パイプライン型字句解析プロセッサの設計と実現†

板野肯三†† 佐藤 豊††† 山形朝義††††

逐次ハッシュ法に基づく連想記憶上に実現された字句テーブルと，文字レベルの逐次検索機構を用いた，高速のパイプライン型ハードウェア字句解析プロセッサを開発した．本プロセッサでは，名前，文字列，定数などすべての字句レベルのトークンを可変長の文字の列として統一的に認識し，コンパクトなコードに圧縮しタイプを付加して出力する．このうち，字句レベルの構文に基づいてトークンを認識する過程と，認識されたトークンを字句テーブルを用いてコード化する過程をパイプライン化して，文字の列をストリームとして高速に処理する．複数の言語に対応するために，実行の対象となる言語の字句レベル構文を制御テーブルの形で実行時にロードする方式を採用し，制御系のハードウェアの柔軟化を計っている．本方式の性能は，CやPASCALのソースプログラムを使って字句解析の実行速度を測定して評価した．

1. まえがき

近年のLSI技術の進歩は，従来はコスト価格比の点で実現が困難であった複雑な論理を内部に持つシステムのハードウェア化を容易にしつつある．しかし，プログラムを開発する際に最も重要な道具の一つであるコンパイラは，依然としてソフトウェアのみで実現されているため，その実行の逐次性によって処理速度が縛られている．また，高度な対話的プログラミング環境を実現するにはソースプログラムの直接実行系が最適であるが，ハードウェアによるサポートなしには，実用上十分な実行速度が得られない[1)~4)]．

これらの言語処理系においては，ソースコードの文字情報を直接取り扱う字句解析部に，特に高速化が要求される[5)]．そこで，主記憶からの文字の読み出し速度と同程度の速度の字句解析の実現を目標として，ハッシュ技法を用いた連想記憶による字句テーブルと，ハードウェアで文字単位の検索を高速に行うアルゴリズム[8)]を使用したパイプライン型の字句解析プロセッサを設計し，実現した．実現に際しては，言語に依存する字句レベルの構文や予約語を書き換え可能な制御テーブルとすることによって，複数の言語への対応とハードウェアの制御機構の単純化を実現した．

本論文では，この字句解析プロセッサのハードウェアの構成方式と字句解析アルゴリズムを述べ，実際に実現されたC言語とPASCAL用の字句解析プロセッサの動作解析と性能の評価を示す．

2. ハードウェアの基本構成と機能

字句解析プロセッサ（LP: Lexical Processor）の基本的機能は，一次元の文字列を入力して字句レベルのトークンを認識し，可変長の文字列からなるトークンをコンパクトなコードに変換し，そのタイプを付加して出力することである．この一連の作業をハードウェアの並列性を生かして高速に処理するために，LPのハードウェアを図1に示すように，文字のクラス認識部とトークンの境界認識部，予約語検査およびコード生成部などに分割し，これらをパイプライン型に結合して構成した．さらに，言語に関して汎用性を確保するために，言語に依存する文字クラス，字句構文，予約語などをテーブル化してこれらのサブユニットを制御する．これらのテーブルは，対象言語に応じて動的に書き換えることができるように設計した．

2.1 トークンの認識

LPに入力された文字の列は，まず，字句レベルの構文に従ってトークンの列に分解される．これを行うトークンの境界認識部は，現在認識中のトークンの種類を状態とし，入力された文字のクラスに従って遷移する，有限オートマトンとして実現されている．このオートマトンの状態遷移は字句構文テーブルで制御され，入力された文字のクラスは，文字クラステーブルで生成される．トークンの境界が認識された状態で，コード化部にそれを知らせる．

2.2 トークンのコード化

トークンのタイプ分けとコード化は，トークンの境界認識と並列に行われ，境界が認識された時点で遅延

---

† Design and Implementation of a Pipelined Lexical Processor by KOZO ITANO (Institute of Information Sciences and Electronics, University of Tsukuba), YUTAKA SATO (Doctoral Program in Engineering, University of Tsukuba) and TOMOYOSHI YAMAGATA (Division of Research Facilitation, University of Tsukuba).

