

東大 HITAC 8800 / 8700 システム

と長時間バックグラウンド・ジョブ

東大 大型計算機センター 石田 晴久

1. はじめに

東大大型計算機センターでは、1973 年 1 月より図 1 に示すような超大型電子計算機システムが本格稼働に入ることになっている。このシステムの主な特徴は次の通りである。

(1) 高速である。平均命令実行時間は HITAC 8700 で 160 μ sec 程度、HITAC 8800 で 700 μ sec 程度と予想される。

したがって H8800 は H5020E より約 10 倍速いと期待される。しかし各命令の実行時間は、パイプライン制御・バッファ・メモリ・アドレス変換（連想レジスタ使用の有無）・コア位置・マルチプロセッシング効果などに大幅に左右されるので、従来の計算機に比して演算速度の推定が困難である。

(2) 主メモリが大きい。コアメモリの大きさは 3 MB（メガバイト）、すなわち 768 kW（1 語 32 ビット）ある。ただしこのうち管理プログラムが 700 KB = 175 kW 程度使うの

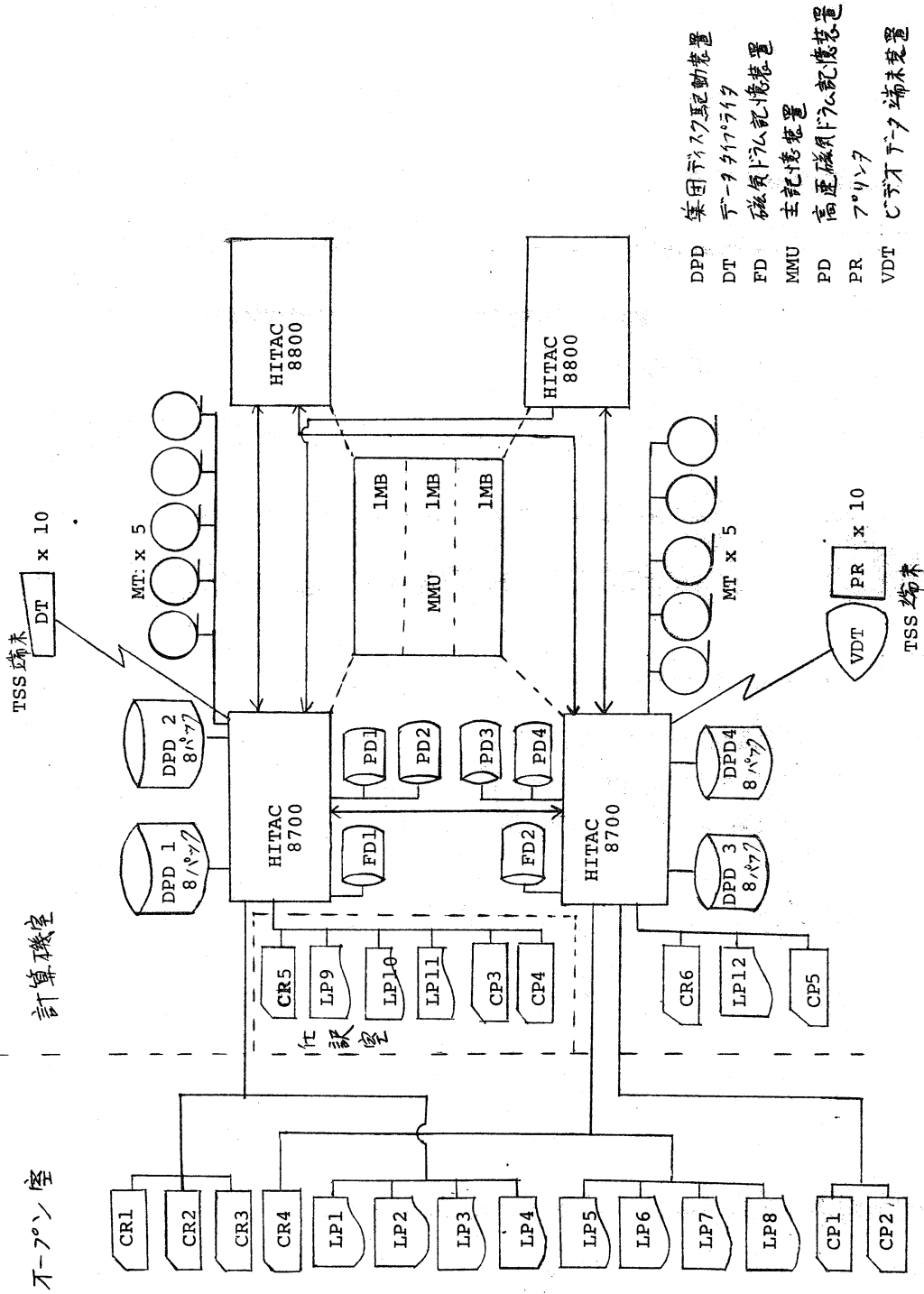


図1 東大 HITAC 8700/8800 システム

ユーザ・エリアは $2.3 \text{ MB} = 593 \text{ kW}$ 程度になる。また将来リモートバッファ端末が増えたとユーザ・エリアはさらにへることになる。この主メモリでは H8800 に対しては 2 MB 分が 8 way (転送速度 $1.25 \mu\text{sec}/64 \text{ B}$)、 1 MB 分が 4 way ($2.5 \mu\text{sec}/64 \text{ B}$)、また H8700 ではすべて 4 way のインターリーブが行われる。

(3) 仮想メモリ機構がある。このため、自動オーバーレイ・ダイナミックリンク・リエントラントプログラミング・ページ (1 kW) あるいはセグメント (64 kW) 単位のアクセス制御・非連続領域の有効利用・スワッピング容易などの利点が出てくるが、一方これらは効率低下の原因となる要素もたくさん含んでいる。

(4) バッファ・メモリがある。これは各 CPU 毎にコアと CPU の中間にあって、当面必要な命令やデータを保持しておいて CPU とメモリのアクセス能率を上げるのに使われる。

