マイクロプログラムシミュレータMIPSについて

萩原宏
(京都大学)
石田勝則　永井靖　淀照隆
(日本アビオトロニクス(株))

1. はじめに

マイクロプログラム方式は，1951年に英国のケンブリッジ大学のM.V.Wilkesによって提案されて以来，多くの中大型電子計算機の論理装置に応用され，近年ではマイクロコンピュータからミニコン，さらに周辺制御装置，端末装置にいたるまで，その応用範囲はますます広範なものになりつつある。

こうした，マイクロプログラミングの発展に伴い，マイクロプログラムの作成は重要な問題となって来た。特にマイクロプログラムが作成される対象となるハードウエアが設計段階にある場合にはマイクロプログラムとハードウェアのトレードオフの問題が生じ，ハードウェアの設計にフィードバックをかけるためのシミュレータが要望される。又ハードウェアにマイクロプログラムを書込んで，動作させる前にシミュレータによってテストを行って，充分デバッグを行っておきたいという要望が生じる。従来はこうした要望に対しては，各ハードウェア毎にマイクロプログラムの仕様が決められ，専用のシミュレーターが必要に応じて作成されていた。MIPSシミュレータは，こうした背景で汎用マイクロプログラム制御計算機記述言語及び汎用マイクロプログラムアッセンブラ言語を開発し，シミュレーションができる汎用性のあるマイクロプログラム作成援護システムとして開発された。

MIPSシステムは次の構成から成立っている。

MIPSシステム
- MIPS言語
- マイクロアッセンブラ
- MIPSシミュレータ

MIPSシステムの特徴は

(1) シミュレーションはマイクロプログラムの機能レベルで行われる。
(2) レジスタ等の初期値の設定ができる。
(3) 中間結果を外部記憶装置にダンプし、再びロードしシミュレーションを再開することができる。
(4) シミュレーション時にマイクロプログラムの簡単な修正ができる。
(5) シミュレーションの停止条件をレジスタ値，アドレス値，時間，によって与えることができる。
(6) 強力なロギング機能をもっている。
(7) 豊富なTrace機能をもっている。
(8) 非同期処理，割込処理を効率よく処理することができる。
(9) I/O入出力機能のシミュレーションを効率よく行わせることができる。
(10) 処理方式は処理速度を考慮したコンパイル方式をとっている。

MIPS言語の特徴は

(1) 記述システムは計算機の設計技術者を対象としている。
(2) マイクロプログラム制御計算機が，見通しよく，記述でき，ドキュメント性が良い。
(3) 広範なマイクロプログラム制御計算機が記述できる。
(4) 組合せ回路レベル，順序回路レベル，Register to Register Transfer レベル

高級言語レベルまで種々の設計段階での論理記述を可能にしている。

(5) 制御の対象となる論理機能をBlock 3にまとめることができる。

(6) タイミング記述がDecoder - Blockのタイミングリング及びFacility - Blockの Block 3のいずれでも記述できる。

(7) 構造が静的な論理回路を記述するFacility-Block，マイクロプログラムの解読を行うDecoder - Block，入出力機能，割込処理，非同期処理を記述するEveut - Blockに分れており全体が容易に見通せる。

(8) 多段Decoderが記述できる。

(9) 入出力インタフェイスの記述が簡単にできる。

マイクロアッセンブラの特徴

(1) マイクロアッセンブラの仕様を定義することにより，どのようなビット構成のマイクロプログラムも記述することができる。

(2) 構成は μ-specification 部と μ-program 部よりなる。

(3) ビットパターンにアッセンブルされたマイクロプログラムは LOAD コマンドにより Control - Memory に読込まれる。

Fig 1 にMIPS システムのゼネラルフロー を示す。MIPS シミュレータは IBM360 - 195 のOSの下でバッチ処理で動作し，PL/I言語にてインプリメントされている。システム全体の流れは，まずターゲットマシンをMIPS 言語で記述した入力データをトランスレーションセクションで PL/I ソースプログラムに翻訳する。コンパイルセクションでは翻訳されたPL/I ソースプログラムとシミュレーションをコントロールするPL/I言語で記述されたシミュレータコントロールを合せてコンパイルを実施し，実行型式プログラムを出力する。シンボリックマイクロプログラムはマイクロアッセンブリセクションで Bit - Pattern に変換されシミュレーションセクションで コントロールメモリに読込まれて実行される。

