論　文

MPG マイクロプログラム・コンパイラ*

馬場敬信** 藤本裕司*** 萩原　宏****

Abstract

A powerful microprogram compiler has been constructed as a subsystem of Microprogram Generator System (MPG). It translates a sequentially-written source microprogram into efficient object microinstructions in a machine-independent fashion.

The compiler separates the translation process into four phases: (1) syntax analysis (2) pre-processing for optimization (3) micro-order generation and microinstruction composition, and (4) addressing and bit encoding.

In implementing the compiler, several important problems of constructing machine-independent microprogram compilers became clear and new techniques were developed for them. The techniques are determination and allocation of available registers, generation of micro-orders from machine description, detection of parallelly executable microoperations, composition of microinstructions, and automatic addressing of microinstructions in a compact control memory area.

In this paper, after an overview of the compilation procedure of each phase, the machine-independent procedures for generating efficient object codes are described in detail. Experimental results show that the object codes generated by this compiler are as efficient both on speed and on space as hand-coded microprograms.

1. はじめに

ダイナミック・マイクロプログラミングの発展に伴い，マイクロプログラム（以下，μP）作成の効率化を目的として μPの記述システムがいくつか提案され，作成されている[1)~6)]．しかし，これらのシステムは，実用的なシステムとして幅広い計算機に使用するには，まだ多くの問題を含んでいる．特に，μP を記述対象とする場合には，機械語をオブジェクト・コードとするコンパイラに比べ，オブジェクト・コードの効率の良さが，より一層求められる．これは，

i）ダイナミック・マイクロプログラミングが，本来効率良く計算機を制御して，能率を上げることを目的として導入されたものである．

ii）μP は，一度作成されれば，繰り返し使用される事が多い．

iii）書き換え可能制御記憶として使用される IC メモリのコストが高い．

などの理由による．一方，プログラマの立場からは，記述言語の高レベル化が望まれ，このため，従来のコンパイラの技法を応用し，あるいは，μP に記述された変数間の相互関係に注目した最適化の手法が考えられている[8)~10)]．しかし，オブジェクト・コードの構成要素であるマイクロ命令*(脚注次ページ参照)が，1つで同時にいくつかのハードウェア資源を制御するため，最適化に当っては，マイクロ操作実行のタイミング，マイクロ操作間の関係，間接符号化方式，あるいは，データ・パスの並列性などについて考慮する必要があり，これらの諸点について，従来の方法は，幅広い計

* Microprogram compiler of MPG by Takanobu BABA (University of Electro-communications), Yuji FUJIMOTO (Fujitsu Co., Ltd.) and Hiroshi HAGIWARA (Faculty of Engineering, Kyoto University)
** 電気通信大学電気通信学部
*** 富士通(株)
**** 京都大学工学部

算機に適用するには不十分である.

また, 効率の良いオブジェクト・コードを得るために, 従来, 全く触れられていない問題として, μP コンパイラにおけるマイクロ命令の制御記憶への割付けの問題がある. 特に, μP 制御方式計算機では効率向上のために次マイクロ命令アドレスの生成法の種類が多く, また1つのマイクロ命令で演算の制御と順序制御を行うという特徴がある. このため従来のコンパイラの技法は応用できず, 新たに計算機に独立な割付けの手法を考えなければならない.

筆者等は, 以上のような問題点を考慮して, 先に提案した μP 記述言語: MPGL[12]のコンパイラーMPG マイクロプログラム・コンパイラーを作成した[17]. 以下, まずシステムの概要について述べると共に, 本システムにおけるマイクロ命令の構成と制御記憶への割付けの手法について述べる. 最後に, 作成した処理システムを用いてコンパイルを行った結果から, これらの手法が水平型と垂直型の異なるタイプの計算機に適用できること, そして, オブジェクト・μP が手書きのものにほぼ匹敵する効率を有し, これがコンパクトな制御記憶領域に割付けられることを示す.

2. システム概要

Fig. 1 に示すように, MPGマイクロプログラム・コンパイラは, MPGL 言語によって記述された, 計算機の記述部 (MDS, Machine Description Section), 及び, μP の記述部 (ADS, Algorithm Description Section) を入力とし, 最終的な出力として, ビット・パターン形式の μP を出力する. 以下, 処理手順とマシン定義テーブルについて簡単に述べる.

2.1 処理の手順

処理は4つのフェーズで行われる. まず, フェーズIでは, 外部記憶 (MDTL) より主記憶にマシン記述を読み込む. 次に, ADS を読み込んで, マシン記述との対応をチェックしながら構文解析を行い, 中間言語である四項系列とシンボル・テーブルを出力する.

フェーズIIでは, 四項系列を解析し, これをシーケンシャルに実行される四項系列 (セグメントと呼ぶ)

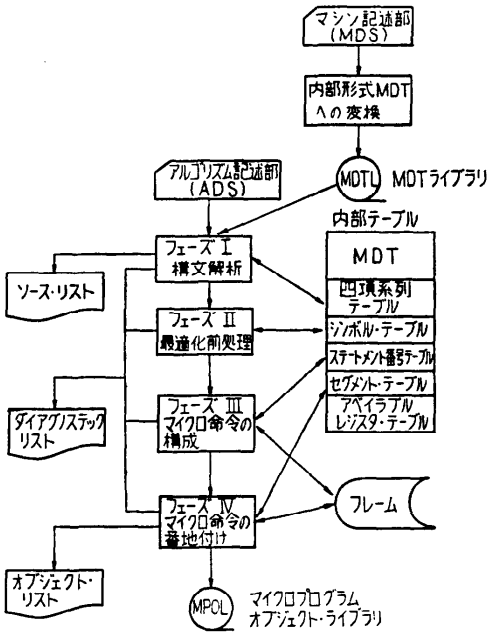

Fig. 1 System configuration of the MPG microprogram compiler.

に区切り, 更に各セグメント内で, コンパイラが一時記憶として利用可能なレジスタ, スクラッチパッド・メモリ (アベイラブル・レジスタと呼ぶ) を見出す. これは『最後にその値が参照されてから, 次に更新されるまでのレジスタ』のことである.

