マイクロコンピュータによる Fire 符号の高速復号法†

岡野博一††

マイクロコンピュータが飛躍的に発展し，データ通信の端末としても用いられている．したがって，通信回線の信頼度を高めるために，符号理論の手法をマイクロコンピュータのソフトによって実現することができれば，経済的で柔軟なデータ通信システムが構成され得る．

本論文において，マイクロコンピュータによる Fire 符号の高速復号法として3つの方法を提案している．

第1に，高速シフト演算法は中国人の剰余定理を用いて復号を行う．本質的には，テーブルは必要としないが，演算時間は長くなる．

第2に，圧縮形テーブル・ルックアップ法はシンドロームを求める演算を剰余表を用いて行い，誤りの訂正は誤りパターンのみを格納している圧縮復号表を用いて行う．復号時間が非常に短くなるが訂正可能バースト長が長くなると圧縮復号表が大きくなる．

第3に，ハイブリッド演算法はシンドロームを求める演算を剰余表を用いて行い，誤りの訂正は高速シフト演算法と同様にして行う．この場合，復号時間は短く，小さいテーブルを必要とするのみである．

さて，以上3つの方法のうちで，ハイブリッド演算法が復号時間，メモリ容量等総合的に評価した場合，最も優れており，マイクロコンピュータのソフトによる実用化が可能である．

1. まえがき

最近，通信や計算機システムなどのディジタルシステムに符号理論が実用されるようになった．すでに実用に供されているのは，計算機の主記憶に用いられている単一誤り訂正/2重誤り検出符号など誤り訂正能力の低いものが主である．通信，監視制御関係において広く実用化されている符号も2重あるいは3重誤り訂正符号にとどまっている．しかし，より高い訂正能力を持つとともに復号の簡単な符号の研究がなされている[5),6)]．これらの研究は主としてハード面から検討されている．一方，近年，マイクロコンピュータはデータ通信の端末としても用いられるようになってきており，マイクロコンピュータのソフトによって符号理論を実現させることも重要な課題である．

筆者は，主としてソフト面から符号理論の研究を行っており，すでに，PL/1 およびマイクロコンピュータのアセンブリ言語を用いて Fire 符号のソフト化を行った[9),11)]．ソフトによれば，特に有効なテーブルを用いることにより簡単な復号が行える可能性がある．

本論文において，マイクロコンピュータを用いた Fire 符号の高速復号法として，高速シフト演算法，圧縮形テーブル・ルックアップ法，ハイブリッド演算法を提案し，ソフトによる実用化が可能であることを示している．

なお，使用した言語はマイクロコンピュータ PFL-16A のアセンブリ言語である．

2. Fire 符号と高速復号法

Fire 符号の詳細については省略する[1)~4)]．本論文で用いた Fire 符号の生成多項式は次式で表わされる．

$$\begin{aligned} P(x)&=P_m(x)\cdot(x^c+1)\\ &=(x^3+x+1)\cdot(x^5+1)\\ &=x^8+x^6+x^5+x^3+x+1 \end{aligned} \tag{1}$$

この符号はバースト誤り訂正可能長 $b=3$，$P_m(x)$ の位数 $e=7$，$c=5$，最大符号長 $n_{max}=35$ である．

ハードによる Fire 符号の復号法を簡単に説明すると，まず，入力符号を $p(x)$ で割って，エラーパターンが一周期遅れて発生するのを利用して復号する方法がある．

つぎに，入力符号を $P_m(x)$，x^c+1 で別々に割り算するとエラーパターンが e シフトごとおよび c シフトごとに発生するのを利用し，両者の剰余の一致を検出することによって復号する方法がある．しかし，これらの方法はどちらも最大 $2n_{max}$ のシフト演算を必要とする．

つぎに，後者の復号法を発展させた高速復号法を説明する．入力符号を x^c+1 で割った剰余（シンドローム）$S_c(x)$ は周期 c でパターンを巡回する．したがって，$S_c(x)$ を x^c+1 で割り算をしてゆき，シンドロームの $c-b$ 個の下位ビットがすべて零となったとき，シンドロームの上位 b ビットにエラーパターン

† High Speed Decoding Methods of Fire Codes by a Microcomputer by HIROKAZU OKANO (Tokuyama Technical College Department of Information Electronics).

