Integer Binary Multiplication and Division

It's simply repeated addition or repeated subtraction coupled with some shifting and control. The goal is to simplify the task to make it suitable for hardware

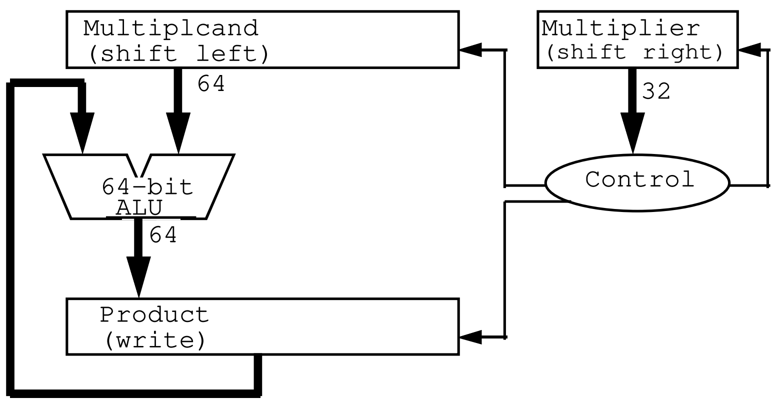
Terminology for * and /

- Multiplying the multiplicand and multiplier produce the product
- Dividing the divisor into the divident produces the quotient and remainder

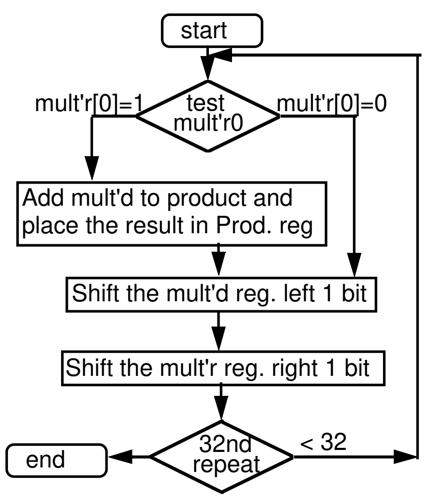
```
Multiplicand 1000                               101 Quotient
Multiplier   0110              Divisor 110|11111 Dividend
             0000                          -110
            1000                              11
           1000                             -000
          0000                               111
Product   0110000                           -110
                                               1 Remainder
```

A Naive Multiplier Design

- Operands of 32 bits yield a product of 64 bits
- Basic registers and ALU components are used



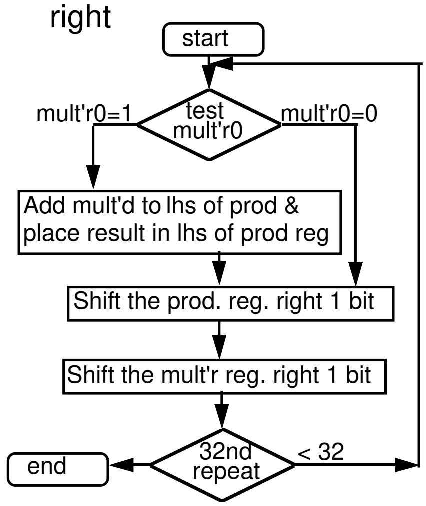
Naive Multiplication



Mult’r	Mult’d	Product
0101	0000 0110	0000 0000
0101	0000 0110	0000 0110
0101	0000 1100	0000 0110
0010	0000 1100	0000 0110
0010	0000 1100	0000 0110
0010	0001 1000	0000 0110
0001	0001 1000	0000 0110
0001	0001 1000	0001 1110
0001	0011 0000	0001 1110
0000	0011 0000	0001 1110
0000	0011 0000	0001 1110
0000	0110 0000	0001 1110
0000	0110 0000	0001 1110

Improved Algorithm

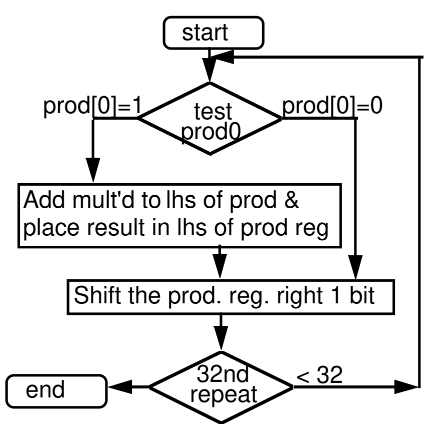
Add multiplicand into hi half of product register & shif right



Mult'r	Mult'd	Product
0101	0110	0000 0000
0101	0110	0110 0000
0101	0110	0011 0000
0010	0110	0011 0000
0010	0110	0011 0000
0010	0110	0001 1000
0001	0110	0001 1000
0001	0110	0111 1000
0001	0110	0011 1100
0000	0110	0011 1100
0000	0110	0011 1100
0000	0110	0001 1110
0000	0110	0001 1110

