記 号 処 理 28－3
（1984. 6. 15）

Design of a 32-bit Virtual Machine for Smalltalk-80†

Norihisa Suzuki, Ichiro Ogata, and Minoru Terada

Department of Mathematical Engineering
The University of Tokyo

ABSTRACT

The Smalltalk-80 virtual machine is specified by Goldberg and Robson based on a 16-bit-word machine.Thus,the number of objects that can exist at the same time is 32,768 and the range of SmallIntegers,the most basic numbers,is from -16,384 to 16383;this is a severe limitations on the scope of application programs that can be written and on the performance of the system.Most computers on which we are now planning to implement Smalltalk-80 have byte-addressable,24 to 32-bit address busses. Thus,we designed a Smalltalk-80 virtual machine based on 32-bit computers.

1. Introduction

We created a Smalltalk-80 system[5] based on a virtual machine specification by Goldberg and Robson[3]. After several iteration of the implementation,the system is now running on a SUN workstation(SUN-1) with 10 M Hz MC68000 as a central processing unit.The performance of the current version is comparable to that of Dolphin implementation[4].

Following is some observations from our experience. We have now 1.25 mega-byte main memory. These are not enough space with this configuration.The system goes into the garbage collection too frequently with a moderate application program.

We used a mark-and-sweep garbage collector.This decision is made by some of us who never have created an interactive system. Even though the time to complete the entire garbage collection phase is only 3-5 seconds,it still blocks the interactive activities of the users completely.The system that we are now implementing uses a real-time garbage collector based on the Deutsch-Bobrow algorithm[1]. This problem is somewhat lessened by our innovation to change the cursor style to indicate the garbage collection phase. The user quickly notifies this and suspends his activity.Another problem of the mark-and-sweep garbage collector is that the all the objects are touched.Smalltalk-80 virtualImage is a large system;over 500k bytes of objects are system objects that are used to implement the basic system and, therefore,are seldom modified.They will hardly ever become garbage.However, a mark-and-sweep garbage collector always marks and touches these objects. On the other hand a reference-counting garbage collector only touches those objects that are changed,so it is a good choice for a large object-oriented system.

The first version allocated contexts in the heap and they are collected. this caused severe performance penalty, since the allocation and deallocation costs are very high.the second version used a linearized contexts[5].This single technique gained three-time speed up.

The speed of BitBlt operation is very crucial to the overall performance of the system.In the published benchmark tests and performance measurements[4] BitBlt operations occupy a minor portion of the entire computation time. So when we created a BitBlt that can more 1 million pixels in 900 milli-seconds, we decided to put our efforts in tuning other parts.However, in the last version we put a 500-millisecond BitBlt,and the performance of most programming activities doubled. This is because when we use Smalltalk-80,we are rewriting windows most of the time, and BitBlt computation occupies a significant parts.It is, however,hard to measure these activities.This is another example that the performance measurement on some particular examples have very little meaning in practice.

16-bit Oop are too short.Oops run out very often.Furthermore,the object table is allocated somewhere in the memory, and the Oop starts from 0,it is the offset from the start of the table.One addition always has to take place to access the object table.Therefore,We decided to design a 32-bit virtual machine in which the Oop directly points to the entry in the object table, and, therefore, an addition is not necessary for an access.

We also optimized the whole architecture to suit MC68000.

2. A 32-bit Virtual Machine

The objective of this architecture is to create a very fast virtual machine on a demand-paged MC68010. Some of the data structures are not suitable for other microprocessors such as MC68020. However, the modifications can be easily done.

2.1. Representation of Objects

Unlike the Smalltalk-80 virtual machine specified in the book[3],our virtual machine uses 32 bits to represent object pointers. This change had effects in all the representations of objects.

†Smalltalk-80 is a trademark of the Xerox Corporation.

2.1.1. Object Pointer

The most common data manipulated by a Smalltalk-80 interpreter are object pointers. The virtual machine specification uses a 16-bit object pointer. Object Pointers are 1 bit tag-coded that indicates how the rest of the bits are to be interpreted. The tag bit is placed in the least significant bit. If the tag bit is 0,then the rests are to be 15 bits represents an object reference (commonly referred as Oop in the Smalltalk book). If the tag bit is 1, then the rest of 15 bits represents a 2's complement integer value (referred as SmallInteger).

