Vol. 19 No. 7　　情　報　処　理　　July 1978

論　文

超言語記述によるマイクロプロセッサ用汎用クロス・アセンブラ*

国立　勉**　上垣俊二**
藤田晨二**　吉田　清**

Abstract

Numbers of new microprocessor (μP) systems are being sold. Therefore, it is necessary to produce assemblers for these many μPs easily and quickly.

We have developed a general cross-assembler (PMP general purpose assembler) which has a meta-language MASP. A MASP program doesn't define a basic assembly algorithm, but only a μP's instruction set in tabular form. Differing from other assembler meta-languages, MASP can define many μP's assemblers easily. In the case of making an assembler for 8080 for example, this method depending on MASP definition, showed 60 times as high productivity as the ordinary method of coding in a general purpose programming language.

1. まえがき

1971 年末に最初のマイクロプロセッサが出現して以来，マイクロプロセッサの応用範囲は着々と拡大し，LSI 技術の発展に伴って，安価で高機能なマイクロプロセッサ（以下 μP と略す）が次々と発表されている．

μP の応用製品の開発には，製品に応じて種々の μP を使う必要があり，それぞれに応じたサポート・ソフトウェアが必要となる．一方，サポート・ソフトウェア提供の側から見ると，種々の μP 向けにそれを迅速に提供するため，生産性の高いサポート・ソフトウェア作成手法が必要である．

サポート・ソフトウェアは μP より上位のミニコンまたは大型機上で走行させる形態（クロス・ソフトウェア）で多く使用されている．クロス・ソフトウェアのうち，クロス・アセンブラを考えるとき，μP ごとにアセンブラ言語仕様を定義することによって，種々の μP のアセンブラとすることができる汎用のアセンブラ（あるいはメタ・アセンブラ）は大変有用である．

アセンブラを汎用化して複数の計算機に適用する試みはこれまでもしばしば行われている[1)~18)]．しかしながら，これらは次のいずれかの点で問題がある．

1) 超言語*** がある程度制限された言語仕様となっているため，現在市販されている数多くの μP の多様なアセンブル機能を十分に包含しておらず，適用対象機種が制限される．
2) 超言語が変数および実行制御文（IF, GOTO など）を導入した手続型言語となっており，アセンブルアルゴリズムを記述する性格から，μP のアセンブラを容易に定義できず，多くの μP のアセンブラを作成する場合の生産性が低い．

ところでコンパイラ言語のような機械独立の言語仕様を追求しない場合，アセンブラ言語の1ステートメントは1つの機械語に対応する．1ステートメントは区切り記号によりラベル，命令，オペランドに区切られ，この構文は機械に共通である．従って，アセンブラの汎用化は命令名と命令コード，各種アドレシング・モードに対応するオペランド表記法と機械語コードの対応関係を定義すれば可能となる．なお，ソース言語の各種区切り記号，機械語のビット長など命令に依存しない基本的規定は別に定めれば良い．またアセンブラ擬似命令は機械に依存しない共通機能としてまとめることができるので定義は不要である．

---

* A General Purpose Cross-Assembler for Microprocessors based on a Meta-language description by Tsutomu KUNITACHI, Shunji UEGAKI, Shinji FUJITA, and Kiyoshi YOSHIDA (Yokosuka Electrical Communication Laboratory, N. T. T.).

** 日本電信電話公社横須賀電気通信研究所データ通信研究部

*** アセンブラ言語を定義する言語を超言語と呼ぶ．

以上のことから，基本的アセンブル処理を記述せず，ソース言語（命令名とオペランド表記法）と機械語との対応のみを定義する表形式の非手続型超言語を考えれば，大部分の μP に適用でき，かつ多数の μP のアセンブラ作成の高生産性が期待できる．この種の表形式超言語を適用したものとして汎用マイクロプログラム・アセンブラ[19)~23)]があるが，言語仕様上の相違から必ずしも機械語レベルのアセンブラには適さない部分がある．そこで，筆者らは多数の μP の機械語レベル・アセンブラとして，超言語 MASP* に基づく汎用クロス・アセンブラ（PMP** 汎用アセンブラ）を実用化した．

