# "SPIDER" マイクロプログラム用シンボリックシミュレータ

松本偉和左，青山正義，古城隆，小林英晴，林孝雄
（日本電気株式会社）

## 1. はじめに

安価で高速なメモリの普及に伴ってマイクロプログラム制御方式がCPUの制御方式として広く使われる様になった。またOSやハードウェア診断機能など，従来ソフトウェアで実現していた機能の一部にマイクロプログラム技術が採用され，システムのスループットを向上させたり，よりきめ細かな機能を実現する事が可能になって来た。この様な傾向から近年マイクロプログラムの作成量は飛躍的に増大し，信頼性の高いマイクロプログラムを効率的に作成する事が重要な課題となった。（例えば，我々の所に於いても従来数キロステップ程度であったプログラム規模が10数キロになる事も珍しくは無くなった。）

マイクロプログラム作成のサポートシステムとしてマイクロプログラム用高級言語の試みがいくつか行われた。しかし主にマイクロプログラムに要求される実行効率の点から多くの言語はマイクロ命令のフィールドとの結びつきが強いアセンブラのレベルにとどまり，その記述能力は比較的低レベルである。この点がマイクロプログラムの生産性を低下させる原因の一つとなっている。

一方，マイクロプログラムに限らず一般にプログラムの正当性を検証するためにいくつかの具体的テストデータを用意し，それらによってシミュレーションを行なったり，実際にプログラムを実行した結果を調べる方法が従来広く行われて来た。しかしこの方法では正当性の充分な証明とはならない事も同時に指摘されて来た。これに対し，テストデータとして具体値を用いず出力結果を入力データの関数で表現するシンボリックシミュレータはプログラムの正当性を証明し得るものとして注目されている。

しかし，実際のシンボリックシミュレータでは，出力結果が人間に容易に理解できない複雑な表現になってしまう場合が多い，条件分岐において条件の判定が行えない場合が有る，等の問題点が有り現実のプログラム開発で実用的に使用した例は少なかった。これは従来のこの種のシミュレータ適用分野が高級言語で書かれたプログラムだった事が一因ではないかと考えられる。高級言語によるプログラムは比較的複雑な機能を実現しているため出力は単純な入力の関数としては表わせない事が多い，またプログラムの機能が単純な場合でも言語の表現力が比較的高いためシミュレータの出力表現とプログラムの表現に大きな差が出ない。

これに対しマイクロプログラムは比較的単純な機能を表現能力の低い言語で記述している場合が多い。このためマイクロプログラムにおけるシンボリックシミュレーションの結果は比較的単純で，出力結果とプログラム仕様の照合が容易な場合が多い。またプログラミング言語の表現能力が低いので単純な機能でもプログラムから機能を読み取る事が難しく，シミュレータの出力結果が重要な意味を持つ。

本稿では，これらの点に着目し作成したシンボリックシミュレータ SPIDER について述べる。本シミュレータは電子交換機用CPUのマイクロプログラム開発言語 HMIP（<u>H</u>igh level <u>Micro</u> <u>p</u>rogramming）専用シミュレータである。[1]
ちなみに，該CPUは16 bit/word のプロセッサであり，マイクロプログラムは

