Mathematical Operations

- Base 2: the native base for computer systems
- Glyphs: 0, 1
- On a computer, we are limited to a certain number of digits.
- Recall, the results are summarized via the use of status flags.
  - For unsigned operations:
    - the final value is Zero (Z)
    - the calculation resulted in final carry (C)
  - For signed values
    - the final value is Negative (S)
    - the calculation resulted in an overflow (V)

Binary Addition:

- We have only two digits
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = ?
- What do we do in base 10
  - 9 + 9 = ?

			1	
				9
+				9
				8

Base 2

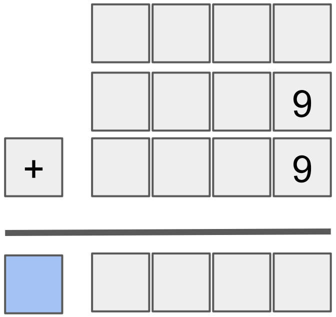
+		B	
		0	1
A	0	0	1
	1	1	?

Base 10

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Binary Addition (1-digit):

- We have only two digits
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = **10**
- What do we do in base 10
  - 9 + 9 = **18**



Base 2

+		B	
		0	**1**
A	**0**	00	01
	1	01	10

Half Adder

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Base 10

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

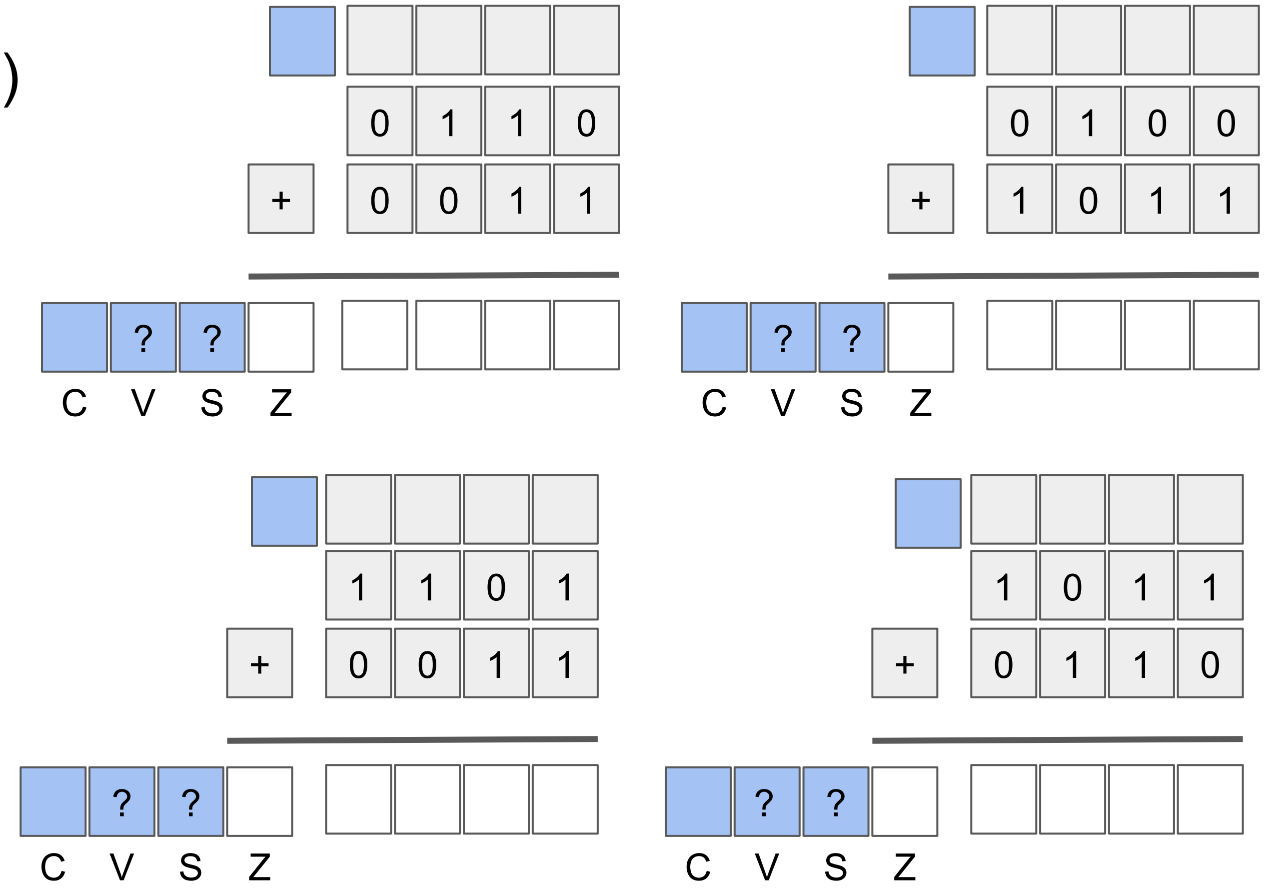
In Binary (before)

