CSC258: Computer Organization

Adders, Counters, and Memory

Device Building

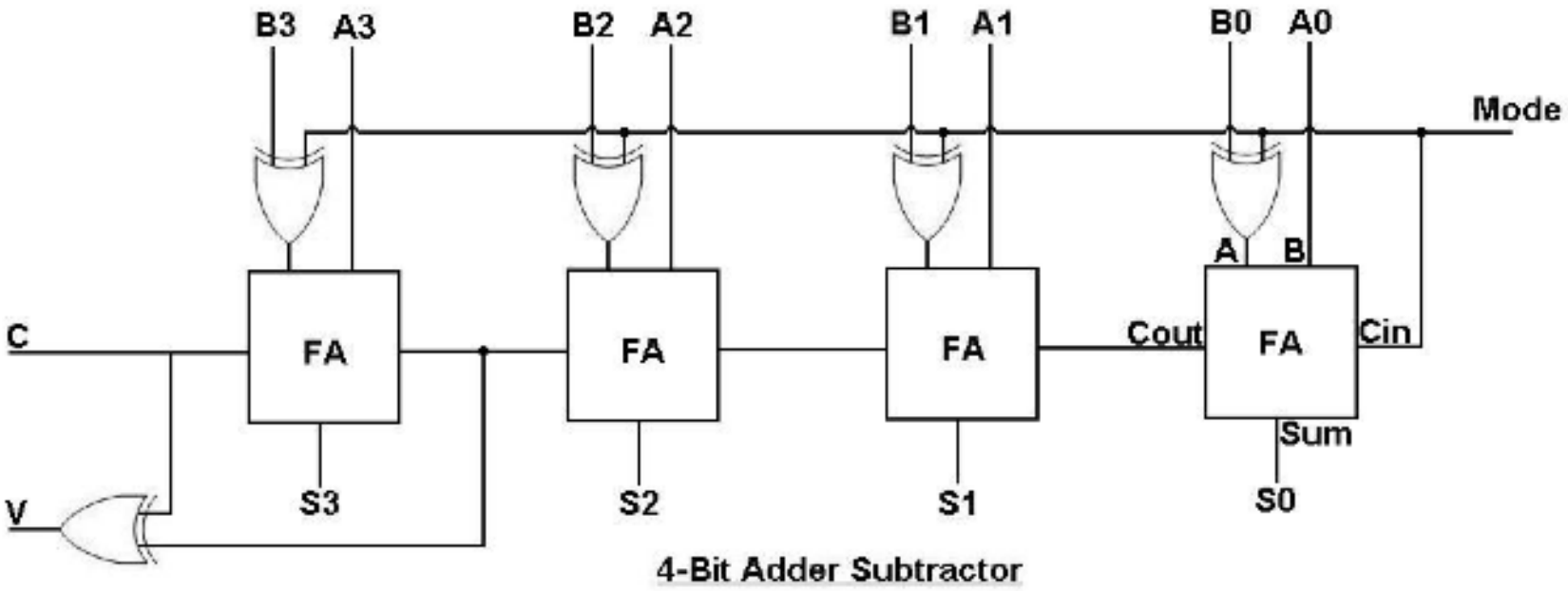
Adder-Subtracter

- Assume you have a 16-bit adder that has been validated on unsigned values.

- How do you create a 16-bit adder for 2's complement values that can perform addition *and* subtraction?
- Hint: Subtracting is adding a negative number.

A Ripple-Carry Adder-Subtracter

- To add, just use adders
- To subtract, use the complement of the second input and add 1



4-Bit Adder Subtractor

Reference: M. Mano, "Digital Design", 3rd ed., Prentice-Hall, 2002, pg. 127

4-bit Up-Down-Hold-Set Counter

- Build a counter that takes a 4-bit data input and a 2-bit control input.
- The control input determines whether the register (a) increments by 1, (b) decrements by 1, (c) holds the current value, or (d) loads a new value from the 4-bit data input.
- You may define which input of control corresponds to which operation.
- You may use any basic devices: registers, muxes, adders, etc.

4-bit Shift Register

- Build a counter that takes a 4-bit data input and a 1-bit control.
- The control input determines whether the register (a) loads a value from the 4-bit input or (b) performs a 1-bit circular shift on the existing input.
- You may use any basic devices: registers, muxes, adders, etc.

