CS61C Summer 2012 Final Review Solutions

a) Fill in the blanks of this C code to write a '1' to *every byte of a single 2 KiB page on the heap,* don't write anything to any other pages. You may assume that there are a few available heap pages. Use as little memory as possible (you might need to ask for more than 2 KiB). memset is not allowed, you can't allocate anything already allocated on the heap, and two consecutive memory requests may not be near each other.

```
#define PAGE 0x800// 2 KiB in hex TouchEveryPageByte() {

uint8_t *ptr, *tmp;
ptr = (uint8_t *) malloc (2 * PAGE);     // one C statement here                    tmp =
(ptr | (PAGE-1)) + 1;    // one C statement here
for (int i = 0; i < PAGE; i++)
        *(tmp+i)   = 1;
}
```

b) Here are 3 different numerical encodings of 32 bits.

Float	Fixed point	Rational
float	XXXXXXXXXXXXXXXX.YYYYYYYYYYYYYYYY ...where Xs are interpreted as 2s complement, Ys are interpreted the standard way bits on the right of a fixed-point representation are interpreted.	NNNNNNNNNNNNNNNNDDDDDDDDDDDDDDDD ...where Ns are interpreted as a biased numerator (the bias is set in the usual way so that roughly half the numerators are positive, half are negative), and Ds as an unsigned denominator. A denominator of 0 means infinity; if both num and denom are zero it's a NaN. This # basically looks like: NNNNNNNNNNNNNNNN <- biased ---------- ---- DDDDDDDDDDDDDDDD <- unsigned

	Float	Fixed	Rational

Distance from “A” of first letter in name	1	1	2
Fewest # of letters in name	1	3	2
Fewest # of zeros	2 (2)	1 (1)	3 (2^16-1)
Smallest positive number	1(2^-149)	3(2^-16)	2(1/(2^16-1))
Closest representable number to -1/3	2 (23 bits of precision)	3 (16 bits of precision)	1 (exact)
Actual number farthest to the left on the number line	1 (about -2^127)	3 (almost -2^15-1)	2 (-2^15 - 1)

4 F SDS Feeling Gray?

1) Gray codes are a different way of ordering binary numbers, such that successive numbers only differ by 1 bit. Gray codes are very valuable in error correction for digital communication. As an example a four bit Gray encoding would be:

0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000

We want to implement a finite state machine that tells whether two two digit numbers are adjacent Gray code numbers: for example 01 and 11 or 00 and 10 but not 11 and 00 or 10 and 10. We will treat the incoming bit as the new most significant bit of a new three digit number composed of it and the 2 digits seen before it (assume all zeros for 'negative' time, time before we start receiving inputs). We will then break these two digits into two two digit numbers by using the middle bit twice. As in if the three bits are ABC then the two numbers are AB and BC. The FSM should output a 1 if these numbers are adjacent Gray code numbers. So for example if we see the bitstream 11010,

Input	Last Three Bits Seen	Numbers	Adjacent Gray Code Numbers?	Output
1	100	10 and 00	Yes	1
1	110	11 and 10	Yes	1
0	011	01 and 11	Yes	1
1	101	10 and 01	No	0
0	010	01 and 10	No	0

We assume initially we have zeros and when we see the first 1 we use it as the MOST SIGNIFICANT BIT so we have 100 which splits to 10 and 00 which are adjacent numbers. Now we shift all the bits over by 1 and append the next input, which is again a 1 and hence the number is now 110 which splits to 11 and 10 and we output a 1 because these too are adjacent numbers.

