信州大学工学部

学士論文

固定した原始多項式による
標数 2 の有限体演算の高速化に関する検討

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　07T2017E
氏名　乙井 良太

2012 年 2 月 27 日

目次

1 序章

1.1 研究の背景と目的

今日の社会では, 携帯電話やコンピューターを 1 人 1 台は所有しており誰もが情報という分野と深い関わりを持っている. それは, 情報化社会と言われるほど顕著である. その情報化社会において, 有限体は重要な役割を担っている. 例えば送信者から送られた情報源を暗号化し受信者がそれを復号化するという一連の作業のなかで, 暗号化と復号化をする際に有限体が用いられている. 有限体の利点は小数をとらず誤差が出ないことであり, 暗号化された情報源を誤りなく正確に復号化することができる. 有限体の演算をするプログラミングのツールとして Number Theory Library(以下 NTL)[1] がある. この NTL は任意の標数の有限体演算が可能であり, コンピュータの構造にあわせた標数 2 の有限体を必要とする情報通信分野や標数に囚われず理論体系の構築を目的とする情報理論の研究者の間で主流である.

有限体をベクトル表現する場合, 乗算や除算をする過程では原始多項式を法とする多項式の剰余計算が必要となる. NTL のように汎用性のあるプログラムでは, 原始多項式やその次数は変数として宣言され, 多項式の剰余計算は乗算や除算をするたびに実行される. さらにその剰余計算には分岐命令が用いられるので, CPU のパイプライン処理による高速化が妨げられている．ところが, 標数や原始多項式を固定した場合, 剰余計算に用いる多くの値を演算済みの定数として事前に用意することが可能である. これにより乗算や除算を実行する時間を短くすることができると考えられる. さらに, 分岐命令が不要になることからも実行時間の短縮が期待できる．本研究では, 現在のコンピューターの構造に合った標数 2 に特化し, 上記の方法による有限体演算の高速化を提案する. この方法のデメリットは, 原始多項式ごとにプログラムを用意しなくてはならない点であるが，与えられた原始多項式から演算済み定数を含んだプログラムを自動生成することでこの問題を解決することも合わせて提案する.

1.2 本論文の構成

本論文は, 次のように構成されている. 2 章では, 有限体の乗算と有限体の除算について説明す. 3 章では従来法と提案法の概要とそれらの実験結果を示し, 比較・考察を行う. 4 章では, 提案法プログラムの自動生成について述べる. 5 章では本論文のまとめを記す.

2　有限体

後に説明する従来法と提案法による有限体乗算と除算の演算方法は共通して次のような手順で二つの多項式を用いて考える.

$$f_a(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x^1 + a_0x^0$$
$$f_b(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1x^1 + b_0x^0$$

係数 $[a_{n-1}, a_{n-2}, \cdots, a_1, a_0]$ と係数 $[b_{n-1}, b_{n-2}, \cdots, b_1, b_0]$ には, 0 または 1 を代入する. また x は不定元と呼ばれ, 形式的な変数で数値を代表するものではない. 以上のことをふまえて $f_a(x)$ と $f_b(x)$ を 2 進数表現する [2].

2.1　有限体の乗算

ここでは, 有限体の乗算について $GF(2^3)$ を例に説明する. なお原始多項式は $x^3 + x + 1$ を使用する.

[手順 1]　$f_a(x)$ と $f_b(x)$ の二つの多項式を用意し, それらを乗算する.
$f_a(x) \times f_b(x) = f_c(x) \quad \Longrightarrow \quad (a_2x^2 + a_1x + a_0) \times (b_2x^2 + b_1x + b_0) = c_2x^2 + c_1x + c_0$
$(a_2x^2 + a_1x + a_0) \times (b_2x^2 + b_1x + b_0)$
$= a_2b_2x^4 + (a_2b_1 + a_1b_2)x^3 + (a_2b_0 + a_0b_2 + a_1b_1)x^2 + (a_1b_0 + a_0b_1)x + a_0b_0$

[手順 2]　次数が 3 以上の係数を原始多項式を用いて次数 2 以下にする.
$x^3 \to x + 1$ より　　$(a_2b_1 + a_1b_2)x^3 \quad \to \quad (a_2b_1 + a_1b_2)x + (a_2b_1 + a_1b_2)$
$x^4 \to x^2 + x$ より　　$a_2b_2x^4 \quad \to \quad a_2b_2x^2 + a_2b_2x$
なので
$f_a(x) \times f_b(x) = f_c(x)$
$\Longrightarrow \quad (a_2b_0 + a_0b_2 + a_1b_1 + a_2b_2)x^2 + (a_1b_0 + a_0b_1 + a_2b_1 + a_1b_2 + a_2b_2)x$
$+(a_0b_0 + a_2b_1 + a_1b_2) = c_2x^2 + c_1x + c_0$

[手順 3]　$[a_2, a_1, a_0]$ と $[b_2, b_1, b_0]$ に 0 または 1 を代入し, その時の $[c_2, c_1, c_0]$ を導く.
例えば, 2×2 の有限体乗算したいとき, a=2より $[a_2, a_1, a_0]$=$[0, 1, 0]$, b=2より $[b_2, b_1, b_0]$=$[0, 1, 0]$ なので $[c_2, c_1, c_0]$=$[1, 0, 0]$ となり演算結果は 4 だとわかる.

