PRACTICE PROBLEMS RELATED TO HOMEWORK 6 (SOLUTIONS SHOULD NOT BE SUBMITTED AND THEY WILL NOT BE GRADED)

Problem 1

Consider a virtual memory system where a TLB access takes 4ns and there is a single level of a set associative, writeback data cache with the following parameters: indexing the cache to access the data portion takes 10ns, indexing the tag array of the data cache takes 8ns, tag comparisons take 3ns, and multiplexing the output data takes 1ns. Assume these are the only parts that affect the cache access time.

Consider four possible configurations for the cache:
(I) physically-indexed, physically-tagged cache
(II) virtually-indexed, virtually-tagged cache
(III) virtually-indexed, physically-tagged cache
(IV) physically-indexed, virtually-tagged cache

Assume that the bits used to index the data cache have different values in the corresponding physical and virtual addresses. Assume that the values of any bits that are needed to index the data cache must be known before the cache is accessed (i.e., the bits cannot be predicted for the cache access). Assume that all accesses that use the TLB are hits and all accesses to the cache hit as well.

Given the above assumptions, for each of the above four configurations, answer the following two questions:

(A) What is the minimum amount of time it takes to get data from the cache, including any needed TLB access time? Please explain your answer.

(B) Does this configuration offer any advantage over the other configurations? If so, explain the advantage. If not, explain why.

Configuration (I): Physically-indexed, physically-tagged cache:

Since this configuration requires a physical tag and index, the virtual address must be translated before anything else can happen. So the 4ns delay of the TLB access must be added into the total. Accessing the data and tag arrays occurs in parallel. The data values are available after 4+10=14ns but the tag hit signal does not appear until 4+8+3=15ns. At this point, the correct way can be selected from the mux, which has an additional 1ns delay. Therefore, the total access time is 16ns.

This cache offers an advantage over all the other configurations in that it does not have a synonym problem.

Configuration (II): Virtually-indexed, virtually-tagged cache:

In a virtually indexed, virtually tagged cache, the TLB is not used unless there is a cache miss, at which point translation would be required. Since the TLB is not used, the access time for this cache will be the same as above, minus the initial 4ns delay due to the TLB. Specifically, the total access time is 12ns.

This cache has an advantage over configurations (I) and (IV) in that it offers a lower latency. It has no significant advantage over (III).
Configuration (III): Virtually-indexed, physically-tagged cache:

Because the tags are physical they require translation by the TLB. This results in a 4ns delay in translating the tag. However, since the index is virtual it can be sent to the cache to initiate the tag array and data array accesses while the TLB translation is happening. After 8ns, the tags have been accessed and sent to the comparators with the translated tag. After 11ns the tag hit signal is generated. Accessing the data array occurs in parallel, so its 10ns delay is hidden. After a 1ns delay from multiplexing the data, the total access time is 12ns.

This cache has an advantage over configurations (I) and (IV) in that it offers a lower latency. It has no significant advantage over (II).

Configuration (IV): Physically-indexed, virtually-tagged cache:

The index is physical and requires translation by the TLB. This results in a 4ns delay in translating the index. Although the virtual tag is available at time zero, nothing can be done with it until an index is available. Once the index is translated, it can be used to access the tag and data arrays. Data is available at 14ns while the tag hit signal comes at 15ns. After a 1ns delay from multiplexing the data, the total access time is 16ns.

This cache offers no advantage over all the other configurations. Synonyms are possible with this cache as multiple virtual tags may map to the same physical address, which can cause the tag comparators to signal a miss when, in fact, the data was actually present. This cache also has the same performance as the physically-indexed, physically-tagged cache (and worse performance than the others). This is due to the fact that nothing can be done in the cache until the index is translated. Physically-indexed, virtually tagged caches are not generally used because of the overhead of the TLB translation cannot be hidden by any other part of the cache access, making it useless to keep the tag as a virtual address and risk synonym issues.

Problem 2

Consider a system with a 4GHz processor and a 667MHz bus to memory. Consider page mode memory with a row buffer size of 16 KB. There is a cache with 64 byte blocks. Assume the cache is large enough that there are no capacity or conflict misses. Use the following simplified memory timing model:

- Assume the time taken by a read is the time to send the address (including the memory access time for that address) plus the time to transfer back the data.
- Every read transfers 64 bytes of data. This data transfer takes 4 bus cycles.
- Sending an address that hits in the row buffer (it is to the same row as the previous read) takes 9 bus cycles.
- Sending an address on another row takes 18 bus cycles.
- Only one memory request can occur at a time. The next request may begin on the bus cycle after data from the last read is returned.

