Vol. 27 No. 1 情報処理学会論文誌 Jan. 1986

数式処理計算機 FLATS のアーキテクチャ†

平木 敬†† 後藤英一†††

数式処理に代表される大規模記号処理プログラムを実行するためには，汎用計算機は記憶容量，演算速度，機能の面で不十分である．したがって，これらのプログラムを本格的に実用化するために専用アーキテクチャの開発が不可欠である．数式処理計算機 FLATS は数式処理プログラムおよびその他の Lisp 応用プログラムを高速かつ安全に実行することを目的とした計算機である．本論文では FLATS で用いられた高速化技法およびアーキテクチャについて述べる．命令形式として Lisp に適したやや低レベルの命令セットを選択し，ハッシング，ガーベジ・コレクション，リスプ基本関数など重要な命令を加えた．アーキテクチャは Lisp 実行に最適化したパイプライン構成の CPU，2本のスタック，3並列読み出しスタックメモリ，命令・スタック・セル領域独立の組のキャッシュメモリ等により，関数呼び出し，条件分岐などが多発するプログラムでも速度低下を防止している．また並列ハッシングをハードウェアで実現することにより，間接アドレシングとほぼ同等の速度でハッシング操作を実現した．その結果，基本素子速度またはクロックが8倍程度速い汎用計算機と同等以上の性能を上げ，Lisp 専用アーキテクチャの有効性を示した．

1. はじめに

本論文では Lisp 言語とその応用分野の一つである数式処理を高速に実行するために設計された計算機，FLATS のアーキテクチャおよび高速化技法について述べる．

Lisp 言語は，非数値的応用分野において，数値的応用分野における Fortran に相当する重要な地位を確立した．Lisp 言語で書かれた大規模な応用プログラムはその計算量，使用する記憶容量ともに数値計算における超大型プログラムに匹敵するものになってきている．そのなかで，数式処理システム[1),2)]は与えられた入力数式に対して多項式演算，微分，積分，因数分解，索表等の手続きを適用し，式の変形，方程式の求解，数値計算式の自動生成などの仕事を行う応用システムである．

アーキテクチャから見た数式処理プログラムの特徴としては，(1)記号演算や多倍長演算では大容量動的記憶が不可欠であり，そのための“ごみ集め”管理が必要であること，(2)変数操作，微積分操作に代表される記号処理と，係数操作に代表される数値演算とが混在し，特に係数操作における数値演算では整数演算を中心に，場合によっては桁数の多い多倍長数の演算を含むこと，(3)配列演算等の数値計算プログラムに比べ，アルゴリズムに直列的要素が多く，並列処理による高速化が困難であること，(4)疎な入力に対する関数値を計算する場合，変数表，数式表を引く場合には，索表操作が高速に実現されることが必要であることなどが列挙される．また，Lisp の特徴として，基本データタイプとしてリストを使用するため，データの局所性が悪いことが知られている[3)]．

従来，これらの特徴をもつ数式処理は専ら汎用計算機で実行されてきた．汎用計算機の最も大きな特長は，汎用であるがゆえにその時点で使用可能な最も高度な論理素子，回路テクノロジーを使用できることである．もちろん計算機の性質上，計算時間，記憶容量を度外視すれば，どのアーキテクチャの計算機でも上記機能を満たすことが可能である．しかしながら多くの汎用計算機アーキテクチャは Lisp 言語実行のために適さない要素を含むため，汎用計算機上に実装した Lisp システムでは，機能または安全性の犠牲なしには高速化，大容量化を行うことが困難である．したがって，アーキテクチャ的アプローチ，すなわち専用アーキテクチャを用いることが必要である．これが FLATS アーキテクチャ開発の動機となった．

Lisp 専用アーキテクチャの研究は，Deutsch[4)]により始められた．実際に製作された Lisp 計算機としては，CONS 計算機[5)]が初期のものである．CONS の後継機種として6) 等へと発展した．これらの個人用 Lisp 専用計算機は，小型かつ安価な計算機システムで，豊富な機能，広い仮想空間およびある程度の速度を得ることを特徴としている．実験機として製作された Lisp 専用計算機としては7)～9)等が知られている．この他，試作された Lisp 専用計算機として，10),11)

† An Architecture of FLATS—A Computer for Symbolic and Algebraic Computations by KEI HIRAKI (Computer Systems Division, Electrotechnical Laboratory) and EIICHI GOTO (Faculty of Science, University of Tokyo and Institute of Physical and Chemical Research).

