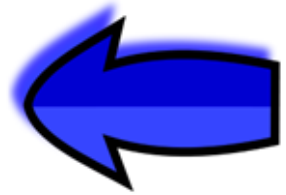
Basic Computer Architecture

Chapter 8: Computer Arithmetic (Part I)

Outline

* Addition
* Multiplication
* Division
* Floating Point Addition
* Floating Point Multiplication
* Floating Point Division



Adding Two 1 bit Numbers

* Let us add two 1 bit numbers – a and b
  * 0 + 0 = 00
  * 1 + 0 = 01
  * 0 + 1 = 01
  * 1 + 1 = 10
* The lsb of the result is known, as the sum, and the msb is known as the carry

Sum and Carry

$$\begin{array}{r} + \quad a \\ \quad\quad b \\ \hline \end{array}$$

carry | sum

a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

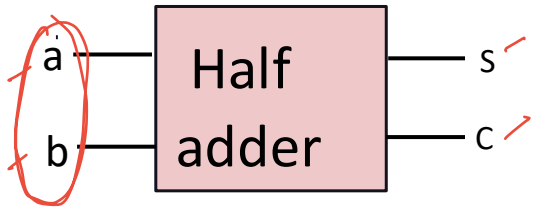
Truth Table

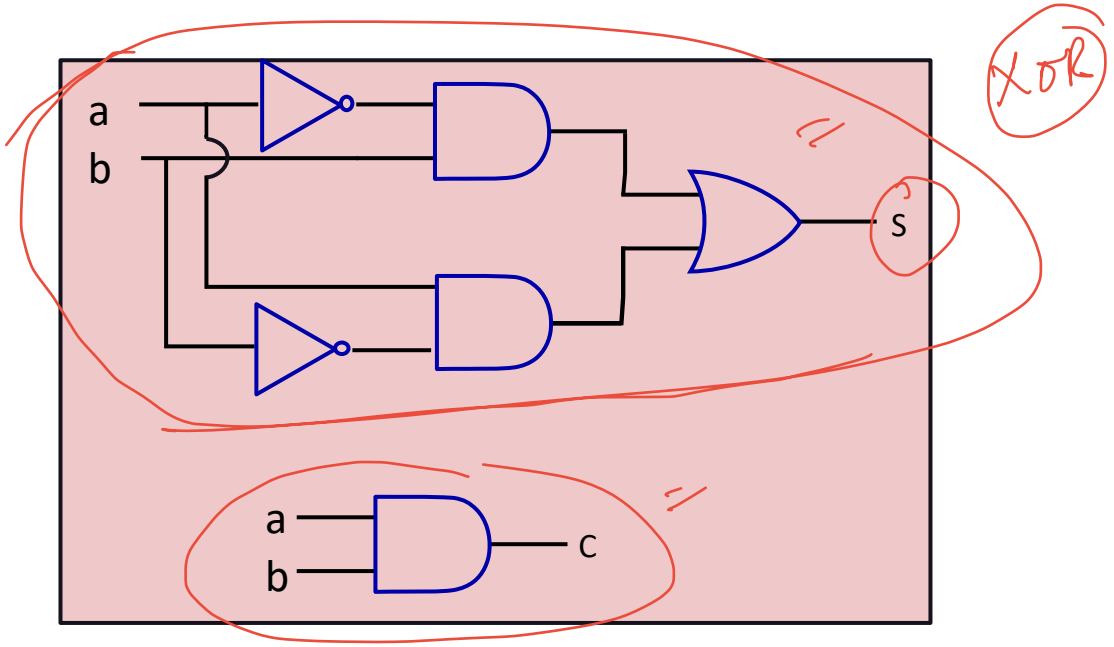
$$S = a \oplus b = \bar{a}.b + a.\bar{b}$$

$$c = a.b$$

Half Adder

* Adds two 1 bit numbers to produce a 2 bit result



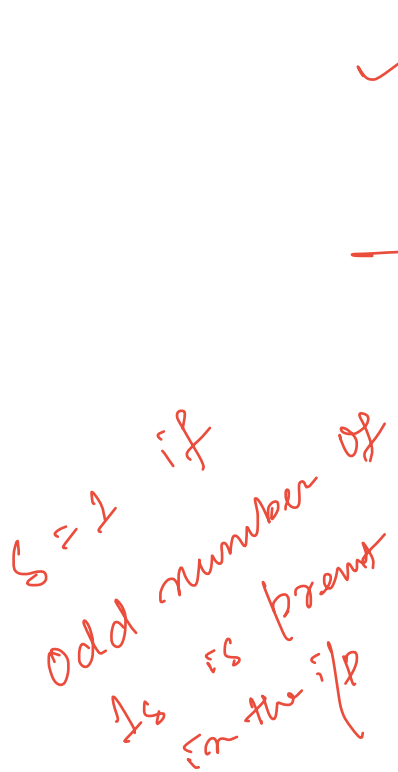


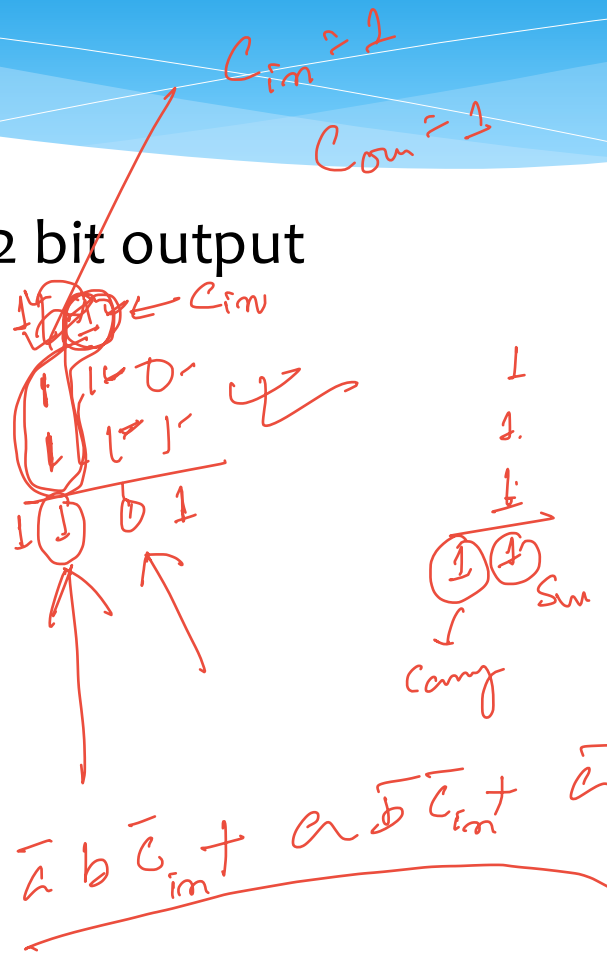
Full Adder

Add three 1 bit numbers to produce a 2 bit output

$$a + b + c_{in} = 2 * c_{out} + s$$

a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1





S=1 if odd no of 1s in a, b, c_in

Equations for the Full Adder

$$s = a \oplus b \oplus c_{in}$$

$$= \left(a.\bar{b} + \bar{a}.\mathrm{b}\right) \oplus c_{in}$$

$$= \left(a.\bar{b} + \bar{a}.\mathrm{b}\right).\overline{c_{in}} + \overline{a.\bar{b} + \bar{a}.\mathrm{b}}.c_{in}$$