2. MIPS 言語

Machine Struveture Description言語としてのMIPS 言語はLDS言語の系列をくむもので，Facility - BlockはLDS言語を母体としている。特にマイクロプログラム制御計算機の場合，制御信号はマイクロプログラムをデコードすることにより取り出される点にあり，Decoder - Blockとして独立させた。又 I/O入出力等外部インター フェイスを記述するために Event - Blockを設けた。

```
MIPS言語 :: = MIPS      facility - block
                        decoder - block
                        event - block    MIPS - END
```

2.1 Facility - Block

Facility - Block はターゲットマシンの構成要素の宣言部とBlock 3から構成される。 Block 3 は階層構造をとることはできない。Faility - Blockは静的な構造を記述するもので実行順序の制御，初期条件の制御，動作時間の指定，非同期な事象の制御等はDecoder - Block 及び Eveut - Block によって記述される。

```
Facility - Block :: = F - Block
                        declaration
                        block 3 [block3]····
                      F - END
```

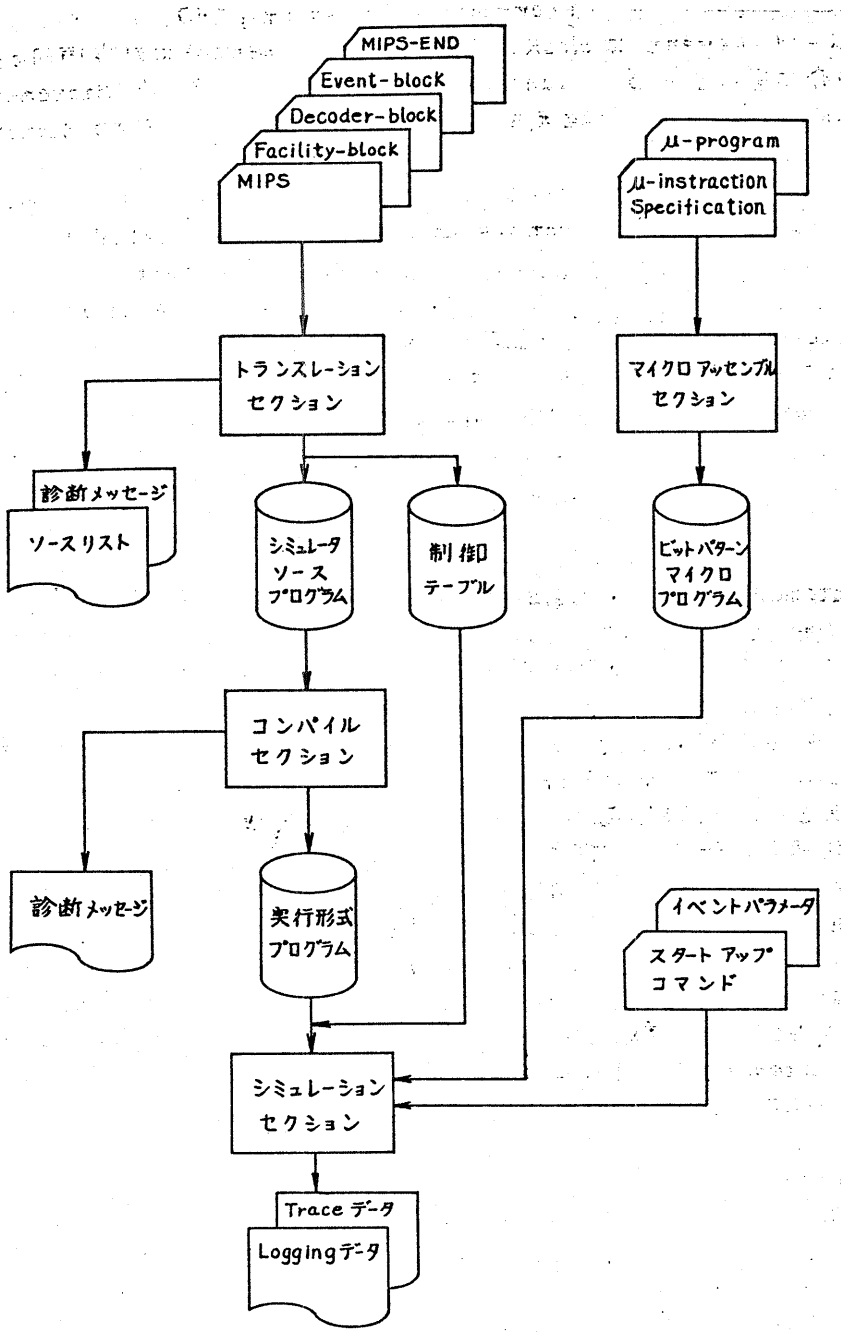

FIG. 1 General Flow of MIPS System

block 3 :: = Label : LEVEL { basic - statement [$ basic - statement]····
| compound - statement } END

Compound - statement は block 3 の中で各 statement 毎に処理時間のタイミング記述を行う場合に用いられる。Transition - part は次に実行する Statement の Label を示し、静的な順序動作の構造を示す。Timing - description はその Statement の実行時間を示す。

Compound - Statement :: = concurrent - Statement [Concurrent-Statement]
Concurrent - Statement :: = Label : [basic - Statement 群]
: transition - part
[: timing - description]

basic - statement は時間に関する記述は含まれない。ある時点で実行される一連の論理操作を記述したもので、assign-statement, module-Statement, wait/post - Statement から構成される。

basic - Statement :: = left - part - valiable - list { : = | = }
boolean - expression [if - Statement] |
module - Statement [if - Statement] |
wait - Statement |
post - Statement [if - Statement]

assign - statement 中の := は register, memory 等遅延時間のあるデータ転送を示し、= は Buss - line, terminal の如く、遅延時間のない転送を示す。

ターゲットマシンを記述するために Facility - Block で宣言することができる要素は Terminal, Register, Memory, Function, Module である。Terminal とは register 又は memory の出力あるいは、複数個の Terminal を変数として何らかの論理演算を行った信号線のことである。Terminal の値は Concurrent - Statement の時間間隔内又は Block 3 の時間間隔内でのみ有効である。Register は値を新しく設定しない限り前の状態を保持する変数である。Function はハードウエアのまとまった論理機能を表わしており、入出力の関係が、遅延時間なしで決定されるもので UNIT で宣言する。Module には入出力の関係が遅延時間 0 できまってしまう PASS と入出力の関係に遅延時間のある ELEMENT と Standard - Module で構成される。
