解　説

# 高級言語マシン*

島田俊夫** 坂村　健*** 山口喜教**

## 1. まえがき

高級言語マシンとは「高級言語で書かれたプログラムを直接実行する計算機」のことであるが,"直接実行する"という意味がいろいろに解釈できるので, Y. Chu の用いた分類[1]で, もう少し厳密に定義してみよう. それによれば計算機は

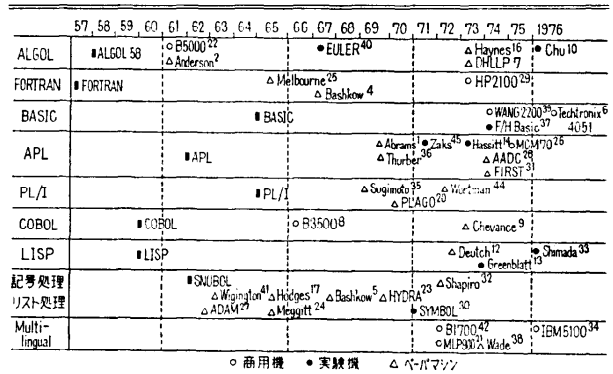
（1） ノイマン型
（2） 文法指向型
（3） 間接実行型
（4） 直接実行型

の4つに分類することができる. 分類には"言語の近接性"という概念が用いられる. ここでは高級言語と機械語の近接性を意味するわけであるが, それは言語間の「語彙」,「文法」,「意味」の3つの要因を比較して決定される.「語彙」というのはその言語で用いることのできる文字, 数字, 名前, 演算名, デリミッタ等を意味し,「文法」とは規則及びそれを構成する要素, たとえばシラブルとか文とかブロック等を意味し,「意味」とは演算子, オペランド, 操作コードの内容を意味する.

（1）のノイマン型は高級言語をコンパイラによってリロケータブル・コードに翻訳し, リンケージ・エディタでアブソルート・コードに変換し, ローダでロードして実行する. 現在の商用機の大部分がこの型である. （2）の文法指向型は, 高級言語をコンパイラによってポーリッシュ・ストリング・コードに翻訳し, 実行する. Burroughs の B 5500[3,22]がこの型の代表である. コンパイラをファームウェアにより実現したものもこの型に含めると, EULER[40], Zaks 等の APL マシン[45], Deutch の LISP マシン[12]等, 大部分の高級言語マシンがこの型に含まれる. （3）の間接実行型は, （2）と同じ処理方式であるが, コンパイラもハードウェアとなっている点が異なる. SYMBOL[30]や Bashkow 等の FORTRAN マシン[4]がこの型である. （4）の直接実行型は高級言語を直接インタプリットする. コンパイラは存在しない. Chu のマシン[10]と島田等の LISP マシン[33]がこの型に入る.

以上4つの型のうち（2）（3）（4）を高級言語マシンと呼ぶことにする.

表-1 高級言語マシン年表




## 2. 歴史と動向

高級言語マシンの研究は 1961 年の Anderson の ALGOL マシン[2], Lonergan の B 5000 システム[22]が始まりといわれている. それから現在までに数多くの高級言語マシンが研究されている. それらを言語別に年表として示す(**表-1**).

これほど多くの高級言語マシンが研究される理由は, 高級言語を高速に効率良く実行したいという要求があるからである. そしてこのことは汎用計算機が高級言語を効率良く処理していないということを意味している.

高級言語は本来, 応用分野のプログラムを容易に, 簡潔に記述するために考えられたものであり, 機械語

* High level language machine by Toshio SHIMADA, Yoshinori YAMAGUCHI (Computer Division, Electrotechnical Laboratory) and Ken SAKAMURA (Desartment Electrical Engineering, Keio University).

** 電子技術総合研究所
*** 慶応義塾大学

にくらべて制御構造とデータ構造がかなり複雑になっている．たとえば制御構造では if-then-else や while 等のループ，begin-end によるブロック構造，リカーシブコール，コルーチン等，データ構造ではリスト，ベクトル，記号処理等がその代表である．これらの複雑な構造を汎用計算機の機械語で記述すると，冗長な部分がでてくるため，プログラムが長くなり，実行速度が遅くなるというのが汎用計算機の効率の悪さに対する説明である．

一方，高級言語マシンは対象とする高級言語の構造を考慮してファームウェアやハードウェアを設計するので，汎用計算機より効率が良いはずである．しかし高級言語マシンは汎用計算機より構造が複雑なため，実現は困難である．そのため初期の高級言語マシンはほとんどペーパマシンに終っている．ところがその後ハードウェア技術が進歩したため，1970 年頃には比較的容易に実現できるようになった．表-1 でもそのことが顕著に表われている．