32 bit/word 構成である。

## 2. SPIDERを含むマイクロプログラムサポートシステムの構成

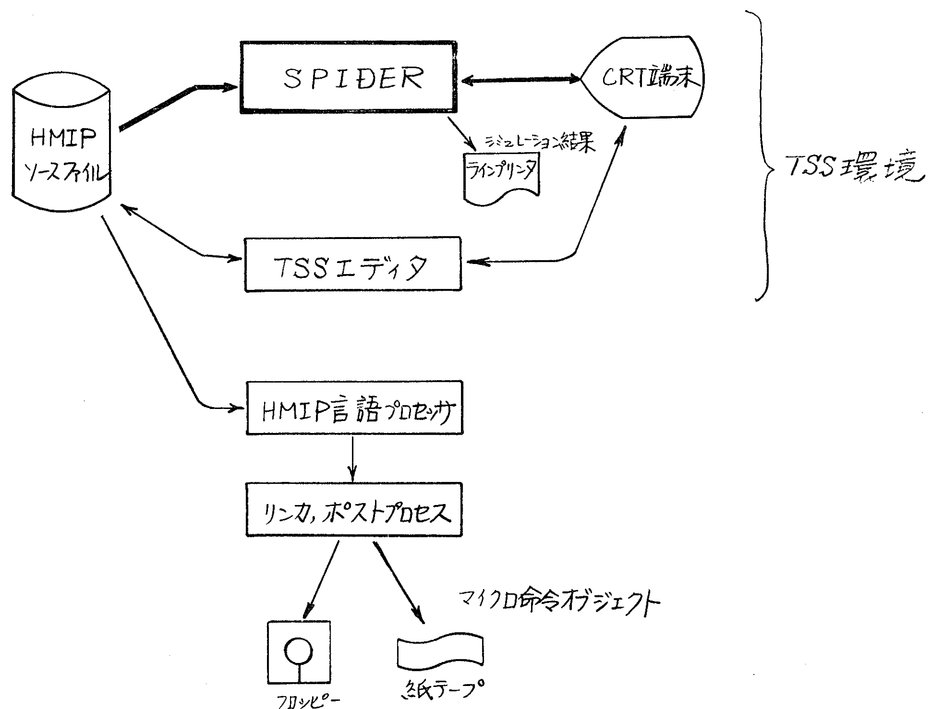
シンボリックシミュレータSPIDERは大型ホストコンピュータ上にHMIP言語プロセッサと共に実現されている。プログラムはB言語と呼ばれる無属性型の比較的低級なコンパイラによって約3キロステップで実現された。

本シミュレータはホストコンピュータのTSS環境のもとに，HMIPのソースステートメントをインタプリタ方式でシミュレートする。このためシミュレーション結果は即座にCRT端末に表示され，簡単に判定できる誤りはその場でTSSエディタにより修正し，再びシミュレートする事ができ迅速なデバッグ作業を可能にしている。また詳細な検討を要する時はラインプリンタに出力する事もできる。

こうしてできあがったプログラムソースは，そのままHMIP言語プロセッサの入力となり，マイクロ命令オブジェクトを得る事ができる。

図4.1にSPIDERを含むサポートシステムの構成を示す。

図4.1 サポートシステムの構成



## 3. HMIP言語

SPIDERのシミュレーション対象であるHMIP言語を簡単に説明する。

HMIPは電子交換機用CPUのマイクロプログラム用に開発されたマイクロプログラム記述言語である。本言語の各ステートメントはマイクロ命令に対応しており、この点ではアセンブラレベルの言語である。しかし各ステートメントの表現方法は、

① 演算，代入命令は

reg1 = reg2 + reg3

reg1 = SFT(reg2, RIGHT, N)

② 条件分岐は

IF flag GOTO label

の様に高級言語風のプログラマの直感にうったえる方法で記述できる。これは決っして任意の演算や分岐条件が書けるわけでなく，マイクロ命令に対応した範囲でしか書く事ができないから筆者らはこれを"高級言語風アセンブラ"と呼ぶ事にする。"高級言語風アセンブラ"のメリットは表現がプログラマの直感にうったえ，プログラムの理解を助ける点にある。実際，コーディング以降のプログラム作成効率を，アセンブラ言語と比較して30%程度改善する事ができた。