本論文では，**2.** で PMP 汎用アセンブラの構成について従来の方式と比較検討を行い，**3.** で MASP についてその言語仕様を述べ，**4.** で PMP 汎用アセンブラの構成を，最後に**5.** で種々の考察を述べる．

2. 汎用アセンブラ

2.1 PMP 汎用アセンブラの位置づけ

汎用アセンブラあるいはメタ・アセンブラと言った場合，それは3種類に大別される．

1) アセンブラ言語仕様を共通化または高級化することをねらったもの[8),11)~16)]．
2) 文字列処理機能に主眼を置いたもので，ソース言語から機械語への変換を文字列処理としてとらえるべきもの[17),18)]．
3) 種々の計算機の機械語を生成することに主眼を置き，アセンブリ・プログラム*** の共通化をねらったもの[1~7),9)]．

1) は機械独立な言語仕様であることから種々の計算機に適用できると言えるが，コンパイラと同様にオブジェクトの冗長さの問題があり，文献 8) で Ferguson が述べているようにコンパイラ・コンパイラの技術との関連が深い．言語仕様の共通化の試みをアセンブラ言語レベルで行うことは実用的でない．

2) は文字列処理機能は重視している点からアセンブラとしてのみ使うには余分な機能も多く，処理速度が遅いと考えられる．

3) はラベル，命令，オペランドの3つから成る通常のアセンブラ・ステートメントにおいて，機械語仕様に対応する命令名，オペランド表記法を計算機ごとに定義し，各機械語を生成するものである．1) と異なり，共通アセンブラ言語仕様化を第1の目的としないが，μP 出現後多種類の μP のクロス・アセンブラの心要性から，その実用的価値は大きい．

筆者らの実用化した PMP 汎用アセンブラは 3) に属している．

2.2 アセンブラ定義言語

前節に述べた 3) の方向で，アセンブラを定義する超言語を考えた場合それは次の3種類に区別される．

1) 命令の定義機能を備えたアセンブラ・マクロ[7)]
2) アセンブラ・アルゴリズムを記述するための手続型言語[1)]
3) 各命令のソース・プログラム上の表記法とオブジェクトとの対応を定義するだけの表形式の言語[9)]

1) は機械語命令およびオブジェクト・コードを定義するための命令を用意したものである．この方法は一般の専用アセンブラにおいても，拡張命令（マクロ命令）を定義するために採用されている例が多い．この命令は，ラベル，命令名はオペランドの形式すなわち，通常のアセンブラ・ソース言語と同様の形式を備えたものであり，擬似命令と考えられる．定義言語の仕様を擬似命令の形式に制限しているため，定義の容易さ，読み易さ，さらに適用性に問題がある．

2) は変数，ラベル，実行制御文 (GOTO 文；IF文) などを持つ手続型言語である．例えば文献 1) では，**Table 1** に示すような，種々の命令に対するオペランド部分の解析処理を記述するプリミティブと変数とをもち，これらによってアセンブル・アルゴリズムを記

Table 1 Elements to describe an assembler-algorithm which are used in METAS[1)].

	分類		形式または例
変数	特定目的変数	入力文字変数	@ INP
		オペコード変数	% OPC
		オペランド変数	% SYT
		ロケーション変数	＃ LOC
		ラベル変数	＃ L1
	作業用変数（文字，数値）		@@ A %% X ＃＃ Z
プリミティブ	分岐		GOTO（ラベル）
	判定		IFN（変数，条件，値）？ IFC（文字変数，条件，値）？
	関数		ILOC(1)《ロケーション・カウンタの更新》 GAV《オペランド値の評価》

* Meta language for Assembler Specification
** Program development system for Microprocessor
*** 超言語に対する言語処理系に対して，アセンブラ言語に対する言語処理系を特にアセンブリ・プログラムと呼ぶこととする．

