汎用マイクロプログラムトランスレータ MARTRAN*のコード生成方式†

平岡良成††** 坂東忠秋†† 福永泰††
平沢宏太郎†† 迫田行介††† 中西宏明††††
馬場敬信†††††

記述性の高いレジスタ転送言語をソースとしてビットパターン形式のマイクロプログラムを生成するためのマシン独立なコード生成方式を開発した．従来高記述性をうたったマイクロプログラム開発システムの多くは，汎用性が乏しいか，レジスタ間のデータ転送径路が複数存在するような大型の商用機を対象としていないため実用性に問題があった．これに対して本方式ではユーザが記述した「マシン定義」と呼ばれる，マシンのハードウェア構成の記述を参照しながら処理が行われるため汎用性が高い．またユーザは実行したいデータ転送の転送元レジスタと転送先レジスタさえ記述すれば，システムが自動的に転送径路を探索するため記述性も良い．さらに，マシン内に複数のデータ転送径路がある場合も，マイクロ命令間のフィールド競合の起こらないビットパターンをバックトラックを行いながら探索するため，複雑な構造を有する大型の商用機にも適用できる．なお本方式を組み込んだマイクロプログラムトランスレータ MARTRAN は，現在10種類以上のプロセッサのマイクロプログラム開発に実用されており，開発工数の低減に大きく貢献している．

1. まえがき

近年，LSI メモリの低価格化に伴い，高機能な機械語命令の実現，OS やファイル処理の高速化のために，マイクロプログラム（以下 μ プログラムと略す）が大量に開発されるようになってきた．一方，μ プログラムの生産性は，ソフトウェアに比較して約1/5と低く，計算機の開発コストを押し上げる要因となっていた．生産性の低い主な要因は以下の2点である．

（1） ユーザがハードウェアに密着した細かい処理を記述しなければならない．特に複雑な処理装置ではデータの転送径路に沿ってデータが流れるように細かいゲートの開閉まで指定しなければならない．

（2） 水平型 μ 命令形式を採用している計算機では μ プログラムの並列実行可能性を考慮してコーディングを行わねばならない．

そのため，最近になって，開発ツールの必要性が認識されるようになり，種々の μ プログラム記述システムが開発されるようになってきた．これらのシステムの多くは，上記の二つの問題点に対応して，

（1） 記述性の向上

（2） 直列に記述した μ プログラムを自動的に並列 μ プログラムに変換する自動並列化機能

を特徴としている．本論文では，このうち主として(1)の記述性の向上について論じる．

μ プログラム記述システムをソースプログラムの記述性により分類すると，次のようになる．

（a） μ プログラムアセンブラ：各フィールドのビットパターンに対応するニモニック（英字名）をユーザが記述し，これをシステムがビットパターン形式の μ プログラムに変換する．

（b） レジスタ転送言語（以下 Register Transfer Language を略して RTL と呼ぶ）：μ プログラムで行う処理をマシン内のレジスタ間の転送・演算として記述しこれをシステムがビットパターンに変換する．さらに高級言語に近いデータタイプ（変数，定数，配列）や制御構造（DO LOOP, BEGIN ブロック）等を用いて μ プログラムを記述するシステムも提案されている．

上記の(b)に分類されるシステムでしかもマシン独立なものもいくつか作成されている[1)~4)]が，商用機向けの汎用 μ プログラム開発システムとしてはいまだに(a)に分類されるものが用いられている．

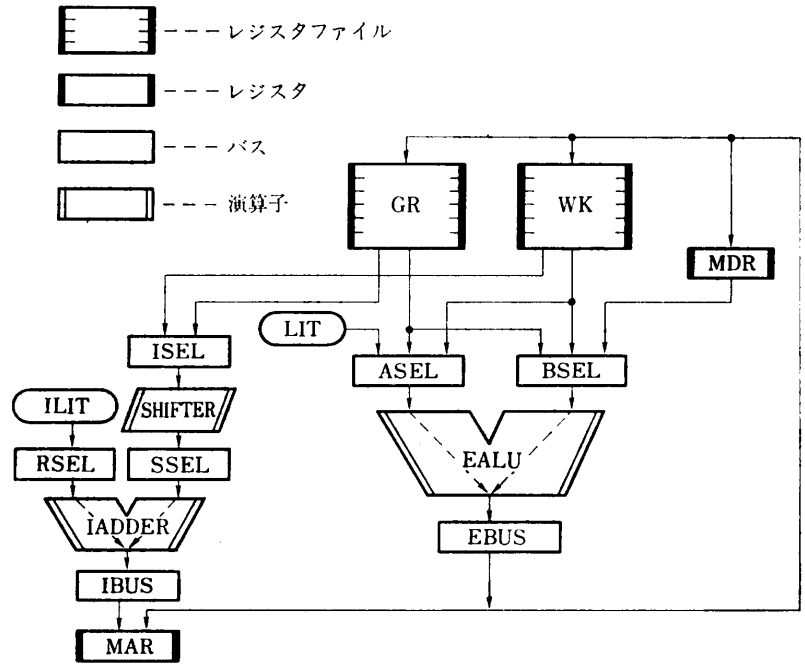
---

† Code Generation Method of Machine Independent Microprogram Translator MARTRAN by RYOSEI HIRAOKA, TADAAKI BANDOH, YASUSHI FUKUNAGA, KOUTARO HIRASAWA (Hitachi Research Laboratory, Hitachi, Ltd.), KOUSUKE SAKODA (Systems Development Laboratory, Hitachi, Ltd.), HIROAKI NAKANISHI (Omika Works, Hitachi, Ltd.) and TAKANOBU BABA (Faculty of Engineering, Utsunomiya University).