本研究は文部省科学研究費補助金・試験研究(1) 58850063および61850061によって補助された．

†† 筑波大学電子情報工学系
††† 筑波大学工学研究科
†††† 筑波大学研究協力課

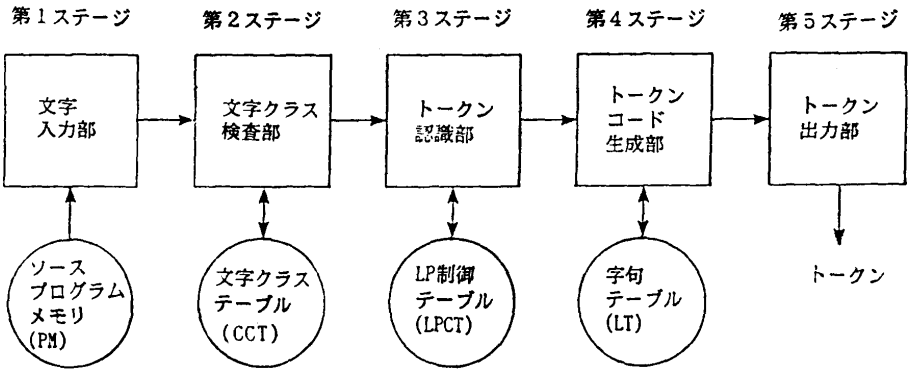


図 1 字句解析プロセッサのパイプラインの構成
Fig. 1 Pipelined organization of the lexical processor.

なく，そのトークンを識別する一意的なコードとそのタイプを出力する．これは，後述する逐次ハッシュ法[8]を用いて構成された特殊な記号表として構成された字句テーブルを使用することによって，実現されている．

この字句テーブルには言語で定義されている予約語および特殊記号があらかじめ登録されており，それ以外の名前や定数は最初の出現時に登録される．個々のトークンは，この字句テーブルのエントリ番号で一意的にコード化され，そのタイプは対応する字句テーブルのエントリに記録される．

3. トークンの認識機構

一般に言語の字句レベルの構文は正規表現で記述され，その認識機構はその正規表現と等価な有限オートマトンで実現できる．そこで，トークンの認識機構は，状態遷移表で制御されるハードウェアによる有限オートマトンとして実現した．

トークン認識機構のハードウェア構成を**図 2** に示す．この図で，L_STATE レジスタは，現在認識中のトークンの種類をオートマトンの状態として保持し，C_CLASS レジスタは，オートマトンへの入力になる，入力文字のクラスを保持する．そして，このオートマトンの状態遷移と，各状態遷移時に行うべき動作を記録した LP 制御テーブル (LPCT) によって，状態遷移とそれに伴う LP 全体の動作が制御される．

3.1 文字クラステーブル

入力文字からそのクラスを知るために，文字クラステーブル (CCT) を用いる．CCT は，文字コードをアドレスとして与えると，その文字のクラスをデータとして出力するメモリとして構成される．このメモリの容量は使用する文字コード体系に依存し，ASCII

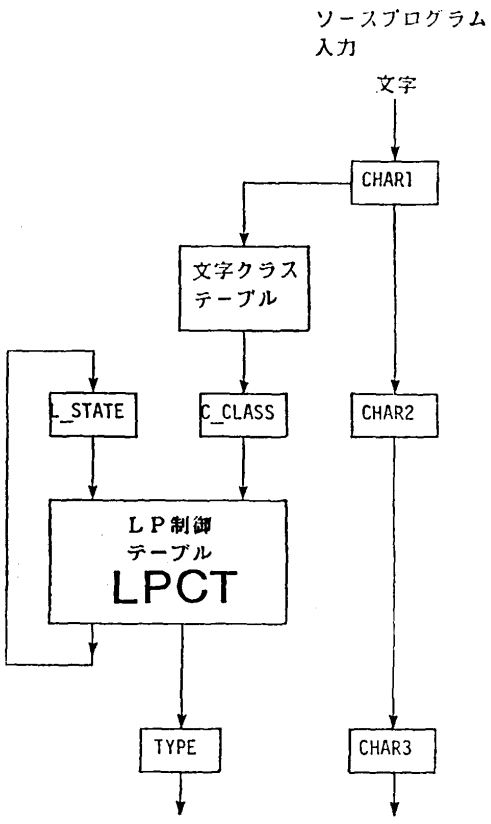


図 2 トークンの認識機構
Fig. 2 Token recognition mechanism.

の場合 128 である．一方，このメモリの幅を決定する文字クラスの数は，主に対象言語の使用する特殊記号の数に依存し，実現例では PASCAL で 13，C 言語で 25 であった．

3.2 LP 制御テーブル

LP 制御テーブル (LPCT : LP Control Table) は，トークン認識のためのオートマトンの状態遷移を制御

する情報を中心として，各々の状態遷移時に LP の各部に送るべき制御情報を保持する．LPCT の各エントリは，オートマトンの状態遷移図の状態遷移を表す各矢印に対応して与えられ，現在の状態番号 L_STATE と入力された文字のクラス C_CLASS をテーブルのインデックスとして読み出される．したがって，LPCT のサイズの最小値は言語の字句構文の複雑さに依存する状態数と，文字クラスの数の積で決まり，これは PASCALでは117，言語Cでは 650 である．

