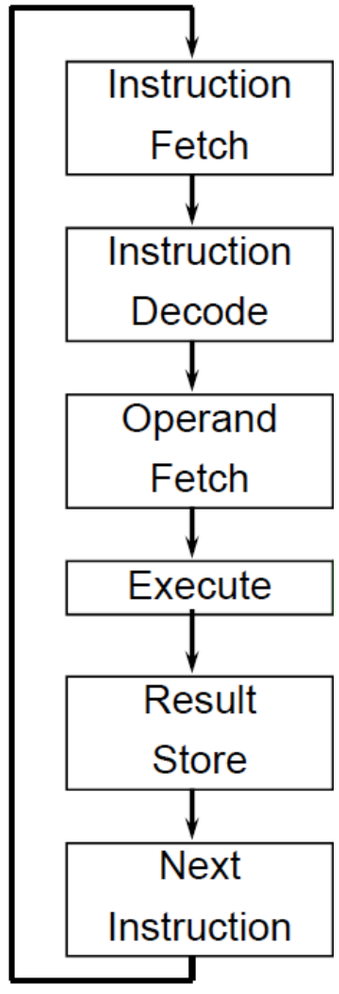
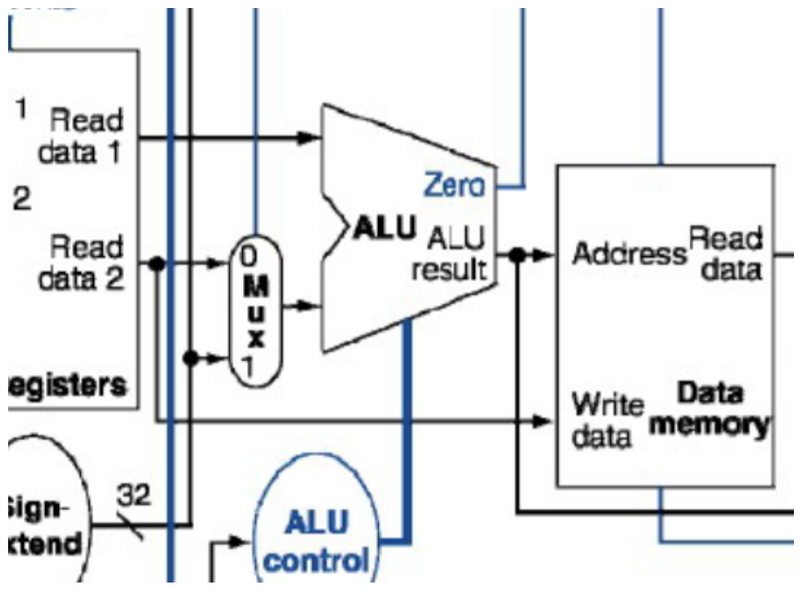
EE116C/CS151B
Fall 2017

Instructor: Professor Lei He
TA: Yuan Liang

Review

- **Arithmetic Logic Unit (ALU) Design**
  - Input: two operand (32-bits), ALU control signal
  - Output: Zero, ALU results, overflow, carryout





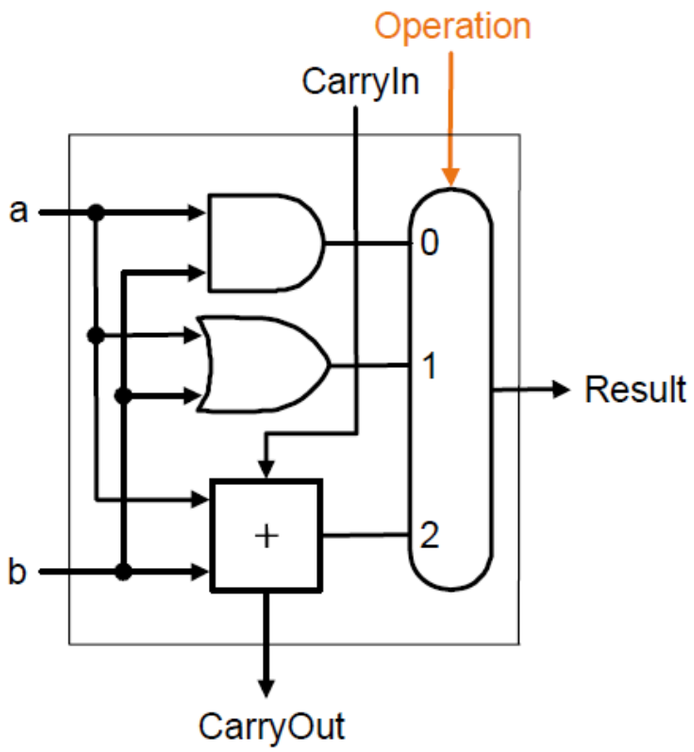
Review

- **Arithmetic Logic Unit (ALU) Design**
  - Function:
    - Add
    - Subtract
    - Overflow dealing
    - And
    - Or
    - Branch
    - Set-On-Less-Than (SLT)

- **How?**

Review

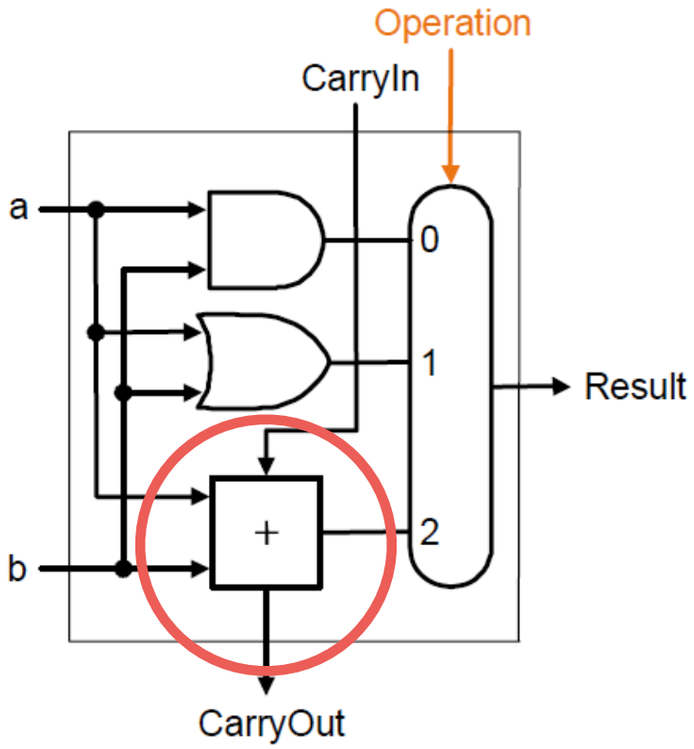
- **1-bit ALU**
  - Input: two operand (1-bits), ALU control signal, **carryin**
  - Output: Zero, ALU results, ~~**overflow**~~, carryout



Operation?
and,
or,
add with carryin,
etc.

Review

- **1-bit ALU**
  - Input: two operand (1-bits), ALU control signal, **carryin**
  - Output: Zero, ALU results, **overflow**, carryout



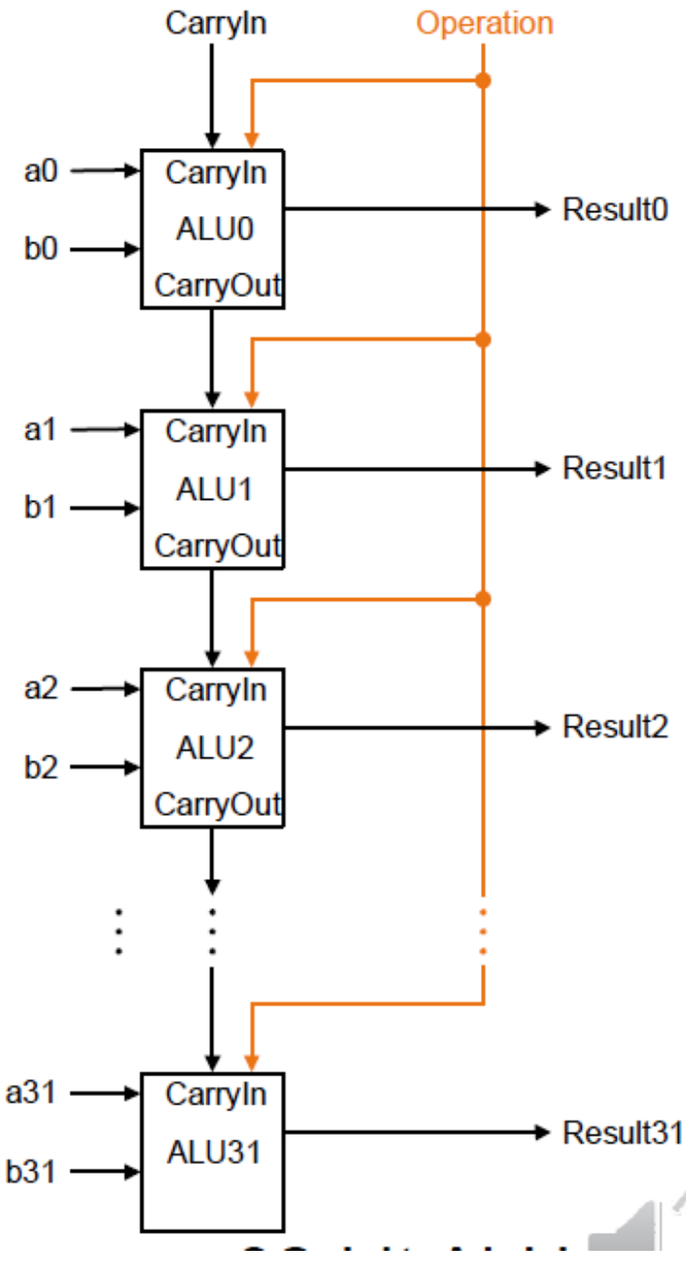
Composed of AND, OR, NOT gates.

CarryOut = (b & CarryIn) | (a & CarryIn) | (a & b)

Sum = (!a & !b & CarryIn) | (!a & b & !CarryIn) | (a & !b & !CarryIn) | (a & b & CarryIn)

Review

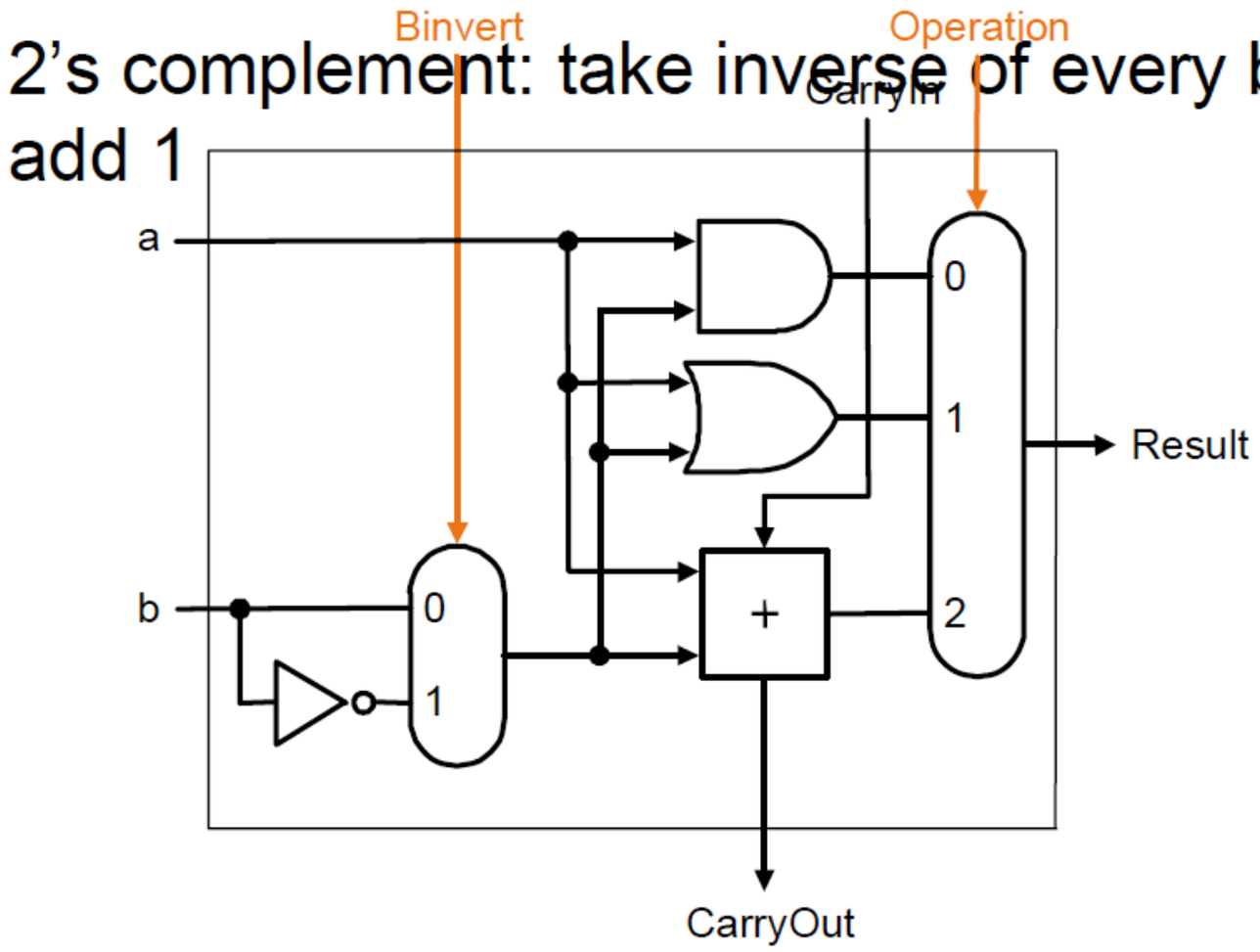
- **Style 1: Ripple Carry Adder (1-bit ALU -> 32-bit ALU)**



