

The Smalltalk-80 Virtual Machine

Glenn Krasner
Learning Research Group
Xerox Palo Alto Research Center
3333 Coyote Hill Rd
Palo Alto CA 94304

The Smalltalk-80 system is a powerful system that encourages the development of large applications programs. The system contains a compiler, a debugger, a storage management system, text and picture editors, and a file system. It also contains a highly interactive user interface based on graphics that include overlapping windows.

Typically the task of bringing up such a powerful system on a new computer includes writing code to implement these pieces. The Smalltalk-80 system is different in that most of these pieces are written in Smalltalk-80 itself. The part that can be written in Smalltalk-80 is called the *Smalltalk-80 Virtual Image*, and it includes the compiler, debugger, editors, decompiler, and the file system.

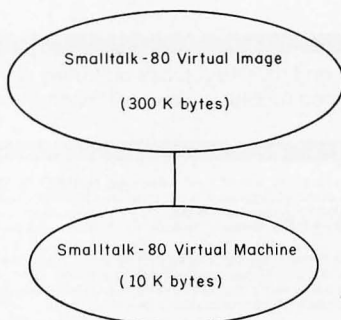


Figure 1: The *Smalltalk-80 Virtual Machine*. Most of *Smalltalk-80* is written in *Smalltalk-80* (the *Virtual Image*), leaving only a small amount of code that has to be rewritten for each processor on which the language is implemented (the *Virtual Machine*).

The remaining part of the Smalltalk-80 system is defined in terms of an abstract machine called the *Smalltalk-80 Virtual Machine* (see figure 1). The Smalltalk-80 compiler translates source code into machine instructions for this virtual machine, rather than translating directly into machine instructions for a particular hardware machine. The task of bringing up a Smalltalk-80 system on a new "target" computer consists only of implementing (writing a program to simulate) the Smalltalk Virtual Machine on the target computer.

In this article, we will present an overview of the elements needed to implement the Smalltalk Virtual Machine. These elements are:

- the *Storage Manager*
- the *Interpreter*
- the *Primitive Subroutines*

Background

A Smalltalk-80 system is made up of *objects* that have state and exhibit behavior. Their state consists of the values of both named and indexed instance variables (which we will call *fields*), and their behavior is exhibited through sending and receiving *messages*. Objects are members of *classes*.

Classes may be *subclasses* of other classes—that is, they may inherit attributes from other classes. Programming in Smalltalk-80 is done by defining the procedures, or *methods*, that are executed when objects receive messages. Typically, messages are

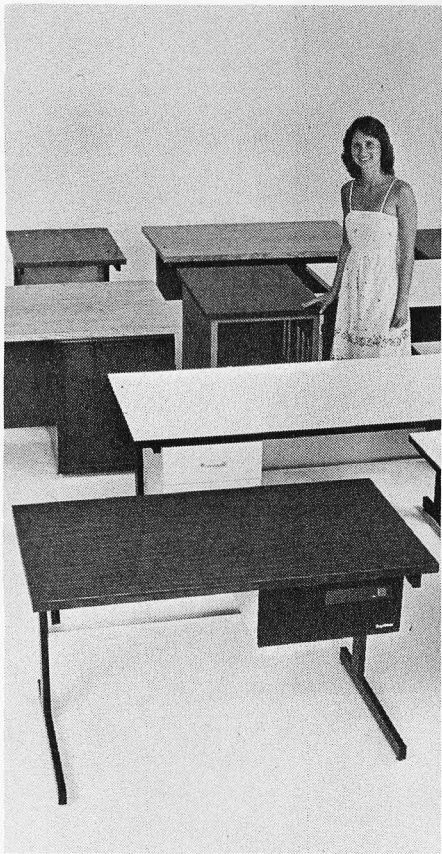
sent to other objects to invoke their methods. Sometimes messages invoke *primitive* (machine-code) *subroutines* rather than Smalltalk-80 methods.

From this brief description of Smalltalk-80, we can consider the information needed to implement each of the three elements of the Smalltalk Virtual Machine:

1. To implement the storage manager, we need the information necessary to represent objects in the computer's memory. This information consists of the amount of memory that each object will occupy, which can be computed from the number of fields the object has, and the representation of fields in memory. Objects that describe classes define the number of fields their instances will have, so we also need to know how this number is represented. With this information, we can design a storage manager for objects in a Smalltalk-80 system that will:

- fetch the class of objects
- fetch and store fields of objects
- create new objects
- collect and manage free space

2. The interpreter executes the machine instructions of the Smalltalk-80 Virtual Machine. The information needed to design the interpreter is a description of these machine instructions, called *bytecodes* (the idea is similar to Pascal p-codes). The bytecodes are contained in methods, so we also need to know the representa-



SIMPLY BEAUTIFUL.

CF&A furniture looks terrific. But beauty is more than skin deep. That's why our line of desks, stands, and enclosures also features rugged construction, low cost, and quick delivery. In a wide range of sizes and configurations. With accessories to meet your individual requirements. With a smile and a thank you.

Call CF&A. We make it simple. We make it beautiful.

CF&A

**Computer Furniture and
Accessories, Inc.**
1441 West 132nd Street
Gardena, CA 90249
(213) 327-7710

Who's Who

The design of the Smalltalk-80 Virtual Machine is based on previous Smalltalk systems implemented by the Learning Research Group at Xerox PARC. The original bytecode interpreter design was made for Smalltalk-76 by Dan Ingalls (Ingalls, Dan. "The Smalltalk-76 Programming System: Design and Implementation." In Fifth Annual ACM Symposium on Principles of Programming Languages, 1978, pages 9 through 16). Smalltalk-76 was implemented on the Xerox Alto by Dan Ingalls, Ted Kaehler, Dave Robson, Steve Weyer and Diana Merry, on the Xerox Dolphin by Peter Deutsch, and on the Xerox Dorado by Bruce Horn. TinyTalk was implemented on a Xerox microcomputer by Larry Tesler and Kim McCall (McCall, Kim and Larry Tesler. "Tiny Talk, a Subset of Smalltalk-76 for 64KB Microcomputers." In Proceedings of the Third Symposium on Small Systems, ACM Sigsmall Newsletter, Volume 6, Number 2, 1980, pages 197 through 198). Smalltalk-78 (a revised version of Smalltalk-76 similar to Smalltalk-80) was implemented on the Xerox microcomputer by Dan Ingalls, Ted Kaehler, and Bruce Horn, on the Xerox Dorado by Jim Stamos, and on a Norwegian microcomputer (under a research license from Xerox) by Bruce Horn. Smalltalk-80 has been implemented on the Xerox Dorado by Peter Deutsch, on the Xerox Dolphin by Kim McCall, and on the Xerox Alto by Glenn Krasner. The designs of these systems were made by the implementors and other members of the Learning Research Group.

tion of methods. From this information we can decide how the interpreter will fetch and execute bytecodes and how it will find methods to run when messages are sent.

