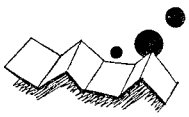
解　説



基本ソフトウェアの記述ツール†

井田昌之†† 中田育男†††

1. はじめに

ソフトウェアの生産性・保守性に対する認識が高まり，そのためのツールに対する議論も大きくとりあげられるようになってきた．エディタその他の軽装のツールから始まって，モジュール化・良形構造を志向した言語，更に全体的な生産性向上を目指す開発支援システムなどの重装備なものまで，さまざまなものが提案され，利用されている．

オペレーティングシステムその他の基本ソフトウェアは，一般の使用者から見ればハードウェアと同様に考えられ，使用者にとっての計算機システムの性能はそれらを含めて考慮されてしまう．特にメインフレームの汎用機の場合にこのことが言える．このため，応用ソフトウェアの場合と比べて，ハードウェア性能を生かした良い実行効率をより一層求められる場合が多い．また，近年急激な普及をみせている 8 bit マイクロプロセッサ上の基本ソフトウェアの場合，機械の小規模性から他とは異なった制約も存在する．総じていうならば，その基本ソフトウェアが置かれたハードウェア環境ならびに開発目的・開発環境の違いは大きく，統一的な扱いをすることは困難である．

例えば，次のように大別できるかもしれない．

1）研究・実験を主眼としたシステム．

2）商用システム．

あるいは対象の別により，

1）不特定多数に公開される汎用システム．

2）限定された使用者のための，あるいは使用者のない制御用などの専用システム．

あるいはそのソフトウェアのライフサイクルの観点から，

1）短命だが実行性能の良さが要求されるシステム

2）寿命がある程度長いことが予想され，保守性に重点を置くシステム．

ここではそうした多様性をもつ基本ソフトウェアの作成に用いられるツール，特に記述言語にしぼって解説を行うこととする．

2. 記述ツールのゴールとその選択

基本ソフトウェアの作成にあたっての記述ツール利用の目的及び期待効果を列挙してみると次のようなものがあげられる．

（1） 生産性向上，書き易さの向上．

（2） 保守性向上，プログラムの読み易さの向上．

（3） 信頼性向上（本来なら「正当性の保証」であろうか），“虫”の減少化．

（4） 管理性向上，Modularity・Clearity の向上．

これらを満たすツールとして高級言語を採用しようとする動向が一般に認められる．ただ，そのツールを利用して作成する基本ソフトウェアの実行性能をいたずらに損うことがあってはならない．また，その逆に実行性能に偏り，全体的な開発・維持性能を損うことも好ましくない．

この Trade off があるので，開発にあたっての記述ツールの選択に際しては，その基本ソフトウェアの置かれた環境や政策（ポリシー）にさまざまな観点があることに注意する必要がある．次にこうした点から見て性格の違いに触れてみたい．

2.1 新アーキテクチャ上の基本ソフトウェア

新アーキテクチャの場合，ハードウェアの開発・拡張が並行して行われることも多い．また作動して日が浅くハードウェアが不安定であるとか，納期が短いなど，諸々の制約を課せられている場合が多い．

新アーキテクチャに基づくマシンの開発に伴う諸欠点は，旧型機との上位互換性を考慮した設計によりある程度補うこともできよう．こうした場合は次の **2.2** のケースとなろう．

記述ツールの開発の余裕がない場合や記述ツール開

† Tools to write basic softwares by Masayuki IDA (Faculty of Science and Engineering, Aoyama Gakuin University) and Ikuo NAKATA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 青山学院大学理工学部
††† 筑波大学電子情報工学系

発の効果に気がつかない場合にはアセンブリ言語が用いられることも多い.

この場合,アセンブリ言語を用いて0次システムを作成し,その上で高級言語により再記述を行う方針をとったり,アセンブラ開発の工数と比べてそれ程ふえない工数と見積られる軽装の言語構造を設計し,その記述ツールの開発から始める方針をとる例もみられる.

例えば UNIX システム[1),2)] はアセンブリ言語により初版を記述し,その上でC言語[3)~5)]の開発及びCによるシステムの再記述を行っている.

