計算機アーキテクチャ32－3
（1978. 9. 13）

LISP マシンの 試作

瀧 和男，金田悠紀夫，前川禎男（神戸大学工学部）

1．まえがき

人工知能の研究や記号処理の発達に伴い、これらの処理用言語である LISP言語には、種種の改良が加えられ、多くのすぐれた処理系が誕生してきた。[1][2]

そこでは、使い易さとともに、実行速度の向上と、大規模なプログラムを効率良く動作させることに対して努力が払われてきたが、LISP言語特有の動的記憶領域の割り付け、関数の再帰呼出し、リスト処理などに対して、従来から存在する汎用計算機のハードウェア構造が不適当であることが指摘されるようになった。こうして、LISP言語の高速処理に、より適した構造を持つ LISP マシンの研究が進められ、いくつかの研究成果が上げられている。[3]-[7] そして LISP マシンの傾向としては、インタープリタのマイクロプログラム化、ハードウェアスタックの実装、ビット処理機能の充実などが見られる。

われわれの LISP マシンは、インタープリタのマイクロプログラム化と市販の LSI を使用することを前提として、まずインタープリタの基本設計を先行させ、その中で必要とされる機能を洗い出し、高速化に役立つ部分をハードウェア化する設計方針をとった。そして規模は研究室レベルで実現できる程度におさえ、そのかわり早期完成をめざすこととした。特にビットスライス ALU、マイクロプログラムシーケンサ LSI の使用は、ハードウェア量を低減させ、またハードウェアスタック、フィールド／ビット処理機能とともに、マイクロプログラムレベルでの再帰呼出しを許したことにより、高速化を推し進めている。

2．LISP マシンシステムの構成

このたび開発した LISP マシンシステムのハードウェア構成図を、図1に示す。すなわち、LISP プログラム処理の中心となって働く、プロセッサモジュールとメモリモジュールを、市販の LSI ミニコン（dec社の LSI-11）のバスに接続した構成である。ミニコンからはメモリモジュールへページモードでアクセスでき、またプロセッサモジュールの CMR（コマンドレジスタ）や WCS（writable control storage）に対しても同様にアクセス可能である。ハードウェア構成上は、ミニコンがマスター CPU であり、独自の主記憶を持ち、LISP プロセッサと並行して動作できる。メモリモジュールは、2つの CPU に対して共有メモリであり、CMR はミニコンに対しては、I/O デバイスである。

ミニコンの仕事は2つあり、1つは始動時や保守時にマスター CPU として、LISP プロセッサのプログラムロードやメモリ、レジスタの初期化を行なう。もう1つは、実行時にソフトウェア上のスレーブ CPU として、LISP プロセッサからの割り込みを受け、入出力プログラムの一部と、入出力機器の制御を担当する。そのほか、時間監視や、外部からの緊急要求の受け付けも行なう。また図1に示すとおり、ミニコンは並列入出力インターフェースを通して、マイクロコンピュータと中型計算機システム（FACOM230-38）に接続されており、各種 I/O デバイ

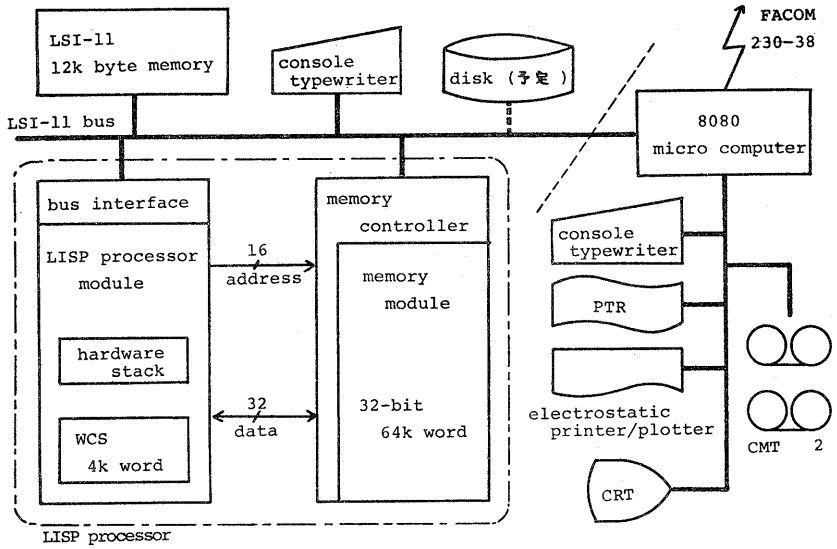


図1. LISPマシンシステムのハードウェア構成

スが利用できる。

メモリモジュールは、32ビット×64k語の構成で、高位ミニコン程度の容量があり、プロセッサモジュールは、処理巾が16ビット、アドレス空間も16ビット、メモリとのデータ受け渡しが、32ビット巾で行なわれる。即ち、メモリ1語中に、car部とcdr部を16ビットずつ持ち、同時に読み書きを行なう。また書きこみは、バイト単位でも可能である。