†† 電子技術総合研究所電子計算機部計算機方式研究室
††† 東京大学理学部情報科学科，理化学研究所情報科学研究室

等が報告されている．しかしながら，これらのシステムは最高速汎用計算機と比較すると，速度および記憶容量の両面で不十分なものであり，FLATS とは目的が異なっている．

2. 基本構成

FLATS 計算機はバック・エンド・コンピュータとして他の計算機に接続して動作するように設計されている．このため I/O 装置，ターミナル等への接続はすべてフロント・エンド・プロセッサで処理され，保守および運転に関する機能はサービス・プロセッサで処理される．

内部構成は，**図1**に示すように CPU，キャッシュメモリ，主記憶管理装置，主記憶装置で構成される．なお，キャッシュメモリは ICH（命令キャッシュメモリ），VCH（スタック・レジスタ・キャッシュメモリ），DCH（データ・キャッシュメモリ）の3組を設置し，さらに VCH は並列な3重キャッシュメモリで構成される．主記憶管理装置はフロント・エンド・プロセッサおよびサービス・プロセッサへの接続インタフェースも含む．主要諸元を**表1**に示す．

記憶空間は 2^{25} の大きさをもち，等しい 2^{24} の大きさをもったI空間とD空間に分割される．I空間は主に命令の格納に用いられ，D空間はスタックおよびデータメモリのために用いられる．D空間はさらにグローバル・レジスタ領域，ローカル・スタック領域，

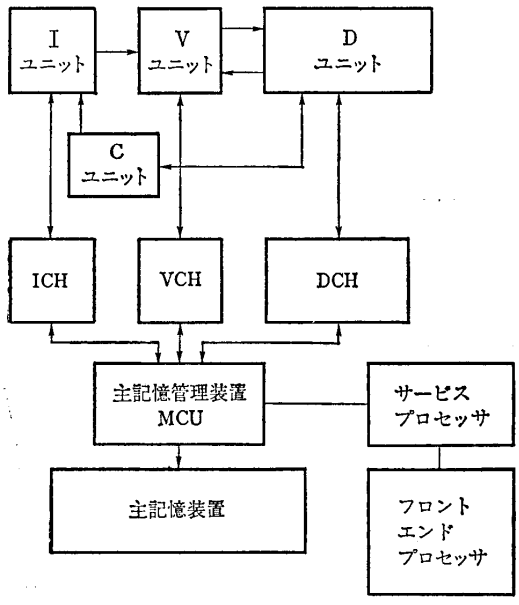

図 1 全体構成
Fig. 1 Block diagram of FLATS.

表 1 主要諸元
Table 1 Machine feature.

使用言語	LISP
マシンサイクル	120 ナノセカンド
制御記憶	1k語+256 語+256 語
語長	32 ビット，64 ビット
データタイプ	30 種
リスト構造	線形化 CDR コーディング
論理記憶容量	32M ワード (128M Byte)
実記憶容量	2M ワード (8M Byte)
キャッシュメモリ	56k Byte
フロントエンド	
プロセッサ	VAX 11/750
論理素子	ECL 10k, 100k (約3万個)
筐体	1150 (W)×1612 (H)×700 (D) 7台

コントロール・スタック領域およびデータ領域の4領域に分けられる．この前二者を V 空間と呼び，VCH を経由して命令の実効オペランドとして読み出され，後二者をデータ領域と呼び，DCH を経由してアクセスされる．

D空間はV空間以外にさらに L，R，H，および C（コントロール・スタック）領域，その他の領域に細分される．L領域は CONS および LIST 2命令でリストを生成する領域である．FLATS では自由領域を連続して取るガーベジ・コレクション法を使用するため，CONS 等の操作は常に自由領域の端に対して行われる．この位置を示すポインタが LPR である．R領域は RCONS 命令のための領域である．RCONS とはリストを CONS の逆方向から生成する基本操作で，ループをプログラムする際に有力な命令である[12]．

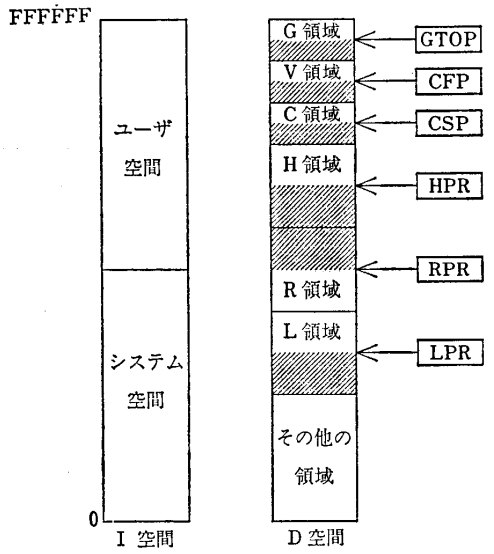

図 2 記憶空間
Fig. 2 Memory space allocation.

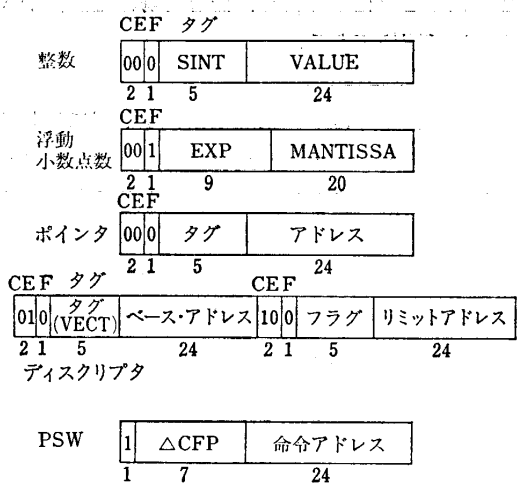

図 3 データ構造
Fig. 3 Data format.

