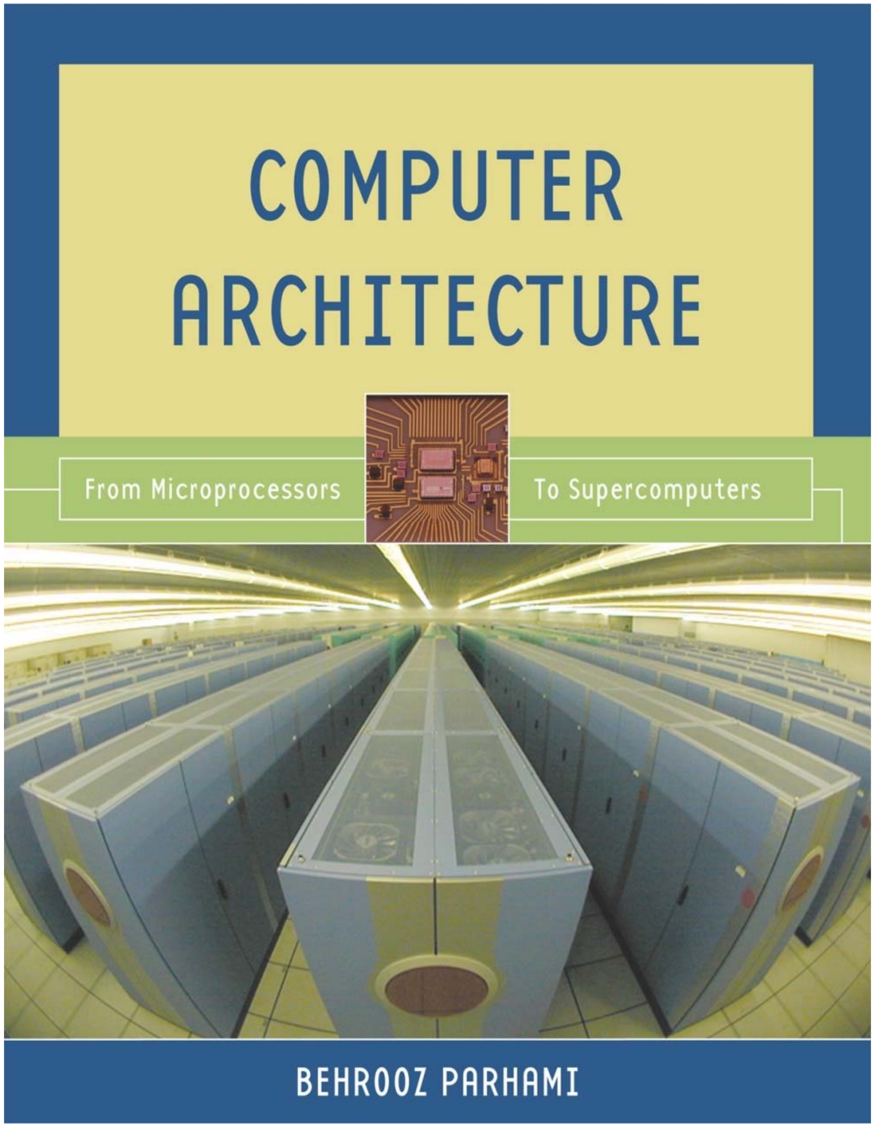
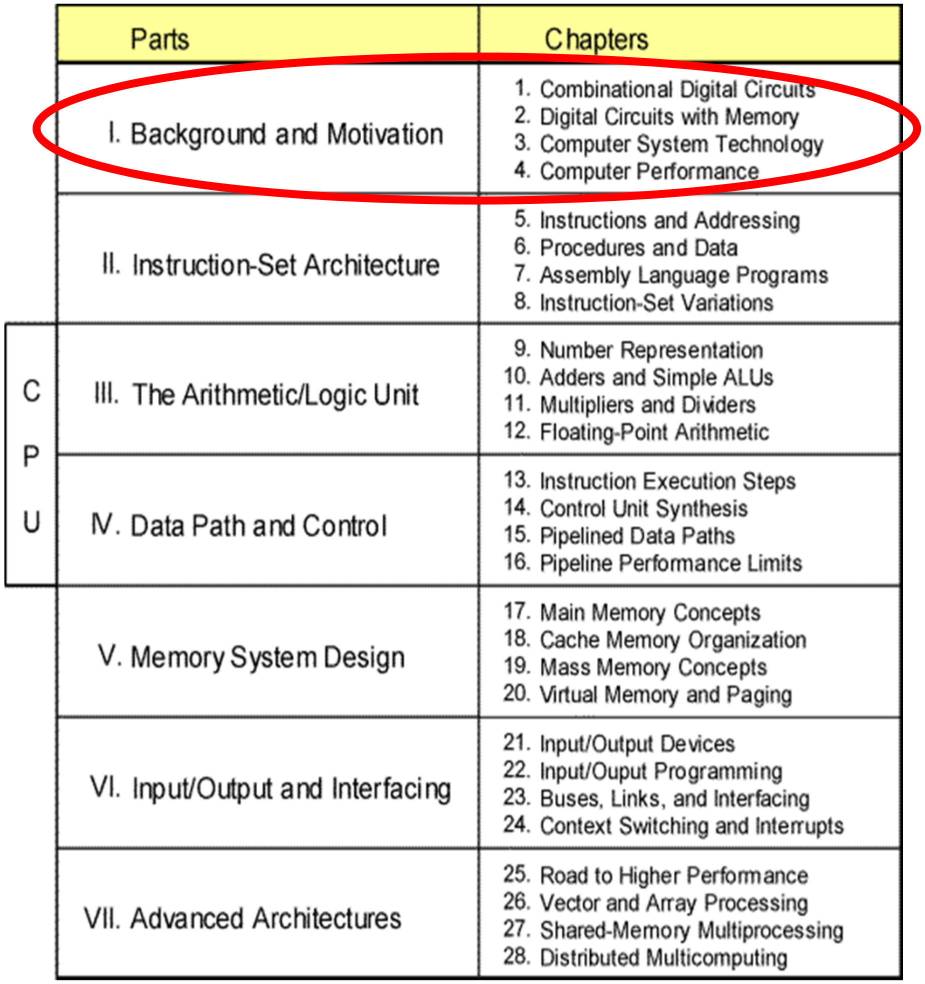


Part I
Background and Motivation



	Parts	Chapters
	I. Background and Motivation	1. Combinational Digital Circuits 2. Digital Circuits with Memory 3. Computer System Technology 4. Computer Performance
	II. Instruction-Set Architecture	5. Instructions and Addressing 6. Procedures and Data 7. Assembly Language Programs 8. Instruction-Set Variations
CPU	III. The Arithmetic/Logic Unit	9. Number Representation 10. Adders and Simple ALUs 11. Multipliers and Dividers 12. Floating-Point Arithmetic
	IV. Data Path and Control	13. Instruction Execution Steps 14. Control Unit Synthesis 15. Pipelined Data Paths 16. Pipeline Performance Limits
	V. Memory System Design	17. Main Memory Concepts 18. Cache Memory Organization 19. Mass Memory Concepts 20. Virtual Memory and Paging
	VI. Input/Output and Interfacing	21. Input/Output Devices 22. Input/Ouput Programming 23. Buses, Links, and Interfacing 24. Context Switching and Interrupts
	VII. Advanced Architectures	25. Road to Higher Performance 26. Vector and Array Processing 27. Shared-Memory Multiprocessing 28. Distributed Multicomputing

About This Presentation

This presentation is intended to support the use of the textbook *Computer Architecture: From Microprocessors to Supercomputers*, Oxford University Press, 2005, ISBN 0-19-515455-X. It is updated regularly by the author as part of his teaching of the upper-division course ECE 154, Introduction to Computer Architecture, at the University of California, Santa Barbara. Instructors can use these slides freely in classroom teaching and for other educational purposes. Any other use is strictly prohibited. © Behrooz Parhami

Edition	Released	Revised	Revised	Revised	Revised
First	June 2003	July 2004	June 2005	Mar. 2006	Jan. 2007
		Jan. 2008	Jan. 2009	Jan. 2011	Oct. 2014
Second					

I Background and Motivation

Provide motivation, paint the big picture, introduce tools:

- Review components used in building digital circuits
- Present an overview of computer technology
- Understand the meaning of computer performance

(or why a 2 GHz processor isn’t 2× as fast as a 1 GHz model)

Topics in This Part	
Chapter 1	Combinational Digital Circuits
Chapter 2	Digital Circuits with Memory
Chapter 3	Computer System Technology
Chapter 4	Computer Performance

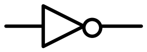
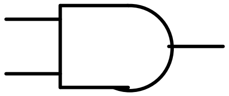
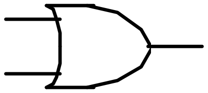
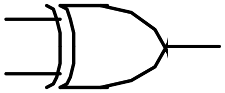
1 Combinational Digital Circuits

First of two chapters containing a review of digital design:

- Combinational, or memoryless, circuits in Chapter 1
- Sequential circuits, with memory, in Chapter 2

Topics in This Chapter	
1.1	Signals, Logic Operators, and Gates
1.2	Boolean Functions and Expressions
1.3	Designing Gate Networks
1.4	Useful Combinational Parts
1.5	Programmable Combinational Parts
1.6	Timing and Circuit Considerations

1.1 Signals, Logic Operators, and Gates

Name	NOT	AND	OR	XOR
Graphical symbol				
Operator sign and alternate(s)	x' $\neg x$ or $\overline{x}$	xy $x \wedge y$	$x \vee y$ $x + y$	$x \oplus y$ $x \not\equiv y$
Output is 1 iff:	Input is 0	Both inputs are 1s	At least one input is 1	Inputs are not equal
Arithmetic expression	$1 - x$	$x \times y$ or xy	$x + y - xy$	$x + y - 2xy$

Add: 1 + 1 = 10

Figure 1.1 Some basic elements of digital logic circuits, with operator signs used in this book highlighted.

The Arithmetic Substitution Method

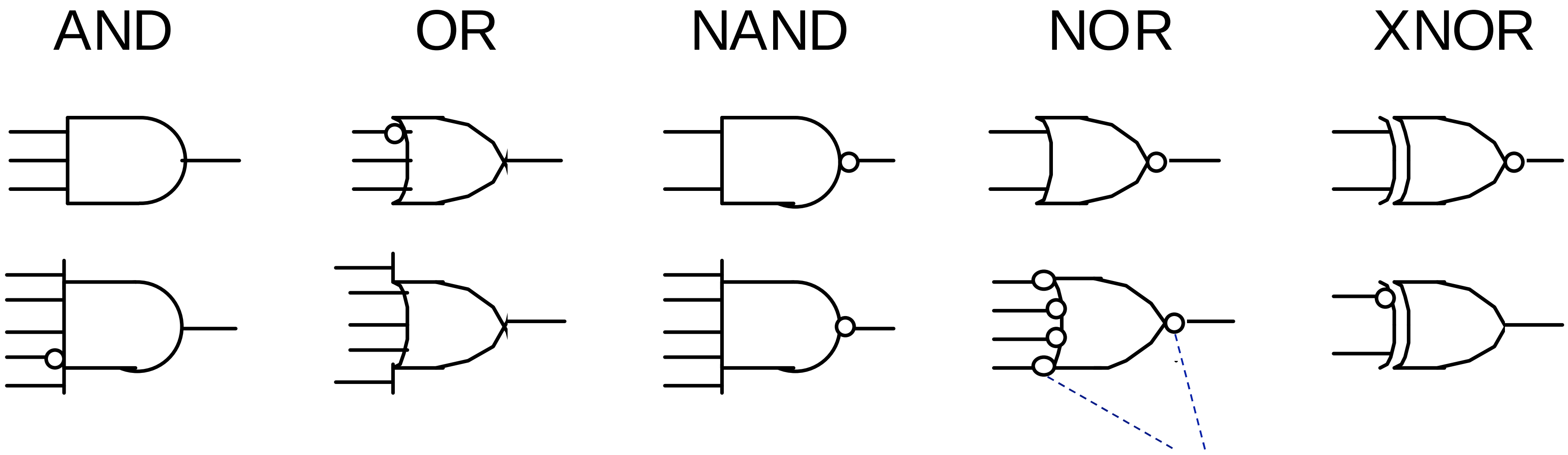
$z' = 1 - z$	NOT converted to arithmetic form
xy	AND same as multiplication
	(when doing the algebra, set $z^k = z$)
$x \vee y = x + y - xy$	OR converted to arithmetic form
$x \oplus y = x + y - 2xy$	XOR converted to arithmetic form

Example: Prove the identity $xyz \vee x' \vee y' \vee z' \overset{?}{\equiv} 1$

$$\begin{aligned}
\text{LHS} &= [xyz \vee x'] \vee [y' \vee z'] \\
&= [xyz + 1 - x - (1 - x)xyz] \vee [1 - y + 1 - z - (1 - y)(1 - z)] \\
&= [xyz + 1 - x] \vee [1 - yz] \\
&= (xyz + 1 - x) + (1 - yz) - (xyz + 1 - x)(1 - yz) \\
&= 1 + xy^2z^2 - xyz \\
&= 1 = \text{RHS}
\end{aligned}$$

This is addition, not logical OR

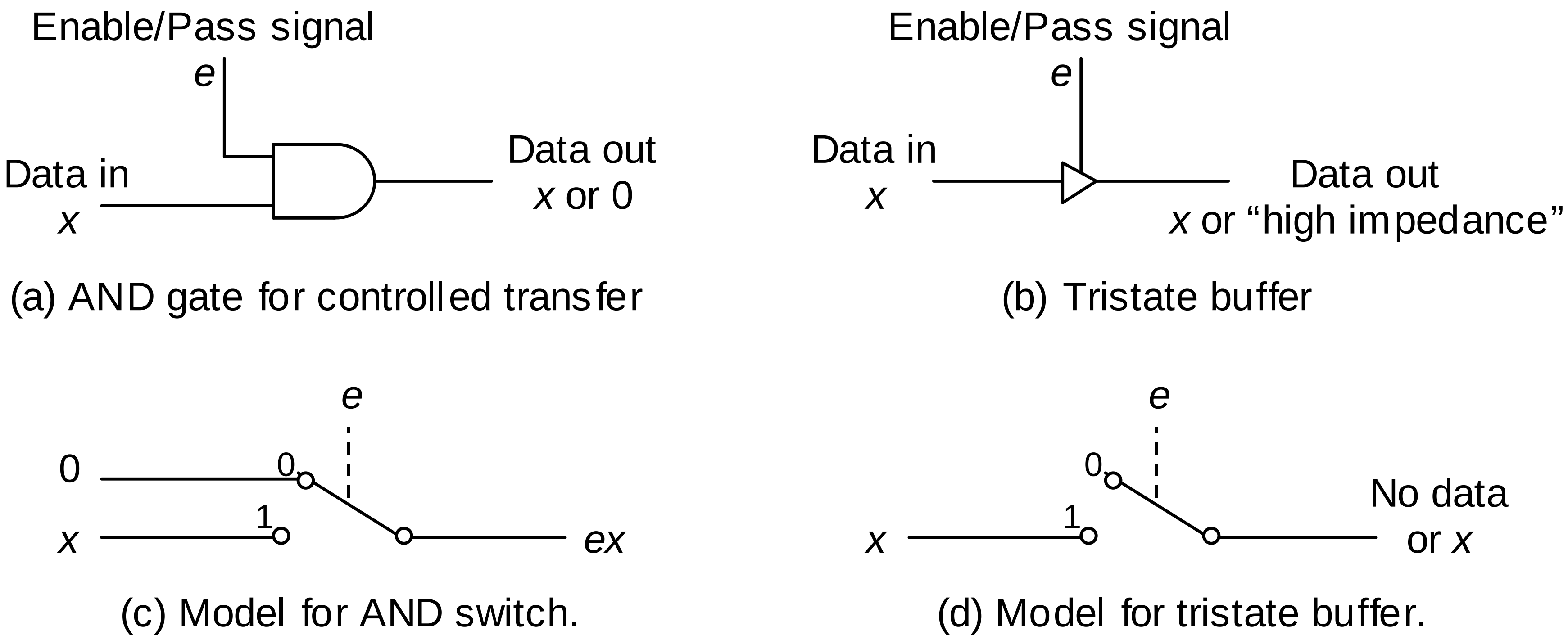
Variations in Gate Symbols



Bubble = Inverter

Figure 1.2 Gates with more than two inputs and/or with inverted signals at input or output.

Gates as Control Elements



(a) AND gate for controlled transfer

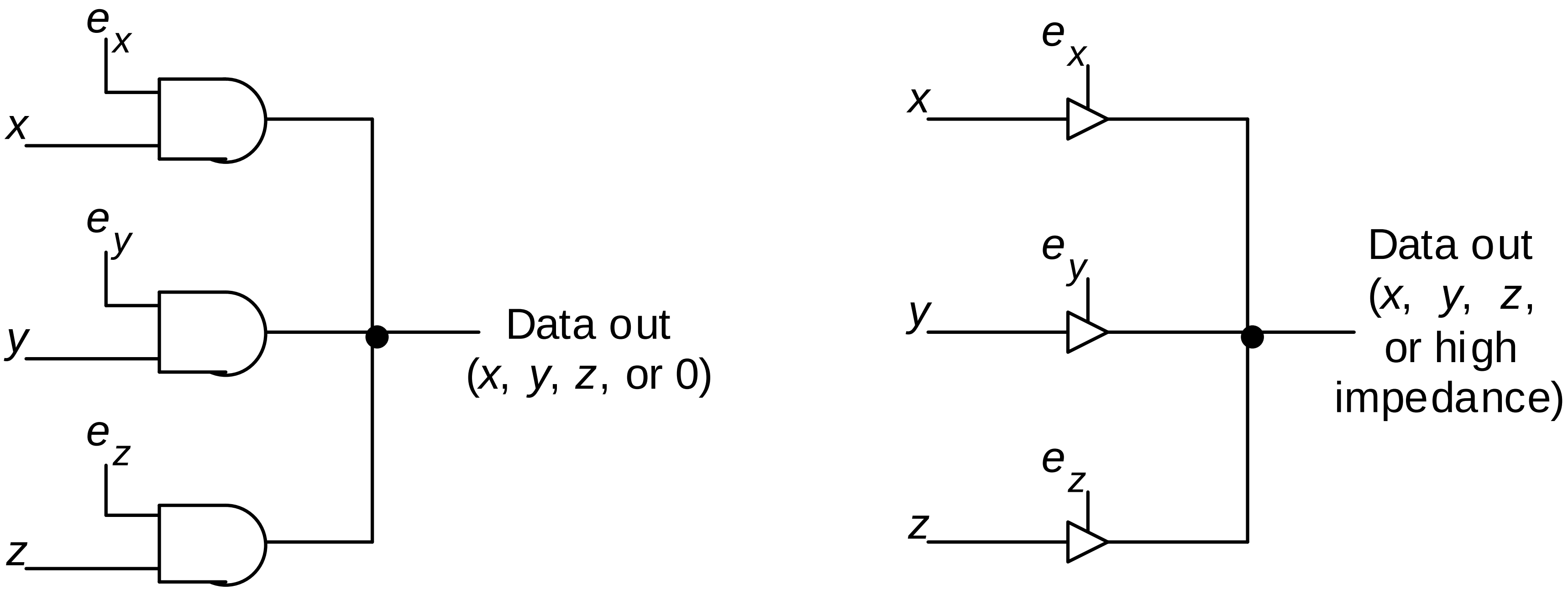
(b) Tristate buffer

(c) Model for AND switch.

(d) Model for tristate buffer.

Figure 1.3 An AND gate and a tristate buffer act as controlled switches or valves. An inverting buffer is logically the same as a NOT gate.

Wired OR and Bus Connections

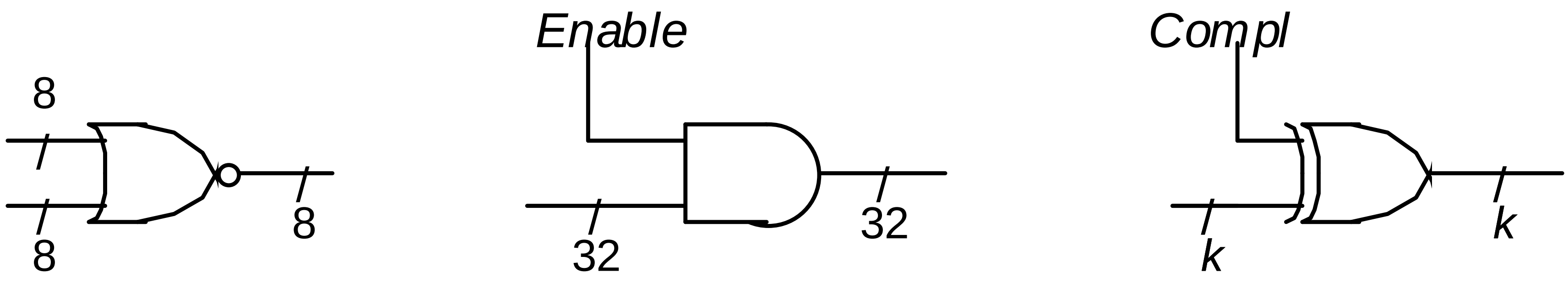


(a) Wired OR of product terms

(b) Wired OR of tristate outputs

Figure 1.4 Wired OR allows tying together of several controlled signals.

Control/Data Signals and Signal Bundles



(a) 8 NOR gates

(b) 32 AND gates

(c) *k* XOR gates

Figure 1.5 Arrays of logic gates represented by a single gate symbol.

1.2 Boolean Functions and Expressions

Ways of specifying a logic function

- Truth table: 2^n row, “don’t-care” in input or output
- Logic expression: $w' (x \vee y \vee z)$, product-of-sums, sum-of-products, equivalent expressions
- Word statement: Alarm will sound if the door is opened while the security system is engaged, or when the smoke detector is triggered
- Logic circuit diagram: Synthesis vs analysis

Manipulating Logic Expressions

Table 1.2 Laws (basic identities) of Boolean algebra.

Name of law	**OR version**	**AND version**
Identity	$x \vee 0 = x$	$x\,1 = x$
One/Zero	$x \vee 1 = 1$	$x\,0 = 0$
Idempotent	$x \vee x = x$	$x\,x = x$
Inverse	$x \vee x' = 1$	$x\,x' = 0$
Commutative	$x \vee y = y \vee x$	$x\,y = y\,x$
Associative	$(x \vee y) \vee z = x \vee (y \vee z)$	$(x\,y)\,z = x\,(y\,z)$
Distributive	$x \vee (y\,z) = (x \vee y)\,(x \vee z)$	$x\,(y \vee z) = (x\,y) \vee (x\,z)$
DeMorgan’s	$(x \vee y)' = x'\,y'$	$(x\,y)' = x' \vee y'$

Proving the Equivalence of Logic Expressions

Example 1.1

- Truth-table method: Exhaustive verification

- Arithmetic substitution

  $x \vee y = x + y - xy$

  $x \oplus y = x + y - 2xy$

  Example: $x \oplus y \overset{?}{\equiv} x'y \vee xy'$

  $x + y - 2xy \overset{?}{\equiv} (1-x)y + x(1-y) - (1-x)yx(1-y)$

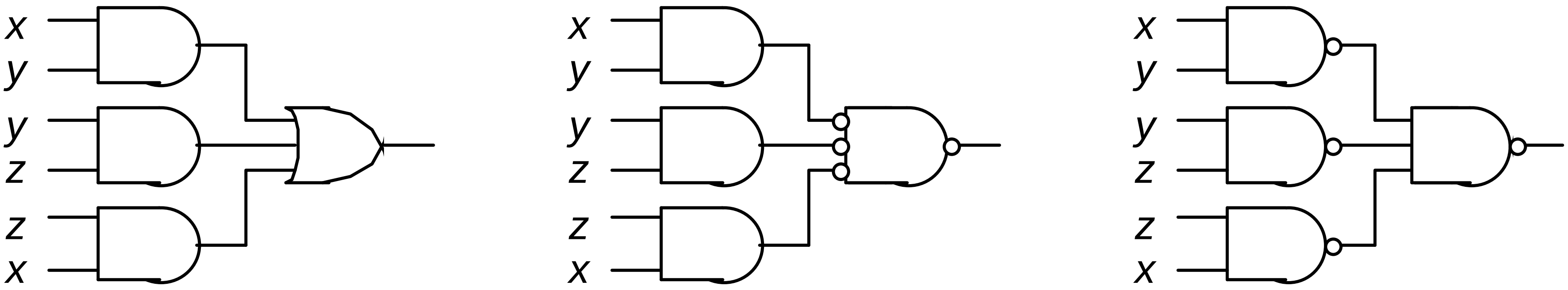
- Case analysis: two cases, $x = 0$ or $x = 1$

- Logic expression manipulation

1.3 Designing Gate Networks

- AND-OR, NAND-NAND, OR-AND, NOR-NOR
- Logic optimization: cost, speed, power dissipation

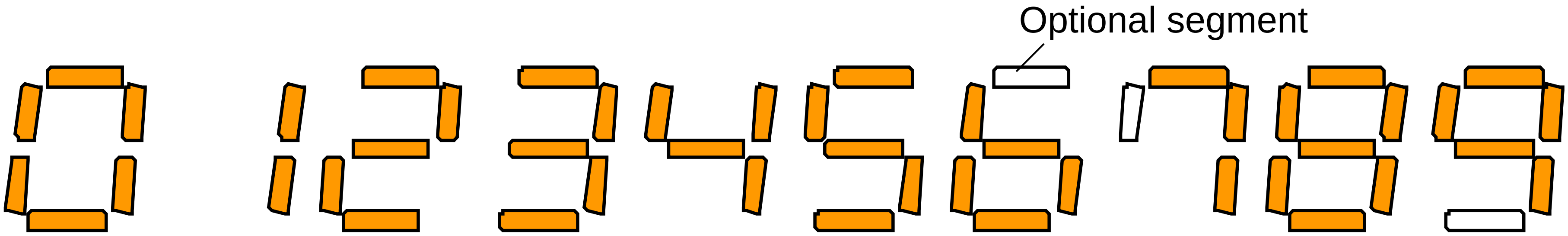
$$(a \vee b \vee c)' = a'b'c'$$

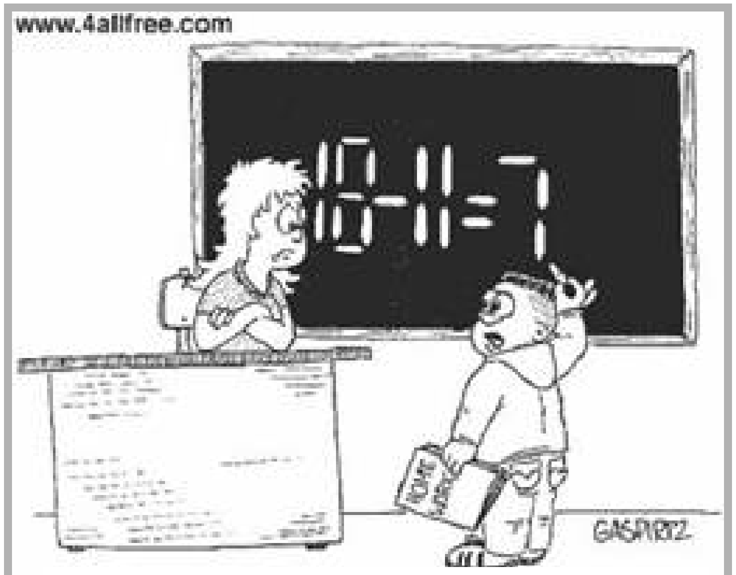


(a) AND-OR circuit (b) Intermediate circuit (c) NAND-NAND equivalent

Figure 1.6 A two-level AND-OR circuit and two equivalent circuits.

