University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2014 | Instructor: Dan Garcia | 2014-05-13

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

Last Name	**Answer Key**
First Name	
Student ID Number	
Login	cs61c-
Login First Letter (please circle)	a b c d e f g h i j k m o p
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)*	Alan | Jeffrey | Kevin | Roger | Sagar | Shreyas | Sung Roa | 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 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. 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) *What's that funky smell? Oh, it's Potpourri* (10 pts, 20 min)

(This is for M1a, M1b) A variant of the MIPS instruction set has been developed on the new 64-bit system for cloud computing. To emphasize the "big" in big data, the size of the *Opcode* field has been increased by one bit, and the number of registers (whose width is now 64-bits) was quadrupled. Assume that instructions "expand" the rightmost field to fill all 64 bits. So here, the *Funct* field would expand; in an I-type instruction, the *Immediate* would expand. *You may use the boxes below as scratch space.*

7	7	7	7	6	30
Opcode	*Rs*	*Rt*	*Rd*	*Shamt*	*Funct*

a) How many total instructions can we have? Please leave your answer as an expression involving powers of 2.

$(2^7 - 1)$ non-R-type + 2^{30} R-type

b) What is the maximum amount of bytes we could change the PC by with a single branch instruction?

(Express your answer in IEC format, e.g., 32 Kibi, 16 Mebi, etc.)

Bits for Imm = 64-7*3=44, counting by instr that's 8 bytes (3 free) = ±2^47 = 128 Tebi range = 64 Tebi for max branch.

```
typedef struct { int32_t len; double *data; } vector;
double min = -0.8, max = 1.7;

vector *filter(char *filename) {
    char *extension = ".txt";
    int32_t i = 0;
    vector *output = malloc(sizeof(vector));
    ... // ← here
```

c) Assume our memory is 32-bit addressed. *How many bytes are allocated in each section of memory as a result of these lines (up to "← here")?* Include allocations ONLY from the lines shown; assume registers are not used.

Stack: 4+4+4 = 12 Heap: 4+4 = 8 Static: 16+4+1 = 21

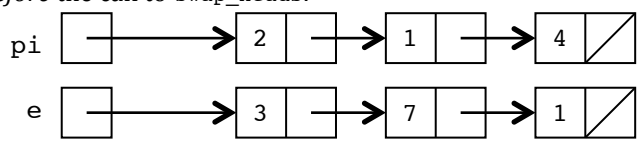
---

d) Complete the following code so it obeys its comments:

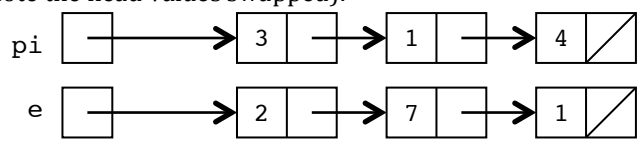
```
typedef struct list {
      int val;
      struct list *next;
} list;

/* Swaps values at the front of lists x and y */
void swap_heads(list *x, list *y) {
      list tmp = {y->val, x}; // new list node
      x = &tmp; y->val = x->next->val;
      x->next->val = x->val;
      _____________________________________
}
```

before the call to `swap_heads`:



...and *after* the `swap_heads(pi,e)` call (note the head values swapped):



---

e) What is the ratio of the # of numbers a `double` can encode between 0.5 and 1 to the # of numbers a `double` can encode between 5 and 7?

(Express your answer in IEC format, e.g., 32 Kibi, 16 Mebi, etc.) (0.5-1 is full range, 5-7 is half of 4-8, so **2**)_

SID: _____

M2) *Cache, money y'all* (10 pts, 20 min)

We have a standard 32-bit byte-addressed MIPS machine with 4 GiB RAM, a 4-way set-associative CPU data cache that uses 32 byte blocks, a LRU replacement policy, and has a total capacity of 16 KiB. Consider the following C code and answer the questions below.

```
#define SIZE_OF_A 2048

typedef struct {
      int x;
      int y[3];
} node;

int count_x(node *A, int x) {
      int k = 0;
      for (int i = 0; i < SIZE_OF_A; i++)
            if (A[i].x == x) {
                  k++;
            }
      return k;
}
```

a) How many bits are used for the tag, index, and offset?

