マイクロプログラムの設計自動化

山田昭彦　川口明洋　高橋篤年　加藤俊一
（日本電気株式会社）

1　はしがき

近年マイクロプログラムの技術は，計算機の中央処理装置，周辺制御装置および端末装置等の種々の領域で使用される様になって来た。マイクロインストラクションは初期の水平形から，機械語に類似した垂直形，さらに最近の1チップ　マイクロコンピュータでは可変長が採用される等，多種多様のものが存在する。

また装置構成の面から見ると，周辺装置の制御を中央処理装置内に統合した統合制御法が採用される様になり，システム構成に適合したマイクロプログラムの生成が要求される様になって来た。

この様に種々の形式のマイクロインストラクションが存在する（種々の機種または装置が存在する）内で，マイクロプログラムの効率良い開発を行うためには，汎用性をもつ設計補助システムが要求される。

Microprogramming Design support System（以下MDSと称する）は，この様な要求を解決するために開発された。

MDSは以下の様な特徴をもっている。

イ）汎用性

マイクロインストラクションの形式，シンボリックなマイクロコマンドに対応するビットパターン，およびシーケンシングの方法等，ハードウェアに依存する部分はマイクロプログラムの定義部として記述することが出来る。DAシステムは，これ等の定義を参照することによって，種々の形式のマイクロインストラクションから成るマイクロプログラムの処理を行う。

ロ）モジュール　プログラミング

一般にマイクロプログラムは，ある機能をもったモジュールの集合であると考えられる。MDS　では各モジュールを個々にアッセンブルしておき，必要なモジュールのみを呼び出して1個のマイクロプログラムを作り上げる。モジュール　プログラミングは開発の際のデバッグが容易な外に，統合制御法を採用した装置のシステム構成に依存したマイクロプログラムを生成する場合に威力を発揮する。

ハ）ライブラリ機能

マイクロプログラムの定義部は，ライブラリとして磁気テープ，または磁気ディスクへ格納しておくことが出来る。そしてマイクロプログラムのアッセンブル時に，制御カードでその等を指定するだけで，各マイクロプログラム　モジュール毎に定義部を記述する必要はない。

FIG. 1　にMDSのゼネラルフローを示す。マイクロプログラムの開発

と製造に対するMDSの主機能は，アッセンブル，ファイル メンテナンス，フローチャートの自動作成，およびポストプロセスである。ポストプロセスは，装置およびマイクロプログラムのデバッグ，製造検査のためにアッセンブルされたビットパターンを紙テープ，カード，カセット磁気テープ等の媒体へ出力する。

2 マイクロプログラムの記述言語

種々の形式のマイクロインストラクションから成る，種々のマイクロプログラムを記述出来る汎用性をもったアッセンブリ言語を開発した。この言語はDEFINE セクションと，MICROPROGRAM セクションと呼ばれる，2つのセクションから成る。DEFINE セクションではハードウェアに依存する各種パラメータ，マイクロインストラクションの形式，シンボリック マイクロコマンドとビットパターンの対応を記述し，MICROPROGRAM セクションでは，マイクロインストラクションのシーケンスを記述する。

< microprogram > :: = < DEFINE section >
< MICROPROGRAM section >

2.1 DEFINE セクション

既述の如くこのセクションはライブラリとして登録しておくことが出来，アッセンブルの際それ等を参照することによって処理が実行される。DEFINE セクション内には，以下の5つのセクションが存在する。

< DEFINE section > :: = < PARAMETER section >
< FIELD DEFINE section >
< LABEL DEFINE section >
< COMMAND DEFINE section >
< MACRO DEFINE section >

1) PARAMETER セクション

PARAMETERセクションはマイクロプログラムが格納される制御メモリの容量，1ワードの長さ（ビット長），シーケンシングの方法（インクリメント方式か，ブランチ方式か），リザーブエリア等をアッセンブラに知らせる。

< PARAMETER section > :: = PARAMETER
[< memory capacity >]
< word length >
[< sequencing method >]
[< reserved area >]