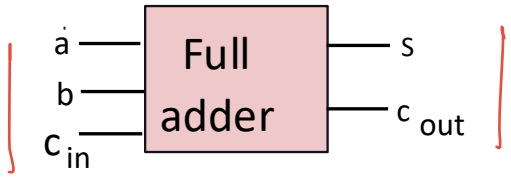
$$= a.\bar{b}.\overline{c_{in}} + \bar{a}.\mathrm{b}.\overline{c_{in}} + \overline{(a.\bar{b})}.\overline{(\bar{a}.b)}.c_{in}$$

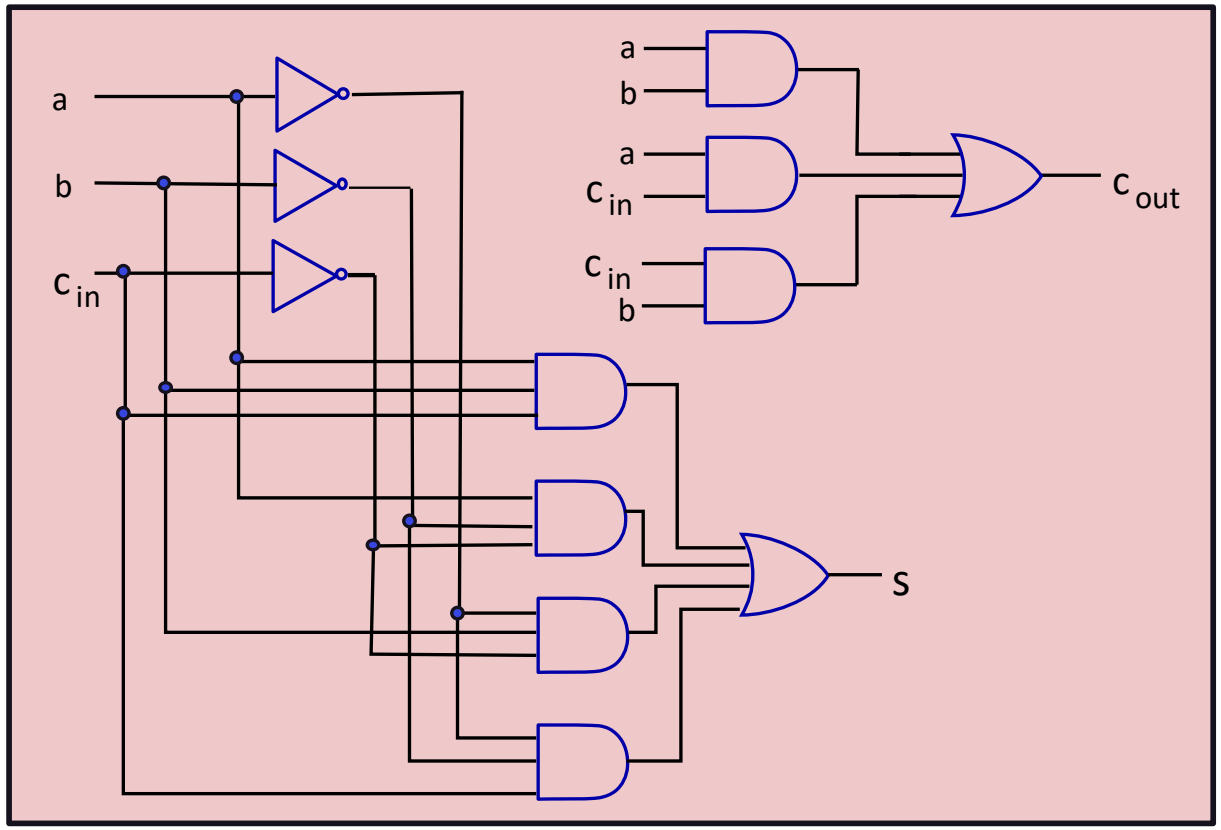
$$= a.\bar{b}.\overline{c_{in}} + \bar{a}.\mathrm{b}.\overline{c_{in}} + (\bar{a} + b).\left(a + \bar{b}\right).c_{in}$$

$$= a.\bar{b}.\overline{c_{in}} + \bar{a}.\mathrm{b}.\overline{c_{in}} + \bar{a}.\bar{b}.c_{in} + a.b.c_{in}$$

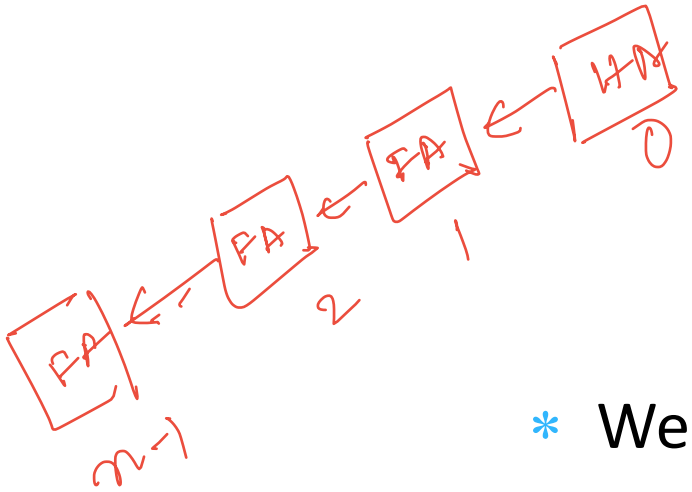
$$c_{out} = a.b + a.c_{in} + b.c_{in}$$

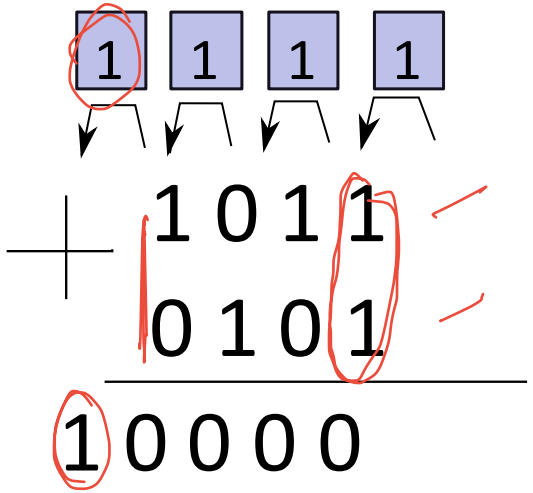
Circuit for the Full Adder





Addition of two *n* bit numbers



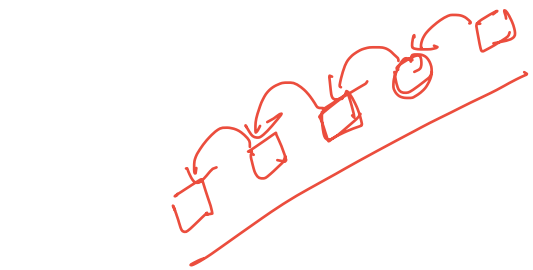


- We start from the lsb
- Add the corresponding pair of bits and the carry in
- Produce a sum bit and a carry out

