計算機アーキテクチャ　68－15
(1987・11・13)

Smalltalk-80専用マシンHobbes
Smalltalk-80 machine Hobbes

飯塚　裕　　木下　康幸　　武内　春夫
Hiroshi Iizuka　　Yasuyuki Kinoshita　　Haruo Takeuchi

沖電気工業㈱　　沖北陸システム開発
OKI Electric Industry Co.,Ltd　　OHSD

あらまし　Smalltalk-80を高速に実行するために、Smalltalk-80専用マシンHobbesを開発した。本論文では、Hobbesのハードウェア構成、バーチャルマシンの実現方法、特にsend、returnバイトコード実行時の処理について述べる。ベンチマークテストを行なった所、パーフォーマンス・レーティング197を得た。

Abstract　We Developed Smalltalk-80 machine Hobbes that accelerates execution speed. This paper describes its hardware architecture, implementation of Virtual Machine and execution of send and return bytecodes. This paper also describes result of benchmark tests. Its performance rating is 197.

1．まえがき

Smalltalk-80はXerox社ＰＡＲＣで開発されたプログラミング言語／開発環境である。[1] Smalltalk-80は、ユーザ・インタフェースに優れ、評価は高いのであるが、実行時のオーバヘッドが多く速度が遅くなる傾向がある。そこで我々は、Smalltalk-80をより高速に実行するために、Smalltalk-80専用マシンHobbesを開発してきた。[2][3][4]

本論文では、Hobbesのハードウェア構成、バーチャルマシンの実装方法、バイトコード・インタプリンタ、特にsendとreturnバイトコード実行時の処理について述べる。

2．ハードウェア構成

2.1　全体構成

図１にHobbesの全体構成を示す。プロセッサは１台のカードラックに実装し、32ビットのＶＭＥバスから、バスを延長する形で多機能ワークステーション[5]を接続している。32ビットＶＭＥバスには68000ボードと、メモリを5.5ＭＢ実装している。そのうち、１ＭＢをＶＩに用い、４ＭＢをヒープ領域としている。68000は、プロセッサの起動などのコントロールを行なう他、ミリセカンドクロック、タイマーとして動作する。多機能ワークステーションは、68010をＣＰＵとするunixマシンである。多機能ワークステーションは、イメージ・データ処理用に29116を用いたイメージ・プロセッサを内蔵している。イメージ・プロセッサ内には２ＭＢのメモリがあり、プロセッサ側から参照可能である。２ＭＢのうち、約1.5ＭＢをform領域として使用している。Smalltalk-80動作時には、多機能ワークステーションのキーボード、マウスからのインタラプトはプロセッサで受付ている。イメージ・プロセッサからのインタラプトは、32ビットＶＭＥバス上の68000で受付る。BitBLT、ScanCharactersなどを実行する時、プロセッサは必要なパラメータを68000のキューにセットするだけで、イメージ・プロセッサのコントロ