Seven-Segment Display of Decimal Digits





I swear I didn't use a calculator!

Figure 1.7 Seven-segment display of decimal digits. The three open segments may be optionally used. The digit 1 can be displayed in two ways, with the more common right-side version shown.

BCD-to-Seven-Segment Decoder

Example 1.2

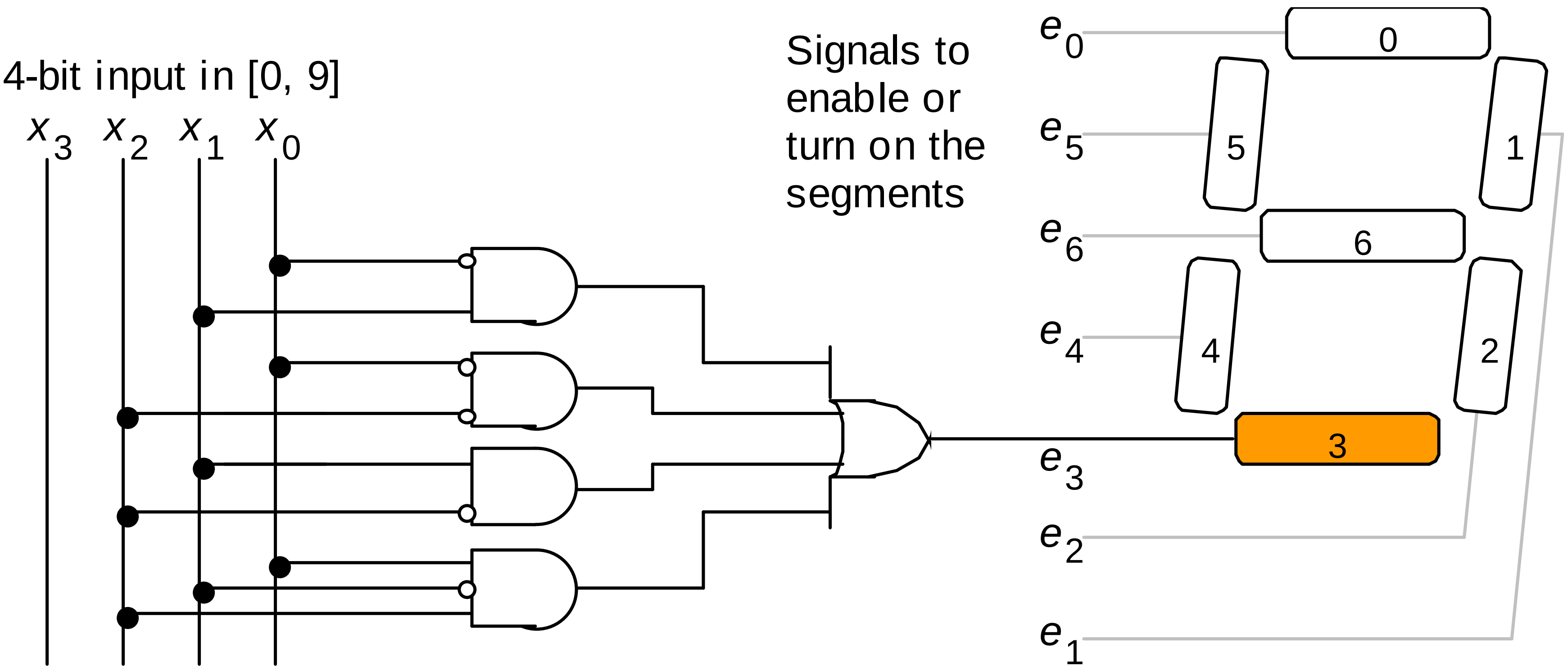
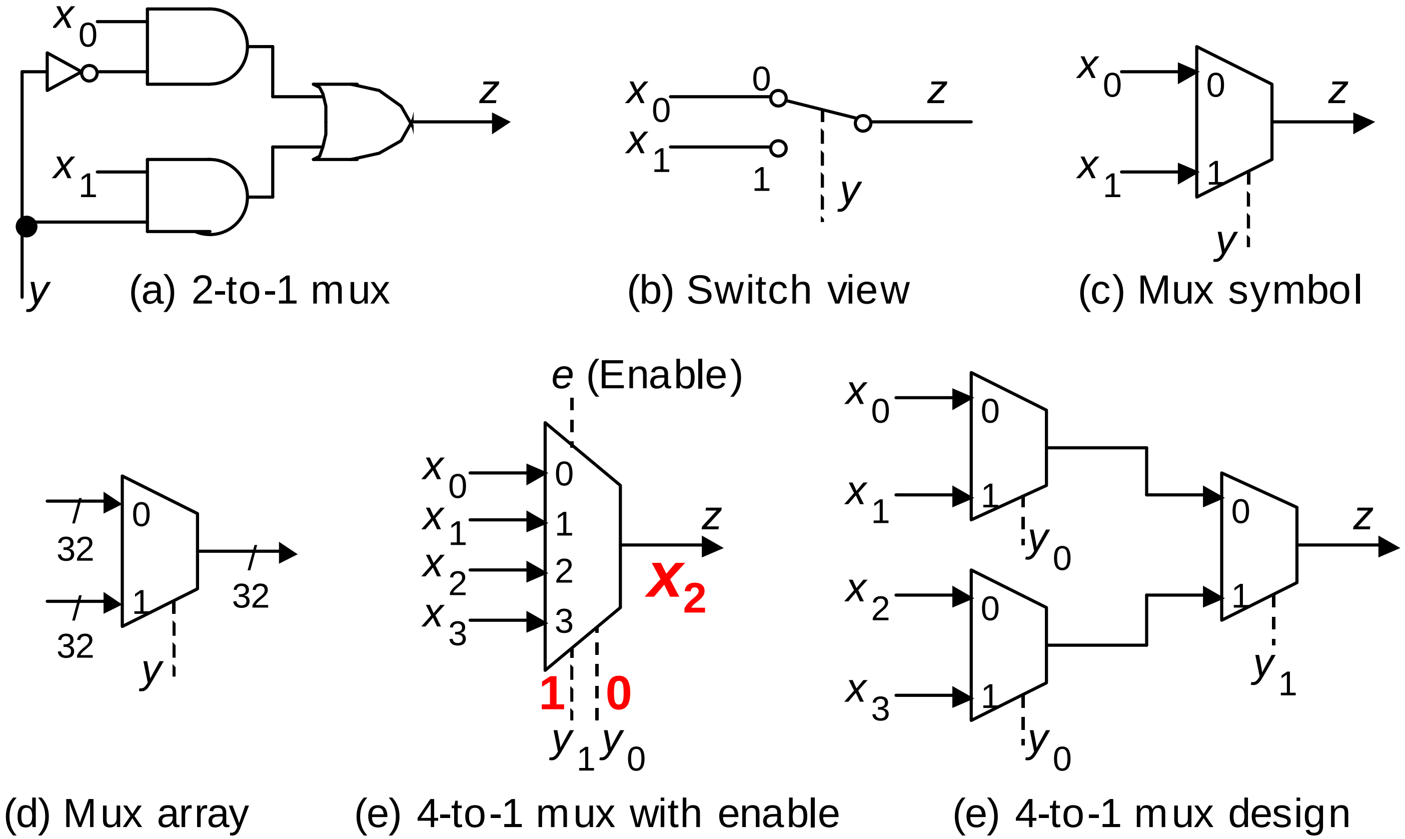


Figure 1.8 The logic circuit that generates the enable signal for the lowermost segment (number 3) in a seven-segment display unit.

1.4 Useful Combinational Parts

- High-level building blocks
- Much like prefab parts used in building a house
- Arithmetic components (adders, multipliers, ALUs) will be covered in Part III
- Here we cover three useful parts: multiplexers, decoders/demultiplexers, encoders

Multiplexers

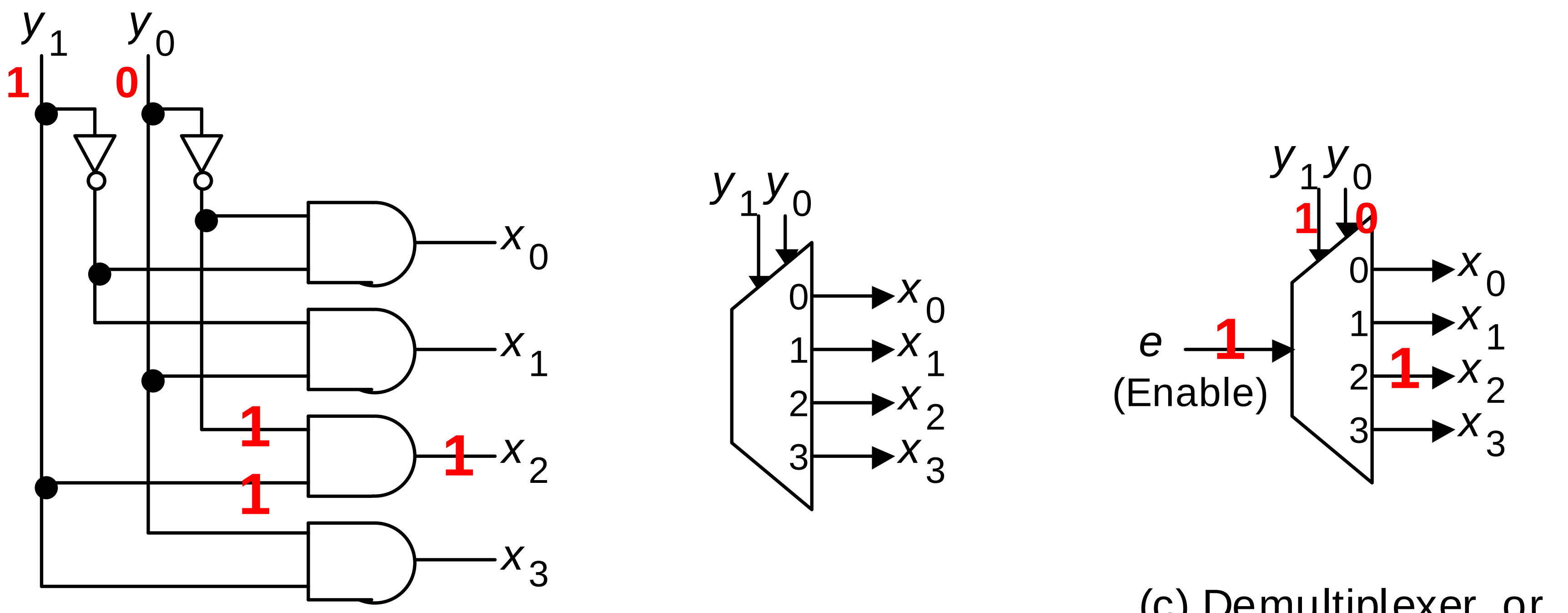


(a) 2-to-1 mux (b) Switch view (c) Mux symbol

(d) Mux array (e) 4-to-1 mux with enable (e) 4-to-1 mux design

Figure 1.9 Multiplexer (mux), or selector, allows one of several inputs to be selected and routed to output depending on the binary value of a set of selection or address signals provided to it.

Decoders/Demultiplexers



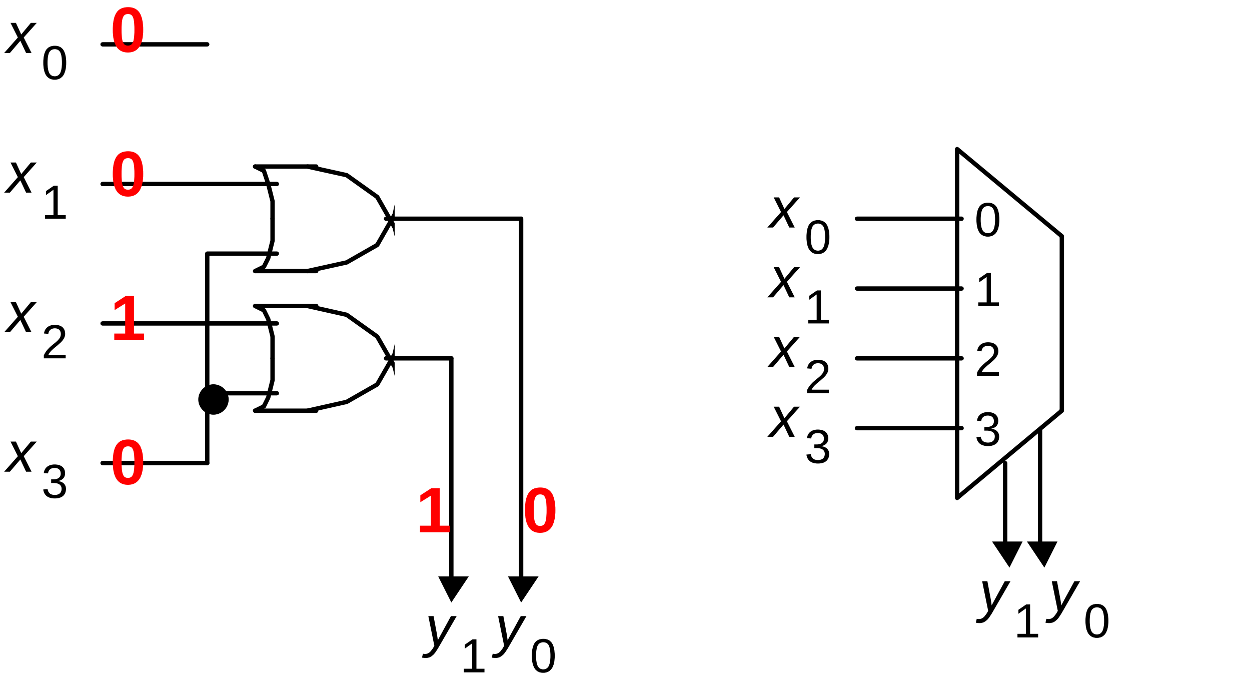
(a) 2-to-4 decoder

(b) Decoder symbol

(c) Demultiplexer, or decoder with “enable”

Figure 1.10 A decoder allows the selection of one of 2^a options using an *a*-bit address as input. A demultiplexer (demux) is a decoder that only selects an output if its enable signal is asserted.

Encoders



(a) 4-to-2 encoder

(b) Encoder symbol

Figure 1.11 A 2^a-to-a encoder outputs an a-bit binary number equal to the index of the single 1 among its 2^a inputs.

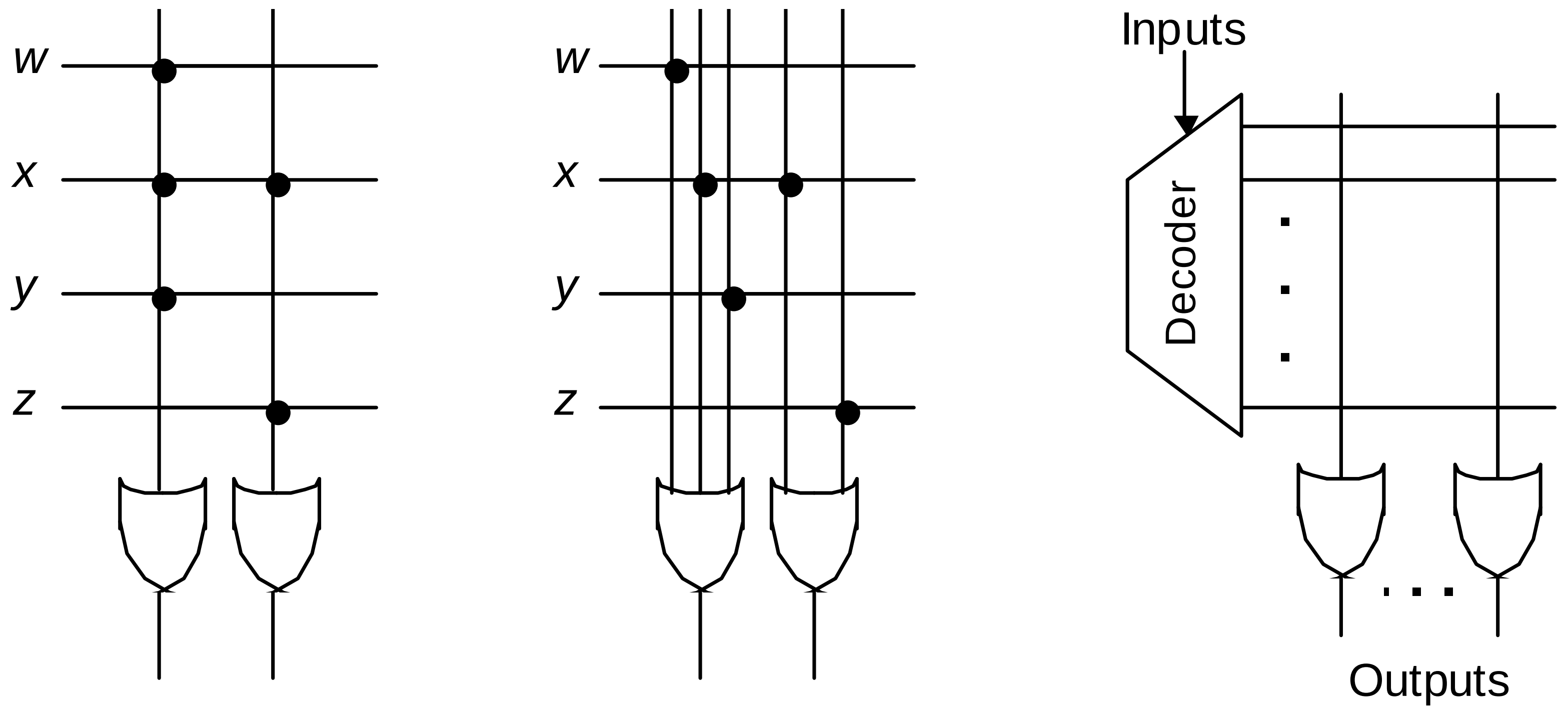
1.5 Programmable Combinational Parts

A programmable combinational part can do the job of many gates or gate networks

Programmed by cutting existing connections (*fuses*) or establishing new connections (*antifuses*)

- Programmable ROM (PROM)
- Programmable array logic (PAL)
- Programmable logic array (PLA)

PROMs



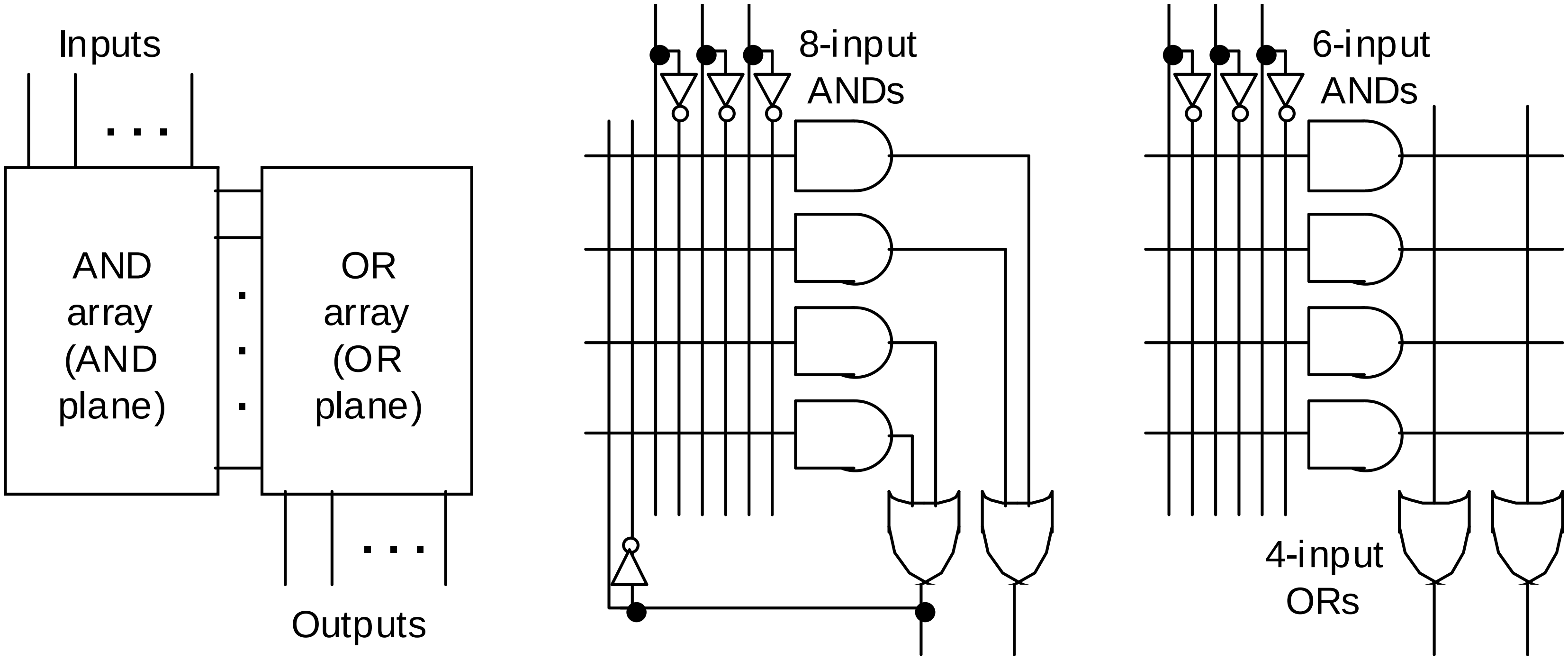
(a) Programmable OR gates

(b) Logic equivalent of part a

(c) Programmable read-only memory (PROM)

Figure 1.12 Programmable connections and their use in a PROM.

PALs and PLAs



(a) General programmable combinational logic

(b) PAL: programmable AND array, fixed OR array

(c) PLA: programmable AND and OR arrays

Figure 1.13 Programmable combinational logic: general structure and two classes known as PAL and PLA devices. Not shown is PROM with fixed AND array (a decoder) and programmable OR array.

1.6 Timing and Circuit Considerations

Changes in gate/circuit output, triggered by changes in its inputs, are not instantaneous

- Gate delay δ: a fraction of, to a few, nanoseconds
- Wire delay, previously negligible, is now important (electronic signals travel about 15 cm per ns)
- Circuit simulation to verify function and timing

Glitching

Using the PAL in Fig. 1.13b to implement $f = x \vee y \vee z$

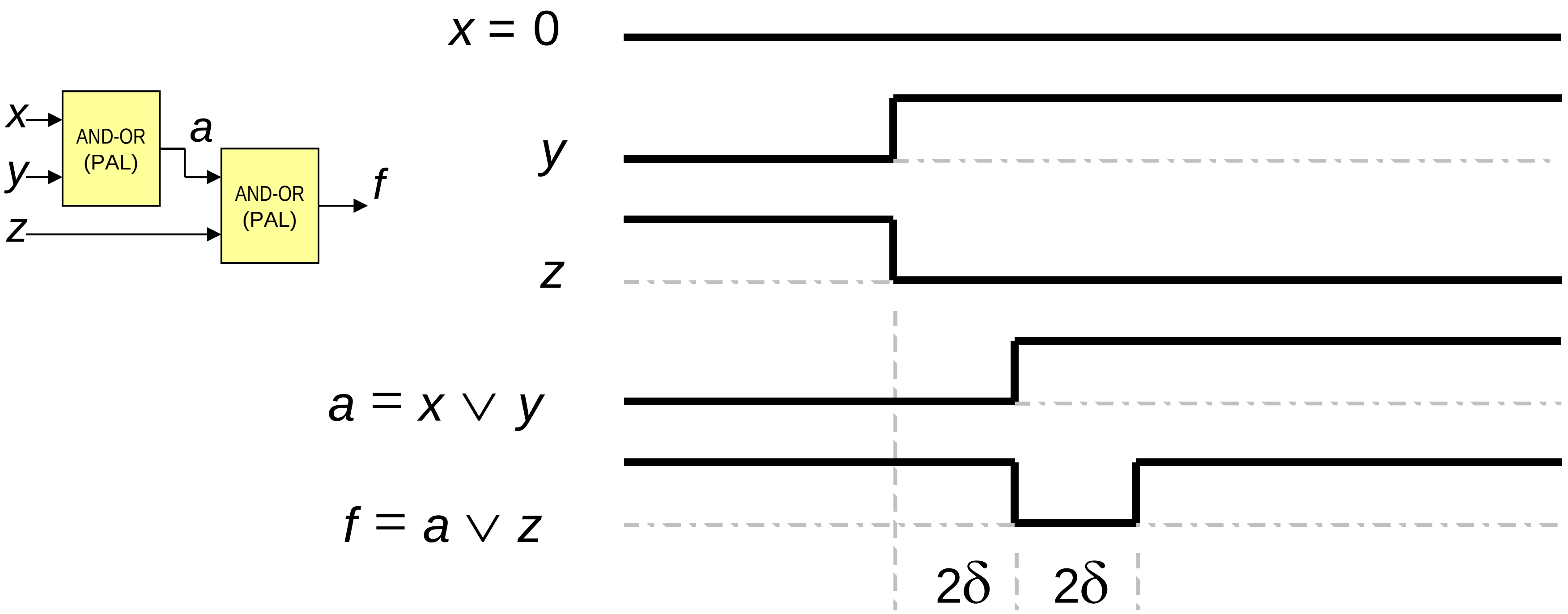
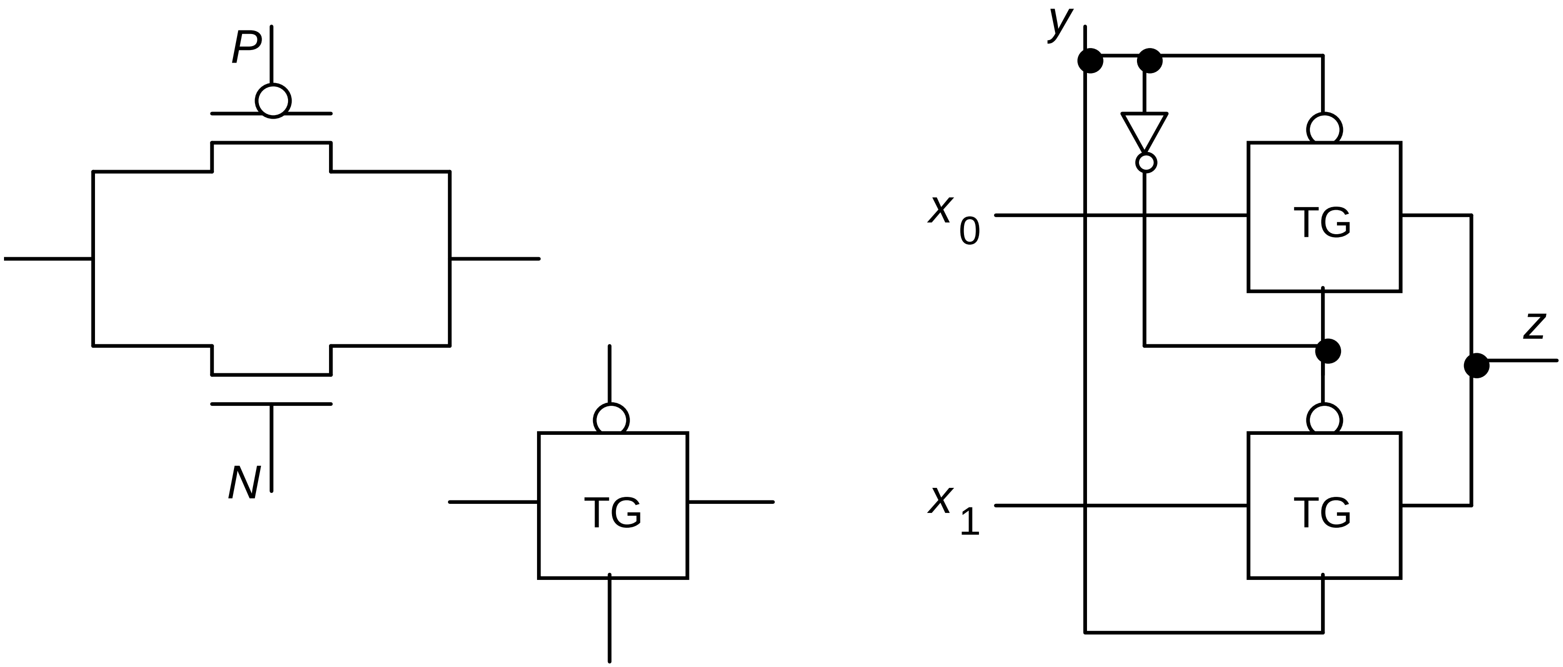


Figure 1.14 Timing diagram for a circuit that exhibits glitching.