Function では Function - Body を PL/1 の code で記述することができる attrifute description がゆるされている。Standard - Module としては RSFF, RSTFF, JKFF, TFF, DFF, Delay. integer がゆるされている。又 Standard - function として AND, NAND, OR, NOR がゆるされている。

2.2 Decoder - Block

Decoder - Block はマイクロインストラクションの記憶場所であるコントロールメモリとマイクロインストラクションレジスタ及びコントロールメモリーの読出し時間を指定する Environment. Description とデコーダの構造及びマイクロインストラクションの解読の結果どの Block 3 を起動するかを記述する部分とどのタイミングで Block 3 を起動又は停止するかを記述する Decoder - description から構成される。

Decoder - Block :: = D - BLOCK environment - description
decoder - description D - END
environment - description :: = ENV
MEMORY = memory - name ;
INST = register - name ;

FETCH-TIME = intiger; END

Decoder-description は decoder の階層構造を記述する decoder-description-part-1 とマイクロインストラクションを解読のフィールド等に分ける field-declaration, マイクロインストラクションが動作するタイミングリングと Block 3 が起動するタイミングポイントを記述する timing-declaration 及び field-declaration で define された Field-name, timing-declaration で define された Timing Point Name を使用して, decode の結果どの Block 3 がどのタイミングポイントで起動し, 停止するかを記述する decode-expression から構成される decode-description-part-1 から成り立っている。

```
decoder-description ::= {decoder-description-part-1 |
                         decoder-description-part-2 }
decoder-description-part-1
                    ::= DECODER decoder-name
                        field-declaration
                        timing-declaration
                        decode-expression   END
decoder-description-part-2
                    ::= DECODER decoder-name
                        select-statement
                        decoder-description[decoder-description]
                        ···· END
select-statement    ::= SEL register-name [(Subscript)]
                        input-value /decoder-name,·····;
field-declaration   ::= FIELD
                        field-declaration-part,·····;
timing-declaration  ::= TIMING
                        timing-declaration-part,·····;
decode-expression   ::= decode-expression-part |
                        intiate-expression-part
decode-expression-part ::= DECODE
                           field-name ;timing-expresion : output-
                           element-list $·····
intiate-expression-part ::= INIT
                            timing-expression : output-name····
```

