計算機アーキテクチャ 32－4
（1978. 9. 13）

ファームウェアジェネレータシステムの構成

三上和敬　房岡璋
（三菱電機中央研究所）

1．まえがき

書換え可能な制御記憶を実装したマイクロプログラム制御方式の計算機はユーザマイクロプログラミングを可能にし，個々の利用目的に適合した計算機システムをマイクロプログラムで実現でき，その計算機の処理能力を大幅に向上させることが可能となる。しかし，一般のユーザがこのような効果を十分に発揮させるマイクロプログラムを作成しようと試みる際に，つぎのような問題に直面せざるをえない。

（1）マイクロプログラムの記述に要求される計算機の詳細なハードウェアに関する知識
（2）計算機の能力を最大限発揮させるマイクロプログラムの最適化技術
（3）マイクロプログラムのデバッグ法や，マイクロプログラムの自由解放に伴うシステムの混乱の防止

これらの問題点に関しては，つぎのような研究がなされてきた。

マイクロプログラムの記述に関する研究は，初期の機械表現形式からアセンブラ語形式へと移り，さらに高級言語形式へと発展している。マイクロプログラム記述用の高級言語[1]としては，MPL，SIMPL，PUMPKINなどがある。これらの高級言語はハードウェアに関係する部分を宣言する必要があったり，多くは記述レベルがハードウェアに近いものである。他方，問題向き言語をそのまま使用する例として，PL/MP[2]があり，本稿で述べるシステムもこれと同じ方向である。

つぎにマイクロプログラムの最適化の研究は，KLEIR[3]らが一般的手法についてまとめている。これらの方法は一般的であるが故に，実際の適用にあたってはそのままの形式で実現できるものは少ない。また最適化は適用計算機のマイクロ命令やハードウェアの構造上の特徴を十二分に活用する技術であるため，適用機種毎に最良の実現方式を考えなければならない。

つぎにオミの問題点の対策としては，マイクロ命令レベルの解放ではなく，高級言語レベルでの解放によって対処する方式が望ましいとされている。[4]

本稿では，上記の問題点を背景として，一般ユーザに適用計算機のハードウェア知識の習得を強要することなしに，高級言語レベルの記述であって，マイクロプログラムを得られるファームウェアジェネレータシステムの構成に関して述べる。ファームウェアジェネレータシステム（FGSと略記す）は，PL/I系の高級言語で記述されたプログラムをMELCOM-COSMO 500のマイクロアセンブラのソースプログラムに変換するトランスレータである。

2．FGSの目的

FGS開発の主目的は，計算機の詳細なハードウェア構造に関して熟知する必要なく，マイクロプログラムを能率よく作成する手段を提供することである。そのためには自動的にレジスタ割付けや最適化を行なう必要がある。また，マイクロアセンブラのソースプログラムを出力するので，人手による高度の最適化

が可能である。 したがって専門的マイクロプログラマにとってはマイクロプログラムの作成サポート手段となりうる。 その他，マイクロプログラムを高級言語レベルで解放することにより，誤ったマイクロプログラムによるシステム破壊を抑止する効果もある。

3. 言語仕様

FGSの入力用言語としては，PL/I系の言語仕様をもつプログラミング言語を選択した。 その基本言語仕様は表3.1に示している。 またPL/Iから除外した主な機能は表3.2の通りである。 この言語で記述されたプログラム例を図3.1に示す。

4. マイクロコードコンパイラの構成

4.1 設計上の留意事項

計算機に依存しないプログラミング言語形の高級言語から，実行効率のよいマイクロプログラムに変換することは難しい問題である。 FGSはこの実行効率の低下に対処するため，システム内部で様々な工夫を行なった。 FGSの設計上の主な留意事項は次の点である。

