計算機アーキテクチャ 59−4
オペレーティング・システム 29−4
(1985・10・24)

プログラム可能論理演算装置を用いた計算機のソフトウェアについて

SOFTWARES FOR THE COMPUTER USING A PROGRAMMABLE LOGIC UNIT

吉岡 良雄 ※　山田 秀和 ※※　中村 維男 ※※　重井 芳治 ※※

Yoshio YOSHIOKA※, Hidekazu YAMADA※※, Tadao NAKAMURA※※, Yoshiharu SHIGEI※※

※ 岩手大学工学部情報工学科、盛岡市。
Department of Computer Science, Faculty of Engineering, Iwate University, Morioka-shi 020 Japan.

※※ 東北大学工学部情報工学科、仙台市。
Department of Information Science, Faculty of Engineering, Touhoku University, Sendai-shi 980 Japan.

1. まえがき

ノイマン型の計算機は、電子回路素子の高集積化・高信頼化と共に、オペレーティングシステムや高級言語コンパイラなどの開発によって、急速に発展した。さらに、このタイプの計算機は、処理アルゴリズムが逐次的に進むために非常に汎用性が高く、算術演算ばかりではなく種々の方面に応用されたことが急速に発展した要因の一つになっている。しかしながら、データ処理分野での応用が拡大して、計算機の処理速度に対する要求が増々高まり、このタイプの処理アルゴリズムの検討が行われ、各種の問題点が指摘されるようになった［1］。この問題点は、次のようなものである。一つは、真に処理の対象となるデータ語以外の情報（命令語やアドレス生成語）が1本のデータパス上を流れる点である。Ｂａｃｋｕｓは、このパスにボトルネックがあることを指摘し、これを『フォン・ノイマン・ボトルネック』と呼んでいる。二つめは、1つのプログラムカウンタによって処理のシーケンスが規定される点である。従って、処理すべきアルゴリズムは、逐次的なプログラムとしなければならない。これは、命令レベルでの並列実行可能性を基本的に持たないことを意味する。また、繰り返し処理においても何度も同じ命令が読み出されることになり、前述のフォン・ノイマン・ボトルネックの一つの要因となっている。三つめは、プログラムと処理データが同一のメモリ領域に格納されている点である。

これらの問題を解決するため、スーパコンピュータなどに見られる複数のパイプライン処理装置を用いた計算機やアレイプロセサなど［2］、さらにはプログラムカウンタを持たないデータフローマシンなど［3］［4］が出現した。また一方では、中央処理装置（ＣＰＵ）からのプログラムによって、演算回路をダイナミックに変えて、演算データの格納と結果データの読出しだけで処理を進める装置が提案された［5］。この装置は、プログラム可能論理演算装置（ＰＬＵ：Ｐｒｏｇｒａｍｍａｂｌｅ Ｌｏｇｉｃ Ｕｎｉｔ）と呼ばれ、ＣＰＵからはメモリ装置と同様に扱うことができる。また、この装置は、従来の処理アルゴリズムＳＩＳＤ、ＳＩＭＤ、ＭＩＳＤ、ＭＩＭＤ、データフローやデマンドフローなどの概念と異なり、非常に興味がある。この装置に対するソフトウェアレベルの検討、特にこのような装置を用いてどのような処理が可能であるかなどの検討が必要であると思われる。

本論文は、このようなＰＬＵを用いた処理ソフトウェアについて検討を行う。まず、2章ではＰＬＵの概念と処理アルゴリズム、および本論文で取り扱うＰＬＵの構造と演算機能について示す。さらに、3章ではＰＬＵを用いた計算機のコンパイラについて示し、このようなＰＬＵや言語コンパイラなどの検討を行う。

2. ＰＬＵの構成

ＰＬＵの構造は、文献［5］および［6］にも述べられているように、その使用目的に応じて種々のものが考えられる。まず本章では、ＰＬＵの概念と処理アルゴリズムについて述べ、次に演算処理を目的として、言語やコンパイラを考慮したＰＬＵの構成について述べる。また、ＰＬＵをソフトウェア面

から考察するため、ＰＬＵをメインＣＰＵの記憶装置上にマイクロプロセサでシミュレートするシミュレータを作成した。このシミュレータの構成についても述べる。

２．１　ＰＬＵの概念と処理アルゴリズム

高速フーリェ変換器やデジタルフイルタなどの処理マシンは、電子回路素子で構成され、素子のゲート遅延時間レベルで演算が可能な専用処理装置である。このため非常に高速である。このような装置は汎用性に欠けるため、非常に高価なものになる。一方このような装置を従来のプロセサ（ＣＰＵ）を用いてプログラムで実現することも可能である。この場合、融通性や汎用性は高いが、高速性に欠ける。これに対して、本論文のプログラム可能論理演算装置（ＰＬＵ）は、これらの装置の中間に位置し、汎用性と高速性を狙ったものである。

さて、本ＰＬＵの処理の概念は、図１に示すように、ＰＬＵに演算データ群Ｘと処理プログラム（操作群）Ｆが書き込まれた後、素子のあるゲート遅延時間後にその処理結果Ｙが出力される構成で表される。そして、このようなＰＬＵの処理アルゴリズムには次の２つが考えられる。

