DIGITAL ARITHMETIC

Miloš D. Ercegovac
Computer Science Department
University of California Los Angeles
and
Tomás Lang
Department of Electrical and Computer Engineering
University of California at Irvine

ABOUT THE BOOK

1. Overview [Chapter 1]
2. Two-Operand Addition [Chapter 2]
3. Multi-operand Addition [Chapter 3]
4. Multiplication [Chapter 4]
5. Division by Digit Recurrence [Chapter 5]
6. Square Root by Digit Recurrence [Chapter 6]
7. Reciprocal, Division, Reciprocal Square RToot, and Inverse Square Root by Iterative Approximation [Chapter 7]
8. Floating-point Representation, Algorithms, and Implementations [Chapter 8]
9. Digit-Serial Arithmetic [Chapter 9]
10. Function Evaluation [Chapter 10]
11. CORDIC Algorithm and Implementations [Chapter 11]

Chapter 1: Review of the Basic Number Representations and Arithmetic Algorithms

- General-purpose processors
  - ◇ Main use: numerical computations
  - ◇ Address calculations

- basic operations

- fixed point and floating point

- IEEE standard

- vector processors

- Special-purpose (application-specific) processors

- for numerically intensive applications

- single computation or classes of computations

Application-specific processor

- Areas of application:
  - signal processing
  - embedded systems
  - matrix computations
  - graphics, vision, multi-media
  - cryptography and security
  - robotics, instrumentation; others?
- Features:
  - better use of technology
  - improvement in speed, area, power
  - flexibility in
    * implementation; decomposition into modules
    * number systems and data formats; algorithms
- Need good design tools; difficult to change; FPGAs?

GENERAL-PURPOSE VS. APPLICATION-SPECIFIC

- Flexibility
- Matching specific applications
- Use of VLSI and special technology
- Use of hardware-level parallel processing
- Lower software overhead

ARITHMETIC PROCESSORS: USER'S VIEW

AP $=$ (operands, operation, results, conditions, singularities)

- Numerical operands and results specified by
  - Set of numerical values $x \in N$ (finite and ordered set)
  - Range $V_{min} \leq x \leq V_{max}$
  - Precision
  - Number representation system (NRS)
- Set of operations: addition, subtraction, multiplication, division, ...
- Conditions: values of the results – zero, negative, etc.
- Singularities: Illegal results – overflow, underflow, Nan, etc.

LEVELS OF DESCRIPTION AND IMPLEMENTATION

- Numerical computations (applications)
- Algorithms
- Arithmetic operations

Operations	Operands-Result	Algorithm
Numerical function	Numbers	Numerical Algorithm
	↓	↓
↓	*Number System*	*Arithmetic design*
	↓	↓
Digit-vector function	Digit vector	Digit-vector Algorithm
	↓	↓
↓	*Digit Coding*	*RTL design*
	↓	↓
Bit-vector function	Bit vector	Bit-vector Algorithm
		↓
		Logic Design

NUMBER REPRESENTATION SYSTEMS

value: $x \in N \Longrightarrow$NRS $\Longrightarrow X \in V$: digit vector

- Digit Vector

$$X = (x_a, \ldots, x_i, \ldots, x_b)$$

  – indexing:

  Leftward Zero Origin (LZ) (integers)

$$X = (x_{n-1}, x_{n-2}, \ldots, x_0)$$

  Rightward One Origin (RO) (fractions)

$$X = (x_1, x_2, \ldots, x_n)$$

  – Digit set D_i - set of values for digit x_i (usually consecutive integers)

NUMBER REPRESENTATION (cont.)

- Number of (unambiguously) representable numbers

$$|N| \leq \Pi |D_i|$$

- Number representation system

$$F : N \to V$$

- Choose NRS to
  - allow efficient computation(s)
  - suitable interface with other systems
- Different implementation-performance constraints

$\Longrightarrow$ variety of NRS

SOME CHARACTERISTICS OF NRS

a) Range: finite set of digit-vector values

b) Unambiguity: two numbers should not have same representation

$$\text{If } x \in N,\ y \in N,\ x \neq y \text{ then } F(x) \neq F(y)$$

c) Nonredundant/redundant

Redundant: $F^{-1}(X) = F^{-1}(Y)$

WEIGHTED NUMBER REPRESENTATION SYSTEMS

Integer x represented by digit vector $X = (x_{n-1} \ldots, x_0)$,

$$x = \sum_{i=0}^{n-1} x_i \cdot w_i$$

where

$$W = (w_{n-1}, \ldots, w_0) \; weight \; vector$$

Define

$$R = (r_{n-1}, \ldots, r_0) \; radix \; vector$$

so that

$$w_0 = 1 \; w_i = w_{i-1} r_{i-1}$$

WEIGHTED NUMBER REPRESENTATION (cont.)

- Fixed-radix NRS

$$r_i = r$$

Then $w_i = r^i$ so that

$$x = \sum_{i=0}^{n-1} x_i r^i$$

- Canonical digit set

$$D_i = \{0, 1, 2, \ldots, |r_i| - 1\}$$

- Conventional number system
  - Fixed radix positive
  - Canonical digit set

NON-CONVENTIONAL FIXED-RADIX SYSTEM

- Negative radix

  $r = -2$, $x = \Sigma_{i=0}^{n-1} x_i(-2)^i$

  1011 = (-8) + 0 + (-2) + 1 = -9

  0111 = 0 + 4 + (-2) + 1 = 3

- Complex radix

  $r = 2j$, $j = \sqrt{-1}$, $x_i \in \{0, 1, 2, 3\}$ (Knuth's quarter imaginary NRS)

  W: -8j -4 +2j +1

  1231 $\Longrightarrow 1 \times (-8j) + 2 \times (-4) + 3 \times (2j) + 1 \times 1 = -7 - 2j$

- Non-canonical digit set, $r = 2$, {-1,0,1} or {0,1,2}

  Example: radix 4 $D = \{-3, -2, -1, 0, 1, 2, 3\}$

  $x = 27$ represented by $(1, 2, 3)$ or $(2, -2, 3)$

REDUNDANT NRS

- Fixed radix r
- Non-canonical digit set

$$D = \{-a, -a+1, \ldots, -1, 0, 1, \ldots, b-1, b\}$$

- Symmetric if $a = b$
- Redundant $a + b + 1 > r \quad (a, b \leq r - 1)$
  - "standard" $a, b \leq r - 1$
  - over-redundant $a, b > r - 1$
  - Redundancy factor

$$\rho \;=\; \frac{a}{r-1}, \quad \rho > \frac{1}{2}$$

EXAMPLES OF REDUNDANT DIGIT SETS

r	a	Digit set	ρ	Comment
2	1	$\{$ -1, 0, 1$\}$	1	minimally/maximally redundant
4	2	$\{$-2, -1, 0, 1, 2$\}$	2/3	minimally redundant
4	3	$\{$-3, -2, . . . , 2, 3$\}$	1	maximally redundant
4	4	$\{$-4, . . . , 4$\}$	4/3	over-redundant
9	4	$\{$-4, . . . , 4$\}$	1/2	non-redundant
10	5	$\{$-5, . . . , 5$\}$	5/9	minimally redundant
10	6	$\{$-6, . . . , 6$\}$	2/3	redundant
10	9	$\{$-9, . . . , 9$\}$	1	maximally redundant
10	13	$\{$-13, . . . , 13$\}$	13/9	over-redundant

MIXED-RADIX NUMBER SYSTEM

- $r_i \neq r_j$
- Example: Representation of time $R = (31, 24, 60, 60)$
- Example: Factorial number system

$$\begin{aligned} r_i &= i + 2, \quad i = 0, \ldots, n-1 \\ R &= (n+1, n, \ldots, 3, 2) \\ w_i &= (i+1)! \end{aligned}$$

Canonical digit set

Integers in range $0 \leq x \leq (n+1)! - 1$

NON-WEIGHTED NUMBER SYSTEMS: RESIDUE (RNS)

- Base vector B of moduli m_i

$$B = (m_{n-1}, m_{n-2}, \ldots, m_0)$$

m_i positive integers and pairwise relatively prime

- Integer x is represented by vector

$$X = (x_{n-1}, x_{n-2}, \ldots, x_0)$$

where $x_i = x \; mod \; m_i$

- Represents uniquely integers in the range

$$0 \leq x < \Pi_{i=0}^{n-1} m_i$$

(more later)

REPRESENTATION OF SIGNED INTEGERS

A. Directly in the number representation

Examples: Signed-Digit Number System

B. With an extra symbol: Sign-and-magnitude

C. Additional mapping on positive integers

Signed integers x

$\downarrow$

Positive integers x_R

$\downarrow$

Digit-vectors X

Examples:

- True-and-Complement (TC):
  - 2's complement
  - 1s' complement
- Biased representation

TRUE-AND-COMPLEMENT SYSTEM

- $-k \leq x \leq k$ signed integer (implicit value)
- x_R positive integer (representation value)
- C – complementation constant
- Mapping $x_e = x \ mod \ C$
- Unambiguous if $k < C/2$
- Equivalent to

$$x_R = \begin{cases} x & \textbf{if } x \geq 0 \\ C - |x| & \textbf{if } x < 0 \end{cases}$$

- Converse mapping

$$x = \begin{cases} x_R & \textbf{if } x_R < C/2 \\ x_R - C & \textbf{if } x_R > C/2 \end{cases}$$

NUMBER REPRESENTATION (cont.)

- x_R represented in any number system
- In fixed-radix system two common choices:
  - 2's complement: $C = r^n$ (Range-complement system)
  - 1s' complement: $C = r^n - 1$ (Diminished-radix-complement system)

BIASED REPRESENTATION

- B – bias
- $-k \leq x \leq k$
- $x_R = x + B$
- $B \geq k$

TYPES OF ARITHMETIC ALGORITHMS

- Bottom-up development

  Primitives

  + Addition/subtraction
  + Multioperand addition
  + Arithmetic shifts
  + Multiplication by digit
  + Result-digit selection (PLA)
  + Table look-up
  + Multiplication

  - Algorithms

  + Composition of primitives

TYPES (cont.)

- (Digit) Recurrences (continued sums)

◇ Residual recurrence: $R[i+1] = f(R[i], X, Y, Z[i], z_{i+1})$

Uses: Add/sub, single-position shifts, multiplication by digit

◇ Output digit selection: $z_{i+1} = g(R[i], X, Y, Z[i])$

(keep $R[i+1]$ bounded)

Uses: Comparisons, PLA

◇ Result recurrence

$$Z[i+1] = Z[i] + z_{i+1}r^{i+1} \text{ (continued sum)}$$

Uses: Concatenation

- Examples:

◇ multiplication $R[i+1] = \frac{1}{r}(R[i] + X \cdot r^n y_i)$

◇ division $R[i+1] = rR[i] - q_{i+1}Y \quad q_{i+1} = g(R[i+1], Y)$

Types of algorithms

- Continued product recurrences

  $R[i+1] = f(R[i], X, Y, Z[i], z_{z+1})$

  Uses: Add/sub, variable shifts, mult. by digit

  $z_{i+1} = g(R[i], X, Y, Z[i])$(keep $R[i+1]$ bounded)

  Uses: Comparisons, PLA

  $Z[i+1] = Z[i](1 + z_{i+1}r^{-(i+1)})$ (continued product)

  Uses: Variable shift, addition

- Example:

◇ division

$$R[i+1] = rR[i](1 + q_{i+1}r^{-(i+1)}) + q_{i+1}$$
$$q_{i+1} = g(R[i], Y)$$
$$Q[i+1] = Q[i](1 + q_{i+1}r^{-(i+1)})$$

TYPES (cont.)

- Iterative Approximations

  $Z[i+1] = f(Z[i], X, Y) \;\; \text{until } g(Z(i)) < \varepsilon$

  Example:

  ◇ reciprocal

  $Z[i+1] = Z[i](2 - Z[i]X)$

- Polynomial Approximations

  $z = a_0 + a_1 x + a_2 x^2 + \cdots$

PERFORMANCE

- Measures
  - + Execution time
  - + Throughput
- Improving speed

a) Arithmetic level

+ Reducing number of steps

  Example: higher radix

  Example: combinational instead of sequential

+ Reducing time of step

  Example: carry-save adder instead of carry-propagate

+ Overlap steps (concurrency/pipelining)

  Example: multiple generation and addition (in mult.)

  Example: simultaneous additions (in mult.)

b) Implementation level

+ Reduce number of logic levels

POWER AND COST

- Measures
  - + Packaging
  - + Interconnection complexity
  - + Number of pins
  - + Number of chips and types of chips
  - + Number of gates and types of gates
  - + Area
  - + Design cost; verification and testing cost
  - + Power dissipation
  - + Power consumption
- Reduction of cost

FAST TWO-OPERAND ADDITION

A. Conventional number system.

Carry-propagate adders (CPA)

- Switched carry-ripple adder
- Carry-skip adder
- Carry-lookahead adder
- Prefix adder
- Carry-select adder and conditional-sum adder
- Variable-time adder

B. Redundant number system.

Totally-parallel adders (TPA); adders with limited carry propagation

- Carry-save adder
- Signed-digit adder

n-BIT ADDITION

$$x + y + c_{in} = 2^n c_{out} + s$$

The solution:

$$\begin{aligned} s &= (x + y + c_{in}) \bmod 2^n \\ c_{out} &= \begin{cases} 1 & \text{if } (x + y + c_{in}) \geq 2^n \\ 0 & \text{otherwise} \end{cases} \\ &= \lfloor (x + y + c_{in})/2^n \rfloor \end{aligned}$$

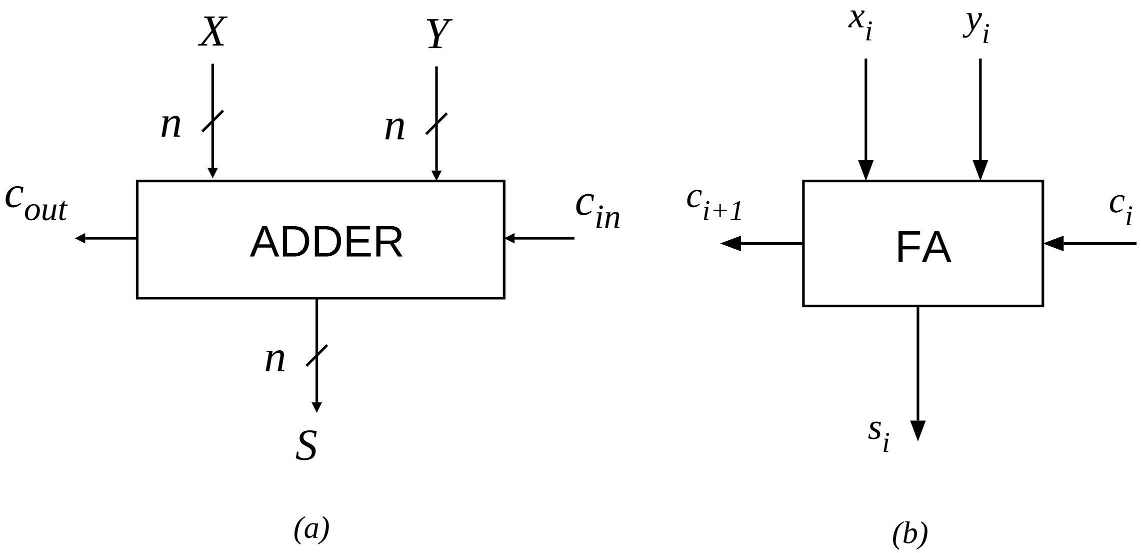


Figure 2.1: (a) An n-bit adder. (b) 1-bit adder (full adder module).

1-BIT ADDITION

- Primitive module *full adder* (FA)

$$x_i + y_i + c_i = 2c_{i+1} + s_i$$

with solution

$$s_i = (x_i + y_i + c_i) \bmod 2$$
$$c_{i+1} = \lfloor (x_i + y_i + c_i)/2 \rfloor$$

ADDITION: TWO-STEP PROCESS

1. Obtain carries (carry at i depends on $j \leq i$)
   – non-trivial to do fast
2. Compute sum bits (local function)

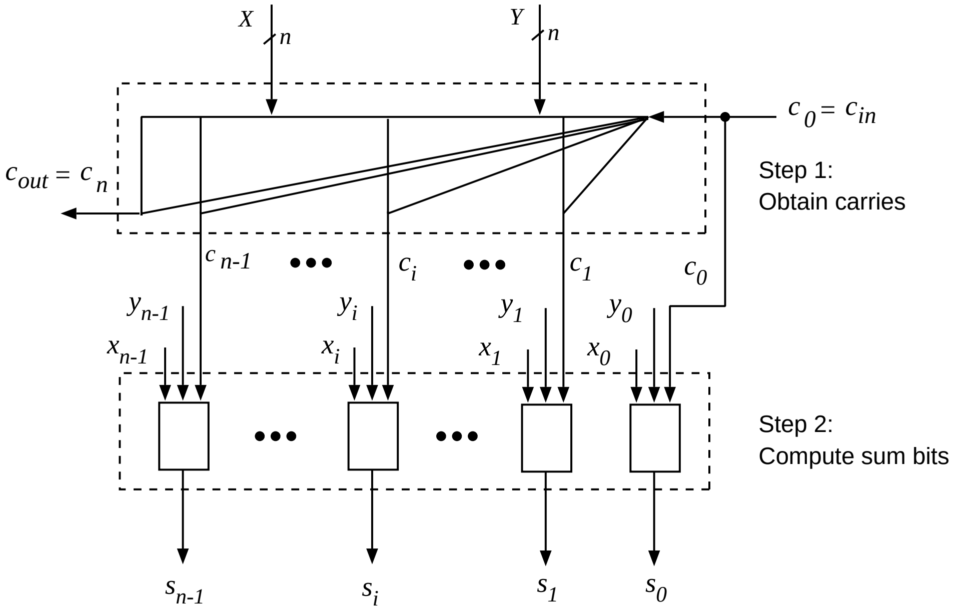


Figure 2.2: Steps in addition.

CARRY-OUT CASES

Case	x_i	y_i	$x_i + y_i$	c_{i+1}	Comment
1	0	0	0	0	kill (stop) carry-in
2	0	1	1	c_i	propagate carry-in
	1	0	1	c_i	propagate carry-in
3	1	1	2	1	generate carry-out

Case 1 (Kill): $k_i = x_i'y_i' = (x_i + y_i)'$
Case 2 (Propagate): $p_i = x_i \oplus y_i$
Case 3 (Generate): $g_i = x_i y_i$
Then

$$c_{i+1} = g_i + p_i c_i = x_i y_i + (x_i \oplus y_i) c_i$$

Alternative (simpler) expression:

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

Since $a_i = k_i'$ we call it "alive"

CARRY CHAINS

Two types:

1-carry chain consisting of carry=1
0-carry chain consisting of carry=0

i	9		8	7		6		5		4	3		2		1		0
x_i	1		0	1		0		1		1	1		1		0		0
y_i	0		0	0		1		0		1	0		0		1		0
	p		k	p		p		p		g	p		p		p		k
	a			a		a		a		a	a		a		a		
c_{i+1}	0	$\leftarrow$	0	1	$\leftarrow$	1	$\leftarrow$	1	$\leftarrow$	1	0	$\leftarrow$	0	$\leftarrow$	0	$\leftarrow$	0

Generalization to group of bits

$$c_{j+1} = g_{(j,i)} + p_{(j,i)}c_i = g_{(j,i)} + a_{(j,i)}c_i$$

or, for $i = 0$

$$c_{j+1} = g_{(j,0)} + p_{(j,0)}c_0 = g_{(j,0)} + a_{(j,0)}c_0$$

Recursive combining of subranges of variables:

$$\begin{aligned}
g_{(f,d)} &= g_{(f,e)} + p_{(f,e)}g_{(e-1,d)} = g_{(f,e)} + a_{(f,e)}g_{(e-1,d)} \\
a_{(f,d)} &= a_{(f,e)}a_{(e-1,d)} \\
p_{(f,d)} &= p_{(f,e)}p_{(e-1,d)}
\end{aligned}$$

Generalization (cont.)

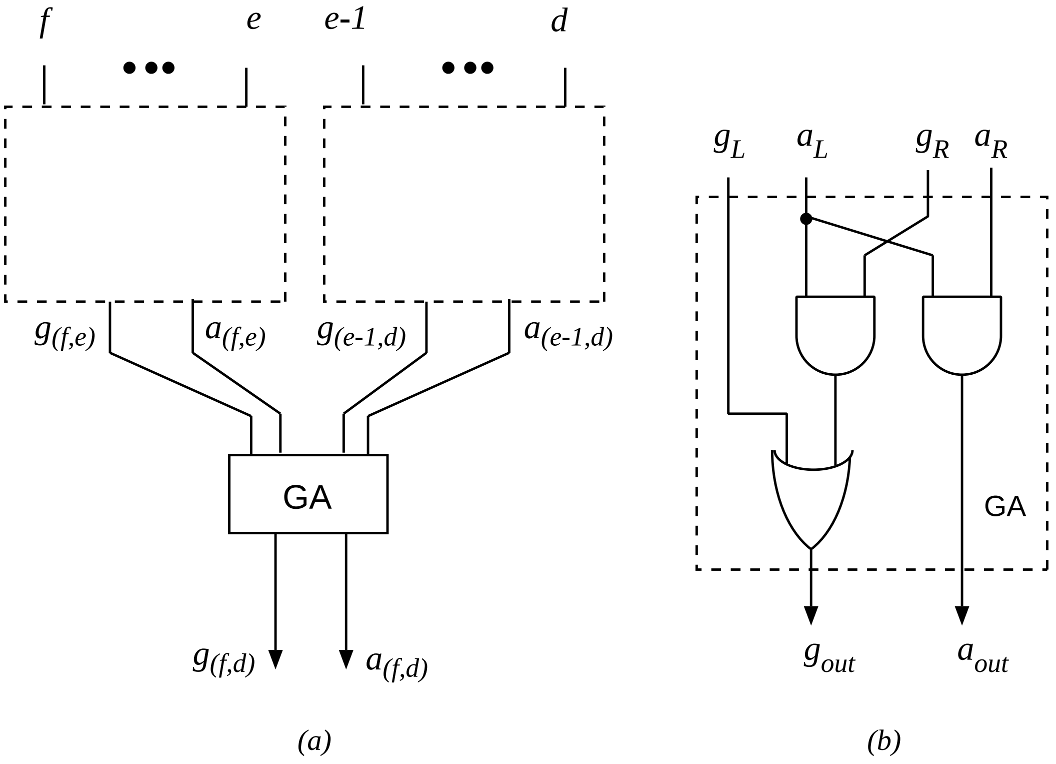


Figure 2.3: Computing $(g_{(f,d)}, a_{(f,d)})$.

BASIC CARRY-RIPPLE ADDER (CRA)

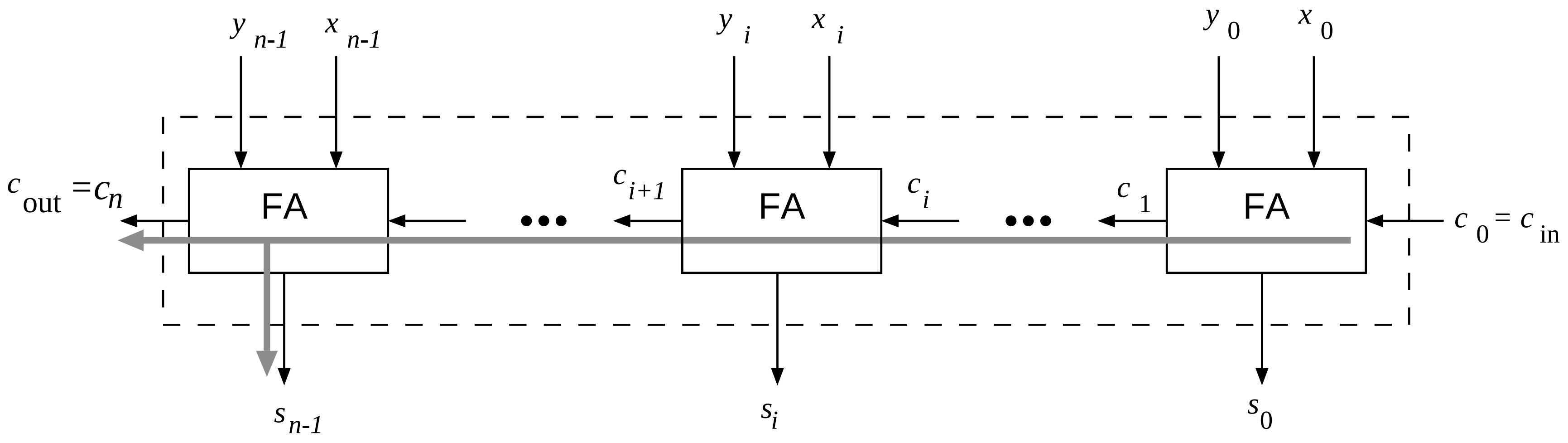


Figure 2.4: Carry-ripple adder.

$$T_{CRA} = (n-1)t_c + \max(t_c, t_s)$$

Implementations of full-adder

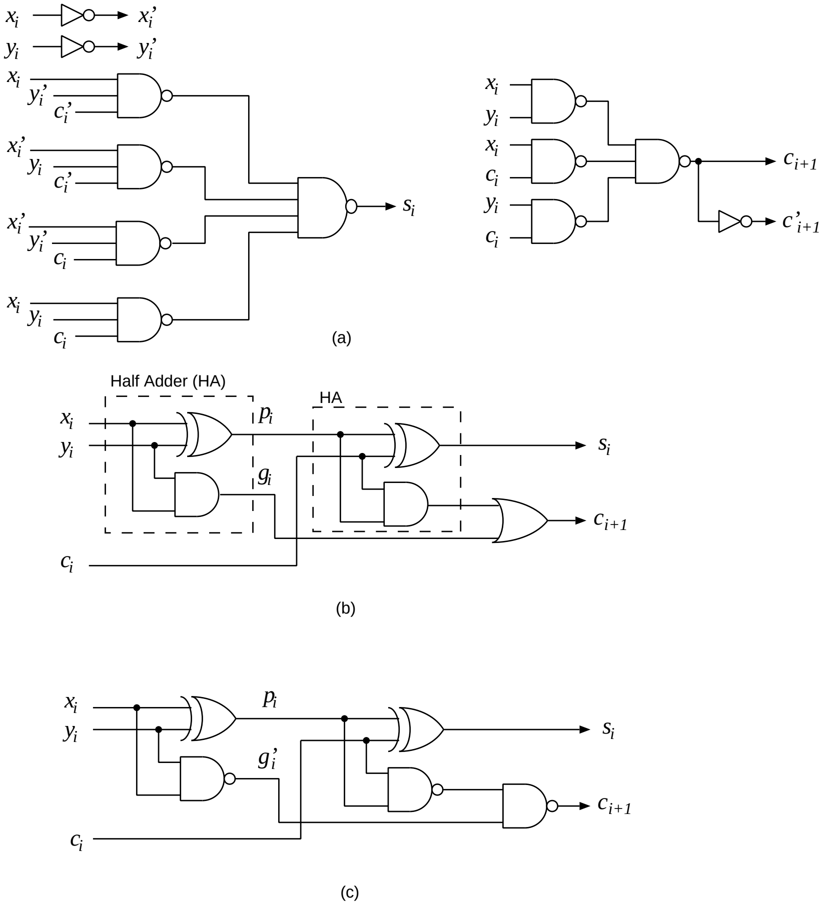


Figure 2.5: Implementation of full-adder. (a) Two-level network. (b) Multilevel network with XOR, AND and OR gates; (c) Multilevel implementation with XOR and NAND gates.

SWITCHED CARRY-RIPPLE (Manchester) ADDER

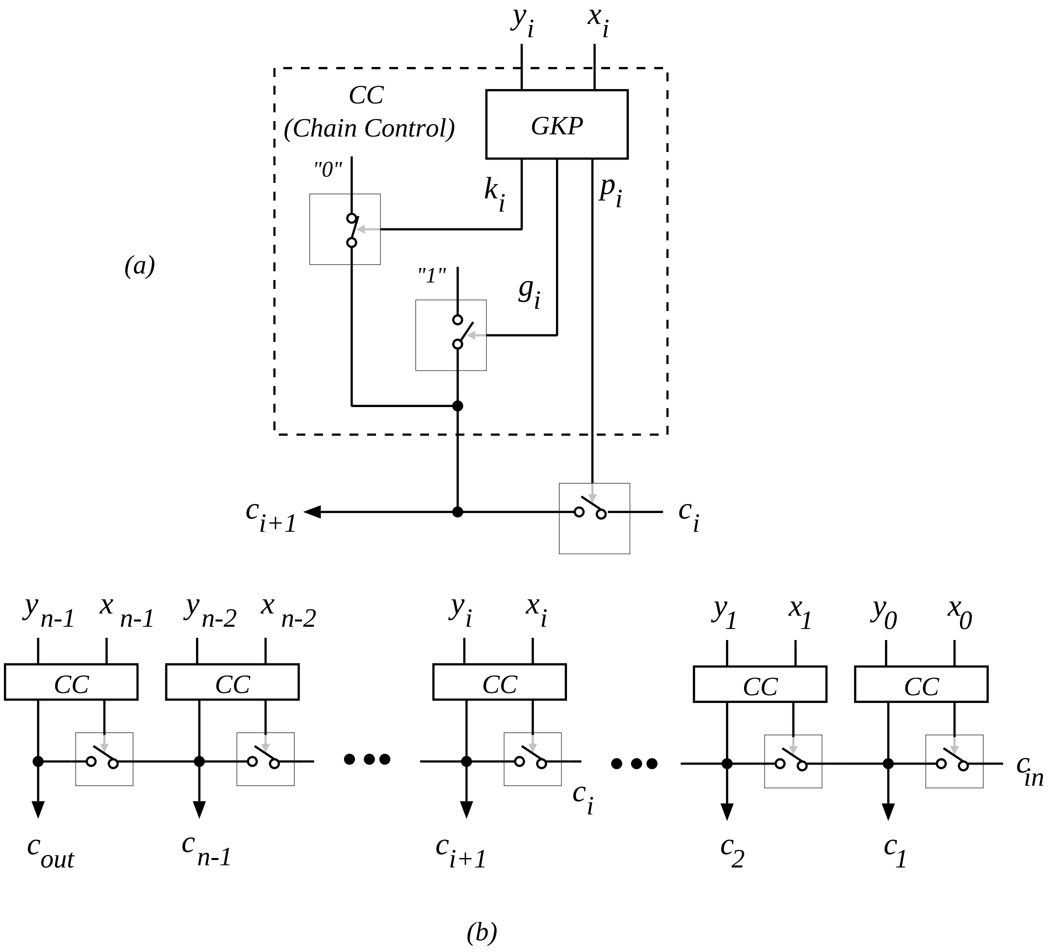


Figure 2.6: Switch carry-ripple network (Manchester circuit)

CARRY-SKIP ADDER

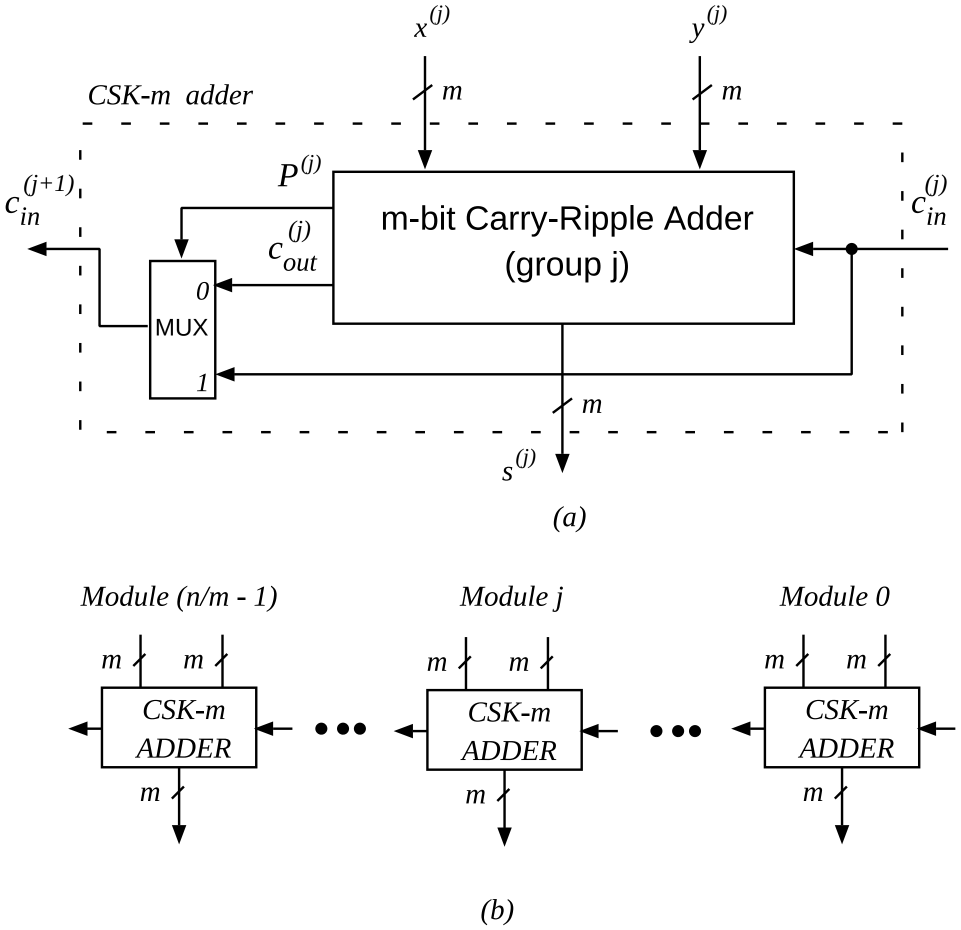


Figure 2.7: Carry-skip adder: (a) A group with carry bypass. (b) n-bit CSK adder.

CARRY CHAINS IN CARRY-SKIP ADDER

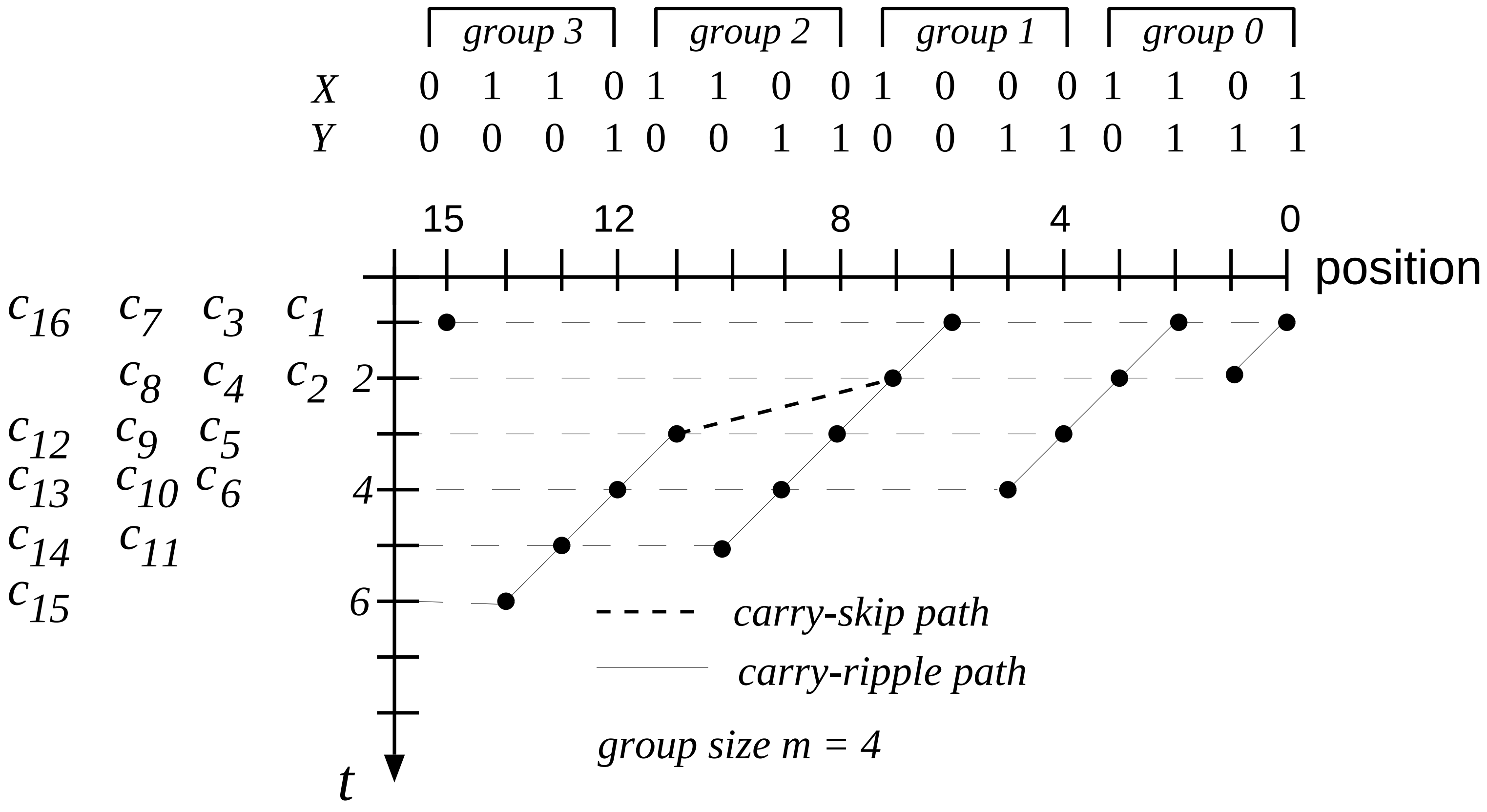
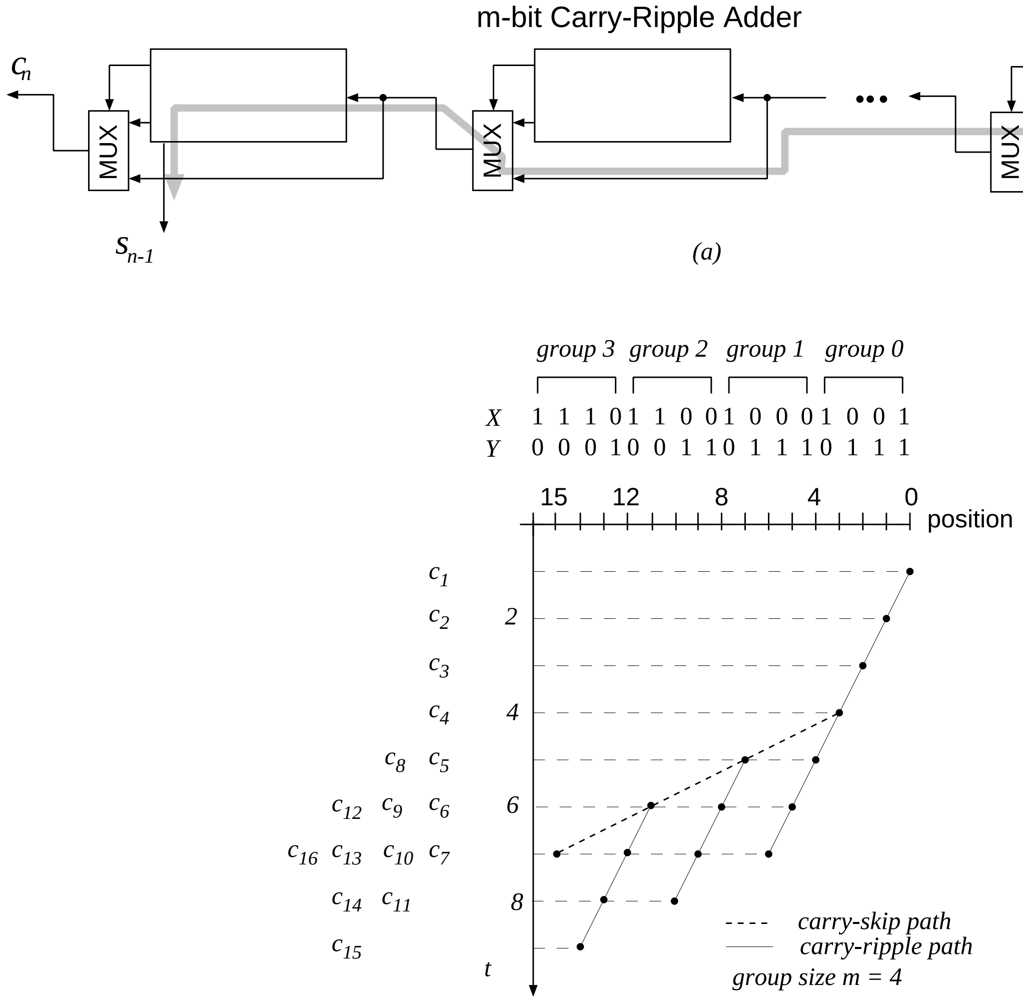


Figure 2.8: Carry chains in carry-skip adder: A case with several carry chains.



(a)

(b)

Figure 2.9: (a) Critical path in carry-skip adder. (b) The worst-case situation for $n = 16$.

WORST-CASE DELAY

$$\begin{aligned} T_{CSK} &= mt_c + t_{mux} + (\frac{n}{m} - 2)t_{mux} + (m-1)t_c + t_s \\ &= (2m-1)t_c + (\frac{n}{m} - 1)t_{mux} + t_s \end{aligned}$$

PROBLEM WITH CLEARING OF CARRIES

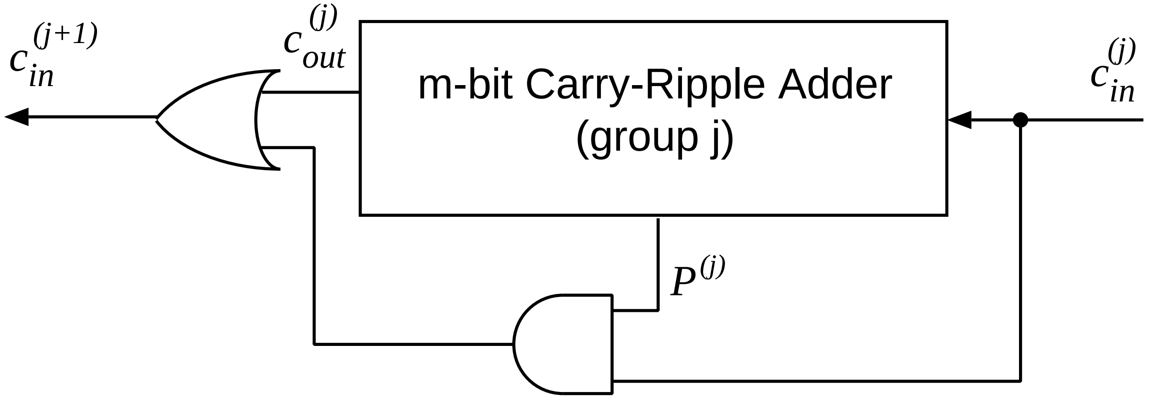


Figure 2.10: Carry-skip adder using AND-OR for bypass

GROUP SIZE IN CARRY-SKIP ADDERS

Fixed-size:

$$\begin{aligned} m_{opt} &= \left(\frac{t_{mux}}{2t_c} n\right)^{1/2} \quad \text{(minimum delay)} \\ T_{opt} &\approx (8t_{mux}t_c n)^{1/2} \end{aligned}$$

Variable-size:

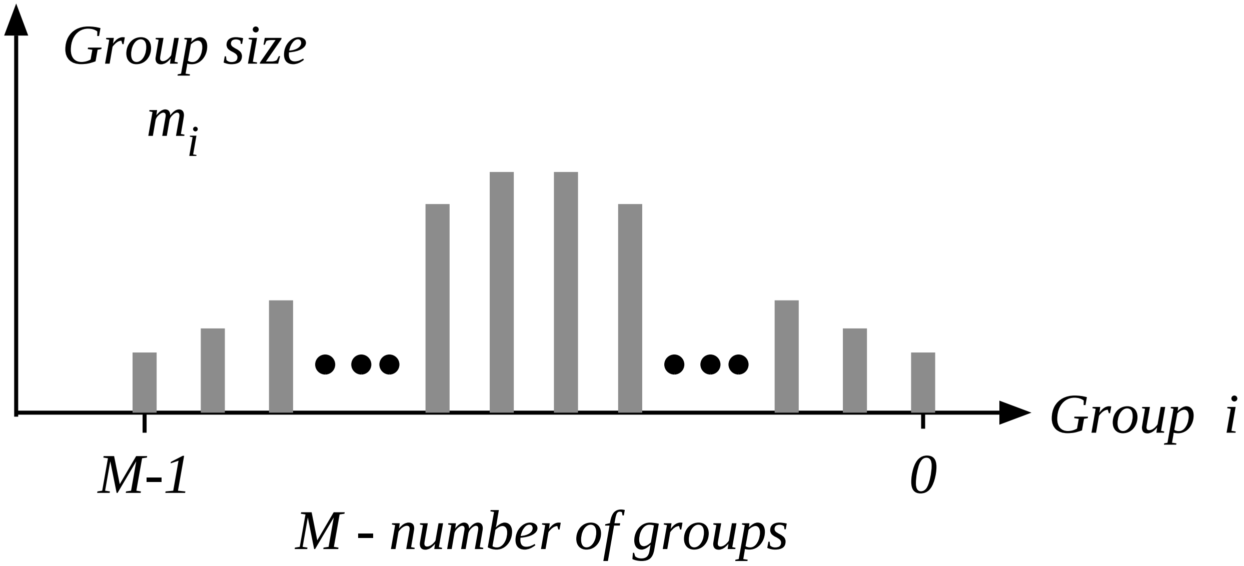


Figure 2.11: Optimal distribution of group sizes in carry-skip adder.

CARRY-LOOKAHEAD ADDER(CLA)

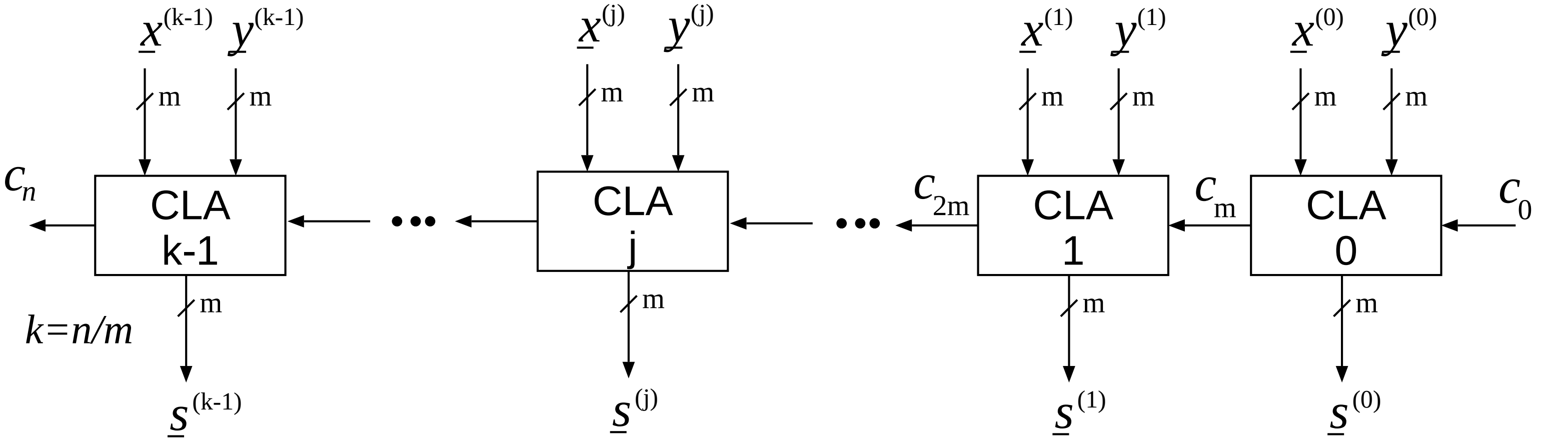


Figure 2.12: One-level carry-lookahead adder

CARRY-LOOKAHEAD MODULE

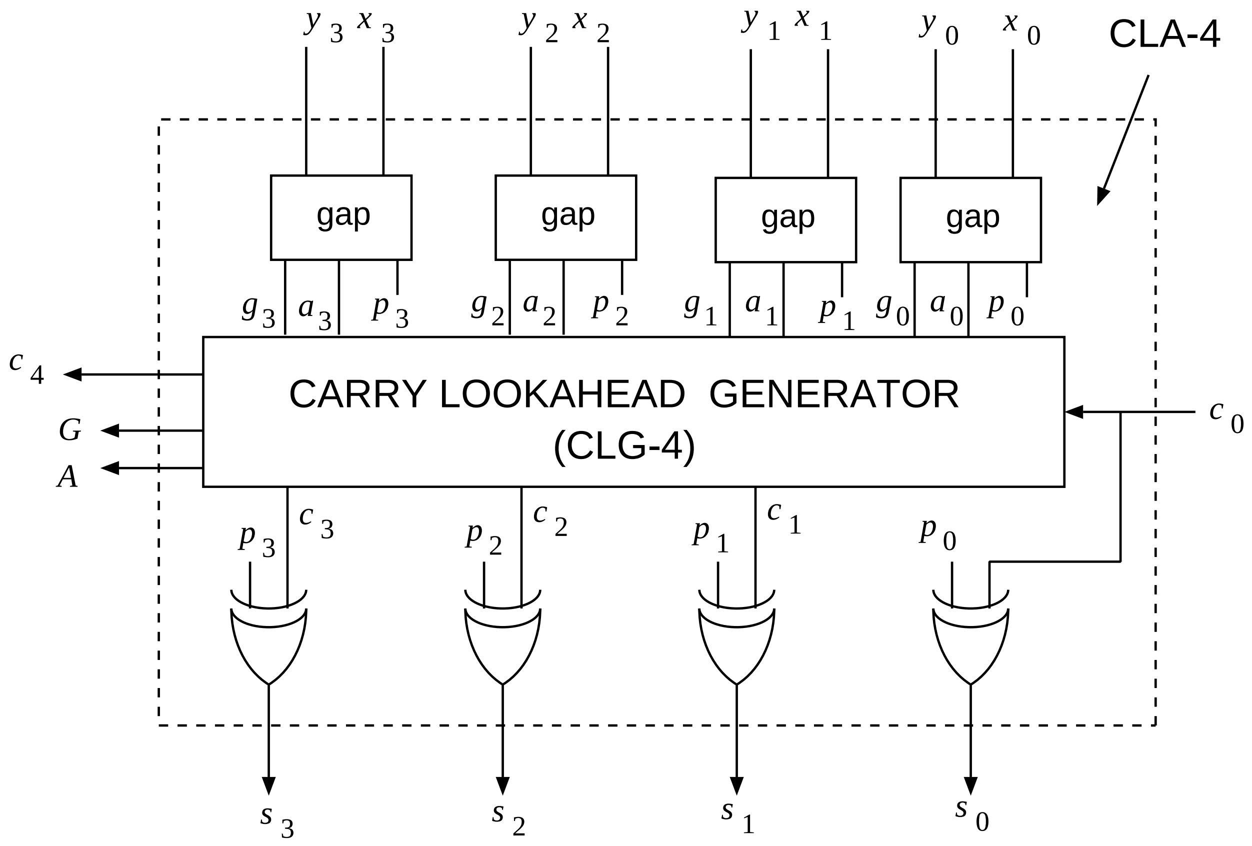


Figure 2.13: Carry-lookahead adder module ($m = 4$).

CARRY-LOOKAHEAD GENERATOR

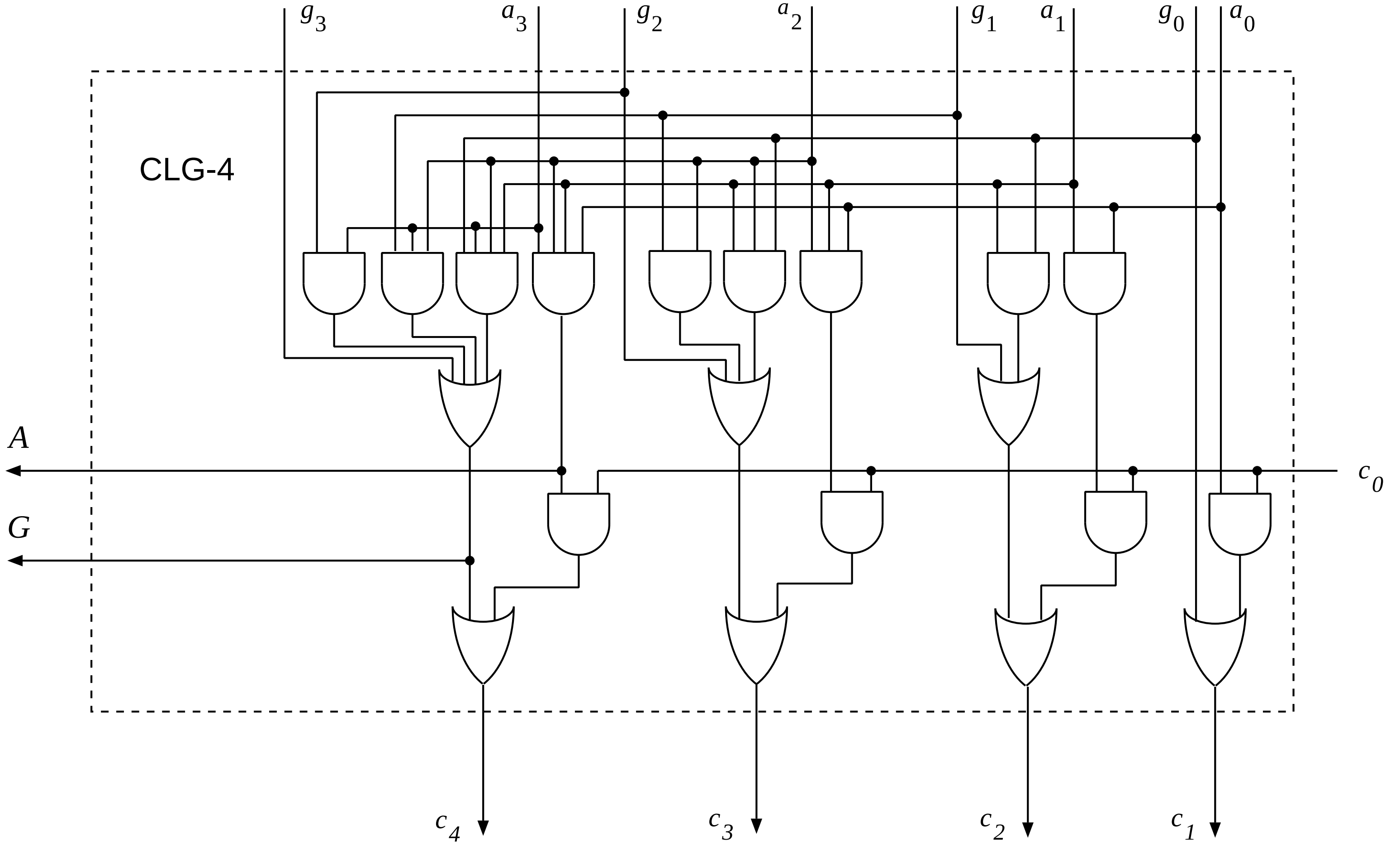


Figure 2.14: 4-bit carry-lookahead generator CLG-4.

TWO-LEVEL CARRY-LOOKAHEAD ADDER

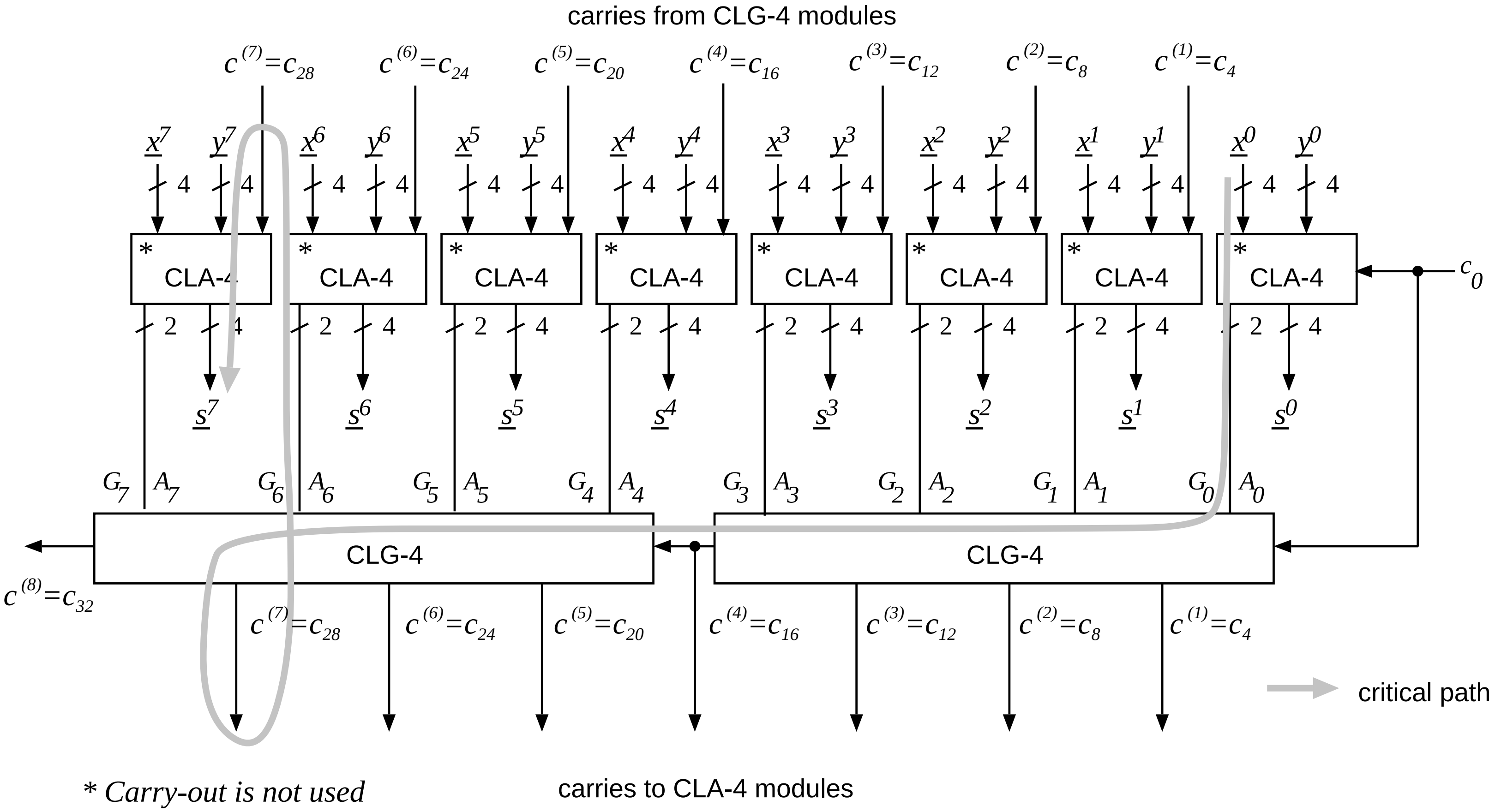
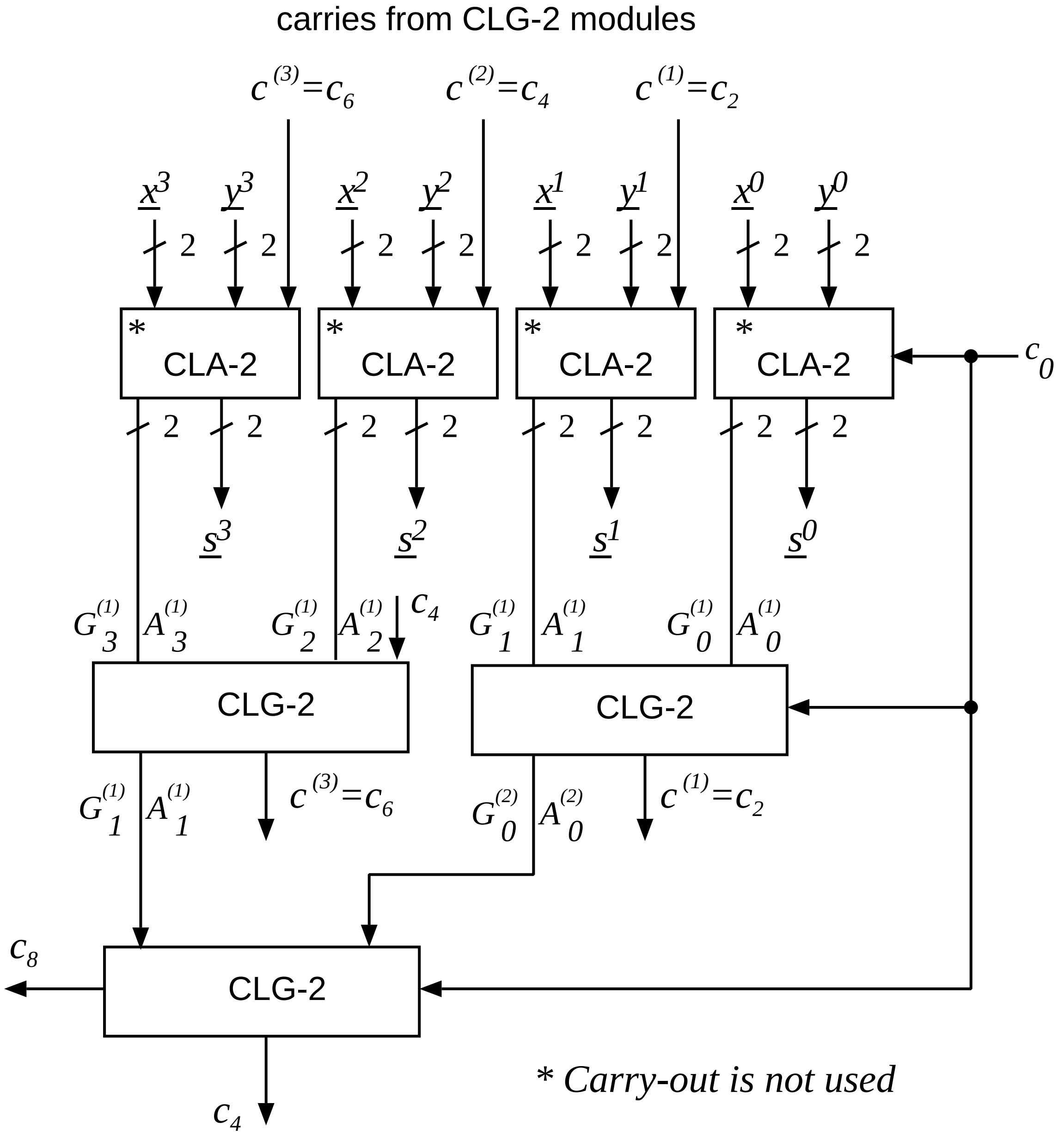


Figure 2.15: Two-level carry-lookahead adder ($n = 32$)



** Carry-out is not used*

Figure 2.16: Three-level carry-lookahead adder ($n = 8$, $m = 2$).

Prefix adders

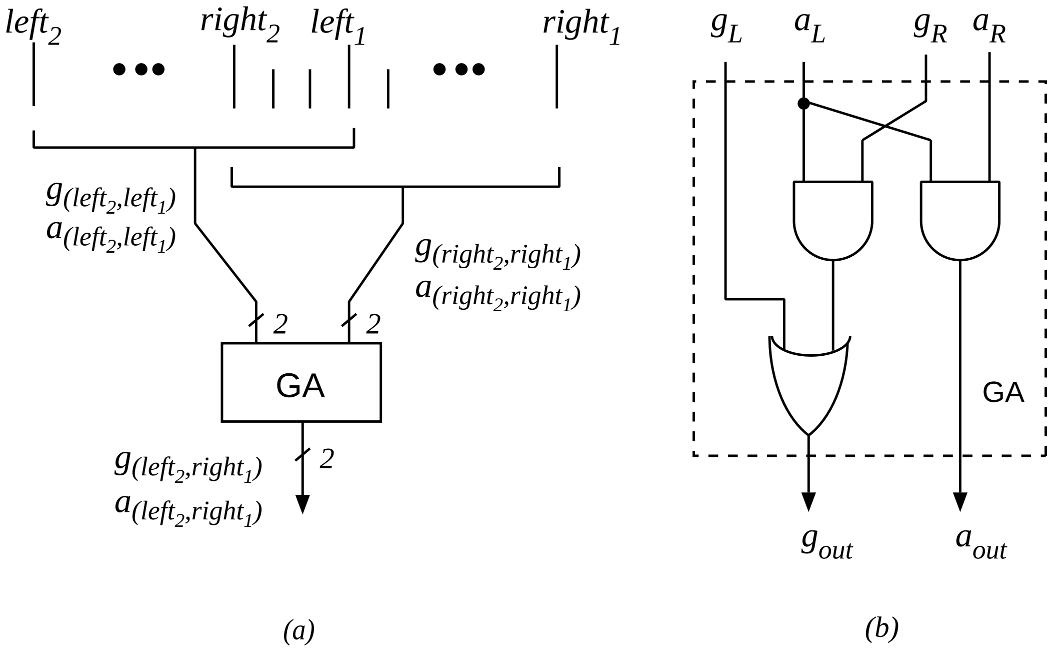


Figure 2.17: Composition of spans in computing (g, a) signals.

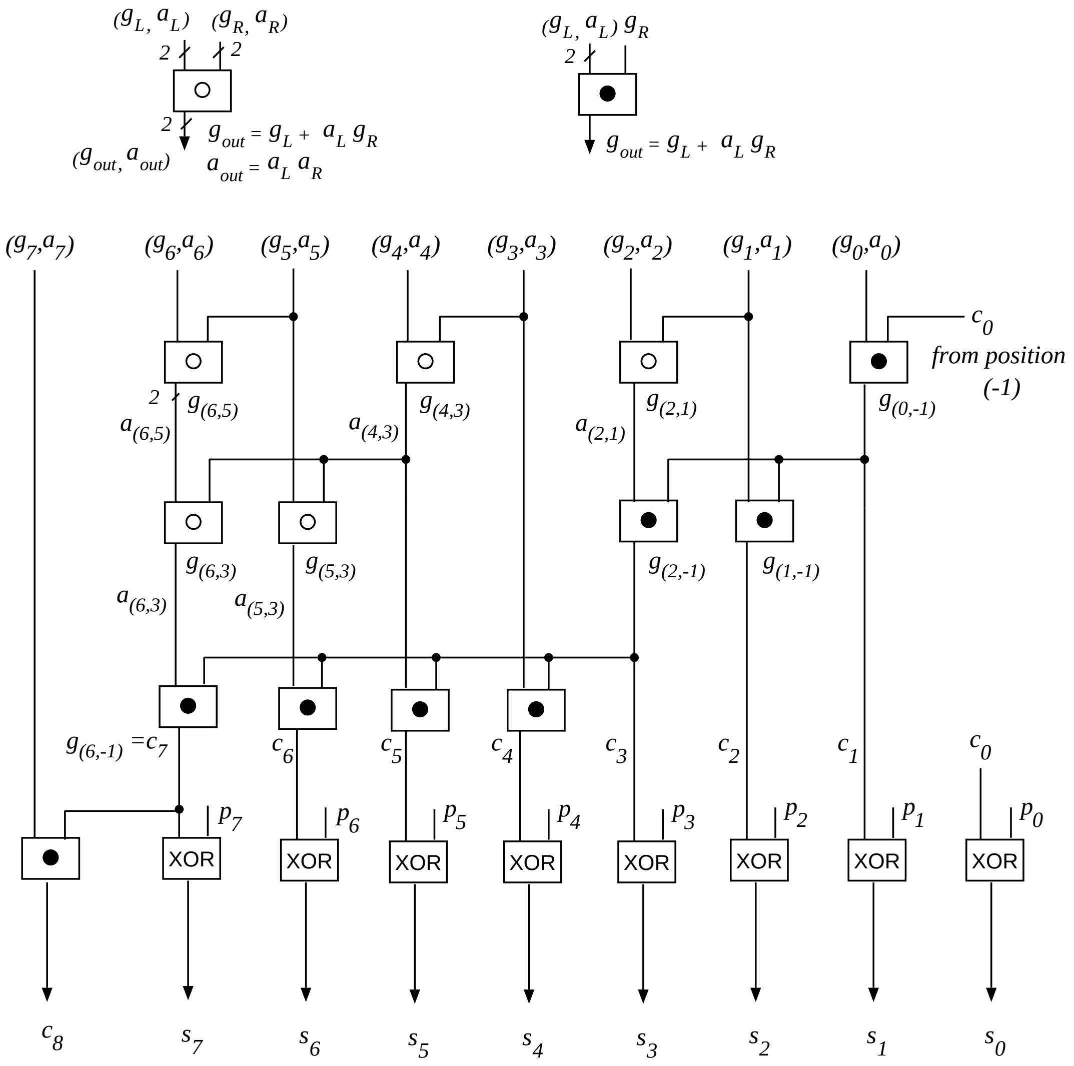


Figure 2.18: 8-bit prefix adder. (Modules to obtain p_i, g_i, and a_i signals not shown.)

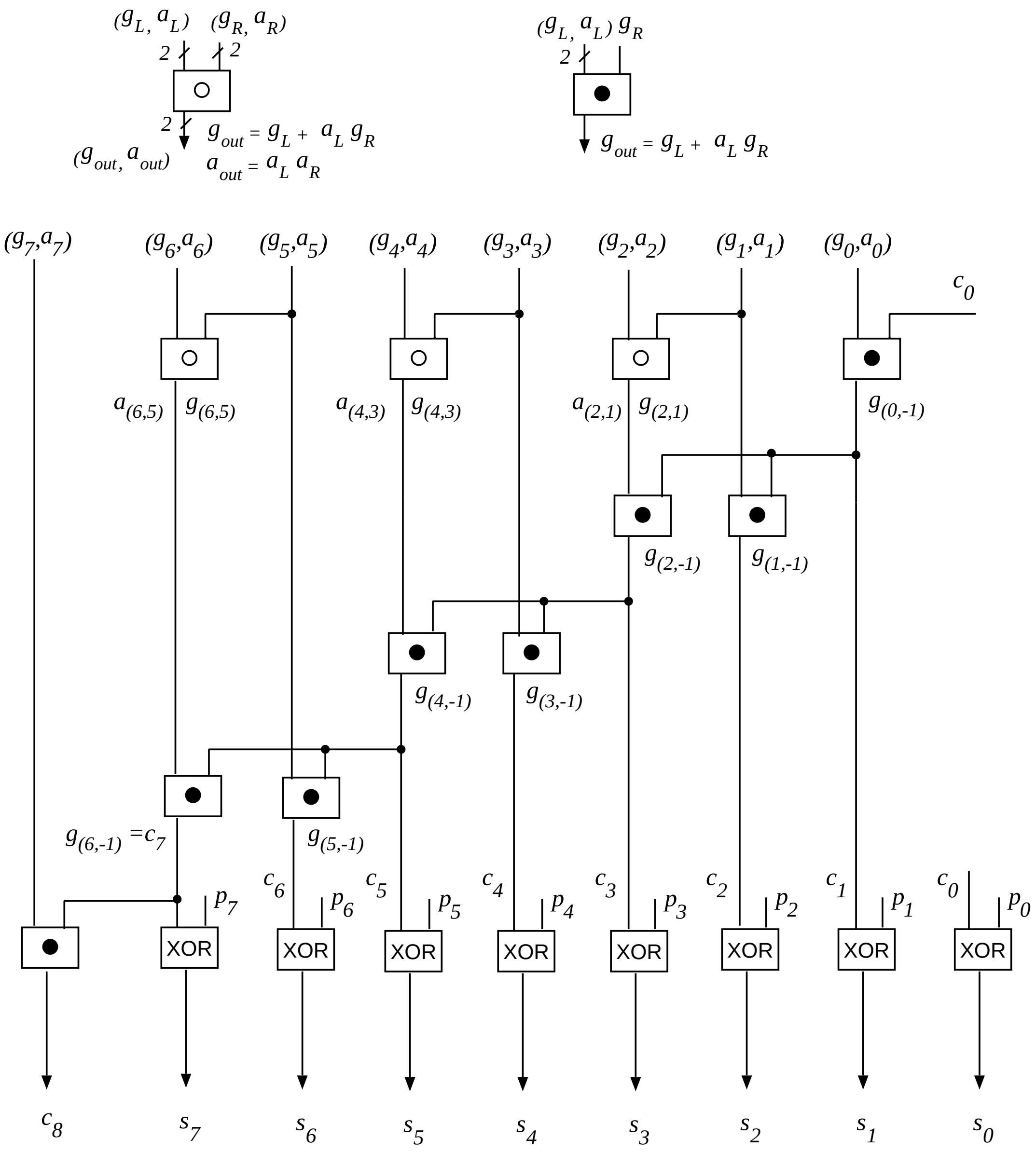
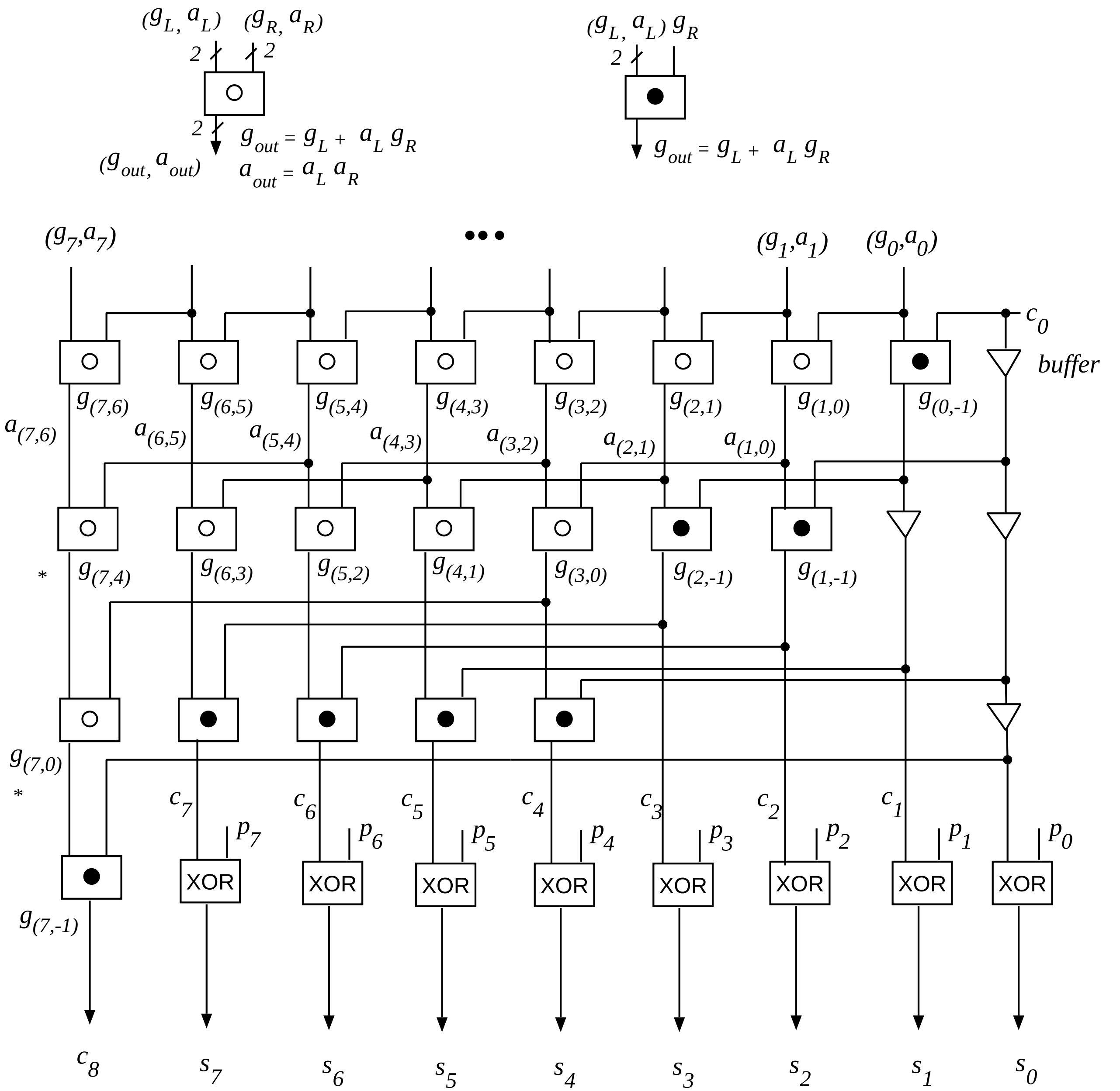


Figure 2.19: 8-bit prefix adder with maximum fanout of three and five levels. (Modules to obtain p_i, g_i, and a_i signals not shown.)



* $a_{(j,k)}$ not labeled

Figure 2.20: 8-bit prefix adder with minimum number of levels and fanout of two.(Modules to obtain p_i, g_i, and a_i signals not shown.)

CONDITIONAL ADDER (COND ADDER)

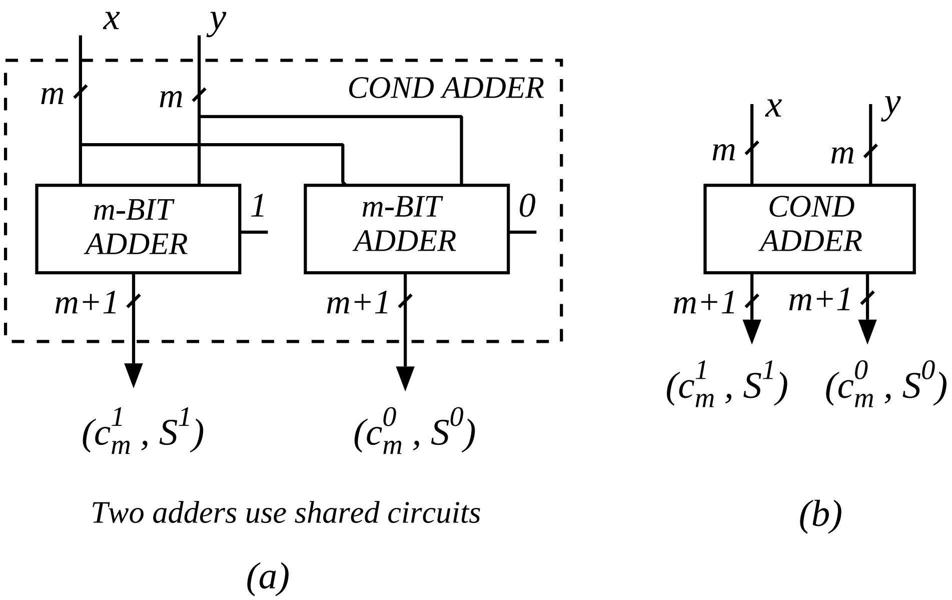


Figure 2.21: (a) Obtaining conditional outputs. (b) Combined conditi onal adder.

CARRY-SELECT ADDER

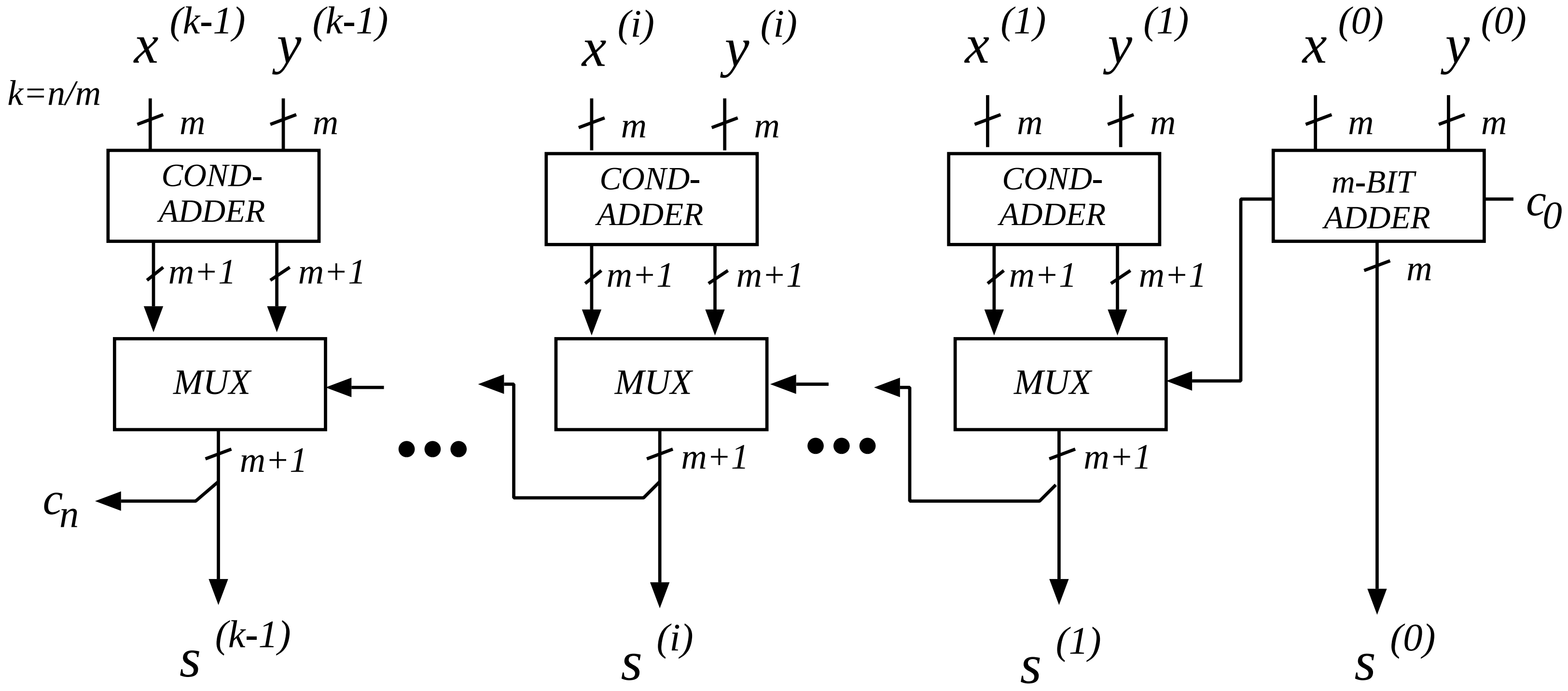


Figure 2.22: Carry-select adder.

CONDITIONAL-SUM ADDER

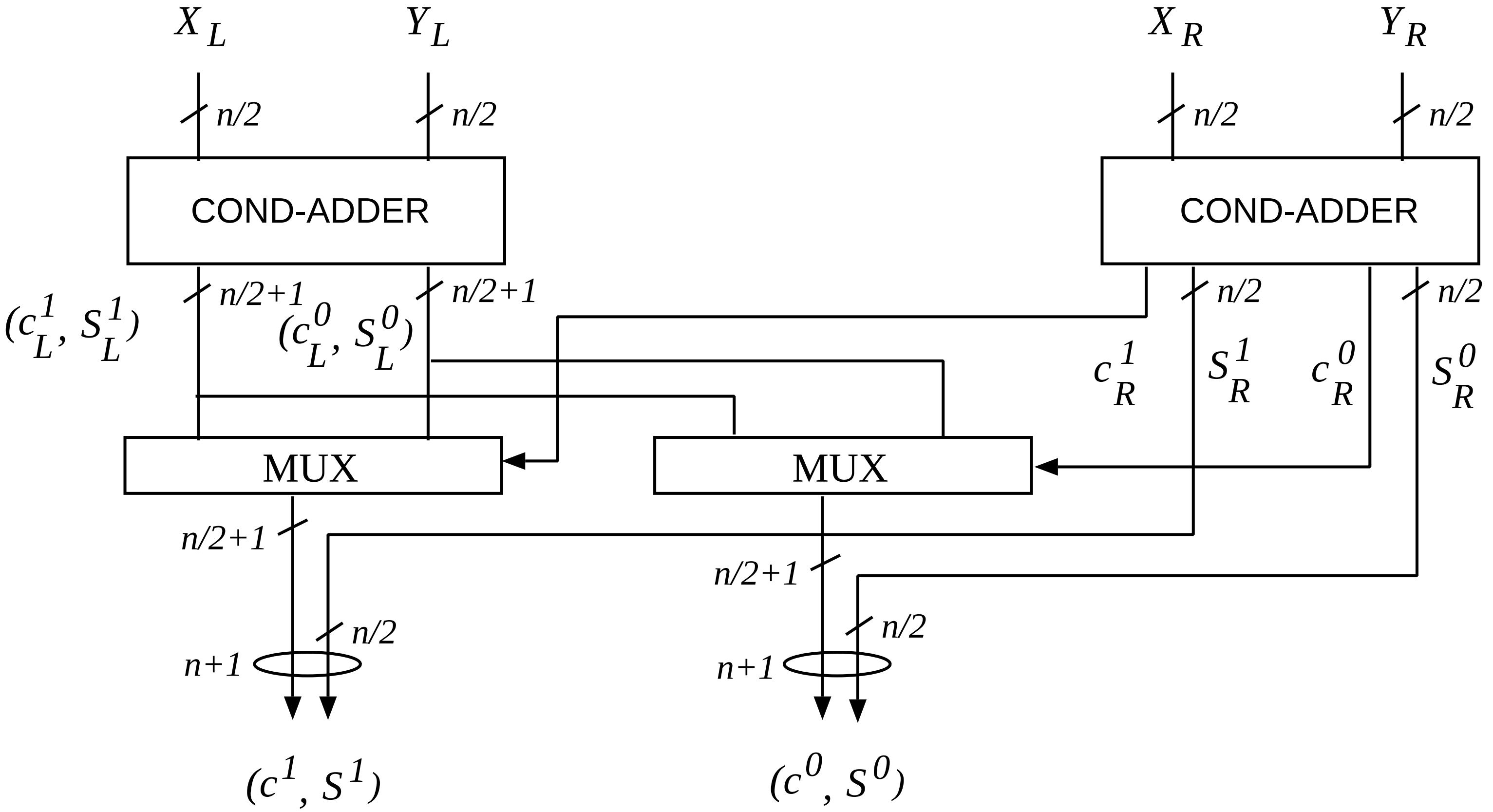


Figure 2.23: Doubling the number of bits of the conditional sum.

16-bit CONDITIONAL-SUM ADDER

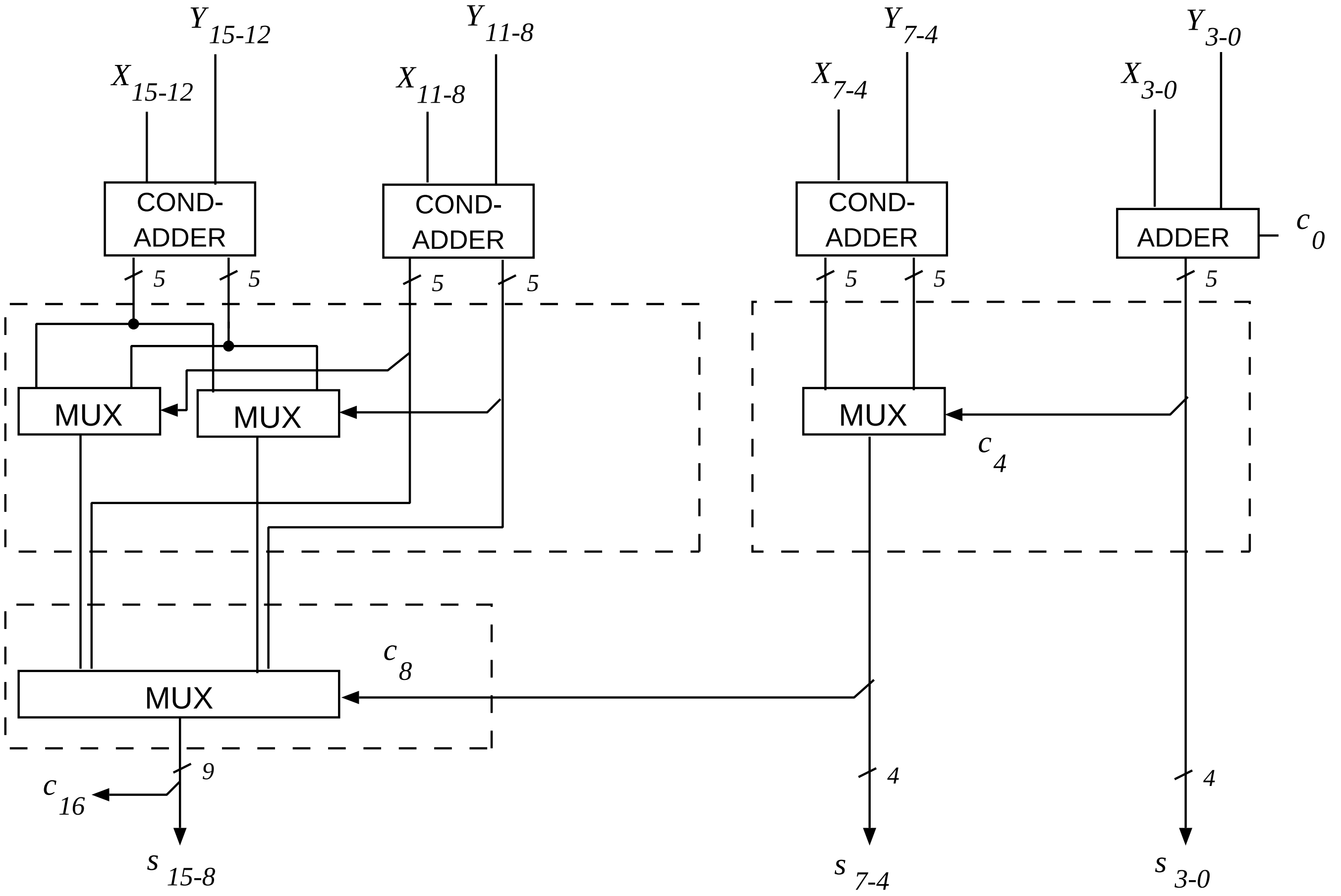


Figure 2.24: 16-bit conditional-sum adder ($m = 4$).

TEMPLATE

s_3^0	s_2^0	s_1^0	s_0^0	Step 2
c_4^0		c_2^0		
s_3^1	s_2^1	s_1^1	s_0^1	
c_4^1		c_2^1		
s_3^0	s_2^0	s_1^0	s_0^0	Step 3
c_4^0				
s_3^1	s_2^1	s_1^1	s_0^1	
c_4^1				

Figure 2.25: Conditional-sum addition for eight bits with $m = 1$: (a) Template. (b) E xample.

	7	6	5	4	3	2	1	0	
x	0	1	0	1	1	0	1	1	$c_0 = 0$
y	0	1	0	1	0	1	0	1	
s^0	0	0	0	0	1	1	1	0	
c^0	0	**1**	0	**1**	0	**0**	0	**1**	
									Step 1
s^1	1	1	1	1	0	0	0		
c^1	0	**1**	0	**1**	1	**1**	1		
s^0	1	0	1	0	1	1	0	0	
c^0	0		**0**		0		**1**		
									Step 2
s^1	1	1	1	1	0	0			
c^1	0		**0**		1				
s^0	1	0	1	0	0	0	0	0	
c^0	0				**1**				
									Step 3
s^1	1	0	1	1					
c^1	0								
s	1	0	1	1	0	0	0	0	

PIPELINED ADDERS

Increase throughput

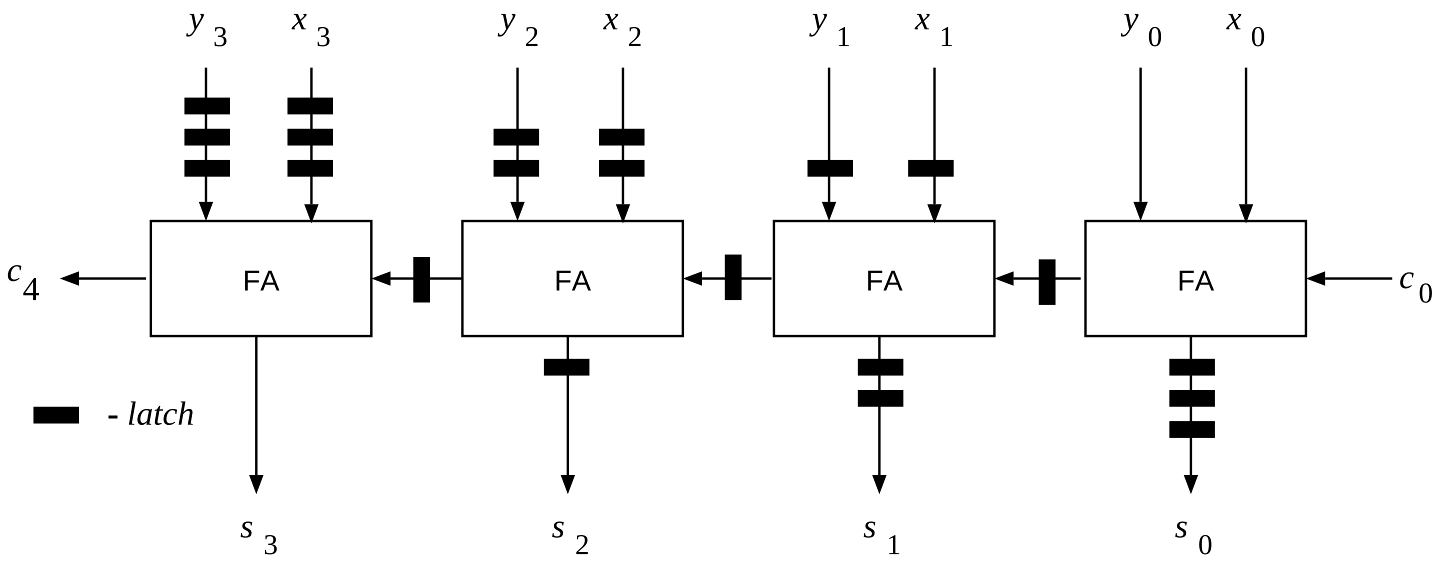
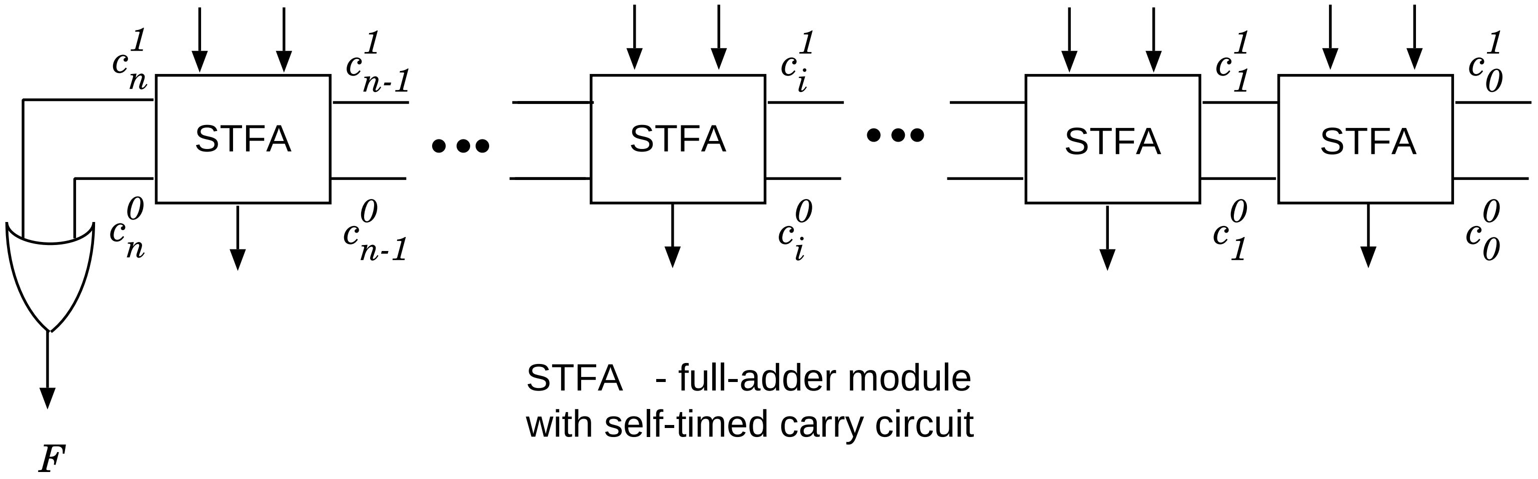


Figure 2.26: Pipelined carry-ripple adder (for group size of 1 and $n = 4$)

VARIABLE-TIME ADDER: Type 1



STFA - full-adder module
with self-timed carry circuit

Figure 2.27: Variable-time adder: Type 1.

Two carry signals:

$$c_i^0 \ \ zero\ carry \quad c_i^1 \ \ one\ carry$$

with coding:

c_i^0	c_i^1	c_i
0	0	not determined (yet)
0	1	1
1	0	0
1	1	does not occur

VARIABLE-TIME ADDER: Type 1 cont.

STFA module expressions:

$$\begin{aligned} c_{i+1}^0 &= k_i(c_i^0 + c_i^1) + p_i c_i^0 = k_i c_i^1 + (p_i + k_i)c_i^0 \\ c_{i+1}^1 &= g_i(c_i^0 + c_i^1) + p_i c_i^1 = g_i c_i^0 + (p_i + g_i)c_i^1 \end{aligned}$$

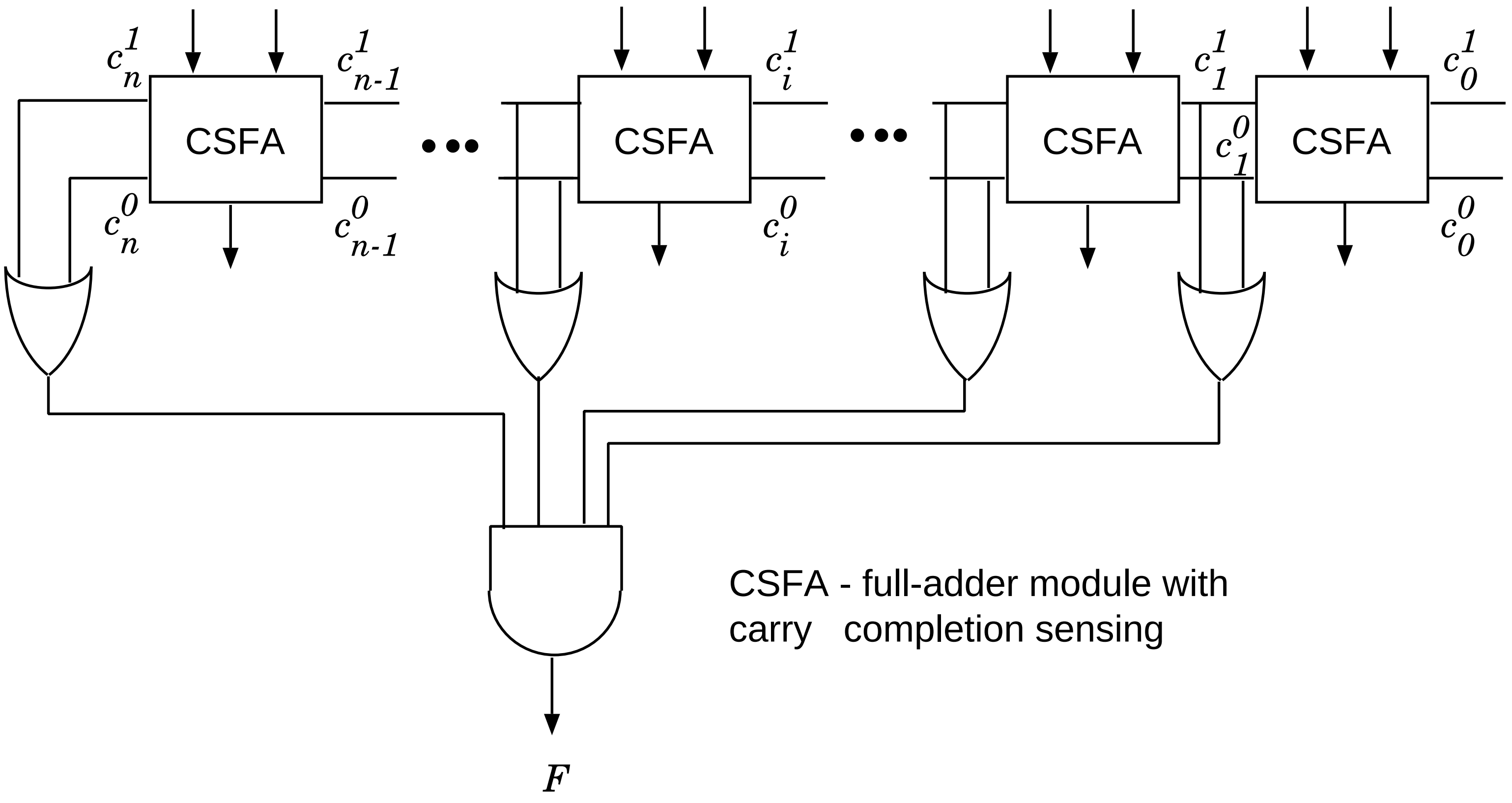
$$s_i = p_i \oplus c_i^1$$

$$k_i = x_i' y_i', \quad g_i = x_i y_i, \quad p_i = x_i \oplus y_i$$

Addition time: based on $actual$ delays, not worst-case

$$T_{var-1} = \sum_{i=0}^{n-1} t_{c,i}$$

VARIABLE-TIME ADDER: Type 2



CSFA - full-adder module with carry completion sensing

Figure 2.28: Variable-time adder: Type 2.

Carry chains initiated simultaneously

CSFA module expressions:

$$c^0_{i+1} = k_i + p_i c^0_i, \quad c^1_{i+1} = g_i + p_i c^1_i$$

Example

X	0	1	1	0	0	0	1	1	1	0	0	1	1	0	1	0	
Y	1	0	1	0	1	1	0	0	1	1	1	0	0	1	1	0	
+	a	a	a	b	c	c	c	c	c	d	d	d	d	d	d	e	Prop.chains

Completion signal:

$$F = \prod_{i=0}^{n-1} (c_i^0 + c_i^1)$$

Addition time: proportional to $log_2(n)$

2's COMPLEMENT AND 1s' COMPLEMENT ADDERS

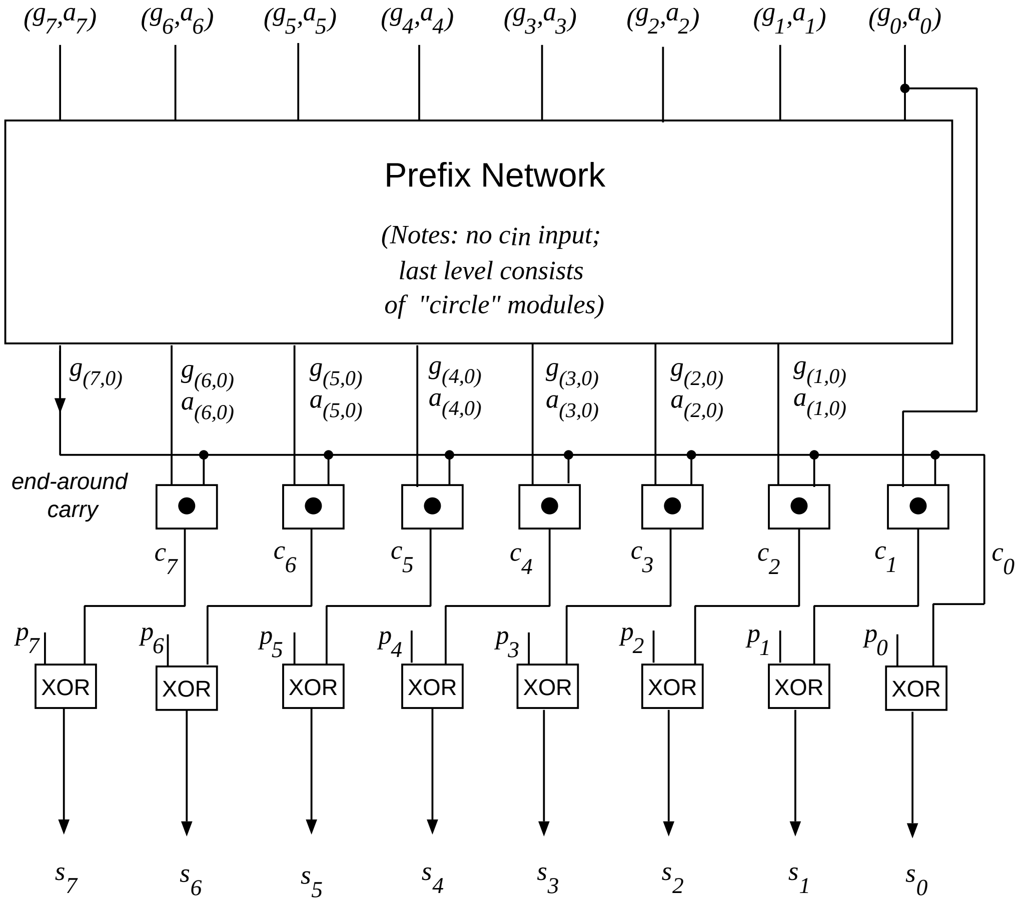


Figure 2.29: Implementing ones' complement adder with prefix network. (Modules to obtain p_i, g_i, and a_i signals not shown.)

ADDERS WITH REDUNDANT DIGIT-SET

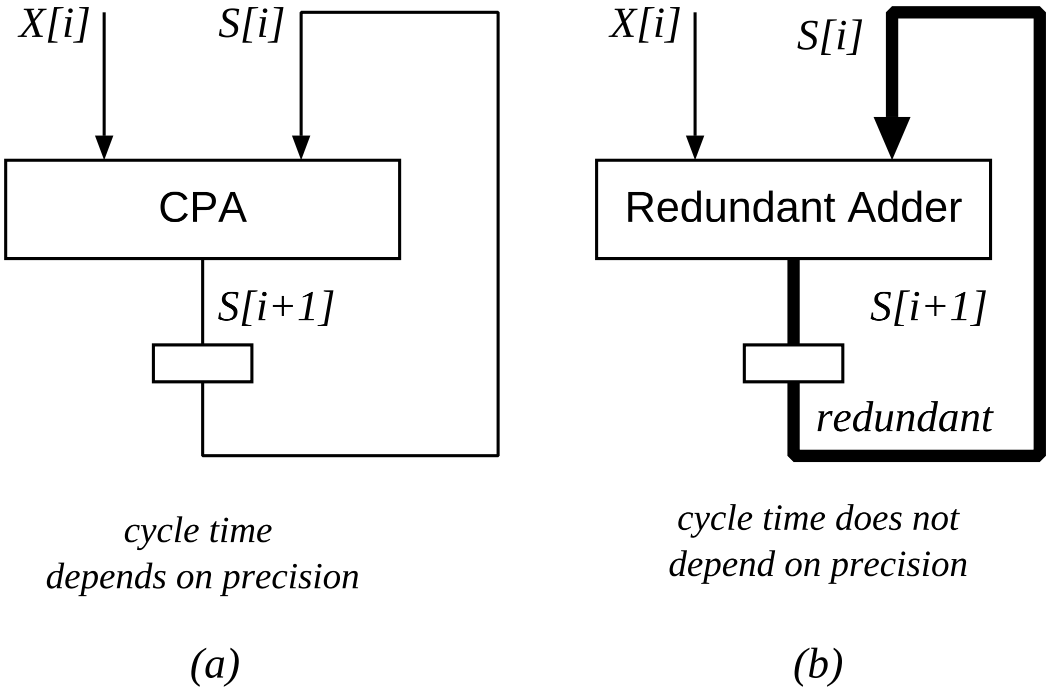


Figure 2.30: Accumulation with (a) non-redundant, and (b) redundant representation of sum.

CARRY-SAVE ADDER

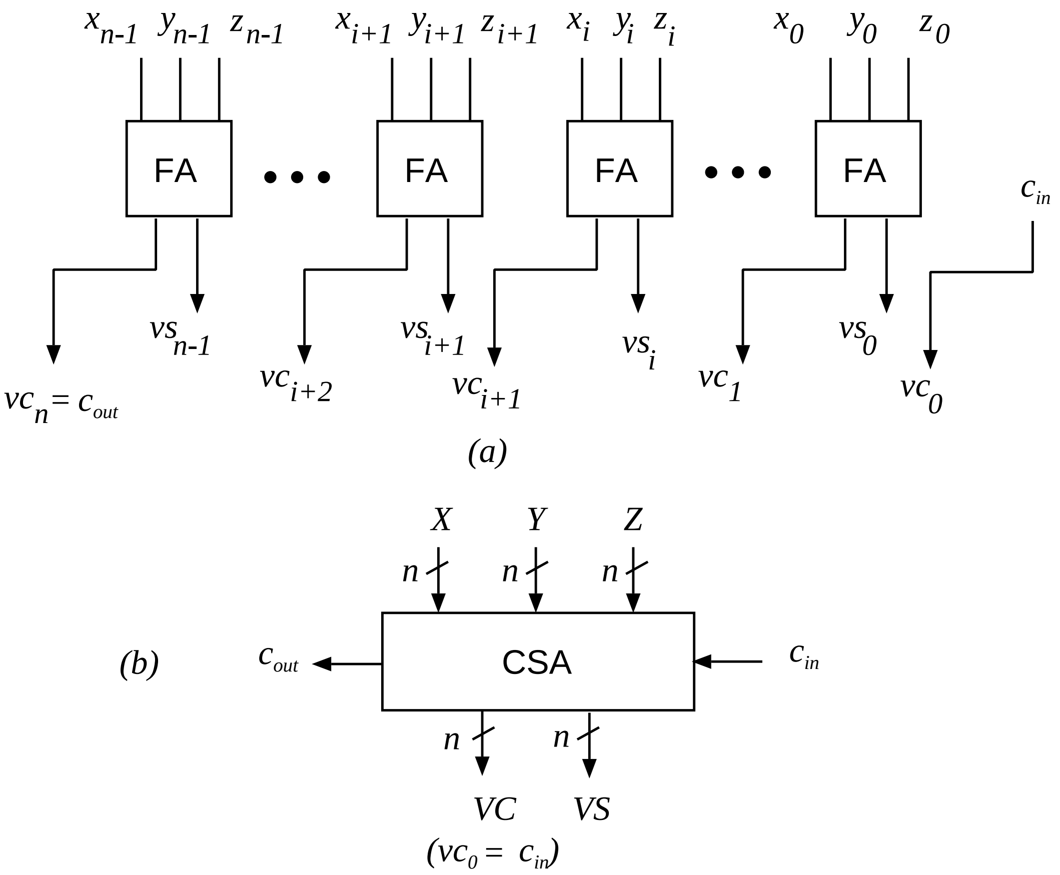


Figure 2.31: Carry-save adder: (a) Bit level. (b) Bit-vector level.

