# 14. マイクロ Plan エンジン

東京大学 工学部

和田英一 戸村 哲 梅村恭司 寺田 実
Eiiti Wada Satoru Tomura Kyoji Umemura Minoru Terada

これまでミニコン，マイコンの上に処理系を作製してきたプログラム言語 Plan を，学生の実習，研究のためと，将来の LSI 化の可能性の調査のためとから，マイクロ Plan 計算機に再構成，再設計しているので，それについて報告したい。

プログラム言語のハードウェア化について

最近のマイクロコンピュータや VLSI 技術の発展は，プログラム言語やソフトウェア工学の研究者にとっても，その哲学に最も適したハードウェアやアーキテクチャが容易に得られるようになったということで福音であり，これまで以上にソフトウェアからの意見をハードウェア，アーキテクチャにフィードバックすることは重要になってくると思われる．特に個人用計算機の分野では，ごく限られた種類のプログラム言語しか使用しないと考えられるから，プログラム言語により密着したハードウェアを構成して，プログラムの作製，検査，実行の能率をあげることができる。たとえば配列の添字の範囲のチェックなど，ひんぱんに行なわざるをえないものは，ハードウェアで監視するのがよいように思う。また抽象的データ型の忠実な実現にも，適切なハードウェアのサポートのあることが望ましい．

ソフトウェア専攻学生の研究対象とする意義

これまで，この種のプログラム言語計算機の研究は主としてハードウェアに関係のある学科で実施されてきた．それはソフトウェアの方からはプロ級のハードウェアには仲々手がだせないからであった．ところが，出来あがった言語計算機はソフトウェアの側からみると，プログラム言語の扱いがまったく未熟であるという例が少なくなかった．また現実に，プログラム言語計算機があまり実用化されなかったという事態は，人間とのインターフェースとしてのソフトウェアの考案が充分でなかったためと思われる．そこでこの境界領域はハード，ソフト両サイドからの研究が望ましいが，最近では LSI の出現によりソフトウェア側からの研究がかなり容易になったように見える．さらに VLSI にしても，その設計技法が Mead, Conway [1] にあるように，あまり最先端の性能を望まないならば，基本的な設計手法の組み合わせに還元できるようになりつつあり，ソフトウェアの守備範囲に近づいていると思われる．さらに従来のソフトウェアの機能も次第に LSI の機能の中にとり入れられつつある．そして今後はそのような LSI を基本にした論理回路，あるいは論理回路そのものを実現する VLSI の設計のできることが，ソフトウェア屋の主要な作業のひとつとなると予想されるので，ソフトウェアを専攻する学生にこの種の研究をさせ，経験をつませておくのはきわめて大切と思われる．

プログラム言語 Plan

実際にハードウェア化するプログラム言語に何を使うかには選択の自由がありすぎるが，今回は当研究室で1973年ごろから手がけてきた Plan をもう一度用いることにした．

Plan は最初，ミニコンによる Fortran 処理系作製コンテストの経験を基にし，その Algol 版のような形で，当研究室のミニコン MACC／7 上に武市が開発した．これはアセンブラによって記述されていたが， Plan による処理系の記述も試みられた [2]．ミニコンの Plan は Hitac-10 へも移植され，前野により性能の改善が実施された [3]．

1977年ごろになってマイコンが使えるようになると，再び Plan 処理系の検討を行ない，戸村 [4] の作製したシステムが研究室の8080系マイコン上で動くようになった．1979年にはマイクロ Plan による会話型コマンド処理システム (PLIS：PLan Interactive System) を松井，岡本 [5] が6800系のマイコン上にあらたに開発し，マイクロ Plan はプログラム言語のみならずコマンド言語としても使用されるようになった．さらに，1980年には，PLIS を母体として引き継ぎ，マイクロ Plan にモジュールの概念を導入した Modular Plan を井出が開発した [6]．マイコン・システムにフロッピーディスクや通信用モデムを付加し，プロセスを扱え，ファイル・システムや通信機能を有した実用的パーソナルコンピュータ・システムを作製中である．

