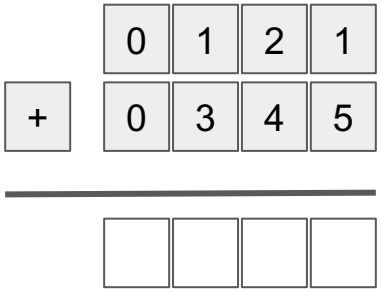
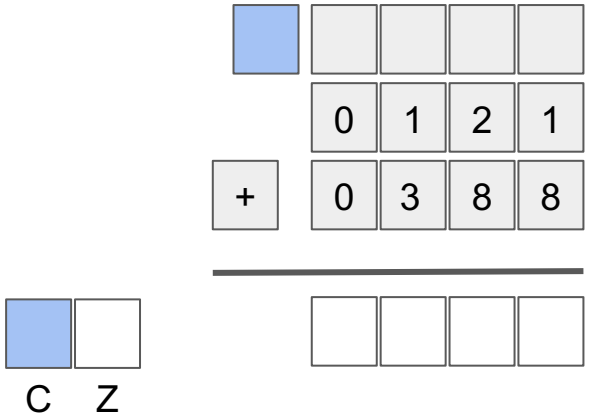
Mathematical Operations

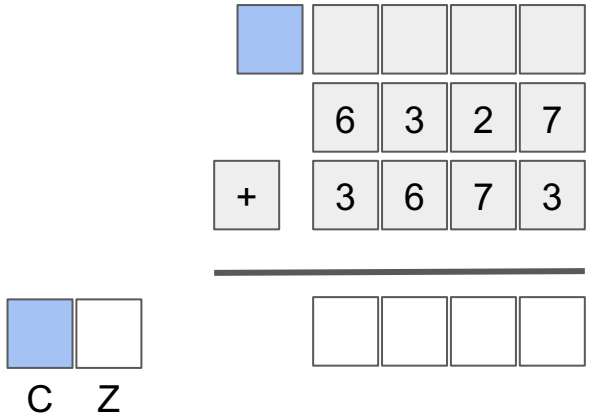
- Base 10: our native base.
- Glyphs: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- The algorithms to evaluate various functions are the same, regardless of base
- On a computer, we are limited to a certain number of digits.
- We can summarize our results: 0 == FALSE, 1 == TRUE
  - For unsigned operations:
    - the 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 ()

Addition: (Before)

- First, introduce some status values:
  - Zero, Carry, (Sign, Overflow)
- Assume a word size of 4
- Notice the notation of "to carry" a value