表 3.1 言語仕様概要

(a) データ構成	スカラ	
	配列	(Max 2次元)
	構造体	(Max 3レベル)
(b) データ属性	整数	(16ビット)
	文字	
	ビット	
(c) 算術演算子	加，減，乗，除，剰余	
(d) ビット演算子	論理積，論理和，排他的論理和	
(e) 比較演算子	EQ, NE, LT, LE, GE, GT	
(f) 記憶割付属性	主記憶，制御記憶	
(g) 割付境界属性	32ビット，16ビット，8ビット	
(h) 初期属性	2進，10進，16進数，文字	
(i) 手続き	サブルーチン型手続き	
(j) DO文のネスティング	Max 4重	
(k) 文	代入文，CALL文	
	宣言文，DO文	
	END文，GOTO文	
	IF文，PROC文	
	RETURN文，PEND文	
(l) 特殊文	インラインマイクロアセンブリ文	
	レジスタ使用禁止文	
	レジスタ使用解除文	
	RETURNA文	

表 3.2 主な除外機能

(a)	動的割付け
(b)	入出力機能
(c)	浮動小数点演算，複素演算
(d)	割込み処理

```
                                        SOURCE LISTING

        /*  PROGRAM NO=25   */
 1  MAIN:  PROC;
 2         DCL A(100)    MM FIXED,
                SW1      MM FIXED,
                SW2      MM FIXED,
                X        MM FIXED,
                Y        MM FIXED,
                W        MM FIXED;
 3  L0:    SW1=0;
 4         SW2=0;
 5         DO X=1 TO 99 BY 2;
 6         IF A(X) < A(X+1) THEN GOTO L1;
 7                          ELSE DO;
 8                               SW1=1;
 9                               W=A(X);
10                               A(X)=A(X+1);
11                               A(X+1)=W;
12                               END;
13 L1:    END;
14        DO Y=2 TO 98 BY 2;
15        IF A(Y) < A(Y+1) THEN GOTO L2;
16                         ELSE DO;
17                              SW2=1;
18                              W=A(Y);
19                              A(Y)=A(Y+1);
20                              A(Y+1)=W;
21                              END;
22 L2:    END;
23        IF SW1=1 THEN GOTO L0;
24                 ELSE GOTO L3;
25 L3:    IF SW2=1 THEN GOTO L0;
26                 ELSE GOTO L4;
27 L4:    RETURNA;
28        PEND;
```

図 3.1 プログラム例

(1) 中間言語として高級言語の文の意味を保存し，かつ適用計算機のハードウェアとの関係づけやマイクロ命令への対応づけのとれたものを選択した。

(2) FGSの内部処理単位はソースステートメントレベルの1ステートメントではなく，シーケンシャルに連続したステートメントの集合とする。
この集合はセクション（4.4参照）と呼ばれ，変数の冗長なメモリーレジスタ間転送を検出するために用いる。

(3) FGSの目的プログラムの形式は，直接手続きを実行する形式と，マイクロルーチンを利用する形式との混合とする。 マイクロルーチンの活用はマイクロプログラムのサイズを短縮できる。

(4) FGSのマイクロジェネレータは，半完成のマイクロ命令シーケンス（これをマイクロスケルトンと呼ぶ）を母体として内蔵し，これに不足情報を付加することで個々のマイクロ命令を生成する。 マイクロスケルトン方式の利点は，並列処理やサブファンクションの先行処理などの組み込み処理が容易となることである。

