Idea 1: memory paging that allows several page sizes(powers of 2) so that fits well with the buddy free space allocation.(image merge two pages of size 2 into a single page of size 4)

Idea 2: combine segmentation and paging (see textbook). Divide the address space into 3 subspaces: Code, Heap, and Stack.(save space)

Idea 3: Multi-level TLB. L1 and L2.

Idea 4: TLB buffer (not sure). Buffer the TLB of the most frequently running process somewhere near MMU.

Idea 5: Using idea 2, we now know to which subspace a memory address belongs. For code memory, we can load those addresses to L1 TLB. Load all other addresses to L2 first. If we hit an address in L2, we move that address to L1. Inside L1, we “sort” entries by the time we access it and deque the unneeded entries to L2 after a period of time.(why: some code and variable are used once vs. over and over again)

When requesting a new free physical memory block, say 8. Let’s assume page size = 2. So, this block should be divided into 4 pages. But we can utilize the fact that these 4 pages have continuous physical addresses to speed up. We can have two “twin bit” in page table entries: prev_twin : indicating vpn-1 is a twin of this vpn. next_twin: indicating vpn+1 is a twin. We can easily add a new TLB entry for a twin without looking up the page table.