3. The last piece of information we need to know is which messages will invoke primitive subroutines; that is, which methods we must implement in machine code to terminate the recursion of message sending and to optimize performance.

Before we go into more detail about these elements of a Smalltalk-80 Virtual Machine implementation, here are a few typical figures that will provide a little "reality" to implementors. For the systems that we have implemented at Xerox, the Smalltalk-80 Virtual Image consists of about 300 K bytes of objects. Our typical implementation of the Smalltalk-80 Virtual Machine is 6 to 12 K bytes of assembly code, or 2 K microcode instructions plus 10 K bytes of assembly code. Of this, about 40% is in the storage manager, 20% in the interpreter, and 40% in the primitive subroutines. Our average is about one person-year to implement a fully debugged version of this code.

The Storage Manager

Although the storage manager tends to be the largest and most complex of the three parts of a Small-

talk-80 implementation, the functions it provides are few and relatively simple to understand.

Everything in a Smalltalk-80 system is an object.

Everything in a Smalltalk system is an object, so from a storage point of view memory needs to be divided into blocks, one for each object, plus a pool of memory that is not yet used. Every time a new object is created, a new block of the appropriate size must be found for that object: when objects are no longer used, their memory block may be returned to the pool (see figure 2).

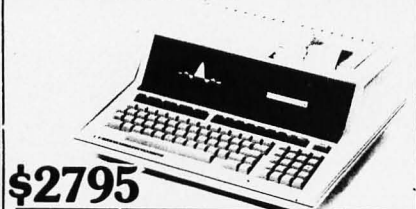
A special entity called an *object pointer* is assigned to each object. If an object pointer were the actual core address of the memory occupied by that object, then there would be fast access to an object given its pointer. However, in the Smalltalk-80 system the object pointer is an *indirect pointer* to the object through a table kept by the storage manager. This allows the storage manager to move an object around in memory without affecting any object that refers to it. It also insures that the storage manager is the only entity in the system concerned with (and allowed to change)

Lowest Prices on Personal Computers



ATARI® 400...\$359

hp HEWLETT PACKARD HP-85



HP-85 Accessories

- 5 1/4" Dual Master Disc Drive List \$2500... \$2125
- 5 1/4" Single Master Disc Drive List \$1500... \$1275
- HP 7225A Graphics Plotter List \$2050... \$1845
- HP-85 16K Memory Module List \$395... \$355
- HP-85 Application Paces Standard List \$95... \$85
- Serial (RS-232C) Interface Module List \$395... \$355
- GPIO Interface Module List \$495... \$445

new HP-83 List \$2250 **\$1895**

NEW! HP-41CV with five times more memory built in.

List \$325 **\$249**
 HP-41C List \$250 **\$199**

- HP-32E Scientific w/Statistics — 53.95
- HP-33C Scientific Programmable 79.95
- HP-34C Advanced Scientific Programmable 123.95
- HP-37E Business Calculator — 49.95

Personal PC Computer Systems

609 Butternut Street
 Syracuse, N.Y. 13208
 (315) 475-6800

Prices do not include shipping by UPS. All prices and offers subject to change without notice.

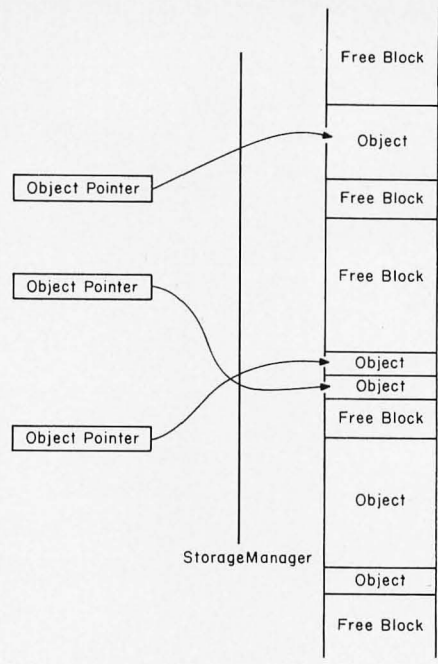


Figure 2: Objects and memory usage in Smalltalk-80. Each Smalltalk-80 object has an object pointer that points to a block of memory that describes the object. When an object is no longer used, its memory is made available for use.

(Length) 4
(Class description) Point
(x-coordinate)
(y-coordinate)

(Length) 5
(Class description) Triangle
(First vertex)
(Second vertex)
(Third vertex)

(Length) 4	
(Class description) ByteArray	
1	2
3	4

Figure 3: Typical object representations in Smalltalk-80.

the actual memory. In the Smalltalk-80 Virtual Image, object pointers are single 16-bit words. This allows for 64 K objects in the system; these objects may take up much more than 64 K words of memory.

Since an object's class and fields are themselves objects, we can see that the block of memory corresponding to an object contains the object pointer of the object's class plus the object pointer for each of the object's fields. The storage manager also keeps the length of the block as one word of the block. This means, for example, that the block corresponding to an object that is an instance of class Point (see figure 3) will have:

- one word that says this block is four words long
- one word that is the object pointer of the object that describes class Point
- one word that is the object pointer of an object that is the x-coordinate field of the point
- one word that is the object pointer of an object that is the y-coordinate field of the point

Similarly, the block corresponding to an object that is an instance of class Triangle will have:

- one word saying this block is five words long
- one word that is the object pointer of the object that describes class Triangle
- one word that is the object pointer of an instance of class Point, representing one vertex field
- one word that is the object pointer of an instance of class Point, for the second vertex field
- one word that is the object pointer of an instance of class Point, for the third vertex field

For performance optimization, the values in the fields of some objects, such as instances of class ByteArray, will be interpreted as the numerical values themselves, rather than as object pointers. The block corresponding to the byte array containing the elements 1, 2, 3, and 4, in order, will have:

- one word saying this block is four words long
- one word pointing to the object that describes class ByteArray
- one byte encoding the number 1
- one byte encoding the number 2
- one byte encoding the number 3
- one byte encoding the number 4

We will represent all objects as having fields interpreted as object pointers or numerical values, not both. Objects may store numerical values as bytes or words, but not both.

