

- cycle_time$_{\text{“with”}}$ = 1.05*cycle_time$_{\text{”w/o”}}$
- IC$_{\text{”with”}}$ = 0.99*IC$_{\text{”w/o”}}$
- CPI$_{\text{”with”}}$ = 1.01*CPI$_{\text{”w/o”}}$

Should you use the optimization in your processor?

Speedup of processor with optimization compared to to processor without optimization is:

A: 1.05
B: 0.95 ✔
C: 1.10
D: 0.90
E: Not sure

Example CPI Calculation

$$\text{Speedup}_{\text{"with vs. w/o"}} = \frac{\text{Time}_{\text{"w/o"}}}{\text{Time}_{\text{"with"}}} = \frac{\text{IC}_{\text{"w/o"}} \times \text{CPI}_{\text{"w/o"}} \times \text{cycle\_time}_{\text{"w/o"}}}{\text{IC}_{\text{"with"}} \times \text{CPI}_{\text{"with"}} \times \text{cycle\_time}_{\text{"with"}}}$$

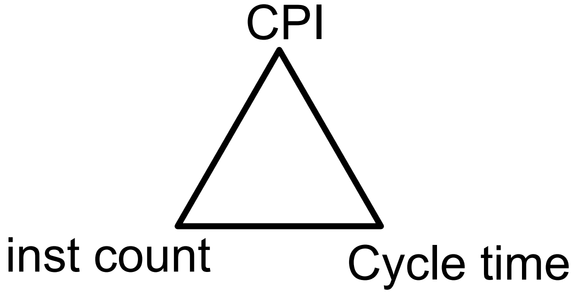
$$= \frac{\text{IC}_{\text{"w/o"}} \times \text{CPI}_{\text{"w/o"}} \times \text{cycle\_time}_{\text{"w/o"}}}{0.99 \times \text{IC}_{\text{"w/o"}} \times 1.01 \times \text{CPI}_{\text{"w/o"}} \times 1.05 \times \text{cycle\_time}_{\text{"w/o"}}}$$

$$= 0.95$$

Performance is ~5% better without this optimization.

Computer Performance

Triangle (at right) is reminder that often when we try to reduce one factor in the processor performance equation another factor increases.



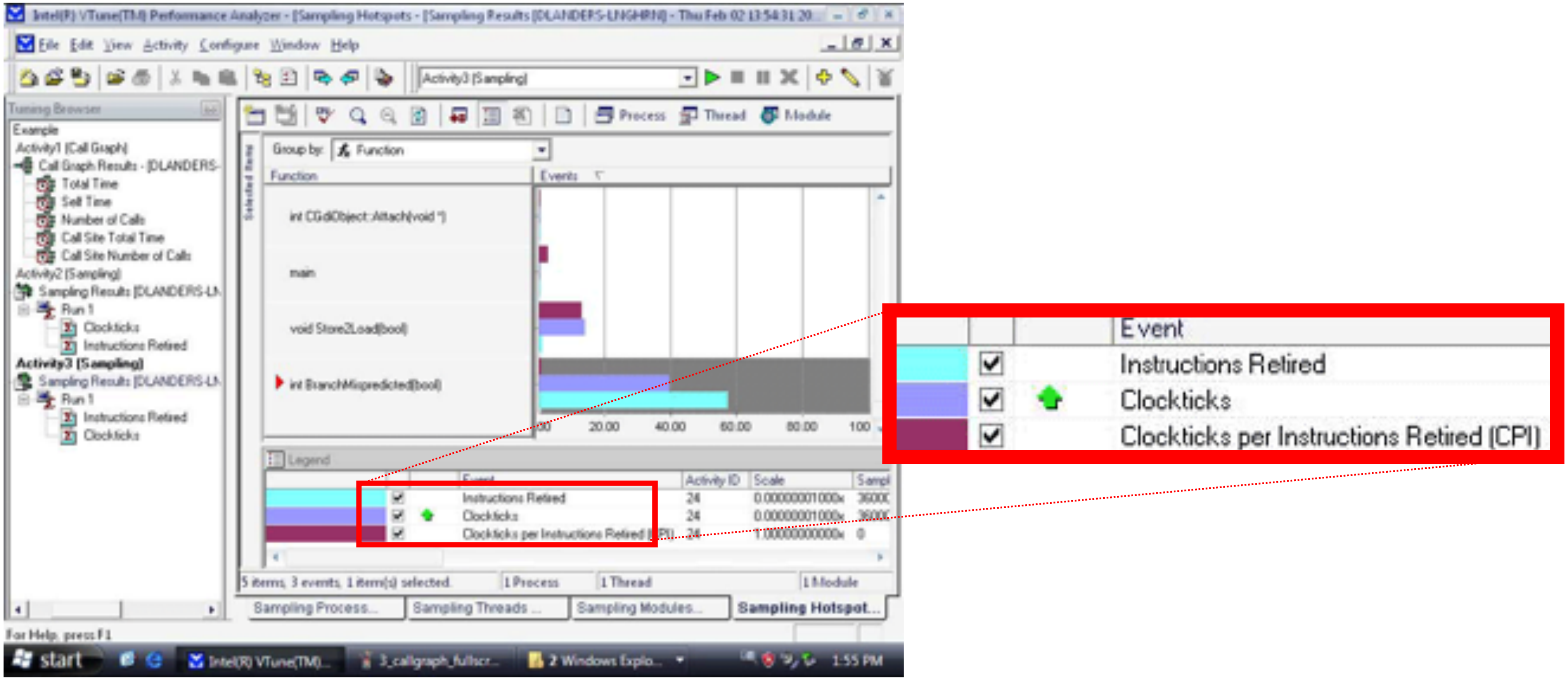
$$\textbf{CPU time} = \frac{\textbf{Seconds}}{\textbf{Program}} = \frac{\textbf{Instructions}}{\textbf{Program}} \times \frac{\textbf{Cycles}}{\textbf{Instruction}} \times \frac{\textbf{Seconds}}{\textbf{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X	X	
Compiler	X	X	
Inst. Set.	X	X	
Micro Arch.		X	X
Technology		X	X

