高級言語向きスタックマシン上でのPascalマシンの実現と評価

REALIZATION AND EVALUATION OF PASCAL MACHINE ON A HIGH-LEVEL LANGUAGE ORIENTED STACK MACHINE

和田 耕一† 仲辻 俊元†† 金田 悠紀夫†† 前川 禎男††
Koichi WADA Toshiyuki NAKATSUJI Yukio KANEDA Sadao MAEKAWA
†神戸大学自然科学研究科 ††神戸大学工学部
Kobe University

1. まえがき

Pascal言語は、簡潔な言語構造・厳格なデータ型のチェックを行うことによる高信頼性等の特徴を有するプログラミング言語で、1968年にN.Wirthによって作成された。その用途は、研究や教育用をはじめとして、最近では実用面においても広く用いられる様になってきている。

Pascalプログラムは多くの場合、Pマシン(Pseudo-machine)と呼ばれる仮想計算機の機械語であるPコード(Pseudo-code)にコンパイルされ、そのPコードを解釈・実行することにより処理が行われる。Pマシンは、内部にいくつかのレジスタやポインタ、評価用スタック等を持つスタックコンピュータである。従って、既存の計算機上でPマシンをソフトウェアによって実現しPascalプログラムを実行する場合、以下の様な欠点が生じる。

1. ソフトウェアによるインタプリタを用いているため、Pコードのフェッチ等に時間を要する。
2. 様々な用途に用いられるスタックをソフトウェアにより実現し、スタック領域を主記憶上にとっているためスタックポインタの操作、スタックを用いた演算・メモリ参照に関する実行効率が悪い。
3. その他の多くの仮想レジスタ、仮想ポインタの管理を行う必要があり、実行時の負担が大きい。

筆者らは以前に、高級言語マシンに関する研究として、関数型言語であるFORTH言語で記述されたプログラムを高速に実行できるFORTHマシンシステムを開発した。[1][2] 本FORTHマシンは、2つの強力なハードウェアスタックや浮数用高速メモリ等を持つマイクロプログラム制御方式の計算機である。また、本マシンはPマシン内の仮想ハードウェア要素に比較的容易に対応できる構造を持っているため、本マシン上にPマシンを実現することにより、Pascalプログラムの効率良い実行が可能であると考えた。

本論文では、Pascalマシンを実現するために以下に示す2つの方法について開発を行い、さらにベンチマークテストを用いて本システムの評価を行った。

1) Pコードインタプリタをマイクロプログラム化する。
2) Pコードをマイクロ目的コードにコンパイルする。さらに、目的コード生成に関して若干の最適化を施す。

第2章では、FORTHマシンの構造・機能について述べ、第3章でPマシンの構造とPコードについて述べる。また、第4章でPマシンの実現方法の基本方針について述べる。第5章ではPマシンの実現に関する上記の2つの方法について詳述し、第6章で本システムに評価を加える。