LPCT の各エントリは，複数の固定長フィールドで構成され，各フィールドは 0 でない場合に意味を持つ．LPCT のすべてのエントリは，状態遷移のための制御情報として，遷移先の状態番号（すなわち次の L_STATE の値）を持つ．また，トークンの末尾が次のトークンの先頭で検出されるような遷移の場合には，入力文字を再度入力することを指示するフラグが立てられる．

一方，LP の動作を制御する情報として，トークンの境界を認識した時の遷移には，認識の終了をトークンコード生成部に知らせることを指示するフラグが立てられる．また，トークンをコード化する字句テーブル検索機構を制御するために，トークンの先頭の認識時には検索機構の初期化を指示するフラグ，また名前または定数の認識中にはその登録を指示するフラグが立てられるとともに，そのタイプが示される．

3.3 トークン認識機構の動作

上述のように，LPCT は LP というプロセッサ全体を制御するための水平型マイクロプログラムの一種と考えることもでき，L_STATE がそのマイクロプログラムカウンタ，C_CLASS がマルチウェイジャンプのためのインデックスレジスタの役割を果たしている．このマイクロプログラムは，次の5ステージで実行される．

（1） 入力文字のコードをアドレスとして CCT から文字クラスを読み出し，C_CLASS に設定する．

（2） L_STATE と C_CLASS をインデックスとして LPCT から制御情報，遷移情報を読み出す．

（3） LPCT からの制御情報出力をコード生成部に送る．同時に，LPCT からの遷移情報出力を L_STATE に設定する．

（4） 字句テーブルを検索してトークンをコード化する．

（5） コード化したトークンにタイプを付加して出力する．

実際には，(1)から(5)を5相のパイプラインの各ステージとして実現し，実行を重ね合わせることで，単一のクロックでこれらを実行する．この際，実行のネックとなり，かつ不定の実行時間を要するのが，（4）のステージにおけるコード生成のための字句テーブルの検索である．

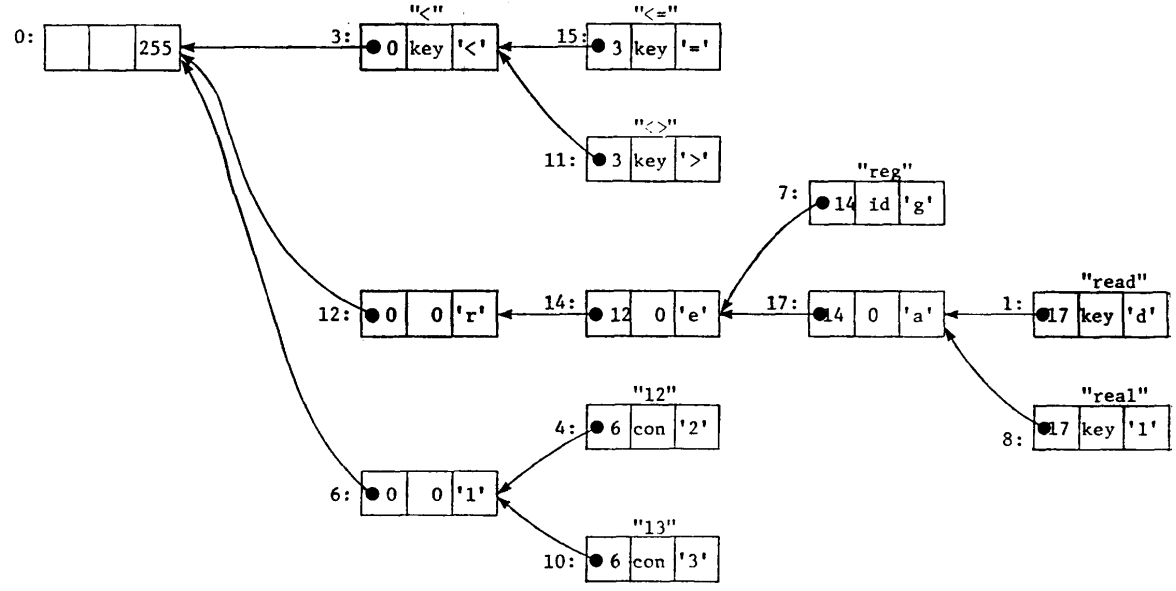


図 3 文字列の表現形式
Fig. 3 String representation scheme.