< memory capacity > :: = MEMORY CAPACITY < number of words >

< word length > :: = WORD LENGTH < number of bits > BITS

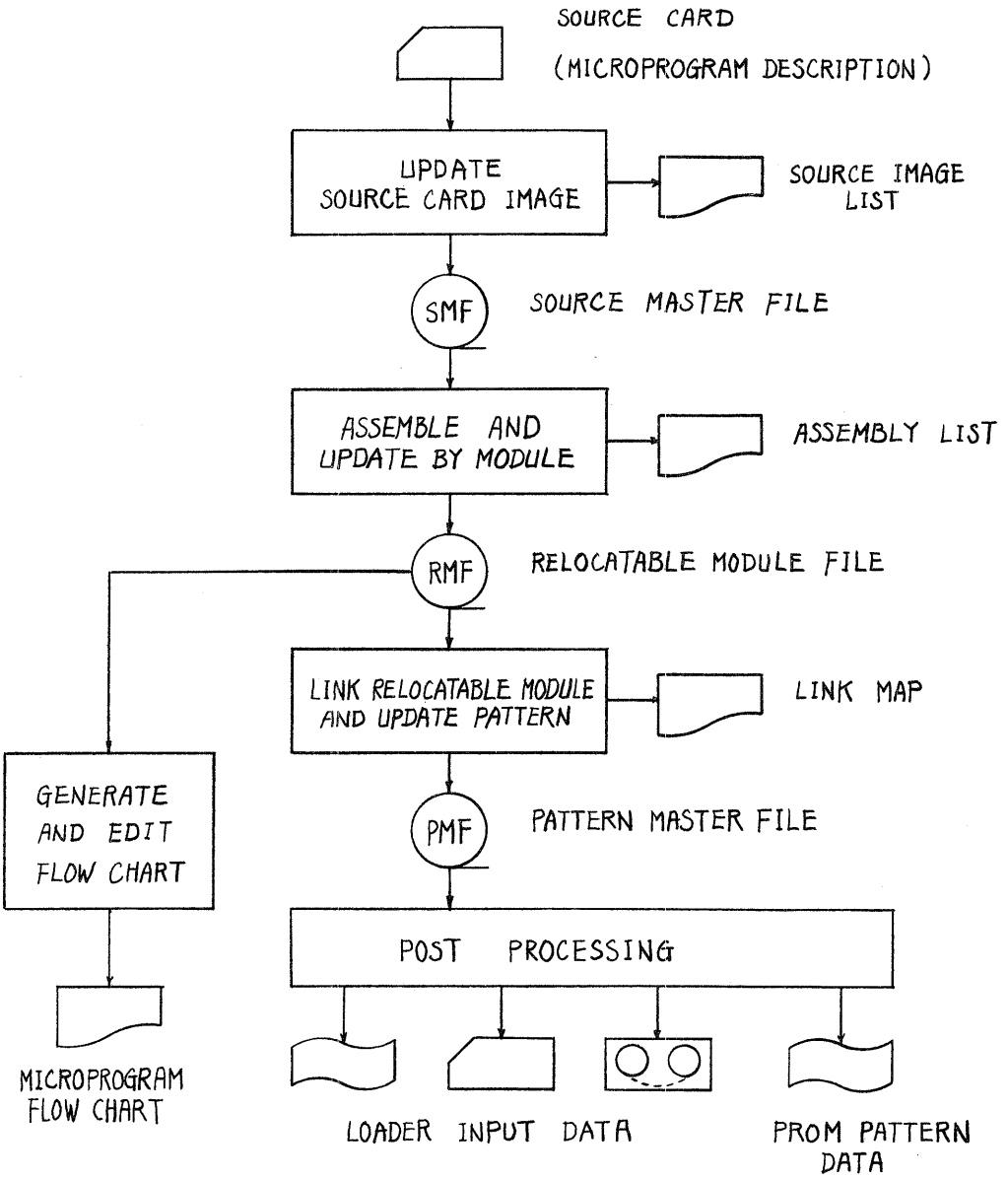


FIG. 1 GENERAL FLOW OF MDS

<sequencing method> ::= <sequence type> SEQUENCE
[<increment value>]