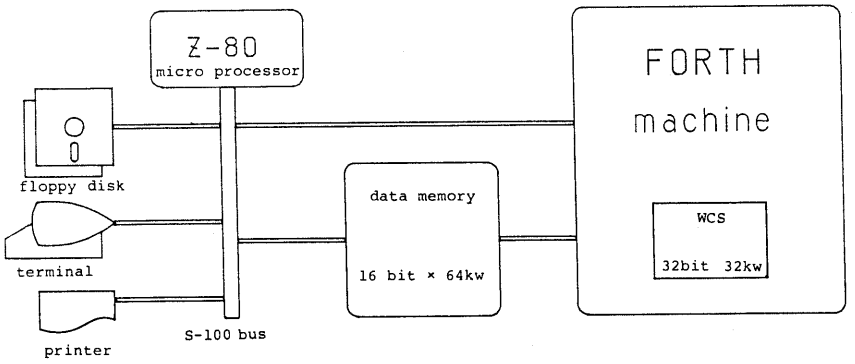


図1. FORTHマシンシステムの構成

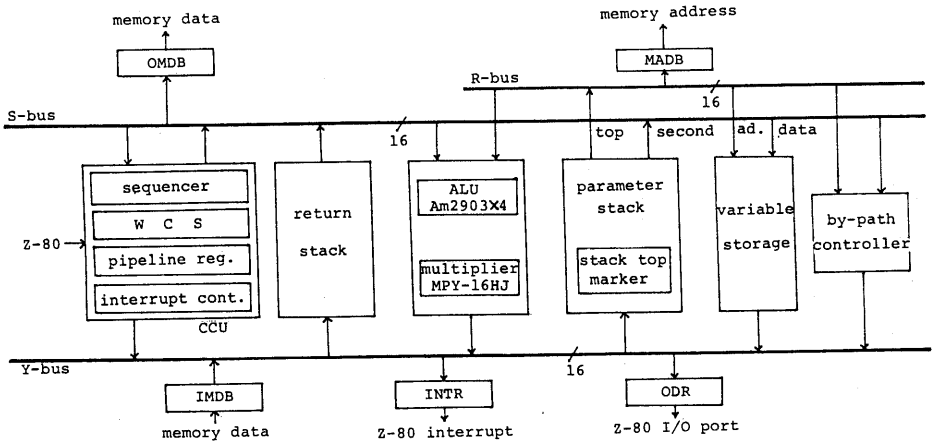


図2. FORTHマシンのハードウェア構成

2. FORTHマシンシステム

2.1 システム構成

FORTHマシンシステムの構成を図1に示す。ホストコンピュータはZ-80マイクロプロセッサで、I/Oポートを介してFORTHマシンが接続されている。FORTHマシンの初期化、起動等の制御はZ-80が行う。データメモリは16ビット、64K語の容量を持ち、Z-80とFORTHマシンの双方から参照可能になっている。データメモリとWCS†は、Z-80の主記憶の上位32Kバイトに接続され、32Kバイトを1ページとするページを切り換えることにより参照できる。

2.2 ハードウェア構成

FORTHマシンのハードウェア構成を図2に示す。本マシンは、マイクロプログラム制御方式の高級言語向き計算機である。内部バスは16ビット幅で、ソースバスとしてR-busとS-bus、デスティネーションバスとしてY-busの合計3つのバスを持っている。また、本マシンは次に示す様なハードウェアモジュールから成り立っている。

1) マイクロプログラムシーケンサのビット巾は15ビットで、32K語までのWCSの番地指定が可能である。また、マイクロ命令のフェッチの高速化のために、1レベルのパイプライン制御を行っている。
2) リターンスタックと呼ばれるハードウェアスタックを持つ。容量は16ビット×4K語である。主として、関数呼び出しやDO-LOOPの制御等に利用される。

†Writable Control Storage (書き換え可能な制御記憶)

3) ALUは、AMD（アドバンストマイクロデバイス）社の4ビットスライス素子Am2903を4個用いている。さらに、TRW社の乗算器MPY-16HJを備えている。このLSIは、16ビット×16ビットの乗算を140 nsecで実行できるものである。

4) スタックトップにある値とセカンドにある値とを同時に読み出すことが可能な、パラメータスタックと呼ばれるハードウェアスタックを持っている。スタックポインタの自動増加、自動減少機能も備えており、あらゆる二項演算に非常に有効である。また、スタックトップにある値をアドレスとしてのメモリ読み出し、さらにセカンドにある値を書き込み値としてのメモリ書き込み等のメモリ操作にも上記の機能が有効である。容量は16ビット×8K語である。

5) 変数の参照を高速化するためのバリアブルストレージを備えており、データメモリ上のデータの参照と比較して、約1/3の実行時間でデータの参照が可能である。容量は16ビット×4K語である。