（ａ）．最初に処理プログラムをＰＬＵに格納してＰＬＵ内に演算回路を構成（プログラムのマッピング）し、その後に演算データをＰＬＵの入力部に格納する。そして、あるゲート遅延時間後にＰＬＵの出力部に演算結果が現れるので、これをＣＰＵが取り出す。この方法は、異なるデータに対して同じプログラムで処理を行うような処理に適している［５］。例えば、ループ処理、ベクトル演算などがこの処理に適していると言える。

（ｂ）．同一のデータ群Ｘに対して、種々の操作Ｆを変えて、その操作に対する結果を得る方法が考えられる。また、図２に示すように、その操作に対する結果が必要とする結果となるような操作を抽出するという処理も考えられる。例えば、Ｐｒｏｌｏｇ言語において、定義文が操作Ｆとなり、質問文がデータ群となる。そして、その質問文に対して操作とマッチングを取り、結果が真となる定義文を取り出すと言う処理に相当する。

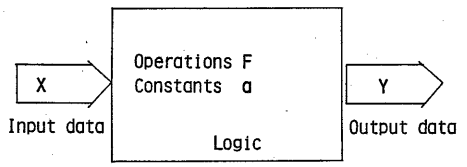

図１　プログラム可能論理演算装置（ＰＬＵ）の概念図

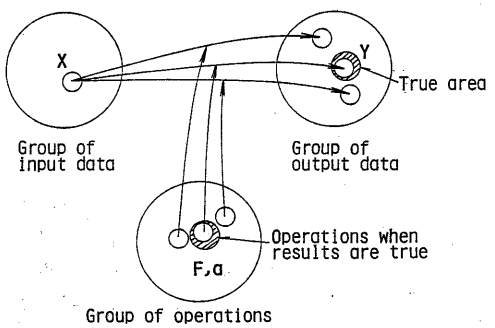

図２　入力データ群（Ｘ）、操作群（Ｆ、ａ）および出力データ群（Ｙ）との関係

これら２つの処理アルゴリズムは、ＰＬＵの持っている特性から類推されるものであり、これらを実現するためには、演算データの転送やプログラムのマッピング、データのフロー制御、演算時のエラー処理など解決しなければならない問題が沢山存在する［６］。また、従来のプログラムでも容易にこのＰＬＵを用いて処理できるというソフトウェアの遺産の問題も考慮しなければならない。

２．２　ＰＬＵの構造

ここで取り扱うＰＬＵは、主に演算用であり、言語コンパイラにおいて変数や定数が容易にレジスタに割当てられるように、図３のような各ステップのデータレジスタ部と演算部が独立になったものを考える。この演算部（ＡＬＵ）の２入力は、各ステップの命令レジスタ（ＩＲ）の内容によって各データレジスタ（ＤＲ）に接続されているバス（Bus_0－Bus_{255}）から２つ選ばれる。また、このバスに各ステップ上でゲート（$Gate_0$－$Gate_{255}$）が設られているので、演算結果が出力されるバスは前の状態から切り離される。さらに、メインの処理装置（ＣＰＵ）のデータバスには、図３に示すように命令レジスタとデータレジスタが接続され、それぞれプログラムとデータがメインＣＰＵによって格納される。なお、ＣＰＵからデータレジスタｎを読み込むとき、このデータレジスタに接続されているバスBus_nの状態がＣＰＵに読み込まれる。

