HW9.1. Virtual Memory

Q1. We want to set up a virtual memory system with **4 GiB of virtual memory**, mapped to **4 MiB of physical memory**.

Q1.1: How many bits long will our virtual addresses be?
Given the size of the virtual memory, how many bits do we need to address each byte of the virtual memory?

32 ✓ 100%

Q1.2: How many bits long will our physical addresses be?
Given the size of the physical memory, how many bits do we need to address each byte of the physical memory?

22 ✓ 100%

Q2. In order to set this up, we decide to use a **page size of 4 KiB** (4096 bytes), with a single layer page table.

Q2.1: How many bits long is the page offset?
The page offset addresses each byte of the page. Given the page size, how many bits do we need to address each byte of the page?

12 ✓ 100%

Q2.2: How many bits long is the virtual page number (VPN)?
The virtual address is partitioned into the virtual page number and the page offset.

20 ✓ 100%

Q2.3: How many bits long is the physical page number (PPN)?
The physical address is partitioned into the physical page number and the page offset.

10 ✓ 100%

Q3. We decide to include metadata and padding for each page table entry such that **each entry in our page table is exactly 32 bits (4 bytes) long**. For the purposes of this class, we assume that our page table is structured like an array of entries, with a slot for each virtual address.

Q3.1: How many page table entries are required for the full page table?
The VPN is the index of the page table. Given the VPN bits computed earlier, how many entries would the page table have?

1048576 ✓ 100%

Q3.2: How many bytes of data does the page table take?
Knowing the number of page table entries and the number of bytes for each page table entry, how many bytes is the whole page table?

4194304 ✓ 100%

Q3.3: How many pages of data does the page table take?
Given the size of each page and the computed page table size, how many pages would the page table consume?

1024 ✓ 100%

Q3.4: How many physical pages can we use for actual data, if we decide to use this page table?

Question

Value:	20
History:	20
Awarded points:	20/20

Attached files

No attached files

Given the size allocated for page tables (page tables are also stored in the physical memory) and the actual size of physical memory, how much space do we have left for the actual data?

0 ? ✓ 100%

Q3.5: With a one-level page table, how many physical memory accesses are required per virtual memory access?
The page table (stored in physical memory) contains the address translation to where the actual data is located (also in physical memory).

2 ? ✓ 100%

Q4. In order to make things more efficient, we decide to use a two-level page table instead, with the **virtual page number bits split evenly** between the two layers. We still use a page table entry size of 32 bits (4 bytes).

Q4.1: How many page table entries are in the L1 page table?
VPN1 indexes the L1 page table. How many bits do we have for VPN1? How many entries would that translate to?

1024 ? ✓ 100%

Q4.2: How many pages does our L1 page table use?
Given the size of each page table entry and the computed number of entries for L1, how many bytes does the L1 page table consume? Given the page size, how many pages would the L1 page table consume?

1 ? ✓ 100%

Q4.3: How many page table entries are in each L2 page table?
VPN2 indexes the L2 page table. How many bits do we have for VPN2? How many entries would that translate to?

1024 ? ✓ 100%

Q4.4: How many pages does each L2 page table use?
Given the size of each page table entry and the computed number of entries for L2, how many bytes does the L2 page table consume? Given the page size, how many pages would the L2 page table consume?

1 ? ✓ 100%

Q4.5: With a two-level page table, how many physical memory accesses are required per virtual memory access?
The L1 page table will point you to the L2 page table. The L2 page table will point you to the actual translation. Both of the page tables are stored in physical memory.

3 ? ✓ 100%

Try a new variant

Correct answer

Q1. We want to set up a virtual memory system with **4 GiB of virtual memory**, mapped to **4 MiB of physical memory**.

Q1.1: How many bits long will our virtual addresses be?
Given the size of the virtual memory, how many bits do we need to address each byte of the virtual memory?

32

Our addresses need to uniquely identify each byte of memory. If there are 4 GiB of virtual memory, then there should be $log_2(4GiB)$ = 32 bits for the virtual address (VA).

Q1.2: How many bits long will our physical addresses be?
Given the size of the physical memory, how many bits do we need to address each byte of the physical memory?

```
22
```

Our addresses need to uniquely identify each byte of memory. If there are 4 MiB of physical memory, then there should be $log_2(4MiB)$ = 22 bits for the physical address (PA).

Q2. In order to set this up, we decide to use a **page size of 4 KiB** (4096 bytes), with a single layer page table.

