ALPSからの2，3の話題

井田昌之，森芳喜，河合繁，遠峰隆好（青山学院大学）

1．はじめに

青山学院大学の間野研究室で進められているLispマシンプロジェクトは，その一号機（ALPS／I）[1]が作動を始めて四年目を迎えることになった。我々の目標は価格／性能比志向のLisp専用機の可能性の追求とLisp自身及びLispに基づくシステムの移入という点にあった。マイクロプロセッサ関連技術の進歩と会話型分散処理への展望をその側面として持っていたことはいうまでもない。従ってALPS／Iの設計に当たっては実用性・高速性さらに安定稼動の要求があった。そこで我々のとった考え方は，「使えなければ良い理念も生きない」というものであった。このためALPS／Iは8ビットマイクロプロセッサを中心に，アクセス幅（語長）の異なる大容量メモリを接続し，アドレス情報の効果的な処理（16ビットでの）などにハード面の特殊性はとどめ，（我々の技量の範囲内で）確実な回路の制作に専念した。処理上の工夫はソフトウェアによって行なうこととし，保守性・信頼性・生産性・管理性の低下を防いだ。

この間いくつかの曲折を経ながらもReduceが動くようになるなど，一応の成果が得られるようになった。また今後の方向性についてのいくつかの基礎資料も得られたので，合わせてここに報告をしたい。

2．ALPS／Iの現状

現在のハードウェアはフロッピーディスクが作動を始めたので，それを中心としている。これによりアセンブラ・エディタなどとのファイルの共有が可能となり，初期ローディングに利用する以外は紙テープを用いずにプログラム開発・データファイルの作成ができるようになった。このための制御プログラムパッケージ（FDOS）については4章で述べる。

FDOSも含めて現在のシステムプログラムはすべて2716を用いてPROM化されており，14Kバイトを占めている。

ハードウェアの保守は一番重要な点であるが，なんとかやって来れたことは我々にとっては大きな自信となった。また，そのためのソフト側の工夫もいくつかなされるようになった。たとえばメモリの部分的なエラーがあってもLispを走らせることができるようにフリーストレージのフリーリストへの初期化はしないとか，RAMは最低2Kバイト生きていれば動くとかのこともされている。事実，インテル社の製作したRAMボードが動作不良になったこともある。その他についてはC2Cなどに記されている。

ソフトウェア面ではReduceが昨年一月に初版（サブセット）が動き，式の展開・微分等ができるようになった。[3]その結果を見ると，サブセット化しなくても動作することが予想されたのでオ＝版のそのままの形での移入に着手し，本年一月にほぼ作動させることができた。現在Procedure関係などに虫がいるが，鋭意デバック中である。このReduceについては5章で述べられる。

ただ第二版の制作に当たっては，制作効率を考え各種の関数の組込み・変更はexprレベルで行なわれた。これにより初版に比べて1.8倍程度時間がかかるシステムとなったが，二月末に三日程をかけてdefineされたもののうち効果がありそうでかつ記述しやすそうな約20の関数を手作業でオブジェクト化した。この結果初版と同等の速さに戻った。（Reduceのみの高効率化が目標ならば極力多くのdefineされた関数をsubr化するのが一番であろう。ただしそれが合理的な決定であるかについては疑問である。）

その際の効果を段階的に測定すると，

GTSのsubr化により15%程度（たとえば22秒かかったものが19秒に），

PTSのsubr化によりさらに10%程度（19秒 → 17秒），

MEMQのsubr化によりさらに10%程度（17秒 → 15秒），

GET，PUTのsubr化によりさらに20%程度（15秒 → 12秒）

などが特に有効であった。（数値については例によって若干の変動がある。）

orderpなどは組み込まれていたプログラムに虫がいたので第二版の制作においてはexprで作成している。orderpなどもsubrにすればさらに有効かもしれない。（いずれにしてもHLisp-Reduceでの苦労が我身になって体験にわかった気がする）

