マクロ・アセンブラ・ジェネレータの開発†

渡辺道生†† 加藤正道††† 中田育男†††

マクロ・アセンブラ・ジェネレータ (MAG) は，対象とする計算機の命令語の形式およびそれに対するアセンブラの言語仕様をマクロ定義の形で与えることにより，(マクロ)アセンブラとして働くマクロプロセッサである．MAG は，アセンブラを定義するのに便利なマクロ機能を持っており，そのマクロ展開は，通常のマクロ機能が行うような文字列の置換え処理だけでなく，マクロ参照が2進データに置き換わるまで，マクロ展開を続行するところに特徴がある，したがって，定義されたアセンブラ言語で記述されたユーザプログラム全体を，マクロ参照の列と考え，それらを，上述のマクロ展開の機構により，2進データ，すなわち，機械コードに変換することができる．また，ラベルの定義および参照についても，それぞれ，マクロ定義およびマクロ参照として扱う．MAG によって，種々のマイクロコンピュータやマイクロプログラム方式のコンピュータなど広範囲のアセンブラを簡単に作成することが可能である．

本論文では，MAG の機能およびその実現方法について，基本的な考え方を述べる．

1. はじめに

一般に，アセンブラは，命令語等が計算機の機械語に一対一に対応していて，計算機に依存する部分が多いため，対象とする計算機ごとに作られることが多いが，対象とする計算機が多い場合，各計算機ごとに別別のアセンブラを作るとすると，非常に多くの工数を必要とすることになる．

そこで，これらのアセンブラの開発を，より簡単に行う試みが，従来からいくつか行われている．それらの比較検討は，第2章で述べるが，マクロ機能の持つ拡張性に注目して，できるだけ簡単なマクロ機能で，したがって，処理プロセッサをできるだけ簡単にして，広範囲のアセンブラを作成できることを目的として，マクロ・アセンブラ・ジェネレータ（以後，MAG という）を開発した．

本稿は，次章以後を次の構成で述べる．第2章は，アセンブラの機能を定義して，各計算機に対応する個別のアセンブラを生成する，あるいは，個別のアセンブラとして働くものを総称して汎用アセンブラということにすると，既存の汎用アセンブラを概観し，その中での MAG の位置付けを明確にする．第3章は，MAG の基本的な考え方を述べる．第4章は，MAG の言語的な仕様について述べる．第5章は，MAG の仕様を実現する方法について述べる．

2. 既存の汎用アセンブラ

種々の計算機用のアセンブラを簡単に作成できるような汎用アセンブラが，これまでにいくつか開発されている．この汎用アセンブラを実現する方式は，大きく分けて，アセンブラ記述用言語を用いる方式，マクロ機能を使用する方式，パラメタによって定義する方式などがある．これらを簡単に説明すると以下のようになる．

(1) アセンブラ記述用言語を用いる方式

アセンブラ記述に際して，よく現われる処理を抽出して，アセンブラ記述に便利な言語を作成して，それを用いて，アセンブラを作成する．この方式の実例としては，METAS[1] がある．

この方式の特徴としては，作成されるアセンブラの効率は，良いが，記述言語のコンパイラを作成する必要があり，また，アセンブラ記述に際して，アセンブラの処理方式を理解する必要がある．

(2) マクロ機能を用いる方式

ソースプログラムの形式をマクロ参照の形式定義命令で定義する．そして，マクロ本体で，シンボルの置換えとシンボルから2進コードへの変換手続きを記述して，アセンブラを定義する．アセンブラは，マクロ定義の集合で構成される．この方式の実例として，XPOP[2]，METAPLAN[3]，PLASMA[4]，UNIVAS[1] などがある．前3つのシステムは，アセンブラだけでなく，コンパイラも記述することを目的としており，

† Development of the Macro Assembler Generator by MICHIO WATANABE (Narashino Works of Hitachi, Ltd.), MASAMICHI KATO, and IKUO NAKATA (Systems Development Laboratory, Hitachi, Ltd.).

†† (株)日立製作所習志野工場
††† (株)日立製作所システム開発研究所

表 1 汎用アセンブラの各方式の比較
Table 1 General comparison of meta-assemblers.

