Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle

David Naccache de Paz
Halim M'Silti

A new modulo computation algorithm

Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, tome 24, n° 3 (1990), p. 307-313.

<http://www.numdam.org/item?id=RO_1990__24_3_307_0>

© AFCET, 1990, tous droits réservés.

L'accès aux archives de la revue « Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle » implique l'accord avec les conditions générales d'utilisation (http://www.numdam.org/legal.php). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

Numdam

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques
http://www.numdam.org/

Recherche opérationnelle/Operations Research
(vol. 24, n° 3, 1990, p. 307 à 313)

A NEW MODULO COMPUTATION ALGORITHM (*)

by David NACCACHE DE PAZ [1] and Halim M'SILTI [2]

Abstract. – *Let X and Y be a couple of integers such that* $2^{n-1} \leqq X \leqq 2^n - 1$ *and* $0 \leqq Y \leqq 2^L - 1$.

A new algorithm which computes rapidly Y mod X is presented.

When the algorithm is executed for the first time a constant K (depending on X and L) is computed.

This K is saved for futur recalls of the procedure with other Ys smallers than $2^L - 1$.

The computation of K requires one division but in later jumps to the procedure only two multiplications, three right-shifts and at most three subtractions will be needed, provided that X remained unchanged.

The following note presents the corresponding scheme.

Keywords : Modulo; Algorithm; integer-division; register right-shifts.

Résumé. – *Soient X et Y deux entiers tels que* $2^{n-1} \leqq X \leqq 2^n - 1$ *et* $0 \leqq Y \leqq 2^L - 1$.

Un nouvel algorithme qui calcule rapidement Y mod X est présenté.

Quand l'algorithme est exécuté pour la première fois, une constante K est calculée.

Ce K est sauvegardé pour des futurs appels de la procédure avec d'autres Ys (toujours inférieurs à $2^L - 1$*).*

Le calcul de K demande une division, mais des appels futurs de l'algorithme utiliseront seulement deux multiplications, trois décalages à droite et au plus trois soustractions, étant entendu qu'X n'a pas varié.

La note qui suit présente cette méthode.

Mots clés : Modulo; Algorithme; division entière; décalages de registres.

1. INTRODUCTION

Solving O.R. problems often requires modular field operations. In practice, it is very frequent to find out that an algorithm is working constantly in the same modular field.

(*) Received in October 1989, revised in December 1989.

[1] I.B.P.-L.I.T.P., Université Pierre-et-Marie-Curie, Paris-VI, Aile 55-65, 4, place Jussieu, 75230 Paris Cedex 05.

[2] LAMSADE, Université Paris-Dauphine, place du Maréchal-de-Lattre-de-Tassigny, 75775 Paris Cedex 16.

Recherche opérationnelle/Operations Research, 0399-0559/90/03 307 07/$ 2.70
© AFCET-Gauthier-Villars

Wherefrom the natural need to build a fast "modulo X" function (where X remains unchanged all along the process or changes rarely).

The most common existing scheme to compute $R=Y \bmod X$ is:

$$R=Y-\operatorname{int}(Y/X)*X \tag{1}$$

or (in *C*-like notation):

```
Unsigned common (Y, X)
Unsigned Y, X;
{
 return (Y-X*(Y/X));
}
```

This direct formula is often too lengthy since it requires a different division each time, but it has the advantage to work on any couple of integers.

Our idea consists in computing (only once) for each X a constant value K which be reused when modulo X is called with different Ys.

The computation of this K requires one division but in later jumps to the procedure only two multiplications, three right-shifts and at most three subtractions will be needed.

The intuitive philosophy of the scheme (*see* section 3) consists in "jumping" into a *well defined* interval in which we refine our calculus by few subtractions.

The "jump" is easy to do since the distance is computed by shifts and multiplications.

2. CONVENTIONS

Throughout this paper the following conventions will be used:

CONVENTION 2.1: n will denote the length of X (in bits).

CONVENTION 2.2: Let L represent the maximal size of the Ys accepted by the procedure.

CONVENTION 2.3: and the minus operation will always be considered as unsigned digit-to-digit (here bit-to-bit) subtraction.

