

電子計算機

オリエンテーション

(財)日本情報処理開発センター



· ·	八世 りますのでき で	きてくなる いま	がはいかからいん	はいるとはないという							
	el de la companya de	門を含めている。		ないない。	ないないない						
	经外面的 医多种的			() () () () () () () () () () () () () (の対するのである				6		
		() () () () () () () () ()	经是特别的								
							4				
						では、 これのでは、 一般の対象				0/1	
		はいいかのかの	となる かない						ار د د د د		
		(1) (1) (1) (1) (1) (1) (1) (1) (1) (1)	がある。			いない					· ·
						から くりん	八年 一年				
							というのでは、				
				1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、							
		まます。 では、2000年 ・ 1000年 ・ 1	というないのからい				がなめ、				
						さい かいきして 質問	作業のおかる			7.7	
T				である。		A TOTAL TOTAL					
	通り かいかい			二字 多洲汉人	(1)						
	1. 「「「「「こう」です。	があるとなっては、			こう でんしょう	が、 1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.					
			こう ない とうしゃ	がある。	かした。		がはいるできた				
	いまとう おかられる			える。例の言う		が変化される。	となる ちょうない	がかがたかか	は、意味ない		化二烷基 化异
			はいいいかという			大き					
						後の一般の					
			いまからい	大学など からかい		がない。					
						明 一					
					は、古人教の		ないとなる				
		強めないという		されているというできますが					***		
		が放送している。									
						というというとはいい					
		ない。	经营业 医骨骨骨 医	というない	語の語の場						a)
				一次の ない ないのい			が大きなな				
			では ないない ないない	一次成長の大き							•
				1000年							
	一直 一										
									とうない 大変ない		
	· · · · · · · · · · · · · · · · · · ·	ない。これにいる				文章がは、ない					
				あれるのかの						-7	,
											*
			阿拉斯斯特的	の対象の	は、ころでは、	ではなるというない。					
		はいっている		源される。		されることも	には多くにはなる				
						は人が大き					
			が、一般のでは、								• 7
			· · · · · · · · · · · · · · · · · · ·						10日本の大学		
	いかがいたが、タ	高いない		はない 大きの あり			はないできょう				
				大学の大学の	では、		京都 できず	学者に対する			
							いけんがあ	The state of the s			
			李·杨·杨· · · · · · · · · · · · · · · · · ·				* 100 mm 1				
			対の行為が								
			関係を持ちま	的感染的學行	10年の日本の						
•		大学 というないこと からの									
	(a)				1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	A CONTRACTOR OF THE PARTY OF TH	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1	A STANDARD OF THE STANDARD OF		

情報処理とコンピュータ

	·					
		~		•		
			•			
						•
					٠	
					•	
					•	

情報処理とコンピュータ

1. 情報処理の基本概念

社会の発展や経済の高度化にともなう情報(information)量の増大と、コンピュータの技術進歩を背景として、コンピュータによる情報処理(またはデータ処理)が急テンポで普及しつつある。

これは時代の要請によるもので、産業界に例をとってみると、めまぐるしい環境の変化に即応した企業の 適切な経営、巨大化した企業活動のコントロール、研究開発のスピード・アップなどは、コンピュータの有 効な利用によってはじめて可能になる。

もともとコンピュータは科学・技術計算の必要性から開発されたので、当初はその名のしめすごとく、電子的な速さで、人間の手におえない複雑な計算をする「巨大な高速計算機械」ぐらいにしか考えられていなかった。その後、より高度な利用を要望する声と、それを解決するための技術の進歩によって、単なる計算機械から、互いに関連し合った大量のデータを取扱うことのできる「情報を処理する機械」としての性格を強めてきた。

情報(データ)処理ということを、ひとことでいい表わすと「目的に合わせて必要な情報を、必要とする時に、必要とする人に提供する機能」であるといえる。このような機能を果すには、物理的につぎの四つの要素が必要となる(図1)。

- ① 情報を収集し、それを処理する段階に送る(コンピュータではインブットするという)。
- ② 送られたデータを整理保管し、計算したり、すでにファイルしてある記録と照合して、その記録を更新する。
- ③ また、ある形式に整えて記録しておく必要のある情報は、それをファイルしておく。
- ④ 処理結果,あるいはファイルされている情報を,必要に応じて利用者に提供する(コンピュータではアウトプットするという)。

情報を処理する機械としてのコンピュータは、このような情報処理の機能を満足させる能力をもっており、その適用範囲は事務の能率化、生産とか販売活動などの計画や管理、各種の技術開発、生産機械の自動制御などあらゆる分野に拡大した。しかも、最近では、通信回線(電話線など)との結合によって遠隔地から利用できるようになり、国鉄の緑の窓口、航空機の座席予約、旅館の予約、銀行の預金窓口などにその例をみることができるように、新たな応用面を開拓しつつある。

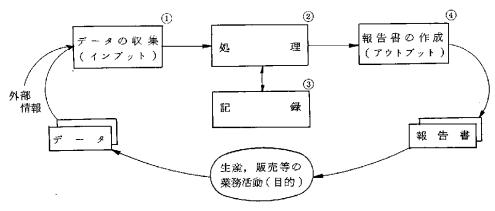


図1 情報処理の四要素

21 コンピュータの仕組み

2.1 コンピュータの仕組み

コンピュータは入力装置(in put), 記憶装置(memory), 制御装置(control), 演算装置(arithmetic) および出力装置(out put) で構成されている。

これらの装置の関係について述べると、ある形式にととのえられた各種の情報(データとかプログラム)は入力装置を介して記憶装置に貯えられる。プログラムは記憶装置から逐次、制御装置に送られて解読され、指令となって他の四つの装置を制御する。つまり、制御装置の指令によって入力装置から読み込まれ、記憶装置に納められたデータは、やはり制御装置からの指令によって記憶装置から演算装置に送られる。演算の結果は記憶装置に送り返される。記憶装置と演算装置の関係はメモ用紙とソロバンに似ている。計算結果は記憶装置から出力装置を介して取出される。この間の操作はプログラムにもとずいてすべて自動的に行なわれる(図 2)。

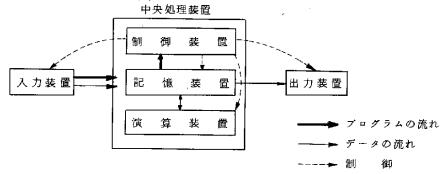


図2 コンピュータの基本的な構造

コンピュータの改良、発展をみると、より大容量の記憶装置、より速い処理能力に加えて、入出力装置、各種の付属装置等についても著しい進展がみられる。すなわち磁気テープ装置の書込み、読出し時間のスピード化および大容量化から、さらに出力装置の高度化などがそれである。さらに活字で印刷された文字をそのまま読みとることのできる光学式文字読取機とか、コンピュータで処理した結果をブラウン管に しだす表示装置などもある。これらと同様に重要なものとして、データ収集装置、問合わせ装置、データ通信との結合などがある。これらの発展は、今後ますますコンピュータの利用促進に寄与することであろう。

2.1.1 入出力装置

コンピュータはプログラムとかデータを機械の中におさめてしまえばあとは高速度で自動的に処理できるが、その出し入れに時間がかかったのではコンピュータの能力が半減することになる。入出力装置はインプット、アウトプットすべき情報の媒体によって異なり、表1に示したようなものがある。

装置の区分	装	置	名	速	度	記録密度	
	カー	ド 読	取 機	200~1,600	枚/分	80,90字/秒	
1 七 #	紙 テー	- プ 該	1. 取機	200~1,000	字/秒	10字/25㎜	
入 力 装 置 	電動タイ	゚プライ:	タのキー	手 動			
	(磁気	テープ	装置)	10,000~15	0,000字/秒	10~1,000字/2	5 π α π
	カー	ドせん	, 孔 機	100~500₺	女/分	80,90字/枚	
	紙テー	・プせ	ん孔機	10~2005	字/秒	10字/25 mm	
出力装置	電動タ	・イプ	ライタ	10字/秒			
	ライ:	ンプ	リンタ	200~1,500	行/分	120字/行	
	(磁気	テープ	装置)	10,000~15	0,000字/秒	100~1,000字/2	5 mm

表 1 入出力装置

表1以外にもマグネチックインクによって文字を記録し、それを読みとる装置(MICR)とか、タイプされた文字を光学的に読みとる光学式文字読取装置、あるいは計算結果を図形にして画く出力装置(XYプロッタ)など特殊なものもあるが、ここでは省略してごく一般的なものをあげた。

人力媒体としては、通常カードか紙テープが使用される。これらの媒体は第一次媒体ともいえる もので、伝票などの原始資料からせん孔機を使用し人手によってせん孔される。出力についても種 々の形態が考えられる。

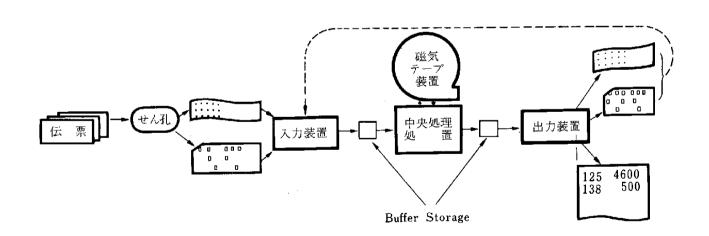


図3 入出力装置と中央処理装置の関係

もっとも一般的なものとしては、結果を連続用紙に印刷する方法で、最終出力ともいえる。この 他に、結果をカードとか紙テープにせん孔する方法もあるが、これは中間的な出力で、再びコンピュータに読みこませる必要があるときに使用される。

入力媒体を読みとる機器が入力装置で、結果を外に出す機器が出力装置である。入出力装置は機

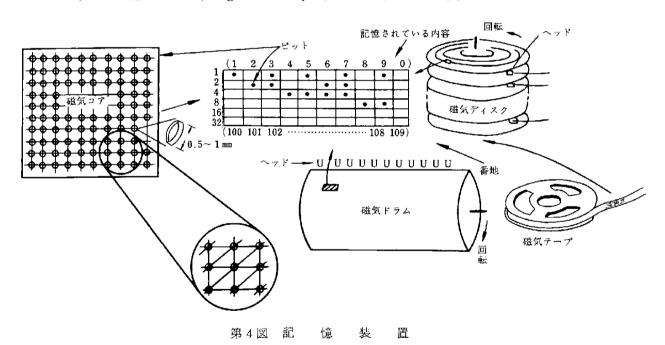
械的な動きが伴なうため中央処理装置の処理速度よりも遅いのが普通で、入出力装置によってコンピュータ全体の速度の低下を避けるためにいろいろな工夫がされている。

すでに述べたように各機器の速度をあげることは当然であるが、その他に入出力装置と中央処理装置との同時操作がある。これは、入出力装置と記憶装置の間に緩衝用の記憶装置(buffer storage)をおき、記憶装置は入出力装置と直接データのやりとりをしないで、この buffer storage を 通して行なう。 buffer storage と中央処理装置の記憶装置との間のデータの移動は電子的な速度であるから、直接に入出力装置と往復するよりもはるかに短時間ですむ。また、入出力装置と buffer storageとの間のデータの移動は中央処理装置での並行できる。

さらに、利用の面でもいろいろ工夫されている。つまり、第一次入力媒体が大量で、しかも、何度もコンピュータに読みこませる必要がある場合は第二次的な入力媒体としての磁気テープに変換し、以後それを使用するとか、中間的な出力には磁気テープを使用する方法である(図3)。

2.1.2 記憶装置

記憶装置は入力装置で読みとった情報を記憶し、その情報のうち、指令となるべきプログラムを制御装置に供給するとともに、必要なデータを演算装置に伝達し、演算結果を以後の演算のために、あるいはアウトブットするまで貯える装置である。現在最も多く使用されている記憶装置には磁気コア(magnetic core)、磁気ドラム(magnetic drum)、磁気ディスク(magnetic disk)および磁気テープ(magnetic tape)がある(図3参照)。



記憶装置に情報を記憶したり、記憶内容を読みとったりするには多小時間がかかる。この時間をサイクルタイム(磁気コアの場合)またはアクセスタイム(磁気ドラム、磁気ディスクおよび磁気テープの場合)とよぶ。所要時間は磁気コアが最も短かく、ここに記した順番にしたがって長くなる(表2)。

記憶装置には、その機能によって主記憶装置と補助記憶装置(大容量記憶装置,ファイルあるいは外部記憶装置ともよばれる)がある。前者には磁気コアが、後者には磁気ドラム、磁気ディスク