As we have mentioned, the objects that describe classes also need to represent the form of instances of those classes. The essential information is the number of fields the instances will have, and whether these will be pointer or nonpointer fields. For example, the describer of class Point says that its instances will have two fields (*x*- and *y*-coordinates) and that these will be pointers (see figure 4). The describer of class ByteArray says that its instances may

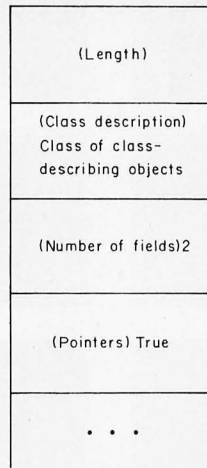


Figure 4: Class-describing object for class Point.

have a variable number of fields and that these fields will not be pointers but will be numerical values stored in bytes.

The purpose of the storage manager is to fetch and store fields of objects, to create objects, and to manage free space. A clean implementation of the storage manager

would be one in which the other parts of the system had access only to the object pointers and made requests of the storage manager only through the following subroutine calls:

- `getClass(objectPointer)` returns the object pointer of the class of the given object
- `getField(objectPointer,fieldOffset)` returns the field
- `storeField(objectPointer,fieldOffset,newValue)` replaces that field with the new value `newValue`
- `newInstance(classObjectPointer,numberOfFields)` returns the object pointer of a new instance of that class, and, if that class can have indexed instance variables, this instance has the given number of fields (`numberOfFields`)

Requests can be made for new storage (with the `newInstance` subroutine), but not to return used storage. In some other systems, storage that is no longer used must be explicitly returned to the free storage pool. The Smalltalk-80 philosophy is that neither the user nor any part of the system other than the storage manager need have such concerns. Therefore the storage manager must know which objects are no longer being used, so that their storage may re-enter the free pool. Typically, Smalltalk-80 Virtual Machine implementations use *reference-counting* to accomplish this. For every object in the system, the storage manager keeps a count of the number of other objects that point to it. This number will change only during execution of the four storage-manager subroutines. When this count reaches zero, the object's memory block may be reused because there are no references to that object anywhere else in the system.

The Interpreter

The interpreter is that portion of the Smalltalk-80 Virtual Machine that performs the actions described in the bytecodes of methods (ie: the machine code of the Virtual Machine). The information needed to implement the interpreter is the

FINDING SOLUTIONS AND BEING COMPETITIVE IS OUR BUSINESS.

Having problems and looking for a computer to help solve them? Are you finding computer dealers come in one of two ways? Either Full system support with Full price or Take it or Leave it with Low price. At Omega we don't believe that you should have to make a choice. Yes, we're in business to sell products but also, to solve your problems. Our prices will be the lowest possible. Our support and product quality will be second to none. Check out our Mail Order prices in this ad (our retail prices will be higher). See if you don't agree with our first claim. For our second claim, call us with your data processing needs and problems. Better yet, come in and see us. Finding solutions and being competitive is our business. We never forget either of them.



APPLE III	SCALL
APPLE II "PLUS" 48K	\$ 1129.00
HEWLETT PACKARD 85 or 83	SCALL
APPLE II ACCESSORIES:	
Disk II with controller	\$ 515.00
Disk II 2nd drive	452.00
Graphics Tablet	665.00
Language System with PASCAL	385.00
Silentye Printer W/Int	526.00
Integer Firmware Card	152.00
Microsoft Z-80 Softcard	259.00
Videx videoterm 80 col Card	256.00
Sanyo 12" Green Monitor	269.00

Supplies:	
Scotch Diskettes - Best of Quality!	
price per box of (10)	
744-0, 10, 32	\$ 27.00
740-0 SS/SD 0 Sector	31.00
741-0 SS/DD 0 Sector	37.00
743-0 DS/DD No Format	44.00
New Products:	
Videx L/C Adapter	110.00
Microsoft 16K RAM Card	169.00
D.C. Hayes Micromodem II	307.00
Novation DCAT Modem	195.00
EPSON MX-80	CALL

Mail Order Terms of Sales: Price based on prepaid orders. NO COD's. Allow 14 working days for personal and company checks to clear. Order under \$100.00 add \$3.00 for shipping and handling. All orders (unless specified in ad) within Continental U.S. shipped U.P.S. no charge. APO or out of Continental U.S. write or call for shipping charges. All prices subject to change and all offers subject to withdraw without notice. CA residents add 6% sales tax.

OMEGA MICRO COMPUTERS

The Problem Solving Company 3447 Torrance Boulevard • Torrance, California 90503 • (213) 328-1760

description of the bytecodes, the representation of methods, and the technique to find the method to run when sending a message.

The bytecodes define the Smalltalk-80 Virtual Machine as a stack-oriented machine. Each bytecode represents one of the following actions:

- push an object onto the stack
- store the top of the stack as the value for a variable
- pop the top of the stack
- branch to another bytecode
- send a message using the top few elements of the stack
- return the top of the stack as the value for this method

In the Smalltalk-80 Virtual Machine, each of these actions is realized by one or more bytecodes. Note that pushing, storing, popping, and branching are standard instruction types for any stack machine, that sending a message corresponds to calling a procedure using the top few

Bytecode	Stack Contents After Execution (Top of Stack to Right)
-1- Push 3	(3)
-2- Push 4	(3 4)
-3- Push 5	(3 4 5)
-4- Send +	(3 9)
-5- Send *	(27)

Table 1: Bytecodes for the Smalltalk expression $3 * (4 + 5)$.

elements of the stack as arguments, and that returning an object from a method corresponds to returning a value from a procedure. The difference between the Smalltalk-80 Virtual Machine and procedure-based stack machines is in the way the procedure is found. In most procedure-based stack machines the address of a procedure is provided in the *execute procedure* instruction; in the Smalltalk-80 system only the "name," called the *selector*, of the message is provided; the method (or procedure) to be executed is found through a strategy involving the receiver of the message and its class. We will first describe the bytecodes, then how

methods are represented, and finally give a strategy for finding methods.