Q2.1: How many bits long is the page offset?
The page offset addresses each byte of the page. Given the page size, how many bits do we need to address each byte of the page?

```
12
```

The offset needs to uniquely identify each byte of the page. If the page size is 4 KiB, then there should be $log_2(4KiB)$ = 12 bits for the offset.

Q2.2: How many bits long is the virtual page number (VPN)?
The virtual address is partitioned into the virtual page number and the page offset.

```
20
```

If the virtual address is 32 bits and the offset is 12 bits, then the upper 32-12 = 20 bits will be the virtual page number (VPN).

Q2.3: How many bits long is the physical page number (PPN)?
The physical address is partitioned into the physical page number and the page offset.

```
10
```

If the physical address is 22 bits and the offset is 12 bits, then the upper 22-12 = 10 bits will be the physical page number (PPN).

Q3. We decide to include metadata and padding for each page table entry such that **each entry in our page table is exactly 32 bits (4 bytes) long**. For the purposes of this class, we assume that our page table is structured like an array of entries, with a slot for each virtual address.

Q3.1: How many page table entries are required for the full page table?
The VPN is the index of the page table. Given the VPN bits computed earlier, how many entries would the page table have?

```
1048576
```

The VPN is the index of the page table. Since there are 20 bits for the VPN, then there are 2^20 = 1048576 page table entries in the page table.

Q3.2: How many bytes of data does the page table take?
Knowing the number of page table entries and the number of bytes for each page table entry, how many bytes is the whole page table?

```
4194304
```

Since there are 1048576 entries in the page table and each page table entry is 4 bytes (32 bits long), then there are 1048576*4 = 4194304 bytes of data for the entire page table.

Q3.3: How many pages of data does the page table take?
Given the size of each page and the computed page table size, how many pages would the page table consume?

```
1024
```

Since there are 4194304 (4 MiB) bytes of data for the entire page table and each page is 4 KiB, then we need 4 MiB / 4 KiB = 1024 pages to contain the page table.

Q3.4: How many physical pages can we use for actual data, if we decide to use this page table?
Given the size allocated for page tables (page tables are also stored in the physical memory) and the actual size of physical memory, how much space do we have left for the actual data?

```
0
```

Note that the page table only contains translations, not data. If the entire page table is 4 MiB (solved earlier) and the physical memory is also 4 MiB (given), then there's 4 MiB - 4 MiB = 0 bytes available for actual data.

Q3.5: With a one-level page table, how many physical memory accesses are required per virtual memory access?
The page table (stored in physical memory) contains the address translation to where the actual data is located (also in physical memory).

```
2
```

We access the page table to get the physical page number corresponding to our virtual address, then access our memory address in physical memory. That makes it 2 accesses to the physical memory.

Q4. In order to make things more efficient, we decide to use a two-level page table instead, with the **virtual page number bits split evenly** between the two layers. We still use a page table entry size of 32 bits (4 bytes).

Q4.1: How many page table entries are in the L1 page table?
VPN1 indexes the L1 page table. How many bits do we have for VPN1? How many entries would that translate to?

```
1024
```

We originally had 20 bits for VPN. Now, we are dividing it into two for a two-level page table. Thus, VPN1 = VPN2 = 10 bits each. VPN1 indexes the L1 page table. If there are 10 bits for VPN1, then there are 2^10 = 1024 entries for the L1 page table.

Q4.2: How many pages does our L1 page table use?
Given the size of each page table entry and the computed number of entries for L1, how many bytes does the L1 page table consume? Given the page size, how many pages would the L1 page table consume?

```
1
```

The L1 page table has 1024 entries (solved earlier). Each page table entry is 4 bytes (given). Thus, the L1 page table is 1024*4 = 4096 bytes (4 KiB). Since the page size is 4 KiB (given), then we need 4 KiB / 4 KiB = 1 page to contain the L1 page table.

Q4.3: How many page table entries are in each L2 page table?
VPN2 indexes the L2 page table. How many bits do we have for VPN2? How many entries would that translate to?

```
1024
```

We originally had 20 bits for VPN. Now, we are dividing it into two for a two-level page table. Thus, VPN1 = VPN2 = 10 bits each. VPN2 indexes the L2 page table. If there are 10 bits for VPN2, then there are 2^10 = 1024 entries for the L2 page table.

