

Addition Strategies: Rearranging to Make Bases (RMB)

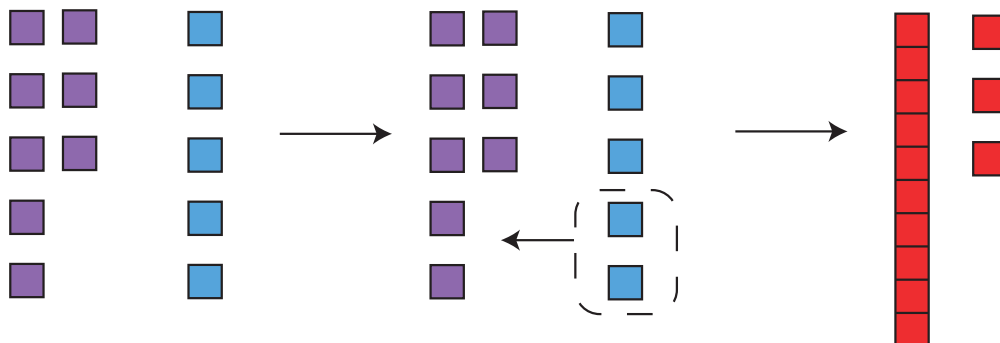
Compiled by: Theodore M. Savich

March 11, 2025

Transcript

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

- **Teacher:** Lucy is eight fish. She buys five more fish. How many fish will Lucy have then?
- **Sarah:** 13.
- **Teacher:** How'd you get 13?
- **Sarah:** Well, because eight plus two is ten, but then two plus three is five. And she wants to buy five more fish. So you take care of two, and you need to add three more. And so I add three more, and you get 13.



Notation Representing Sarah's Solution:

$$\begin{aligned}8 + 5 &= \square \\8 + 2 &= 10 \\2 + 3 &= 5 \\8 + 5 &= 10 + 3 \\8 + 5 &= 13\end{aligned}$$

Description of Strategy:

Objective: Rearranging to Make Bases (RMB) means shifting the extra ones from one addend over to the other so that one of the numbers becomes a complete multiple of the base (a whole “group” of that base). This rearrangement simplifies the addition process because there are established

patterns for adding an exact multiple of the base. In other words, when you add a full group of base units to a number, the ones digit stays the same while only the digit representing the base (like the tens place) increases.

Rearranging to Make Bases (RMB)

Description of Strategy

- **Objective:** Make one of the addends a whole number of bases by moving ones from the other addend.
- **Example:** $8 + 5$
 - Move 2 ones from 5 to 8 to make 10.
 - Remaining ones in the second addend: $5 - 2 = 3$.
 - Add the adjusted numbers: $10 + 3 = 13$.

Automaton Type

Pushdown Automaton (PDA): Needed to handle digits and to remember the number of ones moved via the stack.

Formal Description of the Automaton

We define the PDA as the 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_{0/accept}, Z_0, F)$$

where

- $Q = \{q_{0/accept}, q_1, q_2, q_3, q_4, q_5\}$ is the finite set of states.
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +\}$ is the input alphabet (suitable for representing addends).
- $\Gamma = \{Z_0\} \cup \{x \mid x \in \mathbb{N}\}$ is the stack alphabet, where:
 - Z_0 is the initial (bottom) stack symbol.
 - A symbol x represents the number of ones moved.
- $q_{0/accept}$ is the start state, which is also the accept state.
- Z_0 is the initial stack symbol.
- $F = \{q_{0/accept}\}$ is the set of accepting states.

