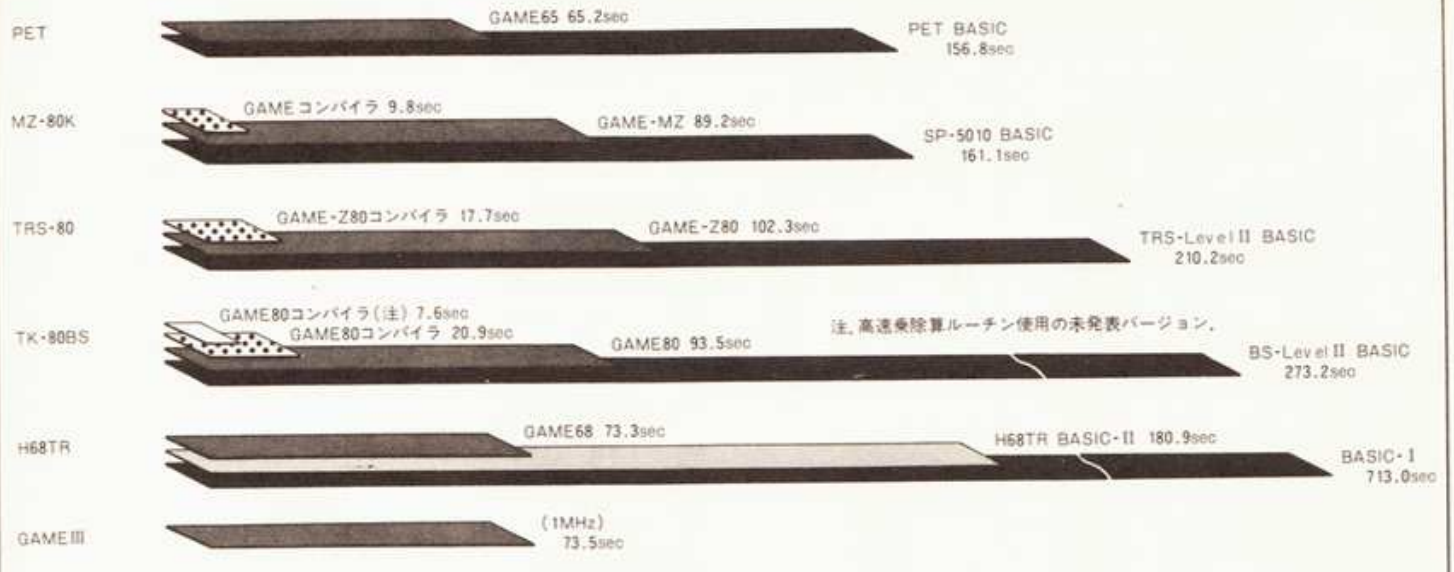

ベンチマークテスト	No.1	2	3	4	5	6	7	合計
PET GAME 65	1.1	3.4	7.2	7.9	9.5	15.2	20.8	65.2
PET BASIC	1.6	10.0	19.0	20.4	22.0	32.6	51.2	156.8
MZ-80K GAMEコンパイラ	0.1	0.1	1.7	1.7	1.8	2.1	2.3	9.8
MZ-80K GAME-MZ	1.8	5.2	7.6	9.2	11.4	21.5	30.5	89.2
MZ-80K BASIC SP-5010	1.1	9.0	16.0	22.3	25.2	36.5	51.0	161.1

単位、秒、PETは内部時計を使用して測定した。

図1　ベンチマークテストによる BASICとGAMEの処理速度の比較

超CPU-GAMEとは?

大西氏の新言語ＧＡＭＥⅢが発表されて以来1年半，今やＧＡＭＥは本来の6800のみならず8080，Ｚ80，6502の各ＣＰＵ・各システムに移植され大きく広がっています．

また，ＧＡＭＥによるアプリケーションやユーティリティープログラムもＡＳＣⅡが号を重ねるごとに充実し多くのホビーストの関心を集めています．

ここではＧＡＭＥを知りたい，あるいは再認識したいと考える人のためにあらためてＧＡＭＥⅢとＧＡＭＥファミリーの特徴と可能性とを調べてみたいと思います．

GAMEの特徴

ＧＡＭＥの魅力の第1に挙げられるのはそのスピードでしょう．上のベンチマークテストの結果をごらんください．各マシンともにインタプリタの速度は2倍以上，コンパイラに至っては10～20倍となっているのがわかると思います．これは文解釈等のアルゴリズムを通常のＢＡＳＩＣよりずっと速いものにしてあるためです．