述する．手続型言語でアセンブラのアルゴリズムまでを記述すれば柔軟性はあろうが，それでも変数・プリミティブが限られているので必ずしもすべての μP に適用できるとは限らない．もし，変数・プリミティブを追加して行くとすると，一般の汎用言語 (FORTRAN, PL/I) などでアセンブラを作成するのと差がなくなり，数多くの μP のアセンブラを作成する場合の生産性，保守性を考慮すると好ましいものではない．

一方，3) では計算機のアセンブル処理が共通化しやすいことに着目して，これを記述せずソース言語と機械語との対応関係のみを定義する表形式の言語を考える．この場合擬似命令セットは固定となり，原則的には1ステートメントが1機械語に対応するが，超言語の仕様を十分なものとすることにより大部分の μP にも適用でき，またアセンブラの定義が容易である．

3) の形式の超言語の適用例としては，マイクロプログラム・アセンブラがある[19~23]が，機械語レベルのアセンブラ用のものと比較して以下の点で相違がある．

① マイクロプログラムのソース・ステートメントはマイクロ・オーダの羅列から成り，マイクロ・オーダはニーモニックでのみ表わされる．従って，機械語レベルのアセンブラ言語に多い仕様——オペランドに式 ("アドレス±定数" など) およびアドレスシング・モードを与える特殊記号と式との組合せを書く仕様——を定義することは考慮されていない．

② マイクロ・オーダをそのままオブジェクトのビット・パターンに対応させることが中心で，機械語レベルのアセンブラとしてオブジェクト決定のためにしばしば必要となる．オペランドの値に対する変換のための関数，演算はあまり考えられていない．例えば文献 23) では関数として $DISP (自己相対アドレス) があるが，他は利用者のオウン・コーディング ($USER) としている．

以上に述べたことから，筆者らは生産性に優れた 3) の表形式の言語を基本とし，多様な μP の機械語レベルのアセンブラ言語仕様を定義できる超言語 MASP を設計した．

3. MASP 言語の設計

3.1 PMP 汎用アセンブラの言語仕様

前章に述べたことから，MASP は基本的アセンブル処理を記述せず，これを固定化している．すなわち，PMP 汎用アセンブラの言語仕様には MASP による定義にかかわらない固定部分がある．以下にそれを示す．

1) 擬似命令の種類，名前および機能は **Table 2** に示すような固定仕様である．

2) 文は自由書式 (カラム・フリー) で，ラベル欄，命令欄，オペランド欄，コメント欄から成り，各欄の区切り記号は MASP プログラム* で指定される．

3) オペランド欄は MASP プログラムで指定される区切り記号で区切られ，第1オペランド，第2オペランド… (これらを総称して第 i オペランドと呼ぶ) というように分割される．

4) 第 i オペランドは，命令の対象となるレジスタ番号，データ，アドレス等の数値を表現する算術式かアドレシング・モード等を指示する特殊記号かのいずれか (またはそれらの組合せ) である．算術式の仕様は固定とし，MASP プログラムでは規定しない．すなわち四則演算子で結合 (乗除算優先) されたもののみとする．

5) 第 i オペランドの算術式中に使用する定数の表記法は固定とする．例えば2進数は B'1010' かまたは，1010B，16進数は X'6F' かまたは 6FH のように書く．

以上のような固定の言語仕様としても，大部分の

Table 2 Functions needed as pseudo instructions.

分 類	機 能	今回採用した擬似命令の名前
データ定義	プログラム中で使用する定数・データをオブジェクト・コードとして生成する．但し，MASP で指定された基本語長を単位とする．	DC
領域確保	プログラム中で使用するデータ領域を確保する．	DS
アドレス制御	プログラムの配置アドレスを制御する．	ORG
プログラム開始	プログラムの開始 (アセンブルの開始) を指示する．	CSECT
外部記号の連絡	別々にアセンブルされるプログラム中で相互に記号を参照することを可能にする．	ENTRY EXTRN
記号の定義	記号 (ラベル) に定数を割り当てる．	EQU
プログラム終了	プログラムの終了 (アセンブルの終了) を指示する．	END
印刷制御	アセンブル結果のリストの出力を制御する．	TITLE EJECT SPACE PRINT