The transition function

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$$

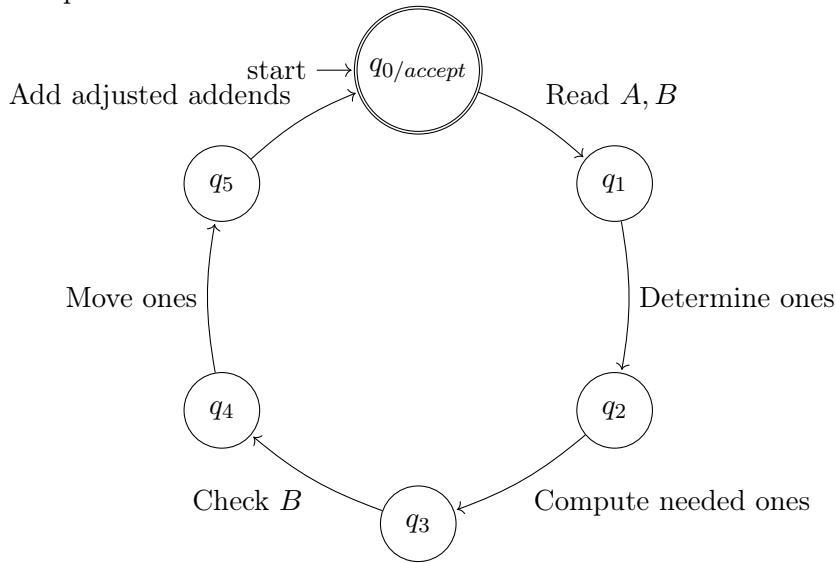
is defined by the following key transitions:

1. $\delta(q_{0/accept}, "A, B", Z_0) = \{(q_1, Z_0)\}$ (Read inputs A and B).
2. $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$ (Determine the ones digits of A and B).

3. $\delta(q_2, \varepsilon, Z_0) = \{(q_3, Z_0)\}$ (Compute the number of ones needed to make A a full base).
4. $\delta(q_3, \varepsilon, Z_0) = \{(q_4, k Z_0)\}$ (If B has at least k ones, push k onto the stack).
5. $\delta(q_4, \varepsilon, k) = \{(q_5, k)\}$ (Move k ones from B to A and adjust the addends).
6. $\delta(q_5, \varepsilon, k) = \{(q_0/accept, Z_0)\}$ (Add the adjusted numbers, output the result, and pop k from the stack).

Automaton Diagram for RMB

The following TikZ picture arranges the 6 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>Rearranging to Make Bases (RMB) Addition</title>
5   <style>
6     body { font-family: sans-serif; }
7     #diagramRMBSVG { border: 1px solid #d3d3d3; } /* Style SVG like canvas */
8     #outputContainer { margin-top: 20px; }
9     .diagram-label { font-size: 14px; display: block; margin-bottom: 5px; } /*
      Improved label styling */
10  </style>
11 </head>
12 <body>
13
14  <h1>Addition Strategies: Rearranging to Make Bases (RMB)</h1>
15
16  <div>
17    <label for="addend1">Addend 1:</label>
18    <input type="number" id="addend1" value="18">
19  </div>

```

```

20 <div>
21   <label for="addend2">Addend 2:</label>
22   <input type="number" id="addend2" value="15">
23 </div>
24
25 <button onclick="runRMBAutomaton()">Calculate and Visualize</button>
26
27 <div id="outputContainer">
28   <h2>Explanation:</h2>
29   <div id="rmbOutput">
30     <!-- Text output will be displayed here -->
31   </div>
32 </div>
33
34 <h2>Diagram:</h2>
35 <svg id="diagramRMBSVG" width="600" height="700"></svg> <!-- Increased height -->
36
37 <script>
38 document.addEventListener('DOMContentLoaded', function() {
39   const rmbOutputElement = document.getElementById('rmbOutput');
40   const rmbAddend1Input = document.getElementById('addend1');
41   const rmbAddend2Input = document.getElementById('addend2');
42   const diagramRMBSVG = document.getElementById('diagramRMBSVG');
43
44   if (!rmbOutputElement || !diagramRMBSVG) {
45     console.warn("Element_rmbOutput_or_diagramRMBSVG_not_found");
46     return;
47   }
48
49   window.runRMBAutomaton = function() {
50     try {
51       const addend1 = parseInt(rmbAddend1Input.value);
52       const addend2 = parseInt(rmbAddend2Input.value);
53
54       if (isNaN(addend1) || isNaN(addend2)) {
55         rmbOutputElement.textContent = "Please_enter_valid_numbers_for_both_addends";
56         return;
57       }
58
59       let output = '';
60       output += '<h2>Rearranging to Make Bases (RMB)</h2><br><br>';
61       output += '<p><strong>Problem:</strong> ${addend1} + ${addend2}</p><br><br>';
62
63       const toMakeBase = (10 - (addend1 % 10)) % 10;
64
65       if (toMakeBase === 0) {
66         output += `${addend1} is already a multiple of 10.<br>';
67         output += 'Directly_add:${addend1}+${addend2}=${addend1+addend2}';
68         rmbOutputElement.textContent = output;
69         drawRMBDiagram('diagramRMBSVG', addend1, addend2, toMakeBase, addend1,
70           addend2, addend1 + addend2);
71         return;
72       }

```