Q4.4: How many pages does each L2 page table use?
Given the size of each page table entry and the computed number of entries for L2, how many bytes does the L2 page table consume? Given the page size, how many pages would the L2 page table consume?

```
1
```

The L2 page table has 1024 entries (solved earlier). Each page table entry is 4 bytes (given). Thus, the L2 page table is 1024*4 = 4096 bytes (4 KiB). Since the page size is 4 KiB (given), then we need 4 KiB / 4 KiB = 1 page to contain the L2 page table. As we have observed, each page table (both L1 and L2) takes up exactly 1 page. This is not by accident; the 10-10-12 page table is considered to be a "magic number" which is often used in 32-bit systems.

Q4.5: With a two-level page table, how many physical memory accesses are required per virtual memory access?
The L1 page table will point you to the L2 page table. The L2 page table will point you to the actual translation. Both of the page tables are stored in physical memory.

3

We need to access the L1 table (in main memory) to find the L2 table (also in main memory), which is then used to access the main memory again for the actual data. As such, it takes 3 memory accesses. Thus, a two-layer page table trades memory efficiency for time efficiency.

Submitted answer correct: 100%
Submitted at 2022-11-20 03:55:14 (PST)

hide

Q1. We want to set up a virtual memory system with **4 GiB of virtual memory**, mapped to **4 MiB of physical memory**.

Q1.1: How many bits long will our virtual addresses be?
Given the size of the virtual memory, how many bits do we need to address each byte of the virtual memory?

32 ✓100%

Q1.2: How many bits long will our physical addresses be?
Given the size of the physical memory, how many bits do we need to address each byte of the physical memory?

22 ✓100%

Q2. In order to set this up, we decide to use a **page size of 4 KiB** (4096 bytes), with a single layer page table.

Q2.1: How many bits long is the page offset?
The page offset addresses each byte of the page. Given the page size, how many bits do we need to address each byte of the page?

12 ✓100%

Q2.2: How many bits long is the virtual page number (VPN)?
The virtual address is partitioned into the virtual page number and the page offset.

20 ✓100%

Q2.3: How many bits long is the physical page number (PPN)?
The physical address is partitioned into the physical page number and the page offset.

10 ✓100%

Q3. We decide to include metadata and padding for each page table entry such that **each entry in our page table is exactly 32 bits (4 bytes) long**. For the purposes of this class, we assume that our page table is structured like an array of entries, with a slot for each virtual address.

Q3.1: How many page table entries are required for the full page table?
The VPN is the index of the page table. Given the VPN bits computed earlier, how many entries would the page table have?

`1048576` ✓ 100%

Q3.2: How many bytes of data does the page table take?
Knowing the number of page table entries and the number of bytes for each page table entry, how many bytes is the whole page table?

`4194304` ✓ 100%

Q3.3: How many pages of data does the page table take?
Given the size of each page and the computed page table size, how many pages would the page table consume?

`1024` ✓ 100%

Q3.4: How many physical pages can we use for actual data, if we decide to use this page table?
Given the size allocated for page tables (page tables are also stored in the physical memory) and the actual size of physical memory, how much space do we have left for the actual data?

`0` ✓ 100%

Q3.5: With a one-level page table, how many physical memory accesses are required per virtual memory access?
The page table (stored in physical memory) contains the address translation to where the actual data is located (also in physical memory).

`2` ✓ 100%

Q4. In order to make things more efficient, we decide to use a two-level page table instead, with the **virtual page number bits split evenly** between the two layers. We still use a page table entry size of 32 bits (4 bytes).

Q4.1: How many page table entries are in the L1 page table?
VPN1 indexes the L1 page table. How many bits do we have for VPN1? How many entries would that translate to?

`1024` ✓ 100%

Q4.2: How many pages does our L1 page table use?
Given the size of each page table entry and the computed number of entries for L1, how many bytes does the L1 page table consume? Given the page size, how many pages would the L1 page table consume?

`1` ✓ 100%

Q4.3: How many page table entries are in each L2 page table?
VPN2 indexes the L2 page table. How many bits do we have for VPN2? How many entries would that translate to?

`1024` ✓ 100%

Q4.4: How many pages does each L2 page table use?
Given the size of each page table entry and the computed number of entries for L2, how many bytes does the L2 page table consume? Given the page size, how many pages would the L2 page table consume?

`1` ✓ 100%

Q4.5: With a two-level page table, how many physical memory accesses are required per virtual memory access?
The L1 page table will point you to the L2 page table. The L2 page table will point you to the actual translation. Both of the page tables are stored in physical memory.

`3` ✓ 100%