C + A + B = C, S

C	A	B	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0

C + A + B = C, S

C	A	B	C	S
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



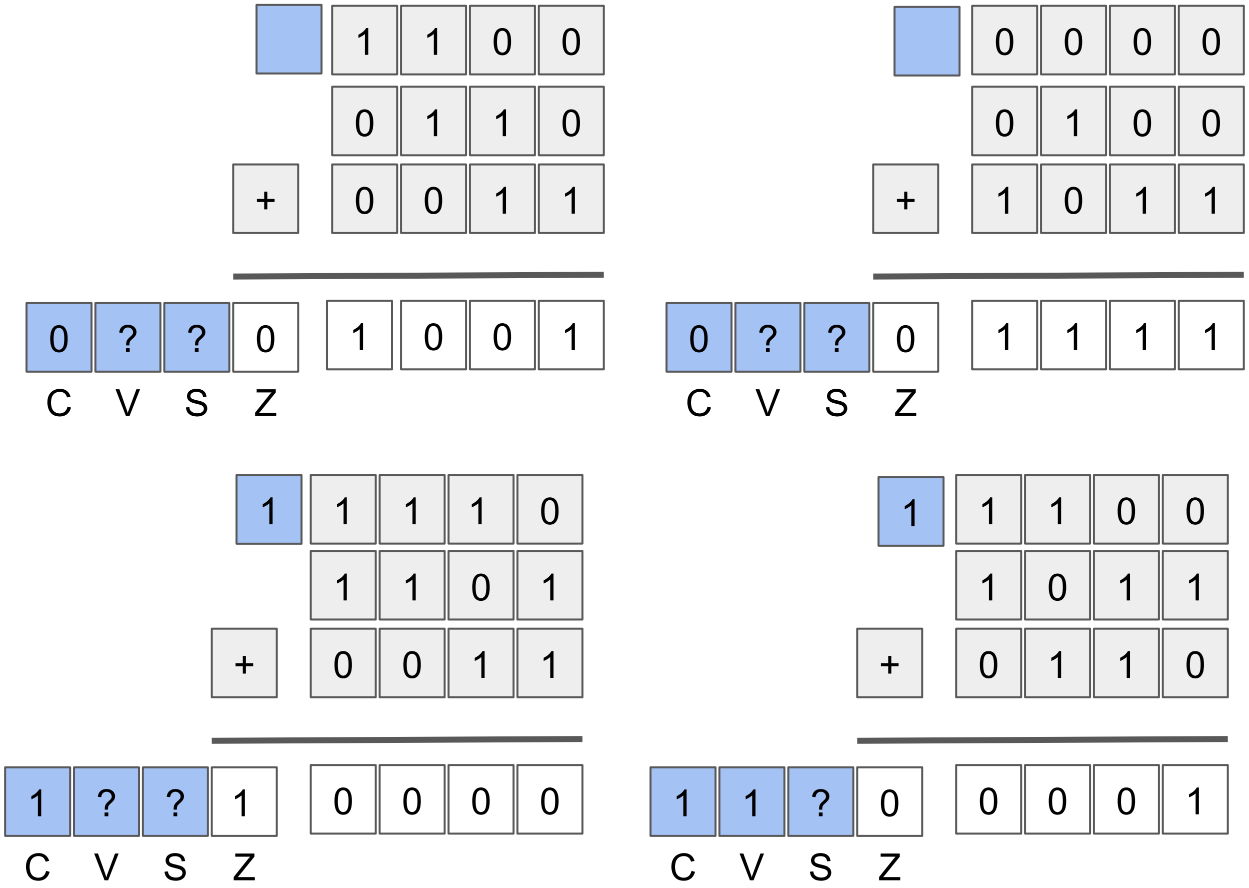
In Binary (after)

