# Bölüm 9

# COMPUTER ARITHMETIC

## Algorithms and Circuit Design

UCSB

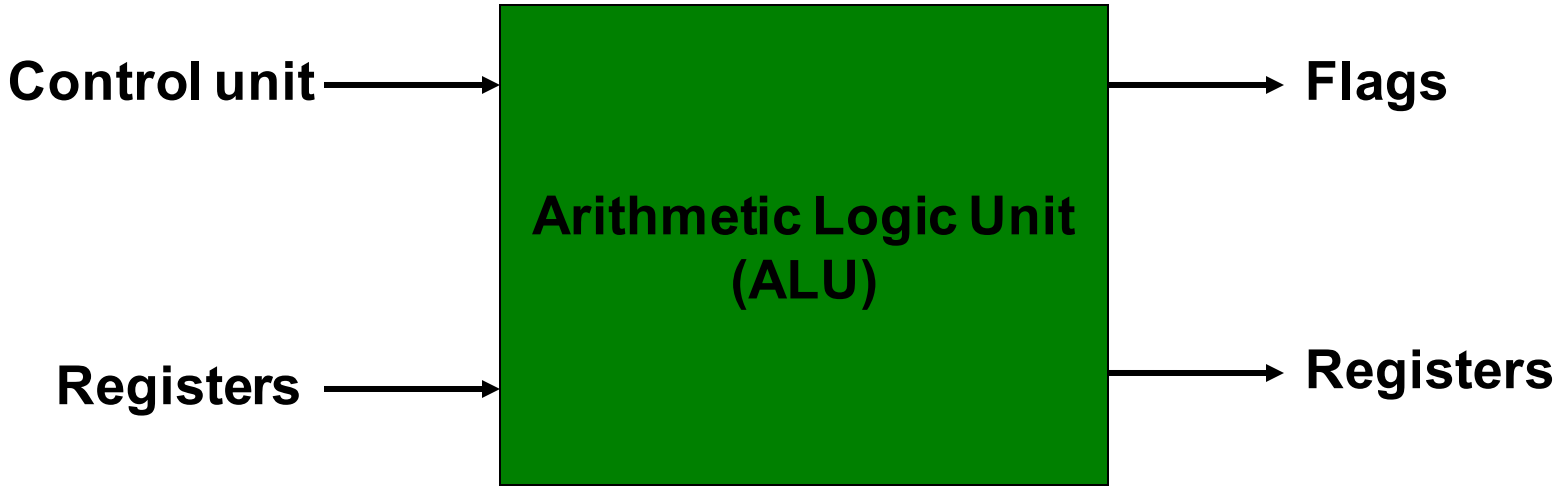
BParhami

# What Goes on Inside ALU?

– Machine instr.: `add $t1, $s1, $s2`

– What it means to computer:

`000000 10001 10010 01000 00000 100000`



# Basic Idea

- Hardware can only deal with binary digits, 0 and 1.
- Must represent all numbers, integers or floating point, positive or negative, by binary digits, called *bits*.
- Devise electronic circuits to perform arithmetic operations: add, subtract, multiply and divide, on binary numbers.

# Motivation Example 1

Pentium Division Bug (1994-95): Pentium's radix-4 SRT algorithm occasionally gave incorrect quotient
First noted in 1994 by T. Nicely who computed sums of reciprocals of twin primes:

$$1/5 + 1/7 + 1/11 + 1/13 + \ldots + 1/p + 1/(p + 2) + \ldots$$

Worst-case example of division error in Pentium:

$$c = \frac{4\ 195\ 835}{3\ 145\ 727} = \begin{cases} 1.333\ 820\ 44... \\ 1.333\ 739\ 06... \end{cases}$$

Correct quotient

circa 1994 Pentium double FLP value; accurate to only 14 bits (worse than single!)

# Motivation Example 2
## Finite Precision Can Lead to Disaster

**Example: Failure of Patriot Missile (1991 Feb. 25)**

Source http://www.math.psu.edu/dna/455.f96/disasters.html

American Patriot Missile battery in Dharan, Saudi Arabia, failed to intercept incoming Iraqi Scud missile

The Scud struck an American Army barracks, killing 28

Cause, per GAO/IMTEC-92-26 report: “software problem” (inaccurate calculation of the time since boot)

Problem specifics:

Time in tenths of second as measured by the system’s internal clock was multiplied by 1/10 to get the time in seconds

Internal registers were 24 bits wide

1/10 = 0.0001 1001 1001 1001 1001 100 (chopped to 24 b)

Error $\approx 0.1100\ 1100 \times 2^{-23} \approx 9.5 \times 10^{-8}$

Error in 100-hr operation period

$\approx 9.5 \times 10 - 8 \times 100 \times 60 \times 60 \times 10 = 0.34$ s

Distance traveled by Scud = (0.34 s) × (1676 m/s) ≈ 570 m

# Computer Arithmetic in the 1940s

Machine arithmetic was crucial in proving the feasibility of computing with stored-program electronic devices

Hardware for addition/subtraction, use of complement representation, and shift-add multiplication and division algorithms were developed.

A seminal report by A.W. Burkes, H.H. Goldstein, and J. von Neumann contained ideas on choice of number radix, carry propagation chains, fast multiplication via carry-save addition, and restoring division

State of computer arithmetic circuit 1950:
Overview paper by R.F. Shaw [Shaw50]

# Computer Arithmetic in the 1950s

The focus shifted from feasibility to algorithmic speedup methods and cost-effective hardware realizations

By the end of the decade, virtually all important fast-adder designs had already been published or were in the final phases of development

Residue arithmetic, SRT division, CORDIC algorithms were proposed and implemented

Snapshot of the field circa 1960:

Overview paper by O.L. MacSorley [MacS61]

# Computer Arithmetic in the 1960s

Tree multipliers, array multipliers, high-radix dividers, convergence division, redundant signed-digit arithmetic were introduced

Implementation of floating-point arithmetic operations in hardware or firmware (in microprogram) became prevalent

Many innovative ideas originated from the design of early supercomputers, when the demand for high performance, along with the still high cost of hardware, led designers to novel and cost-effective solutions

Examples reflecting the sate of the art near the end of this decade:

- IBM's System/360 Model 91 [Ande67]
- Control Data Corporation's CDC 6600 [Thor70]

# Computer Arithmetic in the 1970s

Advent of microprocessors and vector supercomputers

Early LSI chips were quite limited in the number of transistors or logic gates that they could accommodate

Microprogrammed control (with just a hardware adder) was a natural choice for single-chip processors which were not yet expected to offer high performance

For high end machines, pipelining methods were perfected to allow the throughput of arithmetic units to keep up with computational demand in vector supercomputers

Examples reflecting the state of the art near the end of this decade:

Cray 1 supercomputer and its successors

# Computer Arithmetic in the 1980s

Spread of VLSI triggered a reconsideration of all arithmetic designs in light of interconnection cost and pin limitations

For example, carry-lookahead adders, thought to be ill-suited to VLSI, were shown to be efficiently realizable after suitable modifications. Similar ideas were applied to more efficient VLSI tree and array multipliers

Bit-serial and on-line arithmetic were advanced to deal with severe pin limitations in VLSI packages

Arithmetic-intensive signal processing functions became driving forces for low-cost and/or high-performance embedded hardware: DSP chips

# Computer Arithmetic in the 1990s

No breakthrough design concept

Demand for performance led to fine-tuning of arithmetic algorithms and implementations (many hybrid designs)

Increasing use of table lookup and tight integration of arithmetic unit and other parts of the processor for maximum performance

Clock speeds reached and surpassed 100, 200, 300, 400, and 500 MHz in rapid succession; pipelining used to ensure smooth flow of data through the system

Examples reflecting the state of the art near the end of this decade:

- Intel's Pentium Pro (P6) $\rightarrow$ Pentium II
- Several high-end DSP chips

# Computer Arithmetic in the 2000s

Partial list, based on the first half of the decade

Continued refinement of many existing methods, particularly those based on table lookup

New challenges posed by multi-GHz clock rates

Increased emphasis on low-power design

Reexamination of the IEEE 754 floating-point standard

**Design:** Shift of attention from algorithms to optimizations at the level of transistors and wires

This explains the proliferation of hybrid designs

**Technology:** Predominantly CMOS, with a phenomenal rate of improvement in size/speed

**Applications:** Shift from high-speed or high-throughput designs in mainframes to embedded systems requiring

Low cost
Low power

# Aspects of, and Topics in, Computer Arithmetic

**Hardware (our focus in this book)**

Design of efficient digital circuits for primitive and other arithmetic operations such as +, –, ×, ÷, √, log, sin, cos

**Issues:** Algorithms
Error analysis
Speed/cost trade-offs
Hardware implementation
Testing, verification

**Software**

Numerical methods for solving systems of linear equations, partial differential equations, etc.

**Issues:** Algorithms
Error analysis
Computational complexity
Programming
Testing, verification

**General-purpose**

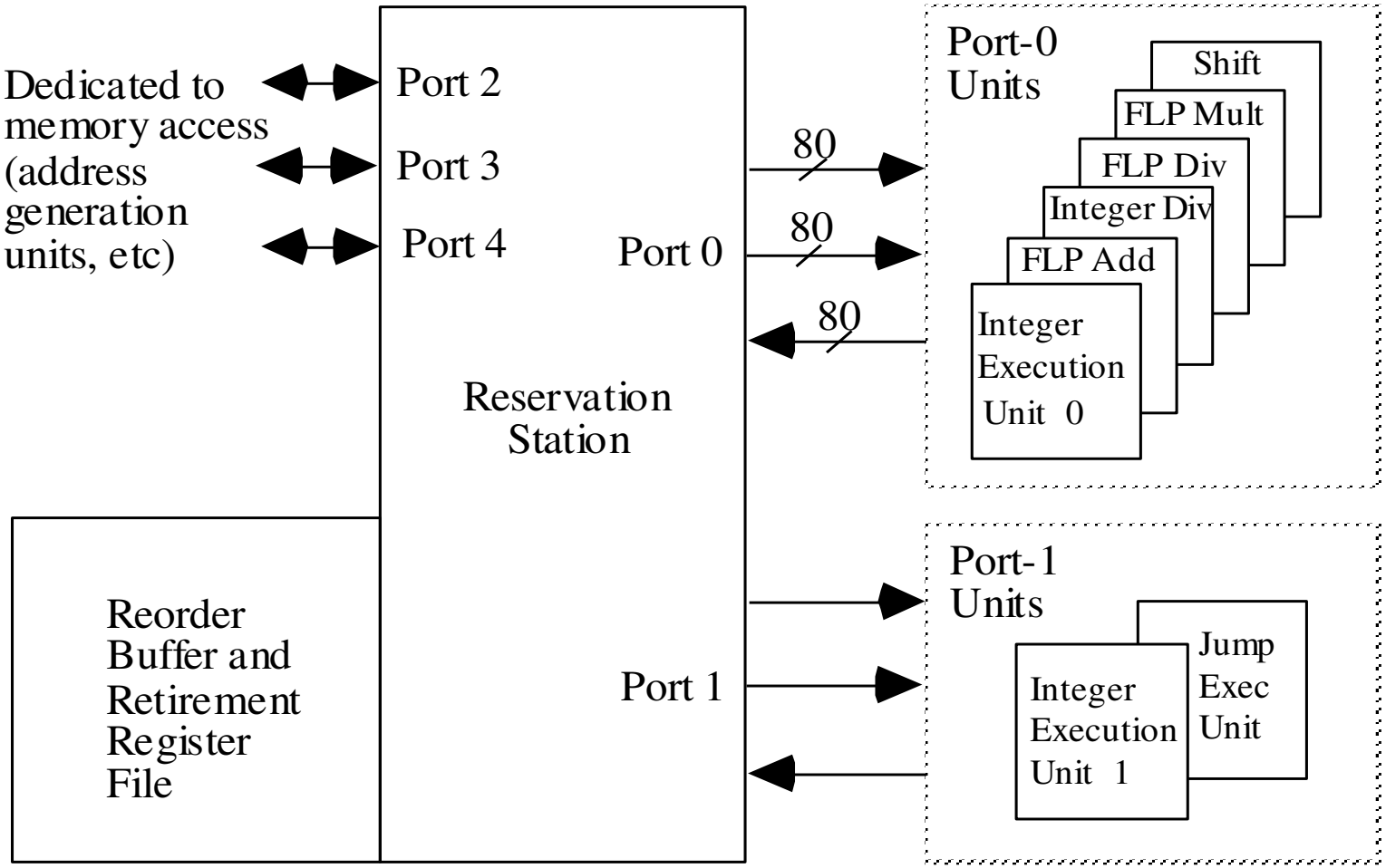
Flexible data paths
Fast primitive operations like +, –, ×, ÷, √
Benchmarking

**Special-purpose**

Tailored to applications like:
Digital filtering
Image processing
Radar tracking

The scope of computer arithmetic.

# Arithmetic in the Intel Pentium Pro Microprocessor




Key parts of the CPU in the Intel Pentium Pro (P6) microprocessor.