ここで高級言語マシンの実現に大きな力のあったハードウェア技術について述べよう．

第一はマイクロプログラム制御技術である．これは 1951 年 M. V. Wilks により提案されたものであるが，高級言語マシンに登場してくるのは 1963 年の Wigington のマシン[41]からであろう．実際にマイクロプログラムを用いて高級言語マシンが実現されたのは 1967 年 Weber の Euler マシンが最初であろう．マイクロプログラムは変更が容易であるため設計に柔軟性が増すこと，製作後の最適化が可能なこと，製作が容易なこと等の特徴があり，マイクロプログラムを用いれば，高級言語の持つ複雑な制御構造やデータ構造をそのままワイアドロジックで作る必要がない．マイクロプログラムは，始め読出し専用の記憶 (ROM*) に書かれていたが，1970 年頃には書き換え可能な記憶 (RAM**) も使われるようになった．RAM を用いるとディバッグや最適化が一そう容易になるため，最近では実験機はほとんどすべて，商用機では B1700[42]，HP 2100[29]等が RAM を使用して高級言語マシンを実現している．

ハードウェア技術の第二は半導体技術である．IC が用いられるようになって計算機のハードウェアが簡潔になり，複雑な機能も比較的容易にハードウェアで実現されるようになった．その具体的な表われは 1971 年の SYMBOL で，高級言語の処理機能とオペレーティングシステムの機能の大部分をハードウェアで実現した．

IC の集積度が更に上がるにつれてハードウェアの価格が下がり，性能も向上したので過去には用いることのできなかったハードウェアを使用することが可能になってきた．先に述べたマイクロプログラム用の RAM もその一つであるが，最近ではビットスライス型のマイクロプロセッサと PLA*** を挙げることができる．ビットスライス型のマイクロプロセッサはレジスタ・トランスファのレベルから高級言語マシンを設計し実現するための有力な手段となるし，PLA もデコード部分の高速化を手始めとして，しだいに多く用いられるようになるだろう．

次に高級言語マシンの機能面での変化をみてみると，複数の高級言語を対象にしたマシンとパーソナルコンピュータが目につく．

複数言語マシンは対象が一つの言語でないので，汎用機能を多く持つことになる．そのため単一言語のマシンにくらべて効率が悪くなるが，ノイマン型の計算機よりは良くなると考えられる．複数言語マシンの例としては B 1700, IBM 5100[34]等がある．

パーソナルコンピュータはハードウェアの価格が下がり，サイズも小さくなったのでコンピュータを電卓なみに個人で専有しようというもので，ユーザの使い易さを考えて高級言語マシンとして実現されることが多い．IBM 5100, WANG 2200[39], Techtronix 4051[6] 等の商用機がある．Greenblatt の LISP マシン[13]はパーソナルコンピュータのネットワークを組むことによって，現在の TSS にかわるシステムの実現を目指している点が興味深い．

次に高級言語マシンのアーキテクチャの変遷を眺めてみると，あまり大きな変化はないようである．マイクロプログラム制御技術とマルチプロセッサシステムは初期のマシンに既に登場しているし，内部リソースの充実，たとえばレジスタ数を増すこと，スタックや高速メモリを持つこと等も初期のマシンで既に提案されている．最近の高級言語マシンの研究は，これら既出の概念を最新の技術で実現した場合の性能，価格，ソフトウェアとファームウェアとハードウェアのトレードオフ等を調べること，パーソナルコンピュータ等新しい機能を追求することが主になっている．

---

* Read Only Memory
** Random Access Memory
*** Programmable Logic Array

表-2 言語の処理過程の内容と負担の大きさ

| Level | Sequencing | Instruction fetch | Instruction decoding | Operand accessing | Data operations |
|---|---|---|---|---|---|
| Machine language | conditional branch, subroutines | | fixed format | immediate, indirect, indexed | add, multiply, or, complement |
| | medium | high | high | medium | low |
| BASIC and many system prog. languages | iteration | | simple syntax | subscripted data structures | sine, cosine, matrix operations |
| | medium | medium | low | medium | medium |
| FORTRAN, PL/1 ALGOL, etc. | block structure recursion, coroutines | | | non-rectangular data structures | I/O formatting |
| | high | low | low | medium | medium |
| LISP, SNOBOL, Simula/67, APL, etc. | parallel processes, synchronization, message sys. | | | linked lists, associative searching | vector operations garbage collection |
| | high | low | low | high | high |

## 3. 高級言語マシンのアーキテクチャ

高級言語マシンのアーキテクチャの特徴は，汎用計算機がソフトウェアで処理する部分をファームウェアやハードウェアで実現している所にある．ここでは高級言語の処理を翻訳過程と実行過程に分け，アーキテクチャの特徴を述べる．