†† 徳山工業高等専門学校情報電子工学科

$B(x)$ が検出でき，さらにその時の演算回数より r_c $(0\leq r_c<c)$ が算出される．エラーパターン位置を i とすると，次式が成立する．

$$i=r_c(\mathrm{mod}\ c) \qquad (2)$$

つぎに，$P_m(x)$ に対するシンドロームが $B(x)$ となるシフト回数より $r_p(0\leq r_p<e)$ が算出され，次式が成立する．

$$i=r_p(\mathrm{mod}\ e) \qquad (3)$$

このとき中国人の剰余定理より i は次式で求まる．

$$i=A_c e r_c+A_p c r_p(\mathrm{mod}\ n_{max}) \qquad (4)$$

ここで，A_c，A_p は次式を満足する整数である．

$$A_c e+A_p c=1(\mathrm{mod}\ n_{max}) \qquad (5)$$

(1)式の生成多項式を用いる場合，$A_c=A_p=3$ であるから次式が成立する．

$$i=21r_c+15r_p(\mathrm{mod}\ n_{max}) \qquad (6)$$

この方法によれば，$P(x)$ での剰余を求めてから，最大 $c+e$ 回のシフトで復号が行われる．

3. シフト演算法による Fire 符号の高速復号法

前項で概説した Fire 符号の高速復号法に基づいてマイクロコンピュータによるシフト演算法[8)~11)]を用いた符号化および復号について述べる．

3.1 符号化

いま，符号長 32 ビット，入力符号長 24 ビット，チェックビット長 8 ビットである．入力符号 24 ビットにさらに 0 を 8 個付け加えたものを生成多項式に対応する JOSU=101101011 で割った剰余を求めれば，それがチェックビットであるから，8 個の 0 の所に剰余を MOD 2 で加算すれば，符号化が行われる．演算は文献10) 11) のシフト演算法と同一であるので，詳細は省略する．ただし，符号長がレジスタ長より長いので最初の 16 ビットに対する剰余 8 ビットを求め，この剰余に続く 8 ビットの入力符号を加算してさらにその剰余を求める．この方法を用いれば，符号長が長い場合でも 16 ビットごとに剰余を求め，続く入力符号を足し込むという演算を繰返し行うことによって，剰余を算出することができる．

3.2 高速シフト演算法による復号

高速復号法をソフトで実現する際には，少し工夫がいる．すなわち，ハードでは，$P_m(x)$ と x^c+1 で並列に割り算を行うが，いま CPU が 1 つなので並列演

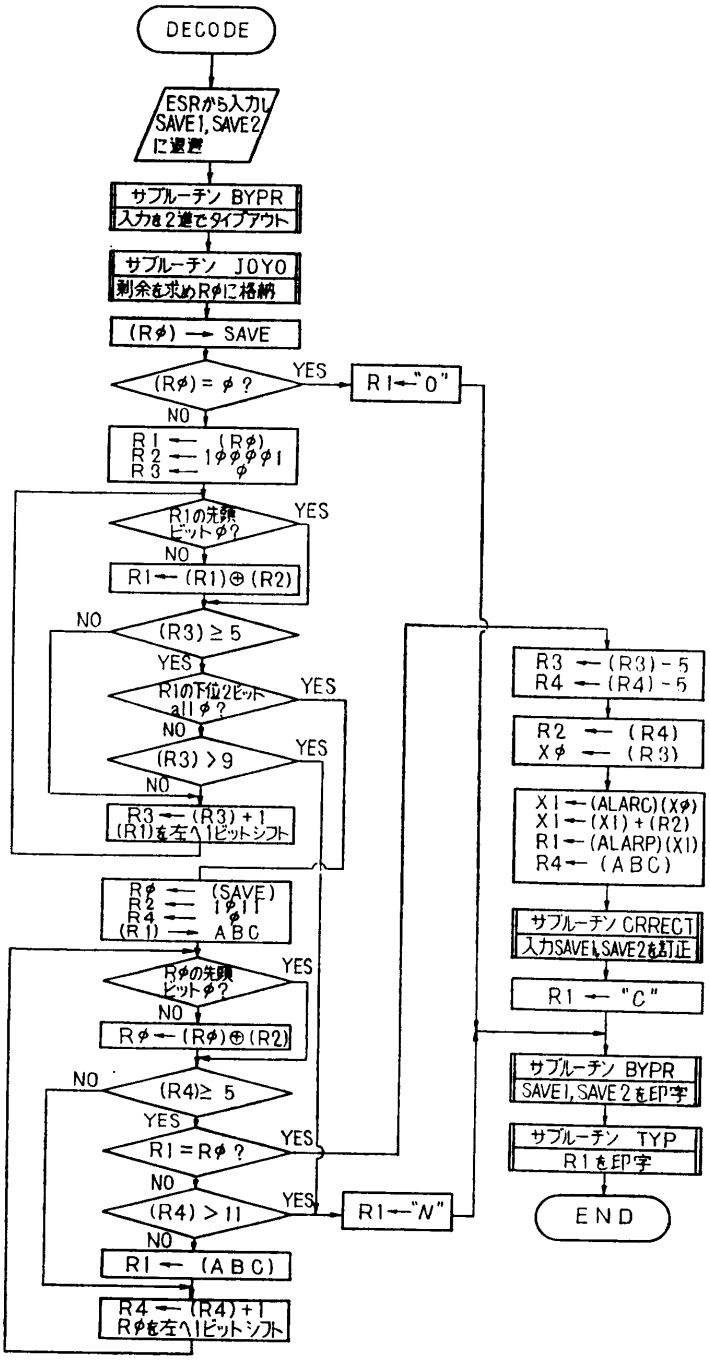

図 1 復号フローチャート（高速シフト演算法）
Fig. 1 Flow chart of the decoding program using high speed shift calculation method.

