Budapest University of Technology and Economics
Faculty of Transportation Engineering and Vehicle Engineering
Department of Control for Transportation and Vehicle Systems

REALIZATION OF BINARY OPERATIONS

Lecture 6.

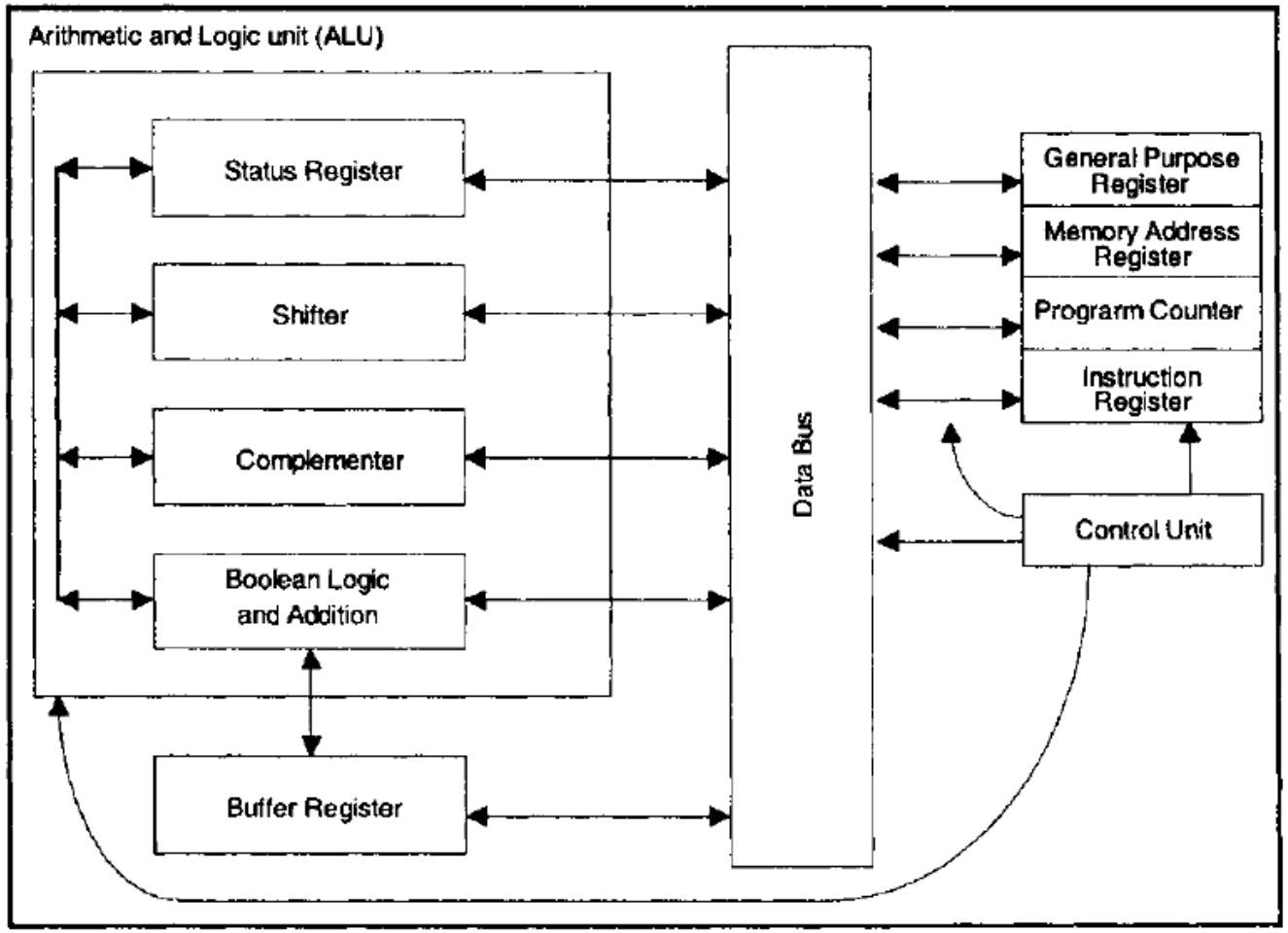
ALU

- Binary operations are realized in the ALU, that can be structured in:
  - serial form:
    - the operations are performed bitwise, from the lowest local value toward to the higher local value,
  - parallel form:
    - all the operations are performed in one step in every local values,
  - mixed form:
    - generally used.

- Every type of artithmetic operations can be realized by addition:
  - substraction,
  - multiplication,
  - division.

ALU

- ALU:

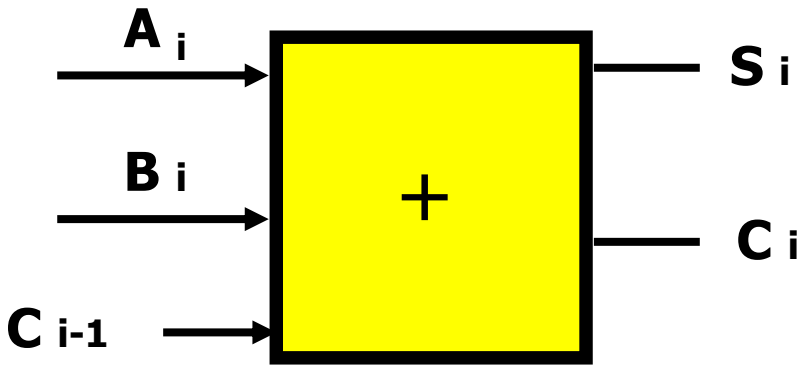


source: M. Rafiquzzaman, Fundamentals of Digital Logic and Microcomputer Design, 5th Edition

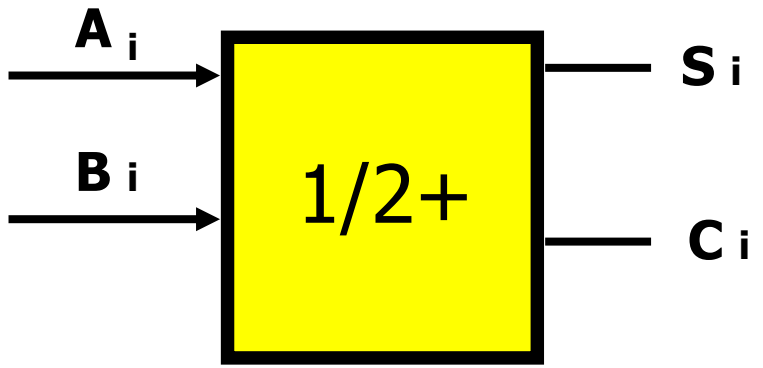
Adder

- Due to the importance of the addition, the time required to add numbers plays an important role in determining the speed of the ALU,