3. ALPS/Eでわかったこと

①16ビットアドレス空間でもかなりのことができそうである。ALPS/Eのアドレス空間は2つに分けられていた。Lispデータを保持するバルクアドレス空間と処理系用の内部空間である。これによりバルクメモリ（ワードアクセスされる64K語のメモリ）はすべてLispデータ保持用とすることができた。こうしたバンク分けによっても処理能力を増すことができる。こうした考え方では65536通りのLisp情報の識別が限界であることはいうまでもないが，16ビットのCPUの構成の容易さを考えるならば一考の余地がある。たとえばALPS/Eのハッシュ領域は二語一組で用いられ，2Kエントリ分用意されているが組にな、た一方の語を別バンクにおくことにより同一アドレス空間に4Kエントリを入れることができる。また，書き換えられないexprオブジェクトないしはそのコンパイルされたコードを別バンクにおくことによりフリーストレージに余裕をもたせることができる。スタックその他の作業域も別バンクにおくことによりハッシュ領域やフルワードスペースを広げることができる。

②やはり独立したGCプロセッサの存在は望ましい。会話型で小さな例題をくり返すときなど，今まで即座に答が返って来たのに突然応答がなくなりびっくりすることがある。ALPS/EのGCは約17秒程度かかるので，ばかにならない。会話型の専有処理形態の場合，入力及び出力の時間はCPU時間に比べてはるかに大きい。こうしたことからもパラレルGCあるいはシリアルリアルタイムGCの有効な組込み形態に対する示唆が得られている。あるいはキーイン中にGCを行なうことによりその後の演算時のGC作動回数を減らすこともできるのではないだろうか。またマルチバンク構成はメモリ共有型マルチプロセッサの実現にも向いていると思われる。

③Lispの仕様の標準化はされないものだろうか？ Reduceの移入での問題点はつきつめると仕様の違いに他ならない。Reduceのみを考えるならStandard Lispもよいが他のシステムの移入も考えるとどうであろうか？

④速度の改善については，平均20%を占めるバルクの高速化，CPUクロックの高速化，16ビット情報の比較・加工性能の付加により3倍程度にあげることはできそうである。bindingはやはりvalue-cellによるshallowの方がよさそうなので次期にはなおしたい。

4. フロッピーディスクの接続

最近，安価になってきたフロッピーディスクを現ALPS/Eに組み込み，ファイルシステムを開発することになった。

開発にあたっての前提条件と目標は

① 割り込みとDMA（ダイレクトメモリアクセス）はバルクメモリが専有しているため使用できない。空いているのは低速I/Oのみである。

② LISPインタープリタの作業域として使用されているメインメモリ上のRAMは，ファイルハンドラでは極力使用しない。

③ ソフトウェア（ハンドラ）は2Kバイト以下で作成する。これは，PROMモジュール1個で2Kバイトあるためである。

④ 速度は紙テープと比べて10倍以上，カセットMTと比べて同等以上とする。

⑤ 1つのメディアに多数のファイルを置けるようにする。しかし，1つのファイルを多数のメディアに分けて記録する必要はない。

⑥ ファイルはシーケンシャルとし，ハンドラは1バイト毎の入力と出力（紙テープと同じ）ができれば十分である。

ハードウェアの開発では，DMAが行なえないにもかかわらずディスクからのデータは32μ秒毎に転送されてくる。CPUは2MHzのSCMOSであり，500μ秒以下で1バイトのデータを転送することは不可能である。そこで，CPUとディスクの間にバッファメモリを設けることにした。ディスクの動作時にはバッファメモリとディスクの間でDMA転送を行ない，このDMA終了後にバッファメモリとCPUの低速I/Oの間でデータの転送を行なうことにした。このようにすることによってCPUでのプログラム（ハンドラ）なタイミングを考慮する必要がなくなった。

フロッピーディスクドライバの制御はたいへん煩雑であるが，最近，各社からFDC（フロッピーディスクコントローラ）と呼ばれる専用LSIが出ているのでこれを利用することにした。このLSIの機能はヘッドのロード，ステップモータの制御，プリアンブル，ポストアンブル，CRC等の直接関係のないデータの処理をおこなってくれる。そのため，インターフェイスではDMA転送回路とCPUの低速I/Oとのデータの転送のための回路のみを作ることが必要になった。

また，CPUからはバッファメモリのアドレスを指定するためのポインタへの値のセットやレコードの指定のコマンドが送られてくるが，このコマンドを簡単にするためと間違ったコマンドがきてもハードウェアを破壊しないようにコマンドコントローラを作成した。

バッファメモリの大きさはI/Oポートが8ビットであるため，1度の出力でアドレスを指定できる大きさとして256バイトにした。使用したFDCが1BMフォーマットであるため，1つのレコードは128バイトである。256