How Do You Measure CPI?

(on real hardware)

- Modern processors contain hardware “performance counters”
  - Can read using special instructions / developer tools
    - Intel VTune Performance Analyzer
    - AMD CodeAnalyst Performance Analyzer



			Event	Activity ID	Scale	Sampl
🟦	☑		Instructions Retired	24	0.000000001000x	3600C
🟪	☑	●	Clockticks	24	0.000000001000x	3600C
🟥	☑		Clockticks per Instructions Retired (CPI)	24	1.000000000000x	0

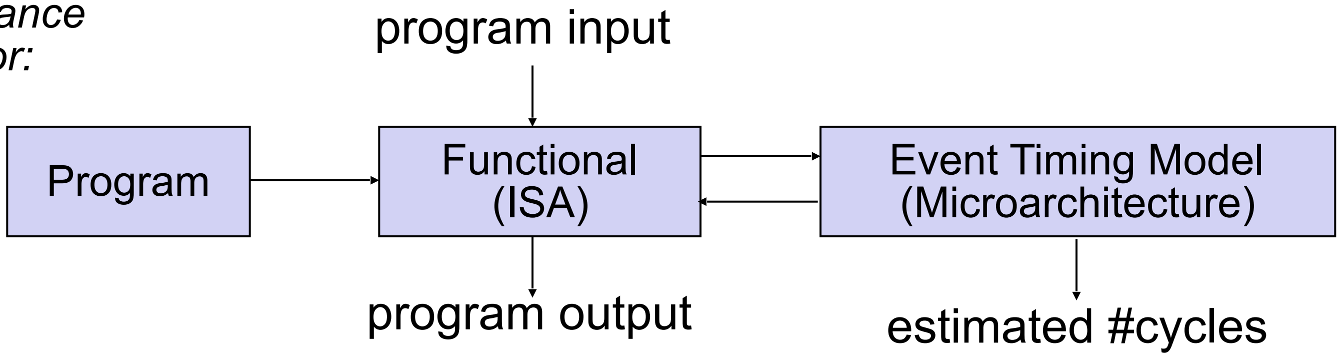
			Event
🟦	☑		Instructions Retired
🟪	☑	●	Clockticks
🟥	☑		Clockticks per Instructions Retired (CPI)

How Do You Measure CPI?

