CS3410 Spring 2014
Problem Set 2, due Saturday, April 19, 11:59 PM

NetID: Name:

200 points total. Start early!
Update March 27: Problem 2 updated, Problem 8 is now a study problem.

Problem 1 Data Hazards (25 points)

(a) Do exercise 4.13.1 (5th Edition Patterson & Hennessy). For part(a), no bypasses and the register file is read before it is written (so you cannot read a register written in the same cycle).

(b) Fill out the cycle chart for this instruction sequence. Resolve hazards by stalling. For part (b), the first two bypasses are not implemented, but the register file is written before it is read.

	Cycle																	
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

(c) Resolve hazards by stalling. For part (c), fully-bypassed (including the register file is written before it is read).

	Cycle																	
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Problem 2 Control Hazards (25 points)

Do exercise 4.14.2 (5th Edition Patterson & Hennessy). *Update March 27: use the following program instead of the program in the book.*

```
        lw r2, 0(r3)
        beq r1, r3, label3      ; taken once, then always not taken
label1: lw r4, 0(r2)
        beq r4, r2, label4      ; never taken
label4: lw r1, 0(r4)
label3: beq r1, r2, label1      ; taken once, then always not taken
label2: add r2, r3, r4
```

	Cycle																	
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Problem 3 Calling Conventions (25 points)

You are given the following function in C and know that the functions `bar()` and `baz()` exist.

```
int foo(int x) {
    int a = 0;
    int b = x;
    int c = b;
    int d;
    for(; a < c; a++) {
        d = bar(b,c);
        b = d+b;
    }
    a = baz(a,b,c,10,d);
    return a-1;
}
```

(a) How many caller/callee save registers will you use and for which variables? Fill in the following assembly code with your register assignments. You do not need to include registers used for intermediate calculations not done explicitly in C. Briefly justify your answer.

(b) Write the prologue and epilogue for the function `foo()`. All the stack space needed for the function must be allocated in the prologue, do not expand the size of the stack during the main section of the function.

(c) Fill in the function calls including passing arguments. Assume the address of the beginning of the function `bar()` is labeled `_bar` and the beginning of `baz()` is label `_baz`.

```
_foo:
; (b) (prologue)
```

```
    addiu ___, $zero, 0        ; Initialize a
    addiu ___, $a0, 0          ; Initialize b
    addiu ___, ___, 0          ; Initialize c
LOOP:
    slt $t1, ___, ___          ; a < c
    bnez  $t1,  EXIT
    nop
; (c)  (function call to bar())
; assign arguments to bar()
; call bar()



    addu  ___, ___, ___        ; b = d + b
    addiu ____, ____, 1        ; a++
    j  LOOP
    nop
EXIT:
; (c) (function call to baz())
; assign arguments to baz()
; call baz()



    addiu  ____, ____, -1      ; return a – 1
; (b) (epilogue)
```

Problem 4 Caches (25 Points)

Consider a fully associative cache with two 8-byte lines. Assume 32-bit virtual address, 32-bit physical address, write-back, write-allocate, and LRU eviction policies. The following program is run:

```
1:      LW $s0 <- M[0]
2:      LW $s1 <- M[4]
3:      LW $s2 <- M[12]
4:      SW $s1 -> M[8]
5:      LW $s3 <- M[0]
6:      SW $s0 -> M[28]
7:      SW $s1 -> M[24]
```

Address	Data
0	0x2A12D02
4	0x53A4F155
8	0x00BEEF00
12	0x80081355
16	0x11111111
20	0xFFFFFFFF
24	0xB1B13000
28	0x01010101

(a) Identify which bits (0-indexed) of a memory address are the tag, the block offset and the offset within a word.

Tag:
Block offset:
Offset within a word:

(b) How many cache misses and hits are there if cache is initially empty?

(c) After the execution of the code above, what is stored in the cache?

V	D	Tag	Data	

(d) How many memory reads are performed by the time line 7 is finished?

(e) Draw what is in the registers and memory afterwards.

Registers:

$s0	
$s1	
$s2	
$s3	

Memory:

Address	Data
0	
4	
8	
12	
16	
20	
24	
28	

(f) Explain the tradeoffs and differences associated with the following:

- Associative and direct mapped caches

- Physically-addressed and virtually-addressed caches

Problem 5 Virtual Memory (25 Points):

(a) Name three benefits of using virtual memory.

1).

2).

3).

(b) Consider a byte addressable virtual memory system with 38-bit virtual addresses, 32-bit physical addresses, and 16 KB pages. Is it possible to run a program that requires 16 GB of data on this system? Explain why or why not.

(c) If each entry in the single-level page table is 4 bytes long,

How much space does the page-table occupy in memory? ______________________

How much total physical memory would a 32MB process occupy? _______________

(d) Suppose now that we have a two-level page table (page table entries are still 4 bytes long).

How much total physical memory would a 32MB process occupy (including physical memory used for associated page tables)? ___________________________

Note: Assume 12 bits are used for the 1st level page table and 12-bits are used for the 2nd level page table.

Problem 6 Traps (15 points)

(a) System calls and exceptions trigger a similar set of actions but also contain specific steps that are syscall only or exceptions only. The below table has a set of actions. Mark whether the action occurs in Syscalls, Exceptions, or Neither. (If it occurs in both then mark both columns.)

