計算機アーキテクチャ 29－2
（1977. 11. 29）

ハッシング・ハードウェア

井田哲雄，後藤英一，相馬篤
（理化学研究所）

§1 序

ハッシング（Hashing）は探索技法の一つである。ハッシングにおける基本的諸演算を高速化するため，(i) ハッシュ鍵及び付加される値を格納するハッシュ表の構成法，(ii) 鍵空間からハッシュ表空間へのマッピングの選択，(iii) ハッシングを用いた応用アルゴリズムの開発，にわたって，1950年代より広く研究がなされてきた。従来，ハッシングは，純ソフトウェアで実現されてきた技法であった。近年，記号処理，数式処理に代表的に見られるように，データ（データ構造）の探索が主たる処理内容であるような応用には，ハッシングが極めて有効な技法であることが見出され［1,2,3］，ハードウェアによる高速化の必要性が認識されるようになった。

既に確立されたハッシング基本アルゴリズム（例えば文献［4］）をそのままファームウェアあるいはハードウェアで置換するのも，ハードウェア・ハッシングの一つの実現法であるが，より効果的な方法はハードウェア・アルゴリズムの特徴である並列処理（Parallelism）を活用した並列処理ハッシュアルゴリズム（Parallel Hash Algorithms）を考え，それをハードウェアで実現するものである。

筆者らは並列ハッシュアルゴリズムを考案し［5］，それに基づいたハードウェアのインプリメンテーションを現在進めているが，本稿では，これまでに至る研究の推移と，今後の方向について報告する。第2節では，ハッシング・ハードウェアのソフトウェア体系について，第3節ではハードウェア構成及び，性能評価について述べる。第4節でインプリメンテーションについて考察する。

§2 ハッシング・ソフトウェア

2.1 基本用語・概念

o（Object）をハッシュ探索の対象とする時，oを表現する記憶内部データ構造へのポインタ $a(o)$ と o の一つの属性 attribute(o) を用いて，計算（ハッシュ）する。k(= attribute(o)) を対象 o を参照する「鍵」と呼ぶ。一般には k のアドレス空間の方が a のアドレス空間より広いため，相異なる対象 o_1, o_2 の鍵 k_1, k_2 をもってしても，同一のアドレスへとハッシュされる可能性がある。同一のアドレスへハッシュされた時 o_1 と o_2 は「衝突」状態にあるという。衝突が生じた時の処理方法と o を表現するデータ構造により，ハッシュ表の構成方法にいくつかの方法が考案されているが，ハードウェア・ハッシングでは「開アドレス法」（Open addressing）を用いる（その理由は後述）。図1，2は開アドレス法を用いた時のハッシュ表構成を示す。図1では，表の各エントリーは鍵と o を表現するデータ構造へのポインタの対である。データ構造が固定長配列の場合には直接，鍵とデータ構造との対を表中に格納することもできる（図2）。開アドレス法では，衝突が生じる限り順次 $h_1(k)$, $h_2(k)$, … とアドレス列を生成し，探索条件を満足する迄，アドレス生成，鍵の比較のサイクル（ハッシュ探索）を繰り返す。

2.2 鍵の生成・削除法

鍵として何を採用するかは応用によって異なる。例えば，従来のアセンブラ，

コンパイラの記号表では対象は変数、ラベル等であり、鍵はそれらを表現する文字コード列である。鍵が固定長の時は、図1，2のような鍵の作成法で表すが、可変長の場合はポインタでリンクすることになる（図3）。図3のような、(i) 鍵が固定長（図3では一単語3文字）の部分鍵に分割され、部分鍵自体が再びハッシュ探索の鍵となりうる点，(ii) 部分鍵の共有による記憶の節約が計れる点、単純なリンク法[6]にくらべて利点がある。図3では文字列'STATION'と'FICTION'の部分鍵'ION'が共有されている。より一般的な、鍵 $k_1, k_2, \cdots, k_e$ をポインタでリンクした新たな鍵 $k=\{k_1, k_2, \cdots, k_e\}$ をハッシングを用いて、図3のように作成していけば、k 自体が鍵として、使用できるのみならず、リンク法を適当に選ぶことにより、鍵 $k_1, \cdots, k_e$ から新たなデータ構造を作成することもできる。リンクの仕方はインプリメンテーションおよび応用によって定められるべきであるが、一つの単純な方法としてはリスト構造によるものがある。この場合、$k=(k_1 \cdot (k_2 \cdot \cdots (k_e \cdot nil)\cdots))$ である。このようにして作られた2つの鍵 k_p, k_q が同一の構造を表現しているか否かのチェックな、k_p, k_q の表現するデータ構造の複雑さにかかわらず、k_p, k_q のポインタとしての値の比較のみで表す。