EXAMPLE OF CARRY-SAVE OPERATION

X		0	1	1	1	0	1	0	0
Y		0	0	1	1	1	0	1	1
Z		1	0	1	0	1	0	1	0
VS		1	1	1	0	0	1	0	1
(c_{out}, VC)	0	0	1	1	1	0	1	0	1
digit value	0	1	2	2	1	0	2	0	2

[4:2] ADDER

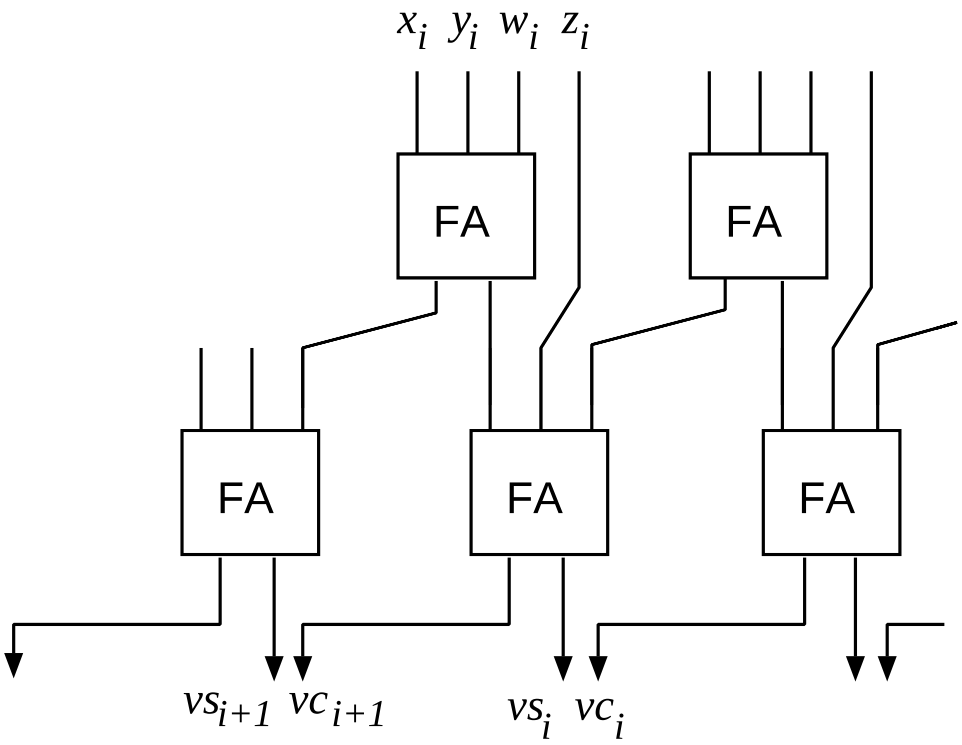


Figure 2.32: [4:2] adder.

HIGH RADIX CARRY-SAVE REPRESENTATION

XS		1	0	1	1	0	1	1	0	0
XC				1			1			0
Y		0	1	0	0	0	1	1	1	1
VS		0	0	0	1	1	1	0	1	1
$(c_{out}$,VC)	1			0			1			0

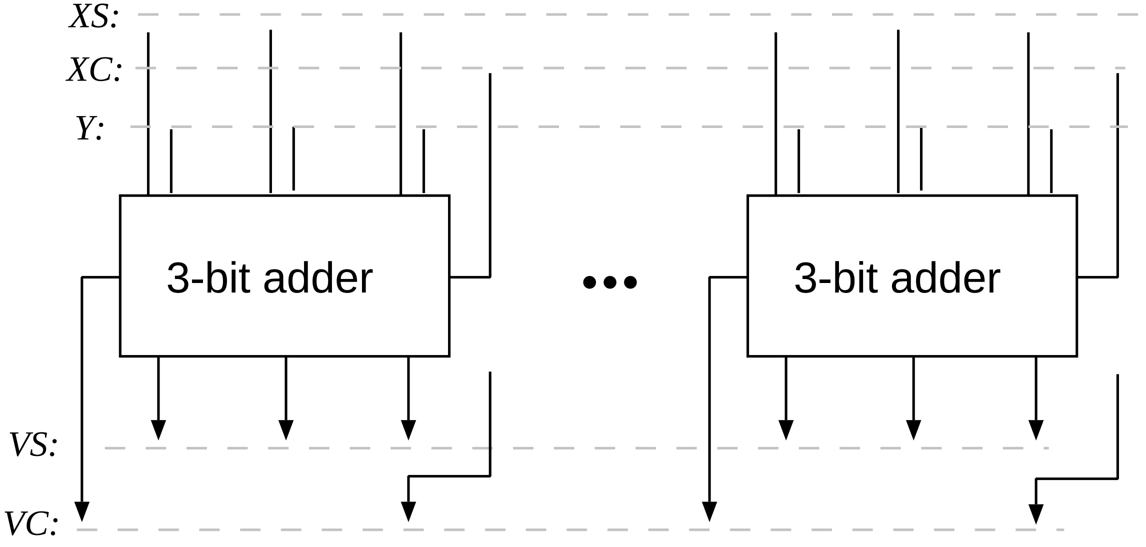


Figure 2.33: Radix-8 carry-save adder.

SIGNED-DIGIT ADDITION

- Uses signed-digit representation (redundant)

$$x = \sum_{0}^{n-1} x_i r^i$$

with digit set

$$D = \{-a, \ldots, -1, 0, 1, \ldots, a\}$$

- Limits carry propagation to next position
- Addition algorithm:

$$\begin{array}{ll} \text{Step 1:} & x + y = w + t \\ & x_i + y_i = w_i + rt_{i+1} \\ & \\ \text{Step 2:} & s = w + t \\ & s_i = w_i + t_i \end{array}$$

- No carry produced in Step 2

SD ADDER

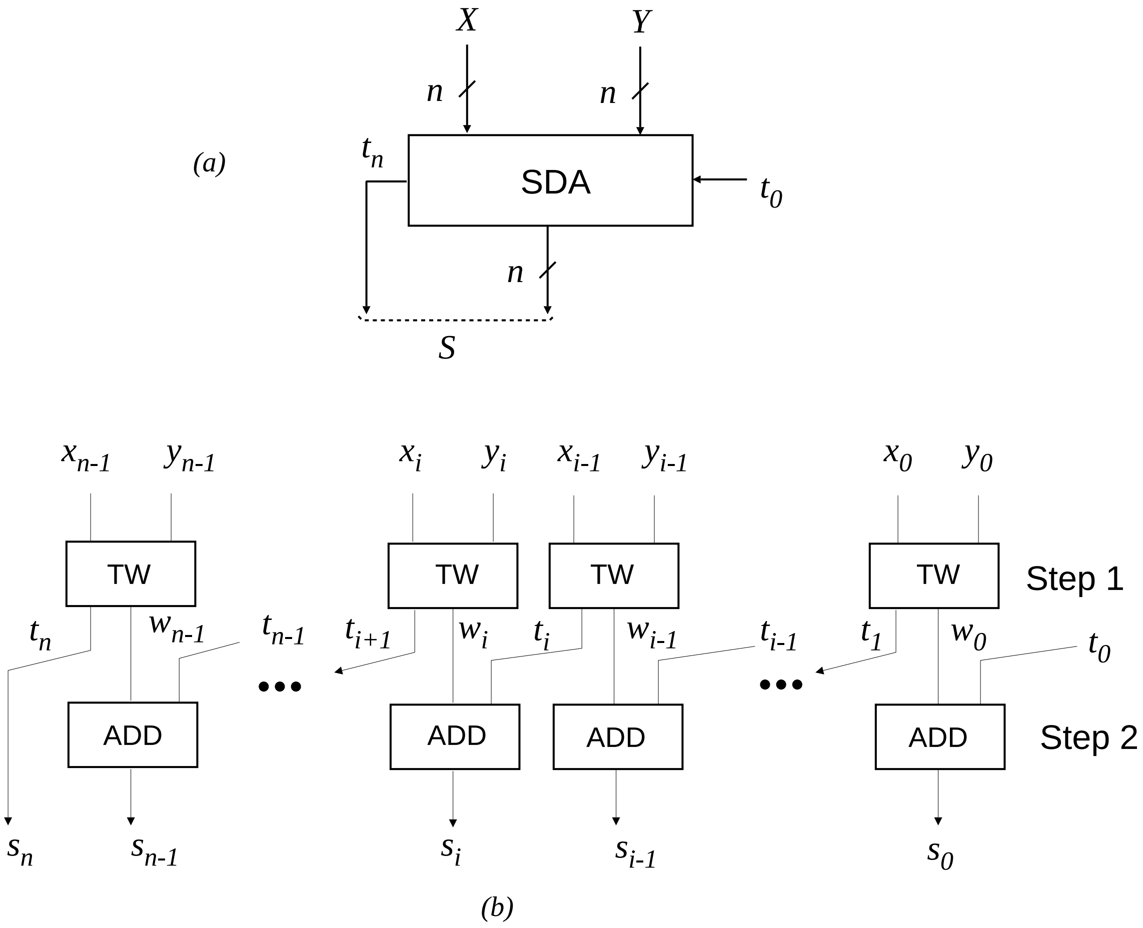


Figure 2.34: Signed-digit addition.

CASES CONSIDERED

Case A : two SD operands; result SD

Step 1:

$$(t_{i+1},\ w_i) = \begin{cases} (0,\ x_i + y_i) & \textbf{if}\ -a+1 \leq x_i + y_i \leq a-1 \\ (1,\ x_i + y_i - r) & \textbf{if}\ x_i + y_i \geq a \\ (-1,\ x_i + y_i + r) & \textbf{if}\ x_i + y_i \leq -a \end{cases}$$

- algorithm modified for $r = 2$

Case B : two conventional operands; result SD

Case C : one conventional, one SD; result SD

SIGNED-BINARY ADDITION: METHOD 1 (DOUBLE RECODING)

RECODING 1:

$$
\begin{aligned}
x_i + y_i = 2h_{i+1} + z_i \quad &\in \{-2,-1,0,1,2\} \\
h_i \in \{0,1\}, z_i &\in \{-2,-1,0\} \\
q_i = z_i + h_i \quad &\in \{-2,-1,0,1\}
\end{aligned}
$$

RECODING 2:

$$
\begin{aligned}
q_i = z_i + h_i = 2t_{i+1} + w_i \quad &\in \{-2,-1,0,1\} \\
t_i \in \{-1,0\}, \quad w_i &\in \{0,1\}
\end{aligned}
$$

THE RESULT: $s_i = w_i + t_i \quad \in \{-1,0,1\}$

METHOD 1 SD ADDER

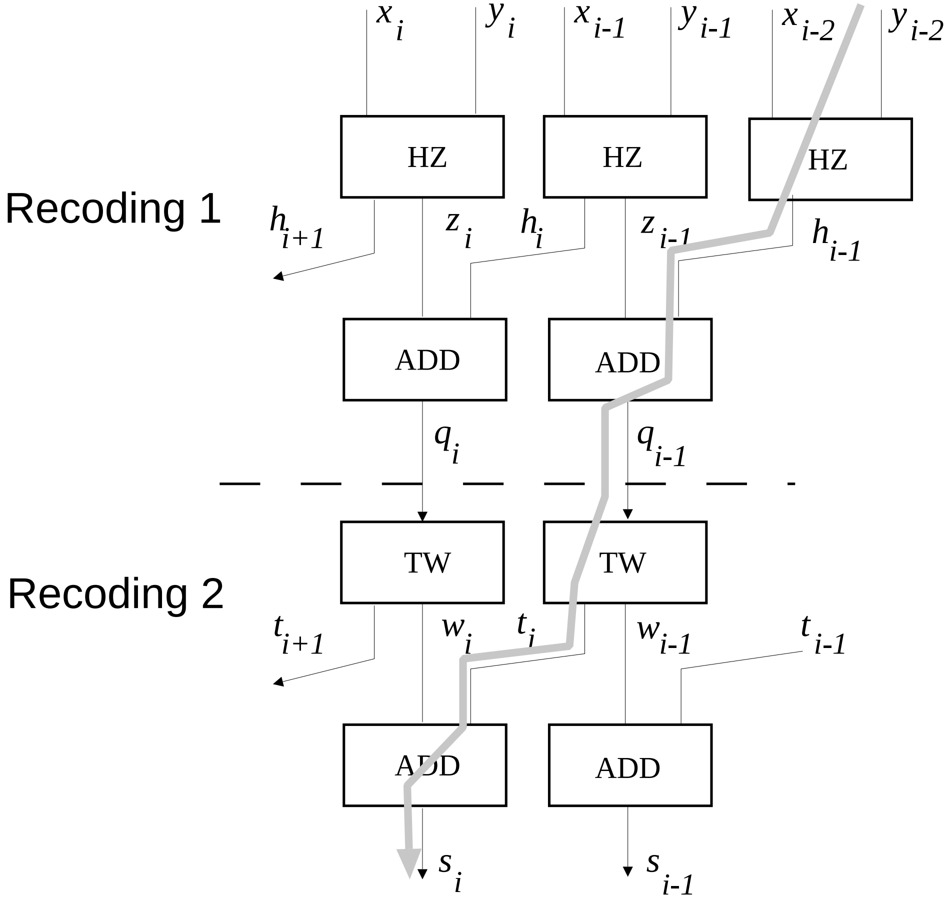


Figure 2.35: Double recoding method for signed-bit addition

SIGNED BINARY ADDITION: METHOD 2 (Using Previous Digit)

$$P_i = \begin{cases} 0 & \text{if } (x_i, y_i) \text{ both nonnegative} \\ & \text{(which implies } t_{i+1} \geq 0) \\ 1 & \text{otherwise } (t_{i+1} \leq 0) \end{cases}$$

$x_i + y_i$	P_{i-1}	t_{i+1}	w_i
2	-	1	0
1	$0(t_i \geq 0)$	1	-1
1	$1(t_i \leq 0)$	0	1
0	-	0	0
-1	$0(t_i \geq 0)$	0	-1
-1	$1(t_i \leq 0)$	-1	1
-2	-	-1	0

METHOD 2 SD ADDER

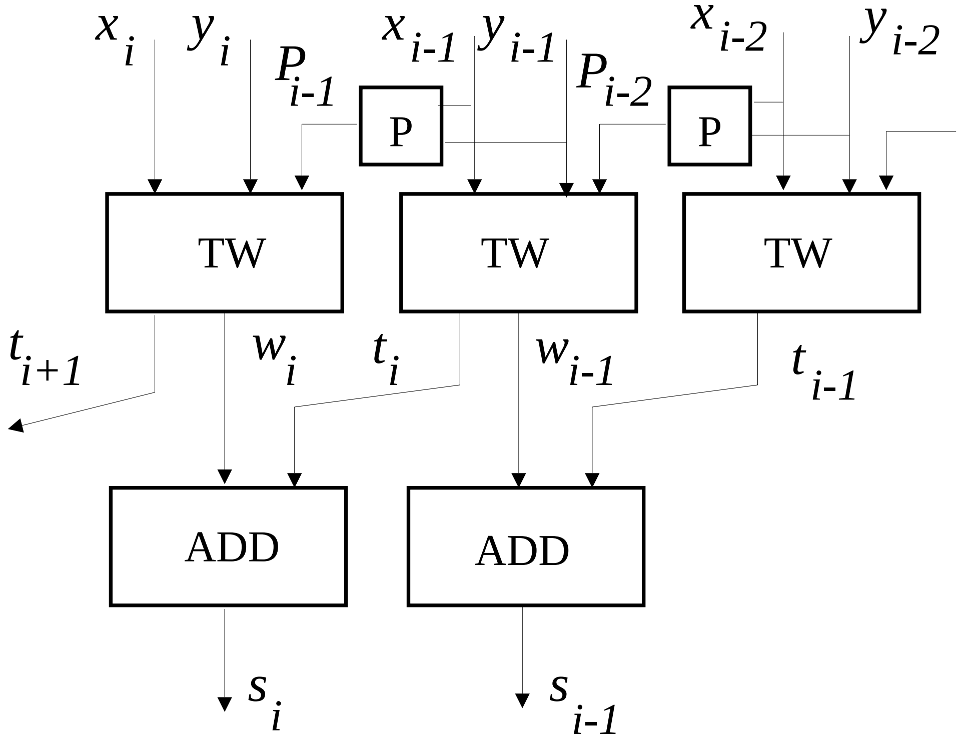


Figure 2.36: Signed-bit addition using the information from previous digit

Example

$$
\begin{array}{c@{\quad}cccccccccc}
X & & 0 & 1 & 1 & 1 & \bar{1} & 1 & 0 & \bar{1} & 1 \\
Y & & 0 & 1 & 1 & 0 & \bar{1} & 0 & 1 & 0 & 1 \\
 & & & & & & & & & & \\
P & & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
 & & & & & & & & & & \\
W & & 0 & 0 & 0 & 1 & 0 & \bar{1} & 1 & \bar{1} & 0 \\
T & 0 & 1 & 1 & 0 & \bar{1} & 1 & 0 & 0 & 1 & 0 \\
 & & & & & & & & & & \\
S & & 1 & 1 & 0 & 0 & 1 & \bar{1} & 1 & 0 & 0
\end{array}
$$

BIT-LEVEL IMPLEMENTATION OF RADIX-2 ALGORITHMS

- Case C: $x_i \in \{0,1\}, \quad y_i, s_i \in \{-1,0,1\}$
- Code: borrow-save $y_i = y_i^+ - y_i^-$, $y_i^+, y_i^- \in \{0,1\}$, sim. for s_i
- $x_i + y_i \in \{-1,0,1,2\}$: recode to (t_{i+1}, w_i), $t_{i+1} \in \{0,1\}$, $w_i \in \{-1,0\}$

$$x_i + y_i^+ - y_i^- = 2t_{i+1} + w_i$$

$x_i + y_i$	-1	0	1	2
w_i	-1	0	-1	0
t_{i+1}	0	0	1	1

SWITCHING FUNCTIONS FOR t_{i+1} AND w_i

x_i	y_i^+	y_i^-	$x_i + y_i$	t_{i+1}	$-w_i$
0	0	0	0	0	0
0	0	1	-1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	2	1	0
1	1	1	1	1	1

$$\begin{aligned} w_i &= (x_i \oplus y_i^+ \oplus (y_i^-)')' \\ t_{i+1} &= x_i y_i^+ + x_i(y_i^-)' + y_i^+(y_i^-)' \end{aligned}$$

$\Longrightarrow$ implemented using a full-adder and inverters (for variables subtracted)

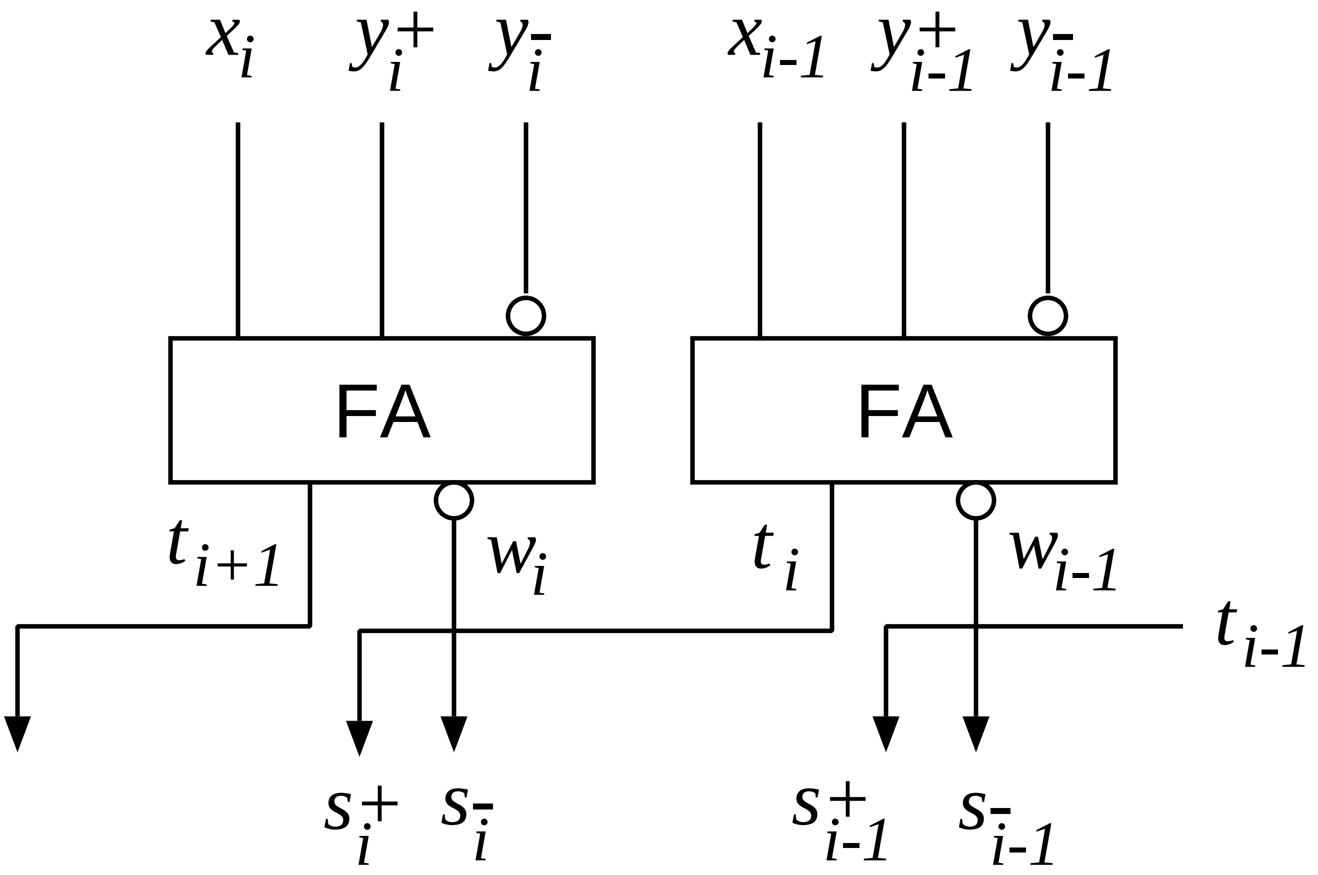


Figure 2.37: Redundant adder: one operand conventional, one operand redundant, result redundant.

BOTH OPERANDS REDUNDANT

- Apply double recoding

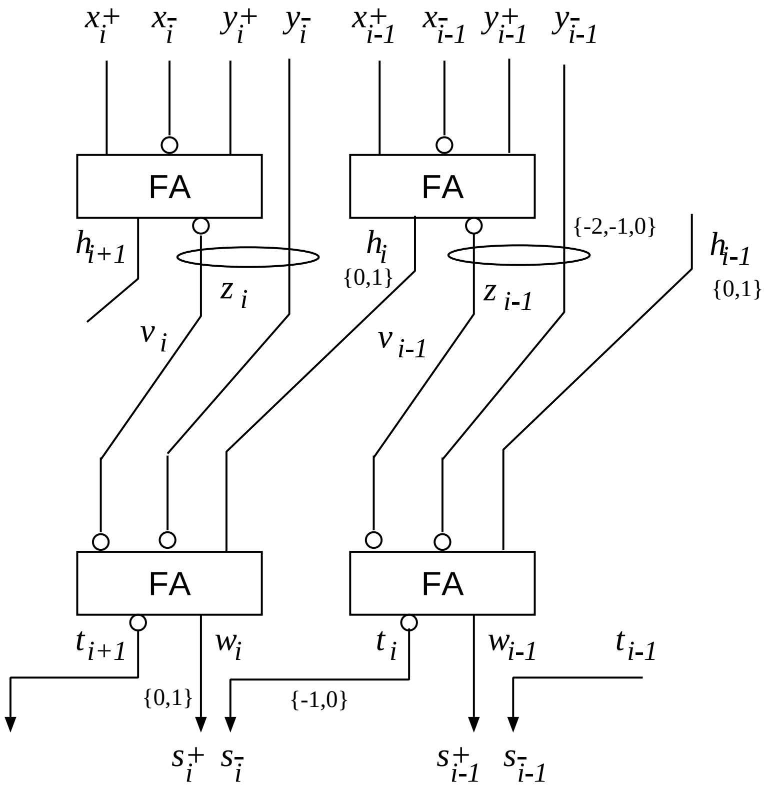


Figure 2.38: Redundant adder: operands and result redundant

SUMMARY

Scheme	Delay proportional to	Area proportional to
Linear structures:		
Carry ripple	n	n
Carry lookahead (one level)	n/m	$(k_m m)(n/m) = k_m n$
Carry select (one level)	n/m	$(k_m m)(n/m) = k_m n$
Carry skip (one level)	$\sqrt{n}$	n
Logarithmic structures:		
Carry lookahead (max. levels)	$2\log_m n$	$(k_m m)(n/m) = k_m n$
Prefix	$\log_m n$	$((k_m m)\log_m n)n$
Conditional sum	$\log_2(n/m)$	$(k_m + \log_2(n/m))n$
Completion signal (avg. delay)	$(\log_2 n)/m$	$k_m m(n/m) = k_m n$
Redundant	$const.$	n

MULTI-OPERAND ADDITION

- Bit-arrays for unsigned and signed operands
  - simplification of sign extension
- Reduction by rows and by columns
  - $[p{:}2]$ modules and $[p{:}2]$ adders for reduction by rows
  - $(p{:}q]$ counters and multicolumn counters for reduction by columns
- Sequential implementation
- Combinational implementation
  - Reduction by rows: arrays of adders (linear arrays, adder trees)
  - Reduction by columns: $(p{:}q]$ counters
  - systematic design method for reduction by columns with (3:2] and (2:2] counters
- Pipelined adder arrays
- Partially combinational implementation

BIT ARRAYS FOR UNSIGNED AND SIGNED OPERANDS

$$\begin{array}{l}
a_0\, a_0\, a_0\, a_0 \,.\, a_1 a_2 \;\ldots\; a_n \\
b_0\, b_0\, b_0\, b_0 \,.\, b_1 b_2 \;\ldots\; b_n \\
c_0\, c_0\, c_0\, c_0 \,.\, c_1 c_2 \;\ldots\; c_n \\
d_0\, d_0\, d_0\, d_0 \,.\, d_1 d_2 \;\ldots\; d_n \\
\underbrace{e_0\, e_0\, e_0}_{\text{sign extension}}\, e_0 \,.\, e_1 e_2 \;\cdots\; e_n
\end{array}$$

Figure 3.1: SIGN-EXTENDED ARRAY FOR $m = 5$.

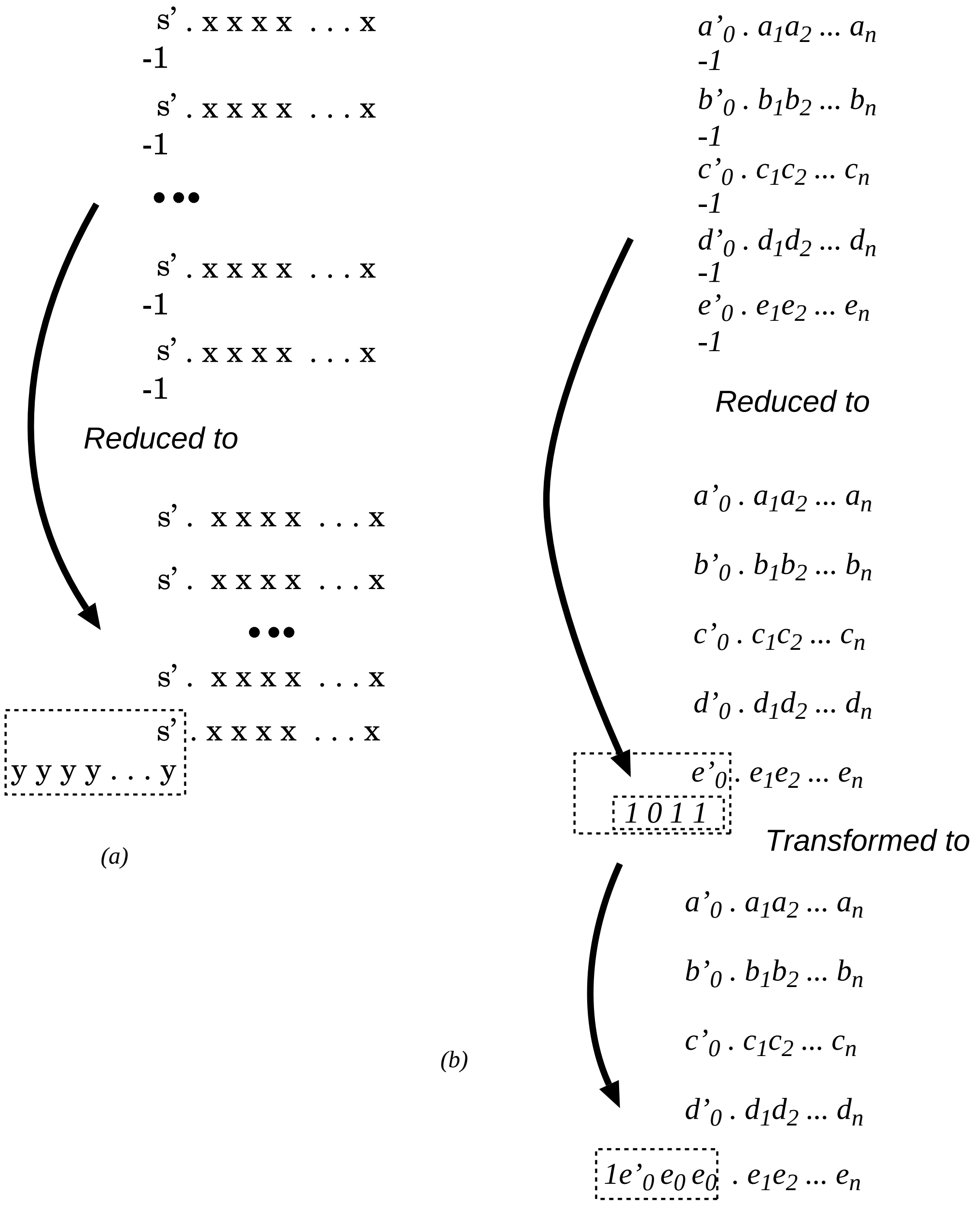


Figure 3.2: SIMPLIFYING SIGN-EXTENSION: (a) GENERAL CASE. (b) EXAMPLE OF SIMPLIFYING ARRAY WITH $m = 5$.

REDUCTION

- By rows
- By columns

[p:2] ADDERS FOR REDUCTION BY ROWS

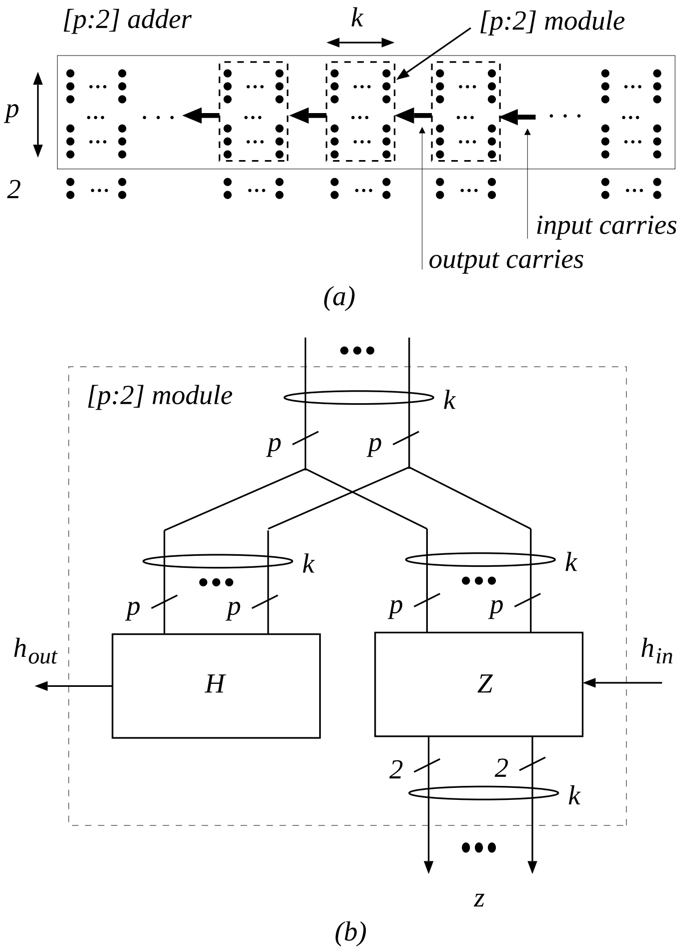


Figure 3.3: A [p:2] adder: (a) Input-output bit-matrix. (b) k-column [p:2] module decomposition.

MODEL OF [p:2] MODULE

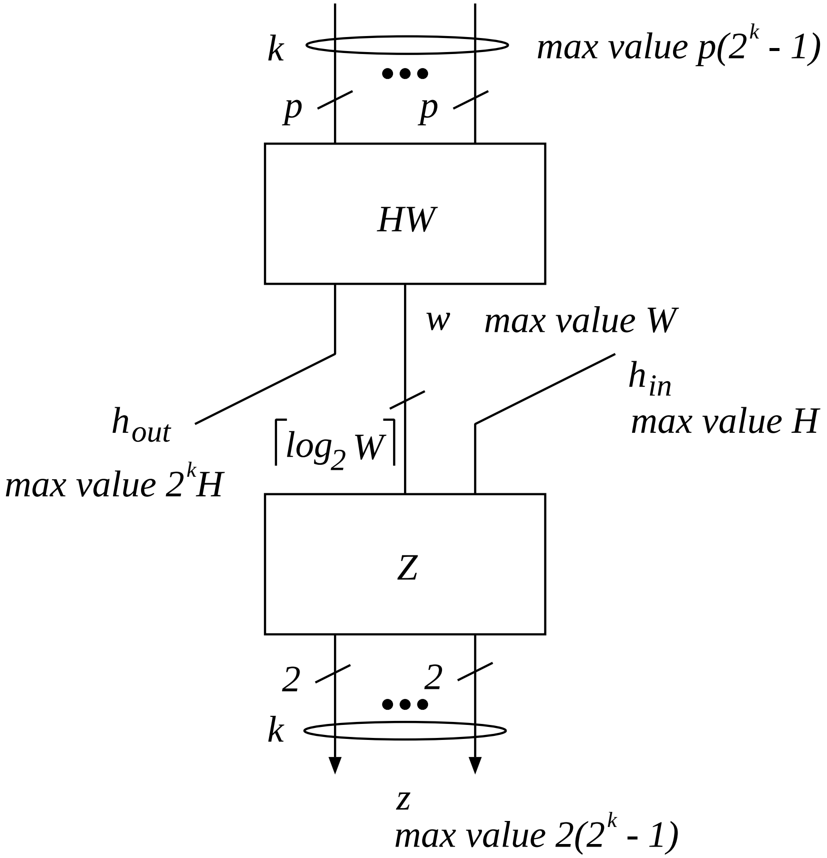
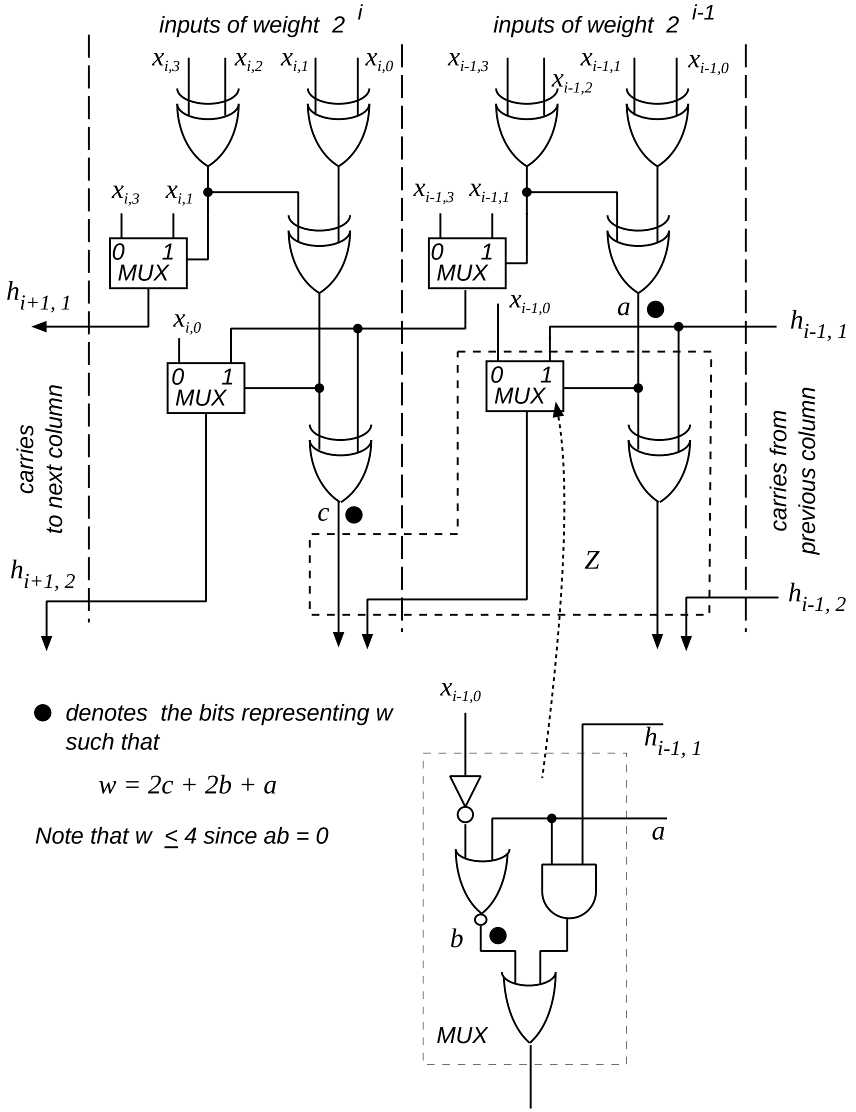


Figure 3.4: A model of a [p:2] module.

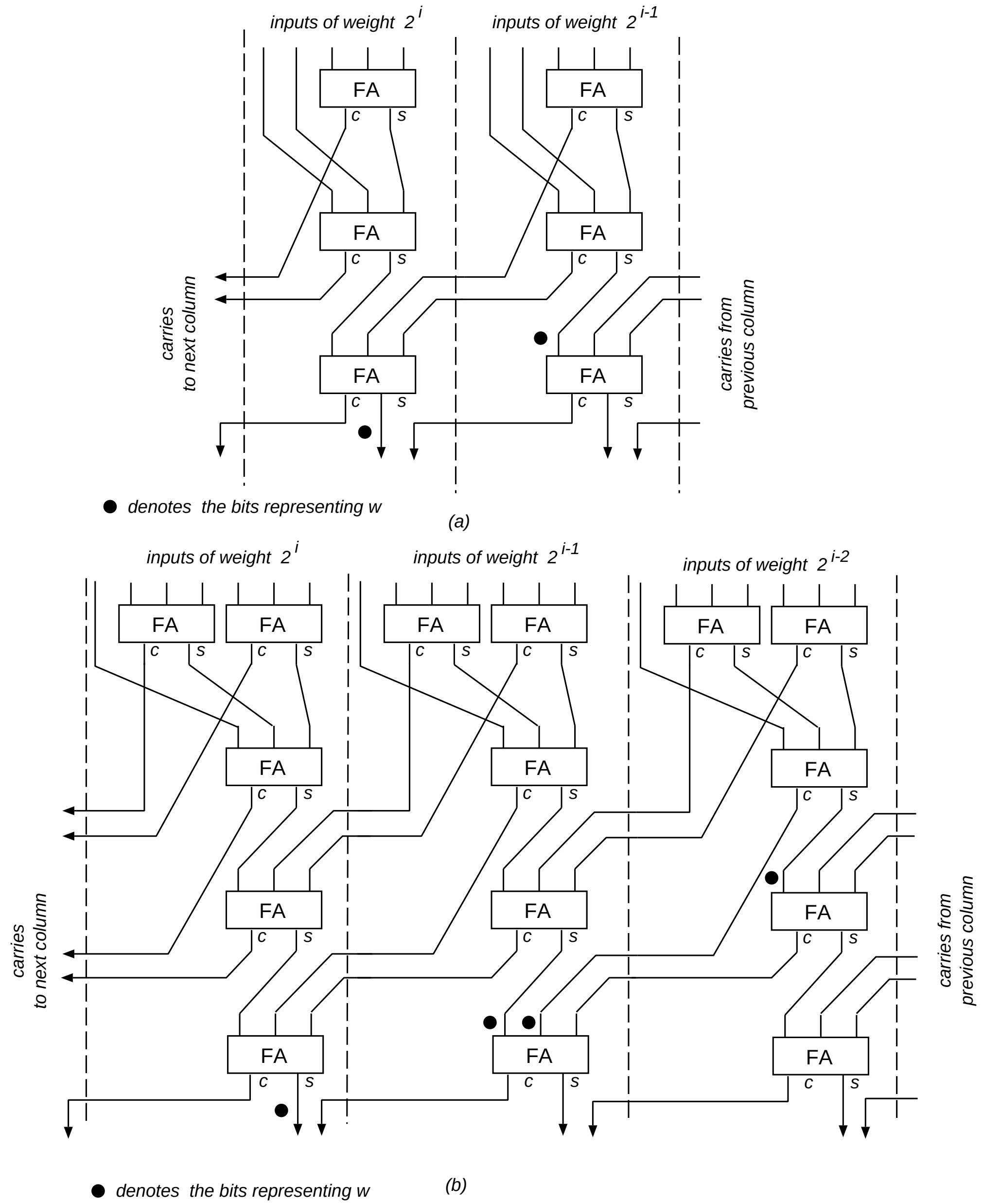


● denotes the bits representing w such that

$$w = 2c + 2b + a$$

Note that $w \leq 4$ since $ab = 0$

Figure 3.5: Gate network implementation of [4:2] module.



(a)

(b)

Figure 3.6: (a) [5:2] module. (b) [7:2] module.

$(p{:}q]$ COUNTERS FOR REDUCTION BY COLUMNS

$$\sum_{i=0}^{p-1} x_i = \sum_{j=0}^{q-1} y_j 2^j$$

$$2^q - 1 \geq p, \text{ i.e., } q = \lceil log_2(p+1) \rceil$$

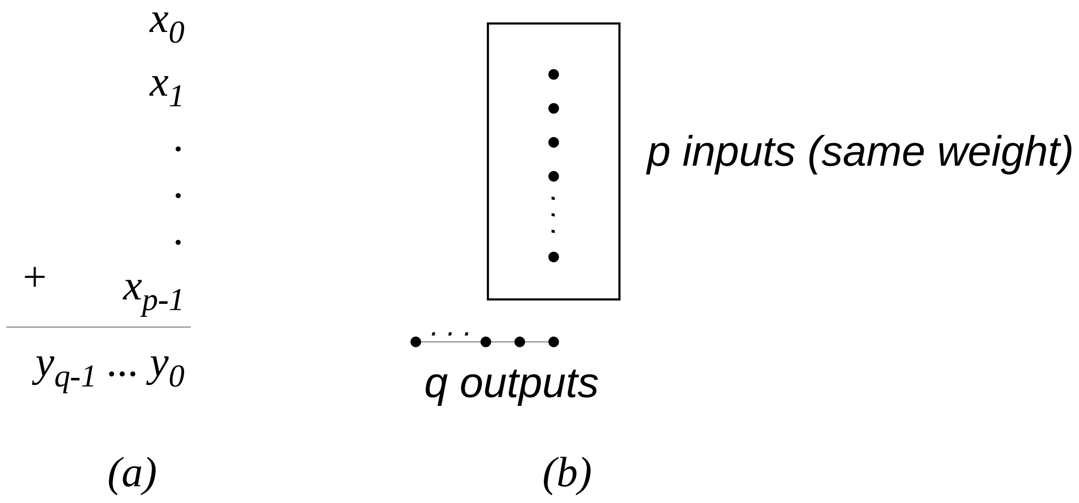


Figure 3.7: (a) $(p{:}q]$ reduction. (b) Counter representation.

IMPLEMENTATION OF $(p{:}q]$ COUNTERS

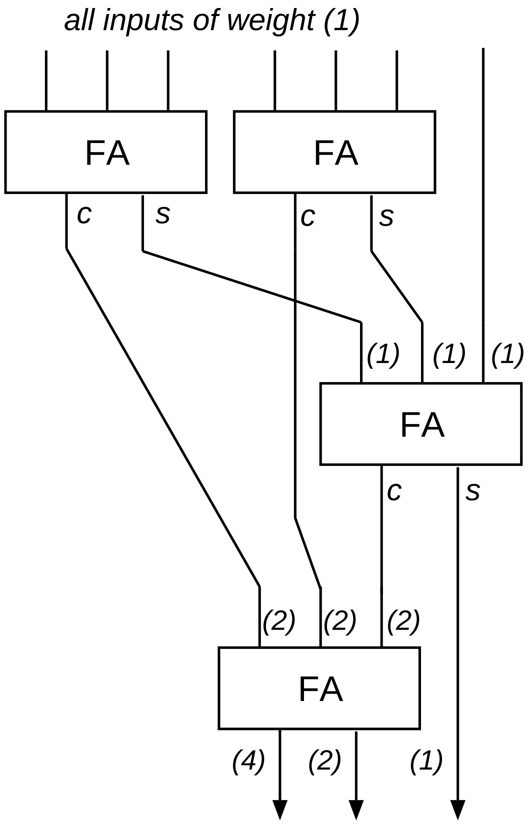


Figure 3.8: Implementation of (7:3] counter by an array of full adders.

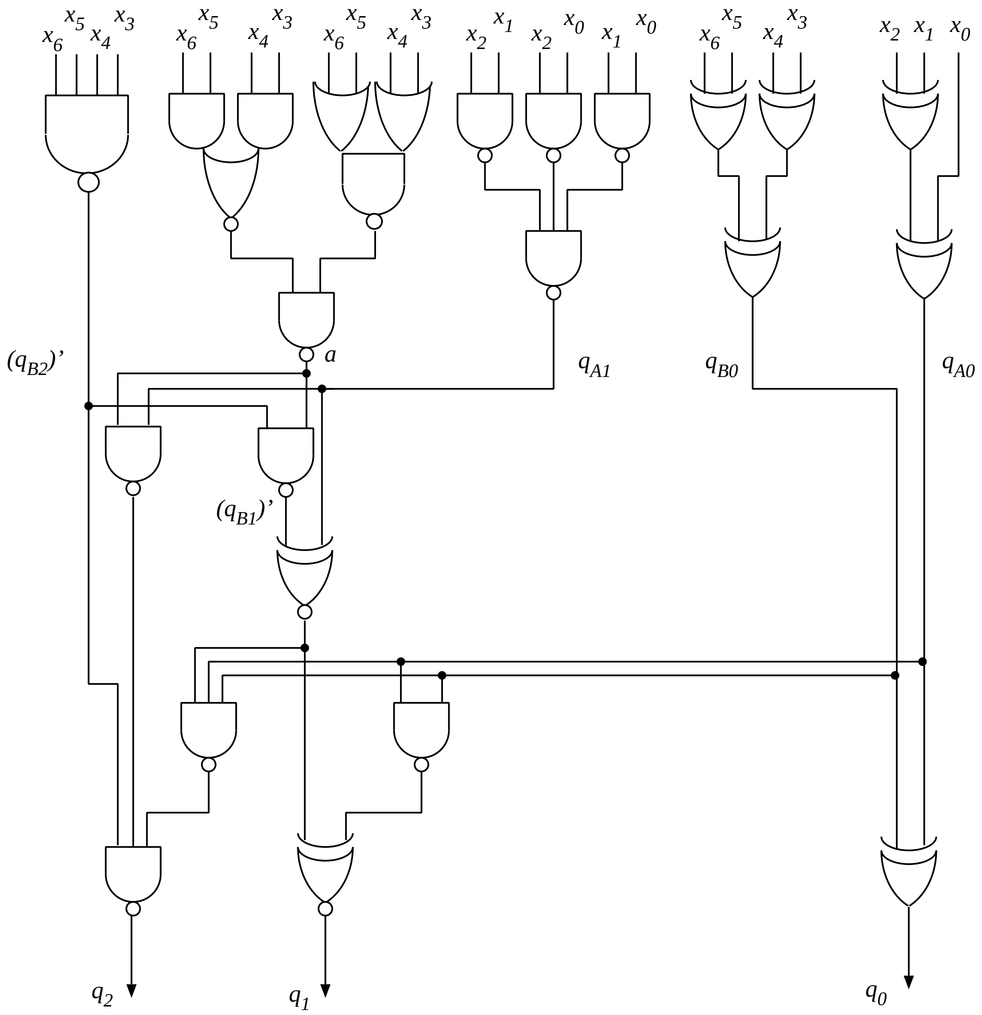


Figure 3.9: Gate network of a (7:3] counter.

MULTICOLUMN COUNTER

$$(p_{k-1}, p_{k-2}, \ldots, p_0 : q]$$

$$v = \sum_{i=0}^{k-1} \sum_{j=1}^{p_i} a_{ij} 2^i \leq 2^q - 1$$

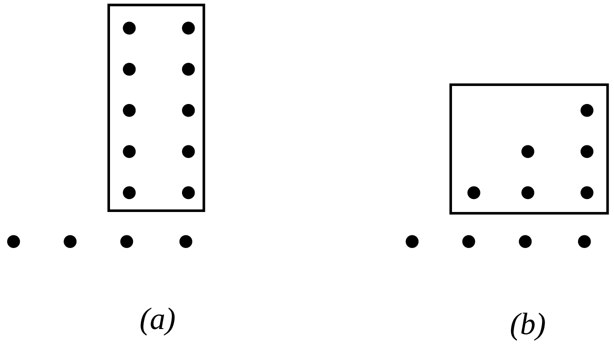


Figure 3.10: (a) (5,5:4] counter. (b) (1,2,3:4] counter.

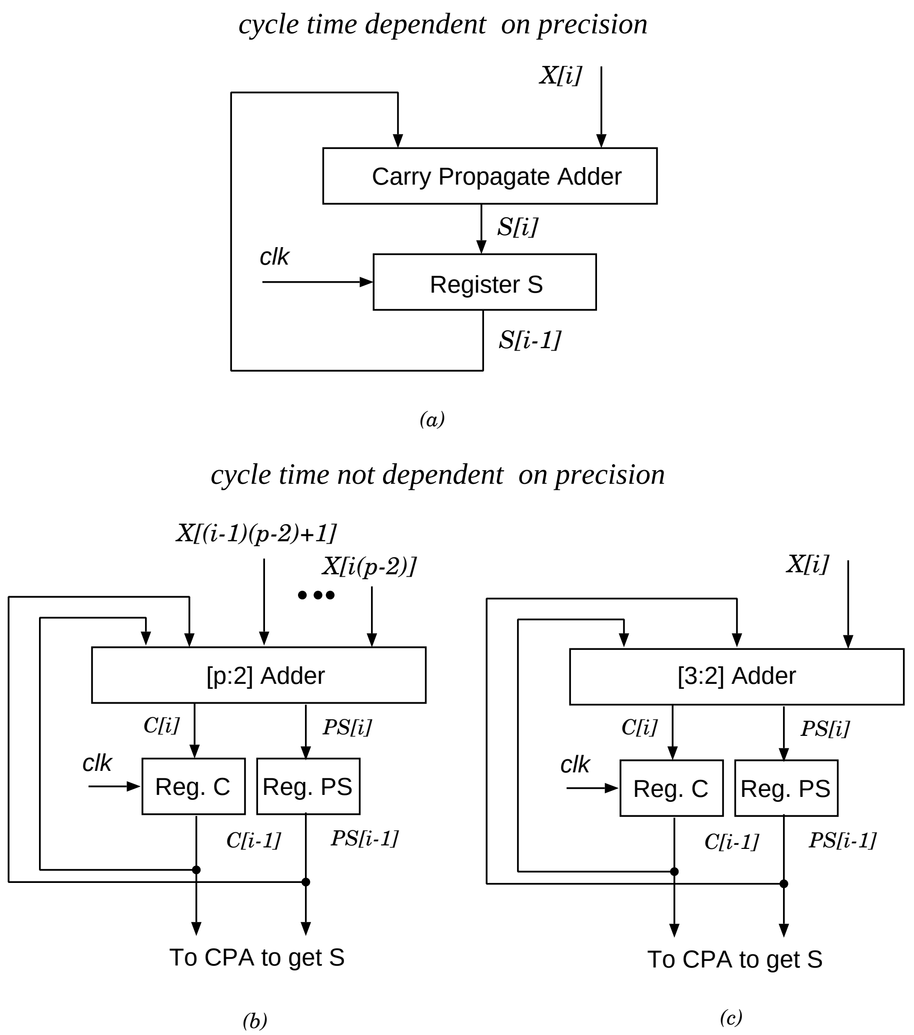


Figure 3.11: SEQUENTIAL MULTIOPERAND ADDITION: a) WITH CONVENTIONAL ADDER. b) WITH [p:2] ADDER. c) WITH [3:2] ADDER.

COMBINATIONAL IMPLEMENTATION

- Reduction by rows: array of adders
  - Linear array
  - Adder tree
- Reduction by columns with (p:q] counters

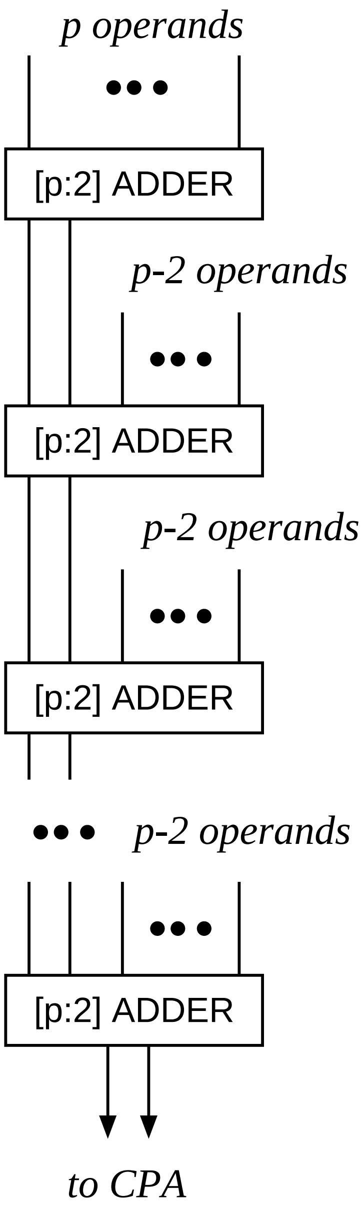


Figure 3.12: LINEAR ARRAY OF [p:2] ADDERS FOR MULTIOPERAND ADDITION.

ADDER TREE

- k - the number of [p:2] CS adders for m operands:

$$pk = m + 2(k-1)$$

$$k = \left\lceil \frac{m-2}{p-2} \right\rceil \qquad \text{[p:2] carry-save adders}$$

- The number of adder levels

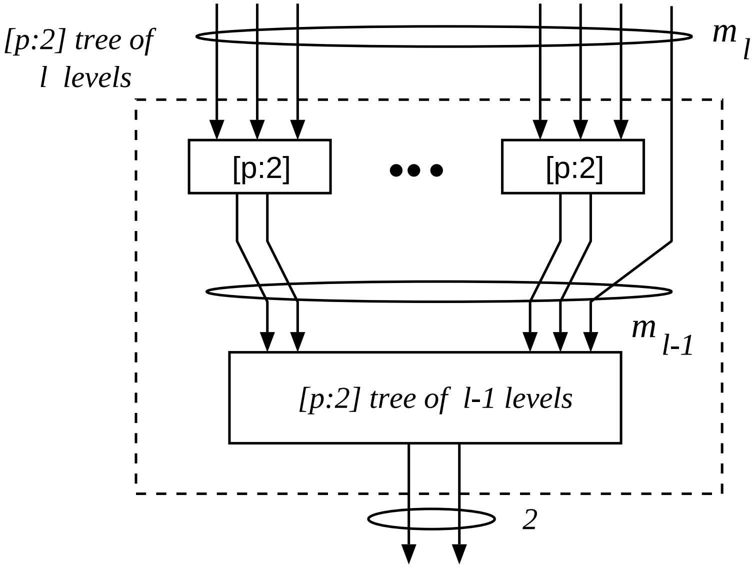


Figure 3.13: Construction of a [p:2] carry-save adder tree.

$$m_l = p \left\lfloor \frac{m_{l-1}}{2} \right\rfloor + m_{l-1} mod\ 2$$

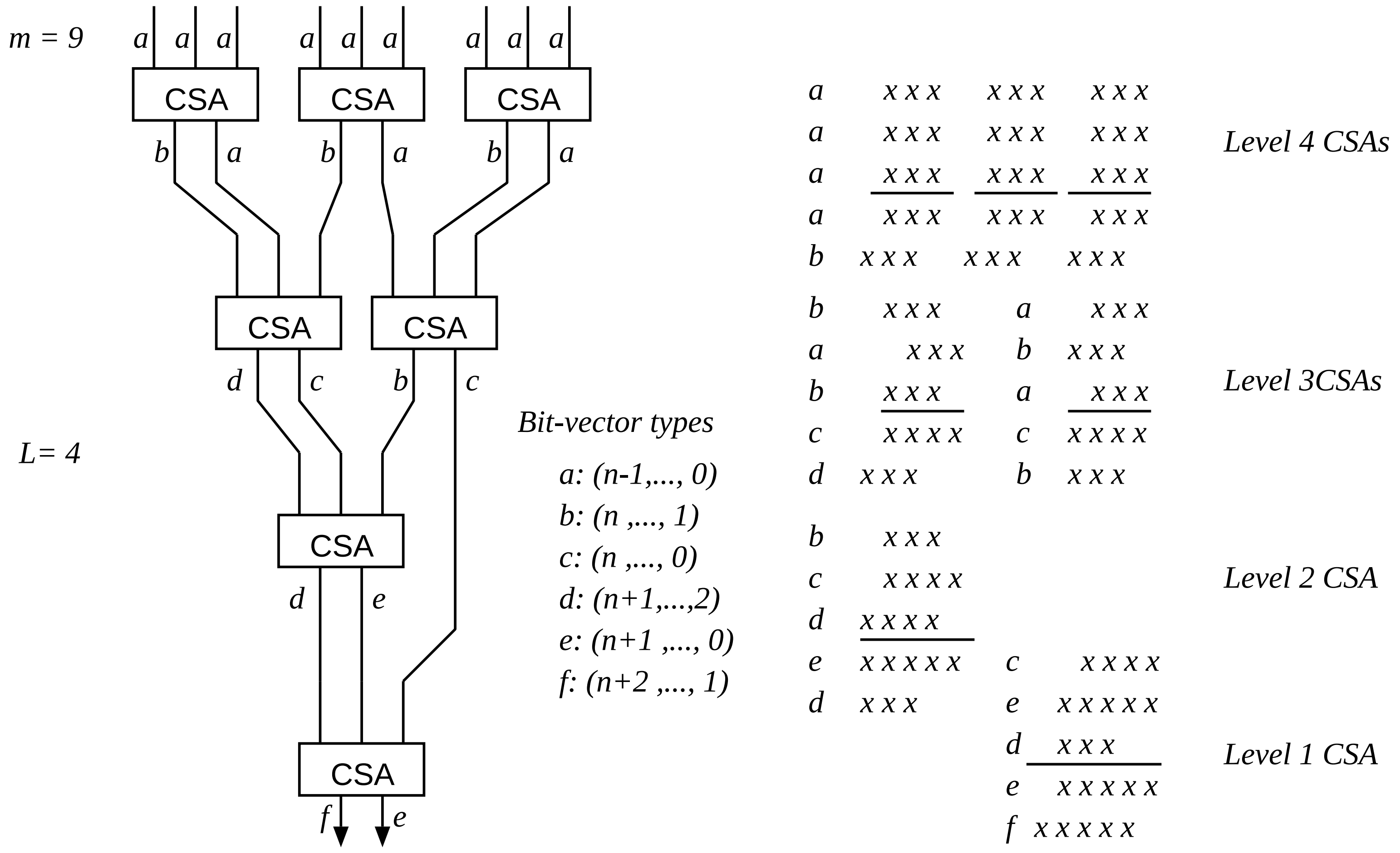
NUMBER OF LEVELS (cont.)

Table 3.1: [3:2] Reduction sequence.

l	1	2	3	4	5	6	7	8	9
m_l	3	4	6	9	13	19	28	42	63

$$m_l \approx \frac{p^l}{2^{l-1}}$$

$$l \approx log_{p/2}(m_l/2)$$



Bit-vector types

a: (n-1,..., 0)
b: (n ,..., 1)
c: (n ,..., 0)
d: (n+1,...,2)
e: (n+1 ,..., 0)
f: (n+2 ,..., 1)

Figure 3.14: [3:2] adder tree for 9 operands (magnitudes with $n = 3$) .

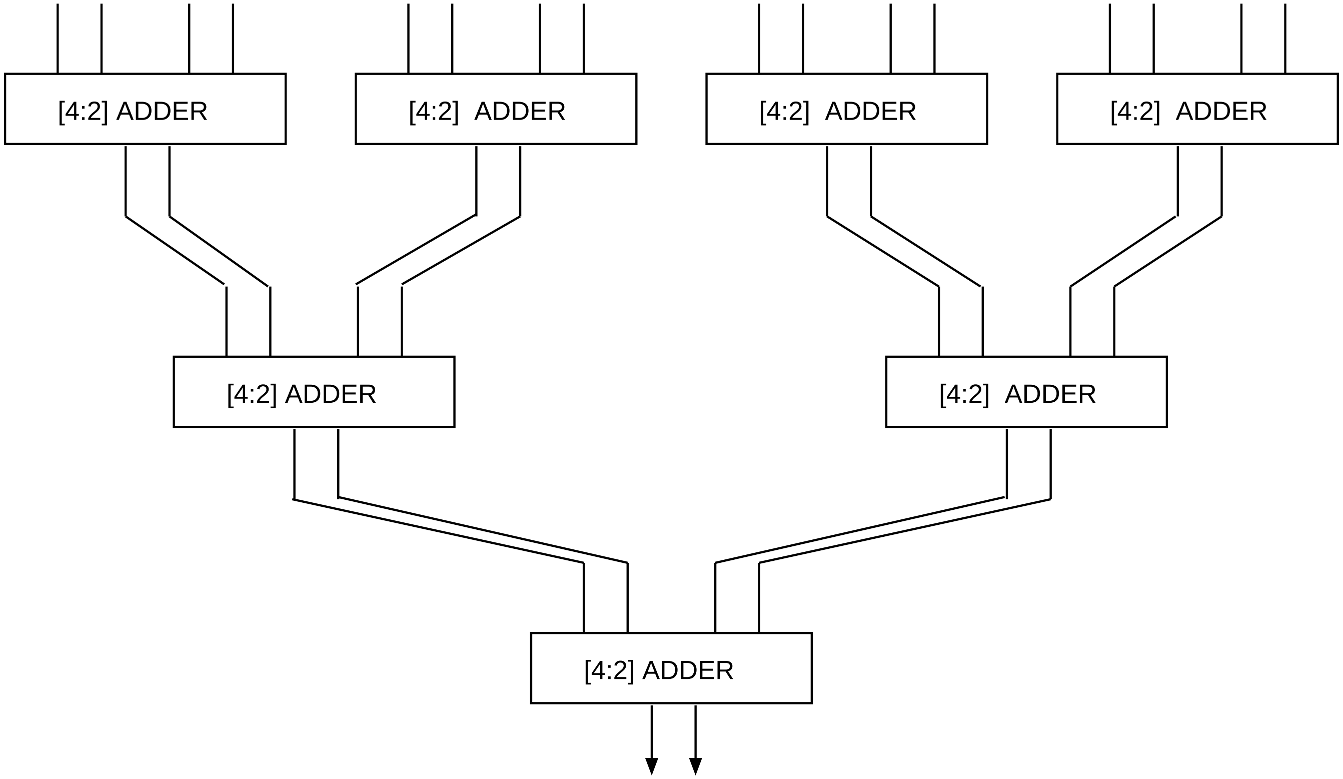


Figure 3.15: Tree of [4:2] adders for $m = 16$.

REDUCTION BY COLUMNS WITH (p:q] COUNTERS

$$
\begin{array}{cccccc}
 & & 1 & \mathbf{0} & 1 & 1 \\
 & & 0 & \mathbf{0} & 1 & 0 \\
 & & 1 & \mathbf{0} & 0 & 1 \\
 & & 0 & \mathbf{1} & 1 & 0 \\
 & & 1 & \mathbf{0} & 1 & 0 \\
 & & 1 & \mathbf{1} & 1 & 1 \\
 & & 0 & \mathbf{1} & 1 & 0 \\
\hline
 & & 0 & \mathbf{1} & 0 & 1 \\
 & 0 & \mathbf{1} & 1 & 1 & \\
1 & \mathbf{0} & 1 & 0 & & \\
\end{array}
$$

Figure 3.16: Example of reduction using (7:3] counters.

NUMBER OF COUNTER LEVELS

$$m_1 = p$$
$$m_l = p\left\lfloor\frac{m_{l-1}}{q}\right\rfloor + m_{l-1}\ mod\ q$$

$$l \approx log_{p/q}(m_l/q)$$

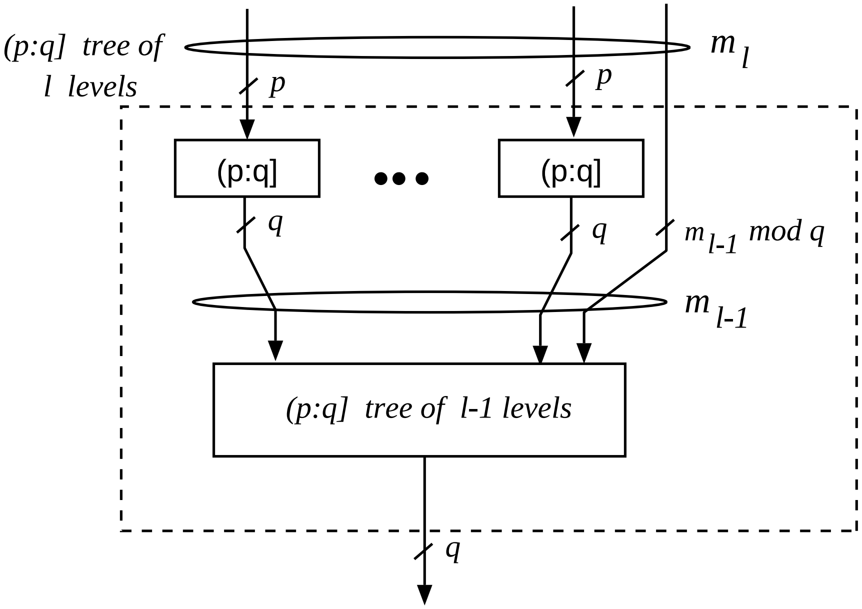


Figure 3.17: Construction of (p:q] reduction tree.

Table 3.2: Sequence for (7:3] counters

Number of levels	1	2	3	4	...
Max. number of rows	7	15	35	79	...

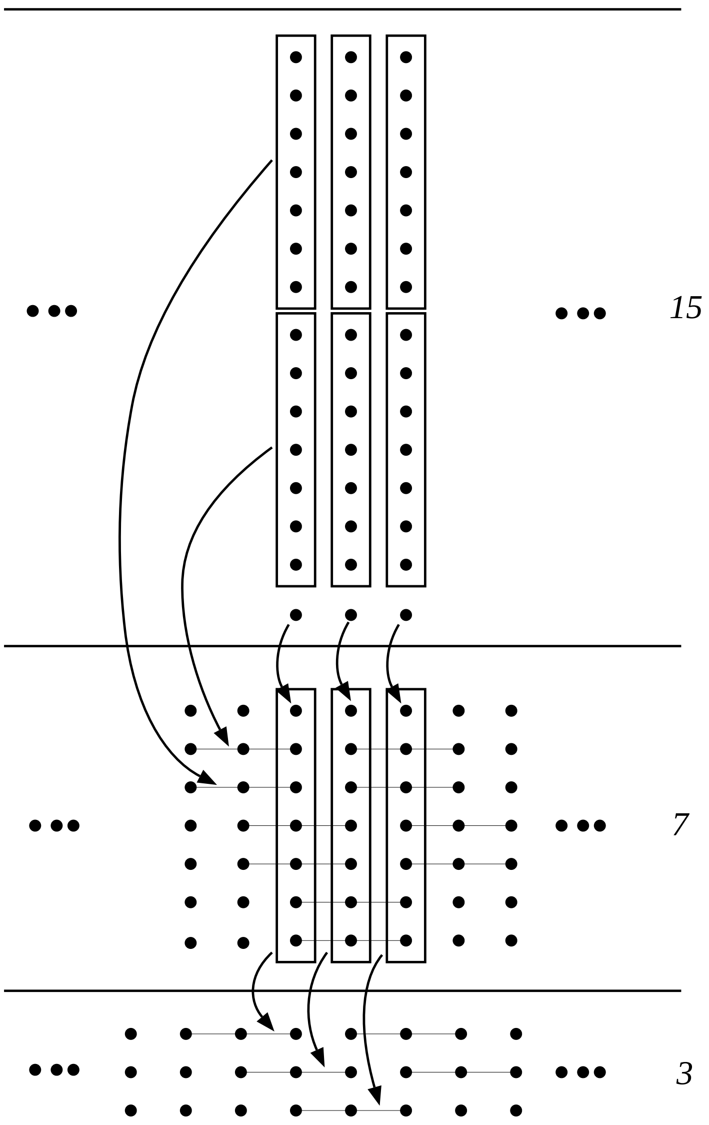
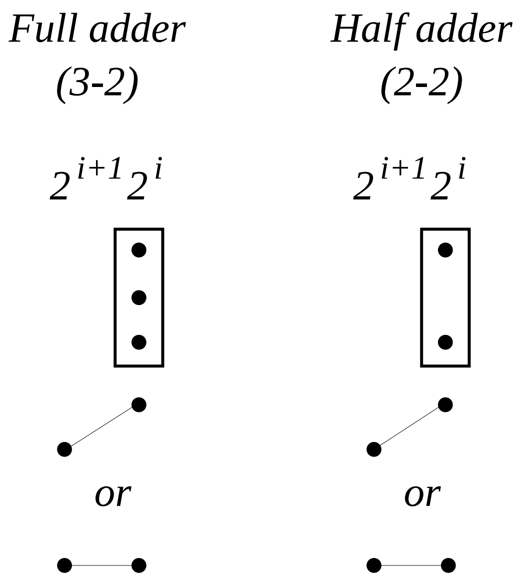


Figure 3.18: Multilevel reduction with (7:3] counters

SYSTEMATIC DESIGN METHOD



● *denotes 0 or 1*

diagonal outputs when representing separately sum and carry bit-vectors is preferrable

horizontal outputs when interleaving sum and carry bits is acceptable

Figure 3.19: Full adder and half adder as (3:2] and (2:2] counters.

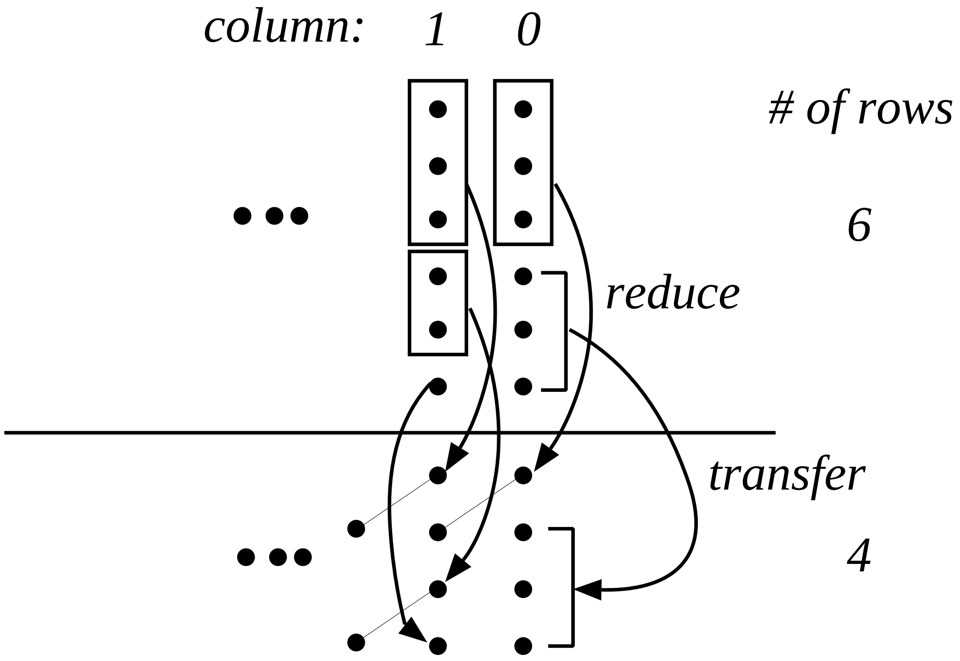


Figure 3.20: Reduction process.

RELATION AT LEVEL l

e_i – number of bits in column i

f_i – number of full adders in column i

h_i – number of half adders in column i

$$e_i - 2f_i - h_i + f_{i-1} + h_{i-1} = m_{l-1}$$

resulting in

$$2f_i + h_i = e_i - m_{l-1} + f_{i-1} + h_{i-1} = p_i$$

Solution producing min number of carries:

$$f_i = \lfloor p_i/2 \rfloor \quad\quad h_i = p_i \; mod \; 2$$

					i		
	6	5	4	3	2	1	0
$l = 4$							
e_i			8	8	8	8	8
m_3			6	6	6	6	6
h_i			0	0	0	1	0
f_i			2	2	2	1	1
$l = 3$							
e_i		2	6	6	6	6	6
m_2		4	4	4	4	4	4
h_i		0	0	0	0	1	0
f_i		0	2	2	2	1	1
$l = 2$							
e_i		4	4	4	4	4	4
m_1		3	3	3	3	3	3
h_i		0	0	0	0	0	1
f_i		1	1	1	1	1	0
$l = 1$							
e_i	1	3	3	3	3	3	3
m_0	2	2	2	2	2	2	2
h_i	0	0	0	0	0	0	1
f_i	0	1	1	1	1	1	0

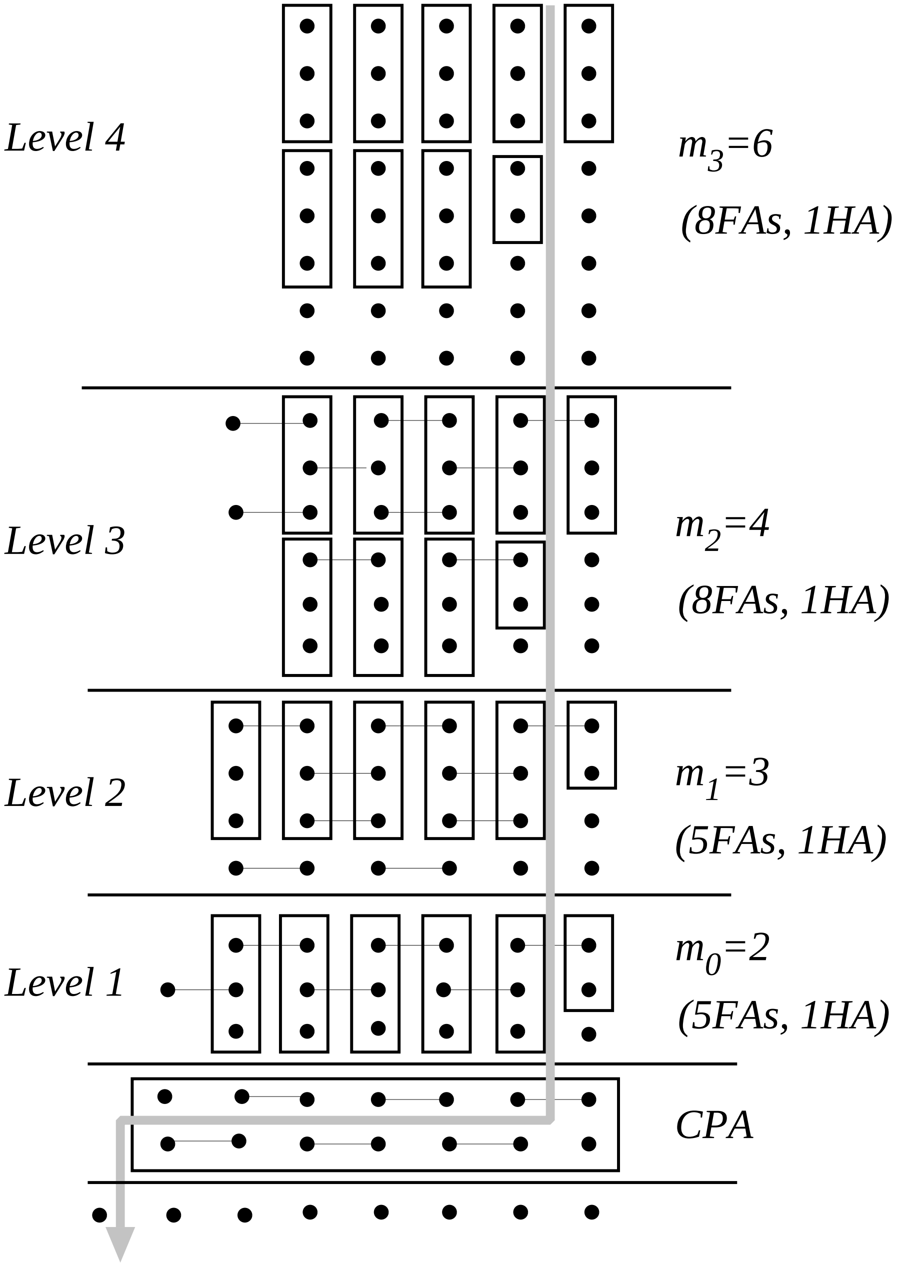


Figure 3.21: Reduction by columns of 8 5-bit magnitudes. Cost of reduction: 26 FAs and 4 HAs.

EXAMPLE: ARRAY FOR $f = a + 3b + 3c + d$

Operands in [-4,3). Result range:

$$-4 + (-12) + (-12) - 4 = -32 \leq f \leq 3 + 9 + 9 + 3 = 24$$

$$\begin{array}{r|cccccc}
a & a_2 & a_2 & a_2 & a_2 & a_1 & a_0 \\
b & b_2 & b_2 & b_2 & b_2 & b_1 & b_0 \\
2b & b_2 & b_2 & b_2 & b_1 & b_0 & 0 \\
c & c_2 & c_2 & c_2 & c_2 & c_1 & c_0 \\
2c & c_2 & c_2 & c_2 & c_1 & c_0 & 0 \\
d & d_2 & d_2 & d_2 & d_2 & d_1 & d_0
\end{array}$$

transformed into

$$\begin{array}{r|cccc}
a & & a_2' & a_1 & a_0 \\
 & & \text{-1} & & \\
b & & b_2' & b_1 & b_0 \\
 & & \text{-1} & & \\
2b & b_2' & b_1 & b_0 & \\
 & \text{-1} & & & \\
c & & c_2' & c_1 & c_0 \\
 & & \text{-1} & & \\
2c & c_2' & c_1 & c_0 & \\
 & \text{-1} & & & \\
d & & d_2' & d_1 & d_0 \\
 & & \text{-1} & &
\end{array}$$

FINAL BIT MATRIX

$$\begin{array}{|cccccc}
1 & 0 & b'_2 & a'_2 & a_1 & a_0 \\
 & & c'_2 & b'_2 & b_1 & b_0 \\
 & & & b_1 & b_0 & \\
 & & & c'_2 & c_1 & c_0 \\
 & & & c_1 & c_0 & \\
 & & & d'_2 & d_1 & d_0
\end{array}$$

	i					
	5	4	3	2	1	0
$l=3$						
e_i	1	0	2	6	6	4
m_2	4	4	4	4	4	4
h_i	0	0	0	1	0	0
f_i	0	0	0	1	1	0
$l=2$						
e_i	1	0	4	4	4	4
m_1	3	3	3	3	3	3
h_i	0	0	0	0	0	1
f_i	0	0	1	1	1	0
$l=1$						
e_i	1	1	3	3	3	3
m_0	2	2	2	2	2	1*
h_i	0	0	0	0	0	0
f_i	0	0	1	1	1	1

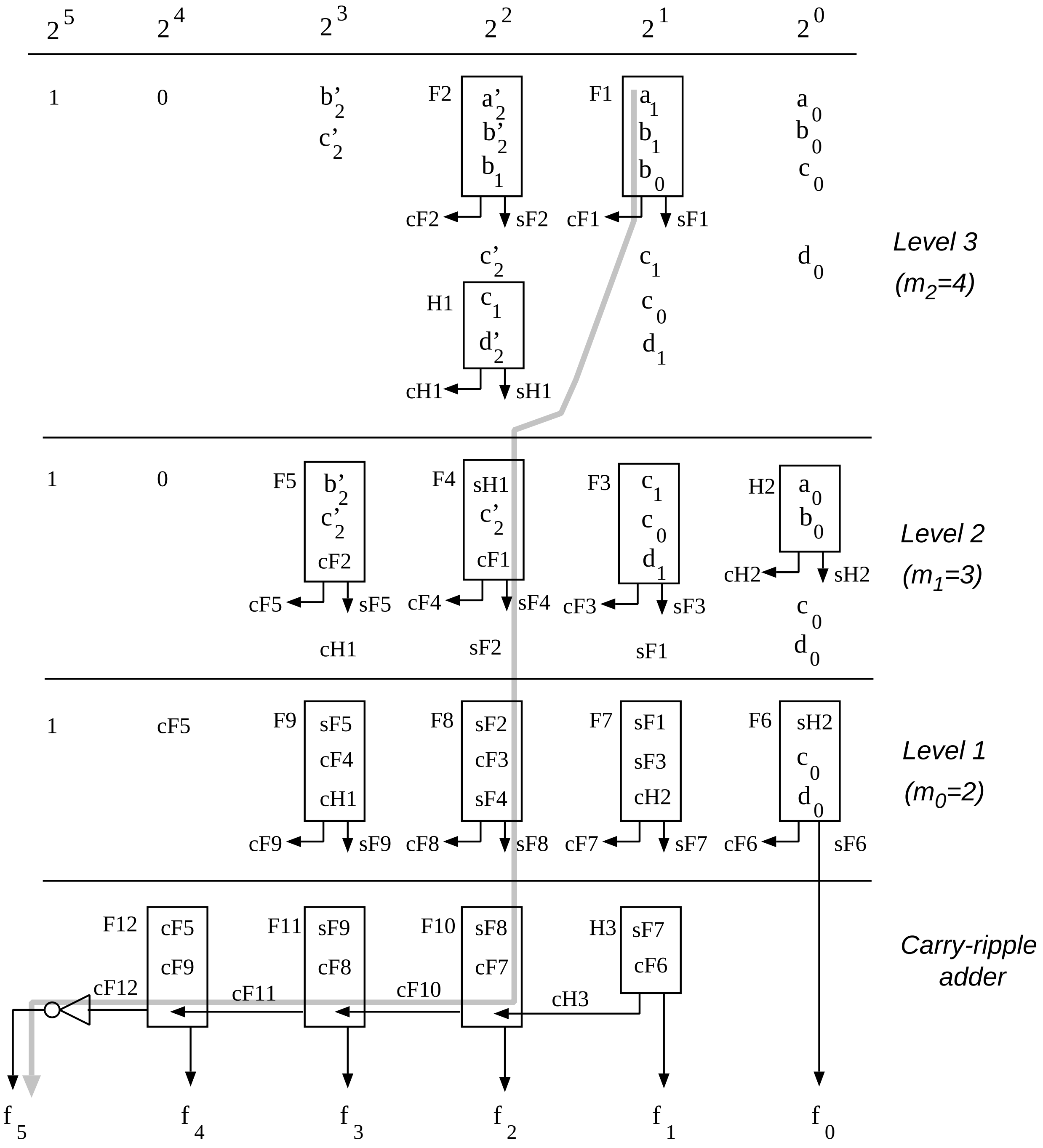


Figure 3.22: Reduction array $f = a + 3b + 3c + d$.

PIPELINED LINEAR ARRAY

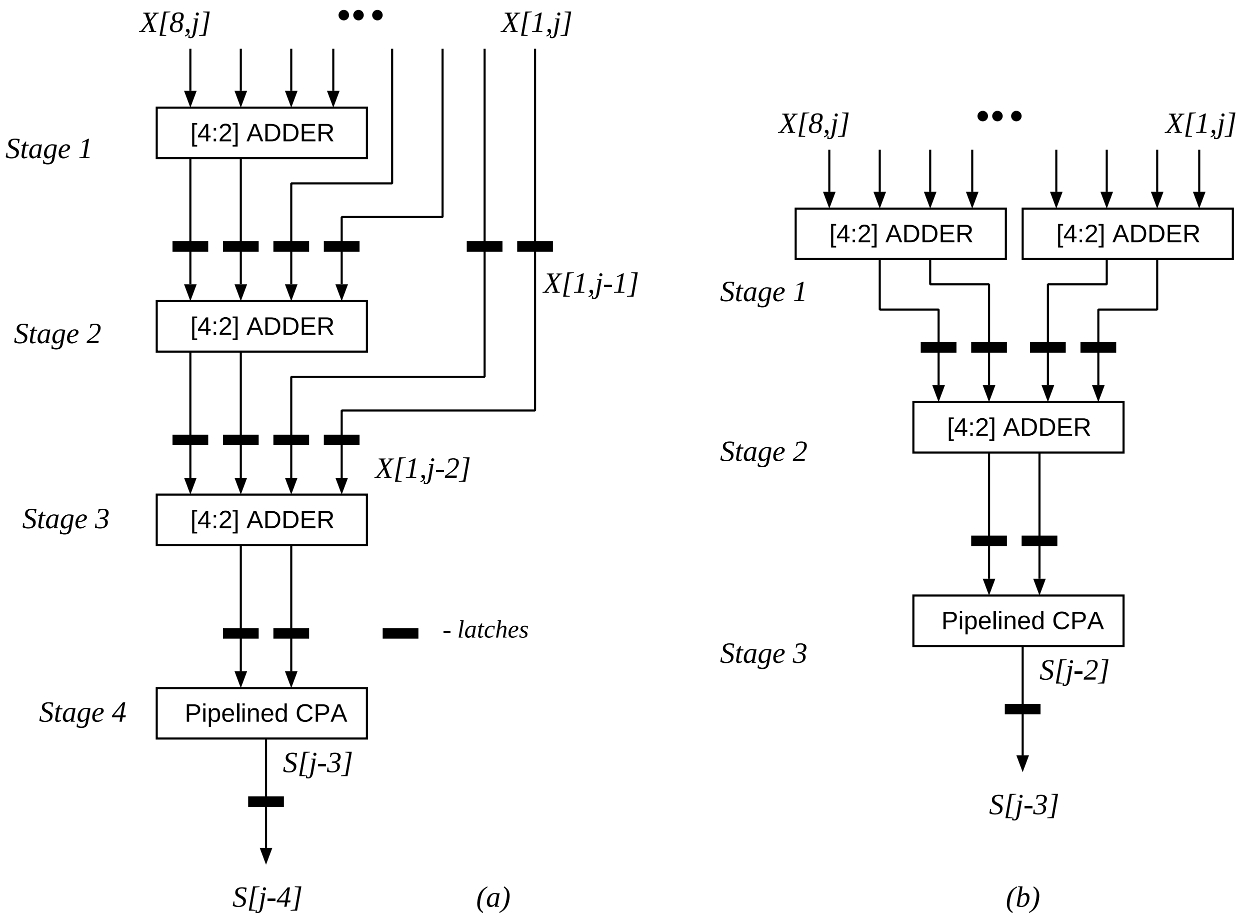


Figure 3.23: Pipelined arrays with [4:2] adders for computing $S[j] = \sum_{i=1}^{8} X[i,j],\ j = 1,\ldots,N$: (a) Linear array. (b) Tree array.

PARTIALLY COMBINATIONAL IMPLEMENTATION

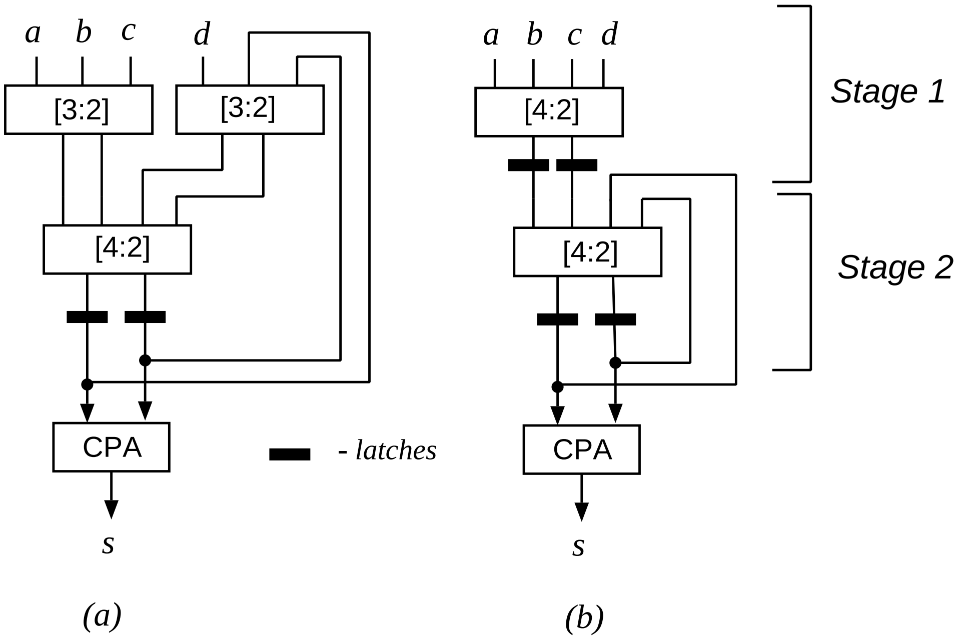


Figure 3.24: Partially combinational scheme for summation of 4 operands per iteration: (a) Nonpipelined. (b) Pipelined.

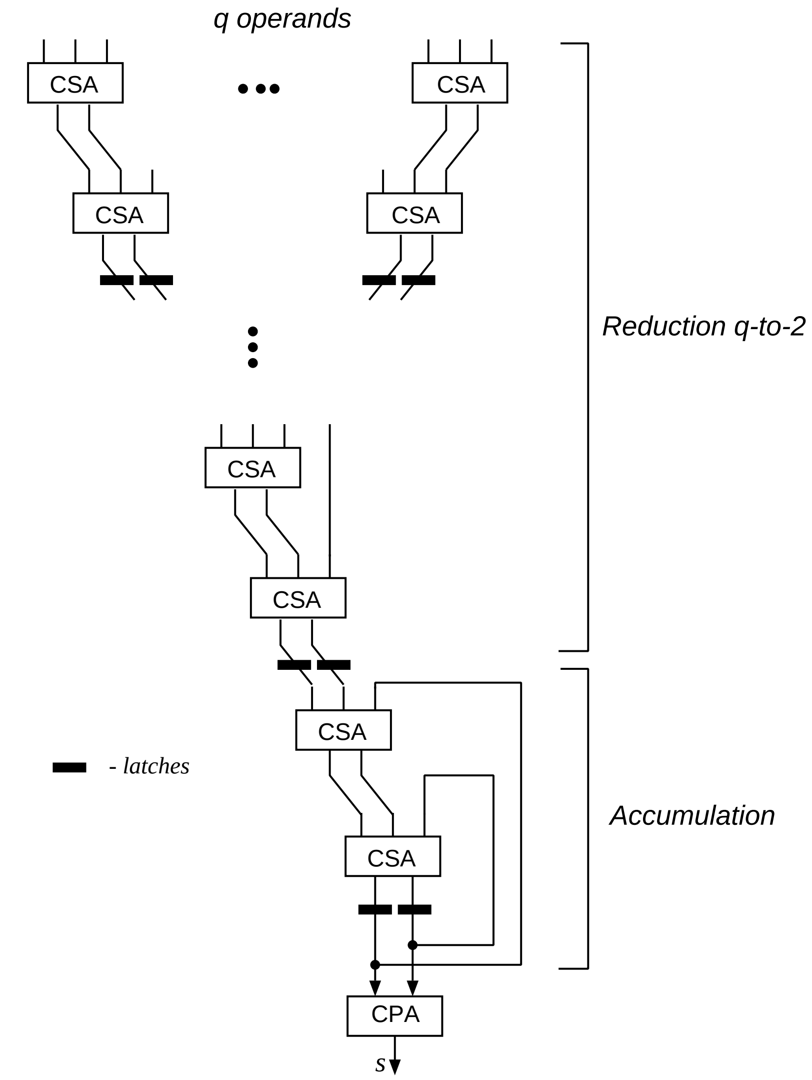


Figure 3.25: Scheme for summation of q operands per iteration.

MULTIPLICATION

$p = x \times y$

x (multiplicand), y (multiplier), and p (product) signed integers

- SCHEMES

  a) SEQUENTIAL ADD-SHIFT RECURRENCE

  - ∗ CPA, CSA, SIGNED-DIGIT ADDER
  - ∗ HIGHER RADIX AND RECODING

  b) COMBINATIONAL

  - ∗ CPA, CSA, SIGNED-DIGIT ADDER
  - ∗ HIGHER RADIX AND RECODING

  c) COLUMN REDUCTION

  d) ARRAYS WITH $k \times l$ MULTIPLIERS

TOPICS (cont.)

- MULTIPLY-ADD AND MULTIPLY-ACCUMULATE
- SATURATING MULTIPLIERS
- TRUNCATING MULTIPLIERS
- RECTANGULAR MULTIPLIERS
- SQUARERS
- CONSTANT AND MULTIPLE CONSTANT MULTIPLIERS

SIGN-AND-MAGNITUDE

- EACH OPERAND:

  sign with value $+1$ and -1 and n-digit magnitude
- RESULT: a sign and a $2n$-digit magnitude
- HIGH-LEVEL ALGORITHM

$$sign(p) = sign(x) \cdot sign(y)$$

$$|p| = |x||y|$$

- REPRESENTATIONS OF MAGNITUDES

$$\begin{array}{lll} X = (x_{n-1}, x_{n-2}, \ldots, x_0) & |x| = \Sigma_{i=0}^{n-1} x_i r^i & \text{(multiplicand)} \\ Y = (y_{n-1}, y_{n-2}, \ldots, y_0) & |y| = \Sigma_{i=0}^{n-1} y_i r^i & \text{(multiplier)} \\ P = (p_{2n-1}, p_{2n-2}, \ldots, p_0) & |p| = \Sigma_{i=0}^{2n-1} p_i r^i & \text{(product)} \end{array}$$

TWO'S COMPLEMENT

- RADIX-2 CASE
- EACH OPERAND: n-BIT VECTOR
- RESULT: $2n$-BIT VECTOR

$$-(2^{n-1})(2^{n-1}-1) \leq p \leq (-2^{n-1})(-2^{n-1}) = 2^{2n-2}$$

- x_R, y_R and p_R – positive integer representations of x, y, and p
- HIGH-LEVEL ALGORITHM

$$p_R = \begin{cases} x_R y_R & \text{if } x \geq 0,\ y \geq 0 \\ 2^{2n} - (2^n - x_R)y_R & \text{if } x < 0,\ y \geq 0 \\ 2^{2n} - x_R(2^n - y_R) & \text{if } x \geq 0,\ y < 0 \\ (2^n - x_R)(2^n - y_R) & \text{if } x < 0,\ y < 0 \end{cases}$$

TYPES OF ALGORITHMS

1. ADD-AND-SHIFT ALGORITHM
    - SEQUENTIAL
    - COMBINATIONAL
2. COMPOSITION OF SMALLER MULTIPLICATIONS

RECURRENCE FOR MAGNITUDES

$$p[0] = 0$$
$$p[j+1] = r^{-1}(p(j) + x \cdot r^n y_j) \text{ for } j = 0, 1, \ldots, n-1$$
$$p = p[n]$$

RELATIVE POSITION OF OPERANDS

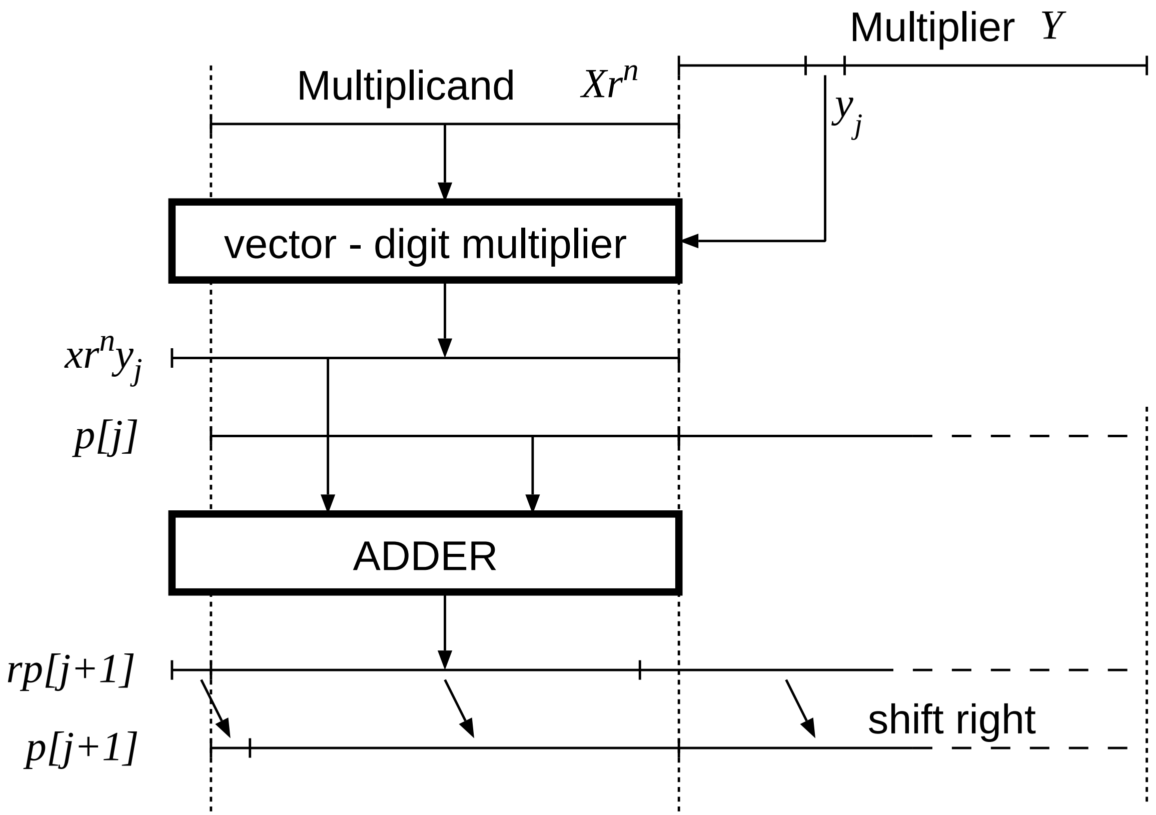


Figure 4.1: RELATIVE POSITION OF OPERANDS IN MULTIPLICATION RECURRENCE

$$T = n(t_{digmult} + t_{add} + t_{reg})$$

SEQUENTIAL MULTIPLIER WITH REDUNDANT ADDER

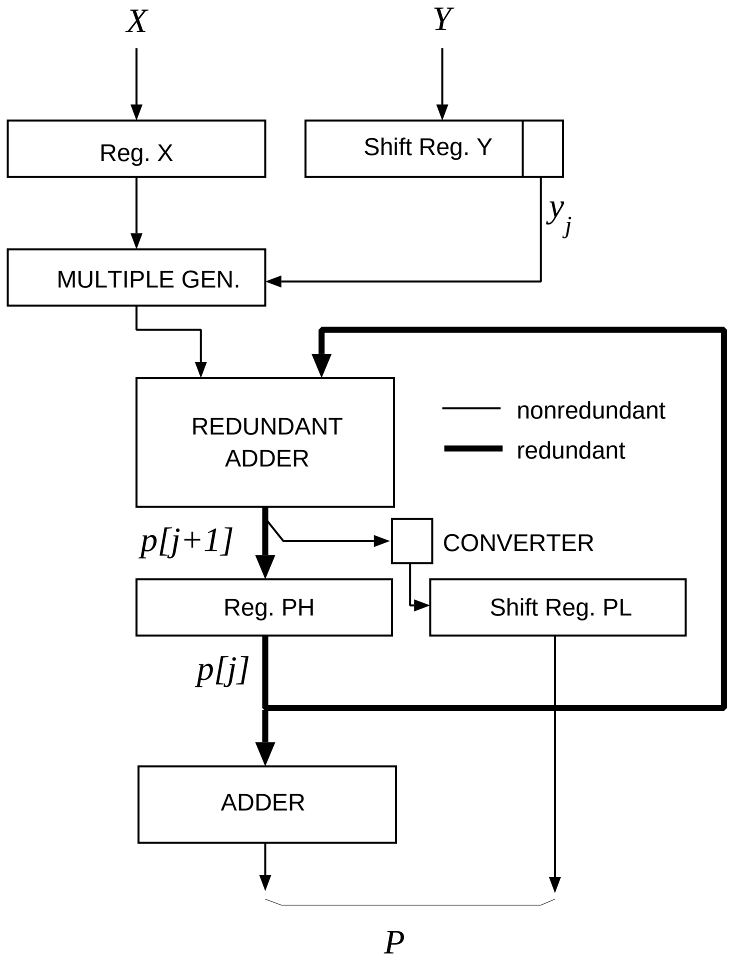


Figure 4.2: SEQUENTIAL MULTIPLIER WITH REDUNDANT ADDER

RADIX-4 SEQUENTIAL MULTIPLIER RECODING

- MULTIPLIER RECODING TO AVOID VALUES $z_i = 3$

$$z_i = y_i + c_i - 4c_{i+1}$$

$y_i + c_i$	z_i	c_{i+1}
0	0	0
1	1	0
2	2	0
3	-1	1
4	0	1

RADIX-4 MULTIPLIER IMPLEMENTATION

THREE PIPELINED STAGES

- Stage 1: MULTIPLIER RECODING
- Stage 2: GENERATING THE MULTIPLE OF THE MULTIPLICAND
- Stage 3: ADDITION AND SHIFT (with conversion of the shifted-out bits).

cycle	0	1	2	3	4	5	...	$m+1$	$m+2$	
	LOAD X LOAD Y									
Stage 1	0	z_0	z_1	z_2	z_3	z_4				
Stage 2	0	0	Xz_0	Xz_1	Xz_2	Xz_3		Xz_{m-1}		
Stage 3	0	0	0	PS[1] SC[1]	PS[2] SC[2]	PS[3] SC[3]		PS[$m-1$] SC[$m-1$]	PS[m] SC[m]	
CPA										Final product

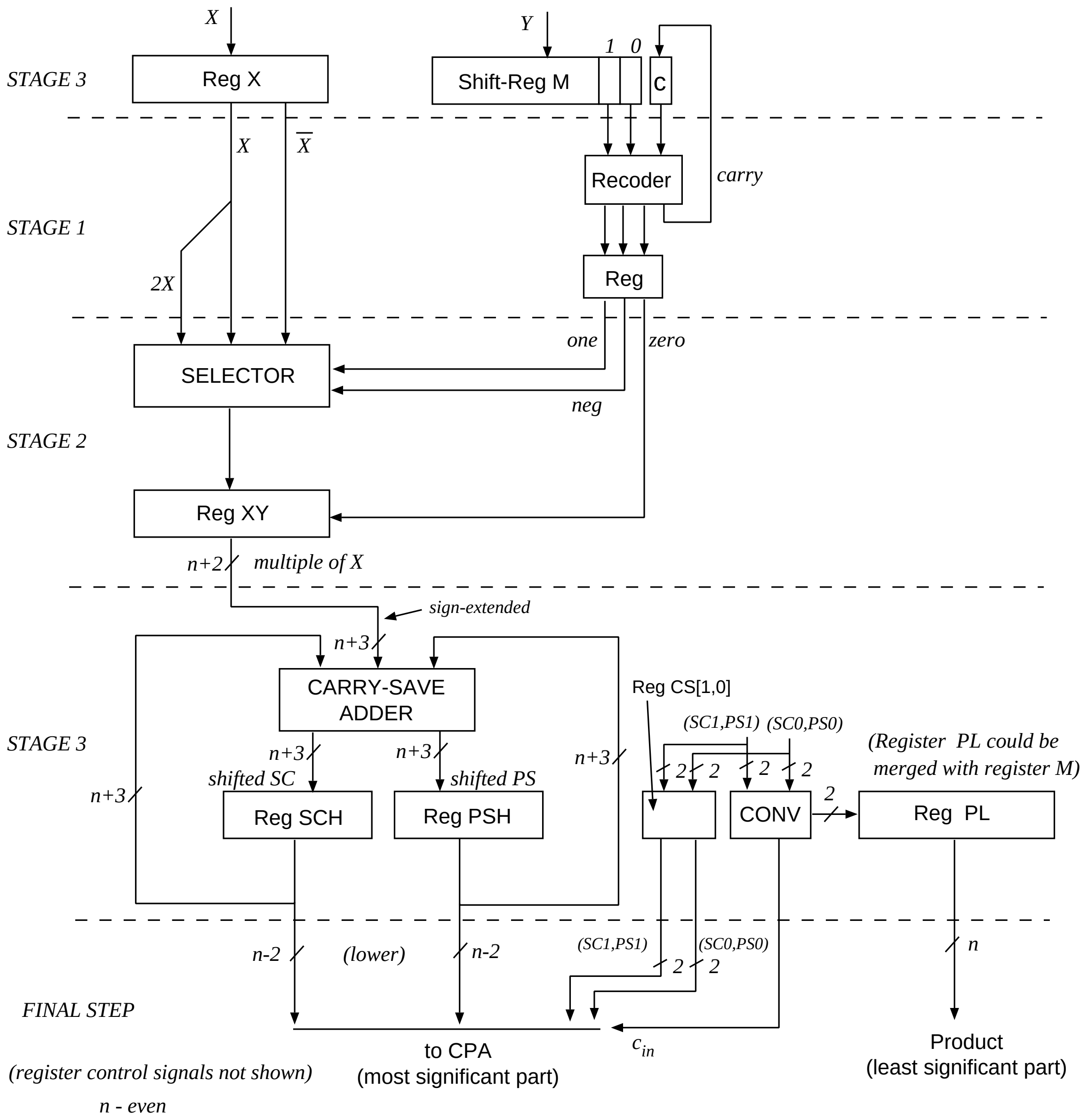


Figure 4.3: RADIX-4 MULTIPLIER.

RECODING IMPLEMENTATION

- BASED ON MULTIPLIER BITS (M_1, M_0) and CARRY FLAG C

$$one = M_0 \oplus C = \begin{cases} 0 & \text{select } 2x \\ 1 & \text{select } x \end{cases}$$

$$neg = M_1 \cdot C + M_1 \cdot M_0 = \begin{cases} 0 & \text{select direct} \\ 1 & \text{select complement} \end{cases}$$

$$zero = M_1 \cdot M_0 \cdot C + M_1' \cdot M_0' \cdot C' = \begin{cases} 0 & \text{load non} - \text{zero multiple} \\ 1 & \text{load zero multiple (clear)} \end{cases}$$

$$C_{next} = M_1 M_0 + M_1 C$$

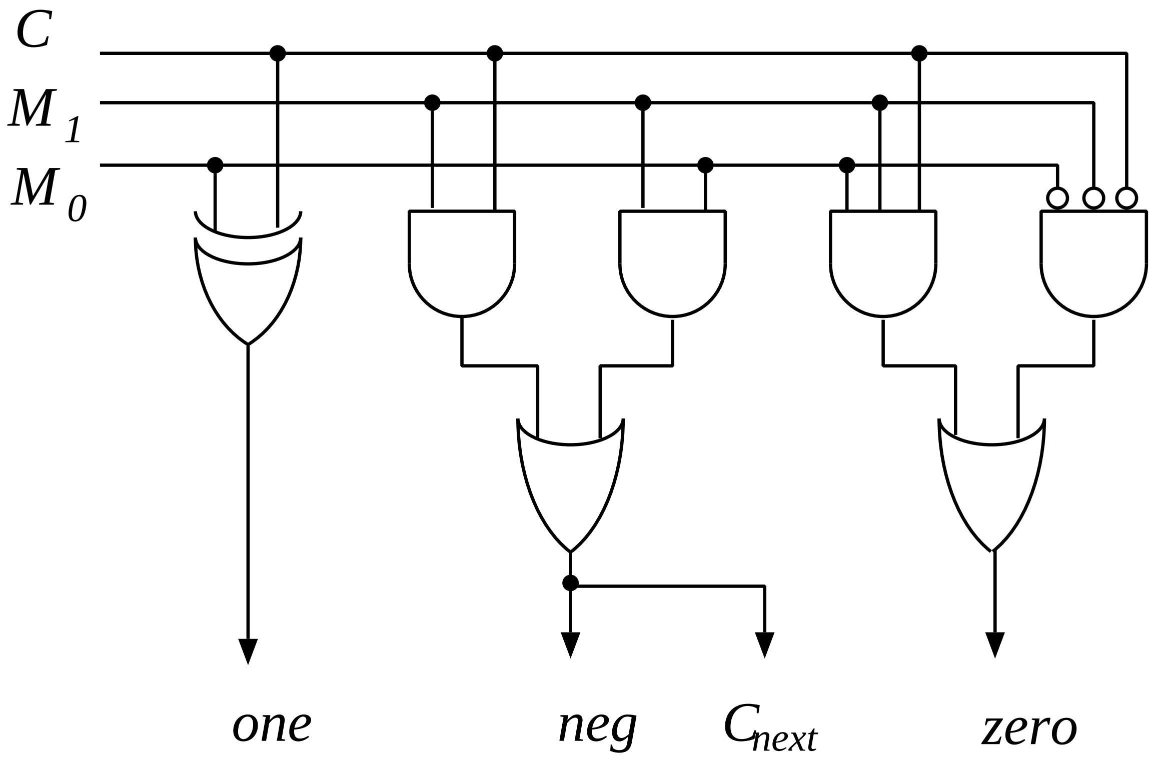


Figure 4.4: RECODER IMPLEMENTATION.