メモリの利用効率が良いことも大きな長所のひとつです．インタプリタ自体もコンパクトなうえにプログラムも短くまとめる事が可能です．

仮想CPU－GAME

図2をご覧ください．これはあるＣＰＵの内部構成を示したものです．

16bit並列処理，26本の汎用レジスタ，プログラムエディット機能を持ちシンボリックインストラクションで動く超高性能ＣＰＵです．

残念ながら現在のところこのようなチップは生産されてはいません．しかし，あなたの持っているシステムを用いてこの超ＣＰＵをシミュレートし，等価な機能を発揮させ得るとしたらどうでしょうか？

ＧＡＭＥはまさにこのＣＰＵをシミュレートしているものなのです．いったんＧＡＭＥが動きはじめればもう68系，80系，65系，といった違いはなくなり，すべてのシステムは**「ＧＡＭＥ」**という名のＣＰＵを搭載したスーパーマシンとなるのです．

ＧＡＭＥを考える際に，「マイクロＢＡＳＩＣ系の言語である．」とする事はむしろまちがいでＧＡＭＥの最大の長所を見のがす危険があります．「8／16bit超ＣＰＵ」と考えるとはるかに理解しやすく，またその実力を活かした使い方ができると思われます．

実際問題として，スピードの速さはインタプリタの域を大きく超え，メモリ操作等の処理の細かさは通常のマシン語以上の能力を持ち，ＢＡＳＩＣと同等なエディタ機能を有しているといえば**「記号言語で動く超ＣＰＵシミュレーションランゲージ」**という呼称が納得ゆくかと思います．

次にこの仮想ＣＰＵがどのような動作をするかを説明しますのでそれを通じてＧＡＭＥの文法，オペレーション，特徴などの理解を深めていただきたいと考えます．

以下の説明は図2と見比べながら読んでいただくとわかりやすいようです．

内部構成

仮想ＣＰＵの内部は**図2**のとうりになっています．26本の16bit汎用レジスタはアキュムレータとしてもインデックスレジスタとしても使える強力なもので，いわゆる「変数」として扱われます．