4. トークンのコード化機構

文字列として入力されたトークンはすべて字句テーブル (LT) の検索・変換機構によって，一意的なコードに変換され，そのタイプが付加される．LT は特殊記号と予約語および名前，数値定数や非数値定数を同一の機構で登録・検索する．

前述のように，LT の検索速度は LP 全体の処理速度を縛るため，性能の点では LT を実際の連想記憶で実現するのが理想であるが，これは現時点では性能価格比の面で現実的でない．そこで，逐次的に入力される文字列の最後の文字の入力と同時にその検索が終了するという意味で，連想記憶とほぼ同等の効果を持ち，かつ単純なハードウェアで実現可能な逐次ハッシュ法[8]によって LT を構成した．

4.1 字句テーブルの構成

逐次ハッシュ法で構成されたテーブルに登録された文字列は図3のように表現される．文字列を構成する各文字は後向きポインタ（これは遷移前の状態に相当する）でリンクされる．前向きのリンクは，現在の状態と次の文字に関するハッシュ値である．図が示しているように，先頭に共通の部分列を持つ文字列間ではその部分列を共有する．図の例では 8 種類の文字列 "<", "<=", "< >", "read", "real", "reg", "12", "13", が表現されているが，例えば，"re" は "read", "real", "reg" に共有されている．このような共有は，逐次ハッシュ法にテーブルの圧縮効果をもたらしている．

逐次ハッシュ法で構成された，LT の構造を図4に示す．この図は，図3で示した文字列の例を格納した状態を表している．テーブルの各エントリは，文字列を構成する文字ごとに与えられ，1文字分の文字コード (character)，逆方向ポインタ (pointer)，およびその文字を末尾とする文字列のタイプ (type) が記録される．未使用のエントリでは文字コードは0である．逆方向ポインタは，その文字を含んでいる文字列での一つ前の文字のインデックスであり，それが先頭文字である場合には0である．文字コードと逆方向ポインタは，ハッシュのためのキーであり，ハッシュ後の一致の検査に使用される．タイプは，その文字を末尾とする文字列の型を示し，それが0であるときはその文字で終了する文字列は登録されていない．

4.2 コード化機構のハードウェア構成

図5にトークンのコード化機構のハードウェア構成

	character	type	pointer
0	255		
1	'd'	key	17
2	0		
3	'<'	key	0
4	'2'	con	6
5	0		
6	'1'	0	0
7	'g'	id	14
8	'1'	key	17
9	0		
10	'3'	con	6
11	'>'	key	3
12	'r'	0	0
13	0		
14	'e'	0	12
15	'='	key	3
16	0		
17	'a'	0	14
・			
・			
・			

図 4 字句テーブルの構造
Fig. 4 Structure of the lexical table.

を示す．これは基本的には，ハードウェアによる字句テーブルのハッシュ検索機構を中心にして構成されている．ハッシュの方式はオープンハッシュ法を採用し，ハッシュ関数で生成されたインデックスに衝突が起こった時はリハッシュを行う．ハッシュ関数は，ハードウェアで簡単に実現できることを考慮して，前述の状態番号（エントリのインデックス）をビット反転し，これと文字コードの排他的 OR をとって生成する．リハッシュはキーに素数を加算して行う．比較の対象となるキーはポインタと1文字のコードのみであり，ハードウェアの構成は全体として単純なものとなっている．

4.3 コード化機構の動作原理

トークンのコード化の過程は，それぞれ1クロックで実行可能な二つのパイプラインステージで構成した．

(i) 直前までの部分文字列へのポインタと，入力

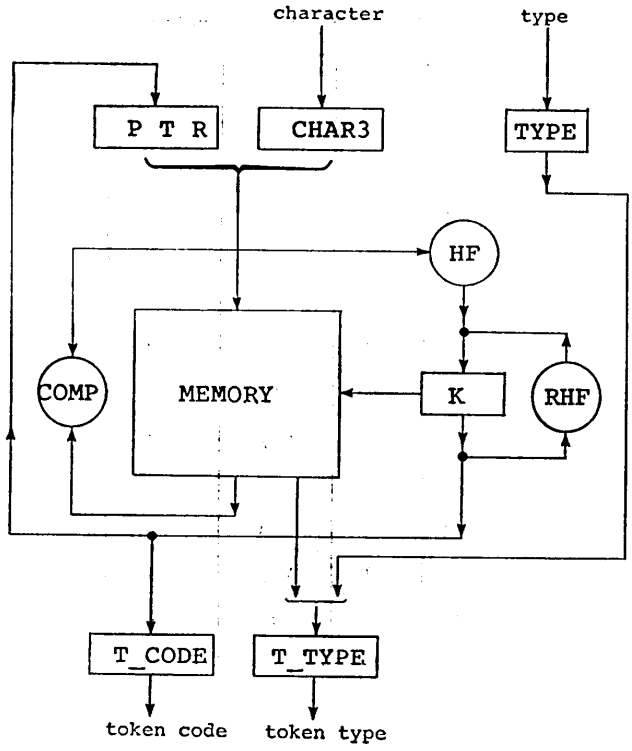


図 5 トークンのコード化機構
Fig. 5 Token code generation mechanism.

された文字コードをハッシュして字句テーブルの対応するエントリを読み出す.

(ii) エントリが検索の対象に一致し，トークンの境界であることの指示がある時はトークンのコードとタイプを出力する.

