58—R010

マイクロコンピュータ応用システムの開発技術

昭和59年3月

JIPDEC

財団法人 日本情報処理開発協会

この報告書は，日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて，昭和58年度に実施した「マイクロコンピュータの応用に関する調査研究」の一環としてとりまとめたものであります。

はじめに

マイクロコンピュータは，今やコンピュータ，ワークステーション，周辺端末機器，通信制御装置などの情報関連機器から，自動車，家庭用電気製品などの家庭用機器まで，あらゆる製品やシステムの中心的構成部品として使用されており，我が国が指向する高度情報化社会を，その土台から支える技術といっても過言ではない。

このようなマイクロコンピュータ応用システムの開発技術は，我が国先端技術開発の基礎的技術として今後ますます重要な位置を占めるものである。

当協会マイクロコンピュータ振興センターでは，昭和53年度以来，このマイクロコンピュータ応用技術に関する調査研究を行っているが，本年度においては，マイクロコンピュータ応用システムの開発技術の全貌を明らかにすることを目的として，開発工程，開発ツール，開発環境などをいくつかの事例を含めてとりまとめるとともに，開発技術の展望を行った。

ここに，調査研究の実施にあたり，ご指導，ご協力をいただいた関係各位に厚くお礼申し上げるとともに，本報告書が，広くマイクロコンピュータ応用システムの研究・開発あるいは教育の場において活用され，わが国の産業の一層の発展に寄与することができれば幸いである。

昭和59年３月

マイクロコンピュータ応用技術調査委員会

（順不同，敬称略）

委員長	西川　禕一	京都大学工学部 電気工学科教授
委　員	阿草　清滋	京都大学工学部 情報工学科助教授
	梶浦　信孝	アイ電子測器（株） 開発技術部部長
	中田　祐造	東芝エンジニアリング（株） マイコン制御技術部 マイコンソフトウェア課課長代理
	原田　睦明	日本総合システム（株） 常務取締役
	松本　建夫	山武ハネウェル（株） プロセス制御事業部技術部システム開発 グループグループマネージャ
	真弓　和昭	松下電器産業（株）中央研究所 計算機応用グループ グループマネージャ
オブザーバ	小椋　正幸	通商産業省機械情報産業局 情報処理振興課
	小松　正則	通商産業省機械情報産業局 電子政策課
事務局	（財）日本情報処理開発協会マイクロコンピュータ振興センター	

目　　次

序　　論

序　　論

近年，先端技術開発の推進ということがしきりに言われている。あるいは，脱工業化社会，情報化社会とか，ソフト化社会といつた言葉もさかんに語られる。一律の規格の工業製品が，固定化された生産設備と生産機能のもとで大量に生産され，大量に消費される，それが従来の工業化社会のイメージであったとすれば，消費者のニーズに応じてよりバラエティに富んだ多様な製品が生産され，好みに応じて消費される時代に移行しつつある。従って，個々の製品にはより高度な機能が内蔵されるようになり，それだけ付加価値が増すことになる。品質や機能・性能を高度化した製品を生み出すこと，それは資源の乏しいわが国に課せられた技術立国という課題を解決していく道でもある。

先端技術の一つの大きな柱は，言うまでもなくエレクトロニクスの技術であり，なかでもコンピュータを中心とするマイクロエレクトロニクスの技術である。それは取りも直さず，個々の製品に一定レベルの情報処理機能，すなわちインテリジェントな機能を付与するものであり，また幾つかの製品を有機的に統合してシステムとしての機能を付与するものである。さらには，遠距離情報伝達すなわち通信の機能を付与することによって幾つかのシステムを結合し，相乗的により拡大された能力と効果を発揮させることもできる。つまり，単一の製品からそれらを統合したシステムにまで，従来に無かった多様で精細な可能性を与えることができるインテリジェントな技術なのである。

1971年にIntel 4004が登場して以来10年余，マイクロプロセッサは4ビットから8ビットへ，そうして12ビット，16ビットへと語長が拡大され，さらに近年には32ビット機も出現してきた。これは主として材料の製造技術と回路の集積技術の進歩，そして品質の安定した市場商品を生み出すハードウェアの生産技術と生産管理技術の進歩によるものであるが，それに伴って応用システムの分野も着々と拡大されてきた。そして語長によっておよその役割分担ができている。すなわち，4ビット・プロセッサは主として電卓や簡単

な制御機能向きとして，例えば家電製品，ゲーム・マシン，自動車，防災警報装置，キャッシュレジスタ，データエントリー・システムといったものから比較的簡単な端末機，機械装置，プロセスなどの制御用に，今日でも数多く使われている。８ビット・プロセッサは主としてやや複雑な制御機能と伝送機能向きとして，例えばゲーム・マシン，印刷機，事務用端末機，通信機器，通信システム，コンピュータ周辺装置，POSターミナル，データ収集装置，データ・ターミナル，工業用計測・計装装置，工業プロセス・オンライン装置，公共設備・装置，図形処理装置などに応用され，また電卓や一般用のパソコンなどにも多く利用されている。さらに，１６ビット・プロセッサは情報処理機能と高度な制御機能を備えたものとして，スタンドアロン形レジスタ，通信機器・システム，データ・ターミナル，コンピュータ周辺装置，工業用計装・オンライン装置，工業用分散処理システム，航空・航海システム，図形処理装置などに普及しつつあり，高級パソコン用としてもこのところ急激にシェアを伸ばしている。ちなみに，１９８２年における国内パソコン市場での８ビット機，１６ビット機のシェアはそれぞれ９３.３％，６.７％であったが，８４年にはそれが６６.８％，３３.２％にまで変わるであろうと推定されている。

このようなマイクロプロセッサ応用市場の量的かつ質的な拡大に伴って，わが国でもこの１０年ほどの間に膨大な生産業界が形成され，そこには多種多様な既成産業の参入，例えば総合電機メーカ，通信機メーカ，コンピュータ・メーカ，半導体メーカ，精密機器メーカなどの参入が見られた。また一方では新しい専業メーカも誕生し，システム・ハウスあるいはソフト・ハウスといったベンチャー・ビジネスも数多く生まれたのである。さらには利用者（ユーザ）側にも，例えば素材メーカ，一般機器メーカ，教育・ホビー用機器メーカなどが，内製段階からやがては一般市場へ参入を図るという傾向も現れた。であるから，現在マイクロプロセッサあるいはマイクロコンピュータを応用したシステムを開発しつつある企業の種類，立場や規模は様々であり，またそこで開発されつつあるシステムの種類も極めて多岐にわたっている。

このような多様性が生まれたのは，マイクロコンピュータ技術（以後便宜上，多少の違和感はあえて無視して，マイクロプロセッサとマイクロコンピュータを同義語的に使用する）の本質に由来している。それは縦割り形の技術の一分野でなく，ほとんどあらゆる分野の技術と結合し得るもの，そしてそれらに新しい質的な発展の可能性を与えるものという，いわば横割り形の基礎を成す技術だからである。さらに形状的には文字通りマイクロであり，コスト的にもあまり負担がかからない。技術の面でも様々なレベルの人達が様々なレベルで，それなりにこなして対応し得るという性格を持っている。このようにまことに重宝なツールであるにも拘らず，あるいはあまりにも重宝なツールであるが故に，さて腰を据えて本格的にマイクロコンピュータ応用システムの開発と取組むとなると，必ずしも道は平坦でない。それどころか，現実には幾つかの困難に直面せざるを得なくなる。従来の技術開発の延長上では把えきれない，質的な新しい問題が立ち現れてくるのである。

この度，本委員会で取り上げたのは，マイクロコンピュータ応用システムの開発技術の本質と実体を探り，さらには将来の発展の方向を展望するという課題である。

上述したように，システムの種類とその開発に携わる企業・技術者の種類はともに極めて多様であるから，システム開発技術についても一様普辺な方法論があると考えたわけではない。また技術のニーズやシーズが急激に，また複雑に変化しつつある現状を見れば，将来の発展方向について透明な見通しが得られると考えることも無理であろう。そもそも物事には共通性・一般性の側面と，一方では個別性・特殊性の側面とが在る。それ故，一般性だけですべてを束ねてしまうことは無意味であると同時に，特殊性だけに目をうばわれていては大勢の把握ができず，進歩の機会を失うことにもなるであろう。本委員会の作業を進めるにあたって，この二つの側面，二つの観点は念頭に置いていたはずと考えている。

報告書をまとめる前に，委員会では幾度か論題の提示とそれをめぐる討論が

行われた。各委員の所属する機関（大学及び民間企業）とそこにおける立場もバラエティに富んだものであったから，特殊的な観点からの意見やコメントと，それらに共通したかなり一般的な問題点とを，いわば自然な形で浮彫りにすることができた。共通の問題点としては，例えばシステム開発を始めるにあたってまず必要な，要求定義や要求仕様が意外と明確でないことが多いと指摘された。特定のユーザからの注文によってシステム開発を行う場合でも，発注者自身の要求定義があいまいであったり，発注者から開発者へのコミュニケーションが不充分であったりするというわけである。システム設計やソフトウェアの設計手順もおよその筋道は整理されていても，依然として手探り的な要素も多く含まれている。進歩し変遷しつつあるハードウェアに対して，それをサポートするシステム開発ツールが追いつかないという現実もある。ソフトウェアの生産に関しては，開発途次の製品を十分に把握・評価しきれないため，作業の工程管理がむづかしいという指摘が多い。また，製品テストの手法や技法もかなり公式化，定量化されてきたとはいえ，個々の具体事例に応じて見れば，未だ十全とはいい難い。さらに，製品に関するドキュメンテーションやユーザのためのマニュアルの整備も，不満足な場合が多い。その他，市場・ニーズの動向の把握や，技術の将来発展の予測とそれに対する方策，新しい開発技術者の養成と技術の継承の問題等々までを含めると，マイクロコンピュータ応用システム開発にはまだ数多くの困難と悩みが見られるのである。

以上のような論議と考察の結果は，本報告書で4つの章にまとめられることになった。まず第1章では，システム開発の全体像を把えるべく試みる。すなわち，一般論的な立場からマイクロコンピュータ応用製品の開発工程を整理し，要求定義からシステム設計を行う段階，次にハードウェアとソフトウェアのプログラムに分割してそれぞれの設計，製造，テストを行う段階，そして組合せ・総合テストを経て製品量産化を行う段階それぞれについて要点をまとめる。さらに，ロジック・アナライザ，インサーキット・エミュレータ，コンパイラなど，ハードおよびソフトの開発支援ツールについて最近の状況を述べる。

第2章では，システム開発の現状を代表的な3つのシステム，すなわちワンチップ・マイコン応用システム，プロセス計測・制御システム，ＯＡシステム（特にワードプロセッサ用ソフトウェア・パッケージ）開発の最近の事例に即して，具体的に紹介する。これらは具体例であるとはいえ，かなり広い範囲の応用システム開発に際して，参考になる点が多いであろう。

第3章はシステム開発技術の課題を整理し，将来の発展に際して考慮すべき要点をまとめたものである。既に上述したように，現在のところ解決すべき課題が特にソフトウェアやシステム化技術の上で多く存在するが，ハードウェアの発展の方向とも絡めて，ソフトウェア工学の応用，ＯＳシステムを利用したワークスベンチの考え方などについて論じる。

第4章はマイコン応用製品及びシステム開発サポートツールの中から，最近のトピックスを解説的に紹介したものである。

以上の各章は委員会での討論を踏まえて各委員により分担執筆されたものであるが，第4章の一部については委員外からのご寄稿を賜った。ご協力に対し深謝の意を表する次第である。

第1章　マイクロコンピュータ応用システム開発の全体像

第1章　マイクロコンピュータ応用システム開発の全体像

1.1　開発工程

マイコンは，電卓の論理回路用LSI化に端を発したといわれているインテル4004の出現以来，その後，息もつかせぬ発展を示し，その応用分野では，我々の周囲にある製品にマイコンが大なり小なり関連しているといっても過言ではない。

マイコン応用製品には，家電等の民生用，計測・制御等にみられる様なシステム内に組込まれた1チップ汎用マイクロプロセッサを応用して，計算機外の「具象」（外界の各種条件を対象）を対象として，機能を解決するための手段として用いる他律的な製品とか，パソコン，マイコン開発システム等でみられる様に，それ自身がコンピュータとして機能する自律的な製品がある。

また，マイコンの性能は，従来の大型・中型コンピュータまたはミニコンに勝るとも劣らないものから，性能は劣るが，安価で小さいものまで種類も多く，マイコン応用システムの開発においては，価格，大きさ，機能，性能等の目的を明確化することが重要であり，その目的により，各工程は必ずしも同じではないが，本章ではソフトウェアを中心に，一般論としての開発工程について述べる。

1.1.1　工　　程

マイコン応用製品の一般的な開発工程は図1－1のとおりである。

各工程において，一般的と思われる主なドキュメントは，表1－1のものがある。しかし，システムの規模等により，必要なドキュメントを作成すべきであり，全てを作成することはない。また，表1－1に記してないドキュメントでも，必要に応じて作成することがある。

各工程は，連続的に進行するのが一般的であるが，費やす時間は，上

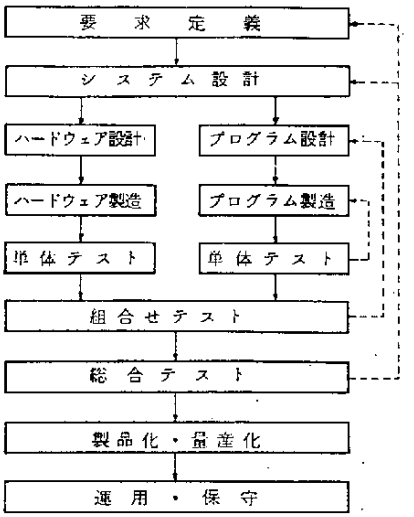


図1－1　開発工程

表1－1　各工程における主なドキュメント

工　程	主なドキュメント
要　求　定　義	開発計画書 システム仕様書 工程表
シ　ス　テ　ム　設　計	ソフトウェアシステム設計仕様書 ハードウェアシステム設計仕様書 工程表
プ　ロ　グ　ラ　ム　設　計	プログラム設計書（流れ図，HIPO等） 操作説明書（内部用）
ハードウェア設計	ハードウェア設計書（論理回路図等）
プ　ロ　グ　ラ　ム　製　造	リステイング プログラム・ソース プログラム・オブジェクト
ハードウェア製造	用品組立図
単　体　テ　ス　ト	単体試験仕様書 テストデータシート
組　合　せ　テ　ス　ト	組合せ試験仕様書
総　合　テ　ス　ト	総合試験仕様書 総合試験成績書
製品化・量産化	取扱説明書（外部用）

位工程ほど多く時間をかけるほうが良い製品が早く出来る結果になるのが常である。しかし，現実は，つい目先の工程に追われて，上位工程をおろそかにし，プログラム製造，テストと進むにつれて，後もどり作業が多くなり，全体的に開発が遅れたと言う話は，日常よく耳にする話である。

図１－１でもわかる様に，各工程の誤りや欠除が発見された時は，点線のパスを通って上位工程へ戻る。したがって，上位工程での誤りや欠除ほど修正パスが長くなり，損失が大きいこととなる。

特にソフトウェアは，「目に見えない」とよく言われるが，これは各工程において，未完成のまま，あるいは他の人の確認なして次工程に移ることにより，製品が出来てみないと，完了したか否かがわからなくなることであり，これはまた後もどり作業が多く発生する要因でもある。

「目に見えない」のは，製品に組込まれたプログラムであり，ソフトウェアの工程は必ず「目に見える」方法を取るべきである。例えば，各工程から次工程に移る時には，必ずデザインレビュー，またはウォークスルーを実施し，他の人の検査を受ける。また，各工程の担当者を分けることにより，中途半端な作業は出来なくなり，誤り，欠除を未然に防止することが出来る。

今後も増々マイコン応用システムの開発が行われるが，ハードウェアの場合は，使用する一つ一つの部品について，性能・特性等を考慮し，極力すでに使用したことのある部品を選択するであろうが，ソフトウェアの場合は，すでに作った部品（一つのモジュールまたはサブルーチン等）の利用が少なく，設計者は，すでにある部品の利用よりも，新規に作りたがることが多く，その結果として，設計時間の浪費，誤りの発生が生まれる。

ソフトウェアは，「作る」より「使え（部品の再利用）」を設計時に実施し，また管理者は実施させることにより，より早く，より安く，よ

り信頼性の高い製品の開発が可能となる。

1.1.2　要求定義

要求は「何をなすべきか」を定義するものである。

「どのようにして実現するか」は，システム設計以降で行う必要がある。

要求定義における大きな作業は，図１－２のとおりである。

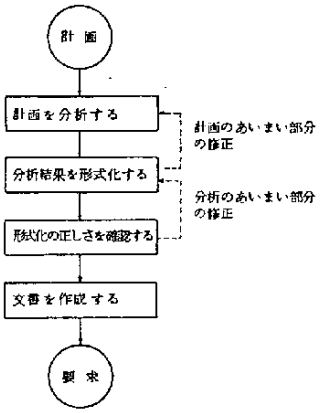


図１－２　要求定義の作業

なお，要求定義における打合せでは，何を行いたいのか討議を行い，"それは無理"，"それは大変"とかの言葉は，その要求を満足させるためのものではないので，慎むべきことである。

要求を文書にて伝えるためのシステム仕様書の記述要素としては，「機能」「入出力情報」「性能」「制約条件」「実時間性」「信頼性」「保守性」「拡張性」「準拠すべき規準」などがあり，これらの要素に対する明細が要求定義として，文書に記述されていることが必要である。

また，使用される名称等の用語は，一貫性をもって作成されなければならない。

(1)　機　　能

客先の要求（自社開発製品の場合は，開発目的）を体系化し，解かりやすくすること。

客先の要求と開発担当が実施しようとしていることを，機能項目の型で整理して盛り込むことが重要である。文章のみでは解りにくい場合が多いので，図等による表現を利用することが必要である。

(2) 入出力情報

マイコンと外部装置（例えば，プラント機器，大型計算機，周辺機器）と結合を行う場合は，いかなるインターフェイスにするかを決定しておかなければならない。これは，システムの性能，コスト，信頼性，保守性，安全性，工事や試験，および寿命等を決めるものであり，要求の時点で明確にする必要がある。

例えば，プラント機器から，1000点の入力信号が必要なシステムに対し，開発した製品が500点までしか処理出来ないものであれば何の役にも立たないのである。

(3) 性　　能

性能には，応答速度と精度があるが，客先の要求する性能は，いかなる範囲かを正しく検討する必要がある。

特に，使用者側と製作者側の一般常識または，業界常識とは，相違することがあるので，十分確認する必要がある。

(4) 制約条件

客先の要求による制約事項は必ず機能，性能等で明記する必要がある。しかし，システムを作る側の条件による制約事項を記すものではない。これは，あくまで，客先の要求を満足させるために必要な，客先からの制約条件にとどめるものである。

(5) 実時間性

機能，性能とも絡むが，リアルタイムで処理するシステムにおいては，絶対に必要な項目であり，システム全体に非常に大きな影響を持っている。すなわち，いかに外界の状態に追随して即座に動作出来るかである。

(6) 信 頼 性

信頼性の悪い製品を初めから作る人はいないわけだが，万一，何らかの原因でシステムが故障した時のことを配慮することが必要である。

例えば，「故障してもあまり影響の少ない，または復元できるシステム」とか，「故障した場合は，致命傷となるので，二重，三重にしたシステム」とか，客先の要求を満足させるシステムを作るために，明記することが必要である。

また，マイコンシステムに入力する信号，出力する信号等については，フィルタ，絶縁回路等を設けたり，チャタリング，誤動作をさける配慮を，実施すべきである。

(7) 保守性・拡張性

システムを作る時には，「システムにゆとり」を持たせることが必要であり，制限ぎりぎりの製品を作ると，改造，追加等が出来ないばかりでなく，故障，誤動作に対する対応も出来なくなる。

価格のこともあろうがゆとりを持ったシステムを開発することが求められる。

(8) 準拠すべき規準

各社各様の規準によって，システムに適合する手法，規準を採用することで良いと思うが，ややもすると，業界の動向，国際・国内の規準を無視している場合があり，客先，メーカ共に，規準に対する配慮が必要と思われる。

(9) 作成にあたっての注意事項

① 「何をするか」を中心に記述する。

② あいまいな表現はさけること。いずれにも受け取れる表現をすると，後でトラブルの原因となる。

③ 図，表等を利用して，理解が容易なものとし，ページ数が多くならない様にする。

④ 常識に注意する。客先と作成者の常識は必ずしも一致してない場合がある。

⑤ 質だけでなく，量も明確にする。

(10) 要求定義の手法

要求定義の手法は数多く発表されているが，代表的な手法としては，表1－2のとおりである。いずれの手法を用いるかは，システムの種類，規模により異なったり，客先とメーカによって異なったりで，統一した手法が在るわけでなく，統一すべきことでもない。

要は，“必要”の解決に必要なシステムの概要的なモデルをどのようなものにするかと言うことをいかに表現するかが重要である。あくまで，手法は利用するものである。手法にこだわっていては，表現がおろそかになるような場合は，解りやすい表現（一般的な文章，図等）で記述することが必要であろう。

表1－2　要求技術システム〔文献(1)による〕

システム名	思　想	開　発　者	特　　徴
SADT	C,D	Sof Tech	ある一定の作図法によって構造化させた図を作らせる。
HOS	C	Hamilton, Zeldin	ある規準に従って必要求トリー状に分解せしめる。
F^2 D^2	C	RCA	ある一定の作図法によって機能流れ図を作らせる。
ADS	C,D	NCR	一定の表様式を提供し，これに書込ませることによって必要を網羅する。
TAG	C,D	IBM	ADSと同様の思想に基づいている。
ISDOS	C,D,S	ミシガン大学	オブジェクトとその間を結ぶ関係言語によって必要を記述させ，計算機で処理する。汎用の仕様文書化ツールが目的。
CSC－Threods	C	CSC	外界の刺激に対応した機能の連鎖を定義する。機能は構造化して考える。
SREM	C,D,S	TRW	ISDOSの機能に図入力やシミュレーション機能を追加した大規模なもの

注：思想の欄のCはコントロール重点形，Dはデータ重点形，Sはその他の思想を含むもの。

1.1.3　システム設計

前に述べた要求定義は，「何をなすべきか」であったが，システム設計では，「どのようにして実現するか」について述べるものである。すなわち，システム設計は，要求定義をいかに実現するかを行う工程である。この工程で作成するソフトウェアシステム設計仕様書，ハードウェアシステム設計仕様書は，システム仕様書の機能・性能・信頼性・保守性等のほとんどを決定する立案設計であり，非常に重要である。また品質がこの工程で決まってしまう。

この工程でよく論議されるのが，ある機能について，ソフトウェアで処理するか，ハードウェアで処理するかである。この要因には，下記の項目があげられる。

① システム価格の制約から，ハードウェアの値段を下げるためにソフトウェアで処理する。

安価で大量製品に多くみられ，ソフトウェアの開発工程が増加し，開発コストは多くなる。

また，このような製品に限って，システムにゆとりが無く，職人的な作り方のソフトウェアとなっている場合が多い。

② ハードウェアの先行または，ハード屋の力加減で，ソフトウェアで処理する。

③ ソフトウェアは万能であると考えている人々によってソフトウェアで処理するように決定される場合。

基本的にハードウェアは，並列処理が可能であるが，ソフトウェアは，直列処理（見かけ上，並列処理に見えることがあるが，あくまで直列処理）のみであり，このことを間違えるとシステムの応答性等に大きな問題が発生することとなる。

④ 目先の工程に追われて，あいまいな判断でソフトウェアとハードウェアの担当を区分する。

等々，色々の要因はあるが，システム設計には十分な時間をかけて検討することと，それにも増して，既成システム（部品単位，モジュール単位でも良い）の流用を積極的に考えることが，開発の短縮であり，完全なシステムへの近道である。

(1) システム設計の活動

システム設計とは次の活動を意味している。

① 要求定義からの内容を，「働き」と「データ」に分けて，それを区分して抽象的定義を行うこと。

② 抽象的定義の内容の役割を明確に決定すること。

③ 具象化せねばならない対象が決まったら，その構造の定義を行うこと。

例えば，ファイル構成，伝送手段，入出力手順等。

④ 「働き」と「データ」の相互間，および外的要因に対する関係の定義を行うこと。

⑤ 要求定義により与えられた制約条件等が，合致しそうかどうかの検討を行うこと。

⑥ 検討のために，必要な解析，シミュレーションを実施すること。

⑦ ハードウェアとソフトウェアの分担を行うこと。一般的には，相当早い時期に，ハードウェアとソフトウェアの分担を行うことが多いが，要求を満足させるためには，全体の検討が済んだ後に，分担する方が良い。

⑧ 文書化すること。

各種手法を用いる。

⑨ 要求に対して満足しているかを見直すこと。

このようなシステム設計は，重要な活動であるにもかかわらず，従来ともすると，ソフトウェア担当者はただちに，プログラムのためのフローチャート等を書くという例が多く，ひいては，ソフトウェアの

品質が，よく問題になっていた。その原因は，ソフトウェアにきちんとした設計がなされてないということにあった。

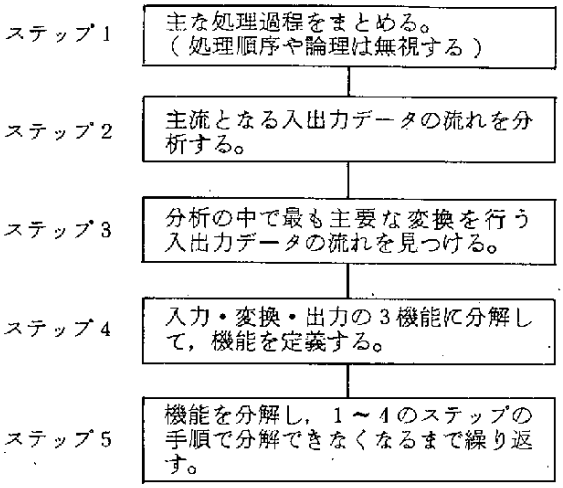


図1－3　構造化設計の分解過程

(2)　ソフトウェアシステム設計書の記載内容

①　ソフトウェアシステム構成

ソフトウェア全体のプログラムの構成とその関係について記載する。

②　ファイル構成

システムで必要とするファイルについて全体的な構成と，その関係および各ファイル毎のデータ内容，配列等の概要を記載する。（図1－4，図1－5参照）

③　機能別処理仕様

システム仕様書に記載された各機能がどのように処理されるかを，プログラム間の関連およびデータファイルの処理等を含めて記載する。（図1－6参照）

④　システムの拡張性

プログラム，ファイル，プロセス入出力などの拡張性について記

載する。

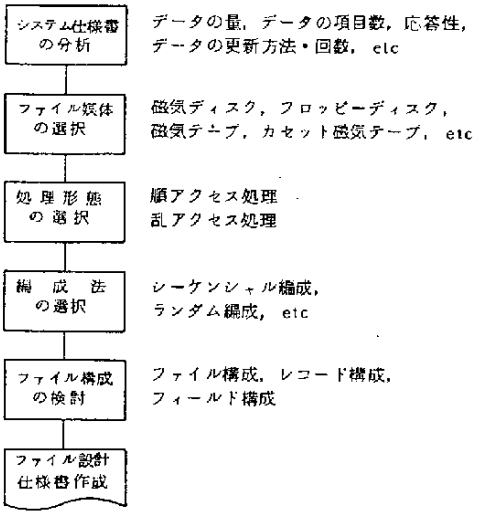


図１－４　ファイル設計の手順

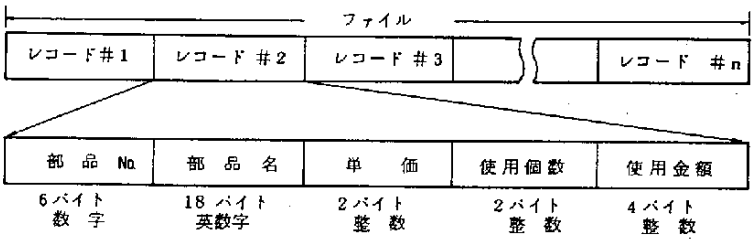


図１－５　ファイル内のデータ説明例

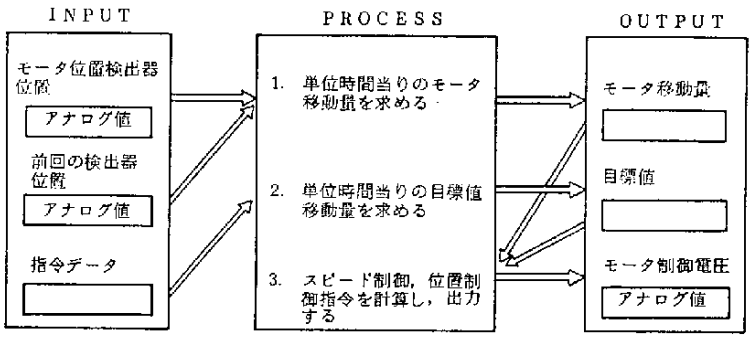


図１－６　HIPOによる概略記述例（IPO図）

その他には，言語の選定，メモリ構成，表示装置（例えばCRT等）等の表示フォーマット，およびＯＳの選定等々を記載する。

(3) ハードウェアシステム設計書の記載内容

① ハードウェアシステム構成

システム仕様書に記載されている内容を理解してもらうために，ハードウェアシステムの概要をブロック図を入れて記載する。

② 機器仕様

システムで使用している機器毎に，機器の仕様，タイミング，割り込み，およびI/O構成等を記載する。

③ 外部入出力のインターフェイス仕様

外部（例えば，伝送，プロセス入出等）装置とのインターフェイスを明確に記載する。

④ システムの拡張性

ハードウェア的な拡張性について記載する。

その他には，温度，湿度，対ノイズ性，電源容量，寸法，振動，衝撃等々を記載する。

1.1.4 プログラム設計

プログラム設計は，システム設計がなされたあと，その結果を文書化してプログラミング作業者へ伝達するためのものであり，各機能をより詳細に，かつ明確にモジュール化するものである。

モジュールとは，対象とするソフトウェアを実現するための体系を形づくるための基本単位要素である。

(1) 設計の展開

設計全体を展開していく上での手法は，トップダウン法とボトムアップ法の対称的な２つの方法がある。

以前の設計には，ボトムアップ法が多かったが，最近の設計には，

トップダウン法による設計に従うものが多く，トップダウン法は，階層型式のプログラム構造を前提としており，またこの構造に従って型式化される。

トップダウン設計は良い手法ではあるが，設計上の前工程において，極めて優秀な判断力と豊かな経験を必要とする。また，実積のある下位モジュールを再利用するためには，ある程度のボトムアップ設計も必要な時がある。

表１－３　トップダウン設計とボトムアップ設計の比較
〔文献(1)による〕

設　　問	トップダウン設計	ボトムアップ設計
作業開始時点から複数の人が同時に作業にかかれるか。	否 （１人のみ）	可
上位モジュールとのインターフェイスについて，統一がとれるか。	可	困　難
エラーの生じたときの追跡は楽か	可	困　難
設計上での難かしさは前工程にあるか，後工程にあるか	前工程	後工程
前工程の誤り，または拙劣の影響度は	大	小

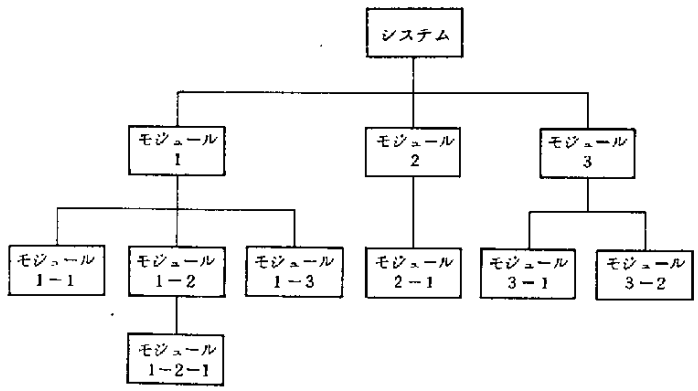


図１－７　モジュールの展開

トップダウン法を上手に使用することにより，プログラム設計の不

具合が早期に発見しやすくなる。

① モジュール構造で記述するため，システムの構造と，機能としての必要条件が明確となり，モジュール・インターフェイスに関する混乱を減らすことができる。

② モジュール独立性により，仕様上の抜けが発見しやすい。

③ モジュールの上位では，詳細な部分を前面に出さないため，全体の構成が解りやすく，また構造上の欠点がより明確となる。

④ 各細分化の前に，最高位レベルのモジュールのみでのテストが可能となる。

(2) ストラクチャード・プログラミング

いかなるプログラムでも，3種類の基本形式の組合せにより，正しく構築できることが証明されている。

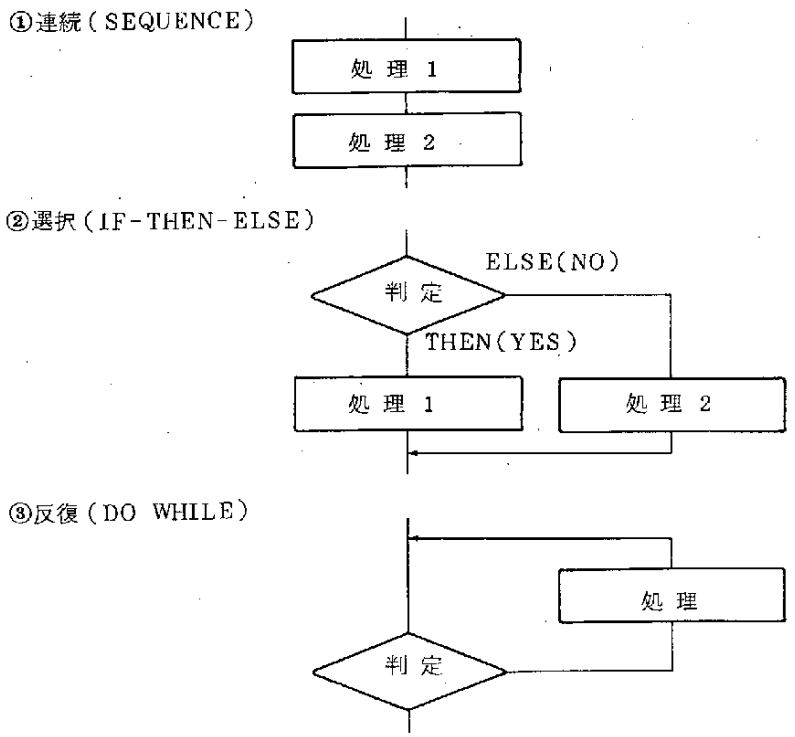


図1－8 プログラムのための基本3形式

図１－８の基本形式から拡張して使用する場合があるが，あまり拡張すると，プログラムの構造が見易という長所が失なわれることにもなりかねないが，うまく運用すると見易くなる場合があり，有効であることも見逃せない。

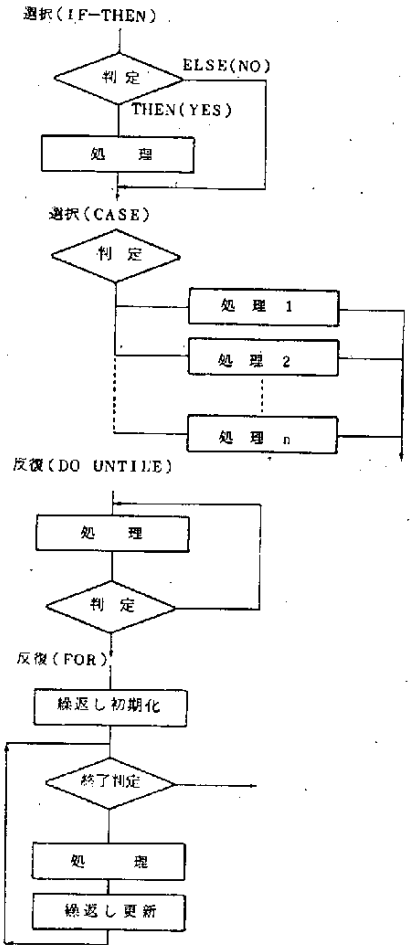


図１－９　拡張形式例

(3)　設計の記述手法

記述手法としては数多く提案されているが，ここでは，流れ図（フローチャート）とHIPO(Hierarchy Plus Input Process Output）について述べてみる。

(a) 流 れ 図

従来から長い間，広く使用されて来た最も一般化した手法である。最近では，流れ図の欠点も言われているが，良い面も数多くあり，特に，プログラム制御の流れが解りやすく，運用しだいでは，まだまだ活用される手法である。

① 流れ図は，自由に書けることから，表現があまりにも自由となりやすいので，ストラクチャード・プログラミングおよびトップダウン法の原則を極力守りながら作成する。

② データ定義およびモジュール間のインターフェイスを明確に記入する。

③ 流れ図記号は原則として，ＪＩＳ情報処理流れ図記号（ＪＩＳ－6270）によるものとする。

④ 流れ図の構成としては下記の項目を含める。

- 機能説明

  プログラムの概要を文章，図，デシジョンテーブル等で説明する。

- ストラクチャード・ダイアグラム

  モジュール間のつながりを，トップダウン法で記入する。

  （ＨＩＰＯの図式目次と同様なもの）

- 流れ図
- データ定義

  使用する入力，出力に対するデータの意味を定義する。

その他，ウォークスルー記録，変更記録等を入れる。

(b) ＨＩＰＯ

ＨＩＰＯは，機能を説明するもので，制御の流れを説明するものではないが，入力／処理／出力により，機能のインターフェイスが明確化しやすい長所がある。

HIPOは，概略レベルから詳細レベルに展開する時のトップダウン法の手段として用いることができる。

HIPOの代表的なパッケージによる構成例としては，下記の項目がある。

① 機能説明

システムまたはプログラムの概要を文章，図，デシジョンテーブル等で説明する。

② 図式目次

モジュールのつながりをトップダウン法で記入する。（図1－10にその例を示す）

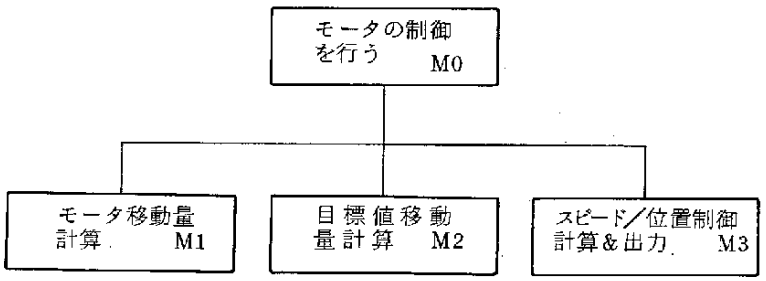


図1－10　図式目次例

③ 入力／処理／出力（IPO図）

上位レベルのモジュールから順に，機能を分割・詳細化していく。（図1－6にその例を示してある）

書き方の規則としては，

- キーワードを用いる。キーワードは，簡単な単語（英字）で設定し，それ以外は日常語を使用する。

キーワードとしては，図1－8に対応させて，例を記してみると，

〔連続〕

DO

処理1

処理2

ENDDO

〔選択〕

IF　判定

THEN　処理1

ELSE　処理2

ENDIF

〔反復〕

DO　WHILE　判定

処理

ENDDO

- 構造上の入れ子を記述する時は，字下げを行い見やすくする。

④ データ定義

使用する全てのデータ名，意味づけ，使われ方等を定義する。

その他，モジュールの強度，モジュール間の結合度，ウォークスルー記録，変更記録等がある。

1.1.5 プログラム製造

プログラム製造において担当する事項は，

- コーディング
- アセンブル／コンパイル
- リンケージ／ロケート

である。

(1) コーディング

プログラム設計からのドキュメント（流れ図／HIPO，サブルーチ

ン説明書等）により，指定言語のコーディングを行う。

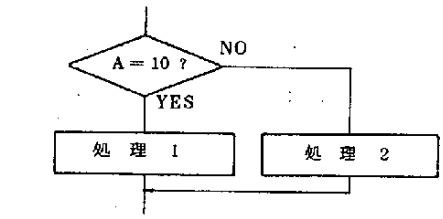
(a) コーディング規約の徹底

① ストラクチャード・プログラミング手法を使用する。極力コーディングは機械的に実施できるように，流れ図または，HIPO等に記述された記号（箱，キーワード等）と，アセンブラ言語または高級言語のコーディング方法の対応表を作成して，コーディングのルールを徹底するのも良い方法である。

② コメント（注釈）は，モジュールの機能説明，モジュールの表題，文の説明など，コメントでモジュールの内容が極力解かるように記入する。

コメント記入方法の説明書を作り，全員が同一ルールで記入することにより，統一された読みやすいリスティングができる。

③ アセンブラ言語の場合は，プログラム・レベルを統一すると，リスティングが解かりやすくなる。（例として，図1−11に示す）




```
IF△△△:
        CMP   A, 10
        JNE   IFELSE△△△
IFTHEN△△△:
      処 理 1
        JMP         ENDIF△△△
IFELSE△△△:
      処 理 2
ENDIF△△△
```

△△△……モジュールの先頭から，入れ子を考慮して，連続番号を入れる。（後からの追加等を考慮して，適当な間隔（10飛び等）とする。）

図1−11　プログラムレベル記入例

(b)　ハードウェア知識の吸収

ハードウェアと密接な関係にあるプログラム（割り込み処理，タイマー処理，伝送処理，入出力処理など）は，必ず，プログラム設計ドキュメントに記入し，プログラム製造に伝達することであるが，コーディング時には，タイミング等を，十分理解して対応すべきである。誤まると，非常に見つけにくい所である。

(2)　アセンブル／コンパイル

アセンブル／コンパイルの方法は，解かつてしまうと簡単なものであるが，方法を誤ると，目的を脱したオブジェクトが生成されることがあるので，十分注意すること。

また，ソースに誤りがあると，編集（エディット）を行って修正するが，誤まってソースを破壊したり，開発システムの誤動作で，ソースが破壊されたりすることは日常よく聞く話しである。このような時の対策のために，ソースは二重，三重に，バックアップを持つことを勧める。

(3)　リンケージ／ロケート

各モジュールのオブジェクトを機能単位程度のグループに寄せ集め（リンケージ），それを物理アドレスに割り付ける作業（ロケート）を行う。

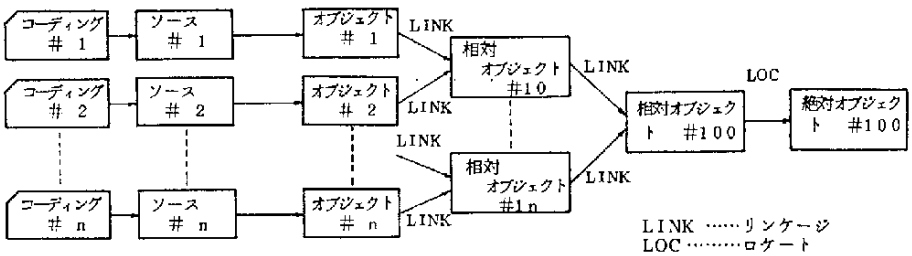


図１－12　コーディング～ロケート流れ

1.1.6 単体テスト

ハードウェア，ソフトウェア共に開発の場合は，プログラムの単体テスト時に，ハードウェアのトラブルが含まれていることがあり，ハードウェアが悪いか，それともソフトウェアが悪いかで，日常，非常に悩まされることがある。ひいては，工程が遅れる原因となることも度々ある。

ハードウェアの単体テスト時には，タイミング，ノイズ等のテストは勿論であるが，十分なハードウェア・テストプログラムを作成し，それを用いて試験することが必要である。

ソフトウェアの単体テストは，プログラム製造後に最初に実施する試験（テスト）であり，個々のモジュール単位の機能について，確認を行うものである。

（一般的にデバッグと言われているが，デバッグとは，プログラムに潜む“虫”を取り除く作業の総称であり，主として試験の結果発生し得る作業といえるので，本章では，工程上の用語としては，デバッグとはせずに，テストとした。）

ソフトウェアの単体テストには，大別して，机上テストと実機テストがある。（実機を使用しないで，シミュレートテスト等もあるが，ここでは解説は省くこととする。）

(1) 机上テスト

アセンブル／コンパイルの文法的な誤りが無くなると，リスティングとプログラム設計ドキュメントとを照合し，誤りを取り除く。その後，実機テストの順備として，模擬入力データを作り，そのデータに従って処理の途中経過および出力データをリスティング上に当てはめて，その経緯を実機テストで使用できるようにテストデータとして作成する。

この机上テストは非常に重要であり，リスティングを追いながら，設計ミス，コーディングミスを早期に発見できる。ややもすると，実

機で実行させてみて，実機の前で考えている人がいるが，整然とした考えも出来なくなるし，デバッグ時間の浪費でもあるので，絶対避けるべきことである。

机上チェックの前に，プログラム設計者などの上位工程担当者が，プログラムの見やすさ，解りやすさ，誤り等のクロスチェックを行うことが誤りを未然に防止する上で必要なことと言える。

机上テストで特に注意すべき主なチェック項目を述べると下記の項目がある。

① クロスチェックの実施。

② 繰返し処理等の繰返し回数や，値のリミットチェック処理においての判定条件等のチェック。

よくあるミスは，繰返しで，最後の1回が無視されたり，余分に1回ループしたりというケースがある。また，等号(=)を含めるか否かのミスも，よくあるケースである。

③ サブルーチン・コール時のコーリング・シーケンスおよび，パラメータのチェック。

④ 共通エリアおよびファイル使用時等のインタロック（排他処理）は正しいかのチェック。

⑤ ハードウェアの使用法（タイミング等）は正しいかのチェック。

⑥ 数値計算等で，誤差は含まれていないか，または誤差範囲に入っているかのチェック。

⑦ 処理方法で，無駄な方法をとってないかのチェック。

⑧ オペレータ操作機能等で操作ミスに対するエラー処理は正しいかのチェック。

⑨ 勿論，システム仕様書を満足しているかのチェックは言うまでもない。

(2) 実機テスト

実機テストは，机上テストでチェックした内容を踏えて，機能が満足されていることを確認する作業である。

実機を使用して単体テストを行う場合に，必ずしも全てのハードウェアが整っていない場合も多く，この様な時は，ハードウェア的なシミュレータまたはソフトウェア的に見せかけのプログラムを作ってシミュレートすることが必要である。