Observations

* We keep adding pairs of bits, and proceed from the lsb to the msb
* If a carry is generated, we add it to the next pair of bits
* At the last step, if a carry is generated, then it becomes the msb of the result
* The carry effectively ripples through the bits

Ripple Carry Adder

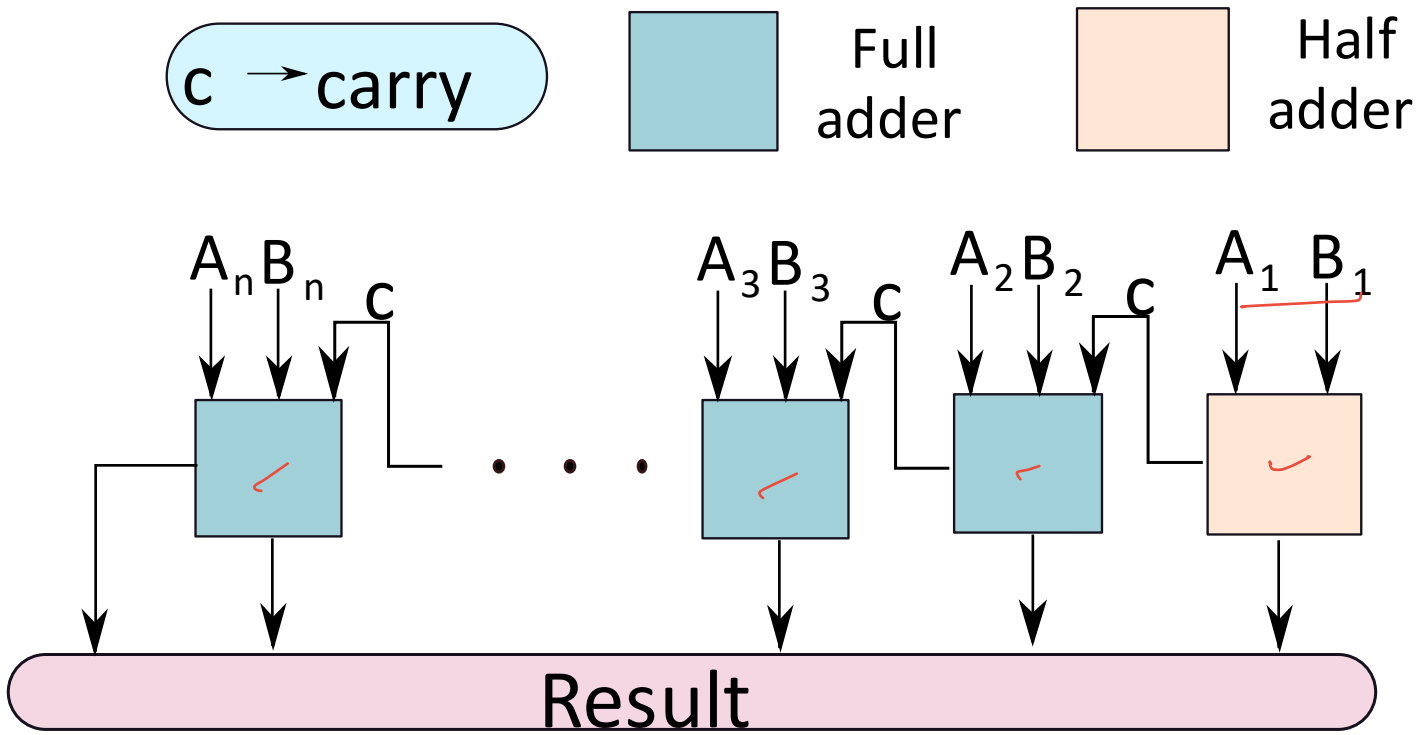


t_w

t_f

$(n-1)\,t_f + t_h$

$= O(n)$



Operation of the Ripple Carry Adder

* Problem : Add A + B
* Number the bits : A_1 to A_n and B_1 to B_n
  * lsb → A_1 and B_1
  * msb → A_n and B_n
* Use a half adder to add A_1 and B_1
* Send the carry(c) to a full adder that adds : $A_2 + B_2 + c$
* Proceed in a similar manner till the msb

How long does the Ripple Carry Adder take ?

* Time :
  * Time of half adder : t_h
  * Time of full adder : t_f
  * Total Time : $t_h + (n-1)t_f$

Asymptotic Time Complexity

* Most of the time, we are primarily interested in the order of the function
* For example : we are only interested in the n^2 term in $(2n^2 + 3n + 4)$
* We do not care about the constants, and terms with smaller exponents
  * 3n and 4
* We can thus say that :
  * $2n^2 + 3n + 4$ is order of (n^2)

The O notation

* Formally :
  * We say that: f(n) = O(g(n))
  * if, $|f(n)| \leq c|g(n)|$, for all $n > n_0$. Here *c* is a positive constant.
* In simple terms:
  * Beyond a certain *n* , g(n) is greater-than-equal to a certain constant times f(n)
    * *For example, beyond 15, $(n^2 + 10n + 16) \leq 2n^2$*

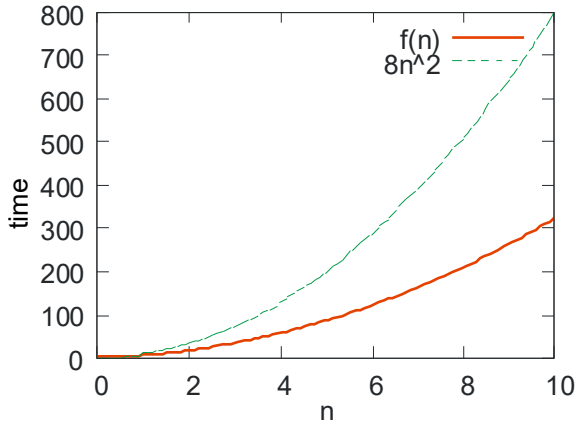
Example of the big O Notation

$f(n) = 3n^2 + 2n + 3$. *Find its asymptotic time complexity.*

Answer:

$$f(n) = 3n^2 + 2n + 3$$
$$\leq 3n^2 + 2n^2 + 3n^2 \ (n > 1)$$
$$\leq 8(n^2)$$

Hence, $f(n) = O(n^2)$.



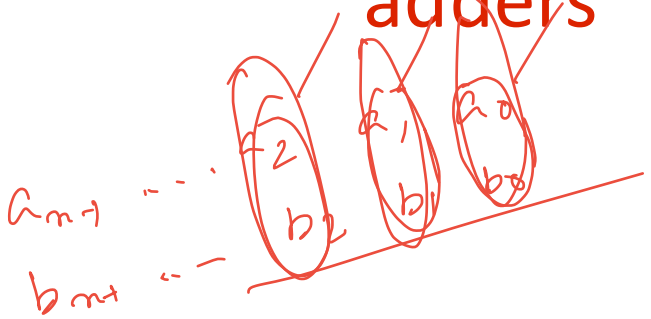
$8n^2$ *is a strict upper bound on* $f(n)$ *as shown in the figure.*

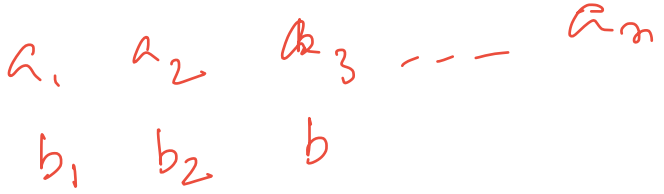
Big O Notation - II

Example:
$f(n) = 0.00001n^{100} + 10000n^{99} + 234344$. Find its asymptotic time complexity.

