マイクロ・プロセサー

基　礎　編

昭和51年度

財団法人　日本情報処理開発協会

この資料は、昭和５０年度に日本小型自動車振興会から小型自動車競走法に基づく小型自動車等、機械工業振興資金の交付を受けて作成した資料の領布用増刷である。

＜講座の内容について＞

すでに二・三ある市販参考書が、どちらかといえば、やや総花的な知識・解説の展開を試みているため、かえって正しい理解へのための混乱を招くのではないかという恐れがあったため場合によっては、特定デバイス、特定方式、特定ソフトウェアを限定的にとりあげ、それだけに限って徹底的な解説を試みるという、大胆な編集方針をとっている。それゆえ「一を知れば、おのずと十を知りうる」の編集方針のもと、重要事項に関しては、一品種に解説の重点が偏ることも辞さず、まず理解する、次いでより知識を深めるという執筆方針を採用した。もちろん、大方の参考となる事項については、もれなく網羅するよう配慮したつもりである。

以上のような基本方針で、マイクロ・プロセッサと、これを主体とするマイクロ・コンピュータの初歩的な理解を得るため、「第１章・マイクロ・コンピュータの概要」次いでプロセッサ本体とそれに関連すをファミリイ・ハードウェアの詳細な理解を目的に「第２章・マイクロ・コンピュータのハードウェア」を、また「第３章・マイクロ・コンピュータのソフトウェアでは日本電気の製品UCOM8によるソフトウェアを、徹底的に解説することにした。これは、はじめに述べたとおり、１機種のソフトウェアの完全理解は、直ちに他機種への理解に通ずるという考えにもとずくものである。

一般に、マイクロ・プロセッサおよびマイクロ・コンピュータ部門での設計者、生産技術者には、ソフトウェアへの理解がとぼしいように見受けられる。また、世上、多くのソフトウェア関係の参考書も出版されているが、多くは汎用コンピュータ向きの高級言語解説書で、マイクロ・プロセッサ、マイクロ・コンピュータ・システムで直ちに必要となる機械語、記号語に関連した良書は少ない。ミニコン、マイクロ・プロセッサともに、デバイス購入後、はじめて詳細なソフトウェア解説書を手にすることが多い。これが、ある意味から、ここではUCOM8を中心に解説を進めたが、その知識に、わずかの差を追加することで、他機種のソフトウェアをも、容易にマスターできるで

あろうことを付言したい。

「第4章・マイクロ・コンピュータ・システムの設計と開発」更に「付表・マイクロ・プロセッサー覧」を付して読者の実際の便に資した。とりわけ、付表の各社一覧表の作成にあたっては、メーカー各社にアンケート調査をお願いしてデータを収集、整理したものである。メーカー各社のご協力を感謝するとともに、万遺漏のないよう努力したつもりであるが、あるいは意にそわぬ点が残るやもしれないことを恐れている。伏してご容赦をいただきたい。

最後に、この執筆グループが発足したのが50年夏。当初から企画案に参画され、グループ間の執筆調整、連絡事務、資料整備の幹事役の労をとられた応用システム研究の佐藤紘一氏が、本年の1月に、急な病いで不帰の客となられた。執筆者一同、謹んで哀悼の意を表するとともに、佐藤氏の遺志をつがれて、応用システム研究スタッフのなみなみならぬご協力をいただいたことを深く感謝する次第である。

昭和51年3月

編集　委員代表者　　安　田　寿　明

はじめに

＜刊行の目的＞

１９７１年末から１９７２年はじめにかけ出現した１石集積回路形のマイクロ・プロセッサは、いわゆるマイクロ・コンピュータ・システムの心臓部となるもので、その豊かな可能性のゆえに電子工業、機械工業だけでなく、社会的に多大のインパクトを与える新技術、新システム、新商品として各方面から多大の注目を浴びている。しかしながら、開発と発展の歴史が浅いことから、参考資料、参考文献のたぐいは、いまだ数少なく、マイクロ・プロセッサを理解し、日常の技術の場、生産活動の場、サービスの場に新知識を活用していこうと志ざす、意欲ある向きには、必ずしも満足すべき現状とはいえない。

そうしたなかで、読者対象を、半導体電子回路に、必ずしも深い知識を必要としない初歩的な人びとをも含み、場合によっては企業内新人教育の教材としても使えるよう、はいりやすく、かつ奥深い内容を盛り込むことにし、

昭和５０年度日本小型自動車振興協会の補助金の交付を受けて編集したものであります。

＜ 編 集 委 員 ＞

目　　次

1, マイクロ・コンピュータの概要

1.1 マイクロ・コンピュータの歴史

1.1.1 MOS 型素子の概要

マイクロ・コンピュータとは何か。この問いに対して、さまざまな定義がある。一般にコンピュータとは、カリキュレターと異なり、それ自体の単体としては用をなさないシステム・マシンであることはよく知られている。それと同じく、マイクロ・コンピュータも、通常のミニコンピュータ本体、あるいは汎用コンピュータの中央処理装置と機能的には似通った部分があるが、それ自体では所望の動作をしない。最低限の入出力装置を接続するなど、システム構成をとることによって、はじめて所望の動作をする。このレベル、つまり、マイクロ・コンピュータと他の周辺機器とによって形成するシステムを、ここでは、一応「マイクロ・システム」とする。

それでは「マイクロ・コンピュータ」そのものの構成が問題となる。ミニコン、汎用機の本体は、大きくわけて論理演等、制御、主記憶、データ・チャンネルの各部から構成されている。マイクロ・コンピュータの場合、これらの主要部分をＬＳＩ化、数個のＬＳＩを中心として形成された電子回路システム、これをマイクロ・コンピュータと呼ぶのが適切なようである。

いまひとつ「マイクロ・プロセッサ」と呼ばれるものがある。これは、最初に出現したマイクロ・コンピュータ素子のうち、最重要なものが「ワン・チップCPU」と銘うたれたことから、CPU機能を有するＬＳＩ、すなわち論理演算・制御部をＬＳＩ化したものを指すと考えられる。もっとも、現在は、これらの機能を１石集積回路ではなく、数個のチップに受け持たせてものも数多く出ているし、必ずしも「ワン・チップＣＰＵ」即「マイクロ・プロセッサ」という概念にはつながらない。

現在では、便宜的に、論理演算・制御機能と、それに主としてレジスタ的用途の小容量内部記憶回路を有するものなどをも含めて「マイクロ・プロセッサ」という呼び名が慣用されている。これも、将来、集積度の極度に高い、つまり大容量内部記憶回路をも内蔵する「マイクロ・プロセッサ」が出現したとき、いまの概念が、そのまま通用するかどうか、疑わしいところである。ともあれ、本章に限らず、ここでは、一応、現在の慣用に従って説明を進めていくことにする。

その本格的なマイクロ・プロセッサが、はじめて世に出たのは、よく知られているとおり１９７１年のことである。これが可能となった基本的技術の背景に、MOS 技術の急速な発達があったことは、忘れてはならない。

現在、最もよく普及しているトランジスタは、PNP, NPN 型の、いわゆるバイポーラ・トランジスタである。これに対しユニポーラ、つまり半導体表面の導電率を表面に垂直な電界で制御する電界効果型トランジスタは、トランジスタそのものの出現まもなくの１９３０年ごろに、すでに提案されていたが、なかなか実用化するに至らなかった。

ほぼ３０年を経て高精度で表面準位の少いシリコンが製造できるようになり、まず接合型電界効果トランジスタが出現、次いで制御電極をシリコン酸化被膜で形成したMOS 型トランジスタが実用化するに至ったのである。

第１－１図は、MOS 型電界効果トランジスタの構造を図示したものである。同図でわかるとおり、N型領域の基板に、P型領域のドレインとソース電極を形成し、その中間域に酸化被膜で絶縁された制御電極（ゲート）をおく。ソース電極を接地の状態にし、ドレイン電極に負電源を接続し、ゲート電圧をＯＶとすると、ゲート・基板間に電位差はなく、ソース・ドレイン間には、わずかなもれ電流しか流れない。

ところが、ゲート電圧が負の方向に増加するにつれ、ゲート直下の表

面に、ゲート電圧による正の誘導電荷、すなわちホールが集中し、ホール密度が表面電子密度を越えたとき、ドレイン・ソース間の基板表面はN型からP型に反転し、P領域で形成されるソースとドレイン間にP型チャンネルの電流通路が開け、ドレイン電流と呼ばれる電流が流れることになる。PチャンネルMOS といわれるのは、このためである。

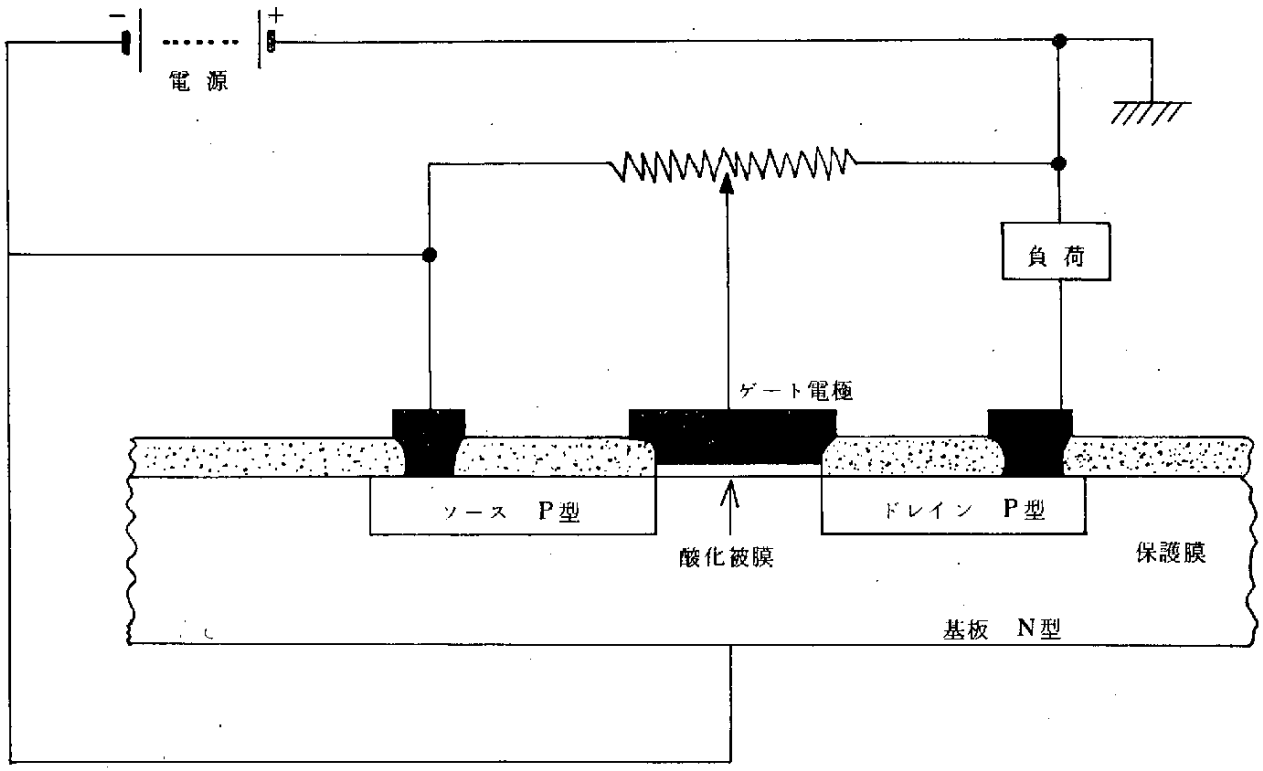
また、ソース、ドレイン電極部をそれぞれN型領域とし、基板をP型領域とすると、正負逆方向の電源で、PチャンネルMOS とまったく逆の動作をすることがわかろう。このタイプのものをNチャンネルMOSといっているわけである。

MOS 型トランジスタでは、制御入力となるゲートがシリコン酸化被膜で絶縁され、バイポーラ・トランジスタの電流制御入力と異なり電圧制御である。そのため入力インピーダンスは高い。またゲート絶縁膜の絶縁性もきわめて高いが、極小薄膜であるため、かんたんに破壊電圧にまで到達する。いわゆるFET と呼ばれるMOS トランジスタやMOS・LSI では、これの保護用に、ダイオードを内臓形成するようにしたものが、最近は一般的であるが、過大電位に対しては、必ずしも安心できない。MOS 型デバイスの購入時、とくに作業時の静電気対策について注意を喚起しているのは、このためである。

1.1.2 LSI 発達の経過

さて、以上のようにMOSトランジスタは、入力インピーダンスが高く、結果的に入力電力は小さくてすむ。これは、また、論理回路のように多段接続が多いとき、全体としての消費電力を少なくすることができる。これは、集積回路化に好都合な条件となる。

一方、ハイ・インピーダンスの入力であるため高速応答も可能なはずであるが、構造上、入力ゲートとソース・ドレイン間に容量が形成され、この入力容量のため、全体としての速度は、バイポーラよりも低速になるという欠点もある。しかし、トランジスタ製造工程では、集積回路化



第１－１図　　MOS型トランジスタの構造と
動作原理（Pチャンネルの場合）

には、MOS型が圧倒的に有利で、現在のところ、SSI(小規模集積回路)やMSI(中規模集積回路)はバイポーラ型で、LSIとなるとMOS型というのがIC製品の主流となっている。

こうしたMOS型の特長を最初にいかしたのは、電子卓上型計算機の分野であった。電子式卓上計算機は、当初、バイポーラ・トランジスタで製造されていたが、バイポーラICの採用で飛躍的なコストダウンが実現したことは、よく知られているとおりである。

そのなかでも、比較的に大規模集積化が実現しやすいPチャンネルMOS・LSIに着目され、この分野での製造技術の改良が、ここ数年の電子式卓上計算機の驚異的な低価格競争の原動力になったといっても過言ではない。同じ製造技術を利用して、まず大容量シフトレジスタからはじまって、大容量ランダム・アクセス・メモリ(RAM)へというコンピュータ用LSIへ進んでいったのである。

いまのところ、半導体メモリと呼ばれる記憶装置は、このMOS・LSIが主流である。これは、バイポーラや、その他の製法の半導体製品にくらべ、高密度集積化が、現在、最もたやすく実現できる製造法であることに起因する。半導体メモリは、マイクロ・コンピュータにとっても重要欠くべからざる構成部品であるので、MOS型メモリについて、以下に若干の解説をしておこう。

MOS型を中心とするLSIメモリは、当初、前述のとおりシフトレジスタの大容量化からはじまり、その後、RAM, PROM, ROMなどへと発展していった。ROMは、リード・オンリ・メモリの略で字義どおり、読み出し専用のメモリである。半導体メーカーでの製造工程で記憶情報は固定化されてしまう。このタイプで最も早く普及したのは、CRTキャラクタ・ターミナル用の文字信号発生用LSI(キャラクタ・ゼネレータ)であった。現在は、マイクロ・コンピュータ用の固定プログラム記憶、いわゆるファームウェアに主用途が拡大しつつある。

PROM は、プログラマブルROMの略。メーカー・ベースではなくユーザの作業現場で記憶内容の消去・再書き込みができるのが特徴である。ROM, PROM ともに、その機能と製法からいって不揮発性であることはいうまでもない。

不揮発性というのは、記憶装置の電源を遮断し、再投入したとき、遮断直前の記憶情報を保存しているかどうかいう。コア・メモリ、磁気ドラムなどの磁性記録では、そうした心配がなく、再投入時でも遮断直前の記憶情報の内容を再現できる。これに対し、次のRAM・LSI では、おおむね電源遮断時に、その記憶内容は失われてしまうことになる。ごく最近、そうした心配のない不揮発性RAM も開発されている。

RAM・LSI は、マイクロ・コンピュータを、ミニコンのようにプログラム・オリエンテッドの汎用目的に使用する場合、主記憶装置として重要な役割をはたす。これにはスタティック型とランダム型とがある。どちらがよいかという優劣比較が、システム設計現場で議論の対象となることが多い。しかし、それぞれに一長一短があって、どちらかに軍配をあげることは危険である。適材適所の法則は、ここでもいきてくる。

スタティックRAM は、フリップ・フロップ回路をMOS・FET で構成し集積化したもので、ダイナミックRAM は、FET 回路の寄生容量に蓄えられる電荷で、論理表現をする。回路的にはダイナミック型の方が、より簡単で、その結果、大容量の記憶LSI が、わりあいたやすく製造できる。ところが寄生容量中の電荷は、FET のもれ電流で、時間経過とともに失われていく。このため記憶情報保持の必要から、一定間隔で再書き込みをする必要がある。これをリフレッシュと呼んでいる。

ダイナミックRAM は、こうしたことから周辺制御回路がやや複雑になり、使いにくいという定評があった。しかし周辺専用のLSI が開発されたり、あるいはRAM 自体に制御部をもなるだけ収容するなどのくふうが進みはじめ、スタティック型の実装のしやすさにせまりつつある。

設計・製作コスト、オペレーション時のコストパフォーマンス、これらを総合評価して、ダイナミックかスタティックかを論じるべきであろう。

1.1.3　P—MOS・CPU の誕生

MOS・LSIによる半導体メモリの大容量化ということは、とりもなおさず集積度の高度化ということになるのであるが、いまひとつの利用法である10進演算回路を主体とした電子式卓上計算機用のLSI も、より高密度の集積化と、より低価格の実現をめざして製造技術が進んでいった。このような環境条件を背景として、当然の帰結として誕生したのが、いわゆる「ワン・チップCPU 」である。

この分野で最初に名乗りをあげたものが、当然のことながら、最初のマイクロ・プロセッサ開発者となったわけだが、これは、よく知られているとおり、インテル社のI4004である。4ビット並列処理のCPUと若干のスクラッチ・パッド・メモリを持ち、発表当時は画期的な新製品であった。

スクラッチ・パッド・メモリというのは、スクラッチ・パッドすなわち石板という意味で、昔、学校教育でノートがわりにメモ書きに使った石板を意味する。小数個のレジスタだけでなく、スタック・メモリなどのほか、多数個のレジスタ類をも構成できる高速一時記憶のことを、一般にスクラッチ・パッド・メモリと呼んでいるわけである。

I4004は、当初、高級プログラマブル電子式卓上計算機のCPU相当部として開発されたもので、10進演算に便利なよう、4ビット構成であった。同じ4ビット並列処理でPチャンネルMOS のものが、のちに、ノースアメリカン・ロックウエル社などからも発表され、この系統のものは、高級電卓ばかりでなく、短時日の間に、電子式キャッシュ・・レジスタに、さかんに用いられるようになっている。

インテル社は、I4004の発表後、まもなく、それに追いうちをかけるように、同じPチャンネルMOS で8ビット並列処理のI8008

を発表した。この８ビットというのには、非常に大きな意味がある。これは、汎用コンピュータの分野で、ＩＢＭ 社が、命令語および記憶の基本単位としてとり入れた概念で８ビット＝１バイトとしている。このバイトという単位を本格的に採用したのは、ＩＢＭ 社のシステム３６０以後であるが、それ以来、汎用コンピュータ分野では、記憶の基本単位としている。またデータ伝送の分野でも８ビット／字のＡＳＩＩ符号、ＩＳＯ符号が早くから制定されており、８ビッド処理マイクロ・プロセッサの出現で、マイクロ・プロセッサの応用面における汎用性が飛躍的に増大したといえる。

1.1.4　ＮＭＯＳ・ＣＰＵへ

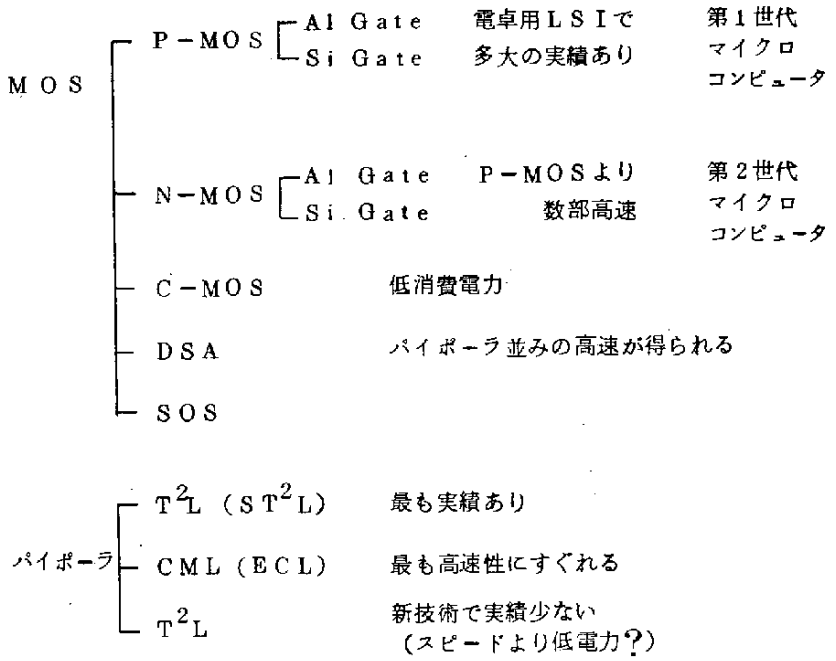
以上のように、かずかずの特徴を持っPMOS・CPU であったが、実際の応用面では、多少、不便が生じることになった。というのはマイクロ・プロセッサの出現以前に、ミニコンなどの普及から、ＴＴＬ(トランジスタ・トランジスタ・ロジック) レベルの論理回路、あるいはディジタル制御装置などが、世間一般に、かなり広く行き渡っていたのである。ＴＴＬ の場合、一般にNPN 接合のトランジスタの電流飽和特性を利用しているので、接地電位をＯＶとしたとき、コレクタ電位は＋５Ｖを供給することになる。

これに対し、ＰチャンネルMOS では、ソース接地論理回路でドレイン供給電位は負電源からとなり、論理レベルは、正負まったく逆になるわけである。このため、ＰチャンネルMOSによるマイクロ・プロセッサやメモリ、その他のファミリイ・デバイスによるマイクロ・コンピュータと、ＴＴＬ 論理回路やシステムを接続する場合、必らずインタフエース回路が必要となってくる。ここで相互に電圧レベルを変換せねばならないわけである。

そうしたことから、ＴＴＬ と共用で使える、いわゆるＴＴＬコンパティブルはＮチャンネルMOS によるマイクロ・プロセッサや、その関連素

子の出現が望まれたのは当然であるといえよう。

ところが、NチャンネルMOS の場合、製法上のあい路から、Pチャンネルほどには集積回路化がたやすくはなく。製造コストも高くつくという欠点があった。そうした難点も、ようやく克服され、１９７４年ごろから、NチャンネルMOS 系のマイクロ・CPU が登場するようになり、TTL コンパティビリティと、P・MOSよりは速度もやや早いという特性がいかされ、大いに歓迎されるようになったわけである。インテル社のＩ８０８０、Ｉ４０４０、モトローラ社のＭ６８００、フェアチャイルド社のＦ－８、国産の日本電気UCOM—8 などがそれである。開発後、大いに歓迎されたことから、価格的にも、PチャンネルMOS の低価性に迫ってくるようになっている。



第１－２図　デバイス製造技術による分類

1.1.5. 他素子ＬＳＩの概要

さて、以上のようにＰチャンネルMOS からはじまったマイクロ・プロセッサの歴史を、それ以後に出現したものを加えてまとめてみると図１－２のようになる。ＰチャンネルMOS・ＬＳＩによるマイクロ・コンピュータを第１世代マイクロ・コンピュータとし、それ以後を第２世代と呼んでいることもあるが、これについては、のちに説明しよう。

ＮチャンネルMOS 以後、市場に実際に出まわりはじめ、いま現在、最も注目を浴びているのは、Ｃ—MOS・ＬＳＩによるマイクロ・プロセッサである。Ｃ—MOSとはコンプリメンタリ（相補性）MOS の略語である。回路的には、ＰチャンネルMOS とＮチャンネルMOS という、正反対の性格を持つ素子を対称的に接続し、論理回路を形成したものである。

このようにすると、無信号入力のとき、アイドリング電流が流れないという特性が得られる。つまり信号入力に関係のない電流エネルギイを抑えることができ、結果的に低消費電力を実現することになる。乾電池で動作するマイクロ・コンピュータを製作することも、Ｃ—MOS・ＬＳＩによれば、さほど困難なことではない。

しかし、同一シリコン基板上に、物性的には正反対の性質を有する素子領域を形成していかなければならず、集積化の技術には、かなり高度なものを必要とする。１９７５年になって、ようやく本格的に市場に出るようになったが、ＬＳＩ を数多く使う必要のあることが多いメモリなどの場合、とりわけ低電力消費であることが圧倒的に有利な条件となりつつある。

また一般的に、MOS 型ＬＳＩ は、ゲート入力容量によって、ある程度以上の速度は、なかなか得られないものだが、高速性を主目的にＤＳＡ，SOS 製法によるMOS・ＬＳＩ の開発もさかんである。しかし、技術的にまだ未確立の点もあり、市場に出現してくるのは、まだかなり先

のことになるもようである。

このようなMOS 型にくらべ、大規模集積化、すなわちＬＳＩ 化が、バイポーラ型では、なかなかにむずかしいといわれていた。しかしＳＳＩ（小規模集積回路）やＭＳＩ(中規模集回路）の分野では、圧倒的にバイポーラ型のＴＴＬ (T^2L) やＳT^2L （シヨットキイ・バリャードＴＴＬ) が用いられていたわけで、その有利さをいかし、バイポーラ型のマイクロ・コンピュータも、いくつか出現している。ＴＴＬ では、テキサス・インスッルメント社の９０００シリーズがあり、また１９７６年中にエミッタ・カップルド・ロジック（ＥＣＬ)によるマイクロ・プロセッサが市場に出てくる見込みである。ＥＣＬ の場合、１ゲートあたりの応答特性は１０００メガ・ヘルツ以上という超超高速性も可能だが、電力消費量が大きく発熱対策に困ることが予想される。汎用目的のマイクロ・プロセッサとしてではなく、とくに高速性が要求されるシステムに向いているといえよう。

1.2. マイクロ・コンピュータの将来

1.2.1 電子回路との置換実装

マイクロ・プロセッサの開発意図は、当初、前述のとおり、電卓用の１０進演算が主目的であったが、８ビット・タイプの汎用目的型や、ビット・スライス型と呼ばれるビット構成を任意にとり得るタイプのものが出現するにおよんで、計側・制御の分野から、汎用報報処理の分野へと、応用範囲は急速に拡大しつつある。

なかでも注目されるのは、制御用電子回路の分野で、一般にこの分野では組み立て工数も多く、大量生産がきかないなどの理由で、量産効果によるコスト低下が望み薄なところであった。しかし、マイクロ・プロセッサの出現で、ROM によるファームウェアと組み合わせて、任意のシステム方式を、ソフトウェアで解決できるようになった。

その結果、製造工程の主要部分を占めるアセングル・ラインの工数を飛躍的に減少できるほか、ある程度の標準作業化も可能となり、生産コストを大幅に低下できる可能性が強くなった。これは、わが国のように、高度知識化産業構造に重点目標をおき、各産業製品のより高度化と技術的付加価値の増大をめざしている国民的経済背景のなかで、きわめて有利に働く環境条件であるといえよう。

1.2.2　ミニコンなどとの比較

マイクロ・プロセッサにも、ある程度の大容量の主メモリを付加し、さらに入出力装置、大容量の外部記憶ファイルなどを付加すると、機能的には、ミニコンあるいは汎用コンピュータと同等の能力を保有することも可能である。それでは、マイクロ・プロセッサによる、この種のコンピュータ・システム構成がコスト的に有利かといえば、必ずしもそうではない。それは、この種のシステムの中心となる大容量記憶ファイルなどの周辺機器コストが、CPU コストにくらべ、比較にならないほど高価であるためである。

逆に、ミニコン、汎用コンピュータは、その特性をいかし、マイクロ・コンピュータ・システムでは実施してあまり効果が望めないデータ・ベース志向のトータル・システム化をめざしていくことになる。とりわけ、ここ二・三年、いわゆる高位ミニコン、スーパー・ミニコンと呼ばれる機種がいくつか出現してきており、これは明らかに、ミニコン、汎用コンピュータ分野でのデータ・ベース志向を、より強く打ち出した現象としてとらえることができる。このような傾向が出現の波及効果であるということもできよう。

2, マイクロコンピュータのハードウェア

2.1　マイクロコンピュータの概要

普通、計算機は、基本的に制御部、算術論理演算部（ALU)、記憶装置から出来ており、その基本動作は、制御部が記憶装置から命令を取出し、デコードし解釈して、それに従って実行を行なうということをくり返す。

実行は、記憶装置に有るデータとALU 内のレジスタに有るデータをALUで演算し、結果をALU 内のレジスタ又は記憶装置へ格納することを行なう。

この制御部とALU を1又はいくつかのLSI で実現したものがマイクロプロセッサであり、このマイクロプロセッサを中央処理装置にしたコンピュータがマイクロコンピュータと呼ばれる。

この章では、LSI 化がコンピュータやプロセッサに与える影響等をふまえながら、マイクロコンピュータを整理してみる。

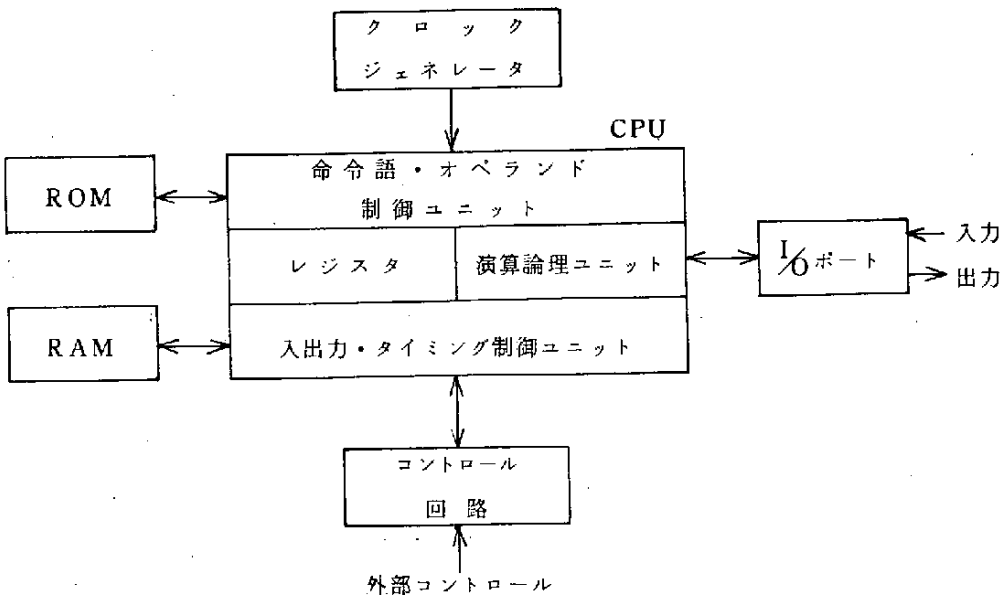
マイクロコンピュータの構成は、一般に第2－1図に示す様にマイクロプロセッサ (CPU) RAM (Random Access Memory)、ROM (Read Only Memory) 及び入出力装置から成立っている。以下にこれらの構成要素の概要を記す。

2.1.1　マイクロプロセッサ

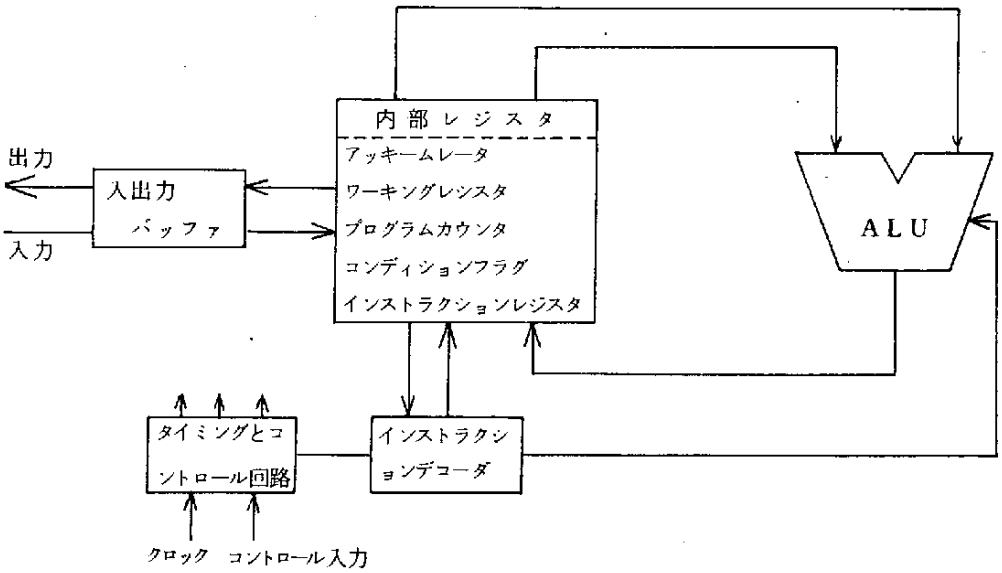
マイクロプロセッサの構成は、機種によってかなり差があるが、一般的には、第2－2図に示すように、命令語およびオペランド処理用制御ユニット、内部レジスタ、ALU、入出力バッファ、タイミング制御ユニットから構成されている。内部レジスタには、データ入出力用や演算用のアキュームレータ、ポインタ・アドレス用のインデックス・レジスタ、命令をロードするインストラクション・レジスタ、ワーキングレジスタ、ステータスレジスタ、プログラムカウンタ等がある。ALU による演算の種類は、加減算、AND、OR等がある。マイクロプロセッサの入出力

ポートは、ＬＳＩ 化によるピン数の制限から入力用と出力用は、共通で時分割により使いわける。又、これがアドレス転送用にも用いられるものもある。次にこれらの構成要素の相互接続は、機能の融通性及び構造の標準化といった観点からバス構造をとるのが普通である。バスの本数としては、データの流れからして３本が最適であるが、チップ面積を節約するため単一バスを時分割で用いることが多い。第２－３図は、代表的な３種類のバス構成を示している。

制御部の制御方式は、インストラクション・レジスタの命令をチップ上のランダムロジックでデコードする方法が効率の点からは最も良いが、命令セットは変更出来ない。これに対し、マイクロプログラム制御と呼ばれる方式がある。これは、マイクロ命令と呼ばれるハードウェアに非常に密着した命令のみを決めておき、機械命令は、このサブルーチンとして実現するものである。この方式は、論理構造を標準化出来、しかも複雑な制御ユニットを単純化出来る点でＬＳＩ 化に適している。



第２－１図　　マイクロコンピュータの一般的な構成



第２－２図　　マイクロプロセッサの基本構成

2.1.2　マイクロコンピュータのメモリ（RAM，ROM）

(1)　ＲＯＭ

読出し専用メモリで、電源を切っても情報は失なわれない。マイクロコンピュータはシステム等に組込まれ、きまった処理に用いられることが多い。又前述のマイクロプログラム等は一度作られると変更される事が少ない。その様なプログラムは、ROM に入れて用いるのが便利である。

(2)　ＲＡＭ

読み書き可能なＩＣメモリをRAM と呼び従来のコアメモリのかわりに用いる。

ミニコンでは、メモリの最小構成が決まっているが、マイクロコンピ

ュータではこれらのメモリを必要量だけ付け、システムを構成出来る点も特徴の一つと言える。

2.1.3　入出力ポート

入出力ポートは、ラッチとデコーダから構成される。マイクロプロセッサから外部への出力データは、ポートが種々の目的に時分割で用いられるため、スタティックな出力が得られる様に、入出力ポート内のラッチにたくわえる。又出力する入出力装置の機番指定は、メモリへのアクセスと同様アドレス指定で行なうためセレクトのためのデコーダが必要となる。入力に対してもスタティック入力に対してはデータに対してはラッチが必要となる。

2.2　マイクロプロセッサのアーキテクチャ

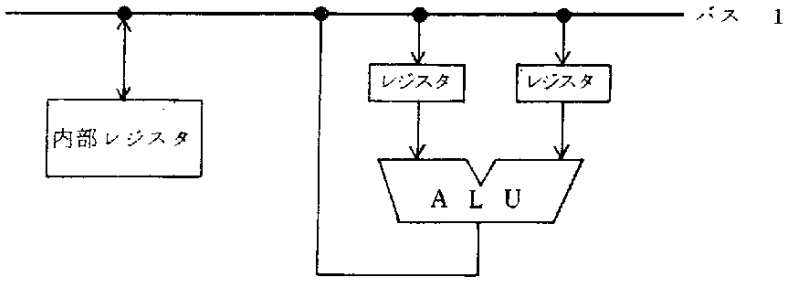
今日、多種多様なマイクロプロセッサが市販され、あるいはされようとしている。そして大部分のアプリケーションは、時間さえ気にしなければ、どのマイクロプロセッサでも処理することが出来る。しかし、マイクロプロセッサは、実時間系の内で用いられることが多く時間が重要な場合が多いし、各マイクロプロセッサには個性があり、マイクロプロセッサの選択がシステムの性能に大きな影響を与える場合が少なくない。この節では、自分のアプリケーションに最も適したマイクロプロセッサを選択するために知っておかなければならないマイクロプロセッサーのアーキテクチャについて述べる。

2.2.1　アプリケーションとアーキテクチャ

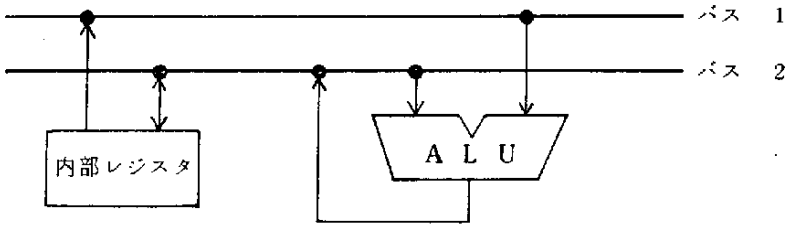
マイクロプロセッサの選択は通常次の3点から検討される。

① ソフトウェア設計：プログラマが直面するマイクロプロセッサの特徴。

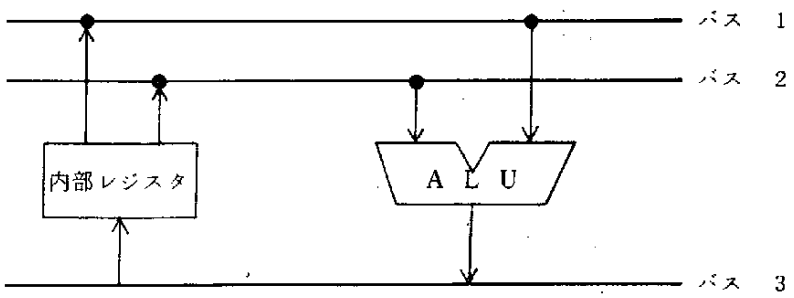
② ハードウェア設計：マイクロプロセッサが動作する環境に関する性質。



(a) シングル・バス構成



(b) 2 バス構成



(c) 3 バス構成

第2－3図　　代表的なバス構成

③　システム設計：ソフトウェアとハードウェアのインターフェース等。

第２－１表は、以上の３点の具体的な項目を示したもので、システム設計者は、この表をもとに、選択を行なえばよい。

この項では、これらのうちアーキテクチャに関係するものを取り上げ説明を加える。但し解説する項目の順序は、表と関係がない。

第２－１表　　マイクロプロセッサの選択基準

- ソフトウェア設計
  - ワードサイズ（データ／命令）
  - アドレス容量
  - レジスタ間の演算時間
  - レジスタの数と種類
  - スタックサイズ
  - アドレス方式
- ハードウェア設計
  - クロック（速度／フェーズ）
  - 必要電圧
  - 消費電力
  - コンパティビリティ
  - パッケージサイズ
  - ファミリーパーツの完備
- システム設計
  - ・特　　徴
    - 割込み
    - プロセッサのチップ数
    - マイクロプログラム能力
    - ＤＭＡ能力
    - ＢＣＤ算術

・メーカの公約

ソフトウェアサポート

ドキュメンテーション

アプリケーションノート

デザインエイド

・製産寿命

セカンドソース

標準化

・価格と可用性

(1) レジスタ

アプリケーションプログラムの基本操作はデータをレジスタに取込み、それに演算を行ない結果をメモリや周辺装置に送るというものである。故にレジスタはマイクロプロセッサの評価にとって非常に重要な項目である。レジスタには、次のような種類がある。

① 算術、論理演算用レジスタ

② インデクスレジスタ

③ 入出力やメモリアドレス用レジスタ

④ スタック操作レジスタ

⑤ 汎用レジスタ

⑥ フラブ (ステータス) レジスタ

算術、論理演算には、演算用データや演算結果を格能するレジスタが必要であり、普通は、1又は数台のアキュームレータや汎用レジスタがこの目的に使用される。もしアキュームレータが1個であれば、他のデータは全て主記憶に格能する必要があり、他のデータがレジスタにある場合に比べアクセスタイムが増大し処理速度が落ちる。一方レジスタ数を増やすと命令の中のレジスタを指定するフイールドが大きくなり他の機能を圧迫する。例えば、7個のレジスタを指定するに

は3ビット、15個のレジスタを指定するには4ビットのフィールドが必要となる。以上の理由から、命令のビット数が限られている場合には、これらのトレードオフとなる。

インデックスレジスタは、アドレッシングに用いられるレジスタである。具体的には、命令のアドレスフィールドとインデックスレジスタの内容を加えて実効アドレスを作る。一般にマイクロプロセッサの命令のアドレスフィールドは小さく、インデックスレジスタは主記憶全体をアドレス出来る程大きいので、これにより十分大きな実効アドレスを作り出すことができる。ただし、インデックスレジスタを用いる場合には、命令の中にインデックレジスタを指定するビットが必要となる。現在市販されているマイクロプロセッサでは、インデックスレジスタを持たないもの、専用レジスタを持つもの、汎用レジスタをアキュームレータとインデックスレジスタに使いわけているもの等がある。

アプリケーションプログラムは、入出力用レジスタやメモリアドレス用レジスタを使う。入出力用として、専用レジスタを備えたものもあり、算術論理演算用と共用されるものもある。メモリアドレス用としては、レジスタが用いられるものや、命令のアドレスフィールドでアドレスレユーザには見えないものもある。

フラグレジスタには、演算結果（正、負、ゼロ、オーバフロー等）、ステータス、マイクロプロセッサ外からの信号、プログラムスイッチ等が割当てられる。

スタックレジスタについては、命令セットの項で述べる。

以上がマイクロプロセッサで使われる主なレジスタの説明であるが、各マイクロプロセッサは、レジスタ数、レジスタの割当て当を様々なトレードオフで決定している。故にシステム設計者は、これらを検討し自分のシステムに合ったものを選ぶ必要がある。第2－4図は、代

表的な3つのレジスタ構成例を示している。

(2) アドレス方式

マイクロプロセッサのアドレッシングには、一般の計算と異なる2つの点がある。第1はマイクロプロセッサの命令長が短かいため、この内にあるアドレスフィールド（オペランド部）も短かくなり、大きなメモリ空間をアドレスするのが難しいという点である。例えば、64K語のメモリを全てアドレスするには16ビットのアドレスポインタが必要となる。第2の点は、マイクロプロセッサでは、アクセスする命令やデータが、現在実行している命令の入っているアドレスの近くに有ることが多いという性質である。第1の点に対する解決策としては、大きなメモリをアドレスする長いアドレスを、命令の内の短いアドレスと、命令外の情報から作り上げる方法がある。このようにして作られるアドレスは実効アドレスと呼ばれる。第2の性質は、命令にメモリ全体をアドレス出来るほど大きなアドレスフィールドを取る必要が無いことを意味し、マイクロプロセッサのようなマイクロ命令長が限られた環境では、利用価値の有る性質である。

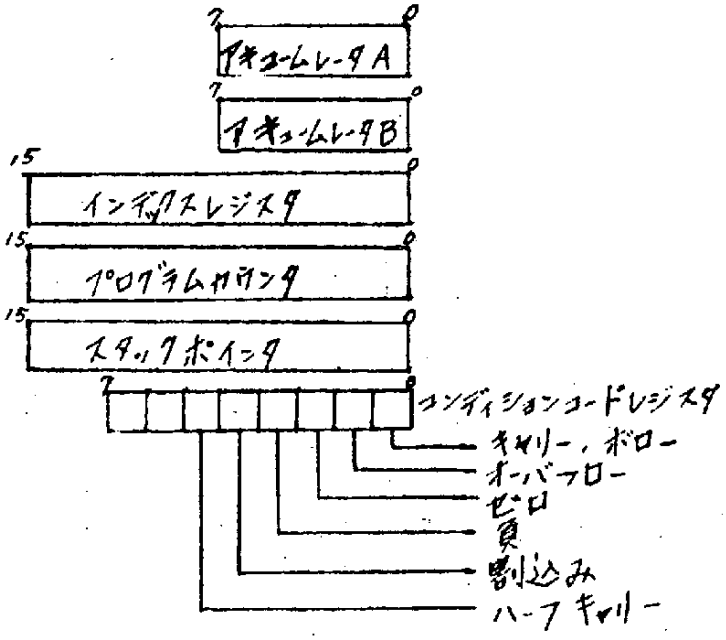
このような事情から、マイクロプロセッサでは、命令が短かいことをカバーするためいろいろなアドレス方式が用いられている。以下にその主なものをあげる。

(イ) 直接アドレス（第2－5図(1)）

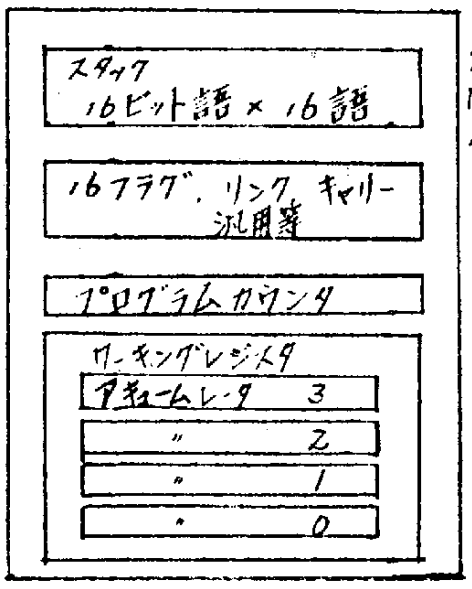
最も一般的なもので、メインメモリ全体をアドレス出来るオペランドアドレス部を持つ命令のオペランド部がそのままアドレスとなる。

(ロ) イミーディエイトアドレス（第2－5図(2)）

プログラムカウンタが、参照されるデータのアドレスとなる。言いかえると命令のオペランド部が参照されるデータとなるものである。この命令には、JUMP、CALL の他LOAD、IMMEDIATE

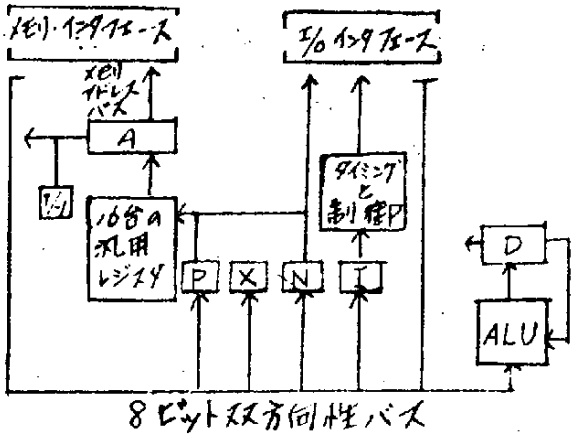


(a) 専用レジスタ構成



(b) 多目的レジスタ構成

P：現在プログラムカウンタとして使われるレジスタへのポインタ
X：メモリアドレスの入っているレジスタへのポインタ
N：汎用レジスタとデバイスのI/Oコントロールへのポインタ
A：外部アドレスレジスタ



(c) 現在の使用目的を定義するポイントを持った汎用レジスタ

第2－4図　　レジスタ構成例

（アキュームレータにオペラント部を移す）等がある。

(ハ) 間接アドレス（第２－５図(3)）

命令のオペランド部で指定されるメモリの番地には、参照されるデータが入っているのではなく、参照されるデータの入っているアドレスが格納されているものをいう。このアドレス方式は、プログラミングを容易にするほかに、参照されるデータのアドレスに１語を使えるので、命令のオペランド部より大きなアドレス空間を取れる利点がある。しかし欠点としては、この命令を実行するためには、２度の１モリアクセスが必要で、実行時間は遅くなる。

又機種によっては、上のようにして間接的にフェッチされたアドレスに、さらに間接指定ビットを置き多重間接を行なえるものもあるが、それが非常に有用であるかは、疑問である。

(ニ) インデックスアドレス（第２－５図(4)）

参照データのアドレスは、インデックスレジスタの内容と、命令のオペランド部を加算して得られる。

(ホ) 相対アドレス（第２－５図(5)）

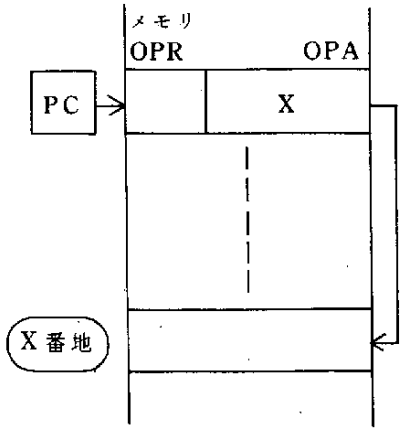
インデックスアドレス方式のインデックスレジスタをプログラムカウンタに置換えたもの。即ち命令のオペランド部と命令の入っているアドレス値の和が参照データのアドレスとなる。

(ヘ) ページアドレス

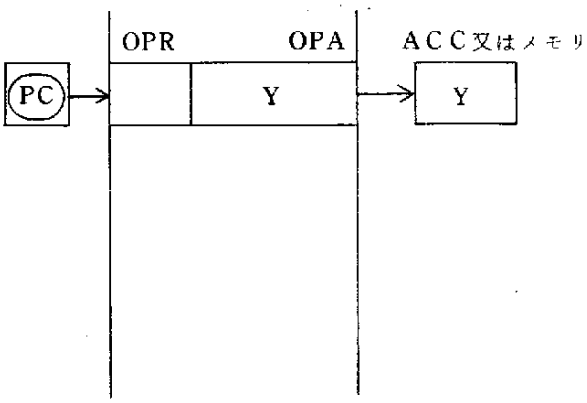
全体のメモリを、いくつかのページに分けオペランド部は、自分の命令の入っているページと０ページの全てをアクセス出来るようにしておく。他のページと共有するデータを０ページにおき、他のページにあるルーチンへのＪＵＭＰ は間 接アドレスを用いる。

(ト) 拡張アドレス

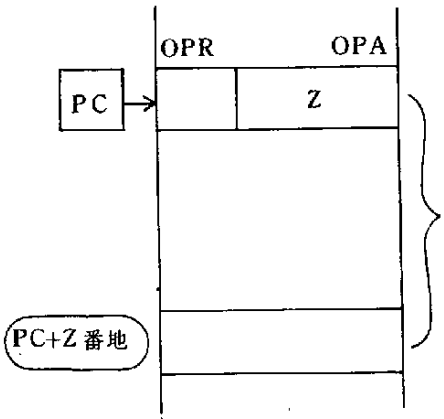
命令のオペランド部で指定しきれないアドレスを、メモリバンクの切換えで実現するものである。たとえば４Ｋメモリがアドレス出



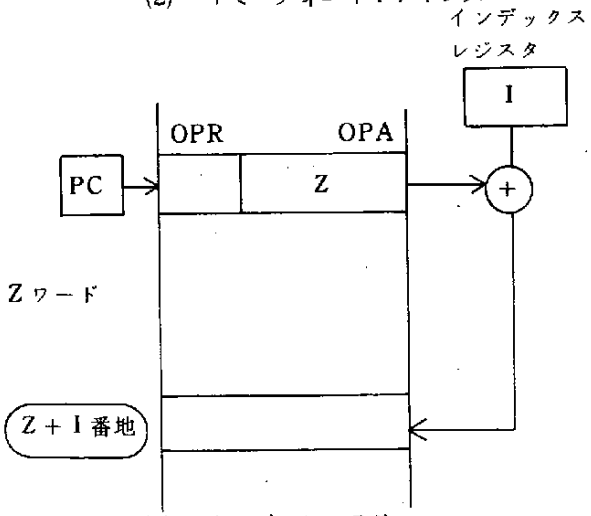
(1)　直接アドレス方式



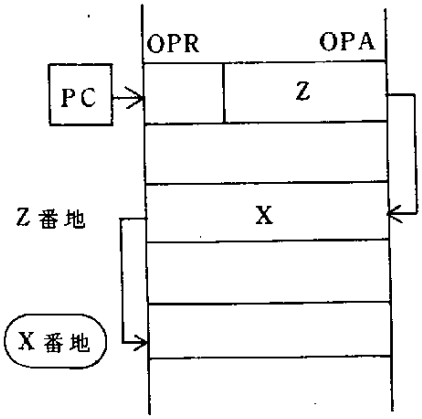
(2)　イミーディエイトアドレス



(3)　間接アドレス



(4)　インデックスアドレス



(5)　相対アドレス

OPR：オペレーションコード

OPA：オペランド

P　C：プログラムカウンタ

：最終的にアクセスした番地

第２－５図　アドレッシングのメカニズム

来る最大のマイクロプロセッサに、何台かの４Ｋメモリと、現在アクセス出来るメモリはどれかを示すレジスタを用いることにより全てのメモリへのアクセスが可能となる。

(3) 命令セット

命令の形式と機能は、アプリケーションプログラムに大きな影響を与える。命令の形式には、ユニペランド方式、２オペランド方式３オペランド方式が有るが、主に用いられるものは、１か２オペランド方式であるので、ここではこの２つを考える。１オペランド方式は、演算のための一方のソースデータのアドレスを命令の内に示し、もう一方のソースと演算結果は、アキュームレータと決めてしまう方法である。２オペランド方式には次の３つの種類がある。①レジスタとレジスタ間のオペレーション、②レジスタとメモリ間のオペレーション。

１オペランド方式は、結果を必ずアキュームレータに入れるため、次にこの結果を用いない演算をしようとすると、アキュームレータの内容を待避しなくてはならない。２オペランド方式の①は、２つのレジスタの内容を演算し結果をレジスタに格納するものである。計算処理の中では、中間結果がいくつも出るが、これらをいちいちメモリへ格納していたのでは、アクセス時間が大きくなり、効率がおちる。そこで中間結果をレジスタにとっておき、レジスタとレジスタ間のオペレーションで処理をすますことは、非常に有効である。しかしレジスタの数には限りが有り、メモリを使わざるを得ない。そこでメモリを使ったオペレーション②、③が必要になる。③は、最も汎用的で、便利な命令であるが、ＣＰＵ の制御が複雑になったり、命令長が長くなるためマイクロプロセッサでは今のところ用いられていない。

命令を機能面から分類すると次の３種類になる。

① 転送命令 (LOAD, STORE, MOVE, SWAPなど)

② 演算命令 (ADD, SUB, AND, OR, MRYなど)

③ 制御命令 (TEST, JUMP, COMPARE, CALLなど)

マイクロプロセッサはこれらの命令 バランス良く備っていることが大切な要素となる。このうち①と②は、計算処理を進めていく上で対になって使われることが多く、総合的に検討する必要がある。

マイクロプロセッサは、コントロール用に使われることが多く、このような時には、①②よりむしろ③が重要な点となる。以下③の中でサブルーチンコール、スタックオペレーションについて述べる。

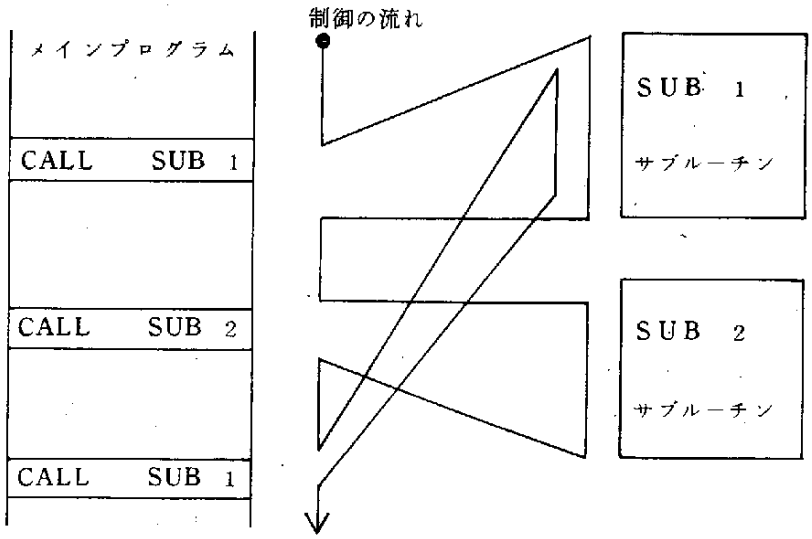
(イ) サブルーチンコール

第2－6図は、メインプログラムからサブルーチンを呼ぶ時の制御の流れを示している。この図から見るとサブルーチコールは、単にJUMP命令と田じに思えるが、1つ違いがある。それは、単にJUMPするだけではサブルーチン終了時にメインプログラムの戻り番地がわからなくなるので、JUMP前に戻り番地を待避している点である。ここで問題となるのは、戻り番地の待避場所であるが先ず考え付くのがレジスタである。即ち一つのレジスタをあらかじめ決めておき、そこを戻り番地待避に使えば、サブルーチンから戻ることが出来る。しかし実際のプログラムでは、サブルーチンの中からまた別のサブルーチンを使うことがよくあり、そのような時には、レジスタが1つでは足りなくなる。このような時便利なものがスタックである。スタックは「データを積み重ねる」という意味で、必ずデータを取り出す時は、最後に積まれたものを取り出すしくみになっている。故にサブルーチンを次々と使う（ネスティング）時は、そのたびにスタッフに戻り番地を乗せていき、サブルーチン終了時には、戻り番地をスタックのトップから取っていけばよい。第2－7図はその様子を示している。

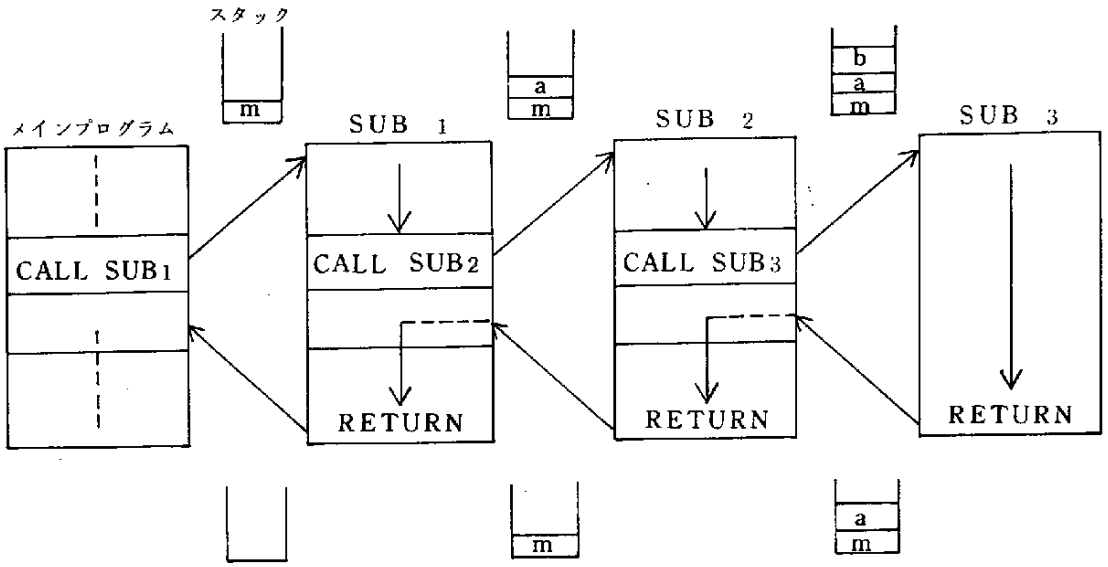
ここで命令とは少し話がそれるがスタックの実現法を記しておく。これには２つの方法が考えられる。第１は、スタックに入るデータはメモリに置かれ、マイクロプロセッサ内のスタックポインタ（レジスタ）に、そのスタックの一番上のデータの入っている番地を格納しておく方法。第２は、ハードウェアスタックを、マイクロプロセッサ内に置き、そのオーバフローはソフトウェアで管理する方法である。