- **Function:**
  - **Add**
  - Subtract
  - Overflow dealing
  - **And**
  - **Or**
  - Branch
  - Set-On-Less-Than (SLT)

Review

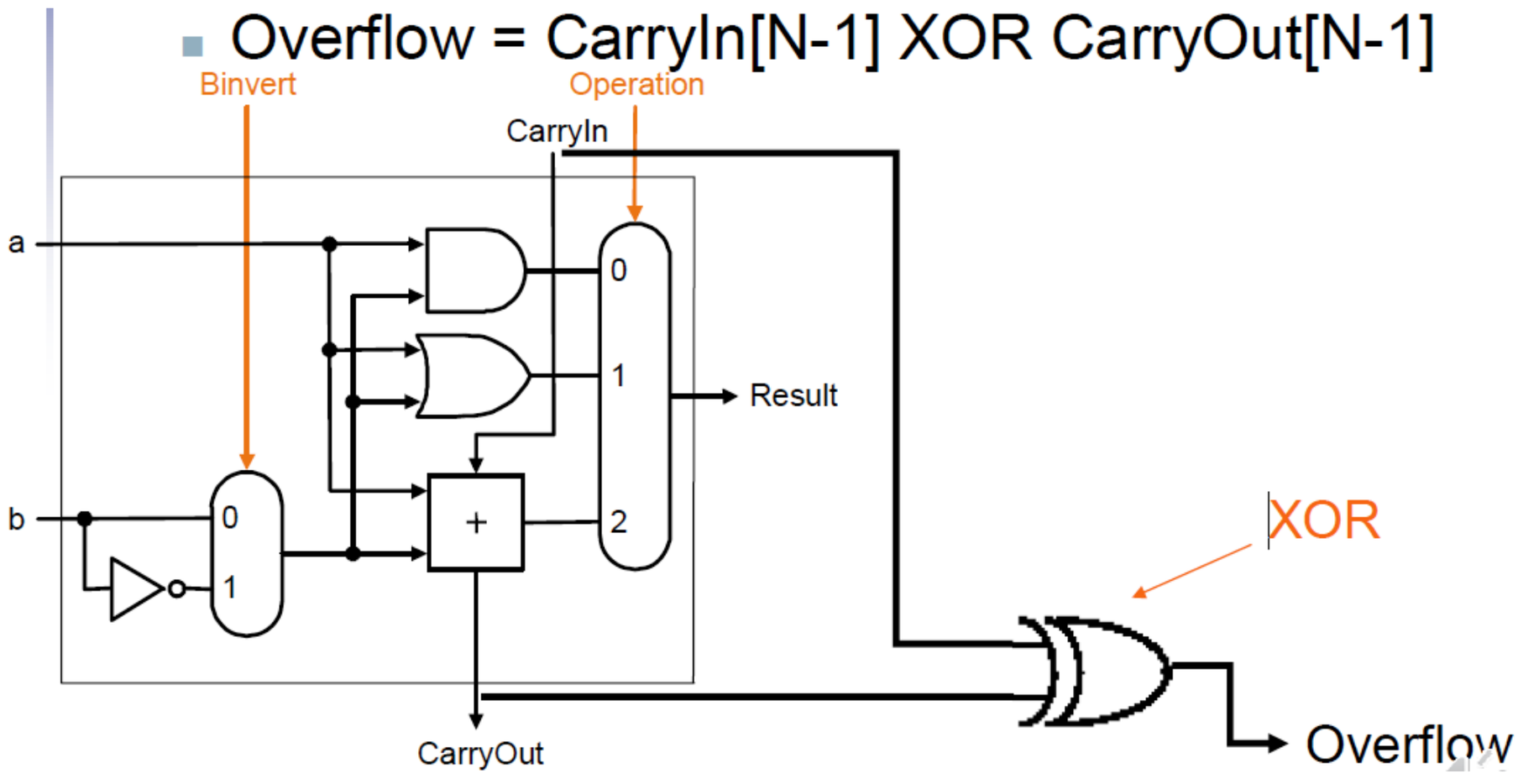
- **Style 1: Ripple Carry Adder (1-bit ALU -> 32-bit ALU)**



- **Function:**
  - **Add**
  - **Subtract**
  - Overflow dealing
  - **And**
  - **Or**
  - Branch
  - Set-On-Less-Than (SLT)

Review

- **Style 1: Ripple Carry Adder (1-bit ALU -> 32-bit ALU)**



only for the most significant bit ALU

- **Function:**
  - **Add**
  - **Subtract**
  - **Overflow dealing**
  - **And**
  - **Or**
  - Branch
  - Set-On-Less-Than (SLT)

Review

- **Style 1: Ripple Carry Adder (1-bit ALU -> 32-bit ALU)**
  - bneq r1, r2, L
  - beq r1, r2, L
  - if (a == b)
  - if (a != b)
  - -> a - b ?= 0

One big NOR gate including all N results.

if any result(i) == 1 -> return 0
if all result(i) == 0 -> return 1

- **Function:**
  - **Add**
  - **Subtract**
  - **Overflow dealing**
  - **And**
  - **Or**
  - **Branch**
  - Set-On-Less-Than (SLT)

Review

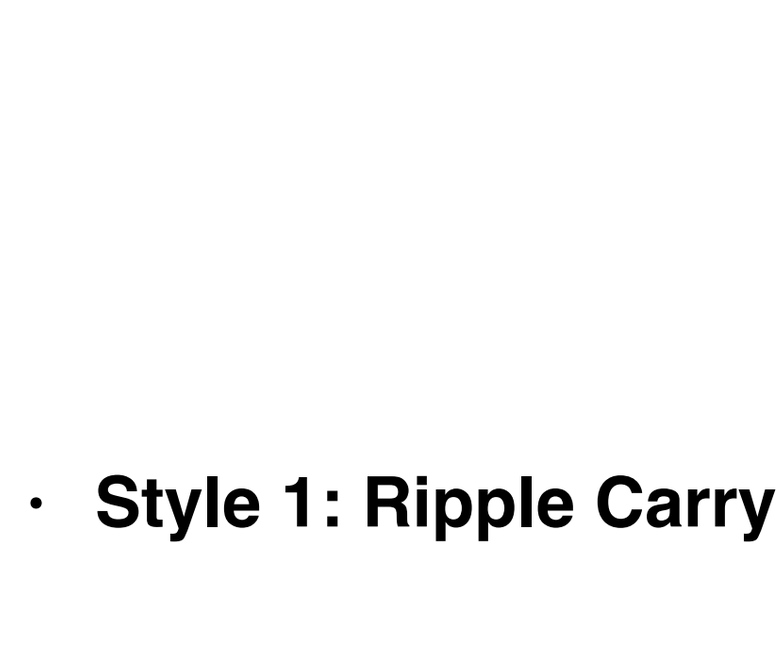
- **Style 1: Ripple Carry Adder (1-bit ALU -> 32-bit ALU)**
  - slt rd, rs, rt
    - if (rs < rt) rd = 1
    - else rd = 0

If R[rs] is input A, and R[rt] is input B, we can subtract, and then take the most significant bit of the result.
- If MSB = 1, then the difference is negative and thus R[rs] is less than R[rt]

- **Function:**
  - **Add**
  - **Subtract**
  - **Overflow dealing**
  - **And**
  - **Or**
  - **Branch**
  - **Set-On-Less-Than (SLT)**

Review

- **Style 1: Ripple Carry Adder (1-bit ALU -> 32-bit ALU)**
  - **Function:**
    - **Add**
    - **Subtract**
    - **Overflow dealing**
    - **And**
    - **Or**
    - **Branch**
    - **Set-On-Less-Than (SLT)**

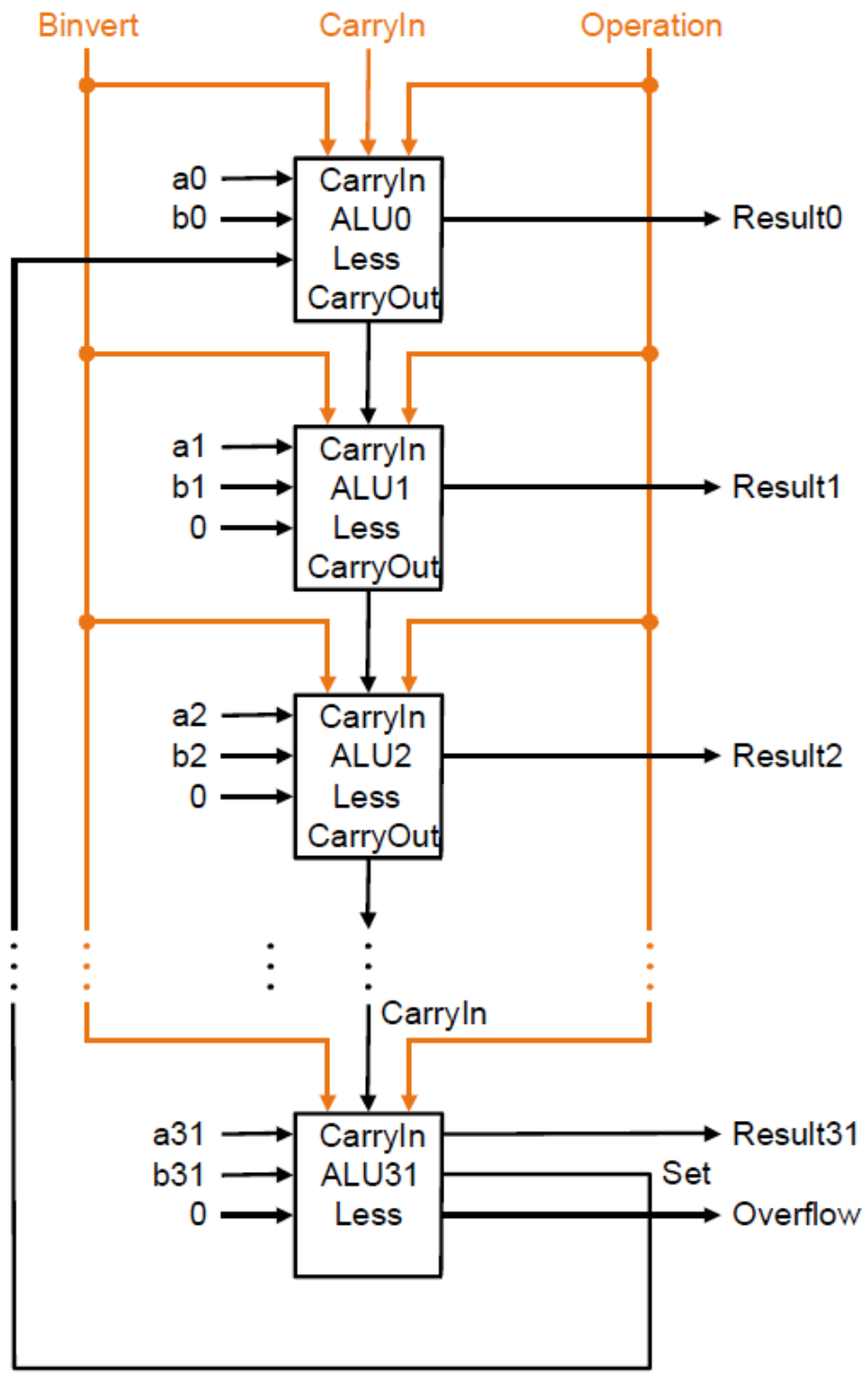


All but MSB

Most Significant Bit

Review

- **Style 1: Ripple Carry Adder (1-bit ALU -> 32-bit ALU)**

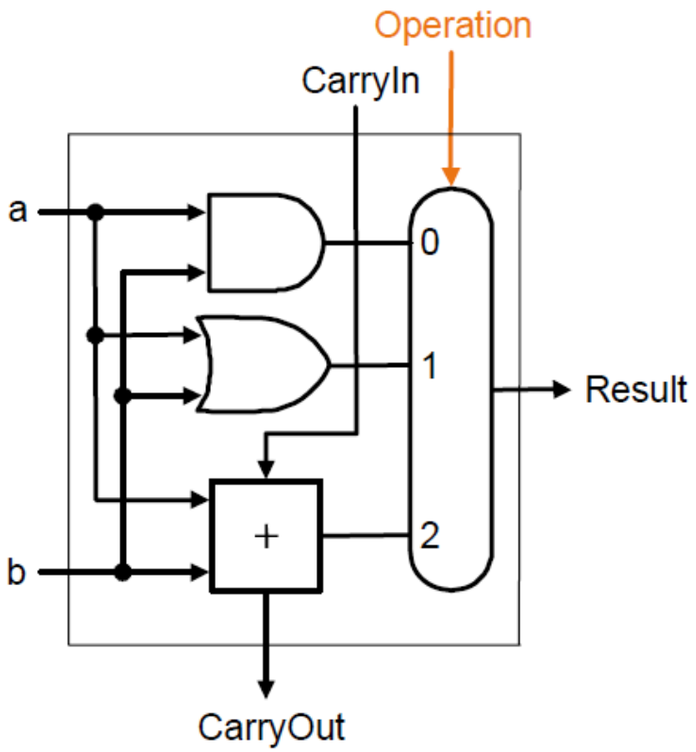


- **Function:**
  - **Add**
  - **Subtract**
  - **Overflow dealing**
  - **And**
  - **Or**
  - **Branch**
  - **Set-On-Less-Than (SLT)**