比較項目 / 方式	アセンブラ定義の簡単さ	汎用アセンブラ実現の容易さ	作成されるアセンブラの効率（メモリ，速度）	拡張性
記述言語による方式	困難（アセンブラの処理方式を理解する必要がある．）	困難（コンパイラを作る必要がある．）	良い	中程度
マクロ機能による方式	中程度（マクロの概念さえ理解すれば，わかりやすい．）	中程度（マクロプロセッサを作る必要がある．）	悪い（インタプリタ方式であるので速度が遅くなる．）	良い
パラメタ形式による方式	簡単（入出力形式，変換方法を書くだけでよい．）	困難（入出力形式に多様性があるので，プロセッサが複雑となる．）	悪い（ 同 上 ）	悪い（記述形式が固定されることが多い．）

システムが大きくなっている．UNIVAS は，机上で検討されたのみで，詳細は不明である．

（3） パラメタによって定義する方式

ソースプログラム，オブジェクトプログラムのそれぞれの形式，および，これらの間の変換規則をパラメタの形で与えるものである．この方式の実例としては，RAPID[5]，EGMA[6]，MPASS[7] などがある．これらは，マイクロ命令用アセンブラの作成を目的としている．

この方式の一般的特徴として，アセンブラ定義が簡単であるが，入出力形式に多様性があり，プロセッサが複雑となり，また，作成されるアセンブラの効率があまり良くなく，フレキシビリティに欠ける欠点がある．

なお，上記の作成されるアセンブラの効率の改善をねらって，命令形式をテーブルで記述しておく方式が PASM[8] で行われている．

上記の各方式の比較を **Table 1** に示した．

本報告の MAG は，これらの方式のうち，（2）のマクロ機能を用いる方式に属するが，MAG の焦点をアセンブラに絞り，簡単なマクロ機能で汎用アセンブラを実現する，従って，アセンブラの定義は，簡単なマクロ定義で済み，かつ，MAG のプロセッサも簡単にインプリメントできることを目標とした．

3. 基本方針

マクロ機能でアセンブラを記述する基本的な考え方を以下に述べる．

通常のマクロ機能（たとえば，大型コンピュータのアセンブラのもつマクロ機能）は，ある文字の列を他の文字の列に置換える働きをするものである．**図 1** に，その例を示すように，（1）の MOVE というマクロ名が定義されるとき，（2）のマクロ参照は，その文字列が，（3）の文字列と置換えられたのと全く同じ効果をもつ．ここで注意すべきことは，文字列の置換えまでしか行わないことである．

```
（1） マクロ定義
        MACRO
        MOVE    &OP1, &OP2
        L       R1, &OP1
        ST      R1, &OP2
        MEND
（2） マクロ参照
        MOVE    A, B
（3） 展開形
        L       R1, A
        ST      R1, B
```

図 1 通常のマクロ機能の例
Fig. 1 An example of ordinary macro facility.

これらの通常のマクロ機能を用いて，ある機種A用のアセンブリ言語で記述されたプログラムを，別の機種Bのマクロアセンブラでアセンブルすることもできる．すなわち，これは，A用のプログラムをBのマクロ機能を用いて，Bのアセンブラで処理可能な形に文字列の変換を行った後，Bのアセンブラの持つ，定数変換機能，アドレス割付け機能，オブジェクトプログラム出力機能などを用いて，アセンブル処理を遂行するものである[9]．

ただし，通常のマクロ機能では，マクロ参照する場合のマクロ名の位置や使用できる区切り記号などが，かなり限定されているため，マクロ定義で定義できる機種A用のアセンブリ言語の仕様も，かなり限定されたものになってしまう．また，機種Bのアセンブラを用いるため，AとBで，アドレッシング方式（バイトアドレッシングか，ワードアドレッシングかなど）や変換されたプログラムの出力形式が異なる場合は，処理が困難である．

以上のように，通常のマクロ機能を用いて，アセンブラを作る場合，

（1） 仕様を自由に設定することができない．

（2） アドレッシング方式や出力形式が異なる場合

は，困難である．

など，ある限られた範囲にしか適用することができない．

一方，前述の XPOP, METAPLAN, PLASMA など既開発のマクロ機能を用いた汎用アセンブラでは，上に述べたような適用範囲の制限はないが，逆に，コンパイラ記述ををも目的としており，大げさになっているきらいがある．また，METAPLAN を例にとると，アセンブラの処理のうち，オペランドの処理の部分にマクロ機能を用いているのみで，全面的にマクロ機能を使っているわけではない．

MAG では，

(1) 定義されるアセンブラの仕様を自由に設定できる．

(2) アドレッシングや出力形式も自由に定義できる．

(3) シンプルな仕様とし，MAG プロセッサを簡単にインプリメントできる．

などを目的として，以下の考え方で，アセンブラ処理に対して，全面的にマクロ機能を使うことにした．

まず，MAG のマクロ機能では，通常のマクロ機能が，文字列から文字列への置換えであるのに対し，文字列から，2進データまでの置換えを行う．すなわち，2進への変換処理をも，マクロ機能の中で行い，アセンブラの2進への変換処理をマクロ展開の延長として行う．もう少し詳しく説明すると，MAG においては，文字列から文字列への置換えは，通常のマクロ機能と同様に，マクロ定義されている文字列との置換えとして行われる．MAG のマクロ機能が通常のマクロ機能と異なるのは，これ以後であり，文字列が2進データに置換えられるまでマクロ展開を続行する点である．ここで，2進データに置換えられるとは，