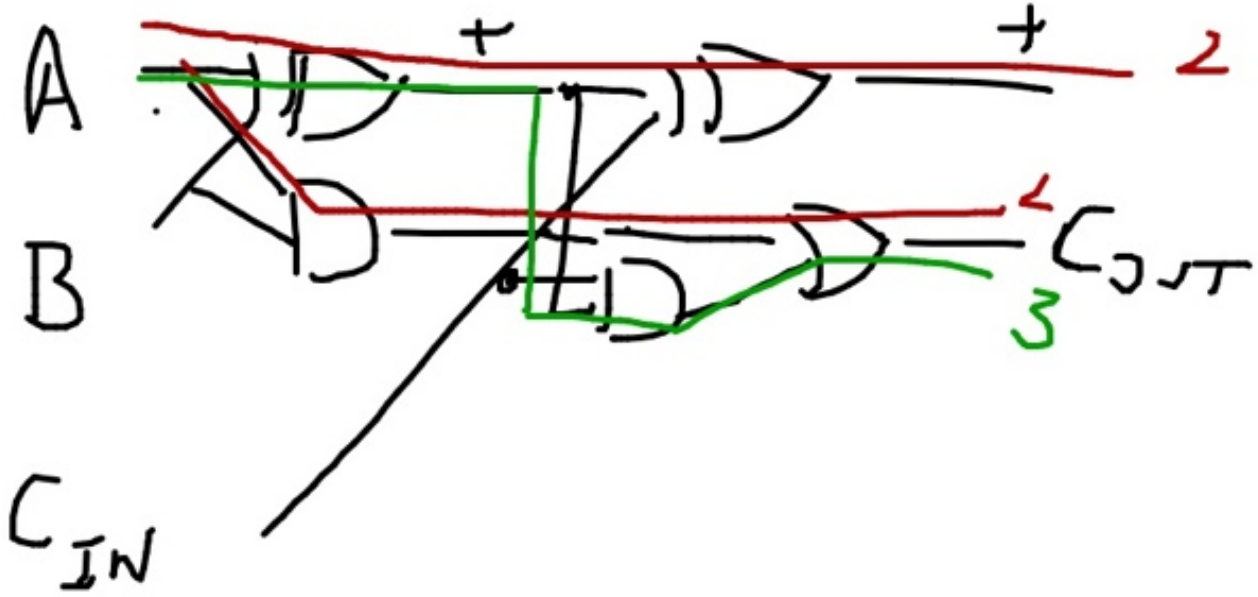
Adders

Full adder

- A full adder adds three numbers, not two: X, Y, and Z (carry-in)
- Two outputs are produced: C (carry-out) and + (the sum)
- To add a multi-bit number, adders must be chained series
  - Is this a problem?