# Ongoing Debates and New Paradigms

Renewed interest in bit- and digit-serial arithmetic as mechanisms to reduce the VLSI area and to improve packageability and testability

Synchronous vs asynchronous design (asynchrony has some overhead, but an equivalent overhead is being paid for clock distribution and/or systolization)

New design paradigms may alter the way in which we view or design arithmetic circuits

- Neuronlike computational elements
- Optical computing (redundant representations)
- Multivalued logic (match to high-radix arithmetic)
- Configurable logic

Arithmetic complexity theory

# The Need for Low-Power Design

Portable and wearable electronic devices

Nickel-cadmium batteries: 40-50 W-hr per kg of weight

Practical battery weight < 1 kg (< 0.1 kg if wearable device)

Total power ≅ 3-5 W for a day's work between recharges

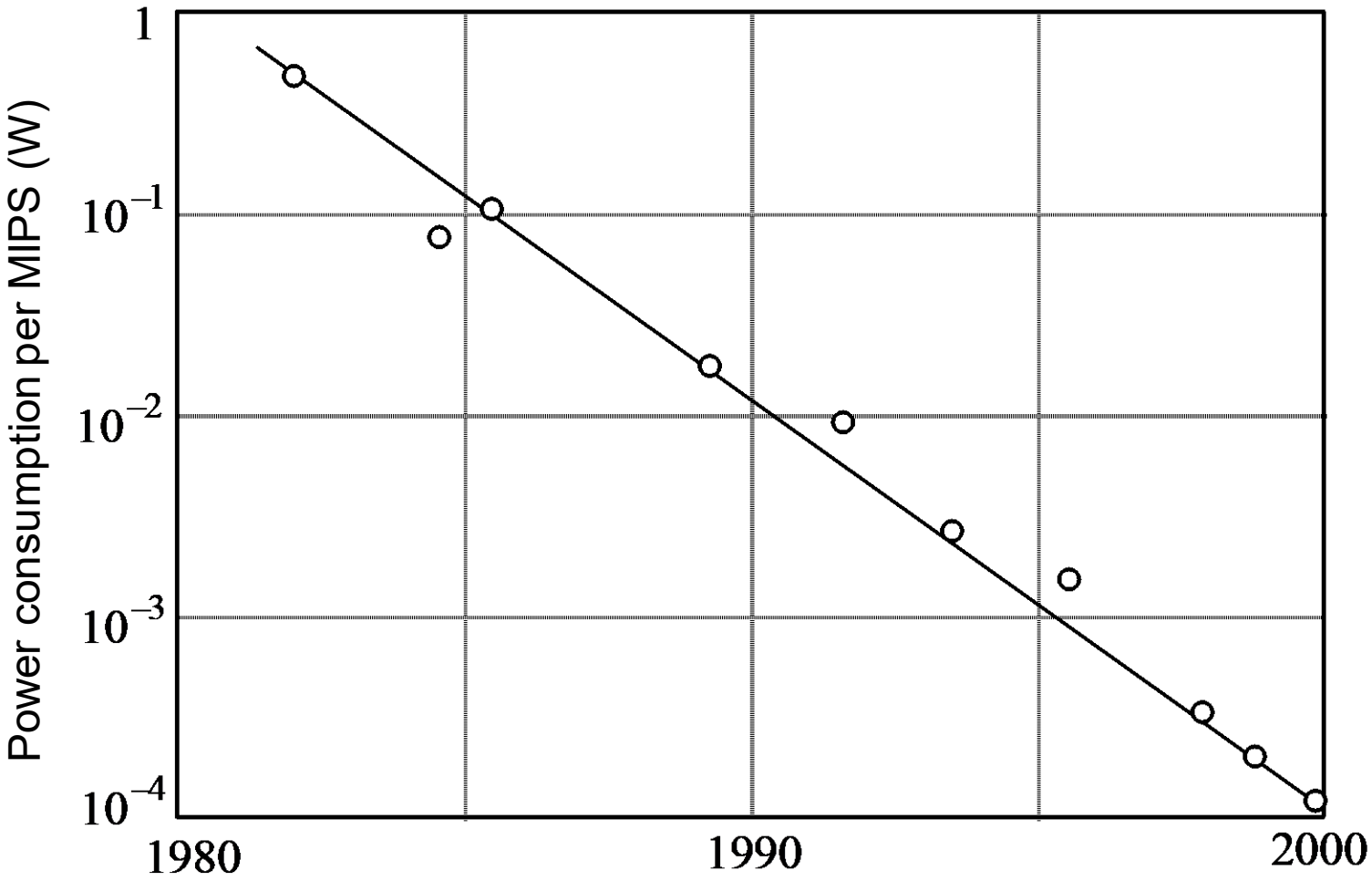
Modern high-performance microprocessors use 10s Watts

- Power is proportional to die area × clock frequency
- Cooling of micros not yet a problem; but for MPPs . . .

New battery technologies cannot keep pace with demand

Demand for more speed and functionality (multimedia, etc.)

# Processor Power Consumption Trends




Power consumption trend in DSPs [Raba98].

# Sources of Power Consumption

Both average and peak power are important

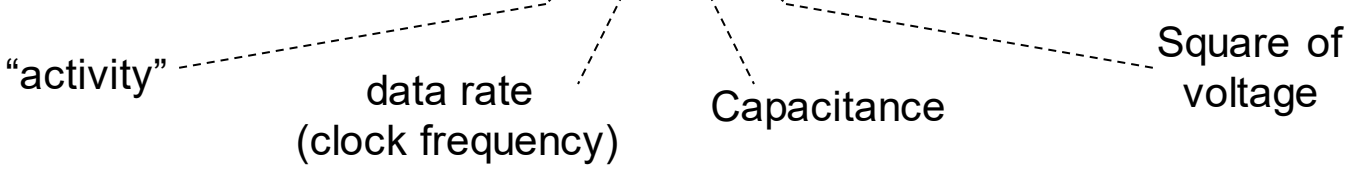
- Average power determines battery life or heat dissipation
- Peak power impacts power distribution and signal integrity

Typically, low-power design aims at reducing both

Power dissipation in CMOS digital circuits

- Static: Leakage current in imperfect switches (< 10%)
- Dynamic: Due to (dis)charging of parasitic capacitance

$$P_{avg} \cong \alpha\, f\, C\, V^2$$



# Power Reduction Strategies: The Big Picture

For a given data rate *f*, there are but 3 ways to reduce the power requirements:

1. Using a lower supply voltage *V*
2. Reducing the parasitic capacitance *C*
3. Lowering the switching activity $\alpha$

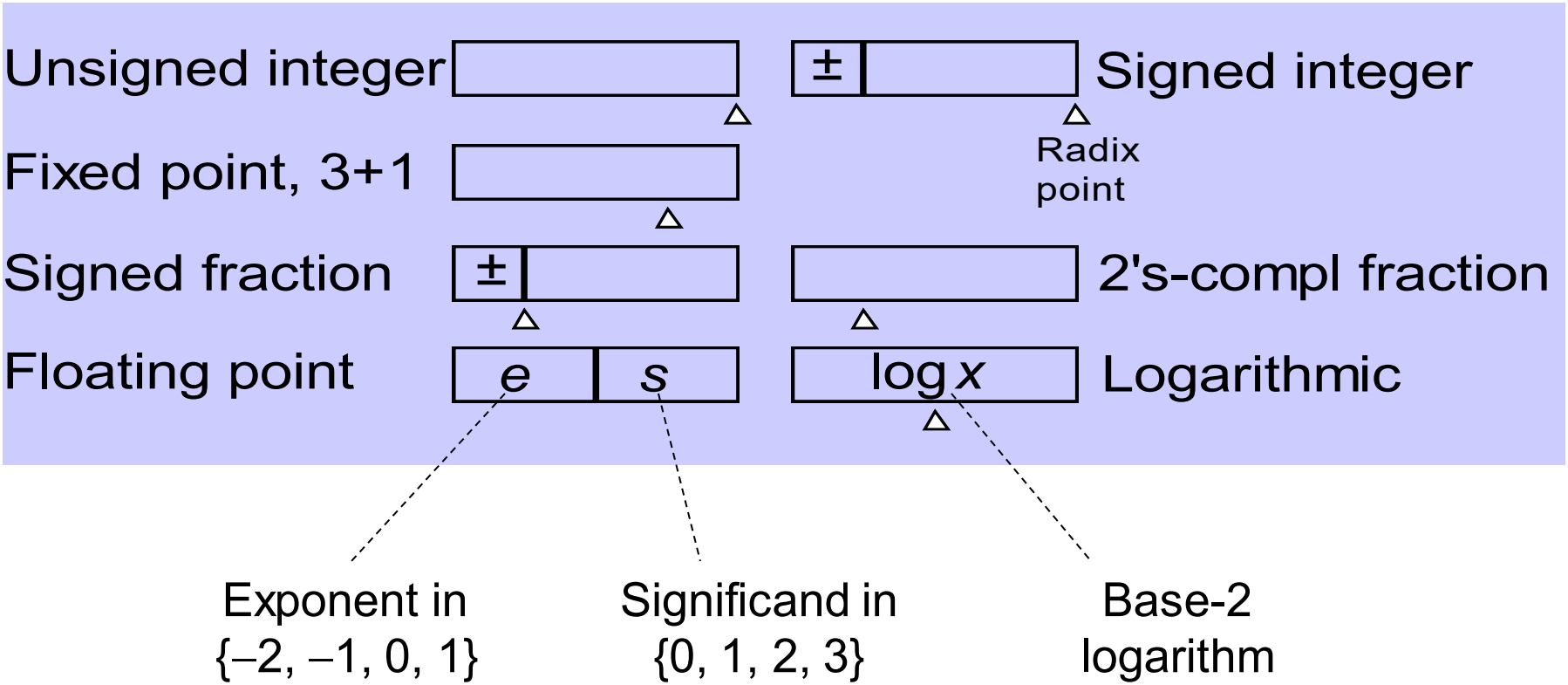
$$P_{avg} \cong \alpha\, f\, C\, V^2$$

**Example:** A 32-bit off-chip bus operates at 5 V and 100 MHz and drives a capacitance of 30 pF per bit. If random values were put on the bus in every cycle, we would have $\alpha = 0.5$. To account for data correlation and idle bus cycles, assume $\alpha = 0.2$. Then:

$$P_{avg} \cong \alpha\, f\, C\, V^2 = 0.2 \times 10^8 \times (32 \times 30 \times 10^{-12}) \times 5^2 = 0.48 \text{ W}$$

# Numbers and Their Encodings

Some 4-bit number representation formats



# Encoding Numbers in 4 Bits

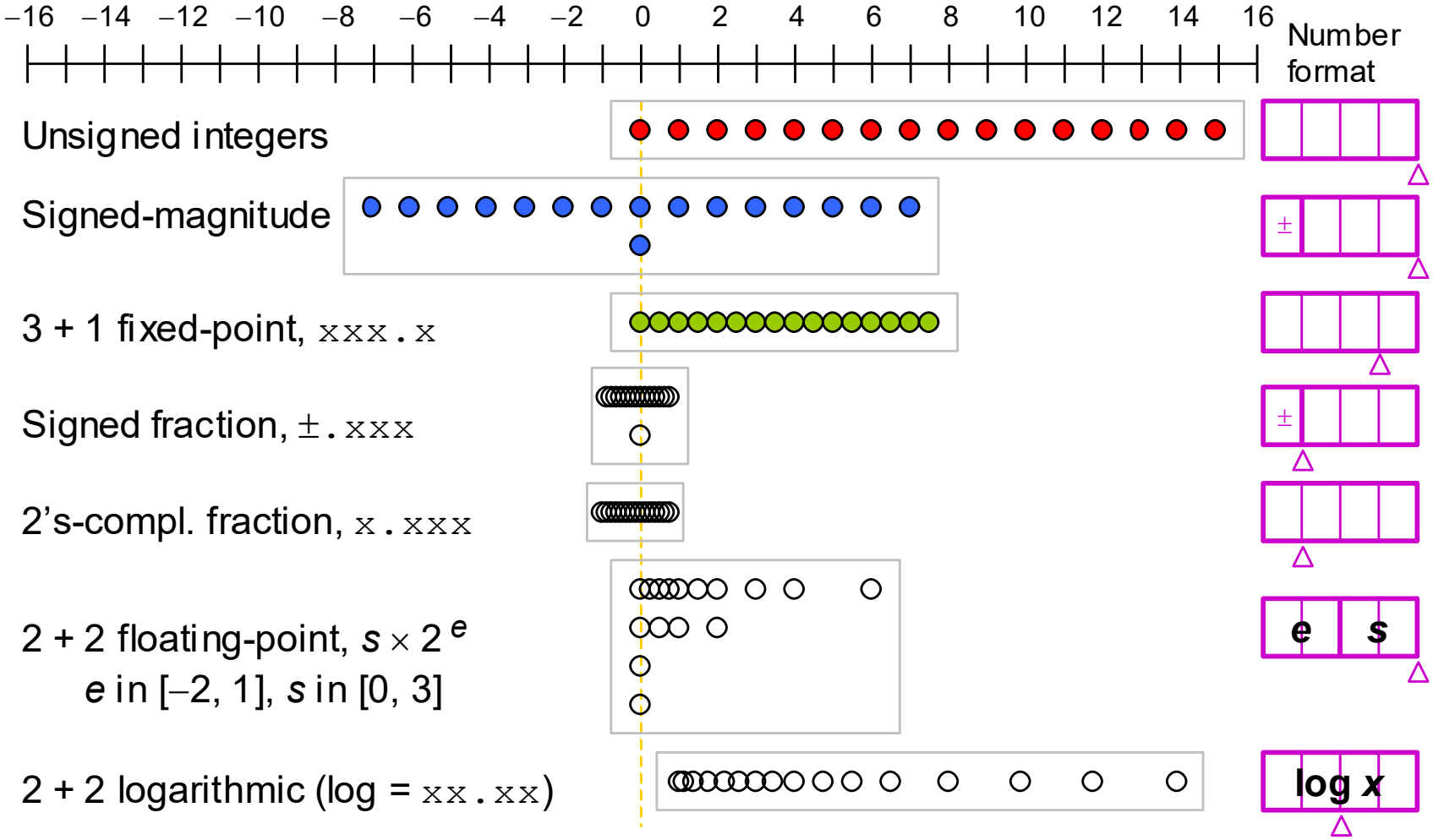



Fig. 1.2 Some of the possible ways of assigning 16 distinct codes to represent numbers.

# 1.1. Fixed-Radix Positional Number Systems

$$( x_{k-1}x_{k-2} \ldots x_1x_0 \,.\, x_{-1}x_{-2} \ldots x_{-l} )_r = \sum_{i=-l}^{k-1} x_i r^i$$

One can generalize to:

Arbitrary radix (not necessarily integer, positive, constant)

Arbitrary digit set, usually $\{-\alpha, -\alpha+1, \ldots, \beta-1, \beta\} = [-\alpha, \beta]$

**Example 1.1.** Balanced ternary number system:

Radix $r = 3$, digit set $= [-1, 1]$

**Example 1.2.** Negative-radix number systems:

Radix $-r$, $r \geq 2$, digit set $= [0, r-1]$

The special case with radix $-2$ and digit set $[0, 1]$

is known as the negabinary number system

# More Examples of Number Systems

**Example 1.3.** Digit set [–4, 5] for $r = 10$:

(3 $^{-}$1 5)ten represents 295 = 300 – 10 + 5

**Example 1.4.** Digit set [–7, 7] for $r = 10$:

(3 $^{-}$1 5)ten = (3 0 $^{-}$5)ten = (1 $^{-}$7 0 $^{-}$5)ten

**Example 1.7.** Quater-imaginary number system:

radix $r = 2j$, digit set [0, 3]

# Classes of Number Representations

Integers (fixed-point), unsigned

Integers (fixed-point), signed
- Signed-magnitude, biased, complement
- Signed-digit, including carry/borrow-save
  - ( using redundancy for faster arithmetic, not how to represent signed values)
- Residue number system:
  - ( use of parallelism for faster arithmetic, not how to represent signed values)

Real numbers, floating-point
Real arithmetic

Real numbers, exact:
- Continued-fraction, slash, . . .