	各 H8800	各 8700
容量	$32 \text{ KB} = 8 \text{ kW}$	$16 \text{ KB} = 4 \text{ kW}$
データ幅	$8 \text{ B} \times 2 = 4 \text{ W}$	$8 \text{ B} = 2 \text{ W}$
サイクル・タイム	100 msec	210 msec

(5) 4 台の CPU によるマルチプロセッシング。4 台の CPU が全部演算命令も実行している状態では、システム全体として 1 命令を平均 66 msec 程度で処理することになる。この 4 台の CPU の他に 4 台の入出力処理装置がコアを共用する。こ

のためコアやチャンネルでのぶっかかり合いなど効率低下の要因もあるが、コアその他のリソースの有効利用が期待でき、またオペレータなど運転要員の数もへらせると期待される。

(6) 補助メモリの容量が大きい。

	台数	容量/台	平均アクセス
ページング・ドラム	4	4.3 MB	10.3 msec
ファイル・ドラム	2	4.5 MB	10.3 msec
基団ディスク	4	233 MB	87.5 msec

このディスクは1974年中に容量800 MBのもの2台と入換える予定になっている。

(7) 機械語はIBM 360のスーパーセットになっている。IBM 360からの拡張部分はほとんど管理プログラムで使われる機能であるから、IBM 360用のユーザプログラムはほとんどのままかかる。

(8) RAS (Reliability, Availability, Serviceability) を支える機能がある。これには命令の再実行(最高15回まで)・2検出・1訂正の誤り訂正コード(情報64ビット+チェック8ビット)の使用などが含まれる。また自動再構成(故障したCPU・コアメモリ・入出力処理装置などの切離し)や故障箇所指通テスト(FLT = Fault Locating Test), エラー情報(最大4KB)の自動ログアウトの機能などもある。

(9) オペレーティング・システムが汎用になっている。このためバッチ処理・リモートバッチ処理・TSSなどのサービス、個人ファイルの使用が出来るようになっている。言語には FORTRAN (compiler/converter/syntax checker), PL/I, BASIC, COBOL などがある。

