プログラム・シミュレータ用ハードウェア記述言語†

神谷芳樹†† 北川愛子†† 仲谷元††
吉田清†† 藤田晨二††

マイクロ・プロセッサ (μP) 用プログラム・シミュレータは μP のプログラムをより高速なデバッグ・マシン上で疑似走行させるプログラミング・ツールで，一般にプログラムの開発系と実行系を異にする μP のプログラム開発において有効なツールとなっている．筆者らはこのプログラム・シミュレータを各種の μP ごとに効率よく，かつ LSI 設計者ではない一般の μP 利用者でも容易に作成可能とすることを目的として新しいハードウェア記述言語 "MHPL" を考案し，その言語処理プロセッサをインプリメントした．

ハードウェア記述言語については従来から多くの研究があるが，これらは主として設計対象としてのハードウェア記述が目的であった．これに対しプログラム・シミュレータ用言語はシミュレータのハードウェア依存機能の定義を目的としてハードウェア仕様を記述するもので，記述レベル，レポート機能の定義などで従来の言語と異なるいくつかの新機能が求められる．しかしながらこのようなプログラム・シミュレータ用言語の必要性が比較的新しいため，これまでこの点に配慮された言語が検討されてこなかった．そこで本論ではプログラム・シミュレータ用言語に必要な機能・プログラム構造についてはじめて考察するとともに，筆者らが MHPL において実現した一つの方式について報告する．そして MHPL を 16 種の μP に適用し記述性・性能が実用上充分であることを明らかにする．

1. まえがき

コンピュータのハードウェア記述言語については従来から多くの研究が発表されているが[1)-13)]，これらは主として設計対象としてのハードウェアの仕様記述が目的であった．これに対して本論で述べる言語はマイクロ・プロセッサ(以下 μP と略す)のプログラム・シミュレータを多数作成する際の生産性向上が目的であり，シミュレータの機能の内のハードウェア依存機能を定義するためにハードウェア仕様を記述するものである．両者の基本的相違はプログラム・シミュレータ用言語が，1) ハードウェア仕様を論理レベルではなく命令セット・レベル (ISP* レベル) で記述すること，2) ハードウェア仕様だけでなくシミュレータのレポート機能を μP ごとに適合させるためのレポート機能定義機能が必要な点にある．近年ハードウェア記述言語によるプログラム・シミュレータ作成の試みもいくつか発表されているが[14)~17)]，SIM/GEN[14)]を除いてはいずれも論理シミュレータと同様の方式を用いており，上記1)，2) の機能を満足せず，また SIM/GEN においても 2) の機能に対し全く配慮されていない．このためプログラム・シミュレータとしては処理速度が遅い，ハードウェア設計者でない一般利用者には記述が難しい，μP ごとに適合したレポート機能が実現できない，などの問題があった．

そこで本論ではプログラム・シミュレータ用言語の設計時に考慮すべきハードウェア記述レベル，レポート機能の定義方式，言語形式とプログラム構造について従来の言語と対比して明らかにし，同時にこの考察にもとづいて MHPL* で実現した一つの方式，すなわち，1) ハードウェア仕様をマシン命令単位のブロック構造によって定義する方式，2) レポート機能を直接そのアルゴリズムを記述することなく，簡易なパラメータ定義とハードウェア仕様の implicit な定義の併用で実現する方式，を導びく．そして MHPL を実際に 16 種の μP に適用して本方式が生産性，性能双方の面で充分な実用性を備えていることを示す．

2. プログラム・シミュレータ用言語

2.1 シミュレータの機能とレポート情報

シミュレーション・デバッグ実行時のプログラム・シミュレータの入力はホスト・マシン上に作られた μP のオブジェクト・プログラムとシミュレーションの実行を制御するためのシミュレータ・コマンドであり，出力は後述するような各種レポート情報である．

その処理の流れを一般的に示すと**図1**となる．全体

† A Hardware Description Language for Program Simulator by YOSHIKI MITANI, AIKO KITAGAWA, HAJIME NAKATANI, KIYOSHI YOSHIDA and SHINJI FUJITA (Yokosuka Electrical Communication Laboratory, N. T. T.).

†† 電電公社横須賀電気通信研究所データ通信研究部

* Instruction Set Processor

* Microprocessor Hardware Programming Language

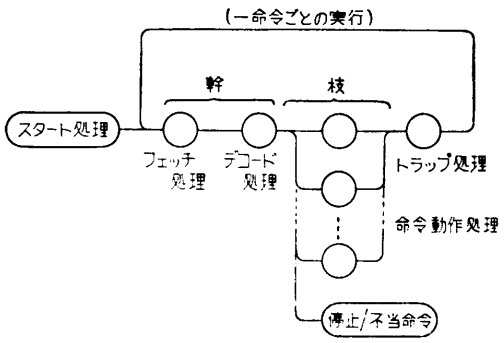

図 1 シミュレーションの流れ図
Fig. 1 Sequence chart of the simulation process.