Review

- **Discussion: ripple style 32-bit adder**

- Worst case delay for N-bit Ripple Carry Adder



CarryOut = (b & CarryIn) | (a & CarryIn) | (a & b)

Delay = 2N

Sum = (!a & !b & CarryIn) | (!a & b & !CarryIn)
| (a & !b & !CarryIn) | (a & b & CarryIn)

Delay = 2N

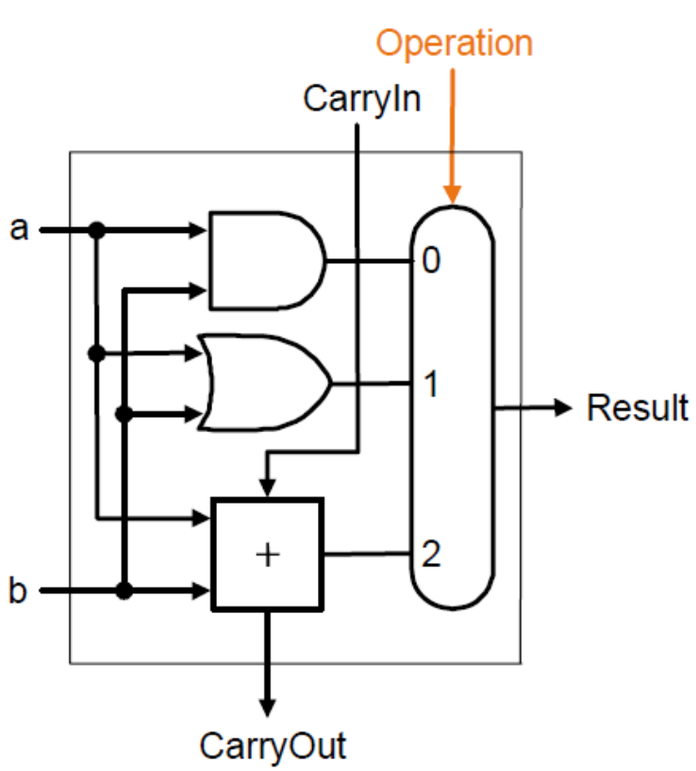
Total delay = 32 * 2N

Review

- **Style 2: Carry Lookahead Adder (1-bit ALU -> 32-bit ALU)**
    - The principle of the Carry Lookahead Adder is to sacrifice space and complexity in the interest of speed.
    - For calculating Cout, we will use two additional signals at 1-bit adder:
        - Generate (Gn) = An * Bn
        - Propagate (Pn) = An ^ Bn

Review

- **Style 2: Carry Lookahead Adder (1-bit ALU -> 32-bit ALU)**



Ripple Carry Adder:

C1 = (A0 * B0) + (A0 * C0) + (B0 * C0)

2T + 3T

Carry Lookahead Adder:

G0 = A0 * B0

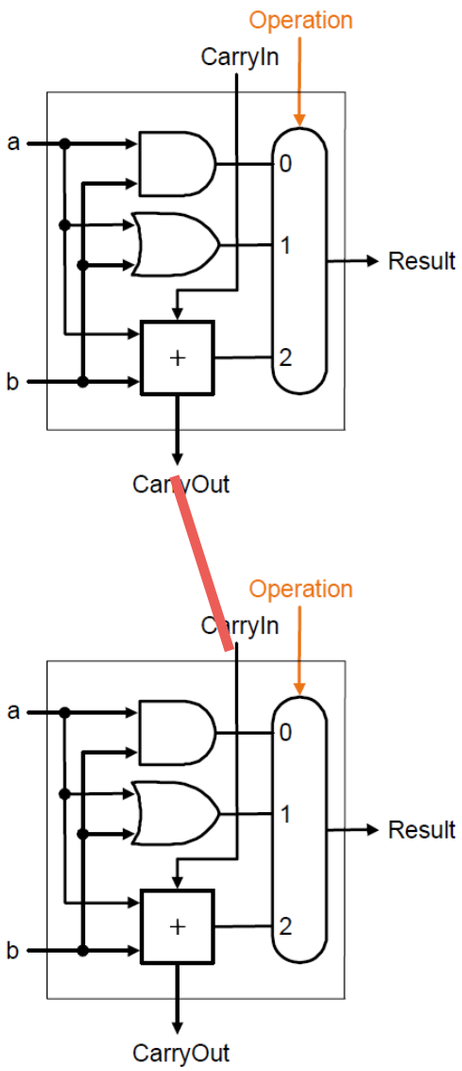
P0 = A0 ^ B0

C1 = G0 + C0 * P0

2T + 2T + 2T

Review

- **Style 2: Carry Lookahead Adder (1-bit ALU -> 32-bit ALU)**



Ripple Carry Adder:

C1 = (A0 * B0) + (A0 * C0) + (B0 * C0)

C2 = (A1 * B1) + (A1 * C1) + (B1 * C1)

2T + 3T + 2T + 3T

Carry Lookahead Adder:

G0 = A0 * B0

P0 = A0 ^ B0

C1 = G0 + C0 * P0

G1 = A1 * B1

P1 = A1 ^ B1

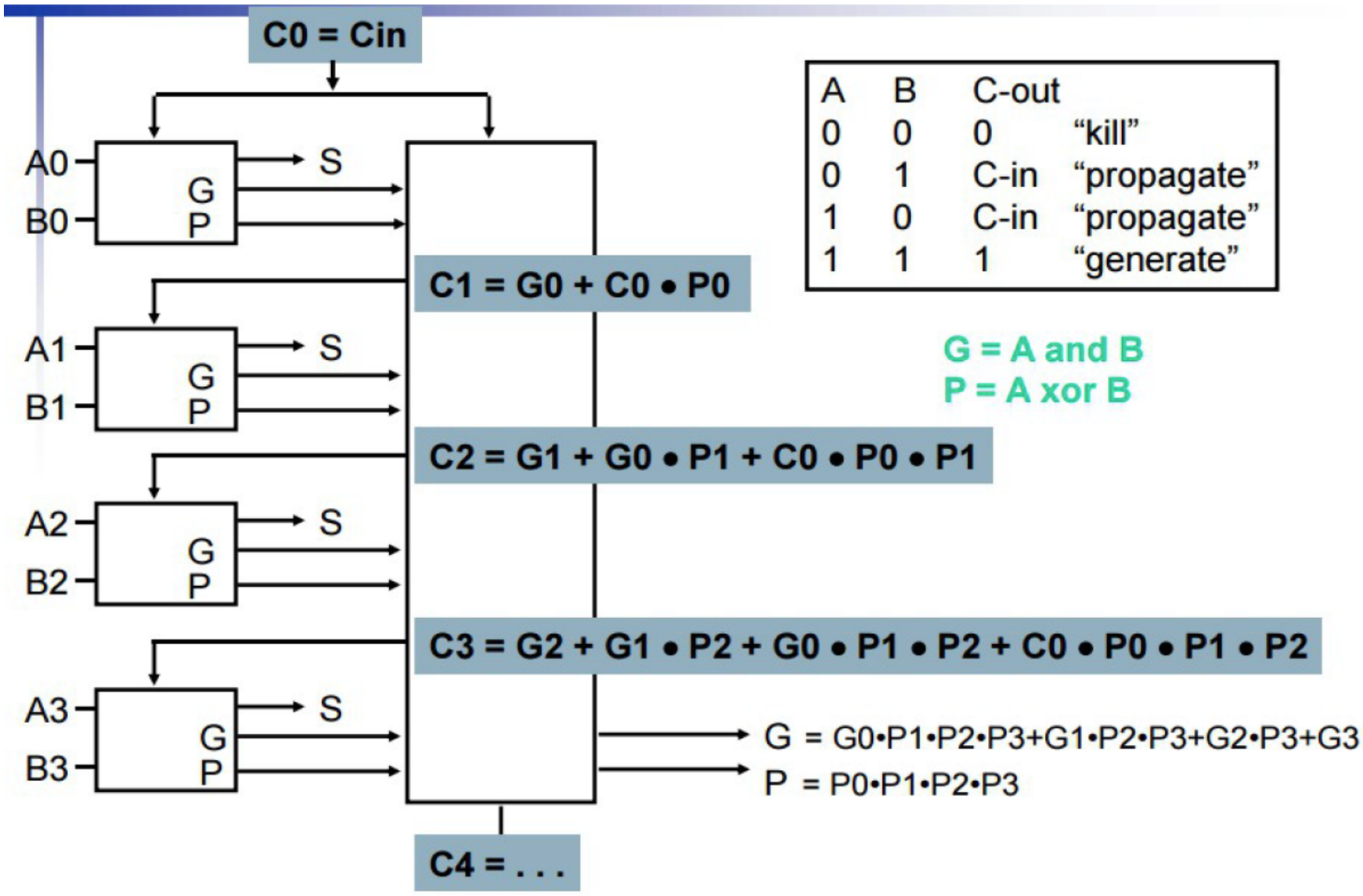
C1 = G1 + C1 * P1 = G1 + (G0 + C0 * P0) * P1

