TO: MSPM Distribution
FROM: T. H. Van Vleck
SUBJ: MSPM BL.4.01
DATE: 02/29/68

This section has been revised to reflect the actual implementation.

Published: 02/29/68
(Supersedes: BL.4.01, 03/24/67)

Identification

Bootstrap 1
A. Bensoussan, T. H. Van Vleck

Purpose

Bootstrap 1 receives control from the bootload program, and must satisfy the interface conditions described in BC.4.1. It reads the rest of itself and loads from the MST all the segments needed to create the environment described in the bootstrap initializer's overview. Then it transfers control to bootstrap 2 which initializes the segments loaded.

Discussion

Bootstrap 1 is written in assembly language; the first part is executed in absolute mode and must be self-relocating, that is, capable of execution in any absolute core location. When the appending mode is set, bootstrap 1 must be a Master Mode procedure because it issues a connect instruction for reading the MST.

It cannot accept any fault; therefore all inter-segment references must be done through base address registers or ITS pairs pre-set.

No call macro can be issued because the base address registers are not paired and there is no stack. Bootstrap 1 does not have any associated linkage section.

Since the hardware configuration is not known, 64K of core is assumed to be available immediately following the base address of the GIOC used in the Bootload.

At the end of Bootstrap 1, control is transferred to the entry point of Bootstrap 2 which is, by convention,

<Bootstrap 2>|0

The different steps executed in Bootstrap 1 are:

Step 1: Set temporary descriptor segment and read the rest of bootstrap 1.

Step 2: Create final descriptor segment and switch to it.

Step 3: Set the fault-interrupt vector.

Step 4: Initialize SLT.

Step 5: Initialize the physical record buffer.

Step 6: Load the rest of MST collection 1 and create SLT entries.

Step 7: Transfer to bootstrap 2.

A description of each step is given in this section.

Tape reading is performed according to the same algorithm used by the initialization Tape Reader (BL.6.02), but status handling is less sophisticated.

For any terminology concerning the MST, see BL.1.01.

Step 1

Read the rest of Bootstrap 1:

This step can be thought as the continuation of the diode program. It is directed by two fundamental ideas:

a- minimize the amount of code in absolute mode

b- make this step as short as possible because all its code must be contained in the first physical record of the MST.

In order to satisfy (b) the following assumption is made:

While the rest of bootstrap 2 is read, no repeated physical records are expected in the MST.

The actions taken in this step are the following:

1. Create a temporary descriptor segment:

   The number of physical records to be read in step 1 is determined and the base address of the descriptor segment is obtained. This descriptor segment is unpaged and describes the following unpaged segments:

   a. Itself

   b. Interrupt vector used by the bootload

c. Mailboxes used by the bootload

d. Bootstrap 1

2. Initialize base address registers 0-3 in such a way that they are external registers containing the appropriate segment numbers.

3. Set the appending mode by executing the two instructions:

```
LDBR    base address of the descriptor segment

TRA     b1|* + 1
```

Control goes to the instruction immediately following the TRA instrucion in the bootstrap 1 segment.

4. Set the interrupt vector:

In any GIOC, status channel 0 and 1 are assigned interrupt cell numbers between 0 and 11. Since the bootstrap 1 program does not know the precise hardware configuration, all the pairs 0 to 11 in the interrupt vector are set in such a way that, when an interrupt occurs:

a. the control unit information is stored in

```
<bootstrap 1>|control_unit
```

b. control goes to

```
<bootstrap 1>|exp_interrupt
```

where the control unit is restored.

Setting the interrupt vector in the way described above implies that the interrupt must be requested through status channel 0 or 1.

5. Read the rest of bootstrap 1:

As many CIOC's as needed are issued. The necessary information concerning the CONNECT operation is found from word 0 in the mailboxes, which was the connect operand word used by the bootload program (see BC.4.01). Also, a CIW will be found in loc. 14 of the mailbox, left over from the hardware bootload.

The interrupt must be requested through status channel 0 or 1.

The physical header and trailer are directed to working storage while the logical information contained in the physical record is loaded at the appropriate location in bootstrap 1.

While waiting for the interrupt, the program loops, testing the status.

All the physical records read are assumed to be correct; no check is made on the next physical record. Following each record, if GIOC status is not identical to that expected, the program stops.

Step 2

Create the Multics Initializer's descriptor segment and Swap DBR:

The descriptor segment created here will be used during all the Multics Initialization. The first part of it will define the Multics template descriptor segment.

Therefore, hardcore supervisor segments will be assigned segment numbers from 0 to n-1, while initialization segments will be assigned segment numbers from n to 2n-1.

The value chosen for n is n = 2048.