表 2 記憶装置の比較

Ä	己憶装置の種類	計算機の大きさ	大体の記憶容量	平均アクセスタイム またはサイクルタイム
主記憶	磁気コア	小 型 中 型 大 型	4 千字~ 3 2 千字 3 2 千字~ 1 3 0 千字 1 3 0 千字~ 1,6 0 0 千字	1/5~10 #s
補助記憶	磁気ドラム 磁気ディスク 磁気 テープ	(ある計算機の例) (") (")	2,600千字(1台) 7,250千字(1台)米 135,000千字(1台) 10,000千字~ 20,000千字(1巻)***	27.5 ms 87 ns 220 ms 2~3 min

μs (マイクロ・セコンド): 百万分の1秒

ms (ミリ・セコンド):千分の1秒

min:分

* 比較的小容量で取換え可能の磁気ディスク (ディスクパックと呼ぶ)。

*** 磁気テープはテープのリールを取換えることにより無限の記憶装置となる。

および磁気テープが使用される。主記憶装置は補助記憶装置に比して高速であるが、記憶容量は少ない。一方、補助記憶装置はファイルの役目を果すため、特に大容量であることが望まれ、装置 1台で数百万字から 2億字位まで記憶できる。計算機の規模によって相違はあるが、このような記憶装置を数台から 10数台連動できる。

なお、どのような種類の記憶装置でも、一度記憶された情報は半永久的に消えないで、何度でも 必要に応じて取りだすことができる。しかしある情報を新たに書きこむと、その場所の以前記憶さ れていたものは消失して、新しい情報におきかわる。

記憶の単位

コンピュータでは、数字、カタカナ、英字および特殊記号を取扱えるが、これらの文字はビット (1ビットは1個の2進符号)の組合せによって表現する(表3)。

ビットの組合せ方にはつぎのような方法があり、組合わされたビットの一群が記憶の単位となる。

表3 数の表現(一例)

十進数	= :	進符 号(ビット)
一	1	2	4	8
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	. 0	0	1
9	1	0	0	1

- ① バイト方式, 1 バイトは 8 個のビットで構成され, これで 1 字を記憶する。ただし数字は 4 ビットの組合せで表現することができるので, 1 バイトで 2 桁の数を記憶できる。
- ② キャラクタ方式, 1キャラクタは6または7ビットで構成され, 1字(数字を含む)を記憶する。
- ③ ワード(語)方式、1ワードは数10ビット(計算機によってビット数は異なる)で、これで 10桁前後の数(文字の場合は、数字より記憶できる字数が少なくなる) を記憶できる。

以上述べた記憶の各単位には、それぞれ番地(address)がつけられている。たとえば、4,000 バイトの容量がある記憶装置では、0番地からはじまって3,999番地まである。ワード方式の記憶装置は1ワードを単位として記憶したり、取り出したりするが、バイト方式とキャラクタ方式は一度に記憶したり、取り出したりする字数の長さをそのつど任意に変えることができる。

2.1.3 制御と演算装置

制御装置は命令レジスタ(order register) と制御計数器(control counter)からなり、記憶装置に貯えられているプログラムからつぎつぎに命令をとり出し、それを解読して他の装置をコントロールする。制御計数器はつぎに実行すべき命令が記憶されている番地を示し、命令レジスタは制御計数器によって指示された番地の内容をとり出し、これを命令として実行する。

演算装置は制御装置からの指令によって働らく。もっとも基本的な演算装置は累算器(accumulator)、被乗数と除数レジスタ(multiplicand and divisor register) ならび に乗数と商レジスタ(multiplier and quotient register)の三つのレジスタで構成されている(図 5)。

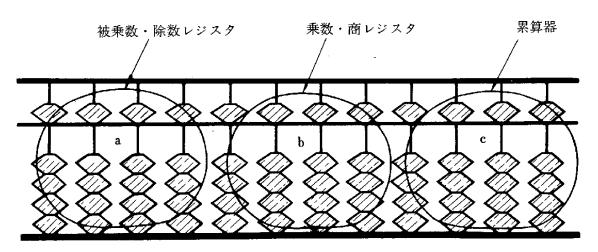


図5 演算装置のレジスタ

加減算では命令によって記憶装置から取り出された数が累算器の内容に加減される。これはソロバンの加減算と同様である。累算器の内容は必要に応じて再び記憶装置に記憶することができる。乗除算では被乗数・除数レジスタ,乗数,商レジスタおよび累算器が使用される。乗算の場合,まず記憶装置から被乗数 a を被乗数・除数レジスタに移す命令を与え a を移す。つぎに記憶装置の乗数 b の入っている番地を指定して乗算の命令を実行すると,乗数 b は自動的に乗数・商レジスタに移り a × b が行なわれ,積 c は累算器に置かれる。除算はこの逆で,被除数を累算器に移し,除数の入ってい

演	算の種	類	コンピュータA	コンピュータ B	コンピュータ C
加	滅	算	4 4 µs	24 #s	8 # s
乗		算	440#s	229#5	24 #s
除		算	1,090#s	550#s	42#s

μs (マイクロセカンド) …………100万分の1秒

表 4 演 算 速 度

いる番地を指定して除算の命令を実行すると、除数は被乗数・除数レジスタに入って c ÷ a の計算が行なわれ商 b は乗数・商レジスタに出る。剰余があれば累算器に残る。また、演算装置の内容は必要に応じてご破算することが可能であり四捨五入とか桁移動、数の大小の比較などができる。

演算装置の機能はソロバンとよく似ているが速度は比較にならない。その一例を示すと表 4 のとおりである。

累算器は第1累算器と第2累算器があり、それぞれ記憶装置の一語分のデータを入れることができる。他の二つのレジスタもそれぞれ一語分の容量である。

図 6 で制御計数器の示す数"0013"はつぎに実行すべき命令の入っている記憶装置の番地を

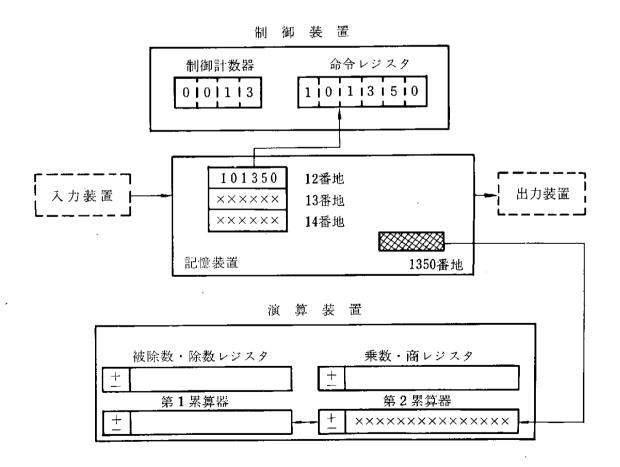


図 6 制御装置と演算装置

指し、命令レジスタの内容"10 1350"はいま実行しつつある命令で、この命令の実行が終ると13番地の内容が命令レジスタに入る。このとき、制御計数器には自動的に1が加えられて"0014"となる。つぎは、13番地より命令レジスタに移された命令が解読されて、それが実行される。この命令の実行が終ると、制御計数器の内容はすでに"0014"となっているので14番地の内容が命令レジスタに移され、まえと同じような操作が繰返される。

2.2 プログラム

与えられた問題を計算機で解くには、その問題を分析して、一連の処理手順を作成し、それを命令として計算機に与えなければならない。この処理手順がプログラムで、プログラムをつくることをプログラミングという。プログラムを構成する個々の命令は、基本的には2つの部分で組立てられている。たとえば、図6の命令レジスタの内容"10 1350"が「累算器に1,350番地の内容を加える」ことを指示する命令とすれば、上位2桁の数"10"の操作部(operation part)といい、これによって累算器に加算することが命令され、下位の4桁"1350"を番地部(address part)とよび、この部分で指定された記憶装置の番地の内容が操作部の指示どおり処理される。

しかし、すべての命令の番地部が記憶装置に関係しているとは限らない。その1例として、第1累算器 に入っている数を第2累算器に移すため右に桁移動する場合、その命令の番地部は移動させる桁数を示 すことになる。

以上2つの例で述べたように、命令は、操作部と番地部に分かれていて、操作部ではどのような処理をするか番地部ではその処理が記憶装置の何番地に関係しているか、あるいは桁数などがきめられる。 ところで、命令の種類とその機能もさまざまで、200種類もの命令をもつものや、数十種類しかもたないものもあり、コンピュータの機種によって異なる。

すでに述べたように、プログラムとは問題の処理手順にしたがって、数十あるいは数百個の命令を順序正しく配列したものである。たとえば、A+B=?という計算をする場合、コンピュータにこのままの形で命令しても計算は行なわない。つぎのように、この計算をもっと細かく分析して命令をつくる。

- ① 記憶装置のa番地に入っているデータAを累算器に移す。
- ② 記憶装置 b 番地のデータ B を累算器の内容に加える。
- ③ 累算器の内容(AとBの和)を記憶装置の任意の番地,たとえばc番地に移す。
- ④ 記憶装置 c 番地の内容を出力装置に出す。

実際には、上述の計算手順の一番まえに、データAとBを入力装置で読んで、記憶装置のa番地とb番地に入れることを指示する命令が必要である。

コンピュータに与えるプログラムは詳細で、しかも完全なものでなければならない。もし誤った手順(プログラム)を教えこまれても、「自分自身で考える力」がないため、そのとおり実行して、自分でそれを訂正するようなことはできない。

「累算器に記憶装置の 1,3 5 0 番地の内容 a を加える」命令を"10 1350"で表わすと仮定すると、この"10 1350"を機械語(マシンランゲージ)とよぶ。このマシンランゲージは、われわれが日常使う言葉と異なるので、プログラミングは面倒で、しかも誤りをおかし易い。

このような問題を解決するために、もっと自然な表現でプログラムを作る方法が開発されている。たとえば、"10 1350"と書く代りに ADD/A とすればよい。しかし、コンピュータはあくまでもマシンランゲージでないと理解できないので、"ADD"を"10"に変換するプログラムを用意しておき、コンピュータ自身に翻訳させなければならない。このような方式をアセンブラ(assembler)

とよぶ。

このほかに、もっと高級な方法としてコンパイラ(compiler)がある。これはアセンブラよりもっと自然な表現、つまり数式や文章にちかい表現でプログラムをつくる方法である。たとえば、 $G=(A\times B+C/D)\times(E-F)$ という算術式を書くだけでGを計算してくれる。この場合も、マシンランゲーシに翻訳するプログラムは当然必要である。コンパイラ言語を使用するとプログラミングが非常に簡単になり、しかも、いろいろなコンピュータに共通に使える。現在、技術計算用としてFORTRANおよびALGOL、データ処理用としてCOBOLがある。

コンピュータそのものをハードウエアとよぶが、このハードウエアを働らかせるプログラムを総称してソフトウエアとよぶ。コンピュータはソフトウエアによって、はじめてすばらしい能力を発揮するもので、今後ともコンピュータの技術はますます進展するであろうが、その進歩と平行して、ソフトウエアの充実が重要であることは言をまたない。

2.3 コンピュータの能力とその限界

コンピュータは与えられた処理手順(プログラム)によって複雑な計算や大量の情報(データ)を迅速正確に処理する。しかも与えられたプログラムの範囲内で、いままで人間の行なってきた定型的な判断をも代行する能力をもっている。コンピュータの特性を要約するとつぎのとおりである。

- ① プログラムや処理に必要な情報を記憶できる。
- ② プログラムにしたがって、定型的な判断業務を含めて、自動的に、しかも正確に情報を処理する。
- ③ 高速である。たとえば、加減算とか2数の大小の比較などは、1秒間に10万~20万回位の速度である。
- ② プログラムできる処理ならば、プログラムを取りかえることによって、どのような業務にも利用できる。

なお、コンピュータの利用に適合する情報の一般的な特質としては、相互作用する変数が多い、妥当な正確性をもった情報、くり返しの多い業務、速度を要求する場合、データ量が多いなどをあげることができる。

コンピュータの機能を正しく理解し、その能力を発揮させれば、データの処理時間が大幅に短縮され、 正確になるとともに、データ処理の多角性が大となり、一つのデータをさまざまに利用できる。さらに、 関連するデータの相関関係など、各種各様の処理が一つのシステム内で可能となり、外部条件に対する 応答性が改善され、フィードバックが速くなる。

このように、コンピュータはすばらしい能力をもっている。しかし、「自分自身で考える機械」になっていない。目的達成のために最も効果的な処理システムを設計し、それにもとずき、正しいデータとプログラムを与えたときに、はじめてすばらしい力を発揮するもので、機械そのものが自分で勝手に理論づけをしたり、独自の判断をしたり、直観的な結論を下したりなど、プログラムされた以上の仕事をすることはできない。処理システムやプログラムおよびデータの良し悪しがコンピュータの利用効果を左右するもので、これらは全て人間の力にかかっている。

