HUMBOLDT-UNIVERSITÄT ZU BERLIN
INSTITUT FÜR INFORMATIK

Lecture 12

ALU (3) - Division Algorithms

Sommersemester 2002

Leitung: Prof. Dr. Miroslaw Malek

www.informatik.hu-berlin.de/rok/ca

FIXED-POINT, FLOATING-POINT ARITHMETIC AND LOGICAL FUNCTIONS

A. FIXED-POINT ARITHMETIC (continued)

- RESTORING DIVISION
- NONRESTORING DIVISION
- DIVISION BY REPEATED MULTIPLICATION
- DIVISION BY USING RECIPROCAL (CRAY)

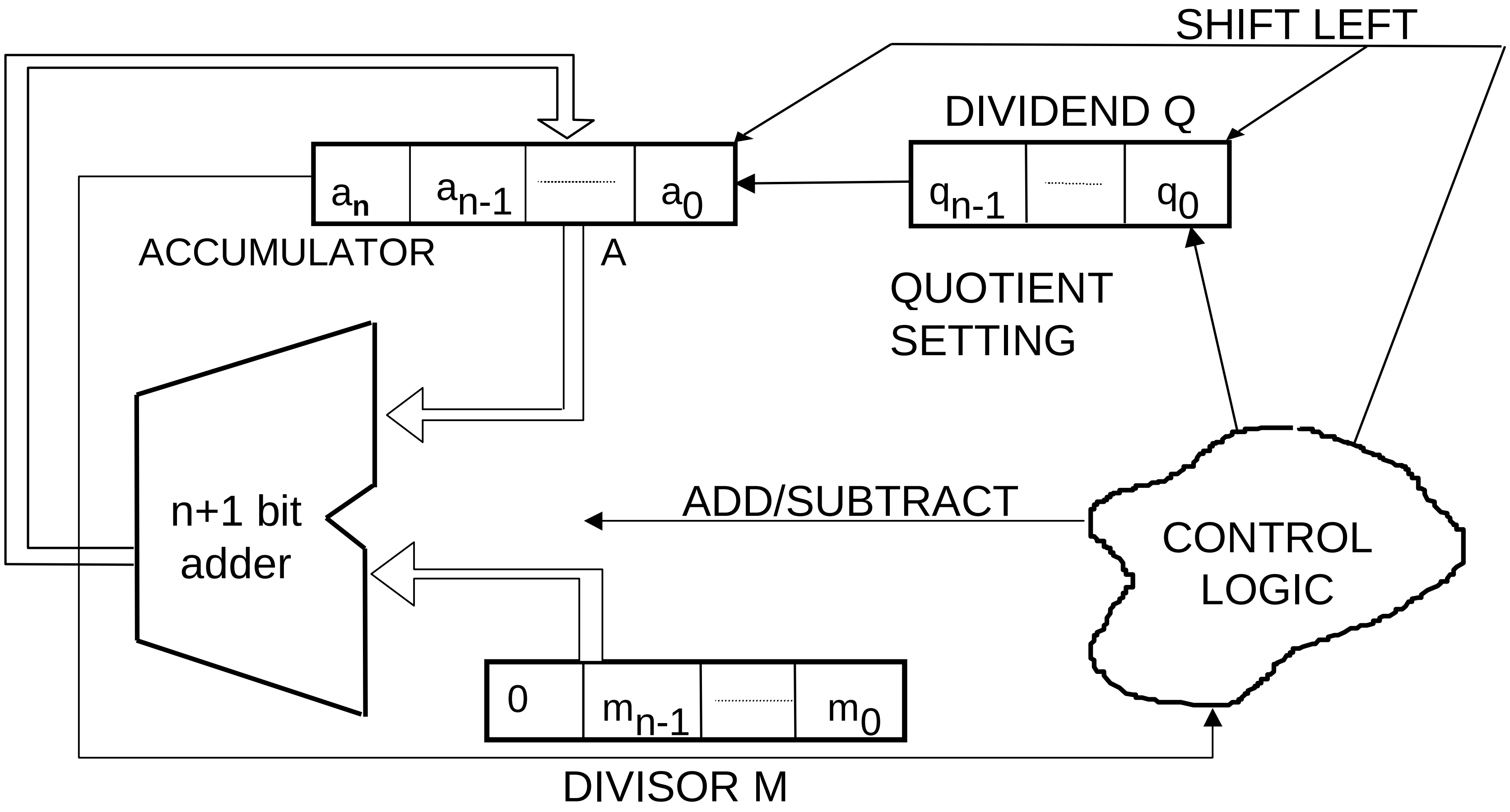
B. FLOATING-POINT ARITHMETIC

- FORMATS
- ADDITION/SUBTRACTION
- MULTIPLICATION/DIVISION

C. LOGIC FUNCTIONS

ALGORITHM FOR RESTORING DIVISION

- DO *n* TIMES
- SHIFT A & Q LEFT ONE BINARY POSITION
- SUBTRACT M FROM A, PLACING THE ANSWER BACK IN A
- IF THE SIGN OF A IS 1, SET q_0 TO 0 AND ADD M BACK TO A (RESTORE A);
- OTHERWISE, SET q_0 TO 1