フェーズIIIでは, 四項系列ごとに, その四項系列で記述された動作を制御するマイクロオーダを生成し, これからマイクロ命令の構成を行う. 処理は, 計算機に独立な最適化手法によって行われ, マイクロ命令はフレームと呼ぶ擬似マイクロ命令エリアに構成される.

フェーズIVでは, フレームに対して MDS に記述された次マイクロ命令アドレス生成法を参照して番地付けを行い, 各マイクロ命令に分岐のためのマイクロオーダを付け加える. 最後に, フレームをビット・パターン形式のオブジェクト・コードに変換し, 外部記憶 (MPOL) に掃き出す.

フェーズ I, II の処理は, 計算機の記述を参照する点を除けば, ほぼ従来の機械語レベルのコンパイラの処理と同様である. しかし, フェーズ III, IV の処理には, μP コンパイラに固有の効率の良いオブジェクト・コード生成のための手法が開発されており, これらに

* 以下, 本論文では制御記憶に記憶される一語をマイクロ命令と呼ぶ. マイクロ命令はいくつかに分割され, 分割された各部分 (これをフィールドと呼ぶ) ごとに, いくつかの制御信号が符号化されている. (即ち, 1つの制御信号にフィールドのビットパターンの1つが対応付けられる) この符号化された制御信号をマイクロオーダと呼び, これに付けられたネモニックをマイクロオーダ・ネモニック (あるいは単にネモニック) と呼ぶ. マイクロオーダは単独で, あるいは相互に関連しあってレジスタ間の転送などの基本的な動作を制御しており, この動作をマイクロ操作と呼ぶ.

ついては **3.** と **4.** で詳述する.

また，本コンパイラは HITAC 8350 のアセンブリ言語を使用して作成したが，その規模は総命令数が 34,000，メモリ所要量が 240 k バイト（オーバレイにより最大長 130 k バイト）であった*.

2.2 マシン定義テーブル

MDS は内部テーブル MDT (Machine Definition Table) に変換された後，いったん外部記憶に掃き出され，これをコンパイラが読み込む．MDT を構成する各部分テーブルの略称と，その内容を **Table 1** に示す．表の中で，FDT から MT が制御部を定義するのに対し，# で示したテーブルは，マイクロ操作とこれを制御するマイクロオーダ・ネモニックの対によって被制御部のマイクロ操作を定義する**.

Table 1 Contents of Machine Definition Table.

MDS 記述でのブロック名	内部テーブル名	内部テーブルに定義される内容
FDT	FDT	マイクロ命令の各フィールドの名称とそのビット長.
TIM	TIM	マイクロ操作の実行されるタイミング.
PST	PST	マイクロ命令のパリティ・ビット.
MT	MT	マイクロオーダ・ネモニックとビット・パターンの対応.
AGS	AGS#	制御記憶アドレス・レジスタへのアドレス生成法.
VT	VT	レジスタ等の各変数について，その属性と幅を登録し，同時に，次の TCT から SPAST への見出しとなる.
	TCT#	テスト条件.
	TRT#	リソース間の転送.
	FFST#	フリップ・フロップのセット条件.
	CCT#	カウンタの値の増減.
	MST#	メモリの読み書き．メモリ・アドレス・レジスタ，メモリ・データ・レジスタ，メモリの容量.
	SPAST#	スクラッチパッド・メモリのアドレスのセット法.
CT	CTH#	ワイヤッド・ロジックによる定数の発生法.
	CTF#	マイクロ命令のフィールドを利用した定数の発生法.
OPT	OPT#	算術論理演算装置による演算.
	EOT	OPT に記述された演算を構成する基本的な演算の機能.

は，マイクロ操作の定義を行う内部テーブルを示す.

* 内訳は，フェーズ I，II が 4,800 命令・27 k バイト，フェーズIII が 14,300 命令・73 k バイト，フェーズIV が 9,700 命令・51 k バイトで，その他各フェーズに対する主プログラムおよび各フェーズ間の共用プログラムなどである.

** 1 つのマイクロ操作を制御する，一般に複数のマイクロオーダ・ネモニック間の関係は『制御記述』として記述される (**3.3** で後述)．従って，**1.** の脚注で述べたマイクロ操作は，MPG においては，制御記述と対になった Table 1 に示す動作としてより具体的に定義される.

3. マイクロ命令の構成

1. で指摘したように，μP コンパイラにおいては，効率のよいオブジェクト・コードを得ることが，特に要求される．このため，筆者等は，既存の技法も含め，最適なマイクロ命令の構成法について検討を行うと共に，MPG コンパイラにおける手法を開発した.

3.1 マイクロプログラムの最適化法

μP の最適化法を分類すると，まず制御記憶が固定記憶の場合には，書き込まれるマイクロオーダ全体が，書き込む前から限られているため，これを利用して最小の制御記憶（語長と語数について）を構成することが考えられている[11]．書き換え可能制御記憶の場合には，マイクロ命令の形式は固定して，最少のマイクロ命令数，実行時間とすることが考えられ，対象とするマイクロ命令の形式が機械語とほぼ同様の垂直型の場合には，従来のコンパイラの技法を，そのまま応用することが試みられている[8]．しかし，水平型の場合には，1 つのマイクロ命令が同時にいくつかのハードウェア資源の制御を行うため，マイクロ操作の同時実行の可能性等について考慮する必要がある．このため，変数間の相互の依存性などに注目して最適化を行う方法が考えられているが，これらの手法は，特定の計算機に依存しない μP コンパイラに使用するには，次に述べるように多くの問題点を残している[9),10)].

i) 高級言語で記述された μP からマイクロ操作を決定し，さらにそのマイクロ操作を制御するマイクロオーダを決定するための，計算機に独立な手法が示されていない.

ii) マイクロ操作レベルでの並列性だけが問題とされ，間接符号化方式等のマイクロオーダレベルでの並列性が考慮されていない.

iii) マイクロ操作相互の関連を無視して最適化を行うのは，実際的でない.

iv) 最適化は，マイクロオーダの生成とマイクロ命令の構成の 2 つを組合せて考える必要がある.

v) 幅広い計算機のアーキテクチャを考慮する必要がある．特に間接機能制御方式等の μP 制御方式計算機で工夫されている手法を処理できなければならない.

