

# Addition Strategies: Counting On By Bases and then Ones (COBO)

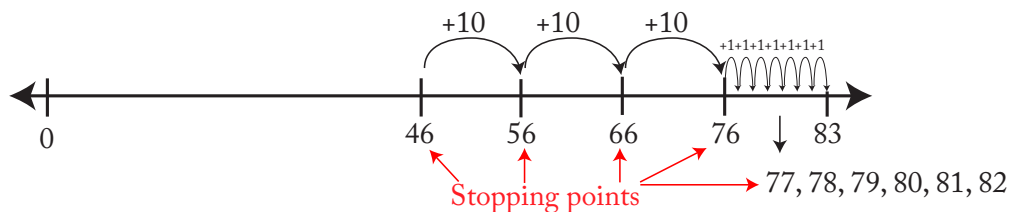
Compiled by: Theodore M. Savich

April 1, 2025

## Transcript

Video from Carpenter et al. (1999). Strategy descriptions and examples adapted from Hackenberg (2025).

- **Teacher:** Max has 46 comic books. For his birthday, his father gives him 37 more comic books. How many comic books does Max have now?
- **Lauren:** Forty-six ...
- **Teacher:** He gets 37 more for his birthday.
- **Lauren:** Ok. 46, 56, 66, 76, 77, 78, 79, 80, 81, 82, 83. It's 83.
- **Teacher:** Good work.



Notation Representing Sarah's Solution:

$$\begin{aligned}46 + 37 &= \square \\46 + 10 &= 56 \\56 + 10 &= 66 \\66 + 10 &= 76 \\76 + 1 &= 77 \\77 + 1 &= 78 \\78 + 1 &= 79 \\79 + 1 &= 80 \\80 + 1 &= 81 \\81 + 1 &= 82 \\82 + 1 &= 83\end{aligned}$$

### Description of Strategy:

**Objective:** Description of Counting On by Bases and Then Ones (COBO) Begin with one of the numbers. Break the other number into its base units and its ones. Then, “count on” by adding each base unit one at a time, followed by each individual one.

Why are number lines useful for demonstrating this strategy? COBO is essentially a jump strategy—you start at one number and make “jumps” equal to the other number’s base units, then add in the remaining ones. Number lines are ideal because they visually display jumps of varying lengths and directions. They serve as a picture of the process: a jump representing a full base is clearly larger (by a factor of the base) than a jump of a single unit.

Good number line illustrations should:

- Clearly represent the relative sizes of the jumps—each base jump should be exactly as many times larger than a single-unit jump as the base indicates, with all base jumps the same size and all one-unit jumps identical.
- Indicate the position of 0, or mark a break if that portion of the line isn’t drawn to scale.
- Use arrows to indicate direction—when adding, the jumps go to the right (or upward); when subtracting, they go to the left (or downward).
- Mark all landing points clearly—the numbers you would speak aloud when counting on by bases and then ones, just as Lauren demonstrated.

### Counting On by Bases and Then Ones (COBO)

#### Description of Strategy

- **Objective:** Start with one addend, add bases from the other addend one by one, then add ones one by one.
- **Example:**  $46 + 37$ 
  - Start at 46.
  - Add tens one by one:  $46 \rightarrow 56 \rightarrow 66 \rightarrow 76$ .
  - Add ones one by one:  $76 \rightarrow 77 \rightarrow \dots \rightarrow 83$ .

#### Automaton Type

**Finite State Automaton (FSA) with Counters:** Counters are used to manage the repeated addition:

- **BaseCounter:** Number of base units (e.g., tens) to add.
- **OneCounter:** Number of ones to add.
- **Sum:** The running total.