C + A + B = C, S

C	A	B	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0

C + A + B = C, S

C	A	B	C	S
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



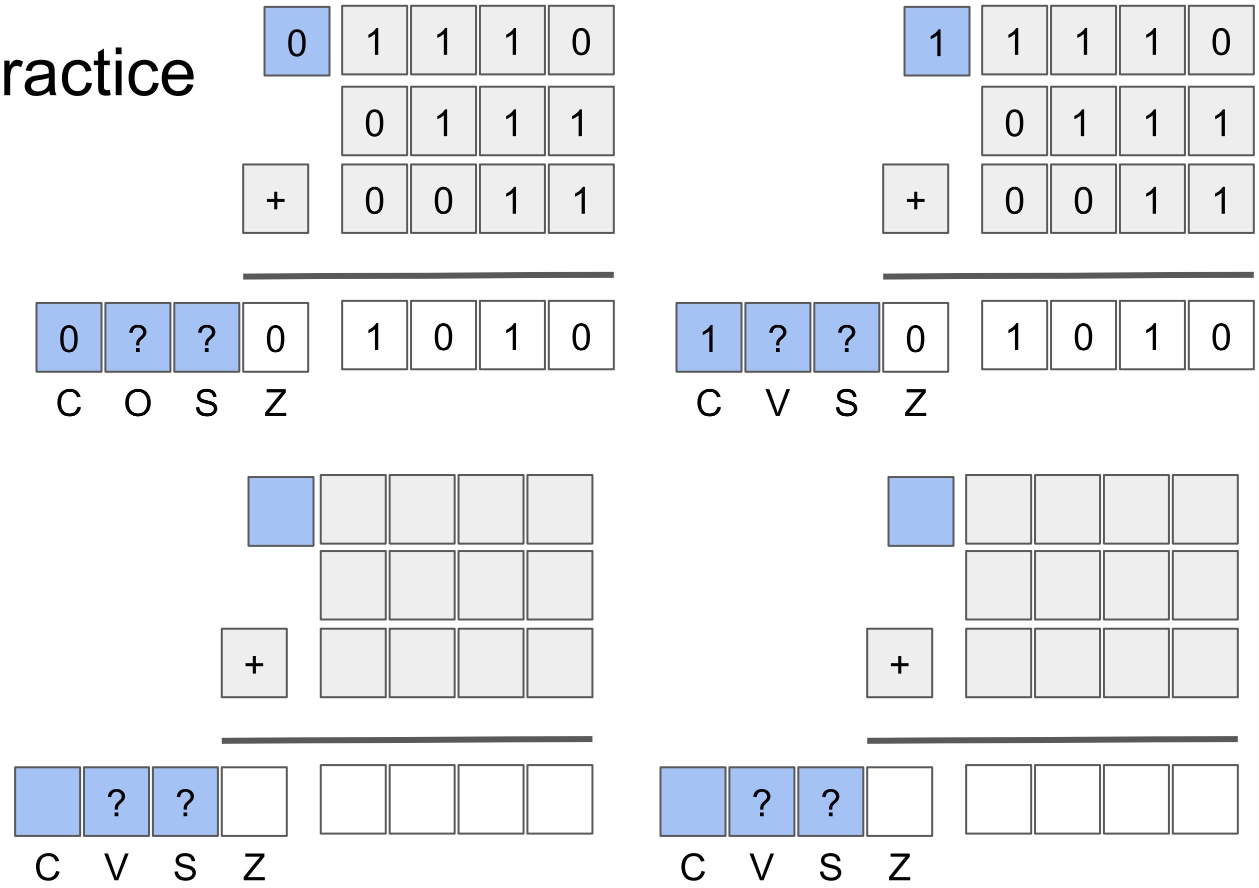
Binary Addition: Practice

C + A + B = C, S

C	A	B	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0

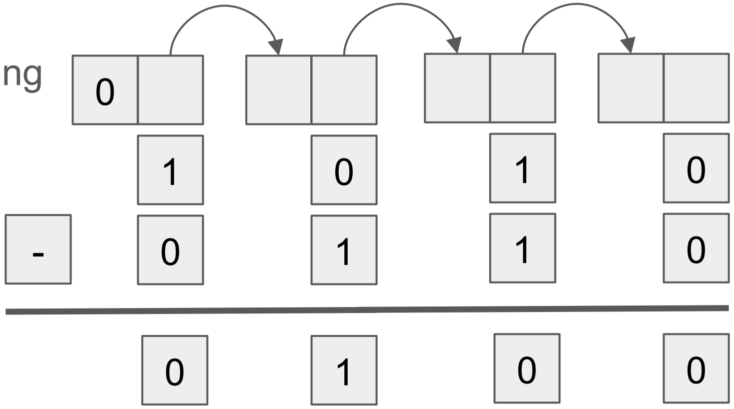
C + A + B = C, S

C	A	B	C	S
1	0	0	1	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Binary Subtraction (via Borrow)

- Traditional Method ⇒
  - Notice the extra squares
  - Notice the extra bookkeeping
  - 10 - 6 = 4



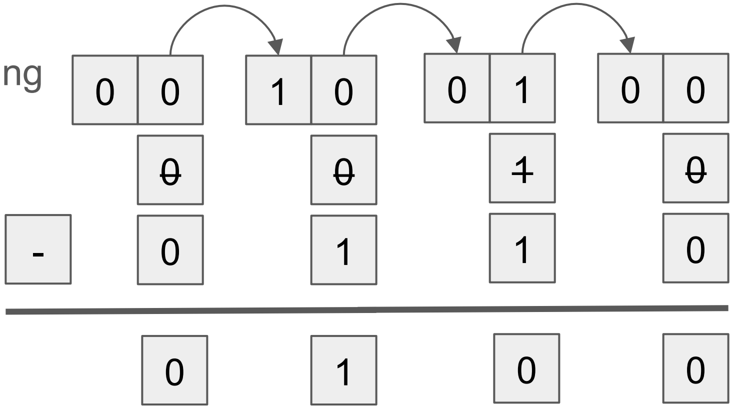
- Recall Method of Complements
  - allows us to leverage binary addition
  - need a method to encode negative numbers

A - B = V

A		B	V
0	0	0	0
0	0	1	x
0	1	0	1
0	1	1	0
1	0	0	x
1	0	1	1
1	1	0	x
1	1	1	x

Binary Subtraction (via Borrow)

- Traditional Method ⇒
  - Notice the extra squares
  - Notice the extra bookkeeping
  - 10 - 6 = 4



- Recall Method of Complements
  - allows us to leverage binary addition
  - need a method to encode negative numbers

A - B = V

A		B	V
0	0	0	0
0	0	1	x
0	1	0	1
0	1	1	0
1	0	0	x
1	0	1	1
1	1	0	x
1	1	1	x

Recall: Method of Complements

V	~V
0	1
1	0

- A technique to encode both positive and negative numbers
  - uses the same algorithm to perform addition
  - subtraction perform my addition of complements

- Complement: *a thing that completes or brings to perfection*