2. バックグラウンド・ジョブ

東大システムでは4台の高速のCPUが768 kWのコア・メモリを共用する形になっているため、次のようなマルチプログラミングが行われることになっている。

	タスクの種類	多重度	コア容量
システム・タスク	ジョブ・スケジューラ	1	0.43 MB
	マスタ・スケジューラ	4	
	会話イニシエータ	1	
	入力リーダー (CR6台)	6	0.20 MB
	出力ライター (LP12台, CP5台)	17	
ユーザ・タスク	バッチ・タスク	14 各 140 KB = 35 kW	1.96 MB
	TSS タスク	2 各 100 KB (ひとつはスワップ中)	0.30 MB
(多重度統計 45)		(計 2.89 MB)	

ユーザのジョブはいちどシステムの中で実行状態になると(タスクになると)、バッチ処理の場合、図2のように、ディ

ク（またはファイル・ドラム）内の入力ジョブキュー、出力ジョブキュー、ディスク内の入力情報・出力情報、スワッピング・ドラムなどを使って実行をやることになる。

この場合、CPUの演算速度が4台で 66 msec / ステップ程度 であるのに比べて、ドラムやディスクの平均アクセス時間が 10 msec ないし 90 msec と長いので、すべてのユーザ・タスクが入出力待ち（ドラムやディスクへのアクセス待ち）になって、CPU（とくに演算担当のH8800）が遊んでしまう状態が起こりうる。そこでCPUの遊休時間を利用して処理をせよというのがここでのバックグラウンド・ジョブである。

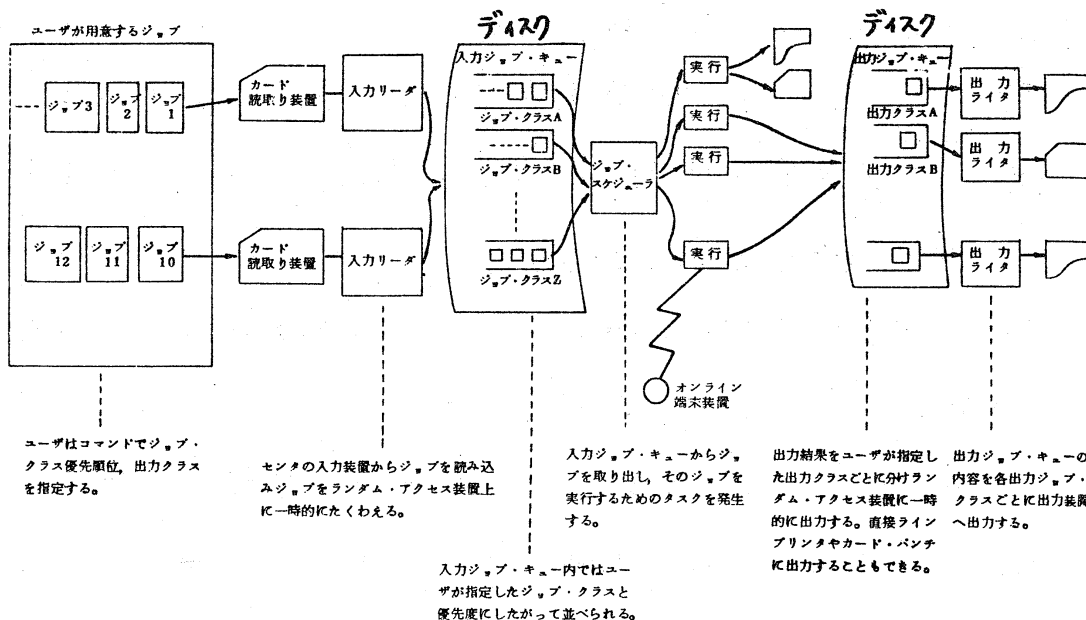


図2 バッチ処理の流れ

この目的には、バックグラウンド・ジョブ用にひとつのジョブクラスを割当て、そのクラスのジョブで発生するタスクの優先順序をうんと低くしておけばよい。バックグラウンド・タスクの優先順位がずっと低ければ、このタスクは普通はほとんど実行されることなく、他のクラスのタスクがすべて入出力待ちで実行されなくなつて始めて実行されることになる。

このタスクの優先順位を指定するプログラムは、センサー・オンライン・コーディングによる「タスク優先順位決定モジュール」として、東大センサーで管理プログラムに組み込む予定である。このモジュールで決定する項目は次の通りである。