## Formal Description of the Automaton

We define the automaton as the tuple

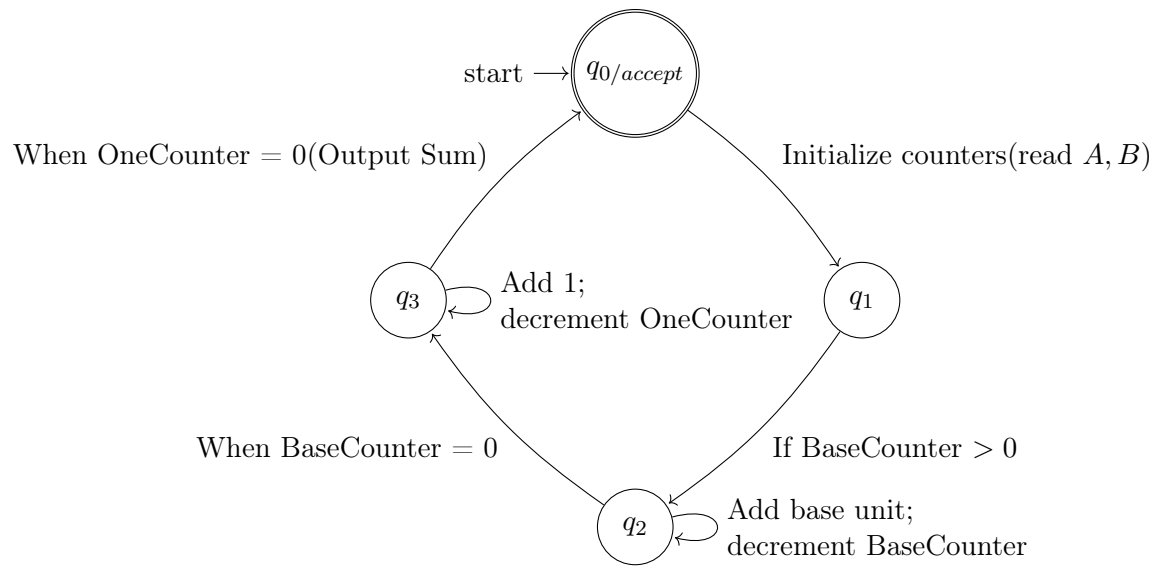
$$M = (Q, \Sigma, \delta, q_{0/accept}, F, C)$$

where:

- $Q = \{q_{0/accept}, q_1, q_2, q_3\}$  is the finite set of states. Here, the start state  $q_{0/accept}$  is also the accept state.
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +\}$  is the input alphabet (suitable for representing the addends).
- $F = \{q_{0/accept}\}$  is the set of accepting states.
- $C = \{\text{BaseCounter}, \text{OneCounter}, \text{Sum}\}$  is the set of counters.
- $\delta$  is the transition function defined by:
  1.  $\delta(q_{0/accept}, "A, B") = (q_1, \text{update: Sum} \leftarrow A, \text{BaseCounter} \leftarrow \lfloor B/10 \rfloor, \text{OneCounter} \leftarrow B \bmod 10)$   
(Read inputs  $A$  and  $B$ ; initialize the Sum to  $A$  and set the counters based on  $B$ .)
  2.  $\delta(q_1, \varepsilon) = (q_2, \text{if BaseCounter} > 0)$   
(If there are base units to add, proceed to add them.)
  3.  $\delta(q_2, \varepsilon) = (q_2, \text{update: Sum} \leftarrow \text{Sum} + \text{baseUnit}, \text{BaseCounter} \leftarrow \text{BaseCounter} - 1)$   
(In state  $q_2$ , repeatedly add one base unit (e.g., 10) to Sum while decrementing BaseCounter.)
  4.  $\delta(q_2, \varepsilon) = (q_3, \text{if BaseCounter} = 0)$   
(When no more base units remain, switch to adding ones.)
  5.  $\delta(q_3, \varepsilon) = (q_3, \text{update: Sum} \leftarrow \text{Sum} + 1, \text{OneCounter} \leftarrow \text{OneCounter} - 1)$   
(In state  $q_3$ , repeatedly add 1 to Sum while decrementing OneCounter.)
  6.  $\delta(q_3, \varepsilon) = (q_{0/accept}, \text{if OneCounter} = 0)$   
(When OneCounter reaches 0, output the final Sum and return to the closed start/accept state.)