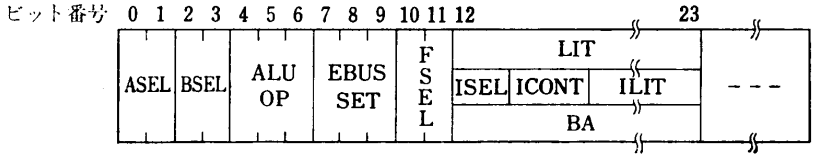
†† (株)日立製作所日立研究所
††† (株)日立製作所システム開発研究所
†††† (株)日立製作所大みか工場
††††† 宇都宮大学工学部情報工学科

* MARTRAN: Microprogram in Assembly language and Register transfer language TRANslator

** 現在 平岡工業(株)



(a) ハードウェア構成



(b) マイクロ命令フォーマット

図 1 マシンアーキテクチャの例
Fig. 1 An example of computer architecture.

その理由の一つは，最近は複数の ALU を内蔵し並列処理が可能な水平型の構造を持つ計算機が多くなったためである．**図 1** にこのような計算機の例を示す．このマシンは，二つの ALU (IADDER と EALU) を有し，命令フォーマットも数 10 ビットと長いため，1 μ 命令で複数の処理を同時に行うことができる．これを例にして，従来の高級言語指向の汎用 μ プログラム言語が実用にならなかった理由を説明する．

(1) 図 1 のような計算機では，同一のマクロ的な記述に対して複数の実現手段（すなわちビットパターン）がありうる．これらの複数のビットパターンのうちどれを選択するかは，同一マシンサイクルで他にどのような処理を行うかによって変わる．例えば図 1 のマシンで WK(5) というレジスタの内容を MAR に転送する処理を考える．これを RTL で記述すると

MAR←WK(5) ①

となるが，WK(5) から MAR に至る径路は図 1 (a) に明らかなように IADDER を通るものと，EALU を通るものの 2 通りがあり，それぞれ異なったビットパターンが対応しどちらを選ぶべきかは一意に決まらない．ただし，

MAR←WK(5), GR(5)←GR(5)〈+〉MDR ②

MAR←WK(5), →LAB ③

のように複数の処理を同一のマシンサイクルで実行させる場合には，上記二つのルートのうちいずれを取るべきかは決まってしまう．②の文では，後半の GR(5) と MDR の和を GR(5) に格納する処理は EALU を用いるので，前半の転送処理は IADDER を通るルートを用いる以外にない．③は，転送処理を行った後に LAB というラベルへの分岐を指定する文であるが，分岐処理は図 1 (b) の BA という命令フィールドを用いるため，そのフィールドと同一ビット位置を

用いる IADDER を制御する ICONT という命令フィールドを用いることはできない．したがってこの場合の転送径路は EALU を通る径路に限定される．

上記のようにユーザは RTL のようなマクロな記述以外にデータ転送径路まで指定せねばならず，結局(a)のような μ プログラムアセンブラに近い言語を使わざるを得なくなる．

(2) μプログラムは機械語が同一のマシンでもアーキテクチャが変われば内容が一変するため，μ プログラム記述システムは，どのマシンにも適用できる汎用性の高いものでなければならない．μ プログラムアセンブラであればニモニックと命令フィールドのビットパターンの対応さえ定義できる構造にしておけば，どのようなマシンにも使えるが，高級言語風の記述が可能なシステムを考えると，μ プログラムは細かいゲートの開閉まで指定するものであるから，各中間コードごとに，複雑なニモニックの組合せを何種類も定義しておく必要がありユーザの負担が大きい．例えば T1 と T2 を加えて T3 に格納する処理を示す．

(〈+〉, T1, T2, T3)

という中間コードを考えると，図1のマシンではT1, T2, T3 がどのレジスタであるかによって表1のような複雑なニモニックの組合せをあらかじめ用意しておく必要がある．

(3) μ プログラム記述システムでは，直列に記述した μ プログラムを自動的に並列 μ プログラムに変換する自動並列化機能を備えていることが望ましい．例えば

MAR←WK(5) ④

GR(5)←GR(5)〈+〉MDR ⑤

のように，④，⑤二つの文を直列にユーザが記述しこれをシステムが自動的に合成して

MAR←WK(5), GR(5)←GR(5)〈+〉MDR ⑥

という．1μ 命令で実行可能な一つの文に合成するわけである．システムが直列に記述された二つの文を合成できるには以下の2条件が成立しなければならない．

(イ) 先行する文④が変更したレジスタを後方の文⑤が参照していないこと．

(ロ) 二つの文が1マシンサイクルで実行可能なこと．すなわちリソースや命令フィールドの競合が発生しないこと．

このうち(ロ)の条件を確認する簡単なアルゴリズムは従来なかった．例えば，④の転送文を取ってみても

表1 (〈+〉, T1, T2, T3) に対するニモニック・リスト
Table 1 Mnemonic list of quadruple (〈+〉, T1, T2, T3).

変数	変数が表すレジスタ名	ニモニック・リスト
T1	GR	(%ALUOP=PLUS)∧{(%ASEL=AGRSEL)∨(%BSEL=BGRSEL)}
	WK	(%ALUOP=PLUS)∧{(%ASEL=AWKSEL)∨(%BSEL=BWKSEL)}
	MDR	(%ALUOP=PLUS)∧(%BSEL=BMDRSEL)
T2	GR	(%ALUOP=PLUS)∧{(%ASEL=AGRSEL)∨(%BSEL=BGRSEL)}
	WK	(%ALUOP=PLUS)∧{(%ASEL=AWKSEL)∨(%BSEL=BWKSEL)}
	MDR	(%ALUOP=PLUS)∧(%BSEL=BMDRSEL)
T3	GR	(%ALUOP=PLUS)∧(%EBUSSET=GRSET)
	WK	(%ALUOP=PLUS)∧(%EBUSSET=WKSET)
	MDR	(%ALUOP=PLUS)∧(%EBUSSET=MDRSET)
	MAR	(%ALUOP=PLUS)∧(%EBUSSET=MARSET)