これに対する例外的な処理は二つあり,

(iii) 読み出したエントリが検索の対象に一致しなかった場合は，リハッシュして(i)のフェイズを再実行する.

(iv) 空のエントリが読み出されたときは，このエントリに現在の文字に対応する情報を書き込んで文字列を拡張する.

この(iii)と(iv)のうちいずれかが発生すると，字句テーブルの検索は1クロックでは実行できなくなるので，他のパイプラインステージを待ち状態にする.

5. 字句解析プロセッサの実現とその性能解析

字句解析プロセッサのための制御テーブルを，PASCAL[10] と C[11] に対して設計し，そのテーブルの静的な構成を調べた．次にこれらの制御テーブルを使用して，ハードウェア字句解析プロセッサと等価に動作し，各種の動作特性を収集できるレジスタ転送レベルのシミュレータを作製し，UNIX/4.2 BSD[9] 中の C, PASCAL 各言語で書かれたソースプログラムを字句解析した結果をもとに，性能の解析を行った.

5.1 制御テーブルの構成

表1に，PASCAL と C を対象として設計された LP の制御テーブルの規模を示す．表中で文字クラスの数は分類された文字の種類，トークンの種類は言語に定義された予約語と特殊記号に名前と定数の種類を加えた数，ステート数は字句構文認識のためのオートマトンの状態数，エントリ数は LP 制御テーブルの大きさを示す.

表が示すように，PASCAL に比べて C では文字クラスの数および状態の数が非常に多い．これは C が多くの特殊記号を使用していることに起因している．既に述べたように，文字クラス数と状態数の両者の積で LPCT の最小サイズが決まることが問題である．しかし，C の場合でもそのサイズはたかだか 650 であり，実現上何ら問題とはならない.

5.2 測定の結果と解析

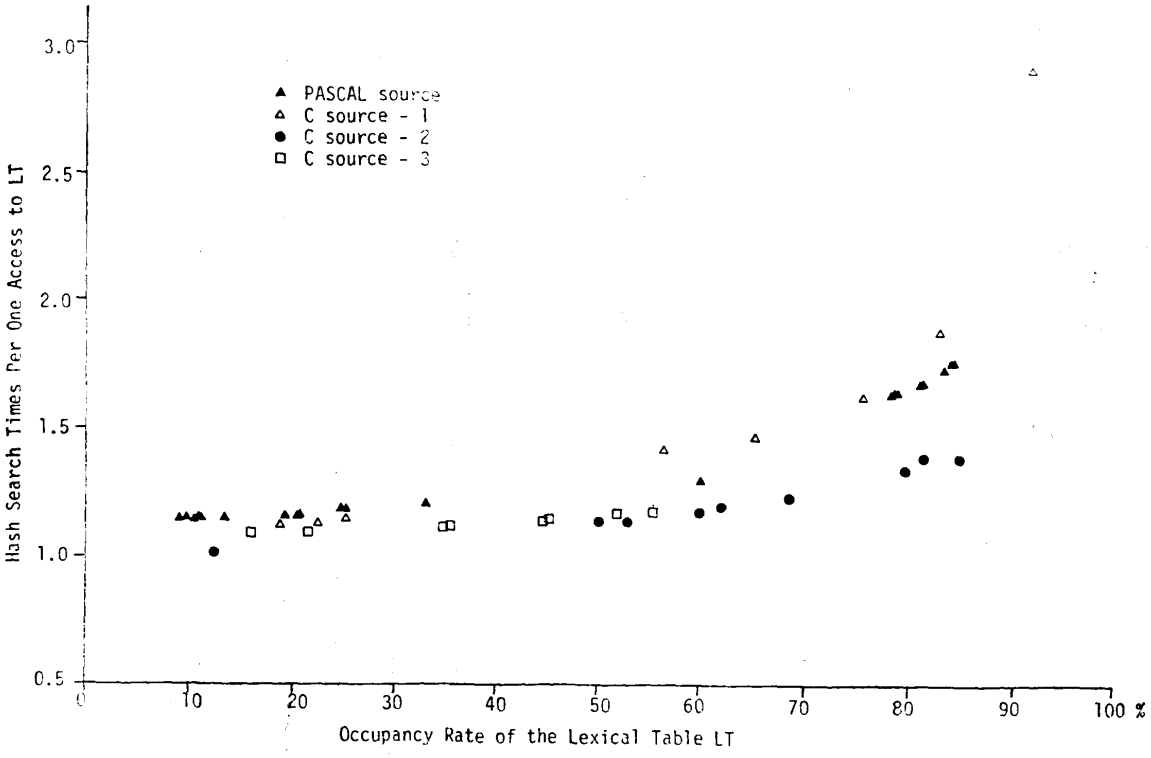
測定に使用したソースプログラムは，四つの傾向の異なるものを選んだ．このうち，一つは PASCAL (PASCAL コンパイラの一部) のソースプログラム，残り三つは C (UNIX カーネルほか) のソースプログラムである．測定は，定数の使用頻度と LT の検索速度について行った.