- Main types of the 1-bit adders:
  - full-adder:



  - half-adder:

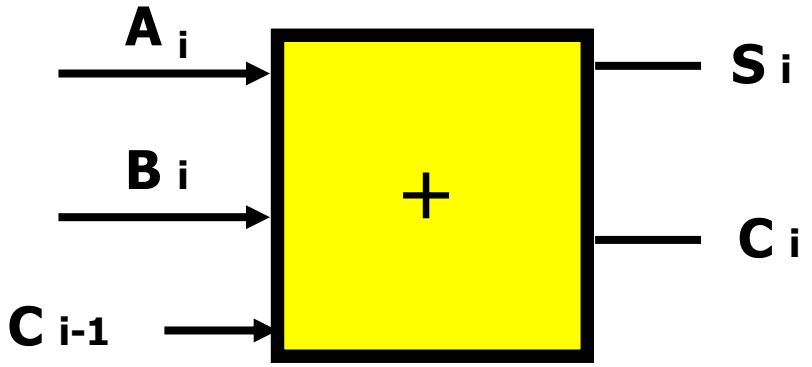


Adder

- If we add three (2+CY) bits:

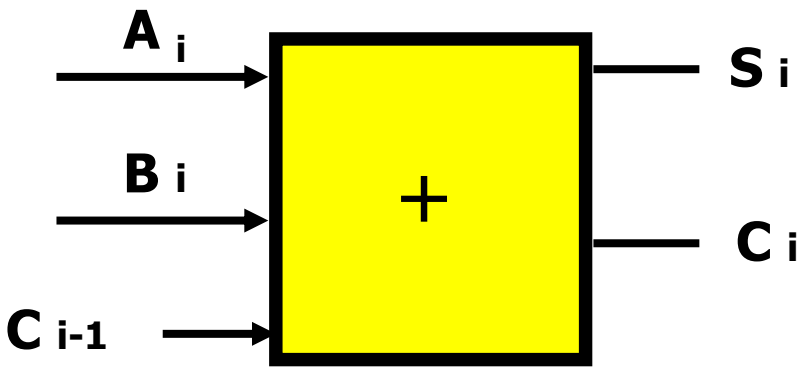
CY CY	CY CY	CY CY	CY CY	CY CY
0_2	0_2	0_2	1_2	1_2
$+0_2$	$+1_2$	$+0_2$	$+0_2$	$+1_2$
0_2	1_2	1_2	10_2	11_2

- full adder: adds binary numbers and accounts for values carried in as well as out,
  - a one-bit full adder adds three one-bit numbers, where A_i and B_i are the operands and C_{i-1} is a bit carried in form the previous less-significant stage,
  - a full adder is usually a component of adders, which adds 8, 16, 32, 64, etc bits binary numbers.



Adder

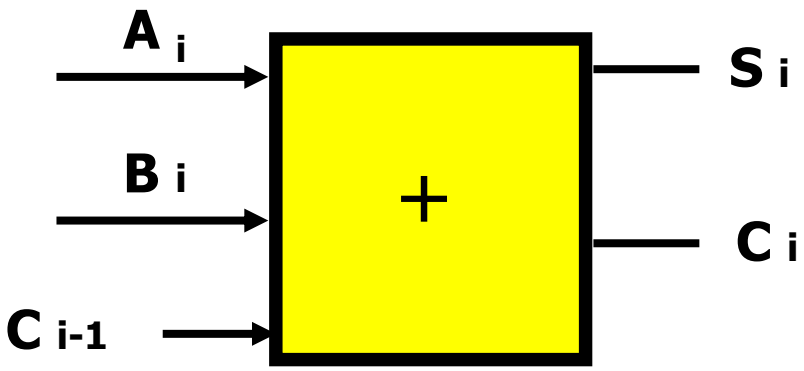
- Thruth table of a one-bit full-adder:

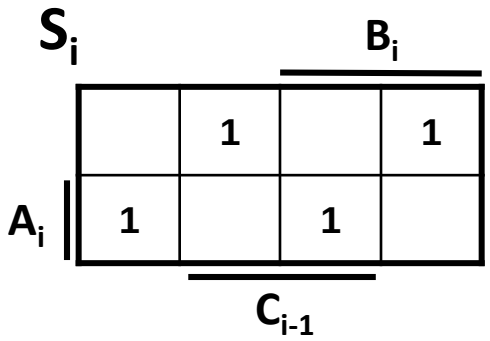


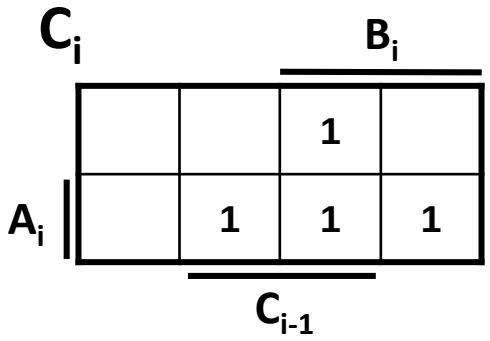
A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Adder

- Karnaugh maps:

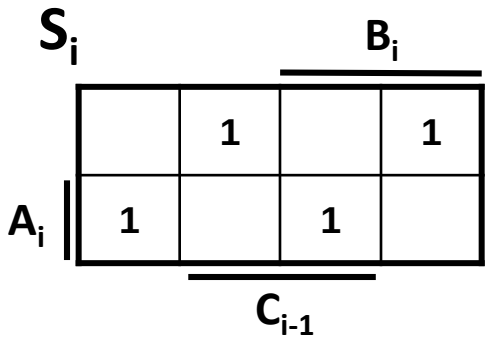


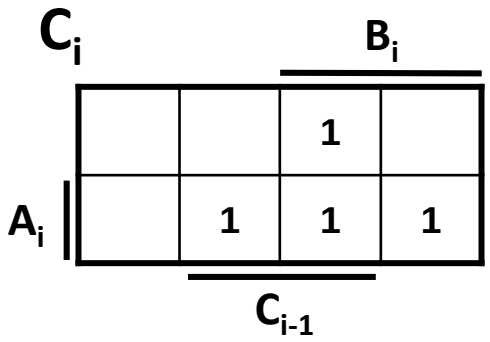




Adder

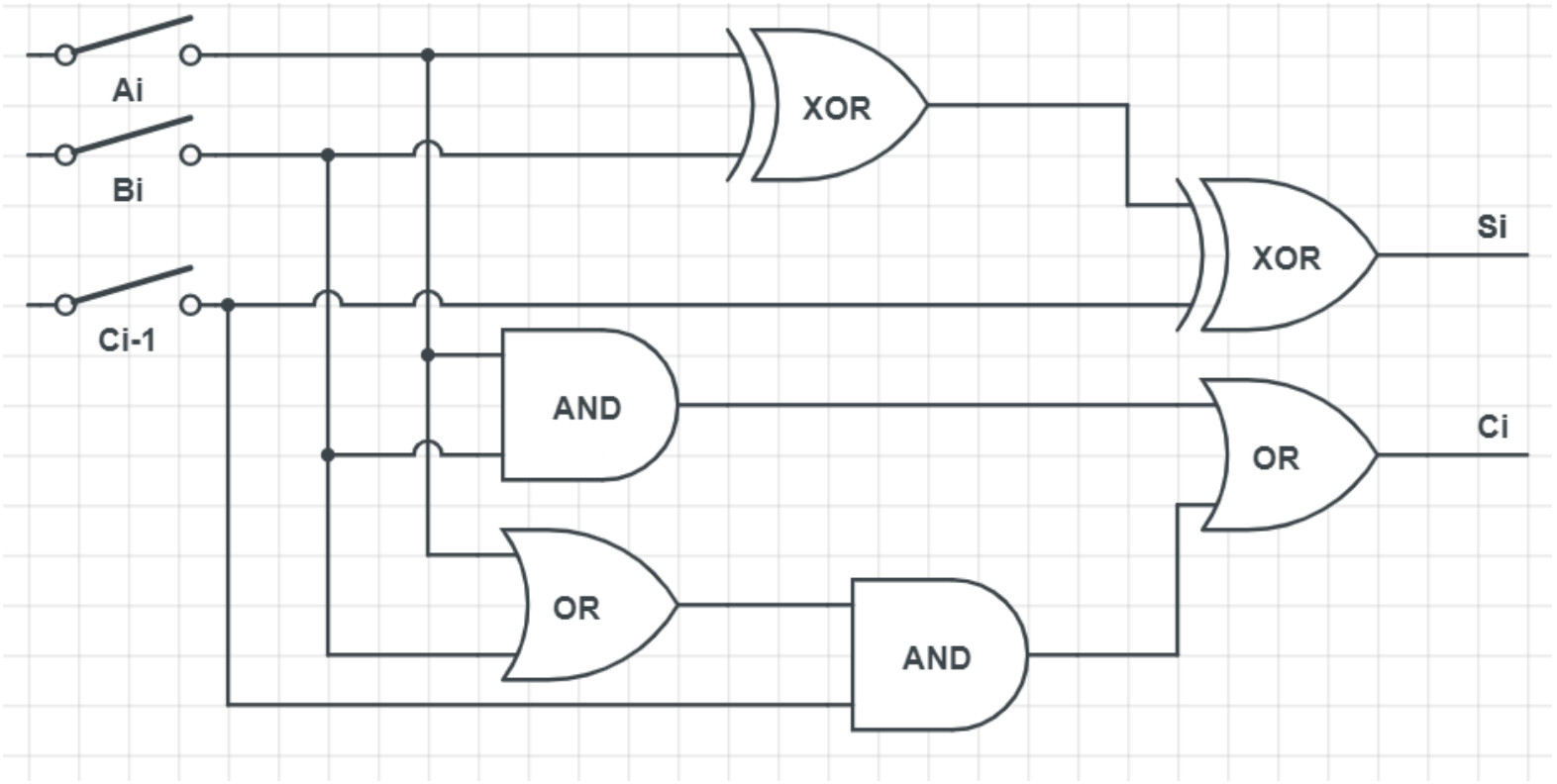
- Logical functions:
  - $S_i = A_i \oplus B_i \oplus C_{i-1}$ (XOR)
  - $C_i = A_i B_i + C_{i-1}(A_i + B_i)$





Adder

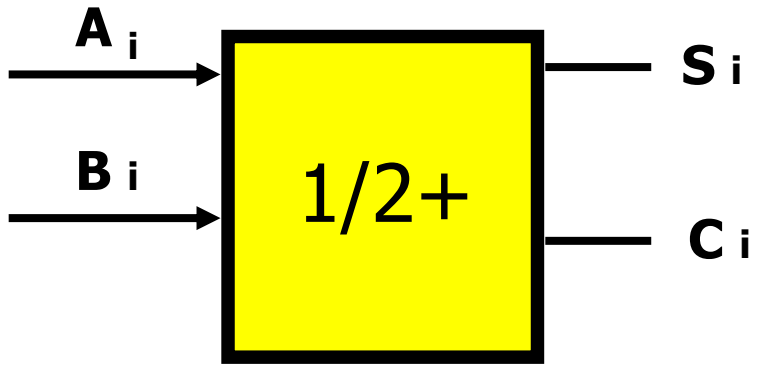
- Realization:
  - $S_i = A_i \oplus B_i \oplus C_{i-1}$ (XOR)
  - $C_i = A_i B_i + C_{i-1}(A_i + B_i)$



Adder

- If we add two bits:

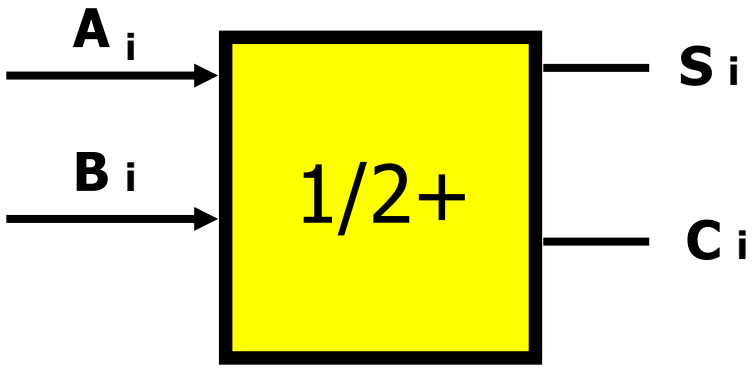
CY	CY	CY
0_2	0_2	1_2
$+0_2$	$+1_2$	$+1_2$
0_2	1_2	10_2



- full adder: adds two single binary digits,
  - a one-bit half-full adder adds two one-bit numbers, where A_i and B_i are the operands,
  - it has two outputs, where the carry represents an overflow in to the next digit.