(3) 連絡メモ

単体テストに限らず，全ての工程において必要であるが，開発を円滑に推進するために，各担当者間の技術的連絡事項を文書化して徹底を図る。

① 設計メモ

設計時に発行するものであるが，テスト段階においてもプロジェクト内の徹底事項，情報などを伝えるのに用いる。

② 問題点メモ

テスト以降で発生するプログラム上およびシステム上の問題点を記入し，解決されたものは，その原因，内容を記入して完了する。

③ 変更メモ

プログラムの中に発見された誤りを修正する場合に記入し，設計者または責任者等の許可を受けて，プログラムの修正を行う。

すなわち，システム全体を見極めている上位工程者が，指示または確認することにより，誤りを早期に解決することができる。

1.1.7 組合せテスト

組合せテストとは，単体テスト完了後に，いくつかのプログラムおよびハードウェアを組合せて，システム上の独立した機能について，確認を行う作業である。

組合せテストは，全てのモジュールの単体テストが終了してから開始

することは実質上少ない。すなわち，単体テストの終了したモジュールを利用することにより，他のモジュールの単体テストの能率が上がることが多いからである。

組合せテストの手法は，下記の項目がある。

(1) ボトムアップテスト法

最下位のモジュールから順次最上位のモジュールに移ってゆく方法。

(2) トップダウンテスト法

トップダウンの設計を行っている過程中でも，下位モジュールは適当なシミュレート・ルーチンを入れることにより，確認できる。

この方法で単体テストの時から実施すると，単体テストが全て完了した時点で，組合せテストも完了していることとなる。

(3) 一斉テスト法

全てのプログラムを一括して，外部よりデータを入れてテストする方法。モジュールの単体テストの品質が良い場合は効果的であるが，単体テストの品質が悪い場合は，どうしようもなくなることが発生する。

いずれの手法を用いるかは，1つのシステムの中で使い分けると効率の良い場合が多い。例えば，共通的に使用するモジュールは，下位レベルのモジュールであっても，ボトムアップテストとし，上位のものはトップダウンテストとして，残りは，トップダウンテストおよび一斉テストで進める。

要は，システムの中のモジュールの重要性とか，他のモジュールとの結合の強さによって，最も良いと思われる手法を採用することである。

1.1.8 総合テスト

総合テストとは，ソフトウェア，ハードウェアを含めて，最終段階の試験であり，単体機能として確認の完了した全ての機能を，総合的に確認

する作業である。

テスト項目の主なものは，以下のものがある。

(1) ハードウェア

- 温度試験
- ノイズ試験
- 電源試験
- 絶縁抵抗試験
- 連続動作試験
- タイミング試験
- マージン試験
- 外部入出力（伝送，プロセス入出力等）の試験

その他，システム仕様書に基づいた，全ての動作試験

(2) ソフトウェア

- 異常時の処理試験
- 意地悪試験
- タイミング試験
- 電源投入処理（イニシャライズ）試験
- 操作性試験

その他，システム仕様書に基づいた，全ての動作試験。

全ての試験が終了したことにより，製品認定が行われる。製品認定時は，全てのドキュメントが完備し，問題点の残件が無いことを条件として認定することが望ましい。ドキュメントは後から修正すれば良いと考えていると，いつまで経つても修正しないし，また，修正しないまま出荷すると，後日の問題発生および仕様変更等に対応できなくなる。

1.1.9 製品化・量産化

全ての試験を完了し，製品として認定されると，製造ライン，試験ラ

インの設備を完備し，市場に出荷される。

製品化においては，製品のデザイン，寸法等の見直しを行い，再度，総合テストを実施後，問題点が無ければ量産化される。

参考文献

(1) 松本：ソフトウェアの考え方・作り方，電気書院

(2) マイクロコンピュータ応用技術調査専門委員会：産業用マイクロコンピュータの応用技術，電気学会

1.2 開発支援ツール

1.2.1 ツールの概要

マイクロプロセッサを利用した応用製品は，今後ますます増加する傾向にある。そこで要求される機能も，8ビットから16ビット,更に将来は32ビットへとプロセッサの性能が向上するにつれ,複雑化している。

このような状況では，応用システムの開発は，最早従来のように，オシロスコープ一つで立ち向うことはできない。その中に組込まれるソフトウェアにしても，単にハードウェアの機能を置換えるのではなく，より高度なデータ処理，機器制御を要求される。

ハードウェアのテストは，まずオシロスコープにより，電気的な機能のチェックから始まるであろう。オシロスコープでは，個々の信号の電気的特性，即ちパルス波高，立上り時間，立下り時間，パルス幅，リンギングなどが調べられる。更に，多チャンネルのオシロスコープを使えば，いくつかの信号の相互のタイミングも調べられる。

しかし，オシロスコープによる観察には，いくつかの制限がある。第1に，原則として，オシロスコープは繰返し波形の観察には便利であるが，マイクロプロセッサの動作中のバスラインの信号のように，多様な変化を示すものには不向きである。第2には，多チャンネル・オシロスコープといえども，一般的には4チャンネル程度で，プロセッサのデータバスの全ての変化を観察したり，他の信号との関係を調べるのは，非常に困難である。

このような制約だけからも，マイクロプロセッサを応用したシステムの開発には，新しい測定器が必要になる。その測定器には次のような機能が要求される。

第1に，十分な入力信号数が必要である。16ビットのプロセッサをみると，アドレスバスに16～20本，データバスに16本，その他の信号も含めると,40本以上の信号が相互に関係して動作している。同時に観

測できる信号の必要数は増加する一方である。

第2に，トリガ機能の充実が必要である。

プロセッサの複雑な動作シーケンスの中から，必要なタイミングを選択できることが重要である。

第3に分かり易い表示方法が用意されなければならない。多数の信号を単に画面に並行に表示したのでは，十分とはいえない。データは測定器内のメモリに，観測時刻の順に記憶されているであろうが，これを時間軸について拡大，縮小表示を行ったり，表示信号を任意に選択したりする必要がある。

データの記憶とトリガ機能によって，オシロスコープではできない，トリガ条件の発生以前のデータも観測できる必要がある。

このような各種の要求に答えるべく開発されたのが，ロジック・アナライザといわれる測定器である。ロジック・アナライザは，マイコン応用システムにおいては，開発から保守まで最も良く使われる測定器であろう。

ロジック・アナライザには，特定のプロセッサの仕様を組込んでいて，バスの状態から，実行した命令のオペコードやオペランドを表示するものもある。

このようにロジック・アナライザは，単なるハードウェアのテスト用のみでなく，ソフトウェアの実行状況を観測することもできる。

しかし，ロジック・アナライサはあくまで，ハードウェア主体のテスト機器であり，ソフトウェアの開発には不十分である。

ソフトウェア開発用のツールとしては，インサーキット型エミュレータが中心である。

インサーキット型エミュレータの目的は，開発中の実機上での，ソフトウェアのデバッグである。

エミュレータには次の機能が要求される。

第1にプログラムの実時間での実行が必要である。マイコン応用システムの開発では，外部機器との間の入出力制御が大きな割合を占める。機械的動作を伴うものでは，実時間動作は欠かせない。

第2は，多様なプログラム・ブレークのサポートである。ブレークの条件は，実行アドレス，データ，プロセッサ・ステータスなどのCPUの状態の他に，外部入力等による制御も必要である。ブレーク・ポイントも，単一アドレスでなく，メモリ領域でも指定できれば便利である。

第3は，トレース機能である。トレースは，実行中のプロセッサの状態を，トレース用のメモリに記録し，実行がブレーク後にそれまでの実行履歴を調べるためのものである。トレース機能によって，不測の分岐などで，プログラムが暴走した時も，その原因を追求することができる。

第4は，代行RAMの使用である。マイコン応用システムの大きな用途に，組込型の制御システムがある。このようなシステムでは，プログラムはROMに記録されている。しかし，ソフトウェアのデバッグ中は，一時的にプログラムの内容を変更したりする必要もあり，ROMは不便である。実機上のROMに対するアクセスを，エミュレータ中のRAMに対するアクセスに置き換えてしまうのが，代行メモリの利用である。代行メモリは，プロセッサのアクセスできるメモリ空間の任意のアドレスに割付けれるように，メモリ管理ユニットにより，アドレス変換を使用し，更に，プログラムの暴走によるメモリ内容の破壊を防ぐために，メモリ保護機能を必要とする。

その他にエミュレータに望まれる機能としては，シングル・ステップ実行，プログラムのロードとダンプ，外部へモニタ信号の出力，外部制御信号の入力，等がある。

このようにエミュレータは，ソフトウェアのデバック・ツールであり，ハードウェアのテスト用のロジックアナライザと相補って，システム開発の強力なツールとなる。

次にソフトウェアの開発手順を考えてみよう。ソフトウェアの開発は，次のような流れで行われる。

① ソース・プログラムの作成

② コンパイラおよびアセンブラによるオブジェクトの作成

③ リンカによるロード・モジュールの作成

④ デバッグ

⑤ 不具合を修正し，再び②より繰返す。

これらの作業の中で使われるツールとしては，エディタ，コンパイラ，アセンブラ，リンカ，デバッガ等がある。

エディタは，ソース・プログラムの作成，修正に用いられ，頻繁に使用されるツールである。エディタには，行単位に編集を行うライン・エディタと，画面上に表示された，複数行のソース・プログラムの任意の場所に編集を加えられるスクリーン・エディタの2種類がある。

一般にライン・エディタは，コマンドの種類も少なく，操作用の端末の種類も特定されず，簡便であるが，編集機能は低い。他方スクリーン・エディタは，編集機能も高く，慣れれば作業効率も良いが，特定の端末を必要としたり，コマンドが多く，習熟に時間がかかることがある。しかし，エディタの使用頻度の多さから，一般にスクリーン・エディタが好まれる。

コンパイラおよびアセンブラは，ソフトウェアの開発環境によって，いくつかの注意が必要である。

第1に，開発用システムと，実機のCPUが同一のものであるかどうかによる。同じであれば，開発用システムの言語処理系がそのまま使用できるであろうが，異なった時は，クロス・コンパイラ，クロス・アセンブラを必要とする。

第2は，一般にマイコン応用システムでは，プログラムはROM上で実行されることが多い。このために，使用するコンパイラ等の生成するオ

プジェクト・モジュールがROM化が可能である必要がある。

第3は，プログラムの実行時のランタイム・ルーチンの必要性である。一般には開発用システムと実機では，オペレーティング・システムが異なるので，必要な時は，実機上でのランタイム・ルーチンを利用できるものでなければならない。

プログラムのデバッグは，3通りの方法がある。まず開発用システム上でのデバッグである。実機と開発用システムが異なる時は，この段階のデバッグはあまり重要ではない。特に外部機器との入出力は，凝似デバイス等を用いるしかなく，主にデータの扱いのチェックのように，一般のプログラムのデバッグと同様である。

実機上のデバッグは，前述のエミュレータによる方法と，目的のソフトウェアに実機上で動作するデバッガをリンクして実行する方法と，実機上にモニタ・プログラムを用意して，それでデバッグする方法がある。後の2方法では，実機上にコンソール・ターミナルが必要であり，ハードウェアの設計時から考慮して置かねばならない。

以下に，これらのツールについて，もう少し詳しく述べよう。

1.2.2　ロジック・アナライザ

ロジック・アナライザとは,図1－13に示されるように，何らかの同期信号（クロック）によって，その瞬間のデータをサンプルし，記憶，表示するものである。クロックとして，観測する信号と同期したクロックを用いるロジック・ステート・アナライザと，信号系とは独立の非同期クロックを用いるロジック・タイミング・アナライザがある。

ロジック・ステート・アナライザは，例えば，マイクロプロセッサのクロックを用いて，データバス，アドレスバス等を観察するような用途で，主にCPUを中心とした作業に用いられる。

ロジック・タイミング・アナライザは，観測する信号系を，独立のク

ロックでサンプルするもので，観測する信号の変化を，多チャンネル・オシロスコープによる波形観測のように表示する。ロジック・タイミング・アナライザがオシロスコープと異なるのは，表示される波形は，電気信号そのものではなく，クロックでサンプルされた瞬間における，論理的な1又は0の状態を表示していることである。

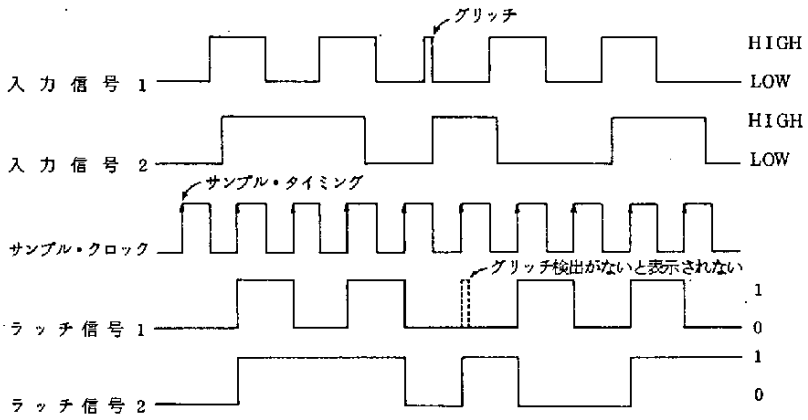


図1－13　ロジック・アナライザの信号のサンプリング

ロジック・アナライザの基本的な構成は，図1－14に示される。入力信号は，サンプラによって，サンプル・クロックのタイミングで論理レベルの判断が行われる。入力信号の論理レベルの決定は，一定のスレッショホルド電圧によって行われるが，入力信号の有限の立上り，立下り時間のため，ターゲット・システム内での論理レベルの確立と，ロジック・アナライザの観測にくい違いの生じることがある。これを防ぐには，ロジック・アナライザのスレッショホルド電圧を，立上りエッジと立下りエッジの両方に独立に設定できることが望ましい。特にこの要求は，高速のロジック・タイミング・アナライザにとって必須であろう。

サンプリング周波数は，観測する信号の帯域によって決められるが，ロジック・ステート・アナライザでは，通常のプロセッサのクロック周波数が10MHz 程度であるので，問題はないが，タイミング・アナライ

ザは，使用環境により大幅に異なる。クロック周波数が高くなれば，信号のタイミングは正確に知ることができる反面，その結果を記録するメモリは有限であるので，一度に観測できる時間幅は小さくなる。

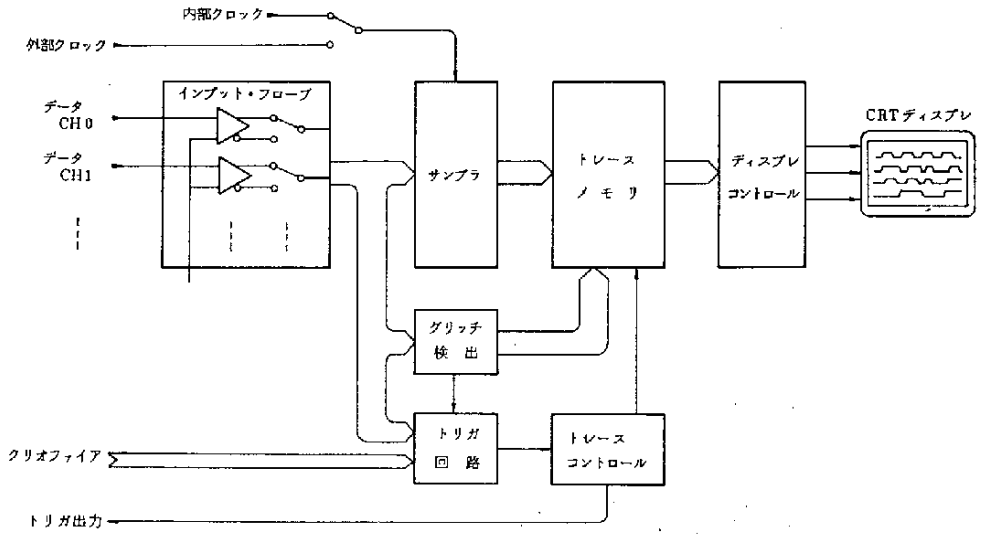


図１－14　ロジック・アナライザの概念図

時間分解能を決めるものとしては，サンプル・クロックの他に，プローブによる遅延時間の差，前述のスレッショホルドによる誤差などがあり，10ns 以下のタイミングでは，ロジック・アナライザの観測には十分な注意が必要である。

サンプルされた信号は，論理的な１又は０として，メモリに記入される。メモリの内容は，観測中は，サンプル・クロック毎に，シフトされ，オーバフローしたものは古いものから順に失われる。メモリの内容は，トレースを終了した時に固定され，読み出すことができる。

表示方法としては，表１－４に示すように，いくつかの型式がある。タイミング表示は図１－15に示されるように,入力信号の状態を，時間経過に従って示すもので，タイミング，アナライザの基本機能である。

ステート表示には,図１－16のような，タイミング・アナライザの観測結果を，２進表示，８進表示，または16進表示で行うものと,マイクロプロセッサの動作を解折して，各バスステートでのデータ，アドレス等から，実行した命令シーケンスを表示する図1－17のようなものまである。マイクロプロセッサをサポートするものは，後述のインサーキット・エミュレータと組合わされることが多い。

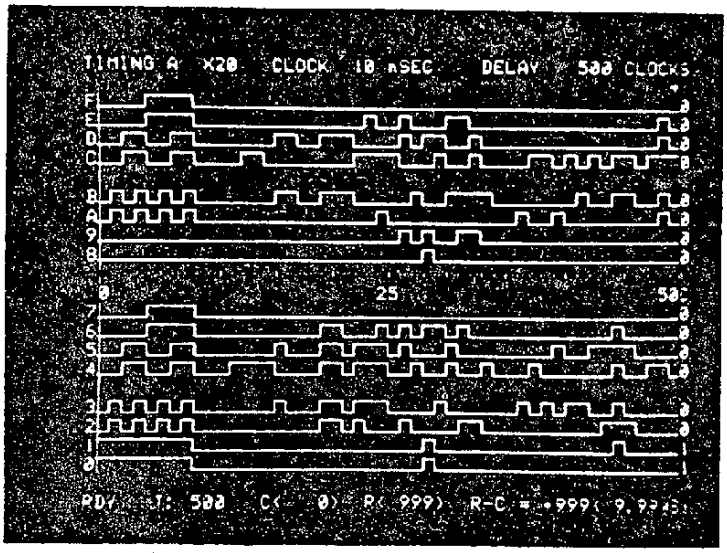


図１－15　タイミング・アナライザの基本表示

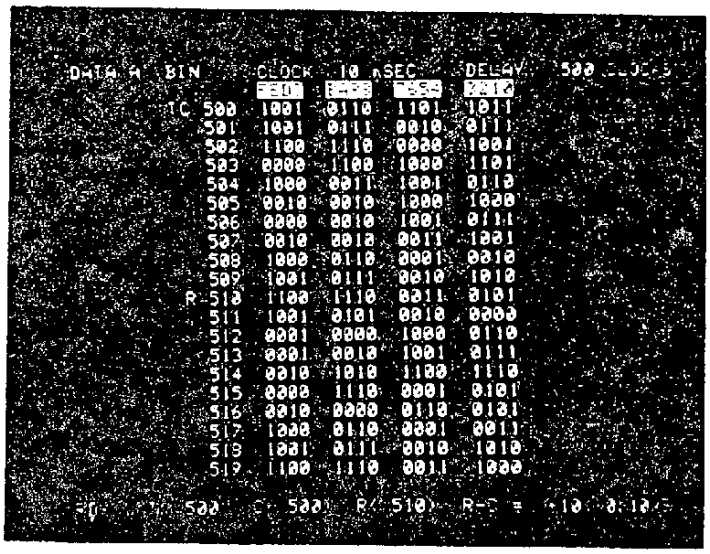


図１－16　タイミング・アナライザのステート表示（２進数）

```
 TGCNT=00001 MCY=00000 TRQFY=JUMP  TS=S
 EXT CK=SYNC  TRACE CK=QUE CPU CK=EXT
[    CPU STOP    ]  EXC TIME=001.499 mS
DISP#=1005        -- Y BUS STATUS-- EML=N

 ADRS DATA         MNEMONIC
000B0 071A           MW    WORD
FF114 E8F205        CALL F709  NEAR
000B8 0117           MW    WORD
FF716 FF160600      CALL W 0006
FF006 0088           MR    WORD
000B0 071A           MW    WORD
FF126 EB10          JMP F138  SHORT
FF138 E86002        CALL F39B  NEAR
000BE 013B           MW    WORD
FF3A4 74F8          JE/JZ F39E
 STOP
STEP 00
```

図１－17　ステート・アナライザによる逆アセンブル表示

表１－４　表示機能

タイミング表示	タイミング・アナライザの基本表示法である。多チャンネル・オシロスコープと同じように，横軸に時間を取り，信号の変化を線で表示する。信号は電圧でなく，論理状態の０または１で表わされる。 入力信号のグリッチは，この表示の中で，１クロック幅のパスルまたは，１クロックより狭い線で表示する。輝度変調などで，識別できるようになっているものもある。
ステート表示	ステート・アナライザの基本表示法である。各サンプル時のステートを，２進，８進，１０進または１６進の数として表示する。 マイクロプロセッサ対応のアナライザでは，逆アセンブル機能により，実行した命令のニモニック表示を行うものが多い。
グラフ表示	データの各ビットに重みをつけ，時間軸に対して，データの変化をグラフで表示する。
マップ表示	表示のＸ軸とＹ軸に，ビット毎に重みをつけたデータを対応させ，それらの相関を，画面上での輝点の位置の移動で示す。
パフォーマンス表示	ステート・アナライザの機能の一つとして，特定の条件で実行時間を測定し，それを棒グラフで，実行時間の比率等で表示する。
複　合　表　示	ステート・アナライザにおいて，ステート表示と，バス・タイミング表示を同一画面に表示する。

ロジック・アナライザに要求される主な機能について述べる。

タイミング・アナライザで，重要な機能にグリッチの検出がある。

ロジック・アナライザの入力のサンプリングは，図１－13に示されるように，サンプル・クロックの入力時の状態をラッチしている。サンプル・クロックの間隔よりも短いパルス入力（グリッチ）は無視される。グリッチを正確にモニタするために，サンプル・クロックの周波数を上げることは，より高速のアナライザに，大容量のメモリを必要とし，実用的でない。

グリッチ検出機能は，高速なアナライザや大容量メモリなしに，このような狭いパルス入力を検出する。図１－18が，グリッチ検出の動作である。狭いパルス入力は，アナライザの内部で，ラッチされる。次のサンプル・タイミングで，その信号のステートが変化していなければ，ラ

ッチされたパルスはグリッチであるとみなされる。グリッチの表示は，一般には１クロック分の幅よりも狭い信号のように表示され，輝度変調などで，容易に識別できるようにする。

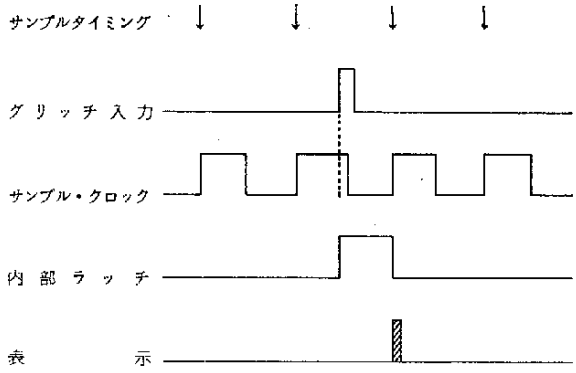


図１－18　グリッチ検出の動作例

次にトリガ機能について述べる。トリガは，観測するデータを選択する重要な機能である。トリガ機能については，一般に表１－５に示されるような種類がある。

表１－５　トリガ機能

トリガ機能	概　　要
ワード・トリガ	最も基本的なトリガ機能である。入力データが指定されたパターンでトリガする。パターンの指定には，ドント・ケアを用いて，トリガ条件に幅を持たせることができる。パターンの発生回数を制御できるものもある。
グリッチ・トリガ	指定した入力信号におけるグリッチの発生を，上記のワード・トリガ条件に加える。 グリッチとは，サンプル・クロックの１周期よりも狭いパルス入力である。
外部トリガ	外部からの入力信号でトリガを行う。
シーケンシャル・トリガ	ワード・トリガを組合わせて，それが指定された順に発生した時に，トリガを行う。 プログラムの複数の実行パスから特定のパスを選択できる。
プレトリガ ポストトリガ ディレイドトリガ	トリガが発生した時点の前または後のデータをトレースする。 プレトリガは，トリガ発生以前の状態，ポストトリガはトリガ発生後の状態を記録する。 ディレイドトリガはトリガ発生からトレースを開始するまでの時間を遅らせる。

表１－５　トリガ機能（つづき）

トリガ機能	概　　要
アーミング エネーブル ディセイブル	ワードトリガと同様に条件設定を行い，トリガの発生を制御する。 外部信号またはクロックをクオリファイアとして使用できる。 アーミングまたはエネーブル後のトリガ発生で，トレースを開始する。 ディセイブルは，トリガの検出を禁止する。 エネーブルとディセイブルの組合せにより，特定の実行範囲を指定し，トレースすることができる。
データ・コンペア	既に取り込んだ標準データと入力データを比較し，一致又は不一致によってトレースをコントロールする。 問欠現象の解析に有効な手段である。

ソフトウェアを含めた，システム・デバッグにおいては，シーケンシャルトリガが非常に重要である。シーケンシャルトリガは，図１－１９に示すように，実行シーケンスがいくつもの分岐を持つ場合に，その中の特定のパスを取り出すことができる。

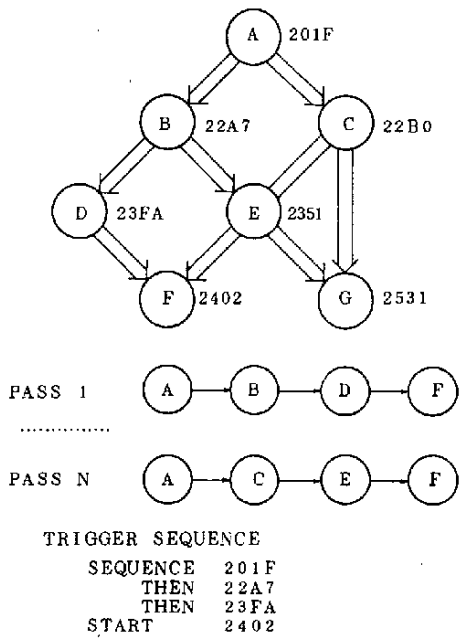


図１－19　シーケンシャル・トリガによる実行パスの選択

図の例では，名分岐点に示されるステートをトリガ条件とし，それを図中に示されるように指定することで，パス１のみを取り出せる。

ステート・アナライザとタイミング・アナライザの同時測定も必要な場合が多い。

ソフトウェアの動作を中心とし，特定の実行シーケンスにおける信号の流れをみるには，ステート・アナライザにより，タイミング・アナライザをスタートする。

逆に，特定入出力時のバグの解明などには，タイミング・アナライザにより，ステート・アナライザをスタートさせ，プログラムの応答を調べる。

一般にはタイミング・アナライザとステート・アナライザは，一方がトリガ発生を行い，他方がトレースを行うものが多いが，最近では，同両同時にトレースを行い，それぞれの時間軸の相関のとれるものも発売されている。

1.2.3　インサーキット・エミュレータ

マイクロプロセッサの性能の向上と共に，マイコン応用システムの開発における，ソフトウェアの比重は，ますます大きくなっている。特にマイコン応用システムの特徴として，リアルタイムでのデバッグの能率化が大きな問題である。

インサーキット・エミュレータは，こうしたシステムの開発用ツールである。

マイコン応用システムのリアルタイム・デバックを行う上での困難は，次のような環境にある。

① 一般に組込システムなどでは，デバック用の入出力システム（コンソール）は用意されていない。

② 応用システムは，専用プログラムを使用することが多く，汎用オペレーティング・システムなどが組込まれない。

③　プログラムは，ROMに記録して使用するので，デバッグ時に一時的な変更などが困難である。

④　プログラム中に，チェック・ルーチン等を挿入すると，実行時間が異なってしまう。

⑤　組込まれる記憶容量の制限のため，デバッガなどを，目的のソフトウェアとリンクして書き込むことができない。

このような状況の下で，ハードウェアと密着したソフトウェアのリアルタイム・デバッグを，能率良く行うためのツールが，インサーキット・エミュレータである。

インサーキット・エミュレータは，図1－20に示されるように，エミュレーションCPU，ROM代行用のエミュレーション・メモリ，エミュレーションCPUの動作を観測するためのトレース・メモリ，実行制御用のブレーク・ポイントやトリガ機構などで構成されている。

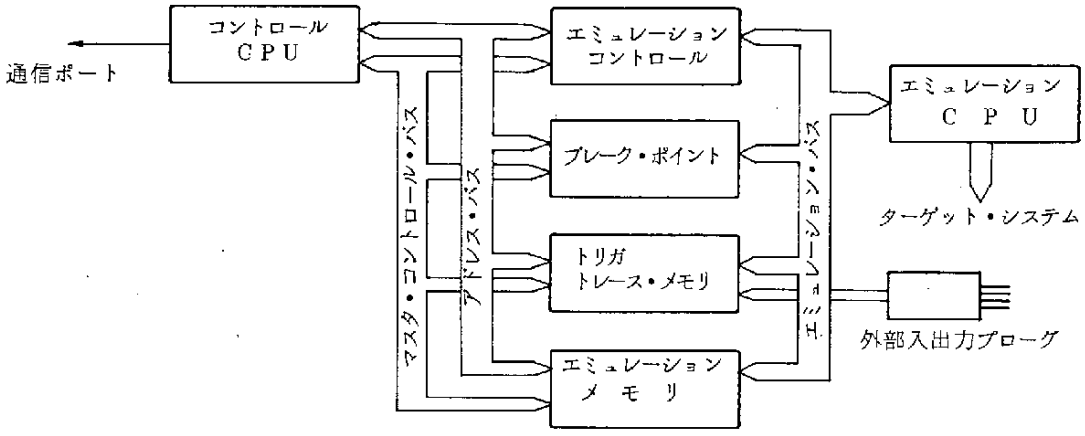


図1－20　インサーキット・エミュレータの概念図

開発中のターゲット・システムのCPUは，エミュレータのCPUプローブを，ターゲット機のCPUソケットに挿し込むことによって，エミュレーションCPUによって代行される。

エミュレーション・メモリは，ターゲット・システム中のROM の任

意のアドレス部分を代行する。代行メモリは，1KB程度の単位で，ターゲット・マシンのROMの配置に応じて割当てられる。

トレース・メモリは，実行中のCPUの状態（ステータス，データ，アドレス等）を，CPUに同期したサイクルで記録する。ロジック・ステート・アナライザにおけるプロセッサの実行シーケンスのトレースと同様の機能である。

ブレーク・ポイントは，ソフトウェア・デバッグ時の，実行制御の手段である。ブレーク・ポイントの拡張として，一定の領域を指定し，その範囲内の全てに対して，実行時ブレークが発生するような方式もある。

トリガは，ロジック・アナライザと同様に，記録のスタート，ストップ制御，実行時間の測定のオン／オフなどに用いる。トリガの発生条件は，アドレスとそこのアクセス方式，割込，入出力，データ等の組合わせで指定できる。

エミュレータの構成は図1－21から図1－23に示されるように，いくつかの変遷がある。図1－21は，開発システムの中に，エミュレーション制御インターフェイスを組込んだものである。エミュレーション制御インターフェイスは，エミュレーションCPUと開発システムへのバスインターフェイスが基本である。代行メモリは，開発システム中のメモリを使用するものが多い。エミュレーションCPUと開発システムのCPUを1個で実行するものもある。

このような組込み型のエミュレータは，インターフェイス自身の価格は安いが，開発システムと一体でしか使用できないので，可搬性には欠ける。更に，開発システムのバスと直結しているので，ターゲット・システムのプロセッサの種類に制限が多い。

しかし，開発システムと一体であることは，プログラム作製から，リアルタイム・デバッグまで一貫して行え，エディット，コンパイル，デバッグの間の移行はスムーズである。

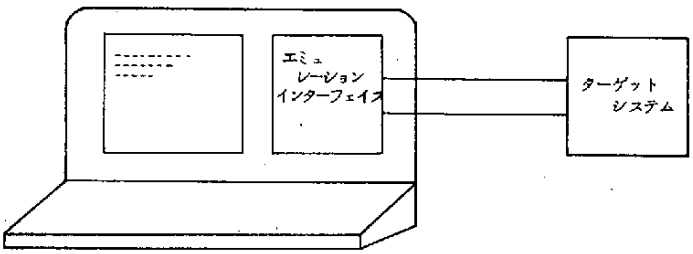


図1－21　汎用マイコン・システム十エミュレーション・インターフェイス

小型，軽量で可搬性に豊み，現場での調整，保守にも便利なツールをという要求は，図1－22のようなエミュレータを生みだした。いわゆるスタンドアローン型のエミュレータである。このタイプのものは，専用プログラムを内蔵し，汎用I/Oや，ディスク等は接続されない。デバッグすべきプログラムは，他の開発システムで作製し，通信手段で読み込む。

スタンドアローン型は，一般に小型・軽量のため，現場作業にも便利である。ホスト・システムとの間は通信により接続されるだけであるので，ホストの機種にとらわれず，各種のプロセッサに対応した製品が多くある。

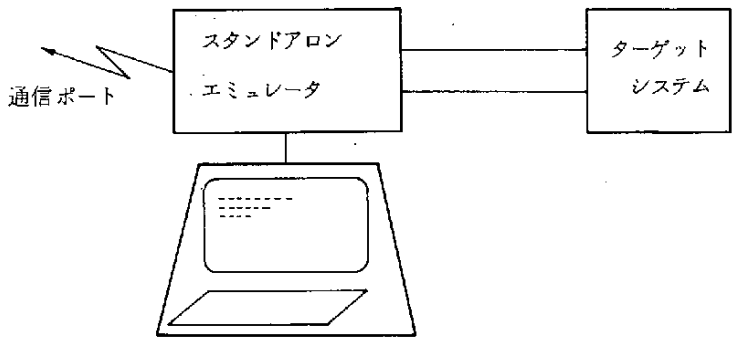


図１－22　スタンドアローン・エミュレータ

しかし，プログラムのローディン時間が，通信速度による制限のため，比較的長くかかることや，デバック機能が，ホストの言語処理プログラムによって制限されることなどから，再び図１－23のように開発システ

ムを付加した型式のものが表れている。この型式は，最初の構成（図1－21）と結果的にほ同一であるが，搭載しているシステムはあくまでエミュレータをサポートするツールを中心に構成されている。更に形状も，可搬性を重視した設計が多い。

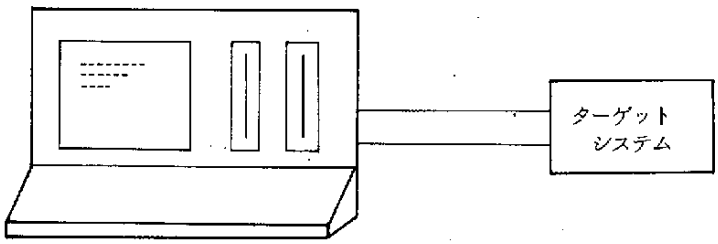


図1－23　エミュレータ＋開発システム

次にエミュレータの各種の機能について述べる。

第1にエミュレーションCPCである。ターゲット・システムのCPUソケットに挿入されたプローブの根元は，エミュレーションCPUにつながっている。エミュレーションCPUは，ターゲット・システムの本来のバスに接続されると同時に，エミュレータ内部のバスにもつながる。この接続が図1－24のように直接CPUがつながつているものと，図1－25のようにバッファICを使用している場合がある。

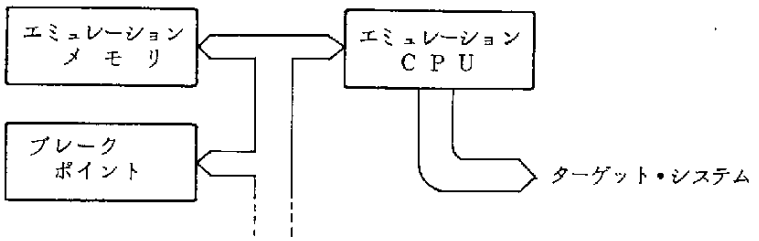


図1－24　ターゲット・システムとエミュレーションCPUを直結する

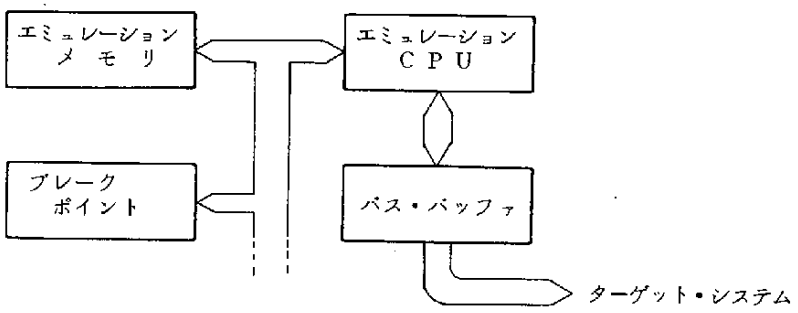


図1－25 ターゲット・システムとエミュレーションCPUの間にバッファを入れる

図1－24では，CPUのバスに対し，プローブの浮遊容量や，その他の負荷が加わる上に，ターゲット・システムのバス上の障害により，エミュレータの動作にも影響を生じる可能性がある。図1－25は，エミュレーションCPUからバッファを通してターゲット・システムのバスに接続されている。この方式では，ターゲット・システムのバスの障害は，エミュレータ内のバッファで切り放されるのでエミュレータの動作に対する影響は少ない。しかし，ターゲットシステムから見て，CPUへの入力信号はLSタイプのIC1個分になり，CPUの入力仕様と異なる。ターゲット・システム単独では動くが，エミュレータを接続すると動かないとか，または，その逆のケースはこのような負荷の差によることがある。

ブレーク・ポイントはデバッグ中に最もよく使われる機能であり，この機能の良し悪しがデバッグの能率を左右する。

最も一般的ブレーク・ポイントの方法は，特定のアドレスを指定し，そこをアクセスすれば，ブレークが発生する。アクセスの種類として，リード，ライト，エクセキューション，I/Oリード，I/Oライトのいずれか，またはそれらの組合わせで指定できることが望ましい。

同時に指定できるブレーク・ポイントの数は，1個では不便で，複数個必要であるが，あまり多くても管理できなくなる。しかし実際のデバッグ中に，ある領域へジャンプした時，またはある領域外を実行した時

（プログラムの暴走などで良くある）には，特定の番地でなくても常にブレークしたい時がある。このような要求のために，ブレークを，特定のアドレスでなく，アドレス領域で示すことができるものがある。このような例として，トラップ・アレイの概念を図１－26に示す。

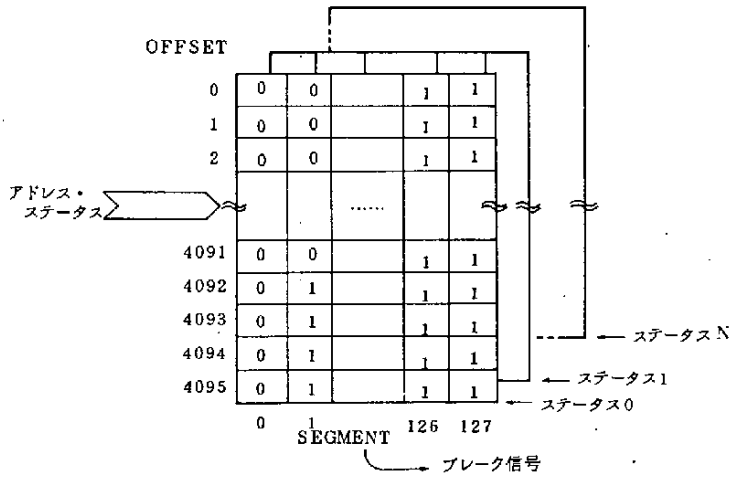
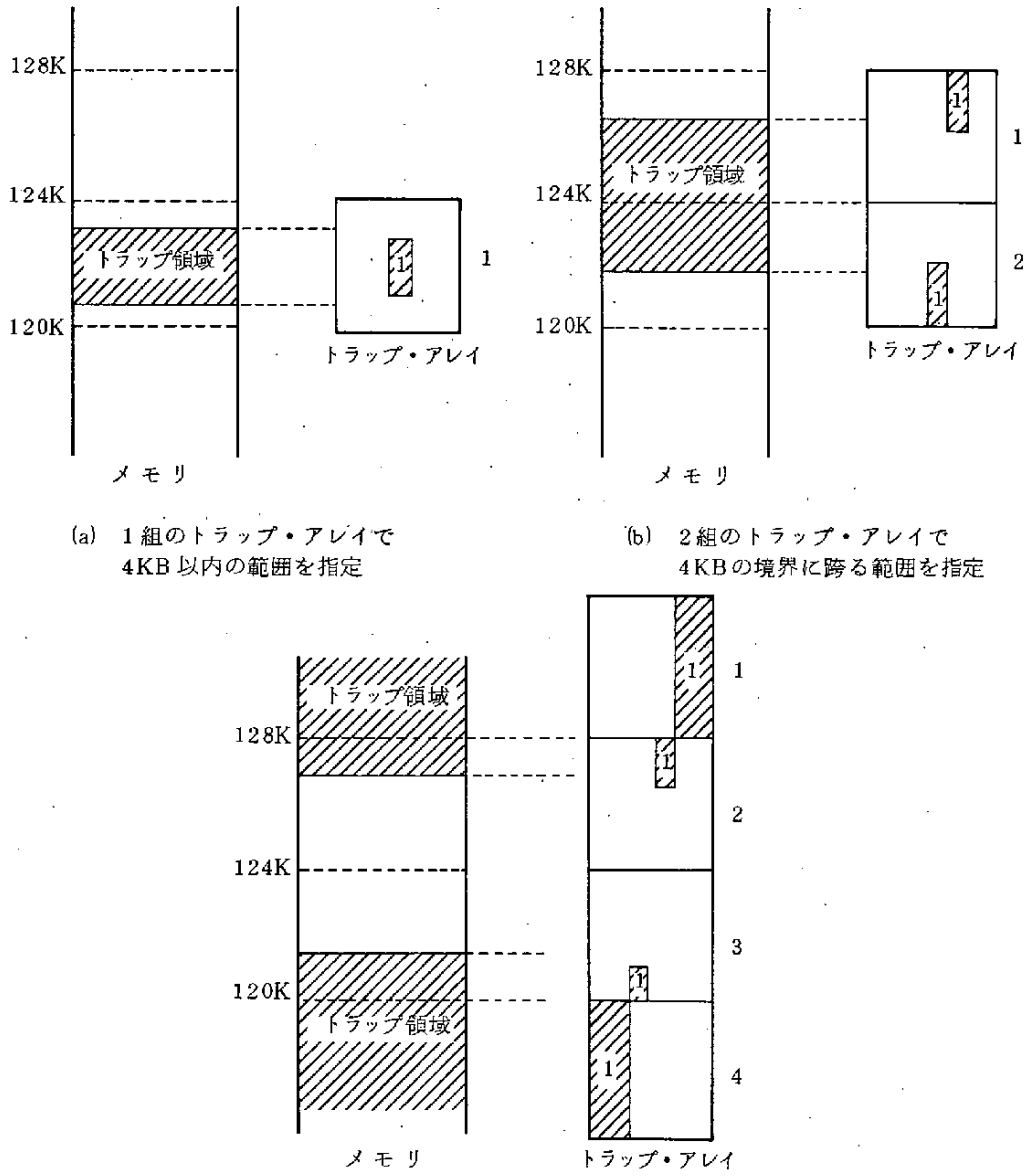


図１－２６　トラップ・アレイの概念図
（１組のトラップ・アレイは，４KBの範囲を指定できる。）

トラップ・アレイは，メモリを256個のセグメントに分け,1個のセグメントの大きさは4KBとして見ている。

トラップ・アレイは256個の，セグメントに対応した4096ビットのレジスタ・アレイで構成され，このレジスタ・アレイが，アクセス・モードの種類分だけ用意されている。与えられたアドレスとステータスで示されるビットの内容が１ならばブレークが発生する。図１－26の例では，ステータス０のプレーンで，セグメント１のオフセット4092以上が，１になっていると，アドレス8188以上のいずれかをアクセスした時にトラップが発生する。このようなトラップ・アレイをいくつか組合わせて，図１－27のような指定が可能になる。ここで示したトラップ・アレイは4KBのセグメント単位で示されるので，図１－27（ｂ）のように，セグメント境界にまたがった場合は，２組のトラップ・アレイを必

要とする。図1－27(c)は，特定の領域外を全て，トラップの対象とする例である。このとき，アレイ1とアレイ4は，オフセットは全てのアドレスでトラップをオンにし，セグメントの指定で，トラップの発生領域を定める。



(a) 1組のトラップ・アレイで
4KB以内の範囲を指定

(b) 2組のトラップ・アレイで
4KBの境界に跨る範囲を指定

(c) 実行領域の外を全て指定する

図1－27　トラップアレイによる領域の指定

代行メモリは，エミュレータの重要な機能である。代行メモリは，エミュレータ自身のCPUが使用するメモリとは別に，専用の高速 RAMを用意している。ターゲット・システムがプログラムをROMに記憶するような場合には，開発中には代行メモリは必須である。

代行メモリ機能の概要を図1－28に示す。代行メモリは，ターゲット・システムのメモリ空間の任意のアドレスに対応させるために，エミュレーションCPUからのアドレスは，メモリ・マネージメント・ユニット（MMU）によって代行メモリの実アドレスに変換される。

MMUは，エクセキュート，リード，ライトのアクセス・モードの指定も可能にし，デバッグ中に，思いがけないメモリの内容の破壊を防ぐことができると共に，プログラムのバグの検出に役立つ。

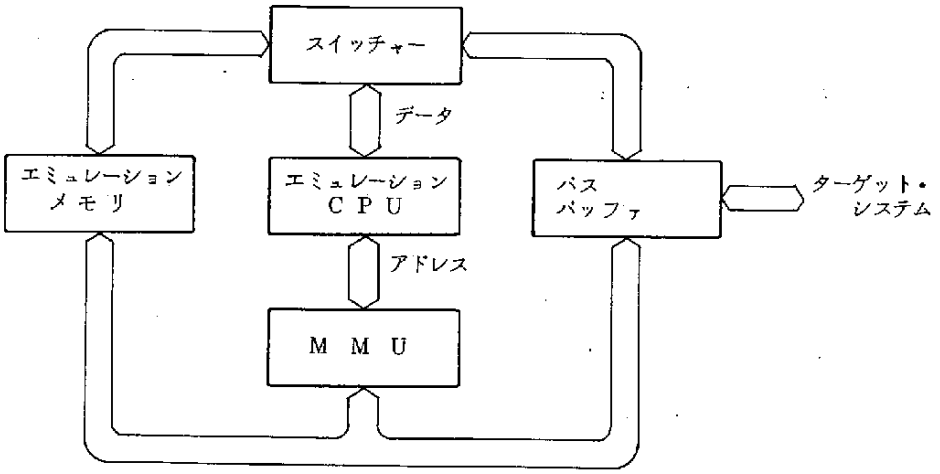


図1－28　エミュレーション・メモリ機能

トレースは，ロジック・ステート・アナライザのトレースと同様である。トレースされる情報は，アドレス，データ，CPU ステータス等の外に，外部からのステータス入力も行えると便利である。トレースのタイミングは，CPUのクロック・タイミングで行えば，ロジック・ステート・アナライザとして使用することも可能であり，ハードウェアに密着するデバッグに有効である。

その他のタイミングとしては，バス・サイクルや，特定の指定したイベント・サイクルを使用する等，各種のモードが用意されている。

トレース機能を使用する時に，重要なもう一つの機能がトリガである。トレースは，エミュレーション中は，指定サイクルごとにトレース・メモリに情報の記録が行われている。このメモリの読み出しは，トレースを停止し，メモリの内容を固定しなければならない。