バイトはその2倍であるから十分である。

ハードウェアの特徴は

① CPUとは低速I/Oポート接続であり，入出力各2バイト（2ポート）である。データ，コマンド各1ポートづつ使用している。このため他のシステムへのポータビリティは大きい。

② TTL50個，LSI（FDCとメモリ）3個で構成されており1枚のボードに実装されている。

③ コマンドコントローラを作成したためCPUからのコマンドは簡単である。

④ ソフトが暴走してもハードウェアを破壊しない。

⑤ コマンドが充実しているのでソフトが小さく作れる。

ソフトウェアの開発では，ファイル名で各ファイルを呼び出すと，ハンドラが大きくなるので各ファイルはファイルナンバで呼び出すことにした。ファイルナンバはCPUが8ビットであるため0～255とした。ハンドラは以下の10個のサブルーチンから成っている。

① OPNR：入力ファイルのオープン
② OPNW：出力ファイルのオープン
③ GET：1バイトの読み出し
④ PUT：1バイトの書き込み
⑤ CLSR：入力ファイルのクローズ
⑥ CLSW：出力ファイルのクローズ
⑦ DLT：ファイルのデリート
⑧ INIT：メディアの初期化
⑨ GSP：空レコードの数を知る
⑩ GST：ファイルの存在を知る

入力ファイルと出力ファイルは同時に各1つだけオープンできるようにした。これは，バッファメモリが256バイトで2レコード分あるから各々を入力，出力ファイル用とした。このことによりデータの1バイト読み出し，書き込み時にはファイルナンバを指定する必要がなくなった。

この10個のサブルーチンがあれば呼び出す側のプログラムは紙テープからデータを入出力する場合と同様なプログラムでできる。

ディスク上でのデータ構造はファイル0～255と空レコードの先頭に必ずポインタが1つだけあり，その先にリンクで結合されたレコード（128バイト中125バイトがデータ）が並ぶようにした。

ハンドラの特徴は

① ハンドラの呼び出し方法は紙テープの場合と同様であり，簡単である。

② リンク構造にしたためデリートしたエリアの回収やライトファイルのオープン時のエリアの確保等のディスクの管理が非常に軽減された。

③ デリートしたファイル域は必ず回収される。

④ ハンドラ自身はPROM1.5Kバイト，RAM<u>7バイト</u>である。

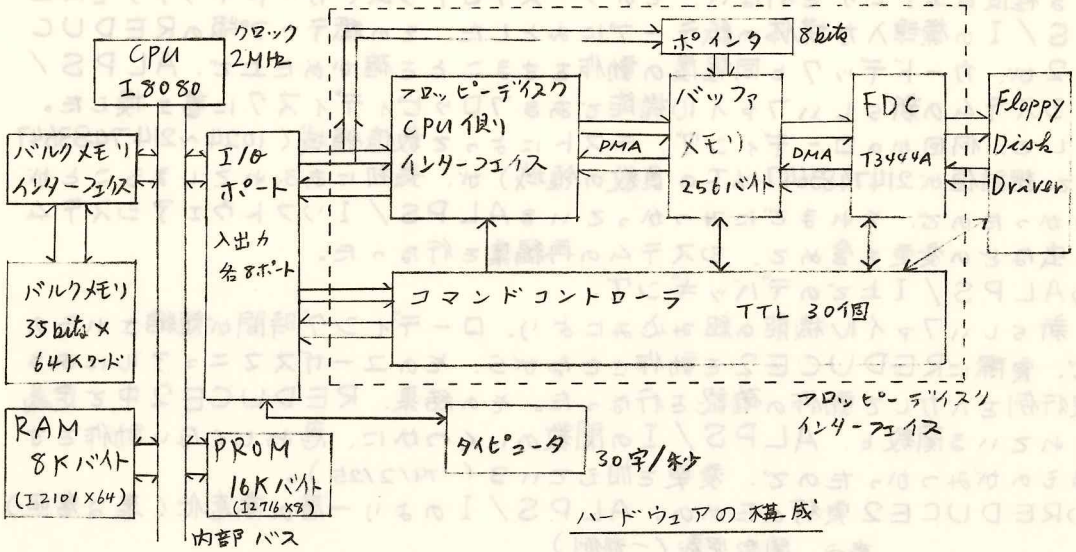
全体的な特徴は

① 1バイトの読み出しは平均1.4m秒，書き込みは2.8m秒である。

② Reduce（約18万字）のローディングは7分かかる。

③ メディアの取り扱いは非常に簡単になった。

このフロッピーディスクを中心とした構成図を以下に示す。



ハードウェアの構成

5. ALPS-Reduce第2版のインプリメント

ALPS/Iシステムにのせられた REDUCE2[4] は '75/8/12 (HLISP '76/1/6) 版で、昨年度にひきつづき、東大の寺島氏（現電通大勤務）よりいただいたソースプログラムを基にしている（残念ながら積分機能をもたない版である）。