3. LISPプロセッサのハードウェア

3.1 LISPプロセッサモジュールの概要

LISPプログラムの高速処理に必要とされるハードウェア機能としては、

a. スタック操作の容易性、高速性　b. フィールド処理、ビット処理

c. 条件ジャンプの多様性

などが上げられてきたが、またこれらは、高級言語計算機のエミュレーター一般に、要求される機能ともいわれている。[3][4][8][9]

われわれのLISPマシンは、市販のビットスライスALUと、マイクロプログラムシーケンサLSIを用いることを前提に、汎用エミュレータをめざすのではなく、インタープリタオリエンテッドなLISPシステムの構成を目的とした。

インタープリタからの要求により、上記の3つの機能を含む、図2のハードウェア構成をとった。即ち、高速かつ高機能のハードウェアスタック、フィールド抽出回路、ビットアドレッシング回路、マイクロプログラムレベルでのマルチウェイジャンプ、間接ジャンプ機能、豊富な条件テスト回路、メモリの番地による利用種別を3ビットのコードに変換するマッピングメモリ回路、等である。また、マイクロプログラム用のWCSは4k語を装備し、インタープリタ全体のマイクロプログラム化を行なう。ハードウェアスタックは、16ビット4k語の固定長である。

内部バス構成は、Aバス、Bバス、Yバスの各16ビットの3バス構成である。Aバスがデータのソースバスであり、Yバスがデスティネーションバス、またBバスには、マイクロプログラム中の定数が与えられる。ALUでの演算は、