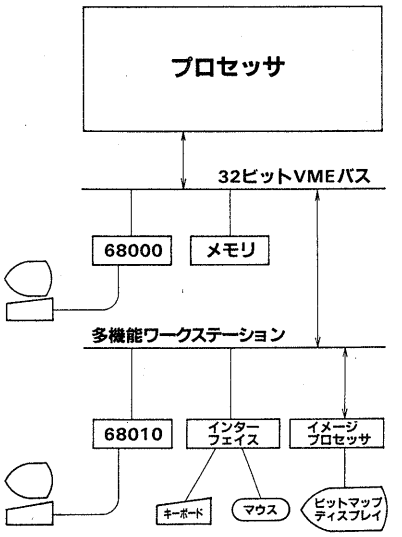


図1　全体構成

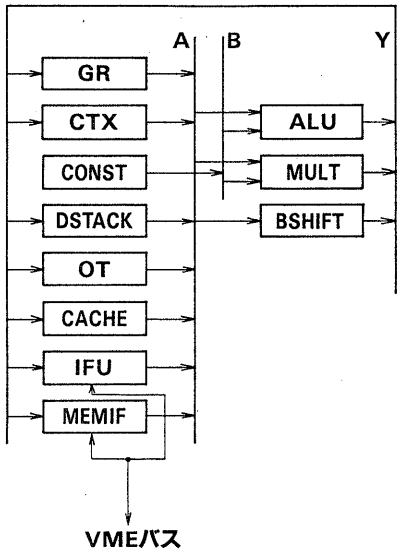


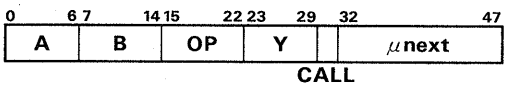
図2　プロセッサブロック図

ールは68000側で行なっている。

2.2　プロセッサの構成

図2にプロセッサのブロック図を示す。A、B、Y3本のデータバスにALU、バレルシフタなどの通常の構成要素の他、コンテクスト・スタック（ctx)、データ・スタック（dstack)、オブジェクト・テーブル（ot)、メソッド・キャッシュ（cache）など、Smalltalk-80を高速に実行するのに要するユニットを接続している。各ユニットは、できる限り独立性を高めている。レジスタ類、データバスなどは、すべて32ビットである。マイクロ命令は、1ワード48ビット構成であり、16kWの容量のマイクロプログラムメモリを実装している。図3にマイクロコードのフィールド割当てを示す。Aフィールドは、Aバスにデータを出力するレジスタ類を指定する。Bフィールドは Bバスにデータを出力するレジスタ類を指定する。OPフィールドは演算の種類を指定する。CALLフィールドはμプログラムのCALL, RETURNなどを指定する。μnextフィールドはページ内ジャンプ(1ページは256W)、ページ外ジャンプ、条件ジャンプ、8ビットコンスタントなどを多重に割当てている。

図4にバスのタイミングを示す。クロックは8MHzであり、図4に示すように、25nsづつずらした3相のクロックを使用して1クロック周



フィールド	ビット数
A	7
B	8
OP	8
Y	7
call	2
μnext	16

図3　マイクロコードのフィールド割当て

期＝125ns内をさらに細かく制御している。マイクロ命令は3段パイプラインで実行する。1クロック周期内の時刻を図4のようにT0～T5とすると、μアドレス・バスが有効になるのは1ステージのT0～T4、μコード・バスが有効になるのは1ステージのT3から2ステージのT1、A、Bバスが有効になるのは、1ステージのT5から2ステージのT3、演算を行なうのは2ステージのT3から3ステージのT1、Yバスが有効になるのは3ステージのT2から3ステージの最後まで

