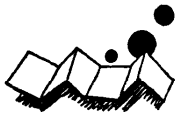
解　説

マイクロコンピュータのソフトウェア開発技法およびツールの動向†



加　藤　肇　彦††

1. はじめに

マイクロコンピュータ（以下，マイクロコン）が出現して以来の15年間に，そのソフトウェア開発技法（以下，技法）及びソフトウェア開発ツール（以下，ツール）は，多くの試行錯誤をともないながら**図-1**に示すように発展してきた[1]．これらの各技法とツールを開発工程の上流から下流に位置づけると**図-2**に示すようになり，歴史的には下流から上流方向へ技法とツールが遡上的に発展してきたことが伺える[2),3)]．

マイクロコンもコンピュータの一種である以上，そのソフトウェアは大型コンピュータとの共通点も多い．しかしながらマイクロコンのソフトウェアを，特に大型コンピュータの場合と対比した特徴として，

（1） 機器組み込み制御への応用が多く，その結果機構部や電子回路とソフトウェアの間でコストとリアルタイム応答性を考慮した**トレードオフ**の技術が必要なこと．

（2） これら三者の間で同期や競合回避などの**協調動作**が必要なこと．

（3） 専用化とコスト制限のため**プログラム**はROM化されて機器に実装されることが多く，このためにはコード圧縮やROM化可能コードの生成，オペレーティング・システム（以下，OS）を含めた**最適設計**が必要なこと．

（4） ROM化されて不特定多数のユーザの手にわたるプログラムは品質に対する要求がきわめて厳しいため，強力なデバッグツールが必要である．その反面，プログラムが組み込まれている機器のコスト制限が厳しいため，プログラムの開発時にのみ必要で完成後に不要になる諸機能はツールとして分離されていること，換言すればプログラムの**開発環境と実行環境が明確に分離**されていること．

† Trends of Microcomputer Software Development Techniques and Tools by Hatsuhiko KATO (Yokohama Works, Hitachi Ltd.)

†† (株)日立製作所横浜工場

があげられる[4)]．

本記事ではこれらの特徴に注目して，特にマイクロコンに特徴的な技法とツールを上流から下流に向けて紹介し，それらの動向を考察する．また，上流工程に関しては，まだ技法として確立していないが，ソフトウェア開発に必要なアプローチや作法と呼ぶべき項目も，広義の技法に含めて論じる．

2. ソフトウェア開発技法及びツール

2.1 ソフトウェア設計の技法とツール

本節では第1章で述べたマイクロコン用の技法とツールの特質を踏まえ，その代表例を紹介する．

（1） 機器組み込み制御のための設計技法[5),6)]

機器組み込み制御へのマイクロコンの応用は，**シーケンス制御**が代表的である．また，表面的に意識されない場合でも，大半の応用はシーケンス制御を内包する．シーケンス制御のソフトウェア設計技法として，**状態遷移図**による仕様記述より出発して，プログラム構造へ展開していく方法が提唱されている．状態遷移図は視覚性に優れており，システムの動作を直観的に把握するのに適している反面，あいまい性をもっている．この欠点を補うために，システムの動作を**表-1**に示すような**イベント／ステータス・マトリクス（状態遷移表）**に展開する．左端には状態を縦に並べ，上端には遷移条件を横に列挙してある．各欄内にはその状態と遷移条件の組み合わせに対する，遷移先状態とシステムの動作を記入してある．未定義の空欄はシステムが正常に動作している限り起こり得ない組み合わせに対応しており，各空欄を埋めることによってあいまい性を除去する必要がある．このためには**決定表**を利用する．決定表は状態ごとに1枚ずつ作成し，**表-2**は表-1の状態S0に対して作成した決定表である．左側の条件エントリにはその状態で注目すべき条件を列挙し，上側の条件スタブには各条件の生起/不生起の組み合わせを列挙する．そして行動エントリには行動を列挙し，条件スタブに対応する欄に○印を記入する．決定表はこのように設計工程での分析に利用でき

