# GRADING/ANSWER SHEET

## Fall 2010 CS61C Final

Your Name: ____________________

Your TA: Andrew Michael Conor Charles

Login: cs61c-____

This exam is worth 186 points, or about 20% of your total course grade. The exam contains 10 questions.

This booklet contains 14 numbered pages including the cover page, plus the 2 pages for the green card. Put all answers on these pages, please; don’t hand in stray pieces of paper.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points (Minutes)</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Justified</td>
<td>30 points (15 minutes)</td>
<td>20.5</td>
</tr>
<tr>
<td>2. Virtuosity</td>
<td>14 (7)</td>
<td>9.8</td>
</tr>
<tr>
<td>3. Mo’ Cache, Mo’ Problems</td>
<td>24 (12)</td>
<td>18.1</td>
</tr>
<tr>
<td>4. My Secret Cache</td>
<td>34 (17)</td>
<td>20.5</td>
</tr>
<tr>
<td>5. Heaven’s Gate</td>
<td>8 (4)</td>
<td>7.4</td>
</tr>
<tr>
<td>6. Off the Map</td>
<td>20 (10)</td>
<td>12.7</td>
</tr>
<tr>
<td>7. Pay It Forward</td>
<td>12 (6)</td>
<td>8.3</td>
</tr>
<tr>
<td>8. Bigger, Stronger, Faster</td>
<td>10 (5)</td>
<td>6.3</td>
</tr>
<tr>
<td>9. Altered States</td>
<td>10 (5)</td>
<td>5.6</td>
</tr>
<tr>
<td>10. Three’s Company</td>
<td>24 (12)</td>
<td>17.0</td>
</tr>
<tr>
<td>Total</td>
<td>186 points (93 minutes)</td>
<td>177</td>
</tr>
</tbody>
</table>

TOTAL 177  MAX 132 AVG 104 MIN 42
1. **Justified. True or False with Justification**

Circle whether each statement in questions a) to g) is True or False, and **give a one-sentence justification.** We reserve the right to dock points for long-winded answers. Be concise! If you feel like you can give a clear justification in just one word, please do!

**3 pts/question; -2 if explanation partially correct**

-1 if explanation “clueless.”

a. Designing a processor with dynamic multiple issue places extra burden on the compiler author to ensure the correctness of emitted code.

**TRUE/FALSE**

Justification:

*The instruction set doesn’t change because of multiple issue.*

b. As a result of hitting the Power Wall, Moore’s Law no longer holds.

**TRUE/FALSE**

Justification:

*Moore's Law is about transistor count, not clock rates, which are what is limited by the Power Wall.*

c. Increasing the depth of a pipeline tends to increase performance primarily by decreasing the amount of time it takes on average to execute a particular instruction.

**TRUE/FALSE**

Justification:

*Deselect time between instructions* → *increased throughput but not decreased latency.*

d. In a single-cycle MIPS CPU during the execution of a jump instruction, the adder block inside the ALU always computes the sum of its inputs.

**TRUE/FALSE**

Justification:

*This hardware is still powered and has inputs, even though its output is not used.*

e. After enough time, a circuit composed of combinational logic without feedback will always show the same output for a given set of inputs.

**TRUE/FALSE**

Justification:

*This is the definition of combinational logic. (It is stateless)*
f. For pipelined execution, if multiple exceptions happen in the same clock cycle for different instructions, the hardware should set the EPC to address of the earliest issued instruction.

\[ \text{\textbf{TRUE/FALSE}} \]

\textbf{Justification:}
(Assuming "earliest issued" means "earliest issued among these exceptions" ignoring out-of-order issue.)

Exception handlers should be invoked as if instructions are executed one at a time.

\[ \text{\textbf{TRUE/FALSE}} \]

\textbf{Justification:}
SIMD widths are growing nearly as fast as core counts.

h. A particular memory access results in a page fault but not a protection error. Which of the following statements about this access is \textbf{always} true? Circle the answer.

- \text{\textbf{Always true/Not always true}} There was a TLB miss.
- \text{\textbf{Always true/Not always true}} The cache(s) will miss.
- \text{\textbf{Always true/Not always true}} Space for the required page is allocated somewhere on disk.

\textit{S(accepted, not always true w/explanation)}

i. Which of the following could be enabled by adding more ALUs to a processor design?
(Check all that apply.)