注) ∧…論理積, ∨…論理和 を示す．

並列に実行される他の処理によって，ビットパターンが変わってくる．したがってあらかじめ，すべての文の組合せについて，一つ一つ並列実行可能性を検討し，テーブルを作成しておく以外にないが，この方法はユーザに多大の負担を強いることになり実用性に乏しい．

そこで本論文では RTL レベルソース文のコード生成を行う方式として，マシンのデータ構造をネットワークとして捉え[5]，このネットワークをユーザが記述しさえすれば，システムがこのネットワークをたどりながらコード生成を行う方式を導入し，ユーザのマシン記述の負担を軽くした．さらに，バックトラック処理を導入することにより，レジスタ間に複数のデータパスがある場合でも，同一命令中の他の処理との関連で，可能なビットパターンを正しく選択することができるようになった．

以下，第2章および第3章で，本方式を組み込んだ汎用 μ プログラム記述システムの言語仕様と処理方式を概説し，第4章以下で，本論文の主題となるコード生成方式の特長について詳しく論じる．

2. MARTRAN の言語仕様

MARTRAN はマシン独立性を実現するために，

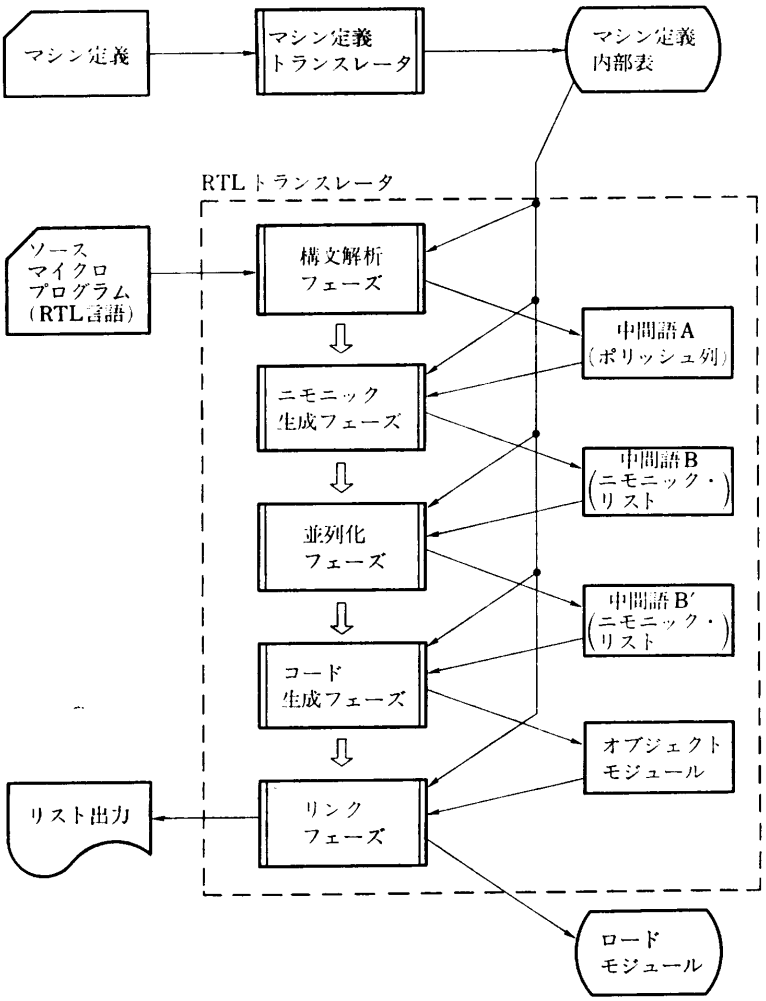


図 2 MARTRAN の処理システム構成
Fig. 2 System configuration of MARTRAN.

ユーザの記述した対象マシンに関する情報を記述した「マシン定義」を参照しながら，レジスタ転送言語 (RTL) で書かれた μ プログラムをビットパターン形式のロードモジュールに変換する構成をとっている (図2)．以下，マシン定義と RTL 言語の仕様を説明する．

(1) マシン定義の仕様

図1に示すアーキテクチャを有するマシンのマシン定義を表2に示す．

(a) フィールド定義表 (FDT)

μ 命令のフィールド名とビット位置，およびそのフィールドに許されるビットパターンと，そのビットパターンに与えた英字名（すなわちニモニック）を記述する(表2(a)参照)．

(b) 変数表 (VT)

レジスタ，レジスタファイル，バス等の，データ構造中のリソースの属性とその間のデータ転送について記述する．表2(b)は，MAR と EBUS の記述例を示している．すなわち，MAR は32ビットのレジスタで，EBUS および IBUS からそれぞれ MESEL および MISEL というニモニックを指定することによってデータが転送されて来ることが記述されている．ここで MISEL 等を，マシンのマイクロ操作*（以下 μ 操作と略す）を制御するニモニックという意味で制御ニモニックと称する．ここで複数のニモニックを同時にまたはいずれかを指定することによって動作が行われる場合には，ニモニックの論理式を記述することができる．表2(c)には論理式となる例を示した．

(c) 定数表 (CT)

定数の発生を記述する表である．表2(c)には，IUNITSEL および ILITSEL という制御ニモニックを指定することによって，% LIT という名称のフィールドの値が RSEL にセットされることを示している．

(d) 演算子表

ALU 等の演算器で実行できる演算子を記述する．各演算子ごとに，名称，入出力端子およびその演算を実行させるための制御ニモニックを記述する．表2(d)の例では，〈+〉という演算子は，入力端子が ASEL と BSEL，出力端子が EBUS で，演算を制御するニモニックは ADD であることを示している．

マシン定義には上記のほかに，μ 命令間の分岐方法を記述した表も含まれている．このようにマシン定義はマシンのデータ構造を宣言的に表現したもので，ハードウェア設計者が短期間で記述することができる．

(2) RTL 言語の仕様

ユーザは，上記のマシン定義で記述したレジスタ，レジスタファイル，定数，演算子等を用いて，RTL

* 本論文では，μ 操作とは，データ構造中で隣り合ったリソース間のデータ転送，演算，定数の発生，分岐等の基本的な動作の単位を言う．各 μ 操作とそれを制御する制御ニモニックはマシン定義で定義されている．

表 2 マシン定義
Table 2 Machine definition.