4.2 基本構成

FGSは4.1で述べた事項を実現するため，図4.1に示す基本構成をとる。

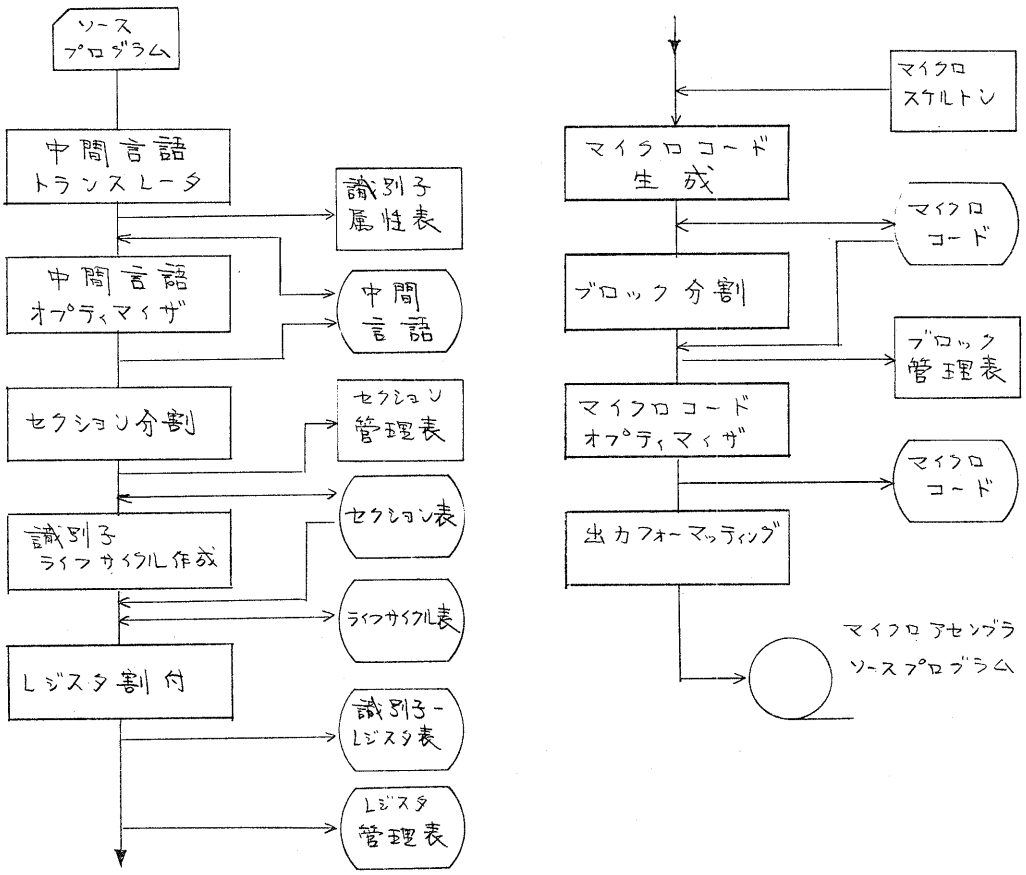


図 4.1 FGSの基本構成

4.3 中間言語

(1) 中間言語の形式

ソーステートメントは一連の中間言語の系列にコンパイルされる。中間言語の形式は，図4.2に示すようなレコード構成で，その主要部は展開表現部とソーステートメントの情報部である。展開表現部は中間言語の命令(10種類)毎に定められた形式で表4.3のように表わされる。なお，アサイン命令の一般形としては，三オペランド形式を用いた。オペランドはユーザ定義の識別子か，またはFASSが生成する内部識別子である。内部識別子には，定数や作業用変数がある。その他，配列M(3,7)のように添字計算可能なものを，スカラ系識別子として取扱うための特殊内部識別子などがある。

(2) 中間言語命令の組合せ

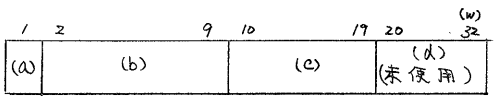
中間言語トランスレータの出力における特徴は，つぎの事項である。

- スカラ項目のみの式のコンパイル結果はアサイン命令のみの組合せとなり，ロード命令，ストア命令を含まない。

このようにした理由は，スカラ系識別子のロード/ストア操作を中間言語で指示するよりも，レジスタ状況で時期を判定する方が，最適化の面で得策であるからである。なお，配列を含む式では中間言語の指示によりロード/ストア操作を行なう。図4.3はソーステートメントの中間言語への変換例を示している。

(3) ハードウェアとの対応関係

中間言語の形式とハードウェアとの対応関係は，原則として数値属性をもつ変数系の識別子に1個のレジスタを対応づける。定数を表現する内部識別子は，マイクロ命令のコンスタントを利用するため，レジスタとの対応関係を持たない。また，ビット系，文字系の識別子は，FASSで定められた特定のマイクロルーチンのインタフェース用レジスタを割当てるので，割付け用の一般レジスタとの対応関係を持たない。



(a) レコード番号
(b) 展開表現部(中間言語命令，演算コード，三オペランド形式)
(c) 原文情報部(文番号，文種，DO LABEL, etc)