# Signed-Magnitude Representation

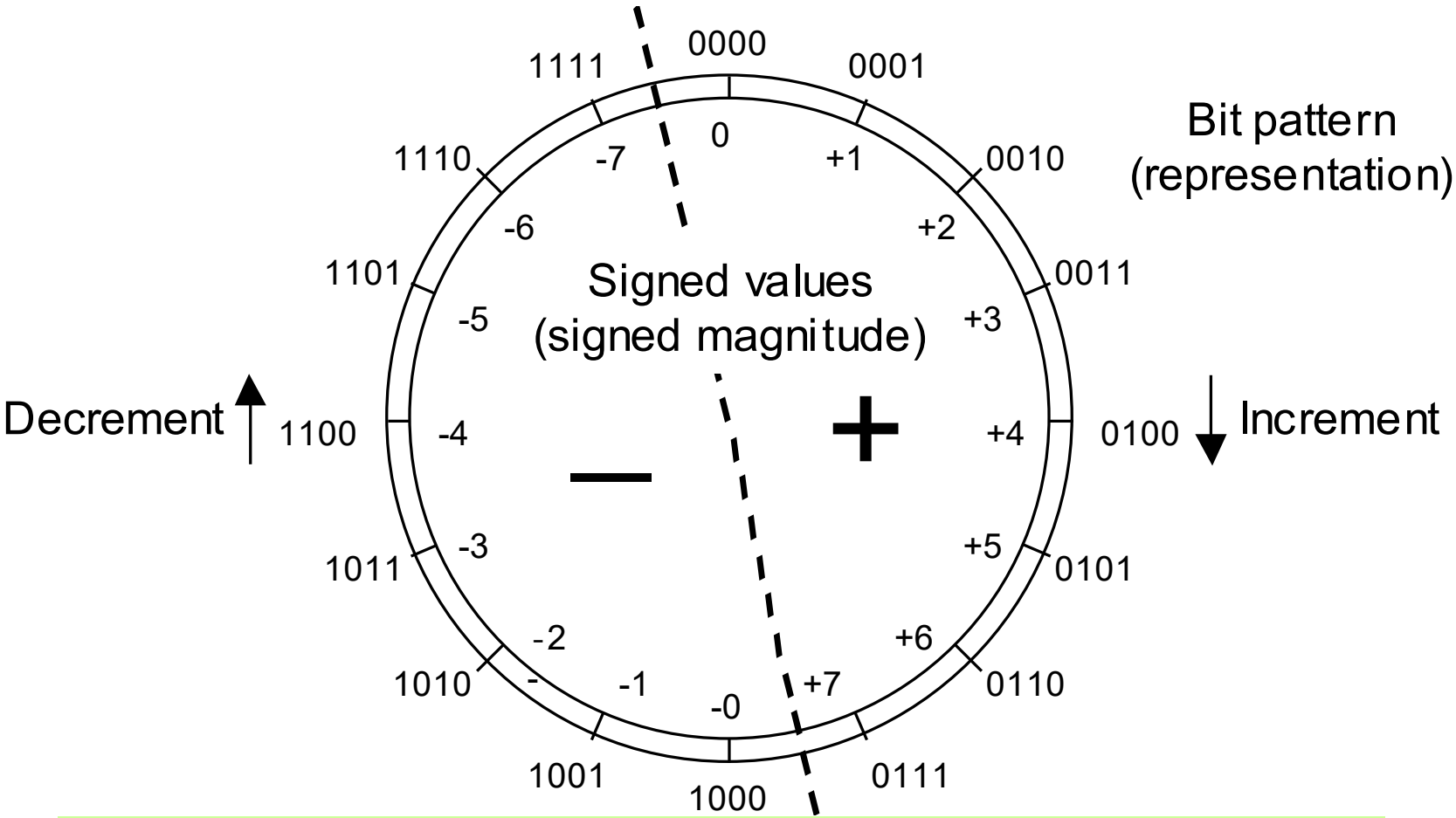



Fig. 2.1 Four-bit signed-magnitude number representation system for integers.

# Signed-Magnitude Adder

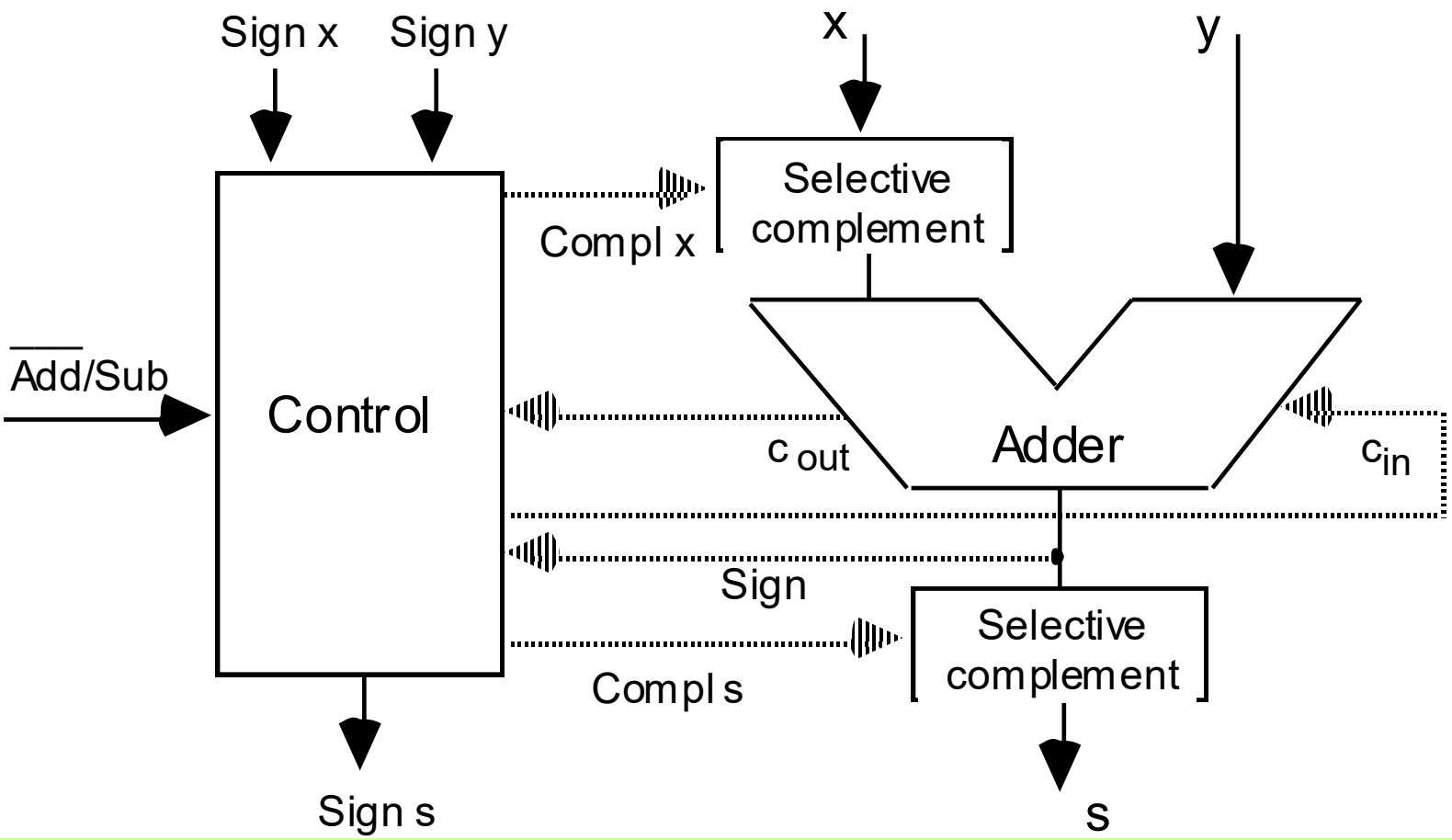



Fig. 2.2 Adding signed-magnitude numbers using precomplementation and postcomplementation.

# Biased Representations

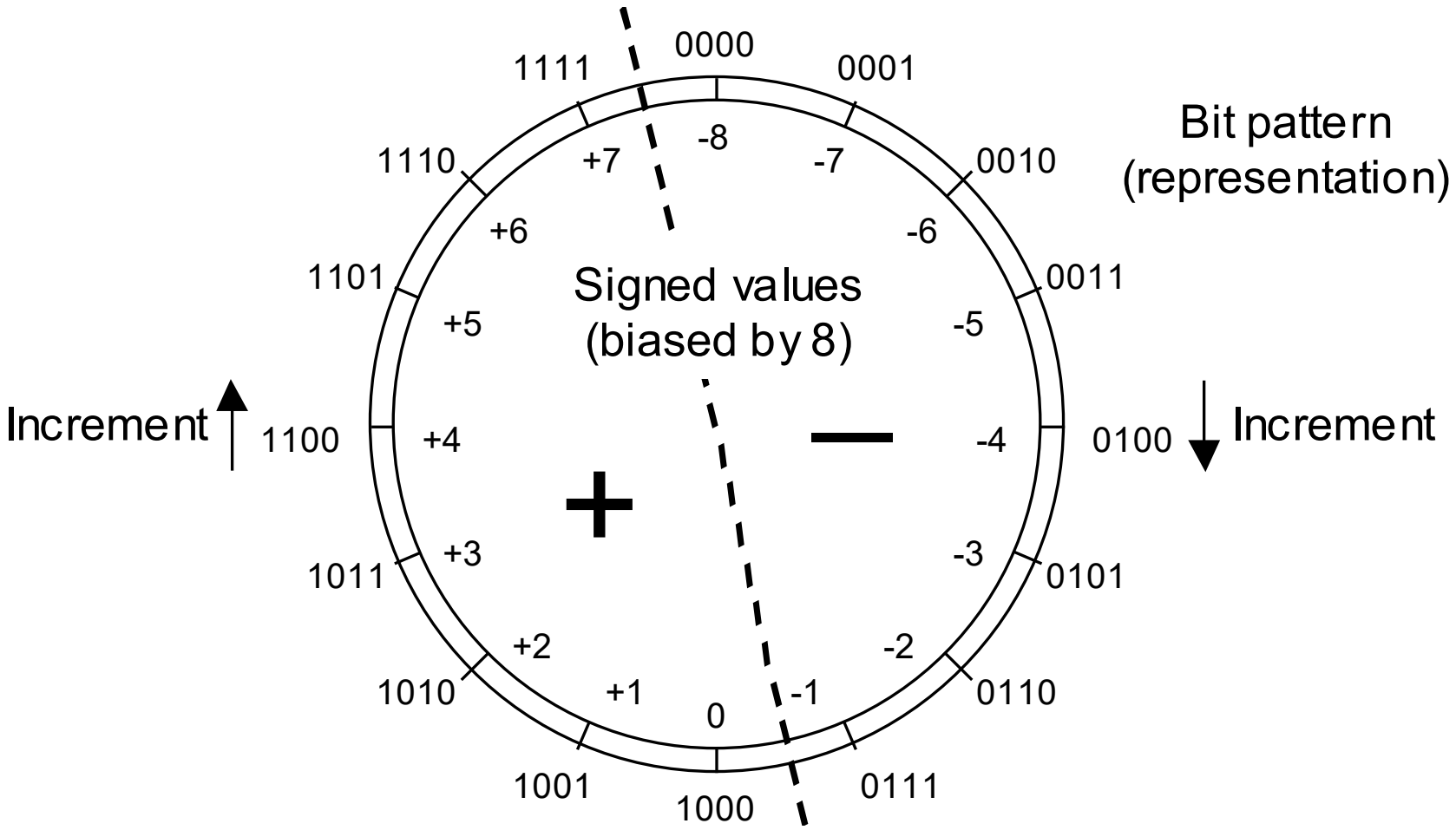



Fig. 2.3 Four-bit biased integer number representation system with a bias of 8.