図3.1にHMIPによるプログラムの一例を示す。

図3.1 HMIPによるプログラム記述例

```
190 B     ENQ1:  IF BYTE GOTO ERRX
200 B            CALL SSENQ
210 A0           IRTN1

830 B     P1S040:CALL ADR
840 A0           MAP=R3+1
850              READ MAP,BR              /BR <-- NEW TAIL TCRB NO
860 A0           WAIT
870              LR=LR-1                  /LR=(SCC)
880 A0           MAP=R2
890              WRITE MAP                /(OLD TCRB) <-- NEW TAIL TCRB NO
900 A0    P1S041:WAIT
910              LR=LR+1                  /LR=(SCC)+1
920              RETURN
930*
940 A1    ADR:   LM=LR+1
950              READ MAI,BR
960 D            SCR=10
970 A0           WAIT
980              WO=SFT(R1,RIGHT,N)       /WO=P1
990 A0           MAP=BR
1000              READ MAP,BR              /BR <-- MSJ
1010 A2           MAI=MAP+1
1020              READ MAI,BR              /BR <-- ADDR1
1030 A0           WAIT
1040              WO=WO+WO                 /WO=P1*2
1050 A2           WAIT
1060              W1=BR                    /W1=MSJ
1070 A0           MAP=BR                   /MAP=ADDR1
1080 B            CALL $LPDCNR
1090 A0           MAP=BR
1100 A1           BR=SFT(R1,LEFT,I)
1110 A0           BR=SFT(BR,RIGHT,N)
1120 A1           SCR=WO
1130 A0           WO=BR+BR
1140 B            CALL $LPDR
1150 D            BR=OF#
1160 A1           BR=R1 AND BR             /BR=P3
1170 A0           BR=SFT(BR,LEFT,N)        /BR=P3*2**N
1180 D            SCR=5
1190 A1           IR=SFT(IR,LEFT,N)
1200 A1           IR=SFT(IR,RIGHT,N)       /IR=D1&07FF#
1210 A0           BR=IR+BR                 /BR=D1+P1*2**N
1220 A1           IR=SFT(BR,RIGHT,I)       /IR=DELTA PEA
1230 A0           BR=SFT(BR,LEFT,N)
1240 A0           BR=SFT(BR,RIGHT,N)       /BR=D2
1250 A1           R2=SFT(R2,RIGHT,I)       /
1260 A1           R2=SFT(R2,LEFT,I)        /MAKE LEA1*2**11
1270 A1           R2=R2+BR                 /R2=LEA1*2**11+D2 (DIST ADDR)
1280 A0           BR=SFT(R2,RIGHT,I)
1290 A2           MAI=W2+W2
1300 A0           MAP=MAI+BR
1310 A0           BR=SFT(IR,LEFT,N)        /BR=DELTA PEA*2**5
1320 A0           BR=WO+BR                 /BR=PEA1*2**5+XXX+(DELTA PEA*2**
1330 ZZ                      MSWR    RSR
```

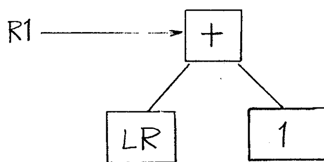
## 4. シンボリックシミュレータの実現

シンボリックシミュレータではシミュレーション中に値のわからないシンボルが出現するとそのシンボルを含む式の形で値を保持する。例えば4.1図の左に示す様なプログラムに於いてLRの値が既知でない場合はシミュレータは右の様な結果を出力する。すなわち才1行目ではLRの値はわからないのでそのままのシンボル名を用いR1はLR+1という値となる。次にR1はLR+1であるからR2はLR+1+1すなわちLR+2である事がわかる。

4.1図 簡単なシンボリックシミュレーション

| プログラム | 結果 |
| --- | --- |
| R1 = LR+1 | R1=LR+1 |
| R2 = R1+1 | R2=LR+2 |

4.2図 バリューツリーの例
(R1がLR+1の場合)




SPIDERはこれらの値を図4.2に示す様なツリーによって保持する。シミュレーションはこの様なツリーの追加，削除，コピー等の作業によって行われる。

マイクロプログラムにおいてはメモリへの参照はメモリアドレスレジスタの様なレジスタの間接指定で行なわれる。すなわちメモリアドレスもまた式で表現される。このためシミュレータはメモリデータを単なるメモリアレイとしてでは無くアドレス式とデータ式の組で保持しなければならない。もしメモリアクセスが行なわれると、シミュレータはそのアドレスがすでに登録されたアドレスかどうかすべてのアドレス式と照合する必要がある。また照合に際しては前にアクセスした時と今とでアドレス表現が完全に一致しているとは限らないから単純にツリーのノード同志を比較したのでは不充分である。このため本シミュレータでは式の等価性評価を行うために若干の式の変形を試みる一方，照合しやすい様に常に式をできる限り簡単化したり論理式を加法標準形で保持する様なくふうがほどこされている。

シミュレーション結果はプログラムの各ステップ毎と最終結果の二つが出力される。各ステップ毎の結果ではシンボルの値の出力の他にCALLやGOTOなどのシーケンスに関するもの，メモリ参照に関するもの等必要な情報が出力される。SPIDERの出力結果の例に関しては6項で示す。

## 5. シンボリックシミュレータの問題点と対策

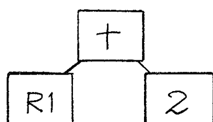
### 5.1 式の簡単化

シンボリックシミュレータによってプログラムをシミュレートして行くと演算が重なるたびに式は複雑になる．これを回避するためにシミュレータは常に式の簡単化を行う必要がある．また前述の様にメモリアクセスの際にはアドレス式同志の等価性評価を行なわねばならない．等価な式が等価であると評価されるためにはできる限り式が簡単化されていて標準的な形式になっている事が望ましい．例えば，