は大きな多方向分岐を含む1つのループで表現でき，シミュレータはこれを情報収集単位ごと，すなわち1マシン命令実行ごとに1回転する．ループに「幹」，「枝」という区分をつけると幹でオペコードのフェッチ，デコード，枝で個々の命令動作を疑似し，最後にトラップ処理と呼ぶ命令動作単位の処理を行う．

トラップ処理はシミュレータ・コマンドの指示に基づいて一命令動作の疑似終了ごとににトレース・リストなどのレポート情報を出したり，シミュレーションの一時停止，再開などの実行制御を行う処理である．そしてこの処理の契機をトラップ契機と呼ぶ．

プログラム・シミュレータの機能の特徴はレポート情報に顕著に示される．レポート情報の出力はシミュレータ・コマンドで様々の項目を選択するのが一般的であるが，その基本はトレース・リストである．**図2**に2種の μP について出力例を示す．トラップ契機ごとに出力される一行分のリストには，ⓐ命令アドレス，ⓑ分岐発生時の分岐先アドレス，ⓒ逆アセンブル・ソース・イメージ，ⓓフェッチしたオブジェクト・コード，ⓔ命令実行後のメモリ・レジスタ類の状態などの情報が含まれている．

シミュレータ生成用言語の目的は，このようなシミュレータの機能のうち μP 機種依存機能の効率よい記述である．その特徴は，1）図1の枝に対応して，ISP レベルでハードウェア仕様を定義すること，および2）レポート出力機能に関して複雑な機種依存情報の定義が必要なことにある．

2.2 ハードウェア記述レベル

ハードウェア記述言語の記述レベルとはハードウェア仕様を記述する細かさのレベルで，一般には文献11) に示されているように PMS レベルからゲート・レベル，回路レベルに至るまで様々のレベルが考えられる．シミュレータの生成を目的とする言語ではシミュレータの処理速度と記述の容易さの2つの点からシミュレーション機能に対応したレベルが必要であり，プログラム・シミュレータではこれがマシン命令の実行を単位とする ISP レベルに相当する．

（1） 展開率と処理速度

シミュレータのハードウェア疑似機能はハードウェア記述言語によって記述されたハードウェア仕様を何らかの言語処理によってホスト・マシンの命令群に翻訳することで実現される．そこでハードウェアの記述レベルが細かすぎると展開率（ターゲット・マシンの単位動作当りのホスト・マシンの動作）が増大し，シミュレータの速度が低下する．また逆に粗すぎるとシミュレータの分解能が低下する．

（2） 記述の容易さ

ハードウェア仕様を記述するシミュレータの作成者はハードウェア設計者ではなくマシン命令の利用者であるため，ハードウェア仕様はこの利用者の着目する

```
**** トレース リスト ****     (I8085)
ADDR          S,IMAGE     OBJECT   AR BR CR DR ER HR LR C A Z P S     #CLOCK SP   M[10D-110]
0100          LHLD        2A 0F 01 00 00 00 00 00 2A 3E 0 0 0 0 0       21 0000 D9 B6 3E 2A
0103          LDA         3A 0D 01 D9 00 00 00 00 2A 3E 0 0 0 0 0       38 0000 D9 B6 3E 2A
0106          SHLD        22 0E 01 D9 00 00 00 00 2A 3E 0 0 0 0 0       61 0000 D9 3E 2A 2A
0109          STA         32 0F 01 D9 00 00 00 00 2A 3E 0 0 0 0 0       79 0000 D9 3E D9 2A

**** トレース リスト ****     (MN1499)
ADDR          S,IMAGE     OBJECT X Y C Z DM[10-13] DM[20-23]     #CLOCK      #STEP A
0000          LX    1     31     1 0 0 0 1 2 0 4   8 1 9 0           1          1 0
0001          L           00     1 0 0 0 1 2 0 4   8 1 9 0           2          2 1
0002          LX    2     32     2 0 0 0 1 2 0 4   8 1 9 0           3          3 1
0003          ST          0A     2 0 0 0 1 2 0 4   1 1 9 0           4          4 1
0004          ICY         2C     2 1 0 0 1 2 0 4   1 1 9 0           5          5 1
0005          CY    4     A4     2 1 0 0 1 2 0 4   1 1 9 0           6          6 1
0006-0000     BNZ         E2 00  2 1 0 0 1 2 0 4   1 1 9 0           8          7 1
0000          LX    1     31     1 1 0 0 1 2 0 4   1 1 9 0           9          8 1
ⓐ ⓑ           ⓒ           ⓓ     ⓔ
```

図 2 シミュレーション・トレース・リスト例
Fig. 2 Example of the simulation trace-list.

レベルで記述できる必要がある．これが ISP レベルである．

（3） 時間更新とトラップ処理の契機

従来のハードウェア設計用シミュレータに対しプログラム・シミュレータの特徴は時間の更新契機とトラップ契機の分離にある．すなわちハードウェア設計用では図1の1本の枝の機能が単位時間内の動作に対応するため，これを1文もしくは1つのブロック構造中に記述することによって時間の更新とトラップ契機を文またはブロックの区切りという形で implicit（陰）に表現することができる．

これに対しプログラム・シミュレータでは複数の時間更新の後にトラップ契機となる．そして1つのトラップ契機から次の契機までの途中経過は不要であり，到達した状態のみが必要とされる．

そこで従来のハードウェア設計用言語を転用すると両者の分離を表現できず，多くはクロック単位のシミュレータを生成することになり，これが前述の展開率の増大から速度低下を招いていた[16),17)]．