6) バイパスコントローラは、R-bus上やS-bus上のデータをY-busへ転送する時に用いられる。この機能はALUを用いても実現可能であるが、バイパスコントローラを利用することにより約3/5の実行時間で転送できる。

2.2 マイクロ命令

FORTHマシンのマイクロ命令のビット幅は32ビットで、動作の中心となるハードウェアにより次に示す6種類の異なるタイプに分かれている。

i) タイプ0,1 : ALUの制御命令。
ii) タイプ2,3 : 乗算器の制御命令。
iii) タイプ4,5 : バイパスコントローラの制御命令。
iv) タイプ6 : メモリ操作命令。
v) タイプ7 : 分岐命令。

1マイクロ命令の実行に要する時間は、マイクロ命令のタイプによって異り、175 nsec～350 nsecである。また、スタックをオペランドとする演算やメモリ操作などの基本的な処理は、ほとんどのものが1～3マイクロ命令で実行できる。

3. Pマシンの構造とPコード[3]

3.1 Pマシンの構造

Pマシンは、Pascalプログラムの中間言語であるPコードを解釈、実行する仮想計算機である。処理のビット幅は16ビットで、スタック上で演算を行うスタックマシンである。

Pascalプログラムをコンパイルすると Pコードファイルが生成される。そして、実行時にはPコードファイルが読み込まれ、図3に示す様なコードセグメント、データセグメント、MSCW（Mark Stack Control Word）が主記憶上に作成される。それぞれについて以下に説明する。

1) コードセグメント：コンパイラによって作成されたPコードファイル中のプロシージャディクショナリや目的Pコード、さらにプロシージャナンバ、レキシカルレベル等で構成されている。
2) データセグメント：呼ばれたプロシージャのためのローカル変数が確保される。
3) MSCW：スタックポインタの値やプロシージャ間のリンク等、プロシージャ呼び出しに関する制御に必要な6語の情報で成り立っている。

また、Pマシンの内部には次に示す様なレジスタが想定されている。

1) SP（Stack Pointer）：メモリの上位から下位へ伸びるスタックのトップを示すポインタ。
2) MP（Most recent Pointer）：現在実行中のプロシージャのMSCWへのポインタ。ローカル変数をアクセスする時に用いられる。
3) BASE（BASE procedure）：最新のベースプロシージャのMSCWへのポインタ。グローバル変数をアクセスする時に用いられる。

4) NP (New Pointer)：メモリの下位から上位の方へ伸びるヒープ領域の先頭を示すポインタ。
5) JTAB (Jump TABle pointer)：現在実行中のプロシージャの属性テーブルへのポインタ。
6) SEG (SEGment pointer)：現在実行中のプロシージャが属するセグメントのプロシージャディクショナリへのポインタ。

3.2 Pコード

Pコードは、8ビットの命令コード部と最大3個までのパラメータ部で構成され、約100種類用意されている。大きく分けると次の6種類である。

1、定数コード：命令コード部が00～7F(16)のもので、パラメータ部を持たない。1バイトコードである。
2、メモリ操作用コード：命令コード部と最大2個までのパラメータを持つ。コード長は1バイトから4バイトである。
3、比較：パラメータ部を持たない1バイトコードである。
4、演算：パラメータ部を持たない1バイトコードである。
5、分岐用コード：1～3個のパラメータを持つ。コード長は2バイトから7バイトである。
6、プロシージャ呼出し及び関数呼出し用コード：プロシージャナンバを表すパラメータを1個持つ。コード長は2バイトである。

4. FORTHマシンによるPマシンの実現

本章では、前章で述べた様な構造を持つPマシン内の仮想レジスタやポインタ、スタック等を、FORTHマシン内のハードウェアにどの様に対応させるかについて述べる。Pascalプログラムの実行時は、スタックの参照が頻繁に行われ、変数参照のためにMP、BASE等のポインタの読み出しも頻繁に行われる。これら実行時の状況を考慮し、本システムでは以下の様に対応させた。