汎用計算機は翻訳をソフトウェアで行うが，高級言語マシンの中にはこれをファームウェアで行うものがある．またSYMBOLのようにハードウェアで行う例もある．しかしそこで用いられるアルゴリズムはソフトウェアの場合と本質的に同じであり，アーキテクチャ的にみて特徴的なものはないようである．

実行過程は命令フェッチ，命令デコード，順序付け，オペランドアクセス，データ操作の5つのフェーズに分けることができる．レベルの異なる様々の言語について各フェーズの内容と負担の大きさを示すと表-2のようになる[49]．

低レベル言語ではフェッチとデコードの比率が高く，他は低い．逆に高級言語ではフェッチとデコードの比率が低く順序付け，オペランドアクセス，データ操作の比率が高い．これは高級言語が複雑な制御構造やデータ構造を採用し，1個の命令で複雑な内容を表わすためその実行に時間がかかり，相対的にフェッチ・デコードの比率が下がるのに対し，低レベル言語では各命令の仕事の内容が単純で実行時間が短いため，フェッチ，デコードの時間が無視できなくなるためである．このことは高級言語マシンの効率改善のためには，どこをファームウェア／ハードウェア化すれば効果があるかを示している．

次に，各高級言語マシンが各フェーズをどのようにファームウェア／ハードウェア化し，効率を改善しようとしているかを述べる．

### 3.1 順序付け

順序付けとは言語の制御構造を処理することである．対象としては，条件付ジャンプ，サブルーチン，イテレーション，ブロック構造，リカーション，コルーチン，並列処理等がある．制御構造を取り扱うハードウェアで最も一般的なものはスタックであろう．

スタックは数式処理や演算にも，しばしば使用されるが，制御構造の処理にもよく用いられ，サブルーチンやリカーションの戻り番地をストアしたり，サブルーチンのパラメータの引渡しに使う方法が良く知られている．こうした操作を高速に行うためスタックの構造を工夫した例[11,52]もある．

しかしスタック使用の最も巧妙な例は Burroughs の計算機による ALGOL のブロック構造の処理であろう．ALGOL では変数の範囲はその変数を使用するプロシージャがプログラム上のどこで定義されるかで決まる．すなわちプログラムの静的な構造で決まる．一方プロシージャを呼出す順序はプログラム上で定義した順序とは必ずしも一致しないからプロシージャの呼出しの順序は動的にしか定まらない．B6500[15]では Mark Stack Control Word (MSCW) を設け，これを DISP と DF という2つのフィールドに分割し，DISP には静的な構造を表わすアドレス・エンビロンメント・リストを置き，DF には動的な構造を表わすスタック・ヒストリ・リストを置いている．この2種のポインタをたぐることによって変数の値の決定と，サブルーチンの管理を行うわけである．例を示そう．

プログラム（図-1）はB6500のスタック上で図-2のように表現される．スタック・ヒストリ・リストを使用すればサブルーチンがリターンするとき1語ずつPOPする手間がはぶけて効率が良くなる．一方変数の値はいちいちアドレス・エンビロンメント・リストをたどって決定せねばならず効率が悪い．B 6500はこれを次のように処理している．まずコンパイラが各変数をレベルとインデックスのペアとして表現する．図-1のプログラムでV3は(3,2)と表わされる．

実行中にV3の値を見るときはディスプレイ・レジスタD3の指す所から2番地上を見るとV3がストアされている．変数V4も(3,2)と表わされるが，実行中にV3とV4が共存することはないのでうまく働く．

条件付ジャンプやサブルーチンの呼出しはそれほど複雑でないのでワイアドロジックで実現することも難しくないが，マイクロプログラムで処理する例も多い．コルーチンや並列処理については，これらの機能を持つ言語があまり普及していないため，実現例[33]はわずかである．

### 3.2 命令のフェッチ

命令フェッチについては高級言語マシンに特有のものはなく，一般の計算機と同じように命令を先取りしたり，キャッシュを用いたりしている．

### 3.3 デコード

高級言語マシンでは命令語やデータ語にタグを多用するので語形式が幾つものフィールドに分割されることになる．これらのフィールド処理を高速に行うものとして高速のシフタやマスカ，間接ジャンプ命令等の汎用機能と，ランゲージボード，ROMジャンプ表，PLAデコーダ等の専用機能がある．単一の高級言語を対象にする時は専用の機能を用いた方が価格，性能共に有利であるが，複数の言語を処理する場合は汎用の機能が必要となる．プロセッサをLSI化する場合は汎用機能の方が一般的であるから大量生産向きであるといえる．具体例としてKnightのCONSマシン[19]の汎用機能を説明しよう．CONSはLISP向きの高級言語マシンであるが，他の言語を実行することも考慮した汎用機能を持っている．構成は図-3のように3バス構成で，マイクロ命令は，(1) ALU操作，(2) バイト操作，(3) 条件ジャンプ，(4) ディスパッチの4種類がある．デコードに用いるのは(2)と(4)で，(2)はMバス上のデータの部分フィールドを抽出し，Aバス上のデータの部分フィールドと置きかえる命令で図-4（次頁参照）のような3つのモ