RCONS 命令で使用する自由領域も CONS の場合と同様一連続として取られる．H領域は HCONS命令のための領域である．HCONS はシステム内での唯一化したリストを作成するための命令である．

C領域はコントロールスタックに割り当てられている領域である．D空間のその他の領域はすべてソフトウェアにより分割される．FLATS メモリ空間分割を図 2 に示す．

FLATS で扱う基本的なデータの長さは 32 ビットであり，場合により 2 倍長すなわち 64 ビットを占める．システムで用いる一部のデータを除き，すべてのデータ型は，そのデータの属性の標識である 3 種類のタグをもつ．FLATS データ型の原則的な構造と解釈を図 3 に示す．

3. 命令セット

FLATS 命令セット設計は，コンパイラを通して命令を使用すること，単一固定長の命令を使用することを前提とした．オペランド形式としては，ローカル・スタック上のオペランドを最も頻繁に使用し，かつ Lisp プログラムでは操作に対して入力オペランドを破壊しないことが好ましい場合が多いため，図 4 に示すように，3 オペランド形式を採用した．各オペランドはローカル・フレーム・レジスタ，グローバル・レジスタまたは短分岐オペランドを取る．

ローカル・フレーム・レジスタは，スタックとして，(1)ローカル変数の格納，および(2)関数，サブルーチンを使用する際，引数および結果の引渡しに使

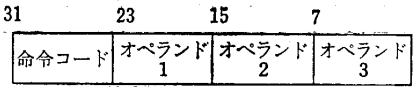

図 4 命令フォーマット
Fig. 4 Instruction format.

用する．サブルーチンまたは関数を呼び出すとき，CALL 命令はフレーム・ポインタを指定された値だけ増加して，新たなフレームを直前のフレームと一部重複する位置に取り，この重複部分を引数および結果の引き渡しに使用する．グローバル・レジスタはV空間上に 128 個単位で切り換えて使用する．グローバル・レジスタは大局変数および T, NIL 等の定数を保持し，ユーザまたはプロセスごとに切り換えて使用する．FLATS では，条件分岐命令ではすべて 8 ビットの相対分岐アドレスを使用した短分岐オペランドを使用する．短分岐オペランドは命令語 (32 ビット)単位のオフセット整数で表示され，+127～−128 の範囲で分岐が可能である．条件分岐命令のオペランドとして短分岐オペランドを採用した理由は，データ検査，分岐と，命令の実行を並行して実行することにより命令数の減少を図るとともに，通常のプロセッサに必須である算術演算のコンディションコードをプロセッサ・ステータス・ワードとしてもたないようにするためである．この結果，汎用計算機に見られるコンディションコードをすべて廃止し PSW を単純化する．このことは保護機能の実現および割り込み動作の高速化に大きく貢献する．

FLATS には現在 230 種の命令が実装されている．ここでは特徴的な命令を紹介する．

3.1 リスプ命令

FLATS では Lisp の 5 基本関数および RPLACA, RPLACD, CADR, CDDR, LIST 2, HCONS および RCONS を直接命令で実現している．

CAR, CDR では第一オペランドには読み出すリストのアドレスが入る．第二オペランドは短分岐オペランドで，第一オペランドがリストへのポインタ型以外である場合に分岐する分岐先相対アドレスを示す．この分岐機能は CDR を順にとる繰り返し操作でリストの終端を検出する場合等に使用する．第三オペランドは CAR, CDR 操作の結果の書き込みオペランドである．

CONS は新たなリスト要素を作成する命令である．FLATS では CDR-コーディング[13] を行うため，第二オペランドのデータがリストへのポインタ型であ

り，かつL領域の LPR 側の端にある場合，すなわちいちばん最近に作成したセルに対して CONS する場合には，LINEAR セルを作成し，他の場合には DOTEND セルを作成する．第三オペランドには生成したセルへのポインタを書き込む．CONS 命令実行中，自由領域がなくなり，セルを生成できなくなる場合は，ページフォルトにより検出される．この他前述の RCONS, HCONS 命令も FLATS 高機能化のために重要な命令である．

3.2 ハッシング命令

FLATS ではハッシュ表として AMT, CAT および HTT 3種を使用する．ハッシング命令も対応する3種類を使用する．

AMT (Associative Membership Table) はキーだけで構成されるハッシュ表であり，8バンク並列オープンハッシュ法を使用する．AMT に対する操作は，索表操作とスイープ操作に分けられる．索表操作は GETAMT (索表)，PUTAMT (索表および挿入)，REMAMT 命令 (索表および削除) で実行する．いずれの場合も第一オペランドには AMT ディスクリプタへのポインタが入り，第二オペランドには索表するキーが入る．