Adder

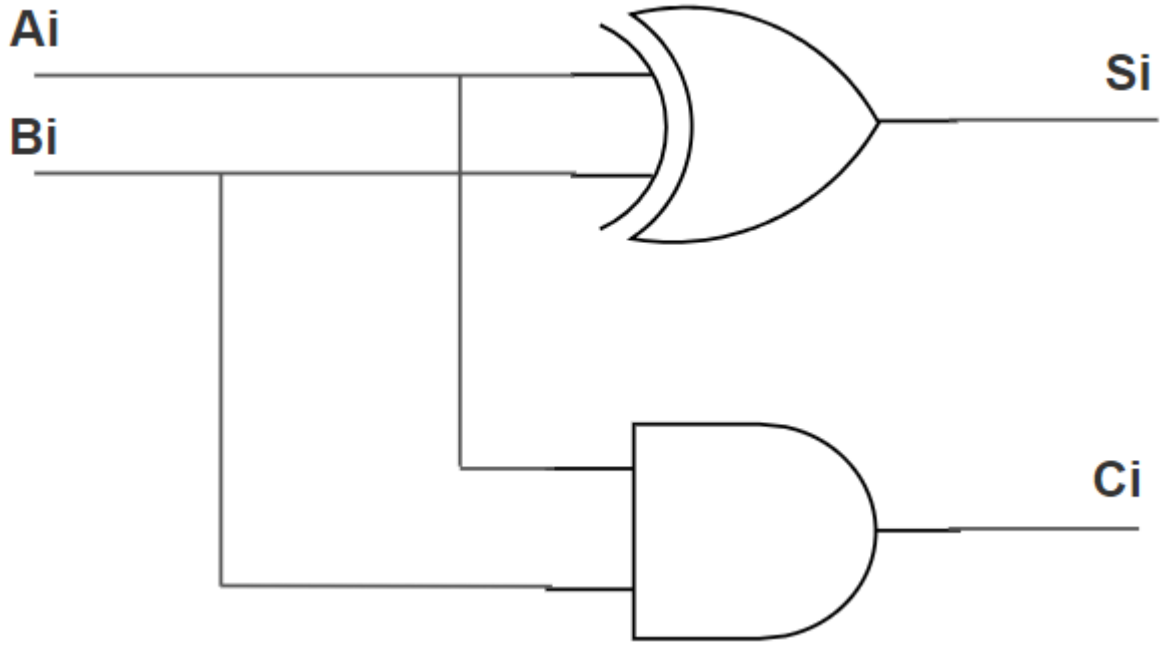
- Thruth table of a one-bit half-adder:



A_i	B_i	S_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

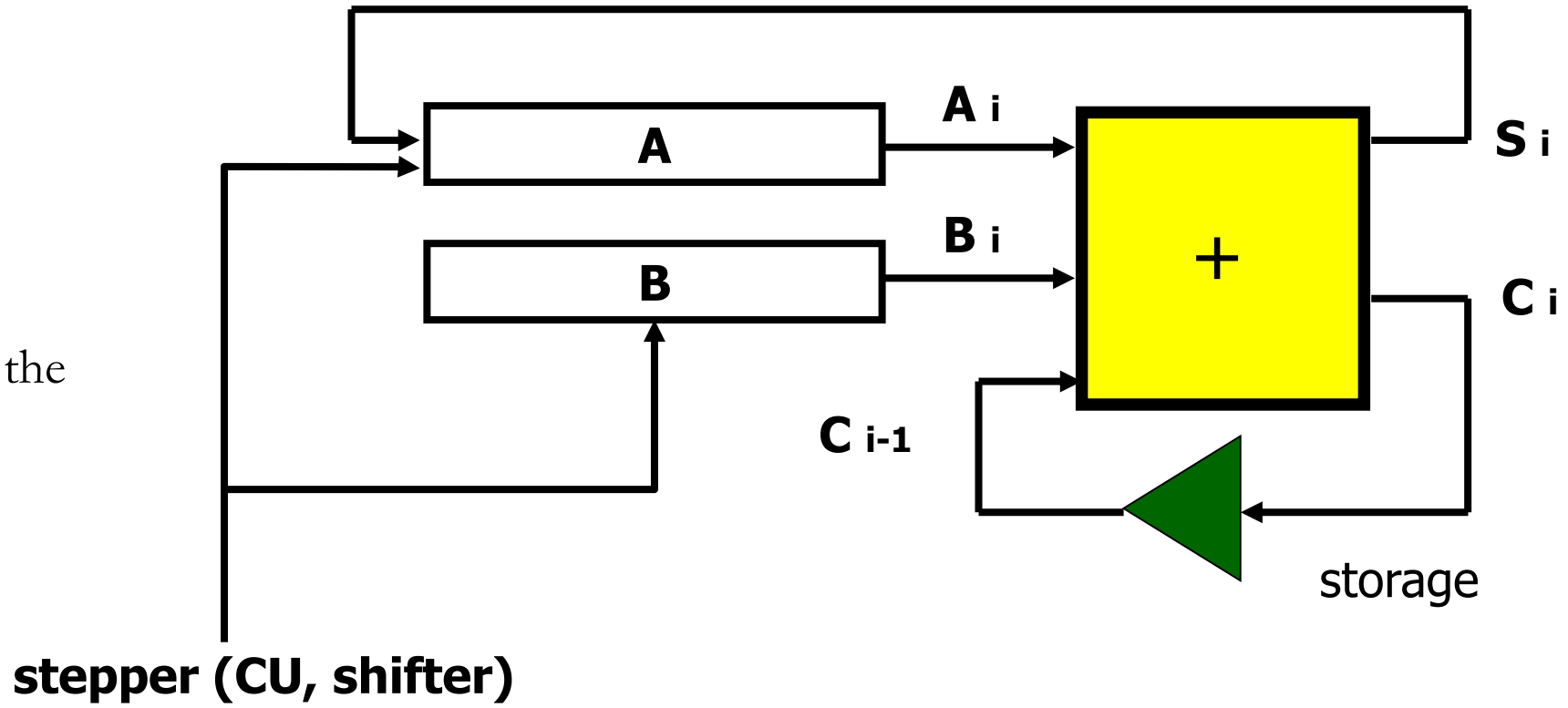
Adder

- Logical functions and realization:
  - $S_i = A_i \oplus B_i$ (XOR)
  - $C_i = A_i B_i$



Adder

- Complex Adders:
  - to add more digits,
- Serial Adder:
  - the result will be in the operand A,
  - it is slow.



