1-bit “state” is carry-in bit
Mealy machine: output (sum) depends on state and inputs
Example of Serial Adder

\[ a = 10110 \quad \text{sum} = 100101 \]
\[ b = 01111 \]

Clock Rising Edge:
- cin \leftrightarrow cout
- recompute sum, cout

Before Next Clock:
- inputs a,b may change
- recompute sum, cout
Another Example

\[ a = 10110 \quad \text{sum} = (1)00001 \]
\[ b = 01011 \]

Clock Rising Edge:
- \( \text{cin} \leftrightarrow \text{cout} \)
- recompute sum, cout

Before Next Clock:
- inputs \( a, b \) may change
- recompute sum, cout

Sum is correct only
- just before clock
2-bit “state” is carry-in bit and (previous) sum Moore machine: output (sum) depends on state only

Serial Adder - Moore

Combinational Logic (FA)
The Same Example - Moore

\[ a = 10110 \quad \text{sum} = (1)00001 \]
\[ b = 01011 \]

Clock Rising Edge:
- \( \text{cin} \leftrightarrow \text{cout} \)
- \( \text{sum} \leftrightarrow z \)
- recompute \( z, \text{cout} \)

Before Next Clock:
- inputs \( a,b \) may change
- recompute \( z, \text{cout} \)

Sum is available at next clock
More Bits At A Time

Let's add two bits at a time...

Is this faster?
Two-Bit Adder
Performance

First bit-serial adder:
- takes $2^N$ clock cycles to add $2^N$ bits
- smaller cycle time

Adding two bits at a time:
- takes $N$ clock cycles to add $2^N$ bits
- larger cycle time

Total time = (number of cycles) $\times$ (cycle period)
Building Blocks For Arithmetic

Binary Addition: recall the full-adder design.

\[ ab + as + bs \]

\[ \overline{ab} + \overline{ab} \]

\[ (\overline{a}b + \overline{a}b)s + (\overline{a}b + \overline{ab})\overline{s} \]
Integer Addition

Full-adder:

- Three input bits $a$, $b$, $s$
- Output: two bits sum and carry

Logic equations and gate diagram derived from truth-tables.

What about 4-bit addition?
Solution 1: write truth-table, derive logic equations, draw gate diagram.

Solution 2:

\[
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
+ & 0 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1
\end{array}
\]

Use a number of full-adders!
Integer Addition

2’s complement? Addition time for $N$ bits?
Observation: all we need is the carry-out...

⇒ compute carry-out \( cout \) for blocks

- **input:** 0 0, \( cout = 0 \) **kill**
- **input:** 1 1, \( cout = 1 \) **generate**
- **input:** 0 1 or 1 0, \( cout = \text{carry-in (cin)} \) **propagate**

\[
cout = cin \cdot P + G \\
G = a \cdot b \\
P = a + b
\]

**Block codes:**

\[
G_{01} = G_1 + G_0P_1 \\
P_{01} = P_0P_1
\]
Integer Addition

**Carry Lookahead adder:** compute block codes to speed up carry computation.
We want to build an n-bit carry-lookahead adder ... 
- a, b, cin are the inputs 
- G, P, sum are the outputs
Build a 2n-bit adder from two n-bit ones

Use “divide-and-conquer” approach

A[2n]
Carry Lookahead

Equations for G,P:

\[ G = G_h + P_h \cdot G_l \]

\[ P = P_h \cdot P_l \]
Equation for cin:

\[ \text{cinl} = \text{cin} \]

\[ \text{cin} = \text{Gl} + \text{cinl} \cdot \text{Pl} \]
A crude approximation: \textit{phases}

\textbf{Phase 1:} compute all G,P values
\begin{align*}
T(1) &= \text{constant} \\
T(2n) &= T(n) + \text{constant}
\end{align*}
Solution: \(T(n)\) is \(O(\log n)\)

\text{\textbf{Phase 2:}} now compute sum \ldots how much longer?
\begin{align*}
S(1) &= \text{constant} \\
S(2n) &= \text{constant} + S(n)
\end{align*}
Solution: \(S(n)\) also is \(O(\log n)\)