CMOS Transmission Gates



(a) CMOS transmission gate: circuit and symbol

(b) Two-input mux built of two transmission gates

Figure 1.15 A CMOS transmission gate and its use in building a 2-to-1 mux.

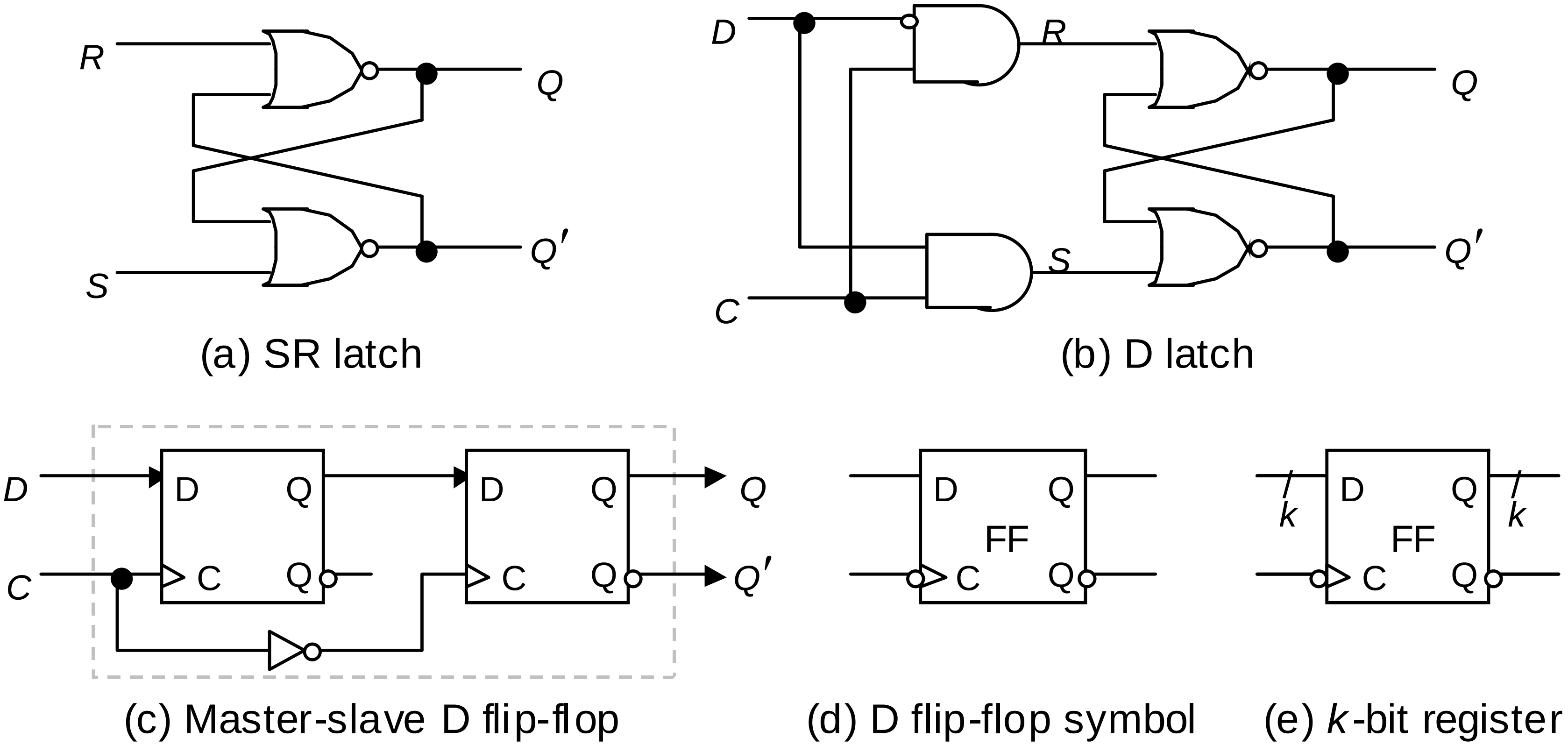
2 Digital Circuits with Memory

Second of two chapters containing a review of digital design:

- Combinational (memoryless) circuits in Chapter 1
- Sequential circuits (with memory) in Chapter 2

Topics in This Chapter	
2.1	Latches, Flip-Flops, and Registers
2.2	Finite-State Machines
2.3	Designing Sequential Circuits
2.4	Useful Sequential Parts
2.5	Programmable Sequential Parts
2.6	Clocks and Timing of Events

2.1 Latches, Flip-Flops, and Registers



(a) SR latch

(b) D latch

(c) Master-slave D flip-flop

(d) D flip-flop symbol

(e) *k*-bit register

Figure 2.1 Latches, flip-flops, and registers.

Latches vs Flip-Flops

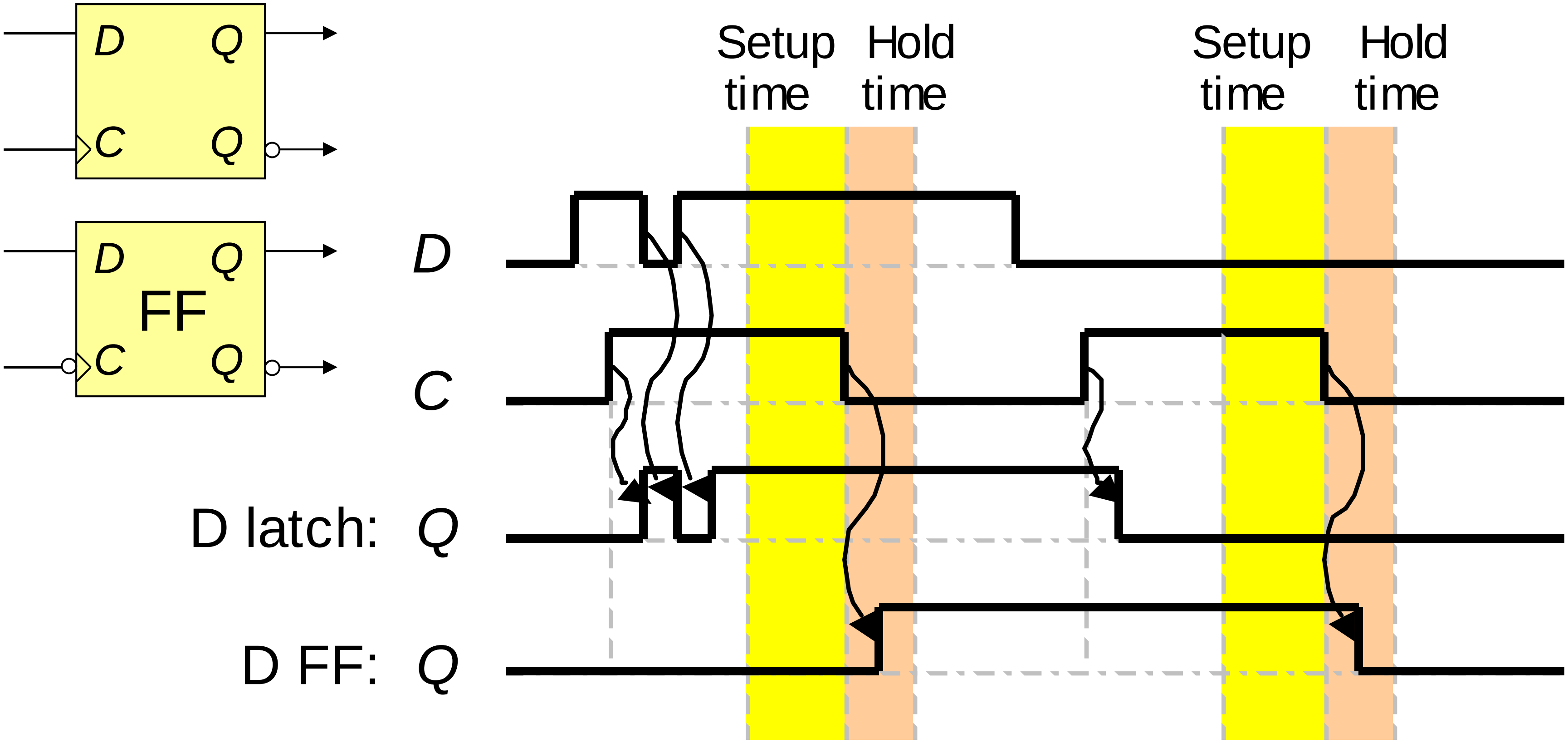


Figure 2.2 Operations of D latch and negative-edge-triggered D flip-flop.

Reading and Modifying FFs in the Same Cycle

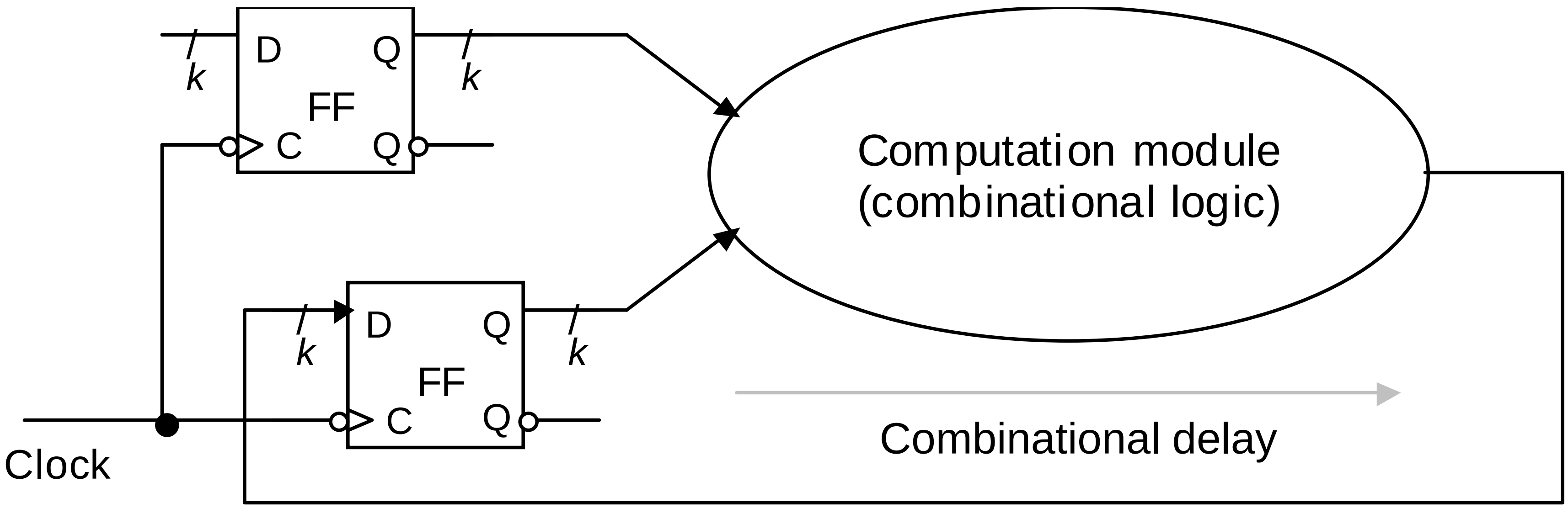


Figure 2.3 Register-to-register operation with edge-triggered flip-flops.

2.2 Finite-State Machines

Example 2.1

Current state ↓	------- Input ------- Dime	 Quarter	 Reset
S_{00}	S_{10}	S_{25}	S_{00}
S_{10}	S_{20}	S_{35}	S_{00}
S_{20}	S_{30}	S_{35}	S_{00}
S_{25}	S_{35}	S_{35}	S_{00}
S_{30}	S_{35}	S_{35}	S_{00}
S_{35}	S_{35}	S_{35}	S_{00}

Next state

S_{00} is the initial state
S_{35} is the final state

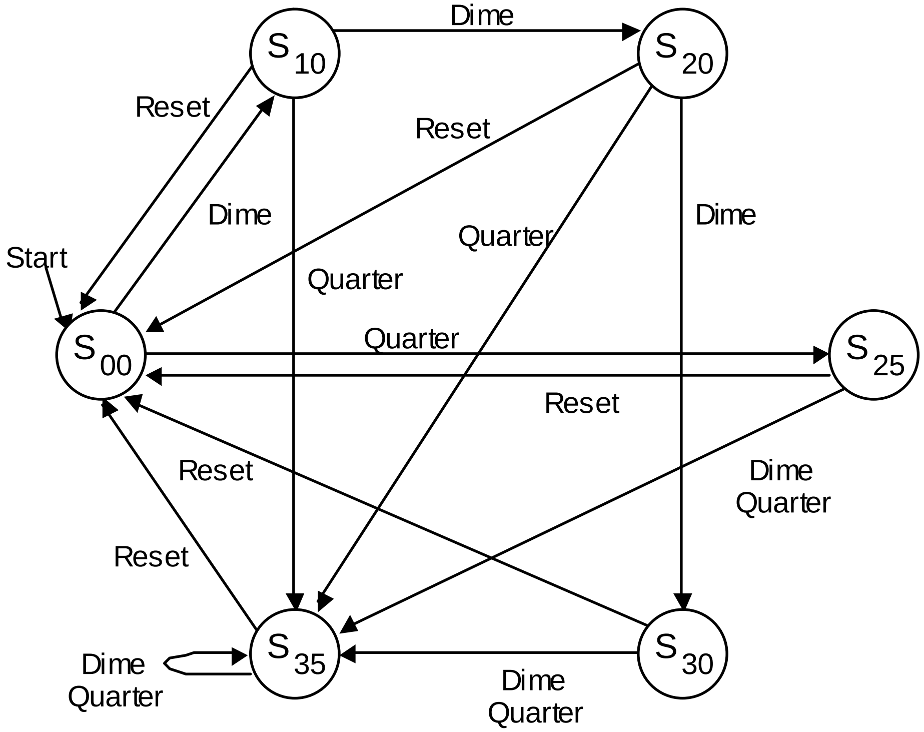


Figure 2.4 State table and state diagram for a vending machine coin reception unit.

Sequential Machine Implementation

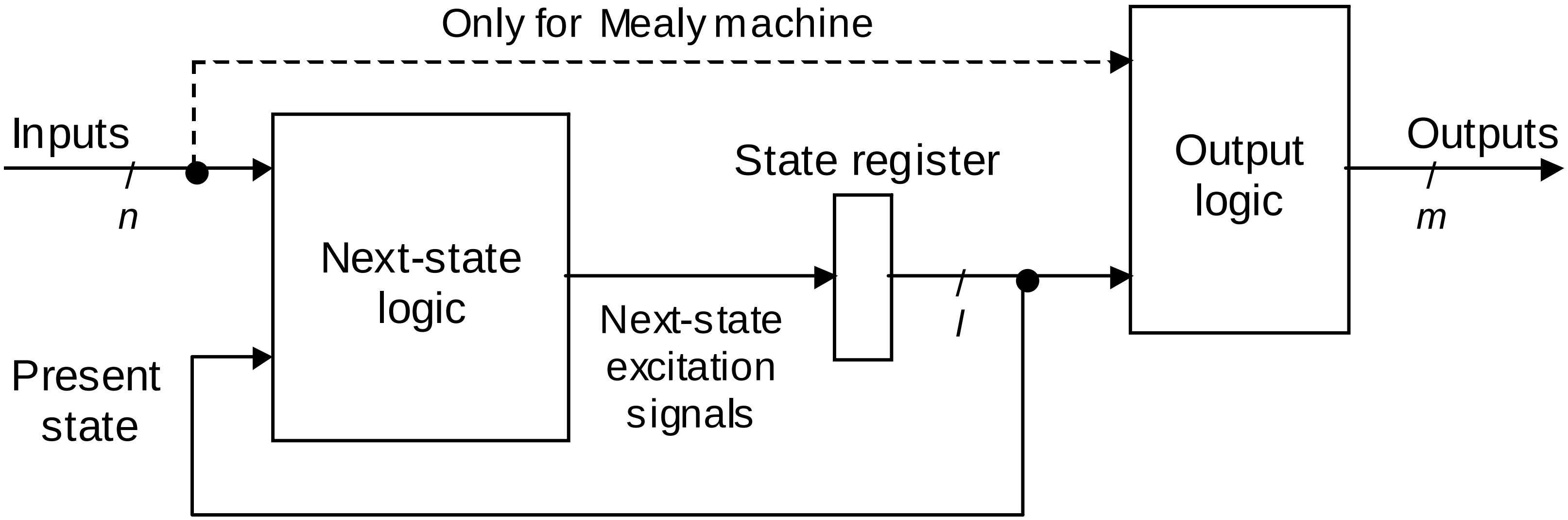


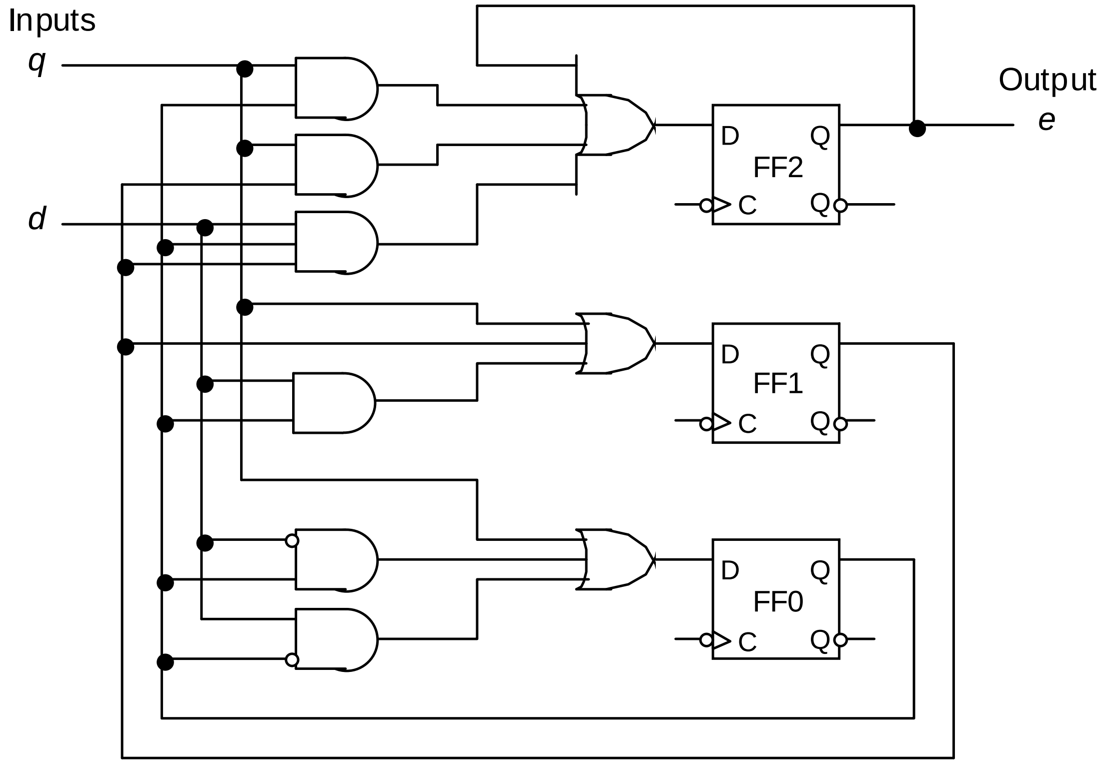
Figure 2.5   Hardware realization of Moore and Mealy sequential machines.

2.3 Designing Sequential Circuits

Example 2.3

Quarter in

Dime in



Final state is 1xx

Figure 2.7 Hardware realization of a coin reception unit (Example 2.3).

2.4 Useful Sequential Parts

- High-level building blocks
- Much like prefab closets used in building a house
- Other memory components will be covered in Chapter 17 (SRAM details, DRAM, Flash)
- Here we cover three useful parts:
  shift register, register file (SRAM basics), counter

Shift Register

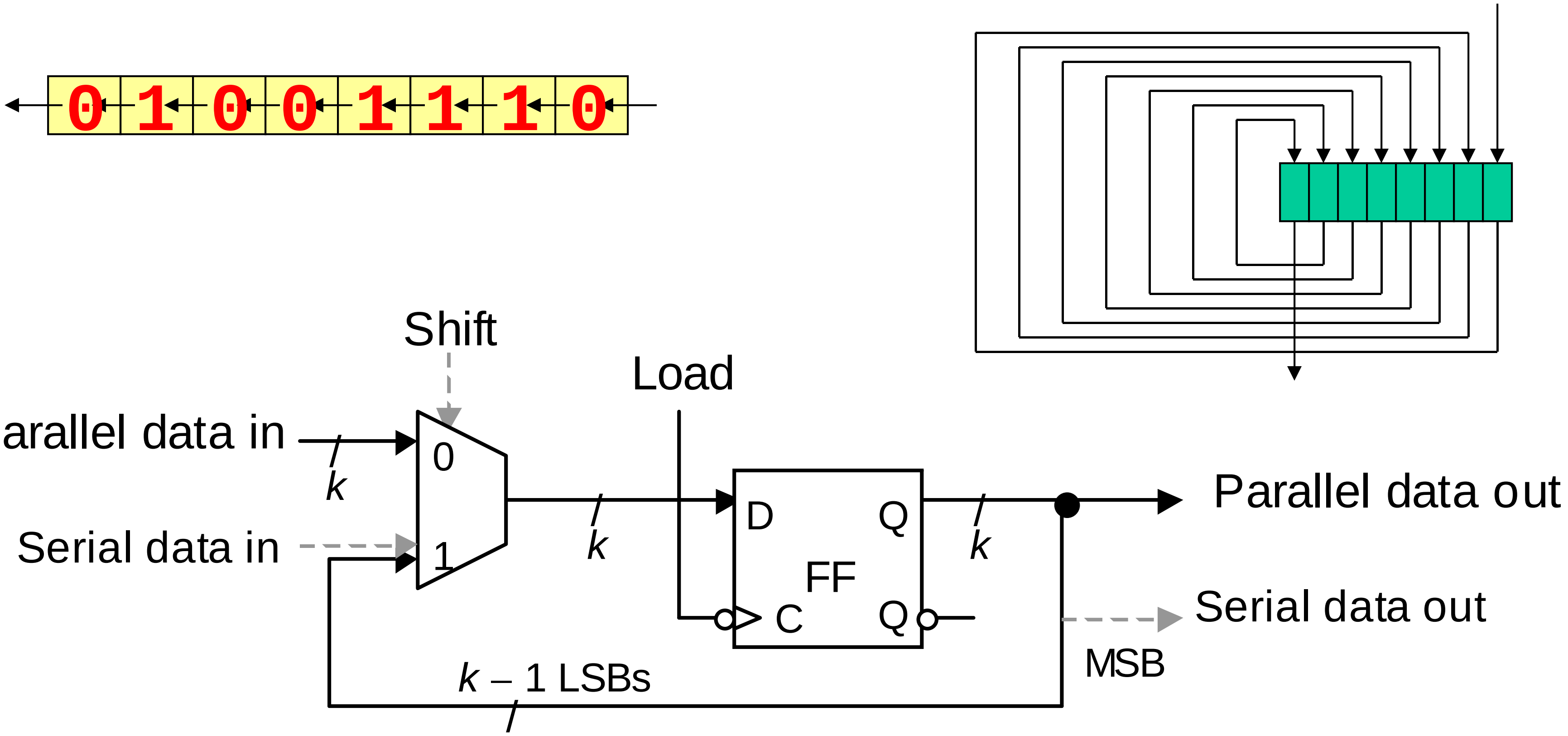
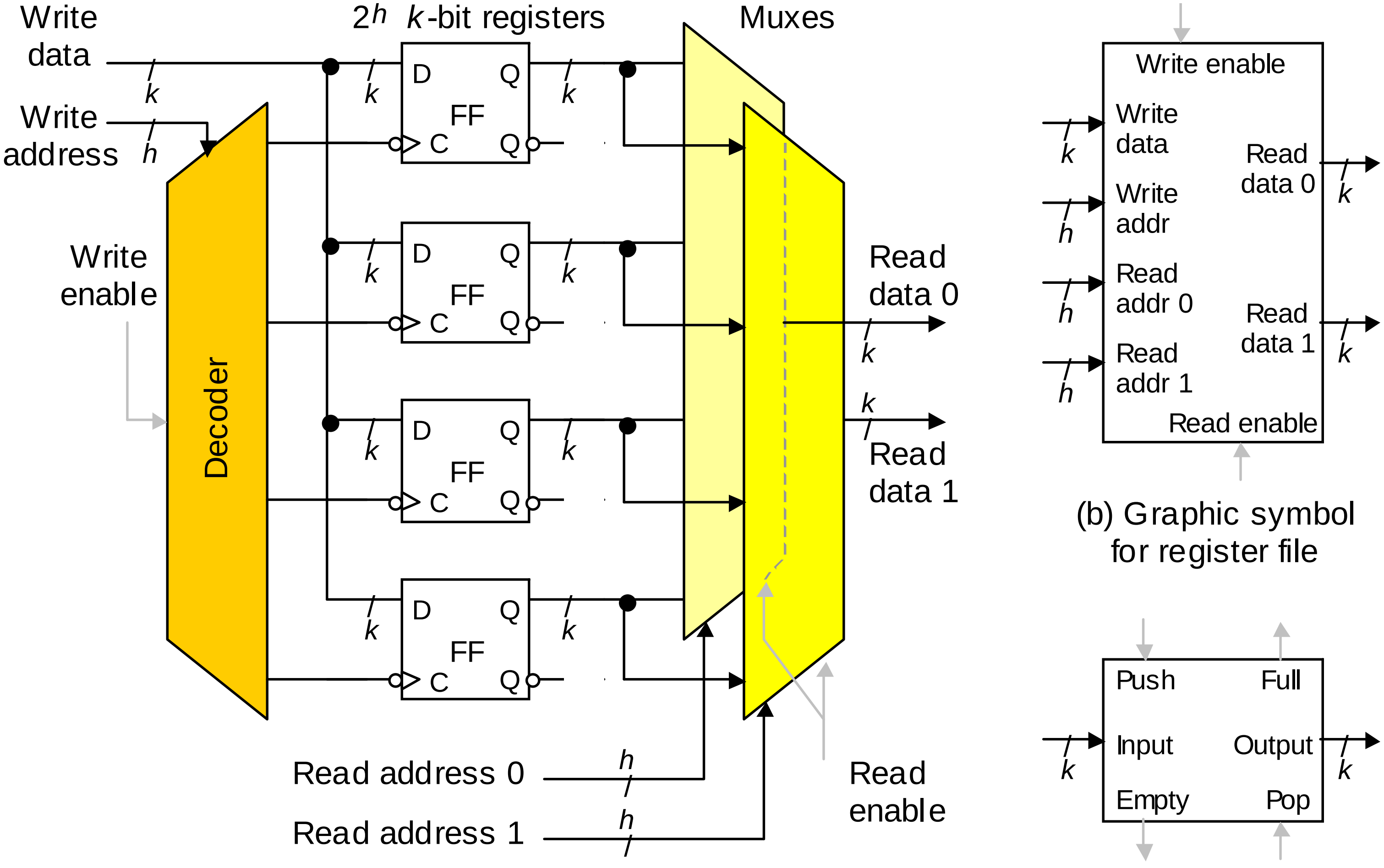


Figure 2.8 Register with single-bit left shift and parallel load capabilities. For logical left shift, serial data in line is connected to 0.