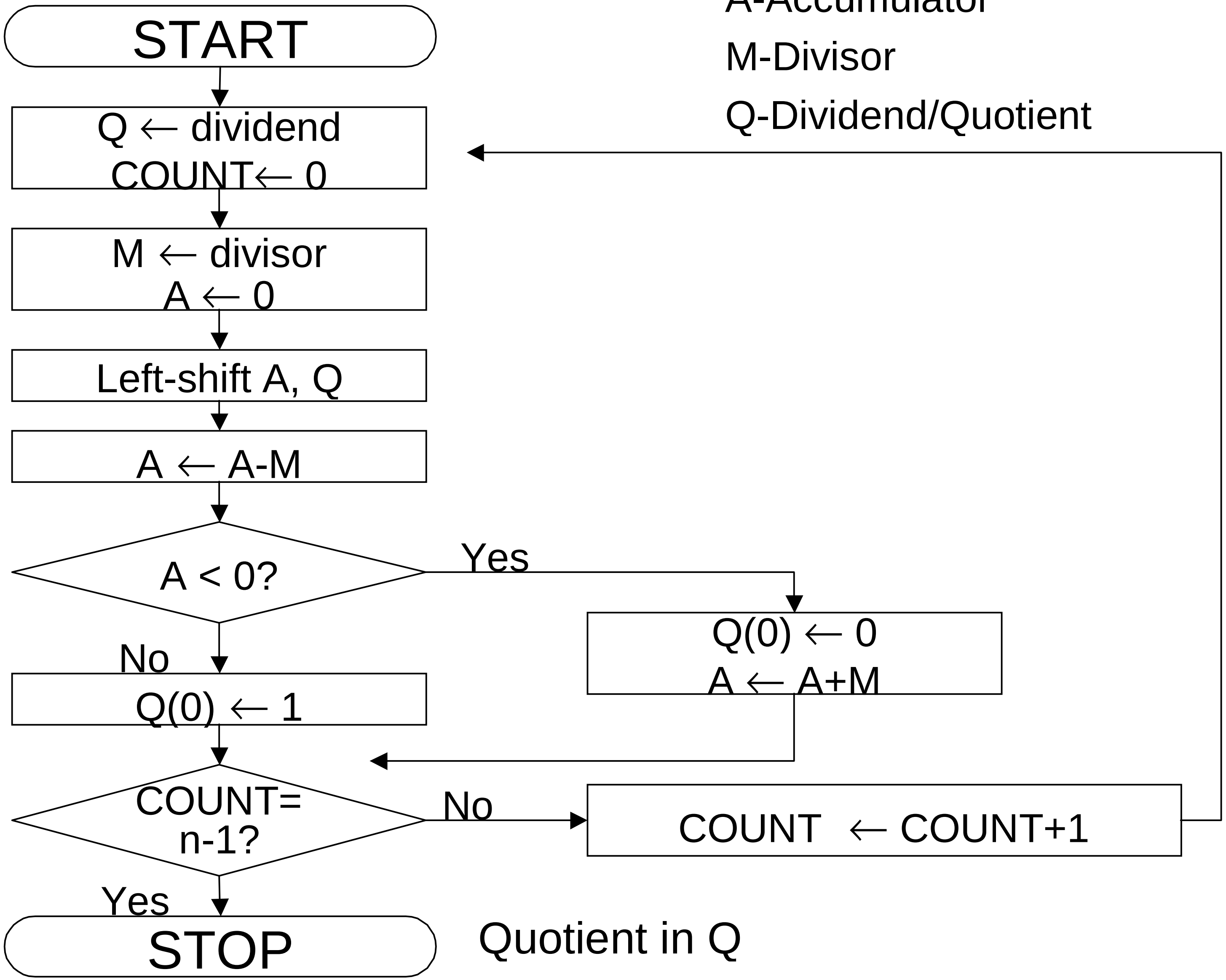


RESTORING DIVISION ALGORITHM FOR POSITIVE INTEGERS

A-Accumulator

M-Divisor

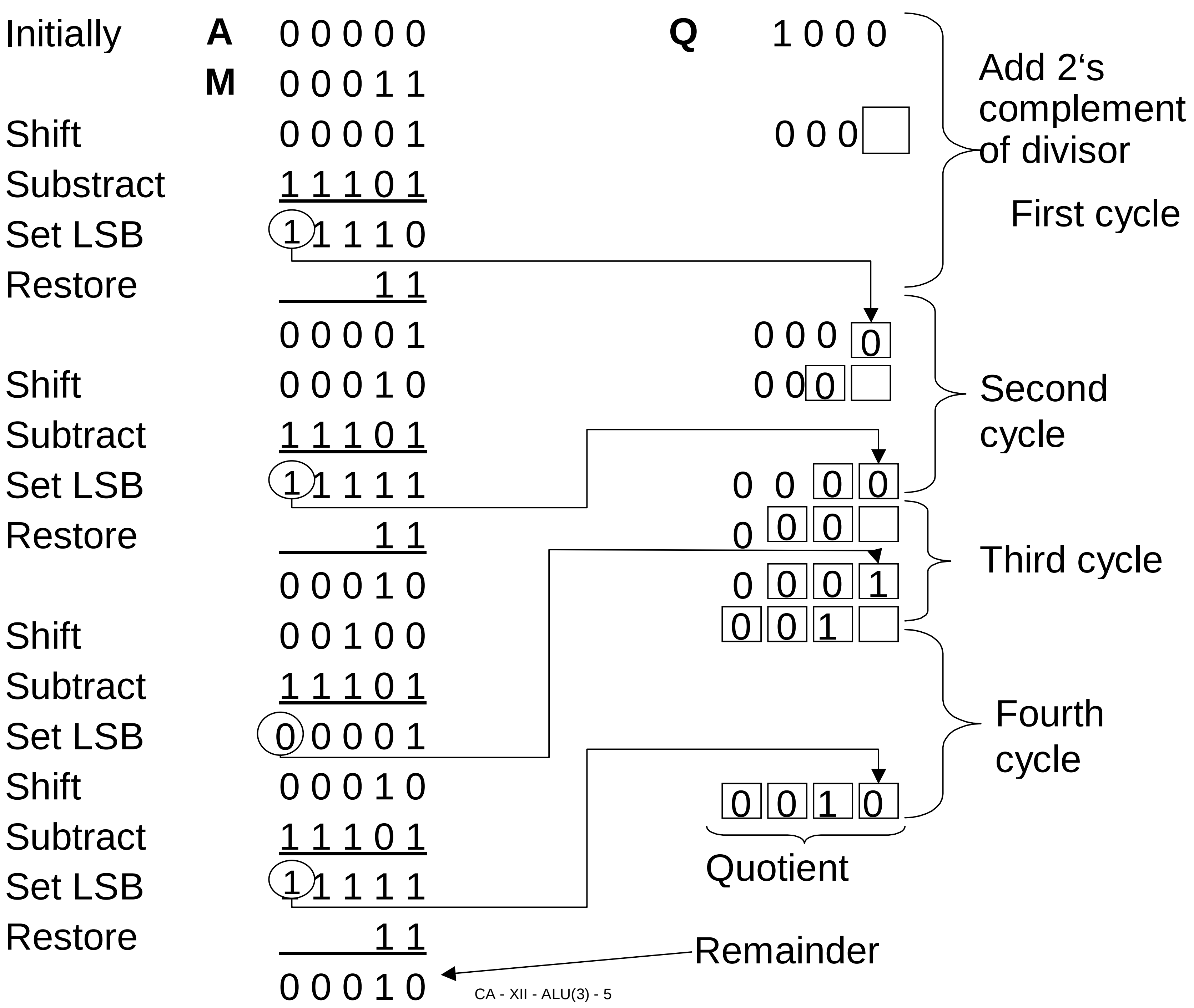
Q-Dividend/Quotient



Quotient in Q

Remainder in A

A RESTORING DIVISION EXAMPLE



Initially A 0 0 0 0 0 Q 1 0 0 0

M 0 0 0 1 1

Shift 0 0 0 0 1 0 0 0 □

Substract 1 1 1 0 1

Set LSB 1 1 1 1 0

Restore 1 1

0 0 0 0 1 0 0 0 0

Shift 0 0 0 1 0 0 0 0 □

Subtract 1 1 1 0 1

Set LSB 1 1 1 1 1 0 0 0 0

Restore 1 1 0 0 0 □

0 0 0 1 0 0 0 0 1

Shift 0 0 1 0 0 0 0 1 □

Subtract 1 1 1 0 1

Set LSB 0 0 0 0 1

Shift 0 0 0 1 0

Subtract 1 1 1 0 1

Set LSB 1 1 1 1 1

Restore 1 1

0 0 0 1 0

0 0 1 0

Add 2‘s complement of divisor

First cycle

Second cycle

Third cycle

Fourth cycle

Quotient

Remainder

NONRESTORING DIVISION

ALGORITHM FOR NONRESTORING DIVISION

- STEP 1: DO *n* TIMES
  - IF THE SIGN OF A IS 0, SHIFT A AND Q LEFT ONE BINARY POSITION AND SUBTRACT M FROM A;
  - OTHERWISE, SHIFT A AND Q LEFT AND ADD M TO A. IF THE SIGN OF A IS 0, SET Q_0 TO 1; OTHERWISE SET Q_0 TO 0.

- STEP 2: IF THE SIGN OF A IS 1, ADD M TO A
  - The negative result is restored by adding, i.e.,

    $\mathbf{R_i \leftarrow (R_i - M) + M} \qquad \mathbf{(1)}$

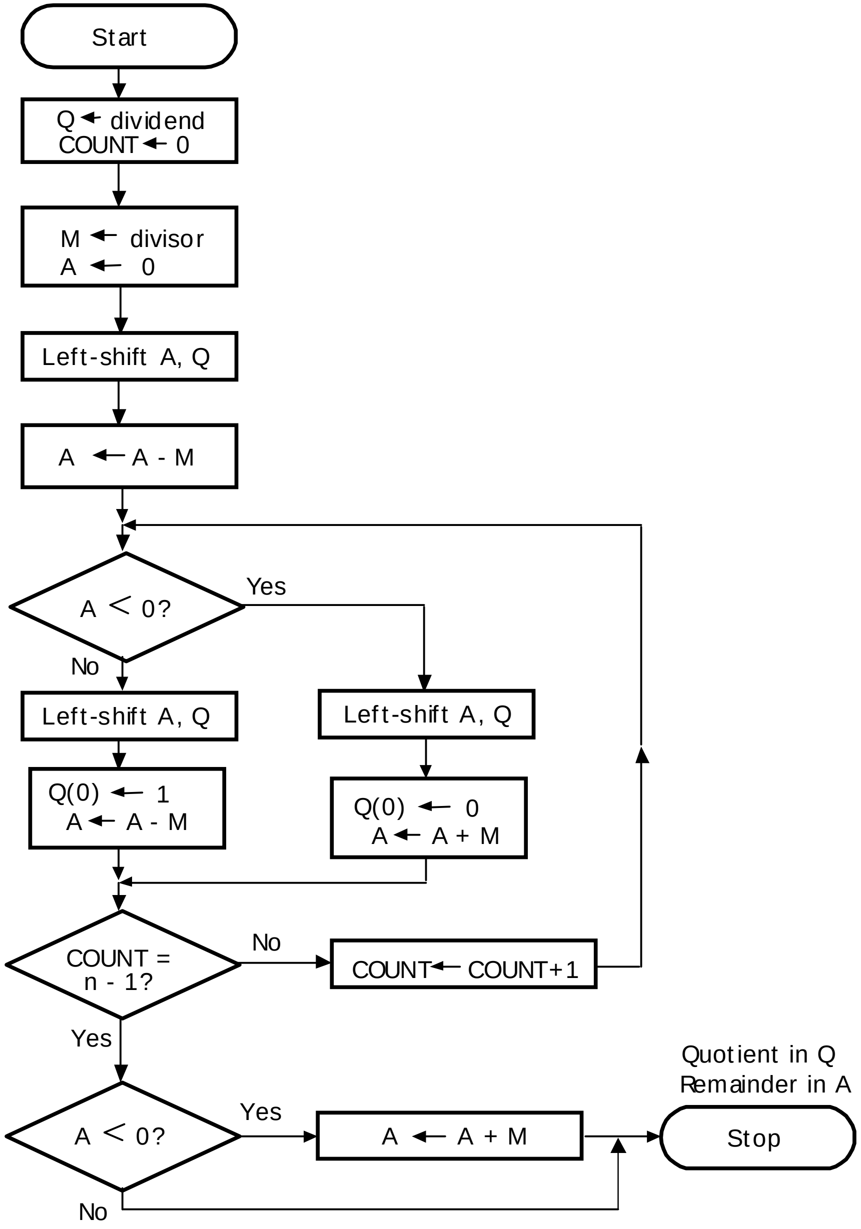
  - and is followed by a shift left one (i.e., multiplication by 2) and subtract:

    $\mathbf{R_{i+1} \leftarrow 2R_i - M} \qquad \mathbf{(2)}$

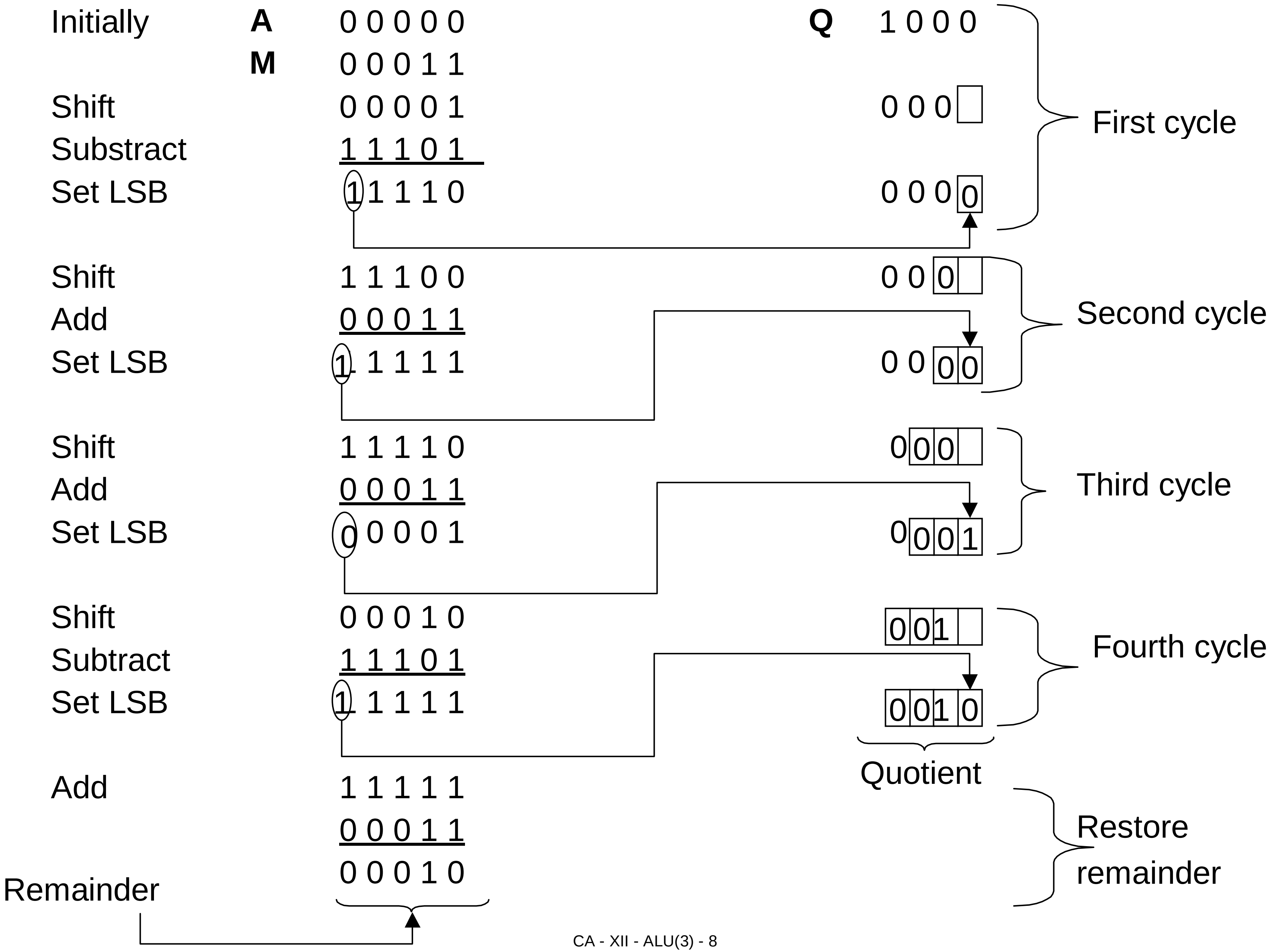
  - The two operations (1) and (2) are then merged into a single one:

$$\mathbf{R_{i+1} \leftarrow 2[(R_i - M) + M] - M = 2R_i - M}$$

A NONRESTORING DIVISION ALGORITHM FOR POSITIVE INTEGERS



A NONRESTORING DIVISION EXAMPLE



Initially	**A**	0 0 0 0 0	**Q**	1 0 0 0	First cycle
	M	0 0 0 1 1			
Shift		0 0 0 0 1		0 0 0 □	
Substract		1 1 1 0 1			
Set LSB		1 1 1 1 0		0 0 0 0	
Shift		1 1 1 0 0		0 0 0 □	Second cycle
Add		0 0 0 1 1			
Set LSB		1 1 1 1 1		0 0 0 0	
Shift		1 1 1 1 0		0 0 0 □	Third cycle
Add		0 0 0 1 1			
Set LSB		0 0 0 0 1		0 0 0 1	
Shift		0 0 0 1 0		0 0 1 □	Fourth cycle
Subtract		1 1 1 0 1			
Set LSB		1 1 1 1 1		0 0 1 0	
				Quotient	
Add		1 1 1 1 1			Restore remainder
		0 0 0 1 1			
Remainder		0 0 0 1 0			

DIVISION BY REPEATED MULTIPLICATION

$$Q = \frac{\text{DIVIDEND} \times F_0 \times F_1 \times \ldots}{\text{DIVISOR} \quad \times F_0 \times F_1 \times \ldots}$$

DIVISOR = V = 1-y

$F_0 = 1+y$ $\qquad$ $F_1 = 1+y^2$

$V \times F_0 = 1-y^2$

$V \times F_0 \times F_1 = (1-y^2)(1+y^2) = 1-y^4$

Multiply top & bottom by (1+y), then $(1+y^2)$, ...

$$\frac{1}{x} = \frac{1}{1-y} = \frac{1+y}{1-y^2} = \frac{(1+y)(1+y^2)}{1-y^4}$$

$$\frac{1}{x} = \frac{(1+y)(1+y^2)(1+y^4)....(1+y^{2^{i-1}})}{1-y^{2i}}$$

$\searrow$ 0

Key idea:

– Find a simple function (factor) so that by a repeated multiplication the value approaches 1

FLOATING-POINT ARITHMETIC (scientific notation)

COMPONENTS OF A FLOATING-POINT NUMBER REPRESENTATION

- Sign
- Exponent (X_e) (Y_e)
- Mantissa (X_m) (Y_m)

$X = X_m * B^{Xe}$

Examples: $+ 1.23 * 10^2$

$9.999\ 999 \times 10^{99}$

a. **Sign** is included as an extension of the mantissa.

b. **Exponent**

(1) **Scale Factor** (B) **base** or **radix**

(2) **Biasing**. Exponent values are usually biased about some excess value. For example if we have an exponent field capable of having values 00 to 99 to represent plus and minus values we would assign exponents excess 50. The exponent -50 would be represented by 0, exponent 0 by 50, and exponent 49 by 99.

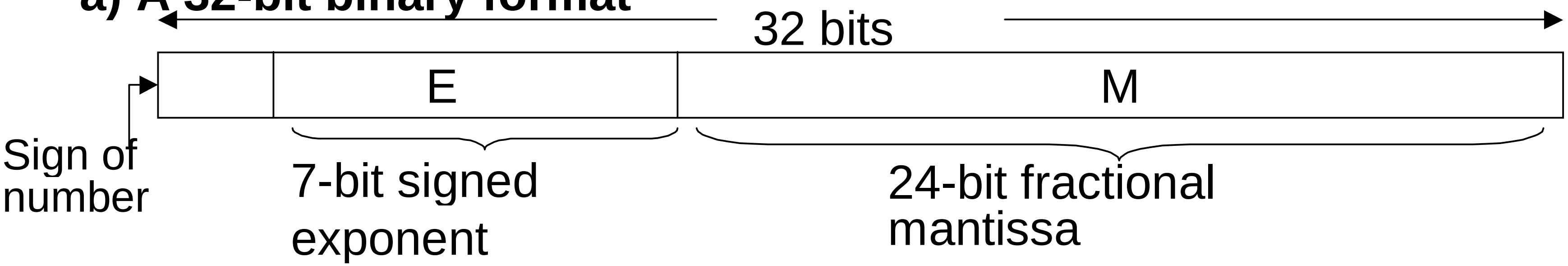
$0 \leq E_B \leq 99$ Exponent Field

$-50 \leq E \leq 49$ Exponent Represented

The reason for using this technique is that the magnitudes of numbers may be compared without regard to whether the number is in floating-point format or not, i.e., magnitudes are compared in the same way as for integer arithmetic.

FLOATING-POINT NUMBERS FORMATS

a) A 32-bit binary format



32 bits

E

M

Sign of number

7-bit signed exponent

24-bit fractional mantissa

b) Binary normalization example

0	0001001	001.......

$+0.001...\times 2^9$ Unnormalized value

0	0000111	1.......

$+0.1...\times 2^7$ Normalized version

Normalization keeps mantissa at the value of

$$\frac{1}{2} \le M < 1 \qquad -64 \le E \le 63$$

excess - 64 format $E' = E + 64$ $0 \le E' \le 127$

c) Hexadecimal (base 16) normalization example with excess 64-exponent (bias 64)

1	0000011	00000101.......

$-0.00000101...\times 16^3$ Unnormalized value

Exception: Underflow E<64 Overflow E>63 - set to 0

1	0000010	0101.......