以下 [手順 3] を繰り返し, $GF(2^3)$ の有限体乗算は以下の表になる.

表 1　$GF(2^3)$ の乗算

b \ a	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

2.2　有限体の除算

ここでは, 有限体の除算について $GF(2^3)$ を例に説明する. なお原始多項式は x^3+x+1 を使用する.

[手順 1]　$f_a(x)$ と $f_b(x)$ の二つの多項式を用意し, それらを除算する.

$f_a(x) \div f_b(x) = f_c(x) \quad \Longrightarrow \quad f_a(x) = f_b(x) \times f_c(x)$

$(b_2x^2+b_1x+b_0) \times (c_2x^2+c_1x+c_0) = b_2c_2x^4 + (b_2c_1+b_1c_2)x^3 + (b_2c_0+b_1c_1+b_0c_2)x^2$
$+(b_1c_0+b_0c_1)x + b_0c_0$

[手順 2]　次数が 3 以上の係数を原始多項式を用いて次数 2 以下にする.

$x^3 \to x+1$ より　　$(b_2c_1+b_1c_2)x^3 \quad \to \quad (b_2c_1+b_1c_2)x + (b_2c_1+b_1c_2)$

$x^4 \to x^2+x$ より　　$b_2c_2x^4 \quad \to \quad b_2c_2x^2 + b_2c_2x$

なので

$f_b(x) \times f_c(x) = f_a(x)$

$\Longrightarrow \quad (b_2c_0+b_1c_1+b_0c_2+b_2c_2)x^2 + (b_1c_0+b_0c_1+b_2c_1+b_1c_2+b_2c_2)x$
$+(b_0c_0+b_2c_1+b_1c_2) = a_2x^2+a_1x+a_0$

[手順 3]　$[a_2,a_1,a_0]$ と $[b_2,b_1,b_0]$ に 0 または 1 を代入し, その時の $[c_2,c_1,c_0]$ を導く.

例えば, $2 \div 2$ の有限体除算をしたいとき, a=2 より $[a_2,a_1,a_0]$=[0,1,0], b=2 より $[b_2,b_1,b_0]$=[0,1,0] から

連立方程式

$$\begin{cases} c_2 \cdot 0 + c_1 \cdot 1 + c_0 \cdot 0 = 0 \\ c_2 \cdot 1 + c_1 \cdot 0 + c_0 \cdot 1 = 1 \\ c_2 \cdot 1 + c_1 \cdot 0 + c_0 \cdot 0 = 0 \end{cases} \tag{1}$$

が導出され解くと $[c_2, c_1, c_0]$=$[0, 0, 1]$ となり, 演算結果は 1 だとわかる.
以下 [手順 3] を繰り返し, $GF(2^3)$ の有限体除算は以下の表になる.

表 2　$GF(2^3)$ の除算

	$\div$	1	2	3	4	5	6	7
		a						
	1	1	2	3	4	5	6	7
	2	5	1	4	2	7	3	6
b	3	6	7	1	5	3	2	4
	4	7	5	2	1	6	4	3
	5	2	4	6	3	1	7	5
	6	3	6	5	7	4	1	2
	7	4	3	7	6	2	5	1

2.3　多倍長リンク

有限体演算に 32 ビット計算機を用いたので, ワード単位では次数 32 までの有限体しか演算することができない. しかし, 本研究では図 1 のようにプログラム上で配列を組み 33 ビット目を配列の上位の 1 ビット目へ 32 ビット以下を配列の下位におく多倍長リンクを実装し, 次数 33 での有限体の演算を可能にした [3].

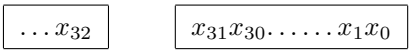

図 1　多倍長リンク

3　従来法と提案法の比較

3.1　従来法とその改善点

乗算や除算をする過程において [手順 2] のような剰余計算が必要となる. NTL ではこの剰余計算で必要となる原始多項式や次数は変数として宣言されている. 本研究では NTL のように原始多項式や次数が変数として宣言されている有限体演算プログラムを従来法と呼び, 実際

にそのプログラムを組んだ. 原始多項式や次数は変数として宣言されているので, 多項式の剰余計算は乗算や除算をするたびに実行される. また, その剰余計算には分岐命令が用いられており, CPU のパイプライン処理による高速化が妨げられている.