* MASP によるアセンブラの定義を MASP プログラムと呼ぶ．

```
$ALPHA パラメータ          ……アセンブラ・ソースプログラム中で
                            使用される英字の定義
$SOURCEパラメータ・リスト……アセンブラ・ソースプログラム中で
                            使用される区切り記号の定義
$RSV   パラメータ・リスト……アセンブラ・ソースプログラム中で
                            使用される予約記号の定義
$ADDR  パラメータ・リスト……メモリの基本語長(ビット数)の定義
$CHAR  パラメータ          ……文字の内部コードの定義
$ZONE  パラメータ・リスト……ゾーン形式10進数の形式の定義
$TYPE                  ┐
  :                    │  1つの命令タイプの定義
  :                    ├  この命令タイプに属するニーモニック
  :                    │  とそのオペランドの表記法および
$TYPE                  ┘  オブジェクトを定義する (Fig. 3 参照)
  :                    ┐
  :                    ├  (同上)
$TYPE                  ┘
  :                    ┐
  :                    ├  (同上)
$END                   ┘
```

Fig. 1 Structure of a MASP program.

μP アセンブラとして適用性の点で問題はなく，MASP によるアセンブラの定義は容易となる．さらに，擬似命令，定数表記法が共通で，区切り記号を MASP で全て共通に定義すれば，各種 μP のアセンブラ言語の習得が容易となる．

3.2 MASP の言語仕様

前節に述べた，PMP 汎用アセンブラ言語仕様の範囲内で，各 μP ごとの相違を MASP で記述する．MASP の定義 (MASP プログラム：**Fig. 1**) は次の2つの部分から成る

(1) アセンブラ・ソース言語の区切り記号，予約記号，メモリのアドレス付与単位など基本的な定義を行う部分．

(2) μP の各命令についての定義を行う部分．

前者は各目的に応じた文により定義される．後者はアセンブル処理が同一となる命令がまとめて定義される．すなわち，μP の全命令名 (ニーモニック) を分類し，命令タイプとして命令タイプごとに定義を行う．

Fig. 2 に 8080，**Fig. 3** に TLCS-12Aの1つの命令タイプの定義例を示す．前者は命令タイプ10の定義，後者は命令タイプ1の定義 (①がそのヘッダにあたる) であり，ともに②でその命令タイプの命令語のフィールド構成を，④でその命令タイプに属する命令 (前者 LXI 命令，後者 L, W, N, … 命令) を定義している．以下にこれを例として MASP 言語仕様の特徴的な部分を述べる．

1) オペランド表記法の指定：:

オペランド表記法は一般に算術式と特殊記号の組合せであるので，その組合せパターン——例えば ⑫ の ¥OPD 文では@S(B)——だけを記述すればよい．S, B は非終端記号であり，その終端記号の値 (ソース上の

```
$TYPE    TYPE 10                                          ……①
  ¥FLD   FN, 3, F1 (0,1), RPP (2,3), F2 (4,7),
           B2(8,15), B3 (16,23)                           ……②
  ¥RELF  FN, MEMAD (B3, B2)                               ……③
  ¥MN    FLD (F1, F2), (LXI, B'00', X'1')                 ……④
  ¥OPD1-1 RP                                              ……⑤
         RPP=COND (MOD(RP,2)=0->RP/2;
         TRUE->ERR(X))                                    ……⑥
  ¥OPD1-2 SPC(SP)                                         ……⑦
         RPP=3                                            ……⑧
  ¥OPD2  B32
         B2=PART(B32,1)                                   ……⑨
         B3=PART(B32,9)                                   ……⑩
```