ブレークによって，エミュレーションを停止すれば，トレースも終り，それまでの記録の読み出しが可能である。しかし，デバッグ中に，エミュレーションを停止させずに，特定の実行範囲のトレースを見たいことが多くある。このために，トレースのオン／オフを行うのが，トリガである。

トリガ機能は，基本的にはロジック・ステート・アナライザと同様である。図1－29の例でプレトリガは，トレースの開始であり，ポストトリガは，トレースの停止である。

ポストトリガの発生後，トレース停止までの間を，ディレイ・カウンタで遅らせることもできる。ディレイ・カウンタには，トレースのタイミング・クロックが用いられる。

トリガの発生条件は，アドレス・データ，外部信号等の組合せで指定できる。特に外部信号を利用することにより，入出力時のプログラムの応答などのデバッグに有効である。

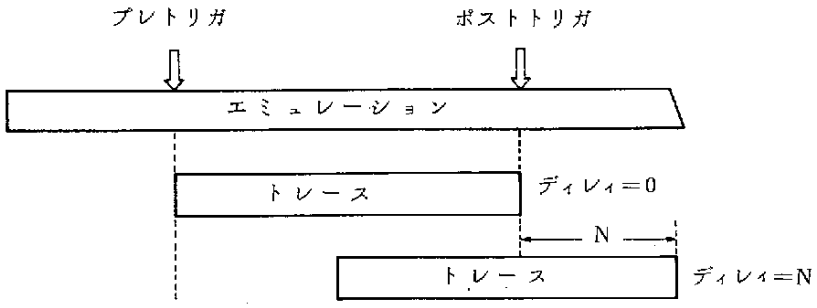


図1－29　プレトリガ，ポストトリガとトレース

1.2.4　コンパイラ

組込み用システムでも，プログラムの生産性や保守性の向上の要求と，メモリ素子のコストの低下により，高級言語での記述が多くなっている。

組込みシステムで，高級言語を用いる時は，次の点に注意が必要である。

① ROM化が可能であること。

② ライブラリは必要なもののみを，リンクできること。

③ ランタイム・ルーチンは公開されているか，代行の手法が明確なこと。

④ アセンブラ・プログラムとのリンクが可能であること。できればインライン・アセンブルが可能であること。

⑤ ロード・モジュールの形式が公開されていること。

⑥ 浮動小数点演算の処理が明らかになっていること。ハードウェアのプロセッサが必要な時がある。

ここでは，構造型言語であるが，柔軟性が高く，システム記述などに用いられているCと，リアルタイム処理を目的として作られたAdaについて取り上げる。

Cは，ベル研究所で開発され，UNIXオペレーティング・システムを記述するのに使用された。

現在Cコンパイラは，UNIXのみでなく，CP/M,MS/DOSなどの，流通システムで動くものが，多く発売されている。

Cは，全ての変数について型宣言を行うこと，if～else，while，swith などの制御方式，外部変数，自動変数，静的変数などの扱い等，構造型言語に類するが，ポインタ，レジスタ変数などのハードウェアを意識した表現や，ビット操作用の演算子など，制御システムを記述しやすい言語である。

Cの標準的教科は，B.W. カーニハン，D.M. リッチーの共著による

(注)　C,UNIXはベル研究所，AdaはDOD（米国国防省），CP/Mはデジタルリサーチ社，MS—DOSはマイクロソフト社の登録商標

"The C Programming Language"[1]で，邦訳も出版されている。

Cの主な特徴を次に述べる。

Cの基本構造は関数である。プログラムは関数の集まりとして記述され，メイン・プログラムはmain()という関数である。main も関数であるので，当然main(argc,argv)という型で，引数を使用できる。これは，プログラムの起動時に与えられたパラメータを受け取る。

関数内部で使用される変数は，特に指定されない限り，局所的であり，関数が呼ばれた時に，一般にスタック領域に作られる。このために，アセンブラで記述されたプログラムのように，変数領域が固定化されず，実行シーケンスによって変化するので，組込型のシステムでは，メモリのRAM領域の大きさに，考慮が必要である。

データの型は次の6通りである。

char	文字型
int	単精度整数型
short	短精度整数型
long	倍精度整数型
float	単精度実数型
double	倍精度実数型

データの精度はマシンに依存し，一例をあげると表1−6のようになる。

表1−6　データ型の精度（ビット長）

	PDP−11	VAX−11
char	8	8
int	16	32
short	16	16
long	32	32
float	32	32
double	64	64

文字列は文字型の配列で示され，文字列の終端はヌル・コード(0)である。

演算式中において，変数の型の整合は，あまり厳密でなく，自動的に変換される。例えばcharは文字型であるが，8ビットの整数として扱うことも可能である。但しこれには8ビットコードの文字では，整数として扱うときは，符号に注意が必要である。このあたりは，マシンに依存するので，汎用ルーチンを作る時は注意しなければならない。

変数として特徴のあるのは，ポインタである。ポインタは，他の変数のアドレスを示す変数である。ポインタは整数型変数として宣言される。Cの特徴は，ポインタに対して，インクレメント，デクレメント，加算，減算を許すことである。ポインタへの演算は，アドレス演算であるので，＋1といっても，1番地アドレスが変化するのでなく，そのポインタが示す変数の型に応じて，例えばcharなら＋1，intなら＋2のようになる。ポインタの利用は，アセンブラ・プログラムにおける，インデックスによる関接参照のようなものであり，用途が多い。

演算子については，表1－7に示す。全ての演算子について，優先順位が定められていると，表の上位のものほど，優先順位が高い。

表1－7　演算子の優先度と評価順序

	演・　算　　子	結合規則
高	( ) [ ] → .	左から右
	! ~ ++ −− − (type)* & sizeof	右から左
	* / %	左から右
優	+ −	左から右
	<< >>	左から右
	< <= > >=	左から右
先	== !=	左から右
	&	左から右
	^	左から右
度	|	左から右
	&&	左から右
	||	左から右
	? :	右から左
	= += −= etc.	右から左
低	,	左から右

＊が２個所にあるが，上位のものは，ポインタによる間接参照で，下位のものは，乗算である。&についても同様で，上位のものは，変数のアドレスの取り出しで，下位のものは，ビット演算のANDである。

ビット演算子は，次の６種である。

&	AND
|	OR
∧	XOR
≪	左シフト
≫	右シフト
～	１の補数

これらの演算子は，特に制御システムの記述に有効である。

Cには，入出力文はない。全ての入出力は，関数によって行われる。このことが，Cプログラムの移植性を高めることになっている。標準的な入出力は表１－８に示される。いずれも，比較的小さな関数であるので，組込システム等でも，独自に作ることも容易である。

表１－８　Cの標準入出力関数

関　数	機　　能
Getchar	標準入力（通常はユーザのコンソール）から，１文字づつ読み込む。
Putchar	標準出力（通常はユーザのコンソール）に１文字出力する。
Printf	書式指定付きで，引数の値を出力する。 書式は，１０進表示，符号なし８進表示，符号なし１６進表示，符号なし１０進表示，文字列，浮動小数表示，指数表示等がある。
Scanf	書式指定付きの入力を行う。 Printf と同符な書式指定で，入力されたデータを，引数に代入する。
Sscanf Sprintf	Scanf，Printfと同様の変換を行うが，その結果，又は，入力を，文字列に対して行う。
Fopen	ファイルをオープンする。 ファイル・アクセスの始めに必要。
Fclose	ファイルをクローズする。 ファイル・アクセスの終了。

表１－８　Cの標準入出力関数（つづき）

関　数	機　　能
Getc	ファイルから１文字入力する。
Putc	ファイルへ１文字出力する。
Fgets	ファイルから文字列を読み込む。
Fputs	ファイルへ文字列を書き出す。
Ungetc	ファイルを１文字を戻す。

Adaは米国国防省に，大規模のリアルタイム処理のソフトウェアを，能率良く開発するために作られた，言語である。

現在いくつかのシステムで，フルセットやサブセットが動いている。

Adaには，今までの言語にみられない，制御用言語としての特徴がある。

Adaの主な特徴は次のようなものである。

① 構造型言語

Adaは，全体の記述はPascalによく似ている。

② 強いデータ型

データ型定義機能も豊富である。

型変換は常に明示的に行わなければならない。

③ 分割コンパイル

分割コンパイルのために，サブプログラムや，パッケージなどのプログラム単位が，宣言部とプログラム本体に分かれている。

④ 並列処理

並列的に処理されるタスクの定義，実行と停止，およびタスク間の同期手段が定められている。

⑤ 例外処理

例外処理の定義，処理手続きの記述およびソフトウェア上での例外条件の発生などが定められている。

並列処理機能を，言語規約として定めることにより，リアルタイム処理のソフトウェアの，移植性が高くなる。

並行処理は，タスクと呼ばれるプログラム単位によって行われる。タスクは，図1－30に示されるように，タスク宣言とタスク本体に分けられる。

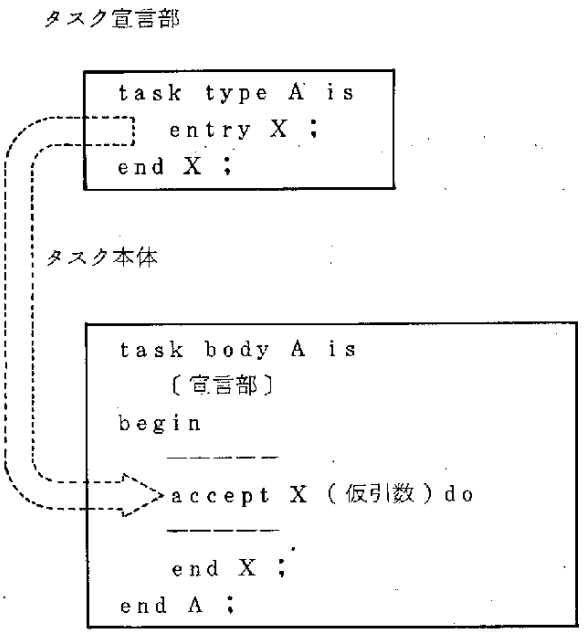


図1－30　タスク宣言とタスク本体

タスク宣言部に他のタスクとの通信受け口が記述され，タスク本体に処理入口がacceptで示される。タスクは，タスク本体に対応する宣言部分が評価された時に起動され，その後は，それを宣言したプログラム単位や，他のタスクと並行に実行する。

タスク間の通信は，ランデブという方式で行う。タスク本体には，宣言部のentry文に対応して，accept文による処理入口がある。タスクへメッセージを送るプログラム単位は，エントリ・コールによって，タスク内の手続きを呼び出す。（図1－31）

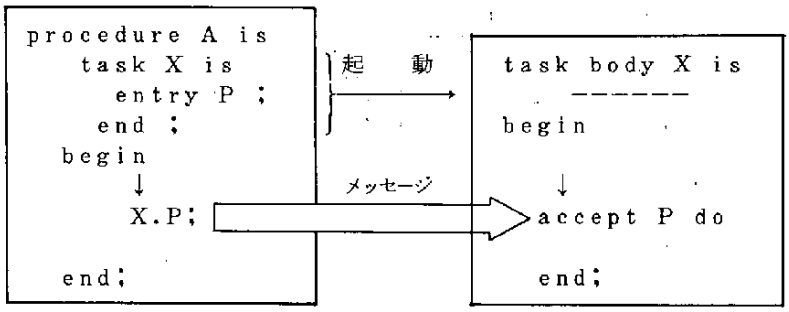


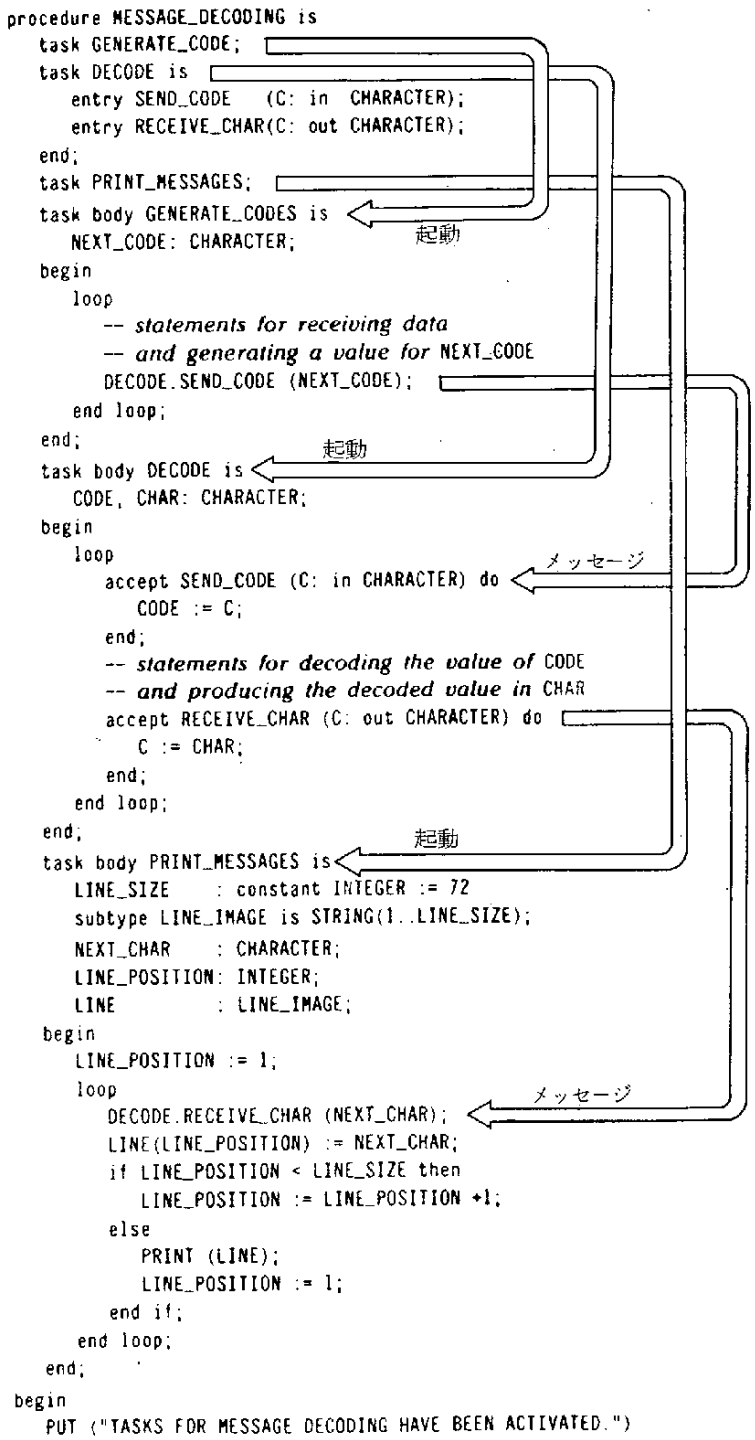
図１－31　タスク間のメッセージ通信

このとき，呼び出されたタスクで，処理準備がととのっていない時は，呼び出し側は，タスクの実行が進むまで待たされる。逆の場合には，タスクは，メッセージの到着まで待っている。

図１－35に,プログラム例とメッセージ通信の流れを示す。この例は，H．レッドガード著「Ada入門／和訳規約」上條史彦 ：他[(2)]訳からとった。

この例では，MESSAGE_DECODING という一つのプロセジャが，GENERATE_CODE , DECODE , PRINT_MESSAGES という三つのタスクで構成されている。GENERATE_CODEは，メッセージを，しかるべき所から受信し，それをDECODEタスクに対し，SEND_CODEエントリを通して，送る。PRINT_MESSAGESは，DECODEタスクが処理した文字を受け取って印刷する。

これら三つのタスクは，それぞれ独立に実行しているが，DECODEタスクは，GENERATE_CODEタスクから，文字を受信しなければ，処理結果を出力できない。PRINT_MESSAGESは，DECODE タスクからの出力がなければ，印刷できない。この間の同期をとっているのが，メッセージによるランデブ（待合せ）である。




```
procedure MESSAGE_DECODING is
   task GENERATE_CODE;
   task DECODE is
      entry SEND_CODE   (C: in  CHARACTER);
      entry RECEIVE_CHAR(C: out CHARACTER);
   end;
   task PRINT_MESSAGES;
   task body GENERATE_CODES is
      NEXT_CODE: CHARACTER;
   begin
      loop
         -- statements for receiving data
         -- and generating a value for NEXT_CODE
         DECODE.SEND_CODE (NEXT_CODE);
      end loop;
   end;
   task body DECODE is
      CODE, CHAR: CHARACTER;
   begin
      loop
         accept SEND_CODE (C: in CHARACTER) do
            CODE := C;
         end;
         -- statements for decoding the value of CODE
         -- and producing the decoded value in CHAR
         accept RECEIVE_CHAR (C: out CHARACTER) do
            C := CHAR;
         end;
      end loop;
   end;
   task body PRINT_MESSAGES is
      LINE_SIZE     : constant INTEGER := 72
      subtype LINE_IMAGE is STRING(1..LINE_SIZE);
      NEXT_CHAR     : CHARACTER;
      LINE_POSITION: INTEGER;
      LINE          : LINE_IMAGE;
   begin
      LINE_POSITION := 1;
      loop
         DECODE.RECEIVE_CHAR (NEXT_CHAR);
         LINE(LINE_POSITION) := NEXT_CHAR;
         if LINE_POSITION < LINE_SIZE then
            LINE_POSITION := LINE_POSITION +1;
         else
            PRINT (LINE);
            LINE_POSITION := 1;
         end if;
      end loop;
   end;
begin
   PUT ("TASKS FOR MESSAGE DECODING HAVE BEEN ACTIVATED.")
end;
```

図１－32　タスクのプログラミング例

1.2.5　ツール一覧

最後に現在発表されている。ロジック・アナライサ，インサーキット・エミュレータの一覧表を示す。

これらの測定器は，新製品の発表が多く，全てを網羅することは不可能であり，この表も不完全であることをお断りしておく。

表1－9は，日経エレクトロニクス[3]に掲げられた製品以外の，その後の新製品を示す。

表1－10は，インサーキット・エミューレタについて，現在発売されている機種について，主な仕様をまとめた。

表1－9　最近のロジックアナライザ

型　名	クロック周波数(MHz) 同期／非同期	最大データ入力チャネル数	メモリ容量 ビット／チャネル	グリッチ 検出巾(ms)	サポートCPU	オプション他	メーカ名(販売元)
205	12／10	ステート入力　48 タイミング入力　16	250	25	6800 6809 8080 8085 Z80 NC80	ワード・ジェネレータ 不揮発メモリ GPIB	RACAL−DANA (関商事)
TR4725	50／100	ステート入力　64 タイミング入力　16	1024	5	8086／88	3.5" フロッピ・ディスク RS・232	タケダ理研
GL6450	50／50	ステート入力　64 タイミング　16	1024	?	8085 Z80 8086 68000	バブルメモリ カセット	グローバル電子
KLA48	50／100	64 (100MHz・32ch)	2048 (オプションで 4096)	5	8085 Z80 6800 6802 6809 8086／87／88 68000 Z8001／2	5" フロッピ・ディスク RS232C GPIB	KONTRON ELECTRONICS (トマスエレクトロン)
LX−701	／10	16	256	15			アーロン

表1－10　インサーキット・エミュレーター一覧

型　名	対象プロセッサ	エミュレーション・メモリ			トレース		ブレーク・ポイント			備　考	メーカ名
		容量	分割サイズ	保護	メモリ容量	内容	個数	領域指定	条件		
ai JCE 86/88	8086/8088+8087	256KB	1KB	有	1K	アドレス・データ ステータス 外部×4	4	可	アドレス・イベント 外部	シンボリック・デバッガ 実行時間測定 スタンド・アローン型	アイ電子測器
ESシリーズ	68000.Z8001/2/3	512KB	2KB	有	1K	アドレス・データ ステータス 外部×16	4	可	アドレス・イベント 外部	スタンド・アローン型	APPLIED MICROSYSTEMS （東陽通商）
AVICE-Z80	Z80	64KB	1KB	無	2K	データ 外部×8	2	不可	アドレス 外部	スタンド・アローン型 小型簡易型	アーバル・コーポレーション
SL4802	8086	256KB	2KB	無	1K	アドレス・データ ステータス 外部×8	4		アドレス・イベント	専用システム型 CP/Mコンパチブル・ フロッピ・ディスク	岩通
I²ICE	80286/287. 80186/188 8086/88+8087	288KB	1KB	有	1K	アドレス・データ ステータス 外部×16	2	可	アドレス・イベント	シンボリック・デバッガ 実行時間測定 100MHz・16ch タイミングアナライザ開発システム	INTEL
JCD/86	8086/88+8087	512KB	1KB	有	1K	アドレス・データ ステータス	6	不可	アドレス	シンボリック・デバッガ スタンド・アローン型	国際データ機器
CRC-BOX	8086 Z80	128KB	64KB	無	1850	アドレス・データ ステータス	2	不可	アドレス	開発システム組込型	コンピュータ・リサーチ
SA700M	8086/88+8087	64KB DRAM 252KB	4KB	有	4KB	アドレス・データ ステータス 外部×4	5	可	アドレス・イベント 外部	実行時間測定 専用システム	ソフィアシステムズ
INTERACT	68000	152KB DRAM 512KB	8KB	有	4KB	アドレス・データ ステータス 外部×8	14以上	可	アドレス・イベント 外部	実行時間測定 アクセス・モニタ 3レベル・シーケンシャル・トリガ	インデータシステムズ
8500シリーズ	8086/88 8085 8080 68000 6800 6802/8/9 Z8001/2.280 他	128KB	4KB	有	256	アドレス・データ ステータス 外部×8	4	可	アドレス・イベント 外部	測行時間測定	Tektronix
DSC-3502	68000 8086 8085 Z80	448K	28KB		240	アドレス・データ ステータス 外部×8	3	可	アドレス・イベント 外部	測行時間測定 専用システム	電産
μCOMICE	μPD780 8085 μPD84シリーズ	4K	1KB		1K	アドレス・データ ステータス 外部×16	2		アドレス・イベント	PDA-880システム	日本電気

型　　名	対象プログラム	エミュレーション・メモリ			トレース		ブレーク・ポイン			備　　考	メーカ名
		容　量	分割サイズ	保護	メモリ容量	内　容	個　数	領域指定	条　件		
ICD178	8086/88+8087	1MB	1KB	有	4K	アドレス・データ ステータス	1 ソフト ウェア8	可	アドレス・イベント 外　部	スタンド・アローン型	ザックス・コーポレーション
HDS・400	68000,68008 68010	288KB	4KB	有	128	アドレス・データ ステータス 外部×8	16 5レナル (バスアナライザ)	可	アドレス・イベント	トレースはバスアナライザ使用 EXORmacs, VME/10　開発システム	MOTOROLA
TICE/4P	8085,Z80	64KB	1KB	有	2K	アドレス・データ ステータス 外部×3	3		アドレス・イベント	4CPUエミュレーション,スクリーン・エディタ,光ファイバ,インターフェイス(30m) TDSA8D開発システム	タイテック
64000	68000　6800/1/2/3 6805/9 8086/88　8080/85 Z80　他	128KB ×4	256/4K	有	256	アドレス・データ ステータス	ハード ウェア1 ソフト ウェア8	可	アドレス・イベント	パフォーマンス・アナライザ タイミング・アナライザ 開発システム	横河：ヒューレット パッカード

参 考 文 献

(1) B.W. カーニハン, D.M. リッチー著
「プログラミング言語C」石田晴久訳

(2) H. レッドガード著「Ada 入門／和訳規約」
上條史彦, 細谷僚一, 永瀬信夫, 石原憲一訳

(3) 「16 ビット・プロセッサの応用開発に向け, ロジック・アナライザ新機種が続々登場」日経エレクトロニクス, 1983 , 8－1 PP 217

第2章　マイクロコンピュータ応用システム開発の現状

第2章　マイクロコンピュータ応用システム開発の現状

2.1　ワンチップマイコン応用システム

ワンチップマイコンはその応用上「部品的応用」と称され，一個の機能要素として利用される。また，応用製品は，不特定多数の使用者を対象として大量生産されることが一般的である。従って，応用製品の使用者はマイコンの存在を意識することなく，製品そのものが提供する本来的機能を利用するので，マイコンの利用技術やプログラミング技術を要求されることはない。さらに，プログラムはマイコンチップに内蔵されるＲＯＭにＬＳＩの製造工程で書き込まれるため，改変できないので，プログラムのバグはハードウェアの不良と等価となる。システムコストに占めるプログラム開発費用に比較して，ＬＳＩそのものに対するコストが大きいため，メモリサイズ，機能ブロック，周辺機能などに最適化がなされる。このため，ワンチップマイコンは極めて多種多様な展開を見せている。

2.1.1　要求仕様

ワンチップマイコンはＬＳＩ１個にマイコンの構成要素を集積するため構造上次のような特徴を持っている。

① プログラムを格納するためのＲＯＭとデータを格納するためのＲＡＭに分割されたメモリ構造を持ち，アドレシングの仕方が互いに異なる。

② ＲＯＭのアドレシングはページ構成を採るものが一般的である。

③ ＲＡＭはアドレスレジスタによる間接アドレシングが標準的であるが，直接アドレス指定のできる領域を有するものもある。

④ メモリサイズは小さくＲＯＭは８キロバイト，ＲＡＭは２５６バイトまでである。

⑤ データの処理幅は４ビットが主体であるが８ビットのものも増加

しつつある。主に内部バスと処理データとの間にはビット幅の不整合が目立つ。

⑥ 内部レジスタは少なく，多目的に使用される。

⑦ サブルーチンスタックの段数が小さく，多重ネスティングが難しい。

⑧ 割込み機能の貧弱なものが多い。

⑨ 多種，多数の入出力端子が用意されており，また端子の機能が多重化されているものもある。

⑩ ビット処理がＲＡＭや入出力端子に配慮されている。

⑪ タイマ・カウンタ，アナログ入出力，表示駆動回路など周辺回路を取込むものが多い。

⑫ 電源消費を小さくする工夫がなされている。

以上の様な構造上の制約や特徴を有するワンチップマイコンは図２－１に示す進化をたどり，またＲＯＭサイズや付加機能の種類などの組合せで同一シリーズでも極めて多種多様になっており；ユーザに最適な選択ができるようになっている。

応用製品が大量生産されること，プログラムが，ＬＳＩの製造工程で書き込まれること，応用製品のユーザはマイコンの存在を意識しないことなどから，プログラムは金型の一種とみなされ，バグはハードウェアの不良と等価となる。これらが要因となって，ワンチップマイコンは応用上，次の様な特徴を持つ。

① プログラムサイズの圧縮が重大である。

② 機器の直接制御を目的とするため，タイミング管理のリアルタイム性が重要である。

③ 応用製品が大量生産されるためプログラムコストは製品価格に比して小さい。

④ マイコンチップに対するコストプッシュが大きく周辺機能を取り

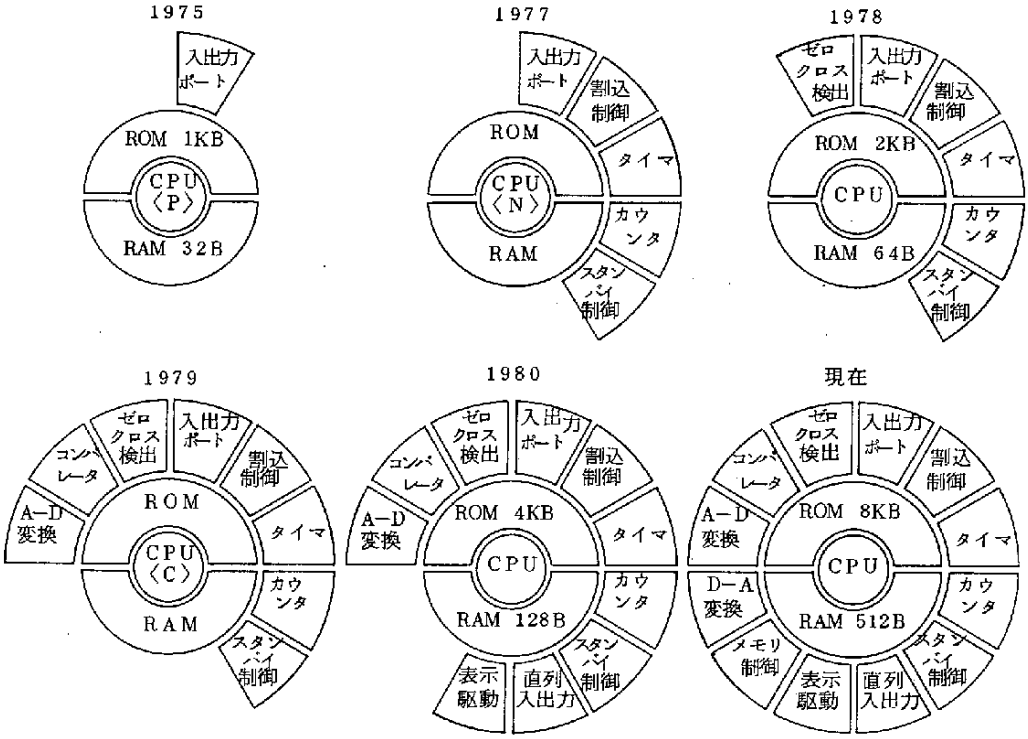


図２－１　ワンチップマイコンの進化

込むことからカスタムマイコン化する。

⑤　電源，センサ，アクチュエータなどのコストがマイコンチップのコストに比し重大となる。

⑥　プログラムの開発期間を短縮することが極めて重視される。

⑦　ＪＯＢ間の切換えが頻繁でマルチタスク処理の実行が不可欠である。

⑧　プログラムの細部における僅かの変更が，特にマルチタスク処理やリアルタイム処理のために，プログラム全体に波及することが多い。

⑨　プログラムのモジュール化，標準化が困難で，再利用されることが少ない。

以上の事柄を要約すると図２－２の様になるが，特にリアルタイム処理，マルチタスク処理やオブジェクトプログラムサイズのコンパクト性に対する要求が大きいためプログラム開発を高位言語で実施することが難しく，このことがプログラム開発の期間圧縮を阻害している。この問題を解決すべく図２－３に示す様に，ハードウェア，ソフトウェア両面からのアプローチがなされている。ハードウェア上は，割込み機能を拡充強化する，マルチタスク・リアルタイムモニタを組込む，ワンチップマルチマイコンとするなどの方策が採られている。

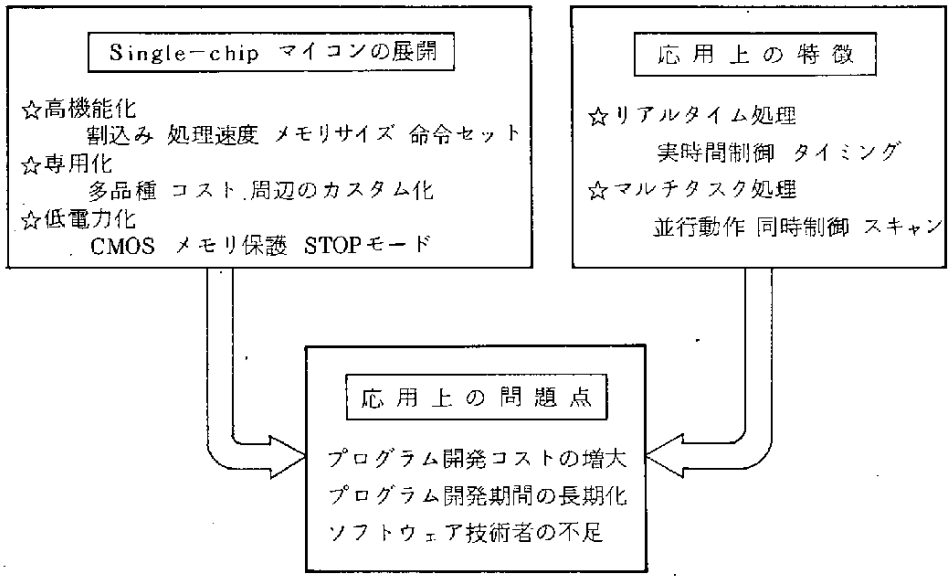


図２－２　ワンチップマイコンの応用環境

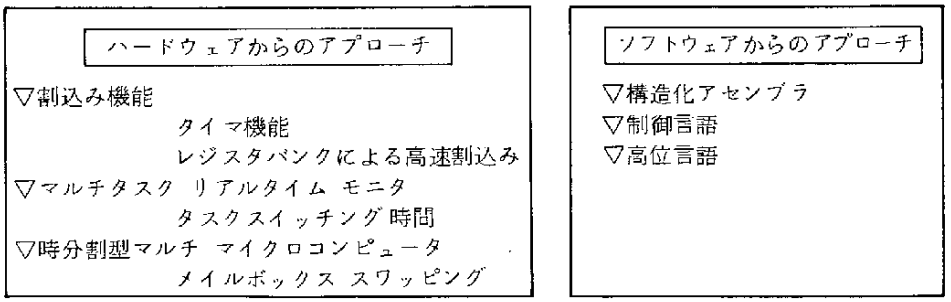


図２－３　ワンチップマイコンの展開

図２－４は昨年６月松下電器が発表したマルチタスクの高速処理に適した疑似デュアルワンチップマイコンMN18941のブロック図である。このマイコンはプログラムカウンタ，ステータスフラッグ，ポインタレジスタを２組備え，これを命令サイクル単位で図２－５の様に切替えて，２個のタスクを並行処理するもので，命令プリフェッチキューやパイプライン制御によって処理を高速化している。命令サイクル500ns で２タスクを並行的に実行するので仮想的に命令サイクル1μsのマイコンが２個動作していることになる。さらに，ROM，RAM，入出力ポートなどを共有するので独立の２CPUを利用する場合に比べプログラムサイズの圧縮，メールボックスによるCPU間情報交換などが可能である。このマイコンの概略は次のとおりで，6.4㎜×6.0㎜のサイズに67,000トランジスタを累積している。

- 仮想的に２個のCPUをワンチップ化
- ROM４Kバイト，RAM２５６バイト
- 効率のよい命令
  - メモリマップドI／O
  - ８／４／１ビット操作
  - 命令リピート機能
    - ストリング操作・多方向分岐
  - 多重ループネスティング
  - 相対分岐命令
  - ROM領域テーブルルックアップ
- 割込み機能の強化
  - 外部・タイマ・シリアル・トラップ
- デュアル１６ビットタイマ／カウンタ
- 直列伝送インターフェイス
- 誤動作防止機能

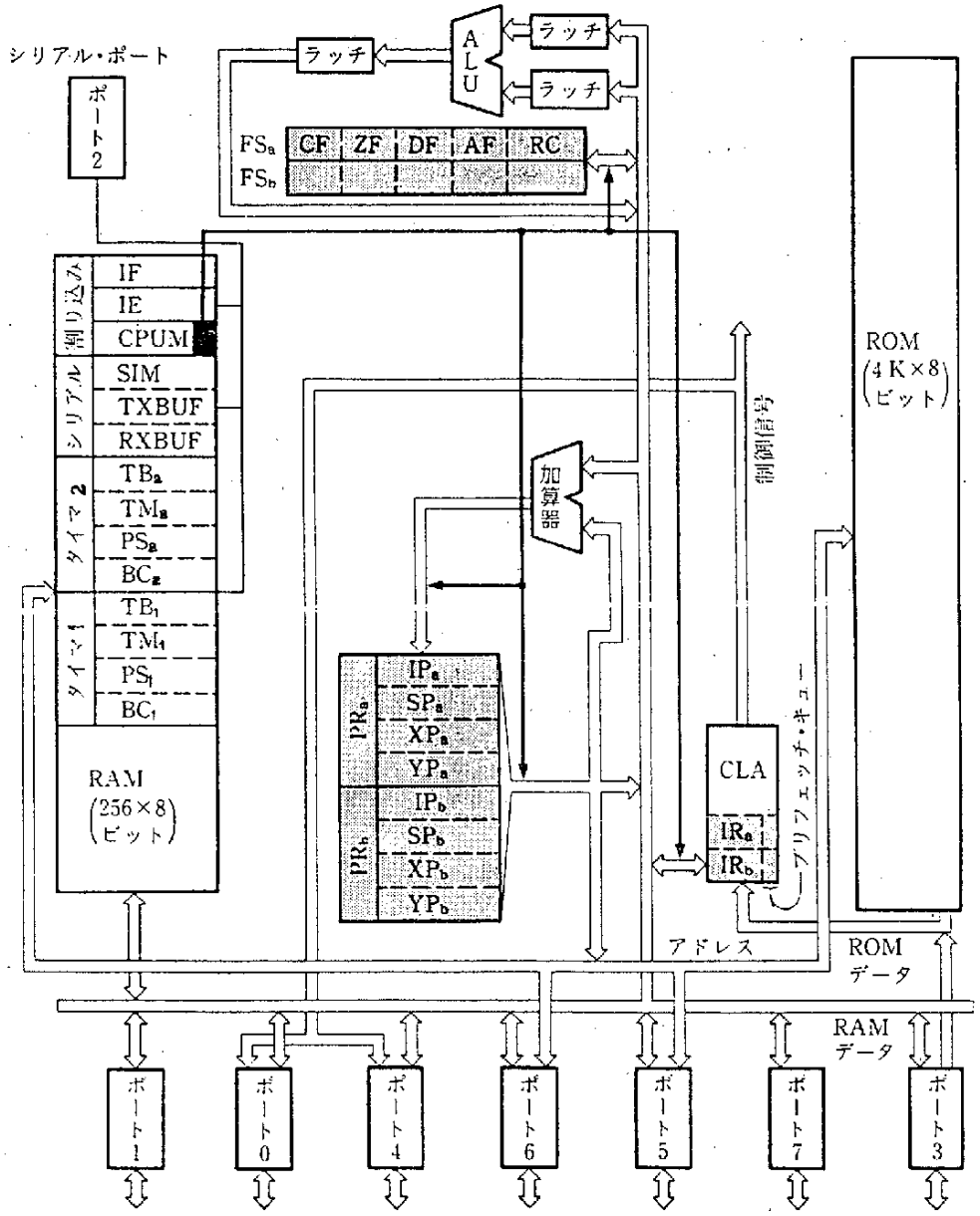


図２－４　擬似デュアルマイコンMN１８９４１

（□の部分が２重になっている）

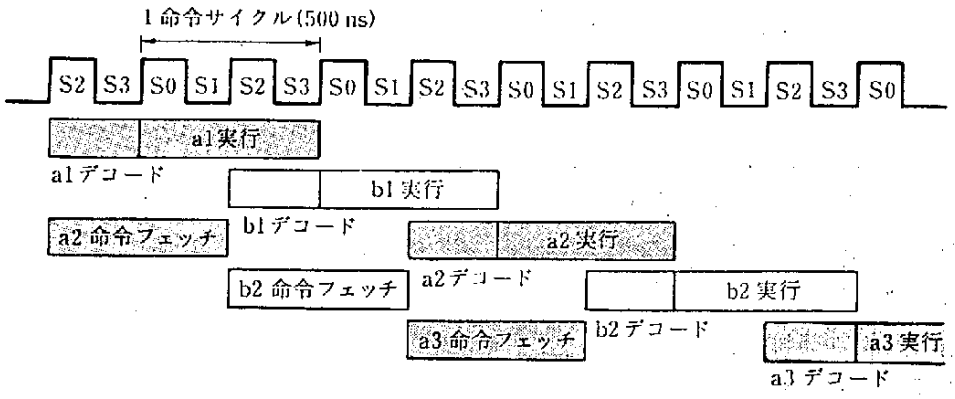


図２－５　MN１８９４１のタイミング

低電圧検出，暴走トラップ

・外部ＲＯＭ，ＲＡＭの接続

最大　各６４Ｋバイト

・クロック回路内蔵

・高速命令実行

最小５００ｎｓ　　平均1.5 μs

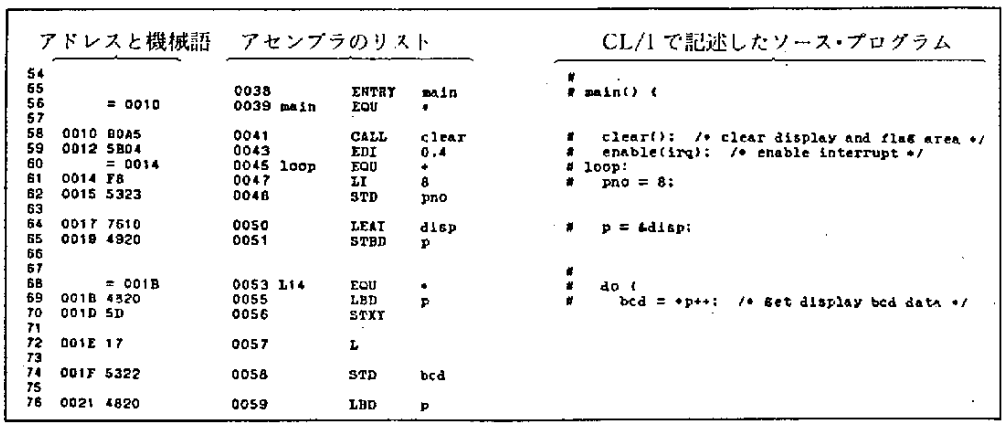
・ｎ－ＭＯＳ　５Ｖ単一電源

・６４ピン　縮少ＤＩＬプラスチック

・入出力ポート　５１本

また，ソフトウェアからのアプローチとしては，プログラム開発を効率化するためのツールの拡充や記述性とオブジェクト効率とのバランスをとりながら，構造化アセンブラ，制御言語，高位言語などが提供されている。第４章で紹介されているワンチップマイコン用コンパイラＣＬ／１もその例である。この言語はＣ言語に準じた演算や制御文を用い，割込みや入出力ポートの処理を可能としたもので，記述ステップ数はアセンブラの１／２以下でオブジェクトは1.2倍以下となるクロスコンパ

イラである。図２－６にプログラムリストの一例を示す。



```
アドレスと機械語   アセンブラのリスト                    CL/1で記述したソース・プログラム
54                                                    #
55                 0038          ENTRY   main         # main() {
56        = 0010   0039 main      EQU     *
57
58  0010 B0A5      0041          CALL    clear        #   clear();  /* clear display and flag area */
59  0012 5B04      0043          EDI     0,4          #   enable(irq);  /* enable interrupt */
60        = 0014   0045 loop      EQU     *            # loop:
61  0014 F8        0047          LI      8            #   pno = 8;
62  0015 5323      0048          STD     pno
63
64  0017 7610      0050          LEAI    disp         #   p = &disp;
65  0019 4920      0051          STBD    p
66
67                                                    #
68        = 001B   0053 L14       EQU     *            #   do {
69  001B 4820      0055          LBD     p            #     bcd = *p++;  /* get display bcd data */
70  001D 5D        0056          STXY
71
72  001E 17        0057          L
73
74  001F 5322      0058          STD     bcd
75
76  0021 4820      0059          LBD     p
```

図２－６　ワンチップマイコン用コンパイラCL／１による
プログラムリストの例

2.1.2　開発工程

ワンチップマイコンは既述のように，ハードウェアに直結した制御が重要でしかもメモリ空間が小さいので，予め用意された汎用サブプログラムを組合せて新しいシステムを構成することが難しく，応用毎に専用プログラムを作らざるを得ない。ワンチップマイコンの応用システムの開発手順は図２－７に示す様にまとめられる。また表２－１は開発手順の各フェーズの作業既要とそのフェーズで作成される主なドキュメントを一覧するものである。これらはソフトウェアを中心にまとめてあるが，ドキュメントは各作業フェーズの結果出力であり，次のフェーズの入力である。これらのドキュメントを整備することがシステム開発であるとも言える。以下各作業フェーズについて順次説明する。

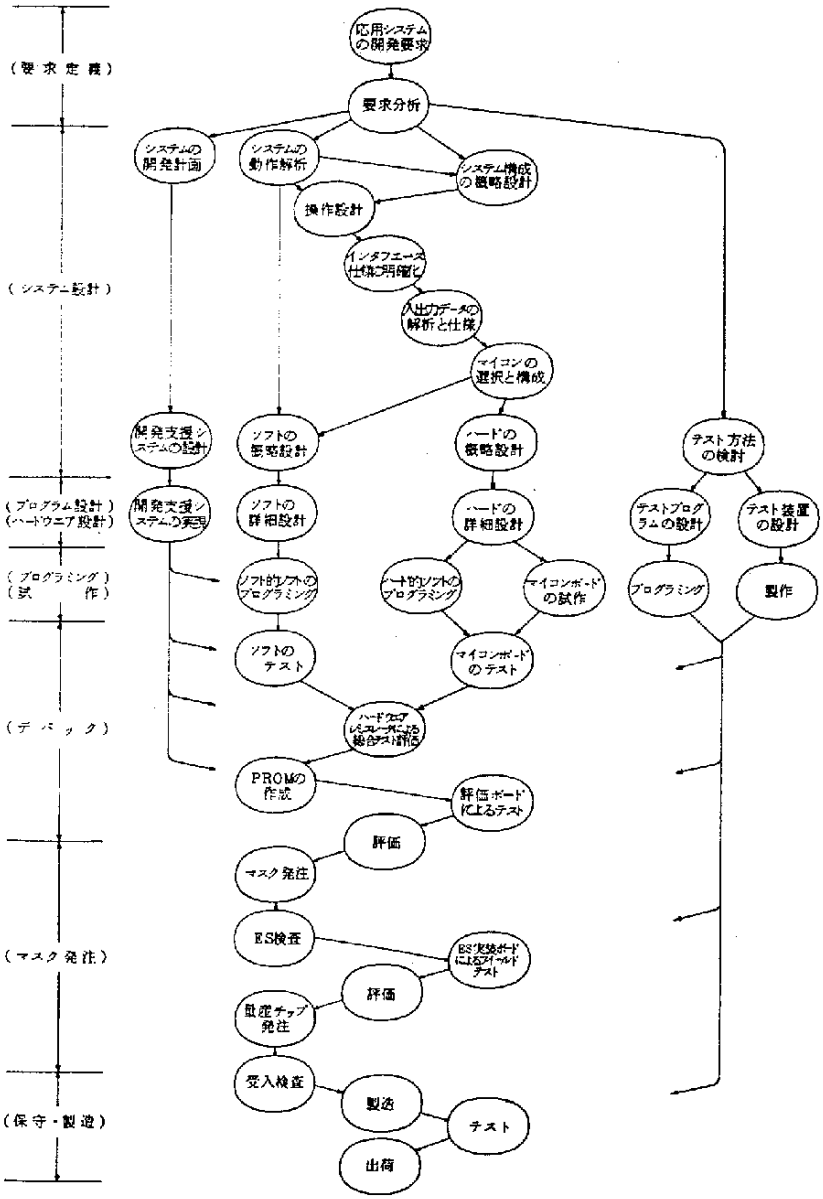


図２－７　ワンチップマイコンのシステム開発の流れ

表2－1　各フェーズの作業内容と文書

フェーズ	内容	文書
要求定義	・機能分析調査や技術調査を行ない、全体戦略と方針の明確化を行なう。	システム仕様書 システム開発の背景・時期・予算 システム開発目標 機能分析表 他社製品比較表 市場動向分析表 技術文献一覧表 特許調査 将来展開構想
システム設計	・製品化の要求、開発費、開発要員、企画台数、納期などの要因を考慮し、システムの開発計画を立てる。 ・システムのハードウエア、ソフトウエア全体についての概略設計を行ない仕様を決める。 ・試作機や検査用治工具、検査プログラムの必要性について検討する。	開発計画書 開発計画 開発担当組織図 開発支援システム ハードウエアシステム仕様書 システム構成 インタフエース仕様 入出力端子割当表 マイコン構成図 ソフトウエアシステム仕様書 システムの動作概要 操作フロー ソフトウエアモジュール構成図 テスト概略仕様書
プログラム設計	機能ブロックをモジュールのレベルに分解し、その仕様を作成する。	モジュール構成図 モジュール説明書 プログラミング規則 擬似コーデイング・詳細フローチャート
プログラミング	コーデイング アセンブル	ソースリスト
デバッグ	モジュール、機能ブロック、システムのテスト・デバッグ	テスト結果表
製造・保守	機能の変更・追加	システムの評価