Tag	Index	Offset
20	7	5

b) We call `count_x` with all values of `x` from 0 to 65535 to count the number of times that each `x` occurs in `A`. The value of `A` is the same in every call. The cache is cold at the beginning of execution. What is the cache hit rate? 50%

Questions (c) and (d) below are two ***independent*** variations on the original problem.

c) Let's say that we increase our CPU cache associativity to 8-way.
What is our cache hit rate now? 50%

d) What would be the approximate cache rate if we changed our CPU cache to use a *Most Recently Used* (MRU) cache replacement policy, and we change the cache to be *fully associative*? 75%

Half the array fits in cache, on the first cold pass it'll be (first-half-minus-last-block, last-block) at 50%. The next iteration will hit all of the first-half-minus-last-block (100% hits), then the "last block in the first half" will kick out the "all but" last block, but then you get a hit on the last block too, so it's exactly half with 100% hits, and half with 50% hits.

SID: _____

M3) *MIPS stands for "MIPS is pretty sweet"!…* (10 pts, 20 min)

```
# a0 is a pointer to an array of ints
# a1 is the length of the array
                                    ##  Use this area for your own notes
1 mystery: la $t2, loop             ##  (Assume this takes only 1 TAL instruction)
2 loop:    addiu $t9, $a0, 0        ##
3          lw  $t0, 0($t9)          ##
4          addiu $t0, $t0, 1        ##
5          sw  $t0, 0($t9)          ##
6          lw $t1, 0($t2)           ##
7          addiu $t1, $t1, 4        ##
8          sw $t1, 0($t2)           ##
9          addiu $a1, $a1, -1       ##
10         bne $a1, $0, loop        ##
11         jr $ra                   ##
```

a) At a functional level, *what does this code do* (for "small" array arguments), called for the first time?

It increments the first $a1 words (in array $a0) by 1.

b) Approximately what is the *most array elements* this code can handle correctly (called for the first time)?
(Express your answer in IEC format, e.g., 32 Kibi, 16 Mebi, etc.)

2^{16} / 4 = 2^{14} = 16 Kibi

c) What would happen if we *exceed* this value by one array element (called for the first time)?
(Mention a line number, an instruction field, and what happens when the code is run to completion in your answer)

The immediate field in line 2 overflows and you end up trying to read and write from a random place in memory (whatever the current value of $a1 happens to be, which is its initial value minus 2^{16}).

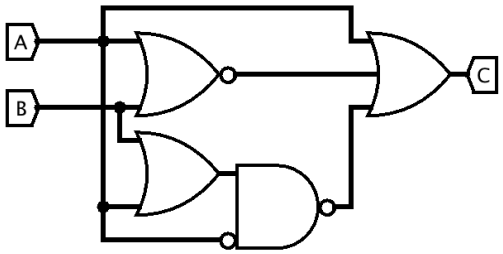
d) You'll notice that we keep saying "called for the first time", because this code can only be called once! Specify three MIPS lines you could put after line `10` (say as `10.1`, `10.2`, `10.3`) that would allow this code to be reused.
(This code should even work if the array size is exceeded, and it didn't crash after what you described in (c) happened)

```
     addiu $t9, $a0, 0
10.1 _____________________________
     lw $t1 36($t0)
10.2 _____________________________
     sw $t1 0($t0)
10.3 _____________________________
```

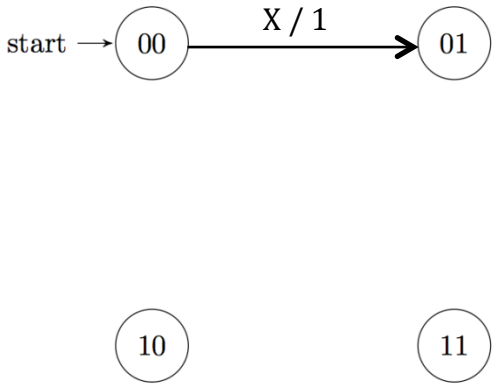
SID: _____

F1) Madonna revisited: *"We Are Living in a Digital World…"* (22 pts, 30 mins)

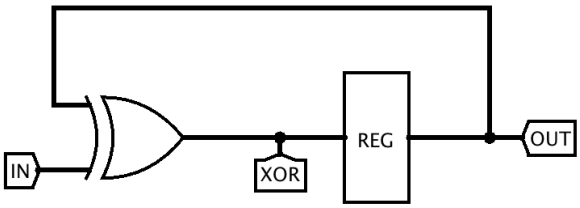
a) Rewrite the following circuit using the minimum number of AND, OR, and NOT gates: _____A+!B_____
You must show your work above to earn points.