Oop is a unique name of the object. If we want to know the real address of the object, we must look up the object table using the Oop as an index.

Smalltalk requires the memory as large as 1 mega byte, so the address field occupies 20 bits or so in the object table. It means that one entry of the object table uses at least 32 bits.

These standard formats are the problem.When we want to test whether a pointer is an Oop or a SmallInteger,we explicitly test the LSB by using machine instruction "bit test". And more, when we want to get the real address of an object, we must calculate the object table entry address by using shift and add instruction. This is not the case for the Xerox machines upon which the Smalltalk-80 system was originally developed. They reflect the LSB's value as the machine flag,so they do not need any explicit test to know the pointer's tag. And they use word addressing,and can use the absolute address from 0, they do no conversion to get the object table entry.So we decided not to use this standard format, instead we use new 32-bit format. (see fig.1)

As MC68000 reflect MSB's value in his sign flag, We have no need to explicitly check whether a pointer is an Oop or a SmallInteger. When we move the pointer to one of MC68000's register, sign flag tells us which the pointer is. Placing tag bit in the LSB eliminates the overhead associated with checking whether the Oop is an SmallInteger, because if we access the object table by SmallInteger,it causes address exception. So we are now free from illegal access checking. The merits of using the 32-bit formats are:

(1) This format enlarges the number of Oops greatly. In standard 16-bit format,the number of Oops are only 32k. And about 24k Oops are used by the system,so users are allowed to use only 8k Oops. This will limit the scope of application to the large AI problem.

(2) We use the entry address of an Oop as its Oop. i.e. the entry address of any Oop is the Oop itself. We can get the entry address of the object table without the overhead of conversion.

(3) This format also enlarges the SmallInteger range.This lessen the need to run the LargepositiveInteger's method in Smalltalk. This also makes the array indexing code simple and fast. Because the largest possible index of an array is 64k, and the largest possible SmallInteger value in the 16-bit version is 16k, the original code must handle the LargePositiveInteger as an index. It makes original array indexing code rather complex.

2.1.2. Object Table

Oops are the real addresses of the entries in the object table. These entries contain information such as the actual location of objects in the heap, the reference counts,and garbage collection information.

The format of the object table is shown below. (fig.2) Each entry occupies aligned 4 bytes;therefore, the valid Oops have zeros in the least significant two bits.

The valid Oops are allocated Oops;the most significant byte contains the reference count and the other three bytes contain the real address of the object. This format allows both reference count and the address to be accessed in one instruction with the least amount of time. The least significant bit of a free Oop is one;even if a free Oops is considered to be a valid because of bugs in the virtual machine,the memory access operation will be trapped. According to the original Smalltalk-80 virtual machine specification,there are three flag fields O,P and F. The O field indicates whether the size of the byte object is odd length or not. This is no longer necessary,since the size of objects is measured in terms of bytes. The P field indicates whether the object is a pointer object or not, but this field is stored along with the object as indicated in section 2.1.3. The F field indicates whether the Oop is free or not,but this is indicated by the least significant bit of the entry.

2.1.3. Object

Objects occupy at least two 4-byte long words; the first long word contains the pointer bit and the size of the object in terms of bytes. The second long word is the Oop for the class of the object. There are four kinds of object: a pointer object, a word object , a byte object, and a compiled method. The formats of these four objects are shown belows. (fig. 3 ~ 7)

2.2. Context

Contexts are Smalltalk-80 terms for procedure activation records.They are treated as first-class objects;pointers can point to them and messages are sent. The strength