LISPに見られるような動的記憶管理機構を備えた言語処理システムでは、上記のような鍵の生成は容易である。鍵の動的生成に加え、鍵と鍵に付加された値の対（連想）を動的に作成することにより、単純な連想処理のみならず、帰納的なアルゴリズムや決定表等の実行を高速化することができる。[7]

以上のようなハッシングの応用では、ハッシュ表から鍵の削除ができなくてはならない。鍵のハッシュ表への挿入に衝突が起る以上、削除された欄は単純に空白化することはできない。このために、衝突の有無を示すタグを各欄に用意し、鍵探索の終了条件を鍵の一致あるいは衝突タグの'0'とみる。衝突タグのたっている空白セルは、鍵が削除されたこと（削除語）を示す。挿入と削除を反復すると、削除語の数が増大し、索表能率が低下するため、衝突数がある一定数を越えた時に、全鍵の再配置（rehash）操作を行い削除語を空白語に変る操作を合せて行う必要がある。衝突タグは鍵の削除を伴わない場合でも、「平均不成功探索回数」（鍵が表中にないことが判明する迄のハッシュ表の平均参照回数）を減らす効果をもたらす。[8]

2.4 基本的ハッシュアルゴリズム

概念的には、ハッシュ探索アルゴリズムはいくつも考えられるが，次の3つがその中でも基本的である。

(1) 'S'： 鍵の表中の有無を調べる探索
(2) 'D'： 既に表にある鍵の削除
(3) 'I'： 表にない鍵の表への新たな登録

これらを実行するハッシング・ハードウェアの命令については第4節でふれる。

§3 ハッシング・ハードウェア

3.1 基本動作

開アドレス・ハッシュ探索は次のような基本操作の反復である。

(a) ハッシュ・アドレス $h_j(k)$ の計算，
(b) アドレス $h_j(k)$ から鍵 k_j の読み出し，
(c) 鍵 k と k_j との比較
(d) 比較によって生成される条件に従い，次にとるべき動作の判断

(a)－(d)は1パンチ・ハッシュ表を前提とした ハッシュ探索の基本ステップである。

(アルゴリズム0)。ハードウェアではステップ(a),(c),(d)は高速に実行でき、しかも(b)とh_{j+1}の計算は並列して行なえるため、ハッシュ探索の速度を決定する因子は(b)のハッシュ表からの鍵の読み出しにある。ハッシングの能率は(a)-(d)の反復回数Pで決まるが、Pはハッシュ表の利用率α_m(=表中の有効鍵総数/格納可能鍵総数)に依存する。純ソフトウェア・ハッシングでも、アルゴリズム0でも、理論計算量(computational complexity)としてのPの値は同一である。実用上も、$\alpha_m \to 1$に近づくにつれて、Pは著しく増大し、Pの回数がハッシングの能率を規定するようになる。

ここで、ステップ(b),(c)に、鍵の読み出し、比較に並列処理を導入する。即ち、ハッシュ表を複数個(J個)の記憶バンクで実現し、(b)で一度にJ個の鍵をハッシュ表から読み出し、(c)でのJ個の鍵の比較も一度に実行する。

3.2 並列ハッシュ・アルゴリズム

ハッシュ・アドレス生成及び、鍵挿入の際の手続きの違いによって次に述べる3通りの並列アルゴリズムが考えられる。

アルゴリズム-1

一度の記憶参照に対して、同一のハッシュ関数列$h_i(k)$, $i=0,1,\cdots$がJ個のバンクに対して共通に用いられる。したがって、ハッシュ表をK[1:M, 1:J]とした時、一回のハッシュ探索で、K[$h_i(k)$, 1], K[$h_i(k)$, 2], ……, K[$h_i(k)$, J]が読み出される。

アルゴリズム-2

互に独立なJ個のハッシュ関数列$h_i^{(j)}$, $i=0,1,\cdots$が各記憶バンク($1 \le j \le J$)に対して用いられる。したがって、一度のハッシュ探索に対してK[$h_i^{(j_1)}(k)$, 1], K[$h_i^{(j_2)}(k)$, 2], ……, K[$h_i^{(j_J)}(k)$, J]が同時に読み出され、$j_1, j_2, \cdots j_J$の順序で、鍵挿入時に空セルが探される。ここで$(j_1, j_2, \cdots, j_J)$は鍵kおよびjに依存して決定される$(1, 2, \cdots, J)$の順列である。