(ロ)　スタック操作命令

これは、スタックに積み上げろというプッシュ命令と、スタックのトップを取出せというポップ命令がある。サブルーチンコール時の、引数の受渡しや、割込み発生時の、ステータスの待避にこれは便利な命令である。



第２－６図　　サブルーチンを使用した時の制御の流れ



第２－７図　　スタックによる戻り番地の制御

(4)　割込み

計算機の処理には、非同期に計算機外部に発生した情報に基づいて処理を行なう場合がある。例えば化学工場でのプロセス制御で、温度や圧力の異常上昇が発生した時には、それに対処するプログラムが走らねばならないし、入出力装置へ出した入出力命令による動作の終了を知りたい場合もある。このようにマイクロプロセッサ外部に発生した状態を内部に取り込む方法には２つある。第１は、マイクロプロセッサが、このような情報を内部に読込んでは調べるという処理をくり返す方法で当然その時他の仕事は行なえない。第２の方法は、外部からマイクロプロセッサに信号線を導びき、外部に所定の変化が生じた時にだけマイクロプロセッサに知らされるというもので、これが割込みである。この方法では、外部に起こる変化

をマイクロプロセッサが常に監視する必要は無く、マイクロプロセッサは外部から信号が入るまで他の仕事をして、割込みが入った時のみその処理を行なえば良い。これからわかるように割込み機能の有無、割込み処理のハードウェアでのサポート程度は、マイクロプロセッサ選択の大きなポイントである。

第２－８図は、割込み発生時のマイクロプロセッサの動作を示している。メインプログラムが走っている時、割込みが発生したとすると、処理中の命令を終了した後で、マイクロプロセッサの内部のプログラムカウンタ等必要なレジスタの内容をスタックにプッシュする①。次に②で示すように割込み処理ルーチンへジャンプする。割込み発生時のスタックへのプッシュは、自動的に行なわれるものと、プログラムカウンタ以外は、割込み処理ルーチンで行なうものががある。割込み処理が終ると最後のリターン命令で③に示す、待避したデータをレジスタに戻す操作を行ない、④のように元のメインプログラム実行を続ける。

割込み信号をマイクロプロセッサに取り込む方法には２種類ある。第１は、全割り込みソースをＯＲして１つの割込みとして取込むもの、これはシングルレベル割込と呼ばれる。第２は、各割込みが独立にマイクロプロセッサに入るもので、ベクトルド割込みと呼ばれる。

(5)　マイクロプログラマビリティ

マイクロプログラム計算機の機械命令は普通マイクロプログラムのサブルーチンとして実現されている。故にマイクロプログラムのサブルーチンを作ることにより、いろいろな機械語が作り出せることになる。これは計算機の構成を変更したとも見る事が出来る。第２－９図は、ＨＰ２１００というミニコンのコンパイラ言語、アセンブラ言語、マイクロプログラム言語の対応を示している。即ち、コ

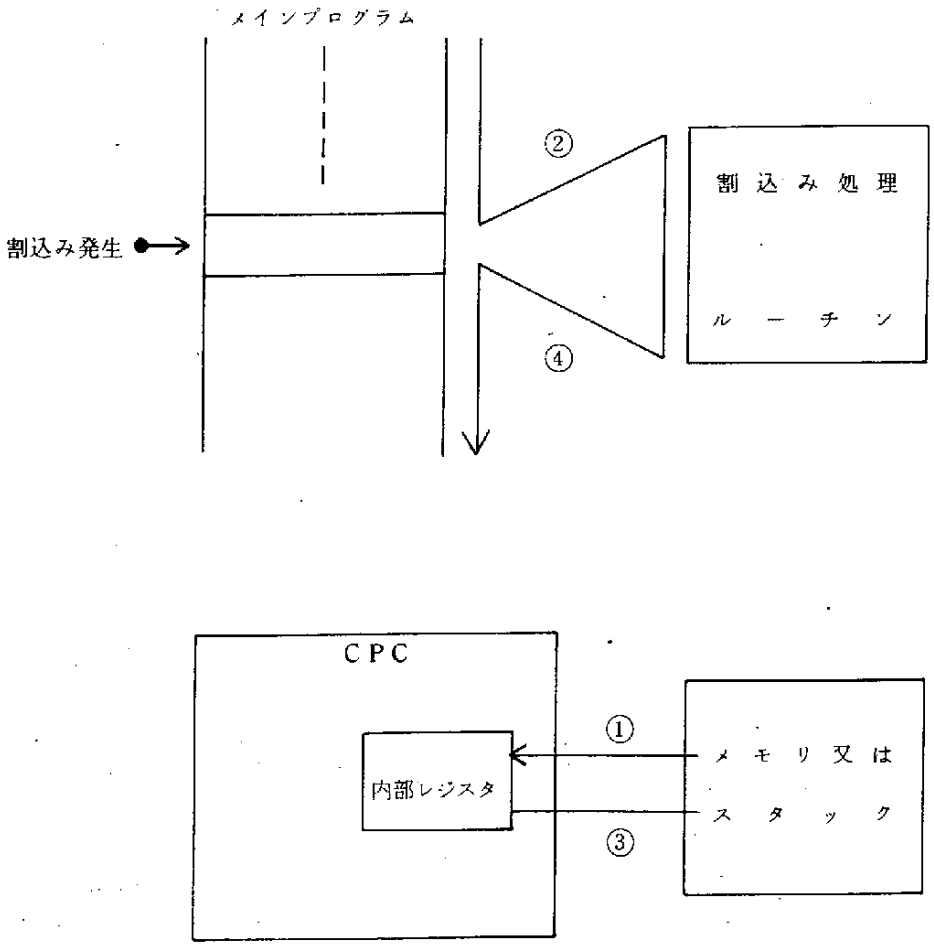
ンパイラ言語の「X番地とY番地の内容を加えてZ番地に結果を入れろ」という命令が、アセンブラ言語では、「X番地の内容をアキュームレータに入れろ」「アキュームレータとY番地の内容を加え結果をアキュームレータに入れろ」「アキュームレータの内容をZ番地へ移せ」という3つの命令で構成されていることを示す。又1つのアセンブラ命令はそのアセンブラ命令をメモリからCPUにフェッチする3ステップのマイクロ命令と、そのアセンブラ命令を実行する3ステップのマイクロ命令から出来ていることがわかる。これは、一例であるが、これから予想が付くとおり、もしマイクロプログラムがユーザに開放されていたとすると、ユーザは、マイクロプログラムを書くことにより、任意の機械語を作り上げることが出来る。また書くプログラムの量が限られている時は、全体をマイクロプログラムで書いてしまうことも可能である。この様にマイクロプログラムをユーザに開放したものをユーザマイクロプログラマブルな計算機という。

(6) 入出力システム

マイクロプロセッサは、システムの中に用いられることが多く、そのような場合マイクロプロセッサにとってまわりのシステムは入出力装置として扱うことになる。このようにマイクロプロセッサの入出力は、システム構成上重要な点である。マイクロプロセッサの入出力機構には、以下に示す3種類がある。

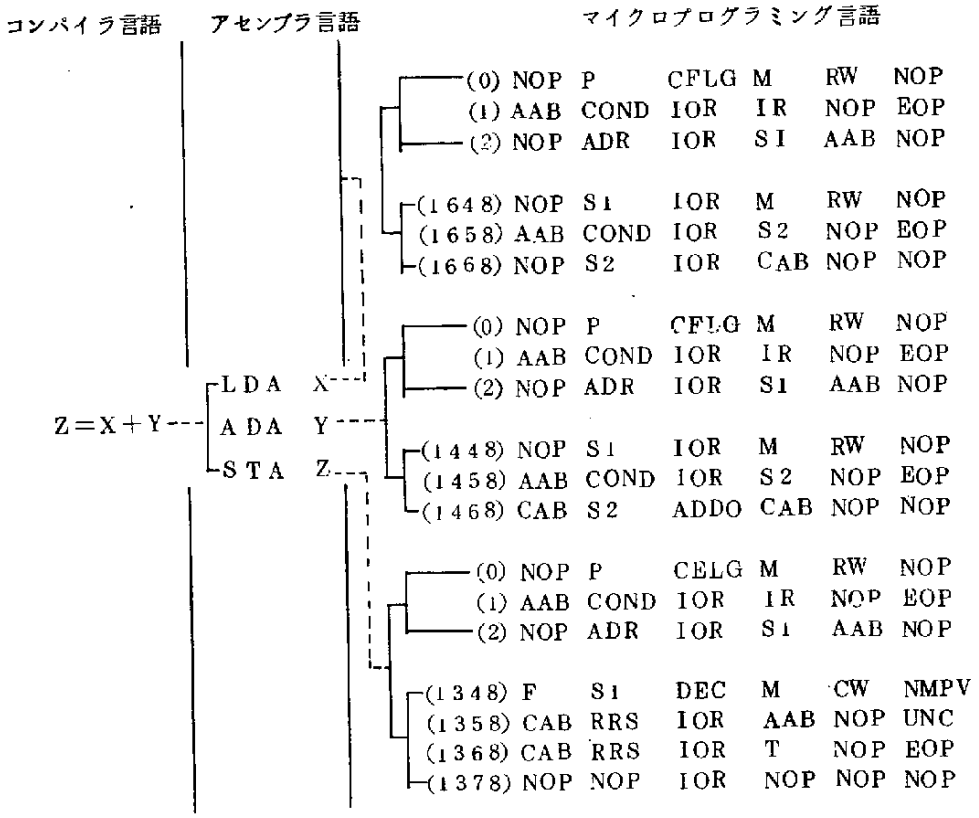
① プログラムによる入出力

一単位データ（1バイト、1ワード等）の入力又は出力を行なうのに、1つの入出力命令を発するもの。故にいくつものデータを転送する場合、1つ1つのデータに対して転送命令を出す必要がある。一般に出力されるデータは汎用レジスタ又は専用レジスタに入れられる。命令が実行される時は、先ず入出力装置に信号

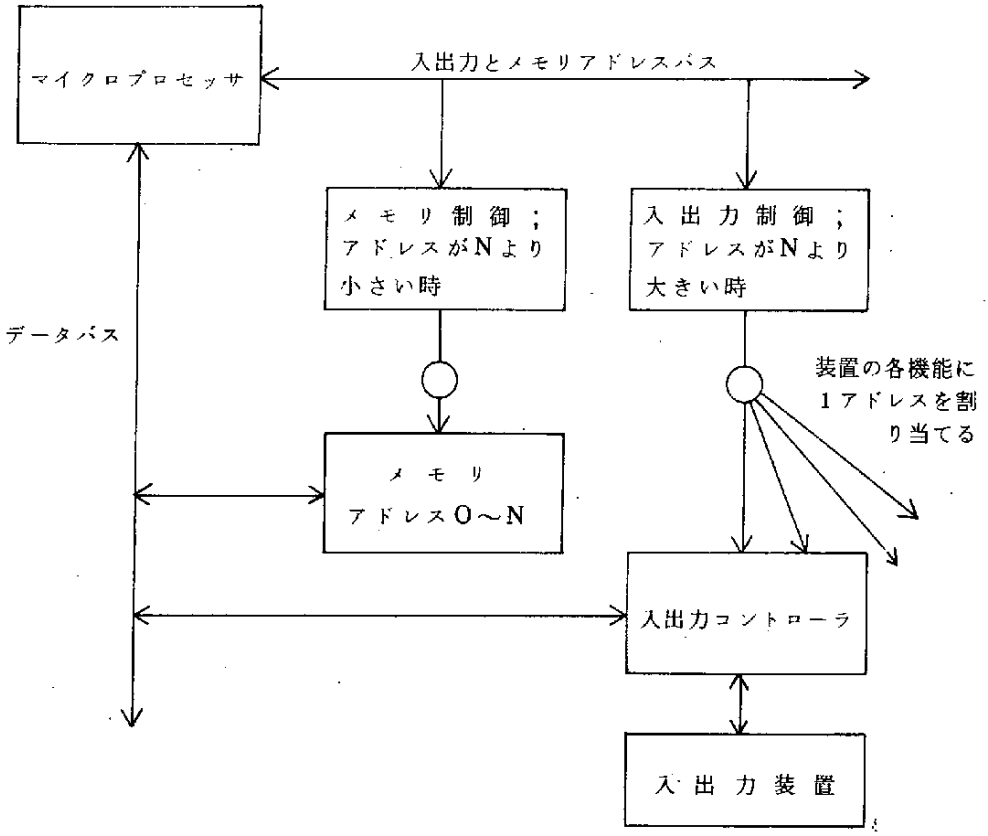


第２－８図　　割込み処理の流れ

を送る必要がある。一般には、入出力装置にメモリ番地の一部を割当て、入出力命令のアドレス部で機器の指定を行なう。第２－１０図はその様子を示している。又マイクロプロセッサによっては、入出力装置番号を専用レジスタにより指定する場合もある。命令が実行されるとデータはバスを通して入出力装置に送られる。マイクロプロセッサが入出力動作の終了を確認する方法は２つある。第１は、プロセッサが入出力装置からステータスを読込みチェックする方法でポーリングと呼ばれる。第２－１１図は、その構成を示している。もう一つの方法は、動作終了時に入出力装置がマイクロプロセッサに信号を送る方法で割込み方式という。これについては、第 2.5.2 項で述べる。



第２－９図　　マイクロプログラム例

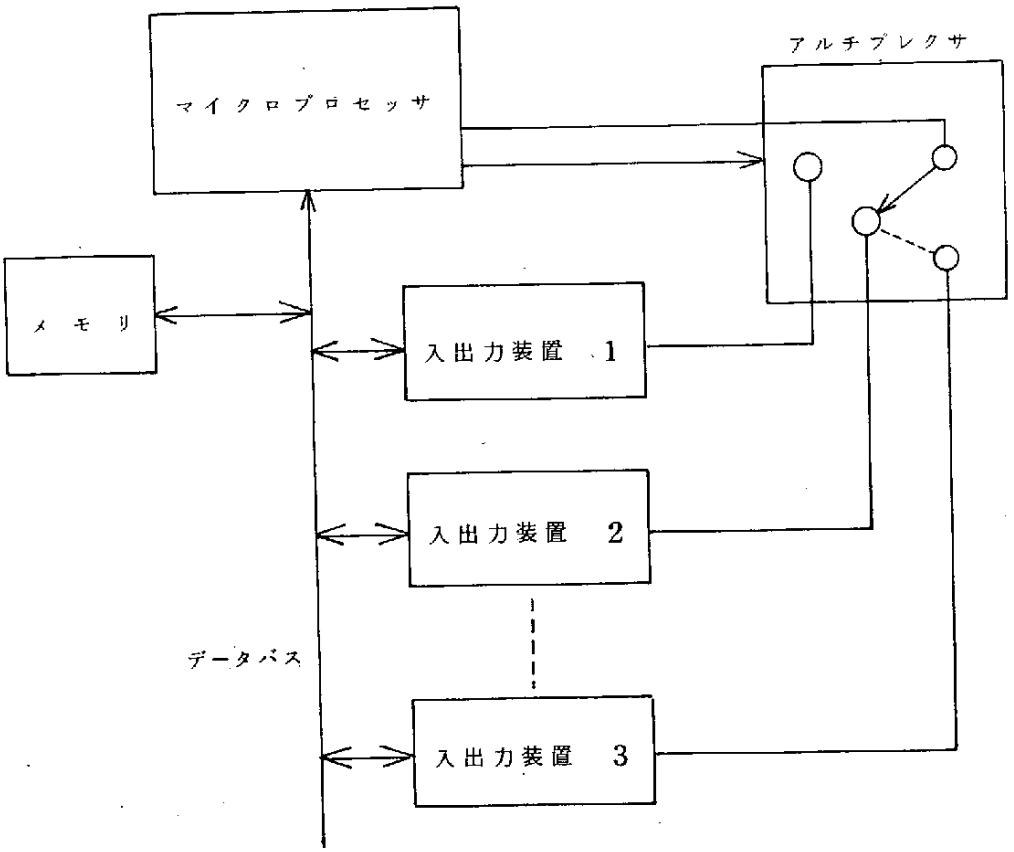


第２－１０図　　プログラムによる入出力：外部バスの数を減らすために、入出力装置にメモリのアドレスを割当てる。

② ダイレクトメモリアクセス（DMA）

入出力装置とメモリの間のデータ転送を行なおうとすると①ではデータがマイクロプロセッサを経由して転送された。これに対し、ダイレクトメモリアクセスは、マイクロプロセッサが一度転送命令を出すと後は、マイクロプロセッサを介さずにメモリと入出力装置の間で直接転送を行なうため高速転送が可能となる。マイクロプロセッサがDMA 実装可能な場合、普通はDMA 転送を行ないやすくするためCPU の動作を一時ストップさせる端子があるのみである。マイクロプロセッサで内部にDMA 機能を持っ

ているものは少ない。



第２－１１図　　ポーリング方式による外部状態の入力

③　ファントム（幻）入出力

これは、各データの転送にプログラムが関与するという点で①に似ているが、①とのちがいは、入出力命令を用いない点である。即ち各入出力装置にメモリ番地を割当てておき、普通のメモリ参照命令を発することにより入出力動作を行なわせるものである。例えばＸ番地の内容にＹ番地の内容を加えてＺ番地に格納するという命令でＺ番地を入出力装置に割当てておけば、演算結果が入出力装置に出力されるようになる。

(7) ワード数

ワードサイズを考える時、マイクロプロセッサは、２つの種類に分けて考える必要がある。第１はミニコンを縮少したタイプのものこれは速度は遅いが機能がクローズされて使いやすく一般にはＭＯＳ－ＬＳＩで作られている。第２は、バイポーラ素子によりビットスライスのプロセッサで、高速ではあるが、ビット巾の大きいＣＰＵを作ろうとするとこれを並列につながなければならない。

前者は、現在４、８、１２、１６ビットのものが世に出ているが、このうち４ビットのものは、８ビットのものを４ビットのものでシミュレートしている感がある。

一般にプログラムの容易さは、次のものに反比例するといわれている。

①ワードサイズ：小さなデータワードサイズではマルチワードオペレーションが必要となる。

②ビット数の異なるレジスタの数

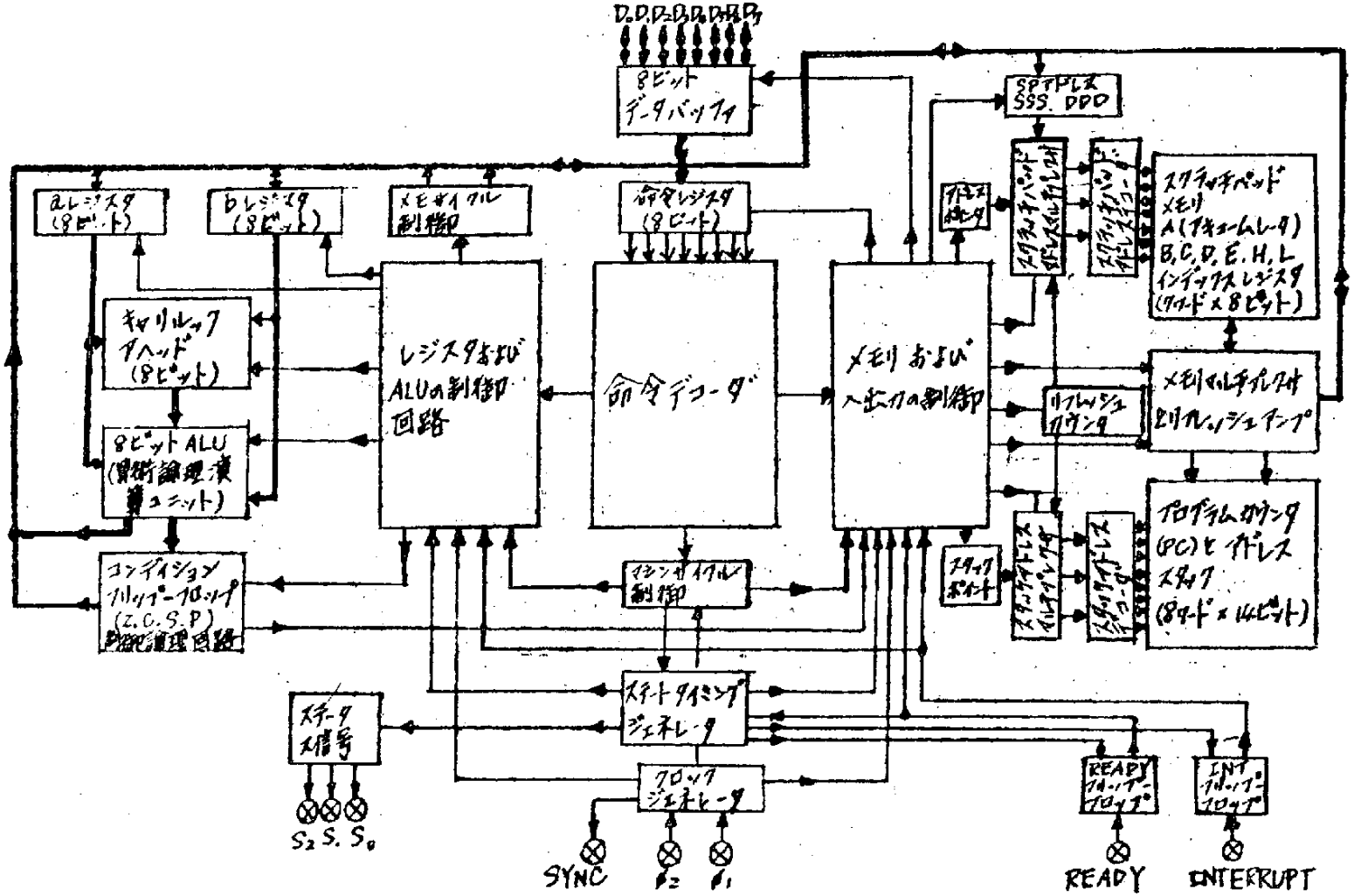
簡単にレジスタ間のデータ転送やオペレーションが出来ない。

2.2.2 マイクロプロセッサの例

ここでは、代表的なマイクロプロセッサの例を示す。

(1) ＩＮＴＥＬ８００８

ＩＮＴＥＬ８００８は、ＰチャネルMOS により作られており、１８ピンを持つ８ビットワンチップCPU である。命令の分析はチップ上で行なわれ、もちろんユーザマイクロプログラムは出来ない。第２－１２図は、８００８のブロック図であり、その主な機能は次のとおりである。



第２－１２図　　８００８のブロック図

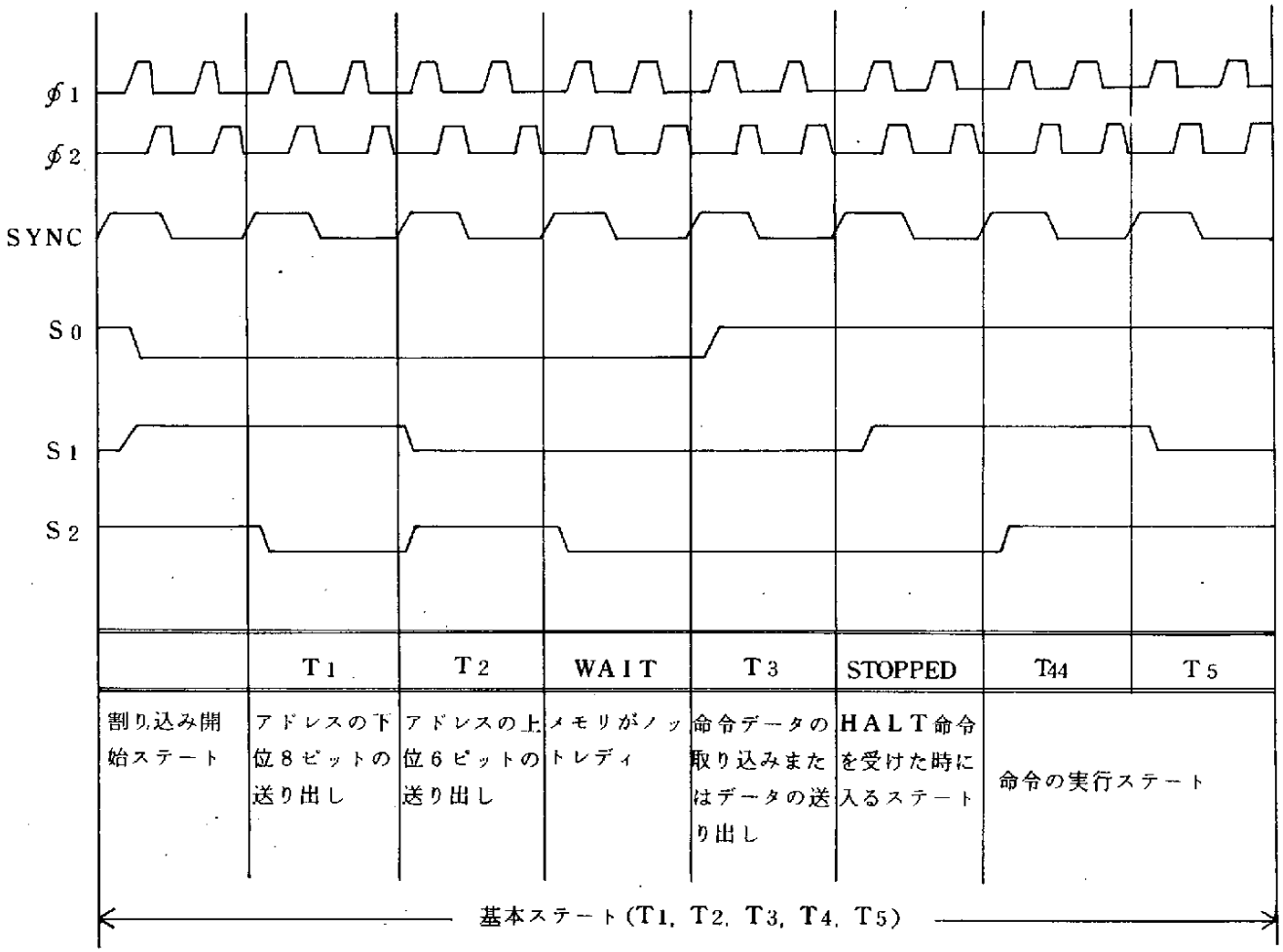
・スクラッチパッドメモリは、1つのアキュームレータ(A)と、6つの汎用レジスタ（B、D、E、H、L）から成る。
・サブルーチンや割込み処理用に1つのプログラムカウンタとクレベルのスタックを持つ。
・算術論理演算の結果をテストする4つのコンディションフラグを持つ。
・ＡＬＵで演算される2つのオペランドは、レジスタａとｂに保持され演算結果はａに戻される。

命令の実行は、最大3マシンサイクルより成立ち、各マシンサイクルは最大５ステート（Ｔ１～Ｔ５）を必要とする。しかし不必要なステートはスキップされる。第２－１３図は基本サイクルタイミングを示している。ここで、Ｔ１Ｉは、割込みが発生した時に使われるＴ１の特別型である。ＣＰＵが現在どのステートに有るかはＳＯ、Ｓ１、Ｓ２のピンに出力される。ステートとＳＯ～Ｓ２の関連も第２－１３図に示されている。次に各ステートでの動作の概要を示す。Ｔ１の間に１４ビットのメモリアドレスの下位８ビットがデータバスへ出力され、プログラムカウンタに１が加えられる。Ｔ２では、メモリアドレスの残り６ビットとマシンサイクルの種類を区別する２ビットの情報をデータバスへ出力する。ここでマシンサイクルの区別とは、命令の最初のマシンサイクル、命令の２又は３バイト目の読込み、入出が命令、メモリライトの４つである。ＣＰＵと外部メモリや入出力装置と間のデータ転送は非同期に行なわれるため、外部ではデータの準備が出来た時ＲＥＡＤＹピンに信号を送る。ＣＰＵはこの信号が出来るまでＷＡＩＴステートのままで待ち　この信号が入ってからＴ３ステートに入る。Ｔ３ではメモリから読込まれたデータをデータバスから取込み、Ｔ４、Ｔ５で実行する。但し実行を必要としないマシンサイクルもあり、その場合はスキップする。

命令は全部で８種類あり、１バイト、２バイト、３バイト命令の３種類に分類出来る。１バイト命令は主にレジスタ・レジスタ命令メモリ参照命令、１／０命令に使われ、２バイト命令はイミディエイトモードの命令に、３バイト命令はジャンプ・コール命令に使われる。第２－１４図には、1.2.3マシンサイクル命令の、実行シーケンスとマシンサイクルの対応の例を示す。ここで、この図の見方を簡単に示しておく。最初のマシンサイクルのＴ1.Ｔ2.でフエッチされる命令のアドレスが出力される。図でＰＣＬ、ＰＣＨＯＵＴとあるのがこれでプログラムカウンタの内容を出力することを意味している。Ｔ３は命令（インストラクション）のフェッチでＦcsch Ｉ stと記されている。ここでフェッチされた命令は、インストラクションレジスタの他にレジスタｂにも格納される。これは、命令によってはこの命令の一部を後でデバイスに転送する必要が有るためである。レジスタ間のバスが１本のため実行シーケンスはシーケンシャルに進む。例えば、レジスタの内容を他のレジスタに移す命令（Ｌ$r_1 r_2$）では、ソースレジスタ（ＳＳＳ）が先ずレジスタｂに移され、別のステートでそれを送り先レジスタ（ＤＤＤ）に移す。これらの例は、ＣＰＵが一本のバスをどう時分割で使っているかを良く示している。８００８の命令表のみを第２－２表に示す。各命令の意味はくＣＯＭ－８の説明を参照されたい。

(2) ＮＥＣ くーＣＯＭ８

ＩＮＴＥＬは、８００８に続いて、この機能を完全に含んだ形の改良機８０８０を発表した。ここで取上げるｒーＣＯＭ８はチップ内部の回路は８０８０と異なるが、これをブラックボックスとして見た時一部の制御信号のタイミングを除いてコンパティブルになっている。又命令においては、ＤＡＡ命令において10進演算補正機能が追加されている。ニーモニック記号、インストラクションコードは完全に一致



第2－13図　8002の基本タイミング

Instruction Coding $D_7D_6D_5D_4D_3D_2D_1D_0$	Operation	# of States to Execute Instruction	Memory Cycle One (1)					Memory Cycle Two					Memory Cycle Three				
			T1(1)	T2	T3	T4(3)		T1	T2	T3	T4(3)	T5	T1	T2	T3	T4(3)	T5

Index Register Instructions

11 DDD SSS	Lr1r2	5	PC_L OUT	PC_H OUT	Fetch Instr	SSS to REG b	REG b to DDD										
11 DDD 111	LrM	8	PC_L OUT	PC_H OUT	Fetch Instr			REG L OUT	REG H OUT	DATA TO REG b		REG b TO DDD					
11 111 SSS	LMr	7	PC_L OUT	PC_H OUT	Fetch Instr	SSS to REG b		REG L OUT	REG H OUT	REG b TO OUT							
00 DDD 110	LrI	8	PC_L OUT	PC_H OUT	Fetch Instr			PC_L OUT	PC_H OUT	DATA TO REG b		REG b TO DDD					
00 111 110	LMI	9	PC_L OUT	PC_H OUT	Fetch Instr			PC_L OUT	PC_H OUT	DATA TO REG b			REG L OUT	REG H OUT	REG b TO OUT		
00 DDD 000	INr	5	PC_L OUT	PC_H OUT	Fetch Instr												
00 DDD 001	DCr	5	PC_L OUT	PC_H OUT	Fetch Instr												

Instruction Coding $D_7D_6D_5D_4D_3D_2D_1D_0$	Operation	# of States to Execute Instruction	Memory Cycle One (1)					Memory Cycle Two					Memory Cycle Three				
			T1(1)	T2	T3	T4(3)	T5	T1	T2	T3	T4(3)	T5	T1	T2	T3	T4(3)	T5

I/O Instructions

01 00M MM1	INP	8	PC_L OUT	PC_H OUT	Fetch Instr			REG A TO OUT	REG b TO OUT	DATA TO REG b	Cond FF out	REG b TO REG A					
01 RRM MM1	OUT	6	PC_L OUT	PC_H OUT	Fetch Instr			REG A TO OUT	REG b TO OUT	X							

Machine Instruction

00 000 00X	HLT	4	PC_L OUT	PC_H OUT	Fetch Instr												
11 111 111	HLT	4	PC_L OUT	PC_H OUT	Fetch Instr												

第２－１４図　８００８　命令実行タイミング例

第２－２表　８００８とμCOM８の命令比較

	8008	μCOM8
RST INSTRUCTION	RST n	RST n
HALT INSTRUCTION	HLT	HLT
NOP INSTRUCTION	NOP (注1)	NOP

1-BYTE DATA TRANSFER INSTRUCTIONS	MOV r1, r2
	MOV r, M
	MOV M, r
	MVI r, ⟨B2⟩
	MVI M, ⟨B2⟩
	STAX rp1
	LDAX rp1
	STA ⟨B3, B2⟩
	LDA ⟨B3, B2⟩
		MOV r1, r2
		MOV r, M
		MOV M, r
		MVI r, ⟨B2⟩
		MVI M, ⟨B2⟩
		STAX rp1
		LDAX rp1
		STA ⟨B3 B2⟩
		LDA ⟨B3 B2⟩

2-BYTE DATA TRANSFER INSTRUCTIONS	—— —— —— —— ——	LXI rp2, ⟨B3B2⟩ SHLD ⟨B3 B2⟩ LHLD ⟨B3 B2⟩ XCHG XTHL SPHL
INTERRUPT FLIP-FLOP INSTRUCTIONS	—— ——	EI DI
PUSH/POP INSTRUCTIONS	—— ——	PUSH rp3 POP rp3
CARRY FLAG INSTRUCTIONS	—— ——	STC CMC

(注1) MOV A, A 命令などで代用する。
(注2) r≠A
(注3) I/O命令は、8008の場合1バイト、μCOM8の場合2バイトを占有する。

	8008	μCOM8
INCREMENT DECREMENT INSTRUCTIONS	INR r (注2)	INR r
		INR M
	DCR r (注2)	DCR r
		DCR M
	——	INX rp2
	——	DCX rp2
ARITHMETIC AND LOGICAL INSTRUCTIONS	ADD r	ADD r
	ADC r	ADC r
	SUB r	SUB r
	SBB r	SBB r
	ANA r	ANA r
	XRA r	XRA r
	ORA r	ORA r
	CMP r	CMP r
	ADD M	ADD M
	ADC M	ADC M
	SUB M	SUB M
	SBB M	SBB M
	ANA M	ANA M
	XRA M	XRA M
	ORA M	ORA M
	CMP M	CMP M
	ADI ⟨B2⟩	ADI ⟨B2⟩
	ACI ⟨B2⟩	ACI ⟨B2⟩
	SUI ⟨B2⟩	SUI ⟨B2⟩
	SBI ⟨B2⟩	SBI ⟨B2⟩
	ANI ⟨B2⟩	ANI ⟨B2⟩
	XRI ⟨B2⟩	XRI ⟨B2⟩
	ORI ⟨B2⟩	ORI ⟨B2⟩
	CPI ⟨B2⟩	CPI ⟨B2⟩
	——	CMA
	——	DAA
	——	DAD rp2
ROTATE ACCUMULATOR INSTRUCTIONS	RLC	RLC
	RRC	RRC
	RAL	RAL
	RAR	RAR

	8008	μCOM8
JUMP INSTRUCTIONS	——	PCHL
	JMP ⟨B3B2⟩	JMP ⟨B3B2⟩
	JC ⟨B3B2⟩	JC ⟨B3B2⟩
	JNC ⟨B3B2⟩	JNC ⟨B3B2⟩
	JZ ⟨B3B2⟩	JZ ⟨B3B2⟩
	JNZ ⟨B3B2⟩	JNZ ⟨B3B2⟩
	JM ⟨B3B2⟩	JM ⟨B3B2⟩
	JP ⟨B3B2⟩	JP ⟨B3B2⟩
	JPE ⟨B3B2⟩	JPE ⟨B3B2⟩
	JPO ⟨B3B2⟩	JPO ⟨B3B2⟩
CALL INSTRUCTIONS	CALL ⟨B3B2⟩	CALL ⟨B3B2⟩
	CC ⟨B3B2⟩	CC ⟨B3B2⟩
	CNC ⟨B3B2⟩	CNC ⟨B3B2⟩
	CZ ⟨B3B2⟩	CZ ⟨B3B2⟩
	CNZ ⟨B3B2⟩	CNZ ⟨B3B2⟩
	CM ⟨B3B2⟩	CM ⟨B3B2⟩
	CP ⟨B3B2⟩	CP ⟨B3B2⟩
	CPE ⟨B3B2⟩	CPE ⟨B3B2⟩
	CPO ⟨B3B2⟩	CPO ⟨B3B2⟩
RETURN INSTRUCTIONS	RET	RET
	RC	RC
	RNC	RNC
	RZ	RZ
	RNZ	RNZ
	RM	RM
	RP	RP
	RPE	RPE
	RPO	RPO
I/O (注3) INSTRUCTIONS	IN n	IN ⟨B2⟩
	OUT n	OUT ⟨B2⟩

r: A, B, C, D, E, H, L
rp1: BC, DE
rp2: BC, DE, HL, SP
rp3: AおよびフラグワードBC, DE, HL
M: HLによってアドレッシングされるメモリアドレス
⟨B2⟩: 第2バイト
⟨B3⟩: 第3バイト

している。ここでは、アーキテクチャの説明に入る前に、８００８からく－ＣＯＭ８への改良点をあげるが、それらの改良はＰ－ＭＯＳに比べて速度、集積度でまさるＮ－ＭＯＳの使用によるところが大きい

(イ) 速度；命令実行時間短縮のため最大クロック周波数を、８００８が５００ＫＨｚであったものを、２ＭＨｚ した。又８００８の持つ４８種類の基本命令に新たな命令を加え７４種類にした。

(ロ) 多重割込み処理；８００８では、スタックに入れられる情報がプログラムカウンタの内容だけであった。しかしその他の一般データも入れるようになっていると多重割込み処理に便利である。そこで－ＣＯＭ８では、１６ビットのスタックポインタをＣＰＵ内に置き、スタックの内容は外部メモリに格納されるようになっている。これによってプログラムカウンタの内容のみならずアキコームレータ、フラグ、６個のデータレジスタなどの内容がソフトウェアの命令で外部メモリにスタックされる。

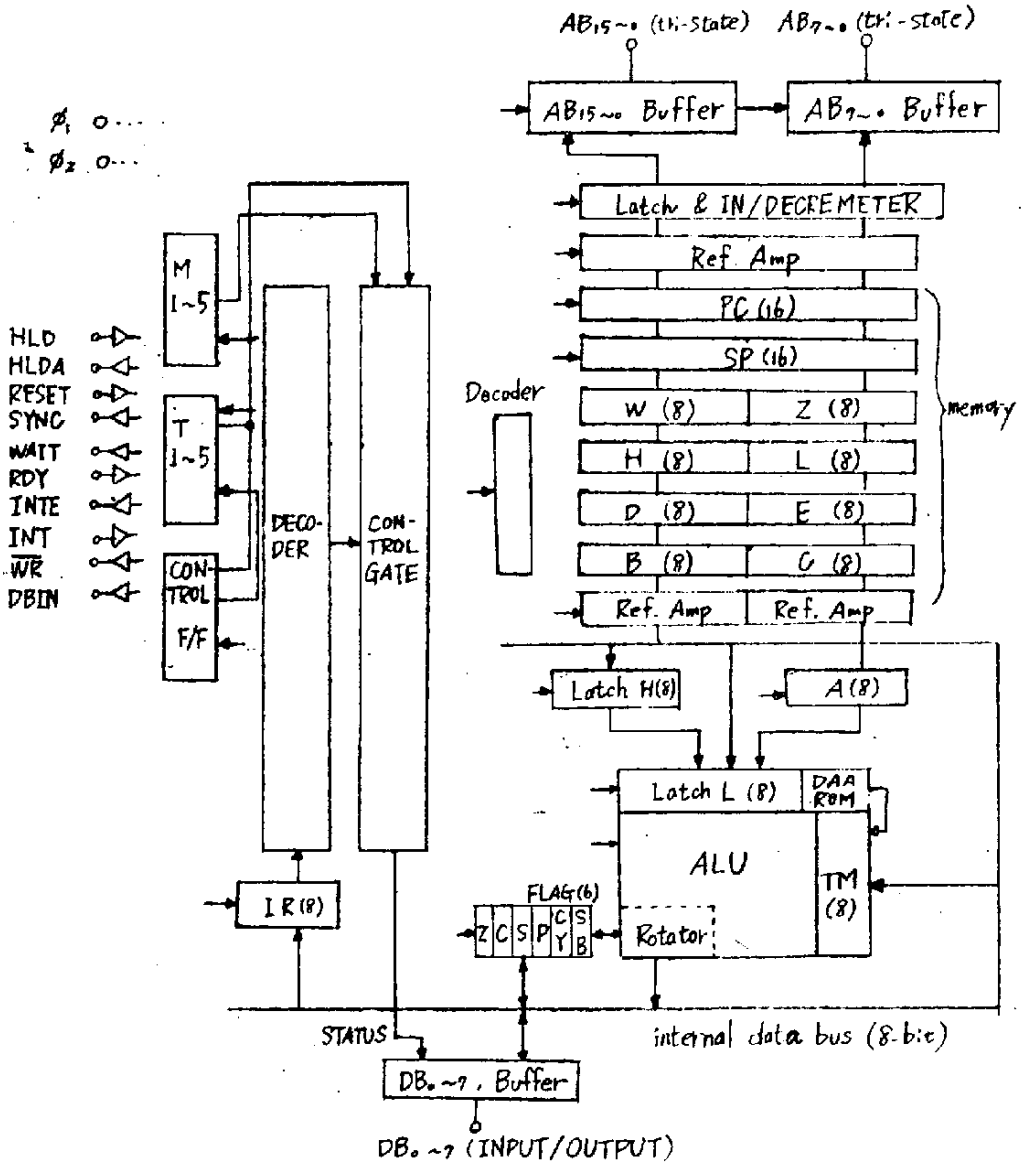
(ハ) インタフェース；８００８では、アドレスとデータが同一端子を通して転送されていたが、く－ＣＯＭ８では、アドレス転送用に16本のピンとデータ転送用に８本のピンがあるため外部にアドレスレジスタを置く必要が無い。ただプロセッサのステータスを送り出すピンがないため、このデータはデータバスを利用して外部に送り出す。またく－ＣＯＭ８では、ＤＭＡやマルチプロセッシング方式を実現し易くするためＨＯＬｂ機能を持ち、外部からの要求がＨＯＬＤ端子に入ると、各マシンサイクルの最後でプロセッサがホールド状態になる。

(ニ) 新命令；第２－３表参照

次にアーキテクチャを述べる。く－ＣＯＭ８は、ＮチャネルＭＯＳで構成された８ビット並列処理用ワンチップＣＰＵである。命令は74種類有り、64 Kバイトのメモリを直接アドレスでき、入出力ポ

ートは最大256個まで制御出来る第2－15図にブロック図を示す。CPUは7つの基本ブロックで構成されている。すなわち、インストラクションレジスタとデコーダ・ズモリ、ALU、フラグ、タイミングジェネレータとコントローラ、アドレスラッチ／バッファ、データバッファである。ここでメモリとは、6個のデータレジスタ（B、C、D、E、H、L）と2つのテンポラリレジスタ（W、Z）からなるスクラッチパッドメモリ、16ビットのスタックポインタ、及び16ビットのプログラムカウンタを言う。アドレスバスからは16ビットのメモリアドレスが出力され、また入出力命令実行時には、アドレスバスの下位8ビットには、256までのデバイスナンバが出力され、上位8ビットには入力命令実行時にアキュームレータの内容が、出力命令実行時には、フラグビットの内容が出力される。又CPUがホールド状態では、アドレスバスはフローティングとなる。データバスは、命令とデータの転送に用いられるほかマシンサイクルの始めには、SYNC信号と同期してマシンサイクルのタイプを示すステータス情報（第2－4表）がバスに出力される。
データバスが入力モードか出力モードかは、DBIN，WRピンで外部に知らされる。CPUホールド時にはデータバスもフローティングとなる。

第2－16図は、CPUの基本タイミングを示している。1命令は1から5までのマシンサイクル（M1～M5）で実行され、各マシンサイクルは、T1～T5の5つのステートから成立っている。T1では16ビットのアドレスがアドレスバス（AB15～0）から出力される。又この時データバスにはステータス信号が出力される。HOLD要求が有るか否か、READY信号（メモリからの命令がデータバスに準備されたことを示す）がどうなっているかのチェックを行なう。また、マシンサイクルがWRITEモードであれば、CPUからデー



第２—１５図　μ—COM8のブロック図

第２－３表　μCOM8 のステータス情報

（ａ）ステータス情報

記　号	データバス	意　　味
INTA	D_0	CPUが割込み信号を受取ったことを示す
WO	D_1	WO＝１；実行中のマシンサイクルがリードメモリ又は入力動作になることを示す。 WO＝０；ライトメモリまたは出力動作になることを示す。
STACK	D_2	アドレスバスがスタックポインタからのプッシュダウン，スタックのアドレスを指示していることを示す。
HLTA	D_3	CPUがHALT命令を受取ったことを示す。
OUT	D_4	アドレスバスが機器番号を出力し、データバスにアキュームレータの内容が出力されることを示す。
M_1	D_5	命令の１バイト目の取出しであることを示す。
INP	D_6	アドレスバスが機器番号を出力し、データバスが入力モードであることを示す。
MEMR	D_7	データバスがメモリの読出しに用いられることを示す。

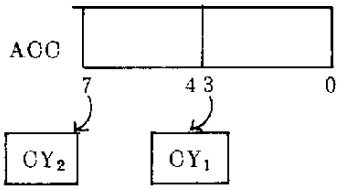
（ｂ）ステータス情報の組合わせ

データバス	ステータス情報	マシンサイクルのタイプ									
		INSTRUCTION FETCH	MEMORY READ	MEMORY WRITE	STACK READ	STACK WRITE	INPOT READ	OUTPUT WRITE	INTERRUPT ACKNOW LEOGE	HALT ACKNOWLEDGE	IMTERRUPT ACKNOW LEDGE WHILE HALT
ステータスワードの番号		1	2	3	4	5	6	7	8	9	10
D_0	TNTA	0	0	0	0	0	0	0	1	0	1
D_1	WO	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0

第２－４表　　μＣＯＭ８ 命令表

ニーモニック	命令コード 1STバイト	2Nb	3rb	サイクル数	ステート数	オペレーション	
MOVM,r₂	01DDDSSS			1	5	(r₁)←(r₂)レジスタr₂の内容をレジスタr₁にロードする。r₂の内容は変らない。r₁=DDD,r₂=SSS(≠110)	
MOVM, r	01110SSS			2	7	(M)←(r)レジスタrの内容をレジスタH.Lの内容で示すメモリ番地にロードする。	
MOV r,M	01DDD001			2	7	(r)←<B₂> 2バイト目B₂をレジスタにロードする。	
MVI r	00DDD110	B		2	7	(r)←<B₂> 2バイト目B₂をレジスタにロードする。	
MVI M	00110110	B₂		3	10	(M)←<B₂> レジスタH.Lで指定されたメモリ番地にB₂をロードする。	
LXI B	00000001	B₂	B₃	3	10	(B)←<B₃> (C)←<B₂>	B₂とB₃の内容をレジスタ対(BC.DE.HL)又はスタックポインタ(SP)にロードする。
LXI D	00010001	B₂	B₃	3	10	(D)←<B₃> (E)←<B₂>	
LXI H	00100001	B₂	B₃	3	10	(H)←<B₃> (L)←<B₂>	
LXI SP	00110001	B₂	B₃	3	10	(SP)H←<B₃> (SP)L←<B₂>	
STAXB	00000010			3	7	[(B)(C)]←(A)	レジスタ対BC又はDEの内容で指定されるメモリ番地にACの内容をストアする。
STAX D	00010010			2	7	[(D)(E)]←(A)	
LDAX B	00001010			2	7	(A)←[(B)(C)]	レジスタ対BC又はDEで指定されるメモリ番地の内容をACにロードする。
LDAX D	00011010			2	7	(A)←[(D)(E)]	
SHLD	00100010	B₂	B₃	2	16	(<B₂><B₃>)←(L) (<B₂><B₃>+1)←(H)	B₂とB₃によって指定されるメモリ番地とレジスタ対HLの内容をストアする
LHLD	00101010	B₂	B₃	5	16	L)←(<B₂><B₃>) (H)←(<B₂><B₃>+1	B₂とB₃で指定されるメモリ番地の内容をレジスタ対HLにロードする
STA	00110010	B₂	B₃	5	13	(<B₂><B₃>)←(A)	ACの内容をB₂B₃で指定されるメモリ番地にストアする。
LDA	00111010	B₂	B₃	4	13	(A)←(<B₂><B₃>)	B₂B₃で指定されるメモリ番地の内容をACにロードする。

ニーモック	命令コード 1STバイト	2nd	3rd	サイクル数	ステート数	オペレーション	
ADD	10000SSS			1	4	(A)←(A)+(r)	
ADCr	10001SSS			1	4	(A)←(A)+(r) +(CA)	レジスタrの内容とアキュームレータの内容とを演算し結果をアキュームレータにセットする。その結果によりすべてのフラッグが影響を受ける。 rの内容は不変
SUBY	10010SSS			1	4	(A)←(A)−(r)	
SBBr	10011SSS			1	4	(A)←(A)−(r) +(BORROW)	
ANAr	10100SSS			1	4	(A)←(A)∧(r) (C)← 0	
XRAr	10101SSS			1	4	(A)←(A)∨(r) (C)← 0	
ORAR	10110SSS			1	4	(A)←(A)∨(r) (C)← 0	
CHPr	10111SSS			1	4	(A)−(r) SSS≠110	CMPの場合アキュームレータは不変。
ADDM	10000110			2	7	(A)←(A)+(M)	
ADCM	10001110			2	7	(A)←(A)+(M) +(CARRY)	レジスタHLで指定されるメモリ番地の内容とアキュームレータの内容を演算し結果をアキュームレータにセットする。その結果によりすべてのフラッグが影響を受ける。
SUBM	10010110			2	7	(A)←(A)−(M)	
SBBM	10011110			2	7	(A)←(A)−(M) +(BORROW)	
ANAM	10100110			2	7	(A)←(A)∧(M) C ← 0	
XRAM	10101110			2	7	(A)←(A)∨(M) C ← 0	
ORAM	10110110			2	7	(A)←(A)∨(M) C ← 0	
CMPM	10111110			2	7	C ← 0	CMPの場合アキュームレータは不変
ADI	11000110	B_2		2	7	(A)←(A)+<B_2>	
ACI	11001110	B_2		2	7	(A)←(A)+<B_2> +(CARRY)	命令の2バイト目B_2の内容とアキュームレータの内容を演算し、結果をアキュームレータにセットする。結果によりすべてのフラッグが影響を受ける。 ホローはキャリF／Fをセットする。
SUI	11010110	B_2		2	7	(A)←(A)−<B_2>	
SBI	11011110	B_2		2	7	(A)←(A)−<B_2> +(BORROW)	
ANI	11100110	B_2		2	7	(A)←(A)∧<B_2> C ← 0	
XRI	11101110	B_2		2	7	(A)←(A)∨<B_2> C ← 0	

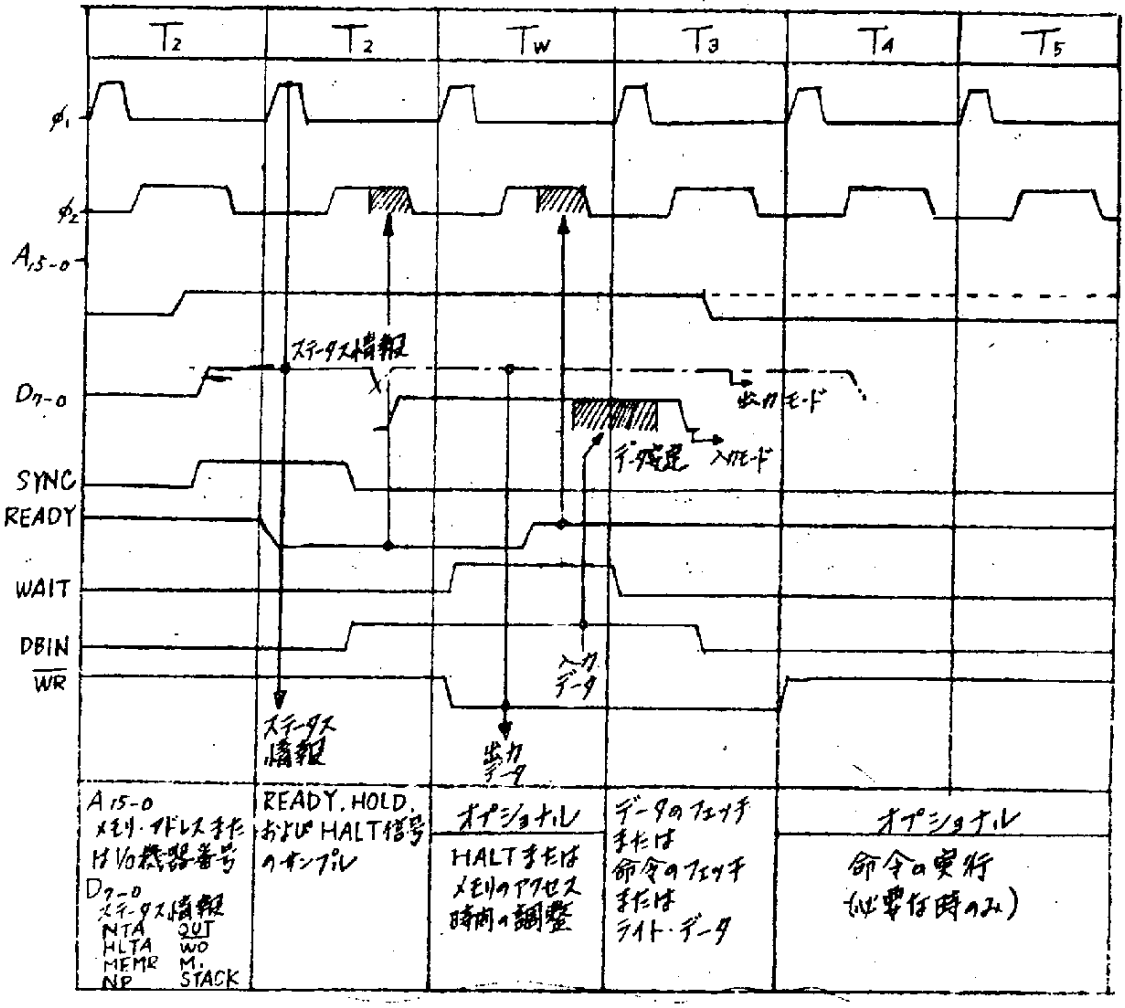
ニーモック	命令コード 1STバイト	2nd	3rd	サイクル数	ステート数	オペレーション
ORI	11110110	B_2		2	7	(A)←(A)V<B_2> C ← 0
CPI	11111110	B_2		2	7	(A)←<B_2>
DADB	00001001			3	10	(H)(L)←(H)(L) +(B)(C)
DADD	00011001			3	10	(H)(L)←(H)(L) +(D)(E)
DADH	00101001			3	10	(H)(L)←(H)(L) +(H)(L)
DADSP	00111001			3	10	(H)(L)←(H)(L) +(SP)
DAA	00100111			1	4	ACCの8ビットのデータが2桁の4ビットBCDにフラグビット(CY)を用いて補正される。  演算の結果 ・Y ≧10のとき (Y)←(Y)+6 その結果 CY_1=1のとき (Y>F) (X)←(X)+1 ・X≧10のとき (X)←(X)+6 その結果により CY_2=1 (X>F)
CMA	00101111			1	4	(A)←($\overline{A}$) アキュームレータの内容の補数(コンプリメント)をとる。全てのフラッグは不変。
STC	00110111			1	4	(C)←1 キャリーF/Fを1にセットする その他のフラッグは不変
CMC	00111111			1	4	(C)←($\overline{C}$) キャリーの内容のコンプリメントをとる。その他のフラッグは不変
RLC	00000111			1	4	(Am+1)←(Am)・(A_0)←(A_7)(C)←(A_7) レジスタAの内容を左へ1ビット回転させる。
RRC	00001111			1	4	(Am−1)←(Am)・(A_7)←(A_0)・(C)←(A_0) レジスタAの内容を右へ1ビット回転させる。
RAL	00010111			1	4	(Am+1)←(Am)、(A_0)←(C)、(C)←(A_7)レジスタAの内容とキャリーF/Fの内容を左へ1ビット回転

ニーモック	命令コード			サイクル数	ステート数	オペレーション	
	1ST バイト	2Nd	3rd				
RAR	00011111			1	4	(Am-1)←(Am)(A7)←(C)(C←(A0) レジスタAとキャリーF/Fの内容を右へ1ビット回転させる	
INRr	00DDD100			1	5	(r)←(r)+1	レジスタの内容を+1又は−1にする。その結果キャリーを除くすべてのフラッグは影響を受ける
DCRr	00DDD101			1	5	(r)←(r)−1	
INRM	00100100			3	10	(M)←(M)+1	レジスタHLで指定される番地のメモリ内容が+1又は−1される。キャリーフラッグ以外すべて影響を受ける。
DCRM	00110101			3	10	(M)←(M)−1	
INXB	00000011			4	5	(B)(C)←(B)(C)+1	レジスタ対(BC、DE HL)又は、SPの内容が+1される。すべてのフラグは影響されない。
INXD	00010011			1	5	(D)(E)←(D)(E)+1	
INXH	00100011			1	5	(H)(L)←(H)(L)+1	
INXSP	00110011			1	5	(SP)←(SP)+1	
DCXB	00001011			1	5	(B)(C)←(B)(C)−1	レジスタ対(BC DE HL)又はSPの内容が−1される。全てのフラグは影響されない。
DCXD	00011011			1	5	(D)(E)←(D)(E)−1	
DCXH	00101011			1	5	(H)(L)←(H)(L)−1	
DCXSP	00111011			1	5	(SP) ← (SP)−1	
XTHL	11100011			3		(L) ← [SP] (H) [SP+1]	レジスタ対HLの内容とレジスタSPで指定する。プッシュダウンスタックの内容を交換する。
XCHG	11101011			5	4	(H) (D) (L) (E)	レジスタ対HLの内容としジスタ対DEの内容を交換する。
OUT	11010011	B_2		1	10	(D_7〜D_0)←(A) (A_7〜A_0)←<B_2>	B_2で指定するI/0デバイスへACの内容を出力する。
IN	11011011	B_2		3	10	(A)←(D_7〜D_0) (A_7〜A_0)←<B_2>	B_2で指定するI/0デバイスから出力されるバス上のデータ(D_7〜D_0)をACにロードする
DI	11110011			1	4	内部の割込みイネブルF/Fをリセットする。	
EI	11111011			1	4	内部の割込みイネブルF/Fをセットする。	
NOP	00000000			1	4	ノーオペレーション	
HLT	01110110			2	7	CPUの停止 レジスタとメモリの内容は変化しない。	