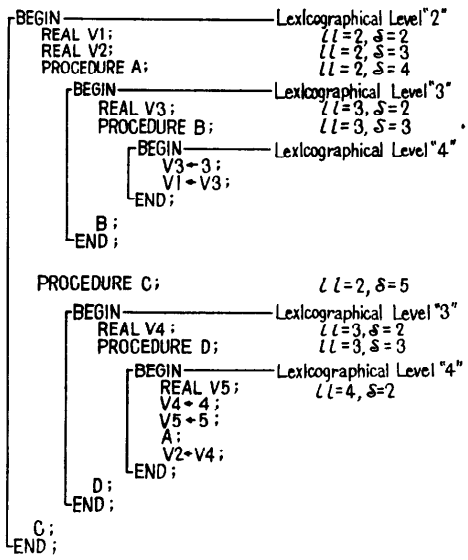



図-1 ALGOLのプログラム

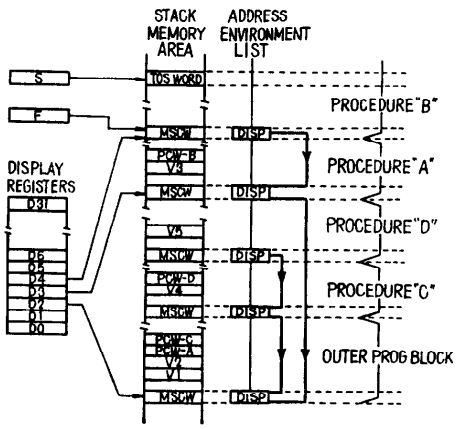



図-2 プロシージャBを実行する時のスタックの状態

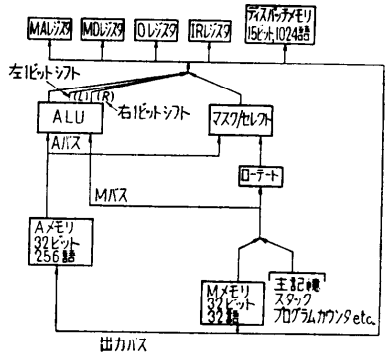



図-3 LISPマシンブロック図

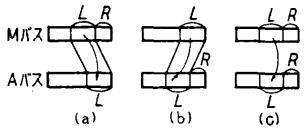



図-4 LISP マシンのバイト操作

ードがある．(4)は M バス上のデータの最大8ビットまでの部分フィールドを選択し，命令語中の別のフィールド（10 ビット）と OR を取って作成した 10 ビットをアドレスとしてディスパッチメモリから 15 ビットのデータを読み出し，その 15 ビットをジャンプコードとジャンプ先アドレスとして，ジャンプを実行する．命令語の解釈等に便利な機能である．

ランゲージボードは MLP 900[21]に採用された機構で，その働きは命令に論理操作を行ってジャンプ用のアドレスを作り出したり，アドレス計算用のレジスタの指定と外部入力データの適当なフィールドをそのレジスタに入れること等であるが，各言語ごとに専用のボードを持つので汎用機能より高速処理が可能である．

ROM と PLA は，ほぼ同じ機能を持っており，デコード論理をワイアドロジックで実現する便利なデバイスである．ROM は既に多く使用されているが PLA はまだあまり使用されていない．しかし使い易さ，高速性，価格等の点で優れたデバイスなので大いに使用されるようになるだろう．

### 3.4 オペランドアクセス

オペランドアクセスに関する問題はデータタイプとデータ構造の処理である．これらを効率良く処理する機構としてはタグ，連想メモリ，キャッシュ等が考えられる．この中でキャッシュは既に商用機で広く用いられているので省略し，タグと連想メモリについて述べよう．

タグ方式とは記憶内のすべてのデータ要素に，そのデータ自身の意味を表わす印をつける方式のことであり，その印のことをタグと呼ぶ．現存する大部分の計算機では，データは語またはバイトという定められた形式の中に並べられた意味のない数列であり，そのデータの意味は，それを操作するプログラムによって与えられるが，タグ式計算機ではデータそれ自身が意味を持っているので，複雑なデータ構造を操作するときに様々の利点が生じる．タグとして表示されるデータタイプの種類の主なものを表-3[50]に示す．