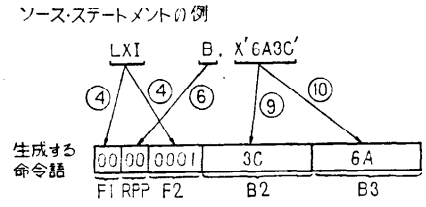

Fig. 2 Instruction-type definition for 8080.

```
$TYPE    TYPE3                                            ……①
  ¥FLD   FN, 2, F(0,3), RR(4,6), M(7,8), BB(9,11),
         SS(12,23)                                        ……②
  ¥RELF  FN, MEMAD (SS)                                   ……③
  ¥MN    FLD(F), (L, X'0'), (W, X'1'), (N, X'2'),
           (O, X'3'), (A, X'4'), (S, X'5'),
           (T, X'E'), (C, X'F')                           ……④
  ¥OPD1  R
         RR=R
  ¥OPD2-1 S(B)                                            ……⑪
         M=1
         BB=B
         SS=PART(S,1)
  ¥OPD2-2 @S(B)                                           ……⑫
         M=3
         BB=B
         SS=PART(S,1)
  ¥OPD2-3 S                                               ……⑬
         M=1
         BB=0
         SS=MOD(DISP(S)-1,4096)                           ……⑮
  ¥OPD2-4 @S                                              ……⑭
         M=3
         BB=0
         SS=MOD(DISP(S)-1,4096)
  ¥OPD2-5 =S                                              ……⑮
         M=2
         BB=0
         SS=PART(S,1)
```

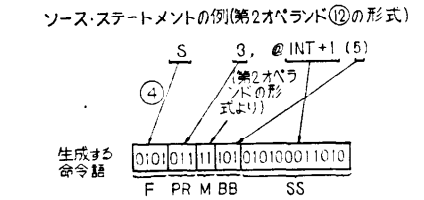

Fig. 3 Instruction-type definition for TLCS-12A.

Table 3 Functions to describe field-values.

関 数	意 味
DISP	ロケーション・カウンタとの相対値
MOD	剰 余
INV	ビット反転
PART	部分ビット列
LNG	記号の長さ属性

算術式の値）が代入文中の非終端記号の値として使用される．なお⑦の SPC(SP) は英字名が終端記号（ソース上の記号）であることを示す．さらにオペランド表記法はアドレシング・モード等により何種類か存在する時は，⑪〜⑮のように場合分けして定義する．オペランド表記法をこのようにパタンとして定義することは一般のアセンブラ・マニュアルにも採用されており，従って MASP によるアセンブラの定義を容易にしている．

2) オペランド表記法とオブジェクト・コードの対応::

⑤と⑥，⑦と⑧に示すように ¥OPD 文でオペランド表記法を，¥OPD 文に続くフィールドへの代入文形式でオブジェクト・コード（フィールド設定値）を指定し，対応関係を明確にした．

3) フィールド設定値の指定::

フィールド設定値はソース上の算術式の値そのものでない場合がある．MASP では非終端記号に対する四則，関数（**Table 3**）演算（例：⑨,⑯）により簡単に記述できる．さらに，条件式（例：⑥）によりオペランドの値に対して大小関係等の条件を与え，条件によって設定値を変更することができる．

4) 複数のフィールド構成がある場合の選択指定::

同一命令で複数のフィールド構成がある場合，どのフィールド構成にアセンブルするかはオペランド表記法で決定され，これを ¥SLCT 文で指定する．例えば2個のオペランドの表記法の組合せ——例えば，第1オペランドが ¥OPD1-1, 第2オペランドが ¥OPD 2-1 で指定される場合——でフィールド構成が決定する場合の例を下に示す．

¥SLCT FORM1, OPD1-1, OPD2-1

ここでFORM1はフィールド構成を定義する¥FLD 文の1つを選択するもの（¥FLD 文の第1パラメータ）である．

5) 再配置可能形式アセンブル・モードに対する記述::

③に示す ¥RELF 文により直接アドレスの入るフィールドを指定すればリンカが再配置時にアドレスの変更を行うので MASP プログラムでは再配置可能形式アセンブル・モードの定義については特に考慮が不要である．直接アドレスが2個または3個のフィールドに分離して入る場合も1文で記述できる．

7) エラー検出の記述が不要::