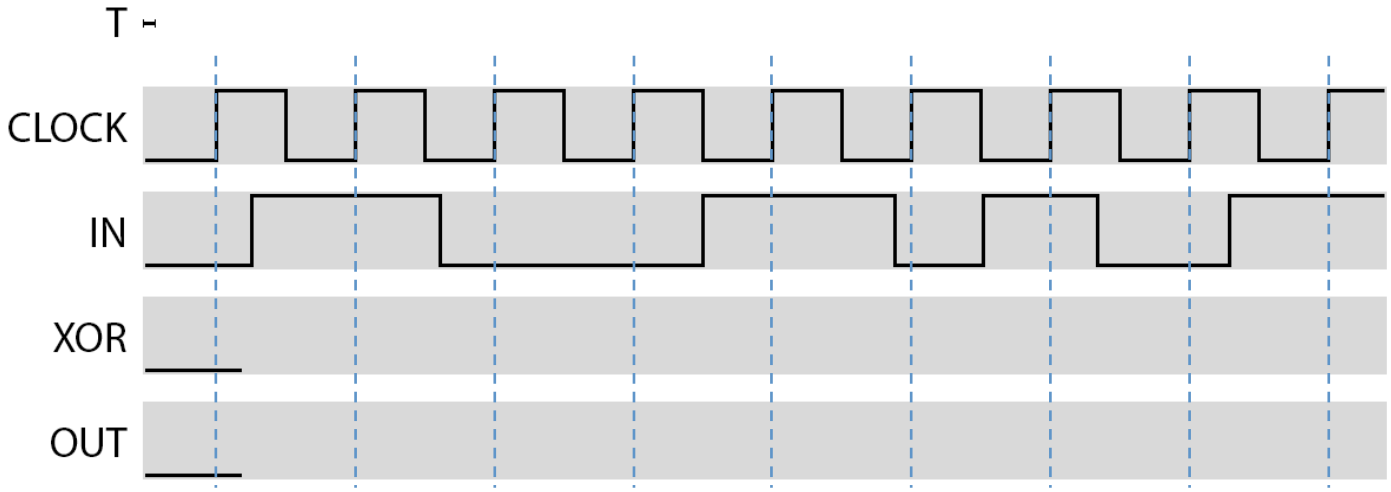


b) Complete the following finite state machine whose input takes on the value of 'X', 'Y', or 'Z'. The machine should output a 1 *only* if it has seen *either* an odd number of Xs *or* an odd number of Ys. Assume you've seen no Xs or Ys at the start state. Otherwise, the machine should output a 0. Add no new states. The labeling of the states as (00, 01, 10, 11) is arbitrary. We've added the first transition arrow for you, which is taken if you've seen an X.



c) Consider the following circuit below. Assuming (1) the signals all start at 0, (2) the XOR gate has no propagation delay, and (3) the register setup, hold, and delay are all given by the small approximate length of T (about 1/10th the clock period), fill out the following timing diagram.





SID: _____

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

For the following questions, assume the following:

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

a) How many virtual pages are there per process? **64**

b) How many bits wide is the *page table base register*? **19**

For questions (c) and (d), assume only the code and the two arrays take up memory, the arrays are distinct, ALL of code fits in 1 page, the arrays are page-aligned (start on page boundary), and this is the only process running.

```
char *mystrcpy(char *dst, char *src)
{
    char *ptr = dst;
    while(*dst++ = *src++);
    return ptr;
}
```

c) If `mystrcpy` were called with a character string of length `S`, how many page faults can occur in the worst-case scenario? **2*ceil(S/1024)**

d) In the best-case scenario, how many iterations of the loop can occur before a TLB miss? You can leave your answer as a product of two numbers. **7 Ki**

For the next three statements, circle **True** or **False**:

e) [**True** / ➔**False**←] RAID was invented at Cal as a way to decrease the number of disk failures in a system.

f) [➔**True**← / **False**] RAID 1 is the most expensive RAID configuration but it offers very high availability.

g) [➔**True**← / **False**] Writing to disk on RAID 1 is faster than on RAID 5.