(1) 該当する文字列に対して，定数など数値データとして，マクロ定義されている場合．

(2) 該当する文字列が，ソースプログラム中のラベルとして用いられていることによって，その文字列がアドレスと結びつけられている場合．

(3) その他，パラメタなどを経由して，該当する文字列に対して，数値データが結びつけられている場合．

のいずれかをいう．

以下に，MAG において，マクロ機能を用いて，ある仮想的な計算機のプログラムをアセンブルする例を示す．(図2参照)

(1) 仮想計算機の仕様の1部

(a) Load 命令は，Type 1 形式の命令に属し，オペレションコードは，20 とする．

(b) Type 1 形式の機械命令の形式は，次の通りとする．

オペレーションコード	アドレス
← 8ビット →	←―― 16ビット ――→

(c) Load 命令のアセンブラの記述形式は，

L/オペランド;

とする．

(2) マクロ定義

```
&DEF L/&AS TYPE 1,20, &END                    ……①
&DEF TYPE 1, &P1, &P2; &AS &P1! &P2!
  &OBJECT (8,16) &END                         ……②
```

(3) ユーザソースプログラム

```
        ⋮
    L/ABC;
        ⋮
ABC: ……
```

図 2 仮想的計算機によるMAGのアセンブル処理例
Fig. 2 A part of specification of a computer and its assembler.

仮想計算機の仕様の1部は，(1)の通りとする．(2)にマクロ定義を示した．ここで，"&DEF α &AS β &END" は，α を β と定義する，すなわち，α が現われたら，それを，β で置換えることを意味する．(3)のユーザソースプログラムが現われたとする．

(a) "L/ABC;" の "L/" で，①のマクロ参照が起る．"L/" は，"TYPE 1,20" で置換えられる．

(b) 入力テキストは，"TYPE 1,20, ABC;" となり，"TYPE 1," で，②のマクロ参照が起る．パラメタマッチングで，

&P1 は，20
&P2 は，ABC

となる．

(c) (b)の入力テキストは，

"&P1 ! &P2 ! &OBJECT (8,16)"

で置換えられる．

(d) &P1 は，20 という数値データであるので，これ以上，マクロ参照は起らず，20 の値がスタックされる．

(e) 次に P2 をスキャンすると，"ABC" であるが，ABC は，また未定義であり，ユーザラベルとして処理し，0という値を仮定して，スタックする．

(f) &OBJECT は，MAG の組込み関数で，スタックにあるものを，この場合は，8ビットおよび 16ビットとして，各々出力することを意味する．これで，オペレーションコードが 20，アドレスが 0 という Load 命令の機械命令が出力される．

(g) 次に，"ABC:" というラベルが現われると，

ABC は，前に，未定義のラベルとして現われ，テーブルに登録されているので，その時点のプログラムカウンタを，ABC のテーブルに入れると同時に，前に ABC が使用され，出力されている Load 命令のアドレス部に入れる．

以上で，Load 命令のオブジェクトプログラムが完成する．

4. 仕　様

4.1 マクロ定義とマクロ参照

MAG におけるマクロ定義は，

&DEF $\alpha[\beta_{i\cdots}]$ &AS マクロ本体 &END

の形式である．ここで α および β_i はトークンでそれぞれマクロ名およびパラメタを表わす．ただし，トークンとは，英字 (A, B, C, ⋯, Z) を先頭とする英字と数字 (0, 1, ⋯, 9) の列（英字名），数字の列（数），&を先頭とする英字と数字の列(キーワード)，あるいは英字，数字，&，空白を除く1文字（特殊文字）のいずれかである．各々の列は，英字あるいは数字以外の文字で区切られる．また，&を先頭とするトークンは MAG で使用するキーワードで他の目的には使用できない．（以後，構文の定義に，AN 記法[10]を用いる．）

ここで，マクロ名としての α は，通常のマクロ機能では，英字名だけであるが，＋や－の演算や各種の区切り記号に対しても，その処理内容をマクロ定義できるよう，英字名の他に，区切り記号などの特殊文字を許すようにした．パラメタ β_i $(i=0, 1, \cdots, n)$ も同じく，マクロ参照に付随する情報の区切りをシステム固定のものでなく，自由に選択できるよう，英字名，数，特殊文字の各トークンおよび &Pn の形のキーワードとした．ただし，&Pn $(n=1, 2, \cdots, k)$ の形のパラメタは，マクロ参照の実パラメタの文字列をマクロ本体中に引渡すための変数（仮パラメタ）とした．その他の形のパラメタは，マクロ参照とマクロ定義のマッチング，&Pn の形のパラメタに対する実パラメタの切出しに用いられる．（マッチングの詳細は，後述する．）

マクロ定義の意味は，ユーザのテキスト中に，

$\alpha[\beta_{i\cdots}]$

と一致する文字列が現われた場合，マクロ本体と置換えることを定義するものである．

マクロ参照には，特定の形式がなく，ユーザのテキスト中の文字列は，一般にすべて，マクロ参照と考える．これによって，通常のマクロ機能のように，特定のフィールドにマクロ名が書いてないとマクロ参照とならないということはなく，自由にマクロ参照することができる．