(1) 要求定義

市場動向や他社動向などからまとめられた製品企画に基づいて，その要求機能を具現するためのシステムの仕様を明確にする作業が要求定義である。

(a) 調査分析

調査分析は応用製品／システムの機能分析と種々の技術調査をなすことである。機能分析はニーズの調査であり，開発すべきシステムが持つべき機能を分析確定することであり，技術調査はシーズの調査であり，他社の類似システムおよびこのシステムを構成する技術要素に関する調査である。

ここで作成されるべきドキュメントは次のとおりである。

① 機能分析表
② 他社製品比較表
③ 市場動向分析表
④ 技術文献一覧表
⑤ 特許調査表
⑥ 将来構想

(b) システム仕様

システム仕様書はシステムをブラックボックスとみなして外部条件を明確にするもので，システムの外部仕様である。この仕様に記述されなければならない事項は

① 背景

システム開発の契機と経過

② 時期・予算

システムが完成すべき時期とコスト

③ 機能・性能

操作機能，システムの有すべき機能性能

④ 開発目標

品質目標，発表価格，保守，商品寿命

(2) システム設計

システム仕様に基づいて，この仕様を実現するためのアルゴリズム，機能要素を設計するフェーズでソフトウェアとハードウェアの両面からのアプローチが必要で，特にワンチップマイコンでは両者のトレードオフが極めて重要である。従って，ハードウェア，ソフトウェア両面に長けた設計者が担当すべきである。

まず，製品のシステム仕様を検討して，ワンチップマイコンを適用すべきか否かを決定する。このとき，量産性・保守性・経済性・開発期間なども配慮すべきである。ワンチップマイコンの適用が可となれば，システムの動作概要，システムの概略構成をまとめ，操作仕様・入出力要素を定め，ワンチップマイコンの入出力端子の割当て，ROM，RAMの所要サイズの推定などから採用すべきマイコン（シリーズおよび品種）を選定する。ここで選択されたマイコンのハードウェア構成に基づいてシステムのハードウェアと機能ブロックの概略設計，ソフトウェアのアルゴリズム，機能ブロックの概略設計を実施する。この作業フェーズで作成されるシステム動作概要，操作フロー，入出力データ，プログラムのモジュール構成図がソフトウェアシステム仕様書であり，システム構成図，インターフェイス仕様，入出力端子の割当て，マイコンの構成がハードウェアシステム仕様書である。

このシステム設計には，システム仕様に記述された要求定義，開発費，開発要員，企画台数，出荷日程などを配慮して，システムの開発計画を立案することも含まれる。また，開発の生産性をよくするため開発支援ツール，検査用治工具，検査用プログラムなども検討しておくべきである。

(a)　システム動作概要

システムは一般に図２－８の様に人間と被制御系と制御系との間の関係である。人間系は制御系に対し操作／指令を与え，また制御系からの情報を受けとり，被制御系は制御系に連結されている制御対象であり，制御系はインターフェイスとマイコンとから成る。

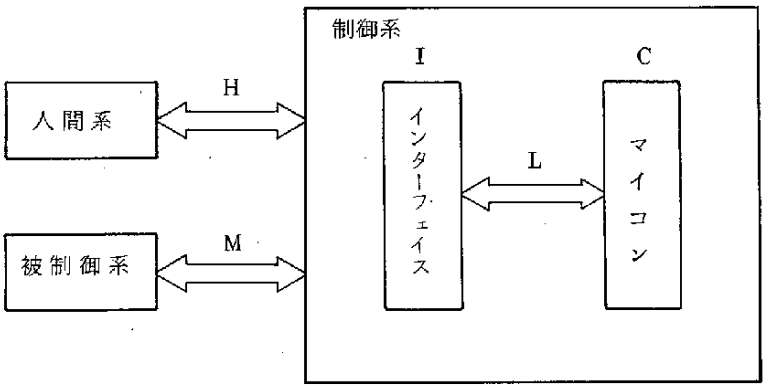


図２－８　システム要素

H：人間がシステムをどのように操作するか，システムがどのように応答するかについての手続き

M：制御される対象とインターフェイスとの間の手続き。

I：センサ，アクチュエータ，表示部品，キー，信号変換（A／D，D／A，レベルシフトなど），マルチプレクサなど，人間系・被制御系とマイコンとの結合をなす回路ブロック

L：インターフェイスとマイコンとの間の情報，制御信号，タイミングなど

C：ワンチップマイコンでありＩを内蔵する場合が多い。

システム動作概要とはＨとＭとの総括的な説明記述であって，箇条書または概略フローで表現される。箇条書の場合

①　動作順序を示す番号の付与

② 人間系と制御系の対話（操作手順）の列挙

③ 操作手順に関するやりとり情報

④ 制御系の所要機能の列挙

が必要である。概略フローの場合は図２－９の例のように人間系または被制御系と，制御系とに分けて，全ての動作を記述する。

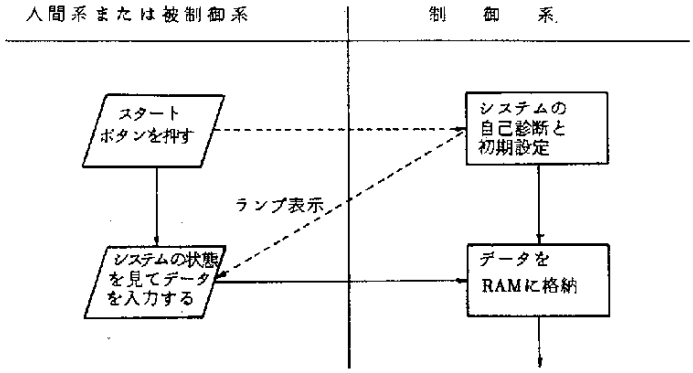


図２－９　動作概略フローの例

さらに人間との対話H，被制御系の制御手順Mおよびマイコンのソフトウェア（プログラム）を詳細に記述するため図２－１０の如き操作フロー図を用いるとよい。

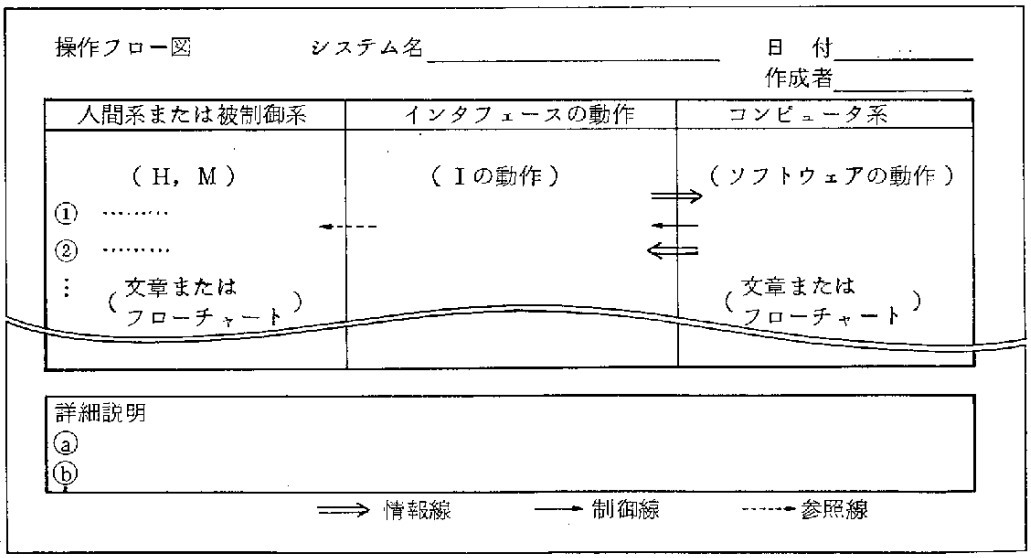


図２－１０　操作フロー図

特にワンチップマイコンの応用製品は不特定多数の使用者を対象とするため，設計時に次の点に留意すべきである。

① 誤操作が致命的な誤動作にならないこと

② 同一カテゴリの操作方法に統一性を持たせること，特に，共通的な操作方法は同一とすること

③ 製品シリーズ内での操作方法の統一は重要で，機能の少ない機器の操作は，より機能の多い機器の操作のサブセットになっていること

④ 誤操作・瞬時停電・サージなどによって安全上の問題が生じないこと

また，表２－２に操作性のチェックポイントを掲げる。

表２－２ 操作性チェックポイント

入力部	□ キー，スイッチの位置・配列は適切か □ キー，スイッチの名称・表記は一貫しているか □ 通常操作しないスイッチ数は容易に触れられない構造か □ 操作手順・流れが表示されているか，自然か
表示出力部	□ 表示は見やすいか □ 表示用語は一貫しているか □ エラーや停電復帰などの表示は理解しやすいか □ 不要な表示がないか □ 登録内容の確認表示は可能か □ 表示の明るさは適当か □ 無意味な点滅はないか

(b) インターフェイス仕様

インターフェイス部とマイコンの入出力仕様と性能について記述するのがインターフェイス仕様である。ワンチップマイコンの適用システムでは入出力端子の割当てもこの部分に含める。ワンチップマイコンは，各種の目的に適するように，並直入出力（４ビット，８ビット），ディスクリート入出力，ストローブ入出力，アナログ

入出力など多種の入出力端子を持っているので，これらの特性を十分配慮した割当てが重要である。

(c) マイコンの選択

マイコンの選択条件は図２－１１に示す様に，技術的要因だけでなく，社内外の条件，環境なども重要である。

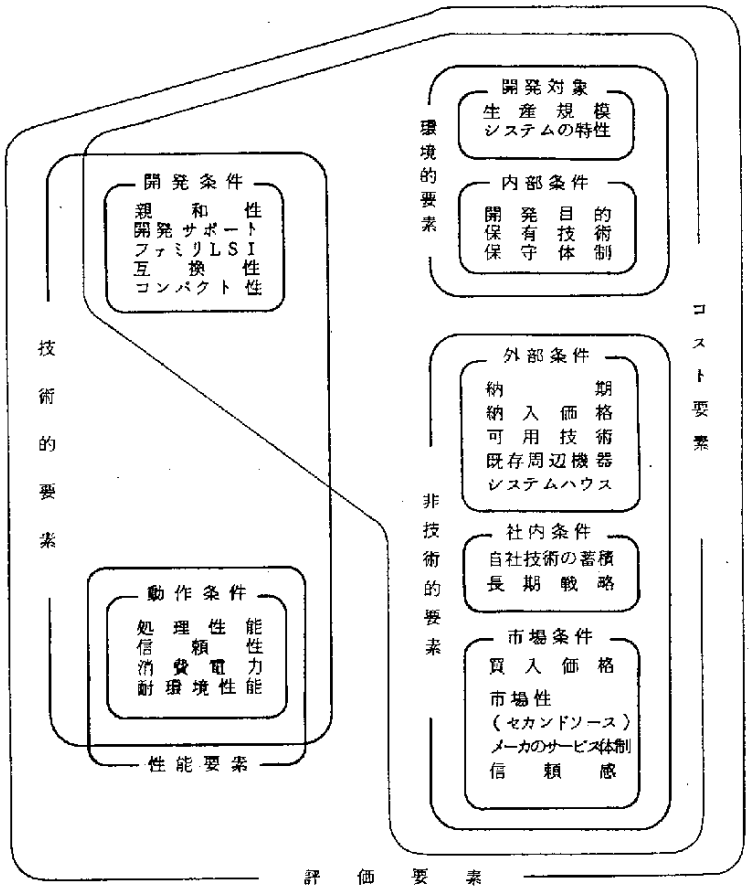


図２－１１　マイコンの選択条件

この図とは別の切り口から見れば次の項目にまとめられる。

① 世評

② 利用経験（親近感，技術蓄積）

③ 開発支援（ツール，サービス，マニュアル，サービス）

④ 政治（方針，セカンドソース）

⑤ 性能・仕様

(d) ソフトウェアモジュール構成図

操作フローに従って機能分析し，必要なプログラムモジュールに分割して階層図を作る。これによってプログラムの構造やプログラムモジュールの数などを明確にする。

(e) 開発計画

この作業フェーズでは開発工程，開発組織，開発環境を明確にする。

① 開発計画 ― 各システムブロックをパート図かガントチャートに並べ日程を書き込む。開発のクリティカルパスを明確にし，各開発段階で即実施すべきことと次の工程のための準備事項を明示することが目的である。

② 開発担当組織 ― 開発担当者を決定し，各担当者の役割を定め，責任と権限を明確にする。システム開発の分担は縦割り組織（サブシステム単位に担当を定める方法）と横割り組織（各職種別に担当を定める方法）があるが後者が一般的であり，通常

- システム設計担当
- ハードウェア担当
- プログラム開発担当

に分れる。

③ 開発環境説明書 ― システム開発のための人的環境や物的環境を明確にするもので主な環境としては開発支援ツールとテスト環境があげられる。マイコンサポートシステムとしてのクロス系システム，ROMライタ，ロジックアナライザ，ICEなどハードウェアと開発用OS，ユーティリティ，組込みモニタ，エディタ，デバッガなどソフトウェアがある。後者にはテスト用専用システ

ム，自動テスタ，保守用ツールなどがある。

(3) プログラム設計

ソフトウェアシステム仕様書に記述されている機能ブロックの機能・性能を実現するためのアルゴリズムの設計，プログラムモジュールへの分割，モジュールの機能・性能仕様とモジュール間のインターフェイス仕様の設計などを含む作業フェーズである。信頼性向上のためのアルゴリズム，テスタビリティを実現するための方策などもこのフェーズでモジュール仕様の中に含めなければならない。従来とかくシステムの要件・目標・性能等を明確にせず，日程の制約でとりあえずスタートすることがあったが，途中での仕様変更，インターフェイス条件の変更などが発生して，かえって開発期間を長くしていた。要求仕様を明確化するため，ドキュメント化の手法が重要となる所である。

大型計算機用のソフトウェア開発の手法がマイコンソフトウェア開発に全面的に採用できる訳ではないが，構造化設計の手法などは大いに参考にすべきである。ワンチップマイコンの場合はソフトウェアの規模は小さいので，設計要員を出来る限り少くして思想統一を重点とする方がよい。また，ここではオブジェクトの生成効率や実行効率が極めて重要となるので，開発言語はほとんどの場合アセンブラが使用されるが，可能な限りマクロ語を追加して，マクロアセンブラを利用出来る様にするとプログラミングの効率はよくなる。

要約するとプログラム設計は

① 機能ブロック内のアルゴリズム設計

② 信頼性機能の機能ブロックへの組込み

③ 機能ブロックのプログラムモジュールへの分割

④ プログラムモジュールの機能・性能仕様の設計

⑤ プログラムモジュール間のインターフェイス仕様の設計

⑥ プログラムモジュール内のアルゴリズムの設計

⑦　開発言語の選択

を成すことである。

(4)　プログラミング

プログラミングはプログラムモジュールの機能・性能仕様書，インターフェイス仕様書，アルゴリズム仕様書などに基づいてプログラムを作成するフェーズである。この作業では

- コーデングシート
- ソースプログラム
- オブジェクトプログラム

が得られる。

プログラム作成には汎用計算機によるクロスコンパイラなども利用されるが，実時間による動作確認やデバッグの容易な様にスタンドアロン型の開発サポートシステムが用意されている。

(5)　ハードウェア設計

ハードウェアシステム仕様書に基づいて回路設計を実施する作業フェーズであるが，ワンチップマイコンの場合はプリント基板設計までを含む。回路設計時に配慮すべきチェックポイントをいくつかあげると次のとおりである。

①　要求される信頼度，設計された回路の予想信頼度はどれほどか

②　標準／推奨回路を利用しているか

③　代替可能な実績確認済の回路はないか

④　新規／特殊回路はあるか，あればそれが選択された理由はなにか

⑤　回路上のスイッチ，ボリュームなど可変部品は固定できないか

⑥　回路を単純化できないか

⑦　入力信号の変動予測値に耐えられるか

⑧　入出力インピーダンスの変動予測値に耐えられるか

⑨　入出力の束線長はどう見積ったか

⑩　電源はどうか（リップル，電圧変動，電流容量，ノイズなど）

⑪　寄生発振はないか

⑫　異常に対する防護機能はあるか

⑬　最悪環境条件を考慮したか

⑭　部品のバラツキを考慮したか

⑮　フェイル・セーフになっているか

⑯　故障は見つけやすいか

また，試験や保守の作業を容易にするために配慮しておくべき事項は次のとおりである。

①　順序回路では初期パタンの設定が容易にできること

②　プリント基板端子やマイコンの入出力ピンは可能な限り診断用に用意すること

③　試験装置の扱いやすい設計にすること

④　マルチバイブレータなど自律的に動作する回路，ＲＯＭ，ＲＡＭなどは単独にテスト可能にしておくこと

⑤　誤操作に対しての保護を考えること

⑥　プローブなどサービス治具が使用できるスペース・構造を配慮すること

⑦　フィードバックループやバスラインがテストのために切り得ること

⑧　接続束線の長さはサービス時の必要長も配慮すること

⑨　ボードの独立性，完結性を保つこと

⑩　予めテストのためのプログラムを組み込むこと

(6)　デバッグ／テスト

プログラムを実行させながら，仕様に合致しない不都合を発見し修復する作業である。

要求定義からコーディングまでのソフトウェアの開発過程では抽象的かつ概括的であったものが次第に具体的かつ詳細となる。デバッグはこの開発過程を逆にたどることになる。図２－１２は作業の流れとデバッグの関係を示す図である。図中の「評価」作業は作業の都度実施するデバッグで次のフェーズにバグが波及しないようにするための作業であるが，場合によってはプログラム設計，システム設計さらには要求定義のフェーズまで戻らねばならないこともある。表２－３はバグの原因とそれが発見される時期を示した例である。この表で組立て時とは機能ブロックのテストおよびシステムテストのフェーズも含まれる。コーディングのバグの3/4はコーディング時に発見されるが，設計のバグの2/3は組立て時まで発見されないことが判る。図２－13は設計不良の原因を大きい順に並べたものである。設計仕様を十分検討し注意深くコーディングすれば８０％の不良は無くなることを示している。

表２－３　バグの原因と発見時期

原因 ＼ 発見時期	設計 ６４％	コーディング ３７％
コーディング時 ４６％	１９％	２８％
組立て時 ５４％	４５％	９％

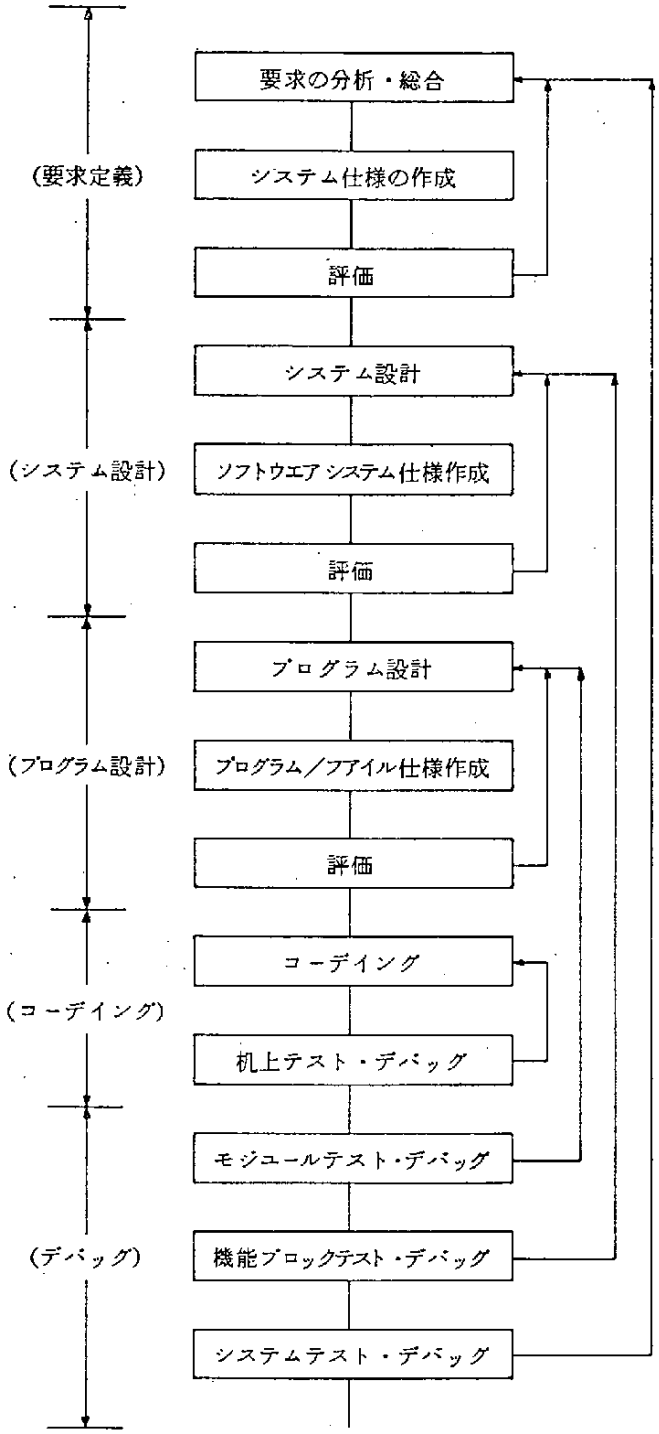


図２－１２　ソフトウェアのテスト／デバッグ

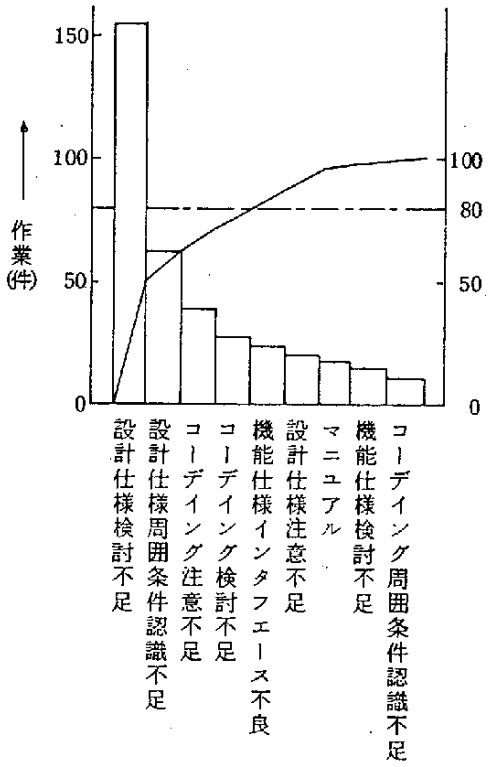


図２－１３　設計不良の発生原因（例）

(7)　保守

一般のマイコンシステムでは，保守は取り残したバグの除去あるいはユーザからの新しい追加要求に対応する性能や機能の変更や追加などである。この場合は，優れたソフトウェア開発が保守費の少ないソフトウェアの開発と等価となる。

しかしワンチップマイコンの場合は，プログラムがチップ上のＲＯＭにＬＳＩの製造工程で書き込まれ，後日の改変は不可能であるから，前述の意味でのソフトウェアの保守はあり得ない。

2.1.3 開発サポート

マイコンシステムの開発はハードウェア開発とソフトウェア開発に分け得るが，ワンチップマイコンの場合は，特に動作確認が両者を分離して実施することが極めて困難である。このため，専用のデバッグツールが用意される。また，マイコンチップの動作確認もプログラムが内蔵され，外部からのメモリアクセスができないことを考慮する必要がある。以下，開発工程で利用される各種のツールを紹介する。

(1) シミュレータ

ミニコンや汎用マイコンを利用して，対象とするワンチップマイコンと等価な機能を有し，同等の動作を行うシステムが用意される。これを利用してオブジェクトプログラムを実行し，応用製品の機能を確認したり，デバッグしたりできる。しかし，これはワンチップマイコンの動作を時間順序は模擬できるが，実時間での動作が難しいので，タイミングが重要となるアクチュエータやディスプレイ，キー入力などを接続して製品の動作を行うことはできない。

このため，ワンチップマイコンの等価機能を汎用ＬＳＩやＩＣを組合せて構成し，マイコンの実時間動作の模擬とデバッグを可能とするものが用意される。また，ＲＯＭ外付けの評価用チップ（エバリュエータ）を利用して上記のものを構成することによって入出力の電気的条件をマイコンと同一にするシミュレータもある。このシミュレータは次の様な機能を持っている。

① クロスアセンブラなどで出力されたオブジェクトプログラムをロードできる。

② スイッチレジスタによってデータ／命令メモリの任意の番地に任意のデータ／命令を書き込むことができる。また読出しができる。

③ プログラムを実時間で実行する。

④　プログラムを１命令毎に実行する。

⑤　プログラムの実行を予め設定した番地で停止できる。

⑥　入出力の状態，内部レジスタ／フラグ，実行番地，実行命令等を表示する。

⑦　命令メモリの内容（オブジェクトプログラム）が出力できる。

また，再書き込み可能なＲＯＭ（ＥＰ−ROM）で命令メモリ部を構成した評価チップが用意される場合もある。この場合は実サイズの最終製品での評価が可能となり，商品モニタへの提供が可能となる。

(2)　ロジックアナライザ

入出力ポートやステータスピンなどの論理状態の動きを追求する計測器である。表示のためのＣＲＴを有し，複数チャネルの論理波形出力や，論理値を確認することができる。

(3)　支援システム

アセンブリ言語で記述されたソースプログラムを機械語（オブジェクトプログラム）に変換するソフトウェアを汎用コンピュータやミニコン，汎用マイコンシステムで実行し，アセンブルしたりシミュレートしたりするシステムである。また，評価チップと組合せて利用されるＰＲＯＭを書き込むＲＯＭライタが接続されているシステムも多い。

図２−１４はスタンドアロン型の支援システムの例で

①　言語処理からデバッグまでできる。

②　高度なデバッグ機能が利用できる。

③　複数のシステムを同時に開発できない。

④　周辺装置の稼動率が低い。

⑤　複数ＣＰＵシステムのデバッグができない。

などの特徴がある。

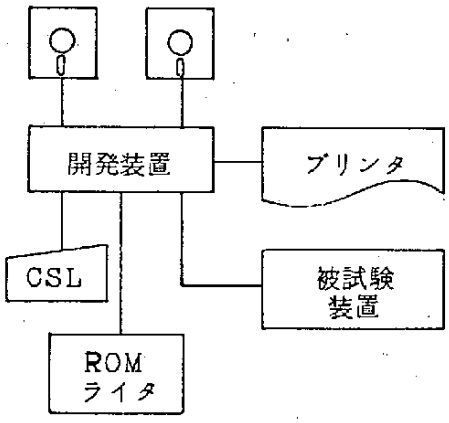


図２－１４　支援システム例１

図２－１５はミニコンや大型コンピュータを利用するシステムで次の様な特徴を持っている。

① 汎用ＣＰＵで言語処理ができる。

② 同時に複数のシステムを低コストで開発できる。

③ 複数ＣＰＵシステムのデバッグができる。

④ 言語処理装置が専有できない。

⑤ 複雑なバグの追求は難しい。

⑥ デバッグのターンアランドタイムは言語処理系で定まる。

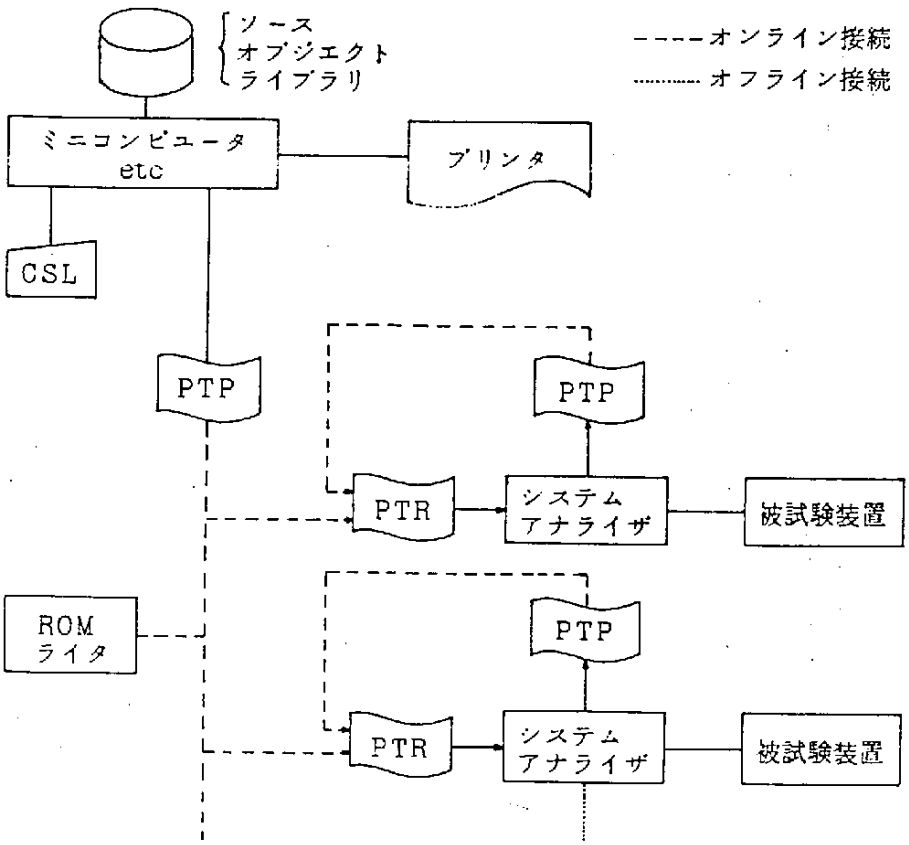


図２－１５　支援システム例２

さらに図２－１６は専用支援システムの例であり，その特徴は次のとおりである。

① 同時に複数システムがデバッグできる。

② 複数なバグに対処できる。

③ 周辺装置の稼動率が高い。

④ デバッグのターンアランドタイムが小さい。

⑤ 複数ＣＰＵシステムのデバッグができる。

⑥ システムが高価である。

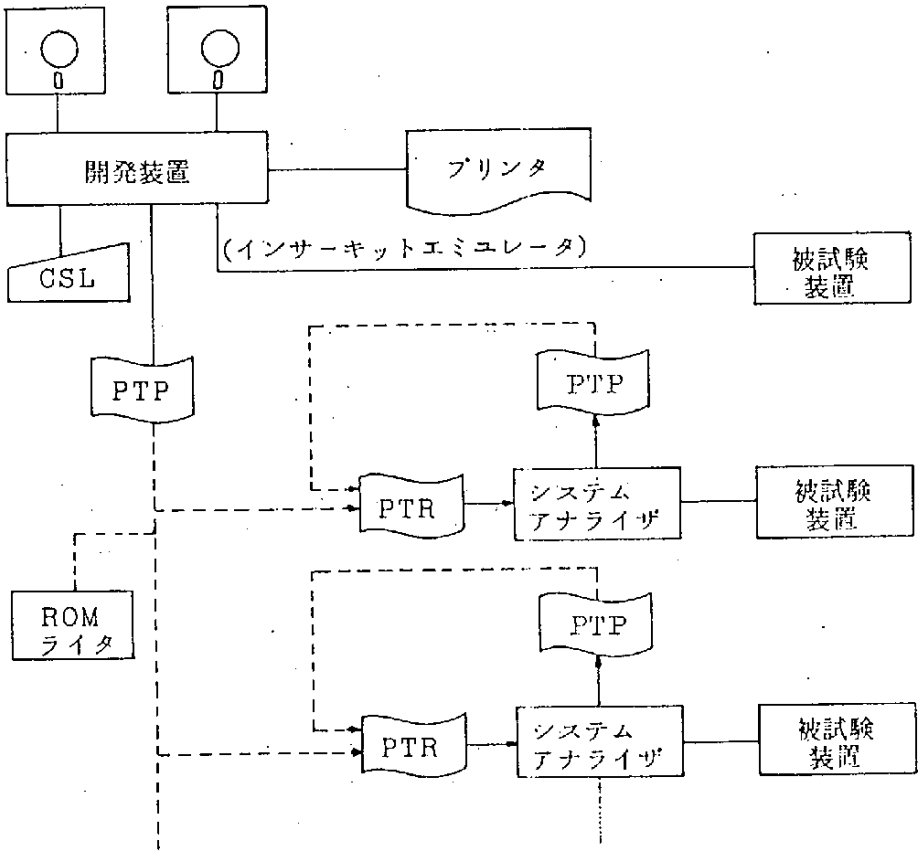


図２－１６　支援システム例３

(4)　言語処理系

ワンチップマイコンではオブジェクト効率や実行タイミングが極めて重要であるので，アセンブリ言語でプログラミングすることが一般的である。しかし，最近では記述性やメンテナビリティを大切にして，ワンチップマイコン用高位言語が東芝や松下で開発された。（松下の高位言語ＣＬ／１については第４章トピックスで紹介している）

(a)　汎用コンピュータ

処理能力も大きく高速な汎用コンピュータを他の業務と共用でマイコンソフトウェアの開発に利用するものである。クローズドショ

ップ制で運用されている場合はデバッグのターンアランドタイムが長いので効率よく利用する工夫が大切である。

(b) ミニコンピュータ

マイコン専用の言語処理系としてオープンショップ制で利用されるのが一般的である。処理能力も専用であることから大型機と遜色はない。

(c) ＴＳＳ端末

少ない投資でセンタマシンの機能が利用できるが，通信速度による制約は免れない。多種多様なマイコンに対する言語処理，シミュレーションなどが用意されており幅広いサービスが期待できる。

(5) 動作の検証

ワンチップマイコンはその入出力ピンの機能が内蔵プログラムによって定まるため，汎用マイコンの様にチップ検査機が利用できない。チップメーカでは，内蔵ＲＯＭの読出しや命令ジャミングによる内部ブロックの検査をしているが，この手順はユーザのソフトウェア保護の見知から公表しない。従って，ワンチップマイコンの受入側では次の様なチップ試験方法が採られている。

(a) 実装試験

機能や性能が確認された製品または等価モデルに被検チップを実装して動作を確認する方法である。この方法は，特別の検査装置を必要としないが，製品を実際に動作させる訳であるから全ての機能を確認するには長い時間を要する場合があり，テストの手順を工夫することが重要である。

(b) 比較試験

予め機能が確認されたチップと被検チップとに同一の入力条件を与えて動作させ，両者の出力を電気的に比較する方法である。この試験装置はこれらの入力条件の設定がプログラムでき，汎用的にな

っている。検査時間はタイマなどが組込まれていると出力が定まるまでに長時間を要し，また，入力数が多くなると入力パタン数が膨大となって，完全なテストが出来ないこともある。

(c) 自己診断

ワンチップマイコンの内蔵プログラムの一部として，機能確認のための診断プログラムを組込んでおいて，製品に適用したときには発生しない入力パタンを診断プログラムへの分岐スイッチとし，特定出力ポートへの出力パタンとして診断結果を出力する方法である。プログラム容量の小さいワンチップマイコンに全ての診断プログラムを組込むことは不可能であり，この方法は極く一部の重要な機能だけの試験に限定される。

(d) 専用試験

応用製品毎に用意される専用試験装置と結合するのに必要なプログラムだけをマイコンチップに組込んでおき，このプログラムを専用試験装置で利用して動作を確認する方法である。この試験装置は応用システムの設計時に応用システムと同一思想で設計されなければならない。

2.1.4 課　題

ワンチップマイコンはコストプッシュが厳しく，使用数も多いので，周辺回路も可能な限り取込みカスタムマイコン化する傾向にある。マイコンを利用した製品は多様で高度な機能を有するので，故障時に同じ症状を呈していても，その原因は全く異なっていることが多く，ＬＳＩ故にユニット交換によるサービスを実施せざるを得ない。ワンチップマイコンはプログラムを内蔵しているため，機器間，モデル間でユニットに互換性が無いため，高価な保守モジュールを多種保有する必要がある。また進歩の速い半導体分野で１０年以上も耐久性のある製品の保守パー

ツとしてマイコンが入手可能かどうかも大きな関心事である。

開発上の課題としては，まず開発技術者の育成・確保がある。ワンチップマイコンの応用の様に，ハードウェアの代替と考えられるソフトウェアは「ソフト屋」と通称される情報処理技術者では上手に作れない。「ハード屋」と通称される回路技術者がソフトウェア技術に通じていくことが不可欠であり，ハードウェアとソフトウェアのトレードオフなども重要である。次に周辺部品，特にセンサ類の充実が待たれる。マイコンのコストに比較すると，使用可能なこれら周辺部品は少なく，これらを充実し安価にすることが応用製品を展開する上での必須条件となる。さらに，家電機器や玩具の様に，商品のライフサイクルが短い場合，プログラム開発のための期間短縮が重要となる。

参考文献

(1) 森　亮一編：マイクロコンピュータハンドブック（朝倉書店）1979年

(2) 藤井克彦編：ミニコン・マイコンハンドブック（朝倉書店）1982年

(3) 日本電子機械工業会編：マイクロコンピュータ応用システム

〔Ⅰ〕ハードウェアの生産技術編

〔Ⅱ〕ソフトウェアの生産技術編

（エレクトロニクスダイジェスト）

2.2　計測・制御システム

計測・制御分野におけるマイクロコンピュータを応用したシステムの開発の事例を本節で紹介する。

石油工業，化学工業，ガス，電力，鉄鋼，製紙，食品などのプラントを管理およびコントロールをするのに，過去においては，プラントの局所的な部分に対する制御性をいかに向上させるかが課題であり，これを満すためにＰＩＤアルゴリズムを内蔵したプロセス用の制御機器（工業計器）の高機能化，高信頼化がメイン・テーマとして取上げられる時代があった。

局所的な制御性の向上のみでは，生産品質の高位化，プラントの経済的な運営が実現できず，プロセス用計算機の導入により，それら問題の解決を図る時代へと変化した。

しかし，機器の信頼性，導入コスト高などから，プラントの最適化運転がある程度可能とはなったものの，必ずしも満足のゆくシステムではなかった。

1970 年の中頃から，マイコンを組込んだ制御機器が開発され，分散型制御システムとして発表された。図２－１７にそのシステム構成例を示す。

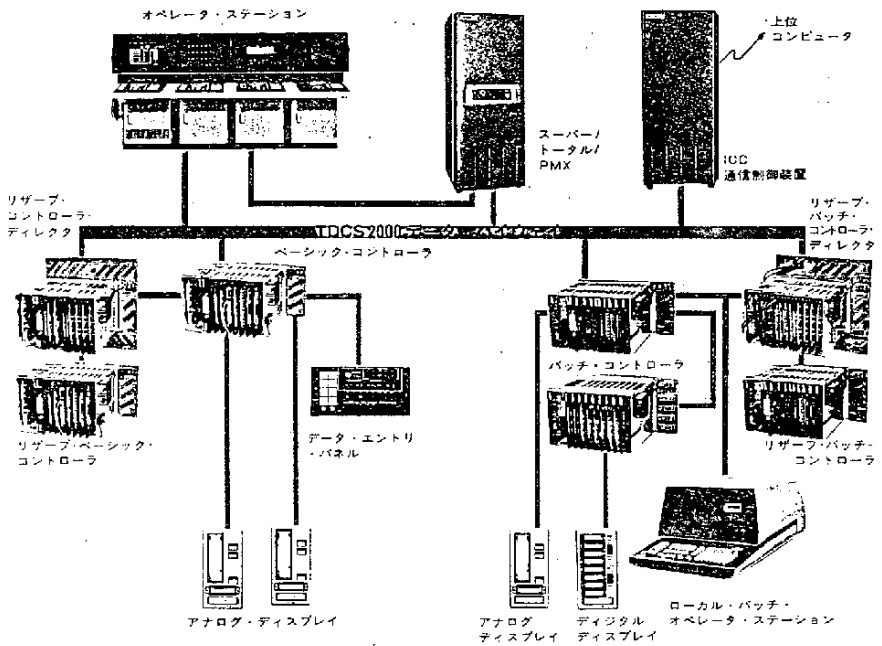


図2－17　TDCS2000全体システム

これは，従来のアナログ型制御機器とプロセス用コンピュータが処理していた分野をマイコンベースの機器で置換え，計装システムのコスト低下，信頼性の向上，ユーザシステム構築の容易性などの優位性から，プロセス制御分野において広くゆき渡り，大方の好評を得た。

しかし，なお図２－１７に示すように，生産管理レベルの作業処理をする機器として，スーパ・バイザリのプロセス・コンピュータ（多くはミニ・コンピュータ）があり，この位置におけるマン・マシンの両立性（オペレータ・ステーションとの表示・操作上の互換性），計装システム・コスト，信頼性などに改善すべき要素が内在していた。

そこで，これら要求に答える形として，生産管理レベルを包含したシステムの構築をした山武ハネウエルのＴＤＣ３０００を例にとり，マイクロコンピュータ応用例を解説する。

2.2.1　要求仕様

どの製品，システムでも求められる要素としてコスト，パフォーマンス，リライアビリティ，アベイラビリティ，サービスアビリティ，インテグリティ，セキュリティなどがある。これらの要因のどの点に力点を置きシステムの構築を図るかは，適用システムの産業分野における位置，アプリケーションの適用対象により，その様相が異なる。例えば，人工衛星や深宇宙探査機に搭載されるコンピュータと，ホビー用マイコン機器とでは，上述した要求要因の優先順位は異なってくる。

今回のプロセス制御システム，プラント情報管理システムでの要求仕様の基本は，リラリアビリティとフレキシビリティの２つである。

リライアビリティの高いシステムを実現するには，フォルト・トレラント（FAULT TOLERANT）技術を採用するものと，システムを構成するコンポーネントの固有の本質的な信頼性を高めることによりシステムの信頼性を向上させる方向との２者が存在する。要求仕様としては，

コストや保守性などの観点から，この2者のバランスをどう配分するかが要点となる。

一方，フレキシビリティとは，システム構成上の小システムから大システムまでに至る物理的，機能的な大小が価格に影響しないことであり，システムの故障に対しては，システム・ダウンに至らない柔構造のハード，ソフトを持つことである。

このような要求仕様は体系的に整理される必要がある。後述する実現仕様に対応する形で要求仕様が存在する。すなわち，全体システム・レベルでの要求仕様，サブ・システム・レベルでの仕様，サブ・サブ・システムでの仕様という形のハイアラキの要求仕様の構成となる。これら要求仕様は要求仕様書（REQUIRED DOCUMENT）として記述される。

記述される内容は，開発を要求する製品の仕様（機能，性能，環境条件など），価格，希望開発スケジュール，製品を必要とする背景などが記述される。要求仕様にはその要求の優先度が，3段階程度に分けて付けられる。

本要求仕様書の記述の適確さ，つまり要求範囲の明確さ，要求内容の透明度が高い程，実現仕様作成が容易となり，開発工程の短縮が図れるので，要求仕様書の持つ意義は重要である。更に付言すれば，要求仕様で求められる事項として，無矛盾性，両立性などがある。これらを欠く場合には，要求仕様の中におけるバグとして設計仕様に反映される以前に除去されねばならない。

システムが大きくなれば，サブ・システム間での統一性が，保たれないままに要求仕様が決定され，それが設計仕様に影響した場合，上流に戻って変更・修正を要する。悪い例として，コーディングを経過し，テストの段階で，スペック上のバグを発見することもあり得るので，源流におけるスクリーニングは大事にすべきものである。

2.2.2 実現仕様

TDCS（TOTAL DISTRIBUTED CONTROL SYSTEM）3000の実現仕様について以下述べる。

TDCS3000は，プロセス工業における制御とそれらに関連する情報の管理を実行する総合分散型のシステムである。

(1) システムの特徴

① ローカル・コントロール・ネットワーク（LCN）と呼ばれる高速のシリアル・バス上に接続される各ノードに，機能を分割化（水平分散化）することにより，高性能システムの実現を可能とした。

② プラントの制御をする上で，3つのレベルに階層化し，それらに対応した，ハード，ソフトの体系的な形態をとっている。

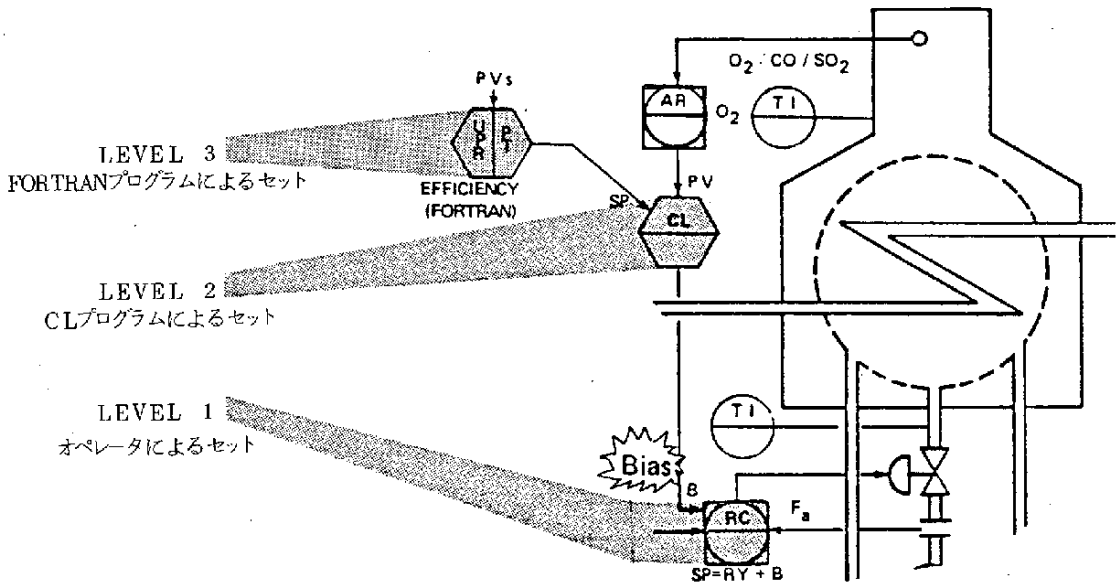
3つのレベルとは次のとおりである。

レベル1は，直接制御を実行するレベルを言い，制御ループに対するアルゴリズム制御（例えばPIDのDIRECT DIGITAL CONTOLなど），非制御ポイントに対するデータの収集とその1次処理（フォーマット変換など）を実行する。