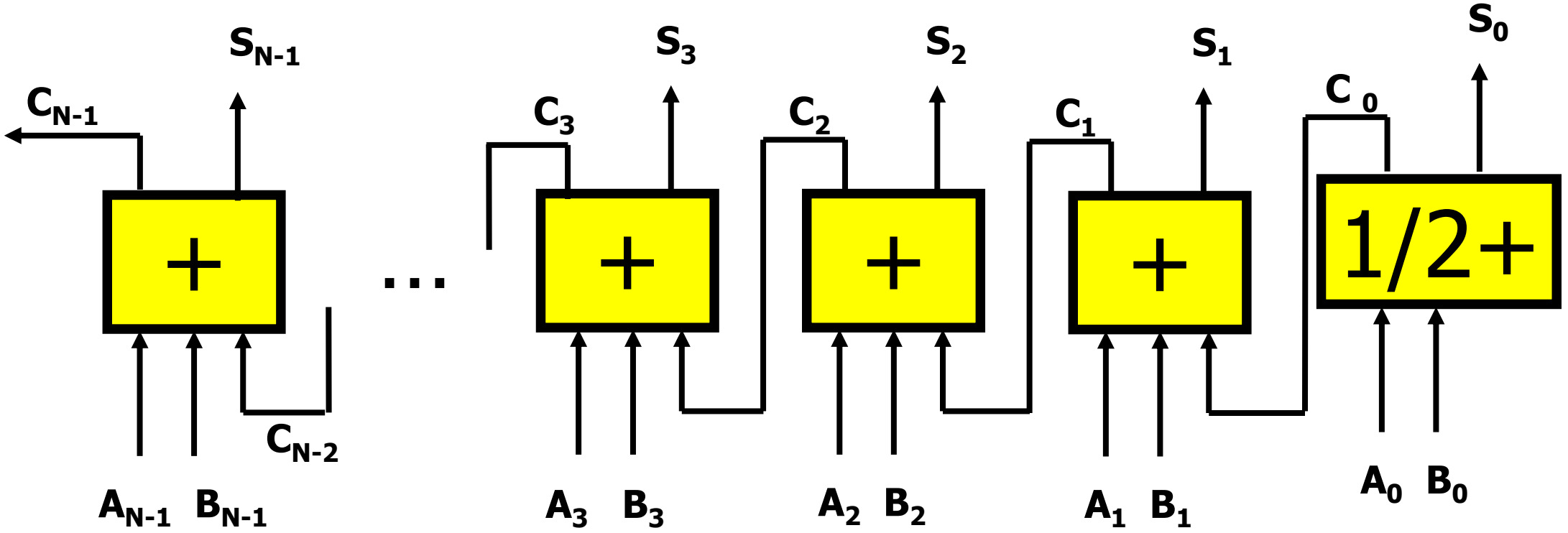
Adder

- Complex Adders:
  - to add more digits,

- Riple-Carry Adder (paralell adder):
  - to add N-bits,
  - in this case, the adder is simple, that allows fast design time,
  - the ripple-carry adder is relative slow, because each full-adder must wait for the carry bit, calculated from the previous adder, this is called the gate-daley (Δt),
  - if the gate delay of a full-adder is 3*Δt, the result will be correct in time: n* Δt, e.g the total gate delay in a case of addition of two 32 bits number: 31*3* Δt (full adders) + 1* Δt (half-adder), the total delay=94* Δt
  - if this time is not acceptable, it must to accelerate the addition

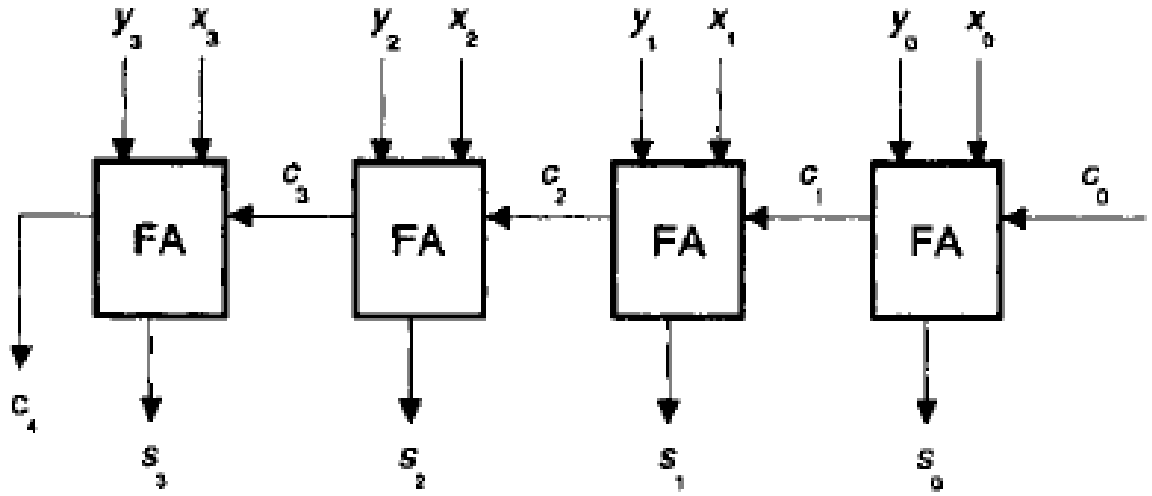
Adder

- Riple-Carry Adder (paralell adder):



Adder

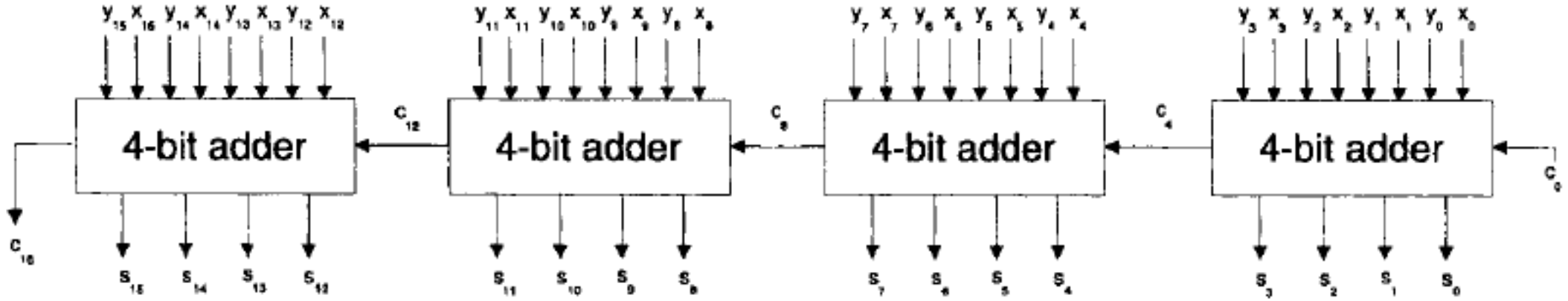
- Riple-Carry Adder (paralell adder):
  - it is possible to build smaller units – 4-bit Riple-Carry Adder (or carry-prpagated adder, CPA) -, because the carry is propagated serially through each full adder.