マクロ定義とマクロ参照の結びつきは，以下のようにして行われる．今，マクロ定義が，

&DEF $\alpha\beta_0\beta_1\cdots\beta_n$ &AS⋯&END

入力テキストが，

$t_1t_2\cdots t_m$

であるとする．ここで，α, β_i $(i=0, 1, \cdots, n)$，t_j $(j=1, 2, \cdots, m)$ は，すべて，トークンであるとする．

まず，t_1 をマクロ参照とみなし，以下の処理を行う．

(1) $t_1=\alpha$ となるマクロ名 α がマクロ定義されている場合

(a) $\beta_0\beta_1\cdots\beta_n$ が空，すなわち，α のマクロ定義に対して，パラメタがない場合，t_1 は，マクロ定義 α をマクロ参照しているものとし，t_1 をマクロ本体で置換える．

(b) β_0=&Pn あるいは，$t_2=\beta_0$ のとき，t_1 は，そのマクロ定義に対するマクロ参照とする．残りのパラメタのマッチングについては，後述する．

(c) $\beta_0\neq$&Pn かつ $t_2\neq\beta_0$ のとき，$t_1=\alpha$，$t_2=\beta_0$ となる他のマクロ定義を探す．

(2) $t_1=\alpha$ となるマクロ定義がない場合

t_1 は，ユーザ定義の名前とし，テーブルに登録する．

以上が，マクロ定義とマクロ参照の結びつきの処理であるが，ここで注意すべき点は，第1に，(1)(b)および(c)に示すように，マクロ参照 t_1 が α とだけの一致をとるのではなく，第1パラメタ β_0 と t_2 との一致（すなわち，区切りの一致）をも必要とすることである（ただし，パラメタがない場合および β_0 が &Pn のときは除く）．これによって，マクロ名 α が同じであっても，第1パラメタ β_0 が異なれば，別のマクロ定義とすることができ，マクロ定義の多様性に対処できる．例えば，アセンブラのニモニック記号を予約語とせず，同じ名前をラベルとして使える方が，一般的には，望ましいが，ニモニック記号のうしろに特定の区切り記号をつけるようなマクロ定義をすることによって，ニモニック記号と同一の名前をラベルとして使えるアセンブラを定義することができる．

注意すべき，第2点は，(2)に示すように，マクロ定義されていないトークンがテキスト中に現われた場合，その名前は，ラベルなどユーザ定義の名前であり，ラベル定義やラベル参照もマクロ定義，参照の一種と

して扱えるようにしたことである．

次に，マクロ定義とマクロ参照の結びつきが行われた後のパラメタマッチングについて説明する．あるマクロ定義に対するマクロ参照であるとき，そのマクロ参照は，マクロ本体と置換えられる．ただし，そのマクロ定義にパラメタがあるときは，パラメタと入力テキストのマッチングが行われる．マッチングの方法は，以下の通りである．

入力テキストを，$t_2t_3\cdots t_j\cdots$

パラメタを，$\beta_0\beta_1\cdots\beta_i\cdots\beta_n$

とする．t_j, β_i は，ともにトークンを表わす．今，既に，t_{j-1} と β_{i-1} までマッチングが行われたとすると，

(1) β_i が &Pk ($k=1,2,\cdots$) のとき，

$i=n$ のとき，および β_{i+1} が &Pk+1 のときは，&Pk $=t_j$ とする．それ以外のときは，$\beta_{i+1}=t_{j+l}$ ($l=1,2,\cdots$) となる t_{j+l} をさがし，&P$k=t_jt_{j+1}\cdots t_{j+l-1}$ とする．

(2) β_i が &Pk でないとき，$\beta_i=t_j$ となるか，チェックし，等しくなければ，エラーとする．

以上のパラメタマッチングの処理からわかるように，パラメタとしては，自由な区切りを選択することができ，通常のマクロ機能のように，“,”でなければならないということはなく，英字名でも，特殊文字でもよく，また．いくつつながっていてもよい．また，&Pn は，通常のマクロ機能におけるパラメタと同様に，マクロ本体の中で参照することができ，さらに，&Pn に割当てられている文字列は，他のパラメタと矛盾がなければ，“(”,”)” や “,” など，どのような文字を含んでいてもよいという自由度がある．

次に，マクロ定義とマクロ参照のメカニズムを例で説明する．

次のようなマクロ定義があったとする．

```
&DEF L,  &AS…&END                    ……①
&DEF L/ &P1,  &P2; &AS…&END
                                     ……②
```

