



**BME**



**KJIT**

*Budapest University of Technology and Economics*

*Faculty of Transportation Engineering and Vehicle Engineering*

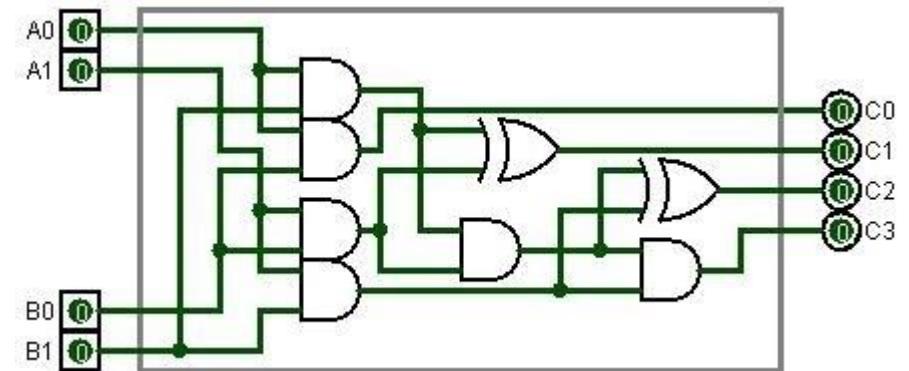
*Department of Control for Transportation and Vehicle Systems*

# ARITHMETIC OPERATIONS, PART 3.

Lecture 5.

# Binary Multiplication

- **Multiplication is usually (generally) realized by repeated addition!**
- Methods:
  - Multiplication by leftward shifting of the multiplicand,
  - Multiplication by rightward shifting of the partial sum,
  - Multiplication by using „ones row”,
  - Multiplication by grouping of digits.



other method: 2\*2 bit multiplier, source:

[http://survivalcraftgame.wikia.com/wiki/Binary\\_Calculator\\_-\\_Multiplication](http://survivalcraftgame.wikia.com/wiki/Binary_Calculator_-_Multiplication)



# Binary Multiplication

- Multiplication by rightforward shifting of the partial sum:

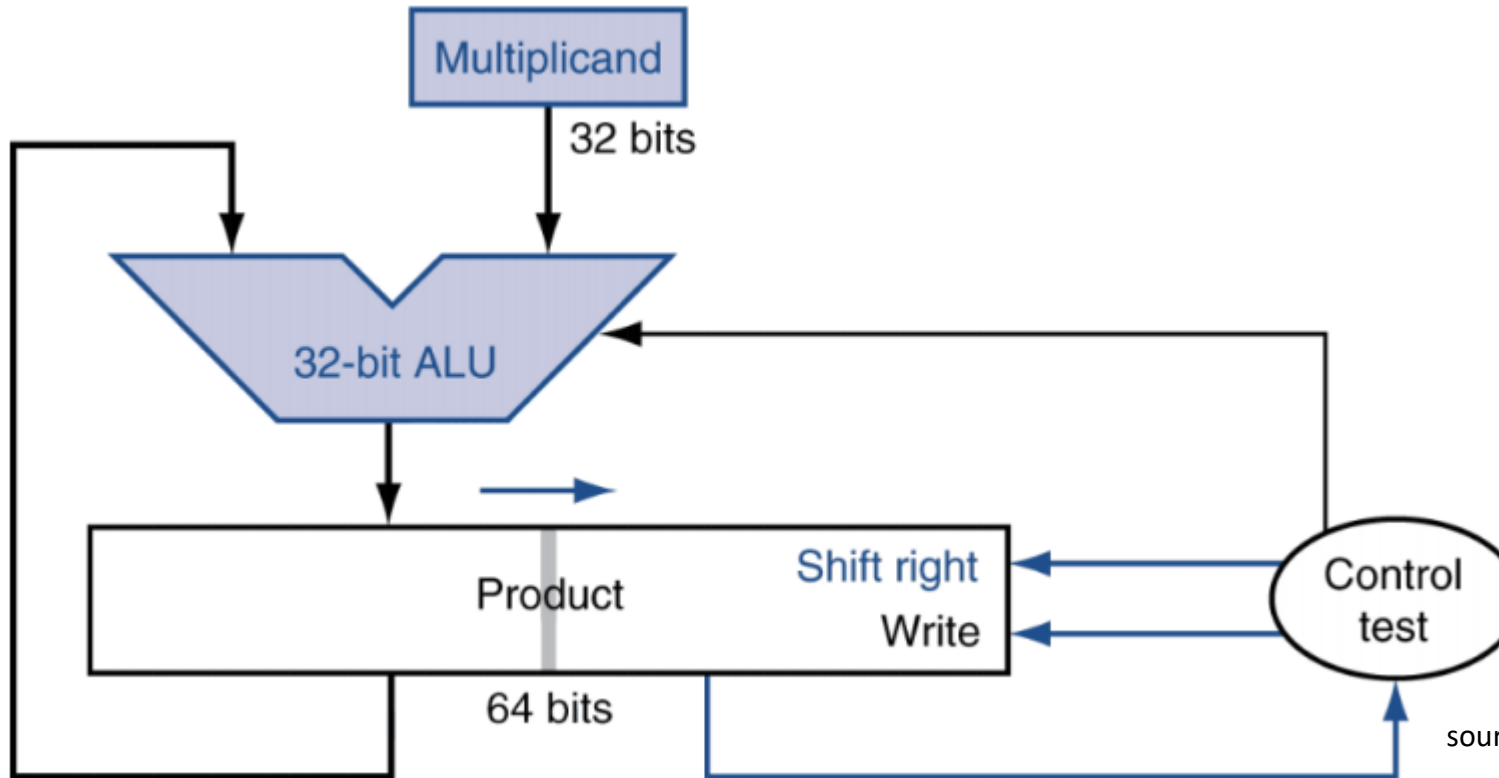
E.g.  $27 \times 11 = 297_{10}$

	11011	x	
			$\overset{A_3}{1} \overset{A_2}{0} \overset{A_1}{1} \overset{A_0}{1}$
A <sub>0</sub>	00000		
+	11011	x	←
A <sub>1</sub>	11011		
A <sub>1</sub> '	↳ 11011		rightward shifting of the partial sum by 1 digit
+	11011	x	←
A <sub>2</sub>	1010001		
A <sub>2</sub> '	↳ 1010001		rightward shifting of the partial sum by 1 digit
+	00000	x	←
A <sub>3</sub>	01010001		
A <sub>3</sub> '	↳ 01010001		rightward shifting of the partial sum by 1 digit
+	11011	x	←
A <sub>4</sub>	100101001		→ the same result :)

↑ partial sum

# Binary Multiplication

- Multiplication by rightward shifting of the partial sum:



source: <https://i.stack.imgur.com/NOrIk.png>

# Binary Multiplication

- Multiplication by using „ones row”:
  - ones row can be written as the followings:

$$2^{n-1} \quad 2^{n-2} \quad \dots \quad 2^1 \quad 2^0$$
$$1 \quad 1 \quad \dots \quad 1 \quad 1 = 2^n - 1$$

- e.g:
  - $5 \cdot 62 = 310_{10}$
  - $310 = 5 \cdot 2^6 - 2 \cdot 5 = 320 - 10$



# Binary Multiplication

- Multiplication by using „grouping of digits:
  - Idea: we divide the multiplier to group of digits:
    - 00 → 0
    - 01 → 1
    - 10 → 2
    - 11 → 3
  - the partial sums will be 0...3 times of the multiplicand

E.g. - multiplicand: 6  
- multiplier: 99

$$6 \times 99 = 594_{10}$$

$0 \times 6 = 0$	$=$	$00000$
$1 \times 6 = 6$	$=$	$00110$
$2 \times 6 = 12$	$=$	$01100$
$3 \times 6 = 18$	$=$	$10010$

$$99_{10} = 1100011_2$$



# Binary Multiplication

- Multiplication by using „grouping of digits:

Handwritten binary multiplication diagram showing the grouping of digits in the multiplier and the resulting partial products.

Multiplier:  $110 \times 0110011$

Multiplier groups (circled in red):  $01$ ,  $10$ ,  $00$ ,  $11$

Annotations: "leftward shifting of the multiplicand groups" (with arrow), "groups"

Partial products (A<sub>0</sub> to A<sub>4</sub>):

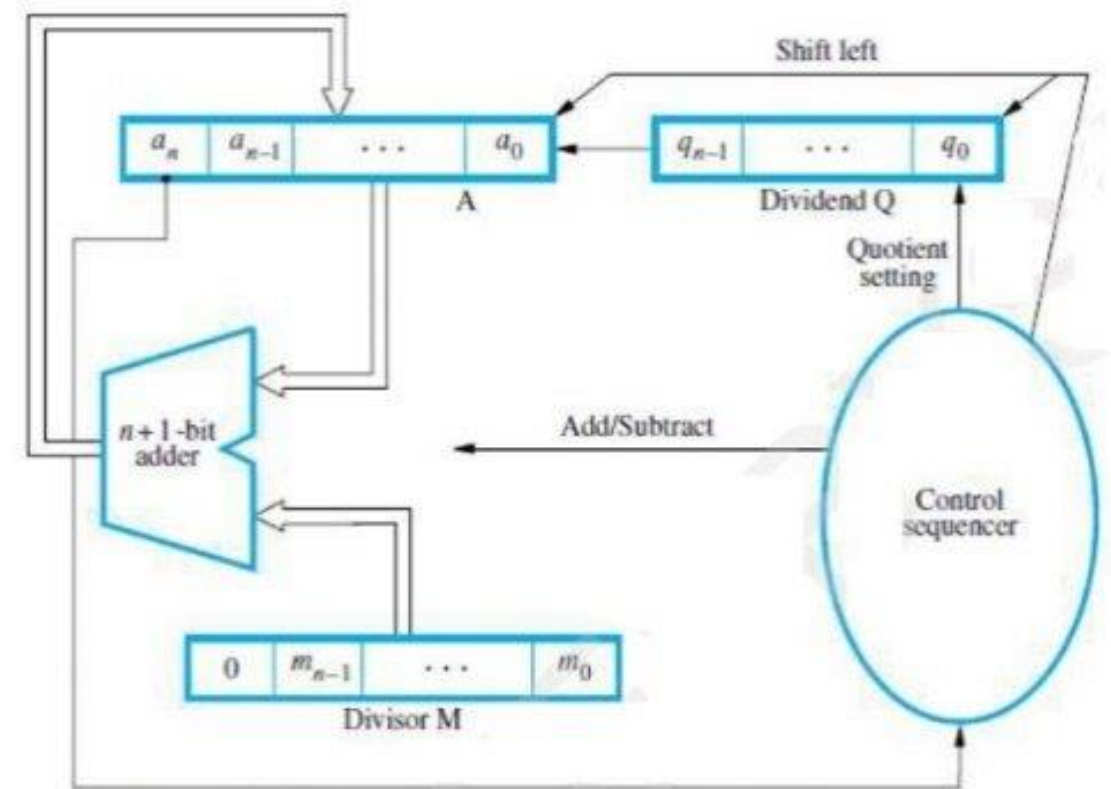
- A<sub>0</sub>:  $00000$
- A<sub>1</sub>:  $10010$  (from  $01 \times 110$ , labeled  $3 \times 6$ )
- A<sub>2</sub>:  $00000$  (from  $10 \times 110$ , labeled  $0 \times 6$ , "shifting 2 digits to left!")
- A<sub>3</sub>:  $0010010$  (from  $00 \times 110$ , labeled  $2 \times 6$ , "shifting 2 digits to left!")
- A<sub>4</sub>:  $11010010$  (from  $11 \times 110$ , labeled  $1 \times 6$ , "shifting 2 digits to left!")