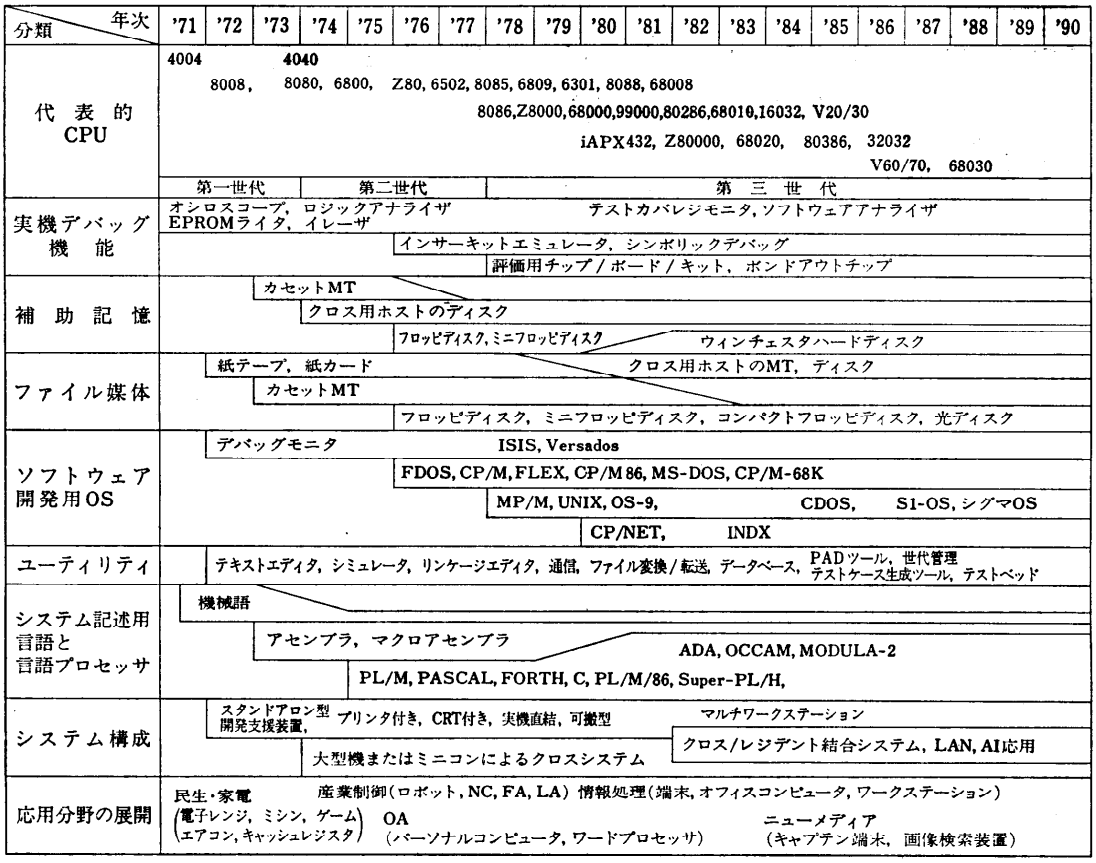


図-1　マイクロコンピュータ用ソフトウェア開発ツールの発展

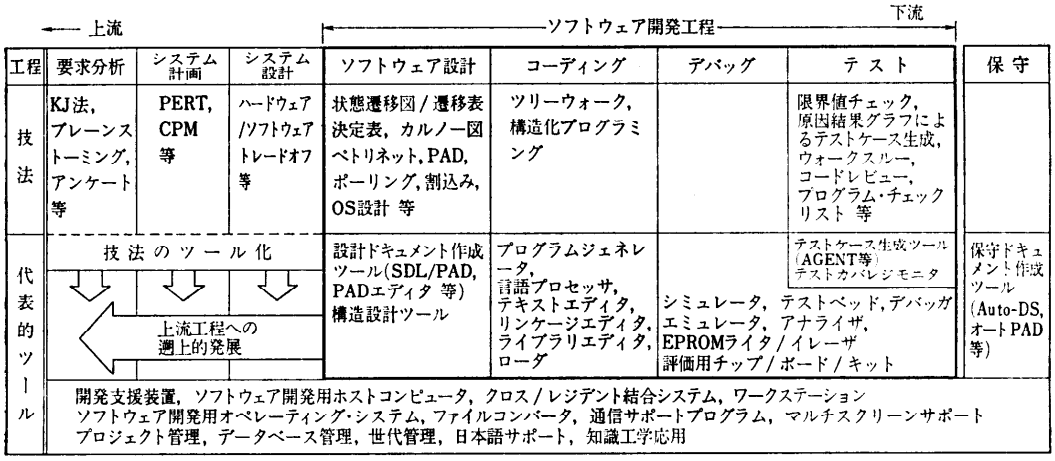


図-2　マイクロコンピュータのソフトウェア開発技法およびツールの位置づけ

るだけでなく，テスト工程で**プログラム・チェック・リスト**としても利用可能である．条件分岐の数を減らし，プログラムの構造を単純にするために，**図-3** の**カルノー図**によって決定表に示された条件を整理す

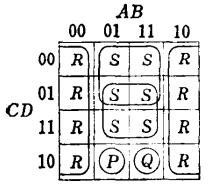
表-1 部品選別システムの動作を示すイベント/ステータスマトリクス（状態遷移表）の一部分

事象（イベント） / 状態（ステータス）	部品到着		h_A 到達 $h=h_A$		h_R 到達 $h=h_R$			θ_B 到達 $\theta=\theta_B$	
						良品	不良品		
	NO	YES	NO	YES	NO	YES		NO	YES
S0	S0 コンベヤA走行	S1 コンベヤA停止							
S1			S1 下降	S2 停止，チャック閉，コンベヤA始動					
S2					S2	良品 S3	不良品 S6		
					上昇判定	停止			
S3								S3 CCW旋回	S4 停止

表-2 事象の組み合せに対する行動を示すデシジョンテーブル（決定表）（ステータス S0 の場合を示す）

			条件スタブ Y: YES N: NO														
			f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
条件エントリ	A	部品到達	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
	B	h_R 到達	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N
	C	θ_A 到達	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
	D	その他の事象のいずれか一つ以上	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
行動エントリ	P	正常動作実行，コンベヤ走行，S0 へ										○						
	Q	正常動作実行，コンベヤ停止，S1 へ		○														
	R	停止，警報発生					○	○	○	○					○	○	○	○
	S	再確認後正常ならば P または Q，その他 R	○		○	○					○		○	○				

注）○印はとるべき行動を示す．



$P=\bar{A}BC\bar{D}$
$\bar{A}B(\bar{C}+D)$

$Q=ABC\bar{D}$
$=AB(\bar{C}+D)$

$R=\bar{B}$

$S=B\bar{C}+BD$
$=B(\bar{C}+D)$

図-3 各行動をとるべき条件を示すカルノー図

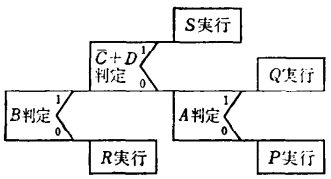


図-4 逐次的決定のプログラム構造

表-3 ハードウェア/ソフトウェア間のトレードオフの対象になる機能

機能	ハードウェアによる実現手段	ソフトウェアによる実現方法
キーボード入力 二重設定除去	ラッチ，プライオリティエンコーダ，カウンタ，マルチプレクサ	スキャンニング再読み取り
チャタリング除去	交叉結合 NAND 回路	多重読み取り
A/D 変換	A/D 変換器	D/A 変換器，比較器と逐次2分法アルゴリズム
符号変換，文字発生	符号変換素子，キャラクタジェネレータ	符号変換プログラム，テーブル
出力信号保持	ラッチ	マルチプレクシング
時間管理	タイマ素子	ソフトウェアタイマ
事象計数	カウンタ素子	カウントルーチン
演算処理	演算用コプロセッサ，数表 ROM	関数サブルーチン
割込み解析	優先割込み制御素子	割込み解析ルーチン
先着順処理	FIFO バッファ素子	エンドレスバッファ
パリティチェック	IC パリティジェネレータ/チェッカ	パリティ生成/検査プログラム

る．これによって最終的に得たプログラム構造は図-4のように単純になる．

（2） ハードウェアとソフトウェアのトレードオフ

マイクロコンの応用の特徴の一つは，その設計過程にハードウェアとソフトウェアのトレードオフを含むことである．多くの機能はハードウェア/ソフトウェアのいずれでも実現可能であり，ここにトレードオフの可能性がある．一般にハードウェアは高速であるがコストは高く，ソフトウェアはこの逆の性質を持っている．性能とコストの兼ね合いによって，ハードウェアとソフトウェアに機能を配分する．特にローエンド応用あるいは高性能を要求される応用の場合には，システム設計工程だけでなく，ソフトウェア設計工程に入ってからも，ハードウェアとのトレードオフの可能性を常に追求しなければならない．表-3 はハードウェアとソフトウェアの間のトレードオフの対象となる機能を示している．