スイープ操作は，ハッシュ表に挿入されている全キーに対して順次キーを読み出す操作である．この操作は，ハッシュ表中のすべての要素に対し演算を行う場面で有効である[14]．

第二のハッシュ表が CAT (Content Addressed Table) である．CAT はキーとデータを一組に表に格納するハッシュ表であり，4バンク並列オープンハッシュ方式を用いる．この場合キーを偶数番地に，データを奇数番地に格納する．CAT に対する命令は AMT とほぼ同様であり，CAT に対しても AMT と同様スイープ操作が可能である．

HTT は Lisp システムが使用するハッシュ表であり，システム内のオブジェクトを唯一化するために使用する．HTT は V-Key を使用した8バンク並列オープンハッシュ方式を使用する．V-Key 方式のハッシュ表では，探索はキーを短縮して得られる V-Key に対してキーの実体をさすポインタを返す．HTT が AMT, CAT といちばん異なる点は，AMT, CAT では一つのキーは表中に一つしか存在しないが，HTT では同一の V-Key が複数個表中に存在しうることである．したがって探索操作は，表中の V-Key 探索を命令で実行し，一致する V-Key の実体の比較を繰り返すことにより行う．命令自身の機能は AMT 命令とほぼ同様である．

3.3 ビット操作命令

ビット操作命令は，D空間上のビット列に対して検査，ビット検索，ビット計数およびガーベジ・コレクション時のポインタ変更操作を行うものである．その中で PADJ 命令はコンパクティファイ・ガーベジ・コレクション[15]に必要なポインタ変更の操作を，マークされたセルのビット表と，ビット表の 32 ビットごとのオフセットの表を使用して1個のセルについて1命令操作で完了する．この命令等によりガーベジ・コレクションの速度は著しく向上する．

4. FLATS アーキテクチャ

FLATS の CPU は，ICH, VCH, DCH に対応する I (命令フェッチユニット)，V (実効オペランドフェッチユニット)，D (実行ユニット) および C (コントロール・スタックユニット) で構成される．これら4個の主要ユニットで4段のパイプライン動作を行う (**図5**)．

Iユニットは FLATS CPU パイプラインの第一段として，命令をメモリのI空間から読み出し，後段に供給する．Iユニットはさらに GOTO, CALL, RET, 条件分岐命令および非パイプライン命令の解釈と実行の一部または全部を行う．GOTO, CALL, RET 命令はIおよびCユニットで完全に解釈実行される．この結果これらの命令はパイプラインの流れから完全に消滅し，命令バッファに先行する命令が蓄えられている場合には，命令実行時間が0となる．

例えば，CALL 命令がI空間メモリから読み出されると，命令バッファが空でない場合でもただちに，IおよびCユニットで解釈実行が開始される．CALL であることが認識されるとただちにCユニットにおいて，仮のスタック操作が行われ，Cユニット内部で PSW が保持される．続いて，CALL の2語目にあた

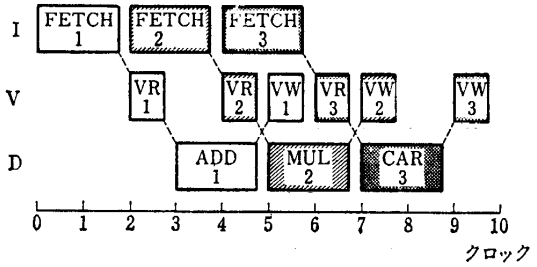

図5 FLATS パイプライン動作
Fig. 5 Typical pipeline operation.