GENERATION OF $(-1)x$

$$
\begin{array}{lllllll}
PS[j] & PS_{n+2} & PS_{n+1} & PS_n & \cdots & PS_1 & PS_0 \\
SC[j] & SC_{n+2} & SC_{n+1} & SC_n & \cdots & SC_1 & SC_0 \\
-x & X'_{n+2} & X'_{n+1} & X'_n & \cdots & X'_1 & X'_0 \\
\hline
CSA & s_{n+2} & s_{n+1} & s_n & \cdots & s_1 & s_0 \\
 & c_{n+2} & c_{n+1} & c_n & \cdots & c_1 & 1^*
\end{array}
$$

$*$ for 2's complement of x

EXAMPLE OF RADIX-4 MULTIPLICATION

$n = 6 \quad m = 3$ radix-4 digits

$x = 29 \quad X = 11101$

$y = 27 \quad Y = 11011$

$Z = 2\overline{11} \;\; (z = y) \quad (-1 = \overline{1})$

	CSA	shifted out
$PS[0]$	00000000	
$SC[0]$	00000000	
xZ_0	11100010	
$4PS[1]$	11100010	
$4SC[1]$	0000000**1**	
$PS[1]$	11111000	11
$SC[1]$	00000000	
xZ_1	11100010	
$4PS[2]$	00011010	
$4SC[2]$	1100000**1**	
$PS[2]$	00000110	1111
$SC[2]$	11110000	
xZ_2	00111010	
$4PS[3]$	11001100	
$4SC[3]$	0110010**0**	
$PS[3]$	11110011	001111
$SC[3]$	00011001	
P	1100	001111 $=$ 783

EXTENSION TO HIGHER RADICES

- EXTENSION TO HIGHER RADICES REQUIRES PREPROCESSING OF MORE MULTIPLES
- ALTERNATIVE: USE SEVERAL RADIX-4 AND/OR RADIX-2 STAGES IN ONE ITERATION

EXAMPLE: RADIX-16 MULTIPLIER DIGIT {0,...,15} RECODED INTO A RADIX-16 SIGNED-DIGIT v_i IN THE SET {-10,...,0,...,10} AND DECOMPOSED INTO TWO RADIX-4 DIGITS u_i and w_i SUCH THAT

$$v_i = 4u_i + w_i \quad u_i, w_i \in \{-2, -1, 0, 1, 2\}$$

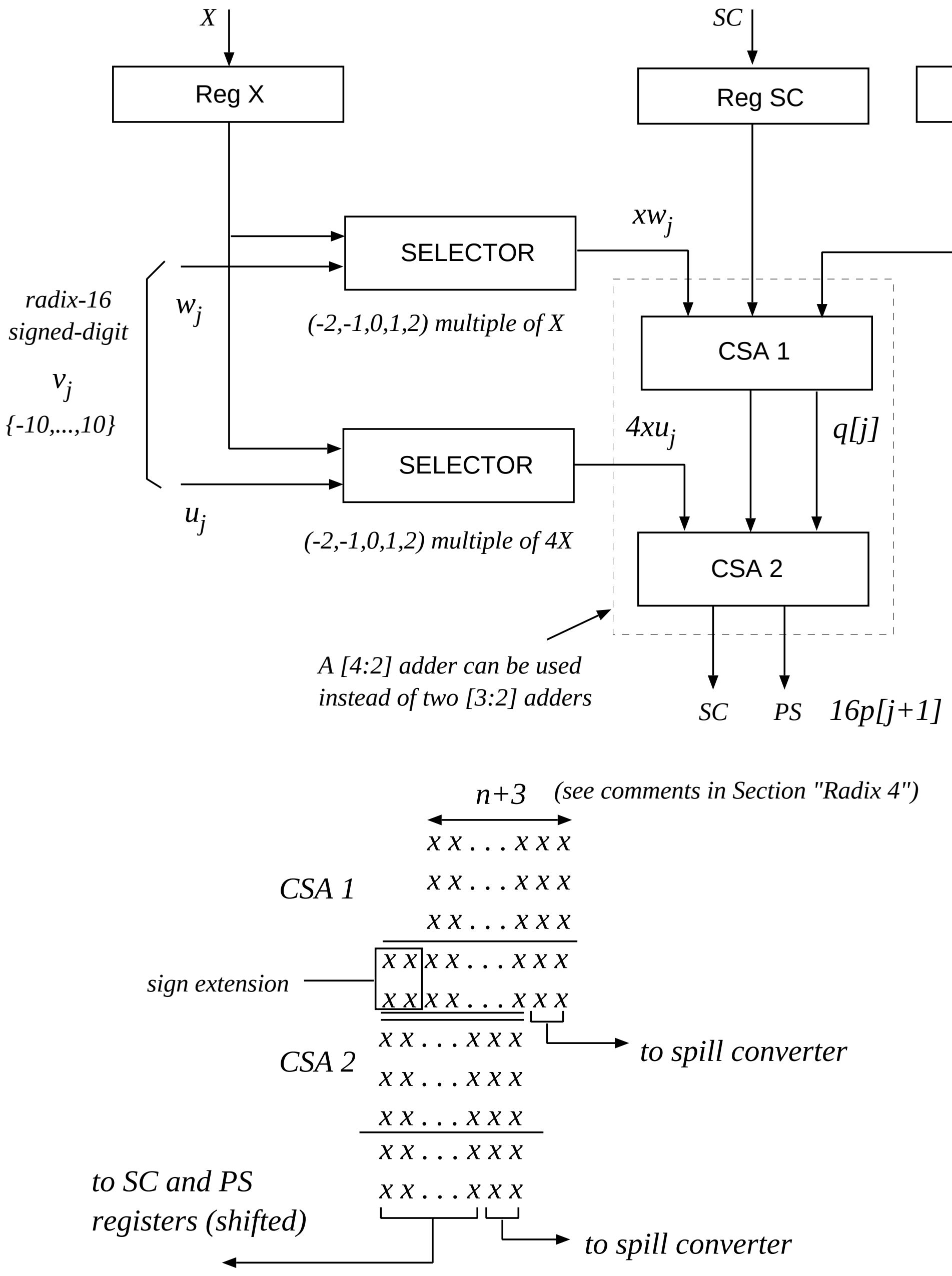


Figure 4.5: RADIX-16 MULTIPLICATION DATAPATH (partial).

TWO'S COMPLEMENT

- MULTIPLICAND IN 2'S COMPLEMENT $\Longrightarrow$ ADDITION AND SHIFT OPERATIONS PERFORMED IN THIS SYSTEM
- THE EFFECT OF 2'S COMPLEMENT MULTIPLIER TAKEN INTO ACCOUNT IN TWO WAYS:

  1. BY SUBTRACTING INSTEAD OF ADDING IN THE LAST ITERATION

     $$y = -y_{n-1}2^{n-1} + \sum_{i=0}^{n-2} y_i 2^i$$

     $\Longrightarrow$ CORRECTION STEP.
  2. BY RECODING THE MULTIPLIER INTO A SIGNED-DIGIT SET

COMBINATIONAL MULTIPLICATION

$$p = \sum_{i=0}^{n-1} x y_i r^i$$

DONE IN TWO STEPS:

1. GENERATION OF THE MULTIPLES OF THE MULTIPLICAND

$$(x \times y_i) r^i$$

2. MULTIOPERAND ADDITION OF THE MULTIPLES GENERATED IN STEP 1.

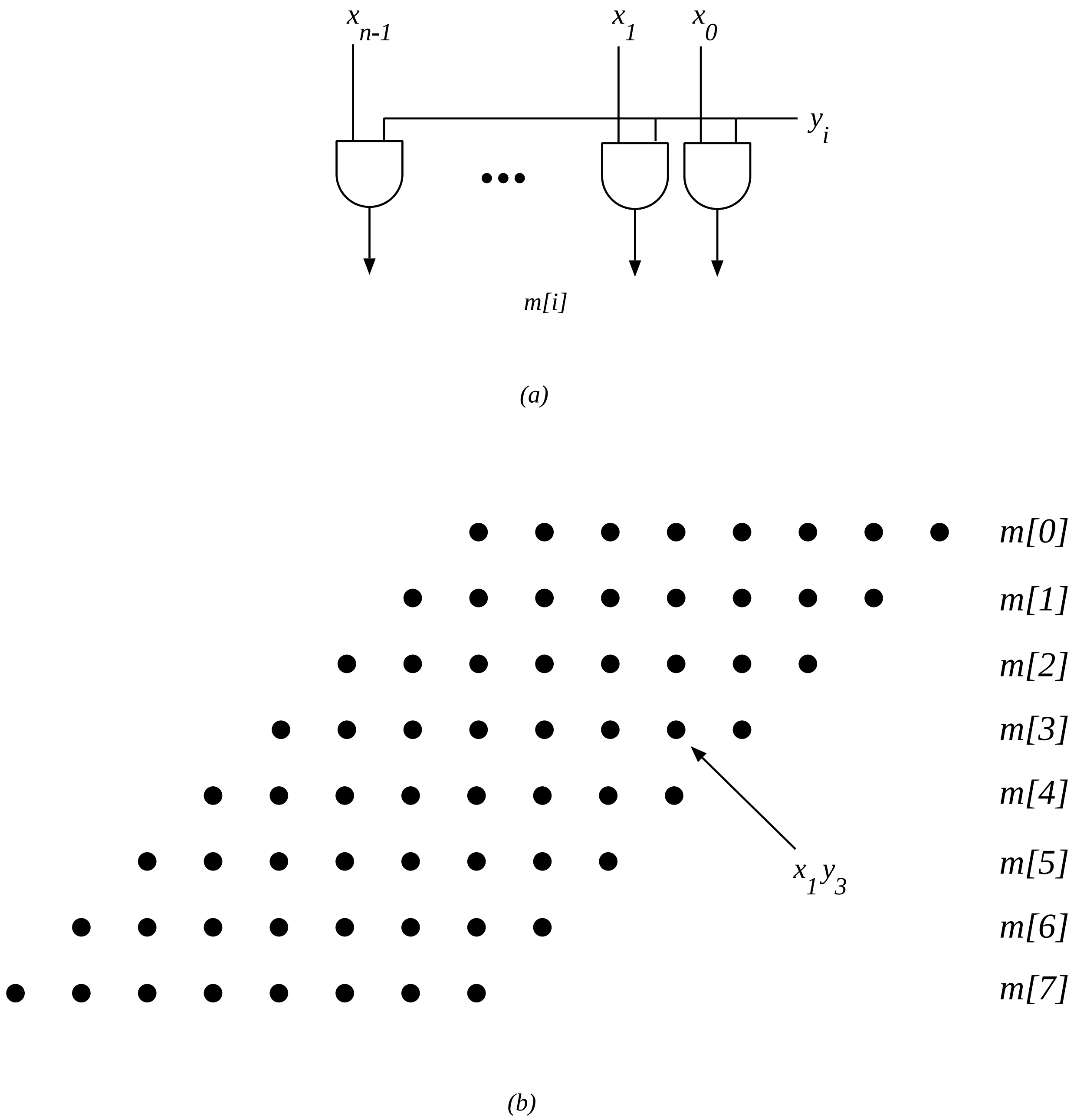


Figure 4.6: (a) RADIX-2 MULTIPLE GENERATION. (b) BIT-MATRIX FOR MULTIPLICATION Of MAGNITUDES ($n = 8$).

RADIX-2 TWO'S COMPLEMENT MULTIPLICATION

1. EXTEND RANGE BY REPLICATING THE SIGN BIT OF MULTIPLES
   - PRODUCT HAS $2n$ BITS

2. THE MULTIPLE $xy_{n-1}2^{n-1}$ SUBTRACTED INSTEAD OF ADDED

$$y = -y_{n-1}2^{n-1} + \sum_{i=0}^{n-2} y_i 2^i$$

   - COMPLEMENT AND ADD

3. RECODE THE (2'S COMPLEMENT) MULTIPLIER INTO THE DIGIT SET {-1,0,1}
   - NO ADVANTAGE IN FOLLOWING THIS APPROACH

BIT-MATRIX IN RADIX-2 2'S COMPLEMENT MULTIPLIER

- Simplification of sign extension based on

$$(-s) + 1 - 1 = (1 - s) - 1 = s' - 1$$

Consequently,

$$\begin{array}{cccc} x_{n-1}y_i & x_{n-2}y_i & \dots & x_0y_i \end{array}$$

is replaced by

$$\begin{array}{cccc} (x_{n-1}y_i)' & x_{n-2}y_i & \dots & x_0y_i \\ \text{-1} & & & \end{array}$$

$$
\begin{array}{cccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
x_3y_0 & x_3y_0 & x_3y_0 & x_3y_0 & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 \\
x_3y_1 & x_3y_1 & x_3y_1 & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & \\
x_3y_2 & x_3y_2 & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & & \\
x'_3y_3 & x'_3y_3 & x'_2y_3 & x'_1y_3 & x'_0y_3 & & & \\
 & & & & y_3 & & &
\end{array}
$$

(a)

$$
\begin{array}{cccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
 & & & & (x_3y_0)' & x_2y_0 & x_1y_0 & x_0y_0 \\
 & & & (x_3y_1)' & x_2y_1 & x_1y_1 & x_0y_1 & \\
 & & (x_3y_2)' & x_2y_2 & x_1y_2 & x_0y_2 & & \\
 & (x'_3y_3)' & x'_2y_3 & x'_ly_3 & x'_0y_3 & & & \\
 & & & & y_3 & & & \\
0 & -1 & -1 & -1 & -1 & & &
\end{array}
$$

(b)

$$
\begin{array}{cccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
 & & & y_3 & (x_3y_0)' & x_2y_0 & x_1y_0 & x_0y_0 \\
 & & & (x_3y_1)' & x_2y_1 & x_1y_1 & x_0y_1 & \\
 & & (x_3y_2)' & x_2y_2 & x_1y_2 & x_0y_2 & & \\
1 & (x'_3y_3)' & x'_2y_3 & x'_ly_3 & (x_0y_3)' & & &
\end{array}
$$

(c)

Figure 4.7: Constructing bit-matrix for two's complement multiplier ($n = 4$).

RADIX-4 MULTIPLICATION

- REDUCE NUMBER OF STEPS TO $n/2$
- PARALLEL OR SEQUENTIAL RECODING
- TWO CASES
  1. BIT ARRAY ADDED BY A LINEAR ARRAY OF ADDERS
     – SEQUENTIAL RECODING INTO {-1,0,1,2} SUFFICIENT
  2. BIT ARRAY ADDED BY A TREE OF ADDERS
     – PARALLEL RECODING INTO {-2,-1,0,1,2} REQUIRED

PARALLEL RADIX-4 RECODING

- RADIX-2 MULTIPLIER

$$y_{n-1}, y_{n-2} \cdots, y_1, y_0$$

y_i – multiplier bit; $v_j \in \{0, 1, 2, 3\}$ – radix-4 multiplier digit

$$v_j = 2y_{2j+1} + y_{2j} \qquad j = (\frac{n}{2} - 1, ..., 0)$$

- RECODING ALGORITHM

1. Obtain w_j and t_{j+1} such that

$$v_j = w_j + 4t_{j+1}$$

2. Obtain

$$z_j = w_j + t_j$$

- TO AVOID CARRY PROPAGATION:

$$-2 \leq w_j \leq 1 \quad 0 \leq t_{j+1} \leq 1$$

PARALLEL RADIX-4 RECODING ALGORITHM

$$(t_{j+1},\ w_j) = \begin{cases} (0,\ v_j) & \text{if } v_j \leq 1 \\ (1,\ v_j - 4) & \text{if } v_j \geq 2 \end{cases}$$

$$z_j = w_j + t_j$$

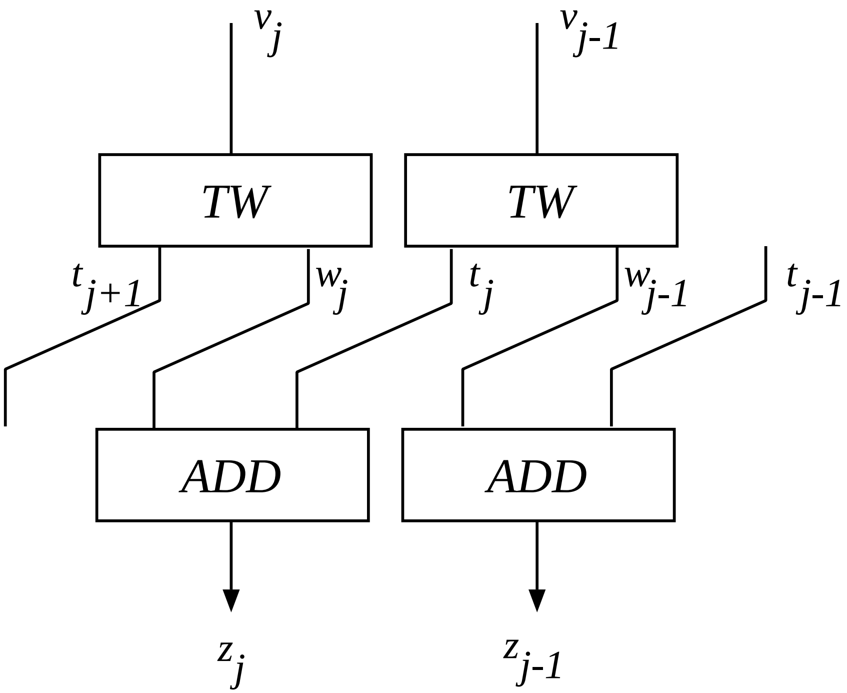


Figure 4.8: RADIX-4 PARALLEL RECODING FROM {0,1,2,3} INTO {-2,-1,0,1,2}.

BIT-LEVEL IMPLEMENTATION

- radix-2 multiplier

$$Y = (y_{n-1}, y_{n-2}, \ldots, y_0) \quad y_i \in \{0,1\}$$

- recoded radix-4 multiplier

$$Z = (z_{m-1}, z_{m-2}, \ldots, z_0) \quad z_i \in \{-2,-1,0,1,2\}$$

y_{2j+1}	y_{2j}	y_{2j-1}	z_j
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

EXAMPLES OF RECODING

$$y = 01011110 \quad y = 10001101$$

$$z = 1 \;\; 2 \;\; 0 \;\; \overline{2} \quad z = \overline{2} \;\; 1 \;\; \overline{1} \;\; 1$$

RECODER IMPLEMENTATION

- $sign = 1$ if z_j is negative
- $one = 1$ if z_j is either 1 or -1
- $two = 1$ if z_j is either 2 or -2.

$$\begin{aligned} sign &= y_{2j+1} \\ one &= y_{2j} \oplus y_{2j-1} \\ two &= y_{2j+1}y'_{2j}y'_{2j-1} + y'_{2j+1}y_{2j}y_{2j-1} \end{aligned}$$

- carry-in: $c = sign$

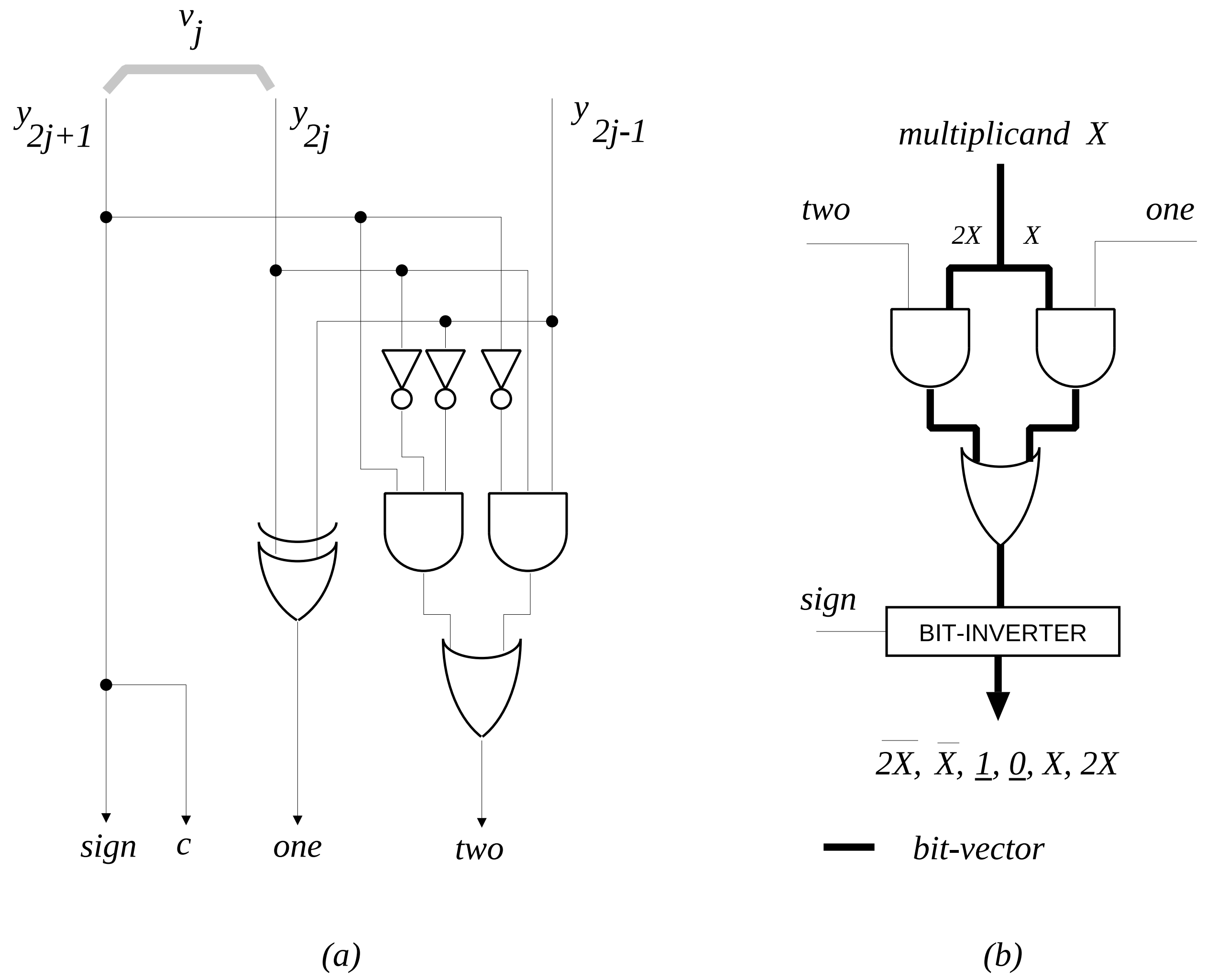


Figure 4.9: (a) IMPLEMENTATION OF RECODER. (b) IMPLEMENTATION OF MULTIPLE GENERATOR.

$$
\begin{array}{lcccccccccccccc}
 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
xz_0: & s_e & s_e & s_e & s_e & s_e & s_e & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} \\
xz_1: & s_f & s_f & s_f & s_f & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & & c_e \\
xz_2: & s_g & s_g & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & & c_f & & \\
xz_3: & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & & c_g & & & &
\end{array}
$$

(a)

$$
\begin{array}{lcccccccccccccc}
xz_0: & & & & & & s'_e & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} \\
xz_1: & & & & s'_f & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & & c_e \\
xz_2: & & s'_g & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & & c_f & & \\
xz_3: & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & & c_g & & & & \\
 & & \text{-1} & & \text{-1} & & \text{-1} & & & & & & & &
\end{array}
$$

(b)

$$
\begin{array}{lcccccccccccccc}
xz_0: & 1 & & 1 & s'_e & s_e & s_e & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} & \mathrm{e} \\
xz_1: & & & & s'_f & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & \mathrm{f} & & c_e \\
xz_2: & & s'_g & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & \mathrm{g} & & c_f & & \\
xz_3: & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & \mathrm{h} & & c_g & & & &
\end{array}
$$

(c)

Figure 4.10: RADIX-4 BIT-MATRIX FOR MULTIPLICATION OF MAGNITUDES ($n = 7$).

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
xz_0:	s_e	s_e	s_e	s_e	s_e	s_e	s_e	s_e	e	e	e	e	e	e	e	e
xz_1:	s_f	s_f	s_f	s_f	s_f	s_f	f	f	f	f	f	f	f	f		c_e
xz_2:	s_g	s_g	s_g	s_g	g	g	g	g	g	g	g	g		c_f		
xz_3:	s_h	s_h	h	h	h	h	h	h	h	h		c_g				
										c_h						

(a)

xz_0:								s'_e	e	e	e	e	e	e	e	e
xz_1:						s'_f	f	f	f	f	f	f	f	f		c_e
xz_2:				s'_g	g	g	g	g	g	g	g	g		c_f		
xz_3:		s'_h	h	h	h	h	h	h	h	h		c_g				
		-1		-1		-1		-1		c_h						

(b)

xz_0:	1		1		1	s'_e	s_e	s_e	e	e	e	e	e	e	e	e
xz_1:						s'_f	f	f	f	f	f	f	f	f		c_e
xz_2:				s'_g	g	g	g	g	g	g	g	g		c_f		
xz_3:		s'_h	h	h	h	h	h	h	h	h		c_g				
										c_h						

(c)

Figure 4.11: RADIX-4 BIT-MATRIX FOR 2'S COMPLEMENT MULTIPLICATION ($n = 8$).

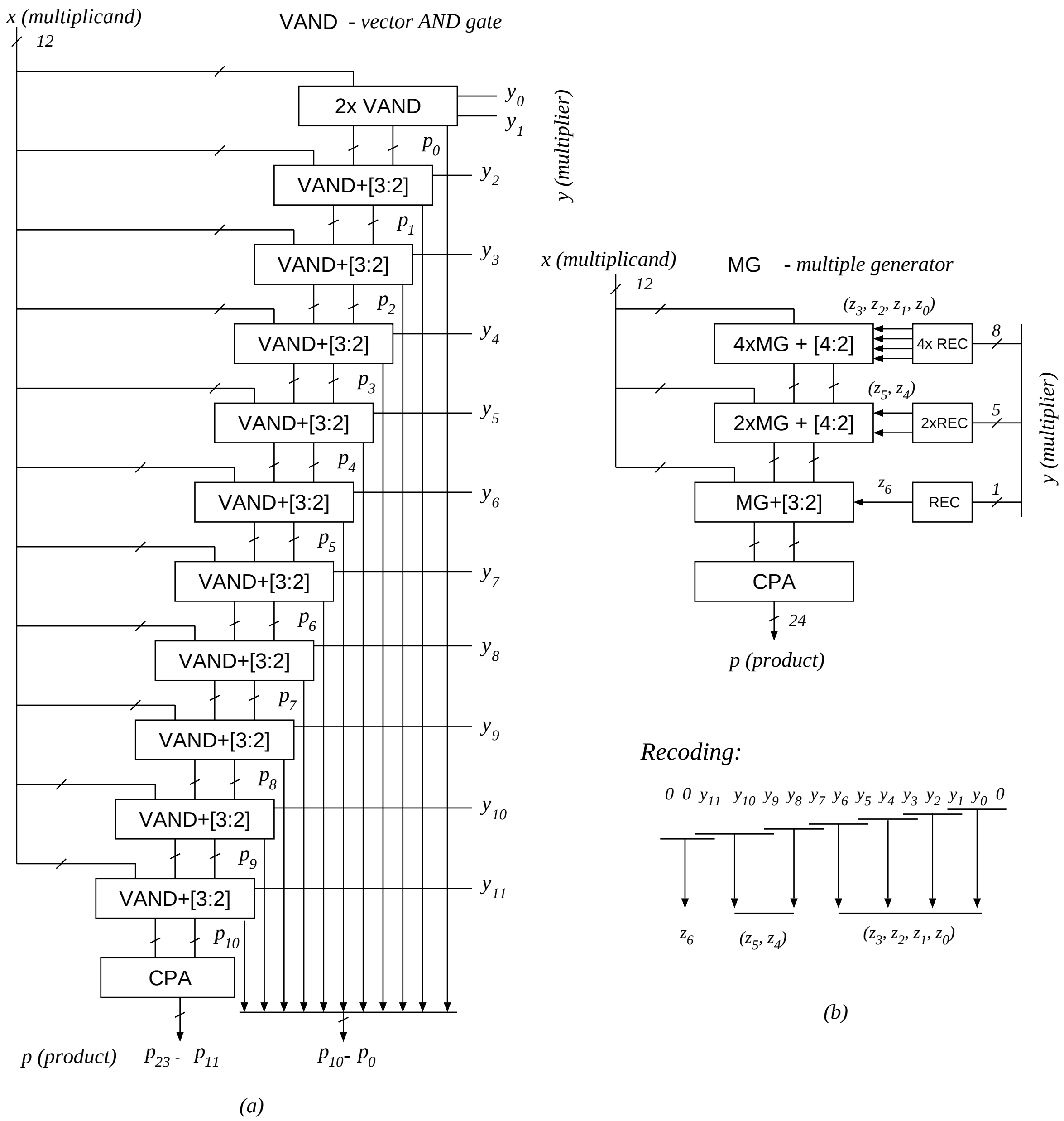


Figure 4.12: LINEAR CSA ARRAY FOR (a) $r = 2$. (b) $r = 4$.

DELAY OF LINEAR ARRAY MULTIPLIERS

- For radix 2,

$$T = t_{AND} + (n-2)t_{fa} + t_{(cpa,(n+1))}$$

- For radix 4

$$T = t_{rec} + t_{AND-OR} + (\frac{n}{2} - 2)t_{fa} + t_{(cpa,n)}$$

REDUCTION BY ROWS: ADDER TREES

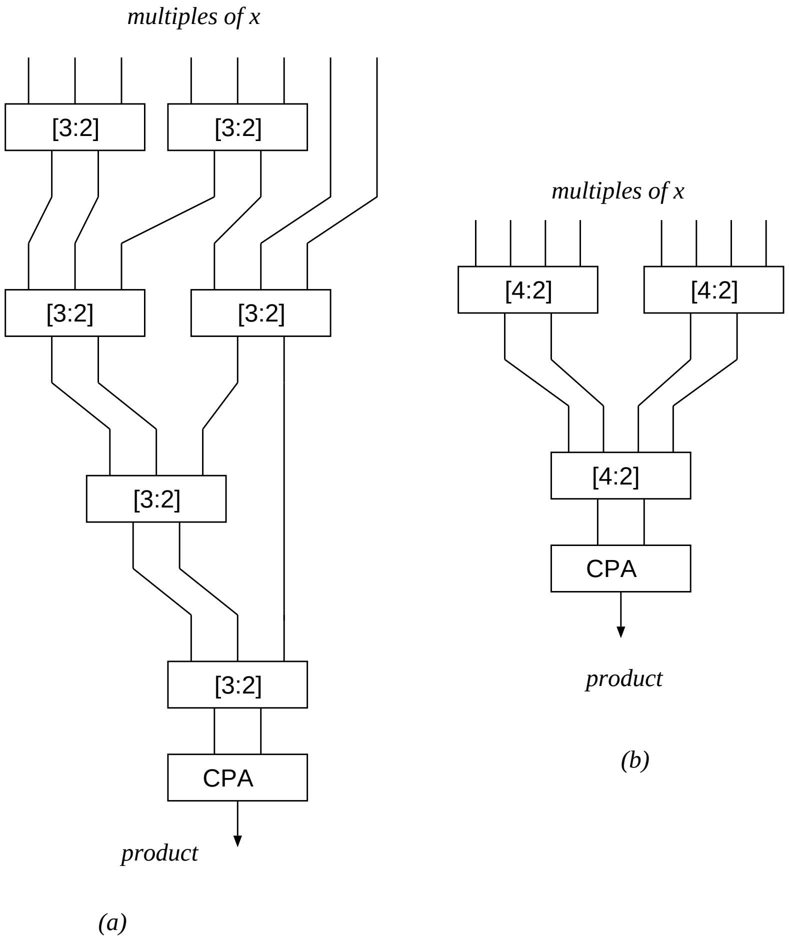


Figure 4.13: TREE ARRAYS OF ADDERS: a) with [3:2] adders. b) with [4:2] adders.

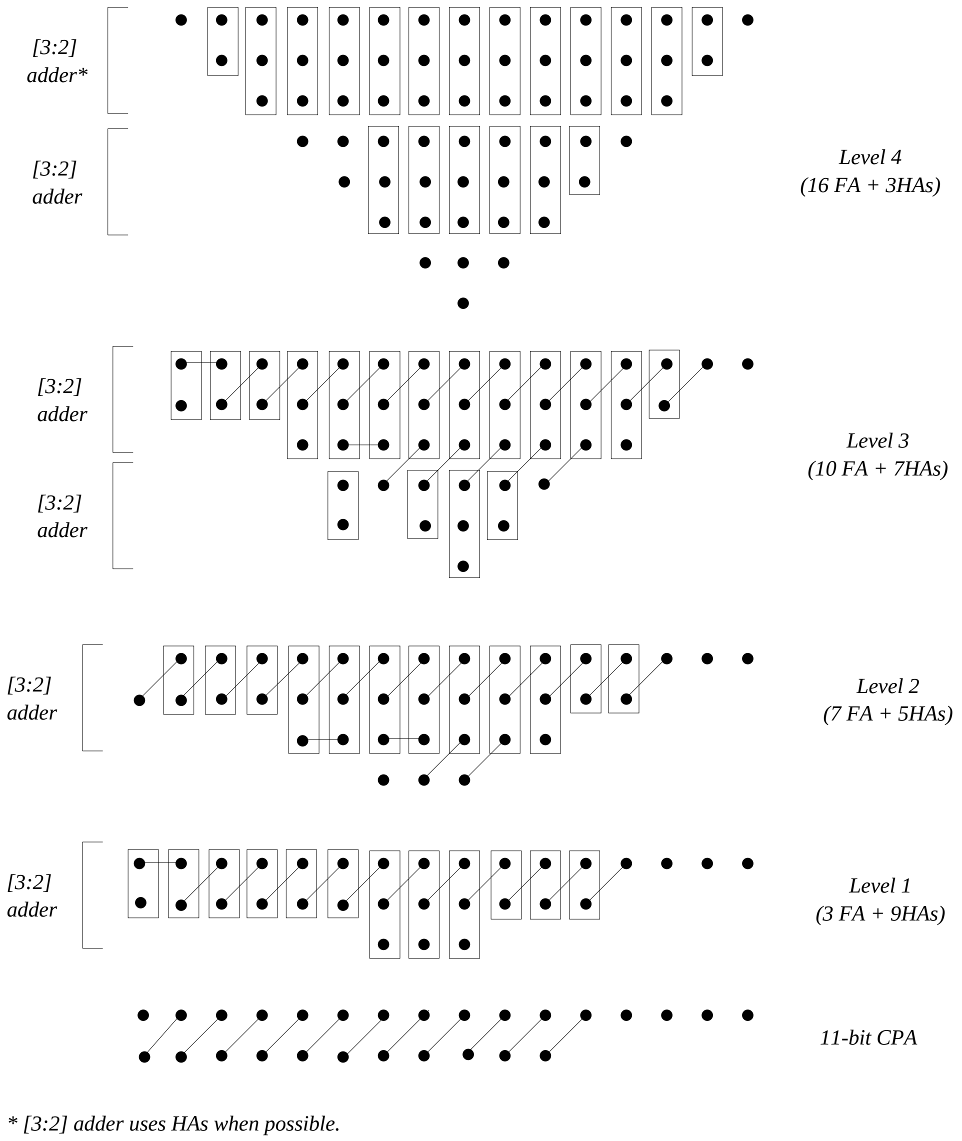


Figure 4.14: REDUCTION BY ROWS USING FAs AND HAs ($n = 8$): Cost 36 FAs, 24 HAs, 11-bit CPA.

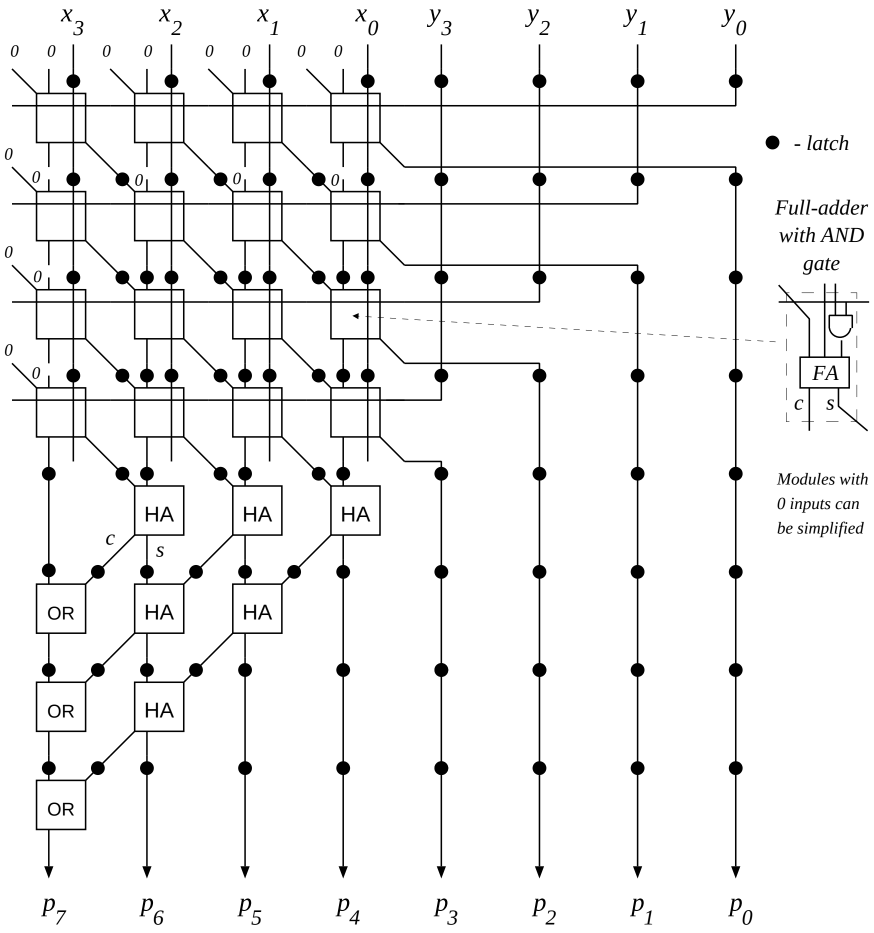


Figure 4.15: PIPELINED LINEAR CSA MULTIPLIER FOR POSITIVE INTEGERS ($n = 4$)

REDUCTION BY COLUMNS USING $(p, q]$ COUNTERS

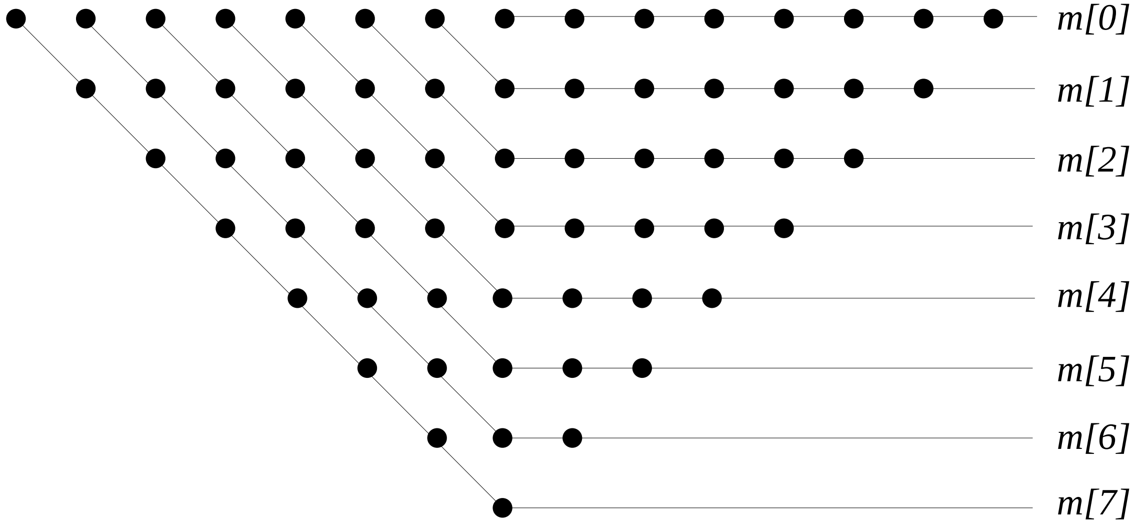


Figure 4.16: BITS OF MULTIPLES ORGANIZED AS BIT-TRIANGLE.

Table 4.3: Reduction by columns using FAs and HAs for 8x8 radix-2 magnitude multiplier.

	i														
	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$l = 4$															
e_i	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
m_3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
h_i	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
f_i	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
$l = 3$															
e_i	1	2	3	4	6	6	6	6	6	6	5	4	3	2	1
m_2	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
h_i	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
f_i	0	0	0	1	2	2	2	2	2	1	0	0	0	0	0
$l = 2$															
e_i	1	2	4	4	4	4	4	4	4	4	4	4	3	2	1
m_1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
h_i	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
f_i	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0
$l = 1$															
e_i	1	3	3	3	3	3	3	3	3	3	3	3	3	2	1
m_0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
h_i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_i	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0
CPA	2	2	2	2	2	2	2	2	2	2	2	2	1	2	1

e_i is the number of inputs in column i; f_i is the number of FAs; h_i is the number of HAs; m_j is the number of operands in the next level in the reduction sequence.

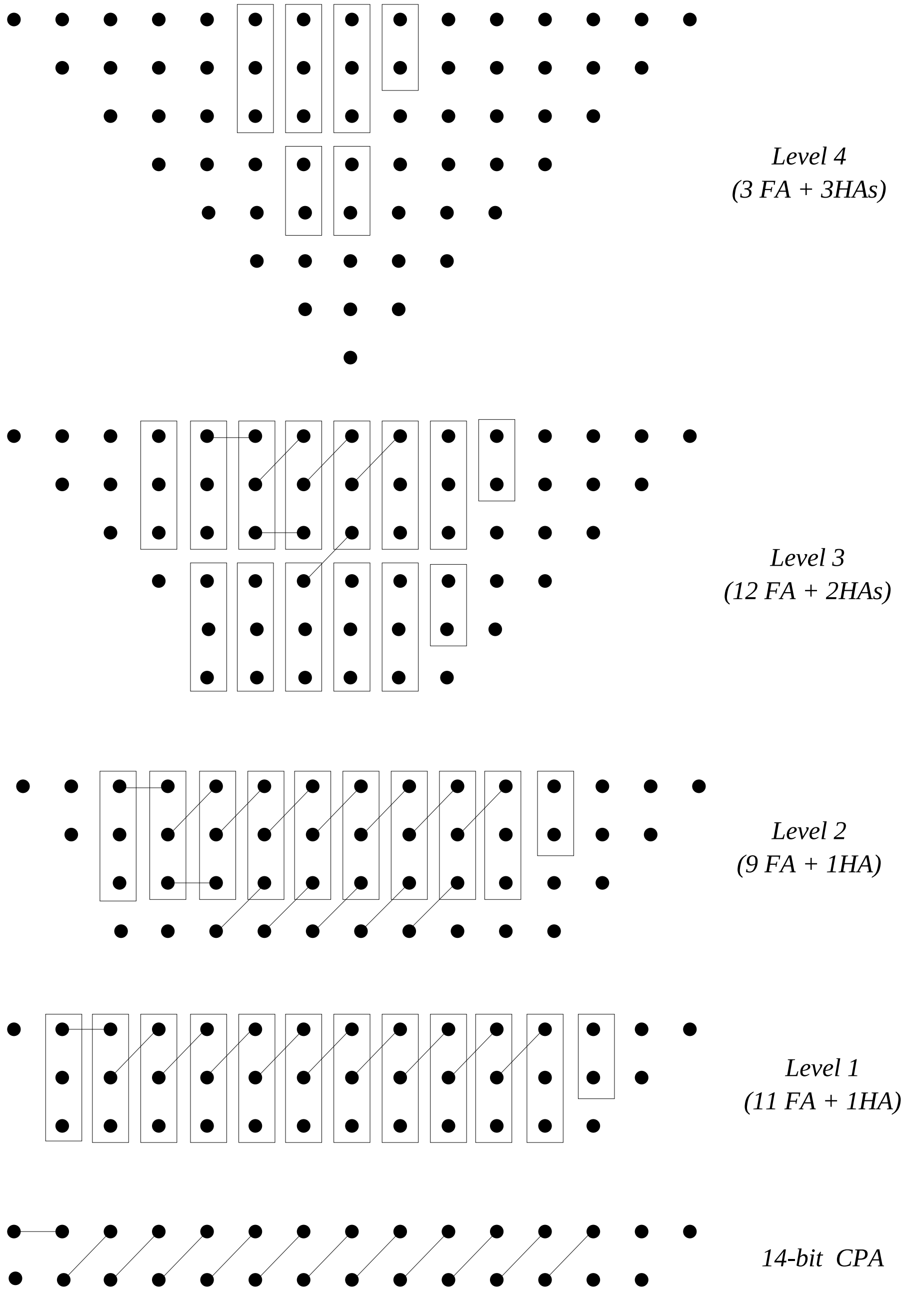


Figure 4.17: REDUCTION BY COLUMNS USING FAs and HAs ($n = 8$): Cost 35 FAs, 7 HAs, 14-bit CPA.

FINAL ADDER

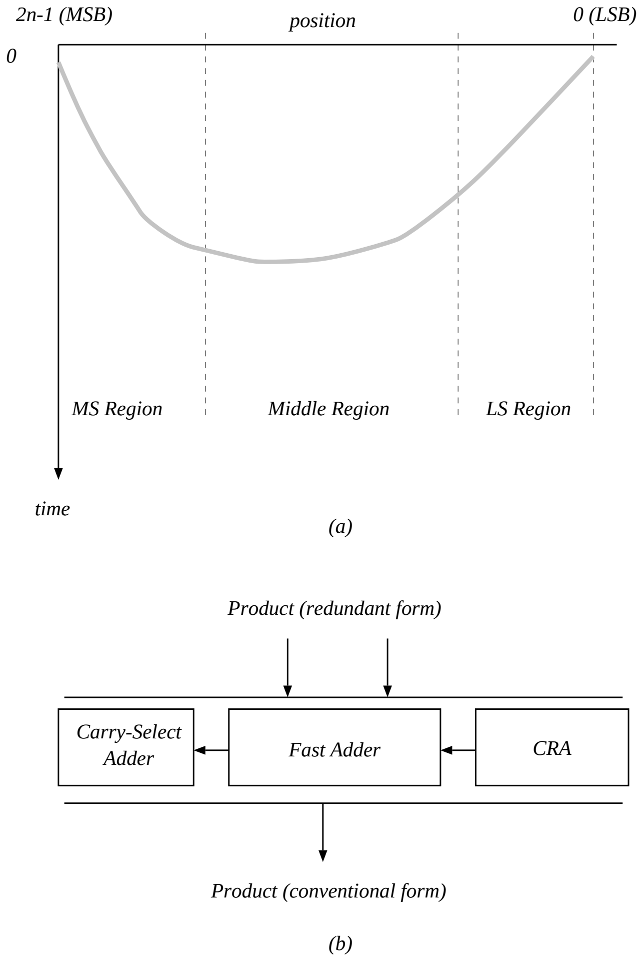


Figure 4.18: Final adder: (a) Arrival time of the inputs to the final adder. (b) Hybrid final adder.

PARTIALLY COMBINATIONAL IMPLEMENTATION

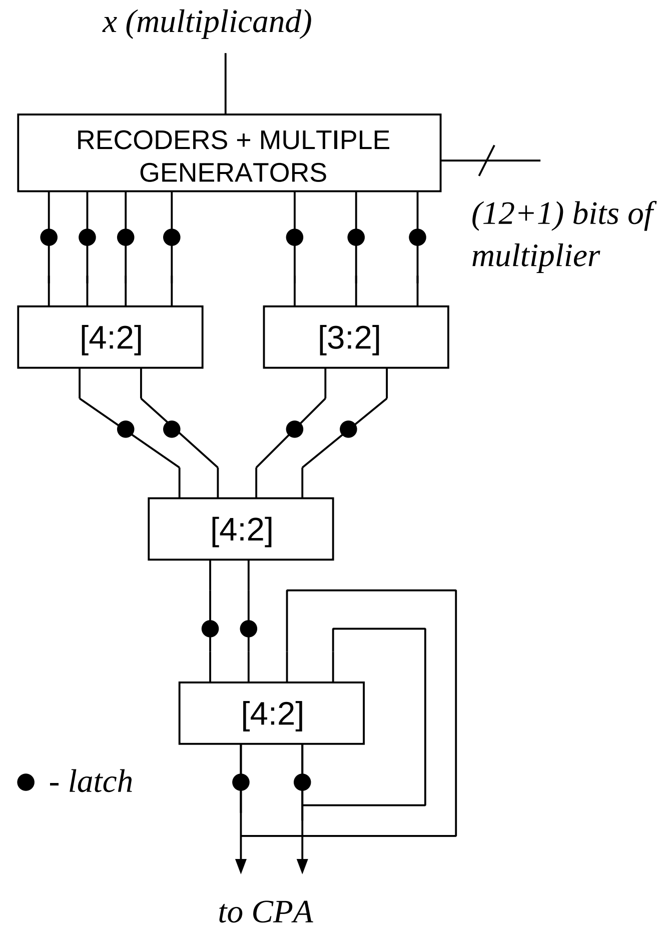


Figure 4.19: RADIX 2^{12} SEQUENTIAL MULTIPLIER USING CSA TREE.

ARRAYS OF SMALLER MULTIPLIERS

$$p = a \times b$$

$$\begin{aligned} A &= (a_{k-1}, a_{k-2}, \ldots, a_0) \\ B &= (b_{l-1}, b_{l-2}, \ldots, b_0) \\ P &= (p_{k+l-1}, p_{k+l-2}, \ldots, p_0) \end{aligned}$$

- USE OF $k \times l$ MODULES
- OPERANDS DECOMPOSED INTO DIGITS OF RADIX 2^k AND 2^l

$$x = \sum_{i=0}^{(n/k)-1} x^{(i)} 2^{ki}$$

$$y = \sum_{j=0}^{(n/l)-1} y^{(j)} 2^{lj}$$

$$\begin{aligned} p = x \cdot y &= \sum_{i=0}^{(n/k)-1} x^{(i)} 2^{ki} \times \sum_{j=0}^{(n/l)-1} y^{(j)} 2^{lj} \\ &= \sum x^{(i)} y^{(j)} 2^{ki+lj} = \sum p^{(i,j)} 2^{ki+lj} \end{aligned}$$

- $(n/k) \times (n/l)$ MODULES NEEDED

EXAMPLE 12×12 USING 4×4 MODULES

$$x = a_x 2^8 + b_x 2^4 + c_x$$
$$y = a_y 2^8 + b_y 2^4 + c_y$$

$$x \times y = a_x a_y 2^{16} + a_x b_y 2^{12} + b_x a_y 2^{12} + a_x c_y 2^8 + b_x b_y 2^8 + c_x a_y 2^8 + b_x c_y 2^4 + c_x b_y 2^4 + c_x c_y$$

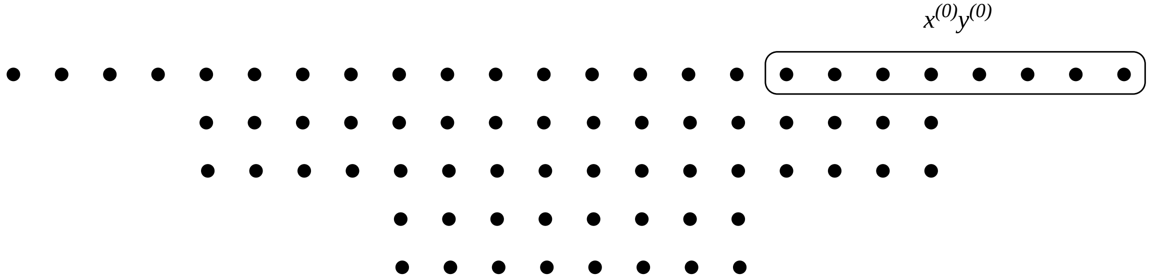


Figure 4.20: 12×12 MULTIPLICATION USING 4×4 MULTIPLIERS: BIT MATRIX.

MULTIPLY-ADD AND MULTIPLY-ACCUMULATE (MAC)

- Multiply-add: $S = X \times Y + W$

	13	12	11	10	9	8	7	6	5	4	3	2	1	0
xz_0:				s'_e	s_e	s_e	e	e	e	e	e	e	e	e
xz_1:			1	s'_f	f	f	f	f	f	f	f	f		c_e
xz_2:	1	s'_g	g	g	g	g	g	g	g	g		c_f		
xz_3:	h	h	h	h	h	h	h	h		c_g				
w:								w	w	w	w	w	w	w

Figure 4.21: Radix-4 bit-matrix for multiply-add of magnitudes ($n = 7$). z_i's are radix-4 digits obtained by multiplier recoding.

- Multiply-accumulate:

$$S = \sum_{i=1}^{m} X[i] \times Y[i]$$

$$S[i+1] = X[i] \times Y[i] + S[i]$$

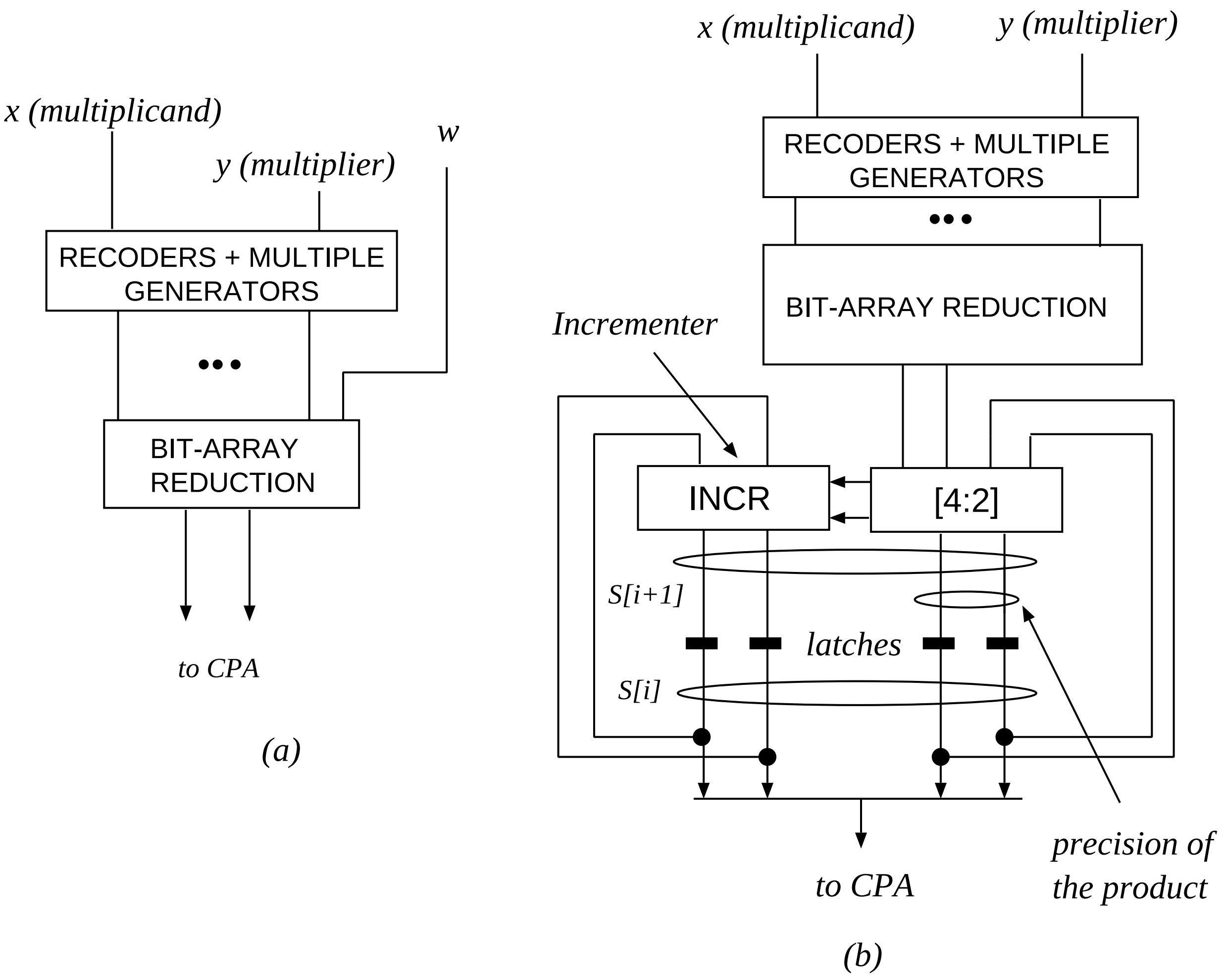


Figure 4.22: Block-diagrams of: (a) Multiply-add unit. (b) Multiply-accumulate unit.

SATURATING MULTIPLIERS

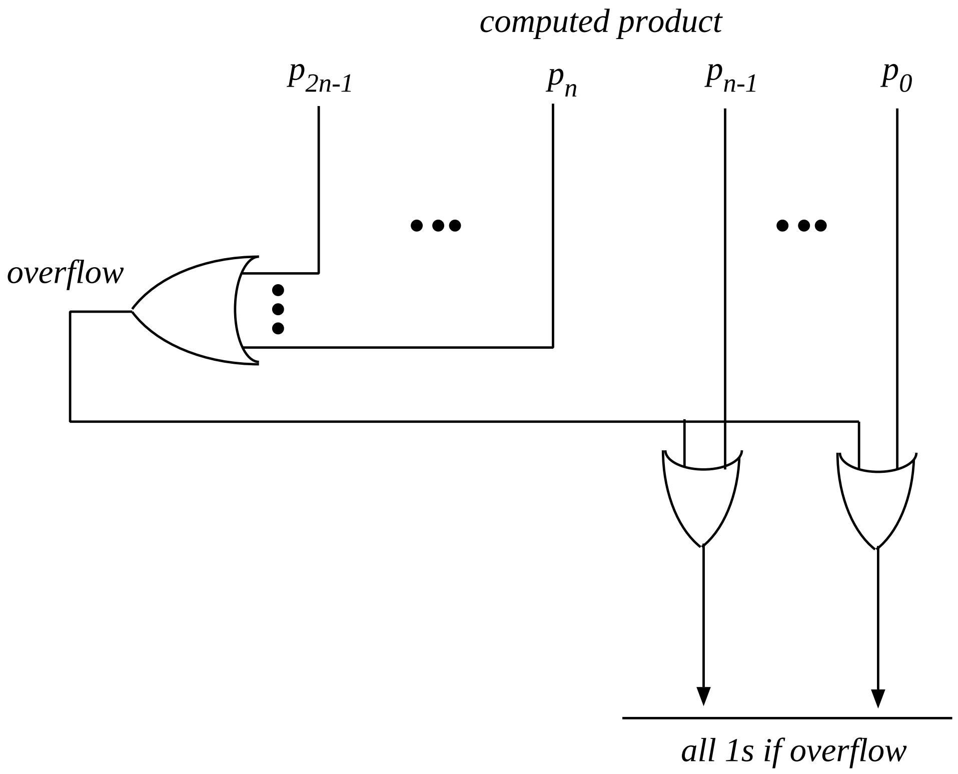


Figure 4.23: Detection and result setting for multiplication of magnitudes.

TRUNCATING MULTIPLIERS

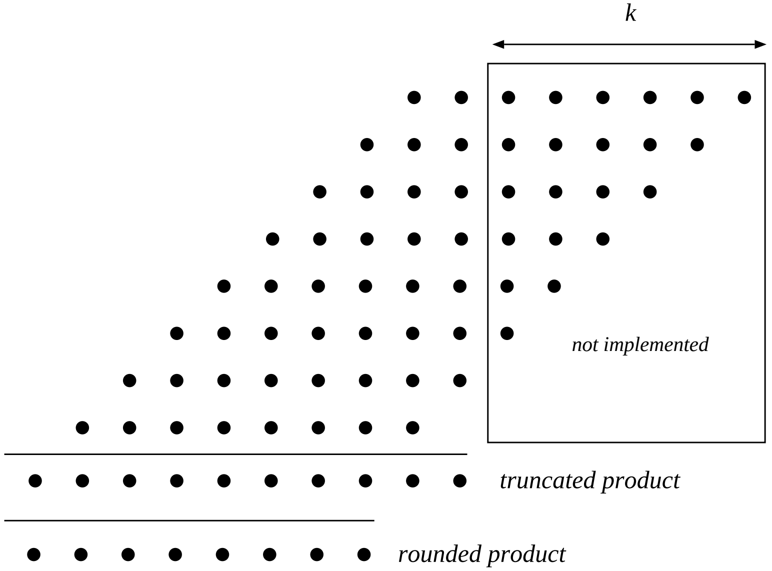


Figure 4.24: Bit-matrix of a truncated magnitude multiplier.

11	10	9	8	7	6	5	4	3	2	1	0
						x_5x_0	x_4x_0	x_3x_0	x_2x_0	x_1x_0	x_0x_0
					x_5x_1	x_4x_1	x_3x_1	x_2x_1	x_1x_1	x_0x_1	
				x_5x_2	x_4x_2	x_3x_2	x_2x_2	x_1x_2	x_0x_2		
			x_5x_3	x_4x_3	x_3x_3	x_2x_3	x_1x_3	x_0x_3			
		x_5x_4	x_4x_4	x_3x_4	x_2x_4	x_1x_4	x_0x_4				
	x_5x_5	x_4x_5	x_3x_5	x_2x_5	x_1x_5	x_0x_5					
	x_5x_4	x_5x_3	x_5x_2	x_5x_1	x_5x_0	x_4x_0	x_3x_0	x_2x_0	x_1x_0		x_0
	x_5		x_4x_3	x_4x_2	x_4x_1	x_3x_1	x_2x_1		x_1		
			x_4		x_3x_2		x_2				
					x_3						

(a)

11	10	9	8	7	6	5	4	3	2	1	0
	x_5x_4	x_5x_3	x_5x_2	x_5x_1	x_5x_0	x_4x_0	x_3x_0	x_2x_0	x_1x_0		x_0
	x_5		x_4x_3	x_4x_2	x_4x_1	x_3x_1	x_2x_1		x_1		
			x_4	x_3x_2	x_3x_2'		x_2				

(b)

Figure 4.25: Bit-array simplification in squaring of magnitudes ($n = 6$).

CONSTANT AND MULTIPLE CONSTANT MULTIPLIERS

$P = X \times C$, C - constant

DONE AS

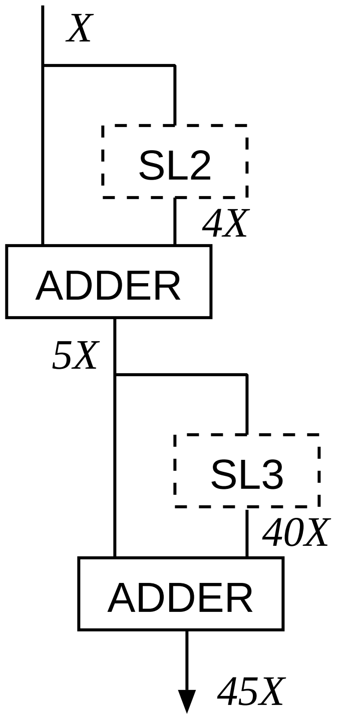
$P = \Sigma_j \, X \times C_j 2^j$
$\{j\}$ corresponds to 1's in binary representation of C

- ADDERS ONLY
- HOW TO REDUCE NUMBER OF ADDERS?

1. Recode to radix 4: max $n/2-1$ adders
2. Apply canonical recoding: $n/3$ adders avg, $n/2-1$ max
3. Decomposition and sharing of subexpressions
4. Multiple constant multiplication - further reductions

CONSTANT MULT. (cont.)

$$45X \ = \ 5X \times 9 = X(2^2+1)(2^3+1)$$



SLk - shift left k positions

Figure 4.26: Implementation of $P = X \times C$ for $C = 45$ using common subexpressions.

MULTIPLE CONSTANT MULTIPLICATION

COMPUTE

$P_1 = 9X = 5X + 4X$

$P_2 = 13X = 5X + 8X$

$P_3 = 18X = 2 \times 9X$

$P_4 = 21X = 5X + 16X$

- WITH SEPARATE CONSTANT MULTIPLIERS: 6 ADDERS

- BY SHARING SUBEXPRESSIONS: 4 ADDERS

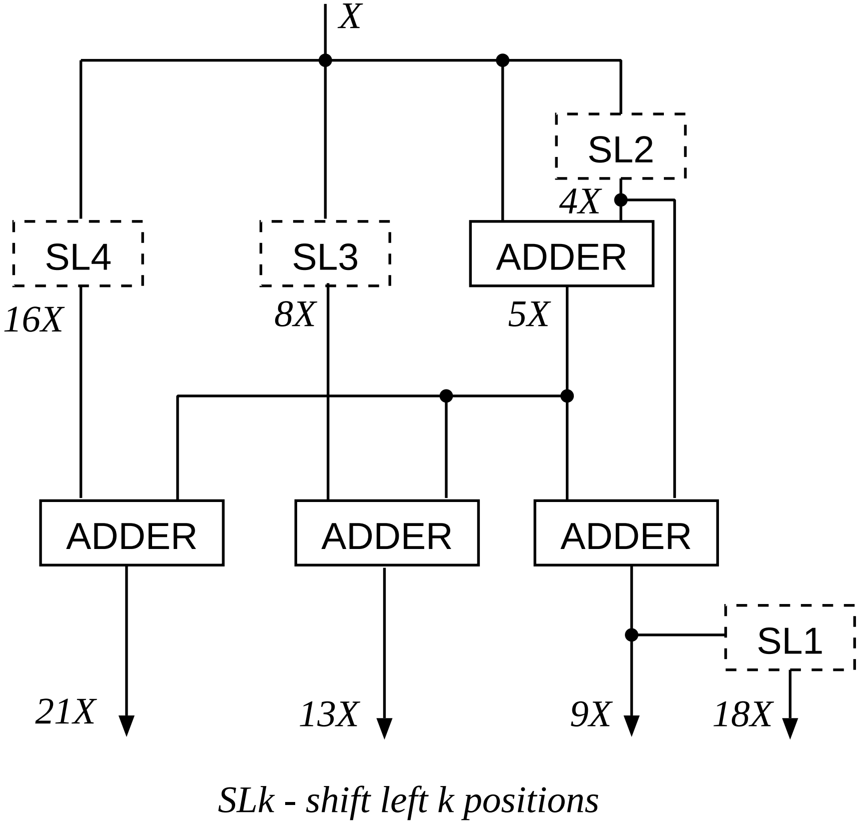


Figure 4.27: An example of multiple constants multipliers.

DIVISION BY DIGIT RECURRENCE

SEVERAL DIVISION METHODS:

- DIGIT-RECURRENCE METHOD – studied in this chapter
- MULTIPLICATIVE METHOD (Chapter 7)
- VARIOUS APPROXIMATION METHODS (power series expansion),
- SPECIAL METHODS SUCH AS CORDIC (Chapter 11) AND CONTINUED PRODUCT METHODS.

IMPLEMENTATIONS:

- SEQUENTIAL
- COMBINATIONAL
  1. PIPELINED
  2. NONPIPELINED
- COMBINATIONAL/SEQUENTIAL

DEFINITION AND NOTATION

$$x = q \cdot d + rem$$

$$|rem| < |d| \cdot ulp \quad \text{and} \quad sign(rem) = sign(x)$$

DIVIDEND x
DIVISOR d
QUOTIENT q
REMAINDER rem

- INTEGER QUOTIENT: $ulp = 1$,
- FRACTIONAL QUOTIENT: $ulp = r^{-n}$

TWO TYPES OF DIVISION OPERATION:

1. INTEGER DIVISION, WITH INTEGER OPERANDS AND RESULT

   USUALLY REQUIRES AN EXACT REMAINDER

2. FRACTIONAL DIVISION

   TO AVOID QUOTIENT OVERFLOW: $x < d$

   QUOTIENT ROUNDED, WHICH CAN RESULT IN A NEGATIVE REMAINDER

FRACTIONAL DIVISION - ASSUMPTIONS

- OPERANDS/RESULT IN SIGN-AND-MAGNITUDE FORMAT $\Longrightarrow$CONSIDER MAGNITUDES ONLY

$$1/2 \leq d < 1; \quad x < d; \quad 0 < q < 1$$

- FOR SIMPLER SELECTION: $q_j \in \mathcal{D}_a = \{-a, -a+1, \ldots, -1, 0, 1, \ldots, a-1, a\}$
- REDUNDANCY FACTOR

$$\rho \;=\; \frac{a}{r-1}, \quad \frac{1}{2} < \rho \leq 1$$

RECURRENCE

- QUOTIENT AFTER j STEPS: $q[j] = q[0] + \Sigma_{i=1}^{j} q_i r^{-i}$
- FINAL QUOTIENT: $q = q[n] = q[0] + \Sigma_{i=1}^{n} q_i r^{-i}$
- FINAL QUOTIENT ERROR BOUND: $0 \leq \epsilon_q = \frac{x}{d} - q \ < \ r^{-n}$
- ERROR AT STEP j:

$$\epsilon[j] = \frac{x}{d} - q[j] \leq \epsilon[n] + \sum_{i=j+1}^{n} max(q_i) r^{-i} = \epsilon[n] + \frac{a}{r-1}(r^{-j} - r^{-n})$$

- FOR CONVERGENCE: $\epsilon[j] \leq \rho r^{-j}$
- FOR SIMPLER SELECTION, ALLOW NEGATIVE ERRORS

$$|\epsilon[j]| = |\frac{x}{d} - q[j]| \leq \rho r^{-j}$$

- ELIMINATE DIVISION FROM ERROR BOUND: $|x - dq[j]| \leq \rho d r^{-j}$

- DEFINE RESIDUAL (PARTIAL REMAINDER): $w[j] = r^j(x - dq[j])$
- RESIDUAL RECURRENCE: $w[j+1] = rw[j] - dq_{j+1} \qquad w[0] = x$

cont.

- BOUND ON $w[j]$: $|w[j]| \leq \rho d$

- SELECT QUOTIENT DIGIT TO KEEP $w[j+1]$ BOUNDED:

$$q_{j+1} = SEL(rw[j],\ d)$$

- REDUNDANCY IN QUOTIENT-DIGIT SET ALLOWS

$$q_{j+1} = SEL(\widehat{y},\ \widehat{d})$$

WHERE $\widehat{y}$ IS TRUNCATED $rw[j]$, etc.

IMPLEMENTATION OF DIVISION STEP

1. ONE DIGIT ARITHMETIC LEFT-SHIFT OF $w[j]$ TO PRODUCE $rw[j]$;

2. DETERMINATION OF THE QUOTIENT DIGIT q_{j+1} BY THE QUOTIENT-DIGIT SELECTION FUNCTION;

3. GENERATION OF THE DIVISOR MULTIPLE $d \times q_{j+1}$; and

4. SUBTRACTION OF dq_{j+1} from $rw[j]$.

5. UPDATE OF THE QUOTIENT $q[j]$ TO $q[j+1]$ BY THE ON-THE-FLY CONVERSION

$$q[j+1] = CONV(q[j], q_{j+1})$$

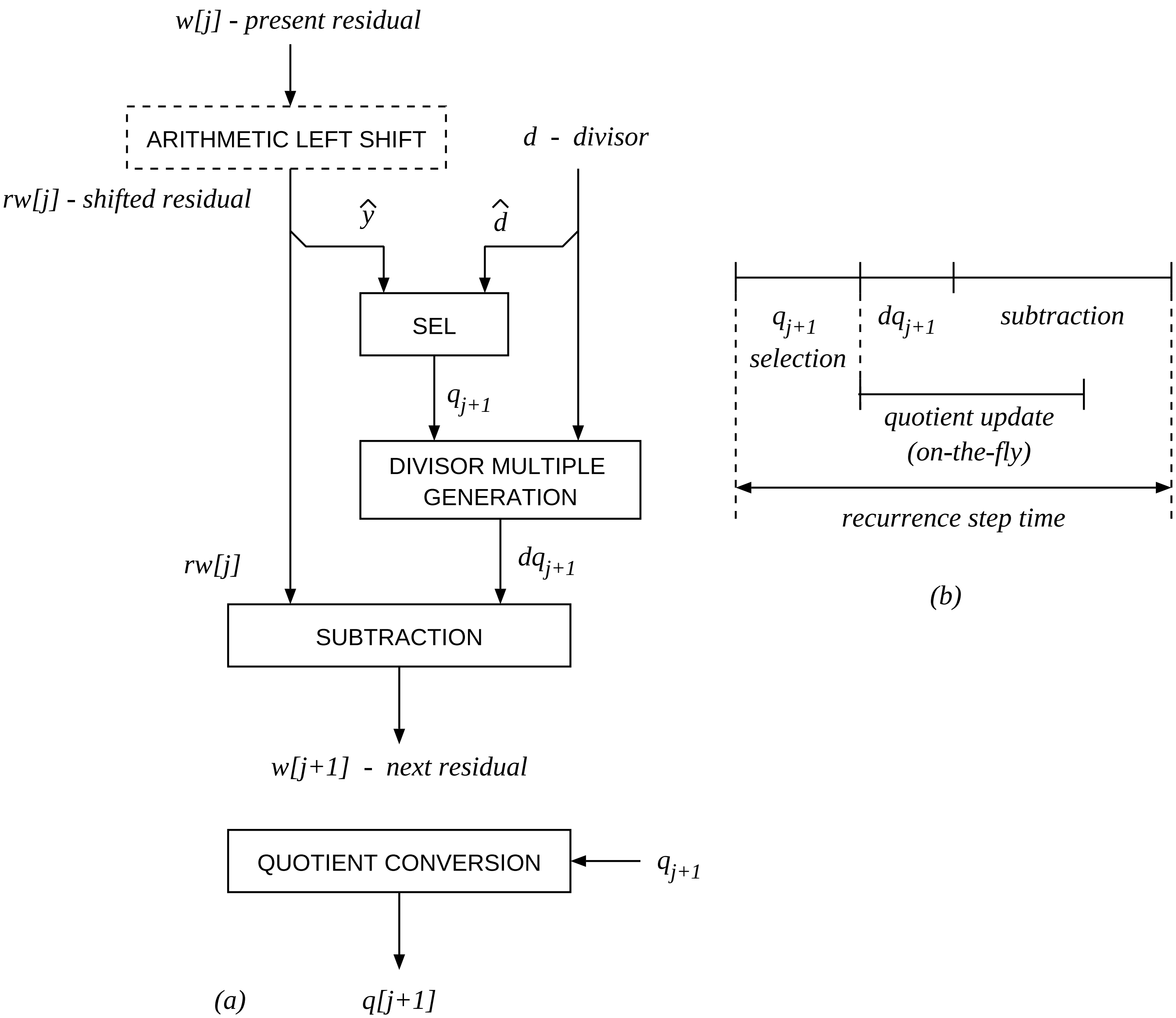


Figure 5.1: (a) COMPONENTS OF A DIVISION STEP. (b) TIMING.

MAIN FACTORS AFFECTING COST AND EXECUTION TIME

- RADIX r
- QUOTIENT-DIGIT SET

  1. CANONICAL: $0 \leq q_j \leq r-1$
  2. REDUNDANT: $q_j \in \mathcal{D}_a = \{-a, -a+1, \ldots, -1, 0, 1, \ldots, a-1, a\}$

- REDUNDANCY FACTOR: $\rho = \frac{a}{r-1}, \quad \rho > \frac{1}{2}$
- REPRESENTATION OF RESIDUAL:

  1. NONREDUNDANT (e.g., 2's complement)
  2. REDUNDANT: carry-save, signed-digit

- QUOTIENT-DIGIT SELECTION FUNCTION

INITIALIZATION AND TERMINATION

Initialization:

- $w[0] = x - dq[0]$ and $|w[0]| \leq \rho d$. Options:
  - Make $q[0] = 0$ and
    - $*$ For $\rho = 1$ we make $w[0] = x/2$.
    - $*$ For $1/2 < \rho < 1$ we make $w[0] = x/4$

    Compensated in the termination step
  - Make $q[0] = 1$ and $w[0] = x - d$. Applicable for $\rho < 1$ because $q > 1 + \rho$ not allowed.

Termination:

- QUOTIENT:

$$q = \begin{cases} q[N] & \text{if } w[N] \geq 0 \\ q[N] - r^{-N} & \text{if } w[N] < 0 \end{cases}$$

N – number of iterations
If dividend shifted in initialization - shift quotient (extra iteration)

ON-THE-FLY QUOTIENT CONVERSION

- j MS DIGITS OF CONVERTED QUOTIENT

$$Q[j] = \sum_{i=1}^{j} q_i r^{-i}$$

- UPDATE

$$Q[j+1] = Q[j] + q_{j+1} r^{-(j+1)}$$

- SINCE q_{j+1} CAN BE NEGATIVE:

$$Q[j+1] = \begin{cases} Q[j] + q_{j+1} r^{-(j+1)} & \text{if } \; q_{j+1} \geq 0 \\ Q[j] - r^{-j} + (r - |q_{j+1}|) r^{-(j+1)} & \text{if } \; q_{j+1} < 0 \end{cases}$$

- DISADVANTAGE: SUBTRACTION $Q[j] - r^{-j}$
  REQUIRES THE PROPAGATION OF A BORROW – SLOW

ON-THE-FLY CONVERSION (cont.)

- DEFINE ANOTHER FORM $QM[j]$

$$QM[j] = Q[j] - r^{-j}$$

- NEW CONVERSION ALGORITHM IS

$$Q[j+1] = \begin{cases} Q[j] + q_{j+1}r^{-(j+1)} & \text{if } q_{j+1} \geq 0 \\ QM[j] + (r - |q_{j+1}|)r^{-(j+1)} & \text{if } q_{j+1} < 0 \end{cases}$$

- SUBTRACTION REPLACED BY LOADING THE FORM $QM[j]$
- UPDATE FORM $QM[j]$ AS FOLLOWS:

$$\begin{aligned} QM[j+1] &= Q[j+1] - r^{-(j+1)} \\ &= \begin{cases} Q[j] + (q_{j+1} - 1)r^{-(j+1)} & \text{if } q_{j+1} > 0 \\ QM[j] + ((r-1) - |q_{j+1}|)r^{-(j+1)} & \text{if } q_{j+1} \leq 0 \end{cases} \end{aligned}$$

- ALL ADDITIONS ARE CONCATENATIONS

ON-THE-FLY CONVERSION ALGORITHM

$$Q[j+1] = \begin{cases} (Q[j], q_{j+1}) & \text{if } q_{j+1} \geq 0 \\ (QM[j], (r - |q_{j+1}|)) & \text{if } q_{j+1} < 0 \end{cases}$$

$$QM[j+1] = \begin{cases} (Q[j], q_{j+1} - 1) & \text{if } q_{j+1} > 0 \\ (QM[j], ((r-1) - |q_{j+1}|)) & \text{if } q_{j+1} \leq 0 \end{cases}$$

INITIAL CONDITIONS $Q[0] = QM[0] = 0$ (for a positive quotient)

EXAMPLE OF RADIX-2 CONVERSION

j	q_j	$Q[j]$	$QM[j]$
0		0	0
1	1	0.1	0.0
2	1	0.11	0.10
3	0	0.110	0.101
4	1	0.1101	0.1100
5	-1	0.11001	0.11000
6	0	0.110010	0.110001
7	0	0.1100100	0.1100011
8	-1	0.11000111	0.11000110
9	1	0.110001111	0.110001110
10	0	0.1100011110	0.1100011101
11	1	0.11000111101	0.11000111100
12	0	0.110001111010	0.110001111001

IMPLEMENTATION OF THE CONVERSION

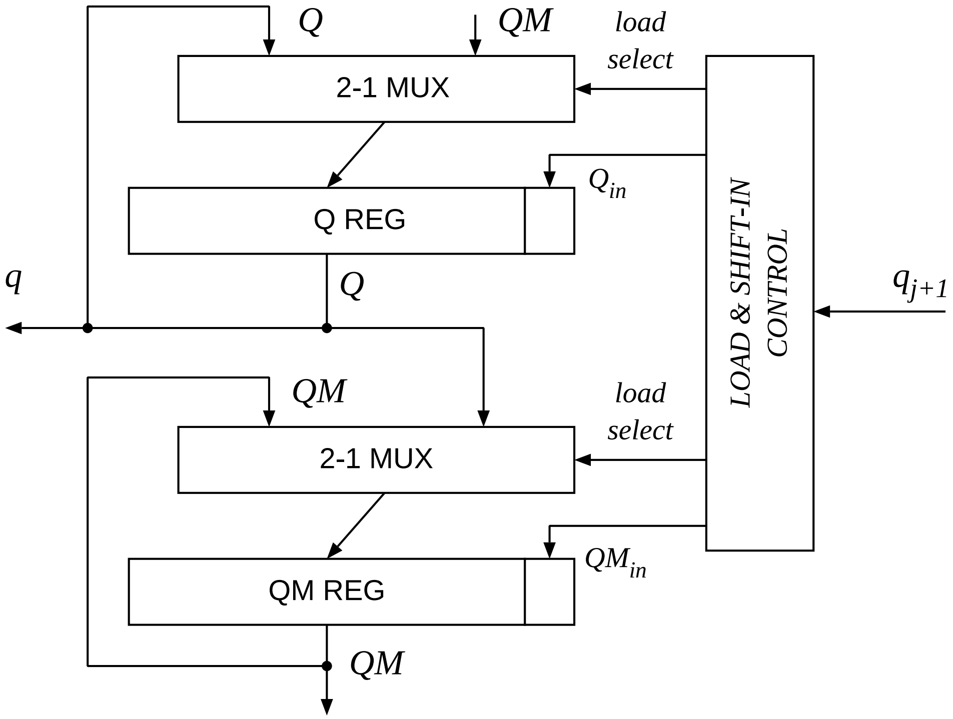


Figure 5.2: Implementation of on-the-fly conversion.

$$Q \leftarrow \begin{cases} shift\ Q\ with\ insert\ (Q_{in}) & \text{if } C_{shiftQ} = 1 \\ shift\ QM\ with\ insert\ (Q_{in}) & \text{if } C_{loadQ} = 1 \end{cases}$$

$$QM \leftarrow \begin{cases} shift\ QM\ with\ insert\ (QM_{in}) & \text{if } C_{shiftQM} = 1 \\ shift\ Q\ with\ insert\ (QM_{in}) & \text{if } C_{loadQM} = 1 \end{cases}$$

cont.

$$Q_{in} = \begin{cases} q_{j+1} & \text{if} \;\; q_{j+1} \geq 0 \\ r - |q_{j+1}| & \text{if} \;\; q_{j+1} < 0 \end{cases}$$

$$QM_{in} = \begin{cases} q_{j+1} - 1 & \text{if} \;\; q_{j+1} > 0 \\ (r-1) - |q_{j+1}| & \text{if} \;\; q_{j+1} \leq 0 \end{cases}$$

REGISTER CONTROL SIGNALS: $C_{loadQ} = C'_{shiftQ}$ and $C_{loadQM} = C'_{shiftQM}$

EXAMPLE OF RADIX-4 CONVERSION

q_{j+1}	Q_{in}	C_{shiftQ}	$Q[j+1]$	QM_{in}	$C_{shiftQM}$	$QM[j+1]$
3	3	1	$(Q[j],3)$	2	0	$(Q[j],2)$
2	2	1	$(Q[j],2)$	1	0	$(Q[j],1)$
1	1	1	$(Q[j],1)$	0	0	$(Q[j],0)$
0	0	1	$(Q[j],0)$	3	1	$(QM[j],3)$
-1	3	0	$(QM[j],3)$	2	1	$(QM[j],2)$
-2	2	0	$(QM[j],2)$	1	1	$(QM[j],1)$
-3	1	0	$(QM[j],1)$	0	1	$(QM[j],0)$

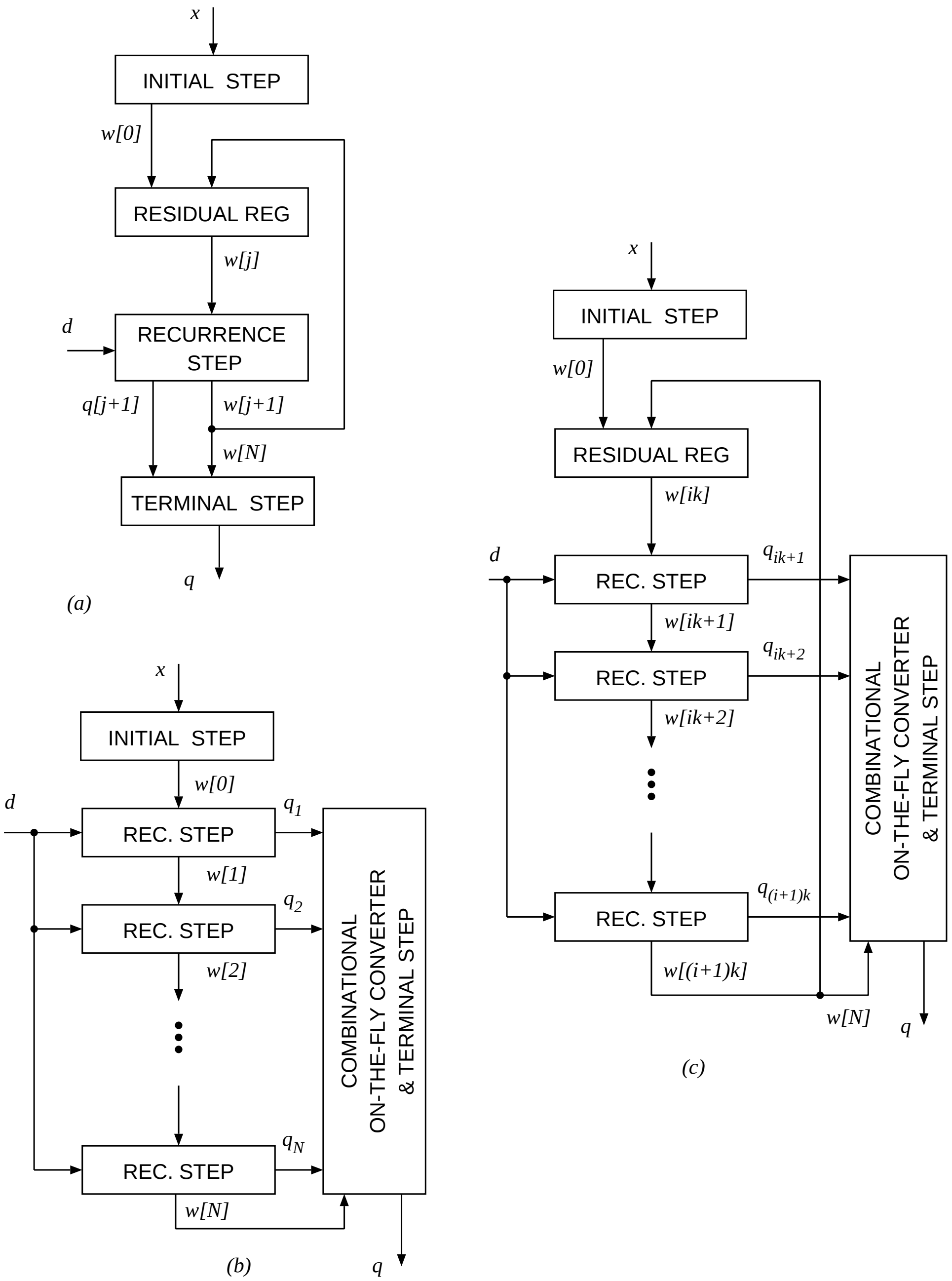


Figure 5.3: DIVISION IMPLEMENTATION: (a) TOTALLY SEQUENTIAL. (b) TOTALLY COMBINATIONAL. (c) COMBINED IMPLEMENTATION.

EXAMPLES OF ALGORITHMS AND IMPLEMENTATIONS

- CELLS: delay function of the load; delay and area in terms of 2-NAND.
- DEGREE OF OPTIMIZATION: the same modules have been used in all designs.
- INTERCONNECTIONS NOT INCLUDED: not considered the delay, area nor load of interconnections.
- EXECUTION TIME AND THE AREA FOR 53-BIT OPERANDS AND RESULT
- INCLUDED DELAY AND AREA OF REGISTERS

SCHEMES COMPARED

r2 Scheme Radix-2 with carry-save residual.

r4 Scheme Radix-4 with $a = 2$ and carry-save residual.

r8 Scheme Radix-8 with $a = 7$ and carry-save residual.

r16over Scheme Radix-16 with two overlapped radix-4 stages.

r512 Scheme Radix-512 with $a = 511$, carry-save residual, scaling, and quotient-digit selection by rounding.

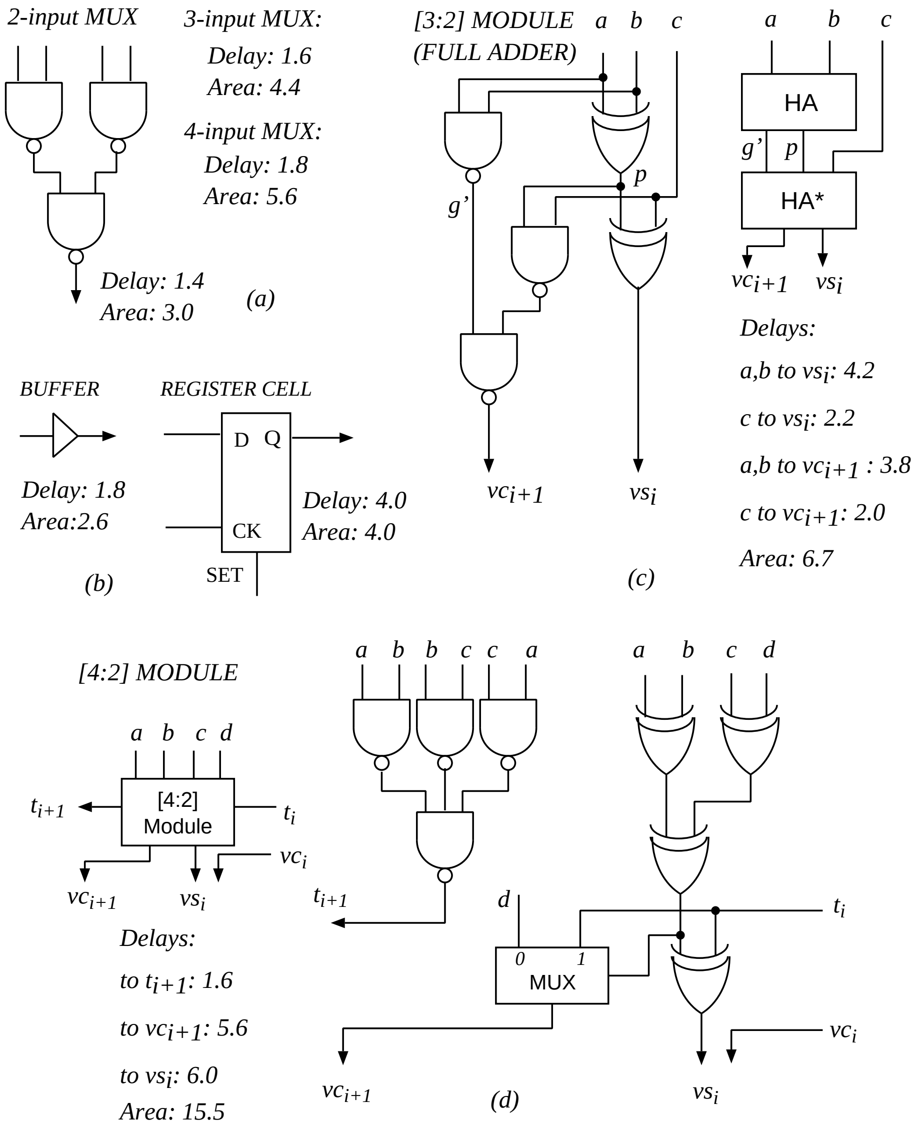


Figure 5.4: Basic modules: (a) Multiplexers. (b) Buffer and register cell. (c) Full-adder. (d) [4:2] module.

RADIX-2 DIVISION WITH CS RESIDUAL

- REDUNDANT RESIDUAL $w[j] = (WC[j],\ WS[j])$

```
1. [$Initialize$]
   $WS[0] \leftarrow x/2;\ WC[0] \leftarrow 0;\ q_0 \leftarrow 0;$Q[-1]=0
2. [$Recurrence$]
   for $j = 0 \ldots n+1$ ($n+2$ iterations because of initialization and guard bit)
   $q_{j+1} \leftarrow SEL(\widehat{y});$
   $(WC[j+1],\ WS[j+1]) \leftarrow CSADD(2WC[j],\ 2WS[j],\ -q_{j+1}d);$
   $Q[j] \leftarrow CONVERT(Q[j-1],\ q_j)$
   end for
3. [$Terminate$]
   If $w[n+2] < 0$ then $q = 2(CONVERT(Q[n+1],\ q_{n+2} - 1))$
   else $q = 2(CONVERT(Q[n+1],\ q_{n+2}))$
```

RADIX-2 ALGORITHM (cont.)

- n is the precision in bits,
- SEL is the quotient-digit selection function:

$$q_{j+1} = SEL(\widehat{y}) = \begin{cases} 1 & \text{if } 0 \leq \widehat{y} \leq 3/2 \\ 0 & \text{if } \widehat{y} = -1/2 \\ -1 & \text{if } -5/2 \leq \widehat{y} \leq -1 \end{cases}$$

  The estimate $\widehat{y}$ has four bits (three integer bits and one fractional bit) of the shifted residual in carry-save form,

- $CSADD$ is carry-save addition
- $-q_{j+1}d$ is in 2's complement form, and
- $CONVERT$ on-the-fly quotient conversion function

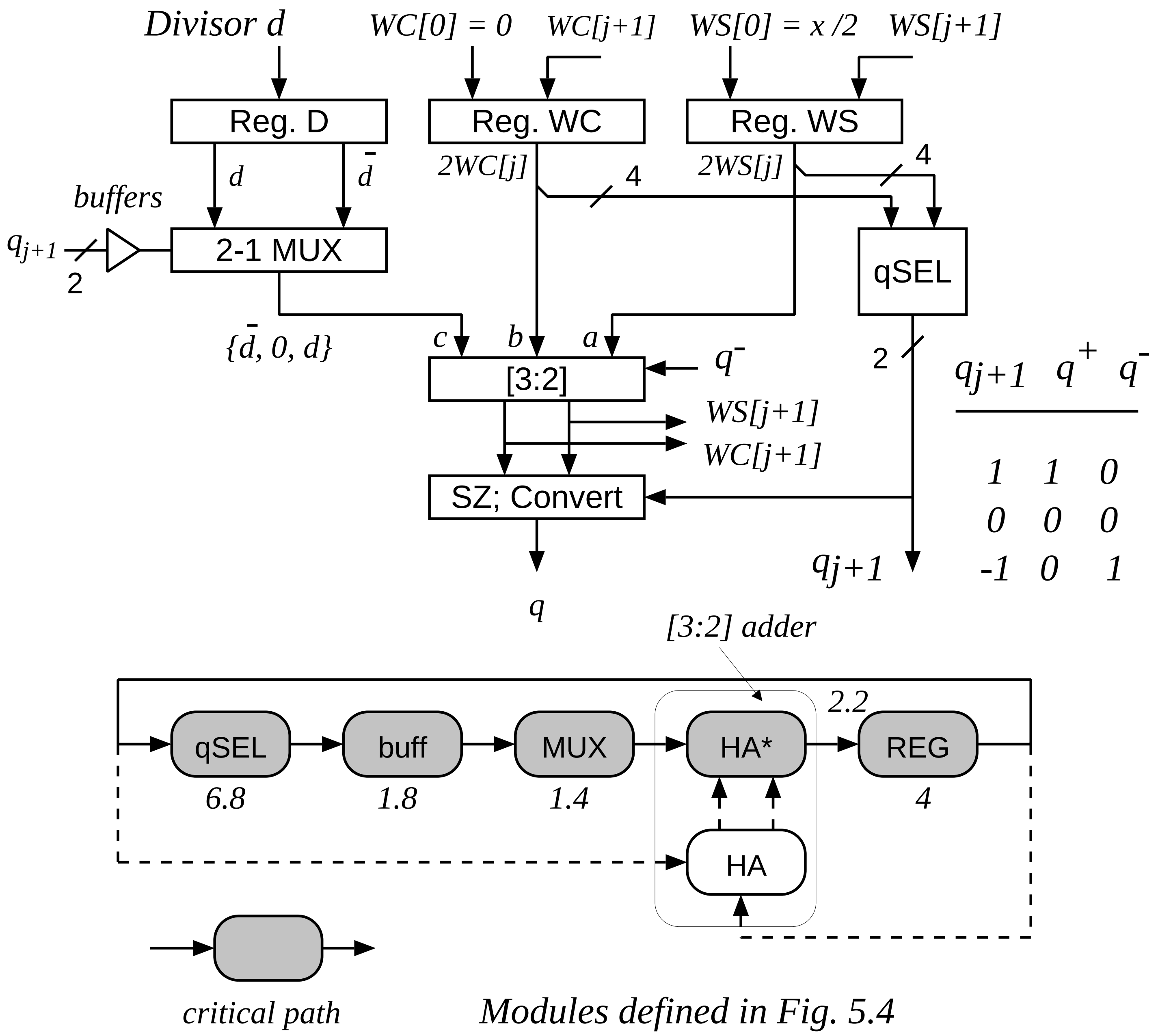
Dividend $x = (0.10011111)$, divisor $d = (0.11000101)$, scaled residual $2w[0] = 2(x/2) = x$, $q_{computed} = q/2$

$$
\begin{array}{rlllll}
2WS[0] = & 000.10011111 & & & & \\
2WC[0] = & 000.00000001 & * & \widehat{y}[0] = 0.5 & & q_1 = 1 \\
-q_1d = & 11.00111010 & & & & \\
\hline
2WS[1] = & 111.01001000 & & & & \\
2WC[1] = & 000.01101100 & & \widehat{y}[0] = -1 & & q_2 = -1 \\
-q_2d = & 00.11000101 & & & & \\
\hline
2WS[2] = & 111.11000010 & & & & \\
2WC[2] = & 001.00110001 & * & \widehat{y}[1] = 0.5 & & q_3 = 1 \\
-q_3d = & 11.00111010 & & & & \\
\hline
2WS[3] = & 011.10010010 & & & & \\
2WC[3] = & 100.11001001 & * & \widehat{y}[2] = 0 & & q_4 = 1 \\
-q_4d = & 11.00111010 & & & & \\
\hline
2WS[4] = & 000.11000010 & & & & \\
2WC[4] = & 110.01101000 & & \widehat{y}[4] = -1.5 & & q_5 = -1 \\
-q_5d = & 00.11000101 & & & & \\
\hline
\end{array}
$$

* for 2's complement of $q_{j+1}d$

$$q = 2(.1\bar{1}11\bar{1}) = .1101$$

Figure 5.7: Example of radix-2 division with residual in carry-save form.



q_{j+1}	q^+	q^-
1	1	0
0	0	0
-1	0	1

Figure 5.8: IMPLEMENTATION OF RADIX-2 SCHEME.

DELAY AND AREA OF RADIX-2 STAGE

element	delay	area
q-digit selection	6.8	50
buffers	1.8	5
MUX	1.4	160
CSA	2.2	360
registers (3)	4.0	650
Convert & Round	(NC)	1360
Cycle time	16.2	
Total area		2585

NC denotes a delay not in the critical path

RADIX-4 DIVISION WITH CS RESIDUAL

1. QUOTIENT DIGIT SET {-2,-1,0,1,2}
2. $\rho < 1$ initialize $WS[0] \leftarrow x/4$
3. THE NEXT RESIDUAL
   $(WC[j+1],\ WS[j+1]) \leftarrow CSADD(4WC[j],\ 4WS[j],\ -q_{j+1}d)$
4. QUOTIENT-DIGIT SELECTION DEPENDS ON ESTIMATES OF SHIFTED RESIDUAL AND DIVISOR described in terms of SELECTION CONSTANTS $m_k(i)$

$$q_{j+1} = k \text{ if } m_k(i) \leq \hat{y} < m_{k+1}(i)$$

5. FINAL QUOTIENT = 4 x OBTAINED QUOTIENT

SELECTION CONSTANTS

- $q_{j+1} = k$ if $m_k(i) \leq \hat{y} < m_{k+1}(i)$
- $i = 16\hat{d}$ and $\hat{d}$ divisor truncated to the 4th fractional bit and
- $\hat{y}$ is $4w[j]$ truncated to the 4th fractional bit.

i	8	9	10	11	12	13	14	15
$m_2(i)^+$	**12**	**14**	**15**	**16**	**18**	**20**	**20**	**24**
$m_1(i)^+$	**4**	**4**	**4**	**4**	**6**	**6**	**8**	**8**
$m_0(i)^+$	**-4**	**-6**	**-6**	**-6**	**-8**	**-8**	**-8**	**-8**
$m_{-1}(i)^+$	**-13**	**-15**	**-16**	**-18**	**-20**	**-20**	**-22**	**-24**

$+$: real value = shown value/16

Dividend $x = (0.10101111)$, divisor $d = (0.11000101)$ $(i = 16(0.1100)_2 = 12)$
scaled residual $4w[0] = 4(x/4) = x$, $q_{computed} = q/4$

$$
\begin{array}{rlll}
4WS[0]^{+} = & 000.10101111 & & \\
4WC[0]^{+} = & 000.00000001 \;\; * & \widehat{y}[0] = 10/16 & q_1 = 1 \\
-q_1d^{+} = & \;\;11.00111010 & & \\
\hline
WS[1] = & \;\;\;\;1.10010100 & & \\
WC[1] = & \;\;\;\;0.01010110 & & \\
\hline
4WS[1]^{+} = & 110.01010000 & & \\
4WC[1]^{+} = & 001.01011000 & \widehat{y}[1] = -6/16 & q_2 = 0 \\
-q_2d^{+} = & \;\;00.00000000 & & \\
\hline
WS[2] = & \;\;\;\;1.00001000 & & \\
WC[2] = & \;\;\;\;0.10100000 & & \\
\hline
4WS[2]^{+} = & 100.00100000 & & \\
4WC[2]^{+} = & 010.10000000 & \widehat{y}[2] = -22/16 & q_3 = -2 \\
-q_3d^{+} = & \;\;01.10001010 & & \\
\hline
w[3] = & \;\;\;\;0.00101010 & &
\end{array}
$$

* least-significant 1 for 2's complement of $q_{j+1}d$
$+$ only one integer bit used in the recurrence, because of the range of $w[j+1]$.

$$q[3] = .10\bar{2}_4 = .032_4$$

Figure 5.9: Example of radix-4 division with residual in carry-save form.(On-the-fly conversion and termination not shown)

DELAY AND AREA OF RADIX-4 STAGE

element	delay	area
q-digit selection	10.8	160
buffers	1.8	10
MUX	1.8	300
CSA	2.2	360
registers (3)	4.0	650
Convert & Round	(NC)	1360
Cycle time	20.6	
Total area		2840

NC denotes a delay not in the critical path

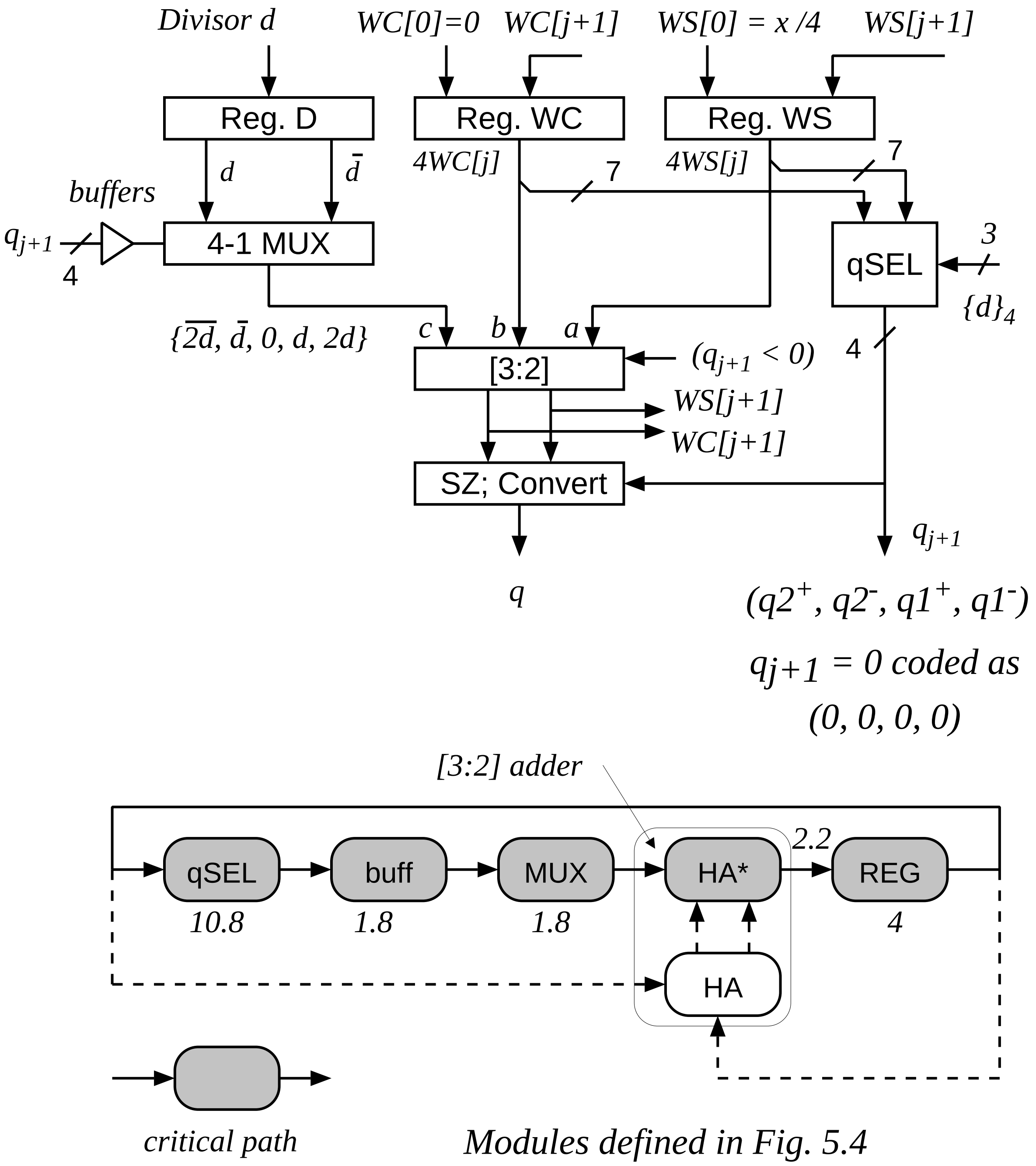


Figure 5.10: IMPLEMENTATION OF RADIX-4 SCHEME.