Answer: $f(n) = O(n^{100})$

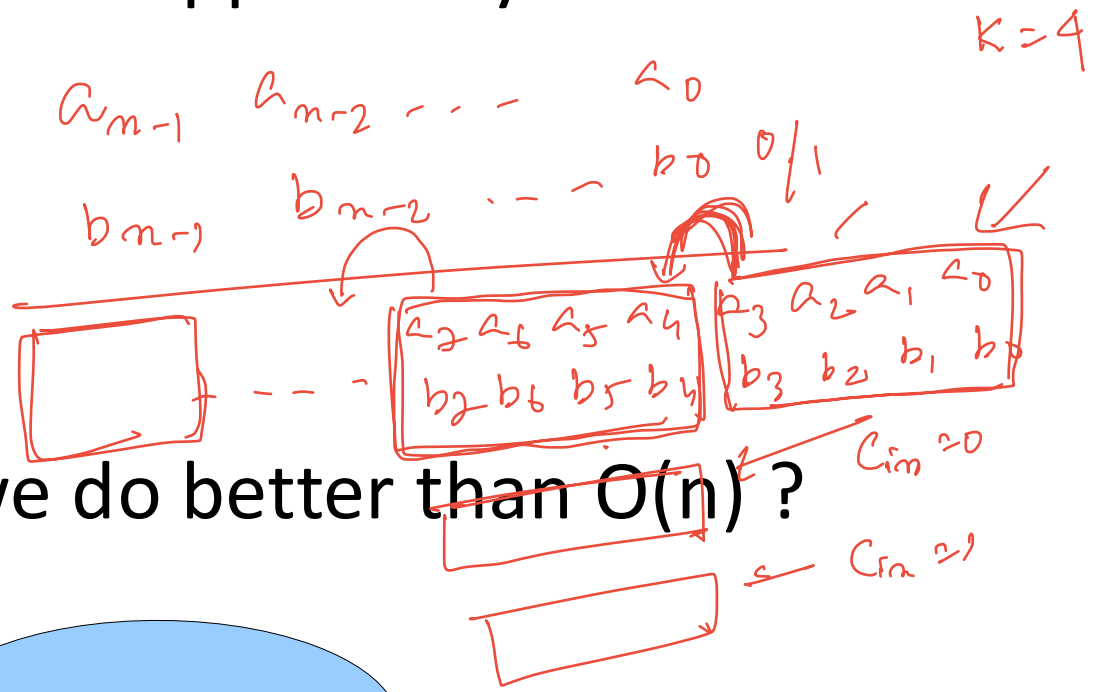
* We shall use the asymptotic time complexity metric (big O notation) to characterize the time taken by different adders





Ripple Carry Adders and Beyond

* Time complexity of a ripple carry adder :
  * O(n)

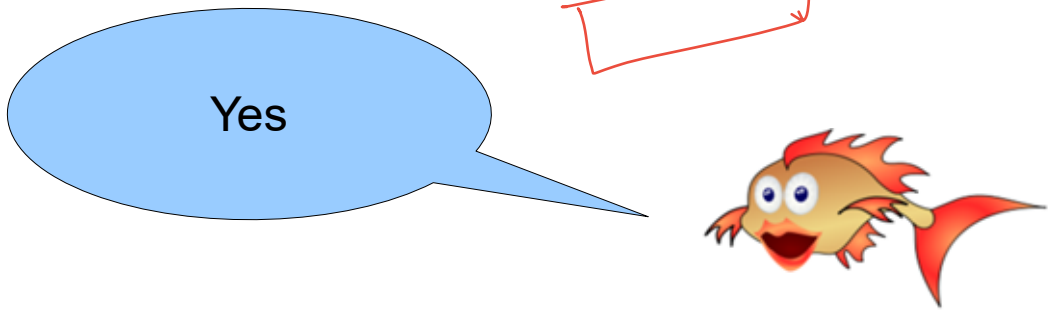


*  Can we do better than O(n) ?

Step 1
O(K)

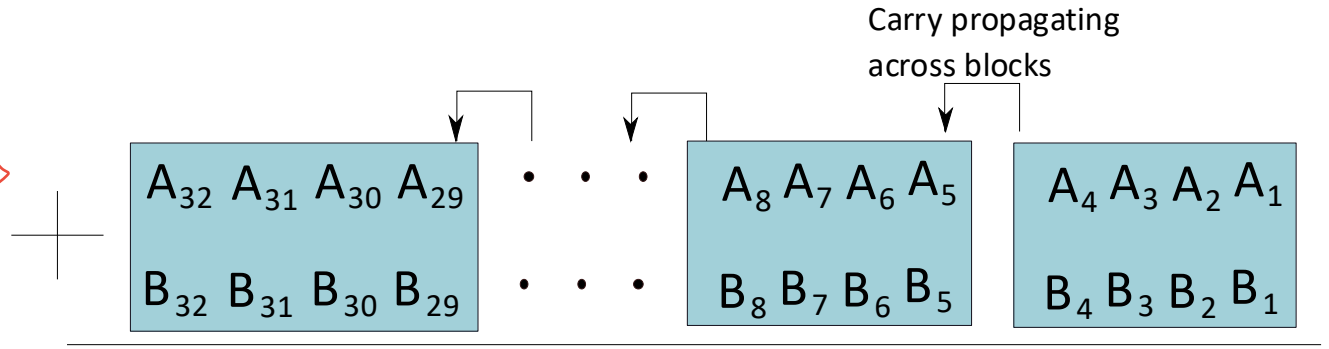
Step 2
$O\left(\frac{n}{K}\right)$

$O(K)+O\left(\frac{n}{K}\right)$



Carry Select Adder O(√n) time

* Group bits into blocks of size (k)
* If we are adding two 32 bit numbers A and B, and k = 4, then the blocks are :



* Produce the result of each block with a small ripple carry adder

Carry Select Adder - II

- In this case, the carry propagates across blocks
- Time complexity is O(n)

Idea :

- Add the numbers in each block in parallel
- Stage I : For each block, produce two results
  - Assuming an input carry of 0
  - Assuming an input carry of 1

Carry Select Adder – Stage II

- For each block we have two results available
- Result → (k sum bits), and 1 carry out bit
- Stage II
  - Start at the least significant block
    - The input carry is 0
    - Choose the appropriate result from stage I
  - We now know the input carry for the second block
    - Choose the appropriate result
    - Result contains the input carry for the third block

Carry Select Adder – Stage II

* Given the result of the second block
  * Compute the carry in for the third block
  * Choose the appropriate result
* Proceed till the last block
* At the last block (most significant positions)
  * Choose the correct result
  * The carry out value, is equal to the carry out of the entire computation.