- Radix 10: *(the radix or base is the number of unique digits to represent a number)*
  - *10's complement*
    - 7 + x = 10 : x is the 10s complements of 7 x = 3
    - 46 + y = 100 : y is the 10s complements of 46 y = 54
  - 9's complement
    - 7 + a = 9 : a is the 9s complements of 7 a = 2
    - 46 + b = 99 : b is the 9s complements of 46 b = 53

- The math:

<u>2nd Grade</u>

$$\begin{array}{r} 45 \\ \underline{-\ 11} \\ 34 \end{array}$$

<u>10's complement</u>

$$\begin{array}{r} 45 \\ \underline{+\ 89} \\ \cancel{1}34 \end{array}$$

<u>9's complement</u>

$$\begin{array}{r} 45 \\ \underline{+\ 88} \\ \cancel{1}33 + 1 = \cancel{1}34 \end{array}$$

Method of Complements

A technique to encode both positive and negative numbers

- MSB used to denote the sign bit (0 positive, 1 negative)
- Table assumes a 4-bit represent

Use 1's complement to represent negative numbers

- Divide the number range in half
- Encode a positive and a negative value for each number
- Pros/cons:
  - ease to compute
  - positive and negative representations of zero

1's Complement

	Positive	Negative
0	0000	1111
1	0001	1110
2	0010	1101
3	0011	1100
4	0100	1011
5	0101	1010
6	0110	1001
7	0111	1000
8		

-1
-1
-1

Method of Complements

A technique to encode both positive and <u>negative</u> numbers

- MSB used to denote the sign bit (0 positive, 1 negative)
- Table assumes a 4-bit represent