レベル2は，ユーザの作成，記述するコントロール・ランゲージ（CL）・プログラムに従った制御を実行するレベルを言う。

レベル3は，ユーザの作成，記述する汎用言語（FORTRAN，PASCAL）のプログラムに従った制御を実行する。

レベル2のCLプログラムは，レベル1の制御系に対して関与し，レベル3のプログラムは，レベル2系に対して関与が可能であり，全体を通じてプラント・レベルの制御・管理を可能としている。図2－18にそのレベル間の関連を示す。



加熱炉のコントロールをレベル1，2，3にて実施例
レベル1では燃料の流量コントロール，温度のフィードフォワード・コントロール，燃料／空気の比率制御を行う。
レベル2ではCLプログラムによるフィードフォワード・コントロール，O_2 コントロールを実行する。
レベル3ではFORTRAN プログラムにより燃焼の効率化制御を実行する

図2－18　制御レベル間の関係図

③　ハイ・ディペンダビリティの実現

図2－19に示すように，ディペンダビリティとは，アベイラビリティ（稼動率），使い易さ，セキュリティから成るものと定義し，これらの要素を顕在化したシステムとしている。

(2)　ハードウェア構成

次に，TDCS3000 システムのハードウェア構成について記述する。図2－20にそのシステム全体図を示す。図2－21にその機能を分散した図を示す。

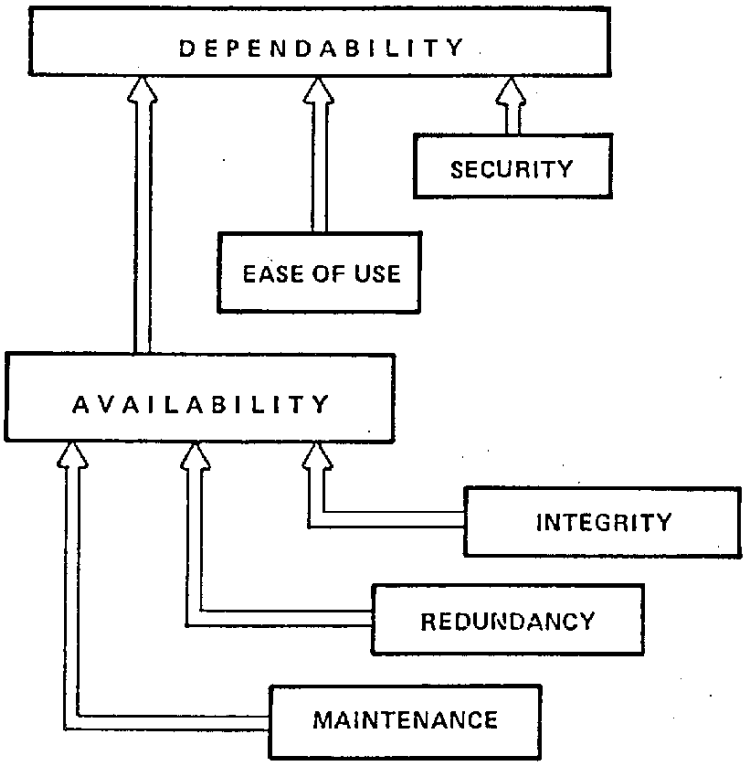


図２－１９　ディペンダビリティ説明図

ローカル・コントロール・ネットワーク上に接続されるそれぞれのノードは，ハードウェアの基本構成を同一の形態としている。しかし，該当するノードの種類に応じて，それを構成する要素（ＰＷＡ）の違いがある。つまり，シャーシ，電源，ＰＷＡ（ＰＲＩＮＴ ＷＩＲＩＮＧ ＡＳＳＥＭＢＬＹ）としてＣＰＵ，ＭＥＭＯＲＹ，ＬＣＮＩ／Ｆ は，どのノード上でも必須のハードであるが，マン・マシンであれば，ＣＲＴのコントローラが，通信ノードであれば，ＣＯＭＭＩ／Ｆがという形で，ノードのパーソナリティに応じてハードウェアが変る。

内部構成図を，図２－２２に示す。ＬＣＮ上に接続されるノードに使用されるＰＷＡと周辺機器は以下のものである。

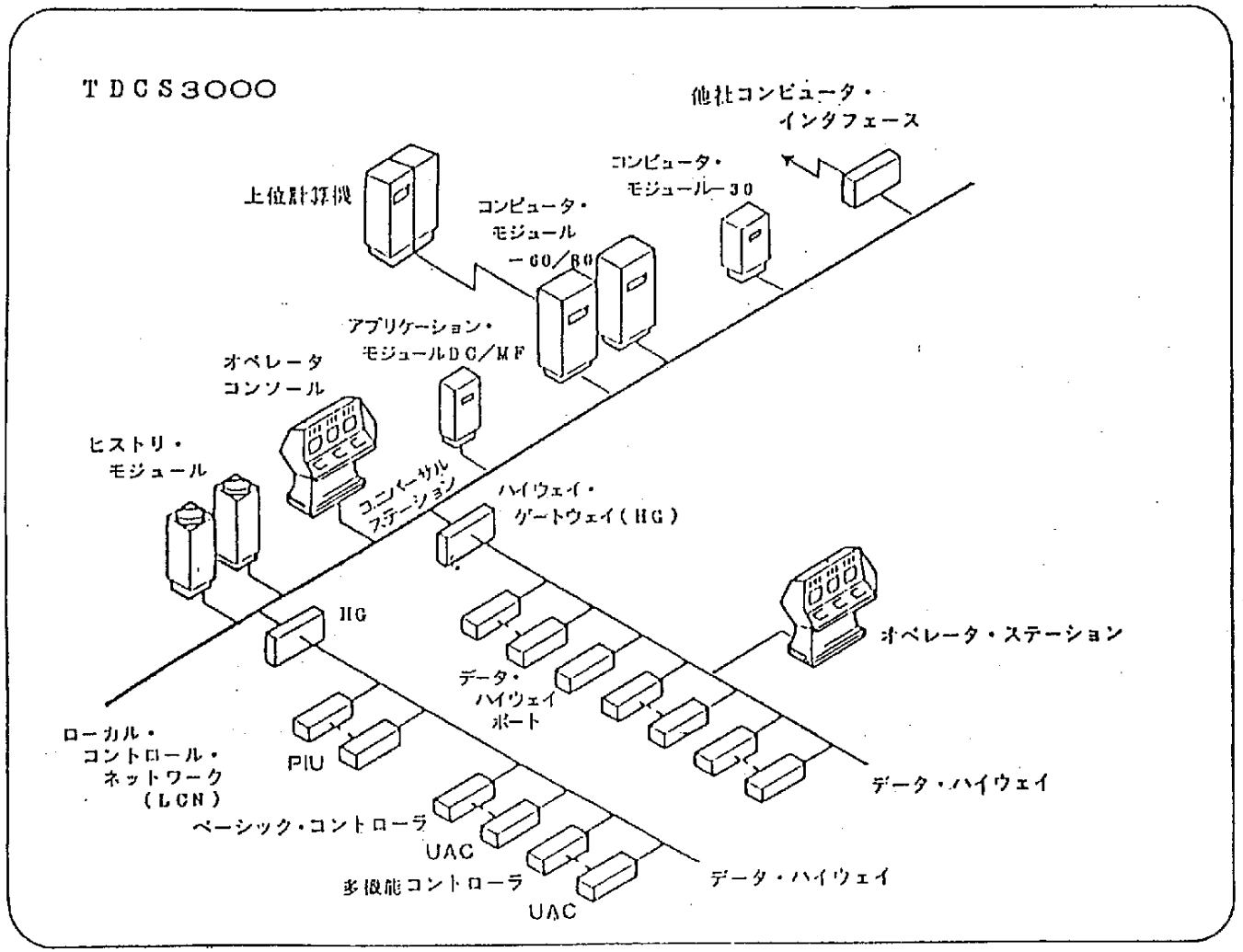


図２－２０　TDCS３０００全体構成図

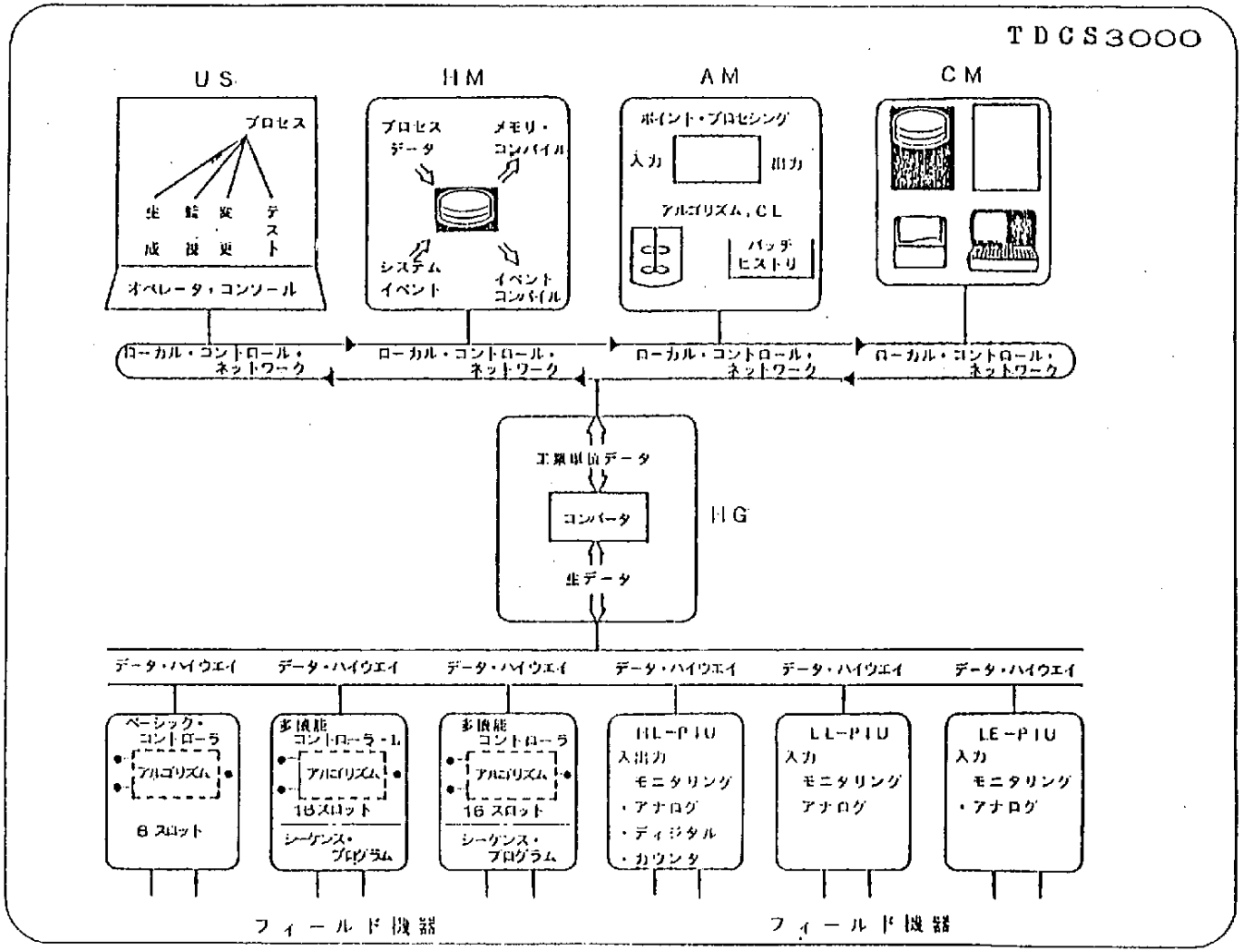


図2－21　TDCS3000全体機能図

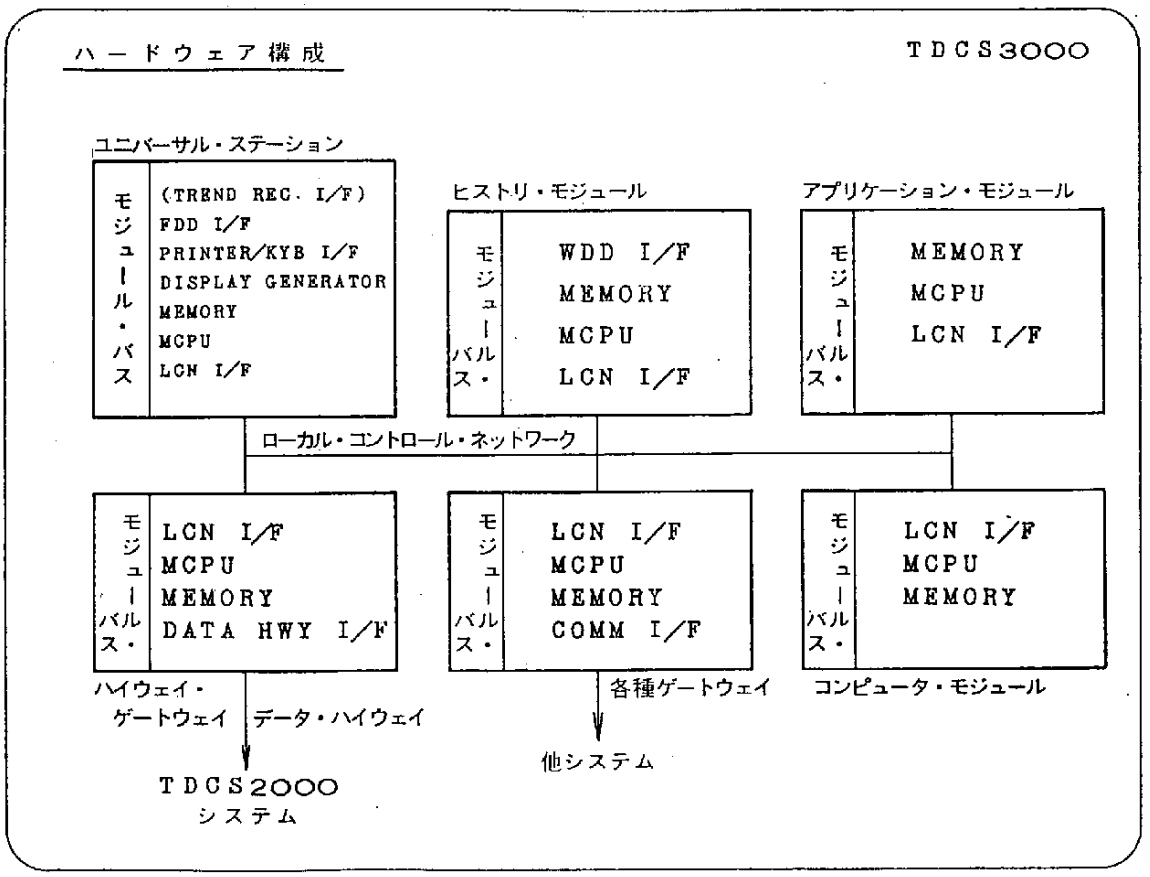


図２－２２　ハードウェア内部構成図

① MCPU
② MEMORY
③ LCN I／F
④ FDD I／F
⑤ WDD I／F
⑥ PRINTER／KYB I／F
⑦ DISPLAY GENERATOR
⑧ DATA HIWAY I／F

⑨ COMMUNICATION I／F

⑩ TREND RECORDER I／F

（周辺機器）

CRT，FDD，WDD，PRINTER，KEY BOARD，TREND RECORDER

次にそれぞれのノードの機能と，その構成について記述する。

(a) ユニバーサル・ステーション（US）

本システムの中心的な機器として位置しマン・マシン・インターフェイスの役割を司る。２０インチのカラーＣＲＴ上に，プロセスに関する状態の表示，プラント操作をするための画面，システムの機器の状態を知るための画面を提供し，運転員がこのＵＳを通して，プラントの操作，運転をする。ハードウェアは，ＣＲＴ以外に，オペレータ・キーボード，タッチ・スクリーン，プリンタ，フロッピ・ディスク，トレンド・ペン・レコーダが周辺機器として接続され，ＰＷＡは，基本ＰＷＡ以外に，ＣＲＴをコントロールするＤＩＳＰＬＡＹ GEN.，FDD I／F，PRINTER I／Fから構成される。図２－２３にコンソールに組込まれた図を示す。

(b) ハイウェイ・ゲートウェイ（HG）

データ・ハイウェイ上に接続される直接制御レベルの機器と，ＬＣＮ上のノード間で通信されるデータの変換や一時蓄積が実行される。

ハードウェア構成は，DATA HIWAY I／Fが基本構成に追加される。

(c) ヒストリカル・モジュール（HM）

本システムのマスメモリの役割を負う。連続またはバッチ・プロセスのヒストリカル・データ，ジャーナル情報，ユニバーサル・ステーションの画面情報，他ノードのパーソナリティを示すプログラ

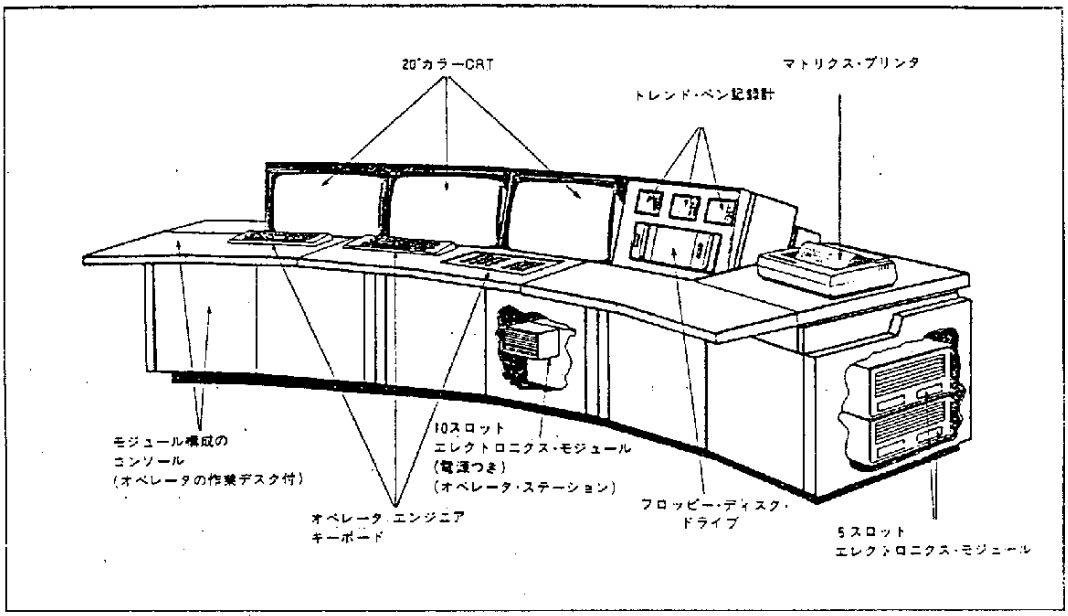


図２－２３　コンソール組込図

ムとデータ（このプログラムとデータは，システムのスタート・アップ時または各ノードのリスタート時期に，ＨＭより指定のノードにダウン・ライン・ローディングされる。またオン・ライン診断，オフ・ライン診断結果は，各ノードから，アップ・ライン・ローディングされる。）を蓄積する。これらのデータとプログラムは他のノードからアクセスが可能である。

周辺機器はウィンチェスタ・タイプのディスクが接続される。

ＰＷＡはＷＤＤ　Ｉ／Ｆが付加される。

(d)　アプリケーション・モジュール－ＤＣ（ＡＭ－ＤＣ）

レベル２におけるコントロールを実行するノードである。

プロセス・コントロール向けに開発された専用の言語（ＣＬ）で，ユーザが予めプログラミングを実施し，このプログラムに従ってＡＭ－ＤＣは処理をする。

ＡＭ－ＤＣのハードウェアは特別な周辺機器及びＰＷＡは存在せず，基本構成のハードウェアのみで構成される。

(e) コンピューティング・モジュール－３０－６０（ＣＭ－３０，－６０）

レベル3におけるコントロールを実行するノードである。

汎用言語（FORTRAN またはPASCAL）で記述されたプログラムを実行する。プラントの総合管理を可能とするノードである。

ＣＭ－３０のハードウェアは基本構成のままであるが，ＣＭ－６０はノードとして従来型のミニ・コンピュータを使う。

(f) コンピュータ・ゲートウェイ（ＣＧ）

大型の計算機との接続をする場合に，TDCS3000 システムと通信ラインで接続するノードである。通信のプロコル上で，物理層，データ・リンク層，ネット・ワーク層，アプリケーション層と階層構造をとり，データ・アクセスの規定をしている。

７２ＫＢＰＳのＢＳＣをデータ・リンクのプロトコルとしている。

ハードウェアとしては基本構成以外に，ＣＯＭＭ　Ｉ／Ｆが追加される。

(g) ローカル・コントロール・ネットワーク（ＬＣＮ）

ＬＣＮは，各ノード間通信の経路であり，二重化された同軸ケーブル上に最大６４個のノード接続を可能としている。ビット転送速度は５ＭＰＳであり，基本距離３００ｍで，トークン・パスのプロトコルを採用している。

(h) プロセス・コネクテド・ボックス（ＰＣＢ）

ＨＧに接続されるデータ・ハイウェイ上に乗る機器をＰＣＢと呼ぶ。ＰＣＢは，従来のTDCS2000 システムで使われている機器であり，制御レベルとしてはレベル１に位置するコントロールを実施する。

以上ハードの構成について概略を述べたが開発期間の短縮化を実現するために，ハードウェア・リソースの共通化について言及する。

本システムのマイコンは68000を用いている。これはメインのCPUのみでなく全てのI／Fには同一の68000を使用しており，各I／F上のファーム・ウェアは共通した手法（METHODOLOGY）に従い，ルーチンの共有化を図っている。

勿論，ハードウェアの回路においても，各I／Fの共通化は徹底した。

こうしたCOMMONARITYはシステムの規模が大になるに従い，製品品質，保守性，開発工数の低減を守る上で重要なポイントとなる。

(3) ソフトウェア構造

本システムのソフトウェア構造およびその特徴について記述する。

それぞれの物理ノードは，図2－24に示すように2つの層から形成される。

1つはソフトウェア・エンバイロメント（SE）と呼ばれる層であり，他の層はSE上に乗るノード固有の機能を実現するための，ベーシック・アプリケーション・ソフトウェア（BAS）である。

またCM－30，－60とAM－DCにはSEの上にユーザ側で記述されるアプリケーション層が乗る。

SEはそれぞれの物理的なノード上に1つ存在する。その機能を次に示す。

① オペレーティング・システム

② データ・アクセス，すなわちシステムにおけるデータ・アクセスのための共通のインターフェイスを提供する。

③ ノード・アドミニストレータ，つまり物理ノードと論理ノードの管理，コントロールをする。

④ エラー・プロセッシング，これはシステムでの統一したエラー

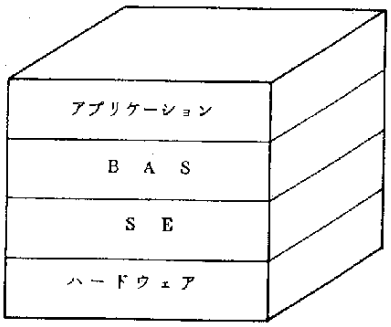
処理を実行する。

⑤　ノード通信コントロール，ＬＣＮを経由するノード間通信処理を実行する。

⑥　テスト・アンド・ダイアゴノンス，ノード上の故障の検出／その結果の報告を実行する。

⑦　ネットワーク・ファイル管理，ネットワーク上のファイル・アクセス機能を提供する。

このＳＥにより各ノード上のＢＡＳはアクセスすべきデータやファイル，また通信相手の物理的存在場所を意識することなく，つまり論理的な名称を使うことで，各種アクセスが可能となっている。これにより，システムの拡張性，冗長性および論理ノードの再配置が可能となる。



BAS : BASIC APPLICASION SOFTWARE
SE : SOFTWARE ENVIRONMENT

図２－２４　ノードのソフトウェア構造

2.2.3　開発工程

前項までに述べたTDCS３０００ システムで開発される工程について記す。

製品およびシステムの開発には，市場のサーベイから始まるが，ここでは，この作業はスキップして，設計レベルから量産に移るまでの過程について記述する。

(1) 概念設計

開発設計の端緒となるものは，要求仕様である。既に2.2.1項で述べたように，TDC3000 システムの場合には，各レベルのＲＤ（REQUIRED DOCUMENT）が，結果として存在するが，始めは全体システム・レベルのＲＤで規定され，これに従って開発部門における概念設計が始まる。本例では，各ノードのタスクの分割，ＬＣＮの転送レートと概略プロトコルおよび通信の媒体，マイコンの選定，ノード上のＯＳの概略機能，データ・ベースの概略設計，マン・マシン機能の概略設計，メカニカルまたはイメージ的な意匠デザイン，目標とする信頼度（ＭＴＢＦ，ＭＴＴＲ，故障時の機能縮退など）の設定，環境条件設定などが，このステージで決められ，これはシステム概略機能仕様書（SUMMARY FUNCTIONAL SPEC）で定義される。

また，サブ・システム・レベルにおいても同様な工程を経過するが，これは，システム・レベルの概念設計が終了してから始まることになる。例えばコンピュータ・ゲートウェイ（ＣＧ）の場合では，サブ・システムＣＧの概念設計とは，他の計算機とのデータ転送レート，そのプロトコル，他の計算機がアクセスするデータ・ポイントの数，フォーマット，プロテクション，エラーリカバリなどが決められる。同様にＣＧのＳＦＳとして仕様書にまとめられる。

この工程では，仕様の要求側と，それを実現する側の間で，差異が生ずる場合があり，この間をいかに埋めてゆくかの調整が重要な課題である。従って，ＲＤとＳＦＳは見直し改訂され，完成度の高いものとされるのが常である。

(2) 設計試作

概念設計が終ると，ハードウェアは設計試作を開始する。TDCS 3000 の場合は実験室レベルで数台のエンジニアリング・モデルを

作成し，システムまたはサブシステムの実現性について追求をする。

ソフトウェア作業におけるこの工程では，システムまたはサブシステムの機能を分解し，その性能などについて設計を行い，システムまたはサブシステム機能仕様書の作成を実施する。更に，このレベルの機能に対応した製品についての内部構造，性能の実現をするための設計を行い，構造設計書としてまとめる。つまり，この工程では，詳細設計の前工程として，ソフトウェアの機能と構成の概略設計を実行するものである。

(3) 詳細設計

サブ・システムまたはサブ・サブ・システムの詳細な設計を行う工程である。

サブ・システム・レベルで実現される機能・性能を明らかにし，他のサブ・システムとの関係，そのインターフェイスについて定義し，その結果を詳細機能仕様書にまとめる。従って，サブ・システム間の機能の関連や分割が明らかになり，機能上の重複，脱落や階層的な整然さについて，実現上の妥当性が明白になる。

詳細機能仕様に従って，ソフトウェアで実現する，内部構造（プログラムの構造，データのフォーマット，データ・ベース，他サブシステムとのインターフェイス手順など）を明確にし，それらの相互関係の設計を行い，詳細設計書に記述する。

(4) プログラム作成

詳細設計書から，モジュール単位に分割しプログラムの作成（コーディング）を実施する。TDCS3000 の場合には，モジュールの大きさは，リストで2頁弱を目標としコーディングを実施した。言語はパスカルである。

机上デバッグ，コンパイル，ロード・モジュールの作成，リンキングを実行する。

この工程でのドキュメント作業として，モジュール設計書の作成がある。これは，モジュール・レベルでの設計の結果として，各モジュールが果すべき機能，性能，内部構造について記述する。

(5) テスト

TDCS3000 システムのソフトウェアテストには，大きく分けて2種類になる。1つは，ソフトウェアの量的に従うテストと，性能を検証または評価するテストである。

量的なテストとは，サブ・システムまたはモジュール・レベルのソフトについて機能的なテストを実施し，その後それらを結合し，システム・レベルのテストとし，また最後には種々の組合せ，総合テストを実施してゆき，仕様通りに実現されているかをチェックするものである。

他方，検証または評価テストとは，設計時における仕様をどの程度満足しているか，つまり性能上，機能上，実現されたソフトが，目標に対する差異を明らかにするテストである。例えば，応答速度とか，異常時の修復の実現度などが，満足のゆくものかをテストする。CGで例をあげれば，72KBPSのBSCで他の計算機と通信をする場合，この仕様をどれだけ，つまり何KBPSまでの通信が可能であるか，という観点でテストする方法を言う。

量的なテストでは，システム結合テスト計画書を作成し，サブシステムを結合しつつ，それらの結合状況が果すべき，機能，性能を満足していることを確認してゆく。

TDCS3000 システムは，分散型であり，またサポートされる機器の量と種類の多さにより，その組合せが異常に多大のため，システムの範囲と，テストの深度を充分に実施することが要求され，この工程での工数は膨大となる。

(6) ドキュメンテーション

各工程で作成した，ドキュメントの整理をする。これはソフトウェア作成者とそのメンテナンス・エンジニアが必ずしも同一でない場合がある。従って，ソフトウェア・デザインレビューの目的としての存在だけでなく，保守用としても，それぞれの仕様書は必要である。

また，ソース・プログラムやその他プログラムは，管理部署で保管され，必要に応じて出荷される。

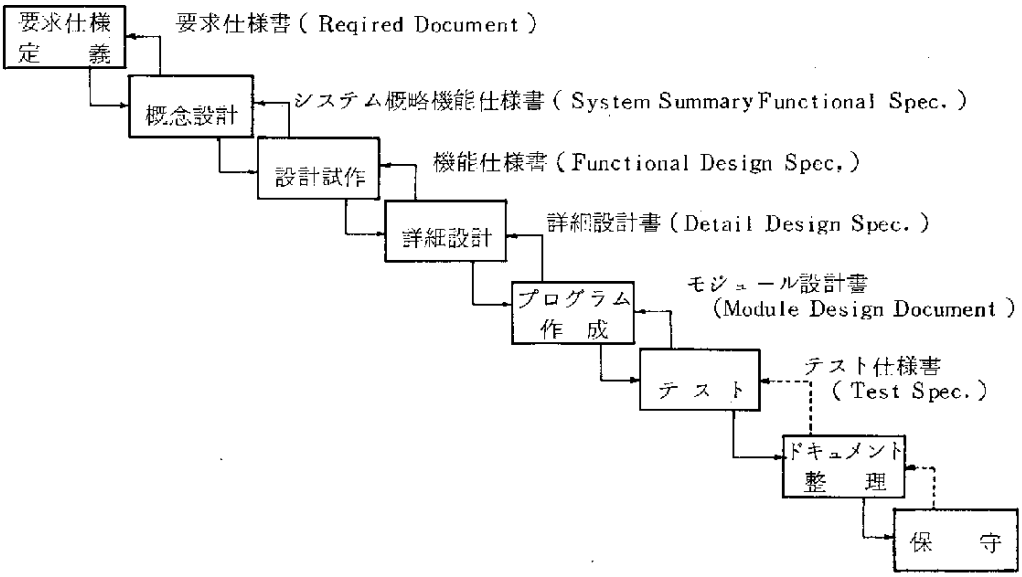


図２－２５　開発環境とソフトウェア・ドキュメント

2.2.4　開発環境・ツール

TDCS３０００ の場合には，プログラム作成用，そのテスト，ドキュメント作成用としての開発環境・ツールに分けることができる。

しかし，提供設備としては，ソフトウェア・ディベロップメント・センタ（ＳＤＣ）が供与する，ミニコンピュータに複数の端末が接続されるというハードウェア構成のものが原形となり，上記３つの環境を満足している。

以下ＳＤＣについて説明をする。

(1)　ハードウェア構成

図２－２６にソフトウェア開発設備のハードウェア構成を示す。複数のミニコンピュータ（負荷分散とバック・アップを兼ねて）に，複数の端末が接続される。ホスト側には大容量のディスクと，高速のライン・プリンタが接続される。

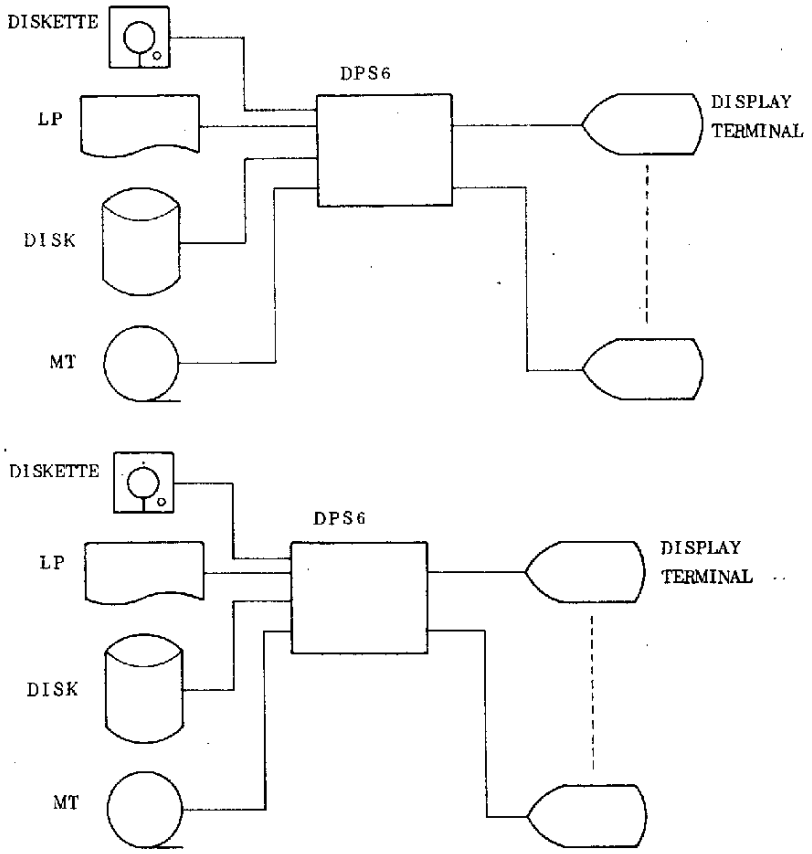


図２－２６

(2)　ソフトウェア環境

ＳＤＣ上で走行するオペレーティング・システムは，タイム・シェアリングのサービスを行う。また，このＯＳの下でサポートされ

る，プログラミング言語は，ＥＣＬ（EXECUTION COMMAND LANGUAGE），PASCAL（ＩＳＯスタンダード），FORTRAN，ミニコンのアセンブリ言語，マクロ・プリプロセッサである。

このようなハードウェア，ソフトウェアのリソースを利用して，M68000 プロセッサの言語処理，リンクが実行でき，ターゲット・システムへのダウン・ローディング，ターゲット・システムからのアップ・ローディングが可能となっている。また，EXORMACS との接続も可能で，ホスト・システム間との通信ができ，ホストからEXORMACSへのローディングもサポートされている。

ソフトウェア，ハードウェアの全てのドキュメントや，プロジェクトを遂行してゆく上での必要なドキュメントは，全てこのＳＤＣで管理され，作成は，各ユーザが，本ファシリティを利用し（英文のワード・プロセッサがサポートされている）実行している。

次に，テスト環境は，ＳＤＣの利用には，限界があるので，ターゲット・システム上にターゲット・ソフトウェアをローディングして，テストの実行となる。テスト用のユーティリティ・ソフトウェアは，各ノードで共通に使用されるツールと，ノード固有に作成されるものがある。オンラインまたはオフラインのデバッギング・エイドなどは前者であるが，他ノードのシミュレータとか，擬似データ・ポイント作成ツールなどは固有ノードで使用されるツールとなる。

また，検証または評価用のツールとしてはパフォーマンス測定，異常メッセージ発生などのツールを作成して，その目的に合わせている。

2.2.5　プロジェクト管理

本項では，開発工程の管理，ドキュメントの管理，開発設備の管理，技術的問題解決法などについて記述する。

(1) 開発工程の管理

開発スケジュールが遅滞なく進捗しているかを把握し，遅延のある場合には，その遅延に対する影響度に応じて，然るべき措置をとることが必要である。

ソフトウェアの工程の進捗度を測るのに，決められたドキュメント（機能仕様書，設計仕様書，構造仕様書，テスト仕様書，テスト結果の報告書）などの完成度を見ることにより分る。また，マイル・ストーンにおけるデザイン・レビューにより，工程上の進捗が把握できる。

TDCS3000 の場合は，作成ソフトウェアが，大量で，また設計者も多い関係で，作成されるソフトウェアの完成時期の相互の関連づけが重要である。例えば，ネットワーク・オペレーティング・システムの機能追加が順次行われ，そのレリースが計画されているとき，他方そのレビジョンのレリース時期に合わせて，各種のベーシック・アプリケーション・ソフトウェアをジェネレーションしてゆくスケジュールが立てられる。このように相互にレリース時期を関連させる図（ディペンデンシーチャート）を作成して，工程管理している。

(2) ドキュメントの管理

作成される大量のドキュメントを体系的に管理してゆくことが必要である。

TDCS3000 では，SDCのコンピュータを利用してそのドキュメント管理をしている。

プログラムのレリースと同じように，ドキュメントがレリースされると，記述された内容は全て，SDCのファイルに入り，他のユーザから端末を通して見ることが可能となっている。

(3) 開発設備の管理

開発プロジェクト・メンバの共通した設備の管理は，組織化して，その作業専任として運営してゆくべきである。TDCS3000 のSD

Cの運営管理は，専任の担当と，ソフトウェア開発メンバの代表が実施している。機器の稼動，設備計画，使用状況，新しく登録されたドキュメント，プログラムの連絡などを業務としている。

(4) 技術的問題の解決法

ここでは，一般的な主題についての解決法を示すのではなく，TDCS3000において用いている方法について述べる。

各マイル・ストーンにおいて，デザインレビューのミーティングを開いていることは既に述べたが，この中から発生する技術上の問題点または日常の開発作業の中から出て来る問題点については，デザイン・カウンセルという機関を設けて，そこで解決を図っている。デザイン・カウンセルの構成メンバは，サブシステム担当責任者，システム統括責任者，問題提起者などから成り，必ずしも固定された人員でなく，問題に応じて参加構成の変動がある。

このカウンセルで決定された事項は，影響が他部門へ及ぼす場合には，その調整を技術部門の人間が，他部門へ働きかけることになる。

2.2.6 課 題

ソフトウェア開発の難しさは，ハードウェアのそれに比べて，差が大きいと言われている。自動化技術・ツールの違いに起因しているとの見方が強い。

TDCS3000 の開発過程を見ると，然りと言える所がある。ハードウェアの論理設計，回路設計，プリント基板のアートワーク設計，などの自動化技術，また量産における自動化機械などの進歩に比較して，ソフトウェアの自動化技術は，遅れている。

これは，物理的に見える製品の製作に従来慣れて来ており，それに対応する人口も多い。従って，体制としても主眼はその方面に傾き易かったと言えるだろう。ハードウェア仕様を決めてから，ソフトウェア仕様

決定に移るという，前時代的開発工程は，見られなくなったが，問題を吸収するための仕様変更は，比重としてソフトウェアにその負担を掛け易い。これは，変更のし易さ，ハードウェア・コストへのインパクト，量産体制への影響などからソフトウェア変更となり勝ちである。

しかし，本当にソフトウェア変更が容易だからと言って，コスト的に割に合うかは疑問の点が多い。一時的な変更，修正の手続上の容易性が大なるが故に，その方法を選んだ結果，修正結果の他への波及が，予測に反して大きく，他のモジュールの変更とか，再テストの工数が膨大になったという例は，少なからずある。

ソフトウェア開発については，自動化技術のみならず，ソフトウェアの持っている性格とそれにまつわる属性の認識を正しく持たないと，開発管理を上手く遂行できないものと思う。

参考文献

(1) TDCS 3000 システム概説書

(2) ソフトウェア生産技術の現状と将来　通信学会誌VOL66 NO.4

(3) 「新分散型制御システムのアーキラクチャ」松本東郷
26回自動制御連合講演会

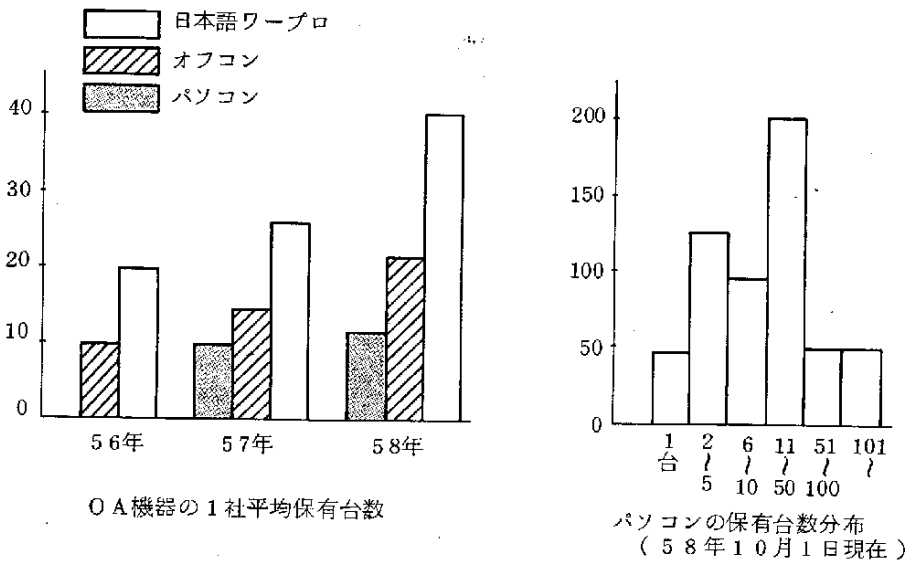
2.3　ＯＡシステム

2.3.1　開発の背景

文書作成など雑用に近いいわゆる非定形業務のＥＤＰ化は定形業務のそれと較べ遅々として進展を見なかったが，マイクロコンピュータ（マイコン）の出現から様相は一変した。

マイコンを内蔵した汎用コンピュータ（パソコン）が，その名のとおり個人で所有できるような価格で出現し，また，フロッピ・ディスク（ＦＤ）が開発され，手軽にデータの蓄積が行えるようになったことなどから，これらを利用した非定形業務処理システム，いわゆるＯＡシステムが実用に供されるようになった。

ＯＡの普及を反映して，パソコンの設置台数も急激に増加しており，日本経済新聞が一部上場の大手製造業６１２社を対象に行った調査ではパソコンの総使用台数は昭和５８年には２万台を越え，一社当りの設置台数も５７年の２５.５台から，３９.２５台に増加している。（図2−27）



ＯＡ機器の１社平均保有台数

パソコンの保有台数分布
（５８年１０月１日現在）

（日本経済新聞　コンピュータ・ユーザー調査58.12.8）

図２−２７　ＯＡ機器の設置台数

ＯＡシステムを導入する場合，専用機の利用とソフト・パッケージ（流通ソフトウェア）を利用するものと２通りがある。

専用機は，ハードウェア，ソフトウェアとも特定の対象業務専用に開発されたものであり一般に高性能であるが高価である。

一方ソフト・パッケージは特定対象業務を汎用パソコンで稼動するよう開発されたソフトウェアであり，対象業務も非常に多く（表２－４）安価であるため，パッケージを何本か購入することにより，１台のパソコンで何種類もの業務処理を行うことができる。

表２－４　パソコンソフトの対象業務

種　類	対　象　業　務
簡易言語	分類，集計，管理資料等の作成
ワープロ	文書作成
グラフ作成	直線，円，棒……各種グラフ作成　図形処理等
データ・ベース検索	データの蓄積及び検索，レポート作成
経営・財務	会計処理，財務諸表の作成／分析
給与計算	給与，賞与，年調計算
販売在庫	売上／仕入／在庫管理等
顧客管理	顧客／会員管理，宛名印刷等
特定業種向け	社会保険労務士，電気工事，土木工事，酒販店，檀家管理，図書管理，特許・年金管理，米穀店，ホテル，薬局，レセプト処理，旅行業……
教育	成績処理，語学，地理，歴史
その他	スケジュール管理，家計簿，住所録，ゴルフスコア，技術・統計計算

ＯＡが個人商店等小規模な企業に迄普及する上での，パッケージの果たす役割は大きい。

流通ソフトの出現は，ソフトウェアハウスにとっても大きな影響を与えた。

今までソフトウェアハウスは主としてオーダメイドのソフトウェア開発に従事し，下請的要素を多分に持っていたが，パッケージの出現により，ソフトウェアハウスはマーケティングを行い，商品を企画し，オリジナル商品を大量販売またはＯＥＭ出荷するといった，下請からメーカへの脱皮も可能となった。

このような意図のもとに多くのソフトウェアハウスがソフトパッケージの開発に着手している。

ソフトパッケージはオーダメイドのソフトウェア開発といろいろな面で異なっている。

本節ではソフトパッケージのうちワープロソフトの開発事例を紹介する。

2.3.2　ソフトパッケージの特徴

ソフトパッケージは流通を目的としたレディメイドのソフトウェアであり，この点，具体的に利用の目的，運用環境を確定してから開発するレディメイドのシステムと異なっている。

ソフトパッケージの特徴として次があげられる。

①　大量販売が前提である。

②　厳しく評価され，その評価が直接販売量に反映する。

③　計画から出荷までの期間を最短にすることが要求される。

④　開発の各ステップが明確に区分できない。

⑤　利用者が広範囲に分布している。

(1)　大量販売を前提にしている

大量販売が前提であることは，流通ソフトウェアの縮命である。

汎用パソコン価格は１００～２００万円程度であり，その利用者に負担を感じさせないパッケージ価格は数万円程度である。

したがって１００本／月の販売量が最底必要であろう。

これを達成するためには適応パソコンはベストセラー機であることが求められる。

当然のことながら，適応パソコンの設置台数が多い程，ソフトパッケージの購入の動機は大きくなる筈である。

しかし，もっとも重要なことは，ベストセラー機にはワープロ以外のソフトパッケージも多数開発され，相乗効果が期待できることである。

すなわち，パソコンとソフトパッケージでおのずから豊富な機能を持つシステムが構成され，それが需要を更に喚起することである（ソフトパッケージが揃っていることがパソコン機種決定の理由にもなっている）。

適応機種は５ケ月後から出荷が始まるPC－９８００と決めたが，これはPC－８８００の実績から，ベストセラーになると推測したものである。

またPC－９８００のCPU8086はCP/M，MS－DOSなど汎用ＯＳが開発されていることから言っても，１６ビットの標準であり，他のパソコンに採用される可能性も高く，移植の際有利である。

結果論であるがPC－９８００用ソフトパッケージ（簡易言語，ワープロ，業務用）のベストセラーの５位迄はPC－９８００用で独占しており，５８年のソフト新製品供給量でも，総本数２５３本中１０８本を占めている[4]。