これまでにこの相違に着目してプログラム・シミュレータ用に新しい言語を設計したものは SIM/GEN に一例をみるのみである．

（4） MHPL の方式

MHPL は記述の容易さの観点からトラップ契機については従来の言語同様陰に表現し，時間更新については直接記述する方式で両者を分離した．すなわち図1の枝を1ブロック内に記述し，ブロックの終了によってトラップ契機を表現する．時間の更新は予約変数を用いてブロック内に直接，式で記述する．

2.3 レポート機能の定義

レポート機能はシミュレータのマン・マシン・インタフェースを確保する機能で，プログラム・シミュレータにおいてはハードウェア設計用シミュレータと異なり各 μP ごとに適合した機能が求められる．

このレポート機能を μP ごとに適合させることはハードウェア仕様の定義と同時にレポート機能を定義することにより可能となるが，このときレポート機能のアルゴリズムを記述していては結果としてシミュレータの大部分の機能を一般の高級言語で記述することと大きな差が無くなり，本来の生産性向上という目的に合致しない．

そこでレポート機能を分析して個別機能を抽出し，その効率良い定義方式の確立が必要となる．以下に前述のトレース・リストの出力機能を例にこの点を明らかにする．

先ずトレース・リスト出力の特徴を示すと次の3点になる．

1） 情報収集契機と出力契機の分離：一命令実行中に様々の契機で情報を収集し保存しておいて，一命令実行終了時に一行に編集し出力する．

2） 分岐の検出：分岐発生時にのみ分岐先アドレスを埋め込む．

3） 逆アセンブル・ソース・イメージの埋め込み．

1）を図2で示すと次のⓐ～ⓔの情報のトラップ契機における一勢出力となる．ⓐ先頭アドレスとして先頭語フェッチ直前のプログラム・カウンタの状態，ⓑ分岐先アドレスとして分岐命令実行後のプログラム・カウンタ，ⓓオブジェクト・コードとして一語フェッチごとに命令レジスタの内容を保存・スタックした内容，ⓔ各種レジスタ，メモリ情報として命令実行後の各々の状態．

そしてこの機能を実現するためには，1）収集対象情報とその収集契機，2）分岐の発生，3）ソース・イメージとその編集方法，が定義されればよい．

MHPL では宣言部，手続き部という言語構成の他に，レポート部という最大7文から成る小さな定義部を設け，ここでパラメータ形式でいくつかの情報を定義し，さらにハードウェア動作を定義する手続き部の記述に一定の規約を設けて陰に上記の情報を定義することでレポート機能の適合化を図った．

2.4 言語形式とプログラム構造

ハードウェア仕様を記述する言語の形式を区分する要因として，1）パラメータ定義とアルゴリズム定義（論理定義），2）手続き型言語と非手続型言語，がある．

（1） パラメータ定義とアルゴリズム定義

これはハードウェア動作をあらかじめ定形化して用意し，個々の μP 仕様をこのパターンに対するパラメータとして定義するか，あるいは μP の動作のアルゴリズムを直接定義するかの相違である．

SIM/GEN の中の命令定義用言語 IDL や FORTRAN の CALL 文を利用したもの[13)]が前者であり，APL[6)~9)]，CDL[10),15),16),18)]などが後者の典型例である．

多機種の μP に関しその動作に共通性が多ければパラメータ方式は有効であるが，現実にはこの方式のみで多様な μP を表現するのには無理があり，1）パターンからはずれた動作の記述が困難，2）パターンの

```
——IDL (SIM/GEN)——                      ——MHPL——
VERBAL EXPLANATION: RAR               ¥¥RAR IBLK;
OPECODE VALUE       : F6
ESTIMATED TIME      : 10.8              /* OPCODE F6 */
IDL STATEMENTS:
                                        A.C=&RR A.C;
  AND TEMP4  SREG1  1                   #CLOCK=#CLOCK+4;
  SHRL SREG1 SREG1  1
  SHLL SREG2 SREG2  3                   END;
  OR   SREG1 SREG1  SREG2
  MOVE SREG2 TEMP4
END INSTR
```

図 3 IDL (SIM/GEN)[14] と MHPL による一機械語命令 (I 4040 の RAR 命令) の記述

Fig. 3 Description of an instruction operation (RAR of I 4040) by IDL (SIM/GEN)[14] and MHPL.

種類を次々に追加する必要が生ずる，などの問題がある．

実際に SIM/GEN の IDL では最近の 4 ビット μP に多く見られる複合的な命令や，Z 80 のブロック転送命令のような高機能の動作は表現できない．

MHPL ではアルゴリズム定義方式を採用すると同時にパラメータ定義方式の利点も考慮して共通ブロックという形式のサブルーチン機能，およびマクロ展開機能を設けた．**図 3** に IDL と MHPL の簡単な表現を比較して示すが，後者の方が自然な表現であり，記述性において優れていると考えられる．

(2) 手続型言語と非手続型言語

これはプログラムの記述順序と実行順序の関係の有無による相違で，記述順に実行されるのが手続型言語，記述順に依存しないのが非手続型言語である*．両者の中間に各種のブロック構造が考えられる．

前者の例として APL，後者の例として CDL があるが，CDL は正確には小さい単位のブロック構造を持っている．両方式の適否は図 1 のプログラム構造で判断できる．