（3） 同期と競合回避

マイクロコンの応用では，往々にして制御対象機器や複数の CPU が，共通のプログラムやデータを利用するために，同期や競合回避の協調動作をする必要が生じる．このような場合には状態遷移図を拡張した，**ペトリネット**によってシステムの動作を記述する[7]．

（4） ドキュメントとドキュメント作成ツール

ソフトウェア設計の後半の作業は，下流工程での利用に備えて設計結果をドキュメント化することである．ドキュメントとしては最近 **PAD** (Problem Analysis Diagram) が普及しており，ここでは PAD に関連したツールを紹介する．特にマイクロコンの場合にはアセンブラ言語やC言語を用いてプログラミングすることが多く，視覚性を補い，構造性を維持するために PAD の効果は大きい[8]．PAD ツールの一例としてソースプログラムから PAD を復元する Auto-DS の出力を図-5 に示す．Auto-DS はこのほかにモジュール間の参照関係を示すクロスリファレンス・テーブルを出力する機能を持つ[9]．PAD ツールにはこのほかに Auto-DS に類似のオート PAD，SDL (Software Design Language) を PAD に変換する SDL/PAD，編集用の PAD エディタなどがある．

最近，国内での利用を想定して，PAD ツールを含む各ドキュメントツールは**日本語入出力機能**を備えはじめている．

（5） ポーリングと割り込み機能の利用

マイクロコン応用では1個の CPU に複数の機能を与えるため，各機能を実行する処理プログラムの間の切り替えを管理する必要がある．応答性要求がさほどきびしくなく，処理要求の発生頻度も低い場合には，**ポーリング**方式で十分である．これは各処理要求源を順次たずねてまわり，処理要求が発生していればその場で処理して次へ進むという方式である．この方式は単純で明快である反面，全処理要求源を平等に見ているため，緊急度の差を考慮した**優先レベル別処理**ができない．対策としてはポーリング順序と処理要求源を対応させたポーリング・テーブルを用意し，優先度の高い処理要求源には複数のエントリを与えておき，1回のポーリングで2回以上ルックインさせる．

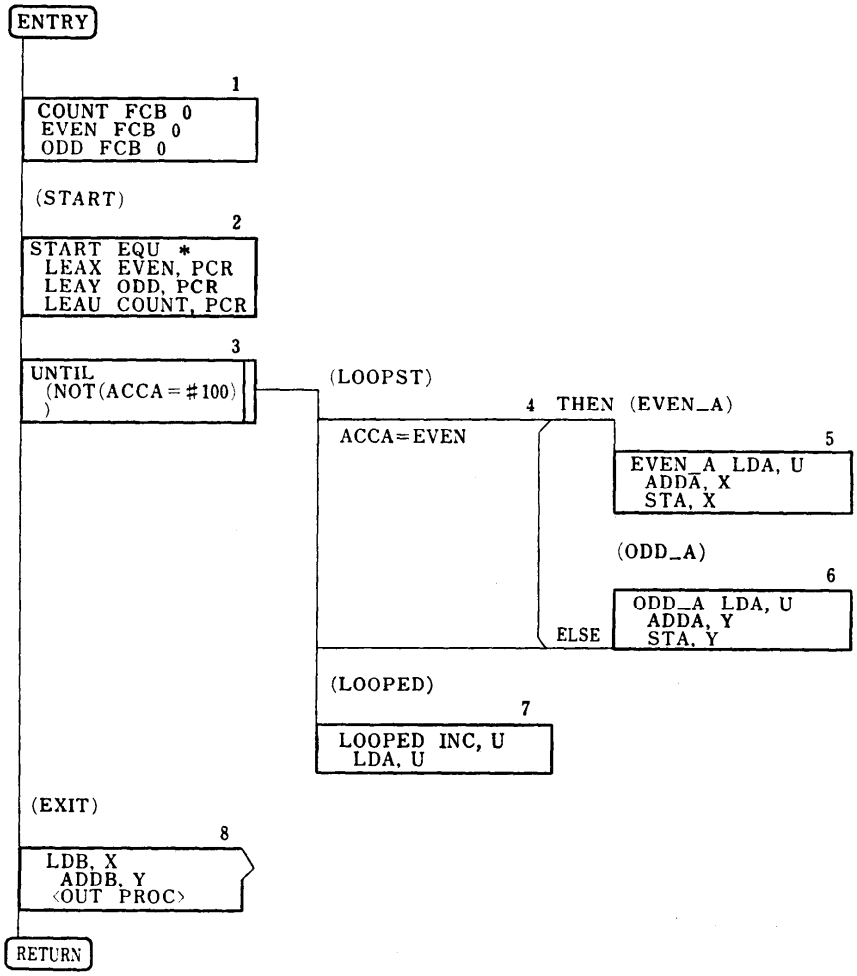


図-5　Auto-DS の出力例（MC 6809 のアセンブリソースプログラムを，PADに自動変換した結果）

応答性要求のきびしい処理要求がある場合には**割込み機能**を利用する．すなわち緊急処理要求信号だけを合成して複合割込み信号を作り，これをトリガとして処理要求源をポーリングする．

上記の方法はいずれも各処理プログラムそのものと，これらを管理する機能が混然一体となっているため，複雑なシステムを設計し，デバッグするのには適さない．そこで処理作業を機能別に分割し，それぞれを**タスク**と呼ばれる処理プログラムに負わせ，処理要求の受けつけと待ち行列への登録，タスクの優先レベル別起動と終了，タスク間共通変数の管理機能等を一括し，**リアルタイムモニタ**と呼ばれる管理プログラムに集中する．前述の同期と競合回避の問題も，リアルタイムモニタがタスク間の同期を管理することによって解決する．リアルタイムモニタはユーザが目的別に開発することもあるが，メーカが ROM あるいはフレキシブルディスクを媒体として提供しているものもある．

リアルタイムモニタによる制御方式を採ることにより開発作業の見通しが良くなり，工程・品質・プログラムサイズの管理が容易になる．また，既製のリアルタイムモニタの利用により，工数の約 30% を占めるリアルタイムモニタの開発作業が軽減される．

（6） OS とファイル設計

マイクロコンの応用が開始された当時は，プログラムの管理はリアルタイムモニタで十分であったが，最

近は事務処理に利用されることが多くなり，一方，機器組み込み制御応用の分野でもプログラムやデータが大規模化し，RAM に常駐させることができなくなってきている．使用頻度の低いプログラムモジュールやデータは，補助記憶に退避可能な**非常駐モジュール/データ**とする必要がある．これらの非常駐モジュール/データの**ファイル管理**のためには，リアルタイムモニタにファイル処理機能を付加するか，あるいは最初からファイル処理を含めたモジュール管理用 OS を設計する必要がある[10]．

