CSM 61C Spring 2020

Virtual Memory

Mentoring 9: April 15, 2020

1 VM Warm Up

1.1 The system in question has 1MiB of physical memory, 32-bit virtual addresses, and 256 physical pages. The memory management system uses a fully associative TLB with 128 entries and an LRU replacement scheme.

(a) What is the size of the physical pages in bytes?

(b) What is the size of the virtual pages in bytes?

(c) What is the maximum number of virtual pages a process can use?

(d) What is the minimum number of bits required for the page table base address register?

(e) Answer True or False to the following questions:
The page table is stored in main memory.
Every virtual page is mapped to a physical page.
The TLB is checked before the page table.
The penalty for a page fault is about the same as the penalty for a cache miss.
A linear page table takes up more memory as the process uses more memory.

2 Virtual Potpourri

2.1 For the following questions, assume the following:

- 16-bit virtual addresses
- 4 KiB page size
- 16 KiB of physical memory with LRU page replacement policy
- Fully Associative TLB with 4 entries and an LRU replacement policy

(a) What is the maximum number of virtual pages per process?

(b) How many bits wide is the page table base register?

2.2 For the following parts, assume that:

- Only the code and two arrays take up memory
- The arrays are both page-aligned (starts on page boundary)
- The arrays are the same size and do not overlap
- ALL of the code fits in a single page and this is the only process running

```
void scale_n_copy (int32_t *base, int32_t *copy, uint32_t num_entries, int32_t scalar) {
    for (uint32_t i = 0; i < num_entries; i++)
        copy[i] = scalar * base[i];
}
```

(a) If scale_n_copy were called on an array with N entries, where N is a multiple of the page size, how many page faults can occur in the worst-case scenario?

(b) In the best-case scenario, how many iterations of the loop can occur before a TLB miss?

3 Context Switch!

3.1 In this question, you will be analyzing the virtual memory system of a single-processor, single-core computer with 4 KiB pages, 1 MiB virtual address space and 1 GiB physical address space. The computer has a single TLB that can store 4 entries. You may assume that the TLB is fully associative with an LRU replacement policy, and each TLB entry is as depicted below.

TLB Entry

Valid Bit	Permission Bits	LRU Bits	Virtual Page Number	Physical Page Number

(a) Given a virtual address, how many bits are used for the Virtual Page Number and Offset?

(b) Given a physical address, how many bits are used for the Physical Page Number and Offset?

3.2 For the next 2 parts, consider that we are running the following code, in parallel, from two distinct processes whose virtual memory specifications are the same as that of above. Both arrays are located at page aligned addresses. As a note, $65536 = 2^{16}$.

Process 0	Process 1

int a[65536];
for (int i = 0; i < 65536; i += 256) {
  a[i] = i;
  a[i + 64] = i + 64;
  a[i + 128] = i + 128;
  a[i + 192] = i + 192;
}
	int b[65536]
	for (int j = 0; j < 65536; j += 256) {
	  int x = j + 256;
	  b[x - 1] = j;
	  b[x - 2] = j+1;
	  b[x - 3] = j+2;
	  b[x - 4] = j+3;
	}

As our computer has only a single processor, the processes must share time on the CPU. Thus, for each iteration of the processes' respective for loop, the execution on this single processor follows the diagram at the top of the next page. A blank slot for a process means that it is not currently executing on the CPU.

Time	Process 0	Process 1
0	a[i] = i;	
1	a[i+64] = i + 64;	
2		int x = j + 256;
3		b[x-1] = j;
4		b[x-2] = j + 1;
5	a[i+128] = i + 128;	
6	a[i+192] = i + 192;	
7		b[x-3] = j + 2;
8		b[x-4] = j + 3;

(a) What is the TLB hit rate for executing the above code assuming that the TLB starts out cold (i.e. all entries are invalid)? Only consider accesses to data and ignore any effects of fetching instructions. You may assume that the variables i, j and x are stored in registers and therefore do not require memory accesses. Remember: you must flush the TLB on a context switch from one process to another!

3.3 As opposed to the TLB architecture described above, let us consider a tagged TLB. In a tagged TLB, each entry additionally contains the Address Space ID (ASID), which uniquely identifies the virtual address space of each process. A tagged TLB entry is shown below.

Tagged TLB Entry

Valid Bit	Permission Bits	LRU Bits	ASID	VPN	PPN

On a lookup, we consider a hit to be if the (VPN, ASID) pair is present in the tagged TLB. This redesign allows us to keep entries in the TLB even if they are not a part of the process running on the CPU, so we do not have to flush the TLB when switching between processes.

Consider that we are using a tagged TLB and running the code in the manner described above.

(a) What is the hit rate for the tagged TLB for time 0-8 assuming it again starts out cold? You may make the same assumptions about the variables i, j, x and ignore the effects of fetching instructions.

(b) What is the hit rate for the tagged TLB for all iterations assuming it again starts out cold? You may make the same assumptions about the variables i, j, x and ignore the effects of fetching instructions.

(c) What is the smallest number of entries the TLB can have to still have the hit rate found in part 4?

4 VM Copy

4.1 For the following questions, assume the following (IEC prefixes are on the green sheet):

- You can ignore any accesses to instruction memory (code)
- 16 EiB virtual address space per process
- 256 MiB page size
- 4 GiB of physical address space
- Fully Associative TLB with 5 entries and an LRU replacement policy
- All arrays of doubles are page-aligned (start on a page boundary) and do not overlap
- All arrays are of a size equivalent to some nonzero integer multiple of 256 MiB
- All structs are tightly packed (field are stored contiguously)
- All accesses to structs and arrays go out to caches/memory (there is no optimization by reusing values loaded into registers)

```
void dblCpy(doubleFun* measurer, double* dblsToCpy) {
    measurer->fun = 0;
    for (uint32_t i = 0; i < ARRAY_SIZE; i += 4) {
        measurer->dbl[i] += dblsToCpy[i];
        measurer->fun += dblsToCpy[i];
    }
    measurer->fun /= ARRAY_SIZE;
}
```

(a) How many virtual page number bits are there and virtual address offset bits are there?

(b) How many physical page number bits are there and physical address offset bits are there?

4.2 (a) Assume the TLB has just been flushed. What TLB hit to miss ratio would be encountered if sizeof(double) * ARRAY_SIZE = 256 MiB and we run the above code?

(b) In the best-case scenario, how many iterations can be executed with no TLB misses? Use IEC prefixes when reporting your answer.