$-0.0101...\times 16^2$ Normalized version

THE IEEE STANDARD FORMAT

$\text{Value}=(-1)^{S}x(1+M_1x2^{-1}+M_2x2^{-2}+...+M_{23}x2^{-23})x2^{E7...E1E0-127}$ (short real)

$\text{Value}=(-1)^{S}x(1+M_1x2^{-1}+M_2x2^{-2}+...+M_{52}x2^{-52})x2^{E10...E1E0-1023}$ (long real)

$\text{Value}=(-1)^{S}x(M_0+M_1x2^{-1}+M_2x2^{-2}+...+M_{63}x2^{-63})x2^{E14...E1E0-16383}$ (temporary real)

- **Short real (32 bits)**
  - Range of value $1,18x10^{-38} < |x| < 3,40x10^{+38}$, precision 24 bits

31	30	23	22	0
S	E_7.....	E_0	M_1 .... M_{23}	

  - S: sign bit (1=negative mantissa, 0=positive mantissa)
  - E7...E0: exponent (8bits, bias 127)
  - $M_1...M_{23}$: mantissa (23 bits plus implicit M_0=1)

- **Long real (64 bits)**
  - Range of value $2,23x10^{-308} < |x| < 1,79x10^{+308}$, precision 53 bits

63	62	52	51	0
S	E_{10}..........	E_0	M_1	M_{52}

  - bias 1023

- **Temporary real (80 bits)**
  - Range of value $3,37x10^{-4932} < |x|<1,79x10^{+4932}$, precision 53 bits

79	78	64	63	0
S	E_{14}.........	E_0	M_1	M_{63}

  - bias 16.383

FLOATING-POINT REPRESENTATION IEEE STANDARD

$(-1)^{S}$ (1.M) $2^{E\text{-bias}}$ (e.g., bias = 127)

S	8-bit exponent	23-bit mantissa

(single precision)

Mantissa with implied 1	24 bits $1 \leq M' < 2$
Largest Error	2^{-24}
Precision	≈ 7 decimal digits
Bias	127
Exponent Range	$-126 \leq E' \leq 127$
Smallest Number	$2^{-126} = 1.2 * 10^{-38}$
Largest Number	$(2 - 2^{-23})\ 2^{127} = 3.4 * 10^{38}$

Special Cases

a) Zero	$(-1)^{S} * 0$	E=0, M=0
b) Infinity	$(-1)^{S}\ \infty$	E=255, M=0
c) Not-a-number		E=255, M≠0
d) Normalized	$(-1)^{S} * (1.M)\ 2^{E-127}$	0 < E < 255
e) Unnormalized	$(-1)^{S} * (0.M)\ 2^{-126}$	E=1, M≠0

(This Number cannot be normalized because a shift to the left would cause an underflow.)

Examples

S	Exponent	Mantissa implied 1			
0	10000001	1.00...0	= 1.0	$* 2^{129-127}$	= 4.0
0	01111111	1.10...0	= 1.5	$* 2^{127-127}$	= 1.5
0	10000000	1.010...0	= 1.25	$* 2^{128-127}$	= 2.5

UNDERFLOW AND OVERFLOW

- Underflow occurs when a resulting biased exponent is less than zero (0). In such case the floating point word is set to all zeros.
- Overflow occurs when a resulting biased exponent is greater than the maximum value allowed for the exponent field.
- In both cases an error bit in a status word usually is set or an interrupt enabled so that a programm can determine what action to take following an underflow or overflow.

NORMALIZATION AND SCALING

- Normalization is a process assuring the maximum available accuracy of a given floating-point number
- Floating-point arithmetic units are usually designed to handle floating-point numbers in a normalized form. If the numbers are not presented to the unit in normalized form the unit often will not yield the correct result. The normalization of a number is related to the base or radius of the exponent. The normalization of a number must occur in shifts of binary digits in units represented by the base. In normalization a digit must appear in the left most mantissa "radix" position adjacent to the decimal. Each left shift of the mantissa represents a subtraction of 1 from the exponent each right shift an addition of 1 to the exponent.
- Normalization and changing the scale factor works like the pen and pencil method in scientific notation.

Examples :

$1.27 \times 10^{5} = .127 \times 10^{6}$

$0.03 \times 10^{2} = .30 \times 10^{1}$

$42.1 \times 10^{-6} = .421 \times 10^{-4}$

$0.022 \times 10^{-4} = .220 \times 10^{-5}$

- For binary shift a single shift by a sinlge position is required while for a hexadecimal shift a shift by four positions nescessary (implied radix 16)

DIFFICULTY AND ROUNDING (OFF)

Arithmetic's Difficulty

a. Multiplication and division are less complicated than addition or subtraction to implement.

b. The addition or subtraction of numbers is complicated by the fact that the smaller number has to be shifted to be decimally aligned with the larger number before the arithmetic operation is performed.

Rounding

Fractional binary arithmetic because of limited bit representation can result in erroneous results. Guard bits or extended precision (larger registers) bits are often used to increase accuracy. The result after an operation is then rounded.

Rounding Methods	Results	Guard Bits	Comments
Chopping (Truncation)	.0010	110	Biased Error:
The result is not rounded.	.0010]		0 - Chopped Bits
Normal Rounding	.0010	110	Unbiased Error:
A one is added to the results	.0011]		-½ to +½ (LSB)
LSB if the result is followed	.0011	110	
by a 1 guard bit.	.0100]		
Von Neumann	.0010	110	More Complex Imp.
A one bit is set in the	.0011]		Unbiased Error:
results LSB if the result is	.0011	110	-1 to +1 (LSB)
followed by a guard bit.	.0011]		

EQUATIONS FOR FLOATING-POINT OPERATIONS

X_E, Y_E - exponents of X and Y

X_M, Y_M - mantissas of X and Y

ADD $X + Y = (X_M * 2^{X_E-Y_E} + Y_M) * 2^{Y_E}$

SUB $X - Y = (X_M * 2^{X_E-Y_E} - Y_M) * 2^{Y_E}$

MULT $X * Y = (X_M * Y_M) * 2^{X_E +Y_E}$

DIV $X / Y = (X_M * Y_M) * 2^{X_E-Y_E}$

FLOATING-POINT ARITHMETIC ALGORITHMS

ADD/SUB

1) CHOOSE THE NUMBER WITH SMALLER EXPONENT AND SHIFT IT'S MANTISSA RIGHT A NUMBER OF STEPS EQUAL TO THE DIFFERENCE IN EXPONENTS.

2) SET THE EXPONENT OF THE RESULT EQUAL TO THE LARGE EXPONENT.

3) PERFORM ADDITION-SUBTRACTION ON THE MANTISSAS AND DETERMINE THE SIGN OF THE RESULT.

4) NORMALIZE THE RESULT IF NECESSARY.

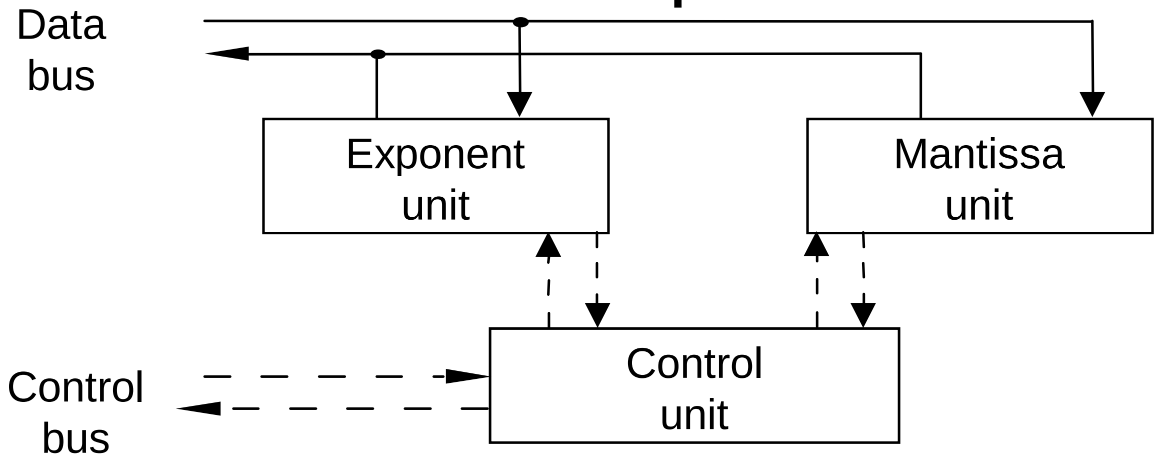
5) CHECK FOR OVERFLOW AND UNDERFLOW.