3.2 提案法のねらい

提案法では, 従来法で変数として扱われていた次数と原始多項式を演算済み定数として扱い, 計算量と分岐命令を減らすことで有限体演算の高速化を目指す. 次数と原始多項式を定数とした場合, [手順 2] の剰余計算に用いる多くの値を演算済みの定数として事前に用意することができる. 図 2 は実際に原始多項式を変数とする乗算プログラムの例であるが, 原始多項式を $x^3 + x + 1$ に固定して演算済み定数を使うことによって乗算プログラムは図 3 のように短くなり, 計算量と分岐命令が減っていることがわかる. なお, 有限体除算の演算プログラムも同様である. この章では除算の演算プログラムは割愛するが, 提案法は従来法よりも計算量と分岐命令が減っている.

```
y0 = (a & 0x01)? b: 0;
y1 = 0;

for(i = 1; i < n; i++){
       y0 ^= ((a & (0x01 << i))? (b << i): 0) & mask;
       y1 ^= ((a & (0x01 << i))? (b >> (n - i)): 0) & mask;
}

y2 = 0x01 << (n - 1);

for (k = 0; k < n - 1; k++){
         y2 = ((y2 << 1) ^ ((y2 & (0x01 << (n - 1)))? p: 0)) & mask;
         if (y1 & (0x01 << k)){
                 y0 ^= y2;
         }
}
```

図 2　原始多項式を変数とした有限体乗算の演算プログラム (従来法)

```
ab[0] = (a & 0x1)? b: 0;
ab[1] = (a & 0x2)? b: 0;
ab[2] = (a & 0x4)? b: 0;

y0 = ((ab[0]) ^ (ab[1] << 1) ^ (ab[2] << 2)) & 0x7;

y1 = (ab[1] >>2) ^ (ab[2] >>1);
r = y0;
r ^= (y1 & 0x01) ? 0x3: 0;
r ^= (y1 & 0x2) ? 0x6 : 0;
```

図 3　原始多項式を演算済み定数とした有限体乗算の演算プログラム (提案法)

3.3 実験結果と検証

表 3　乗算の提案法と従来法の時間比

次数 n	提案法 $t_1[s]$	従来法 $t_2[s]$	$t_1[s]/t_2[s]$
2	17.236	246.954	0.0698
3	9.139	152.950	0.0598
4	4.071	83.292	0.0489
5	18.090	437.169	0.0414
6	8.339	207.389	0.0402
7	3.883	97.872	0.0397
8	1.734	46.000	0.0374
9	7.905	209.077	0.0378
10	3.554	92.582	0.0384
11	15.992	408.138	0.0392
12	41.523	995.164	0.0417
13	46.287	1023.628	0.0452
14	47.945	1083.723	0.0442
15	51.534	1074.919	0.0479
16	54.813	1129.112	0.0485
17	58.023	1168.012	0.0497
18	62.257	1210.95	0.0514
19	67.200	1267.612	0.0530
20	70.130	1305.556	0.0537
21	73.254	1341.874	0.0546
22	77.431	1370.630	0.0565
23	79.657	1430.891	0.0557
24	86.272	1487.843	0.0580
25	89.983	1519.732	0.0592
26	89.795	1577.365	0.0569
27	94.865	1621.064	0.0585
28	99.496	1652.387	0.0602
29	102.143	1695.897	0.0602
30	103.265	1740.142	0.0593
31	108.549	1785.030	0.0610
32	151.153	1806.670	0.0837
33	229.763	1874.543	0.1226

表 4　除算の提案法と従来法の時間比

次数 n	提案法 $t_3[s]$	従来法 $t_4[s]$	$t_3[s]/t_4[s]$
2	41.277	363.174	0.114
3	47.036	352.235	0.134
4	36.618	234.466	0.156
5	24.09	134.425	0.179
6	139.864	704.424	0.199
7	77.393	363.145	0.213
8	40.154	224.995	0.178
9	196.675	825.800	0.238
10	93.799	386.790	0.243
11	469.461	1741.994	0.269
12	199.463	1039.608	0.192
13	102.417	471.483	0.217
14	484.204	2251.978	0.215
15	2193.277	6957.067	0.315
16	9761.684	47544.891	0.217
17	272.355	815.841	0.334
18	304.440	1027.698	0.296
19	349.125	1194.153	0.292
20	370.565	1047.037	0.354
21	403.674	1129.051	0.358
22	440.573	1199.780	0.367
23	485.059	1339.223	0.362
24	524.400	1736.881	0.302
25	562.761	1487.875	0.378
26	605.031	1912.735	0.316
27	654.110	1994.048	0.328
28	697.934	1786.769	0.390
29	747.623	1892.107	0.395
30	816.271	2107.443	0.387
31	847.526	2128.145	0.398
32	934.081	1103.311	0.847
33	1361.521	1957.861	0.695

