**Inter-Office Memorandum**

| | | | |
|---|---|---|---|
| To | Dorado users | Date | February 19, 1983 |
| From | Ed Taft | Location | PARC/CSL |
| Subject | Dorado booting implementation (edition 5) | File | [Indigo]<DoradoDocs>DoradoBootingImpl.press |

XEROX

This memo is a continuation of Dorado booting operation'', file [Indigo]<DoradoDocs>DoradoBooting.press. This is a comprehensive description of how Dorado booting is implemented and how boot files and other components are constructed and maintained.

## 2. How it works

The Dorado contains a microcomputer that performs several functions, including monitoring power and temperature, turning on and off the main power supplies and the disk drive, and initiating bootstrap sequences. The microcomputer can sense the state of the boot button, and it controls the green light. This microcomputer and attendant EProms and other logic are mounted on the BaseBoard, the bottom logic board in the machine.

A 1-push boot is handled entirely by the Dorado microcode itself, with no action by the BaseBoard. Multiple-push boots are handled by the BaseBoard. A 3-push boot involves the following multiple-step sequence.

### *2.1 Loading Bootstrap microcode*

The BaseBoard first halts the Dorado, issues an I/O reset, and resets or disables conditions that could interfere with initial bootstrapping (Hold, Fault task wakeup). It then forcibly loads into the Dorado's control store a small (~50 instruction) microprogram called Bootstrap. Finally, it starts the Dorado executing this microprogram.

Loading of Dorado microcode in this manner is very slow; therefore, the amount of microcode in Bootstrap is kept as small as possible.

### *2.2 Loading Initial microcode*

Next, the Bootstrap microprogram and the BaseBoard microcomputer cooperate to load into the Dorado's control store a somewhat larger (~800 instruction) microprogram called Initial. (This is distinct from the microcode installed on the Initial'' region of the disk, described in previous sections. The reason the same name is used is that much of the microcode is actually the same, and this Initial'' microprogram is generated from the same sources that produce files such as DoradoInitialDisk.eb.)

Initial is stored as data in the BaseBoard's EProm. It is read by the microcomputer and handed to the Dorado (via CPReg) at relatively high speed; the Boostrap microprogram copies this data into the Dorado's control store.

When this operation is finished, the Boostrap microprogram gives control to the Initial microprogram it has loaded. Bootstrap and Initial are arranged to occupy disjoint regions of the control store. The BaseBoard then waits for the Dorado to complete the next step.

### *2.3 Initial execution*

Initial performs a number of operations. First, it does a more thorough initialization of the Dorado. It corrects parity in all registers and memories. It initializes the memory system (cache, map, and storage) and puts it into a usable state. Finally, it notifies the BaseBoard that this step has been completed successfully. (This notification causes the BaseBoard to turn the green light continuously on. If this event fails to occur within a reasonable time, the BaseBoard switches to the 2-flash indication.)

Second, Initial enables the I/O tasks that it requires (disk, display, Ethernet, and junk tasks). It then reads the keyboard in order to determine which emulator microcode to bootstrap and whether to get it from disk or Ethernet.

Third, Initial loads the selected microcode from disk or Ethernet into the Dorado's main memory. (For Ethernet booting, this step employs the Pup Microcode Boot protocol, which is also used for booting Dolphins and is documented in [Maxc]<Pup>EtherBoot.press.)

Finally, Initial calls the LoadRam procedure, which loads the emulator microcode from main memory into the Dorado's control store (IM and IFUM) and transfers control to its starting address. The emulator microcode replaces Bootstrap and Initial. LoadRam is a standard one-page microcode loading routine which is included in Initial and in every emulator. It occupies the same region of control store in each microprogram. Therefore, LoadRam can load a new microprogram without danger of overwriting itself.

### *2.4 Software bootstrap*

When the emulator microcode is started either as a result of the above sequence or by a 2-push boot it does some further memory system initialization, including zeroing out all main storage. It initializes all I/O devices. Finally, in the case of an Alto emulator-based microprogram, it sets the disk partition to 5 (or to the number corresponding to a digit key which is pressed down), and then initiates an Alto-style software boot from either disk or Ethernet, according to the keys that are pressed down.