タグ方式を利用した例としてハードウェアによる自動エラー検出を Basic Language Machine[18),51)]で説明しよう．Basic Language Machine は 1968 年 Illiffe によって開発された計算機で，その語形式は2ビットのタグを持っており，0は整数，1は空，2は実数，3は番地を表わしている．“$X$ ADD $Y$”という演算を行うとき，$X$ のタグ値が0または2であれば実行可能であり，1であれば実行不可能としてエラーとなる．タグ値が2の時は $X$ は番地であり，$X$ の中のタグに続く2ビット（タイプビットという）が調べられる．タイプ値はそれが参照しているデータのタイプを示しており0が整数，1が命令コード，2が番地，3が実数を表わす．もし $X$ のタイプ値が0または3のときは $X$ が参照するデータがもってこられる．1のときはエラーであり，2のときは番地だから再び同じ手順でタイプ値が2以外のものが現われるまで続けられる．$Y$についても同様のことが行われる．番地参照の例を1レベルに限るとして $X$ と $Y$ の取りうるタイプは7通りあるので 7×7=49 通りの組み合わせがあり，そのうち演算可能なものは 4×4=16 通り

表-3 タグの種類

| Standard Hardware-Recognized Type Codes | |
|---|---|
| int: | integer |
| real: | real number |
| long int: | double-precision integer |
| long real: | double-precision real |
| complex: | single-precision complex |
| long complex: | double-precision complex |
| undf: | undefined |
| mixed: | mixed types (indirect only) |
| char: | character (indirect only) |
| Bool: | Boolean (indirect only) |
| vec: | vector of |
| ref. | reference to |
| label: | label in *i*th environment |
| matrix: | matrix of |
| svec: | sparse vector of |
| sll: | single-linked list of |
| dll: | double-linked list of |
| stack: | stack of |
| q: | queue of |
| ms: | machine state of |
| msg: | message from-to |
| ipt: | interrupt of |
| ev: | event |
| ps: | parameter set for |
| proc: | procedure-environment designator |
| name: | name of variable |
| i.d.: | i.d. of process or user |
| instr : | instructions |
| file: | file |
| formal: | formal parameter |
| sema: | semaphore |
| garbage: | garbage |

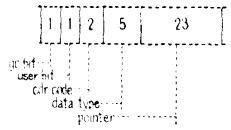



図-5 LISP マシンのデータ語形式

である．その他の場合はハードウェアによりエラーが検出される．この方式はデータのタイプにかかわらず ADD という一個の命令で加算が行えるので，コンパイラの作成も容易になる．

タグを効率改善のために使用した例としては Greenblatt の LISP マシンがある．LISP マシンのデータ語の形式は**図-5** のようになっており，32 ビットのうち9ビットがタグにあてられている．cdr コードはリストセルの cdr がどのような状態にあるかを示すタグで，これを調べることにより，次のデータをフェッチする手間が省けることがある．次の5ビットのデータタイプはリスト，アレー，字，整数，実数等を表わすが，興味深いのはインヴィジブルポインタと呼ばれるものである．これはリストをアレーとして表現した場合，そのリストの変更，修正の能率が悪くなるのを防ぐ手段である．

タグ方式の欠点は従来の計算機との互換性がないこととコスト高になることにある．しかし効率の改善やディバッグには非常に有効な手段なので，ハードウェアのコストが低下すればもっと広く用いられるようになるだろう．

次に連想メモリについて述べよう．

MU-5[53]では最近アクセスされた変数の名前を 64 語のネーム記憶という所に入れ，変数の高速アクセスに利用している．データ構造を取り扱った例としては3値機能メモリを用いて SNOBOL のストリングのマッチを行う提案がある[54]．連想メモリや機能メモリのようにメモリに構造をもたせようという研究は数多くあるが，その有効性は必ずしも明らかでなく，高級言語マシンへの応用もそれが有効であるかどうかを含めて残された課題といえよう．

### 3.5 データ操作

数値演算向きの言語で問題になるのは浮動小数点演算と I/O の formatting であろう．前者については専用のハードウェアを持つのが一般的になっている．後者についてはマイクロプログラムによる操作が十分な速度を与えると考えられている．

APL のようにベクトルを大量に処理する言語では，ベクトル演算用のハードウェアが用いられる．マイクロプログラムによる処理も考えられるが，アレイプロセッサによる処理の方が実行速度が速い．

記号処理系の言語では不用になった記憶スペースを回収するガーベッジコレクションが問題になる．これについては特に考案されたものはない．最近はガーベッジコレクションを並列に行うことに関心が持たれているようである．