= G1 + G0 * P1 + C0 * P0 * P1

2T + 3T + 3T

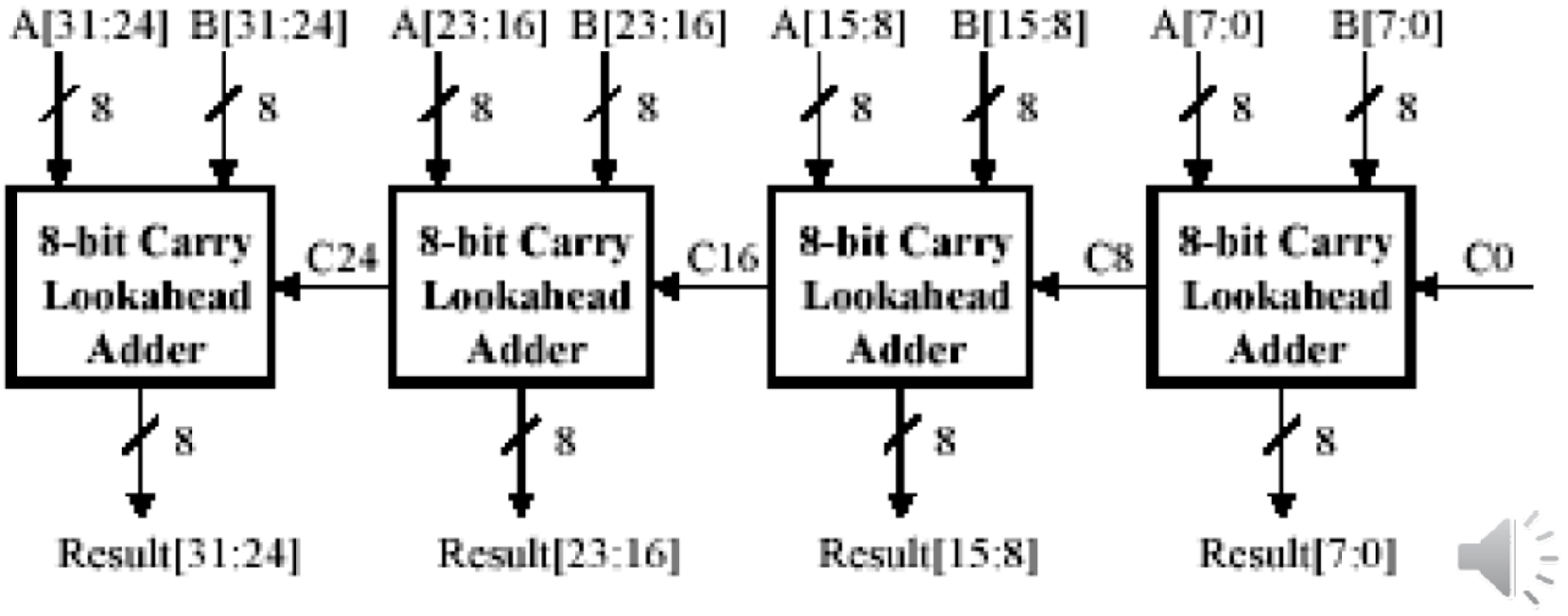
Review

- **Style 2: Carry Lookahead Adder (1-bit ALU -> 32-bit ALU)**



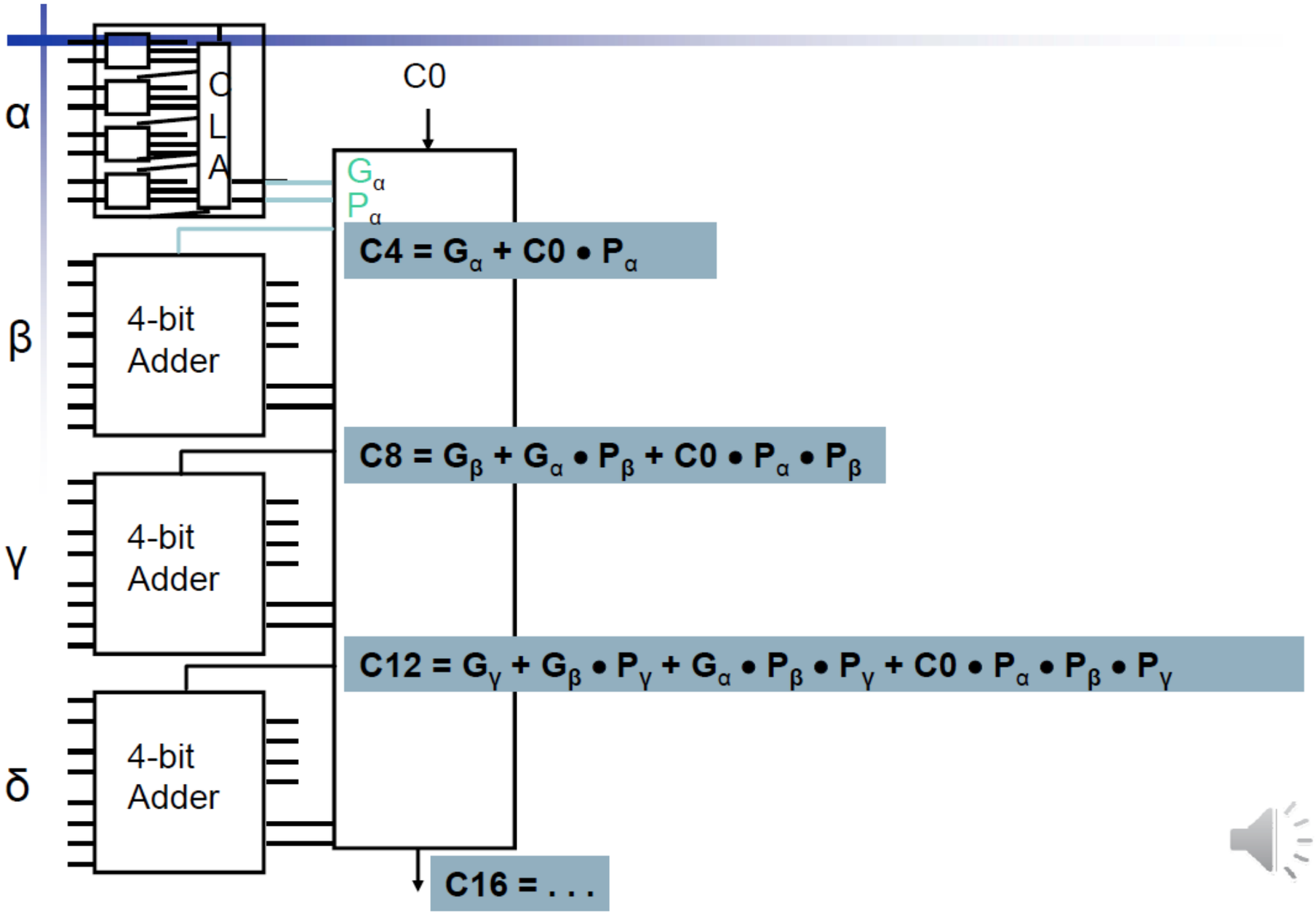
Review

- **Style 2.1: [Partial] Carry Lookahead Adder**
  - Connect several N-bit Lookahead Adders together
  - Four 8-bit carry lookahead adders can form a 32-bit partial carry lookahead adder



Review

- **Style 2.2: [Hierarchical] Carry Lookahead Adder**



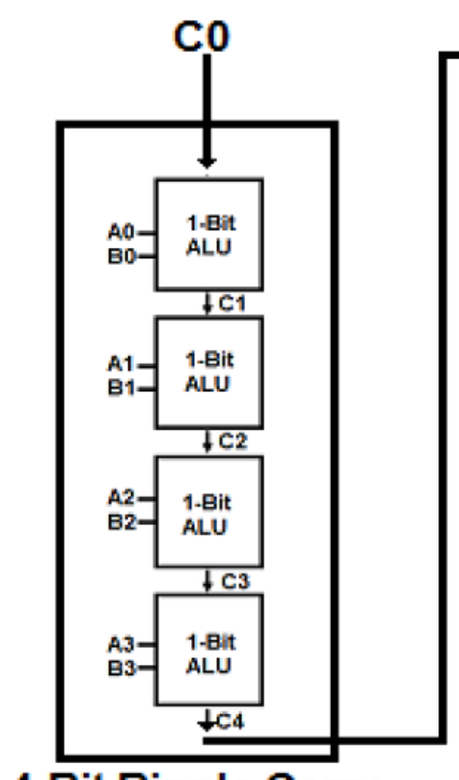
Sample Question 1

Consider the 32-bit adder as shown.

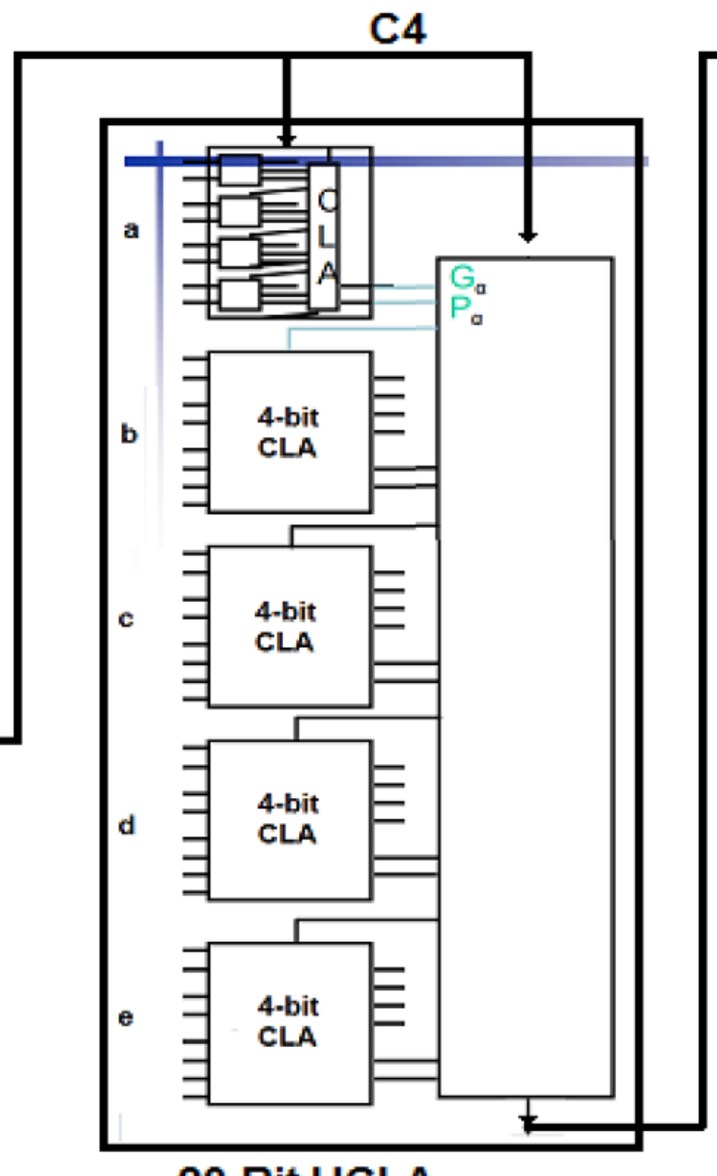
Assume the following.

- Ripple Carry Cn+1 = An*Bn + An*Cn + Bn*Cn
- Sn = Cn^(An^Bn)
- delay time sheet:

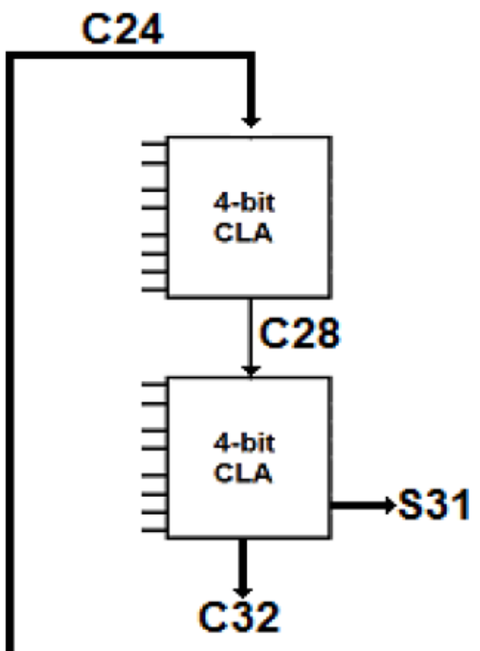
Fan-In	Delay
1	1T
2	2T
3	3T
4	5T
5	8T
6	13T