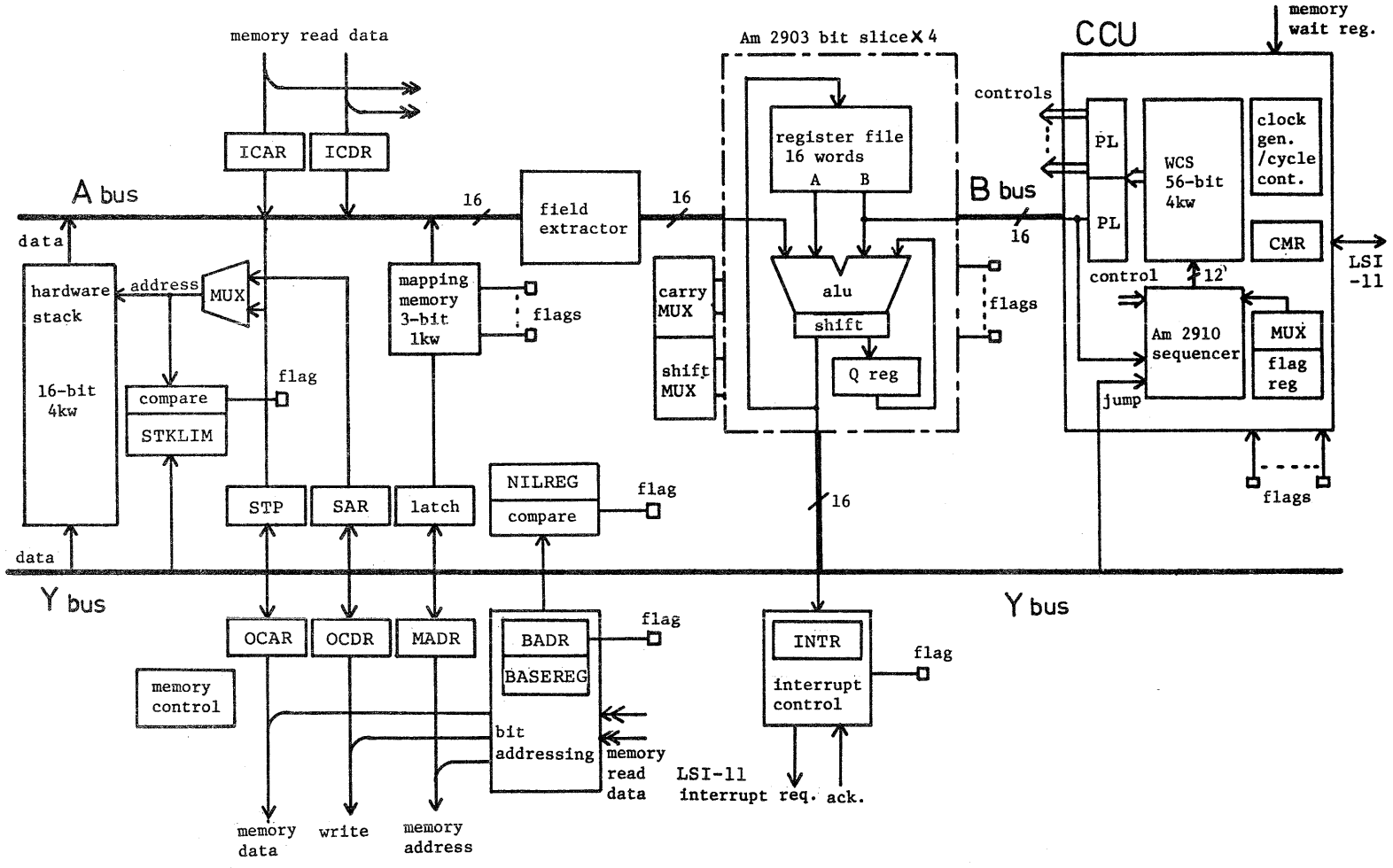


図2. プロセッサモジュールのハードウェア構成

i) ALU中のレジスタファイルどうし　ii) レジスタファイルと定数
iii) Aバスデータとレジスタファイル　iv) Aバスデータと定数
との間で、それぞれ可能である。また、マイクロプログラムシーケンサはYバスとも結ばれていて、演算結果の番地へジャンプができる。このことにより、マルチウェイジャンプや、スタックに保存しておいた番地へのリターンが許され、マイクロプログラムレベルでの再帰呼出しを可能にしている。また、ALUでの演算結果や、ビットアドレッシング回路からの信号は、フラグレジスタを介して、マイクロプログラムシーケンサに結ばれ、条件ジャンプに利用される。

このLISPマシンは、再帰を含むすべての演算を、マイクロプログラムで記述し得るため、マシンコードを取り扱うためのハードウェアを特に準備していない。しかしながら、WCSの容量や、コンパイラとの関係からマシンコードを必要とすることもあり、その実現方法については後述する。

3.2 ビットスライスALU

ビットスライスALUは、Advanced Micro Devices 社の4ビットスライスである、Am2903 [11]を、4個使用した。その特徴は、i) チップ上に16語のレジスタファイルを持ち、任意の2つをデータソースとして独立に指定できる、ii) ALUに対するソースデータの2つを、ともに外部から与えることができる（Aバスと Bバス）、等である。ii)の機能は、ALUをレジスタファイルと切り離して利用できるために便利であり、Aバスソースとマイクロプログラム上の定数との間の演算に利用している。最小サイクルタイムは、100n sec 程度である。

zero, negative, overflow, carry のフラグ用の出力端子を備えており、条件ジャンプのテスト条件に用いている。

3.3 コンピュータコントロールユニット CCU

マイクロプログラム制御による計算機の制御部をCCUと呼んでおり、Am2910 マイクロプログラムシーケンサ[10]、WCS、PL（パイプラインレジスタ）、CMR、クロックジェネレータ／サイクルコントローラ、フラグレジスタから成り立っている。1レベルのパイプライン制御を行なっており、ALUの実行中につぎのマイクロ命令の読出しを並行して行なう。

Am2910 は、12ビット4096語のWCSを指定でき、μ-PC（マイクロプログラムカウンタ）の他に、サブルーチン用の5段のスタック（ハードウェアスタックとは別）と、ループカウンタを持っている。マイクロ命令上の4ビットの指定によりオペレーションが決められるが、主なものはつぎのとおりである。

continue, conditional jump subroutine, conditional return, repeat while counter≠0, test end loop, conditional jump register/PL, etc.

