Exam 2 Practice Problems

Note: *The actual exam might have very different problems/questions and might be of a different length. The following problems are intended for you to practice in addition to studying what is in the slides and practicing with the examples in them.*

Problem 1:

Consider a machine with the following specifications:

- Bus width: 8 bits
- DRAM size: 64 B
- Page size: 4 B
- PTE size: 1 B

A. How many address bits are used for page offset?
B. What is the PT size for each process?
C. What is the higher-level PT size for each process if we add one level to the PT hierarchy?

Problem 2:

Consider a 64-bit machine with a 16GB main memory and 64KB pages.

A. How many bits is the PPN (ignoring the most significant bits that are always 0)?
B. What is the PT size for each process?
C. What is the higher-level PT size for each process if we add one level to the PT hierarchy?
D. What is the higher-level PT size for each process if we add another level to the PT hierarchy?
E. What is the higher-level PT size for each process if we add another level to the PT hierarchy?
F. What is the main downside of adding multiple levels of PTs?

Problem 3:

Consider the following cache system:

- 24B cache
- 4B blocks
- 3-way set-associative

Draw the cache access paper simulation table for accessing the following consecutive addresses: C, 7, 8, 2, 9, 2, 8, 4, 8, 5. Use LRU algorithm for block replacement. All blocks in the cache start as invalid.

Problem 3 variations:

A. *Try a variation of Problem 3 where addresses are accessed in a reversed order*
B. *Try a variation of Problem 3 with half the cache size*
C. *Try a variation of Problem 3 with 8B blocks instead*
D. *Try a variation of Problem 3 with a 2-way set-associative cache instead*

Problem 4:

Consider the following MIPS assembly code of a function that multiplies the contents of two 3x3 matrices. Draw the pipeline diagram for its execution on a 5-stage pipelined MIPS processor with full bypassing, flush, and stall logic.

```
# Assuming $1 holds the start address of M1,
# $2 holds the start address of M2,
# and $3 holds the start address of the resulting matrix

        li $4, 0 # $4 holds i
        li $5, 0 # $5 holds j
        li $6, 0 # $6 holds k
        li $7, 3
LOOP1:  bge $4, $7, END1  #while (i < 3) {
        li $5, 0          #j = 0;
LOOP2:  bge $5, $7, END2  #while (j < 3) {
        mul $8, $4, $7
        add $8, $8, $5    #memory offset = i*3+j
        add $8, $8, $3
        sw $0, 0($8)      #rslt[i][j] = 0;
        li $6, 0          #k = 0;
LOOP3:  bge $6, $7, END3  #while(k < 3) {
        mul $8, $4, $7
        add $8, $8, $6    #memory offset = i*3+k
        add $8, $8, $1    #M1[i][k]
        lw $8, 0($8)
        mul $9, $6, $7
        add $9, $9, $5    #memory offset = k*3+j
        add $9, $9, $2    #M2[k][j]
        lw $9, 0($9)
        mul $8, $8, $9    #M1[i][k] * M2[k][j]
        mul $9, $4, $7
        add $9, $9, $5    #memory offset = i*3+j
        add $9, $9, $3
        lw $10, 0($9)     #rslt[i][j]
        add $8, $8, $10   #rslt[i][j] + M1[i][k] * M2[k][j]
        sw $8, 0($9)      #rslt[i][j] += M1[i][k] * M2[k][j];
        addi $6, $6, 1    #k++;
        j LOOP3           #}
END3:   addi $5, $5, 1    #j++;
        j LOOP2           #}
END2:   addi $4, $4, 1    #i++;
        j LOOP1           #}
END1:
```

Problem 5:

Consider the following MIPS assembly code. Draw the pipeline diagram for its execution on a 5-stage pipelined MIPS processor with full bypassing, flush, and stall logic.

```
main:             addiu $t0, $0, 9
                  addiu $t1, $0, 1
                  subi $sp, $sp, 4
                  sw $t1, 0($sp)
WHILE_0_INIT:     addiu $t2, $0, 13
WHILE_0_COND:     bge $t0, $t2, WHILE_0_EXIT
                  sub $t1, $t2, $t1
                  ble $t1, $t0, IF_0_IF_TRUE
                  j IF_0_ELSE
IF_0_IF_TRUE:     addi $t2, $t2, 1
                  j IF_0_EXIT
IF_0_ELSE:        subi $t2, $t2, 1
IF_0_EXIT:        lw $t1, 0($sp)
                  addi $t1, $t1, 2
                  bgt $t0, $t1, IF_1_IF_TRUE
                  j IF_1_ELSE
IF_1_IF_TRUE:     subi $t2,$t2,2
                  addi $t1, $t1, 7
                  j IF_1_EXIT
IF_1_ELSE:        move $t2, $0
IF_1_EXIT:        sw $t1, 0($sp)
                  j WHILE_0_COND
WHILE_0_EXIT:     li $v0, 10
                  syscall
```

Problems 4 & 5 variations:

A. Try a variation of Problems 4 and 5 without bypassing
B. Try a variation of Problems 4 and 5 with some bypassing (instead of full bypassing)

Theoretical questions (answers found in slides):

A. Why do we need a memory hierarchy?
B. Why is it better to move data in blocks/pages, not bytes, for example?
C. How do we usually solve a functionality problem in computing?
D. Why do we need the virtual memory system on our personal computers?
E. What are the main similarities between interrupts, exceptions, and system calls?
F. What are the main differences between interrupts, exceptions, and system calls?
G. When do we need an OS?
H. What is the maximum file size provided by an inode?
I. What are two ways for the processor to communicate with IO devices?