Register File and FIFO



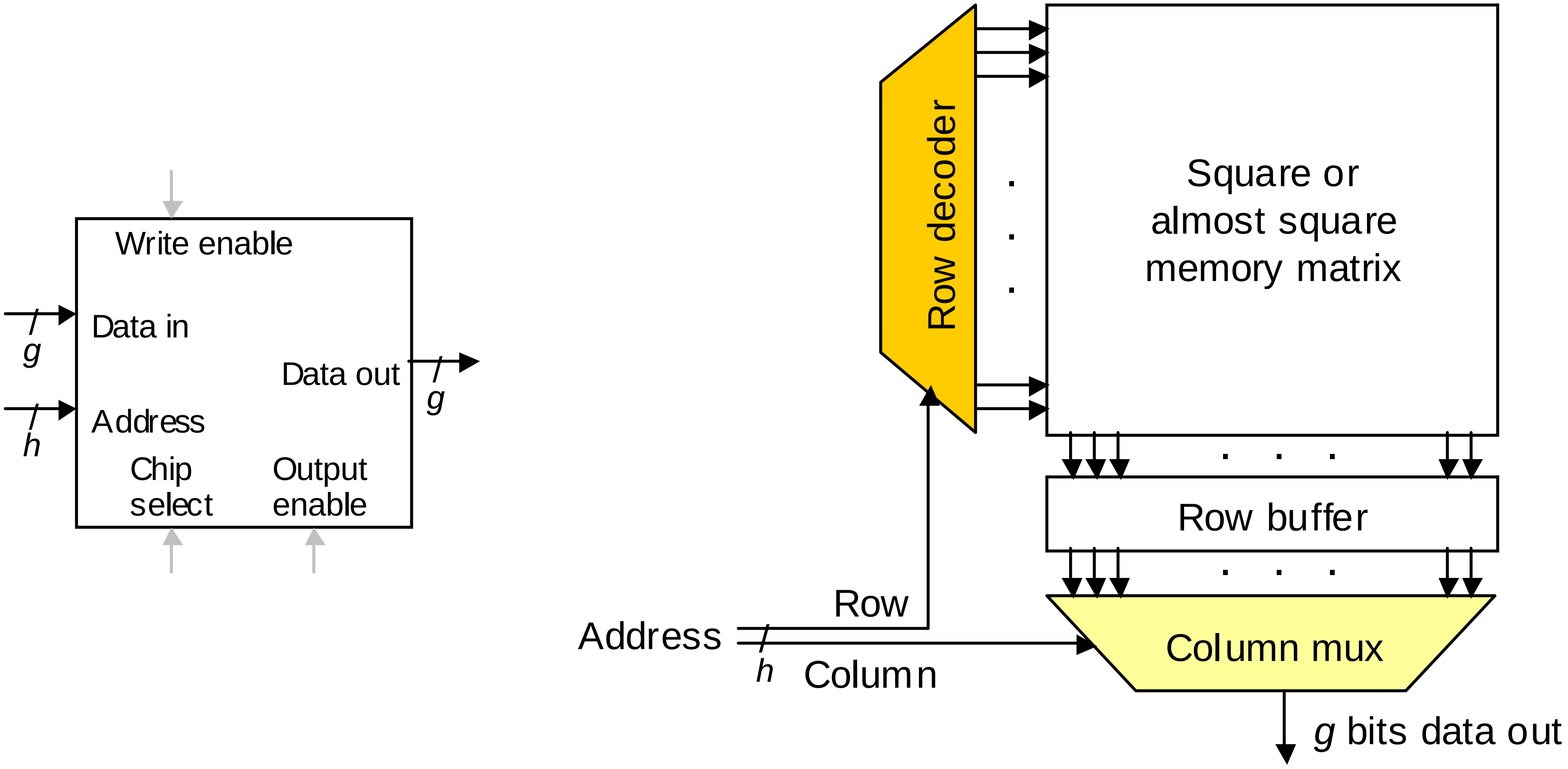
(a) Register file with random access

(b) Graphic symbol for register file

(c) FIFO symbol

Figure 2.9 Register file with random access and FIFO.

SRAM



(a) SRAM block diagram

(b) SRAM read mechanism

Figure 2.10 SRAM memory is simply a large, single-port register file.

Binary Counter

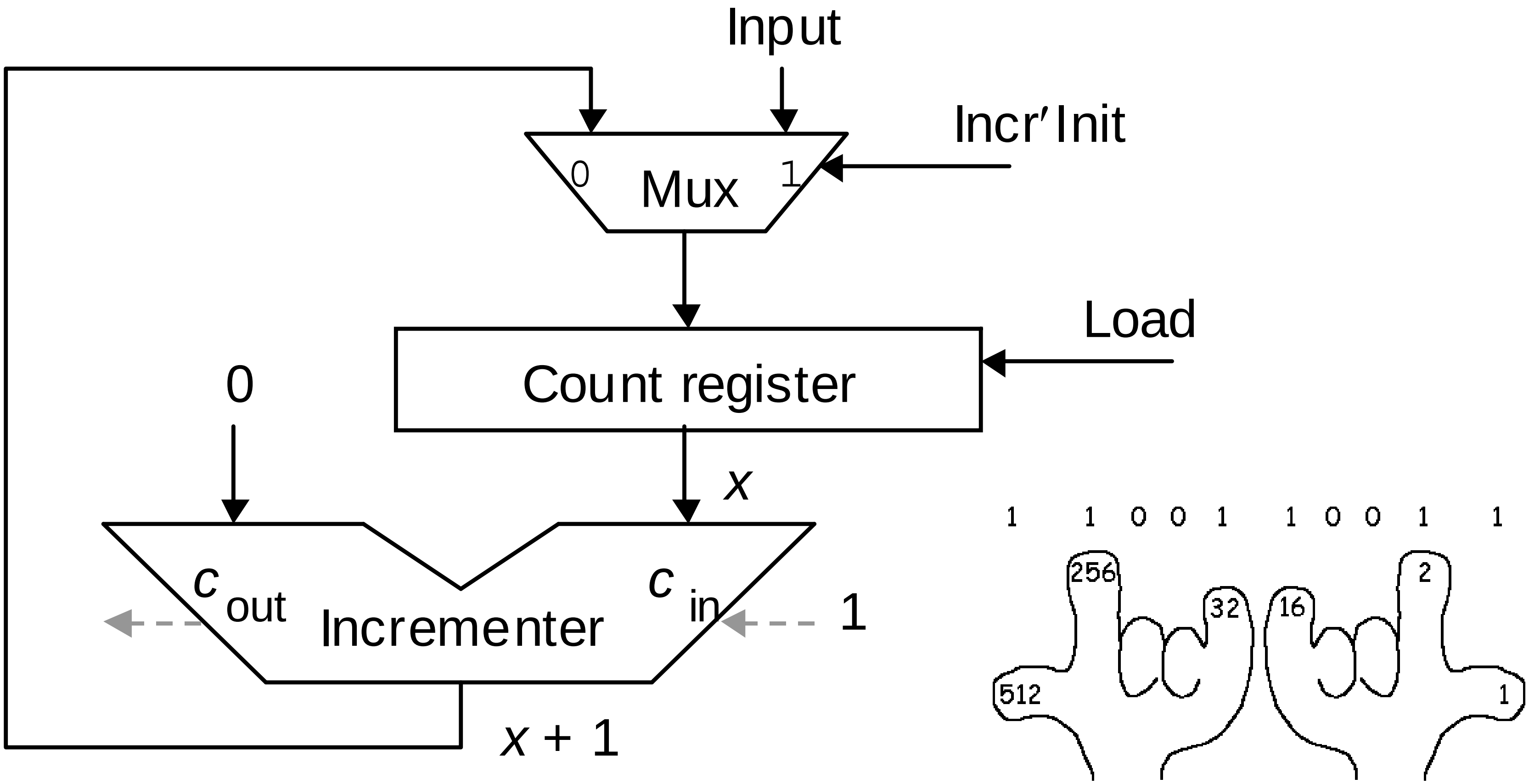


Figure 2.11 Synchronous binary counter with initialization capability.

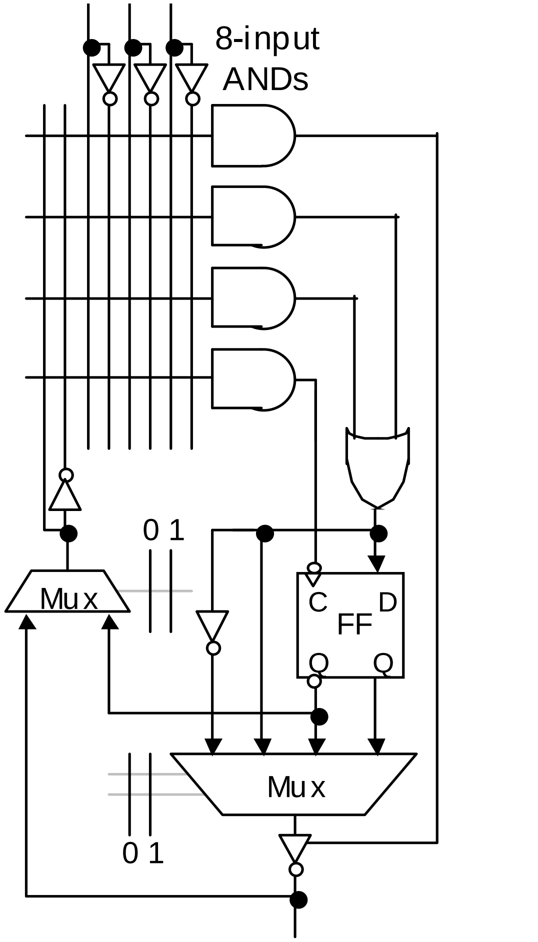
2.5 Programmable Sequential Parts

A programmable sequential part contain gates and memory elements

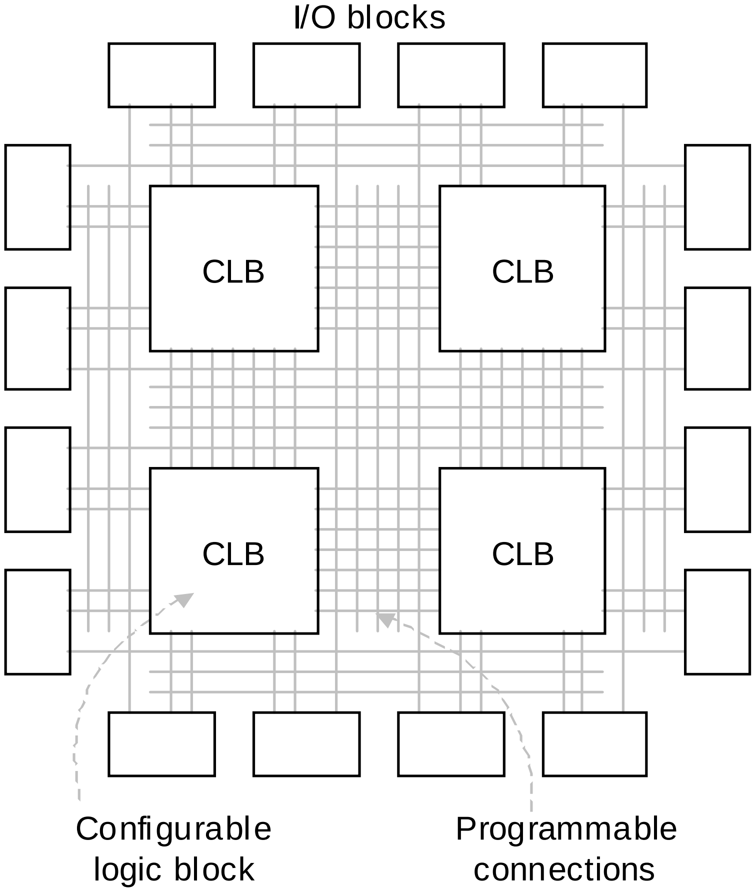
Programmed by cutting existing connections (*fuses*) or establishing new connections (*antifuses*)

- Programmable array logic (PAL)
- Field-programmable gate array (FPGA)
- Both types contain macrocells and interconnects

PAL and FPGA



(a) Portion of PAL with storable output



(b) Generic structure of an FPGA

Figure 2.12 Examples of programmable sequential logic.

2.6 Clocks and Timing of Events

Clock is a periodic signal: clock rate = clock frequency

The inverse of clock rate is the clock period: 1 GHz ↔ 1 ns

Constraint: Clock period $\geq t_{\text{prop}} + t_{\text{comb}} + t_{\text{setup}} + t_{\text{skew}}$

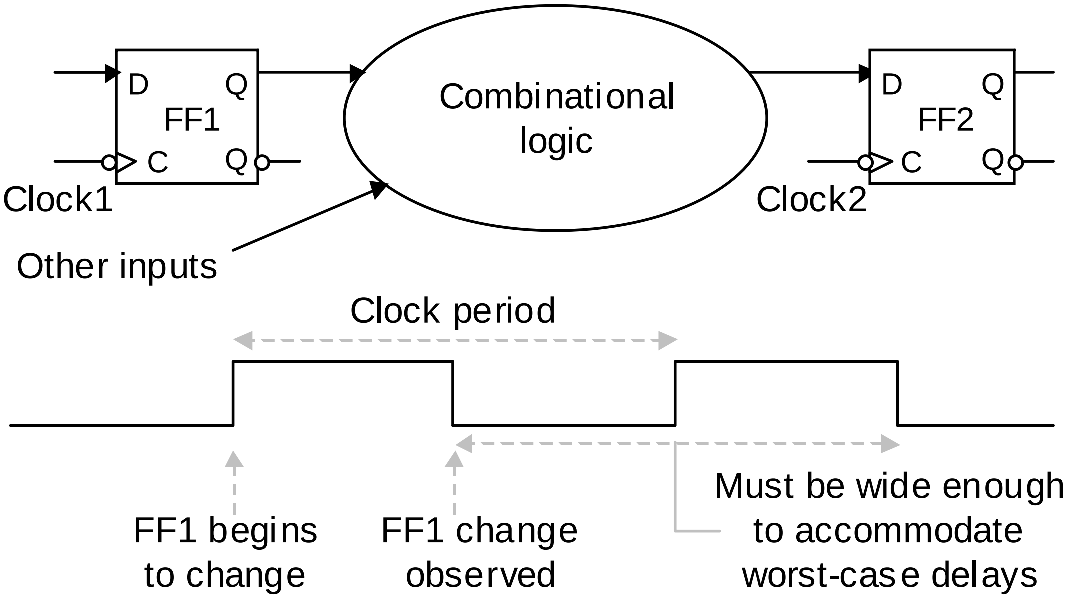
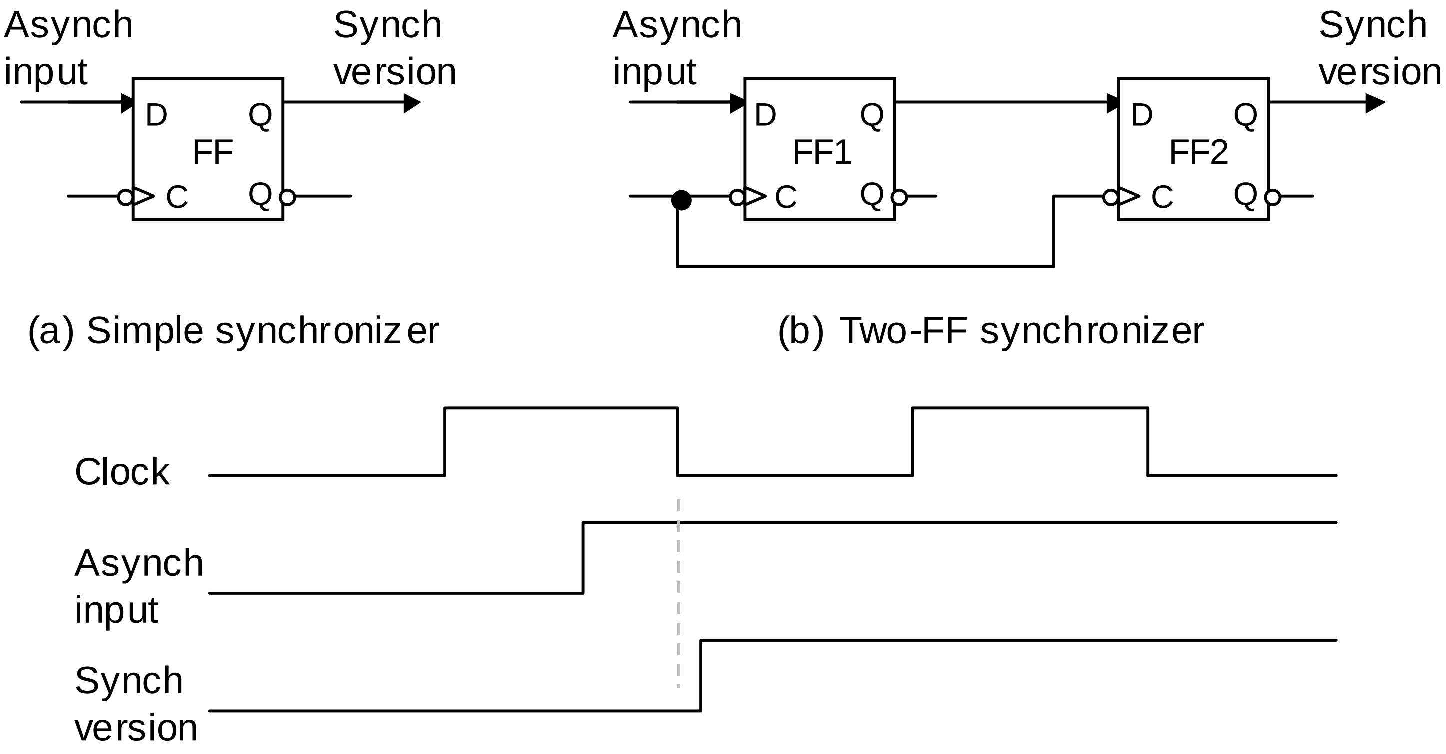


Figure 2.13 Determining the required length of the clock period.

Synchronization



(a) Simple synchronizer

(b) Two-FF synchronizer

(c) Input and output waveforms

Figure 2.14 Synchronizers are used to prevent timing problems arising from untimely changes in asynchronous signals.

Level-Sensitive Operation

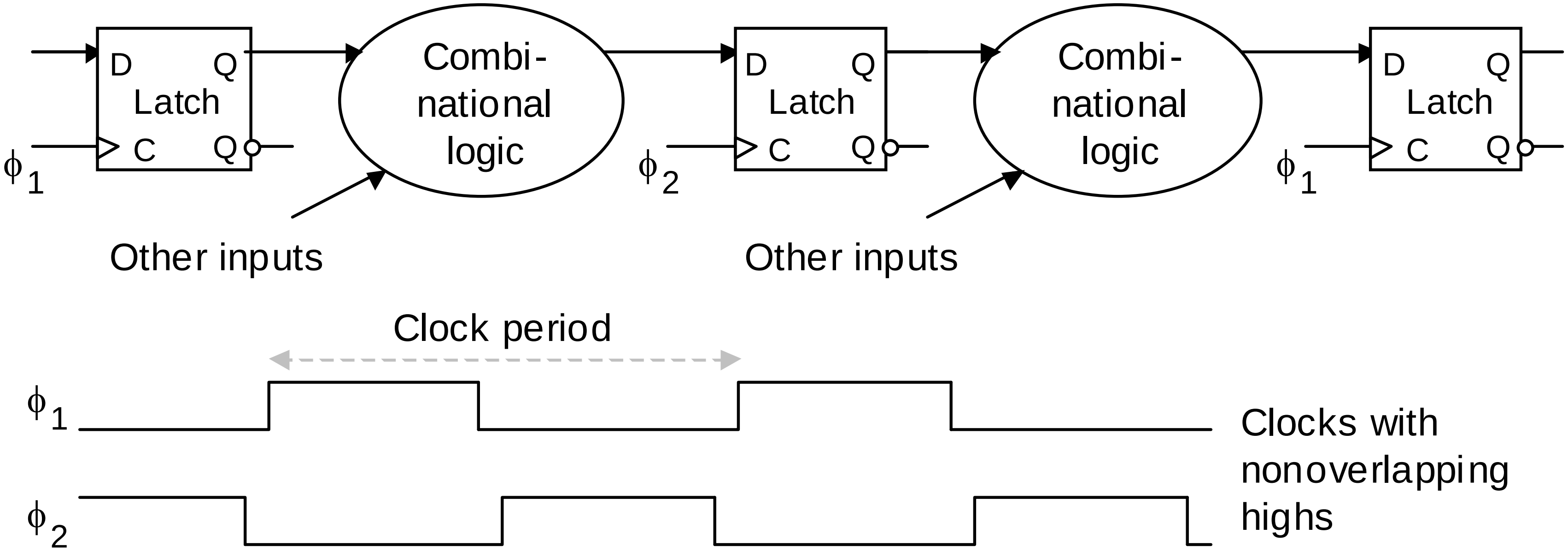


Figure 2.15 Two-phase clocking with nonoverlapping clock signals.

