Addition Strategies: Rearranging to Make Bases (RMB)

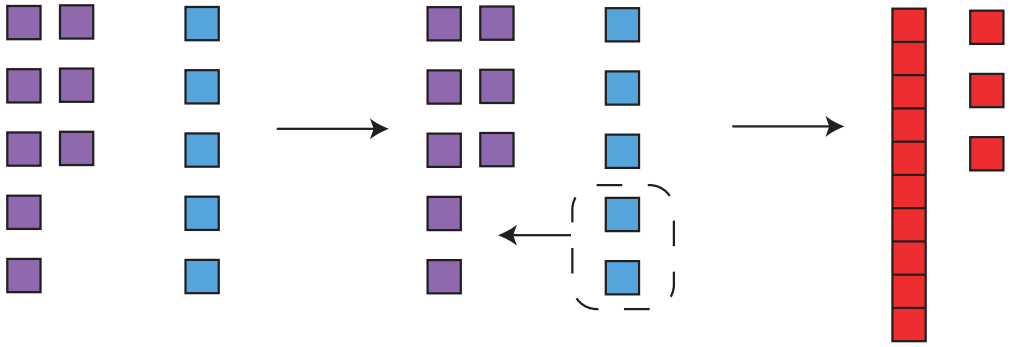
Compiled by: Theodore M. Savich

March 31, 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

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

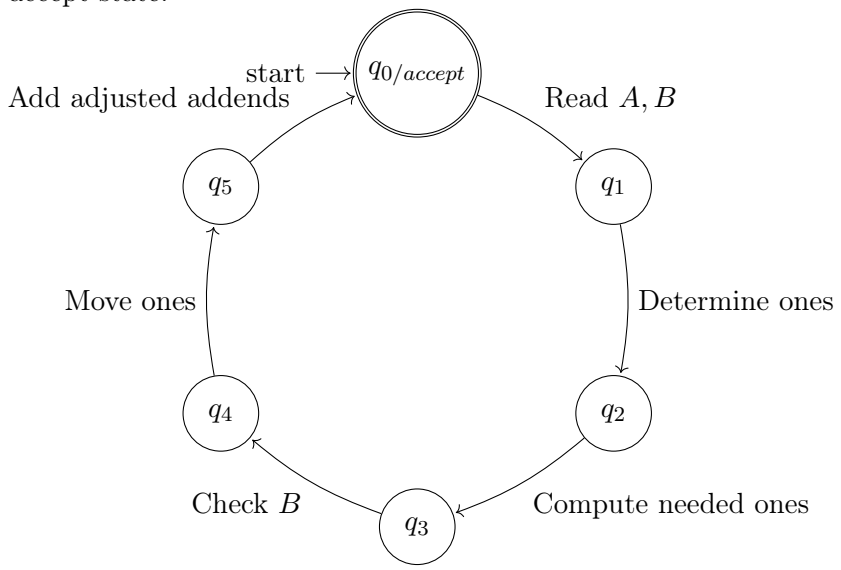
is defined by the following key transitions:

1. $\delta\Big(q_{0/accept}, \text{“}A, B\text{”}, Z_0\Big) = \{(q_1, Z_0)\}$ (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

```
<!DOCTYPE html>
<html>
<head>
    <title>Rearranging to Make Bases (RMB) Addition</title>
    <style>
        body { font-family: sans-serif; }
        #diagramRMBSVG { border: 1px solid #d3d3d3; } /* Style SVG like canvas */
        #outputContainer { margin-top: 20px; }
        .diagram-label { font-size: 14px; display: block; margin-bottom: 5px; } /*
            Improved label styling */
    </style>
</head>
<body>

    <h1>Addition Strategies: Rearranging to Make Bases (RMB)</h1>

    <div>
        <label for="addend1">Addend 1:</label>
        <input type="number" id="addend1" value="18">
    </div>
```

```
    <div>
        <label for="addend2">Addend 2:</label>
        <input type="number" id="addend2" value="15">
    </div>

    <button onclick="runRMBAutomaton()">Calculate and Visualize</button>

    <div id="outputContainer">
        <h2>Explanation:</h2>
        <div id="rmbOutput">
            <!-- Text output will be displayed here -->
        </div>
    </div>

    <h2>Diagram:</h2>
    <svg id="diagramRMBSVG" width="600" height="700"></svg> <!-- Increased height -->

    <script>
document.addEventListener('DOMContentLoaded', function() {
    const rmbOutputElement = document.getElementById('rmbOutput');
    const rmbAddend1Input = document.getElementById('addend1');
    const rmbAddend2Input = document.getElementById('addend2');
    const diagramRMBSVG = document.getElementById('diagramRMBSVG');

    if (!rmbOutputElement || !diagramRMBSVG) {
        console.warn("Element␣rmbOutput␣or␣diagramRMBSVG␣not␣found");
        return;
    }

    window.runRMBAutomaton = function() {
        try {
            const addend1 = parseInt(rmbAddend1Input.value);
            const addend2 = parseInt(rmbAddend2Input.value);

            if (isNaN(addend1) || isNaN(addend2)) {
                rmbOutputElement.textContent = "Please␣enter␣valid␣numbers␣for␣both␣addends
                    ";
                return;
            }

            let output = '';
            output += `<h2>Rearranging to Make Bases (RMB)</h2><br><br>`;
            output += `<p><strong>Problem:</strong> ${addend1} + ${addend2}</p><br><br>`;

            const toMakeBase = (10 - (addend1 % 10)) % 10;

            if (toMakeBase === 0) {
                output += `${addend1} is already a multiple of 10.<br>`;
                output += 'Directly␣add:␣${addend1}␣+␣${addend2}␣=␣${addend1␣+␣addend2}';
                rmbOutputElement.textContent = output;
                drawRMBDiagram('diagramRMBSVG', addend1, addend2, toMakeBase, addend1,
                    addend2, addend1 + addend2);
                return;
            }
```