このようにマイクロ Plan についてはソフトウェア側からの経験が蓄積されているが，常にハードウェアの制約に押えられていたので，今回はアーキテクチャの設計まで堀り下げることとした．

```
<program>::= <globaldecl><stat>.
<globaldecl>::= <constdef><vardecl><arraydecl>{<procfuncdecl>}
<constdef>::= <empty>|const<cid>=<const>{,<cid>=<const>};
<cid>::= <identifier>
<const>::= <cid>|<number>
<vardecl>::= <empty>|var<vid>:=<expr>{,<vid>:=<expr>};
<arraydecl>::= <empty>|array<vid>[<expr>]{,<vid>[<expr>]};
<procfuncdecl>::= <procdecl>|<funcdecl>
<procdecl>::= proc<vid>(<formal>);<localdecl><stat>;
<funcdecl>::= func<vid>(<formal>);<localdecl><expr>;
<vid>::= <identifier>
<formal>::= <empty>|<vid>{,<vid>}
<localdecl>::= <vardecl><arraydecl>
<stat>::= <assign>|<proccall>|<compoundst>|<ifst>|<whilest>
<assign>::= <leftfactor>:=<expr>
<expr>::= <simple>|<simple><rel><simple>
<simple>::= <term>|<simple><add><term>
<term>::= <factor>|<leftfactor>|<term><mult><factor>|<term><mult><leftfactor>
<factor>::= <primary>|<unary><factor>|<unary><leftfact>|@<leftfactor>
<leftfactor>::= <leftprimary>|!<factor>|!<leftfactor>
<primary>::= <element>|<primary>(<actual>)|<leftprimary>(<actual>)
<leftprimary>::= <leftelement>|<primary>[<expr>]|<leftprimary>[<expr>]
<element>::= <const>|(<expr>)|<compoundex>
<leftelement>::= <vid>
<actual>::= <empty>|<expr>{,<expr>}
<compoundex>::= valof{<stat>;}return<expr>end
<proccall>::= <primary>(<actual>)|<leftprimary>(<actual>)
<compoundst>::= begin{<stat>;}<stat>end
<ifst>::= if<expr>do<stat>|if<expr>then<stat>else<stat>
<whilest>::= while<expr>do<stat>
<rel>::= =|<>|<|<=|>|>=
<add>::= +|-|or|xor
<mult>::= *|/|mod|and|<<|>>
<unary>::= minus|not
<number>::= <digit>{<digit>}
<identifier>::= <letter>{<letterordigit>}
<letterordigit>::= <letter>|<digit>
<letter>::= a|b|c...z
<digit>::= 0|1|2...9
```

図1　マイクロ Plan の構文規則

進行中の作業

以下に現在進行中の各レベルの設計概要を示す．

図1 はマイクロ Plan 計算機用に再構成した Plan の構文規則である．ハードウェア化する最初のモデルなので，きわめて簡単にしてある．変数パラメタ，値パラメタの区別をするのは厄介だと思ったので，その代用として PLIS のころから BCPL 流の left value, right value の考えを採用している． valof なども BCPL からの借用であり，ある意味ではマイクロ Plan というよりはマイクロ BCPL という趣きをもっている．（マイクロ Plan のセマンティカルな注意事項は最後にのべる．）

図2 はこのマイクロ Plan で書いたプログラムの例で，上はパスカル三角形を書くもの，下は8クィーンズである．現在は Pascal で書いたこのマイクロ Plan 用コンパイラとインタプリタができているが，これはまだ仮の簡単な中間コードを使用している．あとでのべるマイクロ Plan 計算機用中間コードのための処理系は現在作成中である．この新しい処理系のうちインタプリタの部分をマイクロ Plan 計算機が代行する．コンパイラはこの中間コードにおとしたものを用意すれば，マイクロ Plan 計算機にコンパイルも実行させることができる．

```
const m=10,n=4; var i:=0,j:=0;
proc writed(m,n); if n<0 do if m=0 then begin writed*0,n-1); writes(32) end
else begin writed(m/10,n-1); writes(m mod 10+48) end;
proc writen(m,n); begin writed(m/10,n-1); writes(m mod 10+48) end;
func c(n,i); var x:=1; valof if(0<i)and(i<n) do x:=c(n-1,i-1)+c(n-1,i);
return x end;
while i<=m do
begin j:=0; while j<n*(m-i)/2 do begin writes(32); j:=j+1 end;
j:=0; while j<=i do begin writen(c(i,j),n); j:=j+1 end;
writes(13<<8+10); i:=i+1 end.
```

```
const q=8,t=1,f=0;
var n:=0,k:=0;
array x[q],col[q],up[2*q-1],down[2*q-1];

proc generate();
var h:=0;
while h<q do
begin if col[h] and up[n-h+q-1] and down[n+h] do
begin x[n]:=h; col[n-h+q-1]:=f; down[n+h]:=f; n:=n+1;
if n=q then
begin k:=0; while k<q do begin writen(x[k],4); k:=K+1 end;
writes(32<<8+10) end
else generate();
n:=n-1; down[n+h]:=t; up[n-h+q-1]:=t; col[h]:=t end;
h:=h+1 end;
begin k:=0; while k<q do begin col[k]:=t; k:=k+1 end;
k:=0; while k<q do begin col[k]:=t; k:=k+1 end;
k:=0; while k<(2*q-1) do begin up[k]:=t; down[k]:=t; k:=k+1 end;
generate() end.
```

