Homework #6
Architecture Design for Performance
Due Friday, December 7th

Your mission is to design an efficient computer that sorts numbers. You have (almost) complete flexibility to design the sorting algorithm, the instruction set architecture, and the computing engine. In fact, you even get to decide what is meant by "performance" architecture design. The purpose of this assignment is to force you to think about the fluidity and continuum of software and hardware. Too often in the hardware world we become trapped, conceptually, in believing that software must be broken down into little bits of register and memory transfers. As well, too often in the software domain we think in terms of the same old paradigm of software as recipes for a fixed serialized execution core.

I encourage you to think outside this box.

In this assignment you are to merge the software and hardware design process. For this you will work in a team of two. But, as in industry, you will be expected to do the job of at least four. The four are (1) algorithm designer, (2) software implementer, (3) hardware architect, and (4) hardware implementer. I strongly encourage you not to divide the labor cleanly along the software / hardware interface. The most excitement will come from the synergy between these two.

Your projects will be graded on whether or not they work. But, the teams with the three best "performing" (working) designs will get an extra credit bonus of 20% and an edible prize. In addition they will have the opportunity to present/discuss their design in class (hopefully, this is as time permits). The performance metrics will be:

The fastest design (wall clock time)
The smallest design
The fastest/area design

All of these metrics will be measured with the latest SMOK, with the smok.ini file from the website.

So, with that pre-amble, some ground rules:

- Your design must have a data cache. It can be direct mapped, set-associative, or whatever. It can be any size. It just must have a data cache, of some size (observe the SRAM limit --- see below), it must use this cache, and the cache must work. You may want to use two caches, one for instruction and one for data [that is if you use instructions at all!].

- You may not use any more than 3500 words of SRAM. You can split this into as many SRAM SMOK blocks that you like though. This counts registers' bits as well! Be sure to add up all of the register bits in your design (then divide by 32), add the SRAM words and make sure it is less than 3500. This is a hard limit.

- Your sorting engine and algorithm will be tested on a small data set, but it must be programmable and able to handle a dataset as large as 512 megabytes (if we modify the DRAM to be that large in your design that is... you should use smaller DRAMs --- see below).

- The dataset will be stored in a DRAM SMOK block of size 8192 words. You will be given a sample dataset to test against, but another dataset will be used for performance analysis. You should not assume anything about the distribution of numbers used for sorting (nor its size; do not hard code the algorithm to a fixed size or assume the size is a power of two).

- The numbers will be in 32 bit words.

- You must replace the smok.ini file with the one from the class website.

- Interface to the DRAM in a special way. Assume that the DRAM requires a delay of 8000 + 1000*W to transfer data to or from it. To access the DRAM you should send it an address and then not assume the data you get back (or write) is valid for 8000+1000*W delay ticks. In other words you send it an address and then you must start a counter that waits for N cycles. To calculate N use this formula: N = Ceil [ (8000+1000*W) / MaxPath - W]. What this formula does is use the latency (8000) and adds on the transfer time (1000*W) and then divides that by the MaxPath of your circuit. It then subtracts out the transfer time for the data you will be reading from the DRAM. The net result is after you transfer W words it will have cost (8000+1000*W) / MaxPath cycles. The max W you can transfer at one time is 256 (1024 bytes). You probably want to think about good choices of W that interface well with your cache.

- After sorting, be sure that your machine halts.

Those are the only rules. Beyond that you get to change anything you want!

Some notes:

The dataset will be stored at offset 0 in the DRAM block. It will look like:

0: size: 1 word denoting the size of the array
4:..(size+1): array: an array of [size] words that you must sort:

Your job is simply to sort it and leave the result in place. You can use as much DRAM as you like for scratch, etc. Just adhere to the SRAM limitation.

If this assignment seems vague, it is intentionally so. I want you to pour your natural creativity into the solution you come up with. Just as a sanity check though, you can imagine building a machine like this:

Data DRAM <----> Cache ----> MIPS like core from HW4 <------- Instruction DRAM store

After building the above write a simple sorting algorithm (say bubble sort). Note that this solution will probably not win the contest but you will get a 100% grade on this homework this way ;-).

To get you started on the range of possible design choices you now face, consider your flexibility with:

instruction set design
pipelining
caches
data-movement
algorithm
software optimization

Have fun, and to the victor go the spoils!

While I encourage you to consider alternative (faster) sorting algorithms (search the net for quick-sort, merge-sort, radix-sort, etc), here is an example of "bubble-sort":

```
bubblesort(int  size, int          *array)
        {
        int       i,j;

        for( j=0; j < (size+1); j++ )
                for ( i=0; i < (size+1); i++ )
                        if (array[i] > array[i+1])
                                swap(array[i], array[i+1]);
        }
```

NOTE: Due to constants, not all algorithms perform at their asymptotic behavior on small data sizes. You can take this into account and adapt your sorting algorithm depending upon the data size. To simplify this contest a little, you can assume that the data size we use for performance testing will be the same data size we hand out for development purposes (just different data).