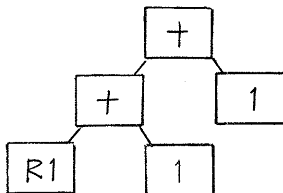
　　式：R1+2　と　式：R1+1+1

は人間にとっては簡単に等価である事がわかるがシミュレータにとっては左はノード３つのツリーであり右はノード５つのツリーであるため等価とみなすのは難しい。

式：R1+2 のツリー




式：R1+1+1 のツリー




SPIDER は以下の項目について式の簡単化を試みる．

(1) 式中に２つ以上の実値がある場合演算可能なら演算する．
　　例　R1+1+1　⇒　R1+2

(2) 結果が自明なもの(両辺が等しい減算，0とのAND等)を実値にする。
　　例　(R1+3)−(R1+3)　⇒　0

(3) ２段以上の同方向シフトをまとめる．
　　例　(R0 ≫ 5) ≫ 3　⇒　R0 ≫ 8

　　※式1 ≫ 式2 は式1を式2ビット右シフトする事を表わす．

(4) ２段以上の異方向シフトをシフトANDマスクとしてまとめる．
　　例　(R0 ≫ 5) ≪ 3　⇒ (R0 ≫ 2) AND 3FF8#

　　※3FF8#は16進数の3FF8を表わす．

(5) シフトANDマスク形式の式に更にシフトがあった場合，両辺それぞれにそのシフトをほどこしシフトANDマスクの形にする。
　　例 ((R0 ≫ 2) AND 3FF8#) ≪ 3
　　　　⇓　　　　---(5)の簡単化
　　((R0 ≫ 2) ≪ 3) AND (3FF8# ≪ 3)
　　　　⇓　　　　----(4)及び(1)の簡単化
　　((R0 ≪ 1) AND FFF8#) AND FFC0#
　　　　⇓　　　　----(1)の簡単化
　　(R0 ≪ 1) AND FFC0#

(6) ビット単位に簡単化を試みる

例えば RØ AND ØØØF# は、ビット単位に簡単化を試みれば、bit15〜bit4 はØ，bit3〜bitØ は RØ3〜RØØ である事がわかる。複雑な式に於いてはこの様なビット単位の簡単化を行なった後，加法標準形になおす事によって簡単化を行う。以下この手順の一例を示す。

(((RØ AND ØØFØ#) OR (R1 AND FØØØ#)) AND ØØFF#) + 1

① bit15〜Ø = { Ø Ø Ø Ø Ø Ø Ø Ø RØ RØ RØ RØ Ø Ø Ø Ø }
② bit15〜Ø = { R1 R1 R1 R1 Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø Ø }
③ bit15〜Ø = { R1 R1 R1 R1 Ø Ø Ø Ø RØ RØ RØ RØ Ø Ø Ø Ø }
④ bit15〜Ø = { Ø Ø Ø Ø Ø Ø Ø Ø RØ RØ RØ RØ Ø Ø Ø Ø }
⑤ bit15〜Ø = { Ø Ø Ø Ø Ø Ø Ø Ø RØ RØ RØ RØ Ø Ø Ø 1 }