また,開発初期段階においてアーキテクチャの設計とシステムプログラムの開発との間の交流が行われる場合,システムプログラムの要求にアーキテクチャをあわせることもみられる.

その大規模な例としてはバロース社の計算機をあげることができる. B 5000 の開発は当初から Algol を意識して設計されている. その OS (MCP) の初版はアセンブリ言語で記述されたが,その後 Algol に書き直したら速くなったという話も,雑誌にではあるが紹介されている[6)].

また,初期開発の苦労は多々報告されている.

例えば,Multics システムは PL/I でほとんどが書かれているが,その初期には PL/I の開発とハード・OS の開発が並行して行われたため相当の苦労があったようである. メーカ製の本格的な PL/I[7)] がサポートされるまでは EPL (Early PL/I) が利用された. EPL はつなぎとして TMG 言語により開発された. しかし,5年間にわたって利用されることになり,EPL には最適化処理フェイズの組込みや機能拡張がシステムの進展と共に持ち込まれ,担当者達はくたくたになったという.そして,こうした状況をふりかえって,もう1度作りなおせるならもっと軽装の言語を作り,機能の豊富さのためにはサブルーチンライブラリの整備をすることをしたかったという感想で文献8はしめくくられている. (但し,これは初期の全開発にタッチした者の感想であって,PL/I による Multics の開発が失敗したという意味ではないようである. 最近になってまとめられた文献9ではそれらについてもふれられている.)

2.2 既存アーキテクチャ上の基本ソフトウェア

蓄積されたソフトウェアが多少なりともある場合には当然それらを利用することができる.

第一に,ルーチン (off-the-shelf components) の流用があげられる. 後から利用できる形で各種の機能を整理し作成することの効果は大きい.

第二に,既開発の高級言語を利用することがあげられる. 使用者による言語 (またはプリプロセッサ) やアプリケーションパッケージのメインフレーム上での開発などは,ほとんどこの範囲にはいる.

また,商用機の基本ソフトウェアの開発に際して,汎用の高級言語の利用も行われている. PL/I による Multics の開発も初期以後はこの分類にはいろう.

第三に,専用のシステム記述言語の開発とそれによるソフトウェア作成があげられる. システム記述には PL/I "系" の言語がよく見られる. その理由は他の言語に比べて,ビット,アドレス,構造データなどのデータ処理機能やメモリ管理その他に秀でている最初の標準的な言語であるからのようである. しかし,フルセットでは機能が大きすぎる点や,PL/I の構文則では書ききれないことがある,などから PL/I サブセット系がよく使われている. それらの言語仕様は多種多様であるが,システム記述のための共通項はいくつか存在するようである[10),11)]. ただ,PL/I に絶対的な支持があることではないようで,例えば PL/I サブセット系である BPL に関する文献[12)]でも PL/I の問題点は指摘されている.

2.3 ポータビリティを重視した基本ソフトウェア

普及性の点からか,Fortran は,portability を主眼としたコンパイラなどの記述によく用いられている. Fortran の扱う世界は基本的には語構成である. また,変数は静的割付けに基づいて処理され,簡明であるといえる. この点も Fortran の利用をふやす一因となっている. マイクロコンピュータ用のクロスソフトウェアなどをはじめ,多くの実例がある. ANS 77[13)] により Fortran の機能は整備されてきており,使い易さも格段に進んでいる. ただ,その仕様がすべてうけ入れられ定着するまでには若干の日々が必要かもしれない.

また,この数年注目されはじめ,実際にそれを用いて開発が行われてきた2つの言語系統がある. BCPL と Pascal である. BCPL[14)] 系の言語は,強力な制御構造の記述力を持ち,簡潔な関数形式により記述を行う. 演算子の豊富さを特徴としたタイプレス言語 (変数に型はなく,使用する演算子によってのみ動的に型が決定される言語) である. BCPL 及びその派生である B[15),16)], C[3)~5)], Eh[17)] などによりいくつかのシステムが作られている.