SID: _____

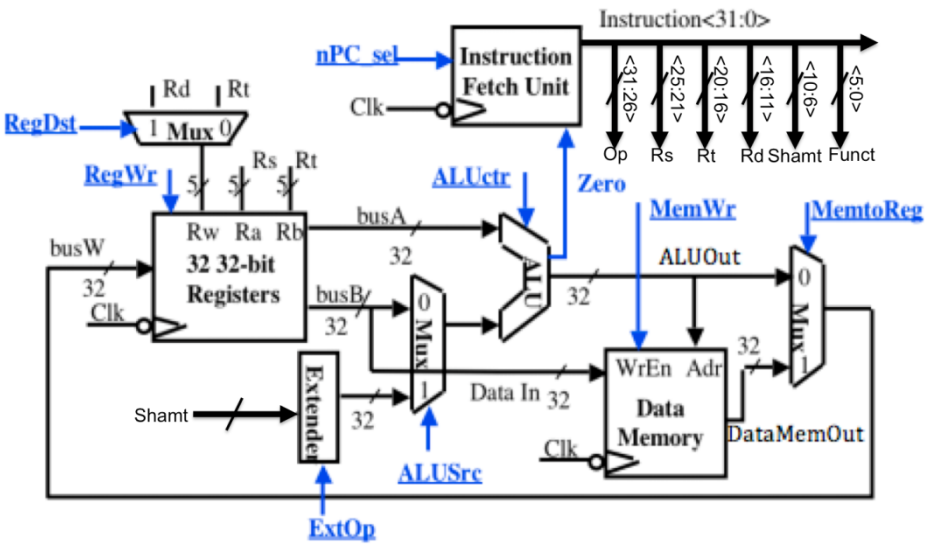
F3) *Datapathology* … (22 pts, 30 mins)

Consider the single cycle datapath as it relates to a new MIPS instruction, store shift left logical:

`ssll rd, rs, rt, shamt`

The instruction does the following:

1) Reads the value of register `rt` and shifts it left by `shamt` bits.
2) Stores the result into memory location `R[rs]` **as well as** register `rd`.



Ignore pipelining for (a)-(c).

a) Write the Register Transfer Language (RTL) corresponding to `ssll rd, rs, rt, shamt`

R[rd] ← (R[rt] << shamt); Mem[R[rs]] ← (R[rt] << shamt); PC += 4

b) Change as *little as possible* in the datapath above **(draw your changes right in the figure)** to enable `ssll`. List all your changes below. Your modification may use muxes, wires, constants, and new control signals, but nothing else. (You may not need all the provided boxes.)

(i)	**Add a mux to select between busA and ALUOut for Data Adr.**
(ii)	**Add a mux to select between busB and ALUOut for Data In**
(iii)	**Add a mux to select between busA and busB for ALUupperin**

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

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg	DataAdr	DataIn	ALUUpIn
1	**1**	**"+4"**	**zero**	**1**	**"shift"**	**1**	**0**	**"busA"**	**"ALUOut"**	**"busB"**

For questions (d)-(f), Assume that you have a 5-stage pipeline with no forwarding, no interlock, and no branch delay slots. Read the code below and answer the following questions.

```
BRANCH: ssll $t0, $t1, $t2, 5
        lw $t3, 0($t1)
        addi $t5, $t1, 11
        beq $t3, $t1, BRANCH
```

d) What kind(s) of hazards exist in these lines of code? _____**Control, Data**_____

e) How many stalls are needed to resolve them? _____**4**_____

f) If you were to have branch delay slots but no forwarding nor interlock, what is the optimal number of cycles one iteration of this code would take? _____**8**_____

SID: _____

F4) *What do you call two L's that go together?* (22 pts, 30 mins)

We will drop the lowest score of parts (a), (b), (c) and (d) below, each are worth 4 points.

