# 会話型人工知能専用機ＡＬＰＳ／IIのＬＩＳＰ処理機構

青山学院大学理工学部
佐藤衛・井田昌之・間野浩太郎

## Ｉ．　はじめに

数式処理で必要とされる会話型のＬＩＳＰマシンとして、すでに、ＡＬＰＳ／Ｉを作製したが、より実用的な数式処理を目ざしてＡＬＰＳ／IIをつくった。ＡＬＰＳ／IIでは、ＲＥＤＵＣＥ２を使用して当学内での研究の手助けができる規模の専用機を目ざした。また、ハードウェアが我々だけで短日月に作製・保守ができるように、インタリービングなどメモリアクセスの高速化が必要になるＣＤＲcodingを採用せず、従来のビット幅の広いセルを採用した。したがって、より大容量のメモリと処理の高速性と（プログラム開発とハードウェア故障診断に役立つ）デバッグ機能を与える事が目標となった。

１．メモリは、ＤＲＡＭを使って２Ｍバイト（５１２ｋセル）までの実装が可能であること。

２．高速性のための機構。ビットスライス型のマイクロプロセサチップを利用し、８０ビット幅のマイクロコードを用いることによってかなり大幅な水平処理を進めたこと。特に、ｉ）ＡＬＵ，ハードウェアスタック，バルクメモリインターフェイスが同時に並行して動作でき、ii）ポップ・プッシュ・絶対アドレス指定・ポインタアドレス指定をそれぞれ１マイクロコードで実現することのできるハードウェアスタックを持ち、iii)マイクロコードの中に、ある仕事を実行しながら他のある条件の成立するまで待つ後続指定制御の機能をもっていること。或る種のデータタイプチェックをハードウェア化したこと。

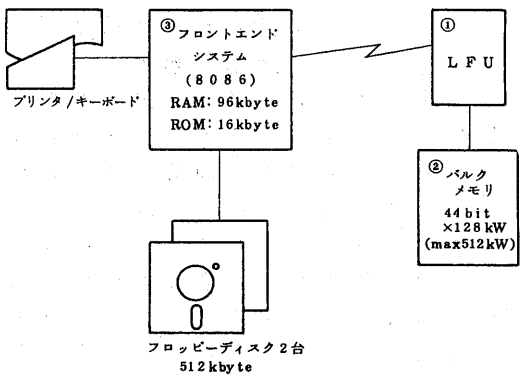
３．デバッグ機能。マイクロコードで任意にセット・リセットできるシステム用フラグをつけたこと。フロントエンドプロセサによる監視機能。

等に特徴がある。

## II．　ＡＬＰＳ／IIのハードウェアと機能の概要

### II．１．　システム構成

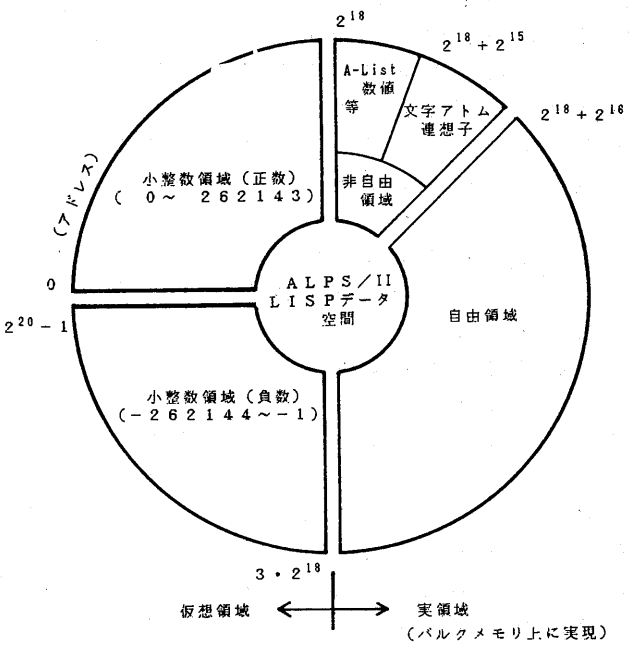
図－１にＡＬＰＳ／IIの全体構成図を示す。全体は、ＬＩＳＰ処理の中心となるＬＦＵ（Lisp Fuction Unit）、ＬＩＳＰデータを格納するバルクメモリとフロントエンドプロセサ（ｉ8086）から構成されている。フロントエンドプロセサは、システム始動時のＬＦＵの初期化と、Ｉ／Ｏ処理の一部（特にファイル操作）を担当する。以下では、ＬＩＳＰデータ空間とバルクメモリ、ＬＦＵの構成とその機能、特に、高速化の為に付け加えた機能、デバッグの為の機能について解説し、フロントエンドプロセサの役割についても解説する。




