University of California, Berkeley – College of Engineering
Department of Electrical Engineering and Computer Sciences

Spring 2008

Instructor: Dr. Dan Garcia

2008-05-19

CS61C Final

After the exam, indicate on the line above where you fall in the emotion spectrum between "sad" & "smiley"...

Last Name	
First Name	
Student ID Number	
Login	cs61c-
Login First Letter (please circle)	a b c d e f g h i j k l m
Login Second Letter (please circle)	a b c d e f g h i j k l m n o p q r s t u v w x y z
*The name of your **SECTION** TA (please circle)*	Ben Brian Casey Dave Keaton Matt Omar
Name of the person to your Left	
Name of the person to your Right	
All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who have not taken it yet. (please sign)	

Instructions (Read Me!)

- This booklet contains 9 numbered pages including the cover page.
  Put all answers on these pages; don't hand in any stray pieces of paper.
- Please turn off all pagers, cell phones & beepers. Remove all hats & headphones. Place your backpacks, laptops and jackets at the front. Sit in *every other* seat. Nothing may be placed in the "no fly zone" spare seat/desk between students.
- You have 180 minutes to complete this exam. The exam is closed book, no computers, PDAs or calculators. You may use two pages (US Letter, front and back) of notes and the green sheet.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. When we provide a blank, please fit your answer within the space provided. "IEC format" refers to the mebi, tebi, etc prefixes.
- **You must complete ALL THE QUESTIONS, regardless of your score on the midterm.** Clobbering only works from the Final to the Midterm, not vice versa. You have 3 hours... relax.

Question	M1	M2	M3	Ms	F1	F2	F3	F4	Fs	Total
Minutes	20	20	20	60	24	30	24	42	120	180
Points	**10**	**10**	**10**	**30**	**18**	**24**	**18**	**30**	**90**	**120**
Score										

Confessional:

Scanned by CamScanner

Name: ____________________ Login: cs61c-____

M1) Hey buddy, can you run these instructions for me? Thanks! (10 pts, 20 min)

Consider the following *non-delayed branch* MIPS function `foo`:

```
foo:  li $v0,0
      la $t9,loop
      sw $a1,0($t9)
      sw $a2,4($t9)
      sw $a3,8($t9)
loop: nop
      nop
      nop
      bne $a0,$0,loop
      jr $ra
```

a) What does the following function call (in C) return? ________

`foo(-1, 0x30880001, 0x00481020, 0x00042042);`

b) You can probably see how **foo** could pose a security threat if misused. For the good of humanity, we must seal its functionality forever, and render it harmless. That is, you're going to call it once with a special set of arguments for **$a0-$a3** (list these below in human-readable form ... not as numbers!) so that every **future** call to **foo** always just returns **$a0 regardless of the value of $a1-$a3**. Oh, and the call to **foo** with the arguments below should cause it (this time only) to return 0 to signal success that it has been "neutralized".

`$a0:` ______________________

`$a1:` ______________________

`$a2:` ______________________

`$a3:` ______________________

Scanned by CamScanner

Name: ______________________ Login: cs61c-____

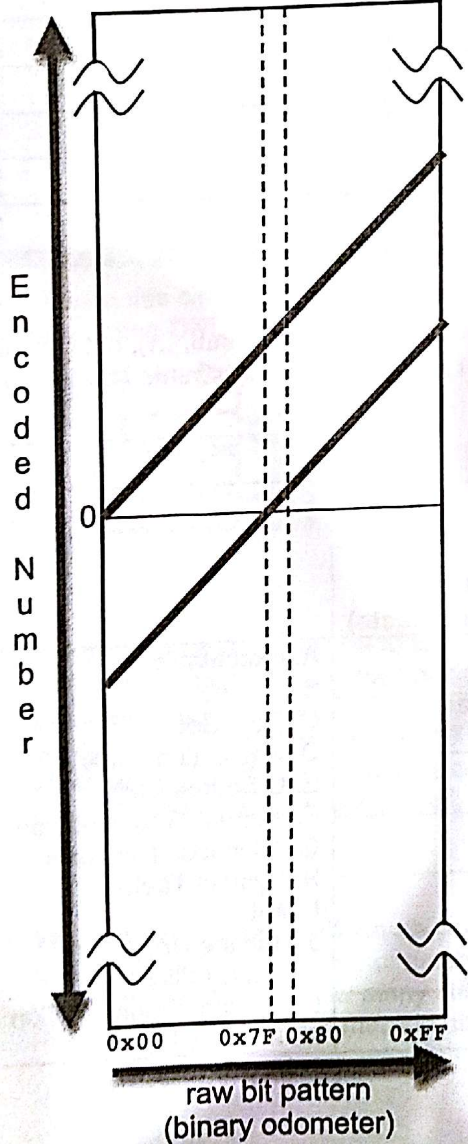
M2) This SEEEEMMMs hard! Yeah, but you're biased! (10 pts, 20 mins)