- (i) 当のジョブを処理するユーザ・タスクのタイム・スライス値 (msec 単位)
- (ii) 上記タスクの実行優先権 (0~255, 大きいほど高い)
- (iii) 上限優先権 (CHAP = CHange Priority マクロで変更できる実行優先権の上限)

バックグラウンド・ジョブ用のジョブ・クラスについては次のような項目を指定しなければならない。

- (a) ラインプリンタ出力のページ数とライン数の上限
- (b) カードパンチ出力上限 (0にしておいても可)
- (c) 仮想メモリ上限 (1kwのページ単位) = プログラムの全体的な大きさの上限

- (d) 1 時に使用できるコアメモリ上限 (はみ出した部分はドラムから自動的にオーバーレイされる)
- (e) CPU 使用時間上限 (単位 10 msec)
- (f) 経過時間 (コア時間ともいう) (単位 10 msec)
- (g) 取出し比率 R_i ($=0 \sim 32767$). X_i をクラス i からすでに取出されたジョブの個数とすると, システムは X_i/R_i を計算し, その小さい順に実行してゆく. たとえばクラスが A, B, C とあって, $R_A:R_B:R_C = 3:2:1$ なら ABCABA の順に取出される. この他に東大センタでは, R_i の値いクラスがたまたますいているために, そのクラスのジョブが取出される^{ということ}を妨ぐために, 各ジョブの入力時刻も考慮してスケジューリングを行う予定になっている.

バックグラウンドジョブとしては

- (A) プログラムが比較的小さいこと. コア内にあっても余りじゃまにならず, スワップも起らない程度のものがよい. たとえばコアの 1 割程度の 65kW ぐらい.
- (B) 入出力のデータ量が少いこと. またなるべくまとめて入出力すること. 入出力の回数が多いと, ディスク結ちになってバックグラウンド・ジョブでなくなってしまう.
- (C) 計算時間は非常に長いものでよい. たとえば 10 時間. た

だし CPU の空き時間にのみ走るから、ターンアラウンド時間は相当長くなると考えられる。東大の現システム HITAC 5020 E での実測によると CPU の稼働率 (busy rate) は平均して 67% , すなわち平均して CPU は 33% の稼働は遊んでいたという結果が出ている。次期システムのマルチプログラミングでこの率がどうなるかは非常に興味深い。

(D) 1日の終わりにシステムをシャットダウンするとき、ジョブが終了していないときに、強制的にそれを中断させて、翌日それを自動的に再開させることはできない。代りの方法としては、SINF (Set INForm) マクロを発行して、それに応えてオペレータから INFORM コマンドを出してもらって止めることはできる。この場合翌日も続行させるときは、オペレータに起動させてもらう。

(E) 中間結果をみるのが難しい。もっとも簡単なのはプログラムの中から WTO (Write To Operand) コマンドでオペレータ用文字ディスプレイに出すことであるが、それをユーザがみるのは事実上難しい。ラインプリンタへの出力はジョブが終了して出力ファイルが閉じられてからのみである。可能な方法としては、中間結果をある時点で自分の個人用ファイルに書出し、アセンブラ・サブルーチンでそのファイルも閉じておいて、別のジョブでそのファイルも取出す方法がある。

別のジョブにしなければならないのは、異なる二つのジョブが同時に同じファイルにアクセス（読出しだけでも）できないように管理プログラムが作られているためである。この中間結果も取捨すジョブ用には、もうひとつのジョブ・クラスが必要かもしれない。

中間結果をみるには TSS モードを使うことも考えられるが、実際には動作中の BATCH ジョブに TSS で手を触れることは難しい。(TSS → BATCH は起動の指定だから可能だが、BATCH → TSS はすでに動いているタスクに対するもので、両方の仮想空間が全く異なるので難しい。原理的には共通セグメントが設けられれば可能であるが)