Stack Operations

The Smalltalk-80 Virtual Machine and corresponding bytecode set are stack oriented. Object pointers are pushed and popped from a stack, and when a message is sent, the top few elements of the stack are used as receiver and arguments of the method. These are replaced by the object returned as the value of that method. For example, the Smalltalk-80 expression:

$$3 * (4 + 5)$$

is encoded by the bytecodes shown in table 1.

As bytecodes labeled -1-, -2-, and -3- are executed by the interpreter, the objects 3, 4, and 5 are pushed onto the stack. When bytecode -4- is executed, the message + is sent to the second object on the stack (4) with the top object of the stack as the argument (5). The 4 and 5 are popped off this stack when the message is sent, and the interpreter begins executing the bytecodes for the method corresponding to the message + in the Smalltalk class of small integers. This method will eventually return an object, in this case 9, as its value, and the interpreter will push the 9 onto the original stack above the 3 and resume execution with bytecode -5-. Bytecode -5- will produce an effect similar to that produced by -4-, leaving the object 27 on the stack. In the same way that other stack machines push *data* onto a *stack* and use the top few data items as *arguments* for a *procedure*, replacing them with the *value* returned from that procedure, the Smalltalk-80 Virtual Machine pushes *object pointers* onto a *stack*

Happy Hands

Offers Discounts on All

TRS-80TM

COMPUTERS

Call Collect For Prices
And Shipping Schedules

505-257-7865

HAPPY HANDS

P.O. DRAWER I

RUIDOSO, NEW MEXICO

88345

We Have What You Are Looking for

PROMPT SHIPPING

AVAILABLE SERVICE CONTRACTS

DISCOUNTED PRICES COMPARE TO ANY OTHERS

NO TAX ON OUT OF STATE SHIPMENTS

or write

and uses the top few as *receiver* and *arguments* of a *message*, replacing them with the *object* returned from that method.

In both machines, values from the top of the stack may be stored as the values of variables. As an example, the Smalltalk expression:

$$a \leftarrow 3 + 4$$

will be represented by the bytecodes in table 2. Here, -1-, -2- and -3- act as before and the interpreter executes bytecode -4- by storing the top of the stack 7 into the variable *a*.

Stack machines in general, and the Smalltalk-80 Virtual Machine in particular, also have the ability to pop the top element off the stack. In the statements:

$$\begin{aligned} a &\leftarrow 3. \\ b &\leftarrow 4 \end{aligned}$$

once the 3 is stored into variable *a*, it is no longer needed, so it is popped from the stack. These statements are represented by the bytecodes shown in table 3.

The top of the stack may be returned as the value for the method. The statements:

$$\begin{aligned} a &\leftarrow 3. \\ \uparrow a \end{aligned}$$

are represented by the bytecodes shown in table 4.

Branching Operations

Conditional and looping messages are used so often that they are represented not by actual messages but by bytecodes for conditional and unconditional jumps. (This is *only* for performance reasons; these branching and looping messages would work if they were actually sent like other messages.) For example:

$$a > 4 \text{ ifTrue: } [a \leftarrow a - 1]$$

(which in the Smalltalk-80 system means execute the code within the brackets only if the object returned from the *>* message is not *false*) is represented in table 5 (ignoring the stack from now on).

Bytecode	Stack Contents After Execution (Top of Stack to Right)
-1- Push 3	(3)
-2- Push 4	(3 4)
-3- Send +	(7)
-4- Store into a	(7)

Table 2: Bytecodes for the Smalltalk expression $a \leftarrow 3 + 4$.

Bytecode	Stack Contents After Execution (Top of Stack to Right)
-1- Push 3	(3)
-2- Store into a	(3)
-3- Pop	()
-4- Push 4	(4)
-5- Store into b	(4)

Table 3: Bytecodes for the Smalltalk expression $a \leftarrow 3. b \leftarrow 4$.

Bytecode	Stack Contents After Execution (Top of Stack to Right)
-1- Push 3	(3)
-2- Store into a	(3)
-3- Pop	()
-4- Push a	(3)
-5- Return top of stack	()

Table 4: Bytecodes for the Smalltalk expression $a \leftarrow 3. \uparrow a$.

DISCOUNT PRICES

Microcomputers & Peripherals



North Star • SWTPC • Lear-siegler
Hazeltime • Centronics • Cromemco
Wabash • Perkin Elmer and others

Fast, off the shelf delivery.
Call TOLL FREE 800/523-5355

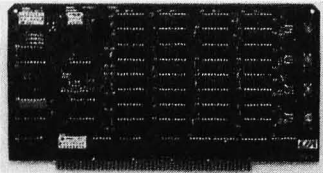
MARKETLINE SYSTEMS, Inc.
2337 Philmont Ave., Huntingdon Valley, Pa. 19006
215/947-6670 • 800/523-5355

Dealer Inquiries Invited

Factory Direct

ELIMINATE THE MIDDLE MAN!

16K^{STATIC} RAM



RAM 16

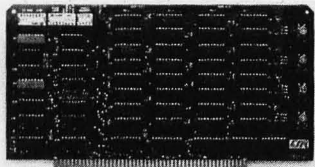
COMPARE OUR FEATURES!

S-100 • 16K X 8 Bit Static RAM • 2114 1K X 4 Static RAM Chip • 2 or 4 MHZ • 4K Step Addressable • 1K Increment Memory • Protection from Top Board Address Down or from Bottom Up • Deactivates up to 6 1K Board Segments to Create Holes for Other Devices • Phantom Line DIP Switch • DIP Switch Selectable Wait States • 8 Bank Select Line Expands to 1/2 Million Bytes • All Data Address and Control Lines Are Input Buffered • Ignores I/O Commands at Board Address • **Our Most Popular Board — Over 5000 Now in Use Worldwide!**

A & T Factory Direct Price:

\$175.00

One Year Warranty
DON'T PAY MORE!