## Automaton Diagram for COBO

The following diagram arranges the four states on a circle with  $q_{0/accept}$  serving as both the start and accept state.



## HTML Implementation

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Addition Strategies: Counting On By Bases and Ones (COBO)</title>
5   <style>
6     body {
7       font-family: sans-serif;
8     }
9     #diagramCOBOSVG {
10       border: 1px solid #d3d3d3;
11     }
12     #outputContainer {
13       margin-top: 20px;
14     }
15     .number-line-tick {
16       stroke: black;
17       stroke-width: 1;
18     }
19     .number-line-break {
20       stroke: black;
21       stroke-width: 1;
22       stroke-dasharray: 5 5;
23     }
24     .number-line-label {
25       font-size: 12px;
26       text-anchor: end; /* Change from middle to start for right-alignment */
27     }
28     /* Thinner jump arrows */
29     .jump-arrow {
30       fill: none;
31       stroke: blue;
32       stroke-width: 1; /* thinner stroke */
33     }
34     .jump-arrow-head {
35       fill: blue;
36       stroke: blue;
37     }
38     .jump-label {
39       font-size: 8px; /* Reduced from 12px to 8px */
40       text-anchor: middle;
41       fill: blue;
42     }
43     .tens-jump-label {
44       font-size: 12px;
45       text-anchor: middle;
46       fill: blue;
47     }
48     .stopping-point {
49       fill: red;
50       stroke: black;
51       stroke-width: 1;
52     }
```

```

53      /* New extended tick styles */
54      .extended-tick {
55          stroke: black;
56          stroke-width: 1;
57      }
58      .extended-tick-label {
59          font-size: 12px;
60          text-anchor: start;
61          fill: blue;
62      }
63      /* Number line arrowhead */
64      .number-line-arrow {
65          fill: black;
66          stroke: black;
67      }
68  </style>
69 </head>
70 <body>
71
72 <h1>Addition Strategies: Counting On By Bases and Then Ones (COBO)</h1>
73
74 <div>
75     <label for="coboAddend1">Addend 1:</label>
76     <input type="number" id="coboAddend1" value="46">
77 </div>
78 <div>
79     <label for="coboAddend2">Addend 2:</label>
80     <input type="number" id="coboAddend2" value="37">
81 </div>
82
83 <button onclick="runCOBOAutomaton()">Calculate and Visualize</button>
84
85 <div id="outputContainer">
86     <h2>Explanation:</h2>
87     <div id="coboOutput">
88         <!-- Text output will be displayed here -->
89     </div>
90 </div>
91
92 <h2>Diagram:</h2>
93 <svg id="diagramCOBOSVG" width="700" height="350">
94
95 </svg>
96
97 <script>
98 document.addEventListener('DOMContentLoaded', function() {
99     const outputElement = document.getElementById('coboOutput');
100     const coboAddend1Input = document.getElementById('coboAddend1');
101     const coboAddend2Input = document.getElementById('coboAddend2');
102     const diagramCOBOSVG = document.getElementById('diagramCOBOSVG');
103
104     window.runCOBOAutomaton = function() {
105         const addend1 = parseInt(coboAddend1Input.value);
106         const addend2 = parseInt(coboAddend2Input.value);

```

