記 号 処 理　19－4
（１９８２．５．２０）

ＡＬＰＳ／IIのフリーストレージ処理機構

佐藤　衛・井田　昌之・間野　浩太郎
（青山学院大学理工学部）

０．　はじめに

ＡＬＰＳ／IIは、マイクロプログラム制御方式のプロセサ Lisp Function Unit（ＬＦＵ）、フリーストレージを格納するための　バルクメモリ、　および既製のマイクロプロセサシステムを用いたフロントエンドシステムの３要素から構成される Lisp 専用システムである。　ＬＦＵとフロントエンドシステムのハードウェアについてはすでに発表した。（文献１．２）

ＬＦＵは設計時、１命令３００nsec で動作させる予定であったところ、先の発表（文献２）まではノイズ等により、６００nsec でしか安定に動いていなかった。　このノイズは、マザーバスを単線で配線していたためにひろってしまっていたことがわかり、マザーバスをすべてツイステッドペア線で配線しなおすことにより、現在、１命令３００nsec でハードウェアは安定して働いている。したがって、現在では　Tarai－n の実行所要時間は、図－１に示すとおりのものとなっている。　くわしくは、文献２を参照されたい。

n	（ハンド）コンパイラ版）	最速版	阪大ＥＶＡＬII（100nsec）
3	４．２msec	１．４msec	— msec
4	７８　msec	２６　msec	４４msec
5	２１４６　msec	６９５　msec	１２０９msec

図－１　TARAI n　の実行所要時間　クロックは300nsec
（阪大ＥＶＡＬIIのデータは記号処理１７－１の発表による）

ＡＬＰＳ／IIでは、実用的な問題の処理を考えているので、安価なダイナミックフリップフロップ型ＩＣメモリ（ＤＲＡＭ素子）を用いたメガセル級のバルクメモリをフリーストレージのために用いる事を基本構想としている。ＤＲＡＭ素子の速度は最近向上してきているが、スタティックフリップフロップ型ＩＣメモリ（ＳＲＡＭ素子）やＴＴＬ回路素子等と比較すると、やはり一段階低速である。この低速なＤＲＡＭ素子（サイクルタイム４５０ｎＳ）をフリーストレージ用バルクメモリとして使っても、ＡＬＰＳ／IIシステム全体の機能を低下させないようにした点に、ＡＬＰＳ／IIのフリーストレージアクセス機構の特徴がある。

そのために、

ｉ）バルクメモリアクセス中にも、ＬＦＵのＡＬＵ部、スタック部、制御部が並行して動作できるようにし、２つのレディ信号によってアクセス完了を知るようにする。

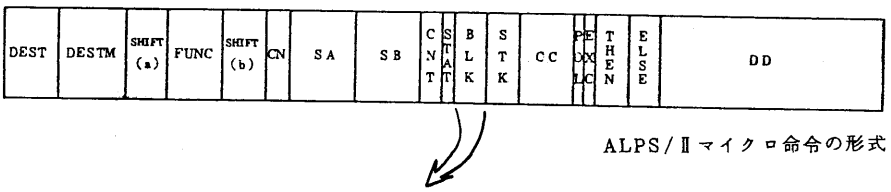
ii）データバッファ、アドレスバッファを各々２個ずつ用意し、既読出しデータの処理中にバルクメモリアクセスを並行して行わせたり、バルクメモリアクセス中に次に書き込むデータを用意したりすることを可能にする。

の２つの機構が組み込まれた。

本文では、このバルクメモリのアクセスに採用した基本操作について解説し、いくつかの例によって上記設計方針の効果を示す。

1。　バルクメモリの基本操作

バルクメモリは既製のメモリボード（東光製ＭＭ２４７）を用いて構成されている。　このボードのアクセスタイムは３００ｎ、サイクルタイムは４５０ｎであり、ＬＦＵの２クロックで１回の　read/write　が可能となっている。　バルクメモリインタフェースに対しては、マイクロコード中に３ｂｉｔの命令フィールドを設けている。（図－２）



