University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2014 | Instructors: Dan Garcia, Miki Lustig | 2014-12-16

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-`
*The name of your **SECTION** TA (please circle)*	Alex | Andrew | David | Fred | Jay | Jeffrey | Kevin | Matthew Riyaz | Rohan | Roger | Sagar | Shreyas | William
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 8 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. 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	30	30	30	30	120	180
Points	**10**	**10**	**10**	**30**	**22**	**23**	**22**	**23**	**90**	**120**
Score										

SID: ________________

M1) C/MIPS Question (10 pts, 20 mins)

a) Finish the code for **`batoui`**, a recursive function that takes in a binary ASCII string of 0s and 1s (no more than 32) and returns the unsigned int the string represents. E.g., **`batoui("110")`**➔**`6`**. You may find the **`strlen`** function handy. E.g., **`strlen("110")`**➔**`3`**. (5 pts)

```
uint32_t batoui(char *ba) {

    if (*ba) {

        return ( ____________________________________________________ ) ;

    }

    return 0;
}
```

b) Write the MAL MIPS function **`reverse_str(char *string, int string_length)`**, that can reverse strings (with an even length) in-place. The MIPS should be non-delayed branch, and you will probably not use all the lines. (5 pts)

```
reverse_str:  beq $a1 $0 done

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              ______________________________________

              j reverse_str

done:         jr $ra
```

M2) ***Cache Money, y'all*** (10 pts, 20 mins)

Assume we are working in a 32-bit virtual and physical address space, byte-address memory. We have two caches: **cache A** is a direct-mapped cache, while **cache B** is fully associative with LRU replacement policy. Both are 4 KiB caches with 256 B blocks and write-back policy. Show all work!

a) For **cache B**, calculate the number of bits used for the Tag, Index, and Offset: T:___ I:___ O:___

Consider the following code:

```
uint32_t H[32768];          // 32768 = 2^15. H is block-aligned.

for (uint32_t i = 0; i < 32768; i += 2048) H[i] += 1;
for (uint32_t i = 1; i < 32768; i += 2048) H[i] += 2;
```

b) If the code were run on **cache A**, what would the hit rate be? ____________________________ %

c) If the code were run on **cache B**, what would the hit rate be? ____________________________ %

d) Consider several modifications, each to the original **cache A**. How much will the modifications change the hit-rate and why?

i. Same cache size, same block size, 2-way associativity

ii. Double the cache size, same block size

iii. Same cache size, block size is reduced to 8B

e) If you were allowed to modify the code while keeping functionality, what would be the maximum hit rate for the original **cache A**? Explain briefly.

SID: ________________

M3) *What is that Funky Smell? Oh, it's just Potpourri…* (10 pts, 20 mins)

a) By now you've heard that the view count on Psy's "Gangnam Style" on YouTube had an integer overflow. Your friend suggests they should have used a **`float`** instead. Respond by filling in the blanks & show your work. Don't worry about off-by-1s: E.g., if it's 1023, say "1 Kibi". (4 pts)

"Whereas an **`int32_t`** failed at (use IEC format) ____________, a **`float`** would have failed at

(use IEC format) ___________, at which point **`f=f+1`** would have…" (state what happens & why):

b) Consider a new scheme to represent *signed* numbers in binary. It functions in the same fashion as any other radix, such as hexadecimal and binary, but uses a base of negative 2. Fill out the table below, assuming 6-bit numbers. The first row has been done for you. (2 pts)
Show your work below:

$$N = \sum_{i=0}^{M-1} b_i \cdot (-2)^i$$

where b_i is the ith bit, M is the total # of bits.

Base -2	Decimal
000110	2
010101	
	-13

c) Complete the code below, using *at most two* TAL MIPS instructions, so that the function returns *false* if **`$a0`** contains an R-type instruction and *true* otherwise. (2 pts)