3. システム設計のすすめ方(情報処理の効果をあげるために)

3.1 社内の空気づくり

情報システムは特定の作業のみを対象としたものではなく、全社的な問題である。そのために情報処理担当部門のみで計画をすすめるのはむずかしいことである。もし、よい計画ができたとしても関係部

門では新らしいシステムへ移行するために必要な知識もないし、心構えもできていない。このようにしてできた計画を実施に移すには多くの障害がある。このようなことから担当部門は、経営者の理解と関係部門の積極的な協力を得るためのPRと社内教育を行なうとともに、場合によっては関係部門の長による委員会の設置も必要であろう。

3.2 目的をハッキリさせる

そのシステムにはどのような使命と役割があるのか、なぜそのシステムが必要なのかといったポリシーが確立されなければならない。

システム設計に入るためには、設計すべきシステムに与えられている目的を解明するとともに、対象 業務の範囲もはづきりさせておく必要がある。長い時間と貴重な労力をかけても使命をはたしていなければ無意味な仕事をしていることになる。その意味では、この段階は以後のシステム設計のすべての前提となるものである。

一般に目的としては、二つの性格のものが存在しなければならない。つまり、このシステムによって成し遂げなければならない結果の状態と、課せられた達成目標を実現するためにどのような過程を選択すべきかを検討するにじゅうぶんな条件である。

3.3 システムの構成

コンピュータの利用効果はシステム設計の良し悪しにかかっている。もしコンピュータが効果的に活用されていないとすれば、それはシステム設計に原因がある。

システム設計の手順は固定しているものではない。目的、対象範囲のそれぞれの立場や条件によって、 いろいろな手順が考えられる。しかし、これらを通してシステム設計には一つの段階があると考えられ る。

① 現 状 調 査

目標達成に必要な要素、基準となる数量関係、それらの要素相互間の順序関係および処理上の問題点、例外処理、各単位業務間の関係について現状を調査し、対象とする業務の輪郭を明らかにする。

② 基本構想の設定

目標を達成するために必要な諸機能を摘出し、それらの機能相互間の連結状態(図 7)を設定するとともに、機能の発揮を分担する要素(人間一機械)とその場所を指定するとともに合理的な相互間の関連を設定する。さらに、機能、要素、場所などの設定にもとづいて組織体系における位置を指定し人間相互間における責任権限と情報伝達の経過を示す。

③ 詳細 設計

システム設計の第三段階は,前段階で構成された基本構想を,いろいろ変化する各種環境条件の下で,好ましいと評価される方法で作動させるための詳細な計画であり,プログラムにつながるものである。

作成資料の内容とフォームの設計、データの収集方法、投入形式、伝票コードの設計、ファイルの設計、処理方法の明細とそのフローチャートチェックの方法、資料送達方法などを決める。このような明細な設計にもとづいてプログラムが作られる。また、機械処理部分のみでなく手作業部分の業務との関連を考えてその合理化を計る必要がある。

④ 手続書の作成と移行計画

最適のシステムが構成されたとしても、ルール通りにオペレーションが実施されなければ、システム設計の具体的な成果を最大限に発揮させることはできない。そこで、対象業務、作業方法、作業手

順、所要時間などについて指示するためのプロセジュアが必要になる。

プログラム作成が完了すれば、コンピュータへの移行が行なわれるわけであるが、そのためにはファイルの切換え、テスト・ラン、現場の訓練、機械処理に伴なう諸規定類の改定、どういう順序で切換えていくのかタイム・スケジュール、人員配置などを事前に計画しておく。

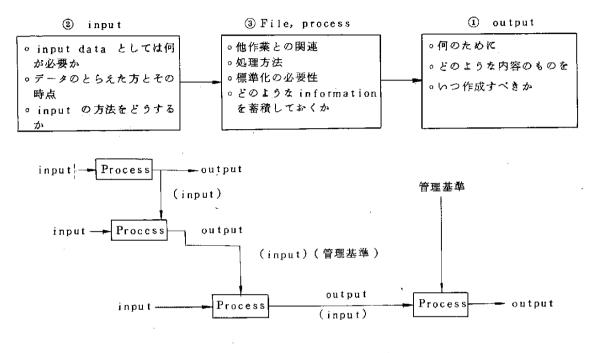


図7 機能相互間の連結

3.4 設備計画

設備計画はコンピュータ、付属機器の選定発注、設置場所、電源とか空調などの設備、什器・備品などその範囲は広いが、最もむずかしいのがコンピュータの機種決定である。

システムを事前に設計することによって、どの程度の大きさのコンピュータでよいか、またその構成をどうすればよいかが決められる。米国の政府では「コンピュータの選定と導入はシステムの明細にも とづいて行なわれなければならない」というポリシーがある。

3.5 1組織と要員計画

これは簡単にいえば、情報処理システムをどのように運営するかの問題で、企業内における情報処理 担当部門の位置づけと推進母体を何にするか。また、コンピュータを中心にした新らしいシステムをう まく管理し、動かしていくためには、要員は何人位必要で、どのような順序で教育をすすめればよいか といったような問題である。

コンピュータの利用効果はシステム・アナリシス,システム・デザイン,プログラミング,オペレーションに左右されるが,これは情報処理要員の質によって決定づけられる。要員の育成を軽視すると,与えられた使命と役割をはたすことは困難であろう。

4. 情報処理システムの方向

4.1 量から質への移行

事務の能率化は、コンピュータの利用によって著しい進歩をみせているが、最初の段階では、基本的な業務活動に包含される作業事務を中心に導入される傾向がある。これは、日常の基本的な業務活動を遂行する上で、大量のデータを速く正確に処理することが要求されており、人手ではとうていさばききれないからである。

このような適用方法は、全体計画をたてることなく、当面する対象業務を個々にとりあげるため、比較的短期間にコンピュータに移行できる。

ところで、経営における情報処理システム本来の目標は、経営の意志決定に寄与するインフォーメーションの提供にあるといわれているが、①そのもとになるデータ処理の合理化なくして意志決定資料の提供はあり得ない。しかし、②大量事務の処理のみにコンピュータ導入の効果を期待するならば、事務処理の高速化、正確性、人員の増加の防止に効果はあらわれるとしても一部の企業を除外して、システム導入の費用に見合った効果を得ることは相当困難であろう。したがって、③経営のための情報処理システムは、大量の事務データ処理を基礎とした量的側面の解決から質への移行を期待している。

量・質両面への情報処理の適用は、情報の発生場所でとらえた正確ななまのデータを材料に、関係する業務の一貫処理を可能にする総合システムの設計が必要である。

4.2 総合処理システム

企業の成長にともなって、情報(データ)の量はますます多くなり、それに加えて、人手中心の事務 処理には多くのむだがある。例えば、各部門で似かよった資料を重複して作るとか、本来は必要でない にもかかわらず、それを作らなければ目的とする結果が得られないといった中間的な帳票がある。また 管理用の資料にしても、その内容についてじゅうぶん検討せず、ただ慣習にしたがって、業務活動の結 果を個々に集計したにすぎないものもある。さらに、その資料が必要なときに間に合わないような場合、 過去にこういうことがあったということを知るにとどまって、それによって適切な処理を講ずることは できない。

「今までの報告書は取引の結果,つまり過去の状態を示すものであったが,これからは過去の結果とともに将来の方向を指し示すものでなければならない」といわれている。これは,経営管理者にとって,将来の方向をどのように決定するかが最大の関心事であることをあらわしている。

企業の規模が大きくなり、企業間の競争が激しくなればなるほど、企業発展のためには科学的な経営が要求される。企業内の事務処理を簡素化し能率を向上させるとともに、経営管理面の効果を増大させるためには、コンピュータの処理能力を背景として、例外管理の考えをとり入れた総合処理システムの確立が望ましい。

One job one record.

From order entry to balance sheet.

Report for exception basis.

コンピュータによる総合処理システムによってつぎのことが可能となる。

- ① データの重複や欠如が避けられる。
- ② 実質的にはデータ量が減少し、中間的な処理が排除され、大幅に事務処理が簡素化される。

- ③ 関連する業務のデータが一括してコンピュータの中に貯蔵されるので多角的な利用が可能となり、 経営管理上必要な情報をそれぞれのデシジョンポイントに適時フィード・バックできる。
- ④ 業務の進行に応じて、つねに計画と実績の比較が行なわれ、状態と傾向、目標に対する達成状況が明確になり、割当てられた責任が効果的に実行されているかどうかの監視とコントロール、さらに時機に応じたダイナミックな計画が可能となる。
- ⑤ 例外管理の導入が容易となる。

例外管理というのは、計画値と実際活動の情報を対比することによって、きめられた基準から外れたもの、例えば、資材の最低在庫基準を割って発注を要するもの、販売達成率の低いもの、その他計画面、実施面になんらかの手をうつ必要のある事項のみを報告する方法である。この例外管理の方法を採用すれば、経営者とか管理者は、貴重な時期の大部分を問題解決のために使うことができ、問題をさがしたすのに時間を費さなくて済むようになり、定形的な業務から解放され、より重要な管理業務に専念できる。

しかし、このようなシステムを有効に運用するには、業務活動の情報を迅速、正確に収集するだけでなく、計画そのものが信頼できるもので、しかもup to date でなければならない。

4.3 総合処理システムの一例

商品販売活動に例をとると、毎日の受注、販売、発注、仕入など販売活動に関連するデータを、その 発生場所から直接収集してコンピュータに投入する。もちろん販売活動から発生した原始データのみで なく、販売計画高とか商品の在庫基準、仕入基準などを与えることによって、それぞれのデータが多角 的に生かされ、さまざまな相関(状態と傾向、目標に対する達成状況)などをみることができる(図8)。

- ① 受注, 販売, 発注, 仕入, 受注残, 発注残, 在庫など実績の迅速正確な把握。
- ② 販売実績にもとづき,必要な時点で自動的に請求書の発行が可能。
- ③ 販売(出荷),受注残,在庫量,在庫基準,仕入基準の総合により,適確な出荷,発注指示。
- ④ 売掛金の回収情報との関連で、売掛金の回収促進および売掛金の管理。
- ⑤ 実績にもとづき、経理情報、資金情報の提供が可能。
- ⑥ 販売実績の分析,販売計画との相関により迅速で適切な販売の管理。
- ⑦ 業務活動の実績と外部情報により、時機に応じたダイナミックな販売計画が可能。

以上に列記したように、総合処理システムは、受注、販売事務、発注、仕入事務、在庫管理、請求書発行事務、売掛金管理、販売、仕入関係の経理などが一貫処理され、さらに基準から外れたもの、つまり管理のポイントがタイムリーに把握できるため、偏在在庫とか品切れによる機会損失を未然に防ぐことができる。

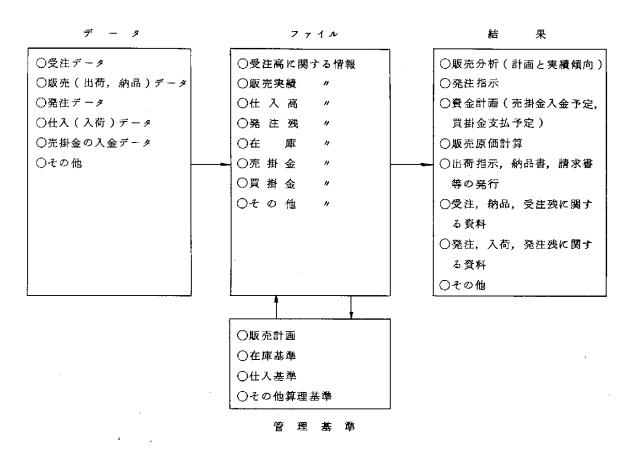


図8 販売におけるデータ処理システム (データベース)

5. 情報処理方式の形態

コンピュータを用いての情報処理は、一括処理(batch process), 実時間処理(real time process) および時分割(time sharing) 方式の三つに大別できる(表 5, 図 9)。

表 5 処理方式とコンピュータの組織

コンピュータ の組織 処理方式	オフライン方式	オンライン方式
バッチ処理	バーッーチ (ローカル・バッチともいう)	リモート・バッチ (オンライン・バッチともいう)
	hard a second se	タイムシェアリング
リアルタイム処 理	オフライン方式でもリアルタイム処理 は可能(インライン処理という)。入 出力タイプライタとかディスプレイ装 置が使用される。	オンライン・リアル タイム