アルゴリズム-3

アルゴリズム-2と同様にJ個の独立なハッシュ関数列が鍵の読み出しに用いられる。一度のハッシュ探索でK[$h_i^{(1)}(k)$, 1], K[$h_i^{(2)}(k)$, 2], ……, K[$h_i^{(J)}(k)$, J]が読み出され、鍵の挿入に当っては、1, 2, …, Jの順で空セルが探される。

3.3 ハードウェアの構成

ハッシング・ハードウェアはJ個の記憶バンクモジュール(各々M語の記憶容量を持つ)、ハッシュ・アドレス生成器(HAG)およびハッシュ演算制御ユニット(HCU)から構成される。(図4) HAGで計算されたアドレスはアドレスバスを介して、各記憶バンクに送られる。ハッシュ表が1列に対して、J個の鍵が同時に読み出される。読み出された鍵k_j($1 \le j \le J$)は比較器cp(Comparator: $k=k_j$ ならば $m_j=1$ しからざれば $m_j=0$)により、入力鍵kと比較される。ED(Empty word Detector)は空白語の検出器(空白語なら$e_j=1$ しからざれば$e_j=0$)である。T/Cは鍵の衝突の有無を記録する衝突タグ/計数レジスタである。ZD(Zero Detector)は衝突検出器(鍵が衝突を受けていれば$z_j=1$ しからざれば$z_j=0$)、およびUDC(Up Down Counter)はアップダウン計数器である。UDCは上記c(counter)を用いる時に必要である。HCUは前述ステップ(d)における判断機構を組み込んだものである。

図4では他の処理装置との接続は明示していないが、アドレスバス、データバスを他の中央処理装置に接続し、ハッシュ表を主記憶として利用することも、あるいはI/Oバス/メモリバスに接続し、単独のプロセッサーとしても稼動可能である。

3.4 ハードウェアの性能予測

ハードウェアの性能はハッシュ探索回数Pで決まる。Pはハッシュ探索アルゴリズム（2.4節）固有の量が存在するが、最も基本的な計算量はPS（成功探索回数）およびPU（不成功探索回数）である。PS, PUは表引用率α_mを一定に保って、その近傍で、鍵の削除，挿入を交互に繰り返した時に最大値PS, PUになることが判明しており、PS, PUはα_mをパラメータとして、数値的にも求めることができる。（ハッシュ関数列および鍵の削除の無作為性を仮定する）[9]

評価の結論のみを言うと、アルゴリズム-3が最も効率が良い。図5、図6にアルゴリズム-3におけるPS, PUのα_mに対する依存性を示す。

§4 インプリメンテーション

4.1 使用環境

ハッシング・ハードウェアは記号・数式処理システムFLATS[10]の一部として、実現を意図しており、現在は具体的な設計段階にある。FLATSではLISPに代表されるリスト処理を高速に実行することを一つの設計目標としている。FLATS LISPでは64ビットセルを一語としている。1セルはcar部，cdr部各々32ビットの半語に分割される。32ビットのうち8ビットは、トラップ法によるタグとしても利用する[11]。論理・算術演算は24ビット幅を基本とする。アドレス空間の指定には24ビットを用いる。

ハッシングに関する限り、バンク数が多ければ、それだけ性能は向上するが、FLATS CPUを含めた他の処理モジュールが広い記憶転送幅を有効に生かし得ない可能性もあり、インプリメンテーションではバンク数を2とした。したがって主記憶のバンド幅は128ビットである。理論上およびシミュレーションではアルゴリズム-3が最も性能が良いが、ハードウェアの規模から言うと、アルゴリズム-1はHAGが1つですむこと、アドレスバスの幅がアルゴリズム-3のそれの半分で済むなど、インプリメンテーション上の利点がある。我々のインプリメンテーションではアルゴリズム-3を採用した。

鍵の削除はガーベッジ・コレクタ（GC）が行う。衝突数の計数はGCが起動した時、ソフトウェアで行う。したがって、前述1/Cの1：衝突タグを記憶バンク内に装備する。衝突タグは各バンクの一語を4ビットに1ビット必要である。この他に空白語を示すタグを一語につき1ビット用意した。前述2Nは不要になる。

実際には、パリティビットを含めて、一語に72ビットを要する。なお、上記タグの存在はハッシング・ハードウェアおよびGC以外のモジュールからは見えない。

4.2 ハッシュ関数の選択

ハッシュアドレス列としては、$(h_0(k), h_1(k), \cdots, h_M(k))$が整数$(1, 2, \cdots, M)$の順列になっているもので鍵kに応じて順列が擬似乱数的に選ばれるものが一般に良い結果をもたらす。ハードウェア作成の見地からは、次のような基準が要求される。