表 1 LP 制御テーブルの規模
Table 1 Size of LP control table.

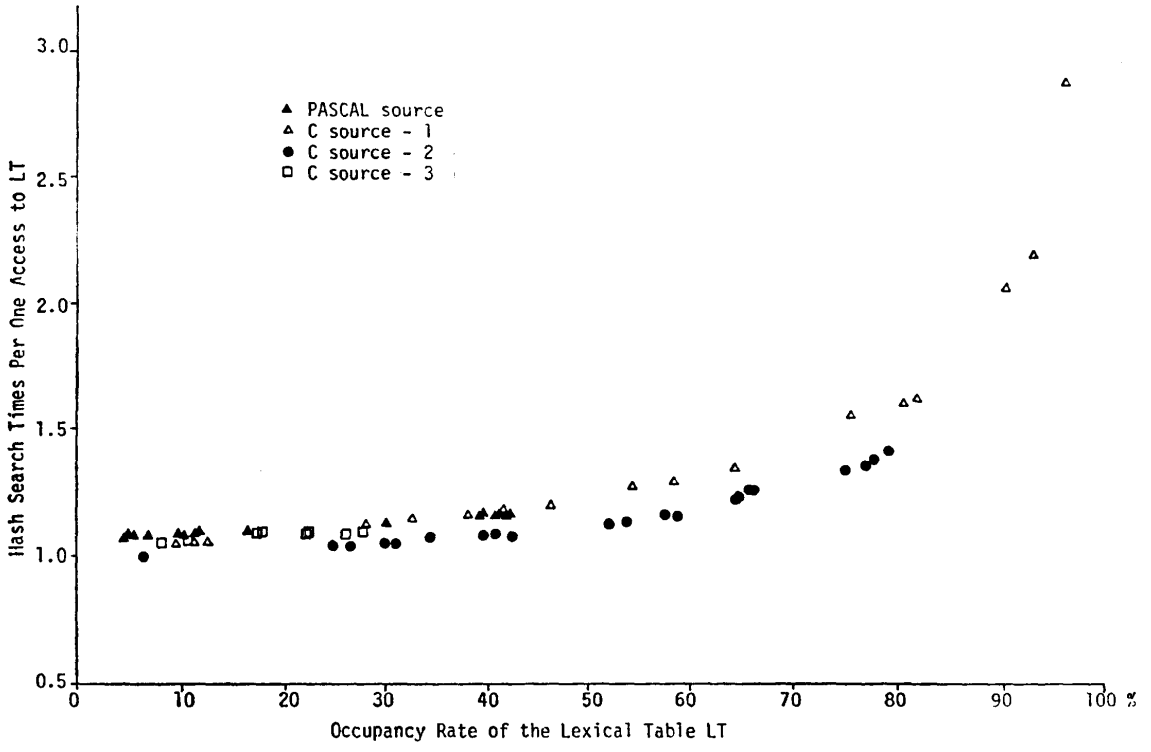
	PASCAL	C
トークンの種類	59	86
文字クラス数	13	25
ステート数	9	26
エントリ数	117	650

表 2 定数の出現頻度
Table 2 Occurrence of constants.

順位	定数名	出現回数 (2515 回中)	出現率 %
1	0	847	33.7
2	1	652	25.9
3	2	315	12.5
4	8	129	5.1
5	3	101	4.0



(1) Lexical Table Size = 2048



(2) Lexical Table Size = 4096

図 6 字句テーブル参照時の平均ハッシュ探索回数
Fig. 6 Average Hash times per one access to the lexical table.

（1） 定数の使用頻度

UNIX のカーネル部のソースプログラムで測定した定数の使用頻度の一部を**表 2** に示す．この表が示すように，プログラム中で使用される定数の大部分は 1 桁の数であり，0, 1, 2 で全体の約 71% を占めている．また，このプログラム中で使用されていた定数は全体で 202 種類，のべ 2,515 回使用されていたが，その出現頻度は普通の名前に比較すると非常に少なく，LT 中に登録されたエントリ数としては 69 個で全エントリのうち約 0.5%であり，無視しうる程度である．