BCPL 系の場合,処理系のポータビリティはあるが

機械記述性の高さも特徴の1つであることもあり，作成されたプログラムの移植性は大きくはない．

一方の Pascal[18),19)] は N. Wirth により設計された言語で，使用者定義を含む豊富なデータタイプなどに実際的な特徴があり，さまざまな応用に使われつつある．また ANSI では Pascal の標準を作ろうという動きもある．

Pascal コンパイラ自身も Pascal で書かれている．Pコードと呼ばれる抽象機械の命令セットを用いて書かれた Pascal-P[20)] をはじめとしていくつかの版があり，それらを用いた移植については興味深いものがある（文献21などには Pascal 及び BCPL の移植過程についての解説がある）．

OS 等に対するポータブルな記述は現在のところではいくつかの例をみるだけであるが，複雑化する OS のモジュラー設計や処理概念の整理・抽象化などの観点からも今後が注目される．

以上に述べたように各種の状況が考えられ，極端な場合，全く正反対のゴールが要求されもする．

例えば Horning のあげたコンパイラ開発のゴール[22)]にはポータビリティははいっていない．また利用すべき言語に対しても機械依存的な意見を述べている．一方 Hansen の考え方の中ではポータビリティは重視されている[23)]．

次章では記述ツールの例を順にとりあげていくことにしよう．

3. 基本ソフトウェア記述ツールの例

3.1 アセンブラ主体の記述ツール

全体的に保守性・生産性を見て，高級言語を用いて記述しようとする動向が高まっているのは事実であるが，高い実行効率が要求される場合にはどうしてもアセンブリ言語に依存することが大型機においても現在なお行われている．また，Dijkstra をして「計算機をとりまく状況を25年前に戻してしまった」と言わしめたというマイクロプロセッサの普及により，依然としてアセンブリ言語しか提供されない，あるいは提供できない計算機の占める数は減っていない．

こうした裸の計算機の場合のアセンブラ主体のツールとしては，通常のセルフアセンブラに加えてクロスソフトウェアや汎用アセンブラが利用される場合もある．クロスソフトは確定したアーキテクチャを持ち，市場性のあるマイクロプロセッサの場合に多く用いられている．汎用アセンブラは未確定のアーキテクチャのためなどに利用できる．

アセンブリ言語によるコーディングの場合，当然のことながら機械記述性は高いので，実行効率に関する目標は達成しやすいが，反面1ステートメントあたりの記述力は小さく生産性が低くなる欠点がある．また生のアセンブリコーディングやトリック的なコードを記述してしまう場合，特に保守性が悪くなる問題点がある．

このために単純なアセンブラ以外のなんのツールも持たない計算機の場合，注釈行による説明付記の強制や pseudo coding などが行われる．更にリロケータブルアセンブラの場合にはルーチンごとの独立アセンブルなどによる管理も利用される．

また，マクロアセンブラを用いる場合には当然のことながらマクロ定義を利用した生産性向上がはかられている．DO-WHILE マクロ，CASE マクロなどの structured coding 用のマクロはその一般的な例である．

筆者の ALPS/I システム[24)]ではハードウェア開発と並行してソフトウェア及びその記述ツール（クロスアセンブラ及びシミュレータ）が開発された．その際，対象となった Lisp の特殊性（独立した小さなモジュールの集まりとして作成しやすい．）もあるが，約20のマクロ定義と注釈行中の M 式[25)]による pseudo coding などを利用して短期間に1万ステップ近いシステムプログラムを作成している[26)]．

一方，強力なマクロプロセッサを持つアセンブリ言語の場合，それを利用してかなり保守性の高いモジュール化を達成できる．

例えば IBM 社のアセンブラマクロ[27)]では，

① マクロ引数に対する型・長さ・文字数その他の属性のチェックとそれを利用した条件付生成．
② マクロ定義間の情報の受渡し．
③ 文字列の生成・連結，部分文字列の取り出し．
④ 整数・文字などの型をもった展開用ローカル変数及び配列．
⑤ ライブラリ化機能．

その他が用意されている．筆者の大学で利用している WATFIV[28)] はこれらを利用して書かれている，WATFIV は約200のモジュールから成っているがそれらはすべてマクロ定義として作成される．上記に示した機能を利用し，モジュール間のインタフェースも（コンパイラの）アセンブル時にチェックされる．マクロ定義でないコーディングは非常に小さく，基本的には