X	Y	Z	C	+
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

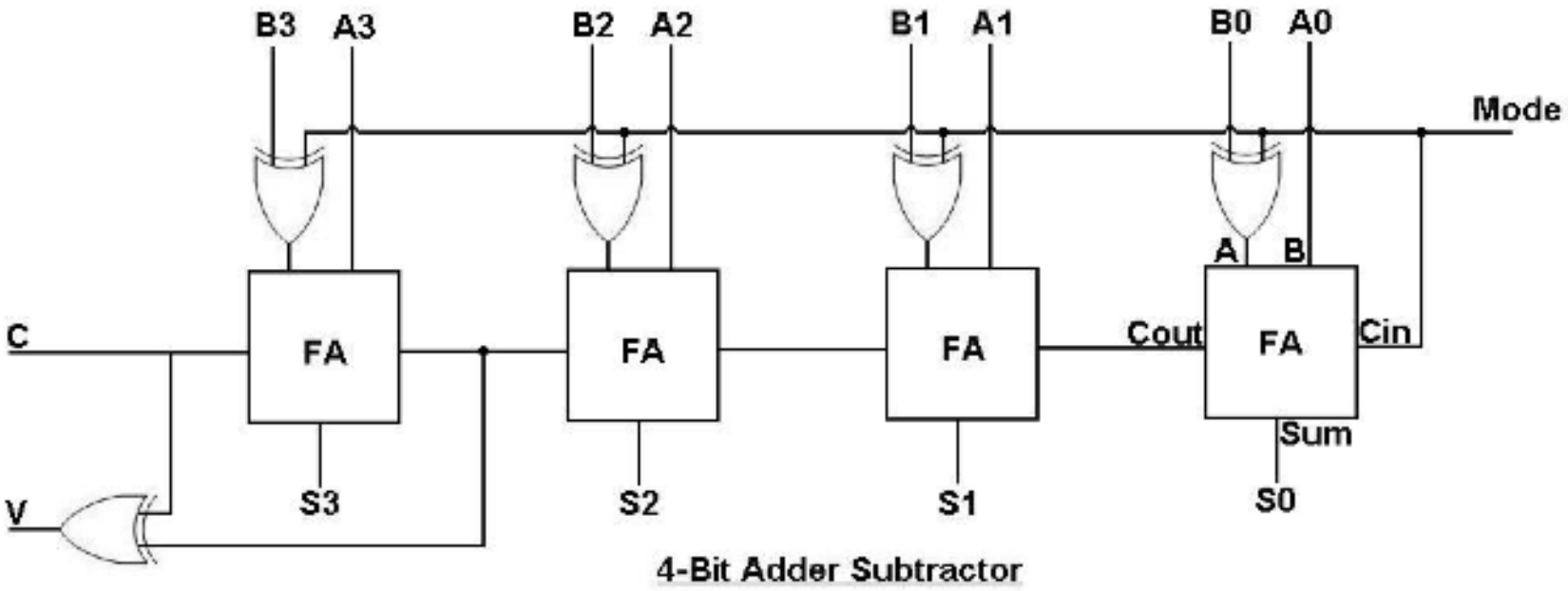
Full Adder Latency



To calculate the latency, find the longest path through the circuit.
In this case, it's the path from A (or B) to Cout -- a total of 3 gates.

A Ripple-Carry Adder-Subtracter

- To add, just use adders
- To subtract, use the complement of the second input and add 1



4-Bit Adder Subtractor

Reference: M. Mano, "Digital Design", 3rd ed., Prentice-Hall, 2002, pg. 127

Adders and Latency

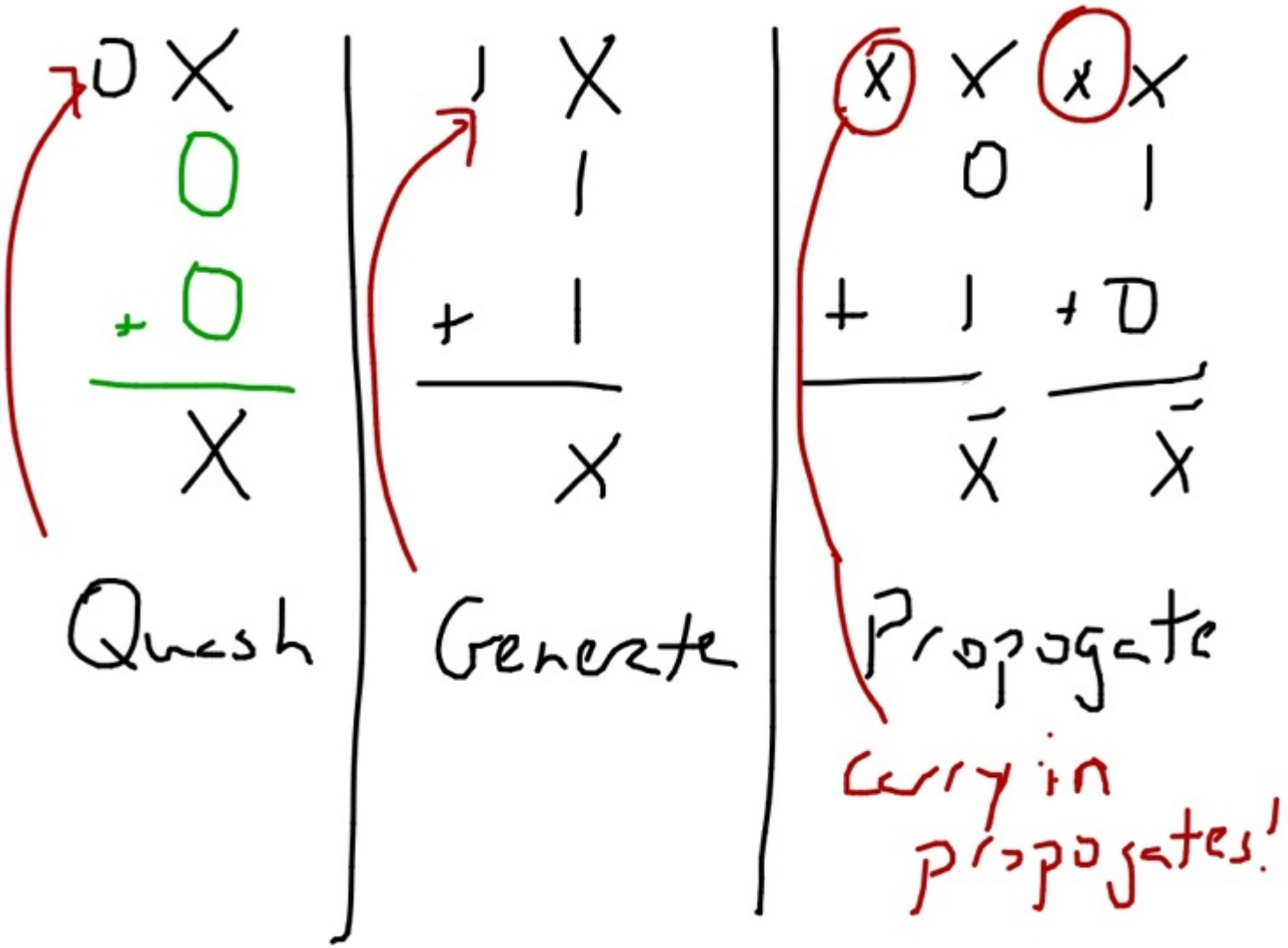
- Just chaining adders in series creates a *ripple-carry* design.
  - The name comes from the behavior of the carry-bits which "ripple" from the first adder to the last.