(F) バックグラウンド・ジョブのユーザーは1人のグループ管理者のもとで、ファイル使用量なども管理することもできる。

3. プログラミング上の注意

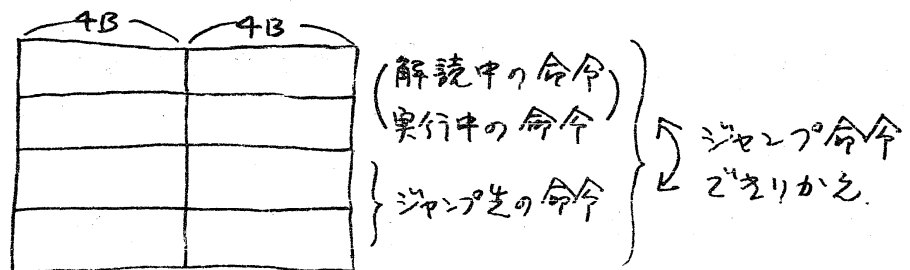
東大の新システムでは、マルチ・プロセッサのぶつかり合い、仮想メモリでのスワッピング、パイプライン制御の乱れ、バッファ・メモリ内の命令の有無など、効率低下の要因がいくつかあるので、それらを避ける注意が必要である。

(a) バッファ・メモリの活用をはかるには、1度に使うデ

66

タとプログラム・ループがなるべく32kB(8kW)に収まるようにする。こうすればプログラムは(データの書き込みを除いて)バッファ・メモリの中だけで実行されることになるので、計算速度が向上する。また書き込みが行われるときは、バッファの内容とコアの内容との一致を常にはかきため、コアへも書き込まれるが、バッファ・メモリに取込まれている部分へ書き込まれるのであれば、連続レジスタにより実アドレスがすぐ分るので、書き込みはそれだけ早い。これは図3の太線の流れに相当する。

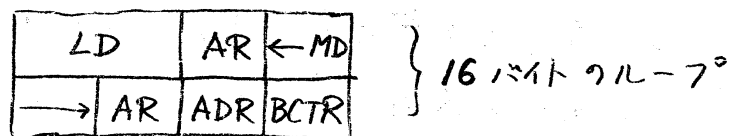
(b) パイプライン制御の効果を生かす。H8800では命令のバッファが次のように32バイト分ある。そこで行列の積



の計算に出こくる

$$S(I, K) = S(I, J) + A(I, J) \cdot B(J, K)$$

の機械語列の場合のように、



16バイト以内のループなら非常に速い（上の例は15サイクル
 $\times 50 \text{ msec} = 0.75 \mu \text{ sec}$ / ループである）ことは頭に入れて
 おくべきである。この場合、先行制御が全く行われていない
 とすると、LD (8サイクル)、AR (3)、MD (13)、ADR (16)、BC
 TR (6) で、上のループは $41 \text{ サイクル} \times 50 \text{ msec} = 2.05$
 $\mu \text{ sec}$ がかかることになる。ただし以上は命令がバッファ・メモ
 リ内にあるとしての話で、もしもバッファ・メモリになけ