Final Algorithm

- Place multiplier in low side of product register



Mult'd	Product
0110	0000 0101
0110	0110 0101
0110	0011 0010
0110	0011 0010
0110	0001 1001
0110	0111 1001
0110	0011 1100
0110	0011 1100
0110	0001 1110

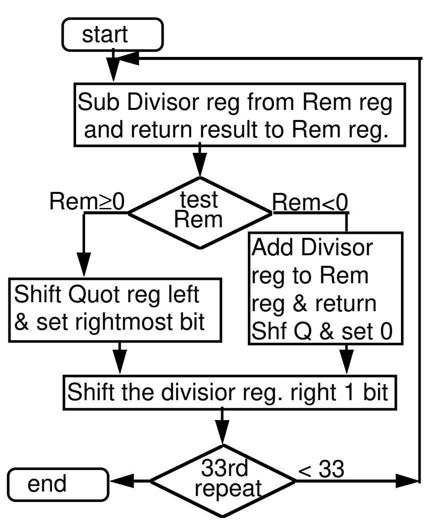
Division

The basic idea is to subtract repeatedly, but unlike humans who can recognize when subtraction is/is not worthwhile, the circuitry has to do it, and then correct it if it was wrong.

```
                    0
                  1/1 Quotient
Divisor 110|11111 Dividend
           -110
              11
            -110
            ****
              111
             -110
                1 Remainder
```

Why can't computers figure this out???

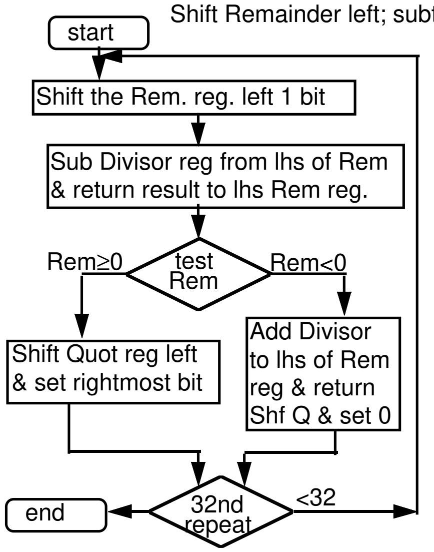
Basic Algorithm



Quo	Divisor	Remainder
0000	0010 0000	0000 0111
0000	0010 0000	1110 0111
0000	0010 0000	0000 0111
0000	0001 0000	0000 0111
0000	0001 0000	1111 0111
0000	0001 0000	0000 0111
0000	0000 1000	0000 0111
0000	0000 1000	1111 1111
0000	0000 1000	0000 0111
0000	0000 0100	0000 0111
0000	0000 0100	0000 0011
0001	0000 0100	0000 0011
0001	0000 0010	0000 0011
0001	0000 0010	0000 0001
0011	0000 0010	0000 0001
0011	0000 0001	0000 0001

Improved Algorithm

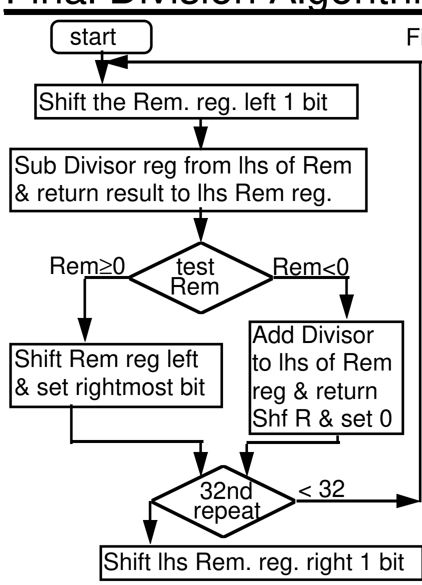
Shift Remainder left; subtract divisior from left.



Quo	Div'r	Remainder
0000	0010	0000 0111
0000	0010	0000 1110
0000	0010	1110 1110
0000	0010	0000 1110
0000	0010	0001 1100
0000	0010	1111 1100
0000	0010	0001 1100
0000	0010	0011 1000
0000	0010	0001 1000
0001	0010	0001 1000
0001	0010	0011 0000
0001	0010	0001 0000
0011	0010	0001 0000

Final Division Algorithm

Fill rhs of Remainder with Quotient.



Div'r	Remainder
0010	0000 0111
0010	0000 1110
0010	1110 1110
0010	0001 1100
0010	1111 1100
0010	0011 1000
0010	0001 1000
0010	0011 0001
0010	0001 0001
0010	0010 0011
0010	0001 0011

Wrap Up

- Signed multiplication/division can be done by converting to positive numbers and then fixing sign of result -- remainder has sign of dividend
- Multiplication and Divide can be performed by the same hardware
- Lo contains product or quotient; Hi contains overflow or remainder