(when designing a microprocessor)

- Functional Simulator (C/C++)
  - Emulate one instruction at a time.
  - Measure F_i then use CPI equations (not very accurate)
- Performance Simulator (C/C++)
  - Create a “timing model” to capture when stalls occur
  - Not exact, but accurate enough for design exploration
- RTL Model (VHDL, Verilog) - EECE 353/379; EECE 479
  - Precise measure of CPI
  - Very slow for large processors

Performance Simulator:



How Do You

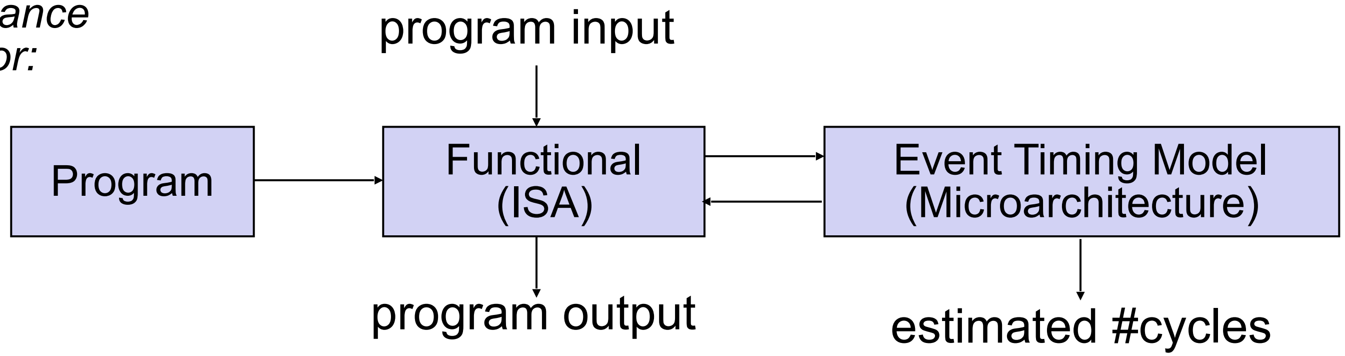
(when designin

How accurate does a “performance simulator” need to predict performance to be to be useful when architecting a microprocessor:

A: Must predict number of cycles exactly
B: 1% max error
C: 5% max error
D: 10% max error
E: Not sure

- Functional Simulator (C/C++)
  - Emulate one instruction at a
  - Measure F_i then use CPI eq
- Performance Simulator (C/C+
  - Create a “timing model” to capture when stalls occur
  - Not exact, but accurate enough for design exploration
- RTL Model (VHDL, Verilog) - EECE 353/379; EECE 479
  - Precise measure of CPI
  - Very slow for large processors

Performance Simulator:



How Do You

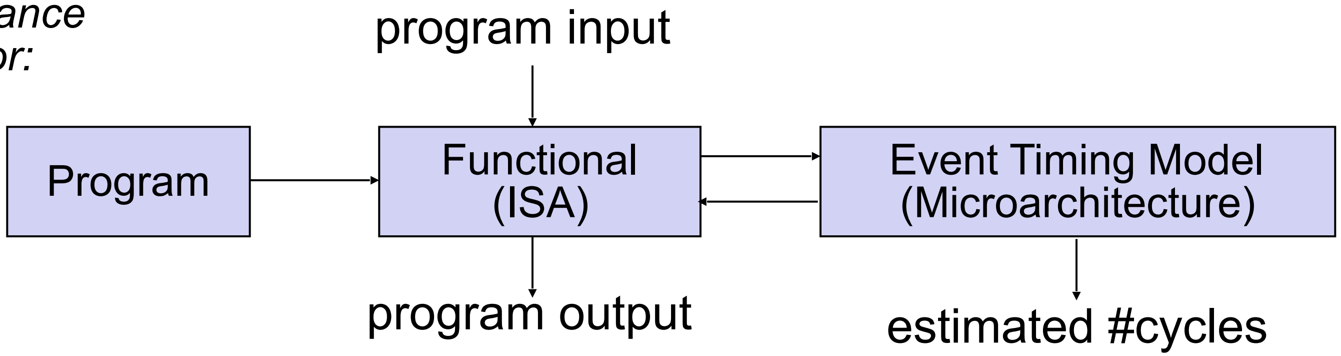
(when designin

How accurate does a “performance simulator” need to predict performance to be to be useful when architecting a microprocessor:

A: Must predict number of cycles exactly
B: 1% max error
C: 5% max error ✔ (5-10% is typical goal
D: 10% max error ✔ in industry)
E: Not sure

- Functional Simulator (C/C++)
  - Emulate one instruction at a
  - Measure F_i then use CPI eq
- Performance Simulator (C/C
  - Create a “timing model” to capture when stalls occur
  - Not exact, but accurate enough for design exploration
- RTL Model (VHDL, Verilog) - EECE 353/379; EECE 479
  - Precise measure of CPI
  - Very slow for large processors

Performance Simulator:



Industry Practice

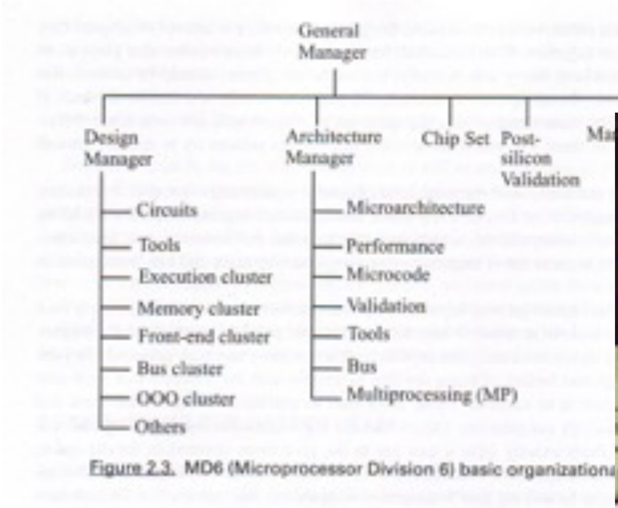
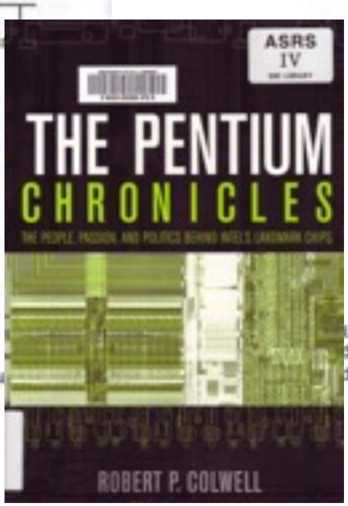
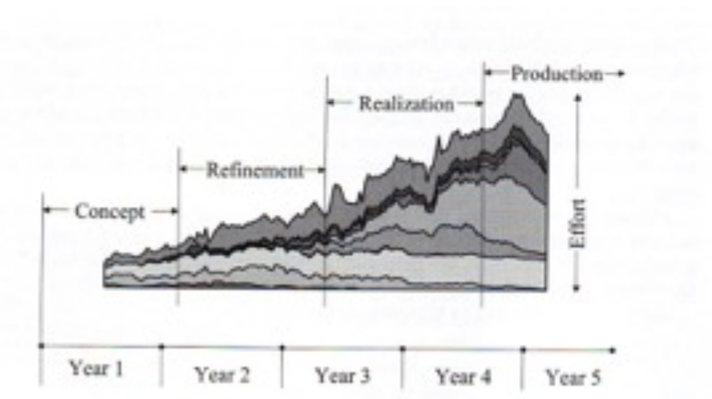


Figure 2.3. MD6 (Microprocessor Division 6) basic organizationa


ASRS
IV
THE PENTIUM
CHRONICLES
THE PEOPLE, PASSION, AND POLITICS BEHIND INTEL'S LANDMARK CHIPS
ROBERT P. COLWELL



re 1.1. Concept, refinement, realization, production, and team size. Different des of gray indicate staffing of various design activities such as circuit design, dation, layout, RTL development, project overhead, and so on.

Take Advantage of Parallelism

Another fundamental principle of computer design is to “take advantage of parallelism”.

Much of this course explores how parallelism is uncovered and exploited in modern microprocessors.