算ができないので，高速復号法のメリットが少なくなる．そこで，入力符号をまず $p(x)$ で割り剰余を求める．ついでこの剰余を $p_m(x)$，x^c+1 で割り算してそれぞれのシンドロームを求める．このシンドロームは上記並列演算によって求めたものと同一であることは明らかである．

つぎに，これらのシンドロームから r_c, r_p を求め，(6)式から誤り位置 i を求める．演算速度を早めるために，r_c, r_p を与えて(6)式をあらかじめ計算しておき，誤り位置検出表として格納しておく．いま，r_c は 0～4，r_p は 0～6 の値をとるので，誤り位置算出表は 35 語のテーブルである．

```
STNO. LABEL    OP.    OPERANDS/COMMENT
   4           BGN    DECODE,,X'0'
   5           L      SP,SPAD
   6           RD     R0,X'04'
   7           MV     R1,R0
   8           H
   9           RD     R0,X'04'
  10           MV     R2,R0
  11           ST     R1,SAVE1
  12           ST     R2,SAVE2
  13           MVI    R0,X'0A'
  14           BAL    (TYP)
  15           BAL    (TYP)
  16           MV     R0,R1
  17           BAL    (BYPR)
  18           MV     R0,R2
  19           BAL    (BYPR)
  22           BAL    (JOYO)
  24           ST     R0,SAVE
  25           SKIP   R0,NZ
  26           B      NOERR
  27           MV     R1,R0
  28           L      R4,N10
  29           L      R2,N51
  30           CLEAR  R3
  31  P0       TBIT   R1,0,Z
  32           EOR    R1,R2
  33           L      R0,N5
  34           S      R0,R3,MZ
  35           B      P4
  36           MV     R0,R1
  37           AND    R0,R4,NZ
  38           B      P5
  39           L      R0,N9
  40           S      R0,R3,PZ
  41           B      NOCRCT
  42  P4       AI     R3,1
  43           SL     R1,RE
  44           B      P0
  45  P5       ST     R3,S1
  46           L      R0,SAVE
  47           L      R2,N310
  48           CLEAR  R4
  49           ST     R1,ABC
  50  P7       TBIT   R0,0,Z
  51           EOR    R0,R2
  52           L      R3,N5
  53           S      R3,R4,MZ
  54           B      P8
  55           S      R1,R0,NZ
  56           B      CRCT
  57           L      R3,N11
  58           S      R3,R4,PZ
  59           B      NOCRCT
  60           L      R1,ABC
  61  P8       AI     R4,1
  62           SL     R0,RE
  63           B      P7
  67  CRCT     L      R3,S1
  68           L      R2,N5
  69           S      R3,R2
  70           S      R4,R2
  71           MV     R2,R4
  72           MV     X0,R3
  73           L      X1,(ALARC)(X0)
  74           A      X1,R2
  75           L      R1,(ALARP)(X1)
  76           L      R4,ABC
  77           BAL    (CRRECT)
  78           MVI    R1,X'C3'
  79           B      P9
  80  NOERR    MVI    R1,X'CF'
  81           B      P9
  82  NOCRCT   MVI    R1,X'4E'
  83  P9       MVI    R0,X'0A'
  84           BAL    (TYP)
  85           L      R0,SAVE1
  86           BAL    (BYPR)
  87           L      R0,SAVE2
  88           BAL    (BYPR)
  89           MVI    R0,X'A0'
  90           BAL    (TYP)
  91           MV     R0,R1
  92           BAL    (TYP)
  93           H
  97           DS     XL24
  98  SPAD     DC     A(*-1)
  99  *
 100  N51      DC     X'8400'
 101  N310     DC     X'8000'
 102  N10      DC     X'0C00'
 103  N9       DC     F'9'
 104  N5       DC     F'5'
 105  N11      DC     F'11'
 106  SAVE     DS     XL2
 107  ABC      DS     XL2
 108  S1       DS     XL2
 109  BYPR     EXTRN  BYPR
 110  TYP      EXTRN  TYP
 111  JOYO     EXTRN  JOYO
 112  CRRECT   EXTRN  CRRECT
 113           GLOBAL SAVE1,SAVE2
 114  SAVE1    DS     XL2
 115  SAVE2    DS     XL2
 116  ALARP    DC     A(*+1)
 117           DC     F'0'
 118           DC     F'15'
 119           DC     F'30'
 120           DC     F'10'
        (テーブル一部省略)
 150           DC     F'19'
 151           DC     F'34'
 152  ALARC    DC     A(*+1)
 153           DC     F'0'
 154           DC     F'7'
 155           DC     F'14'
 156           DC     F'21'
 157           DC     F'28'
 158           END
```

(出力例)

```
11110000 11110000 11110000 10001101
11110000 11110000 11110000 10001101  O

11110000 11110001 00110000 10001101
11110000 11110000 11110000 10001101  C

11110000 11110011 00110000 10001101
11110000 11110011 00110000 10001101  N
```

図 2 復号プログラムとその出力例(高速シフト演算法)

Fig. 2 The decoding program using high speed shift calculation method and its output data.