- \checkmark More structural hazards
- \x More issue slots
- \x Wider SIMD width
- \__ Larger CPI

-1 for each wrong (max -3)

j. Multiple Choice. (Pick one.) Virtual machines are used in Cloud Computer by Amazon to

I. Protect users from each other
II. Offer more price points
III. Simplify introduction of new and faster hardware

A. None of the above
B. I only
C. II only
D. III only
E. I and II
F. II and III
G. I and III
H. I, II, and III

\[ (-1 \text{ for each of I, II, III not identified}) \]
2. **Virtuosity**

A computer has a 32-bit virtual address with **8 KB pages** and a 16 KB cache with 32 Byte blocks that is 2-way set associative.

IV. Show how the 32-bit address is divided into the virtual page number and page offset, showing the number of bits in each field:

<table>
<thead>
<tr>
<th>VPN - 19</th>
<th>Page offset - 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

V. Show how the 32-bit address could be divided into the cache tag, cache index, and block offset, showing the number of bits in each field:

<table>
<thead>
<tr>
<th>Tag - 19</th>
<th>I - 8</th>
<th>0 - 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

VI. Given these layouts, must the lookup in the TLB to translate from virtual address to physical address be done **sequentially** before access the cache? Why or why not?

No. The cache index field is contained within the page offset field, so we can retrieve the relevant blocks from the cache while we look up the translation for the VPN in the TLB. We can verify hits/misses on the blocks by using the PFN as the tag.
3. **Mo’ Cache, Mo’ Problems**

a) A naive hardware developer designs the cache protocol for a 2 CPU system, each CPU with a 2 block direct mapped cache, 1 byte blocks. Memory addresses are 32 bits (byte addressed). The cache policies are Allocate on write, and Write through.

Assume that both caches start out with all blocks invalid. Which of the following access patterns, *executed independently from one another*, will always yield correctly updated results? Assume that each Read/Write finishes completely during its time period.

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Time 1</th>
<th>Time 2</th>
<th>Time 3</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.</td>
<td>1</td>
<td>Write 0x0</td>
<td>---</td>
<td>Read 0x0</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>---</td>
<td>Write 0x0</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>ii.</td>
<td>1</td>
<td>Write 0x0</td>
<td>---</td>
<td>Read 0x1</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>---</td>
<td>Write 0x1</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>iii.</td>
<td>1</td>
<td>Write 0x0</td>
<td>---</td>
<td>Read 0x0</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>---</td>
<td>Write 0x2</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>iv.</td>
<td>1</td>
<td>Read 0x0</td>
<td>---</td>
<td>Read 0x2</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>---</td>
<td>Write 0x2</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>v.</td>
<td>1</td>
<td>Read 0x0</td>
<td>---</td>
<td>Read 0x0</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>---</td>
<td>Write 0x0</td>
<td>---</td>
<td></td>
</tr>
</tbody>
</table>

b) What feature could you add to this cache protocol to make it work in all cases? Be specific; your revision should describe a particular behavior.

*Invalidation on Write.*
4. My Secret Cache

The average memory access time (AMAT) is:

\[ \text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1} \]

(a) Write down the AMAT equation for a three level cache (use HT for Hit Time, MR for Miss Rate, and MP for Miss Penalty):

\[ \text{HT}_{L1} + \text{MR}_{L1} \times (\text{HT}_{L2} + \text{MR}_{L2} \times (\text{HT}_{L3} + \text{MR}_{L3} \times \text{MP}_{L3})) \]

(b) Given the following specification:

For every 1000 CPU-to-memory references
- 40 will miss in L1;
- 20 will miss in L2;
- 10 will miss in L3;
L1$ hits in 1 clock cycle;
L2$ hits in 10 clock cycles;
L3$ hits in 100 clock cycles;
Main memory access is 400 clock cycles;
There are 1.25 memory references per instruction; and
The ideal CPI is 1.