である。ただしＡＬＵは条件フラッグを３ステージのT0に有効とする必要がある。

又、図２には示していないが８ビットの多目的バスがあり、μコードのipl、内部状態の読み取りなどに使用している。

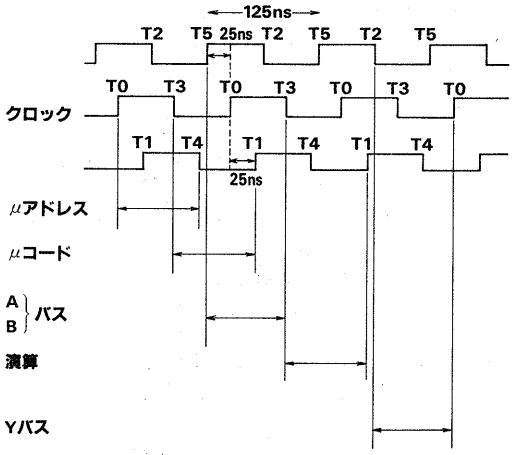


図４　バスのタイミング

2.2.1　レジスタ類

プロセッサ内に汎用レジスタ(gr)、テンポラリ変数などを保持するためのコンテクスト・スタック(ctx)、formal specificationではコンテクスト中に含まれるデータ・スタック(dstack)を持っている。コンテクストをオブジェクトとせず、スタック上に置きＶＭを高速化する手法はよく知られている。[6] 図５にgr、ctx、dstackの構成を示す。Smalltalk-80のプロセスに対応し、gr、ctx、dstackはそれぞれ８セット用意している。grとctxはレジスタpnoの下位３ビットで選択する。dstackはレジスタdspnoの下位３ビットで選択する。ただし、実際には、pno＝７は後述のPrimインタプリンタで使用している。pnoとdspnoが分かれているのは、Primインタプリンタでプリミティブを実行する場合、pno＝７、dspno＝２などという場合があるためである。なお、インタラプト処理はpno＝dspno＝７で行なう。grのサイズは32Wであり、ctxのサイズは16段×16W、dstackのサイズは１kWである。

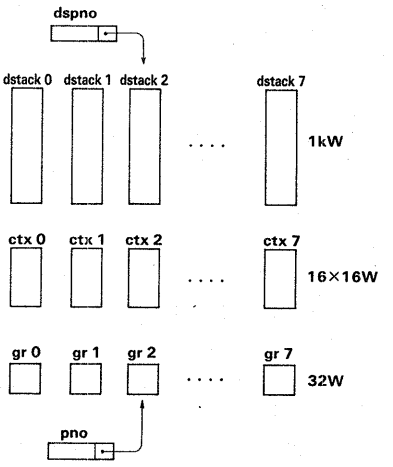


図５　gr, ctx, dstack

図６にctxの構成を示す。16段のうちの１つをレジスタcspの下位４ビットで選択する。cspのサイズは16ビットである。pnoとcspで選択した16Wはμプログラムからは16Wのレジスタと見える。この16Wにコンテクストのうちのスタック以外の部分を置く。コンテクストをスタックとそれ以外の部分に分けたのは、コンテクストのサイズを２ⁿの固定長とするためである。Smalltalk-80実行時にはコンテクストが頻繁に切り変わるが、本方式ではcspの書き変えだけ

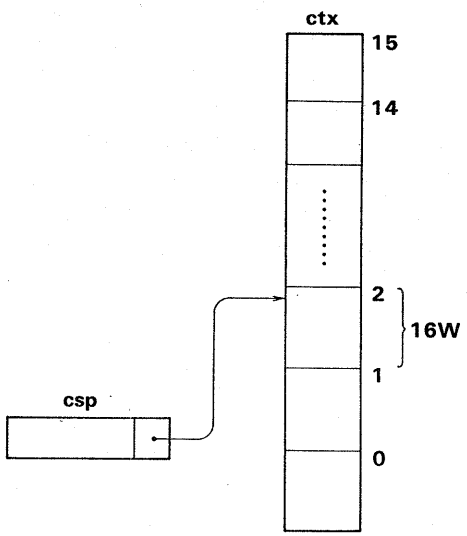


図６　ctx

で簡単に行なえる。又、テンポラリ変数を参照する場合に、ポインタに対して相対参照する必要はない。

2.2.2 演算ユニット

ＡＬＵにはSN74AS181を用いた。入出力32ビットの算術、論理演算を行なう。演算結果をＡバス、又はＢバスに出力する機能もあり、直前に実行した演算結果を入力とすることもできる。ＭＵＬＴはＡバスの下位16ビットとＢバスの下位16ビットの乗算を行なう。ＢＳＩＦＴはＡバス上のデータ０～31ポジションのローテートシフトと、64ビットデータから任意の位置の32ビットデータの抜き出しを行なう。この機能により連続する複数ワードのビット単位のオフセットの転送を行なえる。