# Arithmetic with Biased Numbers

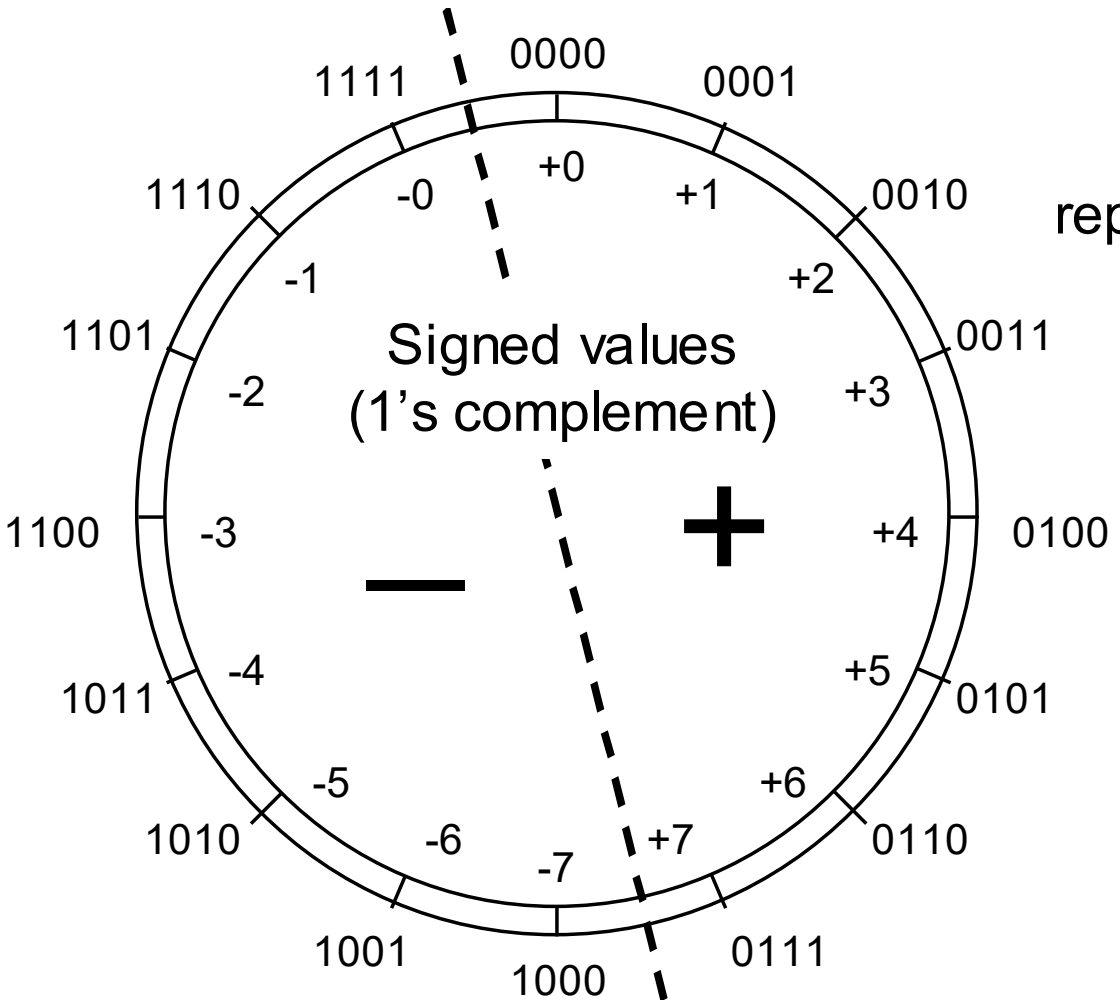
Addition/subtraction of biased numbers

$$x + y + bias = (x + bias) + (y + bias) - bias$$
$$x - y + bias = (x + bias) - (y + bias) + bias$$

A power-of-2 (or $2^a - 1$) bias simplifies addition/subtraction

We seldom perform arbitrary arithmetic on biased numbers
Main application: Exponent field of floating-point numbers

# One's-Complement Number Representation




Unsigned representations

*One's complement* = digit complement (diminished radix complement) system for $r = 2$

$M = 2^k - ulp$

$(2^k - ulp) - x = x^{compl}$

Range of representable numbers in with $k$ whole bits:

from $-2^{k-1} + ulp$ to $2^{k-1} - ulp$

Fig. 2.6 A 4-bit 1's-complement number representation system for integers.

# Three Representations

| Sign-magnitude | 1's complement | 2's complement |
|---|---|---|
| 000 = +0 | 000 = +0 | 000 = +0 |
| 001 = +1 | 001 = +1 | 001 = +1 |
| 010 = +2 | 010 = +2 | 010 = +2 |
| 011 = +3 | 011 = +3 | 011 = +3 |
| 100 = - 0 | 100 = - 3 | 100 = - 4 |
| 101 = - 1 | 101 = - 2 | 101 = - 3 |
| 110 = - 2 | 110 = - 1 | 110 = - 2 |
| 111 = - 3 | 111 = - 0 | 111 = - 1 |
| | | (Preferred) |

# Why 2's-Complement Is the Universal Choice

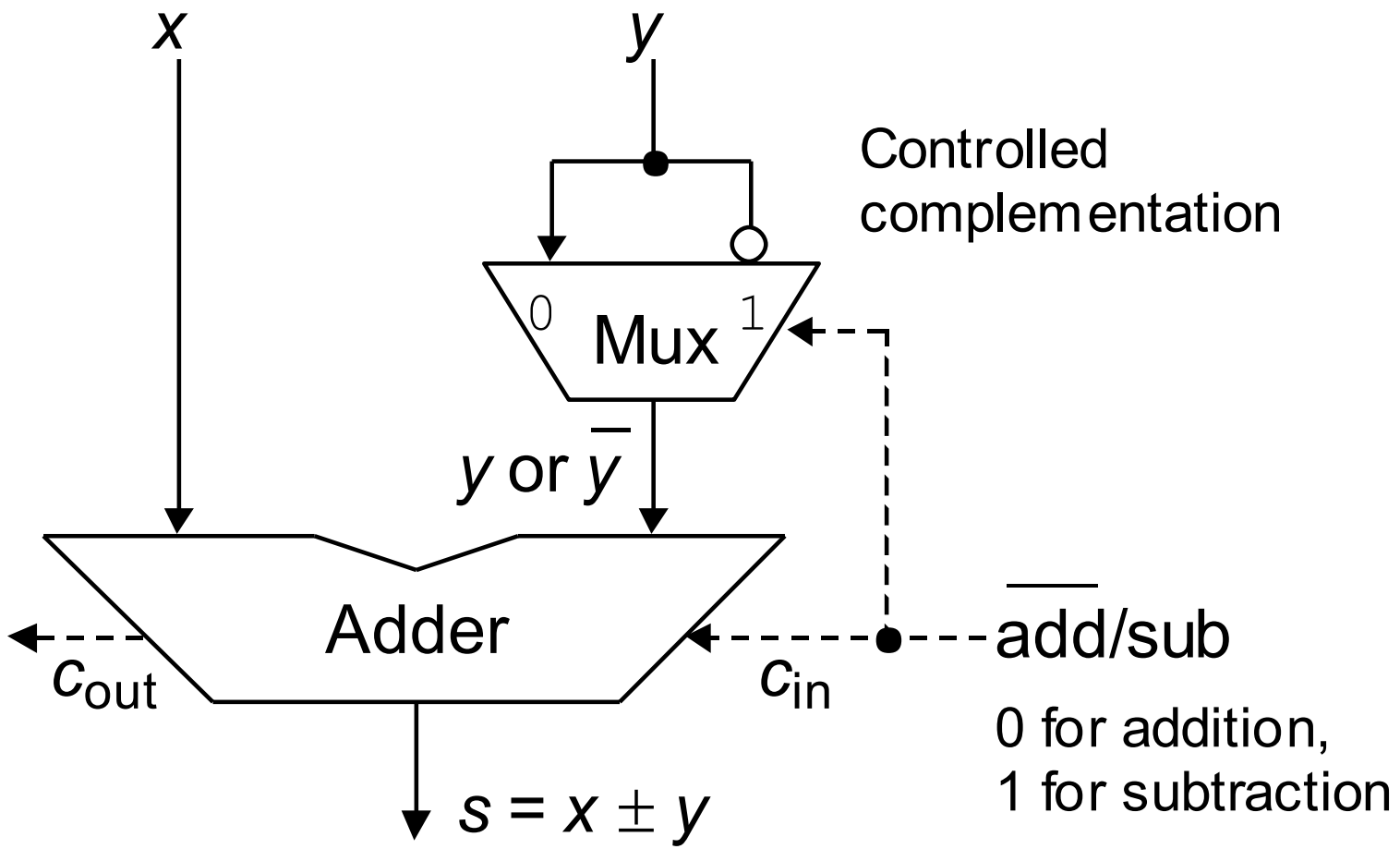


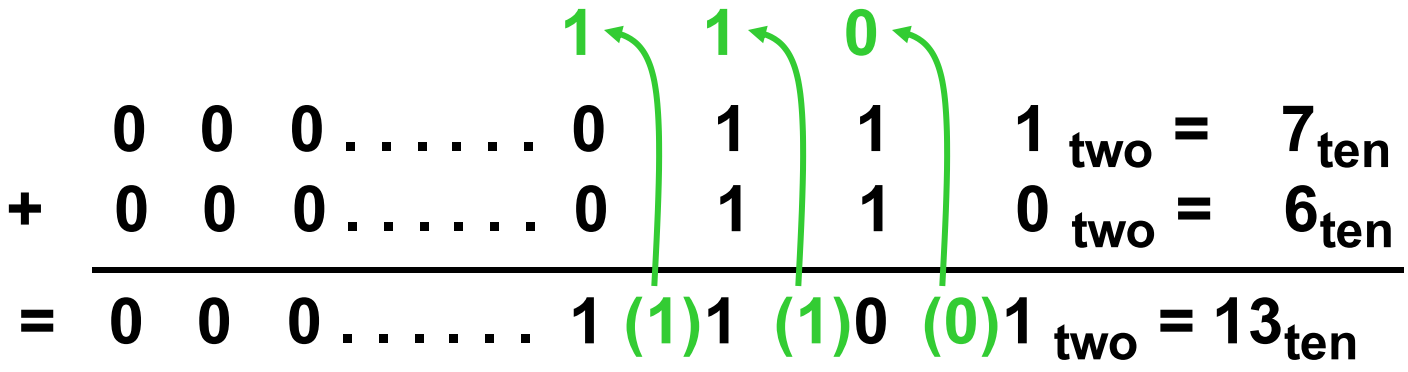

Fig. 2.7 Adder/subtractor architecture for 2's-complement numbers.

# Addition

- Adding bits:
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = (1) 0

carry

- Adding integers:



# Subtraction

- Direct subtraction

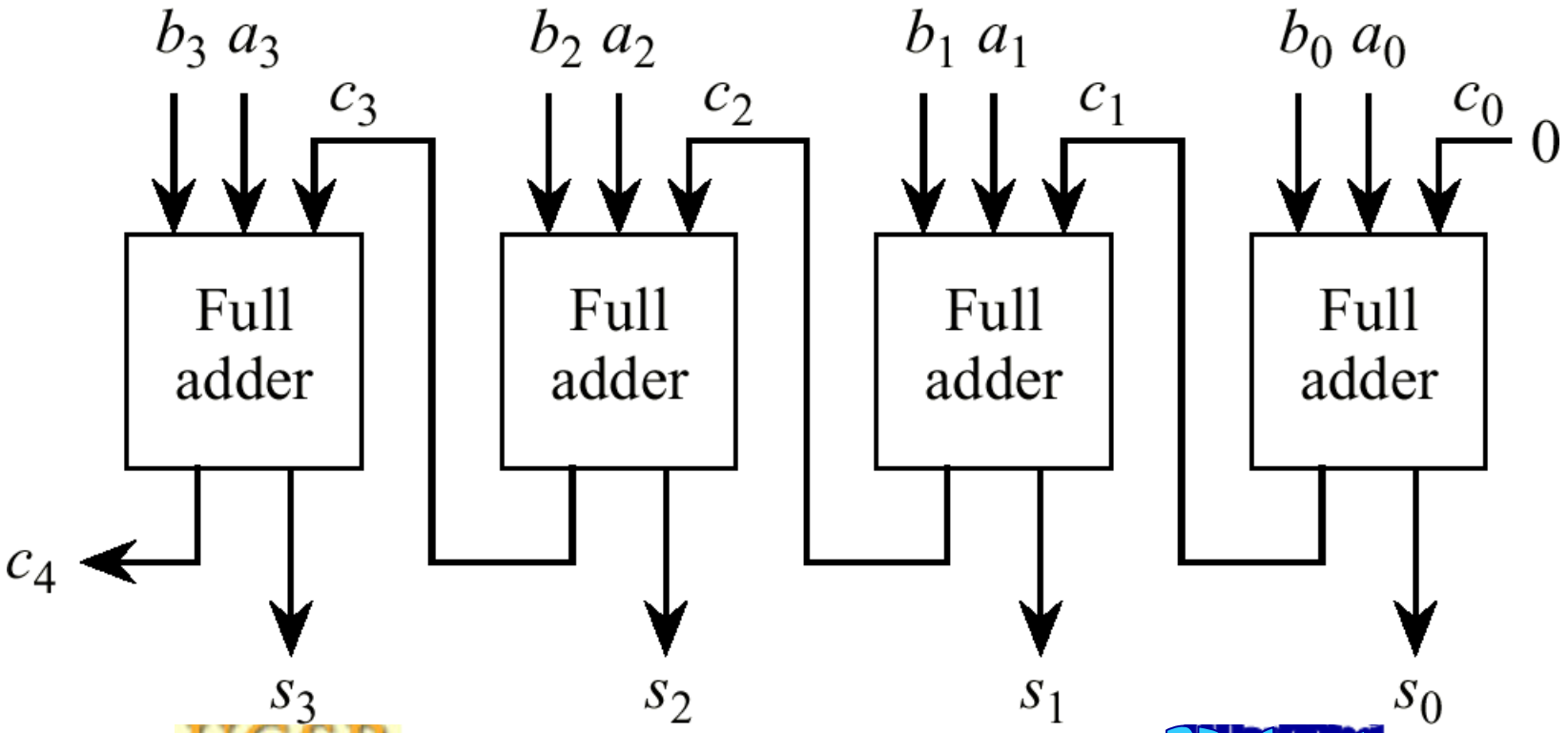
$$
\begin{array}{rl}
 & 0\ 0\ 0\ldots\ldots 0\ 1\ 1\ 1_{two} = 7_{ten} \\
- & 0\ 0\ 0\ldots\ldots 0\ 1\ 1\ 0_{two} = 6_{ten} \\
\hline
= & 0\ 0\ 0\ldots\ldots 0\ 0\ 0\ 1_{two} = 1_{ten}
\end{array}
$$

- Two's complement subtraction

$$
\begin{array}{rl}
 & 1\ 1\ 1\ldots\ldots 1\ 1\ 0 \\
 & 0\ 0\ 0\ldots\ldots 0\ 1\ 1\ 1_{two} = 7_{ten} \\
+ & 1\ 1\ 1\ldots\ldots 1\ 0\ 1\ 0_{two} = -6_{ten} \\
\hline
= & 0\ 0\ 0\ldots\ldots 0\ (1)\ 0\ (1)\ 0\ (0)\ 1_{two} = 1_{ten}
\end{array}
$$

# Ripple Carry Adder

- Two binary numbers *A* and *B* are added from right to left, creating a sum and a carry at the outputs of each full adder for each bit position.