WCSは、1語56ビットを4096語実装しており、素子はアクセスタイム150n secの4kビット MOSメモリである。

クロックジェネレータはシステムクロックを発生し、サイクルコントローラは、マイクロ命令やメモリからのwait要求により、クロックジェネレータに働きかけて、マイクロ命令の実行周期を変化させる。

CMRは、LSI-11から読み書きできる4ビットのレジスタである。run/halt, initializeの各ビットにより、LISP プロセッサの実行、停止、初期化の制御を行ない、また attention 0, 1 の各ビットはフラグレジスタに結ばれていて、LSI-11からLISP プロセッサに対して、何らかの要求のあることを示す。

フラグレジスタは次のフラグを含み、すべて条件ジャンプに利用できる。

ALU関係フラグ　：zero, negative, overflow, carry, S-shiftout, Q-shiftout
マッピングメモリ関係フラグ　：list, literal atom, number, atom
その他のフラグ　：nil, bit test, stack overflow, iack, attention 0,1

3.4 ハードウェアスタック

70n sec の高速メモリによる、4k語の固定長スタックである。スタックへのアクセスは、2つのスタックポインタレジスタ STP（スタックトップポインタ）とSAR（スタックアドレスレジスタ）により、SARは主に、スタック内部へのアクセスに用いる。いずれもカウンタタイプのレジスタで、スタックをソースとしたときは間接後自動減少、デスティネーションとしたときには自動増加後間接のモードを有する。これらにより、ALUとSTP, SAR間のデータ転送の回数を減らすことができる。またSTPとSARを利用して、スタックからスタックへのデータ転送を1マイクロサイクルで完了できる。LISPインタープリタにおけるframeの作成とリターンのとき、またSUBR関数中での引数の移動の際などに、スタックからスタックへのデータ転送は多く、マイクロプログラムのステップ数を減少させることができる。

STKLIM（スタックリミットレジスタ）にはあらかじめ値をセットしておき、この値を越えてスタックが伸びたときにはフラグが立って、スタックオーバフローを簡単に調べることができる。

3.5 フィールド抽出とマッピングメモリ

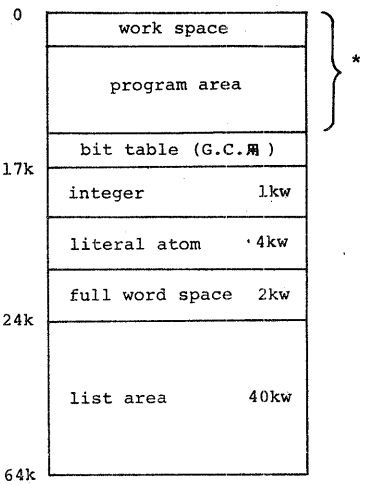
フィールド抽出回路は、AバスとALUの間にはいり、データのマスキングとシフトを同時に行なう。データの素通りを含めて4種類の固定パターンを有するにすぎないが、本LISPマシンは、タグマシンではなくマシンコードも規定しないため、フィールド抽出の多様性はあまり必要としない。抽出位置とビット数は、ジャンパ線により変更できる。

マッピングメモリは3ビット×1k語あり、主記憶アドレスを入力として、表1に示す3ビットのコードを出力する。すなわち、主記憶を64語毎に1024区画に分割し、各区画に、利用種別を表わす3ビットのタグを付けたと考えることができる。図3がLISPマシンのメモリ割当てであるが、マッピングメモリにより、アドレスから、利用種別のコードがただちに得られる。このコードやフィールド抽出回路の出力を定数（アドレス）と加算することにより、インデックスジャンプのような形のマルチウェイジャンプが実現できる。

マッピングメモリの出力は、Aバスの他に、デコーダをとおしてフラグレジスタに接続されており、list, 文字アトム、数値、アトムの判定が簡単に行なえる。

コード	種　別
0	小整数1
1	小整数2
2	小整数3
3	整数
4	文字アトム
5	フルワード
6	フリーリスト1
7	フリーリスト2

表1. マッピングメモリによるコードと種別の対応



* 小整数に対応

図3. メモリ割当て

3.6 その他のレジスタ

メモリとのデータ受け渡しは、ICAR(input car register), ICDR(input cdr register), OCAR (output car reguster), OCDR(output cdr register) を通して行なう。MADR(メモリアドレスレジスタ)にアドレスを書きこむと、メモリオペレーションは自動的に始まる。

メモリオペレーションは、read と read while write の2種である。read while write とは、メモリセルの内容の読み出しと同時に、別のデータの書きこみが行なわれるモードである。最近のダイナミックRAMにおいて可能となったもので、アトムの value cell の書きかえに有効である。