For this entire question, we only have *a byte* and consider different numeric encodings. Let's compare a floating point SEEEEMMM (1 Sign bit, 4 Exponent bits, 3 Mantissa bits) encoding with a biased encoding (the one we use to store the exponent in a 32-bit `float`) and an unsigned number. Given a raw bit pattern (in hex), we can ask each encoding what number it means. E.g., the raw bit pattern `0xFF` is a `NaN` for SEEEEMMM, but `128` for the biased encoding. One could plot the raw bit pattern vs. the number each encoding represents; if the result is not a number (e.g., ∞ or `NaN`), we just don't plot it. We've already plotted the biased and unsigned lines. Sketch the SEEEEMMM curve and answer the following questions. Your sketch can be quite rough. I.e., there's no need to calculate exact points or intersections, as long as *the number of intersections* is correct & it has the right general shape.

a) How many times does the SEEEEMMM curve intersect the unsigned line? ____________

b) How many times does the SEEEEMMM curve intersect the biased line? ____________

Graphs of encoded number vs raw bit pattern
We've done the biased & unsigned lines.



d) Every region where the SEEEEMMM curve has a slope of 1 (i.e., is it equal to the slope of the two lines drawn), write down the ranges of the raw bit patterns and the difference from the unsigned line. E.g., (these are just for illustration, they're wrong) "from `0x3F` to `0xEA` it's 99 more than unsigned, and from `0xF1` to `0xF5` it's 10 less than it." Fill in the table below; you may not need all rows.

From	To	SEEEEMMM compared to unsigned line
0x3F	0xEA	99 more
0xF1	0xF5	10 less

Scanned by CamScanner

Name: ____________________ Login: cs61c-____

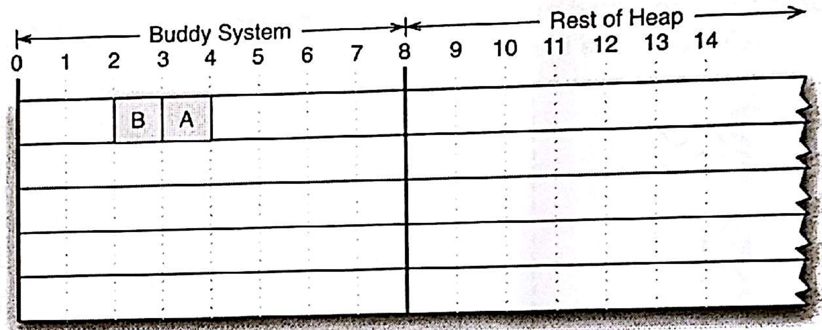
M3) Memories, like the smile we left behind… (10 pts, 20 mins)

a) Here is code and output for a small program, called `memtest`, run on an unknown machine with an unknown architecture. What are **two possible reasons** the program didn't print out `1`?

```
main() {
  char *p = (char *) malloc (1);
  char *q = (char *) malloc (1);
  printf("%u\n", (unsigned int) q - (unsigned int) p);
}
unix% ./memtest
16
```

b) Consider the heap pictured below. **The first 8 bytes are reserved for the buddy system.** The heap is filled left to right when more than one slot can satisfy a request. Any free buddy pairs are consolidated immediately when possible. For the series of memory requests below, fill in the heap snapshot **after every 2nd line**. Be sure to label your allocated space and the block boundaries, as is shown in the initial heap.

```
1  C = malloc(1);
2  free(B);
3  D = malloc(2);
4  free(A);
5  E = malloc(3);
6  F = malloc(2);
7  free(D);
8  G = malloc(3);
```



For the following two questions, we are counting *the number of different instructions* (`add`, `sub`, …), not including the arguments they could accept. E.g., `add $t0,$t0,$t0` and `add $s1,$s2,$s3` are only counted as one, `add`.

c) How many unique TAL MIPS instructions *could* (**not** do) we have? ____________

d) How many unique MAL MIPS instructions *could* (**not** do) we have? ____________

e) Fill the table with **ALL** the choices that fit. Some may be blank, and the letters may be used more than once.