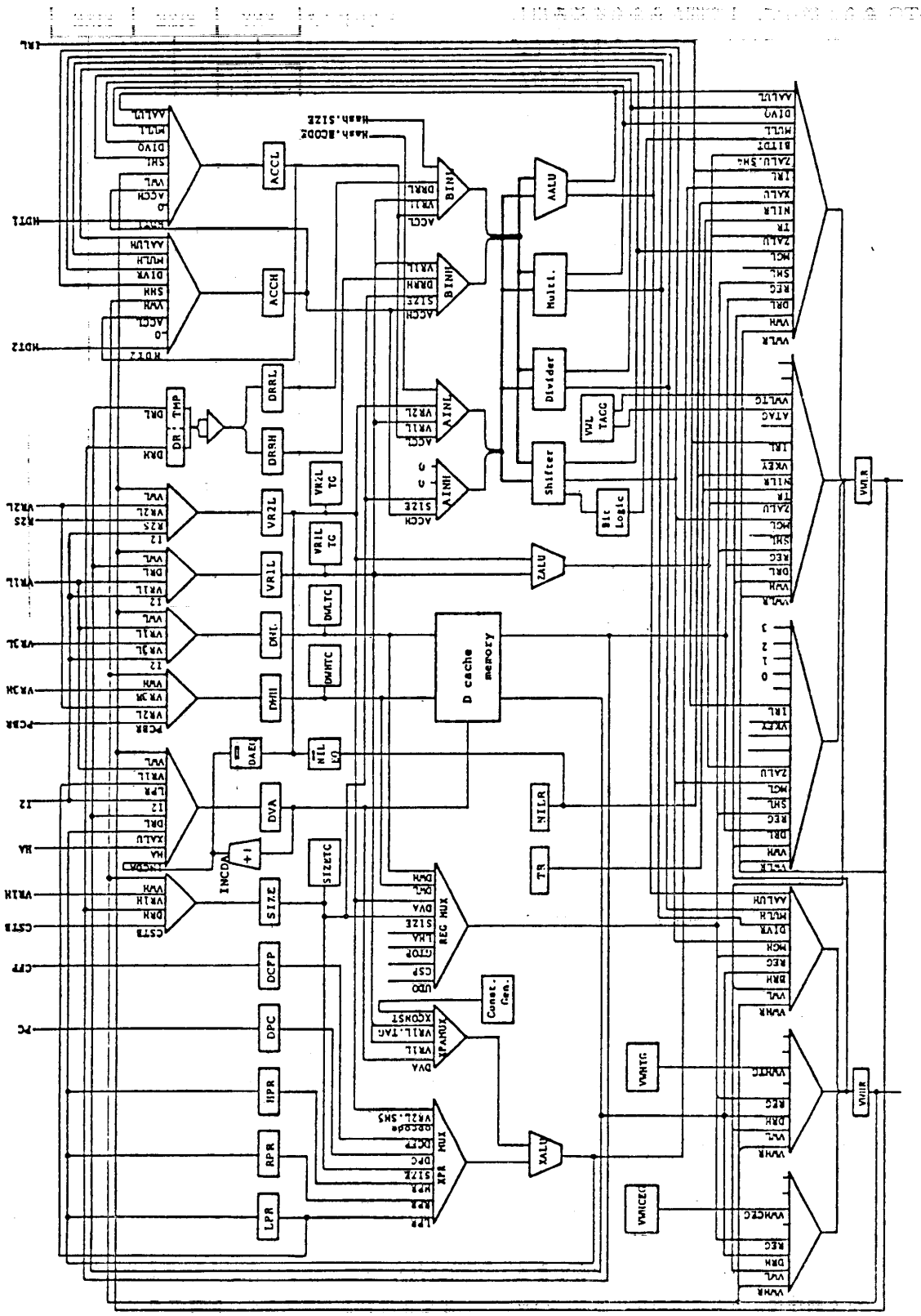

図 6 Dユニット構成
Fig. 6 D unit architecture.

る GOTO 命令に従って，I空間から命令を読み出し命令バッファに詰め，CALL 命令は命令実行パイプラインから消滅する．CALL 命令に先行する命令が条件分岐命令等であり，分岐が成立する場合には，CALL は実行されない．この場合には，仮に行ったスタック操作を取り消すことにより，正常な動作を継続する．このようにして，命令バッファが空でない場合には CALL 命令の実質実行時間はゼロとなる．

条件分岐命令をIユニットが認識すると，まず非分岐側の命令が，次に分岐側の命令がバッファに取り込まれ，Dユニットにより，いずれかが選ばれる．したがって，条件分岐により特別な実行時間の増加はなく，最短2サイクルで命令実行が終了する．

Cユニットはコントロール・スタックの最上位データを高速レジスタに保持するとともに，CALL, RET 等を先取りして0時間で実行することを実現する．

Vユニットは FLATS パイプラインの第二，第四段である．Vユニットの機能は命令のオペランドから実効オペランドのアドレスを求め，VCH をアクセスして実効オペランドを読み出し，命令終了時に実効オペランド読み出しと同時に行ったアドレス計算の結果を使用し，3組のキャッシュメモリに同時に同内容を書き込むことである．

DユニットはVユニットから渡される実効オペランドに対し，命令の実行部分を担当するユニットであり，さらに CPU 全体の管理・制御を行う．Dユニットのデータパスは**図6**に示すように汎用演算回路，メモリアクセス部，整数演算部，ハッシング部，ビット演算部，タグ検査部および出力回路部に分割される．

Vユニットからの実効オペランドは，まずタグ検査部のレジスタに入る．入力レジスタに入力したオペランドは，各レジスタに付属したタグ検査器により命令の要求しているデータ型であるか検査する．

メモリアクセス部はアドレス・レジスタ，CONS, RCONS, HCONS 命令のための自由セル・アドレス・レジスタ (LPR, RPR, HPR)，2個の ALU およびCONS のためのアドレス比較器で構成される．メモリアクセス部の使用目的は，リスプ命令やベクトル命令等Dメモリをアクセスする命令でアドレスを生成し，ベクトル領域の限界検査を行う等の操作をできる限り並行に行って高速化することである．なお，ハッシング演算における並列メモリ・アクセスはここを使用する．