NEW! — RAM 65

All of the Above Advanced Features PLUS: Bank Selection by I/O Instruction Using Any One of 256 DIP Switch-Selectable Codes • This Allows Up to 256 Software Controlled Memory Banks! • **Our Most Advanced 16K Add-On Board.**

A & T Factory Direct Price:

\$185.00

One Year Warranty
DON'T PAY MORE!

All of our Boards Are the Highest Quality MIL SPEC G-14 Fibreglass. All utilize Solder Mask over Copper Technique for Higher Reliability!
MIX AND MATCH FOR BEST PRICING. Include \$3.00 shipping & handling per order. California residents include 6% sales tax.
TERMS: Cash, checks, money orders or purchase orders from qualified firms or institutions. Pricing and availability subject to change without notice. International sales in U.S. funds only. COD's include 25% with order.

Quality Computer Parts
P.O. BOX 743, DEPT. B1
CHATSWORTH, CA 91311
Telephone (213) 882-3142

Bytecode

- 1- Push 4
- 2- Push a
- 3- Send >
- 4- Jump to -10- if the top of the stack is false
- 5- Push a
- 6- Push 1
- 7- Send —
- 8- Store into a
- 9- Pop
- 10- < the next bytecode >

Table 5: Bytecodes for the Smalltalk expression $a > 4$ ifTrue: [a ← a - 1].

Bytecode

- 1- Push a
- 2- Push 4
- 3- Send >
- 4- Jump to -11- if top of stack is false
- 5- Push a
- 6- Push 1
- 7- Send —
- 8- Store into a
- 9- Pop
- 10- jump to -1-
- 11- < the next bytecode >

Table 6: Bytecodes for the Smalltalk expression $[a > 4]$ whileTrue: [a ← a - 1].

Table 6 shows the bytecodes for the looping expression:

$[a > 4]$ whileTrue: [a ← a - 1]

(which means execute the code in the second brackets as long as the code in the first set of brackets evaluates to something other than false).

Addressing Variables

Methods are implemented as objects whose fields contain the bytecodes plus a group of pointers to other objects called the *literal frame*. The interpreter can use the getField subroutine of the storage manager to fetch the next required bytecode to execute. This takes care of returns, jumps, and pops, but for the other bytecodes we need to represent more information. In particular, for the push and store bytecodes, we need to represent where to find the object pointers to push; for the send bytecodes, we need to represent where to find the selector of the message and which stack elements are

the receiver and arguments.

The source code for a method contains variable names and literals, but the bytecodes of the Virtual Machine are defined only in terms of field offsets. From the Virtual Machine's point of view, there are three types of variables: variables local to the method (called *temporaries*), variables local to the receiver of the message (*instance variables*), or variables found in some dictionary that the receiver's class shares (*global variables*). Note that *class variables* are treated in the same way as other global variables. The Smalltalk-80 compiler (itself written in Smalltalk-80) translates references to these variables into bytecodes that are references to field offsets of the receiver, the temporary area, or globals. The instance variables are translated using a field of class-describing objects that associates instance variable names with field offsets. The assignment of offsets to temporaries is done when the compiler translates a method by associating

WHAT'S BETTER THAN AN ISAM

And Will Turn

MICROSOFT'S
BASIC
COBOL
FORTRAN

DIGITAL'S
PL/I-80

CBASIC
PASCAL/MT+
S-BASIC

CROMEMCO 16K BASIC

into first class application languages?

MICRO B+™

The first and most complete implementation of **B-TREE** index structures for micro-computers. **B-TREES** eliminate index file reorganization.

Search

An index of over

10,000 Key Values In Less Than One Second

On A Floppy Disk System for only

\$260.00!

System Houses:

MICRO B+™

Available in Language C

FAIR COM

2606 Johnson Drive
Columbia, MO 65201
(314) 445-3304

©1980 Fair Com

Shipping \$4 USA / \$8 Foreign
We accept VISA and MASTERCARD

PL/I-80 is a trademark of Digital Research
CBASIC is a trademark of Compiler Systems, Inc.
S-BASIC is a trademark of Topaz Programming
PASCAL/MT+ is a trademark of MT Micro Systems

names of temporaries to offsets in the temporary area. The compiler creates instances for the literals, puts their object pointers into the literal frame of the method, and produces bytecodes in terms of offsets into the literal frame. For global variables, the compiler uses system dictionaries that

associate global names to indirect references to objects. Object pointers of the indirect references to the global objects are also placed in the literal frame of the method. The bytecodes for accessing globals are encoded as indirect references through field offsets of the literal frame.