さて，**図1**に高速シフト演算法による復号のフローチャートを示す．受信入力符号32ビットは16ビットずつエントリースイッチ ESR より入力し，SAVE 1 SAVE 2 に退避しておく．$n_{max}=35$ なので，入力符号に0を3つ付加し 35 ビットの入力符号と考えてサブルーチン JOYO によって生成多項式に対応する JOSU=101101011 で割った剰余を求めレジスタ R0 に格納する．剰余を求める演算は符号化の場合と同じであるのでプログラム等は省略する．文献 10)，11) を参照されたい．さて，R0が0のとき誤りなし，R0 が0でないならば，R0 の内容をR1 に移し，x^5+1 での剰余を求めるためにR2 に 100001 を格納する．R1 を R2 で割り算し，剰余の下位2ビットが all 0 sのとき上位3ビットにバースト誤りパターン $B(x)$ が求まるので，ABC 番地に退避する．このときの演算回数より r_e が求まる．

次に，$P_m(x)$ での剰余を求めるために R2に 1011 を格納し，R0をR2 で割り算しその剰余が ABC 番地に格納した $B(x)$ のパターンと一致したときの演算回数より r_p が求まる．そして，メインプログラムの定数定義域の ALARP から格納しておいた誤り位置算出表を r_e, r_p によってテーブル・ルックアップして誤り位置 i を求めR1 に格納する．

つぎに，ABC 番地のエラーパターン，R1の誤り位置 i をもとにサブルーチン CRRECT によって誤りの訂正を行う．

図2に高速シフト演算法による復号プログラムリストとその出力例を示す．JOYO, CRRECT 等のサブルーチンは紙面の都合で省略する．出力例のOは誤りなし，Cは訂正，Nは訂正不可能を示している．

4. テーブル・ルックアップ法による Fire 符号の高速復号法

マイクロコンピュータによって Fire 符号演算を行う場合，テーブルを使用すれば効率よく高速に演算することができる．これには剰余表と復号表または圧縮復号表を用いる．

4.1 符号化

符号化の演算は剰余表を用いて行う．剰余表は入力8ビットと入力8ビットに8個の0を付加したものを生成多項式で割ったときの剰余との対応表である．ここでは，剰余表の作成等については省略する．剰余表は，復改コードの後に入力8ビットパターン，スペースコードの後に剰余がそれぞれ 16 進でパンチしてある紙テープとして入力し，サブルーチン AA 16で2進に変換する．そして，テーブルの先頭番地 $(300)_{16}$ に8ビット入力符号を加えた番地にその剰余を格納する．したがって剰余表は $(300)_{16}$〜$(3FF)_{16}$ 番地に格納される．

符号化するには，まずエントリースイッチ ESR よりの入力符号 24 ビットを SAVE 1, SAVE 2 に格納し，SAVE 1 の上位8ビットをインデックスレジスタ X0 に格納し，X0 に剰余表の先頭番地を加えた番地の内容をR0 に格納する．R0 の内容は入力8ビットに対する剰余である．つぎに，SAVE 1 の下位8ビットを先に求めた剰余に加え新たな入力8ビットと考えて剰余を求める．この操作を3回繰返せば，8ビットのチェックビットを求めることができる．

図3は剰余表をメモリダンプしたものである．なお，符号化プログラム等は省略するが，同様の演算が復号プログラムにおいて使用されている．

4.2 標準形テーブル・ルックアップ法による復号

まず，符号化と同様にして剰余表を用いて入力符号の剰余を求める．そして復号表を用いて誤りの訂正を行う．この方法は文献 10)，11) のテーブル・ルックアップ法と同様であり，復号表とは誤りパターンとその剰余（シンドローム）との対応表である．一般にこの標準形テーブル・ルックアップ法による復号表は，チェックビット長を k とすると 2^k 語を必要とする．いまの場合，256 語である．この方法は，上記文献および次項を参照すれば容易に実現できるのでプログラム等は省略する．

4.3 圧縮形テーブル・ルックアップ法による復号

4.3.1 圧縮復号表

復号表にはエラーパターンのみを格納することが望ましい．高速復号法の原理を用いて復号表を小さくして圧縮復号表とすることができる．

すなわち，高速復号法は入力符号を x^e+1, $P_m(x)$, で割った剰余 JOY 1, JOY 2 によって誤りの訂正ができることを示している．そして JOY 1 により訂正

```
=MDUMP 300,400

(0300)  0000  006B  00D6  00BD  00C7  00AC  0011  007A
(0308)  00E5  008E  0033  0058  0022  0049  00F4  009F
                        (省  略)
(03E8)  003A  0051  00EC  0087  00FD  0096  002B  0040
(03F0)  007E  0015  00A8  00C3  0089  00D2  006F  0004
(03F8)  0093  00F0  004D  0026  005C  0037  008A  00E1
```