(i) アドレス計算に要する時間

最も望ましいのは記憶装置のアクセス時間内に、ハッシュアドレスが計算できるものである。用いる記憶素子の種類にもよるが、アクセス時間を300n.s程度に見積ると、市販のLSI（例えばAM2901）を使用すると、この速度は加算実行マイクロ命令に匹敵することになり、実現は困難である。到達目標として、一サイクル時間内にアドレス計算を終了させることを検討してみると、一サイクル時間を150n.sとすれば、単純な加減，論理演算でアドレス計算ができれば、実現可能である。

なお、再ハッシュ関数列の計算を行なわずに、衝突鍵のリンクによって、衝突

状態を解消する方法である連鎖法も検討した。実際に、連鎖法を使用して連想処理装置を作成した例も報告されているが[12]，ハードウェアでハッシングを行う場合ハッシュ表のアクセスが動作の性能を決定する因子になるため、並列処理で衝突参照回数を減少させることが困難な連鎖法はインプリメンテーションの考慮から外しました。

(iii) 一様性

$(h_0(k), h_1(k), \cdots, h_{M-1}(k))$ が $(1, 2, \cdots, M)$ の順列になっていること。M回の探索で必ず、すべてのセルを参照する保証が得られることが、ハードウェアで探索の終了条件をチェックするうえでも、性能のうえからも望ましい。

以上の考察の結果、ハッシュ・アドレス列生成にはダブル・ハッシュ法[4]を採用した。ハッシュ関数には、Mを素数をとって mod (K/M)（割り算の剰余）が、ソフトウェア・ハッシングでは広く用いられるが[4]、(iii)で述べた速度が得られない。そこで、ハッシュ表の大きさを2のべき（2^m）とし、割り算に代えて (2^m-1) との論理積を使用する。

図7に、望ましいアドレス生成機構を示す。一語64ビットのうち、鍵部は応用ソフトウェアによって選択できるよう、鍵に対してバイトごとのマスクを用いる。マスクを受けた鍵がK-BUSを介して、HAGに入力される。ハッシュアドレス列 $h_0, h_1, \cdots h_{M-1}$ は次のようにして決定される。

第一回目のハッシュ探索：

$h_1 = h_0$ ； $h_w = h_1$ & MASK[ℓ] ； $h_a = h_w$ + BASE[ℓ]

第二回目以降のハッシュ探索：

$h_{i+1} = h_i + \Delta h$ ； $h_w = h_{i+1}$ & MASK[ℓ] ； $h_a = h_w$ + BASE[ℓ]

ここで $h_0 = H0(k')$，$\Delta h = DH(k')$ k' はマスクを受けた鍵である。
Δh が奇数となるよう DH の論理回路を作成すれば、このようにして得られるハッシュ・アドレス列 $\{h_1, h_2, \cdots h_{M-1}\}$ が(iii)の一様性を満足することは容易にわかる。
H0, DH の回路としては Folded-Sum, Folded-EXOR を検討した。シミュレーションの結果では、双方の PS, PU に有意差は見出されなかったため、実現の容易で、かつより高速な Folded-EXOR を採用した。ここで Fold（折り合せ）といっても、文字通りの折り重ねである必要はない。むしろ、ビットの桁混合後のパリティー生成と考えればよい。k' は64ビットで h_0 は1Mバイト実装では16ビットであるため，H0 DH は入力ビット順序の異なる4ビットパリティ生成回路16個の集まりである。
ℓ の値を変えることにより、複数個の各々大きさの異なるハッシュ表を切り替えて使うことができる。1つのハッシュ表に対し、ハッシュ表の先頭アドレス、M-MASK (2^m-1)，最大許容衝突数、K-MASK（読み出し鍵に対するマスク），L-MASK（書き込みバイトを決定するマスク）の5つ組が必要とされる。これらデータは、ハッシュ表初期設定時に、各々対応するレジスタに書き込まれる。現在使用を予定しているハッシュ表は、連想記憶用，アトムヘッダー用，リスト鍵生成用，集合生成用[3]，OSの記号管理表用の5種類である。

実際には市販のLSIを使用する関係上、図8のような回路で実現を計画している。図8では2個のHAGが示されている。$h_0^{(1)}$, $h_0^{(2)}$ は各々独立である必要があるが、Δh は2つのHAGで共通に用いても良い。h_i で $i=1, 2, \cdots$，h_w の値は各々Q，Bレジスターに格納される。