LISP インタープリタにおいて、=NIL を判定する機会は多く、演算結果が NIL であることをフラグに出力する回路を設けた。NIL レジスタがそれである。

また、LSI-11に割込みを発生させるためのレジスタとして、INTR(インタラプトレジスタ)がある。INTR にデータを書きこむと、その値がベクタアドレスとなって LSI-11 に割込みを発生し、受け付けられると、フラグレジスタに ack フラグが立つ。

3.7 ビットアドレッシング回路

主記憶上のビットテーブルに対して、ビットテストとビットセットを行なう。ガーベージコレクション時に、スタックを用いたマーキングを行なうので、1語(リストエレメント)に対し1ビットのマークビットを用意すると、64kビット、2k語のビットテーブルとなる。ガーベージコレクションルーチンでは、テーブルをすべてクリアしたあと、BASEREG(ベースレジスタ)にテーブルの先頭アドレスを書きこむ。以後マーキングアルゴリズムに移り、たとえば8000(16)番地のリストエレメントに対し、マークの有無が知りたければ、BADR(ビットアドレスレジスタ)に、TEST 指定で8000(16)を書きこむ。するとメモリ参照が起り、2つあとのマイクロ命令サイクルで、フラグレジスタにマークの有無が現れる。ビットセットの場合にも同様に、SETの指定で BADR に書きこめば、対応ビットがセットされる。こうして、ビットテーブルへのアクセスが、簡単に行なえる。

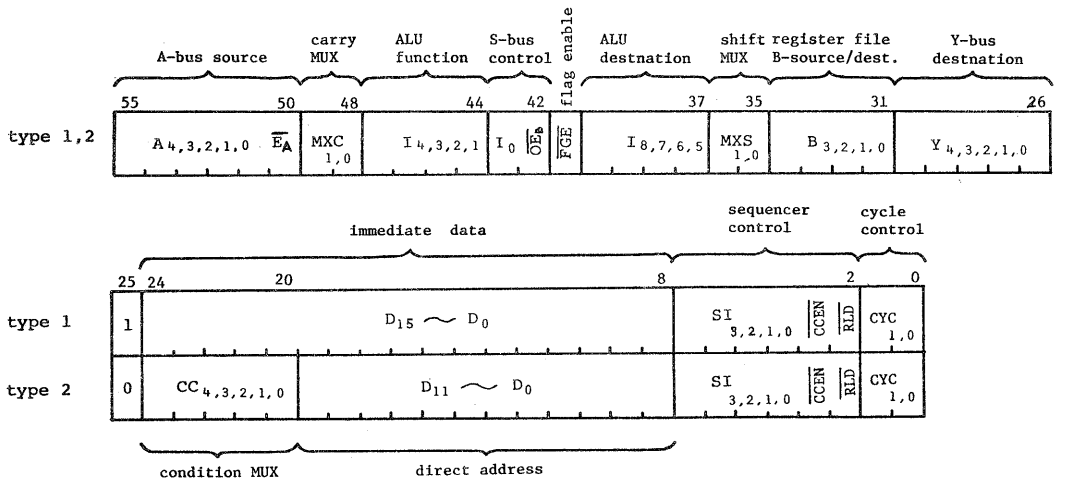


図4. マイクロ命令フォーマット

4．マイクロ命令とコントロールサイクル

4.1 マイクロ命令フォーマット

ビットスライスALU、マイクロプログラムシーケンサ、外付けのスタックやレジスタ類を制御するために、図4に示す56ビットのマイクロコードを使用する。

マイクロ命令は2つのタイプに分かれ、ビット25が1のときをタイプ1、ビット25が0のときをタイプ2と呼ぶ。タイプ1はALUに対して定数を与えることのできるタイプ、タイプ2はフラグをテストして条件ジャンプのできるタイプである。ALU動作については、いずれのタイプでも同様に指定できる。以下にフィールドごとについての説明を簡単に加える。

ビット50～55はAバスソース指定、ビット26～30はYバスデスティネーション指定である。詳細をそれぞれ表2,3に示す。スタックアクセスに対するモード指定は、これらのフィールドにて行なう。タイプ2のビット20～24は、ジャンプ条件の指定であり、3.3節に示したフラグがジャンプ条件として、指定可能である。　ビット37～40、42～47はAm2903のオペレーション指定、ビット35、36、48、49はシフトマルチプレクサとキャリー入力に対する指定、ビット41はフラグレジスタのenable指定、ビット2～7はAm2910に対するオペレーション指定である。[11]