図3 剰余表（メモリダンプ）
Fig. 3 The remainder table (Memory dump).

```
=MDUMP 400.4A0
(0400)  0000  001B  0085  0403  4001  0011  0201  0023
(0408)  0021  002B  8001  0019  0043  0803  0401  0601
(0410)  0063  003B  0031  0C03  0185  C001  0083  0041
                    (省　略)
(0478)  00C9  0000  3001  6003  0183  0303  0181  0381
(0480)  038B  0000  01C9  0703  E003  7001  01CB  00E9
(0488)  7003  0000  01C1  3801  0383  0000  0000  0000
```

図 4 圧縮復号表（メモリダンプ）
Fig. 4 The compessed decoding table (Memory dump).

の可否を識別することができる．いま，5ビットよりなる JOY 1 は 0 を除いて 1 から 31 まであり，訂正可能の場合は，JOY 1 のそれぞれが 3 ビットの JOY 2 との組合せで 0 を除いて 7 個のエラーパターンを持つ．

そして，JOY 1 が誤り訂正可能なのは 20 通りなので，20 行×7 列の圧縮復号表となる．JOY 1，JOY 2 はそれぞれ昇順にならべる．こうして 256 語の標準形復号表は誤りパターンのみの 140 語の圧縮復号表として紙テープに作成され，さらに，$(401)_{16}$〜$(48C)_{16}$ 番地に格納される．

図 4 に圧縮復号表のメモリダンプリストを示す．なお，$n_{max}=35$ の長さ 3 以下のバースト誤りは 140 種類（4 種類のパターン×35）なので，これ以上テーブルは圧縮できない．

図 5 に格納するエラー訂正用パターンを示す．誤りパターンとその位置情報とが格納され，符号長が長い場合でも一語に収めるようにするため，また復号時間を短くするための工夫がなされているが，詳細は省略する．

4.3.2 復 号 法

図 6 に圧縮形テーブル・ルックアップ法による高速復号フローチャートを示す．

前半は剰余表と圧縮復号表をメモリに格納するアルゴリズム，後半は復号処理を行うアルゴリズムである．剰余表は $(300)_{16}$〜$(3FF)_{16}$，圧縮復号表は $(401)_{16}$〜$(48C)_{16}$ 番地に格納される．

さて，つぎに誤り訂正処理について述べる．受信入力符号をエントリースイッチESRより入力し，上位 16 ビットを SAVE 1，下位 16 ビットを SAVE 2 に退避し，符号化の場合と同様にして剰余表をテーブル・ルックアップしながら剰余（シンドローム）を求める．剰余が 0 であれば誤りはない．0 でない場合は剰余をさらに JOSU 1=100001，JOSU 2=1011 でシフト演算し剰余 JOY 1，JOY 2 を求める．

そして，サブルーチン ST 10 は JOY 1，JOY 2 をもとに圧縮復号表より誤り訂正用パターンをテーブル・ルックアップする．すなわち，定数定義域の ST 1 よりの訂正指示表を JOY 1 でテーブル・ルックアップし，その JOY 1 の行（7 個の誤り訂正用パターンから成る）の圧縮復号表における先頭番地を X0 に格納する．X0 が 0 であれば誤り訂正はできない．X 0 が 0 でなければ X0 に JOY 2 を加えた番地の内容をテーブル・ルックアップし R0 に格納する．以上の動作で圧縮復号表より誤り訂正用パターンが R0 に格納されたので，続いてサブルーチン COORE によって誤り訂正用パターンをもとに誤り訂正を行う．サブルーチン COORE の動作は省略する．

図 7 に圧縮形テーブル・ルックアップ法による高速復号プログラムを示す．出力例は図 2 と同様なので省略する．また，**図 8** にサブルーチン ST 10 の中で使用される訂正指示表の一部を示す．なお，サブルーチン ST 10 のプログラムは省略する．

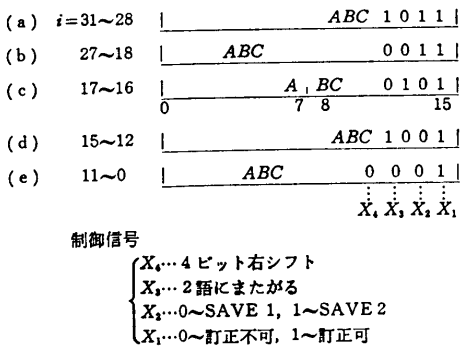

図 5 誤り訂正用パターン
Fig. 5 Patterns for error correcting.