```

72
73     if (addend2 < toMakeBase) {
74         output += 'Cannot make a base from ${addend1} because ${addend2} is too
75             small to provide the needed ${toMakeBase} units.<br>';
76         output += 'Directly add: ${addend1} + ${addend2} = ${addend1 + addend2}';
77         rmbOutputElement.textContent = output;
78         drawRMBDiagram('diagramRMBSVG', addend1, addend2, toMakeBase, addend1,
79             addend2, addend1 + addend2);
80         return;
81     }
82
83     // Apply RMB strategy
84     const newAddend1 = addend1 + toMakeBase;
85     const newAddend2 = addend2 - toMakeBase;
86     const result = newAddend1 + newAddend2;
87
88     output += 'Step 1: Move ${toMakeBase} from ${addend2} to ${addend1}<br>';
89     output += ' ${addend1} + ${toMakeBase} = ${newAddend1} (now a multiple of 10)<
90         br>';
91     output += ' ${addend2} - ${toMakeBase} = ${newAddend2}<br><br>';
92     output += 'Step 2: Add the rearranged numbers<br>';
93     output += '${newAddend1} + ${newAddend2} = ${result}<br><br>';
94     output += 'Result: ${addend1} + ${addend2} = ${result}';
95
96     rmbOutputElement.innerHTML = output;
97
98     // Draw RMB Diagram
99     drawRMBDiagram('diagramRMBSVG', addend1, addend2, toMakeBase, newAddend1,
100         newAddend2, result);
101
102 } catch (error) {
103     rmbOutputElement.textContent = 'Error: ${error.message}';
104 }
105 };
106
107 function drawRMBDiagram(svgId, addend1, addend2, toMakeBase, newAddend1, newAddend2,
108     result) {
109     const svg = document.getElementById(svgId);
110     if (!svg) return;
111     svg.innerHTML = ''; // Clear SVG
112
113     const svgWidth = parseFloat(svg.getAttribute('width'));
114     const svgHeight = parseFloat(svg.getAttribute('height'));
115     const blockUnitSize = 15; // Size of individual unit block
116     const tenBlockWidth = blockUnitSize; // Width of 10-block rectangle
117     const tenBlockHeight = blockUnitSize * 10; // Height of 10-block rectangle
118     const blockSpacing = 5;
119     const sectionSpacingY = 120; // Vertical spacing between sections
120     const startX = 50;
121     let currentY = 50;
122     const colors = ['lightblue', 'lightcoral']; // Colors for addend blocks

```