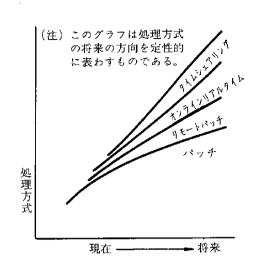


図9 処理方式の動向

5.1 バッチ処理

一括処理(バッチ処理)は、種々の活動から発生したデータ、たとえば、販売データ、資材の入出庫 データ、実験データ、経理データなどを日単位、週単位あるいは月単位などでまとめて処理する方式で ある。現在この方式はコンピュータの使い方の主流を占めている。

バッチ処理におけるインプット・データの収集は、原始帳票(伝票など)によって行なうのが普通である。しかし、データ収集の範囲が広域にわたる場合は、通信回線を介して、データを直接インプットしたり、処理結果をアウトプットすることが可能である。遠隔地から通信回線を介してコンピュータを利用する方法をオンライン方式(on line system) とよび、オンラインでバッチ処理を行なうことをリモート・バッチ処理(remote batch process)という。

なお、オンライン方式でないものをオフライン方式といい、オフラインでのバッチ処理(通常のバッチ処理)をローカル・バッチ処理という。

5.2 リアルタイム処理

ところで、データを週単位とか月単位でまとめて処理したのでは、その結果が必要なときに間に合わないで、過去の状況を知るにとどまり、それによって適切な処置ができないこともある。たとえば、資材の入出庫とか在庫の状況を把握する処理が月単位で行なわれ、月1回報告があるとすれば、その間の資材の動きがつかめず、在庫切れがあっても、あとでその事実を知るにとどまり、資材切れによる損失を取り返すことはできない。

このような業務においては、資材の入出庫があるたびに、発生するデータをインプットし、いつでも必要に応じて現状をアウトプットする方式が必要である。この方式を実時間処理(リアルタイム処理)といい、身近かなシステムとしては、まえに述べた国鉄の緑の窓口、航空機の座席予約などがある。

リアルタイム処理は情報処理の結果が、直ちに物理的活動に役立つように、物理的過程と並行した処理システムで、応答に即時性が要求される。リアルタイム・システムの本質は、処理の結果が外部の活動過程に十分間に合う時間内に得られることであるが、要求される応答(フィードバック)の速度は対象業務によって異なる。通常の場合、リアルタイム・システムではオンライン方式が採用されるので、

オンライン・リアルタイム・システムともよぶ。

5.3 タイムシェアリング・システム

オンライン方式とリアルタイム処理の技術はコンピュータの利用面に革新をもたらしたが、さらにとの二つの技術を基礎にして、第2の革新が進みつつある。コンピュータに数十から数百台の端末機(通信回線を介してコンピュータに結合する入出力装置)を接続し、同時に大勢の人々がオンラインで1台のコンピュータを思い思いの仕事に使用する方法で、これが時分割方式(タイムシェアリング・システム、略してTSSという)である。

タイムシェアリング・システムでは、1人の仕事が全部終ってからつぎの仕事にかかるのではなく、 複数の仕事をそれぞれごく短い時間に分割し、継続的に並行して処理していく方式である。各人は、コ ンピュータをあたかも独占しているかのように使うことができる。

端末機から通信回線を介して送りこまれたデータは、それぞれの目的に応じてリアルタイムあるいは バッチ方式で処理されるので、この点ではオンラインリアルタイムとかリモート・バッチ方式と同じで ある。しかし、タイムシェアリングの場合、利用者が独自の目的をもったオペレーションを自由かつ随 時に行なうことが許されるので、コンピュータを共同で利用できるという大きな特徴がある。

コンピュータは中央処理装置と周辺装置の組合わせによって構成されている(図10)。コンピュータの技術進歩をみると、各装置の高速化は当然のことであるが、とくに、利用者の要求に適合できるような各種の周辺装置の開発に力が注がれている。さらに、コンピュータと通信との結合によるオンライン方式の出現によって、遠隔地からコンピュータを利用することが可能となり、人間とコンピュータが自由に能率よく情報のやりとりができる端末機の開発と相まって、情報処理本来の姿である。必要なときに、必要な情報を得ることができるリアルタイム処理が普及するようになった。

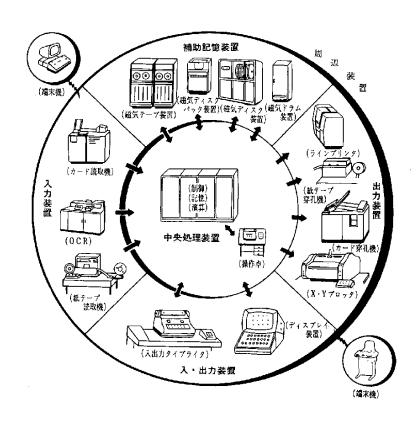
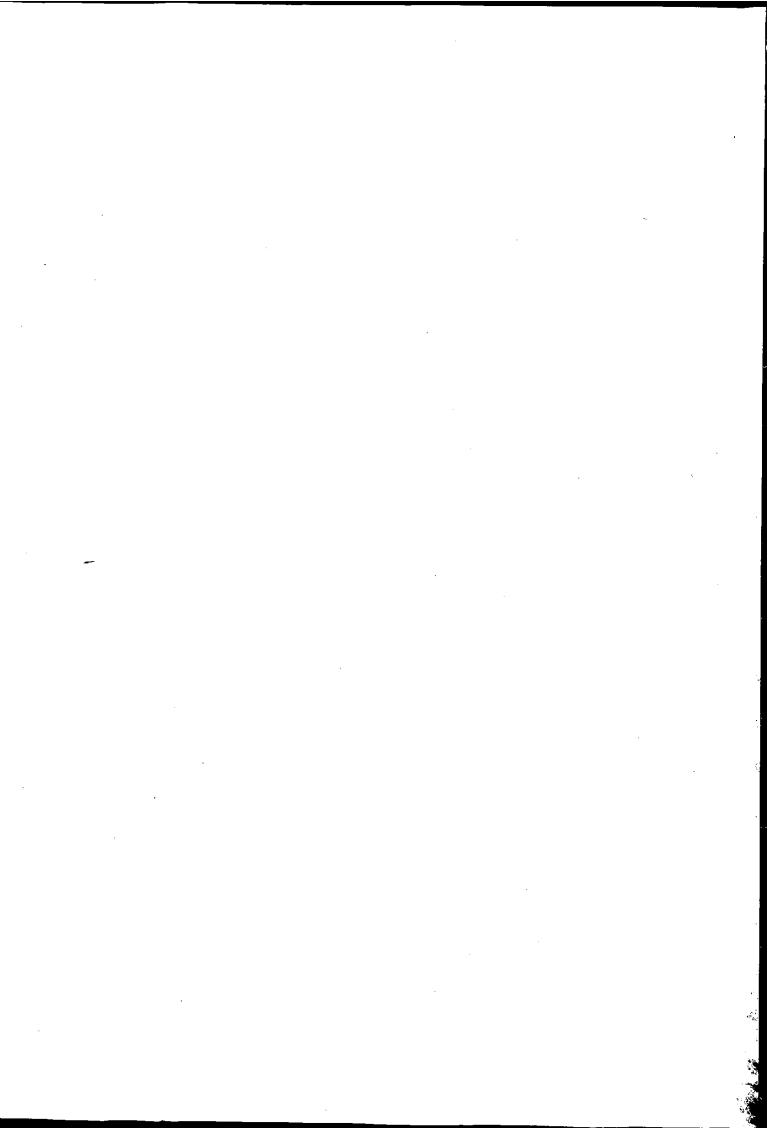


図 1 0 中央処理装置と周辺装置

このように、現在のコンピュータは、利用目的に合わせて各種の機器を組合わせることができるので、 広範囲の仕事に適用できる汎用性をもち、しかも1台のコンピュータでバッチ処理、リモート・バッチ 処理、リアルタイム処理など、あらゆる処理方式が併行できる。

	·			

電子計算機とプログラミング



電子計算機とプログラミング

電子計算機に近づく方法は三つある。その一つは計算機がどんな方面に、どう使われているかというアプリケーションの面から計算機を理解する方法、もう一つは計算機はどういう仕組みで働くのだろうか、加減乗除の方法は、記憶の原理は、ラインプリンターのメカニズムは、などというハードウエアの機構を理解する方法 そして最後は具体的に計算機にある仕事をさせるための動かし方、つまりプログラミングとはどういうものかを理解する方法である。

どんな近づき方をするかは、その目的にもよるが、もっとも手つとり早く計算機を身近なものにする方法は、 プログラミングから入るのがよいのではないかと思う。そしてできることなら、どんな簡単なものでもよいか ら、自分で作ったプログラムで計算機を動かしてみること、始めて計算機が自分の要求通りに働いて、ちゃん とした結果を出した時の喜びは、何ともいえないものである。

さて、電子計算機の働きが世間に宜伝されてすでに久しい。しかし、どちらかというと、計算機はこんなこともできる、あんなこともできる、という華やかな表面の効果のみが宜伝されて、そういう仕事をさせるには、どんなに大変な前準備や、舞台裏の仕事があったかということが、ついなおざりにされがちである。その陰の仕事の重要な部分を占めるものがプログラミングである。電子計算機は自発的意志を持たぬ単なる機械にすぎないから、これに何か仕事をさせようとすれば、予めその仕事の手順を、こと細かに人間が組立てて与えてやらねばならない。その手順書がプログラム(program)で、プログラムを作ることをプログラミング(programming)という。

1. プログラミングとは

人間が計算機に与えるこの手順書の内容というものは、実はいくつかの命令が順序よく並べられたものである。

計算機はそれぞれ独自の命令というものを幾つづつかもっている。その命令の数や機能は、計算機毎にそれぞれ違うのが普通で30種類しか命令を持っていないのもあるし、中には300種類以上もの命令をもつものもあるが、表11にあげたような基本的なものは、大低の計算機が持っている命令だと思って差支えない。

	32 2.4		
命令の種類	機能の説明	命令の種類	機能の説明
Load	記憶装置から演算装置に移す	Store	記憶する
Addition	加える	Read	読む
Subtract	링 〈	Write	書く
Multiply	かける	Jump	とぶ
Divide	割る	Jump Plus (minus)	+(-)ならとぶ
		Stop	とまる

表 1.1 代表的な命令

さて、これらの命令を、どう使うかという話をするには、先づ一通り、計算機の構成というものを簡単に 説明しておいたほうがよさそうである。

1.1 計算機の五つの装置

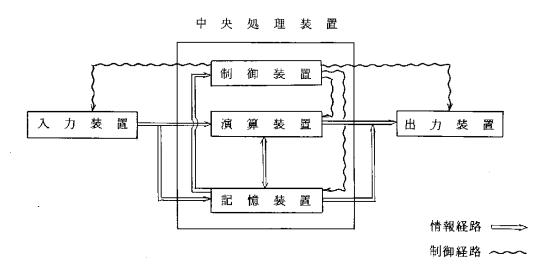


図 1.1 電子計算機の構成

計算機は、図1.1 に示すような五つの装置、入力装置、出力装置、演算装置、記憶装置、制御装置を持っている。世の中には、いろいろな規模の計算機がある。一般に大型計算機、中型計算機、小型計算機、などと区別され、最近はさらに、超大型とか超小型とかいう分類まで表われたが、その規模の大小にかかわらず、電子計算機と名がつけば、どんなものでもこの五つの装置を必ず持っている。大型は大型なりに、それぞれの装置の機能が大きいし、小型はそれなりの機能しか持っていないということである。

先づ入力装置から説明しよう。

(1) 入 力 装 置

計算機に何か仕事をさせるには、そのプログラムなりデータなりを、何らかの方法で計算機の中に読み込ませなければならない。その入口の役目をするのが入力装置である。もっとも一般的な形は、カードや紙テープに穴をあけていろいろな文字を表現し、光を当ててその穴の位置から文字の種類を判別させる方法である。現在の計算機で取扱える代表的な文字は表 1.2 に示すようなものである。

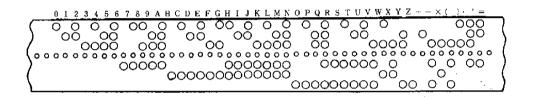
・表 1.2 電子計算機で扱う文字,記号

0 1 2 3 4 5 6 7 8 9

ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz +-*/()・,;=¥@#`':% □ イロハニホヘトチリヌルヲワカヨタレソツネナラムウノオ クヤマケフコエテアサキユメミシヒモセスン。 計算機によっては、このうち小文字はだめだとか、カナ文字は受けつけないとか、それぞれ制限のあるものもあるが、いづれにせよ機械毎にこれらの文字を内部的に何等かのコードで表現し、何の文字であるかを識別して適当な処理を行うことができるようになっている。