# Simple Carry-Skip Adders

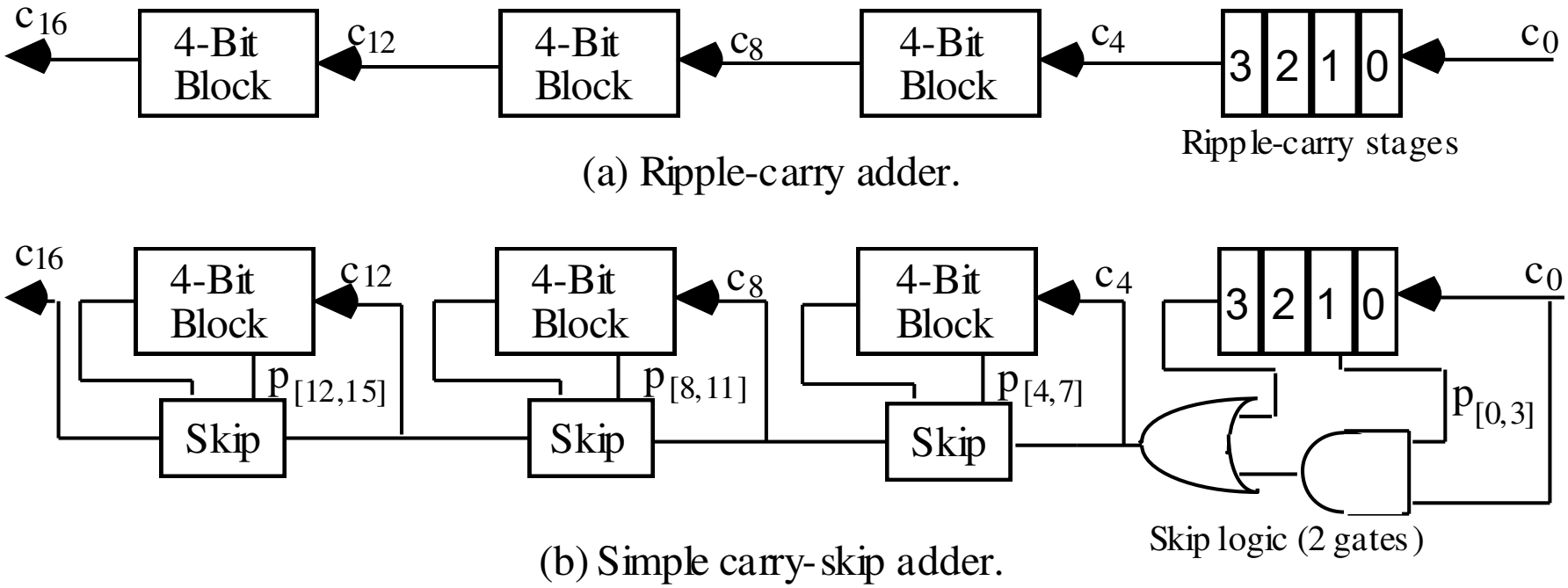
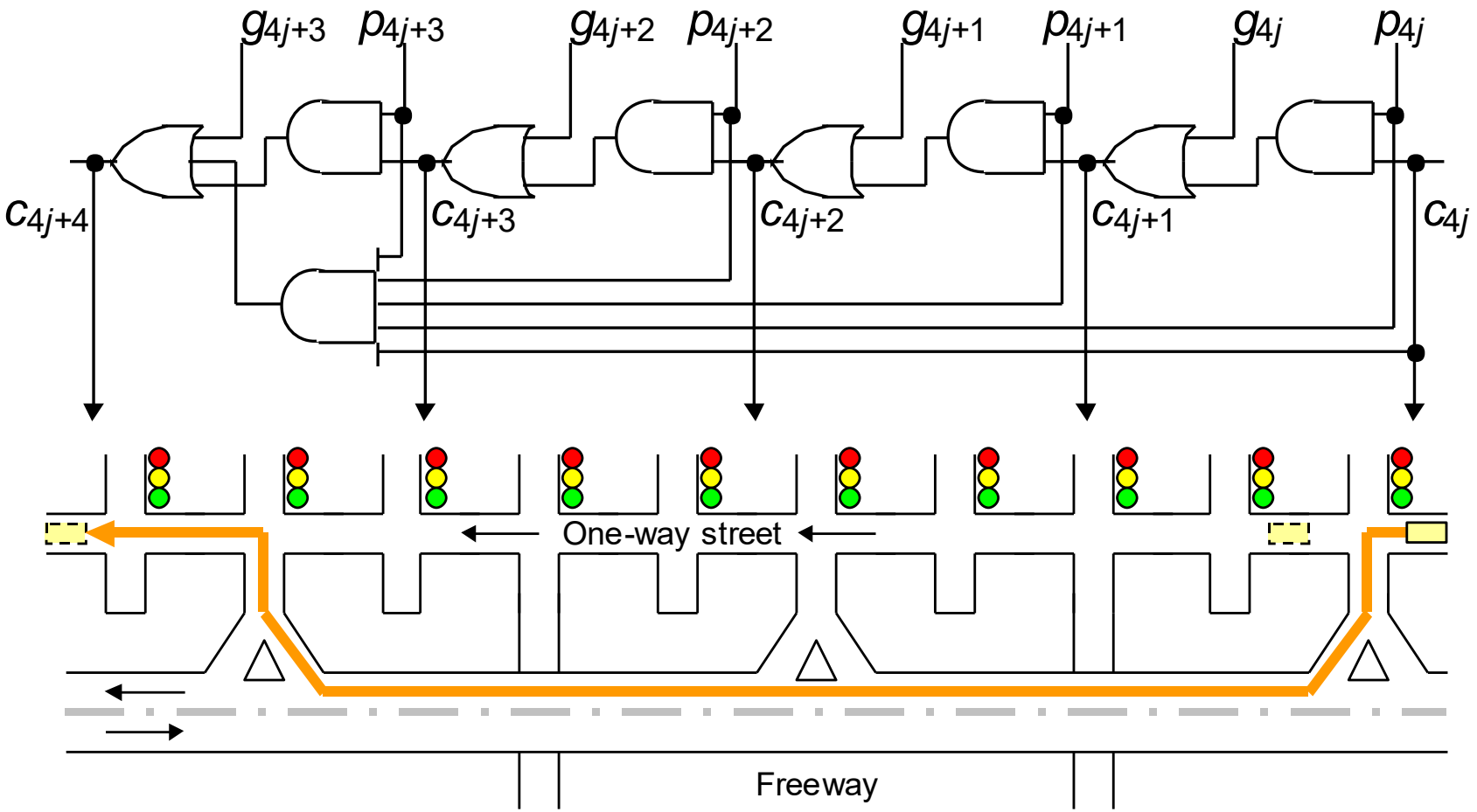



Fig. 7.1 Converting a 16-bit ripple-carry adder into a simple carry-skip adder with 4-bit skip blocks.

# Another View of Carry-Skip Addition




Street/freeway analogy for carry-skip adder.

# Carry-Skip Adder with Fixed Block Size

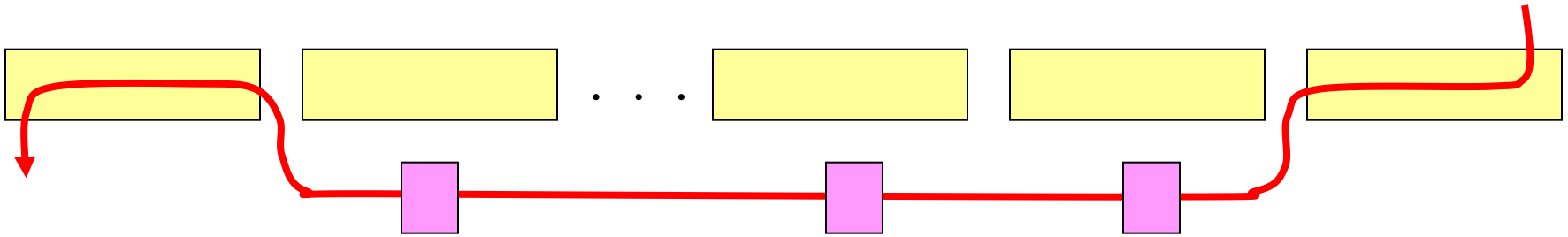
Block width $b$; $k/b$ blocks to form a $k$-bit adder (assume $b$ divides $k$)

$$T_{\text{fixed-skip-add}} = \underbrace{(b-1)}_{\text{in block 0}} + \underbrace{0.5}_{\text{OR gate}} + \underbrace{(k/b-2)}_{\text{skips}} + \underbrace{(b-1)}_{\text{in last block}}$$

$$\cong 2b + k/b - 3.5 \text{ stages}$$

$$dT/db = 2 - k/b^2 = 0 \quad \Rightarrow \quad b^{\text{opt}} = \sqrt{k/2}$$

$$T^{\text{opt}} = 2\sqrt{2k} - 3.5$$



Example: $k = 32$, $b^{\text{opt}} = 4$, $T^{\text{opt}} = 12.5$ stages
(contrast with 32 stages for a ripple-carry adder)

# Carry-Skip Adder with Variable-Width Blocks

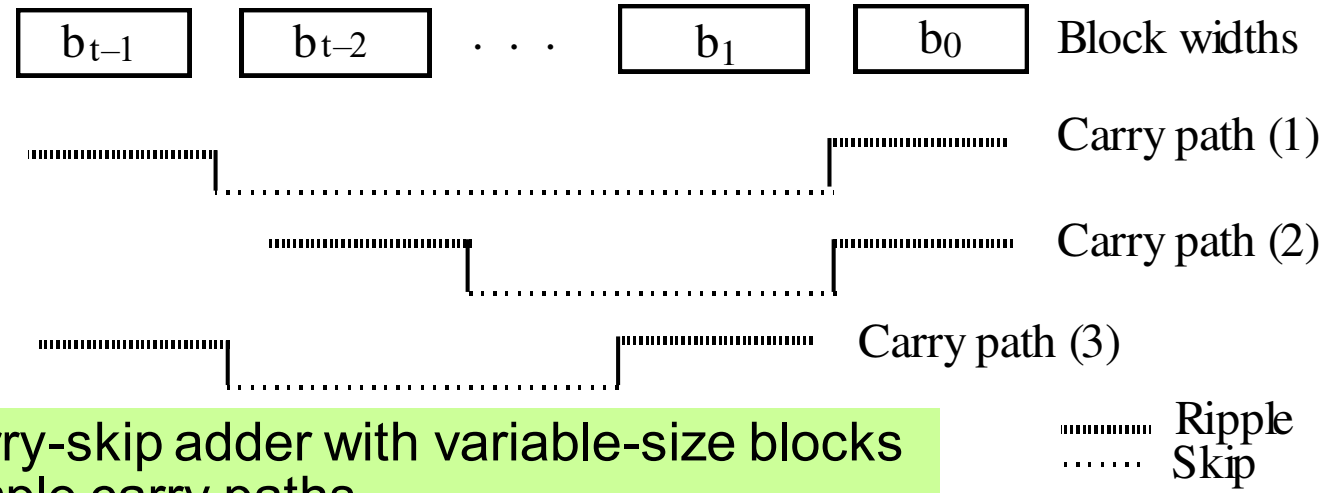



Fig. 7.2 Carry-skip adder with variable-size blocks and three sample carry paths.

The total number of bits in the $t$ blocks is $k$:

$$2[b + (b + 1) + \ldots + (b + t/2 - 1)] = t(b + t/4 - 1/2) = k$$

$$b = k/t - t/4 + 1/2$$

$$T_{\text{var-skip-add}} = 2(b - 1) + 0.5 + t - 2 = 2k/t + t/2 - 2.5$$

$$dT/db = -2k/t^2 + 1/2 = 0 \quad \Rightarrow \quad t^{\text{opt}} = 2\sqrt{k}$$

$$T^{\text{opt}} = 2\sqrt{k} - 2.5 \quad \text{(a factor of } \sqrt{2} \text{ smaller than for fixed-block)}$$

# Carry-Select Adders

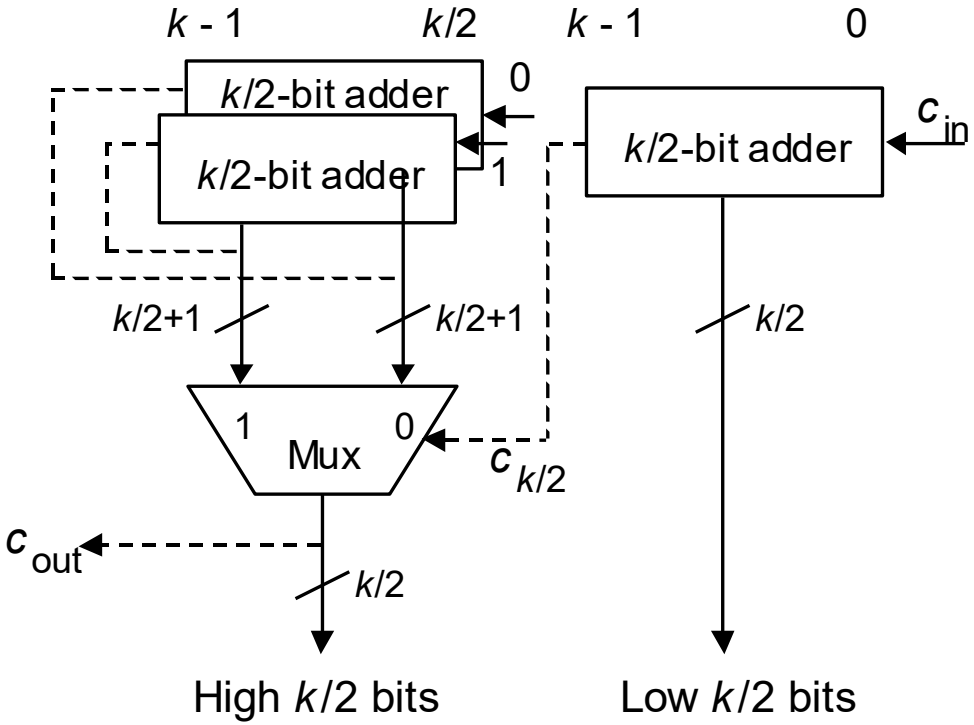



Fig. 7.9 Carry-select adder for $k$-bit numbers built from three $k/2$-bit adders.