3 Computer System Technology

Interplay between architecture, hardware, and software

- Architectural innovations influence technology
- Technological advances drive changes in architecture

Topics in This Chapter
3.1 From Components to Applications
3.2 Computer Systems and Their Parts
3.3 Generations of Progress
3.4 Processor and Memory Technologies
3.5 Peripherals, I/O, and Communications
3.6 Software Systems and Applications

3.1 From Components to Applications

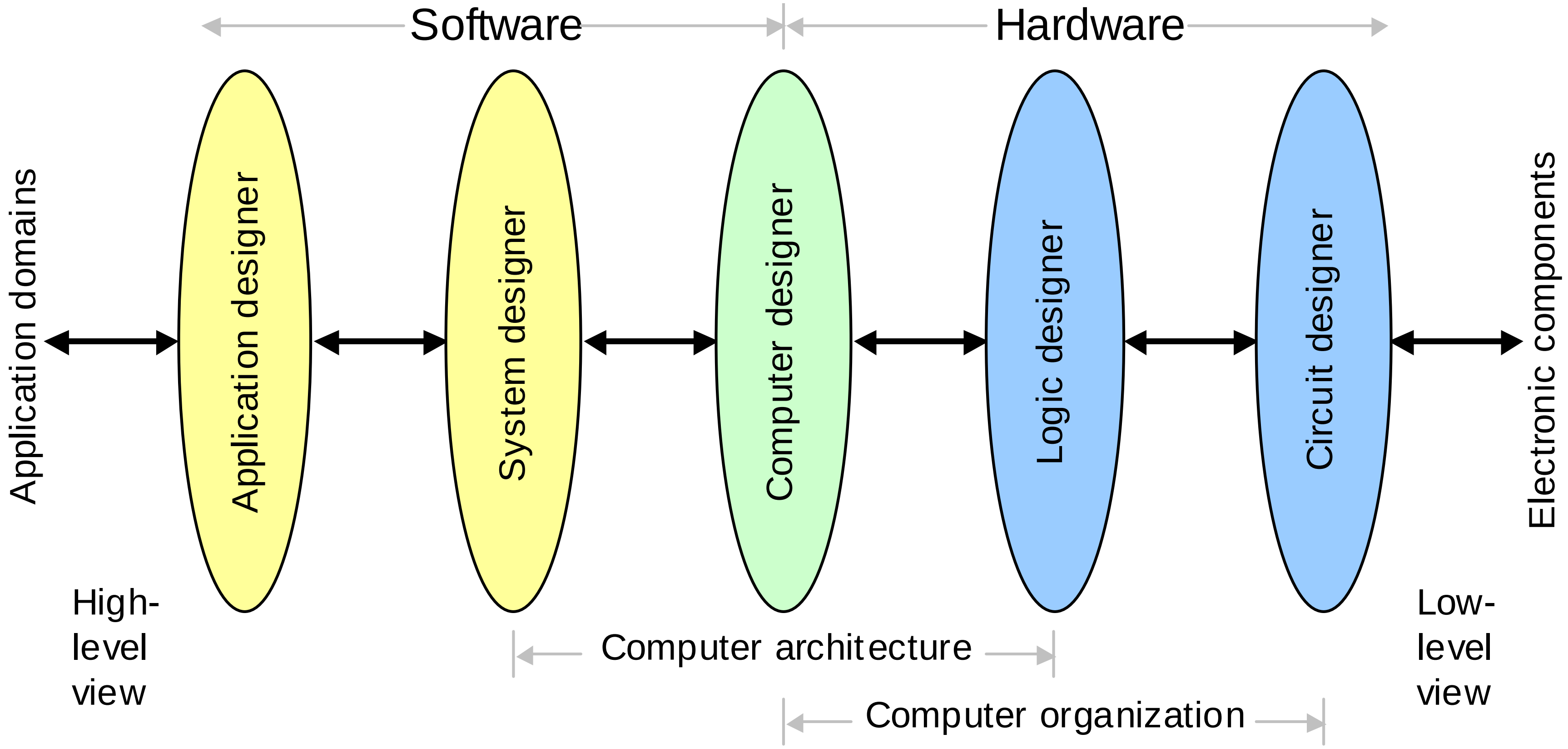


Figure 3.1 Subfields or views in computer system engineering.

What Is (Computer) Architecture?

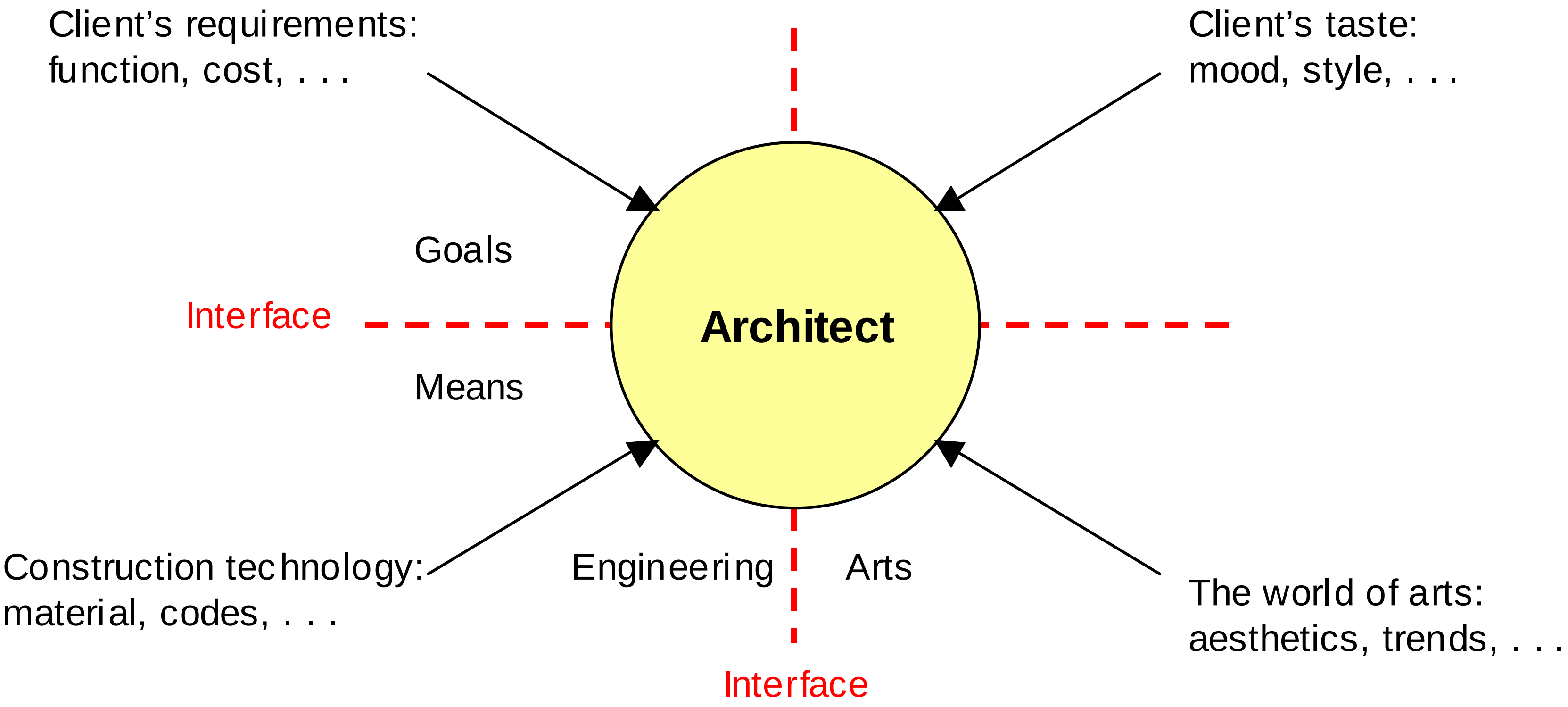


Figure 3.2 Like a building architect, whose place at the engineering/arts and goals/means interfaces is seen in this diagram, a computer architect reconciles many conflicting or competing demands.

3.2 Computer Systems and Their Parts

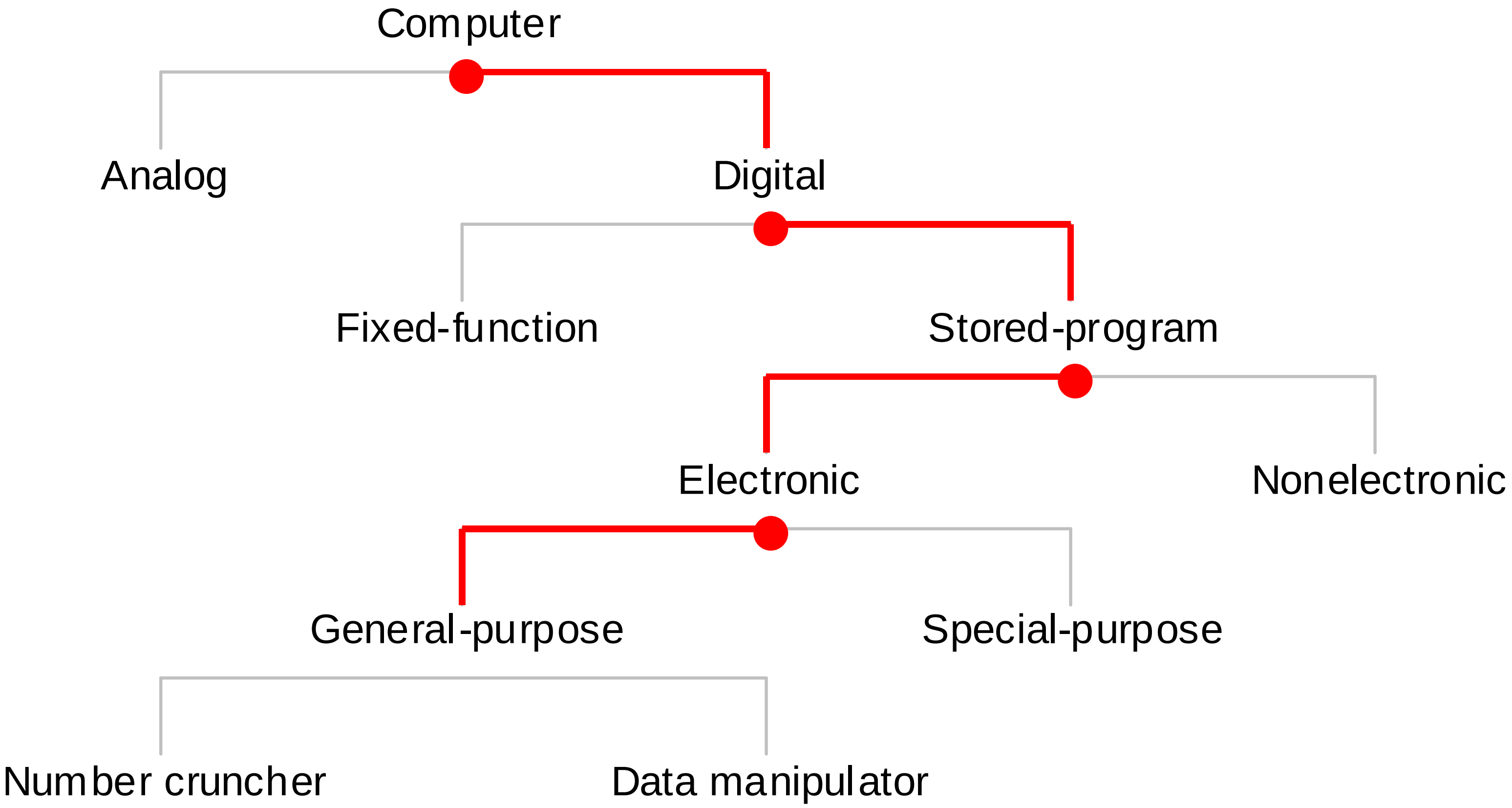


Figure 3.3 The space of computer systems, with what we normally mean by the word “computer” highlighted.

Price/Performance Pyramid

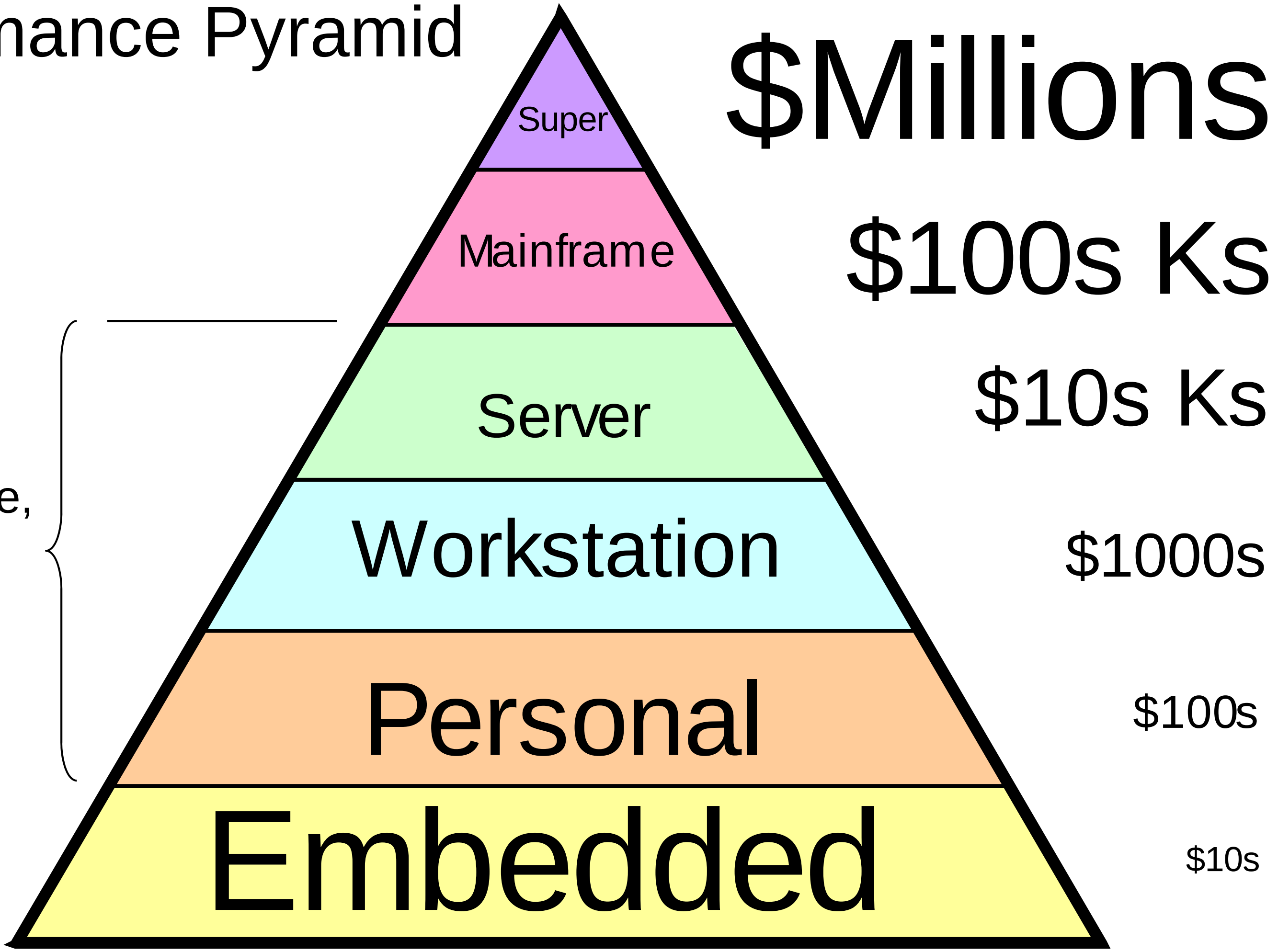


Figure 3.4 Classifying computers by computational power and price range.

Automotive Embedded Computers

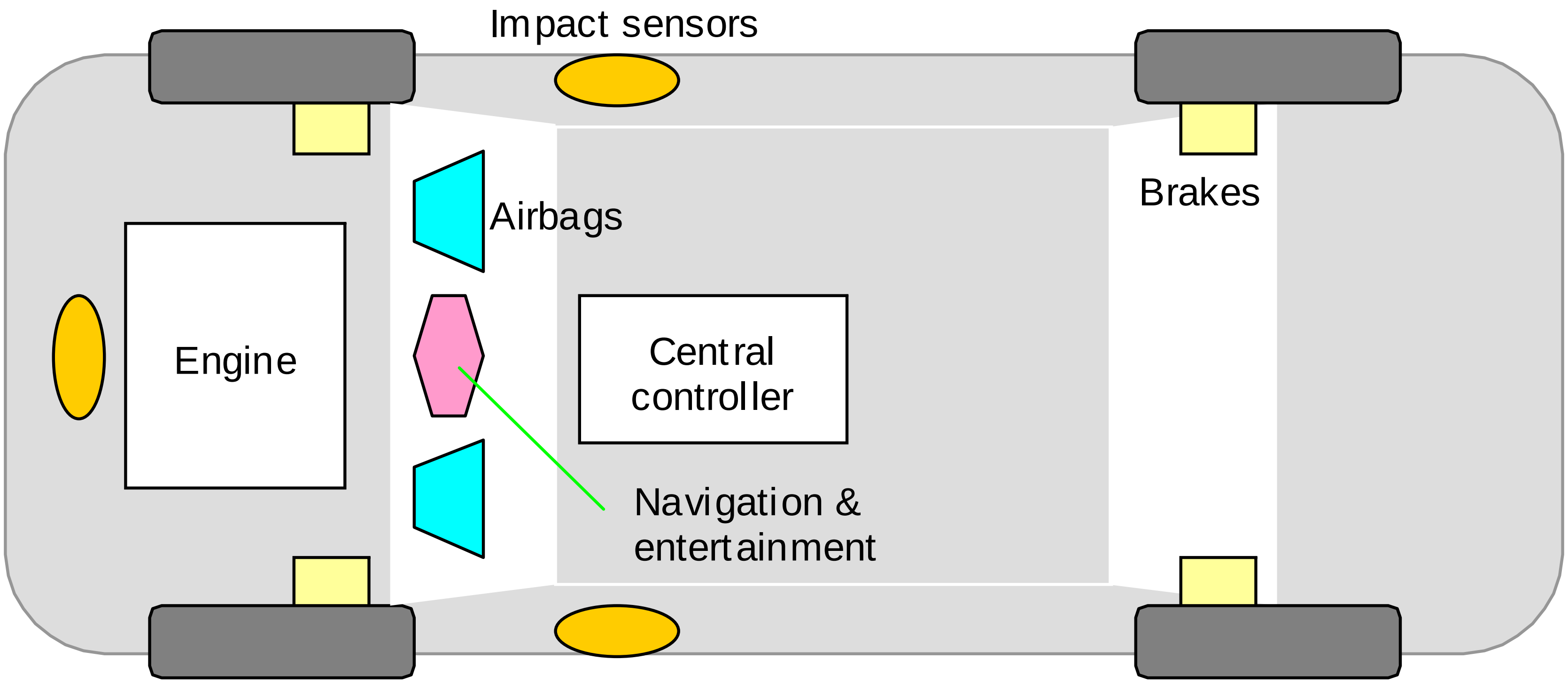


Figure 3.5 Embedded computers are ubiquitous, yet invisible. They are found in our automobiles, appliances, and many other places.

Personal Computers and Workstations

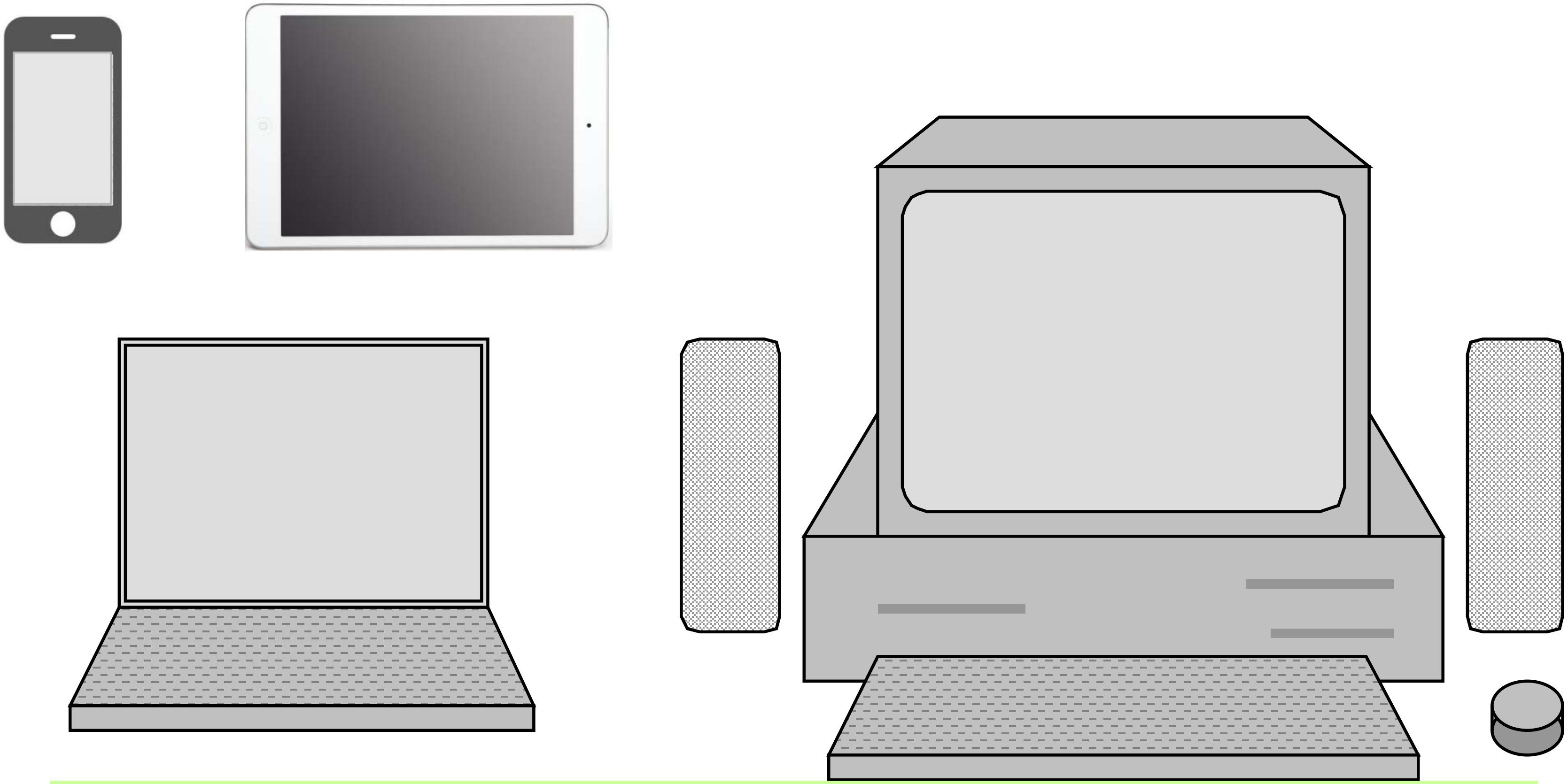


Figure 3.6 Notebooks, a common class of portable computers, are much smaller than desktops but offer substantially the same capabilities. What are the main reasons for the size difference?

Digital Computer Subsystems

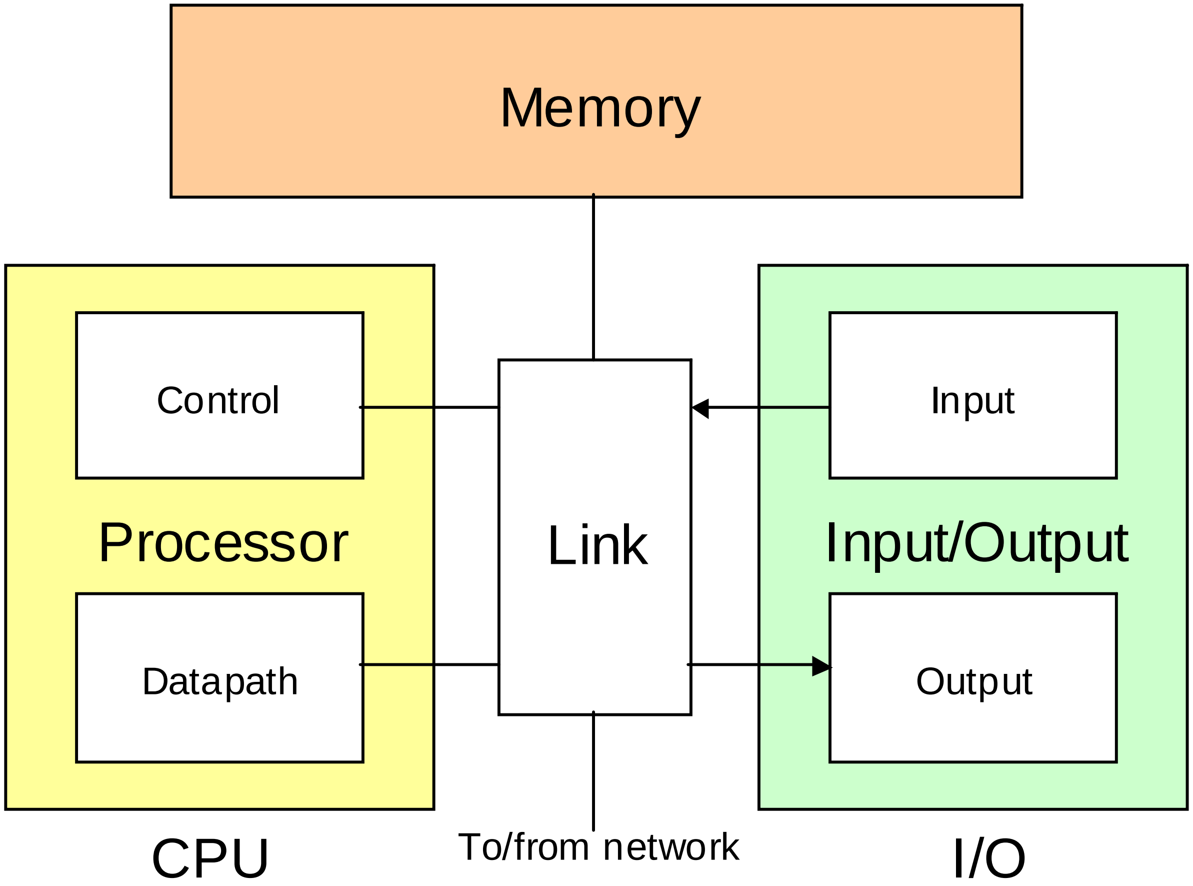


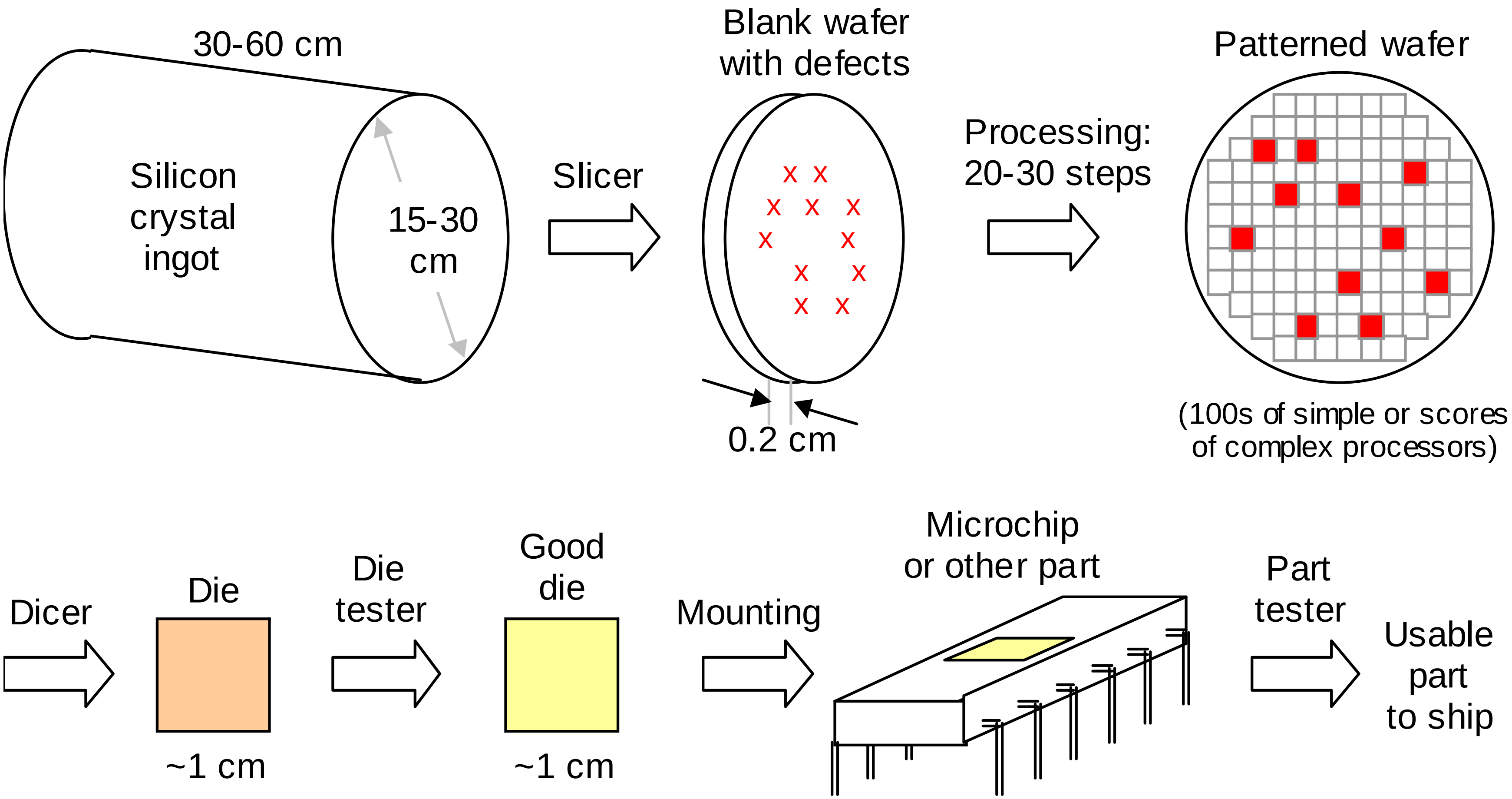
Figure 3.7 The (three, four, five, or) six main units of a digital computer. Usually, the link unit (a simple bus or a more elaborate network) is not explicitly included in such diagrams.

