M3) Got some numbers, down by the C... (10 pts, 20 mins)

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.

```c
#define PAGE 0x__________ // 2 KiB in hex

TouchEveryPageByte() {
    uint8_t *ptr, *tmp;

    __________________________ // one C statement here
    __________________________ // one C statement here

    for (int i = 0; i < PAGE; i++)
        __________________________ = 1;
}
```

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

<table>
<thead>
<tr>
<th>Float</th>
<th>Fixed point</th>
<th>Rational</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>XXXXXXXXXXXX.YYYYYYYYYYYYYYY</td>
<td>NNNNNNNNNNNNNNDDDDDDDDDDDDDDDD</td>
</tr>
<tr>
<td></td>
<td>...where xs are interpreted as 2s complement, ys are interpreted the standard way bits on the right of a fixed-point representation are interpreted.</td>
<td>...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:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NNNNNNNNNNNN &lt;- biased</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-----------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDDDDDDDDDDDDDD &lt;- unsigned</td>
</tr>
</tbody>
</table>

Rank these three in terms of the following categories (ties are allowed). We did the first two for you.

<table>
<thead>
<tr>
<th>Category</th>
<th>Float</th>
<th>Fixed Point</th>
<th>Rational</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance from &quot;A&quot; of first letter in name</td>
<td>1st</td>
<td>1st</td>
<td>2nd</td>
</tr>
<tr>
<td>Fewest # of letters in name</td>
<td>1st</td>
<td>3rd</td>
<td>2nd</td>
</tr>
<tr>
<td>Fewest number of zeros</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smallest positive number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Closest representable number to -1/3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actual number farthest to the left on the number line</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6 Who invited these people again?

Louis Reasoner writes the following self-modifying code:

```plaintext
foo:   la $t0, modify
     sll $a0, $a0, 11
     lw $t1, 0($t0)
     addu $t2, $t1, $a0
     sw $t2, 0($t0)
modify: addu $t0, $0, $a1
       sw $t1, 0($t0)
       jr $ra
```

What happens if we call `foo` with $a0=15 and $a1=15? (3pt)

For each of the following questions, CIRCLE either a, b, c, d, or e.
Which set of inputs will permanently change the behavior of the program? (3pt)
   a) $a0=3 and $a1=3
   b) $a0=5 and $a1=5
   c) $a0=7 and $a1=7
   d) $a0=9 and $a1=9
   e) $a0=11 and $a1=11

Which set of inputs will always cause a bus error? (3pt)
   a) $a0=7 and $a1=7
   b) $a0=8 and $a1=7
   c) $a0=7 and $a1=8
   d) $a0=8 and $a1=8

Alyssa P. Hacker has figured out how `foo` works and wants to use the program for an unintended purpose – copying $t3 to $t4. Which set of inputs should she choose? (She could have just executed addu $t4 $0 $t3 to achieve the same effect). (3pt)
   a) $a0=0x000000CB and $a1=0x000000CB
   b) $a0=0x000000CC and $a1=0x000000CC
   c) $a0=0x000000CD and $a1=0x000000CD
   d) $a0=0x000000DB and $a1=0x000000DB
   e) $a0=0x000000DC and $a1=0x000000D
5/8

F1) Code, Earl Gray, Hot... (22 pts, 30 mins)

Bits can mean anything! Gray codes are a way of ordering the binary encoding of symbols/values such that successive symbols/values only differ by exactly one bit in their representation. (Gray codes are useful for error correction in digital communication!) As an example, here's one of many possible valid orderings of the four-bit Gray codes (with the consecutive differing bits underlined):

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

Any cycle of this sequence will also be a valid ordering, just so long as no code differs more than one bit from its neighboring codes. When two binary representations differ by exactly one bit, we call the pair Gray adjacent.

a) Annotate this cube such that each corner is Gray-adjacent to its 3 neighbors. (Corner a is given.)