RADIX-8 DIVISION WITH CS RESIDUAL

- QUOTIENT DIGIT SET {-7, ..., 7} DECOMPOSED INTO

$$q_H = \{-8, -4, 0, 4, 8\}$$

AND

$$q_L = \{-2, -1, 0, 1, 2\}$$

element	delay	area
q-digit selection	(q_h) 12.2	610
buffers	1.8	20
MUXes	1.8	600
CSAh	2.2	360
CSAl	4.2	360
registers (3)	4.0	650
Convert & Round	(NC)	1360
Cycle time	26.2	
Total area		3960

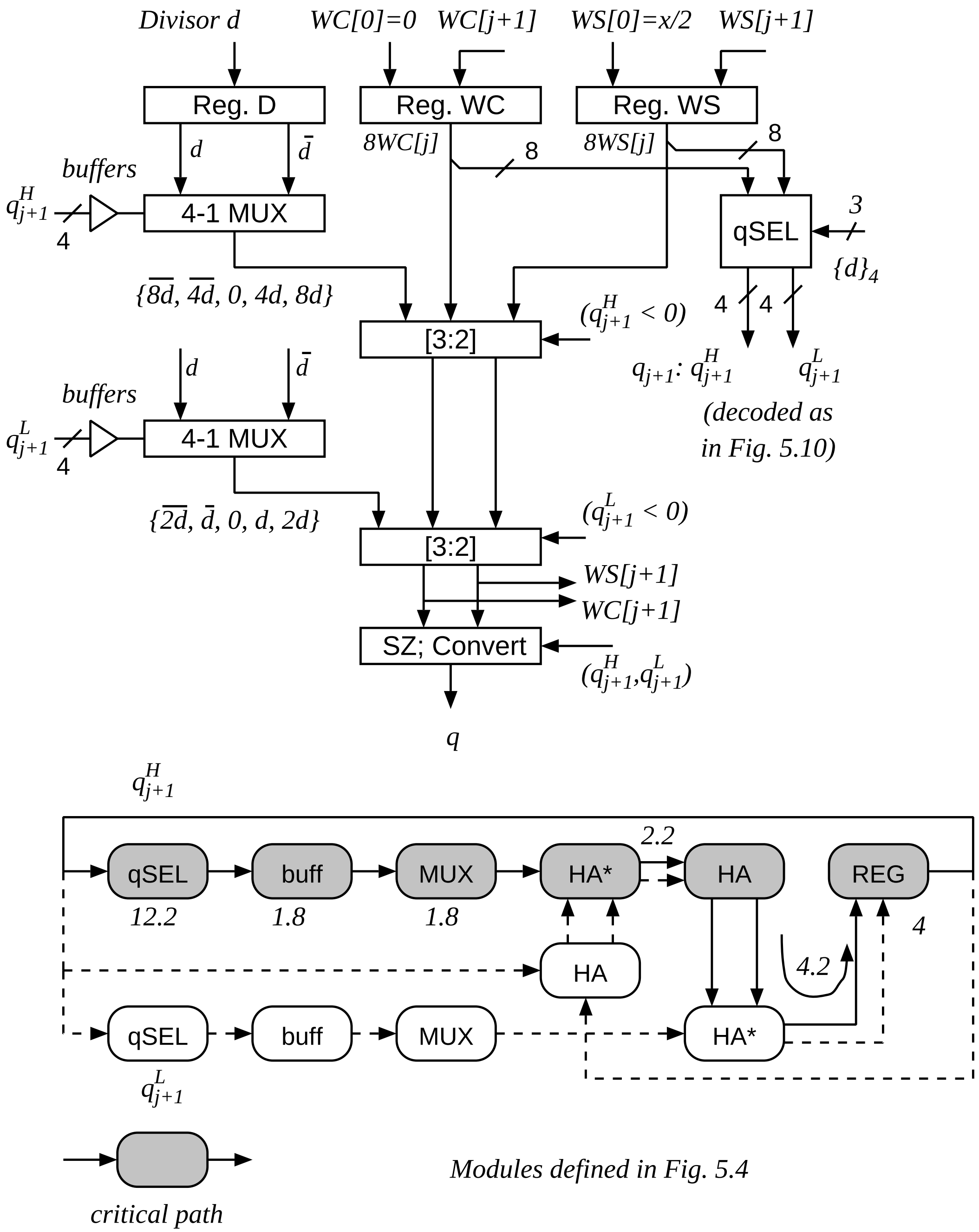


Figure 5.11: Implementation of radix-8 scheme.

RADIX-16 DIVISION WITH TWO RADIX-4 OVERLAPPED STAGES

element	delay	area
CSA	4.2	220
q-digit selection	11.2	820
MUX	1.4	
buffers	1.8	20
MUXes	1.8	600
CSA1	(NC)	360
CSA2	2.1	360
registers (3)	4.0	650
Convert & Round	(NC)	1360
Cycle time	26.6	
Total area		4390

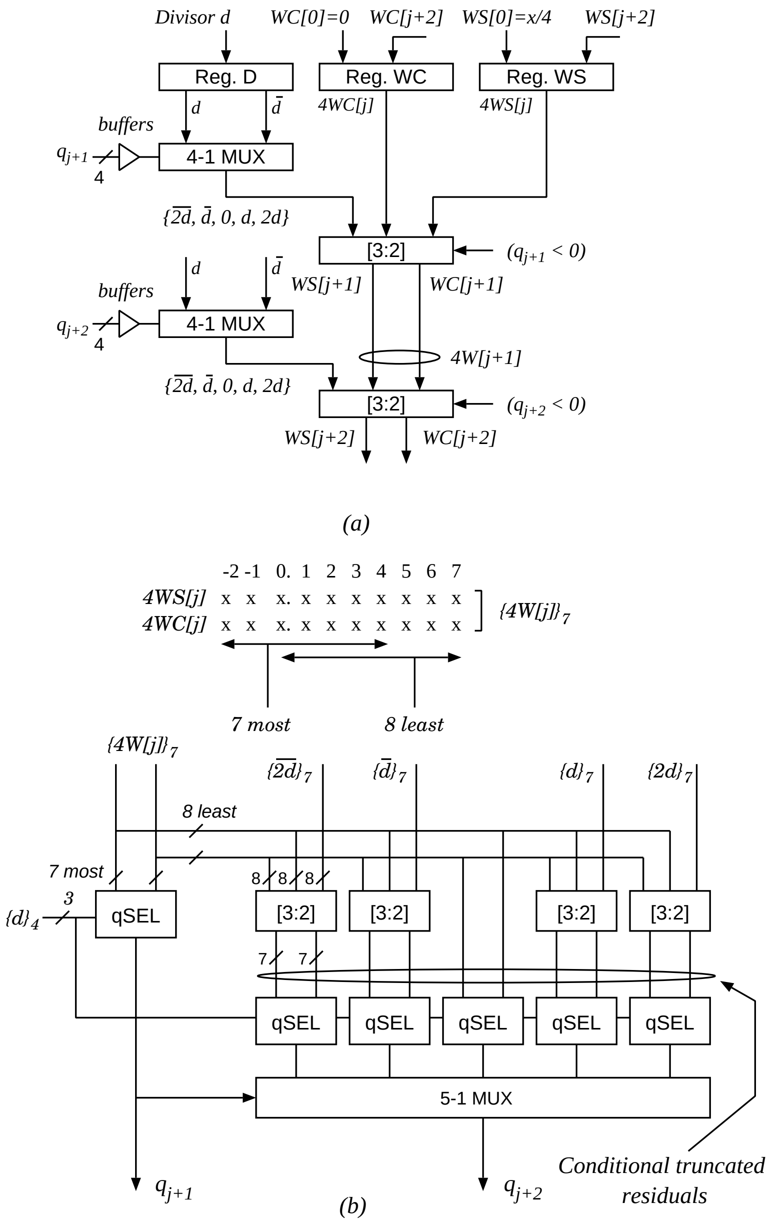


Figure 5.12: Implementation of radix-16 with radix-4 stages.

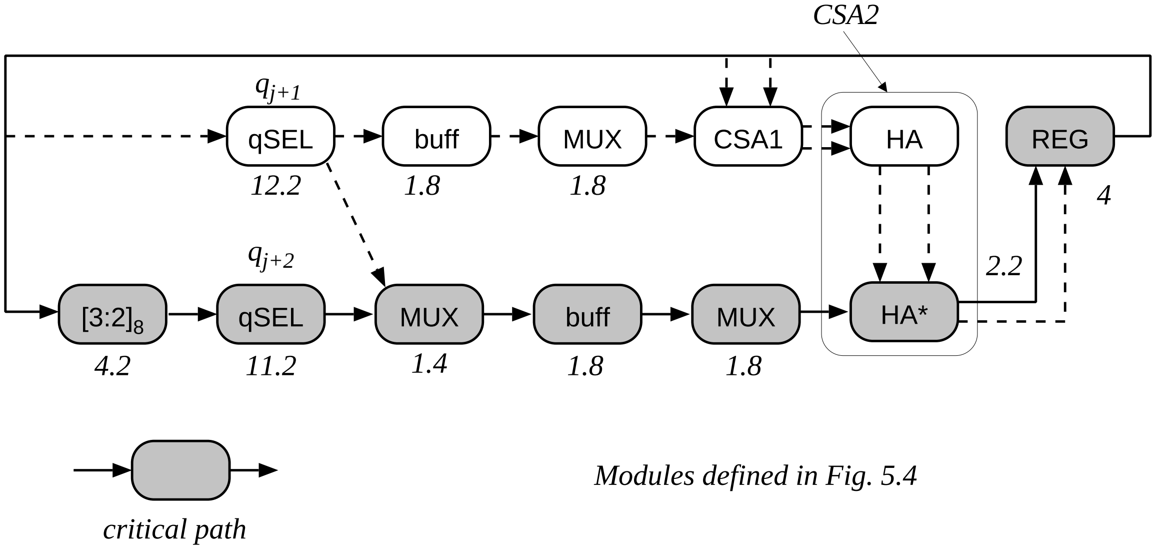


Figure 5.13: Critical path in radix-16 scheme.

RADIX-512 DIVISION WITH PRESCALING AND SELECTION BY ROUNDING

Cycle 1 : COMPUTE $M \approx 1/d$; compare d and x and set g

Cycle 2 : COMPUTE $z = Md$ (in c-s form); $v = 2^{-g}x$

Cycle 3 : INITIALIZE $w[0] = Mv$ (in c-s form); ASSIMILATE z;

Cycles 4 to 9 : ITERATE

$$q_{j+1} = round(\hat{y});\ w[j+1] = 512w[j] - q_{j+1}z$$

Cycle 10 : CORRECTING AND ROUNDING.

RADIX-512 DIVISON (cont.)

- RECTANGULAR MULTIPLIER-ACCUMULATOR
- DELAY-AREA

element	delay	area
M-module	(NC)	1800
MUX	1.4	
recoder	6.0	70
buffer	1.8	
MUX	1.8	3000
2 levels of 4-2 CSA	12.0	3100
registers(3)	4.0	650
Convert & Round	(NC)	1360
Cycle time	27	
Total area		9980

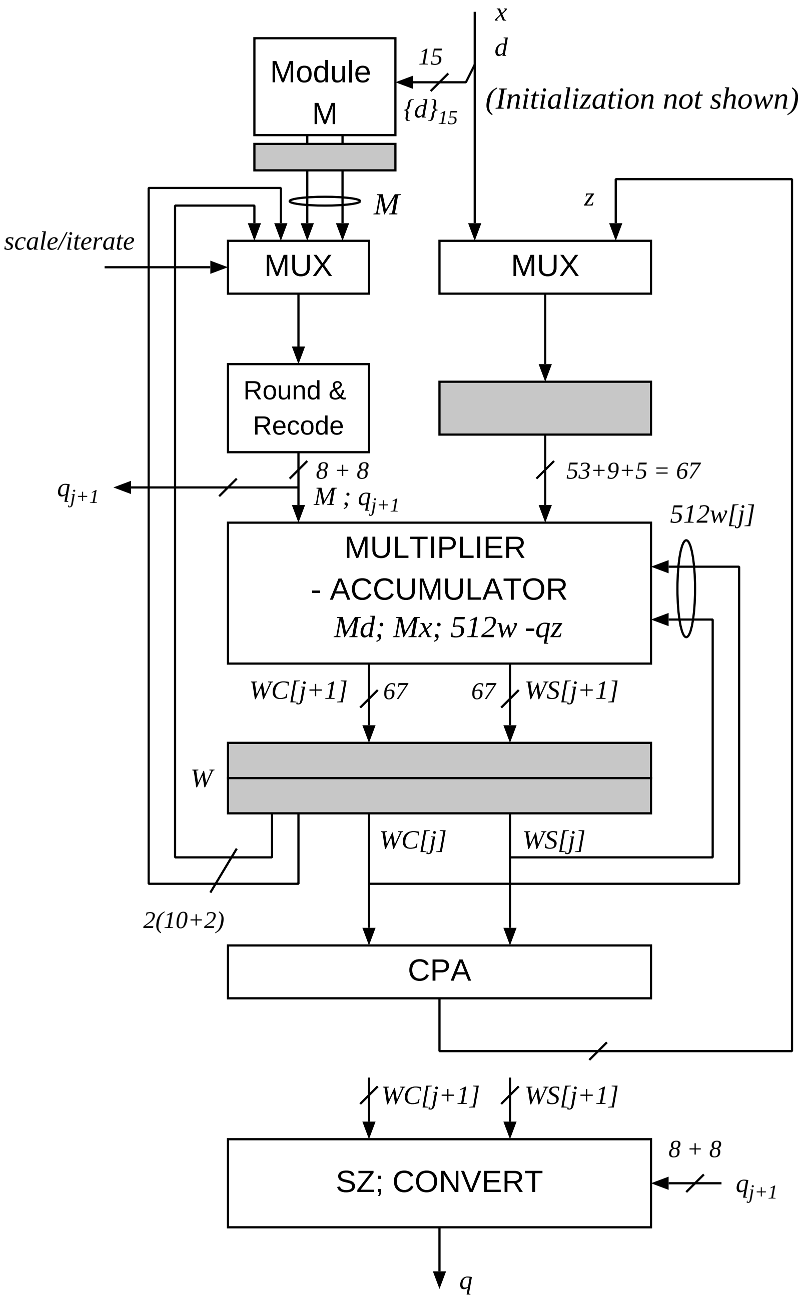


Figure 5.14: Implementation of radix-512 scheme.

OVERALL COMPARISONS

Scheme	r2	r4	r8	r16 (overlapped)	r512
Cycle-time factor	1.0	1.3	1.6	1.6	1.7
Number of cycles†	57	29	20	15	10
Speedup	1.0	1.5	1.8	2.4	3.4
Area factor	1.0	1.1	1.5	1.7	3.9

†Correction: two cycles for radix-2, one cycle for other cases.

QUOTIENT-DIGIT SELECTION FUNCTION

Quotient-digit set:

$$q_{j+1} \in \mathcal{D}_a = \{-a, -a+1, \ldots, -1, 0, 1, \ldots, a-1, a\}$$

Redundancy factor:

$$\rho = \frac{a}{r-1}, \quad \frac{1}{2} < \rho \leq 1$$

- TWO FUNDAMENTAL CONDITIONS FOR q-SELECTION
- CONTAINMENT – must guarantee bounded residual
- CONTINUITY – there must exist a valid choice of q_{j+1} in the range of shifted residual

CONTAINMENT AND SELECTION INTERVALS

- RESIDUAL RECURRENCE

$$w[j+1] = rw[j] - dq_{j+1} \quad |w[j]| \leq \rho d$$

$$\rho = a/(r-1) \quad -a \leq q_j \leq a$$

- SELECTION INTERVALS
- If $rw[j] \in [L_k, U_k]$ then $q_{j+1} = k$ makes $w[j+1]$ bounded

$$L_k \leq rw[j] \leq U_k \;\Rightarrow\; \rho d \leq w[j+1] = rw[j] - k \cdot d \leq \rho d$$

- EXPRESSIONS FOR SELECTION INTERVALS

$$U_k = (k+\rho)d \qquad L_k = (k-\rho)d$$

ROBERTSON'S AND P-D DIAGRAMS

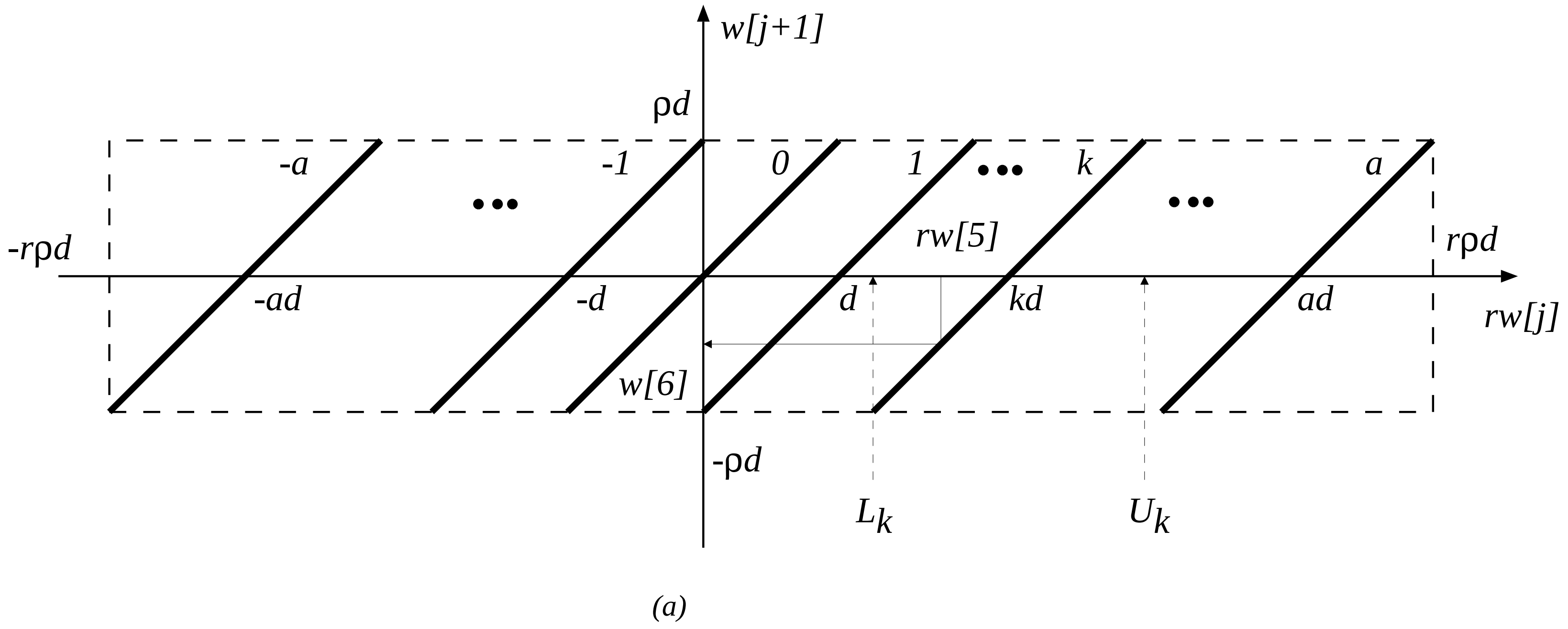


Figure 5.16: ROBERTSON'S DIAGRAM

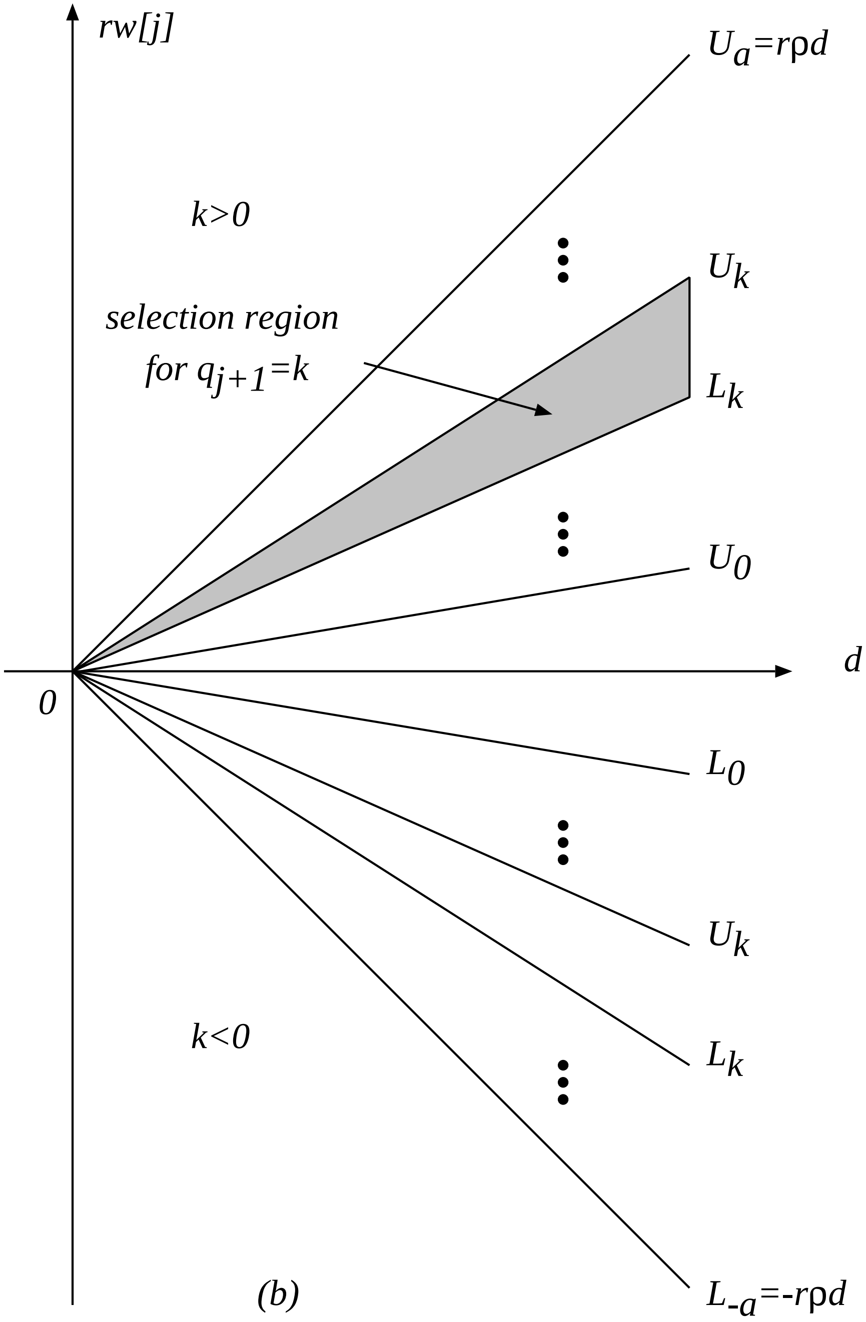


Figure 5.16: P-D DIAGRAM (for $d > 0$).

CONTINUITY CONDITION, OVERLAP, AND Q-DIGIT SELECTION

$$q_{j+1} = SEL(w[j], d)$$

- SEL represented by the set $\{s_k\}, -a \leq k \leq a$,

$$q_{j+1} = k \quad \text{if } s_k \leq rw[j] \leq s_{k+1} - ulp$$

- s_k defined as the **minimum** value of $rw[j]$ for which $q_{j+1} = k$
- s_k's are functions of the divisor d
- CONTAINMENT: $L_k \leq s_k \leq U_k$
- CONTINUITY: $q_{j+1} = k - 1$ for $rw[j] = s_k - ulp \leq U_{k-1}$

$$U_k \geq U_{k-1} + ulp \quad \rightarrow L_k \leq s_k \leq U_{k-1} + ulp \; or \; L_k \leq s_k \leq U_{k-1}$$

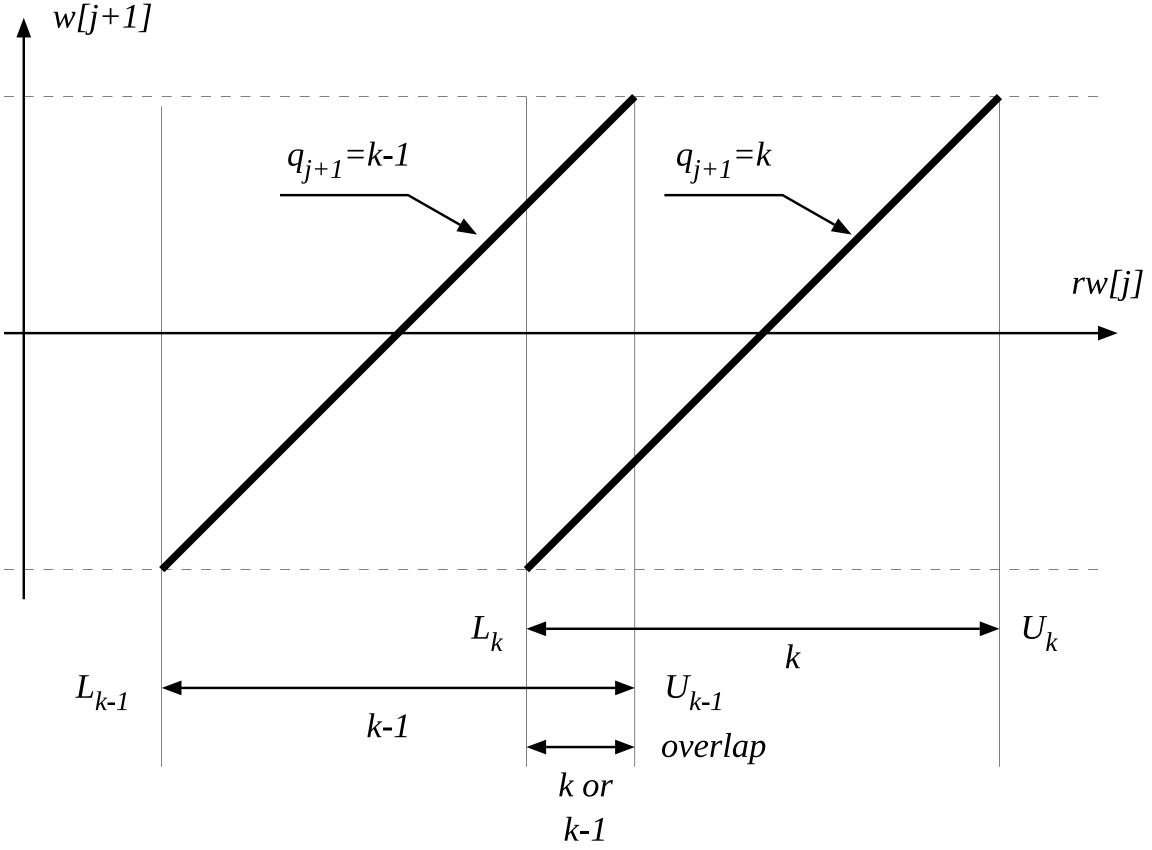
- OVERLAP

$$U_{k-1} - L_k = (k - 1 + \rho)d - (k - \rho)d = (2\rho - 1)d$$

RESULTING IN

$$\rho \geq 2^{-1}$$

- REDUNDANCY IN q-DIGIT SET → OVERLAP BETWEEN SELECTION INTERVALS - SIMPLER SELECTION



(a)

Figure 5.17: Overlap between selection intervals.

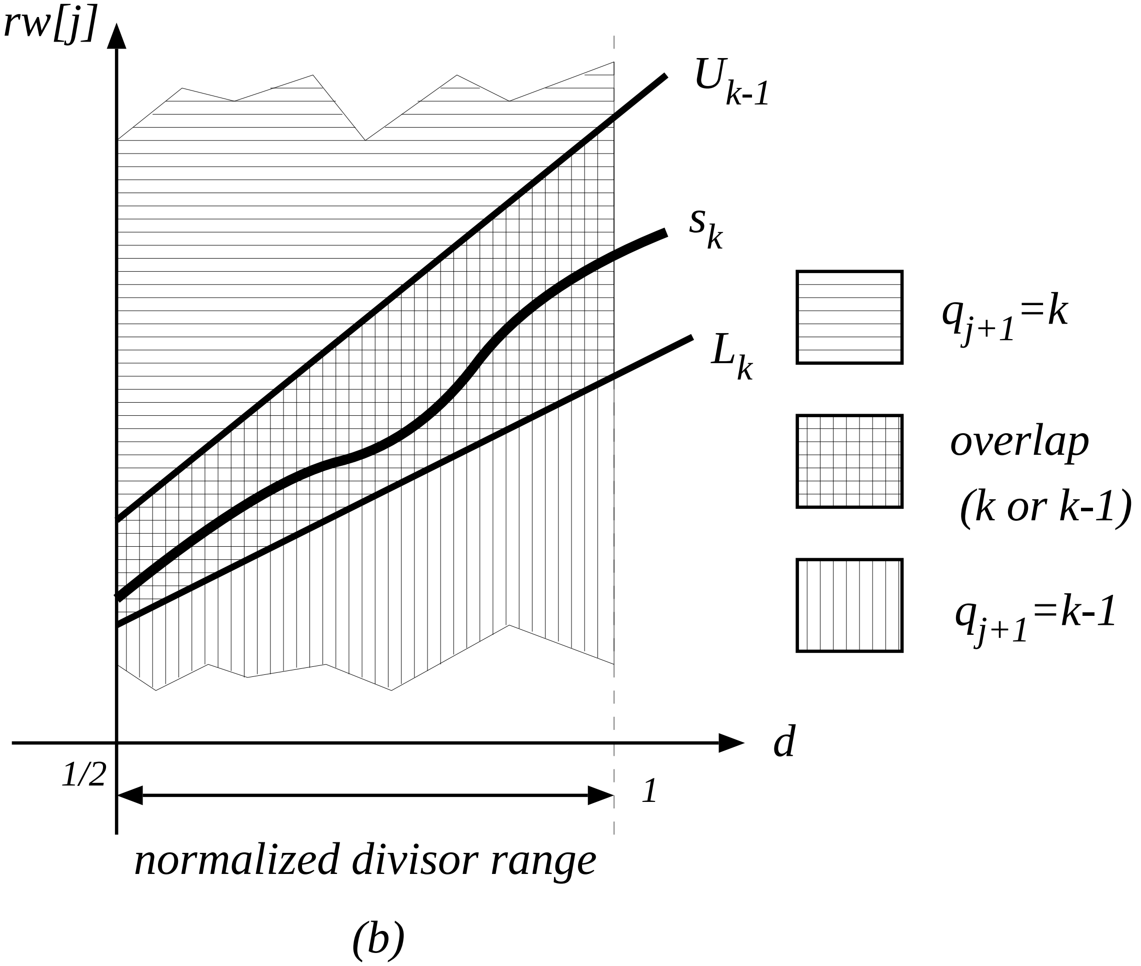


Figure 5.17: Selection function.

SELECTION USING CONSTANTS

- USE CONSTANTS m_k, INDEPENDENT OF DIVISOR

$$max(L_k) \leq m_k \leq min(U_{k-1}) + ulp$$

max and min for the range $2^{-1} \leq d < 1$

- For $k > 0$

$$(k - \rho) \leq m_k \leq (k - 1 + \rho)2^{-1} + ulp$$

which requires

$$\rho \geq \frac{k+1}{3}$$

- For $k \leq 0$

$$(k - \rho)2^{-1} \leq k - 1 + \rho$$

which requires

$$\rho \geq \frac{(-k)+2}{3}$$

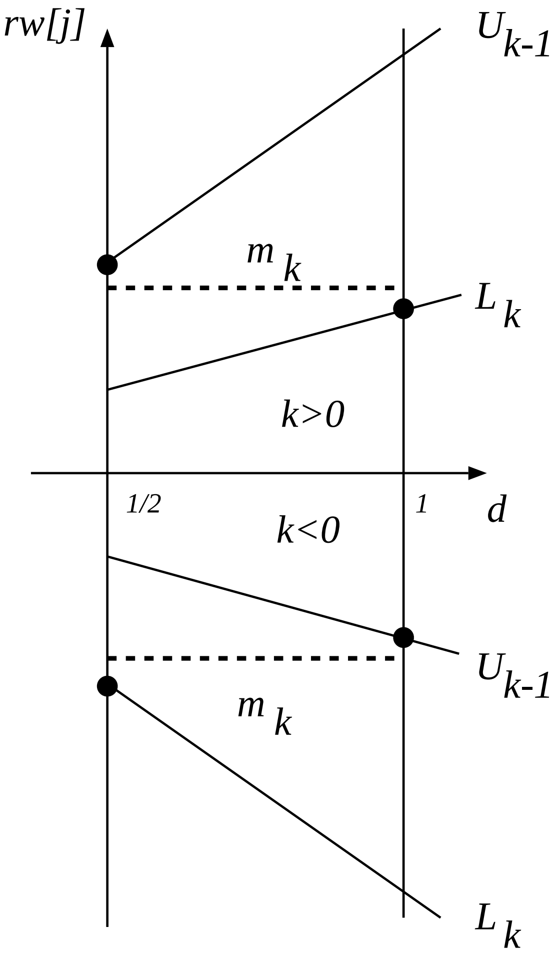


Figure 5.18: BOUNDS ON m_k.

RADIX-2 DIVISION WITH NON-REDUNDANT RESIDUAL

- EXTENSION OF NON-RESTORING DIVISION: {-1,0,1} SRT
- ALLOWS SKIPPING OVER ZEROS

$$\begin{array}{ll} U_1 = 2d & L_1 = 0 \\ U_0 = d \geq 1/2 & L_0 = -d \leq -1/2 \\ U_{-1} = 0 & L_{-1} = -2d \end{array}$$

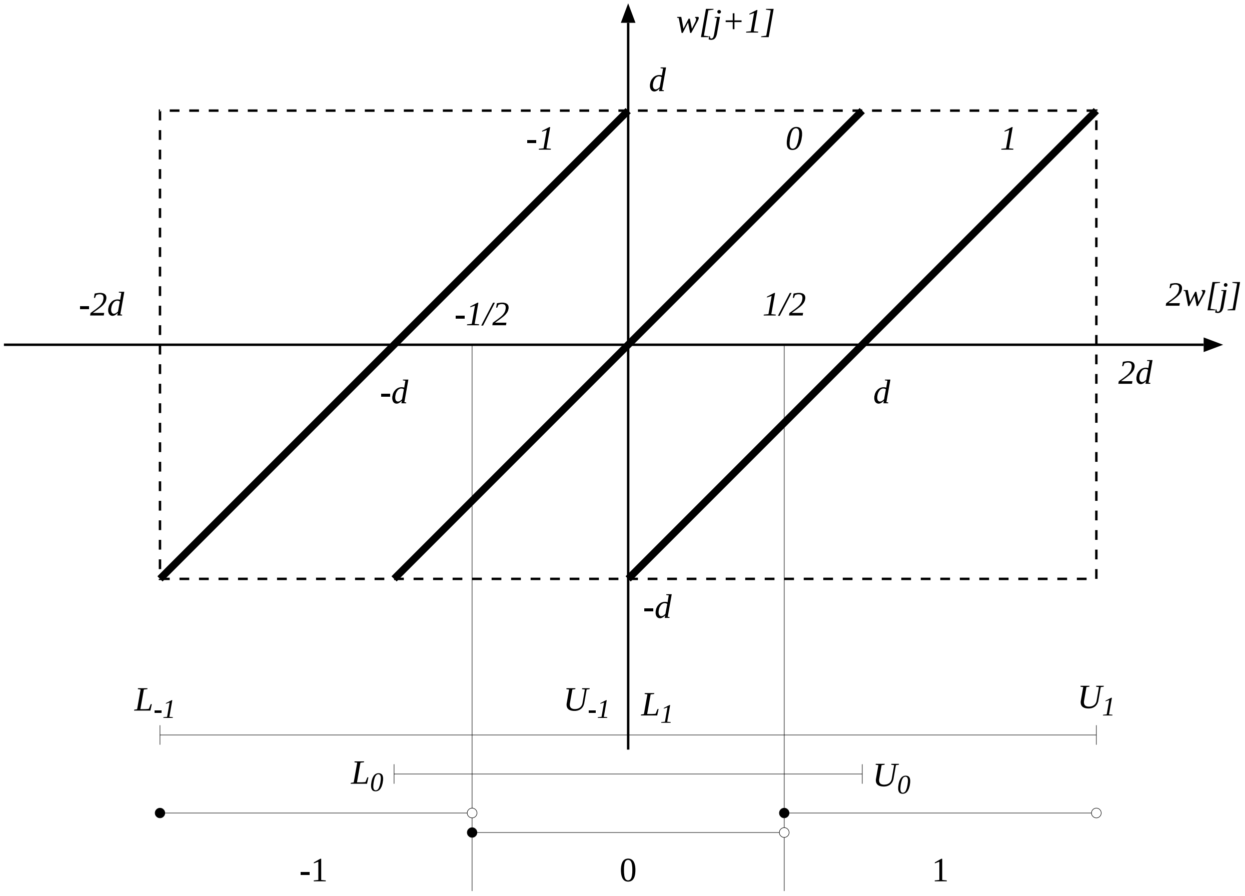
- SELECTION CONSTANTS:

$$0 \leq m_1 \leq 1/2, \qquad -1/2 \leq m_0 \leq 0$$

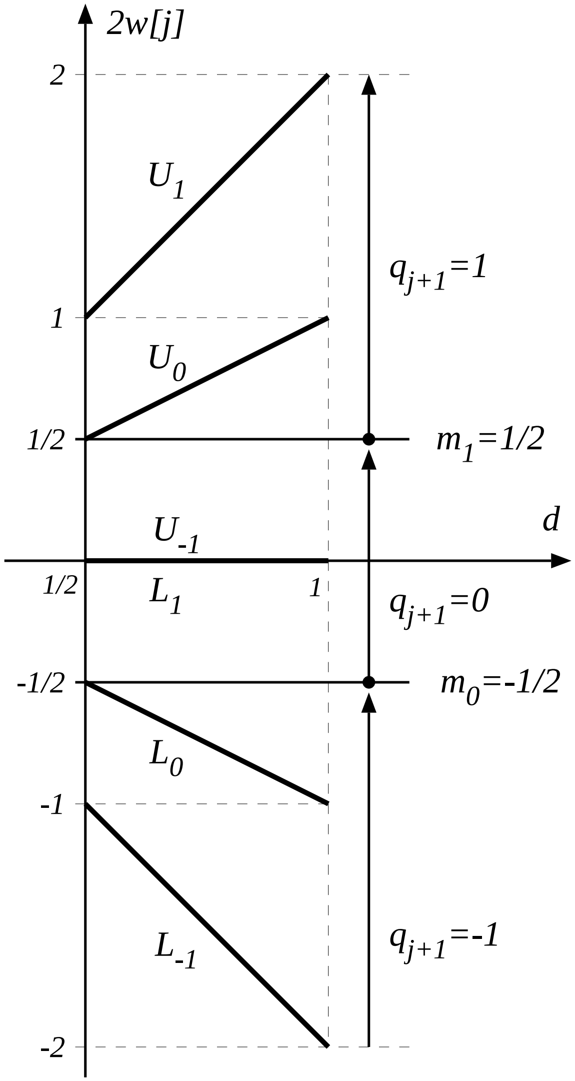
Choose: $m_1 = 1/2$ and $m_0 = -1/2$

- THE QUOTIENT-DIGIT SELECTION FUNCTION

$$q_{j+1} = \begin{cases} 1 \text{ if } 1/2 \leq 2w[j] \\ 0 \text{ if } -1/2 \leq 2w[j] < 1/2 \\ -1 \text{ if } 2w[j] \ < \ -1/2 \end{cases}$$



(a)


2w[j]
2
U_1
$q_{j+1}=1$
1
U_0
1/2
$m_1=1/2$
d
U_{-1}
1/2
L_1
1
$q_{j+1}=0$
-1/2
$m_0=-1/2$
L_0
-1
L_{-1}
$q_{j+1}=-1$
-2

STAIRCASE SELECTION FUNCTION

- FOR $r > 2$, m_k DEPENDS ON DIVISOR

- DIVIDE RANGE OF DIVISOR INTO INTERVALS $[d_i, d_{i+1})$ with

$$d_0 = \frac{1}{2}, \qquad d_{i+1} = d_i + 2^{-\delta}$$

δ MS FRACTIONAL BITS OF DIVISOR REPRESENT THE INTERVAL

- FOR EACH INTERVAL, THERE IS A SET of *selection constants* $m_k(i)$

$$\text{for } d \in [d_i, d_{i+1}), \quad q_{j+1} = k \;\text{ if }\; m_k(i) \leq rw[j] \leq m_{k+1}(i) - ulp$$

DEFINITION OF $m_k(i)$

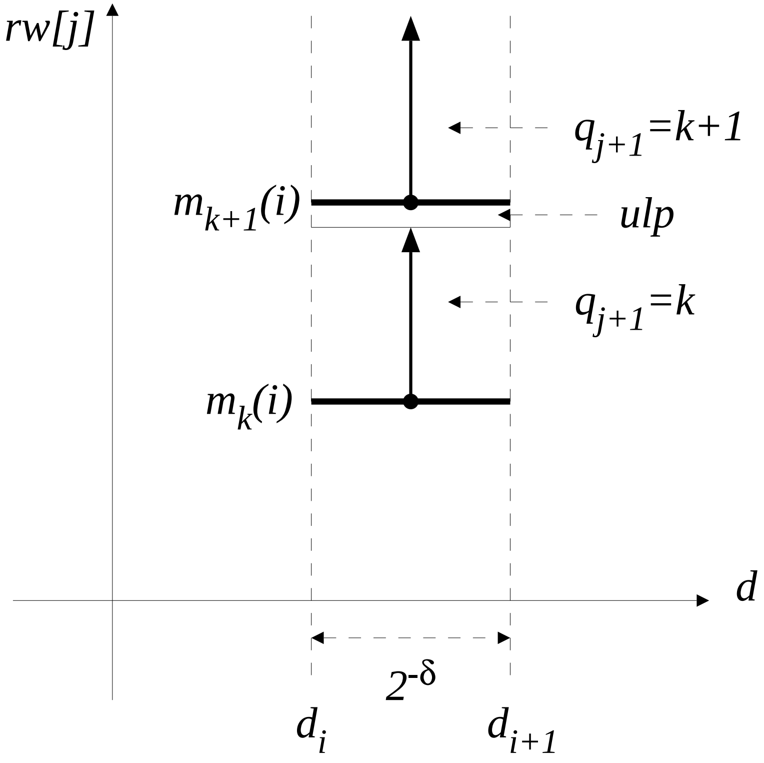


Figure 5.20: DEFINITION OF $m_k(i)$.

RANGE CONDITION FOR SELECTION CONSTANTS

$$max(L_k(d_i), L_k(d_{i+1})) \leq m_k(i) \leq min(U_{k-1}(d_i), U_{k-1}(d_{i+1})) + ulp$$

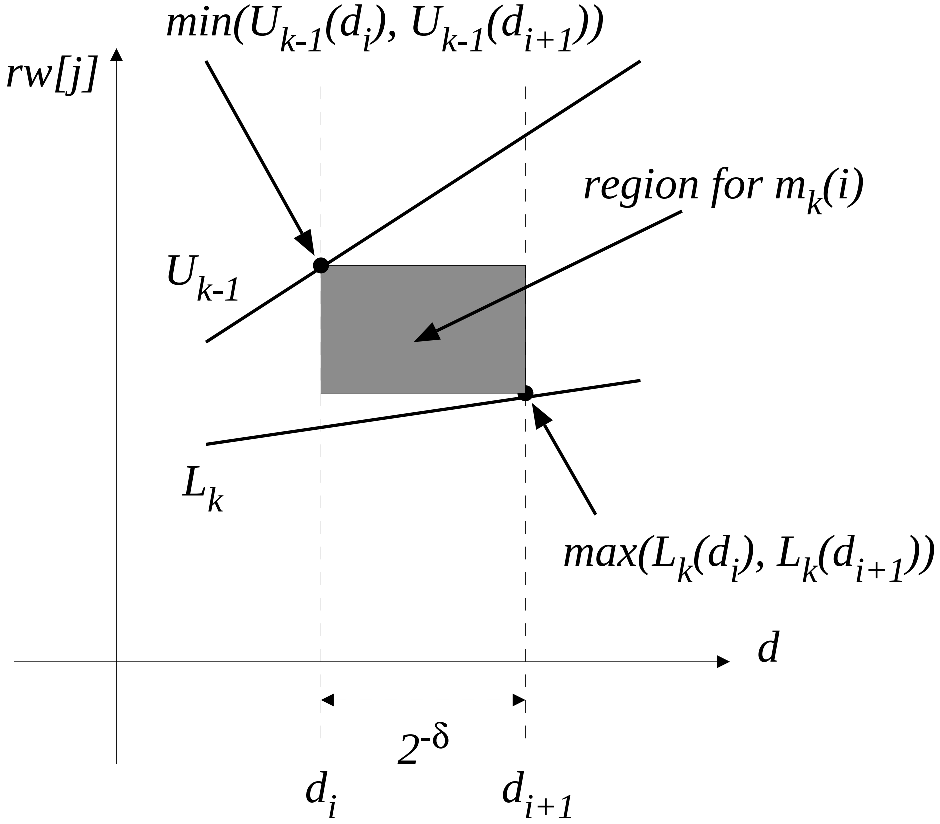

min(U_{k-1}(d_i), U_{k-1}(d_{i+1}))
rw[j]
region for m_k(i)
U_{k-1}
L_k
max(L_k(d_i), L_k(d_{i+1}))
d
2^{-δ}
d_i
d_{i+1}

Figure 5.21: SELECTION CONSTANT REGION.

LOW-PRECISION SELECTION CONSTANTS

$$m_k(i) = A_k(i)2^{-c}$$

WHERE $A_k(i)$ IS INTEGER

- CAN USE TRUNCATED RESIDUAL IN COMPARISONS WITH SELECTION CONSTANTS
- RESIDUAL MUST BE IN 2's COMPLEMENT
- SELECTION CONDITIONS

$$\begin{array}{ll} \text{for } k > 0 & L_k(d_i + 2^{-\delta}) \leq A_k(i)2^{-c} \leq U_{k-1}(d_i) \\ \text{for } k \leq 0 & L_k(d_i) \leq A_k(i)2^{-c} \leq U_{k-1}(d_i + 2^{-\delta}) + ulp \end{array} \qquad (5.1)$$

USE OF SELECTION CONSTANTS

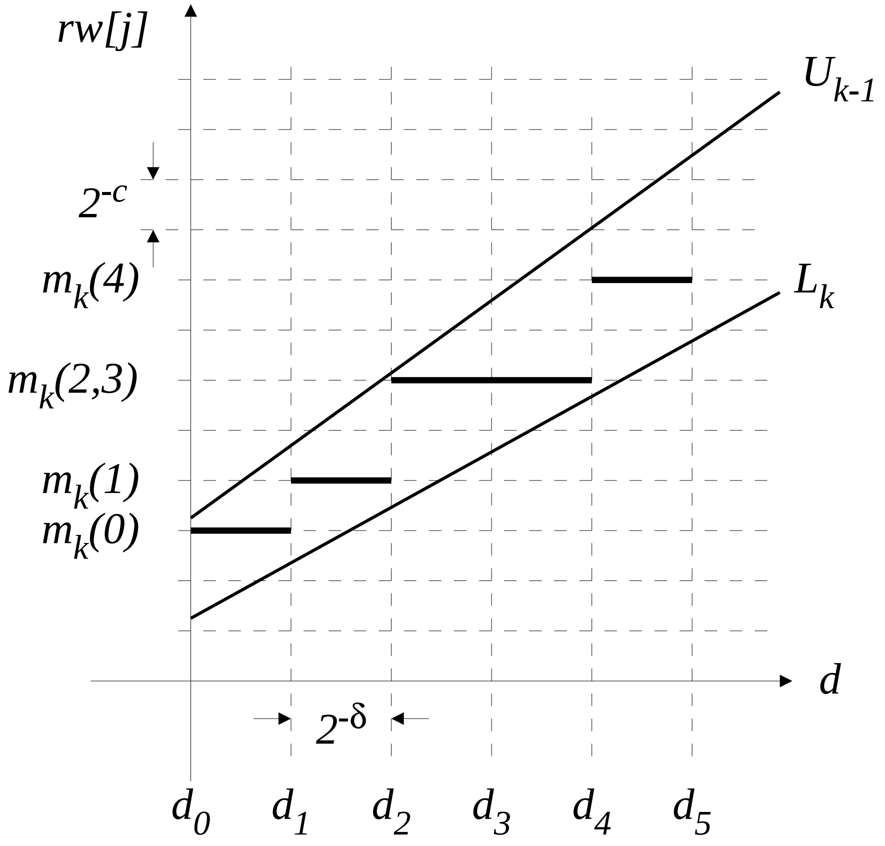


Figure 5.22: QUOTIENT-DIGIT SELECTION WITH SELECTION CONSTANTS.

SELECTION WITH TRUNCATED RESIDUAL AND DIVISOR

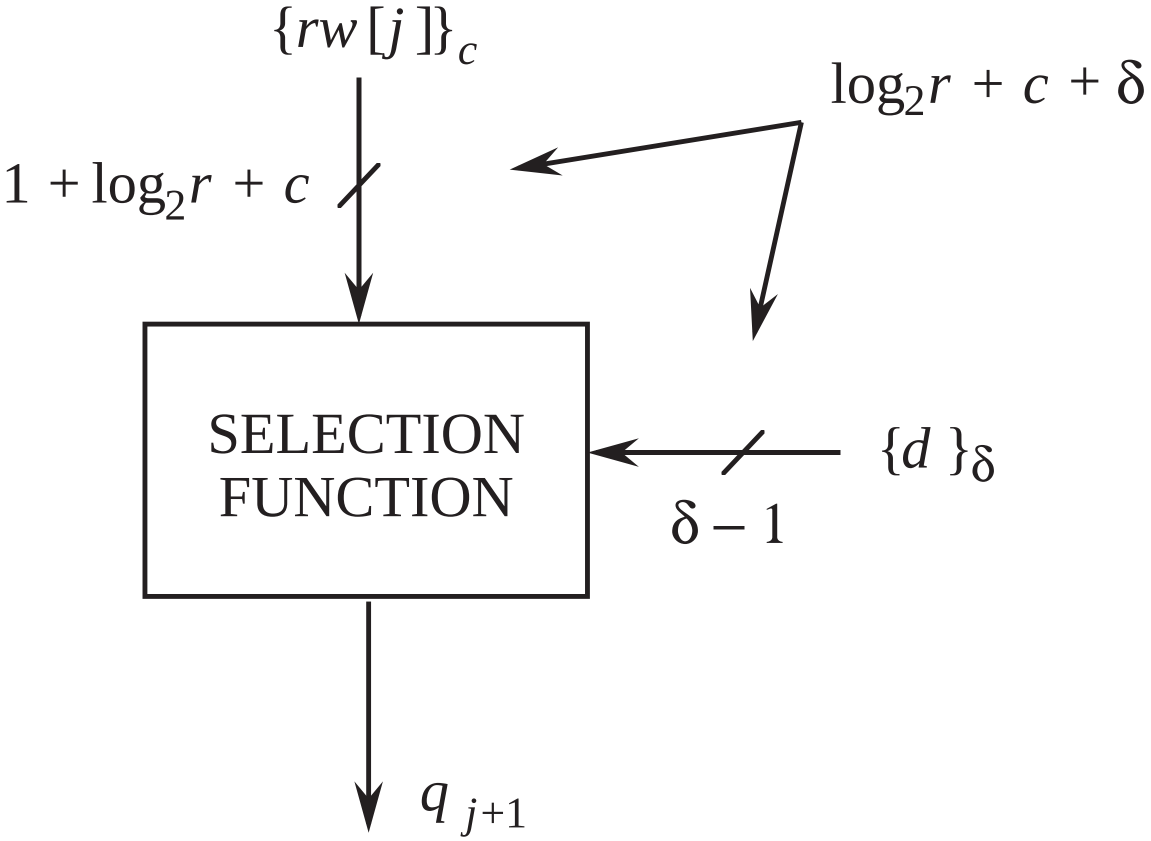


Figure 5.23: SELECTION WITH TRUNCATED RESIDUAL AND DIVISOR.

LOWER BOUND ON δ

- CONSIDER CASE $k > 0$ (similar argument for $k \leq 0$);
- FROM CONTINUITY CONDITION

$$U_{k-1}(d_i) - L_k(d_i + 2^{-\delta}) \geq 0$$

$$(\rho + k - 1)d_i - (-\rho + k)(d_i + 2^{-\delta}) \geq 0$$

$\Longrightarrow$

$$(2\rho - 1)d_i \geq (k - \rho)2^{-\delta}$$

- Worst case: $d \geq 1/2$ and $k \leq a$:

$$2^{-\delta} \leq \frac{2\rho - 1}{2(a - \rho)} = \frac{2\rho - 1}{2\rho(r - 2)}$$

- MINIMUM VALUE OF δ CAN RESULT IN A LARGE VALUE OF c
- OPTIMIZE THE VALUES OF δ AND c TOGETHER

RADIX-4 DIVISION WITH NONREDUNDANT RESIDUAL

- Known as Robertson's division
- $q_j \in \{-2, -1, 0, 1, 2\}$
- $U_k = (\frac{2}{3} + k)d \qquad L_k = (-\frac{2}{3} + k)d$
- BOUND ON δ

$$2^{-\delta} \leq \frac{2\rho - 1}{2(a - \rho)} = \frac{1}{8}$$

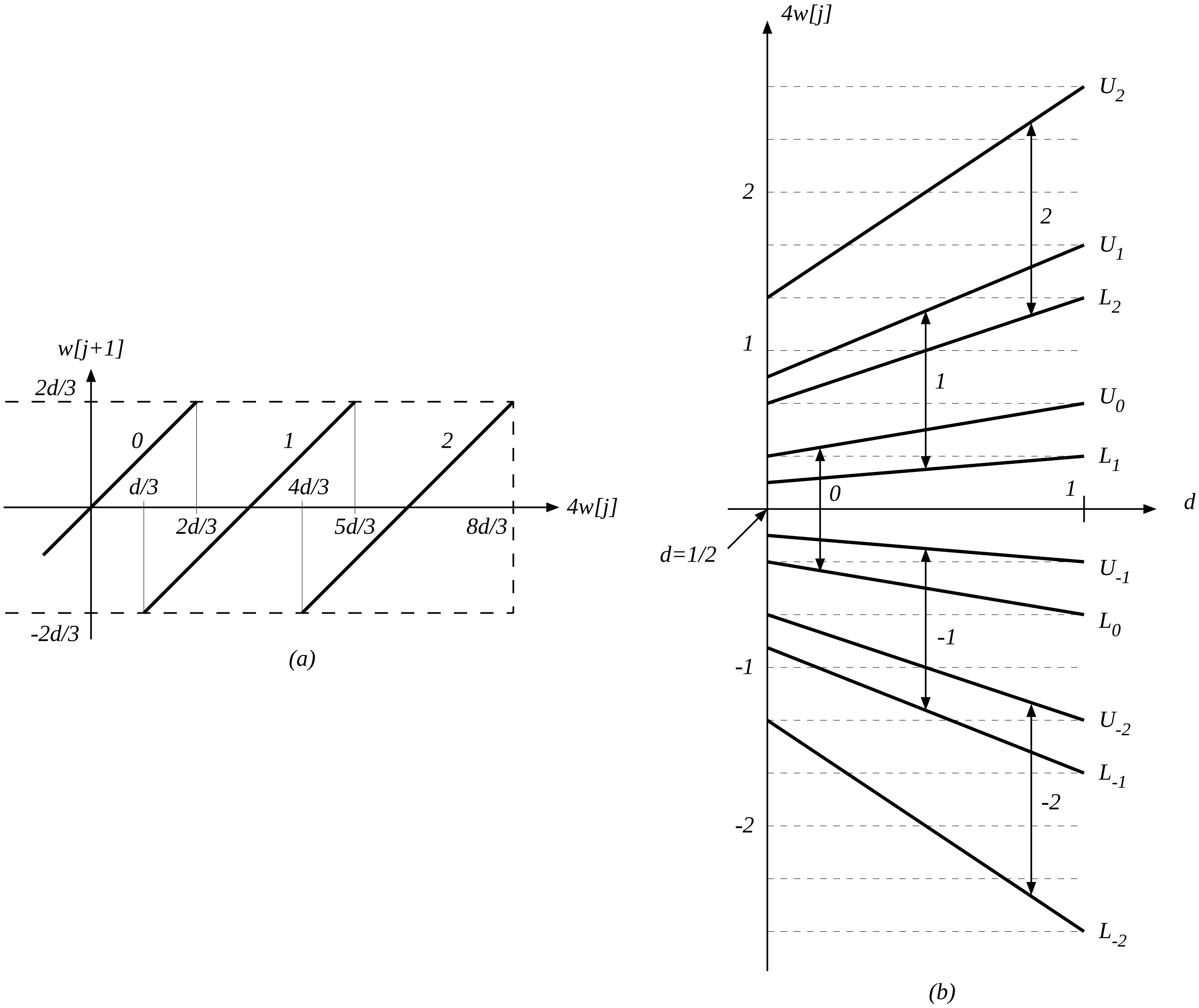


Figure 5.24: ROBERTSON'S AND PD DIAGRAMS FOR RADIX-4 AND $a = 2$.

$[d_i, d_{i+1})^+$	[8, 9)	[9, 10)	[10, 11)	[11, 12)
$L_2(d_{i+1}), U_1(d_i)^{\#}$	36, 40	40, 45	44, 50	48, 55
$m_2(i)^*$	**6**	**7**	**8**	**8**
$L_1(d_{i+1}), U_0(d_i)^{\#}$	9,16	10, 18	11, 20	12, 22
$m_1(i)$	**2**	**2**	**2**	**2**
$L_0(d_i), U_{-1}(d_{i+1})^{\#}$	-16, -9	-18, -10	-20, -11	-22, -12
$m_0(i)$	**-2**	**-2**	**-2**	**-2**
$L_{-1}(d_i), U_{-2}(d_{i+1})^{\#}$	-40, -36	-45, -40	-50, -44	-55, -48
$m_{-1}(i)$	**-6**	**-7**	**-8**	**-8**

$[d_i, d_{i+1})^+$	[12, 13)	[13, 14)	[14, 15)	[15, 16)
$L_2(d_{i+1}), U_1(d_i)^{\#}$	52, 60	56, 65	60, 70	64, 75
$m_2(i)$	**10**	**10**	**10**	**12**
$L_1(d_{i+1}), U_0(d_i)^{\#}$	13, 24	14, 26	15, 28	16, 30
$m_1(i)$	**4**	**4**	**4**	**4**
$L_0(d_i), U_{-1}(d_{i+1})^{\#}$	-24, -13	-26, -14	-28, -15	-30, -16
$m_0(i)$	**-4**	**-4**	**-4**	**-4**
$L_{-1}(d_i), U_{-2}(d_{i+1})^{\#}$	-60, -52	-65, -56	-70, -60	-75, -64
$m_{-1}(i)$	**-10**	**-10**	**-10**	**-12**

QUOTIENT-DIGIT SELECTION FOR RADIX-4 DIVISION; NON-REDUNDANT RESIDUAL

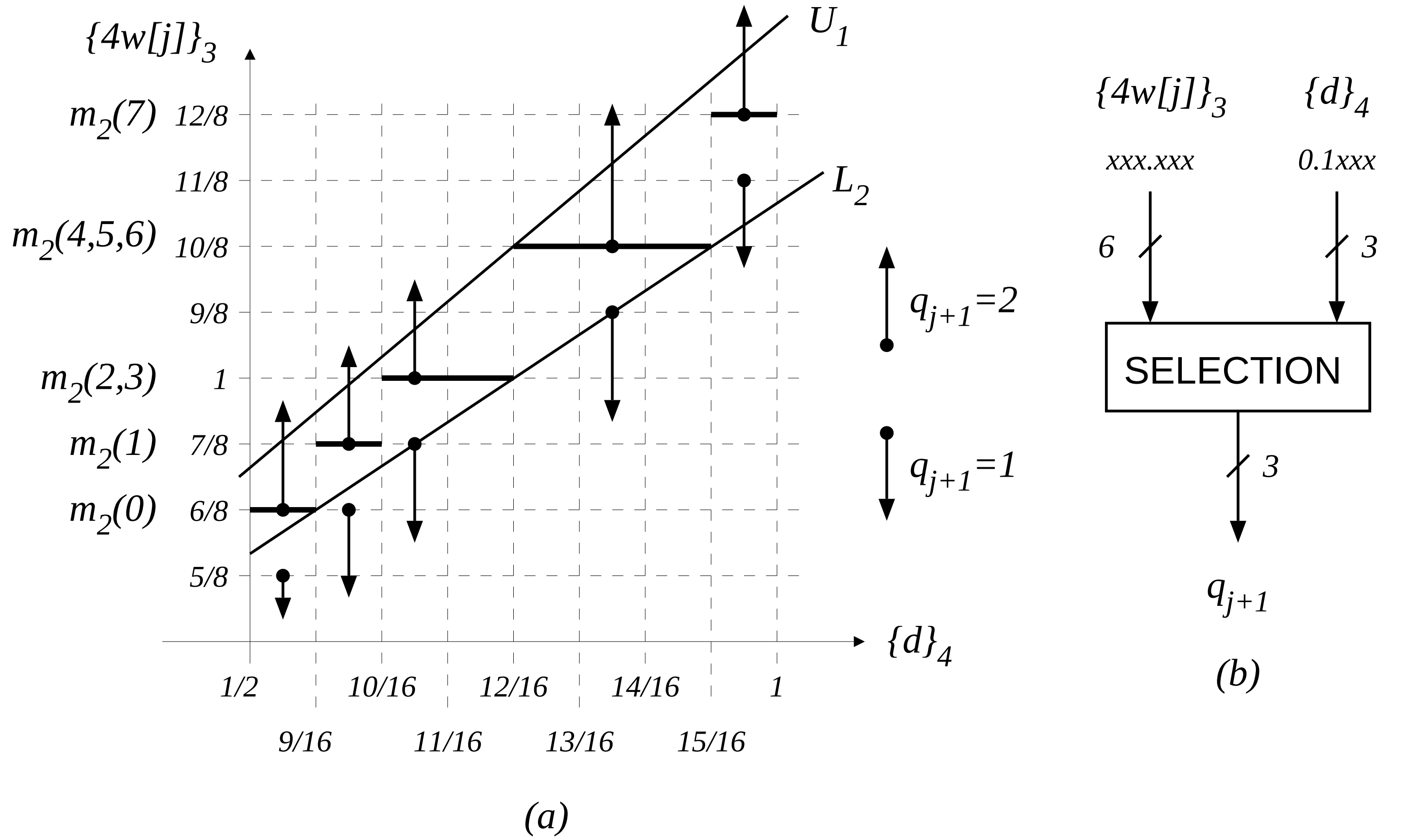


Figure 5.25: QUOTIENT-DIGIT SELECTION: (a) FRAGMENT OF THE P-D DIAGRAM. (b) IMPLEMENTATION.

USE OF REDUNDANT ADDER

- SO FAR: COMPUTE $rw[j]$ IN FULL PRECISION, TRUNCATE, AND COMPARE WITH LOW-PRECISION CONSTANTS
- FULL-PRECISION ADDITION: significant portion of the cycle time
- OVERLAP BETWEEN SELECTION INTERVALS $\Longrightarrow$COULD USE AN ESTIMATE OF $rw[j]$
- ERROR IN ESTIMATE:

$$\epsilon_{min} \leq y - \widehat{y} \leq \epsilon_{max}$$

- BASIC CONSTRAINT: if we choose $q_{j+1} = k$ for an estimate $\widehat{y}$ then

$$y \in [\widehat{y} + \epsilon_{min},\ \widehat{y} + \epsilon_{max}]$$

$$L_k^* = L_k - \epsilon_{min}$$

$$U_k^* = U_k - \epsilon_{max}$$

CONSTRAINTS ON SELECTION CONSTANTS

$$max(L_k^*(d_i), L_k^*(d_{i+1})) \leq m_k(i) \leq min(U_{k-1}^*(d_i), U_{k-1}^*(d_{i+1}))$$

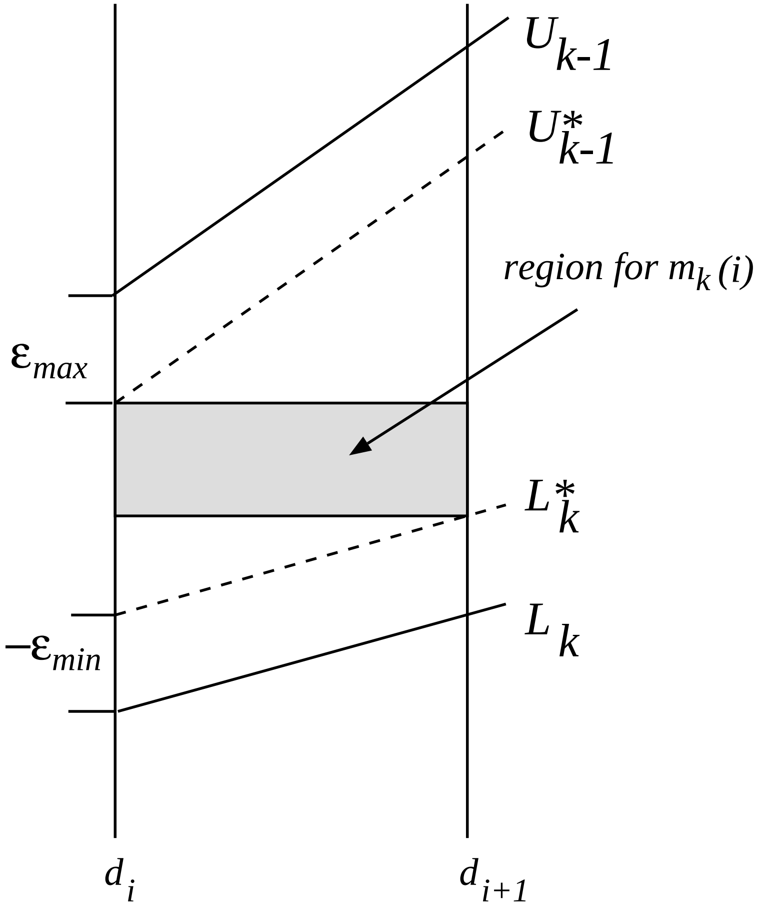


Figure 5.26: CONSTRAINTS FOR SELECTION BASED ON ESTIMATES.

MINIMUM OVERLAP AND RANGE OF ESTIMATE

- OVERLAP

$$min(U^*_{k-1}(d_i), U^*_{k-1}(d_{i+1})) - max(L^*_k(d_i), L^*_k(d_{i+1})) \geq 0$$

- RANGE

$$|rw[j]| \leq r\rho d < r\rho \ \ (\text{for } d < 1)$$

$$-r\rho - \epsilon_{max} < \widehat{y} < r\rho - \epsilon_{min}$$

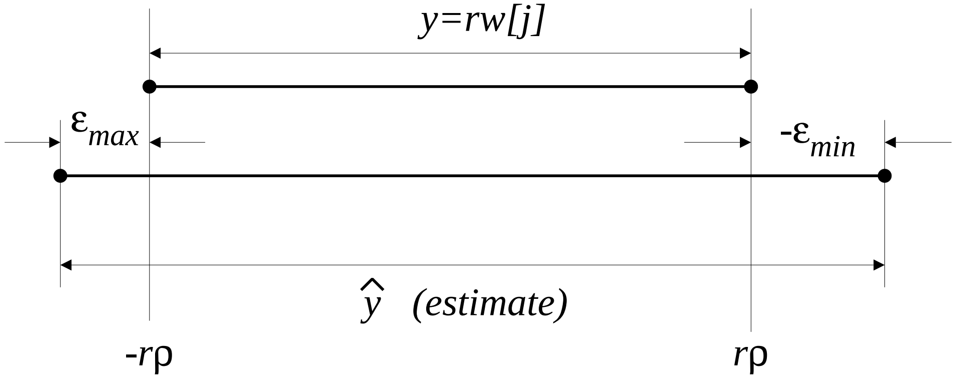


Figure 5.27: RANGE OF ESTIMATE

REDUNDANT ADDER

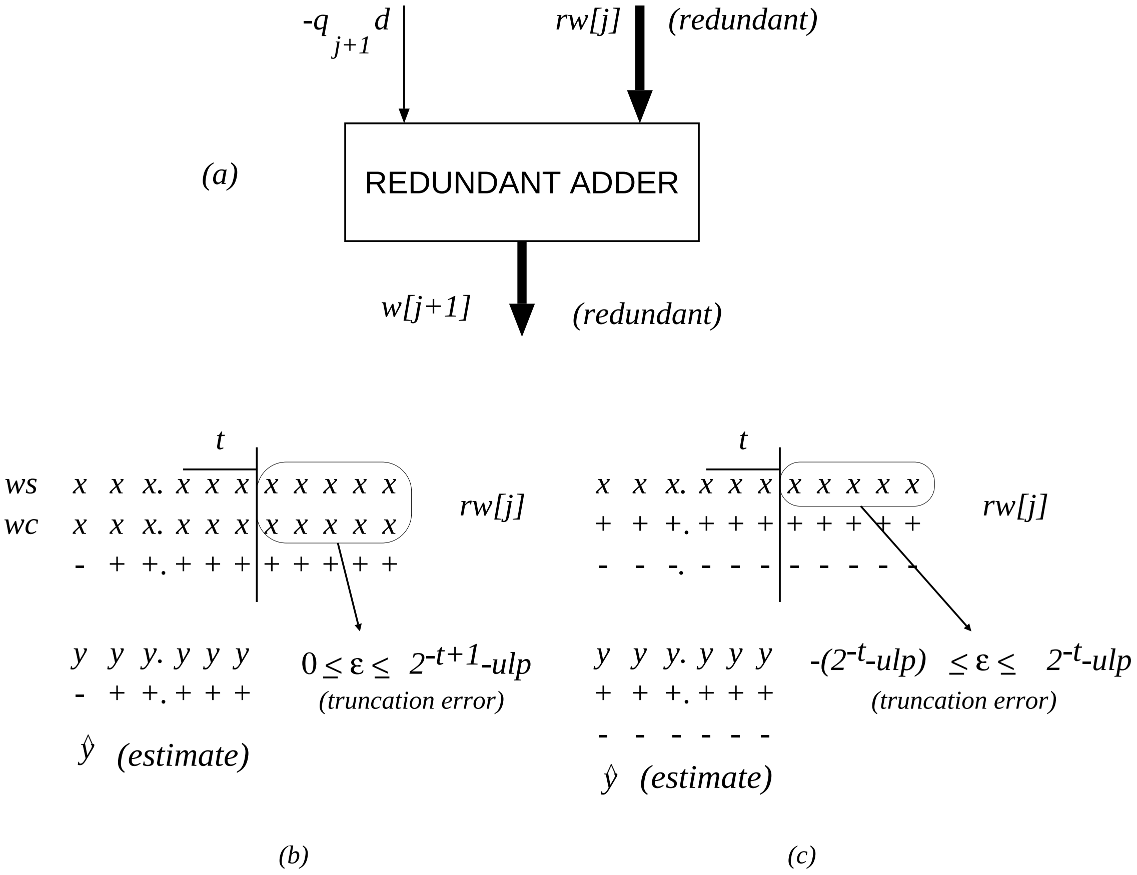


Figure 5.28: USE OF REDUNDANT ADDER: (a) Redundant adder. (b) Carry-save case. (c) Signed-digit case.

CARRY-SAVE ADDER

- ERRORS

$$\epsilon_{min} = 0 \qquad \epsilon_{max} = 2^{-t+1} - ulp$$

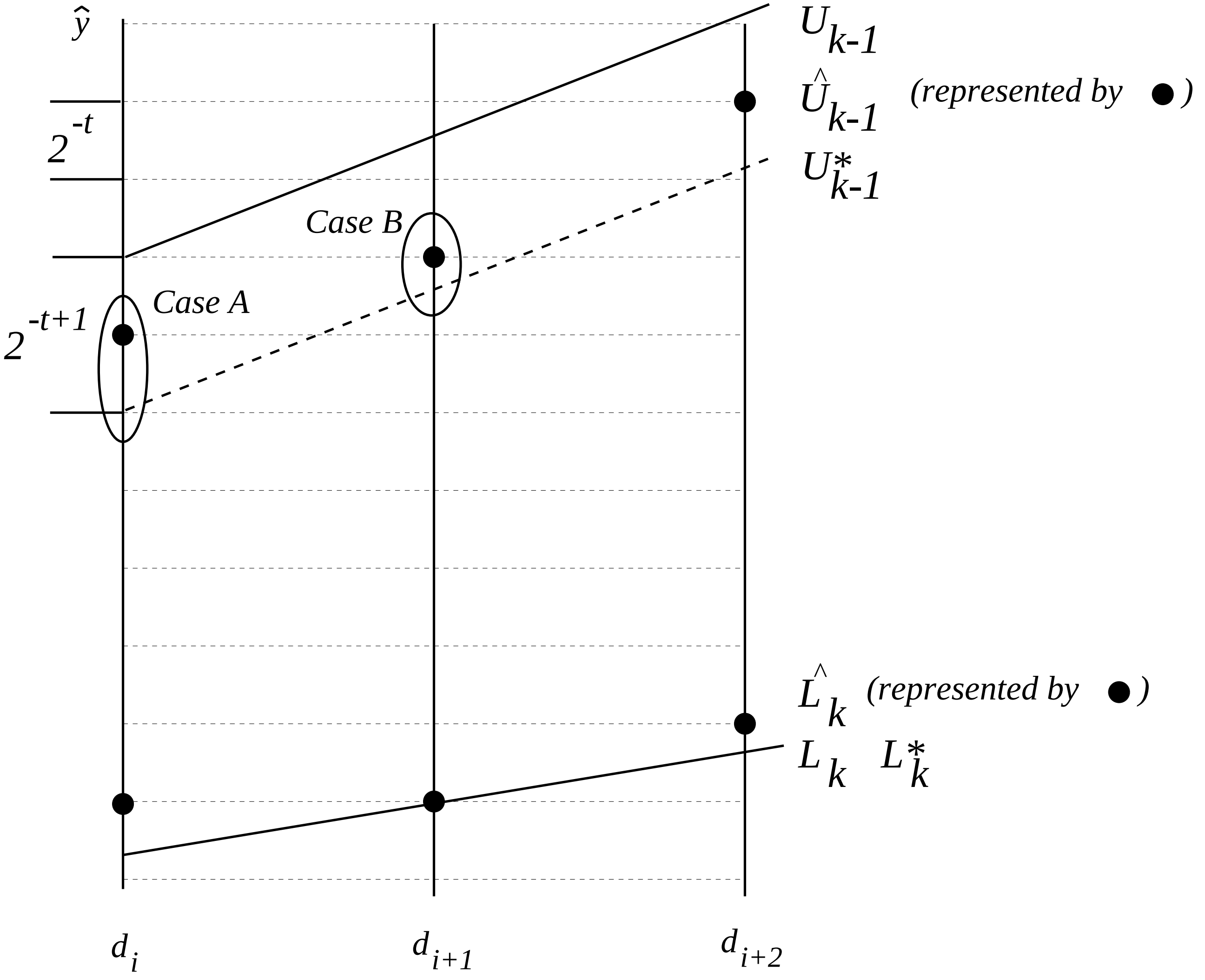
- RESTRICTED SELECTION INTERVAL

$$U_k^* = U_k - 2^{-t+1} + ulp$$

$$L_k^* = L_k$$

$$\widehat{U}_{k-1} = \lfloor U_{k-1}^* + 2^{-t} \rfloor_t = \lfloor U_{k-1} - 2^{-t} \rfloor_t$$

$$\widehat{L}_k = \lceil L_k^* \rceil_t = \lceil L_k \rceil_t$$



Case A: U^*_{k-1} *is on the grid;*

$\hat{U}_{k-1} = U^*_{k-1} + 2^{-t}$ *on the grid*

Case B: U^*_{k-1} *is off the grid;*

$\hat{U}_{k-1} > U^*_{k-1}$ *on the grid*

Figure 5.29: $\widehat{U}$ and $\widehat{L}$ for residual in carry-save form.

LOWER BOUND FOR t and δ

(for positive k)

$$\widehat{U}_{k-1}(d_i) - \widehat{L}_k(d_{i+1}) \geq 0$$

$$U_{k-1}(d_i) - 2^{-t} - L_k(d_{i+1}) \geq 0$$

$$\frac{2\rho - 1}{2} - (a - \rho)2^{-\delta} \geq 2^{-t}$$

- RANGE

$$\lfloor -r\rho - 2^{-t} \rfloor_t \leq \widehat{y} \leq \lfloor r\rho - ulp \rfloor_t$$

$\lfloor z \rfloor_t = 2^{-t} \lfloor 2^t z \rfloor$

RADIX-2 DIVISION WITH CARRY-SAVE ADDER

$$\frac{1}{2} - 0 \times 2^{-\delta} \geq 2^{-t}$$

$$max(\widehat{L}_k(d_i), \widehat{L}_k(d_{i+1})) \leq m_k(i) \leq min(\widehat{U}_{k-1}(d_i), \widehat{U}_{k-1}(d_{i+1}))$$

$$\begin{aligned} \widehat{L}_1(1) &= 0 \\ \widehat{U}_0(1/2) &= 0 \\ \widehat{L}_0(1/2) &= -1/2 \\ \widehat{U}_{-1}(1) &= -1/2 \end{aligned}$$

$$(\widehat{L}_1(1) = 0) \leq m_1 \leq (\widehat{U}_0(1/2) = 0)$$

$$(\widehat{L}_0(1/2) = -1/2) \leq m_0 \leq (\widehat{U}_{-1}(1) = -1/2)$$

This results in the selection constants $m_1 = 0$ and $m_0 = -1/2$

SELECTION FUNCTION

$$\lfloor -2 - 2^{-1} \rfloor_1 \leq \widehat{y} \leq \lfloor 2 - ulp \rfloor_1$$

$$-\frac{5}{2} \leq \widehat{y} \leq 3/2$$

$$q_{j+1} = \begin{cases} 1 & \text{if } \; 0 \leq \widehat{y} \leq 3/2 \\ 0 & \text{if } \; \widehat{y} = -1/2 \\ -1 & \text{if } \; -5/2 \leq \widehat{y} \leq -1 \end{cases}$$

RADIX-2 WITH CSA: PD PLOT

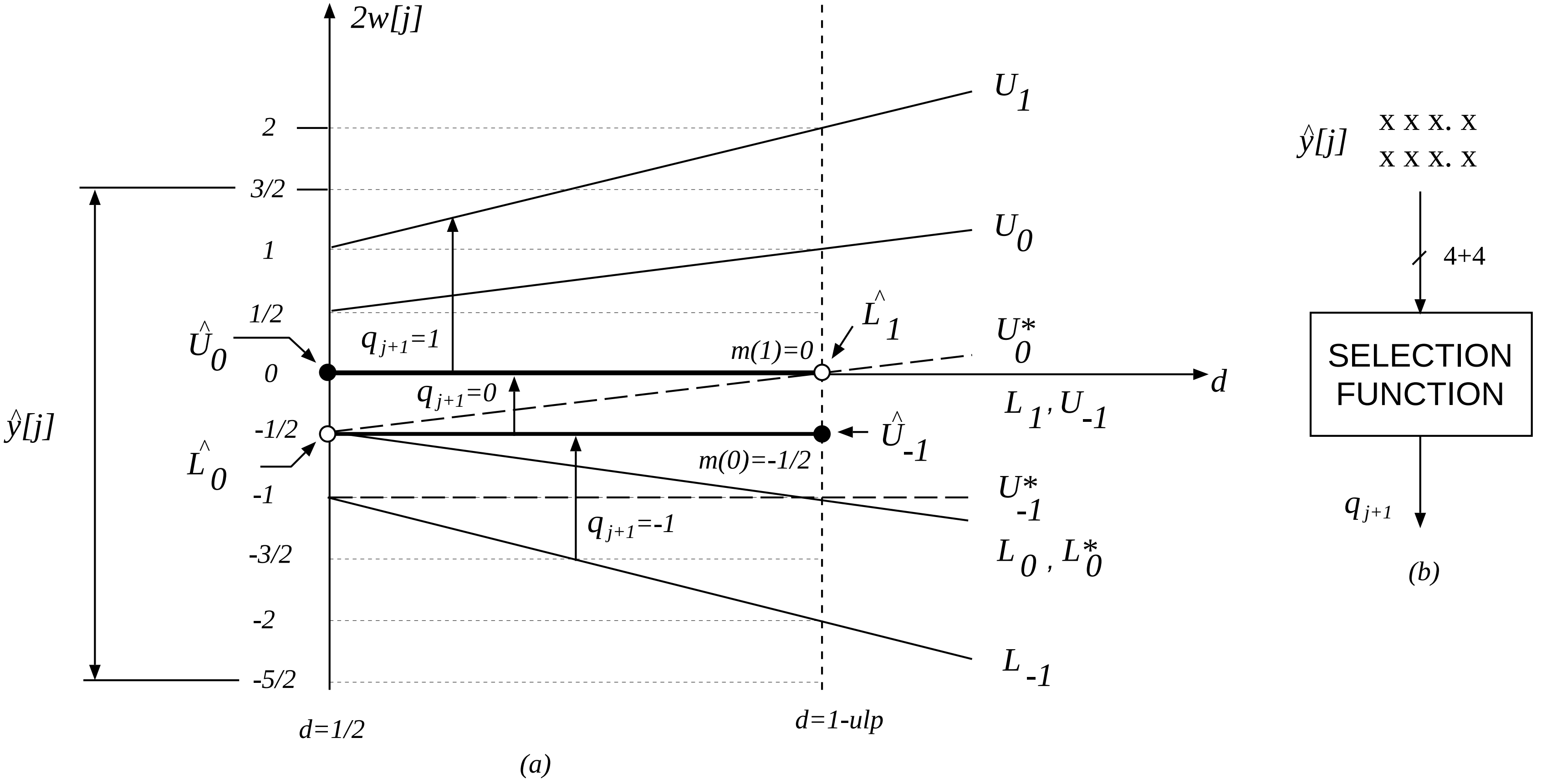


Figure 5.30: RADIX-2 DIVISION WITH CARRY-SAVE ADDER: (a) P-D PLOT. (b) SELECTION FUNCTION.

IMPLEMENTATION

$q_{j+1} = (q_s, q_m)$

$$q_m = (p_{-1}p_0p_1)' \qquad (5.2)$$

$$q_s = p_{-2} \oplus (g_{-1} + p_{-1}g_0 + p_{-1}p_0g_1)$$

where

$$p_i = c_i \oplus s_i \qquad g_i = c_i \cdot s_i$$

and

$$(c_{-2}, c_{-1}, c_0, c_1)$$

$$(s_{-2}, s_{-1}, s_0, s_1)$$

$$q_{j+1} = 0 \qquad (q_s, q_m) = (1, 0)$$

RADIX-4 ALGORITHM WITH CARRY-SAVE ADDER

$$\frac{1}{6} - \frac{4}{3} 2^{-\delta} \geq 2^{-t}$$

$$2^{-t} \leq \frac{1}{6} - \frac{1}{12} = \frac{1}{12}$$

$$\lfloor -8/3 - 1/16 \rfloor_4 \leq \hat{y} \leq \lfloor 8/3 - ulp \rfloor_4$$

$$-\frac{44}{16} \leq \hat{y} \leq \frac{42}{16}$$

$[d_i, d_{i+1})^+$	[8, 9)	[9, 10)	[10, 11)	[11, 12)
$\widehat{L}_2(d_{i+1}), \widehat{U}_1(d_i)^+$	12, 12	14, 14	15, 15	16, 17
$m_2(i)^+$	**12**	**14**	**15**	**16**
$\widehat{L}_1(d_{i+1}), \widehat{U}_0(d_i)^+$	3, 4	4, 5	4, 5	4, 6
$m_1(i)$	**4**	**4**	**4**	**4**
$\widehat{L}_0(d_i), \widehat{U}_{-1}(d_{i+1})^+$	-5, -4	-6, -5	-6, -5	-7, -5
$m_0(i)$	**-4**	**-6**	**-6**	**-6**
$\widehat{L}_{-1}(d_i), \widehat{U}_{-2}(d_{i+1})^+$	-13, -13	-15, -15	-16, -1 6	-18, -17
$m_{-1}(i)$	**-13**	**-15**	**-16**	**-18**

$[d_i, d_{i+1})^+$	[12, 13)	[13, 14)	[14, 15)	[15, 16)
$\widehat{L}_2(d_{i+1}), \widehat{U}_1(d_i)^+$	18, 19	19, 20	20, 22	22, 24
$m_2(i)$	**18**	**20**	**20**	**24**
$\widehat{L}_1(d_{i+1}), \widehat{U}_0(d_i)^+$	4, 7	5, 7	5, 8	6, 9
$m_1(i)$	**6**	**6**	**8**	**8**
$\widehat{L}_0(d_i), \widehat{U}_{-1}(d_{i+1})^+$	-8, -6	-8, -6	-9, -6	-10, -7
$m_0(i)$	**-8**	**-8**	**-8**	**-8**
$\widehat{L}_{-1}(d_i), \widehat{U}_{-2}(d_{i+1})^+$	-20, -19	-21, -20	-23, -2 1	-25, -23
$m_{-1}(i)$	**-20**	**-20**	**-22**	**-24**

+: real value = shown value/16; $\widehat{L}_k = \lceil L_k \rceil_4$, $\widehat{U}_k = \lfloor U_k - \frac{1}{16} \rfloor_4$.

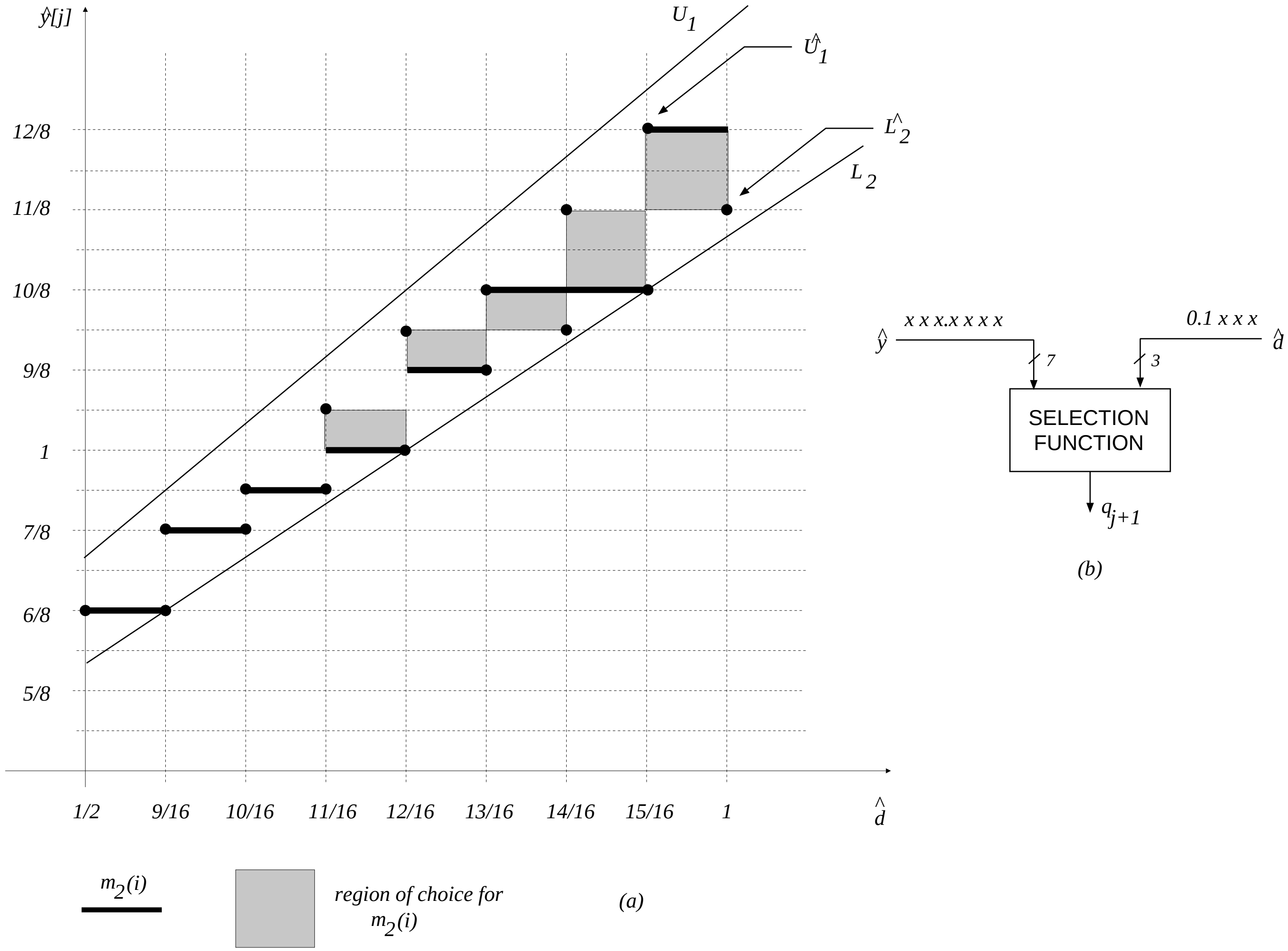


Figure 5.31: SELECTION FUNCTION FOR RADIX-4 SCHEME WITH CARRY-SAVE ADDER: (a) Fragment of P-D diagram. (b) Implementation.

RECIPROCAL, DIVISION, RECIPROCAL SQUARE ROOT AND SQUARE ROOT BY ITERATIVE APPROXIMATION

- AN INITIAL APPROXIMATION OF A FUNCTION ITERATIVELY IMPROVED
- BASED ON MULTIPLICATIONS AND ADDITIONS (vs. only additions and shifts)
- CONVERGES TOWARDS THE RESULT WITH A QUADRATIC OR LINEAR RATE
- QUOTIENT: RECIPROCAL OF THE DIVISOR × THE DIVIDEND
- SQUARE ROOT: INVERSE SQUARE ROOT × THE OPERAND
- ROUNDING HARDER THAN FOR THE DIGIT-RECURRENCE METHOD
- VARIATIONS TO OBTAIN DIRECTLY QUOTIENT AND SQUARE ROOT

NEWTON-RAPHSON'S METHOD FOR RECIPROCAL APPROXIMATION

- BASED ON A GENERAL METHOD TO OBTAIN THE ZERO OF A FUNCTION (THE VALUE OF x FOR WHICH $f(x) = 0$)
- $x[j]$ AN APPROXIMATION OF THE ZERO
- A BETTER APPROXIMATION IS

$$x[j+1] = x[j] - \frac{f(x[j])}{f'(x[j])}$$

  $f'(x[j])$ EVALUATED AT $x[j]$

- APPLY TO RECIPROCAL FUNCTION $f(R) = 1/R - d$ (whose zero is $1/d$)
- RECURRENCE

$$R[j+1] = R[j](2 - R[j]d)$$

- INITIAL APPROXIMATION $R[0]$

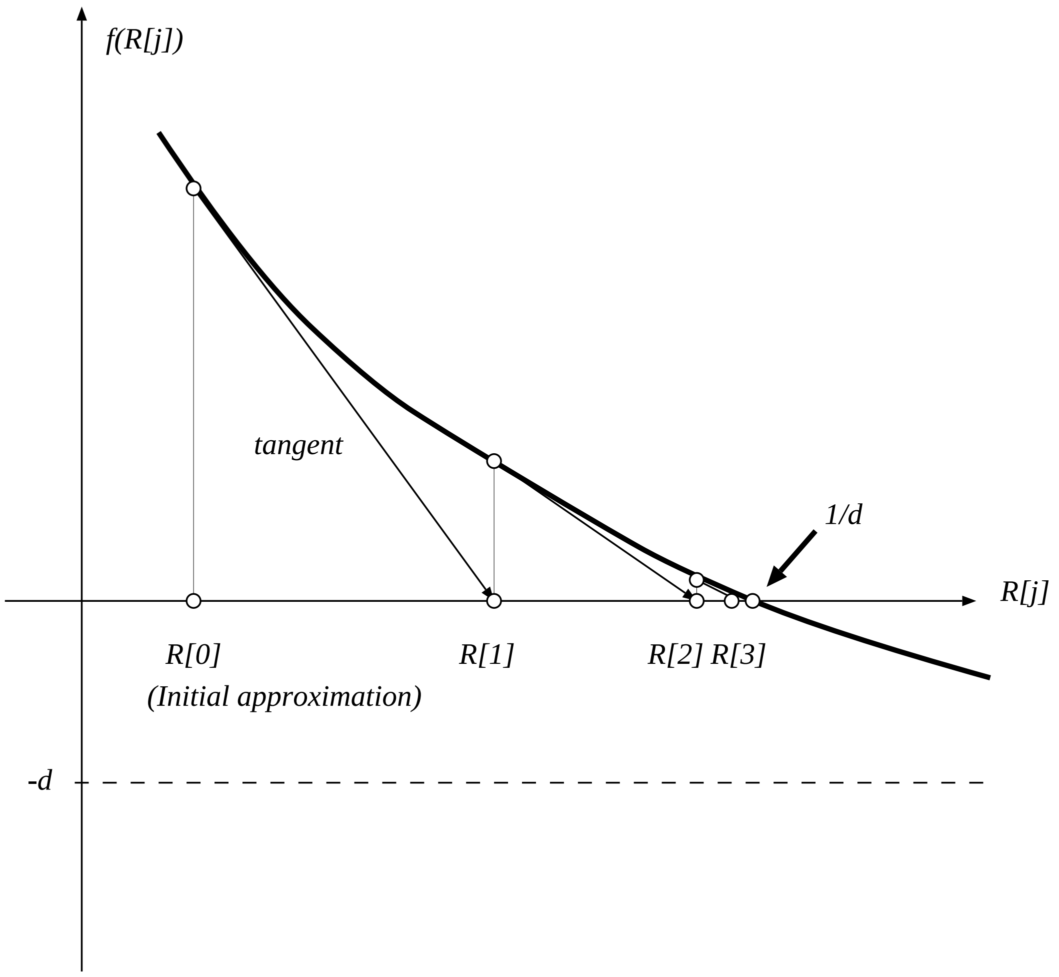


Figure 7.1: Newton-Raphson iteration for finding reciprocal.

RECIPROCAL (cont.)

- EACH ITERATION REQUIRES TWO MULTIPLICATIONS AND ONE SUBTRACTION
- QUADRATIC CONVERGENCE
- RELATIVE ERROR $\epsilon[j]$

$$\epsilon[j] = 1 - dR[j]$$

$$\begin{aligned} R[j+1] &= (\frac{1-\epsilon[j]}{d})(2-(1-\epsilon[j])) \\ &= \frac{1-\epsilon[j]^2}{d} \end{aligned}$$

$\Longrightarrow$

$$\epsilon[j+1] = 1 - dR[j+1] = \epsilon[j]^2$$

RECIPROCAL (cont.)

- NUMBER OF ITERATIONS DEPENDS ON INITIAL APPROXIMATION

$$\epsilon[0] \leq 2^{-k}$$

- TO GET AN ERROR

$$\epsilon[m] \leq 2^{-n}$$

THE NUMBER OF ITERATIONS IS

$$m = \lceil log_2(\frac{n}{k}) \rceil$$

EXAMPLE

RECIPROCAL OF $d = 5/8$

$R[0] = 1$

j	$R[j]$	$dR[j]$	$2 - dR[j]$	$R[j+1]$	$\epsilon[j+1]$
0	1	5×2^{-3}	11×2^{-3}	11×2^{-3}	0.14
1	11×2^{-3}	55×2^{-6}	73×2^{-6}	$803 \times 2^{-9} = 1.5683594$	0. 020
2	803×2^{-9}	4015×2^{-12}	4177×2^{-12}	$3354131 \times 2^{-21} = 1.5993743...$	0.00039

EXACT RESULT: $1/d = 8/5 = 1.6$

MULTIPLICATIVE NORMALIZATION METHOD

- $R = \frac{1}{d} = \frac{1}{d}\frac{P[0]}{P[0]}\frac{P[1]}{P[1]}\ldots\frac{P[m]}{P[m]} = \frac{R[m]}{d[m]}$

  $R = R[m]$ if $d[m] = 1$

- DEFINE APPROXIMATION $R[j] = \Pi_{i=0}^{j} P[i]$ AND $d[j] = dR[j]$

- IMPROVE APPROXIMATION BY

$$\begin{aligned} R[j+1] &= R[j]P[j+1] \\ d[j+1] &= d[j]P[j+1] \end{aligned}$$

MULTIPLICATIVE NORMALIZATION: RECIPROCAL

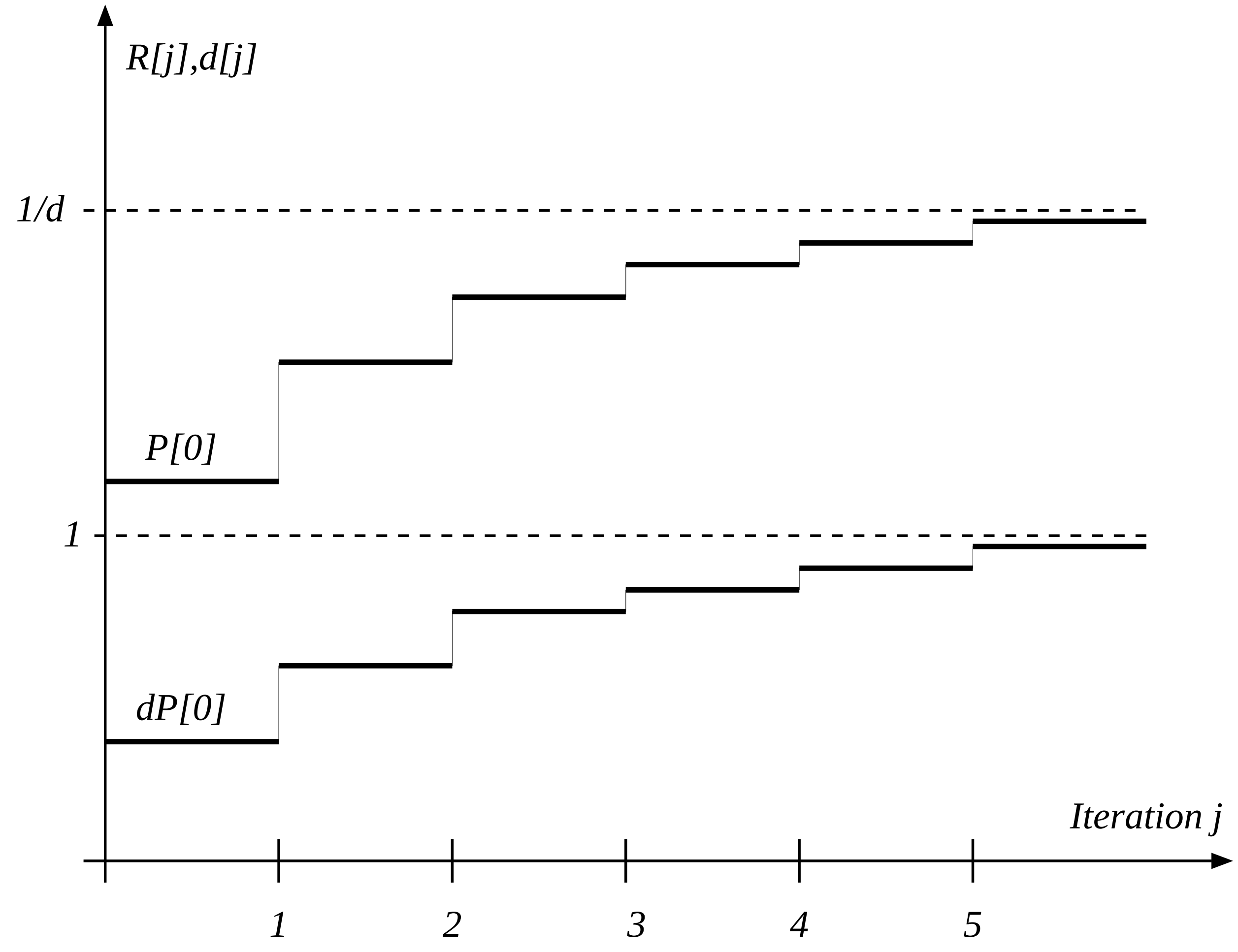


Figure 7.2: Illustration of iterations in the multiplicative normalization method.

DETERMINATION OF $P[j]$ FOR QUADRATIC CONVERGENCE

- DEFINE

$$d[j] = d\prod_{i=0}^{j-1} P[i]$$

- OBTAIN THE RECURRENCE

$$d[j] = d[j-1]P[j-1]$$

- FOR QUADRATIC CONVERGENCE, IF

$$d[j-1] = 1 - \epsilon[j-1]$$

THEN

$$d[j] = 1 - \epsilon[j-1]^2$$

- CONSEQUENTLY,

$$P[j-1] = 1 + \epsilon[j-1]$$

AND

$$d[j-1] + P[j-1] = 1 - \epsilon[i-1] + 1 + \epsilon[i-1] = 2$$

SO THAT

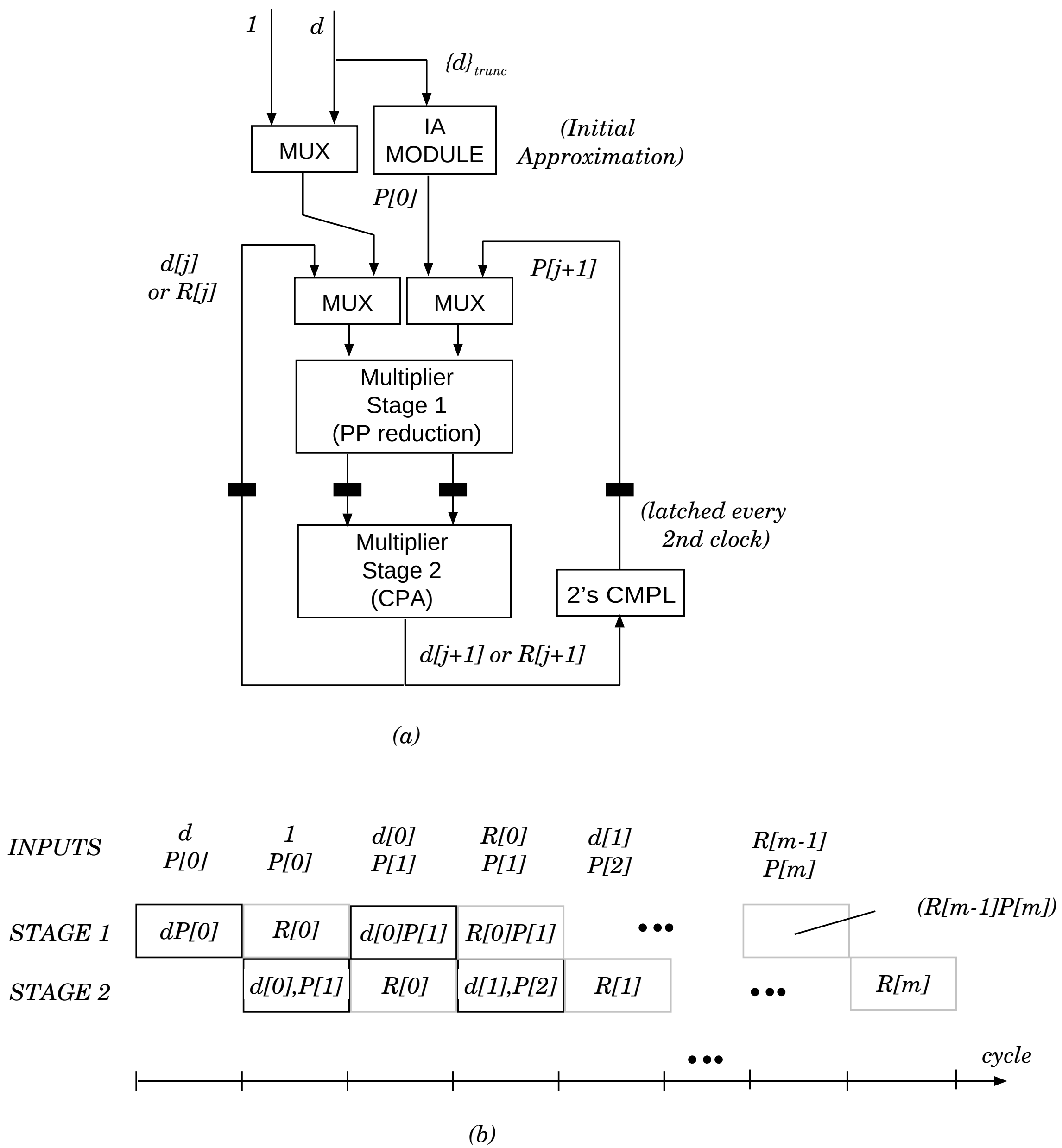
$$P[j-1] = 2 - d[j-1]$$

MULTIPLICATIVE ALGORITHM FOR RECIPROCAL

1. Obtain approximation $P[0]$ to $1/d$
2. $d[0] = dP[0]$; $R[0] = P[0]$
3. For $j = 0, 1, 2, 3, ..., m-2$ do

$$\begin{aligned} P[j+1] &= 2 - d[j] \\ d[j+1] &= d[j]P[j+1]; \quad R[j+1] = R[j]P[j+1] \end{aligned}$$

4. $P[m] = 2 - d[m-1]; R[m] = R[m-1]P[m]$



(a)

(b)

Figure 7.3: Multiplicative normalization for reciprocal: (a) Implementation with a 2-stage multiplier. (b) Timing diagram.

INITIAL APPROXIMATION

1. PERFORM A TABLE LOOK-UP BASED ON TRUNCATED d

   - GOOD FOR RELATIVELY LOW PRECISION INITIAL APPROXIMATION
   - PIECEWISE LINEAR APPROXIMATION IF TABLE TOO LARGE

   $$d = d_t 2^{-k} + d_p 2^{-p} + d_r 2^{-n}$$

   MS k bits of d used to access the table to get coefficients a and b. Then

   $$R[0] = a + b d_p 2^{-p}$$

   - REQUIRES A TABLE LOOK-UP AND A SMALL MULTIPLICATION

2. BIPARTITE METHOD: OBTAIN TWO VALUES FROM TABLES AND PERFORM AN ADDITION

   - USES LARGER TABLES AND ADDER

IMPLEMENTATION AND EXECUTION TIME

- MODULE TO COMPUTE THE INITIAL APPROXIMATION
- MULTIPLIER
- WIDTH OF PRODUCTS:

	R[j]	R[j]d	R[j+1]= R[j](2-R[j]d)
j=0	a	a+n	2a+n
j=1	2a+n	2a+2n	4a+3n
j=2	4a+3n	4a+4n	8a+7n
....			

- AT ITERATION j APPROXIMATION HAS A PRECISION OF $2^j a$ BITS
  $\Longrightarrow$OK TO KEEP PRODUCTS AT THIS PRECISION
  $\Longrightarrow$NEED NOT PERFORM MULTIPLICATIONS AT FULL PRECISION

ALTERNATIVES

1. USE A FLOATING-POINT MULTIPLIER PRODUCING A ROUNDED PRODUCT
2. USE A RECTANGULAR MULTIPLIER
   - A SEQUENCE OF MULTIPLICATIONS AS PRECISION INCREASES
   - RECTANGULAR MULTIPLIER SMALLER AND FASTER THAN THE SQUARE MULTIPLIER

COMPARISON OF NUMBER OF CYCLES FOR FULL AND RECTANGULAR MULTIPLIER ALTERNATIVES

- RECIPROCAL OF 54 BITS STARTING WITH $r[0]$ ACCURATE TO 8 BITS
- MULTIPLIER IN SCHEME A STANDARD FLOATING-POINT MULTIPLIER
- MULTIPLIER IN SCHEME B A DEDICATED MULTIPLIER
- OPERATION REQUIRES AT LEAST THREE ITERATIONS

  EACH CONSISTING OF TWO CONSECUTIVE MULTIPLICATIONS

  IGNORE THE DELAY OF OBTAINING $2 - R[i]d$

COMPARISON OF ALTERNATIVES (cont.)

