

Here's a sample code implementation in Python:

```
def addBinary(a: str, b: str) -> str:
```

```
    # Initialize the result and carry
```

```
    result = []
```

```
    carry = 0
```

```
    # Reverse the strings to make addition easier
```

```
    a, b = a[::-1], b[::-1]
```

```
    # Iterate over the length of the longest string
```

```
    for i in range(max(len(a), len(b))):
```

```
        # Get the current bit from a and b, default to 0 if the string is shorter
```

```
        bit_a = int(a[i]) if i < len(a) else 0
```

```
        bit_b = int(b[i]) if i < len(b) else 0
```

```
        # Calculate the sum including the carry
```

```
        total = bit_a + bit_b + carry
```

```
        # Append the rightmost bit of the total to result
```

```
        result.append(str(total % 2))
```

```
        # Update the carry
```

```
        carry = total // 2
```

```
    # If there's still a carry left, append it
```

```
    if carry:
```

```
        result.append(str(carry))
```

```
    # Reverse the result to get the correct order and join it to form a string
```

```
    return ''.join(result[::-1])
```

```
# Example usage:
```

```
print(addBinary("11", "1"))    # Output: "100"
```

```
print(addBinary("1010", "1011")) # Output: "10101"
```

Explanation:

- The function addBinary takes two binary strings a and b.

- It reverses both strings for easier addition from least significant to most significant bit.
- It iterates through each bit, calculating the sum and managing carryovers.
- After looping through the bits, it checks for any remaining carry and appends it to the result.
- Finally, the list containing binary digits is reversed to restore the correct order and then joined into a string.