ハッシュ演算制御はマイクロプログラムで行なう。表2にマイクロプログラムで

解釈実行される代表的（マクロ）命令セットを示す。表2で 1,2の命令が 2.4節のアルゴリズム'ふ'に、3が'エ'に、5が'わ'に対応する。4は原理的には'S','エ'の組み合せで実現できる複合アルゴリズムであるが、'S','エ'で別個に生成されるハッシュ・アドレス列の重複を避けるため、新たな命令として付け加えた。命令4などアセンブラやコンパイラーの記号表の作成に度々用いられる命令である。命令1〜5では、入力パラメーターの鍵を、一回の間接アドレスで得るようになっているが、Immediate 命令のような変形命令を付けることも考えられる。同様に命令7,8では鍵でなく、実アドレスで衝突タグあるいは空白タグを得るような命令も必要になろう。

具体的なデータ情報、制御情報の流れを、命令 NINSRT を例にとって以下に説明する。

ステップ(1) 入力パラメータ（鍵の格納されているセルのアドレス）をVースタック先頭より受ける。

(2) 記憶の並列読み出しを行い鍵を KEY レジスタに読み出す。

(3) $h_0^{(1)}, h_0^{(2)}$ を計算し $h_a^{(1)}, h_a^{(2)}$ をアドレスバスに出力する。

(4) 比較する鍵の並列読み出し起動
アクセス時間経過後、比較結果 M_0, M_1; 空白タグ E_0, E_1 および $Z=(Z_0 \wedge Z_1)$ が得られる。

(5) (M_0, M_1, E_0, E_1, Z) から 4ビットのエンコードされた条件コードをマッピング ROM を通じ割り出し、マイクロプログラムで、対応する処理ステップへ多分岐ジャンプをする。

(6) (6-1) (0,0,0,0,0) の場合

(6-1-1) $Z_0=1$, $Z_1=1$ とし、書き込みサイクル開始

(6-1-2) $h_i^{(1)} = h_{i-1}^{(1)} + \Delta h$, $h_i^{(2)} = h_{i-1}^{(2)} + \Delta h$; $s^{(1)} = h_i^{(1)} \wedge \mathrm{MASK}[\ell]$, $s^{(2)} = h_i^{(2)} \wedge \mathrm{MASK}[\ell]$ を計算する。

(6-1-3) $h_a^{(1)} = s^{(1)} + \mathrm{BASE}[\ell]$, $h_a^{(2)} = s^{(2)} + \mathrm{BASE}[\ell]$ を計算し、アドレスバスに出力する。

(6-1-4) ステップ (4) へ行く。

(6-2) (0,0,1,0,0) の場合

(6-2-1) KEY レジスタの内容が KBUS を介し、L-MASK でマスクを受けて MDBUF0 にはいる。 $E_0=0$ とし、書き込みサイクル開始。
$h_a^{(1)}$ を DMAMUX へ送て、Vースタックから送り、命令終了

⋮

以下省略（簡単のため、衝突救済のステップは省略した）

実際のプログラミングでは、第1,第2ハッシュ探索を高速化したり、チャネルからのメモリーステールを許すために、一定回数の記憶アクセスごとにサイクルを開放するなどの考慮が必要であるが、細かい道具立ては説明を省略した。

一マイクロ命令実行サイクル時間を 300 ns, 記憶装置のサイクル時間を 750 ns（スプリットサイクル可）で NINSRT の平均命令実行時間は $\alpha_m = 0.6$ で約 3 μsec と見積っている。

§5 今後の研究方向

本ハードウェアは 1 Mバイトの記憶を装備し、ハッシュ表は FLATS の中央処理装置から主記憶としてもアクセスされる予定である。最初は、小型計算機に接続し、試験的稼動を来春に予定している。FLATS との接続は、128ビットのメモリーバス（MDBUF0, ...を介して）に加えて、Vースタックと呼ぶ FLATS 専用オペランド・スタック

と32ビットバスで接続される予定である。ハッシュ・ハードウェアの研究は、理論的な性能予測，ソフトウェアの検討，BUシミュレーションによる動作予測が完了しており、本課題に関する限り、研究（research）のテーマから開発（development）のテーマへと移行しつつある。

引用文献

[1] Schwartz, J.T. (1973) On Programming, an Interim Report of the SETL Project, Installment II: The SETL language and examples of its use, Courant Institute of Mathematical Sciences, New York University, New York