整数演算部は実効オペランド，Dメモリ読み出し値，または ACCL/H 上の値に対し整数の加減乗除算を行う．乗算はビットアレイ型乗算器で，積に対する加算も可能である．除算は 48 ビット幅で，専用高速回路を使用した．

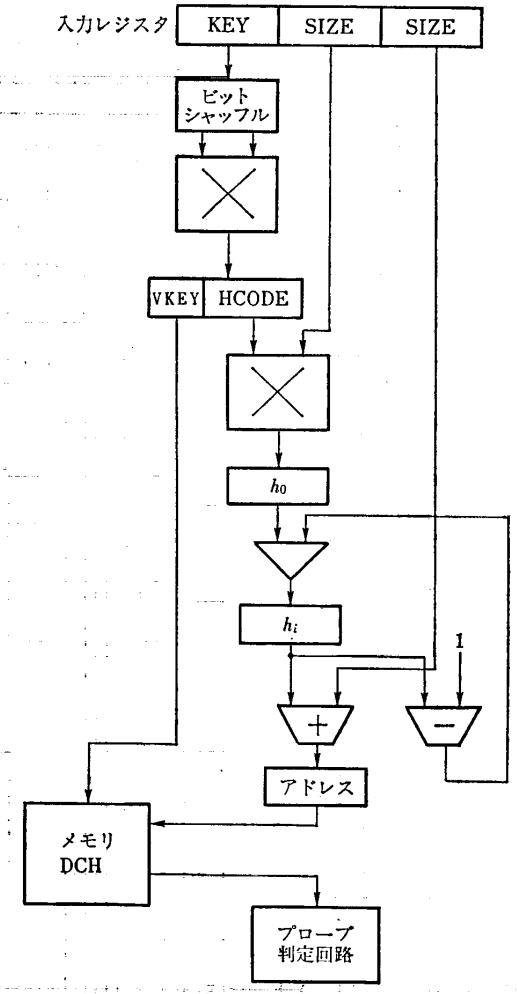

図7 ハッシング・ハードウェア
Fig. 7 Hashing hardware.

FLATS ではハッシング・アルゴリズムとして8バンクのオープン・パラレル・ハッシング・アルゴリズム[16]を使用し，またメモリを通常のデータ・メモリと共用することを特徴としている．

ハッシング命令で使用するハードウェアを**図7**に示す．8バンクのメモリとしては，DCH の1ブロック全体を使用する．比較器はキャッシュメモリに付属し，書き込みデータと読み出しデータとの比較をつねに行い，1ブロック全体について1バイトずつ比較結果を報告する．空フラグ，削除フラグは先に述べたように，データの CE タイプを使用する．比較器，CE タグの出力は全バンク分がプローブ結果判定器に接続される．

プローブ結果判定器では，各バンクからの情報をもとにバンク内の(1)マッチアドレス，(2)空アドレス，(3)削除アドレスとともに，プローブ結果として(4)ブロックに入力キーと一致したバンクが存在することを示すマッチ信号，(5)空セルが存在することを示すエンプティ信号，(6)削除セルが存在することを示すデリーテッド信号を，マイクロ・シーケンサに出力する．これらの出力の生成にあたり，AMT および HTT では全バンクを，CAT では偶数バンクのみ検査の対象とする．また AMT および CAT では一語全体をキー比較の対象とするが，HTT では V-Key 方式を採用するため MSB 側1バイトのみキー比較の対象とする．

ハッシュ表に対するプローブ・アドレスはハッシュ・アドレス発生器で決定する．オープンハッシュ法では，第一プローブアドレス h_0 に引きつづき表が衝突をおこしている場合には第二プローブ以降のプローブ・アドレス $h_1, h_2, \cdots$ を求める．第一プローブアドレス h_0 は，入力キーに対してハッシュ関数 h_0（キー，サイズ）を適用して得る．ハッシュ関数の性質は，ハッシュ表の性能に大きく影響するため，入力キーがハッシュ表中にできるだけランダムに分散する関数を使用した．具体的には，ハッシュ関数回路として，ビット・シャッフル，乗算，シフトおよび加算を組合わせて構成する．まず入力キーをビット・シャッフル回路で2組に混合・分割する．その両者の積を求め，さらにビット・シャッフル回路を経て加算器でハッシュコード (HCODE) を得る．7ビットの V-Key も同時に加算器の出力の一部として得る．HCODE および V-Key は専用の HCGR レジスタに入る．HCODE は固定小数点表示では 1.0〜0.0 の間にほぼ均一に分布すると考えられる．ハッシュ関数は HCODE とハッシュ表のサイズの積として求められる．整数演算部の乗算器は1クロックで演算が終了するので，ハッシュ関数生成に合計2クロック必要である．