<sequence type> ::= INCREMENT | BRANCH

<reserved area> ::= RESERVED AREA <address value>

ロ) FIELD DEFINE セクション

FIELD DEFINE セクションは，マイクロインストラクション ワードの形式を定義する。一般にマイクロインストラクションの1ワードは，いくつかのフィールドに分割される。これらの各フィールドにおける名前，フィールドの長さ（ビット長），およびフィールドの位置を定義する。

<FIELD DEFINE section> ::= FIELD DEFINE
<field definition>
[<field definition> ・・・]

<field definition> ::= <type definition>
<field name definition>
<next address field definition>
<parity definition>

<type definition> ::= <type name>
[<relative branch definition>]

<field name definition> ::= <field name><field length>

<next address field definition> ::= ADDRESS <field length>

<parity definition> ::= PARITY <field length>
<parity type><check field>

<parity type> ::= ODD | EVEN | MARK

type の定義はマイクロインストラクション ワードの形式毎に与えねばならない。<relative branch definition> が指定されると，ブランチするマイクロインストラクションとブランチ先のアドレスとの差が ADDRESS と名付けられたフィールドへ格納される。省略されるとブランチ先の絶対アドレスが ADDRESS フィールドへ格納される。

ハ) LABEL DEFINE セクション

LABEL DEFINE セクションでは，MICROPROGRAM セクション内で定義されず，参照のみがされるレーベルに相対または絶対アドレスまたはビットパターンを与える。従って MICROPROGRAM セクション内で定義さ

れるレーベルをここで定義することは出来ない。

```
< LABEL DEFINE section > ::=  LABEL DEFINE
                               < label definition >
                              [< label definition > ・・・]

< label definition > ::= < label name >< constant > |
                         < label name >< linkage symbol name >
```

他のモジュール内で定義されているレーベルを参照する場合は，リンケージシンボルにより行う。リンケージシンボルはアッセンブラでなくて，リンカー内で処理される。

ニ) COMMAND DEFINE セクション

COMMAND DEFINE セクションでは，マイクロインストラクションの要素であるマイクロコマンドのシンボリック名，格納されるフィールド名，およびビットパターンを定義する。

```
< COMMAND DEFINE section > ::=  COMMAND DEFINE
                                 < command definition >
                                [< command definition > ・・・]

< command definition > ::= < type name definition >
                            < micro command definition >
                           [< micro command definition > ・・・]

< type name definition > ::= < type name >
                             [< field name >< bit patterm >]

< micro command definition > ::= < symbolic name >
                                 < field name >< bit patterm >
```

type 名と field 名は FIELD DEFINE セクション内で定義されていなければならない。

ホ) MACRO DEFINE セクション

MACRO DEFINE セクションでは1ワードを構成する複数個のマイクロコマンドをまとめて名前（マクロ名）を付け，マクロコマンドとして定義する。MICROPROGRAMセクションはこのマクロコマンドとマイクロコマンドで記述することにより，マイクロプログラムのコーデングを簡潔にし，内容の理解を容易にする。

マクロ名で代表される複数個のマイクロコマンドは COMMAND DEFINE セクションで定義されていなければならない。

< MACRO DEFINE section > :: = MACRO DEFINE
< command define for macro >
[< command define for macro > ・・・]

< command define for macro > :: = < type name >
< macro-command definition >
[< macro-command definition > ・・・]

< macro-command definition > :: = < macro name >
< micro-command symbolic name >
[< micro-command symbolic name > ・・・]

2.2 MICROPROGRAM セクション

MICROPROGRAM セクションでは，COMMAND DEFINE セクションで定義されたマイクロコマンドと，MACRO DEFINE セクションで定義されたマクロコマンド，レーベル，および定数を用いてマイクロプログラムを記述する。各ワードは，ワードに付けられたレーベルによって他のワードから参照される。

< MICROPROGRAM section > :: = MICROPROGRAM
[< base address >]
< microinstruction >
[< microinstruction > ・・・]

< base address > :: = BASE ADDRESS < address value >

< microinstruction > :: = [< address assignment >]
[< label name >][< branch address >]
< micro command list >