一方，マクロ参照のトークン列を

```
L/R1,  ABC+2;
```

とすると，“L”をマクロ名とするマクロ定義は，①と②の2つあるが，①の第1パラメタ β_0 が，“,”で，入力テキストの “/” と一致しないので，①のマクロ定義を参照しているのではないことになる．一方，②の定義は，第1パラメタが，“/”で入力テキストの “/” と一致し，②のマクロ定義を参照していることになる．次に，パラメタマッチング処理を行い，“,”と“;”で一致をとり，

```
&P1 は，“R1”
&P2 は，“ABC+2”
```

をそれぞれ割当てられる．また，②のマクロ定義のパラメタ “,” や “;”は，特殊文字でなくてもよい．たとえば，②の定義が，

```
&DEF LOAD &P1 FROM &P2; &AS,
  …, &END
```

とすると，入力テキストが，

```
LOAD R1 FROM ABC+2;
```

であれば，前述と同じ結果となる．

4.2 マクロ本体

マクロ定義とマクロ参照の結合が確立すると，入力テキスト中のマクロ参照は，それに付随するパラメタも含めて，対応するマクロ定義のマクロ本体と置換えられる．マクロ本体として，置換えるべき文字列や数値データを書くことができるのは，当然であるが，その他に，アセンブラの定義が簡単にできるよう，種々の条件によって，マクロ展開の方法を変更できる手段を用意した．

マクロ本体の記述形式は，マクロ処理が，一般的にいって，文字列の置換えであり，その結果として，何らかの値を持つので，次のように，式形式とし，マクロ本体は，式をいくつか並べたもの，すなわち，

〈マクロ本体〉=〈式〉{!}$_{000}$

とした．ここで，“!”は，式の区切りを示すものである．各式の値がそれ以上マクロ展開を必要としない場合（すなわち，1つの数のみからなる場合）は，その値をスタックし，それ以外の場合はそれを入力テキストとして，さらに，マクロ展開を行う．

マクロ本体の各式は，図3に示すもののいずれかである．また，マクロ本体の各式の中には，文字列の操作，スタック操作，テーブル操作などが，簡単にできるよう，各種のオペレータや組込み関数を用意した．

マクロ本体中で，扱うことのできるデータ構造には，ユーザ変数，システム変数（MAG の中で，名前とその意味が決っているもの），パラメタ変数，スタック，テーブル（マクロ名およびユーザの使用する名

```
テキスト式: テキスト（文字列）の連結など
算 術 式: 算術値の加減算
代 入 式: 算術値やテキストの変数への代入
条 件 式: 条件により式を選択
分 岐 式: 無条件に分岐
終 了 式: 1つのマクロ定義に関するマクロ展開を終了する．
```

図3 マクロ本体記述のための式
Fig. 3 Expressions in macro body.

前が格納される）があり，ユーザは，かなりの自由度をもって，使用することができる．

5. インプリメンテーション

MAG のインプリメンテーションに当っては，工数や開発期間などの関係から，できるだけ簡単にインプリメントできる方法をとることにした．

まず，全体の処理方式としては，通常のマクロ処理と同じように，解釈実行方式とし，通常のアセンブラと同じように2パス方式とした．ただし，ここで，2パス方式といっても，プロセッサを簡単にするため，また，パス 1，パス2ともソースプログラムから入力することにすれば，処理内容は，大体，同じにすることができるため，パス1とパス2の処理ルーチンは，同じものを使い，ルーチンの内部のスイッチで，パス1とパス2の処理の違いを切り分けた．パス1とパス2の処理の違いは，パス1で名前に対する値を決定し，パス2で，その値を使ってオブジェクトプログラムを出力することである．マクロ展開の処理は，パス 1，パス2共に，行うことになる．

MAG の処理の概要は，図4に示す通りである．マクロ定義の部分は，後のマクロ展開が簡単になるよう中間言語に変換し，ソースプログラムは，2回入力し，したがって，マクロ展開は，2回行うことになる．

マクロ展開部分の処理手順の概略は，以下の通りである．(図5参照)

（1） 入力テキストから1トークン取出す．

（2） 取出したトークンをもとに，マクロ定義が登録されているテーブルをサーチする．

（3） そのテーブル中の対応するエントリより，マクロ定義の内容をとり出す．

（4） パラメタマッチングを行い，パラメタ変数にトークン列を割付ける．次いで，マクロ展開用スタックに，1つ前のマクロ展開用スタックのスタックポインタ，直前のテキスト・スキャン・ポインタの値を退避する．

（5） テキストスキャンポインタにマクロ本体の先頭アドレスをセットして，マクロ展開を開始する．

（6） マクロ本体のマクロ展開が進み，そのマクロ本体の処理が完了すると，マクロ展開用スタックからスタックポインタおよびテキスト・スキャンポインタの値が1つ浅いレベルのものに置換えられる．そして，(1)の処理にもどる．

