Compatible Forth on a 32-bit Machine

Mitch Bradley

Sun Microsystems
2550 Garcia Avenue
Mountain View, CA. 94043

William L. Sebok

Princeton University
Department of Astrophysical Science
Princeton, N. J. 08544

Abstract

The arrival of affordable 32-bit machines poses some problems for Forth, which explicitly assumes that addresses and stacks are 16 bits wide. A scheme is presented for writing Forth code that will run without change on either a 16-bit or a 32-bit machine. This scheme is based on experiences running Forth on a number of 16 and 32 bit systems and may easily be implemented on top of a standard system.

Introduction

The Forth-83 standard explicitly defines that an item on the stack is 16-bits wide. This specification is a disaster for machines like the VAX or the 68000, which have 32-bit address spaces. Furthermore, the standard specifies that a 32-bit number is represented as two stack items. Our scheme requires a slightly different view of the stack, allowing the stack width to be either 16 bits or 32 bits, whichever is natural for the machine.

We call the width of the stack the "normal" size, and a data item of that size is called a "normal". All of the usual Forth operators (`+` , `DUP` , `SWAP` , `@` , `!` , etc.) deal with "normal" size operands. This means that most things don't change at all, because existing Forth code that doesn't use double numbers usually works just as well when the stack width is 32 bits instead of 16 bits.

Code that presently deals with 32-bit numbers is a problem, since 16-bit Forths require two stack items to hold a 32-bit number, whereas 32-bit Forths only need one stack item. The current Forth standard confuses the concepts of "32-bit number" and "double number". They are not the same. This work makes careful distinction between these two concepts. Understanding of the distinction is the key to using 32-bit numbers in a portable fashion.

A final important distinction is the difference between the size of an address and the size of a data item. On some machines, it is desirable to have a 16-bit Forth, but still allow access to more than 64 Kbytes of data. Our scheme allows this distinction to be made without sacrificing portability.

Address and Data Sizes

Five data sizes are defined:

Normal
A normal is the same width as the Forth parameter stack. Normals are denoted by "n" or "addr", depending on whether the item is a data item or an address. This is the basic data type that the Forth system understands.

Byte (Character)
A byte is 8 bits. The letter "c" (for character) is used to denote a byte.

Word
A word is 16 bits. The letter "w" is used to denote a word. (Unfortunately, the word "word" is already used in Forth to mean several things, and this adds yet another meaning. This nomenclature was chosen anyway because some Forth vendors are already using it.)

Long
A long is 32 bits. The letter "l" denotes a long.

Double
A double is two stack items. It is denoted either by the number "2" or by the letter "D". When a double is stored in memory, the number on top of the stack is stored at the lower address. A double is not necessarily the same as a long.

Stack Width

The stack width is whatever makes sense for the machine, either 16 bits or 32 bits. A good choice for the stack width is the size of the particular machine's address arithmetic.

Data or address items that are not as wide as the stack are stored as the least-significant portion of a stack item, with the high-order bits set to 0. Items that are wider than the stack width are stored as more than one stack item, with the most-significant portion of the item closest to the top of the stack.

Sign extension is considered in a later section.

Fetch and Store Operators

The fetch and store operators move data between memory and the stack. There are 3 classes of operators:

Default Operators
These are the familiar Forth operators @ and !, which operate on a "normal" address and "normal" data. In other words, their address and data operands are the same width as the stack.

Default-Address Operators
These operators take a "normal" address and use an explicitly-specified data size. They are found in the first column of the following table.

Explicit Operators
These operators behave similarly to the default operators, except that the address is explicitly either a "word" or a "long", rather than being whatever the "normal" address is. They are listed in the second and third columns of the table. They are useful primarily in cases where one machine has two kinds of addresses.