すなわち論理シミュレータのレベルではハードウェア動作は基本的に並列である．図 1 で示すと，同時に多数のパスが選択され，また一本の枝に相当する機能が 1 クロック中の単機能であるため小さい．したがって**図 4** に示す CDL (後述) のラベル文のように 1 クロック中の単機能を 1 文もしくは少数の文でブロックとして記述し，この単位ごとにラベルを付与して，ラベルによって文 (またはブロック) の実行を制御する形式の非手続型言語が有効であった．

一方命令セットレベルではこれと反対にハードウェア動作は直列動作であり，また一情報収集単位内の機能も大きい．図 1 で示すと選択されるパスは常に一本で，また一本の枝に含まれる機能が上記に比して大きい．このような場合，手続型言語の方がハードウェア

——CDL——

```
DECORDER, K(0-20)=F      ①**   (宣言部)*
    ⋮                            *他の宣言文省略
TERMINAL, ADD=K(0),      ②***    PC: プログラムカウンタ
          SUB=K(1),      ③        X: アドレスレジスタ
              ⋮                  IR: 命令レジスタ
          STP=K(10),     ④       IR (OP): IR のオペコード部
        FETCH=K(11),     ⑤       IR(ADDR): IR のアドレス部
              ⋮                  M: メモリ
CLOCK, P                 ⑥
    ⋮
                                               (ラベル文)****
/K(13)*P/     X=PC, F=11                                ⑦
/FETCH*P/     IR=M (X), PC=PC. COUNT, F=15              ⑧
/K(15)*P/     F=IR (OP), X=IR (ADDR)                    ⑨
/ADD*P/       IR=M (X), F=14                  ⑩
/K(14)*P/     A=A .ADD. IR, F=13              ⑪
/SUB*P/       IR=M (X), F=17                  ⑫
/K(17)*P/     A=A .SUB. IR, F=13              ⑬
    ⋮
```

** F の値が n のとき $F(n)=1$ となる．
*** $K(n)$ の値が左辺の評価値となる．
**** ラベル中の論理式が正のとき，その文が実行される．
命令フェッチ時の流れは⑦・①・⑤・⑧・①・⑨．次いでオペコードが 0 なら①・②・⑩・①・⑪となる．

——MHPL——

```
/LABEL/ ¥OP[|0Q]; ①**   (言宣部)*   *他の宣言文省略
    ⋮                                PC, X, IR, M は
                                     左と同じ
$PROC                 ②(手続き部)***  IOP: IR のオペコード部
    ⋮                                IAD: IR のアドレス部
  ¥FETCH    X=PC;       ③
            IR=M [X]; PC=PC+1;                     ④   主手続き
            X=IAD; GOTO ¥OP [IOP];                 ⑤   (フェッチ・
  ¥OP [0]   CALL ¥¥ADD ⑥; GOTO ¥FETCH;             ⑦   デコード)
  ¥OP [1]   CALL ¥¥SUB;  ; GOTO ¥FETCH;            ⑧
    ⋮
  ¥OP [|0Q] CALL ¥¥HLT;                            ⑨
    ⋮
  ¥¥ADD IBLK;                                      ⑩   命令ブロック
               IR=M [X]; A=A+IR; END;              ⑪
  ¥SUB IBLK;                                       ⑫
               IR=M [X]; A=A-IR; END;              ⑬
    ⋮
$END                                               ⑭
```

** ラベル配列の宣言，¥XX はラベルの意味．
*** 命令フェッチ時の流れは③・④・⑤．次いでオペコードが 0 なら⑥・⑩・⑪・⑦ となる．¥¥XX はブロックを意味する．

図 4 CDL[10] と MHPL による直列動作の記述

Fig. 4 Description of sequential operation by CDL and MHPL.

* Y. Chu の定義 (文献 10) p. 143) に基づく．文献 2) でも M. R. Barbacci が各種ハードウェア記述言語の比較評価において同じ観点を示している．また文献 15) では D. R. Smith が同様の観点に立って CDL にかわって CSL という手続型言語を提案している．

動作を自然に記述できるという利点があるが，また同時に各文の順序が意味を持つことによって，プログラム開発時の生産性が低下するという欠点がある．

そこで MHPL では次の点を考慮し，図4に CDL と比較して示すようなブロック構造を持った手続型言語とした．

1) 手続型言語で直列動作を自然に記述する．2) ブロックの終了により implicit にトラップ契機を表現する．3) ブロック相互間の記述順序に意味を持たせないことによりブロック単位の非手続性を確保し，記述の容易性とプログラムの読解性を確保して生産性の向上を図る．

図4の比較では，CDL が文と文の実行の間に毎度デコード関数とラベルの評価が介在するのに対し，MHPL では直列動作が自然に記述され，後者の利点が明らかに示されている．