アセンブラ・ソースプログラムの誤りの検出については，以下の理由により大部分が汎用アセンブリ・プログラム（**Fig. 4** 次頁参照）で一意的に行われるので，MASP プログラムではほとんど記述が不要である．

（i）ニーモニックやラベルのつづり誤りは自動的に検出される．（ii）オペランド表記法の誤りは，MASP プログラムで指定された表記法とマッチングできない場合検出される．（iii）オペランドの値はフィールドへの代入文実行時にフィールド長をもとに検査され，誤りが検出される．ディスプレイスメント等のように，負数も入る可能性のあるフィールドへの代入文は修飾子を使用する．

（例） FLD:M=DISP(S)

（iv）オペランドの属性（値が再配置可能値か絶対値かを示す）の誤りはフィールド代入文実行時に検出される．このため右辺に使用する各関数値は引数によって決まる属性をもつ．各関数値を求める際には引数の検査が行われる．

4. PMP 汎用アセンブラの構成

超言語記述に基づく汎用アセンブラの実現には3つの方式がある[1]．

1) 超言語記述を変換して実行可能なプログラム（ホスト計算機の機械語プログラム）とする．
2) 超言語記述を変換して FORTRAN 等のプログラムとする．
3) 超言語記述をテーブル化して，このテーブルを解釈しながらアセンブルするプログラムを構成する．

1), 2) はジェネレータ方式またはコンパイラ方式，3) はテーブル参照方式またはインタプリタ方式と言うことができる．超言語が手続型言語の場合は 1),2) のように超言語記述を実行プログラムに変換する方がテーブルに変換するよりも直接的で容易である．しかし，MASP がアセンブラのソース言語と機械語の対応を表形式で定義する言語で，基本的アセンブル処理を記述していないことから，PMP 汎用アセンブラは 3) のテーブル参照方式で実現されている (Fig. 4 参照)．

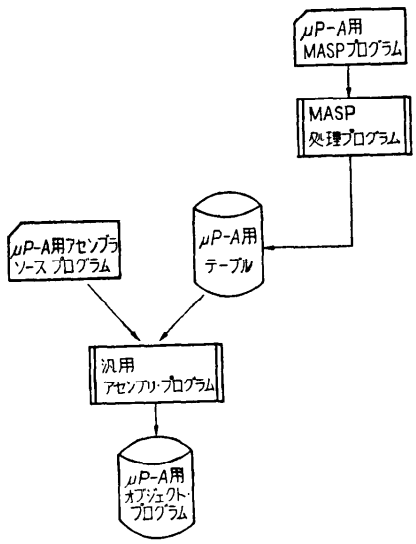

Fig. 4 Organization of PMP general purpose assembler.

5. 考　察

5.1 MASP の評価

MASP 言語は3章に述べた特徴により各種 μP のアセンブラ定義が極めて簡単であり，**Table 4** に示すように代表的な μP の MASP プログラムが数百ステップで記述できた．また μP の命令体系の理解と，MASP プログラムの作成準備作業とコーディング・デバグには，熟練者で 2,3 日で可能である．8080 専用アセンブラを作成した場合と，MASP プログラムを作成した場合とを比べると，ステップ数で約 1/60，工数で約 1/100 であり，MASP によるアセンブラ作成の高生産性を実証できた．

Table 4 MASP-program steps needed to define microprocessors' assembler.

分類	チップ名	ステップ数	分類	チップ名	ステップ数
4ビット	4004	72	8ビット	CDP 1802	74
	4040	80		Z-80	680
	μCOM 4	59		PPS-8	294
	μCOM 41	43	12ビット	TLCS-12	131
	μCOM 43	78		TLCS-12A	169
	PPS-4	87		IM 6100	64
8ビット	8008	101	16ビット	TMS 9900	204
	8080	110		PACE	260
	M 6800	266		PFL-16A	257
	MB 8861	270		LSI-11	998
	SC/MP	115		CP-1600	168
	F-8	143		DT 1235*	346
	CDP 1801	44			

* 公社の端末用プロセッサ

Table 5 Special assembly processes.