- The latency of the design is proportional to the number of adders!
  - Can we improve this latency ... ?

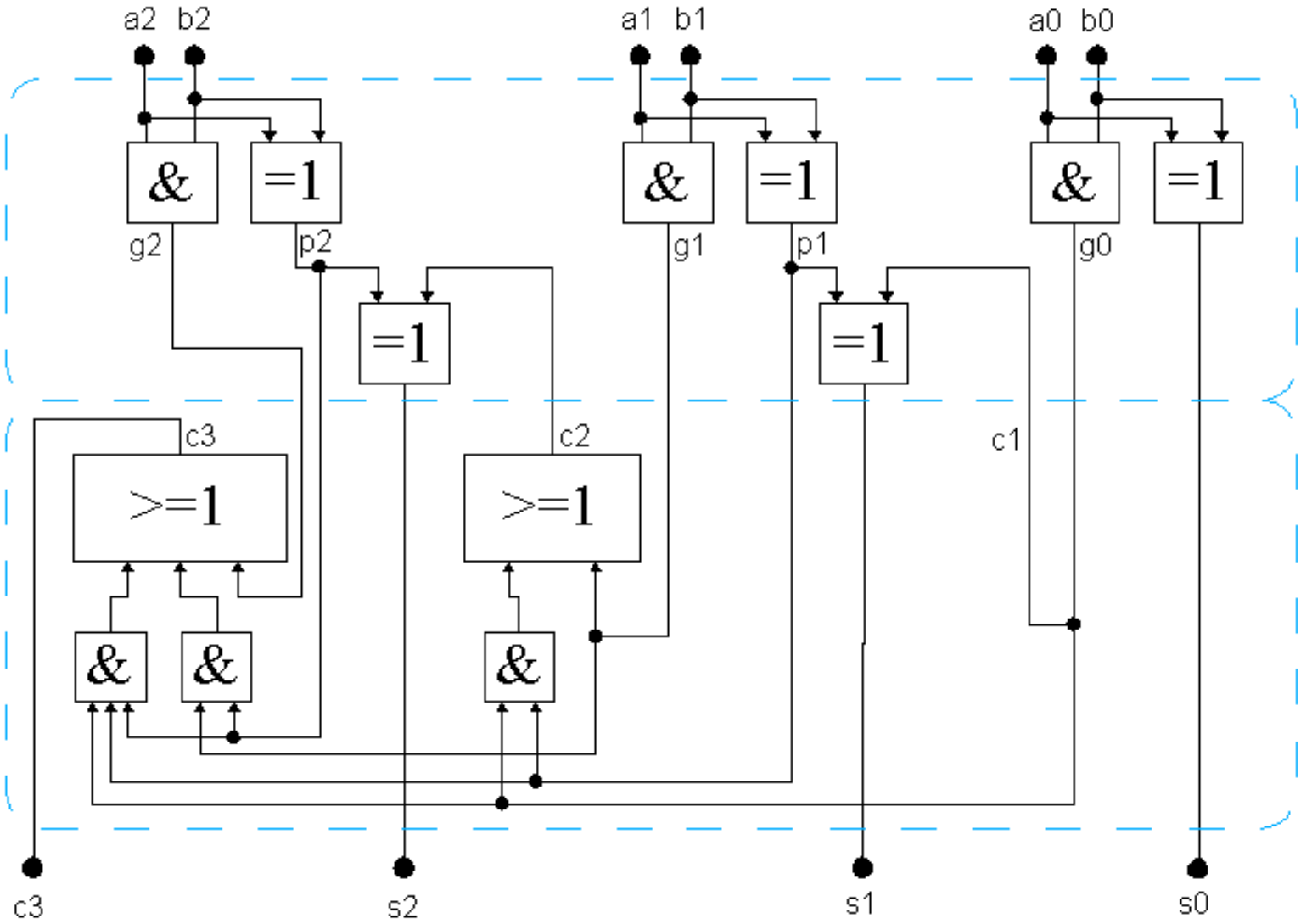
Faster Adders

- Carry look-ahead is a technique for speeding up an adder by calculating the carry-bit quickly.
- Parallel-prefix is a recursive, tree-like structure for computing the carry-bits.
- Adders that use these techniques pre-compute the value of carry-in as much as possible.
  - The key: each pair of bits that is added can ...
    (G)enerate a carry or
    (P)ropagate a carry or
    Quash (or Kill) a carry

