Performance issues

- Addition & subtraction are fundamental operations performed frequently in the course of a computation
  - Performance (speed) of these operations has a strong impact on the overall performance of a computer
- Consider, again, the adder/subtractor unit
Adder/subtractor performance

- We are interested in the largest delay from the time the operands $X$ and $Y$ are presented as inputs until the time all bits of the sum $S$ and the final carry-out, $c_n$, are valid.
- Assume the adder is constructed as a ripple-carry adder and that each bit in the adder is constructed as a full adder as shown.

Adder/subtractor performance

- The delay for the carry-out in this circuit, $\Delta t$, is equal to two gate delays.
- From the discussion of the ripple-carry adder, we know that the final result of an $n$-bit addition is valid after a delay of $n\Delta t$. This is $2n$ gate delays.
- In addition to the delay in the ripple-carry path, there is also a one gate delay introduced in the XOR gates that provide either the true or complement form of $Y$ to the adder inputs.
  - The total gate delay for the adder/subtractor circuit is $2n+1$.
- The speed of any circuit is limited by the longest delay along the paths through the circuit.
  - The longest delay is called the critical-path-delay, and the path that causes this delay is called the critical path.
Carry-lookahead adder

• To reduce delay caused by the effect of carry propagation through the ripple-carry adder, we will attempt to evaluate quickly for each adder stage whether the carry-in from the previous stage will have a value of 0 or 1
  – If we can do this quickly, we can improve the performance of the complete adder
• Essentially we are attempting to reduce the critical-path-delay

Carry-lookahead adder

• Recall the carry-out function for stage $I$ can be realized as
  
  $c_{i+1} = x_iy_i + x_ic_i + y_ic_i$
  
  $c_{i+1} = x_iy_i + (x_i + y_i)c_i$

• Let $g_i = x_iy_i$ and $p_i = x_i + y_i$, so $c_{i+1} = g_i + p_ic_i$
• The function $g_i = 1$ when both $x_i$ and $y_i$ are 1, regardless of the incoming carry $c_i$
  – Since in this case, stage $i$ is guaranteed to generate a carry-out, $g$ is called the **generate** function
• The function $p_i = 1$ when either $x_i$ and $y_i$ are 1
  A carry-out is produced if $c_i = 1$
  – The effect is that the carry-in of 1 is propagated through stage $i$; $p$ is called the **propagate** function
Carry-lookahead (CLA) adder

Let us generate an expression for the output carry of an n-bit adder given,
\[ c_n = g_{n-1} + p_{n-1} c_{n-1} \]
and,
\[ c_{n-1} = g_{n-2} + p_{n-2} c_{n-2} \]
therefore,
\[ c_n = g_{n-1} + p_{n-1} (g_{n-2} + p_{n-2} c_{n-2}) \]
\[ c_n = g_{n-1} + p_{n-1} g_{n-2} + p_{n-1} p_{n-2} c_{n-2} \]

The same expansion for other stages, ending with stage 0, gives
\[ c_n = g_{n-1} + p_{n-1} g_{n-2} + p_{n-1} p_{n-2} g_{n-3} + \ldots + p_{n-1} p_{n-2} \ldots p_1 g_0 + p_{n-1} p_{n-2} \ldots p_0 c_0 \]
Ripple-carry adder critical path

3 gate delays for \( c_1 \),
5 gate delays for \( c_2 \)

In general, \( 2n+1 \) delays for \( n \)-bit ripple-carry adder

Carry-lookahead critical path

3 gate delays for \( c_1 \),
3 gate delays for \( c_2 \),
3 gate delays for \( c_n \)

Total delay for \( n \)-bit CLA adder is 4 gate delays
All \( g_i \) and \( p_i \), one delay
All \( c_i \), two more delays
One more delay for the sums \( s_i \)
Carry-lookahead limitations

- The expression for carry in a CLA adder
  \[ c_n = g_{n-1} + p_{n-1} g_{n-2} + p_{n-1} p_{n-2} g_{n-3} + \ldots + p_{n-1} p_{n-2} \ldots p_1 g_0 + p_{n-1} p_{n-2} \ldots p_0 c_0 \]
- obviously results in a fast solution (since it is only a 2 level AND-OR function)
- Fan-in limitations may effectively limit the speed of a CLA adder
  - Devices with known fan-in limitations (such as an FPGA) often include dedicated circuitry for implementation of fast adders
- The complexity of an \( n \)-bit CLA adder increases rapidly as \( n \) becomes large
  - To reduce this complexity, we can use a **hierarchical** approach in designing large adders

32-bit adder design

- Suppose we want to design a 32-bit adder
- Divide this adder into 4 blocks such that
  - Bits \( b_{7-0} \) are block 0
  - Bits \( b_{15-8} \) are block 1
  - Bits \( b_{23-16} \) are block 2
  - Bits \( b_{31-24} \) are block 3
- Each block can be constructed as an 8-bit CLA adder
  - The carry-out signals from the four blocks are \( c_8, c_{16}, c_{24}, \) and \( c_{32} \)
- There are 2 basic approaches for interconnecting these four blocks
  - Ripple-carry between blocks
  - Second level carry-lookahead circuit
Ripple-carry between blocks

Second level carry-lookahead circuit
Second level carry-lookahead circuit

- For the second level circuit:
  
  \[ P_0 = p_7p_6p_5p_4p_3p_2p_1p_0 \]
  
  \[ G_0 = g_7 + p_7g_6 + p_7p_6g_5 + \ldots + p_7p_6p_5p_4p_3p_2p_1g_0 \]
  
  \[ c_8 = G_0 + P_0c_0 \]

  \[ c_{16} = G_1 + P_1c_8 = G_1 + P_1G_0 + P_1P_0c_0 \]

  \[ c_{24} = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0c_0 \]

  \[ c_{32} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0 \]

Hierarchical CLA analysis

- Assuming a fan-in constraint of four inputs, the time to add two 32-bit numbers involves
  
  - five gate delays to develop the \( G_i \) and \( P_i \) terms,
  
  - three gate delays for the second-level lookahead,
  
  - and one delay (XOR) to produce the final sum bits.

  - Actually the final sum bit is computed after eight delays because \( c_{32} \) is not used to determine the sum bits.

  - The complete operation, including overflow detection \((c_{31} \oplus c_{32})\) takes **nine gate delays** (compared to 65 for the ripple carry adder)