中央が心臓部の命令解読，実行管理を司どる部分で，現実的には6800等のＣＰＵ及びＧＡＭＥインタプリタがこの役割を担っています．

特殊目的のレジスタ群は「＝」「＆」「＊」などのシステム変数にあたるもので，プログラムの編集管理，実行管理のために使われます．

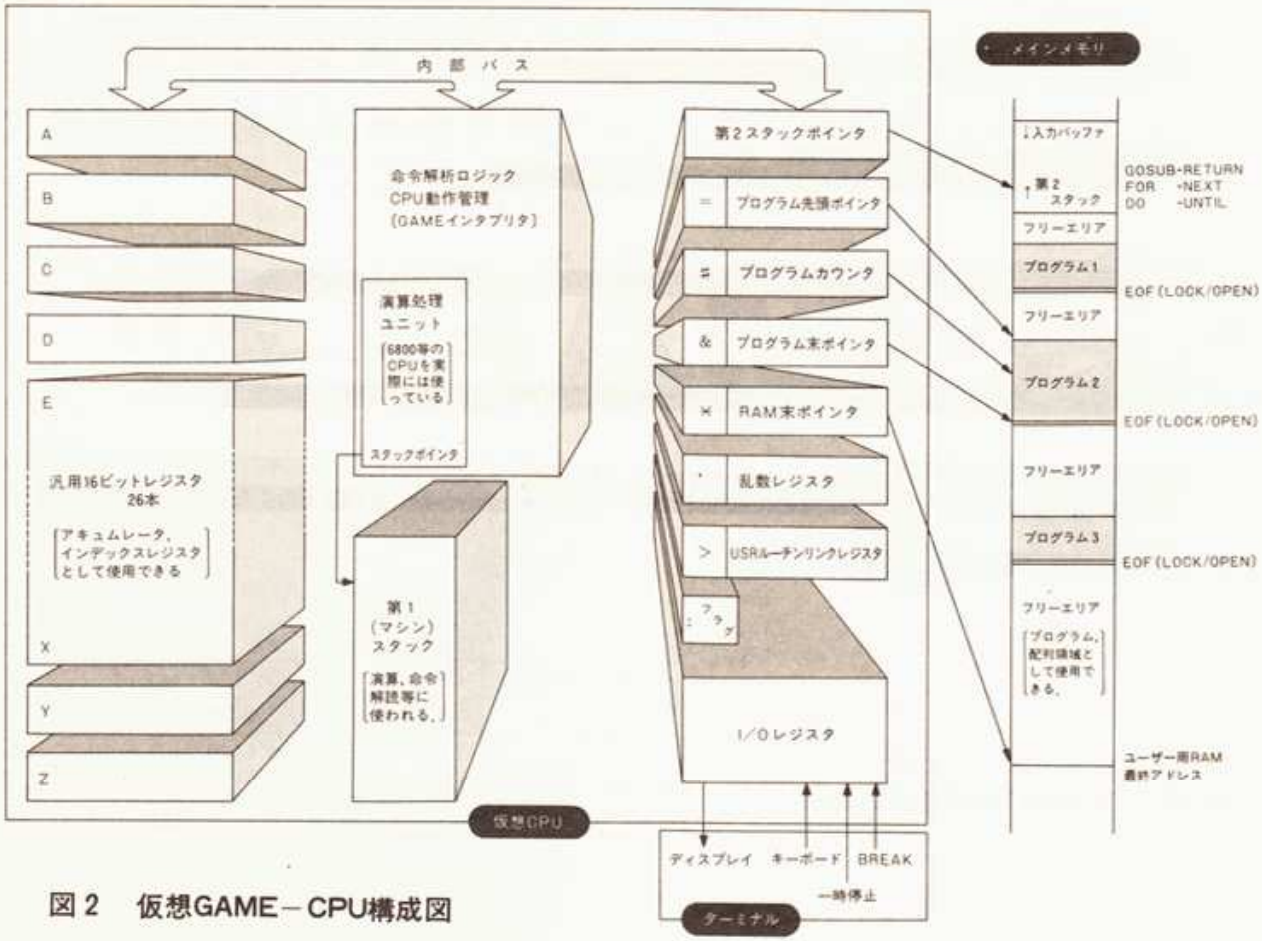

図2　仮想GAME－CPU構成図

プログラム管理

GAMEプロセッサには優れたエディット機能があります．ひとつのプログラムの修正変更等に限らず，このエディタは複数のプログラムの存在を許しています．たとえば，プログラムNo.1は$8000番地から，No.2は$A000番地から，といったようにメモリの許す限りいくつでもプログラムを置くことができます．

現在実行中・エディット中のプログラムがどれかを示すためにプログラム先頭ポインタ「=」があります．=レジスタの値がプログラム先頭アドレスを示すわけです．??==（=レジスタの値を16進4桁でプリント）とすればそのアドレスを読むことができ，=アドレス，とすれば任意のプログラムに移行することができます．

プログラムの最後を示しているのが&レジスタです．これはプログラムの製作中には自動的に更新され，また新しいプログラムに=命令で移った際はサーチエンド（==）によりプログラムの末尾を探してセットされます．

プログラムの最後（&ポインタの示すアドレス）にはEnd of File（EOF）マークが入っています．EOFには$FFと$FOの2種類が規定されており，$FOになっている場合プログラムの変更ができない（LOCK）状態になります．これによりプログラムを保護することができます．LOCK，OPENは，&：0）=$FFあるいは$F0によって行なえます．

レジスタはユーザー用のメモリエンドを示していますので，RAM容量を変更する場合，=最終アドレス，という形で行なうこともできます．プログラムを作成中に&レジスタの値が*レジスタの値より大きくなるとメモリーオーバーですのでそれ以上は作れなくなります．

エディタ自身はシステムにもよりけりですがほとんどBASICと同じ使い方ができます．

プログラムの実行

プログラムカウンタは#あるいは!で示されます．メモリから取り出された命令はインタプリタにより翻訳され実行されます．

実際の実行は68あるいはその他のCPUがインタプリタプログラムにより行なうわけですが，再三述べているようにここではGAMEプロセッサが直接に，A=B*C，といった命令を実行していると考えても結構です．