開発目的に合致した既存 OS があれば，これを利用するとよいが，既存 OS の機能や性能，あるいはサイズが使用目的に適さない場合にはユーザが OS を自作し，ファイルを設計しなければならない．この場合，流通ソフトウェアを自己のシステム上で稼動させるためには，ファイルの互換性に対する配慮も必要である．大型コンピュータの場合には，ほとんどの場合メーカから提供される標準 OS を利用するため，ユーザがファイル設計をする必要がない場合が多いことを思えば，この点はマイクロコンの特徴といえる．

2.2 コーディングの技法とツール

マイクロコンのプログラムは，入出力ポートや割り込み機能などの，ハードウェアの条件を意識してコーディングを進める必要がある．ここではマイクロコンのためのコーディング技法とツールを紹介する．

（1） ツリーウォーク (tree walk)

PAD で設計したプログラムは，PAD をたどることによって機械的にソースプログラムに変換できる．この操作を**ツリーウォーク**と呼ぶ．

ツリーウォークはコーディングだけでなく，一度コーディングされた**プログラムの審査**（コードレビュ）にも利用できる．コーディング時と同様に PAD をたどりながらコードを追跡することにより，漏れのないレビュを行うことができる．

ツリーウォークによるコーディングをツール化したものが**プログラム・ジェネレータ**であり，PAD を PASCAL, PL/M, C などの高水準言語のソースコードに変換する機能を持つ．

（2） 高水準言語と言語プロセッサ[11]

マイクロコンのプログラムにおいても，最近は高水準言語の重要性が増大している．

特にマイクロコンの高水準言語とその言語プロセッサに要求される条件を列挙すると以下のようになる．

（a） 割り込みや内部レジスタの**制御機能**，入出力ポートに対するビット単位での直接操作などがきめ細かく記述できること．

（b） プロセス間の競合防止と同期，並列処理，手続きのリエントラント処理のような，**OS 記述**に必要な機能を備えていること．

（c） ROM と RAM へのアドレス割り付け，スタックサイズの算定など，マイクロコン特有の**コード生成機能**をもっていること．

（d） **豊富な実行時ライブラリ**とこれに対する接続機能に加えて，アセンブラ言語や他の高水準言語で記述されたプログラムとの**接続機能**を有すること．

（e） リンケージエディタによって接続されやすくするために，**分割コンパイル機能**によってリロケータブル・オブジェクト・モジュールを生成できること．さらに ROM 化後も任意のアドレスを与えることができるフロータブルコードの生成機能があれば，一層強力である．

（f） プログラムサイズが直接 ROM コストに影響するので，**オブジェクト効率**が良いこと．このためにはコンパイル速度の多少の犠牲は許容される．ただし，命令の実行順序を変えると実行結果が異なったり，トレースによるデバッグができなくなることがあるので，**最適化抑制モード**の指定機能も必要である．

（g） アーキテクチャの異なる多種のターゲット CPU に対して，使いやすくかつ高品質の言語プロセッサが用意されていること．プログラムの**移植性**は，このようにターゲット CPU 間だけでなく，ターゲット OS 間でも要求される．さらに，マイクロコンの場合は OS の存在を前提としないオブジェクトコードの生成機能も要求される．

（h） 多種のホストコンピュータや開発用 OS に対して言語プロセッサが用意されていること．さらに最近は**ホスト間移植**のため言語仕様はもちろんのこと，ファイル形式や生成されるオブジェクトコードに至るまで，互換性への要求が厳しくなってきている．

（i） 後工程のテストとデバッグに利用されるテストカバレジモニタ，テストベッド，そしてエミュレータによるトレース・ブレーク・変数値の表示と変更などに対する指定機能や，データの**シンボル情報**の受け渡しを，言語プロセッサがサポートしていること．

マイクロコン用として初期に利用された PL/M 系や PASCAL に代って，最近は C が多用され，さらに Ada*, Occam, Modula-2 などが開発され，提唱され

* Ada は米国政府の開発した言語であり，米国政府 AJPO (Ada Joint Program Office) の米国における登録商標である．

ている[12)~15)].

(3) ソフトウェア開発用ユーティリティ[16),17)]

コーディング段階で利用される主要なユーティリティの動向を以下に紹介する.

(a) **テキストエディタ**は，当初の1行単位での編集操作しかできなかったラインエディタに代わり，画面単位でのソース編集機能をもったスクリーンエディタが主流になってきている．また，高水準言語のシンタクスを認識する機能をもち，ソースプログラムの入力時に文法の誤りを指摘し，構造化を支援する言語別専用エディタ (Language Specific Editor) または構造化エディタが実用化されはじめている.

(b) **リンケージエディタ**はマイクロコンの特殊性を考慮して，メモリ素子の実装領域や入出力ポートのアドレスのように，リンク前後で変化してはならない情報の保存機能を有している．最近は次に示すライブラリエディタを兼ねるようになってきている.

(c) **ライブラリエディタ**はモジュールの登録と探索を行うものである．探し出したモジュールが別のモジュールを外部参照している場合でも，再帰的に探索を実行するようになっている.

(d) **ローダ**はプログラムファイルを RAM に読み込み，サムチェック（ビットパターンを2進累算して正常結果と比較），ベリファイ（繰り返し読んで読み込み結果をファイルと比較）などを行う.

(4) レジデントソフトウェアとクロスソフトウェア

初期の言語プロセッサとして利用されたものはレジデントアセンブラのみであった．これは利用対象と同一機種の CPU を内蔵した開発支援装置の上で稼働するもので，その簡便さに特長があった．しかし，プログラムが複雑かつ大規模になるにつれ，IBM 370 系の大型コンピュータや VAX 11 系のミニコンピュータをホストとした TSS や大容量ファイルなどの資源を活用した，クロスソフトウェアが着目されはじめた．クロスソフトウェアの利用はまずアセンブラから始まったが，シミュレータ，リンカ，コンパイラなどが利用されるようになった．また，テキストエディタやファイル管理機能は，ホストコンピュータに標準装備されたものがそのまま利用できる．クロスソフトウェアの特長は，まず，ホストとして利用できるコンピュータがあれば設備投資が少なくて済むことである．また，バッチ処理による計算センタのクローズドサービスを利用することにより，ターンアラウンドタイムの長い TSS 端末の前に長時間座りこむ必要もないので，デバッグの効率は向上する.