A 1-push boot is detected and handled by the emulator microcode itself, without action by the BaseBoard. It performs all actions described in the preceding paragraph except for zeroing memory and resetting the default disk partition.

The Alto-style software boot is described in Alto documentation: the Alto Hardware manual ([Maxc]<AltoDocs>AltoHardware.press, section 3.3), the BuildBoot documentation (<AltoDocs>BuildBoot.tty), and the Alto Boot protocol specification (<Pup>AltoBoot.press). The corresponding Pilot software boot is essentially undocumented, though some information is available from the Pilot Programmer's Manual.

## 3. How it is put together

This section attempts to locate all microcode and software required for Dorado bootstrapping and to set forth procedures that might otherwise be forgotten.

*3.1 Bootstrap, Initial, and BaseBoard microcomputer code*

The Bootstrap and Initial microprograms and the BaseBoard microcomputer software are programmed into a set of EProms that are installed on the BaseBoard. This programming is controlled by files on an Alto disk labelled Sosinski DoradoProms''.

The sources and command files for the BaseBoard microcomputer code are kept in a dump-format file, [Indigo]<DoradoSource>DoradoBaseRom.dm. The Dorado microcode (Bootstrap.mb and Initial.mb) are kept in [Indigo]<Dorado>Bootstrap.dm. The following command files may be executed to perform the specified operations:

**@DoradoBaseRom-Compile.cm@.** This assembles all the BaseBoard microcomputer code, executes all of the following command files, and prints a pile of listings.

**@GetNewDoradoUCode.cm@.** This obtains Bootstrap.mb and Initial.mb from Indigo. It then runs a program called MBtoBase.run, which converts the Dorado microcode into a form that can be loaded along with the BaseBoard microcomputer code. (Bootstrap.mb is transformed into Boot0.mb and Initial.mb into Boot1.mb.)

**@DoradoBaseRom-Load.cm@.** This binds and loads all the microcomputer code, along with Boot0 and Boot1. Parameters in this command line control the placement of the various pieces (see below). The output is DoradoBaseRom.mb, which may be used as input both by PNew (for EProm programming) and Midas (for BaseBoard microcomputer code debugging).

**@DoradoBaseRom-Blow.cm@.** This controls the programming of the four BaseBoard EProms, which are Intel 2716s. The EProms have starting memory addresses and BaseBoard coordinates listed in the following table, and are programmed in the order given:

| | *Rev. A* | *Rev. B* |
|---|---|---|
| F000 | c61 | c67 |
| F800 | b61 | c68 |
| C000 | f60 | b66 |
| C800 | e60 | b61 |

The microcomputer code and Bootstrap and Initial microprograms are placed independently so as to minimize the number of EProms that have to be reprogrammed when a change is made. Specifically, the microcomputer code and Bootstrap are contained in F000 and F800, while Initial and some tables that control loading of both Bootstrap and Initial are contained in C000 and C800. This has the following consequences:

If the microcomputer code is changed, F000 and F800 must be reprogrammed.

If Initial is changed, C000 and C800 must be reprogrammed.

If Bootstrap is changed (which is fairly unlikely), all four EProms must be reprogrammed.

The Bootstrap and Initial microcode are assembled from sources in [Indigo]<DoradoSource>BootstrapSources.dm. This must be done on a disk that already has the Pilot microcode environment, created by loading [Indigo]<DoradoSource>AEmuSources.dm and DMesaSources.dm, retrieving [Indigo]<DoradoSource>D1Lang.mc and D1Alu.mc, and executing @CedarMesaMake@.

### *3.2 LoadMB*

The program LoadMB.run reads a Dorado MB-format file and does one of three things with it:

Immediately loads it into the Dorado's control store and starts it.

Creates an EB-format (Ether-bootable) file, for installation on boot servers.

Creates a BR-format file suitable for loading with BCPL programs. The microcode image is loaded contiguously and pointed to by the static LoadRamImage.