図－1. ALPS／Ⅱシステム概略図

## II．２．　ＬＩＳＰデータ空間とバルクメモリ

ＬＩＳＰデータ空間は、ＡＬＰＳ／Ｉでは、６４Ｋセルを実装していたが、ＡＬＰＳ／Ｉ上にＲＥＤＵＣＥ２をインプリメントした際の経験と、実装技術上と予算上の可能性から、ＣＯＮＳセルとしては５００Ｋセル程度が妥当であると考えた。ＡＬＰＳ／IIでは、ビットスライス型マイクロプロセサチップを使用するＬＦＵの処理幅が４の倍数で設定できることから、１Ｍセル（２０bit アドレス）とした。ＡＬＰＳ／IIのＬＩＳＰデータ空間は、スモールインテジャのための仮想的な空間とＣＯＮＳセル、文字アトムやアソシエータのための実空間とに分けられる（図－２）。スモールインテジャのための空間をデータ空間の始めと終りに置いた（それぞれ正数と負数に対応）ことで、スモールインテジャの取り扱いが容易になる（絶対値の小さな数値が多く使われる）。実空間はバルクメモリ（４４bits＊１２８Kwords）上に実現されている。実空間上には、大きく分けて３つのタイプのデータを格納するが、このタイプ分けは、ハードウェアでチェックするようにした。



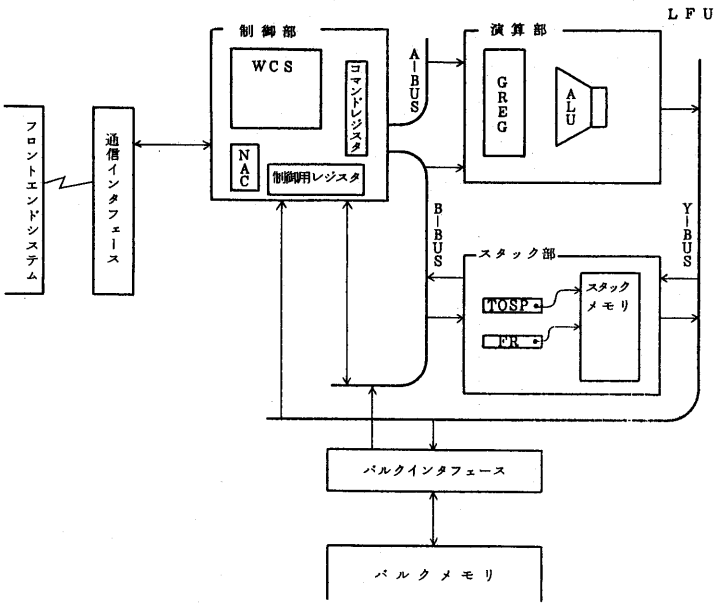

図－２　ＡＬＰＳ／II　ＬＩＳＰデータ空間

## II．３．　ＬＦＵの構成と機能

### II．３．１．　構成の概要

ＬＦＵは、II．２．でのべたように処理幅２０bitsの、リスト処理を高速に実行するためのプロセサである。ＬＦＵは、１語８０bitsのマイクロ命令を持つマイクロプログラム制御方式のプロセサで、制御部、演算部、スタック部、メモリインタフェース部から成っている（図－３）。ＡＬＰＳ／IIでは、ＬＩＳＰインタプリタのみならず、例えばＲＥＤＵＣＥ２の中で特に高速化することが全体の高速化につながるようなFunctionはマイクロプログラム化することを考えている。このとき、マイクロプログラム領域を大きくとること、マイクロプログラムの上でサブルーチンのネストが何重にも可能であること、スタックに関しては局所変数の束縛が簡単に行なえることが重要となってくる。これらはまた、インタ