ALPS/Ⅱマイクロ命令の形式

マイクロコードBULKフィールド	意味
0XX	NOP
100	BULK READ1　（バッファ１に対する命令）
101	BULK WRITE1　（バッファ１に対する命令）
110	BULK READ2　（バッファ２に対する命令）
111	BULK WRITE2　（バッファ２に対する命令）

図－２　バルクインタフェース命令

この命令フィールドで　ＮＯＰ以外のコードが指定されると、その次のＬＦＵのサイクルから２サイクルを使ってバルクメモリのバルクバッファへの読み書きが実行される。　このとき、ＬＦＵはバルクインタフェースとは独立に処理を進めることができる。

バルクインタフェースが busy 中に新らしいバルク命令が出されるのを防ぐためと、バルクバッファ中のデータが安定したか否かをチェックするために、２つのバルクステータスフラグ

ＢＣＲＤＹ　・・・　バルクコマンドレディ
（バルクインタフェースにコマンドを出すことができる）

ＢＤＲＤＹ　・・・　バルクデータレディ
（バルクバッファ中のデータが安定である）

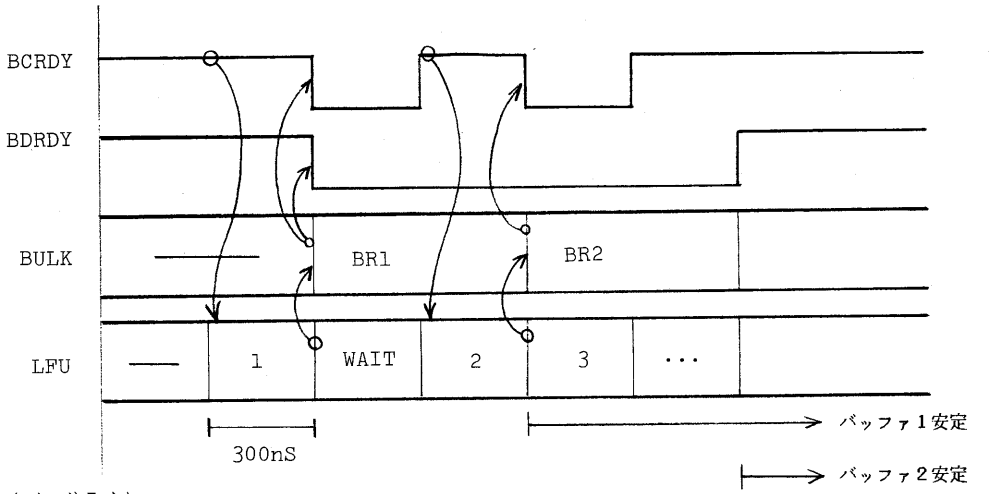
を用意し、マイクロプログラムで制御を行なっている。　例として、読み出しと書き込みについてのマイクロコードとタイミングを図－３に示す。

バルクメモリリード

マイクロ命令

```
1   if bcrdy then bulk=br1 step else wait;
2   if bcrdy then bulk=br2 step else wait;
3   .....
```

