CSSE 332 -- Operating Systems

Paging

Name: Solution Key

Question 1. (5 points) What is the main feature of paging that allows us to avoid external fragmentation?

> **Solution:** If paging is employed, the entire memory (both physical and virtual) is divided into *equal sized* chunks, called pages. We no longer need to make any **contiguity** assumptions about any memory area in the process's address space. In fact, the space's segments (code, globals, heap, stack) are no longer relevant in the physical space.

Question 2. (5 points) In a paged memory architecture, memory is divided into equal sized chunks. In virtual space, those chunks are called **pages**, while in physical space, they are called **frames**. To make address translation possible, each process needs to maintain a **page table**, which is an array of **page table entries (PTEs)**.

Question 3. The following questions assume a 16-bit architecture, which means that all of our addresses are 16 bits wide.

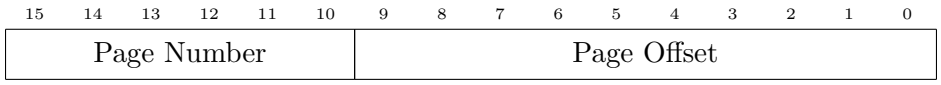
(a) (5 points) If we would like to design our paging system to have **1KB** pages. How many bits should we reserve from the 16-bits address for the page offset?

> **Solution:** 1KB page means we need 10 bits to offset into it.

(b) (5 points) In that case, how many pages can each process have?

> **Solution:** We are left with 6 bits from the address for the page number, which means each process have $2^6 = 64$ pages.

(c) (5 points) In this bit-box below, show how the page number and the page offset are represented.



(d) (5 points) Write down the formula used to translate a virtual address (`VA`) into a physical address (`PA`).

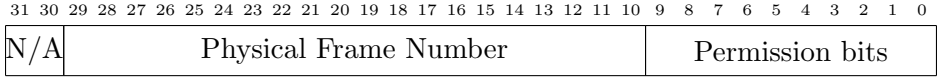
Solution:

$$\texttt{PA} = \texttt{PageTable}\left[\underbrace{\texttt{VA}[15:10]}_{\text{Page Number}}\right] + \underbrace{\texttt{VA}[9:0]}_{\text{Page Offset}}$$

(e) (5 points) Assume that each PTE is 16 bits, what is the size of a page table (in bytes)?

Solution: Each process has a 6-bits page number, which means each page table must have $2^6 = 64$ page table entries (PTEs). If each PTE is 16 bits wide (2 bytes), in total, the page table must be $2^6 \times 2 = 2^7 = 128$ bytes in size.

Question 4. (5 points) Assume now that we are working with a 32-bits RISC-V architecture. Describe the content of a page table entry (PTE) in the bit box below.



Question 5. (5 points) Describe the meaning of the permission bits in RISC-V below.

1. `PTE_V`: **Indicate if the PTE is valid or not.**
2. `PTE_W`: **Indicate if the PTE is writable or not.**
3. `PTE_R`: **Indicate if the PTE is readable or not.**
4. `PTE_X`: **Indicate if the PTE is executable or not.**
5. `PTE_U`: **Indicate if the PTE is accessible by the user or not.**

Question 6. Answer the following question about the `xv6` operating system.

(a) (5 points) Given a process represented by a `struct proc` pointer `p`. You can use `p->pagetable` to access `p`'s page table in the kernel.

(b) (5 points) When in kernel space executing on behalf of a user process (e.g., on a system call), you can use the `myproc()` function to access the current running **user** process.

(c) (5 points) The function `walkaddr` can be used to fetch a physical frame number given a virtual address. The lower **12** bits of the returned physical frame number will always be **zeros**.

Question 7. (5 points) Describe some of the shortcomings of the discussed paging approach.

Solution:

- Each memory access now requires two memory accesses, one to go to the page table and another to go to fetch the data.
- Page tables can grow quite large. We need one for every process, which means we will lose a lot of our main memory for paging overhead.
- There's one more, think about it for possible **engagement points**.