```

107     if (isNaN(addend1) || isNaN(addend2)) {
108         outputElement.textContent = 'Please enter valid numbers for both addends';
109         return;
110     }
111
112     // Build text explanation
113     let output = '<h2>Counting On by Bases and Ones (COBO)</h2>';
114     output += '<p><strong>Problem:</strong> ${addend1} + ${addend2}</p>';
115
116     const tens = Math.floor(addend2 / 10) * 10;
117     const ones = addend2 % 10;
118
119     output += '<p>Step 1: Split ${addend2} into ${tens} + ${ones}</p>';
120
121     let currentSum = addend1;
122     const tensSteps = [];
123     if (tens > 0) {
124         output += '<p>Step 2: Count on by tens</p>';
125         for (let i = 10; i <= tens; i += 10) {
126             tensSteps.push({ from: currentSum, to: currentSum + 10, action: 'Add 10' });
127             currentSum += 10;
128         }
129         tensSteps.forEach(step => {
130             output += '<p>${step.from} + ${step.action} = ${step.to}</p>';
131         });
132     }
133
134     const onesSteps = [];
135     if (ones > 0) {
136         output += '<p>Step ${tens > 0 ? '3' : '2'}: Count on by ones</p>';
137         for (let i = 1; i <= ones; i++) {
138             onesSteps.push({ from: currentSum, to: currentSum + 1, action: 'Add 1' });
139             currentSum += 1;
140         }
141         onesSteps.forEach(step => {
142             output += '<p>${step.from} + ${step.action} = ${step.to}</p>';
143         });
144     }
145
146     output += '<p>Result: ${addend1} + ${addend2} = ${currentSum}</p>';
147     outputElement.innerHTML = output;
148
149     // Draw the diagram
150     drawNumberLineDiagram(addend1, addend2, tensSteps, onesSteps, currentSum);
151 };
152
153 function drawNumberLineDiagram(addend1, addend2, tensSteps, onesSteps, finalSum) {
154     const svg = diagramCOBOSVG;
155     svg.innerHTML = ''; // Clear any previous diagram
156
157     // Dimensions
158     const width = parseFloat(svg.getAttribute('width'));
159     const height = parseFloat(svg.getAttribute('height'));

```

```

160     const marginLeft = 50;
161     const marginRight = 50;
162     const numberLineY = height / 2;
163
164     // Arc heights
165     const TENS_ARC_HEIGHT = 30;
166     const ONES_ARC_HEIGHT = 15;
167
168     // We'll place 0 at x=marginLeft, then a scale break if addend1 > ~0
169     // then line from addend1 to finalSum to scale
170     const zeroX = marginLeft;
171     const breakX = zeroX + 15;
172     const lineStartX = breakX + 15;
173
174     // Extend the line 10 points past the final sum
175     const extendAmount = 10;
176     const numericRange = Math.max(finalSum + extendAmount - addend1, 1); // at least 1
177     // to avoid /0
178     const lineEndX = width - marginRight;
179
180     // Calculate scale after considering the extension
181     const scale = (lineEndX - lineStartX) / numericRange;
182
183     // Draw the number line from zero to the break point
184     drawLine(zeroX, numberLineY, lineStartX, numberLineY);
185
186     // Draw "0" tick
187     drawTick(zeroX, numberLineY, 10);
188     createText(zeroX, numberLineY + 15, '0');
189
190     // If addend1 > 0, draw scale break
191     if (addend1 > 0) {
192         drawScaleBreakSymbol(breakX, numberLineY);
193     }
194
195     // Main line with arrowhead
196     drawLine(lineStartX, numberLineY, lineEndX, numberLineY);
197
198     // Add arrowhead to the right end of number line
199     const arrowSize = 10;
200     const arrowHead = document.createElementNS('http://www.w3.org/2000/svg', 'path');
201     arrowHead.setAttribute('d', 'M ${lineEndX-arrowSize} ${numberLineY-arrowSize/2} L
202         ${lineEndX} ${numberLineY} L ${lineEndX-arrowSize} ${numberLineY+arrowSize/2}
203         Z');
204     arrowHead.setAttribute('class', 'number-line-arrow');
205     svg.appendChild(arrowHead);
206
207     // Convert a value to x-coord
208     function valueToX(v) {
209         return lineStartX + (v - addend1) * scale;
210     }
211
212     // Mark addend1

```



