IEEE Draft Std P1275.6/D5
Standard for Boot (Initialization Configuration) Firmware
64 Bit Extensions

March 20, 1995

DRAFT EDITION FOR SPONSOR BALLOTING

Sponsored by the IEEE Bus Architecture Standards Committee
Prepared by the Open Firmware Working Group

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, PO Box 1331
Piscataway, NJ 08855-1331 USA

Introduction

(This introduction is not a part of IEEE Draft Std P1275.6/D3, *Standard for Boot (Initialization Configuration) Firmware, 64 Bit Extensions*.)

Firmware is the ROM-based software that controls a computer between the time it is turned on and the time the primary operating system takes control of the machine. Firmware's responsibilities include testing and initializing the hardware, determining the hardware configuration, loading (or booting) the operating system, and providing interactive debugging facilities in case of faulty hardware or software.

Open Firmware is the firmware architecture defined by IEEE Std 1275-1994, *Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices.* That standard is bus-independent, instruction-set-independent, and system-independent.

The core requirements and practices specified by IEEE Std 1275-1994 must be supplemented by system-specific requirements to form a complete specification for the firmware for a particular system. This standard establishes such additional requirements pertaining to 64 bit processors and systems.

Working Group Members

The following individuals were members of the Working Group at the time this document was produced:

William M. (Mitch) Bradley, Chairman, FirmWorks
John Rible, Draft Editor, FORCE COMPUTERS Inc.
Mike Tuciarone, FirmWorks
Paul Thomas, Sun Microsystems, Inc.
Ron Hochsprung, Apple Computer Inc.
Martin Walsh, Sun Microsystems, Inc.
David M. Kahn, Vice Chairman, Sun Microsystems, Inc.
Luan D. Nguyen, Secretary, IBM Corporation
David L. Paktor, FORCE COMPUTERS Inc.
Thanos Metzelopoulos, Antel, Inc.
Yongjae Rim, IBM Corporation
Mike Williams, SHL

Sponsor Balloting Body

The following individuals were members of the Sponsor Balloting Body:

This page is intentionally left blank.

Table of Contents

This page is intentionally left blank.

1. Overview

This standard describes the application of IEEE Std 1275-1994, *Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices*, to 64 bit processors and systems. Additional supplements for 64 bit processors and systems should include this supplement by reference.

2. References

[1] IEEE Std 1275-1994, *Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices*

3. Terms

This standard uses technical terms as they are defined in the documents cited in "References", plus the following terms:

core, core specification: refers to IEEE Std 1275-1994, *Standard for Boot (Initialization Configuration) Firmware, Core Requirements and Practices*, i.e., the standard that specifies the system-independent and bus-independent requirements for Open Firmware.

Open Firmware, firmware: The firmware architecture defined by the core specification and this specification or, when used as an adjective, a software component compliant with the core specification and this specification.

4. Generic Requirements

4.1. Cell Size and Client Interfaces

The cell-size for implementations compliant with this standard shall be 64-bits. 64-bit implementations shall export a 64-bit client interface based on a set of bindings to the core specification for that 64-bit instruction set architecture. 64-bit implementations may choose to export both a 32-bit and a 64-bit client interface, depending on the type of standalone program compatibility required by the implementation.

It is recommended that 64-bit implementations only export a single 64-bit client interface, and that 32-bit client programs use a software wrapper layer to convert 32-bit client interface calls to 64-bit calls and convert 64-bit results to 32-bit results.

4.2. Virtual Address Allocation

The firmware may use any portion of the virtual address space, including virtual addresses above the 32-bit address range. The firmware should allocate all virtual addresses from the range 0x0 through 0xffff.ffff, except those virtual addresses that the firmware does not expose through the client or device interfaces. This recommendation allows 32-bit client programs to access firmware-allocated virtual addresses. Client programs that use a 32-bit client interface and manage the upper portion of the virtual address space must do so on their own, and must respect any allocations made by firmware in the upper portion of the address space.

5. Interface Changes and Extensions

5.1. 64-Bit Extensions

This document augments and modifies the core specification in the following ways:

This specification overrides the following definitions in the core specification, Annex A, Section A.1.2.2:

x x1 x2 etc.	Arbitrary 64 bit stack items.
n n1 n2 n3	Normal, signed values (64-bit)
xyz	Arbitrary descriptive text (signed 64-bit value)

This specification adds the following definitions:

o o1 o2 oct1 oct2 etc.	64 bit signed values
oaddr	Octlet-aligned (64 bit aligned) address
octlet	An 8 byte quantity

This specification adds the following FCode functions. A system that implements the FCode Debugging Command Group shall implement commands corresponding to all of the FCode functions listed in this section with the same names and semantics as those FCode functions.

bxjoin **(b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi -- o)** **F** **0x241**

Join 8 bytes to form an octlet.

Combine the eight least significant bits of each operand to form an octlet. Other operand bits are ignored.

<l@ **(qaddr -- n)** **F** **0x242**

Fetch quadlet from *qaddr*, sign-extended.

lxjoin **(quad.lo quad.hi -- o)** **F** **0x243**

Join 2 quadlets to form an octlet.