Carry-bits: 3 Cases

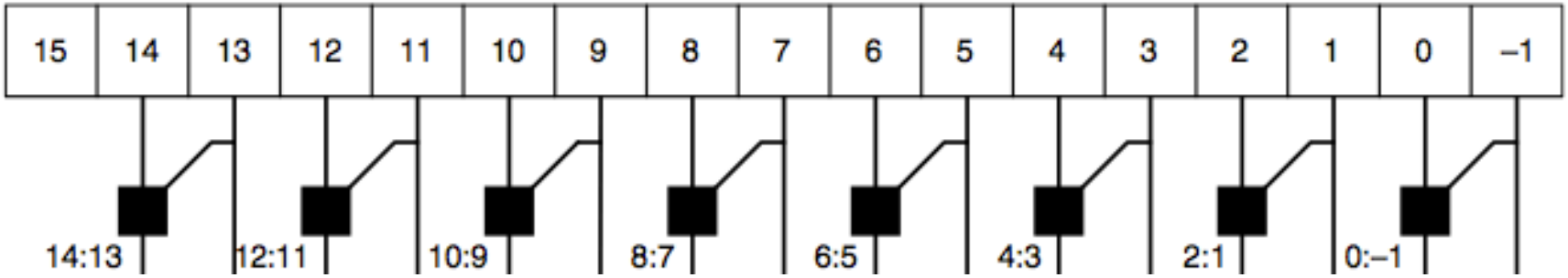


A Carry Look-Ahead Block



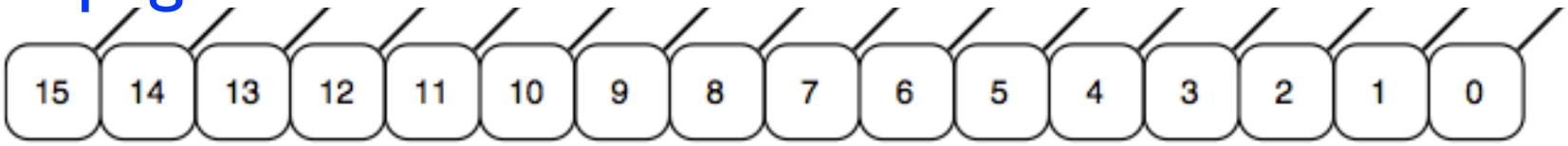
Blocks are wired in series, like the adder we saw earlier.

Prefix Adder Structure

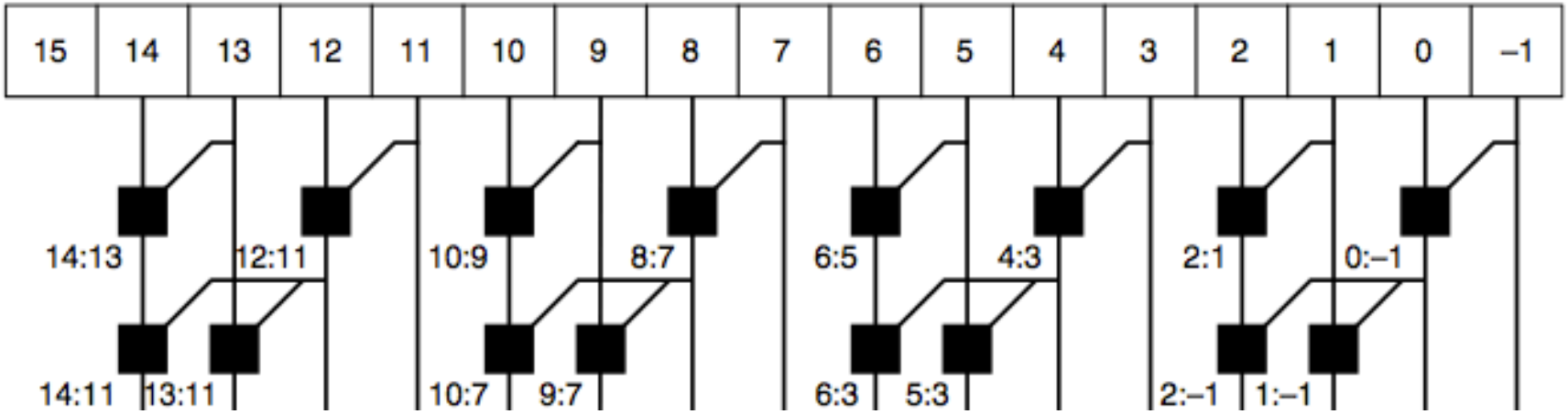


Instead of wiring the blocks in series, prefix adders build a tree-like structure to compute the carry-bit