ニーモニック	命令コード			サイクル数	ステート数	オペレーション
	1STバイト	2Nd	3rd			
JMP	11000011	B_2	B_3	3	10	(PC)←<B_3><B_2>B_2とB_3で指定されるメモリ番地へ無条件にジャンプする。
JNZ	11000010	B_2	B_3	3	10	(Z)=0ならば(PC)←<B_3><B_2> (Z)≠0ならば(PC)←(PC)+3
JZ	11001010	B_2	B_3	3	10	(Z)=0ならば(PC)←<B_3><B_2> (Z)≠0ならば(PC)←(PC)←(PC)+3
JNC	11010010	B_2	B_3	3	10	(C)=0ならば(PC)←<B_3><B_2> (C)≠0ならば(PC)←(PC)+3
JC	11011010	B_2	B_3	3	10	(C)=1ならば(PC)←<B_3><B_2> (C)≠1ならば(PC)←(PC)+3
JPO	11100010	B_2	B_3	3	10	(P)=0ならば(PC)←<B_3><B_2> (P)≠0ならば(PC)←(PC)+3
JPE	11101010	B_2	B_3	3	10	(P)=1ならば(PC)←<B_3><B_2> (P)≠1ならば(PC)←(PC)+3
JP	11110010	B_2	B_3	3	10	(S)=0ならば(PC)←<B_3><B_2> (S)≠1ならば(PC)←(PC)+3
JM	11111010	B_2	B_3	3	10	(S)=1ならば(PC)←<B_3><B_2> (S)≠1ならば(PC)←(PC)+3
CALL	11001101	B_2	B_3	5	17	SPで指定されるプッシュダ 〔SP−1〕〔SP−2〕←PCのスタックにPCの内容を (SP)←(SP)−2　転送しSPを−2にする。 (PC)←<B_3><B_2>　<B_2><B_3>で指定するメモリ番地へジャンプする。
CNZ	11001100	B_2	B_3	5/3	17/11	(Z)=0ならば　CALLと同じ動作 (Z)≠0ならば　(PC)←(PC)+3
CZ	11001100	B_2	B_3	5/3	17/11	(Z)=1ならば　CALLと同じ動作 (Z)≠1ならば　(PC)←(PC)+3
CNC	11010100	B_2	B_3	5/3	17/11	(C)=0ならば　CALLと同じ動作 (C)≠0ならば　(PC)←(PC)+3
CC	11011100	B_2	B_3	5/3	17/11	(C)=1ならば　CALLと同じ動作 (C)≠1ならば　(PC)←(PC)+3
CPO	11100100	B_2	B_3	5/3	17/11	(P)=0ならば　CALLと同じ動作 (P)≠0ならば　(PC)←(PC)+3
CPE	11101100	B_2	B_3	5/3	17/11	(P)=1ならば　CALLと同じ動作 (P)≠1ならば　(PC)←(PC)+3
CP	11110100	B_2	B_3	5/3	17/11	(S)=0ならば　CALLと同じ動作 (S)≠0ならば　(PC)←(PC)+3
CM	11111100	B_2	B_3	5/3	17/11	(S)=1ならば　CALLと同じ動作 (S)≠1ならば　(PC)←(PC)+3

ニーモニック	命令コード 1STバイト	2Nd	3rd	サイクル数	ステート数	オペレーション
RET	11001001			3	10	(PC)←〔SP〕〔SP+1〕 プッシュダウンスタックにストアされていた最後の値によって指定されるメモリ番地の命令へ戻る (SP)←(SP)+2
RNZ	11000000			3/1	11/5	Z=0 ならば RETと同じ動作 Z≠0 ならば (PC)←(PC)+1
RZ	11001000			3/1	11/5	Z=1 ならば RETと同じ動作 Z≠1 ならば (PC)←(PC)+1
RMC	11010000			3/1	11/5	C=0 ならば RETと同じ動作 C≠0 ならば (PC)←(PC)+1
RC	11011000			3/1	11/5	C=1 ならば RETと同じ動作 C≠1 ならば (PC)←(PC)+1
RPO	11100000			3/1	11/5	P=0 ならば RETと同じ動作 P≠0 ならば (PC)←(PC)+1
RPE	11101000			3/1	11/5	P=1 ならば RETと同じ動作 P≠1 ならば (PC)←(PC)+1
RP	11110000			3/1	11/5	S=0 ならば RETと同じ動作 S≠0 ならば (PC)←(PC)+1
RM	11111000			3/1	11/5	S=1 ならば RETと同じ動作 S≠1 ならば (PC)←(PC)+1
PCHL	11101001			1	5	(PC)←(H)(L) レジスタ対HLの内容により指定されるメモリ番地の命令へジャンプ
SPHL	11111001			1	5	(SP)←(H)(L) レジスタ対HLの内容をSPに転送する。
PUSHB	11000101			1	11	〔SP−1〕←(B) レジスタ対BCの内容をSPで指定するプッシュダウンスタックにストアする。 〔SP−2〕←(C) (SP)←(SP)−2
PUSHD	11010101			3	11	〔SP−1〕←(D) レジスタ対DEの内容をSPで指定する。プッシュダウンスタックにストアする。 〔SP−2〕←(E) (SP)←(SP)−2
PUSHH	11100101			3	11	〔SP−1〕←(H) レジスタ対HLの内容をSPで指示するプッシュダウンスタックにストアする。 〔SP−2〕←(L) (SP)←(SP)−2
PUSH PSW	11110101			3	11	〔SP−1〕←(A) レジスタ(Acc)の内容とフラグF/Fの内容をSPで指示するプッシュダウンスタックにストアする 〔SP−2〕←(F) (SP)←(SP)−2

ニーモニック	命令コード 1STバイト	2nd	3rd	サイクル数	ステート数	オペレーション	
						D_0 ——— Carry D_1 ——— 1 D_2 ——— Parfty D_3 ——— 0 D_4 ——— Cy D_7 ——— 0 D_6 ——— Zero D_7 ——— Sign	
POPB	11000001			3	10	(C)←〔SP〕 (B)←〔SP+1〕 (SP)←(SP)+2	SPで指定されるプッシュダウンスタックの最終値をスタ対BCにストアするSPの内容は+2される
POPD	11010001			3	10	(E)←〔SP〕 ( )←〔SP+1〕 (SP)←(SP)+2	SPで指定されるプッシュダウンスタックの最終値をレジスタ対DEにリストアする。SPの内容は+2される
POPH	11100001			3	10	(L)←〔SP〕 (H)←(SP+1) (SP)←(SP)+2	SPで指定されるプッシュダウンスタックの最終値をレジスタ対HLにリストアするSPの内容は+2される
POP SW	11110001			3	10	(F)←〔SP〕 (A)←〔SP+1〕 (SP)←(SP)+2	SPで指定されるプッシュダウンスタックの最終値をレジスタAとフラグF／Fにリストアするの内容+2される
RST	11AAA111			3	11	〔SP−1〕:〔SP−2〕←(PC) (SP)←(SP)−2 (PC)←	PSの内容をSPで指定するプッシュタウンスタックに転送し、SPの内容を−2して



第２－１６図　μCON８の基本タイミング

タバスにデータが送り出され次のマシンサイクルまで保持される。マシンサイクルがＲＥＡＤモードではＤＢＩＮが出され、T_3ステートまで継続される。ＴＷは、ＲＥＡＤＹ信号が到着しない時のステートで到着するとステートはＴ３に移る。Ｔ３ではＷＲＩＴＥモードのときはＷＲ信号を出し、ＲＥＡＤモードでは、命令又はデータを取り込む。Ｔ４、Ｔ５は実行であるが命令やデータをさらに取込む必要があるときは、次のマシンサイクルに移る。

ぐ－ＣＯＭ８は、出力６種、入力４種の制御ラインを持っている。これらの出力制御信号は外部デバイス及びメモリの制御に使われ、先に説明したＲＥＡＤＹ信号をはじめとする入力制御信号は、プロセッサのシーケンスを変えるのに用いられる。

命令については、第２－５表にその概略を示す。又オペレーション内容の説明に使われている代表的記号と記法は次のとおりである。

① r；スクラッチパッドレジスタ

② M；外部メモリ

③ SP；スタックポインタ

④ F；フラグ

⑤ PC；プログラムカウンタ

⑥ ＜B_2＞＜B_3＞；命令の２バイト目、３バイト目

⑦ 〔 〕；メモリ番地、例えば〔SP〕又はスタックポインタの示すメモリ番地

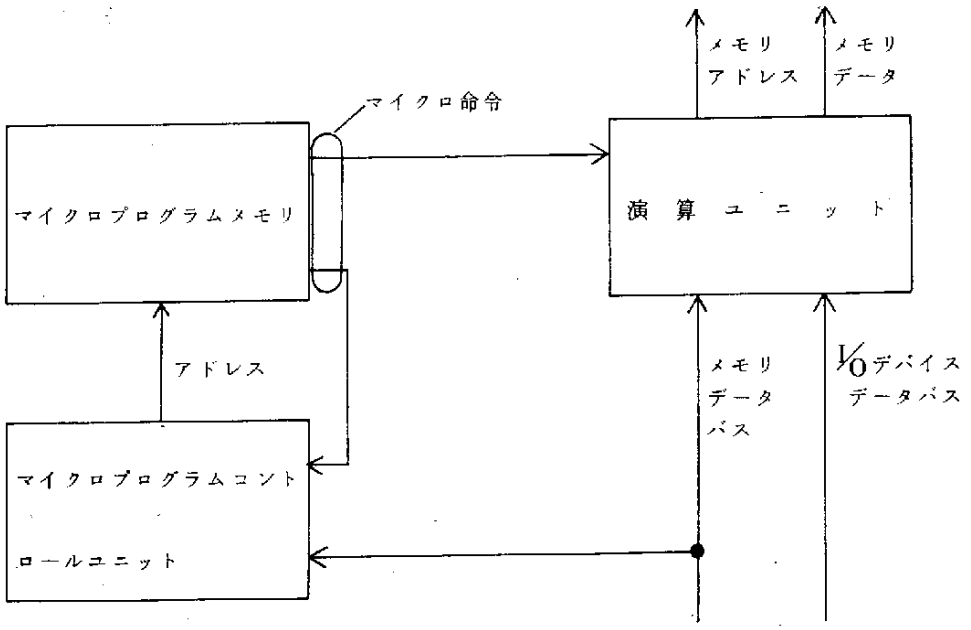
⑧ （ ）；内容；例えば(A)は、アキュームレータの内容

⑨ レジスタペア；２つのレジスタを結合して16 ビットのレジスタにしたもの

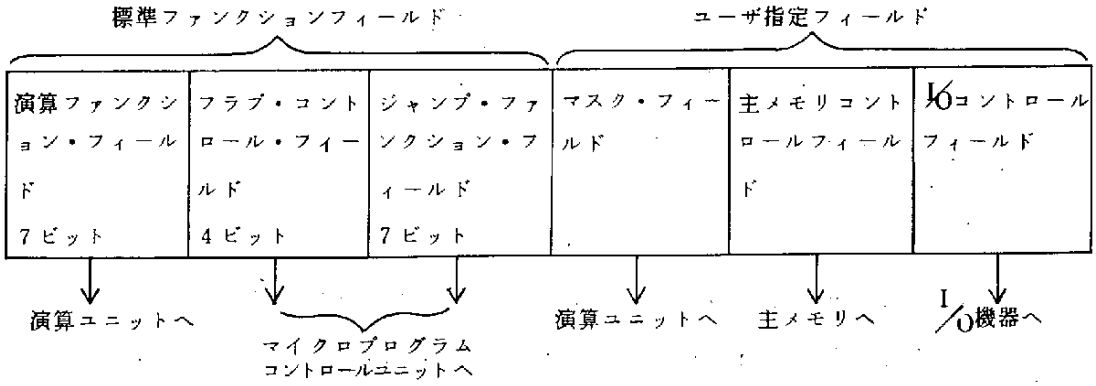
また第２－３表は８００８の命令対称表である。

(3) ＩＮＴＥＩ ３０００

この機種を示す理由は２つある。第１は、マイクロプログラム計算



第２－１７図　Fnetl3000マイクロプロセッサの構成



第２—１８図　マイクロ命令の例

機であるということ、第2はビットスライスタイプであるという点である。

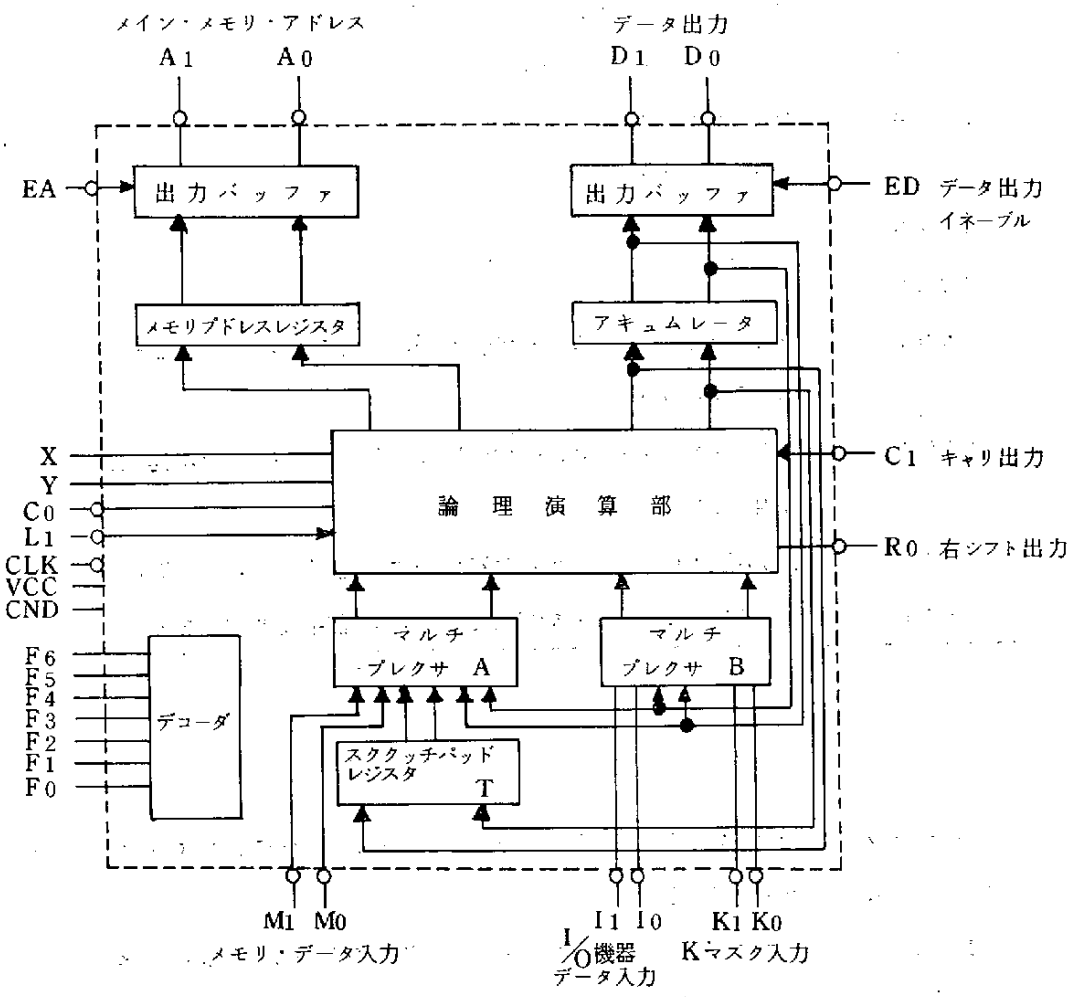
3000シリーズと呼ばれるマイクロプロセッサは、第2－5表に示すファミリチップで構成され、各チップはショットキー・バイポーラで構成されている。この種のマイクロプロセッサの特徴の一つは、システム構成のフレキシビリティにあるが、ここでは説明の都合上一つの標準構成を示す。第2－17図は、その構成を、第2－18図は、マイクロ命令のフォーマット例を示す。　このシステムの基本的動作は、先ずマイクロプログラムコントロールユニットからマイクロプログラムメモリにアドレスが送られ、マイクロ命令が読出される。マイクロ命令の一部（演算ファンクションフィールド）は演算ユニットに送られ、演算の種類を指定する。またマイクロ命令のジャンプファンクションフィールドは、マイクロプログラムコントロールユニットに送られ、次に実行するマイクロ命令の番地決定に使われる。メモリデータバスの一部も又次に実行するマイクロ命令アドレスの決定に使われることがある。フラグコントロールフィールドは、マイクロプログラムラムコントロールユニットに入り、演算ユニットのキャリーセーブ等フラグの制御に使われる。マイクロ命令の以上3フィールドは、標準的なものであるが、システム設計者は任意にマイクロ命令フィールドを増やし、各種制御に使うことができる。次に演算ユニットについて説明する。3002は2,ビット単位でレジスタ操作や演算を行うチップでN個接続することにより、2Nビット語の並列処理が可能となる。第2－19図はそのブロック図を示す。端子としては、マスク入力バス（K）、主メモリからデータを入力する（M）、入出力機器からデータを入力する（I）、主メモリへのアドレスを出力する（外部メモリや機器へデータを出力する（D）、マイクロ命令を入力する（F）の6種がある。チップ内にはR_0～R_8及びTの11個のスク

第２－５表　　INTEL　３０００のファミリ．チップ

３００１	マイクロ命令シーケンス制御ユニット（MCU）
３００２	演算ユニット（CPE：セントラル．プロセッシング．エレメント）
３００３	ルック．アヘッド．キャリ
３２１４	割込制御ユニット
３２１２	多モード、ラッチ、バッファ
３２１６ ３２２６	双方向性バス．ドライバ
３６０１	２５６×４ビット、ヒューズ形P—ROM（プログラマブル読み出し専用メモリ）
３６０４	５１２×８ビット、ヒューズ形P—ROM（プログラマブル読み出専用メモリ）
３３０１	２５６×４ビット、マスクROM（マスク読み出し専用メモリ）
３３０４	５１２×８ビット、マスクROM（マスク読み出し専用メモリ）

ラッチバッドレジマスタと１個のアキュームレータを内蔵しており各２ビット構成となっている。

Fに入るマイクロ命令は、下位４ビットが演算の対象となるレジスタを上位３位ビットで演算の種類を指定する。第２－６表は、この指定を示しており、第２－　表は、これらから決定されるマイクロファンクションを示している。　例えばファンクショングループ０でレジスタグループ１では、「Kバスのデータによりマスクされたアキュームレータの内容と、R_0及びキャリ入力が加算され、結果がR_0とアキュームレータに入る」ことを示している。



第２－１９図　３００２のブロック図

第 2 － 6 表　　演算ユニット（3002）Fバスのファンクッション指定

（a）レジスタ・グループ

F_3	F_2	F_1	F_0	指定レジスタ	レジスタ・グループ
0	0	0	0	R_0	
0	0	0	1	R_1	レジスタ・グループI
0	0	1	0	R_2	
0	0	1	1	R_3	
0	1	0	0	R_4	
0	1	0	1	R_5	
0	1	1	0	R_6	
0	1	1	1	R_7	
1	0	0	0	R_8	
1	0	0	1	R_9	
1	1	0	0	T	
1	1	0	1	AC	
1	0	1	0	T	レジスタ・グループII
1	0	1	1	AC	
1	1	1	0	T	レジスタ・グループIII
1	1	1	1	AC	

（注）　この表で使われている記号の意味はつぎのとおり

Rn　n＝0～9：W番目スクラッチパッド・レジスタ

T：Tスクラッチパッド・レジスタ

AC：アキュームレータ

（b）ファンクション・グループ

F_6	F_5	F_4	ファンクション・グループ
0	0	0	ファンクション・グループ 0
0	0	1	ファンクション・グループ 1
0	1	0	ファンクション・グループ 2
0	1	1	ファンクション・グループ 3
1	0	0	ファンクション・グループ 4
1	0	1	ファンクション・グループ 5
1	1	0	ファンクション・グループ 6
1	1	1	ファンクション・グループ 7

第２－７表　　マイクロファンクションの内容

ファッション・グループ	レジスタ・グループ	マイクロファンクッション
0	Ⅰ	Rn＋(AC∧K)＋CI→Rn．AC
	Ⅱ	M＋(AC∧K)＋CI→AT
	Ⅲ	ATL∧(IL∧KL)→RO LI∧［(IH∧KH)∧ATH］→ ATH［ATL∧(IL∧KL)］∧［ATH∧(IH∧KH)］→ ATL
1	Ⅰ	KVR→MAR　Rn＋K＋CI→Rn
	Ⅱ	KVM→MAR　M＋K＋CI→AT
	Ⅲ	(ATVK)＋(ATVK)＋CI→AT
2	Ⅰ	(AC∧K)－1＋CI→Rn
	Ⅱ	(AC∧K)－1＋CI→AT
	Ⅲ	(I∧K)－1＋CI→AT
3	Ⅰ	Rn＋(AC∧K)＋CI→Rn
	Ⅱ	M＋(AC∧K)＋CI→AT
	Ⅲ	AT＋(I∧K)＋CI→AT
4	Ⅰ	CIV(Rn∧AC∧K)→CO　Rn∧(AC∧K)→Rn
	Ⅱ	CIV(M∧AC∧K)→CO　M∧(AC∧K)→AT
	Ⅲ	CIV(AT∧I∧K)→CO　AT∧(I∧K)→AT
5	Ⅰ	CIV(Rn∧K)→CO　K∧Rn→Rn
	Ⅱ	CIV(M∧K)→CO　K∧M→AT
	Ⅲ	CIV(AT∧K)→CO　K∧AT→AT
6	Ⅰ	CIV(AC∧K)→CO　RnV(AC∧K)→Rn
	Ⅱ	CIV(AC∧K)→CO　MV(AC∧K)→AT
	Ⅲ	CIV(I∧K)→CO　ATV(I∧K)→AT
7	Ⅰ	CIV(Rn∧AC∧K)→CO　Rn⊕(AC∧K)→Rn
	Ⅱ	CIV(M∧AC∧K)→CO　M⊕(AC∧K)→AT
	Ⅲ	CIV(AT∧I∧K)→CO　AT⊕(IVK)→AT

（注）この表で使われている記号の意味はつぎのとおり

I，K，M．：I．KおよびMバスのデータ　MAR：メモリ・アドレス・レジスタの内容

CI，LI：キャリ入力，右シフトの入力データ　L，H：下位ビットまたは上位ビットを示す

CO，RO：キャリ入力，右シフトの出力データ　＋：2の補数方式加算

Rn：nレジタスの内容，レジタスグループ1ではアキュームレータ，TLも含まれる。　－：2の補数方式減算（00…1を引くことは111…11を加算することに相当する）

AC：アキュームレータの内容　∧：AND　V：OR

AT：Tレジスタまたはアキュームレータの内容（F_3～F_6で指定される）　⊕：排他的NOR

→：～に値を入れる

2.3　半導体（IC）メモリとそのインターフェース

マイクロコンピュータの構成要素として重要なものに、マイクロプロセッサと並びメモリが有る。従来電子計算機のメモリとしては、コアメモリが主流であったが、近年半導体技術の進速な進歩により半導体メモリがこれに取って変わる勢いを見せている。半導体メモリの特徴としては、小型、低電力性、高速性など等を上げることが出来るが、これらはいずれもマイクロコンピュータの特徴と合致するものである。従ってマイクロコンピュータのメモリとしては、半導体メモリが主役となっている。以下では、半導体メモリの紹介と、それらの接続について述べる。

半導体メモリは、素子の性質という面から分類すると、バイポーラ・トランジスタを基礎にしたバイポーラICメモリとMOS電界効果トランジスタを基礎としたMOS－ICメモリがある。それらの使用領域区分を考えてみると、バイポーラICは、高速動作が可能であるが、消費電力が大きく高集積化が難しいという性質から小容量メモリとして主に用いられ、MOS－IC は、動作速度が遅いが高集積を達成し易いため大容量メモリに用いられる傾向にある。第２－８表は、ICメモリ（RAM；後述）の性能例を示しており、第２－２０図は、集積度とアクセス時間の関係を示している。ICメモリの集積度に関しては、MOS－ICでは現在１６０００ビット以上のROMや４０９６ビットのRANが、バイポーラICでは１０２４ビットのRANが実用化されている。しかし、半導体技術の進歩は留ることを知らず、研究は１００Kビット以上を指向し進められている。第２－２１図に集積度増大の年次経過を示しているが、このような増大傾向は今後も続くというのが、専門家の間でのほぼ一致した見方である。

IC メモリを機能面から分類すると、リード・ライト・メモリとリード・オンリ・メモリの２つに大別できる。この２種類のちがいは、その名が示すとおり、リード・ライト・メモリは、メモリの内容を任意に読み書き出来るメモリであるのに対し、リード・オンリー・メモリは、一度メモリ

にデータが書き込まれると以後は、そのデータを読出しデータとしてのみ用いるもので、電源を落してもその内容は保存される。またＩＣメモリを、メモリ内のデータへのアクセス法という面から分類すると、メモリ内のどのアドレスにも任意にアクセス出来るランダムアクセスメモリと、例えば磁気ディスクなどのようなアクセス方法すなわち、メモリ内のデータに一定の順序でしかアクセスできないシーケンシャルアクセスメモリがある。シーケンシャルアクセスメモリのＩＣとしては、シフトレジスタが代表であり、これを磁気ディスク替わりに用いようという研究も進められている。以上ＩＣメモリの機能を２つの側面から分類して来たが、一般にＩＣメモリは、これらに従って次のように分類されている。

(イ) RAM（ランダムアクセスメモリ）……ランダムアクセス型のリード・ライト・メモリ。

(ロ) ROM（リード・オンリ・メモリ）……ランダムアクセス型のリード・オンリ・メモリ

(ハ) SR（シフトレジスタ）……シーケンシャルアクセス型のリード・ライト・メモリ。

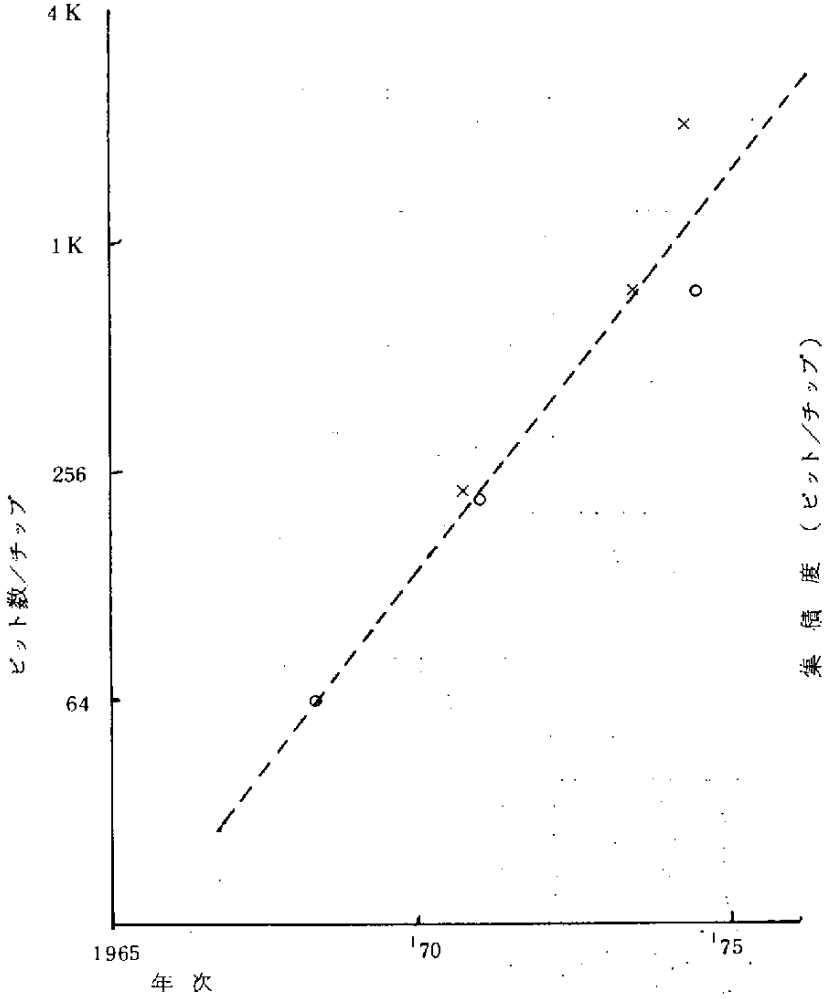
この節では、先ずこれらの３種類のＩＣメモリの概要を述べ、次にこれらのＩＣ素子を結合しメモリを構成する方法と、マイクロプロセッサとの接続について述べる。

第２－８表　ＩＣメモリの性能例（ＲＡＭ）

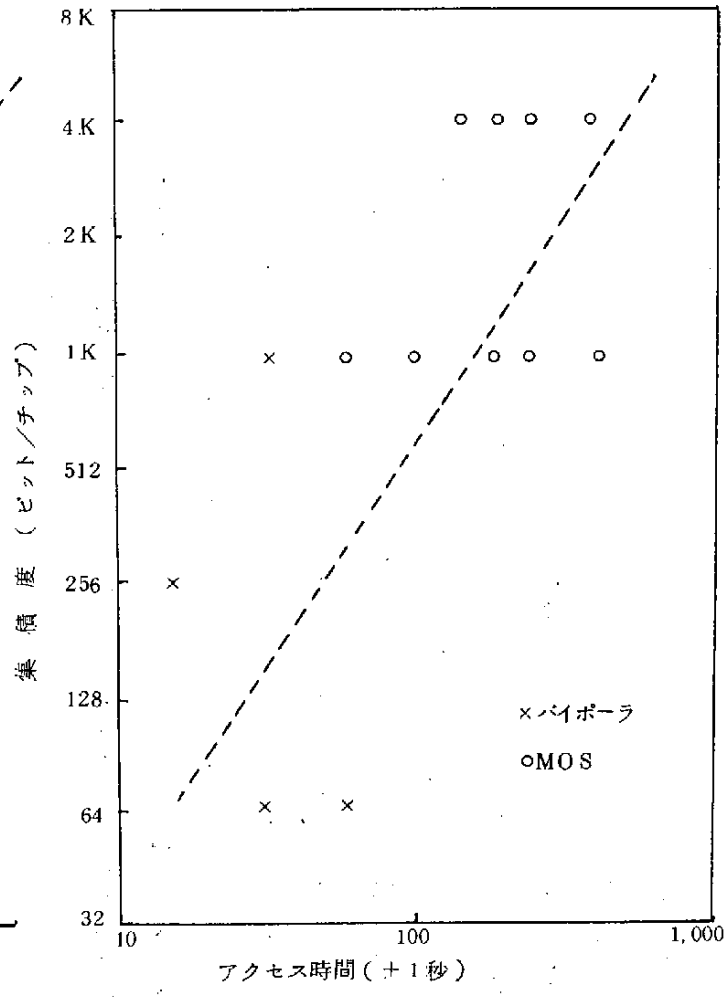
	集積度 (ビット/チップ)	デバイスの種類	アクセス時間 (マイクロ秒)	サイクル時間 (マイクロ秒)	消費電力 アクティブ/スタンドバイ (mw/チップ)	レジスタ
バイポーラ	256	ECL/TTL	0.02～0.05	0.02～0.05	350/350	スクラッチパッド
	1,024	ECL/TTL	0.06～0.09	0.06～0.09	500/500	キャッシュ
ＭＯＳ	1,024	nチャンネル	0.06	0.18	450/60	キャッシュ メイン
	1,024	Dチャンネル	0.3	0.6	450/60	メイン
	4,096	nチャンネル	0.2～0.35	0.4～0.7	350/30	メイン
	1,024	nチャンネル スタティック	0.5	0.5	350/90	小容量システム 周辺機器
	1,024	CMOS	0.6	0.6	30/0.3 (μw)	周辺機器等

2.3.1　ＲＡＭ

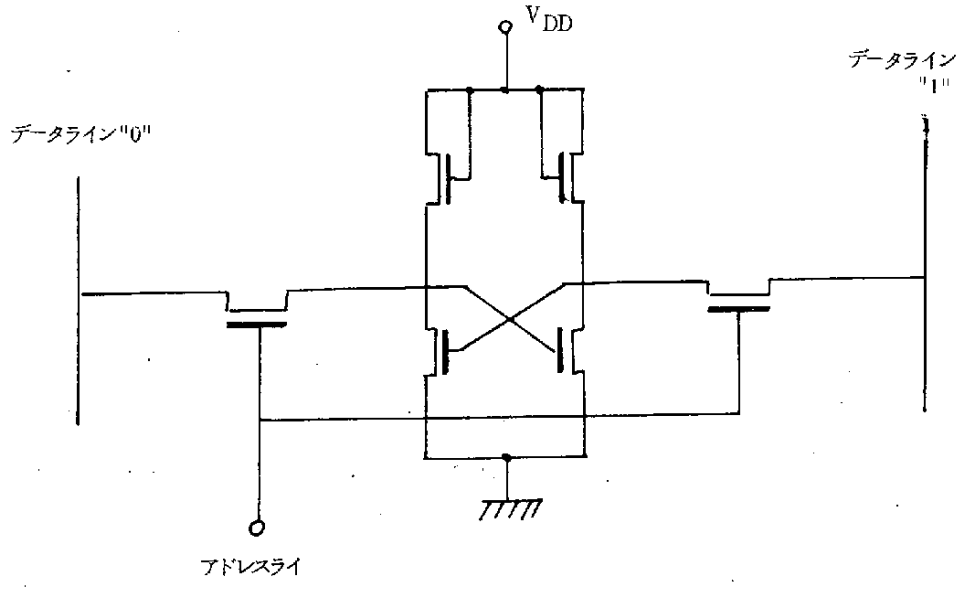
ＭＯＳ－ＲＡＭは、動作上の特性、特に情報の保持方式の面から分類すると、スタティック型とダイナミック型に分かれる。スタティック型は、第２－２２図に示されるように、インバータをたすきがけ接続し、メモリセル内部に帰還回路をつくることにより、いったん書込まれた情報は電源が切れない限り保持される仕組になっている。一方、ダイナミック型は、第２－２３図に示すように、情報保持にＭＯＳトランジスタ回路の寄生容量を用いているため、情報が時間と共に失なわれる。一般にこの時間は数ミリ秒であるため、１～２ミリ秒ごとにリフレッシュと呼ばれる情報再生報作が必要となる。これからわかる様に、ダイナミック型ではリフレッシュのための特殊回路が必要となる欠点があるが、動作速度が速い、消費電力が少ない、１ビット当りのエレメント数が少ないなどの利点があり、一般には大容量メモリと



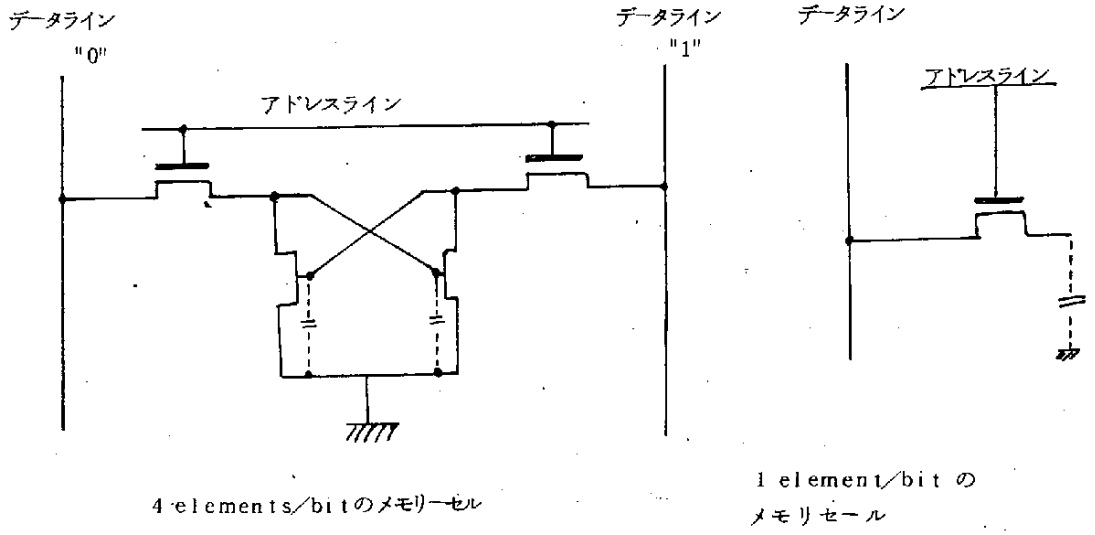
第２－２１図　集積度の年次経過



第２－２０図　集積度とアクセス時間の関係



第２－２２図　スタティック型RAMメモリーセルの例



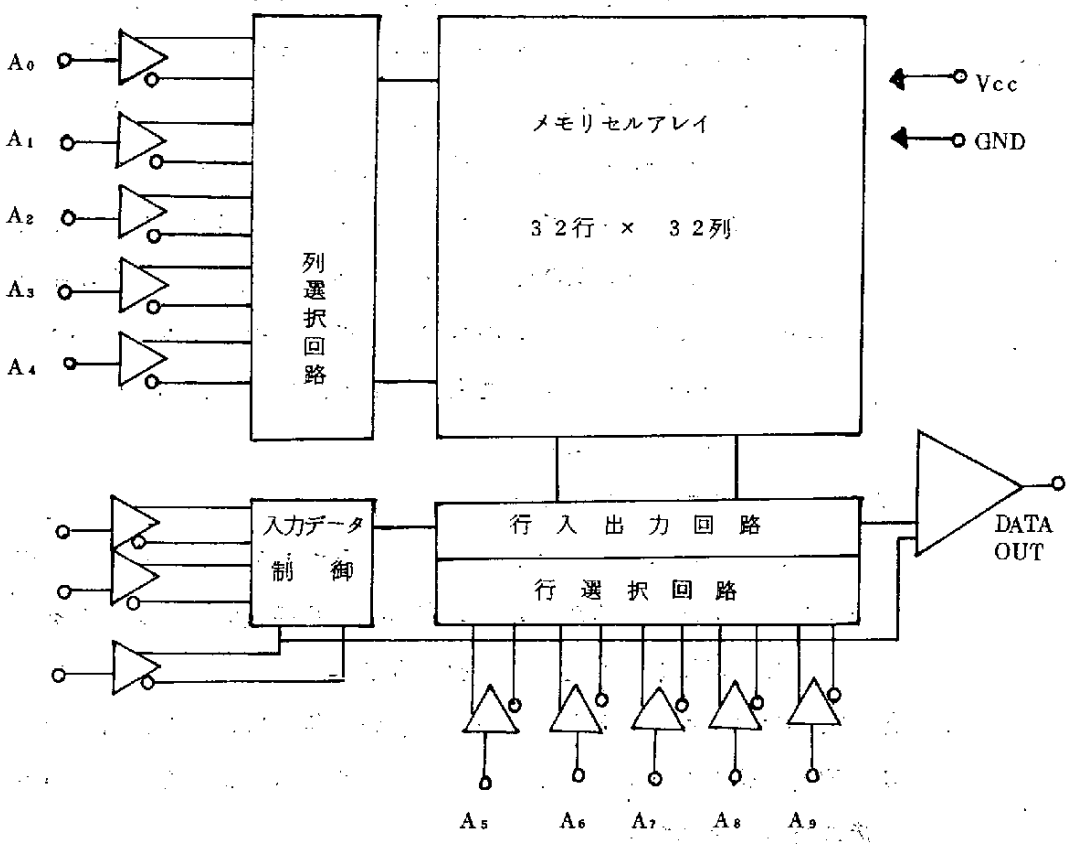
第２－２３図　ダイナミック型RAMメモリセルの例

して用いられる。これに対しスタテック型は、ほとんどのものがTTLやDTL のバイポーラ回路と直接接続ができリフレッシュ回路が不要なため取扱が便利である。しかし動作速度、消費電力、価格、などの点でダイナミック型に及ばず、一般には、小容量メモリに用いられている。以上２種類のMOS－RAMの性質を述べて来たが、次にバイポーラRAM との性能の差について少し、触れておくことにする。バイポーラＲＡＭの利点としては、動作速度が速いことが上げられ、一方MOS－RAMの利点としては、大集積化が可能、歩留りが良く値段が安い、低消費電力等があげられ、多くの点でバイポーラRAM より優っています。又速度の面でも研究が進み、現在では、アクセス時間数＋ナノ秒というものもあり、MOS－RAM でかなり広い応用分野に対処できる。

以上は、RAN の基本記憶単位となるメモリセル自体に関する記述であったが、実際のＩＣメモリは、これらメモリセルとそれらを動作させるための周辺回路から構成されている。そこで次にＩＣメモリチップの構成とその基本動作について述べる。RAM チップの１語のビット数は、1、４、８などと色々あるが、基本構成は大差が無く、ここでは１０２４語（１語１ビット）のスタティック、MOS－RAMの内部等価回路について説明する。第２－２４図がチップのブロック図である。図中の入力端子ＣＥ（チップ・エネイブル）には、このチップを可動状態にするための信号が入る。即ち多くのチップがある場合、その中のどのチップにアクセスするかを決定するのがこの信号であり、ＣＳ（チップ・セレクト）と名付けられている。この信号でチップが選択されると、次に必要な情報は、アクセスしたい語が１０２４語のうちのどれかを示すアドレスであるが、これは端子Ａ$_0$～Ａ$_9$で与える。もう一つ大切な信号としては、アドレスされた番地に書込むのか、そこから読出すのかを指定する信号で、これは端子R/Wの〝１〟、〝０〟で決

まる。又書込むデータは、DATA　1N端子に与え、読出されたデータはDATA、OUT に出て来る。

以上が端子の説明で、次に内部ブロックに移ると、先ず各端子に付いているバッファであるが、これらは、TTL、DTLレベルからMOSレベルに信号を変換するものである。列選択回路と行選択回路は、与えられたアドレスをデコードし、アクセスする語を選び出すための信号を作り出すものであり、例えば読出しでは、列選択で32列あるもののうち1つを選ぶことにより、ある列32ビットが行入出力回路に読出され、行選択回路でそのうちの1ビットが選ばれるといった具合である。



第2－24　RAMチップの構成

2.3.2 ROM

ROMは、読出し専用メモリであるが、読出すための情報は何らかの方法でセットしなければならない。ROMは一般に、この読出し情報のセットの仕方によって次の2つに分類される。

① メーカプログラムドROM

② ユーザプログラマブルROM

①は、あらかじめ用意した読出データを、ROMメーカがマスクパターンに変換し、それをROMとして作り上げるもので、マスクプログラブルROMとかマスクROMと呼ばれている。このタイプのROMは、メーカがROMをマスクパターンから作るため、マスクパターンの作成に費用がかかること、一度マスクパターンを作ってしまうと内容の変更が出来ないこと等の性質が有り、内容変更が無く、かつ大量使用するメモリに向いている。

一方、②は、ROMを製造する段階が情報の書込みと独立しており、ユーザがROMを買って来て、それに情報を自由に書き込むことが可能である。このタイプのROMは、フィールドプログラブルROMとかPROMと呼ばれている。この種のものは、発中等、書込み内容に変更が発生しやすい場合に便利である。PROMは、書換えの方法により、次の4つの種類がある。

ⓐ 電気的に書換え可能なもの。

電気的に、書込み、読出しが可能であるがRAMとの違いは、電源を切っても内容が消えない点、書込みに時間がかかる点である。

ⓑ 一度だけ電気的に書込みが可能なもの。

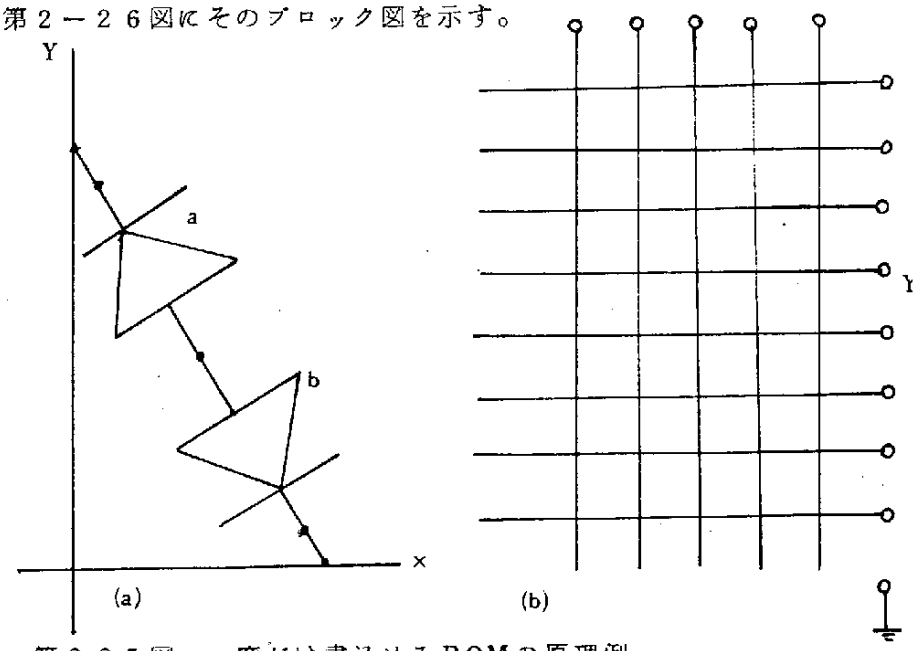
ⓒ 電気的にメモリの内容を消去し、しかる後に情報を書込むもの。

ⓓ X線、紫外線、電子線などによりメモリの内容を消去し、しかる後に情報を書込むもの。

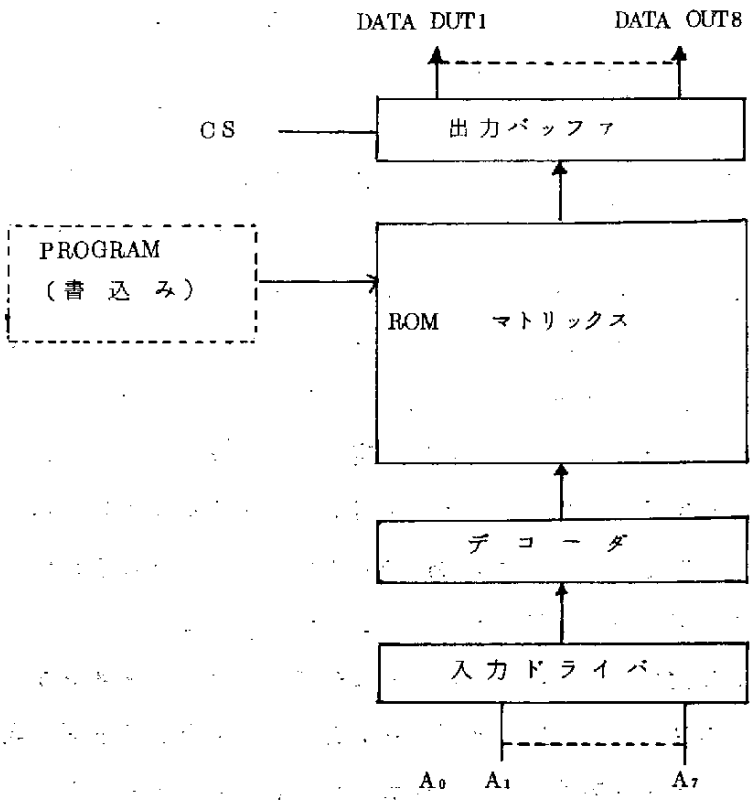
これらのうち現在広く使われているのは、ⓑとⓓの紫外線消去式の

ものであり、これらについて少し触れる。ⓑの例としては、第２－２５(a)に示す様な、ダイオードを２つ背中合わせに結合したものを第２－２５(b)に示すマトリクスのクロスポイントに配置したものがある。故に初期状態ではクロスポイントは、逆方向のダイオードで非導通になっている。そこでダイオードに逆方向降服電圧以上の電圧をパルス的に加えることにより逆方向のダイオードを破壊し短絡状態にすることで、データの書込みが成される。ⓓは、ＩＮＴＥＬが発表したもので、何度でも書換えが出来るという利点から広く使われている。このメモリへの書込みは、４８ボルト程度の電圧を数分間かけることにより行なわれ、一度書込まれた情報は、電源を切っても失なわれない。この書込まれた内容の消去はPROM の真中に石英がラスの窓が付いており、ここに紫外線を照射することにより行なわれる。

最後にROM チップの構成について述べる。リード・オンリーという性質からRAN チップのサブセットとなっており、チップの端子としては、アドレス端子、データ読出し端子、チップセレクトがある。第２－２６図にそのブロック図を示す。



第2.２５図　一度だけ書込めるROMの原理例



第 2.－2 6　ROM、PROMチップの構成

2.3.3　シーケンシャルアクセスメモリ

シフトレジスタあるいはディジタルディレーラインと呼ばれるものがこれである。普通シフトレジスタと言うと8ビットとか16ビットといった小規模のものを連想しがちであるが、LSI 化により大容量のものが可能となって来た今日、磁気ディスクや磁気ドラムに変わるメモリとして研究が進められている。このメモリは、ランダムアクセスメモリとちがい、記憶性報をシフトレジスタ内をぐるぐると高速で巡還させておき、そのループの一部にアクセスポートを付けデータの読み書きを行なう。そのためアクセスタイムが大きくなり高速用には

向かないが、次のような点でMOS－ICメモリに向いているため、今後のICメモリの一つの方向になると思われる。

① MOS－FET回路の高インピーダンス特性を利用して、回路の浮遊容量が情報の一時記憶手段として使える。

② DCオフセットの無い両方向ゲートが実現出来る。

③ RAM、ROMに必要なアドレスデコータ等の回路が不要となる。

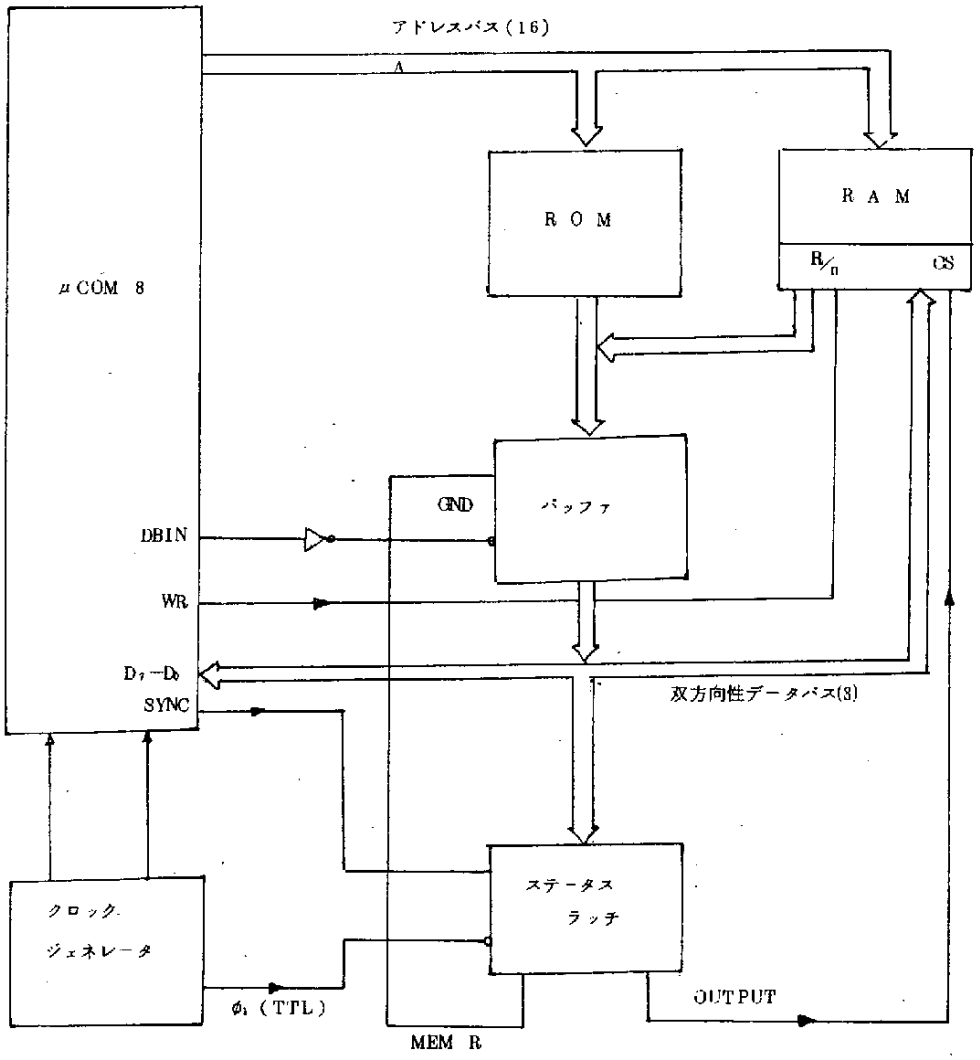
④ シフトレジスタ各段の回路と、それらの結合回路は比較的小さく出来る。

2.3.4 マイクロプロセッサとメモリの接続

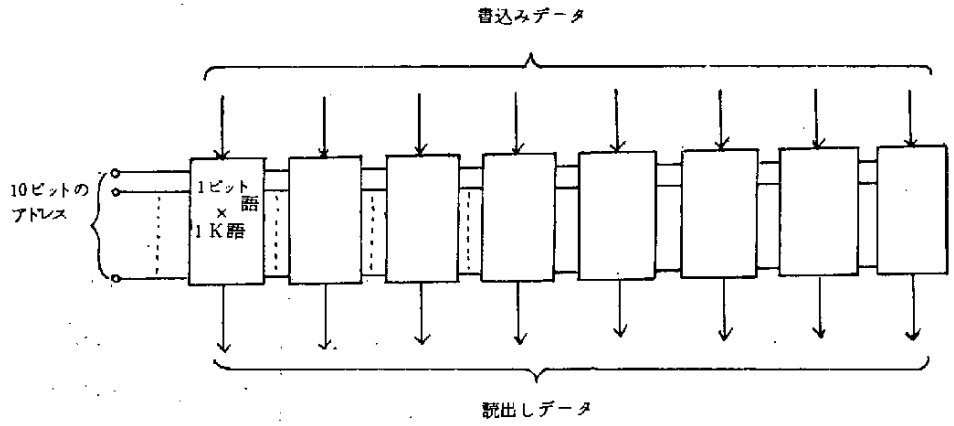
マイクロプロセッサとICメモリの接続をμCOM8の基本構成例をもとに説明する。第2－27図がその構成を示している。μCOM8の動作については第2－2項で述べたが、メモリの読み書きに関係する点を整理してみると次のようになる。マイクロプロセッサは、アドレス転送用に16本のピン（A_0～A_{15}）が接続されている。データの入出力用には8本のピン（D_0～D_7）があり、これにデータバス、（8ビット）が接続されている。しかしこのバスは、入力と出力両方に用いられるため、このどちらかを区別する情報が必要となる。その区別に用いられるのが〝DBIN〟と〝$\overline{WR}$〟ピンである。又各メモリサイクルの先頭（SYNC信号時）では、出力データの準備完了、読込み準備完了等といったマイクロプロセッサの状態が入出力バスに出され、ステータスラッチにラッチされる仕組になっており、メモリとの読み書きは、これらの信号により制御されている。即ち、メモリへの書き込みは、プロセッサがアドレスバスにアドレスを出力し、データバスにアキュームレータの内容が出力されるマシンサイクル（ステータスラッチのOUT、DUTピンがオン）で、ライト信号（$\overline{WR}$）が出された時であり、読み出しは、プロセッサが、データバスをメモリ読出しに使うマシンサイクル（ステータスラッチのMEMRがオン）でDBIN

信号が出た時に行なわれる。

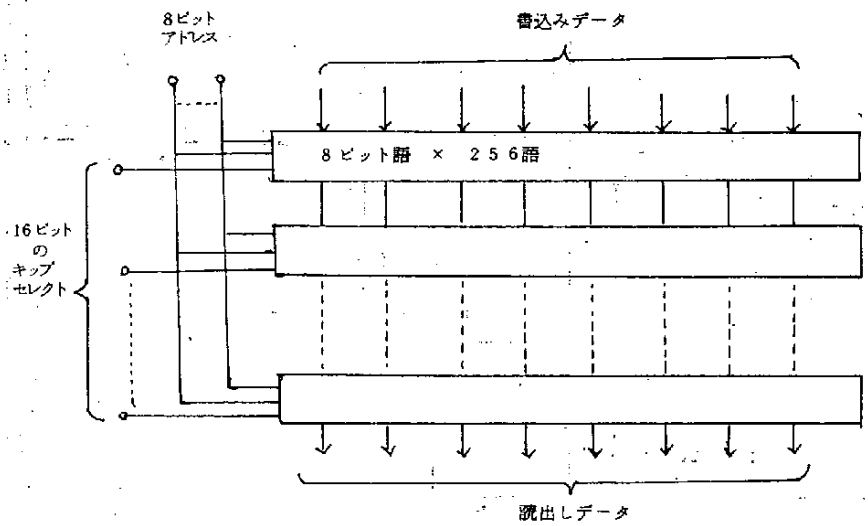
以上がマイクロプロセッサとメモリの接続であるが、このシステムで使うようなメモリを先に設明したＩＣメモリチップで構成しようとすると問題点が２つある。第１は、このマイクロプロセッサでは、１語を８ビットとして扱っているのに対し、ＩＣメモリは、１語が１、４ビットというものもあり、ビット巾を増やすことをしなければならない点、第２の点は、１６ビットのアドレス（６４ＫＢ）をフルに使いたい場合、１チップでそのように大きな容量のものが無いため、チップ数を増やすことが必要となる点である。第１の点に関してはビット数の少ないＩＣメモリをマイクロプロセッサの要求するビット数に足るだけ並列にならべて対処している。第２－２８図は、１語１ビットのメモリを８個並列にならべ、８ビット語１Ｋ語のメモリを構成した例を示している。又第２の点については、メモリを直列に並べ、チップセレクト端子に、アドレスの上位ビットをデコードして入れる方法を用いる。第２－２９図は、８ビット１語で２５６語のチップを用いて４Ｋバイトのメモリを構成した例を示している。実際のメモリ構成では、これら両方の手段を組合わせて用いなければならない場合が多い。第２－３０図にＩＮＴＥＬ８００８標準システムで用いたメモリ構成の例を示す。



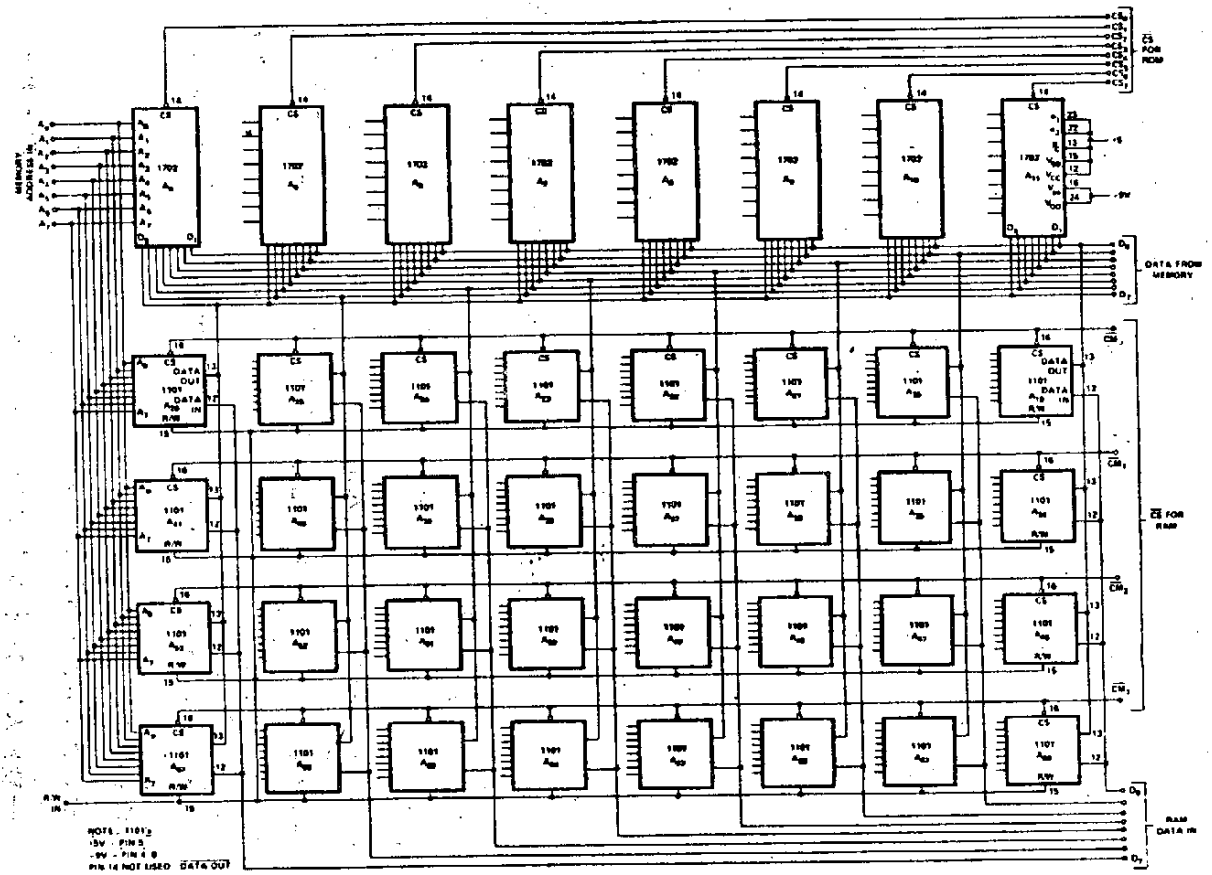
第２－２７図　μCOM8とメモリの接続例



第2－28図　1ビット語×1024語チップを用いた
8ビット語×1024語メモリの構成



第2－29図　8ビット語×256語チップを用いた
8ビット語×4K語メモリの構成



第２－３０図　８００８用メモリ構成例

2.4　入出力インタフェース

一般の電子計算機やミニコンに比べ、マイクロプロセッサの入出力機能は低く、マイクロコンピュータを製作する場合には、ある程度のハードウェアとソフトウェアが必要となる。このことは、せっかくマイクロプロセッサが1又は少数のチップで構成出来ても、入出力装置とのインターフェース用回路の設計が難かしく、ハードウェア量もばかにならなくなる。反面、マイクロプロセッサをシステム内に組込む場合等には、インターフェースを外部の構成とマッチしたものに設計出来るということにもなる。最近は、上述の欠点をカバーするため、マイクロプロセッサメーカは、入出力装置とのインタフェースを実現し易い汎用入出力インターフェースチップであるとか、ある入出力装置専用の入出力インタフェースチップ等を、マイクロプロセッサのファミリとして準備する傾向にあり、やがてマイクロコンピュータの製作は、これらのチップ間の配線を行なうだけで完成するようになろう。