MHPL によるプログラムの全体の構成は**図5**のように，宣言部，レポート部，手続き部の3部構成とした．宣言部でレジスタ，メモリなどのハードウェア構成を宣言し，手続き部でこれらを変数としてハードウェア動作を記述する．図1の幹の機能を主手続き，1つの枝を命令ブロックとして記述する．命令ブロックは主手続きから CALL され，処理終了後 RETURN または END で主手続きに戻る．また共通的な記述は共通ブロックとしてまとめることもできる．

```
$DCL                                                              (宣言部)
  /INDEX/  LSB=0;                   ①
  /MEM/    M[2048,8],DM[64,4];      ②
  /PORT/   PORT[5,12];              ③
  /REG/    DC[7],PC[11],A[4],IR[8],CT[8],SR[2,11],SP[1],P[1],C[1],    ④
           Z[1],S[1],E[1],MOUT[1],W1[4],CP[12],IRU[8],C1[1],
           /***** SUB-RESISTER *****/
           X[3]=DC[6-4],Y[4]=DC[3-0],PCL[8]=PC[7-0],PCU[3]=PC[10-8],
           CL[4]=CT[3-0],CU[4]=CT[7-4],IRL[4]=IR[3-0],IRU[4]=IR[7-4],
           IRLL[2]=IR[1-0];
  /INTEGER/ WORK;                   ⑤
  /CHAR/   SSTAC[256,10];           ⑥
  /CDATA/  SSTAC='NOP      ','TAX      ','TYA      ','TAY      ',   ⑦
                 'AND      ','OR       ','XOR      ','A        ',
                 'CPL      ','C        ','ST       ','STIC     ',
                 ⋮ (省略：逆アセンブル・イメージ用文字データ)
                 'OTIE C   ','OTIE D   ','OTIE E   ','OTIE F   ';
  /LABEL/ ¥BLK[16],¥B0[16],¥B1[16],¥B2[16],¥B3[8],¥BE[16];            ⑧
  /LINE/   SNSC;                    ⑨
$END
$REP                                                              (レポート部)
  /INIT/ SP=1;                      ⑩
  /SIMG/ SSTAC[IR0];                ⑪
  /PCOUNT/ PC;                      ⑫
  /IREG/ IR,2;                      ⑬
  /BRANCH/ ¥BR0,¥BR1,¥BR2,¥BR3,¥BR4;      ⑭
  /EDIT/ HEXA,8;                          ⑮
$END
$PROC                                                             (手続き部)
  ¥START      PC=#STAD;                   ⑯                      (主手続き)
  ¥RESTART;                               ⑰
  ¥FETCH      IR=M[PC]; IR0=IR; PC=PC+1; #STEP=#STEP+1;   ⑱
              GOTO ¥BLK[IRU];                     ⑲
     ¥BLK[0]    CALL ¥¥BLK0; GOTO ¥FETCH;         ⑳
     ¥BLK[1]    CALL ¥¥BLK1; GOTO ¥FETCH;
     ¥BLK[2]    CALL ¥¥BLK2; GOTO ¥FETCH;
      ⋮ (省略：デコード処理)
     ¥BLK[15]  CALL ¥¥BLKF; GOTO ¥FETCH;
  ¥¥BLK0 IBLK;                                                    (ブロック群／命令ブロック)
             #CLOCK=#CLOCK+1;         ㉑
             GOTO  ¥B0[IRL];            ㉒
     ¥B0[0]  RETURN;                  ㉓                    /*  NOP  */
     ¥B0[1]  X=A[2-0]; RETURN;                              /*  TAX  */
     ¥B0[2]  A=Y; CALL ¥¥FLG1; RETURN;                      /*  TYA  */
     ¥B0[3]  Y=A; RETURN;                                   /*  TAY  */
     ¥B0[4]  A=A &AND DM[DC]; CALL ¥¥FLG1; RETURN; ㉔       /*  AND  */
     ¥B0[5]  A=A &OR DM[DC];  CALL ¥¥FLG1; RETURN;          /*  OR   */
           ⋮ (省略)
   ¥B0[13]; ¥B0[14]; ¥B0[15]  A=DM[DC];              ㉕     /*   L   */
             IF IRL=13  THEN GOTO ¥S01;                     /*  LIC  */
             IF IRL=14  THEN Y=Y+1; ELSE Y=Y-1;             /*  LDC  */
      ¥S00   #CLOCK=#CLOCK+1; CALL ¥¥FLG2; RETURN;
      ¥S01   CALL ¥¥FLG1;
  END;                                ㉖
           ⋮ (省略：命令ブロック群)
  ¥¥FLG1 CBLK;                                                    (共通ブロック)
             IF A=0  THEN Z=1; ELSE Z=0;
  END;
           ⋮ (省略：共通ブロック群)
$END
```

図 5 MHPL の記述例 (MN 1499)

Fig. 5 Example of MHPL description (MN 1499).

3. MHPL の言語仕様

MHPL の言語仕様を**表1**に示す．以下に**図5**の記述例を参照して言語仕様を述べる．

(1) 宣 言 文

宣言部では10種の宣言文で，メモリ空間，ポート，レジスタ類，信号線の他，整数変数，文字変数，ラベル配列などを宣言する②〜⑨．/INDEX/ 文①では以降の配列表現の指標づけ方式を宣言する．

(2) レポート文

レポート部には7種のレポート文があり，これらと次に示す手続き部で一定の規約を守ることで μP ごとに適合したレポート機能を定義する．

前述のトレース・リスト出力に関する両者の機能を**表2**に示す．

1) **収集対象情報**：レポート文でプログラム・カウンタ，命令レジスタ，最長命令語長，ソース・イメージ用文字変数を定義する．メモリ・レジスタ情報の対象はシミュレータ・コマンドでシミュレーションの都度，宣言部で定義した変数名の中から任意に指定する方式なので特別な定義は不要である．