Combine the thirty-two least significant bits of each operand to form an octlet. Other operand bits are ignored.

rx@ (FCode Function) **(oaddr -- o)** **F** **0x22E**

Fetch an octlet from device register at *oaddr*.

Data is read with a single access operation.
Result has identical bit ordering as the original register data.

rx@ (User Interface) **(oaddr -- o)**

Fetch an octlet from device register at *oaddr*.

Compilation:

Perform the equivalent of the phrase:

```
h# 22e get-token if compile, else execute then
```

Interpretation:

Perform the equivalent of the phrase:

```
h# 22e get-token drop execute
```

Note: A bus device can substitute (see **set-token**) a bus-specific implementation of **rx@** for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. The given User Interface semantics of **rx@** ensure that such substitutions are visible at the User Interface level.

rx! (FCode Function) **(o oaddr --)** **F** **0x22F**

Store an octlet to device register at *oaddr*.

Data is stored with a single access operation and flushes any intervening write buffers, so that the data reaches its final destination before the next "word" is executed.

Result is stored with the identical bit ordering as the input stack item.

rx! (User Interface) **(o oaddr --)**

Store an octlet to device register at *oaddr*.

Compilation:

Perform the equivalent of the phrase:

```
h# 22f get-token if compile, else execute then
```

Interpretation:

Perform the equivalent of the phrase:

```
h# 22f get-token drop execute
```

Note: A bus device can substitute (see **`set-token`**) a bus-specific implementation of rx! for use by its children. This is sometimes necessary to correctly implement its semantics with respect to bit-order and write-buffer flushing. The given User Interface semantics of **rx!** ensure that such substitutions are visible at the User Interface level.

wxjoin **(w.lo w.2 w.3 w.hi -- o)** **F** **0x244**

Join four doublets to form an octlet.

Combine the sixteen least significant bits of each operand to form an octlet. Other operand bits are ignored.

x, **(o --)** **F** **0x245**

Compile an octlet, *o*, into the dictionary (doublet-aligned).

x@ **(oaddr -- o)** **F** **0x246**

Fetch octlet from an octlet aligned address.

x! **(o oaddr --)** **F** **0x247**

Store octlet to an octlet aligned address.

/x **(-- n)** **F** **0x248**

.Number of address units in an octlet, typically eight.

/x* **(nu1 -- nu2)** **F** **0x249**

Multiply *nu1* by the value of **`/x`**.

xa+ **(addr1 index -- addr2)** **F** **0x24A**

Increment *addr1* by *index* times the value of **`/x`**.

xa1+ **(addr1 -- addr2)** **F** **0x24B**

Increment *addr1* by the value of **`/x`**.

xbflip **(oct1 -- oct2)** **F** **0x24C**

Reverse the bytes within an octlet.

xbflips **(oaddr len --)** **F** **0x24D**

Reverse the bytes within each octlet in the given region.

The region begins at *oaddr* and spans *len* bytes. The behavior is undefined if *len* is not a multiple of **`/x`**.

xbsplit **(o -- b.lo b.2 b.3 b.4 b.5 b.6 b.7 b.hi)** **F** **0x24E**

Split an octlet into 8 bytes.

The bits of greater significance than the eight least significant bits of each of the eight resulting values shall be zero.

xlflip **(oct1 -- oct2)** **F** **0x24F**

Reverse the quadlets within an octlet.

The bytes within each quadlet are not reversed.

xlflips **(oaddr len --)** **F** **0x250**

Reverse the quadlets within each octlet in the given region.

The bytes within each quadlet are not reversed. The region begins at *oaddr* and spans *len* bytes. The behavior is undefined if *len* is not a multiple of `/x`.

xlsplit **(o -- quad.lo quad.hi)** **F** **0x251**

Split on octlet into 2 quadlets.

The bits of greater significance than the thirty-two least significant bits of each of the two resulting values shall be zero.

xwflip **(oct1 -- oct2)** **F** **0x252**

Reverse the doublets within an octlet.

The bytes within each doublet are not reversed.

xwflips **(oaddr len --)** **F** **0x253**

Reverse the doublets within each octlet in the given region.

The bytes within each doublet are not reversed. The region begins at *oaddr* and spans *len* bytes. The behavior is undefined if *len* is not a multiple of `/x`.

xwsplit **(o -- w.lo w.2 w.3 w.hi)** **F** **0x254**

Split an octlet into 4 doublets.

The bits of greater significance than the sixteen least significant bits of each of the four resulting values shall be zero.

5.2. 64-Bit Changes

The following FCode function run-time definition is modified from the core specification:

b(lit) **(-- o)** **F** **0x10**

(F: /FCode-num32/ --)

Numeric literal FCode. Followed by *FCode-num32*.

Note: See the core specification for the complete definition of `b(lit)`.

FCode Evaluation: **(F: /FCode-num32/ --)**

The definition is unchanged from the core specification.

Run-Time:

Return *o*, the sign-extended 64-bit number with the same signed numerical value as the *FCode-num32*.