Final result:  $1001010010_2 = 2^1 + 2^4 + 2^6 + 2^9 = 2 + 16 + 64 + 512 = 594$

Vertical label on the left: "partial sum" with arrows pointing to A<sub>1</sub>, A<sub>2</sub>, A<sub>3</sub>, and A<sub>4</sub>.

# Binary Division

- Methods:
  - Division by repeated subtraction,
  - Restoring division,
  - Division in complement code.



source:

[https://www.researchgate.net/publication/326669218\\_Reversible\\_Signed\\_Division\\_for\\_Computing\\_Systems/figures?lo=1](https://www.researchgate.net/publication/326669218_Reversible_Signed_Division_for_Computing_Systems/figures?lo=1)

# Binary Division

- Division by repeated subtraction:

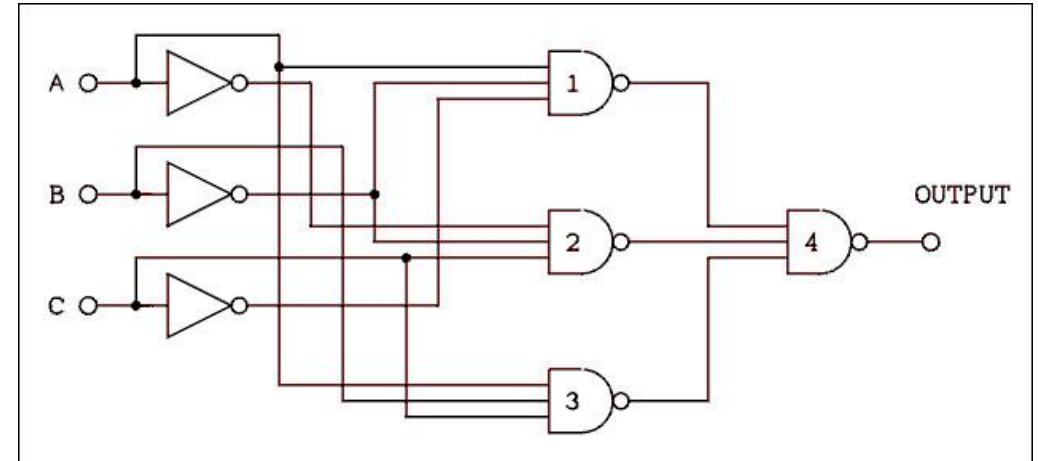
- Algorithm:

- $A = \text{dividend}$ ,  $B = \text{divisor}$ ,  $A_i = \text{partial sum}$

- $A_i = 2 * A_{i-1} - q_i * B$

- where:  $q_i = \begin{cases} 1, & \text{if } 2 * A_{i-1} \geq B \\ 0, & \text{if } 2 * A_{i-1} < B \end{cases}$

- $\frac{A}{B} = Q + \frac{r_n}{B}$ , where  $Q = \text{quotient}$ ,  $r_n = \text{remainder after the step } n$ .