MAG のインプリメントには，トランスレータ記述

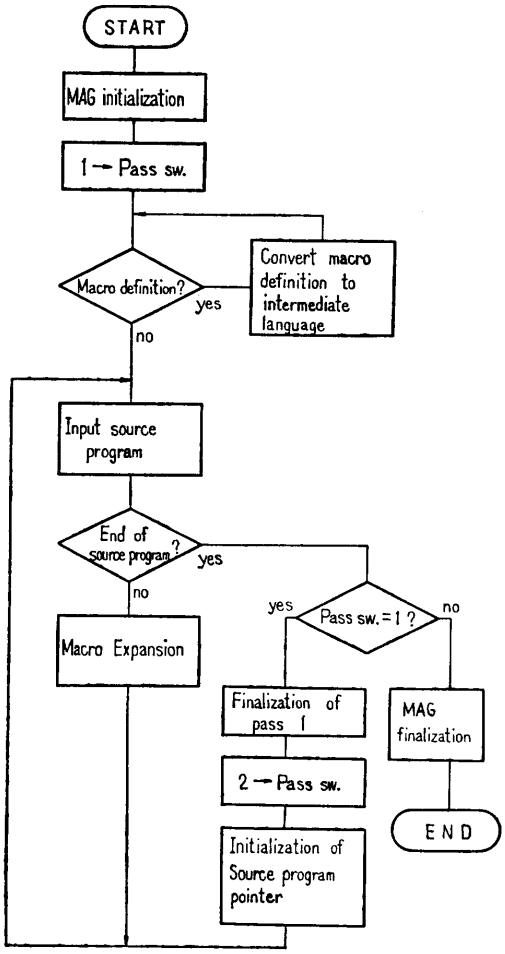

図 4 MAG の処理概要
Fig. 4 General flowchart of MAG.

用言語 UTS[11] を用い，約5,000行の大きさである．所要メモリは，作業領域などを含めて，約 80k バイトである．

MAG を用いて，主にミニコンやマクロコンピュータ（たとえば，ミニコン HITAC 10，マイクロコンピュータ HMCS-4 (Intel 4004 相当品)，HMCS-6800[13] など）の種々のアセンブラを定義してみたが，大体 120～450 行で記述でき，各アセンブラは，2～4週間で完成できた（付録参照）．作成されたアセンブラの処理速度は，HITAC 8450 を用いた場合，約30枚／分である．この速度は，あまり速くないが，その理由は，マクロ展開が，解釈実行で行われること，文字列の置換えを必要とすることなどから，本質的に時間のかかる，ということもあるが，MAG では，さらに，インプリメントを簡単にするために，

（1） 前にも述べたように UTS を使ったが，UTS

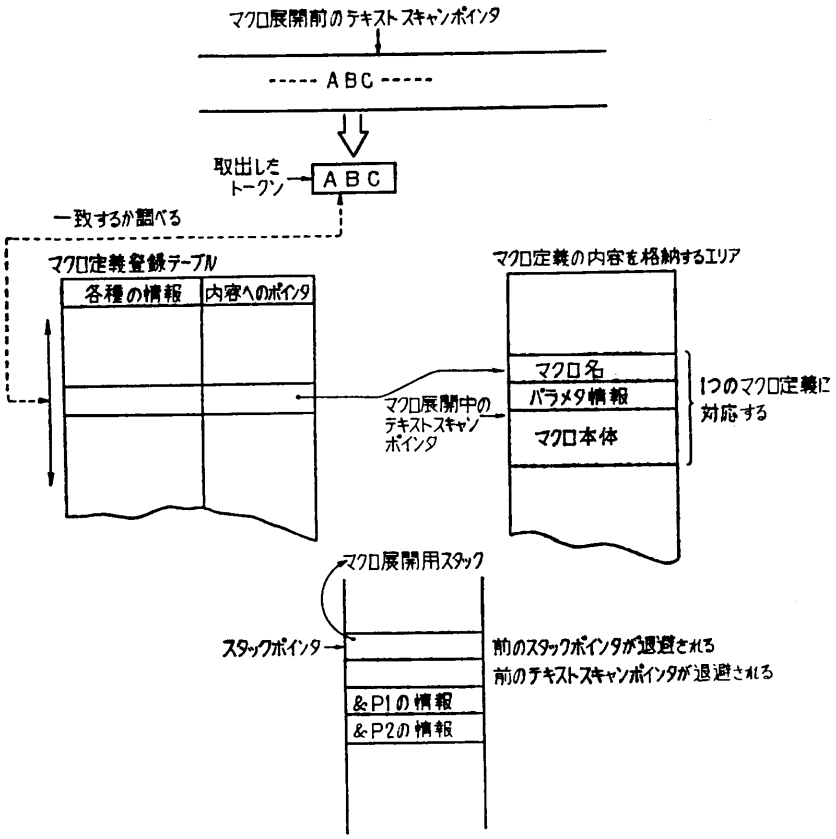

図 5　マクロ展開の処理手順
Fig. 5　Process of macro expansion.