## 4. 高級言語マシンの評価

高級言語マシンの性能を評価する規準として（1）コンパイル時間，（2）コンパイラの大きさ，（3）実行時間，（4）オブジェクトコードの大きさ，（5）診断，ディバッグ機能等が考えられる．これらについて実測した例を紹介しよう．

Euler マシン——IBM 360/30 のマイクロプログラムで実現されている．データは（2）のみで，マイクロプログラムのコンパイラが機械語のコンパイラの 1/6 程度のメモリサイズである．

APL マシン[14]——IBM 360/25 のマイクロプログラムで実現されている．データは（3）についてである．使用されたステートメントは A←B+C で，B，C がスカラーの場合と，長さ $n$ のベクトルの場合が測定された(**表-4**)．$n$ が小さいところで機械語の方が速い理由は，APL マシンはインタプリタ方式で，実行時のスキャン等に時間を取られているのに対し，機械語の方はハンドコンパイルしたベストコードを実行しているためだと説明されている．プログラム単位の比較では 6×6 の行列の逆行列を計算する場合を 360/25 の FORTRAN とくらべている．それによると FORTRAN は 3.8 秒，APL マシンは 3.2 秒となっている．

LISP マシン[33]——HP 2100 のマイクロプログラムとアセンブラを用いて同じ仕様のインタプリタを記述し，比較している(**表-5**)(次頁参照)．この表のステップ数はインタプリタ内の対応する処理ルーチンの大きさを表わしている．

**表-4** APL マシンの性能

| | APL machine | IBM 360 best |
|---|---|---|
| Scalar | 682 μsec | 298 |
| $N=5$ | 2,385 | 1,976 |
| 10 | 3,985 | 3,911 |
| 20 | 7,185 | 7,781 |
| 40 | 13,585 | 15,521 |
| $n$ | $785+320n$ | $41+387n$ |

表-5 LISP マシンの性能

| 命 令 | 機械語のステップ数 | マイクロコードのステップ数 | 速度比 |
|---|---|---|---|
| CAR | 2 | 4 | 3 |
| CDR | 3 | 5 | 4 |
| GET | 24 | 26 | 4 |
| NUMBERP | 30 | 28 | 4〜6 |
| SLOAD | 29 | 34 | 6〜7 |
| PUSHFT | 19 | 28 | 5 |
| POPFT | 16 | 22 | 4.4 |

Basic マシン[37]——Basic のインタプリタをトランスレーションフェーズとインタプリテーションフェーズに分け，各々をアセンブラ，マイクロプログラム，マイクロプログラム＋ハードウェアの3つの手段で実現し，実行速度を比較している(**表-6**)．この表はアセンブラを1としたときの倍率を示しており，Basic ミックスはステートメントの出現頻度を重みとして平均したものである．トランスレーションで約 58 倍，インタプリテーションで約 16 倍の性能向上が見られ，他のマシンの場合より良い値を示しているが，これはアセンブラを実行したマシンとマイクロプログラムを実行したマシンが異なることも原因していると思われる．このマイクロプログラムに更にマスク機能，オートフェッチ／ストア機能，ポインタ操作機能，アソシアティブ機能等のハードウェアを付加すると，トランスレーションで 59%，インタプリテーションで 48% 速度が向上する．付加したハードウェアのコストは CPU 価格の 4% 程度なので価格／性能比がかなり向上したと考えられる．

以上4つの例をあげたが，これらの数値はマイクロコードの記述力，制御記憶と主記憶の速度差等によって大きく変化する．上の例ではマイクロコードがいずれもその言語に合わせて設計されたものでないので少し低目の数値を示していると考えられる．

Illiffe はマイクロプログラムで高級言語マシンを実現した場合，先の5つの規準についての楽観的予想を次のように述べている[18]．(1)語い解析，シンタックス解析は5倍程度速い．(2)コンパイル時間を犠牲にすれば減らすことができる．(3) 10 倍位まで高速化可能．(4) 40〜60% 減らすことが可能．(5) たいしたオーベヘッドなしに実行時のエラー診断情報が取れる．

いずれにしてもマイクロプログラムによる高級言語マシンはある程度の性能向上が見込め，マイクロプログラム制御技術が計算機製作の側からみてもメリットが大きいので，書き換え可能な制御記憶を実現手段として次第に普及していくだろう．一方ハードウェアとファームウェアを併用する高級言語マシンは，どのような機能をハードウェア化すべきかを研究している段階である．

表-6 Basic マシンの性能

| ステートメント | トランスレーション | インタプリテーション | 合 計 |
|---|---|---|---|
| LET | 48.5 | 12.4 | |
| IF | 67.4 | 56.4 | |
| GO TO | 99.5 | 29.4 | |
| GO SUB | 53.2 | 46.2 | |
| RETURN | 108.5 | 52.0 | |
| 入出力 | 68.6 | 8.5 | |
| その他 | 70.3 | 25.8 | |
| BASIC ミックス | 58.4 | 16.2 | 40.2 |

注) 入出力: READ, INPUT, PRINT.
その他: DATA, DIM, REM, RESTORE, PAUSE, END, STOP.