この節は、まずマイクロプロセッサの入出力インターフェースの基本的考え方を示し、次に最近マイクロプロセッサのファミリとして販売されているインターフェースサポート用チップを紹介する。

2.4.1.　マイクロプロセッサの入出力インタフェース

マイクロプロセッサの入出力の方法としては、第2.2項で紹介したように、大きく分けて、プログラムによる入出力と、ダイレクトメモリアクセス（DMA）の2種類である。以下では、これらについて詳細に説明する。

(1)　プログラムによる入出力のインターフェース

一般に入出力を行なう手順は、先ず入出力装置の状態をチェックし、装置が入出力動作可能であることが確認された段階でマイクロプロセッサは、データの入出力命令を出し、データの転送を行なう。ここで注意しなくてはいけないことは、データの入出力命令で、行なわれる

ことは、例えば入力命令であれば、指定された装置からアキュームレータにデータを取り込むことである。言いかえれば、入出力装置のあるレジスタからアキュームレータへのデータ転送にすぎない。そこで実際に入出力操作を行なおうとすれば、入出力装置のレジスタへデータを取込む操作であるとか、レジスタの内容を印字するとか、カセットテープ装置であれば、スタート、ストップ、巻戻しといったことを入出力装置に行なわせる制御指令も必要となる。

一方マイクロプロセッサの入出力命令はと見ると、INTEL8008では、入力用、出力用に1つづつ第2－9表(a)に示す命令があるだけであるし、μCOM8でも第2－9表(b)に示す2つの命令だけである。これだけで以上にあげたデータ入出力の他、装置の状態を調べたり、制御指令を与えたりするのは不可能なように思えるが、工夫しだいで色々な方法が可能となる。

マイクロコンピュータの2つの入出力命令で、種々の機能を実現する方法は、基本的に2つある。第1の方法は、命令の入出力機器指定部に、機能を割当てる方法である。例えば「入力命令の入出力機器指定部のMSBが〝I〟の時は、入出力装置の状態をアキュームレータに取込む」といった具合である。このことは言替れば、入出力装置のI機能（この例でいえば状態レジスタ）をIつの入出力動作として扱っていることになる。第2つ方法は、出力命令で送り出されるデータに、機能に割当てる方法である。この方法では、送り出された機能は、入出力装置制御部に送られ、入出装置制御に使われる。ここでタイプライタを用いた〝一語印字〟と〝一語読み取り〟の例を示す。先ず、先の第一の方法即ち命令の入出力機器指定部を利用して作るデータ入出力命令以外の命令としては、状態入力命令と、機能出力命令である。先の第2の方法すなわち機能出力命令で送り出されるデータに割当てる機能としては、第2－31図のように設定する。ここで、テ

レタイプの装置番号が0番で、印字機能が3番であると、アキュームレータには、〃00011000〃とセットレ、機能出力命令を出す。するとこれがテレタイプ制御装置に送られ、制御装置はテレタイプを、次にデータが送られて来たら印字する様にセットする。マイクロプロセッサでは、次に印字するデータをアキュームレータにセットしデータ出力命令を実行する。第2－32図は、テレタイプ制御装置のデコード回路である。以上は、データの印字プロセスであるが、最初に述べたように、入出力動作を行なう時には、装置が使用可能か否かのチェックが必要である。入出力装置が使用可能か否かのステータスが2番に割当てられているとするとアッキュムレータに〃100100000〃をセットし機能出力命令を出すと、これが制御装置に送られ、制御装置では、入出力装置の状態を状態レジスタにセットする。マイクロプロセッサは、次に状態入力命令を実行することにより、状態をアキュームレータに取込み、判断に使う。以上のフローが第2.33.図である。

(2) DMA（ダイレクトメモリアクセス.）

第.2.2.項で触れたように、高速な入出力装置では、マイクロプロセッサの速度で間に合わない場合が出てくる。LSIメモリも高速動作が可能であることから、入出力装置とメモリ間のデータ転送をマイクロプロセッサを介さずに行なうことにより、高速データ転送を行なおうというのがDMAの考え方である。DMAを実現する方法は2つ有る。第1の方法は、DMA転送中はプロセッサをストップさせてしまい、DMA転送終了時にこれを解第2－9表8008とμCOM8の入力命令

(a) 8008

IND(0100MMM1)；MMMという入力装置からアキュームレ

ータにデータを取込む。

OUT(01RRMMM1)；RRMMMという出力装置にアキュームレータの内容を出力する。

(b) μCOM8

IN(11011011B_2)；B_2 という入力装置からアキュームレータにデータを取込む。

OUT(11011011B_2)；B_2という出力装置にアキュームレータの内容を出力する。

＊ B_2 ；命令の2バイトめの8ビット。

く方法でインタロック方式と呼ばれる。第2の方法は、DMA 転送とプロセッサが同時にメモリをアクセスした時のみプロセッサをストッププロセッサをストップする方法でサイクルスチールとかインタレース方法と呼ばれる。当然第2の方法が効率は上がるが、制御は複雑になる。

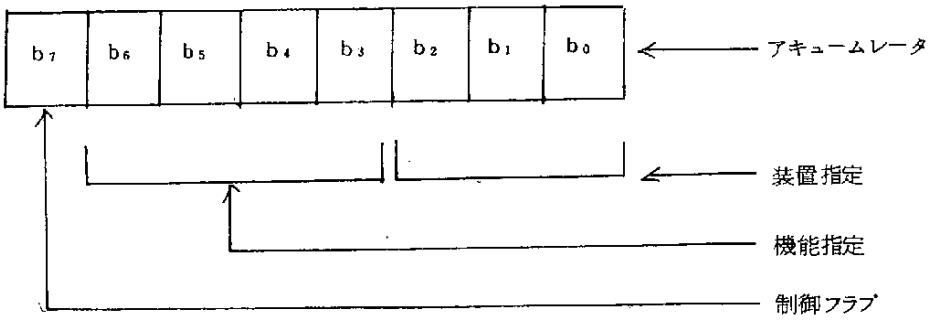
DMA の構成とインターフェースを第2－34図に示す。DMA の起動手順を次に示す。

ステップ1；出力命令で、転送アドレスカウンタにデータ転送を行なうメモリの先頭番地をセット。

ステップ2；出力命令で、転送語数カウンタに転送を行なう語数をセット。

ステップ3；出力命令でDMA をスタート。

この様にして転送が開始されるとDMA は入出力装置とメモリの間で一語づつ転送を行ない一語送ることに転送アドレスカウンタに1を加え、転送語数カウンタから1減じていく。そして転送語数カウンタが0になった時割込み信号をマイクロプロセッサに送り、転送シーケンスを終了する

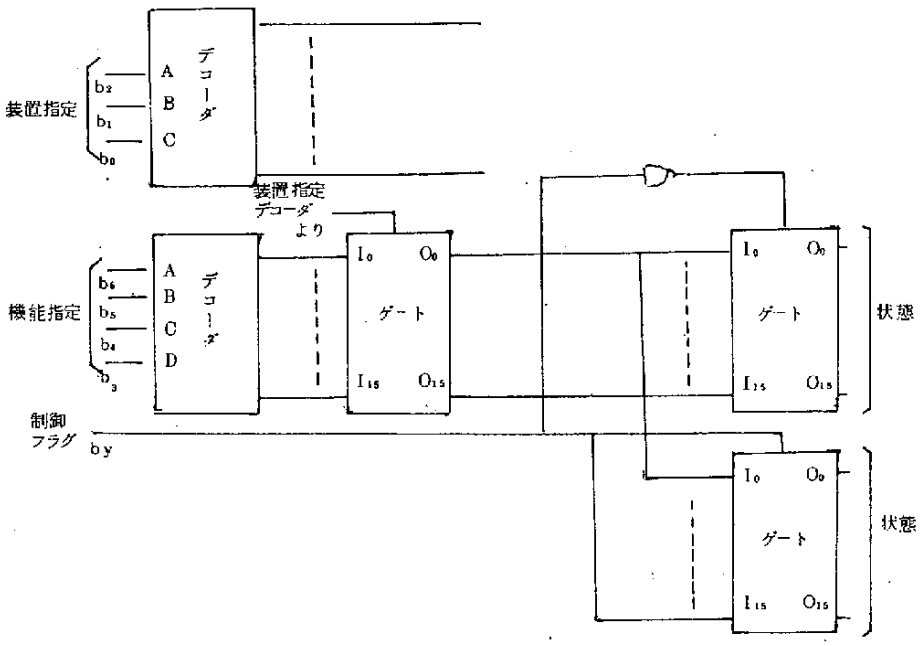


(a)装置指定　　指令を与える入出力装置番号（0～7）

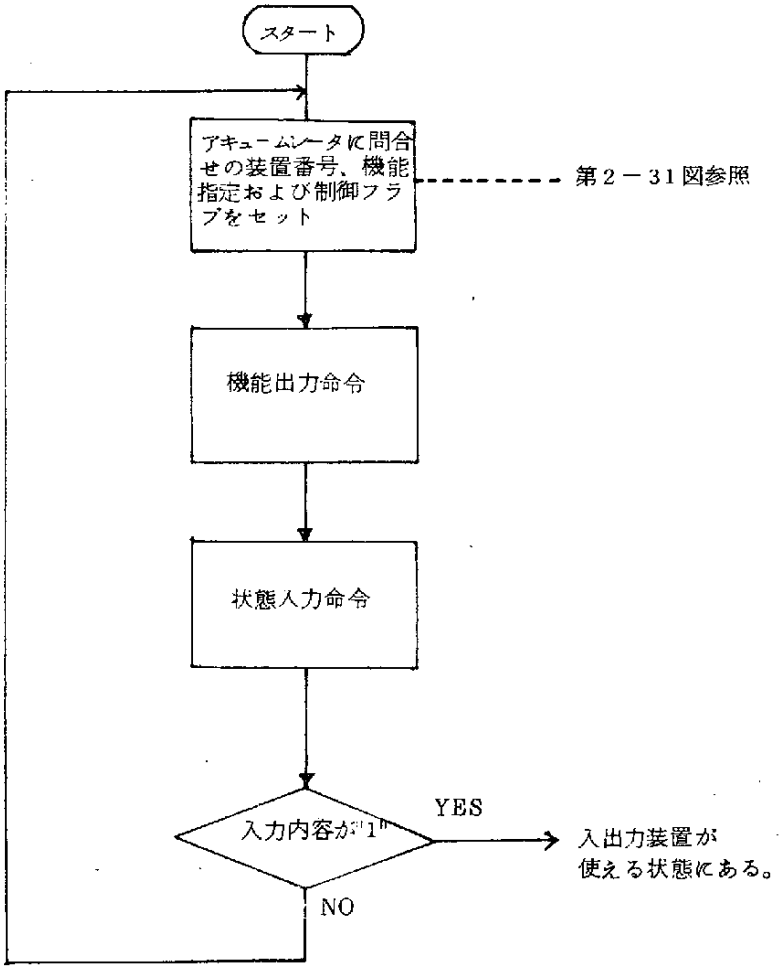
(b)機能指定　　入出力装置に対して与える指令コード（0～15）

(c)制御フラブ　　機能指定〝0〃か状態入力指定〝1〃

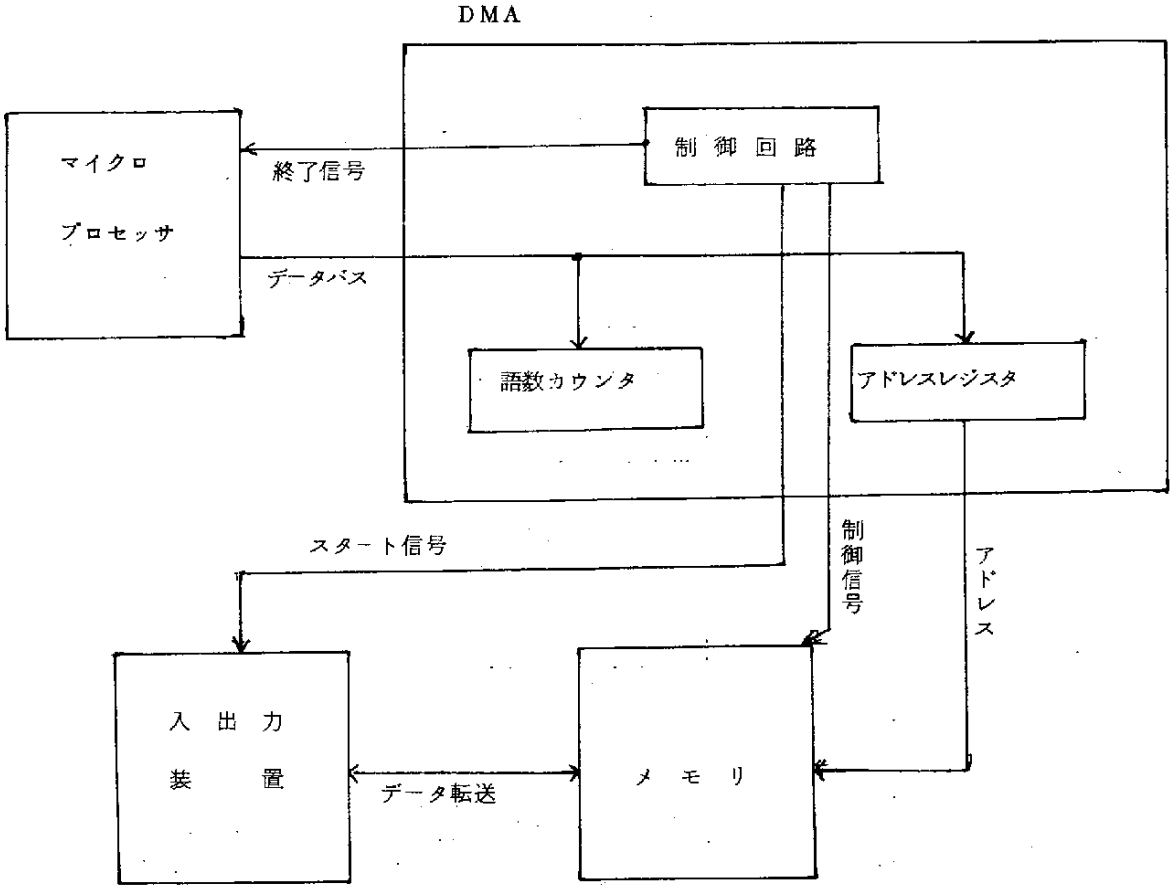
第2－31図　機能出力のフォーマット



第2－32図　機能出力命令デコード回路



第２－３３図　状態問合わせのフローチャート



第２－３４図　DMAの構成とインターフェース

2.4.2　入出力インターフェース用チップ

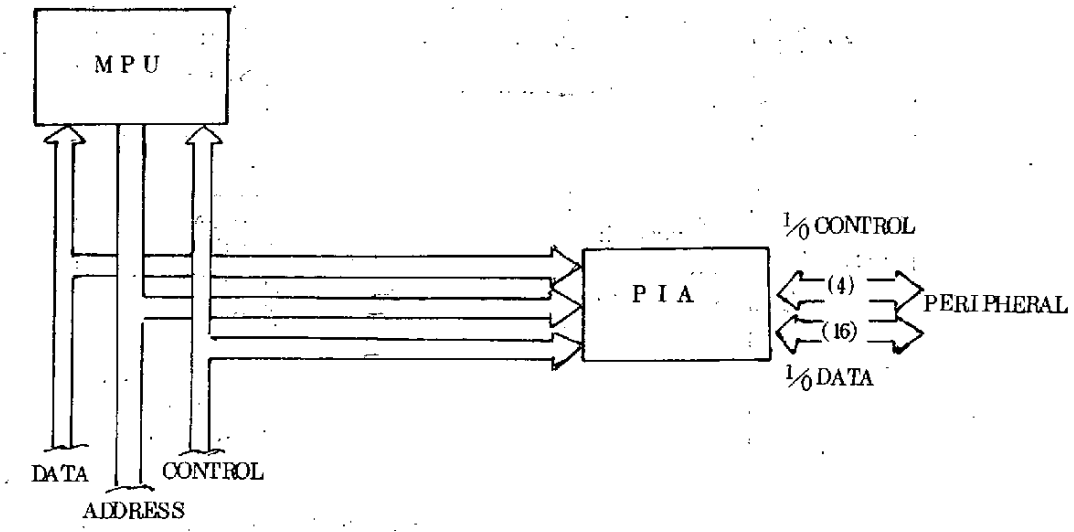
いままでの説明から想像されるとおり、インターフェースを個々ICで作るのは、大変な仕事である。そこでマイクロプロセッサメーカは、入出力力インターフェースを簡単に構成出来るチップを販売している。

この項では、これらのうち代表的なもの３点についてその概要を述べる。

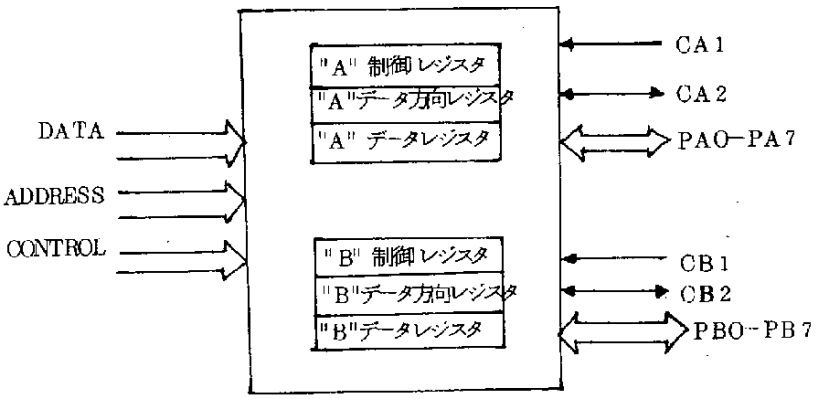
(1)　汎用並列インターフェース素子

この例としては、モトローラ社のMC６８２０というＰＩＡ（ペリフェラル・インターフェース・アダプタ）について述べる。

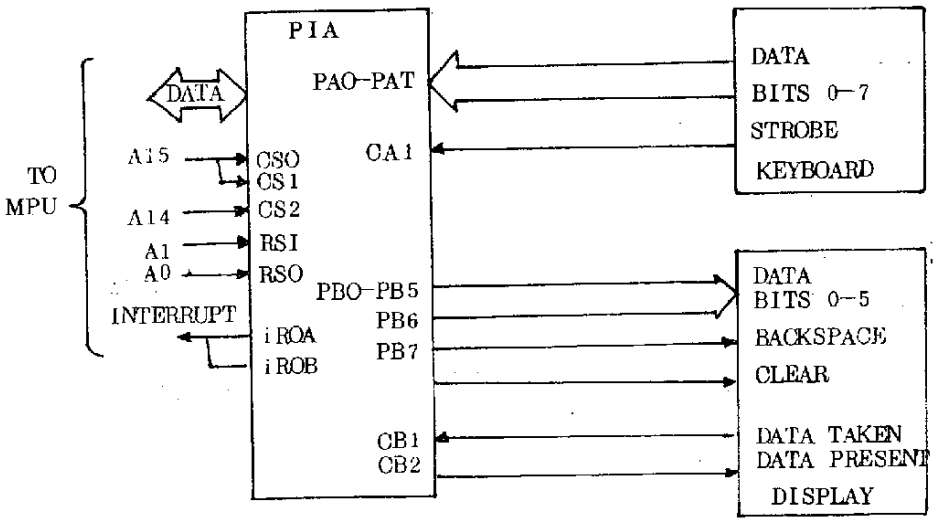
第２－３５図は、ＰＩＡ を用いた代表的なインタフェース例である。ＰＩＡ の第１の特徴は、ＰＩＡ の動作機能の決定が、システムイニシャライズ時にMPU （マイクロプロセッサ）から送られる制御情報で決定するという点である。ＰＩＡ は、入出力装置との間に１６本のデータ線と４本の制御線を持っているが、この制御線を「入出力が行なわれようとしていることをMPU 又は入出力装置に知らせるために使う」と定義することも出来る。こう設定しておくと、入出力装置が、ＰＩＡ を通してMPU にデータを送りたい時は、データをバスに用意し、制御線にパルスを送ると、ＰＩＡ がMPU に割込みを起こす。又MPU が入出力装置にデータを送りたい時は、メモリ参照命令によりデータをＰＩＡ に送ると、ＰＩＡ はこれをセーブレ、入出力装置へのバス乗せる。その時ＰＩＡ が入出力装置へのデータが用意出来たことを知らせる制御パルスを発生する。機能があれていればＰＩＡ をアドレスとしＳＴＯＲＥ命令を出すだけで、〝データプレゼント〟パルス付のデータ転送が行なえる。



第２－３５　ＰＩＡによるインタフェース



第２－３６図　ＰＩＡの内部レジスタ



第２－３７図　キーボード・ディスプレーとのインターフェース

第２－３６図は、ＰＩＡ の内部構成を示しており、６つのレジスタがＡサイドとＢサイドに３コづつ分かれている。各サイドは、８本のデータ線、２本の制御線、３つの８ビットレジスタを持つ。そしてユーザはコントロールレジスタと方向レジスタに情報をセットすることにより、データ線や信号線を機能づけできる。例えば、Ａサイドの方向レジスタ〝０〟をセットするとＡサイドのデータ線は入力用として定義され、〝１〟をセットすると出力用としてセットされる。同様に制御線、ＣＡ 1.2は、Ａサイドのコントロールレジスタに情報をセットすることにより機能付けされる。Ｂサイドについてはサイドと同じである。次に具体例としてＰＩＡ をキーボードとディスプレーのインターフェースに用いた例を示す。第２－３７図はその構成を示しており、Ａサイドがキーボードからの入力用に、Ｂサイドがディスプレーへの出力用になっている。キーボードとのインターフェースは、データ転送用ＰＡ0－ＰＡ7 とＭＰＵ に新しい入力データがあることを知らせるストローブＣＡ1である。ディスプレーとのインターフェースは、データ転送用にＰＢ0－ＰＢ7 が使われる。

又制御線は、シェークハンドモードで働き、新しいデータをディスプレーする時は、CB_2に〝データプレゼント〟信号が出て、それからCB_1に〝データテイクン〟信号がディスプレー装置から返される。ＰＩＡ とＭＰＵ のインターフェースはデータ線、アドレス線、割込み線であるが、重要な点は、チップ内の６つのレジスタは全てＭＰＵ からアドレスされアクセス可能であるということである。以上の機能をＰＩＡ に行なわせるために必要な初期設定は、次のとおりである。

(イ) ＰＡ0－ＰＡ7 は入力ポートである。

(ロ) ＰＩＡ は、ＭＰＵ にＣＡ １入力の立上りで割込みをおこす。

(ハ) ＣＡ ２は、使わない。

(ニ) ＰＢ0－ＰＢ7 は出力ポートである。

(ホ) PIAは、MPUにCB 1入力の立下で割込みをおこす。

(ヘ) MPUからPIAにデータを書込む時CB2にストローブ信号を発生する。

(2) トランスミッタ・レシーバ・チップ

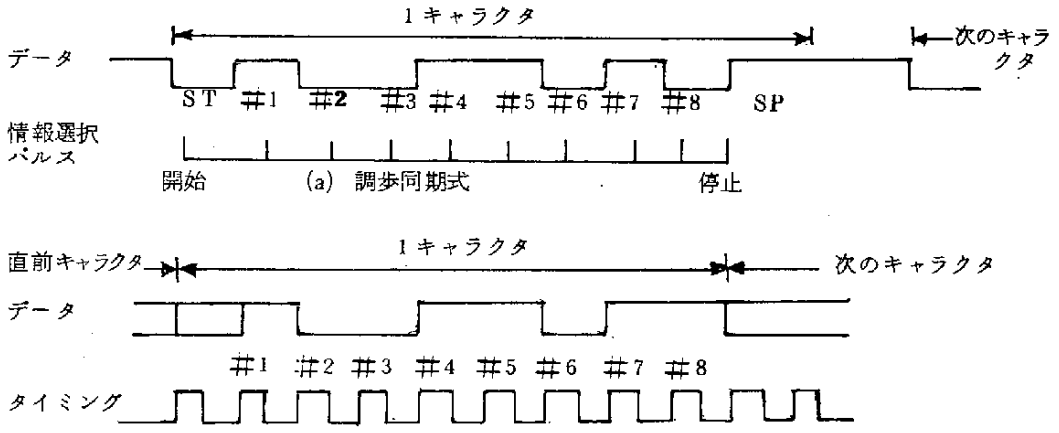
直列インターフェース素子としては、その最も基本的なトランスミッタ・レシーバを選んだ。マイクロプロセッサから出力されるデータは並列データであるが、これを通信回線で送ろうとすると、直列データに変換しなくてはならない。又テレタイプのように通信用として用いられて来た端末は、データの送受信を直列データで行なう。トランズミッタレシーバは、このようにデータの直並変換を必要とするインターフェースを容易に作成する目的で設計されている。

ここでは、トランスミッタレシーバの説明に入る前に、回線を送られるデータの形式について先ず触れる。データ転送の基本的な2つの代表例には、調歩同期式と独立同期式がある。調歩式は、転送する各データ(キャラクタ)の先頭に1ビットのスタートエレメントと、データのあとに、1ないし2ビットのストップエレメントを付けて転送を行ない、データの無い時は回路にハイ信号を与えておく。受信側では、このスタート、ストップエレメントを目印に、データの取込みを行なう。一方独立同期式は、スタート、ストップエレメントが無く、一度転送がはじまると、キャラクタのみを連続して送り続ける。そしてデータが無い時はダミーキャラクタを送る。

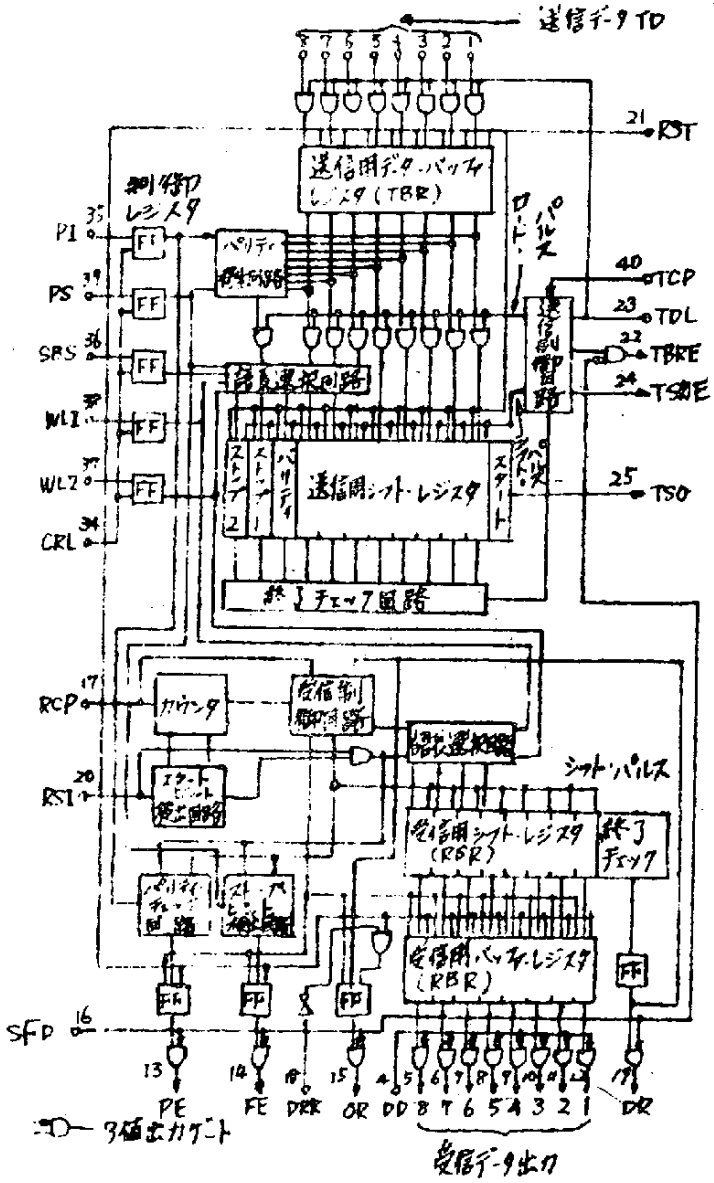
故に、一度同期キャラクタで送受信間の同期を取ったら後は、両者独立に同期信号を作り、データの識別を行なう。第2－38図は、それぞれの信号波形を示している。キャラクタと1ビットのパリテーで構成されている。しかしシステムによっては、キャラクタのビット数が5～8まで色々あり、パリティも偶数パリティ、奇数パリティ、パリティなしといった具合に色々である。

トランスミッタ・レシーバ・チップの構成と、動作について説明する。トランスミッタ・レシーバと一口に言ってもその構成はメーカにより色々である。ここではテレタイプとのインタフェース用に調歩同期式のものを説明する。回線で送られるデータの形式は、先に示したようにいろいろのものがあるが、それぞれについてチップを作るのでは、メーカが大変である。そこで、チップ内のフリップ・フロップ（制御レジスタ）に、データを初期設定することにより、個々のシステムの転送形態に合った機能にプログラムする方法がよく用いられている。第２－３９図に示すチップでは、ＰＩに接続されているＦ・Ｆをセットすることが、パリティビットの付加と、チェックを行なう機能を働かせることを意味し、ＰＳに接続されているFEをセットすることがパリティの内容（偶数パリティ、奇数パリティ）を決定することになる。ＳＢＳは、ストップエレメントのビット数が１ビットか２ビットかを定義する。ＷＬ 1.2は、１キャラクタのビット数が、5.6.7.8のいずれであるかを決定する。これからの情報は、システムイニシャライズ時に、ＣＲＬパルスで制御レジスタにセットされる。機能的には、第２－３９図の上半分がトランミッタ部で、マイクロプロセッサからの８ビットデータがＴＤ端子に与えられると、ＴＤＬパルスでＴＢＲにロードされる。この時ＴＢＲＥはハイになる。次にこのデータはＴＳＲに送られ、スタート、ストップビットとパリティビットを付加されＴＳＯより直列データとなって出力される。この転送終了時には、ＴＳＲＥはハイになる。次に下半分はレシーバ部であるが、テレタイピからの直列データはＲＳＩより入り、ＲＳＲにセットされる。そしてストップビットが検出されると、ＲＳＲの内容をＲＢＲに移し、ＤＲフラグをハイにして転送を終了する。マイクロプロセッサは、この並列データを受取り、ＤＲＲにパルスを送り、ＤＲ・ＦＦをリセットする。以上がトランスミッタ・レシーバの機能である。これを

テレタイプのインターフェースとして用いる時の構成を第2.－４０図に示す

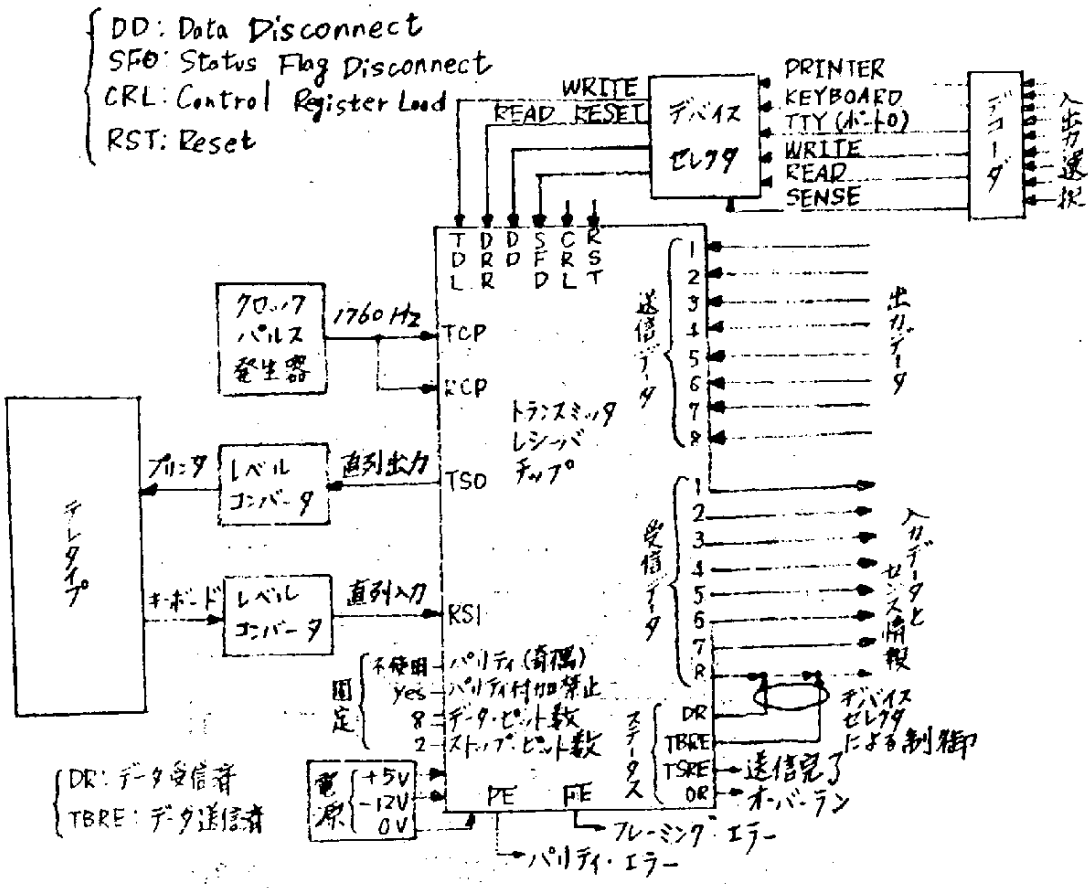


第２－３８図　回線上のデータ構造



○印の端子は入力端子→印は出力端子。端子部に示した数字は端子番号

第２－３９図　トランスミッタ・レシーバの構成例



第２－４０図　テレタイプとのインターフェース例

(3) 専用インターフェースチップ

専用インターフェースチップは、特定のマイクロプロセッサと、特定の入出力装置との間のインターフェース用に設計されたチップで、汎用のものより、多くの機能が組込まれていることが普通で、ユーザにとっては、大変便利なものである。この種のチップとしては、μCOMシリーズ用にいろいろのものが出ているので第2－10表にその一覧を示す。ここでは、このうちカセットMT インターフェースチップ（μPD714D）の概要を示す。

書込み動作時には、4ビットずつ2回に分けて入力された8ビットの並列データを、PE記録型式の直列信号に変換し出力すると同時にCRC 演算を行ない、書込みデータの終了によりCRC ビットを自動的に発生する。又MT に書込んだデータは、リードアフタライトを行なうことにより、CRC 方式によるエラーチェックを行なうことが出来る。読出し動作時には、MT からの直列データを4ビット並列データに変換し出力する。又同時にCRC 方式によるエラーチェックも行なっている。第2－41図にチップのブロック図を示す。主な端子の機能は次のとおりである。

DB0～3；マイクロプロセッサとの間のデータ転送用。

AR1,2 ；カセットMTからの入力端子。

TS ；カセットMTへの出力端子。

AB0～3；チップに対する機能指定。

R／W ；入出力の区別を指示する端子。

このチップの動作は、AB0～3とR／W の値により決定され、第2－11表は、その表である。この表の理解を助けるために2,3の命令についてその働きを示す。

第1は、AB3～0入力が、〝0001〟、R／W が〝1〟の時、即ち第2－11表でいえば、左で、〝書込み入力1〟という命令になる。これは、デ

ータバスDB0～3のデータを入力バッファ1に取込な命令である。

第2は、〝書込みスタート〟という命令をみてみる。これはR/Wが〝1〟AB3～0が〝0100〟、DB3～0が〝0100〟の場合で、入力バッファの内容をTS端子へ出力するものである。

このように専用チップは、かなり具体的動作を指定することが出来、インターフェースの構成と入出力用ソフトウェアは、非常に簡単になる。

2.5 その他のマイクロコンピュータ構成要素

この節は、以上述べて来たマイクロコンピュータ構成要素の他で、マイクロプロセッサメーカがマイクロプロセッサのファミリとしてチップ化しているいくつかのICについてその概要を述べる。

2.5.1 クロックジェネレータ

マイクロプロセッサを作動させるには、各マイクロプロセッサに固有のクロックが必要である。クロックは、単相から多相まであり多相のクロック回路をICで組もうとすると容易ではない。

そこで普通マイクロプロセッサメーカは、各マイクロプロセッサ用クロックジェネレータチップを提供し、ユーザはこれに必要とする速度の水晶又はRC発振子を取付けて必要なクロックをつくり出すようになっている。マイクロプロセッサによっては、チップ内にクロックジェネレート回路を持ち、マイクロプロセッサに、水晶又はRC発振子を付けるだけですむものもある。ここでは、この種のチップの例として1NTEL8224 について述べる。チップは1NTEL8080 のクロックをジェネレートするもので、その内部のブロック図が第2—42図(a)である。基本構成は、クリスタル・コントロールド・オシレータ、9分割カウンタ、2つのハイレベル・ドライバとその他の回路よりなり第2—42図(b)の様な波形を生成する。第2—43図は、これと同

じ波形を出す回路をＩＣで組む時の回路図で、このチップがいかに便利かがわかる。

2.5.2　優先度のある割込制御ユニット

このチップの例としてINTEL8214を示す。このチップは、割込み方式のマイクロコンピュータを簡単に構成するために設計されたもので8レベルまでの割込みを処理出来る。第２－４４図がこのチップ（ＰＩＣＵ）を用いて、割込み方式の入出力操作を実現した例であり、ＰＩＣＵ へは各入出力装置から１本づつの割込み線が入っている。

ＰＩＣＵ の基本動作は、次のとおりである。

ステップ１；１／０からの割込みリクエストを受付ける。

ステップ２；どの１／０からの割込みかを決定。

但し複数台からのリクエストが有る時は、それらのうちから最も優先度の高い１／０に決定。

ステップ３；決定された割込みの優先度を、チップ内のレジスタ（カレントステータスレジスタ）と比較し、割込みの方が優先度が高いときのみ、バスに、割込み処理ルーチンを識別する情報を出しシステムに割込みをおこす。

第２－４５図に、このチップのブロック図を示すものである。入出力装置等外部からの割込み要求は、端子R_0～R_7に入りプライオリティエンコーダに送られる。プライオリティエンコーダでは、割込みの入った端子のうち最も優先度の高い（Ｒ７が最も高くＲＯが最も低い）ものの番号を３ビットにエンコードし出力する。そして、このチップで割込みが一度受付けられるとインタラプト・ディセイブルＦＦ（ＩＮＴ　ＤＩＳ　ＦＦ）がセットされ、割込みラッチが新たな割込みを受付けない状態にされる。プライオリティエンコーダの出力３ビットは、カレントステータスレジスタの出力３ビットと比較され前者の方が大きい時のみマイクロプロセッサに割込みを起こす。カレントテー

第２－１０表　μCOMシリーズの専用インタフェースチップ

(I)　μCOM－４ファミリー

μPD７１４D ；カセットMT・コントローラ

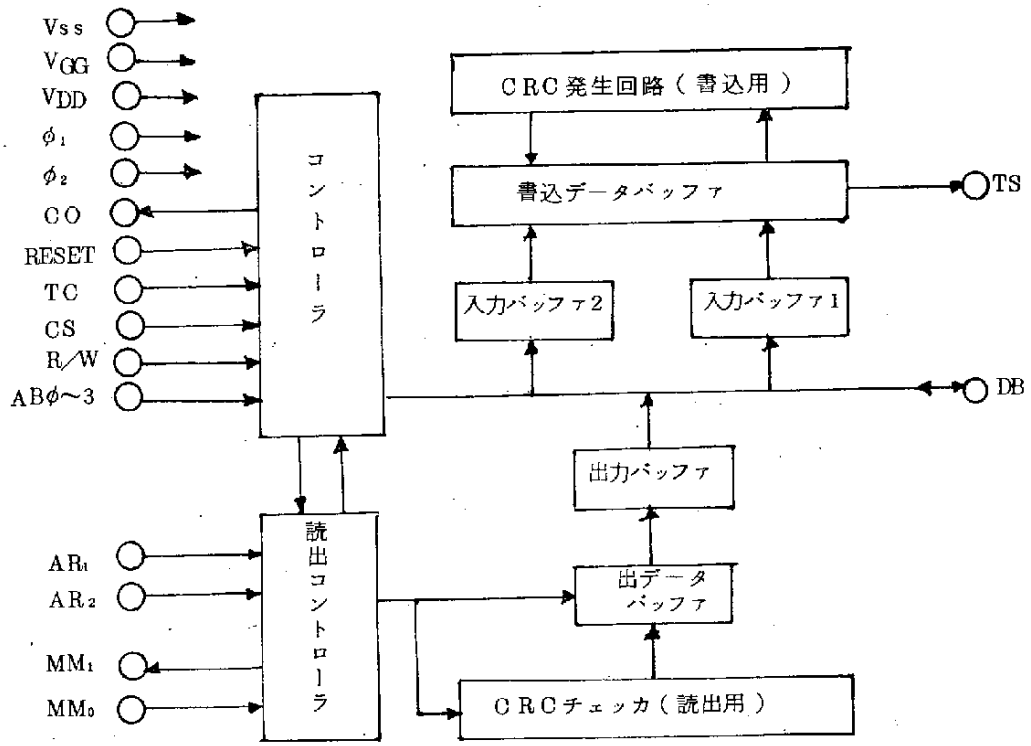
μPD７５７C ；キーボード・ディスプレイ・コントローラ

μPD７５８C ；ディジタルプリンタ・コントローラ

(II)　μCOM－８ファミリー

μPD３７１D ；カセットMT・コントローラ

μPD３７２D ；フレキシブルディスク・コントローラ



第２－４１図　μPD７１４Dブロック図

第２－１１表　コントロールコード表

＊1 OBF：OUTPUT BUFFER FLAG
データが読出データバッファから出力バッファに転送されると〝１〟となる。

＊2 IBF：INPUT BUFFER FLAG
データが入力バッファから書込データバッファに転送されると〝１〟になる。

ＡＢ入力；

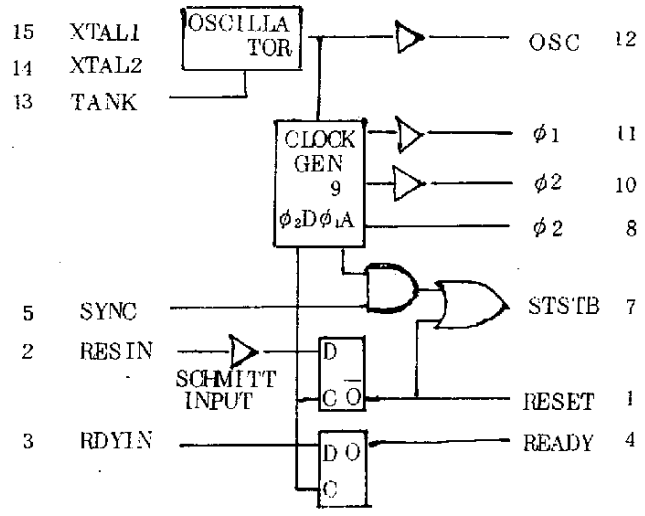
AB_3	AB_2	AB_1	AB_0

R/W＝1（ストア命令）		AB_0	AB_1	AB_2	AB_3
	DB_0	書込データ入力１	書込データ入力２	テープ・スタート　C0→〝０〟	CRC発生器 ON
	DB_1	DB_0 ～3→IB1	DB_0 ～3→IB2	テープ・ストップ・CRC転送 C0→〝１〟、CRCC→OB	CRC発生器 OFF
	DB_2			書込スタート	OBFリセット OBF→〝０〟
	DB_3			IBFリセット　IBF→〝０〟	
R/W＝0（ロード命令）	DB_0	読出データ出力	コピーOBF＊1 OBF－DB_0	コピー　IBF＊2　IBF→DB_0	
	DB_1	(OB)→DB_0			
	DB_2	～3			
	DB_3				

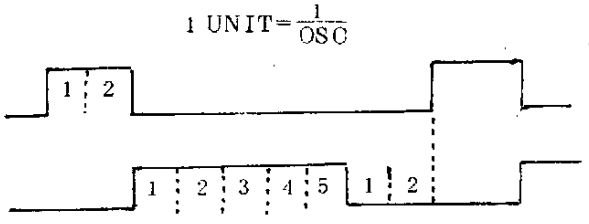
タスレジスタは、4ビットのラッチで、マイクロプロセッサはこれをアドレス可能な出力ポートとして扱い$\overline{ECS}$をローにすることで内容セットを行なう。カレントステータスレジスタの一般的な使い方は、割込みが発生した時にその処理ルーチンがこのレジスタにその割込みの優先度を書き込むことにより、割込み処理中に、より低いレベルの割込みで処理が中断するのを防いでいる。又マイクロプロセッサへの割込みを起こす際に示す割込みレベルは、$\overline{A0}$～$\overline{A2}$に出力される。以上がこのチップの概要である。

2.5.3　入出力ポート

入出力ポートの例としてINTEL8212の概要と、そのアプリケーション例を示す。このチップは、8ビット入出力ポートで、3ステート出力バッファ付の8ビットのラッチと、制御やデバイスセレクションロジックから構成されている。又このチップは、マイクロプロセッサへの割込み生成や制御を行なうサービスリクエストフリップ・フロップを持つ。このチップの用途としては、ラッチやマルチプレクサ等があり、これについては後でその例を示す。第2－45図は、このブロック図である。8個のデータラッチは、D型FFで、非同期リセット入力〔CLR〕によりクリアされる。出力バッファは、ラッチ出力の伝達制御に用いられる。DS1、DS2は、このデバイスのセレクト信号となり、セレクトされると出力バッファがエネイブルになり、サービスリクエスト（SR）FFがセットされる。MDは、出力バッファの状態コントロールとデータラッチのタイミングを制御するのに用いられる。具体的には、MDがハイ（出力モード）の時は、出力バッファはエネイブルされデータのラッチは、DS1、DS2で行なわれる。又MDがロー（入力モード）では、出力バッファの動作は、DS1 DS2で決まり、データのラッチは、STB入力で行なわれる。STB入力は、入力モード時のデータのラッチングとサービスリクエストFF



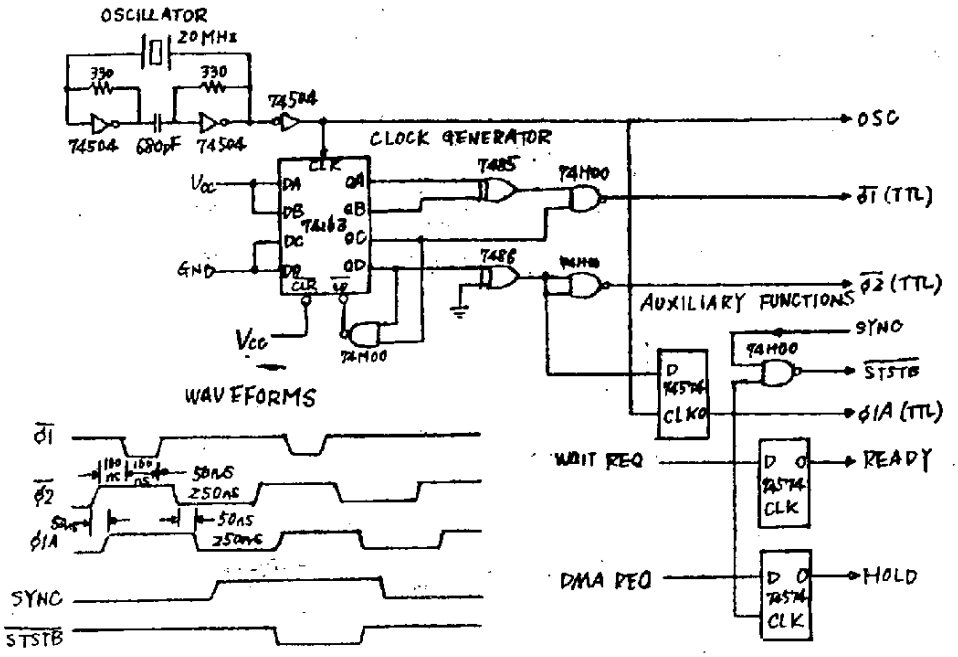
(a) チップのブロック図



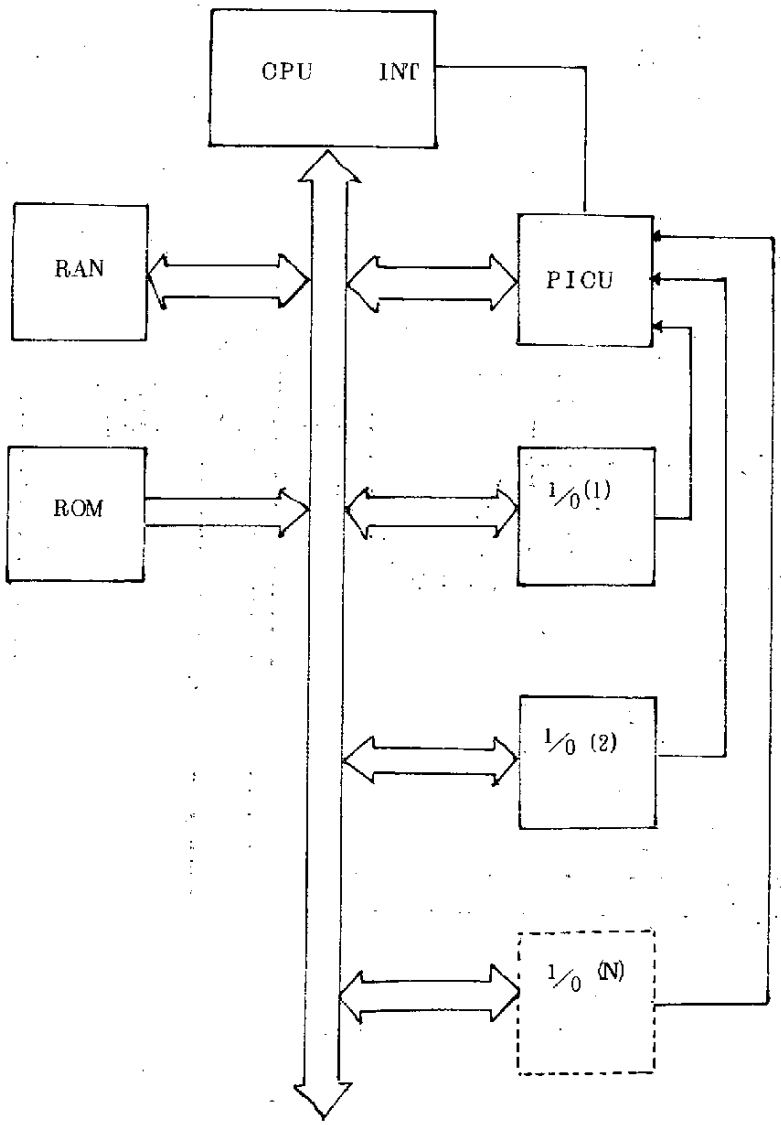
EXAMPLE：(8080 tcy=500ns)
OSC=18mH₂/55ns
$\phi 1$=110ns(2×55ns)
$\phi 2$=275ns(2×55ns)
$\phi 2$=$\phi 1$=110ns(2×55ns)

(b) 発生波形

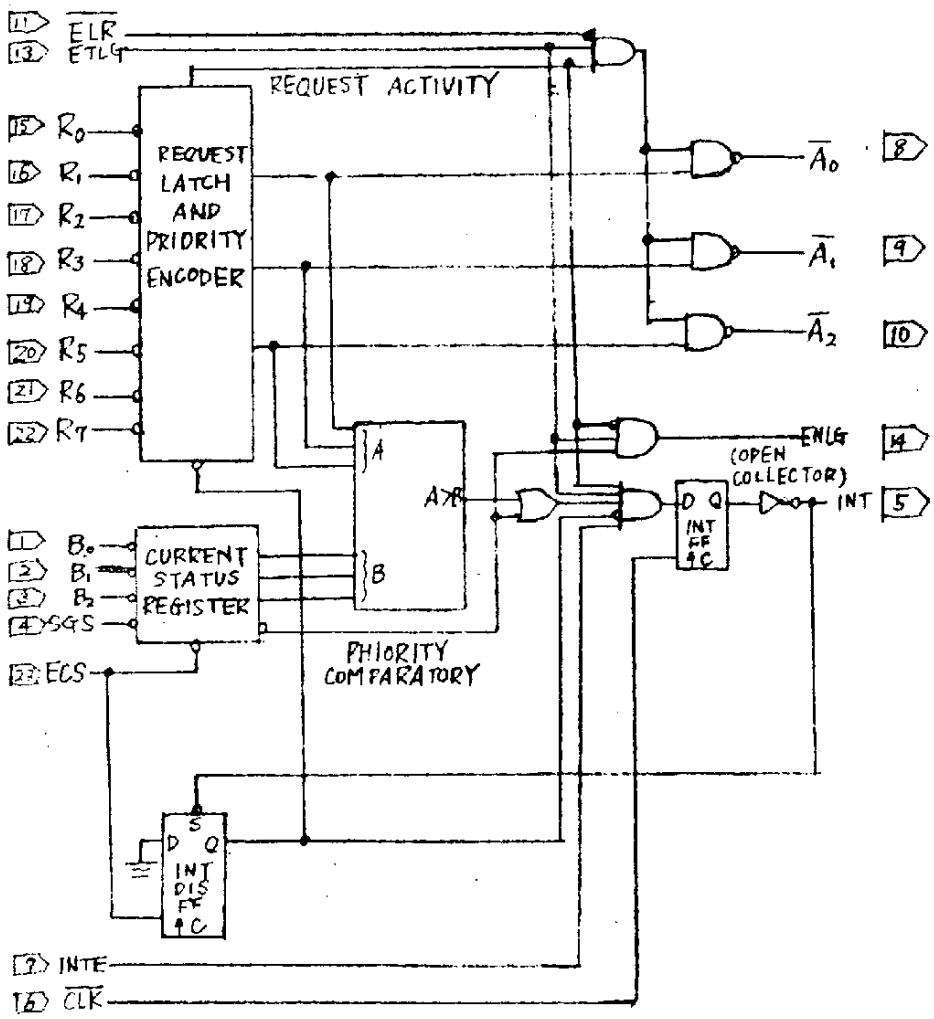
第２－４１図　クロックジェネレータの構成と生成波形



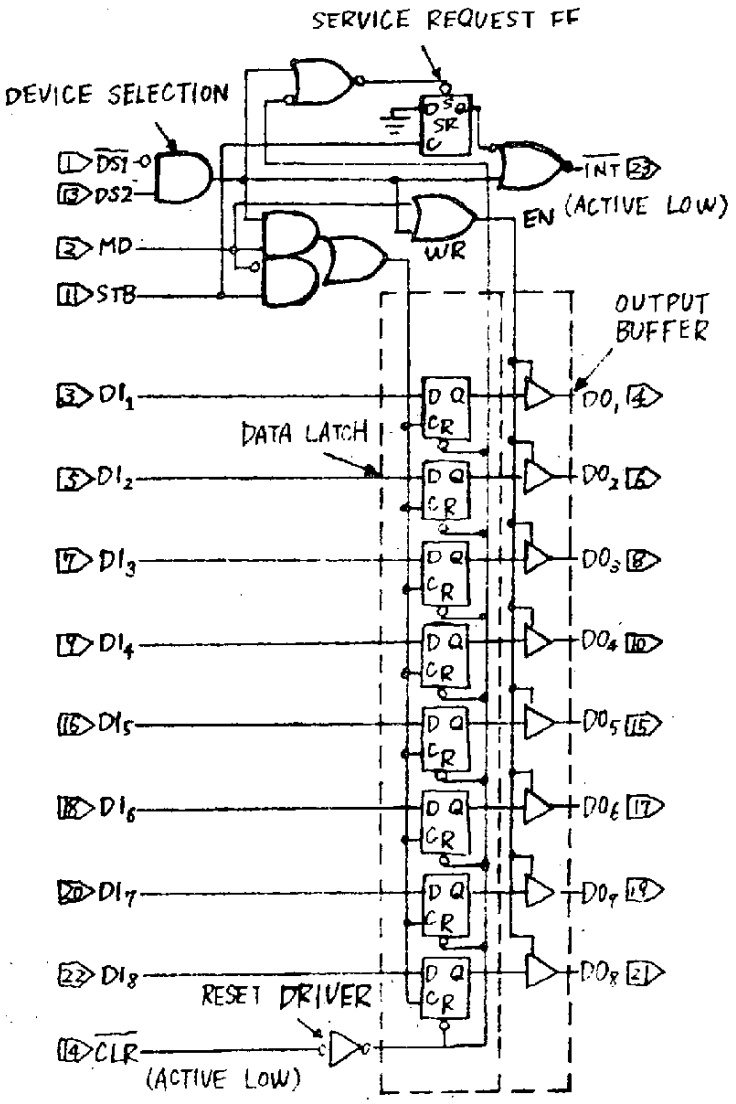
第２－４２図　ＩＣを用いたクロックジェネレータ回路



第２－４３図　ＰＩＣＵを使った割込みシステム

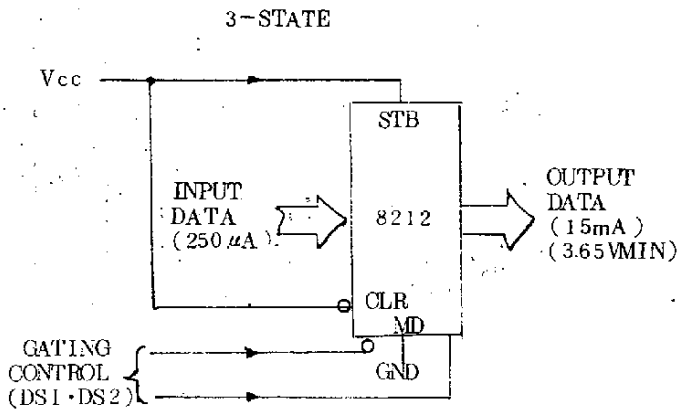


第２－４４図　ＰＩＣＵのブロック図

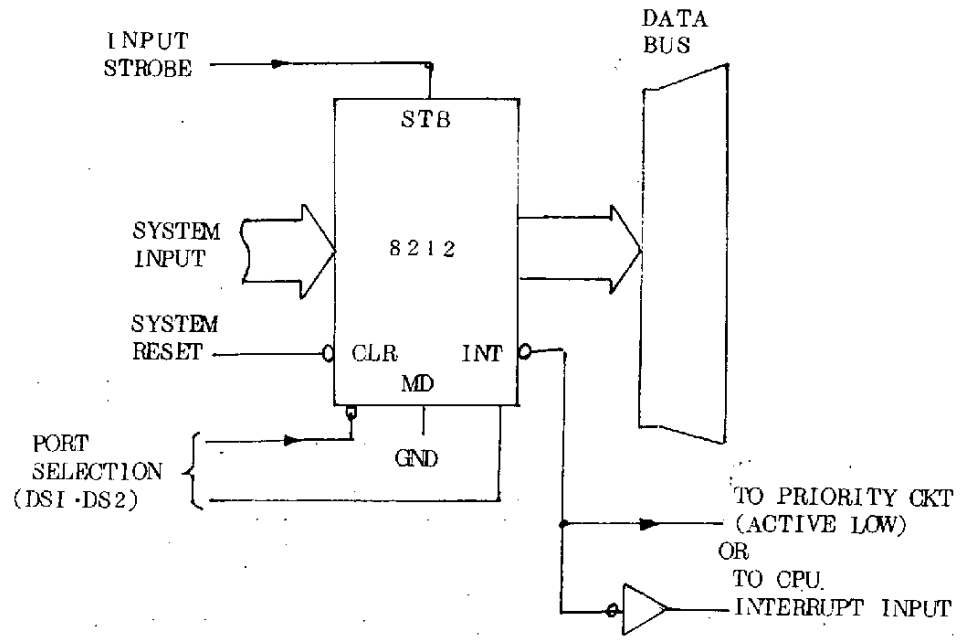


第２－４５図　８ビット入出力ポートの構成

を同期リセットするために用いられる。サービスリクエストＦＦは、マイクロコンピュータシステムで割込みを発生させるために用い、ＣＬＲ 入力でセットされる。但しこのＦＦがセットされている時は、非割込み状態である。次にこのチップを用いた例を３つ示す。第１の例は、ゲーテッドバッファで、第２－４６図に示すものである。即ち、ＭＤをＧＮＤ につなぎ、ＳＴＢ をハイにすると、データラッチはＳＴＢがハイなのですぐ行なわれ、出力バッファもこのディバイスがセレクトされていればエネイブルなのでゲーテッドバッファは出来上る。第２の例は、割込み入力ポートで、このチップはシステム入力ソースから、ストローブ信号をＳＴＢ に受け、サービスリクエストＦＦをセットし割込み処理ルーチンが走り、このデバイスをセレクトすることによりシステムインプットのデータがバスに送られる。第３の例は、８０８０（μＣＯＭ８）のステータスラッチの方法であり、接続図を第２－４７図に示す。