組み込みは、以下にあげる手順によって行なわれた。

i) ユーティリティの作成

ソースプログラムの書き換え・解読のために用いた。

a) CRTG：atom（function, quote など）の利用箇所の洗い出しに用いた。（'78/9/12完成）なお、このプログラムのレキシカルスキャンはALPS/Iシステムのスキャンに合わせてある。

b) PAL-LEV-CHECKER：カードデックを媒体として、まとめて変更を加えていったために、不用意にカッコの対応をくずすおそれがあった。そこでこのカッコにレベル番号をふるプログラムを作成した。

c) PRETYPRINTER：昨年度のものの改訂版。（高速化）

ii) フラグ及びユーザプロパティの概念の、Hashed-arrayへの置き換え

i) a) を用いて洗い出されたフラグ、ユーザプロパティに用いられているatomをarray宣言し、それに伴ない、それらを扱う基本的な関数の定義を行なった。

iii) 関数の定義（表a参照）

a) ALPS/Iの新らしいファイル機能に関するもの：FDOS (9)

b) 仕様の違うもの：DIFF. (13)

c) 入出力に関するもの：IO (8)

d) 未定義であったもの：UNDEF. (69)

e) 上記・その他の補助的なもの：AID (14)　　以上 105

iv) ALPS/I へのテストローディング

今回の組み込みに際しては、昨年度とは異なり、クロスシステムによって、ある程度の虫?とりを行ない、そのソースプログラム(カードデック)をALPS/Iの標準入力媒体の紙テープにおとした。その紙テープ版のREDUCE2が、カードデックと同程度の動作をすることを確かめた上で、ALPS/Iシステムの新らしいファイル機能であるフロッピィディスクに書き換えた。そして、何回かのローディング、テストによって数値領域(1024~2147483647及び絶対値が2147483647以下の負数の領域)が、最初にあふれてしまうことがわかったので、それまでにみつかっているALPS/Iソフトウェアシステムの虫などの変更を含めて、システムの再編集を行なった。

v) ALPS/I上でのデバッギング

新らしいファイル機能の組み込みにより、ローディング時間が短縮されたので、実際にREDUCE2を動作させながら、そのユーザズマニュアルにある実行例を入力して動作の確認を行なった。その結果、REDUCE2中で定義されている関数と、ALPS/Iの関数のいくつかに、思わしくない動作をするものがみつかったので、変更を加えている('79/2/25)。

vi) REDUCE2実行のための、ALPS/Iのより一層の高速化(表a参照)

表a 関数定義(一部例)

assoc	UNDEF	define*	princ	I/O	synonim(prin1)
atsoc	〃	〃 *	prin1	〃	〃
a+	〃	synonim(sum)	prin1	〃	〃
a*	〃	〃 (times)	pts	UNDEF	define*
caaaar~cddddr	〃	define*	put	〃	〃 *
dopn	FDOS	subr	putd	〃	〃
dols	〃	〃	p+	〃	synonim(sum)
dget	〃	〃	p*	〃	〃 (times)
dput	〃	〃	p/	〃	〃 (quotient)
egcar	UNDEF	define*	read1	I/O	〃 (readch)
fixp	〃	synonim(numberp)	remflag	UNDEF	define
flag	〃	define	remprop	〃	〃
flagp	〃	〃	trace1on	AID	Reduce2's expr trace
gep	〃	〃 *	trace1off	〃	
get	〃	〃 *	trace2on	〃	ALPS1's expr trace
getd	DIFF	〃 *	trace2off	〃	
gts	UNDEF	〃 *	trace3on	〃	ALPS2's subr & subr trace
memg	〃	〃 *	trace3off	〃	
nzerop	〃	〃 *	u*mu	〃	ユーザプログラム