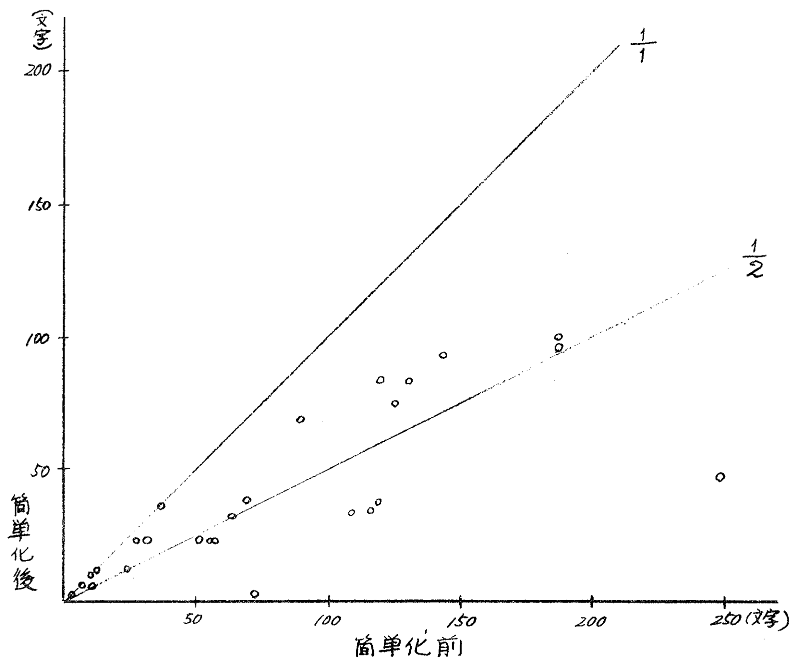
⑤の結果から式の値は

(RØ AND ØØFØ#) OR ØØØ1#

となる。

図5.1 式表現簡単化の効果

図5.1 に本シミュレータの簡単化の効果を示す。横軸は簡単化前の式の長さ(文字数)縦軸は簡単化後の式の長さをとり，あるプログラムのシミュレーション結果の各式表現をプロットした。(平均して 1/2 程度)



## 5.2 条件分岐の判定

シンボリックシミュレータでは条件分岐の条件もまたシンボル式で保持している。このため分岐点でどのルートをとるかシミュレータ自身が決定できない場合がある。その様な場合，SPIDER は TSS 環境下で動くのでユーザの端末からの指示によりルートを決定する。この時すべての可能なルートをシミュレートする様指示する事もできる。

シミュレーションを正確に行うためには，分岐点で設定された条件を以降のシミュレーションに生かす方が望ましいが SPIDER に於いては実現されていない。

例えば，

```
*=RQ
SET STF          }--- RQの値でコンディションフラグをセットする。
IF ZERO GOTO CASEQ
```

の場合，CASEQ 以降のシミュレーションでは RQ=Q（RQの値が R1−R2 だった場合は R1=R2）を仮定して良いが，SPIDER この条件を考慮していない。

SPIDER が対象とするマイクロプログラムでは複雑な条件分岐が少いためか，あまり大きな問題とならなかった。

## 6. 使用例

本シミュレータの使用例を2つ示す。前者は本シミュレータの基本的な機能を示すための簡単な例，後者は実際のプログラムにおける本シミュレータの能力を示すための例で，紙面の都合上出力結果の一部のみを示す。

図6.1 は HMIP の簡単なソースプログラムの例である。RQ〜R3，BR および MAP はマイクロプログラムで使用できる 16 bit のレジスタである。代入文は通常の代入文と同様である。WRITE 文は MAP の示すメモリアドレスに BR の内容を書込むものである。実際のメモリライトは WAIT 文の直後に完結する。図6.2 は図6.1 のプログラムを SPIDER でシミュレートした結果のリストである。① は各ステップ毎の実行結果でソースの各ステートメントに対応する。② は全ステップ実行後の各レジスタの値が式で示されたものである。BR や MAP の結果で値のわからない部分はレジスタ名で示され実値は可能な限り計算されている事がわかる。③ は メモリの値でこの例では R3+5 番地に R1+15 が書かれた事がわかる。また初期値としてシステムが自動的に #MEMQ1 なるシンボルを割当てるので R3+5 番地のもともと持っていた値がこわされた事がわかる。尚，③で PEA とあるのは本シミュレータのターゲットマシンが特殊なセグメントレジスタを利用してメモリをアクセスしているためにそのレジスタの値も同時に表示したものである。

図6.3 はより実際的なシミュレーションの例である。この例は図6.4 に示す様なメモリバッファのエンキューを実行するマイクロプログラムの実行結果の一部である。キューは R3 レジスタで指されるメモリの3ワードに Head バッファのポインタ，Tail バッファのポインタ，キュー内のバッファ数が格納されている。エンキュー後①の Tail バッファのポインタはエンキューすべきバッファのポインタ（R1で示される。）に入れ替り，②のバッファ数カウンタはプラス1されている事が容易にわかる。

図6.1 簡単なHMIPソースの例

```
0010 A0   START: R0 = 10
0020 A0          R2 = R0 + 5
0030 A0          BR = R1 + R2
0040 A0          MAP = R3 + 5
0050             WRITE MAP
0060 A0          WAIT
0070 A0          IRTN0
```