チップ名	処　理	内 容・目 的
PPS-8	コマンド/データ・プール処理	適当なコマンド/データをプールして短い命令語としオブジェクトを短縮する．
	ブランチ・タグのセット	次の命令がブランチ命令である場合にブランチ・タグをセットする．
M 6800	Direct/Extended アドレシング・モードの選択	オペランドのアドレス(値)によってアドレシング・モードを選択し，適宜短い命令語を使用しオブジェクトを短縮する．
IM 6100 PACE	間接アドレシングのポインタの自動生成	直接アドレシングで指定されたオペランドがアクセス範囲外のとき，アセンブラが自動的にポインタを生成して間接アドレシングとする．
CP-1600	BITS 擬似命令	アセンブル時に命令語のビット長をSDBD 擬似命令によって自由に変更できる．
	BITS 命令の自動挿入	オペランドの値によって SDBD 命令(2ワード・データ処理)を挿入する．オブジェクトのサイズの最適化を目的とする．
F-8	タイマ・カウント	CPU 内のタイマがシフト・レジスタ回路で実現されているため，アセンブラはソースプログラムで指定されたタイマ定数を対応表によりコード化する．

5.2 システムの拡張

今回実現したシステムでは，一部の μP の専用的アセンブル処理はできない．**Table 5** に示す処理がその例である．これらは，Fig. 3 のような単純な対応関係のみでアセンブルできず，アセンブル時に作成・更新される管理表を持ち，その内容を検索しつつオブジェクト・コードを決定する．これらに対処するとすれば，MASP 言語に配列が必要となり，配列要素への代入，検索を記述する必要が生じる．つまり，MASP 言語が手続型プログラミング言語となりアセンブラ作成の生産性の低下を引き起こす．このとき，MASP 言語のような超言語を一切放棄して，通常のプログラミング言語によるアセンブリ・プログラムを μP ごとに部分修正・追加（オウン・コーディング）することも考えられるが，アセンブラ設計者以外の人がオウン・コーディングすることは極めて困難となる．今後も新しい μP が続々と出現してくることを考えると，多少の専用的アセンブル処理を除外して，アセンブラ・ソースプログラムの記述法等で対処することとし，迅速にアセンブラの作成を可能とすることは実用的であろう．

一方，アセンブル時に内容の更新される管理表が必要なければ，MASP 言語の特徴を損なうことなく，MASP 言語仕様を改善および拡張することができ，本システムがより多くの機種 (μP に限定しない) に適用できる．例えば，PPS-4，μCOM-43 に見られる分割

型プログラム・カウンタ（プログラム・カウンタがページ・アドレスとページ内アドレスを保持する2つに分離しているもの）のためのエラー検出，さらにTMS-1000，μCOM-42に見られるポリノミアル・プログラム・カウンタ（ページ内アドレスを保持するカウンタがシフト・レジスタ回路で実現されているもの）などは対処可能となる．

5.3 アセンブル処理効率

PMP 汎用アセンブラはソースのオペランド部をMASP プログラムにかかわらず，決定的に処理し，その後 MASP プログラムで定義されたオペランド表記法とマッチングを行う．従って PMP 汎用アセンブラの汎用化によるオーバ・ヘッドは，専用アセンブラよりテーブル・アクセスが若干多いことのみである．

また，ラベル表にはハッシュ表を，ニーモニック表にはバイナリ・サーチを採用しており，1000 ステップのアセンブル時間は約 15 秒で，他[23]と比べても十分に実用的である．

6. あとがき

各種 μP に適用でき，かつアセンブラ定義の容易な超言語 MASP による PMP 汎用アセンブラについて報告した．MASP は，オペランド表記法をパタンとして定義することでオペランド解析手続を記述しないなど，アセンブラに共通的な手続を記述しなくてすむ非手続型超言語であり，さらにマイクロ・アセンブラ用の非手続型超言語にはない，複雑なオペランド表記法の定義，複雑で不規則なフィールド設定値の定義等の機能を備えている．