プリタのマイクロプログラム化のときにもその作製を容易にするはずである。そしてまた、リスト処理ではメモリのリード／ライトが頻繁に行われるが、ＬＦＵと較べて低速なメモリを用いることになるので、メモリ待ちの時間に他の処理（ハッシュ探索のときに、次のリハッシュ番地を求めたり、イニシャライズやガベッジコレクション時に次のセル番地の準備をするなど）が可能であることも処理の高速化に寄与すると考える。演算部、スタック部、メモリインタフェース部の、並列動作が可能であることの他、これらは以下で述べるように実現した。




図－３　Lispファンクションユニット概略図

## ＩＩ．３．２　制御部

制御部は後続指定制御部とフロントエンド制御・通信機構とからなっている。後続指定制御部は図－４で示す構成になっている。大きなマイクロプログラム領域のため、ＷＣＳは最大３２Ｋ語まで実装できる。何重にも、サブルーチンのネストが可能である様にするため既製のシーケンサは用いず自作した。呼ばれた側で再度サブルーチンを呼び出すとき、戻りの情報を保存すればよいので、呼び出すときには戻り情報はレジスタにセーブするだけでよい。また、メモリ待ちの間にも演算部・スタック部の動作が可能であり、かつ必要であれば１命令でメモリ待ちができるようにするため、比較的低速なバルクメモリからのデータの待ち合わせもマイクロ命令で行ない、条件がFALSE であるときに演算部、スタック部及びメモリインタフェース部の動作を抑制することを可能とした。後続指定は、条件部指定と以下の８つの機能

i)　Step　　（次の番地＝μＰＣの値）
ii) Jmp　　（次の番地＝即値）
iii)Jmpb　　（次の番地＝バス信号値）
iv) Jmpcr　　（次の番地＝ ampcrレジスタの内容）
v)　Call　　（次の番地＝即値、μＰＣの値をampcr へセーブ）
vi) Callb　　（次の番地＝バス信号値、μＰＣの値をampcr へセーブ）
vii)Wait　　（次の番地＝現在の番地）
iix)Halt　　（次の番地＝μＰＣの値，ただし再スタート後）

の中の１つをおのおの選択することができる（マイクロ命令中の）THEN部、ELSE部の指定によって決められるようにしたので、上記のサブルーチンのネストやデータの待ち合わせ等の機能が達成できる。