ただし，ワープロソフトの数も多く，競合が激しいのも事実である。ソフトパッケージはゲームと同じで勝つか，負けるかどちらかである。

このような激戦区で勝つ商品を開発しなければ，パッケージ・メーカとして存続することは難しい。

(2) 厳しく評価され，その評価が直接販売量に反映する

オーダーメイドと違って，現物が存在し，あらゆるテストによる評

価が可能だということである。

また同種のパッケージ間での比較も容易に行えるので，利用者のニーズを的確に把握した上での開発が必要である。

一方利用者の側から見れば，各々のパッケージには一長一短があり選択が難かしく販売店（デイラー）の助言に影響されることも事実である。

このため販売店へのＰＲは特に重要で，パッケージの特長，保守の対応，バージョン・アップの予定などの情報を提供し，商品を正しく理解してもらう努力を怠ることはできない。

(3) 計画から出荷までの期間を最短にすることが要求される

「先発の利」は非常に大きく，パソコン発売開始から，パッケージ発売までの期間の長短が鍵となる（表２－５参照，ワープロソフトの場合，早いものは１ケ月後から発売されている）。

表２－５　PC－９８００発売とワープロソフト発売時期

PC－９８００発売後（５７.１０）	発売を開始したワープロソフトの数
１ケ月	２
５〃	２
６〃	２
８〃	２

したがって，移植や流用が容易なソフトウェア設計，開発ツールの利用による工期の短縮化が重要である。

今回は新開発であり，流用や移植の対象になるソフトウェアがなかったため，適応パソコンの販売開始からパッケージ発売まで６ケ月を要した。

しかしこの６ケ月は，高品質を実現するための商品化テスト２ケ月

を含んでおり，かなり厳しいスケジュールである。

対象機種が未出荷で開発ツールを準備できないこと，新規開発であるという条件のもとで開発期間を短縮する方策として，少数精鋭のプロジェクト構成が有効である。

優秀なＳＥにより工期の短縮を図ることは当然で説明するまでもないが，プログラム作成者を少数に絞ることはオーバーヘッドを少なくし，仕様変更等に柔軟に対応でき，この種のシステム開発には有効である。

(4) 開発の各ステップが明確に区分できない

オーダーメイド・システムの開発では各開発工程が明確に区分され，決められた手続きで開発を進めているが，ソフトパッケージの開発では，各ステップの終了が明確でないばかりか，前のステップに逆戻りすることもある。

この理由は，開発期間の短縮化が極度に要求される中で，外部からの情報や要請にもとづく仕様変更が多いこと，このような会話形システムの評価規準が確立しておらず，試行錯誤を繰り返えしながら開発が進められることである。

したがって，作成された商品の質は個人の資質（技術力のほか創造力）に負う所が非常に大きい。少数精鋭のプロジェクトを構成したもう１つの理由である。

2.3.3 開発の方針

ワープロソフトを開発の対象に選んだ理由は，ＯＡの三種の神器はワープロ，パソコン，ファクシミリと言われるように，ＯＡの中でワープロは重要な位置にあること，数年前から専用機によるワープロが発売されているが，汎用パソコンでこれを安い価格で実現すれば大きな需要が期待できること，また利用者のニーズも他の業務処理とくらべて把握し

易かったことなどである。

「専用機を超えるワープロソフト」を開発の目標とした。

ワープロソフトの仕様概要を表２－６に示す。

開発の基本方針

① 汎用ＯＳを採用し，システムに拡張性を持たせる。

② 専用機に匹敵する高速処理

③ 信頼性の高いシステムの実現

とした。

表２－６　仕様の概要

システムの目的	日本語のワードプロセッサ（ソフトパッケージ）
開発期間	昭和５７年～５８年
適応機種	ＰＣ－９８０１

仕様の概要

ディスプレイ
- 表示文字数：３８字×２０行
- 文字ドット構成：１６×１６
- 書体：ゴシック
- 表示文字色：８色
- 仮想画面最大サイズ：３８字×６０００行

入力
- 入力方式：キーボード
- 変換方式：かな漢字変換
- 最長変換単位：熟語
- 使用総文字数：約３５００字
- 登録単語数：約６５,０００語
- ユーザ単語登録数：１０,０００語（最大）
- 同音異義語選択方式：順次

ディスク
- 媒体：フロッピーディスク
- サイズ：８インチ
- 密度：両面倍密度
- ドライブ：２

ファイル
- Ａ４換算ページ数：１００ページ
- 登録可能文書数：６４

プリンタ
- 文字構成ドット：２４×２４，１６×１６
- 印字書体：明朝体，ゴシック体

プリント機能：横書，縦書
- 印字方向
- 文字間隔指定
- 行間隔指定
- 任意のページ印刷
- 自動ページ番号付
- 禁則処理印字
- 画面と異なる書式印字

編集機能
- ブロック移動
- 複写
- センタリング
- 右寄せ
- 切り貼り

補助機能
- 作画機能（矢印キーにする）
- 演算機能（四則演算）

マニュアル
- 判型：Ａ４
- ページ数：約６０ページ

(1) 汎用ＯＳの採用

他のパッケージと稼動環境を合わせることで，データに互換性を持たせることは今後増増重要になってくる。

例えば，文書の作成はワープロ，計算処理は簡易言語，表示は作表パッケージと，その都度，データを手動入力するのは不便である。

同一パソコンでそれぞれの業務パッケージが実行でき，かつ各々のパッケージの出力が他のパッケージの入力になるとすれば，データの入力から結果の出力，蓄積までが入手を介さず効率的に行える（これが統合ソフトウェアの考え方であり，ＯＡシステムはこの方向に発展している）。

このような考え方からＯＳはCP／M－８６，MS－DOS とし，各々その標準ファイルを採用し，ワープロで扱うファイルを他のシステムが容易にアクセスすることを可能にすると共に，ファイル・フォーマットも公開し，システムの拡張性を図った。

(2) 処理の高速化

専用システムに匹敵する処理速度を実現するため記述言語をアセンブラとした。

開発の生産性，システムの移植性を考えると高位言語で記述することが有利であるが，ワープロの場合高速処理は実用性を高める上で大きな効果がある。

すなわち，パッケージの利用者の技術格差が大きく，考えながら操作する初心者（非専門家）には処理速度はあまり気にならないが，ブラインド・タッチができる熟練者にとっては高速処理は必須条件である。

表２－７に，処理速度のテスト結果を示すが５～１０倍の高速化を実現できる。

アセンブラを採用したもう１つの理由は，高位言語のオブジェクト

表2－7　最新ワープロソフト（PC9800）の処理スピード[3]

調査項目	テスト方法	日本語＊ワードプロセッサ	マイレター98	漢神	JWORD＊
スクロール速度	60行をスクロールする時間	3.7秒	115秒	12秒	3.8秒
カーソル移動速度	10秒間カーソルキーを押し続けどれだけ移動したか	221個	19.5個	49個	228個
単純入力速度	10秒間“A”キーを押し続け，どれだけ表示されたか	229個	195個	51個	224個

＊：全プログラムアセンブラで記述

のメモリ効率がある。

64KB のプログラム領域にオーバーレイなしに全プログラムを格納することを前提とすれば高位言語による記述では難しい。

(3)　信頼性の高いシステム

ソフトパッケージは特に高信頼性が要求される。大規模ソフトウェアの場合，誤りを修正するための相対コストは，設計中に発見したものを1とすれば，テスト開始前6.5，テスト中15，リリース後67というデータがある[1]。

ソフトパッケージの場合は，利用者が多くしかも地理的に広範囲に分布しているので，厖大な費用がかかり，事実上リリース後の修正は不可能である。

経験からすれば通常のテストではリリース後発見される誤りは0.4件／Kステップ程度であり，信頼性の高いと報告されているニューヨーク・タイムズのシステムでも0.3件／Kステップ[2]である。ホワイト／ブラック・ボックステストの徹底により，それ以下のエラー発生率（それでも計算上は数件の誤りがあることになるが，致命的なエラーさえ避ければ，バージョン・アップの際の修正で対処できる）を目

標とした。

2.3.4 システムの概要

(1) スケジュール

開発の期間（表２－８）は１２ヶ月であるが実機入手は開始後６ヶ月のため，実機上での作業は正味６ヶ月である。

他のマイコンシステムのようにハードウェアとソフトウェアの開発が同時に進行することはないが，実機の入手前から開発が行われることが多い。

このようにカタログ仕様をベースに設計を行い，実機入手後の限られた期間に商品化テストを含む実機上での作業が行われる。

しかも高信頼性が要求されるため商品化テスト（通常のシステムの総合テスト，運用テストに対応する）の比率は高い。

特に会話形システムではテストを鍵盤操作に頼るためこの傾向が顕著である。

表２－８　開発スケジュール

工程＼期間(月)	1	2	3	4	5	6	7	8	9	10	11	12	特記事項
商品企画	▲												市場調査，類似システム調査，汎用パソコン調査 } 適用パソコン，インプリメント仕様，出荷期日の決定
基本設計		▲			▲								かな漢字変換方式の決定 文書ファイル，作業ファイル，辞書ファイル 操作法の決定 } データ構造の決定
プログラム設計					▲		▲						機能設計 モジュール切り出し
プログラム作成							▲		▲				アセンブラによるコーディング
テスト									▲		▲		単体 結合 } テスト
商品化テスト											▲	▲	処理速度，操作性の評価 （模擬利用者による評価テストを含む）
利用者マニアル				▲ 第１版		▲ 2		▲ 3		▲ 最終版			原案から最終版迄４回ブラシュ・アップ
対外活動					▲ PC9800出荷始まる 営業活動開始							▲ 出荷	出荷以後 20～30 件／日の問合せに対応

(2) 要員構成

要員構成を表２－９に示すが，流通ソフトウェアの開発では，営業部門の役割が重要で，開発部門と表裏一体となって開発を進めなければならない。

流通ソフトウェアと呼ばれるようにディーラ経由の販売量が直販に比べ圧倒的に多く，また利用者に最も近い立場にあるディーラからの情報はシステムの仕様を決める上で参考になる。このような情報の収集，対外の折渉，見込み客の開拓など発売までに営業部門でしなければならない業務は多い。

構成要員で特徴的なのは模擬利用者（専任ではない）である。

模擬利用者は社内の事務部門から随時レビューやテストに参加してもらい，初心者の立場でシステムの評価や助言を行うことが目的である。

特に利用者マニュアルのレビュー等には有効で，用語や操作法など作成者の意図と利用者の理解のギャップの解消には非常に効果がある。

表２－９　プロジェクトの要員構成

部　　門	人　員	分　担　業　務
営　　業	2	情報の集収 対外折渉 ＰＲ
開　　発 システム開発	2	設計，作成 利用者マニュアル
補助（女性）	1	ドキュメント整理 プログラム管理／修正 テスト 連絡
模疑利用者 （社内）		随　時 利用者の立場で レビューに参加， システムのテスト／評価

(3) 商品計画

汎用パソコン，ＯＳ，競合システムの調査から始まって，商品としての輪郭を明確にする工程である。

期間は２ヶ月であるが，業務パッケージの場合は数ヶ月は必要であろう。

例えば，業務処理の中で一番パッケージ化し易いといわれている給与計算，財務会計でも，利用者各々の特殊処理があり，現状分析だけで長い期間が必要である。

ワープロの場合は既に専用機が何種類も発売されており，操作面，機能面での評価や問題点が容易に把握できた分，短縮されている。

専用機で実現している総ての機能や問題点を改善してインプリメントすることは不可能であり，機能としての重要性，機能間のバランス，ソフトウェアでの実現の難易度，開発期間，他のシステムとの関連（移植性など）を考慮して，各々の機能について，

① インプリメントするもの

② バージョン・アップの際に考慮するもの

③ インプリメントしないもの

を前もって決める必要がある。これはテーブルやファイルのフォーマットの設計上意味を持ち，システムの保守性を高める。

(4) 基本設計／プログラム設計

ソフトパッケージの設計に際しては

① データの互換性

② システムの拡張性，融通性

③ 装置独立性

が重要である。

(a) データの互換性

他のパッケージとのデータの互換性は統合ソフトウェアの実現と

いう意味からも重要であり，データをＯＳの標準ファイル形式にすることで互換を図った。

但し，ワープロとしての処理速度はある程度犠牲になるが他のパッケージとファイルを媒体にしてデータ交換ができれば全体としての処理速度は飛躍的に向上する筈である。

(b) システムの拡張性・融通性

バージョンアップ時の拡張機能などは商品企画の工程で確定しており，ファイルやテーブルの項目として折込み済であるが，他の意図しない仕様変更や機能の拡張に対しても対処できるよう各種のファイル，大域データ（共通テーブル等）へのアクセスは総てサブルーチン経由で行う。

(c) 装置独立性

いかなる入出力装置にも対応できるよう，論理データから物理データへの変換処理は完全に独立したモジュールとする。

(5) テスト

テストは単体テスト，結合テスト，商品化テストの三段階がある。

商品化テストは，通常のシステムでの総合テスト或は運用テストに相当するものである。

(a) 単体テスト，結合テスト

1) テスト方法

トップダウンテスト[(2)]が有効であった。その理由は

① 会話形システムであり，１つ１つのコマンドに対応する処理の独立性が高い。

② コマンドに対応して実行されるモジュールが限られている。

③ 入力／出力が文字列でＣＲＴ上に表示されるため誤りの症状が把握し易い。

④ ソース・プログラムに誤りが少い（プログラマの質に依存

する。）

上記の理由は更にテストを容易にし，結合テストの際スタブ等ダミーモジュールを介さず直接トップダウンテストが行え，工期の短縮に有効である。

2) テスト・データ

高信頼性を確保するためには網羅テストのように全パスをすくなくとも１回は通るようなテスト・データが必要である。

高位言語では処理の構造をそのままソースプログラムに反映して記述できるが，アセンブラの場合には構造化して記述できないので（コメントの使用により，ある程度改善はできるが），パスの消込みにかなりの時間を必要とした。

(b) 商品化テスト

商品化テストでは機能，操作性，処理速度，信頼性のテストが行われる。

ソフトパッケージの性格から信頼性のテストは特に重要である。信頼性のテストでは誤りを発見するというより誤りをなくすことである。すなわち，このテストのテストデータは百発百中的でなく，じゅうたん爆撃的でなければならない。

しかし期間や工数には限界があり，利用者マニュアルをベースとした，同値分割[(1)]テストが現実的であろう（内部仕様をベースにした網羅テストは結合テストまでに完了している）。

異常処理（ハードウェアエラー等）については意図的に異常を起させるのが難しく，プログラムに手を入れて，その部分を実行する程度であった。

パソコン会話形システムで障害になるのはテストに時間のかかることである。

鍵盤から入力するコマンドやテストデータをあらかじめ，ファイ

ルに格納しておき，ファイルのデータをあたかも鍵盤からの入力のように取り扱うことができれば，毎回鍵盤入力しなくても済み効率的である。しかし汎用ＯＳの改造を伴うので，その実現は困難である。

(6) 利用者マニュアル

ソフトパッケージの特長は，導入した日から誰でも操作できることである。しかしこの操作法は利用者マニュアルでしか伝えることができない。

何かをしたい時にはどうすれば良いか？，何ができるか？を教えたり，フロッピ・ディスクが何であるか知らない人にも，イニシャライズをさせたり，ドライブに正しくセットさせたり等々，利用者マニュアルは商品の重要な一部である。

このため利用者マニュアルの作成にはかなりの工数と期間が必要である。

一般には開発部門で作成した原案を他の部門でレビューし，何回かブラッシュアップするのが普通である（図２－２８）

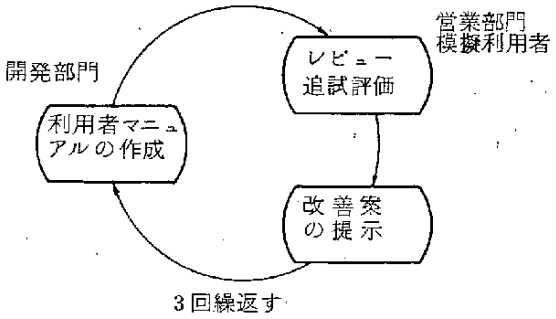


図２－２８　利用者マニュアルのブラッシュアップ

ソフトパッケージの場合，計算機の知識のない人によるテスト（マニュアルに書いてある通りの操作を実際に行って見る）や，レビューへ

の参加の効果は大きい。

利用者マニュアルの書き方として

(a) 難しい用語は避ける

プログラマにとって易しい用語であっても，他の人にとっては馴染みのないものが多い。やむを得ない場合は図や写真を併記する。

(b) 見出しは処理名でする。

「何をする時は」といった見出しとし，各々独立にまとめる。

(c) 入出力の対応の明確化

鍵盤操作とその出力（主としてＣＲＴ画面）の対応関係を１つ１つ図等で明確にする（複数の鍵盤操作をまとめて説明しない）。

(d) 項目の独立性を保つ

説明は項目ごとに独立して行い，他の項目の参照などは避ける。

(7) 保守

保守には利用者への対応とバージョンアップがある。

(a) 利用者への対応

パッケージが完成し出荷が始まると利用者から20－30件／日の問合せ（主として電話，訪問もあるが）は覚悟しなければならない。

内容は（頻度順）

1) 入出力機器関連

主としてプリンタの仕様，例えば現在使用しているパソコンのプリンタで縦書きができるか？（未導入客）と言った問合せ。

2) 環境設定（システムのイニシャライズ）

直接業務に関係のない環境設定は利用者としては理解しにくい（但し，マニュアルで指定した操作を正しく理解しているが確認のための問合せも多い）。

まれには，プログラムと文書用ＦＤを取違えてプログラムＦＤ

をイニシャライズした例などがある。

3) 誤操作，故障

利用者の意図どおりの結果が得られない場合は誤操作か故障（ハードまたはソフト）の何れかである。しかし利用者には判断が難しい場合が多い。

その時の状況により推測して対処しなければならない。

(b) バージョンアップ

バージョンアップは，パッケージソフトを発売してから6ケ月後位に行われる。

理由は最初のバージョンでは不急の機能はサポートせず，バージョンアップの際，その後必要になった機能とともにシステムに組込み，より完成度の高いパッケージとすることである。

パッケージには利用者カードが添付されており，購入後利用者の意見，感想が記入されて返送されるが，バージョンアップの際の追加機能を決める上で参考になる。

2.3.5 課　題

以上ワープロを中心に，パソコンソフトの開発事例を紹介したが，更にパッケージソフトによるOA化を進めるには，より使い易いシステム，統合化による効率的なシステムの実現が望まれる。

また，コンピュータに慣れない一般の利用者にとって操作性の向上も重要である。

操作性の向上には色々な工夫がこらされておりアップル社のパソコンLISA（Local Integrated Software Architecture）は視覚に訴える方法で向上を図っている。即ち，CRT上に機能やオフィス用具を図で表示し，その絵を見ながらパソコンを操作する方式で，初心者でも30分で使えることがキャッチフレーズである。

音声入力は無理としても，動画や音声出力等により親しめるシステムの出現が望まれる。

前述の統合化については，現在のＯＡ用システムでは操作はもとよりデータの互換性も全くないと言える。

現在，企業内の各所でスタンドアロン形式で処理されているものがＬＡＮ（Local Area Network）等で統合される方向に進展しているが，データファイルなどの規格化が待たれる所である。

ワープロの分野でも全く同じことが言え，毎日何千もの標準文書が作成されているが，この文書がワープロをとりかえたことで，再利用できなくなるとすれば一企業でだけでなく国家的損失である。

（ワープロのファイル，フォーマットの規格化に関しては昭和５８年電子協がまとめた「日本語文書交換用ファイル仕様（基本形）ＪＥＩＤＡ－３５」がある。これは機種によって文書ファイル仕様が違うため，互に文書の交換が不可能だったがＪＥＩＤＡ－３５に従った文書ファイルに変換することで異機種間文書交換を図ったものである。趣旨は異なるが規格化に一歩前進したと言える）。

参考文献

(1) ソフトウェア・エンジニアリング序説
(Software engineering a Practitioner's Approach)
Roger S. Pressman
岸田孝一監訳
TBS 出版会

(2) Chief programmer team management of production programming
F. T. Backer
IBM System Journal 1972

(3) PC－9801 最新日本語WPの評価
千葉ちよ子
インホーメーション サイエンス
1983／9
(4) 日経パソコン
58年新製品特集
58. 12. 5 号

第3章　マイクロコンピュータ応用システム開発の将来

第3章　マイクロコンピュータ応用システム開発の将来

3.1　システム開発の課題

マイクロコンピュータ（以後便宜上，マイクロコンピュータという言葉の中にマイクロプロセッサの意味をも含めて使う）が新しいタイプの技術要素として巷間に登場して以来，既に１０年余になるが，マイクロコンピュータ応用上の課題は体系的に解決されたというわけにはいかない。その歳月の間に多くの工夫が加えられ，多くの方法論とノウハウが蓄積され普及してきたのは事実である。しかし一方では，解決を迫られている課題の数は増え，質的にもむしろ困難を増しているのが現状であるといえよう。

それは一つには，マイクロコンピュータは従来の技術に類を見ない新しいタイプの機能素子だからである。従来の技術は伝統的な縦割りの分野の中で把えられ，発展してきた。土木技術，建築技術，機械・精密機械技術，電気・電子・通信技術，無機・有機化学技術，金属技術，・・・，という分野割りはわが国では明治時代以来の伝統を持ち，今日でも重きを置いて考えられているというのがそのよい例である。ところが，マイクロコンピュータはもともとエレクトロニクスの技術として開発されたものであるとはいえ，機能の可能性と応用の現実を見れば，そのような枠を遙かにはみ出したところに位置づけられるものになっている。さまざまの，というより殆んどあらゆる分野の技術製品の中に溶け込み，それらに新しい高度な機能を与え性能を格段に高める。小形，軽量でしかも価格（コスト／パーフォマンス）が優れているために，技術の底辺をかさ上げする機能素子として普及することが容易なのである。応用の対象が多様であれば，当然のこととして，応用のシステム開発の方法論や技法も多様であって，とても一般論的に包括することはできない。

システム開発を困難にしているもう一つの大きな理由は，ハードウェア技術すなわち半導体の加工と回路集積の技術が，極めて急速に変化し進歩してしまうという点である。1971年にIntelから4ビットの4004（ピン数24）が発表された当時，一つのチップに乗せられたトランジスタの数は約2,200

個であり，それはPMOSによるものであった。続く72年の8ビット機8008（ピン数40）では2,300個のトランジスタが集積されたが，これも同じくPMOSであった。これを第1世代とすれば，73年から74年にかけては早くも第2世代機種としてIntelの8ビット機8080，Motorolaの同じくM6800が登場したが，これらは演算速度,集積度がともに2～3倍に向上したNMOS機であった。その後NMOSが主流となり，76年にはZilogの8ビット機Z80（ピン数40）が登場した。これは8,400個のトランジスタを集積したもので，8ビット機としてはベストセラーとなり，この系統のものは今日でも最もよく使われている。75年後半から76年前半にかけては16ビット機種が出現してきたが，78年にIntelの発表した8086はHMOS型であり，トランジスタ数にして29,000個,メモリー容量にして1MBの集積度を持ち，速度もさらに向上してミニコンピュータ並みの機能を有するまでになった。これは8080系の延長線上にある設計思想に基づいており，OSも8ビット機の主要OSであるCP/M系のCP/M-86を用いているために，8ビット時代からのソフトウェアの移行にも便利で，たちまち16ビット機の主流を占めるようになったのである。

短い歳月のうちに，このように激しい進歩を遂げたテクノロジーは過去に例を見ない。その為にさまざまのとまどいが生じた。素子のハードウェア技術が変われば，それに対応した応用のあり方も変わらざるを得ないし，応用システム開発の方策も変わらざるを得ないのであるが，あまりにも急激に変化すると成熟の段階というものが無くなってしまう。早い話が，開発サポート・ツールが整備される前に，次の素子への対応の準備を始めなければならないという具合である。つまり，ある世代の課題が一定レベルの解決を見るまでに次の世代の開発を始めることになり，当然のこととして課題の積残しを生じる。単に積残しを生じるというだけではない。素子の機能と性能が向上すれば，応用システムの規模と複雑さは拡大され，従って課題も質的・量的に増殖されたものとなる。

もちろん，ビット数の増大と世代の交替につれて，テクノロジーや開発サポート態勢にまったく継承性が欠けていたわけではない。例えば，上記の8086の場合のように，上位機種という概念が大いに有用なものとなっている。つまり，従来機種で蓄積されたソフトウェアの継承性と互換性を許容するような新機種が登場すれば，ユーザにとっては大きなメリットがあるわけで，それだけ市場価値は高まり普及が加速される。いわゆる上方（上向き）互換性の思想が重視されるゆえんである。この思想は最近の（８２～８３年以降の）新機種開発に積極的に活用されている。8086の系列でいうと，80186，80286，さらには80386（３２ビット型）という一連の上位機種である。これらはソフトウェアの互換性，周辺ＬＳＩの共通性を持つように8086のアーキテクチェアを基本として開発されており，かつ用途に合わせた素子機能を有するという特徴がある。

以上に見たように，マイクロコンピュータの技術は応用と素子そのものの両面で，従来の技術に例の無いような空間的な拡がりの大きさと，時間的な変化の速さを示しており，それに伴う課題を生み出している。それなりの新しい対応策もいくつか具体化しつつあるが，ここで課題の要点を拾い出して，今後の考察の前提とすることにしよう。

(1) 応用の目的と分野

マイクロコンピュータの応用目的は，一つには種々の産業製品に組込んで，計測・情報処理・制御の機能を持たせる（あるいは高める）ためのものであり，他の一つはそれら製品の生産工程に利用して，自動化，省力化により生産性と品質の向上をはかるものである。さらにもう一つ，パーソナルコンピュータ（パソコン）として小形汎用コンピュータとして計算・情報処理に用いるものも挙げられる。前二者は多岐な産業分野にわたるが，①プロセス（プラント）関係，②機械（電気機械も含む）関係，③オフィス・オートメーション（ＯＡ）関係，④家電・生活機器関係（ホーム・オートメーション，ＨＡも含む）などに大別されよう。

①の中には，発送配電，通信，鉄鋼，化学・食物・薬品，エレクトロニクス素子・デバイス，環境機器などがあり，②には一般機械，工作機械，原動機，自動車・鉄道車両，船舶，航空機，土木・建築機械，精密機械，電気機械，自動化・制御機器等々多様なものが含まれる。細かく見れば，それらの分野ごとに応用目的の違いがあるのはもちろんであるが，一般的な観点から把えれば，①製品の機能・性能の高度化，②製品の低価格化と小形化，③製品の信頼性の向上，④生産工程の自動化・省力化・合理化，⑤生産工程の高度化による製品品質の向上，⑥多品種少量生産への対応，⑦生産工程と製品の標準化などと見ることができる。(注) その他，製品のイメージアップ，ユーザの嗜好や業界のすう勢に合わせるためといったものも現実には存在する。

以上のような一般的に考えられるメリットは，おおむね実際に効果を上げている場合が多いが，必ずしもそうでない場合もある。例えば，機能の高度化をねらったあまり価格は割高になり，製品の操作性や保守性が低下してしまったとか，あるいは生産工程の省力化・合理化には大いに貢献したが，工程の柔軟化や製品の機能向上にはそれほど役立たなかったとかいうように，最初の目的（動機）と導入後の結果（効果）がくい違うことも少なくない。また所期の目的は一応達成したにも拘らず，営業的には失敗に終ったという例も少なくない。

このような見込み違いを無くするためには，まず第一に幾つかの目的を漠然としたイメージの段階から明確な目標の段階まで引上げ，指標化することである。上記の①～⑦の目的項目は往々にして互いに競合関係にあるものであり，そのすべてを同時に満足させることは無理であると考えた方がよい。個々の目的を指標化し，手持ちの技術ツールと開発力を使って，どの目的をどのレベルまで達成できるのか，どの目的に重点を置き，どの

(注)(財)日本情報処理開発協会マイクロコンピュータ応用技術調査委員会報告書「マイクロコンピュータを応用した生産技術の現状と将来動向」，57－R010，第1章（昭和58年3月）

目的は副次的なものと考えるのか，そういったことを整理したうえで全体としての目的指標をはっきりさせるべきである。

第二には，応用対象の技術的ならびに市場的な特徴と特性をできるだけ詳細に分析することである。例えば，対象プラントはその産業で現在どのような位置を占めているのか，現在の技術的成熟度と問題点はどこにあるのか，マイコンを導入したとして何がどう変わり得るのか，また変えるべきなのか，そのプラントの将来性はどうなのか等々，また場合によってはその産業の現状と将来性，将来戦略についての見通しについても検討しておくことが必要である。プラントの自動化・高度化といっても，何が重要でまた何ができるかということはプラントごとにずいぶんと違うはずで，なるべくシステム開発の初期の段階からユーザと共同作業をして，目的を明確化しておくことがキーポイントの一つである。

(2) 要求定義と要求仕様

整理され，指標化された目的項目をさらに具体的な設計仕様としてまとめなければならない。それを通常システムの要求定義あるいは要求仕様の作成と呼んでいる。実は現状のシステム開発において，この段階が大きな障害となっている事例が多い。システムは，ユーザからの発注によって開発される場合と，メーカの自主開発による場合とがある。前者の場合には仕様はユーザが作成したうえで開発先に委託するのが建前である。ところが実際には，“これこれの要求を満足すべきである”という要求仕様が甚だあいまいであったり，矛盾を含んでいたりすることが多いのであって，それでは委託された方では困惑せざるを得ない。あるいは開発工程がある程度進んでから，時によっては開発が終って成果品が出来上ってから，発注者の思わくとだいぶずれていて大きな手戻りが発生するという事態になる。ユーザ（発注者）自体に明確な定義が欠けている場合と，ユーザからメーカ（開発者）へのコミュニケーションにあいまいさが入り込む場合とがある。

要求定義技術は現在のところ必ずしも完成されたものにはなっていない。システムの多様性故に，一般論としてはむづかしい点も多いからである。しかし，多くの事例に即してある程度の整理はされており，システムの基本機能，性能，コスト，信頼性，保守性，柔軟性，拡張性，安全性，技術的制約条件，法規制的制約条件などが基本的な要求項目と考えられる。その他，入出力情報の種類とフォーマット，使用環境条件の規定なども必要である。これらの項目について，発注者がまず整理したうえで一定の形式，一定の用語を用いて文書化し，その後，開発担当者と充分打合わせて最終的な文書の形で要求仕様をとりまとめる。要求の概念が誤って伝えられたり，開発者側の技術レベルや技術的制約が考慮されていないと，不都合を生じることになる。また打合せの段階で発注者側の不備や矛盾，技術的見込み違いが指摘されることもある。

要求定義の初期段階では，“何をなすべきか”が純粋な形で議論され，検討された方がよい。つまり種々の制約を考慮からはずして，要求そのものだけを明らかにした方がよい。そして，その後の段階で技術的，コスト的，時間的，法規制的な制約を順次付け加えていって，それらの枠の中で要求を実現できる範囲とレベルを具体的に絞っていく。言い換えれば，システム開発に関する“必要性（ネセシティ）”と“可能性（ポシビリティ）”を初めは混同せずに分離して検討し，後に両者の融合・妥協をはかるべきである。いずれにしても，要求仕様の仕上げには充分な時間をかけた方がよい。この段階での手戻り（フィードバック）はむしろ必要な手戻りであって，発注者と開発者双方の擦合せには念を入れ，手間を惜しんではならない。

なお要求定義と要求仕様については，本報告書の第1章及び第2章の各所に具体例に即した記述があるので参照されたい。

(3) システム設計とハードウェアの選択

要求定義をwhatの追求とすれば，次の設計はhowの追及である。いう

までもなく，マイクロコンピュータ応用ではハードウェアとソフトウェアの兼合い，分担を決めることが大切であるが，量産品（一般にソフトの比重を上げることが多い）か否か，ハードとソフトそれぞれのツールとリソースがどの程度整っているか，それぞれの開発担当者の質と量はどうか，などにより分担のさせ方が違ってくる。作業効率上からは早期に分けた方がよいが，システム設計の段階で充分に全体的検討を行うこと，分割後も密接な連絡体制を作っておくことは肝要であり，ソフト側への安易なしわ寄せは避けるべきである。

マイコンチップの選定はハード設計の出発点であるが，これについては第2章の諸節を参照して頂くとして，チップそのものの機能の他に，ソフトとならびに開発ツール（開発環境）の整備状況と周辺機器のアベイラビリティが重要なポイントであろう。前述したように，チップ技術は今後とも急速に進歩するであろうが，その際，上方互換性の思想の整っている系列のものはシステムの将来の発展・拡張をはかるうえで有利である。インタフェースと周辺機器は応用システム開発で欠かせない要素であり，それらの標準化，互換性の改良はユーザ側の強い要求である。

ハードウェアの規格の統一化，標準化については強い要望が叫ばれて久しい。しかし一部を除いては依然として事態は進展していない。技術の進歩の激しい高位機種については標準化は事実上無理として，開発のピークを過ぎたような機種（4～8ビット機種）については，なるべく早急に標準化の手段を講じるべきであろう。技術的成熟の段階で，利用上は数量的優位が続いているものは，そのメリットが大きいはずである。

周辺機器のうち，コンピュータ技術の範疇外ではあるが，センサ技術は将来のシステム開発動向を大きく左右するものである。これはディジタルのコンピュータ技術と異なって，物理・化学・生物的なアナログ現象に深く関与している。最近のディジタル世代の技術者の中にはアナログ技術を軽視したり，疎外したりする傾向が見られはしないか。システム開

発者は直接センサの開発に携わることはないかもしれないが，仕事に応じて各種センサの特性と将来動向に絶えず配慮する姿勢が望まれる。

(4) ソフトウェアの開発と管理

ソフトウェアは応用対象システムのハードウェアとその機能を，マイクロコンピュータ・システムのそれらと結合させる技術の総称と解釈できよう。さらに，最終的には人間との結合をはかる重要な側面があるのを見落すわけにはいかない。応用の対象が拡がりマイクロコンピュータが変われば，当然ソフトウェアも開発サポートツールも変わるので，ソフトウェアの生産技術は一向に成熟することがない。生産性の向上のペースは相対的に遅々としている。

ソフトウェアの生産性を上げ，信頼性，保守性，拡張性を改善するためには，開発支援環境の整備，ソフトウェア工学の導入，高位言語や軽装OSの利用をより積極的に考慮すべきであろう。高位言語については，実行速度，記憶容量，制御効率，機種互換性，習得の容易さなどの点で一層の改良が期待される。ソフト開発の効率化については次節で改めて詳論される。

ソフトウェアの生産工程管理にも問題点が存在する。すなわち，製品化の途中段階で作業の進捗状況を計量することが困難であるために，スケジュールのモニタリングがむづかしく，トラブルが生じていたとしても対策が立てにくい。これはソフトウェア評価の問題とも共通点を持っているが，モジュール化の徹底によってある程度の解決をはかり得るのではなかろうか。モジュール化はまた，製品としてのソフトウェア群の管理と再利用のためにも大いに有効であろう。

(5) ドキュメンテーション

コンピュータ応用システムの開発は，すぐれて知的な作業である。しかも，ルーチン・ワークを緻密に組立てていくという種類の仕事だけでは新しい開発はできないので，創造的なプロセスを含んでいる。開発者の頭脳から創り出された成果品はそれを手に取って見ただけでは理解できず，適

切な解説とコメントが必要である。それらを体系的に整理して記述したものがドキュメントである。

ドキュメンテーションは成果品についてばかりでなく，要求定義といった初期の段階からほとんどあらゆる工程を通じて必要なものである。それは関係者相互の的確なコミュニケーションに不可欠であるばかりではない。開発者本人にとっても，思考の一貫性を保障し作業の冗長性を省くために有用であり，また後々の開発と利用にとっても欠かせない。そして最終的な段階で，システム開発者用，システム保守管理者用のドキュメントと，システムの日常の利用者用のマニュアルを適切に整えておくことは最も重要である。

組織上の問題で，開発担当者は数年単位で交替する。そのとき，特にソフトウェア開発の継承性が問題になることが多い。ドキュメンテーションの不完全なソフトウェアは他の開発者にとって理解しにくいもので，結局再利用されることなく埋もれてしまう。そして新しい開発のための二重，三重の無駄な投資を余儀なくされることになる。利用者にとっても，わかり易いマニュアルが無いときはソフトウェアは得体の知れないブラックボックスということになり，効果的な利用にとって大きな妨げとなる。若い技術者は優れた開発者であり得ても，多忙な故もあって，必ずしも優れたドキュメンテータではない。直接の開発者よりも，過去の経験と広い視野を持った成熟したレベルの専門家の方が，そして素人の立場をも理解できる者の方が，むしろ第三者の立場から見た客観的に優れたドキュメントを作成できるであろう。ドキュメントのノウハウを体系的に整理し，それを身につけた専門家の育成を今後真剣に考えるべき時期であると思われる。

3.2 ソフトウェア開発の効率化

マイクロコンピュータの適用範囲の拡がりはシリコンテクノロジィの急速な進歩に支えられたマイクロプロセッサの低価格化，高機能化，高速性に負うところが大である。しかしその一方でマイクロコンピュータの応用技術の進歩の足取りはゆっくりとしており，特にソフトウェア生産技術の進歩の遅れから情報技術者の不足が社会問題化している。

ソフトウェアの定義は難しいが一つの観点としてインターフェイスとして把えることができる。インターフェイスとは2つのものを整合させるものであり，マイクロコンピュータのソフトウェアとはマイクロコンピュータの持つ各種リソース，例えばマイクロプロセッサの演算能力，メモリ，ファイルなどを利用環境に於けるニーズに合致させるものと考える。このためマイクロコンピュータのソフトウェアとは単にプログラムを指すのではなく，プログラムが書き易くなるようなハードウェア設計も含めて考える必要がある。またさらにオペレーションの方法もソフトウェアと考えられる。

マイクロコンピュータの適用範囲が拡がることはそれぞれの適用毎にソフトウェアが必要となることを意味するわけで，ハードウェア技術の進歩が続く限りソフトウェア技術者の不足は解消できないとも言える。しかしながらソフトウェア生産技術の進歩がないのではなく着実にソフトウェアの生産性は向上している。現在はハードウェア技術の進歩と適用範囲の拡大の速度との差が顕著でソフトウェア開発が問題視されている。

汎用大型計算機の大規模で複雑なシステムの開発で問題となったソフトウェア開発における工学的アプローチはソフトウェア工学として広く研究が進められてきた。ここではマイクロコンピュータ応用システムのためのソフトウェアエンジニアリングを展望してみよう。

3.2.1 ソフトウェア工学

ソフトウェア工学とは〝与えられた仕様を満たすソフトウェアを

予定された期日までに予定された費用で構築するための技法，手法，理論などの総称〃と言うことができる。

１９６０年代の後半より〝ソフトウェア急機〃に対処するための研究が盛んとなった。

計算機の大型化に伴ない，座席予約システムや銀行システム，さらにプラント制御からロケット制御などの複雑で大規模なシステムの開発が行われた。またオペレーティングシステムの巨大化も進み，ソフトウェア開発の難しさが露呈してきた。

大規模で複雑なシステムを高い信頼性を持って実現することが要求される。ここで高い信頼性とは〝要求された性能と機能を満足する〃ものである。このためソフトウェア工学はまず高信頼性ソフトウェアの問題として考えられた。

ソフトウェアの信頼性の向上のためにテスト技術の確立が求められた。

この結果として現在出荷されるソフトウェア製品の多くはある基準の製品検査を受け，残存バグの量についての予測ができるようになった。

プログラムの制御フローの各分岐のどれだけを通過するテスト・データを用いたかを示すテストカバレッジは，テストの精度を示す指標として利用されるようになった。

ソフトウェアの検査についての研究とともにプログラミング技術の研究も進み，モジュラー・プログラミング，構造化プログラミングなどの手法が広く一般に用いられるようになった。

しかし，これらの研究が進むにつれて，単にテスト技法やプログラミング技法の改善，向上だけでは信頼性の高いソフトウェアの構築は困難であることも明らかになった。これは開発されたシステムのバグの多くが，プログラミングの過程で生じるというよりもプログラム仕様書に誤りがあることが原因である。

これはプログラム設計の誤りに起因するバグと言える。しかもプログ

ラム仕様書の誤りのいくつかはその前段階のソフトウェア開発過程であるシステム基本設計における誤りや，さらには利用者から出されたソフトウェア要求仕様書にまでさかのぼらなければならない誤りもある。

ソフトウェア・システムの高い信頼性を保証するためには，利用者が真に欲している機能性能の正確な分析とその記述，さらにそれに続く各設計活動の検証を欠くことができない。

これまでソフトウェア開発過程で生じる仕様書のほとんどは我々が日常用いている自然言語で書かれてきた。これはソフトウェアが人間の最も知的活動と考えられる創造活動に属するもので，表現形式による活動の束縛を避けるためと思われる。しかし，その一方で自然言語による記述が冗長性，曖昧性などの点でソフトウェア開発の障害になっていることも多い。

ソフトウェア工学の目指す生産工程の管理のため，ソフトウェア開発の各過程での活動を検証できるような形式性の高い記述が望まれるよう

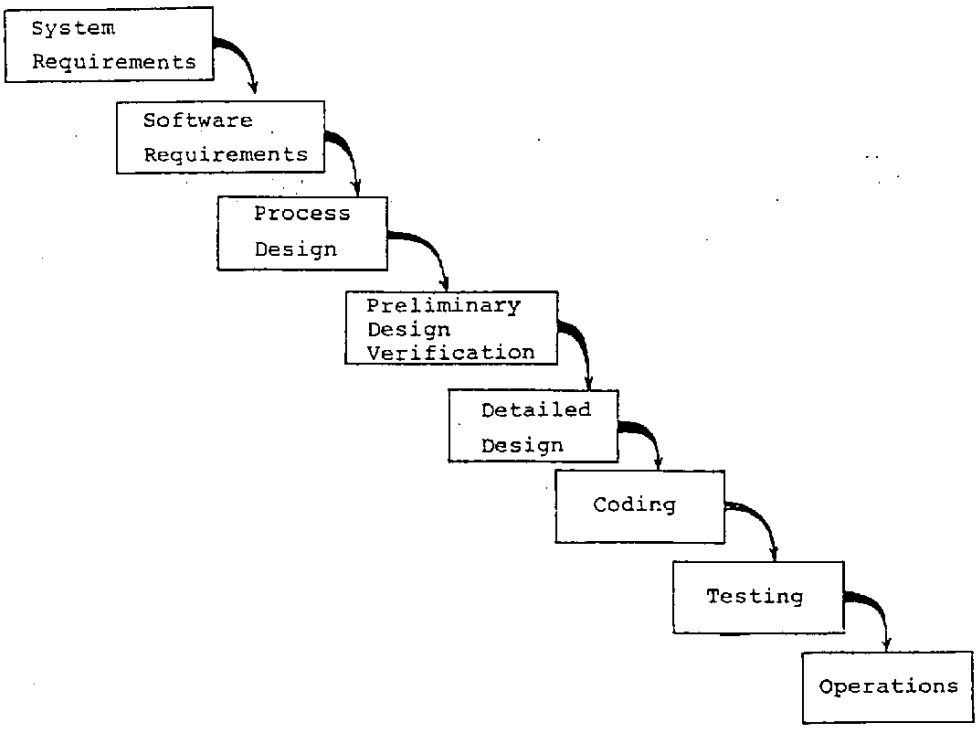


図３－１　ライフサイクルのモデル

になった。これにより開発が進んだ後からの後戻りを減少させることができる。

ソフトウェア・ライフサイクルを滝に見たてて，ウォータフォールモデル（ waterfall model ）と呼ばれることがある。（図3－1参照）

これは水の流れが一方向でさかのぼることのないことを示す。

各過程の出力は次の過程にに対する仕様書となる。この中でシステムの定義を行う要求仕様化ではその過程の入力は仕様書の形で与えられない。このソフトウェア開発の最も初期段階は，ソフトウェア・システムの置かれる環境の設定，ソフトウェア・システムと実社会の切り分け，真の問題の把握など非常に難しい問題を多く含む。この分野は〝要求工学〟として特に区別されることもある。

現在，ソフトウェア開発の問題をライフサイクル論で論ずることの問題点が話題になっている。ソフトウェアの要求仕様は必ずしも明確なものでなく，概念レベルの要求はいくら詳細に検討してもソフトウェア開発の基礎となるほど堅牢なものではありえないのではないかとの疑問が出されている。これに対処するためには，プロトタイプを利用者に提示して，ソフトウェア・システムの要求を確認するべきだとの主張である。

ソフトウェア・システム開発の初期でのプロトタイプということでラピッド・プロトタイピング（ rapid prototyping ）と呼ばれる。

元来，ソフトウェア製品は一品料理でありプロトタイプ作製はコスト的に引き合わないとされてきた。ここで再びプロトタイプの議論が活発になってきた背景としては，要求仕様化技法として優れた方法が見出せなかったことが原因であろう。しかし，一方ではソフトウェアの開発経験の蓄積から，各種ソフトウェア・モジュールの部品化／再利用が着実に進み，ソフトウェア・システムのスケルトンが各々の応用毎に用意されたため，プロトタイプの作製コストが低下しているという積極的な理由もあげられよう。

3.2.2 マイクロコンピュータ応用システムソフトウェアの特徴

ソフトウェア工学は大規模で複雑なシステムの開発の鍵を与えるものとして議論され研究されてきたもので，マイクロコンピュータ応用システムのソフトウェア開発に役立つものであるかどうか疑問視する向きもあった。しかし，現在のマイクロコンピュータ時代を支えている半導体技術の飛躍的な進歩は既に３２ビット・マイクロコンピュータを世に送り出すまでになっている。

その能力は〝ソフトウェア工学〟が生まれた時期の汎用大型計算機の能力を凌駕するものである。

マイクロコンピュータの能力の向上に伴ない，その応用分野も拡大しているが，これまでの応用を考えると４ビットマイクロコンピュータに代表される低価格で大量に消費される応用分野を無視することはできない。すなわち，マイクロコンピュータの高性能化，低価格化が進めば全てのマイクロコンピュータ応用システムの心臓に３２ビット・マイクロコンピュータが埋め込まれるわけではなく，更に低価格なマイクロコンピュータを利用する用途が拡がっている。

単に１つの機械的カムの置換え，１つの時間遅延リレーのタイマ部にマイクロコンピュータが使われるようになることも考えられる。

すなわち，マイクロコンピュータ応用システムにおけるマイクロコンピュータは主役ではなく，ほんの１つの部品にしか過ぎない。

マイクロコンピュータは部品として大量に生産され，利用されるということから，これまでの汎用大型計算機のソフトウェアとはかなり異なるものである。以下にマイクロコンピュータのソフトウェアの特徴を考察して見よう。

(1) システムとしてのソフトウェア

システムとは独立に動作しうる複数のコンポーネントが一体となってある目的を遂行するものと考えられる。ソフトウェアは〝利用技

術〃と呼ばれるように，あるハードウェアが与えられた環境下での，また制約下での解法を与えることと考えられがちである。

マイクロコンピュータ応用システムの場合には，マイクロコンピュータは1つの部品であり，システムを構成する一つの部品にしか過ぎない。すなわち，システム全体の設計を通してマイクロコンピュータの立場，求められる機能が決定される。このためソフトウェアの仕様の決定が非常に曖昧なものとなる。