How much time did we take ?

* Our block size is k
  * Stage I takes k units of time
* There are n/k blocks
  * Stage II takes (n/k) units of time
* Total time : (k + n/k)

$O(\sqrt{n})$

$$\frac{\partial\,(k + n/k)}{\partial k} = 0$$

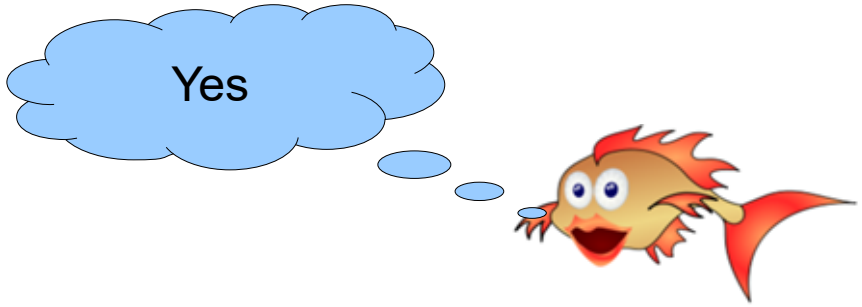
$$\Rightarrow 1 - \frac{n}{k^2} = 0$$

$$\Rightarrow k = \sqrt{n}$$

Time Complexity of the Carry Select Adder

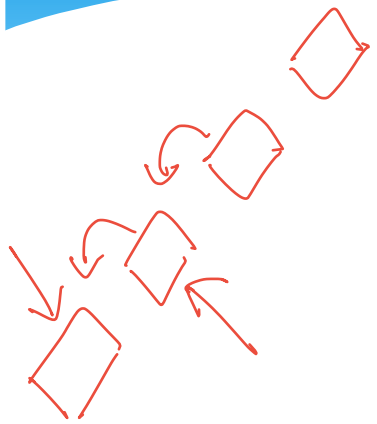
* $T = O(\sqrt{n} + \sqrt{n}) = O(\sqrt{n})$
* Thus, we have a $\sqrt{n}$ time adder

Can we do better ?



$O(\log n) + O(\log n)$

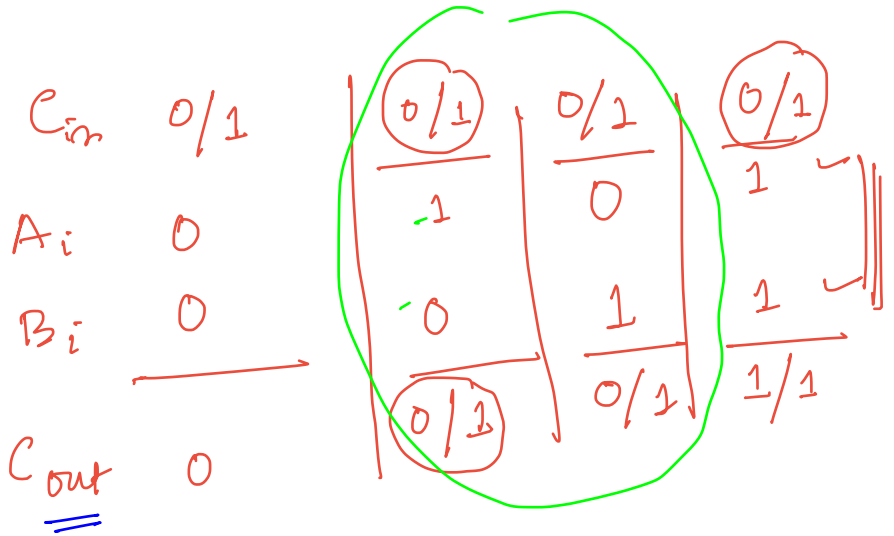
Carry Lookahead Adder (O(log n))



* The main problem in addition is the carry
* If we have a mechanism to compute the carry quickly, we are done
* Let us thus focus on computing the carry without actually performing an addition

Generate Fun → new carry g_i

Propagate Fnr. → carry to be propagated or not



$$g_i = A_i \cdot B_i = 1$$

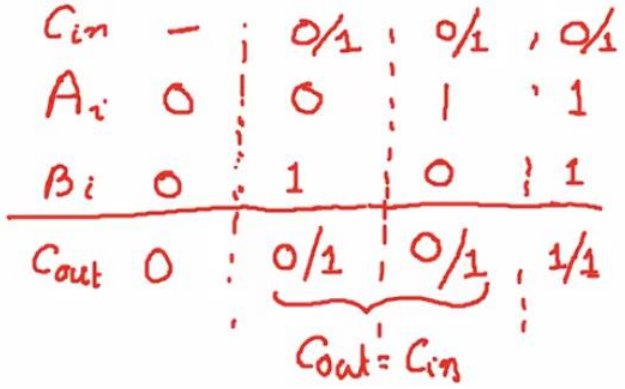
$$p_i = A_i \oplus B_i$$

$$C_{out}^{i} = g_i + p_i C_{in}^{i}$$

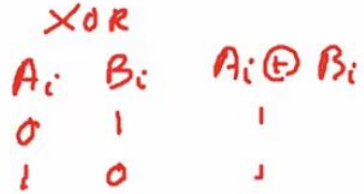
Generate and Propagate Functions

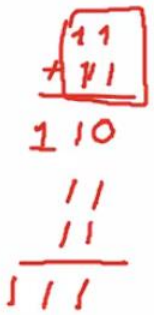
A
+B

* Let us consider two corresponding bits of A and B
  * A_i and B_i
* Generate function : A new carry is generated (C_{out} = 1)
* Propagate function : $C_{out} = C_{in}$
* Generate and Propagate Functions are :



$$g_i = A_i . B_i$$
$$p_i = A_i \oplus B_i$$





Mc Graw Hill Education

Using the G and P Functions

* If we have the generate and propagate values for a bit pair, we can determine the carry out

$$C_{out} = g_i + p_i.C_{in}$$

$C_{out}^{i} = g_i$

Example

Example:

Let A_i = 0, B_i = 1. Let the input carry be C_{in}. Compute g_i, p_i, and C_{out}.

Answer:

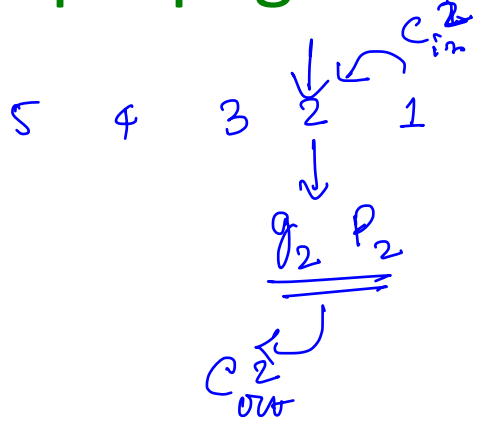
$$g_i = A_i . B_i = 0.1 = 0$$

$$p_i = A_i \oplus B_i = 0 \oplus 1 = 1$$

$$C_{out} = g_i + p_i . C_{in} = C_{in}$$

G and P for Multi-bit Systems

* C_{out}^{i} → output carry for i^{th} bit pair
* C_{in}^{i} → input carry for i^{th} bit pair
* g_i → generate value for i^{th} bit pair
* p_i → propagate value for i^{th} bit pair