There are many different levels of parallelism:

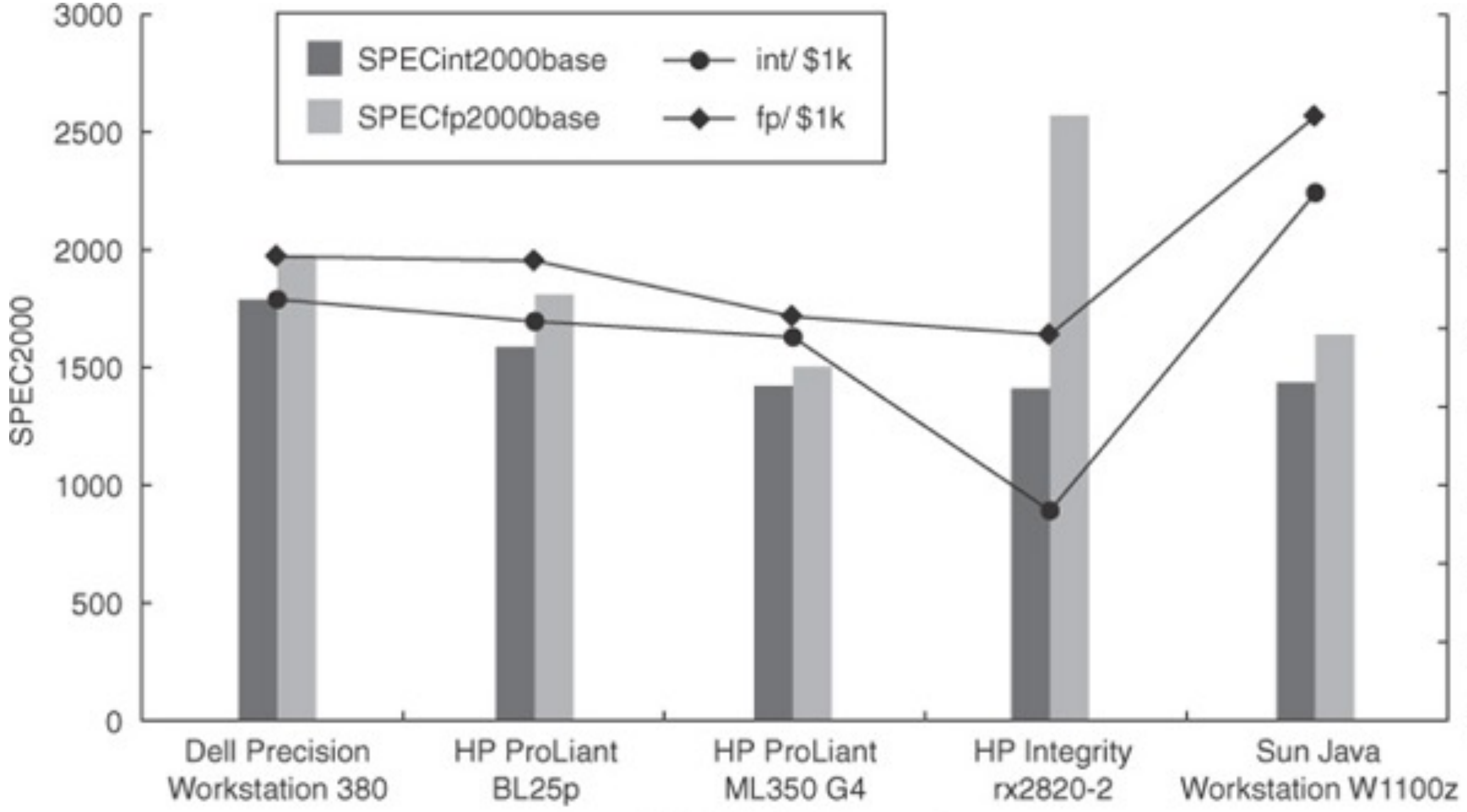
- Independent programs can run on different processors. This is known as “thread level parallelism” (TLP).
- Multiple instances of the *same* program can operate on different input data. This is known as “data level parallelism” (DLP).
- Instructions may be independent. A significant focus of computer architecture over the past 25 years has been exploiting “instruction level parallelism” (ILP)
- Different parts of a digital circuit operate in parallel during clock cycle (e.g., different “processes” in VHDL “architecture”)

Principle of Locality

Real programs are seldom completely “random”. They have structure and purpose. As a result of this, they behave in ways that are often far more “predictable” than the programmer creating the program might suspect. In particular, programs exhibit a property known as “locality”:

- Example: Programs spend 90% of time executing 10% of code.
- There are many other forms of locality that have been observed and that are exploited in microprocessors. We will see several.