Use 1's complement to represent negative numbers

- Divide the number range in half
- Encode a positive and a negative value for each number
- Pros/cons:
    - ease to compute
    - positive and negative representations of zero

Use 2's complete to represent negative numbers

- Hold Zero as special
- Fold the resulting range to assign values
- Pros/cons:
    - Not symmetric: extra negative number
    - Need to flip all bits and add one to form the negative number
    - Consider then the predecessor of -8: -8, -7, -6, … 0, 1, … 7

2's Complement

	Positive	Negative
0	0000	
1	0001	1110 + 1 = 1111
2	0010	1101 + 1 = 1110
3	0011	1100 + 1 = 1101
4	0100	1011 + 1 = 1100
5	0101	1010 + 1 = 1011
6	0110	1001 + 1 = 1010
7	0111	1000 + 1 = 1001
8		1000

-1
-1
-1
-1

Comparison of 1's and 2's Complement Encodings

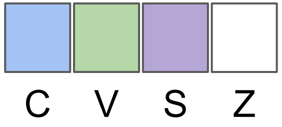
1's Complement

	Positive	Negative
0	0000	1111
1	0001	1110
2	0010	1101
3	0011	1100
4	0100	1011
5	0101	1010
6	0110	1001
7	0111	1000

2's Complement

	Positive	Negative
0	0000	
1	0001	1110 + 1 = 1111
2	0010	1101 + 1 = 1110
3	0011	1100 + 1 = 1101
4	0100	1011 + 1 = **1100**
5	0101	1010 + 1 = 1011
6	0110	1001 + 1 = 1010
7	0111	1000 + 1 = 1001
8		1000

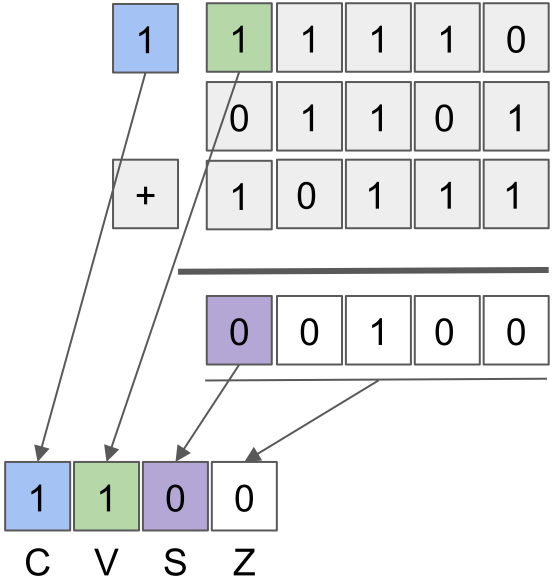
Status Flags Explained!



Example: 13 - 9 ⇒ 01101 + 10111

* 9: 01001 -- -9 : 10110 + 1 = 10111

- C: Carry Flag
  - the last step resulted in a carry value of 1
- V: Overflow Flag
  - the next to the last step resulted in a carry value of 1
- S: Sign Flag
  - the MSB in the result is set (i.e., a 1)
- Z: Zero Flag
  - all bits in the result are cleared (i.e., 0)

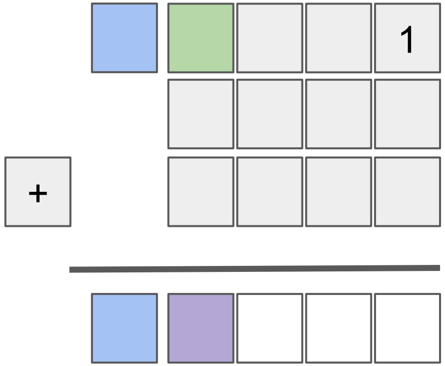


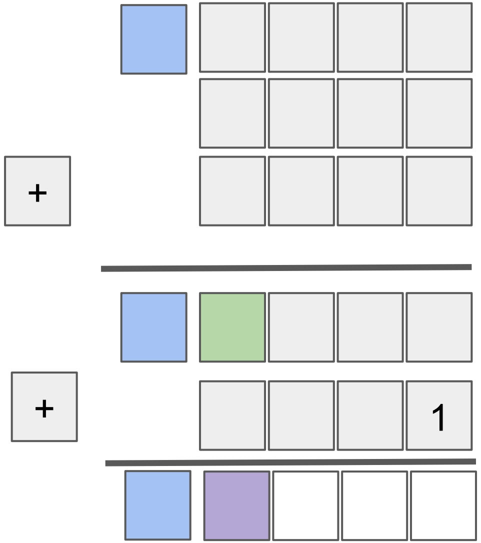
Algorithm: Subtraction via 1's Complements