A. MAIN というマクロの引用のみである．A. MAIN の中で他部分がマクロコールあるいは copy によって次々と引用・結合される．

また，P. C. Poole の分類[29]によれば，Pascal-P や BCPL と同様に抽象機械の中に入れられているが，形態的にはマクロ機能の高度利用により作成維持されている言語に DOS-Fortran IV[30] 及び OS-Fortran (G) があげられる．これらのコンパイラは，抽象機械 POP の命令セットを用いてコーディングされている．POP は 63 種のデータ (roll) を維持管理するスタック領域 (roll area) と work と呼ばれる作業用スタック及び exit と呼ばれる制御スタックに対して動作する．POP 命令は 2 バイトを占め，OP コード 1 バイトと即値または相対ポインタを入れるオペランド 1 バイトにより構成されている．POP 命令は次の命令群にわけられる．

1）転送命令 (42 種)，2）演算命令 (12 種)，3）判定命令 (20 種)，4）分岐命令 (10 種)，5）ロール制御命令 (3 種)，6）コード生成命令 (3 種)，7）アドレス計算命令 (5 種)，8）間接アドレス命令 (1 種) 計 96 種．

この POP 命令を用いて記述されたコンパイラは，アセンブリ言語で書かれたインタプリタにより解釈実行されることによりコンパイルを行う．(部分的なアセンブリコーディングの埋込みはされている．)

この抽象マシンの使用は，Pascal-P 等の portability を主眼としたものとは異なり，処理系の実行環境，特に所要主記憶を小さくしながらかつ保守性を向上させることに主眼があるようである．

3.2 高級言語による記述ツール

高級言語を記述ツールとして採用することにより，生産性・保守性・信頼性・管理性などの達成に一歩近づくことができる．その反面実行効率や機械記述性の低下が生じる．更に，プログラマの力量による品質の変化，その高級言語自身の保守性・信頼性などの問題を考慮して利用する必要がある．多くの場合，記述ツールとして利用する言語の設計思想を理解した上で用いると良い結果が得られるようである．

生産性の向上を定量的に測るのは大変難しい問題であるが，非常に大ざっぱに言えば，プログラマが 1 日に生産できるプログラム平均行数は使用言語によらずほぼ一定であると言われている．高級言語の 1 行がアセンブリ言語の n 行にあたるとすれば，高級言語使用による生産性向上は n 倍ということになる．それでは n を大きくすればよいということになりそうだが，それは出来上るソフトウェアの性能低下につながる．商用機の基本ソフトウェアの記述においてはこの性能低下の防止には特に重点をおかねばならない．そうすると n は小さくなる．例えば PL/I の場合 n は 5 前後といわれているが，PL/I サブセット系ではそれより小さく n は 2 前後であるようだ．

実行効率の低下に関しては高級言語を擁護する側からの反論がある．プログラムの性能は設計の良し悪しできまる．高級言語の方が再設計は容易であり，良いものが出来る可能性があるというものである．

欠点をカバーするための対策はいろいろ考えられており，次のように分類できよう．

（1） 最適化コンパイラを開発する．最適化処理系の作成により目的プログラムの効率は上がるが，処理系開発工数や最適化処理系を動かすために，より多くの主記憶と時間が必要となり，新たな問題が生じる．

（2） 最適化支援の機能を言語に付加する．例えば，ある変数に特定のレジスタの割当てを指示するとか，データフロー解析を助けるために変数の使われ方を表明する，などである．

（3） 言語の機能を単純にし，効率の悪い目的プログラムを作り出すような機能は削る．これにより，多くの欠点を解決することができ，コンパクトに作ることもできる．反面文法上の機能は小さくなることがいえる．このためライブラリプログラムパッケージを用意し補う場合がある．

（4） アセンブラ機能の利用．アセンブリ言語の命令を途中に挿入できるようにするとか，目的プログラムをアセンブリ言語プログラムの形で出力し，それを人手で修正できるようにする．