timing-expression は time-name, [time-name] で実行開始タイムポイントと終了タイムポイントを指定し, output-name で起動すべき Block 3 のレベルを指定する。又 output-element は, Field-name で指定された マイクロオーダーの Bit-String に対応して起動される Block 3 を input-value / output-name の形式で記述する。

デコーダの制御としてはデコーダの起動と停止がある。前者は Eveut-Block の FETCH文により制御され, 後者は Facility Block における WAIT文により実行される。FETCH文の実行は, イベントパラメータカードを読込ませるか, Block 3 からイベントを POST文により起動させることにより行われる。FETCH文を実行するとシミュレータは enviropment-description において指定されたコントロールメモリより Facility-Block で宣言されたコントロールメモリアドレスレジスタの指定する番地の内容を指定されたマイクロインストラクションレジスタに格納する。格納が終了するまでの時間は FETCH-TIME で指定された integer のユニットタイムの倍数である。

2.3 Event-Block

Event-blockはターゲットマシンの外部からFacility-Block又はDecoder-Blockを制御するとき，Block3からBlock3又はDecoder-Blockを制御するときあるいは，ターゲットマシンにおける入出力機能を模擬する必要がある場合に使用する。イベントによりDecoder-Blockを制御する場合としてFETCH文の実行がある。又イベントによりFacility-Block内のBlock3を起動させるときにはINIT文を使用する。Block3から直接マイクロインストラクションの読出しを指示することはできないのでイベント経由で行う。

一般にターゲットマシンの入出力機能は複雑である。MIPS言語では，このためEvent-BlockにINPUT文とOUTPUT文をもうけている。INPUT文では予め用意した外部記憶装置内に格納されている情報をFacility-Blockで指定されたレジスタへ入力する。

Output文では，Facility-Block内のレジスタに格納されている情報がラインプリンタから出力される。これらの入出力動作ではシミュレーション上の時間はかからない。

```
Event-Block ::= E-Block
                    event-declaration [event-declaration]·····
                    E-END
event-declaration ::= EVENT : event-name ; event-body  END
event-body ::= [post-condition] [delay-time]
                    event-command [$ event-command]·····;
event-command ::= initiate-command | decoder-command |
                    input-command | output-command
intiate-command ::= INIT  label
decoder-command ::= FETCH | TERM
input-command ::= INPUT register-name [(subscript)]
                    FILE = file-name
Output-command ::= Output register-name [(Subscript)]
```

intiate-commandはBlock3をイベントカードまたはBlock3のPost文によって起動する場合に，又decoder-commandはイベントカードまたはBlock3のPost文によってデコーダの制御を行う場合に，input/output commandは模擬入出力動作を行わせる場合に各々用いられる。event-bodyのpost-conditionは上記動作を行わせるための条件設定を行うstatamentであり，Block3からPost文でイベント制御を行う場合には必ずevent-codeを用いて記述する必要がある。

```
Post-condition ::= {AND(post-code-list) | OR(post-code-list)};
```

