Arbitrary Precision Algorithm of Online Multiplier

Initialization:

Online Delay: delta;

Unrolling width: U;

Precision: P;

Outer Iterator: n, from 0 to N = floor(P/U);

Inter Iterator: u, from 0 to U-1;

Input: digit xu, digit yu;

Output: digit pu, p[n][0:(U-1)];

Digit Vector: x[n][0:(U-1)], y[n][0:(U-1)],w[n][0:(U-1)],v[n][0:(U-1)], p[n][0:(U-1)];

x[-delta]=y[-delta]=w[-delta]=0;

N = floor(P/U);

for n = 0,1,2,...,N

//the first U generation

if (n = 0)

for u=-delta, -delta+1,...,-1

x[n][u+1] = CA(x[n][u],xu+delta+1);

y[n][u+1] = CA(y[n][u],yu+delta+1);

v[n][u] = 2w[n][u]+(x[n][u] yu+delta+1 + y[n][u+1] xu+delta+1) \* 2^(-3)

w[n][u+1] = v[n][u];

end

for u=0,1,2,…,U-1

x[n][u+1] = CA(x[n][u],xu+delta+1);

y[n][u+1] = CA(y[n][u],yu+delta+1);

v[n][u] = 2w[n][u]+(x[n][u] yu+delta+1 + y[n][u+1] xu+delta+1) \* 2^(-3)

pu+1 = SELM(v[n][u]);

w[n][u+1] = v[n][u] - pu+1;

p[n][u]= pu+1

end

else

// start iterative addition

for u=0,1,2,…,U-1

while (n>0)

x[n][u+1] = CA(x[n][u],xu+delta+1);

y[n][u+1] = CA(y[n][u],yu+delta+1);

v[n][u] = 2w[n][u]+(x[n][u] yu+delta+1 + y[n][u+1] xu+delta+1) \* 2^(-3)

w[n][u+1] = v[n][u];

n = n – 1;

end

//finish iterative addition, generate pu+1.

v[n][u] = 2w[n][u]+(x[n][u] yu+delta+1 + y[n][u+1] xu+delta+1) \* 2^(-3)

pu+1 = SELM(v[n][u]);

w[n][u+1] = v[n][u] - pu+1;

p[n][u]= pu+1

end

end

end

Arbitrary Precision Algorithm of Online Division

Initialization:

Online Delay: delta;

Unrolling width: U;

Precision: P;

Outer Iterator: n, from 0 to N = floor(P/U);

Inter Iterator: u, from 0 to U-1;

Input: digit du;

Output: digit qu;

Digit Vector: d[n][0:(U-1)], w[n][0:(U-1)],v[n][0:(U-1)], q[n][0:(U-1)];

x[-delta]=d[-delta]=w[-delta]=q[0] = 0;

N = floor(P/U);

for n = 0,1,2,...,N

//the first U generation

if (n = 0)

for u=-delta, -delta+1,...,-1

d[n][u+1] = CA(d[n][u],du+delta+1);

v[n][u] = 2w[n][u]+ xu+delta+1 \* 2^(-4)

w[n][u+1] = v[n][u];

end

for u=0,1,2,…,U-1

d[n][u+1] = CA(d[n][u],du+delta+1);

v[n][u] = 2w[n][u]+ xu+delta+1 \* 2^(-4) - q[n][u] du+delta+1 \* 2^(-4)

qu+1 = SELM(v[n][u]);

w[n][u+1] = v[n][u] - qu+1d[n][u+1];

q[n][u+1]= CA(q[n][u],qn+1)

end

else

// start iterative addition

for u=0,1,2,…,U-1

while (n>0)

d[n][u+1] = CA(d[n][u],du+delta+1);

v[n][u] = 2w[n][u] - q[n][u] du+delta+1 \* 2^(-4)

w[n][u+1] = v[n][u];

n = n – 1;

end

//finish iterative addition, generate qu+1.

v[n][u] = 2w[n][u]+ xu+delta+1 \* 2^(-4) - q[n][u] du+delta+1 \* 2^(-4)

qu+1 = SELM(v[n][u]);

w[n][u+1] = v[n][u] - qu+1d[n][u+1];

q[n][u+1]= CA(q[n][u],qn+1)

end

end

end



Fig.1 Data Storage for Arbitrary Precision, where the Precision P = U\*N + u.

Algorithm: Arbitrary Precision of Online Multiplier

Parameters:

* Online Delay: ;
* Unrolling width: U;
* Precision: P;
* Outer Iterator: n, from 0 to N = - 1;
* Inner Iterator: u, from 1 to U;
* Input: digits , digits ;
* Output: digits ;
* Digit Vector: *x*[n][1:U], *y*[n][1:U], *w*[n][1:U], *v*[n][1:U];
* Carry in: cin[n][U];
* Carry out: cout[n][0];

[Initialize]

x[0:N][1:U] = y[0:N][1:U] = w[0:N][1:U] =v[0:N][1:U] = 0;

**for** j = -, -+1,…,-1

*% x[0][(j++1) mod U ] ; y[0][(j++1) mod U ] ;*

x[(j++1) / U][(j++1) mod U ] ; y[(j++1) / U][(j++1) mod U ] ;

v[(j+) / U][1:U] = 2w[(j+) / U][1:U] + (x[(j+) / U][1:U] + y[(j++1) / U] [1:U]);

w[(j++1) / U][1:U] = v[(j+) / U][1:U];

**end for**

[Recurrence]

**for** j = 0, 1, 2, … ,

x[(j++1) / U][(j++1) mod U ] ; y[(j++1) / U][(j++1) mod U ] ;

**for** k = (j++1) / U, (j++1) / U -1, … , 0

{cout[k][0], v[k][1:U]}

= cin[k][U] + 2w[k][1:U] + (x[k][1:U] + y[k] [1:U]);

*% 2w[k][1:U], left shift operator is used, also need shift MSD digit in k to LSD digit position in k-1, e.g.,*

*% x[k][U] x[k][1]*

cin[k-1][U] = cout[k][0];

**if** k > 0

w[k][1:U] = v[k][1:U];

**else**

w[0][1:U] = v[0][1:U] - ;

= SELM (v[0][1:5]);

**end if**

**end for**

**end for**