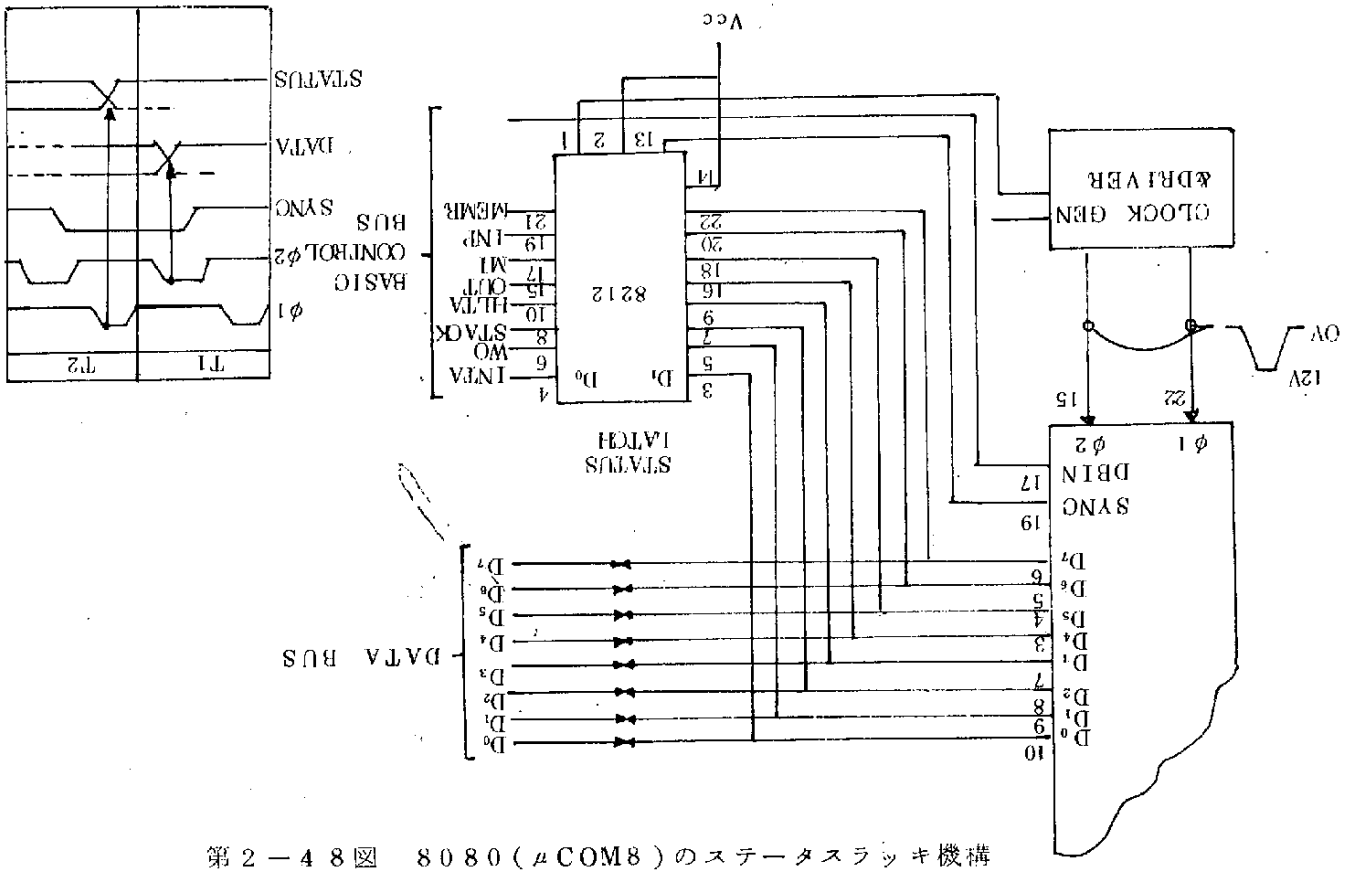
第２－４６図　ゲーテッドバッファ



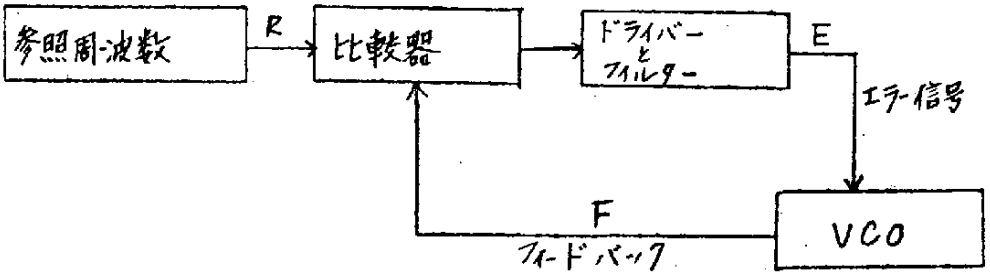
第２－４７図　割込み入力ポート

2.6　制御のしくみ

マイクロプロセッサの制御系への応用例として、フェーズロックループによるモータの制御にマイクロプロセッサを用いる場合を考える。一般的なフェーズロックループの構成は、第２－４９図に示すように、参照周波数発生器、位相比較器、ループフィルタ、ボルテージコントロールドオッシレータ（ＶＣＯ）から成立っている。その基本動作は、出力フェーズと参照フェーズが、フェーズ比較器で計算され、この２つのフェーズ誤差を修正するのに必要なエラー修正電圧がＶＣＯに加えられ、出力フェーズを参照フェーズに合わせていく様になっている。要するに、フィールドバックループがＶＣＯ周波数を参照周波数に同調させる働きをしている。



第２－４８図　８０８０（μCOM8）のステータスラッチ機構



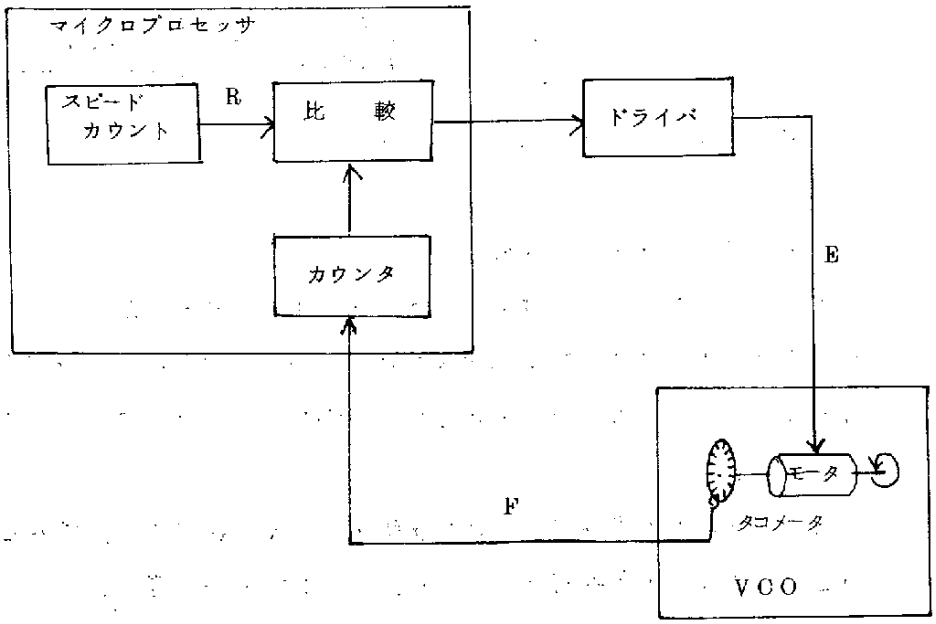
第２－４９図　基本的なフェーズロックループ

モータ制御システムの構成

第２－５０図がフェーズロックループを用いたモータ制御システムで、第２－４９図の一般型と比較すればわかるとおり、フェーズ比較部がマイクロプロセッサで実現され、ＶＣＯがモータとタコメータになっている。

次にこれらの構成要素について説明を加えると、先ずここで考えているモータは、永久磁石ステータを持ったＤＣモータ等スピードとトルクと電流の間に線型な関係のあるもので、スピードの脈動を押える程度の回転質性を持つものを仮定している。タコメータは、モータのシャフトに円盤を取付け、その円盤の円周に沿って小穴をあけ、そこを光が通るようにセットしておき、円盤の回転によりそこを通る光を電流パルスに変えて回転速度を求める型式のものが一般的である。第２－４９図にあるローパスフィルタは、モータの回転慣性が大きく、回転スピードに脈動をおさえる働きがあるため、ここでは不要となる。最後にマイクロプロセッサの働きであるが、これはまずタコメータから送られて来るモータ速度に比例したパルス巾をカウンタし、これを参照波のパルス巾をカウントしたものと比較し、もし参照カウントの方が大きければ、回転スピードの増加を意味するものであるから電源

を切りスピードを遅らせ、反対に参照カウントの方が小さければ、スピードがダウンしたことであるから電力を増して、参照スピードに合わせるという操作を行なう。



第２－５０図　マイクロプロセッサのフェーズロックループへの導入

・　マイクロプロセッサによる制御

以上の説明で、システムの構成と制御の概略はわかったと思われるので、次にプロセッサの詳細な制御のしくみと、被制御系とのインターフェースについて述べる。第２－５１図は、制御系の詳細を、第２－５２図は第２－５１図の３つの信号Ｒ、Ｆ、Ｅの波形を示している。ここでＲは、参照波形を、Ｆは、タコメータの出力即ちモータの回転状態も示す波形を、Ｅは、モータの回転状態を参照波形で示される正しい状態に同調させるためにマイクロプロセッサで計算されたエラー修正信号を示している。次に制御のしくみであるが、この制御系では、

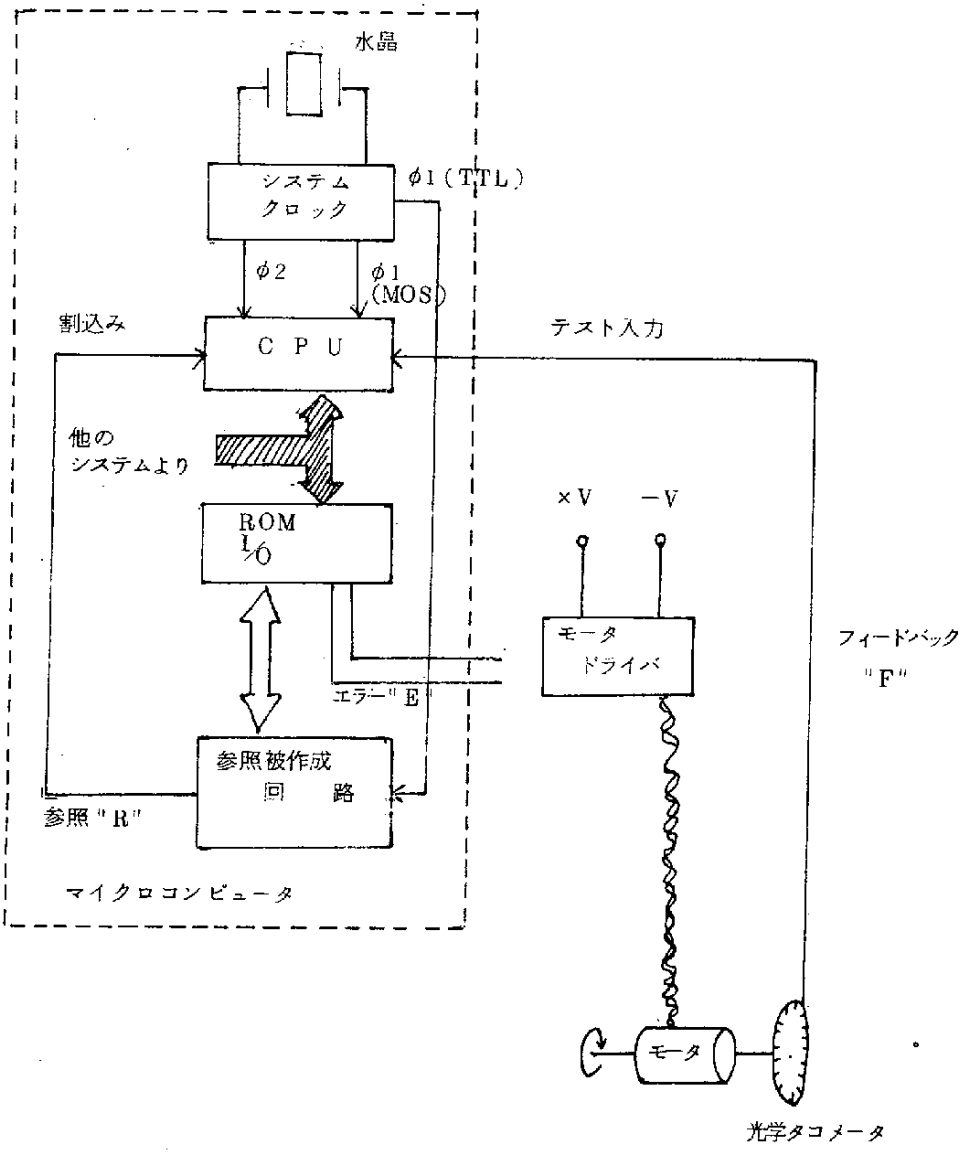
フェーズロックループを２つのオペレーションモードで実現している。第１のモードは、〝インロック〟でモータの周波数は正しいが、フェーズのみがずれる場合、第２のモードは、〝アウトオブロック〟で周波数がずれた場合である。この２つのモードの区別は、許容誤差で決まる。要するに、インロックとはいえ参照周波数とモータの周波数の間には多少の差はあるわけで、これが許容誤差内のときはインロックとし、許容誤差外の時はアウトオブロックモードでオペレーションを行なう。もちろんシステムごとに許容誤差は設定される。次に各モードでの制御のしくみであるが、インロックモードでは誤差修正に低いデューティのパルスを与えることによりフェーズの補正を行なっている。第２－３４図の左上の図がその様子を示している。即ち参照波形とモータ回転による波形とのフェーズの差は、Ｒの立上りとＦの立上りの差で測定出来る。故にマイクロプロセッサは、Ｒの立上りで、モ信号を出し、下の立上りでこれを切ることにより、修正が可能である。第２－５３図は、マイクロプロセッサの処理のフローを示しており、この上半分が以上の操作に対応する部分である。ここではＲの立上りを割込みとしてマイクロプロセッサに送ると、マイクロプロセッサがＥを出力し、Ｆの立上りを検出してＥをオフにするむねが示されている。その原理は、モータの負荷が増加してスピードが遅れれば、Ｅの巾が増しモータへ送られるパワーが増しスピードが上る。又負荷の減少等でモータのスピードが上れば、Ｅの巾が減りモータへのパワー総量が減るためスピードがおくれるというものである。

次に第２のモード〝アウトオブロック〟であるが、これはモータへの負荷が急激に変動して周波数が大きく変わった時のモードであり、システムとしては、全力を上げてモータを元の状態にもどさなくてはならない。具体的には、周波数が減った時は、周波数が参照周波数にもどるまでＥを出し続け、反対に周波数が増えた時にはＥをオフにし

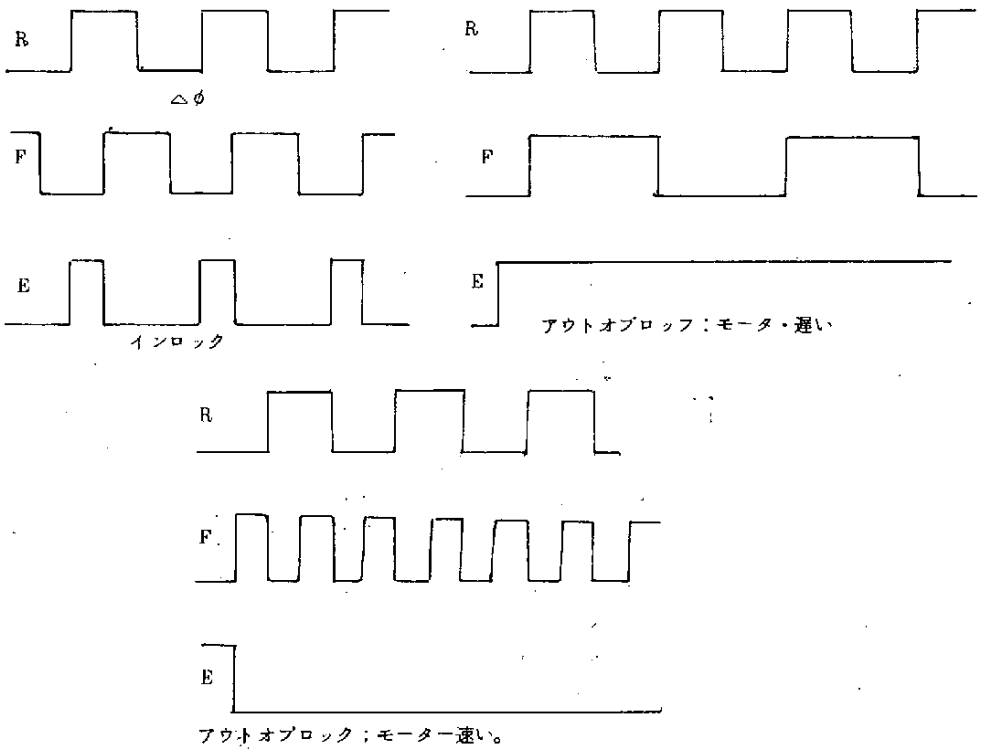
続け、回転スピードが参照周波数にもどるまで待つという方法である。この処理のフローが第２－５３図の下半分である。

さて、ここで周波数が正しい値かを測定する方法であるが、これには信号下のパルス巾を測定することに相当する。マイクロプロセッサを用いての測定は、処理時間のわかっている命令をパルスがオンの間に何度実行出来るかによって行なわれる。

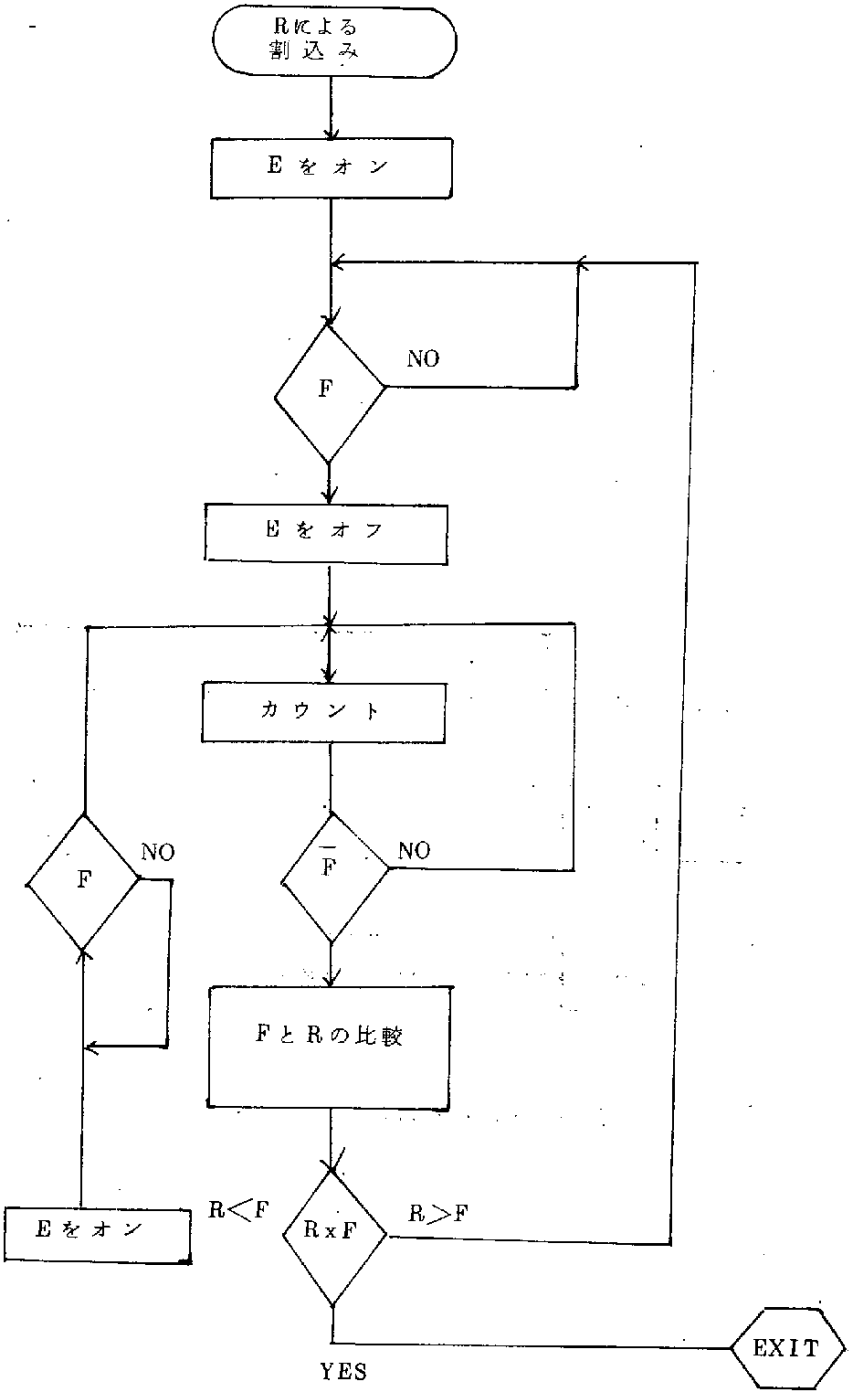
以上が制御のしくみであるが、場合によっては、フェーズを調べる必要は無く、周波数だけをチェックしていけば良いようなことも有る。たとえば、大慣性系では、一度所定のスピードが達成されると、あとはそのはずみ車効果で負荷の変動や入力エネルギの変動が吸収されるような時である。その様な時は周波数のチェックのみを行なえば良いわけであり第２－５４図にその時の処理フローと、信号の波形を示す。



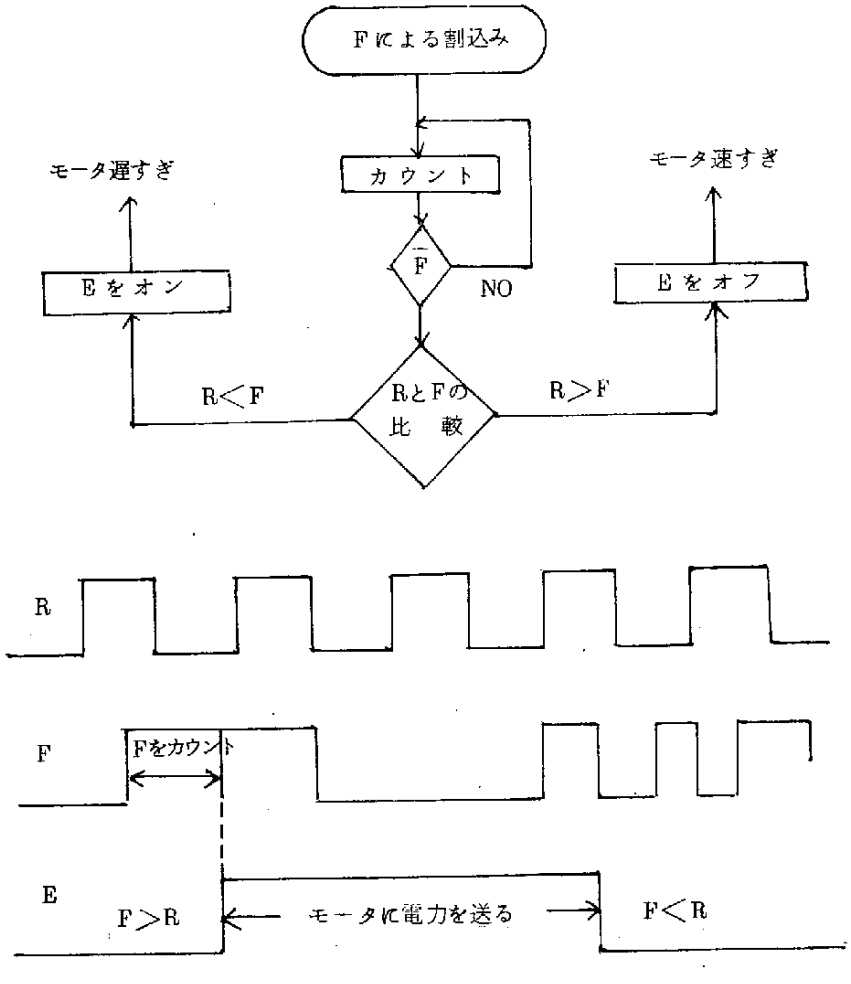
第２－５１図　マイクロコンピュータ構成



第２－５２図　エラー修正信号の発生法



第２－５３図　マイクロプロセッサの制御の流れ



第２－５４図　モータ速度のみをチェックする時のフローと信号

3. マイクロコンピュータのソフトウェア

3.1 ソフトウェアの役割

3.1.1 ハードウェア設計からソフトウェア設計へ

マイクロコンピュータの出現によりシステム設計方法が今までと比較してかなり変ってきた。従来はシステム設計の大部分はハードウェア設計（ここでは主として論理回路の設計をさす）に費やされてきた。したがって出来上った回路が正しく動作するか否かは、まずそのシステムの試作機（プロトタイプ）を作って実際に正常にシステムが動作するまで何度も回路の配線などを変えて修正しながらデバッグ（誤りを取除くこと）しなければならない。そのため1つのシステムが完成するまでかなりの労力と日数（数ケ月ないし数年）が必要だった。またシステムの機能変更をする場合にも再度配線を変更するなどして相当の期間が必要である。

一方マイクロコンピュータを利用してシステムを組んだ場合、システムをいかに制御するかはプログラムによってなされ、プログラムを組むことがシステム設計の大半を占めるようになる。

ソフトウェア・ベースでシステム設計をした場合には繁雑な配線が大幅に減少され、また動作のチェックがコンピュータを使用して容易にできる為、開発期間が大幅に短縮されると共にシステムの機能変更の際にはプログラムを書き換えるだけで新しいシステムが生まれる。

今日の如く技術の急速に進歩する時代には開発期間を長くかけて製品を作っていたのではその製品が完成する頃にはもう陳腐化してしまうから、プログラムを変えるだけで新製品が作れるマイクロコンピュータはこれからの時代に適したものと言える。

3.1.2 ソフトウェアの種類

マイクロコンピュータで使うソフトウェアという言葉には大きく分

けて2つの意味がある。1つはアプリケーション・プログラムのことをさすソフトウェアで、それによってマイクロコンピュータを使用したシステムが如何なる仕事をするかが決まる。

もう一方はこのアプリケーション・プログラムを開発する為の道具として利用されるソフトウェアでサポート・ソフトウェアと呼ばれるものである。後者に関してどのような種類があるかは第3.2項の「サポート・ソフトウェア」で詳細に述べるが、アプリケーション・プログラムに関してその種類は千差万別である。大きなダムを制御するものや身近の家庭電器やゲーム・マシンに至るまでアプリケーション・ソフトウェアの範囲は無限である。しかし、いづれのアプリケーション・プログラムも、マイクロコンピュータのCPU が解決できる機械語に変換されたリプログラムにエラーが存在するか否かを調べたのする為にはサポート・ソフトウェアを利用しなければならない。どのような手段でどのサポート・ソフトウェアを利用すればどんな効果が現れるかをユーザはよく理解して最も効果的なプログラム開発を心掛けねばならない。

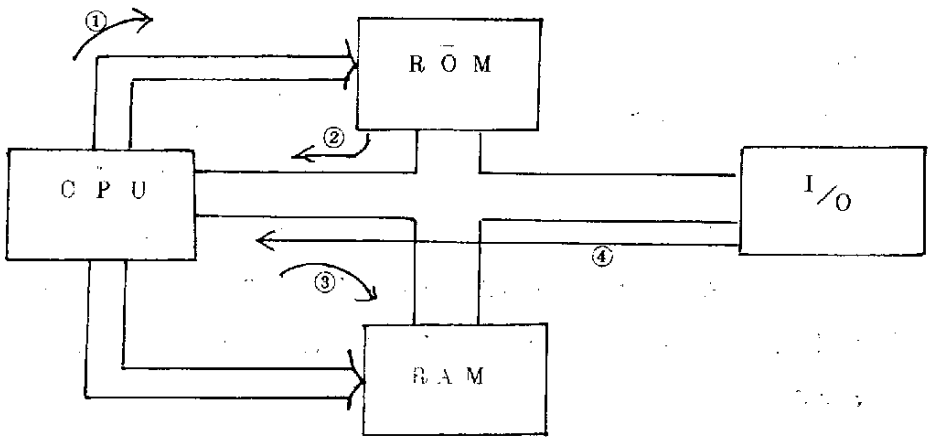
3.1.3 ROMの役割

マイクロコンピュータを使用したシステムが動作するのはほとんどCPU の働きによってコントロールされるが、そのCPU に命令を与えてCPU の動作を制御しているのがアプリケーション・プログラムが格納されているROM（リード・オンリー・メモリ）、またはRAM（ランダム・アクセス・メモリ）である。通常プログラムがもうこれ以上変更されることがなくて固定する場合には、プログラムのメモリとしてROM を用い、変更の可能性がある場合や全く違ったプログラムで同じシステムを別の目的に使う時にはRAM を用いる。

ROMはマスク式ROMとPROM（プログラマブルROMとがあり、マスク式ROM はその製造段階でCPU を制御するプログラムを決定

するものであり、PROM は製造段階ではコードが決らず購入後に書込器を使用してプログラムを書込むものである。PROM の中には一度書込んだコードを消して再書込みの出来るものもある。このようにハードウェアの中に組込まれたソフトウェアを一般にファームウェアと称している。

マイクロコンピータ・システムの代表的な構成を第３－１図に示す。ここで、システムの中における ROM の役割りを考えてみよう。ある１つの仕事をするうちで最も基本になる１サイクルは、まず①に示すように CPU から ROM の中のある番地を指すようなアドレス情報を送ると ROM から CPU にデータバスを通して②の如く機械語命令コードが送られる。CPU はそのコードを解釈して例えば③のように、CPU の中に保持していたデータを RAM(番地は CPU からアドレスバスを通して指定される）に格納して１サイクルの仕事が終了する。次のサイクルでは前と同様に①、②の順に次の命令コードを読み込んで今度は例えば④の如く入力機器から CPU にデータを入力して次のサイクルが終る。このように ROM はマイクロコンピュータを動かす指令部としての役割を果す。



第３－１図　CPUサイクルの動き

3.2　サポートソフトウェア

ここではアプリケーション・プログラムが完成されるまでに利用されるまでに利用される有効なサポート・ソフトウェアについて述べる。これらのソフトウェアが働く為に必要なコンピュータの種類と特徴については第3.3項で述べる。

3.2.1　アセンブラ

前にも述べたようにCPUにある仕事をさせる為にはCPUが解釈できる形式、つまり機械語コードで与えなければならない。例えば8080型マイクロプロセッサの命令でレジスタEの内容とアキュムレータ（レジスタA）との加算は〝10000011〟という8ビットの2進コードで表わされ、このコードがROMからCPUに送られて加算の動作が行なわれる。

しかしプログラムの全ステップ（通常数百～数千ステップ）をこの2進コードで記述していたのでは大変な労力であると共にコードの誤りなしにプログラムが作れるとは考えられない。

そこでこのように1と0による表現ではなくもっと判りやすいかつ覚えやすい記述形式が必要である。例えば先ほどの加算の場合には、〝ADDE〟と記述してEレジスタをアキュムレータに加算することにすれば簡単にプログラムが記述できる。このようにプログラムし易い記述形式でしかも機械語コードと1対1に対応したものをアセンブル言語と称する。またこのアセンブル言語で記述されたプログラムを機械語コードに変換するソフトウェアをアセブラと称する。

第3－2図にアセンブル言語で書かれたプログラムと機械語コードの対応例を示す。この図からもアセンブル言語記述法の簡単さがよくわかる。

アセンブラには利用するコンピュータの違いによりクロス・アセンブラとセルフ・アセンブラの2種類がある。

アドレス	機械語コード記述	アセンブル語記述		
			ORG	100H
		ALPHA	EQU	50H
		BETA	EQU	60H
100	0001_0001		LXI	D, ALPHA
101	01010000			
102	00000000			
103	00100001		LXI	H, BETA
104	01100000			
105	00000000			
106	00001110		MVI	C, 8
107	00001000			
108	10101111		XRA	A
109	00011010	LOOP:	LDAX	D
10A	10001110		ADC	M
10B	00100111		DAA	
10C	00010010		STAX	D
10D	00100011		INX	H
10E	00010011		INX	D
10F	00001101		DCR	C
110	11000010		JNZ	LOOP
111	00001001			
112	00000001			

第３－２図　機械語コードとアセンブル記述の比較

(1)　クロス・アセンブラ

一般的には、あるコンピュータのソフトウェアを開発する場合に、

レベルの異なる他のコンピュータ（ホスト・コンピュータと称する）を使用してプログラムを作成する時に、２つのコンピュータの言語が交わるという意味で、クロスという言葉を使用する。例えばマイクロコンピュータのアプリケーション・ソフトウェア開発をミニコンピュータ、大型コンピュータ、またはＴＳＳ（タイム・シェアリング・サービス）といったレベルの違うコンピュータを使用する時、マイクロコンピュータのアセンブル言語を読んで２進機械語プログラムに変換してくれるソフトウェアはクロス・アセンブラである。

(2) セルフ・アセンブラ

クロス・アセンブラでは対象とするマイクロプロセッサの言語翻訳に他のコンピュータを使用するが、他のコンピュータを使わずに同じマイクロプロセッサで構成されたマイクロコンピュータ・システムを使用してアセンブルした場合には、アセンブルされる側とアセンブルする側が同じコンピュータであることから自分自身の言語をアセンブルすることになり、セルフ・アセンブラと呼ばれる。

このようにクロス・アセンブラとセルフ・アセンブラはアセンブルするコンピュータが異なるだけで、入力となるプログラム（ソース・プログラム）と出力される機械語プログラム（オブジェクト・プログラム）はいづれの場合も同じものである。

(3) プログラムの記述形式

ソース・プログラムは普通レーベル欄、ニーモニック欄（命令文を書く欄）、オペランド欄、コメント欄に分けられる。ここでは読者が判り易いように８０８０型マイクロプロセッサのアセブル言語仕様を代表例として、取り上げ、例を添えて説明する。

① レーベル欄

レーベル（シンボルとも称する）は命令文にプログラマが判り易い名前をつけて分岐先の名前として利用したり、レーベルに定数を

与えて直接数値を扱わないでプログラムできる様に考慮されたものである。先頭文字が英文字とある限られた特殊記号（◎、？など）で始まり、英数字などで構成される１～５文字のレーベルをコロン（：）で結んでレーベル欄に記述する。

（正しい例）

LABEL:

SKP1:

@HERE:

（誤った例）

LABEL　（コロンで終端してない）

1SKP:　（先頭が数字）

〃NG　:　（指定外の特殊記号）

字数が５文字を超えた場合には先頭から５文字が有効になる。

（例）

LONGLABEL:

これはLONGL：と解釈される。

② ニーモニック欄

各種の命令文（後述）を記述する。

③ オペランド欄

命令文に付随したパラメータを記述する。

オペランドにはレジスタ名、シンボル、定数、演算子などが独立に、または混在して使用可能である。

（例）　MOV　A、B　（レジスタ記述）

ADI　10　（定数記述）

JMP　ST　（シンボル記述）

JZ　S−6　（相対記述）

DB　〃A〃　（リテラル記述）

④　コメント欄

コメントはプログラムの実行に何ら影響を及ぼさずプログラマが判りやすく記述する為に任意に使用できる。コメント行はセミコロン(;)で始めればそれ以後コメントとして使える。

(例)

```
XRA  A;CLEAR  ACC
JMP  START;JUMP
TO  START;DECIMAL
ADDTION
```

(4)　アセンブル言語の命令文

単にソース・プログラムをオブジェクト・プログラムに変換するだけがアセンブラの仕事であるが、プログラマが少しでも作りやすいように種々の便利な記述が出来るように最近のアセンブラは考慮されている。アセンブル言語の命令文には機械語命令文、擬似命令文、マクロ命令文、外部参照命令文などがあり、機械語命令文及び擬似命令文はどの種のアセンブラも必ず持っている。

(a)　機械語命令文

例えば第３－２図の中でLXI　D、ALPHAからJNZ　LOOPまでの各命令はすべて機械語命令文でこれらの命令によってマイクロプロセッサが何らかの仕事を必ず行なう。機械語命令文は各プロセッサによってまちまちであるが、実際のCPU の動作を最も良く表わした単語になっているのが望ましく、プログラマも間違いや勘違いをしないように注意しなければならない。

(b)　擬似命令文

記述することによってCPU の動行には何の影響も与えないがプログラマの記述性を高める為にアセンブラ対して種々の指示を与えるのが擬似命令文である。例えばやはり第３－２図でORGやEQU

などがそれである。ORQ 命令文はアセンブラの持っているロケーション・カウンタにある値を設定してそれに続く機械語命令のアドレスを決めることができる。またEQU 命令文はレーベル欄に記述されたレーベル（シンボル）にオペランド欄で記述された値を与えるものである。これによってプログラマは数字を扱わずにシンボルを使ってそれ以降のプログラムを書くことが出来る。擬似命令文もアセンブラの種類によって多種多様であるが大きく分類すると以下の如く考えられる。

① ロケーション・アドレスを設定する命令

② シンボルに値を設定する命令

③ 定数を確保する命令

④ 領域を確保する命令

⑤ 出力リスト制御命令

⑥ アセンブル終了指示命令

⑦ ホスト・コンピューター時停止命令

⑧ 条件付アセンブル指示命令

⑨ マクロ定義文　　e．t．c

第3－1表に擬似命令の例を示す

(c) マクロ参照文

マクロ参照文はマクロ定義の擬似命令MACROとENDM によって定義されたマクロ・ネームをニーモニック欄に記述することによって一連の定義されたステートメントをソース・プログラム中に展開するようにアセンブラに指示する。例えばアキュームレータの内容を2ビット右シフトするだけの簡単なマクロを考えてみる。そのマクロにSHRZ という名前をつけたとすると定義は以下の如く記述する。

```
SHRZ    MACRO  ; MACRO DEF INITION
```

第３－１表　擬似命令の例

擬似命令	機　　能	記　述　例
ORG	アセンブラのロケーション・カウンタにアドレスをセットします。	
EQU	シンボルを定義してオペランドに示されたデータを割当てます。	BETA　EQ　123 GAMMA　EQU　BETA
DB	オペランドで定義されたデータを１バイトづつ確保していきます。	DB　1FH* DB\" STRING " DB　5,-03H* *Hは16進数を意味します
DW	オペランドで定義されたデータを２バイトデータとして確保します。	DW　1F00H DW　START DW　1000.2000
DS	オペランドで定義された数値分のバイト数の領域を確保します。	DS　12 DS　12H
SET	ラベル欄のシンボルの値をオペランドの値に変更します。	SW　SET　5 SW　SET　10
END	アセンブラにプログラムの終を示します	END
IF ENDJF	アセンブラはＩＦ文のオペランドが０のとき、ＩＦとＥＮＤＩＦ文の間の命令文を無視します。	IF　SW : ENDIF
MACRO ENDM	アセンブラはＭＡＣＲＯとＥＮＤＭ文の間ある命令群をＭＡＣＲＯ文のラベル（ネーム）で代表するように登録します	ROOT　MACRO : : ENDM

```
RRC     ; ROTATE ACC RIGHT
RRC     ; ONE NORE ROTATION
ENDM
```

このように定義されたマクロをソース・プログラムから参照する時は以下の如く記述する。

```
        :
        :
        LDA   DATA
        SHRZ
        ANI   3FH
```

アセンブラがSHRZというニーモニックに遭遇するとこの部分に先ほどの定義した命令を挿入して結果的には次のように記述するのと同じことになる。

```
        :
        :
        LDA DATA
        RRC
        RRC
        ANI 3FH
```

マクロとしては通常汎用性に富んだルーチンを定義するが、プログラムを作成する際に度々現われる一連のステートメントをマクロとして定義しても構わないし、一般的な四則演算のプログラムを大きなマクロとして登録してもよいだろう。結局マクロの利点としては①同じ命令群を何度も記述せずに済むからプログラム作成の効率が上る。②サブルーチンを永久的なマクロとしてファイルに保存しておけば新たにそのルーチンをコーディングする必要がなく、ソフトウェア・パッケージとして財産的な価値が出てくる。（第3.2.5項

のマクロ・メインテナンス・プログラム参照）

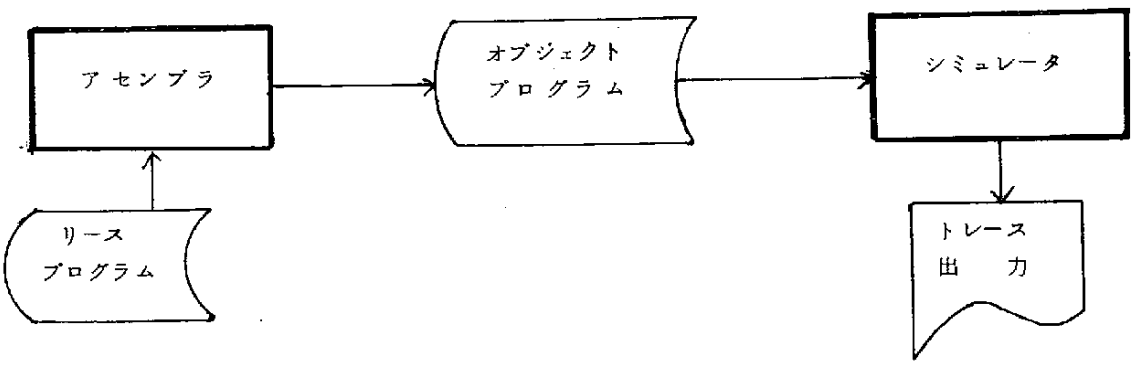
3.2.2　シミュレータ

出来上ったプログラムをアセンブラを使用して機械語に変換してもそれを直ちにROM などに書込んでシステムを働かすわけには行かない。プログラムには必ずと言ってよいほどエラーがあるからである。そこでROM にプログラムを書込む前にソフトウェア的にコンピュータを使用してデバッグできればプログラムの妥当性を極めて少ない費用で調べることができる。

シミュレータはホスト・コンピュータの記憶エリア中に対象とするマイクロプロセッサと同じ構成のレジスタやフラグ・メモリなどを擬似的に設定し、実行命令もプロセッサと同じ動作をするように設計されている。そして単に命令実行をシミュレートするだけでなくプログラムの任意の箇所で実行を中断し、その時のレジスタやメモリの内容をダンプ（印字出力）したり変更したりすることができる。第３－３図にアセンブラとシミュレータの入出力関連図を示します。

シミュレータを使って効果的にプログラムのデバッグが出来るか否かはシミュレータの持つコマンドが機能的かどうかで決まる。

ここでは μCOM －８（日電）のシミュレータを例にとり説明する。



第３－３図　アセンブラとシミュレータの関連図

(1) メモリ設定コマンド

シミュレートする時、最初にメモリをどの様に使用するかを決定する。つまりRAM として使うのかROM として使うのか、または未使用の空き領域なのかを指定する。これによってプログラムにエラーがあって誤まった動作（例えばROM や空き領域に書き込もうとした場合など）をするのがチェックできる。

① ROMコマンド

ROM としてアクセスされる領域とアクセス時間の大きいPOM の場合にはリード遅延クロック数を設定する。

② RAMコマンド

RAM としてアクセスされる領域とRAM のリード及びライト遅延クロック数を設定する。

リード、ライトの遅延クロック数を設定することによりかなり実際のシステムの働きに近いものがシミュレートできる。

(2) メモリ操作コマンド

指定されたメモリ領域の内容を変更したりダンプするコマンドである。

① CLEARコマンド

ROM RAM コマンドで設定された領域の内容を0クリアする。

② CHANGE コマンド

メモリの内容を変更する。

③ DUMPコマンド

メモリの内容を印字出力する。

(3) オブジェクト・プログラム制御コマンド

アセンブラで作成されたオブジェクト・コードをシミュレータに入力し展開させたり、デバッグして完成したオブジェクト・コードを出力してアセンブラからやり直さずに済む。

① LOADコマンド

シミュレータをRUN する以前に必ずオブジェクト・プログラムをシミュレータ内のメモリに展開しなくてはいけない。

② OBJECTコマンド

シミュレータ内のメモリに完成されたプログラムをLOAD コマンドで入力した時と同じ形式で出力する。

(4) シミュレーション・レポート・コマンド

① TRCONコマンド

シミュレーション結果を印字する開始アドレスを設定する。

② TRCOFコマンド

トレース結果を印字しているのを終了するアドレスを設定する。

TRCONコマンドで設定されたアドレスがシミュレートされたプログラム・カウンタの内容と一致した時からレジスタ、フラグの内容のアドレス、命令語などを印字開始し、TRCOFコマンドで設定されたアドレスに至るまでトレース結果を印字する。

(5) 割込み制御コマンド

外部からの割込み要求は実際にはプログラムのどの時点で発生するかは判らないが、シミュレータでは一応割込みの起る箇所をあらかじめ設定しておく。

① INTERRUコマンド

割込みを発生するアドレスと割込み命令を設定する。

(6) 入力デバイス制御コマンド

シミュレータはマイクロプロセッサの入力命令をシミュレートすることも出来る。トレース中に入力命令の実行が起るとシミュレータはユーザに入力データを要求してくる。この際ターミナルやコンソールからその都度直接データを入力するよりもあらかじめ入力データの列を記憶媒体上に作成しておいた方が効果的である。

① INPUTコマンド

IN 命令で読込まれる入力リストを作成する。入力デバイス毎（デバイスアドレス０～２５５）に入力データ列が用意できる。

② INPUTDコマンド

登録した入力データの内容を出力して確認する。

③ INPUTCコマンド

登録した入力データの内容を変更するのに用いる。

(7) シミュレーション制御コマンド

① BPコマンド

BP（ブレーク・ポイント）コマドはプログラムをシミュレート中に指定した箇所で実行を一時停止させる。停止後レジスタ、フラグ、今までの実行クロック数などを出力する。この後にメモリの内容を確かめたり変更するコマンドが使える。ブレーク・ポイントは最大８ケ所まで指定可能である。

② STEPコマンド

実行したステップ（１命令単位）をカウントしていてSTEPコマンドで指定した数を超えたらシミュレータを一時停止させる。これはアプリケーション・プログラムが終りのないループに飛込んでしまってユーザの手から離れてしまうのを防ぐ為のコマンドである。

③ GO コマンド

シミュレータに指定したアドレスからシミュレートを開始するように指示する。

(8) レジスタ・フラグ設定コマンド

CPU の内部レジスタ（A、B、C、D、E、H、L）、フラグ、プログラム、カウンタ、スタック・ポインタ、およびスタック・ポインタの下限値を設定する。

(9) 特殊コマンド

① CANCELコマンド

TRCON、TIME、STEP、レジスタ、スタック、ポインタの下限設定値などをキャンセルまたはクリアする。

② DISPLAYコマンド

各コマンドで設定された値を出力して確認する為に用いられる。

③ JOBコマンド

シミュレータの実行を先頭に戻し新しいシミュレーションの実行に移る。

④ QUITコマンド

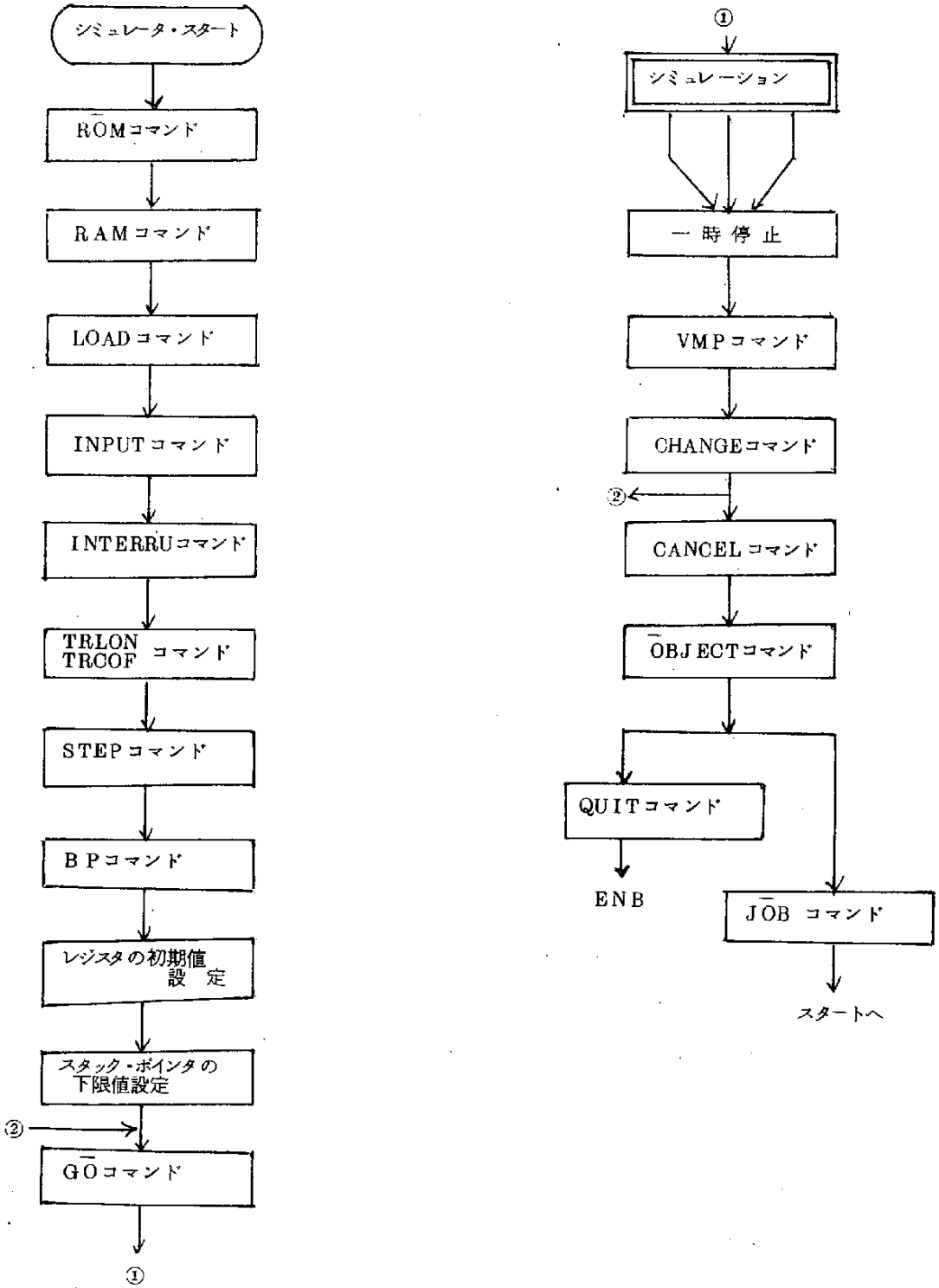
シミュレータ・プログラムの終了を指示する。

MCOM －8シミュレータの場合にはシミュレーション・レポート・ファイルを設けてトレース結果を全部一旦レポート・ファイルに書き込んでおいて後で種々のサーチ・コマンドを使って希望する箇所を見ることができる。第3－4図にはシミュレータを実行させる場合の通常の設定順序を示す。必要のないコマンドは指定しなくても良い。

3.2.3 コンパイラ

アセンブル言語でアプリケーション・プログラムを作成する場合には機械語命令とアセンブルされたオブジェクト・コードは1対1に対応している。またプログラマはCPU の内部のレジスタの状態や外部のデータをストアするメモリの状態を常によく頭に入れてプログラムを作成しなければならない。そんな煩わしいことをコンパイラ（高級言語）は解決してくれる。大型コンピュータなどでコンパイラと称される言語にはFOR、TRAN、PL／1、BPL、COBOL などがあるがマイクロコンピュータの場合、現在使用可能なコンパイラはPL／1をマイクロコンピュータ用にモデル・チエンジしたPL／M コンパイラがある。

PL／M 言語を使用してプログラムを作成した場合の大きな利点は



第3－4図　シミュレータの操作手順

コーディシグのステップ数が大幅に減少することである。従ってプログラムの開発期間や労力が相当短縮される。

例えばPL／M で変数AとBの積にCを加算して結果を変数Dとする演算は

$$D = A * B + C$$

と簡単に記述されるがアセンブル言語で同じ事をやろうとすればかなり大変である。少なくとも２５～３０ステップのコーディングが必要になります。これだけを取上げてもPL／M コンパイラの有効なことが判る。

しかしPL／M の短所としてコンパイルした後のオブジェクト・プログラムの大きさがアセンブラ言語の場合と比較すると数倍大きい事があげられる。つまりプログラムを格納するROM の領域が数倍（1.5～８倍程度）必要になるということである。これについては今後の新コンパイラの開発やPL／M コンパイラの改良を待たねばならない。

第３－2表にPL／M コンパイラの保有語の一覧表を第３－５図には同じプログラムをアセンブル言語で記述した場合とPL／M 言語で記述した場合の比較を示す。

3.2.4　リンケージ・エディヤ

現在多くのアセンブラやコンパイラの出力するオブジェクト・プログラムはそのまま絶対番地形式をとっているから、すぐにマイクロプロセッサを動作させるのには都合が良いが、全てのルーチンが１つのアセンブル単位（１つのソース・プログラム）に含まれていなければならない。またPL／M で記述したメイン・プログラムとアセンブル言語で記述したサブルーチンを結びつけることはできない。そこで登場するのがリンケージ・エディタである。汎用的なサブルーチンなどをアセンブルまたはコンパイルされたオブジェクト形式でライブラリとして保有しておき、メイン・プログラムでは単にこのサブルーチン

第３－２表　PL/Mコンパイラ保有語一覧

条件テスト文
- IF
- THEN
- FLSE

手続き定義文
- DO
- PROCEDURE
- INTERRUPT
- END

データ宣言文
- DECLARE
- BYTE
- ADDDESS
- LABEL
- INITIAL
- DATA
- LITERALLY
- BASED

無条件分岐文と回数制御文
- GO
- TO
- BY
- GOTO
- CASE
- WHILE

サブルーチン入出力文
- CALL
- RETURN

実行停止文
- HLT

割込み制御文
- ENABLE
- DISABLE

論理演算子
- OR
- AND
- XOR
- NOT

演算子
- MOD
- PLUS
- MINUS

プログラム終了文
- EOF

組込み関数
- CARRY
- DEC
- DOUBLE
- HIGH
- INPUT
- LAST
- LENGTH
- LOW
- MEMORY
- OUTPUT
- PARITY
- ROL
- ROR
- SCL
- SCR
- SHL
- SHR
- SIGN
- STACKPTR
- TIME
- ZERO

A PL/M Bubblesovt

```
DECLARE A(256) BYTF;
 (N, I, T1, T2, SWITCHED)BYTE;
SWITCHED=1;
DO WHILE SWITHCHPD;
   SWITCHED=0;
   DO I=1 TO N-1;
    T1=A(I);  T2=A(I+1);
    IF  T1>T2  THEN
        DO;
           A(I+1)=T1;
           A(I)=T2;
           SWITCHED=1;
       END;
    END;
 END;
 EOF
```

メモリ使用量　132バイト
（N, ARRAY除く）

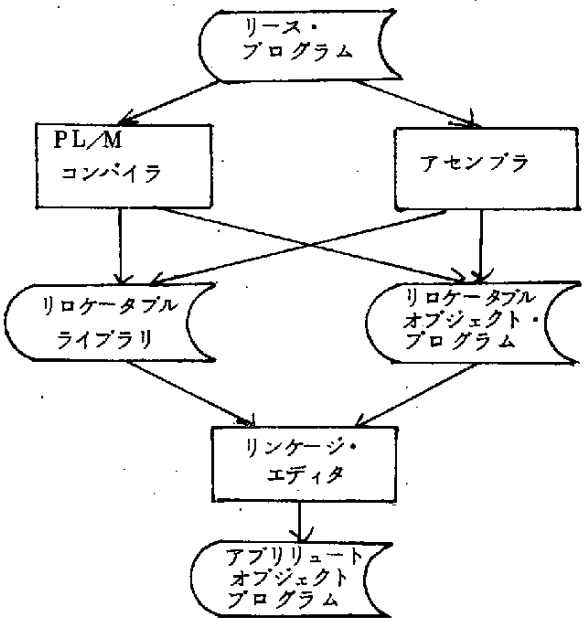
AN ASSEMBLER Bubblesovt

```
        MVI   D, 1
LAB1;MOV   A, D
        ADI   0
        JZ    LAB2
        MVI   D, 0

        LXI   H, N
        MOV   B, M

        LXI   H, ARRAY
LAB3;DCR   B
        JZ    LAB1
        MOV   A, M
        INR   L
        CMP   M
        JP    LAB3
        MOV   C, M
        MOV   C, A
        DCR   L
        MOV   M, C
        INR   L
        MVI   D, 1
        JMP   LAB3
LAB2;HLT
N    ;DB    0
ARRAY;DS    256
        END
```

メモリ使用量（38バイト）
（N, ARRAY除く）

第３－５図　アセンブラとコンパイラのプログラム比較

をその名前で呼ぶだけにする。メイン・プログラムを機械語に変換したものと先ほどのライブラリをリンケージ・エディタで結合すれば最終的な絶対番地形式のプログラムは出来上りである。この場合リンケージ・エディタの入力となるオブジェクトプログラムは絶対番地形式（アブソリュート・オブジェクト）に対して相対番地形式（リロケータブル・オブジェクト）と呼ばれる。第３－６図にアセンブラ、コンパイラ、リンケージ・エディタの関連図を示す。

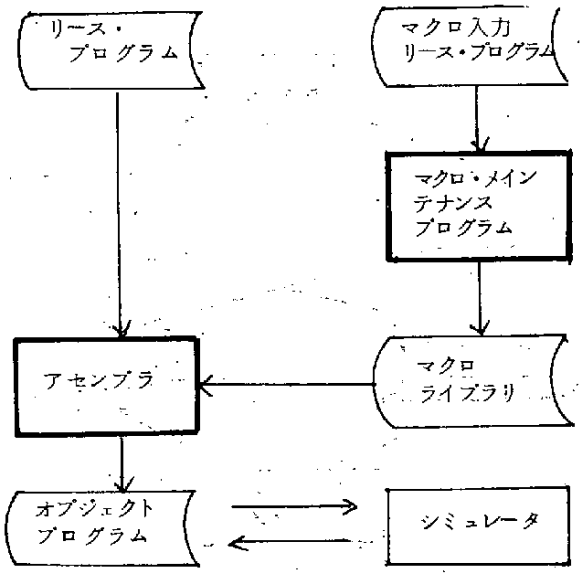


第３－６図　アセンブラ、コンパイラ、リンケージ・エディタの関連図

3.2.5　マクロ・メインテナンス・プログラム

3.2.1 項のアセンブラの説明でマクロ機能につい少し述べたが、リンケージ・エディタが無い場合にはマクロ機能を取り入れてソース・プログラム・レベルでサブルーチンの財産管理をするのが大きな意味を持って来る。この際にマクロ・ライブラリに汎用的なプログラムを登録、削除の管理をするのがマクロ・メインテナンス・プログラムの仕事である。