2) **情報収集契機**：命令フェッチ契機は手続き部中で予約ラベル ¥FETCH で陽に定義する．一方命令

表 1 MHPL 言語構成の概要
Table 1 An outline of MHPL language structure.

```
〈プログラム構成〉::=〈宣言部〉〈レポート部〉〈手続き部〉

〈宣 言 部〉::=$DCL〈宣言文〉; …; $END
〈レポート部〉::=$REP〈レポート文〉; …; $END
〈手 続 き 部〉::=$PROC〈主手続き〉〈ブロック〉…$END

〈宣 言 文〉::=〈/INDEX/文〉|〈/MEM/文〉|〈/PORT/文〉
              |〈/REG/文〉|〈/LINE/文〉|〈/INTEGER/文〉
              |〈/CHAR/文〉|〈/CDATA/文〉|〈FIX/文〉
              |〈/LABEL/文〉
〈レポート文〉::=〈/INIT/文〉|〈/INT/文〉|〈/SIMG/文〉
              |〈/PCOUNT/文〉|〈/IREG/文〉|〈/BRANCH/文〉
              |〈/EDIT|文〉
〈主 手 続 き〉::=[〈文ラベル〉]〈手続き文〉; …;
〈ブ ロ ッ ク〉::=〈命令ブロック〉|〈共通ブロック〉
  〈命令ブロック〉::=〈入口〉IBLK [〈文ラベル〉]〈手続き文〉; …; END
  〈共通ブロック〉::=〈入口〉CBLK [〈文ラベル〉]〈手続き文〉; …; END

〈手続き文〉::=〈転送文〉|〈出力文〉|〈交換文〉|〈GOTO 文〉|〈CALL 文〉
              |〈RETURN 文〉|〈IF 文〉
  〈転送文〉::=〈変数〉=〈式〉|〈文字変数〉=〈文字定数〉
  〈出力文〉::= ? =〈出力要素〉, …
  〈交換文〉::=〈変数〉〈→〉〈変数〉
  〈GOTO 文〉::=GOTO〈行先ラベル〉  〈CALL 文〉::=CALL〈入口〉
  〈RETURN 文〉::=RETURN
  〈IF 文〉::=IF〈式〉THEN〈手続き文〉[; ELSE 〈手続き文〉]

〈変 数〉::=〈メモリ〉|〈ポート〉|〈レジスタ〉|〈アレイレジスタ〉
            |〈信号線〉|〈整変数〉|〈連結変数〉
〈式〉 ::=〈項〉|〈単項式〉|〈二項式〉|〈複合式〉
  〈項〉 ::=〈変数〉|〈定数〉
  〈単項式〉::=〈単項演算子〉〈項〉  〈二項式〉::=〈項〉〈二項演算子〉〈項〉
〈出 力 要 素〉::=〈変数〉|〈文字定数〉|〈文字変数〉
〈文 ラ ベ ル〉::=¥〈名標〉[〈定数指標〉]
〈行先ラベル〉::=〈文ラベル〉|¥名標〈変数指標〉
〈入 口〉::=¥¥〈名標〉

〈単項演算子〉::=&MOD|&EP|&OP|&NOT|&COM|&SR|&SL
              |&RR|&RL
〈二項演算子〉::=+|−|×|/|&AND|&OR|&XOR|〈|〉|=|〈=|〉
              =|∧=

〈予約ラベル〉::=¥START|¥RESTART|¥FETCH|¥INT
〈予 約 入 口〉::=¥¥INVC|¥¥HLT
〈予 約 変 数〉::=#CLOCK|#STEP|#IDT|#ODT|#STAD|#INT
              |#ILEV

〈連結子〉::=・  〈注釈〉::=/*〈文字列〉*/
```

注) …は——部分の繰り返し,[ ] は省略可能; 太字記号は直接記述を示す.

の終了，すなわちトラップ契機は命令ブロックの復帰または終了 (RETURN, END) で，また命令コードのスタック契機は指定された命令レジスタの更新によって陰に定義される.

3) 分岐の発生：これは手続き部の分岐発生時のパス中のラベルをレポート部で列挙しておくことにより定義する (/BRANCH/文⑭).

4) ソース・イメージ編集：シミュレータはトラップ契機にレポート部の /SIMG/文で指定した文字変数をトレース・リストに埋め込む．そこでまず宣言部で

表 2 トーレス・リスト出力機能に関する情報の定義
Table 2 Information definition for trace-list printing facility.

	トレース・リストの項目	収集対象の定義 (レポート部)	収集契機の定義 (手続き部)
ⓐ	先頭アドレス	/PCOUNT/文でプログラム・カウンタ ⑫	予約ラベル ¥FETCH ⑱
ⓑ	分岐先アドレス	対象は上と同じ 分岐発生は /BRANCH/ ラベルを文で指定 ⑭	命令ブロック終了時 {RETURN ㉔ {END ㉖ 分岐発生パスにラベル
ⓒ	ソース・イメージ	宣言部で文字テーブルを定義⑥⑦ レポート部の /SIMG/文でテーブル要素を変数インデックスを用いて指定⑪	命令ブロック終了時 ㉕㉖
ⓓ	オブジェクト・コード	/IREG/文で命令レジスタと命令語長を指定 ⑬	命令レジスタ更新時⑱
ⓔ	メモリ・レジスタ情報	シミュレーションの都度，シミュレータ・コマンドで，宣言部で定義したメモリ・レジスタ名の中から指定	命令ブロック終了時 ㉕㉖

命令コードなどをインデックスに対応させてソース・イメージ用文字テーブルを宣言しておく⑥⑦．次に /SIMG/文で，フェッチ時に命令レジスタをセーブしておく変数をインデックスとしてテーブル要素を指定しておく⑪．これだけで逆アセンブル・ソース・イメージをトレース・リストに埋め込む定義ができる.