などが考えられる．しかしアセンブリ言語で書かれた部分はコンパイラではほとんどチェックできないし，その部分が多くなるのは好ましくないので，

（5） アセンブリ言語でよく使われる機能は高級言語に組み込む．例えばシフト命令やスーパーバイザコール命令を関数呼出しやステートメントの形で書けるようにする．前者は言語の文法上の機能にはあらわれないが，後者はその言語の文法上の機能を拡張することになる．

汎用言語に機械依存の機能を付加することが現実にはよく行われるが，そうした場合にそれらは高級言語記述の利点を抑制する方向に働く場合があるので注意を要する．

3.2.1 Fortran による例

最近の大型機でのコンパイラの最適化技術は進んでいるが，その先駆けとなったのは Fortran-H コンパイラである．このコンパイラの開発者達は，充分な最適化がされるので Fortran 自身を記述ツールとして使っても性能上の問題はないと考えた．そしてその初期開発は /360 ではなく，7094 上で行った．その後3回のブートストラップにより所与の機能を果させている[31]．なお，第1回は 7094 から /360 への移植．第2回は /360 上での機能（ビット処理等）の追加と最適化．（これにより所要メモリを 550 k から 400 k に減らした．）第3回のブートストラップ（セルフコンパイル）は，コンパイルタイムの削減（約 35%）と所要メモリの減少を目指した最適化がなされた．この結果，256 k で約 700 ステートメントをコンパイルできるようになった．

この開発はほとんど Fortran で行われたが，やはり性能と記述性の面から，ビットや文字の処理，コード生成の一部などにアセンブリ言語が一割程度利用されたらしい．

3.2.2 Algol による例

バロース社の OS は MCP (Master Control Program) と呼ばれる．MCP の開発には Algol が用いられている．文献6その他に，B 5000 の開発の当初より，既に Algol を意識したハードウェア構造と，記述言語として Algol を採用した経緯がふれられている．B 5000 にはアセンブラは存在せず，MCP の初版は手作業で機械語におとされ作成された．それを用いて Algol コンパイラが作られ，MCP の Algol 版が作られたという．

B 6500/6700 の MCP の開発に用いられた拡張 Algol は，Algol 60 にビット処理，ストリング処理，リスト処理，イベント処理，非同期処理等の機能が追加されている．また機械へのアクセス機能を持つ同族の言語 ESPOL により，MCP の機械依存部分は記述されているという．

3.2.3 PL/I による例

PL/I を大規模なシステムの記述に使った好例は Multics である．MIT を中心に初期には Bell 研，GE 社，そして現在は Honeywell 社の手により維持されている．プログラムの規模も，PL/I の行数で初期には 250 k ステップ，現在は 1M ステップにのぼっている．PL/I を使った利点として強調されているのは，研究開発の途中での再設計の容易さや保守性の高さである．生産性は平均 1,200 ステップ/人年 であったと報告されている．この数字は OS/360 開発時のアセンブリ言語ステップ数での生産性の数字に近い[32]．

また，Corbató によれば記述上の PL/I の強力な点は，

（i） 長い識別名がかける．
（ii） フル ASCII 文字がかける．
（iii） 構造とデータタイプ
（iv） ポインタ変数とベース付記憶
（v） SIGNAL と ON 条件
（vi） Algol から出ている条件文などの制御構造

であったという．反面，主な問題点の1つとしてプログラマの力量によりオブジェクト効率は 5 から 10 倍低下することをあげている．しかし再設計の容易さから，1か月以内に5万語を要した部分を1万語程度に減らしたり，5〜10 倍の速度向上を達成した部分も数カ所あると述べている[8]．あわせてこれらの効果をうるのに PL/I は初期的には高価についたことにもふれられている．

3.2.4 PL/I サブセット系言語の例

PL/I の中から基本ソフトウェアの記述に必要な機能をとり出し，更にいくつかの追加を行った言語が各種作られている．外国の例では XPL[33]，IBM の PL/S[34]，Philips の SPL[35] など．国内では，日電の BPL[12]，通研の SYSL[36]，日立の PL/IW[37]，東芝の TPL-40[38] などをあげることができる．これらの多くは，アセンブリ言語によるインライン記述，レジスタ指定その他の工夫ができるようになっている．PL/S は，1967 年頃から設計され，5年後程して OS の開発などに利用されたようである．