1) データメモリを主記憶として用いる。すなわち、インタプリタの場合、データメモリ上のPコードを解釈し実行することにな

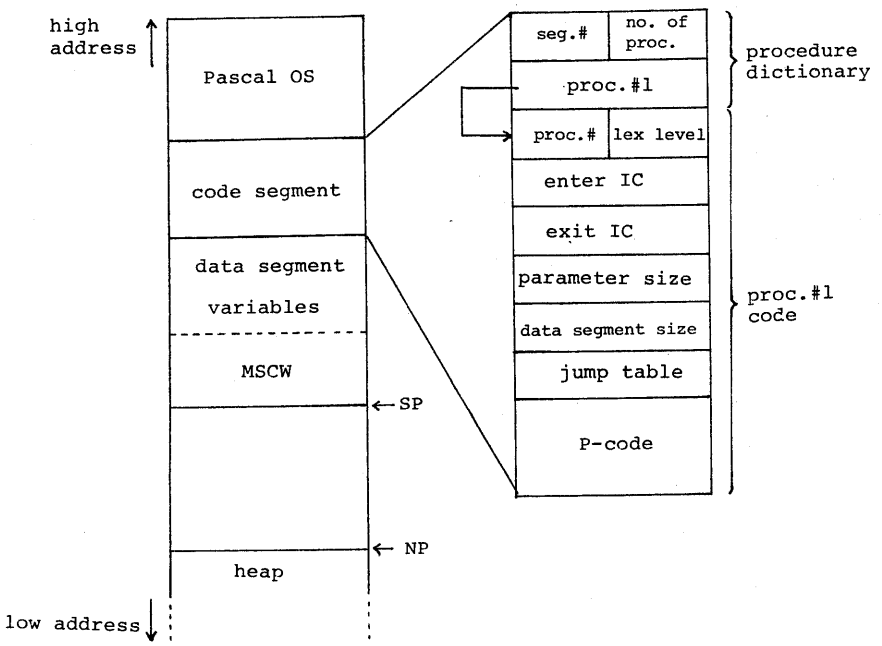
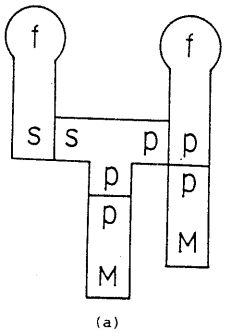
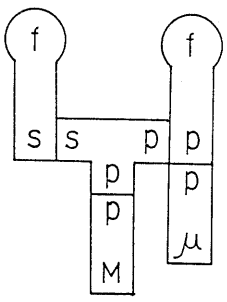


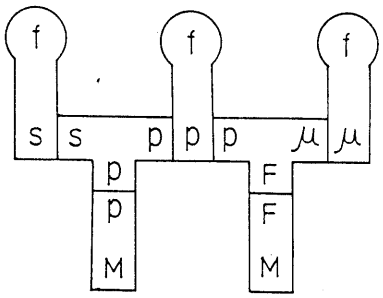
図3. Pascalプログラムの実行時のメモリ割り当て



(a)



(b)



(c)

f：ある機能fを持ったプログラム。
s：原始言語。この場合はPascal言語である。
P：Pコード。
M：機械語。
F：FORTH言語
μ：マイクロプログラム。

図4、処理系の模式図

る。
2) 実行時に非常に重要な役割りを果たすスタックには、パラメータスタックを対応させた。そうすることにより、主記憶とスタックとを異る記憶領域に分離でき、主記憶の参照回数の軽減が可能である。
3) MPやBASE等のポインタは、ALUのレジスタファイルの一部を利用する。
4) インタプリタ方式の場合、インストラクションカウンタはALUのレジスタを用いる。

5、処理系の開発

本システムで開発した処理系の模式図を図4に示す。すなわち、(a)は従来のUCSD-Pascalにおける処理系を表しており、(b)にマイクロプログラム化インタプリタ、(c)に直接マイクロコード生成型コンパイラのそれぞれに対する処理系を図示している。