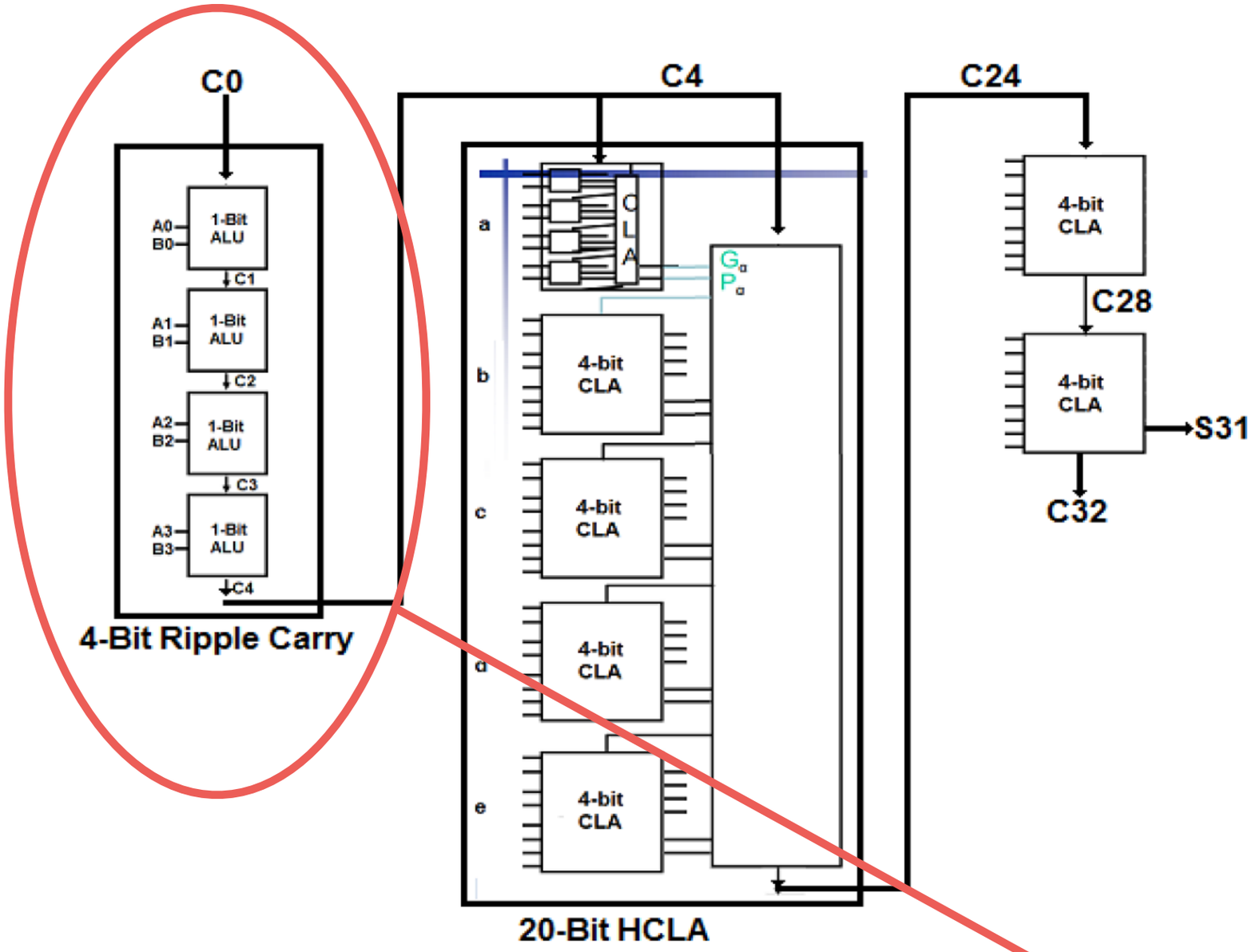
4-Bit Ripple Carry

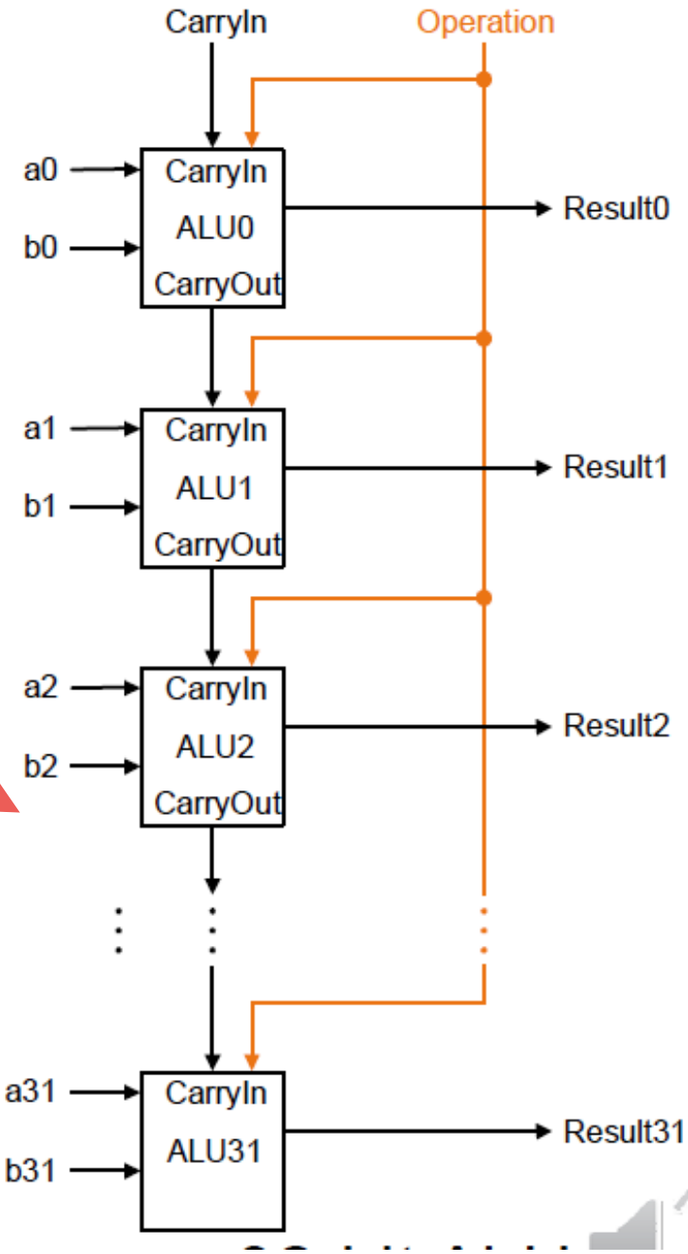


20-Bit HCLA

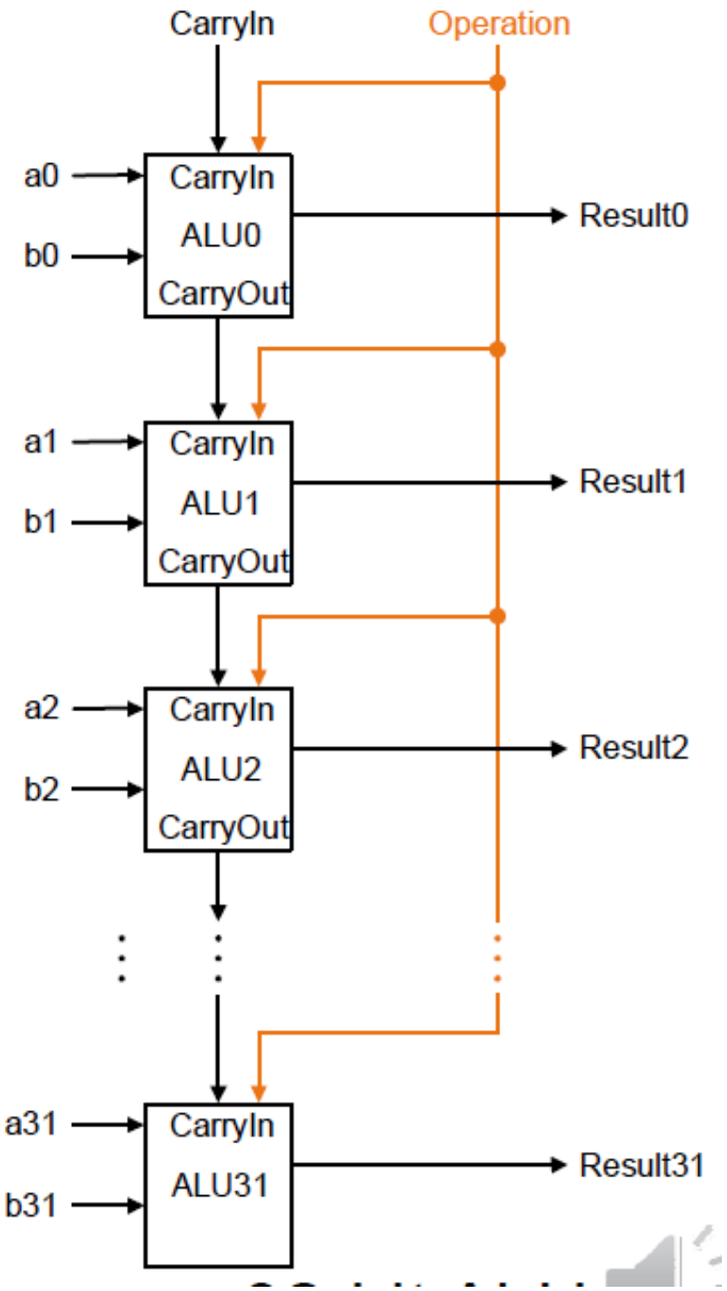


- **Steps:**
  - 1. Find the critical path;
  - 2. Understand the components of the ALU;
  - 3. Calculate time delay for different parts in order.
    - When calculating time delay for different part, decompose it as a simple ALU as we have discussed before.





Part 1:
Ripple type of ALU composed of 1-bit adder



Fan-In	Delay
1	1T
2	2T
3	3T
4	5T
5	8T
6	13T

Cn+1 = An*Bn + An*Cn + Bn*Cn
Delay_1 = 2T + 3T = 5T;
Sn = Cn^(An^Bn)
Delay_2 = 2T + 2T = 4T;

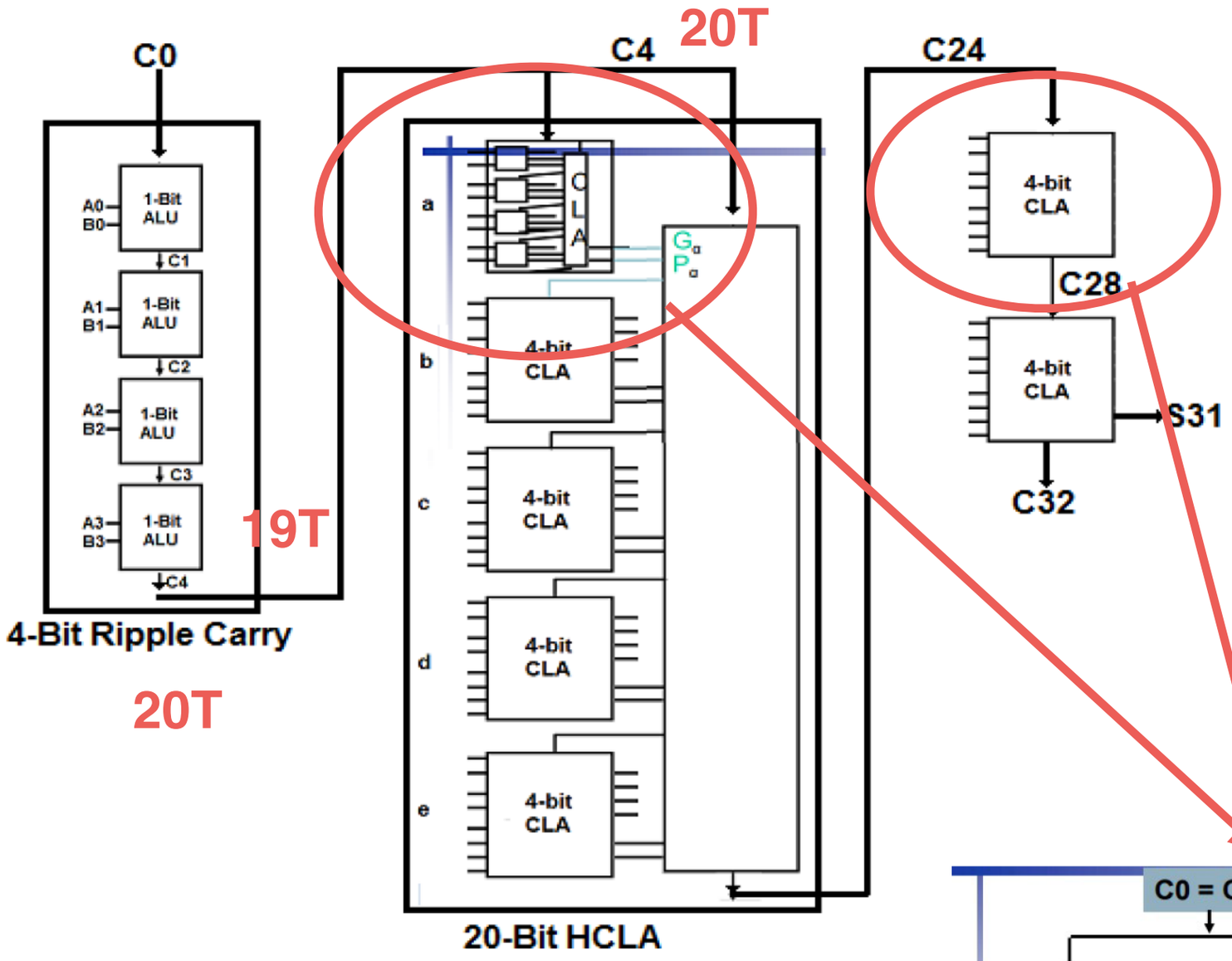
Delay(Cin) = 0;

Delay(C1) = 5T
Delay(S0) = 4T

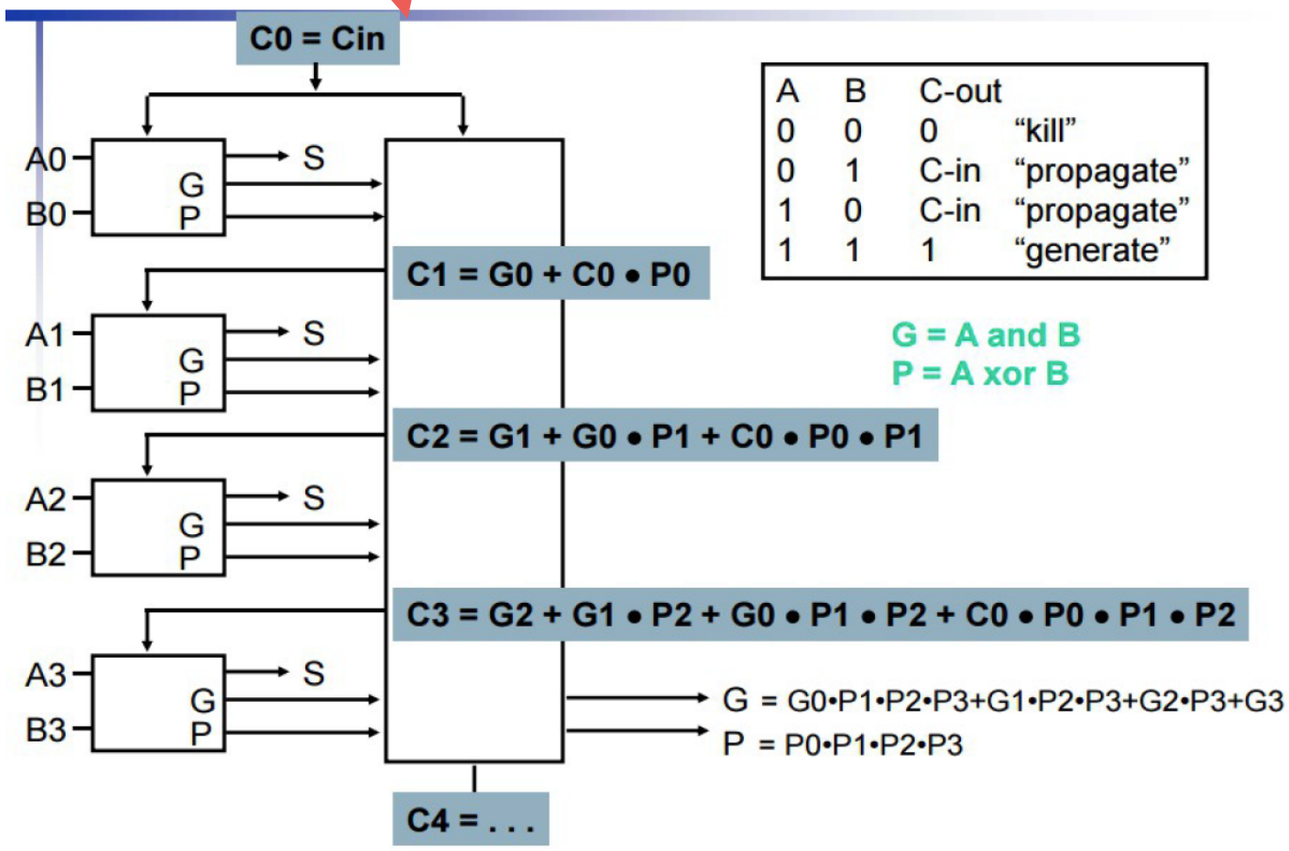
Delay(C2) = 5T + 5T = 10T
Delay(S1) = 5T + 4T = 9T