が解釈実行型であること

（2） 同じプログラムを2つのパスに使っているため，重複した処理が行われる（たとえば，マクロ展開が2回行われる）こと

などが，さらに処理速度を低下させていると思われる．これらのインプリメント上の2項目を改良することによって，処理速度は，100枚／分以上に向上できる可能性がある．

また，MAG を使ってアセンブラを作成する場合，ソースプログラムからオブジェクトプログラムへの変換の処理だけでなく，エラーチェックやエラーの後処理等の機能も実用的なシステムとするために必須である．本稿では，紙面の制約上から記述できなかったが，MAG にはエラーチェックやエラー後処理の内容もユーザが記述できる機能を持っていることを付記しておく．

6. ま と め

マクロ機能を用いて，アセンブラの入力形式，出力形式，その間の変換方法を定義して入力すれば，マクロ機能をもつアセンブラを作成できるという，マクロアセンブラジェネレータを開発し，その考え方等について述べた．

本開発のマクロアセンブラジェネレータは，以下の特徴を持っている．

（1） 簡単にアセンブラを定義できる．

（2） 作成されたアセンブラは，マクロアセンブラであり，ユーザは，さらにマクロ定義を追加して，自分向きにして使うことができる．

（3） 広範囲のアセンブラを定義することができる．

（4） マクロアセンブラジェネレータのプロセッサを小さく，簡単にインプリメントすることができる．

マクロアセンラジェネレータは，アセンブラのように，ソースプログラムとオブジェクトプログラムが1対1に対応する場合で，使用頻度がそれほど高くなく，アセンブラの速度をきびしく要求されない場合に広く適用することができる．特にアセンブラ等を，いくつかの機種について，短期間に作成する必要があるとき，それらのアセンブラ定義さえ入力すれば，アセンブラが生成されるので，有効である．特にマイクロコンピュータやマイクロプログラム方式の処理装置用のアセンブラとして，また，ROM (Read Only Memory) や PLA (Programmable Logic Array) などのビットパターン生成を，わかりやすく，効率良く実現するための道具として使用できると思われる．

今後の方向としては，第1に，マクロアセンブラの処理速度の向上など性能面および，各種の機能面の改良をはかって行く必要があると考えている．第2に，マクロアセンブラジェネレータは，プログラムの生成を行うが，生成されたプログラムのデバッグ等を行う，汎用シミュレータの検討も必要である．

終りに，MAG について討論していただいた立教大

学島内助教授，MAGプロセッサの作成をしていただいた(株)日立ソフトウェア・エンジニアリング，木原康博氏，諸富三千男氏および(株)日立システム開発研究所鶴田節夫氏，大沢恒春氏に深謝します．

参考文献

1) Y. Nitta, A. Nozaki and T. Uehara: On an Efficient Assembler Building System—METAS Meta Assembler, 1st USA—JAPAN Computer Conference, pp. 442-447 (1972).
2) H. I. Halpern: XPOP—A Meta Language Without Meta Physics, Proc. FJCC, Vol. 26, pp. 57-68 (1964).
3) D. E. Ferguson: The Evolution of the Meta-assembly Program, Commun. ACM, Vol. 9, No. 3, pp. 190-196 (1966).
4) M. L. Graham and P. Z. Ingerman: An Assembly Language for Reprogramming, Commun, ACM, Vol. 8, No. 12, pp. 769-773 (1965).
5) RAPID, a configuration independent symbolic assembler for read only store, SMS社 (1971).
6) 飯塚 肇他：モジュール型複合計算機ACE, 情報処理学会計算機アーキテクチャ研資 74-4(1974).
7) 星 元雄他：マイクロプログラムアセンブラの一構成法，情報処理学会設計自動化研資 75-24 (1975).
8) 宮口庄司他：マイクロプロセッサ汎用アセンブラの一構成法，電子通信学会，信学技報 Vol. 76, No. 10, pp. 21-30 (1976).
9) 大原憲治：異機種アセンブラの実現形態，情報処理学会第15回大会 (1975).
10) 和田英一：ALGOL N，情報処理，Vol. 12, No. 9, pp. 556-567 (1971).
11) 野木兼六他：UTS による言語プロセッサの開発，情報処理学会第16回大会 (1975).
12) 加藤正道他：MAG (Macro Assembler Generator) の開発，情報処理学会第16回大会 (1975).
13) HMCS 6800 ユーザーズマニュアル：日立製作所 (1976).