ソース・プログラムの中で一時的に定義されるマクロを一時マクロ
マクロ・ライブラリの中で管理されるマクロを永久マクロと呼ぶ。マ
クロ・ライブラリの形でサブルーチンを提供する場合にはTSS（タ
イム・シェアリング・サービス）形態がファイル・ベースでデータの
授受を行なうことから最も適しているとも言える。第３－７図にマク
ロ・メインテナンス・プログラムと他のソフトウェアの関連図を示す。



第３－７図　マクロ・メインテナンス・プログラムとの関連図

3.2.6　紙テープ出力プログラム

アプリケーション・プログラムがシミュレータによって正しく動作
することが確認されたらもうほとんどプログラム走行上のエラーはな
いと思われるが、シミュレータでは入出力のタイミング等に関するシ
ミュレートは出来ないので最後には実際のシステムで確認しなければ
ならない。方法としてはデスク・トップ型のハドウェア・シミュレー
タにプログラムをロードしてシステムを動作してみるやり方とPROM

にプログラムを格納してシステムと接続し動かすやり方などが一般的であるが、いづれにしても紙テープを媒体としてオブジェクト・プログラムを一旦ホスト・コンピュータから出力してプログラム移送を行なう。

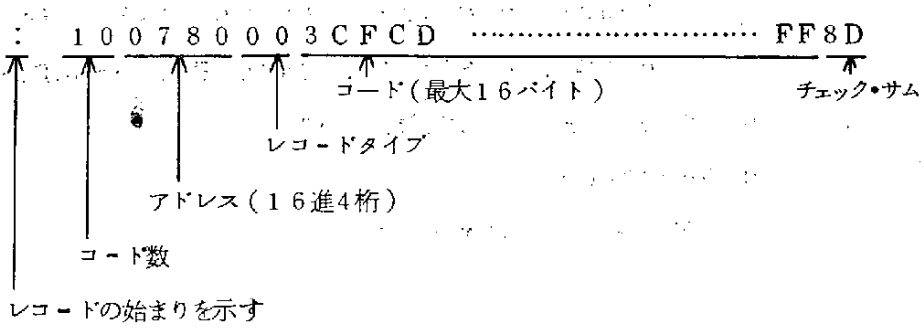
紙テープのパンチ形式はPROM ライタの入力形式としてはBNPF形式、ハードウェア・シミュータへの入力としてはHEX 形式などが最もポピュラーである。

(1) BNPF形式

ASCII コードのBとFで囲まれたN（２進数の０を表わす）とP（２進数の１を表わす）でオブジェクト・コードの１バイト分を構成する。BとFで囲まれた箇所以外の文字は全て無視される。BNPF形式にはアドレス情報が含まれていないから出力されるバイト数は256、512、1024、2048などの単位分のコードが満たされている必要がある。第３－８図にBNPF 形式の紙テープをテレタイプで印字した時のコード形式を示す。

(2) HEX形式

HEX 形式は１バイト８ビットの上位と下位４ビットづつを１６進のASCIIコードで表わしたものである。この形式は普通ハードウェア・シミュレータ等の入力紙テープ形式となる。代表的なHEX 形式の構成は次のようになる。



0000	BNNPPNNNPF	BNNPPNNPNF
0002	BNNNNNNNNF	BPPNNNNPPF
0004	BNNNPPNNPF	BNNNNNNNNF
0006	BNNNNNPPNF	BNNNNNNNNF
0008	BNNNPPPPNF	BNNNNPNNPF
0010	BNPPPPNNPF	BNNNPPPPPF
0012	BNPNNPPPPF	BNNNPPPNPF
⋮	⋮	⋮
0254	BNNNNNNNNF	BNNNNNNNNP

第３－８図　BNPF形式の例

コード数は１６進２桁で示される。コード数が０のときオブジェクトの終了である。

レコード・タイプは常に００ですが将来の拡張のために確保しておく。

チェック・サムはコロン（：）を除く全てのデータを初期値０から順に８ビット単位で減算した値である。この値も１６進２桁である。

第３－９図にHEX 形式の例を示します。内容は第３－８図と同じである。

```
:10000000313200C3190006001E09791F4F1DC87840
:10001000D21400821F47C30A000E17164ACD0600ED
:080020000E5E162BCD060076E2
:0000000000
```

第３－９図　HEX形式の例

3.2.7　システム・モニタ

システム・モニタ・プログラムはハードウェア・シミュレータの中のROM またはPROM に常駐し、オブジェクト・形式のアプリケーション・プログラムをメモリの中にロードしたり、メモリの内容を表示、変更する働作の制御をする。

主な機能

① オブジェクト・プログラム・テープを読み込みメモリにロードする。

② メモリまたはレジスタの内容をテレタイプ上に表示する。

③ メモリまたはレジスタの内容をテレタイプから変更する。

④ メモリの内容を紙テープにパンチする。

システム・モニタの働きによってコンソール・パネルから操作せずにテレタイプを介して会話形式で上記の動作を行なえる為、ハードウェア・シミュレータが使いやすいものになっている。

3.2.8　エディタ

アセンブラまたはコンパイラの実行によってソース・プログラムにエラーを発見した場合にはソース・プログラムを修正しなければならない。そこでエディタが必要になる。エディタはソース・プログラムを読込んでその内容を1行毎に修正したり、削除、挿入して新しいソース・プログラムを出力する。

TSS の場合にはソース・プログラムはファイルに格納されており種々のエデット・コマンドを備えているので比較的容易に修正が行える。

大型のバッチ・コンピュータの場合にはユーティリティ・プログラムを使用してMT（磁気テープ）からMTへ、またはファイルからファイルへコピーしながら修正する。

ソース・プログラムが紙テープにパンチされている場合にはコンピ

ュータ（この場合はミニコンまたはハードウェア・シミュレータ）のメモリにロードして修正後に再び紙テープにパンチする。

主な機能は

① オブジェクト・プログラムをメモリ内にロードする。

② メモリ内容のダンプ、チェンジ、クリア等を行なう。

③ レジスタの内容のダンプ、チェンジを行なう。

④ メモリ内のオブジェクト・プログラムを紙テープにパンチする。

⑤ 実行を1時停止するブレーク・ポイントを設定する。

⑥ ブレーク後の実行を再開する。

⑦ 設定されたブレーク・ポイントをリセットしてブレーク番地の内容を基の命令に戻す。

コマンドは機能的にはリフトウェアのシミュレータと同等のものであるが、シミュレート形能が単なる擬似的なものか実際のハードウェア上で動作するかの大きな違いがあり、入出力のタイミング等も充分チェックされる。

3.2.9 ハードウェア診断プログラム

ハードウェア・シミュレータ内のCPU、メモリ、およびTTYなどのチェックを行なうプログラムで保守用としてのプログラム価値がある。

(1) CPUテスト・プログラム

CPUの全命令の機能をチェックして誤りがあればTTYにエラーがあった命令を出力する。

(2) メモリ・テスト・プログラム

RAMメモリの全エリアにリード・ライトを行ない不良箇所が存在した場合にはその番地をTTYに出力する。

3.3　ホスト・コンピュータの種類と特徴

3.3.1　TTSによるプログラム開発

端末器（ターミナル）を導入して利用契約を結ぶだけでコンピュータが利用できるTTSが最近普及してきた。マイクロプロセッサ用アプリケーション・プログラム開発用のソフトウエアをサービスしているTTSは日本国内では電々公社がDEMOS－Eのプログラム・ライブラリの１部として、また国際的には多くのマイクロコンピュータ製造メーカがG.E（General Electric）の世界的なネットワークMARK－Ⅳを利用して約４００もの都市にソフトウエアを利用可能にしている。TTSの普及度が高いアメリカではアメリカ国内向の商用サービスが１０社以上もあり、特にVCS、TYMSHAREなどを利用して多くの半導体メーカが安い料金で広範囲にソフトウエア・サポートを行なつている。TTSを利用してプログラム開発を行つた場合の特徴としては、

① コンピュータ導入経費が殆んどない。

② １日２４時間のうちいつでも使用できる。（DEMOS－Eの場合は8：００～２０：００の１２時間）ターン・アラウンド時間が短かい。

③ 中央のコンピュータは超大形であるため種々の機能がサポートされている。

④ ソース・プログラムの修正が簡単にでき、各サポート・ソフトウエアの起動はプログラムの名前を指定するだけでできる。

⑤ 会話形式の実行や一括処理形態の２モードが利用できる。

⑥ データの送受手段としても使える。

主なTTSについてそのサポート内容を説明する。

(1) DEMOS－E（科学技術計算サービス）

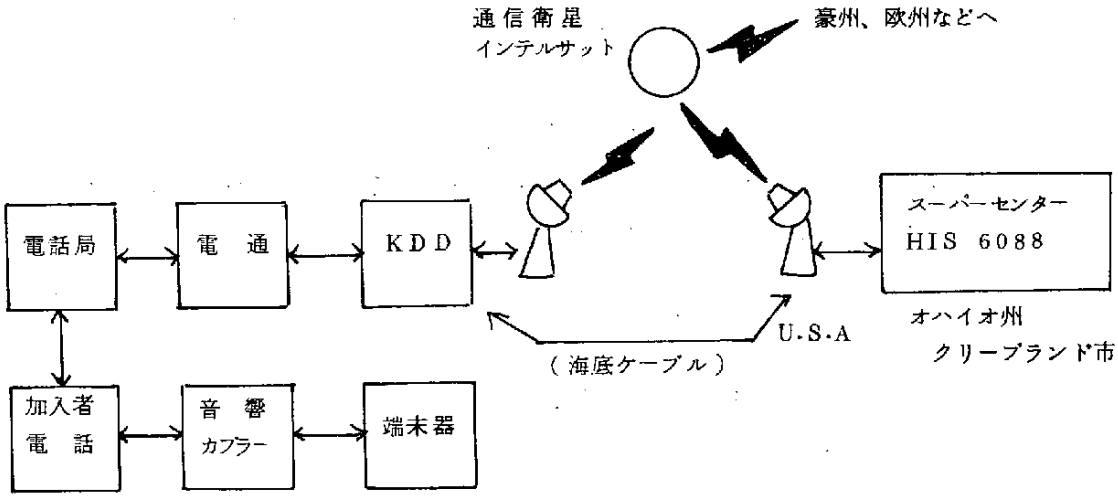
DEMOS－Eでは８ビット系のマイクロプロセッサ用ライブラリとしてPMPアセンブラ、PMPシミュレータ、PMP紙テープ出力プログラムを電々公社が提供してサービスしている。（PMPは

Prgram fr Micr Prcessr の略です。）クロス・アセンブラはマクロ命令が使用でき、いづれのプログラムも即時、一括両形態で使用可能である。また端末器として1200ボー（Baud）高速機が使える。

(2) GE−MARK Ⅲ

主なマイクロプロセッサ・メーカは全てMARK Ⅲに各社のプロセッサ用開発、サポート・ソフトウェアを登録してサービスしている。国内では日本電気、東芝が、また外国ではインテルを始めTI、モトローラ、フェアチャイルド、N、S、RCA、ロックウェル、GIなどが揃ってサポートしている。

第3−10図に国内からのMARK Ⅲ通信経路を示します。国内でのGEの代理店は電通が行っている。



第3−10図　MARKⅢ通信経路

3.3.2　ミニコンによるプログラム開発

ミニコンを使用してアプリケーション・プログラムを開発するにはアセンブラ、シミュレータ、エディタが最低限必要である。ミニコンとTTYの最少システム構成でメモリ容量は普通アセンブラで4KB以上、シミュレータで16KB以上は必要である。また入出力機器とし

てPTR(高速紙テープリーダ)、PTP(高速紙テープパンチャ)、LP(ライン，プリンタ)があれば申し分ない。

ミニコンの場合には大型コンピュータと違つて共通する言語が乏しいのでサポートされるソフトウェアもそのミニコン専用のものとなつてしまう。したがつて1つのミニコンでA社のプロセッサのプログラムは作成できてもB社のプロセッサのプログラム作成は出来ない場合が多い様である。

ミニコンをマイクロコンピュータ用のホスト・マシンとして使うのはその規模から言つて丁度手頃なものと言える。

3.3.3　大型バッチ・コンピュータによるプログラム開発

マイクロプロセッサを供給するメーカなどから最近はユーザの所有する大型、中型の一括処理コンピュータ用に比較的低いレベルのFORTRAN言語を使つて記述されたサポート・ソフトウェアをサービスしてユーザの便宜をはかつている。

例えばインテル社ではアセンブラ、シミュレータ、PL/Mコンパイラなどを ANSI-FORTRAN 言語という低い言語レベルのFORTRANで記述して、磁気テープで販売している。ユーザはそのプログラムをコンパイルする際にユーザの持つシステムに合うように入出力関係のステートメント(READ,WRITE文)の機器番号を変更するだけで適用できる。コンピュータの規模としても1語の32ビット以上で16K語以上のメモリ容量があれば使用可能である。

TTSと比較して入出力のスピードは遙かに高速であるから、問題はターン・アラウンド・タイムとユーテイリテイとしてエデイタが有るか否かにある。コスト的には TSS よりは安く使用できると思われる。

3.3.4　ハードウェア・シミュレータによるプログラム開発

市販されているハードウェア・シミュレータは TTY を含めて100～150万円と値段はかなり高いものであるが、利点はこれがあればともかくプログラム作成からシステムの動作テストまで全て出来るとい

うことである。開発効率を上げるには更にPTR，PTPを追加する。さらにLPやフレキシブル，ディスクが接続できるものであれば開発効率はミニコン以上と言える。

常駐または外部からロードして利用できるソフトウェアとしては

① システム・モニタ

② セルフ・アセンブラ

③ ソース・エディタ

④ デバッガ

⑤ ハードウェア診断プログラム

などがある。

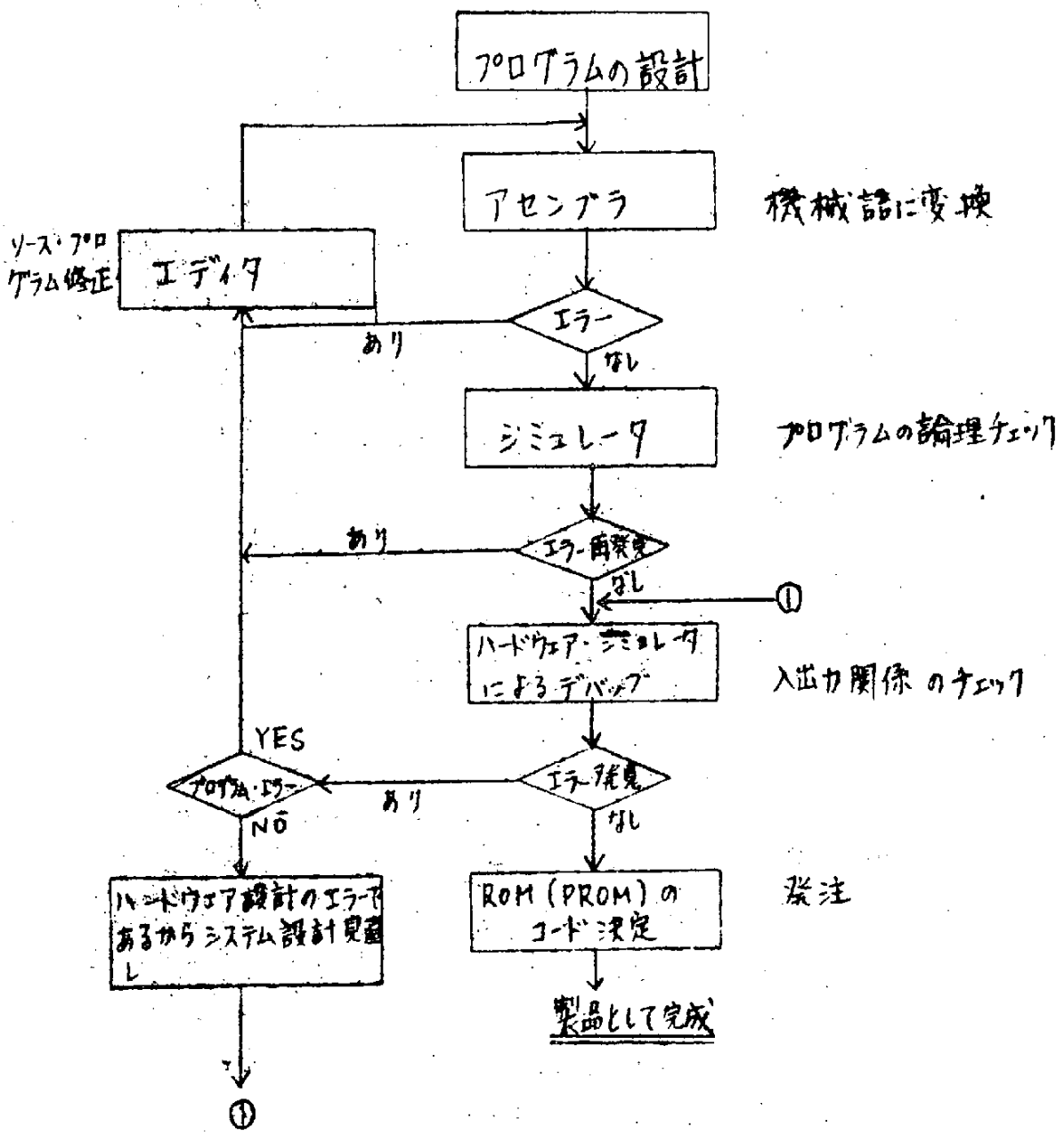
3.4 アプリケーション・プログラムの開発手順

第3－11図に代表的なアプリケーション・プログラムの開発手順を示す。

プログラムを記述する際にはステップ数が最少になるように努めるのが必要となるROMの個数を減らす意味で重要なことである。

アセンブラにはTSS，バッチ型コンピュータ，ミニコン等によるクロス・アセンブラやハードウェア・シミュレータ上で動作するセルフ・アセンブラがあるが、いずれを使用するかはその都度の経済性、効率などを考慮して決める。シミュレータで論理的なプログラム・エラーを除いた時点でそのプログラムはほとんどエラーはないが、入出力のチェックをする為にハードウェア・シミュレータにプログラムを移してリアル・タイム（システムと直結した）のデバッグを行なう。いずれの場合もエラーが発見されたらソース・プログラムをエディタを使って修正するが、シミュレータやハードウェア・シミュレータでも簡単な修正（命令の変更程度）なら行える。

第3.2項で述べた他のプログラムも組合わせて使用すればより効果的にプログラム開発が行える。



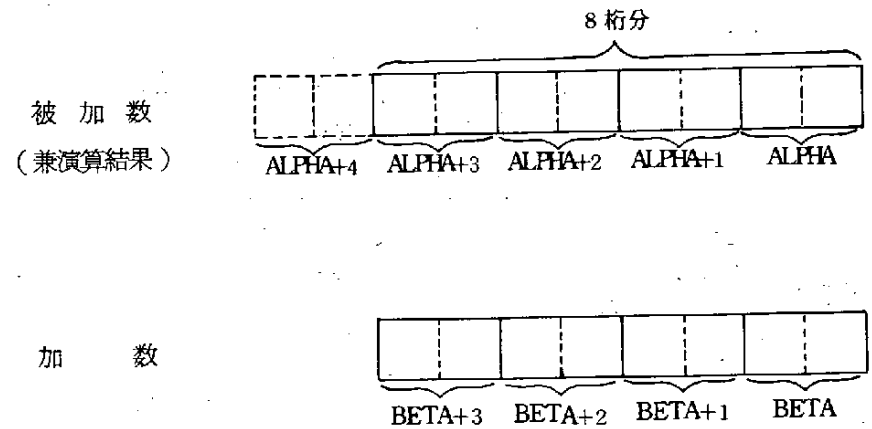
第3－11図　アプリケーション・プログラムの開発手順

3.5　プログラミング例

サポートソフトウェアの種類と内容については大体理解できたと思うが、アプリケーション・プログラムの作成はどのように行なうのかプログラミング例を述べてみよう。ここでは仮に８０８０型マイクロプロセッサを対象にしたプログラムの作り方について例を挙げる。

3.5.1　１０進加算プログラム

メモリの中に格納されている８桁の１０進数の加算プログラムを作ってみよう。１０進数の１桁は４ビットで表わされるからメモリの１バイト８ビットの中には２桁の１０進数が格納されることになる。従って８桁分として４バイト占有する。被加数の最下位バイトにALPHAという名前を、加数の最下位バイトにBETAという名前をつける。被加数の領域には演算結果も格納されるものとする。



被加数のストアされているメモリのアドレス用にＤ，Ｅレジスタを、加数のアドレシング用にＨ，Ｌレジスタを使用する。

```
LXI  D，ALPHA
LXI  H，BETA
```

１回の加算で２桁分（１バイト）行なうから８桁分のカウンタとして４をＣレジシタにセットする。

```
MVI  C，4
```

キャリをクリアしてからアキュムレータ（Aレジスタ）にD，Eレジスタでアドレスされる被加数2桁分をロードしてH，Lでアドレスされる加数との加算を行ない10進補正後に被加数の領域に結果としてストアする。

```
        XRA   A
LOOP1 : LDAX  D
        ADC   M
        DAA
        STAX  D
```

これで最下位2桁分の10進加算が終了した。後はこれと同様に4回繰返せばよいわけだが、被加数と加数のアドレスを、＋1にして次の演算にそなえ、更にカウンターを－1にする。もしカウンタが0でなければ8桁分終了していないので演算処理を続行する。

```
        INX   D
        INX   H
        DCR   C
        JNZ   LOOP1
```

カウンタが0になればこのループを抜け出す。演算結果が8桁を超えた場合にはキャリが出るから、ALPHA＋4番地に1をストアし、キャリがない時は0をストアする。

```
        MVI   A,0
        JNC   SKIP1
        INR   A
SKIP1 : STAX  D
```

第3－3表にプログラム・リストを示します。

第３－３表　１０進８桁加算プログラム

レーベル	ニーモニック	オペランド	〔注　釈〕
	LXI	D，ALPHA	被加数のアドレスを(D)(E)に設定
	LXI	H，BETA	加数のアドレスを(H)(L)に設定
	MVI	C，4	カウンタ＝4
	XRA	A	キャリをクリア
LOOP1：	LDAX	D	〔(D)(E)〕→A
	ADC	M	〔(H)(L)〕＋A＋キャリ→A,キャリ
	DAA		加算の１０進補正
	STAX	D	A→〔(D)(E)〕
	INX	H	(H)(L)＋1→(H)(L)
	INX	D	(D)(E)＋1→(D)(E)
	DCR	C	(C)－1→(C)
	JNZ	LOOP1	加算終了でなければＬＯＯＰへ
	MVI	A，0	(A)＝0
	JNC	SKIPI	結果が８桁以内ならスキップ
	INR	A	(A)＋L→(A)
SKIP1：	STAX	D	ALPHA＋4番地に0または1を格納

3.5.2　コード変換プログラム

(1)　BCD（２進化１０進数）→ＡＳＣＩＩコード

５バイト分のメモリに格納された１０桁の１０進数を１０バイトのＡＳＣＩＩコードに変換して格納する。第３－４表にＢＣＤとＡＳＣＩＩのコード対応表を示す。

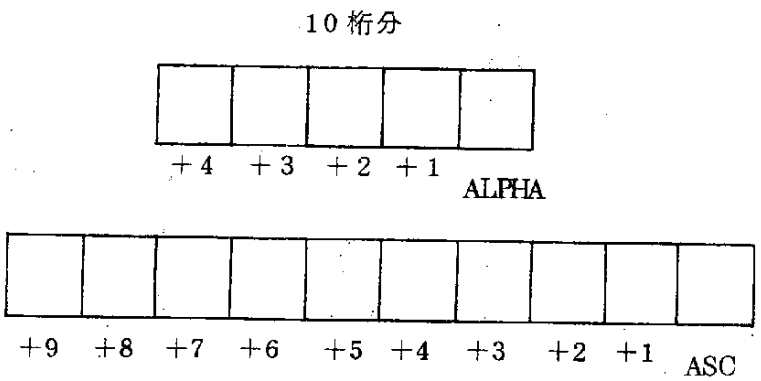
ＢＣＤコードとＡＳＣＩＩコードの下位４ビットは全く同じですから、ＢＣＤ コードに００１１を上位４ビット分として加えればコード変換できる。

１０進数が格納されたエリアの最下位２桁にＡＬＰＨＡという名前

第３－４表　ＡＳＣＩＩとＢＣＤのコード対応表

	ＡＳＣＩＩコード	ＢＣＤコード
0	1 0 1 1 0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	0 1 0 1
6	0 1 1 0	0 1 1 0
7	0 1 1 1	0 1 1 1
8	1 0 0 0	1 0 0 0
9	1 0 0 1	1 0 0 1

が、ＡＳＣＩＩに変換されたコードの最下位桁にＡＳＣという名前がついているとする。



ＢＣＤの格納されているメモリのアドレシング用にＤ、Ｅレジスタを、ＡＳＣＩＩコードが格納されるメモリのアドレシング用にH,Lレジスタを使用する。

```
LXI  D，ALPHA＋4
LXI  H，ASC＋9
```

5バイト分の変換をカウントするためにCレジスタに5をセットする。

```
        MVI   C,5
```

BCDコードけ1バイトに2桁分格納されているから上位桁（左半分）を変換中のときはキャリを1に、下位桁（右半分）のときはキャリが0であるとする。またこのキャリ・フラグを保存するのにスタックを用いるためスタック・ポインタに値を設定する。（仮にGAMMAという名前のついた値としてみる。）

```
        LXI   SP,GAMMA
        STC
LOOP2:  PUSH  PSW
```

次にアキュムレータにBCDコード2桁分をロードする。キャリが1ならば上位桁の処理であるから4ビット右シフトする。
キャリが0のときは4ビットシフトをやらない。

```
        LDAX  D
        JNC   SKIP2
        RRC
        RRC
        RRC
        RRC
```

アキュムレータの下位4ビットに対象とするBCDコードがある時点で上位4ビットを0にして、さらに1011とする。

```
SKIP2:  ANI   0FH       （Hは16進を示します）
        ORI   0B0H
```

変換されたコードをASCIIエリアに格納し、アドレスを－1にしておく。

```
        MOV   M,A
        DCX   H
```

次に保存しておいたキャリを復帰して反転し、今度処理するのが下位4ビットなのか新しいバイトの上位4ビットなのかをチェックし、後者の場合にはBCDエリアのアドレスを－1してカウンタも－1

する。カウンタが0ならば変換終了である。

```
        POP   PSW
        CMC
        JNC   LOOP2
        DCX   D
        DCR   C
        JNZ   LOOP2
        ~
```

第3－5表にプログラム・リストを示す。

第3－5表　BCD→ASCII変換プログラム

レーベル	ニーモニック	オペランド	〔注　釈〕
	LXI	SP,GAMMA	SP設定
	LXI	D,ALPHA+4	BCD最下位を(D)(E)に設定
	LXI	H,ASC+9	ASCII最上位を(H)(L)に設定
	MVI	C,5	カウンタ＝5
	STC		キャリ＝1
LOOP2：	PUSH	PSW	キャリ保存
	LDAX	D	BCD2桁分→A
	JNC	SKIP2	2桁のうち下位ならスキップ
	RRC		4ビット右シフト
	RRC		
	RRC		
	RRC		
SKIP2：	ANI	0FH	上位4ビット＝0000
	ORI	0B0H	上位4ビット＝1011
	MOV	M,A,	ASCIIストア
	DCX	H	アドレス－1
	POP	PSW	キャリ復帰
	CMC		キャリ反転
	JNC	LOOP2	(キャリ)キ0ならLOOP2へ
	DCX	D	BCD　アドレス　－1
	DCR	C	カウンタ－1
	JNZ	LOOP2	(カウンタ)キ0ならLOOP2へ

(2) ASCIIコード→BCD

10バイト分のASCIIの10進文字を5バイトにパックされたBCDコード(2進化10進数)に変換する。前の変換の逆を行なうわけであるが、各エリアに付ける名前は同じとする。

まずペア・レジスタにアドレスの初期条件を、Cレジスタに繰返し回数を設定する。

```
LXI  D,ALPHA+4
LXI  H,ASC+9
MVI  C,5
```

BCDに変換した場合ASC+9番地のコードは1バイトの上位4ビットにASC+8番地のコードは下塩4ビットに格納されるから上位に格納する場合には4ビット左シフトするが、この場合にはADD命令を4回使用してシフトする。

```
LOOP3:MOV  A,H
      DCX  H
      ADD  A
      ADD  A
      ADD  A
      ADD  A
```

勿論ADD命令の代りにPLC命令を4回使用してから下位4ビットを0クリアしても結果は同じになる。求まった上位桁のコードを一旦Bレジスタにストアしておいて、今度はASC+8番地の内容をアキュムレータにロードして下位4ビットを抽出してBレジスタの内容とパッキング(論理ORをとる)すればBCD1バイト分の変換終了である。後はカウンタが0になるまでこの処理を繰返す。

```
MOV  B,A
MOV  A,M
```

```
        DCX   H
        ANI   0FH
        ORA   B
        STAX  D
        DCX   D
        DCR   C
        JNZ   LOOP3
          ～
```

第３－６表にプログラムリストを示す。

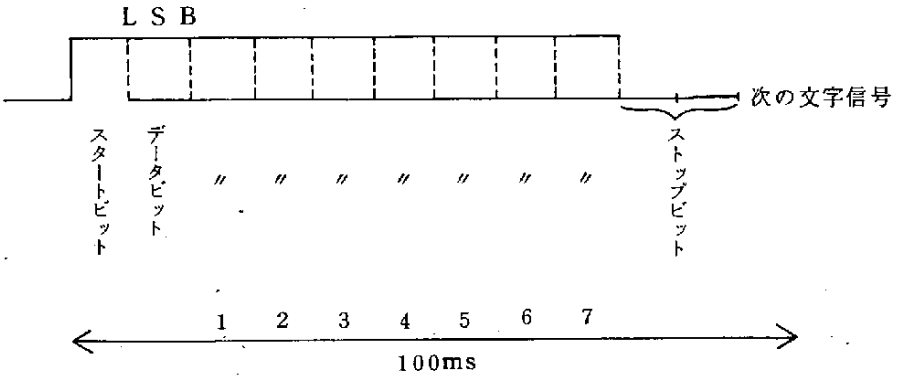
第３－６表　ASCII→BCD変換プログラム

レーベル	ニーモニック	オペランド	〔注　釈〕
	LXI	D,ALPHA+4	BCD最上位アドレスを(D)(E)に設定
	LXI	H,ASC+9	ASC最上位アドレスを(H)(L)に設定
	MVI	C,5	カウンタ＝5
LOOP3 :	MOV	A,M	A＝ASCII文字
	DCX	H	ASCIIアドレス－1
	ADD	A	4ビット右シフト
	ADD	A	
	ADD	A	
	ADD	A	
	MOV	B,A	Bに保存
	MOV	A,M	A＝ASCII文字
	DCX	H	ASCIIアドレス－1
	ANI	0FH	下位4ビット抽出
	ORA	B	上位と下位のパッキング
	STAX	D	BCDエリアにストア
	DCX	D	BCDアドレス－1
	DCR	C	カウンタ－1
	JNZ	LOOP3	（カウンタ）≠0ならLOOP3へ

3.5 3　テレタイプ制御プログラム

一般的なテレタイプASR－33から1文字タイプインしてメモリの中にストアしたり、メモリの中の文字をテレタイプに印字するプログラムを考えてみる。

まずテレタイプの入出力データは、1文字11ビットの直列型で、11ビットの周期は100mS である。第3－12図にテレタイプの入出力データ構成を示す。



第3－12図　テレタイプの入出力データ構成

データビットの8ビットはASCIIコード8ビットに対応して入出力を行なう。

純粋にソフトウェア処理だけでこのデータを入出力する時は第3－12図のように各ビットの間隔を正確に保ってやらなければいけない。従って各ビットの周期は9.09mSであるからその為のWAITルーチンが必要になる。例えば1文字入力する時は常にスタートビットが現れたか否かをチェックしていて検出した場合にはそれ以降9.09mS毎にデータを1ビットづつ計8ビット読込んで1文字の入力が終る。データはなるべく安定した箇所で読む為に各ビットデータの中心付近で読込み命令を働かせるようにWAITルーチンを細工する。

1文字出力のときは逆にスタートビットとエンドビットを前後に付

加して合計１００ｍＳになるように直列に送り出す。

このように完全にソフトウェアで入出力を行なうのはかなり大変なことであるが、この直列データを並列８ビットデータに変換してくれるインターフェイス回路をＣＰＵとテレタイプの間におけばプログラムはかなり簡単になる。この時トランスミッタ・レシーバというインターフェイス用のＩＣを使うが、プログラム作成の際にはこのＩＣが組込まれたハードウェア・シミュレータを利用するのが最も便利である。

ここでは１例として日本電気製のハードウェア・シミュレータPDA－８を使用した場合のプログラム方法について述べてみる。

ＰＤＡ－８の場合にはテレタイプとの１文字入出力を行なう際にインターフェイスの状態が準備ＯＫか否か状態フラグをチェックしてから入出力を行なう。チェックするためにはＩＮ命令で１６進の２０番地のデバイス・アドレスを指定するとアキュムレータにインターフェイス回路からその時のステータス（状態）が返ってくる。この時アキュムレータ内の各ビットの持つ意味は次の通りである。

ビット０……ＷＲＩＴＥ　ＢＵＦＦＥＲ　ＦＬＡＧ

"イ"ならばバッファが使用中

"０"ならばバッファ使用可能

ビット１……ＲＥＡＤ　ＢＵＦＦＥＲ　ＦＬＡＧ

"１"ならばバッファが使用中

"０"ならばバッファ使用可能

従ってライト・バッファ・フラグが１のときはテレタイプが印字中であるからＣＰＵからデータを送り出すことはできない。またリード・バッファ・フラグが１のときはテレタイプからデータを入力バッファに入力中であるからＣＰＵ にデータを読込むことはできない。

次に入出力ルーチンについて説明する。

(1)　入力ルーチン

テレタイプのキーボードからデータをＡレジスタに読込むルーチ

ンである。文字はAレジスタに読込むルーチンである。文字はASCIIコードを取扱うから例えばテレタイプから1というキーを押すとAレジスタにはB1（16進数）が読込まれる。

まずIN命令を使つてステータスを読込みアキュムレータのビット1をキャリに送り、リード・バッファがレディか否か調べる。レディならば1文字を読込む為にIN命令を実行する。（PDA-8の場合にはリード・バッファからアキュムレータにデータを読込む時のデバイス・アドレス指定は00である。）

プログラムは次のようになる。

```
TTYIN：IN   20H
       RRC
       RRC
       JC   TTYIN
       IN   00H
       RET
```

CALL TTYINでメイン・プログラムからこのルーチンを呼ぶとテレタイプから1文字データをアキュムレータに読込むことができる。キーボードが押されない場合には押されるまでIN20H～JC TTYINの命令を実行する。

(2) 印字ルーチン

アキュムレータのASCIIコードをテレタイプに印字するルーチンである。例えばアキュムレータの内容がB3（16進）であるとテレタイプには3が印字される。

OUT命令を使用してライト・バッファにアキュムレータから1文字送り、その後印字が終了したかどうかステータスをチェックして、印字終了ならばルーチンから抜け出す。

```
TTYOT：OUT   001
       STA   WORK
```

```
FLG  : IN    20H
       RRC
       JC    FLG
       LDA   WORK
       RET
WORK : DB    0
```

ここではアキュムレータの内容を保護するためにWORKという所に一度保存して印字終了確認後に再度アキュムレータにデータを復帰している。

(3) テレタイプ入力プログラム

キーボードから8桁の10進数を読込んでメモリにストアするプログラムである。

ASCII文字で格納されるエリアの最下位桁にASCという名前をつける。最初にこのエリアを0クリアしておく。

```
       LXI  H,ASC
       MVI  C,8
       XRA  A
LOOP4 : MOV  M,A
       INX  H
       DCR  C
       JNZ  LOOP4
```

読込んだ文字をASC+7番地からASC番地まで順に格納する。(簡単のため入力データは必ず8桁とする。)0クリアが終了した時点でHLレジスタはASC+8番地を指しているからこれをこのまま利用する。

```
       MVI  C,8
LOOP5 : DCX  H
       CALL  TTYIN
```

```
        MOV   M,A
        DCR   C
        JNZ   LOOPS
```

(4) テレタイプ印字プログラム

メモリにストアされた１０バイトのＡＳＣＩＩ数字をテレタイプに印字する。１０桁の数字を印字する前にキャリッジ・リターン(16進コードで８Ｄ)、ラインフィード(１６進コードで８Ａ)によってテレタイプを改行しておく。第３－７表にテレタイプ文字コード表を示す。

```
        MVI   A,8DH
        CALL  TTYOT
        MVI   A,8AH
        CALL  TTYOT
        LXI   H,ASC+9
        MVI   C,10
LOOP5 : MOV   A,M
        CALL  TTYOT
        DCX   H
        DCR   C
        JNZ   LOOP5
```

この場合は１０個の数字を無条件で印字するが無効なゼロをサプレスしたい時はもう一工夫必要である。

3.5.4 総合的なプログラム

今までのプログラムをまとめて作るとテレタイプから２つの１０進８桁のデータを入力して加算し、その結果をテレタイプに印字するプログラムが作成できる。

まとめる際に次の条件を与える。

(1) テレタイプ入力待ちの状態を表わす意味で＃を印字する。

(2) 実アドレスは

ＡＬＰＨＡ＝１００Ｈ

第３－７　テレタイプ文字コード表

テレタイプ文字	ASCIIコード（16進数）	テレタイプ文字	ASCIIコード（16進数）
ブランク	A 0	@	C 0
!	A 1	A	C 1
"	A 2	B	C 2
#	A 3	C	C 3
$	A 4	D	C 4
%	A 5	E	C 5
&	A 6	F	C 6
'	A 7	G	C 7
(	A 8	H	C 8
)	A 9	I	C 9
*	A A	J	C A
+	A B	K	C B
,	A C	L	C C
-	A D	M	C D
.	A E	N	C E
/	A F	O	C F
0	B 0	P	D 0
1	B 1	Q	D 1
2	B 2	R	D 2
3	B 3	S	D 3
4	B 4	T	D 4
5	B 5	U	D 5
6	B 6	V	D 6
7	B 7	W	D 7
8	B 8	X	D 8
9	B 9	Y	D 9
:	B A	Z	D A
;	B B	[	D B
<	B C	\	D C
=	B D	]	D D
>	B E	↑	D E
?	B F	←	D F
		キャリッジリターン	8 D
		ラインフィード	8 A

BETA　＝１１０H

ASC　＝１２０H

GAMMA　＝１５０H

GAMMAはスタック・ポインタの初期値である。

(3) プログラムは0番地から格納される。

(4) 各プログラム単位はサブルーチン形式にして次のような名前にする。

DECAD＝１０進８桁加算サブルーチン

BTASC＝BCD→ＡＳＣＩＩ変換サブルーチン

ASCTB＝ＡＳＣＩＩ→BCD変換サブルーチン

INPUT＝テレタイプ入力サブルーチン

PRINT＝テレタイプ印字サブルーチン

CRLF　＝テレタイプ改行サブルーチン

TTYIN＝１文字入力サブルーチン

TTYOT＝１文字印字サブルーチン

第３－８表にプログラム全体のアセンブル・リストを示す。

第３－８表　アセンブル・リスト（その１）

```
アドレス  機械語コード      レーベル    ニーモニック   オペランド
                        ;  SAMPLE      PROGRAM
                        ;
                        ;  INPUT    =  TWO  8-DIGIT
                        ;               DECIMAL  NUMBERS
                        ;               FROM  TTY
                        ;  OUTPUT       ADDED  RESULT  IS
                        ;               PRINTED  ON  TTY
                        ;
                        ;  THIS         PROGRAM  OPERATES
                        ;               ON  PDA-8  OR  -80
                        ;
0000      31            STRT:          LXI      SP,   150H
          5001
                        ;  PRINT        "CR"  ,  "LF"  ,  "#"
0003      CD                           CALL          CRLF
          BC00
0006      3EA3                         MVI      A,0A3H    ;#
          CD                           CALL     TTYOT
          D100
                        ;
                        ;  INPUT        'SUMMAND'  IN
                        ;  ASCII        AREA  AND
                        ;  CONVERT      IT   INTO   BCD
                        ;
000B      21                           LXI      H,ASC
          2001
000E      CD                           CALL     INPUT
          9700
0011      21                           LXI      H,ABC+7
          2701
0014      11                           LXI      D,ALPHA+3
          0301
0017      0E04                         MVI      C,4
0019      CD                           CALL     ASCTB
          8400
                        ;
                        ;  PRINT        "CR"  ,  "LF"  ,  "#"
                        ;
001C      CD                           CALL     CRLF
          BC00
001F      3EA3                         MVI      A,0A3H    ;#
0021      CD                           CALL     TTYOT
          D100
                        ;
                        ;  INPUT        'ADD END'  IN
                        ;  ASCII        AREA      AND
                        ;  CONVERT      IT   INTO   BCD
                        ;
```

第３－８表　（その２）

```
アドレス   機械語コード      レーベル   ニーモニック   オペランド
0024       21                        LXI          H,ASC
           2001
0027       CD                        CALL         INPUT
           9700
002A       21                        LXI          H,ASC+7
           2701
002D       11                        LXI          D,BETA+3
           1301
0030       0E04                      MVI          C,4 ; COUNTER
0032       CD                        CALL         ASCTB
           8400
                          ;
                          ; DECIMAL       ADDITION !
                          ; RESULT        IN   'ALPHA'     AREA
                          ;
0035       11                        LXI          D,ALPHA
           0001
0038       21                        LXI          H,BETA
           1001
003B       0E04                      NVI          C,4 ; COUNTER
003D       CD                        CALL         DECAD
           5600
                          ; CONVERT    BCD    INTO    ASCII
                          ;
0040       11                        LXI          D,ALPHA+4
           0401
0043       21                        LXI          H,ASC+9
           2901
0046       0E05                      MVI          C,5   ; COUNTER
0048       CD                        CALL         BTASC
           6900
                          ;
                          ; PRINT     ASCII    NUMBER
                          ; ON    TTY    AND    GO    TO    START
                          ;
004B       21                        LXI          H,ASC+8
           2801
004E       0E09                      MVI          C,9   ; COUNTER
0050       CD                        CALL         PRINT
           AF00
0053       C3                        JMP          START
           0000
                          ; SUBROUTINE          'DECAD'
                          ;
                          ; DECIMAL     ADDITION
                          ;
0056       AF             DECAD  : XRA   A; CLEAR  CARRY
0057       1A             LOOP1  : LDAX  D
0058       8E                      ADC   M
0059       27                      DAA
```

第３－８表　（その３）

```
アドレス    機械語コード    レーベル    ニーモニック    オペランド
005A        12                          STAX        D
005B        23                          INX         H
005C        13                          INX         D
005D        0D                          DCR         C
005E        C2                          JNZ         LOOP1
            5700
0061        3E00                        MVI         A,0
0063        D2                          JNC         SKIP1
            6700
0066        3C                          INR         A
0067        12          SKIP1 :         STAX        D
                                        RET
                    ;
                    ;   SUBROUTINE      'BTASC'
                    ;
                    ;   SCD  TO  ASCII  CONVERSION
                    ;
0069        37          BTASC :         STC
006A        F5          LOOP2 :         PUSH        PSW
006B        1A                          LDAX        D
006C        D2                          JNC         SKIP2
            7300
006F        0F                          RRC
0070        0F                          RRC
0071        0F                          RRC
0072        0F                          RRC
0073        E60F        SKIP2 :         ANI         0FH
0075        F6B0                        ORI         0B0H
0077        77                          MOV         M,A
0078        2B                          DCX         H
0079        F1                          POP         PSW
007A        3F                          CMC
007B        D2                          JNZ         LOOP2
            6A00
007E        1B                          DCX         D
007E        0D                          DCR         C
0080        C2                          JNZ         LOOP2
            6A00
0083        C9                          RET
                    ;
                    ;   SUBROUTINE      'ASCTB'
                    ;
                    ;   ASCII  TO  BCD  CONVERSION
                    ;
0084        7E          ASCTB :         MOV         A,M
0085        2B                          DCX         H
0086        87                          ADD         A
0087        87                          ADD         A
0088        87                          ADD         A
0089        87                          ADD         A
```

第３－８表　（その４）

```
アドレス  機械語コード  レーベル   ニーモニック   オペランド
008A      47                     MOV          B,A
008B      7E                     MOV          B,M
008C      2B                     DCX          H
008D      E60F                   ANI          0FH
008F      B0                     ORA          B
0090      12                     STAX         D
0091      1B                     DCX          D
0092      0D                     DCR          C
0093      C2                     JNZ          ASCTB
          8400
0096      C9                     RET

                ;
                ;  SUBROUTINE          'INPUT'
                ;
                ;  INPUT  8-DIGIT   DECIMAL
                ;  NUMBER  FROM  TTY  AND
                ;  STORE   IT  INTO  ASC  AREA
                ;
0097      0E08  INPUT:         MVI          C,8
0099      AF                   XRA          A
009A      77    LOOP4:         MOV          M,A
009B      23                   INX          H
009C      0D                   DCR          C
009D      C2                   JNZ          LOOP4
          9A00
00A0      0E08                 MVI          C,8
00A2      2B    LOOP5:         DCX          H
00A3      CD                   CALL         TTYIN
          C700
00A6      CD                   CALL    TTYOT;ECHO BACK
          D100
00A9      77                   MOV          M,A
00AA      0D                   DCR          C
00AB      C2                   JNZ          LOOP5
          A200
00AE      C8                   RET

                ;
                ;  SUBROUTINE          'PPINT'
                ;
                ;  PRINT  ASCII  NUMBER
                ;  STORED  IN  ASC  AREA
                ;
00AF      CD    PRINT:         CALL         CRLF
          BC00
00B2      7E    LOOP6:         MOV          A,M
00B3      CD                   CALL         TTYOT
          D100
00B6      2B                   DCX          H
00B7      0D                   DCR          C
00B8      C2                   JNZ          LOOP6
          B200
```

第３－８表　（その５）

```
アドレス  機械語コード  レーベル      ニーモニック      オペランド
00BB      C9                        RET
                      ;       SUBROUTINE          'CRLF'
                      ;
                      ;       PRINT  'CR'         'LF'
                      ;
00BC      3E8D        CRLF :      MVI             A,8DH
00BE      CD                        CALL            TTYOT
          D100
00C1      3E8A                      MVI             A,8AH
00C3      CD                        CALL            TTYOT
          D100
00C6      C9                        RET
                      ;
                      ;       SUBROUTINE          'TTYIN'
                      ;
                      ;       1  CHARACTER  INPUT  FROM
                      ;       TTY  INTO  ACC
00C7      DB20        TTYIN:      IN              20H
00C9      0F                        RRC
00CA      0F                        RRC
00CB      DA                        JC              TTYIN
          C700
00CE      DB00                      IN              00H
          C9                        RET
                      ;       SUBROUTINE          'TTYOT'
                      ;
                      ;       1  CHARACTER   PRINT   ON
                      ;       TTY  FROM  ACC
                      ;
00D1      D300        TTYOT:      OUT             00H
00D3      32                        STA             WORK
00D4      3001
00D6      DB20        FLG:        IN              20H
00D8      OF                        RRC
00D9      DA                        JC              FLG
          D600
00DC      3A                        LDA             WORK
          3001
00DF      C9                        RET
                      ;       ADDRESS   VALUE
                      ;
                      ALPHA         EQU             100H
                      BETA          EQU             110H
                      ASC           EQU             120H
                      WORK          EQU             130H
                      ;
                                    END
```

第３－９表にアセンブル後のオブジェクト・コードのリストを、
第３－１０表にはＰＤＡ－８でこのプログラムを実行した例を示す。

第３－９表　オブジェクト・コードのリスト

```
: 100 000 003 150 01C DBC 003 EA3 CDD 100 212 001 CD9 7C0
: 100 010 000 021 270 111 030 10E 04C D84 00C DBC 003 E58
: 100 020 00A 4CD D10 021 200 1CD 970 021 270 111 130 17B
: 100 030 000 E04 CD8 400 110 001 211 001 0E0 4CD 560 0E4
: 100 040 001 104 012 129 010 E05 CD6 900 212 801 0E0 9A5
: 100 050 00C DAF 00C 300 00A F1A 8E2 712 231 30D C25 775
: 100 060 000 03E 00D 267 003 C12 C93 7F5 1AD 273 000 F68
: 100 070 000 F0F 0FE 60F F6B 077 2BF 13F 026 A00 1B0 D82
: 100 080 00C 26A 00C 97E 2B8 787 878 747 7E2 BE6 0FB 021
: 100 090 001 21B 0DC 284 00C 90E 08A F77 230 DC2 9A0 04F
: 100 0A0 000 F08 2BC DC7 00C DD1 007 70D C2A 200 C9C D5F
: 100 0B0 00B C00 7EC DD1 002 B0D C2B 200 C93 E8D CDD 18A
: 100 0C0 000 03E 8AC DD1 00C 9DB 200 F0F DAC 700 DB0 06C
: 100 0D0 00C 9D3 003 2E0 00D B20 0FD AD6 003 AE0 00C 9D5
: 010 0E0 000 01F
: 010 000 000 0
```

第３－１０表　ＰＤＡ－８でのプログラム実行例

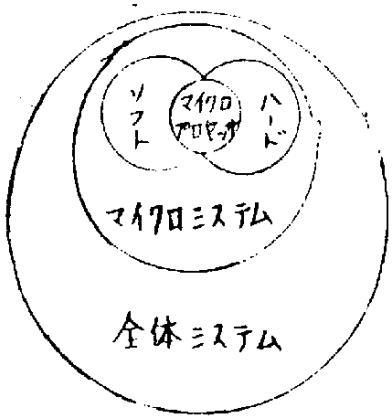
```
＊ G － 0 0 0 0 ――――→ ０番地からプログラムをスタートさせる
# 9 9 9 9 9 9 9 9 ――→ 被　加　数
# 9 9 9 9 9 9 9 9 ――→ 加　数　　}
1 9 9 9 9 9 9 9 8 ――→ 加算結果
# 1 2 3 4 5 6 7 8
# 2 3 4 5 6 7 8 9 }
0 3 5 8 0 2 4 6 7
# 0 0 0 0 0 0 8 1
# 0 0 0 8 8 8 8 6 }
0 0 0 0 8 8 8 8 7
# 4 5 6 7 7 7 7 8
# 5 4 3 2 2 2 2 2 }
1 0 0 0 0 0 0 0 0
# 8 7 6 5 4 3 2 1
# 9 9 9 9 9 9 9 9 }
1 8 7 6 5 4 3 2 0
# 1 1 1 1 1 1 1 1
# 9 9 9 9 9 9 9 9 }
1 1 1 1 1 1 1 1 0
#
```

4. マイクロシステム設計と開発

4.1 概　要

マイクロプロセッサに各種入出力装置、インターフェイス（ハードウエア）を付加し、それを制御する命令群（ソフトウエア）を付加することによつて、マイクロプロセッサは、処理を行なう物、あるいは制御を行なう物となる。これをマイクロシステムと呼ぶ。しかし、このマイクロシステムを包含するような、大きなシステムが存在するはずであり、これを全体システムと呼べば、この全体システムとマイクロシステムの関係に留意して設計に当たらなければならない。（第4－1図）

つまり「マイクロシステムは全体システムにどのような効果を上げうるか」とか「全体システムをどのように改めれば、マイクロシステムはより有効か」とかの工夫がシステム設計には必要である。



第4－1図

尚、マイクロプロセッサの特色を生かし、システムの実現をマイクロプロセッサで行なうべきか、それ以外で行なうかは、システム自体の効率にも関係し、重要なことである。

初期の応用として会計機、電卓、データ収集装置、コンピュータの周辺器機等の情報処理装置に応用されている。これらは比較的小規模なシ

ステムであり、マイクロコンピュータの安価、小型であるという特色が生かされている。

第4－1表

	ミニコン	マイクロプロセッサ	シーケンサ	リレー
作業の繰返しが出来る	○	○	○	○
増設変更に応じやすい	○	○	○	×
納期が早い	×	○	○	○
低　価　格	×	○	○	×
制御盤を小さく出来る	○	○	○	×
小型化できる	×	○	×	×
故障が少ない	○	○	○	×
保守が容易	×	○	○	×
作業記録が出来る	○	○	×	×
耐ノイズ、その他電気的制作が容易	○	○	×	×
出力パワーが大きい	×	×	○	○
算術計算が出来る	○	○	×	×

4.2　システム仕様の決定

4.2.1　仕様決定の主旨

「システムとはある目的を達成するための……」とあるように、まず目的を明確にしなければならない。

すなわち、ある目的を達成するための最短距離、あるいは最適仕様がなされることが必要条件である。

目的自体がブラックボックスであり、抽象的なものであろうがあいまいにしてはならない。目的決定には次の諸項目の決定が必要である。

(1)　何処までをそのシステムの範囲とするか。

(2)　マイクロシステムによる効果は。

(3)　目的を達成するための性能、処理能力は。

(4)　目的を達成するための環境条件。

(5)　マイクロシステムの機能。

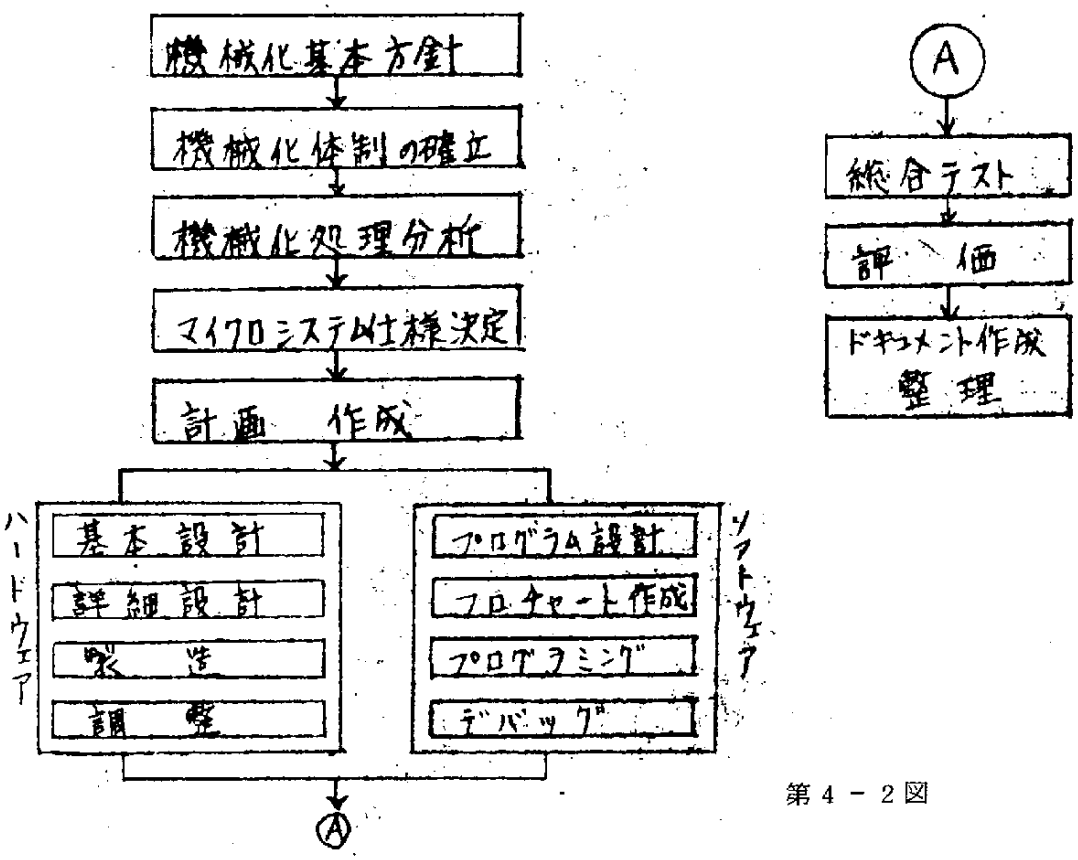
(6)　マイクロシステムの運用方法。

(7)　将来の構想。

これらを明確にすることにより、マイクロシステム自体の評価も上げうるし、開発、製作、検査などに伴う人的、時間的なものの介入物が少なくなり、生産管理にもよい結果を及ぼす。

4.2.2　システム設計開発の手順

システム設計開発の手順はこれでなければならないという決め手は



第4－2図

なく、システム完成期限や要員、システムの種類により、作業がオーバーラップして行なわれたりする。

しかしながら、以下では一般的なパターンを示すことにより、システム開発設計の概要を説明する。

(1) 機械化基本方針の決定

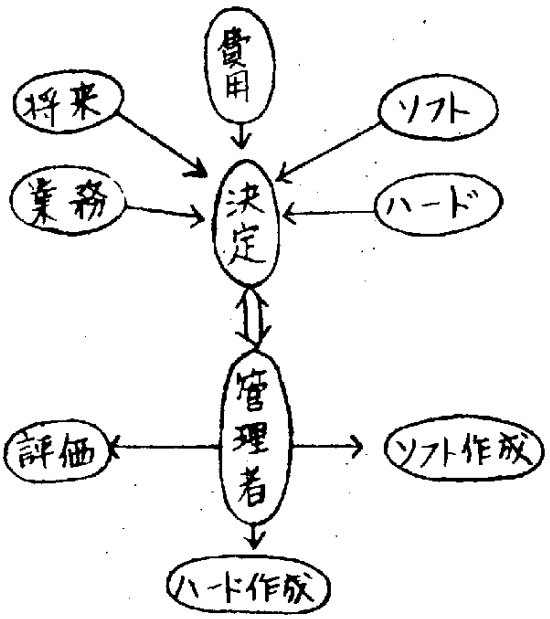
機械化基本方針は目的を決め、機器構成図、制約条件等を明確にする。（目的の決定は前項4.2.1を参照）

機器構成図では入力－出力の機器を決める。

制約条件は性能、運用方法の条件を明確にする。

(2) 機械化体制の確立

機械化を円滑に行なうためには、その体制の確立がポイントとなる。



第4－3図

(3) 機械化処理分析

機械化処理分析は現状分析に他ならない。よって現状の知識や経験が十分であることが要求される。現状を調査するには、面接、観

察、記録調査等があるが、具体的にはシステムが必要とする条件を明確にすることに於きる。このために仕事のパターン、仕事の量、必要度、データの流れと理由、データの内容、問題点および改良点などの各項目がある。

(4) マイクロシステム仕様決定

システム仕様は、プログラミング及びハードウェア設計に必要な内容を整理し、信頼性負荷の検討ののち、具体的な設計を行ない、更に関連するシステムと調和を保ち、業務全体がひとつの新しいシステムとして運用できるように用意するものである。一般にシステム仕様書として記述する。その実例として4.2.4項に記す。

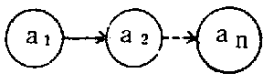
(イ) 信頼度の見積り

一般的に信頼度をあらわす尺度には次のようなものがある。

- MTBF(平均故障間隔：Mean Time Between Failure)
- MTTR(平均修復時間：Mean Time To Repair)
- $\lambda = \frac{1}{\mathrm{MTBF}}$ (故障率)
- $\mu = \frac{1}{\mathrm{MTTR}}$ (故障復旧率)
- 稼動率$= \frac{1}{\mathrm{MTBF}+\mathrm{MTTR}}$

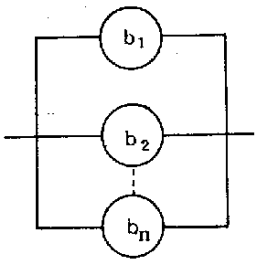
システムの稼動率は各機器の稼動率により求める。

- 直列系接続のとき



各装置の稼動率をa_1，a_2，…a_nと定めればシステムの稼動率は$a_1 \times a_2 \times \cdots\cdots a_n$となる。

- 並列系接続のとき



各装置の稼動率を b_1 , b_2 …… b_n と定めればシステムの稼動率は

$$1-(1-b_1)(1-b_2)\cdots\cdots(1-b_n)$$

となる。

システムの稼動率及び各機器の稼動率が不十分であるときは、稼動率の高い機器に変えるとか、直列系を並列系に変えるなどして信頼度の向上を行う。

(ロ) 負荷見積り

負荷見積りとは処理能力を正しく把握し、システム全体に渡って必要な機能を行うことが出来るかを調べることであり、ここでの負荷とはシステムの持つ能力に対して実際にデータが起こり、それを処理するために何%の能力を使用するかということである。