PMP 汎用アセンブラは電電公社 TSS(DEMOS-E) 上の PMP ライブラリとして MASP 処理プログラムおよび汎用アセンブリ・プログラムの両者共サービスされている．従って TSS 利用者自身が各自の仕様のアセンブラを作成することも容易である．システムはシステム記述言語 SYSL-2[24] で記述されている．

最後に，日頃御指導頂く横須賀電気通信研究所戸田統括調査役ならびに藤原応用プログラム研究室長に感謝致します．

なお，本研究は DIPS-1 プロジェクトの一環として横須賀電気通信研究所および日本電気(株)の共同研究として行われたものであり，ここに関係各位に深謝致します．

参考文献

1) Y. Nitta et al.: On an Efficient Assembler Building System —Metas Meta-Assembler—, Proc. 1st USA-JAPAN Computer Conf. 1972 pp. 442～447.
2) G. R. Johnson et al.: The Automated Generation of Cross-system Software for Supporting MICRO/MINI Computer Systems, SIGPLAN Notices, Vol. 11, No. 4, pp. 45～56 (April 1976).
3) 大星他：富士マイクロコントロール・システムのサポートシステム，富士時報，Vol. 48, No. 11, pp. 573～576 (1975).
4) MCS-X クロスアセンブラ（プログラミング・マニュアル）ASR 社.
5) 吉村他：マイクロコンピュータ用汎用クロスアセンブラ，信学会電子計算機技術研究報告 EC76-16.
6) 寺田他：マイクロプロセッサ用汎用リロケータブル・クロスアセンブラ —CRAMBO—，信学会電子計算機技術研究報告，EC 76-70.
7) 大原：異機種アセンブラの実現形態，情処学会第15回大会予稿集，pp. 431～432 (1974).
8) Ferguson: Evolution of the Meta-Assembly Program, CACM, Vol. 9, No. 3, pp. 190～196 (1966).
9) 上垣他：超言語記述による汎用クロス・アセンブラ，信学会電子計算機技術研究報告 EC76-35.
10) 藤田他：PMP 汎用クロス・ソフトウェアの適用範囲と生産性について，信学会全国大会，p. 6-267 (1977).
11) 大駒：共通アセンブリ言語とその変換プログラム，情報処理，Vol. 14, No. 3, pp. 165～172 (1973).
12) Niklaus Wirth: PL 360, A Programming Language for the 360 Computers, JACM, Vol. 15, No. 1, pp. 37～74 (1968).
13) Bell et al.: An ALGOL-like Assembly Language for a Small Computer, Software-Practice and Experience, Vol. 1, pp. 61～72 (1971).
14) 矢野他：マクロ機能によるあるシステム記述言語の試作，情報処理，Vol. 16, No. 9, pp. 760～766 (1975).
15) Charles Popper: SMAL —A Structured Macro-Assembly Language for a Microprocessor, Proc. Comp. Con. (1974).
16) 田島：SMAC-マクロ命令による構造化プログラミング言語，信学会技術報告，AL 75—55.
17) Graham: A Universal Assembly Mapping Language, Proc. ACM 20th National Conf. pp. 409～420 (1965).
18) Halpern et al.: XPOP: A Meta-Language Without Meta physics, Proc. FJCC., pp. 57～68 (1964).

19) 星他：マイクロプログラム・アセンブラの一構成法，情処学会，設計自動化研究会 75-24.
20) 藤井他：ミニコンピュータのための汎用マイクロアセンブラ，電気学会全国大会，pp. 1651～1652 (1973).
21) AMDASM : Advanced Micro Devices Microassembler AMD 社，マニュアル (1976. 9).
22) RAYASM : General purpose microcode assembler, RAYTHEON 社，マニュアル.
23) 田村他：高速汎用マイクロアセンブラ BARGEN/BASS の開発，信学会論文誌 D, Vol. J 60-D No. 7, pp. 523～530 (1977).
24) 寺島：DIPS-1 における高能率システム製造用言語の実用化，情報処理，Vol. 16, No. 6, pp. 492～498 (1975).

(昭和 52 年 9 月 12 日受付)
(昭和 52 年 12 月 26 日再受付)