図 4.2 中間言語のレコード構成

表4.3 中間言語の展開表現部

命令	符号	シンボリック表現(注1)
アサイン1	as	<label> idi → idj
アサイン2	as	<label> idi ⊗ idj → idk
アドレス1	ad	<label> adi → adj
アドレス2	ad	<label> adi ⊗ adj → adk
ロード	ld	<label> (ad) → id
ストア	st	<label> id → (ad)
ジャンプ	gt	<label> label
判定	if	<label> idi ⊗ idj ◎ l1 l2
コール	cl	<label> name(p1 p2 ··· pn)
プロシージャ	pr	<label> name(p1 p2 ··· pn)
リターン	rt	<label>
リターンA	ra	<label>

(注1) 内部表現は16進数コード

```
LAB:  SUM = A * B + C - D;
```

```
as   lab   ida * idb → idt
as         idt + idc → idt
as         idt - idd → idsum
```

```
DO I=1 TO 15;
    SUM = SUM + I;
END;
```

```
as          '1'          → idi
as   l31    idsum + idi → idsum
as          idi + '1'    → idi
if          idi - '15' ≦ 0  l31 l32
---  l32    ---
```

図 4.3 中間言語変換例

4.4 セクション

セクションは，レジスタ割付けを行なう際の考慮対象範囲を設定した枠組である。 FGSではセクションとして，生成された中間言語の系列の中で，分岐を含まず直線的に実行可能なブループに線引きしたものを選定した。
したがって，セクションの分割は，中間言語系列に出現する分岐関係の命令や制御の流れを変える命令，即ち，*call*，*if*，*goto*，*proc*，*return*，などの命令とラベルを考慮に入れて行う。 セクション分割の結果，中間言語系列の全体は，一般的にセクションのリンク構造として表現され，セクション管理表で管理される。セクション分割の以後，セクションは内部処理の基本的単位として取扱われる。

4.5 識別子ライフサイクル表

識別子ライフサイクル表はレジスタ割付けの際の基礎資料とするためセクション毎に作成され，識別子がある値を保持している期間（中間言語のステップ数に対応）を示すものである。 また同表は識別子の参照状況や新値セット状況，およびある識別子が次に使用されるまでの期間も表示する。 なお，登録される識別子は，一般レジスタの割付け対象となるもので，数値系の属性を有するものに限られる。

中間言語

(1) '1' → a
(2) '2' → b
(3) '3' → c
(4) a * d → t_1
(5) t_1 + b → t_1
(6) t_1 − c → e
(7) a + c → f

	a	b	c	d	e	f	t_1
1	Ⓝ	○	○	○	○	○	○
2	2	Ⓝ	○	○	○	○	○
3	1	2	Ⓝ	○	○	○	○
4	Ⓑ	1	2	Ⓑ	○	○	Ⓝ
5	2	Ⓑ	1	○	○	○	Ⓡ
6	1	○	Ⓑ	○	Ⓝ	○	Ⓑ
7	Ⓑ	○	Ⓑ	○	○	Ⓝ	○

Ⓝ：新値記号 Ⓑ：参照記号
Ⓡ：更新記号 M：キョリ

図 4.4 識別子ライフサイクル表

4.6 レジスタ割付け

(1) 割付け対象レジスタ

FGS全体で使用したレジスタの使用用途，個数を表4.2に示す。 マイクロコードコンパイラの割付け対象レジスタは20個である。これらのレジスタは3群のレジスタファイルから構成され，ALUとの接続関係を図4.5に示す。同図において，レジスタファイルRFCは使用できるマイクロ命令の種類の制限があるため，RFA，RFBの一時退避用レジスタとして使用する。