$$C_{\text{select-add}}(k) = 3C_{\text{add}}(k/2) + k/2 + 1$$

$$T_{\text{select-add}}(k) = T_{\text{add}}(k/2) + 1$$

# Multilevel Carry-Select Adders

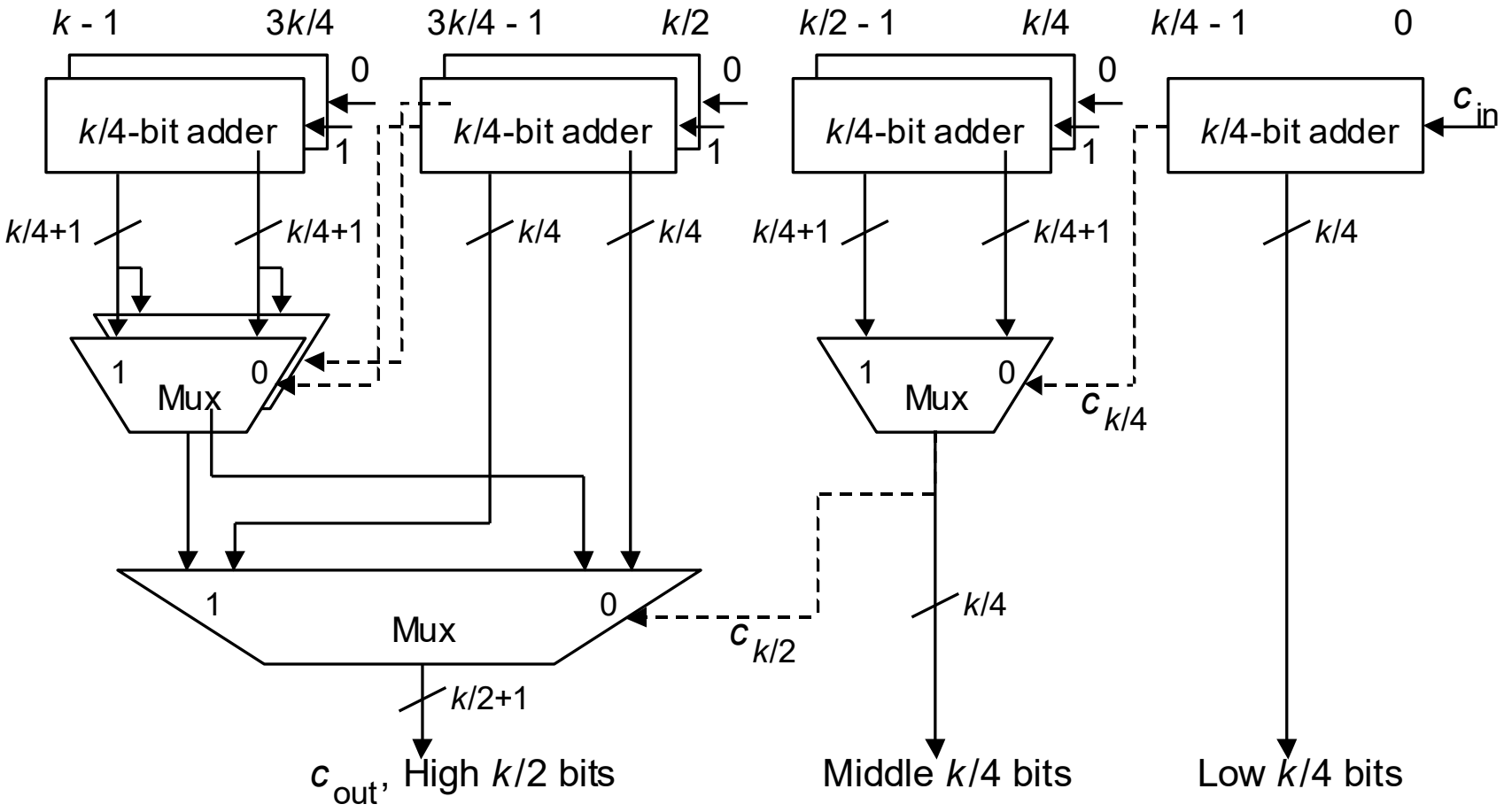



Fig. 7.10 Two-level carry-select adder built of $k/4$-bit adders.

# Carry-Lookahead Addition

$$s_i = \bar{a}_i \bar{b}_i c_i + \bar{a}_i b_i \bar{c}_i + a_i \bar{b}_i \bar{c}_i + a_i b_i c_i$$

$$c_{i+1} = b_i c_i + a_i c_i + a_i b_i$$

$$c_{i+1} = a_i b_i + (a_i + b_i) c_i$$

$$c_{i+1} = G_i + P_i c_i$$

$G_i = a_i b_i$ and $P_i = a_i + b_i$

$c_0 = 0$

$c_1 = G_0$

$c_2 = G_1 + P_1 G_0$

$c_3 = G_2 + P_2 G_1 + P_2 P_1 G_0$

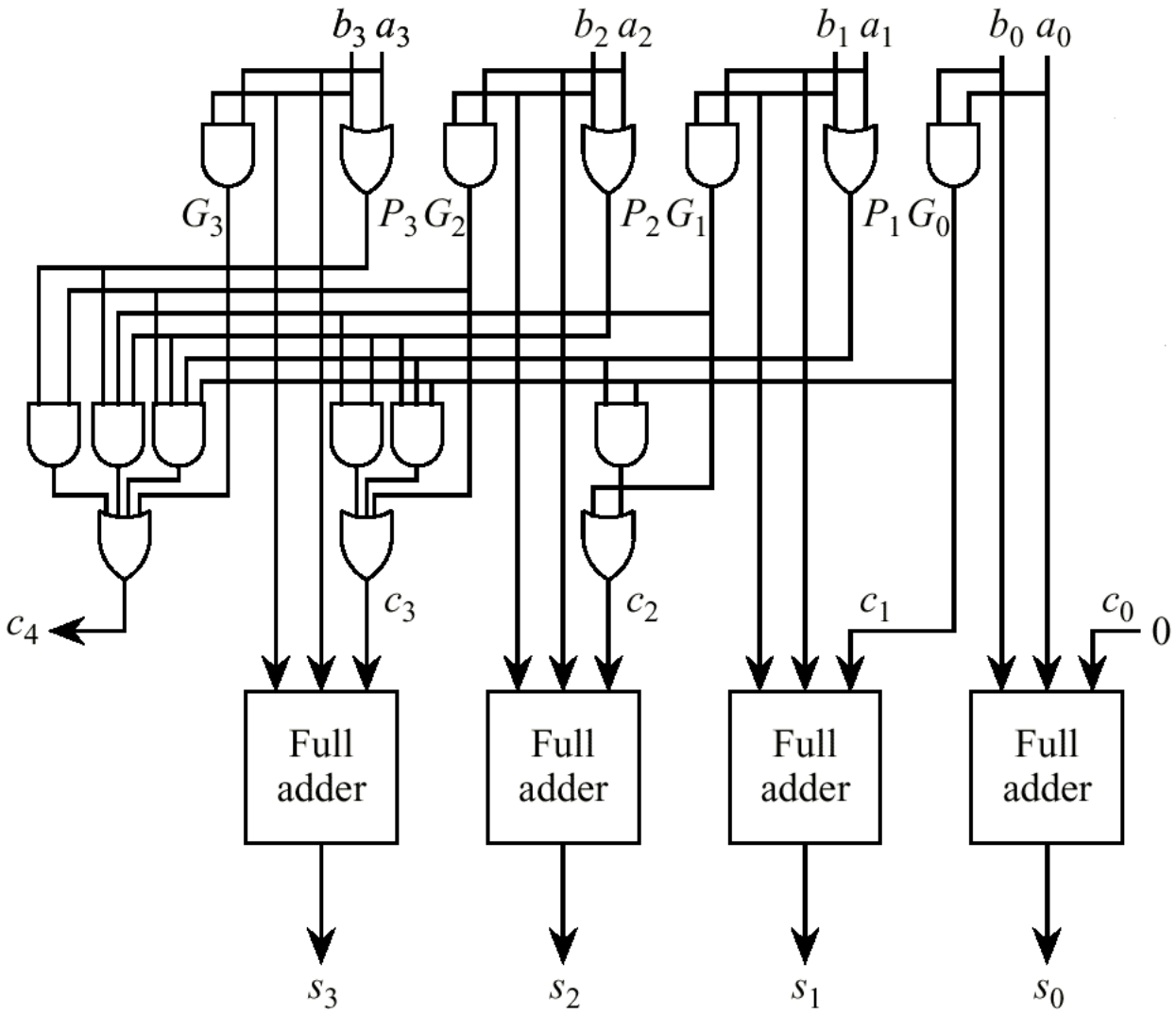
$c_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$

- Carries are represented in terms of $G_i$ (generate) and $P_i$ (propagate) expressions.

UCSB

BParhami

# Carry Lookahead Adder



# Hybrid Adder Designs

The most popular hybrid addition scheme:

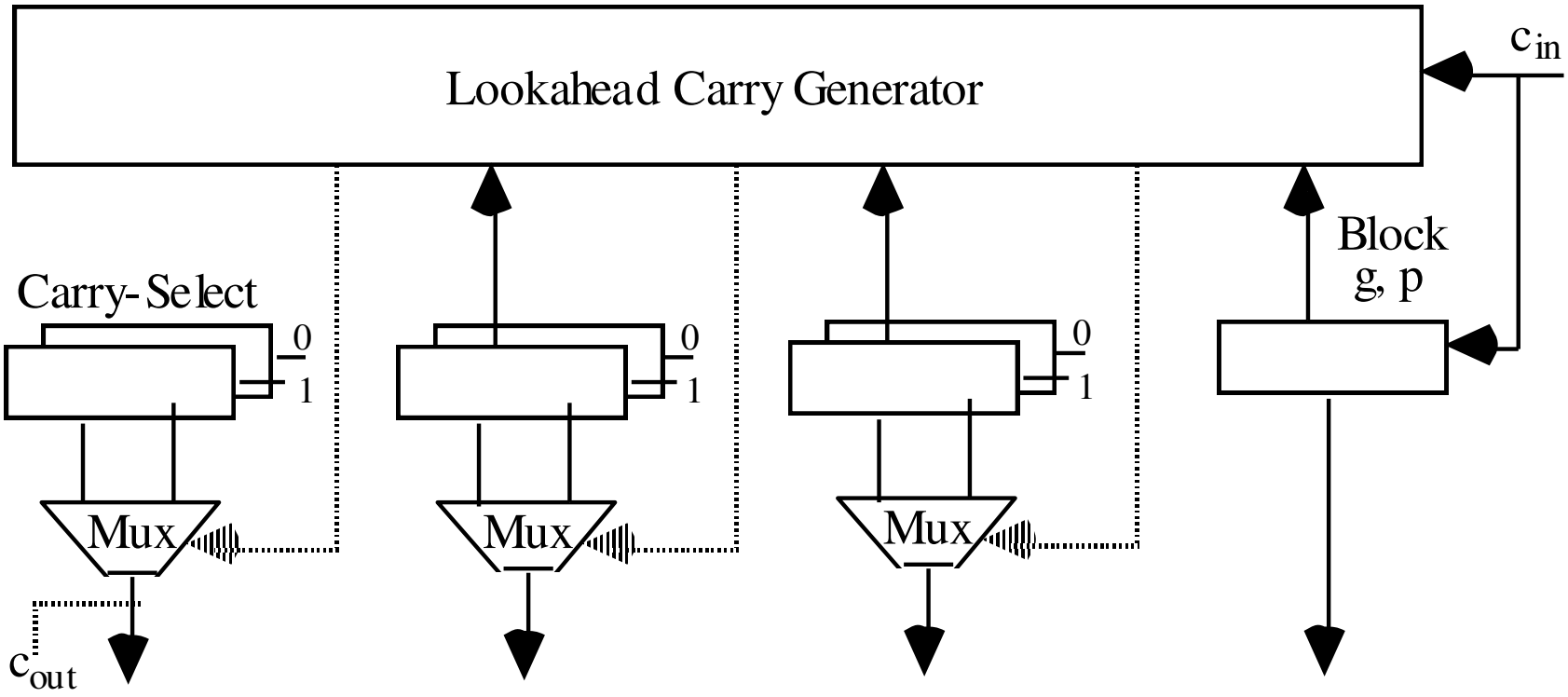



Fig. 7.12 A hybrid carry-lookahead/carry-select adder.

# Any Two Addition Schemes Can Be Combined

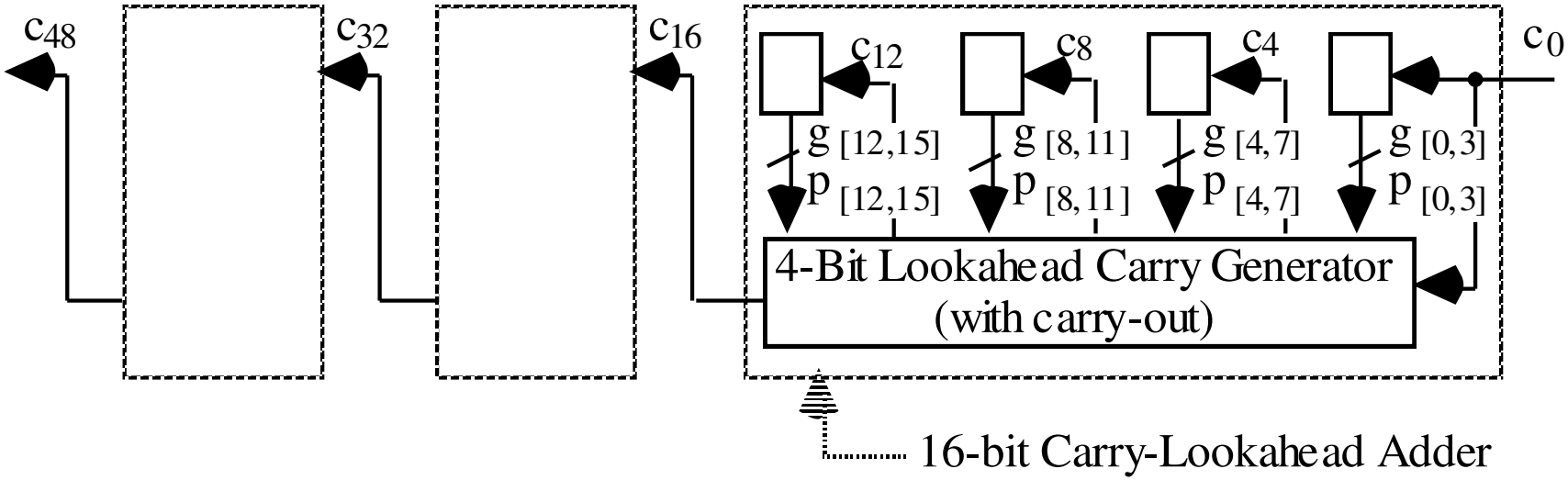



Fig. 7.13 Example 48-bit adder with hybrid ripple-carry/carry-lookahead design.