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

```
        // --- Original Addends (Horizontal Layout) ---
        createText(svg, startX, currentY, `Original Addends: ${addend1} + ${addend2}`); // Label
        currentY += 30; // Space after label

        // Draw Addend 1 (purple) on left
        let addend1X = startX;
        const a1_tens = Math.floor(addend1 / 10);
        const a1_ones = addend1 % 10;
        for (let i = 0; i < a1_tens; i++) {
            drawTenBlock(svg, addend1X, currentY, tenBlockWidth, tenBlockHeight, 'purple');
            addend1X += tenBlockWidth + blockSpacing;
        }
        let a1_onesX = addend1X;
        for (let i = 0; i < a1_ones; i++) {
            drawBlock(svg, a1_onesX, currentY + i*(blockUnitSize + blockSpacing), blockUnitSize, blockUnitSize, 'purple');
        }
        const addend1Width = (a1_tens > 0 ? (a1_tens*(tenBlockWidth + blockSpacing)) : 0) + (a1_ones > 0 ? blockUnitSize : 0);

        // Draw Addend 2 (blue) to the right of Addend 1
        let addend2X = startX + addend1Width + 50; // 50px horizontal spacing between addend groups
        const a2_tens = Math.floor(addend2 / 10);
        const a2_ones = addend2 % 10;
        for (let i = 0; i < a2_tens; i++) {
            drawTenBlock(svg, addend2X, currentY, tenBlockWidth, tenBlockHeight, 'blue');
            addend2X += tenBlockWidth + blockSpacing;
        }
        const addend2OnesX = addend2X;
        let movedBlockTopY = null, movedBlockBottomY = null;
        for (let i = 0; i < a2_ones; i++) {
            drawBlock(svg, addend2OnesX, currentY + i*(blockUnitSize + blockSpacing), blockUnitSize, blockUnitSize, 'blue');
            if (i < toMakeBase) {
                if (movedBlockTopY === null) {
                    movedBlockTopY = currentY + i*(blockUnitSize + blockSpacing);
                }
                movedBlockBottomY = currentY + i*(blockUnitSize + blockSpacing) + blockUnitSize;
            }
        }
        currentY += tenBlockHeight + sectionSpacingY; // Move down for the rearranged addends section

        // --- Rearranged Addends ---
        createText(svg, startX+20, currentY, `Rearranged to Make Base: ${newAddend1} + ${newAddend2}`); // Label
        currentY += 30; // Space after label

        // Draw Rearranged Addend 1 Blocks (Tens only, since newAddend1 is a multiple of 10)
```

```
        let currentX_newAddend1 = startX;
        const newAddend1_tens = Math.floor(newAddend1 / 10);
        for (let i = 0; i < newAddend1_tens; i++) {
            drawTenBlock(svg, currentX_newAddend1, currentY, tenBlockWidth,
                tenBlockHeight, 'red');
            currentX_newAddend1 += tenBlockWidth + blockSpacing;
        }
        // Draw Rearranged Addend 2 Blocks (Split into tens and ones)
        const newAddend2_tens = Math.floor(newAddend2 / 10);
        const newAddend2_ones = newAddend2 % 10;
        let currentX_newAddend2 = currentX_newAddend1 + 40; // Horizontal spacing after
            newAddend1 blocks
        for (let i = 0; i < newAddend2_tens; i++) {
            drawTenBlock(svg, currentX_newAddend2, currentY, tenBlockWidth, tenBlockHeight
                , 'blue');
            currentX_newAddend2 += tenBlockWidth + blockSpacing;
        }
        for (let i = 0; i < newAddend2_ones; i++) {
            drawBlock(svg, currentX_newAddend2, currentY + i*(blockUnitSize + blockSpacing
                ), blockUnitSize, blockUnitSize, 'blue');
        }

        // --- Curved Arrow ---
        if (toMakeBase > 0 && addend2 >= toMakeBase && movedBlockTopY !== null) {
            // Arrow from center of moved (vertical) ones in addend2 to the rearranged
                tens block assembly
            const arrowStartX = addend2OnesX + blockUnitSize/2;
            const arrowStartY = movedBlockTopY + (movedBlockBottomY - movedBlockTopY) / 2;
            const arrowEndX = startX + tenBlockWidth/2;
            const arrowEndY = currentY; // top of rearranged addend1 blocks
            // Use control point midway vertically between arrowStartY and arrowEndY
            const controlY = (arrowStartY + arrowEndY) / 2;
            createCurvedArrow(svg, arrowStartX, arrowStartY, arrowEndX, arrowEndY,