	Consumes (input/uses)	Creates/Causes (output/side-effects)
1. Slab Allocator		
2. Assembler		
3. Linker		
4. Compiler		
5. Reference Counting		
6. Copying		
7. Mark and Sweep		

A. Executable
B. `.o` files
C. Root Set
D. Relocation Table
E. C Source Code
F. Internal Fragmentation
G. Allocation Requests
H. Symbol Table
I. MAL
J. Half the Heap
K. Library Files
L. External Fragmentation
M. Memory Leak

Scanned by CamScanner

Name: ______________________ Login: cs61c-____

F1) How many CS majors to change a lightbulb? None, that's HW! (18 pts, 24 mins)

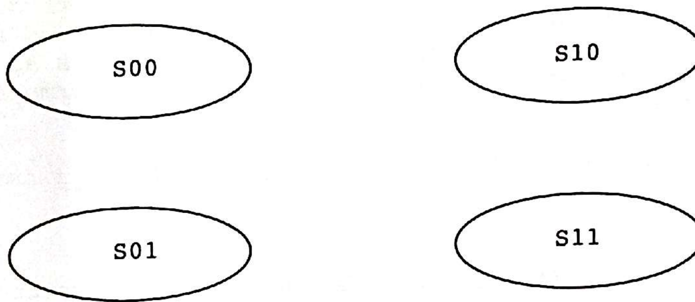
Questions (a), (b), (c) and (d) are independent.

a) You are an intern at a massive hardware firm. Your first task is to design an "odd counter" circuit that receives a single bit input every cycle and outputs a single bit every cycle. It outputs a **1** *if and only if* it has seen an odd number of ones **AND** an odd number of zeros. It starts in a state where it has seen an even number of ones and an even number of zeros (remember, zero is an even number). As an example,

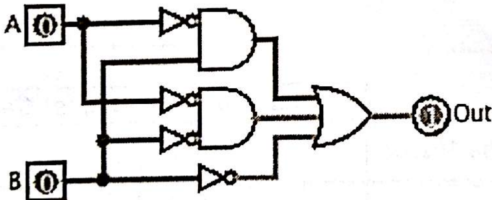
the input: `I: 1 1 0 1 1 1 0 0 0 1 0 1 1 0 0`
will produce the output: `O: 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0`

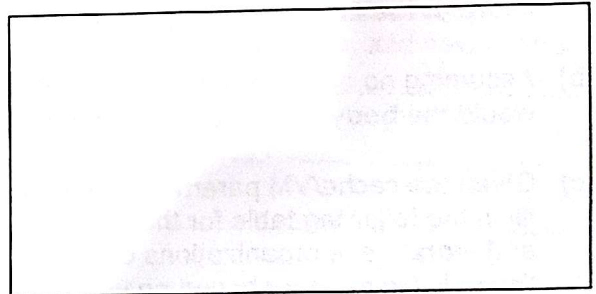
P1	P0	I	O	N1	N0
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Complete the FSM diagram below. The names of the states are arbitrary, but use S00 is the start state. Fill in the truth table on the right. The previous state is encoded in (P1,P0), the next state is encoded in (N1,N0), and the output is encoded as O. Make sure to indicate the value of the *output* on your state.

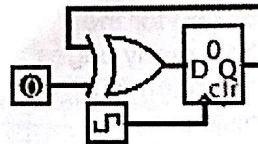


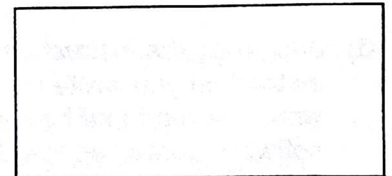
b) Rebuild this circuit with the **fewest** gates in the box to the right, using **ONLY** AND, OR and NOT gates.