< micro command list > :: = < type name >
< micro command sybolic name >
[< micro command symbolic name > ・・・]
[< constant >]

3 アッセンブルとファイルメンテナンス

FIG.1 に示す如く，シンボリックなマイクロプログラムからバイナリーイメージを得るためには，ソースカードイメージのアップディト，アッセンブル，およびリンクの３つのプロセスを実行する必要がある。

3.1 ソースカードイメージのアップディト

ソース マイクロプログラムはソースマスターファイル（SMF）の形で保存される。ソースカードデックを読込むと，SMF 内に新しいソースマイクロプログラムが登録され，変更カードを読込むと SMF 内の指定さ

れたソース　マイクロプログラムがアップデイトされる。1個のソース
マイクロプログラムは，DEFINE セクションとMICROPROGRAM セクション
の両方を含むことも出来るが，既述の如く．DEFINE　セクションは，
ライブラリとして別に登録する方が好ましい。
　プログラムはアップデイトを行う際，その更新の経厂を保持していく
ことが出来る。従って試行的な変更や．過去の古い状態へ戻ることが容
易に出来る。

3.2　アッセンブル

　SMF 内のソース　マイクロプログラムはアッセンブラによってリロケー
タブルモジュールファイル（RMF）に変換される。アッセンブルの際，
ソース　マイクロプログラムが DEFINE セクション　データを含んでい
なければ，どの DEFINE セクション　データを使用するかをアッセンブ
ル指示カードで指定する必要がある。
　アッセンブル結果の RMF 内のモジュールは，ゼロ番地を開始アドレス
としたリロケータブル形式で，ジンボリックなソースイメージも同時に
含まれている。
　アッセンブラは各マイクロインストラクション毎に，リロケータブル
アドレスの割当，シンボリックなマイクロコマンドのビットパターンへ
の変換，リンケージシンボル以外のブランチアドレスの解決を行う。他
のモジュール内を参照するリンケージシンボルは，次のステップのリン
カーで処理される。
　アッセンブルの結果は，ソースイメージとビットパターンを含むアッ
センブル　リストとして編集出力される。レーベル，マイクロコマンド
およびマクロコマンドのクロスリファレンスリストを得ることが出来る。
　アッセンブラより出力されたトランザクション RMF によって，古い
RMF の内容はアップデイトされ，新しい RMF が作り出される。このアッ
プデイトは論理的なモジュールの単位で行われ，モジュール内のマイク
ロインストラクション毎には行えない．

3.3　リンク

　リンカーは1個の完全なマイクロプログラムのバイナリーイメージを
作り上げるために RMF 内の 1～複数個のモジュールを集め，指定された
絶対アドレスにモジュールを配置し，各マイクロインストラクションに
絶対アドレスを割当，リンケージシンボルを解決する。リンケージシン
ボルはモジュール間で制御を移したり，他のモジュール内のデータを参
照したりする場合に使用される特殊なレーベルである。
　最後にパリティビットの計算が行われパターンマスターファイル
（PMF）内にバイナリーイメージを作り出す。そして各モジュールが絶対
アドレスで何番地に配置されたか，およびリンケージシンボルとその絶
対アドレスとの対応を示すリンク　マップが出力される。
　PMF 内のバイナリーイメージは，他のプログラムによって，変更カー
ドにより直接アップデイトすることが出来る。またこのプログラムは全
て人手で作られたバイナリーイメージを PMF 内に格納することが出来る。

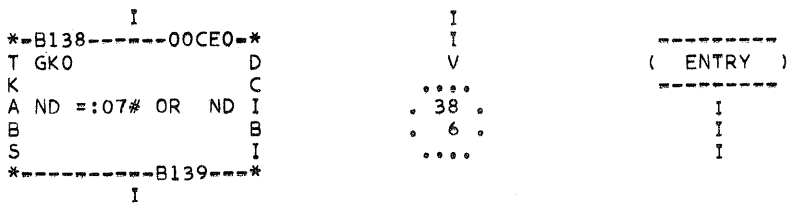
4 フローチャートの自動発生

フローチャータはRMFを入力として，自動的にマイクロプログラムのフローチャートを発生し，一般のラインプリンタへ編集出力する。FIG.3は自動発生したフローチャートの例を示す。