Address:	Normal	Word	Long
Data:			
Normal:	`@` (addr – n)	`@W` (w-addr – n)	`@L` (l-addr – n)
	`!` (n addr –)	`!W` (n w-addr –)	`!L` (n l-addr –)
Byte:	`C@` (addr – c)	`C@W` (w-addr – c)	`C@L` (l-addr – c)
	`C!` (c addr –)	`C!W` (c w-addr –)	`C!L` (c l-addr –)
Word:	`W@` (addr – w)	`W@W` (w-addr – w)	`W@L` (l-addr – w)
	`W!` (w addr –)	`W!W` (w w-addr –)	`W!L` (w l-addr –)
Long:	`L@` (addr – l)	`L@W` (w-addr –)	`L@L` (l-addr – l)
	`L!` (l addr –)	`L!W` (l w-addr –)	`L!L` (l l-addr –)
Double:	`2@` (addr – n1 n2)	`2@W` (w-addr – n1 n2)	`2@L` (l-addr – n1 n2)
	`2!` (n1 n2 addr –)	`2!W` (n1 n2 w-addr –)	`2!L` (n1 n1 w-addr –)

Required Implementation

It is not necessary for a particular Forth system to implement all the operators. Only those address sizes and data sizes which the machine reasonably supports should be implemented. The default operators (`@` and `!`) and the default-address operators should be implemented. An application should use the default operators whenever possible. The explicit operators should be used only when there are special requirements which require specifically choosing the address size, with the understanding that the code may not be portable to some machines. The existence of explicit operators should actually increase portability, since such special dependencies will then not be "hidden". Also, code which uses explicit operators should be portable to machines which support addresses and data of the sizes used (regardless of the stack width), since the definition of the explicit operators is independent of the stack width.

Address Modification Operators

The use of words like `1+` and `2+` to increment addresses is a serious hindrance to portability. For one thing, it works poorly on word-addressed machines, where adding 1 to an address may not be the right way to select the next byte. Further, it fails if code from a 16-bit machine is run on a machine with a 32-bit-wide stack. We propose the following operators which modify "normal" addresses.

`A1+` (addr – addr-of-next-normal)

`CA1+` (addr – addr-of-next-byte)

`WA1+` (addr – addr-of-next-word)

`LA1+` (addr – addr-of-next-long)

```
A+      ( addr n – addr-of-nth-next-normal )
CA+     ( addr n – addr-of-nth-next-byte )
WA+     ( addr n – addr-of-nth-next-word )
LA+     ( addr n – addr-of-nth-next-long )
```

These operators are useful for incrementing pointers, because they scale by the size of the data item specified. Equivalently, they may be used for indexing into an array containing items of the given type.

Stack Operations

A major problem when dealing with multiple data sizes is managing the stack. The following stack operations provide a minimal set of primitives for trying to cope with the problem. Not all possible combinations are listed here, but is should be obvious how to name those that aren't included.

```
WNOVER      ( w n – w n w )
WOVER       ( w1 w2 – w1 w2 w1 )
WNSWAP      ( w n – n w )
WSWAP       ( w1 w2 – w2 w1 )
LNOVER      ( l n – l n l )
LOVER       ( l1 l2 – l1 l2 l1 )
LNSWAP      ( l n – n l )
LSWAP       ( l1 l2 – l2 l1 )
W>R         ( w – )
WR>         ( – w )
L>R         ( l – )
LR>         ( – l )
```

Sign Extension

When a word item is fetched to a 32-bit stack, it is not sign-extended. If sign extension is desired, the following word may be used.

```
<W@     ( addr – sign-extended-w )
```

Type Conversion Operators

These words convert stack items of one type to stack items of another type. When a conversion from a larger type to a smaller type occurs, there may be a loss of information, so these should be used with care.

```
N->L    ( n – l )
```
Convert a normal stack item to a long.

```
W->L    ( w – l )
```
Convert a word stack item to a long without sign extension.

```
S->L    ( w – l.sign-extended )
```
Convert a word stack item to a long with sign extension.

`L->W` (l – w)
Convert a long stack item to a word. The most-significant word of the long is dropped if the stack is 16 bits wide or zeroed if the stack is 32 bits wide.

`L->N` (l – n)
Convert a long stack item to a normal. If the stack is 16 bits wide, this is the same as `L->W`. If the stack is 32 bits wide, this is a no-op.

`LWSPLIT` (l – w.low w.high)
Convert a "long" into two stack items, with the high-order 16-bits of the long being on top of the stack, and the low-order 16-bits below that. Note: for a 16-bit Forth, this word is a no-op. For a 32 bit Forth, the resulting words actually appear on the stack as two 32-bit numbers with zeros in the high-order portions, since "word"s on the32-bit stack are really "long"s with high order zeros.

`WLJOIN` (w.low w.high – l)
Convert two stack items into a long, with the high-order 16-bits of the long being taken from the top of the stack, and the low-order 16-bits of the long from the second stack item. This item is the inverse of `LWSPLIT`, and is a no-op on a 16-bit system.