[2] Feldman, J.A. and Rovner, P.D. (1969) An Algol-Based Associative Language, CACM, Vol. 12, No.8, 439-449

[3] Goto, E. and Kanada, Y. (1976) Hashing Lemmas on Time Complexity with Application to Formula Manipulation, Proc. ACM-SYMSAC'76, New York, 1976

[4] Knuth, D.E. (1973) The Art of Computer Programming, Vol.3, Addison-Wesley

[5] Goto, E., Ida, T. and Gunji, T. Parallel Hashing Algorithms Information Processing Letters, Vol. 6, 8-13

[6] Gries, D. Compiler Construction for Digital Computers, John Wiley & Sons

[7] Goto, E. and M. Tevarshima (1971) MTAC- Mathematical Tabulative Automatic Computing 「数値計算のアルゴリズムとコンピュータ」研究集会予稿，京都大学数理解析研究所

[8] 古川竜一 (1973) コンフリクトフラッグをもったハッシュ記憶法 情報処理 13, pp. 533-549

[9] Ida, T. and Goto, E. Analysis of Parallel Hashing Algorithms with Key Deletion (Journal of Information Processing of Japan に投稿中)

[10] 後藤英一，井田哲雄，相馬薗 (1977) 記号数式処理向き計算機 FLATS の構想 情報処理学会，記号処理研究会資料 1-1

[11] 後藤英一，井田哲雄 (1977) データタイプチェックに関する一考察 第18回プログラミングシンポジウム予稿集

[12] Gall, R.G. and Brotherton, D.E. (1966) Associative List Selector RADC-TR-66-281