5. 各種復号法の比較

前項までに述べてきたように，ソフトによる Fire 符号の復号法としてはシフト演算法，高速シフト演算

```
61  *****  TABLE ST1  *****
62  TBAD     DC     A(ST1)
63  ST1      DC     X'401'
64           DC     X'408'
65           DC     X'40F'
                (省　略)
89           DC     X'0'
90           DC     X'486'
91           DC     X'0'
92           DC     X'0'
93           DC     X'0'
```

図 8 訂正指示表（サブルーチン ST 10 用）
Fig. 8 The instruction table for error correcting (used in subroutine ST 10).

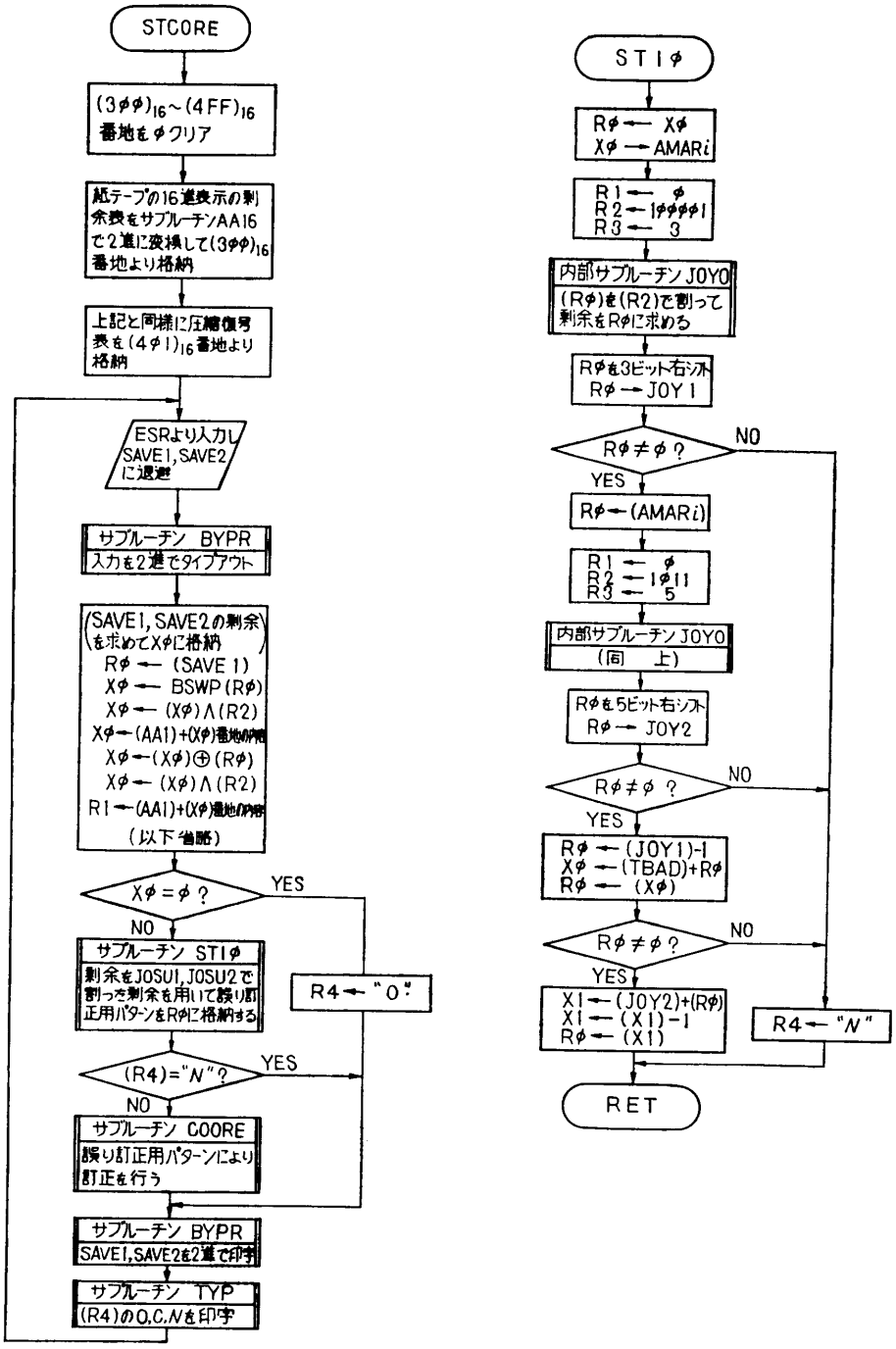

図 6 復号フローチャート（圧縮形テーブル・ルックアップ法）
Fig. 6 Flow chart of the decoding program using compressed table look-up method.

```
STNO. LABEL    OP.     OPERANDS/COMMENT
    1          BGN     STCORE,,X'0'
    2          L       SP,SPAD
    3          CLEAR   X1
    4          L       R0,N1
    5  BH      L       R1,N1
    6          CLEAR   X0
    7          S       R1,R0,Z
    8          B       B0
    9          L       R1,AA1
   10          B       B1
   11  B0      L       R1,AA2
   12  B1      ST      R1,AA
   13  B4      ST      X1,(AA)(X0)
   14          L       R1,BB
   15          S       R1,X0,Z
   16          B       B2
   17          B       B3
   18  B2      AI      X0,1
   19          B       B4
   20  B3      L       R1,N1
   21          S       R0,R1
   22          SKIP    R0,M
   23          B       BH
   24          L       R1,N2
   25          ST      R1,DATA
   26  F0      H
   27  F1      BAL     (PTR16)
   28          CLEAR   R1
   29          MVI     R1,X'0A'
   30          S       R1,R0,Z
   31          B       F1
   32          BAL     (PTR16)
   33          CLEAR   R1
   34          MVI     R1,X'A5'
   35          S       R1,R0,Z
   36          B       F2
   37          L       R1,AA1
   38          B       F3
   39  F2      CLEAR   R1
   40          MVI     R1,X'A6'
   41          S       R0,R1,Z
   42          B       ERR
   43          L       R1,AA2
   44  F3      ST      R1,AA
   45  F4      MVI     R2,X'0A'
   46          BAL     (AA16)
   47          MV      X0,R2
   48          L       R1,BB
   49          S       R2,R1,MZ
   50          B       F5
   51          MVI     R2,X'A0'
   52          BAL     (AA16)
   53          MV      R4,R2
   54          ST      R4,(AA)(X0)
   55          B       F4
   56  F5      L       R1,DATA
   57          L       R2,N1
   58          S       R1,R2
   59          ST      R1,DATA
   60          SKIP    R1,Z