source: M. Rafiquzzaman, Fundamentals of Digital Logic and Microcomputer Design, 5th Edition

Adder

- Riple-Carry Adder (paralell adder):
  - 16-bits CPA:



source: M. Rafiquzzaman, Fundamentals of Digital Logic and Microcomputer Design, 5th Edition

Adder

- Carry-lookahead adder:

- Idea: it is needed to determine the carry before the addition
  - carry-look ahead logic uses the concept of generating and propagating carries,
  - in the case of binary addition, A+B generates carry, if, and only if both A and B are 1.
    - G(A,B)=A*B
  - the addition of two 1-digit inputs A and B is said to be propagate, if the addition will carry whenever there is an input carry,
  - in the case of binary addition, A+B propagates a carry, if and only if at least one of A or B is 1
    - P(A,B)=A+B

Adder

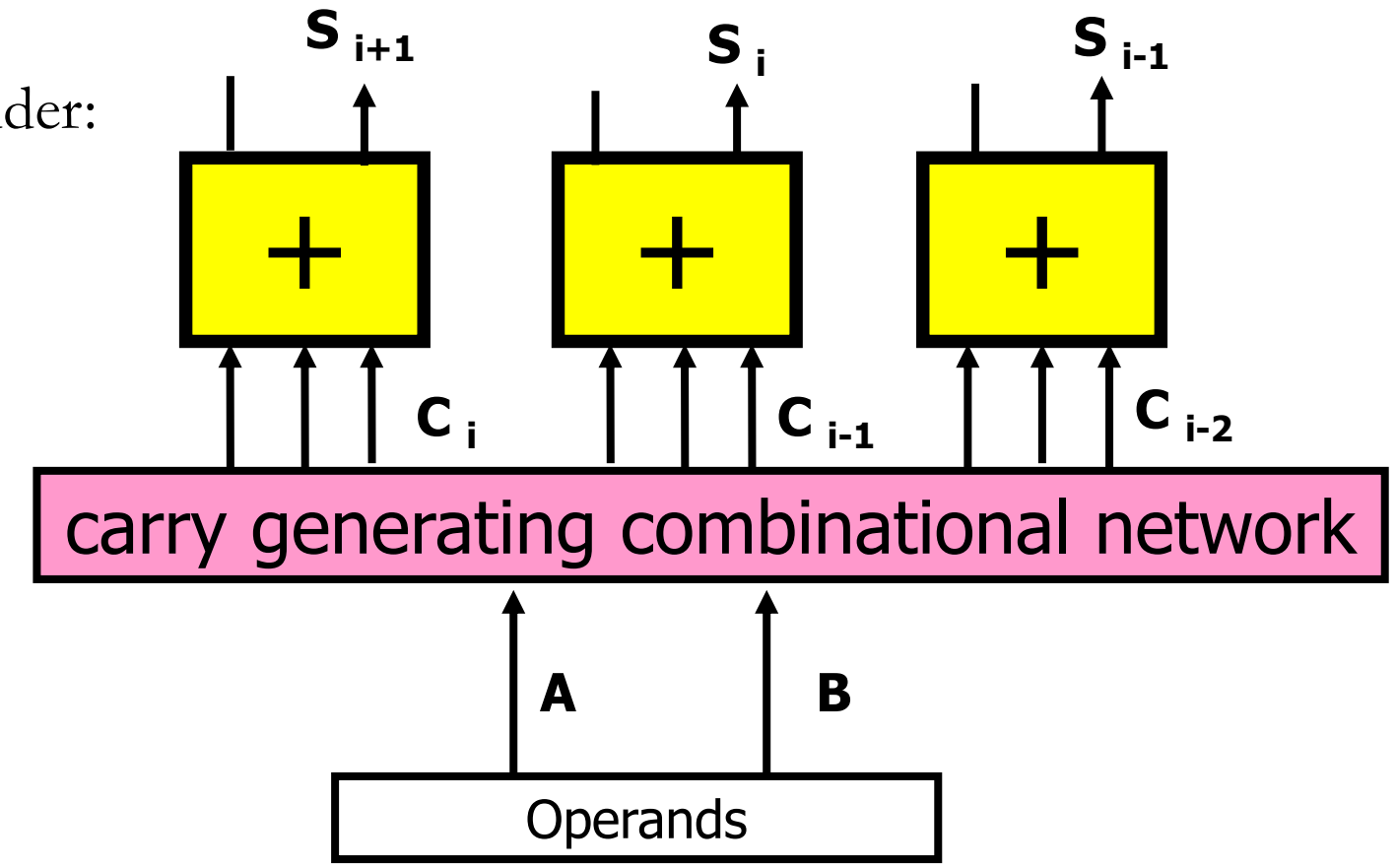
- Recursive Transfer Training method:

- $C_i = G_i + (P_i * C_{i-1})$

- $C_0 = G_0 = A_0B_0$
- $C_1 = G_1 + P_1C_0 = A_1B_1 + (A_1 + B_1)A_0B_0$
- $C_2 = G_2 + P_2C_1 = A_2B_2 + (A_2 + B_2)[A_1B_1 + (A_1 + B_1)A_0B_0] =$
- $= A_2B_2 + (A_2 + B_2)(A_1B_1 + A_0A_1B_0 + A_0B_0B_1) =$
- $= A_2B_2 + A_1A_2B_1 + A_0A_1A_2B_0 + A_0A_2B_0B_1 + A_1B_1B_2 + A_0A_1B_0B_2 + A_0B_0B_1B_2$
- …
- It means a complicated 2-levels combinational logical network, and contains the gate delays
- e.g. a standard 16 bit adder would take 46 gate delays, with this method it is just 5 (2+3) gate delays