クロスソフトウェアの今一つの特長は，ハードウェアの完成を待たずにソフトウェアの開発を進め得る点である．ホストコンピュータの上にターゲットシステムの仮想のハードウェアを構成し，これのソフトウェアの開発やシミュレーションによる評価が可能である.

クロスソフトウェアは多くの長所を持っているにもかかわらず，ユーザがその導入を躊躇する理由は，ホストとなるコンピュータが高価なことである．これに対しては最近，パーソナルコンピュータをホストコンピュータとして利用できるクロスソフトウェアが利用されはじめている.

クロスソフトウェアの限界は，電圧やタイミング波形のような物理的条件まで含めた実機デバッグができないことである．これに対しては各 CPU のエミュレータと，これらを管理するエミュレータプログラムが開発されてきている.

従来はクロス/レジデント両方式の比較に関して多くの議論がなされ，各方式の支持者がそれぞれの方式を採用し，あるいは時期によっていずれか一方式が優勢を占めたことがあった．しかしこのような状況にも終止符が打たれようとしている．クロスソフトウェアに重心を移しながらも，ホストコンピュータとネットワーク経由あるいはフレキシブルディスクを媒体として接続されたエミュレータ付**ワークステーション**によるホストコンピュータへのアクセスと実機デバッグが，今後のツールの利用形態となっていく.

(5) ソフトウェア開発用 OS[18)]

以上に示した言語プロセッサや各種ユーティリティ，そして開発途上のプログラムを管理するために，OS が必要である．ソフトウェア開発用 OS はコーディング工程だけでなく，下流のデバッグやテストの工程でも利用されるほか，最近ではより上流の設計工程のツールの管理にまで利用範囲が拡大している．ソフトウェア開発用 OS は大別して，プログラムにデバッグ操作を加える**デバッグモニタ**と，ディスクに格納された上記のプログラムを管理する **DOS** (Disk Operating System) から成る．デバッグモニタはすべてのプログラムの上位に位置し，DOS の起動，物理アドレス空間の管理，プライマリ・マップ（開発支援装置のアドレス空間）とセカンダリ・マップ（開発対象システムのアドレス空間）の切り換え，ブレークポイントの設定と解除，トレースの実行とこれにともな

うメモリやレジスタの内容の表示と変更などの機能をもつ．最近はアセンブラ言語あるいは高水準言語のソースステートメントによりトレース開始位置やブレークポイントを指定し，変数値の設定や表示は変数名によって指定し，トレース結果もソースプログラムのステートメントで表示される，いわゆる**シンボリック・デバッグ**が一般化してきている．

DOS は補助記憶上のプログラムのファイル管理，セッション管理，ジョブ管理，言語プロセッサやソフトウェア開発用ユーティリティの管理を実行する．

ソフトウェア開発用 OS は開発支援装置の上で走行するものであり，前述のリアルタイムモニタやシステム組み込み用 OS とは異なる起源を持っている．このようにソフトウェアの開発と利用のフェーズが明確に分離されていて，開発時にのみ必要な機能は完成システムに含めない点に，マイクロコンの特殊性を見ることができる．

大型コンピュータあるいはミニコンピュータをホストとするクロスソフトウェア・システムにおいては，IBM の MVS やベル研究所の UNIX オペレーティングシステム* のようなホストコンピュータの OS が，DOS の機能を実行する．また，最近パーソナルコンピュータの OS を，開発支援装置の OS として採用する例が増えている．これらの場合には，モニタに相当する機能は**デバッガ**と呼ばれるユーティリティの一つとして，OS の管理下に置かれる．

ホストコンピュータとワークステーションを接続したソフトウェア開発システムでは，両者の間でファイル転送の機能が必要であり，この場合ファイル転送ユーティリティが必要である．また，ホストコンピュータと開発支援装置の間で，オフラインでファイルを交換する場合は，両者の間で**ファイル形式を統一**する必要がある．一般にホストコンピュータの方が機種が少ないため，統一ファイル形式をホストに合わせ，開発支援装置側でファイルコンバータを稼動させる．最近，すべてのツールの間で**オブジェクト・モジュール**のフォーマットを標準化 (MUFOM, SYSROF) する動きがある．

特に自動車や航空機に搭載されるマイクロコンのソフトウェアの実機デバッグのように，ホストコンピュータと開発支援装置をオンライン接続できない場合は，ファイル変換機能は必須である．

* UNIX オペレーティングシステムは，AT & T で開発されたソフトウェアであり，AT & T がライセンスしている．

2.3 デバッグの技法とツール

デバッグ作業はテスト作業と工程上不可分の関係にあり，技法やツールも両方の機能を兼備したものが多い．デバッグはまずプログラムの論理的な正しさを目標にして行う**論理デバッグ**より開始し，次に実機に搭載した状態での電圧やタイミングまで考慮した**実機デバッグ**を行う．ここでは適用フェーズの順にデバッグの技法とツールを紹介する．

（1） **シミュレータ**[19]

ターゲット CPU の内部レジスタをホストコンピュータの RAM 上に設定し，命令を模擬するものである．命令を1ステップずつ解釈し，レジスタの内容を表示あるいは記録しながら実行する**トレースモード**と，ある区間のプログラム全体を一度ホストコンピュータの命令列に変換してから実行する**エクゼキュートモード**の二種類のモードがある．デバッグの初期にはトレースモードによって細部のバグを除去し，次いでエクゼキュートモードによって区間全体の動作を確認しながらモジュールを接続して，プログラムをインテグレートしていく．

従来のシミュレータは CPU の動作のみを模擬するものであったのに対し，最近は入出力処理を模擬する**入出力シミュレータ**（I/O シミュレータ）が併用される．マイクロコンのアドレスマップ上にある入出力ポートに対するアクセスを，あらかじめ別の命令に置きかえておき，入出力を実行しようとした時には，特定のアドレスに飛んでいって論理的な入出力動作を実行する．論理的な入出力ポートとしてはキーボード，ディスク，CRT ディスプレイ，プリンタなどが利用される．

（2） **デバッガ**

開発支援装置のデバッグモニタに相当する機能は，大型コンピュータ，ミニコンピュータあるいはパーソナルコンピュータの OS には含まれていない．そこで，デバッグ機能を一つのユーティリティとして，これらの OS の管理下に置く．

（3） **テストベッド**

プログラムモジュールのテストと論理デバッグを実行するために，対象モジュールの上位モジュール，下位モジュール，入出力データ，外部データなどの環境を模擬して，実行するものである．**図-6** はテストベッドの機能の一例を示している．24 種のコマンドを持ち，この内 CALL コマンドにより上位モジュールを，STUB コマンドにより下位モジュールをそれ