(a) フィールド定義表

```
FDT
  %ASEL    '0:1'
         AGRSEL    00B
         AWKSEL    01B
         ALITSEL   10B
  %BSEL    '2:3'
         BGRSEL    00B
         BWKSEL    01B
         BMDRSEL   10B
  %ALUOP   '4:6'
         PLUS      000B
         MINUS     001B
         AND       010B
        ⋮
END  FDT
```

(b) 変数表

```
VT
  MAR  REG  32
         EBUS '0:31' ⇒ '0:31' !MESEL!
         IBUS '0:31' ⇒ '0:31' !MISEL!
  EBUS  BUS  32
         ASEL '0:31' ⇒ '0:31' !APASS!
         BSEL '0:31' ⇒ '0:31' !BPASS!
        ⋮
END  VT
```

(c) 定数表

```
CT
%ILIT '0:9' ⇒ RSEL '0:31' !IUNITSEL∧ILITSEL!
        ⋮
END  CT
```

(d) 演算子表

```
OPT
      〈+〉   (ASEL BSEL) ⇒ (EBUS) !ADD!
      〈−〉   (ASEL BSEL) ⇒ (EBUS) !MINUS!
     〈EOR〉  (ASEL BSEL) ⇒ (EBUS) !EOR!
        ⋮
END  OPT
```

形式のマイクロプログラムを記述することができる.

例えば, 図1のマシンで, ユーザが「100B (2進の 100) と GR というレジスタファイルの5番地の内容を加算して, レジスタ MDR にセットする」処理を記述したいとする. RTL 言語を用いると, ユーザはデータの転送経路やセレクタの制御等を一切意識する必要はなく, 図 3(b)のような転送元レジスタ, 転送先レジスタ, 定数, 演算子名のみを用いて, マクロな意味を記述しさえすればよい.

また, 代入文のほかにも, 表3に示すようにビットセット文や各種の分岐文が記述できる.

```
ALITSEL, BGRSEL,
%GRADDR=5, MDRSET,
%LIT=100B, PLUS.
```

⇨

```
MDR←100B〈+〉GR(5).
```

(a) マイクロプログラムアセンブラによる記述　(b) MARTRAN による記述

図 3 MARTRAN による記述性の向上
Fig. 3 Enhancement of readability by MARTRAN.

表 3 MARTRAN の RTL 文 (宣言文を除く)
Table 3 RTL statements of MARTRAN.

文の種類	内　容	例
代入文	右辺の式の値を左辺の変数に代入する.	MDR←WK(5). GR(4)←WK(5)〈−〉MDR. MAR←01B〈I+〉(〈SL8〉WK(4)).
ビットセット文	変数の特定ビットを0/1にセットする. 無条件セットと条件付セットがある.	TRAPFF⇐0. |IOVFSET|MICRST'3'⇐1.
カウント文	変数の値を増減する.	2<COUNT.
GOTO 文	指定したラベルに分岐する.	→LAB2.
条件分岐文	テスト条件の成否により指定したラベルに分岐する.	|COUNT-NE-0|→ODD. |BCCOND|→L1 ; →L2.
CASE文	変数の値に従って多方向に分岐する.	?MJ '0:1' |_00B→L1 |_01B→L2 |_10B→L3 |_11B→L4.
CALL文	マイクロプログラムサブルーチンを呼び出す.	→SUBX.
RETURN文	マイクロプログラムサブルーチンより戻る.	⇒.
間接分岐文	特定変数中に格納されたアドレスに分岐する.	→(INSTDEC).
フィールド直接指定文	マイクロ命令の各フィールドのビットパターンまたはニモニックを直接指定する.	%GRADDR=0010B. #AGRSEL.
制御文	複数のニモニックの組合せに英字名を付けて直接指定する.	READWAIT.

さらに水平型の μ プログラム方式のマシンでは一般に一つの μ 命令で複数の異なった処理が実行できる. このような場合には, 一行に複数の RTL 文を, ", " で区切り記述すれば, 処理システムがこれら複数の処理を同時に実行可能な μ 命令を自動的に生成する.

3. MARTRAN の処理方式

MARTRAN の処理系は, ユーザの記述した マシ

```
MAR←100B⟨I+⟩(⟨SL8⟩GR(5))
```

(a) RTL ソースプログラム

代入文
レジスタ
MAR
定数
100B
レジスタファイル
GR
5
演算子
<SL8>
演算子
<I+>

(b) 中間語A (ポリッシュ列)

```
((%ISEL=GRSEL)
 (%ICONT=IPLUS)
 (%ISHIFT=SL8)
 (%GRADDR=101B)
 (%MARSEL=IBUSSEL)
 (%ILIT=100B))
```

注) 内部ではリンクト・リスト形式で格納されている.

(c) 中間語B (ニモニック・リスト)

図 4 中間語の形式
Fig. 4 Formats of intermediate languages.

ン定義を計算機で処理しやすいマシン定義内部表に変換するマシン定義トランスレータと，ユーザが RTL 言語で記述した μ プログラムをビットパターン形式の μ 命令に変換する RTL トランスレータとから成る (図2参照).

以下システムの主要部分である RTL トランスレータの各フェーズにおける処理を例を用いて概説する.

(1) 構文解析フェーズ

RTL 言語で書かれたプログラムを入力し，マシン定義との対応をチェックしながら構文解析を行い，ポリッシュ列を主体とした中間語Aを出力する. 図4(a)に示す RTL 代入文に対する中間語Aの形式を図4(b)に示す.

