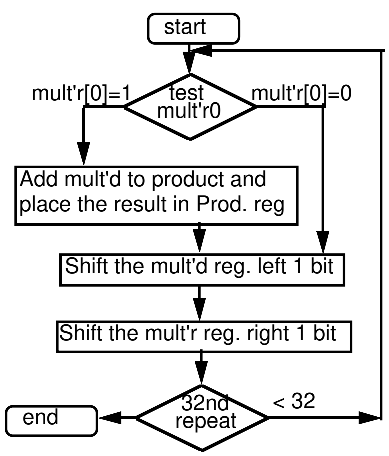
Multiplication/Division

Its simply repeated addition or repeated subtraction coupled with some shifting and control. The goal is to simplify the hardware components.

```
Multiplicand  1000
Multiplier    0110
              0000
             1000
            1000
           0000
Product    0110000
```

```
               101 Quotient
Divisor 110 | 11111  Dividend
             -110
               11
               111
              -110
                 1  Remainder
```

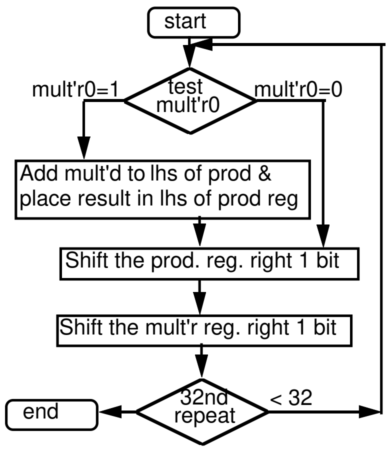

Naive Multiplication Algorithm



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

Second Algorithm

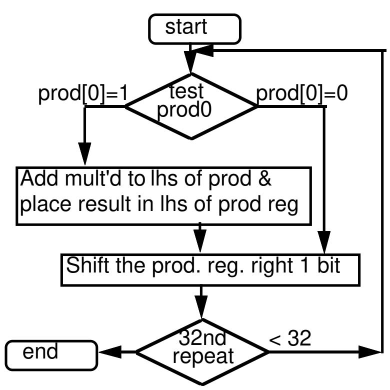
Add multiplicand into high order half of product register and then shift it right.



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 right half of product.



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

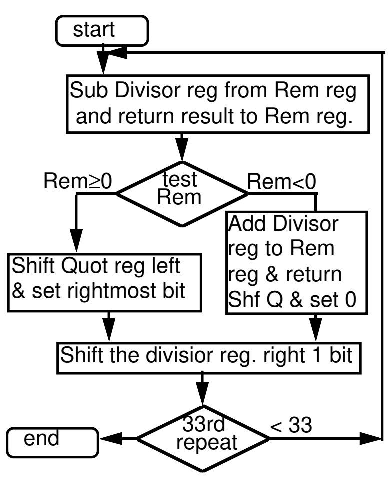
Division

The principle parts of are Dividend, Divisor, Quotient and Remainder.

The basic idea is to subtract repeatedly, but unlike humans who can recognize when subtraction is worthwhile, the circuitry has to do it, and then correct it if it was wrong.

```
                101 Quotient
    Divisor 110 |11111  Dividend
               -110
                  11
                 111
                -110
                   1  Remainder
```

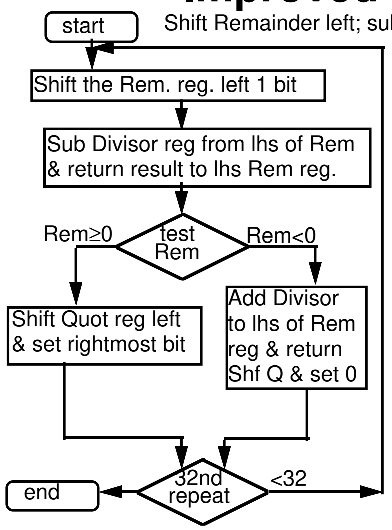

Basic Algorithm



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

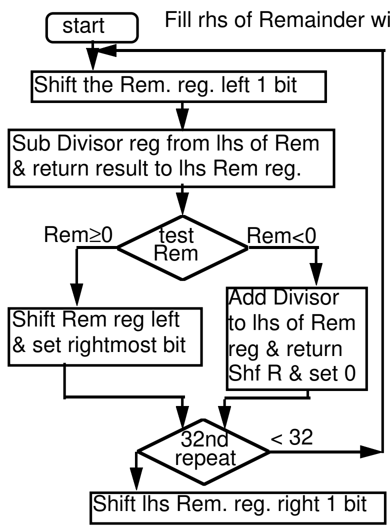
Shift Remainder left; subtract divisior from left.



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 negating the result if signs differ.

-- The remainder has the sign of the dividend.

The same hardware can perform both multiplicaton & division

MIPS stores results in two 32-bit registers: Hi and Lo.

MIPS instructions are R-type, but do not use "rd".

```
mult $8, $9      div $8, $9
multu $8, $9     divu $8, $9
```

Lo contains product or quotient; Hi contains overflow or remainder.

```
mflo   $8   # load $8 from Lo
mfhi   $8   # load $8 from Hi
```