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

Compiled by: Theodore M. Savich

September 2, 2025

Transcript

Video from **Carpenter1999<empty citation>**. Strategy descriptions and examples adapted from **HackenbergCourseNotes<empty citation>**.

- **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.

images/Easy_Pictures/SAR_ADD_COBO/PDF/SAR_ADD_COBO.pdf

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: $76 \rightarrow 77 \rightarrow \ldots \rightarrow 83$.

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.

Automaton Definition SAR_ADD_COBO (Register Machine Model)

To legitimately and deterministically represent the COBO strategy, we define a Register Machine with clearly defined, mutually exclusive conditions.

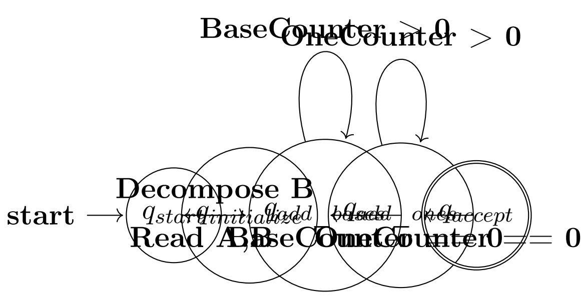
$\mathbf{M} = (\mathbf{Q}, \mathbf{V}, \delta, q_0, F)$

- **Q (States):** $q_{start}, q_{initialize}, q_{add\_bases}, q_{add\_ones}, q_{accept}$
- **V (Registers): A (Input), B (Input), Sum, BaseCounter, OneCounter**
- **Constants: BaseUnit (e.g., 10)**

Transition Function (δ) :

Current State	Condition	Next State	Action	Interpretation
q_{start}	**(Input)**	$q_{initialize}$	**Read A, Read B**	
$q_{initialize}$	**-**	q_{add_bases}	**Sum = ABaseCounter = B // BaseUnitOneCo**	
q_{add_bases}	**BaseCounter > 0**	q_{add_bases}	**Sum = Sum + BaseUnit BaseCounter =**	
q_{add_bases}	**BaseCounter == 0**	q_{add_ones}	**-**	
q_{add_ones}	**OneCounter > 0**	q_{add_ones}	**Sum = Sum + 1 OneCounter = One**	
q_{add_ones}	**OneCounter == 0**	q_{accept}	**Output Sum**	

Automaton Diagram for COBO



Python Implementation and Test

The following Python code implements the corrected COBO automaton.

```
import pandas as pd

class COBOAutomaton:
    """
␣␣␣␣A␣Register␣Machine␣model␣simulating␣the␣'Counting␣On␣By␣Bases␣and␣then␣Ones'␣(COBO)␣
    strategy.
␣␣␣␣"""
    def __init__(self, A, B, Base=10):
        self.A = A
        self.B = B
        self.BaseUnit = Base

        # Registers for internal computation
        self.Sum = 0
```

```python
        self.BaseCounter = 0
        self.OneCounter = 0

        # State
        self.state = 'q_start'
        self.history = []

    def _record_history(self, action, interpretation):
        self.history.append({
            'State': self.state,
            'Sum': self.Sum,
            'BaseCounter': self.BaseCounter,
            'OneCounter': self.OneCounter,
            'Action': action,
            'Interpretation': interpretation,
        })

    def transition(self, next_state):
        self.state = next_state

    def run(self):
        while self.state not in ['q_accept', 'q_error']:
            if self.state == 'q_start':
                self.execute_start()
            elif self.state == 'q_initialize':
                self.execute_initialize()
            elif self.state == 'q_add_bases':
                self.execute_add_bases()
            elif self.state == 'q_add_ones':
                self.execute_add_ones()
            else:
                self.transition('q_error')
                break
        return self.Sum

    def execute_start(self):
        """q_start: Read inputs."""
        self._record_history(f"Read A={self.A}, B={self.B}", "Start.")
        self.transition('q_initialize')

    def execute_initialize(self):
        """q_initialize: Initialize Sum and decompose B."""
        self.Sum = self.A
        # Decomposition (Assuming this skill is prerequisite for COBO)
        self.BaseCounter = self.B // self.BaseUnit
        self.OneCounter = self.B % self.BaseUnit

        action = f"Sum=A; Decompose B ({self.B})"
        interpretation = f"Initialize Sum to {self.A}. {self.BaseCounter} Bases, {self.OneCounter} Ones."
        self._record_history(action, interpretation)
        # Proceed to the base addition phase
        self.transition('q_add_bases')

