CS 356 Virtual Memory Exercises
Redekopp

Name: ____Solutions________________________________
Due: **Score:** ________

1.) Given a virtual memory system with 32-bit virtual addresses, 16 KB pages, and 36-bit physical addresses answer the following questions:

Define lg x = $\log_2 x$

16KB pages => lg 16KB = 14-bit page offset
32-bit VA – 14-bits = 18 VPN bits => 2^{18} pages = 256K pages
36-bit PA – 14-bits = 22 PPF bits => 2^{22} pages = 4 M pages

a. Given a single level page table, how much memory would be required to hold the table assuming each entry in the table requires 4 bytes (this includes the page frame, valid, dirty and other bits).

Page tables are indexed on virtual address so there will be 2^{18} entries each of 4-bytes yielding 2^{20} bytes for the page table = 1 MB

b. Given a three level page table where the 1st level has 32 entries, the 2nd level has 64 entries and the third level contains the rest of the needed entries, show the address bit field breakdown (which bits are used for levels 1, 2, and 3 page tables and which bits are used as the page offset.

1st level has 32 entries and thus requires 5 bits (1st level gets the MS address bits)
2nd level has 64 entries and thus requires 6 bits (next 6 bits after 1st level bits)
3rd level requires 18 total VPN bits – 5 for 1st level – 6 for 2nd level = 7 bits for 3rd level

VPN			Page Offset
1st Level Index	2nd Level Index	3rd Level Index	Page Offset
A31-A27	A26-A21	A20-A14	A13-A0

c. Assuming 4 byte entries in each level of page table, what is the worst case memory usage (in bytes) required for the 3 level page table system described in the previous part if 10 virtual pages are in use.

Worst case scenario is that all 10 pages require different 1st level entries and thus 10 second level tables and 10 third level tables

= 32*4 bytes for 1st level table + 10*64*4 for 2nd level tables + 10*128*4 for 3rd level tables = 128 + 2560 + 5120 = 7808 bytes

d. Assume a 4-way set associative TLB with 128 total entries. Show the mapping (fields) of the virtual address for accessing the TLB. Include the tag, set and page offset fields.

128 entries / 4-ways = 32 sets => 5 set bits

VPN		Page Offset
Tag A31-A19	Set A18-A14	Page Offset A13-A0

For questions 2-4, assume a 2-way set associative D-TLB (data only, no code pages) with 64 entries and LRU replacement. Also assume a virtual memory system with 32-bit virtual addresses and 4 KB pages.

2.) If the D-TLB entries are initially all empty/invalid, how many unique pages could be referenced before a D-TLB entry ***may*** be evicted (replaced). Give your reasoning to receive credit.

Minimum number means fewest page access before something would have to be replaced. The minimum number will come when all accesses map to the same set. Since each set has 2 ways, then we may only be able to reference 2 pages safely before a 3rd page access would cause an eviction..

3.) Now assume a program is run and at a certain point in time all D-TLB entries contain valid translations. What is the maximum **amount of memory** (in bytes) that the program can access w/o causing a page fault/replacement.

There are 64 entries each pointing to 4KB pages. Thus we could address/access 4KB * 64 = 256KB of memory.

4.) Examine the following code operating on three **integer** (word) arrays A, B, and C. Assume i is allocated in a register as is the constant ARRAY_SIZE and neither requires accessing memory. Further assume the arrays are allocated contiguously (B starts after A's last element, etc.) and the right-hand side (RHS) of the assignment is evaluated from left-to-right (A[i] is accessed first, then B[i], etc.)

```
for(i=0; i < ARRAY_SIZE; i++){
   A[i] = A[i] + B[i] + C[i];
}
```

a. The worst-case scenario is that all three array translations map to the same set. What size would the arrays have to be (ARRAY_SIZE=?) so that an access to A[i], B[i], and C[i] require different translations but that the translations all map to the same D-TLB set. There are probably many sizes that would work, pick the smallest. [Remember that each array entry is an integer = word = 4-bytes.]

64 entries / 2-ways = 32 sets => 5 set bits

VPN		4 KB Page Offset
Tag	Set	Page Offset
A31-A17	A16-A12	A11-A0

The worst case will be when the arrays are aligned on boundaries that map to the same set. Because they are all contiguous, the smallest size will have to be equal to boundaries where the set bits and page offset bits are the same and the tag increments by 1. An example is shown below:

	VPN		4 KB Page Offset
	Tag	Set	Page Offset
A	0000 0000 0000 000	0 0000	0000 0000 0000
B	0000 0000 0000 001	0 0000	0000 0000 0000
C	0000 0000 0000 010	0 0000	0000 0000 0000

In this case the size of each array would be 2^{17} bytes=128KB (because it spans 17 address bits). Another way to see it is that since there are 32 sets we would have to go through all 32 sets before getting back to the same one. This would mean addressing 32 * 4KB = 128KB of memory.

Finally ARRAY_SIZE would then be 32768 = 32KB since we have 128KB / 4 bytes per int = 32768

b. After the 0th iteration completes which 2 of the three arrays will have translations in the TLB. [Hint: Keep in mind that the D-TLB is 2-way set-associative and uses LRU replacement.]

Order of Access:

4th	1st	2nd	3rd
`A[i] =`	`A[i]`	`B[i]`	`C[i]`

Since all accesses map to the same set and there are only 2-ways (entries per set) then only the last 2 translations will be available. Thus C (3rd) and A (4th) will have translations.

c. Given the situation after the 0th iteration, how many D-TLB (translation) misses will be incurred by the next iteration? [Hint: Take into account the evaluation order and what the last values accessed would have been from the previous iteration.]

Order of Access:

4th = [B,C] in TLB = MISS replacing B	1st [C,A in TLB] = HIT	2nd [C,A] in TLB = MISS replacing C	3rd = [A,B] in TLB = MISS replacing A
`A[i] =`	`A[i]`	`B[i]`	`C[i]`

= 3 Misses and 1 Hit

d. By simply rewriting/re-ordering the assignment statement, can you reduce the number of D-TLB misses? Hint: remember we said the RHS is evaluated from left to right and then assigned to the left hand side (LHS).

Define the syntax **(A,B,C,A)** represent the order of access where the first three are the RHS order and can be re-ordered and the last A is the LHS assignment which cannot change. There are thus 6 possible orderings (3!). We will show the Miss/Hit pattern corresponding to each access. These are arrived at by remembering the previous 2 arrays access will be the ones in the TLB set.

(A[i],B[i],C[i],=>A[i]) = original ordering = (H,M,M,M) = 3 Misses

(A[i],C[i],B[i], =>A[i]) = similar to first = (H,M,M,M) = 3 Misses

(B[i],A[i],C[i], =>A[i]) = (M,H,M,H) = 2 Misses = GOOD!

(B[i],C[i],A[i], =>A[i]) = (M,M,M,H) = 3 Misses

(C[i],A[i],B[i], =>A[i]) = (M,H,M,H) = 2 Misses = GOOD!

(C[i],B[i],A[i], =>A[i]) = (M,M,M,H) = 3 Misses

Rationale: A is accessed twice each iteration…so we really want to keep that in one of the ways of the set and let B and C swap. To do this the distance between A accesses must be 2 (i.e. only 1 other access between an access to A)