1個のマイクロインストラクションは，フローチャート シンボルの1個のボックスで表わされる。ソースカードの内容，ワードのアドレス，ブランチアドレスがボックスの内または周囲に，コメントはボックスの右側に置かれる。フローチャートのフォーマットは，フローチャータのパラメータカードで指定することが出来る。

4.1 フローチャート シンボル

プロセス，コネクタおよびターミナルを表わす3種類のシンボルが使用される。FIG.2にこれらシンボルの例を示す。



(a) (b) (c)

FIG. 2 FLOWCHART SYMBOLS

プロセスはボックスで表わされる。(FIG.2 (a)) このシンボルは1個のマイクロインストラクション ワードの全機能を表わすことが出来る。ワードにはいっているレーベルは左上部に，そのアドレスは右上部に，次のサイクルで実行されるワードのレーベルが右下部に，マイクロコマンドは内部に置かれる。マイクロコマンドの配置は各マイクロインストラクションのタイプ毎に指定することが出来る。ボックスの右または左端はマイクロコマンドの標識を示す1文字のアルファベットを置くことが出来る。

コネクタはサークルで表わされる。(FIG.2 (b)) このシンボルはページ間，またはページ内のラインで結べない個所を接続するために使用され，サークル内の上部にページ番号，下部にページ内のコネクタの標識を示す番号が記入される。

ターミナル(FIG.2 (c))はフローまたはサブルーチンの開始点，または終了点，を表わすのに用いられる。

4.2 フローチャートの形式

フローチャートは縦形で，フローは原則として，上から下へ向う。フローチャートの1ページは縦にいくつかのゾーン── ボックス ゾーン，コメント ゾーン，コネクタ ゾーン，およびライン ゾーン ──に区分され，フローチャート シンボルは，ボックス ゾーンに置かれる。1ページのサイズ，ゾーンの数，ゾーンのサイズ，およびボックス