負荷見積りで100%以上になったならば制御装置入出力装置、入出力方法を変えなければならない。

基本的には負荷見積りを次の方針で行なう。

- 最も処理が集中したときでも確実に処理可能である。
- 必要な時間以内に所定の処理を行なえる。
- システムの拡張、データの増加を考慮し、余裕を持つ。

負荷を計算するために、1件のデータを処理するために必要なシステム稼動時間を、入力装置からの読取り時間、処理する制御時間、処理後のデータを出力装置に出力する時などを合わせて求める。このシステム稼動時間をもとにして、データが集中すると

き、あるいは1日単位の負荷を求める。

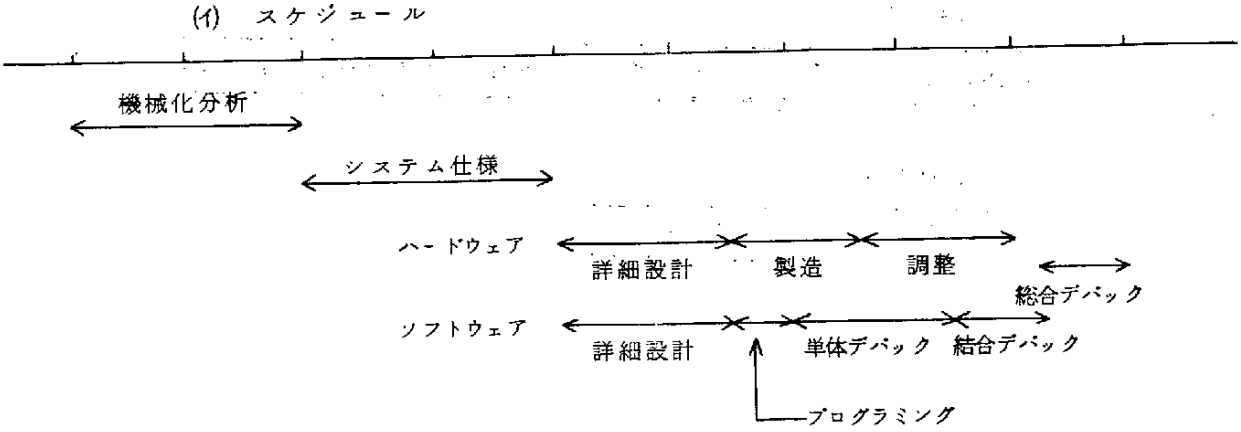
$$負荷率=\frac{データ発生件数\times 1件当りシステム稼動時間}{時\quad 間}\times 100$$

尚、負荷率と共に、1件当りのシステム稼動時間がデータの発生間隔より長いときには、改良点としてデータを待ち合わせる方法をとるか、負荷の大きな装置を変えるなどの工夫が必要である。

4.2.3　システム設計開発の留意点

システム開発における留意点としてスケジュール、要員、費用について記す。

(イ)　スケジュール



上記のような各作業項目に対しての必要作業量、消化できる作業量を十分に検討して日程を決める。

各作業項目の日程の中にスケジュールチェック日を設け、目標作業量と実績作業量の対比を行ない、これら2つに大きな差のでないよう留意し、差が大きくなったときは、別途対策を早めに講じること。又、各作業項目には成果物があり、この完全性を十分に検討して次の作業項目に進むこと。

(ロ)　要　員

各作業項目に基づき作業量、必要技術力、適性を把握して要員を決定する。

(ｲ) 費　用

システムの設計開発にはその製品の価値というよりも、設計開発にかかる人件費を目やすにして行かなければならず、製品の価値とのバランスに留意しなければならない。

4.2.4　システム仕様書　〔例〕

以下にレジスタの仕様例を記術する。

1.1　適用範囲

作成物件名、適用システム名とその機械化部分の概要を記す。

1.2　関連文書

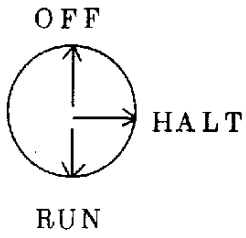
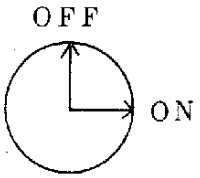
引用文書、参考文書等を記す。

2.1　構　成

本装置の構成は第２－１表のとおりとする。

第２－１表

番号	名　　称	数量	備　　考
1	制　御　装　置	1	次記により構成される。 o マイクロプロセッサ o ＲＡＭ　５１２×８bit o ＲＯＭ　３Ｋ×８bit o インターフェイス回路

番号	名　　称	数量	備　　考
2	鍵	2	○電　源　鍵  ○管 理 者 鍵 
3	ディスプレイ （Ｄ．Ｄ）	1	数字１０桁表示
4	キーボード （Ｋ．Ｂ）	1	３９キー
5	ラインプリンタ （Ｌ．Ｐ）	1	スタンプ動作／印字を行う
6	カセットテープ	1	ジャーナルを書く
7	ペンリーダ	1	バーコードを読み取る
8	ラ　ン　プ	3	○データエラーランプ ○マイナスランプ ○テープチェンジランプ

番号	名　　称	数量	備　　考
9	電　源	1	各部へ電源を供給する
10	ドロア	1	現金格納部
11	カセットテープ制御キー	3	○ＥＪＥＣＴ ○ＲＥＷＩＮＤ ○ＬＩＮＥ
12	バッテリ	1	電源断時使用

2.2　材料・部品及びその加工法

＜材料、部品及びその加工法に指定があれば記述する。＞

2.3　構造・形状・寸法及び重量

本装置の構造、形状、寸法及び重量は、付図に従うもののほか、次によるものとする。

(1)　構造堅ろう、動作円滑で連続使用に耐えること。

(2)　操作は１名で可能であること。

(3)　印字用紙等の入替装置が容易であること。

(4)　騒音及び電気的雑音の発生は極力抑制すること。

(5)　現金保管には安全性があること。

2.4　外　観

外観は付図を参考とし、各部の仕上りは良行であること。

その他塗装等についても記す。

2.5 機　能

2.5.1 機能概要

レジスタの主要な機能は次のとおりである。

(1) 情報入力はペンリーダからの読取りと、キーボードからの打けんによる方法がある。

(2) 入力情報は、オペレータコード、取扱い年月日、品番、金額、支払い区分、操作指定などである。

(3) 情報出力はレシートとしてラインプリンタに、ジャーナルとしてカセットテープに、置数はディスプレイに出力する。

(4) 取引情報を蓄積し、随時ラインプリンタに出力できる。

(5) 各ランプにより状況を表示する。

(6) 電源キー、管理者キーにより誤操作をロックアウトする。

2.5.2 各機器の機能

(1) 鍵

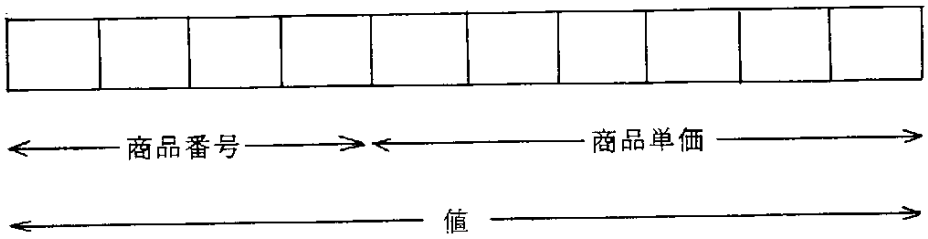
電源鍵、管理者鍵と処理理を第２－２表に示す。

第 2 － 2 表

電源鍵	管理者鍵	操作可能動作	備　考
OFF	X	電源断であり、無動作	
HLT	ON	日付、オペレータコード設定	
RUN	ON	入金 出金 精算 演算 締め 売上げにおける割引及びCR 売掛 返品	電源断のときはバッテリにて蓄積情報を保存するが動作はしない
	X	売上げ 演算	

(2) ディスプレイ

0～9の数字を10桁表示する。数字の指定しないときは無点灯である。



第 2 － 1 図

(3) キーボード

以下のキーボードを有する。

返品	割引1	CR1
入金	割引2	CR2
出金	割引3	CR3
精算		貸売

7	8	9
4	5	6
1	2	3
0	REPET	

商品番号キー

7	8	9
4	5	6
1	2	3
0	#	

金額キー

＋	－
、	Clear
金券	
現金	紙送

第 2 － 2 図

第２－３表

キー種別	キー名	説明
動作	返品	管理者鍵 ON 電源鍵 RUN で有効であり、以下の取引が返品であることを示す。 由に返品動作の最初の動作として必要である。
	入金	管理者鍵 ON 電源鍵 RUN で有効であり、ディスプレイに表示されている額を入金額として、入金処理を行う。
	出金	管理者鍵 ON 電源鍵 RUN で有効であり、ディスプレイに表示されている額を出金額として入金処理を行う。
	精算	管理者鍵 ON 電源鍵 RUN で有効であり、精算処理を行う。精算処理直後に押下すると、カセットテープをクローズする。
	割引１ ～ 割引４	管理者鍵 ON 電源鍵 RUN で有効であり、直前の商品単価に各々の率を掛けた額を割引く。 割引１……１０％ 割引２……２０％ 割引３……３０％ 割引４……　５％
	CR1 ～ CR3 貸売	管理者鍵 ON 電源鍵 RUN で有効であり、支払い区分を示す。金額キーの直後に押される。
商品番号	Ø ～ 9	商品番号を示す。商品番号は４桁以下の数字で示す。 商品番号は、商品単価の前に押されなければならない。

キー種別	キー名	説明
	RePeat	商品番号、商品単価割引区分の3つを一括して繰り返すときに、このキーを押下する。
金額	0〜9	商品単価、日付オペレーターコード、入金／出金額、演算値を入力するとき使用する。
動作		商品番号、単価のタイプインが終了したことを示す。 売上げ及び返品の合計を取る。 売上げ及び返品処理を終了させる。 演算の合計値を印字させ、演算処理を終了させる。 日付オペレータコード入力の最後に押下し、記憶させる。
	＋	ディスプレイ上の数値を演算値に加え、答をディスプレイ、マイナスランプに表示する。
	－	ディスプレイ上の数値を演算値より引き、答をディスプレイ、マイナスランプに表示する。
	取　消	売上げ及び返品において、直前の商品を取消す。
	金　券	売上げ及び返品処理における支払い区分の1つであり、金券による売上げ及び返品のとき使用する。
	現　金	売上げ及び返品処理における支払い区分の1つであり、現金による売上げ及び返品のとき使用する。
	Clear	金額キーの直後、商品番号の直後に押下すると、それぞれが取消される。

キー種別	キー名	説明
動作	紙送	レシートを繰り上げて、紙送りが終了したとき、スタンプを印字する。

(4) ラインプリンタ

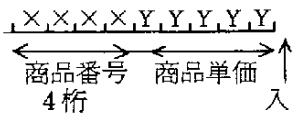
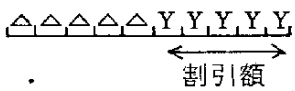
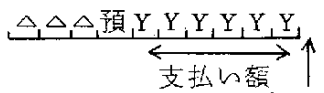
レシートを作成する。

レシートにはスタンプと取引の内容を印字する。

0	0	0	0	0	0	0	0	0	0	+
1	1	1	1	1	1	1	1	1	1	−
2	2	2	2	2	2	2	2	2	2	
3	3	3	3	3	3	3	3	3	3	割1
4	4	4	4	4	4	4	4	4	4	割2
5	5	5	5	5	5	5	5	5	5	割3
6	6	6	6	6	6	6	6	6	6	割4
7	7	7	7	7	7	7	7	7	7	CR1
8	8	8	8	8	8	8	8	8	8	CR2
9	9	9	9	9	9	9	9	9	9	CR3
返	扱	入	預	—				—		現
★	残	出	年	釣		月			日	✡
消	↑	•	•	•	•	•	•	•	•	貸

第２－３図　ラインプリンタードラムレイアウト

第２－４表

番号	印字形式	備考
1	 • 商品単価のリーデングØはスペース印字 • 入力区分 { タイプイン……スペース印字 / ペンリーダ……✡印字 }	商品番号、単価を印字する。
2	 割引額：リーデングØはスペース印字	割引キーにより割引額を印字する。
3	 リーデングØはスペース印字	売上げ返品の合計を　キーにより印字する。
4	 リーデングØはスペース印字	売上げ処理で支払い区分により金額を預かったとき印字する。
5	 リーデングØはスペース印字	売上げ処理の認証　キーにより印字する。釣がないときは印字しない。

番号	印　字　　形　式	備　　考
6	Y Y Y − X X X − Z レジスタ番号　取扱番号　オペレータコード Y Y 年 M M 月 D D 日	売上げ、返品処理の最後に印字する。
7	返扱 • • • • • • • •	返品キーで印字する。返品処理開始を示す。
8	消↑ • • • • • • • •	直前の商品を取消すとき、取消キーにより印字する。
9	入 Y Y Y Y Y Y Y 出 Y Y Y Y Y Y Y リーデングØはスペース印字	入金、または出金キーでその額を印字する。
10	Y Y Y Y Y Y Y Y Y Y + Y Y Y Y Y Y Y Y Y Y − リーデングØはスペース印字	演算の＋値または－値を印字する。
11	Y Y Y Y Y Y Y Y Y Y リーデングØはスペース印字	演算の合計値を印字する。
12	2．5．3(7)項参照	精算処理

スタンプ動作は次の処理のあと行う。

① 　キーにより売上げ及び返品処理を終了させ年月日を印字したあとで。

② 入金、出金、演算処理が終了したあとで。

③ 紙送りキーによる紙送りのあとで。

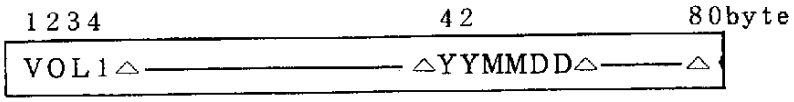
(5) カセットテープ

ジャーナル情報を記録する。

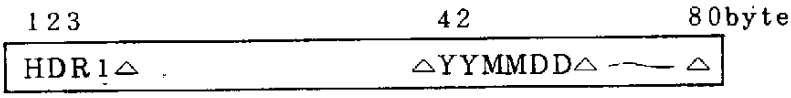
(イ) 記録情報形式

① ヘッダ形式

○ A面の最初のブロック

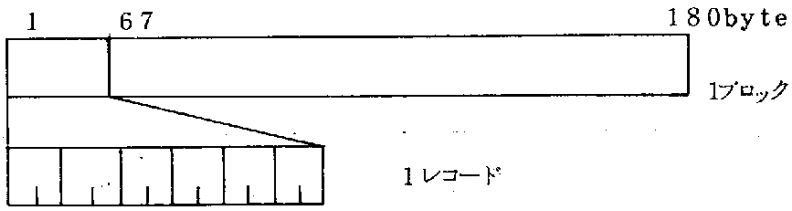


○ B面の最初のブロック



第2－4(a)図

② データ形式



第2－4(b)図

40レコード／1ブロック

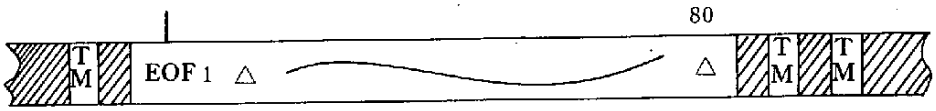
第２－５表

レコード	レコードID	レコード内容
売上げ	8	R－ID Ø ――商品番号――商品単価――
返品	C	
売上げ割引	9	R－ID 割引額 割引種別
返品割引	Ø	
売上げ額	A	R－ID 支払い／売上げ額 支払い／売上げ区分種別
返品額	E	
売上げ情報	B	R－ID M M Ø Ø レジスタ番号 取引番号 オペレータコード MMDDは月日
返品情報	F	
取消	1	1 Ø Ø Ø
未使用	Ø	Ø

レコードＩＤは、１６進数で記す。

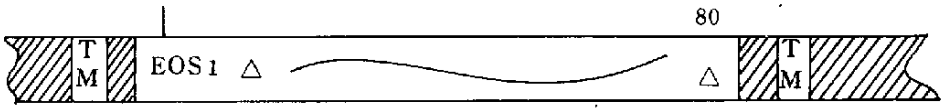
③ トレーラ

∘ ファイルの最後を示すトレーラ

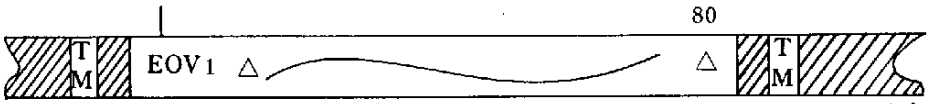


∘ テープリールの終了を示すトレーラ

A面の最後のブロック



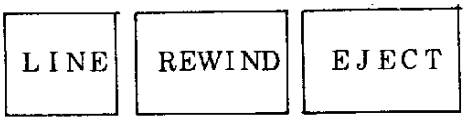
B面の最後のブロック



TMはテープマークを示す

第２－５図

(ロ) カセットテープ操作ボタン及びランプ



第２－６表

ボタン名	ボタン型式	備　　考
LINE	ロック	押下することによりオンライトとなりロックされる。再度押下すると、ボタンは上がりオフラインとなる。
REWIND	ロック	押下するとリワインドを行い、リワインド中はロックされる。
EJECT	アンロック	カセットテープをオープンする。

(6) ランプ

第２－７表の通り３種類のランプを設ける。

第２－７表

ランプ名	点灯及び消灯
データエラー	点灯で置数のオーバーフロー及びベンリーダからの読取りエラーを示す。 Clearボタンによりオーバーフローした数値を取消し、ランプを消灯する。
マイナス	演算結果がマイナス値のとき、点灯する。 売上げ金額＞預額のとき点灯する。 キー押下により消灯する。
テープチェンジ	トレーラを書いたのち点灯し、ヘッダーを書いたのち消灯する。 書込みテープがセットされていないとき点灯し、ヘッダを書いたのち消灯する。

2.5.3　処理機能

(1)　オペレーターコード、日付の設定処理

(イ)　動作前提条件

- 電源鍵がHALTである。
- 管理者鍵がONである。
- その他の処理中ではない。

(ロ)　動作手順

①　商品番号キーにより1桁オペレータコード、金額キーにより6桁の年月日をタイプインする。

②　＃キーをタイプインする。

(ハ)　処理

①　タイプインされた数字を1桁毎右へずらして、ディスプレイに表示する。

②　＃キーがタイプインされたときに表示していた7桁の数字のうち、下位6桁を年月日、上位1桁をオペレータコードとする。表示は消却する。7桁に満たない値は不足桁を0とみなす。

(2)　売上げ処理

(イ)　動作前提条件

- 日付オペレータコードが設定済である。
- 演算の途中ではない。
- 電源鍵がRUNである。

(ロ)　動作手順

①　ペンリーダにより読取るか、キーボードにより商品番号、商品単価をタイプインしたのち、＃キーをタイプインすることにより、商品情報を与える。

②　割引きがあるときは管理者鍵をONとして、各々該当の割引キーを2回に限りタイプインする。

③ ①、②における商品情報を取消すときは最後の商品に限り、取消キーをタイプインすることにより取消す。

④ ①、②における商品を複数個売り上げるとき、①、②の動作の後にRepeatキーにより押下回数だけ①②の動作の作用を指定できる。

⑤ 次の商品を売り上げるとき再度①より行う。売上げ商品がなくなったとき、＃キー押下により合計をとる。

⑥ 金額キータイプインで額を、支払区分を現金、金券、CR1～CR3、貸売キーで指定する。

マイナスランプが消灯するまで⑥を繰り返す。

本動作の途中でも①より動作すれば売上げ商品を追加できる。

⑦ ＃キーを押下し、釣計算させたのち、レシートを得て1件の売り上げ処理を終了する。

(3) 処理

① タグの読取り

タグよりペンリーダで商品番号、商品単価を読み取り、ディスプレイ、ラインプリンタに表示する。読取りエラーのときはデータエラーランプのみを点灯させる。

② 商品番号、商品単価のタイプイン

商品番号4桁以内、商品単価6桁以内がタイプインされたのち、＃キーによりラインプリンタに印字する。

各置数はタイプインされたつど、ディスプレイに表示する。

各桁の不足分は上位を0とみなす。桁オーバーはデータエラーランプを点灯させる。

③ 管理者鍵がONのときは割引キーを2回に限り受け付け、商品単価より割引率をかけそれを引く。この値をディスプレイ、ラインプリンタに表示する。

④ リピートキーが押下されたならば、直前の商品に対しての番号、単価、割引額をラインプリンタに印字する。

⑤ 取消キーが押下されたならばディスプレイを消却し、ラインプリンタに取消マークを印字する。

⑥ 合計＃キーのタイプイン

（商品単価の合計額－割引合計額）をディスプレイ、ラインプリンタに出力する。マイナスランプを点灯する。

既に預金があるとき差額を表示する。

⑦ 預金額のタイプイン

金額キーにより額を入力し、ディスプレイに表示する。支払い区分を現金、金券、貸売、CR１～３で指定されたとき、支払い区分とともに額をラインプリンタに印字して、（売上げ額－預金額）の絶対値をディスプレイに表示する。売上げ額が預金額より大きいときマイナスランプを点灯し、そうでないとき消灯する。

⑧ 認証＃キーのタイプイン

マイナスランプが点灯しているときは無効である。釣額をラインプリンタに印字し、扱いレジスタ番号、オペレータコード、月日を印字し、ディスプレイを消却する。

(4) 返品処理

(イ) 動作前提条件

- 日付オペレータコードが設定済である。
- 演算の途中でない。
- 管理者鍵ON、電源鍵RUNである。

(ロ) 動作手順

① 返品キーを押下し、返扱を印字させる。

② 合計＃キー押下までの動作は売上げ処理と同一である。

合計＃キー押下での合計額を支払って認証＃キーを押下し、処

理を終了する。

(5) 入金処理

(イ) 動作前提条件

管理者鍵ON、電源鍵RUNである。取引、演算の途中ではない。

(ロ) 動作手順

① 金額キーにより入金金額を指定する。

② 入金キーをタイプインする。

(ハ) 処理

① 金額をディスプレイに表示し、入金キーで消える。

(6) 出金処理

(イ) 動作前提条件

- 管理者鍵ON、電源鍵RUNである。
- 取引、演算の途中ではない。

(ロ) 動作手順

① 金額キーにより出金額をタイプインする。

② 出金キーをタイプインする。

(ハ) 処理

① 金額をディスプレイに表示し、出金キーにより消却する。但し、出金額が残っていないときはディスプレイに差額を表示し、マイナスランプを点灯する。

取消キーがタイプインされたとき、ディスプレイ、マイナスランプを消灯する。

(7) 精算処理

(イ) 動作前提条件

- 管理者鍵ON、電源鍵RUNである。
- 取引、演算の途中ではない。

(ロ) 動作手順

① 精算キーを押下する。

精算キーに続いて精算キーを押すと、クローズ処理を行う。

(ハ) 処理

以下の情報をラインプリンタに印字する。

★		売上げ件数	△
★		売上げ物件数	△
★		総売上げ額	△
割1	件数	総金額	割1
割2	件数		割2
割3	件数		割3
割4	件数		割4
CR1	件数		CR1
CR2	件数		CR2
CR3	件数		CR3
貸売	件数		貸
返		返品件数	△
返		返品物件数	△
返		返品総金額	△
入金件数	入	入金総額	△
出金件数	出	出金総額	△
精算回数	残	残高	△
★	年	月 日	△

第2－6図

(8) 演算処理

(イ) 動作前提条件

- 電源鍵RUNである。
- 取引の途中ではない。

(ロ) 動作手順

① 金額キーにより置数する。

② ＋／－を押下する。

③ ①、②を繰り返す。

④ ＃キーを押下し合計を印字する。

(ハ) 処理

① 置数した値はディスプレイに表示する。置数後Clearを入力すると置数を消却し、前の答があればディスプレイに表示する。

置数が10桁を越えたときはデータエラーランプを点灯させる。

② 演算子＋／－の入力により置数と演算子をラインプリンタに印字し、答をディスプレイとマイナスランプに表示する。

③ ＃キーにより答をラインプリンタに印字し、ディスプレイ、マイナスランプを消す。

(9) クローズ処理

(イ) 動作前提条件

- 管理者鍵ON、電源鍵RUNである。
- 直前に精算処理がされている。

(ロ) 動作手順

① 精算キーを押下する。

(ハ) 処理

① カセットテープをクローズする。

2.6 性能

各部における処理速度、外囲条件（温湿度、耐振性）、耐久力等を記述する。

3. 品質保障

3.1 監督検査

監督検査方法を記述する。

4. 付属品、予備品

付属品、予備品について記述する。

4.3 ハードウェアの開発

4.3.1 ハードウェア構成の概要設計

各社から多くの種類のプロセッサチップが発表、発売されており各々特徴があるが、本例では最も一般的と思われる8080相当のプロセッサを使用する。

マイクロコマピュータの構成は大別して、以下の4つに分けられる。

(1) 中央処理装置（CPU）

(2) メモリ

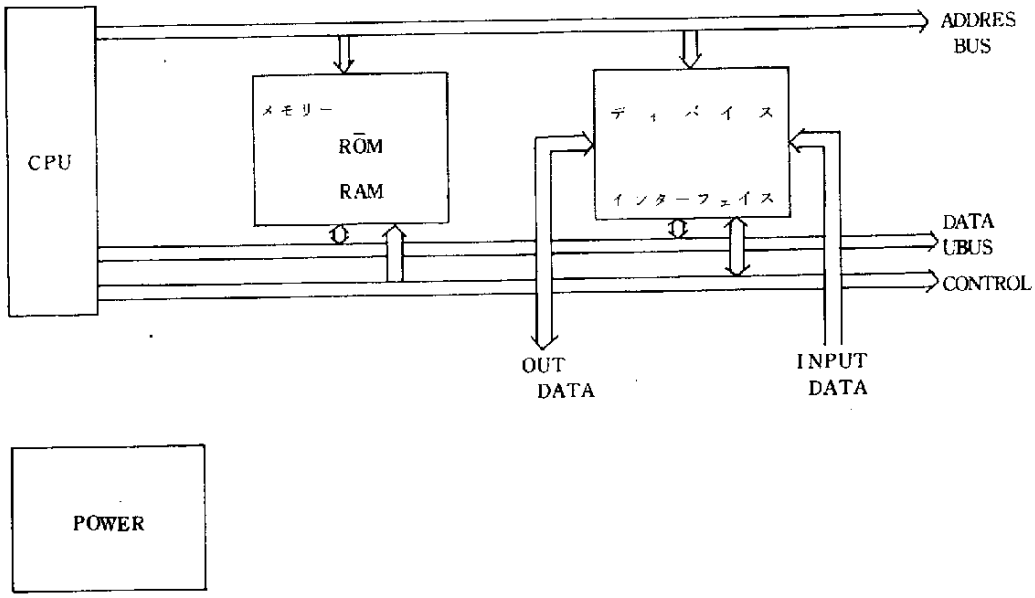
(3) 周辺ディバイスインターフェース（システム開発用として、操作パネルを付けることがある。）

(4) 電源

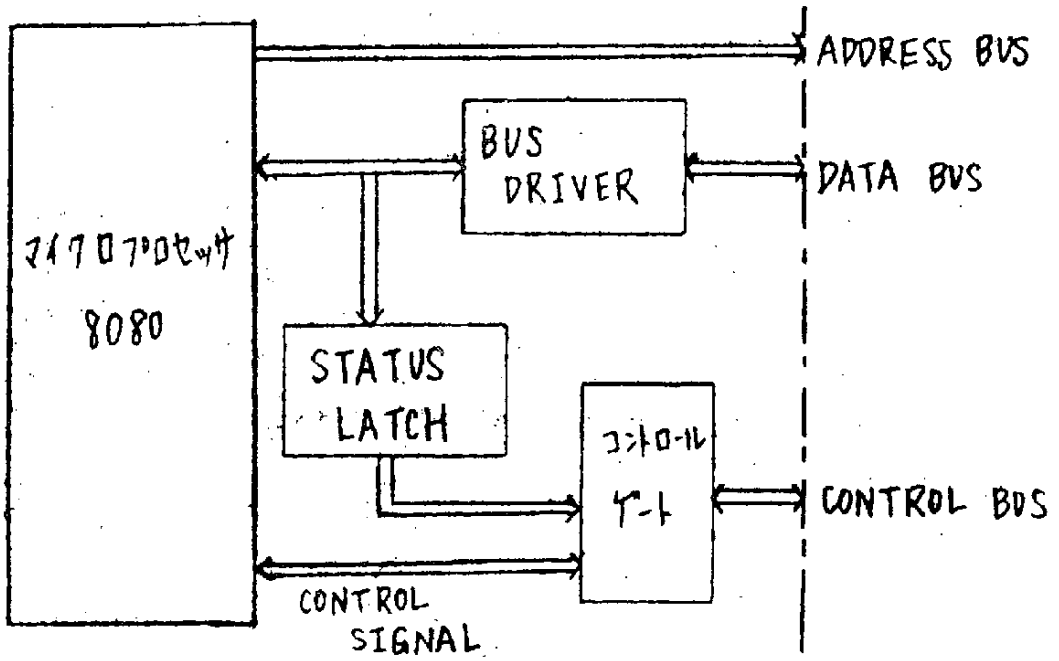
これらは第4－4図の如くバスラインを通して結合され、メモリ、周辺ディバイスを2つ以上接続するときも、全く同様にバスラインに接続される。

(1) CPUの概要

CPU部分はマイクロプロセッサLSI（8080）を中心として、それを正常に作動させるためのクロックジェネレータ・コントロール信号及び外部とのデータのやりとりをするBUS DRIVERからなる。


CPU
メモリー
ROM
RAM
ディバイス
インターフェイス
ADDRES BUS
DATA UBUS
CONTROL
OUT DATA
INPUT DATA
POWER

第４－４図


マイクロプロセッサ
8080
BUS DRIVER
STATUS LATCH
コントロール ゲート
ADDRESS BUS
DATA BUS
CONTROL BUS
CONTROL SIGNAL

第４－５図

（前頁、第４－５図参照）

(イ) マイクロプロセッサ

８０８０型又は相当品

(ロ) ADDRESS BUS

マイクロプロセッサから出ているアドレス（A_0～A_{15}）でこの信号により、メモリの番地・入出力のディバイスセレクト信号を出力する。

(ハ) クロックジェネレータ

マイクロプロセッサ８０８０はφ１、φ２の２相タイミングクロックを必要とする。

(ニ) BUS・DRIVER

入力・出力の両方向性のアンプで、CPVのコントロール信号（DBIN）により入力、出力を切り換える。（このアンプは接続されるメモリ、入出力ディバイスが少ない時は必要ない。）

(ホ) STATVS LUCH

８０８０から、その内部の状態を示す信号８種類がデータバスにより出力される。これを受け取り、メモリ・入出力のコントロールに使用する。

（INTA、HLTA、MEMR、INP、OUT、WO、MI、STACK）

(ヘ) コントロールゲート

外から８０８０をコントロールする信号（READY、HOLD、RESET、INT）及び８０８０から出力される信号（SYNC、DBIN、WALT、WR、HLDA、INTA）を増幅し、タイミングの同期をとる。

(2) メモリ部の概要

マイクロコンピュータのメモリとして半導体メモリが使われる半導体メモリには、ROM（READ ONLY MEMORY）とRAM

（RANDAM ACCESS MEMORY）があり、ROMはプログラムを入れておくため使用され、RAMはデータ処理のバッファとして使われる。

本例では、ユーザがプログラムを書込むことができるPROM(1702)を使用し、データ用としてRAM(1202)を使用する。1702は1チップが256WARD×8BITに作られており、本例の2Kには8個を使用する。RAMの2102は1K×1BITが入っているから、1KWARDを作るには8個を使う。

この他メモリー部には、アドレスからメモリーチップを選ぶ信号のチップセレクト信号、データバスに接続するためバスゲート及びリード・ライトの切換信号を作る。

（第4－6図参照）

(3) ディバイスインターフェース

CPUと外部接続ディバイスとの接続及びコントロールする部分であり、

(イ) スイッチ入力

(ロ) リレー・ランプ出力

(ハ) キーボード、数字ディスプレイ

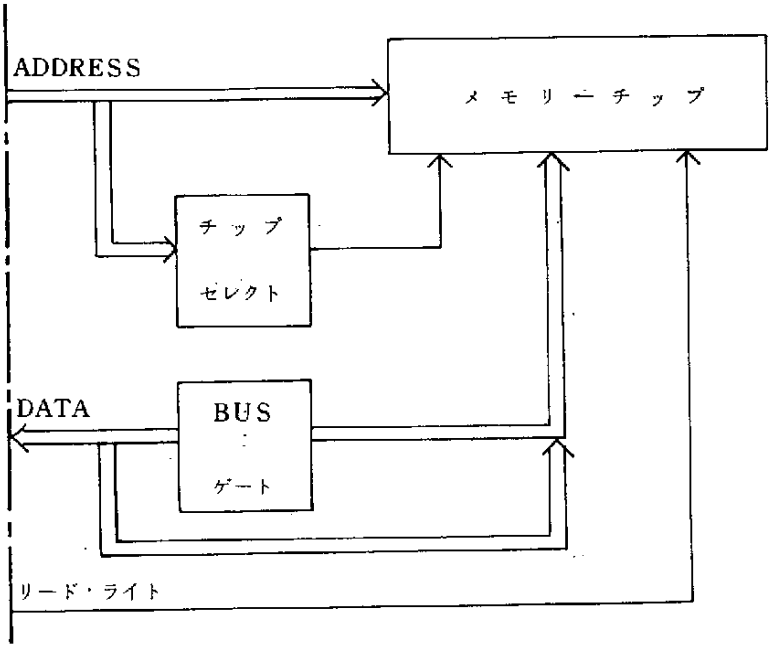
を接続する。

(4) POWER

商用電源AC100Vから、マイクロコンピュータシステムに必要なDC電源＋12V、＋5V、－9Vを作るもので、本例では市販の定電圧電源を使用する。

4.3.2 回路設計

回路の詳細設計にあたっては、使用するCPU(8080)、メモリその他のスペックを理解するため、メーカーの資料を充分調べる必要がある。特に、信号の性質、タイムチャートを理解する。



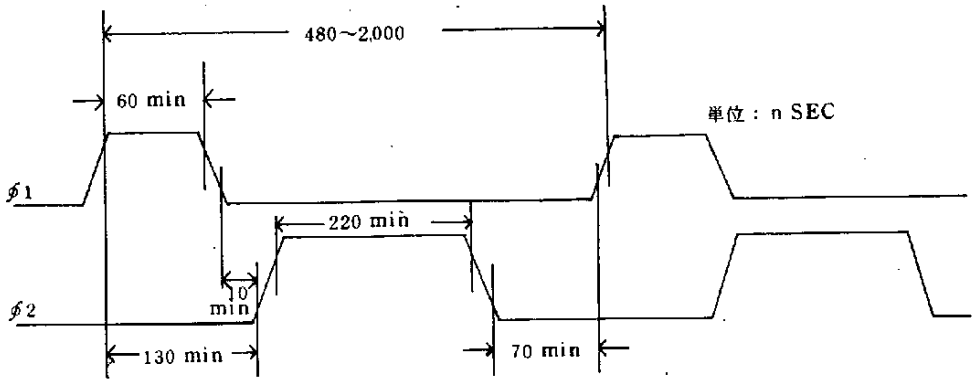
第４－６図

(1)　ＣＰＵ部の設計

(イ)　クロックジェネレータ

８０８０に要求されるクロックのタイミングは下記の通りである。

（第４－７(a)図）



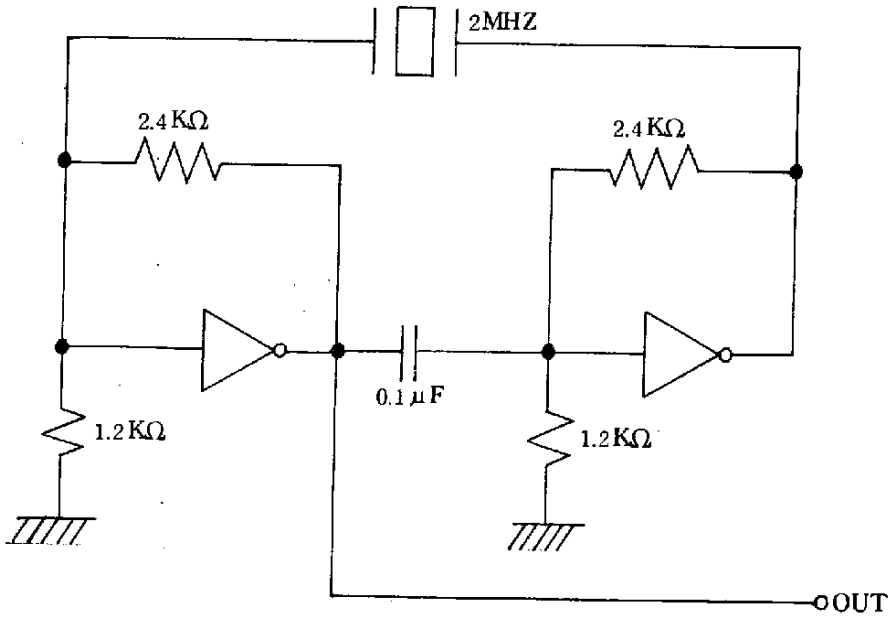
第４－７(a)図

これを実現するために、２～３個のワンショットマルチバイブレータを組合わせて作ることができるが、ここでは、調整が簡単な方法として水晶発振器とフリップフロップによるカウンタによる。

① 発振回路

比較的簡単な回路で安定に発振する回路として、水晶振動子を用いる。

発振回路は第４－７(b)図の如く、ＴＴＬインバータを用いて作ることができる。

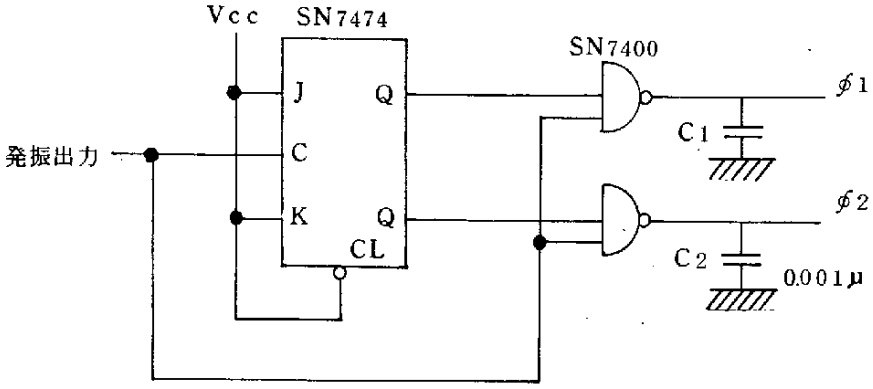


第４－７(b)図

② 信号発生

発振回路の出力をＪ・Ｋ形フリップフロックで分周し、その入出力のＡＮＤ回路により、所要の波形の信号を得る。

（第４－７(c)図参照）



第４－７(c)図

ＳＮ７４７４はＪとＫを（＋）にしておき、クロックを入れると１／２に分周される。その出力とクロックのアンドをＳＮ７４００で行う。

このタイミングを第４－７(d)図に示す。ここで注意を要するのは、Ｆ・Ｆのカウンタ出力信号は入力より遅れるため、ＡＮＤゲートにスパイクノイズが出る。これを除くため、コンデンサC_1、C_2をつける。

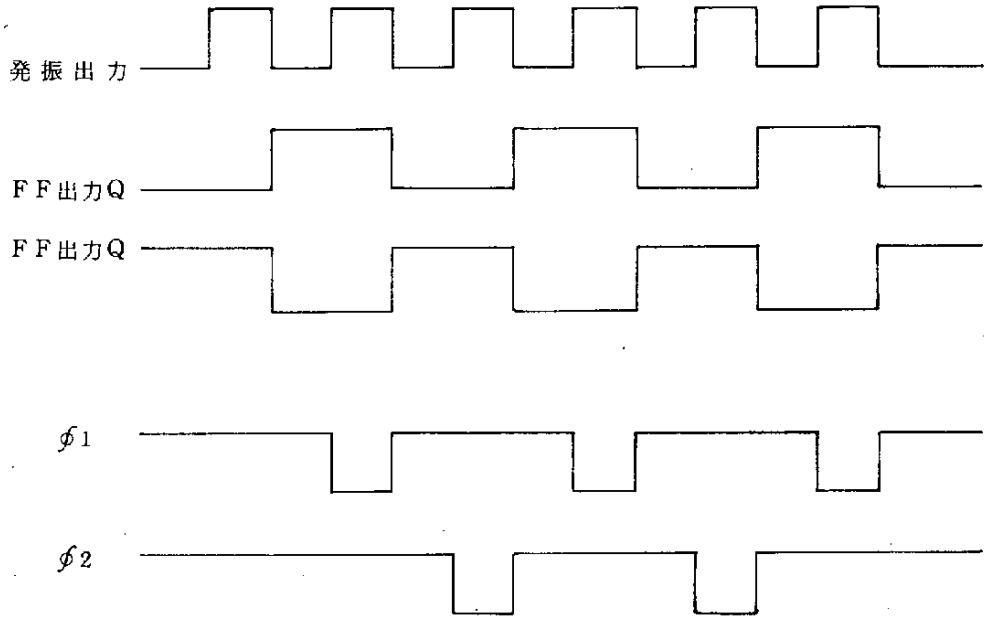
③　レベルシフト

８０８０のクロックϕ_1、ϕ_2は、１２Ｖの信号レベルを必要とするため、ＴＴＬの５Ｖから１２Ｖにレベルシフトしなければならない。このためオープンコレクタのＳＮ７４０６を使用する。

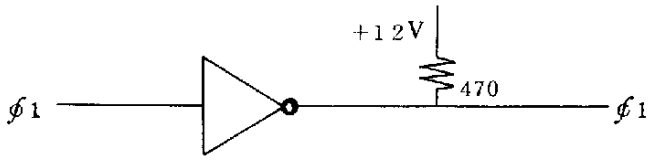
（４－７(e)図）

④　出力波形

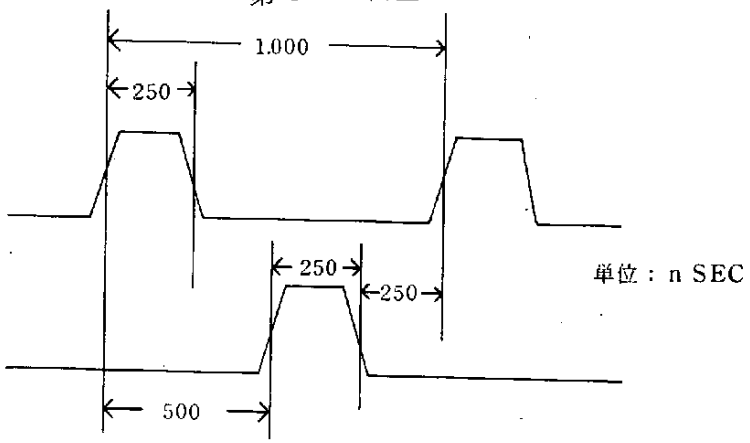
以上の回路で出来たクロックは第４－７(f)図の様になり、上記の要求を満足する。



第 4 － 7 (d)図



第 4 － 7 (e)図

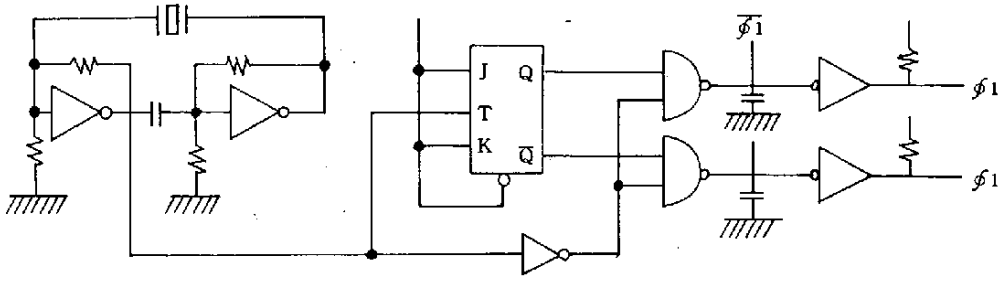


第 4 － 7 (f)図

本例ではクロックを1MHZとしたが、8080の性能いっぱいの2MHZで使用したいときは、発振の小晶振動子を4MHZ用と交換する。

なお、インテル社等ではクロックジェネレータとして(8224)が発売されており、これを使用すると、水晶を接続するだけでよい。

⑤ 回路図



第4－7(g)図

(ロ) コントロール信号

CPU(8080)のコントロール信号として

① READY

② HOLD

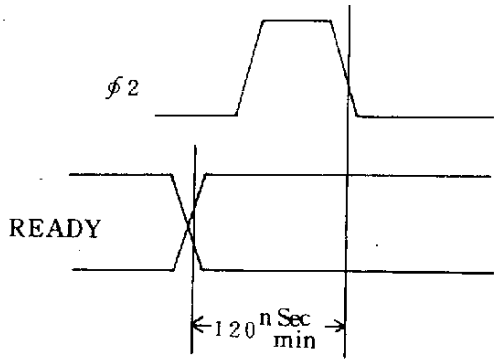
③ RESET

④ INT

がある。

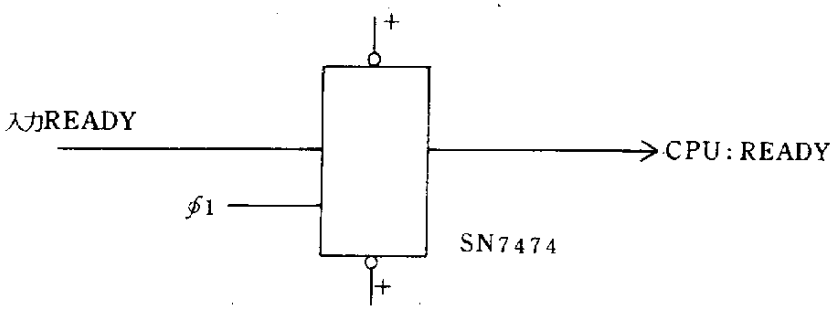
① READY

8080はメモリ、入出力装置などの動作の遅いデバイスからREADYが出ないと、次のステップに進まずWAIT状態になる。この信号は次の(第4－8(a)図)タイミングが要求されている。



第４－８(a)図

即ち、T2からTWまでのφ２の間安定させる必要がある。従ってREADY入力はφ１で同期させるために、D形フリップフロップ（SN7474）で第４－８(b)図の如く行う。

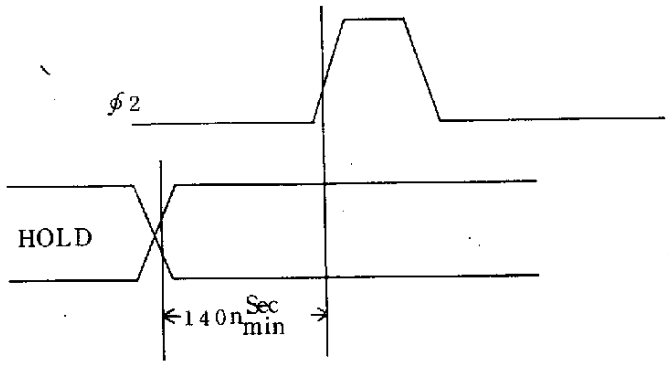


第４－８(b)図

② HOLD

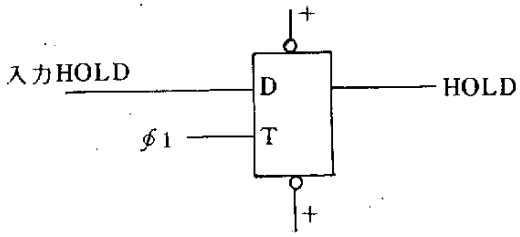
CPU（８０８０）の動作を一時中断させ、アドレスバス・データバスをハイインピーダンスにして、DMAやパネル操作を可能にする。

８０８０は第４－８(c)図の如く、HOLD信号はφ２で安定に保つ必要がある。



第４－８(c)図

従って、READYと同様にφ１で同期させるため、D形Ｆ・Ｆを使用する。



第４－８(d)図

③　RESET

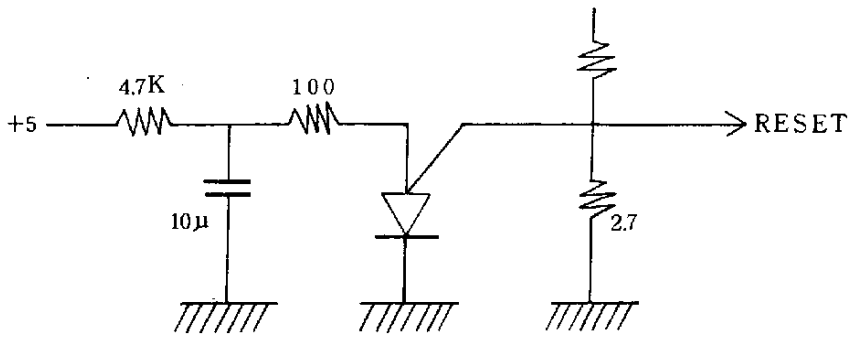
CPUをRESETする信号で、CPUはアドレスカウンタを０に全てのステータスをリセットする。

本例では、電源の投入時、RESET信号を発生し、０番地からスタートさせる。

④　INT

インタラプトの要求信号で、READYおよびHOLD信号と同様なタイミングを要する。

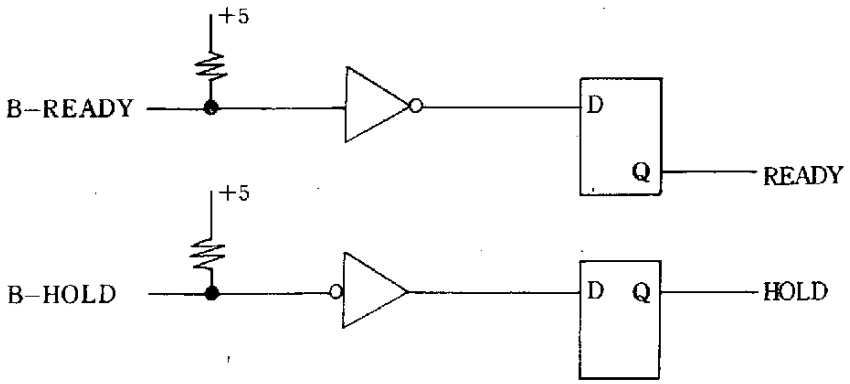
回路は後述する。



第４－８(e)図

⑤　BUSとの接続

コントロール信号は、CPU部の外部から要求され信号で、複数のメモリ・入出力装置などが接続できるように負の信号とする。従って前述の回路に入る前にインバータにより極性を反転させる。

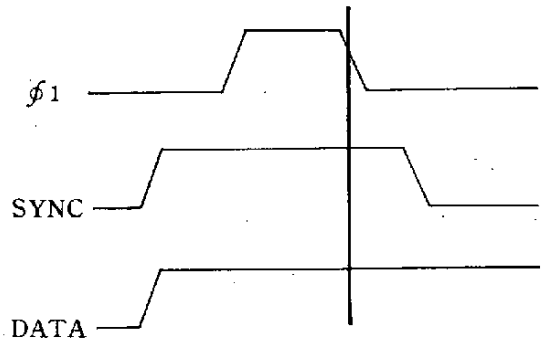


なお、READY は BUS 側から WAIT の必要がなければ負信号を出さないから、F・Fの出力 READY は $\bar{Q}$ からとる。

(ハ) STATUS

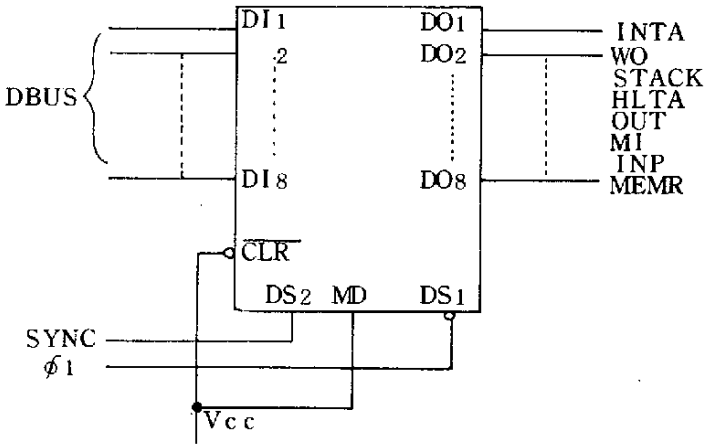
タイミングダイヤグラムに示されるように、命令によりマシンサイクルは1～5からなっており、各々のサイクルのデータバス使用の状態を示す信号が各々シンサイクルの頭で、CPUから出される。

各々の信号は使用する所で説明するが、マニアルを参照のこと。



第4－9(a)図

第4－9(a)図のタイミングϕ1とSYNCのAND出力で同期をとり、F・Fに記録する。この目的のために、8BITのF・Fとドライバーが入ってLSIが各プロセッサ・ICメーカで出されている。

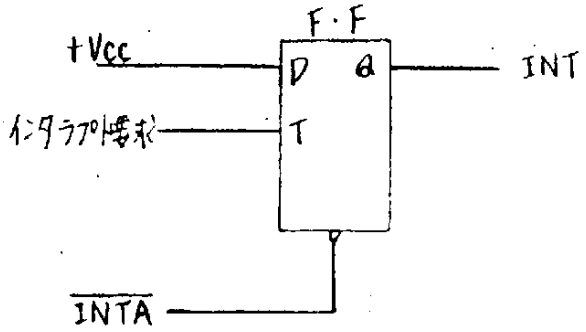


第４－９(b)図

(ニ)　インタラプト

プログラムインタラプトを使用するとき、外部から要求する。本例では、停電時の処理用として、停電を検出してインタラプトをかける。

①　インタラプト要求



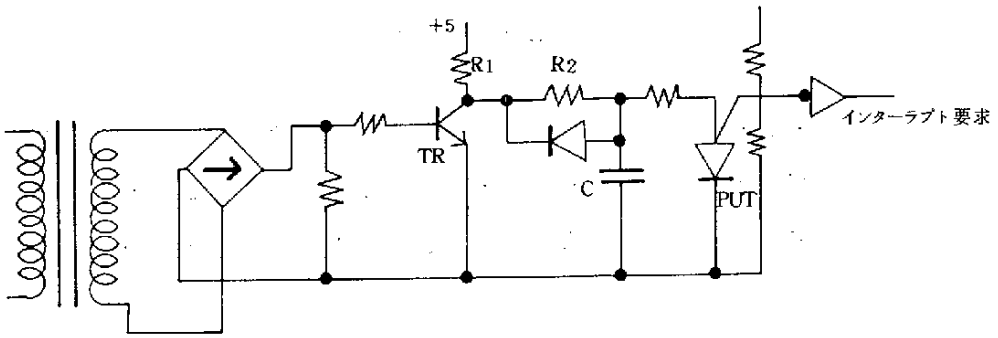
第４－10(a)図

停電により、インタラプト要求があると、このF・Fはセット

され、インタラプトINTが出る。又、このF・Fは、プロセッサでインタラプトが受け付けられ、（INTA）信号でリセットする。

② 停電インタラプト要求

システムに使用するDC電源は、大容量のコンデンサが接続されているため、AC入力が切れても100ms程度（コンデンサの大きさと負荷により変る）電圧が残る。そこでAC入力が切れたときを検出して要求を出す。



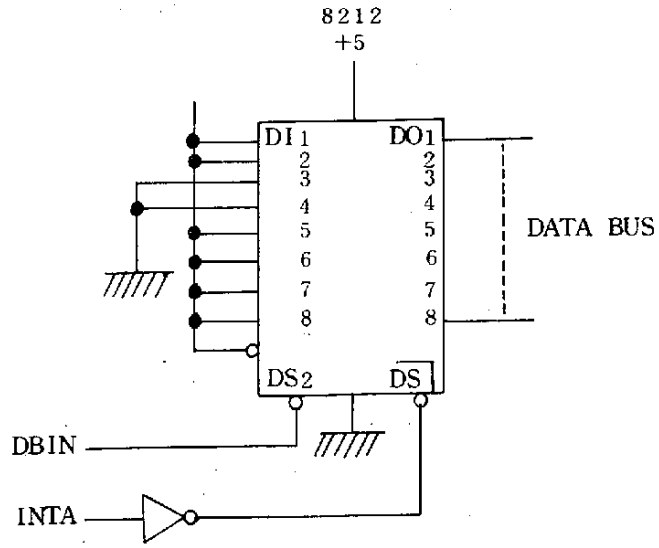
第4－10(b)図

この回路はAC入力があるときは、トランジスタ（TR）は50HZで10msec毎にONされる。その時Cに充電されている電圧は放電される。しかしAC入力が切れると、（R_1+R_2）Cの時定数で充電され、PVTがONされる。そこで、（R_1+R_2）Cの時定数を20msに選んでおくとよい。

③ インタラプト インストラクション

インターラプトが受け付けられると、プロセッサからINTAが出て、インタラプトインストラクションが受け付けられる。インタラプトインストラクションとしては、1バイト命令のPSTが

使用される。



第４－９(c)図

この回路（第４－９(c)図）では、ＬＳＩの８２１２で出力はＴＲＩ ＳＴＡＴＥになっており、データバスに直接接続される。

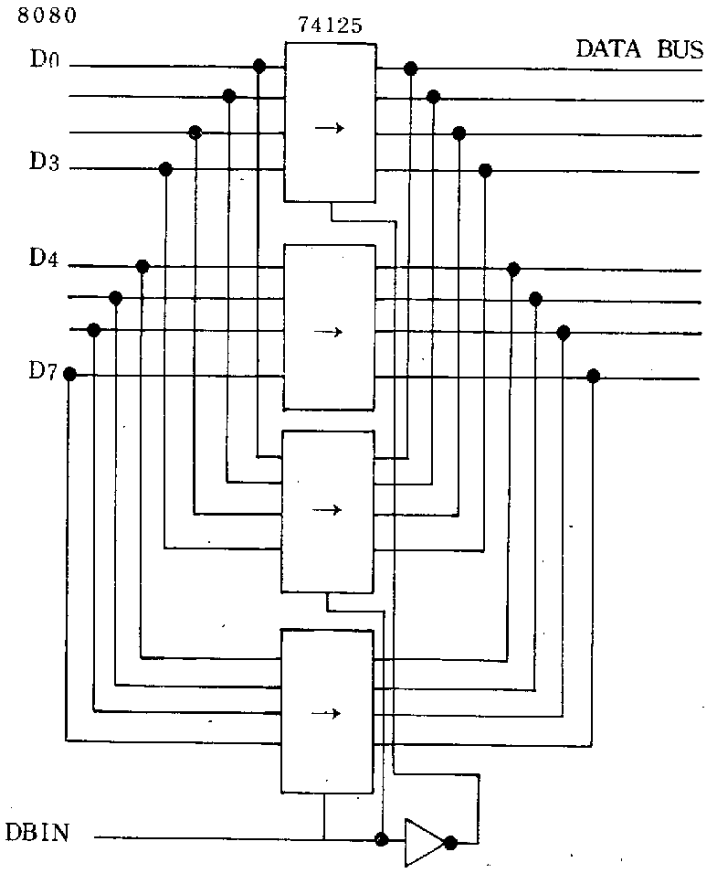
即ち、ゲート信号ＤＢＩＮ・$\overline{\text{INTA}}$が入ったとき、入力のＰＳＴ信号（１１００１１１１）がＤＡＴＡＢＵＳに接続され、その他の時はＨＩＧＨ　ＩＮＰＥＤＡＮＣＥになり、ＢＵＳに影響を与えない。

(ホ)　バス　ドライバ

①　データ　バス

データバスは、複数のＴＴＬ、ＭＯＳを接続できる様にドライバを付ける。データバスは、入出力の双方向に使用するため、２組のドライバを逆に接続し、使用しない方向はｈｉｇｈにする必要があり、ＴＲＩ　ＳＴＡＴＥのＩＣを使用する。