2.2.3 オブジェクト・テーブル

図７にオブジェクト・テーブル(ot)のエントリを示す。オブジェクト・ポインタ(oop)のサイズは32ビットであり、LSBにより、otへのインデックスとSmallIntegerとを区別している。oopのLSBが〝０〟なら31ビットの整数である。oop中のインデックスはotの１つのエントリを指している。otのエントリに含まれるのはオブジェクトのアドレス（24ビット)、リファレンス・カウント（８ビット)、フラッグ（16ビット中３ビットのみ使用）である。otのエントリ中のアドレスは、オブジェクトの先頭ワードを指している。オブジェクトの先頭ワードはオブジェクトのバイトサイズであり、次のワードはオブジェクトのクラスである。

otには、マイクロ命令でoopをotに書き込んだ時に、そのoopがインデックスであるかSmall Integerであるかをチェックする機能がある。otのサイズは64kとしている。otのサイズにより、オブジェクトの最大数が制限されるが、実用上問題はない。

2.2.4 メソッド・キャッシュ

sendバイトコードを高速に実行するためにメソッド・キャッシュ(cache)をハードウェア化した。cache にレシーバのクラスとセレクタを書き込み、cacheがヒットすれば、メソッドを読むと同時にメソッド固有の処理プログラムにジャンプする。ジャンプは、メソッド・ヘッダ、拡張メソッド・ヘッダに含まれる情報をエンコードしたインデックスをμアドレス・バスにorすることにより行なう。図８にメソッド・キャッシュのインデックスを示す。cacheがヒットしなかった場合のインデックスは０である。その他、新コンテクストを生成する場合、プリミティブを実行する場合、インスタンス変数を帰す場合がある。cacheのインデックスは１度計算したらメソッド・ヘッダの上位16ビットに格納しておく。cacheのサイズは４kWである。

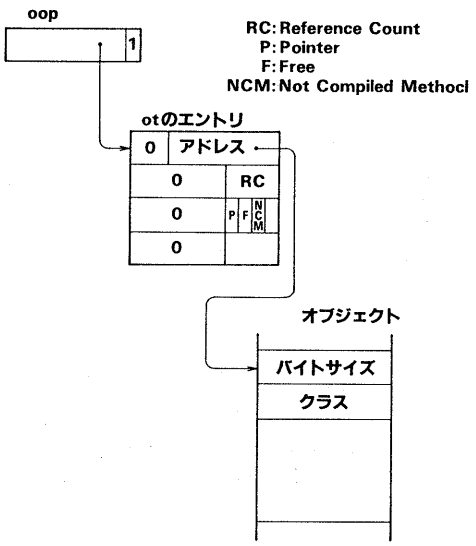


図７　オブジェクト・テーブルのエントリ

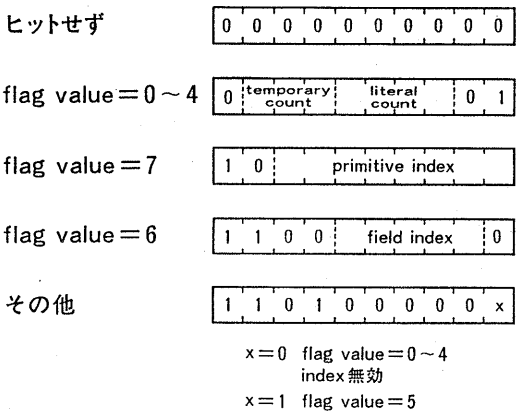


図８　メソッド・キャッシュのインデックス

2.2.5　ifu

バイトコードによるディスパッチを高速化するためにインストラクション・フェッチ・ユニット(ifu) を付加した。ifuは本体とは独立のマイクロ・プログラムで動作し、バイトコードのプリフェッチを行なう。ifuからバイトコードを読み、そのバイトコードによりディスパッチするのは1クロックで行なえる。それと同時にプロセス・スイッチの有無のチェック・インタラプトの必要の有無のチェックも行なう。pushTemporalyVariableなどのバイトコードの実行は1マイクロ命令で行なえるので、バイトコードの最少実行クロックは2クロック(250ns)となる。