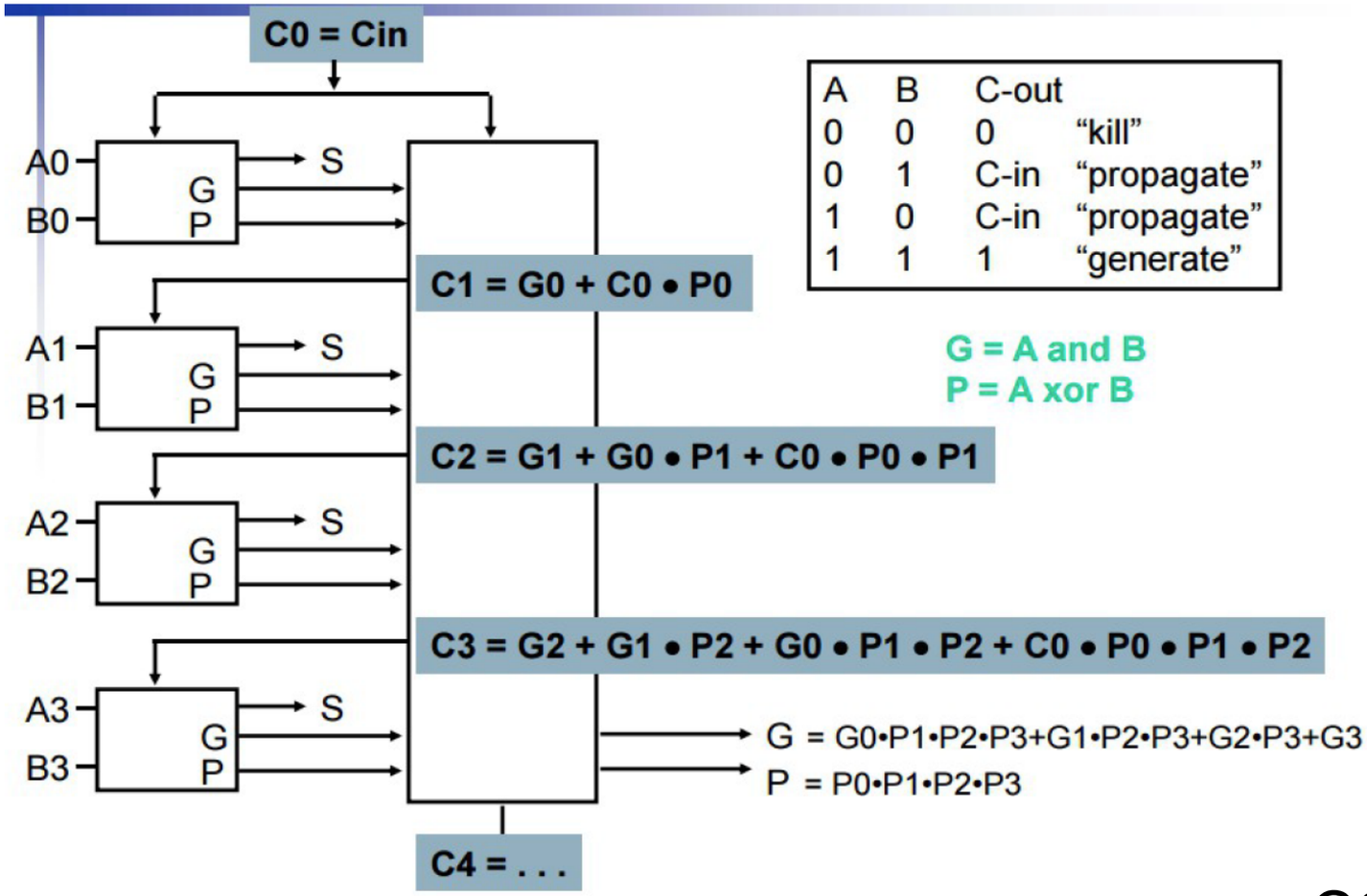
Delay(C3) = 10T + 5T = 15T
Delay(S2) = 10T + 4T = 14T

Delay(C4) = 15T + 5T = 20T
Delay(S3) = 15T + 4T = 19T



Part 2:
Carry Lookahead Adder composed of 1-bit adder





A	B	C-out	
0	0	0	“kill”
0	1	C-in	“propagate”
1	0	C-in	“propagate”
1	1	1	“generate”

G = A and B
P = A xor B

Fan-In	Delay
1	1T
2	2T
3	3T
4	5T
5	8T
6	13T

C0 = 0

G0 = A0 * B0
P0 = A0 ^ B0
C1 = G0 + C0 * P0
S0 = C0^(A0^B0)

Delay(C1) = 2T + 2T = 4T
Delay(S0) = 2T + 2T = 4T

G1 = A1 * B1
P1 = A1 ^ B1
C2 = G1 + C1 * P1
= G1 + G0 * P1
+ C0 * P0 * P1
S1 = C1^(A1^B1)
=G0^(A1^B1) +
C0*P0^(A1^B1)

Delay(C2) = 8T
Delay(S1) = 6T

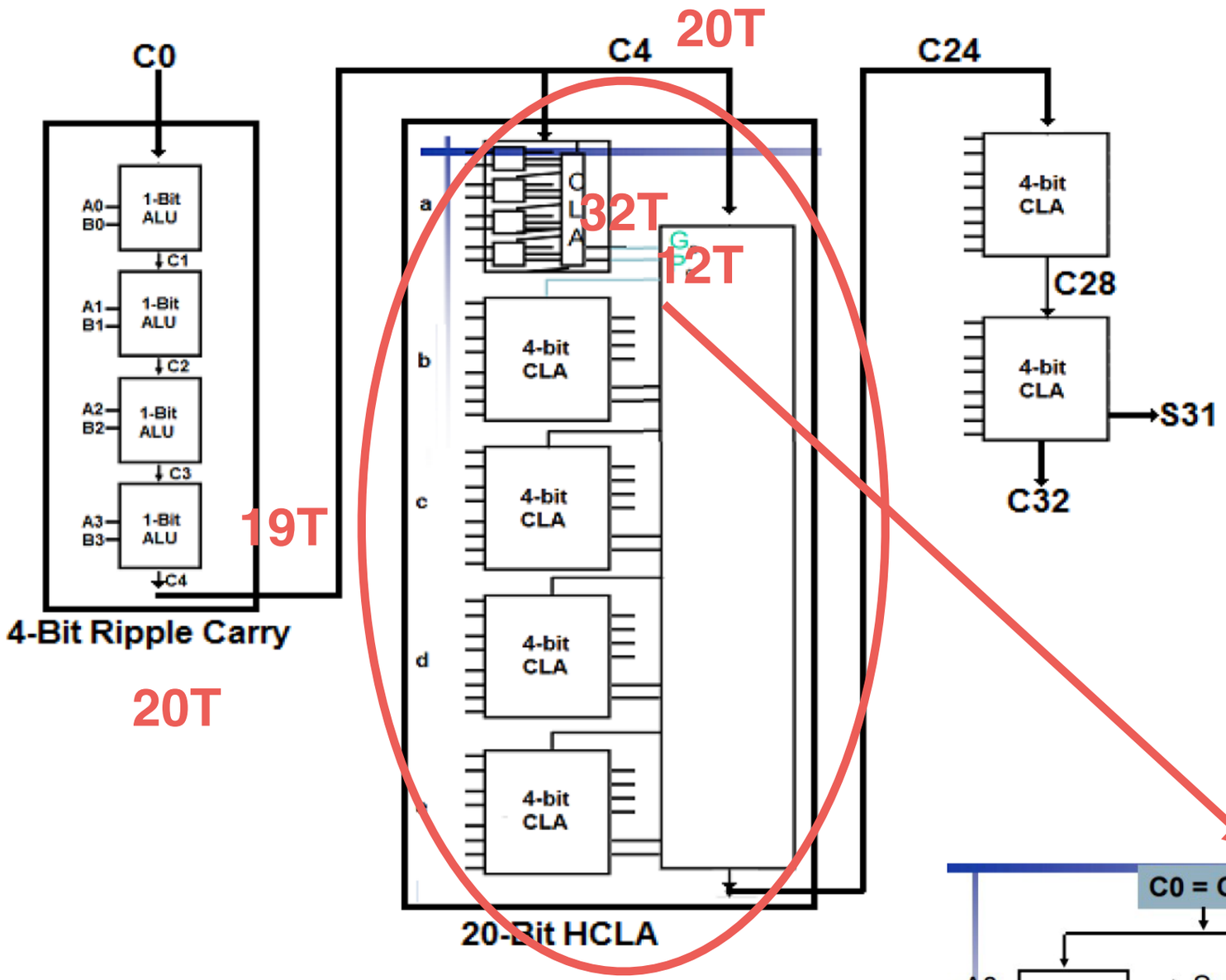
G2 = A2 * B2
P2 = A2 ^ B2
C3 = G2 + C2 * P2
S2 = C2^(A2^B2)

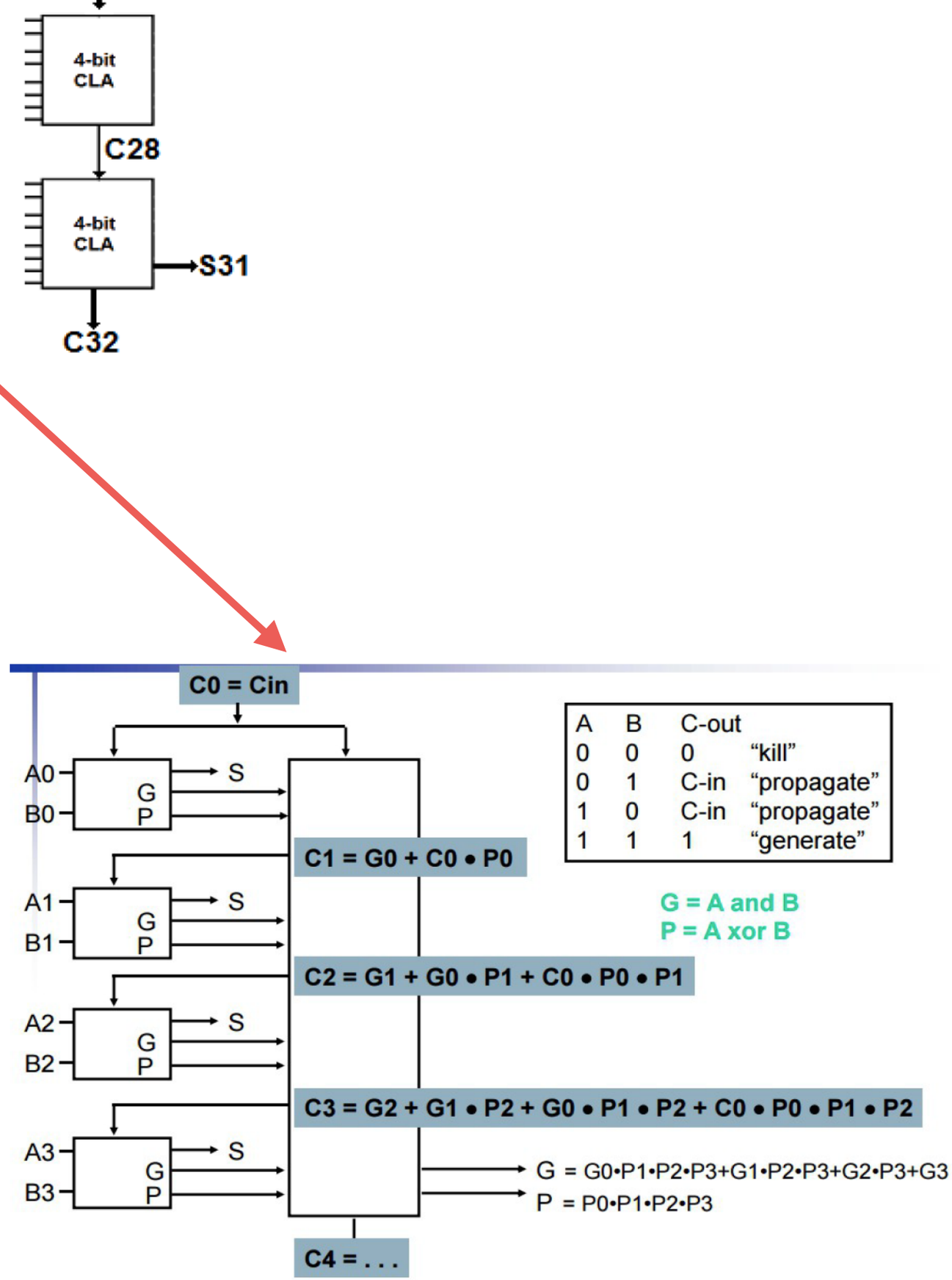
Delay(C3) = 12T
Delay(S2) = 10T

G3 = A3 * B3
P3 = A3 ^ B3
C4 = G3 + C3 * P3
S3 = C3^(A3^B3)

