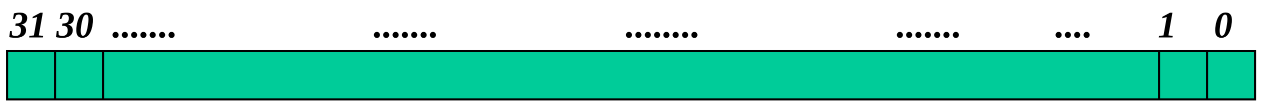
Binary arithmetic

Mohammed Alaa Elkomy

Integers are represented as binary vectors

Suppose each word consists of 32 bits, labeled 0…31.



MSB (most significant bit)

LSB (least)

Value of the binary vector interpreted as unsigned integer is:

$$v\,(b) = b_{31}.2^{31} + b_{30}.2^{30} + b_{29}.2^{29} + ... + b_{1}.2^{1} + b_{0}.2^{0}$$

More generally in N bits,

$$v(b) = \sum_{n=0}^{n=N-1} b_n 2^n$$

We need to represent both positive and negative integers. Three schemes are available for representing both positive and negative integers:

- Sign and magnitude.
- 1's complement.
- 2's complement.

All schemes use the Most Significant Bit (MSB) to carry the sign information:

If MSB = 0, bit vector represents a positive integer.

If MSB = 1, bit vector represents a negative integer

Range of numbers that can be represented in N bits

Unsigned:

$$0 < V(b) < 2^N - 1$$

Sign and magnitude:

$$-2^{N-1} - 1 < V(b) < 2^{N-1} - 1$$

0 has both positive and negative representation

One's complement:

$$-2^{N-1} - 1 < V(b) < 2^{N-1} - 1$$

0 has both positive and negative representation

Two's complement:

$$-2^{N-1} < V(b) < 2^{N-1} - 1$$

0 has a single representation, easier to add/subtract.

Addition and Subtraction

To add two numbers:

a) Add their n-bit representations.
b) Ignore the carry out from MSB position.
c) Sum is the algebraically correct value in the 2's complement representation as long as the answer is in the range -2^{n-1} through $+2^{n-1} -1$.

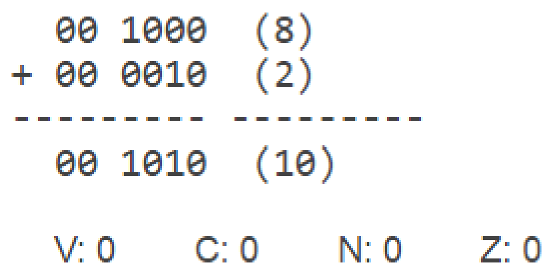
To subtract two numbers *X* and *Y* (*X*-*Y*):

a) Form the 2's complement of *Y*.
b) Add it to *X* using Rule 1.
c) Result is correct as long as the answer lies in the range -2^{n-1} through $+2^{n-1} -1$.

When adding unsigned numbers, carry-out from the MSB position serves as the overflow indicator.
When adding signed numbers, this does not work.

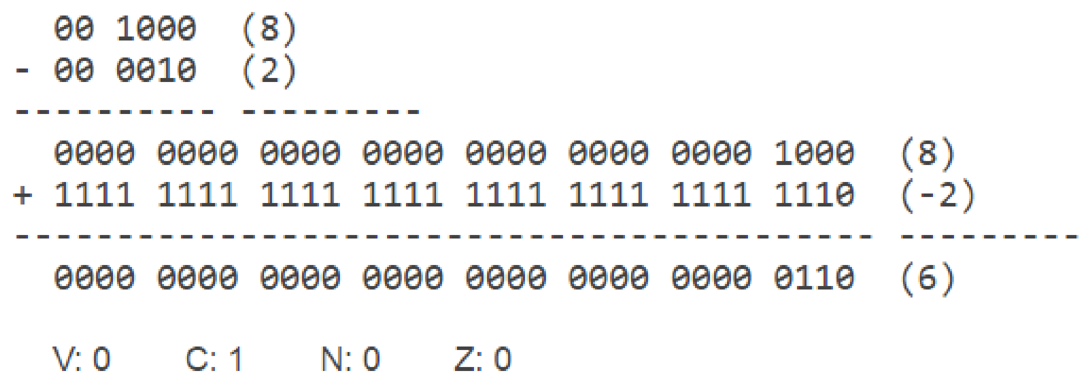
Overflow occurs when both the numbers have the same sign.
Addition of numbers with different signs cannot cause an overflow.