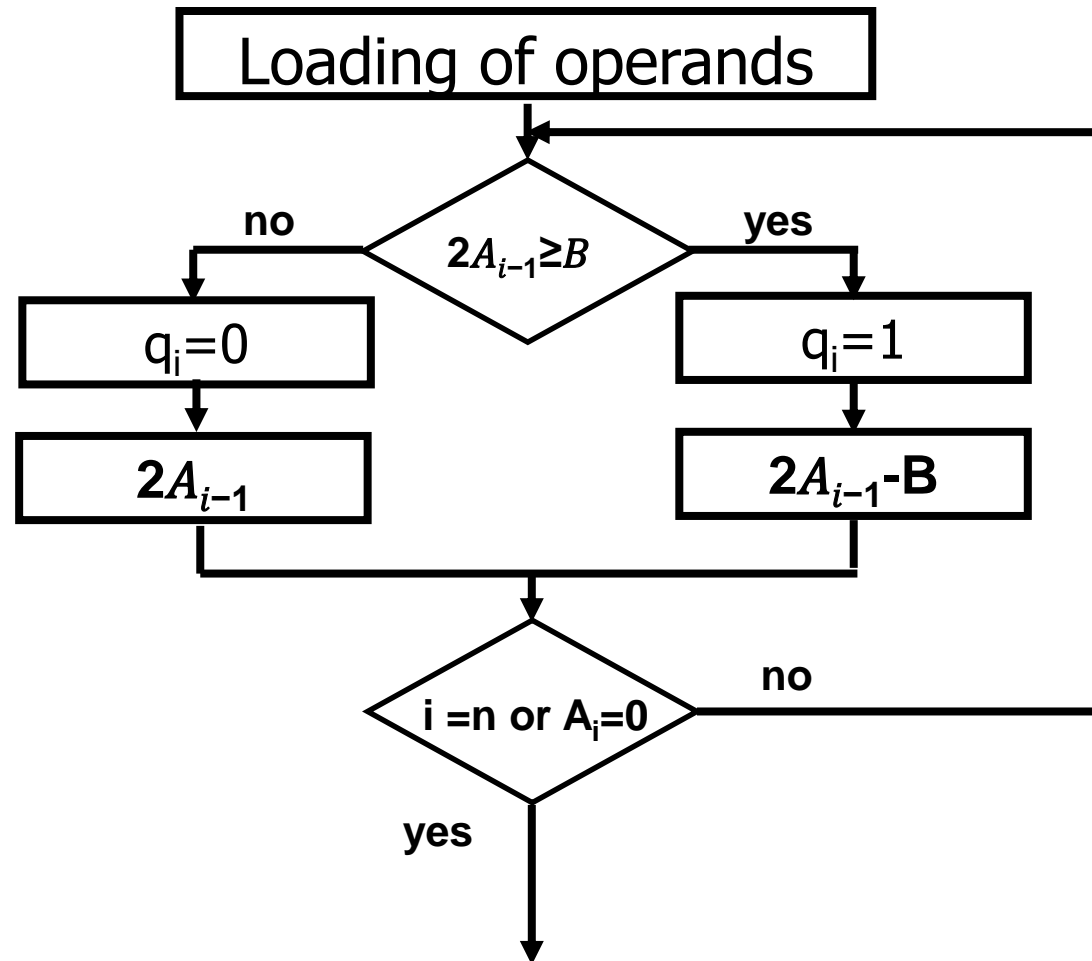
source: <http://golfinamigos.com/binary-division-circuit-diagram/>

# Binary Division

- $A_i = 2 * A_{i-1} - q_i * B$
- $A_1 = 2 * A_0 - q_0 * B$
- $A_2 = 2 * A_1 - q_1 * B = 2^2 * A_0 - 2 * q_1 * B - q_2 * B$
- $A_3 = 2 * A_2 - q_3 * B = 2^3 * A_0 - 2^2 * q_1 * B - 2 * q_2 * B - q_3 * B$
- ...
- $A_n = 2^n * A_0 - B * (2^{n-1} * q_1 + 2^{n-2} * q_2 + \dots + q_n)$
- $* \frac{2^{-n}}{B}$
- $\frac{A * 2^{-n}}{B} = \frac{A_0}{B} - B * (2^{-1} * q_1 + 2^{-2} * q_2 + \dots + 2^{-n} * q_n)$ , where  $A_0 = A$ , and (...) = *local values of the quotient*
- $\frac{A}{B} = Q + \frac{A * 2^{-n}}{B}$
- $\frac{A}{B} = Q + \frac{r_n}{B}$ , where  $Q$  = quotient,  $r_n$  = *remainder after the step n.*

# Binary Division

- Algorithm of the division with repeated subtraction in the ALU



# Binary Division

- Example: 5/8

$$\begin{array}{r} \boxed{A} \quad \boxed{B} \\ 101 : 1000 = \end{array}$$

$$\begin{array}{r} A_0 \\ \underline{- 0000} \\ A_0 \leftarrow 101 \\ 2 \times A_0 \quad 1010 \\ \underline{- 1000} \\ A_1 = 2 \times A_0 - B \leftarrow 0010 \\ A_2 = 2 \times A_1 \quad 100 \\ \underline{- 0000} \\ 2 \times A_2 \quad 1000 \\ \underline{- 1000} \\ A_3 = 2 \times A_2 + B \quad 0000 \end{array}$$

1. step  
 2. step  
 3. step

min. end is greater than the subtrahend?  $\rightarrow$  no  $\rightarrow$  zero  
 $\uparrow$  no  $\uparrow$  yes  $\uparrow$  no  $\uparrow$  yes  
 it was zero, than shifting leftward by 1 digit

# Binary Division

- Division by restoring division:

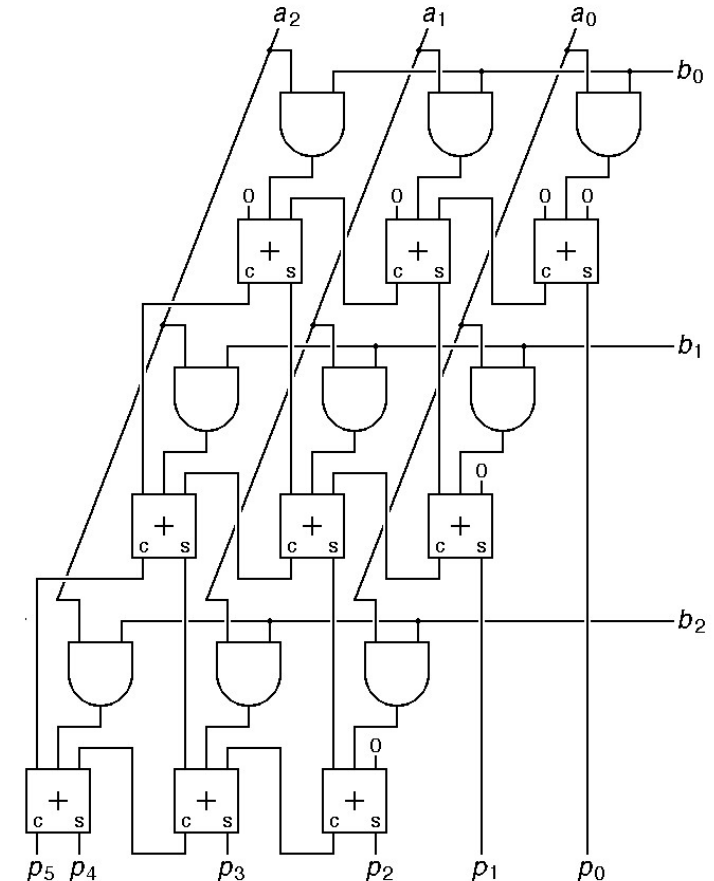
- Algorithm:
- we make the subtraction in every step, and if the difference is negative, we have to add the divisor to the partial sum, than it has to shift leftward the partial sum
- if the difference is positive, the restoring is cancelled
- the subtraction means an addition is 2's complement

- $A = \text{dividend}$ ,  $B = \text{divisor}$ ,  $A_i = \text{partial sum}$

- $B_k = 2^k \text{ complement code of } B$

- $A + B_k = A + 2^k - |B| = (A - B) + 2^k$

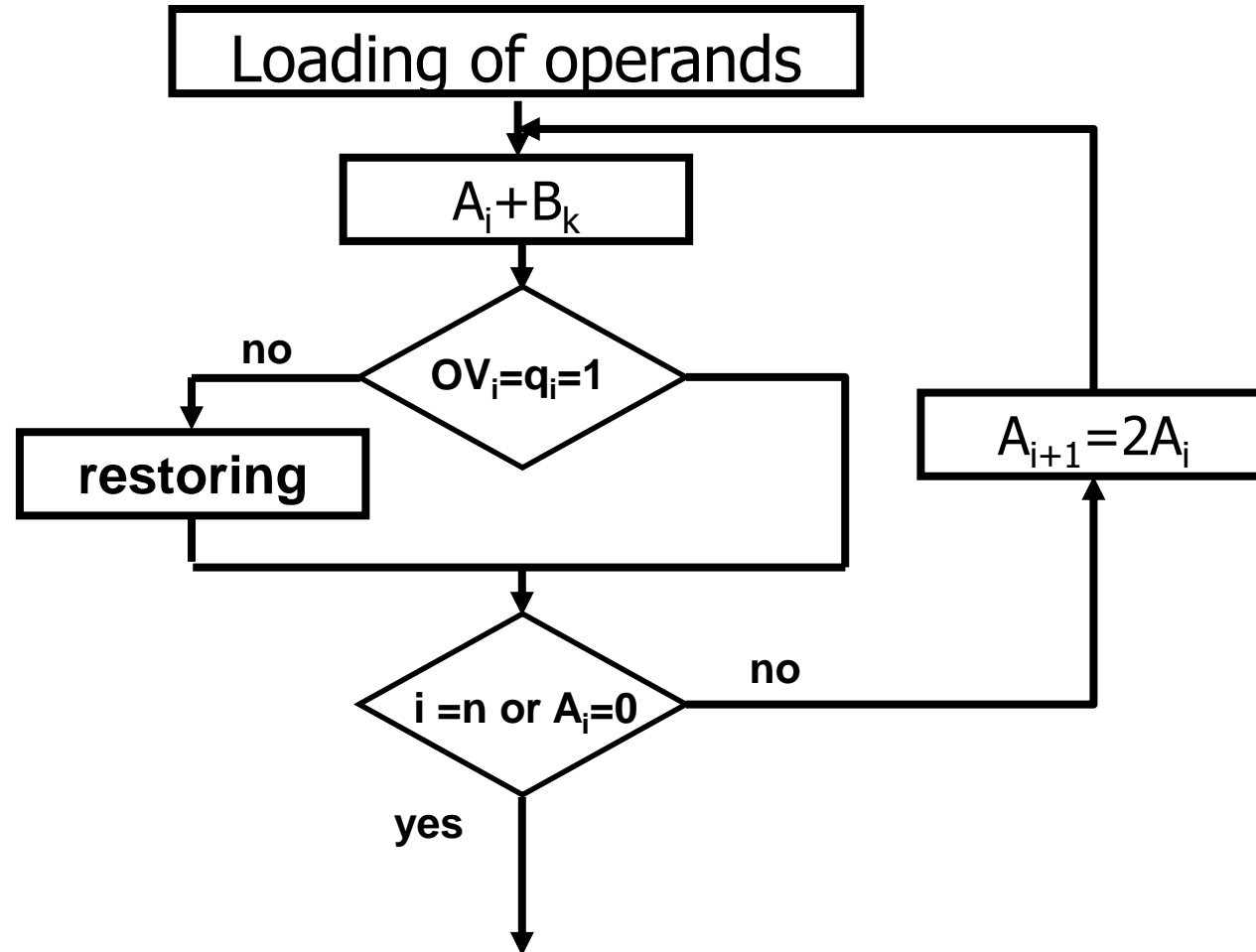
- $A_i = 2 * A_{i-1} - q_i * B, \text{ where } q_i = \text{overflow}$