これらの文字の一部を紙テープおよびカードにパンチした例を図 1.2 に示す。この様にパンチされたものを入出装置から読むわけだが、残念ながらこれ等の穴の明き工合は、必ずしも統一されているわけではなく機種毎に違いがある。特に紙テーブに関しては 1 行に 8 個の穴のあく 8 単位テープと図 1.2 に示すような 6 個の穴のあく 6 単位テープとがあり、カード以上に文字毎の穴の位置は不統一である。

図 1.2 カードと紙テープ穿孔例



01934567896PCTEFGPIJKLMNDPCRSTUVWXYZ+-*/().;=

カードを読むものをカード読取機(Card reader), 紙テープを読むものを紙テープ読取機(Paper tape reader) という。

紙テープもカードも、入力装置ではふつう光を当ててこれを読む。穴のあいたところは光が通り、 あいていないところは光が通らないという原理で、穴の位置から文字を判断しながら読む。

ところで最近は紙に書いたものをこのようにもう一度パンチしなおすことをやめて、人間の手書きや、特殊インクによる文字がそのまま入力できるような入力装置も大いに研究され、現在でも、OMR(Optical mark reader) とか、OCR(Optical character reader) などが実用化されている。

(2) 出力装置

入力装置に対して出力装置がある。計算機が如何に立派に計算をしてくれても、その結果を何ら

かの方法で人間にわかる形で表示してくれなければ困る。もっとも一般的な形は、プリンターにプリントすることである。プリントされる文字の種類は、入力装置でとりあげたものと同じである。 数値は当然のこと、表題を英文でプリントさせたり、給料明細書に片仮名で人の名前をプリントさせたりすることが、プログラムのコントロールによってできるわけである。

プリンターの他に、入力装置のところで説明したカードや紙テープに穴をパンチして出すこともある。これは一旦カードやテープの形にして出した出力データを、いつかまた、入力データとしてそのまま使おうという目的のためである。このほか出力装置には、プロッターで図形を書かせるもの、ブラウン管を使って文字やグラフを表示するディスプレイ(display)などが使われている。

(3) 演 算 装 置

コンピュータ(Computer)を電子計算機と訳したのは失敗だという人がいる。電子計算機というと、やはり計算をする機械というイメージが強い。勿論コンピュータは計算もするが本来の目的はもつと広く、もろもろのデータや情報を処理する機械であるから、データ処理装置とか、情報処理装置とか訳すべきであるという。しかし今更、情報処理装置と名前を変えてみても混乱するだけであるから電子計算機のままで致し方ないとして、矢張り計算をするという機能は重要であるということには変りない。

さて、この計算の舞台が演算装置である。電子計算機が扱う計算はいわゆる四則演算、たす、ひ く、かける、わる、である。

多くの計算機は、演算レジスター(累算器 Accumulatorとも呼ぶ)を持ち、そこにデータを 持ってきて演算を行い、結果もそこにでき上るようになっているが、中には、演算レジスターが表 面にあらわれず、記憶装置内のある場所のデータと他の場所のデータとの演算結果が、そのどちら かの場所に入る様なタイプのものもある。

(4) 記 憶 装 置

電子計算機の大きな特長の一つはこの記憶装置である。プログラムやデータが入力装置からひとまず記憶装置に記憶され、それからすべての仕事が始まる。

計算機に使われている最近の記憶装置の種類はいろいろある。しかし大きくわけると、主記憶装置 (main storage)または内部記憶装置と、補助記憶装置 (auxiliary storage) または外部記憶装置とがある。

主記憶装置には、現在直接演算の対象となるデータや、プログラムが入り、主記憶装置に入りきれないもの、または今すぐには使わないデータやプログラムなどが、一時、補助記憶装置に貯えられ、必要に応じて主記憶装置にとり出されて利用される。したがって記憶装置間の内容のやりとりは、プログラムのコントロールによって自由自在にできなければならない。

主記憶装置の中身は多くの部屋に分かれている。一つ一つの部屋には番地(address)がつき, 0番地,1番地,2番地…………… とつづく。何番地まであるか,一つの番地の中はどの位の 広さがあるかは計算機によって異なるが,沢山の部屋がある方が勿論有利であるから,これが計算 機の大型,中型などの規模の一つの目安となっている。

先づ部屋の広さの方からいくと, 一つの番地に

- 1. 1桁の数字, 文字が入るもの ………桁 (character)を単位とする。
- 2. 2桁の数字か、1字の文字が入るもの……バイト(byte)を単位とする。
- 3. 数桁がまとめて入るもの ……………… 語 (word)を単位とする。

などの種類がある。

それぞれ、キャラクター・マシン (character machine), バイト・マシーン (byte machine), ワード・マシーン (word machine)などと呼ぶこともある。

ワード・マシーンでは1語の大きさは、10進法の6桁ないし12桁位の数値の入るものが多い。 記憶装置の大きさを記憶容量というが、例えば、0番地から9999番地までの番地をもったワード・マシーンだとすれば、記憶容量は1万語であるという。最近は1000を単位として1万語のことを10K語ともいう。キャラクター・マシーンなら10Kキャラクター、バイト・マシーンなら10Kバイトである。

補助記憶装置としては、磁気デスク、磁気カード、磁気テープなどが使われる。補助記憶は主記憶に比べると、容量は大きいが、内容をとり出す時間、つまり待時間(access time)がかかる。磁気ドラムは高速で回転する円筒の表面に磁性体をぬったもので、待ち時間の早い比較的小型のものを主記憶装置としても使われる。

磁気デスクは、表面に磁性体をぬった円盤が高速に回転しており、磁気ドラムと同様、磁気ヘッドによって、磁性体表面に情報の書き込み、読み出しが行われる。これらにも矢張り番地に相等する区画番号がつけられており、主記憶との情報伝送の単位となる。

磁気テープは、録音テープと類似のもので、音声のかわりに計算機コードが記憶される。テープ であるから順おくりにしか情報がとり出せないが、価格の割安の事、保管に便利なことなどの利点 があり、特に入出力装置の補助としても現在非常に多く使われている。

磁気カードは、磁気テープを短冊型に切ったようなもので(巾はかなり広い)、テープの待ち時間の短縮をねらったものである。

ところで、計算機の中ではすべての情報は0と 1,つまり2進法で表現される。0から9までの

- 10進数を2進法で表現した例を表 1.3 に示す。
 - 2 進法の1桁を1ビット(bit) と呼び,
- 10進数値は4ビットで表現される。

キャラクター・マシーンの1桁は6ビット,バイト・マシーンの1バイトは8ビットがふつうであり,ワード・マシーンは,1ワード24ビット乃至48ビット位が多い。2進と10進の桁数の換算は,それぞれn桁,m桁とすると

 $m = n \log_{10} 2$

である。例えば2進法40ビットは10進法でほぼ12桁,32ビットはほぼ9桁である。

表 1.3 2 進法による数値の表現

10進数	2 進数
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1001
	2 ³ 2 ² 2 ¹ 2 ⁰

(5) 制 御 装 置

制御装置は、記憶装置に格納されているプログラムの命令を、一つづつとり出しては、解読し、他の四つの装置に、それぞれの指令をあたえる重要な役割りの装置である。とり出すべきデータや命令の番地を示す番地レジスタ(address register) や、解読中の命令が一時的におかれている命令レジスター(instruction register)、命令の番地に、ある常数を加えるインデッ

クス・レジスター(index register),などを持っている。

1.2 命令とその表現法

表 1.4 の 2 重ワクの中は, 4 命令からなる, ごく簡単なプログラムである。

30 100は記憶装置100番地の 内容を演算レジスタ(以後略してARという)に持って来るロード(LOAD) 命令である。説明文中(n)はnの内容 を意味する。次の20 101はアッド

表 1.4

操作部	番地部	説	明
3 0	100	(1 0 0) →A R	
2 0	101	(AR)+(10	1) → A R
2 1	102	(AR)-(10	2) →A R
11	200	$(AR) \rightarrow 200$	

(ADD)命令で,その時のARの内容に101番地の内容を加える。21-102はサブトラクト(SUB)命令で,ARの内容から102番地の内容を引く。最後の11-200は,ARの計算結果を200番地にストア(STORE) する命令である。つまりこれは(100)+(101)ー(102)の演算を行い,結果を200番地に入れるプログラムである。このように一つの命令は, $30(L\overline{O}\Lambda D)$,20(ADD),21(SUB), $11(ST\overline{O}R)$ などの様に,その機能を表わす操作部と,その操作の対象となるものが,どこにあるかという記憶装置の番地を示す番地部との2つの部分から成り立っている。

ここで問題になるのは操作部の表現法である。表 1.5 に示すように、機種によって命令の表現法がことなり、その上、表現と内容とが全く無関係であるから、憶えにくいばかりでなく、間違いもおこし易いので、最近は第4列目の様に、ある程度その機能を表現できる記号命令を使うのが普通になってきた。また番地部に関しても、100 とか 200 とか実際の番地を直接表示せず DATA とか KEISU とか 記憶場所に、適当な名前をつけて使う方が、プログラムがわかり易く、かつ間違いも少なくなる。そういった表現で表 1.4 のプログラムを書き直してみると表 1.6 の様になる。

表 1.5 命令のさまざまな表現法

機能	機種A	機種B	機種C	記号命令
Load	3 0	0 3	500	LOAD
Add	2 0	0 2	400	ADD
Sub	2 1	0 4	402	SUB
Mult	3 2	12	200	MULT
Divide	1 7	16	220	DIV
Store	11	3 8	601	STOR
Write	61	8 2	640	WRITE
Read	6.6	3 7	540	READ

表 1.6

LOAD	A
ADD	В
SUB	C
STOR	RESULT

2. 簡単なプログラム例

「カードにパンチされているデータを一枚づつ読んで,その中の最少のものをプリントせよ。なおデータ の中に 0 はないものとし、データの尽きた印として最後のカードには 0 がパンチされているものとする」。

まず処理の手順を図(図 2.1)に示そう。これをフローチャートと呼び、プログラムの処理手順を表現す るには大変有効な方法である。先ずこのフローチャートを説明しよう。

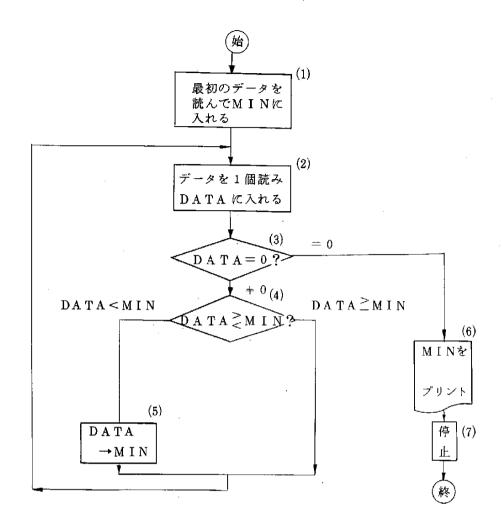
- (1) 最初のデータを読み込んで、とりあえずそれまでの最小値とみなし、MIN(minimum の意)と名 付けた場所にしまっておく。
- (2) 次のデータを読み込んでDATAに入れる。
- (3) 今読んだものが 0 ならもうデータは終ったわけで最後のプリントの仕事にゆく。
- (4) 今読んだものと、すでにMINに入っているものとくらべてみる。MINの中身の方が小さければ次 のデータを読む(2)へ戻る。
- (5) 今読んだものの方が小さければ、MINの中身をいれかえる。そして次のデータを読む(2)へ戻る。
- (6) MINをプリントする。
- (7) 停 止 このプログラムは表 2.1 のようになる。

表 2.1 例題のプログラム

READ	MIN		(1)	最初のカード上のデータを
$R \to A D$	DATA		(2)	次のカードを読みデータを
$L\overline{O}$ A D	DATA		(9)	データが 0 ならL 1へとぶ
J Z	L 1		(0)) > p-0/4-9 H 1 \2 x
SUB	MIN		(4)	DATA-MIN の計算を
J P	L 2		(4)	その結果が土ならL2へと
$L\overline{O}AD$	DATA	_	(E)	DATA の内容をMINV
STOR	MIN		(6)	DAIA ONA WIIN
J	L 2			L 2へとべ
WRITE	MIN		(6)	MINの内容をプリントで
на L Т			(7)	停止
START				このプログラムの最初かり
	READ LOAD JZ SUB JP LOAD STOR J WRITE HALT	READ DATA LOAD DATA JZ L1 SUB MIN JP L2 LOAD DATA STOR MIN J L2 WRITE MIN HALT	READ DATA LOAD DATA JZ L1 SUB MIN JP L2 LOAD DATA STOR MIN J L2 WRITE MIN HALT	READ DATA (2) LOAD DATA (3) JZ L1 (4) SUB MIN (4) JP L2 (5) LOAD DATA (5) STOR MIN (6) HALT (7)