```

121 // --- Original Addends (Horizontal Layout) ---
122 createText(svg, startX, currentY, 'Original Addends: ${addend1} + ${addend2}'); //
    Label
123 currentY += 30; // Space after label
124
125 // Draw Addend 1 (purple) on left
126 let addend1X = startX;
127 const a1_tens = Math.floor(addend1 / 10);
128 const a1_ones = addend1 % 10;
129 for (let i = 0; i < a1_tens; i++) {
130     drawTenBlock(svg, addend1X, currentY, tenBlockWidth, tenBlockHeight, 'purple')
        ;
131     addend1X += tenBlockWidth + blockSpacing;
132 }
133 let a1_onesX = addend1X;
134 for (let i = 0; i < a1_ones; i++) {
135     drawBlock(svg, a1_onesX, currentY + i*(blockUnitSize + blockSpacing),
        blockUnitSize, blockUnitSize, 'purple');
136 }
137 const addend1Width = (a1_tens > 0 ? (a1_tens*(tenBlockWidth + blockSpacing)) : 0)
    + (a1_ones > 0 ? blockUnitSize : 0);
138
139 // Draw Addend 2 (blue) to the right of Addend 1
140 let addend2X = startX + addend1Width + 50; // 50px horizontal spacing between
    addend groups
141 const a2_tens = Math.floor(addend2 / 10);
142 const a2_ones = addend2 % 10;
143 for (let i = 0; i < a2_tens; i++) {
144     drawTenBlock(svg, addend2X, currentY, tenBlockWidth, tenBlockHeight, 'blue');
145     addend2X += tenBlockWidth + blockSpacing;
146 }
147 const addend2OnesX = addend2X;
148 let movedBlockTopY = null, movedBlockBottomY = null;
149 for (let i = 0; i < a2_ones; i++) {
150     drawBlock(svg, addend2OnesX, currentY + i*(blockUnitSize + blockSpacing),
        blockUnitSize, blockUnitSize, 'blue');
151     if (i < toMakeBase) {
152         if (movedBlockTopY === null) {
153             movedBlockTopY = currentY + i*(blockUnitSize + blockSpacing);
154         }
155         movedBlockBottomY = currentY + i*(blockUnitSize + blockSpacing) +
            blockUnitSize;
156     }
157 }
158 currentY += tenBlockHeight + sectionSpacingY; // Move down for the rearranged
    addends section
159
160 // --- Rearranged Addends ---
161 createText(svg, startX+20, currentY, 'Rearranged to Make Base: ${newAddend1} + ${
    newAddend2}'); // Label
162 currentY += 30; // Space after label
163
164 // Draw Rearranged Addend 1 Blocks (Tens only, since newAddend1 is a multiple of
    10)

```

```

165 let currentX_newAddend1 = startX;
166 const newAddend1_tens = Math.floor(newAddend1 / 10);
167 for (let i = 0; i < newAddend1_tens; i++) {
168     drawTenBlock(svg, currentX_newAddend1, currentY, tenBlockWidth,
169         tenBlockHeight, 'red');
170     currentX_newAddend1 += tenBlockWidth + blockSpacing;
171 }
172 // Draw Rearranged Addend 2 Blocks (Split into tens and ones)
173 const newAddend2_tens = Math.floor(newAddend2 / 10);
174 const newAddend2_ones = newAddend2 % 10;
175 let currentX_newAddend2 = currentX_newAddend1 + 40; // Horizontal spacing after
176     newAddend1 blocks
177 for (let i = 0; i < newAddend2_tens; i++) {
178     drawTenBlock(svg, currentX_newAddend2, currentY, tenBlockWidth, tenBlockHeight
179         , 'blue');
180     currentX_newAddend2 += tenBlockWidth + blockSpacing;
181 }
182 for (let i = 0; i < newAddend2_ones; i++) {
183     drawBlock(svg, currentX_newAddend2, currentY + i*(blockUnitSize + blockSpacing
184         ), blockUnitSize, blockUnitSize, 'blue');
185 }
186 // --- Curved Arrow ---
187 if (toMakeBase > 0 && addend2 >= toMakeBase && movedBlockTopY !== null) {
188     // Arrow from center of moved (vertical) ones in addend2 to the rearranged
189     tens block assembly
190     const arrowStartX = addend2OnesX + blockUnitSize/2;
191     const arrowStartY = movedBlockTopY + (movedBlockBottomY - movedBlockTopY) / 2;
192     const arrowEndX = startX + tenBlockWidth/2;
193     const arrowEndY = currentY; // top of rearranged addend1 blocks
194     // Use control point midway vertically between arrowStartY and arrowEndY
195     const controlY = (arrowStartY + arrowEndY) / 2;
196     createCurvedArrow(svg, arrowStartX, arrowStartY, arrowEndX, arrowEndY,
197         arrowEndX, controlY);
198     createText(svg, arrowEndX + 30, controlY + 35, `${toMakeBase} moved`);
199 }
200 // --- Helper SVG drawing functions ---
201 function drawBlock(svg, x, y, width, height, fill) {
202     const rect = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
203     rect.setAttribute('x', x);
204     rect.setAttribute('y', y);
205     rect.setAttribute('width', width);
206     rect.setAttribute('height', height);
207     rect.setAttribute('fill', fill);
208     rect.setAttribute('stroke', 'black');
209     rect.setAttribute('stroke-width', '1');
210     svg.appendChild(rect);
211 }
212 function drawTenBlock(svg, x, y, width, height, fill) {
213     const group = document.createElementNS("http://www.w3.org/2000/svg", 'g'); //
214         Group for 10-block

```