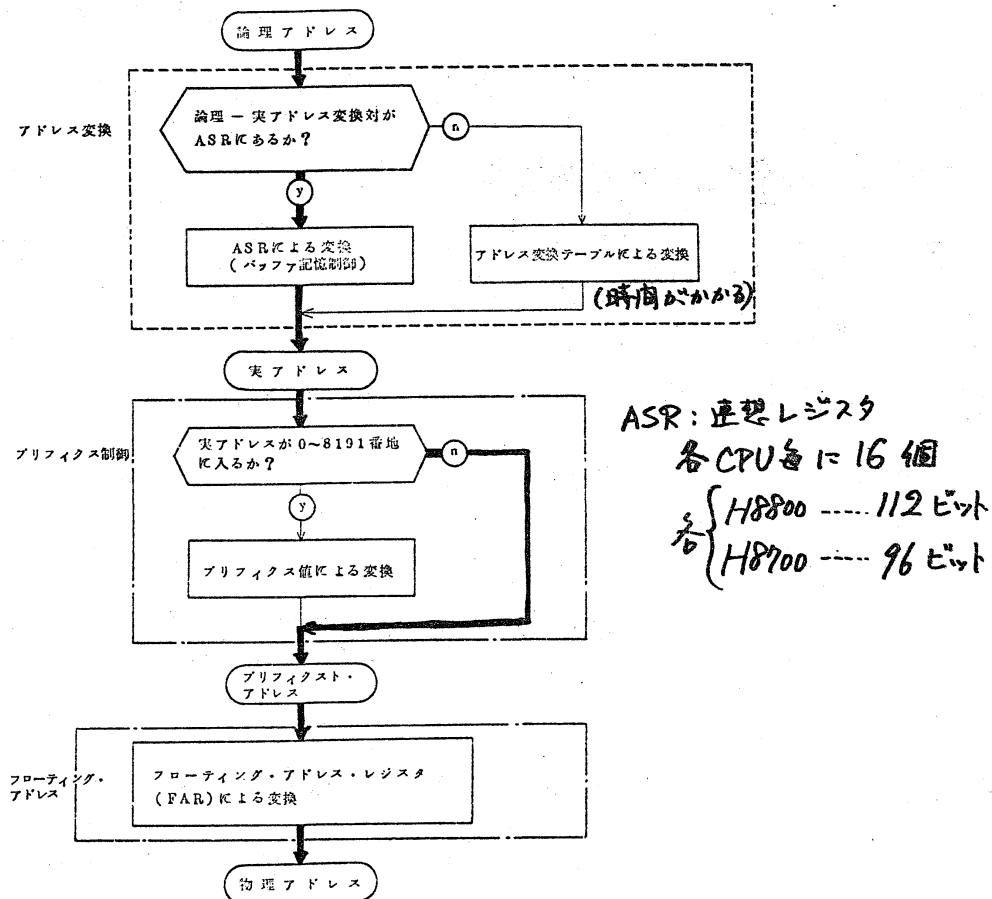


図3 アドレス変換過程

68

いは、アドレス変換やコアからバッファへの転送などごまらに時間がかかる。

(C) H8800/8700 の FORTRAN (およびハードウェア) で扱える数の範囲は次の通りである。

IH) INTEGER *2 $-2^{15} \sim 2^{15}-1$ $-32,768 \sim 32,767$ (4桁)

I) " *4 $-2^{31} \sim 2^{31}-1$ $-2,147,483,648$
 $\sim 2,147,483,647$ (9桁)

ID) " *8 $-2^{63} \sim 2^{63}-1$ $\sim 9,223,372,036,854,775,807$

[負整数は2の補数表示] (18桁)

R) REAL *4 $16^{-64} \sim 16^{63}$ $10^{-78} \sim 10^{75}$ 約7.2桁

D) *8 " " 16.8桁

Q) *16 " " 33.6桁

実数は符号 + 絶対値表示

実数の内部表現、例とあげると次のようになる。

	指数部 (excess 64)	仮数部 (24/56/112 ビット)
無限大	0 1111111	. 111111 - - - - - 11111
無限小 非正規化	0 0000000	. 000000 - - - - - 00001
" 正規化	0 0000000	. 000100 - - - - - 00000
真のゼロ	0 0000000	. 000000 - - - - - 00000

(注) $2^{-24} = 0.596 \times 10^{-7}$

$$2^{-56} = 0.138 \times 10^{-16}$$

$$2^{-112} = 0.192 \times 10^{-33}$$

ハードウェアは次のような4倍長演算機能ももっている。

$$Q \pm Q \rightarrow Q \quad \text{normalized}$$

$$D * D \rightarrow Q \quad "$$

$$Q * Q \rightarrow Q \quad "$$

ただしQのやりぎんとIDの±*/はソフトで行われる。

(a) 演算に関連したやりこめには次のものがある。

	マスク	実行	結果 FORTRAN (ハード)	コンディション・コード FORTRANによる検出	C.C
整数オーバーフロー*	○	○	-----	-----	3
〃 零除数	×	×	メモリ不変	CALL DVCHK(4)	不変
実数・零除数	×	×	メモリ不変	〃	不変
〃 指数オーバーフロー	×	○	∞ (正規化) ($e=e+128$)	CALL OVFLR(4)	はセット */不変
〃 〃 アンダーフロー	○	○	0 (正規化) ($e=e-128$)	-----	3
〃 零仮数	○	○	-----	-----	0

(*) ひきぎんは、I-Jの場合

$$I + 1 + (\text{Jの1の補数}) = \text{答の形で行われる。}$$

したがって

$$0 - (-\infty) = 0 - (100 \dots 00) = 0 + 1 + (011 \dots 11) \rightarrow \text{オーバーフロー}$$

$$(-\infty) - (-\infty) = 100 \dots 00 + 1 + 011 \dots 11 = 0 \rightarrow \text{オーバーフロー}$$