また，マイクロコンピュータ用に PL/M[39] が開発されているが，これには変数のレジスタ割付けなどの機能は付加されていない．またデータタイプも，BYTE（8ビットデータ）と ADDRESS（16ビットデータ）の2種しかない．機能的にはかなりの制限があるが，命令域と作業域との分離や，割込み処理手続きの指定など実用的な機構は含められている．

3.2.5 BCPL 系の例

BCPL 及びその派生である B, Eh はタイプレス言語である．一般に変数の宣言の中に型宣言を含むことは好ましいが基本ソフトウェアの場合，同一の場所（変数）に対して異なる属性でアクセスしたい場合が他に比べて多い．また対象とするハードウェアによってはコンパクトなコンパイラの提供が早期に望まれる

場合も多い．BCPL 系の言語はこうした場合に便利である．

BCPL 系の言語の特徴は，機能の豊富さは実行時ライブラリに求め言語構造から切り離している「軽さ」と，いくつかの制御構造と，豊富な演算子にあるだろう．例えばBのコンパイラ[16]は4k 語であるがライブラリは約 37k 語分が用意されている．演算子として用意された①論理演算子 (&&, ||, !,～), ②ビット演算子 (&, |,～), ③シフト演算子 (<<, >>) その他のものは，フル ASCII コードを充分に利用して設計されている．実例その他については例えば文献 40 にのせられている．

BCPL はオックスフォードにおける OS の記述やコンパイラの記述に，コンパイラ版 B は Honeywell マシン上での Pascal, Lisp, Snobol その他の開発に利用されている．また，Eh はポータブル OS である THOTH の記述に利用されている．

近年 BCPL 系の中で注目されている C 言語はベル研で開発されたもので，UNIX の記述に利用されているのは有名である．C は BCPL, B, Eh とは異なり Typed 言語である．C は上記の基本機能に加えて変数のレジスタ割付け指定や型宣言を備えている．また，構造体の記述もできるようになっている．文献 5 に記されている新版C には更にブロック構造言語志向が強められ，変数スコープの入れ子構造も導入されている．C や UNIX についてはいくつかの文献や実際的動向があり，注目される[2),51)]．

なお，BCPL を基にインタプリタ版 B[15] が作られ，それからコンパイラ版 B[16], C, Eh が独立に作られたようである．

3.2.6 Pascal 系の例

Pascal ないしは Pascal 系の言語を用いてシステムプログラムの記述を行おうという最近の傾向がある．Pascal 系ないしは Pascal の影響を受けた言語としては SUE[41], concurrent Pascal[23),42)], Euclid[43),44)], Modula[45),46)] などがあげられる．基本ソフトウェア専用ではないが，その動向が注目されている言語開発プロジェクトに米国国防総省 (DOD) の HOL がある[47]．HOL の仕様にはかなり Pascal の影響がはいると推測される．

また，P.B. Hansen は彼の設計した concurrent Pascal を用いて solo OS を作成している．solo のほとんどの部分 (92%) は通常の Pascal (Sequential Pascal) で記述されている．4% の部分は非同期タスクの扱いなどのできる concurrent Pascal で記述されている．そして核になる部分 (入出力記述他) はアセンブリ言語が利用されている．

Pascal の使用はふえているが，実際的な入出力記述やストリング処理に弱いなどの声も聞かれる．また前述した B で書かれた Pascal 以外の多くは自分自身の閉じた環境下でのオブジェクトを生成し，他の言語のように完全なリロケーブルオブジェクトを生成しない．このため他のオブジェクトとのリンクに制限があるものが多い．これらの事情からか，現時点では主に研究用の記述に用いられている．

3.2.7 その他の言語

Wirth の開発した PL/360 はアセンブリ言語と同等の機能を Algol 風に書けるようにした言語であり，中級言語とでも言うべきものである．PL-516[48] などにとりいれられている．また，Wulf の BLISS[49] その他多くの言語が発表されている．