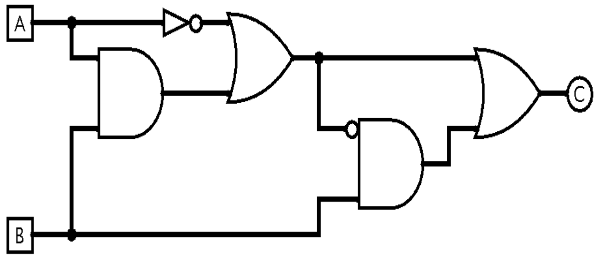
```
NotRType: ______________________________

          ______________________________
          jr $ra
```

d) We are designing a 64-bit MIPS architecture (64-bit words, 64-bit instructions). We must support at least as many instructions as MIPS-32, and all MIPS-32 operations (**`add`**, **`lui`**, etc.) must be supported. If we wanted to maximize the number of registers each register field can address (all register fields should be the same width), what is the maximum number of registers we can address? Put your answer in IEC format and show your work. (2 pts)

F1) Madonna revisited: *"We are Living in a Digital World"* (22 pts, 23 mins)

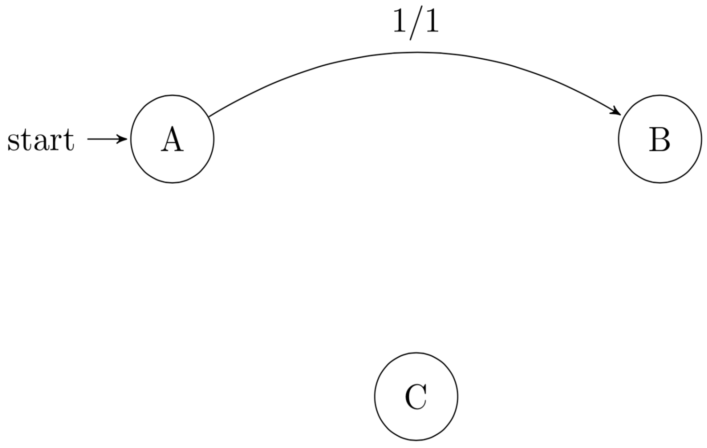
a) Give the simplest Boolean expression for the following circuit in terms of A and B, using the minimum number of AND, OR, and NOT gates:



C = ______________________

(You must show your work above to earn points.)

b) Using as few states as possible, complete the following finite state machine that takes a ternary (base-3) digit as input (0, 1, or 2). This machine should output a 1 if the sequence of ternary digits forms an odd number, otherwise it should output a 0.



Example: 1, 1, 2 → 112_3 (14_{10}) → even

Assume you have seen no digits at the start state. You might not need all of the states, and you should not draw additional states. *Finally, you will receive no credit for drawing something that is not an FSM!*

c) If the delay through a single-bit adder is 3 (measured in gate delays) to the sum output and 2 to the carry output, what is the delay through a k-bit ripple-carry adder? _______

F2) *V(I/O)rtual Potpourri* (23 pts, 30 mins)

For the following questions, assume the following:

- 16-bit virtual addresses
- 4 KiB page size
- 16 KiB of physical memory with LRU page replacement policy
- Fully associative TLB with 4 entries and an LRU replacement policy

a) What is the maximum number of virtual pages per process? ________________________

b) How many bits wide is the *page table base register*? ________________________

For questions (c) and (d), assume that:

- Only the code and the two arrays take up memory
- The arrays are both page-aligned (starts on page boundary)
- The arrays are the same size and do not overlap
- ALL of the code fits in a single page and this is the only process running

```
void scale_n_copy(int32_t *base, int32_t *copy, uint32_t num_entries,
        int32_t scalar)
{
      for (uint32_t i=0; i < num_entries; i++)
          copy[i] = scalar * base[i];
}
```

c) If **`scale_n_copy`** were called on an array with **`N`** entries, where **`N`** is a multiple of the page size, how many page faults can occur in the worst-case scenario?

Answer: ______________

d) In the best-case scenario, how many iterations of the loop can occur before a TLB miss?

Answer: ______________

e) Which type of RAID (0, 1, 2, etc.) does not provide any redundancy? What is the benefit of this type of RAID?

f) In one sentence, explain why polling may be a better choice than interrupts for capturing mouse cursor positions.