又delay-timeはPost-conditionが成立してから一定の時刻を経過した後，各Commandを実行させたい場合に指定する。

3. MIPS言語による記述例

Fig.2にMIPS言語による記述例を示す。9bitからなるマイクロインストラクションを想定した例題コンピュータであり，AL，AR，AF，AD，TEST，IOの6つのフィールドから構成されている。T1のタイミングでLIOA，RIOA，AFUH，IOIN，CMADのBlock3が起動される。T2のタイミングではDNOA，又TESTフィールドがdecodeされて00，01，10，11の値により NOP，ALE，POS，NEGのいずれかのBlock3が起動される。それぞれの終了タイミングはT3，T4である。UNIT Functionのbodyは Attribute-decriptionで記述されている。NOP，ALE，NEGのBlock3ではEvent-code-10がPostされFETCHコマンドがコールされている。

```
MIPS
  F-BlocK
   TERMINAL E(8), F(8), G(8);
   REGISTER A(8), B(8), C(8), D(8),
            CMB(9), CMAD(4);
   MEMORY   CM(9,16,CMAD(0:3));
   UNIT  ADD(16)(U(16),L(16))
       ATTER
        DCL ADD BIT(16);
        ADD = U + L;
        RETURN (ADD);
       ATTER-END;
  AFBI : LEVEL
        E = A IF ¬CMB(0)$
        E = B IF  CMB(0)$
        F= C IF ¬CMB(1)$
        F= D IF  CMB(1)$
        G = E & F IF(¬CMB(2)&¬CMB(3))$
        G = E IF  IF(¬CMB(2)& CMB(3))$
        G= ADD(E,F)IF(CMB(2)&¬CMB(3))$
        G=  E  IF(¬CMB(2)&¬CMB(3))
        END
  DNOA : LEVEL
        A:=G  IF(¬CMB(4)&¬CMB(5))$
        B:=G  IF(¬CMB(4)& CMB(5))$
        C:=G  IF( CMB(4)&¬CMB(5))$
        D:=G  IF( CMB(4)& CMB(5))$
        END
  CMAD : LEVEL
        LA1: CMAD(4):= ADD(10)(CMAD(4),1);
        STOP:5
        END
  NOP: LEVEL
        LQ : POST(10): STOP:5
        END
  ALZ: LEVEL
        L11 : L12(¬G), L13(G): 1'
        L12 : CMAD(4) = ADD(16)(CMAD(4),1):
              L13 : 15'
        L13 : POST(10): STOP: 4
        END
```

FIG. 2

```
POS:LEVEL
     L21:: L22 (¬G(0)),L23(G(0)):1'
     L22 : CMAD(4):=ADD(16)(CMAD(4),1):
     L23 : 15'
     L23 : POST(16): STOP:4
     END
NEG: LEVEL
     L31:: L32(G(0)),L33(¬G(0)):1'
     L32: CMAD(G):ADD(16)(CMAD(4),1):
          L33 : 15'
     L33: POST(16): STOP:4
     END
IOIN : LEVEL
     L41: POST(20): STOP:5
     END
F-END
D-BLOCK
  ENV
     MEMORY = CM;
     INST = CMB;
     FETCH-TIME = 5; END
DECODER DEC
     FIELD AL(0),AR(1),AF(2:3)
           AD(4:5),TEST(6:7),IO(8);
     TIMING 30,
       T1=5, T2=10, T3=15, T4=30,
DECODE
     TEST: T2; T4: 00/NOP,
                   01/ALZ,
                   10/POS,
                   11/NEG; END
INIT  T1,T3:LIOA $
      T1, T3: RIOA $
      T1, T3: AFUH $
      T2,T3 : DNOA $
      T1: IOIN $ T1,T2 : CMAD;
D-END
E-BLOCK EVENT:EVENTA;AND(10)
        FETCH;END EVENT;EVENTB
       ;AND(20)INPUT D(8)FILE =
        CRFIRE; END
E-END
MIPS-END
```

4. MIPS シミュレータ

MIPS 言語で記述されたターゲットマシンはトランスレーションセクションでまず処理される。トランスレーションセクションに於ける処理は Facility-Block の Statement 部を PL/I の Statement に変換することと，シミュレーションをコントロールするシミュレータコントロール機能のソースプログラムを組込むことと，Facility-Block の宣言文及び Decoder-Block, Event-Block を各々次の表に変換することである。

- Facility-Block
  - Module-table
  - State-table
  - Facility-table
- Decoder-Block
  - Decoder-table
  - Decoder-element-table
- Event-Block
  - Post-code-table
  - Event-table
  - Event-command-table

Module 又は Block 3 に対応して各々 module-table が成生される。
トランスレーションセクションで PL/I ソースデータに変換されたターゲットマシンは，シミュレーションを制御するシミュレーションコントロールと組合わされてシミュレーションプログラムを完成する。

シミュレーションコントロールには次の 5 つの機能ブロックがあり event-Block 制御，decoder-block 制御，Facility-Block 制御は，前もってコンパイルされており，リンケージの時点で組込まれる。

(1) スタートアップの制御
(2) シミュレーション実行制御
(3) event-block の制御
(4) decoder-block の制御
(5) facility-block の制御