(2) コード生成フェーズ I

マシン定義を参照しながら，中間語Aを記述された処理を行うために指定すべきニモニック (μ 命令フィールドのビットパターンに付けられた英字名) のリストを決定し中間語Bとして出力する. 図4(b)のポリッシュ列に対応するニモニック・リストを図4(c)に示す. 中間語Bを生成する過程で，以下の処理をマシン独立な手法を用いて行う.

(i) データ転送径路の決定

RTL 言語 (または中間語A) の実行文では，レジスタや演算子の名称しか記述されていない. これらの間のデータ転送径路の決定は，処理システムが自動的に行う.

(ii) 定数発生源の決定

マシンに複数の定数発生源がある場合でも RTL 言語 (または中間語A) の文では，定数の値しか記述されていない. どの定数発生源を用いるかは，処理システムが自動的に決定する.

(iii) μ 命令フィールドの競合チェック

μ 命令は，少しでもビット長を短く押さえるため同一ビット位置に複数のフィールドが定義されることが多い (図1(b)の μ 命令の 12〜23 ビット目参照). したがって，マシンのデータ構造のみを考えると，一つの μ 命令で実行できそうに見える RTL 文 (の列) であっても，実際にビットパターンに変換してみると，フィールド間の競合が発生し，対応する μ 命令が合成できない場合がある. そこで，処理システムはこのようなフィールドの競合を自動的にチェックし，もし競合している場合には，競合が生じないようにデータ転送径路や定数発生源の選択の組合せを捜す処理を行っている.

(3) 並列化フェーズ

水平型の μ プログラムマシンでは，複数の操作が一つの μ 命令で実行できる. 本フェーズでは，ユーザが直列に記述した複数の RTL 文に対応する中間語Bを一つの μ 命令に詰め込む，所謂並列化を行っている. 並列化に際しては，μ 命令のフィールド競合が起こらないこと，および文の順序を変えたことによりプログラムの意味が変わらないことを考慮している. 本フェーズの出力は中間語Bと同じニモニック・リスト形式の中間語 B′ である(詳細は文献 6)参照).

(4) コード生成フェーズ II

ニモニック・リスト形式の中間語 B′ をマシン定義のフィールド定義表を参照しながらビットパターン形式の μ 命令に変換する.

(5) リンクフェーズ

マシン定義中に記述された μ 命令間の分岐方法を

参照して，各 μ 命令にアドレスを割当てるとともに μ プログラムモジュール間のリンクを行いロードモジュールを作成する．

次章以下では，上記の処理のうちコード生成フェーズについて詳しく述べる．

4. コード生成処理方式

4.1 基本処理

本節ではコード生成処理の基本部分を説明し，コード生成が失敗した時の対応については次節で述べる．

まず，一つの RTL 文に対応する中間語Aで規定された動作を行うのに必要な μ 操作の集合を求める．各 μ 操作を実行するのに必要な制御ニモニックはマシン定義に記されているので，それらのフィールド競合を考慮しながら合成しビットパターンに変換した後一つの μ 命令に埋め込む．また，一行に複数の文が記述された場合にも，これらの文に対応するニモニック列を合成し，すべて同一の μ 命令に埋め込む．

RTL 文の中で代入文以外の文は，対応する μ 操作が一意的に決まるが，代入文は一般に一つの文が数多くの μ 操作から成る上，文の機能を実現するのに必要な μ 操作の集合が複数組存在する．そこで，以下代入文のコード生成処理の概要を述べる．

（1） 中間語Aの解釈（代入文の場合）

中間語Aを解釈してニモニック・リストを生成するには，中間語Aを

（a） RTL 文の右辺の変数または定数から，演算子の入力端子または左辺の変数に至る転送

（b） 演算子で規定された演算

（c） 演算子の出力端子から RTL 文の左辺の変数に至る転送

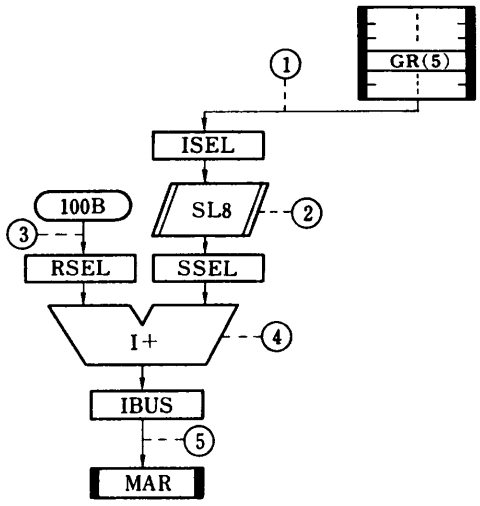
（d） 演算子の出力端子から RTL の左辺の変数に至る転送

等の動作に分解し，これらの動作を制御するニモニック・リストを求めて合成する．

図4の RTL 文に対するニモニック合成例を**図5**に，具体的な処理手順を**図6**に示す．

上記の処理中で，定数または変数から他の変数へのデータ転送を制御するニモニック・リストの作成が必要になるが，一般に実際のマシンではこのような動作を実現する μ 操作の集合は複数組存在する．次にこれらの複数組の集合を求め，その中から一組を選択する方法を述べる．

（2） 転送路および定数源の選択



（a） 転送径路

マイクロ操作の番号	μ 操作の内容	制御ニモニック
①	GR(5)⇨ISEL	(%GRADDR=101B) ∧GRSEL
②	⟨SL8⟩	SL8
③	100B⇨RSEL	(%ILIT=100B)
④	⟨I+⟩	IPLUS
⑤	IBUS⇨MAR	IBUSSEL
	GR(5)⇨MAR	((%ISEL=GRSEL) (%ICONT=IPLUS) (%ISHIFT=SL8) (%GRADDR=101B) (%MARSEL=IBUSSEL) (%ILIT=100B))

（b） ニモニック・リストの生成

図5 コード生成の過程（図4（a）の RTL 文の場合）
Fig. 5 Process of code generation.