3.3 Generations of Progress

Table 3.2 The 5 generations of digital computers, and their ancestors.

Generation (begun)	Processor technology	Memory innovations	I/O devices introduced	Dominant look & fell
0 (1600s)	(Electro-) mechanical	Wheel, card	Lever, dial, punched card	Factory equipment
1 (1950s)	Vacuum tube	Magnetic drum	Paper tape, magnetic tape	Hall-size cabinet
2 (1960s)	Transistor	Magnetic core	Drum, printer, text terminal	Room-size mainframe
3 (1970s)	SSI/MSI	RAM/ROM chip	Disk, keyboard, video monitor	Desk-size mini
4 (1980s)	LSI/VLSI	SRAM/DRAM	Network, CD, mouse,sound	Desktop/ laptop micro
5 (1990s)	ULSI/GSI/ WSI, SOC	SDRAM, flash	Sensor/actuator, point/click	Invisible, embedded

IC Production and Yield



30-60 cm

Silicon crystal ingot

15-30 cm

Slicer

Blank wafer with defects

0.2 cm

Processing: 20-30 steps

Patterned wafer

(100s of simple or scores of complex processors)

Dicer

Die

~1 cm

Die tester

Good die

~1 cm

Mounting

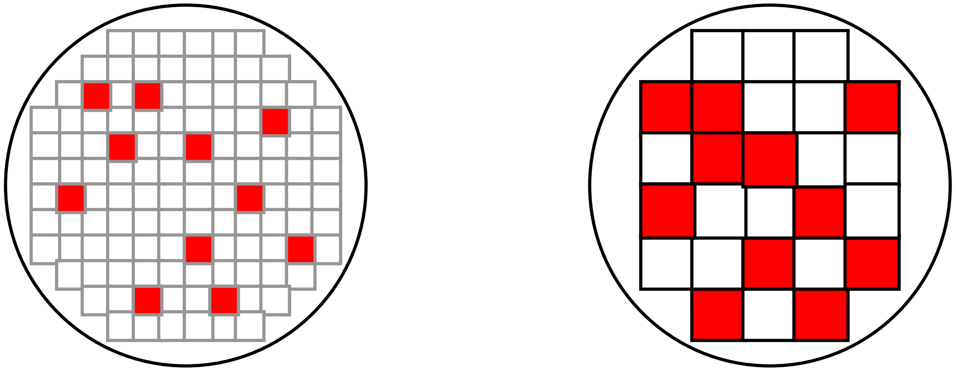
Microchip or other part

Part tester

Usable part to ship

Figure 3.8 The manufacturing process for an IC part.

Effect of Die Size on Yield



120 dies, 109 good

26 dies, 15 good

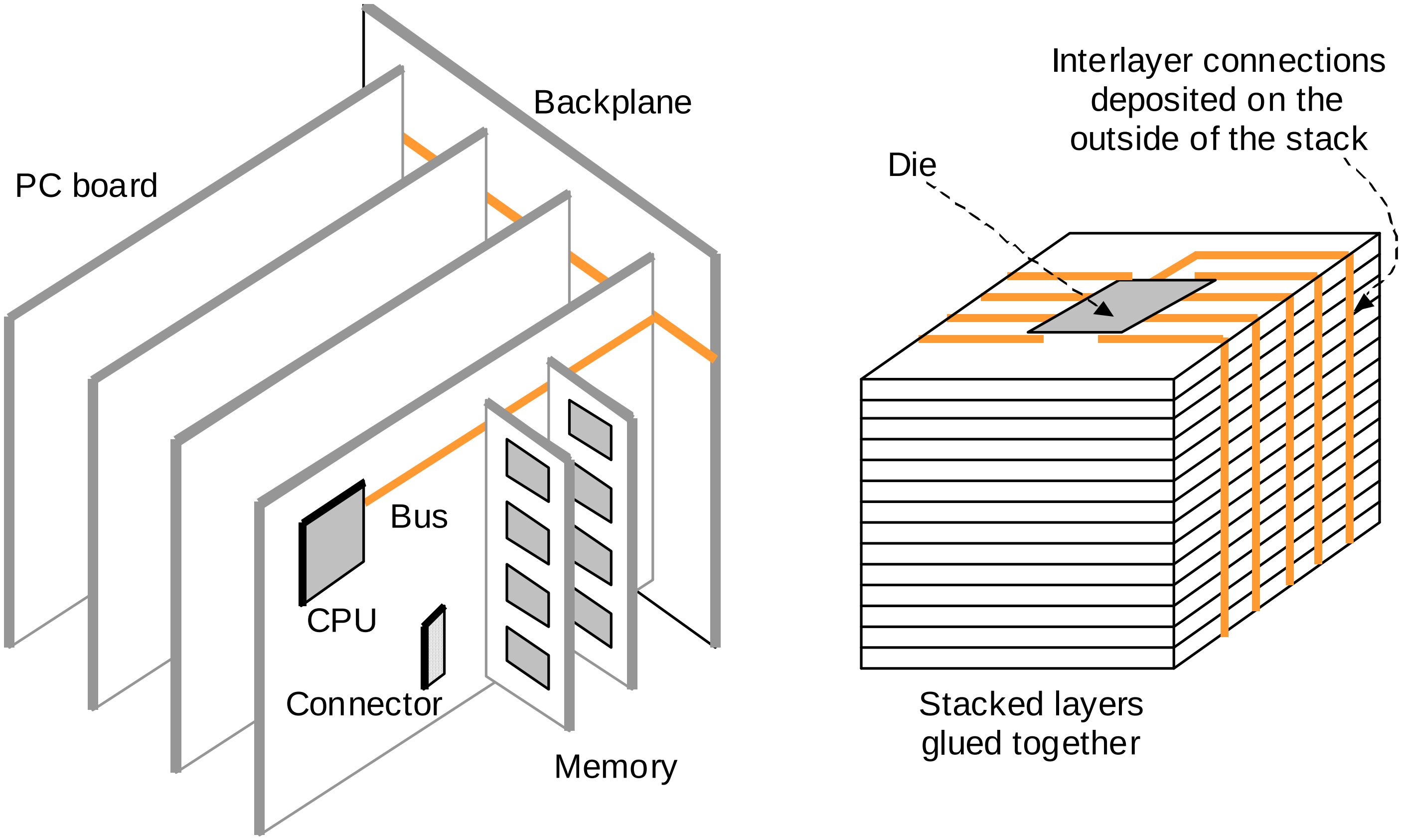
Figure 3.9 Visualizing the dramatic decrease in yield with larger dies.

Die yield $=_{\text{def}}$ (number of good dies) / (total number of dies)

Die yield = Wafer yield × $[1 + (\text{Defect density} \times \text{Die area}) / a]^{-a}$

Die cost = (cost of wafer) / (total number of dies × die yield)
= (cost of wafer) × (die area / wafer area) / (die yield)

3.4 Processor and Memory Technologies


Backplane
PC board
Bus
CPU
Connector
Memory
Interlayer connections deposited on the outside of the stack
Die
Stacked layers glued together

(a) 2D or 2.5D packaging now common

(b) 3D packaging of the future

Figure 3.11 Packaging of processor, memory, and other components.

Moore's Law

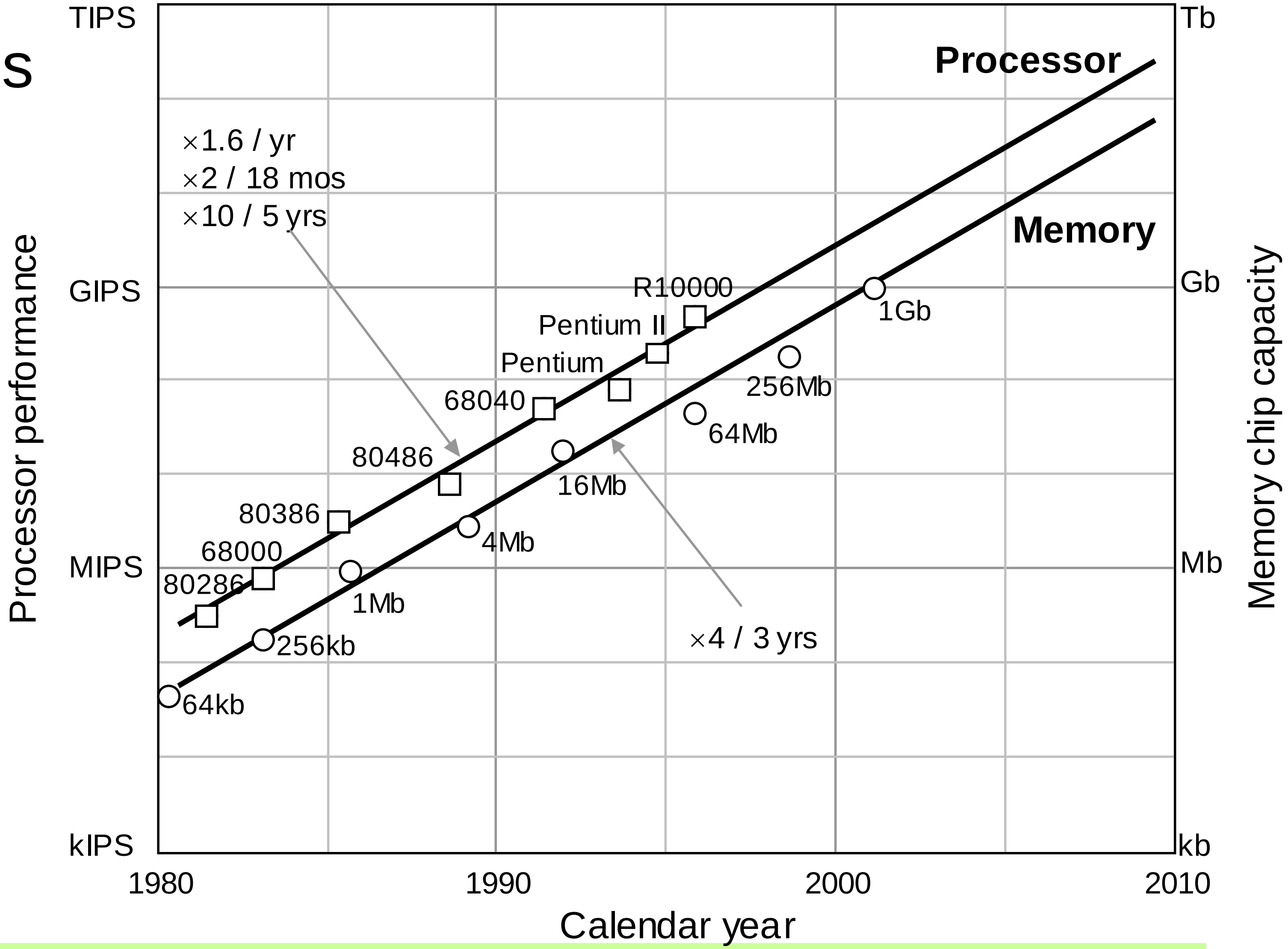


Figure 3.10 Trends in processor performance and DRAM memory chip capacity (Moore's law).

Pitfalls of Computer Technology Forecasting

“DOS addresses only 1 MB of RAM because we cannot imagine any applications needing more.” Microsoft, 1980

“640K ought to be enough for anybody.” Bill Gates, 1981

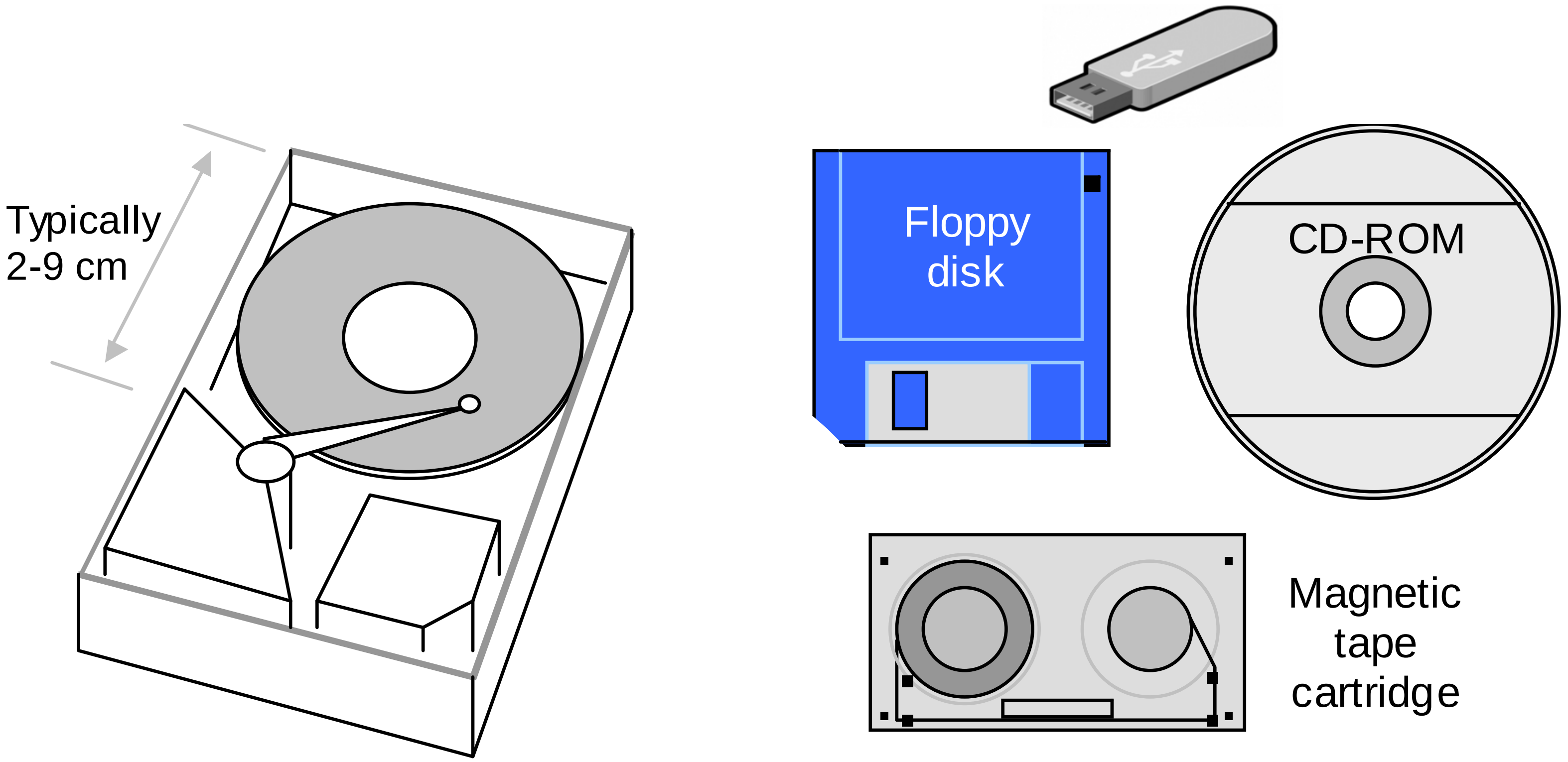
“Computers in the future may weigh no more than 1.5 tons.” *Popular Mechanics*

“I think there is a world market for maybe five computers.” Thomas Watson, IBM Chairman, 1943

“There is no reason anyone would want a computer in their home.” Ken Olsen, DEC founder, 1977

“The 32-bit machine would be an overkill for a personal computer.” Sol Libes, *ByteLines*

3.5 Input/Output and Communications



(a) Cutaway view of a hard disk drive

(b) Some removable storage media

Figure 3.12 Magnetic and optical disk memory units.

Communication Technologies

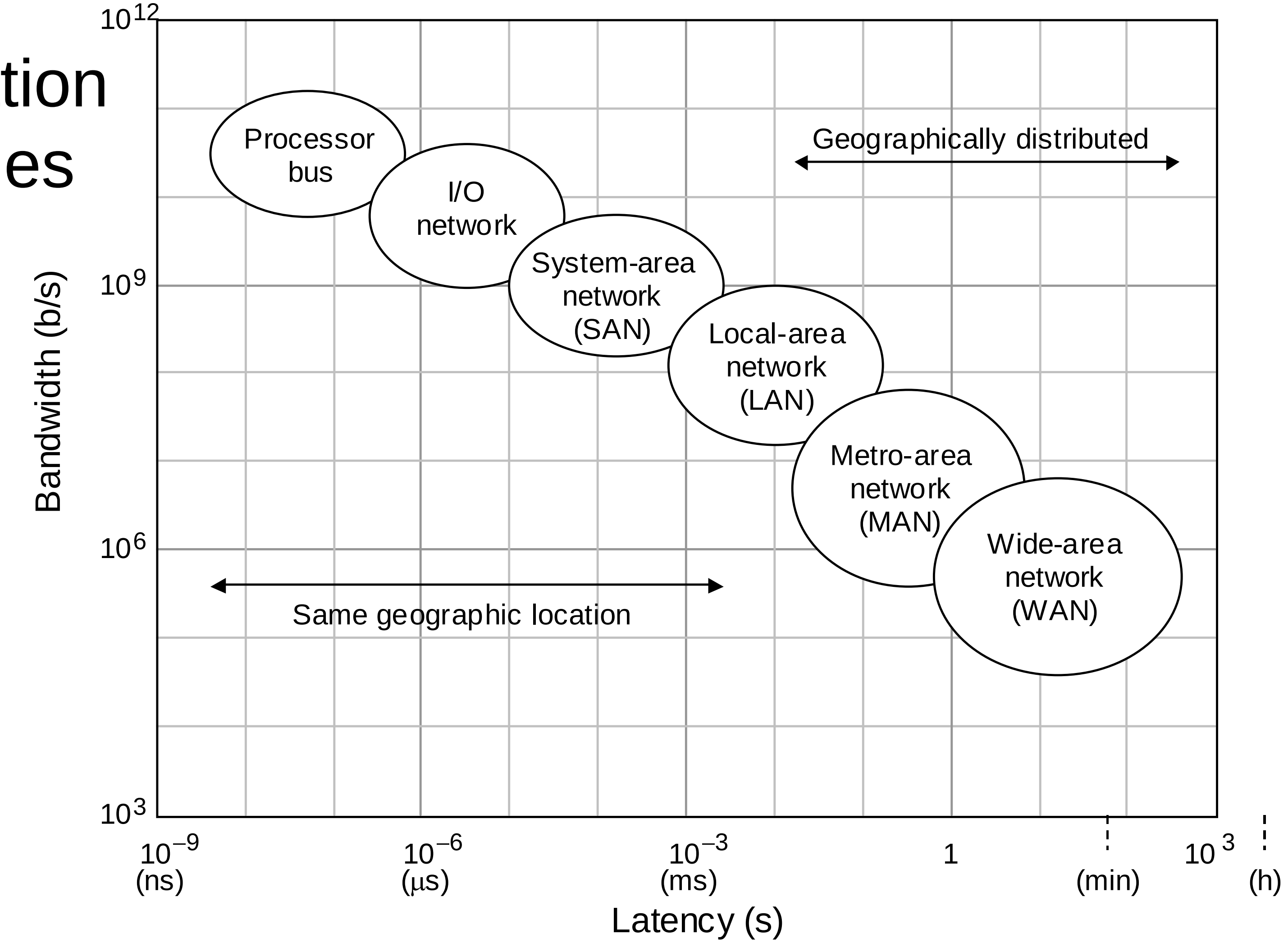
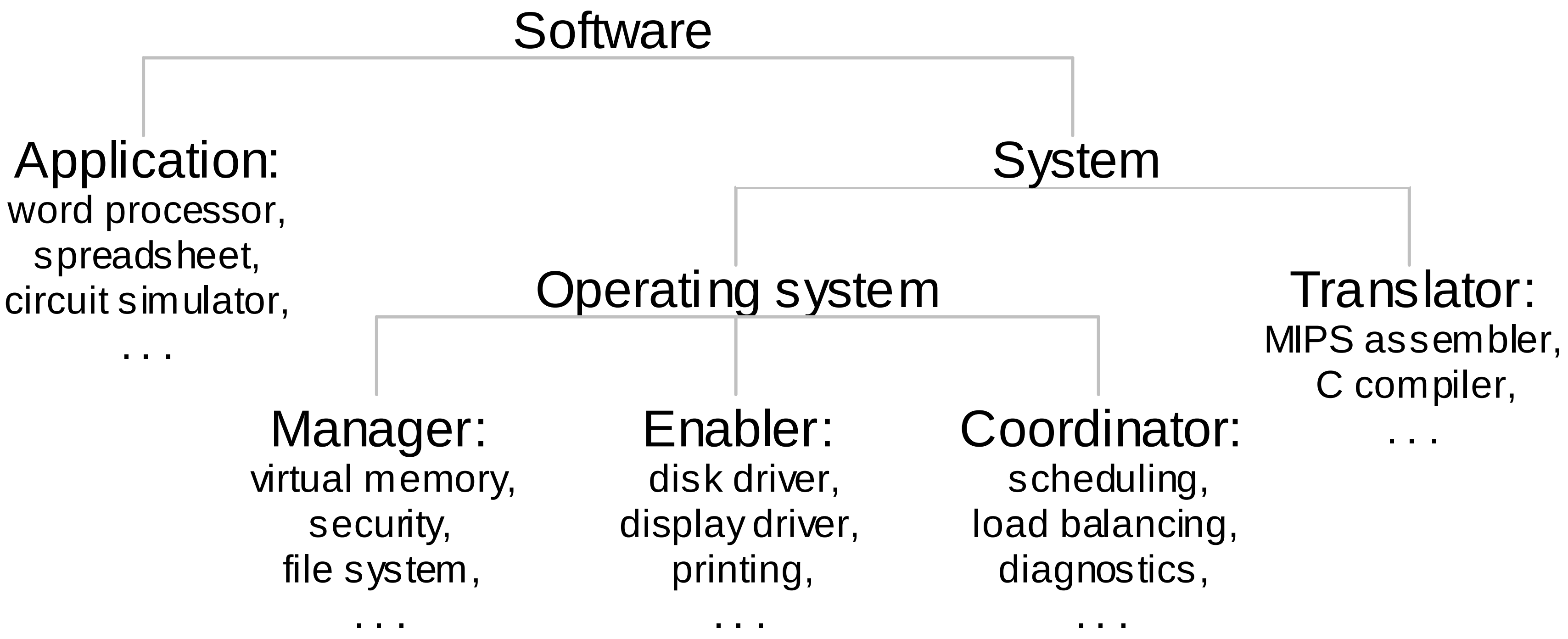


Figure 3.13 Latency and bandwidth characteristics of different classes of communication links.

3.6 Software Systems and Applications



Software

Application:
word processor,
spreadsheet,
circuit simulator,
. . .

System

Operating system

Manager:
virtual memory,
security,
file system,
. . .

