dsp187 - Dharmik Patel dc1533 - Dhruv Chaudhry ilab comp: ice.cs.rutgers.edu

1. **Detailed logic of how you implemented each virtual memory function.**
    a. **set_physical_mem():**
        i. Allocates and initializes memory using malloc
            1. Sets up page table and bitmaps data structures
                a. Interprets the physical memory as an array of unsigned ints and each entry represents an entry in the level 1 page table
                b. Uses log2 to compute the number of bits required to represent the max memory size.
            2. Calculates the number of bits required for various components of a virtual address based on the maximum memory size and page size
                a. Calculates the number of bits required to represent a page size, virtual page number, index into a level 1&2 page table in binary
                b. Calculates the number of virtual and physical pages based on total memory, page sizes, and bits.
                c. Initializes the bit maps for tracking allocation status of physical and virtual pages.
                d. Initializes the bit maps for tracking allocation status of physical and virtual memory.
            3. Initializes TLB entries
    b. **translate():**
        i. Translates a virtual address to a physical address
            1. Extracts indices for L1 and L2 tables, offset within the page, and virtual page number from the virtual address
            2. Checks TLB for a translation. If found, returns the corresponding physical address.
            3. If not found in TLB, walks through the page table:
                a. Finds the L1 entry and then the L2 entry
                b. Updates TLB with the translation
            4. Calculates physical address and returns it
    c. **page_map():**
        i. Maps a virtual page to a physical page
            1. Determines L1 and L2 indices based on the virtual addresses
            2. Checks if an entry exists in the L1 table, if not, allocates a new page for the L2 table.
            3. Checks if an entry exists in the L2 table, if not, allocates a new physical page
            4. Returns the physical page number
    d. **t_malloc():**
        i. Allocates virtual memory for a given size
            1. Calculates the number of pages needed based on size
            2. Finds consecutive free pages in the virtual memory bitmap
            3. Maps each virtual page to a physical page
            4. Returns the virtual address of the allocated memory
    e. **t_free():**
        i. Frees previously allocated virtual memory
            1. Calculates the number of pages needed based on size.

        2. Unmaps each virtual page from its corresponding physical page
        3. Marks virtual pages as a free from its corresponding physical page
        4. Marks physical pages as free in the physical memory bitmap
    f. **put_value():**
        i. Writes data to virtual address in memory
            1. The function uses the translate function to find the physical address of a virtual address and then copies it to a physical memory function
            2. Function returns 0 if completed or -1 if errors occur
    g. **get_value():**
        i. This function reads data from a specific virtual address in memory
            1. Uses translate function to find the physical address of a virtual address and then copies it from a physical memory location to a virtual memory location
            2. Function returns 0 if completed or -1 if errors occur
    h. **mat_mult():**
        i. Multiplies two matrices using the provided virtual addresses
            1. Utilizes **put_value()** and **get_value()** to access memory for matrix elements
            2. Computes the result and stores it back into memory
    i. **add_TLB():**
        i. Adds a translation entry or updates an existing entry in the Translation Lookaside Buffer
            1. Calculates the index in the TLB based on virtual page number and updates the TLB entry at the calculated index with the calculated page numbers
            2. If the entry at a specific address already exists, it is overwritten with the new translation
    j. **check_TLB():**
        i. Checks the TLB for a given virtual page number and determines the occurrence of a translation hit
            1. Calculates the index in the TLB based on virtual page number and check the spot for an entry
            2. If an entry exists, it indicates a translation hit else indicates a translation miss

2. **Support for different page sizes (in multiples of 8K).**
    a. The code supports different page sizes through the following mechanisms
        i. The code calculates the number of offset bits based on page size to ensure is can address each individual byte within a page
        ii. The code calculates number of bits required for indexing a page at different levels (numOfLevelBits and numOfLevel2Bits) to consider the required address bits for addressing different level of page tables
        iii. The code uses bitmasks to isolate the bits corresponding to different components of the virtual address, such as the page number, offset within the page, and indices for the different levels of page tables.
        iv. The **translate()** function uses the calculated offset bits and page table indices to translate virtual address into physical address. Furthermore, it

uses the page table entries to map virtual pages to physical pages based on the hierarchical structure of page tables. By correctly calculating the indices and using the appropriate bitmasks, the translation function ensures that it can handle different page sizes in the multiples of 8K

3. **Possible issues in your code (if any).**
    a. None
4. **If you implement the CS518 part, description of the 4-level page table design and support for different page sizes.**
    a. NA
5. **Collaboration and References: State clearly all people and external resources (including on the Internet) that you consulted. What was the nature of your collaboration or usage of these resources?**
    a. NA