Addition Example



```
  00 1000   (8)
+ 00 0010   (2)
--------- ---------
  00 1010   (10)

  V: 0     C: 0     N: 0     Z: 0
```

Subtraction Example



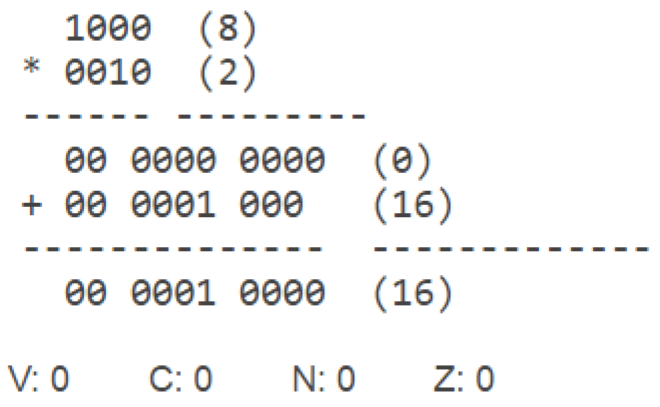
```
  00 1000   (8)
- 00 0010   (2)
---------- ---------
  0000 0000 0000 0000 0000 0000 0000 1000   (8)
+ 1111 1111 1111 1111 1111 1111 1111 1110   (-2)
----------------------------------------- ---------
  0000 0000 0000 0000 0000 0000 0000 0110   (6)

  V: 0     C: 1     N: 0     Z: 0
```

Multiplication

Normal multiplication

a) If the i^{th} bit of the multiplier is 1, shift the multiplicand and add the shifted multiplicand to the current value of the partial product.
b) Hand over the partial product to the next stage
c) Value of the partial product at the start stage is 0.

Example



```
  1000  (8)
* 0010  (2)
------ ---------
  00 0000 0000  (0)
+ 00 0001 000   (16)
--------------  -------------
  00 0001 0000  (16)

V: 0     C: 0     N: 0     Z: 0
```

Booth’s algorithm

For every bit in Q:

a). Examine the bit and its neighbor to the immediate right.

If the bit pair is:

00 – do nothing.

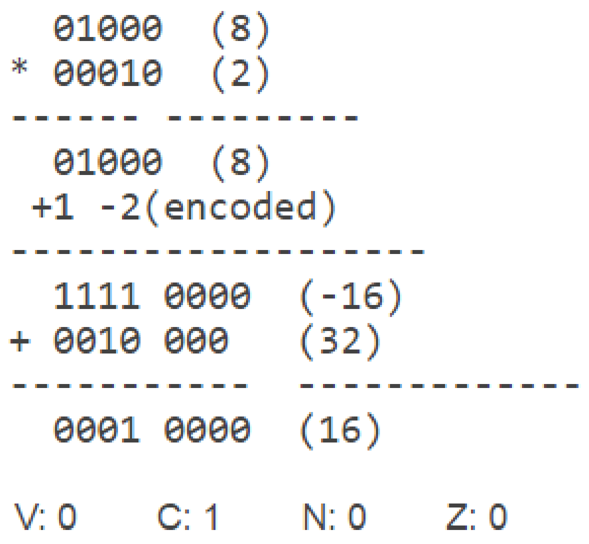
01 – Add the multiplicand to the upper half of PP.

10 – Sub the multiplicand from the upper half of PP.

11 – Do nothing.

b). Shift the PP right by one bit, extending the sign bit.

Example

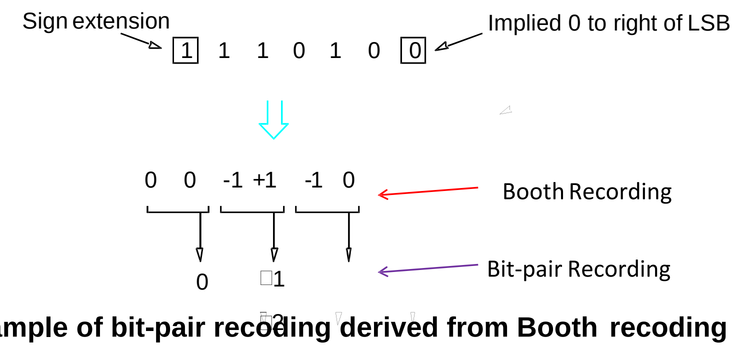


```
  01000  (8)
* 00010  (2)
------ ---------
  01000  (8)
 +1 -2(encoded)
-------------------
  1111 0000  (-16)
+ 0010 000   (32)
-----------  -------------
  0001 0000  (16)

V: 0     C: 1     N: 0     Z: 0
```