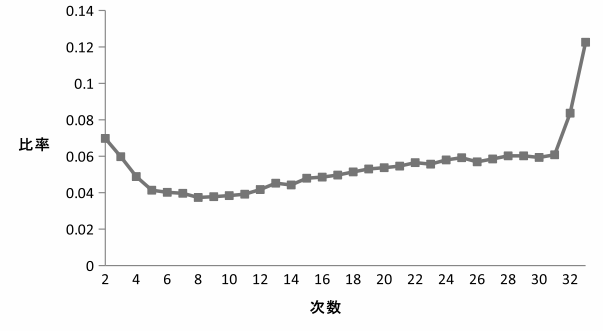

図 4　乗算の提案法と従来法の時間比

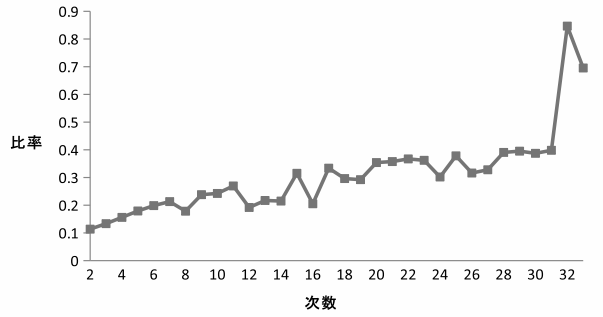

図 5　除算の提案法と従来法の時間比

表 3 と図 4 に乗算の演算時間比較実験の結果を示した. それらの結果より, すべての次数において提案法が従来法よりも演算速度が高速であることが分かった. また, 次数 33 で比率が急激に大きくなっている. これは多倍長リンクを実装するために提案法のプログラミングで配列を用いたことが原因だと考えられる. 次に, 表 4 と図 5 に除算の演算時間比較実験の結果を示した. それらの結果より, すべての次数において提案法が従来法よりも演算速度が高速であることが分かった. 図 5 が大きく振れているのは除算の [手順 3] で説明した連立方程式が関係していると考えられる. 除算のプログラム中における連立方程式を解くプログラムは比重が大きく, 次数や原始多項式によっては演算速度の向上の妨げの要因となっている. さらに, 図 4 と図 5 を比較すると提案法の効果が大きいのは図 4 の乗算だと言える. 除算よりも乗算の方が提案法効果が大きいのは, 除算の連立方程式解法の比重が大きいことによると考えられる.

4　提案法プログラムの自動生成

次数と原始多項式を演算済み定数として有限体演算を実行するためには, 原始多項式とその次数ごとに演算プログラムを用意しなければならない. このデメリットを解決するために以下のような方法で, 与えられた原始多項式から演算済み定数を含んだプログラムを自動生成させることでこの問題を解決する.

[1 段階目のプログラム]

1 段階目のプログラムは, 次数と原始多項式を演算済み定数とした有限体演算プログラムを自動生成するためのものである. 図 6 は乗算の 1 段階目のプログラムを一部抜粋したものである. なお, 図 6 中にある p には原始多項式, n には次数が入る.

```
printf("\t r ^= (y1 & 0x01) ? 0x%x: 0; \n", p);

p0 = p;

for (i = 1; i < n-1; i++){
        p0 = ((p0 << 1) ^ (((1 << (n - 1)) & p0)? p: 0)) & (v - 1);
        printf("\t r ^= (y1 & 0x%x) ? 0x%x : 0;\n", 1 << i, p0);
 }
```

図 6　有限体演算プログラムを自動生成するプログラム

[2 段階目のプログラム]

例に $GF(2^3)$ を挙げ, 1 段階目のプログラムから演算済み定数を含んだ有限体演算プログラムを自動生成させる. 図 7 は自動生成することで得られた演算プログラムの図 6 にあたる部分である. p と n が図 7 中に見られないことより, 次数と原始多項式が演算済み定数となったことがわかる.

```
r ^= (y1 & 0x01) ? 0x3 : 0;
r ^= (y1 & 0x02) ? 0x6 : 0;
```

図 7　自動生成された有限体演算プログラム

5　まとめ

NTL で公開されている有限体演算のプログラムでは原始多項式と次数が変数として宣言されており, 計算量と分岐命令が多くなる. 本研究では, この計算量と分岐命令が有限体演算の高速化の妨げとなっていることに着目した. 原始多項式と次数が変数として宣言されている有限体演算プログラムを従来法とし, またこれを改善するために原始多項式と次数を演算済み定数とした有限体演算プログラムを提案法とした. この提案法には, 有限体演算の際に原始多項式と次数を演算済み定数として実行することで計算量と分岐命令を減らし, 高速化させるねらいがある. 乗算と除算のそれぞれの従来法と提案法のプログラムを組み演算時間の比較実験を行なった. 結果として, 両者で提案法が従来法よりも演算時間が短いことが実証された. また, 両者の時間比率を比較すると乗算のほうが値が低く, 提案法の効果が大きいということがわかった. 今後の課題としては, 今回の実験では標数 2 に限定したが任意の標数でも比較・検討することで提案法の活躍の場が広がると思われる.

謝辞

本研究を行うにあたって，細かく指導してくださった指導教員の西新幹彦准教授, また本研究に関して貴重なご意見をいただきました杉村立夫教授に感謝の意を表する.

参考文献

[1] http://shoup.net/ntl/download.html, 2011 年 5 月閲覧.
[2] 藤原良, 神保雅一,“符号と暗号の数理”, 共立出版株式会社, 2008.
[3] ヘンリー・S・ウォーレンジュニア,“ハッカーのたのしみ”, 株式会社エスアイビー・アクセス, 2004.

付録 A　本研究で用いた原始多項式一覧

本研究で用いた各次数の原始多項式を表 5 に示す.