5、1 マイクロプログラムによるインタプリタの実現

本処理系は、データメモリ上に置かれたPコードを、マイクロプログラムによるインタプリタで解釈、実行することにより高速化をはかるものである。本処理系はコード転送部とインタプリタ部から成り立っている。

5、1、1 コード転送部

Pascalコンパイラによってフロッピーディスク上に作成されたPコードファイルをオープンし、ファイル中にあるコードサイズ、目的Pコード、ジャンプテーブル、レキシカルレベル、プロシージャナンバ、パラメータサイズ、データセグメントサイズなど、実行時に必要なデータをデータメモリ上へ転送する。

5、1、2 インタプリタ部

インタプリタ部は、さらにフェッチ部、エミュレート部、ジャンプテーブルとに分けられる。フェッチ部、エミュレート部で用いられるMPやBASE等のポインタ、フェッチされたPコードの一時記憶、Pコードファイルのサイズの記憶、インストラクションカウンタにはALU内

のレジスタファイルの一部を利用している。

インタプリタのフローチャートを図5に示す。すなわち、インストラクションカウンタが示すアドレスによってデータメモリから1バイトPコードをフェッチする。そして、そのPコードの値が7F(16)以下であれば、定数を表すコードなので直ちにパラメータスタックへプッシュする。80(16)以上であれば、その値に対応するジャンプテーブルの内の一つへジャンプする。ジャンプテーブルはPコードの80(16)からFF(16)に対応して、WCSのアドレス80(16)からFF(16)番地に構成されており、各Pコードの行うべき処理ルーチン(エミュレートルーチン)への分岐マイクロ命令で成り立っている。また、組み込み関数用のジャンプテーブルは30(16)から3F(16)番地に、エミュレートルーチンは192(16)番地以降に構成している。WCSのメモリマップを図6に示す。

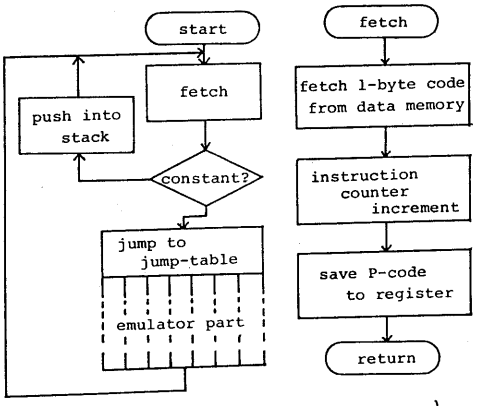


図5、インタプリタのフローチャート

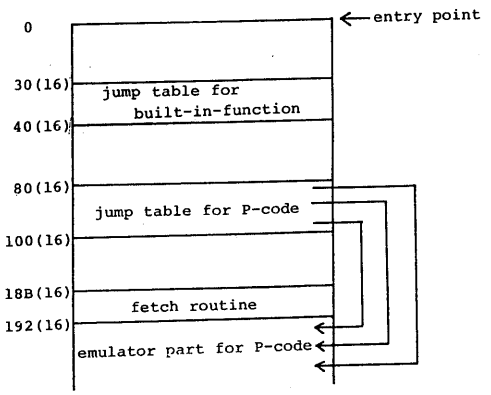


図6、WCSのメモリマップ

5.2 直接マイクロコード生成型コンパイラ

本コンパイラは、UCSD-Pascalコンパイラによって Pascalプログラムをコンパイルし、得られた目的Pコードをもとにしてマイクロ目的コードを生成するものである。Pコード1命令当り、平均2～4マイクロ命令に展開され、それらを連結することによりマイクロ目的コードを得ることができる。