これらの問題点を踏え，筆者等は『記述された μP からのマイクロオーダの生成』と，『マイクロ操作レベルとマイクロオーダ・レベルの双方の並列化の可能性を考慮したマイクロ命令の構成』とを組み合わせる

ことにより，効率の良いオブジェクト・コードを生成する手法を考え実現した．これらは，また計算機に独立な手法として間接符号化方式，間接機能制御方式などを処理可能としている．

以下，MPG マイクロプログラム・コンパイラに実現したマイクロオーダの生成法とマイクロ命令の構成法について具体的に述べる．

3.2 マイクロオーダの生成

まず，四項系列に対して，計算機に独立な生成規則に従ってMDTを参照することにより，その四項系列で指定された動作を行うのに必要なマイクロ操作を決定する．MDT では，マイクロ操作とそれを制御する制御記述が対になっているから(**2.2** 参照)，この制御記述を取り出し，MNとする．同時に，マイクロ操作実行により値を参照されるファシリティ*と参照のタイミングの対 (UV と呼ぶ)，及び，値を更新されるファシリティと更新のタイミングの対 (DV と呼ぶ) を決定する．三項系列 (MN, UV, DV) を決定することを**ジェネレーション**と呼び，**3.3** で述べるフレーミングの入力となる．

次に，各四項系列に対する生成規則の概要を述べる．(文中の略称に対する説明は Table 1 に示してある．)

(i) カウンタ制御 $(\gtrless, k, C,)$

カウンタCの内容をkだけ増加，あるいは，減少するための制御記述を CCT より見出す．Cが UV であると同時に DV でもある．

(ii) 転送制御 $(:=, \mathrm{VAR}\,1, \mathrm{VAR}\,2, n)$

VAR 2 の内容を読み出し，n 個の VAR 1 へ転送するための制御記述を見出す．VAR 2, VAR 1 がメモリ変数の場合には，メモリの読み出し，書き込みの制御記述を MST から生成する．VAR 2 が定数の場合，CTH あるいは CTF により定数を発生する制御記述を決定する．

(iii) 演算制御

$$\begin{cases} \text{単項}: (\mathrm{UOP}, \mathrm{OPND}\,1, , \mathrm{RES}\,n) \\ \text{2項}: (\mathrm{BOP}, \mathrm{OPND}\,1, \mathrm{OPND}\,2, \mathrm{RES}\,n) \end{cases}$$

2項演算について述べる．まず OPT で演算子 BOP を検索し，算術論理演算装置 (ALU) の入力端子と演算を制御する制御記述を見出す．次に，OPND 1, OPND 2 を読み出して ALU へ転送する制御記述を(ii)同様に決定し，これに演算制御の制御記述を付け加える．

(iv) フリップ・フロップ制御 $(\leftarrow, \mathrm{FF}, \mathrm{FFS},)$

FFS に記述された値をフリップ・フロップ FF にセットするための制御記述を，FFST より見出す．

(v) 分岐制御

テスト分岐 (BCT, C, , SBTP) について述べる．

まず，テスト条件Cに適合するものを TCT より検索し，対応する制御記述を生成する．次に，分岐のタイプ (BCT) と分岐先を表すシンボルテーブルへのポインタ (SBTP) を，現在構成されているマイクロ命令の最後のものに書き込む．

他の分岐制御の四項系列[14]に対しても同様の処理が行われる．

以上のようなジェネレーションの過程で，

(a) リソース間のデータ・パスを決定する．

(b) 可能なすべてのジェネレーションの中から，実際にマイクロ命令を構成することにより最適性を評価し，選択を行う．

(c) アベイラブル・レジスタを一時記憶として割付ける．

などを，コンパイラが行っている．

3.3 マイクロ命令の構成

3.1 に述べたように，ジェネレーションが行われるたびに毎回マイクロ命令の構成を行うというヒューリスティックな手法を採用した．このための手続きを**フレーミング****と呼び，計算機に独立な最適化のアルゴリズムに従って，フレーム**と呼ぶ **Fig. 2** (次頁参照) に示すような構造のテーブルにマイクロ命令を構成する．

フレーミングのアルゴリズムを **Fig. 3** (次頁参照) に従って述べる．まず，フレーム・ポインタ (FP) は，空でない***フレーム行に対してその最下行の FN (Fig. 2 参照) を保持する．この FP の指すフレーム行から始めて，UV, DV に関する制限条件を満たすような最も若い FN のフレーム行を決定する．このための条件を両立条件と呼ぶ．

〔両立条件〕 $\mathrm{UV}_1, \mathrm{DV}_1$ を既にフレーミングされた UV, DV とし，$\mathrm{UV}_2, \mathrm{DV}_2$ を新たにフレーミングしようとする UV, DV とする．このとき

i) 両立条件 A : UV_2 と同じ変数が DV_1 になく，かつ DV_2 と同じ変数が UV_1 にも DV_1 にもない．

---

* 本論文では，レジスタ，メイン・メモリ等の記憶要素をファシリティと呼び，ターミナル等の非記憶要素と合わせたものをリソースと呼ぶ．

** Table 1 の FDT で定義されたフィールドごとの枠組みにマイクロ命令を構成してゆくことから，この枠組であるテーブルをフレーム (frame)，構成の手続きをフレーミング (framing) と名付けた．

*** 即ち，いずれかの BOX に1つ以上のマイクロオーダ・ネモニックが存在する．

4	20		20	54	54	4	25	4	4	4	4	15
FN	BOX 1	・・・	BOX 32	UV	DV	SN	QN	BTYPE	BTABP	AA	AGSTP	その他

計 812 バイト

FN ——フレームの各行を1から順に番号付ける.
BOX i ——1つのフィールドに対応し，消去用フラグあるいはネモニックの否定などを表すフラッグの付けられたマイクロオーダが最大5 (i=1～32) 個挿入できる.
UV, DV ——UV, DVに消去用フラグ，あるいは間接的に指定される変数（たとえばメモリ変数）であることを示すフラグを付けたもので最大6個まで挿入できる.
SN ——セグメントの始まりのフレーム行のみに，そのセグメントの番号が書き込まれる.
QN ——挿入された三項系列（MN, UV, DV）を生成した四項系列の番号を最大5個挿入できる.
BTYPE, BTABP ——分岐制御四項系列に対して分岐の型を示す第1要素（B, BCT, …）とシンボル・テーブルへのポインタである第4要素を書き込む.
AA, AGSTP ——フェーズIVで使用されるワークエリアで，このフレーム行に割付けられた番地を AA に，このフレーム行からの分岐法を同定する AGS へのポインタを AGSTP に書き込む. (4. で詳述)

Fig. 2 Structure of a frame row.