図2　マイクロ Plan でのプログラム例

TYPE 0

| 0 1 2 | 3 4 5 6 7 | 8 9 A B C D E F |
|---|---|---|
| 0 * * | o p r | o p d |

opd: displacement

bit 1: 0:global/1:local　　bit 2: 0:noderef/1:deref

opr:

| | | | |
|---|---|---|---|
| 0 eq | 8 or | 16 minus | 24 call' 6 |
| 1 ne | 9 xor | 17 not | 25 call' 7 |
| 2 gt | 10 mult | 18 call' 0 | 26 call' 8 |
| 3 ge | 11 div | 19 call' 1 | 27 call' n |
| 4 lt | 12 mod | 20 call' 2 | 28 load |
| 5 le | 13 and | 21 call' 3 | 29 load index |
| 6 add | 14 lshift | 22 call' 4 | 30 store' |
| 7 sub | 15 rshift | 23 call' 5 | 31 store index |

TYPE 1 (jump)

| 0 1 2 3 | 4 5 6 7 8 9 A B C D E F |
|---|---|
| 1 0 0 * | relative address |

bit 3: 0:unconditional/1:false

TYPE 2 (load constant)

| 0 1 2 3 | 4 5 6 7 8 9 A B C D E F |
|---|---|
| 1 0 1 0 | immediate constant |
| 1 0 1 1 | constant table index |

TYPE 3

| 0 1 2 | 3 4 5 6 7 | 8 9 A B C D E F |
|---|---|---|
| 1 0 0 | o p r | X X X X X X X X |

X: don't care

opr:

| | | | |
|---|---|---|---|
| 0 eq | 8 or | 16 minus | 24 call 6 |
| 1 ne | 9 xor | 17 not | 25 call 7 |
| 2 gt | 10 mult | 18 call 0 | 26 call 8 |
| 3 ge | 11 div | 19 call 1 | 27 call n |
| 4 lt | 12 mod | 20 call 2 | 28 |
| 5 le | 13 and | 21 call 3 | 29 |
| 6 add | 14 lshift | 22 call 4 | 30 store |
| 7 sub | 15 rshift | 23 call 5 | 31 |

TYPE 4

| 0 1 2 | 3 4 5 6 7 | 8 9 A B C D E F |
|---|---|---|
| 1 1 1 | o p r | X X X X X X X X |

opr:

| | | | |
|---|---|---|---|
| 0 dereference | 8 nop | 16 | 24 |
| 1 getarray | 9 I/O | 17 | 25 |
| 2 return proc | 10 | 18 | 26 |
| 3 return func | 11 | 19 | 27 |
| 4 mark stack 2 | 12 | 20 | 28 |
| 5 mark stack 3 | 13 | 21 | 29 |
| 6 discard | 14 | 22 | 30 |
| 7 halt | 15 | 23 | 31 |