4. おわりに

高級言語による基本ソフトウェアの記述は今後ますます広まるだろうが，現在の時点ではさまざまな観点と立場があり，時として正反対のゴールを要求される場合もあることなどもあわせて述べた．例えば機種依存性をなくすかどうか，高級言語化した時の開発効率と実行効率のトレードオフをどうするかなどは状況によって結論は異なろう．

Pascal 及び Pascal 的言語の動向については今後が期待され，興味深いものがある．例えば Los Alamos の CRAY-1 の OS を Pascal で書きはじめたという記事ものせられている[50]．しかし，ここでは実績のある処理系にしぼって紹介を行ったのでそれらについてはあえて省略した．またアセンブリ言語寄りのツールに対する解説は近年あまりみられないが，やはり重要であるので多少の頁をさいた．

基本ソフトウェアをとりまくツールのうち主に言語をとりあげたが，エディタ・各種ユーティリティその他の支援プログラムも完備してはじめて良い開発環境となることはいうまでもない．また，今後をみるならば設計段階も含めた総合的なシステムが必要となるであろう．更に抽象データ型言語などの，現在進められているプログラミング言語についての研究の成果をとり入れることができれば，将来の基本ソフトウェアの作成は現在のそれと比べて格段に進歩することができよう．

参考文献

1) Ritchie, D. M. and Ken Thompson : The UNIX Time Sharing System, CACM, Vol. 17, No. 7, pp. 365-375 (1974).
2) 石田晴久：ベル研究所の軽装 OS-UNIX, 情報処理, Vol. 18, No. 9, pp. 942-949 (1977).
3) Ritchie, D. M. : C Reference Manual, Bell Labs. (Jan. 1974).
4) Kernighan, B. W. : Programming in C-A Tutorial, Bell Labs. (May 1974).
5) Kernighan, B. W. : The C Programming Language, Prentice hall (1978).
6) Bulman, D. M. : Stack Computers, IEEE Computer, pp. 14-16 (May 1977). (邦訳は，井田訳「スタック計算機入門」, bit, Vol. 11, No. 5, 1979).
7) Freiburghouse, R. : The Multics PL/I Compiler, Proc. AFIPS, FJCC pp. 187-199 (1969).
8) Corbató, F. J. : PL/I as a Tool for System Programming, Datamation, pp. 68, 73-76 (May 1969).
9) Van Vleck, T. H. et al. : The Multics System Programming Process, Proc. 3rd int. conf. Software Engineering, pp. 278-280 (May 1978).
10) 筑後道夫：システム記述用言語, 情報処理, Vol. 16, No. 10, pp. 864-870 (1975).
11) 中田育男：システム記述言語の最近の傾向, ソフトウェア工学シンポジウム報告集, pp. 65-74 (1979).
12) 小久保晴世他：コンパイラ記述言語 BPL, 情報処理, Vol. 11, No. 6, pp. 342-349 (1970).
13) ANSI X3J3: ANS FORTRAN 77 unofficial Document X3J3/90.5 (1978-6-1).
14) Richards, M. : BCPL : A Tool for Compiler Writing and System Programming, proc. SJCC 1969 pp. 557-566 (1969).
15) Johnson, S. C. and Kernighan, B. W. : The Programming Language B, Bell Labs. (1972).
16) Gurd, R. P. : User's Reference to B for Honey well series 6000/66, Univ. of Waterloo. (Jan. 1977).
17) Reinaldo Braga, S. C. : Eh Reference Manual Univ. of Waterloo (1977).
18) Wirth, N. : The Programming Language Pascal, Acta Informatica Vol. No. 1. pp. 35-63 (1971).
19) Jensen, K. et al. : Pascal User Manual and Report 2nd edition, Springer-Verlag (1978).
20) Nori, K. V., Anmann, U. et al. : The Pascal 〈p〉 Compiler : Implementation Notes, Berichte des Instituts für Informatik, Eigenössische Technische Hochschale Zürich.
21) 和田英一：ソフトウェア工学とプログラム言語, 情報処理, Vol. 16, No. 10, pp. 848-855 (1975).
22) Horning, J. J. : Structuring Compiler Development, in Lecture Notes in Computer Science 21, pp. 498-513, Springer-Verlag (1976).