基本的には以上の手順でコンパイルされるが、さらに一層の高速化を達成するために若干の最適化を行ったものを含め、次に示す様な5つのコンパイラを開発した。

1) バージョンA
UCSD-Pascalコンパイラによって得られた目的Pコードを、そのまま全てマイクロコードに展開するもの。

2) バージョンB
配列の添字の範囲のチェックを行うPコードである「CHK」を除いてコンパイルするもの。

3) バージョンC
プログラム中の繰り返し制御において、リターンスタックを有効に利用し、ループインデクスの参照においてもリターンスタックを用いる様にコンパイルするもの。

4) バージョンD
バージョンCをさらに拡張し、二重構造から成るくり返し文の制御において、外側ループのループインデクスの参照にもリターンスタックを用いるもの。

5) バージョンE
バージョンDに加えて、while-do文における繰り返し制御にリターンスタックを利用したもの。

6、システムの評価

本章では、各Pコードを一命令実行するために必要なマイクロステップ数から本システムの静的特性について述べ、マイクロプログラム化インタプリタと直接マイクロコード生成型コン

パイラのそれぞれに対してベンチマークテストを行い、その結果をもとに動的特性の評価について述べる。

6.1 静的特性の評価

表1に、Pコードに対応するマイクロ命令のステップ数を示す。表では、インタプリタ方式とコンパイラ方式におけるステップ数とその比を示している。

表より、インタプリタ方式の場合、9ステップから55ステップで、コンパイラ方式では1ステップから9ステップ必要としていることがわかる。また、ステップ数の比は2.2から55である。制御文をコンパイルした時に生成されるPコードの「UJP」において、最も大きい比率が示されている。これは、インタプリタ方式では実行時にジャンプテーブルの参照と実効アドレスの計算が伴うのに対し、コンパイラ方式では、実アドレスへのジャンプ命令を直接生成しているためである。また、その次に大きい31という比率を示しているのは、2バイト定数のスタックへのプッシュ命令である。

これらのPコードは、多くの場合プログラム中に頻繁に現われる。従って、Pコードをマイクロ命令にコンパイルすることは、マイクロ命令の実行ステップの減少に非常に有効であることがわかる。また、コンパイラ方式の場合では、多くのPコードが1から4マイクロステップで実行できることが示されており、仮想計算機であるPマシンとFORTHマシンとの整合性が高いと考えて良いと思う。

6.2 動的特性の評価

ベンチマークプログラムを実行した時の実行時間の測定結果を表2に示す。表では、本システム上でのマイクロプログラム化インタプリタ(μI-Pascal)及び直接マイクロコード生成型コンパイラ(F-Pascal)の5種類の最適化バージョン、Z-80マイクロプロセッサ上のUCSD-Pascal、他の数種のPascalシステム、さらにFORTH、C、FORTRAN言語で同等のプログラムを記述し、実行した時の実行時間の測定結果を示している。

ベンチマークプログラムは、次の2種類を用いた。(付録参照)

表1. インタプリタ方式とコンパイラ方式との各Pコード実行ステップ数

種類	Pコードの例		インタプリタ	コンパイラ	比
定数	SLDC	1	9	1	9
	LDCI	1000	31	1	31
ロード	LDO	200	38	4	9.5
	LAO	3	26	3	8.7
	SIND	0	14	1	14
ストア	STL	200	38	4	9.5
	SRO	20	27	4	6.8
	STO		17	4	4.3
演算	ADI		14	1	14
	SBI		14	1	14
	MPI		15	2	7.5
比較	LEQI	真	18	5	3.6
	GEQI	偽	19	6	3.2
ジャンプ	FJP	真	16	2	8
	FJP	偽	28	2	14
	UJP		55	1	55
	CHK		20	9	2.2
	CSP	2	23	1	23

表2. ベンチマークプログラムの実行時間(sec)

