Question 1:

Download the testmem.c program:

```
wget http://www.ccs.neu.edu/~pjd/csg112/testmem.c
```

You will need to compile it and run it on the following machines:

- **ruchbah (sunblade 100):**
  
  `gcc testmem.c -o testmem-sparc`  
  `/testmem-sparc -m 100 -n 4`

- **swedish (sunblade 1500):**
  
  `/testmem-sparc -m 1024 -n 64`  
  `/testmem-sparc -m 128 -n 8`

- **denali (sunfire 280):**
  
  `/testmem-sparc -n 32 -m 1280`

- **galaga (core2 6600):**
  
  `gcc testmem.c -o testmem-x86`  
  `/testmem-x86 -m 2048 -n 32`

The program measures memory access time, but in doing so it varies the number of active pages of memory. Thus a line like this:

```
152 : 10.744452 ns
```

means that when cycling through 152 pages repeatedly, the average time to read a value from memory is 10.6 ns.

Remember that the TLB holds a certain number of mappings, and that when reading from a memory page that isn't mapped in the TLB, the CPU must walk the page tables in memory. Assume that each memory access takes the same time, whether it is done by the memory management unit to read a page table entry or the CPU itself to read the actual data value. Also note that TLB sizes are powers of 2 on the SPARC machines, and powers of 2 plus a little bit on the x86 machine.

For each computer (ruchbah, swedish, denali, and galaga) provide the output of the testmem program, and answer the following two questions based on that output:

- How large is the TLB for this system, to the nearest power of 2?
- How many levels does the page table have?

If you believe that your measurements are inconclusive, please state so and identify the most likely and next most likely answers based on your reported measurements. Note that I expect answers to be correct according to your measured data, rather than according to the actual specifications of the CPU in question.
**Question 2:**

Here is a portion of the memory map of the homework.c program, comprising 9 pages of memory:

```
08048000-08049000 r-xp  /home/pjd/csg-112/hw1/homework
09000000-09003000 rwxp
bffbc000-bfffc0000 rw-p  [stack]
```

Remember that (on x86) one page = 4096 bytes = 0x1000. Thus the address space consists of 8 virtual pages starting at the following addresses:

```
08048000
09000000, 09001000, 09002000
BFFBC000, BFFBD000, BFFBE000, BFFBF000
```

To map this address space requires 12 physical pages – 8 for the data, 3 for page tables, and one for the page directory. Given 12 physical pages numbered 00000 through 0000B, draw or otherwise describe fully a page table structure mapping this virtual address space to physical pages. For this question we assume that page directory and table entries only contain two fields: the P or *present* bit, and a page number which is only valid if P=1. Your answer will need to provide the following information:

- **CR3 (page table root pointer) value** – physical page number
- **Page directory entries** – give index (0..1023) and page number value for each entry with P = 1.
- **Page table (second level) entries** – give index and page number for each entry with P = 1.

In addition, for each physical page 00000 through 0000B you should identify whether it is being used as the page directory, a page table, or as a data page; for data pages please give the virtual address at which it is mapped.

Note that the physical pages may be used in any order for directory, table, and data pages; the only requirement is that the final result must be a valid page table for the specified virtual address space.

**Question 3**

Create a counting semaphore from mutexes. You will need to provide pseudo-code fragments giving the following:

- **variables**: one or more mutexes, one or more integer or boolean variables
- **init(n) method** – initialize with count='n'
- **enter() method**
- **leave() method**

As a hint, you may wish to use the following:

- One mutex to guard your variables, and a second to block waiting threads.
- One integer to hold the current semaphore count - i.e. 'n' minus the number of threads currently in the semaphore.
- A second integer for a count of the number of waiting threads.

Any correct implementation of a semaphore using mutexes is acceptable. Note that we assume that one thread may lock a mutex and then another thread release it.
Question 4
Given the following access pattern to virtual pages, and 4 pages of physical memory, show the operation of the FIFO, LRU, and OPT algorithms. For each algorithm provide (a) the list of resident pages after each access, and (b) indication of whether each access is a hit or a miss.

1 2 3 4 1 2 5 2 1 6 2 3 7 6 1 2 6 2 1 3

You may submit your homework electronically via Blackboard (pdf preferred) or on paper at the beginning of class.