- をMINに入れる。
- をDATA に入れる。
- ぶ。0でなければ次の命令へ
- とび、+でなければ次の命令へ
- に入れかえる。
- する。
- ら実行を開始する。

図 2.1 例題のフローチャート



READ N はカードから読みこんだデータを N に入れる命令で、最後のWRITE MIN は MINを プリンターにプリントする命令である。

JZ はJump Zero の略でARの内容、ここではDATAが 0 の場合は、番地部を示すL1 にとび、ゼロでなければ次の命令に移るという条件つき飛越命令である。

同じようにJPはJump Plus の略でARの内容が十ならば、つまりこの場合は、DATA-MINの計算結果が十ならば、番地部で示すL2へとび、十でなければ次の命令に移るという命令である。LlおよびL2はラベル(Label)と呼び、立札の役目をする。この例のように、いくつかの命令をとび越えたり、戻ったりする場合に必要な立て札である。JPやJZの番地部は、今までと異なり、記憶装置の番地ではなく、行く先を示している事に注意しなければならない。HALTは停止命令である。最後のSTARTはこのプログラムの実行を指令する。

一般にプログラムそのものも、いろいろなデータと同じように記憶装置に記憶されるので、このプログラムがカードやテープにパンチされて、入力装置から読み込まれ、記憶装置にすべて一旦記憶されてから START の指令で実行が開始される事となる。一つの命令を1ステップと呼び、ステップ数によって、プログラムの規模の目安がつく事になる。

3. プログラミング言語

表 1.4 のようなプログラムを機械語,またはマシーン・ラングェージ(machine language)と言い,表 1.6 のように記号命令や記号番地を使ったプログラムを記号言語,またはシンボリック・ラングェージ(Symbolic language)と呼ぶ。この両者を比較すると後者の方がわかり易く,間違いも少ないが,どちらも命令という形に分解しなければならないという点では大差はない。そこで,命令という考え方を無視して自然語に近い表現や,算術式の形のままでプログラムが書ければもっと便利であろうという事になる。例えば,表 1.6 のプログラムの代りに,

RESULT = A + B - C

と書けばよい。また、プログラムは算術式ばかりかというわけにはゆかないから、

IF DATA ≥ MIN THEN GO TO LABEL, ELSE MOVE DATA TO MIN; という様に、日常の英文に近い表現が許される。こういった形でプログラムを作る事を自動プログラミン グ(automatic programming)という。

さて以上をまとめてみると、プログラミングには、

機械語 (machine language)

記号言語 (Symbolic language)

自動言語 (Automatic language)

の3通りの言語があることがわかった。

ところで困ったことに計算機そのものはやはり機械語しか理解できない。したがって、元はどんな言語で書いてあっても、それが計算機の中で実行される時点では、必ず機械語になおされていなければならない。即ちいつかそれを機械語に翻訳しなければならないのである。そしてその役目を果たすのが翻訳のプログラムである。記号言語のプログラムを機械語に翻訳するものをアセンブラー(assemblor)、自動言語のものを機械語に翻訳するプログラムをである。この翻訳プログラムをつンバイラー(Compiler)と呼ぶ。この翻訳プログラムの名称から、最近は記号言語のかわりにアセンブラ言語、自動言語のかわりにコンパイラ言語という呼び名が専ら使われている。これらの翻訳プログラムを作る仕事というのは、なかなか大変なもので、とくに大規模なコンパイラーになると数万ステップにもなる場合がある。その上、メモリーの容量や、処理速度という点でもそれ相応の要求が出て来ることになる。

4. 国際共通語

自然語に近いプログラミング言語の出現によって、プログラミングの作業が、かなり容易になったと同時 にもう一つの大きな利点は国際共通語への道がひらけることである。

計算機が変わればプログラムも全部変るという事は大変厄介なことである。プログラムを通しての技術の交流という点でも大きなマイナスである。この原因は計算機の機能が夫々異なること、特に命令というものが全くまちまちであるからであるが、それなら世界中の計算機の命令を全部統一してしまえばよいのではないかという意見も出るだろうが、現在まだまだ、あらゆる面で発展途上にある電子計算機に対して命令体系を固定化してしまうということは大きな技術的障害となることは明らかである。そこで内部的ではなく、外部的に、言いかえればハードウェア的にではなく、ソフトウェア的にプログラム言語の方を統一しようという事になった。それも機械語や記号言語のレベルでは無理で、自動言語のレベルの言葉を共通なものとし、個々の機械語に翻訳する翻訳プログラム即ちコンパイラーを計算機毎に大々作ればよいということになった。

現在国際的に共通語となっているものがいくつかあって、そのうち技術計算用には、ALGOL(Algorithmic Language)、FORTRAN(Formula Translator)、事務計算用には COBOL(Common Business Oriented Language)があるが、 夫々の言語について例題を中心に簡単に説明しておこう。

4.1 A L G O L

この言語はヨーロッパを中心とした学者グループによって提案され1958年,スイスのチューリッヒで開かれた国際会議で最初の版がきめられてから,1960年に入出力とある一部の細かい点を除いて殆ど仕様がきまったのでALGOL60 と呼ばれる様になった。その後,さらに1965年に入出力仕様も決定された。

ALGOLの目的は勿論この言語で書いたプログラムが、計算機ごとに作られている ALGOLコンパイラーによって計算機に直接受けつけられているということもさることながら、そもそもこの言語が作られた動機は、プログラム手順や解析のアルゴリズムをこの言語で公表するためのPublication language としての目的があったと言われる。その意味からして同じ技術用のFORTAN に比べるとより文章的な部分が多いようである。

簡単な例題を示そう。

「例題 1. 100個のデータを読んで、その中の最小のものをプリントせよ」。

アンダーラインのあるものは ALGOL できめられた単語である。; で区切られた一つずつの言葉や算式をステートメント(statement)と呼びプログラムの一単位とする。end の直前のステートメンには; がいらない。ALGOL では一区切りのプログラムをブロック(block)と呼び、begin とend でくくる。普通のプログラムはいくつかのブロックからなり立つが、この例題のように簡単なものは1つのブロックでプログラムが完結している。

real A、MIN; integer N;はこのプログラムの中にはA、MIN, Nという名の変数を使っていることを宜言しており、ALGOLではプログラムの冒頭に以下で使うすべての変数の名前をこのような形で宜言しなければならない。このうちinーtegerとは0、1、2、………等の整数のことで、realとは整数以外の実数で、例えば1.5、

-0,00328,0.476×10¹⁰などは すべてrealである。先ず最初に読

```
begin real A, MIN;
    integer N;
    inreal (A);
    MIN:=A; N:=1;
    L1: inreal (W);
    if MIN>A then MIN:=A;
    if N=99 then go to PRINT;
    N:=N+1; go to L1;
PRINT: outreal (MIN)
end
```

んだAをMIN (minimun 最小の意)とし、同時に個数 Nを1とする。これまでは準備である。 MIN:=Aは $A \rightarrow MIN$ といったような意味でN:=1も $1 \rightarrow N$ 即ち、Nを1とする意味である。次のL1:は ν -ベル(label)と呼び、立て札の役目をする。L1の ν ベルのついた inreal(A)のステートメントから go to L1; の間がg9回繰返される。

if MIN>A then MIN:=A;は殆んど文章の通りでMIN>A ならAが新しいMINにな

るわけであるからMIN:=A が行われ,そうでない時は次のステートメントへ行く,次のif ステートメントはNが 9 9 Kなったか否かをしらべるもので 9 9 Kなればもうデータは尽きたわけであるから go to PRINT; vertional vertional vertion <math>vertional vertional vertion <math>vertional vertion of the contract of the contract of <math>vertional vertion of <math>vertional vertional vertion of <math>vertional vertion of vertion of <math>vertional vertion of <math>vertional vertion of <math>vertional vertional vertion of <math>vertional vertion of <math>vertional vertional vertional vertion of <math>vertional vertional vertion of <math>vertional vertional vertion of <math>vertional vertional vertion of <math>vertional vertional vertional vertion of <math>vertional vertional vertional vertional vertion of <math>vertional vertional vertional vertion of <math>vertional vertional vertional vertion of <math>vertional vertional vertional vertional vertion of <math>vertional vertional vertional

「例題 2. A, B 夫々 2 0 0 個ず つのデータが記憶されている。夫々 対応する順序に掛け合せ, 2 乗した 和を作れ。」

この例はSUM= $\sum_{i=1}^{200}$ (AiBi)²

を計算せよということである。

real array A, B [1:200];

begin real SUM;
 real array A, B {1:200};
 integer I;
 SUM=0;
 for I:=1 step 1 until 200 do
 SUM:=SUM+(A (I) ×B (I)) † 2

という宜言はAとBがともに realで夫々1から200までの添字をもった200個のグループになっていることを示す。仮にAが10ケ,Bが50ケなら real array A[1:10],B[1:50];と別々に書けばよいし,マトリックスのように2次元のものは例えば30×40ならA[1:30,1:40]又,3次元で5×10×20ならA[1:5,1:10,1:20]のように表現すればよい。また添字の下限は必ずしも1でなくてもA[-10:10]という表現もゆるされる。for I:=1 step 1 until 200 do もほぼ文章通りの意味で,Iが1から200まで1おきに以下のステートメント,ここではSUM:=SUN+(A[I]×B[I])↑2を実行せよということである。()↑2は()²のことである。

4.2 FORTRAN

FORTRANとはFormula translatorの略といわれ、1956年 IBM 704のために開発されたプログラム言語であるが、その後 IBM の他の機種にも採用され、IBM 系以外の計算機にも広く利用されるようになった。

ここではALGOL との比較の意味も含めて同じ例題で説明しよう。

「例題1 100個のデータを読んで,その中の最小のものをプリントせよ。」

FORTAN では単一変数(Simple variable)について原則として宜言を必要としない。変数名が I, J, K, L, M, Nのいずれかから始まるものは integer,他は real ということになっている。ただし特にREALとか INTEGERとか, typeを宜言すれば頭文字にこだわらずに使うこともできる。この例では特に宜言をしないで, A, XMINは共に real, Nは integer である。

IFステートメントは,

IF(論理式)ステートメント1

XMIN=A

N=1

10 READ(u, 20)A

IF(XMIN.GT.A)XMIN=A

IF(N.GE.99)GOTO11

N=N+1

GO TO 10

11 WRITE(u, 30)XMIN

STOP

20 FORMAT(F10.3)

READ (u, 20) A

30 FORMAT (F15.3)

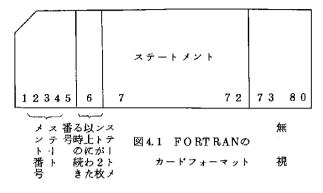
END

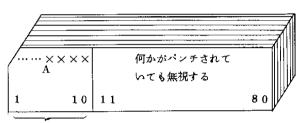
という形をとり、()内の論理式が成り立てば、右のステートメント1を実行し、成り立たなければ、ステートメント1を実行せずに次へ行く。

XMIN,GT,AはXMIN>Aの意味であり、N.GE,99 はN≥99の意味である。

左側の10,11等の数字はステートメント番号(statement number)と呼ぶレーベルで、普通5桁以内の整数である。元来FORTANのプログラムはカードから読むのが普通で、ステートメントのカード上のコラム(column)の指定は図4.1のようにきめられている。

FORTRAN の入出力は一般的に READ(u,n)入力リスト WRITE(u,n)出力リスト という形をとり、uは入出力の機器の指定、 nは入出力データーのフォーマット(FO-RMAT) 指定のステートメント番号である。 ここでは仮に、入力はカード、出力はプリ ンターとすると、先ず入力は、





10 col 全部に パンチされていな 図4.2 例題1のデータカード くてもよい

という形で、これは図 4.2 に示すように、A という変数がカードの1 colから 1.0 col までに実数値としてパンチされ、小数以下は 3 桁であることを示している。

次に出力のWRITE ステートメントのフォーマット,

「例題 2. A, B夫々 2 0 0 個ずつ のデータが記憶されている時 S U M

FORTRANはarray に限り宜言が必要である。DIMENSION ステートメントをそれに使う。

この例ではA, Bとも 1 次元の 2 0 0 個ずつの array であることを 宜言している。Cが 5×1 0 0 2 次元なら C

(5,10)というような形で書く。FORTRAN は原則として3次元までゆるされる。DOステートメントは一般に前記のように書き、<math>nはステートメント番号で、IがLからMまでNおきにnまでの間のステートメント群を反復実行せよということになる。Nが1ならこの例のように省略してもよい。D***2は D^**2 のことで、-般にD****Eは D^**E を示す。