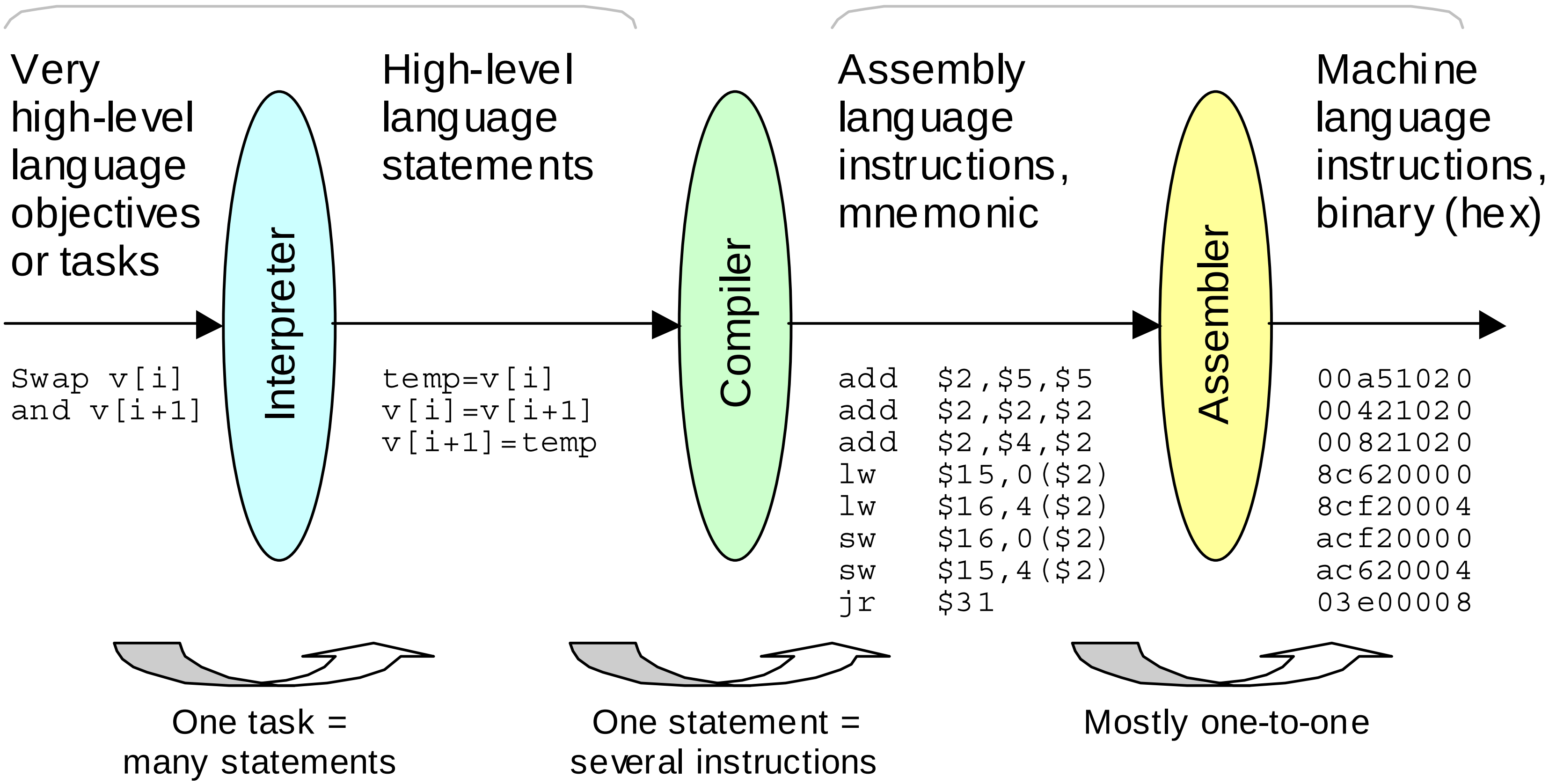
Enabler:
disk driver,
display driver,
printing,
. . .

Coordinator:
scheduling,
load balancing,
diagnostics,
. . .

Translator:
MIPS assembler,
C compiler,
. . .

Figure 3.15 Categorization of software, with examples in each class.

High- vs Low-Level Programming



More abstract, machine-independent; easier to write, read, debug, or maintain

More concrete, machine-specific, error-prone; harder to write, read, debug, or maintain

Very high-level language objectives or tasks

```
Swap v[i]
and v[i+1]
```

High-level language statements

```
temp=v[i]
v[i]=v[i+1]
v[i+1]=temp
```

Assembly language instructions, mnemonic

```
add   $2,$5,$5
add   $2,$2,$2
add   $2,$4,$2
lw    $15,0($2)
lw    $16,4($2)
sw    $16,0($2)
sw    $15,4($2)
jr    $31
```

Machine language instructions, binary (hex)

```
00a51020
00421020
00821020
8c620000
8cf20004
acf20000
ac620004
03e00008
```

One task = many statements

One statement = several instructions

Mostly one-to-one

Figure 3.14 Models and abstractions in programming.

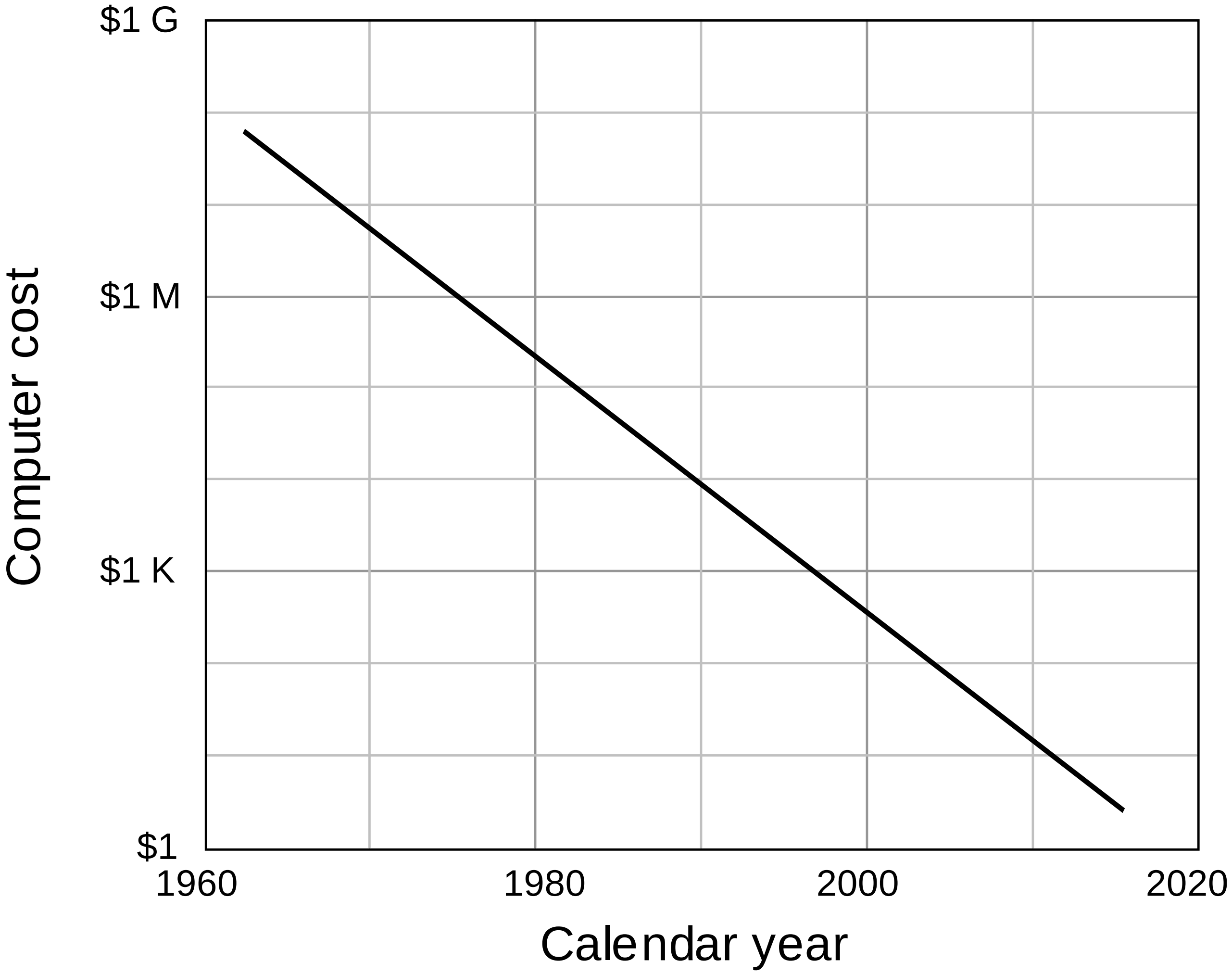
4 Computer Performance

Performance is key in design decisions; also cost and power

- It has been a driving force for innovation
- Isn’t quite the same as speed (higher clock rate)

Topics in This Chapter
4.1 Cost, Performance, and Cost/Performance
4.2 Defining Computer Performance
4.3 Performance Enhancement and Amdahl’s Law
4.4 Performance Measurement vs Modeling
4.5 Reporting Computer Performance
4.6 The Quest for Higher Performance

4.1 Cost, Performance, and Cost/Performance



Cost/Performance

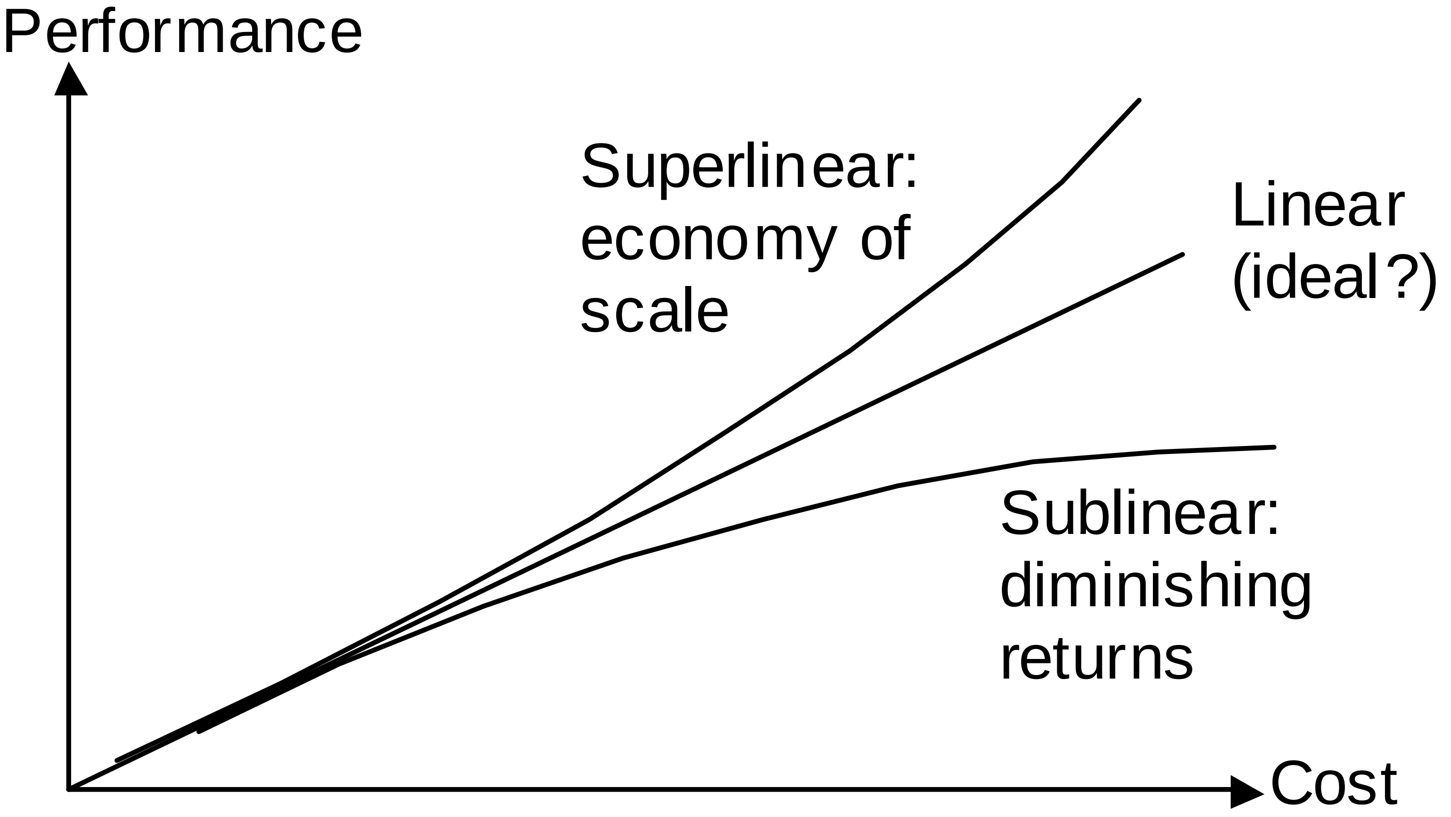


Figure 4.1 Performance improvement as a function of cost.

4.2 Defining Computer Performance

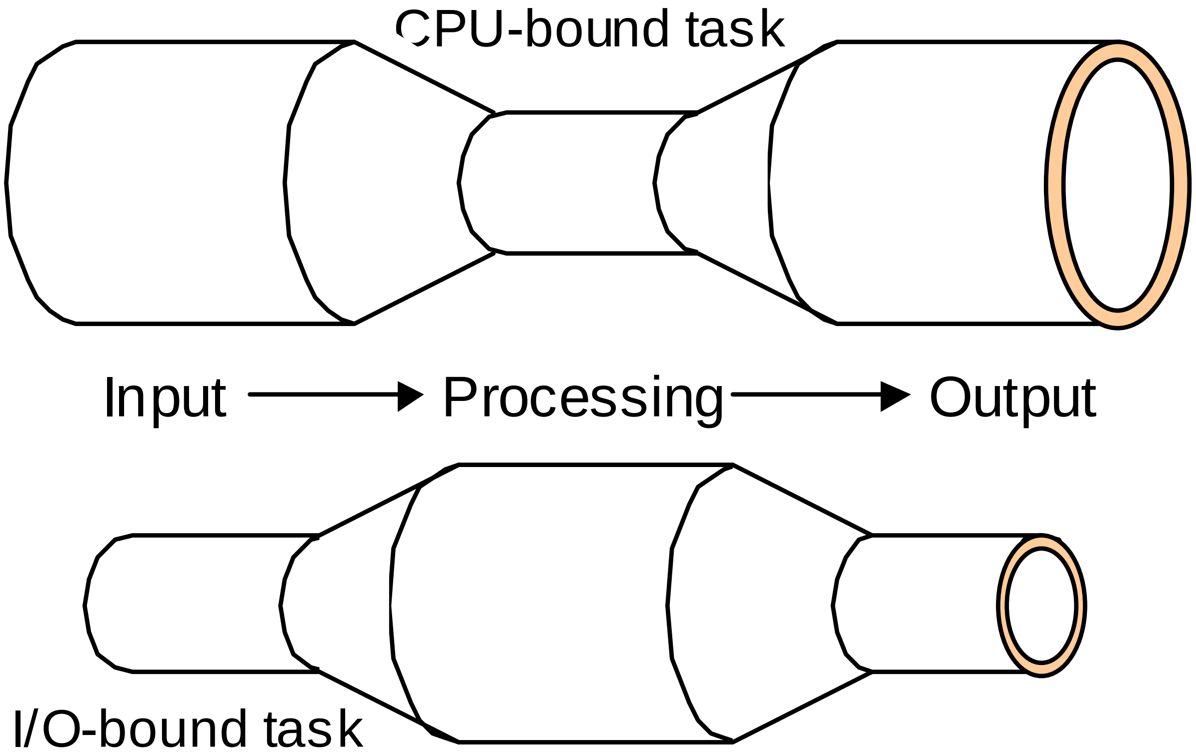


Figure 4.2 Pipeline analogy shows that imbalance between processing power and I/O capabilities leads to a performance bottleneck.

Six Passenger Aircraft to Be Compared

B 747

DC-8-50

Performance of Aircraft: An Analogy

Table 4.1 Key characteristics of six passenger aircraft: all figures are approximate; some relate to a specific model/configuration of the aircraft or are averages of cited range of values.

Aircraft	**Passengers**	**Range (km)**	**Speed (km/h)**	**Price ($M)**
Airbus A310	250	8 300	895	120
Boeing 747	470	6 700	980	200
Boeing 767	250	12 300	885	120
Boeing 777	375	7 450	980	180
Concorde	130	6 400	2 200	350
DC-8-50	145	14 000	875	80

Speed of sound ≈ 1220 km / h

Different Views of Performance

Performance from the viewpoint of a passenger: Speed

Note, however, that flight time is but one part of total travel time. Also, if the travel distance exceeds the range of a faster plane, a slower plane may be better due to not needing a refueling stop

Performance from the viewpoint of an airline: Throughput

Measured in passenger-km per hour (relevant if ticket price were proportional to distance traveled, which in reality it is not)

Airbus A310	250 × 895 = 0.224 M passenger-km/hr
Boeing 747	470 × 980 = 0.461 M passenger-km/hr
Boeing 767	250 × 885 = 0.221 M passenger-km/hr
Boeing 777	375 × 980 = 0.368 M passenger-km/hr
Concorde	130 × 2200 = 0.286 M passenger-km/hr
DC-8-50	145 × 875 = 0.127 M passenger-km/hr

Performance from the viewpoint of FAA: Safety

Cost Effectiveness: Cost/Performance

Table 4.1 Key characteristics of six passenger aircraft: all figures are approximate; some relate to a specific model/configuration of the aircraft or are averages of cited range of values.

					Larger values better	Smaller values better
Aircraft	**Passengers**	**Range (km)**	**Speed (km/h)**	**Price ($M)**	Throughput (M P km/hr)	Cost / Performance
A310	250	8 300	895	120	**0.224**	**536**
B 747	470	6 700	980	200	**0.461**	**434**
B 767	250	12 300	885	120	**0.221**	**543**
B 777	375	7 450	980	180	**0.368**	**489**
Concorde	130	6 400	2 200	350	**0.286**	**1224**
DC-8-50	145	14 000	875	80	**0.127**	**630**

Concepts of Performance and Speedup

Performance = 1 / Execution time   is simplified to

Performance = 1 / CPU execution time

(Performance of M_1) / (Performance of M_2) = Speedup of M_1 over M_2
= (Execution time of M_2) / (Execution time M_1)

Terminology: M_1 is x times **as fast as** M_2 (e.g., 1.5 times as fast)
M_1 is $100(x - 1)\%$ **faster than** M_2 (e.g., 50% faster)

CPU time = Instructions × (Cycles per instruction) × (Secs per cycle)
= Instructions × CPI / (Clock rate)

Instruction count, CPI, and clock rate are not completely independent, so improving one by a given factor may not lead to overall execution time improvement by the same factor.

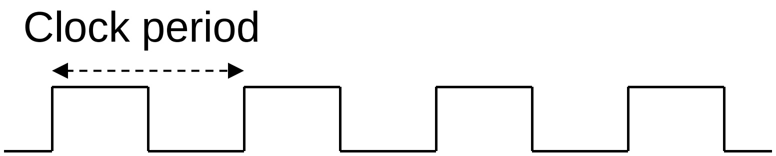
Elaboration on the CPU Time Formula

CPU time = Instructions × (Cycles per instruction) × (Secs per cycle)
= Instructions × Average CPI / (Clock rate)

Instructions: Number of instructions executed, not number of instructions in our program (dynamic count)

Average CPI: Is calculated based on the dynamic instruction mix and knowledge of how many clock cycles are needed to execute various instructions (or instruction classes)

Clock rate: 1 GHz = 10^9 cycles / s (cycle time 10^{-9} s = 1 ns)
200 MHz = 200 × 10^6 cycles / s (cycle time = 5 ns)



Dynamic Instruction Count

How many instructions are executed in this program fragment?

Each “for” consists of two instructions: increment index, check exit condition

```
250 instructions
for i = 1, 100 do
20 instructions
   for j = 1, 100 do
   40 instructions
      for k = 1, 100 do
      10 instructions
      endfor
   endfor
endfor
```

Static count = 326

12,422,450 Instructions

2 + 20 + 124,200 instructions
100 iterations
12,422,200 instructions in all

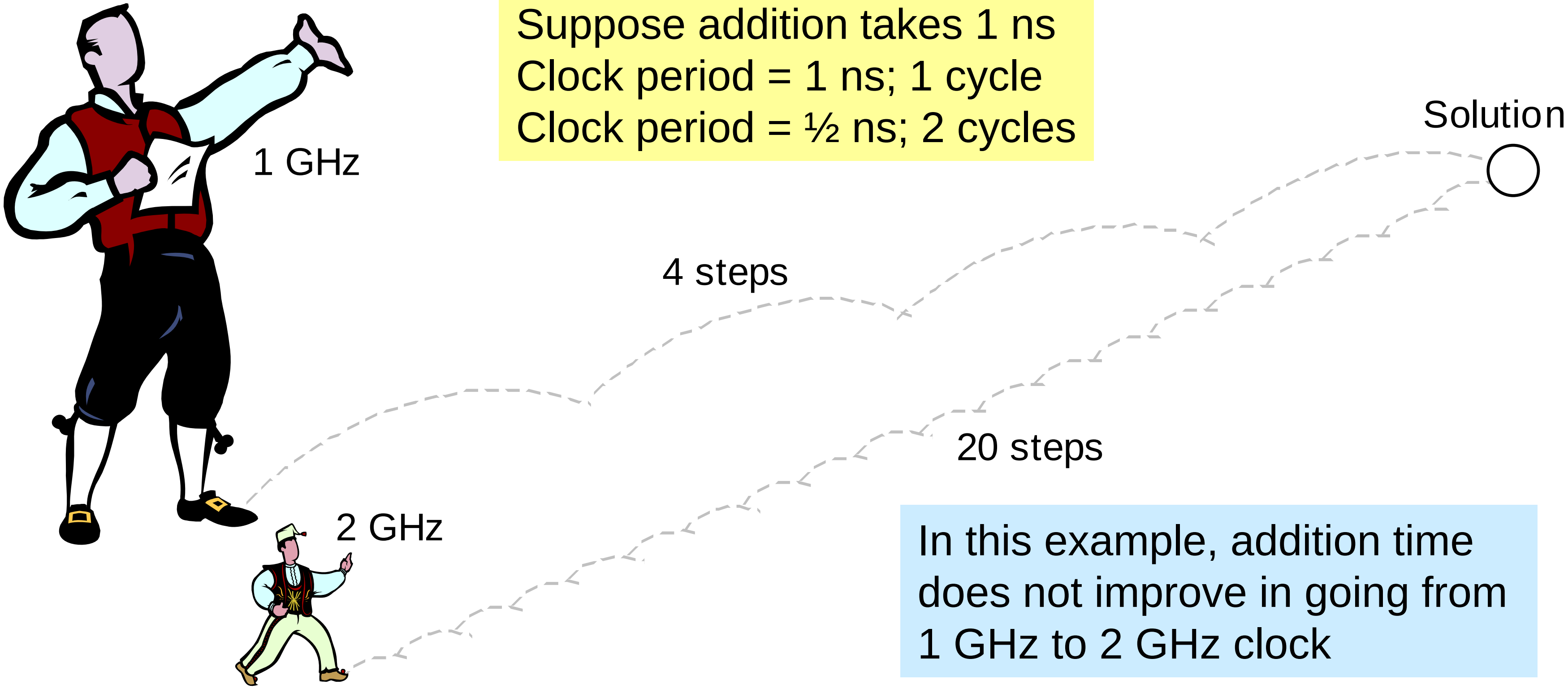
2 + 40 + 1200 instructions
100 iterations
124,200 instructions in all

2 + 10 instructions
100 iterations
1200 instructions in all

for i = 1, n
while x > 0

Faster Clock ≠ Shorter Running Time

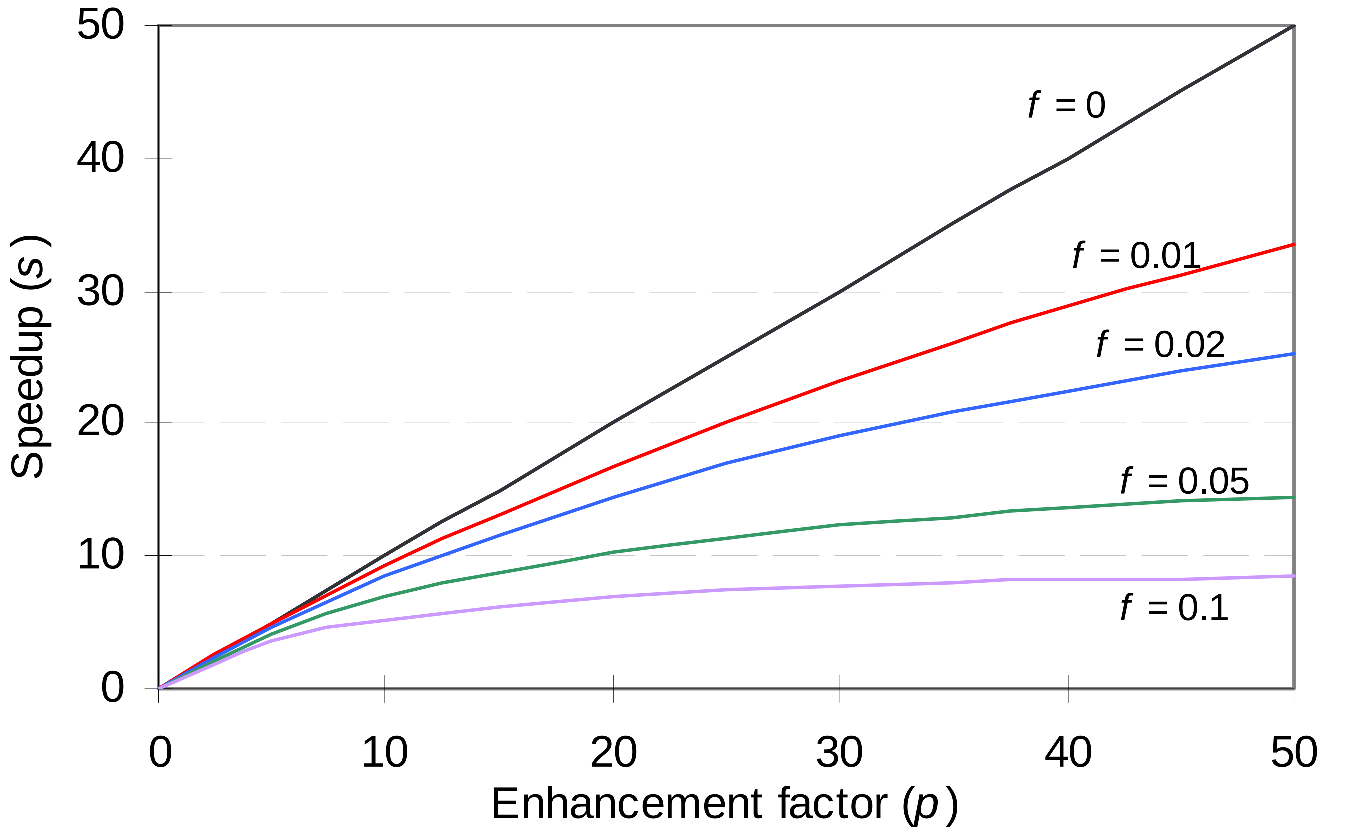
Suppose addition takes 1 ns
Clock period = 1 ns; 1 cycle
Clock period = ½ ns; 2 cycles



In this example, addition time does not improve in going from 1 GHz to 2 GHz clock

Figure 4.3 Faster steps do not necessarily mean shorter travel time.

4.3 Performance Enhancement: Amdahl's Law



f = fraction unaffected

p = speedup of the rest

$$s = \frac{1}{f + (1 - f)/p}$$

$$\leq min(p, 1/f)$$

Figure 4.4 Amdahl's law: speedup achieved if a fraction *f* of a task is unaffected and the remaining 1 – *f* part runs *p* times as fast.

