CS152/252A
Fall 2026

Virtual Memory

Discussion 5

1 Quick Reference

- **Virtual memory:** lets multiple processes run location-independently and protected from each other, and uses main memory as a cache for disk.
- **Segments (base + bound):** addresses are relative to a base register and checked against a bound register; whole segments are evicted to disk.
- **External fragmentation:** free space is not contiguous (holes between segments), so a job may not fit even when enough total memory is free.
- **Page table:** maps each fixed-size virtual page (VA = page number | offset) to a physical page, so contiguous virtual memory can use non-contiguous physical pages.
- **Internal fragmentation:** a process's size is not a multiple of the page size, so part of its last page is unused.
- **Page fault:** the page table walk finds no valid PTE (page swapped out, not yet committed, copy-on-write, etc.). Usually not an error; must be a restartable exception.
- **Hierarchical page table:** the VPN is split into per-level indices (Sv32: 10 | 10 | 12-bit offset); the root is in `satp`. A **page table walk** is one sequential memory access per level.
- **TLB:** small, typically fully associative cache of PTEs (V/R/W/D bits, VPN tag, PPN). A TLB miss triggers a page table walk; it is not the same as a page fault.

2 CPI Equation Correction

The CPI equation given in Discussion 2, (total cycles after first commit) / (# committed instructions − 1), is **not** the definition used in this class. Per Lecture 3 (Pipelining), CPI includes the first instruction's commit cycle:

CPI = (last commit cycle − first commit cycle +1) / (# committed instructions)

The Discussion 2 version starts counting the cycle *after* the first commit and therefore also drops the first instruction from the count; the class definition counts from the first commit cycle inclusive and divides by all N committed instructions. Both give 1 for a no-stall sequence, but they differ on when to start counting the CPI. (See Ed #149.)

Below is the completed interlocked pipeline diagram from Discussion 2 for the linked-list traversal program (write-through register file, branches always predicted taken and squashed if wrong, single-cycle memory, 2 loop iterations).

```
       ADDI  x2, x0, 0x700
LOOP:  ADD   x1, x2, x0
       LW    x2, 4(x2)
       BNE   x2, x0, LOOP
       SW    x3, 4(x1)
```

Corrected CPI walkthrough: interlock pipeline (given)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
addi	F	D	X	M	W																									
(1) add		F	D	D	D	X	M	W																						
lw			F	F	F	D	X	M	W																					
bne						F	D	D	D	X	M	W																		
(2) add							F	F	F	D	X	M	W																	
lw										F	D	X	M	W																
bne											F	D	D	D	X	M	W													
(3) add												F	F	F	D	-														
lw															F	-														
sw																F	D	X	M	W										

- How many bubbles were needed?

- What is the CPI for this program with interlocking? (CPI is measured from when the first instruction commits to when the last instruction in the sequence commits.)

- What is the CPI assuming branches are always predicted correctly?

- Follow-up: why is the `lw` held in the fetch stage and not the decode stage, as prescribed by our definition of interlocking?

3 Page Size Trade-Offs

3.1 What are the benefits of a larger page size? What are the benefits of a smaller page size?

Larger:

Smaller:

4 Linear vs Hierarchical Page Tables

4.1 Consider 4 GiB (32-bit) of addressable virtual memory, 4 KiB pages, and 4-byte PTEs.

Bits in the page offset:

Bits in the virtual page number:

Number of pages:

4.2 Consider a linear page table for a process with only 1 page mapped to physical memory (paged in).

Number of valid page table entries (PTEs):

Total size of the page table:

4.3 Consider a 2-level page table for a process with only 1 page mapped to physical memory (paged in). Assume that VPN bits are split equally between the two levels.

Number of valid page table entries (PTEs):

Total size of the page table structure:

5 Hierarchical Page Table Example

Assume: 8-bit virtual addresses, 32-bit words, 32-bit PTEs, 16-byte pages, a two-level page table (VPN bits split equally between levels), and an LRU 4-entry TLB.

At the beginning, the TLB is empty and the free pages list contains 0x9, 0x5, 0xA, 0x7, 0x1, 0x3, 0xB, 0xD, 0xE, and 0xF in that order. PTBR is set to 0.

5.1 How many bytes of virtual memory are addressable?

5.2 How many bytes of physical memory are addressable?

5.3 Why might DRAM size > virtual address space size be useful?

5.4 Divide the virtual address into:

Offset bits:

Second-level index bits:

Top-level index bits:

5.5 How many entries are in the:

Top-level table?

Second-level tables (each)?

6 Page Table Walk Example

Same setup as above. Fill in the table for the following sequence of accesses, updating the page table in memory, the TLB, and the free pages list as you go.

6.1

Virtual Address	Index1	Index2	TLB hit/ miss	Page hit / page fault	Physical Address
0x68					
0x14					
0x6C					
0x90					
0x74					
0xE4					
0x18					
0xD0					

TLB (final contents)

VPN				
PPN				

Free pages: 0x9, 0x5, 0xA, 0x7, 0x1, 0x3, 0xB, 0xD, 0xE, 0xF
Free pages (after):

Addr	Contents
0x00	0x06
0x04	0x04
0x08	0x02
0x0C	
0x10	
0x14	
0x18	
0x1C	
0x20	0x08
0x24	
0x28	
0x2C	
0x30	
0x34	
0x38	
0x3C	
0x40	
0x44	
0x48	0x12
0x4C	0x11
0x50	
0x54	
0x58	
0x5C	
0x60	
0x64	0x13
0x68	
0x6C	