## 5. あ と が き

高級言語マシンの研究も既に 15 年程の歴史を持っているがまだ広く普及してはいない．高級言語を処理する限り，汎用計算機より高級言語マシンの方が様々の利点があるはずである．それにもかかわらず高級言語マシンが普及しない理由を2つばかり挙げておこう．

第一は現在使用されている高級言語の大部分は FORTRAN と COBOL で，その占有率は 90% 以上であり，これらの言語は実行時の処理が比較的少ないため，現在の汎用計算機でかなり効率良く処理できるということである．また汎用計算機の方も FORTRAN や COBOL 向きの命令を付加して処理効率を改善しているので，高級言語マシンがこの2つの言語に関して圧倒的にすぐれた性能を示すことができないでいる．

第二は過去のマシンとの互換性がないことである．ソフトウェアの蓄積の影響が特に大きい中・大型機ではアーキテクチャの大幅な変更を必要とする高級言語マシンの普及は難しいだろう．

最後に高級言語マシンと機能分散の関係について述べておこう．現在の高級言語の大部分は，並列に処理できる部分が非常に少ない．そこで処理過程をパイプライン化したり，入出力プロセッサを分離するといった小規模の機能分散が少しずつ行われている．また高級言語マシンを専用端末と考え，ネットワークを組んで全体として TSS システムと同じような機能を実現するという方向も研究されている．

## 参考文献


**高級言語マシン関係**

1) P. Abrams.: An APL machine, SLAC Report No. 114. Stanford Univ. Stanford, California (1970)
2) J. P. Anderson: A Computer for direct execution of algorithmic languages, Proc. EJCC 1961, pp. 184～193 (1961)
3) R.S. Barton: A new approach to the functional design of a digital computer, Proc. WJCC (1961)
4) T. R. Bashkow, A. Sasson and A. Kronfeld: System design of a FORTRAN machine, IEEE Trans. Electron. Comput. pp. 485～499 (Aug. 1967)
5) T. R. Bashkow, D. Kroft and A. Sasson: Study of a computer for direct execution on a list processing language, AFCRL-68-0063. Columbia University, New York (1968)
6) S. C. Baunach: Graphics in a language directed machine, COMPCON, (1976)
7) H. M. Bloom: Structure of a direct high-level-language processor, Proc. Symp. High-level Language Computer Architecture. pp. 70～80 (Nov. 1973)
8) Burroughs Corporation: Burroughs B 2500 and B 3500 Systems Reference Manual (1969)
9) R. J. Chevance: A COBOL machine, Proc. ACM Interface Meeting Programming Languages-Microprogramming, pp. 139～144. Harriman, New York (Jun. 1973)
10) Y. Chu: Interactive high-level Language direct-execution microprocessor system, IEEE Trans. SE-2, No. 2 (1976)
11) Y. Chu: High level language computer architecture, Academic Press (1975)
12) L. P. Deutch: A Lisp machine with very compact programs, Proc. of 3rd IJCAI (1975)
13) R. Greenblatt: "The LISP machine", MIT AI Lab. working paper 79 (1974)
14) A. Hassit, J. W. Lageschulte, and L. E. Lyon: Implementation of a high-level language machine, Commun. ACM 16, pp. 199～212 (1973)
15) E. A. Hauck and B. A. Dent: Burroughs' B 6500/7500 stack mechanism, SJCC (1968)
16) L. S. Haynes: Structure of a Polish string language for an ALGOL 60 language processor, Proc. Symp. on High-Level-language Computer Architecture, pp. 131～140. Univ. Maryland (Nov. 1973)
17) D. Hodges: IPL-VC, a computer system having the IPL-V instruction set, ANL-6888. Argonne Natl. Lab., Appl. Math. Div Argonne, Illinois (1964)
18) J. K. Illiffe: Basic machine principles, MacDonald, London (1968)
19) T. Knight: CONS, MIT AI Lab. working paper 80 (1974)
20) H. W. Lawson, et al.: A PL/I machine implementation, EE Dept. Polytechnique Institute of Brooklyn (1970)
21) H. W. Lawson and B. K. Smith: Functional charicteristics of a multilingual processor, IEEE Trans. C-20, No. 7 (1972)
22) W. Lonergan and P. King: Design of the B 5000 system, Datamation 7, pp. 28～32 (1961)
23) C. McFarland: A language-oriented computer design, Proc. AFIPS FJCC, pp, 629～640 (1970)
24) J.E. Meggitt: A character computer for high-level language interpretation, IBM Syst. J. 3, pp. 68～78 (1964)
25) A. J. Melbourne and J.M. Pugmire: A small computer for the direct processing of FORTRAN statements, Comput. J. pp. 24～27 (Apr. 1965)
26) Micro Computer Machines, Inc.: Portable microcomputers with APL Language offers large computing power, Computer Design (Nov. 1973)
27) A. P. Mullery, R. F. Schauer and R. Rice: ADAM-A problem-oriented symbol processor, Proc. AFIPS SJCC, pp. 367～380 (1963)
28) S. M. Nissen and S. J. Wallach: An APL microprogramming structure, Proc. Sixth Annual Workshop on Microprogramming, pp. 50～57. Univ. Maryland (Sept. 1973)
29) H. Park: FORTRAN Enhancement, Proc. Sixth Annual Workshop on Microprogramming pp. 156～159 (Sept. 1973)
30) R. Rice and W. R. Smith: SYMBOL-A major departure from classic software dominated von Neumann computing systems, Proc. SJCC, pp. 575～587 (1971)
31) S. C. Schroeder and L. E. Vaughn: A high order language optimal execution processor, Proc. Symp. High-Level Language Computer Architecture, pp. 109～116. Univ. Maryland (Nov. 1973)
32) M. D. Shapiro: A SNOBOL machine: a higher-level language processor in a conventional hardware framework, COMPCON 41～44 (1972)
33) 島田, 山口, 坂村: LISP マシンとその評価, 信学誌, Vol. 59-D, No. 6, pp. 406～413 (1976)
34) P. A. Smith et al.: Personal language dire-