このようなＰＬＵにおけるデータレジスタの数および演算部のステップ数は、それぞれDR_0－ＤＲ

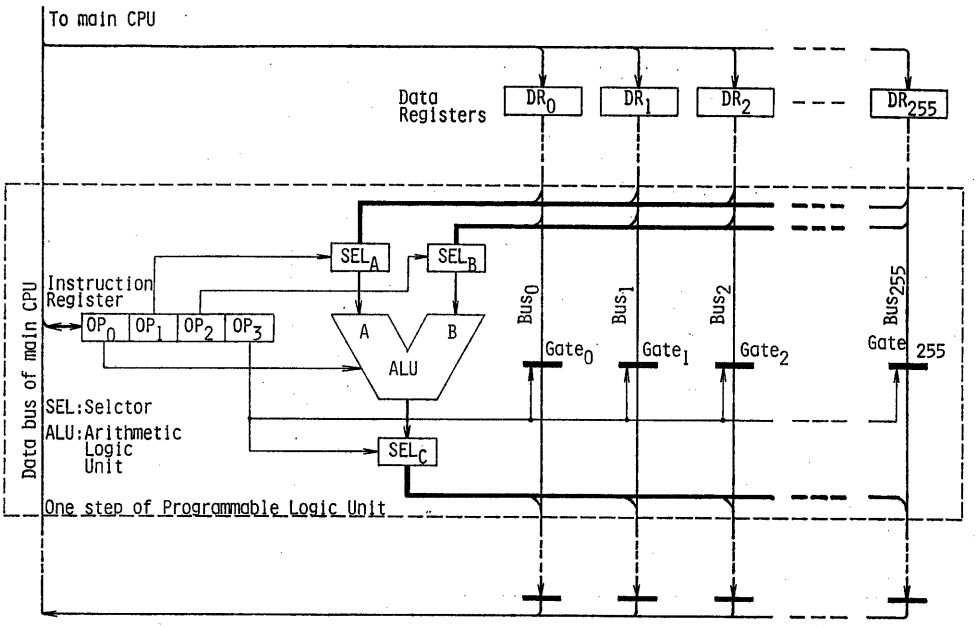

図３　本論文で取り扱うＰＬＵの構造

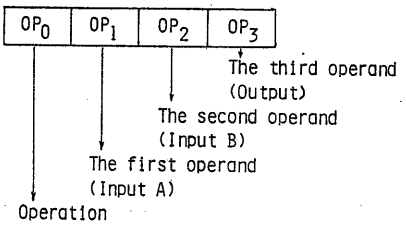

図４　ＰＬＵの命令形式

２５５の２５６個および数十Ｋステップを考える。また、命令レジスタの構成は、図４に示すように、命令部（OP_0）、第１オペランド部（OP_1）、第２オペランド部（OP_2）および第３オペランド部（OP_3）から成り、それぞれ１バイトが割当てられている。ここで、命令部の機能は、表１に示すように、加減乗除、指数や対数など高レベルの固定少数点演算を取り入れた。また第１オペランド部および第２オペランドによってDR_0－DR_{255}に対応しているバスから演算部の２入力が決められ、第３オペランド部によって結果データの出力するバスが決められる。なお、表１において、Ａ、ＢおよびＣは、それぞれ演算部の２入力および出力を決めるためのデータレジスタに接続されているバス番号である。また、命令コード（２０）および（２１）は、それぞれアドレス（Ａ＋４×Ｂ）のデータをバス番号Ｃへ出力する命令およびバス番号Ｃの内容をアドレス（Ａ＋４×Ｂ）へ格納する命令である。この命令は、配列処理命令であり、論理回路で構成されたＰＬＵでは実行不可能である。むしろ、メインＣＰＵで実行すべき命令であると思われる。

２．３　ＰＬＵシミュレータ

本ＰＬＵシミュレータは、図５に示すようにメインのＣＰＵとしてＭＣ６８００を用いたパーソナルコンピュータ・ベーシックマスタ"Ｊｒ"に、ＰＬＵをシミュレートするＭＣ６８０９を接続して作成された。このメモリマップは図６のように成っている。ここで、ＰＬＵのデータレジスタ部および命令レジスタ部（プログラム領域）は、それぞれ４０００－４３ＦＦ（４バイト×２５６データレジスタ）および４４００－５ＦＦＦ（４バイト×１７９２ステップ）の番地が割当てられている。また配列データやメッセージの格納領域（データ領域）として６０００－９ＦＦＦの番地が割当てられている。従ってメモリ領域は、データレジスタ領域、プログラム領域およびデータ領域の３種類に分離されていることになる。そして図３に示すＰＬＵは、次のようにして実現される。各ステップの処理は、プログラム領域のOP_1とOP_2によってデータレジスタ領域から２つのデータを取り出し、演算を行い、OP_3の示すデータレジスタ領域へ格納する。さらに、こ

表1　ＰＬＵで実行する命令

Operation Code	Operation	Comment
01	A → C	
02	B → C	
03	A+B → C	Add
04	A-B → C	Subtract
05	A*B → C	Multiply
06	A/B → C	Divide
07	A**B → C	A^B
08	A//B → C	$A^{1/B}$
09	ABS(A) → C	Absolute
0A	MOD(A,B) → C	Modulus
0B	-A → C	
10	LOG2(A) → C	Log_2A
11	2**A → C	
12	LOG10(A) → C	$Log_{10}A$
13	10**A → C	
14	LOGE(A) → C	Log_eA
15	EXP(A) → C	e^A
20	(A+4*B) → C	
21	C → (A+4*B)	
22	AND(A,B) → C	Logical AND
23	OR(A,B) → C	Logical OR
24	EOR(A,B) → C	Exclusive OR
25	LSHFT(A,B) → C	Logical shift
26	ASHFT(A,B) → C	Arithmetic shift
30	SIN(A) → C	Sine
31	COS(A) → C	Cosine
32	TAN(A) → C	Tangent
FF	End	To top of program
Otherwise	Nop	No operation

表2　メインＣＰＵが実行する命令

Operation Code	Operation	Comment
81	A → C	
82	A+B → C	Add
83	A-B → C	Subtract
90	IF	Test and skip
91	TEST	Test and skip
92	GOTO	Jump
93	CALL	Subroutine call
94	RETURN	Return
E0	WAIT	Wait
F0	READ	Read
F1	WRITE	Write
FF	End	End of program
Otherwise	Nop	No operation