ソフトウェアとハードウェアを統一的に論ずるシステムとしての工学の確立の下で，ハードウェア／ソフトウェアの融合が図られなければならない。（図3－2参照）

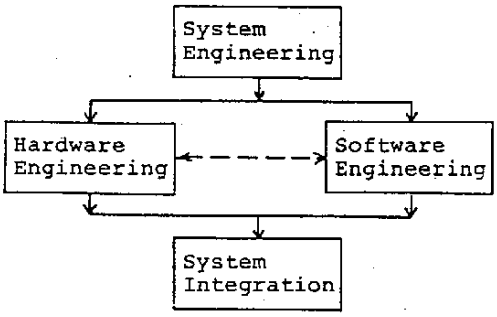


図3－2　ハードウェア／ソフトウェアの融合

ハードウェアの設計活動はかなりCAD（Computer Aided Design）が進み，CADなしには設計が考えられないところにまでなっているが，ソフトウェアも同様のレベルに急いで持ち上げなければならない。

(2) リアルタイム性

汎用大型計算機の利用技術の進歩を顧みると，単純な計算処理をバッチで行うものから始まり，プラント制御，ロケットの飛行制御，座席予約システム，銀行のオンライン化などの高度の利用形態に移ったと言われている。ここでの高度の利用形態とは，オンラインでリアルタイム処理を要求されるということである。

マイクロコンピュータの多くは制御機器に組込まれ，本質的にリアルタイム・オンライン処理に利用される。すなわち計算機利用としては最も高度な形態であるものを，最も簡単な計算機で実現しなくてはならない。マイクロコンピュータ応用システム開発とはある意味で最も難しいものと言える。

(3) 計算機ハードウェア資源の制約

マイクロコンピュータの性能が速度の面だけでなく，そのアドレス空間の拡大，命令セットの正規化（アドレス指定モードが命令の種類によって制約を受けないこと）などプログラミングのし易さ，すなわちプログラミング効率の面でも向上してきている。汎用大型計算機の歴史を見ると，メモリ空間の制約からオーバレイ技術が開発され，次にオーバレイをユーザから見えなくするために仮想記憶方式が提案された。このように計算機を使い易くするためには計算機ハードウェア資源の制約からの解放が重要である。

マイクロコンピュータのアドレス空間は十分に大きくなり，アドレス空間の制約を受けなくなったと言われる。しかし，マイクロコンピュータ応用システムの多くはそのコストの減少，電力消費量の減少，サイズの最小化などのため，1チップ・マイクロコンピュータの採用を図る場合が多い。この場合，1チップ・マイクロコンピュータに置かれているＲＯＭ／ＲＡＭは当然限られたものであり，またその容量はかなり少ない。外部にメモリを拡張できるものもあるが，この場合

① 1チップ・マイクロコンピュータを使う意味が薄れる。

② 入出力制御に利用するピン数が減少する。

③ コストが上昇する。

などの不都合がある。

このため，1チップ・マイクロコンピュータを応用したシステムでは記憶容量の制限を強く受ける。また，多チップ構成のマイクロコン

ピュータであっても，ROM／RAMチップの個数の削減をコストの点から強く求められることは少なくない。これはマイクロコンピュータ応用システムが，テレビ受信機，エアコンディショナ，自動車などのような大量生産製品に組み込まれるためである。このような大量生産製品では設計コストの最終生産品のコストに占める割合は大きくなく，多少設計に費用がかかっても製品の材料費を小さくすることが望まれる。すなわちプログラマに無理強いしてもROM／RAMのチップ数を減らす必要が生じる。

(4) モニタの欠如

計算機の利用技術の向上は単にハードウェア技術の向上に支えられてきたのではなく，ソフトウェア資産の蓄積によって支えられてきたと言える。汎用大型計算機の利用者の多くは図3に示すように，裸の計算機の上に幾層かのソフトウェアを着せ，その外側から利用していると言える。

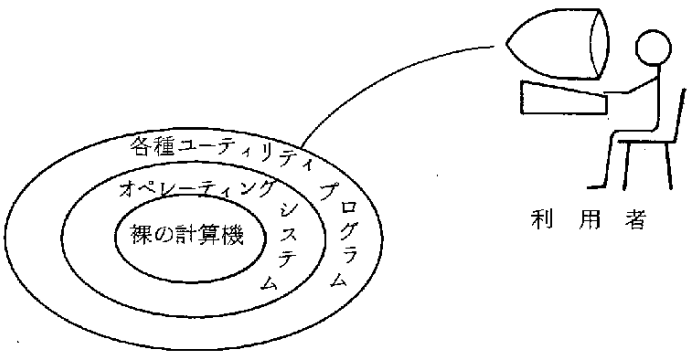


図3－3　ソフトウェアの階層化

汎用大型計算機の利用者にとっては階層化されたソフトウェアによって提供されるサービスが重要であって裸の計算機の与えるサービス，すなわち命令セットは必ずしも意味を持たない。特にプログラム開発を行わない単なる応用プログラムの利用者にとっては計算機がいかなるものかは関心外である。

プログラム開発者にとっても，もしその人が高級言語を用いてプログラム開発を行っているとすれば，プログラミング言語の機能が重要で計算機自身は重要ではない。マイクロコンピュータ応用システムにおいても，例えば，UCSDの p－system を利用する者にとってはそれを実行しているマイクロコンピュータが i8080であろうが TMS9900であろうが関係はない。また，現在広く利用されているMicro soft 社のBASICにより，80系と68系のマイクロコンピュータの区別なしに利用できることはよく知られている。

このように計算機が幾層かの階層化されたソフトウェアによって高度な機能を供給できることが，ソフトウェア開発者の労力削減を進め，より複雑なシステムの開発を可能としたと言える。

マイクロコンピュータ応用システムの多くは組込み用であり，モニタ（或いはオペレーティング・システム）の下で実行されるものは少ない。現在リアルタイム・オペレーティング・システムと呼ばれるものがいくつか採用されているが，それほど一般的ではない。

モニタの機能は大きく分けて2つある。1つはプログラム開発者の手作業の軽減であり，他の1つはシステム資源の有効な利用である。勿論，動作速度の極めて遅い人間（プログラム開発者）とのインタラクションを減らせばシステム資源が有効に利用できるので両者は必ずしも相反する目的ではない。

2つの機能，側面を持つモニタは組込み時には1つの機能のみ，すなわちシステム資源の有効利用のみが要求されるので，その切り離しができなければならない。また，モニタの重要な機能であるプロセス切換え，プロセス間通信が十分に高速でなくてはならない。

そのようなモニタが提供されればプログラム開発者はひとかたまりの仕事を1つの論理的なプロセスとして記述でき，問題の見通しが良くなってプログラム開発の効率化が図れるものと考える。

(5) 開発支援環境の不備

マイクロコンピュータの高性能化により，マイクロコンピュータとミニコンピュータの区別がつきにくくなってきた。計算機の処理能力自身は向上してきたが，プログラム開発環境としてはいまだ貧弱な状況である。その原因としては

① チップの価格からシステムの価格を類推して開発支援システムへの投資が少ない。

② ハードウェア開発用ツールの発達の割にソフトウェア開発用ツールが遅れている。

③ ソフトウェアに対する認識が低い。

④ これまでどうにか人海戦術で解決ができてきた。

などがあげられる。

開発支援環境の整備は単にプログラム生産性の向上ばかりではなく，各組織の技術の蓄積，伝承の役割も果す。ソフトウェア開発は人間の知的活動であり，一般に知的活動とは経験や学習を通して得られた知識をもとに行うものと言える。そこで知的活動の支援ができるシステムとはかなりの知識がその中に収められているはずであり，それはその組織の知的資産となる。

3.2.3 マイクロコンピュータとソフトウェア工学

マイクロコンピュータ応用システムの開発過程はソフトウェア工学で提唱されたライフサイクルとかなり異なる。すなわちハードウェア設計・開発の過程がソフトウェア開発設計・開発の過程と並行に存在するため，それとの同期，インタラクションを避けることができないためである。

現在多くのマイクロコンピュータ応用システムの開発過程は図 3－4 に示されるようなものであろう。

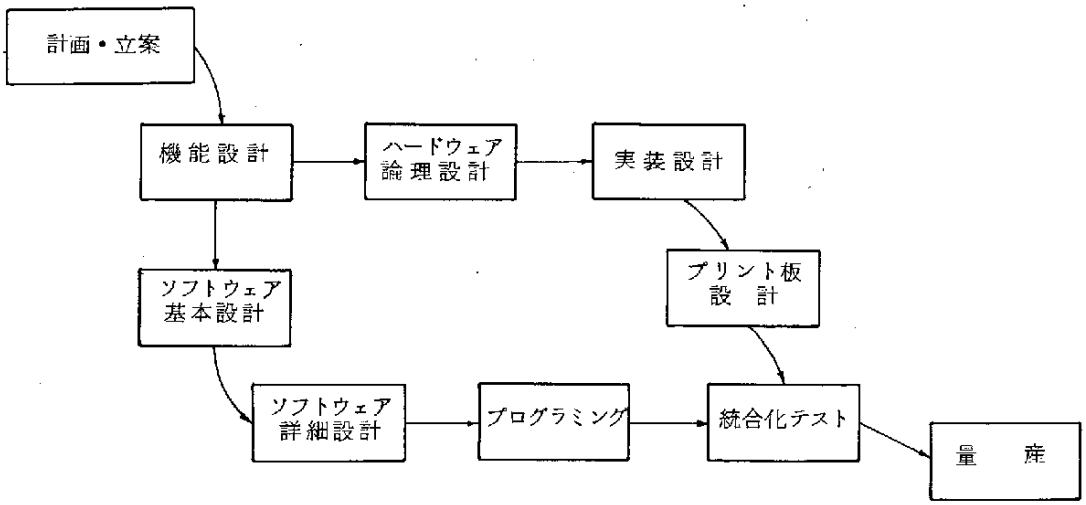


図3－4　マイクロコンピュータ応用システムの開発

すなわち，計画・立案と呼ばれる要求分析，市場調査の後にマイクロコンピュータ応用システムの像を明確化し，機能設計としてそれをとりまとめる。

その後，コスト，速度，技術動向などを勘案して，ハードウェアで実現する部分機能とソフトウェアで実現する部分機能に分割する。

ハードウェアの設計・製作には論理設計，実装設計，プリント板設計などが必要となり，これまでの多くの経験から実装設計後，実装済みのプリント板が得られるまでの日数が知られている。このため最終製品を必要とする時期から逆算してハードウェア／ソフトウェアの切分けの時期が決められる。

当然のことではあるがソフトウェア開発期間の見積りも行われる。

しかし，ソフトウェア工数の見積りの算出の根拠が曖昧であり，一般的には弱い立場にある。そのためハードウェア開発スケジュールに合わせた形でソフトウェア開発スケジュールがたてられる。これにより，開発期限が近づくと大量の人員が投入され開発が行われる。また，それが可能である。

ソフトウェア工学の一つの分野にソフトウェアの工程管理がある。工程管理のためには作業量の見積りと製品検査の手法を欠かすことはできない。このためには

① ハードウェア設計過程で生じるソフトウェア変更要求を認めない。

② ソフトウェア要求を明確化しておく。

③ テスト基準をはっきりさせておく。

ことが重要である。

ソフトウェアは固定されたハードウェアと多種多様な利用形態のインターフェイスを司るものといえる。このため，ハードウェアが異っていたとしても同じ利用形態を提供することが可能である。例としてはＢＡＳＩＣインタプリンタは中心となるマイクロコンピュータの種類，ハードウェア設計に関係なく同じ機能を利用者に与えている。

このソフトウェアの汎用性がマイクロコンピュータを含めた電子計算機の最大のパワーと言える。しかし，その反面でソフトウェア開発の効率化の阻害要因ともなっている。すなわち，ハードウェアの設計上であるゲートがどうしても収容できない時に，ソフトウェアで解決すればよいとされることが多い。

たとえばある時間幅のパルスを生成する場合にモノマルチバイブレータ用のＩＣを利用しても実現できるし，１つのラッチを用いてソフトウェアでＯＮ－ＯＦＦしても実現できる。このようなハードウェアの変更は容易にソフトウェアで対処できる。しかし，たとえ小さな変更でも度重なると収拾のつかない状況に陥る危険がある。

ソフトウェアを系統だてて開発してゆくためには外乱を減らすため，安易にハードウェア側からのソフトウェア変更要求を受け入れるべきではない。一つの解決法はハードウェア仕様とソフトウェア仕様を明確に分離しておかず，ソフトウェア機能，すなわち低レベルのモジュールとしてハードウェアの仕様を定義することである。

こうすることでモジュール・インターフェイス規約を遵守する限りハードウェアの変更は自由になる。そしてそのハードウェアの変更はソフトウェアの変更を要求しない。これはソフトウェアの分野で広く認められてきている。

情報隠蔽（ information hiding ）

抽象データ型（ abstract data type ）

の考え方に沿うものである。

人間の思考能力には制限があるので，不必要に詳細なことは見えなくしてしまうという考え方である。また，ある規約を満たすインターフェイスを介してのみ，ある機能にアクセスすることになるので，システムの信頼性の向上にも寄与する。

例えばあるモータをＯＮ－ＯＦＦするプログラムとして，

```
outp ( motor , ON ) ;
```

と書くとする。このとき，ソフトウェア設計者は motor の出力ポート・アドレス，ＯＮの極性（１でＯＮか，０でＯＮか）を知る必要はない。

しかし，

```
outp ( motor 1, ON ) ;
outp ( motor 2, ON ) ;
```

と２つのモータをＯＮする場合はどうであろうか。もし，motor 1 と motor 2 が同じ出力ポート・アドレスが割付けられているとすると

```
outp ( motors , 1-2-ON ) ;
```

としなければならないかも知れない。ソフトウェア設計者がそのような詳細を知る必要はなく，また知ろうとしてもハードウェア設計完了を待たなければならないこともある。

そこで，モータをＯＮするときは

```
outp ( motor , ON ) ;
```

とすると決めておき，これが呼ばれたときの動作の保証はハードウェア

設計者の責任とするものである。マイクロコンピュータ応用システムにおけるハードウェア技術者はハードウェアのドライバ・ルーチンを含めて設計しなければならない。ハードウェアのデバッグ作業に簡単なドライバ・ルーチンを作ることは必ず行われており，この作業は新たな仕事を意味するものでない。

ソフトウェアは一般に人間インターフェイスを与えるもので，どのようなインターフェイスを与えるべきかは人間工学的，心理学的な配慮が必要とされる。ソフトウェアの要求仕様はこのような複雑な要素を含み，その記述は曖昧性を含むことが多い。

曖昧性，冗長性を含む要求仕様書に基づくソフトウェアの開発は当然ながら失敗に終る。

ときとしてシステム完成後，その利用者からシステムの振舞が利用者の意図していたものと異なることが指摘される。この場合，システム開発の全体のやり直しにもつながり，システム開発期間の大幅な延長と開発コストの増加をもたらす。

正確な要求仕様の記述はシステム開発の成功のための十分条件ではないが必要条件である。要求の分析およびそれに続く要求の記述は防衛的役割であるのであまり重要視されていなかったと言える。

要求の記述，すなわち要求として示されたシステム・モデルが利用者の期待しているものと同一であるかのチェックが重要である。

この検証のためには要求記述言語として従前より利用されている自然言語ではなく，より形式性の高い言語が必要となる。

形式言語としてはミシガン大学ＩＳＤＯＳプロジェクトで開発されたＰＳＬ（Problem Statement Language），ＰＳＬを基にリアルタイム性を考慮して拡張されたＴＲＷ社のＲＳＬ（Requirement Statement Language），ソフトウェア・システムの青写真を目指したSoftech社のＳＡＤＴ（Structured Analysis and Development

Technique ）のＳＡ図式などがある。

これらはいずれも要求定義仕様書の形式的記述とその計算機処理を行うためのシステムで，ソフトウェア・システム要求に含まれる矛盾，一貫性の欠如，曖昧性などを検出することができる。また各種のレポートの出力も可能であり，システム開発の契約書としての利用もできるようになっている。

ソフトウェア仕様書が明確なテスト基準を与えることがある。すなわち，開発過程で得られる各種の仕様書はテストの過程でそれぞれのレベルでテスト基準として利用される。

各モジュールの機能を定義するプログラム仕様書は作られたモジュールの機能とそのインターフェイスが正しいかどうかの基準を与える。

また，ソフトウェアの要求仕様書は運用時における利用者との間のインターフェイスが定義されており，運用のテストにおいて正しさの根拠を与える。

しかし，ソフトウェア要求仕様書の正しさの検証をシステム開発の終了時に行っていては，もしそれに誤りがあると分っても無意味である。

そこで，ソフトウェア・ライフサイクルの各過程での検証が望まれる。

図３－５に示すようにソフトウェア要求仕様書に基づき設計された設計をもって要求仕様書通りに作れるか（実現可能性：feasibility），利用者の要求がどんな形で実現されるのかを調べるような ➡ で示された経路が必要であろう。この矢印の経路は１つ前の開発過程の検証と考えられるが，見方を変えると，次の開発過程の活動が指示されたものであるかどうかの検証とも考えられる。いずれにせよソフトウェア・ライフサイクルでの大きな後戻りを排除し，各過程毎にその活動の正しさを確認しておこうとするものである。

マイクロコンピュータ応用システムでは単にソフトウェアの議論にとどまらず，必ずハードウェアの影がつきまとう。

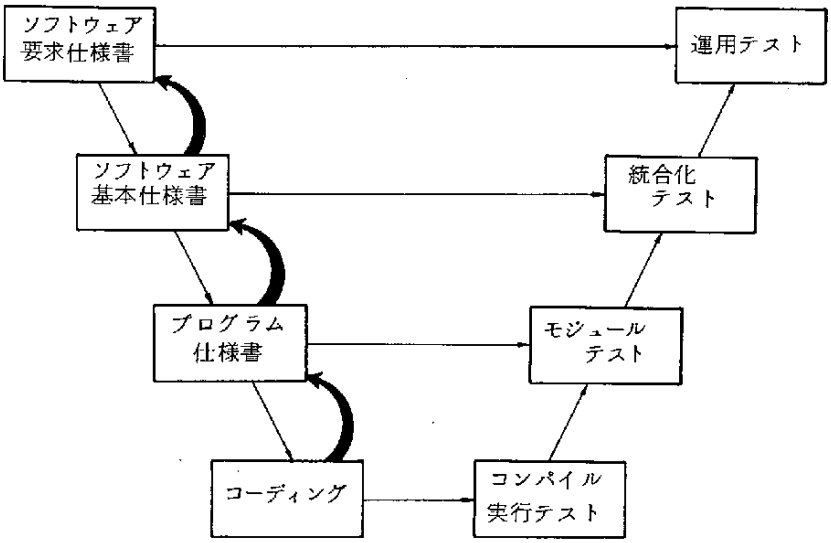


図３－５　仕様書とテスト

しかし，ハードウェアとソフトウェアは完全に分離して考えられるものではないので，逆に積極的に統合化して考えるべきであろう。すなわち，ハードウェアもソフトウェアも情報の論理的操作を行うものであり，論理機能の明確化，詳細化を行う作業がマイクロコンピュータ応用システムの設計であると把える。

統合化設計においてはハードウェアとソフトウェアの切り分けは設計が充分進んで行うため

① ハードウェア／ソフトウェアのトレードオフの設定が容易である。

② ハードウェア／ソフトウェアのインターフェイスに矛盾が生じる危険性が少ない。

③ システム設計の自由度が高い。

などの利点が得られるものと思われる。

マイクロコンピュータ応用システムの特徴であるハードウェアの選択の自由度の大きさを十分に生かしたソフトウェアの開発手法はいまだ確

立されていない。今後ソフトウェア工学での成果を踏まえ，ハードウェアと融合した形でのマイクロコンピュータ工学とも言える研究が進むことが望まれる。

3.3 ソフトウェア・ワークスベンチ

— より優れたソフトウェア開発環境を目指して —

ソフトウェア技術者の管理は非常に困難なものと言える。ソフトウェア開発の中でプログラミングの工程ではどれだけのステップ数のプログラムを書いたかでかなり進捗状況が把握される。しかし設計過程ではボンヤリと天井を見ているのか考察を進めているのかの区別はつかない。

ソフトウェア工程管理の難しさはソフトウェア設計活動の状況が把握しにくく，活動の最終成果である文書が提出されるまでどのように作業を進めているのか分らないからである。ソフトウェアの設計活動を効率化するにはその作業を顕在化させ，全ての活動を陽に記録することが必要である。これは設計活動を通して得られる経験・知識のデータベース化にもつながる重要なポイントである。

ここでは開発過程の後半とも言えるプログラム設計からプログラミングにおける作業環境を考える。

3.3.1 ソフトウェア・ワークベンチ

ソフトウェア技術者の作業台を考えよう。

どの分野の技術者にもその技術者の活動に適した作業机が与えられる。例えば建築家には製図板と構造計算をするための電卓，そして建築に関する法規集があるであろう。技術者の行うべき仕事が明確化されればそれに適した作業机が与えられる。

Bell 研究所で開発されたＵＮＩＸオペレーティング・システムの利用形態の一つにプログラマズ・ワークベンチ（ＰＷＢ）と呼ばれるもの

がある。これは汎用大型計算機のソフトウェア開発において，細々した仕事をプログラマの傍らのミニコンピュータ上のＵＮＩＸを利用して行うことでソフトウェアの生産性を上げようとするものである。

ＰＷＢでは各種のソース・ファイルの編集，ソフトウェアの世代管理，カタログ化したジョブ制御，そして汎用大型機との通信の管理などを行う。これによりミニコンピュータの小回りの良さと汎用大型機の大きな計算処理能力を与える。

ソフトウェア技術者の多くはプログラマであるが，ソフトウェア設計における支援を考えると単にプログラムの編集，デバッグ機能だけでは問題とならない。

ソフトウェアに限らず設計とは与えられた仕様から１つの解決モデルを作り，そのモデルが仕様に照らして妥当なものかどうかの評価を行う。

そしてその評価結果が満足のいくものであれば，それを決められた表現形式で記述する。

例えば，ある電気技術者が電子回路の設計を行う場合を考えよう。彼は各種の設計ハンドブックやそれまでの経験を基にラフなスケッチを描く。その中でクリティカルな抵抗やコンデンサの値をある方程式を解いて算出する。そしてその値をいくらか変化させて全体に及ぼす影響を理解する。そして希望する機能，性能が出そうだと確信がもてると，他の詳細な部分の設計を進め，電子回路図として書く。計算機援用として電子回路シミュレータや電子回路図描画などが与えられるだろう。

ソフトウェア設計者にとっては〝クリティカル・パス・アナライザ〟，〝抽象実行ツール〟〝文書管理支援ツール〟などが作業机に必要であろう。

ソフトウェア設計の難しさは性能と機能のバランスにあると言える。

この意味でオンライン・リアルタイム・システムはその構築が難しい。設計によって詳細化されていく機能がそれぞれどの位の時間で実現され

なければならないかを考える必要がある。システム全体としての性能を決定する処理のパスを解析し，そこで仮定されている処理時間が妥当なものであるかどうかを調べるクリティカル・パス・アナライザは強力な設計道具となろう。

ソフトウェアの機能を定義する手法には

① 操作的意味記述

② 公理的意味記述

③ 関数的意味記述

がある。現在，ソフトウェア科学の研究者の多くは関数的意味記述に目を向けている。しかし，ソフトウェアがある処理の並びで与えられた機能の実現を図るものであることを考えれば，操作的にその意味を考えることは極自然である。

すなわち設計されたシステムを何らかの形で動作させ，その振舞いを観察することで自分の設計が正しいかどうか調べようとするものである。

勿論，設計された結果がプログラムであるようなソフトウェア開発の下流では単にコンパイルしてランさせるだけでよい。

しかし，ソフトウェア開発の上流ではその設計活動の結果はすぐに計算機で実行されるものではない。

設計の深度に合わせた抽象的な計算機を考え，設計結果をその抽象計算機のプログラムと把える。これにより設計活動の成果を抽象実行させ，設計の確認作業を行う。ソフトウェア設計結果の評価に際して，ある1つの仮りの入力がどのように処理されるかを設計書の記述の中で辿っていくことに対応する。

最近，ソフトウェア工学の分野を賑わしているプロトタイピングや実行可能な要求仕様記述（ executable requirement ）などもこの流れの1つとも言える。

ソフトウェアの権益保護に関する議論が盛んとなり，ソフトウェアは

著作権法に馴染むかどうかが話題となっている。ソフトウェアの開発とは文書の書換えの連続であるとも言える。すなわちソフトウェアは書物である。

利用者，または発注者の意見，希望を理解し，その利用環境を掌握し，それをシステム要求仕様書として記述する。次の設計活動はその記述を読み理解し，ハードウェアの制約や予算，納期などを考慮してシステム基本設計書を書く。このようにソフトウェア設計過程の各段階毎に文書が作られ，その文書を頼りに仕事が進められる。

ハードウェア設計においても論理式，タイミング・チャート，論理回路図，実装図など多く記述が必要となるが，それらのかなりの部分は計算機援用がされている。計算機援用による図画の管理の利点は

① 管理が容易である。

② 他のドキュメントが生成できる。

③ 保守・変更が容易である。

ことなどがあげられよう。

この中で特にドキュメントの自動生成はある意味での自動設計を意味する。すなわち論理回路図が計算機に収められていると，そのデータから布線表，テスト・データなどの生成が考えられる。

ソフトウェア設計においても積極的に計算機への文章入力を行うと各種の文章作成の工数が削減できるものと期待される。一つの例として，要求仕様記述はシステム機能と性能を外部から眺めたものとも考えられるので，要求仕様記述からユーザ・マニュアルや操作マニュアルのかなりの部分が作られる。

ソフトウェア開発効率の向上には〝文書化〟が重要であることは衆知の事実であるが，文書化とは開発過程を全て文書で残すべきであるという主張と，文書化の労力を減らすべきであるという主張がある。

莫大になりすぎた開発ドキュメントをいかに整理し提示するかと，い

かに文書化の効率を高めるかが課題となっている。

ソフトウェア技術者の作業机には高機能ワードプロセッサが欠かせない。単に清書のためのものではなく，ソフトウェア開発の各過程，開発者の観点に対応した視点（ view ）からの切り出し作業の自動化，過去のソフトウェア開発の文書からの類似性を調べるためのデータベースと結合されたものが望まれる。

またソフトウェア設計で利用される各種図式，例えば，ＨＩＰＯ（ Hierarchical Input , Process and Output)図，ＰＡＤ（ Problem Analysis Diagram ）図，流れ図（ flow chart ）などが扱えることも必要である。

3.3.2 ソフトウェアの部品化と再利用

ソフトウェアの開発は人間でなければできない創造的な仕事であると思われてきた。確かにある特殊なソフトウェアには独創性を感じさせるものがある。しかし，これまでのいくつかの報告ではソフトウェア製品の７５％以上は過去のプログラムの再構成で得られていることが示されている。

ソフトウェア開発の多くの部分は

① 過去のモジュールの組合せ

② モジュール・インターフェイスの変更

③ ハードウェア環境変化への対応

によって行われていると言える。現在パーソナル・コンピュータの多くに簡易言語と呼ばれるノン・プログラマ用の言語が用意されている。これはオフィスでの処理の基本要素を用意し，それらの組合せでほとんどのオフィスの仕事ができていることを示す。

ソフトウェアは決して個人的なものではあり得ず，必ず他の人に変更され利用される公共の資源と考えなければならない。他の人がそのソフ

トウェアの機能の一部だけでも利用できるように，各機能毎に部品化されたモジュールの組合せとしてソフトウェアは実現されなければならない。

ソフトウェアの部品化とは，１つずつの部品が自己完結（self contained ）しており，しかも単一の機能を提供するようなものである。

部品として利用できるためにはその機能とインターフェイス条件が明確化されており，また十分信頼のおけるものでなければならない。部品を利用する立場からは部品の内部構造は知る必要もなく情報隠蔽が実現される。

ハードウェアの論理設計においては標準のＳＳＩ／ＭＳＩを用いて行うことが多い。設計者はＩＣマニュアルのピン配置，動作速度，ファンイン，ファンアウト，電気的特性などを考慮してそのＩＣを選択する。

しかし，その内部回路に立入ることはしない。特にトランジスタ・レベルでの考察を行うことは極めて稀であり，内部の論理的等価回路までに限られる。また，１つのＩＣのいくつかのゲートが余ったからといって気にもとめない。

ソフトウェアの部品もそのように利用されるためには

① 機能説明の標準化

② インターフェイス条件の標準化

③ ハードウェアからの独立

が欠かすことのできない条件である。利用する立場からはその部品を加工しようとすることを避けなくてはならない。ソフトウェアの特質である可変性，適応性がソフトウェアの部分化を阻げている主要な要因である。

これまでに各社でソフトウェアの部品化と再利用が図られてきたものの必ずしも成功はしていない。しかし，数１００Ｋステップを越すプログラムの開発の陰にはソフトウェアの再利用がある。例えば，特殊なマ

クロ命令やサブルーチン，ファイル構成などは以前のシステム開発からの継承であることが多い。

しかし，部品化を明確に意識してのソフトウェア開発がこれからは望まれるであろう。

3.3.3 ソフトウェア開発用オペレーティングシステムとしてのＵＮＩＸ

１６ビット・マイクロコンピュータの標準オペレーティング・システムとしての米国ベル研究所のトンプソン（Ken Thompson）らによって開発されたＵＮＩＸが注目されている。

ＵＮＩＸは巨大オペレーティング・システムＭＵＬＴＩＣＳの開発での反省から生まれ，個人の趣味（チェスゲームをするためと言われている）として作られたオペレーティング・システムで簡潔さ・使い勝手に特徴がある。また文書処理機能は優れた応用プログラムが多い。

ＵＮＩＸの特徴はいくつかあるが

① 移植性が高い

② ソフトウェア・ツール群が豊富

③ 安 価

などが，１６ビット・マイクロコンピュータでの主導権が得られた理由であろう。

ＵＮＩＸはその９０％以上が言語Ｃで記述されており，しかもソース・プログラムとして入手できるので，他機種（元来はＤＥＣ社のＰＤＰまたはＶＡＸシリーズの計算機用）への移植が容易である。また，その規模も４００Ｋステップ程度であり，本格的なオペレーティング・システムとしてはコンパクトである。

ＵＮＩＸを搭載した場合のライセンス料も余り高くなく，新たにオペレーティング・システム開発することを考えると非常に安くつく。また多くの応用プログラムのベンダーがＵＮＩＸ上のソフトウェアを提供し

始めており，マーケット的に有望である。

しかし，ＵＮＩＸシステムは単なるオペレーティング・システムとは異なる特徴を持つ。オペレーティング・システムは計算機システム内のハードウェア／ソフトウェアの各種資源を有効に管理し，システムのスループットを最大とするものと言える。この観点からは人間のシステムへの介在は最も避けるべきものであり，非人間的なシステムほどシステムの構成要素を最大限利用できる。

巨大なオペレーティング・システムがスループットやセキュリティを追求する余り，ユーザフレンドリなものとはかけ離れてしまっている。

ＵＮＩＸはソフトウェア開発のためのオペレーティング・システムであり，開発されたプログラムを効率よく実行するためのオペレーティング・システムではない。ソフトウェア開発環境としてのＵＮＩＸはコマンド・インタプリタshellに特徴を持つ。ShellはUNIXとユーザとの対話を助けるソフトウェアで，条件文，ループなどの制御構造が用意されており一種のプログラミング言語と見ることもできる。Shell をプログラミング言語として見て，ＵＮＩＸに用意されているソフトウェア・ツールを組合せることでかなりの仕事ができる。ＵＮＩＸの報告ではソフトウェアの３０％はshellで書かれており，その機能が満足できるもので，しかも性能に不満がある時には言語Ｃで記述し直すとなっている。これは一種のプロトタイプと見ることができる。

ソフトウェア・ツールを１つの部品として考えればＵＮＩＸでは自然な形でソフトウェアの部品化とその再利用が行えている。ここで部品化が行えるのはUNIXの標準入・出力の機能とその変更（redirection）が役に立っている。

ＵＮＩＸでソフトウェア・ツールとして使えるものは全て標準入力からデータを取得し，処理し，標準出力に結果を送出するものである。

また，そこで流れるデータは複雑な構造を持たず，単なるバイト列と

なっている。これにより部品化のための条件の１つであるモジュール・インターフェイスの統一がなされている。そしていくつかの部品を組合せることが可能となっている。

次に，ＵＮＩＸのカーネルとしてのプロセス間通信の方式にも特筆すべきものがある。ＵＮＩＸにおけるプロセス間通信の手段としてパイプ（ pipe ）と呼ばれるデータ転送方式がある。

パイプはメモリに置かれたバッファを介してデータを送受するものであり，標準入出力の結合が行われる。パイプで接続された２つのプロセスは当然独立に動き得るので，ユーザとの対話のあるようなソフトウェアも Shell で記述できる。ファイルを介して通信しなければならないシステムではファイルへの書き込みが終了するまで次のプロセスが待たされ非同期処理はできない。

また，ＴＳＳにおいて数秒の待ちは思考の中断をもたらし，ソフトウェア開発を著るしく遅らせると言われている。このように人間の活動を中断なく続けるにはコンパイルやリンクと言った時間のかかるジョブはバック・グランドで処理されなければならない。ＵＮＩＸでは

ＣＣ　　test．Ｃ　＆

とすれば，test．Ｃのプログラムのコンパイルをバック・グランドで処理できる。また，フォーグランドで実行中のジョブをバック・グランドに回したり，その逆も簡単にできる。

これらをうまく使うことにより，次々とプログラム・モジュールを編集することができる。

また，Ｃ－shell（カリフォルニア大学バークレー校での拡張 shell）ではコマンドの履歴が管理されていて，しかもその修正変更を可能としており，コマンドの再入力が簡単になっている。

ＵＮＩＸのソフトウェア・ツールとして最も強力なものはコンパイラ・コンパイラ yacc であろう。これはKuuth の提案になるＬＲ（１）法に

もとづく構文解析ジェネレータであり，構文規則入力からCで書かれた構文解析プログラムを出力する。我々もFORTRANプログラムでの変更波及効果解析システムの作成に利用したが，ソフトウェア・システムの多くが構文解析部にかなり人工を必要としていることを考えると有用なツールである。また，字句解析用に lex も用意されている。

ＵＮＩＸにおける主なソフトウェア・ツールを図３－６に示す。この他に文書処理プログラム nroff, spell, eqn, 関係データベースシステム ingvess, 通信制御プログラムUUCP（Unix-Unix Communication Program ）など多くの機能が盛り込まれている。

sort	ファイルのソート，マージ
uniq	同一行の削除
tr	文字の変換
diff	２つのファイルの比較（相異）
comm	〃 （共通）
grep	正規表現された文字列の検索
wc	文字数，語数，行数のカウント
cref	クロスリファレンスの作成

図３－６　UNIXにおける主なソフトウェアツール

単にソフトウェアの開発を容易にするだけでなくＵＮＩＸが１６ビット・マイクロコンピュータの標準的なオペレーティング・システムとなることで，ハードウェアから独立した標準インターフェイスが与えられることになる。これによりソフトウェアの共通化，流通性ができソフトウェア開発の励みとなる。

しかし，組込み用のシステム開発を考えると，ＵＮＩＸのカーネルがないと動かないようなシステムでは不都合で，実行用オペレーティング・システムはＵＮＩＸとは同一のものではない。現在リアル・タイム

ＵＮＩＸなどの話もあるが，ＵＮＩＸはソフトウェア開発用としてのオペレーティング・システムであり，そのようなアプローチは疑問である。

しかし，ＵＮＩＸは組込用オペレーティング・システムを開発する強力な支援システムである。

日本の技術者にとって日本語処理は必須の機能であり，特にソフトウェア開発は文書作成であり，ＵＮＩＸは強力な文書処理機能を提供することを考えると，日本語対応ＵＮＩＸが望まれる。

日本のいくつかのメーカも日本語処理を載せているが，オフィス・オートメーション対応の日本語処理とは異なるソフトウェア技術者のための日本語処理についての機能拡充が望まれる。

ソフトウェア技術の向上はハードウェア技術のもたらした大きな恵みを十分に利用し，大胆な進展を図らなければならない。そのためにはソフトウェア技術者がソフトウェア技術者自身の活躍し易い環境作りをまず進めなければならない。

第4章　トピックス

第4章　トピックス

4.1　ワンチップマイコン用コンパイラCL／1

CL／1は松下電器が開発したワンチップマイコンのソフトウェア開発用高位言語でC言語に類似した構造化言語で，割込み，入出力ポートの処理などハードウェアの制御に適した文法を持ちアセンブラレベルの細かな記述が可能である。

このコンパイラによるオブジェクトプログラムはアセンブラと比較して10％程度の増加になる。

CL／1の特徴は次の通りである。

① C言語類似の構造化記述が可能な文法構造を持ち，マクロ記述が可能である。

② 論理・算術演算等制御に適した29種の演算機能を持っている。

③ 割込み，入出力，タイマ／カウンタ，ROMテーブル読出し等ハードウェアに即したプログラム記述ができる。

④ インラインアセンブル機能により，アセンブルプログラムをソースプログラムに挿入できる。

⑤ アセンブラで書かれたプログラムとのリンクが可能である。

⑥ チップ専用のデバッグボードと組合せて，リアルタイム・シンボリックデバッグ機能を実現している。

⑦ ソースプログラムとコンパイラ出力のアセンブラプログラムを並列にリスト出力できる。

現在，このコンパイラが対象としているワンチップマイコンは松下製4ビットワンチップマイコンMN1500シリーズとデュアル8ビットワンチップマイコンMN1890シリーズである。図4－1はCL／1のシステム構成であり，表4－1はプログラムの概要である。以下，MN1500シリーズ用コンパイラCL／1（MN1500）の文法を略述する。

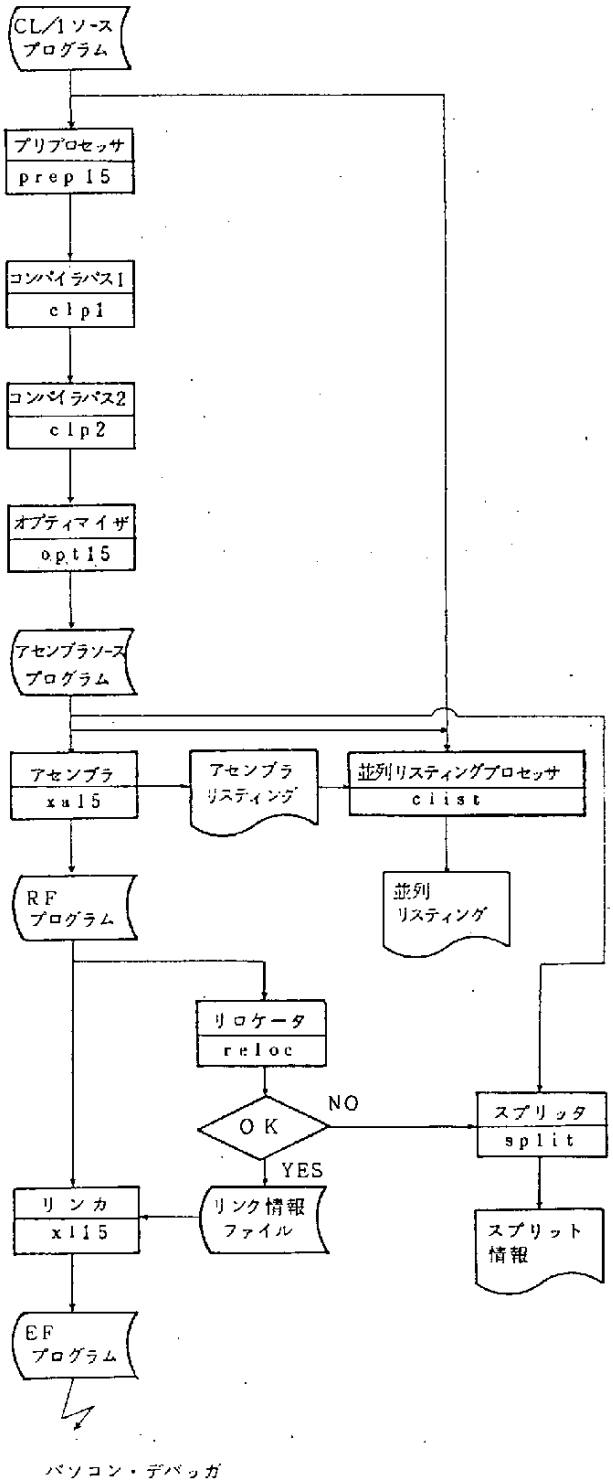


図4－1　CL／1 システムの構成

表４－１　ＣＬ／１システム構成プログラムの概要

構成プログラム	概　　要
プリプロセッサ	マクロ定義やソースの組み込み，条件付コンパイルを行う。
コンパイラ	ＣＬ／１ソースを入力とし，アセンブラソースを出力するコンパイラ本体。
オプティマイザ	コンパイラ本体で最適化できなかった部分の最適化を行う。
アセンブラ	アセンブラソースを入力とし，相対形式オブジェクトプログラムＲＦを生成する。
並列リスティングプロセッサ	ＣＬ／１ソースとアセンブラリストを並置して出力し，デバッグしやすくする。
リロケータ	ＲＦを入力とし，ページ境界を配慮して配置可能かどうかを決定する。 決定不可能な場合，スプリッタに制御を渡す。
スプリッタ	アセンブラソースファイルの分割情報を出力する。
リンカ	複数のＲＦプログラムを入力して実行可能なオブジェクトプログラムＥＦを生成する。

ＣＬ／１プログラムの単位はram 宣言，ポート宣言，割込み関数宣言，関数定義を複数個この順に並べた構成をとる。

(1)　ram 宣言

ram 宣言は，ram 変数名にram ラベルで指定されるＲＡＭ の絶対アドレスと占有するサイズ（ニブル単位）を与える。ram 変数名の各ビット（０～３）にビット名前を与えることが可能である。

この宣言中ではram ラベルとして（ram 名前±定数式）を記述す

ることにより，RAMアドレスの相対表記が可能である。また同一のRAMアドレスに複数の相異なるram変数名，ビット名前を与えることが可能である。

(2) ポート宣言

ポート宣言は，入出力ポートにポート名前とプログラム中に使用される状態を割当てる。同一ポートに複数のポート名前を割当てることも可能である。本宣言で割当てた状態以外の使用をプログラム中に記述するとコンパイル時に警告される。

(3) 関数宣言

関数宣言は，外部定義関数，バンク1に配置される関数，値を返す関数について宣言を行う。

extern …… 外部定義関数である宣言

bank …… バンク1に配置される関数であることの宣言

bank 宣言がなされていないバンク0に配置されるものとみなす。

(4) 割込み宣言

割込み宣言はリスタート関数宣言，割込み関数宣言とから成る。この宣言はROM空間の先頭に配置されるモジュール中に記述する必要がある。リスタート関数宣言はCPUリセット後最初に実行される関数名を記す。割込み関数宣言は割込み関数名を記す。関数名 sinq，irq，tcirq，sbirq は予約語である。

(5) 関数定義

関数定義は関数本体の実行文を記述する。関数本体として記述できるのは文のみである。

文の種類は条件文，while 文，do 文，switch 文，break 文，continue 文，return 文,goto文，コマンド文，テーブル文，インラインアセンブラ文，ラベル宣言文，関数コール文がある。

このうちコマンド文はMN1500シリーズのハードウェア固有の命

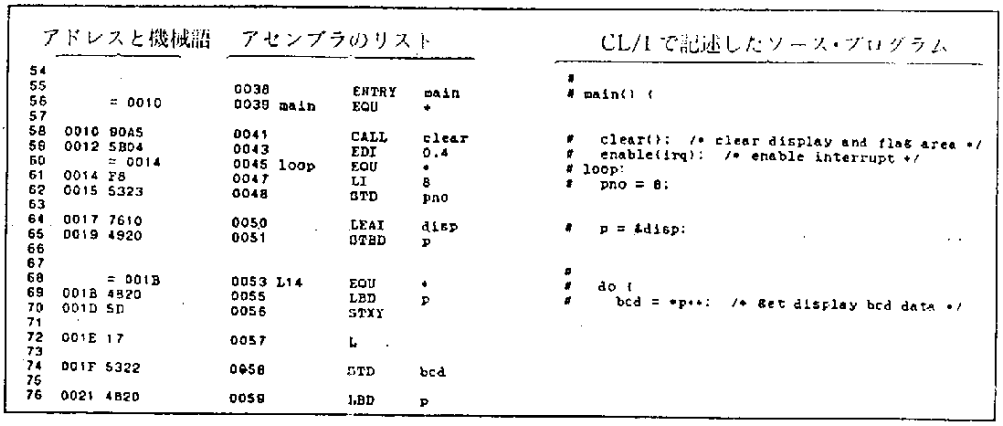
令を記述する文で命令セットに対応している。

表４－２にＣＬ／１の言語仕様をまとめて記す。

また図４－２はプログラムリストの一例である。

表４－２　ＣＬ／１の主な言語仕様

データ型		1～4語長（1語＝4ビット）の整数型 (0～65535)，ビット型（1ビット）
演算子	算術	+, −
	シフト	≫, ≪
	論理	&, ^, |, &&, ||
	代入	=, +=, −=, &=, ^=, |=, ≫=, ≪=
	単項	++, −−, &, *, !, ～
	比較	>, <, >=, <=, ==, !=
制御構造		if 文 while 文，do while 文 switch 文 go to 文，計算型 go to 文 関数呼び出し
その他		割り込みの禁止/解除 アセンブラ文の混在記述 入出力ポートへのアクセス レジスタ，フラグの参照，演算



アドレスと機械語　アセンブラのリスト　CL/1で記述したソース・プログラム

```
54
55                          0038          ENTRY   main          #
56          = 0010          0039 main     EQU     *             # main() {
57
58  0010 90A5               0041          CALL    clear         #   clear();  /* clear display and flag area */
59  0012 5B04               0043          EDI     0,4           #   enable(irq);  /* enable interrupt */
60          = 0014          0045 loop     EQU     *             # loop:
61  0014 F8                 0047          LI      8             #   pno = 8;
62  0015 5323               0048          STD     pno
63
64  0017 7610               0050          LEAI    disp          #   p = &disp;
65  0019 4920               0051          STBD    p
66
67                                                            #
68          = 001B          0053 L14      EQU     *             #   do {
69  001B 4820               0055          LBD     p             #     bcd = *p++;  /* Get display bcd data */
70  001D 5D                 0056          STXY
71
72  001E 17                 0057          L
73
74  001F 5322               0058          STD     bcd
75
76  0021 4820               0059          LBD     p
```

図４－２　ＣＬ／１のプログラムリストの一部

4.2　インテリジェント電磁流量計変換器〝ＮＮＸ〟

マイコンをプロセス制御分野の現場に応用した例について紹介する。

流量の測定は温度・圧力の測定とならんでプロセス工業ではもっとも重要なものである。電磁流量計は，オリフィスによる流量測定，差圧変換による流量測定などに比較して，その測定にさいして管路をしぼらないので，固形物を含んだ流体や高粘度の流体の測定にも障害はなく，また腐食性液体の測定も可能とする利点があり，流体物の比重や粘度の変化に関係せず正確な体積流量の測定ができるという長所を兼ねている所から，注目されていたが，電気的な取扱い（電磁汚れによるゼロシフト）が解決出来なかった。山武ハネウエルでは，矩形波励磁方式による電磁流量計を１９７４年に開発以来，こうした問題の解決をし，高精度，高信頼な流量計をプロセス工業へ提供可能としている。