```

211     const backgroundRect = document.createElementNS("http://www.w3.org/2000/svg",
212         'rect');
213     backgroundRect.setAttribute('x', x);
214     backgroundRect.setAttribute('y', y);
215     backgroundRect.setAttribute('width', width);
216     backgroundRect.setAttribute('height', height);
217     backgroundRect.setAttribute('fill', fill);
218     backgroundRect.setAttribute('stroke', 'black');
219     backgroundRect.setAttribute('stroke-width', '1');
220     group.appendChild(backgroundRect);
221
222     // Draw 10 unit blocks inside - vertical column
223     for (let i = 0; i < 10; i++) {
224         const unitBlock = document.createElementNS("http://www.w3.org/2000/svg", '
225             rect');
226         unitBlock.setAttribute('x', x ); // Same x for vertical column
227         unitBlock.setAttribute('y', y + i * blockUnitSize); // Stacked vertically
228         unitBlock.setAttribute('width', blockUnitSize);
229         unitBlock.setAttribute('height', blockUnitSize);
230         unitBlock.setAttribute('fill', fill); // Same fill as outer rect
231         unitBlock.setAttribute('stroke', 'lightgrey'); // Lighter border for units
232         unitBlock.setAttribute('stroke-width', '0.5');
233         group.appendChild(unitBlock);
234     }
235     svg.appendChild(group);
236
237     function drawGroupRect(svg, x, y, width, height) {
238         const rect = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
239         rect.setAttribute('x', x);
240         rect.setAttribute('y', y);
241         rect.setAttribute('width', width);
242         rect.setAttribute('height', height);
243         rect.setAttribute('fill', 'none'); // No fill for group rect
244         rect.setAttribute('stroke', 'black');
245         rect.setAttribute('stroke-dasharray', '5,5'); // Dashed border for grouping
246         rect.setAttribute('stroke-width', '1');
247         svg.appendChild(rect);
248     }
249
250     function createText(svg, x, y, textContent) {
251         const text = document.createElementNS("http://www.w3.org/2000/svg", 'text');
252         text.setAttribute('x', x);
253         text.setAttribute('y', y);
254         text.setAttribute('class', 'diagram-label');
255         text.setAttribute('text-anchor', 'start');
256         text.setAttribute('font-size', '14px');
257         text.textContent = textContent;
258         svg.appendChild(text);
259     }
260
261     function createCurvedArrow(svg, x1, y1, x2, y2, cx, cy) {
262

```



```

263     const path = document.createElementNS("http://www.w3.org/2000/svg", 'path');
264     path.setAttribute('d', 'M ${x1} ${y1} Q ${cx} ${cy} ${x2} ${y2}');
265     path.setAttribute('fill', 'none');
266     path.setAttribute('stroke', 'black');
267     path.setAttribute('stroke-width', '2');
268     svg.appendChild(path);
269
270     // Arrowhead
271     const arrowHead = document.createElementNS("http://www.w3.org/2000/svg", 'path');
272     const arrowSize = 5;
273     arrowHead.setAttribute('d', 'M ${x2} ${y2} L ${x2 - arrowSize} ${y2 - arrowSize} L ${x2 + arrowSize} ${y2 - arrowSize} Z');
274     arrowHead.setAttribute('fill', 'black');
275     svg.appendChild(arrowHead);
276 }
277
278 }
279
280 });
281 </script>
282
283 </body>
284 </html>

```

References

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