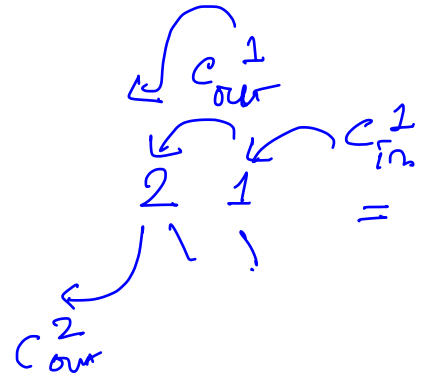


G and P for Multibit Systems - II

$$C_{out}^{1} = g_1 + p_1.C_{in}^{1}$$

$$C_{out}^{2} = g_2 + p_2.C_{out}^{1}$$
$$= g_2 + p_2.\left(g_1 + p_1.C_{in}^{1}\right)$$
$$= (g_2 + p_2.g_1) + p_2.p_1.C_{in}^{1}$$

$$C_{out}^{3} = g_3 + p_3.C_{out}^{2}$$
$$= g_3 + p_3.\left((g_2 + p_2.g_1) + p_2.p_1.C_{in}^{1}\right)$$
$$= (g_3 + p_3.g_2 + p_3.p_2.g_1) + p_3.p_2.p_1.C_{in}^{1}$$



G and P for multibit Systems - III

$$C_{out}^4 = g_4 + p_4.C_{out}^3$$
$$= g_4 + p_4.\left((g_3 + p_3.g_2 + p_3.p_2.g_1) + p_3.p_2.p_1.C_{in}^1\right)$$
$$= (g_4 + p_4.g_3 + p_4.p_3.g_2 + p_4.p_3.p_2.g_1) + p_4.p_3.p_2.p_1.C_{in}^1$$

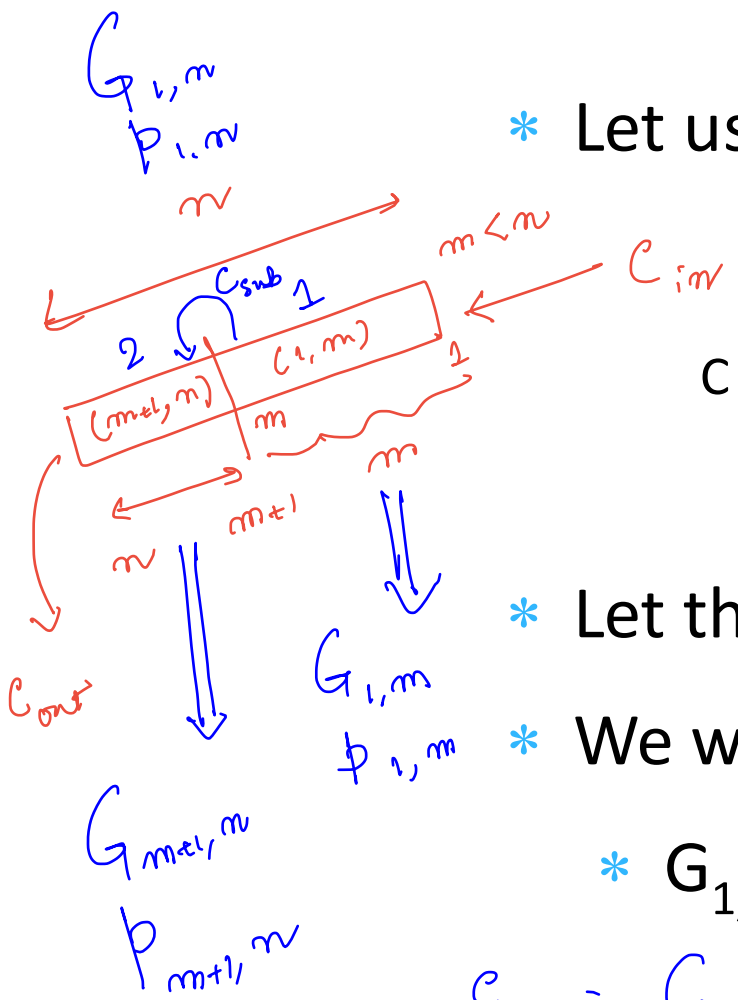
Patterns

1 bit	$C_{out}^{1} = \underbrace{g_1}_{G_1} + \underbrace{p_1}_{P_1} . C_{in}^{1}$
2 bit	$C_{out}^{2} = \underbrace{g_2 + p_2 . g_1}_{G_2} + \underbrace{p_2 . p_1}_{P_2} . C_{in}^{1}$
3 bit	$C_{out}^{3} = \underbrace{g_3 + p_3 . g_2 + p_3 . p_2 . g_1}_{G_3} + \underbrace{p_3 . p_2 . p_1}_{P_3} . C_{in}^{1}$
4 bit	$C_{out}^{4} = \underbrace{g_4 + p_4 . g_3 + p_4 . p_3 . g_2 + p_4 . p_3 . p_2 . g_1}_{G_4} + \underbrace{p_4 . p_3 . p_2 . p_1}_{P_4} . C_{in}^{1}$
n bit	$C_{out}^{n} = G_n + P_n . C_{in}^{1}$

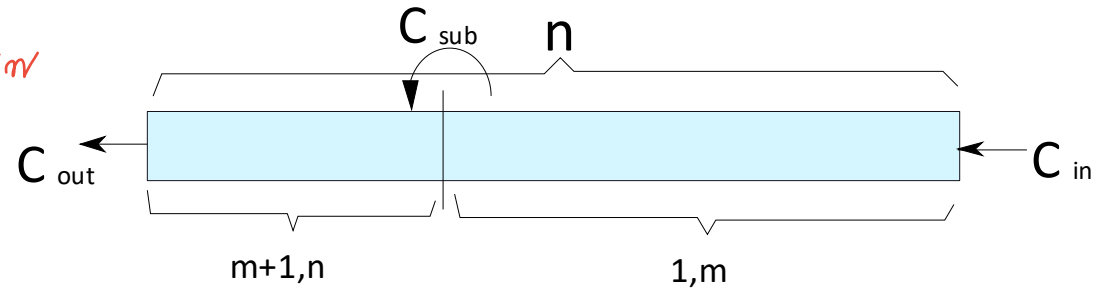
$(g_n + p_n g_{n-1} + p_n p_{n-1} g_{n-2} + \cdots)$

$P_n = p_n p_{n-1} p_{n-2} \cdots p_1$

Computing G and P Quickly



* Let us divide a block of *n* bits into two parts



* Let the carry out and carry in be : C_{out} and C_{in}
* We want to find the relationship between
  * $G_{1,n}$, $P_{1,n}$ and ($G_{m+1,n}$, $G_{1,m}$, $P_{m+1,n}$, $P_{1,m}$)

$$C_{out} = G_{m+1,n} + P_{m+1,n} C_{sub}$$
$$= G_{m+1,n} + P_{m+1,n} (G_{1,m} + P_{1,m} C_{in})$$
$$= G$$