source: <http://golfnamigos.com/binary-division-circuit-diagram/>

# Binary Division

- Algorithm of the restoring division in the ALU





# Binary Division

- Example: 5/8

OV  $\frac{0}{0+}$  | Signed  $\frac{0}{0+}$

$A_0 = A$  0 0 0 1 0 1 : 0 0 1 0 0 0 = 0. 1 0 1

+  $B_k$  0 1 1 0 0 0

---

1. step

0 1 1 1 0 1  $\rightarrow$  negative  $\rightarrow$  restoring!  $\Rightarrow +B \rightarrow A_0 - B < 0, q = 0$

+ B 0 0 1 0 0 0

---

~~A~~ 0 0 1 0 1  $\rightarrow$  during the restoring, the OV bit is unnecessary

2.  $A_0$  0 0 1 0 1 0  $\leftarrow$  after shifting is compulsory

+  $B_k$  0 1 1 0 0 0

---

2. step

$A_1 = 2A_0 - B$   1 0 0 0 1 0  $\rightarrow 2A_0 - B > 0$

$2A_1$  0 0 0 1 0 0  $\rightarrow$  restoring is not needed

$2A_1 + B_k$  0 1 1 0 0 0

---

3. step

+ B  0 1 1 1 0 0  $\rightarrow$  negative,  $2A_1 - B < 0 \rightarrow$  restoring!

0 0 1 0 0 0

---

~~A~~ 2 0 0 1 0 0

$2A_2$  0 0 1 0 0 0

+  $B_k$  0 1 1 0 0 0

---

4. step

$A_3$   1 0 0 0 0 0  $\rightarrow$  The result  $-A_3$  is zero, this is the end of the division

# Binary Division

- Division in complement code :

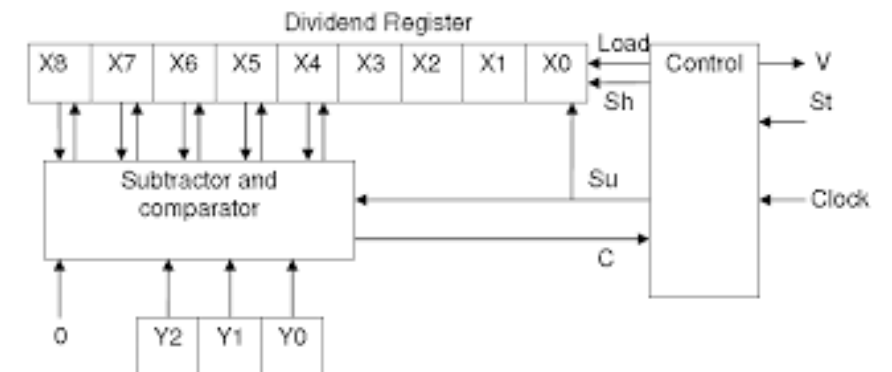
- Algorithm:
- we make the addition and subtraction in turn, and if the signed bits are the same, then the actual bit of the quotient is 1 and we have to make a subtraction, if the signed bit is different, than the actual bit of the quotient is 0, and we have to make an addition in the next step.

- $A = \text{dividend}, B = \text{divisor}, A_i = \text{partial sum}$

- $$A_1 = \begin{cases} 2 * A_0 - B \text{ and } q_1 = 1, \text{ if } \text{sign}(A_0) = \text{sign}(B) \\ 2 * A_0 + B \text{ and } q_1 = 0, \text{ if } \text{sign}(A_0) \neq \text{sign}(B) \end{cases}$$

- ...

- $$A_i = \begin{cases} 2 * A_{i-1} - B \text{ and } q_i = 1, \text{ if } \text{sign}(A_{i-1}) = \text{sign}(B) \\ 2 * A_{i-1} + B \text{ and } q_i = 0, \text{ if } \text{sign}(A_{i-1}) \neq \text{sign}(B) \end{cases}$$