FLATS では，プローブ・アドレスを順次減少させるリニア・プローブ法を使用し，

$h_i=(h_{i-1}-1) \bmod N$, N は表の大きさ．

として求める．

FLATS ハッシング命令ではDメモリに対するプローブ，次プローブ・アドレス計算，プローブ回数計数，プローブ結果判定がすべて並行に行われ高速化している．

出力回路部は，Dユニットのすべての機能部分に接続した選択器と出力レジスタで構成される．出力回路部の出力は，出力レジスタがVユニットに対し書込オペランド値となるとともに，出力選択器の出力が，Dユニット内部の各種レジスタの入力選択器に接続され，Dユニット内部でのデータ分配にも使用される．また命令間で，出力データを直接入力データとして扱ういわゆるオペランドの Bypass にも使用する．

Dユニット内のすべてのデータパスおよびコントロールおよび他ユニットに接続している制御信号は，マイクロプログラムにより制御される．マイクロプログラムは完全水平型マイクロプログラムであり，容量 1,024 語，語長 240 ビットである．命令記述のために約 10,000 行のマイクロプログラムを使用した．

記憶装置の基本方式として，3個のキャッシュメモリ，主記憶装置，二次記憶装置の3階層をもつデマンド・ページング方式を採用した．各階層間のデータ転送単位は，CPU とキャッシュメモリ間では 64 ビット，キャッシュメモリと主記憶装置間では 32 バイトの大きさをもつブロック，主記憶装置と二次記憶装置の間では 2 k バイトから 8 k バイトの大きさをもつページである．キャッシュメモリの方式としては，いずれも論理アドレスを用いるセットアソシアティブ方式で，ICH は4ウェイ，VCH は1ウェイ，DCH は4ウェイ構成である．

キャッシュメモリからの要求に対しては論理・物理アドレス変換を行い主記憶装置へ伝える．主記憶管理装置内のアドレス変換表は 10 バンク並列ハッシュ表で構成し，全論理アドレスについて平均 1.04 クロックで変換を終了する．ハッシング・アルゴリズムとしてはリニア・プローブ法を使用する．

5. 性　　能

命令，実効オペランド，データがキャッシュメモリ上に存在する場合，非分岐型パイプライン命令，例えば加算等の命令では，図5に示すように完全にパイプライン的に実行され，実行時間はDユニット実行時間で決定される．Dメモリをアクセスする命令，リスプ命令の CONS, LIST 2, RCONS, HCONS, ベクトル命令等では，キャッシュメモリでは1回のメモリアクセスに最低2クロック必要であるため，命令実行の第一クロックで唯一回のメモリアクセスを行う命令が2クロックで実行される．

EQ, GE, GT など整数演算の結果に従う分岐型命令は，すべて命令の第一クロックで判定するため，分岐

表 2 命令の実行速度
Table 2 Instruction execution speed.

算術命令		ハッシング命令	
ADD	2	GETAMT	6～
MUL	2	PUTAMT	8～
DIV	15	REMAMT	7～
条件分岐命令		サブルーチン命令	
BEQ	2	CALL	0
BGE	2	RET	0
ATOM (BTSM)	2		
GOTO	0		
リスプ命令			
CAR	2～3	RPLACA	3～4
CDR	2～4	RPLACD	3～6
CONS	2～3	LIST 2	3～4

表 3 プログラムの実行速度
Table 3 Program execution speed.

プログラム名	FLATS	M 380
	(単位ミリセカンド)	
tarai (10 5 0)	900	1,392
tarai (12 6 0)	25,360	51,087
srev (0 1 2 3 4 5 6 7 8 9)	2,540	1,040
for i=1 : 20 do $(x+1)**10$	120	92
for i=1 : 20 do df $((x+1)**10, x, 10)$	180	132

に伴う余分の時間遅延が発生せず，2クロックで命令実行を終了する．分岐命令と無条件分岐命令 (GOTO) が複合して用いる場合には，GOTO 命令は連続しないときに0時間で実行されるため，GOTO 命令を含めて2クロックで実行を終了する．CAR, CDR, ハッシング命令およびビットアクセス命令は入力引数またはアクセスしたメモリの内容に従って分岐動作も行う．しかしながら，分岐動作自身は前項と同様特別な時間遅れを作らず，実行時間はメモリアクセスで決定される．例えば，CAR はリストが間接ポインタを含む場合3クロックで，それ以外の場合には2クロックで実行を終了する．ハッシング命令の実行速度はハッシュ表の状況により大きく変わる．ハッシュ表がほとんど空である状態では，ディスクリプタの読み出しとハッシュアドレス生成に3クロック，プローブに1クロック，プローブ結果の検査および後処理に2クロック，計6クロックが実行時間である (GETAMT の場合)．もしハッシュ表が満員に近い場合には，衝突が多く発生し，プローブが繰り返し実行される．しかしながら，平均プローブ回数は8バンク並列ハッシュ方式を使用しているため，低く抑えられる．

純パイプライン命令，すなわち GOTO, CALL, RET 命令は I, C および V ユニットだけですべての操作が実行されるため，連続して実行しない限り実行時間は0である．代表的な命令の速度を**表 2** に示す．

