This application note describes the process of setting up the on-chip table lookaside buffers (TLBs) for translation, without using page tables, for classic PowerPC™ processors with multiple areas of memory mapped one-to-one. The application note applies only to MPC745x and MPC744x (with software table search enabled in non-extended mode).

1 Introduction

TLBs are the on-chip “caches” for page table entries for classic PowerPC memory management units (MMUs). On processors that support software table searching and on-chip TLBs, you can set up the TLBs indirectly and use them as extra BATs without the need to set up page tables in memory. The source code works exclusively on the MPC745x and MPC744x. On processors with software table searching enabled, after a TLB miss exception is taken, the interrupt handler can load a TLB entry for the offending address by executing a \texttt{tlbld} (or \texttt{tlbli}) instruction. For MPC745x and MPC744x processors, the TLB entry information that is loaded is contained in TLBMISS, PTEHI, and PTELO registers. This application note describes how you can load the on-chip TLBs by using these registers as conduits of information with some restrictions (see Section 5, “Limitations”).
2 Terminology

The following terms are used in this document:

**BAT** Block address translation mechanism. A set of registers that contain the translation information and access privilege for blocks of memory.

**MMU** Memory management unit. This on-chip unit manages memory accesses on a processor.

**MSR** Machine state register. Contains information on various states of the processor.

**Page** 4 Kbytes of contiguous memory starting at a 4-Kbyte boundary.

**PTE** Page table entries. These 8-byte entities hold translation information for 4 Kbytes of memory (a page).

**SRR1** Machine status save restore register 1. This register stores information when an exception is taken.

**TLB** Translation lookaside buffers. These on-chip storage elements store (cache) recently accessed PTEs.

3 Types of Translation

Processor-generated memory accesses require address translation before they go out to the memory subsystem. Instruction and data access translations are enabled through two bits (IR and DR, respectively) in the machine state register (MSR). When translation is disabled, the processor is said to be in real addressing mode. In this mode, all memory is mapped one-to-one with effective memory/cache attributes (WIMG) settings of 0001 or 0011. When translation is enabled, address translation is performed either through BATs or page tables/TLBs. Figure 1 summarizes the translation types.

![Figure 1. Address Translation Types](image-url)
For details on the translation types, refer to the *Programming Environments Manual for 32-Bit Implementations of the PowerPC™ Architecture (PEM)*.

### 4 Configuring On-Chip TLBs

This section describes the steps to set up the on-chip TLBs when software table searching is used. Note that the MMU should be off (MSR[IR], MSR[DR] cleared) when this setup is run. After setup completes, the MMU is turned back on.

#### 4.1 Enabling Software Table Searching

Software table searching should be enabled (HID0[SWEN]=1) to perform the steps described in the following subsections.

```asm
//enable software table searching HID0[SWEN]=1
mfspr r3, 1008
oris r3,r3,0x100
mtspr 1008, r3
sync
```

#### 4.2 Invalidating the TLB Entries

Before programming the on-chip TLB entries, you should invalidate all of them first to ensure that only the ones programmed later are valid. The following assembly code invalidates both ways of the on-chip TLBs for a two-way, set-associative TLB array.

```asm
//invalidate all TLB entries
li r5, 64
mtctr r5
invtlb1:
   tlbie r3
   addi r3, r3, 0x1000
bdnz invtlb1
```

//invalidate all TLB entries
li r5, 64
mtctr r5
invtlb1:
   tlbie r3
   addi r3, r3, 0x1000
bdnz invtlb1
4.3 Configuring the Segment Registers

The segment registers contain the virtual segment IDs (VSIDs) for a process. The upper 4 bits of the effective address specify which segment register to use. If more than one segment register is used, each one must have a unique VSID. To accomplish this, the following code loads the VSIDs with consecutive numbers. In this code, r3 and r4 contain the starting and ending addresses, respectively, of the memory area to be covered by the TLBs.

```
//set up SRx
rlwinm r8, r3, 4, 28, 31 //extract 4 MSBs
rlwinm r9, r4, 4, 28, 31 //extract 4 MSBs
srx_set:
   bl      set_srx                 //expects r8=value r9=SR index
   addi    r8, r8, 1
   cmpw    r8, r9
   ble     srx_set
Where set_srx is defined as:

//set srx registers
.globl set_srx
set_srx:
   cmpwi   r9, 0
   beq     mtsr0
   cmpwi   r9, 1
   beq     mtsr1
   //fill in the same sequence for SR2 up to SR14 here
   cmpwi   r9, 15
   beq     mtsr15
mtsr0:
   mtsr    0, r8
   blr
mtsr1:
   mtsr    1, r8
   blr
   //fill in the same sequence for SR2 up to SR14 here
mtsr15:
   mtsr    15, r8
   blr
```
4.4 Loading the TLB Entries

TLB entries have a format similar to the PTEs shown in Figure 2. To load the on-chip TLB, the upper and lower portions of a TLB entry are loaded to different registers. Then a tlbld (TLB load data) or tlbli (TLB load instruction) is executed.