図3　マイクロ Plan 計算機用中間コード

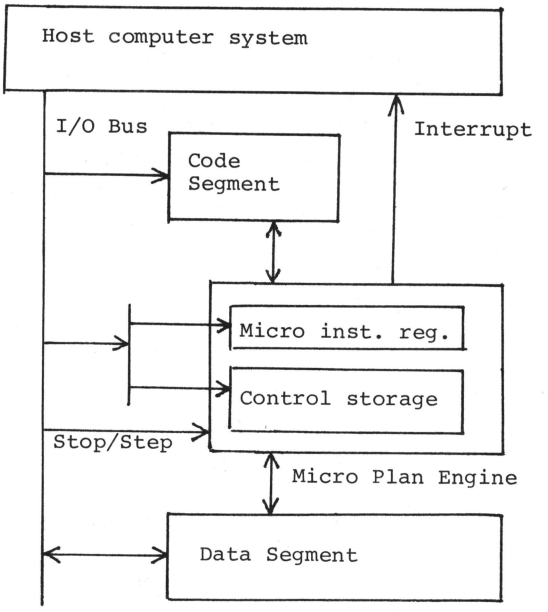



図4　マイクロ Plan 計算機の構成概念図

図3 はマイクロ Plan 計算機用の中間コードである．マイクロ Plan 計算機はそれぞれ1語16ビットのコード用領域とデータ用領域をもっている．（図4 参照）つまり扱うデータはつねに16ビット単位とする。中間コードの方も16ビット単位とする．8ビットで済む中間コードもあるけれども，今回の設計では右半分の8ビットは使わないことにした．

中間コードの機能の主なものは次の通りである．

type 0 はスタックトップとコードのオペランドのデータとの演算用である．そのほか call'0 から8までの手続き，関数呼び出しのコードがある．添字は実パラメタの数を示す．この call は left primary が vid の形のときに用いる．実パラメタの数が9または9以上のときは call'n を用いる． type 1 は相対番地へのジャンプ． type 2 は8ビットに収まらない定数のロードに用いる。 type 3 はスタックトップ内での演算のためで，また後半の call はとび先がスタックにつんであるときのための手続き，関数呼び出し用である． type 4 は雑命令である．

図4 はこのマイクロ Plan 計算機のハードウェアの概念図である．システムとしてはホスト計算機をもち，I／O，記憶装置へのローディングはホストが行なう．マイクロ Plan 計算機の中心となる処理装置は中央やや下の箱であり，今回は Am 2901，2910 系の LSI のまわりに構成してある．

図5 がその部分の構成図である．この部分はマイクロプログラムで制御されるようになっており，図6 にマイクロ命令の形式を示す。

図7 は代表的な中間コードに対するマイクロプログラムの例である．

これらの設計段階での試算によれば，中間コードの一命令の実行時間は約数マイクロ秒の程度で，以前に6800系のマイコンにインプリメントしたときの中間コード実行時間は100マイクロ秒の程度なので，約数10倍高速になったと考えてよい．ハードウェアの方も現在詳細な設計中であり，1981年3月完成を目指している．