こうしたやりこめが起こったときの処置はFORTRANでは、

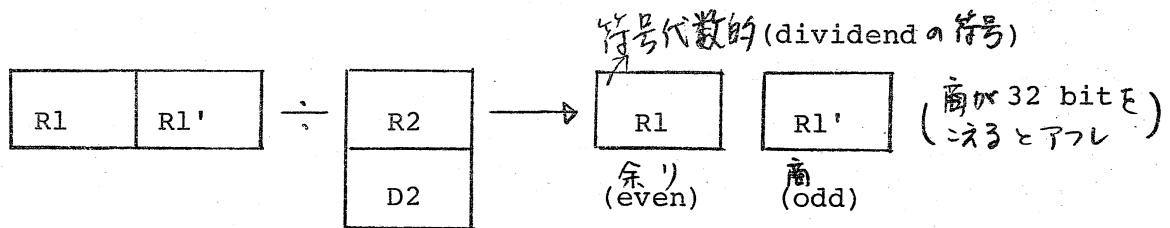
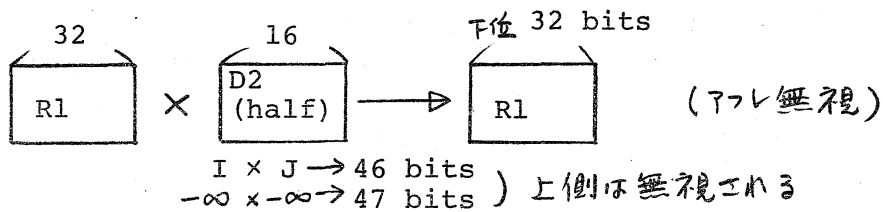
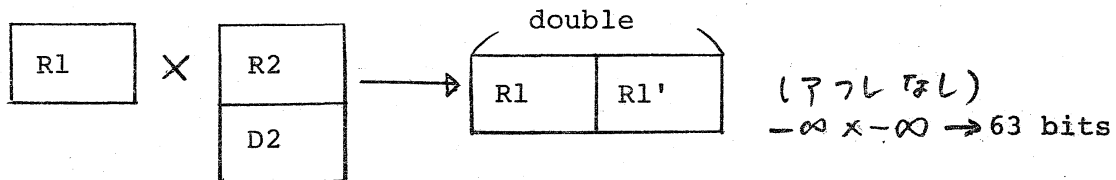
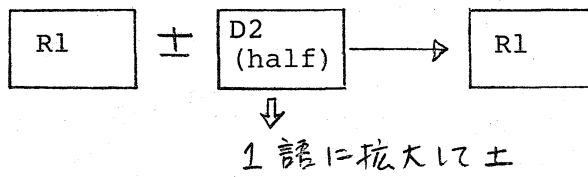
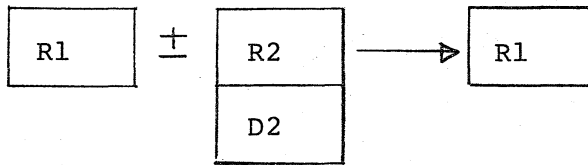
CALL ERSET (IE, IN,)

で指定できる。ただし IE はエラー番号, IN は打ち切り回数
を表わす。バックグラウンド・ジョブではエラー処理につ
いては万全の配慮が必要である。

参考文献

1. 中沢ほか: "超高性能電子計算機のシステム設計" ほか 4
件, 情報処理学会大会予稿集, pp. 359-368 (1970)
2. "超大型機特集号", 情報処理学会誌, Vol. 12, No. 8 (1971)
3. P. J. Denning: "Virtual Memory", Computing Surveys,
Vol. 2, No. 3, pp. 153-189 (September 1970)
4. "HITAC 8700 機能説明書", 日立製作所 (1971)
5. "HITAC 8700 処理装置 (拡張モード)", 日立 (1971)
6. "HITAC 8700 管理プログラム概説", 日立 (1971)

(1) 整数演算



72

(2) 浮動小数点演算