23) Hansen, P. B. : The Architecture of Concurrent Programs, Prentice-hall (1977).
24) 井田昌之，間野浩太郎：マイクロプロセッサを用いた Lisp マシン ALPS/I, 情報処理, Vol. 20, No. 2, pp. 113-121 (March 1979).
25) McCarthy, J. et al. : Lisp 1.5 Programmers Manual, MIT Press (1966).
26) 井田昌之：Lisp マシン製作奮戦記, bit, Vol. 10, No. 14, No. 15, (1978).
27) IBM : Assembly Language Manual
28) WATFIV User's Manual, Univ. of Waterloo (1975).
29) Poole, P. C. : Portable and Adaptable Compilers, in Lecture Notes in Computer Science 21, pp. 427-497, Springer-Verlag (1976).
30) IBM : /360 DOS Fortran IV Program Logic Manual, No. GY 28-6394-2 (1975).
31) Lawry, E. and Medlock, C. : Object Code Optimization, CACM, Vol. 12, No. 1, pp. 13-22 (1969).
32) Brooks, F. F. : The Mythical Man-Month, Addison Wesley (1975).
33) McKeeman, W. M., Horning, J. J. : A Compiler Generator, Prentice-hall (1970).
34) Brittenham, W. R. et al. : The System Programming Language Problem, in Machine Oriented Higher level language, v.d. Poel/Maarssen eds., North-Holland, pp. 29-47 (1974).
35) Klunder, J. : Experiences with SPL, ibid, pp. 385-396 (1974).
36) 寺島信義他：システム製造用言語SYSL-2の設計, 情報処理, Vol. 16, No. 8, pp. 692-697 (1975).
37) 中田育男：HITAC 5020 TSS の記述言語としての PL/I サブセット, 昭和 43 年連合大会 2529 (1968).
38) 大箸 豊他：ミニコン用 PL/I サブセット TPL-40, 情報処理, Vol. 19, No. 5, pp. 406-411 (1978).
39) Intel Japan : PL/M-80 プログラミングマニュアル (1977).
40) 井田昌之：システム記述言語の記述性, ソフトウェア工学シンポジウム報告集, pp. 75-86 (1979).
41) Clark, B. L. et al. : The System Language for Project SUE, SIGPLAN Notices Vol. 6, No. 9, pp. 79-88 (1971).
42) Hansen, P. B. : The Programming Language Concurrent Pascal, IEEE Trans. on Software Eng., Vol. 1, No. 2, pp. 197-207 (1975).
43) Lampson, B. W. et al. : Report on the Programming Language Euclid, SIGPLAN Notices Vol. 12, No. 2 (1977).
44) Aseltine, E. G. et al. : Isolation of Machine Dependences in Euclid, SIGPLAN Notices, Vol. 13, No. 3, pp. 43-48 (1978).

45) Wirth, N.: Modula: A Language for Modular Multiprogramming, Software-Practice and Experience, Vol. 7, pp. 3-35 (1977).
46) Wirth, N.: Design and Implementation of Modula, ibid, pp. 67-84 (1977).
47) 上条史彦: プログラミング言語への期待——米国国防省の HOL プロジェクトについて——, 情報処理 Vol. 19, No. 3, pp. 266-274 (1978).
48) Wichmann, B.A. et al.: An Appraisal of a High-level Assembly language, in Machine Oriented Higher level Languages, North-holland, pp. 377-381 (1974).
49) Wulf, W.A. et al.: BLISS: A Language for System Programming, CACM, Vol. 14, No. 12, pp. 780-790 (1971).
50) Sites, R.L.: Programming Tools: Statement Counts and Proceduce Timings, SIGPLAN Notices, Vol. 13, No. 12, pp. 98-101 (1978).
51) The Bell Sys. Tech Journal Vol. 57, No. 6, UNIX 特集号. (1978).

(昭和54年3月23日受付)