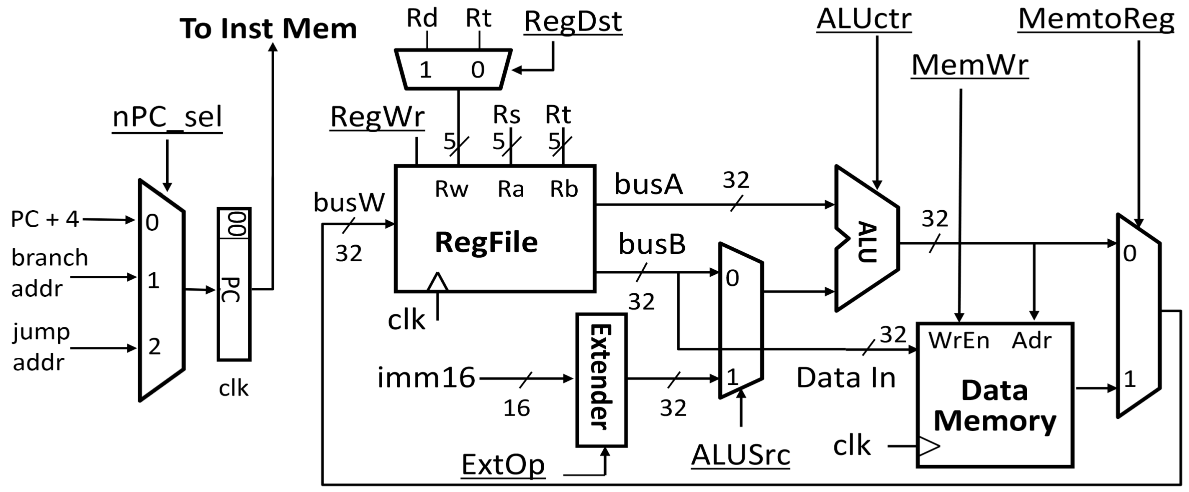
g) In one sentence, name a benefit Magnetic Disks have over Flash Memory (SSDs).

F3) *Datapathology* (22 pts, 30 mins)

Consider the following instruction: `jals $rt $rs imm`. The instruction stores `PC + 4` in register `$rt`. At the same time, it sets the `PC` to the value in register `$rs` offset by the sign-extended `imm` value.

a) Write the register transfer language (RTL) corresponding to `jals`:

b) Change *as little as possible* in the 1-stage datapath below to support **jals**. In case of ties, pick the set of changes that maximizes the number of control signals that can be set to "don't care". Draw your changes directly in the diagram and describe your changes below. You may only add multiplexers, wires, splitters, tunnels, adders, and add or modify control signals.



Describe your changes below:

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

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg			

For the following questions, assume we have taken the above CPU, converted it into a 5-stage CPU, and implemented forwarding.

d) The code on the right was written for a non-pipelined CPU. After which instructions do `nops` need to be inserted? For each instruction, write the line number and number of `nops`.

```
1    la $t0, someFunc
2    addi $sp, $sp, -4
3    sw $ra, 0($sp)
4    lb $t1, 0($a0)
5    addi $a0, $t1, 0
6    jals $ra, 0($t0)
7    lw $ra, 0($sp)
8    addi $sp, $sp, 4
```

F4) *What do you call two L's that go together?* (23 pts, 30 mins)

The Hamming distance between two bitstrings of equal length is the number of locations in which the bits differ. For example, **`hamming(0b1011101, 0b1001001)`** ➔ **`2`**. Consider the code below:

```
uint32_t hamming(uint32_t x, uint32_t y) {
    uint32_t mask, ham_dist = 0;
    for (int i = 0; i < 32; i++) {
        mask = 1 << i;
        if ((x & mask) != (y & mask)) {
            ham_dist++;
        }
    }
    return ham_dist;
}
```

For questions a-c, assume we parallelize the **`for`** loop using OpenMP.

a) For each variable below, designate it as "shared" or "private" among threads. Do not mark a variable as private if it can be safely shared.

x	**mask**	**i**

b) Is a data race on **`ham_dist`** possible? If yes, explain how to fix it. If no, explain why not.

c) Is false sharing of the variables **`x`**, **`y`** possible? If yes, explain how to fix it. If no, explain why not.

d) Rank techniques A-C from best to worst for the given problems below. If there is a tie between improvements, you may list them in any order.

A: parallel threads (e.g. OpenMP)
B: parallel data (e.g. Intel SSE)
C: distributed computing (e.g. MapReduce)

i. We are computing on bit strings of length 1024 bits?

ii. We want to find the pairwise hamming distance between all the works of Shakespeare?

e) Your friend is analyzing a website and notices that database operations take up 2/3 of the webpage's total loading time. Database operations previously took 200ms per request, and your friend reduced this down to 100ms. What is the speedup observed by the user?