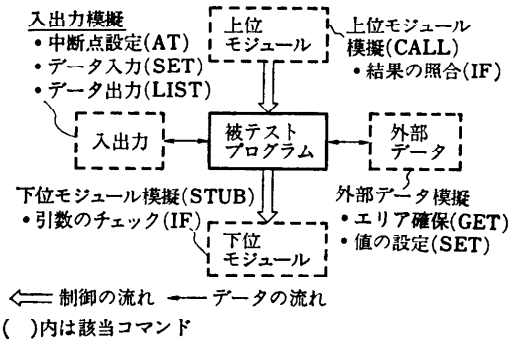


図-6　テストベッドの機能例

ぞれ模擬する．入力と外部データの設定には SET コマンドを，出力結果のチェックには IF コマンドを使用する．これらのコマンドを組み合わせることによってテスト手続きを記述することができる．頻繁に利用されるコマンド列はCLISTコマンドによって定義し，マクロコマンドとして利用できる[20]．

（4）**エミュレータ**(インサーキットエミュレータ)

エミュレータは実機デバッグに利用するハードウェアで，ターゲット CPU と同機種の CPU を内蔵している．ユーザプロトタイプから CPU を抜去し，エミュレータを挿入してターゲット **CPU の動作を代行**させる．プローブによって CPU あるいは周辺回路の信号を採取し，これらの同時的あるいは時系列的組み合わせによる**ブレーク条件**によって停止し，停止までの数百サイクルの**リアルタイム実行時の信号を凍結して**，物理的な波形と逆アセンブラによるニーモニックで表示する．ブレーク条件によって停止させることなく，条件成立時にトレース信号だけを出力して処理を続行させることもできる．このトレース信号はオシロスコープのトリガ信号として利用できる．ソフトウェアアナライザにより，リアルタイム実行中の各モジュールの滞留時間や実行回数などの統計データを採取し，システム性能向上の障害になっているモジュールを同定することができる．リアルタイム実行時だけでなく，**シングルステップトレース実行時**にも，各種条件で停止させることが可能である．この場合には，リアルタイム実行時よりも細かい停止条件の指定が可能である．

リアルタイムあるいはシングルステップでのトレースは，各命令単位だけでなくジャンプ/トレース命令のみを対象にも実行可能である．これによってプログラムが停止するまでに通過したパスのみに着目し，各分岐条件の成立/不成立時のパス上の信号あるいはレジスタやメモリの内容を確認することが可能である．

従来のエミュレータは開発支援装置本体との分離が不完全で，使用上には各種の制約があった．本体とエミュレータのメモリ共用によるアドレス空間の競合(メモリコンテンション)，一部の信号端子や命令の使用禁止などが制約の具体的項目である．これらの制約を除去するために，最近は本体とエミュレータの間を疎結合し，プライマリマップとセカンダリマップを分離するなど，開発支援装置の存在を意識することなく開発できるよう，**トランスペアレント化が進んでいる**[21]．

最近は言語プロセッサとのシンボル情報の授受により，エミュレータ利用時にもシンボリックデバッグが可能になってきている．

各変数のリアルタイムトレースは，現在まだ一定アドレスを割り付けられた，いわゆるスタティック変数だけを対象とするものが多い．これに対し一部の開発支援装置では，アドレスがスタック内にあって実行時に動的に変化する**オートマチック変数のトレース**が可能になってきている．さらに今後はレジスタ内に割り付けられた変数のリアルタイムトレース機能へと，開発が進んでいく．

図-7 は開発支援装置とエミュレータを利用してプログラムの開発を進める過程を示している．テストと論理デバッグの段階までは開発支援装置内のプライマリマップの上で作業を進め，実機デバッグの大半はエミュレータのセカンダリマップ上で実行する．デバッ

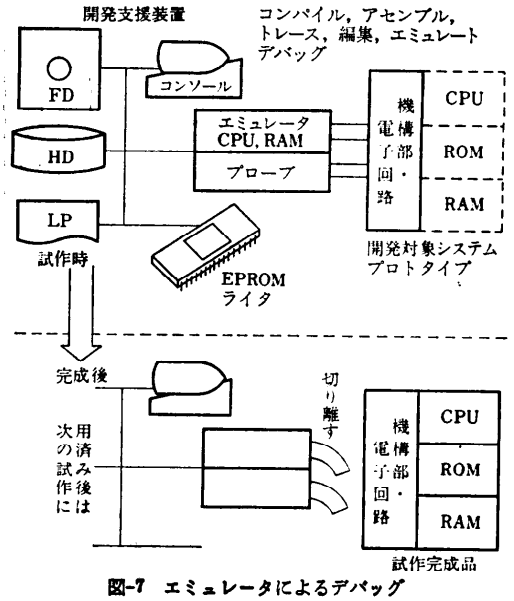


図-7　エミュレータによるデバッグ

グの完了したプログラムを順次プロトタイプの ROM 上に移して動作確認し，最後にエミュレータを抜去し，CPU を挿入してプロトタイプを独立に動作させる．

（5） EPROM ライタとイレーザ

EPROM ライタは EPROM（電気的書込み ROM）にプログラムや定数を書き込む装置であり，エミュレータによるデバッグの完了後，セルフスタンディング状態で動作確認するために使用される．その機能は次の三段階に分けて実行される．

（a） ピンの接続異常や逆差しのチェック，ブランクチェックを行う．最近は EPROM 自身に記憶されている**シリコンシグネイチャ**を読み取って EPROM の種類や仕様を判断し，書込み条件を設定するようになってきている．

（b） EPROM の内容を 1 バイトずつ読み出し，書き込んだデータと比較し，不一致の場合書き込んだ後読み出してチェックする．書込みデータの入力方法はマスタ ROM からのコピー，キーボードからのマニュアル入力，開発支援装置からの転送などがある．

（c） 書込み終了後，EPROM ライタ内の RAM の内容と，書込み結果を比較する．マスタ ROM あるいはファイルの内容と比較する場合もある．

イレーザは EPROM の内容を書き替える前に，書込み済みの内容を消去するものである．**紫外線消去方式の EPROM (UV PROM)** の場合は，**紫外線源**とタイマを内蔵して紫外線照射時間を適正に維持する．