変数間の転送を制御するニモニック・リストを作成するには，まず変数をノードとしノード間の転送路をリンクとするネットワークをマシン定義の変数表から作成する．次にこのネットワーク上で転送元から転送先に至る径路を短い順に求める．ただし1マシンサイクル内での処理を考えているので，データ保持能力のある変数（レジスタ，レジスタファイル）を通過する径路は除外する．リンクに相当する直接転送路にデータを転送する μ 操作を制御する制御ニモニックは変数表に記述されているので，求める転送径路を構成するすべての直接転送路についての制御ニモニックを合成することによって，目的とするニモニック・リストが作成される．

例えば，MDR から MAR へデータを転送しよう

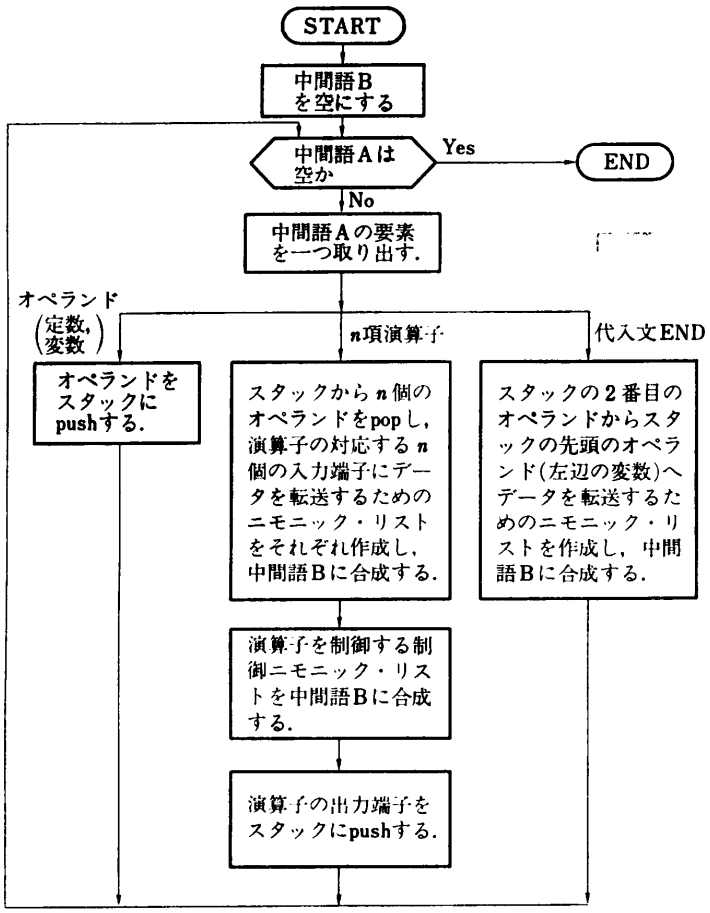


図 6 代入文のコード生成処理
(1行に RTL 1文のみの場合)
Fig. 6 Code generation of assignment statement.

とすると, MDR→BSEL, BSEL→EBUS, EBUS→MAR の四つの直接転送路にデータを転送する μ 操作を実行する必要があるから, これらを制御する制御ニモニックを変数表から取り出し, 合成すればよい.

ここで転送路が複数本ある場合は, 短いものを優先して利用するが, ある径路に対するニモニック・リストは, 同一マシンサイクルで行われる他の処理のニモニック・リストとフィールド競合を起こし命令が合成できない場合がある. その時は次に短いルートを選択する.

定数を変数に転送する処理も同様に, 転送先に一番近い定数発生源から順番に候補を選び, RTL 文に記述されている定数の値を発生できるか, 定数発生に必要な制御ニモニックを, 他の動作のニモニック・リストと合成できるかの二つの基準で決定する.

4.2 バックトラック処理

前節で MARTRAN のコード生成手法について述べたが, 実際の水平型マシンを対象に考えると, 決定論的 (deterministic) なアルゴリズムで, マシン独立なコード生成を行うことはできない. 次にその理由を述べる.

実用的な μ プログラム方式のマシンは一般に次の性質がある.

(1) 単独では実行可能な動作も, 同一マシンサイクルで実行される他の動作が, マシン内のリソース (転送路, 演算器, 定数発生源) や μ 命令のフィールドをすでに使っているため実行できない場合がある.

(2) 変数間のデータ転送, 定数の発生, 演算等の動作は, 同一機能を複数の方法 (すなわち複数のニモニック・リスト) で実行できる場合が多い.

したがって, ある動作を実行したい時, どのようなニモニックを用いるべきかは, あらかじめ一意に決定することはできず, 一行に記述されているすべての RTL 文に対するニモニック・リストを作成し合成してみなければわからない.

そこで, 上記の μ プログラムの性質を踏まえ, 実行可能なコードを作り出すため, コード生成の過程でフィールド競合が発生した場合に, 以前に選択した転送径路や定数発生源を見直し, 他の候補を選択してから再び処理をやり直すいわゆるバックトラック処理を導入した.

具体的には, **図 7** に示すデータ構造を利用して**図 8** の処理を実行する. 処理を元に戻して他の選択肢を選択するには, 最後に何番目の選択肢を選択したかを記憶しておく必要がある. この情報を記憶するテーブルが図 7 に示した選択肢番号表である. この表は各 RTL 行の処理始めに初期化され, 処理システムが中

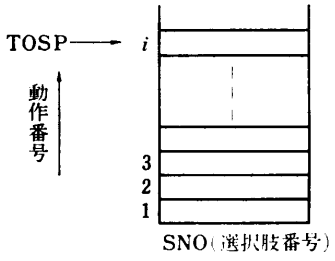


図 7 選択肢番号表
Fig. 7 Stack mechanism for backtrack.