の処理は、プログラムの先頭から終りまでエンドレスの形でＭＣ６８０９によって行われる。

次に、メインＣＰＵで実行する命令は、ＰＬＵで実行する命令とオペランド部の意味は異なるがほぼ同じ形式であり、表２に示す通りである。この命令は、ＰＬＵの実行命令と同様にＰＬＵのプログラムメモリにマッピングされる。ＰＬＵでは、この命令は無操作（Ｎｏｐ）であるから、ＰＬＵに対して影響を及ぼさない。そして、この命令は、メインＣＰＵによって逐次的に実行される。また、メインＣＰＵにおいて、ＰＬＵの命令は、無操作（Ｎｏｐ）として取り扱われる。なおメインＣＰＵによって実行される命令は、表２のように、代入、加算、減算、２種類のテスト・スキップ（ＩＦとＴＥＳＴ）、ジャンプ（ＧＯＴＯ）、サブルーチンコール（ＣＡＬＬ）、リターン（ＲＥＴＵＲＮ）、ウエイト（ＷＡＩＴ）、入力（ＲＥＡＤ）、出力（ＷＲＩＴＥ）および終了の１２命令だけである。特に、ＩＦ、ＴＥ

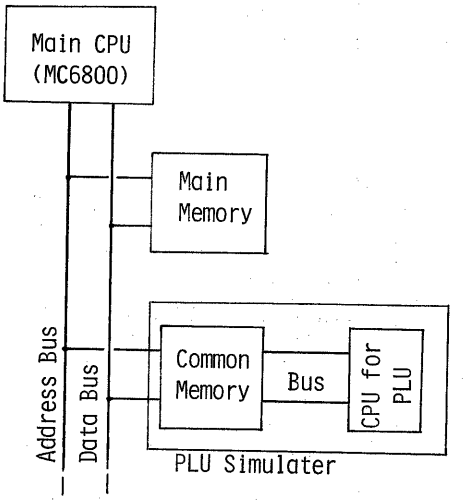

図5　ＰＬＵシミュレータの構成図

0000 ~ 00FF	Monitor work area
0100 ~ 03FF	Video RAM
0400 ~ 0FFF	PLU work area
1000 ~ 1FFF	Not use
2000 ~ 3FFF	Source program area
4000 ~ 43FF	Data registers of PLU (4 byte x 256 DR's)
4400 ~ 5FFF	Program area of PLU (4 byte x 1792 steps)
6000 ~ 9FFF	Data area of PLU
A000 ~ AFFF	Stack area
B000 ~ CFFF	Compiler and simulater of PLU
D000 ~ DFFF	Monitor and editor
E000 ~ EFFF	I/O address
F000 ~ FFFF	"Jr" monitor

図6　ＰＬＵシミュレータのメモリマップ

ＳＴ、ＧＯＴＯ、ＣＡＬＬ、ＲＥＡＤ、ＷＲＩＴＥは、それぞれ次のような命令となる。

```
ＩＦ　Ｘ，ｃｃ　Ｙ　→　９０，ＸＸ，ｃｃ，ＹＹ
ＴＥＳＴ　Ｘ　　→　９１，ＸＸ，００，００
ＧＯＴＯ　ＡＤＤＲＸ→　９２，００，ＡＤＤＲＸ
ＣＡＬＬ　ＡＤＤＲＸ→　９３，００，ＡＤＤＲＸ
ＲＥＡＤ　Ｘ　　→　Ｆ０，２０，００，ＸＸ
ＷＲＩＴＥ　Ｘ　　→　Ｆ１，２０，００，ＸＸ
又は、ＷＲＩＴＥ　″（ＭＥＳＳＡＧＥ）″　→
　　Ｆ１，４Ｄ，ＡＤＤＲＭ
```

ここで、ＸＸおよびＹＹは、それぞれ変数Ｘおよび Ｙに割当てられたレジスタ領域のレジスタ番号であり、ＡＤＤＲＸおよびＡＤＤＲＭは、それぞれプログラム領域の番地およびデータ領域の番地である。また、ｃｃはスキップ条件であり、００（ＥＱ）、０１（ＮＥ）、０２（ＧＴ）、０３（ＬＴ），０４（ＧＥ），０５（ＬＥ）の値を取る。

以上、ＰＬＵの概念とその処理アルゴリズム、本論文で取り扱うＰＬＵの構成、およびＰＬＵのハードウェアシミュレータの構成について述べた。

３．言語コンパイラとその検討

本章では、表１および表２に示す命令を用いて作成したＰＬＵのための言語とこのコンパイラについて述べる。次に、この言語によるプログラム例を示す。そして、プログラムを作成するための問題点などを指摘する。