Obviously, LoadMB can only run on a Dorado when performing the first function, but it can perform the latter two functions running on an Alto.

The complete form of the LoadMB command line is:

>LoadMB/*global-switches InputFile*.mb *parameter*/*switch* ...

If there are no global switches and no parameters, LoadMB simply loads the input file into the Dorado's control store and starts it at address 1070B, the standard soft restart'' entry point. The Dorado is not booted, and control simply returns to the Executive in the same disk partition as before. (The full bootstrap'' entry point, 1076, may be selected by means of the /S switch, described below.)

The global switches have the following effects:

| | |
|---|---|
| /B | Writes the microcode onto a BR-format file, whose file name defaults to *InputFile*.br. |
| /E | Writes the microcode onto an EB-format file, whose file name defaults to *InputFile*.eb. |
| /V | Pauses and awaits confirmation after reading the MB file and before loading the Dorado's control store. |

The following parameters may be included on the command line:

| | |
|---|---|
| *inputFile*/I | Specifies an input file name (the /I is optional). |
| *label*/L | Specifies the label name whose value is to be a pointer to the microcode image (rather than the default LoadRamImage). This is applicable only in conjunction with the /B global switch. |
| *filename*/O | Specifies the output file name, overriding the default implied by the global switch. If neither /B nor /E has been specified, defaults to EB-format. |
| *address*/S | Specifies the octal starting address of the microcode in the input file whose name *precedes* this parameter. If the microcode is being written to a file (BR or EB), this defaults to 1076, the address of the label InitMap (full bootstrap) in all emulators. However, if the microcode is being loaded directly into the Dorado's control store, this defaults to 1070, the address of the label RestartEmulator, which bypasses the boot sequence. |

It is permissible to specify more than one input file. In that case, LoadMB creates multiple microcode images concatenated together (and put in a single file if one is being written). When the resulting file is read into memory and passed to LoadRam, only the first microcode image is loaded into the Dorado's control store and started; but that microcode is passed a pointer to the next microcode image. This is how microcode overlays are managed.

For example, the command:

>LoadMB/e DoradoMesa.eb/o Mesa.mb 1076/s

reads microcode from Mesa.mb and creates an Ether-bootable file DoradoMesa.eb which has a starting address of 1076B. The command:

>LoadMB/e DoradoInitialMesa.eb/o InitialSelect.mb 406/s Mesa.mb 1076/s

creates an Ether-bootable file DoradoInitialMesa.eb consisting of InitialSelect.mb (with a starting address of 406B) followed by Mesa.mb (starting address 1076B).

### *3.3 Installing emulator microcode on boot servers*

At present, the only boot servers capable of booting Dorado and Dolphin microcode are the ones in Alto and Pilot Gateways and in IFSs. Installing microcode boot files is best done by a Gateway maintainer. The following instructions are for the Gateway maintainer's benefit and are therefore not tutorial.

Boot file numbers beginning with 3000 octal are set aside for microcode booting. The Dolphin uses numbers starting at 3000, and the Dorado starting at 3100. The current assignments for Dorado microcode boot files are:

| | |
|---|---|
| 3110 | DoradoMesa.eb |
| 3111 | DoradoSmalltalk.eb |
| 3112 | DoradoLisp.eb |
| 3113 | DoradoCedar.eb |
| 3114 | DoradoTest.eb |

These correspondences must be established in the Gateway's parameter file (*gatewayName*.txt).

When a Dorado or Dolphin issues a microcode boot request, it specifies a microcode file number *offset*, and the server adds 3000 to it in order to select the correct boot file. That is, the numbers 110 through 114 are wired into the Dorado's Initial microprogram for selecting Alto/Mesa, Smalltalk, Lisp, Cedar, and Test respectively. However, the boot servers update these boot files using the normal boot update protocol (as opposed to the special microcode boot protocol), and use the regular boot file numbers beginning at 3000 while doing so.

To update a microcode boot file, use LoadMB to create the desired EB-format file, and then use FTP to store it on the nearest Alto Gateway.