Steps	Exceptions	Syscall	Neither
Switches the sp to the kernel stack			
Saves the old (user) FP value			
Saves the old (user) PC value (= return address)			
Saves the old privilege mode			
Saves cause			
Sets the new privilege mode to 1			
Sets the new PC to the kernel handler			
Saves callee-save registers			
Saves caller-save registers			
Examines the syscall number			
Examines the cause			
Checks arguments for sanity			
Allocate new registers			
Performs operation			
Stores result in v0			
Restores callee-save registers			
Restores caller-save registers			
Performs a return instruction, which restores the privilege mode, SP and PC			

(b) Why does syscall save and restore much less state than an exception handler?

Problem 7 Multicore Performance (15 points)

(a) Describe a possible scenario where multiple cores working on the same task might be slower than a single core of the same architecture and clock speed.

(b) Suppose we have a program of which 50% can be parallelized to any extent, 30% can be parallelized to up to 10 processors, and 20% cannot be parallelized at all. If the program takes 120 seconds to run on one processor, how long will it take to run on 2 processors working in parallel? On 10? On 20? What is the limit as the number of processors tends to infinity? What is the maximum speedup?

2 processors: ______________

10 processors: ______________

20 processors: ______________

N processors as N goes to infinity: ___________ Speedup: __________

Problem 8 Multicore Synchronization (Study Question, 0 points)

Update March 27: since HW2 is now due before the synchronization lecture this problem will not be graded. However, you should do it since synchronization may appear on Prelim 2.

Suppose we have two threads with the following instructions both in C and in MIPS assembly. The threads operate in parallel and use a shared memory space. Array c is set such that c[0] = 1, c[1] = 2, and c[2] = 3.

Thread A	Thread B

c[0] = c[0] + 2;
c[1] = c[1] + 1;
	c[1] = c[1] – 2;
	c[2] = 4;
	c[0] = 0;

MIPS Thread A	MIPS Thread B
LW $t0, 0($s0) ADDIU $t0, $t0, 2 SW $t0, 0($s0) LW $t0, 4($s0) ADDIU $t0, $t0, 1 SW $t0, 4($s0)	LW $t0, 4($s0) ADDIU $t0, $t0, -2 ADDIU $t1, $zero, 4 SW $t1, 8($s0) SW $zero, 0($s0)

(a) Assuming the instructions can be executed in any order, what are all the possible values of c[0], c[1], and c[2] after both threads finish?

c[0] may be
c[1] may be
c[2] may be

(b) Write the assembly code for the threads using the load linked and store conditional instructions. Assume the base address for c is in $s0.

Thread A	Thread B

Problem 9 C Hashtable (45 points)

Note there are three parts to this problem, (a), (b), and (c).

A hash table is an array of buckets. Each bucket may contain a list of data or be empty. Hash table data consists of a hashable key and an associated value. We call this the key/value pair. Let us define a hashtable with chaining and a load factor of 0.75.

Chaining: Let h = hash(key). The key/value pair is stored in a list stored at bucket (h mod N) where N is the table size. Hash collisions are handled by adding the key/value pair to the list stored at the bucket.

Load factor 0.75: load factor is the # of elements in the hashtable divided by the total # of buckets. The performance of chaining drops significantly when the load factor exceeds some threshold. To avoid this, the table is doubled in size when the load factor is > 0.75.

For this problem we will consider a hash table with integer keys and integer values.

(a) After N insertions how big can the table be (worst case space)? How many insert operations are needed (worst case time)? Give your answer as an estimate of the constant factors (e.g., 2N, 3N log N, etc) for both questions.

Space: ____________________

Time: ____________________

(b) Implement the hash table in C and submit the code and binary to CMS. Note: you may use your arraylist from Clab-2. The following functions are required:

```
void hashtable_create(struct hashtable *self);
void hashtable_put(struct hashtable *self, int key, int value);
int hashtable_get(struct hashtable *self, int key);
void hashtable_remove(struct hashtable *self, int key);
void hashtable_stats(struct hashtable *self);
```

Example usage:

```
struct hashtable a;
hashtable_create(&a);
hashtable_put(&a,0,99);
hashtable_stats(&a); /* prints: "length = 1, N = 2, puts = 1\n" */
hashtable_put(&a,1,42);
hashtable_stats(&a);
assert(hashtable_get(&a,0)==99);
hashtable_remove(&a,0);
hashtable_get(&a,0); /* causes program to exit with exit code 1 */
hashtable_remove(&a,0); /* would also cause program to exit with exit code 1 */
```

Output:

```
length = 1, N = 2, puts = 1
length = 2, N = 4, puts = 3
ERROR: key 0 not found.
```

Output explained: On the first insertion, the table is initialized with a size of 2. The second insertion will cause the load factor to exceed 0.75. So, prior to insertion the table is resized to N=4. Changing the table size requires that we re-insert all existing key/value pairs. (Why?) Therefore the total number of insertions is 3.

Error handling policy: your program must return 0 from main if there are no errors. Your program may choose to exit with exit code 1 if it detects a runtime error. Aborting the program with an assert is also acceptable.

Hash function: you may use any hash function or this provided hash function (modified djb2 from http://www.cse.yorku.ca/~oz/hash.html):

```
unsigned long hash (int* key) {
        unsigned long hash = 5381;
        char* str = (char*)key;
        int c;

        while ((c = *str++))
                hash = ((hash << 5) + hash) + c;

        return hash;
}
```

(c) Suppose we would like to handle complex keys and values (e.g. string keys and/or struct values). One idea is to use the same code but pass pointers as integers. For example,

```
char *key="hello";
struct mystuff value;
... /* initialize value */
hashtable_put((int)key,(int)(&value));
struct mystuff *p=(struct mystuff *)hashtable_get((int)"hello");
assert((*p)==value); /* is this true? */
```

Would this work? Explain in 1-2 sentences.