３．１　言語の説明

本論文で取り入れた言語は、次のような文によって記述される。

（１）．コメント文
　’（Ｍｅｓｓａｇｅ）／
（２）．配列文
　ＤＩＭＥＮＳＩＯＮ　Ａ，１０００
（３）．Ｄａｔａ文
　ＤＡＴＡ　Ａ＝１０１／
（４）．ＢｅｇｉｎおよびＥｎｄ文
　ＢＥＧＩＮ；／（文）／ＥＮＤ；／
（５）．演算文
　Ｃ＝Ａ＋Ｂ／Ｃ＝Ａ－Ｂ／－－－ｅｔｃ．
（６）．Ｉｆ文
　ＩＦ　Ｘ，ｃｃ，Ｙ／ＴＨＥＮ／（文）／；
　／ＥＬＳＥ／（文）／；／
　※　ｃｃ：ＥＱ－－Equal　（００）
　　ＮＥ－－Negative Equal　（０１）
　　ＧＴ－－Greater Than　（０２）
　　ＬＴ－－Less Than　（０３）
　　ＧＥ－－Greater than and Equal　（０４）
　　ＬＥ－－Less than and Equal　（０５）
（７）．Ｃａｓｅ文
　ＣＡＳＥ　Ｘ　ＯＦ／０：／（文）／；／
　１：／（文）／；／２：／－－－／
　１６：／（文）／；／ＥＮＤ；／
（８）．Ｗｈｉｌｅ文
　ＷＨＩＬＥ　Ｘ，ｃｃ，Ｙ／（文）／；／
（９）．Ｒｅｐｅａｔ文
　ＲＥＰＥＡＴ／（文）／
　ＵＮＴＩＬ　Ｘ，ｃｃ，Ｙ／
（１０）．Ｆｏｒ文
　ＦＯＲ　Ｉ＝１／ＴＯ　Ｎ／ＳＴＥＰ　Ｋ／
　（文）／；／
（１１）．入出力文
　ＲＥＡＤ　Ｘ／
　ＷＲＩＴＥ　Ｘ／
　ＷＲＩＴＥ　″（Ｍｅｓｓａｇｅ）″／

ここで、″／″はキャリジレターンである。また、文の変数や定数にはコンパイラによって自動的に使用されていないレジスタ番号が割当てられる。特に定数の場合レジスタ内にもその値が入れられる。

これらの文の中で算術演算文は、表１に示す命令にそのまま変換される。ここで重要と考えられる文は、″ＩＦ″や″ＦＯＲ″などの制御文である。これらの文は、表２で示されている命令を用いて付録に示すように実現される。

３．２　プログラム例

３．１節で示した言語を用いて、素数を求めるプログラム、２次方程式の根を求めるプログラムおよびＮ＝８のＦＦＴのプログラムをそれぞれ図７、図８および図９に示す。ここで、素数を求めるプログラムは、メインＣＰＵで実行する命令が非常に多いので、ＰＬＵを用いた利点を十分発揮しているとは言えないが、配列処理も可能であることを示している例である。

また、図８のプログラムは、演算文がどこにあっても同じであるから、２乗根の中が正負の場合を先に計算した後２乗根の中の正負を判定し、正負に応じた処理（結果の出力）を行っている例である。このプログラムの動作は、次の通りである。このプログラムの演算文は、入力データが確定していなくてもＰＬＵ内で実行されている。そしてメインＣＰＵで″ＲＥＡＤ″文が実行され、入力データが確定し，

```
No. Addr Instruc. Source program
  1                 ' SOSUU KEISAN
  2 4400            DIMENSION A,1000
  3 4400            BEGIN;
  4 4400 F14D6FA0   WRITE "INPUT N "
  5 4404 F0200001   READ N
  6 4408 08010302   N2=N//2
  7 440C 81040405   FOR I=1
  8 4410                TO N
  9 4410 21000505   A,I=I
 10 4414 E0000000   ;
 10 4418 82050405
 10 441C 90010305
 10 4420 92004410
 11 4424 81030306   FOR J=2
 12 4428                TO N2
 13 4428 E0000000   WAIT
 14 442C 20000607   GG=A,J
 15 4430 90070008   IF GG,EQ,0
 16 4434 92004444   THEN
 17 4438 92004488   GOTO LLL
 18 443C E0000000   ;
 19 4444            ELSE
 20 4444 E0000000   ;
 21 4448 8106060A   FOR K=J
 22 444C                TO N
 23 444C E0000000   WAIT
 24 4450 05060A0B   JJ=J*K
 25 4454 900b0501   IF JJ,LE,N
 25 4458 92004470
 26 445C            THEN
 27 445C F120000B   WRITE JJ
 28 4460 010B080C   KK=JJ
 29 4464 21000C08   A,KK=0
 30 4468 E0000000   ;
 30 446C 92004478
 31 4470            ELSE
 32 4470 92004488   GOTO LLL
 33 4474 E0000000   ;
 34 4478 E0000000   ;
 34 447C 820A040A
 34 4480 9001030A
 34 4484 9200444C
 35 4488            LLL:
 36 4488 E0000000   ;
 36 448C 82060406
 36 4490 90020306
 36 4494 9200444C
 37 4498 8104040D   FOR MM=1
 38 449C                TO N
 39 449C E0000000   WAIT
 40 44A0 20000D0E   HH=A,MM
 41 44A4 900E0108   IF HH,NE,0
 41 44A8 920044BC
 42 44AC            THEN
 43 44AC F14D6FA9   WRITE "SOSUU = "
 44 44B0 F120000E   WRITE HH
 45 44B4 E0000000   ;
 45 44B8 920044C0
 46 44BC            ELSE
 47 44BC E0000000   ;
 48 44C0 E0000000   ;
 48 44C4 820D040D
 48 44C8 9001030D
 48 44CC 9200449C
 49 44D0 F14D6FB2   WRITE "PROGRAM END"/
 50 44D4 FFFFFFFF   END;
***ERROR COUNT = 00
```