4.3 COBOL

事務計算と技術計算の大きな違いの一つは事務計算の入出力データ構成の複雑さである。COBOLがALGOLやFORTRANと一番異るのもこの点である。1959年アメリカの国防省が中心となり、多くの計算機メーカが加わってCOBOL言語を作る作業が開始された。そして1960年にCOBOL60、同61年COBOL61、1963年初頭に、COBOL61 EXTENDED、1965年にCOBOL65 が発表され現在に至っている。

COBOL のプログラムは次の4つの部門にわけて書く。

- IDENTIFICATION DIVISION
 プログラム名, プログラマー名, 日付等プログラムの見出しを書く。
- ENVIRONMENT DIVISION
 使用する機械名,入出力装置の割り当てなどを行う。
- DATA DIVISION
 そのプログラムにあらわれるすべてのデータの性質、即ちファイルやレコードの構成、次元の指定、数値の型、析数等を指定する。
- PROCEDURE DIVISION
 せまい意味のプログラムで、演算手続きを書く。
- COBOL の特長は次のような点にある。
- (1) ハードウェアとの関連を明記してあるので、新らしい計算機におきかえる時に、主としてEN-VIRONMENT DIVISIONを変更すればよい。
- (2) プログラムがそのまま文書として利用できる。
- (3) 事務計算特有の入出力の複雑さを充分考慮に入れてある。
- (4) 反面,要素が多すぎてプログラムが冗長になりやすい。 すべての機能を含むというわけにはいかないが,簡単な例題を1つあげておこう。

「例題 図4.3 の給与計算の入力データをもとに、失業保険、差引給与額を計算して出力テープを作れ。」

図 4.3 給与計算のデータの例

入力データ

		j	給	j	——— ∌	
	表 示	項目		支給項目		
所	番	氏	職	基	特	3
				本	別手	Å

出

名 位 給

カ

Į			NAD:		7			/3		
	表示項目				支給項目			控除項目		
	所	番	氏	職	基	特		消	雑	
	i				本	別手		費組	控除	
	属	号	名	位	給	当		合	計	
	(5)	(5)	(21)	(3)	(6)	(6)		(6)	(6)	

(6)(5) (21) (3) (6) (6) (6) (6) (6) (4) (6)

当 計

給

出

消 支

組

力 控除項目

控 費

保

料

差

31

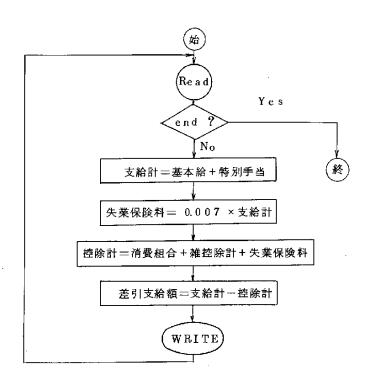
支

紿

額

この手順を流れ図に書くと図4.4のようになる。

図4.4 COBOLの例題の流れ図



これを COBOL で書くと次のようになる。

IDENTIFICATION DIVISION.

PROGRAM-ID.

KYUYO-KEISAN

AUTHOR. HASIMOTO-N.

DATE-WRITTEN.

1969-1-14

REMARKS.

"REIDAI"

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. JIPDAC.

JIPDAC. OBJECT-COMPUTER.

INPUT-CUTPUT SECTION.

FILE-CONTROL.

SELECT KYUYO-INPUT-FILE ASSIGN TO TAPE1.

SELECT KYUYO-OUTPUT-FILE ASSIGN TO TAPE 2.

DATA DIVISION.

FILE SECTION.

FD KYUYO-INPUT-FILE BLOCK CONTAINS 10 RECORDS.

LABEL RECORD IS STANDARD

DATA RECORD IS KYUYO-INPUT.

- 01 KYUYO-INPUT.
 - 02 HYOZI-KOMOKU.
 - 03 SYOZOKU PICTURE 18 9 (5).
 - 03 BANGO PICTURE IS 9 (5).
 - 03 SIMEI PICTURE IS A (21).
 - 03 SYOKUI <u>PICTURE IS</u> 9 (3).
 - <u>02</u> SIKYU-KOMOKU.
 - 03 KIHON-KYU PICTURF IS S9 (6) V.
 - 03 TOKUBETU-TEATE PICTURE IS S9 (6) V.
 - 03 SIKYU-KEI PICTURE IS S9 (6) V.
 - 02 KOZYO-KOMOKU.
 - O3 SYOHI-KUMIAI PICTURE IS S 9 (6) V.
 - O3 ZATU-KOZYO-KEI <u>PICTURE IS</u> S9 (6) V.
 - 03 SITUGYO-HOKEN-RYO PICTURE IS S9 (4) V.
 - 0 3 KOZYO-KEI
- PICTURE IS S9 (6) V.
- 02 SASIHIKI- SIKYU PICTURE IS S9 (6) V.

FD KYUYO-OUTPUT-FILE BLOCK CONTAINS 10 RECORDS

LABEL RECORD IS STANDARD

DATA RECORD IS KYUYO-OUTPUT

<u>0 1</u> KYUYO-OUTPUT

 $\underline{PICTURE}$ IS X (80).

CONSTANT SECTION.

77 KEISU PICTUPE IS SV999

VALUE IS 007
PROCEDURE DIVISION.

HAZIME. OPEN INPUT KYUYO-INPUT-FILE OUTPUT KYUYO-OUTPUT-FILE.

YOMU. READ KYUYO-INPUT-FILE AT AND GO TO OWARI.

COMPUTE SITUGYO-HOKEN-RYO = KEISU * SIKYU-KEI.

 $\frac{\text{COMPUTE}}{\text{KOZYO-KEI}} = \text{SYOHI-KUMIAI} + \text{ZATU-}$ KOZYO-KEI + SITUGYO-HOKEN-RYO.

COMPUTE SASIHIKI-SIKYU = SIKYU-KEI - KOZYO-KEI.

WRITE KYUYO-OUTPUT FROM KYUYO-INPUT GO TO YOMU.

OWARI. <u>CLOSE</u> KYUYO-INPUT-FILE KYUYO-OUTPUT-FILE.

STOP RUN.

アンダーラインのあるものは COBOL 固有の単語である。 ENVIRONMENT DIVISION でわかるように(Compile) する機械(SOURCE-COMPUTER)とオプジエクトプログラムの入る機械(OBJECT-COMPUTER)とが違ってもよい。

事務計算で扱われるデータの構造は、ファイル(file)、レコード(record)、項目(item)という順に細分類され、更に項目には集団項目(group item)とか基本項目(elementary item)とかいくつかのレベルがある。COBOLのDATA DIVISIONでは、このデータの構造を最高49レベルまで使用することができる。この例ではファイルとしてFD KYUYOーINPUTーFILEとFD・KYUYOーOUTPUTーFILE、レコードとして01 KUYOー

FD ファイル 01 レコード 02 集団項目 ↓ :: 49 基本項目 INPUTと01 KUYO-OUTPUT,集団項目として02 HYOZI-KOMOKU など,基本項目として03 SYOZOKU などの3つのレベルにわけてある。常に最低のレベルが基本項目となり、これにはPITURE としてデータの形を表示する必要がある。PITURE の意味は9(5)は5 の数字で9999と同じである。A(21)は21字の文字、S9(6)Vは符号のついた6桁の数字で小数点が最後につく、同様にS9V999 は小数点が上から2桁目にある4桁の符号付数字である。

この例では入力データの一部に結果を書き込んで出力データとする形をとっているので、ファイルの構造は入出力とも同じであり、入力ファイルについてのみ明細を表示してある。こういう場合には PROCEDURE DIVISION の中で演算結果を

WRITE KYUYO-OUTPUT FROM KYUYO-INPUT という形で出力テープに書き出す。

5. プログラミングの手順

プログラムの最終的な形は、何等かのプログラミング言語で仕事の手順を記述したものになるが、どんな 仕事にしろ、最初からすぐあるプログラミング言語、例えばFORTRANPCOBOLで書き流せるもので はない。そこに至るまでには計算機特有のいくつかの段階がある。

まず最初はシステム設計とか問題解析とかいわれるもので、これは仕事の種類によって、やるべき事はそれぞれ違う。

例えば、事務データの処理のような仕事なら、現行のシステムを分析し、機械化し易い様に改良を加える。 取り扱うデータのコード化や標準化を行い、張票の設計や、レポートの基準をきめる。 科学技術計算なら、問題を表現する関係式や方程式を導びき、計算機で出来る計算、すなわち四則演算に おきかえる。これらの仕事を通常、数値解析と呼んでいる。

そのほかオペレーションズ リサーチや統計解析のような分野では、予測式の導入とかシミュレーション モデルの設定とか、それぞれの基本的な仕事がある。

これ等の仕事は一般にはプログラミングの中には含めない。そしてふつうこれ以後の仕事をプログラミングという。すなわち、プログラム設計、流れ図作成、コーディング、デバックドキュメンテーションの作成という分け方が普通おこなわれている。

5.1 プログラム設計

何をするプログラムか、大きさはどの位か対象とするコンピュータの規模は、などによってプログラム設計の内容も重点のおき方も異なるのは当然だが、共通的な仕事としては次のようなものがある。

- 1. モジュール化: 大きなプログラムでは、全体をひとまとめにして大きなものにせず、いくつかの独立した機能のプログラムに分けて作る方がよいとされている。その分け方をどうするか、お互いの関連をどう持たせるか、それぞれの情報の受け渡しの条件などをきめねばならない。
- 2. 使用言語の選択: これはもっと前の段階できめられることもあるし、コンピュータの性能から 殆んど選択の余地のない場合もあるが、選択の余地があるとすれば、判断基準としては、仕事の内 容にマッチしている、プログラムが作り易い、互換性、処理効率のよさ、今後の保守のしやすさ などが考えられる。これらはお互いに矛盾した条件である場合もあるが、必要なら一つの仕事の中 で部分的に違う言語を使いわけた方がよいこともある。
- 3. 記憶装置の割当て: 主記憶装置と補助記憶装置の分担をきめる。容量,演算速度,ハードウェアの機能などの条件から最適の使いわけを定めねばならない。場合によっては主記憶装置を何回か重複使用するオーバーレイ(overlay) の必要性もでてくる。記憶装置の割当ではモジュール化とも大いに関連がある。また補助記憶装置の種類の選択は,あとでのべるファイル設計の仕事でもある。
- 4. 入出力装置の割当て: システム設計の段階で,主な入力や出力の媒体はすでに決定しているだろうが,具体的な装置の割り当て,チャンネルの使いわけ,マルチ処理で効率を上げるなどの細かい設計が必要となる。
- 5. コード設計: システム設計の段階で行われたコード体系を,さらにプログラム,テクニック上の要求を入れて詳細化しなければならない。将来の変更や拡張が予想される場合は,当然それを考慮に入れて設計しておかないと取り返しがつかなくなる。
- 6. 入出力フォーマットの設計: 基本的なことは、やはりシステム設計の段階できめられているだろうが、プログラム技術上の細かい点をとりきめねばならない。
- 7. ファイル設計: ファイルの構造は、実際の処理効率に大きな影響を及ぼす。記録媒体の種類、 処理の頻度や内容、基礎的なソフトウェアのサポート条件などによって決定せねばならない。
- 8. チェック方式の設計: プログラムは、誤った入力データや、オペレーションミス、ハードウェアのトラブルなどによっておこる障害を未然に防ぎ、あるいはそれによる事故を最少限にくい止めねばならぬ使命を持っている。そのための適当なチェック機構をプログラムの中に組み込んでおかねばならない。処理時間や誤りの発生の可能性をよく検討した上で、どこでどんなチェックを行うかを定める必要がある。
- 9. テストデータの作成: そのプログラムが正しく働くかどうかをたしかめるには,テストデータ

を通して見なければならない。テストデータには勿論正しい答が何等かの方法で既に出されている 事が必要である。下手なテストデータを作ると、実際には存在しているプログラムのエラーが、う まくかくれてしまって発見されなかったり、プログラムのある部分は全くテストされなかったりす る事が多いので、この仕事は大へん重要である。