FLOATING-POINT MULTIPLICATION AND DIVISION

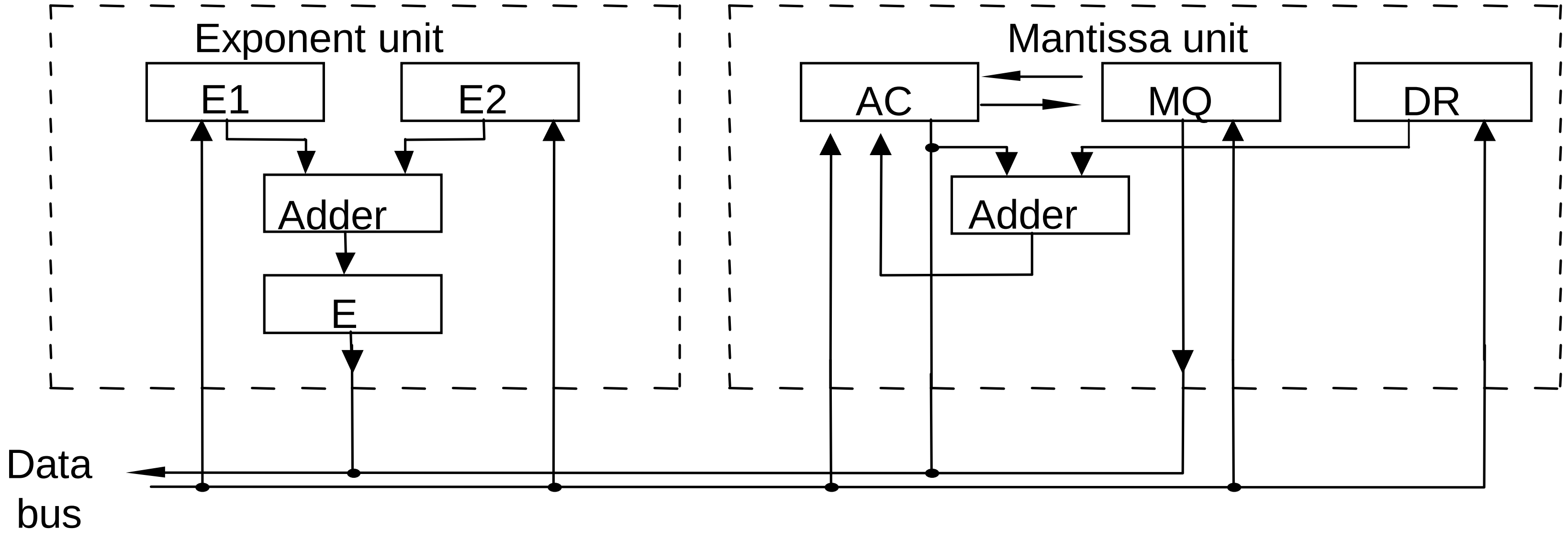
MULT	1) ADD EXPONENTS.
	2) MULTIPLY MANTISSAS AND DETERMINE SIGN OF RESULT
	3) NORMALIZE THE RESULTING VALUE IF NECESSARY
	4) CHECK FOR OVERFLOW AND UNDERFLOW
DIV	1) SUBTRACT EXPONENTS
	2) DIVIDE MANTISSAS & DETERMINE SIGN OF RESULT
	3) NORMALIZE THE RESULTING VALUE IF NECESSARY
	4) CHECK FOR OVERFLOW AND UNDERFLOW

OVERFLOW	EXP > + RANGE
UNDERFLOW	EXP < - RANGE

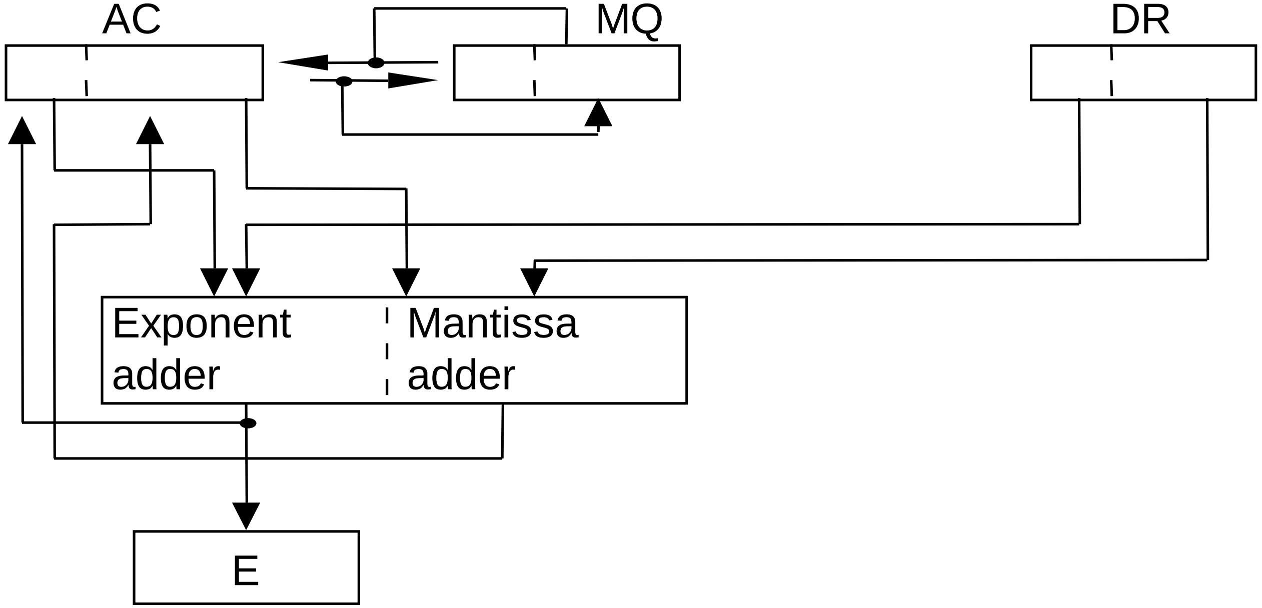
A floating-point arithmetic unit viewed as two fixed-point arithmetic units



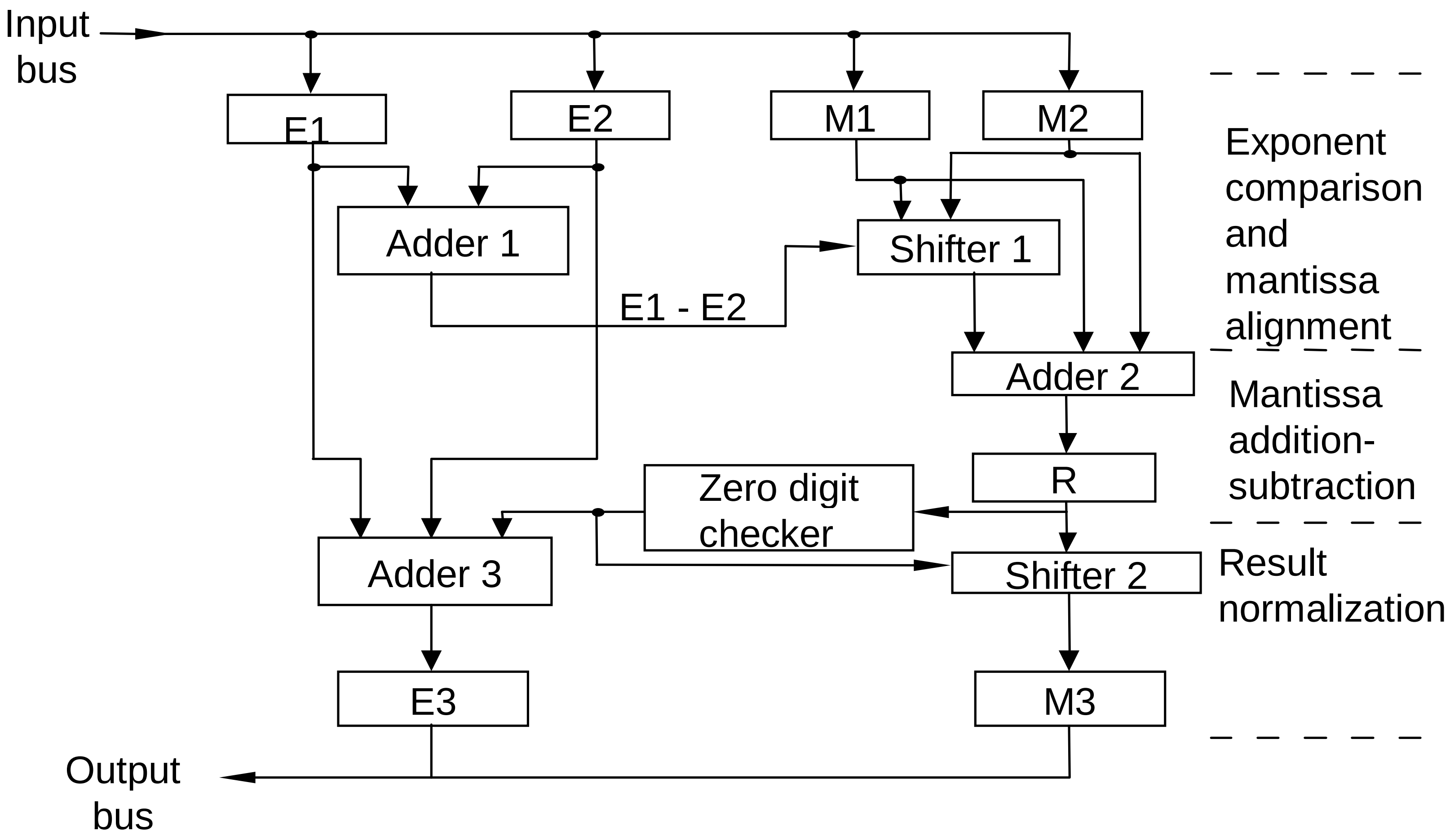
Data-processing part of a simple floating-point arithmetic unit