レポート文には他に 0 以外の初期設定が必要な変数への初期設定，割込み条件設定用コマンドの機能定義，定数の編集方式などの定義を行う文⑮がある.

(3) 手 続 き 文

手続き部にはハードウェア動作を記述する 7 種の手続き文がある．これらによってデータの転送，交換，各種演算，比較などを表現する．また出力文により変数あるいは文字定数の印字出力を直接指示できる．GOTO 文ではラベル配列を用いてデコード処理を効率よく記述できる⑲㉒.

(4) 処理用データ

宣言部で宣言する変数および整定数，文字定数の他に 7 つの予約変数がある．そのうち #CLOCK, #STEP はクロック数，ステップ数を表現し，手続き部中でこれらの積算を式によって記述する⑱㉑.

ビット列データと整数データの型変換は言語処理プロセッサが自動的に行い MHPL では意識しない.

(5) 制御用データ

ラベルの他，ブロックを示す入口がある．ラベルにはラベル配列を許す．また，各々特別な機能の予約ラベル，予約入口があり，プログラムの開始位置や，不当命令・停止命令デコード時に呼び出すシステム側の

ブロック名などを示す.

(6) 演 算 子

演算子は 22 種あり，これらを用いた式を転送文の右辺あるいは IF 文中に記述することによりハードウェアの論理動作を記述する.

4. プログラム構成

MHPL の処理系は MHPL ソース・プログラムを翻訳し，これを既作成の共通機能と結合して，μP 対応のシミュレータを生成する．宣言部，レポート部の記述から，(i)レジスタ・メモリなどに対応した領域，(ii)以降の言語処理自身のためのテーブル群，(iii)レジスタ名などを登録するシミュレーション制御のためのテーブル群が生成される．主として手続き部は図1に示した幹，枝の部分，共通機能がトラップ処理に対応した手続きとなる.

シミュレータ・コマンドやブレーク・ポイントなどのデバッグ処理は，レポート出力機能を含めて基本的には共通機能によって実現されるが，これらは上記(ii)，(iii)のテーブル群の働きによって μP ごとに適合した処理となる．(ii)は言語処理系がシミュレータ中に必要な手続きを埋め込むのに参照し，(iii)はシミュレーション実行時にシミュレータの手続きが参照する.

例えばトレース・リスト条件指定用コマンドの基本機能は各 μP 共通であるが，そのパラメータで指定するレジスタ名は各 μP 固有のものとなる.

言語処理方式は，本システム全体の記述言語でもある高能率言語 SYSL-2[22] を中間言語とするコンパイラ方式である．MHPL は商用 TSS (DEMOS-E) 上で PMP* 汎用支援ソフトウェア群[19)-21)]の一環として提供され，生成されるシミュレータもまた同システム上で，23 種のシミュレータ・コマンドを用いて会話的に利用できる.

表 3 一プロセッサのシミュレータ定義に要する MHPL ステップ数

Table 1 MHPL-program steps needed to define one microprocessor's simulator.

語長	マイクロ・プロセッサ名	ステップ数	語長	マイクロ・プロセッサ名	ステップ数
4ビット	MN 1499	228	8ビット	M6800	534
	SM 4	331		MB 8861	574
	MELPS-4	437		SCAMP	440
8ビット	I 8080	558		COSMAC(1802)	382
	μCOM 80	575	12ビット	TLCS-12A	369
	I 8085	685	16ビット	PFL-16A	424
	Z-80	1339		TMS 9900	535
	F-8	446		I 8086	1637

5. 評 価

MHPL の 16 種の μP に対する適用結果を記述ステップ数で示せば表3のとおりである．特に記述性に関する問題はなく，MHPL の高い記述性を実証できた．記述量は命令数，アーキテクチャおよびデコーダの複雑さにほぼ比例し，Z-80，I 8086 だけが1K ステップを超えている以外はあまり語長に関係なく 500 ステップ前後である．そしてこのうちレポート機能の explicit な定義に要する記述量は全体の 5% 以下である.

また I 8080 を例にとればシミュレータ作成時の生

表 4 シミュレーション速度比較

Table 4 Comparison of simulation speed.

ハードウェア記述言語	インプリメント言語	ターゲット・マシン	ホスト・マシン	測定単位	10Kステップ当り所要時間* (秒)	出典**
CDL	FORTRAN (インタプリタ)	PDP-8 I 8080	370/155	マシン命令 (平均)	2,500 15,000	16)
FALOS	— (インタプリタ)	大型マシン	TOSBAC 5600/170	マイクロ命令 (サンプル)	12,000	18)
LFM	PL/I (コンパイラ)	M6800	HITAC 8800/8700	マシンサイクル (平均) (マシン命令)***	267 1,068	17)
MHPL	SYSL-2 (コンパイラ)	I 8080	DIPS-1	マシン命令 (サンプル)	FL**** 53.50 NL 22.75	—

* 条件の異なる発表データを 10K ステップ当りに換算した.

