University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

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

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

Last Name	**ANSWER KEY (Version A)**
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	**10**	**10**	**10**	**30**	**22**	**23**	**22**	**23**	**90**	**120**

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) {
                ((uint32_t)(*ba-'0') << (strlen(ba)-1)) + batoui(ba+1)
        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
              addu $t0 $a0 $a1      save pointer (string + string_length) in
                                            register $t0
              ____________________________________________
              addiu $t0 $t0 -1          subtract that value by 1
              ____________________________________________
              lbu $v0 0($t0)      load last character into register $v0
              ____________________________________________
              lbu $v1 0($a0)      load first character into register $v1
              ____________________________________________
              sb $v0 0($a0)
                                        swap the first and last
              ____________________________________________
              sb $v1 0($t0)                characters
              ____________________________________________
              addiu $a0 $a0 1          increment the string pointer
              ____________________________________________
              addiu $a1 $a1 -2      decrement the string_length by 2
              ____________________________________________

              ____________________________________________

              ____________________________________________

              ____________________________________________

              ____________________________________________

              ____________________________________________

              j reverse_str

done:         jr $ra
```

SID: _____

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:___

24, 0, 8

O = log2(256B) = 8, I = 0 (since fully associative), T = 32 - 0 - 8 = 24

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;
```

2 accesses each (read + write)

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

Read is a comp / conflict miss write a hit within the loop 50%

In the first for-loop, the read access is a compulsory miss since it is the first time that the block is being accessed in the cache. The block is retrieved from memory and placed into the cache. The write access is a hit, giving a hit rate of 50%.
In the second for-loop, the accesses will be the same as the first loop, but the read access will be a conflict miss since the block has been in the cache previously but is no longer there. Similarly, the block will be retrieved from memory and placed into the cache. Write access is a hit again, giving a total overall hit rate of 50%

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

For 1st loop miss on read, hit on write. For 2nd loop hits 100%

75%

Cache B is a fully associative cache, so the first available block get filled. The cache holds 2^12/2^8 = 2^4 = 16 blocks. The array size is 2^15 integer = 2^17 bytes and the step size is 2^11 integer = 2^13 bytes. Therefore, each loop accesses 2^17/2^13 = 16 values meaning that every block accesses is stored in the cache. In the first loop, it is 50% with a compulsory read miss and write hit. However, in the second loop, we have a 100% hit since none of the blocks get replaced. This results in a combined hit rate of 75%.

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

The step size of 2^13 bytes means we step over 2^5 = 32 blocks for each loop. This means that we step over the cache twice, hitting the same index each time. Therefore, the blocks in the first set of the cache would be constantly replaced before getting to the second loop.

Associativity too low, capacity misses first entries replaced before 2nd loop -- no change

ii. Double the cache size, same block size

Doubling the cache size means that the cache can hold 2^13/2^8 = 2^5 = 32 blocks. However, we still step over 32 blocks per loop, replacing the same blocks before the second loop.

Still too many conflicts first entries replaced before 2nd loop—no change

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

Block size reduced to 8B means that the cache holds 2^12/2^3 = 2^9 = 512 blocks. However, this still does not change the fact that each loop steps over 2^13 bytes of data, meaning we step over 2^10 blocks each loop, still stepping through the cache twice, hitting the same index each time. It is still the same issue as the above two parts.

Still conflict misses first entries replaced before 2nd loop -- No change

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.

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

two read/writes in the loop, 1 miss 3 hits, so 75% hit rate

This modification allows you to functionally compute both for loops within one loop. This adds more accesses meaning more hits per block. This takes advantage of temporal locality by combining both for loops into one, and spatial locality by accessing H[i] and H[i+1].

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):

int32_t = 2 Gibi, float = 16 Mebi. f would have failed to increment (f would have stayed the same) because floats lose the ability to count by ones OR would increment by 2 if rounding mode set to "toward +inf". This is because floats have 23 mantissa bits, and since $2^{24} + 1 = 2^{24} + 2^0$, it would require 24 mantissa bits to represent.