1. SCHEME A: Full multiplier $55 \times 55 \rightarrow 55$ (rounded);
   3 cycles per multiply; total: 1 $+$ 3 x 2 x 3 $=$ 19 cycles
2. SCHEME B: Rectangular multiplier $55 \times 16 \rightarrow 55$;
   1 cycle per multiply; total: 1$+$ 2 $+$ 2 $+$ 4 $=$ 9 cycles

- $R[1] = R[0](2 - dR[0])$ to 16 bits we use 55×16 multiplier twice (2 cycles);
- $R[2] = R[1](2 - dR[1])$ to 32 bits we use 55×16 multiplier twice (2 cycles);
- $R[3] = R[2](2 - dR[2])$ to 54 bits we use 55×16 multiplier four times (4 cycles).
- A COMPLEMENTER (2's OR 1s')

DIVISION

- TO GET QUOTIENT

$$Q = R[m]x$$

EXAMPLE OF IMPLEMENTATION: AMD-K7 FLPT UNIT

- DIVISION (20 CYCLES) AND SQUARE ROOT (27 CYCLES)
- DOUBLE PRECISION (53 bits); INTERNAL PRECISION: 76 bits (FOR EXTENDED FORMAT)
- USES 4-STAGE PIPELINED MULTIPLIER: 76 × 76 PRODUCING 152 BITS
- RADIX-8 MULTIPLIER RECODING WITH {-4, ..., 4}
- INITIAL APPROXIMATION: BIPARTITE TABLE LOOKUP (69K BITS + ADDER)

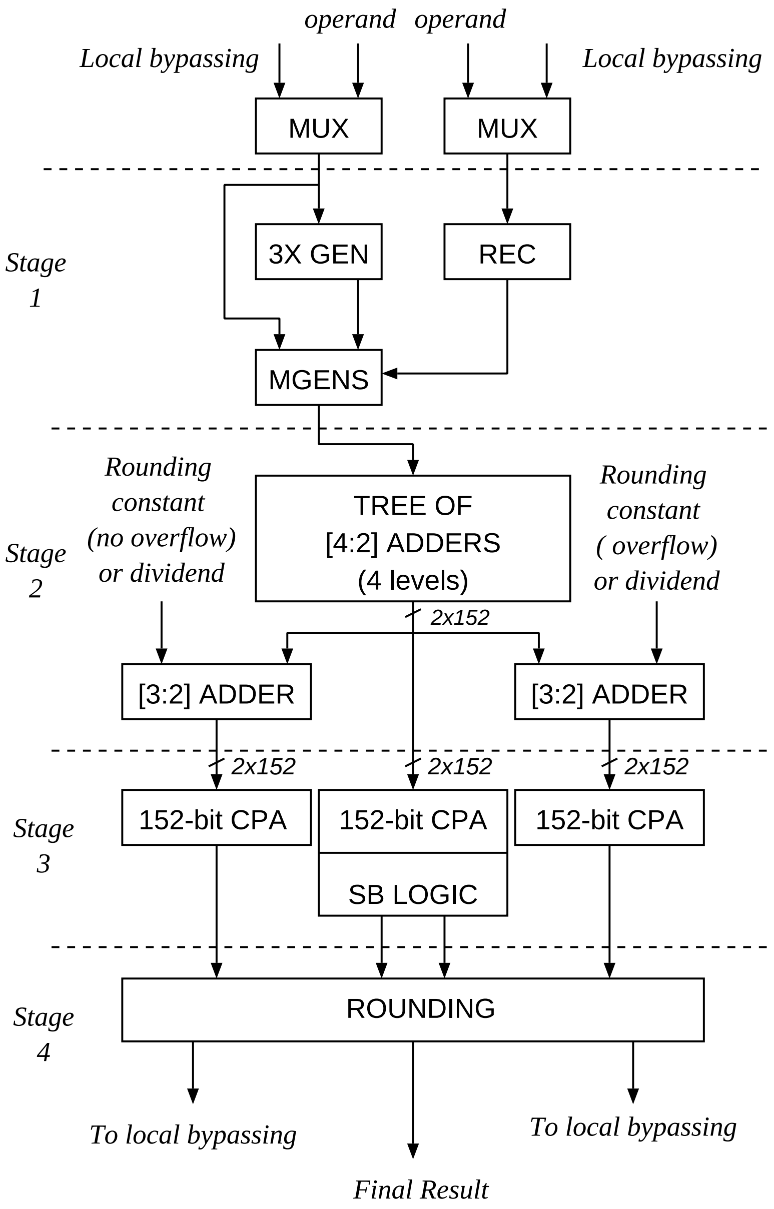


Figure 7.4: Block diagram of a division/square-root unit (Adapted from Oberman 1999)

```
1. [$Initialize$]
     $P[0] \leftarrow RECIP(\hat{d})$
     $d[0] \leftarrow d;\ q[0] \leftarrow x$
2. [$Iterate$]
   for  $j = 0, 1$
     $d[j+1] \leftarrow d[j] \times p[j];\ q[j+1] \leftarrow q[j] \times p[j]$
     $p[j+1] = CMPL(d[j+1])$
   end for
3. [$Terminate$]
     $q[3] \leftarrow q[2] \times p[2]$
     $REM \leftarrow d \times q[3] - x$
     $q \leftarrow ROUND(q[3], REM, mode)$
```

where

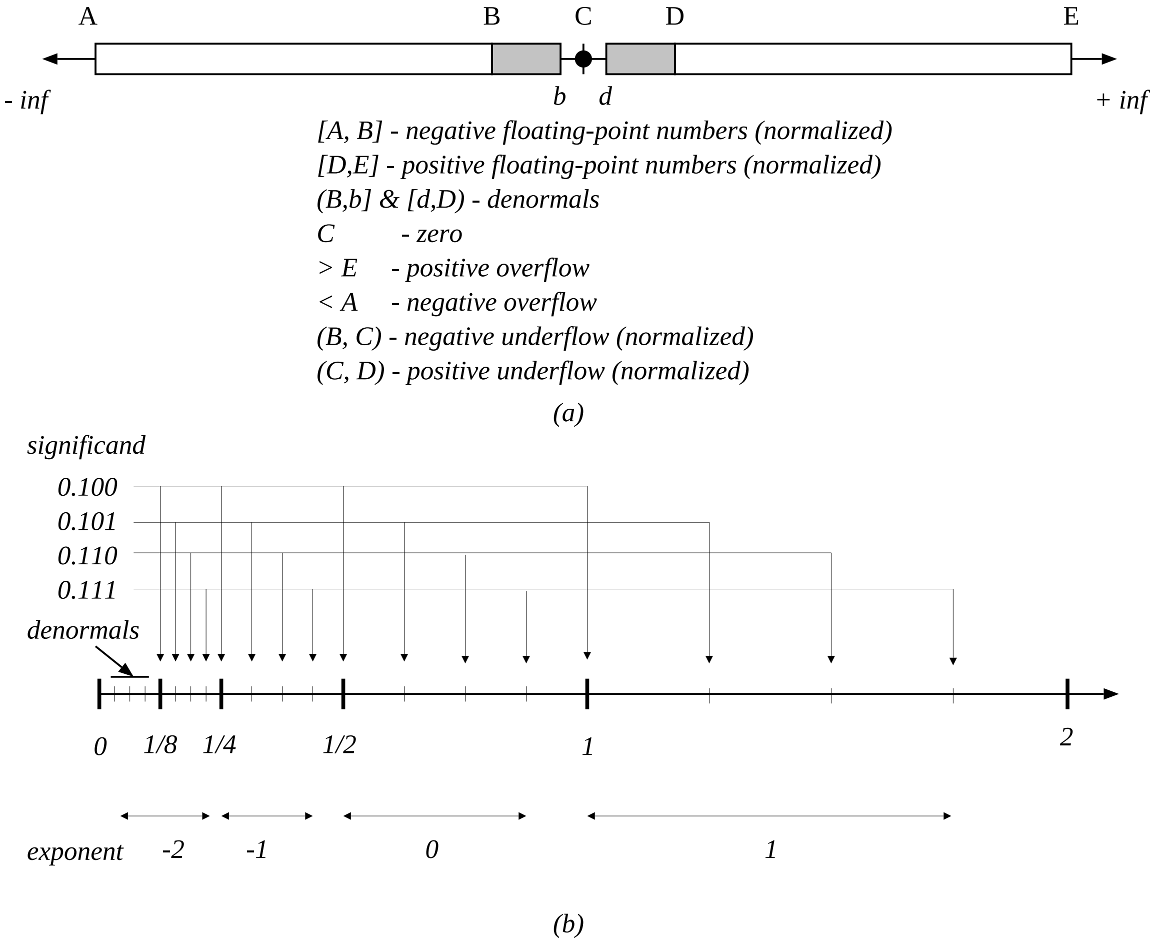
- $RECIP$ produces the initial approximation of $1/d$ in three cycles.
- $CMPL(a)$ performs bit complementation of a.
- REM is a negated remainder.
- $ROUND$ produces a quotient rounded according to the specified $mode$

Figure 7.5: Multiplicative division algorithm (double precision).

FLOATING-POINT ARITHMETIC

- Floating-point representation and dynamic range
- Normalized/unnormalized formats
- Values represented and their distribution
- Choice of base
- Representation of significand and of exponent
- Rounding modes and error analysis
- IEEE Standard 754
- Algorithms and implementations: addition/subtraction, multiplication and division

VALUES REPRESENTED IN FLPT SYSTEM



[A, B] - negative floating-point numbers (normalized)
[D,E] - positive floating-point numbers (normalized)
(B,b] & [d,D) - denormals
C - zero
> E - positive overflow
< A - negative overflow
(B, C) - negative underflow (normalized)
(C, D) - positive underflow (normalized)

Figure 8.1: a) Regions in floating-point representation. b) Example for $m = f = 3$, $r = 2$, and $-2 \leq E \leq 1$ (only positive region).

	Floating-point system	
	Normalized	Unnormalized
A	$-(r^{m-f} - r^{-f}) \times b^{Emax}$	
B	$-r^{m-f-1} \times b^{Emin}$	$-r^{-f} \times b^{Emin}$
C	0	
D	$r^{m-f-1} \times b^{Emin}$	$r^{-f} \times b^{Emin}$
E	$(r^{m-f} - r^{-f}) \times b^{Emax}$	

DISTRIBUTION FOR $b = 2$, $m = f = 4$, and $e = 2$

Significand	2^E			
	1	2	4	8
0.1000	1/2	1	2	4
0.1001	9/16	9/8	9/4	9/2
0.1010	10/16	10/8	10/4	5
0.1011	11/16	11/8	11/4	11/2
0.1100	12/16	12/8	3	6
0.1101	13/16	13/8	13/4	13/2
0.1110	14/16	14/8	14/4	7
0.1111	15/16	15/8	15/4	15/2

DISTRIBUTION FOR $b = 2$, $m = f = 3$, and $e = 3$

Significand	2^E							
	1	2	4	8	16	32	64	128
0.100	1/2	1	2	4	8	16	32	64
0.101	5/8	5/4	5/2	5	10	20	40	80
0.110	6/8	3/2	3	6	12	24	48	96
0.111	7/8	7/4	7/2	7	14	28	56	112

DISTRIBUTION FOR $b = 4$, $m = f = 4$, and $e = 2$

Significand	4^E			
	1	4	16	64
0.0100	1/4	1	4	16
0.0101	5/16	5/4	5	20
0.0110	6/16	6/4	6	24
0.0111	7/16	7/4	7	28
0.1000	1/2	2	8	32
0.1001	9/16	9/4	9	36
0.1010	10/16	10/4	10	40
0.1011	11/16	11/4	11	44
0.1100	12/16	3	12	48
0.1101	13/16	13/4	13	52
0.1110	14/16	14/4	14	56
0.1111	15/16	15/4	15	60

DISTRIBUTION OF FLPT NUMBERS

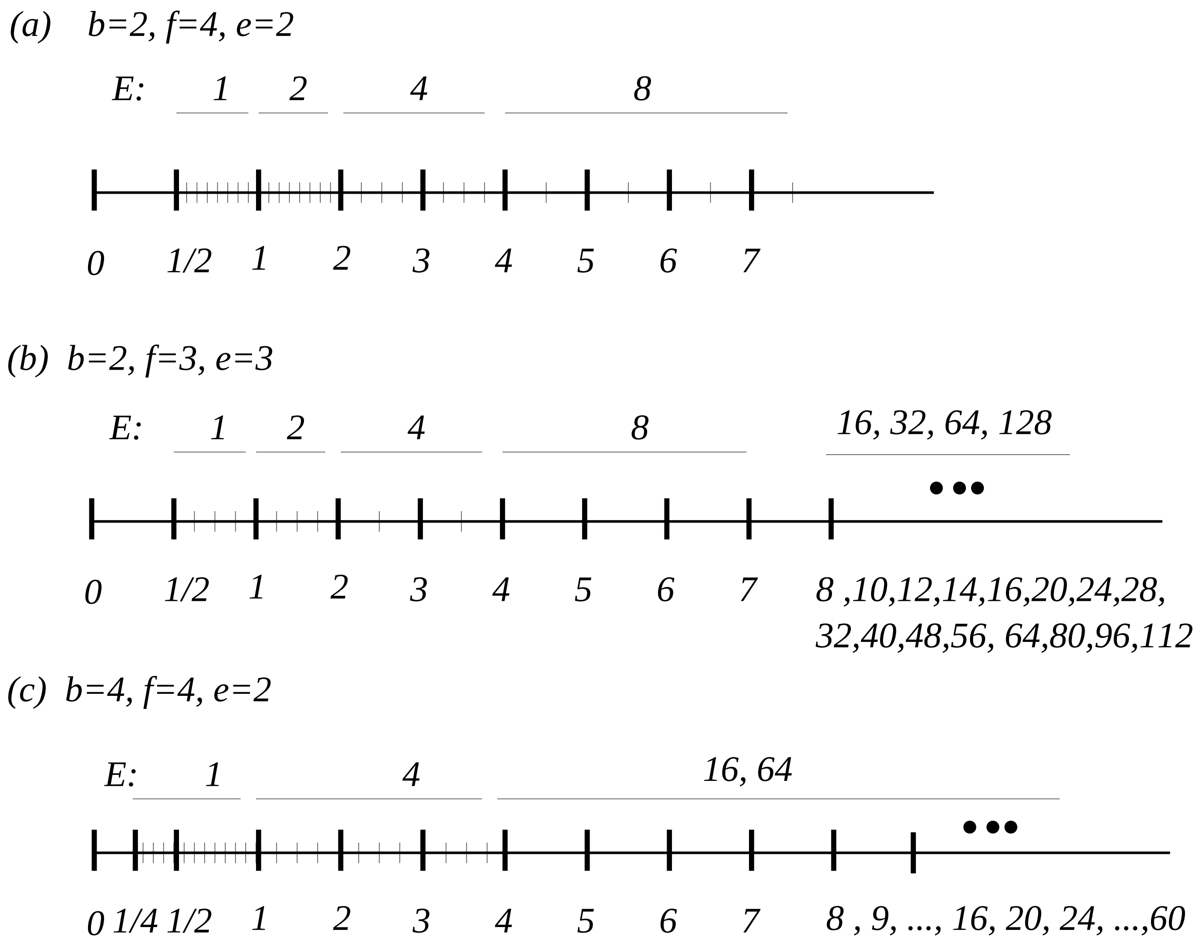


Figure 8.2: EXAMPLES OF DISTRIBUTIONS OF FLOATING-POINT NUMBERS.

REPRESENTATION OF SIGNIFICAND AND EXPONENT

- SIGNIFICAND: SM with HIDDEN BIT

- EXPONENT: BIASED $E_R = E + B$, min$E_R = 0 \quad \Rightarrow \; B = -E_{min}$
- Symmetric range $-B \leq E \leq B \quad \Rightarrow \; 0 \leq E_R \leq 2B \leq 2^e - 1$
- for 8-bit exponent: $B = 127$, $-127 \leq E \leq 128$, $0 \leq E_R \leq 255$
- $E_R = 255$ not used

- SIMPLIFIES COMPARISON OF FLOATING-POINT NUMBERS (same as in fixed-point)

- MINIMUM EXPONENT REPRESENTED BY 0 SO THAT FLOATING-POINT VALUE 0: ALL ZEROS
  (0 sign, 0 exponent, 0 significand)

SPECIAL VALUES AND EXCEPTIONS

- Special values - not representable in the FLPT system
  - NAN (Not A Number)
  - Infinity (pos, neg)
  - allow computation in presence of special values
- Exceptions: result produced not representable - set a flag
  - Exponent overflow
  - Underflow

ROUNDOFF MODES AND ERROR ANALYSIS

- Exact results (inf. precision): x, y, etc.
- FLPT number representing x is $R_{mode}(x)$ with rounding mode $mode$
- Basic relations:

  1. If $x \leq y$ then $R_{mode}(x) \leq R_{mode}(y)$
  2. If x is a FLPT number then $R_{mode}(x) = x$
  3. If $F1$ and $F2$ are two consecutive FLPT numbers then for $F1 \leq x \leq F2$ x is either $F1$ or $F2$

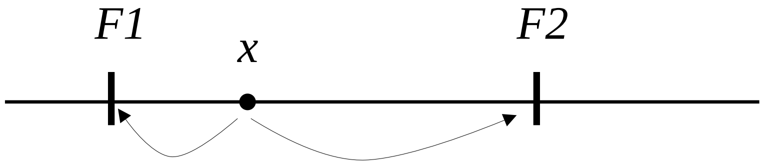


Figure 8.3: Relation between x, $Rmode(x)$, and floating-point numbers $F1$ and $F2$.

ROUNDING MODES

- Round to nearest (tie to even). $Rnear(x)$ is the floating-point number that is closest to x.

$$Rnear(x) = \begin{cases} F1 & \text{if } |x - F1| < |x - F2| \\ F2 & \text{if } |x - F1| > |x - F2| \\ even(F1, F2) & \text{if } |x - F1| = |x - F2| \end{cases}$$

- Round toward zero (truncate). $Rtrunc(x)$ is the closest to 0 among $F1$ and $F2$.

$$Rtrunc(x) = \begin{cases} F1 & \text{if } x \geq 0 \\ F2 & \text{if } x < 0 \end{cases}$$

- Round toward plus infinity. $Rpinf(x)$ is the largest among $F1$ and $F2$

$$Rpinf(x) = F2$$

- Round toward minus infinity. $Rninf(x)$ is the smallest among $F1$ and $F2$

$$Rninf(x) = F1$$

ROUNDING ERRORS

1. The (maximum) absolute representation error ABRE (MABRE))

$$ABRE = Rmode(x) - x$$

so that

$$MABRE = max_x(|ABRE|)$$

2. The average bias (RB)

$$RB = \lim_{t\to\infty} \frac{\Sigma_{M\in\{M_{m+t}\}}(Rmode(M) - M)}{\#M}$$

where $\{M_{m+t}\}$ is the set of all significands with $m+t$ bits, and $\#M$ is the number of significands in the set.

3. The relative representation error (RRE)

$$RRE = \frac{Rmode(x) - x}{x}$$

ERRORS AND IMPLEMENTATION CHARACTERISTICS OF R-MODES

- x described $exactly$ by the triple $(S_x,\ E_x,\ M_x)$
- M_x normalized but having infinite precision
- M_x decomposed into two components M_f and M_d:

$$M_x = M_f + M_d \times r^{-f}$$

- M_f has precision of significand in the FLPT system
- M_d represents the rest, $0 \leq M_d < 1$

ROUNDING TO NEAREST - UNBIASED, TIE TO EVEN

- Value represented - closest possible to the exact value
- The smallest absolute error - the default mode of the IEEE Standard
- Round to nearest specification:

$$Rnear(x) = \begin{cases} M_f + r^{-f} & \text{if } M_d \geq 1/2 \\ M_f & \text{if } M_d < 1/2 \end{cases}$$

- The addition of r^{-f} can produce significand overflow
- Equivalently

$$Rnear(x) = (\lfloor (M_x + \frac{r^{-f}}{2}) r^f \rfloor) r^{-f}$$

- Example: The exact value 1.100100011101 is rounded to nearest with 8-bit precision

```
  1.100100011101
+          1
  --------------
  1.10010010
```

ROUND TO NEAREST (cont.)

- The absolute error is

$$ABRE[Rnear] = \begin{cases} -M_d r^{-f} \times b^E & \text{if } M_d < 1/2 \\ (1 - M_d) r^{-f} \times b^E & \text{if } M_d \geq 1/2 \end{cases}$$

- The maximum absolute error occurs when $M_d = 1/2$

$$MABRE[Rnear] = \frac{r^{-f}}{2} \times b^{Emax}$$

- **unbiased round to nearest**

$$Rnear(x) = \begin{cases} M_f & \text{if } M_d < 1/2 \\ M_f + r^{-f} & \text{if } M_d > 1/2 \\ M_f & \text{if } M_d = 1/2 \text{ and } M_f = \text{even} \\ M_f + r^{-f} & \text{if } M_d = 1/2 \text{ and } M_f = \text{odd} \end{cases}$$

- For this mode

$$RB[Rnear] = 0$$

ROUND TOWARD ZERO (TRUNCATION)

- rounded significand is obtained by discarding M_d.

$$Rzero(x) = (\lfloor M \times r^f \rfloor) r^{-f} = M_f$$

- The absolute error

$$ABRE[Rzero] = -M_d r^{-f} \times b^E$$

and

$$MABRE[Rzero] \approx r^{-f} \times b^{Emax}$$

- Absolute error always negative, the average bias is significant

$$AB[Rzero] \approx -\frac{1}{2} r^{-f}$$

ROUND TOWARD PLUS/MINUS INFINITY

- These two directed modes useful for interval arithmetic (operands and the result of an operation are intervals)
- This permits the monitoring of the accuracy of the result
- Specs:

$$Rpinf(x) = \begin{cases} M_f + r^{-f} & \text{if } M_d > 0 \text{ and } S = 0 \\ M_f & \text{if } M_d = 0 \text{ or } S = 1 \end{cases}$$

$$Rninf(x) = \begin{cases} M_f + r^{-f} & \text{if } M_d > 0 \text{ and } S = 1 \\ M_f & \text{if } M_d = 0 \text{ or } S = 0 \end{cases}$$

- The addition of r^{-f} can produce a significand overflow

ILLUSTRATIONS OF ROUNDING MODES

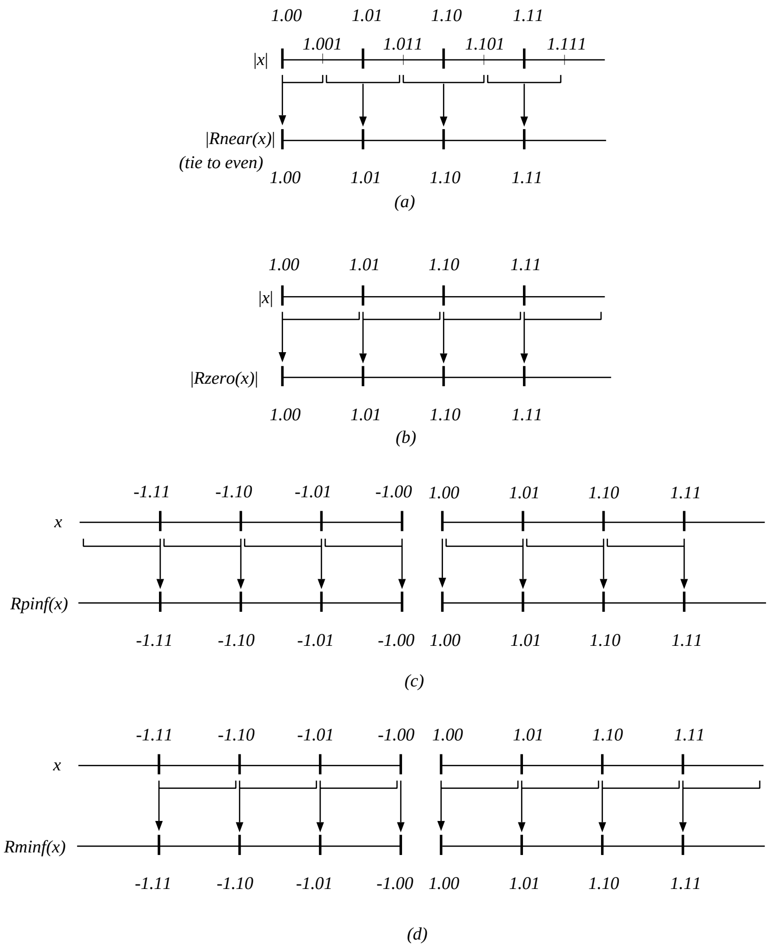


Figure 8.4: ROUNDING TO (a) NEAREST, TIE TO EVEN. (b) ZERO. (c) PLUS INFINITY. (d) MINUS INFINITY.

IEEE FLOATING-POINT STANDARD 754

- Minimizes anomalies
- Enhances portability
- Enhances numerical quality
- Allows different implementations

REPRESENTATION AND FORMATS

1. The significand in SM representation:
   - *Sign* S. One bit. $S = 1$ if negative.
   - *Magnitude* (also called the significand). Represented in radix 2 with one integer bit. That is, the normalized significand is represented by

     $$1.F$$

     where F of f bits (depending on the format) is called the **fraction** and the most-significant 1 is the **hidden bit**.

     The range of the (normalized) significand

     $$1 \leq 1.F \leq 2 - 2^{-f}$$

2. *Exponent*. Base 2 and biased representation; the exponent field e, depending of the format; biased with bias $B = 2^{e-1} - 1$.

SPECIAL VALUES

- The representation of floating-point zero: $E = 0$ and $F = 0$. The sign S differentiates between positive and negative zero.
- The representation $E = 0$ and $F \neq 0$ used for denormals; in this case the floating-point value represented is

$$v = (-1)^S 2^{-(B-1)}(0.F)$$

- The maximum exponent representation $\left(E = 2^e - 1\right)$ represents not-a-number (NAN) for $F \neq 0$ and plus and minus infinity for $F = 0$.

BASIC AND EXTENDED FORMATS

- The basic format allows representation in single and double precision

1. Basic: single (32 bits) and double (64 bits)

   - single: S(1),E(8),F(23)

     (a) If $1 \leq E \leq 254$, then $v = (-1)^S 2^{E-127}(1.F)$ (normalized fp number)

     (b) If $E = 255$ and $F \neq 0$, then $v = NAN$ (not a number)

     (c) If $E = 255$ and $F = 0$, then $v = (-1)^S \infty$ (plus and minus infinity)

     (d) If $E = 0$ and $F \neq 0$, then $v = (-1)^S 2^{-126}(0.F)$ (denormal, gradual underflow)

     (e) If $E = 0$ and $F = 0$, then $v = (-1)^S 0$ (positive and negative zero)

   - double: S(1) E(11) F(52)

     – Similar representation to single, replacing 255 by 2047, etc.

2. Extended: single (at least 43=1+11+31) and double (at least 79=1+15+63)

ROUNDING, OPERATIONS, AND EXCEPTIONS

- Rounding

  Default Mode:

  round to nearest, to even when tie

  Directed modes:

  round toward plus infinity

  round toward minus infinity

  round toward 0 (truncate)

- Operations

  Numerical:

  Add, Sub, Mult, Div, Square root, Rem

  Conversions

  Floating to integer

  Binary to decimal (integer)

  Binary to decimal (floating)

cont.

Miscellaneous

Change formats

Compare and set condition code

- Exceptions: By default set a flag and the computation continues

Overflow (when rounded value too large to be represented). Result is set to ± infinity.

Underflow (when rounded value too small to be represented)

Division by zero

Inexact result (result is not an exact floating-point number). Infinite precision result different than floating-point number.

Invalid. This flag is set when a NAN result is produced.

FLOATING-POINT ADDITION/SUBTRACTION

- x and y - normalized operands represented by (S_x, M_x, E_x) and (S_y, M_y, E_y)

1. Add/subtract significand and set exponent

$$\begin{aligned} M_z^* &= \begin{cases} (M_x^* \pm (M_y^*) \times (b^{E_y - E_x})) \times b^{E_x} & \text{if } E_x \geq E_y \\ ((M_x^*) \times (b^{E_x - E_y}) \pm M_y^*) \times b^{E_y} & \text{if } E_x < E_y \end{cases} \\ E_z &= max(E_x, E_y) \end{aligned}$$

```
Ex - Ey = 4

Mx                1.xxxxxxxxxxx
My(2^(Ey-Ex))     0.0001yyyyyyyyyyy
                  -----------------
                  z.zzzzzzzzzzzzzzz
```

2. Normalize significand and update exponent.
3. Round, normalize and adjust exponent.
4. Set flags for special cases.

BASIC ALGORITHM

1. **Subtract exponents** ($d = E_x - E_y$).
2. **Align significands**
   - Shift right d positions the significand of the operand with the smallest exponent.
   - Select as exponent of the result the largest exponent.
3. **Add (Subtract) significands and produce sign of result**. The effective operation (add or subtract):

Floating-point op.	Signs of operands	Effective operation (EOP)
ADD	equal	add
ADD	different	subtract
SUB	equal	subtract
SUB	different	add

cont.

4. **Normalization of result**. Three situations can occur:

   (a) The result already normalized: no action is needed

```
           1.10011111
           0.00101011
      ADD  ----------
           1.11001010
```

   (b) Effective operation addition: there might be an overflow of the significand. The normalization consists in

   - Shift right the significand one position
   - Increment by one the exponent

```
          1.1001111
          0.0110110
      ADD ---------
         10.0000101
    NORM  1.00000101
```

cont.

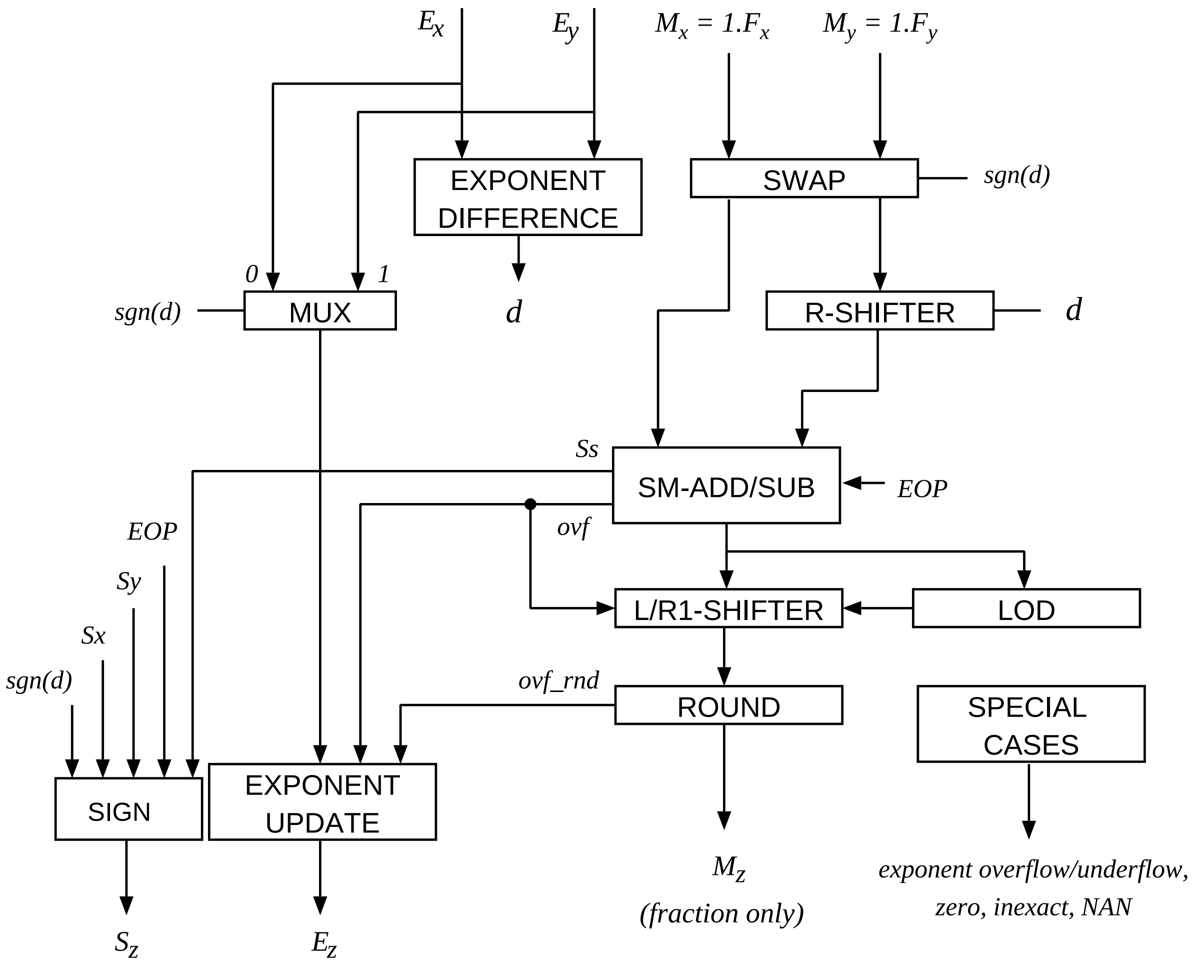
(c) Effective operation subtraction: the result might have leading zeros. Normalize:

- Shift left the significand by a number of positions corresponding to the number of leading zeros.
- Decrement the exponent by the number of leading zeros.

```
         1.1001111
         1.1001010
  SUB    ---------
         0.0000101
 NORM    1.0100000
```

5. **Round**. According to the specified mode. Might require an addition. If overflow occurs, normalize by a right shift and increment the exponent.

6. **Determine exception flags and special values** : exponent overflow (special value ± infinity), exponent underflow (special value gradual underflow), inexact, and the special value zero.



EOP: effective operation
R-SHIFTER: variable right shifter
L/R1-SHIFTER: variable left/one pos. right shifter
LOD: Leading One Detector

Figure 8.5: BASIC IMPLEMENTATION OF FLOATING-POINT ADDITION.

COMMENTS ON BASIC IMPLEMENTATION

- Significand normalized and in SM
- Base of exponent is 2

1. One alignment shifter: swap the significands according to the sign of the exponent difference.
2. The adder: SM adder. Complicated - several options can be used:
   (a) Use a two's complement adder
   (b) Use a ones' complement adder
   (c) Use a two's complement adder; complement the smallest operand so that the result is positive and no complementation is required.

   To determine the smallest operand, two cases:

   - The exponents are different: the operand with smallest exponent shifted right and complemented
   - The exponents are the same: compare the significands in parallel with the alignment

COMMENTS (cont.)

3. The normalization step requires:
   - The detection of the position of the leading 1 uses LOD (Leading-One-Detector)
   - A shift performed by the shifter:
     - no shift
     - right shift of one position, or
     - left shift of up to m positions
4. The rounding step uses several guard bits

GUARD BITS AND ROUNDING

- Keep all $2m$ bits? No, a few additional bits sufficient: **guard bits**
- How many?
- For rounding toward zero (truncation): f fractional bits
- For rounding to nearest: one additional bit is required ($f+1$ fractional bits).

  For unbiased rounding to even: necessary to know when the rest of the bits are all zero
- For rounding toward infinity: necessary to know when all the bits to be discarded are zero

EFFECTIVE ADDITION/SUBTRACTION CASES

1. Effective addition:

- Result either normalized or produces an overflow
- Normalization: a 1-bit right shift (if overflow); no left shift required
- $\Rightarrow f+1$ fractional bits of the result required (R)
- Determine whether all the discarded bits are zero: *sticky bit* T, corresponds to the OR of the discarded bits

```
           1.0101110
           0.00010101010
ADD        -------------
           1.01110001   T=OR(010)=1
```

cont.

2. Effective subtraction. Two sub-cases:

(a) The difference of exponents d is larger than 1.

- the smallest operand is aligned so that there are more than one leading zeros
- the result is either normalized or, if not normalized, has only one leading zero
- the normalization is performed by a left shift of one position, in addition to the bit for rounding to nearest, another bit is required in the result of the addition.

$\Rightarrow f + 2$ fractional bits of result required

- During the subtraction, a borrow produced when sticky $= 1$

$\Rightarrow f + 3$ bits required in subtraction (GRT)

cont.

Example: After alignment

```
         1.0000011
         0.000011011001
 SUB     ----------------
```

```
During alignment compute T=OR(001)=1 resulting in
```

```
         1.0000011
         0.0000110111
   SUB   -------------
         0.1111100001
   NORM  1.1111000010
```

cont.

(b) The difference of exponents is either 0 or 1.

- Result might have more than one leading zeros
- Left shift of up to m positions required
- Since alignment shift only of zero or one position, at most one non-zero bit is shifted in during the normalization

$\Rightarrow$ only one additional bit required

```
       1.0000011
       0.11111001
SUB    ----------
       0.00001101
NORM   1.10100000
```

SUMMARY OF GUARD BITS

- in all cases three additional bits sufficient:
  guard (G), round(R), and sticky (T)
- After normalization guard bits labeled as follows:

```
          LGRT
1.xxxxxxxxxxxx
  ----f----
```

- During normalization sticky bit recomputed (OR of the previous T and the previous R)

ROUND TO NEAREST

- Round up (add rnd to position L)
  - If $G=1$ and R and T are not both zero, $rnd = G(R+T)$
  - If $G=1$ and $R=T=0$ then $rnd = G(R+T)'L$ – tie case

Combining both cases,

$$rnd = G(R + T) + G(R + T)'L = G(L + R + T)$$

```
L 1 1 0 1 1 1     G=1, R=1, T = 1 -> rnd = 1

L 1 0 0 0 0 0     G=1, R=0, T=1  -> rnd = 1 (tie case)

L 0 x x x x x     G=0   rnd = 0
```

DIRECTED ROUNDINGS

- Round toward zero: after normalization, truncate at bit L
- Round toward infinity:

  Positive infinity

$$rnd = sgn'(G + R + T)$$

  Negative infinity

$$rnd = sgn(G + R + T)$$

EXCEPTIONS AND SPECIAL VALUES

- Overflow:
  - detected by an exponent $E \geq 255$
  - set overflow flag, set result to ± infinity
- Underflow:
  - detected when during the left shift the exponent $E = 1$ and the significand not normalized
  - set underflow flag, set result exponent to $E = 0$
  - fraction left unnormalized (denormal, gradual underflow)
- Zero: the significand of the result of addition is 0

  The result is $E = 0$ and $F = 0$
- Inexact:
  - detected before rounding: the result is inexact if $G + R + T = 1$
  - set inexact flag
- NAN: if one (or both) operand is a NAN, the result set to NAN.

DENORMAL AND ZERO OPERANDS AND/OR RESULT

- Operand(s):
  - Operand a denormal number ($E = 0$ and $F \neq 0$): no hidden 1
  - Set operand of addition to $E = 1$ and $0.F$
- Zero operand ($E = 0$ and $F = 0$): treated as a denormal number
- Result:
  - detected during left shift: partially updated exponent $E = 1$ and significand not normalized
  - If resulting significand is not 0 then it is a denormal,
    if it is 0 then the result is zero
    exponent set to $E = 0$

CRITICAL PATH

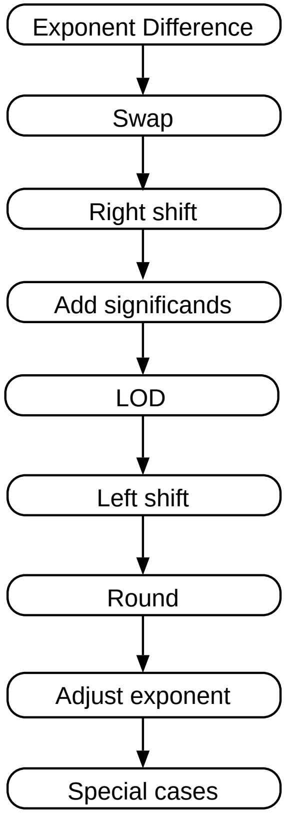
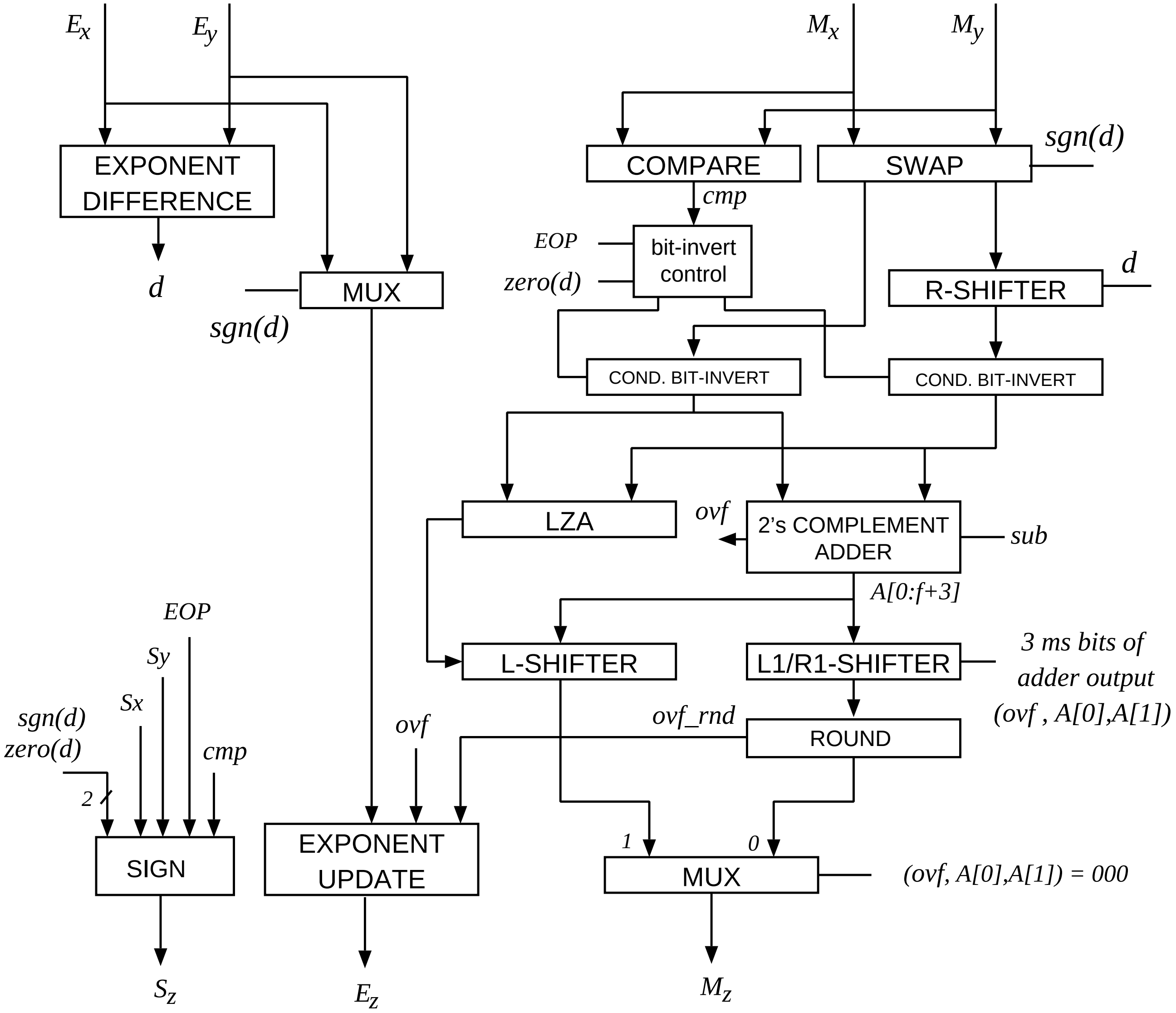


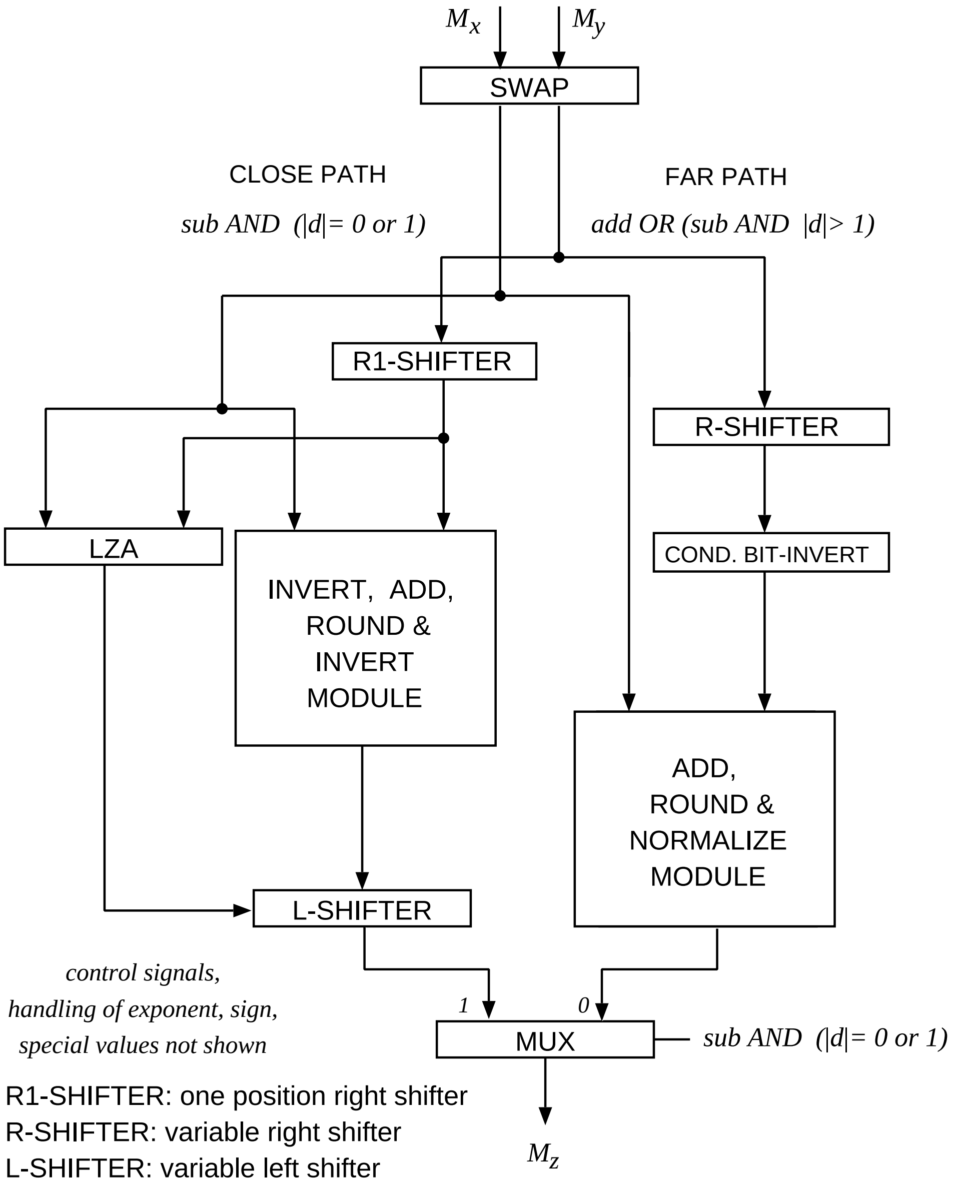
Figure 8.6: FLPT ADDITION: Critical Path.



(handling of special cases not shown)

EOP: effective operation
R-SHIFTER: variable right shifter
L-SHIFTER: variable left shifter
L1/R1-SHIFTER: one position left/right shifter

Figure 8.7: IMPROVED SINGLE-PATH FLOATING-POINT ADDITION.



control signals, handling of exponent, sign, special values not shown

R1-SHIFTER: one position right shifter
R-SHIFTER: variable right shifter
L-SHIFTER: variable left shifter

Figure 8.8: DOUBLE-PATH IMPLEMENTATION OF FLOATING-POINT ADDITION.

DEPENDENCE GRAPH FOR DOUBLE-PATH SCHEME

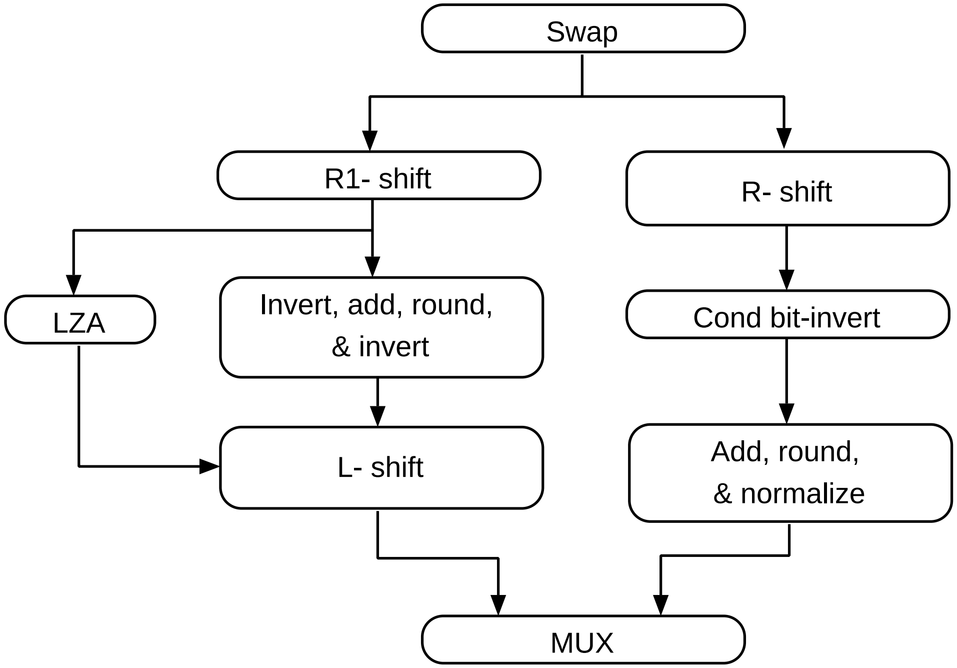
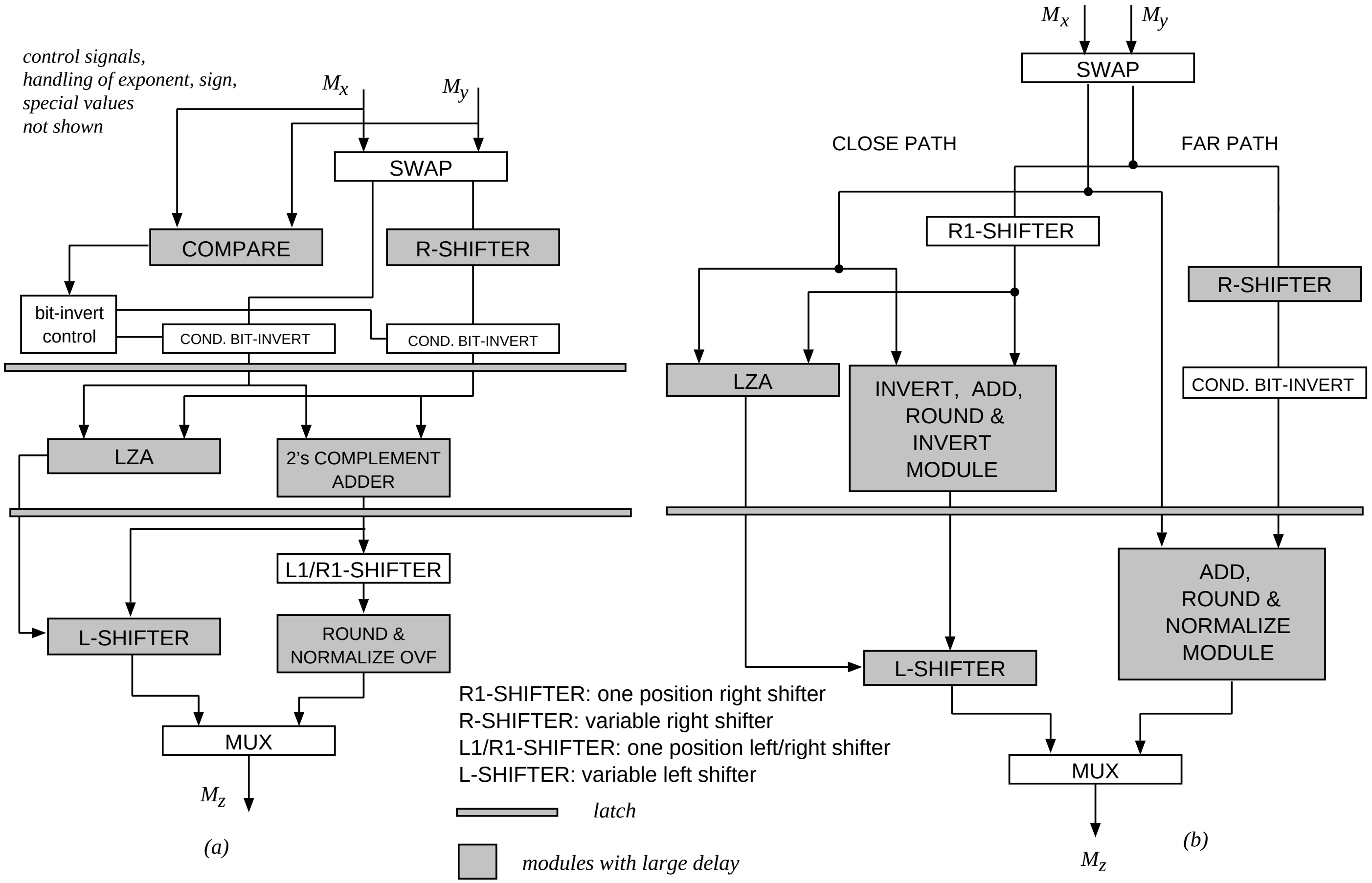


Figure 8.9: Dependence graph for double-path scheme.

PIPELINED IMPLEMENTATION



R1-SHIFTER: one position right shifter
R-SHIFTER: variable right shifter
L1/R1-SHIFTER: one position left/right shifter
L-SHIFTER: variable left shifter

▭ *latch*

▨ *modules with large delay*

Figure 8.10: PIPELINED IMPLEMENTATIONS: (a) SINGLE-PATH SCHEME. (b) DOUBLE-PATH SCHEME.

FLPT MULTIPLICATION

- x and y - normalized operands represented by (S_x, M_x, E_x) and (S_y, M_y, E_y)

1. Multiply significands, add exponents, and determine sign

$$\begin{aligned} M_z^* &= M_x^* \times M_y^* \\ E_z &= E_x + E_y \end{aligned}$$

2. Normalize M_z^* and update exponent
3. Round
4. Determine exception flags and special values

BASIC IMPLEMENTATION

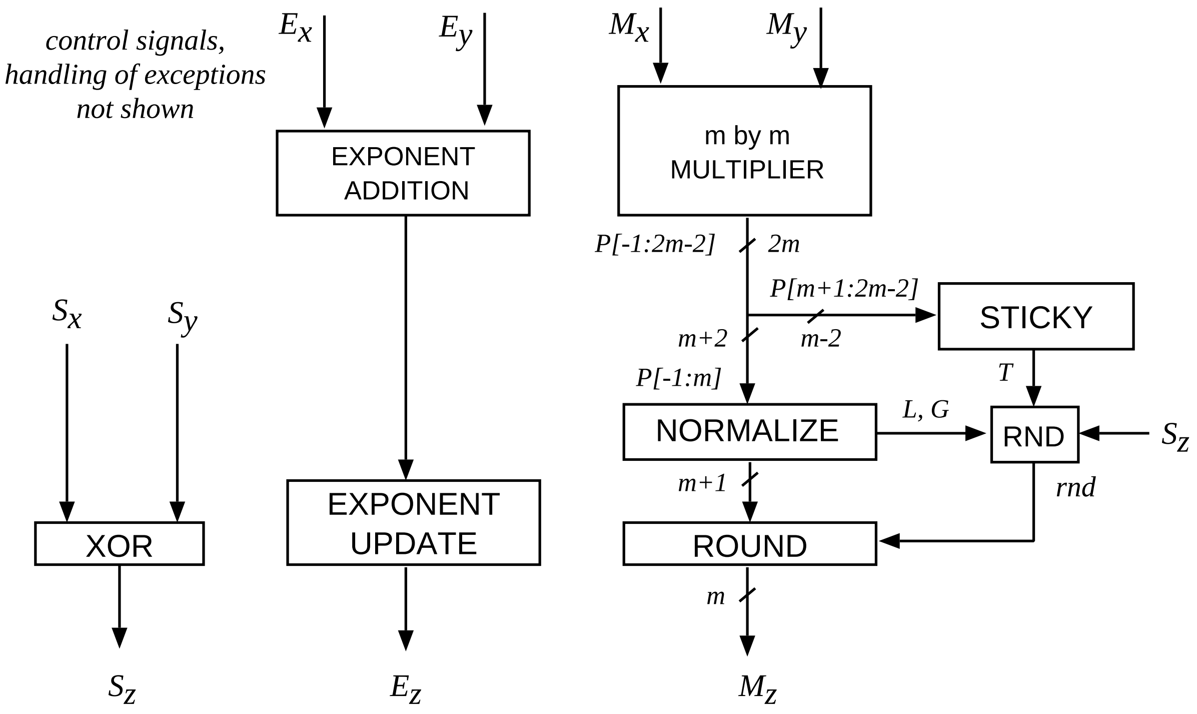


Figure 8.11: BASIC IMPLEMENTATION OF FLOATING-POINT MULTIPLICATION.

COMMENTS ON IMPLEMENTATION

1. Multiplication of magnitudes

   - produces magnitude P of $2m$ bits - only m bits in result: one guard bit and the sticky bit

```
Output of multiplier module P:
Bit position: (-1)0.123...(m-2)(m-1) m (m+1)...(2m-2)
```

2. Exponent of result

$$E_z = E_x + E_y - B$$

3. Sign of result

$$S_z = S_x \oplus S_y$$

cont.

4. Normalization: $1 \leq M_x, M_y < 2$, the result in range $[1, 4)$

```
Output of multiplier module P:

Bit position: (-1)0.123...(m-2)(m-1) m (m+1)...(2m-2)

If P[-1]=0, P is normalized:

   L = P[m-1], G = P[m], T = OR(P[m+1],...,P[2m-2])

If P[-1] = 1, normalize P by shifting right one position

   L = P[m-2], G = P[m-1], T = OR(P[m],...,P[2m-2])
```

cont.

5. Rounding: four rounding modes with guard bit (G) and sticky bit (T)

- Round to nearest

$$rnd = G(T) + G(T)'L = G(T + L)$$

  with G and T the two bits following L AFTER the normalization.

- Round toward zero
  Result after normalization truncated at bit L

- Round toward infinity
  positive infinity add

$$rnd = sgn'(G + T)$$

  negative infinity

$$rnd = sgn(G + T)$$

EXCEPTIONS AND SPECIAL VALUES, ETC.

- Overflow: exponent too large;
  detected after exponent update;
  overflow flag set; result value is ±infinity
- Underflow: resulting exponent too small;
  underflow flag set; exponent set to $E = 0$
  significand shifted right to represent a denormal
- Zero: when one of the operands has value 0 and the other is not ± infinity;
  - zero result set

EXCEPTIONS, SPECIAL VALUES, ETC. (cont.)

- Inexact: result inexact if, after normalization, $G + T = 1$
- NAN: result NAN if one (or both) of the operands is a NAN or if one of the operands is a 0 and the other ± infinity
- Denormals: result denormal if one or both operands are denormal;

  left shift necessary;

  if exponent underflow, right shift (gradual underflow); set E=0

ALTERNATIVE IMPLEMENTATION

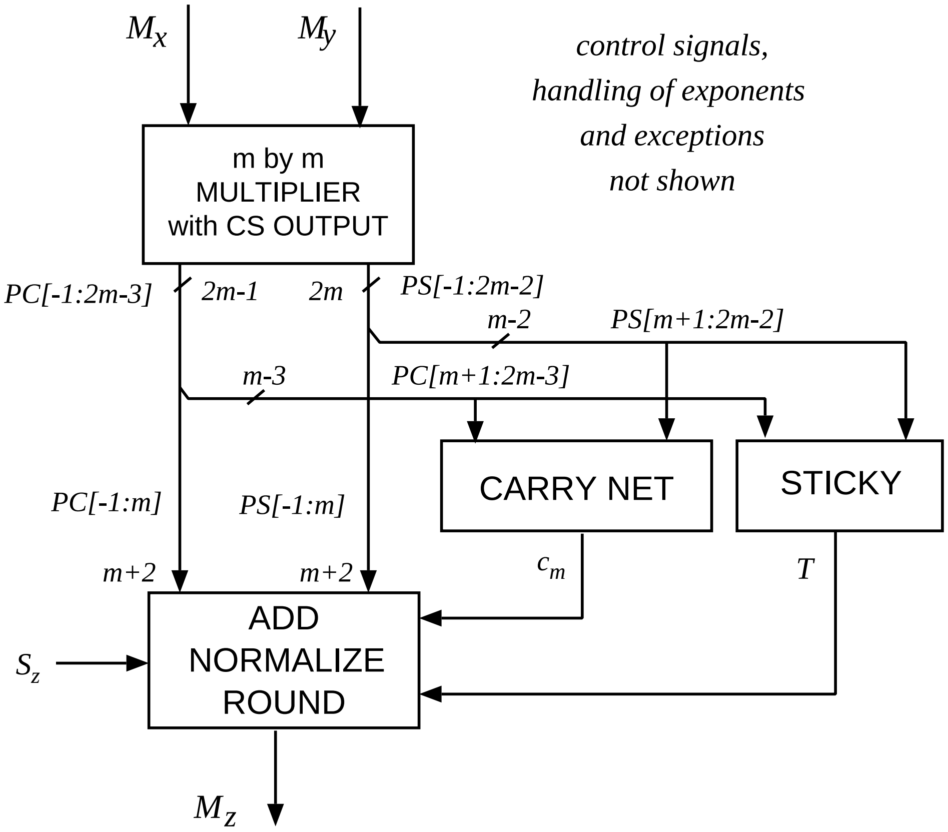


Figure 8.12: ALTERNATIVE IMPLEMENTATION.

REDUCING THE LATENCY

- Compute MS half (+ guard bit) in conventional form using c_m;
  c_m in the critical path
- Determine sticky from the operands;
  needs detector of trailing zeros, adder, and comparator

- Determine sticky from CS form of the LS half

```
S       s s s s s s s s
C       c c c c c c c c
-1      1 1 1 1 1 1 1 1
        ----------------
        z z z z z z z z
        t t t t t t t
```

$$\begin{aligned} z_i &= (s_i \oplus c_i)' \\ t_i &= s_{i+1} + c_{i+1} \end{aligned} \qquad (8.3)$$

Compute

$$w_i = z_i \oplus t_i \qquad (8.4)$$

Sticky bit is

$$T = NAND(w_i) \qquad (8.5)$$

CARRY FROM LS PART

```
Product P[-1:2m-2]

-------(m+2)----- -----(m-2)-----
xxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx xxxxxxxxxxxxxx
                c_m

    c_m is the carry produced by
    the least-significant m-2 bits of product P
    and added in position m.
```

Figure 8.13: ADDING CARRY FROM THE LEAST SIGNIFICANT HALF.

ROUNDING POSITION

```
Bit position: (-1)(0).123...(m-2)(m-1) m
                0  1 .xxx...  x    x   x
                                       c_m
                                       1
                   (a)

Bit position: (-1)(0).123...(m-2)(m-1) m
                1  x .xxx...  x    x   x
                                   1   c_m
                   (b)
```

Figure 8.14: ROUNDING POSITION: (a) NORMALIZED PRODUCT. (b) UNNORMALIZED PRODUCT.

ADDING CARRY AND ROUNDING

- Product in CS form - normalized?
- Combine final addition and rounding. Select the correct result.
- $PM = PS + PC$ – the MS of the product up to position m
- Compute

$$P0 = PM + (c_m + 1) \times 2^{-m}$$

and

$$P1 = PM + (c_m + 2) \times 2^{-m}$$

and then select

$$P = \begin{cases} P0 & \text{if } P0[-1] = 0 \\ 2^{-1}P1 & \text{if } P0[-1] = 1 \end{cases}$$

```
     (-1) 0. 1 2 3 ... (m-2)(m-1)(m)
PS    x   x  x x x       x    x   x
PC    x   x  x x x       x    x   x
                              c_m c'_m <=> (c_m+1)2^(-m)
    ---------------------------------
PS*   x   x  x x x       x    x   x
PC*   x   x  x x x       x    x

Get P0 and P1 = P0 + 2^{-m}:
PS*   x   x  x x x       x    x   x
PC*   x   x  x x x       x    x   0
   -----------------------------------
P0   ovf  x  x x x       x    x   x
P1    x   x  x x x       x    x   x

After selection:
P         1. x x x ...   x    L
```

Figure 8.15: ADDING CARRY c_m AND ROUNDING.

IMPLEMENTATION

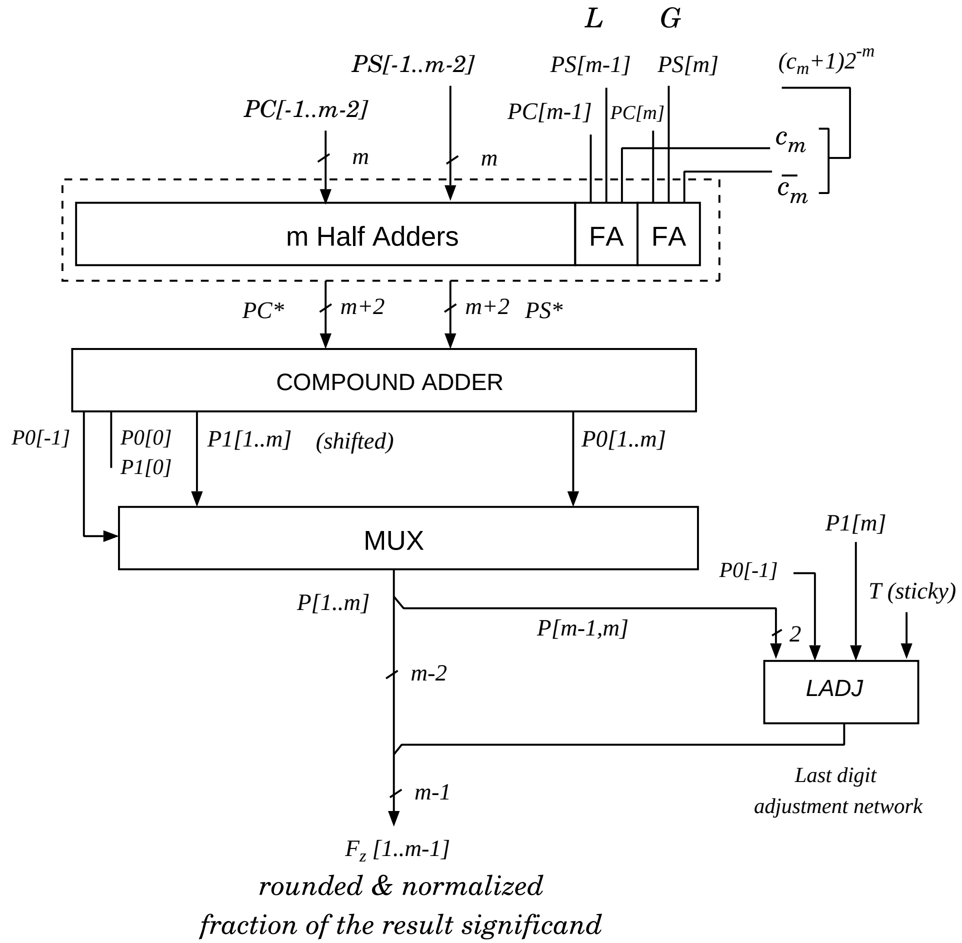


Figure 8.16: ADDING CARRY, NORMALIZATION, AND ROUNDING IMPLEMENTATION

IMPLEMENTATION (cont.)

1. A row of HAs and FAs to add $(c_m+1)2^{-m}$ to $PS[-1,m]$ and $PC[-1,m]$.
2. A compound adder that produces the sum $P0$ and the sum plus 1 ($P1$).
3. A multiplexer which selects $P0$ or the normalized (shifted) $P1$ depending whether $P0$ does not overflow or overflows
4. A module $LADJ$ which determines the least-significant bit of the significand.

   sticky bit update:

$$T^* = T + P1[m] \cdot P0[-1] \quad \text{update sticky bit}$$

adjustment of the least-significant bit

$$L = P[m-1](P[m] + T^*)$$

REMOVING c_m FROM CRITICAL PATH

carry+sum in pos. m	range of Σ before pre-add	range of c_{m-1}	pre-add 1?	range of Σ after pre-add	range of c_{m-1}
0	[1,3]	[0,1]	NO	[1,3]	[0,1]
1	[2,4]	[1,2]	YES	[0,2]	[0,1]
2	[3,5]	[1,2]	YES	[1,3]	[0,1]

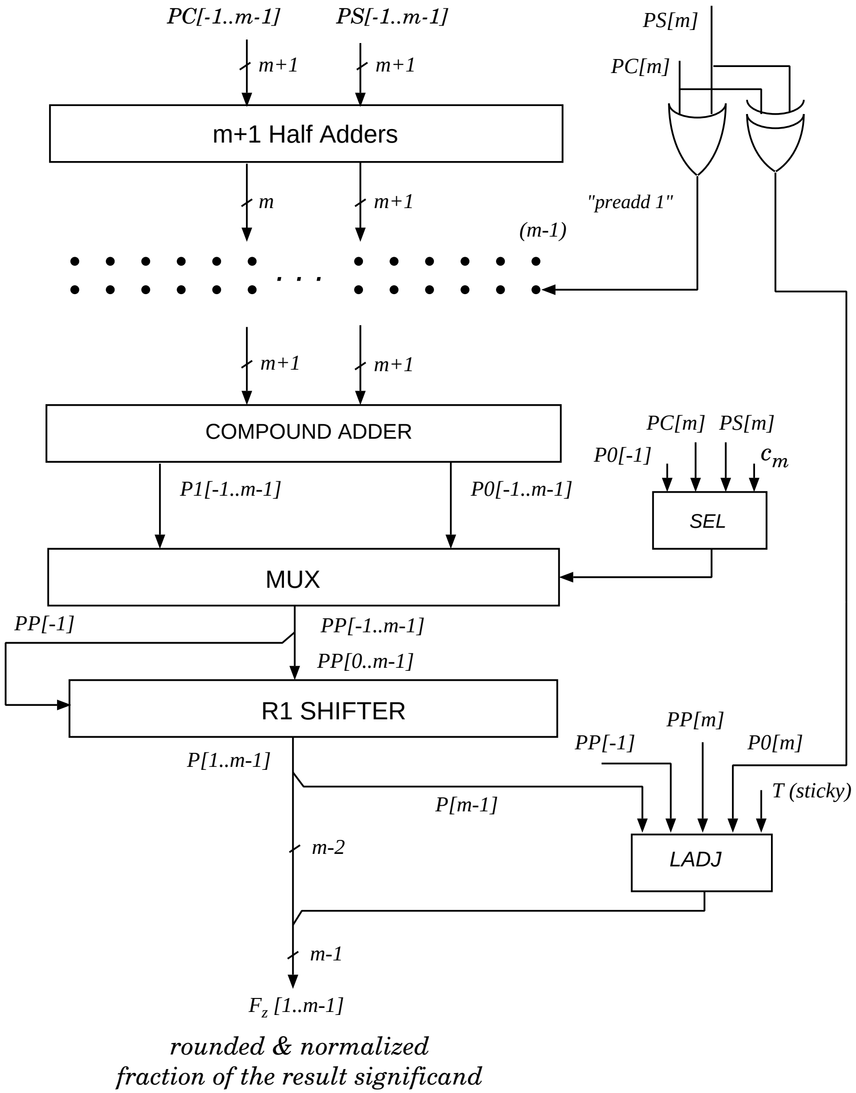
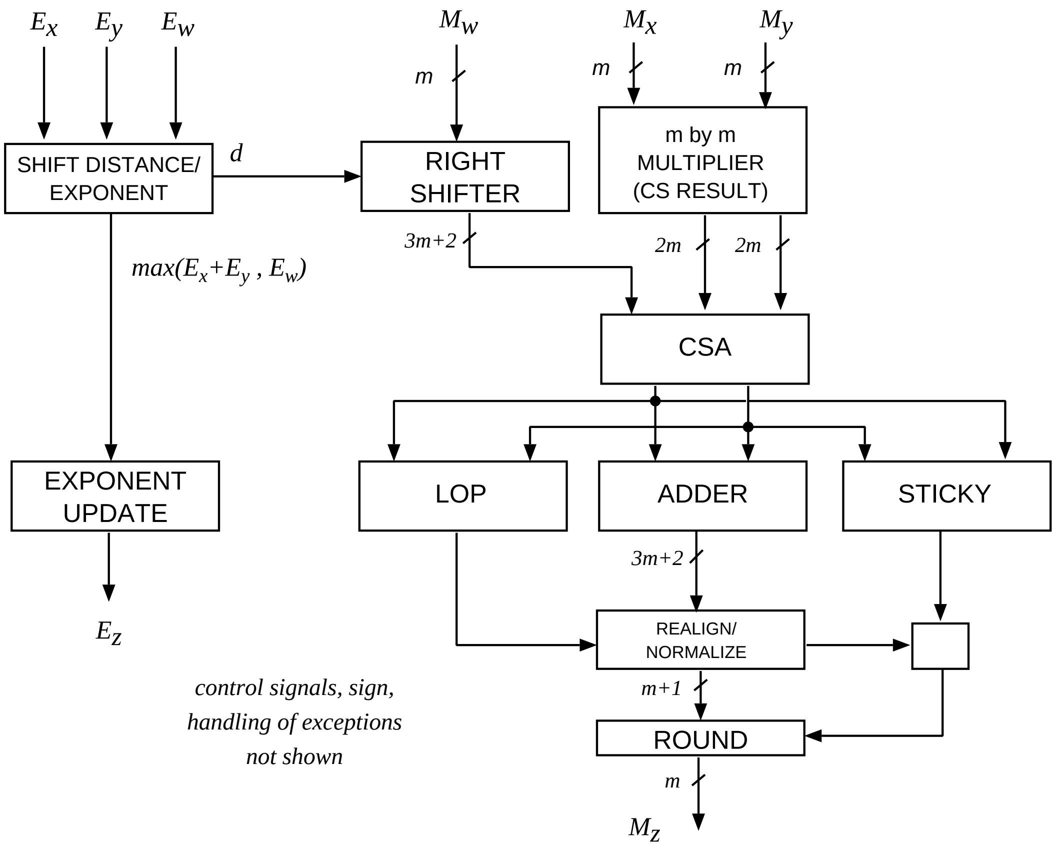


Figure 8.17: Adding carry, normalization, and rounding implementation with carry out of critical path.

MULTIPLY-ADD FUSED (MAF)



control signals, sign,
handling of exceptions
not shown

Figure 8.18: BASIC IMPLEMENTATION OF MAF OPERATION.

MAF (cont.)

```
               ------ m ------  -------- 2m ---------
Product x*y:                  00xx.xxxxx...xxxxxxxxxx
Addend:        1.xxxxxxxxxxxxx
                |--- m-1+4 -------|
                         (a)


                                -------- 2m ---------
Product x*y:                    xx.xxxxx...xxxxxxxxxx
Addend:                                             01xxxxxxxxxxxxx
Shift distance:                   |--- 2m-2+1 -------|

                          (b)
```

Figure 8.19: Position of addends using bidirectional shift: (a) Maximum left shift. (b) Maximum right shift.

AVOIDING BIDIRECTIONAL SHIFTER

- Position addend M_w $m+3$ bits to the left of the product
- shift right by the distance

$$d = E_x + E_y - E_w + m + 3 \tag{8.6}$$

- for biased exponent performed as

$$d = E_x^B + E_y^B - E_w^B - B + m + 3 \tag{8.7}$$

- No shift performed for $d \leq 0$ and the maximum shift is $3m+1$

```
Initial position:
               ------ m ------  -------- 2m ------------
Product x*y:                  00xxxxxxxx....xxxxxxxxxxxx
Addend:        1.xxxxxxxxxxxxx
                |--- m-1+4 -------|
                                |--- sticky ------------|
                                     region
                         (a)

Alignment when Exy = Ew:
               ------ m ------  -------- 2m --------------
Product x*y:                  00xx.xxxxx....xxxxxxxxxxxxxxx
Addend:                          1.xxxxxxxxxxxxx
                                                   |-sticky|
Shift distance: |--- m-1+4 -------|

                         (b)
```

```
Alignment when Exy - Ew = k:
               ------ m ------  -------- 2m --------
Product x*y:                  00xx.xxxxx....xxxxxxxx
Addend:                                       1xxxxxxxxxxxxx
Shift distance: |----- m+3 -------|----- k ----|

                         (c)

Alignment when Exy - Ew >= 2m-1:
               ------ m ------  -------- 2m --------
Product x*y:                  00xx.xxxxx....xxxxxxxx
Addend:                                            01xxxxxxxxxxxxx
Shift distance: |----- m+3 -------|----- 2m-1 -------|

                         (d)
```

Figure 8.20: Alignment with right shifter.

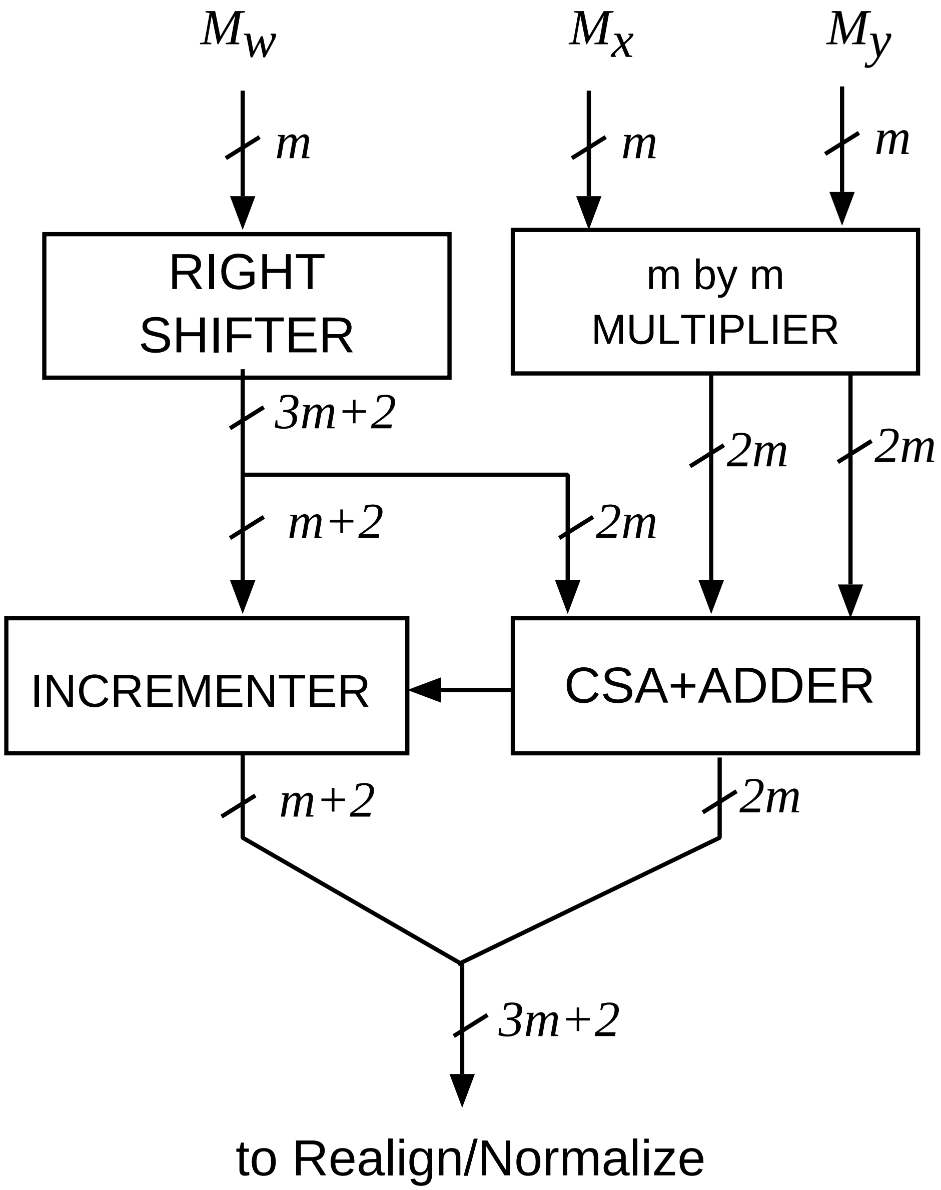


Figure 8.21: Implementation of MAF adder.

LEFT SHIFTING OF ADDER OUTPUT

```
Adder output       |----- m+2 -----|--------- 2m ----------|

Before shift        000000000000000001.xxxxxxxxxxxxxxxxxxx

After shift         1.xxxxxxxxxxxxLGRT
```

Figure 8.22: Left shifting of the adder output.

PIPELINED MAF

- MAF unit usually pipelined.
- Three-stage pipeline:
  - Stage 1 implements the multiplication, alignment and 3-2 carry-save addition;
  - Stage 2 performs 2-1 addition and predicts the leading one in the sum;
  - Stage 3 performs normalization and rounding

FLOATING-POINT DIVISION

- Operands: x and d represented by $(M_x^*,\ E_x)$ and $(M_d^*,\ E_d)$, with M_x^* and M_d^* signed and normalized. The result

$$q = x/d \tag{8.8}$$

represented by $(M_q^*,\ E_q)$, with M_q also signed and normalized.

- The high-level description of the floating-point division algorithm

1. Divide significands and subtract exponents

$$\begin{aligned} M_q^* &= M_x^*/M_d^* \\ E_q &= E_x - E_d \end{aligned} \tag{8.9}$$

2. Normalize M_q^* and update exponent
3. Round
4. Determine exception flags and special values

IMPLEMENTATION

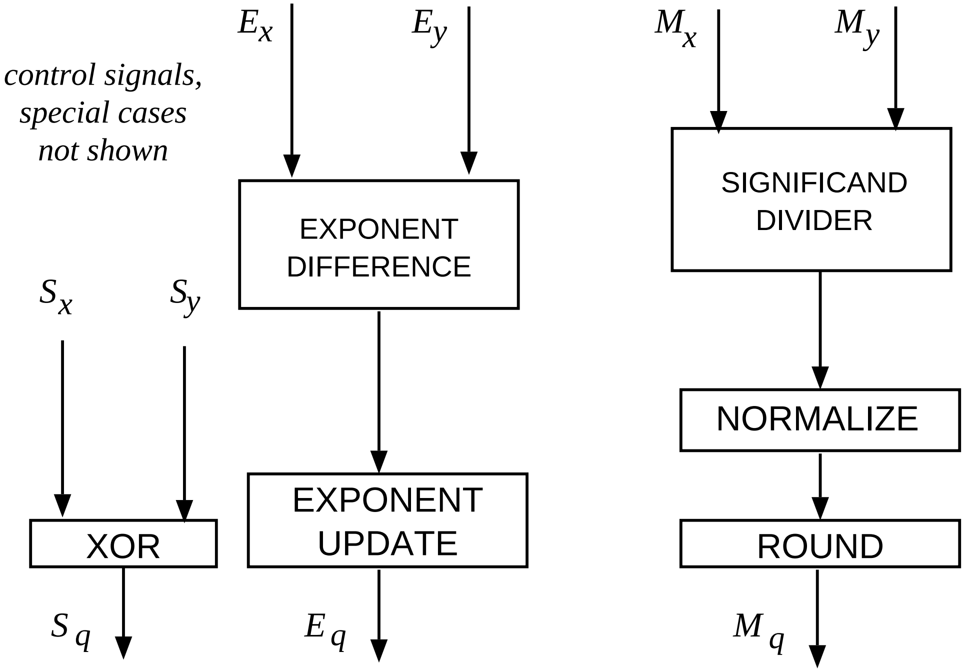


Figure 8.23: Basic implementation of floating-point division.

DIGIT-SERIAL ARITHMETIC

- Modes of operation:LSDF and MSDF
- Algorithm and implementation models
- LSDF arithmetic
- MSDF: Online arithmetic

TIMING PARAMETERS

- radix-r number system: conventional and redundant
- Serial signal: a numerical input or output with one digit per clock cycle
- For an n digit signal, the clock cycles are numbered from 1 to n
- Timing characteristics of a serial operation determined by two parameters:
  - *initial delay* δ: additional number of operand digits required to determine the first result digit
  - *execution time* T_n; the time between the first digit of the operand and the last digit of the result

$$T_n = \delta + n + 1$$

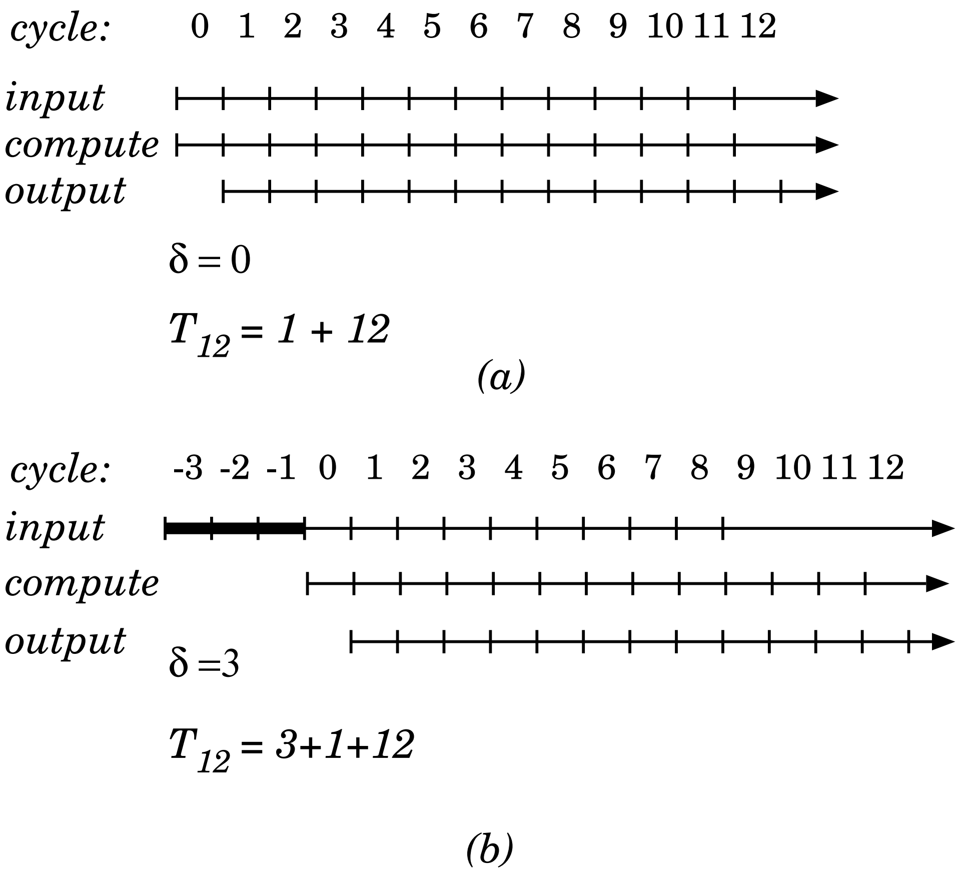


Figure 9.1: Timing characteristics of serial operation with $n = 12$. (a) With $\delta = 0$. (b) With $\delta = 3$.

LSDF AND MSDF

1. *Least-significant digit first* (LSDF) mode (right-to-left mode)

$$x = \sum_{i=0}^{n-1} x_i r^i$$

2. *Most-significant digit first* (MSDF) mode (left-to-right mode)

   also known as *online arithmetic*

   initial delay called *online delay*

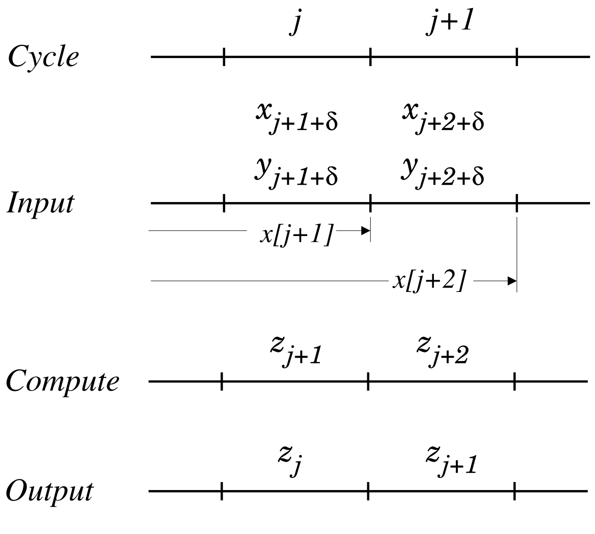
$$x = \sum_{i=1}^{n} x_i r^{-i}$$

ALGORITHM MODEL

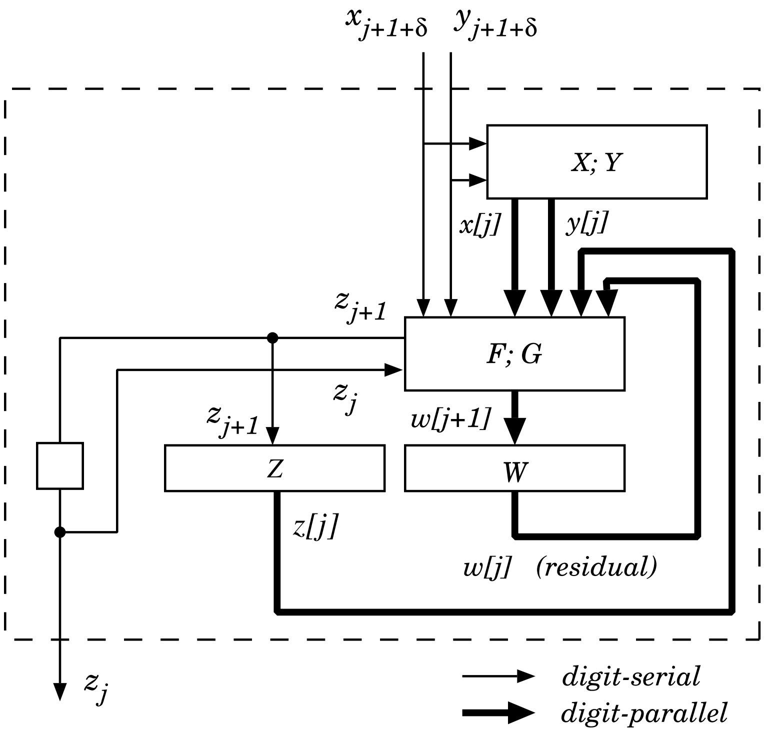
- Operands x and y, result z: n radix-r digits
- In cycle j the result digit z_{j+1} is computed
- Cycles labeled $-\delta, \ldots, 0, 1, \ldots, n$
- In cycle j receive the operand digits $x_{j+1+\delta}$ and $y_{j+1+\delta}$, and output z_j
- $x[j]$, $y[j]$ and $z[j]$ are the numerical values of the corresponding signals when their representation consists of the first $j + \delta$ digits for the operands, and j digits for the result.

In iteration j

$$\begin{aligned}
x[j+1] &= (x[j], x_{j+1+\delta}) \\
y[j+1] &= (y[j], y_{j+1+\delta}) \\
z_{j+1} &= F(w[j], x[j], x_{j+1+\delta}, y[j], y_{j+1+\delta}, z[j]) \\
z[j+1] &= (z[j,], z_{j+1}) \\
w[j+1] &= G(w[j], x[j], x_{j+1+\delta}, y[j], y_{j+1+\delta}, z[j], z_{j+1})
\end{aligned}$$


j
j+1
Cycle
$x_{j+1+\delta}$
$x_{j+2+\delta}$
$y_{j+1+\delta}$
$y_{j+2+\delta}$
Input
x[j+1]
x[j+2]
z_{j+1}
z_{j+2}
Compute
z_j
z_{j+1}
Output

(a)


$x_{j+1+\delta}$
$y_{j+1+\delta}$
X; Y
x[j]
y[j]
z_{j+1}
F; G
z_j
z_{j+1}
w[j+1]
Z
W
z[j]
w[j] (residual)
z_j
digit-serial
digit-parallel

(b)

Table 9.1: Initial delay (δ)

Operation	LSDF	MSDF
Addition	0	2 ($r = 2$)
		1 ($r \geq 4$)
Multiplication	0	3 ($r = 2$)
		2 ($r = 4$)
(only MS half)	n	
Division	$2n$	4
Square root	$2n$	4
Max/min	n	0

```
a) LSDF mode

n-digit addition:
  Cycle:     0 1 2 . . .
            LSD             MSD
  Inputs:    x x x x x x x x x
  Output:      x x x x x x x x x

n by n --> 2n multiplication:

            LSD             MSD
  Inputs:    x x x x x x x x x
  Output:      x x x x x x x x x x x x x x x x x x
                                 -----------------
                                      MS half

b) MSDF mode

  Cycle:    -2 -1 0 1 2 . . .
n-digit operation:
            MSD                   LSD
  Inputs:    x  x x x x  x  x  x  x
  Output:           x x  x  x  x  x  x  x  x
             ---
          online delay = 2
```

Figure 9.3: LSDF and MSDF modes.

COMPOSITE ALGORITHM: EXAMPLE

- Givens method for matrix triangularization
- Rotation factors:

$$c = \frac{x}{\sqrt{x^2 + y^2}} \quad s = \frac{y}{\sqrt{x^2 + y^2}}$$

- The online delay of the network

$$\Delta_{rot} = \delta 1 + \delta 2 + \delta 3 + \delta 4 = 13$$

- Execution time (latency): $D_{rot} = \Delta_{rot} + n + 4$

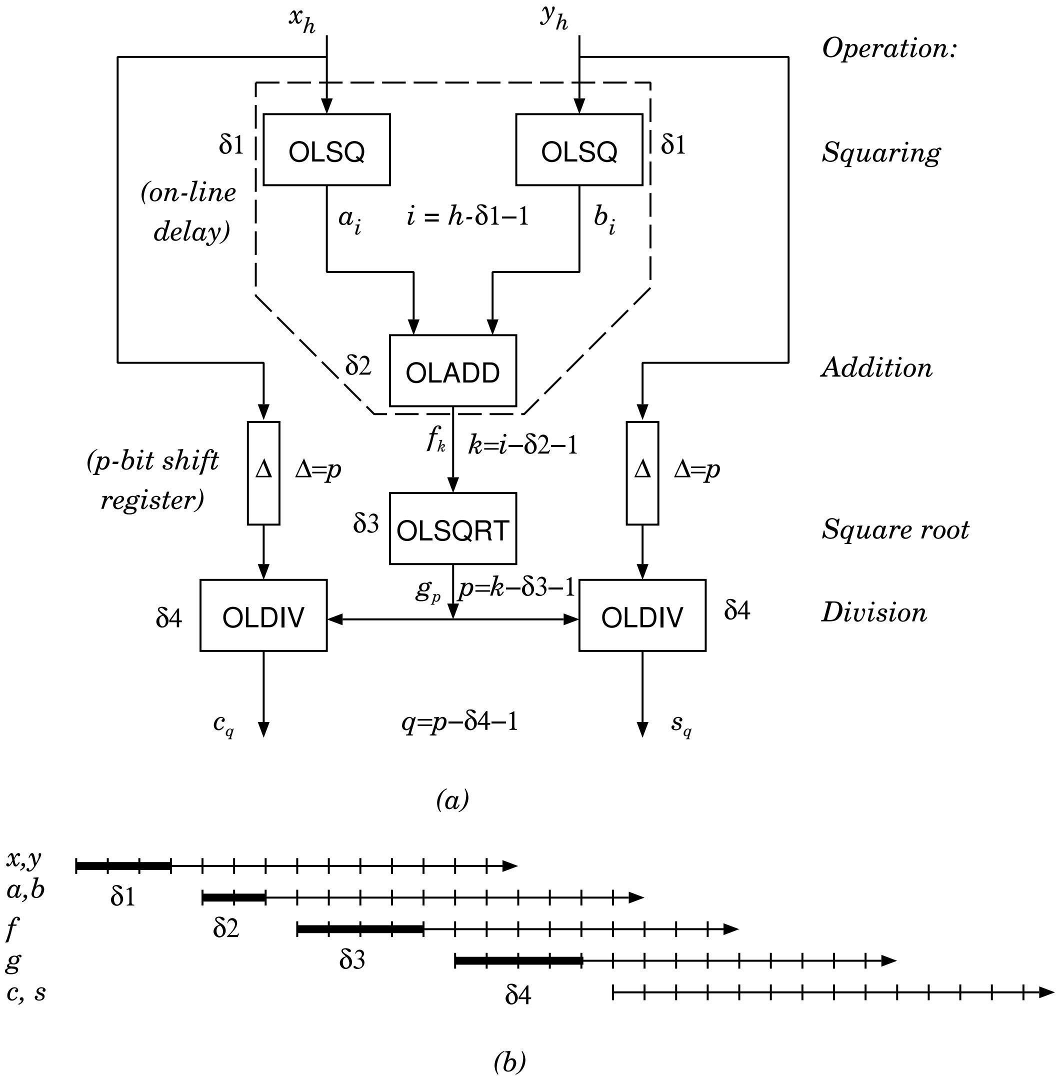


Figure 9.4: Online computation of rotation factors: (a) Network. (b) Timing diagram.

LSDF ARITHMETIC: ADDITION/SUBTRACTION

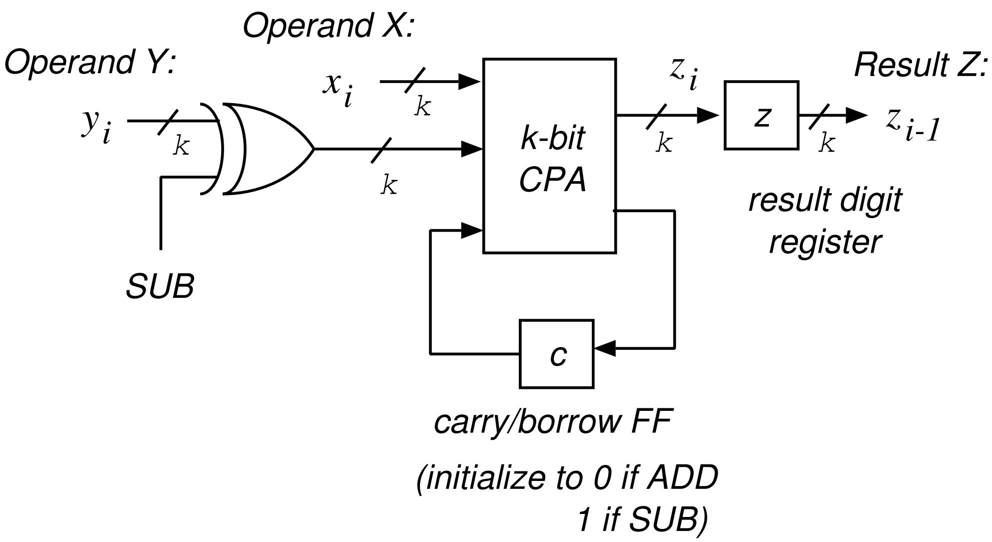
- The cycle delay

$$t_{LSDFadd-k} = t_{CPA(k)} + t_{FF}$$

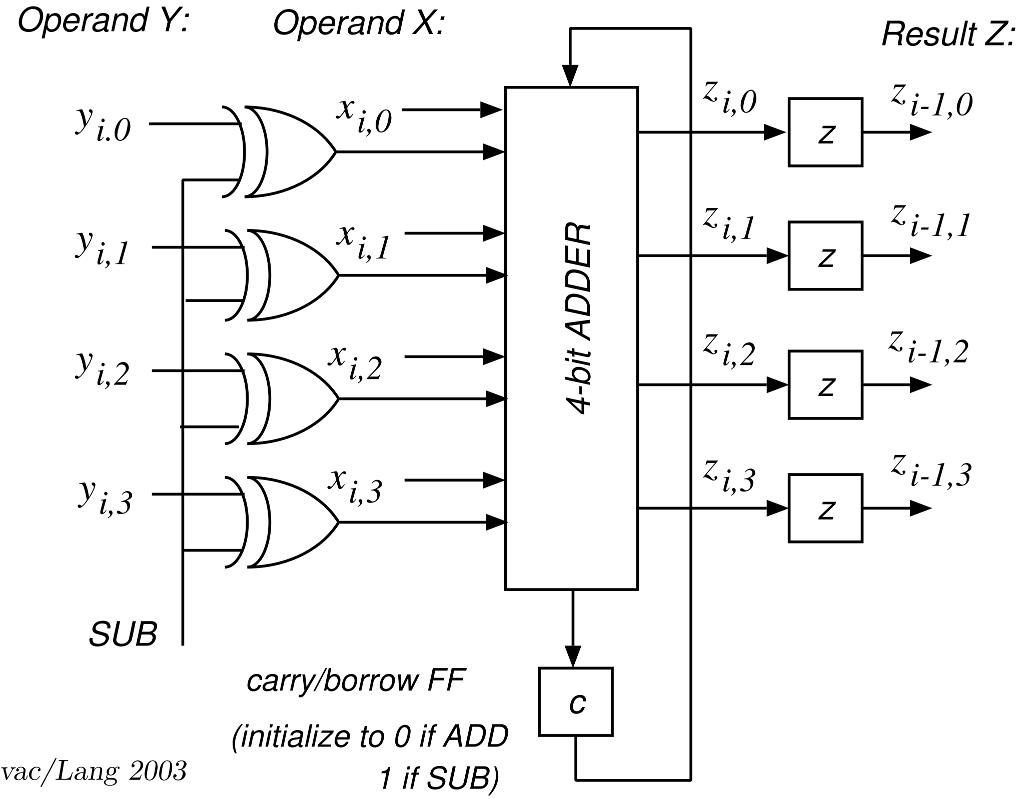
- The total time for n-bit addition

$$T_{LSDFadd-n} = (\frac{n}{k} + 1)t_{LSDFadd-k}$$

- The cost: one k-bit CPA, one flip-flop, and one k-bit output register


Operand X:
Operand Y:
x_i
k
y_i
k
k
SUB
k-bit
CPA
z_i
k
z
k
Result Z:
z_{i-1}
result digit
register
c
carry/borrow FF
(initialize to 0 if ADD
1 if SUB)

(a)


Operand Y:
Operand X:
Result Z:
$y_{i.0}$
$x_{i,0}$
$y_{i,1}$
$x_{i,1}$
$y_{i,2}$
$x_{i,2}$
$y_{i,3}$
$x_{i,3}$
4-bit ADDER
$z_{i,0}$
$z_{i,1}$
$z_{i,2}$
$z_{i,3}$
z
z
z
z
$z_{i-1,0}$
$z_{i-1,1}$
$z_{i-1,2}$
$z_{i-1,3}$
SUB
carry/borrow FF
(initialize to 0 if ADD
1 if SUB)
c

LSDF MULTIPLICATION

- For radix-2 and 2's complement representation:

1. Serial-serial (LSDF-SS) multiplier, both operands used in digit-serial form.
2. Serial-parallel (LSDF-SP) multiplier, one operand first converted to parallel form

- Operation cannot be completed during the input of operands

SERIAL-SERIAL MULTIPLIER

- Define internal state (residual)

$$w[j] = 2^{-(j+1)}(x[j] \times y[j] - p[j])$$

where $x[j] = \Sigma_{i=0}^{j} x_i 2^i$ and similarly for $y[j]$ and $p[j]$.

- Both operands used in serial form; the recurrence is

$$\begin{aligned} w[j+1] &= 2^{-(j+2)}(x[j+1] \times y[j+1] - p[j+1]) \\ &= 2^{-(j+2)}((x[j] + x_{j+1}2^{j+1})(y[j] + y_{j+1}2^{j+1}) - (p[j] + p_{j+1}2^{j+1})) \\ &= 2^{-1}(w[j] + y[j+1]x_{j+1} + x[j]y_{j+1} - p_{j+1}) \end{aligned}$$

- This can be expressed as

$$v[j] = w[j] + y[j+1]x_{j+1} + x[j]y_{j+1}$$

and

$$\begin{aligned} w[j+1] &= \lfloor 2^{-1}v[j] \rfloor \\ p_{j+1} &= v[j] mod\ 2 \end{aligned}$$

	Position								
Cycle	8	7	6	5	4	3	2	1	0
0									y_0x_0
1								x_0y_1	
							y_1x_1	y_0x_1	
2						x_1y_2	x_0y_2		
					y_2x_2	y_1x_2	y_0x_2		
3				x_2y_3	x_1y_3	x_0y_3			
			y_3x_3	y_2x_3	y_1x_3	y_0x_3			
4		x_3y_4	x_2y_4	x_1y_4	x_0y_4				
	y_4x_4	y_3x_4	y_2x_4	y_1x_4	y_0x_4				

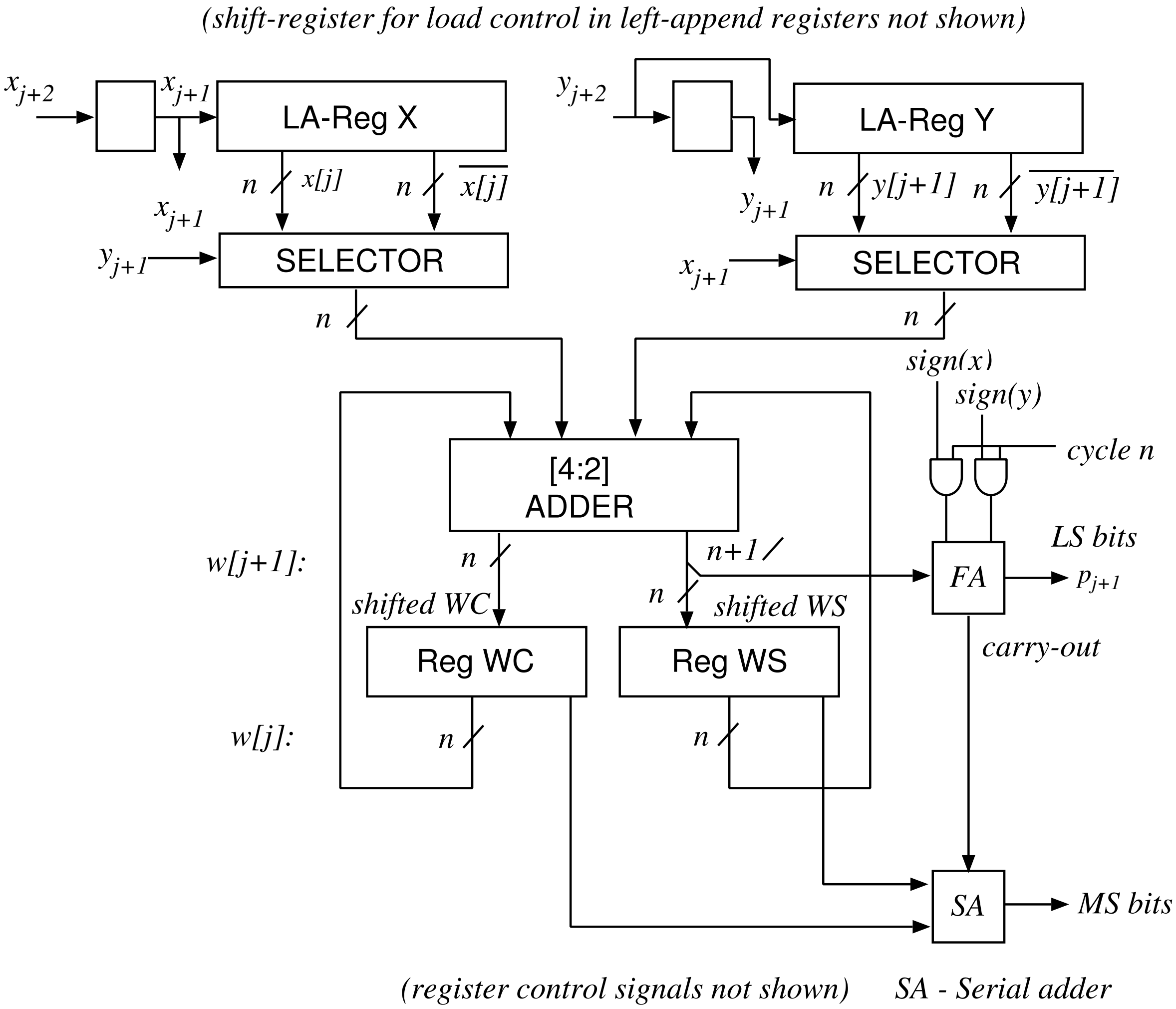


Figure 9.6: Serial-serial 2's complement radix-2 multiplier.

cont.

- The total execution time

$$T_{SSMULT} = 2nt_{cyc}$$

- The delay of the critical path in a cycle

$$t_{cyc} = t_{SEL} + t_{4-2CSA} + t_{FF}$$

- Cost: one n-bit [4:2] adder, 5 n-bit registers, and gates to form multiples

SERIAL-PARALLEL MULTIPLIER

- One of the operands is a constant,

One possibility: perform operation in $3n$ cycles

Phase 1: Serial input and conversion of one operand to parallel form;

Phase 2: Serial-parallel processing and output of the LS half of the product.

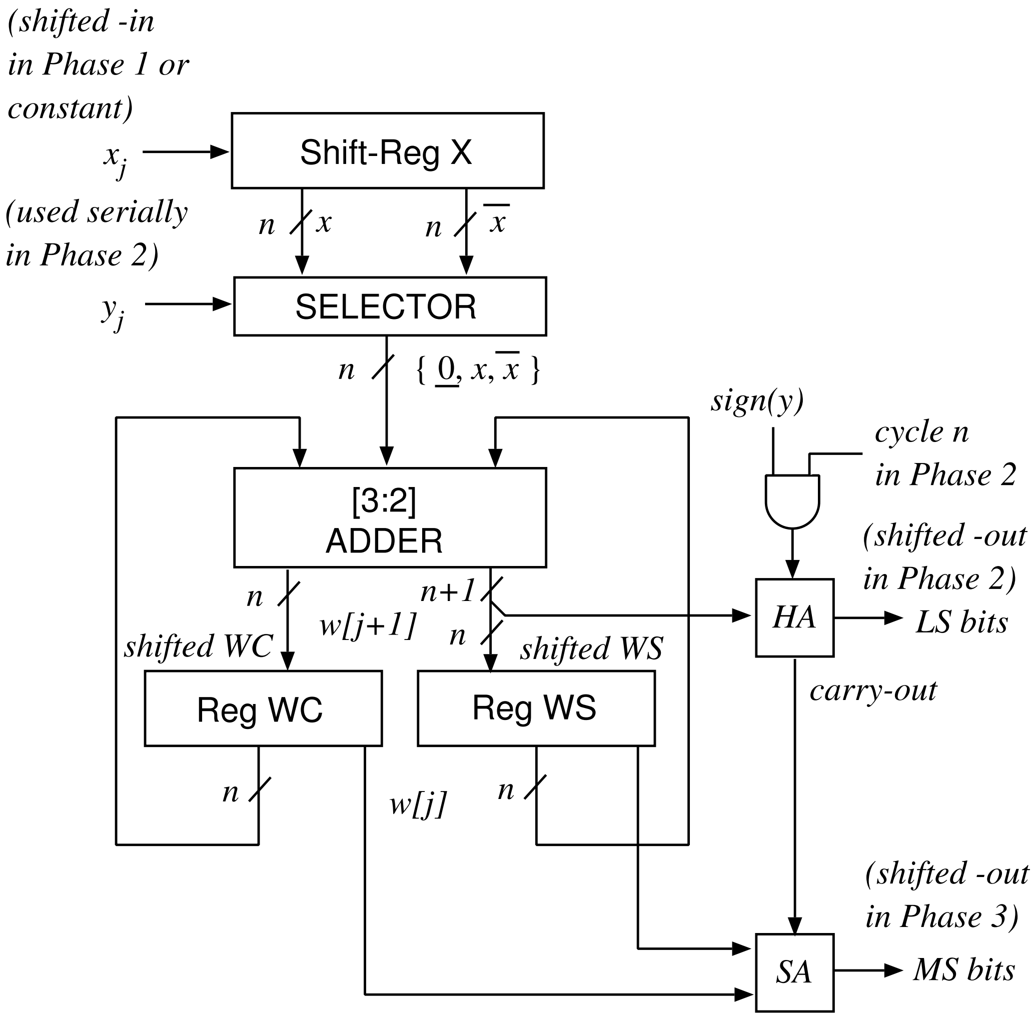
Phase 3: Serial output of the MS half of the product.