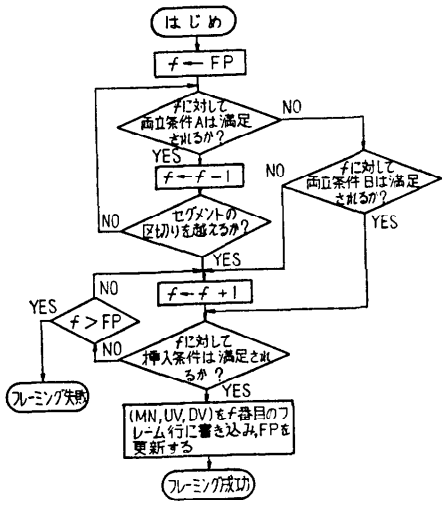

Fig. 3 Framing procedure.

ii) 両立条件 B: UV_2 または DV_2 の参照あるいは更新されるタイミングが，同じ UV_1 または DV_1 のタイミングよりも遅い.

条件Aは，ADS 上でのステートメントの実行順序を変更できる条件であり，条件Bは，実行順序が変更できない場合に，タイミングまで考慮すると同一マイクロ命令内で実行可能となる条件である．即ち，両立条件はソース・ステートメント・レベルからマイクロ操作レベルまでを考慮した並列化の条件となっている.

次に，MNについてマイクロオーダレベルでのマイクロ命令の構成可能性を調べる．このための条件が挿入条件で，与えられた FN のフレーム行を基準として，MNの要素である制御記述に記述された条件がすべて満足されるとき，挿入条件は満足されるとする.

以下，まず制御記述の仕様をバッカス記法で示し，更に各項の挿入条件に対する意味を具体的にのべる.

```
⟨CONTROL DESCRIPTION⟩ :: =
    /* ⟨CONTROL CONTENT⟩ */
⟨CONTROL CONTENT⟩ :: =
    # | ⟨POINTED MNEMONICS⟩
⟨POINTED MNEMONICS⟩ :: =
    ⟨POINTED MNEMONIC⟩ |
      ⟨POINTED MNEMONICS⟩ ,
        ⟨POINTED MNEMONIC⟩
⟨POINTED MNEMONIC⟩ :: =
    ' ⟨MNEMONIC EXPRESSION⟩ '
    (⟨MNEMONIC POINTER⟩) |
        ⟨MNEMONIC EXPRESSION⟩
⟨MNEMONIC POINTER⟩ :: =
    +1 | −1 | +2 | −2
⟨MNEMONIC EXPRESSION⟩ :: =
    ⟨MNEMONIC SECONDARY⟩ |
      ⟨MNEMONIC EXPRESSION⟩
        # ⟨MNEMONIC SECONDARY⟩
⟨MNEMONIC SECONDARY⟩ :: =
    ⟨MNEMONIC PRIMARY⟩ |
      ⟨MNEMONIC SECONDARY⟩
        @ ⟨MNEMONIC PRIMARY⟩
⟨MNEMONIC PRIMARY⟩ :: =
    ⟨MNEMONIC IDENTIFIER⟩ |
      ¬ ⟨MNEMONIC IDENTIFIER⟩ |
        《 ⟨FIELD IDENTIFIER⟩ 》 |
    ((MNEMONIC EXPRESSION)) |
        ¬ ((MNEMONIC EXPRESSION))
⟨MNEMONIC IDENTIFIER⟩ :: =
    ⟨IDENTIFIER⟩
```

⟨MNEMONIC IDENTIFIER⟩ は記述されたネモニック（例えばMとする）の属する BOX に他のネモニックが存在しないか，あるいは，存在したとしても

¬M* でなくかつ対応するビットパターンの冗長部まで考慮すると一致する** とき，BOX に挿入可能とする．同様に ¬〈MNEMONIC IDENTIFIER〉は記述されたネモニックがその属する BOX にないとき挿入可能***である．また《〈FIELD IDENTIFIER〉》は指定されたフィルドにそのフィールドの無操作****以外のマイクロ操作を指定するネモニックが存在することを条件とする．これらの条件はさらに論理和 (♯)，論理積(@)を用いて相互の関係が記述されると共に，論理式全体に 〈MNEMONIC POINTER〉 を付けることにより，異なるマイクロ命令間のマイクロオーダ・ネモニックの関係をも条件として記述することができる．

挿入条件が満足されれば，そのフレーム行に MN が書き込まれる．また，空フレーム行に挿入不可能のときフレーミングは失敗とする．

3.4 マイクロ命令の構成例

Fig. 4(a), (b) に HITAC 8350[15] に対する乗算の μP から四項系列 $(B)_{16}(C)_{16}(F)_{16}$ に対するジェネレーション例を示した．$(B)_{16}(C)_{16}(D)_{16}(E)_{16}$ に対する三項系列 (MN, UV, DV) は FN 11 と 12 のフレーム行に構成される．その後で $(F)_{16}$ に対する三項系列(MN, UV, DV) をフレーミングすると，まず両立条件については $(B)_{16}(C)_{16}(D)_{16}(E)_{16}$ の UV, DV と $(F)_{16}$ の UV, DV は同じものがなく，両立条件Aが成立する．しかし FN 11 がセグメントの先頭にあるためこのフレーム行に対して挿入条件が調べられ (Fig. 3 参照)，DG 1 の属する CL フィールドが空であるためこの行に挿入される．(Fig. 4(c)において，$(F)_{16}$ に対して挿入されたものを破線で囲んだ.)