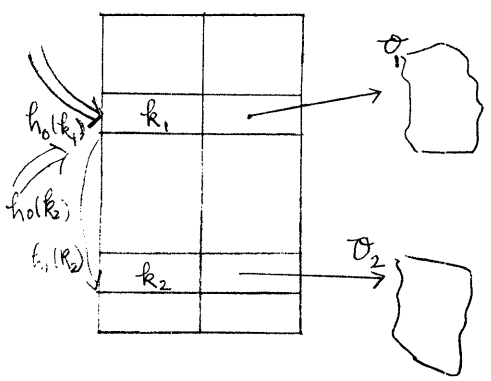

図1 開掘アドレスを利用したハッシュ表

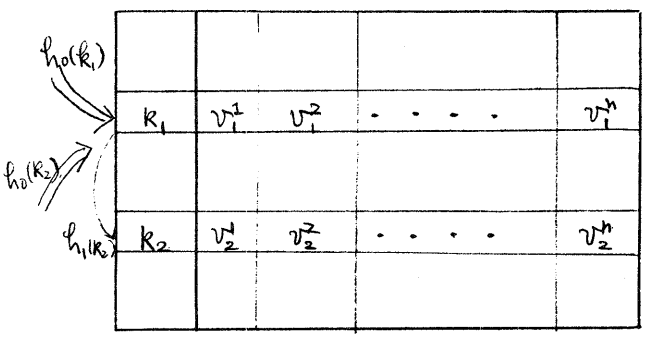

図2 直接探索方式ハッシュ表

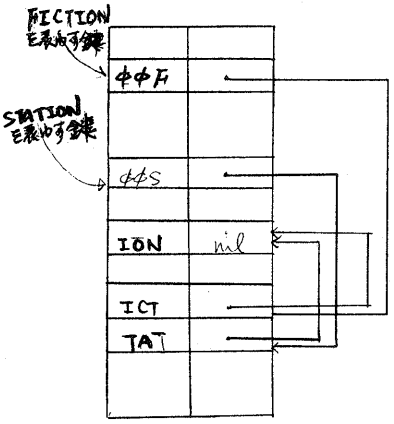

図3 可変長鍵の作成法

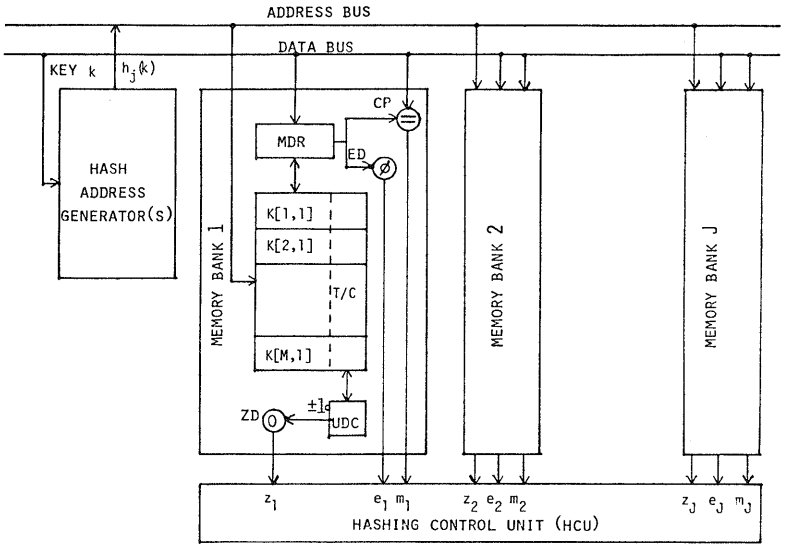

図4 ハッシング・ハードウェアの概念構成図

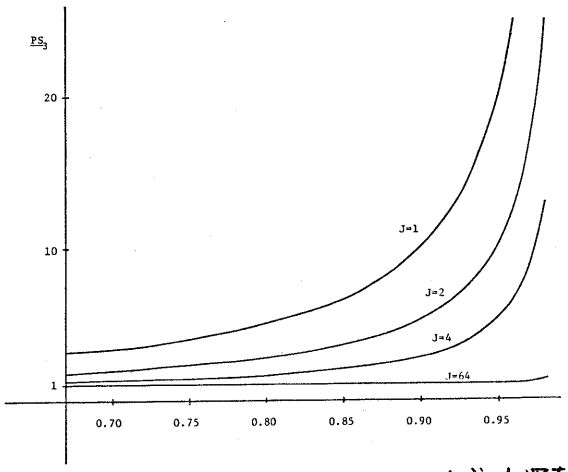

図5 アルゴリズム-3における平均成功探索回数のバンク数Jに対する依存性

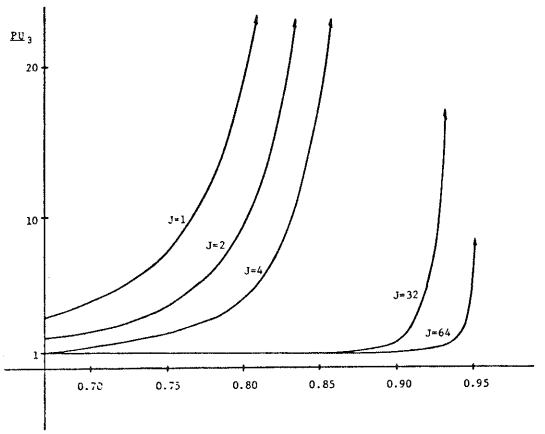

図6 アルゴリズム-3における平均不成功探索回数のバンク数Jに対する依存性

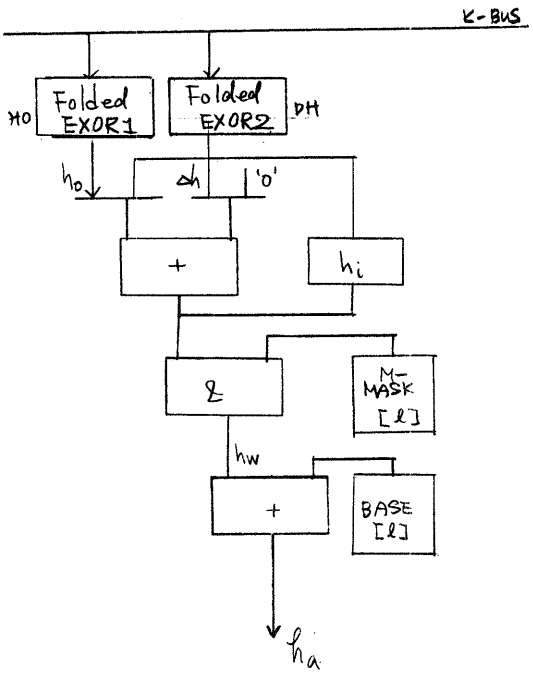

図7 望ましいアドレス生成機構

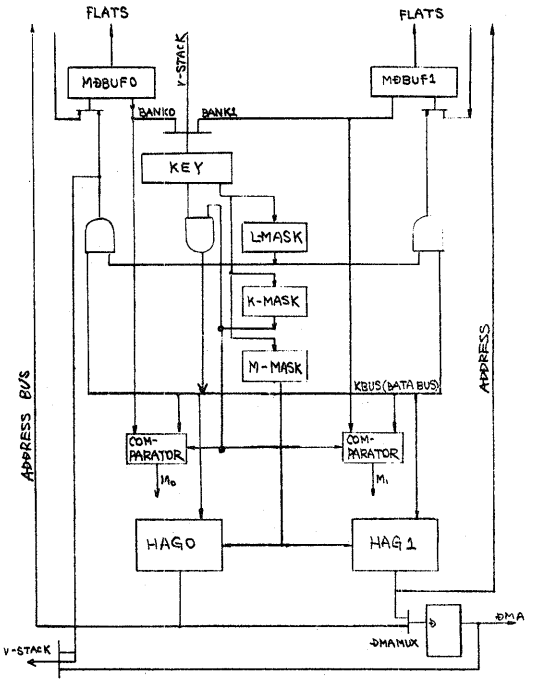

図9 ハッシング・ハードウェア 全体ブロック図

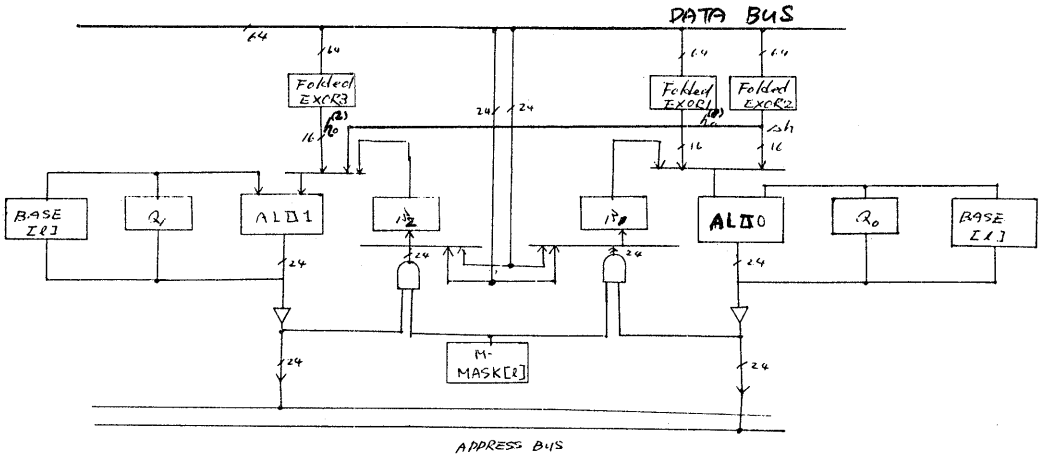

図8 実現を計画中のハッシュ・アドレス生成器 (HAG's)

	α_m=0.6	0.7	0.8	0.9	0.95
PS_1	1.67	2.09	2.94	15.46	1.05×10
PS_3	1.56	1.96	2.78	5.26	1.03×10
PU_1	1.98	3.66	1.42×10	1.22×10^3	1.48×10^5