The segment number assignment is as follows (see figure 1):

Segment 0 = Descriptor segment

1 = Fault vector

2 = Mailbox for bootload GIOC

3 = Second GIOC mailbox

4 = Drum mailbox

Segment n+0 = SLT

n+1 = name table (associated with the SLT)

n+2 = physical record buffer

n+3 = bootstrap 1.

All segments, even the descriptor segment, are paged.

For each of these segments a page table must be manufactured and pages allocated in such a way that no missing page fault will occur in the rest of the bootstrap initializer (1 and 2).

The actions taken in step 2 are as follows:

1. Determine the processor base address Po:

   Po is assumed to be an address zero modulo 1024.

   In the code, Po is obtained from Bo by setting the 10 least significant bits to zero.

2. Create a page table and a segment descriptor word for the following segments:

   - Descriptor segment
   - Fault vector
   - Mailbox for bootload GIOC
   - Mailbox for 2nd GIOC
   - Mailbox for drum
   - SLT
   - Name table
   - Physical record buffer
   - Bootstrap 1

3. Switch to the new descriptor segment:

   a. Copy the segment descriptor word of bootstrap1 from the current descriptor segment to the new one, at the same position.

   b. Load the DBR with the base address of the new descriptor segment's page table.

   c. Initialize base address registers as described above.

Step 3

Set the fault-interrupt vector:

The fault vector segment contains one 64 word block for faults (Block 0) followed by eight 64 word blocks for interrupts (Blocks 1 to 8). Contiguous with these 9 blocks, it contains 4 additional 64 word blocks (Blocks 9, 10, 11, 12).

Block 9 is reserved for 32 ITS pairs; ITS pair #i ($0 \leq i \leq 31$) is used by the TRA instruction located in the pair #i of the fault vector.

Block 10 is reserved for 32 ITS pairs for the SCU instructions in the fault vectors.

Block 11 is reserved for 32 ITS pairs; ITS pair #j ($0 \leq j \leq 31$) is used by the TRA instruction located in the pair #j in any of the blocks 1 to 8, of the interrupt vector.

Block 12 is reserved for 32 ITS pairs for the SCU instructions in the interrupt vector.

These 13 blocks are initialized by Bootstrap 1 in such a way that:

a- Following any fault or interrupt the control unit is stored in <bootstrap 1> | control_unit.

b- Following any fault, control goes to <bootstrap 1> | unexp_fault which causes the program to stop.

c- Following interrupts 0 to 11 coming from the bootload GIOC (they are the only expected interrupts), control goes to <bootstrap 1> | exp_interrupt, where the control unit is restored.

d- Following any other interrupt, control goes to <bootstrap 1> | unexp_interrupt which causes the program to stop.

Step 4

Initialize the SLT:

The SLT and the name table associated with the SLT are initialized as described in BL.2.01.

One entry is created in the SLT for each existing segment. These are canned entries assembled into the program.

Step 5

Initialize the physical record buffer:

This step has to be done so that the input routine described at the end of this section can work properly.

The structure of this physical record buffer is as follows:

word 0: bits 0-17 contain the count of words that have already been moved from the physical record. The count is an integer such that:

$$0 \leq \text{count} \leq 256$$

words 1 to 8: contain the physical header

words 9 to 265: contain the logical information

words 266 to 272: contain the physical trailer

words 1 to 272 constitute the "current buffer".

Contiguous with the "current buffer", 272 words are reserved for the "next buffer". The next buffer is used in the input routine to check the validity of the current buffer.

Step 5 consists of the following actions:

1. Copy in the current buffer the last physical record read in step 1.

2. Store in word zero the count of words which are part of bootstrap 1 in the current buffer.

Step 6

Load the rest of MST collection 1.

A tape input routine is used that is responsible for reading physical records from the MST and moving a requested number of words from the physical record buffer (described in step 2) to the requested area.

This input routine is not called with a call macro but with a TSXi instruction and returns to the caller using index register i. It needs two arguments:

a. the length, in words, of the logical MST to be moved.

b. the initial location where the requested words have to be moved. This initial location isigiven by an ITS pair.

The following segments and their associated linkage sections are required in collection 1 (see also BL.1.01):

- Bootstrap2
- SLT manager
- Hardcore stack ("stack_dummy")

The algorithm used in step 6 for loading the segments mentioned above is the following:

1- Call the input routine to move the "control word" to a local buffer of the bootstrap 1.

   - If the control word is a "mark control word", call the input routine to read the "mark" and go to step 7 (below).
   - If the control word is a "segment control word", go to ERROR.

2- Call the input routine to move the "Header" to a local buffer of the bootstrap 1. (The length of the header was contained in the "control word").

   - Create entry in the SLT
   - Create entry in the descriptor segment
   - Create page table according to the maximum length
   - Allocate pages for current length

3- Call the input routine to move the "control word" to a local buffer of the bootstrap 1.