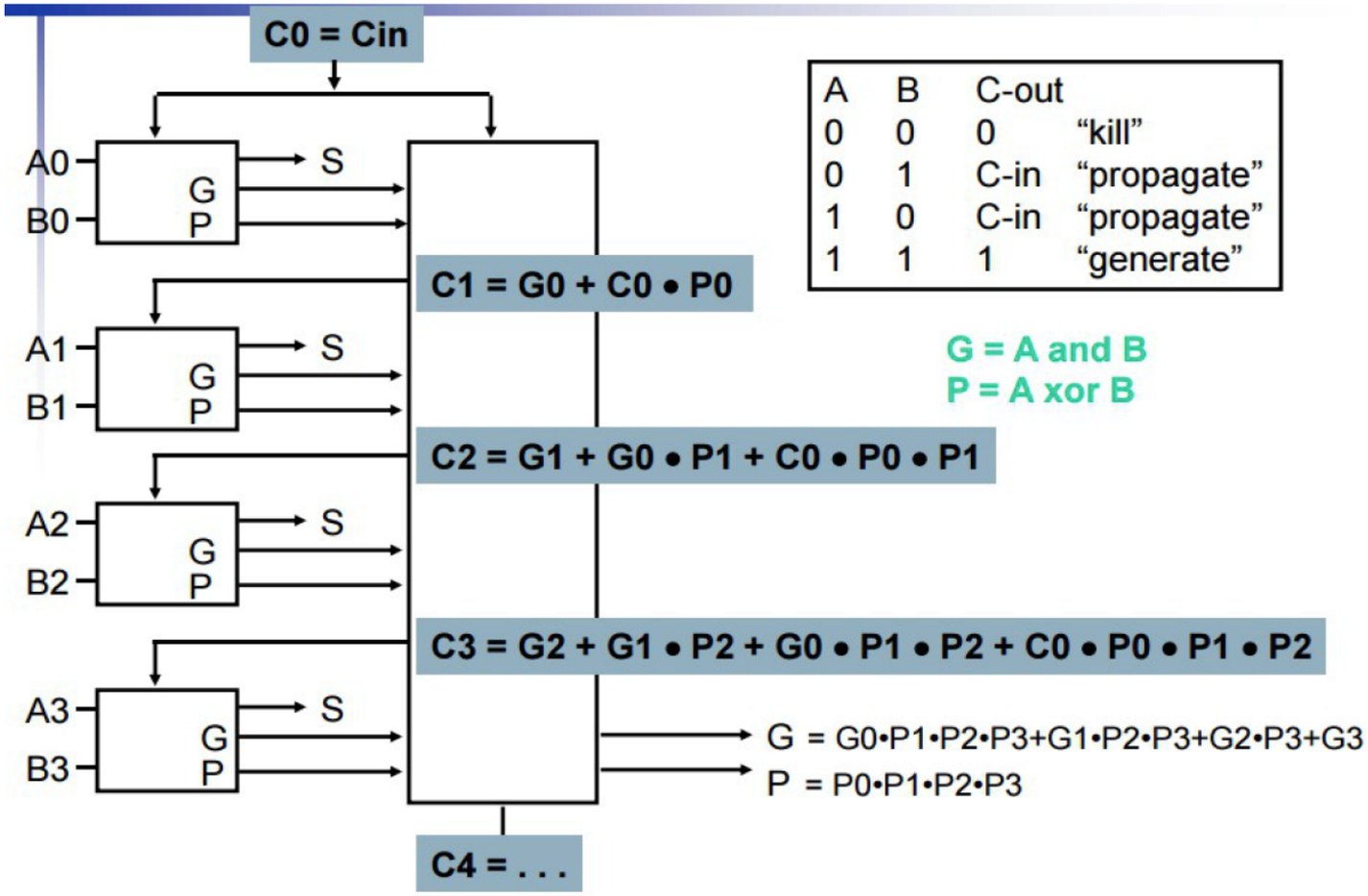
Delay(C4) = 16T
Delay(S3) = 12T

G = 12T
P = 7T





Part 2:
Carry Lookahead Adder composed of 1-bit adder



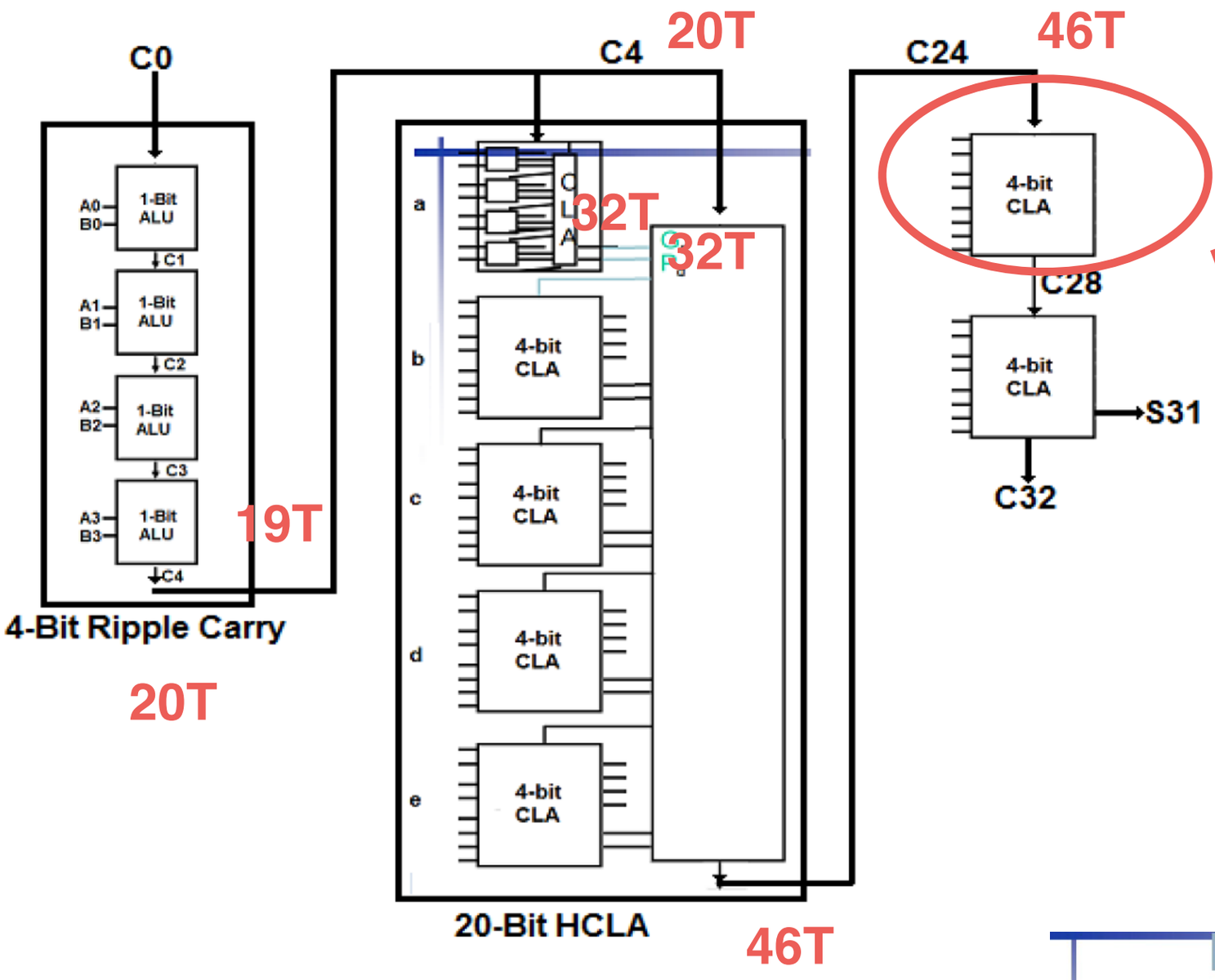
A	B	C-out	
0	0	0	“kill”
0	1	C-in	“propagate”
1	0	C-in	“propagate”
1	1	1	“generate”

G = A and B
P = A xor B

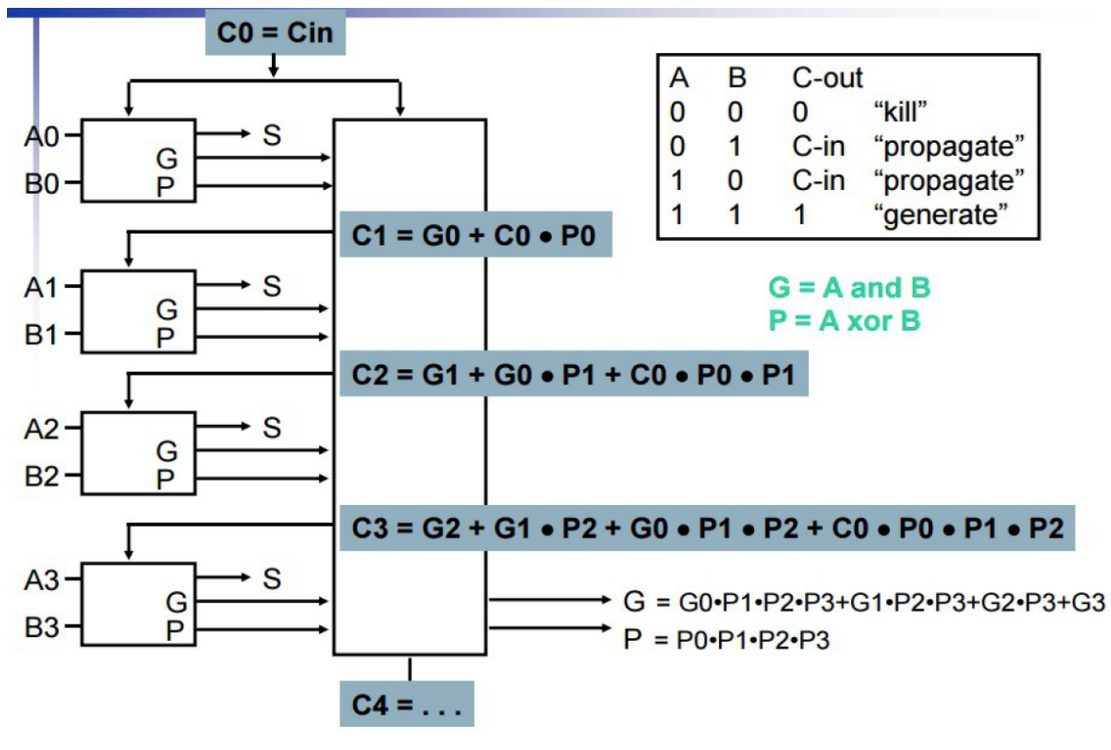
Fan-In	Delay
1	1T
2	2T
3	3T
4	5T
5	8T
6	13T

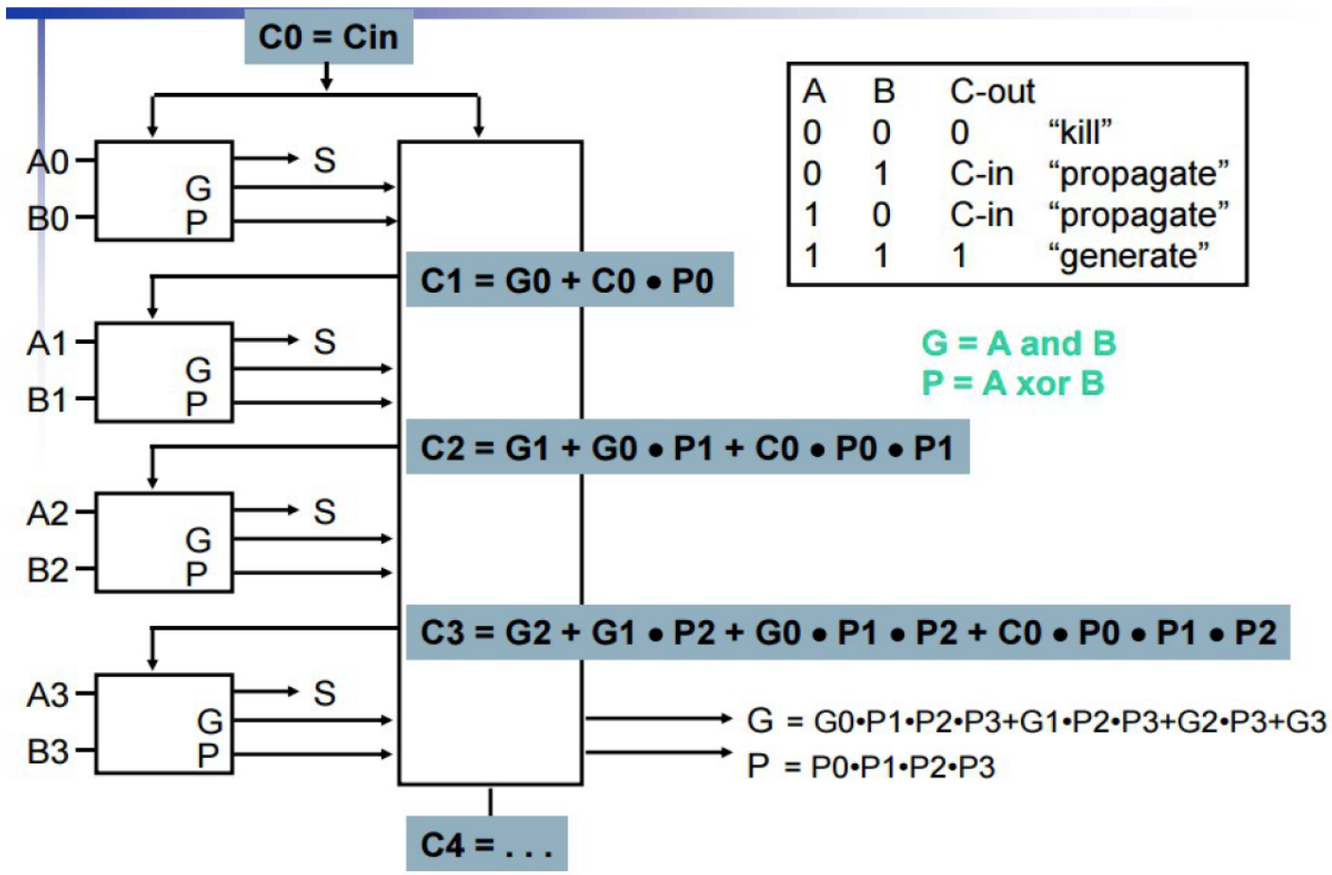
Ce = Ge + Gd*Pe +
Gc*Pd*Pe +
Gb*Pc*Pd*Pe +
Ga*Pb*Pc*Pd*Pe +
C4*Pa*Pb*Pc*Pd*Pe