![Figure 2. Page Table Entry Format](image)

Each TLB entry is selected by EA[14–19]. All TLBs on the MPC745x and MPC744x devices are two-way, set-associative, and EA[31] selects the way that is written on a tlbld or tlbli instruction. Take care not to write to a TLB entry more than once (see caveats in Section 5, “Limitations”).

4.4.1 Way 0 Selection and Loop Setup

Before we load TLB entries for each of the 4 Kbytes of memory address, we set up the way and loop counter. The loop counter counts up to 64. In the following code, r3 and r4 contain the starting and ending page addresses, respectively, of the memory area to be covered.

```c
//EA[31] = 0 by default

void load_TLB_entries()
{
    //LOAD TLB entries
    ldtlbs: cmpw r3,r4
        bgt exit_ldtlbs
        ldtlbs: cmpw r3,r4
        bgt exit_ldtlbs

        //set PTEHI = 0x800000 || 0b00 || EA[4:9]
        rlwinm r5,r3,10,26,31
        oris r5,r5,0x8000
        mtspr 981, r5
        sync
}
```

4.4.2 The TLBMISS Registers

The TLBMISS register is a read-only register in the MPC745x/744x. We do not write to this register.

4.4.3 The PTEHI Register

This register contains the upper portion of the PTE/TLB entry upon a TLB miss. We load these registers with what we want in the upper portion of the PTE/TLB. In this code, r3 contains the effective address.

```c
//set PTEHI = 0x800000 || 0b00 || EA[4:9]

void load_PTEHI()
{
    //set PTEHI = 0x800000 || 0b00 || EA[4:9]
    rlwinm r5,r3,10,26,31
    oris r5,r5,0x8000
    mtspr 981, r5
    sync
}
```
4.4.4 The PTELO Register

The PTELO register contains the lower portion of the PTE/TLB entry upon a TLB miss. We load it with what we want in the lower portion of the PTE/TLB. We set up a WIMG of 0000 and R and C are both set to indicate that the processor does not need to perform any PTE updates in the page table. This is important because in this setup we do not have page tables in memory.

```asm
//set PTELO = PA[0:19] || 0x182
rlwinm r5, r3, 0, 0, 19
ori r5, r5, 0x182
mtspr 982, r5
sync
```

4.4.5 Executing the tlbld/tlbli Instruction

After the PTEHI and PTELO registers are set up, the tlbld (or tlbli) instruction is executed to load the information into the on-chip TLB. The tlbld/tlbli instruction takes the effective address as a parameter.

```asm
//load tlb entry indexed by EA[14:19]
tlbld r3  //or tlbli for instruction TLB
sync     //just to be safe
```

4.4.6 Way 1 Selection and Loop Completion

After loading Way 0 entries we switch to Way 1. The address range of the memory should not exceed 64 pages (or 32 pages) for each way (see restrictions in Section 5, “Limitations”).

```asm
//increment EA/PA by 0x1000
addi r3, r3, 0x1000
bdnz ldtlbs  //continue if counter > 0
//if we filled all 64 entries of one way, toggle to the second way
//reset counter
li r5, 64
mtctr r5

//toggle EA[31/WAY]
xori r3, r3, 1
//continue
b ldtlbs
```
5 Limitations

Using TLBs as discussed in this document has some restrictions. First, this technique applies only to processors that implement on-chip TLBs and that can also use software table walk. The source code works only on MPC745x and MPC744x devices. Second, we do not get to choose which TLB to use. The TLB entry to be used is selected by EA[14–19] of the effective address. Therefore, there are limitations on the size of the memory area that can be covered. The memory area to be covered should be within two memory area windows described as follows:

- Each window is contiguous
- Each window is no more than 64 pages

For example, 0, 0x40000 and 0x80000 lie over three different 64-page windows of memory. All three windows cannot be covered by the TLB entries at the same time. The two ways of TLB entry 0 handle two of the addresses, but the third address cannot have a TLB entry without overwriting one of the ways.

0, 0x3F000, and 0x40000 can be covered by the TLBs at the same time. 0 and 0x3F000 lie within 64 pages of each other (using TLB entry 0 and TLB entry 63 respectively, and 0x40000 is covered by Way 1 of TLB entry 0).

6 Document Revision History

Table 1 provides a revision history for this application note.

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Substantive Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10/2004</td>
<td>Initial public release</td>
</tr>
<tr>
<td>1</td>
<td>5/2006</td>
<td>Corrected an error in section 4.2.</td>
</tr>
</tbody>
</table>