MN		UV	DV
U	0	U'0:31' T5	
NAB	#		
SRL@<BB>	# 0		SC'0:0' T8
UI	# 0		U'0:31' TA

(a) Generated triplet (MN, UV, DV) for quadruplets $(B)_{16}$ and $(C)_{16}$.

MN		UV	DV
DG1	# 0	G'0:7' T9	G'0:7' T9

(b) Generated triplet (MN, UV, DV) for quadruplet $(F)_{16}$.

```
##############
## FN=0011 ##
##############
 ### MNEM. ###
     BB   U    00 #          AB   NAB 00 #          AA   #
     BA   UI   00 #          SP   #                OP   SRL 00 #
     EX   #                  CL   DG1 00 #          TS   #
     JA   #                  TP   #                MN   #
     I    #
 ### SN =  00000004   ### BTYPE =          ### BTABP = 00000000
 ### AA =  00000000   ### AGSTP = 00000000 ### TK =
 ### QN =0000000B   0000000C   0000000F
 ### UV ###
        U     001F T5 00    G'0:7' T9 00    &
 ### DV ###
        SC    0000 T8 00    U     001F TA 00    G'0:7' T9 00    &
```

(c) Computer output of a frame row.

Fig. 4 An example of generation and framing for the HITAC 8350 machine.

* Mの指定の禁止を表す．*** を参照．

** 即ち，存在するネモニックを Me とすると，Mと Me が同じネモニックであるか，あるいは対応するビット・パターンがたとえばMが 11XX で Me が X100 である場合 (X は冗長ビット)．

*** 挿入可能な場合には ¬ を表すフラグと共に挿入する (Fig. 2 BOX *i* 参照)．

**** マイクロ命令の各フィールドには，制御信号を出力しないネモニックが用意されている場合が多く，これを無操作 (no operation) と呼ぶ．

***** アドレッシングとシーケンシングを総称して「割付け」と呼ぶ．

4. 制御記憶への割付け

4.1 アドレッシング及びシーケンシングについて

μP 制御方式計算機では，オブジェクト・コードの効率向上のため，次マイクロ命令アドレスの生成法の種類が多い．また，1つのマイクロ命令で演算の制御と順序制御を同時に行う．このため，機械語をオブジェクト・コードとするコンパイラと比較して，制御記憶に番地付けを行い (アドレッシングと呼ぶ)，かつ順序制御のためのマイクロオーダを付け加える (シーケンシングと呼ぶ) のは，かなり複雑なアルゴリズムを要する*****．

普通のマイクロアセンブラの場合には，割付けは演算の制御と同様に，プログラマが記述する[15),16)]．また，特定の計算機専用の μP コンパイラでは，その計算機の次マイクロ命令アドレス生成法についての知識を利用して，割付けのアルゴリズムが考えられる．これに対して，特定の計算機に依存しない μP コンパイラでは，記述された次マイクロ命令アドレス生成法をもとに，計算機に独立なアルゴリズムで割付けを行わねばならない．現在のところ，このためのアルゴリズムは報告されておらず，筆者等は，全く独自の手法を考えこれを実現した．以下，このアルゴリズムと実際の処理について述べる．

4.2 割付けのアルゴリズム

割付けに対する入力情報は，(i) AGS に定義された次マイクロ命令アドレス生成法 (文献 12) 参照)，及び，(ii)フレームの各行に書き込まれた分岐の型 (BTYPE) と分岐先 (BTABP) (Fig. 2 参照)，である．

これらの入力情報を使用して割付けは行われるが，その処理のアルゴリズムは次に述べる方針のもとに考えた．

(i) コンパクトな領域に割付ける.

(ii) 割付け不能に陥らないようにする. 即ち, 次マイクロ命令アドレス生成法により, 分岐できる範囲が制限されるため, 割付け不能に陥ることがあり, これをできるだけ避ける工夫をする.

(iii) オブジェクト・コードの効率向上のため, 分岐のためだけのマイクロ命令 (ダミー・マイクロ命令と呼ぶ) は, できるだけ付け加えない.

次に, 各フレーム行を節点に対応させ, その間の分岐の関係を有向弧に対応させた有向グラフ表現を用いて, 割付けのアルゴリズムを述べる.

まず, 割付けを行っている任意の時点で, ある節点 a の親節点と子節点について, 既に番地が割付けられているものを $P(a), S(a)$ で, まだ割付けられていないものを $p(a), s(a)$ で表わす. このとき, 次に述べる条件(i), (ii) が『既に割付けられた節点間の分岐可能性を損わずに, 番地 A に a を割付けられる』ための条件になっていることは明らかである.

(i) A に節点が割付けられていない.

(ii) a を A に置いたとき, AGS に記述された適当な次マイクロ命令アドレス生成法により, 次の事が可能である.

① $P(a)$ の各節点 n より $S(n)$ への分岐を行うためのマイクロオーダのジェネレーションとフレーミング.

② a から $S(a)$ への分岐を行うためのマイクロオーダのジェネレーションとフレーミング.

Fig. 5 は, フェーズ IV における割付けの概略フローを示したもので, 上記の条件を満たすように, 1番目の節点から次々とヒューリスティックな方法で割付けを行っている. 図の右側に, 手続きに対応する条件を示した. また, 図の中に示したように, 割付けの可能性を増すために, リロケーションとダミー節点の作成を行う. リロケーションとは, ある節点の割付けのために, 既に割付けられた節点を, (i), (ii) の条件を満たす番地の中で再配置することである. 再配置は更に他の節点の再配置を生む可能性がある. また, 節点 a に対するダミー節点の作成とは, 新たに分岐のためだけの節点 d を付け加え, a から d へは無条件分岐とし, d からは, a の子節点への分岐を行わせるものである. これによって割付け可能性が増す原因は,