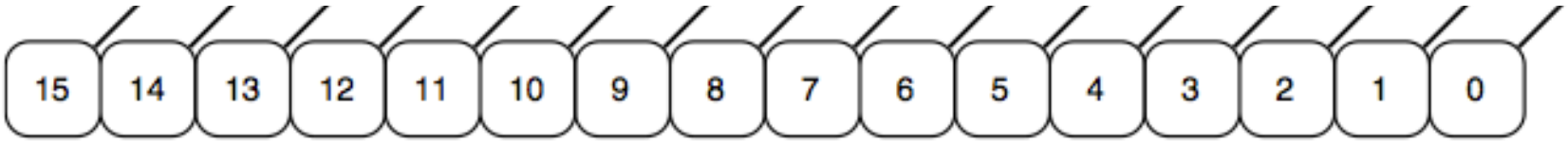
The first level computes the (G)enerate and (P)ropagate bits for two bits at a time.



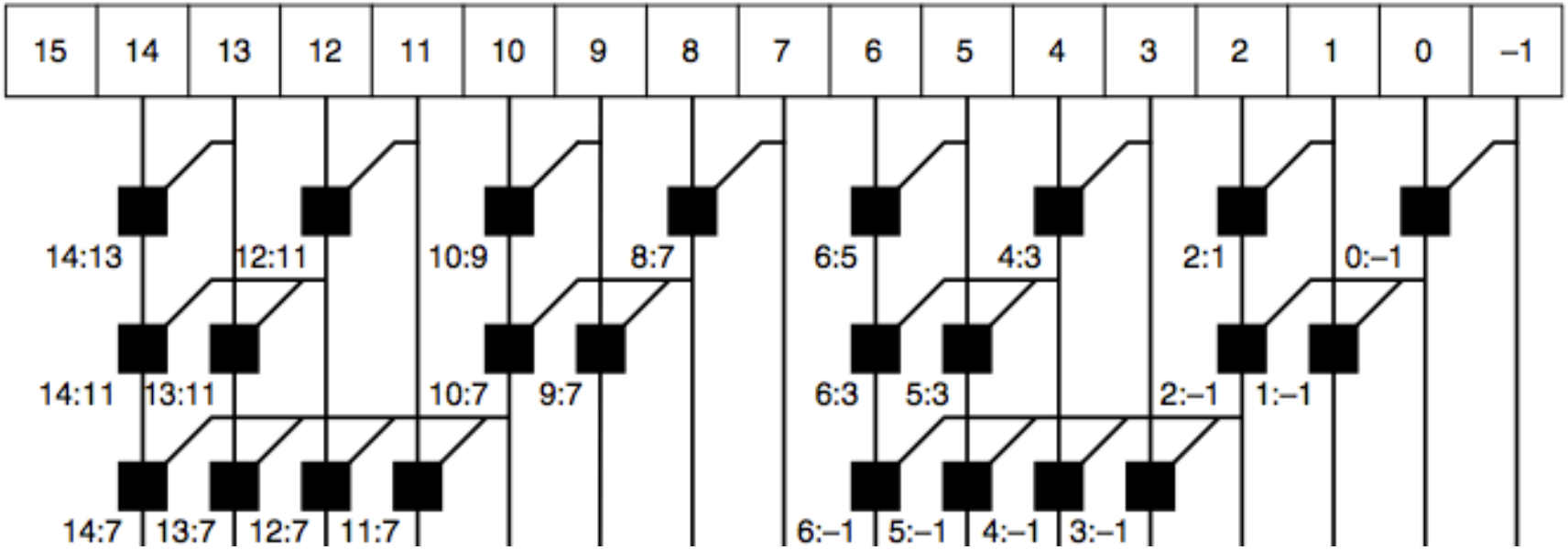
Prefix Adder Structure

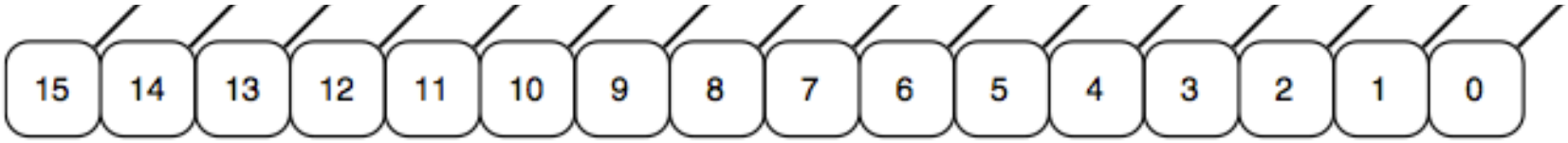


The second level uses the previous result to compute for four bits.