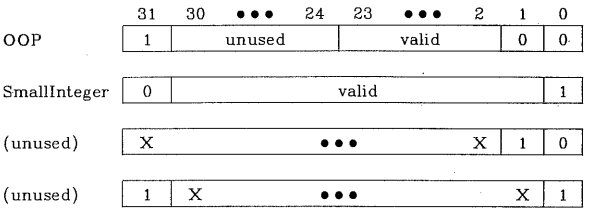

fig. 1 Object pointer

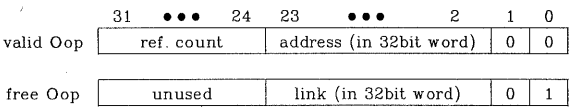

fig. 2 Object table entry

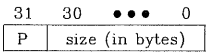

fig. 3 Object header

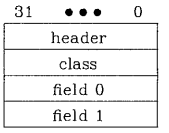

fig. 4 Pointer Object

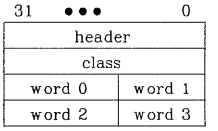

fig. 5 Word Object

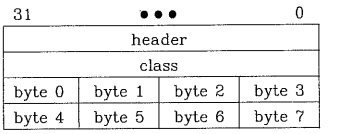

fig. 6 Byte Object

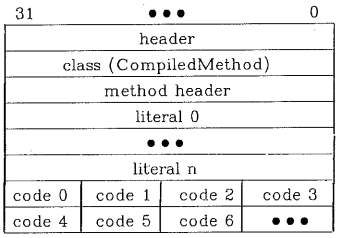

fig. 7 a CompiledMethod

of Smalltalk-80 system comes from treating contexts as objects;complex control structures such as process switching can be easily implemented in the language,and a powerful debugger has been implemented using this feature. However, the overhead for allocating and deallocating objects is the major performance bottleneck. We came up with the idea of linearly allocating contexts on the stack until either reference are made, messages are sent to or process switch occurs[5] independently. Peter Deutsch also used similar,but more complex treatment of context[2]. We adopt Deutsch's terminology to describe our algorithm.

We call the linearized contexts in the stack volatile.Volatile contexts are not objects. the evaluation stack part of a calling side is shared with the parameter part of a called side to eliminate copying of parameters. Information in the volatile context such as the stack pointers and the instruction pointers are stored as the real address to achieve very efficient context switching.On the other hand the contexts allocated in the heap and treated as real objects are called stable. In our first implementation we have only two representation of contexts: volatile and stable.When a context has to be treated as an object, a volatile context is con-

verted to a stable context.

In our current implementation we have another representation of contexts call hybrid. Hybrid contexts are similar to volatile contexts in that reference counts are not done for Oops in these contexts.Correct counts are restored when transactions finish[1]. However,hybrid contexts are similar to stable contexts in that they are allocated in a heap area. It is,however,necessary to enumerate all the hybrid contexts at the end of transactions to restore all the reference counts. So we have a special heap area that only contains hybrid contexts. Hybrid contexts are transformed to stable contexts when messages are sent to that contexts.

2.3. Garbage Collection

We use a reference counting garbage collection based on Deutsch-Bobrow transaction garbage collection[1].

2.4. Register Allocation

Some of the 16 registers of MC68000 are used for special purposes. We cached real address of some important Oops,such as Instruction pointer , Stack pointer , home contexts in machine register. And also we allocate the receiver's Oop and method's Oop in machine register.

3. Conclusion

We specified a Smalltalk-80 virtual machine for 32-bit,byte-addressable computers. We have MC68000 particularly in mind so that this virtual machine is most suitable if used for MC68000-based computers.

We also described other techniques we are using for our new virtual machine implementation.

Acknowledgement We appreciate the assistance from Takashi Aoki for implementing a disk system.

References

[1] Deutsch,L.P.,and Bobrow,D.G., "An efficient,incremental,real-time garbage collector", CACM 9,9 (Sep.1976) pp.522-526

[2] Deutsch,L.P.,and Schiffman,A.M. "Efficient Implementation of the Smalltalk-80 System", Proc. of 1984 ACM POPL conference pp.297-302

[3] Goldberg , A. , and Robson,D., "Smalltalk-80:The Language and its implementation", Addison-Wesley, Reading,MA,1983

[4] Kransner,Glenn,Ed., "Smalltalk-80:Bits of History, Words of Advice", Addison-Wesley, Reading,MA,1983

[5] Suzuki,N.,and Terada,M., "Creating Efficient Systems for Object-Oriented Languages", Proc. of 1984 ACM POPL conference pp.290-296