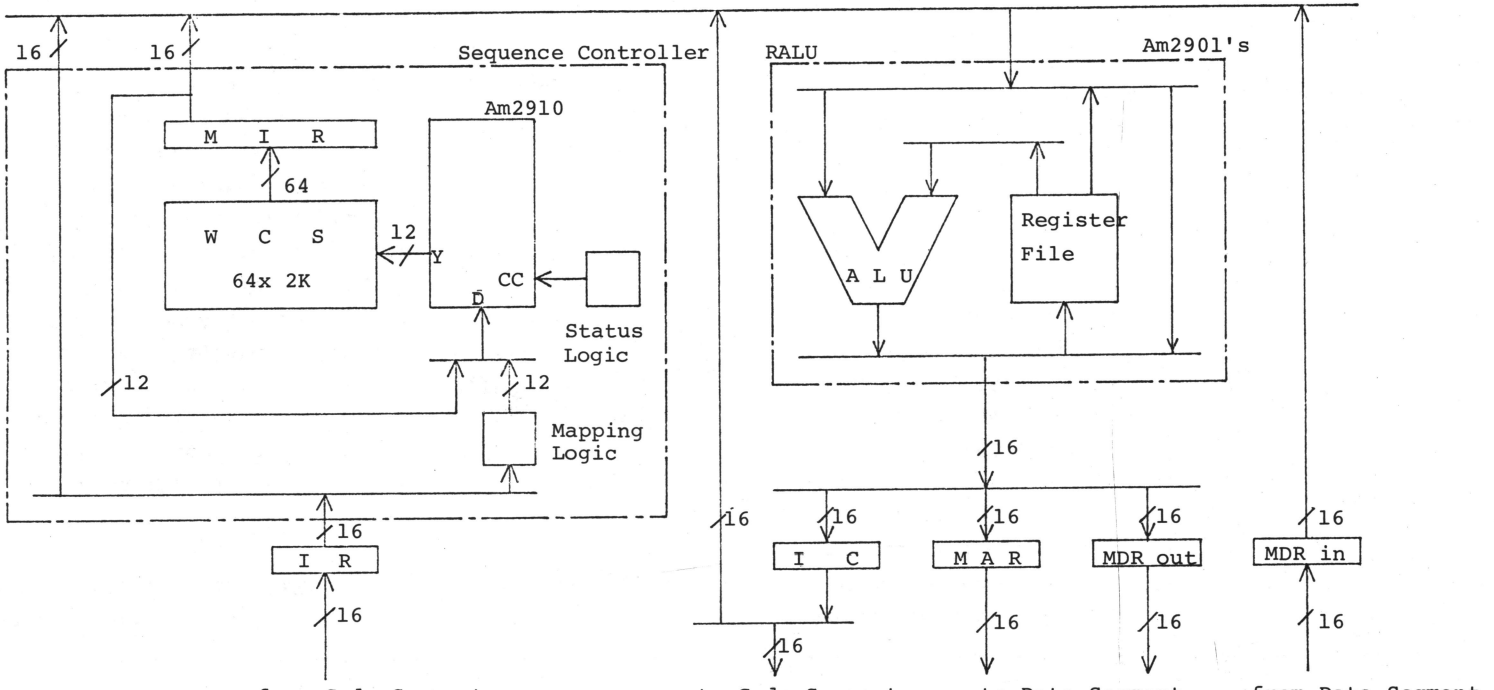
付録　　マイクロ Plan のセマンティカルな注意

1. Pascal は型あり（typed）言語だが， Plan／マイクロ Plan は最初から伝統的に型なし（monotype）言語である．今回のマイクロ Plan では，一応，変数（vid）に，単純変数の初期値，配列の先頭番地，手続き・関数のコードの先頭番地が入るから3種の型があるようだが，相互に代入できる．ただし使い方をまちがえると何がおきるかわからない．
2. 配列は宣言時に [<式>] で大きさを指定する。式の値をnとすると下限0，上限n－1のn個の要素をもつ一次元の配列が配列領域にとられて，添字0に相当する要素の番地が配列名の変数に代入される．

3. BCPL と同様に left valueをもつ構文， left factor, left primary, left element が存在する． left value は代入文の左辺と @ の被演算子として使われる．
4. 入出力については規定しない．適当な標準手続き関数があるものとする。
5. 構文からわかるように，手続き・関数の宣言は一重である．つまり手続き・関数の中で更に手続き・関数を宣言することはできない．しかし手続き・関数を再帰的に呼ぶのは構わない。
6. 手続き・関数の仮パラメタに，変数パラメタは存在しない．パラメタメカニズムはすべて値パラメタとする．変数パラメタを実現するには，実パラメタを @ 変数の型として，変数の番地をわたし，手続き・関数本体で ! パラメタへの代入を実行すればよい．
7. 乗除演算子（< mult >）の '／' は整数除算（Pascal の 'div'），'<<'と'>>' はそれぞれ左シフト，右シフト（第二被演算子の値だけシフトする）を表す。
8. 単項演算子（< unary >） の 'minus' はそのうしろの因子（ <factor> ） の値を正負反転する演算を表わす．