フロントエンド制御・通信機構については後述する。

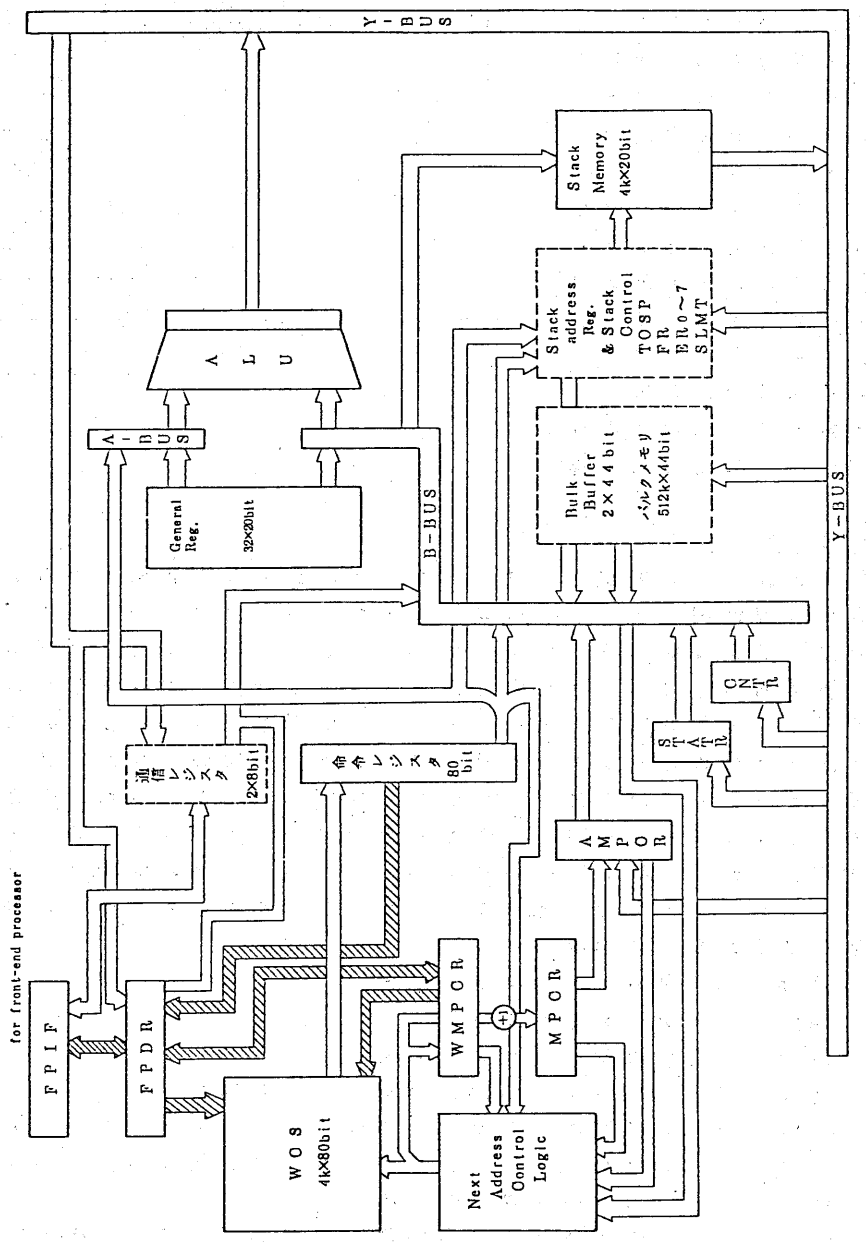



図-4 LFUの構成（制御部中心）

## II．３．３．　演算部

演算部にはＡＭＤ社の４bitsスライスチップAm2903を５つ使用した。主な機能としては、ADD,SUBTRACT,OR,AND,XOR などがある。ＬＩＳＰデータの比較などはこの演算部で行なう。関数引数、値など演算部の汎用レジスタに納めることとし、頻繁に使われる特殊な値――NIL、Ｔ、－１、０、１やFree-List、A-Listへのポインタ等――を納め、マイクロコードの即値欄をジャンプ先指定のために残すように汎用レジスタを３２個に拡張した（表－１）。これは外付けの定数レジスタとして作る場合より実装が大幅に楽になるので採用した。

| アドレス | 用途・ニモニック | アドレス | 用途・ニモニック |
|---|---|---|---|
| ００ | ワ　Ｒ０ | １０ | 函数の値　VALR |
| ０１ | ー　Ｒ１ | １１ | 第１引数　AR |
| ０２ | キ　Ｒ２ | １２ | 第２引数　BR |
| ０３ | ン　Ｒ３ | １３ | 第３引数　CR |
| ０４ | グレ　Ｒ４ | １４ | 第４引数　DR |
| ０５ | ・ジ　Ｒ５ | １５ | それ以上の引数へのポインタ　MR |
| ０６ | ス　Ｒ６ | １６ | 予備　NR |
| ０７ | タ　Ｒ７ | １７ | ４０００Ｈ（即値）　16K |
| ０８ | 通 | １８ | フリーセルへのポインタ　FLNR |
| ０９ | 信 | １９ | フリーセルへのポインタ　FREE |
| ０Ａ | ・ | １Ａ | A-Listへのポインタ　ALIST |
| ０Ｂ | シ | １Ｂ | －１（即値）　－１ |
| ０Ｃ | スレ | １Ｃ | ０（即値）　０ |
| ０Ｄ | テジ | １Ｄ | １（即値）　１ |
| ０Ｅ | ムス | １Ｅ | Ｔ（ポインタ）　TR |
| ０Ｆ | 用タ | １Ｆ | ＮＩＬ（ポインタ）　NILR |

表-1 ALU内部レジスタの割り付け
定数レジスタにはイニシャライズ時に値をロードする。

## II．３．４．　スタック部

スタック部は、２０bits＊４Kwordsのメモリと数個のポインタレジスタ、スタック制御レジスタおよびコントローラからなる（図－６）。スタックの機能としては、ポップ・プッシュの他に、絶対アドレス指定・ポインタ相対アドレス指定を可能とすることで、局所変数の取り及いが簡単となる。イニシャライズやデバッグの時にスタックポインタやマイクロコードを書き換えることなく、可能とするため、スタック制御レジスタ（ＳＣＲ）をもうけ、これに適当な命令語をロードすることによって動的なアドレシングが可能となっている。（図－９）