（2） LT の検索速度

実際のサンプルプログラムの字句解析で LT の検索時に使用したハッシュの性能について**図 6** に示す．LT の大きさは，2K（図6-(1)）と 4K（図6-(2)）で測定した．横軸は LT の占有率，縦軸は 1 回の検索に要したハッシュの平均回数を示している．最終的に選んだハッシュ関数がこの応用に適合しているために，LT の占有率が高くなっても衝突はほとんど起きず，理想的な連想検索の機構が実現されていると言える．例えば，占有率 50% でのリハッシュの平均回数は 0.2〜0.4 程度であり，実用的には無視できる値である．これは，実際のソースプログラムに頻繁に現れる特殊記号や予約語などが LT の占有率の低い状態であらかじめ登録されているために，頻繁に現れるものほど衝突の起こりにくい場所にハッシュされているためであると考えられる．実際名前だけに限ってキーの衝突率を測定してみると，上述の占有率 50% で 0.3〜0.6 程度と少し高くなるが，それでも十分小さな値である．

（3） パイプラインの効率

5 ステージで構成されている字句解析プロセッサの

```
                pipeline phases
clock   p1    p2    p3    p4    p5

  1      p   [ ]   [ ]   [ ]   [ ]
  2      r     p   [ ]   [ ]   [ ]
  3      o     r     p   [ ]   [ ]
  4      g     o     r     p   [ ]
  5    [g]   [o]   [r]    r*   [ ]
  6      r     g     o     r   [ ]
  7      a     r     g     o   [ ]
  8      m     a     r     g   [ ]
  9    [m]   [a]   [r]    r*   [ ]
 10      (     m     a     r   [ ]
 11    [(]   [m]   [a]    a*   [ ]
 12    [(]   [m]   [a]    a*   [ ]
 13      i     (     m     a   [ ]
 14    [i]   [(]     (     m   [ ]
 15      n     i     (   [m]   [ ]
 16      p     n     i     (    7b {program
 17      u     p     n     i    28 {(}
 18      t     u     p     n   [ ]
 19      ,     t     u     p   [ ]
 20      o     ,     t     u   [ ]
 21    [o]   [,]     ,     t   [ ]
 22      u     o     ,   [t]   [ ]
 23      t     u     o     ,    1dd {input}
 24      p     t     u     o    2c {,}
 25      u     p     t     u   [ ]
 26      t     u     p     t   [ ]
 27      )     t     u     p   [ ]
 28      ;     )     t     u   [ ]
 29    [;]   [)]     )     t   [ ]
 30     NL     ;     )   [t]   [ ]
 31      v    NL     ;     )    9d {output}
 32      a     v    NL     ;    29 {)}
 33      r     a     v    NL    3b {;}
  .
  .
  .

p1:fetch; p2:class; p3:separation;
p4:LT search; p5:token output;
[ ] indicates the phase waiting
    for the synchronization.
*   indicates the phase where
    a rehash occured.
```

図 7 パイプライン処理の例
Fig. 7 An example of the pipelined processing.

表 3 パイプライン処理の効率
Table 3 Performance of pipelined processing.

	字句テーブルサイズ	入力文字数	出力トークン数	処理クロック数	1文字当たりクロック数	1トークン当たりクロック数
PASCAL	2K	26192	6479	43136	1.6	6.5
	4K			32265	1.2	5.0
C-1	2K	17401	3534	38287	2.2	10.8
	4K			22651	1.3	6.4
C-2	2K	32039	9903	70289	2.2	7.1
	4K			41475	1.3	4.2
C-3	2K	27852	13591	42559	1.5	3.1
	4K			37453	1.3	2.8

表 4 字句テーブルの検索特性
Table 4 Reference property to the lexical table.

	字句テーブルサイズ	新エントリ作成回数	既存エントリ参照回数	リハッシュ回数	字句テーブルの最終占有率
PASCAL	2 K	3158	13918	11858	85%
	4 K			1987	42%
C-1	2 K	3336	7197	17769	92%
	4 K			2133	46%
C-2	2 K	3373	20025	32518	93%
	4 K			3704	46%
C-3	2 K	2698	21146	6377	75%
	4 K			1271	38%

パイプラインの動作の一例を図 7 に示す．この例では，字句テーブル検索時にリハッシュが部分的におきるようにテーブルサイズを 512 に制限している．パイプラインが部分的に停止するのは，リハッシュによる場合の他に，名前などの最後を認識するためのデリミタになる文字が処理される時と，字句テーブルに新しくエントリを作成して書き込む場合などがある．後者の例は図 7 には示されていない．いくつかのサンプルプログラムの実行によるパイプラインの動作効率を示したのが表 3 であり，文字レベルでの実行に要した平均クロック数はたかだか 2 程度，トークンレベルでは 3～6 程度であった．ここでは字句テーブルのサイズを実際のシステムで用いている値よりも小さく制限しているので，やや字句テーブルの検査に時間がかかっている．この様子は表 4 に示されている．先にも述べたように，テーブルの占有率が上がると検索効率がやや落ちている．実際には占有率をたかだか 50% 以下で使用するとして，総合的にみたパイプラインの効率を，1 クロック当たりに処理している文字数から見積ると，77%～83% 程度（1 クロックに 1 文字処理するとき 100%）であり，きわめて高い効率が得られている．