A_4	A_3	A_2	A_1	A_0	EA	A bus source	
X	A_3	A_2	A_1	A_0	0	register file	1)
0	0	0	AF_1	AF_0	1	ICAR	2)
0	0	1	AF_1	AF_0	1	ICDR	
0	1	0	AF_1	AF_0	1	STP	
0	1	1	AF_1	AF_0	1	MAPPING MEMORY	
1	0	0	AF_1	AF_0	1	(STP)	3)
1	0	1	AF_1	AF_0	1	(STP)-	4)
1	1	0	AF_1	AF_0	1	(SAR)	
1	1	1	AF_1	AF_0	1	(SAR)-	

1) A_3～A_0 はレジスタファイルのアドレス指定
2) AF_1, AF_0 はフィールド抽出回路の指定
3) (STP) はスタックポインタによる　スタックの間接アドレス指定
4) (STP)- は間接後スタックポインタを自動減少させる指定

表2.　Aバスソース指定

4.2 マイクロ命令コントロールサイクル

マイクロ命令の実行サイクルは、基本的に300n sec であり、またALUおよびシーケンサLSIは、1相クロックで制御されるよう設計されているが、システムクロックとしては、図5に示す4相クロックを用いている。これは主にスタックコントロールに用いられる。AバスソースやYバスデスティネーションにスタックを指定した場合、$\overline{\phi_0}$と$\overline{\phi_1}$がスタックからの読出しフェーズ、$\overline{\phi_3}$がスタックへの書きこみフェーズになる。また、LSIへのク

Y_4	Y_3	Y_2	Y_1	Y_0	Y bus destination	
0	0	X	X	X	未使用	
0	1	0	0	0	OCAR	
0	1	0	0	1	OCDR	
0	1	0	1	0	STP	
0	1	0	1	1	SAR	
0	1	1	0	0	BASEREG	
0	1	1	0	1	STKLIM	
0	1	1	1	0	NILREG	
0	1	1	1	1	INTR	
1	0	0	0	0	MADR R	1)
1	0	0	0	1	MADR W CAR	2)
1	0	0	1	0	MADR W CDR	3)
1	0	0	1	1	MADR W BOTH	4)
1	0	1	0	X	BADR TEST	5)
1	0	1	1	X	BADR SET	6)
1	1	0	0	X	(STP)	
1	1	0	1	X	+(STP)	7)
1	1	1	0	X	(SAR)	
1	1	1	1	X	+(SAR)	

1) メモリリード
2), 3), 4)　メモリライト、　それぞれcar部のみ、cdr部のみ、両方
5), 6) ビットテーブルのテストとセット
7) スタックポインタの自動増加後間接アドレス指定

表3.　Yバスデスティネーション指定

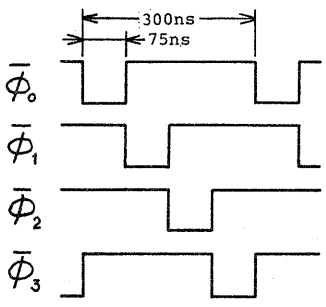


図5.　システムクロック

CYC_1	CYC_0	$\overline{\phi_0}$のパルス巾(ns)
0	0	75
0	1	150
1	0	225
1	1	wait要求がなくなるまで

表4.　マイクロコードによる$\overline{\phi_0}$のコントロール

ロックをはじめ、各レジスタに対する書きこみクロックにはϕ_3を用いる。

またϕ_0のパルス巾は、マイクロコードのビット0,1の指定により、表4のように75n secの整数倍だけ延長される。たとえば、ALUの演算結果をジャンプアドレスに使うようなときには、パルス巾の延長を行ない、WCSにnext addressが与えられるのが遅れた分だけ、サイクルを延ばしてやる。またメモリオペレーション時には、wait要求がなくなるまで、ϕ_0パルスが延長されてメモリに同期する。

4.3 メモリコントロール

メモリオペレーションは、表3に示したとおり、Yバスデスティネーション指定によってMADRまたはBADRにアドレスを書きこんだ時点から開始される。MADR等への書きこみは、ϕ_3の立上りで行なわれ、ただちにメモリrequestが発せられる。次のϕ_0は、メモリからのacknowledge信号が帰るまで(wait信号がなくなるまで)延長され、メモリからのリードデータは、次のϕ_3の立上りでICAR, ICDRレジスタにラッチされる。こうしてメモリオペレーションは、MADRまたはBADRへの書きこみの、次のサイクル中に完了する。メモリ動作が進行している間の1サイクルは、メモリに関係のあるレジスタを除いて、普通のALUオペレーションが可能である。