- The critical path in a cycle

$$t_{cyc} = t_{SEL} + t_{CSA} + t_{FF}$$

- The delay of the LSDF-SP multiplier

$$T_{SPrnd} = 3n \times t_{cyc}$$



(register control signals not shown)

SA - Serial adder

Phase 1: shift-in operand X (n cycles)

Phase 2: serial-parallel carry-save multiplication (n cycles)
shifted sum and carry bit-vectors loaded bit-parallel

Phase 3: MS bits obtained using bit-serial adder SA operating
on bits shifted out of WC and WS shift-registers (n cycles)

Figure 9.7: 3-phase serial multiplier.

MSDF: ONLINE ARITHMETIC

- Online arithmetic algorithms operate in a digit-serial MSDF mode
- To compute the first digit of the result, $\delta + 1$ digits of the input operands needed
- Thereafter, for each new digit of the operands, an extra digit of the result obtained
- The *online* delay δ typically a small integer, e.g., 1 to 4.

Cycle	-2	-1	0	1	2	$\cdots$
Input	x_1	x_2	x_3	x_4	x_5	$\cdots$
Compute			z_1	z_2	z_3	$\cdots$
Output				z_1	z_2	$\cdots$

$$\delta = 2$$

Figure 9.8: Timing in online arithmetic.

REDUNDANT REPRESENTATION

- The left-to-right mode of computation requires redundancy
- Both symmetric $\{-a, \ldots, a\}$ and asymmetric $\{b, \ldots, c\}$) digit-sets used in online arithmetic
- Over-redundant digit-sets also useful
- Examples of radix-4 redundant digit sets:
  {-1,0,1,2,3} (asymmetric, minimally-redundant),
  {-2,-1,0,1,2} (symmetric, minimally-redundant), and
  {-5,-4,-3,-2,-1,0,1,2,3,4,5} (symmetric, over-redundant)
- Heterogeneous representations to optimize the implementation
- Conversion to conventional representation: use on-the-fly conversion

ADDITION/SUBTRACTION

- The online addition/subtraction algorithm: the serialization of a redundant addition (carry-save or signed-digit)
- Radix $r > 2$

$$(t_{j+1}, w_{j+2}) = \begin{cases} (0, x_{j+2} + y_{j+2}) & \text{if } |x_{j+2} + y_{j+2}| \leq a - 1 \\ (1, x_{j+2} + y_{j+2} - r) & \text{if } x_{j+2} + y_{j+2} \geq a \\ (-1, x_{j+2} + y_{j+2} + r) & \text{if } x_{j+2} + y_{j+2} \leq -a \end{cases}$$

and

-

$$z_{j+1} = w_{j+1} + t_{j+1}$$

where $x_j,\ y_j,\ z_j \in \{-a, \ldots, a\}$.

EXAMPLE OF ONLINE ADDITION ($r = 4$, $a = 3$)

- Operands $x = (.12\bar{3}30\bar{1})$ $\qquad y = (.2\bar{1}\bar{3}322)$
- The result $z = (1.\bar{1}0\bar{1}221)$.

j	x_{j+2}	y_{j+2}	t_{j+1}	w_{j+2}	w_{j+1}	z_{j+1}	z_j
-1	1	2	1	-1	0*	1	0*
0	2	-1	0	1	-1	-1	1
1	-3	-3	-1	-2	1	0	-1
2	3	3	1	2	-2	-1	0
3	0	2	0	2	2	2	-1
4	-1	2	0	1	2	2	2
5	0	0	0	0	1	1	2
6	0	0	0	0	0	0	1

* latches initialized to 0.

RADIX $r > 2$ ONLINE ADDER

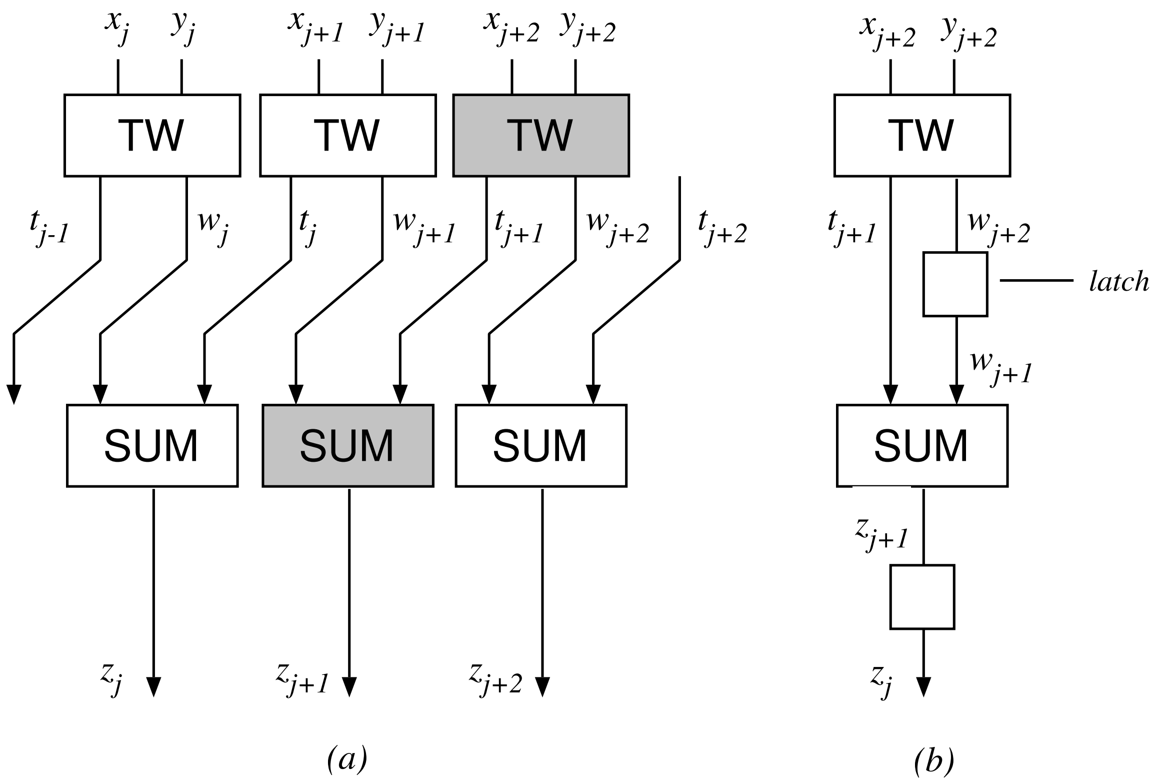


Figure 9.9: (a) A segment of radix-$r > 2$ signed-digit parallel adder. (b) Radix-$r > 2$ online adder. All latches cleared at start.

RADIX-2 ONLINE ADDER

- Digit-parallel radix-2 signed-digit adder converted into a radix-2 online adder with online delay $\delta = 2$

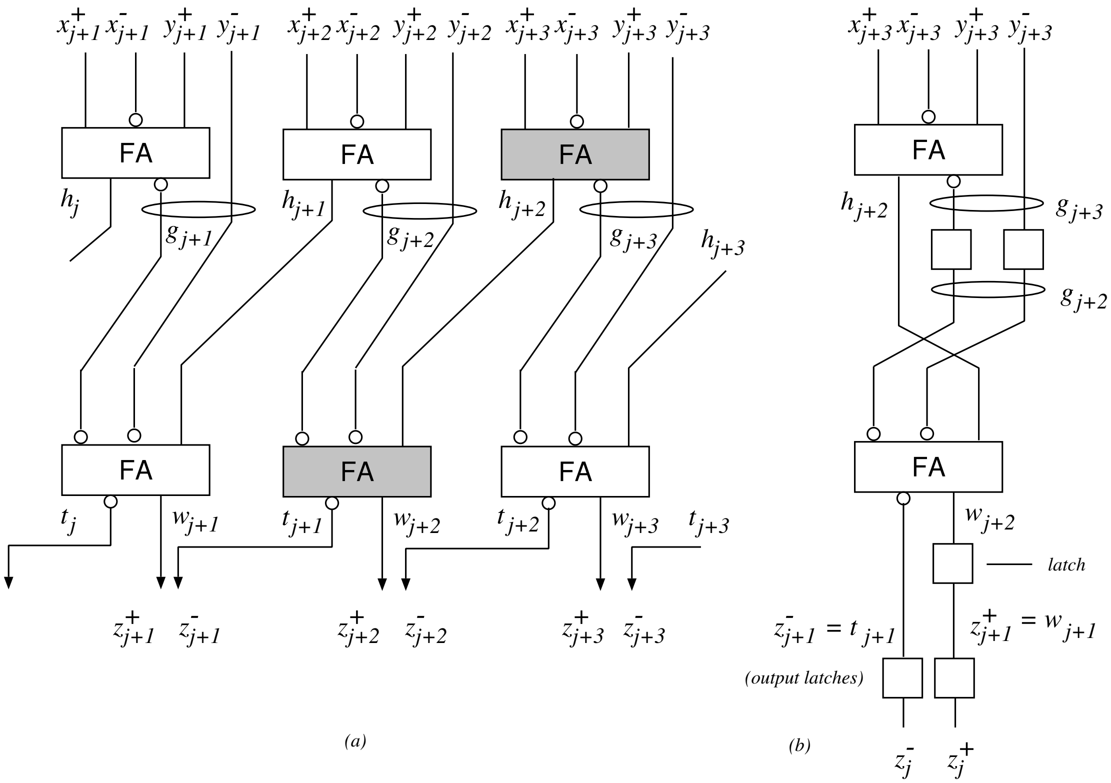


Figure 9.10: (a) A segment of radix-2 signed-digit parallel adder. (b) Online adder.

cont.

- The cycle time is $t_{cyc} = 2t_{FA} + t_{FF}$
- The operation time $T_{OLADD-2} = (2 + n + 1)t_{cyc}$
- The cost 2 FAs and 5 FFs.
- To reduce the cycle time, pipeline the two stages:
  reduces the cycle time by one t_{FA}; increases online delay to $\delta = 3$

EXAMPLE OF RADIX-2 ONLINE ADDITION

$$\begin{aligned} x &= (.010\bar{1}110\bar{1}) \\ y &= (.10\bar{1}01\bar{1}\bar{1}0) \\ z &= (1.\bar{1}0100\bar{1}01) \end{aligned}$$

j	x_{j+3}	y_{j+3}	$x^{+}_{j+3}x^{-}_{j+3}$	$y^{+}_{j+3}y^{-}_{j+3}$	h_{j+2}	g_{j+3}	g_{j+2}	$t_{j+1}w_{j+2}$	$z^{+}_{j+1}z^{-}_{j+1}$	z_j
-2	0	1	00	10	1	10	00*	0 1	- -	-
-1	1	0	10	00	1	10	10	0 0	10	-
0	0	-1	00	01	0	01	10	1 1	01	1
1	-1	0	01	00	0	10	01	1 1	11	-1
2	1	1	10	10	1	00	10	0 0	10	0
3	1	-1	10	01	0	11	00	0 1	00	1
4	0	-1	00	01	0	01	11	1 0	11	0
5	-1	0	01	00	0	10	01	1 1	01	0
6	0	0	00	00	0	00	10	1 1	11	-1
7	0	0	00	00	0	00	00	0 0	10	0
8	0	0	00	00	0	00	00	0 0	00	1

* g latches initialized to 00.

METHOD FOR DEVELOPING ONLINE ALGORITHMS

- Part 1: development of the residual recurrence

$$w[j+1] = G(w[j], x[j], x_{j+1+\delta}, y[j], y_{j+1+\delta}, z[j], z_{j+1})$$

  for $-\delta \leq j \leq n-1$ where

$$x[j] = \sum_{i=1}^{j+\delta} x_i r^{-i},\ \ y[j] = \sum_{i=1}^{j+\delta} y_i r^{-i},\ \ z[j] = \sum_{i=1}^{j} z_i r^{-i}$$

  are the online forms of the operands and the result

- Part 2: the result digit selection

$$z_{j+1} = F(w[j], x[j], x_{j+1+\delta}, y[j], y_{j+1+\delta}, z[j])$$

Part 1: RESIDUAL AND ITS RECURRENCE

Step 1. Describe the online operation by the error bound after j digits

$$|f(x[j], y[j]) - z[j]| < r^{-j}$$

Step 2 Transform expression to use only

- multiplication by r (shift),
- addition/subtraction,
- multiplication by a single digit

$$\underline{B} < G(f(x[j],\ y[j]) - z[j]) < \overline{B}$$

where G is the required transformation and $\underline{B}$ and $\overline{B}$ are the transformed bounds

Example: division error expression $|x[j]/y[j] - z[j]| < r^{-j}$ transformed into

$$|x[j] - z[j] \cdot y[j]| < |r^{-j} y[j]|$$

cont.

Step 3 Define a scaled residual

$$w[j] = r^j(G(f(x[j], y[j]) - z[j]))$$

with the bound

$$\underline{\omega} = r^j\underline{B} < w[j] < r^j\overline{B} = \overline{\omega}$$

and initial condition $w[-\delta] = 0$. $\underline{\omega}$ and $\overline{\omega}$ are the actual bounds determined in Step 6

Step 4 Determine a recurrence on $w[j]$

$$w[j+1] = rw[j] + r^{j+1}(G(f(x[j+1], y[j+1]) - z[j+1]) - G(f(x[j], y[j]) - z[j]))$$

Step 5 Decompose recurrence so that H_1 is independent of z_{j+1}

$$w[j+1] = rw[j] + H_1 + H_2(z_{j+1}) = v[j] + H_2(z_{j+1})$$

cont.

Step 6 Determine the bounds of $w[j+1]$ in terms of H_1 and H_2

$$\overline{\omega} = r\overline{\omega} + max(H1) + H2(a)$$

resulting in

$$\overline{\omega} = -\frac{\max(H_1) + H_2(a)}{r-1}$$

Similarly,

$$\underline{\omega} = -\frac{\min(H_1) + H_2(-a)}{r-1}$$

Part 2a: SELECTION FUNCTION WITH SELECTION CONSTANTS

$$z_{j+1} = k \text{ if } m_k \leq \widehat{v}[j] < m_{k+1}$$

where $\widehat{v}[j]$ is an estimate of $v[j]$ obtained by truncating the redundant representation of $v[j]$ to t fractional bits.

Selection constants need to satisfy

$$\max(\widehat{L}_k) \leq m_k \leq \min(\widehat{U}_{k-1})$$

where $[\widehat{L}_k,\ \widehat{U}_k]$ is the selection interval of the estimate $\widehat{v}[j]$

Step 7 Determine $[\widehat{L}_k,\ \widehat{U}_k]$

First, determine $[L_k,\ U_k]$ for $v[j]$

$$\overline{\omega} = U_k + H_2(k) \quad \underline{\omega} = L_k + H_2(k)$$

Substituting $\overline{\omega}$ and $\underline{\omega}$ the selection intervals for $v[j]$ is,

$$\begin{array}{rcl} U_k &=& -\frac{max(H_1)+H_2(a)}{r-1} - H_2(k) \\ L_k &=& -\frac{min(H_1)+H_2(-a)}{r-1} - H_2(k) \end{array}$$

cont.

Now restrict the intervals because of the use of the estimate $\widehat{v}[j]$

$$e_{min} \leq v[j] - \widehat{v}[j] \leq e_{max}$$

producing the error-restricted selection interval $[L_k^*,\ U_k^*]$ with

$$U_k^* = U_k - e_{max} \quad L_k^* = L_k + |e_{min}|$$

The errors are

- For carry-save representation $e_{max} = 2^{-t+1} - ulp$ and $e_{min} = 0$.
- For signed-digit representation $e_{max} = 2^{-t} - ulp$ and $e_{min} = -(2^{-t} - ulp)$.

$$\begin{array}{lcl} \widehat{U}_{k-1} & = & \lfloor U_{k-1}^* + 2^{-t} \rfloor_t \\ \widehat{L}_k & = & \lceil L_k^* \rceil_t \end{array}$$

where $\lfloor x \rfloor_t$ and $\lceil x \rceil_t$ indicate x values truncated to t fractional bits.

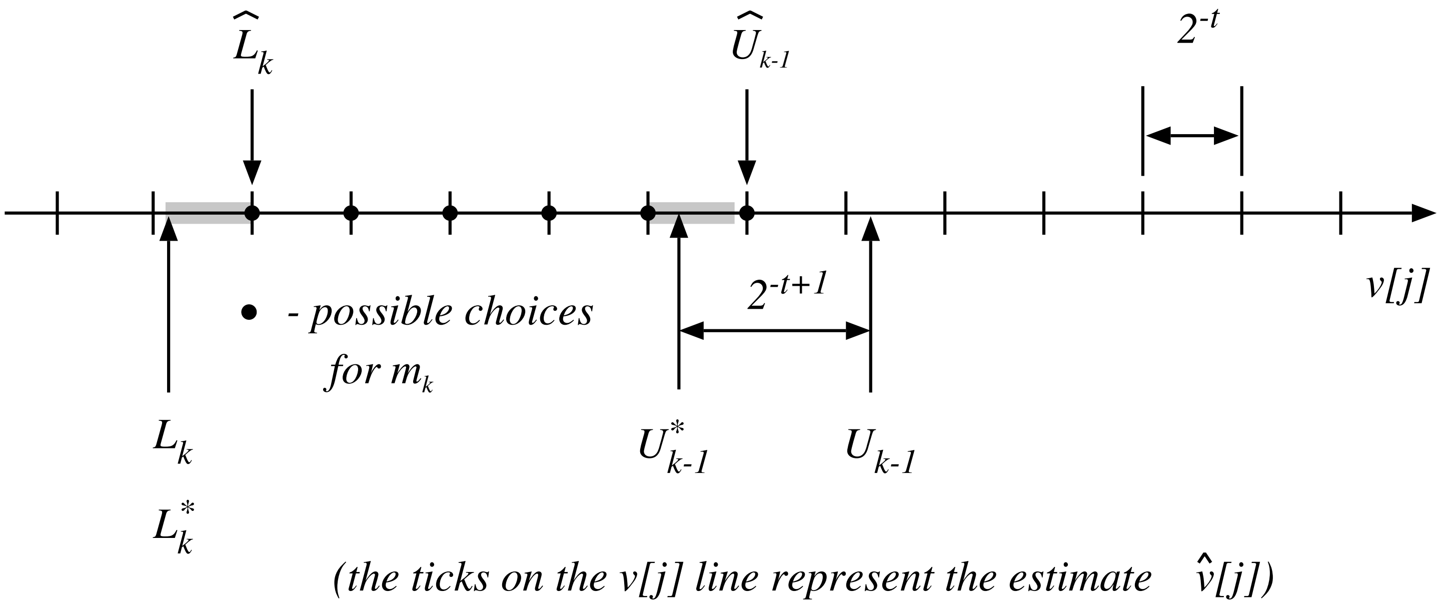

$\widehat{L}_k$
$\widehat{U}_{k-1}$
2^{-t}
$v[j]$
● - possible choices for m_k
2^{-t+1}
L_k
L_k^*
U_{k-1}^*
U_{k-1}
(the ticks on the v[j] line represent the estimate $\hat{v}[j]$)

Figure 9.11: The choices of selection constant m_k.

cont.

Step 8 Determine t and δ. To determine m_k, we need

$$\min(\widehat{U}_{k-1}) - \max(\widehat{L}_k) \geq 0$$

This relation between t and δ is used to choose suitable values.

Step 9 Determine the selection constants m_k and the range of $\widehat{v}[j]$ as

$$\lfloor r\underline{\omega} + \min(H_1) - e_{max} \rfloor_t \leq \widehat{v}[j] \leq \lfloor r\overline{\omega} + \ max(H_1) + |e_{min}| \rfloor_t$$

Part 2b: SELECTION BY ROUNDING

- In algorithms using a higher radix ($r > 4$)

$$w[j+1] = rw[j] + H_1 + H_2(z_{j+1}) = v[j] + H_2(z_{j+1})$$

- In the rounding method, the result digit is obtained as

$$z_{j+1} = \lfloor v[j] + \frac{1}{2} \rfloor$$

with $|v[j]| < r - \frac{1}{2}$ to avoid over-redundant output digit.

$$w[j+1] = v[j] + H_2(\lfloor v[j] + \frac{1}{2} \rfloor)$$

- For CS form of t fractional bits, the estimate error

$$e_{max} = 2^{-t+1} - ulp$$

- When $\hat{v}[j] = m_k - 2^{-t}$ it must be possible to choose $z_{j+1} = k - 1$

$$m_k - 2^{-t} + e_{max} = \frac{2k-1}{2} + 2^{-t} \leq \widehat{U}_{k-1}$$

GENERIC FORM OF EXECUTION AND IMPLEMENTATION.

- Execution: $n + \delta$ iterations of the recurrence, each one clock cycle
- Iterations (cycles) labeled from $-\delta$ to $n - 1$
- One digit of each input introduced during cycles $-\delta$ to $n - 1 - \delta$ and digits value 0 thereafter
- Result digits 0 for cycles $-\delta$ to -1 and z_1 is produced in cycle 0
- Result digit z_j is output in cycle j (one extra cycle to output z_n)

cont.

- The actions in cycle j:
  - Input $x_{j+1+\delta}$ and $y_{j+1+\delta}$.
  - Update $x[j{+}1] = (x[j], x_{j+1+\delta})$ and $y[j{+}1] = (y[j], y_{j+1+\delta})$ by appending the input digits.
  - Compute $v[j] = rw[j] + H_1$
  - Determine z_{j+1} using the selection function.
  - Update $z[j+1] = (z[j], z_{j+1+\delta})$ by appending the result digits.
  - Compute the next residual $w[j+1] = v[j] + H_2(z_{j+1})$
  - Output result digit z_j

IMPLEMENTATION

- Similar structure of algorithms → all implemented with same basic components, such as

  (i) registers to store operands, results, and residual vectors;

  (ii) multiplication of vector by digit;

  (iii) append units to append a new digit to a vector;

  (iv) Two-operand and multioperand redundant adders, such as signed digit adders, [3:2] carry-save adders and their generalization to [4:2] and [5:2] adders;

  (v) converters from redundant representations (i.e., signed digit and carry save) to conventional representations;

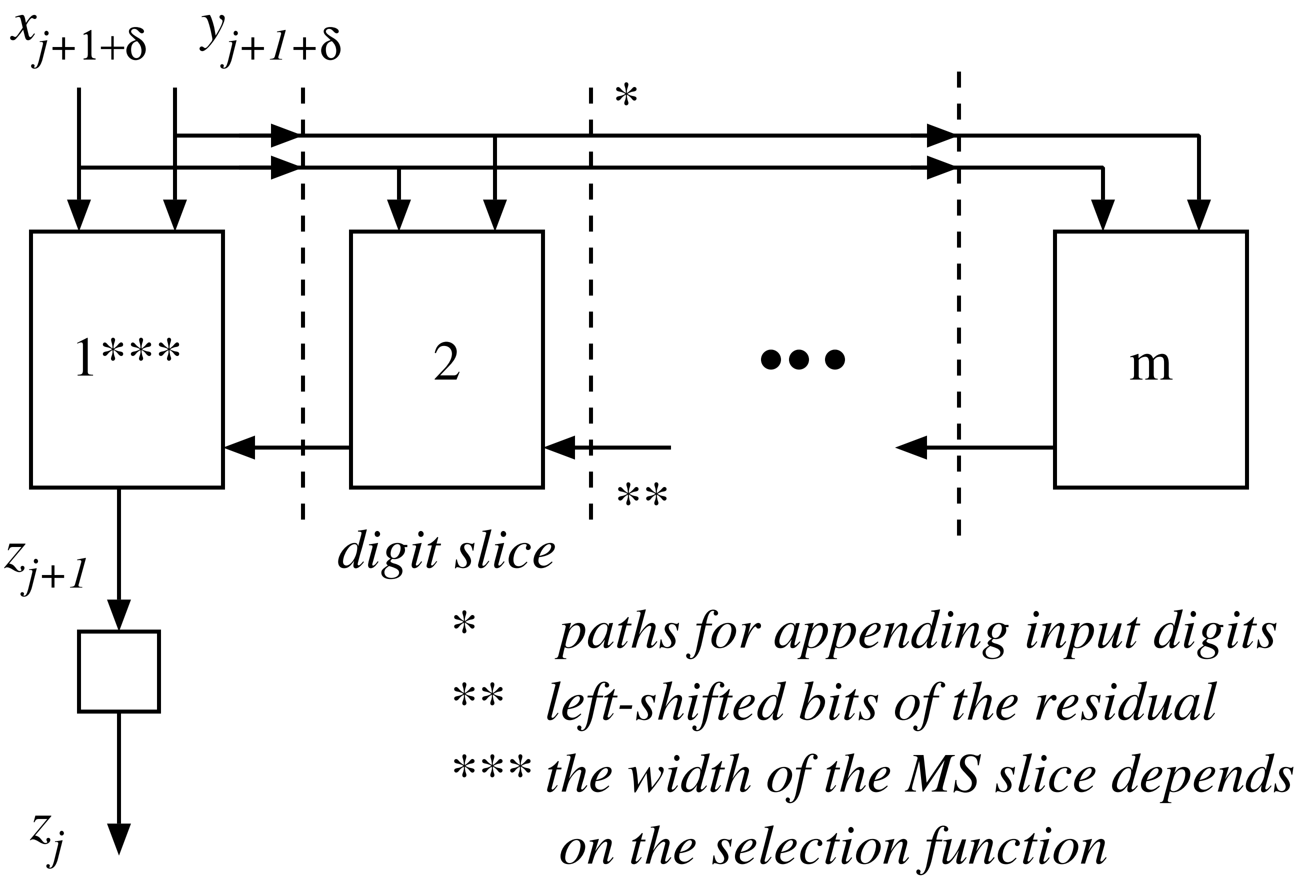
  (vi) carry-propagate adders of limited precision (3 to 6 bits) to produce estimates of the residual functions; and

  (vii) digit-selection schemes to obtain output digits.

cont.

- Online algorithm implementation similar to implementation of digit-recurrence algorithms
- Algorithms and implementations developed for most of basic arithmetic operations and for certain composite operations
- Larger set of operations possible than with LSDF approach

DIGIT-SLICE ORGANIZATION



* paths for appending input digits

** left-shifted bits of the residual

*** the width of the MS slice depends on the selection function

Figure 9.12: A typical digit-slice organization of online arithmetic unit

ONLINE MULTIPLICATION

- Online forms

$$x[j] = \sum_{i=1}^{j+\delta} x_i r^{-i}, \; y[j] = \sum_{i=1}^{j+\delta} y_i r^{-i}, \; p[j] = \sum_{i=1}^{j} p_i r^{-i}$$

- The error bound at cycle j

$$|x[j] \cdot y[j] - p[j]| < r^{-j}$$

- The residual

$$w[j] = r^j (x[j] \cdot y[j] - p[j])$$

with the bound $|w[j]| < \omega$

- The residual recurrence

$$\begin{aligned} w[j+1] &= rw[j] + (x[j]y_{j+1+\delta} + y[j+1]x_{j+1+\delta})r^{-\delta} - p_{j+1} \\ &= v[j] - p_{j+1} \end{aligned}$$

SELECTION FUNCTION

- Decomposition

$$H_1 = (x[j]y_{j+1+\delta} + y[j+1]x_{j+1+\delta})r^{-\delta} \quad H_2 = -p_{j+1}$$

- Bound

$$\overline{\omega} = -\underline{\omega} = \omega = \rho(1 - 2r^{-\delta})$$

- Selection intervals

$$\begin{aligned} U_k &= \rho(1 - 2r^{-\delta}) + k \\ L_k &= -\rho(1 - 2r^{-\delta}) + k \end{aligned}$$

- With carry-save representation for $w[j]$ and $v[j]$, the grid-restricted intervals are

$$\begin{aligned} \widehat{U}_k &= \lfloor \rho(1 - 2r^{-\delta}) + k - 2^{-t} \rfloor_t \\ \widehat{L}_k &= \lceil -\rho(1 - 2r^{-\delta}) + k \rceil_t \end{aligned}$$

- The expression to determine t and δ:

$$\lfloor \rho(1 - 2r^{-\delta}) + k - 1 - 2^{-t} \rfloor_t - \lceil -\rho(1 - 2r^{-\delta}) + k \rceil_t \geq 0$$

resulting in

$$\lfloor \rho(1 - 2r^{-\delta}) \rfloor_t \geq 2^{-1}(1 + 2^{-t})$$

cont.

- Several examples of relations between r, ρ, t, and δ

Radix	ρ	t	δ
2	1	2	3
4	1	2	2
	2/3	3	3
8	2/3	2	3

RADIX-2 ONLINE MULTIPLICATION

- $\delta = 3$ and $t = 2$
- Selection constants m_k's obtained from

$$\widehat{L}_k \leq m_k \leq \widehat{U}_{k-1}$$

where

$$\begin{aligned} \widehat{U}_k &= \lfloor 1 - 2^{-2} + k - 2^{-2} \rfloor_2 = k + 2^{-1} \\ \widehat{L}_k &= \lceil -1 + 2^{-2} + k \rceil_2 = k - 3 \times 2^{-2} \end{aligned}$$

- Since $\widehat{U}_{k-1} = k - 2^{-1}$ and $\widehat{L}_k = k - 3 \times 2^{-2}$, $m_k = k - 2^{-1}$ is acceptable. The selection constants are

$$m_0 = -2^{-1}, \quad m_1 = 2^{-1}$$

- Range of $\widehat{v}[j]$ is

$$-2 \leq \widehat{v}[j] \leq 7/4$$

- The selection function $SELM(\widehat{v}[j])$ is

$$p_{j+1} = SELM(\widehat{v}[j]) = \begin{cases} 1 & \textbf{if} \;\; 1/2 \leq \widehat{v}[j] \leq 7/4 \\ 0 & \textbf{if} \;\; -1/2 \leq \widehat{v}[j] \leq 1/4 \\ -1 & \textbf{if} \;\; -2 \leq \widehat{v}[j] \leq -3/4 \end{cases}$$

IMPLEMENTATION OF SELECTION FUNCTION

- Estimate $\hat{v}$ represented by (v_{-1}, v_0, v_1, v_2)
- Product digit $p_{j+1} = (pp, pn)$ with the code

p_{j+1}	pp	pn
1	1	0
0	0	0
-1	0	1

- Switching expressions:

$$\begin{aligned} pp &= v'_{-1}(v_0 + v_1) \\ pn &= v_1(v'_0 + v'_1) \end{aligned}$$

$\hat{v}$	$v_{-1}v_0v_1v_2$	p_{j+1}
7/4	01.11	1
6/4	01.10	1
5/4	01.01	1
1	01.00	1
3/4	00.11	1
1/2	00.10	1
1/4	00.01	0
0	00.00	0
-1/4	11.11	0
-1/2	11.10	0
-3/4	11.01	-1
-1	11.00	-1
-5/4	10.11	-1
-6/4	10.10	-1
-7/4	10.01	-1
-2	10.00	-1

```
1. [$Initialize$]
      $x[-3] = y[-3] = w[-3] = 0$
   for  $j = -3, -2, -1$
      $x[j+1] \leftarrow CA(x[j], x_{j+4});\ \ y[j+1] \leftarrow CA(y[j], y_{j+4})$
      $v[j] = 2w[j] + (x[j]y_{j+4} + y[j+1]x_{j+4})2^{-3}$
      $w[j+1] \leftarrow v[j]$
   end for
2. [$Recurrence$]
   for  $j = 0 \ldots n-1$
      $x[j+1] \leftarrow CA(x[j], x_{j+4});\ \ y[j+1] \leftarrow CA(y[j], y_{j+4})$
      $v[j] = 2w[j] + (x[j]y_{j+4} + y[j+1]x_{j+4})2^{-3}$
      $p_{j+1} = SELM(\widehat{v[j]});$
      $w[j+1] \leftarrow v[j] - p_{j+1}$
      $P_{out} \leftarrow p_{j+1}$
   end for
```

Figure 9.13: Radix-2 online multiplication algorithm.

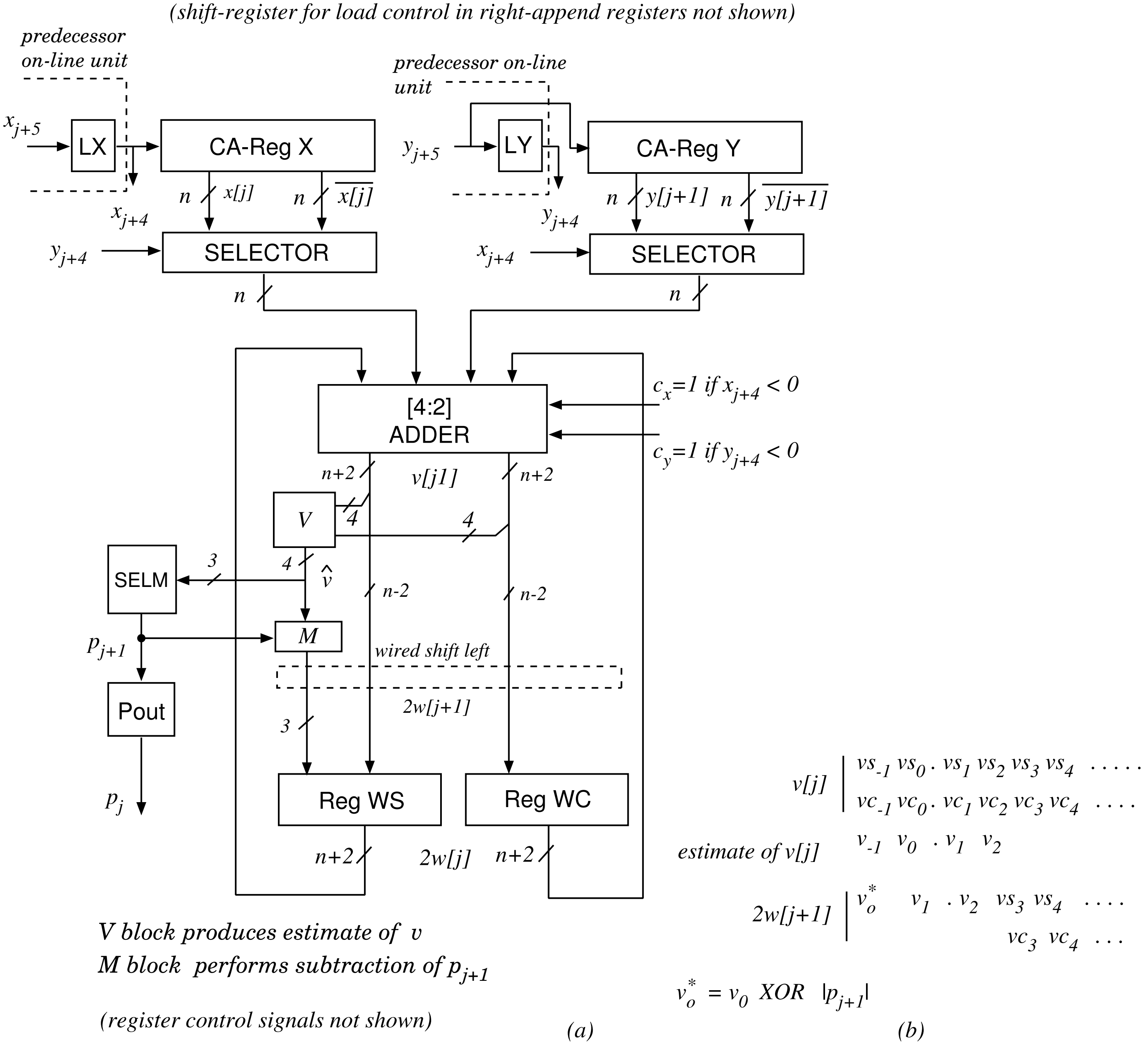


Figure 9.14: (a) Implementation of radix-2 online multiplier. (b) Calculation of $2w[j+1]$.

EXAMPLE OF RADIX-2 ONLINE MULTIPLICATION

Operands:

$$\begin{aligned} x &= (.110\bar{1}10\bar{1}1) \\ y &= (.101\bar{1}\bar{1}110) \end{aligned}$$

j	x_{j+4}	y_{j+4}	$x[j+1]$	$y[j+1]$	$v[j]$	p_{j+1}	$w[j+1]$
-3	1	1	.1	.1	00.0001	0	00.0001
-2	1	0	.11	.10	00.00110	0	00.00110
-1	0	1	.110	.101	00.011110	0	00.011110
0	-1	-1	.1011	.1001	00.1100011	1	11.1100011
1	1	-1	.10111	.10001	11.10000111	0	11.10000111
2	0	1	.101110	.100011	11.001001010	-1	00.001001010
3	-1	1	.1011011	.1000111	00.0100111101	0	00.0100111101
4	1	0	.10110111	.10001110	00.10110000010	1	11.10110000010
5	0	0	.10110111	.10001110	11.0110000010	-1	00.0110000010
6	0	0	.10110111	.10001110	00.110000010	1	11.110000010
7	0	0	.10110111	.10001110	11.10000010	0	11.10000010

cont.

- Computed product: $p = (.10\bar{1}01\bar{1}10)$
- The exact double precision product $p^* = (.0110010110000010)$
- The absolute error wrt to the exact product truncated to 8 bits:

$$|p - p^*_{tr}| = 2^{-8}$$

- Note: $p[8] + w[8]2^{-8} = p^*$

ONLINE DIVISION

- Online forms

$$x[j] = \sum_{i=1}^{j+\delta} x_i r^{-i}, \;\; y[j] = \sum_{i=1}^{j+\delta} y_i r^{-i}, \;\; q[j] = \sum_{i=1}^{j} q_i r^{-i}$$

- Error bound at cycle j

$$|x[j] - q[j]d[j]| < d[j]r^{-j}$$

- Residual

$$w[j] = r^j(x[j] - q[j]d[j]) \;\; \|w[j]\| < \omega \leq d[j]$$

- Residual recurrence

$$\begin{aligned} w[j+1] &= rw[j] + x_{j+1+\delta}r^{-\delta} - q[j]d_{j+1+\delta}r^{-\delta} - d[j+1]q_{j+1} \\ &= v[j] - d[j+1]q_{j+1} \end{aligned}$$

RADIX-2 ONLINE DIVISION

- $\delta = 4$ and $t = 3$
- Selection intervals and selection constants

$$\begin{aligned}\min \widehat{U}_0 &= \widehat{U}_0[d[j+1] = 1/2] = 2^{-1} - 2^{-3} + 0 - 2^{-3} = 2^{-2}\\ \max \widehat{L}_1 &= \widehat{L}_1[d[j+1] = 1] = -1 + 2^{-3} + 1 = 2^{-3}\end{aligned}$$

resulting in $m_1 = 2^{-2}$

$$\begin{aligned}\min \widehat{U}_{-1} &= \widehat{U}_{-1}[d[j+1] = 1] = 1 - 2^{-3} - 1 - 2^{-3} = -2^{-2}\\ \max \widehat{L}_0 &= \widehat{L}_0[d[j+1] = 1/2] = -2^{-1} + 2^{-3} = -3 \times 2^{-3}\end{aligned}$$

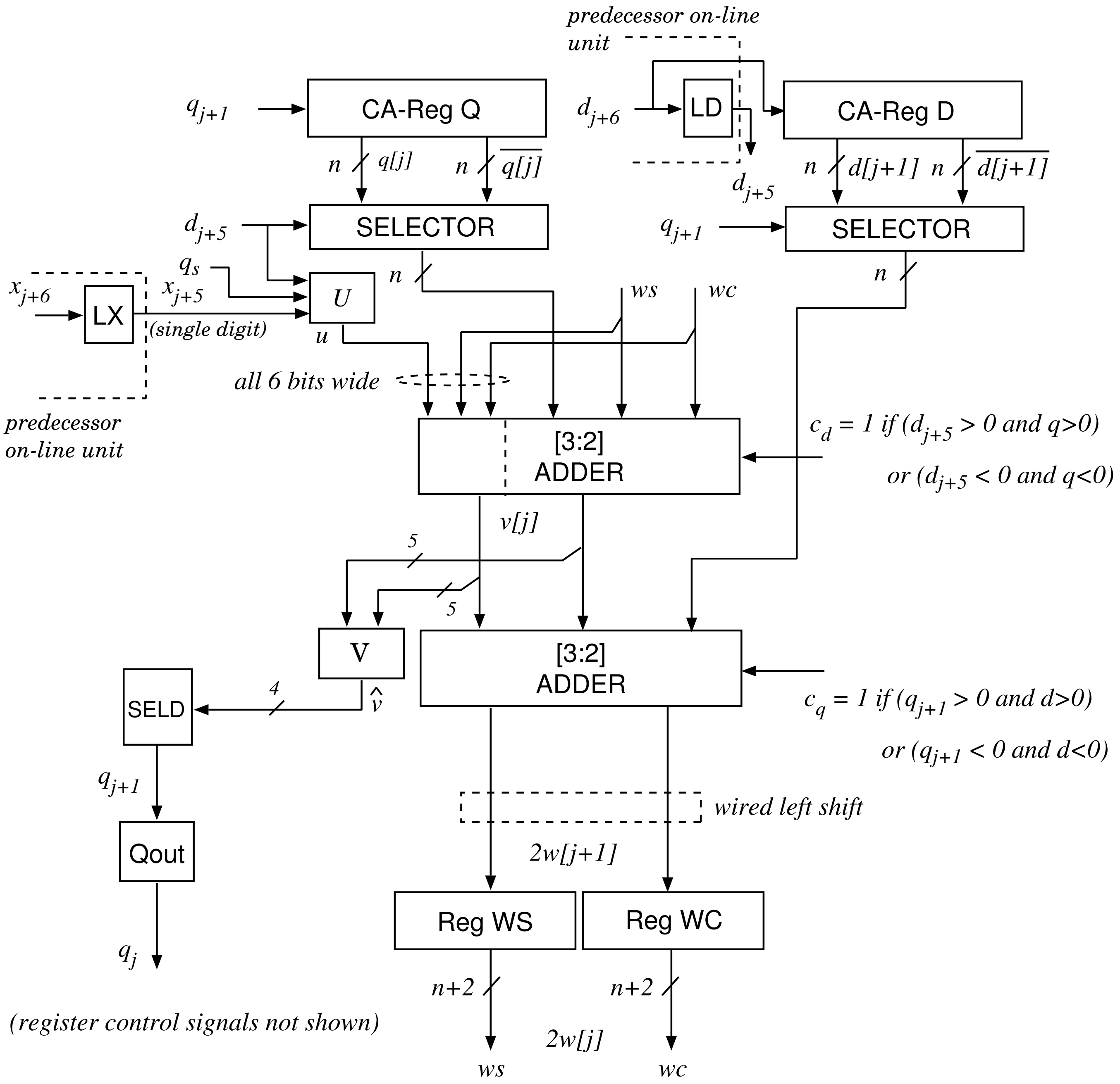
so that $m_0 = -2^{-2}$.

$$q_{j+1} = SELD(\widehat{v}[j]) = \begin{cases} 1 & \textbf{if} \;\; 1/4 \leq \widehat{v}[j] \leq 15/8 \\ 0 & \textbf{if} \;\; -1/4 \leq \widehat{v}[j] \leq 1/8 \\ -1 & \textbf{if} \;\; -2 \leq \widehat{v}[j] \leq -1/2 \end{cases}$$

RADIX-2 ONLINE DIVISION: ALGORITHM

```
1. [Initialize]
     $x[-4] = d[-4] = w[-4] = q[0] = 0$
   for $j = -4, \ldots, -1$
     $d[j+1] \leftarrow CA(d[j], d_{j+5})$
     $v[j] = 2w[j] + x_{j+5}2^{-4}$
     $w[j+1] \leftarrow v[j]$
   end for
2. [Recurrence]
   for $j = 0 \ldots n-1$
     $d[j+1] \leftarrow CA(d[j], d_{j+5})$
     $v[j] = 2w[j] + x_{j+5}2^{-4} - q[j]d_{j+5}2^{-4}$
     $q_{j+1} = SELD(\widehat{v}[j]);$
     $w[j+1] \leftarrow v[j] - q_{j+1}d[j+1]$
      $q[j+1] \leftarrow CA(q[j], q_{j+1})$
     $Q_{out} \leftarrow q_{j+1}$
   end for
```

(shift-register for load control in right-append registers not shown)



(register control signals not shown)

Figure 9.16: Block diagram of radix-2 online divider.

REDUCTION OF DIGIT-SLICES

- Selection valid if (t fractional bits)

$$p - 2h + \delta \geq t$$

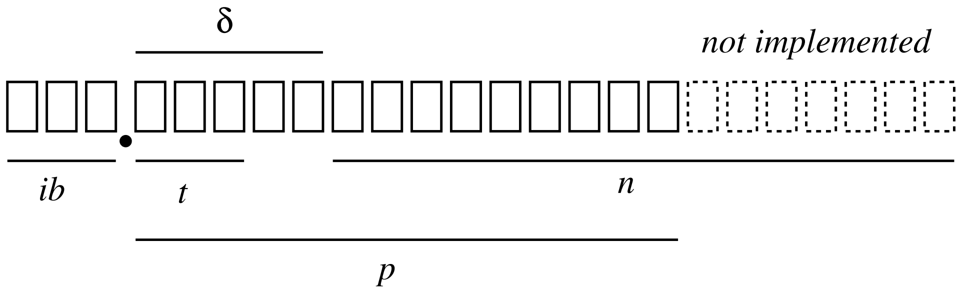
- $p + h = n + \delta$

$$p = \left\lceil \frac{2n + \delta + t}{3} \right\rceil$$

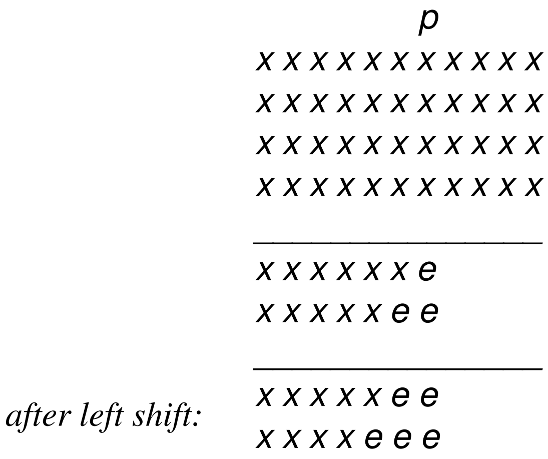
- Total number of bit-slices: $ib + p$, ib - no. integer bits
- For example, the number of bit-slices for 32-bit radix-2 online multiplication is

$$2 + \left\lceil \frac{2 \times 32 + 3 + 2}{3} \right\rceil = 2 + 23 = 25$$

compared to 34 in implementation without slice reduction.



(a)



(b)

Figure 9.17: Reduction of bit-slices in implementation.

MULTI-OPERATION AND COMPOSITE ONLINE ALGORITHMS

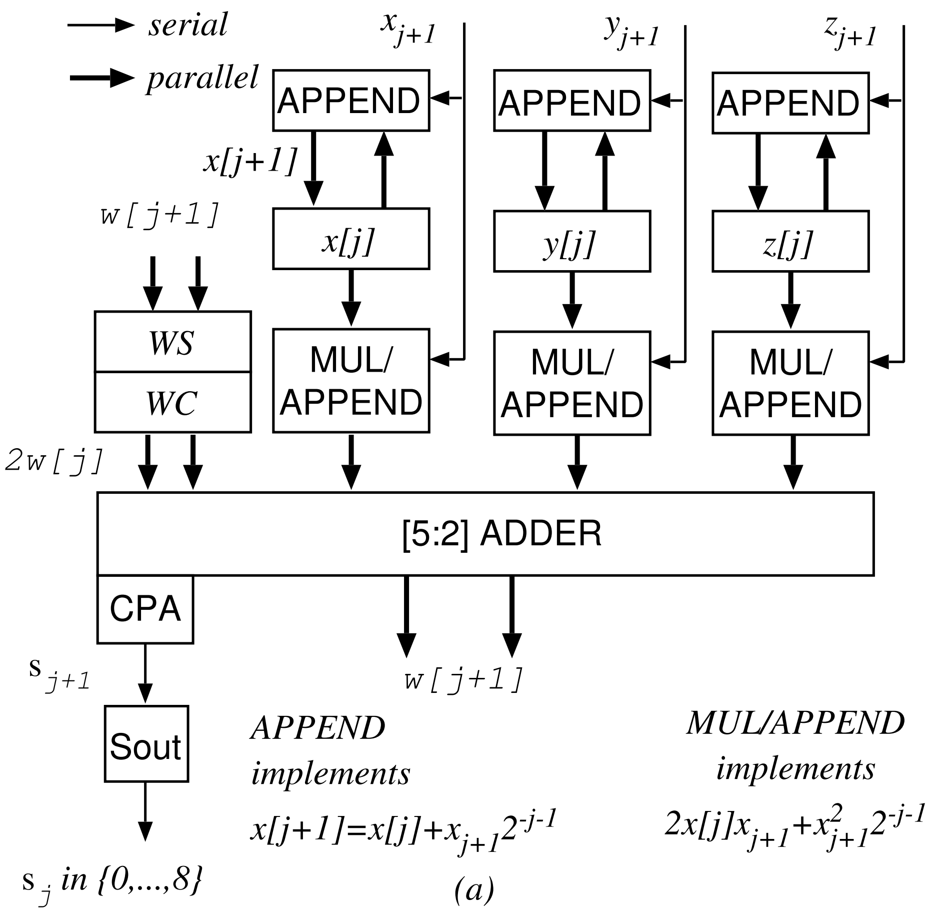
- To reduce the overall online delay of a group of operations

  - combine several operations into a single *multi-operation online algorithm*
- Example: $x^2 + y^2 + z^2$
- Inputs in [1/2,1), output in [1/4, 3)
- Online delay $\delta_{ss} = 0$ when the output digit is over-redundant.
- Online delay (3+2+2=7) of the corresponding network

ALGORITHM FOR SUM OF SQUARES

```
1. [Initialize]
     $w[0] = x[0] = y[0] = z[0] = 0$
2. [Recurrence]
   for  $j = 0 \dots n-1$
     $v[j] = 2w[j] + (2x[j] + x_j 2^{-j})x_j + (2y[j] + y_j 2^{-j})y_j + (2z[j] + z_j 2^{-j})z_j$
     $w[j+1] \leftarrow csfract(v[j])$
     $s_{j+1} \leftarrow csint(v[j])$
     $x[j+1] \leftarrow (x[j], x_{j+1});\ y[j+1] \leftarrow (y[j], y_{j+1});\ z[j+1] \leftarrow (z[j], z_{j+1})$
     $S_{out} \leftarrow s_{j+1}$
   end for
```

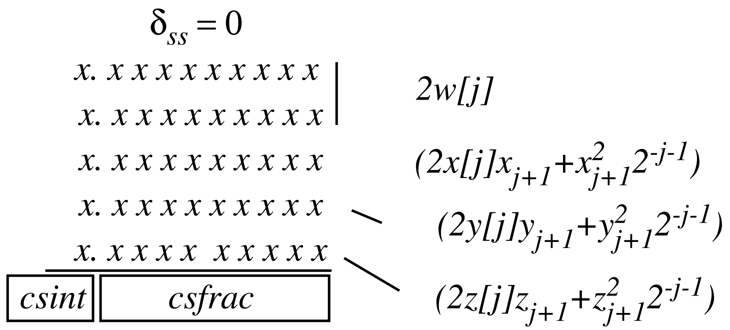
Figure 9.18: Radix-2 online sum of squares algorithm.



APPEND implements $x[j+1]=x[j]+x_{j+1}2^{-j-1}$

MUL/APPEND implements $2x[j]x_{j+1}+x_{j+1}^2 2^{-j-1}$

(a)



max(csint) = 8

Note: the fractional portion of the 5-2 CSA produces at most three carries

COMPOSITE ALGORITHM

- $d = \sqrt{(x^2 + y^2 + z^2)}$
- Overall online delay of 5
- A network of standard online modules: online delay of 11

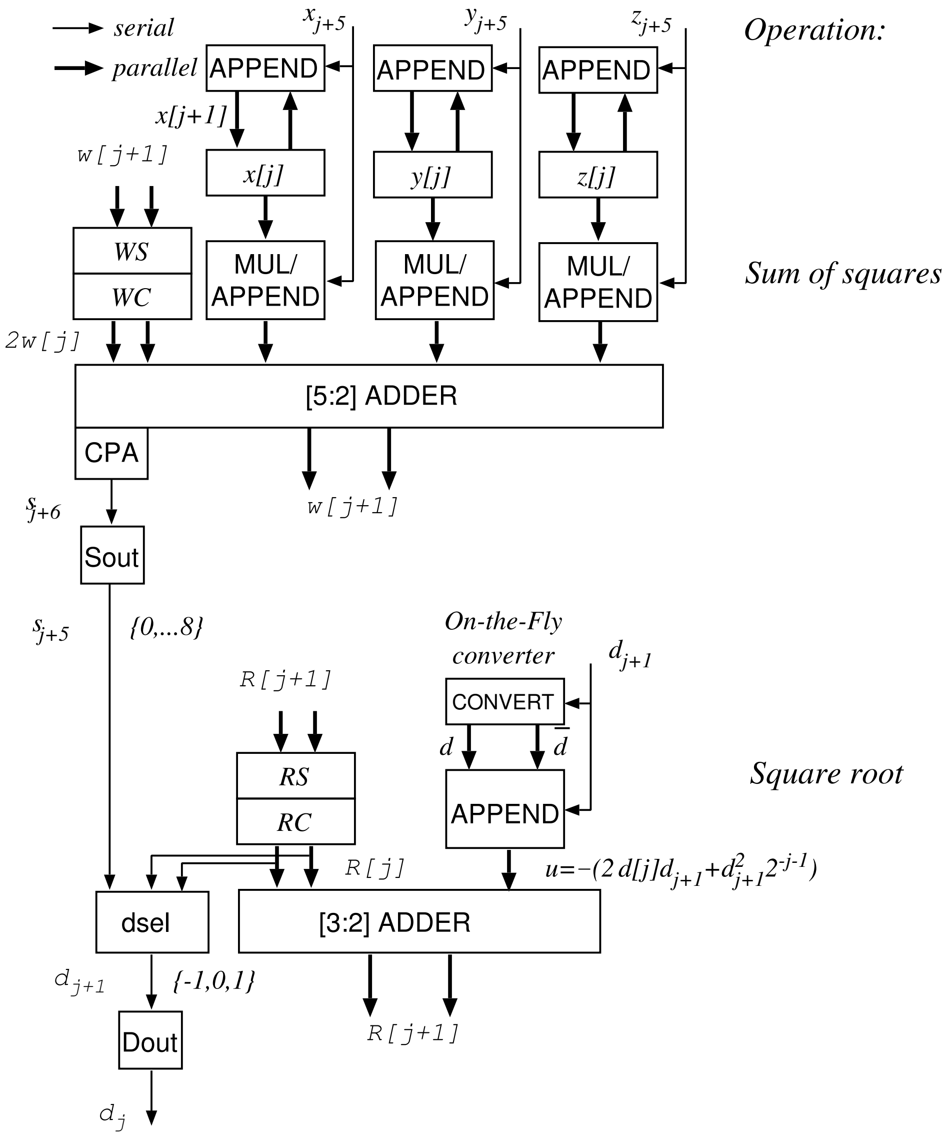


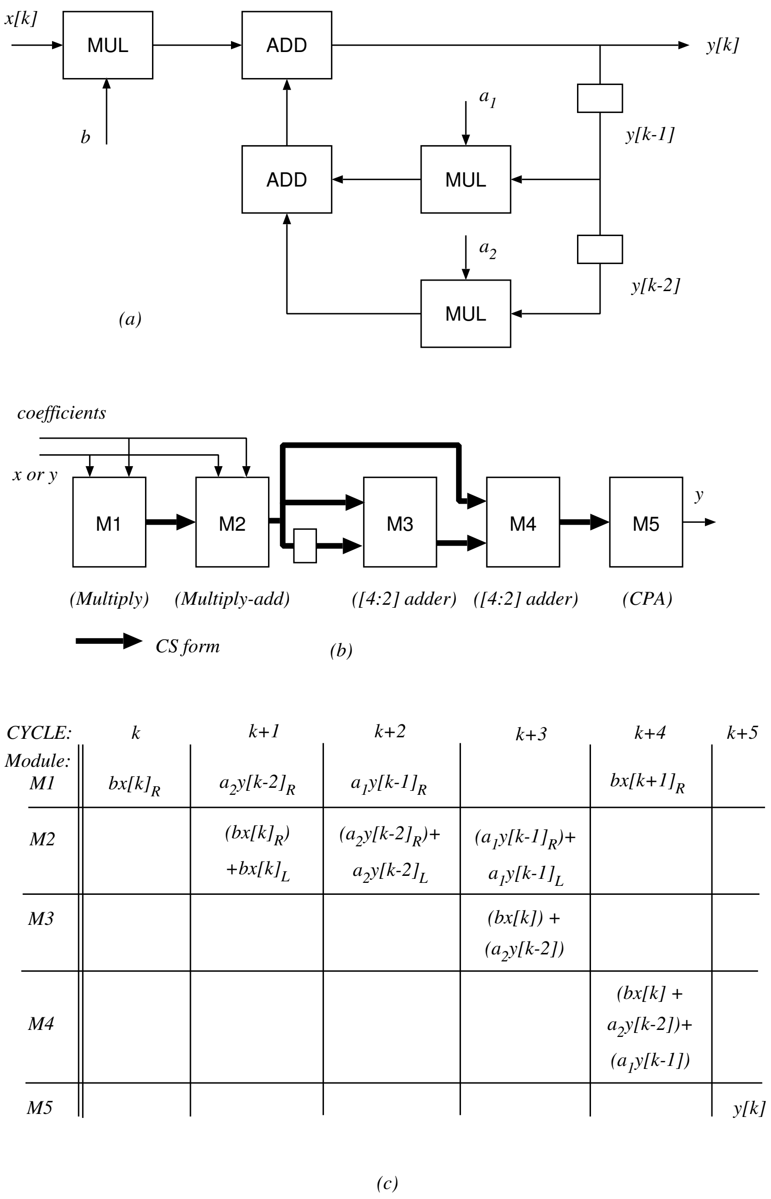
Figure 9.20: Composite scheme for computing $d = \sqrt{(x^2 + y^2 + z^2)}$.

ONLINE IMPLEMENTATION OF RECURSIVE FILTER

- IIR filter

$$y(k) = a_1 y(k-1) + a_2 y(k-2) + bx(k)$$

- **Conventional parallel arithmetic**
  - time to obtain $y[k]$: $T_{CONV} = 6t_{module}$.
  - $t_{module} \approx \ 6t_{FA}$
  - rate of filter computation: $R_{CONV} \approx 1/(4 \times 6t_{FA})$
- LSDF serial arithmetic
  - time to obtain $y[k]$: $T_{LSDF} = nt_{FA}$.
  - rate of filter computation: $R_{LSDF} \approx 1/(n \times t_{FA})$
- **Online arithmetic**
  - Multioperation modules of type $vu + w$, online delay of 4
  - cycle time $t_M \approx 3t_{FA}$
  - Throughput independent of working precision but not the number of online units
  - Rate: $R_{OL} = 1/(\Delta_{iter} \times t_M) \approx 1/(12t_{FA})$



CYCLE: Module:	k	k+1	k+2	k+3	k+4	k+5
M1	$bx[k]_R$	$a_2y[k-2]_R$	$a_1y[k-1]_R$		$bx[k+1]_R$	
M2		$(bx[k]_R)$ $+bx[k]_L$	$(a_2y[k-2]_R)+$ $a_2y[k-2]_L$	$(a_1y[k-1]_R)+$ $a_1y[k-1]_L$		
M3				$(bx[k]) +$ $(a_2y[k-2])$		
M4					$(bx[k] +$ $a_2y[k-2])+$ $(a_1y[k-1])$	
M5						y[k]

Figure 9.21: Conventional implementation of second-order IIR filter: (a) Filter. (b) 5-stage pipeline. (c) Timing diagram.

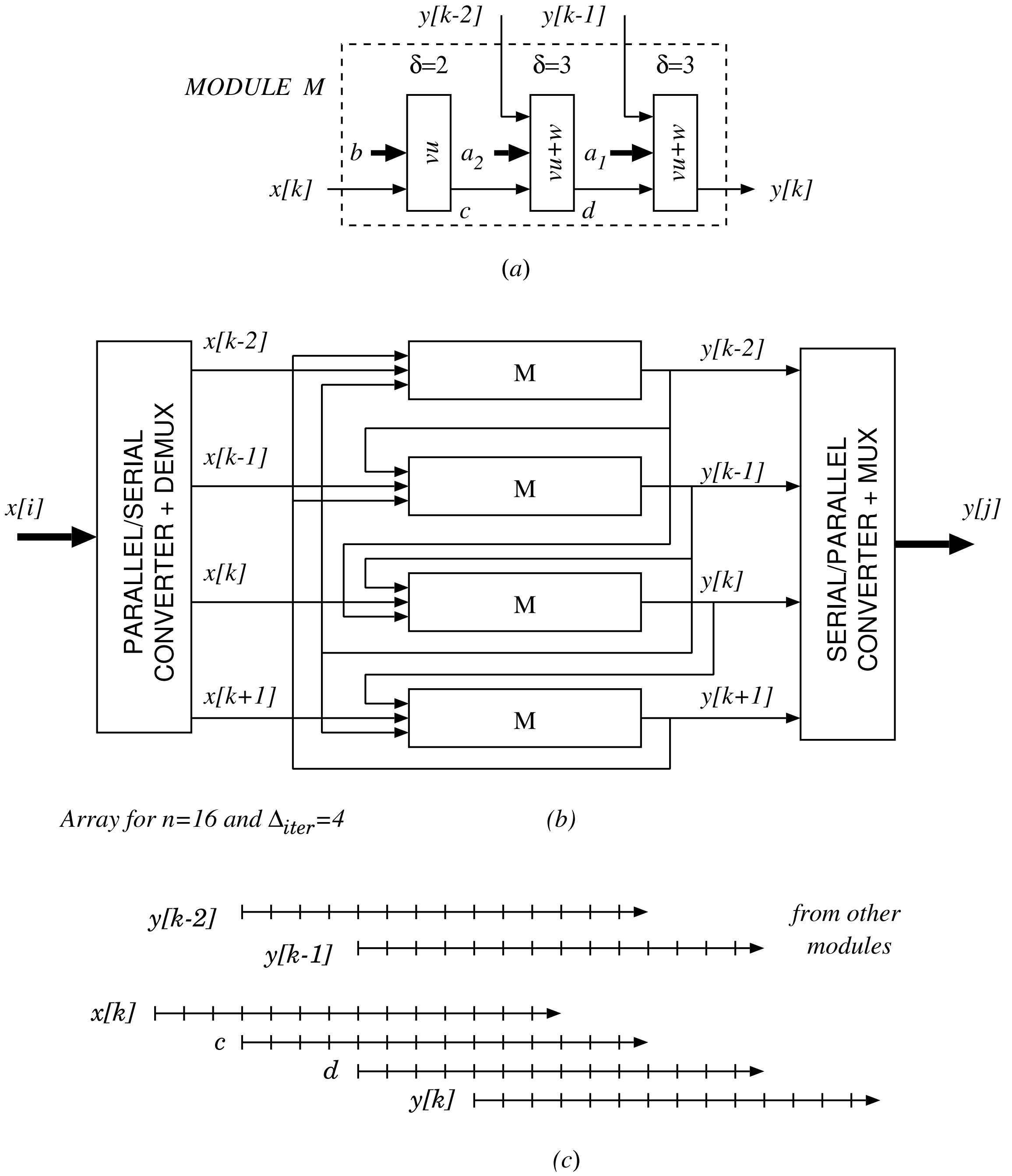


Figure 9.22: Online implementation of second-order IIR filter.

CORDIC ALGORITHM AND IMPLEMENTATIONS

- CORDIC METHOD
- ROTATION AND VECTORING MODE
- CONVERGENCE, PRECISION AND RANGE
- SCALING FACTOR AND COMPENSATION
- IMPLEMENTATIONS: word-serial and pipelined
- EXTENSION TO HYPERBOLIC AND LINEAR COORDINATES
- UNIFIED DESCRIPTION
- REDUNDANT ADDITION AND HIGH RADIX

MAIN USES

- REALIZATION OF ROTATIONS
- CALCULATION OF TRIGONOMETRIC FUNCTIONS
- CALCULATION OF INVERSE TRIGONOMETRIC FUNCTION $\tan^{-1}(a/b)$
- CALCULATION OF $\sqrt{a^2+b^2}$, etc.
- EXTENDED TO HYPERBOLIC FUNCTIONS
- DIVISION AND MULTIPLICATION
- CALCULATION OF SQRT, LOG, AND EXP
- FOR LINEAR TRANSFORMS, DIGITAL FILTERS, AND SOLVING LIN. SYSTEMS
- MAIN APPLICATIONS: DSP, IMAGE PROCESSING, 3D GRAPHICS, ROBOTICS.

CORDIC ALGORITHM

- CIRCULAR COORDINATE SYSTEM

- PERFECT ROTATION:

$$\begin{aligned} x_R &= M_{in}\cos(\beta+\theta) = x_{in}\cos\theta - y_{in}\sin\theta \\ y_R &= M_{in}\sin(\beta+\theta) = x_{in}\sin\theta + y_{in}\cos\theta \end{aligned}$$

- M_{in} – THE MODULUS OF THE VECTOR
- β – THE INITIAL ANGLE

- IN MATRIX FORM:

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} = ROT(\theta) \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}$$

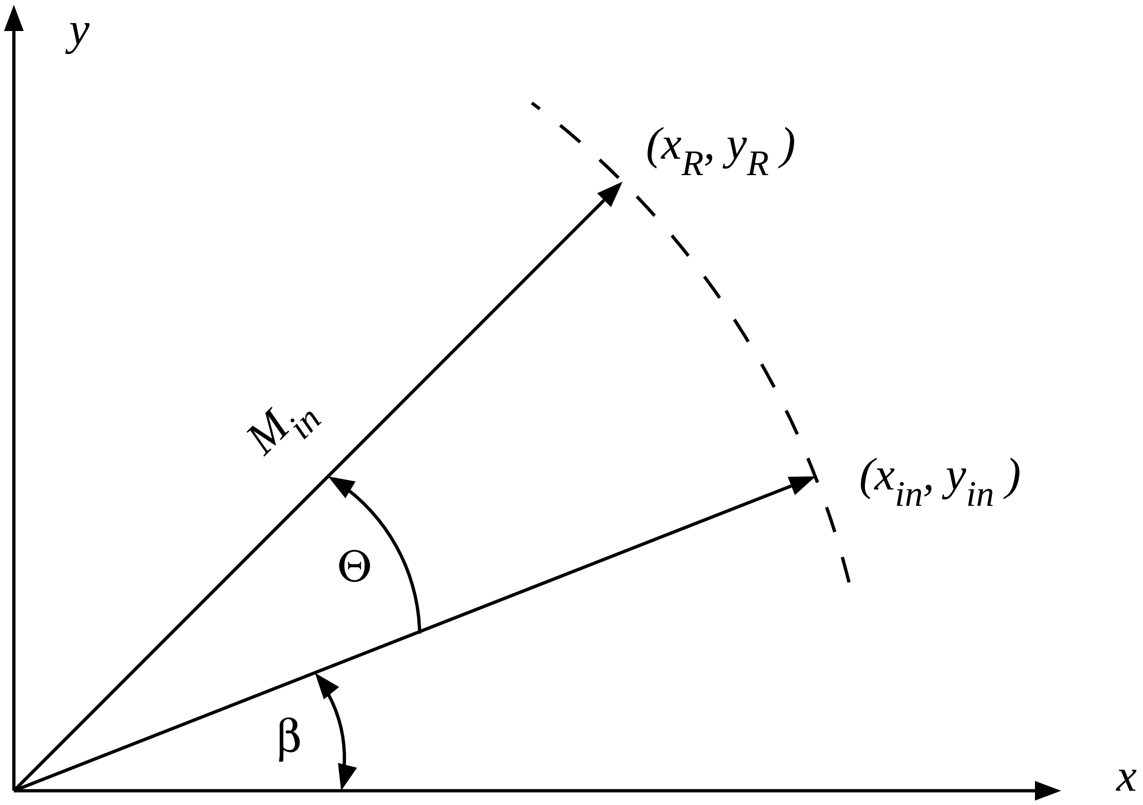


Figure 11.1: VECTOR ROTATION

MICRO-ROTATIONS

- USE ELEMENTARY ROTATION ANGLES α_j
- DECOMPOSE THE ANGLE θ:

$$\theta = \sum_{j=0}^{\infty} \alpha_j$$

SO THAT

$$ROT(\theta) = \prod_{j=0}^{\infty} ROT(\alpha_j)$$

- THEN $ROT(\alpha_j)$:

$$\begin{aligned} x_R[j+1] &= x_R[j]\cos(\alpha_j) - y_R[j]\sin(\alpha_j) \\ y_R[j+1] &= x_R[j]\sin(\alpha_j) + y_R[j]\cos(\alpha_j) \end{aligned}$$

SIMPLIFYING MICRO-ROTATIONS

- HOW TO AVOID MULTIPLICATIONS?

1. DECOMPOSE ROTATION INTO:

   SCALING OPERATION AND ROTATION-EXTENSION

$$\begin{aligned} x_R[j+1] &= \cos(\alpha_j)(x_R[j] - y_R[j]\tan(\alpha_j)) \\ y_R[j+1] &= \cos(\alpha_j)(y_R[j] + x_R[j]\tan(\alpha_j)) \end{aligned}$$

2. CHOOSE ELEMENTARY ANGLES

$$\alpha_j = \tan^{-1}(\sigma_j(2^{-j})) = \sigma_j \tan^{-1}(2^{-j})$$

WITH $\sigma_j \in \{-1, 1\}$

RESULTS IN ROTATION-EXTENSION RECURRENCE WITHOUT MPYs

$$\begin{aligned} x[j+1] &= x[j] - \sigma_j 2^{-j} y[j] \\ y[j+1] &= y[j] + \sigma_j 2^{-j} x[j] \end{aligned}$$

$\Longrightarrow$ONLY ADDITIONS AND SHIFTS

ROTATION-EXTENSION (cont.)

- ROTATION-EXTENSION SCALES MODULUS $M[j]$

$$M[j{+}1] = K[j]M[j] = \frac{1}{\cos\alpha_j}M[j] = (1{+}\sigma_j^2 2^{-2j})^{1/2}M[j] = (1{+}2^{-2j})^{1/2}M[j]$$

- TOTAL SCALING FACTOR

$$K = \prod_{j=0}^{\infty}(1 + 2^{-2j})^{1/2} \approx 1.6468$$

CONSTANT, INDEPENDENT OF THE ANGLE

- RECURRENCE FOR DECOMPOSITION/ACCUMULATION OF ANGLE:

$$z[j+1] = z[j] - \alpha_j = z[j] - \sigma_j \tan^{-1}(2^{-j})$$

IMPLEMENTATION OF CORDIC ITERATION

CORDIC MICROROTATION

$$\begin{aligned} x[j+1] &= x[j] - \sigma_j 2^{-j} y[j] \\ y[j+1] &= y[j] + \sigma_j 2^{-j} x[j] \\ z[j+1] &= z[j] - \sigma_j \tan^{-1}(2^{-j}) \end{aligned}$$

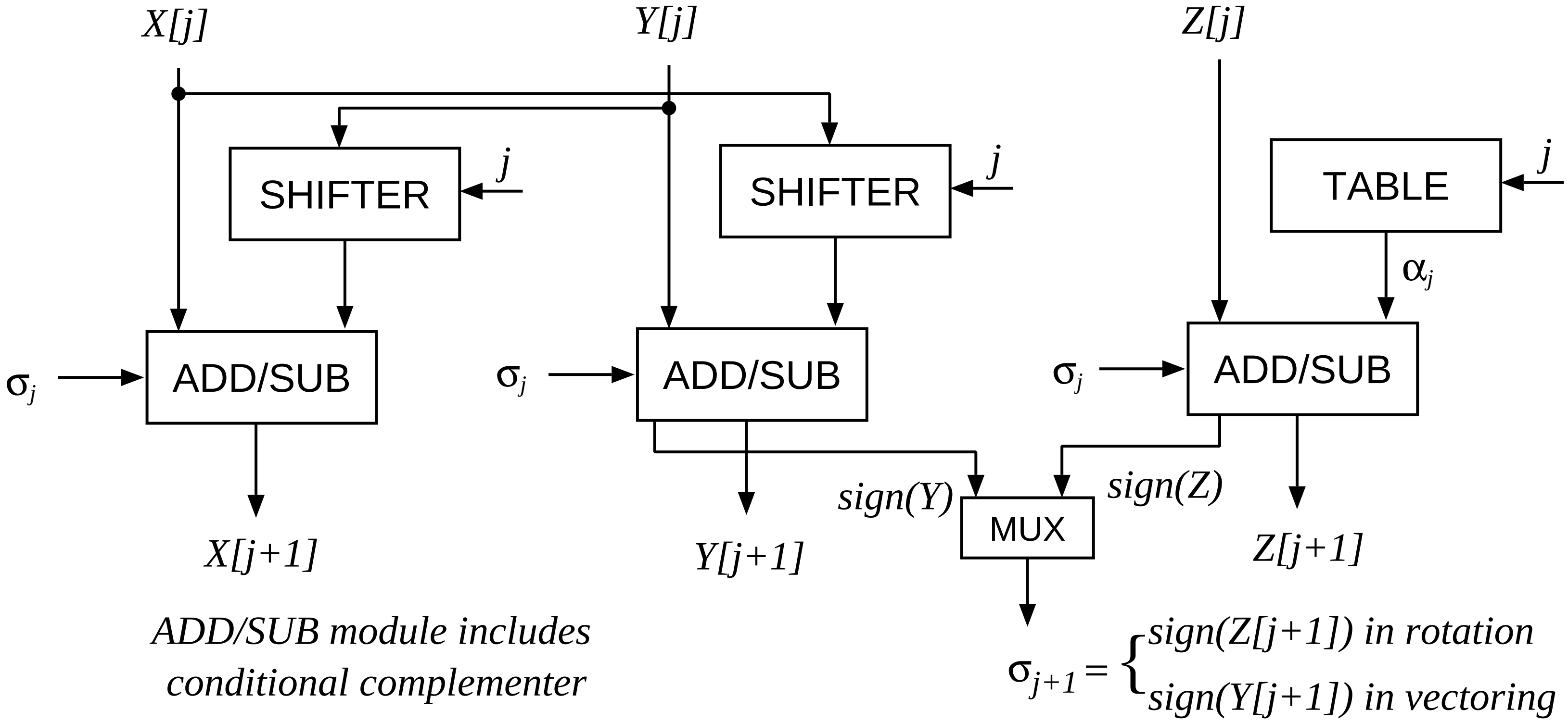


Figure 11.2: IMPLEMENTATION OF ONE ITERATION.

ROTATION MODE

- ROTATE AN INITIAL VECTOR (x_{in}, y_{in}) BY θ
- DECOMPOSE THE ANGLE

$$z[j+1] = z[j] - \sigma_j \tan^{-1}(2^{-j})$$

$$z[0] = \theta \quad x[0] = x_{in} \quad y[0] = y_{in}$$

$$\sigma_j = \begin{cases} 1 & if \ z[j] \geq 0 \\ -1 & if \ z[j] < 0 \end{cases}$$

- PERFORM MICRO-ROTATIONS
- FINAL VALUES

$$\begin{aligned} x_f &= K(x_{in} \cos\theta - y_{in} \sin\theta) \\ y_f &= K(x_{in} \sin\theta + y_{in} \cos\theta) \\ z_f &= 0 \end{aligned}$$

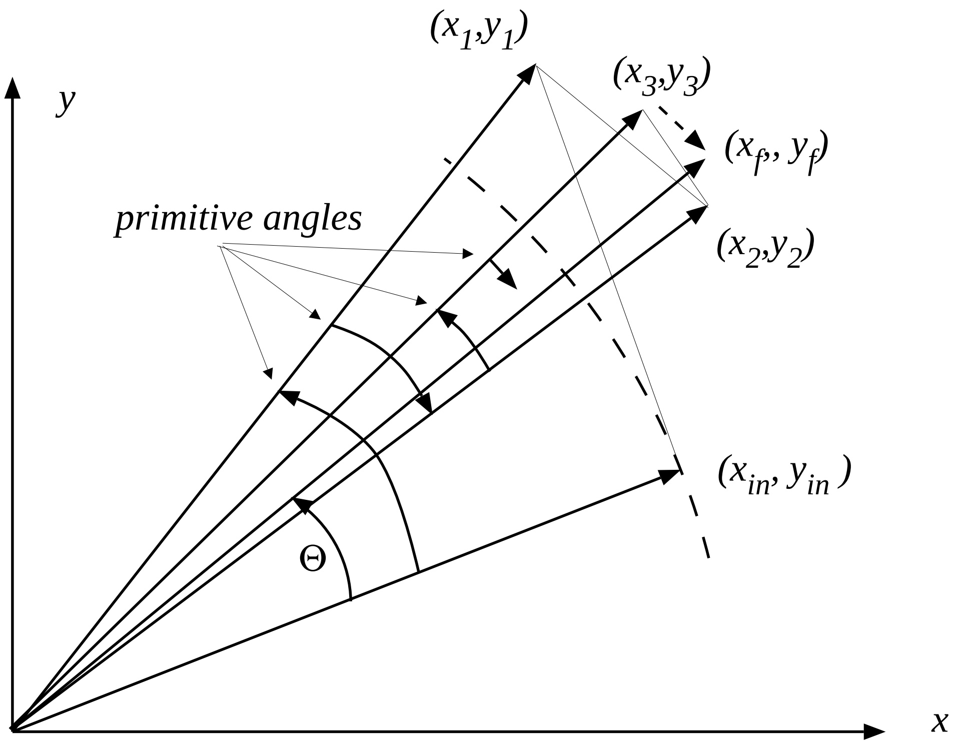


Figure 11.3: Rotating a vector using microrotations.

EXAMPLE OF ROTATION

ROTATE (x_{in}, y_{in}) BY 67° USING $n = 12$ MICRO-ROTATIONS

INITIAL COORDINATES: $x_{in} = 1$, $y_{in} = 0.125$

FINAL COORDINATES: $x_R = 0.2756$, $y_R = 0.9693$

j	$z[j]$	σ_j	$x[j]$	$y[j]$
0	1.1693	1	1.0	0.125
1	0.3839	1	0.875	1.125
2	-0.0796	-1	0.3125	1.1562
3	0.1653	1	0.7031	1.4843
4	0.0409	1	0.5175	1.5722
5	-0.0214	-1	0.4193	1.6046
6	0.0097	1	0.4694	1.5915
7	-0.0058	-1	0.4445	1.5988
8	0.0019	1	0.4570	1.5953
9	-0.0019	-1	0.4508	1.5971
10	0.0000	1	0.4539	1.5962
11	-0.0009	-1	0.4524	1.5967
12	-0.0004	-1	0.4531	1.5965
13			0.4535	1.5963

EXAMPLE 11.1 (cont.)

- AFTER COMPENSATION OF SCALING FACTOR $K = 1.64676$ COORDINATES ARE $x[13]/K = 0.2753$ and $y[13]/K = 0.9693$

- ERRORS $< 2^{-12}$

SPECIAL CASES

- TO COMPUTE $\cos\theta$ AND $\sin\theta$

  MAKE INITIAL CONDITION $x[0] = 1/K$ AND $y[0] = 0$

- IN GENERAL, FOR a AND b CONSTANTS

  $a\cos\theta - b\sin\theta$

  $a\sin\theta + b\cos\theta$

  COMPUTED BY SETTING $x[0] = a/K$ AND $y[0] = b/K$

VECTORING MODE

- ROTATE INITIAL VECTOR (x_{in}, y_{in}) UNTIL $y = 0$
- FOR INITIAL VECTOR IN THE FIRST QUADRANT:

$$\sigma_j = \begin{cases} 1 & \text{if } y[j] < 0 \\ -1 & \text{if } y[j] \geq 0 \end{cases}$$

- ACCUMULATE ROTATION ANGLE IN z
- FOR $x[0] = x_{in}$, $y[0] = y_{in}$ and $z[0] = z_{in}$, THE FINAL VALUES ARE

$$\begin{aligned} x_f &= K(x_{in}^2 + y_{in}^2)^{1/2} \\ y_f &= 0 \\ z_f &= z_{in} + \tan^{-1}(\frac{y_{in}}{x_{in}}) \end{aligned}$$

EXAMPLE OF VECTORING

- INITIAL VECTOR $(x_{in} = 0.75,\ y_{in} = 0.43)$

- y FORCED TO ZERO IN $n = 12$ MICRO-ROTATIONS

- ROTATED VECTOR: $x_R = \sqrt{x_{in}^2 + y_{in}^2} = 0.8645$, $y_R = 0.0$

- ROTATED ANGLE $z_f = \tan^{-1}(\frac{0.43}{0.75}) = 0.5205$

j	$y[j]$	σ_j	$x[j]$	$z[j]$
0	0.43	-1	0.75	0.0
1	-0.32	1	1.18	0.7853
2	0.27	-1	1.34	0.3217
3	-0.065	1	1.4075	0.5667
4	0.1109	-1	1.4156	0.4423
5	0.0224	-1	1.4225	0.5047
6	-0.0219	1	1.4232	0.5360
7	0.0002	-1	1.4236	0.5204
8	-0.0108	1	1.4236	0.5282
9	-0.0053	1	1.4236	0.5243
10	-0.0025	1	1.4236	0.5223
11	-0.0011	1	1.4236	0.5213
12	-0.0004	1	1.4236	0.5208
13			1.4236	0.5206

EXAMPLE 11.2 (cont.)

- ACCUMULATED ANGLE $z[13] = 0.5206$

- AFTER PERFORMING COMPENSATION OF $K = 1.64676$, $x[13]/K = 0.864$

- ERRORS $< 2^{-12}$

CONVERGENCE, PRECISION, AND RANGE

- ROTATION MODE
- CONVERGENCE

$$|z[i]| \leq \sum_{j=i}^{\infty} \tan^{-1}(2^{-j})$$

$$\theta_{max} = z[0]_{max} = \sum_{j=0}^{\infty} \tan^{-1}(2^{-j}) \approx 1.7429 \; (99.88^{o})$$

FOR THIS ANGLE ALL $\sigma_j = 1$ and $z[j] > 0$.

- CONSIDER $\theta < \theta_{max}$

$$|z[i]| \leq \tan^{-1}(2^{-(i-1)})$$

- CONSEQUENTLY

$$\tan^{-1}(2^{-i-1}) \leq \sum_{j=i}^{\infty} \tan^{-1}(2^{-j})$$

OR

$$\tan^{-1}(2^{-i}) \leq \sum_{j=i+1}^{\infty} \tan^{-1}(2^{-j})$$

SATISFIED FOR ALL i

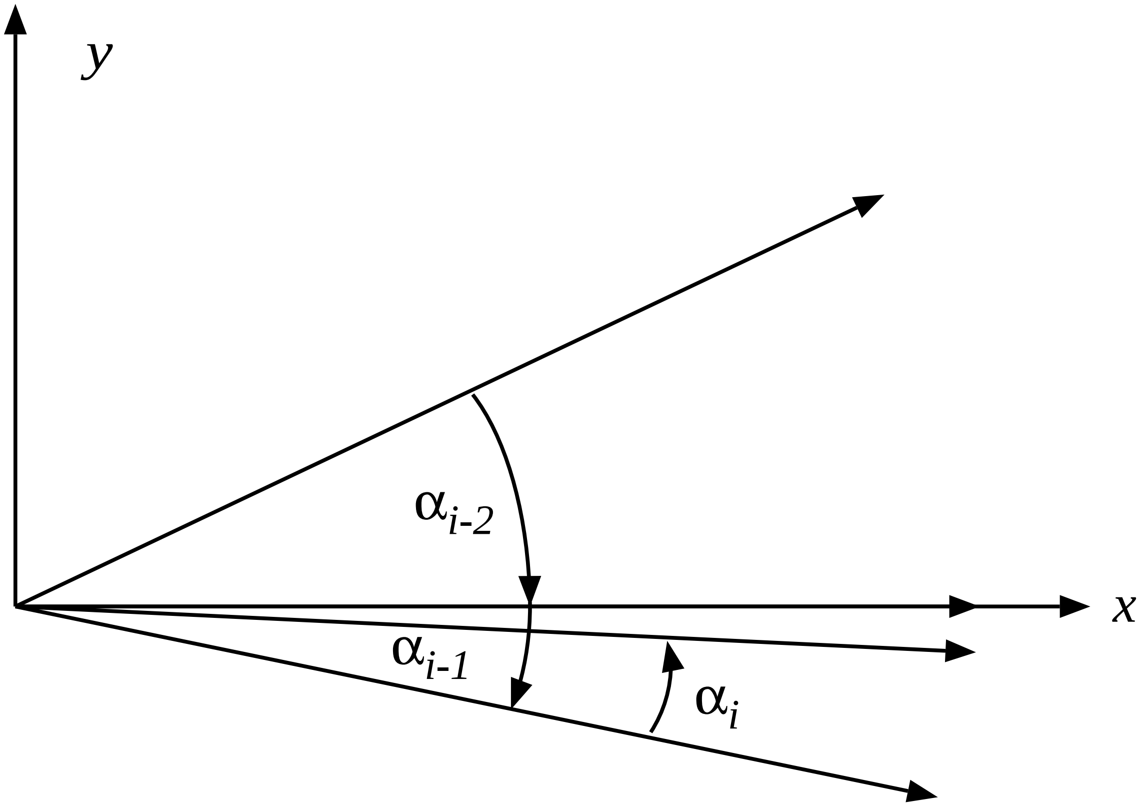


Figure 11.4: CONVERGENCE CONDITION: THE MAXIMUM NEGATIVE CASE.

PRECISION AND RANGE FOR n ITERATIONS

- n ITERATIONS (FINITE SEQUENCE)
- RESIDUAL ANGLE AFTER n ITERATIONS $z[n]$

$$|z[n]| \leq \tan^{-1}(2^{-(n-1)})$$

$$2^{-n} < \tan^{-1}(2^{-(n-1)}) < 2^{-(n-1)}$$

- THE MAXIMUM ANGLE FOR CONVERGENCE

$$\theta_{max} = \sum_{i=0}^{n-1} \tan^{-1}(2^{-j}) + 2^{-n+1}$$

- 2^{-n+1} THE MAXIMUM RESIDUAL ANGLE

COMPENSATION OF SCALING FACTOR

- MOST DIRECT METHOD: MULTIPLY BY $1/K$
- USE SCALING ITERATIONS OF THE FORM (1 ± 2^{-i})

$$x_s = x \pm x(2^{-i})$$

- USE REPETITIONS OF CORDIC ITERATIONS

$$|z[i+1]| \leq \tan^{-1}(2^{-i})$$

- OPTIMIZATION: FIND THE MINIMUM NUMBER OF SCALING ITERATIONS PLUS REPETITIONS SO THAT THE SCALE FACTOR IS COMPENSATED.

Table 11.4: Scale factor compensation for $n = 24$

Scaling iterations	(-1)(+2)(-5)(+10)(+16)(+19)(+22)
Scalings + repetitions	(-2)(+16)(+17) 1,3,5,6

IMPLEMENTATIONS

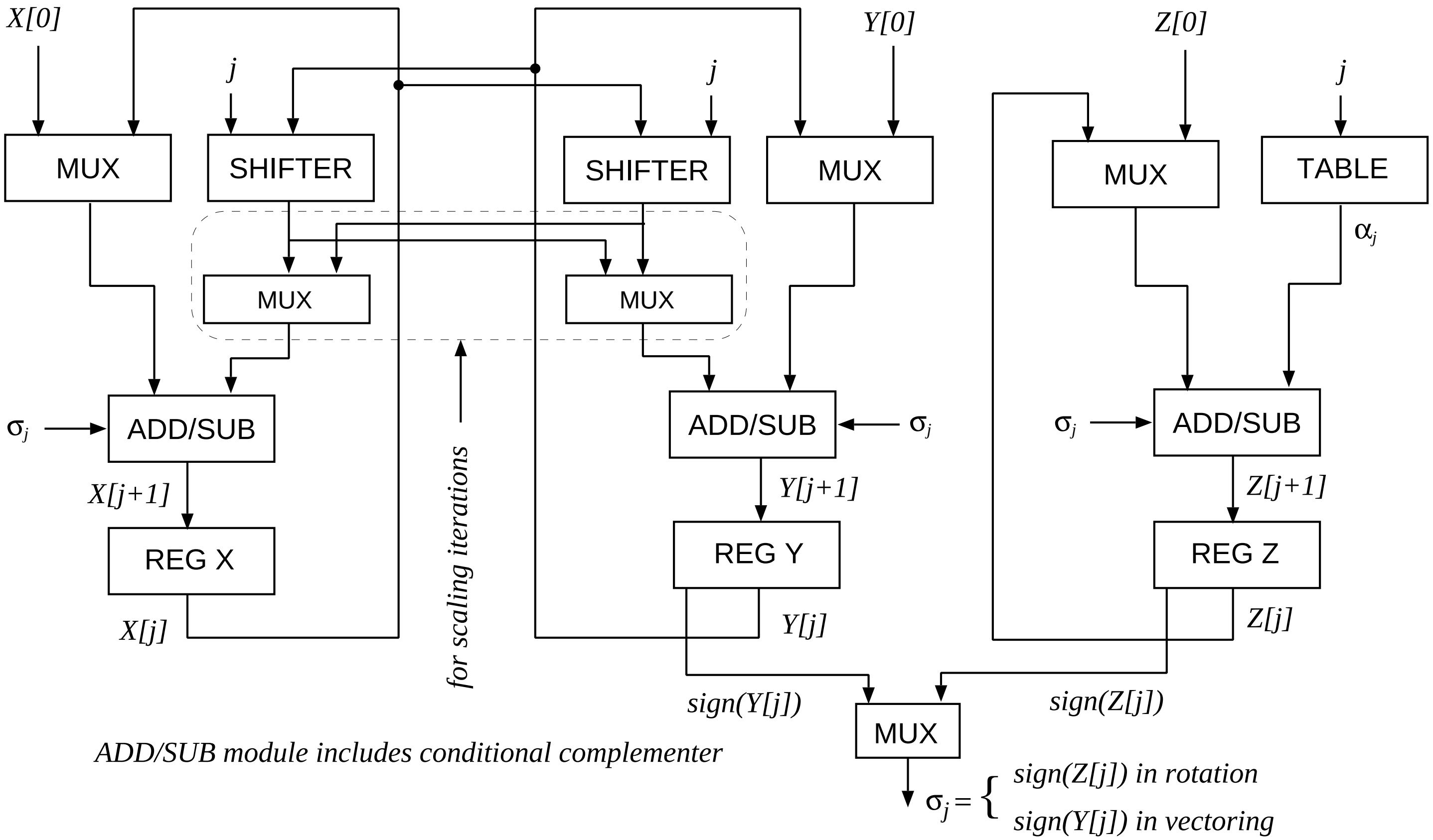
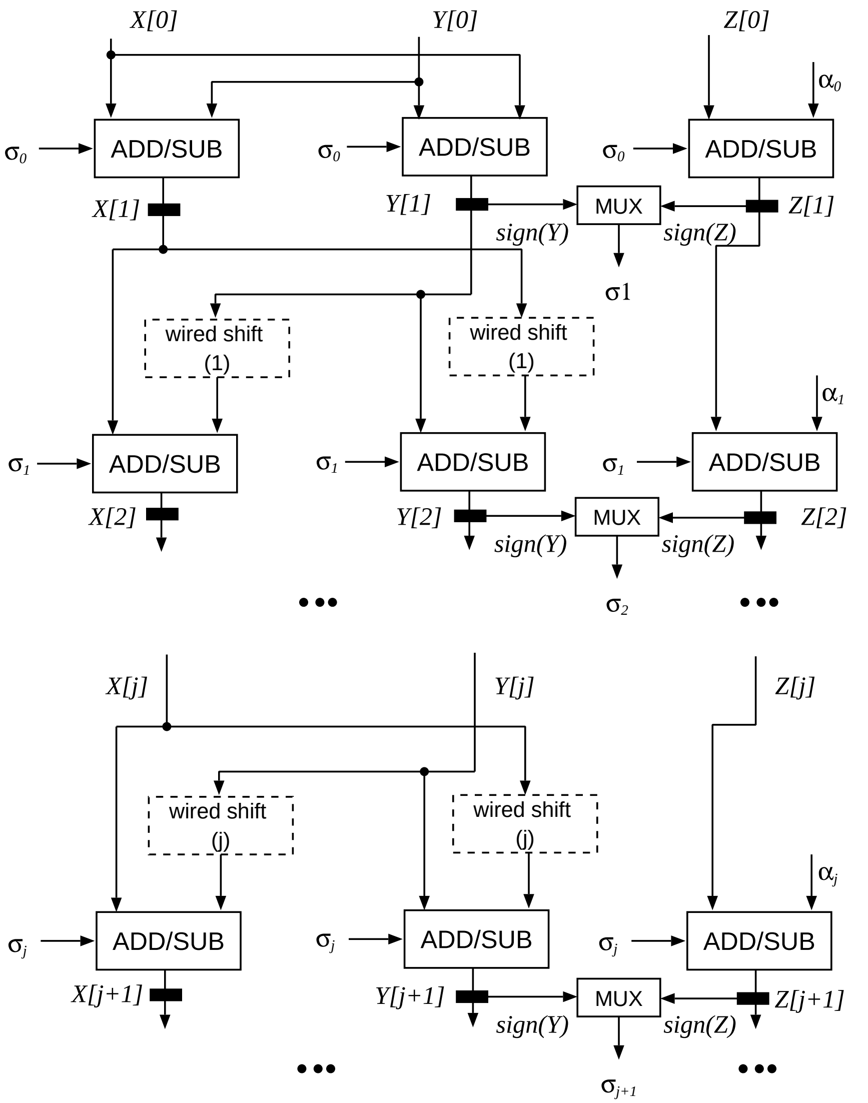


Figure 11.5: WORD-SERIAL IMPLEMENTATION.



ADD/SUB module includes conditional complementer

$$\sigma_j = \begin{cases} sign(Z[j]) \text{ in rotation} \\ sign(Y[j]) \text{ in vectoring} \end{cases}$$

Figure 11.6: PIPELINED IMPLEMENTATION.

EXTENSION TO HYPERBOLIC AND LINEAR COORDINATES

- HYPERBOLIC COORDINATES

$$\begin{bmatrix} x_R \\ y_R \end{bmatrix} = \begin{bmatrix} \cosh\theta & \sinh\theta \\ \sinh\theta & \cosh\theta \end{bmatrix} \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix}$$

- CORDIC HYPERBOLIC MICROROTATION:

$$\begin{aligned} x[j+1] &= x[j] + \sigma_j 2^{-j} y[j] \\ y[j+1] &= y[j] + \sigma_j 2^{-j} x[j] \\ z[j+1] &= z[j] - \sigma_j \tanh^{-1}(2^{-j}) \end{aligned}$$

- SCALING FACTOR IN ITERATION j

$$K_h[j] = (1 - 2^{-2j})^{1/2}$$

- $\tanh^{-1} 2^0 = \infty$ (and $K_h[0] = 0$) $\Longrightarrow$NECESSARY TO BEGIN FROM ITERATION $j = 1$

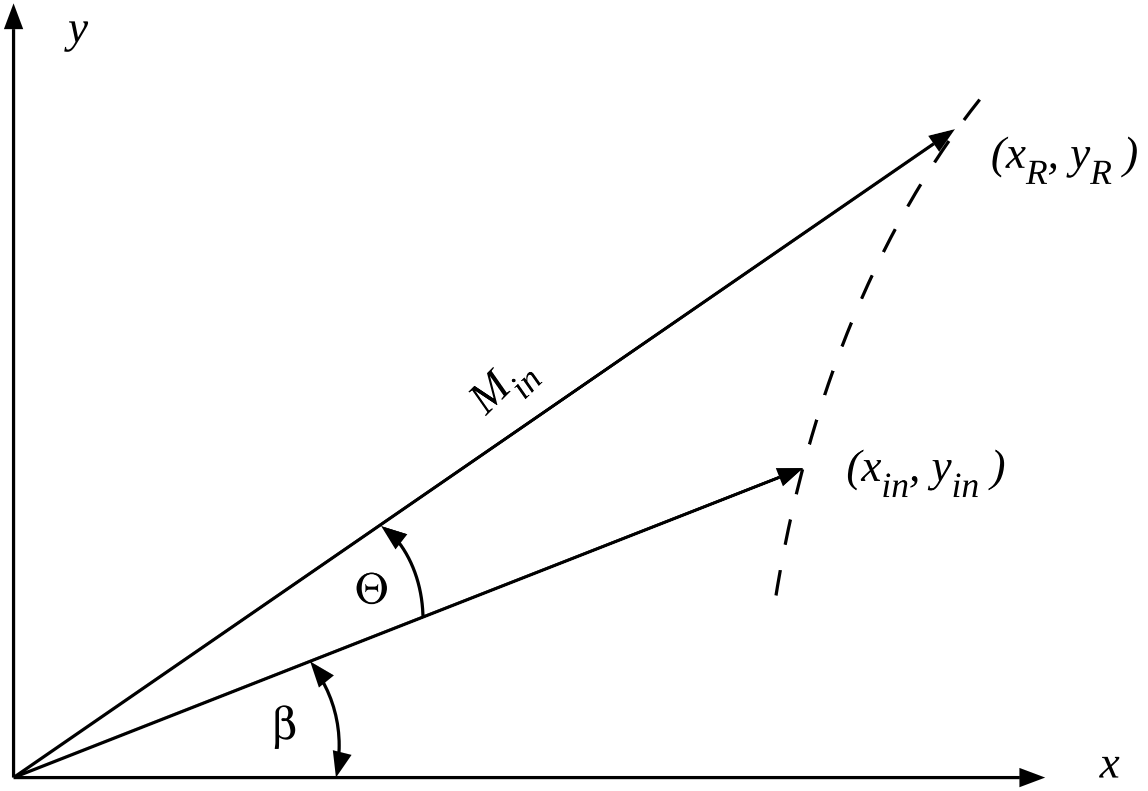


Figure 11.7: ROTATION IN HYPERBOLIC COORDINATE SYSTEM.

CONVERGENCE PROBLEM

- DOES NOT CONVERGE WITH SEQUENCE OF ANGLES $\tanh^{-1}(2^{-j})$ SINCE

$$\sum_{j=i+1}^{\infty} \tanh^{-1}(2^{-j}) < \tanh^{-1}(2^{-i})$$

- A SOLUTION: REPEAT SOME ITERATIONS

$$\sum_{i=j+1}^{\infty} \tanh^{-1}(2^{-i}) < \tanh^{-1}(2^{-j}) < \sum_{i=j+1}^{\infty} \tanh^{-1}(2^{-i}) + \tanh^{-1}(2^{-(3j+1)})$$

$\Longrightarrow$REPEATING ITERATIONS $4, 13, 40, ..., k, 3k + 1, ...$ RESULTS IN A CONVERGENT ALGORITHM.

- WITH THESE REPETITIONS

$$K_h \approx 0.82816$$

$$\theta_{max} = 1.11817$$

HYPERBOLIC ROTATION AND VECTORING

FINAL VALUES:

- FOR ROTATION MODE

$$\begin{aligned} x_f &= K_h(x_{in}\cosh\theta + y_{in}\sinh\theta) \\ y_f &= K_h(x_{in}\sinh\theta + y_{in}\cosh\theta) \\ z_f &= 0 \end{aligned}$$

- FOR VECTORING MODE

$$\begin{aligned} x_f &= K_h(x_{in}^2 - y_{in}^2)^{1/2} \\ y_f &= 0 \\ z_f &= z_{in} + \tanh^{-1}(\frac{y_{in}}{x_{in}}) \end{aligned}$$

LINEAR COORDINATES

$$\begin{aligned} x_R &= x_{in} \\ y_R &= y_{in} + x_{in} z_{in} \end{aligned}$$

$$\begin{aligned} x[j+1] &= x[j] \\ y[j+1] &= y[j] + \sigma_j 2^{-j} x[j] \\ z[j+1] &= z[j] - \sigma_j (2^{-j}) \end{aligned}$$

THE SCALING FACTOR IS 1.
FOR THE VECTORING MODE THE FINAL VALUES

$$\begin{aligned} x_f &= x_{in} \\ z_f &= z_{in} + \frac{y_{in}}{x_{in}} \end{aligned}$$

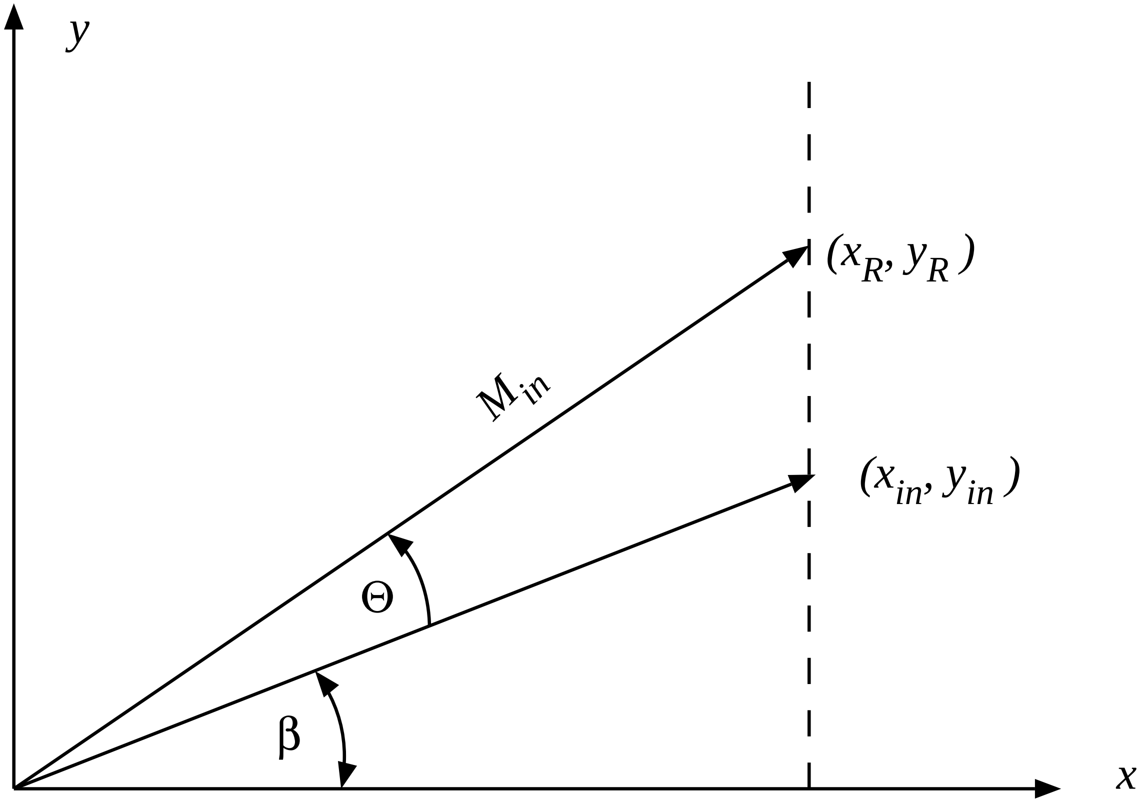


Figure 11.8: ROTATION IN LINEAR COORDINATE SYSTEM.

UNIFIED DESCRIPTION

- $m = 1$ FOR CIRCULAR COORDINATES
- $m = -1$ FOR HYPERBOLIC COORDINATES
- $m = 0$ FOR LINEAR COORDINATES
- UNIFIED MICROROTATION IS

$$\begin{aligned}
x[j+1] &= x[j] - m\sigma_j 2^{-j} y[j] \\
y[j+1] &= y[j] + \sigma_j 2^{-j} x[j] \\
z[j+1] &= \begin{cases} z[j] - \sigma_j \tan^{-1}(2^{-j}) & \text{if } m = 1 \\ z[j] - \sigma_j \tanh^{-1}(2^{-j}) & \text{if } m = -1 \\ z[j] - \sigma_j (2^{-j}) & \text{if } m = 0 \end{cases}
\end{aligned}$$

ALSO $z[j+1] = z[j] - \sigma_j m^{-1/2} \tan^{-1}(m^{1/2} 2^{-j})$

- THE SCALING FACTOR IS

$$K_m[j] = (1 + m2^{-2j})^{1/2}$$

Table 11.5: UNIFIED CORDIC

Coordinates	Rotation mode $\sigma_j = sign(z[j])^+$	Vectoring mode $\sigma_j = -sign(y[j])^+$
Circular $(m = 1)$ $\alpha_j = \tan^{-1}(2^{-j})$ initial $j = 0$ $j = 0, 1, 2, .., n$ $K_1 \approx 1.64676$ $\theta_{max} \approx 1.74329$	$x_f = K_1(x_{in} \cos(z_{in}) - y_{in} \sin(z_{in}))$ $y_f = K_1(x_{in} \sin(z_{in}) + y_{in} \cos(z_{in}))$ $z_f = 0$	$x_f = K_1(x_{in}^2 + y_{in}^2)^{1/2}$ $y_f = 0$ $z_f = z_{in} + \tan^{-1}(\frac{y_{in}}{x_{in}})$
Linear $(m = 0)$ $\alpha_j = 2^{-j}$ initial $j = 0$ $j = 0, 1, , 2, ..., n$ $K_0 = 1$ $\theta_{max} = 2 - 2^{-n}$	$x_f = x_{in}$ $y_f = y_{in} + x_{in} z_{in}$ $z_f = 0$	$x_f = x_{in}$ $y_f = 0$ $z_f = z_{in} + \frac{y_{in}}{x_{in}}$
Hyperbolic $(m = -1)$ $\alpha_j = \tanh^{-1}(2^{-j})$ initial $j = 1$ $j = 1, 2, 3, 4, 4, 5...13, 13, ...$ $K_{-1} \approx 0.82816$ $\theta_{max} \approx 1.11817$	$x_f = K_{-1}(x_{in} \cosh(z_{in}) + y_{in} \sinh(z_{in}))$ $y_f = K_{-1}(x_{in} \sinh(z_{in}) + y_{in} \cosh(z_{in}))$ $z_f = 0$	$x_f = K_{-1}(x_{in}^2 - y_{in}^2)^{1/2}$ $y_f = 0$ $z_f = z_{in} + \tanh^{-1}(\frac{y_{in}}{x_{in}})$

$^+$ $sign(a) = 1\ if\ a \geq 0, \quad sign(a) = -1\ if\ a < 0.$

OTHER FUNCTIONS

Table 11.6: SOME ADDITIONAL FUNCTIONS

m	Mode	Initial values			Functions	
		x_{in}	y_{in}	z_{in}	x_R	y_R or z_R
1	rotation	1	0	θ	$\cos\theta$	$y_R = \sin\theta$
-1	rotation	1	0	θ	$\cosh\theta$	$y_R = \sinh\theta$
-1	rotation	a	a	θ	ae^{θ}	$y_R = ae^{\theta}$
1	vectoring	1	a	$\pi/2$	$\sqrt{a^2+1}$	$z_R = \cot^{-1}(a)$
-1	vectoring	a	1	0	$\sqrt{a^2-1}$	$z_R = \coth^{-1}(a)$
-1	vectoring	$a+1$	$a-1$	0	$2\sqrt{a}$	$z_R = 0.5\ln(a)$
-1	vectoring	$a+\frac{1}{4}$	$a-\frac{1}{4}$	0	$\sqrt{a}$	$z_R = \ln(\frac{1}{4}a)$
-1	vectoring	$a+b$	$a-b$	0	$2\sqrt{ab}$	$z_R = 0.5\ln(\frac{a}{b})$

Note: the final values x_R and y_R are obtained after compensation of the scale factor.

REDUNDANT REPRESENTATION

- CRITICAL PATH of CORDIC ITERATION: ADDER (CPA)
- TO REDUCE IT: USE OF REDUNDANT ADDER
- PROBLEM WITH SIGN DETECTION:
  - If $\sigma \in \{-1, 1\}$, must convert to conventional - NO GOOD
  - If $\sigma \in \{-1, 0, 1\}$, can use estimate in selection

    $\Rightarrow$ SCALING FACTOR NO LONGER CONSTANT
- TWO APPROACHES FOR $\sigma \in \{-1, 0, 1\}$
  1. CALCULATE VARIABLE SCALING FACTOR AND PERFORM COMPENSATION
  2. DOUBLE-ROTATION APPROACH
- TWO APPROACHES FOR $\sigma \in \{-1, 1\}$
  1. USE ADDITIONAL ITERATIONS (Correcting iterations)
  2. USE 2 CORDIC MODULES (Plus/Minus)

DOUBLE ROTATION APPROACH

- σ_j is $\{-1, 0, 1\}$
- To maintain the constant scale factor, perform a double rotation
  - $\sigma_j = 1$. Both rotations are by angle $\tan^{-1}(2^{-(j+1)})$
  - $\sigma = 0$. The two rotations are by the angles $\tan^{-1}(2^{-(j+1)})$ and $-\tan^{-1}(2^{-(j+1)})$.
  - $\sigma_j = -1$. Both rotations are by the angle $-\tan^{-1}(2^{-(j+1)})$.
- Consequently, the scaling factor is constant and has value

$$K = \prod_{j=1}^{n}(1 + 2^{-2j})$$

- The elementary are $\alpha_j = 2\tan^{-1}(2^{-(j+1)})$

RECURRENCES FOR DOUBLE ROTATION

$$\begin{aligned}
x[j+1] &= x[j] - q_j 2^{-j} y[j] - p_j 2^{-2j-2} x[j] \\
y[j+1] &= y[j] + q_j 2^{-j} x[j] - p_j 2^{-2j-2} y[j] \\
z[j+1] &= z[j] - q_j (2 \tan^{-1}(2^{-(j+1)}))
\end{aligned}$$

- Two control variables $(q_j,\ p_j)$: (1,1) for $\sigma_j = 1$; (0,-1) for $\sigma_j = 0$; and (-1,1) for $\sigma_j = -1$
- The value of σ_j determined from an estimate of variable ($z[j]$ for rotation and $y[j]$ for vectoring)
- since the variable converges to 0, the estimate of the sign uses the bits $j-1$, j, and $j+1$.
- Advantage: uses a redundant representation and produces a constant scaling factor
- Disadvantage: the recurrence requires three terms instead of two

DIGITAL ARITHMETIC

Miloš D. Ercegovac and Tomás Lang

Morgan Kaufmann Publishers, an imprint of Elsevier Science, ©2004

COMMENTS AND ERRATA

Updated: September 25, 2004

Chapter 1

page 36, line 14 and 16: replace $\tilde{w}[j]$ with $\tilde{w}[j+1]$

page 39, lines 8, 10, and -4: Replace q_{n-1-j} with q_{n-j}

page 39, in Algorithm NRD, move **endfor** before Step 4.

page 39, line 17: replace "fractions" with "integers"

page 45: Exercise 1.22: replace "right" with "left"

page 46, line -5: replace "Hennessay" with "Hennessy"

Chapter 2

page 64, line -1: after "... buffers required" insert "for"

page 65, line 4: replace "(2.21)" with "(2.22)"

page 70, line 1 in Section 2.5.2: (2.30) should be (2.31)

Comment on delay calculation for Carry-lookahead adder (CLA): Because of the implementation of the CLG module shown in Figure 2.14, in delay expressions 2.43, 2.48, and 2.52 the term t_{clg} corresponds to the delay between module input c_0 and module output c_i, $i \neq 0$.

page 82, line -11: Change "The number of cells is the same as for the basic scheme." to "The number of cells is reduced by two compared to the basic scheme."

page 115: Exercise 2.2: replace "Table 2.4" with "Table 2.2"

Chapter 3

page 155, in Figure 3.14, line 13: replace "d xxxx" with "d xxx" (only 3 x's)

Chapter 4

page 183, line 1: insert ”multiplying” between ”for” and ”magnitudes”.

page 186, Figure 4.3: eliminate ”Stage 3” in the top left corner.

page 193, line -9: replace ”page 286” with ”page 198”

page 196, Figure 4.9(b,c), column 4, row 4: replace x_l with x_1

page 216, expression (4.41): put parentheses around superscripts in the rightmost term.