a 000 b ___ c ___ d ___ e ___ f ___ g ___ h ___

Now imagine a one-bit stream of data, from which we’re constantly assembling two-bit values. We use the bit received at time t, b_t, in conjunction with the previously streamed-in b_{t-1} to make the code “b_t b_{t-1}”. Note that the most recent bit is the more significant. We want a finite state machine that indicates whether the two most recent two-bit values – that is to say “b_t b_{t-1}” and “b_{t-1} b_{t-2}”, are Gray code adjacent. (Take two zeros as the initial condition of the stream – if we stream in bit b_0 at t = 0, the three most recent bits we’ve seen will be taken as “b_000”, giving us the values “b_0” and “00”.) Here’s an example for the bit-stream 1101 (with our initial condition bits bracketed):

<table>
<thead>
<tr>
<th>Input</th>
<th>Most recent 3 bits</th>
<th>Most recent codes</th>
<th>Adjacent?</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1[00]</td>
<td>10 and 00</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1[10]</td>
<td>11 and 10</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>01</td>
<td>01 and 11</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10 and 01</td>
<td>No</td>
<td>0</td>
</tr>
</tbody>
</table>

b) Let's presume a state variable such that the three most recent bits are Input, (Current) S_2, (Current) S_1 (telling us that the next state bits must be Input, (Current) S_2). Fill in the Output column of the table.

c) Add the state transition arrows, along with the input and output bits on each transition, to the state diagram below. One is done for you, as well as an arrow for the 00 initial condition:

a) Give the simplest Boolean expression for Output in terms of Input, (Current) S_2, and (Current) S_1. Show your work.
Consider the single cycle datapath as it relates to a 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`.

**Ignore pipelining for parts (a)-(c).**

a) Write the RTL corresponding to `sdup rt, rs, imm`

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

<table>
<thead>
<tr>
<th>(i)</th>
<th>(ii)</th>
<th>(iii)</th>
<th>(iv)</th>
</tr>
</thead>
</table>


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. Don’t allow `sdup` to access any memory below the stack pointer.

```
<table>
<thead>
<tr>
<th>RegDst</th>
<th>RegWr</th>
<th>nPC_sel</th>
<th>ExtOp</th>
<th>ALUSrc</th>
<th>ALUctr</th>
<th>MemWr</th>
<th>MemtoReg</th>
</tr>
</thead>
</table>
```

d) Now consider `sdup` run on the 5-stage MIPS pipeline. Consider the following pairs of instructions (i) through (iii) independently. Circle each pair that poses a data hazard.

- For each circled pair, explain how to resolve the hazard.
- If stalling is needed, provide the number of stall cycles required. (Please resolve the hazard with a minimal number of stall cycles.)
- If forwarding is needed, state after which stage the data is available and at which stage the data is needed.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>add $sp $sp -8</td>
<td>add $t0 $t0 $t1</td>
<td>lw $t0 0($a1)</td>
</tr>
<tr>
<td>sdup $t0 $a0 0</td>
<td>sdup $t0 $a1 4</td>
<td>sdup $t1 $t0 0</td>
</tr>
</tbody>
</table>

7/8
F2) *(I/O)rtual Potpourri* ... (23 pts, 30 mins)

Consider a system with 1 GiB of physical memory, a 32-bit virtual address space, and 2 KiB pages.

a1) How many bits to uniquely identify each page of virtual memory? ________ Physical memory? ________

a2) How many entries does the page table contain? ________

The following code runs on the above system. `arr` is an integer array of size `ARRAY_SIZE`. `SKIP` is a positive integer less than or equal to the page size. 32 page faults were detected while running the loop, all due solely to reading from `arr`.

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

b1) What’s the smallest possible value of `ARRAY_SIZE`? ________

b2) The TLB for this system has four entries, each listing a single VPN-PPN pair, and it uses an LRU replacement policy. In the middle of running the above code, another process was given CPU time – but for some reason an entry of the TLB didn’t flush during the switch to this process. This unflushed entry corresponds to a page filled with elements of `arr`. Both processes executed without raising an exception. Why might `total` be incorrect? At worst, how many incorrect values were added to `total`?

b3) Assume we are using RAID 1 for this system’s disk, and the above scenario unfolds.