For instance: 10397-21033=89364.

3. THE ALGORITHM

```
Unsigned modulo (Y, X)
Unsigned Y, X;
{
  Static unsigned COPY_X, K;
  Unsigned A;
  if(COPY_X!=X){K ← 2^L/(COPY_X ← X);}
  A ← X*((K*(Y>>(n-1)))>>(L-n+1));
  A ← Y%2^(n+2) - A%2^(n+2);
  While (A≥X) {A ← A-X;}
  Return(A);
}
```

It should be pointed out that the while loop is executed at most twice and that the operation % 2^{n+2} is simply a subtraction performed on the $(n+2)$ LSBs of Y and A.

Before proving that modulo (Y, X) works let us show that the algorithm works when the line:

$A \leftarrow Y \% 2^{n+2} - A \% 2^{n+2}$;

is substituted by:

$A \leftarrow Y - A$;

Proof: prior to the while loop we subtract form Y:

$$((K(Y\rangle\rangle(n-1)))\rangle\rangle(L-n+1)) \text{ times } X$$

consequently we still have

$$Y - X((K(Y\rangle\rangle(n-1)))\rangle\rangle(L-n+1)) \equiv R$$

or :

$$Y - X((K(Y\rangle\rangle(n-1)))\rangle\rangle(L-n+1)) - \Delta X = R.$$

Let us evaluate Δ.

$$Y - X((K(Y\rangle\rangle(n-1)))\rangle\rangle(L-n+1)) - \Delta X = Y - X \operatorname{int}\left[\frac{Y}{X}\right]$$

$$X((K(Y\rangle\rangle(n-1)))\rangle\rangle(L-n+1)) + \Delta X = X \operatorname{int}\left[\frac{Y}{X}\right]$$

$$((K(Y\rangle\rangle(n-1)))\rangle\rangle(L-n+1)) + \Delta = \operatorname{int}\left[\frac{Y}{X}\right]$$

$$\exists \alpha < X \quad / \quad \operatorname{int}\left[\frac{Y}{X}\right] = \frac{Y}{X} - \frac{\alpha}{X}$$

$$((K(Y\rangle\rangle(n-1)))\rangle\rangle(L-n+1)) + \Delta = \frac{Y}{X} - \frac{\alpha}{X}.$$

Similarly:

$$\exists \beta < 2^{n-1} \quad / \quad Y\rangle\rangle(n-1) = \frac{Y}{2^{n-1}} - \frac{\beta}{2^{n-1}}$$

$$\exists \gamma < X \quad / \quad K = \frac{2^L}{X} - \frac{\gamma}{X}$$

$$\left\{\left(\frac{Y}{2^{n-1}}-\frac{\beta}{2^{n-1}}\right)\left(\frac{2^L}{X}-\frac{\gamma}{X}\right)\right\}\rangle\rangle(L-n+1)-\frac{Y}{X}+\frac{\alpha}{X}=-\Delta$$

$$\left[\frac{Y}{X2^{n-1-L}}-\frac{\beta}{X2^{n-1-L}}-\frac{\gamma Y}{X2^{n-1}}+\frac{\gamma\beta}{X2^{n-1}}\right]\rangle\rangle(L-n+1)-\frac{Y}{X}+\frac{\alpha}{X}=-\Delta.$$

And finally:

$$\exists\varepsilon<2^{L-n+1}\quad/\quad\left[\frac{Y}{X2^{n-1-L}}-\frac{\beta}{X2^{n-1-L}}-\frac{\gamma Y}{X2^{n-1}}+\frac{\gamma\beta}{X2^{n-1}}\right]\rangle\rangle(L-n+1)$$

$$=\frac{(Y/X2^{n-1-L}-\beta/X2^{n-1-L}-\gamma\, Y/X2^{n-1}+\gamma\beta/X2^{n-1}]}{2^{L-n-1}}-\frac{\varepsilon}{2^{L-n-1}}$$