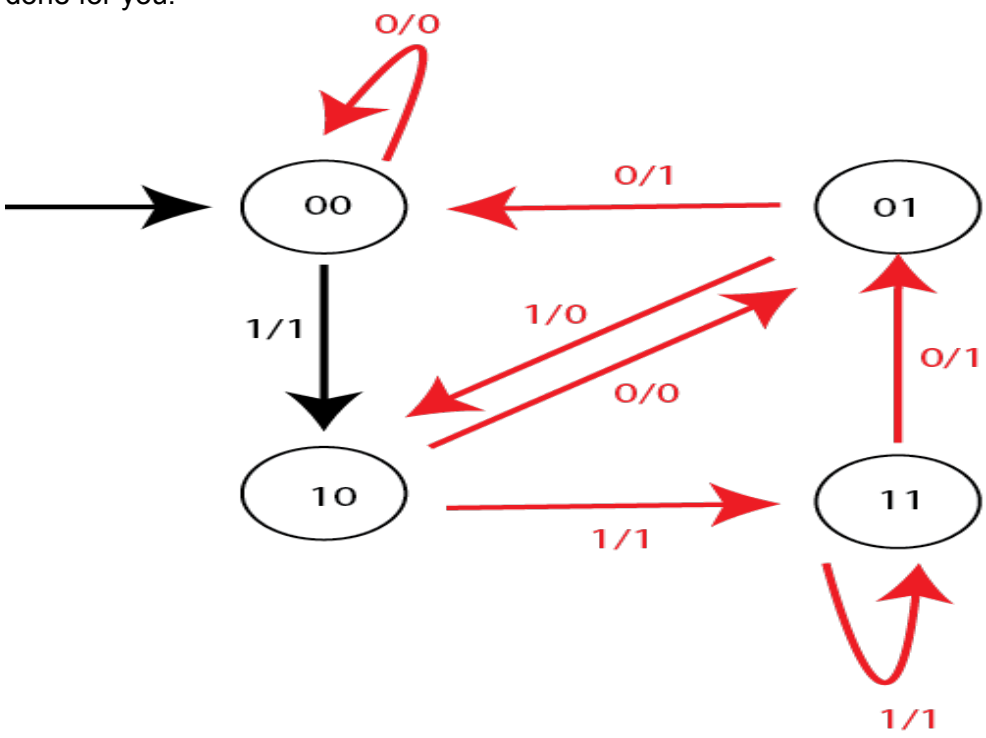
a) First fill in the below truth table, the next states have been done for you, all you need to fill in is the Output column. Note the three last bits are [In][Current S2][Current S1] in that order. So your job is to determine if [In][Current S2] and [Current S2][Current S1] are adjacent gray code numbers.

Input	Current S2	Current S1	Next S2	Next S1	Output
0	0	0	0	0	0
1	0	0	1	0	1
0	0	1	0	0	1

1	0	1	1	0	0
0	1	0	0	1	0
1	1	0	1	1	1
0	1	1	0	1	1
1	1	1	1	1	0

b) Now for the FSM. We will use 4 states to keep track of the 2 most recently seen bits. These have been drawn in for you.

Your job is to add the transitions and mark the input and output bits on each transition. One is done for you:



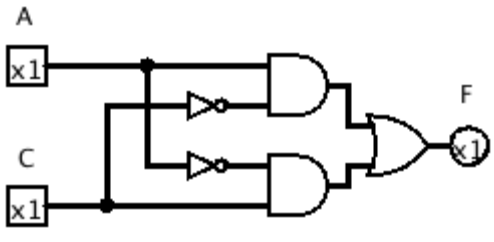
c) From the table in part (a) write an expression for F in boolean algebra. Call Input: A, Current S2: B, Current S1: C (hint: use sum of products):

F = (A)(~B)(~C) + (~A)(~B)(C) + (A)(B)(~C) + (~A)(B)(C)

d) Now simplify to use as few 2 input `and` and `or` gates as possible (use as many `not` gates as you like, and no other gates):

F = (A)(~C)(B + ~B) + (~A)(C)(B + ~B) = (A)(~C) + (~A)(C)

e) Now draw the gates that correspond to the above table and equations:



f) If we allow you to use any 2 input logic gate is there a way you could simplify part (d) to use just one gate (You cannot use additional `not` gates). If yes what is it?:

No Yes: You can use the _XOR_ gate

Datapath

On the right is the single cycle datapath you all know. Your job is to modify the diagram to support this new MIPS instruction, save and duplicate:

```
sdup rt, rs, imm
```

The instruction does the following:
1) stores the value in `rs` into memory at the address stored in $sp offset by `imm`.
2) copies the value in `rs` into `rt`.

In short, it copies `rs` into `rt` as well as onto the stack.

Ignore pipelining for parts a and b.

a) Change as *little as possible* in the datapath above **(draw your changes right in the figure)** to enable `sdup` and list all changes below. Your modification may use muxes, wires, constants, and new control signals, but nothing else. You may not need all boxes. You may not need all boxes. Assume the output of Data Memory is unknown it's Write Enabled.

(i)	Add mux to select either Rs or $sp (ie 29) for Ra.
(ii)	Add mux to select either Rs or Rt for Rb.
(iii)	Instead of the current busW, add a mux to select between busB and busW.
(iv)	A control signal sdup to control all of the above muxes.
(v)	

b) 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. Don't allow `sdup` to write off the stack.

RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUctr	MemWr	MemtoReg	sdup		
0	1	+4	Zero	Imm	add	1	X	1		

c) Now consider the 5-stage MIPS pipeline. Consider the following sets of instructions independently. Circle each set that poses a data hazard without forwarding. For each of the ones you circle, give methods to solve the hazard. For stalls, give the number of cycles to stall. For forwarding, state after which stage the data is available and at which stage the data is needed. You should minimize the stalls necessary for each case.

i)
```
add $sp $sp -8
sdup $t0 $a0 0
```
Forward $sp's value after the add's ALU stage to sdup's ALU stage.

ii)
```
add $t0 $t0 $t1
sdup $t0 $a1 4
```
No data hazards.

iii)
```
lw $t0 0($a1)
sdup $t1 $t0 0
```
Forward $t0's value after lw's Mem stage to sdup's Mem stage.

Virtual Memory

Assume:
1 GiB of physical memory
32 bit virtual addresses
Pages of size 2KiB

a) How many bits is the virtual page number? 32 - 11 = 21

b) How many bits is the physical page number? 30 - 11 = 19

c) How many entries does a page table contain? 2^21

Consider the following loop. `arr` is an int array of size `ARRAY_SIZE`. `SKIP` is a positive integer less than the page size.

```
for(int i = 0; i < ARRAY_SIZE; i+=SKIP) {
      total += arr[i];
}
```

32 page faults were detected while running the loop.

d) Assuming that page faults only occurred from loading elements of `arr`, what is the smallest number `ARRAY_SIZE` can be?
2+30*2^9

e) The TLB has four entries, a block size of one entry, and an LRU replacement policy. In the middle of running the loop, another process was given CPU time but an entry of the TLB didn't flush during the switch for some reason. The entry was for a page full of `arr` elements. Both processes continued fine but `total` was incorrect at the end. Why might `total` be incorrect? At worst case, how many incorrect values were added to `total`?
The other process might have performed writes to memory with the same virtual address as some of the array elements on that page. 512B/SKIP

f) Assume we are using RAID 1.
In the best case, how many pages would the loop have written to the disks?
0

In the worst case, how many pages would the loop have written to the disks?
64

F4) *http://inst.eecs.berkeley.edu/~parallel* ... (22 pts, 30 mins)

Berkeley is looking at opening CS61C to the masses - but the glookup service is far too slow! In addition to ordering more servers, the inst staff have asked you to look at optimizing the glookup service. They've already determined that spinlocks are a performance issue, but there's no way to replace them. Perhaps you can make the locks themselves faster, though! You identify two possible definitions for a spinlock:

```
typedef struct spinlock {
char value; // 1 if the lock is currently held else 0

} spinlock_t;

typedef struct spinlock_padded {
char value; // 1 if the lock is currently held else 0
char padding[63]; // 63 bytes of padding (never used)

} spinlock_padded_t;
```

a1) Implement the generic spin_lock and spin_unlock functions in C, which will be called on pointers to spinlock_t and spinlock_padded_t. You use the following test-and-set function:

int CAS(char *dest, char test, char value) CAS compares the values of *dest and test:
If they're equal, *dest is set to value, and CAS returns 1
If not, CAS returns 0, changing nothing in the heap State any assumptions you need to make.

```
void spin_lock(void *lock) {
        while(CAS(&(lock->value), 0, 1));
}

void spin_unlock(void *lock) {
        lock->value= 0;
}
```

You evaluate the performance of both types of spinlocks using two different parallelism benchmarks. They are both locked and unlocked using the same generic spin_lock and spin_unlock procedures.

a2) On one benchmark, the padded spinlock worked faster. Why might this be? Describe what the program could be doing to cause this.
False sharing (ping ponging).

a3) On the other benchmark, the unpadded spinlock worked faster. Why might this be? Describe what the program could be doing to cause this.
Less spatial locality. Cache blocks with locks only frequently accessed by single thread.

Part of the glookup upgrade includes some convenient and speedy statistical reporting. The system has *exactly one* record for each student of the following format:

(Student Name, Student's Advisor's Name, Student's Course Load)

where “course load” encodes the number of credits the student took in the most recent semester. To get a sense of which faculty are pushing students to take more or fewer classes, the administration have asked inst to ask *you* to write a MapReduce scheme for finding, for each advisor, the *greatest* course load any of that advisor’s students are taking. The output record should resemble: (Advisor Name, Max Course Load)

b1) Describe the process by which your mappers would read an input record and produce fodder for the reducers. Please explicitly include the (key, value) pairs your map() function would emit.

(key, value): (Student’s Advisor’s Name , Student’s Course Load)

b2) Describe how your reducers would transform the pairs received from the mappers into the output records.

(Student’s Advisor’s Name, max of Student’s Course Load)