   61          B       F0
   62  B11     H
   63          RD      R0,X'04'
   64          ST      R0,SAVE1
   65          H
   66          RD      R0,X'04'
   67          ST      R0,SAVE2
   68          MVI     R0,X'0A'
   69          BAL     (TYP)
   70          BAL     (TYP)
   71          L       R0,SAVE1
   72          BAL     (BYPR)
   73          L       R0,SAVE2
   74          BAL     (BYPR)
   75          L       R4,AA1
   76          ST      R4,AA
   77          L       R2,FF
   78          L       R0,SAVE1
   79          BSWP    X0,R0
   80          AND     X0,R2
   81          L       X0,(AA)(X0)
   82          EOR     X0,R0
   83          AND     X0,R2
   84          L       R1,(AA)(X0)
   85          L       R0,SAVE2
   86          BSWP    X0,R0
   87          EOR     X0,R1
   88          AND     X0,R2
   89          L       X0,(AA)(X0)
   90          EOR     X0,R0
   91          AND     X0,R2
   92          SKIP    X0,NZ
   93          B       OK
   94          BAL     (ST10)
   95          CLEAR   R1
   96          MVI     R1,X'4E'
   97          S       R1,R4,NZ
   98          B       PRI
   99          BAL     (COORE)
  100  PRI     MVI     R0,X'0A'
  101          BAL     (TYP)
  102          L       R0,SAVE1
  103          BAL     (BYPR)
  104          L       R0,SAVE2
  105          BAL     (BYPR)
  106          MVI     R0,X'A0'
  107          BAL     (TYP)
  108          MV      R0,R4
  109          BAL     (TYP)
  110          B       B11
  111  OK      MVI     R4,X'CF'
  112          B       PRI
  113  ERR     MVI     R0,X'3F'
  114          BAL     (TYP)
  115          BAL     (TYP)
  116          B       BH
  117  N1      DC      X'1'
  118  N2      DC      X'2'
  119  AA1     DC      X'300'
  120  AA2     DC      X'400'
  121  BB      DC      F'255'
  122  FF      DC      X'00FF'
  123  AA      DS      XL2
  124          GLOBAL  SAVE1,SAVE2
  125  SAVE1   DS      XL2
  126  SAVE2   DS      XL2
  127  DATA    DS      XL2
  128          DS      XL30
  129  SPAD    DC      A(*-1)
  130  PTR16   EXTRN   PTR16
  131  AA16    EXTRN   AA16
  132  COORE   EXTRN   COORE
  133  TYP     EXTRN   TYP
  134  BYPR    EXTRN   BYPR
  135  ST10    EXTRN   ST10
  136          END
```

図 7 復号プログラム（圧縮形テーブル・ルックアップ法）

Fig. 7 The decoding program using compressed table look-up method.

表 1 Fire 符号の復号法の比較
Table 1 Decoding performance of some Fire codes.

