1. Reverse Engineering of Page Table

Assuming dynamic relocation is performed with a linear page table, then given the page table, we can translate the virtual address to physical address. Now let’s try to reverse the process, i.e., given a series of VA-PA translation, reconstruct some entries of the page table.

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000c17c</td>
<td>0x003a017c</td>
</tr>
<tr>
<td>0x00009e4d</td>
<td>0x00137e4d</td>
</tr>
<tr>
<td>0x00004021</td>
<td>Invalid</td>
</tr>
<tr>
<td>0x0000e8e5</td>
<td>0x001208e5</td>
</tr>
<tr>
<td>0x0000fb97</td>
<td>0x00121b97</td>
</tr>
</tbody>
</table>

The page table entry is formatted in 32 bits and the high-order (left-most) bit is the valid bit. Therefore, you should represent the page table entry in the **32-bit hexadecimal format**, e.g., 0x8000019e, for the valid entry. If the entry is not valid, fill in 0x00000000. If the entry cannot be determined from this translation table, fill in "TBD".

(a) Assuming there are **16 entries** in the page table.

i. What is the minimum page size?

ii. Construct the page table with this page size
iii. Assuming the TLB is initially empty and if these 5 virtual addresses in
the translation table are accessed by the program sequentially, will there
be any TLB hit? If so, list the VPNs that cause a TLB hit.

(b) Assuming there are 8 entries in the page table.
   i. What is the minimum page size?
   
   ii. Construct the page table with this page size

   iii. Assuming the TLB is initially empty and if these 5 virtual addresses in
      the translation table are accessed by the program sequentially, will there
      be any TLB hit? If so, list the VPNs that cause a TLB hit.
2. **TLB**

Assume a system with a simple linear page table and a hardware managed TLB. Parameters:

- Virtual address space size = 64 bytes
- Physical memory size = 128 bytes
- Page size = 16 bytes

Let’s assume we have an array of **ten 4-byte integers** in memory, starting at **virtual address 0x10**. Let’s further assume that we access all the elements in the array, sequentially, starting at array index 0. The references to these array elements are the only memory references that we are interested in. Let’s also assume the TLB was initially empty before accessing the array.

(a) For each element that is accessed in the array, write if it’s a **TLB Hit (H)** or a **TLB Miss (M)**. (e.g. HMMHMHmmM)

(b) What is the **TLB hit ratio** for accessing the entire array?

(c) If the page size is increased from 16 bytes to **32 bytes**, for each element that is accessed in the array, write if it’s a **TLB Hit (H)** or a **TLB Miss (M)**. (e.g. HMMHMHmmM)

(d) What is the **TLB hit ratio** for accessing the same array with the page size of 32 bytes?
3. Multilevel Page Table

Assume a system with a 2-level page table.

Parameters:
- page size = 32 bytes
- virtual address space size = 32 KB
- physical memory size = 4 KB
- Size of one Page Directory Entry (PDE) = 1 byte
- Size of one Page Table Entry (PTE) = 1 byte
- Value of Page Directory Base Register (PDBR) = \text{122} \text{ (decimal)} \text{ [This means the page directory is held in this page]}

The format of the PDE and the PTE is simple. The high-order (left-most) bit is the VALID bit. If the bit is 1, the rest of the entry is the PFN. If the bit is 0, the page is not valid.

You are given two pieces of information to begin with. First, you are given the value of the page directory base register (PDBR), which tells you which page the page directory is located upon. Second, you are given a complete dump of each page of physical memory in the next 2 pages. A page dump looks like this:

page 0: 1d 1d 06 08 05 00 0e 10 ...
page 1: 16 1b 14 19 00 1d 16 12 ...
page 2: 7f 7f 7f 7f 7f 7f 7f 7f ...
...

which shows the 32 bytes found on pages 0, 1, 2, and so forth. The first byte (0th byte) on page 0 has the value 0x0d, the second is 0x0f, the third 0x06, and so forth. For each virtual address:

- write down the physical address it translates to AND the data value at this physical address, OR
- if it is a segmentation fault (an out-of-bounds address) write the reason for this segmentation fault (Invalid PDE OR Invalid PTE).

Write all answers in hexadecimal.

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address / Seg Fault</th>
<th>Data Value</th>
<th>Reason for Seg fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x21e1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0325</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x7570</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x7a36</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>