EE 352 Homework 5
Spring 2010 Nazarian

Name: ________________________ **Score:______**

Assigned: Saturday April 17

Due: Friday, April 23 at 3:00pm (slip your papers under the door of EEB340)

NOTE: I WILL POST THE SOLUTIONS RIGHT AFTER THE DEADLINE. NO LATE SUBMISSION IS ACCEPTED.

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

- **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).

- **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.

- **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.

- **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.

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, what is the minimum number of pages that could be referenced before a D-TLB entry must be evicted (replaced). Give your reasoning to receive credit.

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.

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.]

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.]

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.]

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).

5-Stage Pipeline Scheduling

5) Show the time-space diagram for the following code sequence on our simple 5-stage pipeline assuming full forwarding, early branch determination (in the DECODE stage), no delay slots, and predict NT (i.e. just keep fetching down the sequential path). Use X's to indicate an instruction that is flushed/deleted/turned into a bubble.

a) (7 pts.)

```
lw          $8,0($3)
add         $3,$4,$2
sub         $5,$3,$8
lw          $6,20($3)
add         $7,$3,$6
sw          $7,0($3)
```

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
lw $8,0($3)															
Add $3,$4,$2															
Sub $5,$3,$8															
lw $6,20($3)															
Add $7,$3,$6															
sw $7,0($3)															

b) 7 pts.

```
        add      $5,$7,$2
        lw       $6,0($5)
        sub      $8,$6,$3
        bne      $6,$0,L1  (Taken)
        xor      $5,$8,$10
L1:     sw       $6,0($2)
```

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
`add $5,$7,$2`															
`lw $6,0($5)`															
`sub $8,$6,$3`															
`bne $6,$0,L1`															
`xor $5,$8,$10`															
`sw $6,0($2)`															

6) (7 pts.) Now assume 1-delay slot (1 instruction after a branch will always be executed). Using the same code as part 1b and assuming the 'bne' is **taken**, re-order the code to fill the branch delay slot in such a way as to maximize the CPI. Use the time-space diagram below filling in entries for only the first 6 instructions that will be executed. Hint: After re-arranging the code for the branch delay slot, remember that we've added forwarding paths from other stages back to the decode stage to satisfy the case when a branch depends on another instruction.

```
        add      $5,$7,$2
        lw       $6,0($5)
        sub      $8,$6,$3
        bne      $6,$0,L1
        xor      $5,$8,$10
L1:     sw       $6,0($2)
        lw       $12,0($3)
```

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
`add $5,$7,$2`															
`lw $6,0($5)`															
`bne $6,$0,L1`															
`sub $8,$6,$3`															
`sw $6,0($2)`															
`lw $12,0($3)`															