Answer the following questions:

(i) What is the local miss rate in the L2$?

\[ \frac{20}{40} = 50\% \]

(ii) What is the global miss rate in the L2$?

\[ \frac{20}{1000} = 2\% \text{ or } 4\% \times 50\% = 2\% \]

(iii) What is the local miss rate in the L3$?

\[ \frac{10}{20} = 50\% \]

(iv) What is the global miss rate in the L3$?

\[ \frac{10}{1000} = 1\% \text{ or } 50\% \times 50\% = 1\% \]

(v) What is the AMAT with all three levels of cache?

\[ 1 + 4\% \times (10 + 50\% \times (100 + 50\% \times 400)) \]
\[ = 1 + 1.4 + 2\% \times (100 + 200) = 1.4 + 6 = 7.4 \]

(vi) What is the AMAT for a two-level cache without L3$?

\[ 1 + 4\% \times (10 + 50\% \times 400) \]
\[ = 1 + 1.4 + 4\% \times 200 = 9.4 \]
(vii) What is the AMAT for a one-level cache without L2$ and L3$?

\[ 1 + 4 \% (400) = 1.7 \]

(viii) What is the average memory stalls per reference in the system of question (v)?

\[ 7.4 - 1 = 6.4 \]

(ix) What is the average memory stalls per reference in the system of question (vi)?

\[ 9.4 - 1 = 8.4 \]

(x) What is the average memory stalls per reference in the system of question (vii)?

\[ 17 - 1 = 16 \]

(xi) What is the average memory stalls per instruction in the system of question (v)?

\[ 6.4 \times 1.25 = \frac{8}{5/4} = 1.6 \]

(xii) What is the average memory stalls per instruction in the system of question (vi)?

\[ 8.4 \times 1.25 = \frac{10}{5/4} = \frac{10}{1.25} = 8 \]

(xiii) What is the average memory stalls per instruction in the system of question (vii)?

\[ 16 \times 1.25 = 20 \]

(xiv) What is the performance of the system of question (vii) versus that of question (v)? (Long-division challenged can give just the ratio.)

\[ \frac{16}{6.4} = \frac{2.5}{1.3} \]

(xv) What is the performance of the system of question (vi) versus that of question (v)? (Long-division challenged can give just the ratio.)

\[ \frac{8.4}{6.4} = 1.3 \]
5. Heaven’s Gate
What is the truth table for the following circuit? (It might be simpler to first write the equation)

\[ (\overline{A \cdot B} \cdot C) = A + \overline{B} + \overline{C} \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1 POINT PER CORRECT ANSWER, UNLESS (NUPTED AND US NAND), THEN GET 5 POINTS OUT OF 8
6. Off the Map

Given:

typedef struct llnode {
    struct llnode *next;
    int contents;
} llnode;

Implement map in C. map should take a pointer to an llnode and a map_fun, which is just a
pointer to a function taking one int argument and returning int. It should return a pointer to a new
linked list, where the contents of each of the new nodes is equal to the value returned by the map
function called on the contents of the corresponding old node. Don’t forget to fill in the return
value correctly (pointers to function are tricky so we’ve done the arguments for you). Assume
llnode lists are terminated by a NULL pointer. You should not modify any part of the list
identified by head. You may assume map_fun has been properly declared by a typedef
somewhere else.

```c
llnode *map(llnode *head, map_fun f)
{
    if (head)
        return NULL;
    llnode *new = malloc(sizeof(llnode));
    new->contents = f(head->contents);
    new->next = map(head->next, f);
    return new;
}
```

(Note that to call a function pointer, you just use it like any other function. So, in the code
template given above, saying

```c
int result = f(3);
```

would correctly call the function pointed to by f with the value 3 and put the return value into
result.)
7. Pay It Forward
Consider the excerpt below of a 5-stage pipelined MIPS datapath.

![Diagram of a 5-stage pipeline](image)

a. Consider the following sequence of instructions

   [A] srl $zero, $zero, 0
   [B] addu $t0, $t1, $t2
   [C] addu $t0, $t0, $t2
   [D] lw $s0, 0($t3)
   [E] subu $t3, $s0, $t0

   During which of these instructions' decode stages in the sequence above should ControlRS be 1 to avoid pipeline stalls? Use the labels [A], [B], ...

   \[ \begin{align*}
   &c \quad -3 \text{ if } C \text{ omitted } \quad -1 \text{ for } [B] \text{ alone} \\
   &1 \quad \text{ if others added} \quad \text{(any stop)}
   \end{align*} \]

b. Which fields of which instructions from part a does the control logic need to compute the value of ControlRS?