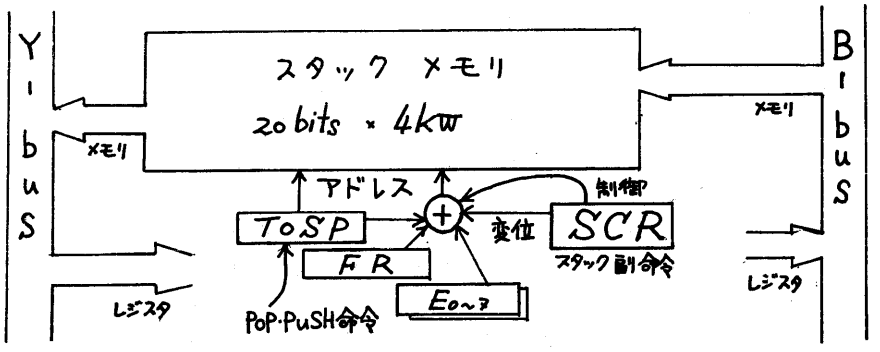



図-6
ALPS/II
ハードウェア
スタック
の構造

## II．３．５．　メモリインタフェース部

メモリインタフェース部にはポインタの付け換えに便利なように２つのバッファがある。おのおののバッファはアドレス部、データ部からなっており、データ部はさらにＣＡＲ欄，ＣＤＲ欄およびＦＬＡＧ欄を抽出することができる。文字

アトム、アソシエータの取り扱いの為にＣＡＲ欄はＡＴＴＲ欄、ＶＡＲＩＡＮＴ欄にわけて参照する機能も用意している（図－７）。

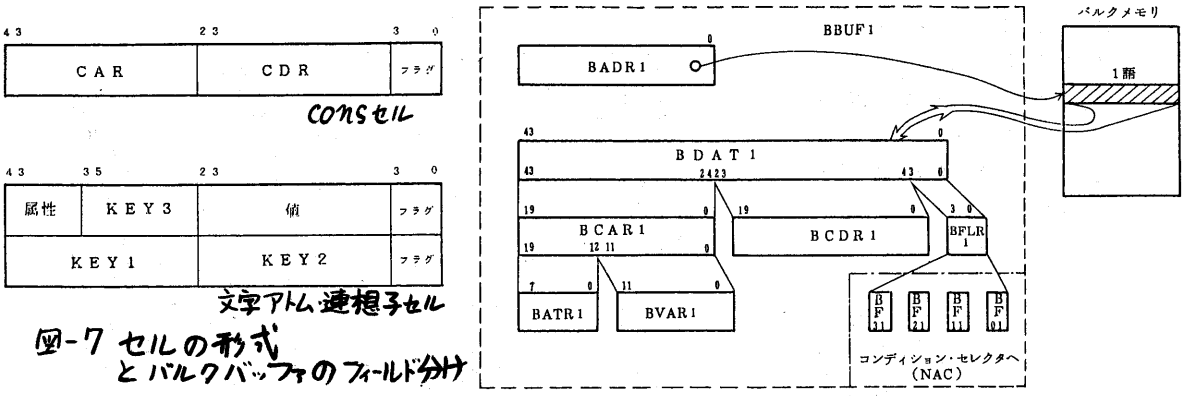



図-7 セルの形式
とバルクバッファのフィールド分け

遅いバルクメモリのアクセスの間に、可能であれば他の処理をするために、メモリインタフェースのビジー／レディーは、マイクロ命令でテストする。メモリインタフェースのレディー信号には以下の２種を用意した。