訂正可能バースト長		3	5	7
生成多項式		$(x^3+x+1)(1+x^9)$	$(x^5+x^2+1)(1+x^9)$	$(x^7+x^3+1)(1+x^{13})$
チェックビット長		8	14	20
最大符号長		35	279	1651
シフト演算法	復号時間の上限	4.0(1.5)	31.6(12.0)	186.6(71.0)
標準形テーブル・ルックアップ法	復号時間の上限	0.18(0.07)	0.75(0.65)	10.4(10.3)
	剰余表	256	256	512
	復号表	256	16K	2M
圧縮形テーブル・ルックアップ法	復号時間の上限	0.53(0.07)	1.3(0.65)	11.2(10.3)
	剰余表	256	256	512
	訂正指示表	32	512	8192
	復号表	140	4.5K	212K
高速シフト演算法	復号時間の上限	2.2(1.0)	12.5(9.6)	145(137)
	誤り位置算出表	35	279	1651
ハイブリッド演算法	復号時間の上限	1.2(0.07)	3.6(0.65)	18(10.3)
	剰余表	256	256	512
	誤り位置算出表	35	279	1651

(注 1) 復号時間の上限 ( ) 内は剰余算出時間, 単位 ms.
(注 2) 各表の大きさの単位は語, Kは 10^3, M は 10^6

法, 標準形テーブル・ルックアップ法, 圧縮形テーブル・ルックアップ法がある. ここで, これらの復号法を比較し, 最適な復号法を検討することとする. さて, 復号時間を短くするには標準形テーブル・ルックアップ法がよいが, 復号表のために大きなメモリが必要である. 高速シフト演算法は前者より復号時間が長くなるが, 誤り位置算出表のための少しのメモリを必要とするのみである. そして, 高速シフト演算法の復号時間は剰余算出時間で占められ符号長が長くなるほどこの傾向は著しい. これらのことから最適復号法は剰余の算出を剰余表を使ったテーブル・ルックアップ法により行い, それ以後の誤りの訂正を高速シフト演算法によって行う方法であることが分かる. この方法をハイブリッド演算法と呼ぶこととする.

表 1 に以上述べた復号法の比較を示す. なお, 訂正可能バースト長が 7 の場合, チェックビットが 1 語長 (16 ビット) を越えるので, 各テーブルの 1 パターンが 2 語長となり, シフト演算も倍長演算を仮定して演算時間を計算してある.

さて, 4800 ボーの通信回線を例にとると 1 パルス幅は 208 μs である. 一方, 訂正可能バースト長 5 のとき, 最大符号長を用いたとすると, 入力符号 1 パルス当りの復号時間はハイブリッド演算法によると約 13 μs であり, このとき 256 語の剰余表と 279 語の誤り位置算出表を用いるのみで良い.

したがって, ハイブリッド演算法は 4800 ボーおよび 48K ボーの高速データ通信回線に対しても有効であり, ソフトによる Fire 符号の実用化が充分に期待される.

6. むすび

Fire 符号の高速復号法として, 復号時間, メモリ容量等総合的に評価した場合, ハイブリッド演算法が最も優れており, 高速データ通信回線に対して, ソフトによる実用化が可能であることを示した.

さらに, BCH 符号等に対しても, ソフト化の検討を行いたい. また, ROM, マイクロプログラム技術を応用したより高速な復号法を開発したいと思う.

最後に, 日頃ご指導頂く広島大学工学部川野董教授に深甚なる謝意を表すとともに, プログラムの作成に協力頂いた本学守川和夫講師, 学生桂哲也君, 山本聰君に感謝の意を表します.

参考文献

1) Peterson, W. W. and Weldon, E. J.: Error-Correcting Codes, 2nd Edition, MIT press, Mass (1972).
2) Berlekamp, E. R.: Algebraic Coding Theory, McGraw-Hill Book co., New York (1968).
3) 宮川, 岩垂, 今井: 符号理論, 昭晃堂 (1973).
4) 嵩, 都倉, 岩垂, 稲垣: 符号理論, コロナ社 (1975).
5) 今井, 藤谷: 復号の簡単な誤り訂正符号の一構成法, 信学論 (A), J 62-A, 5 (1979).
6) 杉村, 笠原, 滑川: 復号の容易な能率の良い多重誤り訂正符号の一構成法, 信学論(A), J 61-A, 11 (1978).
7) 桑原, 西野, 岡野, 米岡: Fire Code を用いた無線印刷電信方式, 信学論 (B), 56-B, 7 (1973).
8) 岡野: ソフトプログラム(HPL)による Fire 符号の演算とその解析, 信学技報, AL 76-33 (1976).
9) 岡野: PL/1 による Fire 符号のシミュレーションプログラム, 情報処理, Vol. 18, No. 12 (1977).
10) 岡野: マイクロコンピュータによる Fire 符号のシミュレーションプログラム—データ通信への応用—, 信学技報, CS 79-145, No. 8 (1978).
11) 岡野: マイクロコンピュータによる Fire 符号のシミュレーションプログラム—データ通信への応用—, 情報処理学会論文誌, Vol. 20, No. 6 (1979).

(昭和 54 年 11 月 19 日受付)
(昭和 55 年 5 月 15 日採録)