CS398 Exam 3, 2\textsuperscript{nd} Chance  
December 17th, 2012

Name:  
NETID:  

Circle the section that attend (so we can hand back your exam).

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
</tr>
</thead>
<tbody>
<tr>
<td>AYA (1-3pm) Craig</td>
<td>AYE (9-11am) Maria</td>
</tr>
<tr>
<td>AYB (2-4pm) Jon</td>
<td>AYF (10am-noon) Ting</td>
</tr>
<tr>
<td>AYC (3-5pm) Michael</td>
<td>AYG (11am-1pm) Ting</td>
</tr>
<tr>
<td>AYD (4-6pm) Ting</td>
<td>AYH (noon-2pm) Jon</td>
</tr>
<tr>
<td></td>
<td>AYI (1-3pm) Ryan</td>
</tr>
<tr>
<td></td>
<td>AYJ (2-4pm) Michael</td>
</tr>
<tr>
<td></td>
<td>AYK (3-5pm) Michael</td>
</tr>
<tr>
<td></td>
<td>AYL (4-6pm) Ryan</td>
</tr>
<tr>
<td></td>
<td>AYM (5-7pm) Ryan</td>
</tr>
</tbody>
</table>

- This exam has 7 pages; the final sheet is provided as a reference to you.  
- You have 120 minutes.  
- No calculators or other electronics are allowed. You may bring one 8.5” x 11” sheet of handwritten notes.  
- To make sure you receive credit, please write clearly and show your work.  
- We will not answer questions regarding course material.  

The 2\textsuperscript{nd} chance test is done at the granularity of the 3 questions; if you choose to have a question graded, we will grade ALL PARTS of that question and use that to update your score.

<table>
<thead>
<tr>
<th>Question</th>
<th>Maximum</th>
<th>Your Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Question 1: Pipelining (40 points)

Consider the 6-stage pipeline shown below; in this pipeline, the MEM stage occupies two pipeline stages (MEM1, MEM2) with loads completing in the MEM2 stage (arithmetic instructions complete in the EX stage as normal). Full bypassing is provided. Assume that all branches and jumps are predicted as not-taken. Conditional branches and indirect branches (e.g., \( \text{jr} \)) are resolved in the EX stage. Unconditional jumps are resolved in the ID stage. Assume \( \text{mul} \) is a single real instruction that is executed by the ALU and completes in the EX stage.

For all of this question consider the MIPS assembly code on the following page. Corresponding C code is shown below.

Part (a) Annotate the MIPS assembly to indicate all of the true data dependences. (5 points)

For the next two parts, use your scantron form. We recommend that you first answer on the next page and then copy your answers to the scantron form. In any case, we won’t give credit for any answers not on the scantron.

Part (b) Indicate which instructions will be stalled: a) no stall b) 1-cycle stall c) 2-cycle stall (5 points)

Part (c) Indicate how each of the forwarding muxes (forwA, forwB) will set in the cycles when each instruction is in the EX stage. Answer a-d as labeled in the diagram above. (10 point)

typedef struct pixel {
    int x, y, z;
    struct pixel *next;
} pixel_t;

void map (pixel_t *pixel, int scale) {
    while (pixel != NULL) {
        pixel->z = scale * pixel->x + pixel->y;
        pixel = pixel->next;
    }
}
map:       beq $a0, $0, done

loop:      lw  $t1, 0($a0)     stall    forwA   forwB
           mul  $t1, $t1, $a1  ___1   ___10  ___11
           lw   $t2, 4($a0)   ___2   ___12
           add  $t2, $t2, $t1  ___3   ___13  ___14
           sw   $t2, 8($a0)   ___4   ___15  ___16
           lw   $a0, 12($a0)  ___5   ___17
           bne  $a0, $0, loop  ___6   ___18  ___19

done:      jr   $ra

**Part (d)** Compute how many cycles each loop iteration takes on average. Explain your answer for partial credit. (10 points)

**Part (e)** Re-schedule/re-write the function to make it faster. Faster code will achieve more points, but your answer must fit in the space below. (10 points)
**Question 2: Cache Analysis (25 points)**

For an 8KB 2-way set-associative, write-back cache with 32B blocks on a machine with 32-bit address spaces (both virtual and physical) and no hardware prefetching, consider the following code:

```c
struct hoof { int has_horseshoe, shoe_size; };  
struct unicorn {  
    int horn_length;  
    char *name;  
    struct hoof *hooves[4]; // this is an array of pointers  
};

struct unicorn unicorns[1000]; // that's a whole lotta unicorns  
int longest_horn = 0, biggest_shoe = 0;

for (int i = 0 ; i < 1000 ; i ++) {  
    if (unicorns[i].horn_length >= longest_horn) {  
        longest_horn = unicorns[i].horn_length;
    }
    for (int j = 0 ; j < 4 ; j ++) {  
        if (unicorns[i].hooves[j]->has_horseshoe &&  
            (unicorns[i].hooves[j]->shoe_size >= biggest_shoe)) {  
            biggest_shoe = unicorns[i].hooves[j]->shoe_size;
        }
    }
}
```

*Assume that everything is in registers, except the data structure unicorns.*
Part (a) Compute the MINIMUM number of cache misses per outer-loop iteration that is possible for the code on the previous page. Explain how you computed it and the assumptions you made! (15 points)

Part (b) Compute the MAXIMUM number of cache misses per outer-loop iteration that is possible for the code on the previous page. Explain how you computed it and the assumptions you made! (10 points)
Rewrite the following codes to optimize cache performance on a system with hardware stream prefetching (i.e., if you are fetching sequentially no software prefetching is necessary) and a single-level cache. The cache is a 2-way set associative 16KB cache with 32B blocks. Software prefetch syntax, if you choose to use it, is shown on the right.

a) (10 points)

```c
#define N 5000

for (int i = 0 ; i < N ; i ++) {
    for (int j = 0 ; j < N ; j += 2) {
        A[i][j] = A[i][j+1];
        B[j][i] = B[j+1][i];
    }
}
```

```c
void __builtin_prefetch(const void *addr, unsigned rw, unsigned locality);

addr: the address of the memory to prefetch.
rw: (optional) a compile-time constant 1 or 0; 1: the program anticipates writing the data soon, 0 (default) in the near term, the program expects to only read the data.
locality: (optional) a compile-time constant from 0 to 3. 0: the data has no temporal locality, so need not be left in the cache after the access, 3: (default) the data has a high degree of temporal locality and should be retained in all levels of cache if possible. Use 0 or 3 based on the expected reuse.
```
Question 3: Cache-aware Programming (20 points), cont.

b) (10 points)

#define N 5000
double A[N][N][N], B[N][N], C[N][N];

for (int i = 0 ; i < N ; i ++) {
    for (int j = 0 ; j < N ; j ++) {
        double temp = 0.0;
        for (int k = 0 ; k < N ; k ++) {
            temp += B[i][0] * A[k][j][i];
        }
        C[i][j] = temp;
    }
}