                arrowEndX, controlY);
            createText(svg, arrowEndX + 30, controlY + 35, `${toMakeBase} moved`);
        }

        // --- Helper SVG drawing functions ---
        function drawBlock(svg, x, y, width, height, fill) {
            const rect = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
            rect.setAttribute('x', x);
            rect.setAttribute('y', y);
            rect.setAttribute('width', width);
            rect.setAttribute('height', height);
            rect.setAttribute('fill', fill);
            rect.setAttribute('stroke', 'black');
            rect.setAttribute('stroke-width', '1');
            svg.appendChild(rect);
        }

        function drawTenBlock(svg, x, y, width, height, fill) {
            const group = document.createElementNS("http://www.w3.org/2000/svg", 'g'); //
                Group for 10-block
```

```
        const backgroundRect = document.createElementNS("http://www.w3.org/2000/svg",
            'rect');
        backgroundRect.setAttribute('x', x);
        backgroundRect.setAttribute('y', y);
        backgroundRect.setAttribute('width', width);
        backgroundRect.setAttribute('height', height);
        backgroundRect.setAttribute('fill', fill);
        backgroundRect.setAttribute('stroke', 'black');
        backgroundRect.setAttribute('stroke-width', '1');
        group.appendChild(backgroundRect);

        // Draw 10 unit blocks inside - vertical column
        for (let i = 0; i < 10; i++) {
            const unitBlock = document.createElementNS("http://www.w3.org/2000/svg", '
                rect');
            unitBlock.setAttribute('x', x ); // Same x for vertical column
            unitBlock.setAttribute('y', y + i * blockUnitSize); // Stacked vertically
            unitBlock.setAttribute('width', blockUnitSize);
            unitBlock.setAttribute('height', blockUnitSize);
            unitBlock.setAttribute('fill', fill); // Same fill as outer rect
            unitBlock.setAttribute('stroke', 'lightgrey'); // Lighter border for units
            unitBlock.setAttribute('stroke-width', '0.5');
            group.appendChild(unitBlock);
        }
        svg.appendChild(group);
    }

    function drawGroupRect(svg, x, y, width, height) {
        const rect = document.createElementNS("http://www.w3.org/2000/svg", 'rect');
        rect.setAttribute('x', x);
        rect.setAttribute('y', y);
        rect.setAttribute('width', width);
        rect.setAttribute('height', height);
        rect.setAttribute('fill', 'none'); // No fill for group rect
        rect.setAttribute('stroke', 'black');
        rect.setAttribute('stroke-dasharray', '5␣5'); // Dashed border for grouping
        rect.setAttribute('stroke-width', '1');
        svg.appendChild(rect);
    }


    function createText(svg, x, y, textContent) {
        const text = document.createElementNS("http://www.w3.org/2000/svg", 'text');
        text.setAttribute('x', x);
        text.setAttribute('y', y);
        text.setAttribute('class', 'diagram-label');
        text.setAttribute('text-anchor', 'start');
        text.setAttribute('font-size', '14px');
        text.textContent = textContent;
        svg.appendChild(text);
    }


    function createCurvedArrow(svg, x1, y1, x2, y2, cx, cy) {
```

```
            const path = document.createElementNS("http://www.w3.org/2000/svg", 'path');
            path.setAttribute('d', `M ${x1} ${y1} Q ${cx} ${cy} ${x2} ${y2}`);
            path.setAttribute('fill', 'none');
            path.setAttribute('stroke', 'black');
            path.setAttribute('stroke-width', '2');
            svg.appendChild(path);

             // Arrowhead
            const arrowHead = document.createElementNS("http://www.w3.org/2000/svg", 'path
                ');
            const arrowSize = 5;
            arrowHead.setAttribute('d', `M ${x2} ${y2} L ${x2 - arrowSize} ${y2 -
                arrowSize} L ${x2 + arrowSize} ${y2 - arrowSize} Z`);
            arrowHead.setAttribute('fill', 'black');
            svg.appendChild(arrowHead);
        }

    }

});
    </script>

</body>
</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].