図7　プログラム例1（素数計算）

```
No. Addr Instruc. Source program
  1                 ' 2JI NO KON KEISAN
  2 4400            BEGIN;
  3 4400 F14D6000   WRITE "INPUT A= "
  4 4404 F0200000   READ A
  5 4408 F14D600A   WRITE "INPUT B= "
  6 440C F0200001   READ B
  7 4410 F14D6014   WRITE "INPUT C= "
  8 4414 F0200002   READ C
  9 4418 05010103   A1=B*B
 10 441C 05000204   A2=A*C
 11 4420 05060405   A3=4*A2
 12 4424 04030507   A4=A1-A3
 13 4428 08070908   A5=A4//2
 14 442C 040B010A   A6=-B
 15 4430 030A080C   A7=A6+A5
 16 4434 040A080D   A8=A6-A5
 17 4438 0509000E   A9=2*A
 18 443C 060C0E0F   X1=A7/A9
 19 4440 060D0610   X2=A8/A9
 20 4444 040b0711   AA=-A4
 21 4448 08110912   AB=AA//2
 22 444C 060A0E13   Y1=A6/A9
 23 4450 06120E14   Y2=AB/A9
 24 4454 E0000000   WAIT
 25 4458 9007040B   IF A4,GE,0
 25 445C 92004478
 26 4460            THEN
 27 4460 F14D601E   WRITE "X1 = "
 28 4464 F120000F   WRITE X1
 29 4468 F14D6024   WRITE "X2 = "
 30 446C F1200010   WRITE X2
 31 4470 E0000000   ;
 31 4474 9200448C
 32 4478            ELSE
 33 4478 F14D602A   WRITE "Y1 = "
 34 447C F1200013   WRITE Y1
 35 4480 F14D6030   WRITE "Y2 = "
 36 4484 F1200014   WRITE Y2
 37 4488 E0000000   ;
 38 448C FFFFFFFF   END;
***ERROR COUNT = 00
```

図8　プログラム例2（2次方程式の根）

ある時間（"ＷＡＩＴ"文で待つ）経過すると全ての演算結果が確定する。その後、"ＩＦ"文で2乗根の中の正負を判定し、その結果に従って結果データを出力している。これは、図7のプログラムと比較して、ＰＬＵの特徴を表しているよい例である。

さらに、図9のプログラムは、ＦＦＴの計算式をＰＬＵで計算できるように変形して作成した例である。なおＦＦＴの一般式をＰＬＵで行うこともできるが、この場合制御文が非常に多くなる。そこで、この例のようにＮの値をＰＬＵにマッピングし易い形にした方がよい。すなわち、ループや再帰処理のような繰り返し処理は、ＰＬＵで計算し易いように式を変形したり、展開したりした方がよい。

３．３　プログラム作成における問題点

ここで示したＰＬＵのための言語および言語コンパイラなどにおいて、種々の問題点がある。以下にこの問題点とその解決法について述べる。

（ａ）．メインＣＰＵとＰＬＵの同期

ＰＬＵが本論文で示したシミュレータような演算専用プロセッサで行うと、ＰＬＵでの演算時間を無視することができなくなるので何らかの同期フラグをメインＣＰＵとＰＬＵの間に設ける必要がある。本シミュレータでは、同期フラグを設けてＰＬＵで実行する一連の命令の実行終了まで待つ命令"ＷＡＩＴ"によって同期を取っている。ＶＬＳＩ技術が進歩し、高速の素子を用いた論理回路で作成されたＰＬＵが実現されれば、この問題はある程度解決されるであろう。

（ｂ）．データの流れを変える制御文

演算結果によってデータの流れが変わるようなプログラムの場合、次の2通りが考えられる。一つは従来の逐次型計算機で用られているように、"ＩＦ

```
NO ADDR CODE     SOURCE
 1 4400          ' FFT (N=8)
 2 4400          DATA W1=0.7071
 3 4400          DATA W2=0
 4 4400          DATA W3=-0.7071
 5 4400          BEGIN;
 6 4400 F0200003 READ X0
 7 4404 F0200004 READ X1
 8 4408 F0200005 READ X2
 9 440C F0200006 READ X3
10 4410 F0200007 READ X4
11 4414 F0200008 READ X5
12 4418 F0200009 READ X6
13 441C F020000A READ X7
14 4420 0303070B A04=X0+X4
15 4424 0403070C B04=X0-X4
16 4428 0304080D A15=X1+X5
17 442C 0404080E B15=X1-X5
18 4430 0305090F A26=X2+X6
19 4434 04050910 B26=X2-X6
20 4438 03060A11 A37=X3+X7
21 443C 04060A12 B37=X3-X7
22 4440 05000E13 W1B15=W1*B15
23 4444 05011014 W2B26=W2*B26
24 4448 05021215 W3B37=W3*B37
25 444C 05010D16 W2A15=W2*A15
26 4450 05011117 W2A37=W2*A37
27 4454 05020E18 W3B15=W3*B15
28 4458 05001219 W1B37=W1*B37
29 445C 030B0F1A D0426=A04+A26
30 4460 030E111B D1537=A15+A37
31 4464 030C141C E0426=B04+W2B26
32 4468 0313151D E1537=W1B15+W3B37
33 446C 040B0F1E F0426=A04-A26
34 4470 0416171F F1537=W2A15-W2A37
35 4474 040C1420 G0426=B04-W2B26
36 4478 03181921 G1537=W3B15+W1B37
37 447C 031A1B22 Y0=D0426+D1537
38 4480 031C1D23 Y1=E0426+E1537
39 4484 031E1F24 Y2=F0426+F1537
40 4488 03202125 Y3=G0426+G1537
41 448C 041A1B26 Y4=D0426-D1537
42 4490 041C1D27 Y5=E0426-E1537
43 4494 041E1F28 Y6=F0426-F1537
44 4498 04202129 Y7=G0426-G1537
45 449C E0000000 WAIT
46 44A0 F1200022 WRITE Y0
47 44A4 F1200023 WRITE Y1
48 44A8 F1200024 WRITE Y2
49 44AC F1200025 WRITE Y3
50 44B0 F1200026 WRITE Y4
51 44B4 F1200027 WRITE Y5
52 44B8 F1200028 WRITE Y6
53 44BC F1200029 WRITE Y7
54 44C0 FFFFFFFF END;
***ERROR COUNT = 00
```