### *3.4 LoadRam*

LoadRam is a microcode routine that is present in Initial and in all emulators. It loads the Dorado's control store and registers (IM, IFUM, and RM) with microcode and data taken from main memory, and optionally starts it. LoadRam may be invoked by opcodes in the Alto and Mesa instruction sets.

The microcode is taken from an indefinitely long array of Items in main memory. (An EB-format file consists of an overhead page followed by such an array.) Each Item is a 4-word block, as follows:

```
Item: TYPE = MACHINE DEPENDENT RECORD
  [
  extraIM: [0..17B], unused: [0..777B], type: [0..7B],
  addr: WORD,
  word0: WORD,
  word1: WORD
  ]
```

```
ItemType: TYPE = {IM, IFUM, End, RM}
```

The array consists of any number of IM, IFUM, and RM items and is terminated by an End item.

In an IM item, addr is the absolute IM address, word0 and word1 are the left and right halves of the data, and extraIM contains the concatenation of LHParityBad, RStk[0], RHParityBad, and Block. In an IFUM item, addr is the IFUM address (including the instruction set number as the two high-order bits), and word0 and word1 are the left and right halves of the data. In an RM item, addr is the RM address and word0 is the data. (Any data for IM addresses 7600B through 7677B is ignored, as this is where the LoadRam microcode resides. There must not be any data for RM addresses 32B through 36B, else LoadRam will malfunction.)

In an End item, word0 is a checksum and word1 is the microcode starting address. The checksum should be such that the two's-complement sum of all the words of all the items (including the End item) is zero. LoadRam itself does not check the checksum; this is the responsibility of higher-level microcode or software. (For example, before calling LoadRam, the Initial bootstrap program checks the checksum of the microcode that it loads from the disk or receives from a boot server.)

In the Mesa instruction set, LoadRam is opcode MISC 3, and it is invoked by:

```
LoadRam[firstItemMinus1: LONG POINTER, flag: BOOLEAN]
```

where *firstItemMinus1* points to the word *before* the first Item of the array.

In the Alto instruction set, LoadRam is opcode 61036B, and it accepts a *short* pointer to the first Item (not ItemMinus1) in AC0 and the flag in AC1. (These instructions are defined identically on the Dolphin, though of course the interpretation of the Items is different.)

The interpretation of *flag* is as follows. If *flag* is TRUE, LoadRam turns off tasking (and does in-line memory refresh on the Dolphin), loads the control store from the Item array, and jumps to the starting address given in the End item with tasking still turned off. This operation is suitable for complete microcode replacement. The only requirement is that the new microcode have LoadRam in the same place, since LoadRam will refuse to overwrite itself.

If *flag* is FALSE, LoadRam leaves tasking on, loads the control store from the Item array, ignores the End block, and resumes normal emulation at the next opcode (Alto or Mesa). This operation is intended for microcode overlays. By convention, it must be assured that the new microcode is placed in such a way that it does not overwrite any existing microcode that is currently being executed by any task.

If LoadRam returns, it passes back a pointer to the first word after the End block, which may be the beginning of another microcode image (e.g., an overlay). More precisely, the Mesa LoadRam leaves a long pointer *above* the top-of-stack, from where it can be recovered by doing two Inline.PUSHes. (This pointer is *not* offset by one.) The Alto LoadRam simply returns the updated pointer in AC0.

### *3.5 Software-initiated boot*

The effect of a user-initiated 3-push boot may also be invoked under program control. At the microcode level, this is accomplished by executing Dorado manifold operation 2264B and then halting. Currently the only language-level access to this operation is ProcessorFace.BootButton in Pilot.

During a software-initiated boot, it is possible to pass a boot parameter to Initial that specifies what microcode Initial is to load. The parameter is a microcode file number, which is a boot file number minus 3000B. At the time of the boot, the hardware stack must contain exactly three items (STK[1] through STK[3]): the microcode file number, a seal of 56623B, and a checksum whose value is such that the sum of the three words is zero. The language-level access to this operation is MicrocodeBooting.BootSpecificMicrocode.