PU_3	1.45	2.48	9.45	9.27×10^2	1.41×10^5

表1　バンク数2における PS_1（アルゴリズム1）PS_3（アルゴリズム3）
PU_1, PU_3

表2. ハッシング・ハードウェア命令表

Mnemonic	意味	パラメータ	結果（成功）	結果（不成功）
1. KSRCH	Key Search	(1) 鍵のアドレス, (2) ハッシュ表番号	鍵の格納されているセル)のアドレス	NULL
2. ASRCH	Associative Search	(1) 鍵のアドレス, (2) ハッシュ表番号 (3) 0/1（半語の位置を示す）	設定されてセル64ビットのうち,指定された半語を返す	NULL
3. NINSRT	New key Insert	(1) 鍵のアドレス (2) ハッシュ表番号	鍵の挿入されたセル)のアドレス	
4. INSRT	Insert	(1) 鍵のアドレス (2) ハッシュ表番号	鍵の挿入されたセル)のアドレス	鍵の格納されているセル)のアドレス
5. DELETE	Delete an Exsisting Key	(1) 鍵のアドレス (2) ハッシュ表番号	削除された鍵のあったセルのアドレス	
6. INTLZ	Initialize	(1) 次のパラメータリストのアドレス ○ K-Mask（バイトごと） ○ L-Mask（〃） ○ 最大許容衝突数 ○ 2^M-1（ハッシュ表の大きさを 2^M と与る ○ ハッシュ表の先頭アドレス	0	1（パラメータの指定の誤り）
7. LDCTG	Load Collision Tag	(1) 鍵のアドレス (2) ハッシュ表番号	0/1	
8. LDETG	Load Empty Tag	(1) 鍵のアドレス (2) ハッシュ表番号	0/1	