表 5　使用した原始多項式

次数 n	n 次の原始多項式
2	$1+x+x^2$
3	$1+x+x^3$
4	$1+x+x^4$
5	$1+x^2+x^5$
6	$1+x+x^6$
7	$1+x+x^7$
8	$1+x+x^2+x^7+x^8$
9	$1+x^4+x^9$
10	$1+x^3+x^{10}$
11	$1+x^2+x^{11}$
12	$1+x+x^2+x^{12}$
13	$1+x+x^2+x^5+x^{13}$
14	$1+x+x^2+x^{12}+x^{14}$
15	$1+x+x^{15}$
16	$1+x+x^3+x^{12}+x^{16}$
17	$1+x^3+x^{17}$
18	$1+x^7+x^{18}$
19	$1+x+x^2+x^5+x^{19}$
20	$1+x^3+x^{20}$
21	$1+x^2+x^{21}$
22	$1+x+x^{22}$
23	$1+x^5+x^{23}$
24	$1+x+x^2+x^7+x^{24}$
25	$1+x^3+x^{25}$
26	$1+x+x^2+x^6+x^{26}$
27	$1+x+x^2+x^5+x^{27}$
28	$1+x^3+x^{28}$
29	$1+x^2+x^{29}$
30	$1+x+x^2+x^{23}+x^{30}$
31	$1+x^3+x^{31}$
32	$1+x+x^2+x^{22}+x^{32}$
33	$1+x^{13}+x^{33}$

付録 B　ソースコード

B.1　従来法による有限体乗算プログラム

```
#include<stdio.h>

void
rshift(unsigned int xi[], unsigned int xo[],int n ,int s)
{
        int  k;

        if ( s % 32 == 0 ){
              for (k = 0; k < s/32; k++){
                     xo[n - k - 1] = 0x00;
         }
              for ( k = n - s/32 ; 0 < k; k--){
                     xo[k - 1] = xi[k - 1 + s/32];
               }

         } else {
             for( k = 0; k < (s / 32); k++){

                         xo[n - k - 1] = 0x00;

              }

               xo[n - 1 - (s/32)] = xi[n - 1] >> ( s % 32);


               for(k = n - 2 - (s/32) ; 0 <= k; k-- ){

                       xo[k] = (xi[k + (s/32) ] >> (s % 32)) | (xi[k + (s / 32) + 1] << (32 - (s % 32)));


                }
          }

         return;
 }

void
lshift(unsigned int xi[], unsigned int xo[],int n ,int s)
{
        int  k;

        if ( s % 32 == 0 ){
                for (k = 0; k < s / 32; k++){
                        xo[k] = 0x00;
                 }
                for (; k < n; k++){
                      xo[k] = xi[k - (s  / 32)];
                 }

         } else {
           for( k = 0; k < (s / 32); k++){

                       xo[k] = 0x00;

           }

           xo[s/32] = xi[0] << ( s % 32);


           for(k = (s / 32) + 1; k < n; k++ ){
```

```
                xo[k] = (xi[k - (s/32)] << (s % 32)) | (xi[k - (s / 32) - 1] >> (32 - (s % 32)));


            }
        }


        return;
 }

/************************************/
#define MULTIWORD 2


void
prod(unsigned int p[], unsigned int a[], unsigned int b[], unsigned int c[], int m)
{
        int i;
        unsigned int y0[MULTIWORD];
        unsigned int y1[MULTIWORD];
        unsigned int y2[MULTIWORD];
        unsigned int  d[MULTIWORD];
        unsigned int mask;

        mask = (0x01 << (((m - 1) % 32) + 1)) - 1;

        y0[0] = (a[0] & 0x01)? b[0]: 0;
        y0[1] = (a[0] & 0x01)? b[1]: 0;
        y1[0] = 0;
        y1[1] = 0;

        for (i = 1; i < m; i++){
                if (a[i / 32] & (0x01 << (i % 32))){

                        lshift(b, d, MULTIWORD, i);
                        y0[0] ^= d[0];
                        y0[1] ^= d[1];
                        y0[(m - 1) / 32] &= mask;
                        if (m < 33){
                                y0[1] = 0;
                        }
                        rshift(b, d, MULTIWORD, m - i);
                        y1[0] ^= d[0];
                        y1[1] ^= d[1];
                }
          }


         if (m < 32){
                  y2[0] = 0x01 << (m - 1);
                  y2[1] = 0x00;
          } else {
                  y2[0] = 0x00;
                  y2[1] = 0x01 << ((m - 1) % 32);
          }


          for (i = 0; i < m - 1; i++){
                  if (y2[(m - 1) / 32] & (0x01 << ((m - 1) % 32))){
                          y2[(m - 1) / 32] ^= (0x01 << ((m - 1) % 32));
                          lshift(y2, d, MULTIWORD, 1);
                          y2[0] = d[0] ^ p[0];
                          y2[1] = d[1] ^ p[1];
                  } else {
                          lshift(y2, d, MULTIWORD, 1);
                          y2[0] = d[0];
                          y2[1] = d[1];
                  }
                  if (y1[i / 32] & (0x01 << (i % 32))){
```

```
                        y0[0] ^= y2[0];
                        y0[1] ^= y2[1];
                }
          }
          c[0] = y0[0];
          c[1] = y0[1];

          return;
}



int
main()
{
        unsigned int p[MULTIWORD] = {0x003, 0x0000};
        unsigned int a[MULTIWORD];
        unsigned int b[MULTIWORD];
        unsigned int c[MULTIWORD];

        for (a[0] = 0 ; a[0] < 8; a[0]++){
        for (a[1] = 0 ; a[1] < 1; a[1]++){
        for (b[0] = 0 ; b[0] < 8; b[0]++){
        for (b[1] = 0 ; b[1] < 1; b[1]++){

        prod(p, a, b, c, 3);

        printf("%08x %08x\n", a[1], a[0]);
        printf("%08x %08x\n", b[1], b[0]);
        printf("%08x %08x\n", c[1], c[0]);

        }
        }
        }
        }

         return(0);
}
```