レジスタの使い方

レジスタ（変数）には大きく分けて2つの種類があります．ひとつは「&」「=」などの特殊レジスタ（システム変数）で，プログラムの管理を行ないます．もうひとつが汎用レジスタです．これはBASICの整数型変数と同様でA～Zまでの26のレジスタが開放されています．各々冗長形が許されるので同じ頭文字を使わないように注意すれば読みやすいプログラムを作ることができます．

また，レジスタをインデックスモードのベースに使うことにより配列を使うことができます．

例えばレジスタAの値を$8000にしたとします．（A=$8000）　すると，Aレジスタの示しているアドレス（$8000番地）を中心にしたインデックスモードでメモリを操作できます．A：0）とすると$8000の内容1バイト分，A（0）とすると$8000，$8001の2バイト分を示します．もちろん他の変数や&などのポインタをベースに使うこともできます．

配列宣言の方法はベースとしたいレジスタに値を代入すれば良いのです．たとえば，

A=&+1　　B=A+20

とするとAを使った配列エリアをプログラム末から20バイト確保し，Bを使ったエリアをその後に設定した，という意味になります．

BASICの配列とは違い，好きな位置に8bit／16bitのエリアを設定してアクセスすることができますのでキメ細かい作業やマシン語的タスクに向いている（だからこそ仮想CPUと呼ぶのですが……）と言えます．

2バイト数値をメモリに書き込む際68系GAMEは上位バイトがLOWアドレスになりますが80系と65系では反転します．

GAMEの命令セット

表1をご覧ください．これがGAMEIII基本仕様のインストラクションセットです．

後に開発されたGAME68や今回のMZ，PET版などは各々命令を強化してありますので各記事を参照してください．

命令は，プログラム・システムを管理するグループ，入出力を行なうグループ，演算グループ，ジャンプグループ，ループ処理を行なうグループ，等に大別できます．

システム管理命令の多くは特殊レジスタ（=，&等）を制御する形をとります．

入・出力，特に出力には多くのフォーマットが指定でき，16進表示なども使用できるため非常に使いやすいものとなっています．

演算は括弧優先の他は乗除算を含めて左から右へ順に行なわれますので注意してください．四則演算の他に関係演算も使えます．

ジャンプは，＃＝式の形で飛び先を指定できますので，トリッキィなプログラムを作れます．サブルーチンコールは「/」です．

ループの処理については少々くわしく説明しましょう．

GAMEのループ処理はFOR－NEXT型とDO－UNTIL型の2種類あります．

FOR－NEXT型は，

```
I=1, 10
……（プログラム）
@=I+1
```

という形で使います．最後の「I＋1」の部分を変えればループの増分パラメータを変えることができます．ここから強硬に抜け出すことは禁止されていますが，

```
@=10
```

を実行すれば併害なしに途中から脱出することができます．

DO－UNTIL型のループはある特定の条件が成立するまで繰り返すタイプのループ（これに対しFOR－NEXTはあらかじめ決まった回数を実行するタイプ）です．

```
@
……（プログラム）
@=（I=10）
```

という形で使います．この場合，Iが10になるまでその間のプログラムを繰り返します．

GAMEを使おう！

ここまででGAMEの特徴の一端がある程度理解していただけたと思います．実際に使ってみるとそのスピードにきっと満足し，またプログラムを作る場合にもいったん慣れてしまえば記号化言語の良さ（キーを押す回数が少ない！）がわかってくるでしょう．以下にMZ－80K，PET用のGAMEを紹介しますのでぜひお試しください．

表1　G A M E

インストラクションセット

・表中n, mは算術式が使用できる
・演算は乗除の優先はなく左から行なわれる．（ ）は優先．
行番号　1～32767　定数　10進　$16進
変数名　A～Z　若干の特殊文字（A，ALPHA，ACBは皆同じ）
配列については本文中の間接モード参照，プログラム中断，LOAD・SAVEはシステムにより異なる．