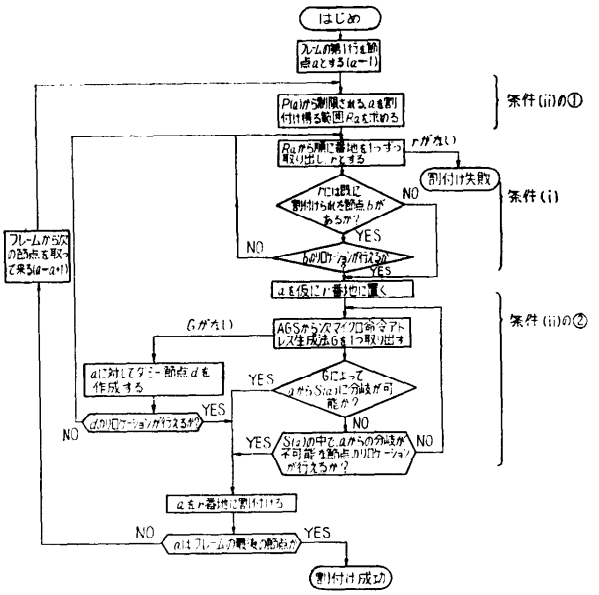

Fig. 5 Addressing procedure.

(1) フェーズ III で挿入されたマイクロオーダの影響によるフレーミング不能の解消, (2) a を置ける範囲は, $P(a)$ と $S(a)$ から制限されるが, 一方を無条件分岐とすることにより, 一般的にこの範囲が広げられる*, の2点である.

また, 処理は外から与えるパラメータによって, 親節点数と子節点数の多少による割付けの順序の制御, 及びリロケーションやダミー節点の作成の処理の順序の制御が行えるようになっており, 割付け可能性と処理効率の向上を計っている.

4.3 割付けの処理法

4.2 で述べたアルゴリズムを, 処理システム上で実現するために用いた手法について述べる.

4.3.1 分岐可能範囲の計算

分岐可能性を調べるために, ある節点 a, a の番地 ($A(a)$ と記す), a からの分岐の型及び a の子節点を入力とし, $S(a)$ を固定したままで, $s(a)$ を割付け得る範囲を求める必要がある (**Fig. 6**(a)(次頁参照)). このため, CMAR の各ビットに対して, AGS に記述されたアドレス生成法で生成される値を計算する. フィールドの値をセットする場合には任意の値がセットできるため, これが CMAR の右端に対するもののとき, ある幅を持った番地を生成し, そうでないとき, 飛び飛びの番地を生成する. これ以外の生成法では,

* 普通の計算機では, 無条件分岐による分岐可能範囲が最も広い.

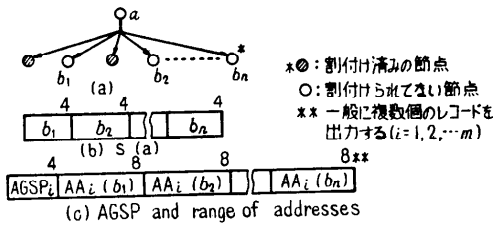

Fig. 6 Data formats of unaddressed successors and their areas to be addressed.

生成される値は固定されている．出力の形式を Fig. 6 (b), (c)に示す．図中，(b)に $s(a)$ を並べ，(c)には $s(a)$ の各節点を割付け得る範囲を示す．(c)の各レコードの先頭には，AGS のアドレス生成法を同定するための AGS へのポインタ (AGSP) を置く．

4.3.2 可動範囲の計算

$P(a)$ から制限される a を割付け得る範囲は，$P(a)$ の各節点に対して，**4.3.1** の計算を行い求める．この結果に更に a から $S(a)$ への分岐可能性の制限を付け加えることにより，$P(a)$ と $S(a)$ から制限された，a を割付け得る範囲が求められる(**Fig. 7**(a))．この出力の形式を Fig. 7(b)～(e) に示す．図中，(b)は $P(a)$ の各節点と割付けられた番地で，(c)の AGSP は (b) の各節点からその子節点への分岐法を同定する．(d)は，$s(i)(i \in P(a))$ と $s(a)$ を並べたものであり，(e)は，(d)の各節点に対応してその割付け可能範囲を表し，その先頭の AGSP は a から a の子節点への分岐法を同定する．(c)と(e)は，レコードごとに対応する．

4.3.3 フェーズ IV の処理

フェーズIVでは，次に述べる手順で割付けを行う．

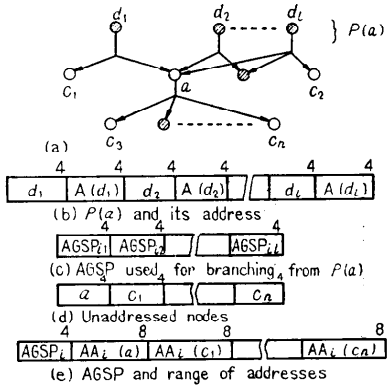

Fig. 7 Data formats for representing the limitation from addressed predecessors and successors.

(i) 親節点を求めるため，節点間のクロスリファレンステーブルを作成し，また，割付けの状態を書き込むための制御記憶イメージを作成する．

(ii) **4.3.1**, **4.3.2** で述べた処理を利用し，**4.2** で述べたアルゴリズムに従って割付けを行う．フレームの各行には，割付けられた番地が AA に，フレーム行からの分岐法を同定する AGS へのポインタが AGSTP に書き込まれる．また，AA と AGSTP に基づき，分岐のためのマイクロオーダがフレーミングされる．

(iii) MDT の FDT, MT, PST を参照してフレームをビット・パターン形式に変換する．

5. 処理結果と考察

MDS に記述した HITAC 8350 に対して ADS で記述した乗算の μP をコンパイルした時の各フェーズの出力を **Fig. 8** から **Fig. 12** (次頁参照) までに示す．フェーズIIIの出力であるフレームの第 11 番目の行は Fig. 4 に示した．

更に MPG と HITAC 8350 専用のマイクロアセンブラを使用して4つの μP (A, B, C, D) を作成し，オブジェクト・μP の効率等について比較した (**Table 2** (次頁参照))．また，水平型マイクロ命令をもつ HITAC 8350 に対して垂直型マイクロ命令をもつ HP 2100 A[16] を記述し，μP の作成を行った．これらの結果を以下に要約する[13]．

