COMPSCI 210 S1T
Computer Systems

Data Representation

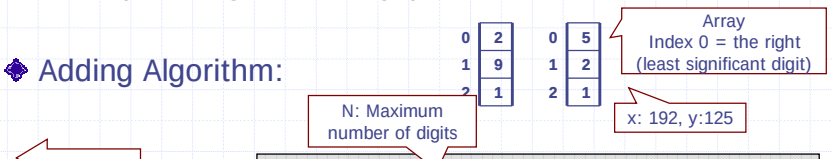
Arithmetic Operations

Agenda & Reading

- Agenda:
  - Arithmetic Operations
    - Addition
    - Subtraction
    - Comparison
    - Shifting
    - Multiplication
    - Division
- Java Example:
  - 02\NumOps.html

Addition

- We need a notion of a “carry” representing the excess from the computations of the previous digits
- Process the digits from “right” (least significant digit) to “left” (most significant digit)
- Adding Algorithm:




```
Carry = 0;
for (i = 0; i < N; i++)  {  // right to left scan
  z[i] = x[i] + y[i] + Carry;    // the add
  if (z[i] >= base)   {     // digit overflow!!
    Carry = 1;         // carry to next stage
    z[i] = z[i] - base;    // correct overflow
  } else
    Carry = 0;         // no carry to next stage
}
```

- **Base 10:**

  192
  + 125
  100 carries
  317

Addition

- Binary/Octal/Hexadecimal:

Base 2:	**Base 8:**	**Base 16:**
1010	237	13a
+ 0011	+ 162	+ 1b9
0100 carries	110 carries	010 carries
1101	421	2f3
Base 2:	Base 8:	Base 16:
01011111	363	F3
+ 00010001	+ 247	+ a7

Subtraction

- Subtraction is similar, except we have a “borrow” instead of the carry.
- We need a notion of a “borrow” of 0 or 1, initially 0, representing the deficit from the computations of the previous digits.
- Process the digits from “right” (least significant digit) to “left” (most significant digit).

- Subtract Algorithm:

- **Base 10:**

```
  416
- 263
  100 borrows
  153
```

```
Borrow = 0;
for (i = 0; i < N; i++) {
  z[i] = x[i] - y[i] - Borrow;
  if (z[i] < 0) {
    Borrow = 1;
    z[i] = z[i] + base;
  } else
    Borrow = 0;
}
```

x: 416, y:263

Subtraction

- Binary/Octal/Hexadecimal:
  - Base 2:

```
  0101
- 0011
  0100 borrows
  0010
```

  - Base 2:

```
  01011111
- 00010001
```

  - Base 8:

```
  237
- 160
  100 borrows
  57
```

  - Base 8:

```
  363
- 247
```

  - Base 16:

```
  139
-  ba
  110 borrows
  7f
```

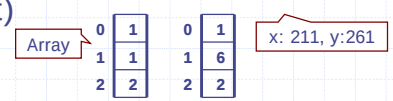
  - Base 16:

```
  F3
- a7
```

- Java Example:
  - NumOps.html

Comparison

- Compare two numbers and returns a value indicating whether one is less than, equal to or greater than the other
- Compare(a,b)
  - Returns Less than zero -> a is less than b.
  - Returns Zero -> a equals b.
  - Returns Greater than zero -> a is greater than b
- Process the digits from “left” (most significant digit) to “right” (least significant digit)
- Algorithm:



From left to right

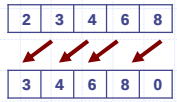
```
for ( int i = N - 1; i >= 0; --i )
  if ( a[ i ] != b[ i ] )
    return a[ i ] - b[ i ];
return 0;
```

- **Base 10:**

```
  211
  261
```


Shifting Left

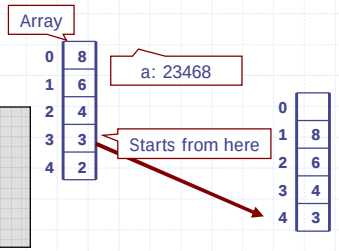
- Shifting left by one digit
  - A fill digit to insert: 0
  - A digit is lost



- Process the digits from “left” (most significant digit) to “right” (least significant digit)
- Algorithm:

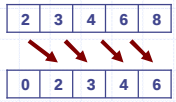
From left to right

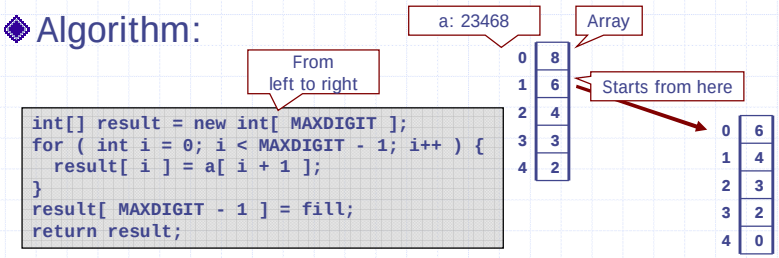
```
int[] result = new int[ N ];
for ( int i = N - 1; i > 0; --i ) {
  result[ i ] = a[ i - 1 ];
 }
result[ 0 ] = fill;
return result;
```



Shifting Right

- Shifting right by one digit
  - A fill digit to insert: 0
  - A digit is lost
- Process the digits from "right" (least significant digit) to "left" (most significant digit)
- Algorithm:



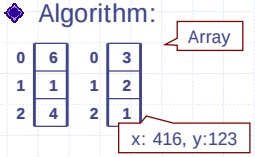



```
int[] result = new int[ MAXDIGIT ];
for ( int i = 0; i < MAXDIGIT - 1; i++ ) {
  result[ i ] = a[ i + 1 ];
}
result[ MAXDIGIT - 1 ] = fill;
return result;
```


Multiplication

- Multiple two numbers and returns the result
- May require a multiplication table
- Process the digits from "left" (most significant digit) to "right" (least significant digit)
- Note: If we have a limited number of digits we can represent, we may lose the top digit(s)
- Algorithm:

```
int[] product = new int[ N ];
int[][] multTable = timesTable( a );
for ( int i = N - 1; i >= 0; --i ) {
  product = shiftLeft( product, 0 );
  product = add( product, multTable[ b[ i ] ] );
}
return product;
```



	416*digit	Trimmed
0	0	000
1	416	416
2	832	832
3	1248	248
4	1664	664
	...	

i	Product (after shifting)	416*digit	Trimmed Result (product)
2	0	416	0+416=416
1	416(lsh)=160	832	160+823=992
0	992(lsh)=920	248	920+248=1168=168

```
  416
*  123
 1248
 8320
41600
51168
```


Example: Hex, Octal

- Steps are almost the same. But when you do the addition, you only have change carry to 1 if the sum exceeds the base.
- Example: Hex: base=16

i	Product (after shifting)		416*digit	Trimmed Result (product)
2	0	1	416	0+416=416
1	416(lsh)=160	2	82C	160+82C=98C
0	98C(lsh)=8C0	3	C42	98C+C42=1502=502

```
  416
*  123
  C42
 82C0
41600
4A502
```

	416*digit	Trimmed
0	0	000
1	416	416
2	82C	82C
3	C42	C42
	...	

- Example: Oct: Base = 8

i	Product (after shifting)	416*digit	Trimmed Result (product)
2	0	416	0+416=416
1	416(lsh)=160	034	160+034=214
0	214(lsh)=140	452	160+452=612

```
  416
*  123
  452
 1034 0
41600
53612
```

	416*digit	Trimmed
0	0	000
1	416	416
2	1034	034
3	1452	452
	...	

Example: Binary

- Steps are almost the same. But when you do the addition, you only have change carry to 1 if the sum exceeds the base.
- Example: Binary: base=2

i	Product (after shifting)	d	416*digit	Trimmed Result (product)
2	0	0	0	0+0=0
1	0(lsh)=0	1	110	0+110=110
0	110(lsh)=100	0	0	100+0=100

```
  110
*  010
  000
 1100
00000
01100
```

x: 416, y:12

	110*digit	Trimmed
0	0	0
1	110	110

- Exercise: 11010 * 01100 (binary)

Division

- Divide one number (the dividend) by another (the divisor) to get a quotient and residue.
- Process the digits from "left" (most significant digit) to "right" (least significant digit)
- Algorithm:
  - Shift the next digit of the dividend into the right of residue,
  - Divide the residue by the divisor to get the next digit of quotient and a new residue

i	Dividend Digit	Residue	Lsh R + d[i]	Quotient digit	New Residue
2	4	0	4+0=4	0	4
1	1	4	40+1=41	3	41-36=5
0	6	5	50+6=56	4	56-48=8

0	6
1	1
2	4

```
  034_
12|416
  0
  41
  36
   56
   48
    8
```

Quotient=34
Residue = 8

Example: Hex, Octal

- Steps are almost the same. But the base is different. Check carefully on addition/subtraction
- Example: Hex: base=16

416/12 (Hex)

i	Dividend Digit	Residue	Lsh R + d[i]	Quotient digit	New Residue
2	4	0	0+4=4	0	4
1	1	4	40+1=41	3	41-36=B
0	6	B	B0+6=B6	A	B6-B4=2

Quotient=3A
Residue = 2

```
  03A_
12|416
  0
  41
  36
   B6
   B4
    2
```

	12*digit
0	0
1	12
2	24
3	36
4	48
5	5A
6	6C
7	7E
8	90
9	A2
A	B4
B	C6

- Example: Oct: Base = 8

416/12 (Octal)

i	Dividend Digit	Residue	Lsh R + d[i]	Quotient digit	New Residue
2	4	0	0+4=4	0	4
1	1	4	40+1=41	3	41-36=3
0	6	3	30+6=36	3	36-36=0

Quotient=33
Residue = 0

```
  033_
12|416
  0
  41
  36
   36
   36
    0
```

	12*digit
0	0
1	12
2	24
3	36
4	50
5	62
6	74
7	106

Example: Binary

- Steps are almost the same. But when you do the addition, you only have change carry to 1 if the sum exceeds the base.
- Example: Binary: base=2

101/11 (Binary)

i	Dividend Digit	Residue	Lsh R + d[i]	Quotient digit	New Residue
2	1	0	0+1=1	0	1
1	0	1	10+0=10	0	10
0	1	10	100+1=101	1	101-11=10

Quotient=11
Residue = 10

```
  001_
11|101
  0
  10
   0
  101
   11
   10
```

- Exercise: 110/11 (binary)