   How many pages on disk would have been modified in the best case? ________ Worst case? ________

c) In which situations might some form of DMA be used? Check all that apply.

   Handling a page fault ________
   Communication between servers in a datacenter ________
   Playing music from a file already in memory ________
   Scrolling through a (fully downloaded) web page ________

d) Explain (in the provided space) how, why or when the statements below are true:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polling can be faster than interrupt-driven I/O.</td>
<td></td>
</tr>
<tr>
<td>High frequency polling is undesirable.</td>
<td></td>
</tr>
<tr>
<td>Low frequency polling can make data transfer slow.</td>
<td></td>
</tr>
<tr>
<td>I/O interrupts can be combined with I/O polling to get the best of both.</td>
<td></td>
</tr>
</tbody>
</table>

e) We have a system to which we can instantaneously add and remove cores – adding more cores never leads to slowdown from things like false sharing, thread overhead, context switching, etc. When the program `foo` is executed to completion with a single core in the system, it completes in 10 minutes. When `foo` is run with a total of three cores in the system, it completes in 6 minutes. How long would it take to execute `foo` on this magical system as the number of cores approaches infinity?


2 Thread Level Parallelism

For the following snippets of code below, Circle one of the following to indicate what issue, if any, the code will experience. Then provide a short justification. Assume the default number of threads is greater than 1. Assume no thread will complete before another thread starts executing.

Assume arr is an int array with length len.

a) // Set all elements in arr to 0
   int i;
   #pragma omp parallel for
   for (i = 0; i < len; i++)
      arr[i] = 0;
   Sometimes incorrect Always incorrect Slower than serial Faster than serial

b) // Set element i of arr to i
   #pragma omp parallel
   for (int i = 0; i < len; i++)
      arr[i] = i;
   Sometimes incorrect Always incorrect Slower than serial Faster than serial

c) // Set arr to be an array of Fibonacci numbers.
   arr[0] = 0;
   arr[1] = 1;
   #pragma omp parallel for
   for (int i = 2; i < len; i++)
      arr[i] = arr[i - 1] + arr[i - 2];
   Sometimes incorrect Always incorrect Slower than serial Faster than serial
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:

```c
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:

```c
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.

```c
void spin_lock(void *lock) {
}

void spin_unlock(void *lock) {
}
```

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.

a3) On the other benchmark, the unpadded spinlock worked faster. Why might this be? Describe what the program could be doing to cause this.

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): ( ___________ , _________________ )
```

b2) Describe how your reducers would transform the pairs received from the mappers into the output records.
Q2  Warehouse Scale Questions (F)

a) Name two reasons to replicate data in a Warehouse Scale Computer.

b) Name one reason why the MapReduce model of programming requires that the Mapper and Reducer functions do not have side-effects (lasting effects beyond creating output as a function of input).

c) Name two ways that multicore systems maintain cache coherence during writes.
5 Synchronous Digital Circuits

a) You are an intern at a massive hardware firm. Your first task is to design a “prime number checker.” The circuit you must design with take as input an integer between 0 and 7. It will have three input bits, corresponding to the relevant integer. Your circuit should output 1 if the integer is prime and 0 otherwise. Neither zero nor 1 is prime.

Complete the Truth Table below. A, B and C correspond the 4’s place, 2’s place and 1’s place respectively.

<table>
<thead>
<tr>
<th>Input A</th>
<th>Input B</th>
<th>Input C</th>
<th>Output X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

b) Write the Boolean expression formed by the output in Sum-of-Products form. Reduce it to its simplest form.

c) Construct the circuit with the fewest gates using only AND, OR and NOT gates.

Consider a system that can output a value between 0-3 with the ability to increment and decrement. This system will have two 1-bit inputs: increment and decrement and a 2-bit output (the count).
- If increment is high, the count should increase by one for the next cycle (wrap around if necessary).
- If decrement is high, the count should decrease by one for the next cycle (wrap around if necessary).
- If neither is high, the system should stay at the same value.
- They will never both be high at the same time.

Draw a state machine for this system with appropriate transitions for each input pair.