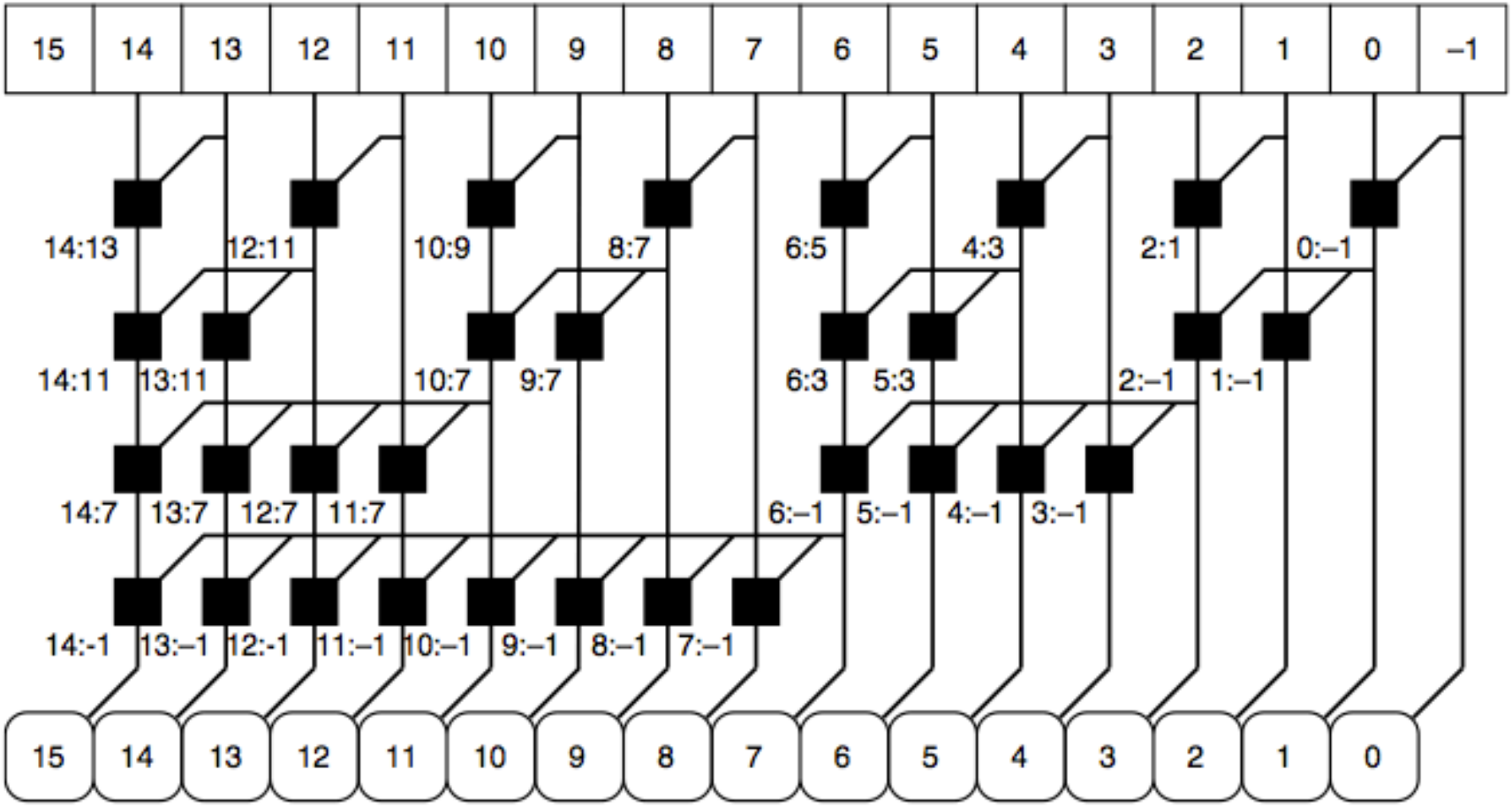


Prefix Adder Structure





Prefix Adder Structure



The structure growths in depth logarithmically.

Table-Based Hardware

Tables Instead of Gates

- Many fast operations (like fast division) rely on table look-ups.

- This requires a *memory* that can be accessed to get the value of the row.
- The memory stores the values of each row of the table.