3．VM

Virtual Machine(VM) は、大部分をマイクロ命令で記述し、一部をプリミィティブ記述用命令セット(Prim)で記述している。図9にマイクロコードのメモリマップを示す。VMはバイトコード・インタプリンタ、プリミィティブ、メモリ管理に分けられる。その他にPrimインタプリンタが必要である。

Prim命令セットで記述するのはプリミィティブのうち時間的に余裕のあるもの、システムの初期化、インタラプト処理プログラムなどであり、容量は約8kBである。

バイトコード・インタプリンタはコンテクストがctx上にある場合とヒープ上にある場合用に2種類持っている。

メモリ管理は、オブジェクトのアロケーション、ガベージ・コレクション、ヒープのコンパクションなどを行なうプログラムである。ガベージ・コレクションにはマーキング方式を使っている。

3.1　レジスタの変数割当て

図10にgrの割当てを示す。gr0～15は作業用に使用している。ctxspsaveには、プロセスが切替わる時にcspをセーブする。ctxspbtmはctx中の最下位のコンテクストを指す。+17というのは正しい値+17を保持することを意味している。

dataspsaveにはプロセスが切替わる時にデータ・スタックポインタ(dsp) をセーブする。

dataspbtmはdstackの有効エリアの1W下を指す。

argcnt、tempcnt、selectorは、sendバイトコードの実行時に使用する。actctxp、actctxa、homectxp、homectxaはコンテクストがヒープ上にある時にアクティブ・コンテクストとホーム・コンテクストのポインタとアドレスを保持する。senderは、ctxspbtmが指すコンテクストのセンダーである。processにはそのgrが含まれるプロセスを保持している。プロセスが切替わる時、コンテクストがctx上にある場合、プロセス中

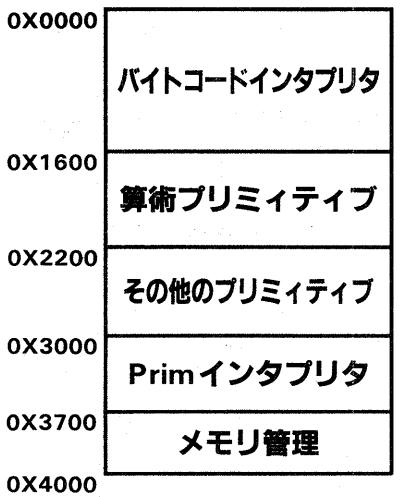


図9　マイクロコードメモリマップ

16	ctxspsave
17	ctxspbtm+17
18	dataspsave
19	dataspbtm
20	argcnt
21	tempcnt
22	selector
23	actctxp
24	actctxa
25	homectxp
26	homectxa
27	sender
28	process
29	priority
30	procstatus
31	flags

図10　grの割当て

のSuspendedContextのフィールドにはpnoをSmallIntegerにして格納する。ctxをより高いプライオリティのプロセスに明け渡す場合、コンテクストをヒープ上に作成しなおし、SuspendedContextを書き直す必要がある。priorityはプロセスの優先度を保持する。procstatusはプロセスの状態を保持する。flagsは各種のフラッグを保持する。

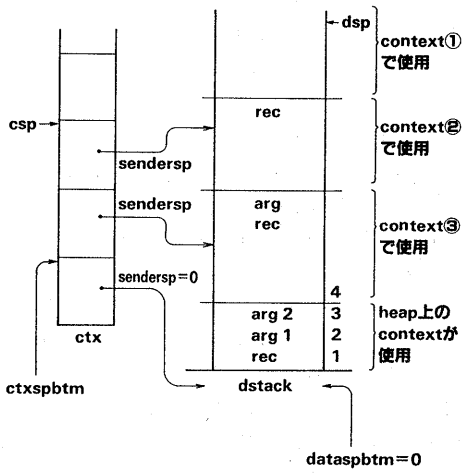
図11にctxの割当てを示す。senderipは1つ下のコンテクストのインストラクション・ポインタをアドレスの形で持つ。sendrspは1つ下のコンテクストのdspを保持する。methpとmethaは現在実行中のメソッドのポインタとアドレスである。recpとrecaはレシーバのポインタとアドレスである。temp0～9はアーギュメントを含むテンポラリ変数である。テンポラリ変数が10を越える場合はdstack上に置くが、その割合は極めて少ない。