Other possibilities: hybrid carry-select/ripple-carry
hybrid ripple-carry/carry-select
. . .

# N-bit Adder Design Options

| Type of adder | Time complexity (delay) | Space complexity (size) |
|---|---|---|
| Ripple-carry | O(N) | O(N) |
| Carry-lookahead | $O(\log_2 N)$ | $O(N \log_2 N)$ |
| Carry-skip | $O(\sqrt{N})$ | O(N) |
| Carry-select | $O(\sqrt{N})$ | O(N) |

Reference: J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, Second Edition*, San Francisco, California, 1990, page A-46.

# Sequential (Ardışık) Multiplication

Çarpan $X=x_{n-1}x_{n-2}...x_1x_0$

Çarpılan $A=a_{n-1}a_{n-2}...a_1a_0$.

Burada $x_{n-1}$ ve $a_{n-1}$ işaret bitleridir.

Ardışık çarpma işlemi n-1 adımdan oluşur.

Bu işlemde, herhangi bir $x_j$ bitine bakılır ve $x_jA$ önceki kısmi çarpımla $(P^{(j)})$ toplanır.

$$P^{(j+1)}=( P^{(j)}+x_j.A).2^{-1}$$

# Example

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | | 1 | 0 | 1 | 1 | | | | | $-5$ |
| $X$ | $\times$ | 0 | 0 | 1 | 1 | | | | | 3 |
| $P^{(0)} = 0$ | | 0 | 0 | 0 | 0 | | | | | |
| $x_0 = 1 \Rightarrow$ Add $A$ | $+$ | 1 | 0 | 1 | 1 | | | | | |
| | | 1 | 0 | 1 | 1 | | | | | |
| Shift | | 1 | 1 | 0 | 1 | 1 | | | | |
| $x_1 = 1 \Rightarrow$ Add $A$ | $+$ | 1 | 0 | 1 | 1 | | | | | |
| | | 1 | 0 | 0 | 0 | 1 | | | | |
| Shift | | 1 | 1 | 0 | 0 | 0 | 1 | | | |
| $x_2 = 0 \Rightarrow$ Shift only | | 1 | 1 | 1 | 0 | 0 | 0 | 1 | | $-15$ |

# Speeding Up Multiplication

- Multiplication involves 2 basic operations - generation of partial products + their accumulation
- 2 ways to speed up - reducing number of partial products and/or accelerating accumulation
- 3 types of high-speed multipliers:
- **Sequential multiplier** - generates partial products sequentially and adds each newly generated product to previously accumulated partial product
- **Parallel multiplier** - generates partial products in parallel, accumulates using a fast multi-operand adder
- **Array multiplier** - array of identical cells generating new partial products; accumulating them simultaneously
  - No separate circuits for generation and accumulation
  - Reduced execution time but increased hardware complexity

# Reducing Number of Partial Products

- Examining 2 or more bits of multiplier at a time
- Requires generating A (multiplicand), 2A, 3A
- Reduces number of partial products to n/2 - each step more complex
- Several algorithm which do not increase complexity proposed - one is Booth's algorithm
- Fewer partial products generated for groups of consecutive 0's and 1's

# Fast Multiplication: Booth's Algorithm

- Group of consecutive 0's in multiplier - no new partial product - only shift partial product right one bit position for every 0
- Group of m consecutive 1's in multiplier - less than m partial products generated
- ...01…110... = ...10...000... - ...00...010...
- Using SD (signed-digit) notation =...100...010...
- Example:
- ...011110... = ...100000... - ...000010... = ...100010... (decimal notation: 15=16-1)
- Instead of generating all m partial products - only 2 partial products generated
- First partial product added - second subtracted - number of single-bit shift-right operations still m

UCSB

BParhami

# Booth's Algorithm - Rules

| $x_i$ | $x_{i-1}$ | Operation | Comments | $y_i$ |
|---|---|---|---|---|
| 0 | 0 | shift only | string of zeros | 0 |
| 1 | 1 | shift only | string of ones | 0 |
| 1 | 0 | subtract and shift | beginning of a string of ones | $\bar{1}$ |
| 0 | 1 | add and shift | end of a string of ones | 1 |

- Recoding multiplier xn-1 xn- 2...x1 x0 in SD code
- Recoded multiplier yn-1 yn-2 ... y1 y0
- xi,xi-1 of multiplier examined to generate yi
- Previous bit - xi-1 - only reference bit
- i=0 - reference bit x-1=0
- Simple recoding - yi = xi-1 - xi
- No special order - bits can be recoded in parallel
- Example: Multiplier 0011110011(0) recoded as 0100010101 - 4 instead of 6 add/subtracts

UCSB

BParhami

# Booth Algorithm Example

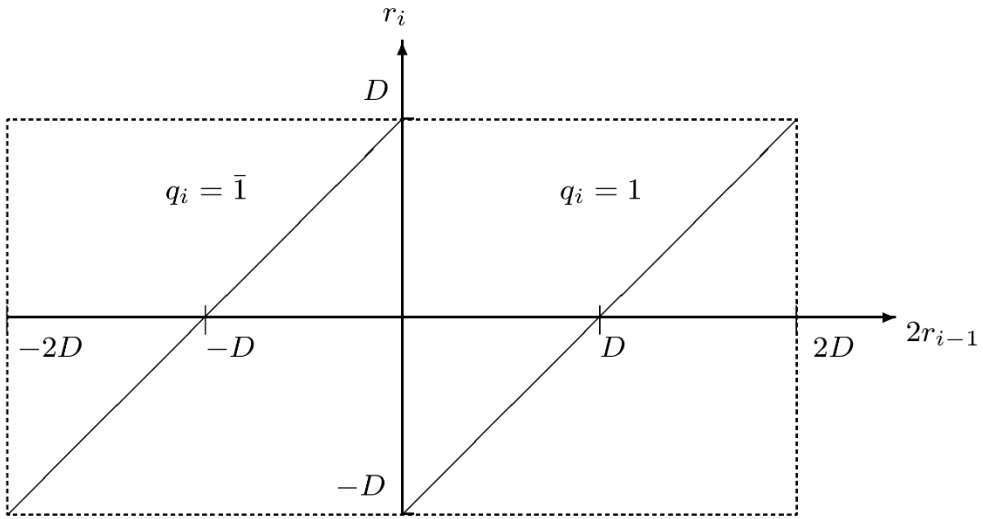
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $A$ | | 1 | 0 | 1 | 1 | | | $-5$ | |
| $X$ | $\times$ | 1 | 1 | 0 | 1 | | | $-3$ | |
| $Y$ | | 0 | $\bar{1}$ | 1 | $\bar{1}$ | | | recoded multiplier | |
| Add $-A$ | | 0 | 1 | 0 | 1 | | | | |
| Shift | | 0 | 0 | 1 | 0 | 1 | | | |
| Add $A$ | $+$ | 1 | 0 | 1 | 1 | | | | |
| | | 1 | 1 | 0 | 1 | 1 | | | |
| Shift | | 1 | 1 | 1 | 0 | 1 | 1 | | |
| Add $-A$ | $+$ | 0 | 1 | 0 | 1 | | | | |
| | | 0 | 0 | 1 | 1 | 1 | 1 | | |
| Shift | | 0 | 0 | 0 | 1 | 1 | 1 | 1 | |

UCSB

BParhami

# Nonrestoring Division

- Simpler and faster than selection rule for restoring division - $2r_{i-1}$ compared to 0 instead of to D
- Same equation for remainder - $q_i = 2r_{i-1} - q_i D$
- Divisor D subtracted if $2r_{i-1} > 0$ , added if $< 0$
  - $|r_{i-1}| < D$
  - $q_i$ selected so $|r_i| < D$
  - $q_i \neq 0$ - at each step, either addition or subtraction is performed

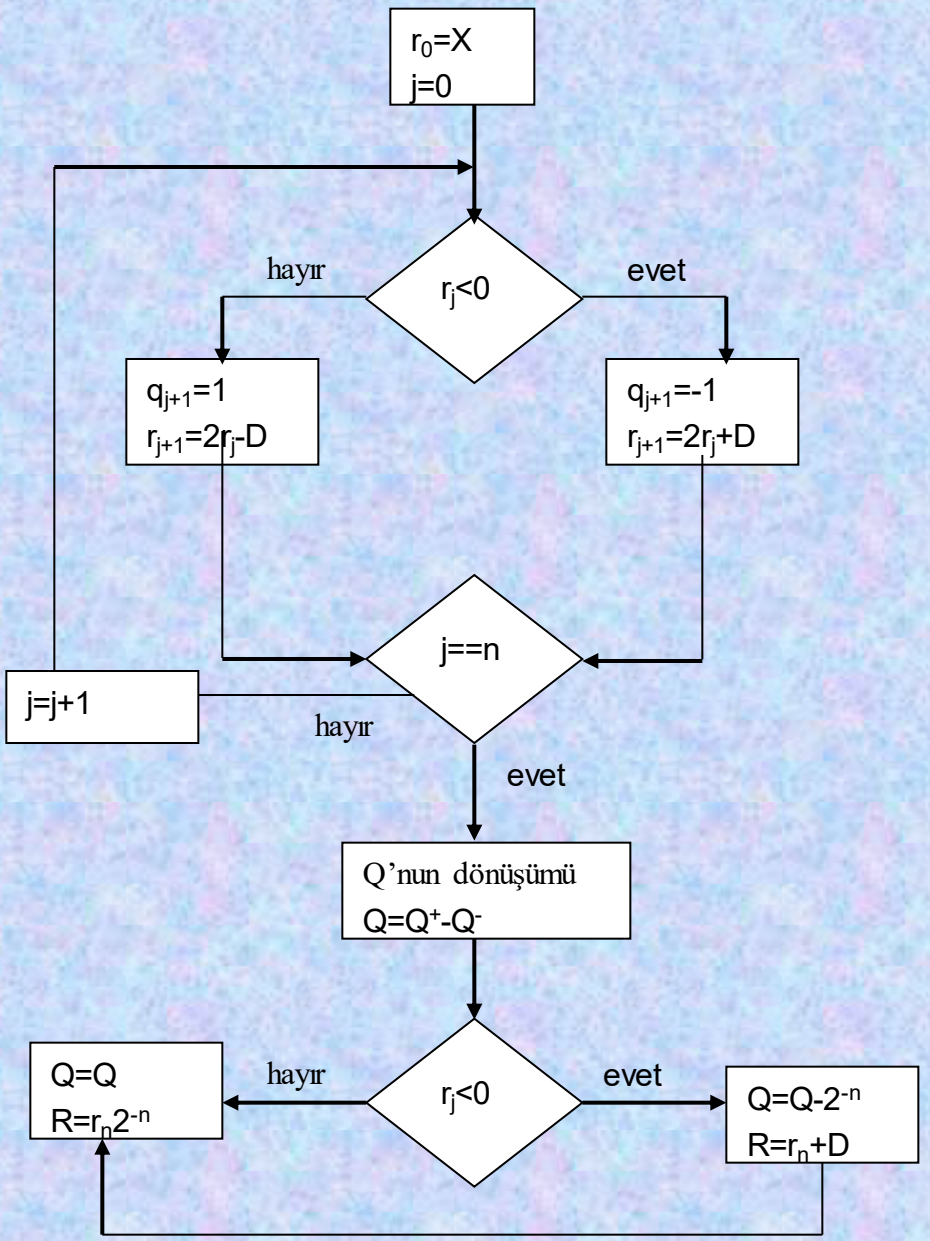



- Not SD representation no redundancy in representation of quotient
- Exactly m add/subtract and shift operations

# Non-Restoring Division

Restore işlemi yoktur, $q_j=\{\overline{1},1\}$

$\overline{1}=-1$

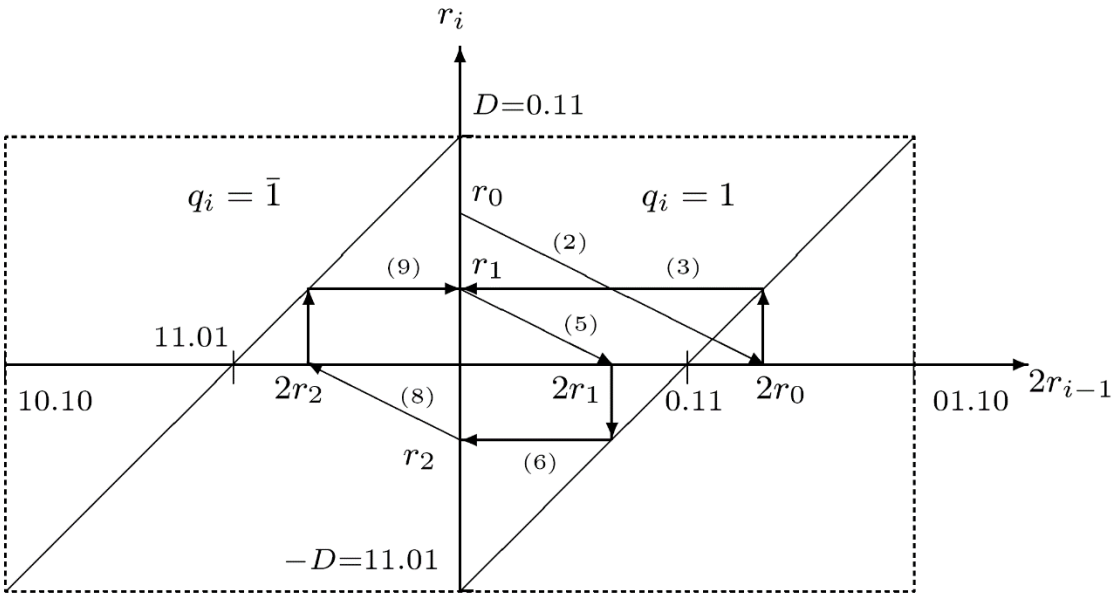


# Nonrestoring Division - Example 1

- $X=(0.100000)_2=1/2$
- $D=(0.110)_2=3/4$
- Final remainder - as before
- $Q=0.11\bar{1}=0.101_2=5/8$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (1) | $r_0 = X$ | | | 0 | .1 | 0 | 0 | |
| (2) | $2r_0$ | | 0 | 1 | .0 | 0 | 0 | set $q_1 = 1$ |
| (3) | Add $-D$ | + | 1 | 1 | .0 | 1 | 0 | |
| (4) | $r_1$ | | 0 | 0 | .0 | 1 | 0 | |
| (5) | $2r_1$ | | 0 | 0 | .1 | 0 | 0 | set $q_2 = 1$ |
| (6) | Add $-D$ | + | 1 | 1 | .0 | 1 | 0 | |
| (7) | $r_2$ | | 1 | 1 | .1 | 1 | 0 | |
| (8) | $2r_2$ | | 1 | 1 | .1 | 0 | 0 | set $q_3 = \bar{1}$ |
| (9) | Add $D$ | + | 0 | 0 | .1 | 1 | 0 | |
| (10) | $r_3$ | | 0 | 0 | .0 | 1 | 0 | |