（6） 評価用チップ/ボード/キット

マスク ROM 内蔵の CPU 素子のプログラムの開発には，評価用チップ，評価用ボードまたは評価用キットが利用される．

評価用チップ (evaluation chip) は，マスク ROM 内蔵 CPU の EPROM 版，あるいはマスク ROM を除いたものである．高価で長納期のマスクを作る前に，内蔵または外付けの EPROM を内蔵マスク ROM とみなして，この上に開発途上のプログラムを格納して動作確認する．そのためにシングルステップ機能があり，EPROM 外付けの場合には，EPROM 接続用端子を持っている．

最近 CPU の集積化が進み，マスク ROM に加えて入出力ポート，AD 変換器，DA 変換器等を内蔵するに至り，内部信号を端子から検出することによる詳細な構造テストやデバッグが困難になってきている．そこでデータバスやアドレスバスのような内部信号を，LSI のパターンの途中から外部に引き出してモニタできるようにした評価用チップ，いわゆる**ボンドアウトチップ**も，用意されはじめている．

評価用ボードは評価用チップと EPROM 用ソケッ

```
Module Name : [ CLOCK ],  Coverage Mode : [ Audit ]          Assembled Date & Time    1985/ 8/21 (Wed)  9:54    PAGE    1

** Summary_Report **
      +-------------------------------------------------+---------+---------+
      |          Contents                               | Current |  Total  |
      |-------------------------------------------------+---------+---------|
      | (1) |  Executable Instructions                  |      58 |      58 |
      | (2) |  Executed Instructions                    |      23 |      53 |
      | (3) |  Total Branch Instructions                |      26 |      26 |
      | (4) |  Total Branch Directions                  |      39 |      39 |
      | (5) |  Executed Branch Directions               |       9 |      25 |
      |-----+-------------------------------------------+---------+---------|
      | (6) |  C0 Measure  =  (2)/(1)                   |    39 % |    91 % |
      |-----+-------------------------------------------+---------+---------|
      | (7) |  C1 Measure  =  (5)/(4)                   |    23 % |    64 % |
      +-----+-------------------------------------------+---------+---------+

** Measure_Graph **  0%        10%       20%       30%       40%       50%       60%       70%       80%       90%       100%
      +----+---------+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
      |    | Current |##########################################)                                                          |
      | C0 +---------+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
      |    |  Total  |##################################################################################################)|
      +----+---------+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
      |    | Current |#######################)                                                                              |
      | C1 +---------+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
      |    |  Total  |#############################################################)                                  |
      +----+---------+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+

** Step_Report **
      +---------------------+-------------+
      | Static Steps        |          58 |
      +---------------------+-------------+
      | Current Dynamic Steps |       16492 |
      +---------------------+-------------+
      | Total Dynamic Steps |       32862 | +a
      +---------------------+-------------+
```

図-8　テストカバレジモニタによる出力例（上段が実行した命令と分岐の数と割合を示し，中央はそれをグラフによって表している．下段は動的ステップ数を示す．本図は(株)日本コンピュータ研究所のご厚意による）．

トを一枚のプリント基板に実装したもので，プロトタイプに接続してプログラムとハードウェアの評価と確認に利用できる．

評価用キットは評価用ボードにプログラム開発支援機能を強化したものである．プログラム開発用のモニタ，アセンブラ，テキストエディタを格納した ROM と，これを実行する別の CPU，作業領域用の RAM，EPROM ライタを持っている．デバッグすべきプログラムは RAM 上で編集，アセンブル，デバッグなどの操作を受け，EPROM に書き込まれる．

2.4 テストの技法とツール

本節では特にテストにのみ利用可能で，それ自身でデバッグに利用されない技法とツールを紹介する[22]．

テストは，プログラムの外部仕様にのみ注目し，その入力と出力の関係にのみ注目する**機能テスト**と，プログラムの内部構造にまで立入ってテストする**構造テスト**に大別される[23]．

機能テストを効率的に行う技法およびツールとして，原因結果グラフと，これをもとにテストケースを生成する AGENT が知られている[24]．また，構造テストのツールとしては C 0/C 1 網羅率を測定するテストカバレジモニタがあり，図-8 はその出力例を示す．

2.5 ソフトウェア管理ツール

本章で紹介した各ツールが工程ごとに利用されるものであったのに対し，最近は全工程にわたって**進捗状況管理**や，**バージョン管理**を行うツールが実用化されはじめている[25],[26]．生産性向上の究極の策はソフトウェアを生産しないことであるという観点からの，**ソフトウェア再利用技術**の有効性がマイクロコンの分野でも注目され，この種のツールが導入されはじめている．

3. 今後の動向

以上に示した各工程ごとの技法およびツールは，今後どのような進化をたどるであろうか．以下にユーザ側の希望を含めて今後数年間に起こり得る動向を予測してみる．

（1） マイクロコンとしての特殊性の減少

ハードウェアに対するソフトウェアの相対的なコストが増大するにつれ，マイクロコンのソフトウェア開発の条件が大型コンピュータのそれに類似してきている．今後，開発の主流が 16 ビットから 32 ビットへ移行するにつれ，技法・ツールともに上流工程への遡上が一層進行し，マイクロコン用としての**特殊性は急激に減少**していく．

（2） 体 系 化

従来は起源を異にするものが混在して技法群あるいはツール群を構成していたが，今後はこれらが上流から下流に至る**一貫したソフトウェア開発環境**という観点から再編成される．そして共通のホスト，OS ならびにデータベースの上で稼働し，記述言語，ファイル形式，コマンドが統一され，全体が一貫した思想のもとに構築されていく．情報処理振興事業協会のシグマシステムは，その具体化の例であり，今後注目する必要がある[27]．

手工業としてはじまったマイクロコンのソフトウェア開発は，こうして遠からず**装置産業**となる[28]．

（3） アーキテクチャ上の技術革新

CPU のアーキテクチャが複雑になると，プログラムのデバッグはますます困難になる．たとえばパイプライン処理のための命令先取り制御はブランチトレースや C 1 カバレジテストの障害になり，スーパバイザモードの装備やキャッシュメモリの内蔵は外部信号によるモニタを困難にした．従来の CPU の技術革新に対しては，開発ツールはその都度対応し，これらを解決してきた．しかしこのような努力にもこのままではいずれ限界がくることは明らかである．CPU の開発元は，その機能と性能の向上に注いできた努力の何％かをデバッグ機能の充実に振り向けるべきであるし，また，その徴候が見える．命令トレース機能の CPU への内蔵や評価用チップの同時開発は，**デバッグ機能のアーキテクチャへの作り込み**の第一歩である．今後はさらにリアルタイム実行時の命令のパイプライン実行ステージ，内部レジスタやキャッシュメモリの内容を外部から直視できる端子の付加などによる，ツール開発への支援が期待できる．

（4） 生産性と操作性の向上

ソフトウェア開発量の増大と開発従事者層の拡大への要求から，生産性と操作性の向上が進む．

ドキュメントの**日本語**入出力と，マルチスクリーン表示の OS と高水準言語によるサポートは，現在より一層一般的となる．

開発作業の重点がソフトウェアの構造設計から仕様設計へ移り，仕様のドキュメントから直接プログラムを自動生成するコードジェネレータやプリプロセッサが普及し，**コーディング作業の必要性はますます低下**する．これにともなってテスト作業も設計ドキュメントに対する直接のトレースやデバッグの操作が中心に

なる.