So that:

$$\frac{(Y/X2^{n-1-L}-\beta/X2^{n-1-L}-\gamma\, Y/X2^{n-1}+\gamma\beta/X2^{n-1})}{2^{L-n-1}}-\frac{\varepsilon}{2^{L-n-1}}-\frac{Y}{X}+\frac{\alpha}{X}=-\Delta$$

$$\frac{Y}{X}-\frac{\beta}{X}-\frac{\gamma Y}{X2^L}+\frac{\gamma\beta}{X2^L}-\frac{\varepsilon}{2^{L-n+1}}-\frac{Y}{X}+\frac{\alpha}{X}=-\Delta$$

$$-\frac{\beta}{X}-\frac{\gamma Y}{X2^L}+\frac{\gamma\beta}{X2^L}-\frac{\varepsilon}{2^{L-n+1}}+\frac{\alpha}{X}=-\Delta$$

$$\frac{\beta}{X}+\frac{\gamma Y}{X2^L}-\frac{\gamma\beta}{X2^L}+\frac{\varepsilon}{2^{L-n+1}}-\frac{\alpha}{X}=\Delta$$

$$\Delta\leqq\frac{\beta}{X}+\frac{\gamma Y}{X2^L}+\frac{\varepsilon}{2^{L-n+1}}<\frac{\beta}{X}+\frac{Y}{2^L}+\frac{\varepsilon}{2^{L-n+1}}<\frac{\beta}{X}+\frac{Y}{2^L}+1$$

and since $2^{n-1}\leqq X$ and $Y\leqq 2^L-1$ (see 2.1 and 2.2) $\Delta<3$.

This proves that in the worst case the while loop will be executed twice.

As the length of X is n (bits) $3X$ is at most $n+2$ bits long.

Consequently the subtraction:

$$Y-X((K(Y\rangle\rangle(n-1)))\rangle\rangle(L-n+1))$$

can be done on the $n+2$ LSBs of each number and the line:

$A\leftarrow Y-A;$

replaced by:

$A\leftarrow Y\,\%\,2^{n+2}-A\,\%\,2^{n+2};$

4. EXAMPLES

Compute $R = 48619 \; mod \; 93 \; (\equiv 73)$
and $S = 47711 \; mod \; 93 \; (\equiv 2)$.

$R = ?$

Let $L = 17$

$$X = (93)_{10} = 1011101 \longrightarrow n = 7$$
$$K = 2^L / X = 10110000001$$
$$Y = (48619)_{10} = 1011110111101011$$
$$Y \rangle\rangle (n-1) = 1011110111$$
$$K * (Y \rangle\rangle (n-1)) = 10110000001 * 1011110111 = 10000010100010111 0111$$
$$(K * (Y \rangle\rangle (n-1))) \rangle\rangle (L-n+1) = 1000001010$$
$$X * (K * (Y \rangle\rangle (n-1))) \rangle\rangle (L-n+1) = 1011110110100010$$
$$X * (K * (Y \rangle\rangle (n-1))) \rangle\rangle (L-n+1) \;\%\; 2^{n+2} = 110100010$$

$$\text{Y} \;\%\; 2^{n+2} = 111101011$$

$$\begin{array}{r} 111101011 \\ - \; 110100010 \\ \hline 1001001 \end{array} = (73)_{10} = R$$

$S = ?$

Let $Y = (47711)_{10} = 1011101001011111$
$$Y \rangle\rangle (n-1) = 1011101001$$
$$K * (Y \rangle\rangle (n-1)) = 10110000001 * 1011101001 = 100000000010001101001$$
$$(K * (Y \rangle\rangle (n-1))) \rangle\rangle (L-n+1) = 1000000000$$
$$X * ((K * (Y \rangle\rangle (n-1))) \rangle\rangle (L-n+1) = 1011101000000000$$
$$X * ((K * (Y \rangle\rangle (n-1))) \rangle\rangle (L-n+1) \;\%\; 2^{n+2} = 000000000$$

$$Y \;\%\; 2^{n+2} = 001011111$$

$$\begin{array}{r} 001011111 \\ - \; 000000000 \\ \hline 1011111 \\ - \; 1011101 \\ \hline 10 \end{array} \quad \text{(While loop)}$$

$$10 = (2)_{10} = S.$$

5. SPEED

In this section we wish to estimate the theoretical gap between the computation times of *common* (one division, one multiplication and one subtraction) and *modulo* (two multiplications, three $\rangle\rangle$ (Sh*r*), two % (Sh*l*), three subtractions and a division in the first activation).