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 i[th] bit, M is the total # of bits.

Base -2	Decimal
000110	2
010101	21
110111	-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)

```
                lui $t0, 0xFC00 (or, it's just 1 line: "srl $v0 $a0 26")
NotRType:       _____________________________
                and $v0, $a0, $t0
                _____________________________
                jr $ra
```

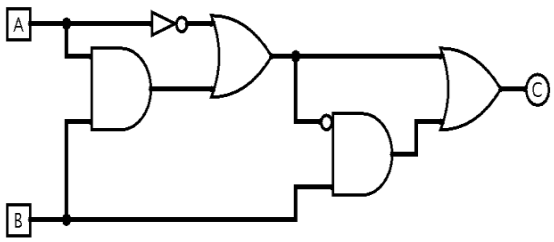
d) We are designing a 64-bit MIPS architecture (64-bit words, 64-bit instructions). We must support at least as many instructions as MIPS-32, and all MIPS-32 operations (`add`, `lui`, etc.) must be supported. If we wanted to maximize the number of registers each register field can address (all register 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)

For R-type instructions: We need at least 6 bits for opcode, 6 bits for funct, 6 bits for shamt. This leaves us with 46 bits to be divided among three register fields.
For I-type instructions: We need 6 bits for opcode and 32 bits for imm (for lui), leaving 26 bit to be divided among 2 instructions.
Since we need to support both, we take the minimum, so each field can be at most 13 bits wide. Thus we can address 2^13, or 8 Kibi registers.

SID: _____

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 = (~(AB + ~A))B + (AB + ~A)
  = ((~A + ~B)A)B + AB + ~A
    = (~BA)B + AB + ~A
      = AB + ~A
       = ~A + B
(true if ~A or if A, can only be true if B)
```

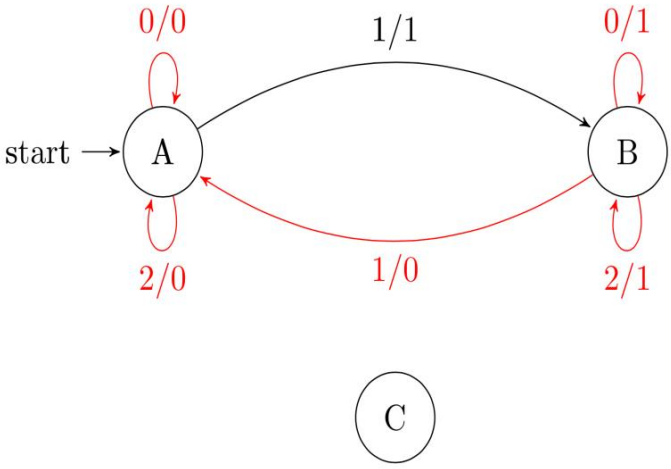
C = ___~A + B_____

(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!*