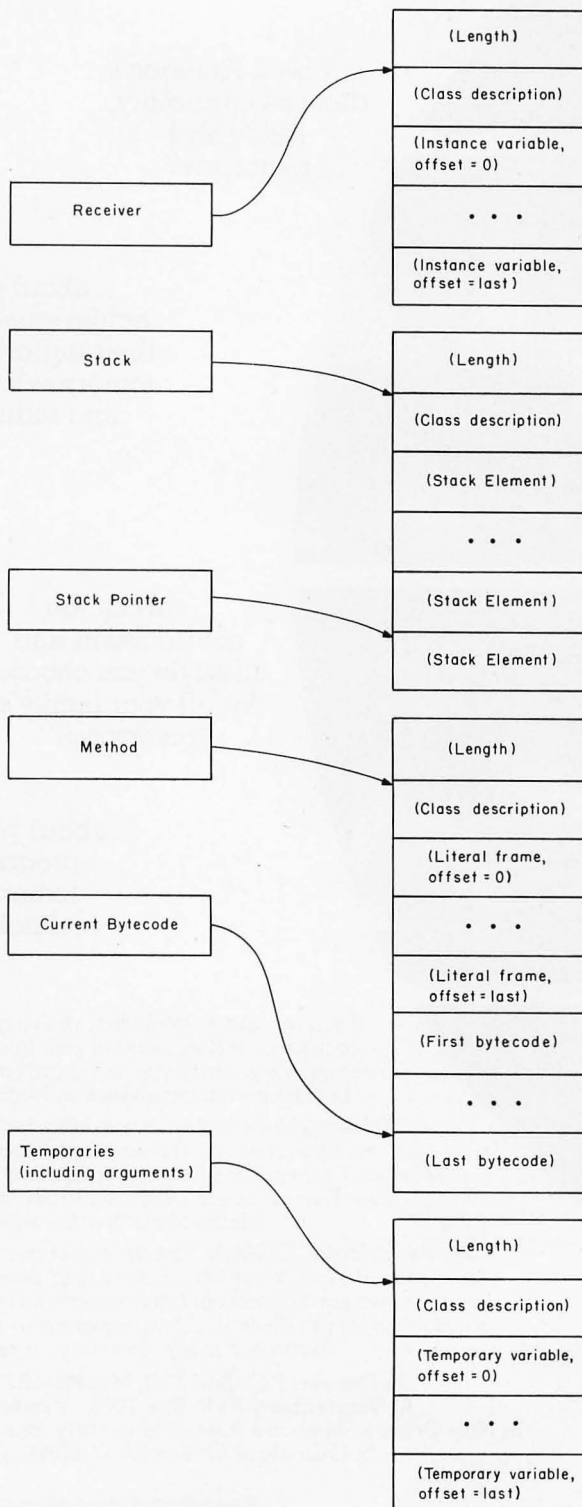


Figure 5: Object pointers held by the interpreter.

APPLESOFT® Basic Compiler

\$167.50

Compiles APPLESOFT® BASIC programs into native 6502 code, allowing programs to run up to 10 times faster. Handles graphics and shape tables. Requires 48K, autostart ROM, language system and at least one drive. List Price: \$200.00

This means that when the interpreter is executing a method, it has to keep a stack, a temporary area, a pointer to the receiver and arguments of the method, and a pointer to the method itself (see figure 5). It uses the storage manager's *getField* and *storeField* subroutines to push and pop pointers from the stack object, to retrieve and set values of variables in the temporary area, to retrieve and set values of variables of the receiver, and to get bytecodes and values of global variables from the method.

Finding Methods

When a message is sent, the receiver and arguments must be identified, and the appropriate method must be found by the interpreter. The technique used in Smalltalk-80 is to include in each class-describing object a dictionary, called the *method dictionary*, that associates selectors with methods. Pointers to the selectors that will be sent by any method are kept in the method (along with global variable pointers and bytecodes). The bytecodes that tell the interpreter to send a message encode a field offset in the literal frame where the selector is found, plus the number of arguments that that method needs. By convention, the top elements of the stack are the arguments and the next one down is the receiver. For example, the send bytecode for the expression:

3 + 4

will stand for "send the selector in field X of the method (which will be +), and it takes one argument." The interpreter will ask the storage manager for the X field of the method, will get the top of the stack (4) as the argument, and the next element down (3) as the receiver. It will locate the receiver's class, its method dictionary, search it for an association of the + selector with some method, and, when found, execute that method.

If no such association is found, the searching does not end. The receiver's class may be a subclass of another class, called its *superclass*. If this is the case, the method for + may be

Microhouse™

SPELLSTAR

New! Option for Wordstar. Compares words in your text to its 20,000-word compressed dictionary. Jumps back to WordStar for correction of errors or addition of new words to the dictionary. Price includes update of registered 2.x and earlier WordStar (must send master disk).
List Price: \$250.00
Microhouse Price: \$165.00/NA

WORDSTAR

Version 3.0! Now featuring horizontal scrolling and column moves.
List Price: \$495.00
Microhouse Price: \$322.00/\$40.00

WORDSTAR for APPLE

List Price: \$375.00
Microhouse Price: \$245.00/\$40.00

WORDSTAR CUSTOMIZATION NOTES

Package includes manual and diskette.
List Price: \$150.00
Microhouse Price: \$95.00/NA

MAILMERGE

Option for Wordstar.
List Price: \$150.00
Microhouse Price: \$110.00/\$40.00

MAILMERGE for APPLE

List Price: \$125.00
Microhouse Price: \$85.00/\$25.00

VISICALC for APPLE

List Price: \$150.00
Microhouse Price: \$107.00/NA

SUPERSORT I

List Price: \$250.00
Microhouse Price: \$170.00/\$40.00

SUPERSORT II

List Price: \$200.00
Microhouse Price: \$145.00/\$40.00

SUPERSORT for APPLE

List Price: \$200.00
Microhouse Price: \$130.00/\$40.00

DATASTAR

List Price: \$350.00
Microhouse Price: \$245.00/\$40.00

WORDMASTER

List Price: \$150.00
Microhouse Price: \$119.00/\$40.00

CALL OR WRITE FOR FREE CATALOG

MICROSOFT BASIC 80

List Price: \$350.00
Microhouse Price: \$273.00/\$30.00

MICROSOFT BASIC 80 COMPILER

Language compatible with MBASIC but code runs 3-10x faster.
List Price: \$395.00
Microhouse Price: \$308.00/\$30.00

MICROSOFT FORTRAN 80

Compiler is ANSI '66 compatible (except for COMPLEX).
List Price: \$500.00
Microhouse Price: \$345.00/\$30.00

muSIMP/muMATH by Microsoft

List Price: \$250.00
Microhouse Price: \$195.00/\$25.00

COBOL 80 by Microsoft

List Price: \$750.00
Microhouse Price: \$562.50/\$30.00

MACRO 80 by Microsoft

List Price: \$200.00
Microhouse Price: \$140.00/\$20.00

EDIT 80 by Microsoft

List Price: \$120.00
Microhouse Price: \$84.00/\$20.00

WHITESMITHS C

List Price: \$630.00
Microhouse Price: CALL/\$30.00

TINY C Interpreter

List Price: \$100.00
Microhouse Price: \$79.00/\$50.00

TINY C Compiler

List Price: \$250.00
Microhouse Price: \$195.00/\$50.00

SPELLGUARD

Fast stand-alone program works with nearly any CP/M® word processor.
List Price: \$295.00
Microhouse Price: \$230.00/\$25.00

STACKWORKS FORTH

For Z80 or 8080 (specify).
List Price: \$175.00
Microhouse Price: \$160.00/\$30.00

WHITESMITHS PASCAL

Includes Whitesmiths C Compiler.
List Price: \$850.00
Microhouse Price: CALL/\$45.00

Microhouse

P.O. BOX 498
BETHLEHEM, PA 18016
(215) 868-8219

(Length) 7
(Class description) Class of class- describing objects
(Number of fields) 2
(Pointers) True
(Instance Variable Names) "xCoordinate yCoordinate"
(Global Variable Dictionaries)
(Method Dictionary)
(SuperClass)

Figure 6: Class-describing object for class Point, revisited.

defined in the superclass, so the interpreter must check there. This means that each class must have a field that refers to its superclass (see figure 6). The interpreter searches the method dictionary of the superclass, its superclass, and so on, until either an appropriate method is found or it runs out of superclasses, in which case an error occurs.