表4.2 FGSの使用レジスタ

用途	個数
マイクロコードコンパイラ	20
マイクロルーチンおよびマイクロルーチンインタフェース	10
共通任意利用	9

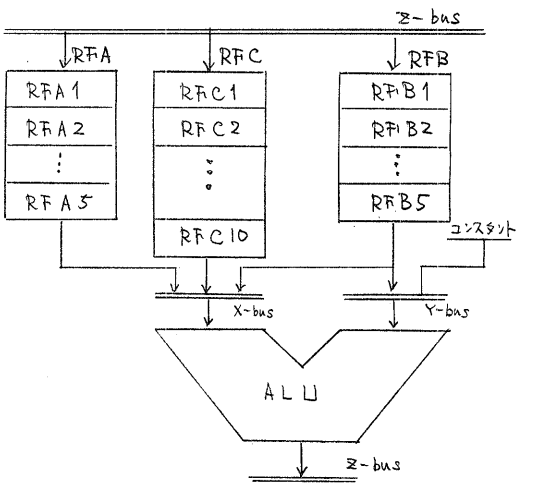


図 4.5 ハードウェア構成図

(2) ハードウェア上の使用制約条件

条件として以下の項目がある。

(a) 定数はALUのX入力として使用できない。
(b) レジスタファイルRFAは，ALUのY入力として使用できない。
(c) 同一レジスタファイル間の演算は許されない。
(d) 演算結果は，特殊レジスタを除き，X入力，Y入力で指定した以外のレジスタにセットすることはできない。

(3) レジスタ割付け方針

レジスタ割付けの最適化は，前項のレジスタ制約条件を十分に反映させることである。しかしながら，一度に全領域の全識別子に対して，レジスタ制約条件を考慮に入れながら最適な割付けを行なうことは難しいため，FAISではつぎのような方針に基づいて段階的に実施した。

(a) 対象範囲の設定：識別子とレジスタの組合せの妥当性を調べる範囲として4.4で設定したセクションを用いる。

(b) レジスタ数の理想化：仮割付ではレジスタ数は無限にあるものと想定して，レジスタ制約条件に従って，識別子が要求するレジスタをレジスタタイプの形式で割当てる。この仮割付けをレジスタタイプ割付けと呼ぶ。

(c) レジスタ数の有限化：本割付では，対象識別子に割付けられたレジスタタイプを基礎として，レジスタ割付け状況を考慮に入れて，実際のレジスタ番号を定めていく。この本割付けをレジスタ名割付けと呼ぶ。

(4) レジスタタイプ割付け

レジスタタイプ割付けはセクション内の中間言語の対象識別子にレジスタタイプ（RFAまたはRFB）を割付けるものである。レジスタタイプ割付け規則：

(i) 初期条件はセクションの境界条件とする。

(ii) アサイン命令 idi op idj → idk 形において，op＝'−'のとき，idiはRFA，idjはRFB，idkは不定タイプとする。

(iii) idi op idj → idk 形において op＝'+'のとき，idiがRFA(B)ならばidjはRFB(A)，idkは不定タイプとする。

(iv) 定数 → idi 形で idiがアドレス変数のとき，idiはRFBタイプとする。

(v) 同一識別子に対しては，同一レジスタを割付ける。（伝播規則）

レジスタタイプの割付け例は，図4.6に示す。同図の(a)は，1回目のタイプ割付け（該当規則は(ii)のみ）を行なった結果である。同図の(b)は割付規則(iii)，(v)を繰返し適用した最終状態を示したものである。同図の(c)は得られたセクションの境界条件を示す。

つぎにセクションSiに接続するセクションSjがあれば，Siの最終状態をSjの初期状態として，同様の操作を最終セクションまで続ける。なお，レジスタタイプ割付けが終了した際の各識別子の状態は，RFAタイプ，RFBタイプ，または不定タイプのいずれかとなる。

	a	b	c	d	e	f	t_1
(1) '1' → a	1	◇					
(2) '2' → b	2		◇				
(3) '3' → c	3			◇			
(4) a＊d → t_1	4	◇			◇		◇
(5) t_1＋b → t_1	5		◇				◇
(6) t_1 − c → e	6			Ⓑ		◇	Ⓐ
(7) a＋c → f	7	◇		◇		◇	