A combined fixed-point and floating-point arithmetic unit

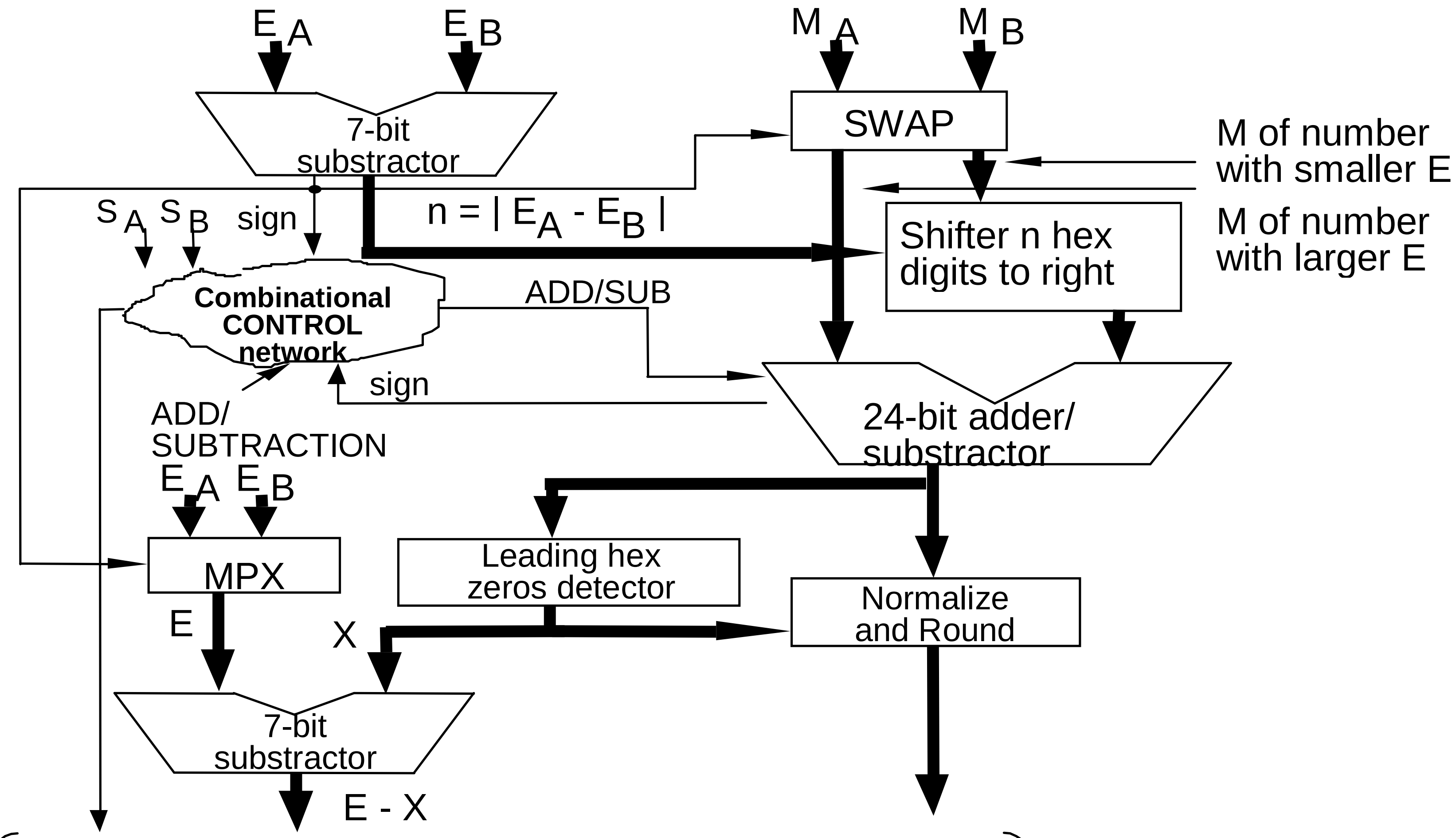


Floating-point adder of the IBM mainframe



Floating-point addition-subtraction unit

32-bit operands { A: S_A, E_A, M_A
B: S_B, E_B, M_B }



E_A

E_B

M_A

M_B

7-bit substractor

SWAP

M of number with smaller E

M of number with larger E

S_A S_B sign

n = | E_A - E_B |

Shifter n hex digits to right

Combinational CONTROL network

ADD/SUB

sign

ADD/ SUBTRACTION

24-bit adder/ substractor

E_A E_B

MPX

Leading hex zeros detector

Normalize and Round

E

X

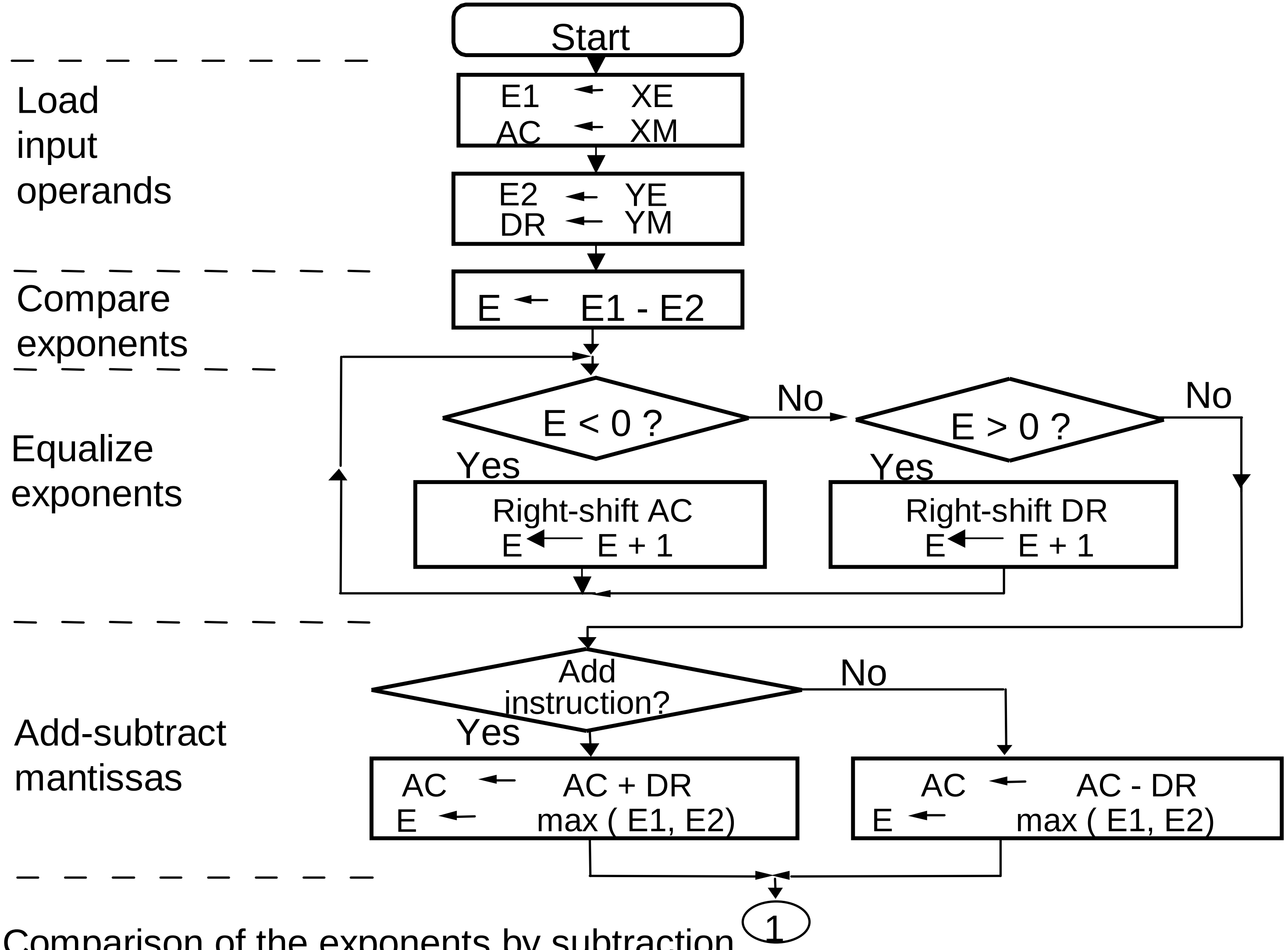
7-bit substractor

E - X

{ R: S_R , E_R , M_R } 32-bit result

R = A ± B

FLOATING-POINT ADDITION-SUBTRACTION ALGORITHM



Load input operands

Compare exponents

Equalize exponents

Add-subtract mantissas

Start

E1 ← XE
AC ← XM

E2 ← YE
DR ← YM

E ← E1 - E2

E < 0 ?

No

Yes

Right-shift AC
E ← E + 1

E > 0 ?

No

Yes

Right-shift DR
E ← E + 1

Add instruction?