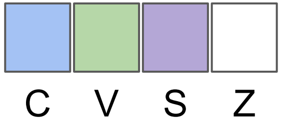
Example: 13 - 9 ⇒ 0013 + - 0009

1. Convert 13 and 9 into binary (01101 & 01001)
2. Take the **1's complement** of the subtrahend (9)
    - 01001 → 10110
3. Add the complement to the minuend
4. Drop the leading "1"
5. Add 1

- Optimization:
  introduce initial carry in



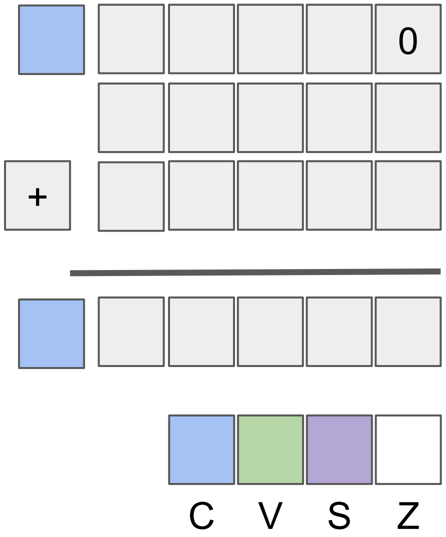




Algorithm: Subtraction via 2's Complement

Example: 13 - 9 ⇒ 00013 + - 0009

1. Encode 13 and 9 into binary (01101 & 01001)
2. Take the **2's complement** of the subtrahend (9)
   - 01001 → 10110 + 1 = 10111
3. Add the complement to the minuend
4. Drop the leading "1", i.e., the carry bit.



Providing the answer:

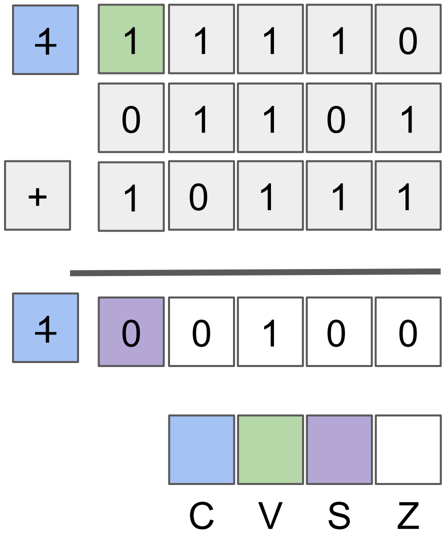
- ~~Optimization~~: Addition of adding one is baked in!

Algorithm: Subtraction via 2's Complement

Example: 13 - 9 ⇒ 0013 + - 0009

1. Encode 13 and 9 into binary (01101 & 01001)
2. Take the 2's complement of the subtrahend (9)
   - 01001 -> 10110 + 1 = 10111
3. Add the complement to the minuend
4. Drop the leading "1", i.e., the carry bit.

Providing the answer: 4



- ~~Optimization~~: Addition of adding one is baked in!

Practice: Addition and Subtraction

C	V	S	Z
1	1	1	0

Steps:

1. transform the operation to addition
2. convert abs(X) to binary
3. encode both numbers in 2's complement
4. perform binary addition
5. convert result to decimal
    - if negative result: compute 2's complement first

- `8 + 9 = 17`
- `7 - 4 = 3`
- `-5 + 2 = -3`
- `-16 - 3 = -19`
- `-25 - 4 = -29  => (-25) +  (-4)`

	1	1	1	1	0	0	0
		1	0	0	1	1	1
+		1	1	1	1	0	0
		1	0	0	0	1	1

Practice: Addition and Subtraction

C	V	S	Z

Steps:

1. transform the operation to addition
2. convert abs(X) to binary
3. encode both numbers in 2's complement
4. perform binary addition
5. convert result to decimal
    - if negative result: compute 2's complement first