cted systems, theory and practices, COMPCON (1976)
35) M. Sugimoto: PL/l reducer and direct processor, Proc. ACM pp. 519～538 (1696)
36) K. J. Thurber and J. W. Myrna: System design of a cellular APL computer, IEEE Trans. Computers C-19, pp. 291～303 (1970)
37) 山本他: 高級言語計算機の一実験 (I), (II) 電子通信学会電子計算機研究会資料 (1974)
38) B. W. Wade and V. B. Schneider: A general-purpose high-level language machine for minicomputers, Proc. ACM Interface Meeting on Programming Languages-Microprgramming, pp. 169～171. Harriman, New York (Jun. 1973)
39) Wang Lab. Inc.,: Wang 2200 data sheet (1974)
40) H. Weber: A microprogrammed implementation of EULER on IBM System 360 Model 30, Commun. ACM 10, pp. 549～558 (1967)
41) R. L. Wigington: A machine organization for a general purpose list pocessor, IEEE Trans. Electron. Comput. pp. 707～714 (1963)
42) W. T. Wilner: Design of the Burroughs B 1700, Proc. AFIPS FJCC, pp. 489～497 (1972)
43) N. Wirth and H. Weber: EULER: a generalizatiion of ALGOL, and its formal definition, Commun. ACM 9, pp. 13～25, 89～99 (1966)
44) D. B. Wortman: A study of language directed computer design, Tech. Rep. CSRG-20. Computer Syst. Res. Group, Univ. Toronto (1972)
45) R. Zaks, D. Steingart and J. Moore: A firmware APL time-sharing system, Proc. AFIPS SJCC 38, 179～190 (1971)

**中間言語関係**

46) 有沢: 中間言語方式のコンパイラと中間言語マシンについて, 電子通信学会電子計算機研究会資料 (1974)
47) F. R. Broca: Direct microprogrammed execution of the intermediate text from high-level-language compiler, ACM Sigplan Notices (1973)
48) L. W. Hoevel: Ideal, directly executed languages: an analytical argument for emulation IEEE Trans. C-23 (1974)

**その他**

49) S. H. Fuller, et al.: The effects of emerging technology and emulation requirements on microprogramming, IEEE Trans. C-25, No. 10 (1976)
50) E. A. Feustel: On the advantages of tagged architecture, IEEE Trans. C-22, No. 7 (1973)
51) J. K. Illiffe: 新世代計算機システム, 日本電子工業振興会. (1975)
52) 飯塚, 古谷: マイクロプロセッサアーキテクチャの一設計, 信学誌, Vol. J 59-D, No. 3 (1976)
53) T. Kilburn, A system design proposal, Proc. IFIP Cong. (1963)
54) G. E. Rossmann and L. H. Jones: Functional memory-based dynamic microprocessor, ACM Proc. SIGPLAN-SIGMICRO interface meeting. (1973)

(昭和51年12月13日受付)
(昭和52年1月22日再受付)