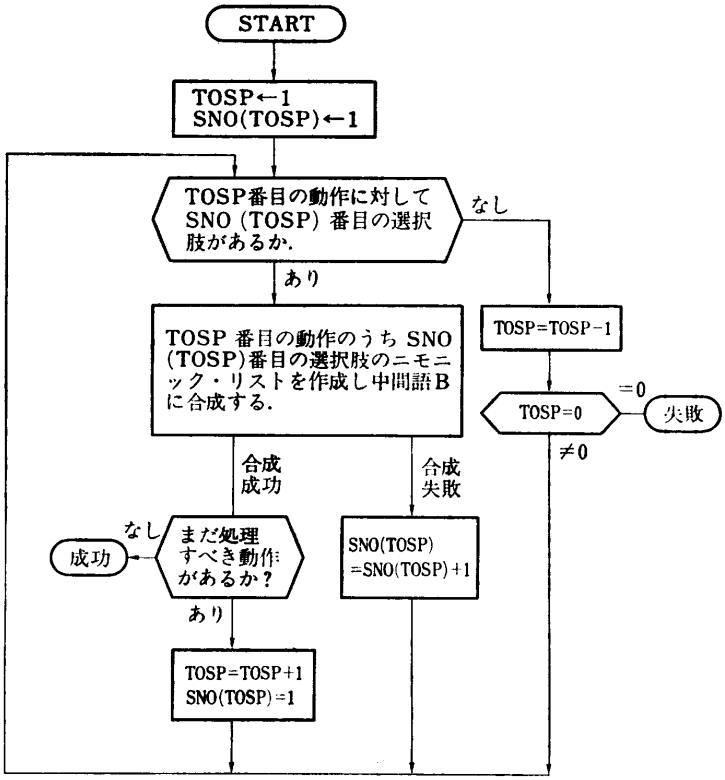


図 8 コード生成時のバックトラック処理
Fig. 8 Backtracking algorithm of code generation.

間語Aを解釈して変数間のデータ転送，定数発生，演算，分岐といったまとまった動作を実行するのに必要なニモニック・リストが作られるたびに，複数ある実現方法のうち何番目を選択したかを選択肢番号として，この表に次々にスタックしておく．ここで TOSP をスタックの先頭を指すポインタとすると TOSP に対応する動作のコード生成中は，成功するまで次々に別の選択肢を試みる．もはや選択すべき候補がなくなった時には TOSP を一段下げて，新たな TOSP で指された選択肢番号に1を加え再びコード生成を試みる．

表 4 転送径路の変化
Table 4 Variation of transfer route.

行 No.	RTL 行	WK(5) から MAR への転送径路
①	MAR←WK(5), →LAB.	WK(5)⇨ASEL⇨EALU ⇨EBUS⇨MAR
②	MAR←WK(5), GR(5)←GR(5)〈+〉MDR.	WK(5)⇨ISEL⇨SHIFTER ⇨SSEL⇨IADDER⇨IBUS ⇨MAR
③	MAR←WK(5), GR(5)←GR(5)〈+〉MDR, →LAB.	(コード生成失敗)

次に前記のマシンの場合にバックトラックの起こる例を紹介する(表4参照)．

今，図1のマシンで WK(5) から MAR へデータを転送しようとすると，図1(a)のように EALU を通る径路と IADDER を通る径路の2通りが考えられる．このどちらを選ぶかは同一マシンサイクルで他にどのような動作を行うかによって変わる．

例えば，表4に示した3行の RTL を例にとると，まず①では，二つの径路のうち短い方の EALU を通る径路が選ばれるが，同一マシンサイクルで実行される分岐文はこの径路とフィールド競合しないのでコード生成は無事終了する．

次に②の行では，同じ転送文に対していったん EALU を通る径路が選択されるが，GR(5) に値を代入する RTL 文が EALU を使用するため，%ALUOP フィールドで競合が発生しコード生成は失敗する．するとここでバックトラックが起こり，第1の RTL 文に対して，IADDER を通る径路が選択された後，再び第2の RTL 文の処理に移る．今度は第1の文で EALU を使っていないので第2の文のコード生成も成功する．

③の行では，第1の転送文でもし EALU を通る径路が選ばれると，GR(5) に値を代入する代入文とフィールド競合が起こるし，もし IADDER を通る径路が選ばれると，%ICONT フィールドが，分岐文が用いる%BA フィールドと競合するので，いずれもコード生成は失敗する．このような場合は処理システムはあらゆる可能な選択の組合せを調べた後，エラーメッセージを出力する．

5. MARTRAN のコード生成方式の効果

前章で述べたコード生成方式を採用することによって，以下の効果が得られる．

(1) ユーザは，データ転送径路や定数源の選択を行う必要がない．データ転送元レジスタ，定数の値，演算子の種類さえレジスタ転送言語 (RTL) の形で記

述すれば，処理システムが自動的に μ 命令を生成する．

（2） 同一 μ 命令で複数の処理が可能な水平型マシンにおいては，一つの RTL 文に対して複数のビットパターンが生成されるが，本システムでは，バックトラック処理を行っているので，同一命令で実行される他の処理と競合しないコードを自動的に選択する．

（3） マシン記述は，計算機のデータストラクチャをネットワーク形式で記述する方式を取っているため，コード生成規則を中間コードごとに記述する方式等に比べ，記述が格段に容易になり，実用レベルの汎用性・マシン独立性が得られた．

（4） 自動並列化処理を行う際には，並列に実行したい複数の RTL 文をまとめて 1 μ 命令の処理として本方式によるコード生成ルーチンにかければ，同一 μ 命令で実行可能な場合は目的とするコードが生成されるほか，もしリソースやフィールドの競合があり，1 μ 命令で実行できない場合はエラーメッセージを返してくる．したがって並列化処理は本コード生成処理を用いることによって容易に実現可能となる．

なお，MARTRAN は，実際に作成され，実用されている．システムは FORTRAN 22.6 K ステップ，アセンブラ 1.8 K ステップから成り，M180H 上で実現されている．処理時間は，バックトラックを含んだコード生成処理や並列化処理等複雑な処理を行っているにもかかわらず，RTL 一行当り平均約 44 ms にすぎない．