B.2　従来法による有限体除算プログラム

```
#include<stdio.h>
#include<time.h>
void
rshift(unsigned int xi[], unsigned int xo[],int n ,int s)
{
        int  k;

        if ( s % 32 == 0 ){
              for (k = 0; k < s/32; k++){
                     xo[n - k - 1] = 0x00;
         }
              for ( k = n - s/32 ; 0 < k; k--){
                     xo[k - 1] = xi[k - 1 + s/32];
               }

         } else {
             for( k = 0; k < (s / 32); k++){

                         xo[n - k - 1] = 0x00;

              }

               xo[n - 1 - (s/32)] = xi[n - 1] >> ( s % 32);


               for(k = n - 2 - (s/32) ; 0 <= k; k-- ){

                       xo[k] = (xi[k + (s/32) ] >> (s % 32)) | (xi[k + (s / 32) + 1] << (32 - (s % 32)));


                }
          }

         return;
 }

void
lshift(unsigned int xi[], unsigned int xo[],int n ,int s)
{
        int  k;

        if ( s % 32 == 0 ){
                for (k = 0; k < s / 32; k++){
                        xo[k] = 0x00;
                 }
                for (; k < n; k++){
                      xo[k] = xi[k - (s  / 32)];
                 }

         } else {
           for( k = 0; k < (s / 32); k++){

                       xo[k] = 0x00;

           }

           xo[s/32] = xi[0] << ( s % 32);


           for(k = (s / 32) + 1; k < n; k++ ){

                    xo[k] = (xi[k - (s/32)] << (s % 32)) | (xi[k - (s / 32) - 1] >> (32 - (s % 32)));
```

```
            }
        }


        return;
 }
/************************************/
#define MULTIWORD 2


void
div(unsigned int p[], unsigned int a[], unsigned int b[], unsigned int c[], int m)
{
    int i, j, k;


    unsigned int  d[MULTIWORD];
    unsigned int mask;
    unsigned int s[64][MULTIWORD];
    unsigned int t[64][MULTIWORD];
    unsigned int u[MULTIWORD];
    unsigned int tmp;
    unsigned int p2[MULTIWORD];


    mask = (0x01 << (((m - 1) % 32) + 1)) - 1;

    for (i = 0; i < m;i++){
        lshift(b, s[i], MULTIWORD, i);
        s[i][(m - 1) / 32] &= mask;

    }

    if (m < 33){
        p2[0] = 0x01 << (m - 1);
        p2[1] = 0x00;
    } else {
        p2[0] = 0x00;
        p2[1] = 0x01 << (m - 32);
    }
    for (i = 1; i < m; i++){
        if (p2[(m - 1) / 32] & (0x01 << ((m - 1) % 32))){
            p2[(m - 1) / 32] ^= (0x01 << ((m - 1) % 32));
            lshift(p2, d, MULTIWORD, 1);
            p2[0] = d[0] ^ p[0];
            p2[1] = d[1] ^ p[1];
        } else {
            lshift(p2, d, MULTIWORD, 1);
            p2[0] = d[0];
            p2[1] = d[1];
        }

        for (j = 0; j < m; j++){
            if (p2[(m - j - 1) / 32] & (1 << ((m - j - 1) % 32))){
                rshift(b, u, MULTIWORD, i);
                s[j][0] ^= u[0];
                s[j][1] ^= u[1];
            }

         }
     }

     for (i = 0; i < m; i++){
         if (a[(m - i - 1) / 32] & (1 << ((m - i - 1) % 32))){
             if (m > 32){
                 t[i][0] = 0xffffffff;
                 t[i][1] = mask;
              } else {
```

```
            t[i][0] = mask;
            t[i][1] = 0x00000000;
          }
        } else {
            t[i][0] = 0x00000000;
            t[i][1] = 0x00000000;
        }

    }

    for (i = 0; i < m; i++){
    for (j = i; j < m; j++){
        if (s[j][(m - i - 1) / 32] & (1 << ((m - i - 1) % 32))){
            break;
        }
    }
    if (j >= m){

        return;
    }

    tmp = s[i][0], s[i][0] = s[j][0], s[j][0] = tmp;
    tmp = s[i][1], s[i][1] = s[j][1], s[j][1] = tmp;
    tmp = t[i][0], t[i][0] = t[j][0], t[j][0] = tmp;
    tmp = t[i][1], t[i][1] = t[j][1], t[j][1] = tmp;

    for (k = 0; k < m; k++){
        if ((k != i) && (s[k][(m - i - 1) / 32] & (1 << ((m - i - 1) % 32)))){
            s[k][0] ^= s[i][0];
            s[k][1] ^= s[i][1];
            t[k][0] ^= t[i][0];
            t[k][1] ^= t[i][1];

        }
     }
  }

  c[0] = 0;
  c[1] = 0;
  for (i = 0; i < m; i++){
      c[i / 32] ^= (t[i][i / 32] & (1 << (i % 32)));
  }
  return;
}

int
main()
{
        unsigned int p[MULTIWORD] = {0x0003, 0x0000};
        unsigned int a[MULTIWORD];
        unsigned int b[MULTIWORD];
        unsigned int c[MULTIWORD];

        for (a[0] = 0 ; a[0] < 8; a[0]++){
        for (a[1] = 0 ; a[1] < 1; a[1]++){
        for (b[0] = 0 ; b[0] < 8; b[0]++){
        for (b[1] = 0 ; b[1] < 1; b[1]++){

                div(p, a, b, c, 3);

                printf("%08x %08x\n", a[1], a[0]);
                printf("%08x %08x\n", b[1], b[0]);
                printf("%08x %08x\n", c[1], c[0]);


        }
        }
        }
        }
```

```
        return(0);
}
```