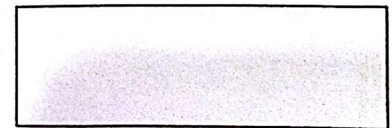


c) Finally, your boss wants you to choose an XOR gate for the circuit to the right: The clock speed is 2Ghz, the setup, hold, and clock-to-q times of the register are 40, 70, and 60 picoseconds (10^{-12} s) respectively. What range of XOR gate delays is acceptable? E.g., "at least W ps", "at most X ps", or "Y to Z ps".





d) You're asked to create **all** the unique 3-to-2 circuits (i.e., 3 inputs: `I2,I1,I0` and 2 outputs), with one minor catch. Your circuit must ignore the value of `I1` if the value of `I2` is 1. How many different circuits will you have to make? Use IEC terminology, like 128 mebicircuits, 512 tebicircuits, etc.



Scanned by CamScanner

Name: ______________________ Login: cs61c-____

F2) You're using circular linked list reasoning (24 pts, 30 mins)

You have a 32-bit MIPS system with...

- 1 MiB of RAM (max)
- Virtual Memory with P-word pages (P is a power of 2, overall page size in the KiB range)
- an L1 write-through data cache with 5 offset bits, 2-way set associative, 4 sets total, LRU replacement

Now, Consider the following code to set up a circular linked list (via an array of `next` pointers).

```
#define NUM_NODES <super-big-number> // Power of 2, much bigger than 2^20

typedef struct node {                   // A pointer to its own type, like a linked list
    struct node *next;                  // without the 'data' field, only a next ptr.
}   node_t;                             // E.g., a cons with only a cdr. sizeof(node_t) = 4

node_t nodes[NUM_NODES];                // Now let's make an array of these pointers

// Set up each 'next' pointer to point to another element in the array to make a circular
// linked list. If we traversed the pointers, we'd visit every element and return to
// the beginning. E.g., If NODES=4, this function could set the pointers so that we
// would visit 0->1->2->3->0->etc OR 0->2->1->3->0->etc OR any other permutation.
CreateCircularLinkedListOfPtrs(nodes);

SomeFunctionWhichTouchesTonsOfMemory(); // After this, nodes are flushed from cache & VM

node_t *ptr = &nodes[0];
for (int i = 0; i < NUM_NODES; i++)
    ptr = ptr->next; // body (let's visit all the nodes once and then return home)
```

a) What single line of C would a **really** smart compiler interpret **the entire `for` loop** code as? ____________________

b) Assuming no optimization, what single MIPS instruction would the **body** of the `for` loop compile to if `ptr` is in `$s0`? ____________________

c) Given the cache/VM parameters above, fill in the following table for the **best** and **worst** case organizations of the linked list in memory based on what `CreateCircularLinkedListOfPtrs` might do.

	Best Case	*Worst Case*
# of Data Cache Misses		
# of Page Faults		

d) *Just to do the instruction fetch* (IF) for the instruction you wrote in (b), how many pages would be read and how many written to a software-controlled RAID 3 disk array **in the worst case?** Assume no disk failures.

____________ read, ____________ written.

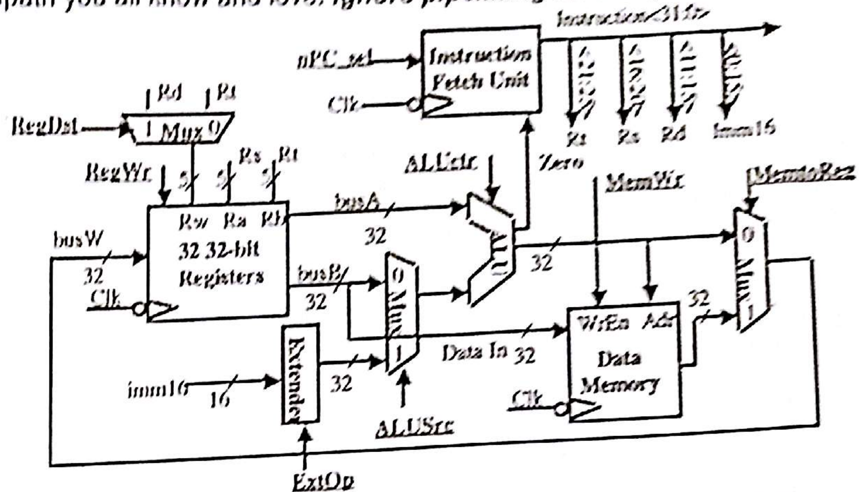
Scanned by CamScanner

Name: ______________________ Login: cs61c-____

F3) You have a case of the soiflz? Go to Datapathology! (18 pts, 24 mins)