0	senderip
1	sendersp
2	methp
3	metha
4	recp
5	reca
6	temp0
7	temp1
8	temp2
9	temp3
10	temp4
11	temp5
12	temp6
13	temp7
14	temp8
15	temp9

図11 ctxの割当て

3.2 send、return時の処理

図12にctxとdstackの関係を示す。①を現コンテクストとすると、①のsendrspは①のレシーバの1W下を指す。sendrspを保持するのは、return時にdspを復帰するためとdstack上のテンポラリ変数をアクセスするためである。図13にsend実行時の処理を示す。cspは現コンテクストを指し、dspは1番上のアーギュメントを指している。sendを行なう時は、cspをインクリメントし、レシーバとアーギュメントを新コンテクストにコピーし、新コンテクストのsenderはレシーバの下を指すようにする。新コンテクストは、dstackのうちsendが行なわれた時のdspが指すワードの上側を使用することになる。図14にreturn実行時の処理を示す。sendrspをdspに代入し、リターン値をpushし、cspをデクメリントする。sned実行時にctxがオーバーフローする場合はctxspbtmが指すコンテクストのみヒープ上にはき出す。ctx上で新コンテクストを生成する場合のsendの実行は35クロック程



①:active context

②:active contextのsender

図12 ctxとdstackの関係

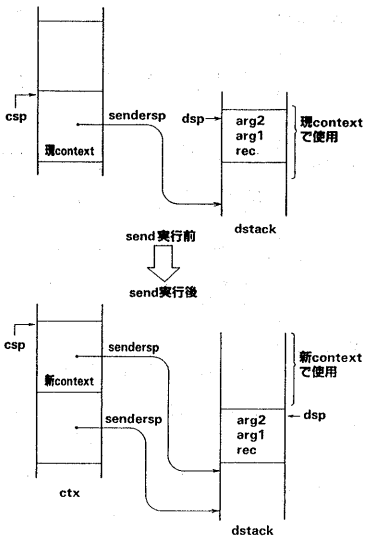


図13 send実行時の処理

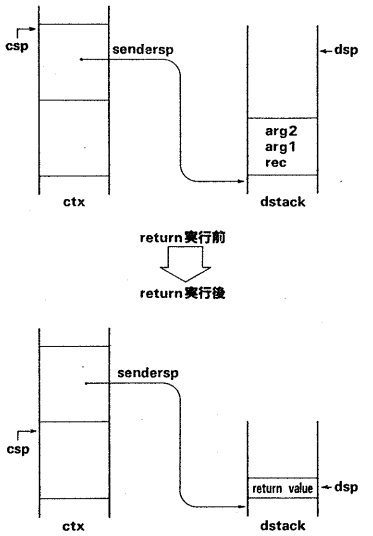


図14　return実行時の処理

r0〜r15：汎用レジスタ　　(32ビット)
sp　　：スタックポインタ　(16ビット)
pc　　：プログラムカウンタ(24ビット)
sr　　：ステータスレジスタ