6. ALPS-REDUCE第2版の作動状況

ALPS-REDUCE第2版は、初版(展開・整頓・微分)に加え、次の機能が確認されている。(MATRIX及び負の入力(A-B)**2; に対しては調整中:'79/2/25)

i) SYMBOLIC モード
ii) 代入, 置き換え(:=, LET, DEF etc)
iii) くり返し演算(FOR文)
iv) ARRAY演算
v) on, off 機能
vi) COEFF

その他は確認中。

また、初版及び第2版の実行時間の推移を表bに示す。

表b. 初版及び第2版の実行時間比較

(x+y+z)^n の展開	初版	第2版	第2版改良版(表a中の*印をsubr化したもの)
n=2	17	26	17
n=6	130	217	149
n=7	195	284	213
n=10	670	1022	645

(単位:秒。入力後から印字終了まで。GC含む)

7. 数式処理用プログラム REDUCEの数理問題への利用

河合 繁

ALPS/I上にREDUCEを乗せることができ、数理問題解法の手助けとして使用することを考えてみた。 以下その外かくを述べてみる。

1) 待ち行列の解析への利用

REDUCEを使用するにあたっての、問題選択基準は、次のことを考えなければならなかった。1つには、メモリを大量に使用する計算、例えばマトリックス演算などは、容量を非常に多く使用するため、現在充分な演算をすることはできない。さらに、現在使用できるREDUCEの関数内での演算でなければならない、等の制約があるため、なるべく線形に近い式を処理するような問題から扱っていくことがこれからの使用に対して必要である。

待ち行列問題を選んだ理由としては、システムの微分方程式を解くにあたり、その状態が平衡状態であるならば、微分方程式を平衡方程式に置き換えて表現出来る点である。この平衡方程式を解く手法としては、ラプラス変換、母関数の利用などが上げられるが、現在のALPS/I REDUCEにおいての制約条件を考え、母関数を利用してこの平衡方程式を解いてみようと考えている。

待ち行列問題の本来の価値は、実際の問題が与えられ、それを解析してこそ意味が出てくるのであるが、今回はREDUCEの1つの利用という立場で、モデルを単純に分類して考えた。

2) 待ち行列問題の概括

一般に待ち行列の問題は、あるシステムにおける客の母集団および扱者、到着、およびサービス時間などの確率法則ならびに行列の規律を規定して、客の滞りや流れの様相を記述する数学モデルである。ここで、扱者は、サービス窓口であり、客とは扱者のサービスを受けようとする個々の要求のことである。特に窓口が複数になると、数学的解析はたいへん困難となり、シミュレーションにたよらざるを得なくなる。このような問題を解析的に解くためには、非常に多くの計算が要求されるため、なかなか思うにまかせない問題がある。

以下、待ち行列の代表的基本モデルを述べていく。

3) 待ち行列の各種のモデル

(1). 行列に制限のない単一窓口 M/M/1 (∞) 型

このモデルは、客の到着がポアソン分布となり、窓口におけるサービス分布が、指数分布となるシステムである。サービスできる窓口は1コである。以上のことを、ケンドールの記号を用いて書くと、M/M/1 (∞) と表す。また客の数(行列の長さ)は無限とする。

以上の条件のもと、上のモデルを解析してみる。

$P_n(t)$: 時刻 t で系内に n 人いる確率

$P_n(t+\Delta t)$: 時刻 $(t+\Delta t)$ に n 人いる確率 とする。

$n \geq 1$ の場合には、以下の状態確率が成り立つ

$$P_n(t+\Delta t) = P_{n-1}(t)\cdot\lambda\Delta t\cdot(1-\mu\Delta t) + P_n(t)\cdot(1-\lambda\Delta t)(1-\mu\Delta t) + P_{n+1}(t)\cdot\mu\Delta t(1-\lambda\Delta t)$$

上式において、 λ : 到着率 μ : サービス率

$n = 0$

$P_0(t+\Delta t) = P_0(t)\cdot(1-\lambda\Delta t) + P_1(t)\cdot\mu\Delta t\cdot(1-\lambda\Delta t)$ となる

以上の2式より、以下の微分方程式が成り立つ

$$\frac{dP_0(t)}{dt} = -\lambda P_0(t) + \mu P_1(t) \qquad (n=0)$$

$$\frac{dP_n(t)}{dt} = \lambda P_{n-1}(t) - (\lambda+\mu) P_n(t) + \mu P_{n+1}(t) \qquad (n \geq 1)$$