   - If it is not a "segment control word" go to ERROR.

4- Call the input routine to move the "segment" in pages allocated in 2. (The length of the segment was contained in the segment control word).

5- Go to 1.

Step 7

We have now loaded all of collection 1, and are ready to call bootstrap2. First, we must perform the following "cleanup" actions:

a) read the collection mark from the MST. Its value is ignored.

b) load index registers with numbers interesting to Bootstrap2.

Set X1 = segment # of slt_manager

X2 = " " of SLT

X3 = processor tag

c) pair the bases. Bootstrap2 executes in slave mode, so that base SB in particular must be loaded with the segment number of the stack and locked by bootstrap1.

Then we transfer through an ITS pair to location 0 of bootstrap 2.

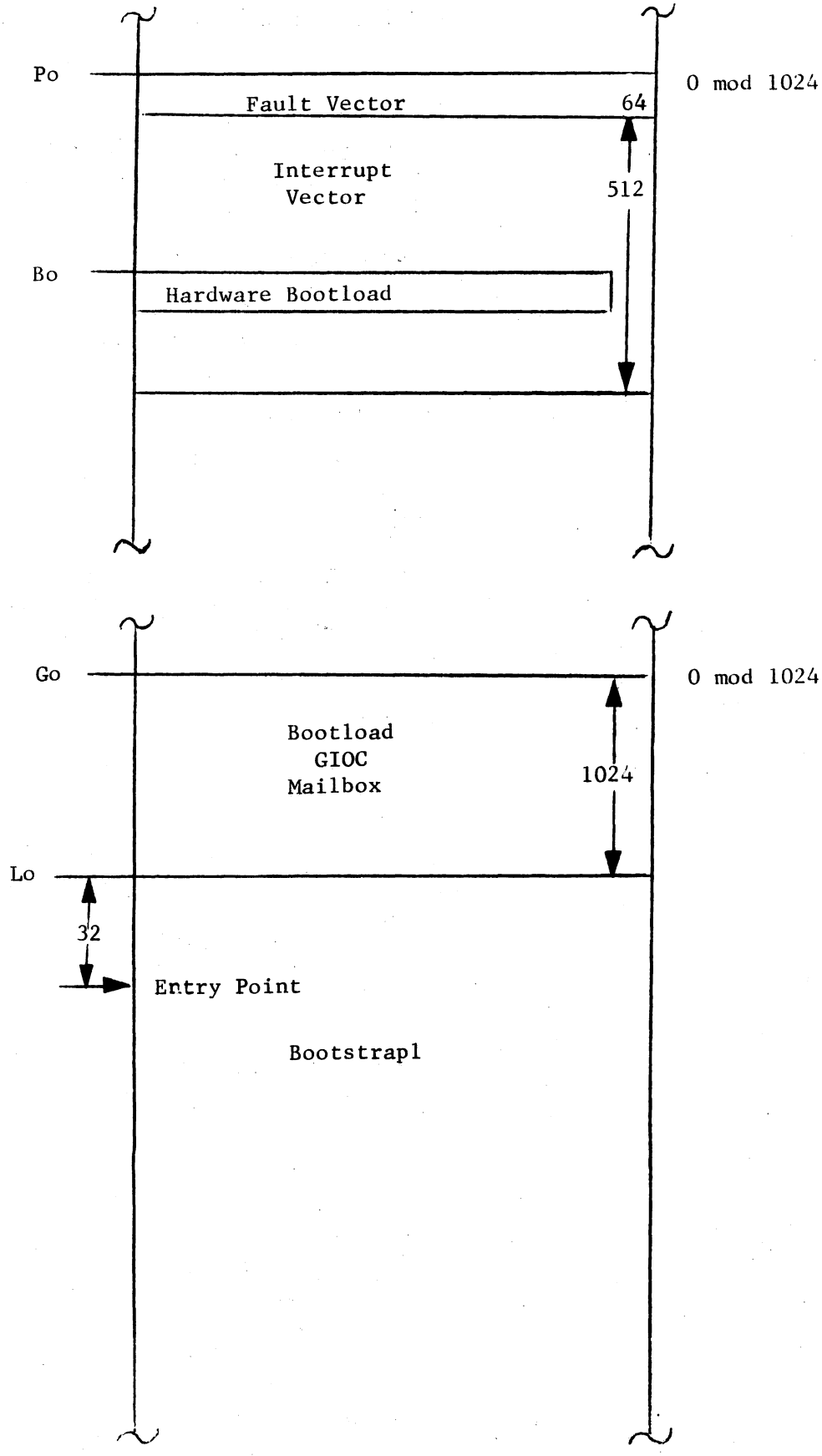

Figure 1: Environment of Bootstrap 1