Ｉ）ＢＣＲＤＹ（バルクコマンドレディ：コマンド受付可）

II）ＢＤＲＤＹ（バルクデータレディ　：バッファアクセス可）

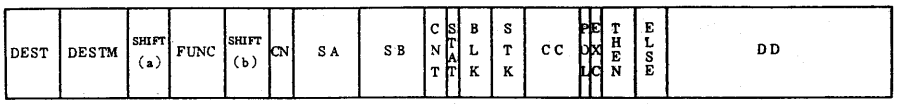
書き込みの時にはＢＣＲＤＹをテストするだけで次々にバルクライト命令を出すことができる。読み込みの後にもＢＣＲＤＹをテストすれば次のバルク命令を出せる。ＢＤＲＤＹは、バルクリード命令の後で（指定された方の）バッファのデータの保証ができるようになった後、真になる。ＢＤＲＤＹの方がＢＣＲＤＹより遅れて真になるので、レディ信号を上記の２つにわけることで、バルクメモリに対する連続した操作が必要な時にはＬＦＵの空き時間を減らせる。

## III．　マイクロ命令

以上の機能を取り扱うため、マイクロ命令は特にタイプ分けせず、その語長は80ビットと長くとった。20ビットの即値欄をもうけたためと、Ｂ．１に述べた並列性を実現するために、あえて短くすることは避けた。命令のフォーマットは図－８のとおりである。

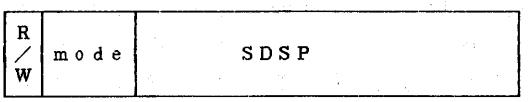
常時マイクロ命令で指定できるスタックに対する命令はポップとプッシュしかない。このままでは、スタックメモリの内容のダンプやコンパイラ化した際に使われるフレーミングのためにはいちいちスタックポインタを書き換えてやる必要があり、スタック操作を高速に実行できない。そこで、スタックに対しては副命令をもうける。副命令では、絶対アドレシング、ポインタ相対アドレシング等を指定できるようにし、マイクロ命令で指定するほか、動的なアドレシングができる（ダンプの時にはコマンドで指定された番地からカウンタをインクリメントしながら読んでくる）ようにスタック部にはＳＣＲ（スタック制御レジスタ）をもうけて、ＳＣＲに副命令をロードしても、スタックに対する特殊な命令を実行できるようにする。副命令はi)ＳＣＲにロードして使用することも多い（レジスタの初期設定は、主に即値を代入する）ii）マイクロ命令中に専用の欄を作るとさらにマイクロ命令の語長が長くなる、iii)そのわりにはそれほど頻繁には使わな

い、という理由でマイクロ命令の即値欄におく。ポップ・プッシュ以外の機能はそれほど頻繁に使わないのでこの副命令を使って実行する（図－９）。




| フィールド名称 | 長さ | 指定する内容 |
|---|---|---|
| DEST＊ | 5 | Y-BUSに対してDestinationとなるGREG以外のREGISTERの指定を行なう。 |
| DESTM＊ | 6 | Y-BUSに対してDestinationとなるGREGの指定を行なう。 |
| SHIFT＊ (a),(b) | 8 | ALU出力のSHIFTの方向等を指定する。 |
| FUNC＊ | 5 | ALU（５×Am 2903）の命令$I_{4-0}$に対応する。ADD,SUB,AND,OR等，算術，論理演算を指定する。 |
| CN＊ | 2 | ALUのCarry-inの指定，０，１，CR（STATR内）等を指定する。 |
| SA＊ | 6 | A-BUSに対するSOURCEとなるGREG，即値を指定する。 |
| SB＊ | 6 | B-BUSに対するSOURCEとなるREGISTER（GREGを含む）を指定する。 |
| CNT＊ | 2 | CNTRのup／down count,diseble を指定する。 |
| STAT＊ | 1 | STATRを更新するか否かを指定する。 |
| BLK＊ | 3 | バルクインタフェースコマンドを指定する。（BR1,BW2,NOP等） |
| STK＊ | 3 | スタックコントローラに対する命令を指定する。（PUSH,POP,DD等） |
| CC | 5 | 選択するCondition の指定。 |
| POL | 1 | CCで選択されたCondition の内容のNOTをとるか否かの指定 |
| EXC | 1 | ＊印の付けられた部分の命令を，「いつでも実行する」か，「POL ⊕ Condition = TRUEのときのみ実行する」か，を指定する。 |
| THEN | 3 | NEXT Address control logic に対する指定。POL ⊕ Condition かTRUE のとき選択される。 |
| ELSE | 3 | NEXT Address control logic に対する指定。POL ⊕ Condition がFALSE のとき選択される。 |
| DD | 20 | 即値データを指定する。SA,SB,STK,THEN及びELSE 部の内容によってデータの用途は定義される。 |