Size Constants

The following words are constants. They leave on the stack a number which is the size of a particular operand type.They are useful in cases where it is necessary to perform explicit address arithmetic, such as when `ALLOT`'ing several longs or when using `+LOOP` to step through an array. Size is measured in "addressable units", not bytes, because some machines are not byte-addressable. An "addressable unit" is the size of the operand that an address normally refers to. On a byte-addressed machine, it is a byte; on a word-addressed machine, it is a word.

`/C` (– n)
Leaves the size of a character (1 on a byte-addressed machine).

`/W` (– n)
Leaves the size of a word. (2 on a byte-addressed machine).

`/L` (– n)
Leaves the size of a long. (4 on a byte-addressed machine).

`/N` (– n)
Leaves the size of a stack item.

Escape Hatches

If you come across a case where you just can't think of a reasonbable or efficient way to write portable code, the following words are useful. They allow selective compilation of different code for different systems, or explicit specification that a bit of code is size-dependent.

`16-BIT` (--) Immediate
No-op on a 16-bit system. Causes an abort on a 32-bit system.

`32-BIT` (--) Immediate
No-op on a 32-bit system. Causes an abort on a 16-bit system.

`16\` (--) Immediate
No-op on a 16-bit system. Causes the rest of the line to be ignored on a 32-bit system.

`32\` (--) Immediate
No-op on a 32-bit system. Causes the rest of the line to be ignored on a 16-bit system.

Number Input Syntax

A string representing a number is explicitly a "long" if it contains a decimal point. A numeric string that does not contain a decimal point is a "normal" size number. This applies only to input, not to output.

64 Bit Items

In anticipation of the future, the word "extended" is suggested for denoting explicit 64-bit items. The associated mnemonic letter is "x". The names of words for use with 64 bit items is an obvious extension of the above proposal.

Usage, or How to Write Portable Code

The default operators (`@ !`) should be used whenever it doesn't matter whether the operand is 16 bits or 32 bits. Experience shows that this is most of the time. This way your code will work on both 16 bit and 32 bit Forth systems.

In those cases where a 32 bit number is needed, use the "L" operators instead of the traditional "2" operators (`2@` , `2SWAP` , etc.). The "2" operators should be reserved for their original purpose, i.e., to manipulate 2 stack items at a time. Remember that "2 stack items" is not the same thing as a 32 bit number. In particular, on a 32 bit system, a 32 bit number is one stack item. It does not suffice to define, for example, `2SWAP` to mean a 32 bit number regardless of stack width, because there are many times when `2SWAP` is needed for interchanging pairs of stack items. Thus, the "L" operators are really needed for manipulating 32 bit numbers.

The double number arithmetic operators as listed in the "Double Number Extension Word Set" of the 83 standard are okay to use on 32 bit numbers because their definition is consistent with 32 bit numbers. For consistency in naming, it is recommended that additional copies of these operators be made, having names beginning with "L". For instance, `: L+ D+ ;` The "D" names should be kept around for compatibility with old code.

The use of `ROT` instead of `WLSWAP` is discouraged, as it is not portable. Similarly, `-ROT` should not be used for `LWSWAP`.

PICK and ROLL

Use of `PICK` and `ROLL` is not recommended. However, if their use can't be avoided, and there are both "normal"s and "long"s on the stack, the code may be made portable by a compile time calculation of the argument. For example, if the desired effect is the following stack transformation: (`nx n n 1 n -- n n 1 n nx`), this code will do the trick: `[ /N 3 * /L + /N / ] LITERAL ROLL`.

Important Point

It is not necessary to implement all the above words. It is important to agree on a nomenclature, however, so that code may be shared without confusion about what it means. The implementation of any one of these words is trivial, so it should be practical to implement them as needed, rather than keeping the whole package around all the time. The different address sizes need not all be implemented; in fact, many machines cannot reasonably support more than one address size. There are machines that do have different address sizes, so the complete matrix of words is defined in order to prevent the proliferation of different words for the same function.

In fact, on any particular system, many of the words in this scheme are just the same as already-existing words. For instance, on a 16-bit Forth, `W@` and `W@W` are just the same as `@`. On a 32-bit Forth, `L@` and `L@L` are just the same as `@`.

Manuscript received July 1984.