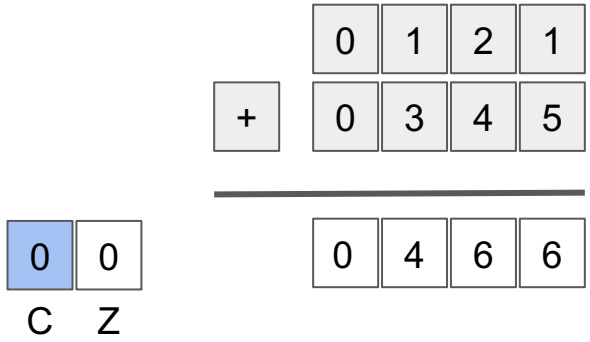


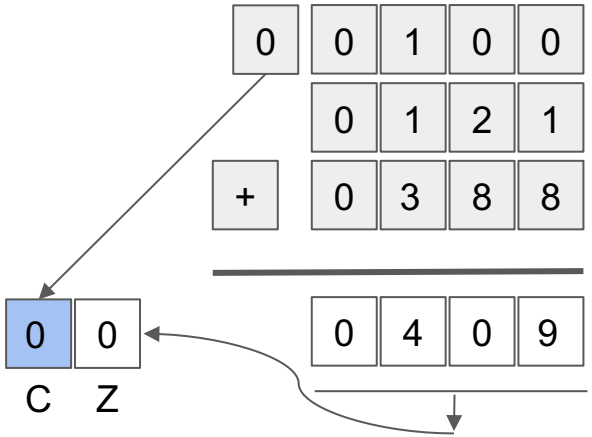


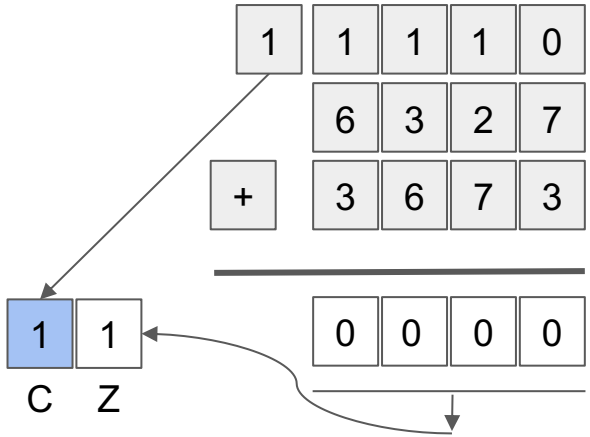


Addition: (After)

- First, introduce some status values:
  - Zero, Carry, (Sign, Overflow)
- Assume a word size of 4
- Notice the notation of "to carry" a value

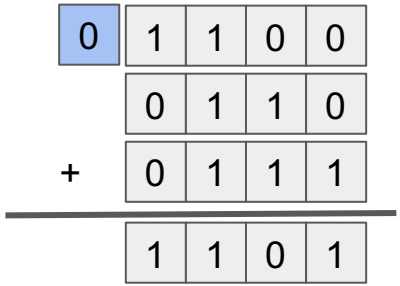




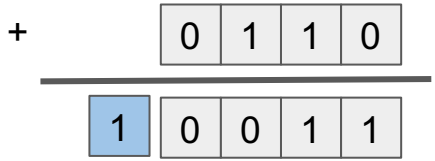


BCD: Addition

- Addition performed on the nibble level: 6+7



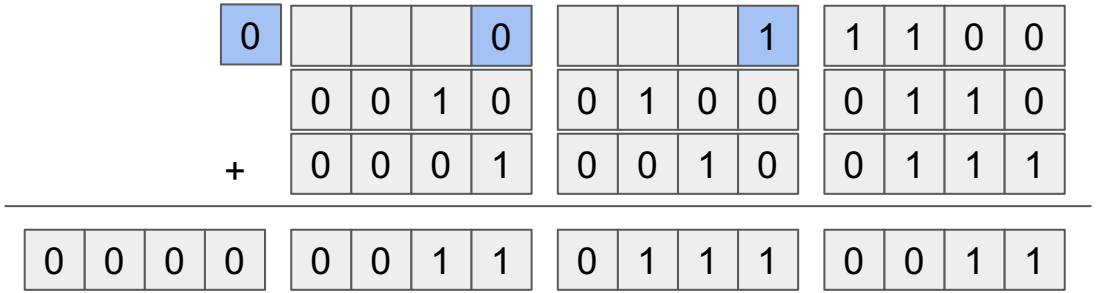
if (carry or invalid code) then



N	Code	N	Code
0	0000	8	1000
1	0001	9	1001
2	0010		1010
3	0011		1011
4	0100		1100
5	0101	x	1101
6	0110		1110
7	0111		1111

BCD: Binary Coded Decimal

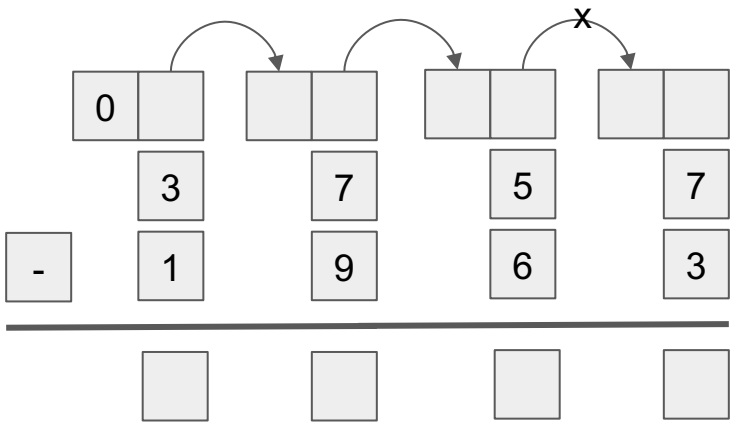
- Another encoding for numbers,
  where precision is required
- Four bits are used to encode each digit
- Perform addition per nibble
- Example: 246 + 127 = 373