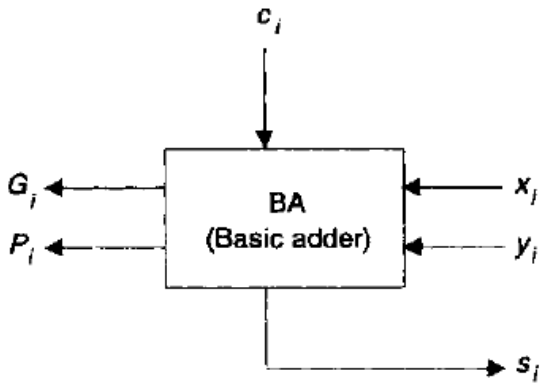
Adder

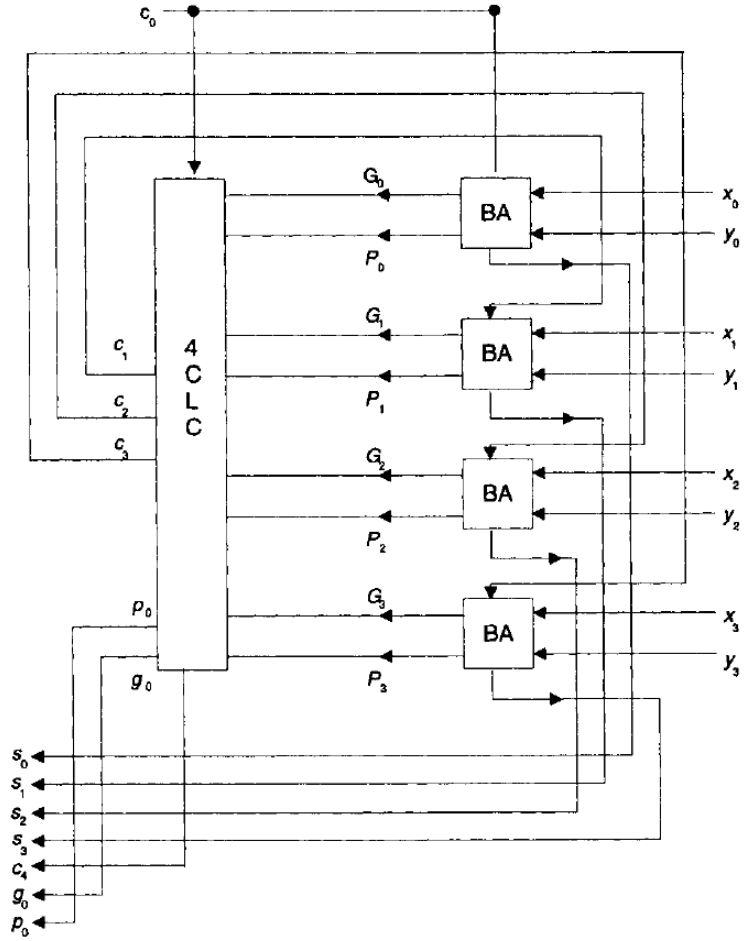
- Carry-lookahead adder:



Adder

- Carry-lookahead adder:
  - 4-bits CLA, e.g.
  - BA: Basic-Adder





source: M. Rafiquzzaman, Fundamentals of Digital Logic and Microcomputer Design, 5th Edition

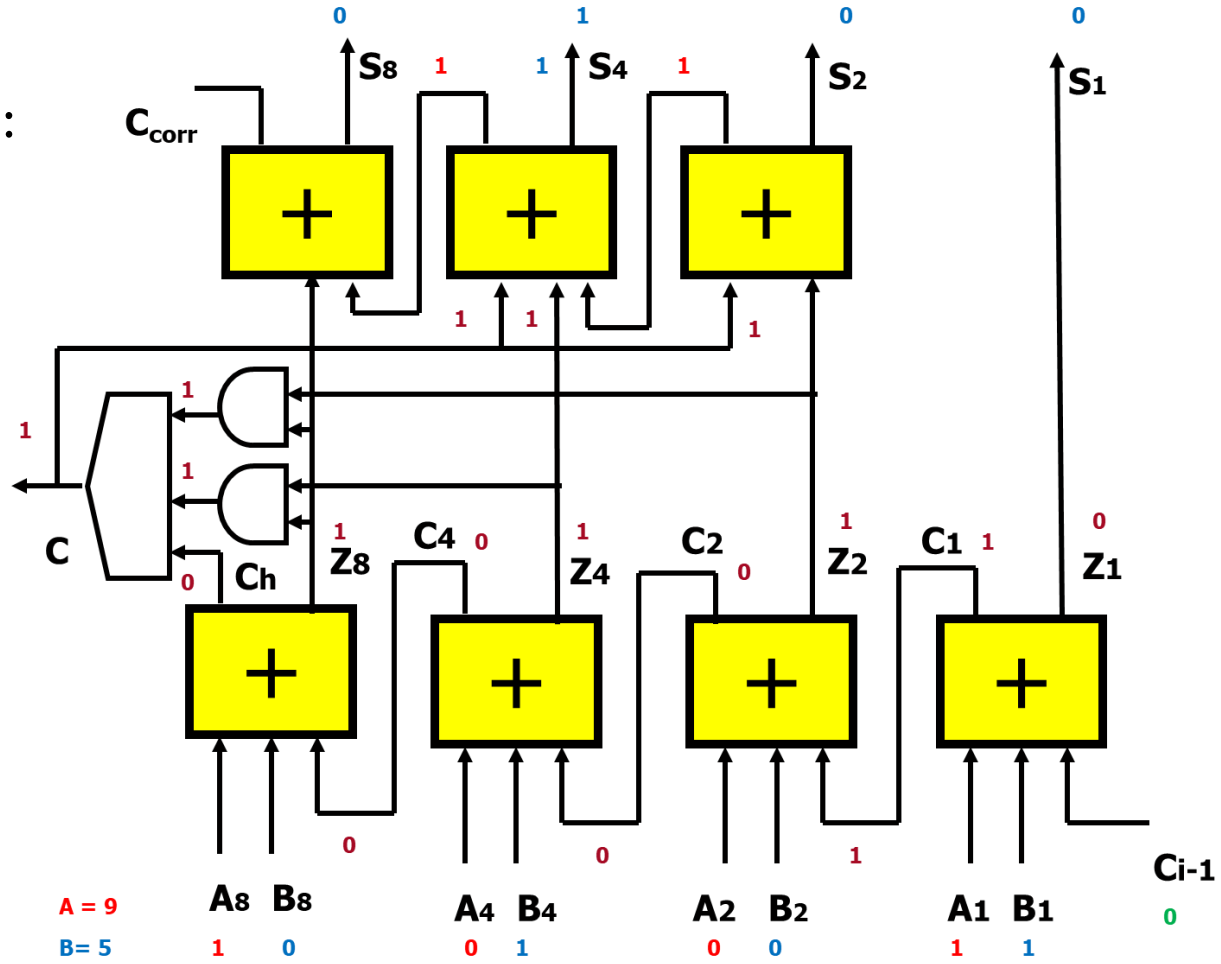
Adder

- BCD Adder:
  - Truth Table (see earlier):
  - we have to create an addition if the result is beetween 9<Res<16,
  - if the result is bigger then 15, it is needed to create the decimal adjust
  - $C=Z_4*Z_8+Z_2*Z_8+C_h$