Amdahl's Law Used in Design

Example 4.1

A processor spends 30% of its time on flp addition, 25% on flp mult, and 10% on flp division. Evaluate the following enhancements, each costing the same to implement:

a. Redesign of the flp adder to make it twice as fast.
b. Redesign of the flp multiplier to make it three times as fast.
c. Redesign the flp divider to make it 10 times as fast.

Solution

a. Adder redesign speedup = 1 / [0.7 + 0.3 / 2] = 1.18
b. Multiplier redesign speedup = 1 / [0.75 + 0.25 / 3] = 1.20
c. Divider redesign speedup = 1 / [0.9 + 0.1 / 10] = 1.10

What if both the adder and the multiplier are redesigned?

Amdahl’s Law Used in Management

Example 4.2

Members of a university research group frequently visit the library. Each library trip takes 20 minutes. The group decides to subscribe to a handful of publications that account for 90% of the library trips; access time to these publications is reduced to 2 minutes.

a. What is the average speedup in access to publications?

b. If the group has 20 members, each making two weekly trips to the library, what is the justifiable expense for the subscriptions? Assume 50 working weeks/yr and $25/h for a researcher’s time.

Solution

a. Speedup in publication access time = 1 / [0.1 + 0.9 / 10] = 5.26

b. Time saved = 20 × 2 × 50 × 0.9 (20 – 2) = 32,400 min = 540 h
Cost recovery = 540 × $25 = $13,500 = Max justifiable expense

4.4 Performance Measurement vs Modeling

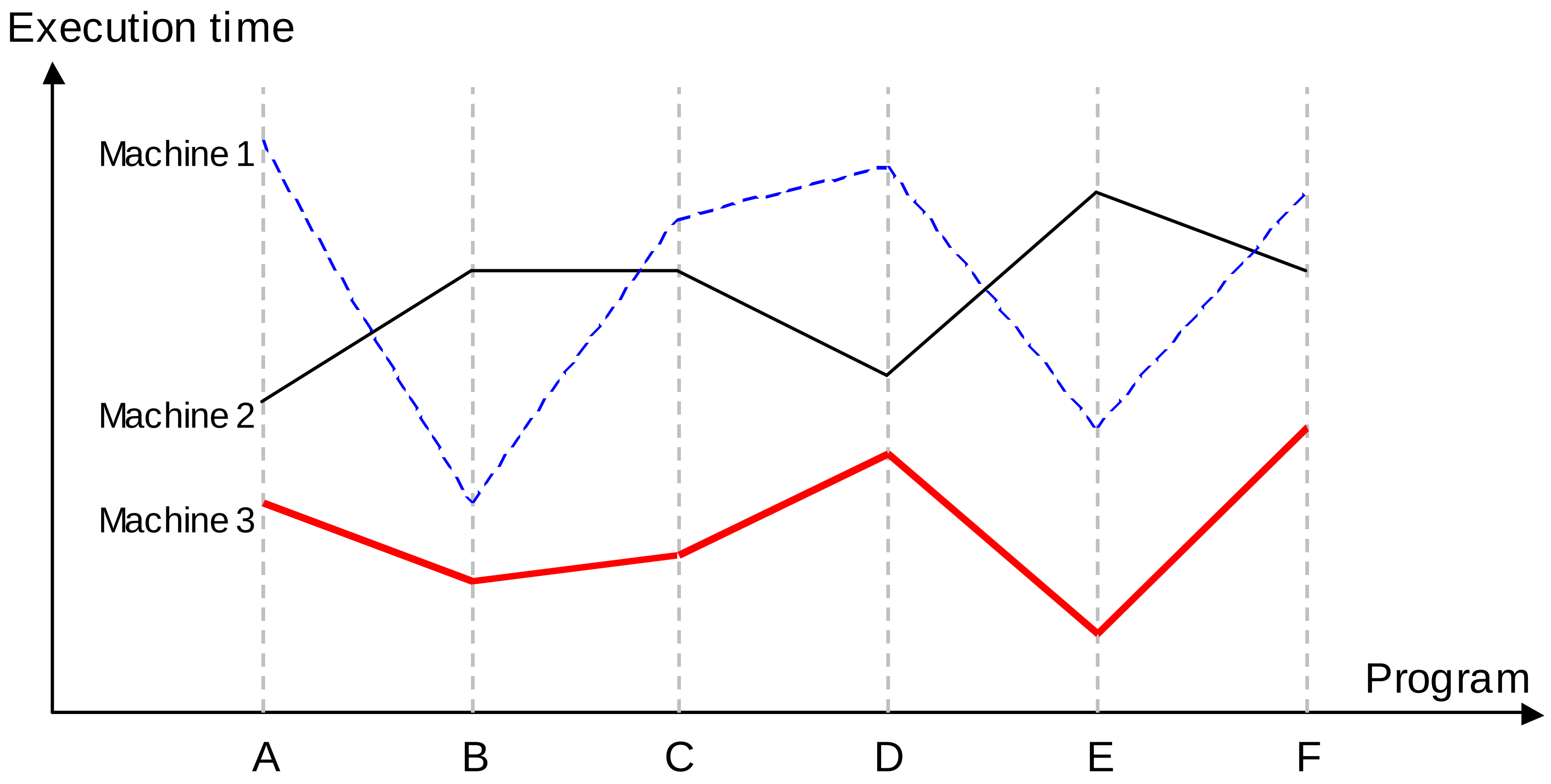


Figure 4.5 Running times of six programs on three machines.

Generalized Amdahl’s Law

Original running time of a program = 1 = $f_1 + f_2 + \ldots + f_k$

New running time after the fraction f_i is speeded up by a factor p_i

$$\frac{f_1}{p_1} + \frac{f_2}{p_2} + \ . \ . \ . \ + \frac{f_k}{p_k}$$

Speedup formula

$$S = \frac{1}{\dfrac{f_1}{p_1} + \dfrac{f_2}{p_2} + \ . \ . \ . \ + \dfrac{f_k}{p_k}}$$

If a particular fraction is slowed down rather than speeded up, use $s_j f_j$ instead of f_j / p_j, where $s_j > 1$ is the slowdown factor

Performance Benchmarks

Example 4.3

You are an engineer at Outtel, a start-up aspiring to compete with Intel via its new processor design that outperforms the latest Intel processor by a factor of 2.5 on floating-point instructions. This level of performance was achieved by design compromises that led to a 20% increase in the execution time of all other instructions. You are in charge of choosing benchmarks that would showcase Outtel's performance edge.

a. What is the minimum required fraction *f* of time spent on floating-point instructions in a program on the Intel processor to show a speedup of 2 or better for Outtel?

Solution

a. We use a generalized form of Amdahl's formula in which a fraction *f* is speeded up by a given factor (2.5) and the rest is slowed down by another factor (1.2): $1 / [1.2(1 - f) + f / 2.5] \geq 2 \Rightarrow f \geq 0.875$

Performance Estimation

$$\text{Average CPI} = \sum_{\text{All instruction classes}} (\text{Class-}i \text{ fraction}) \times (\text{Class-}i \text{ CPI})$$

$$\text{Machine cycle time} = 1 / \text{Clock rate}$$

$$\text{CPU execution time} = \text{Instructions} \times (\text{Average CPI}) / (\text{Clock rate})$$

Table 4.3 Usage frequency, in percentage, for various instruction classes in four representative applications.

Application → **Instr'n class ↓**	**Data compression**	**C language compiler**	**Reactor simulation**	**Atomic motion modeling**
A: Load/Store	25	37	32	37
B: Integer	32	28	17	5
C: Shift/Logic	16	13	2	1
D: Float	0	0	34	42
E: Branch	19	13	9	10
F: All others	8	9	6	4

CPI and IPS Calculations

Example 4.4 (2 of 5 parts)

Consider two implementations M_1 (600 MHz) and M_2 (500 MHz) of an instruction set containing three classes of instructions:

Class	CPI for M_1	CPI for M_2	Comments
F	5.0	4.0	Floating-point
I	2.0	3.8	Integer arithmetic
N	2.4	2.0	Nonarithmetic

a. What are the peak performances of M_1 and M_2 in MIPS?

b. If 50% of instructions executed are class-N, with the rest divided equally among F and I, which machine is faster? By what factor?

Solution

a. Peak MIPS for $M_1 = 600 / 2.0 = 300$; for $M_2 = 500 / 2.0 = 250$

b. Average CPI for $M_1 = 5.0 / 4 + 2.0 / 4 + 2.4 / 2 = 2.95$;

for $M_2 = 4.0/4 + 3.8/4 + 2.0/2 = 2.95$ → M_1 is faster; factor 1.2

MIPS Rating Can Be Misleading

Example 4.5

Two compilers produce machine code for a program on a machine with two classes of instructions. Here are the number of instructions:

Class	CPI	Compiler 1	Compiler 2
A	1	600M	400M
B	2	400M	400M

a. What are run times of the two programs with a 1 GHz clock?
b. Which compiler produces faster code and by what factor?
c. Which compiler's output runs at a higher MIPS rate?

Solution

a. Running time 1 (2) = (600M × 1 + 400M × 2) / 10^9 = 1.4 s (1.2 s)
b. Compiler 2's output runs 1.4 / 1.2 = 1.17 times as fast
c. MIPS rating 1, CPI = 1.4 (2, CPI = 1.5) = 1000 / 1.4 = 714 (667)

4.5 Reporting Computer Performance

Table 4.4 Measured or estimated execution times for three programs.

	Time on machine X	Time on machine Y	Speedup of Y over X
Program A	20	200	0.1
Program B	1000	100	10.0
Program C	1500	150	10.0
All 3 prog’s	2520	450	5.6

Analogy: If a car is driven to a city 100 km away at 100 km/hr and returns at 50 km/hr, the average speed is not (100 + 50) / 2 but is obtained from the fact that it travels 200 km in 3 hours.

Comparing the Overall Performance

Table 4.4 Measured or estimated execution times for three programs.

	Time on machine X	Time on machine Y	Speedup of Y over X	**Speedup of X over Y**
Program A	20	200	0.1	10
Program B	1000	100	10.0	0.1
Program C	1500	150	10.0	0.1
	Arithmetic mean		6.7	3.4
	Geometric mean		2.15	0.46

Geometric mean does not yield a measure of overall speedup, but provides an indicator that at least moves in the right direction

Effect of Instruction Mix on Performance

Example 4.6 (1 of 3 parts)

Consider two applications DC and RS and two machines M_1 and M_2:

Class	Data Comp.	Reactor Sim.	M_1's CPI	M_2's CPI
A: Ld/Str	25%	32%	4.0	3.8
B: Integer	32%	17%	1.5	2.5
C: Sh/Logic	16%	2%	1.2	1.2
D: Float	0%	34%	6.0	2.6
E: Branch	19%	9%	2.5	2.2
F: Other	8%	6%	2.0	2.3

a. Find the effective CPI for the two applications on both machines.

Solution

a. CPI of DC on M_1: $0.25 \times 4.0 + 0.32 \times 1.5 + 0.16 \times 1.2 + 0 \times 6.0 +$
$0.19 \times 2.5 + 0.08 \times 2.0 = 2.31$

DC on M_2: 2.54 RS on M_1: 3.94 RS on M_2: 2.89

4.6 The Quest for Higher Performance

State of available computing power ca. the early 2000s:

Gigaflops on the desktop
Teraflops in the supercomputer center
Petaflops on the drawing board

Note on terminology (see Table 3.1)

Prefixes for large units:
Kilo = 10^3, Mega = 10^6, Giga = 10^9, Tera = 10^{12}, Peta = 10^{15}

For memory:
K = 2^{10} = 1024, M = 2^{20}, G = 2^{30}, T = 2^{40}, P = 2^{50}

Prefixes for small units:
micro = 10^{-6}, nano = 10^{-9}, pico = 10^{-12}, femto = 10^{-15}

Performance Trends and Obsolescence

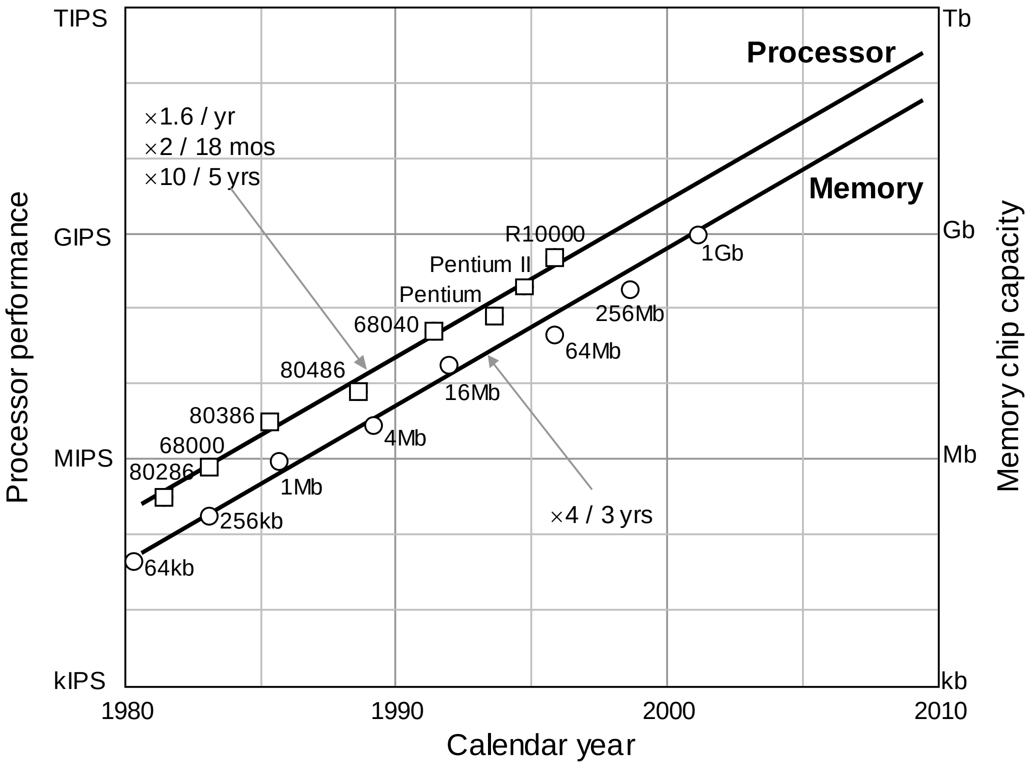


Figure 3.10   Trends in processor performance and DRAM memory chip capacity (Moore's law).

"Can I call you back? We just bought a new computer and we're trying to set it up before it's obsolete."

Super-computers

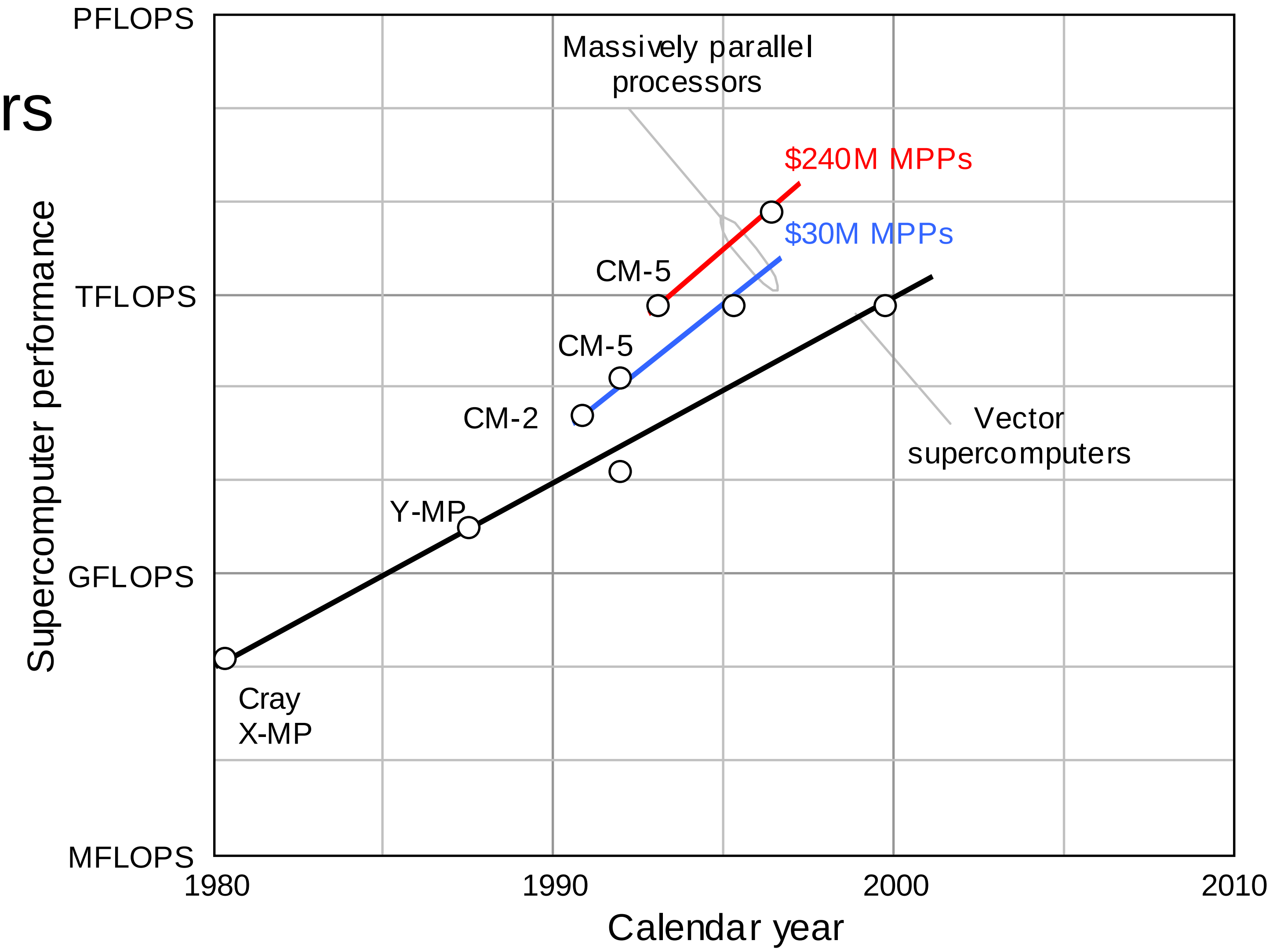


Figure 4.7 Exponential growth of supercomputer performance.

The Most Powerful Computers

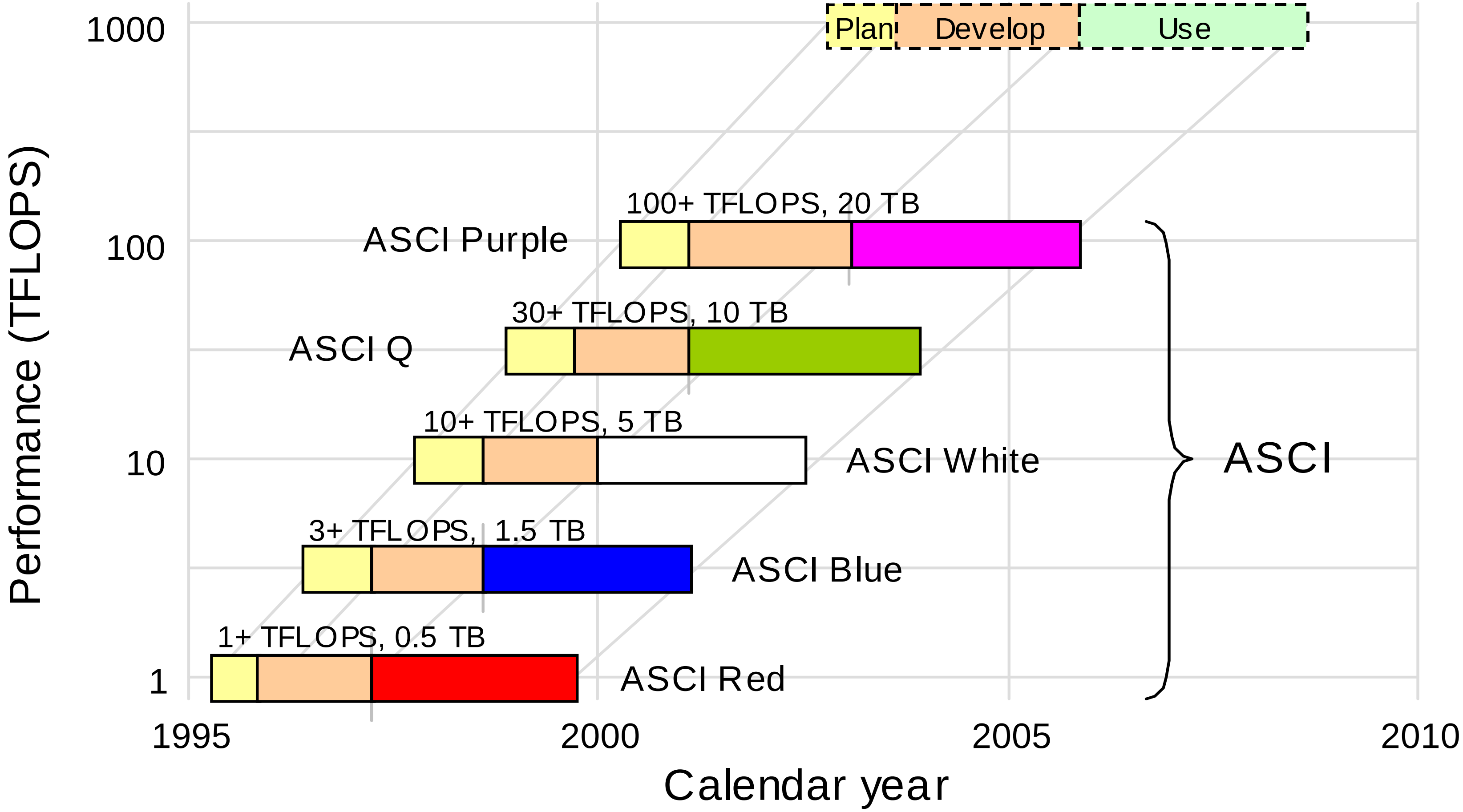


Figure 4.8 Milestones in the DOE’s Accelerated Strategic Computing Initiative (ASCI) program with extrapolation up to the PFLOPS level.

Performance is Important, But It Isn't Everything

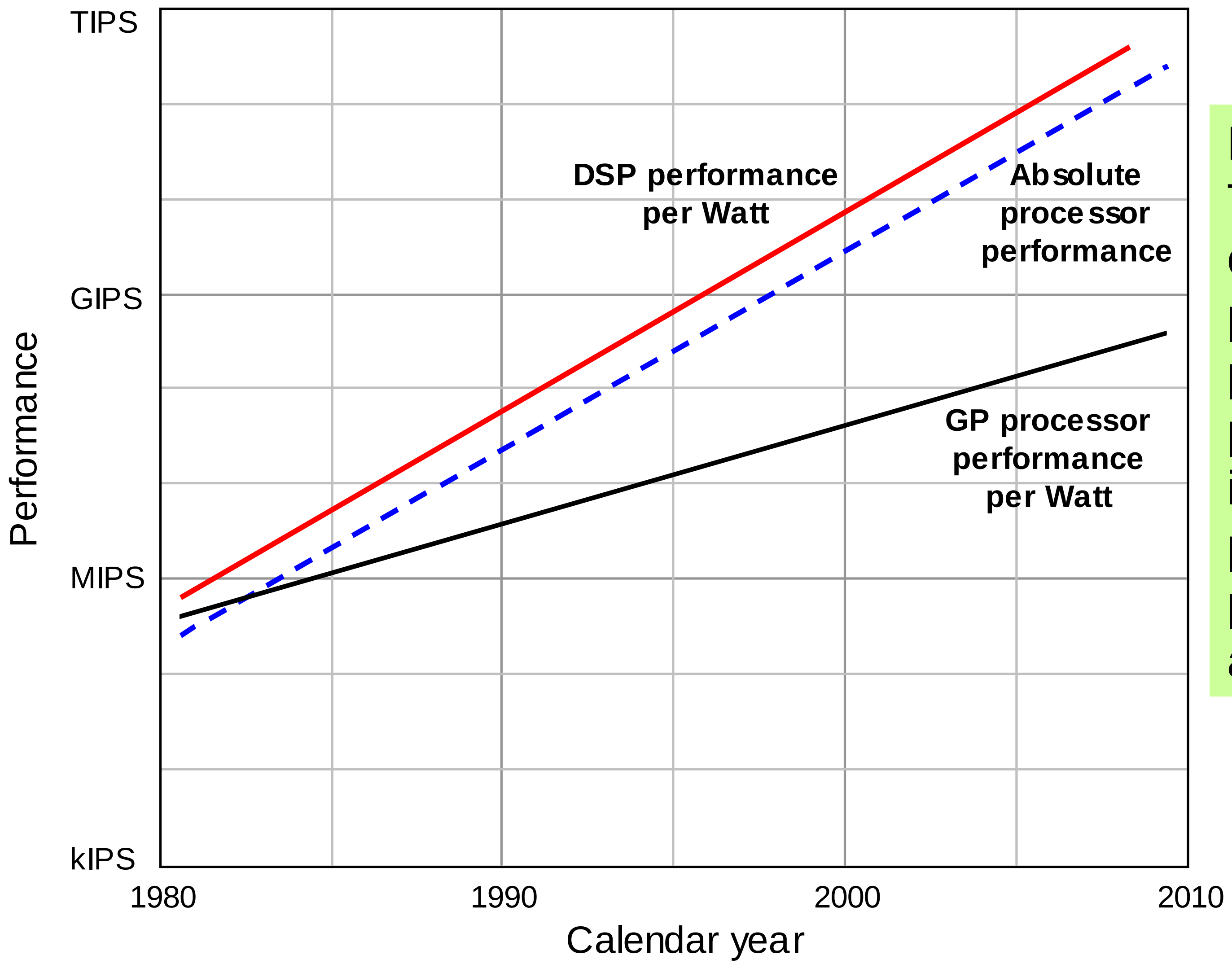


Figure 25.1 Trend in computational performance per watt of power used in general-purpose processors and DSPs.

Roadmap for the Rest of the Book

Fasten your seatbelts as we begin our ride!

Ch. 5-8: A simple ISA, variations in ISA

Ch. 9-12: ALU design

Ch. 13-14: Data path and control unit design

Ch. 15-16: Pipelining and its limits

Ch. 17-20: Memory (main, mass, cache, virtual)

Ch. 21-24: I/O, buses, interrupts, interfacing

Ch. 25-28: Vector and parallel processing