入出力の切換は、プロセッサのデータバスコントロール信号のＤＢＩＮを使用する。即ちＤＢＩＮのとき、外部からプロセッサに、



第４－１０(a)図

その他はプロセッサから外部バスの方向にゲートがあけられる。

② アドレス　バス

アドレスバスは、負荷が少ない時は必要ないが、メモリ、ディバイスが多く接続されたり、アドレスデコードにＴＴＬを使用するためにはドライバをはける必要がある。

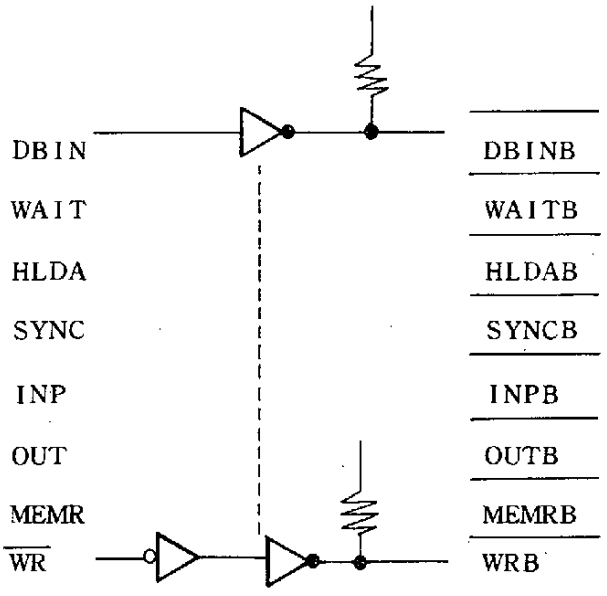
小規模のシステムで、アドレスバスに接続されるものがＭＯＳ形なら入力電流が少なく、特にドライバを、必要としない。

③ ステータス・コントロールバス

ステータス信号は、そのまま外部に出してもよいが、本例では

入出力装置側でも（パネルコントロール等）、同一目的の信号を作り、共通に使用可能にするため、オープンコレクタ形インバータを使用してワイヤーORを可能にする。

（第４－１０(b)図参照）



第４－１０(b)図

(ハ) CPU部の電源

８０８０に必要な電源は、

VDD	＋１２ V
VCC	＋ ５ V
VBB	－ ５ V

の３種類がある。

その他の素子は全て、＋５Vのみを供給するが、クロックには＋１２Vを使用する。

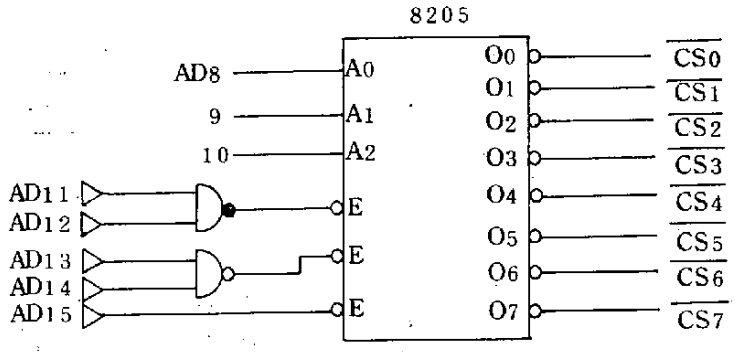
(2) メモリ部の設計

(イ) PROM

1702PROMは、256WARD×8BITであり、本例において3Kには12個を必要とする。

① アドレス・チップセレクト

CPUの出力アドレスのうち、AD0〜7は、PROM1702のアドレスに直接接続し、AD8〜15はデコードして、チップセレクトCSを作る。



第4－11図

本例では、0番地〜IK番地をPROMとするため、チップセレクトを上図（第4－11図）の如く、AD11〜15が0のとき発生させる様イネーブル端子に接続、AD8〜10でCS0〜CS7の8つのチップセレクトCSを作る。

ここで使用したデコーダLOW INPUT LOARD CURRENTのものを使用したが、TTLの7442等を使用するときは8080のアドレスバスの負荷が重くなるので、バファーアンプを間に入れなければならない。ここでAD11〜15はカードセレクトする。

② データバスゲート

アドレス及びチップセレクトが選ばれると、チップのデータ端

子に出力が出る。これをデータバスに接続するために、TRI STATEのゲート回路8212を使用する。

ゲートを開くタイミングはAD11～15のカードセレクトとDBIN・MEMRのときである。

③ 電源

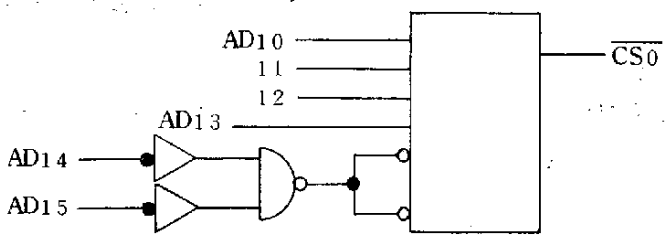
1702はプログラム書込みのときと、セットに組み込みREADするときとで電源接続が異る。

ピン	
12, 13, 15, 22, 23	VCC＝＋5V
16, 24	VGG＝－9V

(ロ) RAM

① アドレス・チップセレクト

2102形RAMは1K×1BJTであるから、1KWARDを作るには8個を使用し、アドレスAD0～9を各チップに並例に接続し、さらにAD10～15をデコードしてチップセレクトを作る。



第4－12(a)図

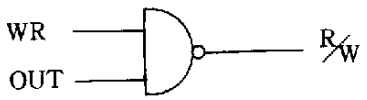
本例では、８Ｋ番地を使用する。

② データバスゲート

PROMと同様に、データバスのRAMのデータ出力を接続するゲートはカードセレクト及びＤＢＩＮ・MEMRで作る。

③ READ、WRITEの切換

RAMはREADとWRITEの両方に使用されるから、その切換が必要となる。そのためにCPUコントロール信号WR（メモリーリード）とステータス信号のOUTが使用される。



第４－12(b)図

④ データインプット

WRITEのとき、データバスの内容をメモリーに書くが、メモリーチップのＩNPUTにデータバスからダイレクトに各ビット毎に接続される。

(3) パラレルインプット回路

外部からの信号をプログラムコントロールで８ＢＩＴ（１WARD分）をＩNPUTするもので、命令のＩＮで実行される。

本例では入力データとして、スイッチ接点に接続される。

① バスとの接続

ＩNPUTの命令により、メモリ回路と同様にゲート回路で接続する。ゲートを開くタイミングとして

(a) ＤＢＩＮ（CPUのコントロール信号）

(b) ＩNP（CPUのステータス）

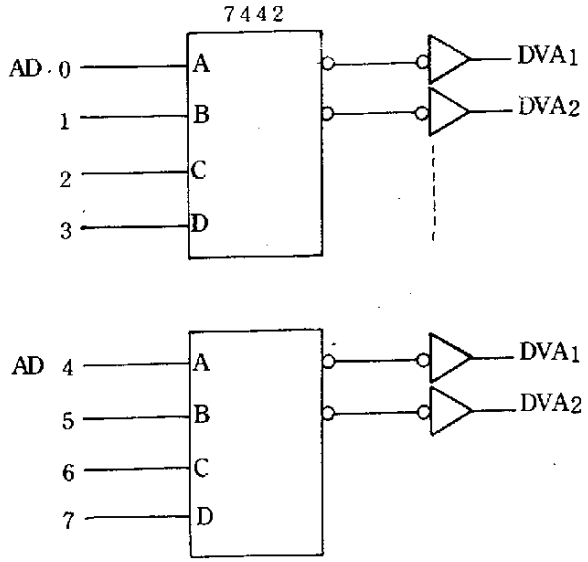
(c) アドレス（ディバイスナンバ）

アドレスはアドレスバスAD 0～7に入出力ディバイスナンバが、INPUT命令の実行時に出力されるから、あらかじめディバイス毎にアドレスナンバを決めておく。

ディバイスはアトレスバスAD 0～7の8BITをデコードすると2^8＝256種類を作れるが、4BITずつ2つに分けると考え易い。

即ち、AD 0～3でディバイスを決め、AD 4～7でディバイス内の入力、出力等の区別に使用し、この2つを組合せて使用する。

（第4－13(a)図参照）

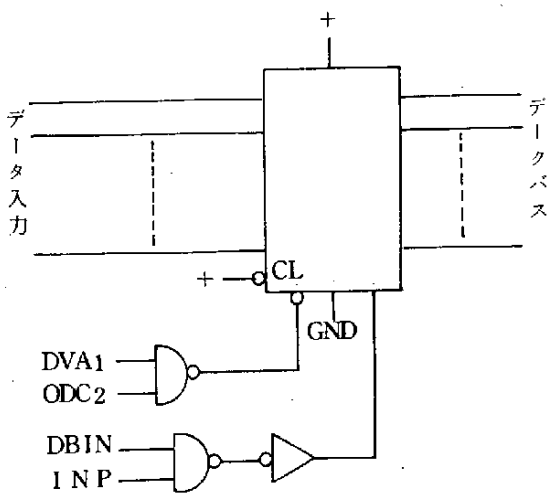


第4－13(a)図

第4－13(b)図の如く、データバスとの接続には、PROM、RAMインタラプインストラクションと同様に、TRI STATEのゲートを用いて接続する。ここではゲートとして8212を使用する。

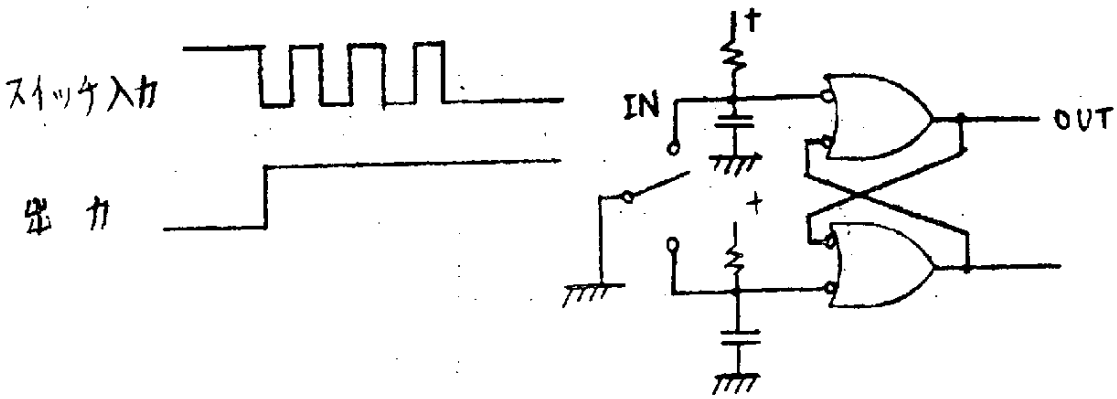
② スイッチ入力

スイッチ・リレーは一般的に作動時にチャッタリングがあり、CPUの読取り誤差を起こすことがある。これを防ぐため、スイッチ接



第４－１３(b)図

点には下記のような回路を通したのち、上記バス接続のゲートに接続する。

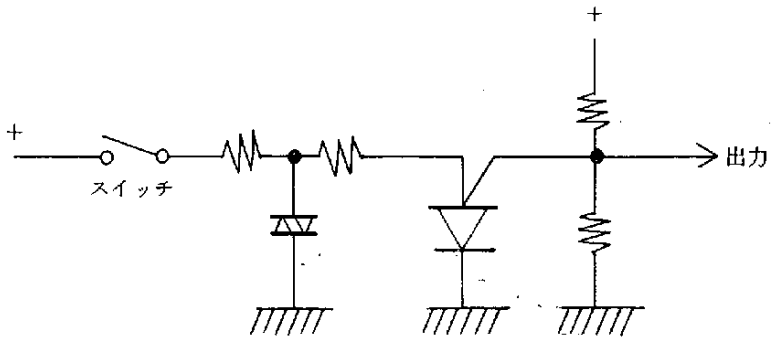


第４－１３(c)図

この回路は第４－１３(c)図の如く、スイッチが入っても瞬間的にバウンドして離れる、いわゆるチャッタをフリップフロップ回路のセット・リセットにより、防ぐものである。

しかしこの回路では、スイッチのバウンドが反対側まで及ぶときは効果がない。

この様な場合には、タイマ回路（積分回路）でチャッタ時間（10～100msec）をマスクする。



第4－13(d)図

③ ビジーチェック

上記のスイッチ入力の他、パラレル入力はデータ入力として種々の入出力装置に接続される。この場合、データが刻々新しくなるので、CPUが読み取る時、それをチェックする必要がある。これがビジーチェックといわれる方法で、データがセットされるとビジーフリップフロップをセットし、CPUでデータを読むとリセットする。CPUのプログラムは常にこのビジー信号を読んで判断する。ビジー信号の読み方としては、INPUT命令が行いデバイスコードをデータと同一にし、オーダコードを別にする。（第13(e)図）

ここでストローブはペーパーテープのスプロケット信号等であり、データはバスとの接続の項で記述した通りである。

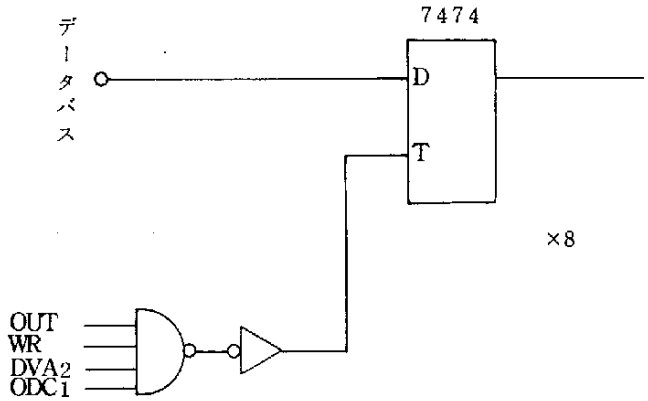
(4) パラレル アウトプット

プログラムコントロールでCPUから8BIT(1WARD)をOUTPUT

する。本例では出力信号でランプ表示及びリレーのドライブをする。

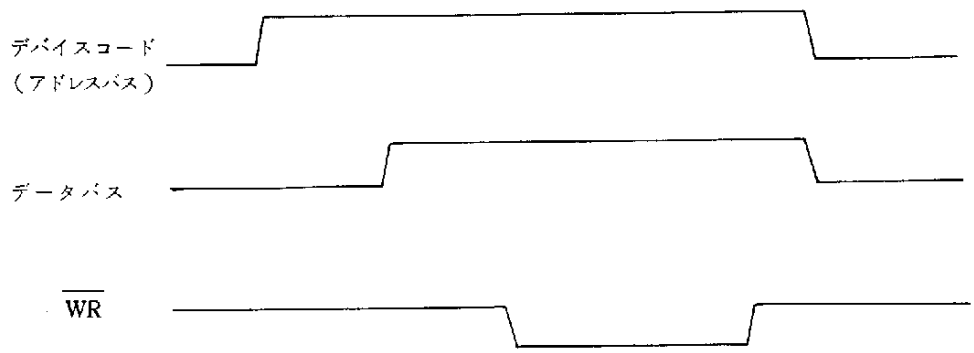
① バスとの接続

データバスにOUTPUT命令により一定のタイミングでOUTPUTされるので、これをフリップフロップで受け取る。



第4－14(a)図

データバスにデータがのるタイミングは下図（第4－14(b)図）のように出る。

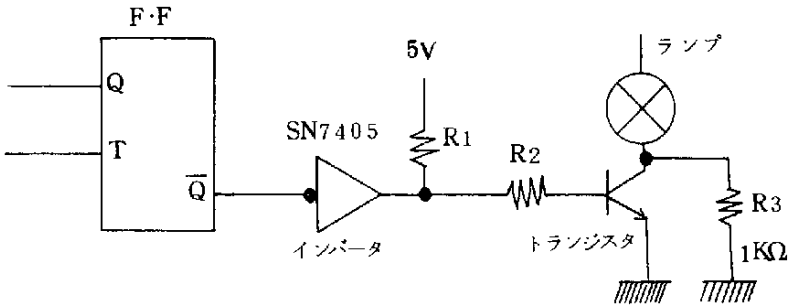


第4－14(b)図

したがって、アドレスバスをデコードしたディバイスコードとステータスのOUT及びWR信号のANDをとり、F・Fをトリがする。

② ランプドライブ

F・Fにラッチした出力によりランプを駆動するためのもので下図（第4－14(c)図）のようにトランジスタにより電流増幅する。



第4－14(c)図

トランジスタのベース入力はランプ負荷電流の$\frac{1}{\beta}$である。今100Aの負荷を駆動するとき、ベースに流す電流は、トランジスタのβを最少20とすると

$$IB = \frac{100}{20} = 5\,mA$$

従って

$$R_1+R_2 \leqq \frac{5-VBE}{5\times10^{-3}} \leqq \frac{4.5}{5\times10^{-3}} \leqq 0.9\times10^{3}\,\Omega$$

インバーダがONのとき、シンク電流は17mAMAXであるが、10mAに選ぶとすると

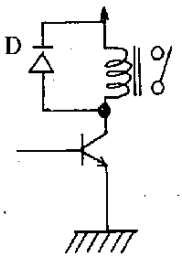
$$R_1 \leqq \frac{5-0.5}{10\times10^{-3}} \leqq 450\ \Omega$$

従って$R_1 = R_2 = 420\ \Omega$とする。

ランプは点灯時の突入電流が、定格時より数倍の電流が流れるので、これを軽くするためR_3により少電流をランプに流しておく。

③　リレードライブ

ドライブはランプと同様に、トランジスタにより行うが、リレーのインダクタンスに逆電圧が生じ、トランジスタを破壊しないようにダイオードDを持続する。



第４－14(a)図

(5)　キーボード・ディスプレイ　インターフェース

各々のファンクション機能や数字等の押ボタンを多数並べられたキーボードの信号入力には、入力転送の回数を減らすために、コード化してＩＮＰＵＴする。そのために、２つ以上押した時の処理、押したことをCPUに知らせるBUSY信号を作る必要があり、かなり複雑な回路となる。そこでキーボード用ＬＳＩとして作られたものがあるので、本例ではこれを利用する。

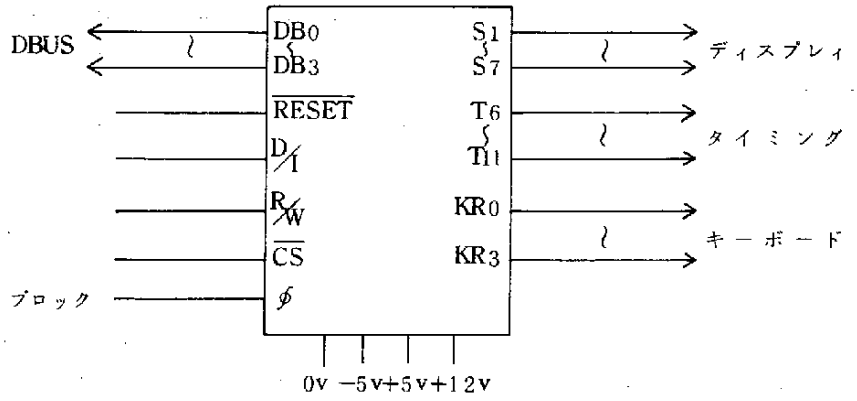
①　μＰＤ７５７

このＬＳＩは４ＢＩＴCPUのために作られたもので、データバスの０～３の４ＢＩＴと結合するもので、キーボードと同時に数字表示ディスプレイを駆動することができる。

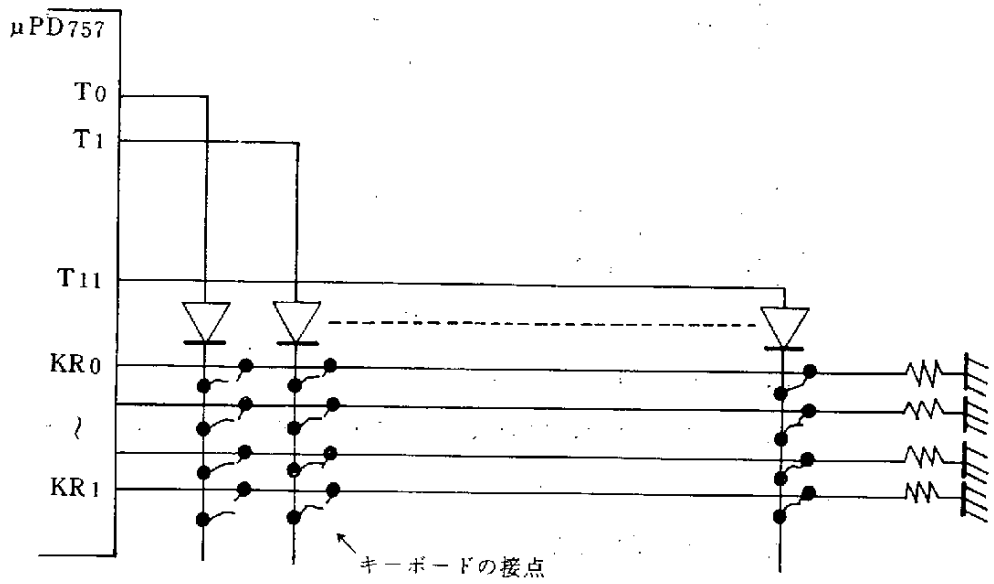
②　キーボードの接続

T_0～T_{11}のタイミング出力にキーボードマトリックスを接続し、出力をKR_0～KR_3に接続する。

この接続で４×１２＝４８個のキーボード接点を接続できる。



第 4 － 15 (a)図



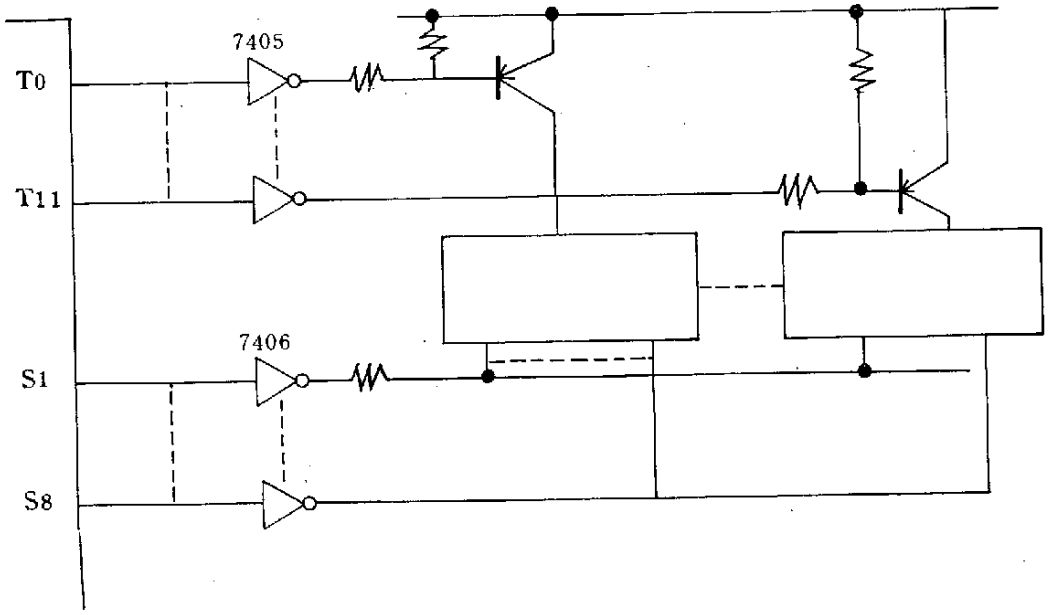
第 4 － 15 (b)図

③ ディスプレイ

7つのセグメント数字表示素子を接続できる。この素子の点灯は、パルス点灯式で行ない、T_0～T_{11}の出力タイミングで点灯する。

（次頁、第 4 － 15 (c)図）

④ バスライン・コントロールラインとの接続



第４－15(c)図

- データバス０～３にDB$_0$～DB$_3$を接続
- $\overline{CS}$、D／I、R／W端子

  $\overline{CS}$はディバイスセレクト(2)に接続

  D／Iはオーダセレクトに接続

  R／WはCPUのR／Wに接続、即ちINPUT、OUTPUT命令によりコントロールされる。

  （第４－15(d)図参照）

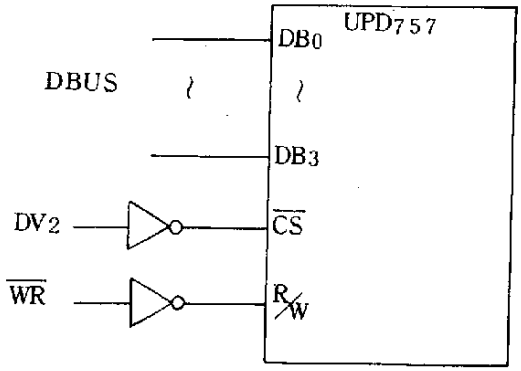
(6) 組立上の注意

① LSIチップの配置

CPU8080及びメモリーには、データバスは８本、アドレスには16本の線が並列に接続されるので、LSIの位置、方向を考えて取り付ける。

② ハンダ付

MOS形LSIは静電気やハンダゴテのリーク等で破壊することが



第４－15(d)図

あるので、ハンダゴテはアースを取りリークを防ぎ、ＬＳＩはハンダ付する前は、導電性の容器に入れておく。

③　ＰＲＯＭ

ＰＲＯＭは書込み器で書いてから接続し、又デバックのとき書かえを行ない易いように、ＩＣ用ソケットを使うとよい。

④　ＩＣ、ＬＳＩの電源にはノイズがのりやすいので、各ＩＣの電源ラインの近くには、0.1 μＦ程度のセラミックコンデンサを取り付ける。又、この目的のため、電源バイパス用タンタルコンデンサも販売されており使い易い。

4.3.3　調整

最終的な調整はソフトウェア完成後、ＰＲＯＭに書込んで行うが、それ以前に部分的に調整・作動確認できる。

(1)　ＲＥＳＥＴ信号の確認

電源の投入時にＲＥＳＥＴ信号が出ることを確認する。

(2)　クロック信号

仕様通りのクロックφ１、φ２のパルス中、相互のデレイタイムをチェックする。

(3) 各コントロール信号、ゲートの確認

調整用としてPROMにNOP命令の繰り返し及びJMP命令でもとに戻る。簡単なプログラムを書込むか、又は全てのPROM素子を除き、メモリ素子のOUT端子をアースすれば、CPUは常に0000……0を読み、NOPを繰り返す。

そこでアニマルのタイミング図に従って、CPUのコントロール信号のタイミング・バスゲートの開くタイミング、ステータスラッチをチェックする。

(4) メモリカード・入出力カード

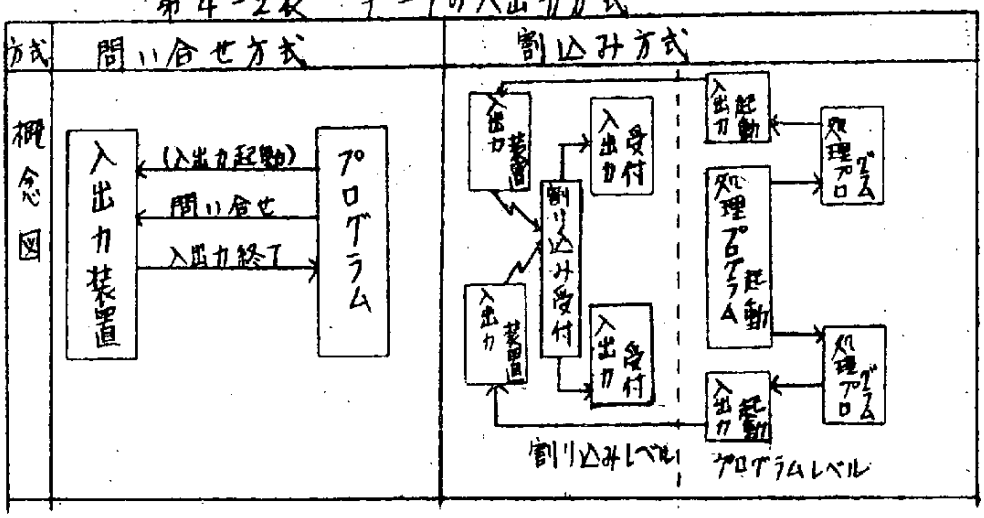
同様にPROMに調整用のプログラムでMOV命令、IN命令、OUT命令を繰り返すプログラムを入れれば、RAM、入出力装置用のカードのタイミング・作動をチェックできる。

4.4. ソフトウェア開発

4.4.1 ソフトウェア概要設計

ソフトウェアの概要設計では、ソフトウェアの回り、つまりハードウェアとのインターフェース、データの入出力方式を明らかにし、ソフトウェアのみの仕様書を作成する。この仕様書をプログラム外部仕様書とし、次段の詳細設計で記される仕様書をプログラム内部仕様書として分類する。外部仕様書には機能が記述されるのに対し、内部仕様書には実現方法、手段、手順が記述される。

第4-2表 データの入出力方式



方式	問い合せ方式	割り込み方式
説明	入出力装置からの入出力終了通知をプログラムで調査し、通知が発見されたならば、そのデータの処理又は新たなデータの起動をかける方式。	入出力装置に起動をプログラムで行い、その起動による入出力動作終了の割り込みを待つ。この待つ間に他にプログラム処理があればその処理を行う。 動作終了を割り込みにより受けそれを蓄積し、実行中の処理プログラムがなくなったときに、処理プログラム起動プログラム起動元の待ち状態を解き、実行させる方式
特色	○プログラム構造が簡素であり、それによって短い期で完成することが出来る。 ○プログラム間のインターフェースが簡素になる。 ○入出力起動から動作終了までの間、処理を行うことが出来ない。	○入出力起動から動作終了までの間に他の処理を並行して行えるので、マイクロプロセッサの有効利用が行える。 ○プログラム間のインターフェースが多くなる。 ○プログラムが複雑になり、完成までに期間がかかる。

4. 4. 2　ソフトウェア詳細設計

ソフトウェア概要設計が終ると、ソフトウェアの構造等を含め、ソフトウェアの詳細な設計を行う。この設計内容を内部仕様書とかプログラム仕様書とか呼ぶ。

内部仕様書はプログラムの構造に基づき、各プログラムの仕事の指図を明確に行うためのものである。

(1)　プログラム構造の確立

入出力方式に合ったプログラム構造を決定する。

(2)　メモリ　レイアウトの決定

プログラムの構造が決定すると、各プログラムのデータの受け渡し方法を定める。

(3)　サブルーチンの決定

各プログラム間に共通に使用する部分を明確にし、サブルーチンとする。既成のサブルーチン、パッケージなどの利用も開発工数をおさえるのに役立つ。

(4)　デバッグ方法の検討

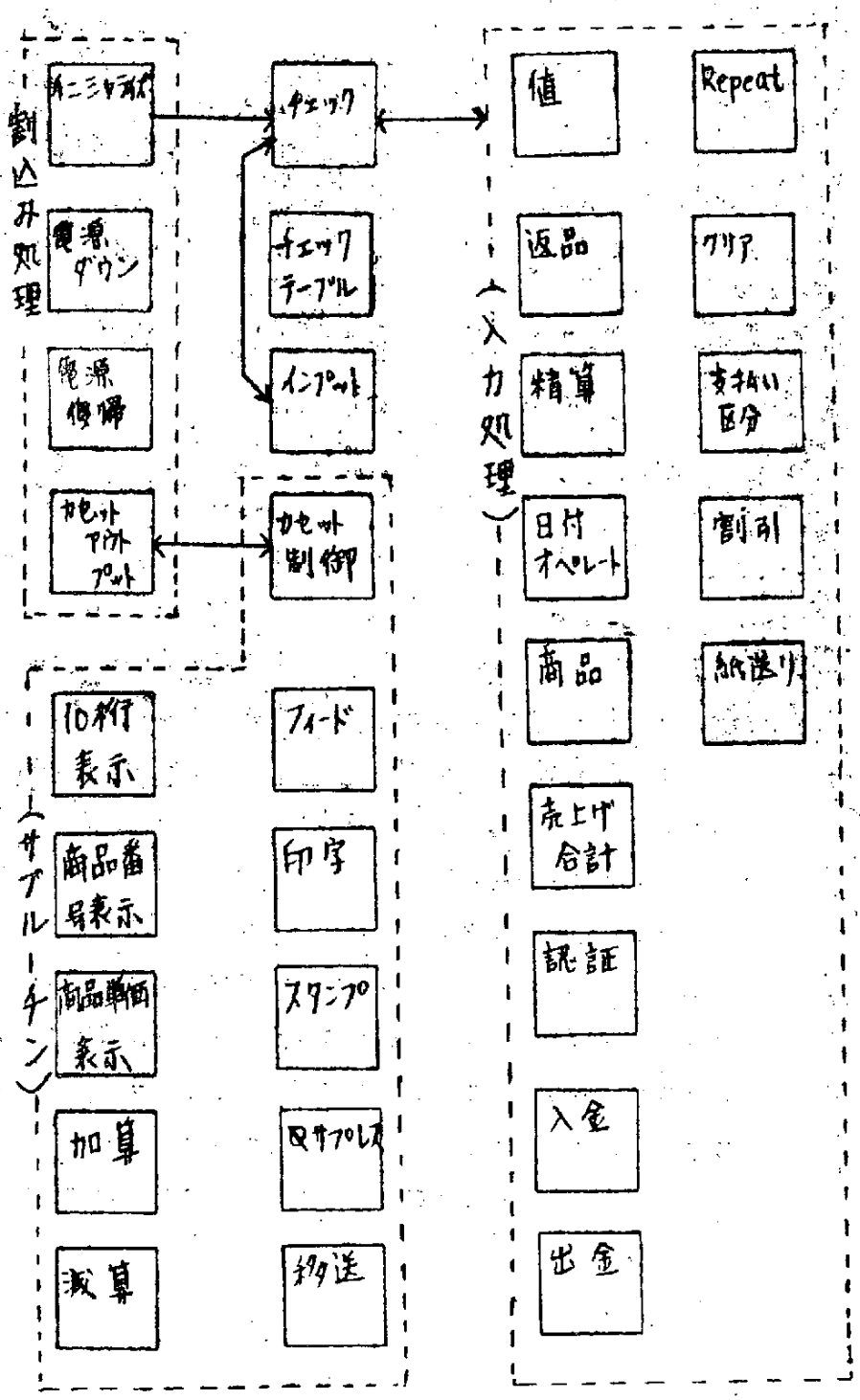
ソフトウェアを開発する工数の７０％～８０％はデバッグ作業であると言われ、このデバッグ作業の短縮化はプログラム間インターフェース、プログラムの大きさ、プログラム構造、プログラム仕様の明確度によるところが多い。よってプログラム仕様を作成するときには、このことに十分留意すべきである。

次にプログラム内部仕様書の実例を記述する。

1.　システム概要

1. 1.　機能概要

1. 2.　プログラム関連図


割込み処理
イニシャライズ
電源ダウン
電源復帰
カセットアウトプット
チェック
チェックテーブル
インプット
カセット制御
フィード
印字
スタンプ
Rサプレス
移送
(サブルーチン)
10桁表示
商品番号表示
商品単価表示
加算
減算
(入力処理)
値
返品
精算
日付オペレート
商品
売上げ合計
認証
入金
出金
Repeat
クリア
支払い区分
割引
紙送り

第 1 － 1 図

1. 3. プログラム一覧表

第 1 － 1 表

名　　称	ENTRY NAME	OPEN CIOSE	処　理　概　要
イニシャライズ		O	φ番地より実行され、RAMのイニシャライズを行う。
電源ダウン		O	8番地より実行され、各種レジスタを退避し、停止する。
電源復帰		O	10番地より実行され、各種レジスタを復帰し、停止前の状能にする。
チェック		O	システムテータにより、有効キーを抽出し、各入力処理に制御を渡す。
支払い区分		O	ディスプレイに表示されている額を支払い区分により売上げならば預として、返品ならば支払として蓄積し、印字する。
割　　引		O	割引コードにより、直前の商品単価から割引き、蓄積印字する。
10桁表示		C	ディスプレイに指定された値を表示する。
商品番号表示		C	ディスプレイの商品番号を1桁表示する。
商品単価表示		C	ディスプレイの商品単価を1桁表示する。

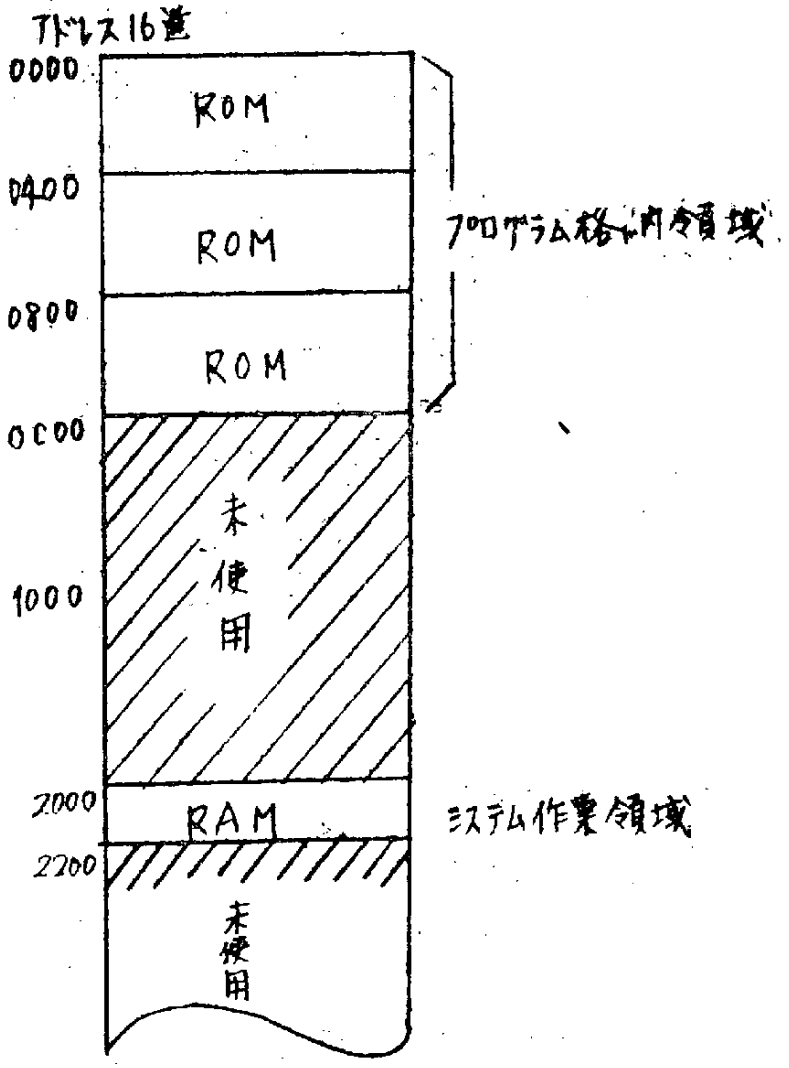
名　称	NAME	OPEN OLOSE	処　理　概　要
加　算		C	加数、被加数のフィールドナンバーにより加算を行う。
減　算		C	減数、被加数のフィールドナンバーにより加算を行う。
印　字		C	プリントエリアの内容を印字する。
フィード		C	ラインプリンタの5桁フィードを行う。
スタンプ		C	スタンプを印字する。
0サプレス		C	指定されたアドレスにより0サプレスを行う。
移　送		C	データを指定されたアドレスにより、指定されたアドレスへ移す。
紙送り		O	ラインプリンタのフィードを行ない、スタンプを印字する。

2. 入出力条件

入出力のための命令、データ形式、タイミングなどを記述する。

3. メモリ構成

3.1. メモリマップ



第 3 － 1 図

(1) システム作業領域

第３－１表

アドレス	説　　明
ⓐ WORK ２０００ ２００５ ２００A	 蓄積部 5byte×32フィールド 先頭アドレスのシンボル 名はⓐWORKである
２０A０	退避領域部
２０E０	システム情報部
２１１８	カセットレコード イメージ蓄積部
２１４C ２１FF	カセットブロック イメージ出力部

(イ) 蓄 積 部

各フイールドは全て 5 byte とする。

第 3 － 2 表

フィールドナンバー	内容		フィールドナンバー	内容	
0	値 ＝ 1		16	CR3	件数
1	売上げ件数		17		金額
2	売上げ物件数		18	貸	件数
3	売上げ額		19	売	金額
4	割引1	件数	20	返	件数
5		金額	21		物件数
6	割引2	件数	22	品	金額
7		金額	23	入金総額	
8	割引3	件数	24	出金総額	
9		金額	25	精算回数	
10	割引4	件数	26	商品番号	
11		金額	27	商品単価	
12	CR1	件数	28	割引額A	
13		金額	29	割引額B	
14	CR2	件数	30	金額合計	
15		金額	31	預金額	

(ロ) 退避領域部

命令 PUSH, POP により使用する。

(ハ) システム情報部

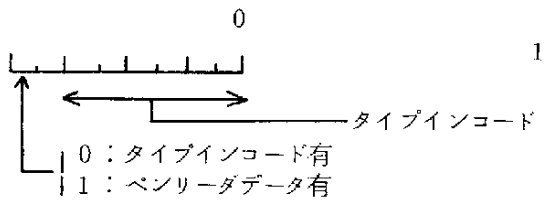
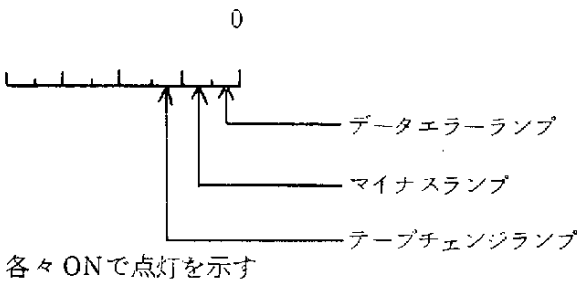
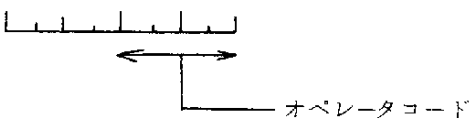
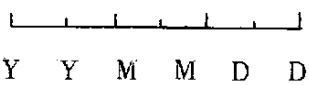
① システム情報部の構造

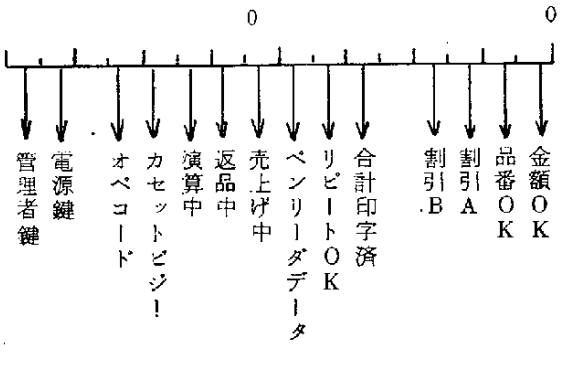
第 3 － 3 (a) 表

Symbol	内　容		Symbol	内　容
ⓐ KEY	タイプインコード		ⓐ CMT	カセット レコード 編　集
ⓐ LMD	ランプフラブ			
ⓐ OCD	オペレータコード			
ⓐ DAY	年 月 日			未 使 用 21 バイト
ⓐ SYS	システム ステータス			
ⓐ CM1	カセットテープ用	蓄積 ポインタ		
ⓐ CM2		出力 ポインタ		
ⓐ CMS		シークエン スナンバ		
ⓐ CMC		コントロール		
ⓐ YRB	割引コード A/B			
	未使用 2 バイト			
ⓐ LP	ラインプリンタ 出力イメージ			

② システム情報部の内容

第 3 － 3 (b) 表

Symbol	内　　容
ⓐ KEY	
ⓐ LMP	 各々ONで点灯を示す
ⓐ $\overline{\text{O}}$CD	
ⓐ DAY	

Symbol	内　　容
ⓐ　SYS	 各状態のときONとなる

以降この項については省く

3.2.　システムステータス

システムステータス（ⓐSYS）の各ビットは次表の通り更新する。

第 3 － 4 表

フラグ	ONとなるとき	OFFとなるとき
管理者鍵 ON	タイプインキーを読み取るとき、管理者鍵がONであればONとする	タイプインキーを読み取るとき、管理者鍵がOFFであればOFFとする

フラグ	ＯＮとなるとき	ＯＦＦとなるとき
電源鍵 RUN	タイプインキーを読み取るとき、電源鍵がRUNであればONとする。	タイプインキーを読み取るとき、電源鍵がRON でないときOFFとする。
オペレータコードOK	オペレータコードが入力されたとき、ＯＮとする。	イニシャライズでＯＦＦ。
カセットビジー	カセットテープに書き込み中のときＯＮ	カセットテープに書き込んでいないときOFF。
演算中	十又は一処理を行なったときＯＮ	演算合計を印字終了したときOFF
返品中	返品キーがタイプインされたとき	返品の認証印字が終了したとき。

以降この項については省く

4. プログラム詳細説明

4.1. イニシャライズプログラム

(1) 機　能

電源投入により0番地より実行され、システム作業領域の初期値設定などを行う。

(2) 入　口

0番地

(3) 出　口

チェックプログラム（EXTERNAL NAME XCHKO）

(4) 入出力情報

(イ) 入力情報

なし

(ロ) 出力情報

システム作業領域の蓄積部、システム情報部。

(5) 処　理

(イ) 蓄積部内の値1（フィールドナンバ0）に値1をセットする。

その他の蓄積部、システム情報部は全バイト0"0とする。

(ロ) システム情報部内のカセットテープ用蓄積ポインタ、出力ポインタをそれぞれ"2110"、"214C"にセットする。

(ハ) スタックポインタに"20DF"をセットする。

(ニ) 入出力制御用チップのイニシャライズを行う。

(ホ) ランプ、ディスプレイを消灯。

(ヘ) 紙送り5行の後、スタンプを印字する。

(ト) 割込み可能モードにする。

(6) 使用サブルーチン

(イ) 移　送

B、Cレジスタ：移送データアドレス

D、Eレジスタ：移送先アドレス

A　　レジスタ：移送バイト数

CALL　　ZMOVE

——→Return Point

(7) 無限条件

レジスタの使用制限、メモリ、制限などを記述する。

4.2. 電源ダウンプログラム

以降、4項については省く。

5. 制限条件

使用言語など、プログラム作成にあたっての制限条件を記述する。

4.4.3. プログラミング

ソフトウェア詳細設計が終ると、それに基づいてプログラムの作成にかかる。

(1) フローチャート作成

フローチャートの利点は次のものであろう。

o データ処理の手順、処理の動作順序が論理的によく整理される。

o フローチャートを書いた本人以外の人も理解しやすい。

このようなフローチャートを利用して、それに多角的な検討を加え、完全なソフトウェアにし、整然としたプログラムの流れを考え出し、効率のよいプログラムを作成できる。

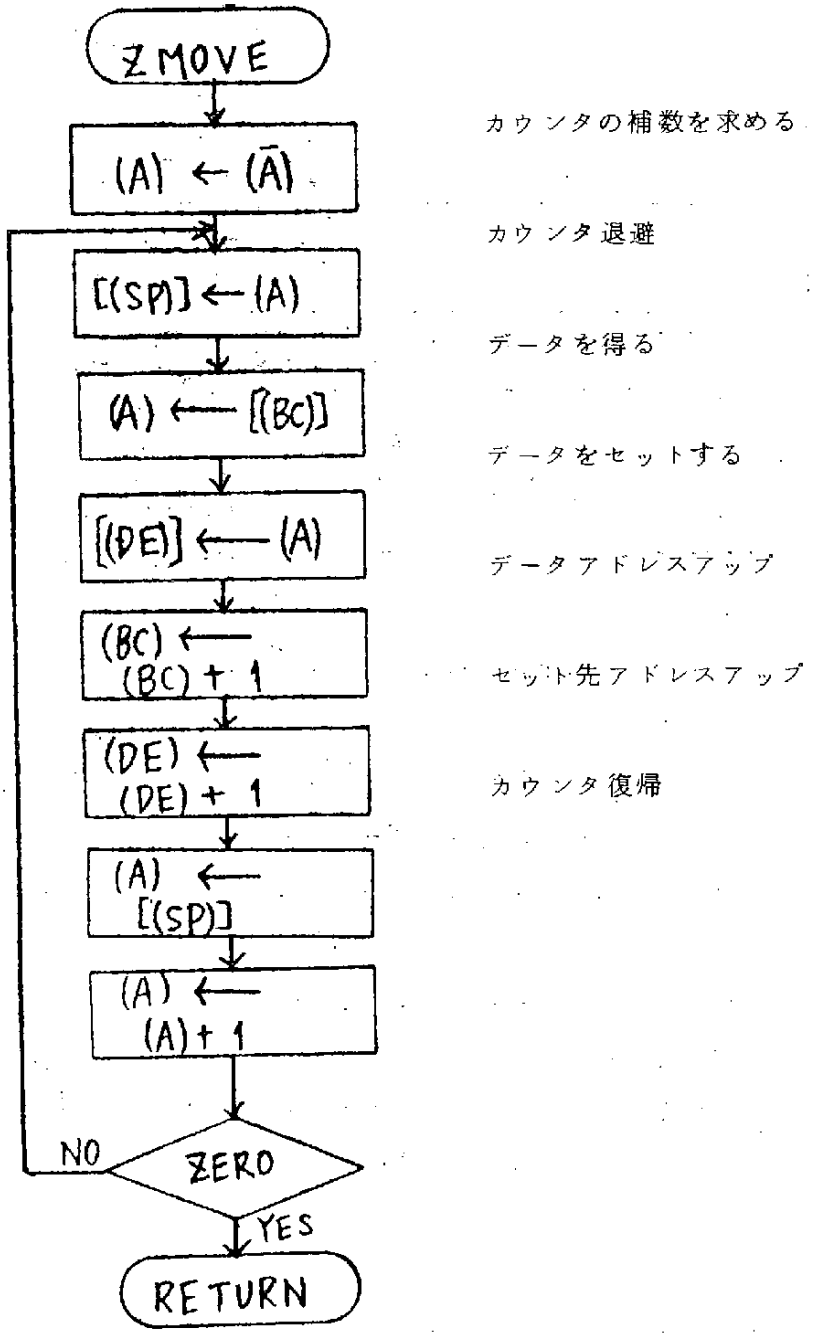
また、他の人にプログラムを理解させる有効な文書になる。

フローチャートを書く記号は統一されているが、その記号内に書く内容、方式についてもシステムで統一すべきであろう。

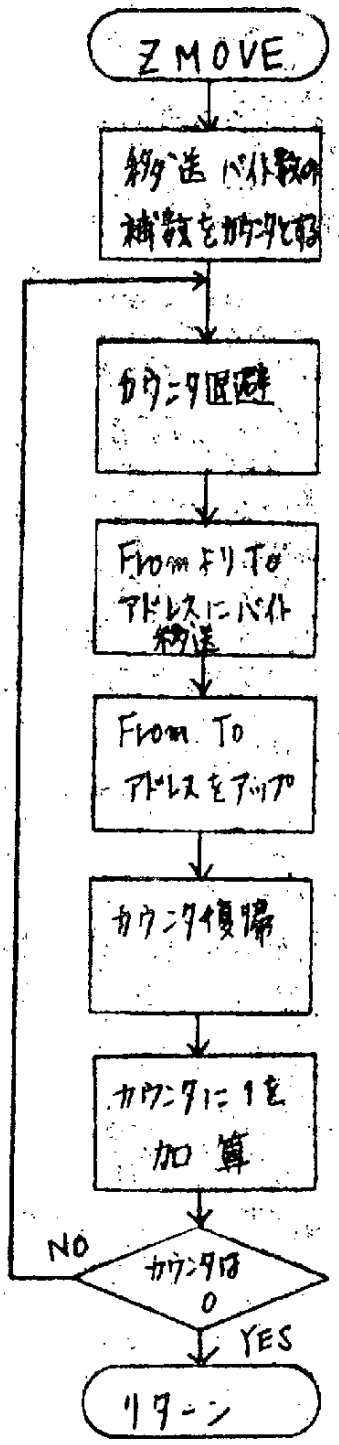
次にはフローチャートの実例として、2つの様式を記述する。こ

れはアセンブリ言語を使用する前提で書いた一般的なものであり、その他のプログラム言語ではまた異った様式になる。

（第４－１６図参照）



第４－１６(a)図　フローチャート例（様式１）



第４－１６（b）図図フローチャート例（様式２）

(2) コーディング

コーディングはフローチャートに従って、各プログラム言語で記述し、誤りをなくするためや、プログラムを理解しやすいようにコメントを書く。これを紙テープあるいは紙カードにパンチし、ホストコンピュータなどによりアセンブルし、オブジェクトテープを作成する。

(3) デバッグ

デバッグとはプログラムの中の誤りを見つけて、それを修正し、完全なプログラムを作り上げる作業である。システムは、目的とする処理が正しく行えることが大前提であり、運用上のあらゆる異常事態が発生してもそれを検出できることである。あらゆる場合を想定し、デバッグを行うためには次の点に注意する。

- o デバッグ日数に合わせた安易な妥協をしない。
- o すべてのロジェックに渡ってあらゆる状態を想定したチェックを行う。
- o 数人でデバッグを行う場合はチームワークに気を配る。
- o デバッグは効率よく行う。
- o デバッグ期間、工数は少ない程よい。
- o 計算機の使用時間は少ない程よい。

(イ) デバッグ方法

デバッグ方式はシュミレータ、デバッガなどと呼ばれるソフトウェアサポートによるものと、デバッグ評価マシン、試作機などのハードウェアサポートによる方式がある。

共通して行われる方法として、ほぼ次の2つがある。

- o ハードウェアによる方法

コンピュータを中心に使ってデバッグする方法である。この方法は小さなプログラムを短期間でデバッグするのに有効であるが、コンピュータの使用時間の割にデバッグ効率は悪い。

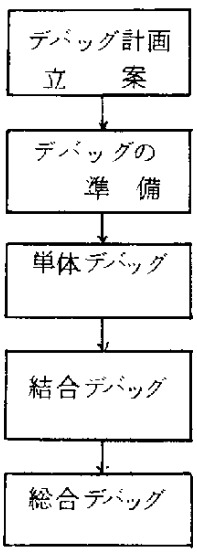
o 机机上デバッグによる方法

ドキュメントの修正、検討を十分に行い、デバッグする準備を行なってコンピュータで実験し、その結果をコンピュータの前から離れて机上で検討することを繰り返す。この方法はコンピュータの使用時間が少なくなる。

前記の２つの方法で、プログラムの大きさ、デバッグする人の経験などによりデバッグを行う。

(ロ) デバッグの手順

一般的には、次の第４－１７図に示すように進められる。



第４－１７図

(ハ) デバッグ計画立案

デバッグ計画を立てるに当って、デバッグに利用できる全ての資材を整理し、各資材の効果的な利用方法を確認しておく。

資材には以下のようなものがある。

o ソフトウェア

言語プログラム

デバッグコーティリティ

サポートプログラム

シュミレーション

o ハードウェア

ホストコンピュータ

デバッグ評価マシン

試作機

ROMライタ

o 要　員

o その他

パンチセンタ

各種マニアル

これらを総合し、デバッグ計画を立案する。

第4－3表に、デバッグ計画の1つの案を示す。

(ニ) 単体デバッグ

プログラムの分割が徹底しているシステムでは、プログラム1本毎がステップ数も少なく、デバッグも比較的容易である。

しかし、単体デバッグを緻密に行うことがシステムの早期安定に最も必要なことであり、十分に行うことが大切である。

o 正常テスト

テストするプログラムが正常なデータで正しく処理することを確

第４－３表　デバッグ計画表

項　　目	期間 / 担当	8/1　9/1　10/1　11/1	備　考
		外部装置　ラインプリンタ　キーボード・ランプ　カセットテープ等	
検査項目リスト作成		←→ 汎用マイクロコンピュータと外部装置の結合	
汎用マイクロコンピュータの使用		←→	
クロスアセンブルシステム及びPROMライタ使用		←→	週１回予定
単体デバッグ		←→ イニシャライズ	
		←→ チェック	
		←→ インプット	
		←→ カセット制御・カセットアウトプット	
		←→ サブルーチン各種	
		←→ その他	
結合デバッグ		←→	
総合デバッグ		←→	

認する。データも許される境界限度のぎりぎりのものまでテストする。

o エラー処理テスト

システムに投入されるデータの誤りに対してどのように処理するかはシステムの信頼性にもつながり、異常データテスト入力、出力のエラー処理などを繰り返しテストする。

o 総合テスト

正常テスト、エラー処理テストをプログラムの中で個々の条件を組み合わせて行う。

(ハ) 結合デバッグ

単体デバッグが完了したプログラムを結合して、各プログラム間のインターフェースをテストする作業である。

結合デバッグの要領として次のことに留意する。

o 一度にシステム全体をテストせずにある単位に分割し、それ毎にテストする。

o サポートシステムを十分利用する。

o 単体デバッグと同様に、正常テスト→エラー処理テスト→総合テストの手順で行う。

(ニ) 総合テスト

システム全体に渡って、その正常性及び性能をテストする作業を総合デバッグという。

この時点でメモリーがM－ROMであることは難かしく、P－ROMで通常テストされる。

総合テストの要領として次のような点をあげることができる。

o ソフトウェアは完全に結合デバッグが終了していること。

o ハードウェアも十分に調整済であること。

o テスト準備でテスト方法を決めておくこと。

o操作ミスを行なってもシステムが暴走しないこと。

o運用上のすべてのタイミング条件を作り出すよう考慮すること。

総合テストで一番の障害は、ソフトウェアの原因によるか、ハードウェアの原因によるか判定できかねるときであり、この解決のためにソフトウェア、ハードウェアの担当者は積極的に協力しなければならない。

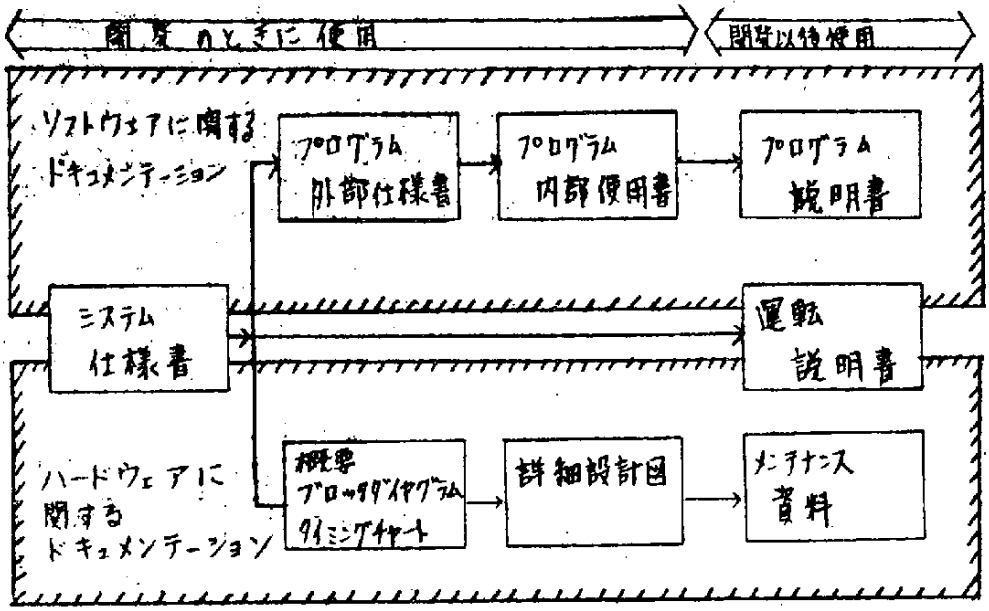
(4) P－ROM、M－ROM化

P－ROMともM－ROMともプログラムが格納されるが、格納するためにP－ROMはライタが手元にないときは日のオーダで、M－ROMは月のオーダで日数が必要である。

とくにM－ROMのときは、システムの完成以後行うべきで見切りで行なってはいけない。このM－ROM化のタイミングは十分に検討を必要とする。

またP－ROMにおいても各チップに空を作っておき、プログラムの変更、修正がある場合にも全チップを書き直す、あるいは作り直すことを避ける配慮も必要である。

4.5 ドキュメンテーションとメンテンス



第４－１８図

(1) ドキュメンテーション

ドキュメンテーションには第４－１８図のようなものがある。

(イ) プログラム説明書

プログラム内部あるいは外部仕様書は「プログラムが何をするか」を、プログラムを作る前に明確化したものであるのに対し、プログラム説明書はプログラム開発後、「プログラムは何をどのようにしたか」を明確にしたものである。

尚、プログラム説明書の内にはフローチャートも含まれ、プログラ

ムの処理の流れを図示する。

(ロ) メンテナンス資料

故障及び定期点検を行うためのマニアルである。記述すべき内容は作動原理、構成、各部の点検項目とその調整法、故障診断方法等である。

(2) メンテナンス

定期点検はMTBFを考慮して一定の間隔をあらかじめ設定し、ハードウェア各部をメンテナンス資料に従って行う。

昭和５２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３－５－８
機械振興会館内
〒105　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
〒101　TEL(254)5840(代表)