source: <http://needpixies.com/binary-divider-schematic.html>

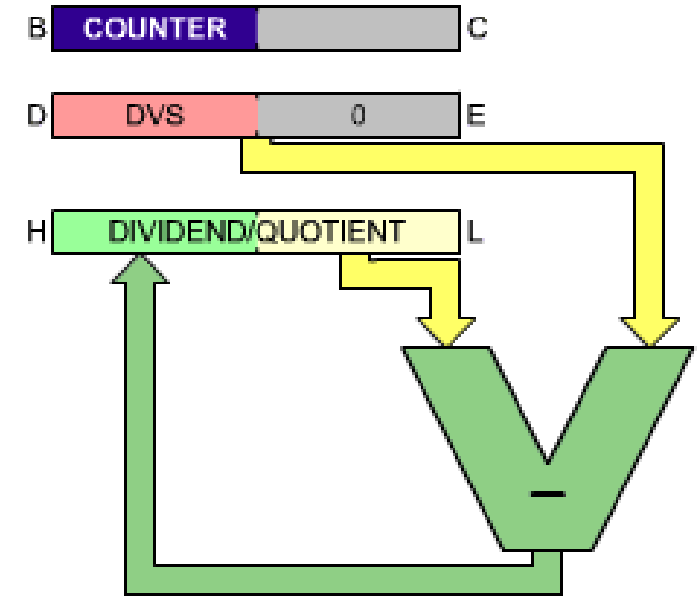
# Binary Division

- $A_i = 2 * A_{i-1} + (1 - 2 * q_i) * B$
- $A_1 = 2 * A_0 + (1 - 2 * q_1) * B$
- $A_2 = 2 * A_1 + (1 - 2 * q_2) * B =$
- $= 2^2 * A_0 + 2 * (1 - 2 * q_1) * B + (1 - 2 * q_2) * B$
- ...
- $A_n = 2^n * A_0 + B * [2^{n-1}(1 - 2q_1) + 2^{n-2}(1 - 2q_2) + \dots + (1 - 2q_n)]$

- $* \frac{2^{-n}}{B}$

- $\frac{A * 2^{-n}}{B} = \frac{A_0}{B} + (2^{-1} + 2^{-2} + \dots + 2^{-n}) - 2(q_1 2^{-1} + q_2 2^{-2} + \dots + q_n 2^{-n})$

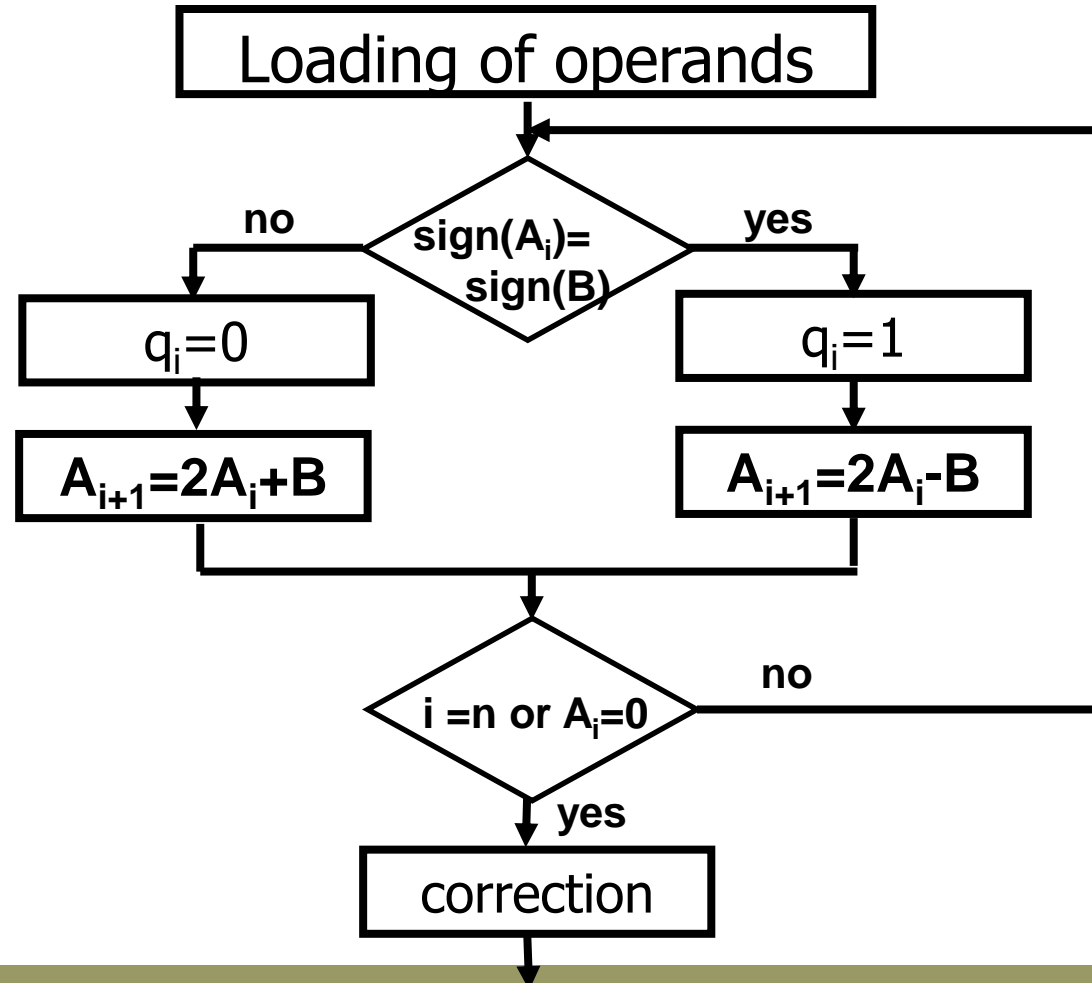
- $\frac{A}{B} = -1 + 2^{-n} + 2 \sum_{i=1}^n q_i 2^{-i} + \frac{2^{-n} A_n}{B}$



source: <http://rvbelzen.tripod.com/z80prgtemp/z80prg03.htm>

# Binary Division

- Algorithm of the complement coded division in the ALU



# Binary Division

- Example: 3/12

$$\begin{array}{r}
 A_0 = \underline{0011} \div 1100 = 10 \\
 \text{shift left } 2A_0 \quad \underline{0110} \leftarrow \\
 \text{sign } A_0 = \text{sign } B \rightarrow -B \quad - \underline{1100} \Rightarrow q_1 = 1 \\
 \hookrightarrow 2A_0 - B = A_1 \quad \underline{11010} \Rightarrow 2^1 \text{ complement of } -6 \\
 \text{shift left } 2A_1 \quad \underline{11010} \leftarrow \\
 \text{sign } A_1 \neq \text{sign } B \quad + \underline{1100} \Rightarrow q_2 = 0 \\
 \hookrightarrow 2A_1 + B = A_2 \quad \underline{1000000} \\
 \text{cy} \nearrow \text{signed}
 \end{array}$$