操作性向上の障害となっていた，人間の知覚とホストコンピュータの動作の相違（セマンティックギャップ）を解消するために，**知識工学**（AI）が開発ツールに組み込まれて利用される[29].

4. おわりに

以上にマイクロコンのソフトウェア開発用の技法とツールを紹介してきた．大型コンピュータのソフトウェア開発用の技法やツールと重複する記述を意識的に避けたが，実際には前章に述べたようにマイクロコンと大型コンピュータの技法やツールの差異は減少しつつある．大型コンピュータより四半世紀遅れて登場したマイクロコンが，大型コンピュータから多くを学びながら発展してきたことは，“個体発生はその種族の進化過程（系統発生）を再現する”という進化論の常識に合致している．しかし，“ソフトウェアの危機”をマイクロコンに招来することは，なんとしても避けねばならない．幸い自由度が比較的残されているマイクロコンの分野では，最近のソフトウェア工学の成果を採り入れることによって，ソフトウェアの生産性を向上することが可能である．

本記事が一人でも多くの研究者や開発者の目に止まり，これを技法とツールの改良発展あるいは適定利用のヒントにして頂けるならば，筆者らにとってこれ以上の幸いはない．

謝辞 本記事の執筆時に，取材にご協力賜り，討論に応じていただいた多くの方々，とりわけ横河ヒューレット・パッカード(株) の清水千博氏と根城寿氏，ソニー・テクトロニクス(株) の浦野鐵男氏，ならびに(株)日立製作所システム開発研究所の渡辺坦氏に心からお礼申しあげます．

参考文献

1) 渡辺 茂，正田英介，矢田光治（編）：マイクロコンピュータ・ハンドブック，pp. 573-574，オーム社，東京（1985）.
2) Fujino, K., Stucki, L. G. (Chairpersons), Branstad, M. A., Kato, H., Muraoka, Y. and Zelkowitz, M. V. (Panelists): Impact of New Technologies on Software Engineering, Panel Session PC 2, 6 th International Conference on Software Engineering (1982).
3) 小林正和，野木兼六，片岡雅憲，林 利弘：ソフトウェア一貫生産システム “ICAS” 基盤技術の確立，日立評論，Vol. 66, No. 3, pp. 1-6 (1984).
4) 加藤肇彦：マイクロコンピュータ応用システムの開発手順，計測と制御，Vol. 19, No. 4, pp. 392-400 (1980).
5) 加藤肇彦：機械制御へのマイクロコンピュータの応用，精密機械，Vol. 47, No. 2, pp. 104-110 (1981).
6) 当麻喜弘（編）：マイクロコンピュータ応用ハンドブック，pp. 513-519，昭晃堂，東京（1983）.
7) 山崎秀記：ペトリネットの理論と応用，情報処理，Vol. 25, No. 3, pp. 188-198 (1984).
8) 二村良彦：プログラム設計法 PAD/PAM，情報処理，Vol. 25, No. 11, pp. 1237-1246 (1984).
9) 鵜飼純一，橋本祐宏，増井光幸，上藤博司：ソフトウェア保守支援のためのドキュメント・システム Auto-DS，情報処理学会（昭和60年度前期）第30回全国大会，1S-7, pp. 577-578 (1985).
10) 加藤肇彦，筒井茂義，茶木英明：マイクロコンピュータ用オペレーティングシステムの現状と動向，情報処理，Vol. 25, No. 3, pp. 232-243 (1984).
11) 渡辺 坦：プログラミング言語，電学誌，Vol. 103, No. 5，特集：マイクロコンピュータ応用技術，pp. 470-473 (1983).
12) B. W. カーニハン（石田晴久訳）：プログラミング言語C，共立出版，東京（1981）.
13) 情報処理振興事業協会（編）：最新 Ada 基準文法書，bit 別冊，共立出版，東京（1984）.
14) INMOS Ltd.: Occam Programming Manual, Prentice-Hall, Englewood Cliffs (1984).
15) Wirth, N.: Programming in Modula-2, Springer-Verlag, Berlin (1983).
16) 68000 CRT エディタユーザーズマニュアル，日立製作所，東京（1981）.
17) 68000 リンケージエディタユーザーズマニュアル，日立製作所，東京（1982）.
18) FDOS ユーザーズマニュアル，日立製作所，東京（1981）.
19) Mシリーズ/IBM 用 68000 クロスアセンブラユーザーズマニュアル，日立製作所，東京（1982）.
20) Chusho, T. and Honda, A.: HITS: A Symbolic Testing and Debugging System for Multilingual Microcomputer Software, National Computer Conference, pp. 75-80 (1983).
21) 68000 ASEII ユーザーズマニュアル，日立製作所，東京（1985）.
22) Myers, G. J.（長尾 真訳）：ソフトウェア・テストの技法，近代科学社，東京（1980）.
23) 中所武司：ソフトウェアのテスト技法，情報処理，Vol. 24, No. 7, pp. 842-852 (1983).
24) 古川善吾，野木兼六，徳永健司：AGENT：機能テストのためのテスト項目作成の一手法，情報処理学会論文誌，Vol. 25, No. 5, pp. 736-744 (1984).
25) ユーザ指向のロジック開発システム64000シリーズ，アプリケーションノート 64000-3，横河ヒューレットパッカード，東京（1985）.
26) 今居和男，山東 滋，木村伊九夫：ライブラリ管理システム LIME，情報処理学会（昭和57年前期）第24回全国大会，4N-5, pp. 339-340 (1982).
27) 情報処理振興事業協会シグマシステム開発本部（編）：シグマシステム基本計画，ソフトウェア生産工業化システム構築基本計画書（1986）.
28) 特集：高機能ワークステーション，情報処理，Vol. 25, No. 2 (1984).
29) 特集：知識工学，情報処理，Vol. 26, No. 12 (1985).

（昭和61年7月3日受付）