(i) 実行文の数と比較してフェーズIIIの出力マイクロ命令数は減少しており，これは最適なマイクロ命令構成のための手法の効果の概略*を示

```
ST-NO.    SOURCE   STATEMENT
          ADS MULT
     1     AVAIL SPM0(X"39") ;
     2     EXTRN ALG END (X"A20") ;
     3      ALG MULT (X"A50") ;
          ### SPM0(X"18") : ADDRESS OF MULTIPLIER
          ### SPM0(X"19") : ADDRESS OF MULTIPLICAND
     4           SPM0(X"10") := MM(SPM0(X"18")'8:29') ;
     5           SPM0(X"11") := MM(SPM0(X"19")'8:29') ;
     6           G := XL5"20" ;
     7           U := XL32"0" ;
     8           L := XL32"0" ;
     9      L0 : SPM0(X"10") := <SRLA> SPM0(X"10") ;
    10           IF SC'01' = BL1"0" THEN GOTO L1 ;
    11           U := U <+>  SPM0(X"11") ;
    12      L1 : U := <SRL>  U ;
    13           L := <SRC> L ;
    14           > G ;
    15           IF G<= XL8"0" THEN GOTO L0 ;
    16           MM(SPM0(X"18")'8:29') := U ;
    17           SPM0(X"18") := SPM0(X"18") <+> XL32"4" ;
    18           MM(SPM0(X"18")'8:29') := L ;
    19           GOTO END ;
    20        GLA;
          SDA;
```

Fig. 8 Source listing of a sample microprogram.

* 例えば，HITAC 8350 の主記憶の読み書きには2つのマイクロ命令を要するため，主記憶の読み書きを含む実行文に対しては，最適なマイクロ命令の構成が行われても出力マイクロ命令数は増加する．

```
ST-  QN
NO.  (Hex)          QUADRUPLET
 4    1 (:=,SPM0(X"10"),MM(SPM0(X"18")),1)
 5    2 (:=,SPM0(X"11"),MM(SPM0(X"19")),1)
 6    3 (:=,G'0:7',XL8"20",1)
 7    4 (:=,U'0:31',XL32"0",1)
 8    5 (:=,L'0:31',XL32"0",1)
 9    6 (<SRLA>,SPM0(X"10")'0:31', ,#R(01)E)
      7 (:=,SPM0(X"10")'0:31',#R(01)E,1)
10    8 (BCT,SC'0'=BL1"0", ,L1)
11    9 (<+>,U'0:31',SPM0(X"11"),#R(02)E)
      A (:=,U'0:31',#R(02)E,1)
12    B (<SRL>,U'0:31', ,#R(03)E)
      C (:=,U'0:31',#R(03)E,1)
13    D (<SRC>,L'0:31', ,#R(04)E)
      E (:=,L'0:31',#R(04)E,1)
14    F (>  ,1,G'0:7' , )
15   10 (BCT,G'0:7'¬=XL8"0", ,L0)
16   11 (:=,MM(SPM0(X"18")),U'0:31',1)
17   12 (<+>,SPM0(X"18")'0:31',XL32"4",#R(05)E)
     13 (:=,SPM0(X"18")'0:31',#R(05)E,1)
18   14 (:=,MM(SPM0(X"18")),L'0:31',1)
19   15 (B, , ,END)
```

Fig. 9 Quadruplet table (Phase I).

SYMBOL	TYPE	QN	FN	AA
END	A**X	0	0	A20
MULT	A***	1	0	A50
L0	L***	6	0	0
L1	L***	B	0	0

Fig. 10 Symbol table (Phase I).

```
DECLARED AVAILABLE REGISTER

          SPM0(X"39")'0:31'

AVAILABLE REGISTER
  SEGMENT     NAME AND          QUADRUPLET NO.
  NO.         BIT POSITION      (START)   (END)
    1                               1        5
          SPM0(X"10")'00:31'        1        1
          SPM0(X"11")'00:31'        1        2
          G          '00:07'        1        3
          U          '00:31'        1        4
          L          '00:31'        1        5
    2                               6        8
          SPM0(X"10")'00:31'        6        7
    3                               9        A
          U          '00:31'        9        A
    4                               B       10
          U          '00:31'        B        C
          L          '00:31'        D        E
    5                              11       15
          SPM0(X"18")'00:31'       12       13
```

Fig. 11 Available register and program segment listing (Phase II).

Table 2 Comparison of the object code efficiencies between MPG microprogram compiler and HITAC 8350 micro-assembler.

ADS		A	B	C	D
MコPンGパイラ	実行文の数	13	42	52	99
	オブジェクト・マイクロ命令数	10	42	43	66
マイクロアセンブラによるマイクロ命令数		10	40	39	55

している．即ち，最適化の結果1つのマイクロ命令でより多くのマイクロ操作が実行され，資源の有効利用が実現されている．

ii) 制御記憶への割付けは必要最小限の記憶容量で行われている．また，ダミーマイクロ命令も必要最小限のものが補われていることが判った．例えば Table 2 のAからDの μP はそれぞれ 10, 42, 48*, 66 の制御記憶容量に割付けられ，ダミーマイクロ命令数はすべて0である．

(iii) 最終的な出力であるフェーズIVの出力マイクロ命令数は，ほぼマイクロアセンブラによるマイクロ命令数に近い効率を有する．

(iv) 処理時間の大半はフェーズIII，IVで費される．例えば Fig. 8 から Fig. 12 に示した例では，フェーズI，IIで1秒，フェーズIIIが12秒，フェーズIVが13秒であった．