B.3 提案法による有限体乗算プログラムを自動生成するプログラム

```
#include<stdio.h>

void
gen_prod(int n, unsigned int p)
{
   int i;
   int v = 1 << n;
   unsigned int p0;

   printf("#include <stdio.h>\n\n");
   printf("#define MULTIWORD 2\n\n\n");
   printf("void\n");
   printf("prod(unsigned int a[], unsigned int b[], unsigend int c[])\n");
   printf("{\n");

   if(n < 32){
        printf("\tunsigned int ab[%d];\n", n );
        }else{
        printf("\tunsigned int ab[%d][2];\n", n);
   }

   if(n < 32){
        printf("\tunsigned int y0, y1;\n");
        printf("\tunsigned int r;\n");
        printf("\n");
        }else{
        printf("\tunsigned int y0, y1;\n");
        printf("\tint i;\n");
        printf("\n");
    }

    for (i = 0; i < n; i++){
          printf("\tab[%d] = (a & 0x%x)? b: 0;\n", i, 1 << i);
    }

    printf("\ty0 = ((ab[0])");

    for (i = 1; i < n; i++){
           printf(" ^ (ab[%d] << %d)", i, i);
    }

    printf(") & 0x%x;\n", (1 << n) - 1);

    printf("\ty1 = (ab[1] >>%d)",n - 1);

    for (i = 2;i < n; i++){
            printf(" ^ (ab[%d] >>%d)" ,i , n-i);
    }

    printf(";\n");

    printf("\tr = y0;\n");

    printf("\tr ^= (y1 & 0x01) ? 0x%x: 0;\n", p);

    p0 = p;

    for (i = 1; i < n-1; i++){
           p0 = ((p0 << 1) ^ (((1 << (n - 1)) & p0)? p: 0)) & (v - 1);
```

```
            printf("\tr ^= (y1 & 0x%x) ? 0x%x : 0;\n", 1 << i, p0);
    }
    printf("\n");
    printf("\treturn(r);\n");
    printf("}\n");
    printf("\n");

    printf("int\n");
    printf("main()\n");
    printf("{\n");
    printf("\tint a,b;\n");
    printf("\tfor(a=0 ; a<%d; a++){\n",v);
    printf("\t\tfor(b=0 ; b<%d; b++){\n",v);
    printf("\t\t\tprintf(\"%s\\n\", a, b, prod(a,b));\n", "%d * %d = %d");
    printf("\t\t}\n");
    printf("\t}\n");
    printf("\treturn(0);\n");
    printf("}\n");

    return;
}

int
main()
{
    gen_prod(3, 0x03);
    return(0);
}
```

B.4 提案法による有限体乗算プログラム

```
#include <stdio.h>

#define MULTIWORD 2


void
prod(unsigned int a[], unsigned int b[], unsigend int c[])
{
       unsigned int ab[3];
       unsigned int y0, y1;
       unsigned int r;

       ab[0] = (a & 0x1)? b: 0;
       ab[1] = (a & 0x2)? b: 0;
       ab[2] = (a & 0x4)? b: 0;
       y0 = ((ab[0]) ^ (ab[1] << 1) ^ (ab[2] << 2)) & 0x7;
       y1 = (ab[1] >>2) ^ (ab[2] >>1);
       r = y0;
       r ^= (y1 & 0x01) ? 0x3: 0;
       r ^= (y1 & 0x2) ? 0x6 : 0;

       return(r);
}

int
main()
{
        int a,b;

        for(a=0 ; a<8; a++){
                        for(b=0 ; b<8; b++){
                                        printf("%d * %d = %d\n", a, b, prod(a,b));
                        }
```

```
        }
return(0);
}
```