Delay(Ce) = 26T



Part 2:
Carry Lookahead Adder composed of 1-bit adder



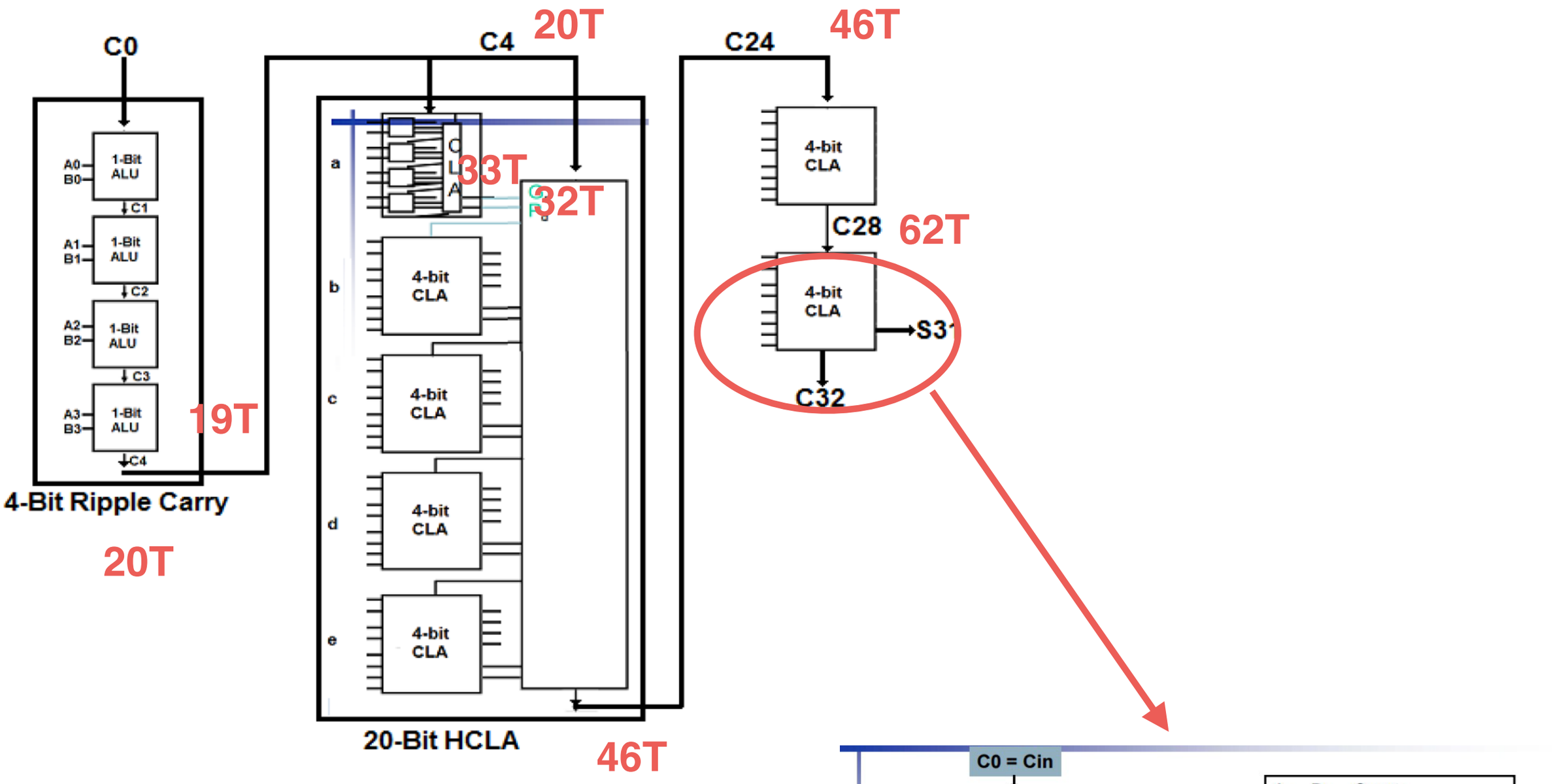


A	B	C-out	
0	0	0	“kill”
0	1	C-in	“propagate”
1	0	C-in	“propagate”
1	1	1	“generate”

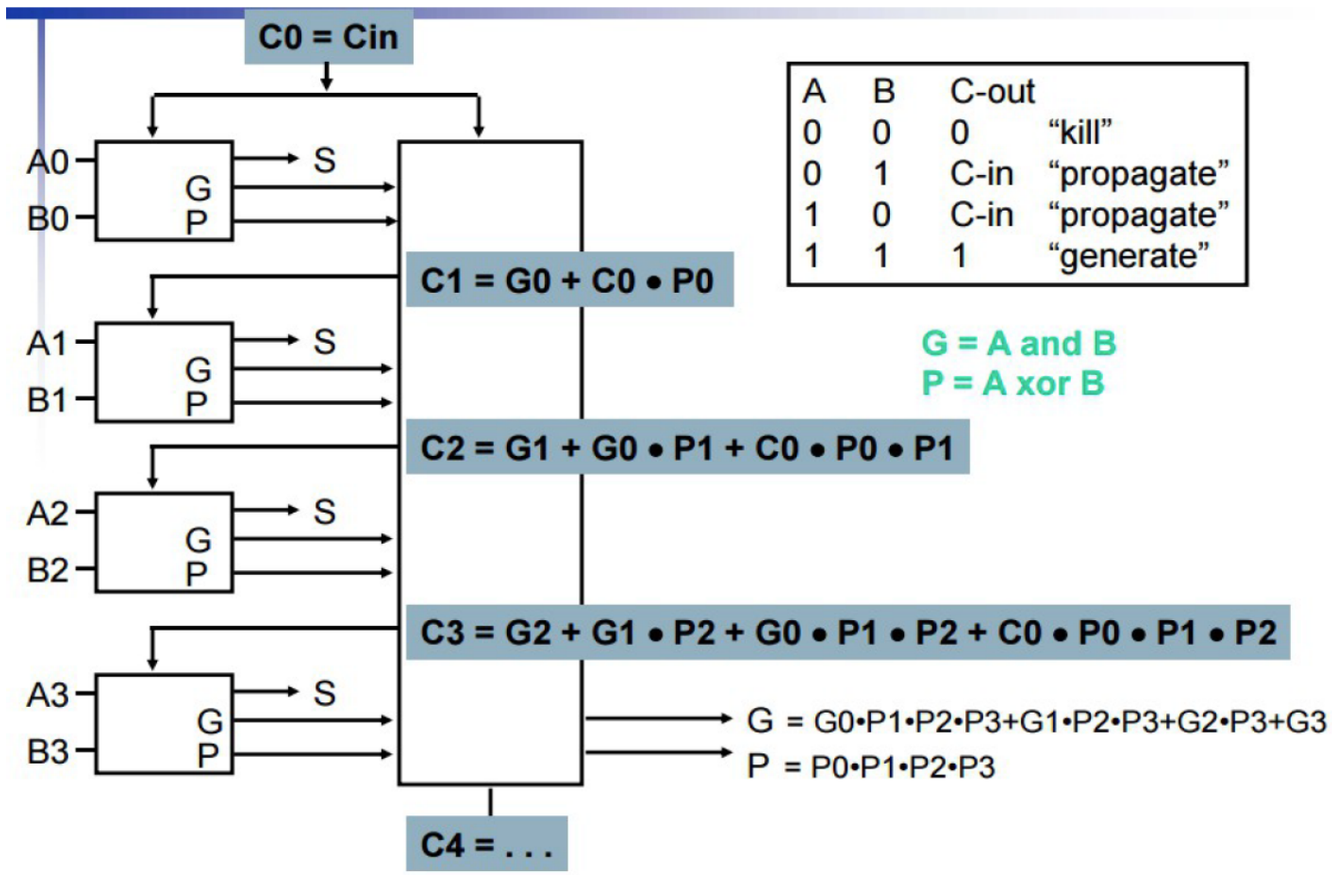
G = A and B
P = A xor B

Fan-In	Delay
1	1T
2	2T
3	3T
4	5T
5	8T
6	13T

Delay(C4) = 16T



Part 2:
Carry Lookahead Adder composed of 1-bit adder

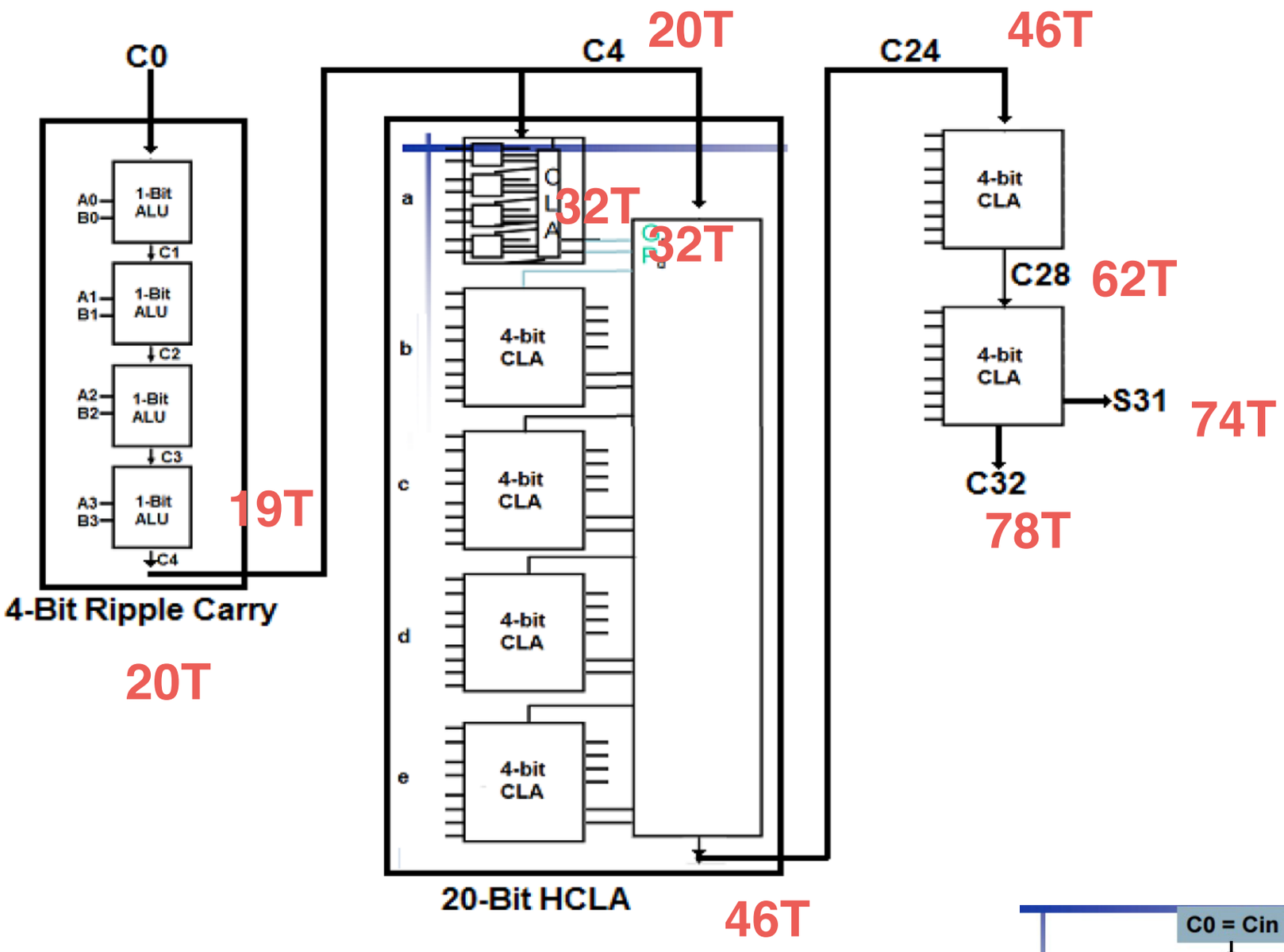


A	B	C-out	
0	0	0	“kill”
0	1	C-in	“propagate”
1	0	C-in	“propagate”
1	1	1	“generate”

G = A and B
P = A xor B

Fan-In	Delay
1	1T
2	2T
3	3T
4	5T
5	8T
6	13T

Delay(C4) = 16T
Delay(S3) = 13T



Part 2:
Carry Lookahead Adder composed of 1-bit adder