CS 3410 Homework 6 NetID_____ Date:_____
Due Tuesday, April 24th, 2012, 11:59pm Name:_____
Note: No slip days are allowed for this homework. This homework must be turned in by the deadline.

1. A) What is the power wall, and what can we do about it?

   B) Suppose we have a program of which 80% can be parallelized, and 20% cannot. If we have 5 processors working in parallel, use Amdahl's law to estimate the speedup we get over 1 processor. What if we have 20 processors? What if the number of processors tends towards infinity?

   C) List three events that can cause a process to transition from user mode to kernel mode. One of these events should be an interrupt, one of these events should be caused by an internal error, and one of these events should be caused internally, but should not be an error.
   Note: Here, assume that the term "interrupt" is *not* synonymous with the term "exception."

2. Suppose the programmers over at NoobTech Inc. are writing a synchronization library for use by concurrent programs. This library includes an atomic increment function atomicInc. This function takes two arguments, add0 and lock0, in that order. The argument add0 contains the address of the integer to be incremented, while the argument lock0 contains the address of that resource's lock (see Section 2.11 of textbook). Note that a lock value of 0 indicates free and a lock value of 1 indicates locked.
   The programmers wrote the function as follows:

```
atomicInc:
addi $sp, $sp, -24
sw $ra, 20($sp)
sw $fp, 16($sp)
addi $fp, $sp, 20

lockloop:
lw $t0, 0($a1)
bne $t0, $zero, lockloop
nop
addi $t1, $zero, 1
sw $t1, 0 ($a1)

lw $t2, 0($a0)
addi $t2, $t2, 1
sw $t2, 0($a0)

sw $zero, 0($a1)

lw $ra, 20($sp)
lw $fp, 16($sp)
addi $sp, $sp, 24
jr $ra
nop
```

   a) This function will not work correctly. What is wrong with it?

b) Rewrite this function using the ll and sc MIPS instructions so that it works correctly. Don't worry about optimization.

3. Suppose we call in our program a LW instruction. We do not call this with a physical address, but with a virtual address. A series of events then occur to deliver the requested information to the processor.
   In answering the questions below, use only events listed in the following table. Some events will need to be used more than once; some will not need to be used at all.
   Note 1: Although the TLB is itself a cache, when we use the term "cache" in this problem, we refer to main memory's cache, not the TLB.
   Note 2: Assume that we have one level of caching and that it is addressed with physical addresses.

a)	Cache hit; memory contents retrieved	**b)**	Cache miss; stall pipeline while memory retrieves data
c)	TLB hit; entry retrieved	**d)**	TLB miss; OS TLB miss handler invoked
e)	Page table entry is found valid, so request data from memory	**f)**	Page table entry is found not valid; OS exception handler invoked
g)	Page brought into cache	**h)**	Page brought into main memory
i)	Handler retrieves page table entry and places it in the TLB	**j)**	Handler retrieves page table entry and places it in cache
k)	Memory gives data to pipeline and places it in cache	**l)**	Memory places data in cache
m)	TLB loads data from main memory	**n)**	TLB loads data from disk

   a) In the *best case* scenario, write down the set of events that occur, and in the correct order. Some of these events have been filled in for you.

      1. Execute LW instruction
      2.
      3.
      4.
      5. Execution continues

   b) Now consider the *worst case* scenario. This is particularly slow, as a page fault will occur. Why will the entries in the cache and TLB likely be different after the page fault has been resolved?

c) Write down the set of events that occur in the *worst case scenario*, and in the correct order. Assume for simplicity that no exceptions or interrupts external to this process occur. Some of these events have been filled in for you.

1. Execute LW instruction
2.
3.
4. Execute LW instruction
5.
6.
7.
8. Execute LW instruction
9.
10.
11. Execute LW instruction
12.
13.
14.
15.
16. Execution continues

4. In this problem you will implement another famous, ancient algorithm. This one is known as the Sieve of Eratosthenes, and computes all the prime integers between 2 and a given number n. You will write MIPS assembly code that will be executed in a MIPS simulator called QtSpim. You can download QtSpim at http://sourceforge.net/projects/spimsimulator/files/. It should work right out of the box. If you use the GUI-based version, under Simulator->Settings->MIPS tab, under the MIPS Simulation Settings box, make sure Enable Delayed Branches and Accept Pseudo Instructions are checked and the other three unchecked. If you use the command line-based version, run with the -delayed_branches flag.
Download sieveoferatosthenes.s from CMS, and load it into QtSpim using File->Reinitialize and Load File. Run the assembly file under Simulator->Run/Continue, or by hitting F5. You should see a message pop up on the console indicating that the code is working.

The code in sieveoferatosthenes.s prompts the user for an integer, and then calls the function sieve with that integer as an argument. sieve then prints out all prime numbers from 2 to n.

You are to implement sieve (replace the code that is currently there, it's just filler). The pseudocode that you are to translate to MIPS is the following.

```
sieve(n) {
arr = malloc(n+1);

initialize arr[2] to arr[n] as true

for p = 2 to n
  if arr[p]
    for (k = 2*p; k <= n; k+= p)
      arr[k] = false;
    endfor
  endif
endfor


printindices(arr, n+1, 2)
}
```

Note that you MUST use proper calling conventions. This is not about optimization; even if you know something won't be used, if it's part of the conventions, you must implement it.

Submit your file sieveoferatosthenes.s in CMS. We should be able to run your file in QtSpim and produce outputs like the following:

```
Enter the largest integer to check for primality: 11
Allocating a block of 48 bytes.
Prime numbers: 2 3 5 7 11
```

No errors should pop up during execution, but if some of the registers on the left turn red at the end, don't worry about it.