(a) 識別子－レジスタ表（初回）

	a	b	c	d	e	f	t_1
(1) '1' → a	1	Ⓐ*					
(2) '2' → b	2		Ⓑ*				
(3) '3' → c	3			Ⓑ*			
(4) a＊d → t_1	4	Ⓐ*			◇		Ⓐ*
(5) t_1＋b → t_1	5		Ⓑ*				Ⓐ*
(6) t_1 − c → e	6			Ⓑ		◇	Ⓐ
(7) a＋c → f	7	Ⓐ*		Ⓑ*		◇	

(b) 識別子－レジスタ表（最終）

	a	b	c	d	e	f	t_1
初期状態	◇						
最終状態	Ⓐ	Ⓑ	Ⓑ	◇	◇	◇	Ⓐ

(c) セクションの境界条件表

図 4.6 レジスタタイプ割付例

(5) レジスタ名割付け

レジスタ名割付けは，レジスタタイプ割付けで定まったレジスタタイプを基礎として，実際のレジスタ個数を考慮しながらセクション毎にレジスタ番号を確定してゆく。 レジスタRFA(B)がフル状態の場合には，レジスタ退避アルゴリズムが動作して，最も不要な識別子を選び同識別子が占有していたレジスタの内容を退避用レジスタRFCやメモリに返す操作をレジスタ管理表に登録する。

レジスタ管理表は中間言語の各命令ステップに対応した全レジスタ状態を定めるものであり，その構成は図4.7に示す。

したがって，レジスタ名割付けはレジスタ管理表を作成することであり，つぎの3段階の処理を経過する。

(i) 前ステップのレジスタ状況を現ステップに継続させるコピー処理.

(ii) 識別子ライフサイクル表より寿命の終った識別子が占有していたレジスタの解放処理.

(iii) 対象識別子のレジスタ名割付け処理.

同アルゴリズムを図4.8に示す.

4.7 コードジェネレーション

マイクロジェネレータは1個の中間言語を核として，図4.9に示すような一連のマイクロ命令を作成する。 図中の破線部はレジスタの割付け状況に伴って生成されるマイクロ命令である。 本図の例では識別子bが Reg_j を使用するため，以前同レジスタを使用していた識別子fの値は必要に応じて，メモリまたは退避用レジスタにストアされるか，もしくは無視される。 値がストアされる際には，ストア用マイクロ命令を作成する。

ロード用マイクロ命令も同様に作成する。

図中の実線部はすでに値がレジスタ上に存在する時に，中間言語で指示される操作を行なうマイクロ命令の系列である。

一連のマイクロ命令の作成は，下位Sが内蔵するマイクロスケルトンと呼ばれる半完成のマイクロ命令の系列を利用して行なう。 マイクロ命令の未指定部は，

		Rfa1 Rfa2 …	Rfb1 Rfb2 Rfb3 …	Rfc1
'1' → a	1	a	φ	
'2' → b	2	a	b φ	
'3' → c	3	a φ	b c φ	
a+d → t_1	4	a t_1	b c d	
t_1 + b → t_1	5	a t_1	b c φ	
t_1 − c → e	6	a t_1	e c φ	
a+c → f	7	a φ	f c φ	