Computing G and P Quickly - II

$$C_{out} = G_{m+1,n} + P_{m+1,n}.C_{sub}$$
$$= G_{m+1,n}$$
$$+ P_{m+1,n}.(G_{1,m} + P_{1,m}.C_{in})$$
$$= \underbrace{G_{m+1,n} + P_{m+1,n}.G_{1,m}}_{G_{1,n}} + \underbrace{P_{m+1,n}.P_{1,m}}_{P_{1,n}}.C_{in}$$

$$C_{out} = G_{1,n} + P_{1,n}.C_{in}$$

$$G_{1,n} = G_{m+1,n} + P_{m+1,n}.G_{1,m}$$

$$P_{1,n} = P_{m+1,n}.P_{1,m}$$

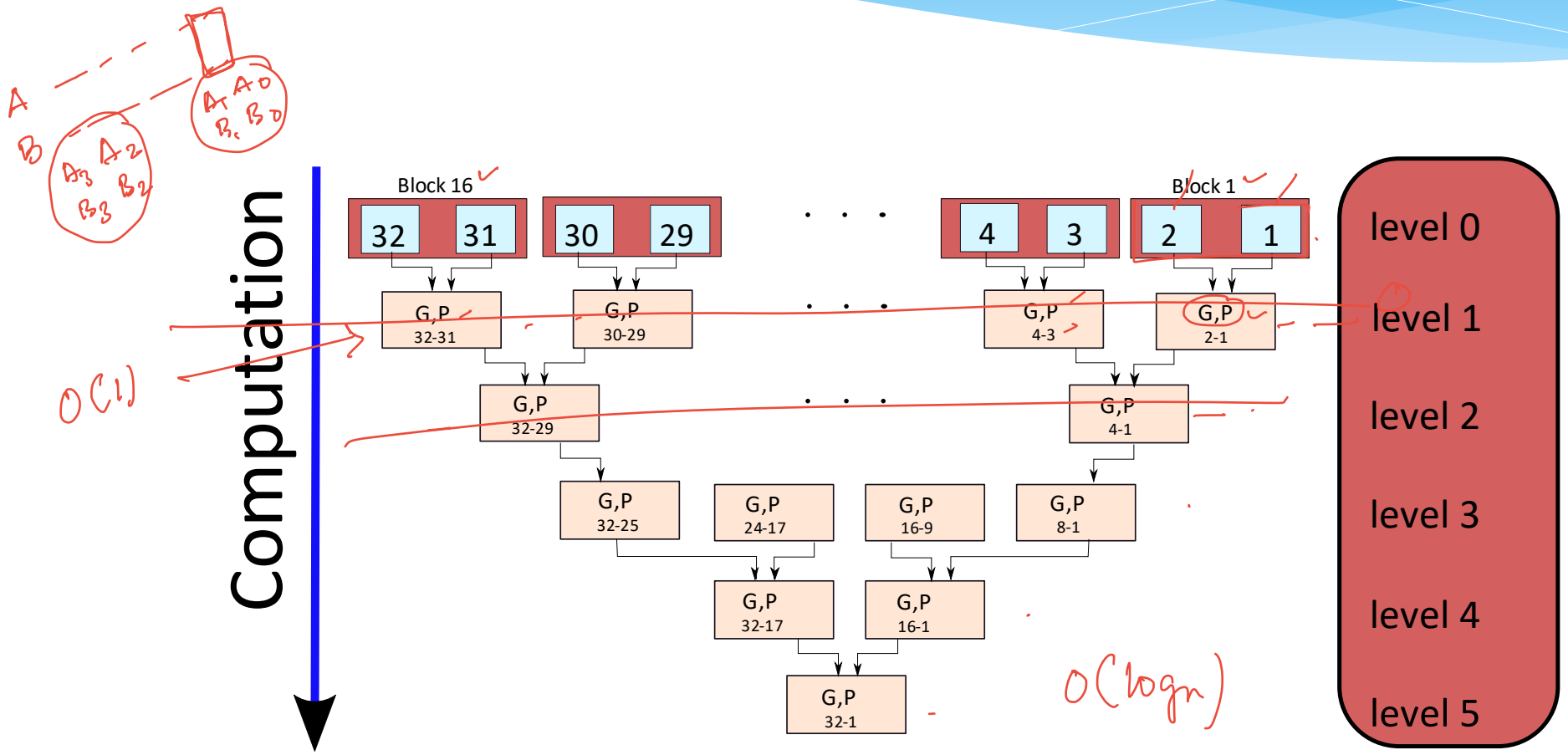
Insight into Computing G and P quickly

* Insight :
  * We can compute G and P for a large block
    * By first computing G and P for smaller sub-blocks
    * And, then combining the solutions to find the value of G and P for the larger block
  * Fast algorithm to compute G and P
    * Use divide-and-conquer
    * Compute G and P functions in O (log (n)) time

Carry Lookahead Adder – Stage I

* Compute G and P functions for all the blocks
* Combine the solutions to find G and P functions for sets of 2 blocks
* Combine the solutions fo find G and P functions for sets of 4 blocks
* ….
* ….
* Find the G and P functions for a block of size : 32 bits