付 録

4ビットマイクロプロセッサ HMCS-4 (Intel 4004 相当品) のアセンブラ定義例の1部(全部で約120行)

```
        EBCDIK    CODE  CARD  LIST
&DEF NOP/&AS ATYPE,&X00 &END
&DEF WRM/&AS ATYPE,&XE0 &END
&DEF WMP/&AS ATYPE,&XE1 &END
&DEF WRR/&AS ATYPE,&XE2 &END
&DEF WPM/&AS ATYPE,&XE3 &END
&DEF WR0/&AS ATYPE,&XE4 &END
        :
&DEF SUB/&AS FTYPE,&X90 &END
&DEF LOD/&AS FTYPE,&XA0 &END
&DEF XCH/&AS FTYPE,&XB0 &END
&DEF ISZ/&AS GTYPE,&X70 &END
&DEF BBL/&AS HTYPE,&XC0 &END
&DEF LDM/&AS HTYPE,&XD0 &END
&DEF ATYPE,&P1 ;&AS &P1! &OBJECT(8)                                    &END
&DEF BTYPE,&P1 &P2,&P3;&AS &G1 &= &P1 ! &H1 &= &P2 ! &G2 &= &P3 !
     &IF &H1 &EQ &'D'     &THEN &GOTO &L1 &EIF!
     &IF &H1 &EQ &'T'     &THEN &G1 &= &G1 &+ &X1! &GOTO &L1 &EIF!
     &IF &H1 &EQ &'C'     &THEN &G1 &= &G1 &+ &X2! &GOTO &L1 &EIF!
     &IF &H1 &EQ &'TC'    &THEN &G1 &= &G1 &+ &X3! &GOTO &L1 &EIF!
        :
     &IF &H1 &EQ &'NCA'   &THEN &G1 &= &G1 &+ &XE ! &GOTO &L1 &EIF!
     &IF &H1 &EQ &'NTCA'  &THEN &G1 &= &G1 &+ &XF ! &GOTO &L1 &EIF!
     &PRINT(&'***ERR C ***')!
 &L1 &G1! &IF &G2 &LE &XFF &THEN &GUTO &L2 &EIF!
     &PRINT(&'***ERR A ***')! &G2 &= 0 !
 &L2 &G2! &OBJECT(8,8)                                                 &END
&DEF CTYPE;&P1 &P2,&P3;&AS &G1 &= &P1 ! &G2 &= &P2 ! &G3 &= &P3 !
     &IF &G2 &LE 7 &THEN &GOTO &L1 &EIF!
     &PRINT(&'***ERR R ***')! &G2 &= 0 !
 &L1 &IF &G3 &LE &XFF &THEN &GOTO &L2 &EIF!
     &PRINT(&'***ERR D ***')! &G3 &= 0 !
 &L2 &G1 &+ &G2 &* 2 ! &G3 ! &OBJECT(8,8)                              &END
        :
&DEF HTYPE,&P1 &P2;&AS &G1 &= &P1 ! &G2 &= &P2 !
     &IF &G2 &LE &XF &THEN &GOTO &L1 &EIF!
     &PRINT(&'***ERR D ***')! &G2 &= 0 !
 &L1 &G1 &+ &G2 ! &OBJECT(8)                                           &END
&DEF +&P1 &AS &P1 &CAT &' ' ! &POP(&G4) ! &POP(&G5) ! &G5 &+ &G4      &END
&DEF -&P1 &AS &P1 &CAT &' ' ! &POP(&G4) ! &POP(&G5) ! &G5 &- &G4      &END
&DEF *&P1 &AS &P1 &CAT &' ' ! &POP(&G4) ! &POP(&G5) ! &G5 &* &G4      &END
&DEF %&P1 &AS &'&X' &CAT &P1                                          &END
&DEF :&AS &VAL(&TBCP) &= &PC ! &ATR(&TBCP) &= &ATR(&TBCP) &OR 8       &END
&DEF DSA/&P1;&AS &G7 &= &P1 !
          &L1 0 ! &OBJECT(4) ! &G7 &= &G7 - 1 !
              &IF &G7 &NE 0 &THEN &GOTO &L1 &EIF                      &END
&DEF DSC/&P1;&AS &P1 ! &OBJECT(4)                                     &END
&DEF DDC/&P1;&AS &P1 ! &OBJECT(8)                                     &END
&DEF DHC/'&P1';&AS &H2 &= &P1 ! &G6 &= &LENGTH(&H2) ! &G7 &= 0 !
          &L1 &SUBBIT(&CHAR(&SUBTXT(&H2,&G7,1)),4,4) !
              &OBJECT(4) ! &G7 &= &G7 + 1 !
              &IF &G7 &LT &G6 &THEN &GOTO &L1 &EIF                    &END
&DEF EQU/&P1=&P2;&AS &P2 ! &POP(&G6) ! &H2 &= &P1 ! &G7 &= &SRCH(&H2)!
              &VAL(&G7) &= &G6 ! &ATR(&G7) &= &ATR(&G7) &OR 8         &END
&DEF ORG/&P1;&AS END/ ! &PUT(0,32) ! &PUT(0,32) ! &PC &= &P1 !
              &PUT(&PC,16)                                            &END
&DEF END/&AS &PUT(0,0) ! &PUT(0,32) ! &PUT(0,32)                      &END
```

(昭和51年9月30日受付)
(昭和53年5月30日採録)