5. マシンコード

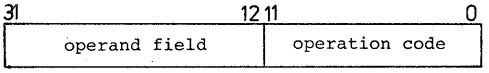


図6. マシンコードの設定

コードの長さは、メモリの読出し巾に合わせて32ビットとする。そして、LISP言語との対応の良い中間言語を設定し、その中間言語をマイクロプログラム化して、そのマイクロプログラムのエントリーアドレス12ビットを、そのままマシンコードと考えてしまう(図6)。fetch段階では、下位12ビットを抽出し、マイクロプログラムジャンプする。残り20ビットのオペランドフィールドは、フィールド抽出回路を使って必要に応じ利用する。ビット利用率が悪いが簡単ではある。

6. LISP言語の仕様

6.1 データの種類

以下の4種類のデータ型を取扱う。

a. 小整数: +8191 ~ −8192である。

b. 整数: 整数域のメモリセル1個に1つずつ格納され、32ビットの2の補数表現。

C. 文字アトム: 図7のとおり、32ビット4語連続の領域をとる。属性別飛び先は、EVAL中で評価されるときのマイクロプログラムのジャンプアドレス(12ビット)であり、属性は、アトムの属性を4ビットにコード化したもの(表5)である。関数本体は、関数の種類によりマイクロプログラムのアドレス、マシンコードのアドレス、またはリストのアドレスである。変数のbindingには、shallow-bindingを用い、value部に値を置く。

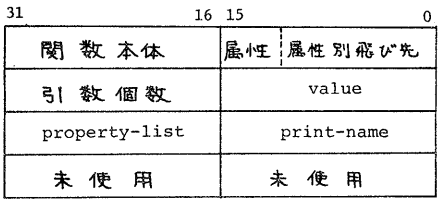


図7. 文字アトム

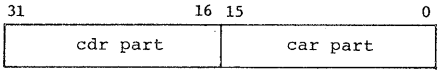


図8. リストエレメント

d. リストエレメント: 図8にリストエレメントを示す。

これらのデータ型は、メモリの決まった領域に決まったデータ型をかためて置くことによって番地で区別するが、実際にはマッピングメモリを使って、高速にテストできる。

6.2 関数型の種類

関数型は、LISP 1.5 で用意された SUBR、FSUBR、EXPR、FEXPR の他に、システム関数用として MSUBR、MRSUBR、MFSUBR、MRFSUBR を加えた。SUBR、FSUBR がマシンコードで記述されるのに対し、MRSUBR、MRFSUBR はマイクロコードで記述され、再帰呼出しも許している。一方再帰を許さず、マイクロサブルーチンの形で記述されるのが MSUBR、MFSUBR である。主な関数を次に示す。

MRSUBR : EVAL, APPLY, MAP, GET, EQUAL, EVLIS, etc.
MRFSUBR: COND, PROG, GO, AND, OR, LIST, TIMES, etc.
MSUBR : CAR, CDR, CONS, EQ, NULL, ATOM, ADD1, etc.
MFSUBR : DE, EF, QUOTE, SETQ, etc.

コード	属性
0	normal atom
1	MSUBR
2	MRSUBR
3	SUBR
4	MFSUBR
5	MRFSUBR
6	FSUBR
7	EXPR
8	FEXPR
9	未使用
10	〃
11	〃
12	〃
13	〃
14	〃
15	APVAL

表5. アトムの属性とコード

6.3 スタックの構成

スタック上には、関数の実行毎に frame というものが作られる。frame の構造を図9に示す。frame head には関数本体へのポインタ、μ-PC（マイクロプログラムカウンタ）と PC（プログラムカウンタ）を保存する領域、および1つ前の frame head の位置を示す old FP が含まれる。local pdl (push down list) には、関数本体の実行時には、計算の中間結果やマイクロサブルーチンへの引数などが置かれる。FP は frame pointer で frame head の位置を示し、STP は stacktop pointer で現在のスタックの先頭を示す。この frame の構造は、リスト、マイクロコード、マシンコードで記述された各種レベルの関数を、統一的にコール、リターンするために利用されるもので、Bobrow モデル [10] のものとは異なる。なお PC、FP は Am2903 のレジスタファイル中にとられる。