```

```python
    def execute_add_bases(self):
        """q_add_bases: Iteratively add BaseUnits."""
        # Condition: BaseCounter > 0 (Loop Iteration)
        if self.BaseCounter > 0:
            prev_sum = self.Sum
            self.Sum += self.BaseUnit
            self.BaseCounter -= 1

            action = f"Sum += {self.BaseUnit}; BaseCounter -= 1"
            interpretation = f"Count on by base: {prev_sum} -> {self.Sum}."
            self._record_history(action, interpretation)
            # Stay in the same state
        # Condition: BaseCounter == 0 (Loop Exit)
        else:
            self._record_history("BaseCounter == 0", "All bases added. Transition to adding ones.")
            self.transition('q_add_ones')

    def execute_add_ones(self):
        """q_add_ones: Iteratively add Ones."""
        # Condition: OneCounter > 0 (Loop Iteration)
        if self.OneCounter > 0:
            prev_sum = self.Sum
            self.Sum += 1
            self.OneCounter -= 1

            action = "Sum += 1; OneCounter -= 1"
            interpretation = f"Count on by one: {prev_sum} -> {self.Sum}."
            self._record_history(action, interpretation)
            # Stay in the same state
        # Condition: OneCounter == 0 (Loop Exit)
        else:
            self._record_history("OneCounter == 0", "All ones added. Accept.")
            self.transition('q_accept')

    def display_history(self):
        print(f"\n--- COBO Execution History ({self.A} + {self.B}) ---")
        df = pd.DataFrame(self.history)
        print(df.to_markdown(index=False))

# Test the automaton with Lauren's example: 46 + 37.
cobo_automaton = COBOAutomaton(A=46, B=37)
result = cobo_automaton.run()
cobo_automaton.display_history()
print(f"\nFinal Result: {result}")
```

Execution Trace (46 + 37):

```
--- COBO Execution History (46 + 37) ---
| State          |   Sum |   BaseCounter |   OneCounter | Action                        | Interpr
|:---------------|------:|--------------:|-------------:|:------------------------------|:-------
| q_start        |     0 |             0 |            0 | Read A=46, B=37               | Start.
| q_initialize   |    46 |             3 |            7 | Sum=A; Decompose B (37)       | Initial
```

```
| q_add_bases    |    56 |              2 |              7 | Sum += 10; BaseCounter -= 1  | Count o
| q_add_bases    |    66 |              1 |              7 | Sum += 10; BaseCounter -= 1  | Count o
| q_add_bases    |    76 |              0 |              7 | Sum += 10; BaseCounter -= 1  | Count o
| q_add_bases    |    76 |              0 |              7 | BaseCounter == 0             | All bas
| q_add_ones     |    77 |              0 |              6 | Sum += 1; OneCounter -= 1    | Count o
| q_add_ones     |    78 |              0 |              5 | Sum += 1; OneCounter -= 1    | Count o
| q_add_ones     |    79 |              0 |              4 | Sum += 1; OneCounter -= 1    | Count o
| q_add_ones     |    80 |              0 |              3 | Sum += 1; OneCounter -= 1    | Count o
| q_add_ones     |    81 |              0 |              2 | Sum += 1; OneCounter -= 1    | Count o
| q_add_ones     |    82 |              0 |              1 | Sum += 1; OneCounter -= 1    | Count o
| q_add_ones     |    83 |              0 |              0 | Sum += 1; OneCounter -= 1    | Count o
| q_add_ones     |    83 |              0 |              0 | OneCounter == 0              | All one
```