タイミング

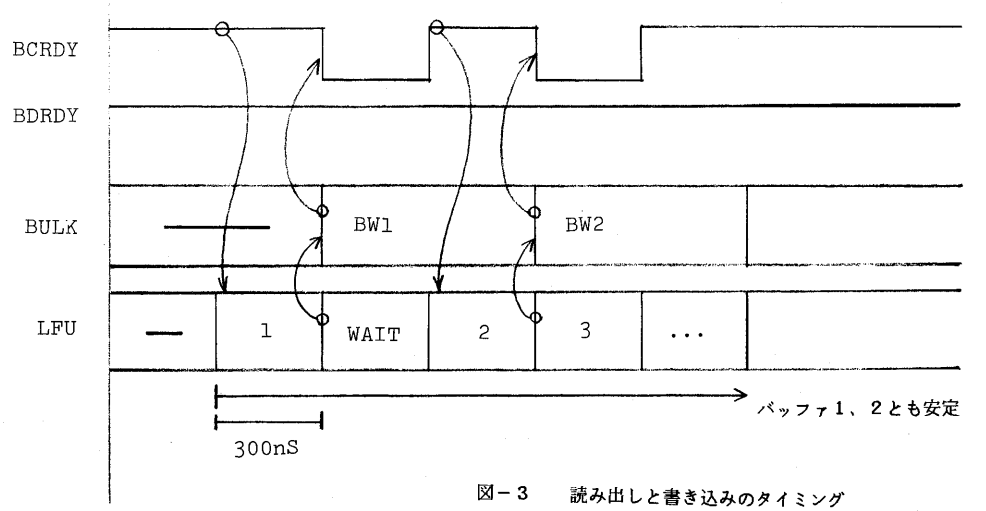


バルクメモリライト

マイクロ命令

```
1   if bcrdy then bulk=bw1 else wait;
2   if bcrdy then bulk=bw2 else wait;
3   .....
```

タイミング



図－3　読み出しと書き込みのタイミング

２。　Ｌｉｓｐ基本操作１．ＣＡＲ．ＣＯＮＳ．ＥＱ

Lisp の基本操作のうち、ＣＡＲ．ＣＯＮＳ．ＥＱを例にとって、ＡＬＰＳ／IIでどのようにマイクロプログラム化されているかを示す。

２。１。　ＣＡＲ

ＣＡＲのコーディングを図－４に示す。　以下、図－４に記された番号にしたがって説明をしていく。

１　ＣＡＲへのエントリ。　第１引数のタイプがステータスとなってわたってくるので、引数がアトムの場合、エラー処理ルーチンへの jump も同時になされている。

２　バルクアドレスレジスタへ引数の値をロードし、BULK READ を起動。

３　バルクデータバッファが安定するのを待って函数の値をセットし、　return　する。

```
1   if not.atom then dest=badrl func=sb sb=ar step else jmp=errat;
2   if bcrdy then bulk=brl step else wait;
3   if bdrdy then destm=valr func=sb sb=bcarl jmpcr else wait;
```

図－４　函数ＣＡＲのマイクロプログラム

ここでは、バルク　メモリからのデータを待って次の処理が進むので、ＬＦＵのidle time は２クロックとなっている。　ＣＤＲについても同様である。このような　idle　time　のできる函数については、上位の函数中でマイクロコードに展開して最適化をはかる必要がある。そうすることによって　idle　time　を０にできる例を後にあげる。

２。２。　ＣＯＮＳ

ＣＯＮＳ　のコーディングを図－５に示す。　現在、ＣＯＮＳ操作は次のような方法で行なわれている。

１　バッファ１を使用して、FREEL レジスタの内容で示されている空きセルの先頭を読み込む。（Free list は、CDR chain になっている）

２　引数から、ＣＡＲ部、ＣＤＲ部を構成し、FREEL レジスタの内容で示されている空きセルにバッファ２を使用して書き込む。

３　１で読み出したセルのＣＤＲ部の内容を FREEL　レジスタにロードして、次のＣＯＮＳのための空きセルの番地を準備しておく。もし、次のセルがなければＧ．Ｃ．へ飛ぶ。

図－５にしたがって、

1　(FREEL)をバルクアドレスバッファ１にロードし、BULK READ を起動。
2　(FREEL)をバルクアドレスバッファ２にロード。
3　ＣＡＲ部をバルクデータバッファ２にロード。
4　ＣＤＲ部をバルクデータバッファ２にロード。
5　ＦＬＡＧ部をバルクデータバッファ２にロード。(BULK WRITE を起動)
6　FREEL にバルクデータバッファ１のＣＤＲ部をロードして、次のＣＯＮＳに備える。　もし、それがＡＴＯＭであればＧ．Ｃ．へ飛び、さもなければ return 。

```
1   if bcrdy then dest=badrl func=sb sb=freel bulk=brl step else wait;
2   dest=badr2 func=sb sb=freel;
3   dest=bcar2 func=sb sb=ar;
4   dest=bcdr2 func=sb sb=br;
5   if bcrdy then dest=bflr2 func=sb sb=dd dd=list bulk=bw2 step else wait;
6   destm=freel func=sb sb=bcdrl stat=new;
7   if not.atom then destm=valr func=sb sb=badrl jmpcr else jmp=gc;
```