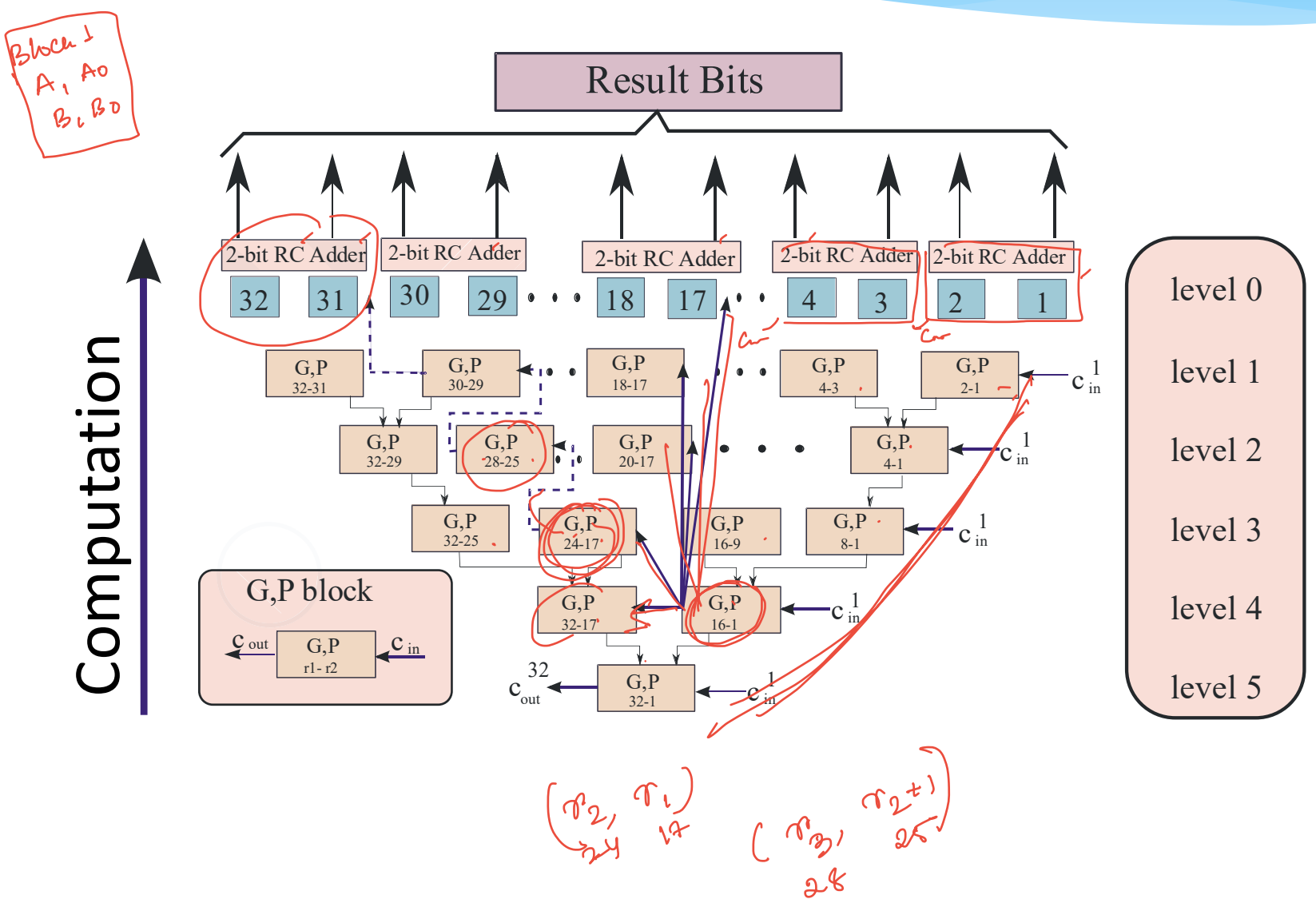
Carry Lookahead Adder – Stage I



CLA Adder – Stage I

* Compute G, P for increasing sizes of blocks in a tree like fashion
* Time taken :
  * Total : log(n) levels
  * Time per level : O(1)
  * Total Time : O(log(n))

CLA Adder – Stage II



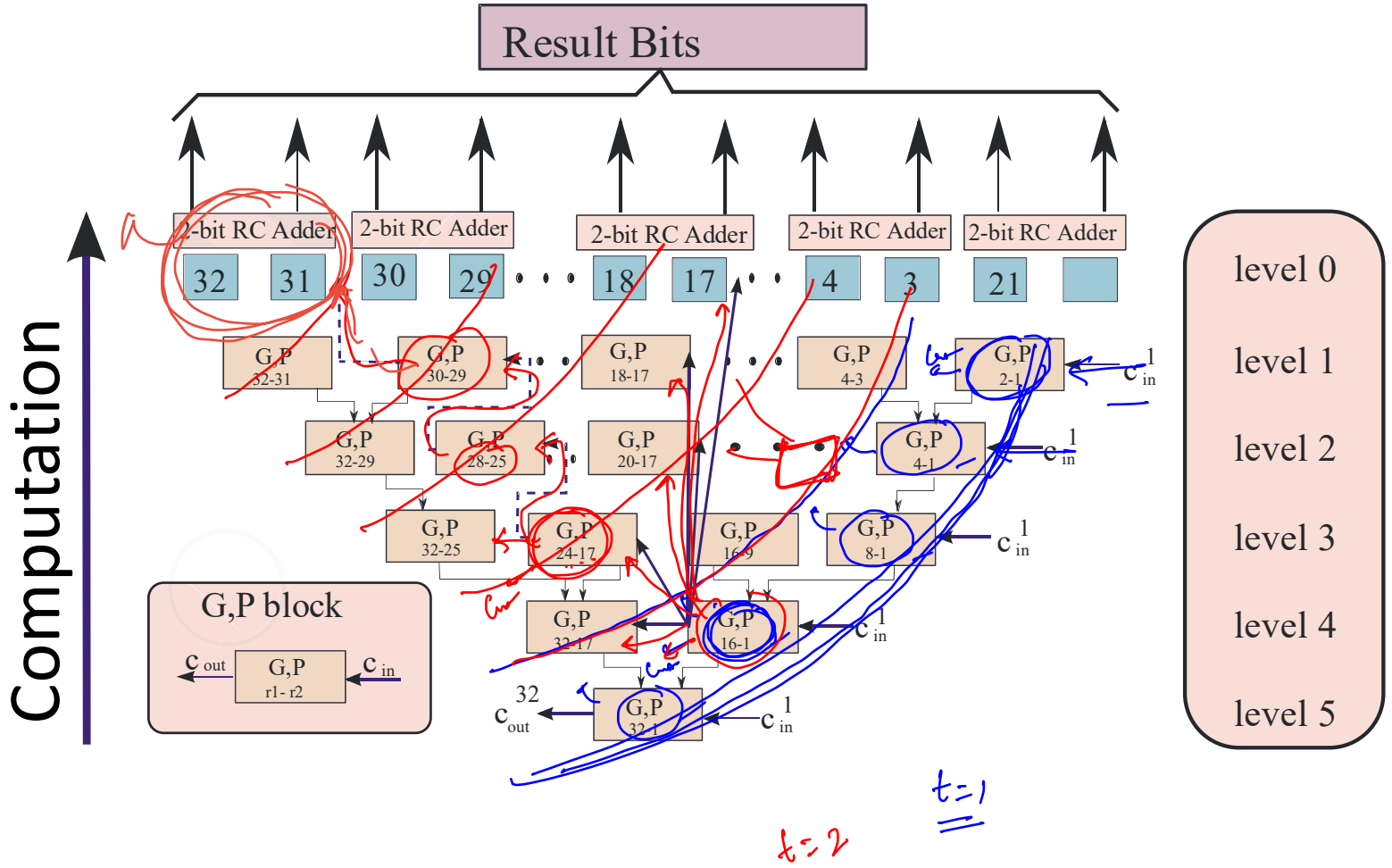
Connection of the G,P Blocks

* Each G,P block represents a range of bits (r2, r1) (r2 > r1)
  * The (r2, r1) G,P block is connected to all the blocks of the form (r3, r2+1)
  * The carry out of one block is an input to all the blocks that it is connected with
* Each block is connected to another block at the same level, and to blocks at lower levels

Operation of CLA – Stage II

- We start at the leftmost blocks in each level
  - We feed an input carry value of C_{in}^{1}
  - Each such block computes the output carry, and sends it to the all the blocks that it is connected to
- Each connected block
  - Computes the output carry
  - Sends it to all the blocks that it is connected to
- The carry propagates to all the 2 bit RC adders

CLA Adder – Stage II



Time Complexity

* In a similar manner, the carry propagates to all the RC adders at the zeroth level
* Each of them compute the correct result
* Time taken by Stage II :
  * Time taken for a carry to propagate from the (16,1) node to the RC adders
  * O(log(n))
* Total time : O(log(n) + log(n)) = O(log(n))

Time complexities of different adders:

- Ripple Carry Adder: $O(n)$
- Carry Select Adder: $O(\sqrt{n})$
- Carry Lookahead Adder: $O(\log(n))$

Outline

* Addition
* Multiplication
* Division
* Floating Point Addition
* Floating Point Multiplication
* Floating Point Division