これらの関係を Fig.3 に示す。

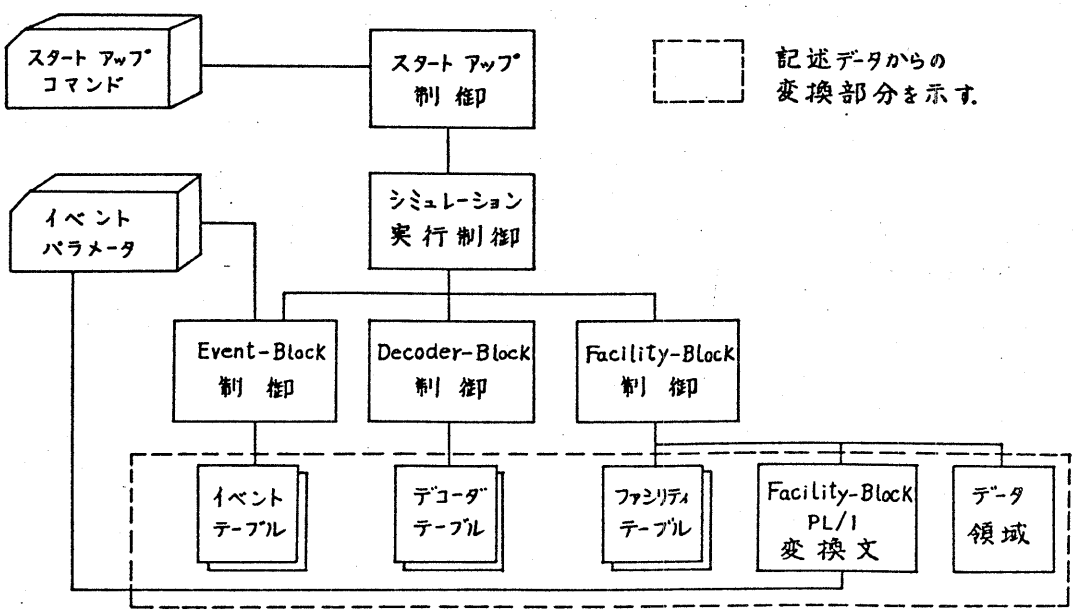

FIG. 3

4.1 スタートアップ制御

スタートアップ制御ではシミュレーションランの初期値,初期条件の設定を行ないます。初期条件の設定が終るとシミュレーション実行制御へコントロールをわたします。

4.2 シミュレーション実行制御

シミュレーション実行制御ではシミュレーション時刻の設定,シミュレーションの進行管理を行います。

4.3 Event-Blockの制御

Event-Blockの制御ではイベントパラメータカードの処理, Post文によるイベントの起動,イベント制御対象の処理を行います。

4.4 Decoder-Blockの制御

Decoder-Blockの制御はマイクロインストラクションの読出し,タイミングリングの開始と終了,タイミングポイントの開始と終了を行います。

4.5 Facility-Blockの制御

Facility-Blockの制御は起動される moduleはModuletable内でリンクがとられActive-listを構成しており これらの Active-listの制御, Stata-tableにリンクされたBasic-statementの実行, transition-part, timing-discriptionの評価,実行終了時の処理等を行います。処理のフローを Fig. 4 に示します。又MIPSシミュレータのRUNの結果を Fig.5 に示します。