(v) 水平型 (HITAC 8350) と垂直型 (HP 2100 A) の異なるタイプの計算機に適用できる．

(vi) アベイラブル・レジスタはフェーズIIIで一時記憶として有効に使用された．例えば，HP 2100 A の μP において 30 の代入文のうち 14 の代入文で使用された例がある．

```
MICROPROGRAM GENERATOR                    OBJECT LISTING
          ** MACHINE:HITAC8350    ** ALGORITHM:MULT      ** DATE:03-25-76
ADDRESS  FRAME-NO. BB  AB  AA  BA  SP  DP  EX  CL  P1  TS  JA  TP  MM  I   P2
                   (6) (3) (3) (5) (5) (5) (5) (7) (1) (7) (6) (3) (2) (1) (1)
0A50     0001      1B  0   0   00  12  00  00  00  1   29  11  0   1   1   0
0A51     0002      28  0   0   14  10  00  00  00  0   29  12  0   0   0   0
0A52     0003      1B  0   0   00  13  00  00  00  0   29  13  0   1   1   1
0A53     0004      28  0   0   14  11  00  00  00  1   29  14  0   0   0   0
0A54     0005      39  0   0   0A  00  00  00  00  1   29  15  0   0   0   1
0A55     0006      00  0   0   01  00  00  00  00  0   29  16  0   0   0   1
0A56     0007      00  0   0   02  00  00  00  00  0   29  17  0   0   0   0
0A57     0008      00  0   4   14  10  11  00  00  1   29  18  0   0   0   0
0A58     0009      00  0   0   00  00  00  00  00  1   27  1A  1   0   0   1
0A59     000A      08  0   4   01  11  0A  00  00  0   29  1A  0   0   0   1
0A5A     000B      08  0   0   01  00  11  00  0F  1   29  1B  0   0   0   0
0A5B     000C      09  0   0   02  00  13  00  00  1   16  17  1   0   0   1
0A5C     000D      1B  4   4   14  12  0A  00  00  1   29  1D  0   2   1   0
0A5D     000E      08  0   0   00  00  00  00  00  0   29  1E  0   0   0   0
0A5E     000F      1B  0   0   00  12  00  00  00  1   29  1F  0   2   1   1
0A5F     0010      09  0   0   00  00  00  00  00  1   28  20  0   0   0   0
```

Fig. 12 Object code listing of a sample microprogram (Phase IV).

* C の μP がオブジェクト・マイクロ命令数(43)以上の制御記憶領域 (48 語) を必要とするのは，48 語の範囲に点在する番地へ EBCDIK コードの値による機能分岐を行うためである．分岐先番地は EBCDIK コードの値によって決定されるため，これを変更することはできない．

6. むすび

本論文では，計算機に独立な μP 記述言語 MPGL のコンパイラとして開発した MPG マイクロプログラム・コンパイラのシステム概要，最適なマイクロ命令の構成法，及び，制御記憶への割付け法について述べた．特に，特定の計算機に依存しない μP の最適化法は，機械語をオブジェクト・コードとするコンパイラに比して，本システムの大きな特徴である．

また，実際にコンパイラを使用して処理を行った結果，異なるタイプの計算機に適用可能であること，また，オブジェクト・コードの効率がほぼ手書きのものに匹敵し，コンパクトな制御記憶に割付けられることが確かめられた．これらの結果から，本システムは汎用 μP コンパイラの可能性を実証したものと考える．

謝辞 御討論戴いた萩原研究室の各位，並びに，御援助戴いた，電気通信大学 武井健三教授，有山正孝教授に感謝する．

参考文献

1) R. H. Eckhouse : A high level microprogramming language (MPL), Proc. SJCC, pp. 169～177 (1971).
2) E. W. Dubbs, R. L. Parsons, J. E. Petersen : A microprogram design system translator, Compcon 72, Sixth Annual IEEE Computer Society International Conference, pp. 95～97 (Sept. 1972).
3) C. V. Ramamoorthy, M. Tsuchiya : A high-level language for horizontal microprogramming, IEEE Trans. on Computer. Vol. C-23, No. 8, pp. 791～801 (1974).
4) B. R. S. Buckingham, W. C. Carter, W. R. Crawford, G. A. Nowell : The Control Automation System, 6th Annual Symposium on Switching Circuit Theory and Logical Design (1965).
5) D. J. Dewitt, M. S. Schlansker, D. E. Atkins : A microprogramming language for the B-1726, 6th Annual Workshop on Microprogramming, pp. 21～29 (1973).
6) R. K. Clark, Mirager : the "Best Yet" approach for horizontal microprogramming, Proc. of the ACM National Conference, pp. 554～571 (1972).
7) P. W. Mallet, T. G. Lewis : Considerations for Implementing a High Level Microprogramming Language Translation System, COMPUTER, Vol. 8, No. 8, pp. 40～52 (1975).
8) R. L. Kleir, C. V. Ramamoorthy : Optimization strategies for microprograms, IEEE Trans. on Computers, Vol. C-20, No. 7, pp. 783～794 (1971).
9) M. Tsuchiya, M. J. Gonzalez, Jr. : An Approach to optimization of horizontal microprograms, Seventh Annual Workshop on Microprogramming, pp. 85～90 (1974).
10) S. S. Yau, A. C. Schowe, M. Tsuchiya : On Storage Optimization of Horizontal Microprograms, 1974 Seventh Annual Workshop on Microprogramming, pp. 107～118 (1974).
11) C. Montangero : An Approach to the Optimal Specification of Read-Only Memories on Microprogrammed Digital Computers, IEEE Trans. on Computers, Vol. C-23, No. 4, pp. 375～389 (1974).
12) 馬場，萩原，藤本：マイクロプログラム記述言語：MPGL，情報処理，Vol. 18, No. 6, pp. 558～565 (1977).
13) 馬場，萩原：マイクロプログラムの自動作成について，情報処理，Vol. 19, No. 1, pp. 61～69 (1978).
14) 馬場，藤本，萩原：マイクロプログラム・ジェネレータの作成，情報処理学会計算機設計自動化研究会資料，DA 27-1 (1975).
15) 日立製作所：HITAC 8350 RCM 処理システムマニュアル (RCM 機構仕様書その他).
16) Hewlett Packard Company : Model 2100 A Computer Mannual (Microprogramming Guide, etc.).
17) T. Baba : A Microprogram Generating System-MPG, Proc. IFIP Congress 77, pp. 739～744 (1977).

(昭和 51 年 6 月 15 日受付)
(昭和 52 年 5 月 27 日再受付)