Comment on Figure 4.26 (page 220): The figure is generic: it does not imply that for a 8-bit result $k = 6$. The references cited on pages 236-237 should be consulted for details about determining truncation precision.

page 227, line -7: replace with ”For the design exercises use the circuit data from Table 2.4 and Figure 5.4. Note also that the delays in Figure 5.4 are given in t_{NAND2} units, whereas those in Table 2.4 are in nanoseconds. It might be best to report the results of the exercises in t_{NAND2} units.

page 229, in Exercise 4.8, replace ”Exercise 4.7” with ”Figure 4.7”

Chapter 5

page 264, line 2: replace ”$ps + pc > 0$” with ”$ps + pc < 0$”

page 264: in ”Radix-4 division algorithm with the residual in carry-save form” there are some additional differences with respect to the radix-2 algorithm in Figure 5.6 which should be considered. Namely,

a) In the recurrence step, the number of iterations is $N = \left\lceil \frac{n+2+1}{2} \right\rceil$ (because of the initialization step and the guard bit) where n is the number of bits of the operands.

b) In the termination step, instead of the radix-2 expression

$$q = 2\left(CONVERT\left(Q\left[n+1\right], q_{n+2} - 1\right)\right)$$

use the corresponding radix-4 expression

$$q = 4\left(CONVERT\left(Q\left[N-1\right], q_{N} - 1\right)\right)$$

and instead of the expression

$$q = 2\left(CONVERT\left(Q\left[n+1\right], q_{n+2}\right)\right)$$

use

$$q = 4\left(CONVERT\left(Q\left[N-1\right], q_{N}\right)\right)$$

page 266, Figure 5.6: The fact that $\hat{y}$ is represented by three integer bits and one fractional bit, does not imply that its range is [-4, 3.5]. The range of $\hat{y}$ in the selection function is obtained from expression (5.102) and determined by expression (5.107) for radix-2 division with carry-save adder.

page 270, Figure 5.9, line 14: the least significant bit of $4WC[2]$ should be 0, eliminate *

page 270, Figure 5.9, line 16: the least significant bit of $w[3]$ should be 0

page 278: line -4, replace "$0 \leq d \leq r^n - 1$" with "$0 < d \leq r^n - 1$"

page 278: In Table 5.8 footnote replace "Correction" by "Termination step"

page 278: In Section 5.4 there is ambiguity because of the use of r (radix) for two different purposes:

- On page 278 lines -5 and -4, the r refers to the radix in the representation of the operands. Usually, this radix will be 2. This also corresponds to the r in expression 5.46, 5.47, and 5.49.
- On page 279, line after expression 5.45, the $r = 2^k$ refers to the radix of the quotient-digit, as produced by the division algorithm. That, is for example in a radix-4 division algorithm, this radix would be 4.

To avoid this ambiguity, the caption of Figure 5.15 should say $n = 8$ bits, instead of $n = 4$ (radix-4 digits), since the operands are in radix 2.

page 279: Expression 5.44: replace the term "$... = 2^m \lfloor x/d^* \rfloor$" with

$$... = \lfloor 2^m \times (x/d^*) \rfloor$$

page 279: replace expression 5.48 with

$$N = \lceil (m+v)/k \rceil$$

That is, the +1 is incorrect and the k is missing. The argument for eliminating the +1 is based on the fact that the quotient obtained by a fractional division algorithm is $1/2 \leq q < 2$, as indicated in item 2 of the same page.

page 280: Example 5.1: replace the expression for N with

$$N = \lceil (m+v)/2 \rceil = 4$$

page 282: expression 5.53 should be $|w[j]| \leq \rho d$.

page 297: Figure 5.25, number the m_k constants beginning from $m_2(0)$ to $m_2(7)$.

page 311: Exercise 5.8: line -11: Remove sentence "Show all details.".

page 312: Exercise 5.15: line -9: Replace sentence "Draw the corresponding P-D diagrams (first quadrant only)." with "Draw the corresponding P-D diagrams (give the portion for $k = 6$, first quadrant only)."

page 313: Exercise 5.17: line 8: Expression $q_{j+1} = integer\,(rw\,[j] + 0.5)$ is valid if $w\,[j]$ is expressed in two's complement. When considering a sign and magnitude representation for the residuals, the expression has to be replaced by $q_{j+1} = round\,(rw\,[j])$.

page 313: Exercise 5.17: line 14: Replace "a fast radix-2 division algorithm." with "other low radix division algorithms.".

Chapter 6

page 352: line 15: replace sentence "$\max(L_k(I_i))$ j is positive" with "$\max(L_k(I_i))$ the term depending on j is positive".

page 358: Exercise 6.5, line 16: the expression for t_{cycle} is

$$t_{cycle} = t_{SEL_{SQRT}} + t_{buff} + t_{mux} + t_{HA} + t_{reg} = 4 + 1 + 1 + 1 + 2 = 9t_g$$

page 359: Exercise 6.8, line 6: replace sentence "Perform the integer division algorithm for radix 4 with residual in carry-save representation for $x = 53$ and $d = 9$." with "Perform the integer square root algorithm for radix 4 with residual in carry-save representation for $x = 53$."

Chapter 7

page 371, expression (7.11): replace $P[j]$ with $R[j]$

page 382, line 11: replace $R[j]$ with $S[j]$

page 388: Exercise 7.6: a and b are defined in Exercise 7.5.

Chapter 8

page 407, line 9: replace "are not representable in the floating-point system" with "do not correspond to real numbers"

page 420, footnote No. 14: replace "bised" with "biased"

page 427, line 9: replace "put" with "plus".

page 434, Paragraph 3, line 4: replace "significants" with "significands".

page 428: In Figure 8.8 the Exponent Update module should have also an input to update when the significand is shifted after the adder.

Chapter 9

pages 499 and 501, Figure 9.6 and 9.7: Replace "Shift-Reg WC" with "Reg WC"; replace "Shift-Reg WS" with "Reg WS"; replace "2w[j] with "w[j]"

page 510, line 6: eliminate one "the"

page 516, Table 9.4, row 2: replace 6 with 3. Add to caption: The initial number of bits/operand is $log_2 r \times \delta$.

page 535: In Exercise 9.3 the initial conditions are $x[-1] = y[-1] = w[-1] = 0$. In the illustration of the input sequence, replace $x[j]$ and $y[j]$ with x_j and y_j, respectively.

Chapter 11

page 620: The sequence of scaling iterations is incorrect. A suitable sequence is (-1)(+2)(-5)(+8)(-10)(+15)(-17)(-19). Note that the scaling (-1) corresponds to a multiplication by 2^{-1}, so no scaling iteration is required.

The sequence of scalings plus repetitions is correct. However, it assumes that the CORDIC iterations begin at $j = 1$. This is acceptable because the repetitions make the convergence range as large as that without repetitions beginning at $j = 0$.

page 627: Table 11.4. For clarity and consistency the initial values x_i, y_i, and z_i should be denoted with x_{in}, y_{in}, and z_{in}, respectively.

page 628: row 7, last column of Table 11.5 should read: $z_R = 0.5 \ln(4a)$

page 629, line 16: replace "hight" by "high"

page 629, last line: add "(See Exercise 11.4)"

page 630, replace lines -6 to -3 by: It has been shown that when m digits are used for the estimation of the sign, the distance between repetitions is $m - 2$ iterations for the rotation mode and $m - 5$ iterations for the vectoring mode.

More specifically, in iteration j the following digits are inspected:

- For rotation inspect digits with weights from $2^{-(j-1)}$ to $2^{-(j+m-2)}$.
- For vectoring inspect digits with weights from $2^{-(j-2)}$ to $2^{-(j+m-3)}$.

page 635: Exercise 11.1. Interpret "a precision of seven bits" as "perform the minimum number of iterations required to reduce the angle to 0 with the given data-path width". With respect to part c) we have not found a systematic solution.

page 636: Exercise 11.2. Perform the minimum number of iterations required to reduce the angle to 0 with the given datapath width. With respect to part c) we have not found a systematic solution.

page 636: Exercise 11.3. Perform the minimum number of iterations to reduce y to 0 with the given datapath width. With respect to part c) we have not found a systematic solution.

page 636: Exercise 11.4. The sequence α_i should be a decreasing sequence. That is, the relation between α_i and α_{i+1} should be

$$\alpha_{i+1} < \alpha_i \leq 2\alpha_{i+1}$$

page 637: Exercise 11.15 According to the Errata for page 630 replace "Use a selection function with an estimate of the sign with two digits..." by "... with four digits..."

page 637: Exercise 11.16 According to the Errata for page 630 use an estimate of seven digits.

DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier, ©2004

Chapter 1: Solutions to Exercises

Exercise 1.1

(a)

1. 9 bits since $2^8 \leq 297 \leq 2^9$
2. 3 radix-8 digits since $8^2 \leq 297 \leq 8^3$
3. 3 radix-17 digits since $17^2 \leq 297 \leq 17^3$
4. The weights are 120, 24, 6, 2, and 1. To represent 297, 5 mixed-radix digits are needed: $2 \times 120 + 2 \times 24 + 1 \times 6 + 1 \times 2 + 1 \times 1 = 297$

(b)

1. $x_{max} = 2^9 - 1 = 511$
2. $x_{max} = 8^3 - 1 = 511$
3. $x_{max} = 17^3 - 1 = 4912$
4. $x_{max} = 5 \times 120 + 4 \times 24 + 3 \times 6 + 2 \times 2 + 1 \times 1 = 719$

(c)

1. Binary representation uses 9 bits; $E = 1$
2. Radix-8 digits represented in binary with 3 bits per digit. Digit-vector: $3 \times 3 = 9$ bits; $E = 9/(3 \times 3) = 1$
3. Radix-17 digits represented in binary with 5 bits. Digit-vector: $3 \times 5 = 15$ bits; $E = 9/(3 \times 5) = 0.6$
4. The digit sets for the mixed-radix representation and their lengths in binary representation of digits are

d_0	1,0	1
d_1	2,1,0	2
d_2	3,2,1,0	2
d_3	4,3,2,1,0	3
d_4	5,4,3,2,1,0	3

Digit-vector: 3+3+2+2+1 = 11; $E = 9/11 = 0.82$

Exercise 1.2

X_{RNS} - digit-vector in RNS representation;

$X_{RNS-bin}$ - bit-vector of X_{RNS};

x	X_{RNS}	$X_{RNS-bin}$
0	(0 0 0 0)	(000 000 00 0)
13	(6 3 1 1)	(110 011 01 1)
15	(1 0 0 1)	(001 000 00 1)
19	(5 4 1 1)	(101 100 01 1)
22	(1 2 1 0)	(001 010 01 0)
127	(1 2 1 1)	(001 010 01 1)

To compute the efficiency need to determine the number of bits for the binary representation. This number depends on the range of integers represented; we consider two situations:

i) The largest integer is 127. In such a case, the number of bits is 7 and the efficiency is

$$E = n_{r2}/n_{RNS-bin} = 7/9$$

ii) The largest integer is the maximum allowed by the moduli of the RNS representation. This value is 7x5x3x2-1= 209. Consequently, 8 bits are needed for the radix-2 representation, resulting in

$$E = 8/9$$

Exercise 1.3

If the moduli are not relatively prime, different values may have the same representation. For example, if P = (4,2), $x = 3$ and $x = 7$ have the same RNS digit-vector (3,1).

Exercise 1.4

1. $1 \leq x \leq 2^{8+8} - 1$, $E = 1$
2. $1 \leq x \leq 10^4 - 1$, $E = (10^4 - 1)/(2^{16} - 1) = 0.152$
3. $1 \leq x \leq 16^4 - 1 = 2^{16} - 1$, $E = 1$

Exercise 1.5

(a) Representation values

r	x_R
2	43
8	$8^5 + 8^3 + 8 + 1 = 33289$
10	$10^5 + 10^3 + 10 + 1 = 101,011$
16	$16^5 + 16^3 + 16 + 1 = 1,052,689$

(b) Largest values for $n = 6$

r	x_{Rmax}
2	63
10	$10^6 - 1$
16	$16^6 - 1$

Exercise 1.6

x	$C = 16$	$C = 15$	$C = 19$	$C = 127$
6	0110	0110	00110	0000110
5	0101	0101	00101	0000101
4	0100	0100	00100	0000100
3	0011	0011	00011	0000011
2	0010	0010	00010	0000010
1	0001	0001	00001	0000001
0	0000	0000	00000	0000000
-0	-	1111	10011	1111111
-1	1111	1110	10010	1111110
-2	1110	1101	10001	1111101
-3	1101	1100	10000	1111100
-4	1100	1011	01111	1111011
-5	1011	1010	01110	1111010
-6	1010	1001	01101	1111001

Exercise 1.7

(a) For $r = 2$, $x_R = 11_{10}$. For $r = 7$, $x_R = 351_{10}$. For $r = 16$, $x_R = 4113_{10}$.

(b) For $r = 2$, $x_R = 11$; for 2's complement, $C = 16$; since $x_R > C/2$ we have $x < 0$ and $x = 11 - 16 = -5$.

For $r = 4$, $x_R = 69$; for 1s' complement, $C = 4^4 - 1 = 255$; since $x_R < C/2$, we have $x > 0$ and $x = 69$.

For $r = 8$, $x_R = 521$; for 1s' complement, $C = 8^4 - 1 = 4095$, since $x_R < C/2$, we have $x > 0$ and $x = 521$.

Exercise 1.8

	Value x	Value x_R	Digit vector X
(a)	-39_{10}	4057_{10}	333121_4
(b)	-41_{10}	215_{10}	11010111
(c)	-3_{10}	29_{10}	11101

Exercise 1.9

Number system	Radix r	No. of Digits n	Value x	Value x_R	Digit-vector X
SM	10	4	-837	-837	1837
2's compl.	2	6	-10	54	110110
RC	3	4	-37	44	1122_3
RC	8	3	-149	363	551_8
1s' compl.	2	8	-83	172	10101100
2's compl.	2	7	-19/64	1+45/64	1.101101
DC	8	4	-681	3415	6527_8
1s' compl.	2	7	-19/64	1+44/64	1.101100

Exercise 1.10

NRS	x_{max}	X_{max}	x_{min}	X_{min}
SM	3+15/16	011.1111	-(3+15/16)	111.1111
2's	3+15/16	011.1111	-4	100.0000
1s'	3+15/16	011.1111	-(3+15/16)	100.0000

Exercise 1.11

NRS	integer	fraction
SM	-5	-5/16
2's	-11	-11/16
1s'	-10	-10/16

Exercise 1.12

(a) In the integer case, 2's complement, $x = -5$. Extending to $n = 6$ produces $X_{int-2} = (1, 1, 1, 0, 1, 1)$.

In the 1s' complement system, $x = -4$, and the 6-bit vector is $X_{int-1} = (1, 1, 1, 0, 1, 1)$.

Note that in the case of integers, the extended bit-vectors are the same for 2's complement and for 1s' complement.

(b) We suppose that ”Do not change the position of the radix point” means that the extended value should also be a fraction (having only the ”sign bit” as integer bit).

In the two’s complement fraction case $x = -5/8$. Extending to $n = 6$ produces $X_{frac-2} = (1, 0, 1, 1, 0, 0)$.

In the 1s’ complement fraction case $x = -4/8$ and the extended bit-vector is $X_{frac-1} = (1, 0, 1, 1, 1, 1)$.

Note that in the fraction case the extended bit-vectors are different.

Exercise 1.13
Sign-and-magnitude

- $x + y$.

  Since $x < 0$, we complement x (2’s complement) and add

  ```
  101110
  001001
       1
  ------
  111000
  ```

  The result is negative (sgn=1). We complement to obtain magnitude 00111+1=01000.

- $y - x$.

  Change sign of x and add. Both operands of addition are positive. Sign of result sgn=0.

  ```
  001001
  010001
  -----
  011010
  ```

- $x - y$.

  Change sign of y and add. Both operands of addition are negative. Consequently, add magnitudes and sign of result is sgn=1.

  ```
  010001
  001001
  ------
  011010
  ```

- $-x - y$. This is $-(x + y)$. So, perform $(x + y)$ and change sign. Result is sgn=0 and magnitude 01000.

- $|x - y|$. Perform $x - y$ and make sgn=0. The magnitude is 11010.

2's complement and 1s' complement
Consider the following table:

Operation	2's Complement	1s' Complement
x	101111	101110
y	001001	001001
	111000	110111
c_{in}/e-a-c	0	0
$x+y$	111000	110111
y	001001	001001
$\bar{x}$	010000	010001
c_{in}/e-a-c	1	0
$y-x$	011010	011010
x	101111	101110
$\bar{y}$	110110	110110
c_{in}/e-a-c	1	1
$x-y$	100110	100101
$\bar{x}$	010000	010001
$\bar{y}$	110110	110110
c_{in}/e-a-c	1	1
	000111	001000
	1	-
$-x-y$	001000	001000
x	101111	101110
$\bar{y}$	110110	110110
c_{in}/e-a-c	1	1
$x-y$	100110	100101
$\overline{x-y}$	011001	011010
c_{in}/e-a-c	1	-
$\|x-y\|$	011010	011010

Exercise 1.14

The effective operation to compute $z = |x|-|y|$ in the 2's complement system as a function of the signs of the operands is shown in the following table:

x	y	$\|x\|-\|y\|$
+	+	$x-y$
+	-	$x+y$
-	+	$-(x+y)$
-	-	$-x+y$

The algorithm is

```
**case of** (sign(x), sign(y)):
(0,0): $z = ADD(x, \bar{y}, 1)$;
```

(0,1): $z = ADD(x, y, 0)$;

(1,0): $z = ADD(\underline{0}, \overline{(ADD(x, y, 0))}, 1)$;

(1,1): $z = ADD(\bar{x}, y, 1)$;

Exercise 1.15

As discussed in this chapter, the change of sign operation in the 2's complement system is performed as

$$z_R = (2^n - 1 - x_R) + 1$$

which corresponds to inverting each bit and adding 1. Let

$$X_b = (X_k, X_{k-1}, \ldots, X_0) = (1, 0, \ldots, 0)$$

and

$$X_a = (X_{n-1}, \ldots, X_{k+1})$$

.

1. After bit-inverting X_b and X_a we get

$$\begin{array}{rcl} \overline{X_b} & = & (0, 1, \ldots, 1) \\ \overline{X_a} & = & (X'_{n-1}, \ldots, X'_{k+1}) \end{array}$$

2. After adding 1, $\overline{X_b}$ is reverted to X_b, while $\overline{X_a}$ remains unaffected.

Since the algorithm produces X_b and $\overline{X_a}$, it performs the change of sign operation.

Exercise 1.16

(a) We show two proofs: in the first we consider all possible cases and in the second we manipulate the expressions.

First proof:

x_{n-1}	y_{n-1}	s_{n-1}	c_{n-1}	c_n	overflow?
0	0	0	0	0	n
0	0	1	1	0	y
0	1	0	1	1	n
0	1	1	0	0	n
1	1	0	0	1	y
1	1	1	1	1	n

Second proof:

The overflow in addition may only happen if the operands are of the same sign, i.e., $x_{n-1} \oplus y_{n-1} = 0$ and, consequently, in this situation

$$s_{n-1} = x_{n-1} \oplus y_{n-1} \oplus c_{n-1} = c_{n-1}$$

On the other hand,

$$\begin{aligned} c_n \oplus c_{n-1} &= (x_{n-1}y_{n-1} + x_{n-1}c_{n-1} + y_{n-1}c_{n-1}) \oplus c_{n-1} \\ &= x_{n-1}y_{n-1}c'_{n-1} + x'_{n-1}y'_{n-1}c_{n-1} \\ &= x_{n-1}y_{n-1}s'_{n-1} + x'_{n-1}y'_{n-1}s_{n-1} \end{aligned}$$

which is the expression for overflow.

(b) The overflow detection using c_n and c_{n-1} does not work in the 1s' complement system since $(-0) + (-2^{n-1} + 1)$ produces $c_n = 1$ and $c_{n-1} = 0$ indicating an overflow which does not exist. For example,

$x = -3 = 100$, $y = -0 = 111$

x	100	
y	111	
	1011	$c_n = 1$, $c_{n-1} = 0$, $c_n \oplus c_{n-1} = 1$ Overflow
s	100	No overflow

Exercise 1.17

(a) 1. Signed integers

NRS	Range
SM	$[-(2^{15}-1), 2^{15}-1]$
2's	$[-2^{15}, 2^{15}-1]$
1s'	$[-(2^{15}-1), 2^{15}-1]$

2. Unsigned integers: $[0, 2^{16}-1]$

(b) 1. With 2's complement adder and flags:

Case	Adder	Z	SGN	$C0$	OVF
unsigned add	yes	yes	no	yes	no
unsigned sub	yes	yes	no	yes	no

2. With 1s' complement adder and flags:

Case	Adder	Z	SGN	$C0$	OVF
unsigned add	no	no	no	yes	no
unsigned sub	no	no	no	yes	no

(c) We consider here only the case for 2's complement representation for signed integers. The case for the other two representations can be determined in a similar manner.

For the comparison of A and B we perform $A - B$ and set the flags. The three conditions are determined as follows:

- For signed integers in 2's complement representation:
  Equal $Z = 1$
  SMALLER $(OVF = 0$ AND $NEG = 1)$ OR $(OVF = 1$ AND $NEG = 0)$ (no overflow and negative or overflow and not negative)
  GREATER $(OVF = 0$ AND $NEG = 0)$ OR $(OVF = 1$ AND $NEG = 1)$ AND $Z = 0$ (not smaller and not zero)
- For unsigned:
  Equal $Z = 1$
  For the other cases we need to consider the effect of converting the second operand to 2's complement and adding. So the operation $A - B$ is performed as

$$D = A + (2^{16} - B) = 2^{16} + (A - B)$$

  Consequently, the flag CO is set when $A - B \geq 0$. So,
  GREATER $(CO = 1$ AND $Z = 0)$
  SMALLER $CO = 0$

From these expression we see that only the branch on equal can be the same for both signed and unsigned integers.

Exercise 1.18

(a) Integers a and b represented by A and B:

C	a	b
10^4	-2638	3216
$10^4 - 1$	-2637	3216

(b) Extended to six digits:

$$A = (9, 9, 7, 3, 6, 2), \quad B = (0, 0, 3, 2, 1, 6)$$

(c) $d = 10a$, $e = a/10$ (integer), with seven digits

$$D = (9, 9, 7, 3, 6, 2, 0), \quad E = (9, 9, 9, 9, 7, 3, 6)$$

Exercise 1.19

For $x \geq 0$ we have that $z_R = x_R$. Consequently, since $X_{n-1} = 0$, the algorithm is correct. For $x < 0$, $z_R = C_z - |x|$ and $x_R = C_x - |x|$ where C_z and C_x are the corresponding complementation constants. Consequently,

$$z_R = C_z - C_x + x_R \tag{1}$$

Since for both the 2's and 1s' complement systems

$$C_z - C_x = 2^m - 2^n \tag{2}$$

we obtain

$$z_R = 2^m - 2^n + x_R \tag{3}$$

But $2^m - 2^n$ is represented by the vector

$$(1, 1, ..., 1, 0, 0, ..., 0)$$

Consequently,

$$Z = (1, 1, ..., 1, X_{n-1}, \dots, X_0) \tag{4}$$

which corresponds to the given algorithm.

Exercise 1.20

Left shift. By definition $z = 2x$. i) If $x \geq 0$ the representation is the same as in the sign-and-magnitude system and, therefore, the same algorithm holds.

ii) If $x \leq 0$ then $x = x_R - C$ and $z = z_R - C$. Therefore, $z_R - C = 2(x_R - C)$ and $z_R = 2x_R - C$. Moreover, since $x \leq 0$ we have $X_{n-1} = 1$ and

$$2x_R = 2 \cdot 1 \cdot 2^{n-1} + 2X_{n-2}2^{n-2} + \ldots + 2X_0$$

In the 2's complement system, since $C = 2^n$ we obtain

$$\begin{array}{rcl} z_R & = & 2x_R - 2^n \\ & = & 2 \cdot 1 \cdot 2^{n-1} + 2X_{n-2}2^{n-2} + \ldots + 2X_0 - 2^n \\ & = & X_{n-2}2^{n-1} + X_{n-3}2^{n-2} + \ldots + X_0 2 + 0 \cdot 2^0 \end{array}$$

From the last expression we infer the corresponding left-shift algorithm for the 2's complement system. Note that overflow occurs when $X_{n-2} \neq X_{n-1}$.

In the 1s' complement system $C = 2^n - 1$ so that

$$\begin{array}{rcl} z_R & = & 2x_R - (2^n - 1) \\ & = & 2x_R - 2^n + 1 \end{array}$$

Using the expression for $2x_R$ developed in the previous proof,

$$z_R = X_{n-2}2^{n-1} + X_{n-3}2^{n-2} + \ldots + X_0 2 + 1 \tag{5}$$

This corresponds to the indicated algorithm.

Right shift. By definition $z = 2^{-1}x - \epsilon$. If $x \geq 0$, the same algorithm as in the sign-and-magnitude case holds.

If $x \leq 0$ then $z_R - C = 2^{-1}(x_R - C) - \epsilon$ and $z_R = 2^{-1}(x_R - C) + C - \epsilon$.

For the 2's complement system $C = 2^n$, so

$$2^{-1}(x_R - C) = -2^{n-1} + X_{n-1}2^{n-2} + \ldots + X_1 2^0 + X_0 2^{-1} \tag{6}$$

and

$$\begin{array}{rcl} z_R & = & 2^n - 2^{n-1} + X_{n-1}2^{n-2} + \ldots + X_1 + X_0 2^{-1} - \epsilon \\ & = & 2^{n-1} + X_{n-1}2^{n-2} + \ldots + X_1 + X_0 2^{-1} - \epsilon \end{array} \tag{7}$$

Assuming $\epsilon = X_0 2^{-1}$ (this satisfies $|\epsilon| < 1$), we obtain the corresponding algorithm.

In the 1s' complement system $C = 2^n - 1$, so that

$$2^{-1}(x_R - C) = -2^{n-1} + X_{n-1}2^{n-2} + \ldots + X_1 2^0 + (X_0 + 1)2^{-1} \qquad (8)$$

and

$$z_R = 2^n - 2^{n-1} + X_{n-1}2^{n-2} + \ldots + X_1 2^0 + (X_0 + 1)2^{-1} - 1 - \epsilon \qquad (9)$$

Assuming now $\epsilon = 1 - (X_0 + 1)2^{-1}$ the same algorithm is obtained.

Exercise 1.21

2's complement:

X	00101101	45
$SL(X)$	01011010	90
$SR(X)$	00010110	22
Y	11010110	-42
$SL(Y)$	10101100	-84
$SR(Y)$	11101011	-21

1s' complement:

X	00101101	45
$SL(X)$	01011010	90
$SR(X)$	00010110	22
Y	11010110	-41
$SL(Y)$	10101101	-82
$SR(Y)$	11101011	-20

Exercise 1.22

Overflow happens in the arithmetic shift-left if

$$X_{n-2} \neq X_{n-1}$$

This is because in this case the sign would change by the shift.

Exercise 1.23

Given

$$\begin{array}{ll} A = 1101 & (a = -3) \\ B = 110 & (b = -2) \\ C = 0101 & (c = 5) \\ D = 10101 & (d = -21) \end{array}$$

compute $z = -3 + (-2) + 8 * 5 - 2 * (-21) = -7$.

$$\begin{array}{cl} A & 1111101 \\ B & 1111110 \\ 8C & 0101000 \\ 2D & 1101010 \\ \hline z & 1111001 \end{array}$$

Exercise 1.24

The multiplication is shown in Figure E1.24.

$n = 5 \quad x = 21\ (X = 10101) \quad y = 14\ (Y = 01110)$

$p[0]$	00000	
$2^5 x Y_0$	00000	
	00000	
$p[1]$	00000	0
$2^5 x Y_1$	10101	
	10101	0
$p[2]$	01010	10
$2^5 x Y_2$	10101	
	11111	10
$p[3]$	01111	110
$2^5 x Y_3$	10101	
	100100	110
$p[4]$	010010	0110
$2^5 x Y_4$	00000	
$p[5]$	10010	0110 = 294

Figure E1.24

Exercise 1.25

(a) The multiplication for 2's complement representation is given in Fig. E1.25a.

(b) The multiplication for 1s' complement representation is in Fig. E1.25b. Note that we complement the multiplier and then complement the result.

Exercise 1.26

The execution time of the basic multiplication scheme for n-bit non-negative integers is

$$T_{basic} = (t_{vd} + t_{add} + t_{reg}) \times n$$

The execution time can be reduced by using the multiplier as a radix-4 digit-vector to about $T_{basic}/2$ as follows:

- Precompute $3X = 2X + X$ and store it in a register.
- In each iteration consider two bits of the multiplier as a radix-4 digit $z_j \in \{0, 1, 2, 3\}$. Select $0 \times X$, $1 \times X$, $2 \times X$ (left shifted X produced by wiring - no extra delay), or $3 \times X$ (precomputed using shift and add) depending on the value of z_j using a multiplexer.
- Perform $n/2$ iterations.

Since $n/2$ iterations are performed and one additional cycle is required for the precomputation of $3x$, the reduced execution time is

$n = 6 \quad x = 21\ (X = 010101) \quad y = -17\ (Y = 101111)$

$p[0]$	0 000000	
2^5xY_0	0 010101	
	0 010101	
$p[1]$	0 001010	1
2^5xY_1	0 010101	
	0 011111	1
$p[2]$	0 001111	11
2^5xY_2	0 010101	
	0 100100	11
$p[3]$	0 010010	011
2^5xY_3	0 010101	
	0 100111	011
$p[4]$	0 010011	1011
2^5xY_4	0 000000	
	0 010011	1011
$p[5]$	0 001001	11011
-2^5xY_5	1 101011	
$p[6]$	1 110100	11011 = xy = -357

Figure E1.25a 2's complement multiplication.

$n = 6 \quad x = 21\ (X = 010101) \quad y = -17\ (Y = 101110)$
$-y = 17\ (010001)$

$p[0]$	0 000000	
2^5xY_0	0 010101	
	0 010101	
$p[1]$	0 001010	1
2^5xY_1	0 000000	
	0 001010	1
$p[2]$	0 000101	01
2^5xY_2	0 000000	
	0 000101	01
$p[3]$	0 000010	101
2^5xY_3	0 000000	
	0 000010	101
$p[4]$	0 000001	0101
2^5xY_4	0 010101	
	0 010110	0101
$p[5]$	0 001011	00101
complement		
$p[6]$	1 110100	11010 = xy = -357

Figure E1.25b 1s' complement multiplication.

$$T_{reduced} = (t_{MUX} + t_{add} + t_{reg}) \times (n/2 + 1)$$

Exercise 1.27

The recurrence for the left-to-right multiplication of non-negative integers is

$$\begin{aligned} &p[0] = 0 \\ &p[j+1] = rp[j] + xY_{n-1-j} \quad j = 0, 1, \ldots, n-1 \\ &p = p[n] \end{aligned} \tag{10}$$

It can be shown by substitution that

$$p[j+1] = r^{j+1}p[0] + x \sum_{k=n-1-j}^{n-1} Y_k r^{k-(n-1-j)}$$

so that

$$p[n] = r^n p[0] + xy$$

The adder has $2n-1$ digits. The relative position of the operands in the left-to-right recurrence is shown in Figure E1.27

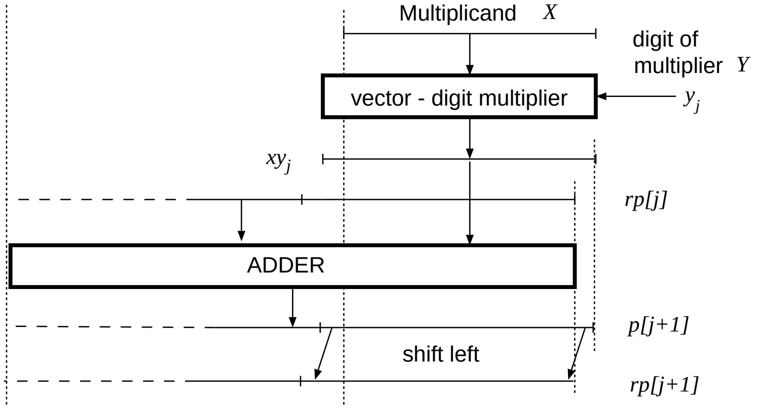


Figure E1.27: Relative position of operands in left-to-right multiplication.

Since the adder is twice as wide as in the right-to-left (basic) multiplication, the execution time is significantly increased.

Exercise 1.28

From Algorithm NRD for integer division of $2n$-bit dividend x and n-bit divisor d we have:

$d^* = d2^n \qquad w[0] = x$

For $j = 0$, $w[1] = 2w[0] - q_{n-1}d^*$
For $j = 1$, $w[2] = 2w[1] - q_{n-2}d^* = 2^2w[0] - (2q_{n-1} + q_{n-2})d^*$
For $j = n - 1$, $w[n] = 2^nw[0] - (2^{n-1}q_{n-1} + 2^{n-2}q_{n-2} + \ldots + 2q_1 + q_0)d^*$

The last scaled remainder (corrected if negative) is

$$2^{-n}w[n] = w[0] - (\sum_{j=0}^{n-1} q_j 2^j)d^*2^{-n} = x - q \cdot d$$

since $w[0] = x$ and $q = \sum_{j=0}^{n-1} q_j 2^j$. Therefore,

$$x = q \cdot d + w$$

Since the quotient-digit selection function guarantees bounded residuals $|w[j]| < d^*$, the algorithm is correct.

Exercise 1.29

Perform non-restoring integer division for the following operands.

$$\text{Dividend } x = 14_{10} = (00001110)_2, \text{ divisor } d = 3 = (0011)_2$$

$$\begin{array}{rlll}
w[0] = & 0\ 0000 & 1110 & \\
2w[0] = & 0\ 0001 & 1100 & \\
-d^* = & 1\ 1101 & & \\
\hline
w[1] = & 1\ 1110 & 1100 & q_3 = 0 \\
2w[1] = & 1\ 1101 & 1000 & \\
+d^* = & 0\ 0011 & & \\
\hline
w[2] = & 0\ 0000 & 1000 & q_2 = 1 \\
2w[2] = & 0\ 0001 & 0000 & \\
-d^* = & 1\ 1101 & & \\
\hline
w[3] = & 1\ 1110 & 0000 & q_1 = 0 \\
2w[3] = & 1\ 1100 & 0000 & \\
+d^* = & 0\ 0011 & & \\
\hline
w[4] = & 1\ 1111 & 0000 & q_0 = 0 \\
\hline
w[4] = & 0\ 0010 & & (\mathit{corrected})
\end{array}$$

Quotient $q = (0100)_2 = 4$, remainder $w = (0010)_2 = 2$. Check: $14 = 3 \times 4 + 2$.

Exercise 1.30

We consider the alternative with quotient-digit set $\{-1, +1\}$. If the divisor is signed, the quotient-digit selection depends on the sign of the divisor. To have a bounded residual, the selection function is

$$q_{n-j} = \begin{cases} 1 & \text{if } sign(w[j]) = sign(d) \\ -1 & \text{if } sign(w[j]) \neq sign(d) \end{cases}$$

We also want the quotient to be in 2's complement representation. This is accomplished by making the quotient

$$q = P + N$$

where P is the weighted sum of all digits having value 1 and N is the weighted sum of all digits with value -1. Consequently, the 2's complement representation is obtained by adding P and N (2's complement addition). For this, N (which is negative) should be represented in 2's complement.

It is also possible to do the conversion considering only P as follows. Since all bits of q are either 1 or -1 we get

$$P - N = 2^n - 1$$

and

$$(P + N) + (P - N) = 2P = q + 2^n - 1$$

so that

$$q = -2^n + 2P + 1$$

Morover, since the maximum absolute value of the quotient is $2^{n-1} - 1$ (remember the the $n - th$ bit is the "sign bit"), The two most-significant signed digits of q cannot be of the same sign. Consequently, in P the most-significant two bits are either 10 (positive quotient) or 01 (negative quotient). Therefore, when subtraction 2^n from $2P$, we get a 2's complement representation, as follows:

P=10... then $2P - 2^n = 0...$ (that is, bit $n-1$ is 0 and the result is positive)
P=01... then $2P - 2^n = 1....$(that is, bit $n-1$ is 1 and the result is negative)
This can be implemented during the iterations by

- Replacing -1's with 0's
- Shifting the resulting vector one position to the left
- Inverting the quotient bit in position $n-1$ and inserting 1 in the least-significant position. If quotient correction is needed, 0 is inserted in its least-significant position.

DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier, ©2004
– Updated: February 13, 2004 –

Chapter 2: Solutions to Selected Exercises

– with contributions by Elisardo Antelo –

Exercise 2.1

Assuming that c_i is connected to the XOR input with load factor 1.1 (Fig. 2.5(c)), the average delay of the carry-out is

$$T_1 = t_{NAND}(1) + t_{NAND}(2.1) = 0.07 + 0.033 + 0.07 + 0.033 \times 2.1 = 0.242ns$$

Adding an inverter and changing the XOR into XNOR, we obtain for the carry delay:

$$T_2 = t_{NAND}(1) + t_{NAND}(2) = 0.239ns$$

This represents a 1.4% reduction in the carry delay. Note that the difference is very small because of the XOR input with load factor 1.1. A larger reduction would result if the XOR input load factors were symmetrical at 2.

Exercise 2.4

$T_{SRA} = t_{sw} + (n-1)t_p + (n/m)t_{buf} + t_s$ (Expression (2.27))

$t_{sw} = max(t_{gi}, t_{ki}, t_{pi}) + t_{NAND-2(L=2)} = t_{pi} + t_{NAND-2(L=2)} = 0.329 + 0.136 = 0.465ns$

where, assuming a switch has one standard load,

$$t_{gi} = t_{AND-2} = 0.16 + 0.027 \times 1 = 0.187ns$$

$$t_{ki} = t_{NOR-2} = 0.07 + 0.046 \times 1 = 0.116ns$$

$$t_{pi} = t_{XOR-2} = 0.30 + 0.029 \times 1 = 0.329ns$$

$t_p = t_{NAND-2} = 0.07 + 0.033 \times 2 = 0.136ns$ $(L = 2)$
$t_{buf} = 1.5 \times 0.136 = 0.204$
$t_s = 0.46 + 0.03 \times L = 0.46ns$ (Table 2.2, delay c_i to s_i with $L = 0$)
Therefore,
$T_{SRA} = 0.465 + 31 \times 0.136 + 8 \times 0.204 + 0.46 \approx 6.8ns$

From Exercise 2.2, $T_{CRA} = 13.8ns$ so the SRA aproximately halves the delay. Note that to reduce the load the network for computing the sum bits uses separately obtained p_i signals

Exercise 2.5

Figure E2.5 shows the carry chains for the given operands.

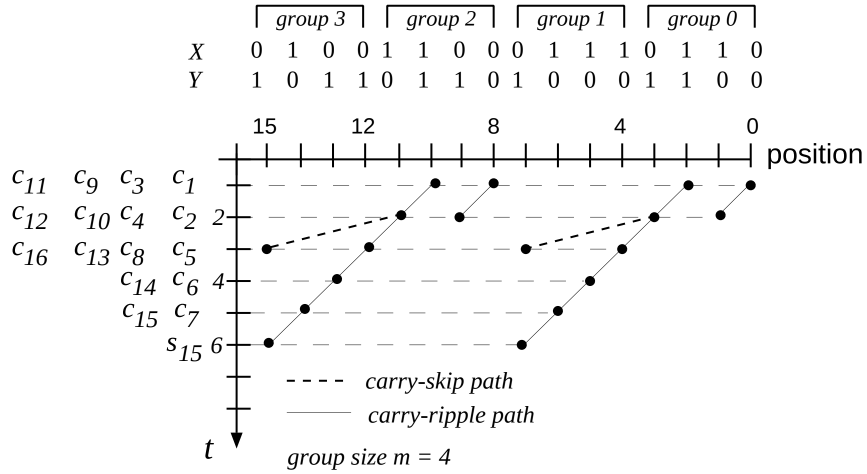

group 3
group 2
group 1
group 0
X 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 0
Y 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0
15
12
8
4
0
position
c_{11} c_9 c_3 c_1
c_{12} c_{10} c_4 c_2 2
c_{16} c_{13} c_8 c_5
c_{14} c_6 4
c_{15} c_7
s_{15} 6
t
carry-skip path
carry-ripple path
group size m = 4

Figure E2.5: Carry chains in carry-skip adder (Exercise 2.5).

Exercise 2.10

(a) $T = mt_c + (s-1)t_{mux} + (p-2)t_{mux} + (s-1)t_{mux} + (m-1)t_c + t_s$.

(b) Let $t_c = t_{mux}$ and $m = s$. $T = (4m - 3 + n/m^2)t_c + t_s$ and $m_{opt} = (n/2)^{1/3}$.

Exercise 2.13

The g_i and a_i signals are

x	0	1	0	1
y	1	0	0	1
g_i	0	0	0	1
a_i	1	1	0	1

The expressions and values for the CLG-4 carries are

$$
\begin{aligned}
c_0 &= 1 \\
c_1 &= g_0 + a_0c_0 = 1 + 1 \cdot 1 = 1 \\
c_2 &= g_1 + a_1g_0 + a_1a_0c_0 = 0 + 0 \cdot 1 + 0 \cdot 1 \cdot 1 = 0 \\
c_3 &= g_2 + a_2g_1 + a_2a_1g_0 + a_2a_1a_0c_0 = 0 + 1 \cdot 0 + 1 \cdot 0 \cdot 1 + 1 \cdot 0 \cdot 1 \cdot 1 = 0 \\
c_4 &= g_3 + a_3g_2 + a_3a_2g_1 + a_3a_2a_1g_0 + a_3a_2a_1a_0c_0 \\
&= 0 + 1 \cdot 0 + 1 \cdot 1 \cdot 0 + 1 \cdot 1 \cdot 0 \cdot 1 + 1 \cdot 1 \cdot 0 \cdot 1 \cdot 1 = 0
\end{aligned}
$$

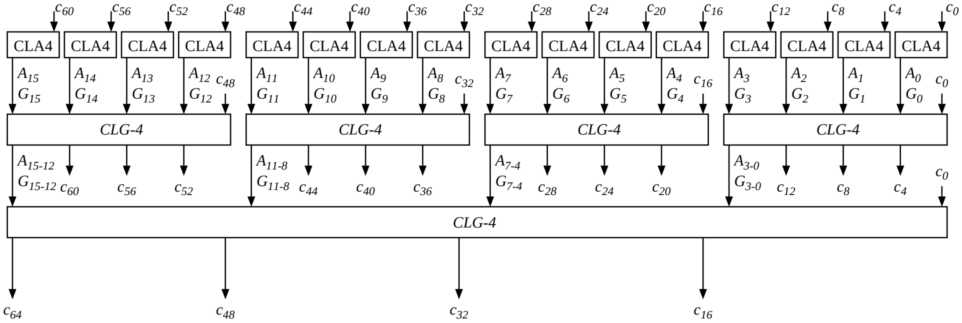
Exercise 2.15

A 64-bit, three-level carry-lookahead adder is shown in Figure E2.15.

Exercise 2.17

$$n = 128,\ m = 4,\ t_{clg} = t_{AG} = 6t_{ag} = 3t_s$$

$$T_{1-CLA} = t_{ag} + (n/m)t_{clg} + t_s = 1 + 32 \times 6 + 2 = 195t_{ag}$$



c_{60} c_{56} c_{52} c_{48} c_{44} c_{40} c_{36} c_{32} c_{28} c_{24} c_{20} c_{16} c_{12} c_8 c_4 c_0
CLA4
A_{15} G_{15} A_{14} G_{14} A_{13} G_{13} A_{12} G_{12} c_{48}
A_{11} G_{11} A_{10} G_{10} A_9 G_9 A_8 G_8 c_{32}
A_7 G_7 A_6 G_6 A_5 G_5 A_4 G_4 c_{16}
A_3 G_3 A_2 G_2 A_1 G_1 A_0 G_0 c_0
CLG-4
$A_{15\text{-}12}$ $G_{15\text{-}12}$ c_{60} c_{56} c_{52}
$A_{11\text{-}8}$ $G_{11\text{-}8}$ c_{44} c_{40} c_{36}
$A_{7\text{-}4}$ $G_{7\text{-}4}$ c_{28} c_{24} c_{20}
$A_{3\text{-}0}$ $G_{3\text{-}0}$ c_{12} c_8 c_4 c_0
CLG-4
c_{64} c_{48} c_{32} c_{16}

Figure E2.15: 64-bit three-level carry-lookahead adder.

$$T_{2-CLA} = t_{ag} + t_{AG} + (n/m^2)t_{clg} + t_{clg} + t_s = 1 + 6 + 8 \times 6 + 6 + 2 = 63 t_{ag}$$

$$T_{3-CLA} = t_{ag} + 2t_{AG} + (n/m^3)t_{clg} + 2t_{clg} + t_s = 1 + 12 + 12 + 12 + 2 = 39 t_{ag}$$

For the 4-level CLA we use another level with a group size of 2. Because of the smaller size of this group the delay of this level is smaller, we assume it to be $t_{clg2} = 2t_{a,g}$.

$$T_{4-CLA} = t_{ag} + 2t_{AG4} + t_{clg2} + 3t_{clg} + t_s = 1 + 12 + 2 + 18 + 2 = 35 t_{ag}$$

Exercise 2.20

i	8	7	6	5	4	3	2	1	0
x_i		0	1	0	1	0	1	1	1
y_i		1	1	1	0	0	1	1	1
g_i		0	1	0	0	0	1	1	1
a_i		1	1	1	1	0	1	1	1
p_i		1	0	1	1	0	0	0	0

Level 1 outputs:

$$\begin{aligned}
g_{(0,-1)} &= 1 = c_1 \\
g_{(1,0)} &= g_1 + a_1 g_0 = 1, \quad a_{(1,0)} = a_1 a_0 = 1 \\
g_{(2,1)} &= g_2 + a_2 g_1 = 1, \quad a_{(2,1)} = a_2 a_1 = 1 \\
g_{(3,2)} &= g_3 + a_3 g_2 = 0, \quad a_{(3,2)} = a_3 a_2 = 0 \\
g_{(4,3)} &= g_4 + a_4 g_3 = 0, \quad a_{(4,3)} = a_4 a_3 = 0 \\
g_{(5,4)} &= g_5 + a_5 g_4 = 0, \quad a_{(5,4)} = a_5 a_4 = 1 \\
g_{(6,5)} &= g_6 + a_6 g_5 = 1, \quad a_{(6,5)} = a_6 a_5 = 1 \\
g_{(7,6)} &= g_7 + a_7 g_6 = 1, \quad a_{(7,6)} = a_7 a_6 = 1
\end{aligned}$$

Level 2 outputs:

$$\begin{aligned}
g_{(1,-1)} &= g_{(1,0)} + a_{(1,0)}c_0 = 1 = c_2 \\
g_{(2,-1)} &= g_{(2,1)} + a_{(2,1)}g_{(0,-1)} = 1 = c_3 \\
g_{(3,0)} &= g_{(3,2)} + a_{(3,2)}g_{(1,0)} = 0, \quad a_{(3,0)} = a_{(3,2)}a_{(1,0)} = 0 \\
g_{(4,1)} &= g_{(4,3)} + a_{(4,3)}g_{(2,1)} = 0, \quad a_{(4,1)} = a_{(4,3)}a_{(2,1)} = 0 \\
g_{(5,2)} &= g_{(5,4)} + a_{(5,4)}g_{(3,2)} = 0, \quad a_{(5,2)} = a_{(5,4)}a_{(3,1)} = 0 \\
g_{(6,3)} &= g_{(6,5)} + a_{(6,5)}g_{(4,3)} = 1, \quad a_{(6,3)} = a_{(6,5)}a_{(4,3)} = 0 \\
g_{(7,4)} &= g_{(7,6)} + a_{(7,6)}g_{(5,4)} = 1, \quad a_{(7,4)} = a_{(7,6)}a_{(5,4)} = 1
\end{aligned}$$

Level 3 outputs:

$$\begin{aligned}
c_4 &= g_{(3,0)} + a_{(3,0)}c_0 = 0 \\
c_5 &= g_{(4,1)} + a_{(4,1)}g_{(0,-1)} = 0 \\
c_6 &= g_{(5,2)} + a_{(5,2)}g_{(1,-1)} = 0 \\
c_7 &= g_{(6,3)} + a_{(6,3)}g_{(2,0)} = 1
\end{aligned}$$

Level 4 outputs:

$$\begin{aligned}
s_0 &= p_0 \oplus c_0 = 1 \\
s_1 &= p_1 \oplus c_1 = 1 \\
s_2 &= p_2 \oplus c_2 = 1 \\
s_3 &= p_3 \oplus c_3 = 1 \\
s_4 &= p_4 \oplus c_4 = 1 \\
s_5 &= p_5 \oplus c_5 = 1 \\
s_6 &= p_6 \oplus c_6 = 0 \\
s_7 &= p_7 \oplus c_7 = 0 \\
c_8 &= g_{(7,0)} + a_{(7,0)}c_0 = 1
\end{aligned}$$

Exercise 2.23

A diagram of a 4-bit conditional-adder module is shown in Figure E2.23.

Exercise 2.26

X	01	01	01	11
Y	10	10	11	11
S^0	11	11	00	10
c^0	0	0	1	1
S^1	00	00	01	11
c^1	1	1	1	1
S^0	11	11	01	10
c^0	0		1	
S^1	00	00	01	11
c^1	1		1	
S^0	00	00	01	10
c^0	1			
S^1	00	00	01	11
c^1	1			

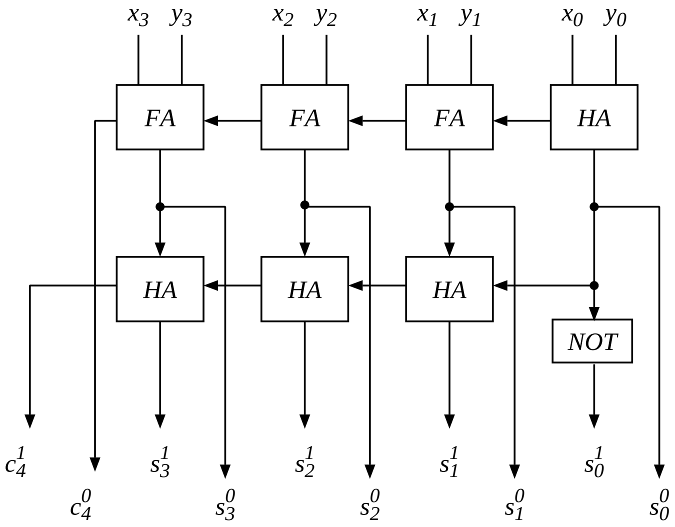


Figure E2.23: 4-bit conditional adder for Exercise 2.23.

The result is (c^0, S^0) because $c_{in} = 0$.

Exercise 2.29

a) Type 1 adder:

x	1000	100	111
y	0111	000	110
c_i^0	11111	110	011
c_i^1	00000	001	100
c_i	00000	001	100
s_i	01111	101	101

The actual delay, assuming critical path in producing F, is

$$T_{Type1} = t_{XOR} + t_{OR-2} + 10 \times t_c + t_{OR-2}$$

where t_c is the delay of producing a carry:

$$t_c = t_{AND-2} + t_{OR-2}$$

Given that t_c has the same expression for the carry-ripple adder and that the actual delay of t_c is 15% smaller than its worst-case delay and assuming the same variation for t_{XOR} and t_{OR-2}, we get:

$$T_{Type1} \approx 0.85 T_{CRA}$$

b) Type 2 adder:

x	1000100111
y	0111000110
chains	jihgfedcba
timing	6543211111

In this example, the longest chain is zero-carry chain efghij of 6 positions. The actual delay is

$$T_{Type2} = t_{XOR} + t_{max} + t_{OR-2} + t_{AND-10}$$

where $t_{max} = 6t_c$.

Consequently, including the delay of AND-10, for this input pattern the addition delay is rougly 70% of that of the adder of type I.

Exrecise 2.32

a)

X		0	1	1	1	1	1	0	0	0	0	1	1	0	0	1	1
Y		1	1	1	0	0	0	1	0	0	1	1	0	1	1	0	1
W		1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
$S*$		0	0	1	1	0	1	0	0	1	1	1	1	0	1	0	0
$C*$	1	1	1	0	1	0	1	0	0	0	1	0	1	0	1	1	0a
Z		1	0	1	0	1	1	1	1	0	1	1	1	0	1	1	1
S	1	0	1	0	0	1	1	1	1	1	1	0	1	0	1	0	1
C	1	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	

a carry in

b)

X		0	1	1	1	1	1	0	0	0	0	1	1	0	0	1	1
Y		1	1	1	0	0	0	1	0	0	1	1	0	1	1	0	1
W		1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Z		1	0	1	0	1	1	1	1	0	1	1	1	0	1	1	1
T	1	1	1	0	1	0	1	0	0	0	1	0	1	0	1	1	0a
P		1	0	0	1	1	0	1	1	1	0	0	0	0	0	1	1
S	1	0	1	0	0	1	1	1	1	1	1	0	1	0	1	0	1
C	1	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	

a carry in; P output of Odd-parity module.

Exercise 2.35

	101	110	110	011
	1	1	0	1
	011	100	111	011
	001	011	101	111
1	1	1	0	0

Exercise 2.39

Method 1:

X			0	1	$\bar{1}$	1	$\bar{1}$	0	1	$\bar{1}$
Y			1	0	1	0	1	$\bar{1}$	$\bar{1}$	1
H		1	1	0	1	0	0	0	0	
Z			$\bar{1}$	$\bar{1}$	0	$\bar{1}$	0	$\bar{1}$	0	0
Q		1	0	$\bar{1}$	1	$\bar{1}$	0	$\bar{1}$	0	0
T	0	0	$\bar{1}$	0	$\bar{1}$	0	$\bar{1}$	0	0	
W		1	0	1	1	1	0	1	0	0
S		1	$\bar{1}$	1	0	1	$\bar{1}$	1	0	0

Method 2:

X		0	1	$\bar{1}$	1	$\bar{1}$	0	1	$\bar{1}$
Y		1	0	1	0	1	$\bar{1}$	$\bar{1}$	1
P		0	0	1	0	1	1	1	1
T	1	0	0	0	0	$\bar{1}$	0	0	
W		$\bar{1}$	1	0	1	0	1	0	0
S	1	$\bar{1}$	1	0	1	$\bar{1}$	1	0	0

Exercise 2.43

Radix-2 signed digit addition of one conventional and one signed-digit operand:

X		0	1	1	1	0	1	1	0
Y^+		1	0	1	0	0	0	1	1
Y^-		0	1	0	0	0	1	0	0
W		1	0	0	1	0	0	0	1
T	1	0	1	1	0	0	1	1	
S^+	1	0	1	1	0	0	1	1	0
S^-		1	0	0	1	0	0	0	1
S	1	$\bar{1}$	1	1	$\bar{1}$	0	1	1	$\bar{1}$

DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier, ©2004

Chapter 3: Solutions to Selected Exercises

– with contributions by Elisardo Antelo –

Exercise 3.1

As explained in the text, for two's complement representation the most-significant bit of each operand is inverted and $-m$ is added, with its least-significant bit aligned with the most-significant bit of the operands. For $m = 7$ we add -7 = 1001. Moreover, to avoid an extra row, we evaluate $1001 + g_0' = 10g_0'g_0$. The resulting matrix is

$$\begin{array}{rcccc}
a_0'. & a_1 & a_2 & \dots & a_n \\
b_0'. & b_1 & b_2 & \dots & b_n \\
c_0'. & c_1 & c_2 & \dots & c_n \\
d_0'. & d_1 & d_2 & \dots & d_n \\
e_0'. & e_1 & e_2 & \dots & e_n \\
f_0'. & f_1 & f_2 & \dots & f_n \\
10g_0'g_0. & g_1 & g_2 & \dots & g_n
\end{array}$$

Exercise 3.3

A [5:2] module is shown in Figure E3.3a. and an array of these modules to reduce five 8-bit operands in Figure E3.3b.

To determine the critical path we use the following delay model, simplified from the model given in Table 2.2:

	FA		HA	
from/to	c_{out}	s	c_{out}	s
(x, y)		2	0.7	1.2
x	2			
y	1.5			
c	1	1.2	-	-

where the delay is normalized to the delay t_{c-c}.

Figure E3.3a indicates the module delays using this model. Consequently, the critical path delay is $5t_{c-c}$. The implementation uses 22 FAs and 2 HAs.

For comparison, an array of [3:2] modules to reduce 5 8-bit operands is shown in Figure 3.3c.As shown, the critical path has a delay of $5.5t_{c-c}$. The network cost is cost 22 FAs and 3 HAs. We conclude that both networks have the same cost and that the network using [5:2] modules is somewhat faster than the network using [3:2] modules.

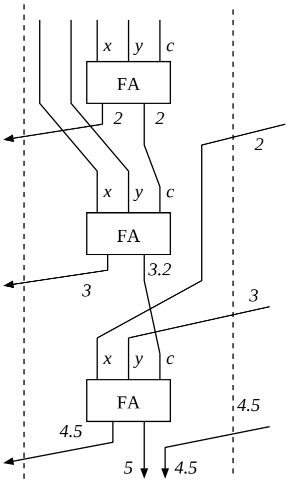


Figure E3.3a: The [5:2] module for Exercise 3.3.

Exercise 3.5

To determine the critical path we use the following delay model, simplified from the model given in Table 2.2:

	FA	
from/to	c_{out}	s
(x, y)		2
x	2	
y	1.5	
c	1	1.2

where the delay is normalized to the delay t_{c-c}.

A [9:2] module is shown in Figure E3.5. The delay in the critical path is $T = 8t_{c-c}$.

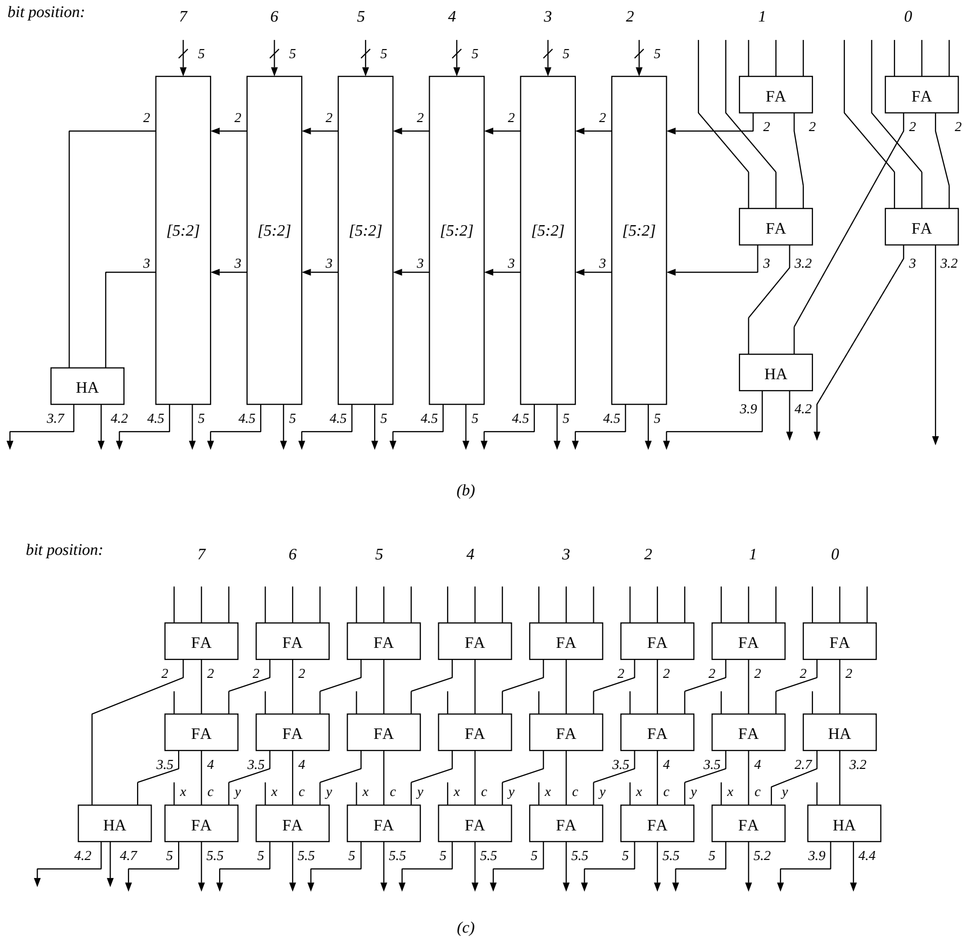


Figure E3.3: (b) Network of [5:2] modules to reduce 5 8-bit operands. (c) Network of [3:2] modules to reduce 5 8-bit operands.

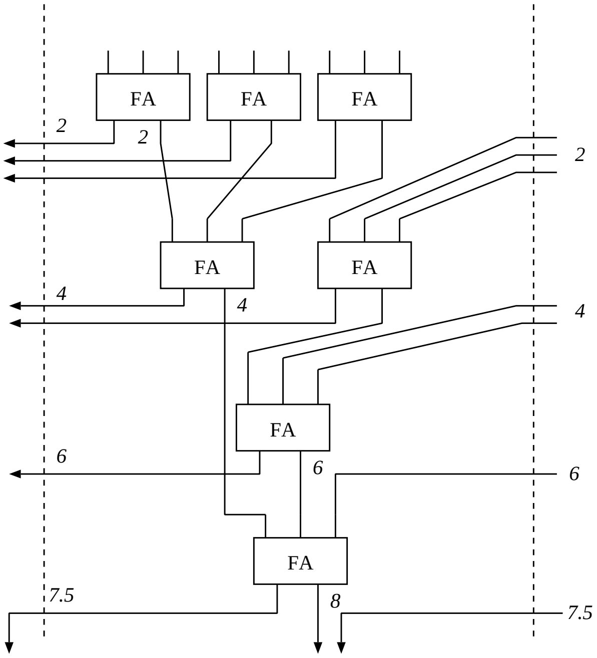


Figure E3.5: The network of FAs for Exercise 3.5.

Exercise 3.8

A network of full-adders implementing a (15:4] counter is shown in Figure E3.8.

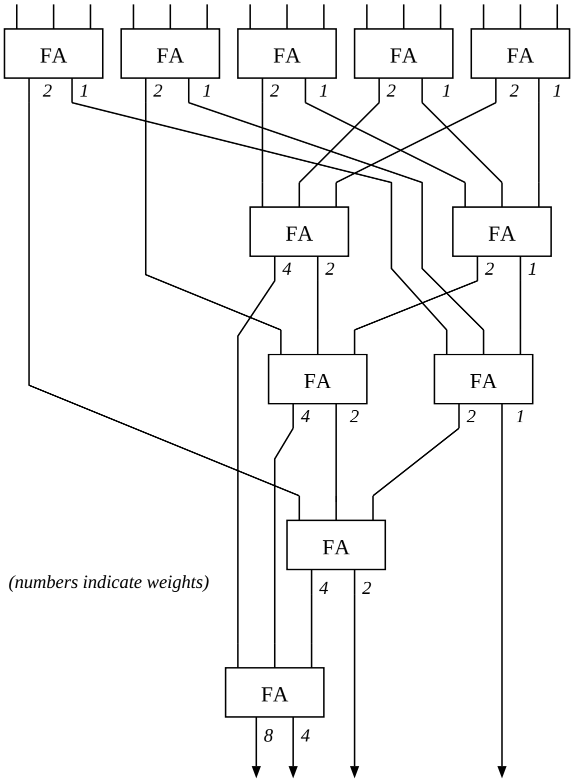


Figure E3.8: A network of FAs implementing (15:4] counter in Exercise 3.8.

Exercise 3.10

The maximum value of the sum is $S = 32 \times 127$. Since $2^{11} < S = 2^{12} - 2^5 < 2^{12}$, 12 bits are necessary.

1. The logic diagram of a bit-slice showing only CSA and registers is given in Figure E3.10(a).

2. The block diagram at the word level is shown in Figure E3.10(b).

3. The critical path delay: $t_s + t_{reg}$ where t_s is the delay of the sum output of a FA.

4. The latency: $32 \times (t_s + t_{reg}) + t_{CPA} = 32 \times (t_s + t_{reg}) + 11t_c + t_s$ where t_c is the delay of the carry output of a FA.

5. Use a CRA instead of the CSA. In this case the adder has 11 bits plus the carry-out. The critical path is $10t_c + t_s + t_{reg}$. Assume that $t_s = 2t_c$ and $t_{reg} = t_s$. Then the ratio of cycle times in the two alternatives is:

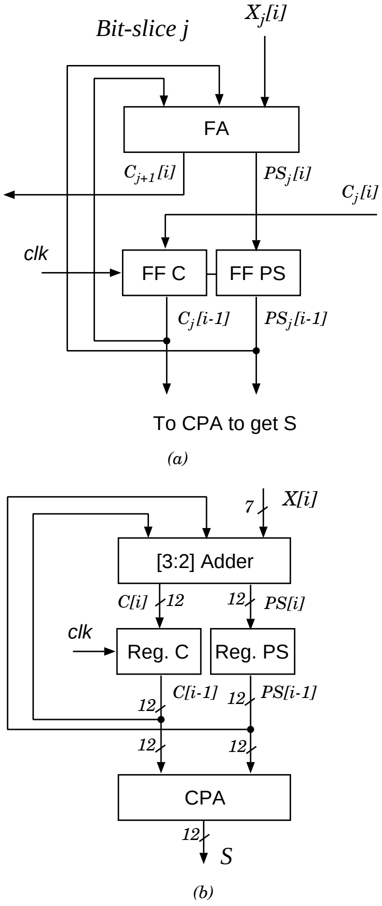


Figure E3.10: (a) Bit-slice of multi-operand adder. (b) Multi-operand adder of Exercise 3.10.

$$(10t_c + t_s + t_{reg})/(t_s + t_{reg}) = 7t_s/2t_s = 3.5$$

The latency of the alternative with CRA is $32 \times (10t_c + t_s + t_{reg})$ and the ratio of latencies is

$$(32 \times (10t_c + t_s + t_{reg})/(32 \times (t_s + t_{reg}) + 12t_c + t_s)$$

$$= (32 \times 7t_s)/(32 \times 2t_s + 6.5t_s) = 224/70.5 = 3.2$$

In terms of hardware, the alterantive with CRA uses only one register and an 11-bit adder. The alternative with CSA uses two registers and two adders. This is roughly twice as much hardware.

Exercise 3.13

To determine the critical path we use the following delay model, simplified from the model given in Table 2.2:

	FA		HA	
from/to	c_{out}	s	c_{out}	s
(x, y)		2	0.7	1.2
x	2			
y	1.5			
c	1	1.2	-	-

where the delay is normalized to the delay t_{c-c}.

The [5:2] module shown in Fig. E3.13a has a critical path of $5t_{c-c}$.

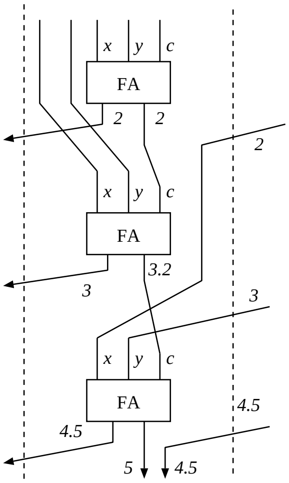


Figure E3.13a: [5:2] module.

To reduce the ten 4-bit operands we use an array of [5:2] modules (forming two adders of 5 inputs each) followed by a [4:2] adder, as shown in Figure E3.13b. The critical path delay is $8t_{c-c}$. The implementation uses 28 FAs and 6 HAs.

For comparison, Figure E3.13c shows an array of [3:2] adders to reduce 10 4-bit operands. At the full-adder level, this array is implemented as shown in Figure E3.13d. The corresponding critical path delay is $9.2t_{c-c}$.

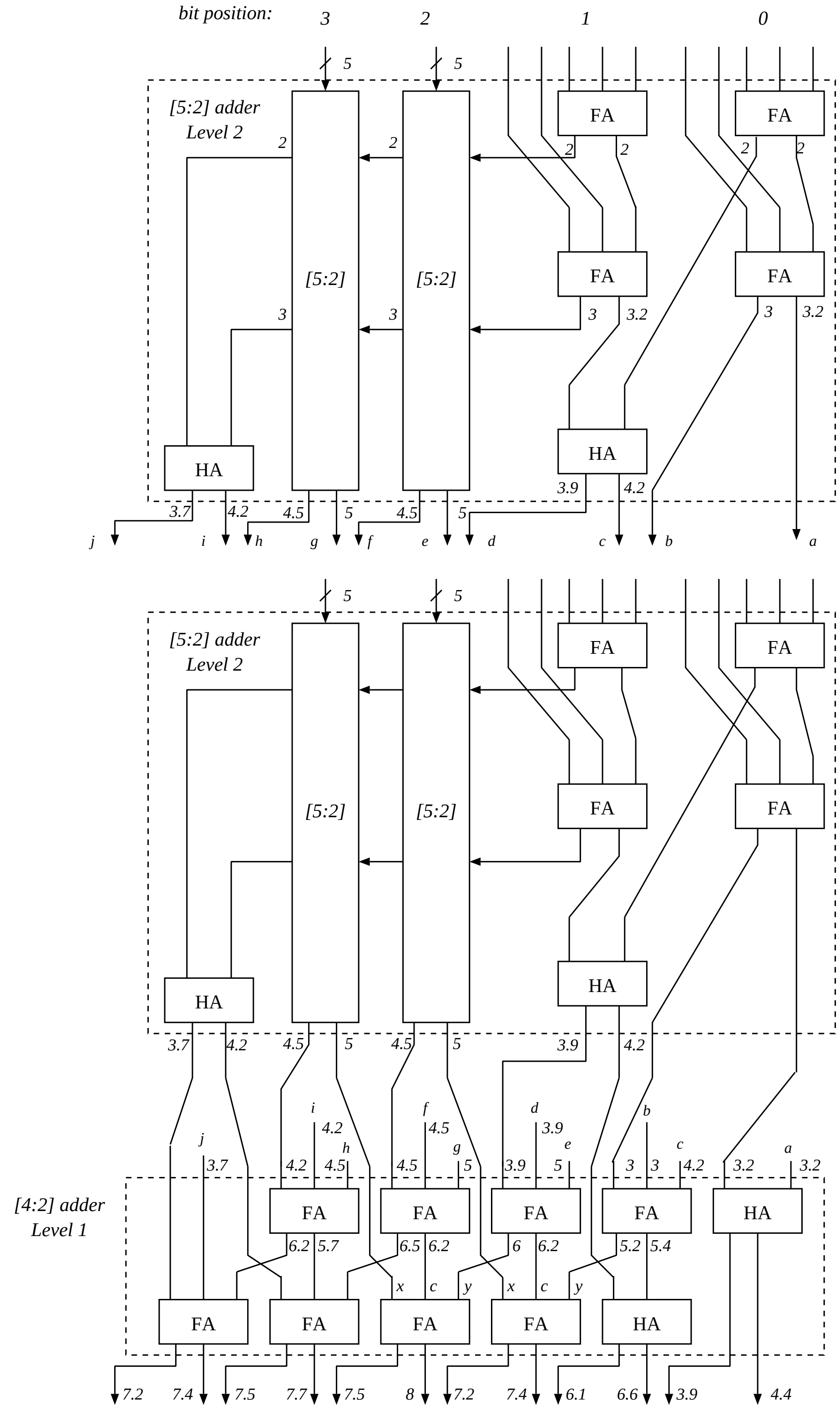


Figure E3.13b: Network of [5:2] and [4:2] modules to reduce 10 4-bit operands.

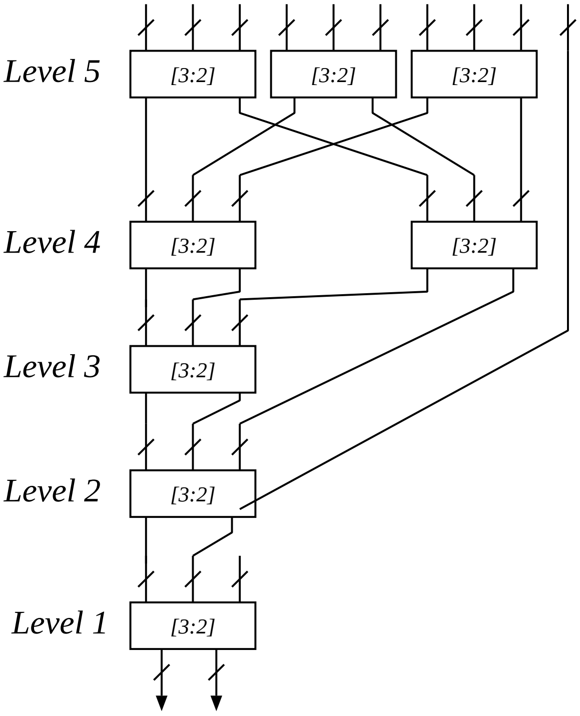


Figure E3.13c: Network of [3:2] adders to reduce 10 4-bit operands.

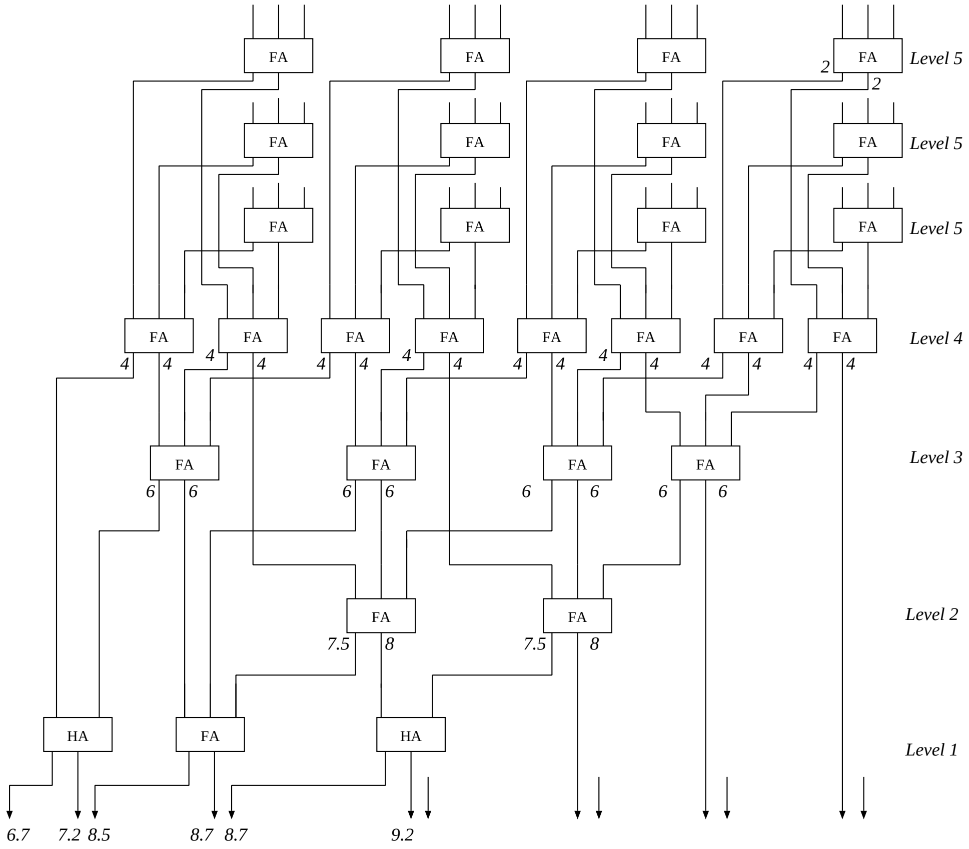


Figure E3.13d: Network of FAs and HAs to reduce 10 4-bit operands.

Exercise 3.18

We use two [4:2] adders in the first level. Assuming that the range of each operand is -128,127 we get a range of the output of each [4:2] adder of -512,508 requiring a width of 10 bits. Note that the sign extension could be simplified, as done Section 3.1, reducing the width of the adders.

Performing the [4:2] addition using the modules of Figure 2.41, described by

$$t_{i+1} = MAJORITY(x_i, y_i, w_i)$$

$$c_{i+1} = \begin{array}{ll} t_i & \textbf{if} \quad (x_i + y_i + w_i + z_i) mod\ 2 = 1 \\ z_i & \textbf{otherwise} \end{array}$$

$$s_i = (x_i + y_i + w_i + z_i + t_i) mod\ 2$$

we get

```
  73 0001001001            - 31     1111100001
- 52 1111001100              17     0000010001
  22 0000010110              47     0000101111
-127 1110000001             -80     1110110000
      ---------                      ---------
t    0010011000               t     0001000010
     -----------                    ----------
s    0010001010               s     0000101101
c    1100100010               c     1110100100
```

Now one second-level[4:2] adder. The range of the result is -1024,1016, requiring a width of 11 bits.

```
    00010001010
    11100100010
    00000101101
    11110100100
    -----------
t   00001010100
    -----------
s   00001110101
c   11100001000
    -----------
    11101111101 = -131
```

Exercise 3.22

a) From the Figures we see that the reduction by columns (Figure 3.21) has a CPA of 7 bits whereas the reduction by rows (Figure 3.27) has only 5 bits.

b) From the Figures, the critical path for reduction by columns is $4t_s + 5t_c + t_s = 5t_c + 5t_s$ and that for reduction by rows is $5t_s + 4t_c$.

c) Including the CPA, reduction by columns has 32 FA and 4 HA and reduction by rows has 32 FA and 3 HA.

Exercise 3.26

A pipelined linear array of adders is shown in Figure E3.26. For the final adder we use a CRA with four pipelined stages, each stage having a delay similar to a [4:2] adder.

```
m=8, n=6, [0,63]x8 = [0,504] --- 9 bits

Bit-matrix:

   xxxxxx
   xxxxxx  Stage 1
   xxxxxx
   xxxxxx
  ----------
  ooooooo
  oooooo
   xxxxxx  Stage 2
   xxxxxx
  ----------
  ooooooo
  oooooo
  oxxxxxx  Stage 3
  oxxxxxx
 ----------
 oooooooo
 ooooooo   (CPA with 4 pipelined stages)
 ----------
sssssssss
```

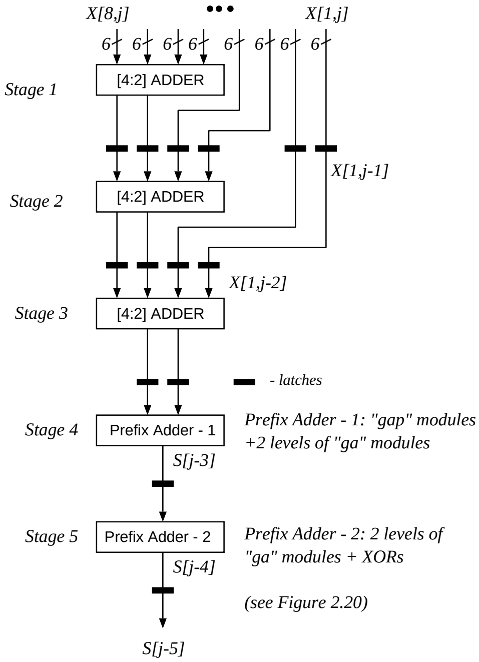


Figure E3.26: Pipelined linear array of [4:2] adders.

DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier Science, ©2004
– Updated: February 13, 2004 –

Chapter 4: Solutions to Selected Exercises

– With contributions by Elisardo Antelo –

Exercise 4.1

x= 30 X = 011110
y = -25 Y = 100111 Z= (-2)2(-1)

	CSA	shifted out		
$PS[0]$	00000000			
$SC[0]$	00000000			
xZ_0	11100001			
$4PS[1]$	11100001			
$4SC[1]$	0000000**1**			
$PS[1]$	11111000	10		
$SC[1]$	00000000			
xZ_1	00111100			
$4PS[2]$	11000100			
$4SC[2]$	01110000			
$PS[2]$	11110001	0010		
$SC[2]$	00011100			
xZ_2	11000011			
$4PS[3]$	00101110			
$4SC[3]$	1010001**1**			
$PS[3]$	00001011	010010		(cin=1)
$SC[3]$	11101000			
P	110100	010010	=	-750

From Figure 4.4 we determine that the number of cycles to obtain $PS[3], PC[3]$ is 6 (including one cycle to load X and Y).

In the last pass through the pipeline the register values are :
Register X = 011110 Register Y = ...10 Register C=0
Register XY = 11000100
Register SCH = 11101000 Register PSH=00001011
Register CS[1,0]=(10,11) Register PL = 0010

Exercise 4.3

To reduce the effect on the cycle time, the outputs of the carry-save adder are latched before being used as inputs to the converter. The input/output arithmetic relation is

$$2(PS_1[j-1] + SC_1[j-1]) + (PS_0[j-1] + SC_0[j-1] + w_0[j-1])$$

$$= 4w_0[j] + 2p_{2j+1} + p_{2j}$$

where $w[0]$ is the state. Since $0 \leq 2(PS_1[j-1] + SC_1[j-1]) + (PS_0[j-1] + SC_0[j-1] \leq 6$ and $0 \leq 2p_{2j+1} + p_{2j} \leq 3$ we get $0 \leq w_0 \leq 1$.

This is implemented with a 2-bit adder with $w_0[j-1]$ as the carry-in and $w_0[j]$ as the carry-out. The corresponding delay is $T_{conv} = t_{ab-c} + t_{c-c}$ which is somewhat larger than t_{ab-s} of the CS adder.

To keep the cycle time at t_{ab-s} as determined by the CSA, the scheme requires additional pipelining. The latency of the converter pipeline should not exceed the latency of the CPA used to obtain the MS bits of the product.

Exercise 4.5

A two's complement sequential multiplier with operands X and Y of 16 bits is designed similarly to the sequential multiplier in Figure 4.3. Note that the scheme in Figure 4.3 uses positive n-bit operands. This requires extension by two bits to handle negative multiples in radix 4. In this exercise, the operands are in the two's complement, thus one bit extension is suffucient. To reduce the cycle time, the design is pipelined (Figure E4.5a).

The delay and area of components are obtained with respect to NAND-2 using Tables 2.4 and 5.4 and summarized next

	delay	area
NOT	0.7	1
NAND-3	1.2	2
NOR-3	1.7	2
NOR-2	1.1	1
XOR	1.7	3
buffer	1.8	2.6
MUX-2	1.4	3
FA	4.2	6.7
flip-flop	4	4

The modules are

- Stage 1: Radix-4 recoder

  The sequential recoder for magnitudes described on p.185 and implemented in Fig. 4.5 produces radix-4 digits in the set {-1,0,1,2}. Since the multiplier in this exercise is in the two's complement system, the most significant radix-4 digit

  $$z_7 = -2y_{15} + y_{14} + c_7$$

  is in the set {-2,-1,0,1,2}.

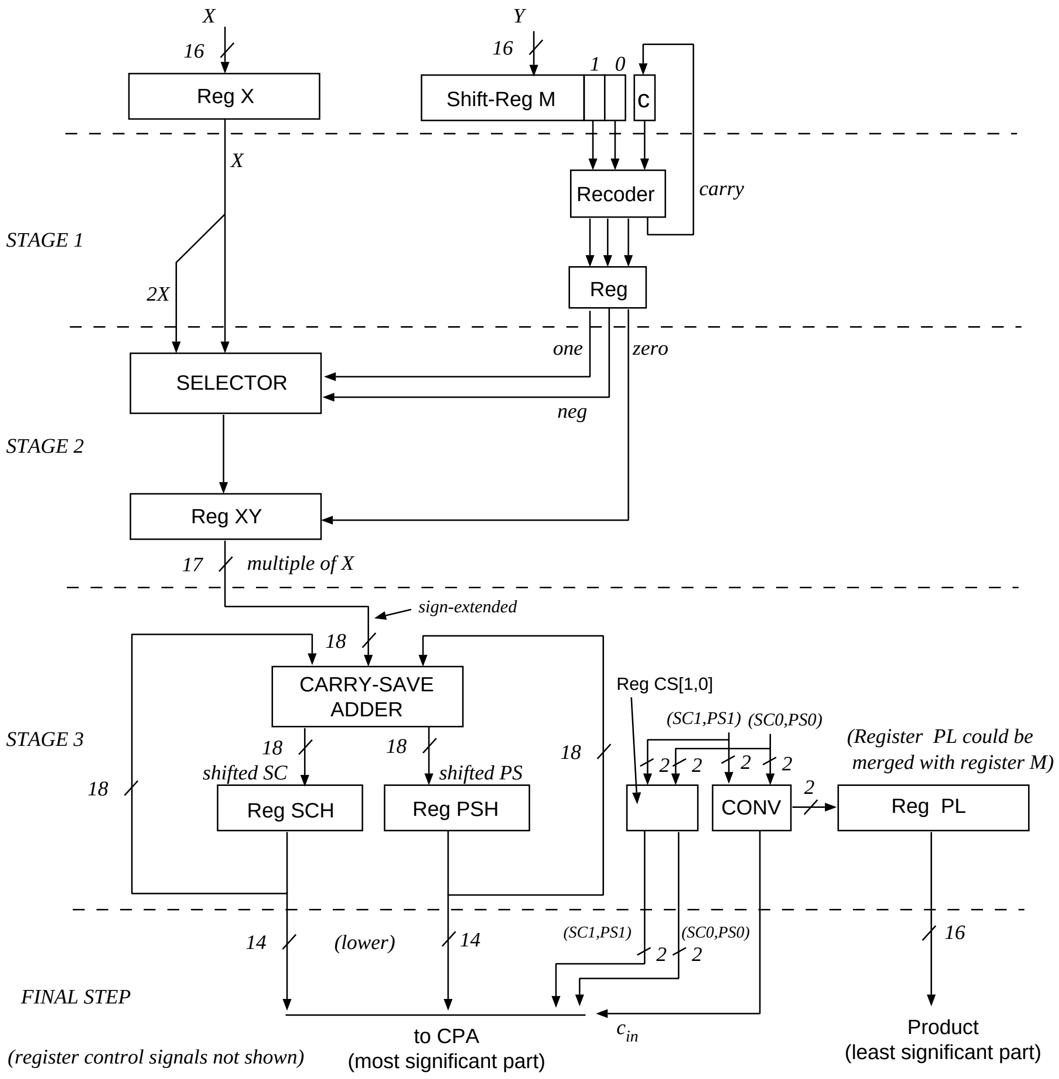


Figure E4.5a: 16-bit two's complement sequential multiplier. (Exercise 4.5)

The recoder of Fig. 4.5 is modified to produce a (-2) when $M1 = 1$, $M0 = 0$ and $C = 0$ in the cycle when z_7 is produced ($last = 1$). This results in a modified expression for neg while one, $zero$, and C_{next} remain unchanged:

$$neg = M1C + M1M0 + last \cdot M1M0'C' = M1(C + M0 + last \cdot M0'C')$$

$$= M1(C + M0 + last)$$

The modified recoder is shown in Figure E4.5b.

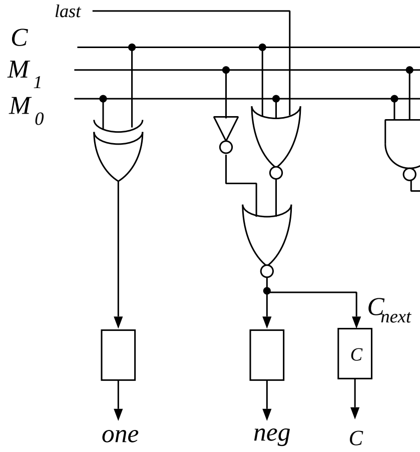


Figure E4.5b: Radix-4 recoder. (Exercise 4.5)

The delay and area of the recoder are:

	delay	area
1 XOR	1.7	3
2 NAND-3	1.2	4
2 NAND-2	1	2
1 NOR-3	1.7	2
1 NOR-2	1.1	1
3 NOT	0.7	3
4 FF	4	16
Total	2.9 +4	31

- Stage 2: Multiple generator

  The multiples ±2 × X, ±1 × X, and 0 × X are obtained as shown in Figure E4.5c.

  The delay and area of the multiple generator are:

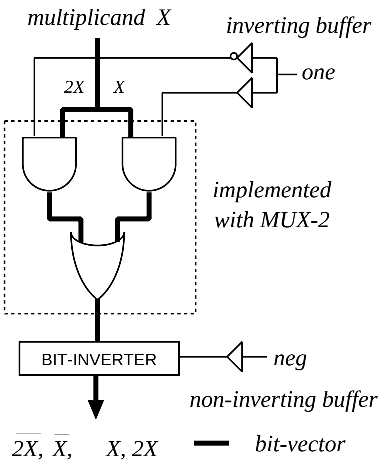


Figure E4.5c: Multiple generator. (Exercise 4.5)

	delay	area
3 BUFF	1.8	7.8
18 MUX-2	1.4	54
18 XOR	1.7	54
18 FF	4	72
Total	4.9+4	$\approx$ 188

- Stage 3: CSA

  The CSA adder consists of 19 FAs. The carry and sum are stored in two 19-bit registers SCH and PSH. The delay and area are:

	delay	area
19 FA	4.2	127.3
2x19 FF	4	152
Total	4.2+4	$\approx$ 280

The converter uses two FAs. To reduce the critical path, the 2-bit adder is pipelined so that only one FA is in the critical path. Four extra FFs are needed for pipelining. There is also a 16-bit register PL which stores the least-significant 16 bits of the product. The cycle time of the converter is 4.2 +4 = 8.2. Its area is $2 \times 6.7 + 8 \times 4 \approx 45$. For PL register the area is $16 \times 4 = 64$.

The cycle time of the multiplier is determined by the delay of Stage 2: 8.9 NAND-2 delays. To reduce this delay, a faster multiple generator could be designed using a 4-to-1 multiplexer to select ±2 and ±1 multiples. This would also require a change in the recoder design. The total area uses 544 equivalent gates.

Exercise 4.8

- The cycle time of a radix-2 multiplier is

  $$t_2 = t_{buf} + t_{NAND} + t_{c-s} + t_{reg}$$

  Using the values from Figure 5.4 we get

  $$t_2 = 1.8 + 1 + 2.2 + 4 = 9t_{NAND}$$

- To reduce the cycle time of the radix-16 implementation we pipeline as shown for radix 4 in Figure 4.3. The cycle time is the maximum of the critical paths of the three stages. We assume it is the adder, implemented as a [4:2] adder (Figure 2.41). Consequently, the cycle time is

  $$t_{16} = t_{[4:2]} + t_{reg}$$

  Using the values from Figure 5.4 we get

  $$t_{16} = 6 + 4 = 10t_{NAND}$$

- The total delay corresponds to the iterations (n for radix 2 and $n/4$ for radix 16) plus the two pipeline cycles for radix 16, plus the delay of the final adder). The speedup is

  $$S = \frac{t_2 \times n + t_{CPA}}{t_{16} \times (2 + n/4) + t_{CPA}} = \frac{36n + 4t_{CPA}}{10n + 80 + 4t_{CPA}}$$

- As seen in the expression, the speedup depends on n. This is because of the two additional cycles in radix 16 and of the carry-propagate adder.

  For instance, for $n = 16$ and using a carry-ripple adder we get

  $$S = \frac{36 \times 16 + 4(2.0 \times 16}{10 \times 16 + 80 + 128} = 1.9$$

Exercise 4.11

a) Radix-4 bit-matrix for multiplication of magnitudes with $x = 67$ and $y = 76$ is shown next. The recoded radix-4 multiplier is (11(-1)0).

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1		1	1	0	0	0	0	0	0	0	0	0	0
			0	1	0	1	1	1	1	0	0		0
	1	0	1	0	0	0	0	1	1		1		
0	1	0	0	0	0	1	1		0				
0	1	0	0	1	1	1	1	1	0	0	1	0	0

The result checks: $x \times y = 5092$.

b) Radix-4 bit-matrix for multiplication of 2's complement operands $x = -67$ and $y = -76$. The recoded radix-4 multiplier is ((-1)(-1)10).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1		1		1	1	0	0	0	0	0	0	0	0	0	0
					0	1	0	1	1	1	1	0	1		0
			1	0	1	0	0	0	0	1	0		0		
	1	0	1	0	0	0	0	1	0		1				
									1						
0	0	0	1	0	0	1	1	1	1	1	0	0	1	0	0

The result checks: $x \times y = 5092$.

Exercise 4.13

The reduced bit-matrix for radix-4 multiplication of magnitudes with $n = 12$, corresponding to Figure 4.14(b) is shown in Figure E4.13(a). The linear array has three stages.

- Stage 1 consists of a [4:2] adder and converter K1. The inputs to the converter in Stage 1 are denoted with "k".
- Stage 2 also has a [4:2] adder and converter K2.
- Stage 3 uses a [3:2] adder and a converter.

The partial inputs to Stage 2 and Stage 3 are shown in Figure E4.13(b) and (c), respectively. Each converter produces a conventional radix-4 digit ({0,1,2,3}) and a carry.

- Converter K1 consists of two HAs and its delay is clearly shorter than that of a [4:2] adder.
- Converter K2 uses one FA and one HA, again having a delay not greater than that of a [4:2] adder.
- Converter in Stage 3 could also use one FA and one HA. However, its delay would be longer than $t_{[3:2]} = t_{FA}$. To reduce its delay, bits denoted with "c" are used to produce two conditional 3-bit results (carry + 2 sum bits) in Stage 2. The delay of a 2-bit conditional adder (CA) is not larger than the delay of [4:2] adder. The correct sum is obtained using a MUX in Stage 3 based on the carry produced by converter K2 in Stage 2. This MUX has a shorter delay than a FA. Therefore, conversion of the least-significant radix-4 redundant digits does not increase the delay in the critical path.

Since in each stage two bits of the product are obtained, the final adder has 24 - 6 = 18 bits.

```
                                       [4:2] K1
                                         --- ----
1   1   1   1   1 s’s s x x x x x x x x x x|k k
                  s’x x x x x x x x x x x x|  k
              s’x x x x x x x x x x x x   x|
          s’x x x x x x x x x x x x   x ___|
      s’x x x x x x x x x x x x   x
  s’x x x x x x x x x x x x   x
  x x x x x x x x x x x x x
```

(a)

```
                                [4:2] CA  K2
                                 ---  ----  ----
                . . . . x x x x x x|c c x x p p
                . . . . x x x x x x|c c x k
            . . . . . . x x x x   x|
          . . . . . . . x x   x ___|
```

(b)

```
                              [3:2]
                              ---
                . . . . . x x x x|x     p p p p
                . . . . x x x x x|    k|MUX control
            . . . . . x x x   ___|c c c|MUX data
                                  c c c|MUX data
                                  -------
                                   MUX
```

(c)

```
                        CPA
                --------------------------
                . . . . x x x x x x p p p p p p
                . . . . x x x x x c
```

(d)

Figure E4.13: A linear array of [4:2] and [3:2] adders for 12 × 12 multiplication of magnitudes: (a) Reduced bit-matrix. (b) Inputs to Stage 2. (c) Inputs to Stage 3. (d) Inputs to CPA.

Exercise 4.15

Tables to determine the number of full and half adders in column reduction for multiplication of 8-bit operands for the following cases are:

(a) Radix-2 operands in two's complement representation, $n = 8$

Bit-matrix:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	$(x'_7y_7)'$	$(x_7y_6)'$	$(x_7y_5)'$	$(x_7y_4)'$	$(x_7y_3)'$	$(x_7y_2)'$	$(x_7y_1)'$	$(x_7y_0)'$	x_6y_0	x_5y_0	x_4y_0	x_3y_0	x_2y_0	x_1y_0	x_0y_0
		x'_6y_7	x_6y_6	x_6y_5	x_6y_4	x_6y_3	x_6y_2	x_6y_1	x_5y_1	x_4y_1	x_3y_1	x_2y_1	x_1y_1	x_0y_1	
			x'_5y_7	x_5y_6	x_5y_5	x_5y_4	x_5y_3	x_5y_2	x_4y_2	x_3y_2	x_2y_2	x_1y_2	x_0y_2		
				x'_4y_7	x_4y_6	x_4y_5	x_4y_4	x_4y_3	x_3y_3	x_2y_3	x_1y_3	x_0y_3			
					x'_3y_7	x_3y_6	x_3y_5	x_3y_4	x_2y_4	x_1y_4	x_0y_4				
						x'_2y_7	x_2y_6	x_2y_5	x_1y_5	x_0y_5					
							x'_1y_7	x_1y_6	x_0y_6						
							y_7	$(x_0y_7)'$							

Reduction table:

								i							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$l = 4$																
e_i	1	1	2	3	4	5	6	8	8	7	6	5	4	3	2	1
m_3	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
h_i		0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
f_i		0	0	0	0	0	1	2	1	0	0	0	0	0	0	0
$l = 3$																
e_i	1	1	2	3	4	6	6	6	6	6	6	5	4	3	2	1
m_2	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
h_i		0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
f_i		0	0	0	1	2	2	2	2	2	1	0	0	0	0	0
$l = 2$																
e_i	1	1	2	4	4	4	4	4	4	4	4	4	4	3	2	1
m_1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
h_i		0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
f_i		0	0	1	1	1	1	1	1	1	1	1	0	0	0	0
$l = 1$																
e_i	1	1	3	3	3	3	3	3	3	3	3	3	3	3	2	1
m_0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
h_i		0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
f_i		0	1	1	1	1	1	1	1	1	1	1	1	0	0	0
CPA	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1

e_i is the number of inputs in column i; f_i is the number of FAs; h_i is the number of HAs; m_j is the number of operands in the next level in the reduction sequence.

(b) Radix 4, magnitudes, multiplier recoding, $n = 7$

Bit-matrix:

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	s'_g	1	s'_e	s_e	s_e	e	e	e	e	e	e	e	e
h	h	g	s'_f	f	f	f	f	f	f	f	f		c_e
		h	g	g	g	g	g	g	g		c_f		
			h	h	h	h	h		c_g				

Reduction table:

							i							
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$l = 2$														
e_i	2	2	3	4	4	4	4	4	3	4	2	3	1	2
m_1	3	3	3	3	3	3	3	3	3	3	3	3	3	3
h_i	0	0	1	0	0	0	0	0	1	1	0	0	0	0
f_i	0	0	0	1	1	1	1	1	0	0	0	0	0	0
$l = 1$														
e_i	2	3	3	3	3	3	3	3	3	3	2	3	1	2
m_0	2	2	2	2	2	2	2	2	2	2	2	2	2	2
h_i	1	0	0	0	0	0	0	0	0	0	1	1	0	0
f_i	0	1	1	1	1	1	1	1	1	1	0	0	0	0
CPA	2	2	2	2	2	2	2	2	2	2	2	2	2	2

(c) Radix 4, two's complement, multiplier recoding, $n = 8$

Bit-matrix:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	s'_h	1	s'_g	1	s'_e	s_e	s_e	e	e	e	e	e	e	e	e
		h	h	g	s'_f	f	f	f	f	f	f	f	f		c_e
				h	g	g	g	g	g	g	g		c_f		
					h	h	h	h	h		c_g				
									c_h						

Reduction table:

								i							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$l = 3$																
e_i	1	1	2	2	3	4	4	4	4	5	3	4	2	3	1	2
m_2	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
h_i		0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
f_i		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$l = 2$																
e_i	1	1	2	2	4	4	4	4	4	4	3	4	2	3	1	2
m_1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
h_i		0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
f_i		0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
$l = 1$																
e_i	1	1	2	3	3	3	3	3	3	3	3	3	2	3	1	2
m_0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
h_i		0	1	0	0	0	0	0	0	0	0	0	1	1	0	0
f_i		0	0	1	1	1	1	1	1	1	1	1	0	0	0	0
CPA	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

Exercise 4.20

(a) The precision of S is 18 because $2^{17} < 127^2 * 16 < 2^{18}$.

(b) Since one pair of elements is available per cycle, a suitable algorithm is

$$S[i] = S[i-1] + A[i]B[i]$$

with $S = S[16]$ and $S[0] = 0$.

The recoding of $B[i]$ produces radix-4 digits. The resulting pipelined linear array with [3:2] adders is shown in Figure E4.20b.

(c) The cycle time is $t_{cycle-b} = max(t_{REC} + t_{buf} + t_{mux}, 2t_{FA})$

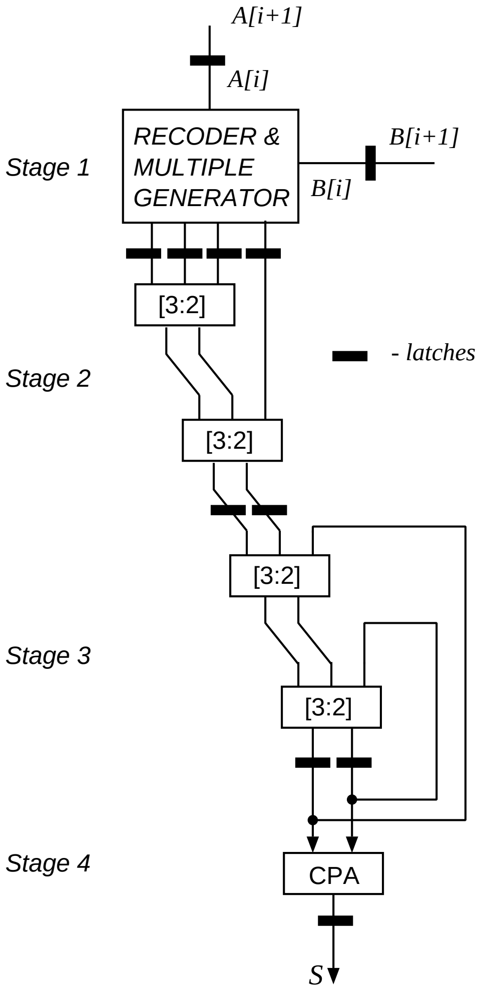


Figure E4.20b: A linear array of [3:2] adders for Exercise 4.20(b).

(d)

```
  1     2     3     4     5     6             19    20
|-----|-----|-----|-----|-----|-----|  . . . . |-----|-----|
compute               S[1]  S[2]  S[3]          S[16]
output                      S[1]  S[2]     . . .      S[16]
```

The latency is $T = 3 + 16 + 1 = 20$ clock cycles.

(e) A pipelined linear array with[4:2] adders is shown in Figure E4.20e.

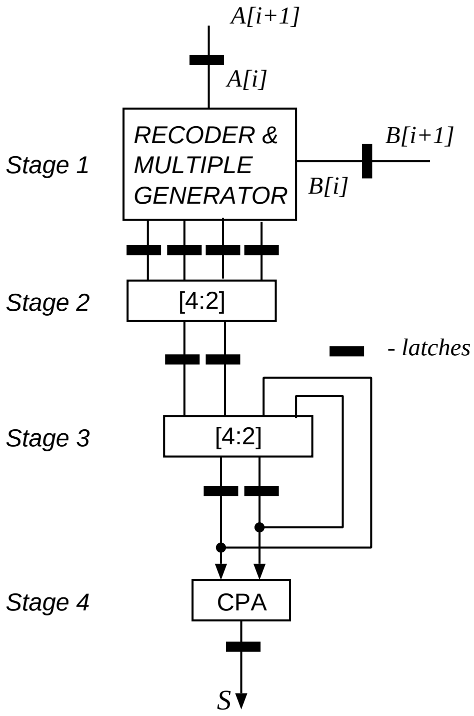


Figure E4.20e: A linear array of [4:2] adders for Exercise 4.20(e).

$$t_{cycle-e} = max(t_{REC} + t_{buf} + t_{mux}, t_{4-2})$$

Comparing with the linear array of part (b): The cycle time is the same if $t_{cycle-e} = t_{REC} + t_{buf} + t_{mux}$. Otherwise it depends on implementation of the [4:2] adder. If implemented with two [3:2] adders, there is no difference. If a gate network is used in implementing [4:2] module with a delay smaller than $2t_{FA}$, this implementation would have a shorter cycle time.

Exercise 4.26

The constant $C = 2925 = 0101101101101$ requires 8 additions.

Using canonical recoding we get C as $2925 = 10\bar{1}00\bar{1}00\bar{1}0\bar{1}01$ which requires 6 additions/subtractions.

Using factoring we get C as $2925 = (4+1)(8+1)(64+1) = (2^2+1)(2^3+1)(2^6+1)$ which requires 3 additions.

We use the factoring approach. The two designs are shown in Figure E4.26.

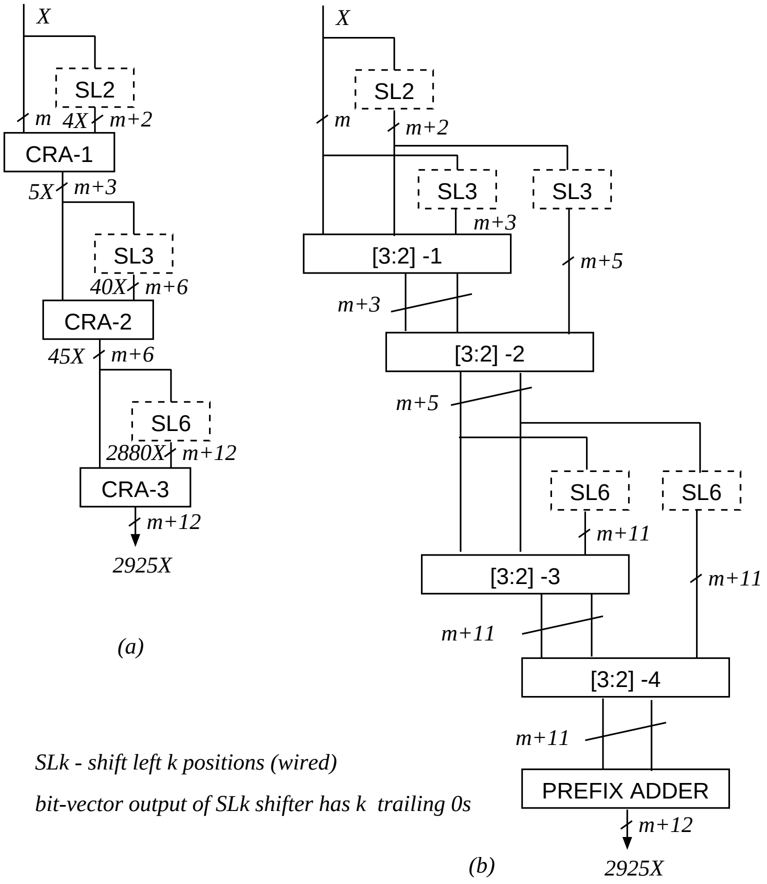


Figure E4.26: Constant multiplier networks: (a) With CRAs. (b) With [3:2] and prefix adder. (Exercise 4.26).

- Implementation with CRAs. To determine delay consider the following input/output diagram. FA and HA are denoted with "f" and "h". All delays are in terms of t_{FA}, and $t_{HA} = 0.5t_{FA}$ (same for sum and carry outputs). We show $m = 8$ in the diagram and generalize the result to arbitrary m.

```
                            xxxxxxxx
                          xxxxxxxx
CRA-1                     hhfffffh
                         xxxxxxxxxxx
                      xxxxxxxxxxx
CRA-2                 hhhfffffffh
                      xxxxxxxxxxxxxx
                xxxxxxxxxxxxxx
CRA-3           hhhhhhfffffffh
                xxxxxxxxxxxxxxxxxxxx
```

The critical path is: h+f+h+f+f+f+h+(fx(m-1))+h+h+h+h+h+h resulting in

$$T_{CRA} = 9t_{HA} + (m+3)t_{FA} = (m+7.5)t_{FA}$$

The equivalent number of full adders is:

$$C_{CRA} = (m-3)FA + 3HA + (m-1)FA + 4HA + (m-1)FA + 7HA$$
$$= 14HA + (3m-2)FA \approx (3m+5)FA$$

- Implementation with [3:2] adders and prefix adder.