	ソーティング	素数
マイクロインタプリタ μI-PASCAL	75.437	26.269
F-PASCAL バージョンA	19.037	3.339
F-PASCAL バージョンB	13.529	2.914
F-PASCAL バージョンC	12.565	2.331
F-PASCAL バージョンD	10.776	
F-PASCAL バージョンE		1.598
UCSD PASCAL (Z-80)	1705.6	233.9
UCSD PASCAL マイクロエンジン		63.0
INTEL PASCAL-86		9.05
FORTH (FORTHマシン)	2.705	1.497
BD.S C	251.6	50.2
Whitesmith C, Z80		15.6
FORTRAN (ACOS-900)	3.201	0.737

文献(7)より

1、素数の計算

0から8190までの間にある素数の個数を求めるプログラムである。方法は、0から8190までの数に対応するサイズ8191の配列を確保し、各要素にその数が素数か否かの真偽値を持たせる。まず、全要素を真にセットしておき、3以上の奇数に注目し、その数に関連する素数でない数のフラグを全要素にわたって偽にセットしながら素数の個数を数える。

2、ソーティング

配列内のあるデータとそれ以降のデータとの比較を行い、後者の方が大きい場合はその都度、前者とのデータの交換を行う方法をとっている。1,2,…,1000のデータを記憶させ、1000から1へと逆順に並べ換えを行う。

測定の結果、Z-80 UCSD-Pascalと比較して、インタプリタをマイクロプログラム化することにより8.9～22.6倍、直接マイクロコンパイラを用いることにより70.1～89.6倍の実行速度が得られた。

次に、直接マイクロコード生成型コンパイラ（バージョンA）による測定結果を用いて評価を行う。

表3に、総ステップ数と平均サイクルタイムを示す。表より、1マイクロステップ当り平均220 nsecで実行されていることがわかる。

また、ある特定のハードウェアを用いたオペレーション回数の総ステップ数に対する比率を利用率とし、評価の基準とした。表4にマイクロ命令のタイプ別の利用率、表5にハードウェアモジュールの機能別の利用率を示す。

表4より、タイプ4とタイプ5、すなわちバイパスコントローラの制御命令が最も多く利用されていることがわかる。これは、データ転送の占める割り合いが大きい事を示しており、高速データ転送用のバイパスコントローラが有効に利用されていることを表している。タイプ0とタイプ1はALUの制御命令で、演算に用いられる他、レジスタファイルの読み出しに利用されている。タイプ2と3の乗算器制御命令は低い利用率を示しているが、配列のインデクスの計算に用いられるので、多次元配列の操作を

表3．総ステップ数と平均サイクルタイム

	ソーティング	素数
総ステップ数	84,975,963	12,268,004
平均サイクルタイム（ns）	224.0	216.5

表4．マイクロ命令のタイプ別利用率

	利用率（%）	
	ソーティング	素数
タイプ0	22.9	23.6
タイプ1	1.2	3.4
タイプ2	3.5	1.9
タイプ3	0.0	0.0
タイプ4	21.2	21.1
タイプ5	30.0	25.9
タイプ6	10.6	12.2
タイプ7	10.6	11.9

表5．オペレーション別利用率

	オペレーション	利用率（%） ソーティング	素数
パラメータスタック	トップ ソース	28.8	31.5
	トップ デスティネーション	61.2	64.7
	セカンド ソース	31.2	28.0
	セカンド （）+	26.5	24.9
	セカンド -（）	60.0	62.8
リターンスタック	ソース	1.2	3.2
	デスティネーション	1.2	1.9
	（）+	1.2	1.9
	-（）	1.2	1.9
PSIR	ソース	3.5	1.9
	デスティネーション	3.5	1.9
メモリ	読出し	8.2	9.2
	書込み	2.4	4.2
ジャンプ	無条件	1.2	2.7
	条件	9.4	9.2
レジスタ	R13	0.6	0.7
	R14	6.5	12.1