```
State A represents even number,
state B represents odd number
```

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?

2k+1

SID: _____

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

The virtual address is made of [VPN | offset]. # offset bits = log2(page size) = 12. This leaves 4 bits for the VPN. Max number pages representable with 4 bits is 2^4 = 16 pages

a) What is the maximum number of virtual pages per process? **16**

b) How many bits wide is the *page table base register*? **14**

The page table base register points to the beginning of the page table of this current process in memory. There is 16 KiB of physical memory, represented by 14 bits

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?

*The physical memory can hold 4 pages. 1 pages is taken up by the code. Worse case scenario is a page fault for every page access. Considering 32-bit integers, there are 2^12 / 2^2 = 2^10 integers per page. Since there are 2 arrays, there are (N/(2^10))*2 = N / 2^9 faults for array access. We must add 1 more for a page fault in accessing the page with the code.*

Answer: **N/(2^9) + 1**

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

The best case scenario is if the TLB is filled with valid entries, 1 for the code, 1 for copy, 1 for base, and 1 other (either copy or base). Since we can only get through one page before either copy or base has to be fetched, we can get through 2^10 iterations (# of integers in one page).

Answer: **2^10**

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

Raid 0 has no redundancy. Large accesses are faster since disk transfer is parallelized.

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

Interrupts have more overhead than polling, so polling can be faster for slow, often ready devices like mice

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

Magnetic disks do not have a finite number of write cycles, unlike SSDs

SID: _____

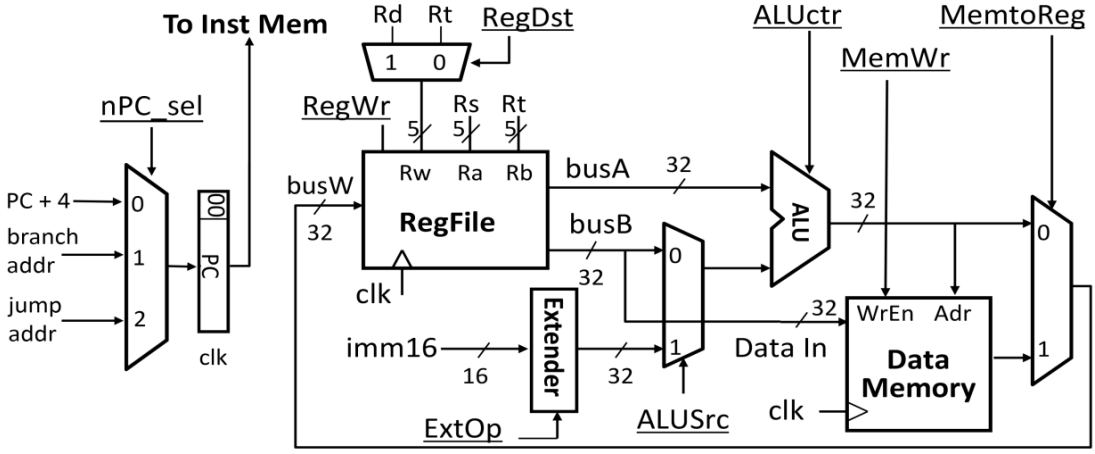
F3) *Datapathology* (22 pts, 30 mins)

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`**:

`R[rt] <- PC + 4; PC <- R[rs] + SignExtImm`

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:

Add a mux to busW and add a new control signal "RegSrc" to control the mux. Connect the existing busW input and PC + 4 to the new mux. Connect the ALU output to the mux controlled by nPC_sel under port 3 (nPC_sel itself doesn't need to be changed since it's already a 2-bit signal)

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	RegSrc		
0	1	3	Sign	1	Add	0	X	ALU out		

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

```
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
```

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`**.

After line 4: 1 stall
After line 6: 2 stalls

After line 4, we need 1 stall to wait for $t1 value. However, since forwarding is implemented, we only need to stall once to forward from MEM of line 4 to EX of line 5. After line 6, we must stall by 2 cycles to determine the new PC after the EX stage.

SID: _____

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
shared	**private**	**private**

x can be shared since x does not get modified. mask and i must be private since the values are specific to each for loop iteration

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

Yes, we need to add a #pragma omp critical and partial sums in each thread or use the reduction keyword. *Since we are incrementing ham_dist by 1, it is dependent on its current value. Each thread is modifying the global value of ham_dist, causing a data race.*

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

No, since we are not writing to them.

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)

When computing a 1024 bit string, the scale is so small that distributed computing would amount a lot of overhead. However, with all the works of Shakespeare, it would be the most efficient to spread out the work among many distributed systems as opposed to parallelizing in one system.

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

A, B, C or B, A, C

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

C, A, B or C, B, A

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?

3/2

speedup = 1 / ((1 - F) + F/S_E) = 1/((1/3) + (2/3)/2) = 3/2

lecture 18, slide 43