No

Yes

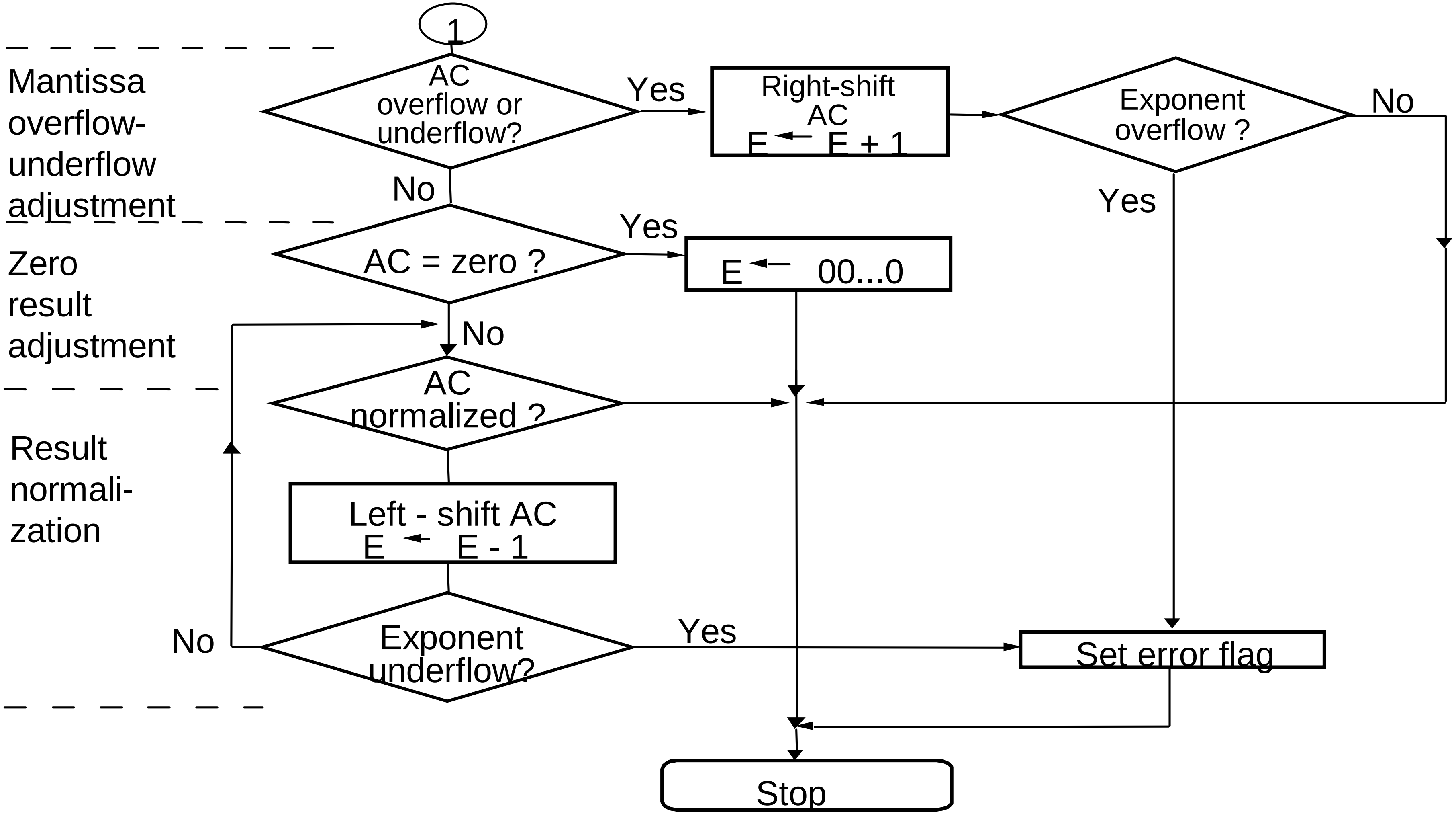
AC ← AC + DR
E ← max (E1, E2)

AC ← AC - DR
E ← max (E1, E2)

1

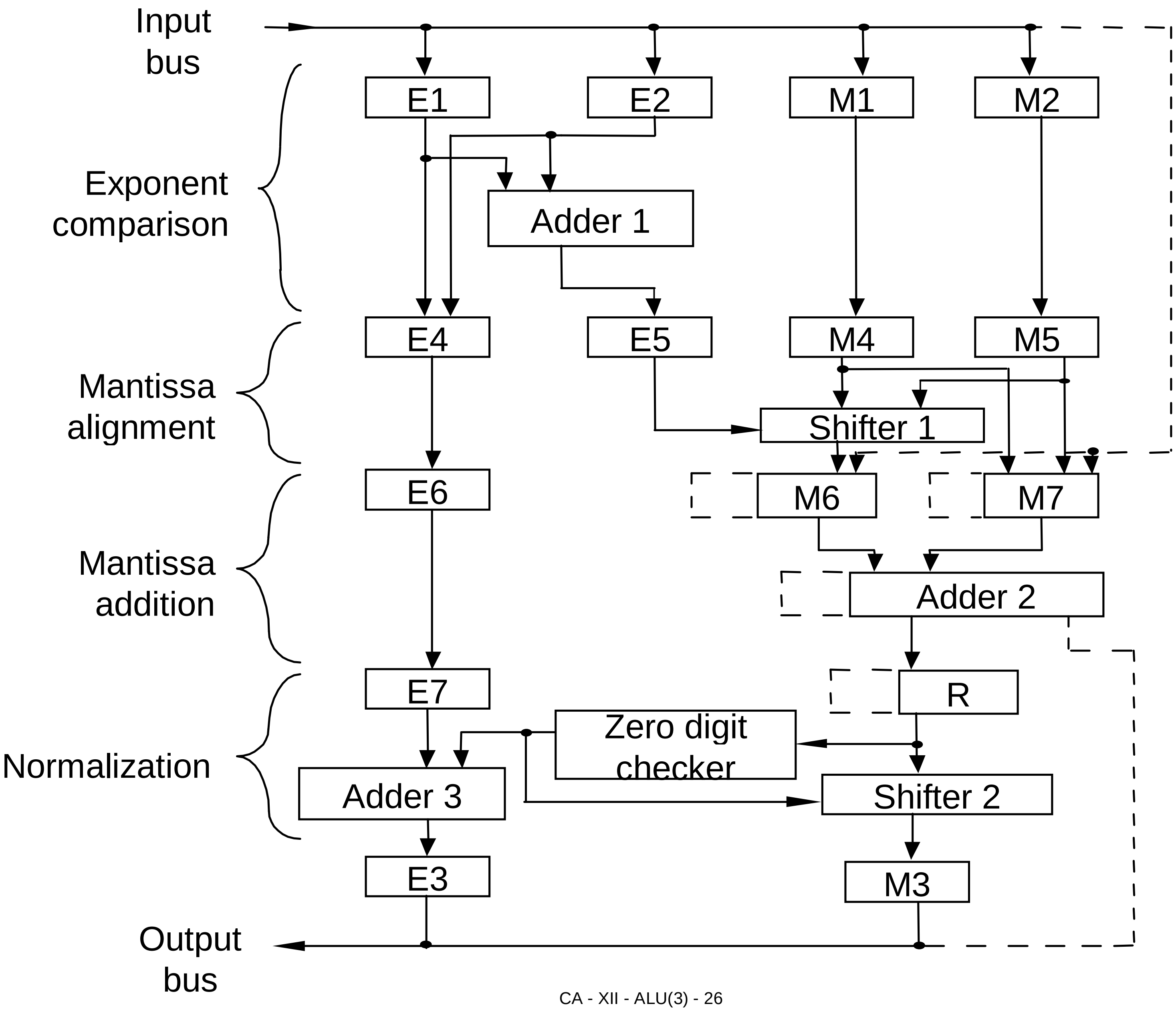
1. Comparison of the exponents by subtraction
2. Alignment of the mantissas by shifting
3. Addition or subtraction of the mantissas
4. Normalization of the result