多く含むプログラム等では一層有効に利用されると考えられる。

表5より、パラメータスタックの利用率は非常に高く、スタックトップの読み出しが約30%書き込みが約63%、スタックのセカンドの読み出しが約30%の割り合いである。スタックのトップとセカンドの値の読み出しが同じ頻度で行われている点は興味深く、本マシンのパラメータスタックの機能が有効に利用されていることを表している。スタックポインタの読み出し後自動増加と自動減少後書き込みの機能もスタック上のデータへのアクセスと同程度の利用率を示しており、不可欠と考える。

リターンスタックは、比較的低い利用率を示しているが、マイクロ目的コードの最適化を行った場合に効果的に利用されているのは前述の通りである。

パラメータスタックインデクスレジスタ(PS-IR)は、スタックトップからあるオフセットを持ったスタック中のアドレスからデータを読み出す時に用いられるレジスタで、ここでは配列の添字の配用のチェックを行う時に用いられているのみである。

7. むすび

強力なハードウェアスタックを有するFORTHマシンシステムによるPascalマシンの実現とその静的、動的特性の評価について述べた。実現方法については、FORTHマシンはPマシンの持つ仮想レジスタやスタックの構造に効率良く対応できるハードウェアを持っていることを示し、その対応関係について述べた。また、ベンチマークテストの結果、最適化を施した直接マイクロコード生成型コンパイラの場合、Z-80マイクロプロセッサ上のUCSD-Pascalに比べ、最高158倍の実行速度が得られたことを示した。その結果、本FORTHマシンの構造が複数の言語に対して有効であることが確認できた。

なお、既に決定されているPコードに合わせて本処理系を開発したため、冗長はマイクロ命令を実行している部分もあり、リターンスタックを有効に利用するPコードがないなどの点から、Pコード自体を若干修正・追加することにより、さらに一層の高効率化がはかれると思う。

文献

(1) 和田, 金田, 前川：" FORTHマシンシステムのシステム設計とハードウェア構成", 信学技報, Vol. J65-D No. 3 (1982-03).

(2) 和田, 金田, 前川：" FORTHマシンシステムの評価", 信学技報, Vol J65-D No.3 (1982-03).

(3) K. A. Shillington, G. M. Ackland：" UCSD Pascal Version 1.5", Institute for Information Systems. (1978-09)

(4) D. W. Barron：" Pascal - The Language and its Implementation", John Wiley & Sons Ltd. (1981)

(5) 中田育男：" コンパイラ", 産業図書(1981-09).

(6) 中田育男：" スタック・マシンのための最適コード生成のアルゴリズム", 情報処理学会論文誌, Vol. 22 no. 5 (1981)

(7) Jim Gilbreath：" A High-Level Language Benchmark", BYTE Publications Inc. (1981-09)

付　録

(1) ソーティングのプログラムのリスト

```
program SORTING;

  type list = array [1..1000] of integer;
  var  idata : list;
       i,j,k,l,m,work : integer;

    begin
      for i:=1 to 1000 do idata[i]:=i;
      for m:=1 to 999 do
        begin k:=m+1;
          for j:=k to 1000 do
            if idata[m] < idata[j] then
               begin  work:=idata[m];
                      idata[m]:=idata[j];
                      idata[j]:=work
               end
        end
    end.
```

(2) 素数を求めるプログラムのリスト

```
( Eratosthenes Sieve Prime Number Program in PASCAL )
program PRIME;

const
  size =8190;

var
  flags : array[0..SIZE] of boolean;
  i,prime,k,count,iter : integer;

begin
writeln('10 iteration');
for iter := 1 to 10 do begin
        count :=0;
        fillchar(flags,sizeof(flags),true);
        for i := 0 to size do
                if flags[i] then begin
                        prime := i+i+3;
                        k := i + prime;
                        while k <= size do begin
                                flags[k] := false;
                                k := k + prime
                                end;
                        count := count + 1
                        ( writeln(prime) )
                        end;
        end;
writeln(count,' primes')
end.
```