図－５　ALPS/Ⅱマイクロ命令の形式




R／W　0：read，1：write.

| mode | 実効アドレス |
|---|---|
| 0 | |
| 1 | 禁止 |
| 2 | |
| 3 | (SDSP)　絶対アドレス |
| 4 | (FR) + (SDSP) |
| 5 | (TOSP) + (SDSP) |
| 6 | (TOSP) + (SDSP) * |
| 7 | (TOSP) + (SDSP) ** |
| 8 | (E0) + (SDSP) |
| ・ | ・ |
| ・ | ・ |
| F | (E7) + (SDSP) |

図－9
スタック副命令の形式

＊ TOSPインクリメント後アドレス計算
＊＊アドレス計算後TOSPディクリメント

## IV．　デバッグ機能

ＡＬＰＳ／IIでは、２つのデバッグ機能

１．　ハードウェア故障、１マイクロ命令ごとの診断。
２．　ＬＩＳＰ関数レベルでの診断。

を備えることを考えた。１．については、フロントエンドプロセサがＬＦＵの１マイクロ命令だけの動作を指定できることとＬＦＵの内部レジスタに間接にアクセスすることができることで達成する（その他バルクメモリ、スタックメモリ、ＷＣＳのパリティチェック、スタックリミットオーバー等の監視もフロントエンドプロセサが行なう　　）。２．については、ＡＬＰＳ／Ｉで採用したＷＡＬＫ、ＴＲＡＣＥの機能を受けつぐ（ＷＡＬＫ：函数ＥＶＡＬ、ＥＶＬＩＳの引数・値の印刷を行なう、ＴＲＡＣＥ：指定された函数の引数・値の印刷を行なう）。これらの機能が使われているかどうかはインタプリタ（ＥＶＡＬ、ＡＰＰＬＹ、ＥＶＬＩＳ）で頻繁にチェックされる。このチェックが他に影響をおよぼさずに、高速にできるようにすれば、処理の高速化が可能である。ＡＬＰＳ／IIでは、ＬＦＵのフラグ（後続処理指定部に対する条件としてマイクロ命令の中にかけるもの）レジスタにマイクロ命令で定義できるものをもうけ、それによって一種の割込みとしてＷＡＬＫ、ＴＲＡＣＥのための分岐を行う（図－１０）。

```
#ENTRY WALKON;
 dest=stat func=or sa=dd sb=stat  dd=walkonsw;
 destm=valr func=sb sb=nilr jmpcr;
```

セット

```
#ENTRY WALKOFF;
 dest=stat func=and sa=dd sb=stat dd=walkoffsw;
 destm=valr func=sb sb=nilr jmpcr;
```

リセット

```
if walk then ..... jmp=evwalk else step;
```

テスト

図-10 WALKフラグ セット・リセットの函数（WALKON, WALKOFF）とWALKフラグのテスト

## V．　フロントエンド制御機構、フロントエンドプロセサ

ＡＬＰＳ／IIでは、ある程度実用になる比較的安価なＬＩＳＰ専用機を短日月で開発したかった。そこで、フロッピーディスクを外部記憶として持ち、入出力装置としてインクジェットプリンタを使用する、モニタプログラムやアセンブラ・トレーサ等の既に開発されているi8086をフロントエンドプロセサとして採用し、ファイルの操作や入出力の制御プログラム作成の手間を省いた。また、これによって、ＬＦＵの開発時のトラブルシューティングも容易に行なえるようにした（外部記憶に２０Ｍバイトのハードディスク装置も予定している、すでに装置は購入済）。

フロントエンドプロセサは、フロントエンド制御機構を介して、

１．　ＷＣＳへの書き込み。
２．　ＬＦＵの起動、停止。
３．　ＬＦＵの状態（メモリパリティエラー等）の監視、エラー時復帰作業。
４．　Ｉ／Ｏ処理の１部。

を行なう。フロントエンド制御機構は、ＬＦＵへの割込み・状態の保存が可能で

あるように、ＬＦＵの中のレジスタ、メモリはすべて間接にアクセスできるようにした（図－１２）。これによって処理途中のＬＦＵのダンプ、デバッグモードの切り換えも可能で、デバッグの大きな手助けとなる。

```
dest=fpdr func=sa sa=ar if true then halt else halt;
                                    レジスタarをフロントエンドへ

dest=fpdr stk=dd dd="00011xxxxxxxxxxxxxxx"b if true then halt else halt;
                                    メモリ（×××番地）をフロントエンドへ
```