Tests are performed for overflow and for a zero mantissa in the

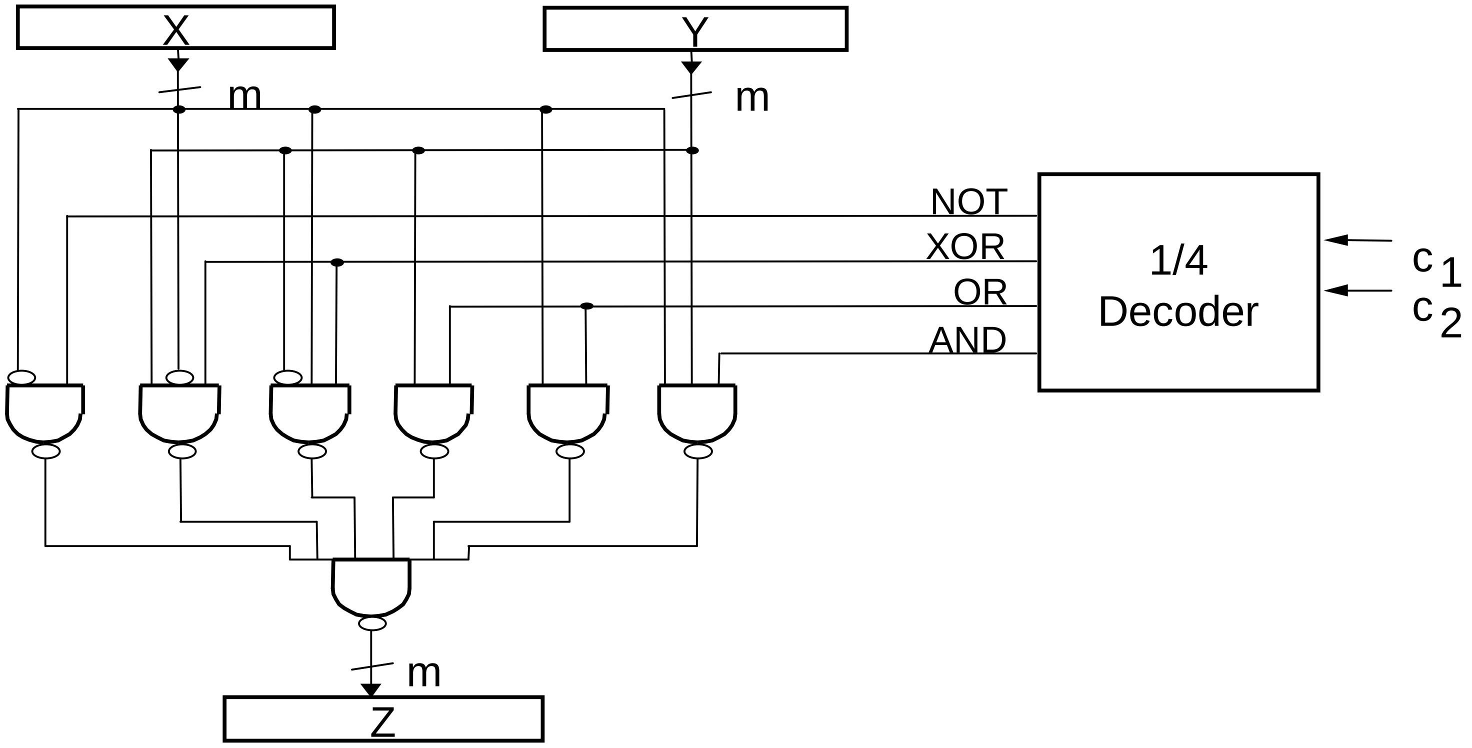


1
Mantissa overflow-underflow adjustment
AC overflow or underflow?
Yes
Right-shift AC
E ← E + 1
Exponent overflow ?
No
Yes
No
Zero result adjustment
AC = zero ?
Yes
E ← 00...0
No
AC normalized ?
Result normali-zation
Left - shift AC
E ← E - 1
No
Exponent underflow?
Yes
Set error flag
Stop

PIPELINED VERSION OF THE FLOATING-POINT ADDER

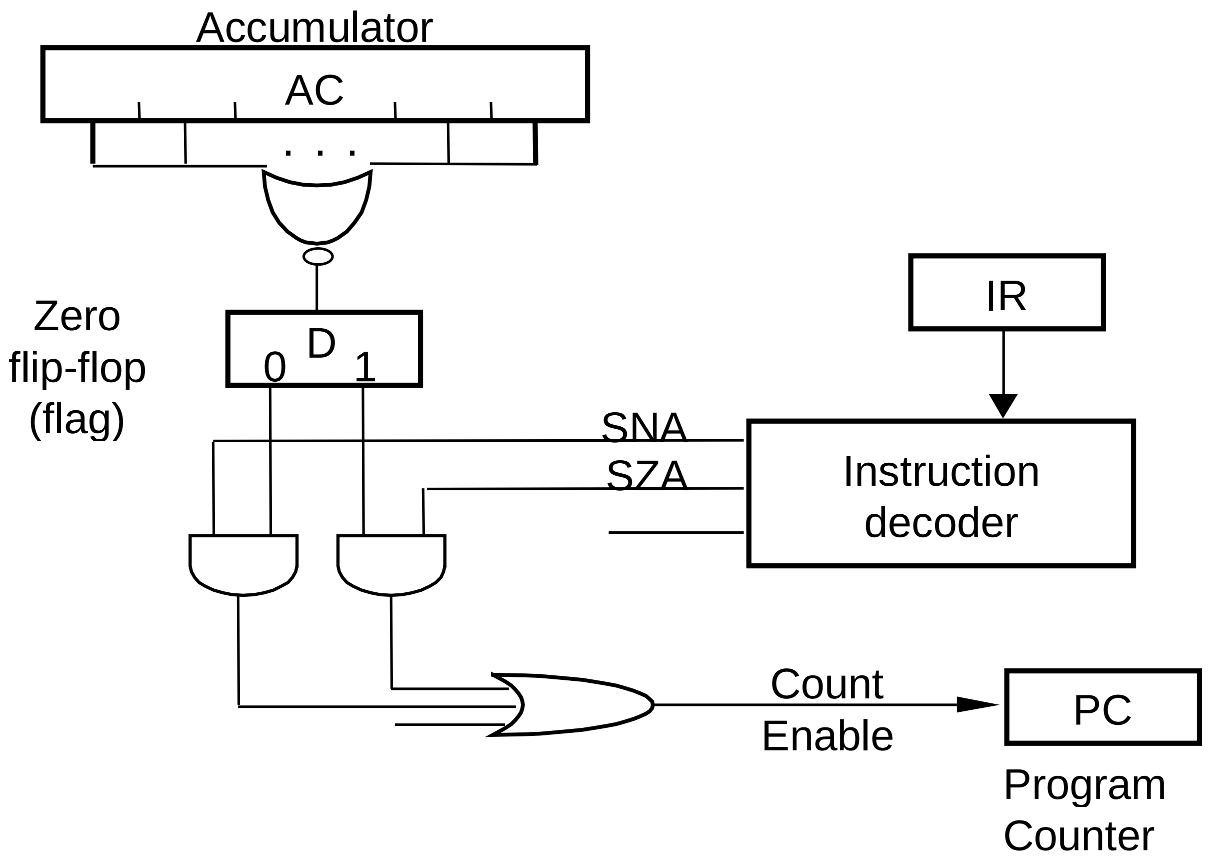


Implementation of four logical instructions
AND, OR, EXCLUSIVE-OR and NOT



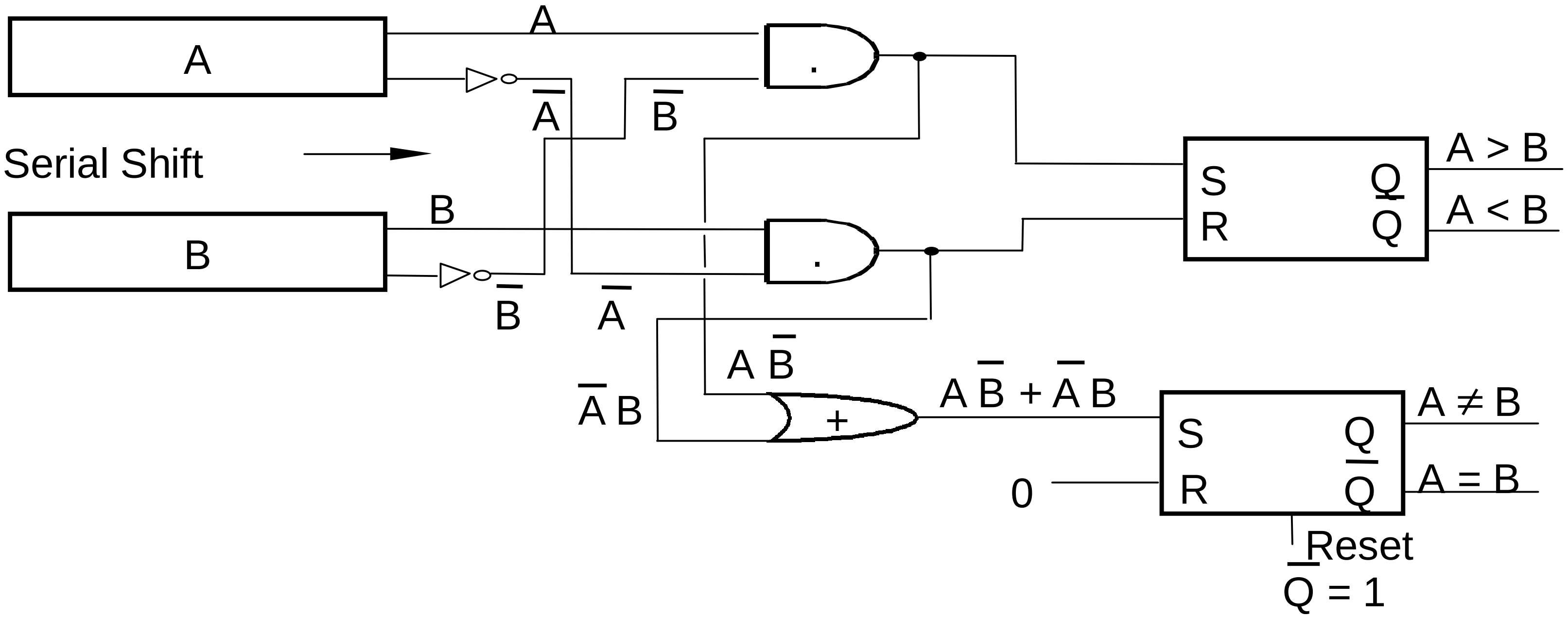
$$Z = \underbrace{\bar{c}_1 \bar{c}_2 X Y}_{\text{AND}} + \underbrace{\bar{c}_1 c_2 X + \bar{c}_1 c_2 Y}_{\text{OR}} + \underbrace{c_1 \bar{c}_2 \bar{X} Y + c_1 c_2 X \bar{Y}}_{\text{EX - OR}} + \underbrace{c_1 c_2 \bar{X}}_{\text{NOT}}$$

Implementation of two conditional branch instructions SZA (skip on zero accumulator) and SNA (skip on nonzero accumulator)



TEST FOR A>B (or A<B)

AND A=B (or nonequality A≠B)


A
A
$\bar{A}$
$\bar{B}$
Serial Shift
B
B
$\bar{B}$
$\bar{A}$
.
.
S
R
Q
$\bar{Q}$
A > B
A < B
$A\bar{B}$
$\bar{A}B$
+
$A\bar{B} + \bar{A}B$
S
R
Q
$\bar{Q}$
A ≠ B
A = B
0
Reset
$\bar{Q}$ = 1