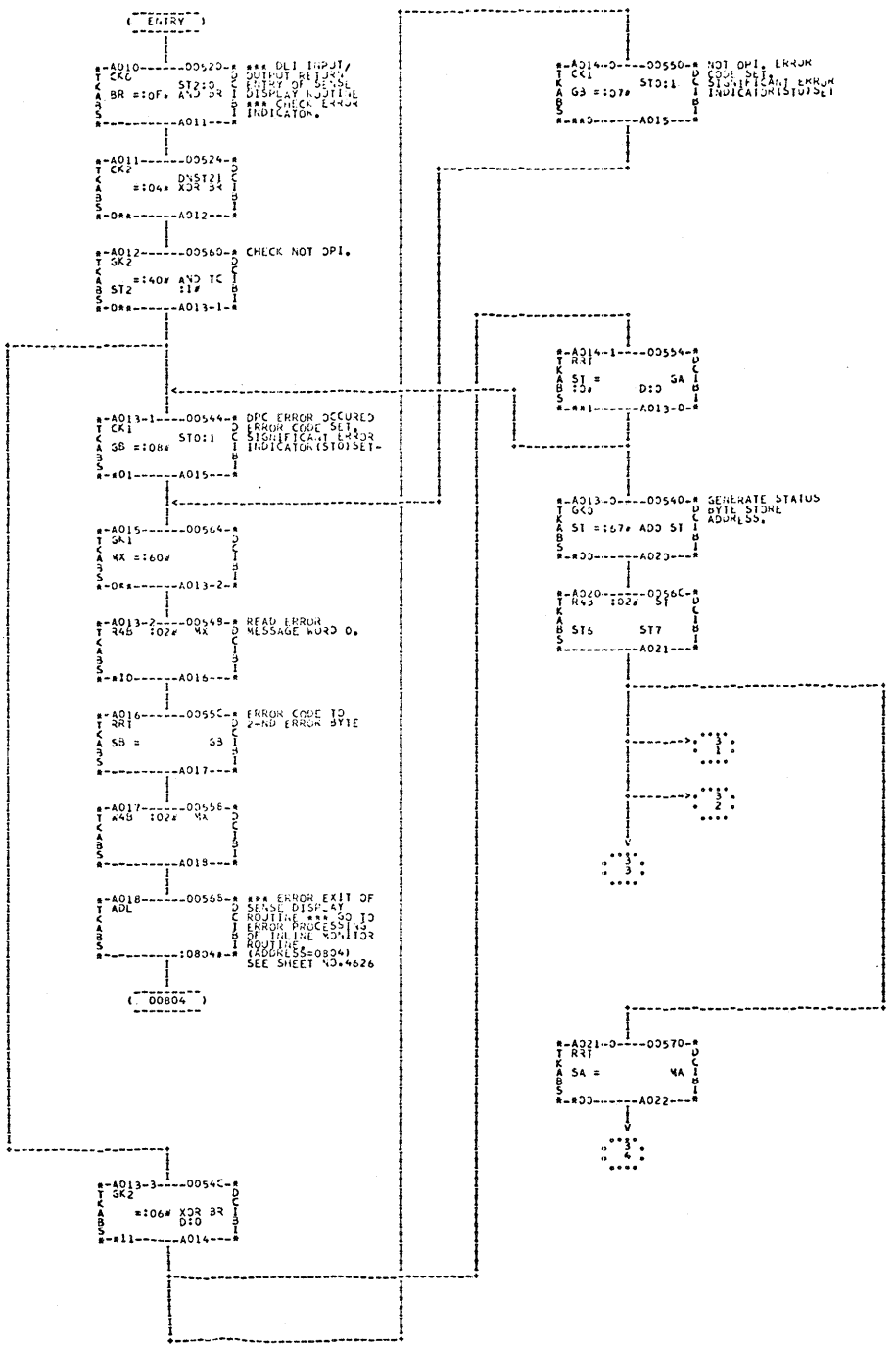


FIG. 3 A SAMPLE FLOWCHART

のサイズは，パラメータ　カードで指定することも出来る。

4.3　フローチャートの発生

フローチャートの発生は以下の２ステップで行われる。始めのステップでは，RMF からフローチャートの発生に必要なデータが抽出され，各マイクロインストラクション毎にフローチャートの要素に変換される。それから各要素が必要とするスペースが計算され，もし外部よりの指定がなければ，各シンボルのロケーションがアドレス順に割当られる。この処理の結果が中間ファイルに蓄えられる。

次のステップでシンボル間のラインの発生が，中間ファイルの内容を使用して，経験的発見法により行われ，ラインの配置の最適化が行われる。１ページ分のイメージが作られると，標準ラインプリンタの２ページにプリントアウトされる。尚アドレスとシンボルの位置に対するクロスリファレンス　リストが，同時に得られる。

5　ポストプロセス

ポストプロセスのプログラムは，アッセンブル，リンクされた PMF 内のバイナリーイメージを種々の媒体に変換出力する。バイナリーイメージは最終的には ROM　または RWM　に格納される。

5.1　PROM　パターン　データ

PROM　はマイクロプログラムの ROM　として，しばしば使用される様になった。PROM 用のパターン　データは PMF　から，紙テープへパンチされる。この処理において PROM IC と PROM カードの版数が管理されて居り，PROM パターン　データが変更されると，PROM IC の版数は１だけ増され，PROM カードの版数も更新される。

5.2　ローダ用　データ

リードライト　メモリへ，バイナリーイメージを格納するために数種類のローダがあり，これらのローダに対するデータとして，紙テープ，カード，カセット磁気テープ等へ指定のフォーマットで出力する。

6　あとがき

アッセンブル，リンク，ファイルメンテナンス，フローチャートの自動発生，およびポストプロセッシングの機能をもった，汎用のマイクロプログラミング設計補助システムが充分実用性があることが確かめられた。このシステムの主な利点は，汎用性をもつマイクロプログラムの記述法とモジュールプログラミングの機能である。

システムは NEAC シリーズ 2200 モデル 500 上で動作する様に実現され，種々のマイクロプログラムの設計に使用されている。

謝辞

日頃御指導いただく北村部長，およびシステムの実現に協力していただいた方々に深謝します。