上式を直接解くと、計算量が非常に多くなるが、このモデルでは、平衡状態つまり、$t \to \infty$ を考えているので、$\lim_{t\to\infty} P_n(t) = P_n$ が成り立つ。ゆえに上式の微分方程式は以下のように書くことが出来る。

$$\begin{cases} -\lambda P_0 + \mu P_1 = 0 \\ \lambda P_{n-1} - (\lambda+\mu) P_n + \mu P_{n+1} = 0 \end{cases}$$

上式を $\rho = \lambda/\mu$ （利用率）で置き換える。

$$\begin{cases} -\rho P_0 + P_1 = 0 & \text{—(i)} \\ \rho P_{n-1} - (1+\rho) P_n + P_{n+1} = 0 & \text{—(ii)} \end{cases}$$

ここでさらに、$\sum_{n=0}^{\infty} P_n = 1$ という正則条件が成り立つ、これより、$P_n = \rho^n P_0$

$P_0 = 1-\rho$ が求まり、$P_n = \rho^n (1-\rho)$ が求まる。

以上のことを、母関数を使用して解いてみる。母関数の使用は系が多少複雑になると、よく利用する方法であり、REDUCEには向いている性質を持っている。

母関数 $F(z) = \sum_{n=0}^{\infty} P_n z^n$ とおき、上式(ii)の両辺に z^n を掛けて、$n=1$ から ∞ まで総和する。

$$\rho z \sum_{n=1}^{\infty} P_{n-1} z^{n-1} - (1+\rho) \sum_{n=1}^{\infty} P_n z^n + \frac{1}{z} \sum_{n=1}^{\infty} P_{n+1} z^{n+1} = 0$$

$$\rho z F(z) - (1+\rho)(F(z) - P_0) + \frac{1}{z}(F(z) - P_1 z - P_0) = 0$$

上式を $F(z)$ について解く。

$F(z) = P_0/(1-\rho z)$　　$F(1) = \sum_{n=0}^{\infty} P_n = 1$ なので、$P_0/(1-\rho) = 1$

ゆえに $F(z) = (1-\rho)/(1-\rho z)$ となる。

この母関数が求まると、系の平均人数 L を求めることができる。

$$L = \frac{dF(z)}{dz}\Big|_{z=1} = \rho/(1-\rho)$$

次に列人数 L_q の母関数を同様に求めてみると、

$Q(z) = (1-\rho)\{\rho^2 z/(1-\rho z) + (1+\rho)\}$ となり

$L_q = \frac{dQ(z)}{dz}\Big|_{z=1} = \rho^2/(1-\rho)$ として求めることが出来る。

以上のように、直接微分方程式を解かずに、母関数を用いて解くと、系平均人数、列平均人数さらに分散も容易に求めることが可能となる。系、列平均待ち時間は省略。

(2). (1)の他に、行列に制限のある単一窓口 M/M/1 (N) 型、窓口が S 個の M/M/S (∞) 型、M/M/S (N) 型があるが、ここでは省略する。

4) あとがき

現在のところ、単一窓口型の平衡方程式を解くのが、メモリ、ALPS/I REDUCEの使用可能な関数等を考え合せると、限度のようである。一次方程式を解くためには、マトリックスを用いれば容易なのであるが、現在のREDUCEにはこの演算を充分に実行することができないため困難である。

このような平衡方程式の解法のような例では、数値演算がほとんどないため、数式処理を行なうメリットが大きいようである。

8. ALPS－Reduceの実行例

以下に第2版での実行例を示す。初期化・フロッピーからのローディングをおえるのに現在約9分を要している。ローディング時間についてはコアイメージのsave，restoreにより半分程度に減らすことができるので現在製作中である。（なお，紙テープからの印字を含むローディングでは2時間，印字なしでも1時間を要する。また，以前のデジタルカセットからのローディングでは約17分を要した。）

所要メモリ量は，現在無駄な部分をはいっているので縮少化を進めているが，フリーストレージ56Kセルのうち46Kセル程を要している。ハッシュ領域は現在800H個のエントリーが可能であるが，Reduceの制御下にはいった時点で6EBH個を使用している。従って約300_{10}個の自由H分子があることになる。またreduceがロードされた時点でpname文字列あふれ領域が約1260語を占め，Fullword領域が500個程度に圧迫されている。