図－５　函数ＣＯＮＳのマイクロプログラム

ここでは、バルク読み出し中に書き込みデータの準備をすることになるので、ＬＦＵの　idle time　は０になる。　これは、２バッファにした効果ということができる。

２．３．　ＥＱ

ＥＱ　のコーディングを図－６に示す。

1　第１引数と第２引数を比較（ＸＯＲ）し、フラグをセットする。
2　結果が０であればＴを値として return する。
3　その他の場合はＮＩＬを値として return する。

```
1   func=xor sa=ar sb=br stat=new;
2   if z then destm=valr func=sb sb=tr jmpcr else step;
3   destm=valr func=sb sb=nilr jmpcr;
```

図－６　函数ＥＱのマイクロプログラム

この場合は、ＴとＮＩＬを０とそれ以外とに置き換えれば、２または３ステップが１ステップになるので、他の函数との整合を洗い出して、その置換が可能かどうかを検討していくつもりである。

３。　Ｌｉｓｐ基本操作２．ＥＱＵＡＬ．ＡＰＰＥＮＤ．

２。で示された recursive な定義の特に必要のないものをもとに recursive な定義ができるとコーディングが楽になると思われる。　２つの例、ＥＱＵＡＬ、ＡＰＰＥＮＤについて、ＡＬＰＳ／IIでどのようにマイクロプログラム化するかを示す。

３。１．　ＥＱＵＡＬの定義を図－７．１に、ＥＱＵＡＬのコーディングを図－７．２に示す。

```
equal[x;y] →
    [atom[x] → eq[x;y];
     atom[y] → NIL
     T → prog[[xx;yy:stack];
              xx:= cdr[x];
              yy:= cdr[y];
              [equal[car[x];car[y]]
               → return[equal[xx;yy]];
               T → return[nil]]]]
```

図－７。１　函数ＥＱＵＡＬの定義

```
#ENTRY EQUAL;
 FUNC=SB SB=BR STAT=NEW IF ATOM THEN JMP=EQ ELSE STEP;
 IF ATOM THEN DESTM=VALR FUNC=SB SB=NILR JMPCR ELSE STEP;
 IF BCRDY THEN DEST=BADR1 FUNC=SB SB=AR BULK=BR1 STEP ELSE WAIT;
 SB=FR STK=PUSH;
 IF BCRDY THEN DEST=BADR2 FUNC=SB SB=BR BULK=BR2 STEP ELSE WAIT;
 SB=AMPCR STK=PUSH;
 IF BCRDY  THEN SB=BCDR1 STK=PUSH;
 DESTM=AR FUNC=SB SB=BCAR1 STAT=NEW;
 IF BDRDY THEN SB=BCDR2 STK=PUSH;
 DEST=FR FUNC=SB SB=TOSP;
 DESTM=BR FUNC=SB SB=BCAR2 CALL=EQUAL;
 FUNC=XOR SA=VALR SB=TR STAT=NEW;
 IF Z THEN DESTM=BR STK=POP STEP ELSE JMP=EQUAL01;
 DESTM=AR STAT=NEW STK=POP CALL=EQUAL;
EQUAL01:
 DEST=TOSP FUNC=ADD SA=DD SB=FR DD=2;
 DEST=AMPCR STK=POP;
 DEST=FR STK=POP JMPCR;
#END;
```