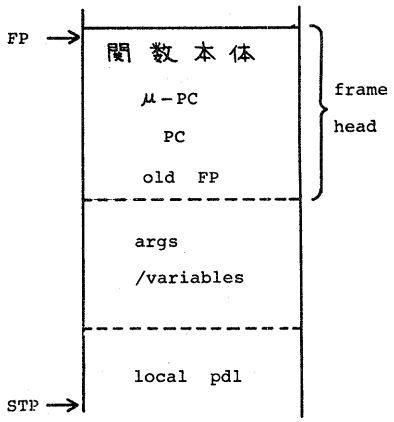


図9. frame の構造

6.4 式の評価

式の評価の手順は、つぎのようになる。

1. frame head の作成
2. 引数の評価
3. 関数本体の実行
4. 結果のリターンと frame の消滅

インタープリタで次の式を評価する場合を例にとって、解説を加える。

（関数1　引数1　引数2
　引数3 ）

EVAL の中で式がアトムでないことが分かった時点で、frame head が作られ、関数1の性質が調べられて、引数の評価方法と評価後の関数本体への渡し方が決まる。引数を評価しては、frame head の下へ積んでゆき、完了後、関数本

```
MKFRAME :
    SAR ← FP ;
    +(SAR) ← MPC ;
    +(SAR) ← PC ;
    MPC ← 'initial adr. ;
    PC ← (STP) ;
    SAR ← STP +3 ;
    (SAR) ← FP ;
    FP ← STP ;
    STP ← FP +3 ;

RETURN :
    SAR ← FP ;
    (SAR) ← (STP) ;
    STP ← FP ;
    SAR ← FP +3 ;
    FP ← (SAR)- ;
    PC ← (SAR)- ;
    MPC ← (SAR) ;
```

図10. MKFRAME と RETURN オペレーションの記述

体へ制御を渡す。関数本体の中で、引数を用いて計算が終了すると、結果を持って呼び出し元のframeへもどる。もし、引数1に再び（関数2　引数1'）の形が現れたならば、引数リスト（引数2　引数3）と引数の評価方法・関数本体への渡し方の情報を、frame head（PCとμ-PC位置）へ保存して、新しいframeを作り、引数1'の評価に移る。SUBR関数の場合には、おおよそこのようになる。

MKFRAME（frame headの作成）とRETURN（frameをたたみ、関数の値を返す）オペレーションの記述を、図10に示す。STPやSARの自動増加、減少モードが利用される。

7。あとがき

本LISPマシンは、現在ハードウェアの製作中であり、またマイクロプログラムアセンブラ、シミュレータの作成も並行して行なっている。

LISP処理向きに用意したそれぞれのハードウェアが、どの程度効率向上に役立っているか、その評価と、ソフトウェアの充実がこれからの課題である。

謝辞

研究に協力してくださった、システム工学科第4講座の、小林康博氏、多田光弘氏、滝本博道氏に感謝します。また御討論いただいた、電子技術総合研究所の島田俊夫氏、山口喜教氏に深謝いたします。

参考文献

[1] W.Teitelman: INTERLISP Reference Manual, Xerox,(Feb. 1974).
[2] LISP 1.9 User's Manual, EPICS-5-2,
[3] 島田, 山口, 坂村: LISPマシンとその評価, 信学 論文誌D, J59-D, 6, (June 1976).
[4] 山口, 島田, 他: ACE上のLISPマシン, 信学 研, EC-76-13, (May 1976).
[5] 山口, 島田: 仮想マシンによるLISPプログラムの動的特性, 信学 研, EC-77-16, (1977).
[6] 長尾, 中村, 他: LISPマシン NK3のアーキテクチャとそのマイクロインストラクション, 信学 研, EC-77-17, (1977).
[7] Aian Bawden et al: LISP Machine Progress Report, MIT AI Lab. Memo No.444, (Aug. 1977).
[8] R.Greenblatt: The LISP Machine, MIT AI Lab. Working Paper 79,(Nov. 1974).
[9] Tom Knight: CONS, MIT AI Lab. Working Paper 80,(Nov. 1974).
[10] D.G.Bobrow: A Model and Stack Implementation of Multiple Environments, C.ACM Vol.16,No.10,(1973).
[11] The Am2900 Family Data Book: Advanced Micro Devices, Inc.