次に Lisp および数式処理の例題について実行速度を示す．プログラムはすべてクロス・コンパイラで FLATS 機械語に変換したものを用いて測定を行った．比較対象として，M 380 とともに実行時間を**表 3** に示す．なお，FLATS および M 380 のクロック時間はおのおの 120 および 15 ナノセカンドであり，双方とも基本命令を2クロックに1命令の速度で実行する．

6. おわりに

数式処理計算機 FLATS は Lisp 言語を高速に実行するとともに，数式処理システムに必要な機能をサポートすることを目的に設計製作した．汎用計算機と比較した FLATS の特徴は，(1)短い4段のパイプライン構成により，Lisp プログラムに多発する条件分岐命令をオーバヘッドなしに実現すること，(2)スタックをコントロール・スタックとフレーム・スタックに分割し，さらにフレーム・スタックに3ポート並列読み出しメモリを使用することにより，メモリ転送能力を高め命令実行時間を短縮すること，(3)コントロール・スタックを専用ハードウェアにより実現することによって，Lisp プログラムで頻発するサブルーチン呼び出し，復帰命令の実行時間をほとんどの場合について0とすること，(4)3オペランド命令形式およびショート・ジャンプ・オペランドにより，スタック上の移動，操作，データ型検査を並行に実行可能となり，Lisp プログラム中での実行する命令数を減少可能とすること，(5)並列ハッシングハードウェアを使用して，間接アドレシングとほぼ同じ速度で連想記憶を実現すること，(6)ガーベジ・コレクションをハードウェアでサポートすることにより，ガーベジ・コレクションのオーバヘッドを軽減すること，および(7)並列ハッシングを使用した仮想記憶ページ変換表により，全物理ページを高速にアドレス変換することを可能としたことである．

上記の特徴は一個の計算機としてまとめられて初めて総合的な有効性を発揮する．これらの特徴により，FLATS は素子の速度またはクロック速度が8倍程度速い汎用計算機と同程度の速度で動作した．すなわちアーキテクチャによる加速の度合が約8倍である．

謝辞 FLATS 計画の実現に対し理化学研究所情報科学研究室の相馬，出沢，佐々木，井田，稲田の各氏

に感謝する．ソフトウェアに関連した項目の決定には，東京大学の鈴木，理化学研究所の稲田両氏に負う点が多い．製作にあたっては東京大学の清水製作のシミュレータを使用した．

参考文献

1) Hearn, A. C.: REDUCE-2 User's Manual, University of Utah, Utah (1973).
2) M. I. T. The MATHLAB Group: MACSYMA Reference Manual, LCS, M. I. T. (1974).
3) Clark, D.W. and Green, C. C.: An Empirical Study of List Structure in Lisp, *Comm. ACM*, Vol. 20, No. 2, pp. 78-87 (1977).
4) Deutsch, L. P.: A LISP Machine with Very Compact Program, Proc. IJCAI, pp. 697-703 (1973).
5) Knight, T.: *CONS*, AI Lab. M. I. T. (1974).
6) Greenblatt, R. et al.: *The LISP Machine*, AI Lab. M. I. T. (1974).
7) Kenneth, A. P.: A Retrospective on the Dorado, A High-Performance Personal Computer, Proc. of 10th Annual Symp. on Comp. Arch., pp. 252-269 (1983).
8) Hayashi, H. et al.: ALPHA: A High-Performance LISP Machine Equipped with a New Stack Structure and Garbage Collection System, Proc. of 10th Annual Symp. on Comp. Arch., pp. 342-348 (1983).
9) 日比野，渡辺，大里：LISP マシン ELIS の基本設計，情報処理学会記号処理研究会，12-15 (1980).
10) 前川他：試作 EVLIS マシンの EVAL II と開発支援機能，情報処理学会記号処理研究会，17-1 (1982).
11) Taki, K. et al.: The Experimental LISP Machine, Proc. of 6th IJCAI, pp. 865-867 (1979).
12) Suzuki, M. and Ono, K.: A Primitive for Non-recursive Lisp Programming, *J. Inf. Process.*, Vol. 4, No. 4, pp. 208-210 (1981).
13) Hansen, W. J.: Compact List Representation: Definition, Garbage-collection and System Implementation, *Comm. ACM*, Vol. 12, No. 9, pp. 499-507 (1969).
14) Goto, E., Sassa, M. and Kanada, Y.: Algorithms and Programming with CAMs, *J. Inf. Process.*, Vol. 3, pp. 13-22 (1980).
15) Terashima, M. and Goto, E.: Generic Order and Compactifying Garbage Collectors, *Inf. Process. Lett.*, Vol. 7, No. 1, pp. 27-32 (1978).
16) Goto, E., Ida, T. and Gunji, T.: Parallel Hashing Algorithms, *Inf. Process. Lett.*, Vol. 7, No. 1, pp. 8-132 (1978).

(昭和 60 年 3 月 12 日受付)
(昭和 60 年 7 月 18 日採録)