On the right is the **single-cycle** MIPS datapath you all know and love. ***Ignore pipelining for the question.*** Your job is to modify the diagram to support a new MIPS instruction to perform the following C code in **one** MAL instruction (ptr is an array of ints):

```
if(ptr[IMMEDIATE] == 0) {
    ptr[IMMEDIATE] = 1;
}
```



We'll call our new instruction soiflz, for <u>s</u>tore <u>o</u>ne <u>i</u>f <u>l</u>oad <u>z</u>ero. If the word (that is stored IMMEDIATE integers past the base pointer in rs) is 0, then set that word to be 1.

a) Make up the syntax for the MAL MIPS instruction that does it (show an example where ptr lives in $t0, and IMMEDIATE is 8). On the right, show the register transfer language (RTL) description of soiflz.

Syntax	*RTL*

b) Change as *little as possible* in the datapath above **(draw your changes right in the figure)** to enable soiflz and list all changes below. Your modification may use adders, shifters, muxes, wires, and new control signals. If necessary, you may replace existing labels. You may not need all boxes.

(i)	
(ii)	
(iii)	
(iv)	

c) We now want to set all the control lines appropriately. List what each signal should be: an intuitive name or {0, 1, x – don't care}. Include any new control signals you added.

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg			

d) Your smart friend argues that because of **this very instruction**, you should have a <u>fourth</u> instruction format (in addition to R, I and J). There's a clear downside: it would cause more complexity with control & datapath. That said, what would be the upside?

Scanned by CamScanner

Name: ____________________ Login: cs61c-____

F4) Please Pass Professor's Pretty Pipeline-pourri Problem... (30 pts, 42 min)

Given a standard five (5) stage pipelined processor with:

- **No Forwarding**
- Stalls on ALL data and control hazards; no out-of-order execution ; non-delayed branches
- Branch comparison occurs during the second stage; instructions are not fetched until branch comparison is done
- Memory CAN be read & written on the same clock cycle
- The same register CAN be read and written on the same clock cycle

a) *Fill in the corresponding pipeline stages* (F,D,E,M,W) at the appropriate times in the table below for the following six MIPS instructions assuming the above properties of your CPU. You don't need to fill anything in for instruction (7).

Instruction	Cycle 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
(1) add $s0 $s1 $t0																			
(2) addi $t0 $t0 4																			
(3) sw $s0 0($t1)																			
(4) and $t1 $t1 $t2																			
(5) lw $t2 4($t1)																			
(6) bne $t2 $t1 -6 #goto 1																			
(7) sub $s0 $s0 $t2																			

b) Assuming we've been in this loop for 1000 iterations, *how many cycles does it take to evaluate each loop?* [ ]

c) You (as the one who wrote the code in part (a)) are pretty unhappy with all the hazards you discovered when filling out the table. Suddenly, Sir Mips-a-lot – a representative of the MIPS fabrication plant – runs in and shouts: "Don't sulk!! We just implemented *delayed branches*!". You decide to move instruction (2) into the delayed branch slot after (6). Again, assuming we've been in the loop for 1000 iterations, *NOW how many cycles to evaluate each loop?* [ ]

d) What *recent breakthrough* allows improved bit density on disk drives? ____________

e) *What's the speedup* (over a 1-core machine) for a 20%-serial program on a 16-core machine? ____________

f) Two threads run `cal` and `bears` below.
What can go wrong and how do we fix it? ____________

```
cal() {
        lock(&lockA);
        lock(&lockB);
        sharedVar++;
        unlock(&lockB);
        unlock(&lockA);
}
```

```
bears() {
        lock(&lockB);
        lock(&lockA);
        sharedVar++;
        unlock(&lockA);
        unlock(&lockB);
}
```

g) A processor run on a particular benchmark has the instruction mix and CPI shown in the table at the bottom right. *How many times faster* would the benchmark run if we quadruple the CPI of the ALU from 2 to 8? ____________

h) Which is *the best way* to communicate with a remote sensor measuring lunar eclipses, via polling or interrupts? ____________

i) What protocol *guarantees* delivery on a network? ____________

	Frequency	CPI
Memory	30%	4
Branch	20%	4
ALU	50%	2 → 8

j) What is one reason MapReduce *is better than* MPI? ____________

Scanned by CamScanner