〈命　令〉	〈一般形〉	〈例〉	〈意　味〉
〈システム関係〉			
LIST ALL	0		・すべてのプログラムをリストする．
LIST FROM	行番号／	200／	・行番号の行から最後までリストする．
RUN	＃＝1		・プログラムの初めから実行．
RUN FROM 注	＃＝n	＃＝350	・nの値の行番号から実行（なければそれより大きい最初の行）
NEW	&＝0		・PG先頭アドレス(=)をPG末ポインター(&)に代入し，その位置にEOFマーク（$FF）を書き込む．
OPEN FILE	&：0)＝$FF		・EOFマークをOPEN状態（$FF）にする．
LOCK FILE	&：0)＝$F0		・EOFマークをLOCK($F0)(修正禁止)にする．
SEARCH END	＝＝		・PG末ポインター(&)をEOFマークのアドレスにセットする．
CHANGE PROGRAM	＝n	＝$1000	・PG先頭ポインター(=)をnのアドレスにし，PG末ポインター(&)をそこから最初にあるEOFマークにセットする．
BIGIN POINTER			
CHANGE RAM END	＊＝n	＊＝$A0FF	・RAM末ポインター(＊)をnのアドレスにセットする．
〈出力関係〉		n＝$1841のときの出力例(m＝5)	
PR LEFT	?＝n	6209	・nの値を10進左づめ出力
PR RIGHT	?(m)＝n	␣6209	・nの値をmケタで10進右づめ出力(1≦m≦255)
PR 4HEX	??＝n	1841	・nの値を4ケタの16進数で出力
PR 2HEX	?$＝n	41	・nの値の下位バイトを2ケタの16進数で出力
PR CHR	$＝n	A	・nの値の下位バイトを一文字のASCIIコードで出力
TAB	．＝n	．＝5	・nの値だけ空白を出力(0≦n≦255)
CRLF	／	////	・改行する
文字列出力	"文字"	/"AB"/CD/"	・"で囲まれた文字列を出力する．(改行子/と並用できる)
〈入力関係〉（入力変数）			
INPUT	?	A＝?	・変数として使用　「算術式」を一行入力してその値をとる．
INPUT CHR	$	A＝$	「一文字」入力してのASCIIコードを下位1バイトとする．(上位1バイトは0)
INPUT STR			・一行入力サブルーチンを使用
〈制御関係〉			
LET	変数＝n	A＝B＋C	・nの値を左辺の変数に代入する．
IF	：＝n	：＝A＜ ＞B	・nの値が真(≠0)なら次の「文」に行く．nの値が偽(＝0)なら次の「行に行く．
GOTO	＃＝n	＃＝200	・nの値の行番号へ行く．(n＝0なら何もしない)
END(STOP)	＃＝－1		・実行を終る．
GOSUB	!＝n	!＝300＋J	・文末位置を第2スタックに納めて，nの値の行番号へ行く．(n＝0なら何もしない)
RETURN	]		・文末位置を第2スタックから取り出し，そこに戻る．
CALL	＞＝n	＞＝$E010	・nの値のアドレスの機械語サブルーチンを呼ぶ．
FOR-TO	代入文，n	J＝1，100	・代入文を実行し，nの値を終値パラメーターとして，文末位置と共に第2スタックに納める．
NEXT(STEP)	@＝n	@＝J＋1	・nの値を＝の次の文字(制御変数)に代入し，その値が第2スタック中の終値パラメーターを超えていれば，次の文を実行する．越えていない場合は，第2スタック中の文末位置へ戻る．
		(FOR J＝1 TO 100 NEXT J に同)	
DO	@	@	・0を終値パラメーターとし文末位置と共に第2スタックに納める．
UNTIL	@＝(n)	@＝(K＝M)	・nの値が0(第2スタック中の終値パラメーター)より大なら（真）次の行へ行く．0が負なら（偽）第2スタック中の文末位置へ戻る．
		(K＝Mとなるまでくり返す．)	
〈その他〉			
SET RND	'＝n	'＝$F014	・乱数レジスターにnの値を設定する．
RND(n)	変数＝'n	A＝'6	・1～nまでの乱数を発生，変数に代入する．
REM	行番号S	100REM…	・行番号のあと空白を置かなければ注釈文となる．
MOD	n＝%m または n＝%	C＝%(A/B)	・除算のあまりをとる．バージョンにより使用法が異なる．
マルチステートメント	行番号 ␣S␣S	"A"␣A＝?	・空白で区切ればマルチステートメントになる．

注．RUN（＃＝）は変数をクリアしない．バージョンにより多少異なるため各記事を参照．
（この表は拡張機能は省略してあります）