** 他に文献 15) にデータの発表があるが条件が揃わないので割愛した

*** 1マシン命令≒4 マシン・サイクルとして換算した.

**** FL: フル・リスト・オプション, NL: ノン・リスト・オプション

* Program development system for Micro-Processors

産性は同一環境で個別に作成した場合に比べ約 20 倍の向上があった.

一方性能は表 4 に一例を示すようにこれまで報告されているシステムに対しホスト・マシンの差を考慮しても 10 倍以上の速度と考えられ,さらに同一システム上の専用シミュレータ (PMP-8 S)[23]に対しても 30% 減以内の性能であり,実用上問題の無いことが確認できた.

6. むすび

本論文では先ずハードウェア設計用シミュレータとプログラム・シミュレータの比較からプログラム・シミュレータ生成用言語設計時の 2 つの問題,すなわち,1) シミュレーション・レベルに合った記述レベルとプログラム構造, 2) レポート機能の定義機能, について明らかにした. 次いで従来の言語には存在しないかあるいは省みられなかった本問題解決のために, 前者に対しては拡張されたブロック構造と時間変数の導入, 後者にはパラメータ定義およびハードウェア定義中に埋め込んだレポート機能の定義という一つの方式を示した. そして本方式を MHPL としてインプリメントし, これを 16 種の μP に適用してその有効性を実証した.

MHPL によれば LSI 設計者ではない一般利用者によって標準的な μP を 500 ステップ前後で記述でき, また生成されるシミュレータは従来の個別作成のシミュレータに対しレポート機能, 処理速度において遜色が無い. この記述の容易さと生産性, 性能はプログラム・シミュレータ用言語を新たに設計することによって可能となったもので, 従来のハードウェア設計用言語の転用では実現し得なかったものである.

最後に, 日頃御指導頂く横須賀電気通信研究所戸田データ通信研究部長ならびに伊藤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 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く感謝します.

参考文献

1) Chu, Y., gest editor: special issue of "Hardware Description Languages", Computer, Vol. 7, No. 12 (1974).
2) Barbacci, M. R.: A Comparison of Register Transfer Languages for Describing Computer and Digital Systems, IEEE Trans. on computer Vol. C-24, No. 2, pp. 137-150 (1975).
3) Su, S. Y. H., symposium chairman: Proceedings of the 1975 International Symposium on C. H. D. L. & their Applications, p. 191 (Sept. 1975).
4) Su, S. Y. H., gest editor: special issue of "Hardware Description Language Applications", Computer, Vol. 10, No. 6 (1977).
5) Reed, I. S.: Symbolic design techniques applied to a generalized computer, Computer, (May/June, 1972).
6) Falkoff, A. D., Iverson, K. E., Saussenguth E. H.: A formal description of SYSTEM/360, IBM Systems Journal, Vol. 3, No. 3, pp. 198-262 (1964).
7) Brame, J. L., Ramamoorthy, C. V.: An interactive simulator generating system for small computers, SJCC, pp. 425-449 (1971).
8) Blaauw, G. A.: Digital System Implimentation, Prentice-Hall, p. 384 (1976).
9) Hill, F. J., Peterson, G. R.: Digital Systems; Hardware Organization and Design, John Willy & Sons, p. 481 (1973).
10) Chu, Y.: Introduction to Computer Organization, Prentice-Hall, p. 376 (1976).
11) Bell, C. G., Newell, A.: Computer Structure: Readings and Examples, McGraw-Hill, p. 668 (1971).
12) Breuer, M. A., editor: Digital System Design Automation: Languages, Simulation & Data Base, Computer Science Press, p. 417 (1975).
13) Kline, R. M.: Digital Computer Design, Prentice Hall, p. 429 (1977).
14) Mueller, R. A., Johnson, G. R.: A Generator for Microprocessors Assemblers and Simulators, Proc. of IEEE, Vol. 64, No. 6 (1976).
15) Smith, D. R.: Computer Structure Language, 文献 3) pp. 153-160.
16) Heath, J. R., Carroll, B. D., Cwik, T. T.: CDL-A Tool for Concurrent Hardware and Software Development?, Proc. 14 th Annual Design Automation Conference, pp. 445-449 (June. 1977).
17) 上森, 元岡: 論理装置の汎用機能シミュレータの性能評価, 52 年信学会情報部門全国大会 344.
18) 倉地, 足立, 末次: 逆ポーリッシュ形式を用いた機能シミュレータ (FALOS), 52 年信学会総合全国大会 1330.
19) 神谷, 小林, 仲谷, 藤田: シミュレータ用ハードウェア記述言語仕様の設計, 52 年情処学会第 18 回全国大会 218.
20) 藤田ほか: マイクロプロセッサ用汎用支援システム, 研究実用化報告第 27 巻第 4 号 (1978).
21) 高島: マイコン・プログラミング, 企画センター, p. 407 (1978).
22) 寺島ほか: DIPS-1 における高能率システム製造言語の実用化, 情報処理, Vol. 16, No. 6, pp. 492-498 (1975).
23) 宮口ほか: DEMOS-E マイクロプロセッサ用シミュレータ (PMP-8 S) の設計, 50 年情処学会第 16 回全国大会 286.

(昭和 54 年 5 月 21 日受付)
(昭和 55 年 2 月 8 日採録)