また，本システムは，日立の制御用スーパーミニコンピュータ HIDIC-V90/50 の CPU から，各種 OA ワークステーション用専用プロセッサやマイクロプロセッサまで，10 種類以上の異なったプロセッサの μ プログラム開発に実際に用いられており，その汎用性は十分実証された．

さらに MARTRAN を使うことによって，従来用いられていた汎用 μ プログラムアセンブラに比べ，μ プログラムの生産性が約 2 倍になることが報告されている．これは，ソース言語が RTL という記述性の高い言語になったこと，自動並列化機能の効果，ドキュメント性の向上によるデバッグ時間の減少等によるものである．

6. あとがき

記述性の高いレジスタ転送言語をソースとして，ビットパターン形式の μ プログラムを生成するための，マシン独立な μ プログラムコード生成方式を開発した．

本方式は，マシンのデータ構造をネットワークとみなして，ソース文で記述されたレジスタ間のデータ転送径路を探索しながらコードを生成してゆく処理を基本とし，1 μ 命令内で実行するよう記述された複数の動作の間に μ 命令フィールドの競合が起こった場合にもバックトラックを行いながら，フィールド競合の起こらないコードパターンを自動的に探すことができる．

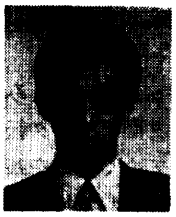
その結果，ユーザは，データの転送径路や定数発生源の選択，フィールドの競合等を考慮することなく，μ プログラムを記述できるようになり，μ プログラム開発工数が半減し，ソース μ プログラムのドキュメント性も向上した．

謝辞 本研究に際しご指導いただいた宇都宮大学情報工学科奥田健三教授，日立製作所日立研究所所長川本幸雄博士，同第 8 部部長西原元久博士，同大みか工場井手寿之氏，ならびに本システムの開発にご尽力いただいた日立プロセスエンジニアリング太田孝徳氏ほか多数の関係者の方々に深く感謝いたします．

参考文献

1) Eckhouse, R. H.: A High Level Microprogramming Language (MPL), *Spring Joint Computer Conference*, pp. 169-177 (1971).
2) Ramamoorthy, C. V. and Tsuchiya, M.: A High Level Language for Horizontal Microprogramming, *IEEE Trans. Comput.*, Vol. C-23, No. 8, pp. 791-801 (1973).
3) Yamamoto, M. et al.: A Microprogrammed Computer Design and Evaluation System, *First USA-JAPAN Computer Conference*, pp. 139-144 (1972).
4) Dubbs, E. D. et al.: A Microprogram Design System Translator, *Compcon 72, Sixth Annual IEEE Computer Society International Conference*, pp. 95-98 (1972).
5) 馬場，藤本，萩原：MPG マイクロプログラムコンパイラ，情報処理，Vol. 19, No. 1, pp. 16-25 (1978).
6) 迫田行介：拡張データ依存関係グラフを用いたマイクロプログラムの大局的並列化法，情報処理学会論文誌，Vol. 23, No. 3, pp. 304-311 (1982).

（昭和 60 年 10 月 3 日受付）
（昭和 61 年 9 月 10 日採録）



平岡　良成（正会員）

昭和26年生．昭和49年東京大学工学部電気工学科卒業．52年UCLA計算機科学科修士課程修了，53年東京大学工学部修士課程修了．同年日立製作所日立研究所入社．59年より平岡工業(株)勤務，現在に至る．プログラミング言語，計算機アーキテクチャ，自然言語処理，数値制御等に興味を持つ．ACM 会員．

坂東　忠秋（正会員）

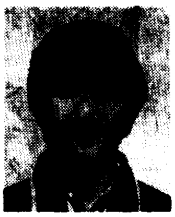
昭和43年東京大学工学部電気工学科卒業．同年日立製作所入社．日立研究所にて，制御用計算機，マルチプロセッサ，画像処理，ワークステーション等の研究に従事．昭和51年スタンフォード大電気工学科修士．昭和60年東京大学より工学博士受理．現在，日立研究所主任研究員．

福永　　泰（正会員）

昭和25年生．昭和50年京都大学大学院修士課程電気系工学研究科修了．同年(株)日立製作所入社．現在同社日立研究所研究員．計算機アーキテクチャ，グラフィックス，マンマシンインタフェースの研究に従事．電子通信学会，ACM 各会員．

平沢宏太郎（正会員）

昭和16年生．昭和39年九州大学電気工学科卒業．昭和41年同大学大学院修士課程修了．同年(株)日立製作所入社．工学博士．最適制御，制御用計算機，画像処理，ワークステーション，知識工学の研究に従事．現在，同社日立研究所第10部部長．



迫田　行介（正会員）

昭和20年生．昭和43年東京工業大学理工学部制御工学科卒業．昭和45年同大学院修士課程修了．同年(株)日立製作所中央研究所に入社．昭和51年同社システム開発研究所勤務，現在に至る．システム・ソフトウェア，ファームウェア，言語の研究・開発に従事．IEEE 会員．

中西　宏明

1970年東京大学工学部電気工学科卒業．同年(株)日立製作所入社．同社大みか工場にて制御用計算機システムのハードウェア開発．オペレーティング・システムの開発・計画に従事．1978年米国スタンフォード大学計算機学科に留学．1979年 MS 取得．現在同工場計算制御設計部主任技師．

馬場　敬信（正会員）

昭和22年生．昭和45年京都大学工学部数理工学科卒業．昭和50年同大学院博士課程修了．工学博士．同年より電通大助手，講師を経て，現在宇都宮大学工学部情報工学科助教授．昭和57年より1年間メリーランド大学客員教員．計算機アーキテクチャ，データベースシステムなどの研究に従事．著書「マイクロプログラミング」(昭晃堂)．電子通信学会，IEEE 各会員．