実行例
（下線部は使用者の入力を示す。添付するために若干のつめ合わせが行なわれている。）

```
END OF EVALQUOTE, VALUE IS..
REDUCE INITIALIZATION HAS BEEN COMPLETED!

EVALQUOTE ENTERED, ARGUMENTS...


END OF EVALQUOTE, VALUE IS..
NIL

EVALQUOTE ENTERED, ARGUMENTS...
BEGIN(()())
REDUCE 2 (JAN-6-76) ....
LET X = C+B+A;
X**3;
 3        2         2           2              2     3       2         2     3
A  + 3*A *B + 3*A *C + 3*A*B  + 6*A*B*C + 3*A*C  + B  + 3*B *C + 3*B*C  + C

DF(/*ANS,A,2);
6*(A + B + C)

DF(LOG(SIN(Y)),Y);


COS(Y)/SIN(Y)

ARRAY A(5);


FOR N:=1:3 DO WRITE A(N) := X**N;
A(1) := A + B + C

            2                      2            2
A(2) := A  + 2*A*B + 2*A*C + B  + 2*B*C + C

          3       2         2           2              2     3       2          2
A(3) := A  + 3*A *B + 3*A *C + 3*A*B  + 6*A*B*C + 3*A*C  + B  + 3*B *C + 3*B*C

          3
      + C
SYMBOLIC IF "TOMMY" EQ "CHEESE" THEN 'HAPPY ELSE 'SORRY;
SORRY


DF(A(3),C,2);
6*(A + B + C)
```

```
ARRAY XX(6);
COEFF((Y+Z)**6,Y,XX);

6

XX(6);
1

XX(5);
6*Z

XX(4);
    2
15*Z

XX(3);
    3
20*Z

XX(2);
    4
15*Z

XX(1);
   5
6*Z
```

7章の話題からの簡単な例

M/M/1 (の)

```
D:=DF((1-V)/(1-V*W),W);
                        2  2
D := (V*( - V + 1))/(V *W  - 2*V*W + 1)

W:=1;
W := 1
D;
                   2
(V*( - V + 1))/(V  - 2*V + 1)

ON GCD;
0()
D;
( - V)/(V - 1)
```

系平均人数の母関数

value に出ているらしい(?)

系平均人数

9. ALPS/II の製作へ向かって

筆者らの意向としてはALPS/Iで採用したマイクロプロセッサ関連技術の利用とその延長が，我々の力量の範囲内で確実に作動するシステムとするためには，とられるべき道だと思っている。ただ，スタックまわりやLisp向きの演算機構の設置などの特殊化や，メモリーオーバーラップやfetch-a-headなどは部分的に入れられるであろう。Lisp情報に対するアドレス空間の拡大については，情報の幅が対応して全て広くなるので具体的にはきめかねている。

システム記述言語に対する考慮は事前に充分にしておきたい。[5] アセンブラによるコーディングは，個人的にはモジュラーにも書け，それ程不自由はしなかったがやはり問題が残る。Lispのdata Objectはタイプレスである。従ってタイプのある言語を使って記述するのは高価になる可能性がある（その言語の開発に対して）。またfunctionalな記述や，Lispに似た変数の束縛あるいはLispよりもっと単純な束縛（例えば完全なグローバルと完全なローカルしか存在しないとか）形式をもつ言語を利用できるなら色々と楽しみが多い。BCPL系のB言語はかなり扱いやすいと思っている。CになるとLispを書くにはもったいない。Bのサンプルは[6]に示しておいたが，ALPS/IIはそのアーキテクチャを確定しその上でBのコンパイラを作ることからはじめたいと思っている。

おわりに ： 総花的な記述になってしまったがALPSをとりまく現状とその将来について記した。紙面の都合で部分的には予定の原稿も省くことになった。わかりにくい点があれば席上で補いたい。最後に日頃ご指導を頂きます経営工学科・間野浩太郎教授に感謝いたします。

参考文献　［1］井田，間野：マイクロプロセッサを用いたLispマシンALPS/I；情報処理論文誌Vol20. No.2 pp113~121(March 1979)；［2］井田：Lispマシン製作奮戦記；bit Vol.10 No.14及びNo15.(1978)，［3］井田他：ALPS－Reduceのインプリメンテーション；情処19回大会，［4］A.C.Hearn：Reduce 2 Users Manual(1973)［5］井田，中田：基本ソフトウェアの記述ツール；情報処理1979年6月号掲載予定，［6］井田：システム記述言語の記述性；ソフトウェア工学シンポジウム報告集pp75－86(1979)