図6.2 図6.1のシミュレーション結果

```
       HIMIP-71 SIMULATION LIST

0000  *** GOTO START ***
0010  R0 = 10
0020  R2 = 15
0030  BR = R1 + 15
0040  MAP = R3 + 5
0050  *** WRITE MAP ***
0060  *** WAIT ***
0070  VALUE : R1 + 15
0070  *** IRTN0 ***
```

①各ステップの実行結果

```
     ###  VALUE LIST ###

>>> REMAINED MEMORY R/W REQUEST

 NO REQUEST

>>> REGISTER VALUE
BR     = R1 + 15
MAP    = R3 + 5
MCR    = MCR
R0     = 10
R1     = R1
R2     = 15
R3     = R3
```

②実行後の各レジスタの値

```
>>> MEMORY VALUE

PEA : #MS001 AND FFE0#
DISP: (R3 + 5) AND 07FF#
    INITIAL: #MEM01
    FINAL  : R1 + 15
```

③メモリの値

```
>>> MS VALUE

MS NO  : MCR AND 01F0#
MS ADDR: (R3 + 5) AND F800#
    INITIAL: #MS001
```

図6.3 エンキュープログラムシミュレーション結果（抜粋）

```
R1     = TAIL
R2     = (((#MEM14 AND 07FF#) + ((TAIL AND 000F#) <
R3     = R3
SCR    = 5
TAIL   = TAIL
TCBPEA = TCBPEA
TTCBN  = TTCBN
W0     = #MEM13
W1     = MSJ
W2     = MCR AND 01F0#
ZERO   = ZERO(CNT)

>>> MEMORY VALUE

PEA : #MS001 AND FFE0#
DISP: (LR + 1) AND 07FF#
    INITIAL: ADR

PEA : #MS002 AND FFE0#
DISP: R3 AND 07FF#
    INITIAL: HEAD
```

HEADの値

```
PEA : #MS003 AND FFE0#
DISP: (R3 + 1) AND 07FF#
    INITIAL: TAIL
    FINAL  : R1
```

① TAILの値

```
PEA : #MS004 AND FFE0#
DISP: (R3 + 2) AND 07FF#
    INITIAL: CNT
    FINAL  : CNT + 1
```

② CNTの値

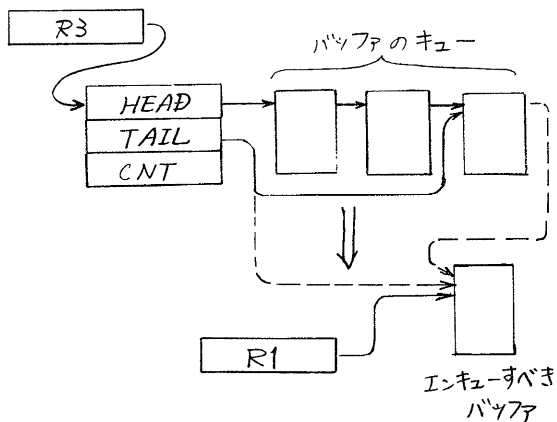
```
PEA : #MS005 AND FFE0#
DISP: ADR AND 07FF#
    INITIAL: MSJ

PEA : #MS006 AND FFE0#
DISP: (ADR + 1) AND 07FF#
    INITIAL: H1TBLA

PEA : #MS007 AND FFE0#
DISP: (H1TBLA + ((TTCBN >> 9) AND 007E#)) AND 07FF#
    INITIAL: N

PEA : #MS008 AND FFE0#
DISP: (H1TBLA + ((TTCBN >> 9) AND 007E#) + 1) AND 07
    INITIAL: H2TBLA
```

図6.4 メモリバッファのエンキュー操作



## 7. まとめ

先に述べた様にマイクロプログラムは比較的単純な機能を実現しているに過ぎない。それでもなお充分な式の簡単化を行わねば実用に耐え得るシミュレータとはならない。マイクロプログラム内では特に1ワード内の部分ビットフィールドを扱う様な論理演算が極めて頻繁に行なわれる。このため本システムでは論理演算に対して特に強力な簡単化が行なわれる様に作られ効果をあげる事ができた。

シンボリックシミュレータは

(1) プログラムの正当性の証明が実値よりも容易

(2) テストデータを用意する必要が無い

などの特長を持ち，今後マイクロプログラムの開発が多くなるにつれその有効性が更に明らかとなるであろう。

[参考文献]

<1> 山田 他．"マイクロプログラム記述の汎用化の一手法"
電子装置設計技術研究会資料 4-2