Dot Notation Example

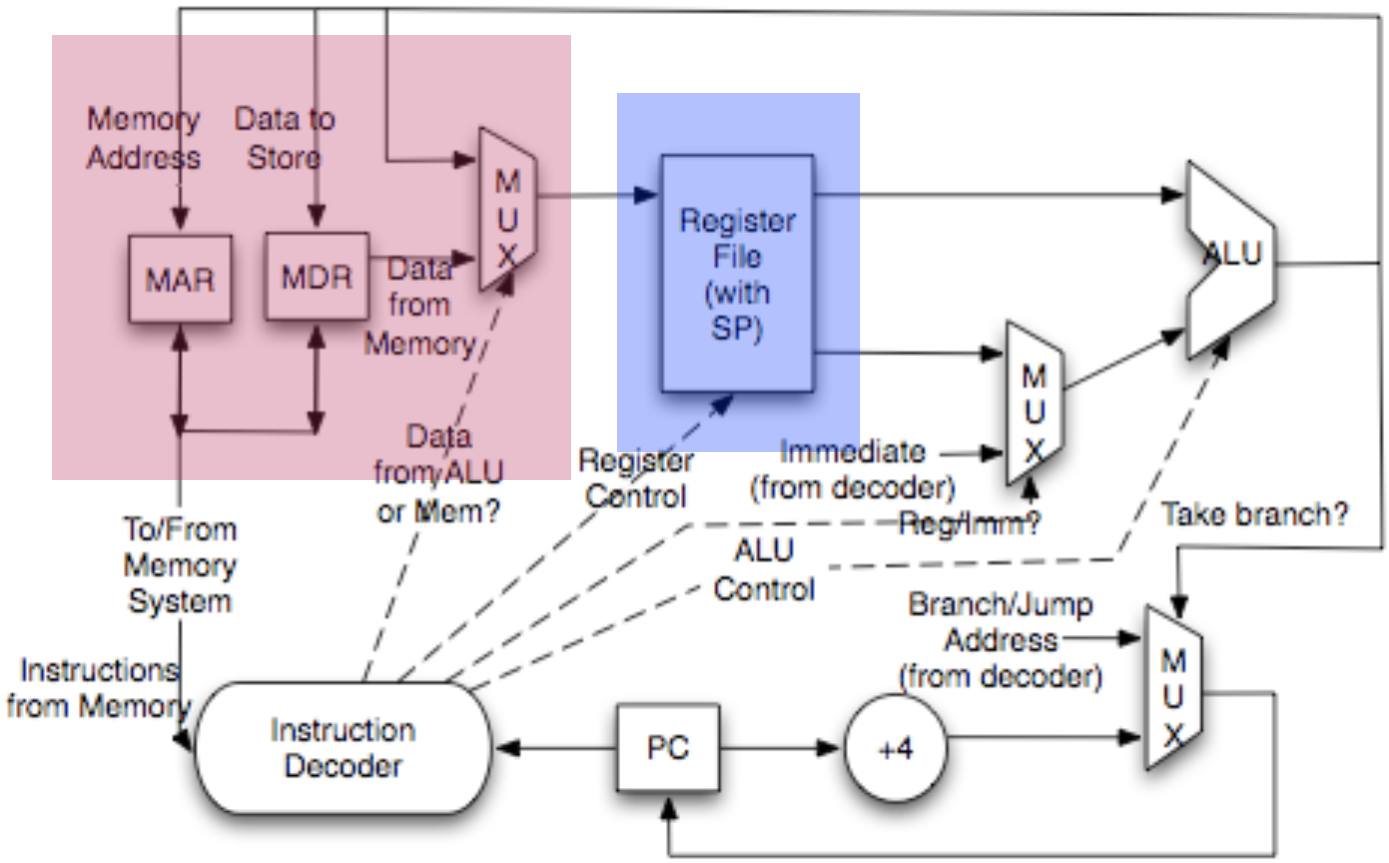
- Use dot notation to demonstrate how a 16-2 ROM can implement the following function:

  $X = ABC + (\neg B)D$
  $Y = (\neg A)CD + AB(\neg D) + \neg(BD)$

- Hint: think of the decoder as generating the rows of a truth table.

Memory

Memory in a Processor



Memory Addressing

- Memory is structured in bytes, words, and lines
- Each byte has an address: it is *addressable*
  - We often write addresses in base 16 (hex)

- Word-size differs from machine to machine, but it is typically 4 or 8 bytes.
- A line is a small set of words. We use lines to maximize performance. (More on this in week 12!)

Instructions and Data

- Everything in memory is stored in binary.
- There is no good way to determine if an arbitrary binary word is data ... or an instruction.
  - Many hacks use this to their advantage.
- This is actually a feature, not a bug. It means that we can treat instructions and data in the same way.
  - They're *both* data ... we just use instructions to run the machine.

Foreshadowing: Memory Layout