  We determine the delay in the critical path and the cost as in the case with CRAs. To reduce the precision of the final adder, we apply [2:1] reduction where applicable.

```
                            xxxxxxxx
                          xxxxxxxx
                         xxxxxxxx
[3:2]-1                   hhfffffh
                         xxxxxxxxxxx
                         xxxxxxxx
                      xxxxxxxx
[3:2]-2                  fffffhhh
                      xxxxxxxxxxxxx
                        xxxxxxxx
                  xxxxxxxx
[3:2]-3               hhffhhhhhh
                  xxxxxxxxxxxxxxxxx
                     xxxxxxxxxx
                xxxxxxxxxxxxx
[3:2]-4           hhhffffffffhh
                xxxxxxxxxxxxxxxxxxx
                 xxxxxxxxxxxxx
PA              ==============
               xxxxxxxxxxxxxxxxxxxx
```

The precision of the PA adder is m+7 - reduced from m+12 by 5 positions. Using expression (2.61) the delay of the prefix adder is estimated as

$$T_{PA}(m) = t_{ga} + log_2(m)t_{cell} + t_{XOR} \approx 0.5t_{FA} + log_2(m) \times 0.6t_{FA} + 0.5t_{FA}$$

$$= [1 + 0.6 \times log_2(m)]t_{FA}$$

Using expression (2.62), we get the equivalent number of full adders

$$C_{PA}(m) \approx m \times FA + (m/2)log_2(m) \times 0.5FA$$

The critical path is: f+f+f+f+PA(m+7) resulting in

$$T_{[3:2]+PA} = 4t_{FA} + T_{PA}(m+7) < T_{CRA}$$

The equivalent number of full adders is:

$$C_{[3:2]+PA} = (m{-}3)FA{+}3HA{+}(m{-}3)FA{+}3HA{+}(m{-}6)FA{+}8HA{+}mFA{+}5HA{+}C(PA)$$

$$= (4m - 12)FA + 19HA + (m+7)FA + 0.25(m+7)log_2(m+7)FA$$

$$\approx [5m + 0.25(m+7)log_2(m+7)]FA > C_{CRA}$$

Without reducing the precision of the final adder, the input/output diagram is

```
                            xxxxxxxx
                          xxxxxxxx
                         xxxxxxxx
[3:2]-1                   hhfffff
                         xxxxxxxxxxx
                         xxxxxxx x
                      xxxxxxxx
[3:2]-2                  fffff
                      xxxxxxxxxxxxx
                        xxxxx xx x
                  xxxxxxx x
[3:2]-3               hhf
                  xxxxxxxxxxxxxxxxx
                     xxx xxxx xx x
                xxxxxxxxxxxxx
[3:2]-4           hhhffffffff
                xxxxxxxxxxxxxxxxxxx
                 xxxxxxxxxxx  xx x
PA              ===================
               xxxxxxxxxxxxxxxxxxxx
```

Calculation of the delay and cost is left to the reader.

DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier Science, ©2004
– Updated: September 9, 2003 –

Chapter 5: Solutions to Selected Exercises

– With contributions by Elisardo Antelo and Fabrizio Lamberti –

Exercise 5.2

In the following, two iterations of the division recurrence using a radix-16 implementation with two overlapped radix-4 stages for $x = 0.1001001110100101$ and $d = 0.110$ are shown.

- First iteration

$$\begin{array}{rlll}
4WS[0] = & 000.1001001110100101 & & \\
4WC[0] = & 000.0000000000000001^{*} & \hat{y}\,[0] = \frac{9}{16} & q_1 = 1 \\
-q_1 d = & 111.0011111111111111 & & \\
\hline
WS[1] = & 111.1010110001011011 & & \\
WC[1] = & 000.0010011101001010 & &
\end{array}$$

Speculative computations

- *Case a)* $q_1 = 2$

$$\begin{array}{rlll}
4^2\widehat{WS}\,[0] & 010.01001 & & \\
4^2\widehat{WC}\,[0] & 000.00000 & & \\
4 \times (-2 \times d) & 001.11111 & & \\
\hline
 & 011.1011 & & \\
 & 000.1001 & & \\
\hline
 & 100.0100 & \hat{y}\,[1] = -60/16 & \widehat{q}_2 = -2 \text{ (tentative)}
\end{array}$$

- *Case b)* $q_1 = 1$

$$\begin{array}{rlll}
4^2\widehat{WS}\,[0] & 010.01001 & & \\
4^2\widehat{WC}\,[0] & 000.00000 & & \\
4 \times (-1 \times d) & 100.11111 & & \\
\hline
 & 110.1011 & & \\
 & 000.1001 & & \\
\hline
 & 111.0100 & \hat{y}\,[1] = -12/16 & \widehat{q}_2 = -1 \text{ (tentative)}
\end{array}$$

- *Case c)* $q_1 = 0$

$$\begin{array}{rlll}
4^2\widehat{WS}\,[0] & 010.0100 & & \\
4^2\widehat{WC}\,[0] & 000.0000 & & \\
\hline
 & 010.0100 & \hat{y}\,[1] = 36/16 & \widehat{q_2} = 2 \text{ (tentative)}
\end{array}$$

- *Case d)* $q_1 = -1$

$$\begin{array}{rlll}
4^2\widehat{WS}\,[0] & 010.01001 & & \\
4^2\widehat{WC}\,[0] & 000.00000 & & \\
4 \times (-1 \times d) & 011.00000 & & \\
\hline
 & 001.0100 & & \\
 & 100.0000 & & \\
\hline
 & 101.0100 & \hat{y}\,[1] = -44/16 & \widehat{q_2} = -2 \text{ (tentative)}
\end{array}$$

- *Case e)* $q_1 = -2$

$$\begin{array}{rlll}
4^2\widehat{WS}\,[0] & 010.01001 & & \\
4^2\widehat{WC}\,[0] & 000.00000 & & \\
4 \times (-2 \times d) & 110.00000 & & \\
\hline
 & 100.0100 & & \\
 & 100.0000 & & \\
\hline
 & 000.0100 & \hat{y}\,[1] = 4/16 & \widehat{q_2} = 0 \text{ (tentative)}
\end{array}$$

Since $q_1 = 1$, we select case b). Therefore we have $q_2 = -1$. We can complete the first iteration as follows:

$$\begin{array}{rl}
4WS[1] = & 110.1011000101101100 \\
4WC[1] = & 000.1001110100101000 \\
-q_2 d = & 000.1100000000000000 \\
\hline
WS[2] = & 110.1110110001000100 \\
WC[2] = & 001.0010001001010000
\end{array}$$

- Second iteration

$$\begin{array}{rlll}
4WS[2] = & 011.1011000100010000 & & \\
4WC[2] = & 100.1000100101000000 & \hat{y}\,[2] = \frac{3}{16} & q_3 = 0 \\
-q_3 d = & 000.0000000000000000 & & \\
\hline
WS[3] = & 111.0011100001010000 & & \\
WC[3] = & 001.0000001000000000 & &
\end{array}$$

Speculative computations

- *Case a)* $q_3 = 2$

$$\begin{array}{rlll}
4^2\widehat{WS}\,[2] & 110.11000 & & \\
4^2\widehat{WC}\,[2] & 010.00100 & & \\
4 \times (-2 \times d) & 001.11111 & & \\
\hline
 & 101.0001 & & \\
 & 101.1100 & & \\
\hline
 & 010.1101 & \hat{y}\,[3] = 45/16 & \widehat{q_4} = 2 \text{ (tentative)}
\end{array}$$

- *Case b)* $q_3 = 1$

$$\begin{array}{r|llll}
4^2\widehat{WS}\,[2] & 110.11000 & & \\
4^2\widehat{WC}\,[2] & 010.00100 & & \\
4\times(-1\times d) & 100.11111 & & \\
\hline
 & 000.0001 & & \\
 & 101.1100 & & \\
\hline
 & 101.1101 & \hat{y}\,[3]=-35/16 & \widehat{q}_4=-2 \text{ (tentative)}
\end{array}$$

- *Case c)* $q_3 = 0$

$$\begin{array}{r|lll}
4^2\widehat{WS}\,[2] & 110.1100 & & \\
4^2\widehat{WC}\,[2] & 010.0010 & & \\
\hline
 & 000.1110 & \hat{y}\,[3]=14/16 & \widehat{q}_4=1 \text{ (tentative)}
\end{array}$$

- *Case d)* $q_3 = -1$

$$\begin{array}{r|lll}
4^2\widehat{WS}\,[2] & 110.11000 & & \\
4^2\widehat{WC}\,[2] & 010.00100 & & \\
4\times(-1\times d) & 011.00000 & & \\
\hline
 & 111.1110 & & \\
 & 100.0000 & & \\
\hline
 & 011.1110 & \hat{y}\,[3]=62/16 & \widehat{q}_4=2 \text{ (tentative)}
\end{array}$$

- *Case e)* $q_3 = -2$

$$\begin{array}{r|lll}
4^2\widehat{WS}\,[2] & 110.11000 & & \\
4^2\widehat{WC}\,[2] & 010.00100 & & \\
4\times(-2\times d) & 110.00000 & & \\
\hline
 & 010.1110 & & \\
 & 100.0000 & & \\
\hline
 & 110.1110 & \hat{y}\,[3]=-18/16 & \widehat{q}_4=-1 \text{ (tentative)}
\end{array}$$

Since $q_3 = 0$ we select case c). Therefore we have $q_4 = 1$. We can complete the second iteration as follows:

$$\begin{array}{rl}
4WS[3] = & 100.1110000101000000 \\
4WC[3] = & 100.0000100000000001^* \\
-q_4 d = & 111.0011111111111111 \\
\hline
WS[4] = & 111.1101011010111110 \\
WC[4] = & 000.0101001010000010
\end{array}$$

The digits of the result are $q_1 = 1$, $q_2 = -1$, $q_3 = 0$ and $q_4 = 1$. Therefore, we have $q = 00110001$.

Exercise 5.5

Let $Q\,[j]$ be the digit vector of the converted quotient consisting of the j most-significant digits, that is

$$Q\,[j] = \sum_{i=1}^{j} q_i r^{-i}$$

We have $Q\left[j+1\right] = Q\left[j\right] + q_{j+1}r^{-(j+1)}$. Since we are considering a radix-2 positive redundant representation with $q_i \in \{0, 1, 2\}$, we can use the following algorithm for the addition:

$$Q\left[j+1\right] = \begin{array}{ll} Q\left[j\right] + q_{j+1}2^{-(j+1)} & \text{if} \quad q_{j+1} \leq 1 \\ Q\left[j\right] + 2^{-j} & \text{if} \quad q_{j+1} = 2 \end{array}$$

This algorithm has the disadvantage that the addition $Q\left[j\right] + 2^{-j}$ requires the propagation of a carry and therefore it is slow. To avoid this propagation we define $QP\left[j\right]$ with value

$$QP\left[j\right] = Q\left[j\right] + 2^{-j}$$

Using this second form, the conversion algorithm is

$$Q\left[j+1\right] = \begin{array}{ll} Q\left[j\right] + q_{j+1}2^{-(j+1)} & \text{if} \quad q_{j+1} \leq 1 \\ QP\left[j\right] & \text{if} \quad q_{j+1} = 2 \end{array}$$

It is necessary to update also the form $QP\left[j\right]$, as follows:

$$QP\left[j+1\right] = Q\left[j+1\right] + 2^{-(j+1)} = \begin{cases} Q\left[j\right] + 2^{-(j+1)} & \text{if} \quad q_{j+1} = 0 \\ Q\left[j\right] + \left(1 + q_{j+1}\right) 2^{-(j+1)} & \text{if} \quad q_{j+1} = 1 \\ QP\left[j\right] + 2^{-(j+1)} & \text{if} \quad q_{j+1} = 2 \end{cases}$$

Using the definition of $QP\left[j\right]$, the expression for $QP\left[j+1\right]$ when $q_{j+1} = 1$ can be rewritten as follows:

$$Q\left[j\right] + \left(1 + q_{j+1}\right) 2^{-(j+1)} = Q\left[j\right] + 2^{-j} = QP\left[j\right]$$

Therefore, the expression $QP\left[j+1\right]$ when $q_{j+1} = 1$ and $q_{j+1} = 2$ can be condensed as follows:

$$QP\left[j+1\right] = QP\left[j\right] + \left(q_{j+1} - 1\right) 2^{-(j+1)} \quad \text{if} \quad q_{j+1} \geq 1$$

In conclusion, the algorithm for $QP\left[j+1\right]$ can be rewritten as follows:

$$QP\left[j+1\right] = \begin{array}{ll} Q\left[j\right] + 2^{-(j+1)} & \text{if} \quad q_{j+1} = 0 \\ QP\left[j\right] + \left(q_{j+1} - 1\right) 2^{-(j+1)} & \text{if} \quad q_{j+1} \geq 1 \end{array}$$

All the additions are now expressed by means of concatenations and no carry is propagated. In terms of concatenations, the on-the-fly conversion algorithm for a radix-2 positive redundant representation with digit set $\{0, 1, 2\}$ is

$$Q\left[j+1\right] = \begin{array}{ll} \left(Q\left[j\right], q_{j+1}\right) & \text{if} \quad q_{j+1} \leq 1 \\ \left(QP\left[j\right], 0\right) & \text{if} \quad q_{j+1} = 2 \end{array}$$

$$QP\left[j+1\right] = \begin{array}{ll} \left(Q\left[j\right], 1\right) & \text{if} \quad q_{j+1} = 0 \\ \left(QP\left[j\right], q_{j+1} - 1\right) & \text{if} \quad q_{j+1} \geq 1 \end{array}$$

with the initial conditions $Q\left[0\right] = 0$ and $QP\left[0\right] = 1$.

As an example, consider the conversion into conventional representation of the result 10211202.

j	q_j	$Q[j]$	$QP[j]$
0		0	1
1	1	0.1	1.0
2	0	0.10	0.11
3	2	0.110	0.111
4	1	0.1101	0.1110
5	1	0.11011	0.11100
6	2	0.111000	0.111001
7	0	0.1110000	0.1110001
8	2	0.11100010	0.11100011

Exercise 5.7

a) Implementation

An implementation of the retimed digit recurrence division (radix-4 with carry-save adder) is illustrated in Figure E5.7a. Details regarding the size of the most significant slice are presented in Figure E5.7b.

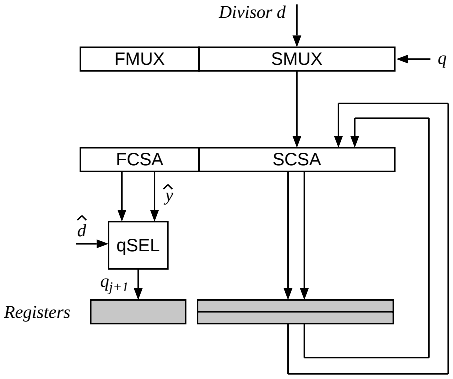


Figure E5.7a: Retimed implementation.

b) Delay analysis

– *Conventional design*

Computing the delay in the critical path we have (from Figure 5.4)

$$t_{cycle} = t_{qsel}(10.8) + t_{buff}(1.8) + t_{mux}(1.8) + t_{HA}(2.2) + t_{reg}(4).$$

Therefore, $t_{cycle} = 21t_{nand2}$. The number of iteration for IEEE double precision operands ($\rho < 1$) is $\lceil \frac{53+1+2}{2} \rceil = 28$. The latency of the conventional implementation can be computed as $(28+1) \times t_{cycle} = 29 \times 21t_{nand2} = 609t_{nand2}$.

– *Retimed version*

Computing the delay in the critical path (fast part) we have

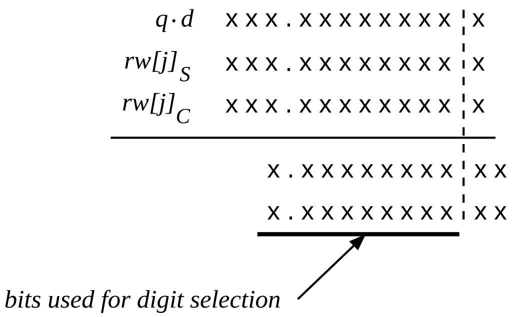


Figure E5.7b: Size of the most significant part of the path (size of FCSA is 7 bits, size of FMUX is 8 bits).

$$\begin{aligned} t_{cycle} = \quad & t_{buff}(1.8) \times \tfrac{40}{100} + \\ & + (t_{mux}(1.8) + t_{HA}(2.2)) \times \tfrac{80}{100} + t_{qsel}(10.8) + t_{reg}(4) \end{aligned}$$

Therefore, $t_{cycle} = 19t_{nand2}$. Computing the latency of the retimed version we get $(28 + 1 + 1) \times t_{cycle} = 30 \times 19t_{nand2} = 570t_{nand2}$.

Exercise 5.10

We normalize d to produce $d^* = 10010000 = 2^m d$ with $m = 4$. We define $d_f = d^* \times 2^{-n}$, where $n = 8$ is the number of bits of the operands. Assuming a redundant quotient digit-set with $q_i \in \{-2, -1, 0, 1, 2\}$, the redundancy factor is $\rho = \frac{a}{r-1} = \frac{2}{3}$. Since $\rho < 1$, we have $v = 2$. In order to obtain a correct remainder, the last digit of the quotient has to be aligned with a radix-4 boundary. For this, it must be $(m + v + s) \bmod k = 0$. Therefore we have $(4 + 2 + s) \bmod 2 = 0$ (with $k = 2$ and $m = 4$) and $s = 0$. We define $x_f = x \times 2^{-n}$ (as for the divisor). To achieve the required alignment, we shift x_f right by $v + s = 2$ bits. The initial condition is therefore

$$w\,[0] = \frac{x_f}{4} = .0001111000$$

Moreover, since the truncated divisor $\widehat{d} = 0.1001 = \frac{9}{16}$, we can compute $i = 16\widehat{d} = 9$. The corresponding selection constants are given by the following table:

i	8	9	10	11	12	13	14	15
$m_2(i)^+$	12	14	15	16	18	20	20	24
$m_1(i)^+$	4	4	4	4	6	6	8	8
$m_0(i)^+$	-4	-6	-6	-6	-8	-8	-8	-8
$m_{-1}(i)^+$	-13	-15	-16	-18	-20	-20	-22	-24

Finally, we can compute the number of iteration, $N = \lceil \frac{m+v}{k} \rceil$. Here $k = 2$ (since $r = 2^k$ where r is the radix of the quotient digit as produced by the division algorithm) and we get $N = \lceil \frac{4+2}{2} \rceil = 3$.

$$\begin{array}{rlll}
4WS\,[0] = & 000.01111000 & & \\
4WC\,[0] = & 000.00000001^{*} & \widehat{y}\,[0] = 000.0111 = \frac{7}{16} & q_1 = 1 \\
-d_f = & 111.01101111 & & \\
\hline
WS\,[1] = & 111.00010110 & & \\
WC\,[1] = & 000.11010010 & & \\
\hline
4WS\,[1] = & 100.01011000 & & \\
4WC\,[1] = & 011.01001000 & \widehat{y}\,[1] = 111.1001 = -\frac{7}{16} & q_2 = \overline{1} \\
+d_f = & 000.10010000 & & \\
\hline
WS\,[2] = & 111.10000000 & & \\
WC\,[2] = & 000.10110000 & & \\
\hline
4WS\,[2] = & 110.00000000 & & \\
4WC\,[2] = & 010.11000001^{*} & \widehat{y}\,[2] = 000.1100 = \frac{12}{16} & q_3 = 1 \\
-d_f = & 111.01101111 & & \\
\hline
WS\,[3] = & 011.10101110 & & \\
WC\,[3] = & 100.10000010 & &
\end{array}$$

Since $w\,[3] > 0$ the correction step is not needed. The quotient and the remainder are

$$q = 1\overline{1}1 = (13)_{10}$$
$$rem = w\,[3] \times 2^{n \log_2 2 - m} = w\,[3] \times 2^4 = 11 = (3)_{10}$$

Exercise 5.12

For signed-digit representation of the residual we get

$$\epsilon_{min} = -2^{-t} + ulp \qquad e_{max} = 2^{-t} + ulp$$

and

$$L_k^* = L_k - e_{min} = L_k + 2^{-t} - ulp$$
$$U_k = U_k - e_{max} = U_k - 2^{-t} + ulp$$

resulting in

$$\widehat{U}_{k-1} = \lfloor U_{k-1}^* + 2^{-t} \rfloor_t = \lfloor U_{k-1} \rfloor_t$$
$$\widehat{L}_k = \lceil L_k^* \rceil_t = \lceil L_k + 2^{-t} \rceil_t$$

For a necessary condition on δ and t (for $k > 0$) we get

$$U_{k-1}(d_i) - L_k(d_{i+1} + 2^{-t} \geq 0$$

that is,

$$(k - 1 + \rho)d_i - ((k - \rho)(d_i + 2^{-\delta}) + 2^{-t}) \geq 0$$

The worst case is for $k = a$ and $d_i = 1/2$ resulting in

$$\frac{2\rho - 1}{2} - (a - \rho)2^{-\delta} \geq 2^{-t}$$

which is the same as for carry-save representation of the residual (expression 5.101). For radix 2 ($\rho = a = 1$ we get $t \geq 1$ and it is possible to use the same constant for the whole range of the divisor. We use $t = 1$ and obtain

$$\widehat{U}_0(1/2) = 1/2 \qquad \widehat{U}_{-1}(1) = 0$$

$$\widehat{L}_1(1) = 1/2 \qquad \widehat{L}_0(1/2) = 0$$

Consequently, we get $m_1 = 1/2$ and $m_0 = 0$.

The range of the estimate $\hat{y}$ is

$$\lfloor -r\rho - (2^{-t} - ulp) \rfloor_t \leq r\rho + 2^{-t} - ulp \rfloor_t$$

which for $r = 2$ and $\rho = 1$ results in

$$-2 \leq \hat{y} \leq 2$$

The selection function is then

$$q_{j+1} = \begin{cases} 1 & \text{if } 1/2 \leq \hat{y} \leq 2 \\ 0 & \text{if } \;\; \hat{y} = 0 \\ -1 & \text{if} - 2 \leq \hat{y} \leq -1/2 \end{cases}$$

The execution for $x = 128 \times 2^{-8}$ and $d = 6 \times 2^{-3}$ is as follows:

$$\begin{array}{rlll}
2W[0] = & 0.10000000 & \widehat{y}[0] = 0.5 & q_1 = 1 \\
-q_1 d = & 0.\bar{1}\bar{1}000000 & & \\
\hline
2W[1] = & 0.\bar{1}000000 & \widehat{y}[1] = -0.5 & q_2 = -1 \\
\hline
-q_2 d = & 0.11000000 & & \\
\hline
2W[2] = & 0.10000000 & \widehat{y}[2] = 0.5 & q_3 = 1 \\
-q_3 d = & 0.\bar{1}\bar{1}000000 & & \\
\hline
2W[3] = & 0.\bar{1}0000000 & \widehat{y}[3] = -0.5 & q_4 = -1 \\
\hline
\end{array}$$

Since the pattern is periodic (and final residual is negative) we get

$$q = 2(0.1\bar{1}1\bar{1}1\bar{1}1\bar{1}0 = 0.10101010$$

Exercise 5.14

From expression 5.100, we obtain the lower bound for t and δ by requiring

$$U_{k-1}\,(d_i) - 2^{-t} - L_k(d_{i+1}) \geq 0$$

Using the definitions of L_k and U_k and considering the worst case condition $d_i = \frac{63}{64}$ (for a range of the divisor restricted to $\left[\frac{63}{64}, 1\right)$) and $k = a = 2$ (since $\rho = \frac{2}{3}$) we get

$$2^{-\delta} \leq \frac{3}{4} \times \left(\frac{21}{64} - 2^{-t}\right)$$

If we try $t = 2$ we get $2^{-\delta} \leq \frac{15}{256}$. We can use $\delta \geq 5$. In this case, if we use $\delta = 5$ we don't have dependence on d in the selection function since the interval of d is of width 2^{-6}.

We compute the selection intervals for $t = 2$. For $k = 2$ we get $\widehat{L}_2 = \lceil L_2 \rceil_2$ and $\widehat{U}_1 = \lfloor (U_1 - 2^{-t}) \rfloor_2$. Since $L_2 = \left(2 - \frac{2}{3}\right) \times 1 = \frac{4}{3}$ and $U_1 = \left(1 + \frac{2}{3}\right) \times \frac{63}{64} = \frac{5}{3} \times \frac{63}{64}$ we get $\widehat{L}_2 = \frac{6}{4}$ and $\widehat{U}_1 = \frac{5}{4}$. Being $\widehat{L}_2 \geq \widehat{U}_1$, $t = 2$ is not a possible solution.

We select $t = 3$. The corresponding selection intervals and selection contants are presented in Table E5.14.

Only one fractional bit of $\widehat{y}$ is necessary for the selection function. A possible implementation is presented in Figure E5.14.

Exercise 5.17

$[d_i, d_{i+1})$	$\left[\frac{63}{64}, 1\right)$
$\widehat{L}_2\left(d_{i+1}\right), \widehat{U}_1\left(d_i\right)^+$ $m_2\left(i\right)$	$11, 12$ 12
$\widehat{L}_1\left(d_{i+1}\right), \widehat{U}_0\left(d_i\right)^+$ $m_1\left(i\right)$	$3, 4$ 4
$\widehat{L}_0\left(d_{i+1}\right), \widehat{U}_{-1}\left(d_i\right)^+$ $m_0\left(i\right)$	$-5, -4$ -4
$\widehat{L}_{-1}\left(d_{i+1}\right), \widehat{U}_{-2}\left(d_i\right)^+$ $m_{-1}\left(i\right)$	$-13, -12$ -12

Table E5.14: Selection interval and m_k constants. Note: $^+$: real value= shown value/8

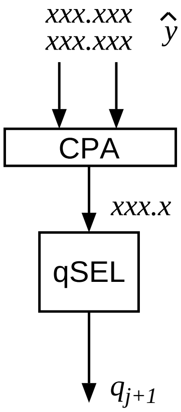


Figure E5.14: Implementation of the digit selection block.

a) Range of the divisor

From expression 5.16 we have

$$w[j+1] = rw[j] - q_{j+1}d = rw[j] - q_{j+1} - q_{j+1}(d-1)$$

Since $|q_{j+1}| \leq a$ we get

$$-a(d-1) \leq -q_{j+1}(d-1) \leq a(d-1)$$

From the expression for quotient digit selection

$$q_{j+1} = integer\,(rw[j] + 0.5) \leq a$$

we have

$$-\frac{1}{2} < rw[j] - q_{j+1} < \frac{1}{2}$$

From expression 5.15 we have

$$|rw[j]| \leq r\rho d$$

We obtain the following bounds on the shifted residual

$$max\left(-a + \frac{1}{2}, -r\rho d\right) < rw[j] < min\left(a - \frac{1}{2}, r\rho d\right)$$

Since the most critical restriction is the positive bound, we get

$$\frac{1}{2} + a\,|(1-d)| < min\left(\frac{2a-1}{2r}, \rho d\right)$$

In this case, since $d > 1$, we have

$$\frac{1}{2} + a(d-1) < \frac{2a-1}{2r}$$

Solving for d we get

$$d < 1 + \frac{2a-r-1}{2ar}$$

and therefore for convergence it must be

$$\beta < \frac{1}{r} - \frac{(r+1)}{2ar}$$

b) Possible implementation

An implementation of a high radix digit recurrence division with scaling and selection by rounding for nonredundant residuals is presented in Figure E5.17.

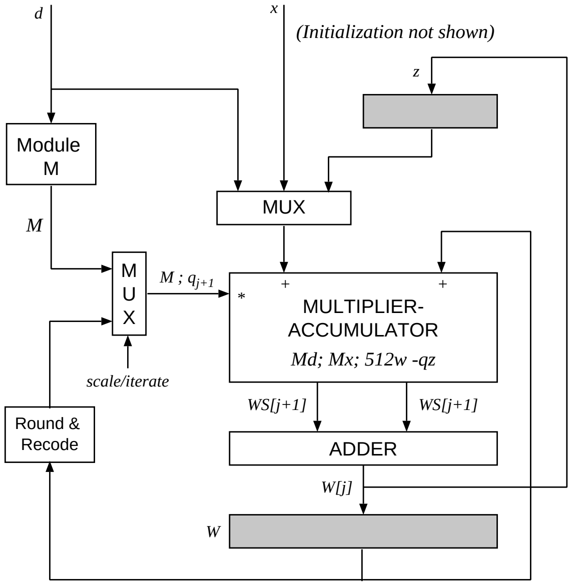


Figure E5.17: Implementation of a high radix division unit with scaling and selection by rounding (nonredundant residuals).

The hardware cost is higher in the high radix unit with respect to other low radix implementations due to the MAC block, additional registers and the module to compute the prescaling factor M. In the high radix unit, the number of cycles is reduced but t_{cycle} is larger. In the proposed implementation the speed-up with respect to other low radix implementations is limited by the nonredundant adder required to handle nonredundant residuals. To achieve a higher speed-up, we should consider a redundant representation of the residuals and a faster adder (see Chapter 5 for a fast implementation of a radix-512 division unit with residuals in carry-save form).

c) Example of execution for $r = 100$*,* $x = 0.83703960$ *and* $d = 1.00827040$

In the following we illustrate the method by finding the first three radix-r quotient digits. The recurrence is as follows:

$$w\,[j+1] = 100 \times w\,[j] - q_{j+1}d$$

The expression for quotient digit selection (for residuals in two's complement form) is

$$q_{j+1} = integer\,(100 \times w\,[j] + 0.5)$$

From a) we get

$$\beta < \frac{1}{r} - \frac{(r+1)}{2ar}$$

In this case, using $a = r - 1 = 99$ we get $\beta < 0.004899$. For convergence, it must be

$$1 \leq d \leq 1.004899$$

We compute the scaling constant $M = 1/1.005 \approx 0.995$. We scale the divisor thus obtaining $z = M \times d = 1.00322904$. We compute $M \times x$ and initialize $w\,[0] = M \times x = 0.83285440$.

$$\begin{aligned}
w\,[0] &= 0.83285440 \rightarrow q_1 = round(83.285440) = 83 \\[1em]
w\,[1] &= 100 \times 0.83285440 - 83 \times 1.00322904 = \\
&= 0.01742968 \rightarrow q_2 = round(1.742968) = 2 \\[1em]
w\,[2] &= 100 \times (0.01742968) - 2 \times 1.00322904 = \\
&= -0.26349008 \rightarrow q_3 = round(-26.349008) = -26
\end{aligned}$$

DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier Science, ©2004
– Updated: September 23, 2003 –

Chapter 6: Solutions to Exercises

– With contributions by Elisardo Antelo and Fabrizio Lamberti –

Exercise 6.1

a) *Radix-2, $s_j \in \{-1, 0, 1\}$, conventional (nonredundant) residual*

We have $x = 144 \times 2^{-8} = 0.10010000$ and $\rho = 1$. We choose $s_0 = 0$. Therefore the initialization is $w[0] = x - s_0 = 0.10010000$.

We use the result-digit selection function for redundant residual but we consider only 2 integer bits since the range of the residual estimate is smaller than in the redundant case.

$$\begin{array}{rrll}
2w[0] = & 001.00100000 & \widehat{y} = 1 & s_1 = 1 \\
F_1[0] = & 11.10000000 & F_{-1}[0] = 11.10000000 & \\
\hline
w[1] = & 00.10100000 & &
\end{array}$$

$$\begin{array}{rrll}
2w[1] = & 001.01000000 & \widehat{y} = 1 & s_2 = 1 \\
F_1[1] = & 10.11000000 & F_{-1}[1] = 00.11000000 & \\
\hline
w[2] = & 00.00000000 & &
\end{array}$$

$$\begin{array}{rrll}
2w[2] = & 000.00000000 & \widehat{y} = 0 & s_3 = 1 \\
F_1[2] = & 10.01100000 & F_{-1}[2] = 01.01100000 & \\
\hline
w[3] = & 10.01100000 & &
\end{array}$$

$$\begin{array}{rrll}
2w[3] = & 100.11000000 & \widehat{y} = -4 & s_4 = -1 \\
F_{-1}[3] = & 01.10110000 & F_1[3] = 10.00110000 & \\
\hline
w[4] = & 10.01110000 & &
\end{array}$$

$$\begin{array}{rrll}
2w[4] = & 100.11100000 & \widehat{y} = -4 & s_5 = -1 \\
F_{-1}[4] = & 01.10011000 & F_1[4] = 10.01011000 & \\
\hline
w[5] = & 10.01111000 & &
\end{array}$$

$$\begin{array}{rrll}
2w[5] = & 100.11110000 & \widehat{y} = -4 & s_6 = -1 \\
F_{-1}[5] = & 01.10001100 & F_1[5] = 01.10001100 & \\
\hline
w[6] = & 1110.01111100 & &
\end{array}$$

(*continues on next page*)

$$\begin{array}{rrll}
2w\,[6] = & 100.11111000 & \widehat{y} = -4 & s_7 = -1 \\
F_{-1}\,[6] = & 01.10000110 & F_1\,[6] = 10.01110110 & \\
\hline
w\,[7] = & 10.01111110 & & \\
\\
2w\,[7] = & 100.11111100 & \widehat{y} = -4 & s_8 = -1 \\
F_{-1}\,[7] = & 01.10000011 & F_1\,[7] = 10.01111011 & \\
\hline
w\,[8] = & 10.01111111 & & \\
\\
2w\,[8] = & 100.11111110 & \widehat{y} = -4 & s_9 = -1 \\
F_{-1}\,[8] = & 01.10000001 & F_1\,[8] = 10.01111101 & \\
\hline
w\,[9] = & 10.01111111 & &
\end{array}$$

We perform 9 iterations to compute the additional bit required for rounding. Since $w\,[9] < 0$ the correction step has to be performed. Thus $s_9 = -2$. The result is

$$s = 0.111\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{2} = (0.11000000)_2$$

b) Radix-2, $s_j \in \{-1, 0, 1\}$, carry-save residual

$$\begin{array}{rrll}
2WS\,[0] = & 0001.00100000 & \widehat{y} = 1 & s_1 = 1 \\
2WC\,[0] = & 0000.00000000 & & \\
F_1\,[0] = & 111.10000000 & F_{-1}\,[0] = 111.10000000 & \\
\hline
WS\,[1] = & 110.10100000 & & \\
WC\,[1] = & 010.00000000 & & \\
\\
2WS\,[1] = & 1101.01000000 & \widehat{y} = 1 & s_2 = 1 \\
2WC\,[1] = & 0100.00000000 & & \\
F_1\,[1] = & 110.11000000 & F_{-1}\,[1] = 000.11000000 & \\
\hline
WS\,[2] = & 111.10000000 & & \\
WC\,[2] = & 000.10000000 & & \\
\\
2WS\,[2] = & 1111.00000000 & \widehat{y} = 0 & s_3 = 1 \\
2WC\,[2] = & 0001.00000000 & & \\
F_1\,[2] = & 110.01100000 & F_{-1}\,[2] = 001.01100000 & \\
\hline
WS\,[3] = & 000.01100000 & & \\
WC\,[3] = & 110.00000000 & & \\
\\
2WS\,[3] = & 0000.11000000 & \widehat{y} = -4 & s_4 = -1 \\
2WC\,[3] = & 1100.00000000 & & \\
F_{-1}\,[3] = & 001.10110000 & F_1\,[3] = 110.00110000 & \\
\hline
WS\,[4] = & 101.01110000 & & \\
WC\,[4] = & 001.00000000 & &
\end{array}$$

(*continues on next page*)

$$\begin{array}{rlll}
2WS\,[4] = & 1010.11100000 & \widehat{y} = -4 & s_5 = -1 \\
2WC\,[4] = & 0010.00000000 & & \\
F_{-1}\,[4] = & 001.10011000 & F_1\,[4] = 110.01011000 & \\
\hline
WS\,[5] = & 001.01111000 & & \\
WC\,[5] = & 101.00000000 & &
\end{array}$$

$$\begin{array}{rlll}
2WS\,[5] = & 0010.11110000 & \widehat{y} = -4 & s_6 = -1 \\
2WC\,[5] = & 1010.00000000 & & \\
F_{-1}\,[5] = & 001.10001100 & F_1\,[5] = 001.10001100 & \\
\hline
WS\,[6] = & 001.01111100 & & \\
WC\,[6] = & 101.00000000 & &
\end{array}$$

$$\begin{array}{rlll}
2WS\,[6] = & 0010.11111000 & \widehat{y} = -4 & s_7 = -1 \\
2WC\,[6] = & 1010.00000000 & & \\
F_{-1}\,[6] = & 001.10000110 & F_1\,[6] = 110.01110110 & \\
\hline
WS\,[7] = & 001.01111110 & & \\
WC\,[7] = & 101.00000000 & &
\end{array}$$

$$\begin{array}{rlll}
2WS\,[7] = & 0010.11111100 & \widehat{y} = -4 & s_8 = -1 \\
2WC\,[7] = & 1010.00000000 & & \\
F_{-1}\,[7] = & 001.10000011 & F_1\,[7] = 110.01111011 & \\
\hline
WS\,[8] = & 001.01111111 & & \\
WC\,[8] = & 101.00000000 & &
\end{array}$$

$$\begin{array}{rlll}
2WS\,[8] = & 0010.11111110 & \widehat{y} = -4 & s_9 = -1 \\
2WC\,[8] = & 1010.00000000 & & \\
F_{-1}\,[8] = & 001.10000001 & F_1\,[8] = 110.01111101 & \\
\hline
WS\,[9] = & 001.01111111 & & \\
WC\,[9] = & 101.00000000 & &
\end{array}$$

We perform 9 iterations to compute the additional bit required for rounding. Since $w\,[9] < 0$ the correction step has to be performed. Thus $s_9 = -2$. The result is

$$s = 0.111\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{2} = (0.11000000)_2$$

c) *Radix-4,* $s_j \in \{-2, -1, 0, 1, 2\}$*, carry-save residual*

Since $\rho = \frac{a}{r-1} = \frac{2}{3} < 1$, s_0 should be 1. Therefore $w\,[0] = 1 - s_0 =$ 111.10010000.

$$\begin{array}{rrll}
4WS\,[0] = & 1110.01000000 & \widehat{S} = 1.0000 & S\,[0] = 1 \\
4WC\,[0] = & 0000.00000000 & \widehat{y} = 1110.010 & s_1 = -1 \\
F_1\,[0] = & 001.11000000 & & S\,[1] = 0.11 \\
\hline
WS\,[1] = & 11.10000000 & & \\
WC\,[1] = & 00.10000000 & &
\end{array}$$

$$\begin{array}{rrll}
4WS\,[1] = & 1110.00000000 & \widehat{S} = 0.1100 & s_2 = 0 \\
4WC\,[1] = & 0010.00000000 & \widehat{y} = 0000.000 & S\,[2] = 0.1100 \\
\hline
WS\,[2] = & 00.00000000 & & \\
WC\,[2] = & 00.00000000 & &
\end{array}$$

$$\begin{array}{rrll}
4WS\,[2] = & 0000.00000000 & \widehat{S} = 0.1100 & s_3 = 0 \\
4WC\,[2] = & 0000.00000000 & \widehat{y} = 0000.000 & S\,[3] = 0.110000
\end{array}$$

Since $w = 0$, the rest of the digits of S are 0. We perform 4 iterations to take into account the generation of the additional bit required for rounding. The radix-4 digits of the result are $s_0 = 1$, $s_1 = -1$, $s_2 = 0$, $s_3 = 0$, $s_4 = 0$ and $s_5 = 0$. The result is

$$s = (0.11000000)_2$$

Exercise 6.3

a) *Use $S[j]$ in its original signed digit form*

In this case it is not necessary the on-the-fly conversion of $S[j]$ for implementing the recurrence. Neverthless the register $K[j]$ is stil necessary. $F[j]$ is computed as

$$-S_{j+1}\left(2S[j]+S_{j+1}r^{-(j+1)}\right)$$

which requires a single concatenation of S_{j+1}, and a digit multiplication by S_{j+1}. Since $F[j]$ is represented in signed-digit form, the adder of the recurrence is more complex, that is, both operands are redundant.

b) *Convert $S[j]$ to two's complement representation*

The conversion is on-the-fly, and since this conversion is already necessary, it does not introduce additional complexity. The adder is simpler that in *a*) since one operand is in nonredundant form. More specifically the term $-S_{j+1}\left(2S[j]+S_{j+1}r^{-(j+1)}\right)$ is generated in nonredundant form as follows:

- $S_{j+1} \geq 0$

  Concatenate S_{j+1} to $2S[j]$ in position $j+1$. Set the most significant digit to one to have a negative operand (the weight of the most significant digit is negative). Then perform digit multiplication.

- $S_{j+1} < 0$

  In this case

  $$\left(2S[j]+S_{j+1}r^{-(j+1)}\right)=2\left(S[j]-r^{-j}\right)+\left(2r-S_{j+1}\right)r^{-(j+1)}$$

  The term $S[j]-r^{-j}$ is available from the on-the-fly conversion module. The term $2r-S_{j+1}$ is precomputed for every digit and is concatenated to $2\left(S[j]-r^{-j}\right)$ in postion $j+1$. Finally, the digit multiplication is performed.

Exercise 6.5

a) Network for digit selection

Figure E6.5a shows the network for the selection of s_{j+1} and s_{j+2} in a radix-2 square root implementation using two radix-2 overlapped stages.

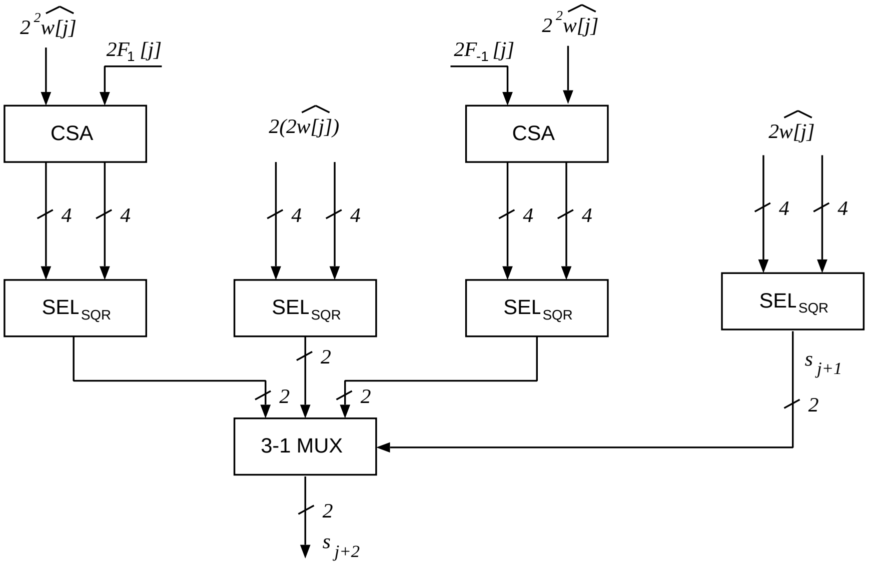


Figure E6.5a: Network for digit selection.

b) Network to produce the next residual

In Figure E6.5b the network producing the next residual is illustrated.

c) Delay analysis

- *Conventional implementation*

  Computing the delay in the critical path we have

  $$t_{cycle} = t_{SELSQRT}(4) + t_{buff}(1) + t_{mux}(1) + t_{HA}(1) + t_{reg}(2) = 9t_g$$

  The latency of the conventional implementation (8 fractional bits) can be computed as $8 \times t_{cycle} = 8 \times 9t_g = 72t_g$.

- *Overlapped implementation*

  Computing the delay in the critical path we have that the delay to produce $W[j+1]$ (that is, the delay from $W[j]$ to $W[j+1]$) is

  $$t_{SELSQRT}(4) + t_{buff}(1) + t_{mux}(1) + t_{HA}(1) = 7t_g$$

  Moreover, the delay to produce s_{j+2} (delay of CSA + delay of selection network + delay of 3-1 multiplexer) is

  $$t_{CSA}(2) + t_{SELSQRT}(4) + t_{mux}(1) + t_{buff}(1) = 8t_g$$

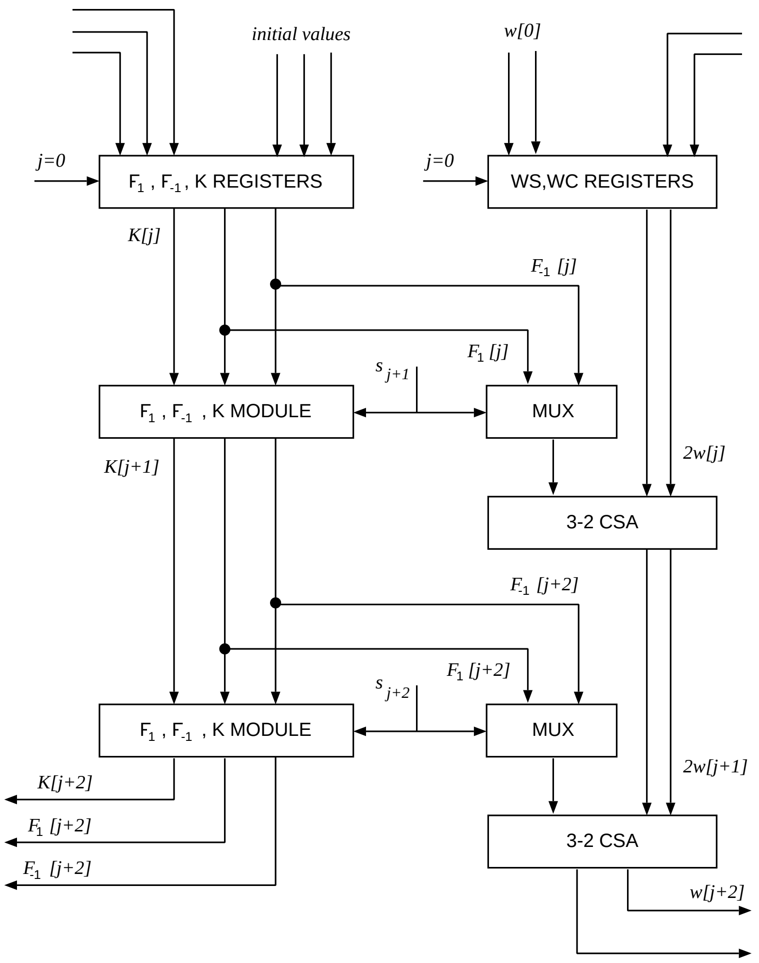


Figure E6.5b: Network to produce the next residual.

Finally, the delay to produce $W[j+2]$ (delay to produce s_{j+2} + delay of buffer + delay of mux + delay of HA) can be computed as

$$8t_g + 1t_g + 1t_g + 1t_g = 12t_g$$

Adding the register delay we get $t_{cycle} = 11t_g + 2t_g = 13t_g$. Computing the latency of the overlapped implementation (8 fractional bits) we get $4 \times t_{cycle} = 4 \times 13t_g = 52t_g$

Exercise 6.8

We compute the radix-4 square root of $x = (53)_{10} = (00110101)_2$. Since $n = 8$, we perform a right-shift of $m = 2$ bits and produce $x^* = .11010100$.

The number of bits of the integer result is $\frac{8-2}{2} = 3$. Consequently, two radix-4 iterations are necessary. We have $S[0] = 1$ and $w[0] = x^* - 1 = 11.11010100$.

Note that no alignment to digit boundary is needed, since the square root algorithm does not require to compute a remainder.

The iterations are as follows:

$$\begin{array}{rrlll}
4WS[0] = & 1111.01010000 & & & \\
4WC[0] = & 0000.00000000 & \widehat{y} = 1111.0101 & s_1 = -1 & S[1] = 0.11 \\
F_{-1}[0] = & 001.11000000 & & & \\
\hline
WS[1] = & 10.10010000 & & & \\
WC[1] = & 10.10000000 & & & \\
 & & & & \\
4WS[1] = & 1010.01000000 & & & \\
4WC[1] = & 1010.00000000 & \widehat{y} = 0100.0100 & s_2 = 2 & S[2] = 0.1110
\end{array}$$

We do not need to compute $w[2]$. Therefore the result is

$$s = 2^3\,(0.111) = 111 = (7)_{10}$$

Exercise 6.13

We develop a radix-4 selection function for $J = 3$, $t = 3$ and $\delta = 4$.

– $k > 0$

$$\min\left(U_{k-1}\left(I_i\right)\right) = 2 \times \left(\frac{1}{2} + i \times 2^{-4}\right) \times \left(k - \frac{1}{3}\right)$$

$$\max\left(L_k\left(I_i\right)\right) = 2 \times \left(\frac{1}{2} + (i+1) \times 2^{-4}\right) \times \left(k - \frac{2}{3}\right)$$

– $k \leq 0$

$$\min\left(U_{k-1}\left(I_i\right)\right) = 2 \times \left(\frac{1}{2} + (i+1) \times 2^{-4}\right) \times \left(k - \frac{1}{3}\right)$$

$$\max\left(L_k\left(I_i\right)\right) = 2 \times \left(\frac{1}{2} + i \times 2^{-4}\right) \times \left(k - \frac{2}{3}\right) + \left(k - \frac{2}{3}\right)^2 \times 4^{-4}$$

$$\widehat{L}_k = \max\left(\lceil L_k\left(I_i\right)\rceil_3\right) \leq m_k\left(i\right) \leq \min\left(\lfloor U_{k-1}\left(I_i\right)\right) - 2^{-3}\rfloor_3 = \widehat{U}_{k-1}$$

To improve the presentation of results, we use a bound for $\max\left(L_k\left(I_i\right)\right)$. More specifically, we want an upper bound of the term $\left(k - \frac{2}{3}\right)^2 \times 4^{-4}$. For $k = 0$ we have $\frac{4}{9} \times 4^{-4} = \frac{1}{576} < \frac{1}{512}$. For $k = 1$ we have $\left(-\frac{5}{3}\right)^2 \times 4^{-4} = \frac{25}{2304} < \frac{1}{64}$.

The selection constants are presented in Table E6.13. Note that we give only half of the table (for $\widehat{S}[j] = 8, 9, 10, 11$) since there is an interval $\widehat{U}_{-2} - \widehat{L}_{-1}$ that is negative. Consequently, there is no selection function for $t = 3$ and $\delta = 4$.

$\widehat{S}[j]$	8	9	10	11
$\widehat{L}_2, \widehat{U}_1$	$12,\ 12$	$14,\ 14$	$15,\ 15$	$16,\ 17$
m_2	12	14	15	16
$\widehat{L}_1, \widehat{U}_0$	$3,\ 4$	$4,\ 5$	$4,\ 5$	$4,\ 6$
m_1	4	4	4	4
$\widehat{L}_0, \widehat{U}_{-1}$	$-5,\ -4$	$-5,\ -5$	$-6,\ -5$	$-7,\ -5$
m_0	-4	-5	-6	-6
$\widehat{L}_{-1}, \widehat{U}_{-2}$	$-13,\ -13$	$-14,\ -15$	$-16,\ -16$	$-18,\ -17$
m_{-1}	-13	X	-16	-18

Table E6.13: Selection interval and m_k constants.
$\widehat{S}[j]$: real value= shown value/16.
$\widehat{L}_k,\ \widehat{U}_{k-1}$ and m_k: real value = shown value/8.

DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier Science, ©2004
– Updated: December 22, 2003 –

Chapter 7: Solutions to Exercises

– With contributions by Elisardo Antelo –

Exercise 7.1

From

$$\begin{aligned}\epsilon[j] &= 1 - d \cdot R[j]\\ \epsilon[j+1] &= 1 - d \cdot R[j+1] = (\epsilon[j])^2 = (1 - d \cdot R[j])^2\end{aligned}$$

we get

$$\begin{aligned}1 - d \cdot R[j+1] &= 1 - 2d \cdot R[j] + (d \cdot R[j])^2\\ d \cdot R[j+1] &= 2d \cdot R[j] - (d \cdot R[j])^2\\ R[j+1] &= 2R[j] - d \cdot R[j]^2 = R[j](2 - d \cdot R[j])\end{aligned}$$

Exercise 7.4

Find the reciprocal of $d = 29/256$ by the multiplicative normalization method. For the maximum error less tha $2^{-12} \approx 0.00024$ in the range $1/2 \leq d < 1$ we scale the input as follows:

$$\frac{1}{d} = \frac{1}{29/256} = \frac{1}{29/32} \times 2^3$$

and compute $\frac{1}{29/32}$

$P[0] = \lfloor 2 - 29/32 \rfloor_4 = 1.0001_2 = 1.0625$

j	$P[j]$	$d[j]$	$R[j]$	$\epsilon[j]$
0	1.0625	0.962891	1.0625	0.037
1	1.037109	0.998623	1.101929	1.38×10^{-3}
2	1.001377	0.999998	1.103446	1.9×10^{-6}
3	1.000002	0.999999	1.103448	3.6×10^{-12}

The answer is $R[3] \times 2^3 = 8.827586...$ compared to $256/29 = 8.827586...$ with an error less than 2^{-12}. Three iterations are used to guarantee that the error is smaller than 2^{-12} for $1/2 \leq d < 1$: for $d = 1/2$, $\epsilon[2] = 3.91 \times 10^{-3} > 2^{-12}$ so another iteration is needed.

Exercise 7.6

Optimal 5-bit input, 4-bit output reciprocal table is shown below. The actual input and output bits are underlined. The case 1.00000 produces the same output as for 1.1111x and needs to be detected.

5-bit input	4-bit output	5-bit input	4-bit output
$1.\underline{00000}$	$1.0\underline{0000}$	$1.\underline{10000}$	$0.1\underline{0101}$
$1.\underline{00001}$	$0.1\underline{1111}$	$1.\underline{10001}$	$0.1\underline{0101}$
$1.\underline{00010}$	$0.1\underline{1110}$	$1.\underline{10010}$	$0.1\underline{0100}$
$1.\underline{00011}$	$0.1\underline{1101}$	$1.\underline{10011}$	$0.1\underline{0100}$
$1.\underline{00100}$	$0.1\underline{1100}$	$1.\underline{10100}$	$0.1\underline{0100}$
$1.\underline{00101}$	$0.1\underline{1011}$	$1.\underline{10101}$	$0.1\underline{0011}$
$1.\underline{00110}$	$0.1\underline{1011}$	$1.\underline{10110}$	$0.1\underline{0011}$
$1.\underline{00111}$	$0.1\underline{1010}$	$1.\underline{10111}$	$0.1\underline{0010}$
$1.\underline{01000}$	$0.1\underline{1001}$	$1.\underline{11000}$	$0.1\underline{0010}$
$1.\underline{01001}$	$0.1\underline{1001}$	$1.\underline{11001}$	$0.1\underline{0010}$
$1.\underline{01010}$	$0.1\underline{1000}$	$1.\underline{11010}$	$0.1\underline{0010}$
$1.\underline{01011}$	$0.1\underline{1000}$	$1.\underline{11011}$	$0.1\underline{0001}$
$1.\underline{01100}$	$0.1\underline{0111}$	$1.\underline{11100}$	$0.1\underline{0001}$
$1.\underline{01101}$	$0.1\underline{0111}$	$1.\underline{11101}$	$0.1\underline{0001}$
$1.\underline{01110}$	$0.1\underline{0110}$	$1.\underline{11110}$	$0.1\underline{0000}$
$1.\underline{01111}$	$0.1\underline{0110}$	$1.\underline{11111}$	$0.1\underline{0000}$

Exercise 7.9

(a) With full multiplier ($55 \times 55 \rightarrow 55$, rounded)

- Rounding error of multiplication: $\pm 2^{-56}$ ($\pm 1/2$ ulp)
- Error due to ones' complement: 2^{-55} (1 ulp)

We now determine the bound on the generated error $\epsilon_G[j]$ by incorporating the bounds of errors associated with each iteration:

$$R[j+1] = R[j](2 - (R[j]d \pm 2^{-56}) - 2^{-55}) \pm 2^{-56}$$

$$= R[j](2 - R[j]d) \mp R[j]2^{-56} - R[j]2^{-55} \pm 2^{-56}$$

$$= R[j](2 - R[j]d) - \epsilon_G[j]$$

We assume that $R[j] < 1$ resulting in

$$-2^{-56} + 2^{-55} - 2^{-56} < \epsilon_G[j] < 2^{-56} + 2^{-55} + 2^{-56}$$

That is,

$$0 < \epsilon_G[j] < 2^{-54}$$

To get the final error, we use $\epsilon_T[j] = \epsilon_T[j-1] + \epsilon_G[j]$

$$\begin{aligned}
-2^{-8} &< \epsilon_T[0] < 2^{8} \\
\epsilon_T[1] &< \epsilon_T[0]^2 + \epsilon_G[0] = 2^{-16} + 2^{-54} \\
\epsilon_T[2] &< (2^{-16} + 2^{-54})^2 + 2^{-54} \\
\epsilon_T[3] &< ((2^{-16} + 2^{-54})^2 + 2^{-54})^2 + 2^{-54} \\
&= (2^{-32} + 2^{-108} + 2^{-69} + 2^{-54})^2 + 2^{-54} = \\
&= 2^{-54} + 2^{-64} + O(2^{-86})
\end{aligned}$$

(b) With rectangular multiplier (55 × 16 → 55, rounded)

j=0

$$\begin{aligned}
R[1] &= R[0](2 - (R[0]d \mp 2^{-56}) - 2^{-55}) \pm 2^{-16} \\
|\epsilon_G[0]| &\leq 2^{-56} + 2^{-55} + 2^{-16} \\
\epsilon_T[1] &= \epsilon_T[0]^2 + \epsilon_G[0] = (2^{-8})^2 + (2^{-56} + 2^{-55} + 2^{-16}) \\
&= 2^{-15} + 2^{-55} + 2^{-56}
\end{aligned}$$

j=1

$$\begin{aligned}
R[2] &= R[1](2 - (R[1]d \mp 2^{-56}) - 2^{-55}) \pm 2^{-32} \\
|\epsilon_G[1]| &\leq 2^{-56} + 2^{-55} + 2^{-32} \\
\epsilon_T[2] &= \epsilon_T[1]^2 + \epsilon_G[1] = (2^{-15} + 2^{-55} + 2^{-56})^2 + 2^{-56} + 2^{-55} + 2^{-32}
\end{aligned}$$

j=2

$$\begin{aligned}
R[3] &= R[2](2 - (R[2]d \mp 2 \times 2^{-56}) - 2^{-55}) \pm 2 \times 2^{-32} \\
|\epsilon_G[2]| &\leq 2 \times 2^{-56} + 2^{-55} + 2 \times 2^{-56} = 2^{-54} + 2^{-55} \\
\epsilon_T[3] &= \epsilon_T[2]^2 + \epsilon_G[2] = [(2^{-15} + 2^{-55} + 2^{-56})^2 + 2^{-56} + 2^{-55} + 2^{-32}]^2 \\
&+ 2^{-54} + 2^{-55} \\
&= 2^{-54} + 2^{-55} + 2^{-60} + O(2^{-64})
\end{aligned}$$

Exercise 7.13

$x = 1310/4096 = 0.010100011110$, $d = 2883/4096 = 0.101101000011$

The initial value: $R[0] = 2.98 - d = 1.100100101000$. As indicated on p.373, the maximum relative error is about 10^{-1}. For an error of 2^{-12}, two iterations are sufficient.

a) Using Newton-Raphson method (results truncated to 12 fractional bits):

j	$R[j]$	$\epsilon[j]$
0	1.100100101000	-0.107
1	1.0110011111001	0.011
2	1.011010111010	1.3×10^{-4}

The error in the computed quotient $q = x \times R[2] = 0.011101000100$ is smaller than 6×10^{-5} which is less than 2^{-12}.

b) Using multiplicative method: $P[0] = 2.98 - 2d = 1.5722 = 1.100100101000$ (Results truncated to 12 bits)

- Step 1:
  $d[0] = d \cdot P[0] = 1.000110110100$; $q[0] = x \cdot P[0] = 0.100000001011$
- Step 2:
  $P[1] = 2 - d[0] = 0.111001001011$
  $d[1] = d[0] \cdot P[1] = 0.1111111010001$; $q[1] = q[0] \cdot P[1] = 0.011100110000$
- Step 3:
  $P[2] = 2 - d[1] = 1.000000101110$;
  $d[2] = d[1] \cdot P[2] = 0.111111111111$; $q[2] = q[1] \cdot P[2] = 0.011101000100$
  Again, the error in the computed quotient is less than 2^{-12}.

The error in the quotient is 5.9×10^{-5}.

Exercise 7.17

The algorithm to implement is:

$$X[0]=x, \quad S[0]=x, \quad P[0]=A$$

where A is an approximation to $1/\sqrt{x}$ with an error less than 2^{-8}.

```
for $j = 0$ to 3
    $P[j] = 1 + \frac{1}{2}(1 - X[j])$
    $P2[j] = P[j]P[j]$
    $X[j+1] = X[j]P2[j]$
    $S[j+1] = S[j]P[j]$
```

(a) Alternative with a full 55 × 55 multiplier, a 3-stage pipeline.

- $P[0] = A$ – one cycle;
- Scheduling of an iteration in the pipelined multiplier is shown in Figure E7.17. It takes 4 cycles to obtain $S[j+1]$. An iteration takes 6 cycles.
- Latency:

  1 cycle for initial approximation
  3 full iterations, each 6 cycles for a total of 18 cycles
  partial iteration to obtain $S[4]$ in 4 cycles
  total: 23 cycles

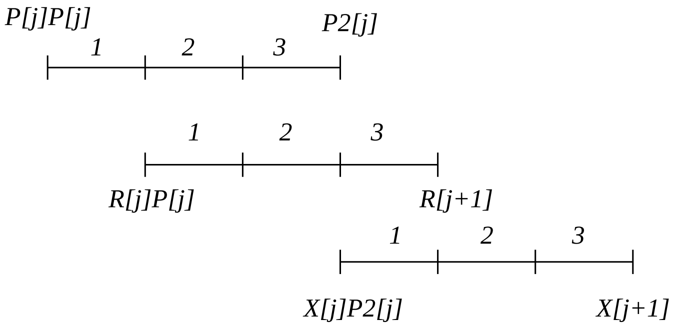


Figure E7.17: Scheduling of one iteration

(b) With 55 × 16 rectangular multipliers (single stage)

- $P[0] = A$, a 9-bit approximation; 1 cycle
- First iteration:

  $x[1] = x[0] \cdot P[0]$; (55 × 9); 1 cycle
  $x[1] = x[1] \cdot P[0]$; (55 × 9); 1 cycle
  $S[1] = S[0] \cdot P[0]$; (55 × 9); 1 cycle
- Second iteration:

$P[1] = 1 + \frac{1}{2}(1 - x[1])$; rounded to 16 bits
$x[2] = x[1] \cdot P[1]$; (55×16); 1 cycle
$x[2] = x[2] \cdot P[1]$; (55×16); 1 cycle
$S[2] = S[1] \cdot P[1]$; (55×16); 1 cycle

- Third iteration:

$P[2] = 1 + \frac{1}{2}(1 - x[2])$; rounded to 32 bits
$x[3] = x[2] \cdot P[2]$; (55×32); 2 cycles
$x[3] = x[3] \cdot P[2]$; (55×32); 2 cycles
$S[3] = S[2] \cdot P[2]$; (55×32); 2 cycles

- Termination:

$P[3] = 1 + \frac{1}{2}(1 - x[3])$; rounded to 55 bits
$S[4] = S[3] \cdot P[3]$; (55×55); 4 cycles

- Latency: 1+3+3+6+4 = 17 cycles. This can be reduced to 13 cycles if two rectangular multipliers are used.

DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier Science, ©2004

Chapter 8: Solutions to Exercises

– with contributions by Fabrizio Lamberti –

Exercise 8.1

- *Fixed-point representation*

  Planck's constant:

  $$6.63 \times 10^{-27} \rightarrow 0.\underbrace{00000000000000000000000000663}_{29}$$

  Avogadro's number:

  $$6.02 \times 10^{23} \rightarrow \underbrace{602000000000000000000000}_{24}.0$$

  To represent the approximation of Planck's constant 6.63×10^{-27}, 29 radix-10 fractional digit are needed, while representing the approximation of Avogadro's number 6.02×10^{23} requires 24 integer digits. In conclusion, to represent the approximations of both Planck's constant and Avogadro's number in a fixed-point number format, $29 + 54 = 53$ radix-10 digits are needed.

- *Floating-point representation*

  In the considered radix-10 base-10 biased representation for the exponent (such that $E_{biased} = E + 50$), the exponent of both Planck's constant 6.63×10^{-27} and Avogadro's number 6.02×10^{23} can be represented using 2 digits, since $-27+50 = 23$ and $23+50 = 73$. To represent the significands, 3 radix-10 digits are needed. Therefore, to represent the approximations of both Planck's constant and Avogadro's number in a floating-point radix-10 base-10 number format, $3 + 2 = 5$ digits are needed.

Exercise 8.4

Since in a normalized representation the most significant digit of the significand is always different from zero, if we assume a floating point representation with f digits for the significand and e digits for the exponent, the number of values for the first digit of the significand depends on the base that is being considered. For instance, the first four bits (one hexadecimal digit) have 8 values for radix 2, 12 values for radix 4 and 15 values for radix 16. The values that can be represented using the remaining $f-4$ digits of the significand and e digits of the exponent remain unchanged for different bases. Therefore we have

(a) *System A has base* 16 *and system B has base* 2

Since the number of normalized significands for system A is $15 \times 2^{f-4}$ and the number of normalized significands for system B is $8 \times 2^{f-4}$, the ratio between the number of floating-point numbers that are represented by systems A and B is $\frac{15}{8}$.

(b) *System A has base* 16 *and system B has base* 4

Since the number of normalized significands for system A is $15 \times 2^{f-4}$ and the number of normalized significands for system B is $12 \times 2^{f-4}$, the ratio between the number of floating-point numbers that are represented by systems A and B is $\frac{15}{12}$.

Exercise 8.7

In a normalized base-64 floating-point representation, the number of values that can be represented with the first digit is limited to 63. Therefore the number of different significands that can be represented with 48-bit significands is $63 \times 2^{48-6} = 63 \times 2^{42}$.

Exercise 8.10

Notice that for rounding toward zero only f fractional bits are required. For rounding to nearest, one additional bit is required to take into account all discarded bits (since the sticky bit T is not provided, we assume $T = 0$ for ties). For rounding toward plus infinity it is necessary to know the sign as well as when all the bits to be discarded are zero.

s	exp	fraction	guard	round mode
0	00011111	1111111111111	1	
0	00100000	0000000000000		RNE
0	00011111	1111111111111		RNO
0	00011111	1111111111111		RZ
0	00100000	0000000000000		RPINF

s	exp	fraction	guard	round mode
0	11111110	1111111111111	1	
0	11111111	0000000000000		RNE
0	11111110	1111111111111		RNO
0	11111110	1111111111111		RZ
0	11111111	0000000000000		RPINF

s	exp	fraction	guard	round mode
1	11111110	1111111111111	1	
1	11111111	0000000000000		RNE
1	11111110	1111111111111		RNO
1	11111110	1111111111111		RZ
1	11111110	1111111111111		RPINF

Exercise 8.12

Hex-vector	Value
00000000	0.0
80000000	-0.0
A73FF801	$(1.01111111111100000000001)_2 \times 2^{-51}$
A6800000	-1.0×2^{48}
7F7FFFFF	$\left(2 - 2^{-23}\right) \times 2^{127}$
00800000	1.0×2^{-126}
7F800000	$+\infty$
FF800000	$-\infty$
7FC00000	NAN

Exercise 8.16

	Operation	X	Y
A	Add	000110001001111000	000110011100011101
B	Add	000110001001111000	100110011100011101
C	Sub	000110001001111000	000110001001110111
D	Sub	011111110111100011	111111100001010101

(A) *EOP is ADD*

Output of blocks in Fig. 8.5	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	2
$OUTPUT_{MUX}$	00110011
$INPUT_{R-SHIFTER}$	1.001111000
$OUTPUT_{R-SHIFTER}$	0.01001111000
$OUTPUT_{SM-ADD/SUB}$	1.11011101100
$OUTPUT_{L/R1-SHIFTER}$	1.11011101100
$OUTPUT_{ROUND(RNE)}$ $OUTPUT_{EXPONENT\ UPDATE(RNE)}$	1.110111011 00110011
$OUTPUT_{ROUND(RZ)}$ $OUTPUT_{EXPONENT\ UPDATE(RZ)}$	1.110111011 00110011
$OUTPUT_{ROUND(RPINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RPINF)}$	1.110111011 00110011
$OUTPUT_{ROUND(RMINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RMINF)}$	1.110111011 00110011
$OUTPUT_{SIGN}$	0

(B) *EOP is SUB*

Output of blocks in Fig. 8.5	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	2
$OUTPUT_{MUX}$	00110011
$INPUT_{R-SHIFTER}$	1.001111000
$OUTPUT_{R-SHIFTER}$	0.01001111000
$OUTPUT_{SM-ADD/SUB}$	1.00111111100
$OUTPUT_{L/R1-SHIFTER}$	1.00111111100
$OUTPUT_{ROUND(RNE)}$ $OUTPUT_{EXPONENT\ UPDATE(RNE)}$	1.001111111 00110011
$OUTPUT_{ROUND(RZ)}$ $OUTPUT_{EXPONENT\ UPDATE(RZ)}$	1.001111111 00110011
$OUTPUT_{ROUND(RPINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RPINF)}$	1.001111111 00110011
$OUTPUT_{ROUND(RMINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RMINF)}$	1.001111111 00110011
$OUTPUT_{SIGN}$	1

(C) *EOP is SUB*

Output of blocks in Fig. 8.5	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	0
$OUTPUT_{MUX}$	00110001
$INPUT_{R-SHIFTER}$	1.001110111
$OUTPUT_{R-SHIFTER}$	1.001110111
$OUTPUT_{SM-ADD/SUB}$	0.000000111
$OUTPUT_{L/R1-SHIFTER}$	1.110000000
$OUTPUT_{ROUND(RNE)}$ $OUTPUT_{EXPONENT\ UPDATE(RNE)}$	1.110000000 00101010
$OUTPUT_{ROUND(RZ)}$ $OUTPUT_{EXPONENT\ UPDATE(RZ)}$	1.110000000 00101010
$OUTPUT_{ROUND(RPINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RPINF)}$	1.110000000 00101010
$OUTPUT_{ROUND(RMINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RMINF)}$	1.110000000 00101010
$OUTPUT_{SIGN}$	0

(D) *EOP is ADD*

Output of blocks in Fig. 8.5	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	2
$OUTPUT_{MUX}$	11111110
$INPUT_{R-SHIFTER}$	1.001010101
$OUTPUT_{R-SHIFTER}$	0.01001010101
$OUTPUT_{SM-ADD/SUB}$	10.00111100001
$OUTPUT_{L/R1-SHIFTER}$	1.000111100001
$OUTPUT_{ROUND(RNE)}$ $OUTPUT_{EXPONENT\ UPDATE(RNE)}$	1.000111100 11111110
$OUTPUT_{ROUND(RZ)}$ $OUTPUT_{EXPONENT\ UPDATE(RZ)}$	1.000111100 11111110
$OUTPUT_{ROUND(RPINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RPINF)}$	1.000111101 11111110
$OUTPUT_{ROUND(RMINF)}$ $OUTPUT_{EXPONENT\ UPDATE(RMINF)}$	1.000111100 11111110
$OUTPUT_{SIGN}$	0

Exercise 8.20

(a) *Determine the delay of the floating-point adder in Fig. 8.5 for single and double precision*

Module	Delay for Single precision	Delay for Double precision
Exponent difference	1.4 ns	1.7 ns
Swap (incl. buffer for control)	0.5 ns	0.5 ns
Right shift	1.0 ns	1.2 ns
Add significands (s+m)	2.5 ns	2.8 ns
LOD	1.5 ns	1.8 ns
Left shift (includes buffer)	1.7 ns	2 ns
Round	1.0 ns	1.2 ns
Right shift (one pos., incl. buf.)	0.5 ns	0.5 ns
Special cases	0.8 ns	0.8 ns
Delay	10.9 ns	12.5 ns

(b) *Pipeline the floating-point adder (for single and double precision) for a clock rate of 200 Mhz (stage delay should not be larger than 80% of the clock cycle)*

Since a clock rate of $200Mhz$ correspond to a clock cycle of 5 ns, stage delay should not be larger that 4 ns. The floating-point adder for single precision could be pipelined as follows (3 stages):

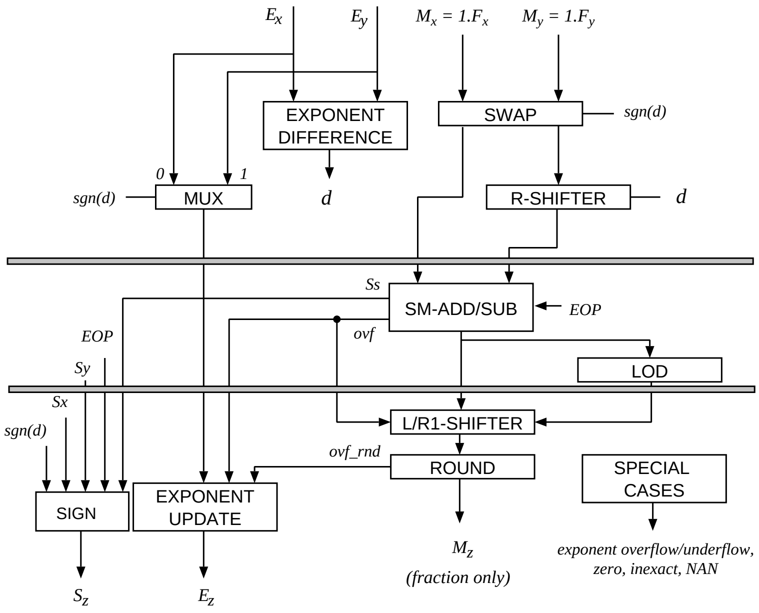


Figure E8.2: Pipelined implementation of the floating-point adder in Figure 8.5 for single precision.

The floating-point adder for double precision could be pipelined as follows (4 stages):

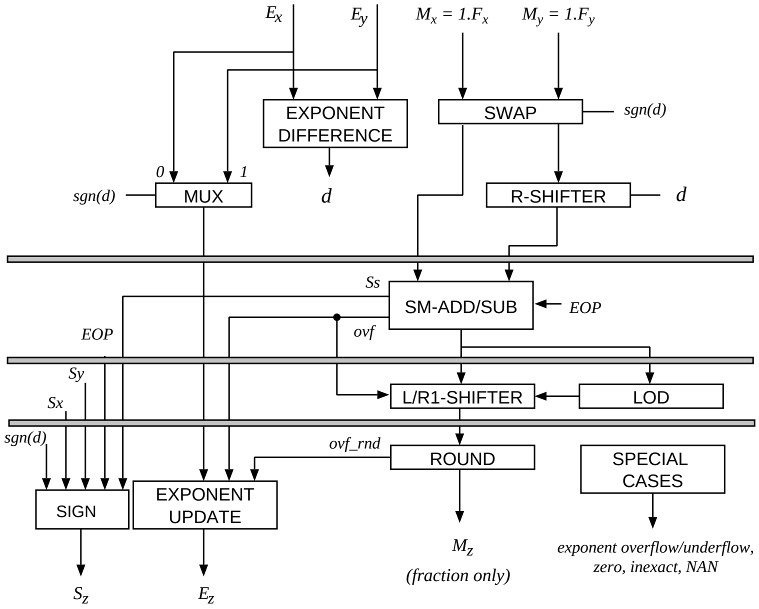


Figure E8.3: Pipelined implementation of the floating-point adder in Figure 8.5 for double precision.

Exercise 8.23

	Operation	X	Y
A	Add	000110001001111000	001001100100011101
B	Sub	000110001001111000	101001100100011101
C	Sub	000110001001111000	000110001001110111
D	Sub	011111110111100011	111111100001010101

(A) *EOP is ADD*

Output of blocks in Fig. 8.8	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	27
$OUTPUT_{MUX}$	01001100
$INPUT_{R1-SHIFTER}$	
$OUTPUT_{R1-SHIFTER}$	
$INPUT_{R-SHIFTER}$	1.001111000
$OUTPUT_{R-SHIFTER}$	0.000000000 001
$OUTPUT_{COND.BIT\ INVERT}$	0.000000000 001
$OUTPUT_{INVERT,ADD,ROUND\&INVERT}$	
$OUTPUT_{L-SHIFTER}$	

$OUTPUT_{ADD,ROUND\&NORMALIZE}$
$RNE(Sum)$
$RNE(Sum + one)$
$Normalized$
	1.100011101 001
	1.100011101
	
	1.100011101

$OUTPUT_{MUX}$	1.100011101
$OUTPUT_{EXPONENT\ UPDATE}$	01001100
$OUTPUT_{SIGN}$	0

(B) *EOP is SUB*

Output of blocks in Fig. 8.8	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	27
$OUTPUT_{MUX}$	01001100
$INPUT_{R1-SHIFTER}$	
$OUTPUT_{R1-SHIFTER}$	
$INPUT_{R-SHIFTER}$	1.001111000
$OUTPUT_{R-SHIFTER}$	0.000000000 001
$OUTPUT_{COND.BIT\ INVERT}$	1.111111111 001
$OUTPUT_{INVERT,ADD,ROUND\&INVERT}$	
$OUTPUT_{L-SHIFTER}$	

$OUTPUT_{ADD,ROUND\&NORMALIZE}$
$RNE(Sum)$
$RNE(Sum + one)$
$Normalized$
	1.100011101 001
	1.100011101
	
	1.100011101

$OUTPUT_{MUX}$	1.100011101
$OUTPUT_{EXPONENT\ UPDATE}$	01001100
$OUTPUT_{SIGN}$	1

(C) *EOP is SUB*

Output of blocks in Fig. 8.8	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	0
$OUTPUT_{MUX}$	00110001
$INPUT_{R1-SHIFTER}$	1.001110111
$OUTPUT_{R1-SHIFTER}$	1.001110111
$INPUT_{R-SHIFTER}$	
$OUTPUT_{R-SHIFTER}$	
$OUTPUT_{COND.BIT\ INVERT}$	
$OUTPUT_{INVERT,ADD,ROUND\&INVERT}$	0.000000001
$OUTPUT_{L-SHIFTER}$	1.000000000
$OUTPUT_{ADD,ROUND\&NORMALIZE}$ $RNE(Sum)$ $RNE(Sum + one)$ $Normalized$	
$OUTPUT_{MUX}$	1.000000000
$OUTPUT_{EXPONENT\ UPDATE}$	00101000
$OUTPUT_{SIGN}$	0

(D) *EOP is ADD*

Output of blocks in Fig. 8.8	Value
$OUTPUT_{EXPONENT\ DIFFERENCE}$	2
$OUTPUT_{MUX}$	11111110
$INPUT_{R1-SHIFTER}$	
$OUTPUT_{R1-SHIFTER}$	
$INPUT_{R-SHIFTER}$	1.001010101
$OUTPUT_{R-SHIFTER}$	0.010010101 010
$OUTPUT_{COND.BIT\ INVERT}$	0.010010101
$OUTPUT_{INVERT,ADD,ROUND\&INVERT}$	
$OUTPUT_{L-SHIFTER}$	
$OUTPUT_{ADD,ROUND\&NORMALIZE}$ $RNE(Sum)$ $RNE(Sum + one)$ $Normalized$	10.001111000 10.001111000 1.000111100 $E_z = 255 \rightarrow M_z = 0$ (overflow)
$OUTPUT_{MUX}$	1.000111100 $E_z = 255 \rightarrow M_z = 0$ (overflow)
$OUTPUT_{EXPONENT\ UPDATE}$	11111111
$OUTPUT_{SIGN}$	0

Exercise 8.25

Operation	X	Y
A	001010101010110011	101111111101110011
B	110011110101110010	111000111011111100

(A) $S_z = 1$

$OUTPUT_{EXP.\ BIASED\ ADDITION}$: 01010101

$OUTPUT_{m\ by\ m\ MULTIPLIER}$: $P[-1, 2m-2] = 10.010101000000110001$

$P[-1] = 1 \Rightarrow$ normalize by shifting right by one (exponent must be incremented by one).

$OUTPUT_{NORMALIZE}$: 1.00101010 0 01

L GT

Rounding: RNE (round down), RZ (round down), RPINF(round down), RMINF(round up).

$OUTPUT_{EXPONENT\ UPDATE}$: 01010110

(B) $S_z = 0$

$OUTPUT_{EXP.\ BIASED\ ADDITION}$: 11100110

$OUTPUT_{m\ by\ m\ MULTIPLIER}$: $P[-1, 2m-2] = 10.100100100000111000$

$P[-1] = 1 \Rightarrow$ normalize by shifting right by one (exponent must be incremented by one).

$OUTPUT_{NORMALIZE}$: 1.01001001 0 01

L GT

Rounding: RNE (round down), RZ (round down), RPINF(round up), RMINF(round down).

$OUTPUT_{EXPONENT\ UPDATE}$: 11100111

Exercise 8.29

Operation	X	Y
A	001010101010111000	001010101010010000
B	010000000001100000	001010101011000000

(A) Performing the computation of the multiplication using the basic implementation, the output of *m by m MULTIPLIER* block: is $P[-1, 2m-2]$ = 01.101111011110000000. Since $P[-1] = 0$, $T = 1$. To determine the value of the sticky bit directly from the operands of the multiplier we have to compute the sum of the number of trailing zeros of X and Y (that is, 3 + 4 = 7). Since no normalization is required, we can say that not all the discarded bits are zeros and, as a consequence, $T = 1$, as expected. If we want to compute the value of the sticky bit using the carry-save representation of the second half of the product, we need

$PC[-1 : 2m-3] = 00.10111100000000000$ and

$PS[-1 : 2m-2] = 01.000000011110000000$.

$PC[m+1 : 2m-3] = 0000000$ and

$PS[m+1 : 2m-2] = 10000000$.

```
10000000 s
00000000 c
01111111 z
0000000  t
0111111  w
```

Therefore, $T = NAND(w_i) = 1$ as expected.

(B) Performing the computation of the multiplication using the basic implementation, the output of m *by* m $MULTIPLIER$ block: is $P[-1, 2m-2]$ =01.101000100000000000. Since $P[-1] = 0$, $T = 0$. To determine the value of the sticky bit directly from the operands of the multiplier we have to compute the sum of the number of trailing zeros of X and Y (that is, $5 + 6 = 11$). Since no normalization is required, we can say that all the discarded bits are zeros and, as a consequence, $T = 0$, as expected. If we want to compute the value of the sticky bit using the carry-save reresentation of the second half of the product, we need

$PC[-1 : 2m - 3] = 00.01001000000000000$ and

$PS[-1 : 2m - 2] = 01010110100000000000$.

$PC[m + 1 : 2m - 3] = 0000000$ and

$PS[m + 1 : 2m - 2] = 00000000$.

```
00000000 s
00000000 c
11111111 z
0000000  t
1111111  w
```

Therefore, $T = NAND(w_i) = 0$ as expected.

Exercise 8.31

(a) *Round to zero*

For rounding to zero, the result is simply truncated to m bits and no additional operation is required.

(b) *Round to plus infinity*

$$R_{pinf} = \begin{array}{lll} M_f + r^{-f} & \text{if} & M_d > 0 \text{ and } S = 0 \\ M_f & \text{if} & M_d = 0 \text{ or } S = 1 \end{array}$$

In this case, a 1 should be added to position R (bit m) if $S = 0$ (where S is the sign of the result) and $M_d > 0$ (that is if the sticky bit $T = 1$). However, the result can be either normalized or unnormalized, while the rounding if performed before knowing whether the result is normalized. Therefore, the following quantities have to be calculated:

$$\begin{aligned} P0 &= PM + \left(c_m + \bar{S} \cdot T\right) \times 2^{-m} \\ P1 &= PM + \left(c_m + \bar{S} \cdot T + 1\right) \times 2^{-m} \end{aligned}$$

up to position L (bit $m-1$).

The rounded result is obtained by selecting

$$P = \begin{array}{lll} P0 & \text{if} & P0[-1] = 0 \\ 2^{-1}P1 & \text{if} & P0[-1] = 1 \end{array}$$

that is if there is no overflow, select $P0$, while if there is overflow, select $P1$, shift right and truncate at resulting bit L.

Proof

In all cases cm needs to be added to position R (bit m). In case there is no overflow the result is truncated at position L. In the following cases a 1 needs to be added to position L:

- $\bar{S} \cdot T = 1$
- $\bar{S} \cdot T = 0$ and bit of sum in position $R = 1$

Both cases are accounted for by adding $\bar{S} \cdot T + 1$ in position R. In case there is overflow the result is truncated at bit $L-1$ and shifted one bit right. Before shifting a 1 needs to be added to position $L-1$ in the following situations:

- $\bar{S} \cdot T = 1$
- $\bar{S} \cdot T = 0$ and bit of sum in position L or $R = 1$

All cases are accounted for by adding $\bar{S} \cdot T + 1$ in position R and selection $P0+1$ in case of overflow. This is because if $\bar{S} \cdot T = 1$ adding 2 to position R corresponds to adding 1 to position L, so selection $P0+1$ corresponds to adding 2 to position L or 1 to $L-1$. On the contrary, if $\bar{S} \cdot T = 0$, if $R = 1$ then when 1 is added to R there is a carry to position L, so 1 is added to L, while if $R = 0$ and $L = 1$ then adding 1 to $P0$ produces a carry to bit $L-1$ so that $P0+1$ truncated to bit $l-1$ corresponds to adding 1 to bit $L-1$.

The implementation consists of an array of HAs and FAs, which adds 1 to 3 to position R (that is, add $\overline{\left(c_m \oplus \bar{S} \cdot T\right)}$ to bit R and $\left(c_m + \bar{S} \cdot T\right)$ to bit L), a compound adder producing P0 and $P0+1$, The complete process then requires a row of HAs and FAs, a compound adder that computes the sum $P0$ and the sum plus 1 and a multiplexer which selects $P0$ or the normalized (shifted) $P1$ depending whether $P0$ overflows or not.

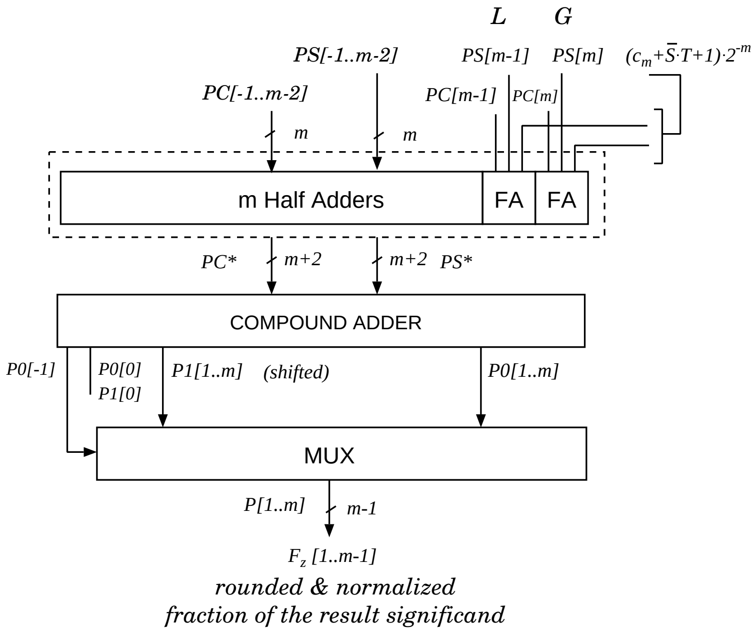


Figure E8.6: Alternative implementation modified to perform round to plus infinity.

(c) *Round to minus infinity*

$$R_{minf} = \begin{array}{lll} M_f + r^{-f} & \text{if} & M_d > 0 \text{ and } S = 1 \\ M_f & \text{if} & M_d = 0 \text{ or } S = 0 \end{array}$$

The algorithm for rounding to minus infinity is therefore the same used for rounding to plus infinity, except that $\overline{S}$ should be substituted with S.

Exercise 8.34

X	Y	W
001010101010110011	101111111101110011	110011110101110010

Output of the m *by* m $MULTIPLIER\ CS$:

PS 01.101000001101101001
PC 00.101100110000000000

Computing $d = -42 + 0 - 31 + m + 3$ we get $d = -60$ (since $m = 10$). Therefore no right shift is needed and the output of the $RIGHT\ SHIFTER$ block is 1.101110010.

```
PS                            01.101000001101101001
PC                            00.101100110000000000
Addend         1.101110010 00
--------------------------------------------------
S              1.101110010 00 01 000100111101101001
C              0.000000000 00 01 010000000000000000
Adder output   1.101110010 00 10 010100111101101001
                         L GR T
```

The output of the adder does not require any realignement/normalization left shift since it is already normalized (leading 1 in the left most position).

Rounding mode	
RNE	Round down
RZ	Round down
RPINF	Round down
RMINF	Round up

The output of the $EXPONENT\ UPDATE$ block is $max(E_x + E_y, E_w) = E_w$. Finally, the result is negative ($S_z = 1$).

Exercise 8.38

	X	Y
A	001010101011010011	101111111110110011
B	110011110001011010	111000111101011101

(A) Let Q be the result. The sign and exponent of the result are then

$$\begin{aligned} S_q &= S_x \otimes S_d = 1 \\ E_q &= E_x - E_d + 127 = 01010101 \end{aligned}$$

The significand of the result is then calculated as

$$M_q = \frac{M_x}{M_d} = \frac{1.011010011}{1.110110011} = \frac{0.1011010011}{0.1110110011}$$

The last conversion is necessary in order to be able to use the quotient-digit selection function of the implementation presented in Section 5.3.1. Since $n = 10$, the number of iterations to be performed is $n+2 = 12$. The initialization is as follows:

$$\text{scaled residual } 2w[0] = 2(x/2) = x, \; q_{computed} = q/2$$

2WS[0]	000.1011010011		
2WC[0]	000.0000000001	$\hat{y}[0]$=0.5	$q_1 = 1$
$-q_1 d$	111.0001001100		
2WS[1]	111.0100111100		
2WC[1]	000.0100000100	$\hat{y}[1]$=-1/2	$q_2 = 0$
$-q_2 d$	000.0000000000		
2WS[2]	110.0001110000		
2WC[2]	001.0000010000	$\hat{y}[2]$=-1	$q_3 = -1$
$-q_3 d$	000.1110110011		
2WS[3]	111.1110100110		
2WC[3]	000.0011000001	$\hat{y}[3]$=0	$q_4 = 1$
$-q_4 d$	111.0001001100		
2WS[4]	001.1001010110		
2WC[4]	100.1100010000	$\hat{y}[4]$=-1	$q_5 = -1$
$-q_5 d$	000.1110110011		
2WS[5]	011.0111101010		
2WC[5]	011.0001001000	$\hat{y}[5]$=-2	$q_6 = -1$
$-q_6 d$	000.1110110011		
2WS[6]	001.0000100010		
2WC[6]	101.1110101000	$\hat{y}[6]$=-1	$q_7 = -1$
$-q_7 d$	000.1110110011		
2WS[7]	000.0001110010		
2WC[7]	111.1010001000	$\hat{y}[7]$=-1/2	$q_8 = 0$
$-q_8 d$	000.0000000000		
2WS[8]	111.0111110100		
2WC[8]	000.0000000000	$\hat{y}[8]$=-1/2	$q_9 = 0$
$-q_9 d$	000.0000000000		
2WS[9]	110.1111101000		
2WC[9]	000.0000000000	$\hat{y}[9]$=-3/2	$q_{10} = -1$
$-q_{10} d$	000.1110110011		
2WS[10]	100.0010110110		
2WC[10]	011.1010000000	$\hat{y}[10]$=-1/2	$q_{11} = 0$
$-q_{11} d$	000.0000000000		
WS[11]	111.1000110110		
WC[11]	000.0100000000	$\hat{y}[11]$=-1/2	$q_{12} = 0$

Since the last residual is negative, the last bit has to be corrected, therefore $q_{12} = -1$. The computed result is then, which however has to be shifted left 1 position since the computed result is $q/2$. The significand before normalization and rounding is then M_q=0.11000011011.

After normalization (M_q=1.1000011011 and E_q=01010100) the result has $f+1$ fractional bits. For round-to-nearest, $2^{-(f+1)}$ has to be added to the result; therefore the rounded significand is

```
1.1000011011 +
0.0000000001
------------
1.1000011100
```

The final result expressed in the IEEE Standard format is

$$Q = 0|01010100|1000011100$$

(B) Let Q be the result. The sign and exponent of the result are then

$$\begin{array}{rcl} S_q & = & S_x \otimes S_d = 0 \\ E_q & = & E_x - E_d + 127 = 11010110 \end{array}$$

The significand of the result is then calculated as

$$M_q = \frac{M_x}{M_d} = \frac{1.001011010}{1.101011101} = \frac{0.1001011010}{0.1101011101}$$

The last conversion is necessary in order to be able to use the quotient-digit selection function of the implementation presented in Section 5.3.1. Since $n = 10$, the number of iterations to be performed is $n+2 = 12$. The initialization is as follows:

$$\text{scaled residual } 2w[0] = 2(x/2) = x,\ q_{computed} = q/2$$

2WS[0]	000.1001011010		
2WC[0]	000.0000000001	$\hat{y}[0]$=0.5	$q_1 = 1$
$-q_1d$	111.0010100010		
2WS[1]	111.0111110010		
2WC[1]	000.0000001000	$\hat{y}[1]$=-1/2	$q_2 = 0$
$-q_2d$	000.0000000000		
2WS[2]	110.1111110100		
2WC[2]	000.0000000000	$\hat{y}[2]$=-1	$q_3 = -1$
$-q_3d$	000.1101011101		
2WS[3]	100.0101010010		
2WC[3]	011.0101010000	$\hat{y}[3]$=-1/2	$q_4 = 0$
$-q_4d$	000.0000000000		
2WS[4]	110.0000000100		
2WC[4]	001.0101000000	$\hat{y}[4]$=-1/2	$q_5 = 0$
$-q_5d$	000.0000000000		
2WS[5]	110.1010001000		
2WC[5]	000.0000000000	$\hat{y}[5]$=-1	$q_6 = -1$
$-q_6d$	000.1101011101		
2WS[6]	100.1110101010		
2WC[6]	010.0000100000	$\hat{y}[6]$=-1	$q_7 = -1$
$-q_7d$	000.1101011101		
2WS[7]	100.0110101110		
2WC[7]	011.0010100000	$\hat{y}[7]$=-1/2	$q_8 = 0$
$-q_8d$	000.0000000000		
2WS[8]	110.1000011100		
2WC[8]	000.1010000000	$\hat{y}[8]$=-1/2	$q_9 = 0$
$-q_9d$	000.0000000000		
2WS[9]	100.0100111000		
2WC[9]	010.0000000000	$\hat{y}[9]$=-1	$q_{10} = -1$
$-q_{10}d$	000.1101011101		
2WS[10]	101.0011001010		
2WC[10]	001.0001100000	$\hat{y}[10]$=-1	$q_{11} = -1$
$-q_{11}d$	000.1101011101		
WS[11] =	100.1111110111		
WC[11] =	010.0010010000	$\hat{y}[11]$=-1	$q_{12} = -1$

The computed result is then

$$q = .10\bar{1}00\bar{1}\bar{1}00\bar{1}\bar{1}\bar{1} = .010110011001$$

which has to be corrected by subtracting one in the last position since the last residual is negative and thus

$$q = .010110011000$$

Moreover, the result has to be shifted left 1 position since the computed result is $q/2$. The significand before normalization and rounding is then $M_q = 0.10110011000$. After normalization ($M_q = 1.0110011000$ and $E_q = 11010101$) the result has $f+1$ fractional bits. For round-to-nearest,$2^{-(f+1)}$ has to be added to the result; therefore the rounded significand is

```
1.0110011000 +
0.0000000001
------------
1.0110011001
```

The final result expressed in the IEEE Standard format is

$$Q = 0|11010101|011001100$$

Exercise 8.41

	X	Y
A	001010101011010011	101111111110110011
B	110011110001011010	111000111101011101

(A) Let Q be the result. The sign and exponent of the result are then

$$\begin{array}{rcl} S_q & = & S_x \otimes S_d = 1 \\ E_q & = & E_x - E_d + 127 = 10011111 \end{array}$$

The significand of the result is then calculated as

$$M_q = \frac{M_x}{M_d} = \frac{1.011010011}{1.110110011}$$

The method requires the calculation of an initial approximation of the reciprocal of the divisor (of 4 bits in this case), which can be obtained, for instance, by means of a lookup table. The initial approximation is 0.1000. The number of iterations to be performed is then

$$m = \left\lceil \log_2 \left(\frac{n}{k} \right) \right\rceil = \left\lceil \log_2 \left(\frac{9}{4} \right) \right\rceil = 2$$

Since this algorithm is not self-correcting, all multiplications are performed using a 16 bits multiplier. The algorithm is as follows (assuming that multiplications are performed using a floating-point multiplier with rounding to nearest):

1. $P[0] = 0.1000$ (initial approximation of $1/d$)
2. $d[0] = d \times P[0] = 1.110110011000000 \times 2^{-1}$
   $R[0] = x \times P[0] = 1.011010011000000 \times 2^{-1}$
3. $P[1] = 2 - d[0] = 1.000100110100000 \times 2^{0}$
   $d[1] = d[0] \times P[1] = 1.111111010001101 \times 2^{-1}$
   $R[1] = R[0] \times P[1] = 1.100001001010111 \times 2^{-1}$
4. $P[2] = 2 - d[1] = 1.000000010111010 \times 2^{0}$
   $R[2] = R[1] \times P[2] = 1.100001101110001 \times 2^{-1}$

The final q, rounded to the final number of bit, is then 1.100001110. The final result expressed in the IEEE Standard format is

$$Q = 0|01010100|100001110$$

(B) Let Q be the result. The sign and exponent of the result are then

$$\begin{aligned} S_q &= S_x \otimes S_d = 0 \\ E_q &= E_x - E_d + 127 = 01111100 \end{aligned}$$

The significand of the result is then calculated as

$$M_q = \frac{M_x}{M_d} = \frac{1.001011010}{1.101011101}$$

The method requires the calculation of an initial approximation of the reciprocal of the divisor (of 4 bits in this case), which can be obtained, for instance, by means of a lookup table. The initial approximation is 0.1001. The number of iterations to be performed is then

$$m = \left\lceil \log_2 \left(\frac{n}{k}\right) \right\rceil = \left\lceil \log_2 \left(\frac{9}{4}\right) \right\rceil = 2$$

Since this algorithm is not self-correcting, all multiplications are performed using a 16 bits multiplier. he algorithm is as follows (assuming that multiplications are performed using a floating-point multiplier with rounding to nearest):

1. $P[0] = 0.1001$ (initial approximation of $1/d$)
2. $d[0] = d \times P[0] = 1.111001000101000 \times 2^{-1}$
   $R[0] = x \times P[0] = 1.010100101010000 \times 2^{-1}$
3. $P[1] = 2 - d[0] = 1.000011011101100 \times 2^0$
   $d[1] = d[0] \times P[1] = 1.111111101000000 \times 2^{-1}$
   $R[1] = R[0] \times P[1] = 1.011001001111000 \times 2^{-1}$
4. $P[2] = 2 - d[1] = 1.000000001100000 \times 2^0$
   $R[2] = R[1] \times P[2] = 1.011001011111110 \times 2^{-1}$

The final q, rounded to the final number of bit, is then 1.011001100. The final result expressed in the IEEE Standard format is

$$Q = 0|11010101|011001100$$

Exercise 8.44

Round to nearest is performed by adding $2^{-(\mathrm{f}+1)}$ and truncating to f bit. Overflow can occur if $q + 2^{-(f+1)} \geq 2$.

Since the normalized significand is in the range $1 \leq 1.F \leq 2 - 2^{-f}$, the quotient is comprised in the range $\frac{1}{2-2^{-f}} \leq q \leq \frac{2-2^{-f}}{1}$.

Therefore we obtain $q \leq 2 - 2^{-f} \Rightarrow q + 2^{-(f+1)} \leq 2 - 2^{-f} + 2^{-(f+1)} = 2 - 2^{-(f+1)} < 2$. Since $q + 2^{-(f+1)} < 2$, the overflow condition is never satisfied.

DIGITAL ARITHMETIC
Miloš D. Ercegovac and Tomás Lang
Morgan Kaufmann Publishers, an imprint of Elsevier Science, ©2004
– Updated: December 20, 2003 –

Chapter 11: Solutions to Exercises

Exercise 11.1

Compute $\sin(30^o)$ and $\cos(30^o)$ to a precision of seven bits 7 using the CORDIC algorithm.

The number of iterations performed depends on the datapath width, so that the angle becomes 0 for that width.

a) Datapath width of 7 fractional bits. We perform 7 iterations.

j	$z[j]$	σ_j	α_j	$x[j]$	$y[j]$
0	0.1000011	1	0.1100100	0.1001101	0.0000000
1	-0.0100001	-1	0.0111011	0.1001101	0.1001101
2	0.0011010	1	0.0011111	0.1110011	0.0100111
3	-0.0000101	-1	0.0001111	0.1101010	0.1000011
4	0.0001010	1	0.0000111	0.1110010	0.0110110
5	0.0000011	1	0.0000011	0.1101111	0.0111101
6	0.0000000	1	0.0000001	0.1101110	0.1000000
7				0.1101101	0.1000001

The angle decomposition is in radians. Values given in sign and magnitude.

The errors are $|\cos^o(30) - x[7]| = |0.866 - 0.852| = 0.014$ and $|\sin(30^o) - y[7]| = |0.5 - 0.508| = 0.008$

b) Datapath width of 10 fractional bits:

j	$z[j]$	σ_j	α_j	$x[j]$	$y[j]$
0	0.1000011000	1	0.1100100100	0.1001101101	0.0000000000
1	-0.0100001100	-1	0.0111011010	0.1001101101	0.1001101101
2	0.0011000111	1	0.0011111010	0.1110100011	0.0100110111
3	-0.0000101100	-1	0.0001111111	0.1101010110	0.1000011111
4	0.0001010011	1	0.0000111111	0.1110011001	0.0110110101
5	0.0000010100	1	0.0000011111	0.1101111111	0.0111101111
6	-0.0000001011	-1	0.0000001111	0.1101101111	0.1000001001
7	0.0000000100	1	0.0000000111	0.1101110111	0.0111111100
8	-0.0000000011	-1	0.0000000011	0.1101110100	0.1000000010
9	0.0000000000	1	0.0000000001	0.1101110110	0.0111111111
10				0.1101110101	0.1000000000

The result truncated to 7 fractional bits is

$$x[10] = 0.1101110 = 0.8594 \quad y[10] = 0.1000000 = 0.5$$

The errors are $|\cos(30^o) - x[10]| = |0.866 - 0.859| = 0.007$ and $|\sin(30^o) - y[10]| = |0.5 - 0.5| = 0$

c) we have not found a systematic solution method.

Exercise 11.3

The number of iterations performed depends on the datapath width, so that the last α_i becomes 0 for that width.

a) Datapath width of 7 fractional bits. We perform 7 iterations.

j	$y[j]$	σ_j	α_j	$z[j]$	$x[j]$
0	10.0010000	-1	0.1100100	0.0000000	11.0100000
1	-01.0010000	1	0.0111011	0.1100100	101.0110000
2	01.1001000	-1	0.0011111	0.0101001	101.1111000
3	00.0001010	-1	0.0001111	0.1001000	110.0101010
4	-00.1011011	1	0.0000111	0.1010111	110.0101011
5	-00.0101001	1	0.0000011	0.1010000	110.0110000
6	-00.0010000	1	0.0000001	0.1001101	110.0110001
7	-00.0000100			0.1001100	110.0110001

The angle decomposition is in radians. Values given in sign and magnitude.

The result values are $z[7] = 0.101100 = 0.580$ and $x[7] = 110.0110001 = 6.3828$.The compensated value is $x_R = x[7] \times 1/K[7] = 6.3828 \times 0.6072 = 3.876$.

The errors are $|\tan^{-1}(2.13/3.25) - z[7]| = |0.580 - 0.594| = 0.014$ and $modulus(2.13, 3.25) - x_R = 3.8856 - 3.876 = 0.009$

b) Datapath width of 10 fractional bits. We perform 10 iterations.

j	$y[j]$	σ_j	α_j	$z[j]$	$x[j]$
0	10.0010000101	-1	0.1100100100	0.0000000000	11.0100000000
1	-01.0001111011	1	0.0111011010	0.1100100100	101.0110000101
2	01.1001000111	-1	0.0011111010	0.0101001010	101.1111000010
3	00.0001010111	-1	0.0001111111	0.1001000100	110.0101010011
4	-00.1011010011	1	0.0000111111	0.1011000011	110.0101011101
5	-00.0100111110	1	0.0000011111	0.1010000100	110.0110001010
6	-00.0001110010	1	0.0000001111	0.1001100101	110.0110010011
7	-00.0000001100	1	0.0000000111	0.1001010110	110.0110010100
8	00.0000100111	-1	0.0000000011	0.1001001111	110.0110010100
9	00.0000001110	-1	0.0000000001	0.1001010010	110.0110010100
10	00.0000000010			0.1001010011	110.0110010100

The result truncated to 7 fractional bits is

$$z[10] = 0.1001010 = 0.578 \quad x[10] = 110.0110010 = 6.391$$

We compensate $x_R = x[10] \times 1/K[10] = 6.391 \times 0.6072 = 3.8806$

The errors are $|\tan^{-1}(2.13/3.25 - z[10]| = |0.580 - 0.578| = 0.002$ and $|modulus(2.13, 3.25) - x[10]| = 3.8856 - 3.8806 = 0.005$.

c) we have not found a systematic method to get a solution.

Exercise 11.4

Note that the sequence of α's should be decreasing. That is,

$$\alpha_{i+1} < \alpha_i \leq 2\alpha_{i+1}$$

From the definition of A and the values of s_i, we obtain that the range of A is

$$0 \leq A \leq A_{max} = \sum_{i=0}^{\infty} \alpha_i \tag{1}$$

The recurrent algorithm using $s_i \in \{0, 1\}$ converges iff for all j the residual value

$$W[j] = A - \sum_{i=0}^{j} s_i \alpha_i$$

is bounded by

$$0 \leq W[j] \leq \sum_{i=j+1}^{\infty} \alpha_i \tag{2}$$

From (1) and (2) we see that the algorithm converges while the values of s_i are all 1. Consider therefore the value $i = k$ for which the first $s_i = 0$ is selected. In the iteration

$$W[k+1] = W[k] - s_k \alpha_k$$

to have a non-negative residual $W[k+1]$, we need to make $s_{j+1} = 0$ when $W[k] \leq \alpha_k - ulp$. For the largest value $(\alpha_k - ulp)$ we get $W[k+1] = \alpha_k - ulp$. Moreover, to have convergence, from (2) we have

$$\alpha_k - ulp \leq \sum_{j=k+1}^{\infty} \alpha_j = \alpha_{k+1} + \sum_{j=k+2}^{\infty} \alpha_j \tag{3}$$

Now from the hypothesis $\alpha_i \leq 2\alpha_{i+1}$ we obtain

$$\alpha_i \leq \sum_{j=i+1}^{\infty} \alpha_j \tag{4}$$

This results from the well-known fact that if $a_i = 2a_{i+1}$ then $a_i = \sum_{j=i+1}^{\infty} a_j$. Introducing (4) in (3) we conclude that the algorithm converges.

For $s_i\{-1, 1\}$ we apply the same technique. Now the convergence condition is

$$|W[j]| \leq \sum_{i=j+1}^{\infty} \alpha_i$$

Again, the algorithm converges while $s_j = 1$. We choose $s_j = -1$ when $W[j] < 0$. The most negative value of $W[j]$ occurs when $W[j-1] = 0$. Consequently,

$$W[j] \geq -\alpha_{j-1}$$

So, for convergence,

$$\alpha_{j-1} \leq \sum_{i=j}^{\infty} \alpha_i$$

and the same proof as before follows.

Exercise 11.9

The Taylor series expansion of $\tan^{-1}(2^{-j})$ is

$$\tan^{-1}(2^{-j}) = 2^{-j} - \frac{2^{-3j}}{3} + \frac{2^{-5j}}{5} - ...$$

Consequently,

$$|\tan^{-1}(2^{-j}) - 2^{-j}| = \frac{2^{-3j}}{3} - \frac{2^{-5j}}{5} + ... \leq 2^{-n}$$

results in

$$j \geq J = \frac{n-1}{3}$$

This implies that for $j \geq J$ there is no need to store the value of $\tan^{-1}(2^{-j})$ in the table, since the value 2^{-j} can be used.

Exercise 11.12

According to Table 11.5 we perform hyperbolic CORDIC in vectoring mode with initial conditions $x_{in} = 1.17$, $y_{in} = -0.83$, and $z_{in} = 0$. Performing eight iterations with a datapath width of 8 bits, we obtain

j	$y[j]$	σ_j	α_j	$z[j]$	$x[j]$
1	-0.11010100	1	0.10001100	0.00000000	1.00101011
2	-0.00111111	1	0.01000001	-0.10001100	0.11000001
3	-0.00001111	1	0.00100000	-0.11001101	0.10110010
4	0.00000111	-1	0.00010000	-0.11101101	0.10110001
4	-0.00000100	1	0.00010000	-0.11011101	0.10110001
5	0.00000111	-1	0.00001000	-0.11101101	0.10110001
6	0.00000010	-1	0.00000100	-0.11100101	0.10110001
7	0.00000000	-1	0.00000010	-0.11100001	0.10110001
8	-0.00000001	1	0.00000001	-0.11011111	0.10110001
9	-0.00000001	-	0.00000000	-0.11100000	0.10110001

The result is $2z[10] = -1.11000000 = -1.75$. The error is $|\ln(0.17) - 2z[10]| = |-1.772 + 1.75| = 0.022$