We've done a lot of work throughout the semester with dot products (also called inner products), let's switch things up and take some outer products. Where the dot product between two column vectors is defined to be $x^T y$, the outer product between two is defined to be xy^T. If we let $O = xy^T$, then it's straightforward to see that $O_{ij} = x_i y_j$. Our goal in this problem is going to be to try to parallelize this computation in a couple of different ways, starting from the following source code:

```
void outer_product(float* dst, float *x, float *y, size_t n) {
      for (size_t i = 0; i < n; i += 1)
            for (size_t j = 0; j < n; j += 1)
                  dst[i*n + j] = x[i] * y[j];
}
```

a) Using the openMP directives we learned in lab and lecture, parallelize **`outer_product()`**. You should optimize performance while still guaranteeing correctness. You may not need every blank.

```
void outer_product(float* dst, float *x, float *y, size_t n) {
      #pragma omp parallel for
      _____________________________________________
      for (size_t i = 0; i < n; i += 1)

            ____________________________________
            for (size_t j = 0; j < n; j += 1)

                  ____________________________
                  dst[i*n + j] = x[i] * y[j];
}
```

b) Now use SSE intrinsics to optimize **`outer_product()`**. You may find the following useful:

- **`_mm_loadu_ps`**`(__m128 *src)`
- **`_mm_storeu_ps`**`(__m128 *dst, __m128 val)`
- **`_mm_load1_ps`**`(float *src)`
- **`_mm_mul_ps`**`(__m128 a, __m128 b)`

You may assume that `n` is a multiple of 4 for this part of the problem.

```
void outer_product(float* dst, float *x, float *y, size_t n) {
      for (size_t i = 0; i < n; i += 1)
            for (size_t j = 0; j < n; j += 1)
                  __m128 a = _mm_load1_ps(&x[i]);
                  ____________________________________________________
                  __m128 b = _mm_loadu_ps(&y[j]);
                  ____________________________________________________
                  __m128 products = _mm_mul_ps(a, b);
                  ____________________________________________________

                  _mm_storeu_ps(&dst[ i*n + j ], products);

                  j += 3;
}
```

SID: _____

F4) *What do you call two L's that go together?* (Continued)

c) Now write a CUDA kernel to compute outer products

```
__global__ void outer_product_kernel(float *dst, float *x, float *y, size_t n) {
      size_t j = threadIdx.x+blockDim.x*blockIdx.x;
      size_t i = threadIdx.y+blockDim.y*blockIdx.y;
                  i < n && j < n
      if (_____________________________________________) {
                  dst[i*n + j] = x[i] * y[j];
            ____________________________________________________________
      }
}
```

d) Finally, we're going to use MapReduce to generate outer products. We're going to assume the existence of a Tuple datatype that has a constructor taking in two Longs (or LongWritables), and creates a pair from the two of them. Also assume that vectors are represented as a collection of KV pairs mapping from scalar indices to values, while matrices are represented as KV pairs mapping tuple indices to values. Complete `Map` so that it will work correctly with `Reduce`. You may call `getN()` at any time to grab the length of the vectors being processed.

```
void Map(LongWritable idx, DoubleWritable val) {
                        z < getN()              z++)
      for ( long z = 0 ; ___________________ ; ___________________ ) {
            if (is_x_vector()) // magically detects if we're currently processing a value from x
                                    new Tuple(idx, z), val
                  context.write( ____________________________________________ );
            else
                                    new Tuple(z, idx), val
                  context.write( ____________________________________________ );
      }
}

void Reduce(Tuple idx, Iterable<DoubleWritable> vals) {
      context.write(idx, vals.next() * vals.next());
}
```

e) Which of these approaches would you expect to operate the most efficiently on medium sized inputs (order 1000 elements in each vector)? Assume that we're using hive machines, or an Amazon™ cluster like the one used in project 2 in the case of mapreduce. Give a brief 1-2 sentence explanation of why the other approaches would fare worse.

OpenMP would probably fare best – 4 way SIMD give less speedup than 16 threads on easily parallelized problems like these, and the data is too small for MapReduce or the GPU to perform well.

f) Which of these approaches would scale to the largest input before computing incorrect answers, or failing to compute answers? Which would fail on the smallest input? Explain in 1-2 sentences.

MapReduce would fail latest, as it automatically stores the data as files rather than in memory. The GPU would fail first, given its limited memory space.