図９　プログラム例３（Ｎ＝８のＦＦＴ）

〃や代入などの制御文を用いてデータの流れを変える方法である。もう一つは、図８のプログラム例のように、考えられる演算をすべて行ってから、必要とする結果を得る方法である。前者の方法は、〃ＩＦ〃などの制御文中の演算（〃；〃までの命令）をメインＣＰＵで実行しなければならないためメインＣＰＵの負荷が大きくなるので、良い方法とは言えない。従って、ＰＬＵを用いた計算機では、後者の方法を提唱する。

（ｃ）．〃ＷＡＩＴ〃命令の使用

〃ＷＡＩＴ〃は、ＰＬＵの一連の演算が終るまで待つと言う同期命令であるため、メインＣＰＵにとって処理効率を下げる要因となる命令である。従って、この命令の多用を避けるような工夫がプログラムに必要である。またメインＣＰＵの処理効率を上げる意味において、次のような方法を取ると良い。すなわち、複数のＰＬＵにそれぞれタスクを割当てて、〃ＷＡＩＴ〃命令の実行中、ＰＬＵの演算が終るまで他のタスクを実行すると言うマルチタスクの方法である。

（ｄ）．代入文〃Ｉ＝０〃と演算文〃Ｉ＝Ｉ＋１〃

代入文〃Ｉ＝０〃や演算文〃Ｉ＝Ｉ＋１〃を用いるプログラムは次の点で注意しなければならない。これらの文をＰＬＵで実行させた場合、常に演算が行われているので、〃Ｉ＝０〃は常にＩ＝０となっており、〃Ｉ＝Ｉ＋１〃は発振を起こす。従ってこのような文は、メインＣＰＵで実行されなければならないので、コンパイラがこれを判断し、メインＣＰＵで行う命令に変える工夫が必要である。

（ｅ）．再帰関数

再帰関数は、次の２通りの方法でＰＬＵにマッピングすることができる。一つは、再帰関数をそのままＰＬＵにマッピングしてメインＣＰＵによって、添字の制御を行う方法である。もう一つは、再帰関数を添字の最大値まで展開してＰＬＵにマッピングする方法である。前者の方法は、メインＣＰＵの負荷が大きくなるので好ましくなく、この場合においても後者を提唱する。

（ｆ）．エラー検出について

ノイマン型計算機のような逐次型処理であれば、演算実行時にエラーを検出することができるが、本ＰＬＵのように入力データが確定しなくても演算を行う場合があるので、エラー検出は非常に困難である。そこで、エラー検出には次の２通りが考えられる。一つは、各演算毎の状態をメインＣＰＵが調べる方法である。この方法は逐次的に各演算毎の状態を調べなければならないので、よい方法とはいえない。もう一つは、一連の処理を文毎あるいは手続き毎にクラスタ分類を行い、入力データが確定したクラスタ中でのエラーをメインＣＰＵが検出する方法である。この場合ＰＬＵにプログラムのマッピングによってクラスタ分割できるような機構を必要とする。

（ｇ）．メインＣＰＵの機能について

言語コンパイラは、メインＣＰＵで実行され、タスク毎あるいはプログラム毎に割当てられたＰＬＵに直接マッピングしていく。従って、メインＣＰＵは従来のノイマン型計算機の持っている機能が必要である。このことは、ＰＬＵを用いた計算機に特別な機能を必要とせず、今までの計算機にメモリ装置のようにＰＬＵを増設するだけでよいことになる。

そして、ＰＬＵ上にマッピングされた制御命令は、メインＣＰＵにてインタプリタ的に実行されることになる。

また、図８および９の例ようにプログラムがＰＬＵにマッピングされると、メインＣＰＵは、ＰＬＵで実行する命令をも読み出すことになるので、無駄である。そこでＰＬＵで実行する命令をメインＣＰＵが読み出さない工夫が必要である。例えば、図８において８行めの後に、メインＣＰＵで実行する次の命令へジャンプする"ＧＯＴＯ"命令を挿入することである。これはコンパイラが自動的に判断して挿入することが必要であると思われる。