Bit-Pair Recoding of Multipliers

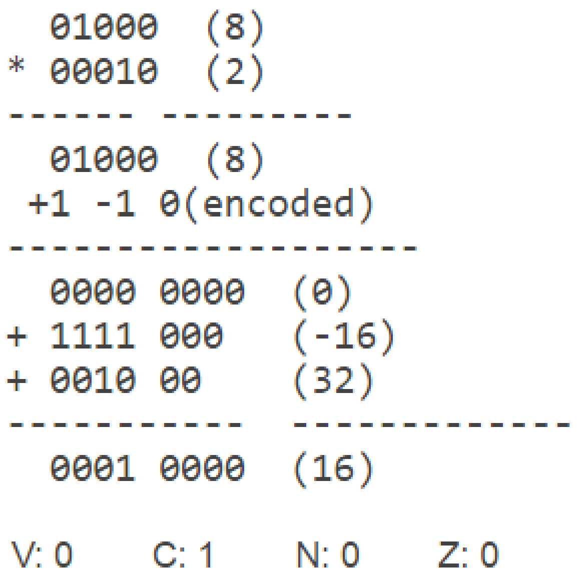
a) For each pair of bits in the multiplier, we require at most one summand to be added to the partial product

b) For *n*-bit operands, it is *guaranteed* that the max. number of summands to be added is *n*/2



Example of bit-pair recoding derived from Booth recoding

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at positioin
$i+1$	i	$i-1$	
0	0	0	0 X M
0	0	1	+1 X M
0	1	0	+1 X M
0	1	1	+2 X M
1	0	0	−2 X M
1	0	1	−1 X M
1	1	0	-1 X M
1	1	1	0 X M



```
  01000  (8)
* 00010  (2)
------ ---------
  01000  (8)
 +1 -1 0(encoded)
-------------------
  0000 0000  (0)
+ 1111 000   (-16)
+ 0010 00    (32)
----------- -------------
  0001 0000  (16)

V: 0      C: 1      N: 0      Z: 0
```

Unsigned Division

Restoring division

a) Shift the dividend one bit at a time starting from MSB into a register.
b) Subtract the divisor from this register.
c) If the result is negative ("didn't go"):
   - Add the divisor back into the register.
   - Record 0 into the result register.
d) If the result is positive:
   - Do not restore the intermediate result.
   - Set a 1 into the result register.

Example

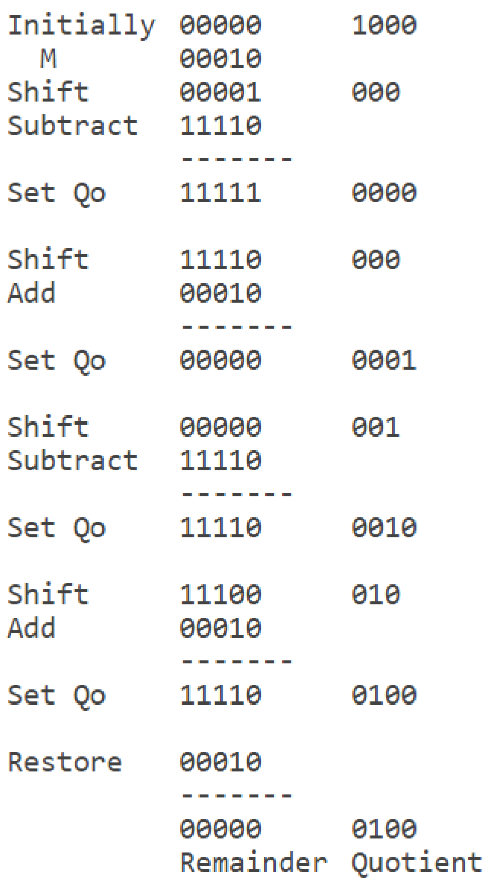
```
Initially 00000     1000
  M       00010
Shift     00001     000
Subtract  11110
          -------
Set Qo    11111
Restore   00010
          -------
          00001     0000
Shift     00010     000
Subtract  11110
          -------
Set Qo    00000
          00000     0001
Shift     00000     001
Subtract  11110
          -------
Set Qo    11110
Restore   00010
          -------
          00000     0010
Shift     00000     010
Subtract  11110
          -------
Set Qo    11110
Restore   00010
          -------
          00000     0100
          Remainder Quotient
```

Restoring division is only concerned with unsigned integers.

Restoring division

a) Shift the dividend one bit at a time starting from MSB into a register.
b) Subtract the divisor from this register.
c) If the result is negative (“didn’t go”):
   - Add the divisor back into the register.
   - Record 0 into the result register.
d) If the result is positive:
   - Do not restore the intermediate result.
   - Set a 1 into the result register.

Example



```
Initially  00000     1000
  M        00010
Shift      00001     000
Subtract   11110
           -------
Set Qo     11111     0000

Shift      11110     000
Add        00010
           -------
Set Qo     00000     0001

Shift      00000     001
Subtract   11110
           -------
Set Qo     11110     0010

Shift      11100     010
Add        00010
           -------
Set Qo     11110     0100

Restore    00010
           -------
           00000     0100
           Remainder Quotient
```

Non-Restoring division is only concerned with unsigned integers.