	0	1	0	0	0	0
	0	0	1	0	0	0
+	0	0	1	0	0	1

- ➢ `8 + 9 = 17`
- `7 - 4 = 3`
- `-5 + 2 = -3`
- `-16 - 3 = -19`

```
2. 8: 001000, 9: 001001
3.
5. 17
```

Practice: Addition and Subtraction

C	V	S	Z

	1	1	1	1	0	0	0
		0	0	0	1	1	1
+		1	1	1	1	0	0
		0	0	0	0	1	1

Steps:

1. transform the operation to addition
2. convert abs(X) to binary
3. encode both numbers in 2's complement
4. perform binary addition
5. convert result to decimal
    - if negative result: compute 2's complement first

- 8 + 9 = 17
- ➢ 7 - 4 = 3
- -5 + 2 = -3
- -16 - 3 = -19

```
2. 7: 000111, 4: 000100
3. -4: 111011+1 ⇒ 111100
5. 3
```

Practice: Addition and Subtraction

C	V	S	Z

	0	0	0	0	1	0	0
		0	0	0	0	1	0
+		1	1	1	0	1	1
		1	1	1	1	0	1

Steps:

1. transform the operation to addition
2. convert abs(X) to binary
3. encode both numbers in 2's complement
4. perform binary addition
5. convert result to decimal
   - if negative result: compute 2's complement first

- 8 + 9 = 17
- 7 - 4 = 3
- ➢ -5 + 2 = -3
- -16 - 3 = -19

```
2.  5: 000101, 2: 000010
3. -5: 111010+1 ⇒ 111011
5. 000010+1 ⇒ 000011 = -3
```

Practice: Addition and Subtraction

C	V	S	Z

Steps:

1. transform the operation to addition
2. convert abs(X) to binary
3. encode both numbers in 2's complement
4. perform binary addition
5. convert result to decimal
    - if negative result: compute 2's complement first

	1	1	0	0	0	0	0
		1	1	0	0	0	0
+		1	1	1	1	0	1
		1	0	1	1	0	1

- 8 + 9 = 17
- 7 - 4 = 3
- -5 + 2 = -3
- ➢ -16 - 3 = -19

```
2.  16: 010000   3: 000011
3. -16: 110000  -3: 111101
5. 010010+1 ⇒ 010011 = -19
```

Recal: Algorithm: Multiplication

- Problem: 109 x 13 = 1417
- Approach: Successive Additions
  - Consider: 9 + 9 + 9 .. + 9 (13 times) = ?
  - What is carry value for the 10's column?
- Approach: Long Multiplication

Requires (at worst) 10^N additions

		11	7

```
  0 1 0 9
  0 1 0 9
  0 1 0 9
  0 1 0 9
  0 1 0 9
  0 1 0 9
  0 1 0 9
  0 1 0 9
  0 1 0 9
  0 1 0 9
  0 1 0 9
  0 1 0 9
+ 0 1 0 9
  ---------
  1 4 1 7
```

x13

Algorithm for Multiplication

```
sum = 0;
for (d = 0 ; d < 4 ; d ++ ) {
    sum += A * B[d];
    A = A * 10 ;  // Shift to the left
}


// B[0] = 9
// B[1] = 0
// B[2] = 1
```

reframe:

```
    013   (A)
*   109   (B)
-------
    117    (A*1)   *9
+   000    (A*10)  *0
-------
    117
  01300    (A*100)*1
-------
   1417
```

original:

```
    013   (A)
*   109   (B)
-------
    117    (A*9)*1
+   000    (A*0)*10
-------
    117
  01300    (A*1)*100
-------
   1417
```

Algorithm for Binary Multiplication

sum = 0;
for (d = 0 ; d < 4 ; d ++) {
    if (B[d] == 1) {
        sum += A ~~* B[d]~~;
    }
    ~~A = A * 2 ; // Shift to the left~~
    A << 1 ;
}

Requires *word_size* additions

original:

```
    010   (A = 2)
*   101   (B = 5)
-------
    010    (A*1)*1
+  0000    (A*0)*2
-------
   0010
  01000    (A*1)*4
-------
  01010    (A*B = 10)
```

reframe:

    010   (A = 2)
*   101   (B = 5)
----
    010   (A*1) *1
~~+ 0000   (A*2) *0~~

   0010
  01000   (A*4) *1

  01010   (A*B = 10)