図-12 LFU内部のレジスタ・メモリの内容のフロントエンドデータレジスタ(FPDR)へのロード

## VI．　ＬＦＵの評価－－ＬＩＳＰ函数のコーディング

ＡＬＰＳ／IIでは、マイクロプログラムのレベルでサブルーチンのネストが可能である。これによって、再帰的定義ができれば簡潔に記述できるＬＩＳＰ函数のコーディングが大変楽になる。図－１３は『ｅｑｕａｌ』をＡＬＰＳ／IIのマイクロアセンブラで記述した例である。この例では、ｅｑｕａｌの再帰的定義が生かされており、コメント行（枠取された部分）中の定義とよく対応したコーディングとなっている。第１行は外部参照名の宣言、第２行から第７行はコメント、第８行から第２１行までが函数定義の本体である（第１１行は継続行）。ＡＬＰＳ／IIでは、２引数のＬＩＳＰ函数は、レジスタＡＲに第１実引数（枠内ｘ）、ＢＲに第２実引数（枠内ｙ）をのせ、第１実引数のタイプをステータスレジスタにのせてコールされる。第８行で第１引数のタイプによって函数ＥＱへの分岐がおきる。第１引数と第２引数のタイプがちがうと第９行で値ＮＩＬが函数の値レジスタＶＡＬＲにセットされてリターンする。第１０行以降で引数のＣＡＲどおしがｅｑｕａｌかどうかＥＱＵＡＬをコールし、ＣＡＲどおしがｅｑｕａｌであれば、ＣＤＲどおしのｅｑｕａｌを値としてリターンするため、ＥＱＵＡＬへジャンプする（第２１行）さもなければそのまま（ＮＩＬを値にもったまま）リターンする。後続処理指定の機能（次の実行番地の選択がＴＨＥＮ部、ＥＬＳＥ部で指定できる機能）も多く使われている（ｅｑｕａｌの場合全体の６／１３）。マイクロアセンブラは現在大型計算機上に作製されており、マイクロプログラム作製の平易さは、追加予定のマクロ機能等によって向上すると思っている。


## 謝辞

ＡＬＰＳ／Ｉの開発に協力した、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）に今回も多大な御助言と御協力をいただいた。感謝します。また、配線などを快く手伝ってくれた間野研その他の学生諸兄に感謝します。

```
 1:  #ENTRY EQUAL;
 2:   %.....................................................................
 3:    .  equal[x;y] = [atom[x]--> eq[x;y];                                .
 4:    .                  atom[y]--> NIL;                                  .
 5:    .                 T--> [equal[car[X];car[y]]--> equal[cdr[x];cdr[y]]; .
 6:    .                       T--> NIL]]                                  .
 7:    .....................................................................%
 8:     if not.atom then func=sb sb=br stat=new step else jmp=eq;
 9:     if atom then destm=valr func=sb sb=nilr jmper else step;
10:     if bcrdy then dest=badr1 func=sa sa=ar sb=amper
11:                              blk=br1 stk=push step else wait;
12:     dest=badr2 func=sb sb=br;
13:     if bdrdy then sb=bcdr1 blk=br2 stk=push step else wait;
14:     destm=ar func=sb sb=bcar1;
15:     if bdrdy then sb=bcdr2 stk=push step else wait;
16:     destm=br func=sb bcar2 call=equal;
17:     destm=br stk=pop;
18:     destm=ar stk=pop;
19:     dest=amper stk=pop;
20:     func=sub sa=valr sb=tr stat=new;
21:     if z then func=sb sb=ar stat=new jmp=equal else jmper;
```

図-13 函数equalのコーディング例


文献

1)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.

2)佐藤、井田 & 間野：「Ｌｉｓｐマシン ＡＬＰＳ／IIの機能とその設計」青山コンピュータ・サイエンス 第９巻 第２号 Ｐ１７～Ｐ４９ 1981, Dec.

3)井田、森、河合、遠峯：「ＡＬＰＳからの２、３の話題」記号処理７－１ 記号処理研究会資料７ 1979, Mar.