- Graphical representation
  - * Horizontal lines - add ±D
  - * Diagonal lines - multiply by 2



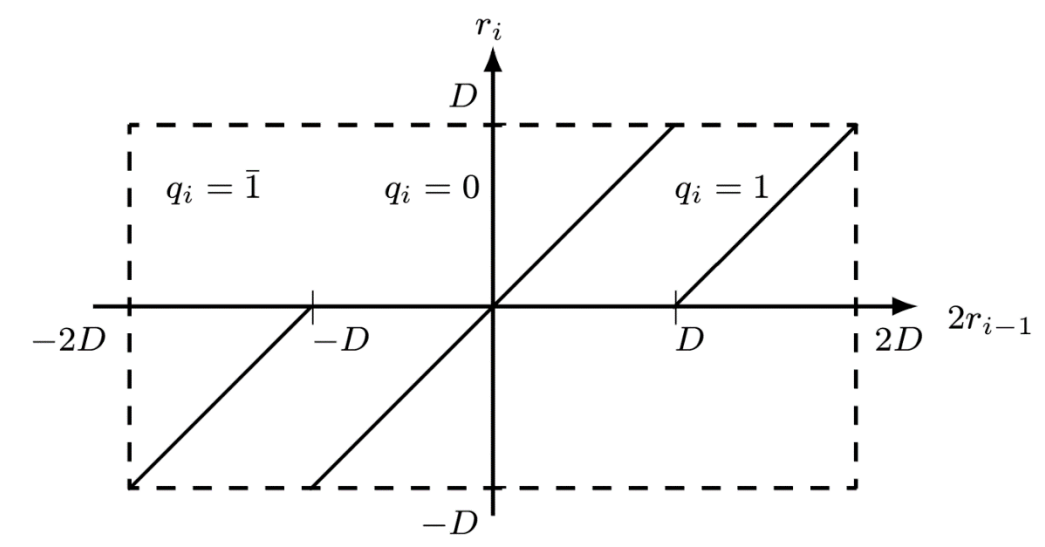
# Nonrestoring Division - Example 2

- X=0.110$_2$= 5/8
- D=0.110$_2$= 3/4

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $r_0 = X$ | | | 0 | .1 | 0 | 1 | |
| $2r_0$ | | 0 | 1 | .0 | 1 | 0 | set $q_1 = 1$ |
| Add $-D$ | + | 1 | 1 | .0 | 1 | 0 | |
| $r_1$ | | 0 | 0 | .1 | 0 | 0 | |
| $2r_1$ | | 0 | 1 | .0 | 0 | 0 | set $q_2 = 1$ |
| Add $-D$ | + | 1 | 1 | .0 | 1 | 0 | |
| $r_2$ | | 0 | 0 | .0 | 1 | 0 | |
| $2r_2$ | | 0 | 0 | .1 | 0 | 0 | set $q_3 = 1$ |
| Add $-D$ | + | 1 | 1 | .0 | 1 | 0 | |
| $r_3$ | | 1 | 1 | .1 | 1 | 0 | |

- Final remainder negative - dividend positive
- Correct final remainder by adding D to $r_3$ - 1.110+0.110=0.100
- Correct quotient - Qcorrected = Q - ulp
- Q=0.111 - Qcorrected=0.110$_2$=3/4

# Modified Nonrestoring Division – Fast Division Algorithm: SRT



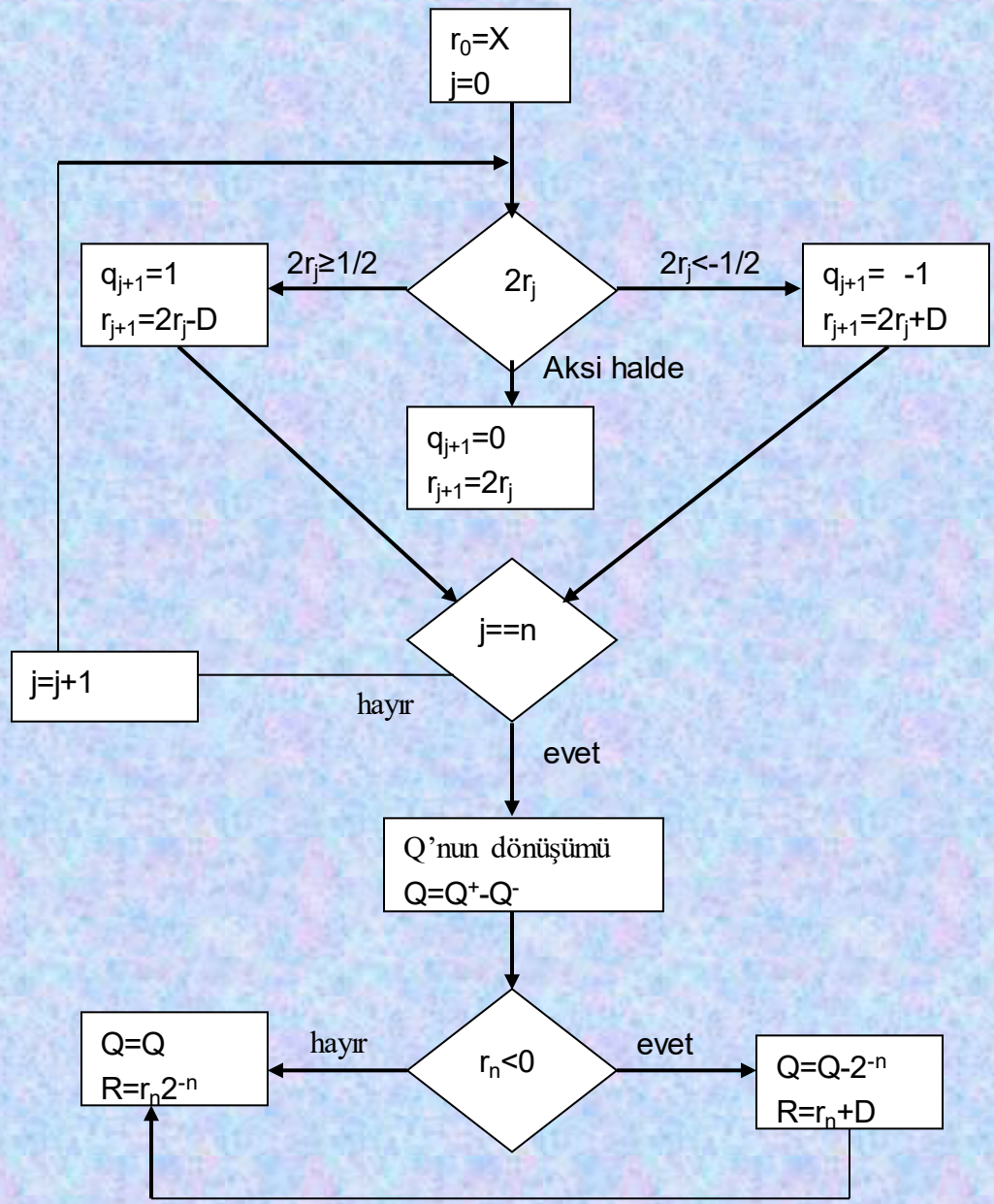

- Problem: full comparison of 2ri-1 with either D or -D required
- Solution: restricting D to normalized fraction $1/2 \leq |D| < 1$
- Region of 2ri-1 for which qi=0 reduced to

$$-D \leq -\frac{1}{2} \leq 2r_{i-1} < \frac{1}{2} \leq D$$

# Modified Nonrestoring → SRT

- Advantage: Comparing partial remainder 2ri-1 to 1/2 or -1/2, not D or -D
- Binary fraction in two's complement representation
  - ≥ 1/2 if and only if it starts with 0.1
  - ≤ -1/2 if and only if it starts with 1.0
- Only 2 bits of 2ri-1 examined - not full comparison between 2ri-1 and D
  - In some cases (e.g., dividend X>1/2) - shifted partial remainder needs an integer bit in addition to sign bit - 3 bits of 2ri-1 examined
- Selecting

$$q_i = \begin{cases} 1 & \text{if } 2r_{i-1} \geq 1/2 \\ 0 & \text{if } -1/2 \leq 2r_{i-1} < 1/2 \\ \bar{1} & \text{if } 2r_{i-1} < -1/2. \end{cases}$$


$r_0=X$
$j=0$
$2r_j$
$2r_j \geq 1/2$
$q_{j+1}=1$
$r_{j+1}=2r_j-D$
$2r_j<-1/2$
$q_{j+1}=-1$
$r_{j+1}=2r_j+D$
Aksi halde
$q_{j+1}=0$
$r_{j+1}=2r_j$
j==n
j=j+1
hayır
evet
Q'nun dönüşümü
$Q=Q^+-Q^-$
$r_n<0$
hayır
$Q=Q$
$R=r_n2^{-n}$
evet
$Q=Q-2^{-n}$
$R=r_n+D$

# Example

- X=$(0.00111111)_2$=63/256  D=$(0.1001)_2$=9/16

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $r_0 = X$ | | 0 | .0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | |
| $2r_0$ | | 0 | .0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | $< 1/2$ set $q_1 = 0$ |
| $2r_1$ | | 0 | .1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | $\geq 1/2$ set $q_2 = 1$ |
| Add $-D$ | + | 1 | .0 | 1 | 1 | 1 | | | | | |
| $r_2$ | | 0 | .0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | |
| $2r_2$ | | 0 | .1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | $\geq 1/2$ set $q_3 = 1$ |
| Add $-D$ | + | 1 | .0 | 1 | 1 | 1 | | | | | |
| $r_3$ | | 0 | .0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | |
| $2r_3$ | | 0 | .1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | $\geq 1/2$ set $q_4 = 1$ |
| Add $-D$ | + | 1 | .0 | 1 | 1 | 1 | | | | | |
| $r_4$ | | 0 | .0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | zero final remainder |

Q =$0.0111_2$=7/16 - not a minimal representation in SD form

Number of add/subtracts can be reduced further

# Extension to Negative Divisors

$$q_i = \begin{cases} 0 & \text{if} \quad |2r_{i-1}| < 1/2 \\ 1 & \text{if} \quad |2r_{i-1}| \geq 1/2 \quad \& \quad r_{i-1} \text{ and } D \text{ have the same sign} \\ \bar{1} & \text{if} \quad |2r_{i-1}| \geq 1/2 \quad \& \quad r_{i-1} \text{ and } D \text{ have opposite signs} \end{cases}$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $r_0 = X$ | | 0 | .0 | 1 | 0 | 1 | |
| $2r_0$ | | 0 | .1 | 0 | 1 | 0 | $\geq 1/2$ set $q_1 = 1$ |
| Add $-D$ | + | 1 | .0 | 1 | 0 | 0 | |
| $r_1$ | | 1 | .1 | 1 | 1 | 0 | |
| $2r_1 = r_2$ | | 1 | .1 | 1 | 0 | 0 | $\geq -1/2$ set $q_2 = 0$ |
| $2r_2 = r_3$ | | 1 | .1 | 0 | 0 | 0 | $\geq -1/2$ set $q_3 = 0$ |
| $2r_3$ | | 1 | .0 | 0 | 0 | 0 | $< -1/2$ set $q_4 = \bar{1}$ |
| Add $D$ | + | 0 | .1 | 1 | 0 | 0 | |
| $r_4$ | | 1 | .1 | 1 | 0 | 0 | negative remainder & positive $X$ |
| Add $D$ | + | 0 | .1 | 1 | 0 | 0 | correction |
| $r_4$ | | 0 | .1 | 0 | 0 | 0 | corrected final remainder |

- Dividend X=(0.0101)=5/16
- Divisor D=(0.1100)=3/4
- Before correction Q=(Q+)- (Q-) = 0.1000 – 0.0001
- minimal SD repr. of Q=0.0111 - minimal number of add/subtracts
- After correction, Q = 0.0111-ulp = 0.0110 = 3/8; final remainder = 1/2 * 1/16=1/32

# *Arithmetic Functions*

- Karekök Alma / Square-Roothing
- Üstel Fonksiyonlar / Exponential Functions
- Logaritmik Fonksiyonlar
- Trigonometrik Fonksiyonlar
- Hiperbolik Fonksiyonlar

# Referanslar

- Textbook: *Computer Arithmetic Algorithms*, I. Koren, 2nd Edition, A.K. Peters, Natick, MA, 2002
- Textbook web page: http://www.ecs.umass.edu/ece/koren/arith
- Recommended Reading:
  - B. Parhami, *Computer Arithmetic: Algorithms and Hardware Design*, Oxford University Press, 2000
  - M. Ercegovac and T. Lang, *Digital Arithmetic,* Morgan Kaufman, 2003