N	Code	N	Code
0	0000	8	1000
1	0001	9	1001
2	0010		1010
3	0011		1011
4	0100		1100
5	0101		1101
6	0110		1110
7	0111		1111

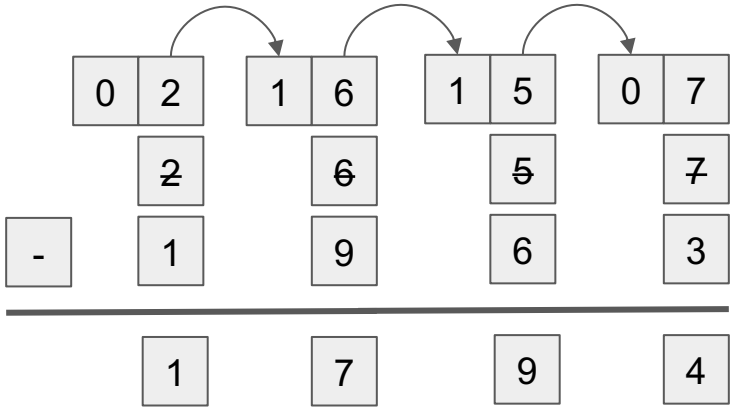
Subtraction (before)

- 3757 - 1963 = 1794
- Traditional Method:
  - Notice the notation of "to borrow" a value
- Other Methods: (common core)
  - Left → Right (Mental Math)
  - Singapore (No Borrow)
  - Counting Up (Giving Change)
- Via Method of Complements



Subtraction (after)

- 3757 - 1963 = 1794
- Traditional Method:
  - Notice the notation of "to borrow" a value
- Other Methods: (common core)
  - Left → Right (Mental Math)
  - Singapore (No Borrow)
  - Counting Up (Giving Change)
- Via Method of Complements



Method of Complements

- 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

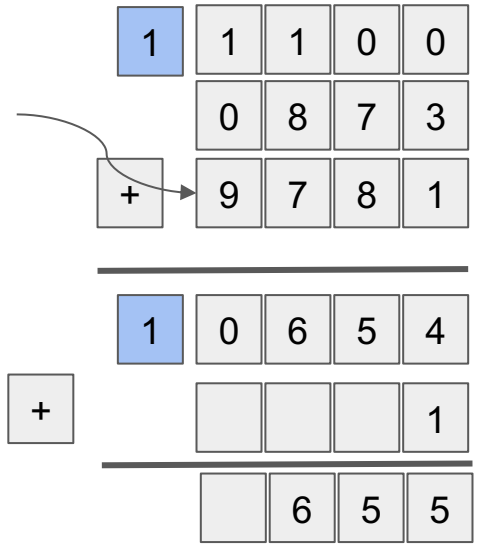
- The math:

2nd Grade	10's complement	9's complement
45 - 11 34	45 + 89 ~~1~~34	45 + 88 ~~1~~33 + 1 = 34

Algorithm: Subtraction via 9's Complements

- Example: 873 - 218 ⇒ 0873 - 0218

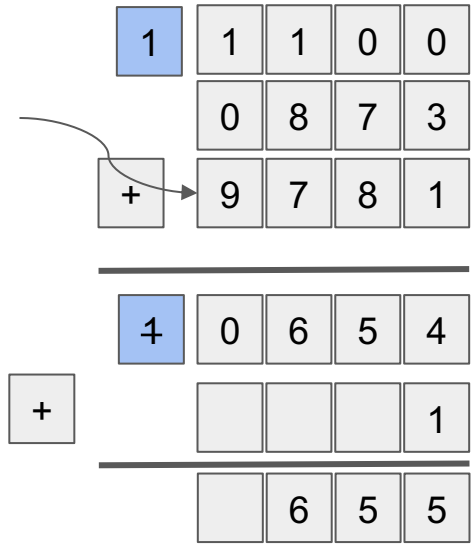
1. Take the nines complement of the subtrahend (0218)
2. Add the complement to the minuend (0873)
3. Drop the leading "1"
4. Add 1