この電磁流量計に対する運転前の流量スパン設定，出力パルスのスケーリングや機器のトラブル要因の解析などのために変換器の必要性がある。従来，この種の変換器は単一的で，機能的に不足であるのみならず，情報処理機器との接続において貧弱であった。

以下紹介するＮＮＸは，従来の機能不足を補うと共に，時代の要求するデータ管理の自動化，ＦＡに対応できるものと思う。

(1)　ＮＮＸのシステムにおける位置

図４－３は，電磁流量計，ＮＮＸ，ＭＣＩ，その他の計装機器で構成されたシステム図を示す。

流量の測定のために，ＮＮＸより流量計へ励磁電流を流す。流量計は磁界を発生し，電導性流体の移動により起電力を発生する。その電位はＮＮＸで変換され，記録計，調節計，カウンタなどの入力情報となる。調節計は流量の設定値に対するアルゴリズムの演算結果をバルブ出力として管を流れる流量のコントロールをする。

他方，複数のＮＮＸから出力されるデータは，ＭＣＩでマルチプレック

スされ，ＲＳ２３２Ｃで規定された接続方法でパーソナルコンピュータまたは汎用機器でデータ処理ができる。

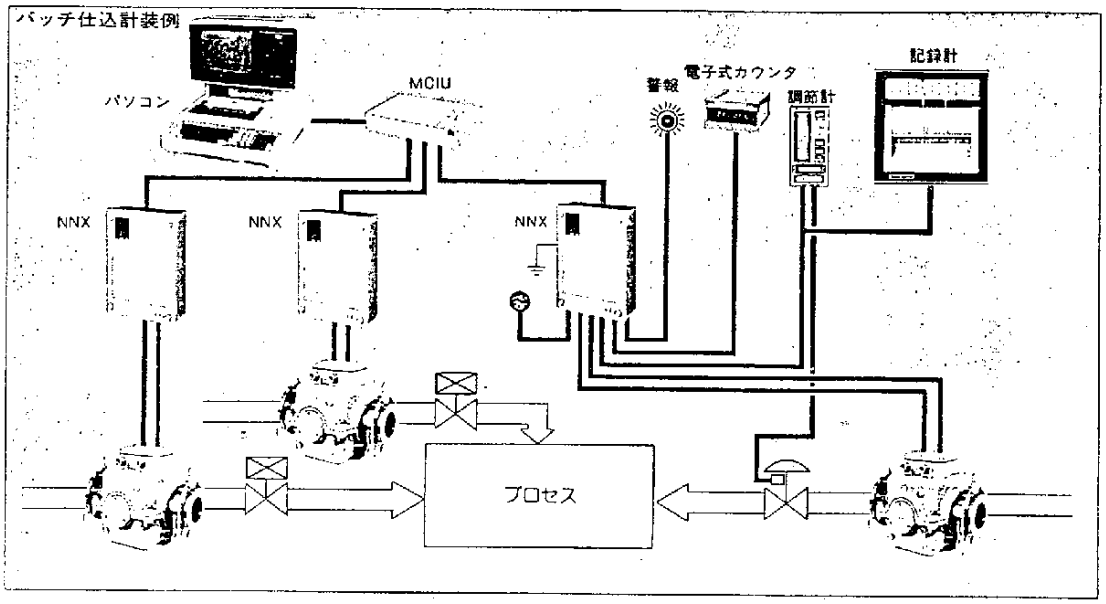


図４－３　ＮＮＸシステム

(2)　ハードウェア構成

写真４－１がＮＮＸの外観である。

図４－４に内部のハードウェア構成を示す。

マイクロプロセッサは８０系のＣＭＯＳタイプを使っている。また設定データの保持をするために不揮発性ＲＡＭ（ＮＶＭ）を使用している。

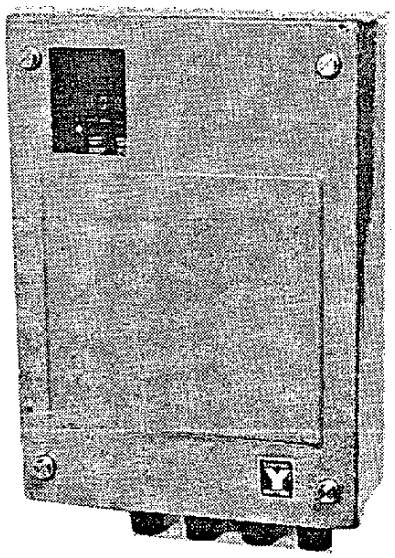


写真４－１　マイコン変換器〝ＮＮＫ〟

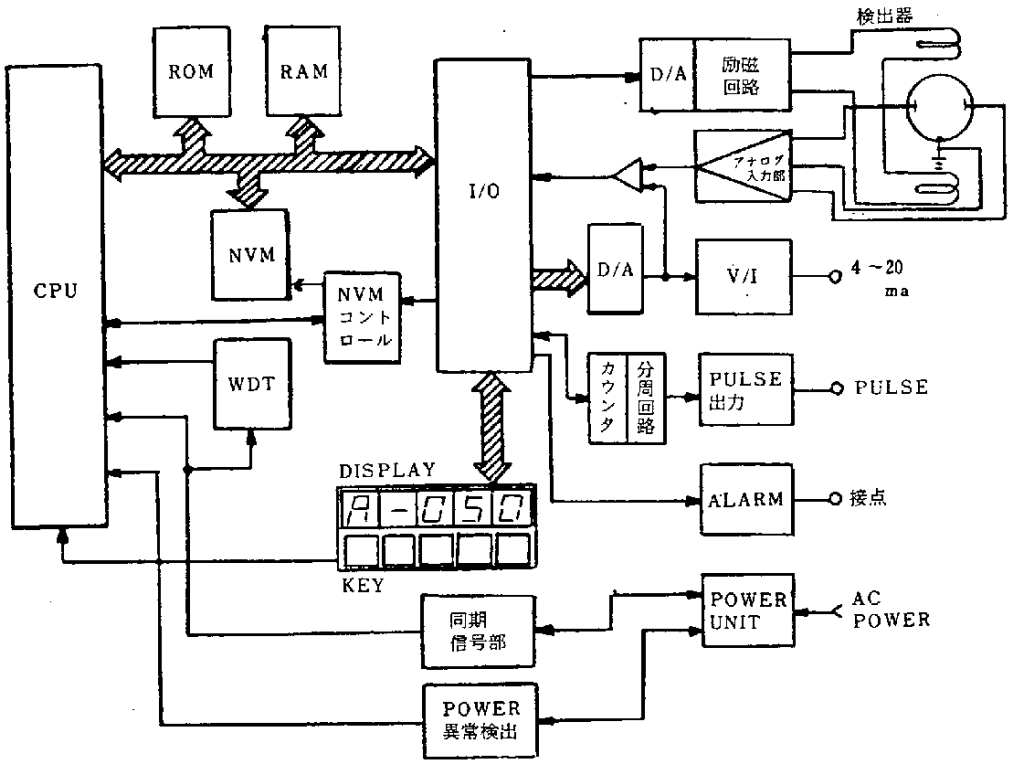


図４－４　ハードウェア構成

(3) 機　　能

次にＮＮＸの機能における特徴的な点について説明する。

(a) 表示と操作

流量計の測定した流量スパンの％表示のみでなく，工業単位（m^3／H，ℓ／min，m／sec　）の表示と，その積算値がＬＥＤ５桁で表示ができる。

これら表示の変更はパネル上にあるガイドに従って容易に設定ができる。

マイコンを導入することによりマン・マシン機能，工業値変換，積算機能が従来に比べ格段に向上している。

(b) 診　　断

診断には，ＮＮＸの自己診断と，電磁流量計を含めたシステム診断に分けられる。

自己診断には，ＲＯＭ，ＲＡＭ，ＮＶＭ，Ｄ／Ａなどの自己のハードウェアの診断を実行する。

システムの診断は，検出器，変換器，配線の状態が正しいかどうかのテストをする。これにはＮＮＸでは，次のような工夫をしている。

測定中に励磁電流の方向を逆転して，変化する前の出力値と比較して，入力された値が正しく逆転しているかどうかを調べる。システムとして正常であれば，流量信号は励磁電流に比例するので，配線または検出器側に不具合があれば，励磁電流の逆転時に，逆転前の値と一致しないことになる。

このシステムチェックは複数回連続して実行され，一致しない場合にはエラーとして表示される。

(c) 信号処理

電磁流量計では，電極に気泡や電解質がぶつかったり，電流ノイズが入った場合に異常な値を出力することがある。このようなスパイク的ノイズの除去をするために，直前のサンプリング値に比べて，流量スパン

の１０％以上の差がある場合には，このデータはノイズが重畳されたものとみなし除去する。

同様な目的のために，移動平均値データをとり正規データとしている。

また，検出器内に流体が存在しない場合の状態検出機能のエンプティ・チック機能が特色としてあげられる。従来器の場合はこの機能がないために，積算カウンタが不要にカウント・アップすることがあった。

ＮＮＸでは，図４－５に示すように，検出器が空になった状態では，出力が必ず＋側に振りきれるようにしてあり，この状態をＣＰＵで検知し，出力を０％にコントロールすると共に，パネル前面の表示部にその状態を示す。

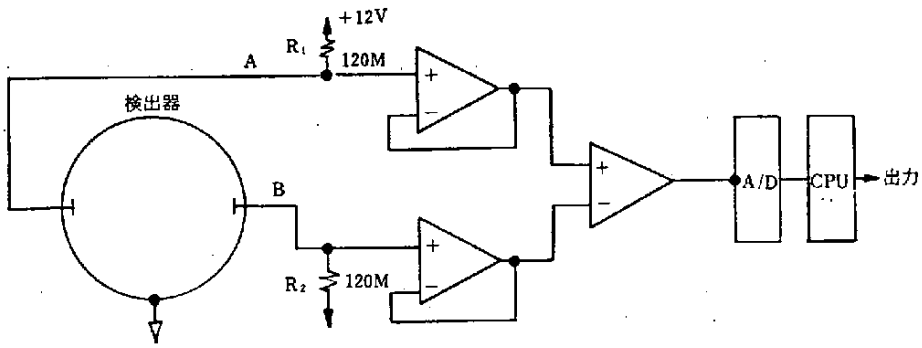


図４－５　エンプティ・チェック

(d)　コミュニケーション機能

電磁流量計を情報処理機に直接接続する概念は今までなかった。ＮＮＸではＭＣＩを経由してパーソナル・コンピュータなどのデータ処理装置と結び，流量の管理をするという発展が可能となった。これにより以

下のような機能ができる。

① リモート・セッティング…… 初期データの設定・変更がパソコンから行える。

② データ・ロギング…… 瞬時流量値，積算値，エラ表示がパソコン上に表示でき，これにより，日報・月報の作成ができる。

③ リモート・メンテナンス…… エラ・リセット，カウンタ・リセット，ゼロ点調整，電流出力調整（４～２０ｍＡ）などがパソコンからＮＮＸに対して行える。

④ 集中管理…… １台のＭＣＩに１６台までＮＮＸが接続できるので，現場に点在している電磁流量計の管理，メンテナンスが可能となった。

以上，紹介したように，マイコンを組込んだ変換器〝ＮＮＸ〟は，電磁流量計使用上のノウハウをソフト化することにより，高性能，多機能であり，かつ，操作上の煩雑さを取除き，信頼性の高い流量計を実現した。

参考文献

松本，吉備，〝マイコンを駆使したインテリジェント電磁流量計変換器ＮＮＸ〟計測技術’８３増刊号

4.3 組込みOS MTOS－68K

(1) 概　要

MTOS－68Kは，米国IPI社が開発したリアルタイムOSであるMTOSファミリーの1つであり，モトローラ社MC68000上で動作する〝リアルタイム・マルチタスキング・マルチプロセッサ・オペレーティング・システム〟である。

MTOS－68Kを使用することにより，新製品開発の時間と経費を節約し，リスクの低減，信頼性の向上，保守コストの低減も計れる。

オーバーヘッドが低く，早いスループットレートをもたらすマシンサイクル割り込みも可能であり，リアルタイムOSとして定期的・非定期的にアベイラブルとなるリソースへの完全かつ迅速なアクセスを，プライオリティ・スケジューリングにより管理され，応用分野として，①データ収集，②プロセス制御，③生産工程制御，④通信制御等，幅広い応用製品への組み込みが行われている。

(2) MTOSサービス

リアルタイムOSは，複数の動作を同時に処理し障害なしで機能を共有することによってコンピュータの性能を向上させる。すなわちMTOS－68Kは，複数のタスクのどのタスクであろうとサービスおよびファンクションを提供することにより，個々のタスクを最も効率的にサポートする。

MTOSサービスは，図4－6に示す様に広範囲に渡るカテゴリに分類され，プライオリティ，イベント・フラグ，メール・ボックス，セマフォ，メモリ・プール，ビット・ファンクション，ポーズおよびウェート，タイム＆デイト等の機能を有する。

コーディネーション・オプションに関しては，表4－3に示す様に各ササービスに対しタスク間の同期を取ることを可能にしている。イベント・フラグは，グローバル・イベント・フラグ・グループとは対称的に各タスクに対しローカル・イベント・フラグ・グループを有し，サービス・リク

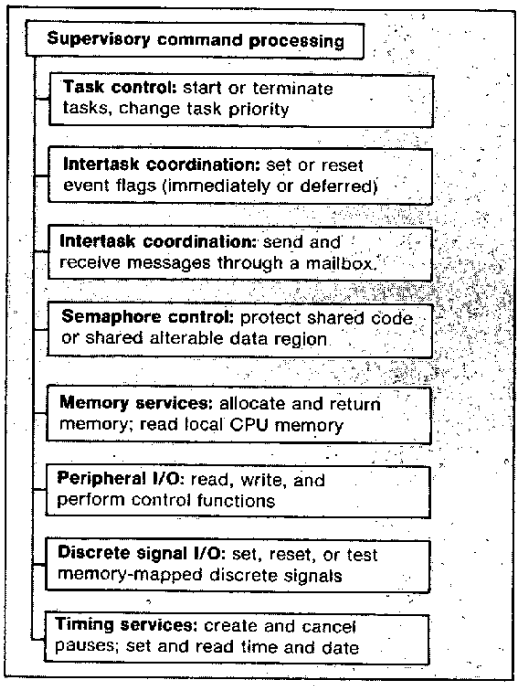


図4－6　All service functions provided by the MTOS－68K real-time operating system are independent of each other and need only a common supervisory interface to function.

表4－3　The main coordination modes of MTOS－68K

Service	Immediate coordination			Deferred coordination
	Continue	Wait unconditionally until completed	Wait for completion with time limit	Continue and set event flag when done
Allocate memory	No	Yes	Yes	Yes
Send/receive mailbox message	Yes	Yes	Yes	Yes
Peripheral I/O	Yes	Yes	Yes	Yes
Start task	Yes	Yes	Yes	Yes
Wait until semaphore is free	Yes	Yes	Yes	Yes

エストを発行したタスクに対しローカル・イベント・フラグのセットをＭＴＯＳが行う。セマフォはコモン・データやコード・セクションを排他的にロックし，他のタスクによるデータの更新を防ぐ。すなわち，ノン・リエントラント・サブプログラムは，バイナリ・セマフォを使用することにより保護領域とすることが可能である。

(3) コーディング

多くのアプリケーション・システムにおいて，システム・ソフトウェア開発の占める割合は，全プログラム開発に対して２０％以下ということはなく，５０％にもなることがある。ＭＴＯＳを使用することにより，そのストラクチャード環境とデバッキング・ツールをもたらすので時間の節減が可能である。

さらに，アプリケーション・ソフトウェア開発においてハイ・レベル言語を使用することにより効率的にアプリケーション・プログラムの開発を可能としている。（ＰＡＳＣＡＬおよびＣ）

ＭＴＯＳコードは，アセンブリ言語で開発されており標準モトローラ・ニモニックを使用し，モトローラのＭＡＣＲＯおよび条件アセンブリ機能を利用している。ソフトウェア・ソース・モジュールは，①ソース・プログラム・モジュール（基本コードおよびデータ・テーブルが入っている），②補助ライブラリ・ファイル（ＭＡＣＲＯ定義およびパラメータ値が入っている），③ＣＨＡＩＮファイル（リンク・インフォメーション）④補助モジュール（ＰＡＳＣＡＬおよびＣインターフェイス，ダイアグノスティック・プログラム）によって構成される。

アセンブリ言語によるコーディングは，ＭＴＯＳサービスを要求したいＳＶＣ　パラメータ・ブロックの先頭アドレスをプッシュして次に〝ＴＲＡＰ〟をかけることにより達成される。ＰＡＳＣＡＬまたはＣによる場合は，例えば２５０ｍｓ のＰＡＵＳＥを要求するためには function としてＰＡＵＳＥ（ＭＳ，２５０）となり容易に行える。

(4) マルチ・プロセッシング（ＭＰ）

ＭＴＯＳは，１台のプロセッサまたはコモン・メモリを共有している最高１６台までのマルチ・プロセッサ・システムにおいて実行することが可能である。ＭＰの組み込みにおいては，すべてのプロセッサは同等のものとみなされ，〝マスター〟および〝スレーブ〟の関係は存在しない。

すなわち，〝密結合マルチプロセッシング〟を構成しＭＴＯＳコードはグローバルに１つ持つことによってすべてのプロセッサで同時に実行される。各プロセッサは，ＭＴＯＳがタスク・ワークを割り当てる対象となるリソースとみなされ，通常プロセッサが使用可能時にレディ状態にあるタスクのうちで最も高いプライオリティを持つものが割り当てられる。

また，そのタスクがサービス・リクエストを要求し完了した後，同じプロセッサ上でリスタートするとは限らない。しかし，一定のタスクを一定のプロセッサに限定することは可能である。

密結合マルチ・プロセッシングの利点は，いくつかのタスクが実際に同時に実行されるのでスループットが向上することである。しかし，３台のプロセッサを使用しても３倍のスループットが得られることではない。

つまり，シェアド・メモリが１つなので，バス・アービットレーションが必要となる。この問題は，ＭＴＯＳコード（カーネル，ドライバ，システム・タスク）を，ローカル・メモリに置くことによってより迅速な応答が達成される。

各プロセッサの持つローカル・メモリのイメージは同一である必要があり，また各プロセッサにそれぞれ違った特殊な周辺装置を持つことも可能である。そのイニシャライズ・ルーチンは，システム・イニシャリゼーション時に自動的にコールされる。

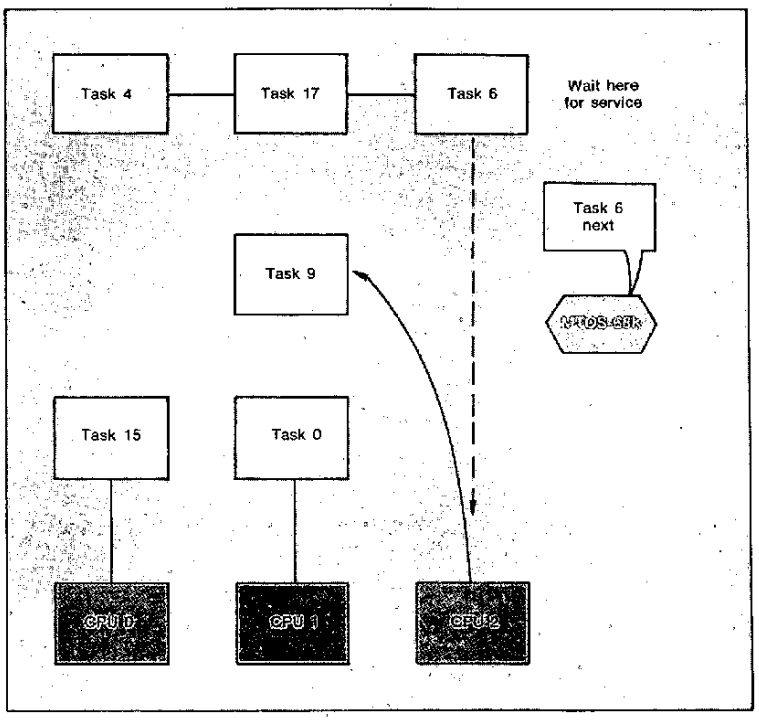


図４－７　In a multiprocessor system, tasks wait in a common queue for the next available CPU. When CPU 2 has completed task 9, the operating system assigns it task 6.

ＭＴＯＳマルチ・プロセッシング・システムにおいて重要なことは，ユーザ・タスク・コード内でマルチ・プロセッシングを全く意識する必要がないことである。すなわち，同じアプリケーション・プログラムはプロセッサの数が変化してもタスク・コードの変更は必要なく，補助ライブラリ・ファイルの変更で済むことである。シングル・プロセッサ・システムでシミュレーションを行いスピード・アップのためにマルチ・プロセッサ・システムにシステム・アップしても，何らトータル・プロセッシング・イメージは変化しないでスループットの向上を得ることができる。

(5)　ＭＴＯＳのインストール

ＭＴＯＳ－６８Ｋは，ＭＣ６８０００に基づいた任意のハードウェアで実行されるが，リアル・タイム・クロックのための定期的なインタラプトのみが必要とされる。

クロック・タイムはユーザ・オプションで，１～２５５ms までの範囲であり，この設定値がインターナル・クロックの最小単位となる。通常の値は，５～１０ms が選択される。

ＭＴＯＳでは，任意のハードウェアおよびアプリケーション・プログラムのために，ユーザは２つのパラメータ群を指定する必要がある。

１つのパラメータ群は主としてハードウェアに関連しており，一定のシステム構成に対し固定的である。他の１つのパラメータ群は主としてアプリケーションに関連しており，ユーザ・コードの開発中に変更される場合がある。いずれも，補助ライブラリ・ファイル中の２つのモジュール内で指定された一定のパラメータの値により，自動的に入れなければならないＭＴＯＳモジュールがジェネレートされる。この場合ＭＴＯＳの核のサイズは，5.8Ｋ～１0.5ＫByteと可変である。

デバッガ・サポートについては，ＭＴＯＳではデバッグ・モジュールとしてデバッガ・タスクがありアプリケーション・タスクと同等にリンクし，オン・ボード・モニタとしても機能する。モニタ機能として通常の機能の他にタスク・オリエンテッド・コマンドとして，タスク・レジスタのモディファイとディスプレイ，タスク・スタックの表示，タスク・ステータスの表示，タスクのホールトと実行，イベント・フラグおよびディスクリートの操作，セマフォの表示等の機能を持っている。

ユーザは，組み込みＯＳ・ＭＴＯＳ－６８Ｋを使用することによって高性能１６ビット・マイクロ・プロセッサＭＣ６８０００を載せた，より高いパフォーマンスを持ったリアルタイム・プロセッシング・コンピュータを作り上げることが可能となる。

参考文献

David L. Ripps：

① On Operating System.

② MTOS－68K User's Guide.

③ Multitasking OS manages a team of processors.

4.4 ソフトウェア・パフォーマンス・アナライザ

YHP社の64310Aソフトウェア・パフォーマンス・アナライザ（以下SPAと略す）は，YHP社の64000ロジック開発システムの開発ステーションに内蔵されたエミュレータとともに使用される1枚ボードのアナライザである。SPAは，その名のとおりソフトウェアの効率評価を行うためのもので，用途としては，ソフトウェアの最適化設計のために用いられる。

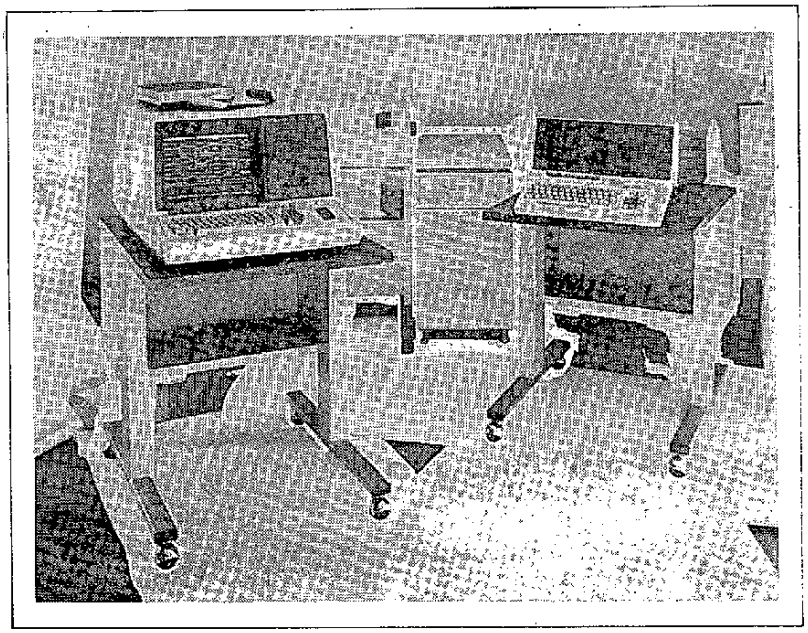


写真4－2　YHP社の64000ロジック開発システム

本節では，このＳＰＡが必要となった背景，ＳＰＡの機能および実際の応用例について説明する。

(1) 背　景

近年，マイクロプロセッサ応用機器は高機能化の一途をたどっている。そのためのハードウェアは，マイクロプロセッサおよび周辺チップの高機能化，高集積化が進んでいるが，ソフトウェアは大規模化，複雑化する一方である。そのため，一つのプログラムを複数のモジュールに分割し，複数の設計者により開発する方法が一般的である。また，製品のライフ・サイクルを考えた場合，機能追加，改良などのメンテナンスの比率も増え続け，最近では７０％近くにまでなってきている。

このような状況で要求どおりの性能のプログラムを作り，グレード・アップしていくために，ソフトウェアの最適化設計が重要なポイントになってきている。しかし，複数のモジュールが複雑に絡みあったプログラムの効率を上げるには，ソフトウェアの効率評価を行うため専用ツールが不可欠である。

以上のような背景により，ＳＰＡが開発されたわけである。

(2) ＳＰＡの機能

ＳＰＡは，基本的にはソフトウェアの各モジュールがどのように，どのくらい使用されているのか，実行時間はどのくらいかなどを測定するためのもので，エミュレータとともに使用される。エミュレータとともに使用するということは，ハードウェアが完成していない状態でのモジュール・テストの段階から，ハードウェア／ソフトウェア統合段階まで使用可能ということである。

ＳＰＡは，次の６種の測定モードを持つ。

(a) メモリ・アクティビティの測定

この測定モードは，各メモリ・レンジに対するアクセスの実行回数，実行時間を測定する。

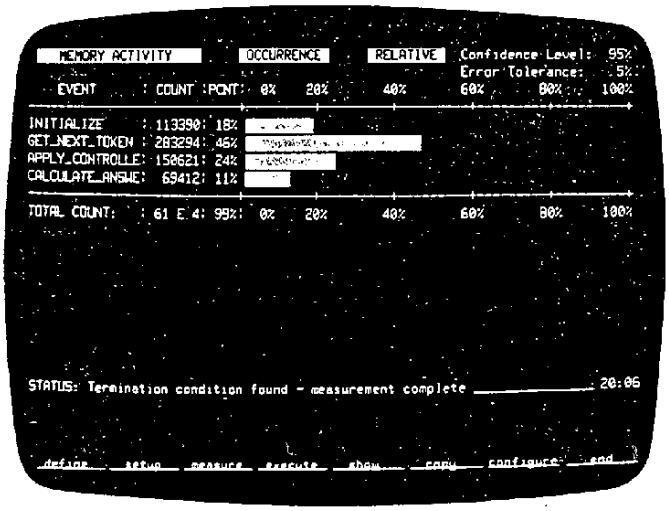


写真４－３　メモリ・アクティビティの測定例

つまり，各メモリがどのように使用されているかを測定することができるわけである。この時，マイクロプロセッサのステータス情報を使って，メモリ・アクセス情報のクォリファイが可能である。例えば，メモリ・ライト，Ｉ／０オペレーション，ＤＭＡ転送などを単独でも組み合わせてでも使用できる。

この機能は，例えばメモリ・エリアにＤＲＡＭとＳＲＡＭをどの様に配置すべきかの測定に使用できる。各メモリ・ブロックの使用頻度を測定し，使用頻度の高いメモリ・ブロックには高速のＳＲＡＭ，使用頻度の低いメモリ・ブロックには安価なＤＲＡＭを配置することにより，高速でしかも安価なシステムが実現できるわけである。

(b) プログラム・アクティビティの測定

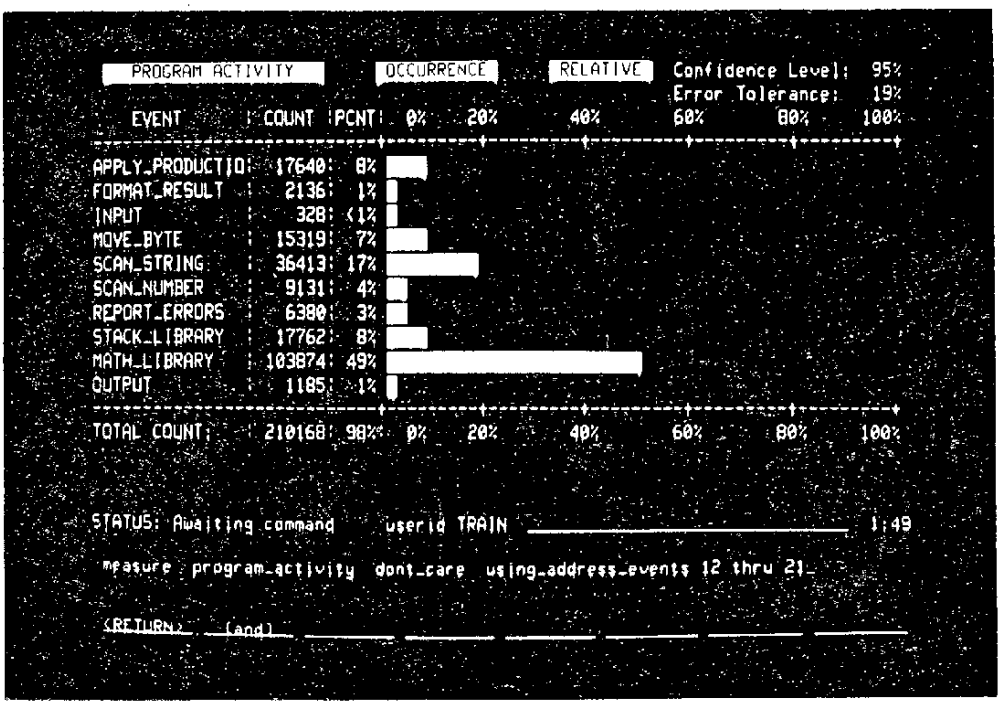


写真4－4　プログラム・アクティビティの測定例

この測定モードは，各プログラム・モジュールでの命令実行回数，実行時間を測定する。つまり，各プログラム・モジュールがどのように実行されているかを測定できるわけである。前述のメモリ・アクティビティ測定との根本的な違いは，データ領域のアクセスがデータ領域のアクティビティとしてカウントされずに，アクセスしたプログラム領域のアクティビティとしてカウントされる点である。

この機能は，例えばプログラムの実行速度を上げる場合，どのモジュールを改良すべきかの測定に使用できる。というのは，当然のことながら実行時間の短かいモジュールを改良してもほとんど効果は無く，実行時間の長いモジュールを改良すべきなのだが，実行時間の長さはプログラム・サイズには必ずしも比例しない。そこでこの機能により，各モジ

ュールの命令時行回数，もしくは実行時間を測定し，最も実行回数が多い，実行時間の長いモジュールを見つけ出し，改良すると良いわけである。

(c) モジュール・デュレーションの測定

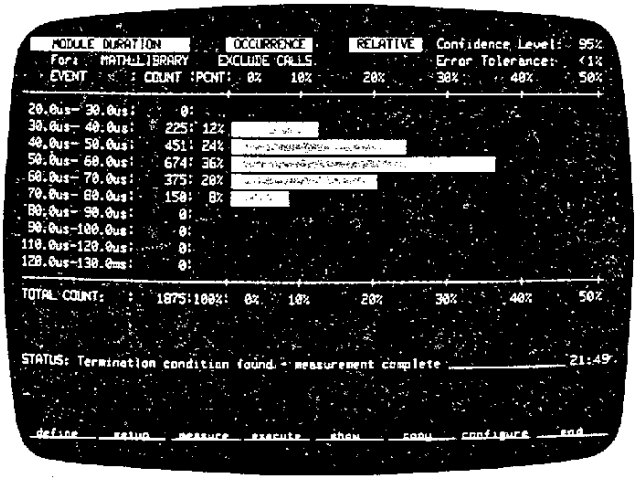


写真４－５　モジュール・デュレーションの測定例

この測定モードは，ある特定のプログラム・モジュールの実行時間を測定し，その時間分布を表示する。測定するモジュールから他のモジュールをコールしている場合や，インタラプトで他のモジュールへ脱け出ている場合には，他のモジュールの実行時間も含めてカウントするか（インクルード測定），含めないでカウントするか（エクスクルード測定）を選択可能である。

この機能は，例えば設計したプログラム・モジュールの実行時間が設計値に合っているかどうかの測定に使用できる。というのは，モジュールの実行時間というのは条件によって変化する場合が多い。その場合，１回測定しただけでは，モジュールの実行時間を把握できない。なるべく数多く，様々な条件で測定し，実行時間などのような分布になるかを把握するべきである。特に，インタラプトがランダムに発生しており，

実際のモジュールだけの実行時間が測定しにくい場合は，エクスクルード測定で正確な実行時間が測定可能である。

(d) モジュール・ユーセージの測定

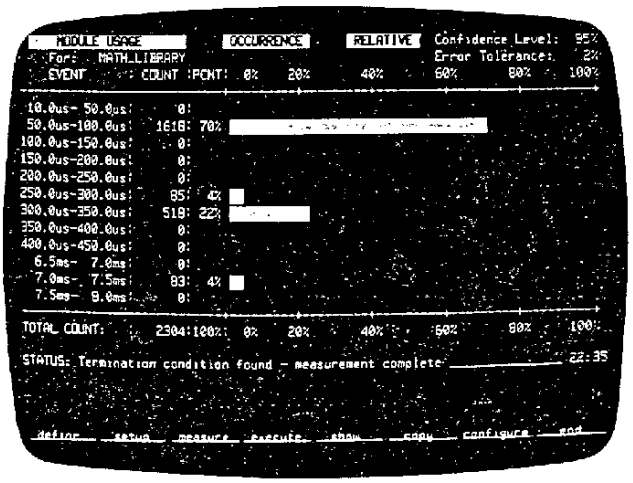


写真４－６　モジュール・ユーセージの測定例

この測定モードは，ある特定のプログラム・モジュールを脱け出た時から，再びもとのモジュールに戻るまでの時間を測定する。つまり，そのモジュールがどのくらいの時間間隔で呼ばれているかを測定できるわけである。

この機能は，例えばインターラプトがどのくらいの時間間隔で発生しているかの測定に使用できる。インターラプト処理ルーチンがどのくらいの頻度で呼ばれているかを測定し，その結果により，インターラプトが予期していたより短かい間隔で発生していないか，逆に長い間隔で発生していないかなどが判定可能である。

(e) インターモジュール・デュレーションの測定

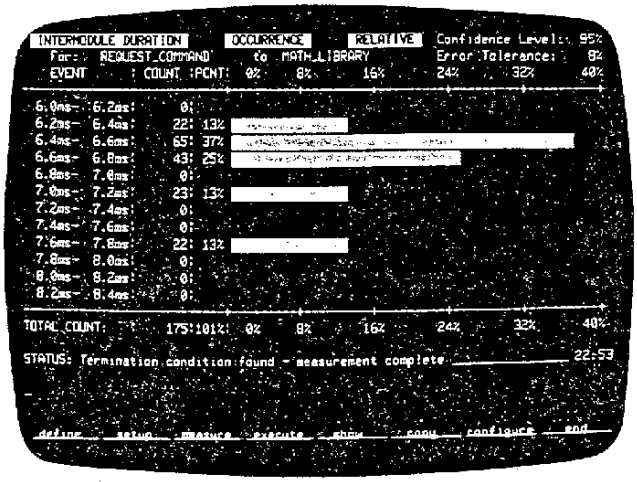


写真４－７　インターモジュール・デュレーションの測定例

この測定モードは，あるプログラム・モジュールを終了してから，別のモジュールに入るまでの時間を測定する。

この機能は，例えば次のような場合に使用できる。あるプログラムは，最初低速入出力装置から高速バッファへブロック転送するハードウェアを起動するルーチン（ＸＦＥＲルーチン）をコールする。次に，転送が終了するまでの時間，別のタスクを実行し，その後高速バッファからシステムのメモリへロードするルーチン（ＬＯＡＤルーチン）をコールする。その場合，ＸＦＥＲルーチンとＬＯＡＤルーチンの間の時間を測定し，ハードウェアによるブロック転送に十分な時間があるか，もしくは余裕がありすぎないかを確認することが可能である。

(f) インターモジュール・リンケージの測定

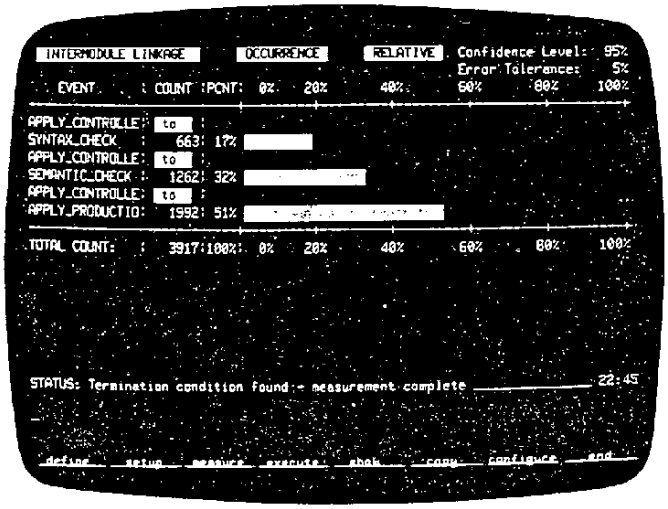


写真４－８　インターモジュール・リンケージの測定例

この測定モードは，ある特定のプログラム・モジュールからいくつかのモジュールを呼び出している場合に，各モジュールがどの程度呼ばれているかを測定する。つまり，各モジュール間の結びつきの程度が測定できるわけである。

この機能は，例えば起こるはずの無いリンケージが起きていないかどうかの測定に使用できる。起こる可能性のあるすべてのリンケージを測定し測定結果の合計が１００％になったら他のリンケージは起きていないと判断できるわけである。

ＳＰＡは，この６種の測定モードの結果をヒストグラムとデータ・リストの２種で表示する。さらに，測定結果に対し，統計的手法を用い，信頼率や誤差率を表示する。また，データ・リスト表示の場合は平均値，標準偏差も表示するため，より正確に実行状況が把握できるわけである。

(3) 実際の応用例

ＳＰＡのこれらの機能を実際に，どのように使用できるかを説明する。

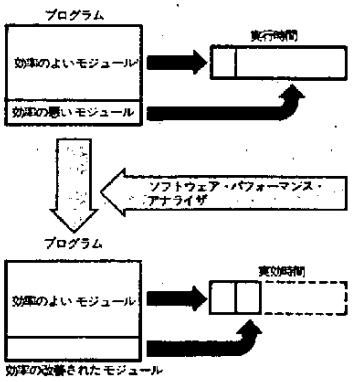


図４－７　ソフトウェア・パフォーマンス・アナライザによるプログラムの改善例

例として，数式を読み込み，計算するプログラムを作ったとする。この時，要求されていた速度より遅かった場合，このプログラムのどのモジュールを改良すべきであろうか。

一般的には，最も長いモジュールを考えるが，プログラムの長さと実行時間の長さは必ずしも比例しない。短かいモジュールでも，ループが多かったり，多数のモジュールから頻繁に呼び出されるために実行時間では最も長い場合はよくあることである。そこで，ＳＰＡのプログラム・アクティビティ測定により，各モジュールの実行時間の比率を測定したところ，最初は読み込んだ変数名によって変数テーブルから値を探し出すＳＣＡＮ－ＳＴＲＩＮＧルーチンが原因だろうと思われていたが，実際には値を計算するＭＡＴＨ－ＬＩＢＲＡＲＹルーチンが原因であった。

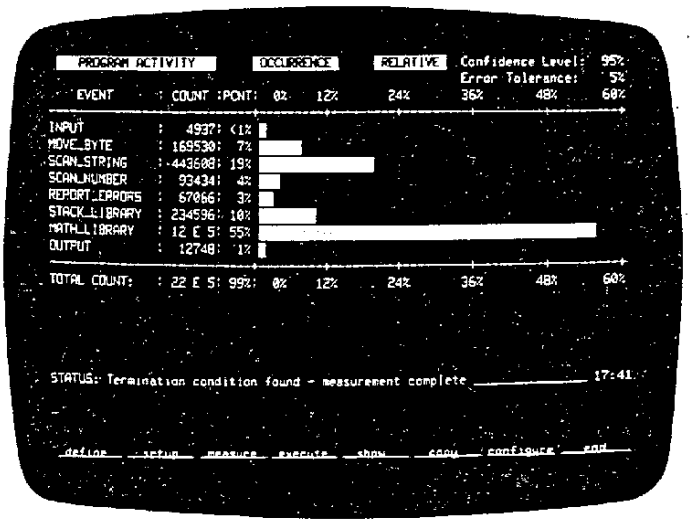


写真４－９　改善前の実行時間の比率

次にＳＰＡのモジュール・デュレーション測定によりＭＡＴＨ－ＬＩＢＲＡＲＹルーチンの実行時間を測定すると，仕様で示されていた実行時間よりはるかに長い事がわかる。

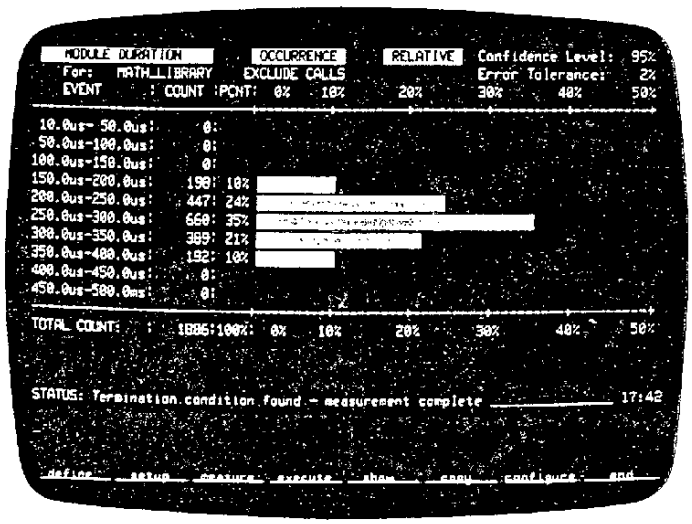


写真４－１０　改善前のＭＡＴＨ　ＬＩＢＲＡＲＹ
ルーチンの実行時間

そこでこのルーチンのアルゴリズムを調べ，ムダな部分を改良し，もう

一度実行時間を測定する。また，各モジュールの実行時間の比率も測定する。今度は，実行時間，各モジュールの比率とも予想したとおりになっていることであろう。

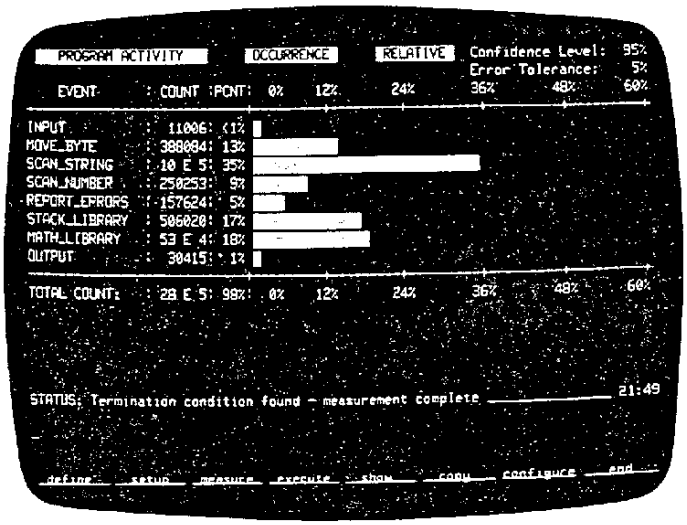


写真４－１１　改善後の実行時間の比率

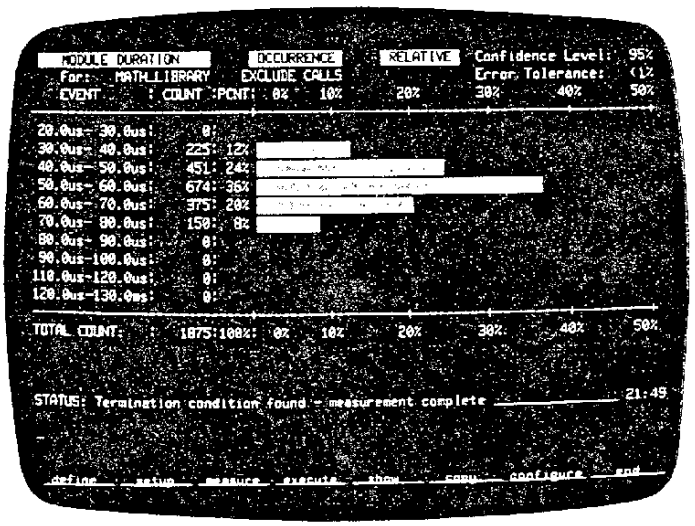


写真４－１２　改善後のＭＡＴＨ　ＬＩＢＲＡＲＹ
ルーチンの実行時間

この時，もっと速度を上げようとするなら，今度はＳＣＡＮ－ＳＴＲＩ

ＮＧルーチンを改良すべきである。というのは，改良されたプログラムでは，実行時間の比率が最大なのは，ＳＣＡＮ－ＳＴＲＩＮＧルーチンになっているからである。

ソフトウェアの大規模化，複雑化は今後も進む一方と予想される現在，ソフトウェアの最適化設計は，ソフトウェア・エンジニアにとって最も重要な課題である。その最適化設計を強力にバック・アップするＳＰＡは，今後のマイコン開発システムには必要不可決の機能である。現在このＳＰＡの機能を持つ開発システムは，ＹＨＰ社の６４０００ロジック開発システムのみであるが，今後開発されるマイコン開発システムは，すべてこのＳＰＡの機能を持つことは必至であろう。

昭和 59 年 3 月発行

発行所　財団法人 日本情報処理開発協会
東京都港区芝公園3丁目5番8号
機械振興会館内
Tel (434)8211(代表)

印刷所　株式会社 タケミ印刷
東京都千代田区神田司町2－16
Tel (254)5840

58-R010

原本（持出厳禁）

受付No.	
受付年月日	
作成課	