For n-bit numbers:

The instructions "$\rangle\rangle n$" and "$\% n$" are done in n clock cycles each. $A-B$ and $A+B$ take about $2n$, $A*B$ is of $2n^2$ and a division is generally admitted to be equivalent to four multiplications.

Let us denote by $C_n(k)$ the time required to execute k *commons* on n-bit numbers and by $M_n(k)$ the similar time for *modulo*.

Then $C_n(k)=k(4\times 2n^2+2n^2+2n)=k(10n^2+2n)$.

$$M_n(k)=4\times 2n^2+k(2\times 2n^2+2n+2n+3\times 2n)=8n^2+k(4n^2+10n).$$

For $n=10$ we have:

$$C_{10}(k)=1{,}020\,k \qquad \text{and} \qquad M_{10}(k)=500\,k+800$$

$C_{10}(1)<M_{10}(1)$ but for any $k>1$ $M_{10}(k)>C_{10}(k)$.

6. FURTHER IMPOVEMENTS

It is easy to prove that for all $u<L$ we have:

$$\operatorname{int}\left[\frac{\operatorname{int}(2^L/X)}{2^u}\right]=\operatorname{int}\left[\frac{2^{L-u}}{X}\right].$$

This can be applied for rewriting the algorithm under an improved form:

```
Unsigned modulo_1 (Y, X)
Unsigned Y, X;
{
  Static unsigned int COPY_X, K;
Unsigned int A, Z, l;
if(COPY_X!=X){K←2^L/(COPY_X←X);}
  l←Number_of_digits (Y):
  Z←K⟩⟩(L−l−1);
  A←X*((Z*(Y⟩⟩(n−1)))⟩⟩(l−n+2);
  A←Y%2^(n+2)−A%2^(n+2);
```

```
  While (A ≧ X) { A ← A − X; }
  Return (A);
}
```

To illustrate the difference with modulo let us compute:

$T = 1000$ modulo_1 93 $(\equiv 70)$

$T = ?$

Let $Y = (1000)_{10} \quad = 1111101000 \longrightarrow \quad 1 = 10$

$Y \rangle\rangle (n+1) \quad = 1111$

$Z = K \rangle\rangle (L - l - 1) = 10110000001 \rangle\rangle (17 - 10 - 1) = 10110$

$Z * (Y \rangle\rangle (n-1)) = 10110 * 1111 \quad = 101001010$

$(Z * (Y \rangle\rangle (n-1))) \rangle\rangle (1 - n + 2) \quad = 1010$

$X * ((Z * (Y \rangle\rangle (n-1))) \rangle\rangle (1 - n + 2)) \quad = 1110100010$

$X * ((Z * (Y \rangle\rangle (n-1))) \rangle\rangle (1 - n + 2)) \ \% \ 2^{n+2} = \ 110100010$

$Y \ \% \ 2^{n+2} = 111101000$

$$\begin{array}{r} 111101000 \\ - \ 110100010 \\ \hline 1000110 \end{array} = (70)_{10} = T$$

While in 1000 modulo 93 we must calculate:

$K * (Y \rangle\rangle (n-1)) = 10110000001 * 1111 \quad = 101001010001111$

$(K * (Y \rangle\rangle (n-1))) \rangle\rangle (L - n + 1) \quad = 1010$

$X * ((K * (Y \rangle\rangle (n-1))) \rangle\rangle (L - n + 1)) \quad = 1110100010$

$X * ((K * (Y \rangle\rangle (n-1))) \rangle\rangle (L - n + 1)) \ \% \ 2^{n+2} = \ 110100010$

$Y \ \% \ 2^{n+2} = 111101000$

$$\begin{array}{r} 111101000 \\ - \ 110100010 \\ \hline 1000110 \end{array} = (70)_{10} = T$$

7. DEDICATION

The authors wish to call the presented method *Prince Leonard's algorithm*, as a dedication to His Highness Prince Leonard of the Hutt River Province.

May the Lord bless him, his familly and all the subjects of his prosperous principality.