   - opcode of all instructions (to check destination, R/I-type)
   -_rs of R-type instructions
   -_rd of R-type instructions

   \[ \begin{align*}
   -1 \text{ pt - if of R-type instruction} \\
   -2 \text{ pts - rd} \\
   -2 \text{ pts - } rs \\
   -3 \text{ pts - no opcode/selecting by instruction type} \\
   \end{align*} \]

   (Additional points for selectivity by instruction type/number of fields used)
8. Bigger, Stronger, Faster:
Suppose that you are running an algorithm for various problem sizes, and have obtained the data below. Sketch a weak scaling plot of parallel code performance that shows speedup over the serial implementation. **Be sure to label the Y-axis.**

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Gflop/s (serial)</th>
<th>Threads</th>
<th>Gflop/s (parallel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>200</td>
<td>5</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>400</td>
<td>5</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>600</td>
<td>5</td>
<td>6</td>
<td>25</td>
</tr>
<tr>
<td>800</td>
<td>5</td>
<td>8</td>
<td>35</td>
</tr>
<tr>
<td>1000</td>
<td>5</td>
<td>10</td>
<td>36</td>
</tr>
<tr>
<td>1200</td>
<td>5</td>
<td>12</td>
<td>37</td>
</tr>
<tr>
<td>1400</td>
<td>5</td>
<td>14</td>
<td>37</td>
</tr>
<tr>
<td>1600</td>
<td>5</td>
<td>16</td>
<td>38</td>
</tr>
</tbody>
</table>

Rubric:
- 5 y-axis nonexistent
- 3 plot is above linear line

Weak Scaling of Speedup over Serial

![Graph showing linear speedup]

Threads
Question State Diagram

Given the following timing chart, and assuming 0 set-up, hold, and propagation times, as well as a positive edge-triggered synchronous system,

<table>
<thead>
<tr>
<th>Reset</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>State</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Out</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Clk</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fill-in the transitions and outputs of the following State Diagram so it has the identical behavior to the timing diagram above:

always 1 in State 0
always 0 in State 1
Consider the following datapath with an Arithmetic Logic Unit (ALU) and an eight-register register file organized around a single bus. The ALU is to apply add, subtract, etc. operations to its two input operands to generate an output result. The register file has an asynchronous read and a synchronous write. That is, as soon as the Read Enable (RE) is asserted, the register file selects the indicated 32-bit register and presents its value on the Data Out (DO). On the other hand, the Write Enable (WE) is sampled only on the rising edge of the clock, and only writes the indicated register from the Data In (DI) lines on the same edge that WE is asserted. The ALU and Register File share the Bus via a 32-bit wide 2:1 multiplexer. When SelALU is set to 1, the ALU path is connected to the Bus. Otherwise the Register File path is connected to the Bus.

The datapath must support three-address instructions of the form \( R_s \leftarrow R_x \ <\text{op}> \ R_y \).

To make use of a single bus architecture, the ALU can be surrounded by one, two, or three 32-bit temporary registers, labeled A, B, and C, as shown below (the temporary registers are shown as dotted lines – the correct solution requires at least one and possibly all three of the registers):
Using the fewest of the A/B/C registers and possible clock cycles, what is the fewest number of each to implement the register transfer for the instructions of the three-address type (circle one for each):

- Registers: 1
- Cycles: 1

For your answer, on the previous cross out the registers you don’t need, and fill-in the outline of the registers that you do. For each cycle you need, write below the control signals that must be asserted to implement the register transfers for the three-address instructions:

**Cycle 1:**  
Sel ALU: 0, \( R_{E1} \), Reg Rd: 1, Reg Rd Sel: X, CD A,  
(during cycle 1, Reg File \( x \) \( \rightarrow \) Tmp A)

**Cycle 2:**  
Sel ALU: 0, Reg Rd: 1, Reg Rd Sel: Y, CD B,  
(during cycle 2, Reg File \( y \) \( \rightarrow \) Tmp B)

**Cycle 3:**  
Sel ALU: 1, Reg Wr: 1, Reg Wr Sel: Z,  
\( \text{ALU} \rightarrow \text{Op} \) \( \text{Not shown on diagram} \)  
(during cycle 3, ALU \( \rightarrow \) Reg File Z)