図 4.7 レジスタ管理表

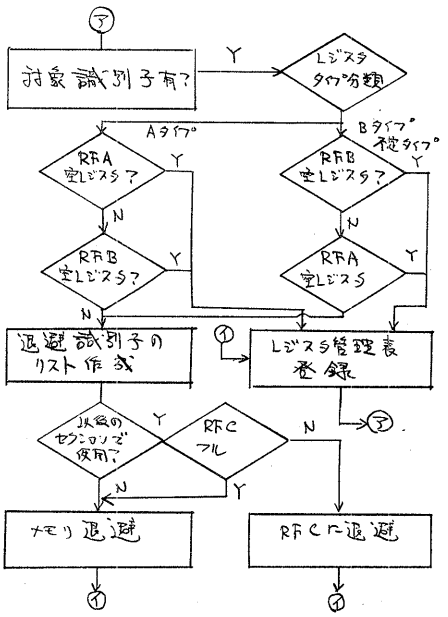


図 4.8 レジスタ名割付けブロック図

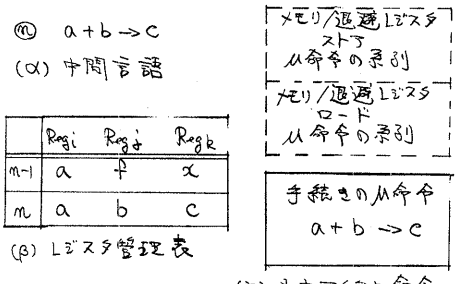


図 4.9 中間言語とマイクロ命令との対応

レジスタ部やサブファンクション部，およびシーケンスコントロール部などであり，レジスタ管理表や中間言語から不足情報を補う。マイクロステルトンは，中間言語の命令別，オペレーション別，属性別にそれぞれ用意されている。

4.8 オプティマイゼーション

4.7節までの処理でマイクロ命令が生成された。本節では最適化のために，以下の処理を行なう。最適化の前処理として，一連のマイクロ命令の系列をブロックに分割する。ブロックは分枝マイクロ命令を含まず，直線的に実行可能なマイクロ命令の集団とする。ブロックの利用目的は，並列実行可能なマイクロ操作を，他のマイクロ命令に埋め込む際の探査範囲を規定することである。

最適化の処理はつぎの三タイプを取扱う。

タイプ1：サブファンクションのみのマイクロ命令を他のマイクロ命令と合体化する。

タイプ2：冗長なマイクロ命令の削除

タイプ3：冗長なマイクロジャンプ命令の削除

```
         BEGIN    MAIN                                                    00000000
ZZRIOT   DS       130                                                     00010000
ZZRIIT   DS       62                                                      00020000
         EJECT                                                           00030000
MAIN     WA                                                              00040000
L0       WA       V1          <- 0                                       00050000
         WA       V2          <- 0                                       00060000
         WA       V3          <-          X'0001'                        00070000
         WA       W2          <- 0                                       00080000
         WA       ;1110->SAA                                             00090000
         WA       V(1,SAA4)   +           W2          ->Y                00100000
         WA       W3          <-          X'0001'                        00110000
         WA       V(1,SAA4)   +           W3          ->Y        △       00120000
         WA       W4          <- 0                                       00130000
         WA       V(1,SAA4)   +           W4          ->Y                00140000
         WA       W5          <- 0                                       00150000
         WA       V(1,SAA4)   +           W5          ->Y                00160000
         WA       W6          <-          X'0001'                        00170000
         WA       V(1,SAA4)   +           W6          ->Y                00180000
         WA       V4          <-          X'0001'                        00190000
         WA       V(1,SAA4)   ->          MD                             00200000
         WA       V4          +           MD          ->X                00210000
ZZL005   MEM      MM(W2) ->MD     ;                                       100220000
         W2       +           0           ->X     ;                      200230000
                  MEM.SYNC.                                              00240000
         WA       V5          <-          MD                             00250000
         DO       FHD3        <-          X'3'                           00260000
         WA       V2          ->          MD                             00270000
         WA       V(1,SAA4)   <-          MD                             00280000
         MEM      MM(W3) ->MD     ;                                       100290000
         W3       +           0           ->X     ;                      200300000
                  MEM.SYNC.                                              00310000
         WA       V2          <-          MD                             00320000
         WA       V2          ->          TR                             00330000
         WA       V5          -(*1)       TR          ->Y                00340000
         WA       ;TR15->T0                   ;                          100350000
                  GOTO ZZL015  ,L1        ,(,,T0)                        00360000
ZZL015   WA       V1          <-          X'0001'                        00370000
         MEM      MM(W4) ->MD     ;                                       100380000
         W4       +           0           ->X     ;                      200390000
                  MEM.SYNC.                                              00400000
         WA       V2          <-          MD                             00410000
         WA       V2          ->          MD                             00420000
         WA       V5          <-          MD                             00430000
         MEM      MM(W6) ->MD     ;                                       100440000
         W6       +           0           ->X     ;                      200450000
                  MEM.SYNC.                                              00460000
         WA       V2          <-          MD                             00470000
         WA       V2          ->          MD                             00480000
         MEM      MM(W5) <-MD     ;                                       100490000
         W5       +           0           ->X     ;                      200500000
                  MEM.SYNC.                                              00510000
         WA       V5          ->          MD                             00520000
         WA       V4          ->          TR                             00530000
```