# Binary Division

- Example: 3/12

Correction,  $n=2$

$2^0$	$2^{-1}$	$2^{-2}$	
1	0		$q_i^0$
-1		+1	Correction $\rightarrow 2^0$ and $2^{-4}$
<hr/>			
0	0	1	

# Binary Division

- Example:  $-3/12$

2's complement  
↓ code of -3

$$\begin{array}{r} \underline{11101} \\ 11101 \quad \text{2A}_0 \rightarrow \text{shift left} \\ + 01100 \quad \text{Sign A}_0 \neq \text{Sign B} \Rightarrow +B \\ \hline 100110 \quad \text{A}_1 \\ \text{cy} \uparrow \frac{1}{5} \\ \text{2A}_1 \rightarrow \text{shift left} \quad 001100 \\ \text{Sign A}_1 = \text{Sign B} \Rightarrow -B \\ \hline 001100 \\ - \quad \underline{00000} \\ \hline 001100 \quad \text{A}_2 \end{array} \quad \div \quad \underline{01100} = 01$$

$q_1 = 0$

$q_2 = 1$

# Binary Division

- Example:  $-3/12$

Correction  $n=2$

$$2^0 \quad 2^{-1} \quad 2^{-2}$$

$$0 \quad 1$$

$$\begin{array}{r} -1 \qquad \qquad \qquad +1 \\ \hline 1. \quad 1 \quad 1 \end{array}$$

$q_i$

correction

$2^1$  complement of  $-0.25$

$$= 2 - 0.25 = 1.75$$

$$\begin{array}{r} 1 \quad 0. \quad 0 \quad 0 \quad 0 \quad (2) \\ - \quad 0. \quad 0 \quad 1 \quad (0.25) \\ \hline 0 \quad 1. \quad 1 \quad 1 \quad 0 \end{array}$$



# Binary Division

- Example: 3/-12

Handwritten binary division example for 3/-12:

Dividend: 0011 ÷

Divisor: 0110

Step 1: Shift left by 2 positions (2A<sub>0</sub>)

Step 2: Sign A<sub>0</sub> ≠ sign B (1 ≠ 0) → +B

Step 3: 2A<sub>1</sub>

Step 4: Sign A<sub>1</sub> = sign B (-1 = -1) → -B

Step 5: Carry (cy) →

2's complement of -12: 10100 = 01

2's complement of -0.25: 1.011

Correction table (n=2):

$2^0$	$2^{-1}$	$2^{-2}$
0	1	q <sub>i</sub>
-1		+1 corr
<hr/>		
1	0	1 1

# Binary Division

- Example:  $-3/-12$

Handwritten binary division for  $-3/-12$  using 2's complement.

$2^5$  complement code of  $-3$ :  $\underline{11101}$   
 $2^5$  complement code of  $-12$ :  $\underline{10100} = 10$

Shift left,  $2A_0$ :  $111010$  (with a red arrow pointing left)  
 Sign  $A_0 = \text{sign } B \Rightarrow -B$ :  $\underline{10100}$   $q_1 = 1$

$A_1$ :  $\underline{00110}$   
 $2A_1$ :  $001100$   
 Sign  $A_1 \neq \text{sign } B \Rightarrow +B$ :  $\underline{10100}$   $q_2 = 0$

$C_4$ :  $\underline{100000}$

Arrows indicate that the quotient bits  $q_1 = 1$  and  $q_2 = 0$  are placed above the final result  $10$ .

# Binary Division

- Example:  $-3/-12$

correction,  $u = 2$

$2^0$	$2^{-1}$	$2^{-2}$	
1	0		
-1		+1	
<hr/>			
0.	0	1	

$q_i$   
conv



**BME**



**KJIT**



*Budapest University of Technology and Economics*

*Faculty of Transportation Engineering and Vehicle Engineering*

*Department of Control for Transportation and Vehicle Systems*

**End of Lecture 5.**

**Thank you for your attention!**



**BME**

*Budapest University of Technology and Economics*



**KJIT**

*Faculty of Transportation Engineering and Vehicle Engineering*

*Department of Control for Transportation and Vehicle Systems*

# REALIZATION OF BINARY OPERATIONS

Lecture 6.

# Binary Division

Budapest University of Technology and Economics

Faculty of Transportation Engineering and Vehicle Engineering

Department of Control for Transportation and Vehicle Systems

- <https://www.youtube.com/watch?v=K79wfflmLNo>



**BME**

*Budapest University of Technology and Economics*



**KJIT**

*Faculty of Transportation Engineering and Vehicle Engineering*

*Department of Control for Transportation and Vehicle Systems*

**End of Lecture 6.**

**Thank you for your attention!**