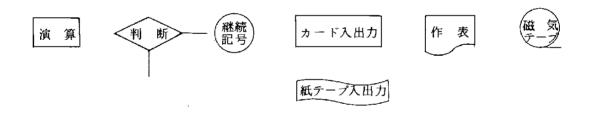
10. デバッグ方法の検討: プログラムが正しいか否かをテストし,エラーを退治する仕事をデバッグという(後出)。大きなプログラムや特殊なプログラムでは,予めこの方法を考えておくのは必要なことである。デバッグのための特別なプログラムを作る必要が起る場合もある。

以上のほかに基本的には、使用する計算機のハードウェア、ソフトウェアの両機能をよく理解している事が勿論前提となり、実際の作業上の問題としては、作業工程、機械使用予定などの作成、また複数人でやる場合はその分担、使用する記号や表現の標準化、お互いの打ち合わせ、または連絡の方法などもきめておく必要がある。

5.2 流れ図の作成

仕事の手順を図的に表現するということは、問題の流れを手取り早くつかむ上にも、また間違いの発生を防ぎかつ訂正をたやすくする上からも是非必要なことである。普通、大まかなブロックダイヤグラム($block\ diagram$)と、もう少し細かいフローチャート($flow\ chart$)と二段構えに作ることが望ましいとされている。また図 5.1 に示すようにそれぞれの作業をあらわす図形がほどきまっている。

図 5.1 フローチャート表現の一例



5.3 コーディング

処理手順を、計算機が理解できることば、すなわちプログラミング言語に書き直す仕事がコーディング(coding)である。

機械語によるコーディングを機械コーディング(machine coding), 記号言語によるコーディングを記号コーディング(symbolic coding), 自動言語によるコーディングを自動コーディング(automatic coding)という。

コーディングは計算機への直接の入力をつくるところで不注意による誤りが介入しやすいところであるとともに、プログラミング入門の基本となるところでもある。

実際の仕事の手順とは別に、プログラミングの勉強はまずこのコーディングから始めるのがふつうである。

5.4 デバッギング

一通りでき上ったプログラムをまちがいがないかどうか調べる仕事をデバッギング(debugging) という。主として計算機を使ってまちがいを見つけるが、その方法の上手か下手かが、プログラマーの 熟練の一つのバロメータともなっている。紙の上では充分検討を加え、もう絶対にまちがいはないと、かなり自信のある場合でも計算機で実行させてみると、かくれた誤りがいろいろ見つかるという。なかなかやっかいなことも起こりがちである。

デバッグのデ(de)は除く意味で、バッグ(bug)は南京虫である。南京虫を除く、つまり不要なプログラムの誤りを除くという意味のことばである。

5.5 文 書 化

プログラミングの仕事の三分の一は文書(documentation)をまとめる仕事だという人がいる。 最近、計算機で処理する仕事は大規模のものが多くなったので、一人が一つの仕事を始めから終りまで 仕上げるというよりは、数人、場合によっては数十人が共同して一つの仕事を仕上げることのほうがふ つうである。お互いの意志の疎通、コミュニケーションを円滑にするには、確認事項、連絡事項をその つど文書化してきちんとまとめておくことが必要である。

最後にでてきたが、この仕事はけっして最後にまとめて行なうものではなくて、プログラム設計、流れ図、コーディング、デバッギングなどの各段階でそのつど行なわねばならない仕事である。

文書の内容として必要なものは、プログラムの設計書、プロックダイアグラム、フローチャート、コーディングリスト、使用法説明書、テストデータと結果などであるが、そのまとめ方、形式などは、各社、各センターあるいは各業務ごとに標準化し、規格化する必要があろう。そして重要なことは、後日システムの一部を変更したいときに、その文書を頼りに仕事ができること、しかもかってその仕事に従事した者でない、まったくの第三者でも、その文書だけで作業が完全にできることが望ましいとされている。したがって、文書は正確、綿密であるばかりでなく、だれにでもわかるように平易に書かれていることも必要であろう。

6. ソフトウエアとは

電子計算機の能力は、(ハードウエア)+(ソフトウェア)であるといわれる。ハードウェア(Hard-ware) という言葉は昔からあった。これは計算機そのもので日本語では「金物」といわれる。これに対してソフトウェア(Software)という言葉は電子計算機固有なもので「利用技術」などと訳することがある。広い意味では、何もかも一切のプログラムを含めてソフトウェアという場合もあるが、一般には特定の一ユーザーのみに必要なプログラムではなくて、少くとも幾つかのユーザーに共通して使われるプログラムを総称する場合が多い。

そういった意味から一通りのソフトウェアは計算機メーカー側で整備して、ハードウェアとともにユーザーに提供し、ユーザーはハード、ソフト両者を含めたものがその計算機の能力であるという感覚で使用するというのが普通である。

さて、このソフトウェアの種類や機能は、計算機の規模、使用目的などにより違いもあり、また時代とともにその内容や分類も変化してゆくものであるから一概にはいえないが、さしあたり表 6.1 のように分類してみる事が出来る。

- (1) 技術計算用ライブラリ
 - 線型計算
 連立一次方程式,逆行列,行列演算,固有值
 - 2. 代数方程式
 - 3. 数值積分,数值微分
 - 4. 補 間 法
 - 5. 徵分方程式, 積分方程式
 - 6. 関数 sin x, cos x, e, log x, tan⁻¹ x, Bessel等
 - 7. 関数近似
- (2) オペレーティング・システム
 - 1. 制御プログラム
 - 2. 言語処理プロセッサー コンパイラ, アセンブラ
 - サービス・プログラム ローダー 入出力処理プログラム ソーティング・プログラム ファイル処理プログラム ライブラリー編集プログラム ディバッグ・プログラム
- (3) アプリケーション・プログラム
 - 1. オペレーションズ・リサーチ用プログラム
 - 2. シミュレーション用プログラム
 - 3. 統計計算用プログラム
 - 経営管理用プログラム 日程計画,生産管理,在庫管理,販売管理,需要予測等
 - 5. 工学用プログラム 土木建築,原子力,機械工学,化学工業,電力,造船等
 - 6. 情報検索用プログラム

6.1 技術計算用ライブラリ

技術計算で日常さかんに使われるようなプログラムは、

個々の使用者が、その都度作製するという重複をさけるため、共通性のあるものを予め一通り誰かが 作っておいて、他の人はそれを信用して利用しようというのが目的である。この性質からも当然、プロ グラムとしてのすぐれた特長を幾つか兼ね備えていなければならない。

例えば,

- (1) ステップ数をできるだけ少なくする工夫がしてあること。
- (2) 演算時間が早いこと。
- (3) 計算誤差が少なく数値解析的に吟味されること。
- (4) 一般性があること。(例えば連立方程式なら何元でも解けるように)
- (5) 使い易いこと。(特にパラメータの種類や書き方があまり複雑でないこと)
- (6) 使用者の誤りが指摘できること。(データ数の不足やパラメータの書き方の不備など)

以上の(1)~(6)はお互いに矛盾する条件もあり、なかなかむずかしい注文であるが、これらに必らずしもライブラリープログラム(Library Program)に限ったわけではなく、何回も繰り返し使用するもの、あるいは多くの人が共通に利用するプログラムは以上のような点の吟味が充分されていることが望ましい。

ライブラリープログラムは普通、磁気テープにまとめて保管されるが、同時に次のような内容をもった詳細な説明書をつけておく必要がある。

題名,整理番号,目的,語数,精度,時間,使用上の注意(丁寧に),解析法,流れ図,コーディングシート,作成者名,年月日等

6.2 オペレーティング・システム

最近の計算機は高速、大容量、多岐の入出力装置、多重処理など、ハードウェアの機能がますます増大しつつある。このような機能の向上はまことに結構なことではあるが、一方これを使う立場からいうと、これらの機能を十二分に無駄なく発揮させるにはどうしたらよいかと苦労することになる。

また、計算機が高速になってくると、1日にかける仕事の種類も相当増えてくる。1つの仕事を計算機にかけるには、プログラムを読ませたり、磁気テープを取り換えたり、指定通りスイッチをセットしたりする、いろいろな作業が付随することが常である。1日にかける仕事の量が増えるということは、それだけ手作業の仕事も増えるということで、例えば、1日8時間稼動時間があったとしても、そのうち3時間はオペレーターの手作業の時間であるというようなことにもなりかねない。これでは甚だ能率が悪いので、1日又は半日分の仕事をあらかじめあたかも一つの仕事のように編集してしまい、最初一度オペレータがスタートさせると、あとは次から次と自動的に仕事を処理して、途中に入力を介入させないですませるような自動運転が必要になる。

このようにハードウェアの機能を助けて、それをより有効に生かし、機械の稼動効率をあげるための ソフトウェアをオペレーティング・システム(Operating system 略してOS)と呼んでいる。 オペレーティング・システムの役目は次のようなものがある。

1. 異なった言語によるプログラムの連続処理

OSはいろいろなアセンブラ、コンパイラ等を全部支配下においているので、どんな言語で書かれたプログラムがきても、その都度必要な翻訳プログラムを自由に呼び出してきて使うことができる。

2. プログラム・エラーの検出とディバック機能

入力の介入を許さないということは、プログラマーが自分で計算機を操作しながらディバッキング するという事も不可能になることである。そのためにはプログラムの誤りを検出し、どんな誤りかと いう情報を出し、プログラマーに訂正の手がかりをあたえる手段が講じてなければならない。また、 自由に呼び出せる強力なディバックプログラムが内蔵されている必要もある。

3. ライブラリの管理

ライブラリの追加、変更に応じられるように管理を行い、いろいろなジョブが要求するライブラリ

をとり出してきて、うまく連結して使うことができるようにする。

4. 時間の管理

内蔵の時計で時間の管理を行い、時間によって仕事を中断したり、切りかえたりする。

5. 入出力制御

チャンネルやユニットの管理、割りあて、入出力割込みの処理等を行う。

6. 多重処理スケジューリング

仕事の優先順位にしたがって処理順序のスケジューリングを行ない, なるべく効率のよい多重処理 を実行する。

7. メモリーの割りあて

各ジョブにメモリーを割りあてる。とくに複数個のジョブが同時に働くようなシステムでは、外部 記憶も含めたスワッピング(Swapping)の管理も必要となる。

8. セグメンテーション

一つのジョブが、いくつかのセグメント(segment)に分かれている場合は、セグメント間のコントロールの受け渡しが必要となる。

9. ファイルの管理

プログラムファイル,データファイルを含め,ファイルの作成,更新,保管等一切の管理を行なう。

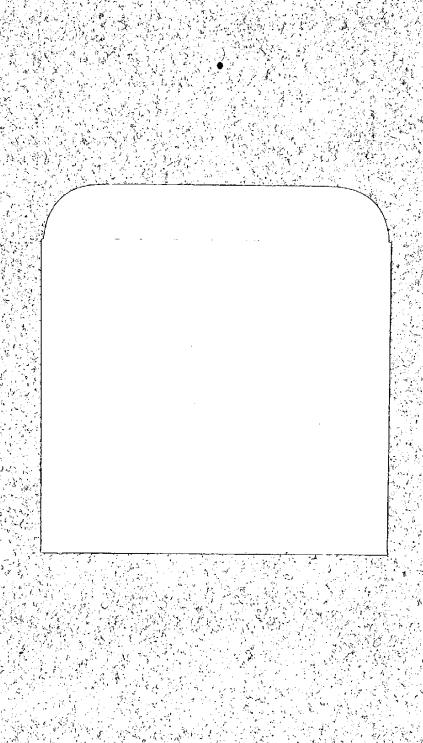
10. エラーの処理

できるだけ計算機を停めないという方針で、予測しないハードウェアのエラーに対しても、できる限り自己回復の処置が必要である。

オペレーティング システムの管理下で仕事を行う場合は、プログラム本体の他に、OSに対するいくつかの指定が必要になる。例えば使用する言語の種類、ライブラリーの名前、翻訳のみか、演算も行うか、演算続行希望時間、使用する磁気テープの本数、失々の仕事の始まりや終了の表示等、いくつかの項目を、定められた規則にしたがって指定しなければならない。これらをコントロール・パラメータと呼ぶ。

6.3 アプリケーション・プログラム

計算機のアプリケーションの範囲は無限に広い。したがって、アプリケーションプログラム((application program)の数にも限りがないわけであるが、とくに、これは時代と共にその 内容や種類の変化が甚だしい。したがって、アプリケーションプログラムとは、これこれあると限定するのはむずかしいが、現在比較的広く使われている分野を表 6.1 にあげておいた。特に工学分野では専門家のグループによる、共通に使用できる言語やプログラムの開発がさかんである。今後はどの分野においても、アプリケーション・プログラムの種類も質も急速に向上してゆくことが期待される。



JIPDEC

財団法人 日本情報処理開発センター