B.5 提案法による有限体除算プログラムを自動生成するプログラム

```
#include<stdio.h>

void
gen_div(int n, unsigned int p)
{
        int i,j;
        int v = 1 << n;
        unsigned int q[32];

        printf("#include<stdio.h>\n");

        printf("int\n");
        printf("div(int a, int b)\n");
        printf("{\n");
        printf("\tint i, j, k;\n");
        printf("\tunsigned char c[%d];\n", n );
        printf("\tunsigned char e[%d];\n", n );
        printf("\tunsigned char tmp;\n");
        printf("\n");

        q[0] = (1 << n) - 1;
        q[1] = p;

        for (i = 2; i < n; i++){
                q[i] = (q[i - 1] << 1) ^ (((1 << (n - 1)) & q[i - 1])? p: 0);
        }

        for(i = 0 ; i < n ; i++){
                printf("\tc[%d] = ((b << %d)", i, i );
                for (j = 1; j < n; j++){
                        if(q[j] & (1 << (n - 1 - i))){
                                printf(" ^ (b >> %d)", j);
                        }
                }

                printf(") & 0x%x;\n", (1 << n) - 1);
        }

        for( i = 0 ; i < n ; i++){
                printf("\te[%d] = (a & 0x%x) ? 0x%x: 0x00;\n", i , 1 << (n - i - 1), (1 << n)-1);
        }
        printf("\n");

        printf("\tfor (k = 0; k < %d ; k++){\n", n);
        printf("\t\tfor (j = k ; j < %d ; j++){\n", n);
        printf("\t\t\tif (c[j] & (0x%x >> k)){\n", 1 << (n - 1));
        printf("\t\t\t\tbreak;\n");
        printf("\t\t\t}\n");
        printf("\t\t}\n");
        printf("\t\tif (j >= %d){\n", n);
        printf("\t\t\treturn(0);\n");
        printf("\t\t}\n");
        printf("\t\ttmp = c[k] ,c[k] = c[j], c[j] = tmp;\n");
        printf("\t\ttmp = e[k] ,e[k] = e[j], e[j] = tmp;\n");
        printf("\t\tfor(i = 0 ; i < %d ; i++){\n" , n);
        printf("\t\t\tif ((i != k) && (c[i] & (0x%x >> k))){\n", 1 << (n - 1));
        printf("\t\t\t\tc[i] = c[k] ^ c[i];\n");
```

```
        printf("\t\t\t\te[i] = e[k] ^ e[i];\n");
        printf("\t\t\t}\n");
        printf("\t\t}\n");
        printf("\t}\n");
        printf("\treturn((e[0] & 0x01)");

        for( i = 1 ; i < n ; i++){
                printf(" ^ (e[%d] & 0x%x)" , i , 1 << i);
        }

        printf(");\n");
        printf("}\n");
        printf("int\n");
        printf("main()\n");
        printf("{\n");
        printf("\tint a,b;\n");
        printf("\tfor(a=0 ; a<%d; a++){\n",v);
        printf("\t\tfor(b=0 ; b<%d; b++){\n",v);
        printf("\t\t\tprintf(\"%s\\n\", a, b, div(a,b));\n", "%d / %d = %d");
        printf("\t\t}\n");
        printf("\t}\n");
        printf("\treturn(0);\n");
        printf("}\n");

        return;
}

int
main()
{
        gen_div(3, 0x0000003);

        return(0);
}
```

B.6 提案法による有限体除算プログラム

```
#include<stdio.h>
int
div(int a, int b)
{
        int i, j, k;
        unsigned char c[3];
        unsigned char e[3];
        unsigned char tmp;

        c[0] = ((b << 0) ^ (b >> 2)) & 0x7;
        c[1] = ((b << 1) ^ (b >> 1) ^ (b >> 2)) & 0x7;
        c[2] = ((b << 2) ^ (b >> 1)) & 0x7;
        e[0] = (a & 0x4) ? 0x7: 0x00;
        e[1] = (a & 0x2) ? 0x7: 0x00;
        e[2] = (a & 0x1) ? 0x7: 0x00;

        for (k = 0; k < 3 ; k++){
                for (j = k ; j < 3 ; j++){
                        if (c[j] & (0x4 >> k)){
                                break;
                        }
                }
                if (j >= 3){
                         return(0);
                }
                tmp = c[k] ,c[k] = c[j], c[j] = tmp;
```

```
                tmp = e[k] ,e[k] = e[j], e[j] = tmp;
                for(i = 0 ; i < 3 ; i++){
                        if ((i != k) && (c[i] & (0x4 >> k))){
                                c[i] = c[k] ^ c[i];
                                e[i] = e[k] ^ e[i];
                        }
                }
        }
        return((e[0] & 0x01) ^ (e[1] & 0x2) ^ (e[2] & 0x4));
}
int
main()
{
        int a,b;
        for(a=0 ; a<8; a++){
                for(b=0 ; b<8; b++){
                        printf("%d / %d = %d\n", a, b, div(a,b));
                }
        }
        return(0);
}
```