```

211 drawTick(valueToX(addend1), numberLineY, 10);
212 createText(valueToX(addend1), numberLineY + 15, addend1.toString());
213
214 // Draw tens jumps
215 tensSteps.forEach((step) => {
216     const x1 = valueToX(step.from);
217     const x2 = valueToX(step.to);
218     createJumpArrow(svg, x1, numberLineY, x2, numberLineY, TENS_ARC_HEIGHT);
219     // Mark landing
220     drawTick(x2, numberLineY, 10);
221     createText(x2, numberLineY + 15, step.to.toString());
222
223     // Add "+10" label above the arc
224     const midX = (x1 + x2) / 2;
225     const labelY = numberLineY - TENS_ARC_HEIGHT - 5;
226     const txtTensLabel = document.createElementNS('http://www.w3.org/2000/svg', '
        text');
227     txtTensLabel.setAttribute('x', midX);
228     txtTensLabel.setAttribute('y', labelY);
229     txtTensLabel.setAttribute('class', 'tens-jump-label');
230     txtTensLabel.textContent = '+10';
231     svg.appendChild(txtTensLabel);
232 });
233
234 // Draw ones jumps
235 onesSteps.forEach((step, index) => {
236     const x1 = valueToX(step.from);
237     const x2 = valueToX(step.to);
238
239     // Create jump arrow as before
240     createJumpArrow(svg, x1, numberLineY, x2, numberLineY, ONES_ARC_HEIGHT);
241
242     // Create extended tick with increasing length based on index
243     const tickLength = 10 + (index * 10); // Increase by 10px for each subsequent
        tick
244     const extendedTick = document.createElementNS('http://www.w3.org/2000/svg', '
        line');
245     extendedTick.setAttribute('x1', x2);
246     extendedTick.setAttribute('y1', numberLineY);
247     extendedTick.setAttribute('x2', x2);
248     extendedTick.setAttribute('y2', numberLineY + tickLength); // Now going down
        instead of up
249     extendedTick.setAttribute('class', 'extended-tick');
250     svg.appendChild(extendedTick);
251
252     // Add the number label at the end of the tick mark
253     createText(x2, numberLineY + tickLength + 15, step.to.toString());
254
255     // Add "+1" label above the arc
256     const midX = (x1 + x2) / 2;
257     const labelY = numberLineY - ONES_ARC_HEIGHT - 5;
258     const txtOneLabel = document.createElementNS('http://www.w3.org/2000/svg', '
        text');
259     txtOneLabel.setAttribute('x', midX);

```

```

260     txtOneLabel.setAttribute('y', labelY);
261     txtOneLabel.setAttribute('class', 'jump-label');
262     txtOneLabel.textContent = '+1';
263     svg.appendChild(txtOneLabel);
264 });
265
266 // Mark finalSum if not already marked in ones steps
267 if (onesSteps.length === 0 || onesSteps[onesSteps.length - 1].to !== finalSum) {
268     drawTick(valueToX(finalSum), numberLineY, 10);
269     createText(valueToX(finalSum), numberLineY + 15, finalSum.toString());
270 }
271
272 // ----- Drawing Helpers -----
273
274 function drawLine(x1, y1, x2, y2) {
275     const line = document.createElementNS('http://www.w3.org/2000/svg', 'line');
276     line.setAttribute('x1', x1);
277     line.setAttribute('y1', y1);
278     line.setAttribute('x2', x2);
279     line.setAttribute('y2', y2);
280     line.setAttribute('class', 'number-line-tick');
281     svg.appendChild(line);
282 }
283
284 function drawTick(x, y, size) {
285     const tick = document.createElementNS('http://www.w3.org/2000/svg', 'line');
286     tick.setAttribute('x1', x);
287     tick.setAttribute('y1', y - size / 2);
288     tick.setAttribute('x2', x);
289     tick.setAttribute('y2', y + size / 2);
290     tick.setAttribute('class', 'number-line-tick');
291     svg.appendChild(tick);
292 }
293
294 function createText(x, y, textContent) {
295     const txt = document.createElementNS('http://www.w3.org/2000/svg', 'text');
296     txt.setAttribute('x', x);
297     txt.setAttribute('y', y);
298     txt.setAttribute('class', 'number-line-label');
299     txt.textContent = textContent;
300     svg.appendChild(txt);
301 }
302
303 function drawScaleBreakSymbol(x, y) {
304     // Two small diagonal lines crossing
305     const breakLine1 = document.createElementNS('http://www.w3.org/2000/svg', '
306         line');
307     breakLine1.setAttribute('x1', x);
308     breakLine1.setAttribute('y1', y - 8);
309     breakLine1.setAttribute('x2', x + 8);
310     breakLine1.setAttribute('y2', y + 8);
311     breakLine1.setAttribute('class', 'number-line-break');
312     svg.appendChild(breakLine1);

```