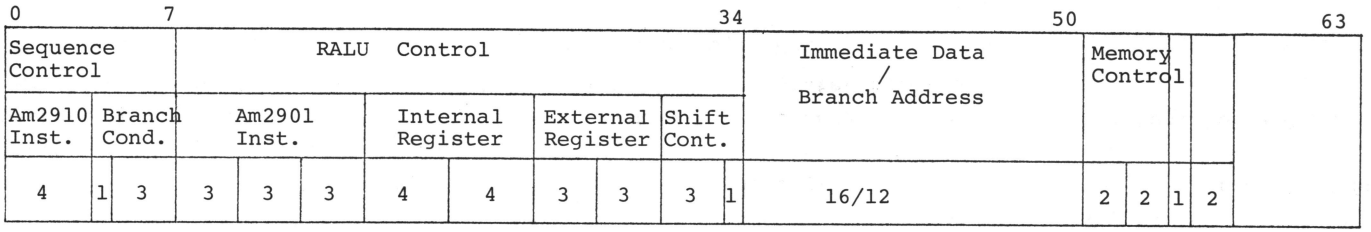
参考文献


[1] Mead, C. and Conway, L.： "Introduction to VLSI Systems", Addison-Wesley
[2] 武市正人： "ミニ言語のミニコンパイラ"，bit 1974年8月-12月
[3] 前野年紀： "PLAN その後" (PTT 資料)
[4] 戸村 哲 ： "マイクロ Plan のプロセッサ"， bit 1978年2月，マイクロコンピュータのプログラミング，臨時増刊
[5] 岡本恵里，松井俊浩： 東京大学工学部計数工学科卒業論文 1980年3月
[6] 井出一郎： "Modular Plan" (研究室輪講資料)
[7] Advanced Micro Devices, Inc.： "The Am2900 Family Data Book", 1979年




MIR: Micro Instruction Register
IR: Instruction Register
IC: Instruction Counter
WCS: Writable Control Storage
MAR: Memory Address Register
MDR out: Memory Data Register for output
MDR in: Memory Data Register for input
ALU: Arithmetic Logic Unit
RALU: Register and ALU

図5　マイクロ Plan 計算機の処理装置




Micro Instruction (0:63)
  Sequence Control (0:7)
    Instruction (0:3) -- Am2910 I0:I3
    Condition Control (4:7)
      Polarity (4)
      Flag Selection (5:7)
  RALU Control (8:34)
    Instruction (8:16) -- Am2910 I0:I8
    Internal Register Selection (16:23)
      Register A (16:19)
      Register B (20:23)
    External Register Selection (24:29)
      Output Register (24:26)
      Input Register (27:29)
    Shift Control (29:34)
      Shift Mode (29:33)
      Carry Control (34) -- to LSS C0_input
  Immediate Data (35:50) / Branch Address (39:50)
  Memory Control (51:54)
    Code Segment Control (51:52)
      Read/Write Control (51)
      Strobe (52)
    Data Segment Control (53:54)
      Read/Write Control (53)
      Strobe (54)
  PC Control (55)
  F/F Control (56:57)
    Set F/F (56)
    Reset F/F (57)
  Not Used (58:63)

図6　マイクロ命令の形式

```
* def MAR = extout(1);
* def MDRout = extout(2);
* def ICout = extout(3);
* def MDRin = extin(1);
* def IR = extin(2);
* def SP = intr(0);
* def LB = intr(1);
* def IA = intr(2); -- Instruction Address
* def TOS = intr(3); -- Top Of Stack register
* def X = intr(15); -- temporary register
* def EMPTY = cond(1); -- F/F test
FETCH: ICout:=IA;
       cRD; -- cRD: code read
       cRD,STB; -- STB: strobe
       jmap;
-----------------------
TYPE0-ADD: -- add/local/deref
       cjp,not,EMPTY,L0; X:=IR and X'00FF';
       MAR:=SP;
       SP:=SP-1; dRD; -- dRD: data read
       MAR:=X+LB; dRD,STB; incIC; -- incIC: increment IC
       TOS:=MDRin; dRD;
       dRD,STB;
       jmap; TOS:=TOS+MDRin; setF/F;
L0:    MAR:=X+LB; incIC;
       dRD; cRD;
       dRD,STB; cRD,STB;
       jmap; TOS:=TOS+MDRin; setF/F;
-----------------------
TYPE0-LOAD: -- load/local/deref
       cjp,EMPTY,L1; X:=IR and X'00FF';
       MAR:=SP:=SP-1;
       MDRout:=TOS; dWT; -- dWT: data write
       MAR:=X+LB; dWT,STB; incIC;
       dRD; cRD;
       dRD,STB; cRD,STB;
       jmap; TOS:=MDRin; setF/F;
L1:    MAR:=X+LB; incIC;
       dRD; cRD;
       dRD,STB; cRD,STB;
       jmap; TOS:=MDRin; setF/F;
```

図7　マイクロプログラムの例