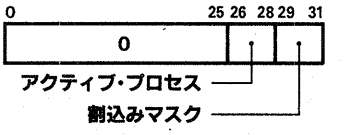


図15　Prim命令セットのレジスタ

程度で行なえる。returnはifuにsenderipをセットし、ディスパッチ可能になる時間で決り、約12クロックで実行できる。

3.3　プロセスに対するレジスタ類の割当て

プロセスに対するレジスタ類の割当てと解放は以下の方針で行なう。

①プロセスがactiveになる時にはレジスタ類を割当てる必要がある。コンテクストがヒープ上にある場合も必要である。
②プロセスがactiveでない時にはレジスタ類が割当てられてなくともよい。
③コンテクストがヒープ上にある時、プロセスがactiveでなくなる時にはレジスタ類は解放する。
④解放はプロセスの状態(suspended、wait、sleepの順）と優先度により行なう。

3.4　Prim　命令セット

Prim命令セットは今回新たに作成したスタック操作を中心にした命令セットである。図15にPrim命令セットのレジスタを示す。r0〜15はpno＝7のctxを割当てている。この他にdstackも使用する。Prim命令セットはifuをバイトコード・インタプリタと共用する関係から1バイトのオペレーション・コード＋ソース・オペランド＋デスティネーション・オペランドという命令構成を持つ。Primの命令セット・インタプリタの速度は約1MIPSである。

4．評価

Smalltalk-80では、インプリメンテーション毎の性能算出方法としてベンチマークテストが定められている。[7] 総合評価のためのベンチマークのためのプログラムはＶＩ中に含まれている。総合評価値パーフォーマンス・レーティン

表1　ベンチマークテスト結果

テスト	Dorado(秒)①	Hobbes(秒)②	①/②*100
BitBLT	0.399	0.005	7980
TextScanning	0.312	0.029	1076
Class(rganizer	1.207	1.222	99
PrintDefinition	0.849	0.854	99
PrintHierarchy	1.000	0.672	149
AllCallsOn	1.769	1.518	117
AllImplementors	0.674	0.748	90
Inspect	0.910	0.570	160
Compiler	2.256	1.948	116
Decompiler	1.243	1.170	106
KeyboardLook Ahead	0.849	0.583	146
KeyboardSingle	2.181	1.840	119
TextDisplay	1.279	0.785	163
TextFormatting	1.146	0.752	152
TextEditing	4.218	1.844	229
		パーフォーマンスレーティング	197

グを求めるのは次の手順による。

①定められた15種類のテストを実行する。

②Doradoを100とした相対性能を求める。

③15種のテストの相乗平均を求める。

表1にベンチマークテストの結果を示す。BitBLTとTextScanningが特に高速になっているのは、マイクロ命令からは68000にパラメータを渡すだけで、イメージ・プロセッサの動作終了を持つ必要がないためである。パフォーマンス・レーティングは197を得た。

5．結論

Smalltalk-80を高速に実行するためのハードウェア構成、ＶＭの実装方式について述べた。Smalltalk-80の使用感はウィンドーの表示スピードに大きく影響される。Hobbesでは、restore display（ウインド全体を再表示する）を実行する場合のウィンドー１面当りの表示時間は0.1秒のオーダである。又、メモリアクセスの高速化、ブロック・コンテクストもctx上に置くなどＶＭ実装方式の開発、プロセッサのVLSI化などにより、さらに高速化も可能と思われる。

参考文献

1) Goldberg, A. Robson,D., Smalltalk-80 : The language and Its Implementation, Addison-Wesley, 1983.

2)飯塚、山田、Smalltalk-80専用マシンのハードウェア構成、5S-1、情報処理学会32回全国大会。

3)木下、武内、中沢、Smalltalk-80専用マシンのソフトウェア構成、5S-2、情報処理学会32回全国大会。

4)飯塚、木下、武内、Smalltalk-80専用マシンHobbesの概要、B-64、昭和61年度電気関係北陸支部連合大会。

5)峰本、瀬賀、新田、多機能ワークステーションの開発、沖電気研究開発124、1984.

6)鈴木、小方、「多態実行環境」：高級言語の制御機械の高性能実現法、情報処理、1985、10.

7)Krasner,G.,Smalltalk-80 : Bits of History, words of Advice, Addison-Wesley, 1983.