以上、ＰＬＵを用いた計算機の言語コンパイラ、問題点について述べてきた。上述したとおり、ＰＬＵを用いた計算機においては、言語コンパイラにかなりの工夫が必要であることが分かる。しかしながら数値演算が非常に多いプログラムでは、メインＣＰＵで実行する命令がかなり少なくなることから、スループットの向上が計れることは明らかである。さらに、オーバフローなどのエラー処理、プログラムのＰＬＵへのマッピングアルゴリズム、並列性や実行時間、ＰＬＵを用いた計算機の性能評価などの検討が必要であると思われるので、これらに関しては、後の報告に譲りたい。

４．まとめ

ノイマン型計算機におけるバスボトルネックや並列処理などの問題を解決する目的で提案されたプログラム可能論理演算装置（ＰＬＵ）について、ＰＬＵの処理アルゴリズムや言語レベルから見たＰＬＵの構成を述べた。このＰＬＵは、変数や定数がレジスタに容易に割当てることができるように、データレジスタ領域、プログラム領域およびデータ領域に分かれた構成となっている。そして、このＰＬＵのシミュレータをマイクロコンピュータシステム上に実現した。

次に、ＰＬＵのための言語を取り上げ、この言語コンパイラを作成した。このコンパイラによって、ＰＬＵにプログラムをマッピングする際、ＰＬＵシミュレータで行う演算命令とデータの流れを変える制御命令をも同じＰＬＵシミュレータのプログラム領域に格納する方法を取った。そして、言語レベルから見た種々の問題点をあげ、その解決方法を述べた。

このようなＰＬＵは、まだまだ解決しなければならない問題が沢山ある。今後、並列実行の可能性、プログラムのマッピングアルゴリズム、計算機としての性能評価などの研究を進めて行く予定である。

[文献]

[1]．Backus, J.:"Can programming be liberated from the von Neumann style ? A fundamental style and its algebra of programs," CACM, vol.21, No.8, pp.613-641, Aug. 1978.

[2]．例えば、小高、長島、河辺、村山：＂スーパーコンピュータＨＩＴＡＣ Ｓ－８１０アレイプロセッサシステム＂、日立評論、Ｖｏｌ．６５、Ｎｏ．８、ｐｐ．５４１－５４６（１９８３．８）。

[3]．Misunas, D.P.: "Performance Analysis of a data-flow processor", Proc. of the 1976 ICPP, pp.100-105 (Aug. 1976).

[4]．Kluge,W.E.: " Cooperating reduction machines, " IEEE, Trans. on Com., vol. C-32, No. 11, Nov. 1983.

[5]．吉岡 良雄：＂メモリステップにプログラム可能な演算機能をもつメモリ装置の提案＂、信学論（Ｄ）、 ｖｏｌ．Ｊ６６－Ｄ、Ｎｏ．１０、ｐｐ．１１５３－１１６０（１０－１９８３）。

[6]．山田、吉岡、中村、重井：＂プログラム可能論理演算装置＂、情報理論とその応用研究会第７回シンポジウム資料、ＩＩ－Ｄ－４、ｐｐ．２９９－３０４、（１１－１９８４）．

[付　録]

ここでは、＂ＩＦ＂、＂ＣＡＳＥ＂、＂ＷＨＩＬＥ＂、＂ＲＥＰＥＡＴ＂および＂ＦＯＲ＂文についてその記述と制御命令で実現する方法を示す。

（ａ）．ＩＦ文

記述：
```
ＩＦ　Ｘ，ｃｃ，Ｙ
ＴＨＥＮ
（文）
；
ＥＬＳＥ
（文）
；
```

```
実現：          90，XX，cc，YY
                92，00，ADDR1
                (THEN 文)
                E0，00，00，00
                92，00，ADDR2
      ADDR1     (ELSE 文)
                E0，00，00，00
      ADDR2     -----
(b)．CASE文
記述：  CASE X OF
        0：
        (文)
        ；
        1：
        (文)
        ；
        ---
        16：
        (文)
        ；
        END；
実現：          91，XX，00，00
                92，00，ADDR0
                92，00，ADDR1
                ---
                92，00，ADDRG
      ADDRX     92，00，ADDRY
      ADDR0     (0：文)
                E0，00，00，00
                92，00，ADDRX
      ADDR1     (1：文)
                E0，00，00，00
                92，00，ADDRX
                ---
      ADDRG     (16：文)
                E0，00，00，00
                92，00，ADDRX
      ADDRY     ---
(c)．WHILE文
記述：  WHILE X，cc，Y
        (文)
        ；
実現：  ADDR0   90，XX，cc，YY
                92，00，ADDR1
                (文)
                E0，00，00，00
                92，00，ADDR0
      ADDR1     ---
(d)．REPEAT文
記述：  REPEAT
        (文)
        UNTIL X，cc，Y
実現：  ADDR0   (文)
                90，XX，cc，YY
                92，00，ADDR0
                ---
(e)．FOR文
記述：  FOR I＝J
        TO N
        STEP K
        (文)
        ；
実現：          81，JJ，00，II
      ADDR0     (文)
                E0，00，00，00
                82，II，KK，II
                90，NN，GT，II
                92，00，ADDR0
                ------
```

なお、"IF"文、"CASE"文、"WHILE"文、"FOR"文に対するセミコロン"；"には、自動的に"WAIT"命令を付加した。これはこれらの制御文によって、変数の値の変化に対するPLUとの同期をとるためである。