Each program begins with an empty cache, and no row buffer open in memory. Consider the following code.
int A[1024*1024]; // 4 byte numbers. Begins on a row boundary
int max = MIN_INT;
for (int row = 0; row < 1024; ++row) {
    for (int col = 0; col < 1024; ++col) {
        if (max < A[row + col*1024]) {
            max = A[row + col*1024];
        }
    }
}

How many read requests for the array A will be sent to main memory? How many bus cycles will these requests take, if they are issued in order with no extra delay between requests (assume all scalar variables are allocated in registers)? What is the average number of CPU cycles per memory request?

Solution: Each read returns 64/4 = 16 elements, so there are 1024*1024/16 = 1024*64 read requests for array A sent to memory.

Each read takes 4 cycles to return data, and 9 cycles for the column address. Each row buffer holds 4096 array entries, so every fourth memory access must send a new row address as well. In total, the memory accesses take 13*64*1024 + 9*16*1024 = 999424 bus cycles.

There are 1024*64 requests, so the average number of bus cycles per request is 999424/(1024*64) = 15.25 bus cycles. Each bus cycle takes 1000/667 = 1.5ns, or 6 CPU cycles. Therefore, the average memory request time is 6*15.25 = 91.5 CPU cycles.

Problem 3

A processor has virtual memory with 4KB pages, 48-bit virtual addresses, and 36-bit physical addresses.

Part A

Could virtual address 0x776572656e6f map to physical address 0x737472616? Explain your answer.

No. The lower 12 bits are the page offset, and are not translated. 0xe6f is not the same as 0x616.

Part B

If a 24 KB shared library is already present in memory (page aligned and contiguous in virtual address space), how many page table entries are needed to map the library into the address space of a new program?

24/4 = 6 pages need to be mapped, needing 6 page table entries.
Problem 4

Part A

Consider a virtual memory system with the following parameters:
- 64-bit virtual address
- 41-bit physical address
- 8KB page size

1. How many bits of the virtual and physical address comprise the page offset?

\[ \log 8 \text{ KB} = \log 2^{13} = 13 \text{ bits for both virtual and physical address.} \]

2. How many bits long is the physical page number?

The page number is the part of the physical address not including the page offset bits: 41 - 13 = 28 bits.

3. How many bits long is the virtual page number?

The virtual page number is all the bits of the virtual address not including the page offset: 64 - 13 = 51 bits.

Part B

A program is running on a computer with a four-entry fully associative translation lookaside buffer (TLB) with the following initial contents:

<table>
<thead>
<tr>
<th>Virtual page #</th>
<th>Physical page #</th>
<th>Entry valid (1 = yes, 0 = no)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td>1</td>
</tr>
</tbody>
</table>

The initial relevant page table entries are as follows.

<table>
<thead>
<tr>
<th>Virtual page #</th>
<th>Physical page #</th>
<th>Present in memory?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>Yes</td>
</tr>
</tbody>
</table>
The following is a trace of the virtual page numbers accessed by the program. Assume the TLB employs an optimal replacement policy; i.e., one that will minimize the number of misses in the following access stream. Assume physical main memory is large enough to hold any page that is accessed. For each access below, indicate whether it produces a TLB hit or miss. For a TLB miss, indicate which virtual page # entry is replaced in the TLB, the contents of the TLB after the access, and whether the access caused a page fault.

<table>
<thead>
<tr>
<th>Virtual Page # Accessed</th>
<th>TLB hit or miss?</th>
<th>TLB entry (virtual page #) replaced</th>
<th>TLB contents after the access (only virtual page #s)</th>
<th>Write “yes” if there is a page fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Miss</td>
<td>7</td>
<td>1, 5, 6, 10</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Miss</td>
<td>1</td>
<td>5, 6, 7, 10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Miss</td>
<td>5</td>
<td>6, 7, 9, 10</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Hit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Hit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Hit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Miss</td>
<td>No need to fill this one</td>
<td>No need to fill this one</td>
<td></td>
</tr>
</tbody>
</table>