```

313     const breakLine2 = document.createElementNS('http://www.w3.org/2000/svg', '
        line');
314     breakLine2.setAttribute('x1', x);
315     breakLine2.setAttribute('y1', y + 8);
316     breakLine2.setAttribute('x2', x + 8);
317     breakLine2.setAttribute('y2', y - 8);
318     breakLine2.setAttribute('class', 'number-line-break');
319     svg.appendChild(breakLine2);
320 }
321
322 /**
323  * Draws a curved jump (quadratic Bezier) from (x1, y1) to (x2, y2),
324  * with control point arcHeight above the line, and attaches a manual arrowhead.
325  */
326 function createJumpArrow(svg, x1, y1, x2, y2, jumpArcHeight) {
327     // Quadratic Bezier arc
328     const cx = (x1 + x2) / 2; // midpoint in x
329     const cy = y1 - jumpArcHeight; // arc above the line
330
331     // Main arc path
332     const arcPath = document.createElementNS('http://www.w3.org/2000/svg', 'path')
        ;
333     arcPath.setAttribute('d', 'M ${x1} ${y1} Q ${cx} ${cy} ${x2} ${y2}');
334     arcPath.setAttribute('class', 'jump-arrow');
335     svg.appendChild(arcPath);
336
337     // Compute angle for arrowhead
338     // derivative at end of Q-bezier ~ direction from control point to end
339     const dx = x2 - cx;
340     const dy = y2 - cy;
341     const angleRad = Math.atan2(dy, dx);
342     const angleDeg = angleRad * (180 / Math.PI);
343
344     // Manual arrowhead as small filled triangle
345     const arrowSize = 5;
346     const arrowHead = document.createElementNS('http://www.w3.org/2000/svg', 'path
        ');
347     arrowHead.setAttribute('class', 'jump-arrow-head');
348     arrowHead.setAttribute('d', 'M 0 0 L ${arrowSize} ${arrowSize/2} L ${arrowSize
        } ${-arrowSize/2} Z');
349     arrowHead.setAttribute('transform', 'translate(${x2}, ${y2}) rotate(${angleDeg
        + 180})');
350     svg.appendChild(arrowHead);
351 }
352 };
353 });
354 </script>
355
356 <!-- New button for viewing PDF documentation -->
357 <button onclick="openPdfViewer()">Want to learn more about this strategy? Click here.</
    button>
358
359 <script>
360     function openPdfViewer() {

```

```
361     // Opens the PDF documentation for the strategy.
362     window.open('../SAR_ADD_COBO.pdf', '_blank');
363 }
364 </script>
365
366 </body>
367 </html>
```

## References

- Carpenter, T. P., Fennema, E., Franke, M. L., Levi, L., & Empson, S. B. (1999). *Children's mathematics: Cognitively guided instruction* [Includes supplementary material: Children's mathematics: Cognitively guided instruction – videotape logs]. Heinemann; The National Council of Teachers of Mathematics, Inc.
- Hackenberg, A. (2025). *Course notes* [Unpublished course notes].