Published: 03/23/67

Identification

LDBR Procedure
R. L. Rappaport

Purpose

Ldbr (Load Descriptor Segment Base Register) can only be executed in master mode. The ldbr procedure is a master mode procedure used to isolate the ldbr instructions needed in the Process Switching Module (see Sections BJ.5.00-BJ.5.02).

Discussion

An ldbr instruction cannot be executed in slave mode and in the Process Switching Module there are three places where this instruction must be executed. In each of these places a call will be made to one of the three entry points provided by the ldbr procedure. The reason for three distinct entries is that each ldbr must be executed within a certain context of instructions, which is different in each case. The three entry points are ldbr_1, ldbr_2, and ldbr_3. They are all called with a standard calling sequence. That is:

1 - call ldbr_1 (ds);

2 - call ldbr_2 (ds);

3 - call ldbr_3 (ds);

where in each case <u>ds</u> is the value with which the descriptor segment base register is to be loaded.

Ldbr 1

Ldbr_1 is called in swap_dbr (see Section BJ.5.01). The context in which the ldbr instruction is executed in ldbr_1 is dictated by the nature of swap_dbr. Swap_dbr is called when one process (the caller) wants to give unconditional control of a processor to another process (the target). In order for the target process to be able to service interrupts on this processor, certain information must be accessible in the target's address space. In particular, the Processor Data Segment, of this processor, must be a segment in this address space and the target's process id must appear in this Processor Data Segment before any interrupts can be serviced. Therefore the ldbr instruction must be followed by three instructions which store these data items into the target's address space and the three instructions must be executed while the processor is inhibited in order to prevent the servicing of interrupts during this time.

The steps taken by ldbr_1 are tabulated below. It should be noted that this routine does not do a standard save. This facilitates the creation of a stack for loading processes. Also note that the instructions before the ldbr are executed in the address space of the caller and all references to the descriptor segment or the Process Data Segment refer to those of the caller process while after the ldbr, such references refer to the segments of the target.

1. The caller stores the current value of base register sp into its Process Data Segment. This enables the caller to reset its stack pointer the next time it resumes control.

2. Index register 1 is loaded with the segment number of the Processor Data Segment. This step implies this segment has the same number in each process. This register will be used as an index into the descriptor segment in order to pick up and store the segment descriptor word for the Processor Data Segment.

3. The segment descriptor word of the Processor Data Segment, for this processor, is loaded into the A-register. This is done in order to pass along this word to the target. The segment descriptor word is obtained from the caller's descriptor segment.

4. (Inhibit on) The ldbr is executed.

5. (Inhibit on) The A-register is stored into the location in the target's descriptor segment reserved for it.

6. (Inhibit on) The combined AQ register is loaded with the process id of the target. This id is obtained from the target's Process Data Segment.

7. (Inhibit on) The AQ register is stored into the Processor Data Segment.

8. Base register sp is loaded with the value stored the last time the target was running.

9. The other base registers are restored with their previous values. The values were stored in the process concealed stack at the time of the call to ldbr_1.

10. The registers are restored with the values they had when the call to ldbr_1 was made by the target.

11. A return transfer is made to swap_dbr.

The actual machine code contained in ldbr_1 is listed below. <pds>, <prds>, <ds> are Process Data Segment, Processor Data Segment and descriptor segment respectively.

```
ldbr_1:  stbsp   <pds>|[last_sp]

         ldx1    <prds>|[segno]

         lda     <ds>|0,1

         inhibit on

         ldbr    ap|2,*

         sta     <ds>|0,1

         ldaq    <pds>|[processid]

         staq    <prds>|[processid]

         inhibit off

         ldbsp   <pds>|[last_sp]

         ldb     sp|o

         lreg    sp|8

         rtcd    sp|20
```

Ldbr_2

Ldbr_2 is called from ready-him (see Section BJ.5.02). It is called using the Processor Stack (contained in the Processor Data Segment). Ldbr_2 is simpler than ldbr_1 in that the value of sp need not be saved and restored since both processes use the same stack and also in that the target's process id is not stored into the Processor Data Segment since the caller is still considered the process in charge. The other steps are quite similar to the ones in ldbr_1 and the code is presented below.

```
ldbr_2:   ldx1      <prds>|[segno]
          lda       <ds>|0,1
          inhibit   on
          ldbr      ap|2,*
          sta       <ds>|0,1
          inhibit   off
          ldb       sp|0
          lreg      sp|8
          rtcd      sp|20
```

Ldbr_3

Ldbr_3 is called in ready-him in order to return the processor to the caller. At this point, all that needs to be done is to switch descriptor segments, restore the bases and return. The code is presented below.

```
ldbr_3    ldbr      ap|2,*
          ldb       sp|0
          rtcd      sp|20
```

Wrapup

Since this is a master mode procedure, entry can only be made at its initial entry. There a few instructions will be located which validate the call. In particular, these instructions will verify that the address specified by the argument in the call actually points to a descriptor segment. If trouble is observed an error condition will be noted and action will be taken similar to the action taken at the time of a trouble fault. If no trouble is encountered, a branch will be made to the appropriate entry point.