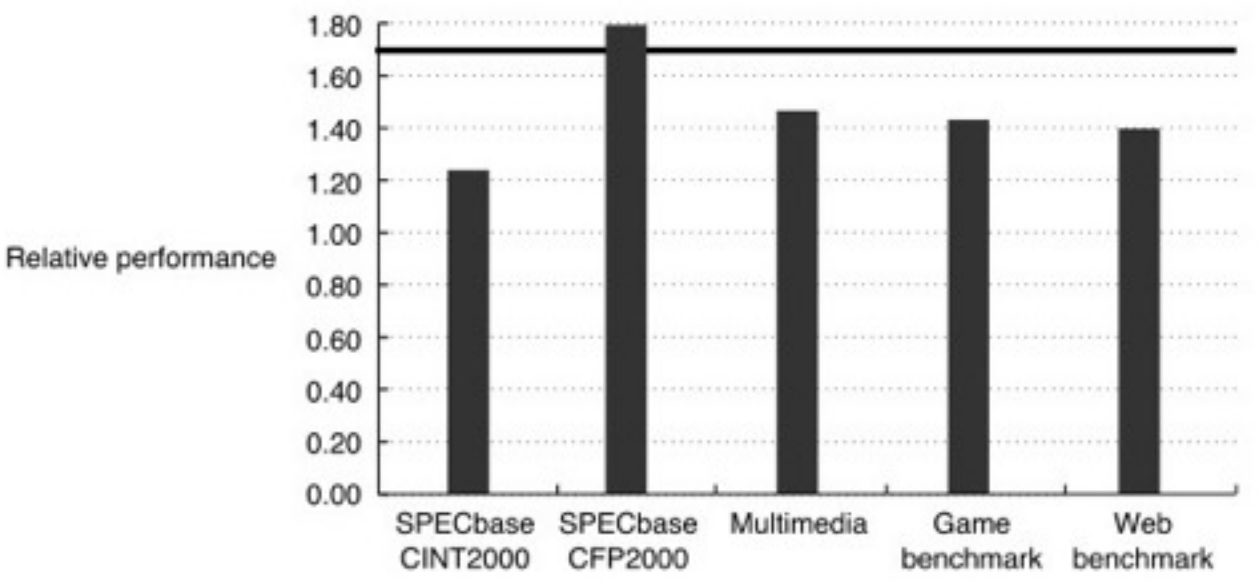
!/$ (Bang for the Buck) [Price-Performance]



© 2007 Elsevier, Inc. All rights reserved.

Fallacies and Pitfalls…

- Fallacy: Relative performance can be judged by performance on a single benchmark suite:



- Perf. of Pentium 4 (1.7GHz) versus PIII (1.0 GHz)

Fallacies and Pitfalls…

- Pitfall: Falling Prey to Amdahl's Law.
- Fallacy: Benchmarks remain valid indefinitely.
- Pitfall: Comparing hand-coded assembly and compiler-generated, high-level language performance.
- Fallacy: Peak performance tracks observed performance.
- Fallacy: The best design for a computer optimizes the primary objective w/o considering implementation.
- Pitfall: Ignoring cost of software.
- Fallacy: Synthetic Benchmarks predict performance for real programs.
- Fallacy: MIPS (millions of instructions per second) is an accurate measure for comparing performance among computers.

Summary

- In this (long) slide set we explored the fundamentals of computer architecture. We focused in particular on the quantitative principles of computer architecture.

- In the next slide set, we will apply quantitative principles when considering how best to design an instruction set architecture. Among other things, this will help us see why MIPS64, which we also learned about in this slide set, is a good instruction set.
- In later slide sets, we will see how quantitative principles have influence the hardware microarchitecture of today’s microprocessors.