A_i+B_i	wrong Res $Z_8\ Z_4\ Z_2\ Z_1$	good Res $S_8\ S_4\ S_2\ S_1$	DA	correction
0	0000	0000	No DA	Not necessary
1	0001	0001		
2	0010	0010		
3	0011	0011		
4	0100	0100		
5	0101	0101		
6	0110	0110		
7	0111	0111		
8	1000	1000		
9	1001	1001		
10	1010	0000	No DA, it has to generate it	+6 (+0110)
11	1011	0001		
12	1100	0010		
13	1101	0011		
14	1110	0100		
15	1111	0101		
16	(1)0000	0110	it generates	
17	(1)0001	0111		
18	(1)0010	1000		

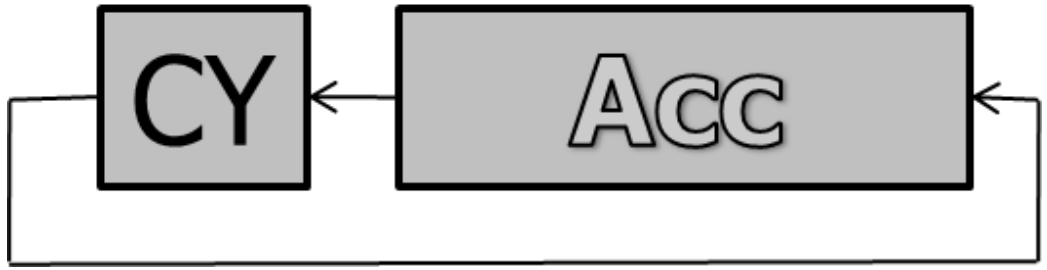
Adder

- BCD adder for two tetrades:

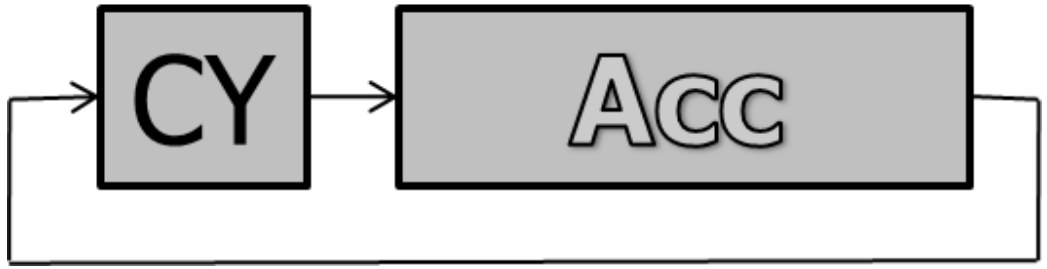


Shifting

- Multiplication by 2:
  - or with 2^x



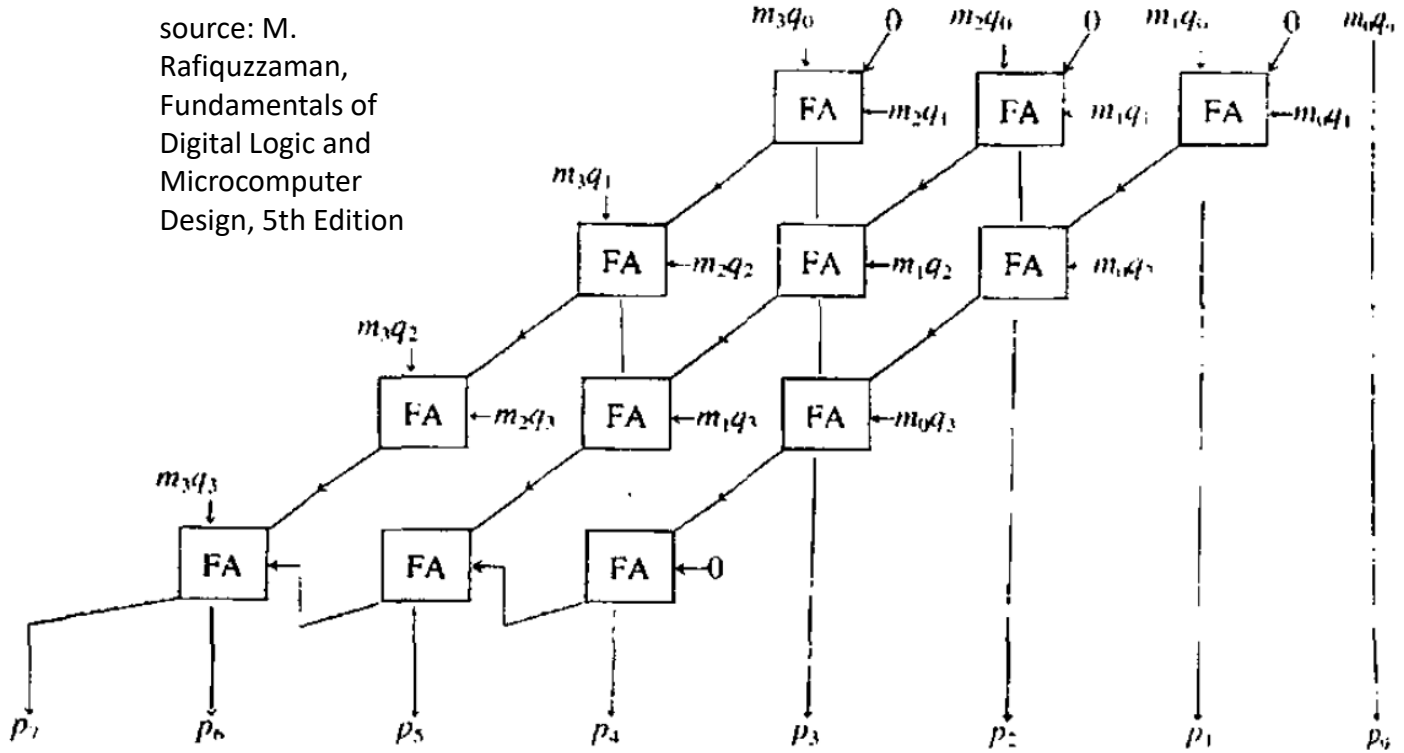
- Division by 2:
  - or with 2^x



Multiplication

- Multiplication by leftforward shihting, 4x4 Array Multiplier:

source: M. Rafiquzzaman, Fundamentals of Digital Logic and Microcomputer Design, 5th Edition



ALU

- https://www.youtube.com/watch?v=K79wfflmLNo
- https://www.youtube.com/watch?v=1I5ZMmrOfnA
- https://www.youtube.com/watch?v=fpnE6UAfbtU
- https://www.youtube.com/watch?v=FZGugFqdr60

End of Lecture 6.

Thank you for your attention!