To execute a method, the interpreter needs a place for temporaries and a stack for that method. In the Smalltalk-80 Virtual Machine, this is done by allocating an object that is an instance of class MethodContext. Objects in MethodContext keep track of the method, the stack for that method, a pointer to the next bytecode to be executed in that method, the temporary variables for that method, and the context from which that method was invoked, called the caller of that method (see figure 7). When a method returns, the value returned is pushed on the stack of the caller context, and execution continues at the next bytecode of the caller's method.

The New ADDS Viewpoint Video Terminal. \$649

Features a detachable Tektronics-made keyboard with keypad. Function keys. Reverse video, half-intensity, underlining by fields. Printer port. ADDS quality construction.

Microhouse™

CALL FOR LOW PRICES ON IMS Series 5000 and 8000 Computers

C ITOH STARWRITER I

Letter-quality printer uses Diablo plastic printwheels and ribbons. 25 cps bidirectional, logic-seeking. Parallel interface.

List Price: \$1895.00
Microhouse Price: \$1431.00

C ITOH STARWRITER I (Serial)

List Price: \$1960.00
Microhouse Price: \$1502.00

DIABLO 630

Letter-quality printer uses plastic and metal printwheels. 40 cps, bidirectional, logic-seeking. Optional tractor: \$225.

List Price: \$2710.00
Microhouse Price: \$1999.00

EPSON MX70

Includes GRAFTRAX II dot-addressable graphics. Monodirectional. 80 cps. Adjustable tractor. Parallel only.

List Price: \$450.00
Microhouse Price:

EPSON MX80

Removable print head, bidirectional, logic-seeking, adjustable tractor, parallel interface. Easily converted to RS232, IEEE 488, Apple or ATARI. CALL FOR INFORMATION ON THE NEW GRAPHICS ROM PACK!

List Price: \$645.00
Microhouse Price:

EPSON MX-80 FT

Friction AND tractor feed version of the MX-80. Parallel interface included.

List Price: \$745.00
Microhouse Price:

TELEVIDEO 910 Terminal

List Price: \$699.00
Microhouse Price: \$595.00

TELEVIDEO 950

List Price: \$1195.00
Microhouse Price: \$995.00

CALL OR WRITE FOR FREE CATALOG

PRICES AND SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE



Software Manual & Manual/Only

ASK ABOUT THE NEW Televideo COMPUTER SYSTEMS

IDS PAPER TIGER 560G

List Price: \$1695.00
Microhouse Price: \$1464.00

VIDEX VIDEOTERM SPECIAL!

Carried over by popular demand. Converts your Apple screen to 80x24 upper and lower case. Purchase VIDEOTERM with WordStar and save!

If purchased separately: \$290.
List Price: \$345.00
Microhouse Price: \$270.00

MORROW DISCUS 2D

8 inch single-sided double-density floppy disk drive subsystem. Includes CP/M* and MBASIC.

List Price: \$1199.00
Microhouse Price: \$995.00

MORROW HARD DISK SUBSYSTEM

10 Megabyte. Includes S-100 controller card, CP/M*, and enclosure.

List Price: \$3695.00
Microhouse Price: \$3062.00

MICROSOFT APPLE SOFTCARD

Purchase the softcard with MicroPro's WordStar and save \$70! Converts your Apple II or II plus to a CP/M* system. Includes MBASIC! Price if purchased separately: \$295.

List Price: \$349.00
Microhouse Price: \$279.00

TCS/Atlanta INTERACTIVE ACCOUNTING SYSTEM

for small businesses. Ver. 5.0. Each package can be used alone or post automatically to the General Ledger. Compiled version (no support language needed). Price listed is per package. Generalledger, Accounts Receivable, Accounts Payable, and Payroll packages available. Call for details on new Order Entry & Inventory packages. ALSO AVAILABLE FOR APPLE II. Also available in source.

List Price: Compare at \$530
Microhouse Price: \$79.00/\$25.00

ALL FOUR TCS PACKAGES (compiled)

List Price: Compare at \$530/pkg.
Microhouse Price: \$269.00/\$90.00

CP/M is a registered trademark of Digital Research UNIX is a registered trademark of Bell Labs APPLE is a registered trademark of Apple Computers TRS80 is a registered trademark of Tandy Corp.

SHIPPING: Add \$5 per manual or software package. Add \$2.50 for COD orders. Call for shipping charges on other items. Pennsylvania residents add 6 per cent sales tax.

Microhouse

P.O. BOX 498
BETHLEHEM, PA 18016
(215) 868-8219

**If you're looking for
the best prices
in the U.S.A. on**



**TRS-80
MICROCOMPUTERS**

We have consistently offered the TRS-80 line at **savings up to 20%**. You can save up to \$1500 by buying from Computer Discount of America.

**ATARI®
MICROCOMPUTERS**



We have the full line of ATARI personal computers and systems.

Model II		
26-4002	64K, 1 disc	\$3385.00
Model III		
26-1061	4K, Level I	\$ 610.00
26-1062	16K, Level III	\$ 845.00
26-1066	48K, Level II 2-drive/RS-232	\$2115.00
Color Computer		
26-3001	4K	\$ 329.00
26-3002	16K w/Ext. Basic	\$ 499.00
EPSON		
MX70	Printer	\$ 375.00
MX80	Printer	\$ 485.00
MX80FT	Printer	\$ 639.00

Our savings are as big on expansion interfaces, printers, diskettes, Apple Computers, OKIDATA Microline, C-ITOH Starwriter, Lexicon Modems — everything for your computer.

We have the **largest inventory in the Northeast**, and most models are in stock, for immediate delivery. Our full price catalog or a price quote is as near as your phone.