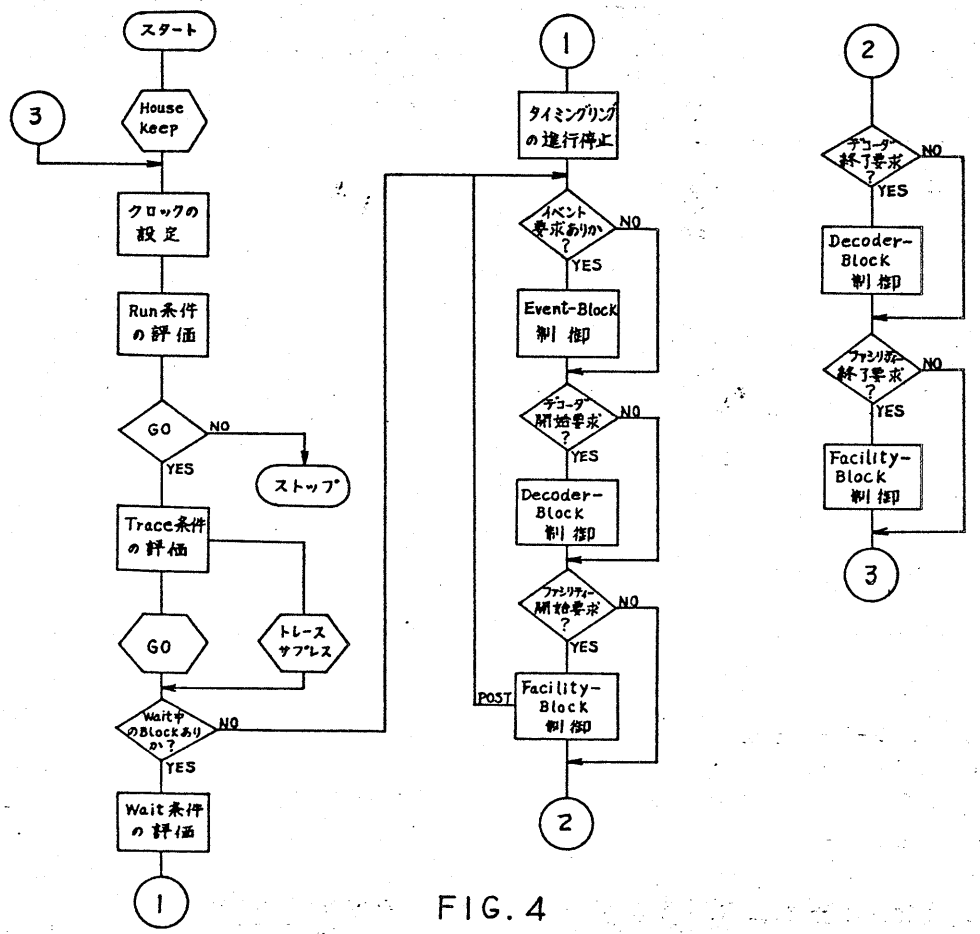

FIG. 4

```
                    **** MIPS  SIMULATION ****

SIM001      SIMULATION START.

***TIME= 0 ***

SIM003      TRACE BEGIN.

***TIME= 2 ***

LOG001      POSTED EVENT : 'START'(C)(2)

***TIME= 12 ***

LOG002      EXECUTE EVENT : 'START'
                            INIT:  RUNB
TRACE       MOD = RUNB : L1
LOG003      POST(10)
LOG001      POSTED EVENT : 'FETCH'(P)
LOG002      EXECUTE EVENT : 'FETCH'
                            FETCH
TRACE       FCL = S
            VAL = '10000000'B
TRACE       MOD = RUNB : STOP

***TIME= 13 ***

LOG010      START OF MICRO INSTRUCTION.
                  COM(0) = 01000CC000000000100'B
                  SELECT(RRT)
                  TIMING : T0(0) =
                  TIMING : T1(1) = CAB0(CA), READ(RW), INIT2
                  TIMING : T2(2) = INIT4
                  TIMING : T3(3) = INIT3
                  TIMING : T4(4) =
LOG011      TIMING POINT : T0(13)

***TIME= 14 ***

LOG011      TIMING POINT : T1(14)
                            INIT:  CAB0
                            INIT:  READ
                            INIT:  INIT2
TRACE       FCL = CMAD
            VAL = '000001'B
TRACE       MOD = INIT2 : STOP

***TIME= 15 ***

LOG011      TIMING POINT : T2(15)
                            INIT:  INIT4
TRACE       MOD = CAB0 : STOP
TRACE       FCL = M
            VAL = '00000000'B
TRACE       MOD = READ : STOP
TRACE       MOD = INIT4 : STOP

***TIME= 16 ***
```

FIG.5

5. むすび

本シミュレータの性能については、まだ判断できないが今後本格的なシミュレーションを実施し明らかにしていく必要がある。最後に本シミュレータの開発の機会を与えられた日本電子工業振興協会並びに御指導いただいた東京大学元岡教授，当協会論理設計自動化専門委員会マイクロプログラム分科会の委員の方々に深謝する。

参考文献

(1)論理設計の自動化に関する研究 - 第3報 - 日本電子工業振興協会