6. むすび

文字列の逐次走査をパイプライン化し，字句テーブルを高速に検索するアルゴリズムを開発して，これを用いて字句解析プロセッサを設計し，試作した．この字句解析プロセッサのハードウェア制御機構は，言語の字句レベルの構文をそのまま反映した制御テーブルを実行時に設定することで，対象とする言語に適合した形に動的に構成される．また，その性能に関しては，実現されたハードウェアと等価に動作するレジスタ転送レベルのシミュレータ上で，実際に PASCAL や C のソースプログラムを入力して字句解析の速度を測定し，字句テーブル上での部分文字列を共有することによる文字列の表現効率の向上[8] に加えて，実用上十分な高速処理が行えることが確認された．特に，本方式に適合したハッシュ関数の発見により，1 回当たりの字句テーブルの検索がメモリの参照 1 回強でできるので，文字をメモリから読み出すのに近い速度のきわめて高速な字句解析が可能となった．このような高速性に加え，長さに制限のない可変長の名前の取り扱いや，文字列などの非数値定数や多様な数値定数に対応するために，すべてのトークンを統一的に文字列として取り扱うことにより，精度や内部表現などの実行系や言語仕様に依存する処理から独立させることができた．

本方式に基づいて試作された字句解析プロセッサは既に，直接実行型計算機 UDEC[6),7)] に使用されており，また，将来開発を予定しているハードウェアコンパイラの字句解析ユニットとして使用することも予定している．

参考文献

1) Chu, Y., Itano, K., Fukunaga, Y. and Abrams, M.: Interactive Direct-Execution Programming and Testing, *Proc. of COMPSAC '82*, Chicago (1982).
2) Itano, K. and Chu, Y.: A Pascal Interactive Direct-Execution Computer: PASDEC, Part I: High-Level Design, Technical Report TR-1198, Department of Computer Science, University of Maryland (1982).
3) Itano, K.: A Pascal Interactive Direct-Execu-

tion Computer: PASDEC, Part II: CDL Design and Simulation, Technical Report TR-1202, Department of Computer Science, University of Maryland (1982).
4) Itano, K.: PASDEC: A Pascal Interactive Direct-Execution Computer, *Proc. of High-Level Language Computer Architecture Conference*, pp. 152-169 (1982).
5) Itano, K.: CDL Design of a Pipelined Lexical Scanner, Technical Report TR-1093, Department of Computer Science, University of Maryland (1981).
6) 板野肯三, 佐藤 豊: 汎用直接実行型計算機UDECのアーキテクチャ, 情報処理学会論文誌, Vol. 27, No. 8, pp. 747-753 (1986).
7) Itano, K. and Sato, Y.: Architecture of the Universal Direct-Execution Computer UDEC, *Proc. of Hawaii International Conference on System Sciences* (1987).
8) 板野肯三, 佐々木日出美, 山形朝義: 連想記憶に基づくパイプライン型文字列検索アルゴリズム, 情報処理学会論文誌, Vol. 26, No. 6, pp. 1152-1155 (1985).
9) UNIX Programmer's Manual, Department of Electrical Engineering and Computer Science, University of California, Berkeley (1983).
10) Jensen, K. and Wirth, N.: *PASCAL User Manual and Report*, 2nd ed., Springer-Verlag, New York (1974).
11) Kernighan, B. W. and Ritchie, D. M.: *The C Programming Language*, Prentice-Hall, New Jersey (1978).

(昭和61年1月8日受付)
(昭和61年10月8日採録)

板野 肯三（正会員）

昭和23年生. 昭和46年東京大学理学部物理学科卒業. 昭和48年同大学大学院修士課程修了. 昭和51年同博士課程単位取得後退学. 理学博士. 筑波大学計算機センタ準研究員, 同大学電子情報工学系助手, 講師を経て, 現在, 同助教授. コンピュータアーキテクチャ, オペレーティングシステム, プログラミング言語に興味を持つ. ソフトウェア科学会, IEEE, ACM 各会員.

佐藤 豊（正会員）

昭和35年生. 昭和57年筑波大学第三学群情報学類卒業. 昭和59年同大学院修士課程理工学研究科修了. 現在同大学院博士課程工学研究科に在学中. プログラミング・システムのユーザ・インタフェースおよび構成法の研究に従事. ソフトウェア科学会会員.

山形 朝義（正会員）

昭和34年生. 昭和53年秋田県立米内沢高等学校電気科卒業. 現在, 筑波大学電子情報工学系に文部技官として勤務. 計算機管理, 学生実験などの補助をする傍ら, 高級言語計算機のプロジェクトに従事している.