**CALL TOLL FREE:
800-526-5313**

**Computer
Discount
of America**

COMPUTER DISCOUNT OF AMERICA, INC.
15 Marshall Hill Road, West Milford Mall
West Milford, New Jersey 07480
In New Jersey Call 201-728-8080

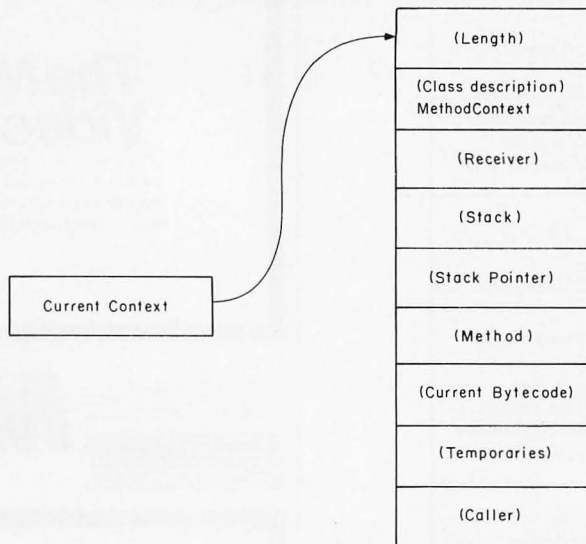


Figure 7: The only object pointer used by the Smalltalk-80 interpreter is a reference to a MethodContext.

**The Smalltalk-80
Virtual Machine
Implementation is a
program running in
the machine language
of the target
computer.**

Primitive Subroutines

The Smalltalk-80 Virtual Machine implementation is a program running in the machine language of the target computer. The storage manager is the collection of subroutines in this program that deals with memory allocation and deallocation. The interpreter is the collection of subroutines in this program, one of which fetches the next bytecode from the currently running method and calls one of the others to perform the appropriate action for that bytecode. In addition to these functions, we have found that there are several other places in the Smalltalk-80 system where performance considerations make it necessary, or at least desirable, to implement certain functions as machine-code subroutines in the Smalltalk-80 Virtual Machine. These places are:

- input/output: connecting the

Smalltalk-80 system to the actual hardware

- arithmetic: basic arithmetic for integers
- subscripting indexable objects: fetching and storing indexable instance variables
- screen graphics: drawing and moving areas of the screen bitmap quickly
- object allocation: connecting the Smalltalk-80 code for creating a new instance with the storage manager subroutines

We call this set of subroutines the *primitive subroutines*.

The primitive subroutines are represented in the Smalltalk Virtual Image as methods with a special flag that says to run the corresponding subroutine rather than the Smalltalk-80 bytecodes. When the interpreter is executing the code to send a message and finds one of these flags set, it calls the subroutine and uses the value returned from it as the value of the method. The number of these methods in Smalltalk-80 is small (around one hundred) in order to keep the rest of the system as flexible and extensible as possible. We will not list those methods that are primitives, but will refer the reader to *Smalltalk: the Language and Its Implementation* (Goldberg, Robson, and Ingalls, 1981) for details.

Need a Real-Time Multi-Tasking Executive for 8080 and Z80?

CP/M®
Interface
Also Available

AMIX

- Faultless operation proven in world wide use
- Truly hardware independent
- Optimized for fast interrupt response
- Minimal memory requirements
- ROMable for control applications
- Terminal Handler is CP/M BDOS compatible
- Console Driver supports Intel iSBC boards
- SYSGEN speeds user system configuring
- Program in PL/M, Fortran, Pascal or Assembler
- Source code included (Intel or Zilog mnemonics)
- Unlimited use licence agreement
- Complete documentation (available separately)
- Low cost

Circle 418 on inquiry card.

Dealer enquiries invited

CP/M is a trademark of Digital Research Corp.; RMX/80, iSBC are trademarks of Intel Corp.



KADAK Products Ltd.

206-1847 West Broadway Avenue
Vancouver, B.C., Canada V6J 1Y5
Telephone (604) 734-2796

ADA A NEW BEGINNING

ADA/M - Compiles **ADA** program into the Host assembly language. Excellent for learning **ADA** and converting existing programs. Includes **ADA** Compiler and Compiler **ADA** Programming Support Environment (CAPSE) \$495.

ADA/P - Compiler and Kernal APSE (KAPSE) for Apple / ATRI \$995.

ADA/R - Compiler and KAPSE for TRS-80 \$995.

ADA/H8 - **ADA** Language System (ALS). Includes Compiler, KAPSE, Minimal APSE (MAPSE) and Library APSE (LAPSE). Z80,8080 \$2,995.

ADA/H16 - ALS. Includes Compiler, KAPSE, MAPSE, and LAPSE. PDP-11, Z8000, 8086, 9900 \$3,995.

ADA Programmers Manual \$25.

ADA Syntax Reference Card \$6.

Credit for purchase of **ADA/M** will be given toward purchase of larger ALS. Royalty will be paid to customers who convert existing programs into **ADA** for inclusion in the ALS library.

DIGITAL ELECTRONIC SYSTEMS, INC.
Box 5252, Torrance, California 90510

A few of these primitive methods are executed so often that even the cost of looking them up in their classes' method dictionaries would be excessive. These methods are instead represented as special versions of the Send Message type of bytecodes. The message + , for example, is represented this way. When this bytecode is executed and the top two elements of the stack are small integers, then the primitive method is called as a subroutine. When this bytecode is executed and the top two elements of the stack are not small integers, then the + message is sent normally.

Conclusion

The Smalltalk-80 Virtual Machine is a fairly small computer program that consists of a storage manager, an interpreter, and a set of primitive subroutines. The task of implementing a Smalltalk-80 Virtual Machine for a new target computer is not large (especially when compared with the task of implementing other large programming systems) because most of the functions that must usually be implemented in machine code are already part of the Smalltalk-80 Virtual Image that runs on top of the Virtual Machine.

The Smalltalk-80 Virtual Machine could also be implemented in hardware, although this has not yet been done. Such an implementation would sacrifice some of the flexibility of software, but it would result in the performance benefits that hardware provides. Given the evolving nature of Smalltalk, it may not yet be time to implement the Virtual Machine in hardware: new Smalltalks that are more powerful would likely need at least small changes in Virtual Machine definition and implementation. However, hardware assists to Smalltalk-80 Virtual Machine software can greatly improve performance. Writable microcode stores for the pieces of code that are frequently run, hardware assists for graphics, or hardware assists for the fetching of bytecodes could all potentially improve the performance of a Smalltalk-80 Virtual Machine implementation. ■