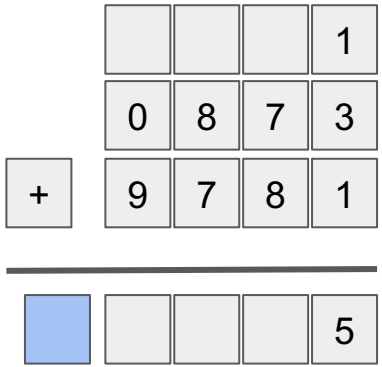
Algorithm: Subtraction via 9's Complements

- Example: 873 - 218 ⇒ 0873 - 0218

1. Take the nines complement of the subtrahend (0218)
2. Add the complement to the minuend (0873)
3. Drop the leading "1"
4. Add 1



- Optimization:
  introduce initial carry in

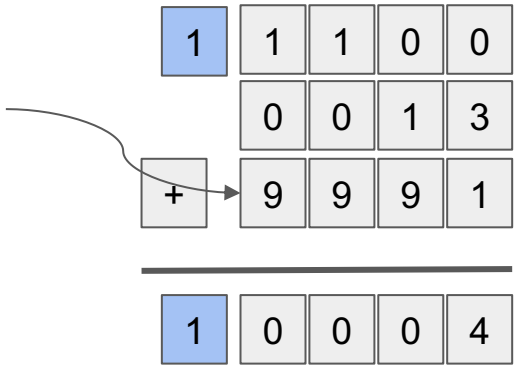


Algorithm: Subtraction via 10's Complements

- Example: 13 - 9 ⇒ 0013 - 0009

1. Take the 10s complement of the subtrahend (0009)
2. Add the complement to the minuend
3. Drop the leading "1".

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



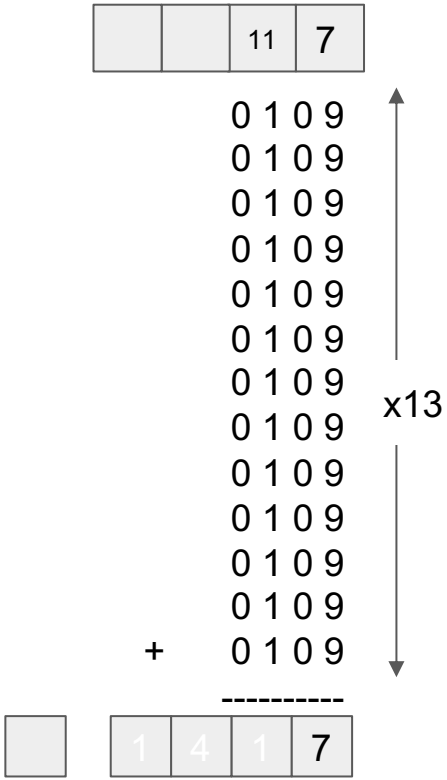
Multiplication

- 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

```
    013   (A)
*   109   (B)
-------
    117   (A*9)*1
   0000   (A*0)*10
+ 01300   (A*1)*100
-------
   1417
```

```
    013   (A)
*   109   (B)
-------
    117    (A*9)*1
+  0000    (A*0)*10
-------
    117
  01300    (A*1)*100
-------
   1417
```

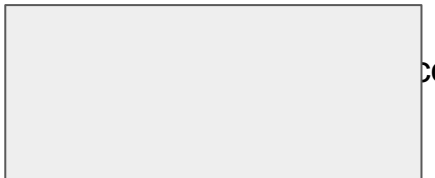
Requires (at worst) 10^N additions



Algorithm for Multiplication

```
sum = 0;
for (d = 0 ; d < 3 ; d ++ ) {
      sum += A * B[d];
     A = A * 10 ;  // Shift to the left
}
```

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



original:

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

reframe:

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

commutative operation

Algorithm for Binary Multiplication

```
sum = 0;
for (d = 0 ; d < 3 ; 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)
```