図－７。２　函数ＥＱＵＡＬのマイクロプログラム

３．２．ＡＰＰＥＮＤの定義とコーディングを図－８に示す。

```
append[x;y] ⟶
   [atom[x] ⟶ y;
    T ⟶ prog[[xx:stack;yy:register];
              xx:= car[x];
              yy:=append[cdr[x];y];
              return[cons[xx;yy]]]]
```

```
#ENTRY APPEND;
 IF ATOM THEN DESTM=VALR FUNC=SB SB=BR JMPCR ELSE STEP;
 IF BCRDY THEN DEST=BADR1 FUNC=SB SB=AR BULK=BR1 ELSE WAIT;
 SB=FR STK=PUSH;
 SB=AMPCR STK=PUSH;
 IF BDRDY THEN SB=BCAR1 STK=PUSH ELSE WAIT;
 DEST=FR FUNC=SB SB=TOSP;
 DESTM=AR FUNC=SB SB=BCDR1 STAT=NEW CALL=APPEND;
 DESTM=BR FUNC=SB SB=VALR;
 DESTM=AR STAT=NEW STK=POP;
 DEST=TOSP FUNC=ADD SA=DD SB=FR DD=1;
 DEST=AMPCR STK=POP;
 DEST=FR STK=POP JMP=CONS;
#END;
```

図－８　函数ＡＰＰＥＮＤの定義とマイクロプログラム

このＥＱＵＡＬとＡＰＰＥＮＤでは、前述したＣＡＲ・ＣＤＲでの idle timeを０にするように工夫することができる。すなわち、ＣＡＲ・ＣＤＲを外部函数とみなしてサブルーティンジャンプせずに直接マイクロコードに展開し、メモリからのデータ待ちの間にスタックフレームを作成するなどの最適化をすることが可能である。そのとき、２つのバッファを用意した効果が最大限に発揮できるであろう。

３．３．　実行例

図－９．１のようなＬＩＳＴどうしのＥＱＵＡＬ、ＡＰＰＥＮＤの所要総ステップ数と所要時間を計測すると、図－９．２の結果を得る。

(A B C D E F G H I J)

図－９．１　評価に使用したＬＩＳＴ

	所要総ステップ	所要時間（μsec）
EQUAL	200	60
APPEND	190	57

図－９．２　ＥＱＵＡＬ・ＡＰＰＥＮＤの所要総ステップと時間

4。まとめ

ＡＬＰＳ／IIで採用した２つの機構　i）バルクメモリアクセス中にＬＦＵが動作できること、　ii）２つのバッファを用意したこと、　によって、フリーストレージ格納のためのバルクメモリに低速なＤＲＡＭ素子を用いても、ＬＦＵでの処理を妨げず、全体のスピードアップをはかることができる。しかしながら、ＣＡＲ・ＣＤＲ等のように、バルクメモリアクセスのためにＬＦＵの動作にidle time ができてしまう場合がある。これらについては文中でも述べたように、上位の函数の中でマイクロコードに展開し、最適化することで idle time をなくすことができる場合があるので、そのような最適化を検討したい。

謝辞

細部設計などで多大な御助言と御協力をいただいた卒業生の森芳喜氏・小林茂男氏（現インフォメーションアンドコントロール研究所）、配線などを快く手伝ってくれた間野研究室などの学生諸兄に感謝致します。

文献

１）佐藤・井田・間野：「会話型人工知能専用機ＡＬＰＳ／IIのLisp処理機構」記号処理１７－２　記号処理研究会資料１７　1982, Jan

２）佐藤・井田・間野：「ＡＬＰＳ／IIの機能とその設計」理化学研究所シンポジウム　1982, Mar

３）間野：「マイクロプロセサを用いた人工知能・数式処理専用システム」昭和５６年度科学研究費補助金（一般研究Ｂ）研究成果報告書　1982, Mar

４）Ida,M & Mano,K:"An Adaptable Lisp Machine Based on microprocessors." proc. of Int'l Micro & Mini Computer conference at Houston p210-p215 1979, Nov

５）前川・土井・西川・斉藤・安井：「試作ＥＶＬＩＳマシンのＥＶＡＬIIと開発支援機能」　記号処理１７－１　記号処理研究会資料１７　1982, Jan