図4.10　出力マイクロプログラム例（1部）

5. 性能評価例

FGSによって生成されたマイクロプログラムの実行速度を評価するために，行列処理を含む簡単な数値計算プログラムに対して，FORTRANの出力アセンブラ，人手で書かれたアセンブラ等との速度比較を行なった。表5.1はこの結果を示している。FGSによるマイクロプログラムとFORTRANの出力アセンブラの速度比較では，前者の方が，2.5倍～5倍強高速である。人手によるマイクロプログラムとの比較では，FGSの方が2倍遅く，また語数では約4割冗長度が高いという結果が得られた。またコンパイル時間は，現在プログラムが2部に分割されているために遅く，表5.2の結果となった。但し，コンパイル時間には最終のマイクロソースプログラムがMTに出力されるまでの時間を含む。

プログラム種類（データ数）	FGSによるマイクロプログラム	FORTRANコンパイラのアセンブラ	人手で書かれたアセンブラ
2次元配列の要素の総和（200）	1630 μsec	9550 μsec	
1次元配列の要素の総和（200）	1300 μsec <語数43> *	3500 μsec	2000 μsec
配列ならし面線形プログラム（5）	15.9 μsec	43.2 μsec	30 μsec

* 手書きマイクロプログラム 610 μsec <語数30>

表5.1 速度性能比較表

ソースプログラム長	中間言語	セクション	μプログラム長	処理時間I**	処理時間II**
8	15	3	155	2m 38sec	4m 5sec
6	10	3	91	1m 46sec	2m 26sec

表5.2 コンパイル時間 ** CPU時間のみ

6. むすび

問題向き高級言語で記述されたプログラムをマイクロプログラムに変換するシステムにつき，その基本構成，中間言語，Lジスタ割付け，コード生成，最適化の機能，および性能評価例に関して述べた。FGSで作成されたマイクロプログラムとFORTRANとの比較では，前者でアクセスできるLジスタ数が多く，メモリーLジスタ間の転送を少なくすることができた結果が表われていると考えられる。また評価で用いたプログラムが比較的小さいものであるため，人手で書かれたマイクロプログラムに対する速度比では2倍遅く，冗長度1.4倍程度のデータが得られているが，更にサイズの大きいプログラムに対しては，冗長度，速度比とも更に劣化する可能性がある。これはグローバルなLジスタ割付けの最適化が不十分であるためであるが，他方，1000ステップを超えるファームウェアでは，人手による作成でも最適化されたものを得ることが難しく，高級言語の方がかえって質のよいマイクロプログラムを作成するとの見解もあり，今後，更に最適化と評価実験をくりかえして行きたい。

謝辞　本研究にあたり，御協力いただいた三菱総合研究所の堀田正利氏に，またマイクロアセンブラの提供，および各種の助言をいただいた当社．計算機製作所 計算機製造部の諸氏に深く感謝いたします。

参考文献

1. P.W. Mallett, T.G. Lewis, "Considerations for Implementing a High level Microprogramming Language Translation System", Computer pp 40-52 Aug (1975)
2. C.J. Tam, "Code optimization technique for micro-code compilers," Proc. of NCC '78, vol. 47, pp 649-655, (1978)
3. R.L. Kleir, C.V. Ramamoorithy, "Optimization Strategies for Microprograms," IEEE Trans. Comput, Vol C-20, No.7, July, (1971)
4. "マイクロプログラミング特集"，情報処理，Vol-14, No.6.(1973)
5. 萩原，マイクロプログラミング，産業図書，1977
6. 三上，房岡，"ファームウェアジェネレータシステムの処理方式，" 情報処理18回全国大会，369
7. 三上，房岡，"ファームウェアジェネレータシステムにおけるレジスタアサインメントの一方式，" 昭52電気関係学会，関西支部大会 G6-11
8. 三上，房岡，"ファームウェアジェネレータシステムの性能評価，" 情報処理19回全国大会 6C-9