Alright, more problems, some pretty difficult. If a problem is unspecified, make some reasonable assumption to solve the problem. If there is no obvious reasonable assumption, maybe I messed up, so email me at jhug@eecs.berkeley.edu and let me know what is broken.

If you get stuck, don't worry about it. Post on the newsgroups/email me/wait until Wednesday.

**Sleep:**
Unless you're really behind, being well rested is more important than cramming that last little bit of knowledge into your brain. Do your cramming Tuesday so you have some time to sleep on it and let your mind digest everything. You want to be quick on your feet during the test!

**C Programming and MIPS Basics:**

Somewhere on the order of half of the final (which I haven't seen) will be based on the stuff before the first midterm. The midterm clobbering (where we increase your midterm score) is going to be based only on the material from the first midterm.

**Thus, make sure you can understand the answers to the previous midterm!!** In particular, know how mallocing and freeing works, make sure you can do the sizeof questions, and make sure you can do MAL/TAL conversion and vice versa. Also understand everything else on the midterm.

You had better not do something like the following on the final because it makes us all a little sad:

```
void freelist(struct node* currentnode)
{
  if (currentnode==NULL)
        return;
  freelist(currentnode->next);
  free(currentnode);
  free(currentnode->next);
}
```

If this seems at all reasonable, make sure you understand that free isn't something you do to a pointer, but something you do to a memory location that has been mallocated. If it seems unreasonable, make sure you understand which line here is a problem.

**Pipelining:**

For the problems below, unless otherwise specified, assume the 5 stage non-forwarding processor implementation given in P&H.

**Problem P1:** A CS 61C student is extremely paranoid and wants to have a special instruction which immediately erases data memory with as little delay as possible, but which never erroneously erases all memory. Assume that our data and instruction memory has a special "clear all" input. Show how we'd implement this instruction in the data path. In what stage does this instruction complete execution?

**Problem P3:** Give an example of a structural hazard that we might encounter on a modern implementation of MIPS.

**Problem P4:** Imagine that we find that our ALU is the bottleneck in our MIPS processor, and is taking 1.75 times as long as the second slowest stage of our pipeline. Would it be appropriate to add a pipeline register in the middle of our ALU [thus making two execute stages]? Why or why not?

**Problem P5:** Suppose that we implement our instruction memory with an ultrafast SRAM that takes almost no time at all relative to our other hardware. Would it be appropriate to merge the decode and instruction fetch stages? What would we have to do to ensure correct code execution if we combined these two stages?

**Problem P6:** The processors at the end of the Pentium IV line had 31 pipeline stages. Why is this especially problematic for a program with a large number of branches?

**Problem P7:** You are at a pub, possibly illegally, on a stormy night. In walks an old grizzled sailor, gray with age and grimy with oil, his skin cracked and broken by the salty sea air. He has heard about your single stage MIPS processor that you created for this class, and wishes to challenge you to a game of skill. Let's assume your processor can run at 2 GHz. The old man says he has implemented a pipelined version of the very same processor you have implemented, and that it too runs at 2 GHz.

Is there any program that will run faster on his processor than yours? Why or why not?

**Problem P7b:** x86 processors have been about the same clock speed for the last several years. If deeper pipelines don't help, then how are newer processors capable of executing more instructions per second than newer processors?

**Problem P8:** How long will each iteration of the following program take to execute (in cycles)?

```
loop:
        lw      $t0,0($s1)
        addu    $t0,$t0,$s2
        sw      $t0,0($s1)
        addiu   $s1,$s1,-4
        bne     $s1,$0,loop
        nop
```

**Problem P9:** Now assume that we implement data forwarding as discussed in class. How long will the program from P8 take to execute (in cycles)?

**Caching:**

**Problem C1:** Suppose that we used our virtual memory address to index into our cache (instead of the physical memory address). Why would this be a problem?

**Problem C2:** Suppose we get a TLB hit. Will we use always check our data cache for data? Will we always find it there if we look?

**Problem C3:** If we have a 4-way set associative cache with 32,768 blocks, which bits of our physical address will we use as index bits to pick the correct set? If there's not enough information to answer, what else do we need to know?

**Problem C4:** If cache is so fast, why don't we just have 4 gigabytes of cache instead of RAM?

**Problem C5:** Assume we have an N-way set associative cache with 16,384 blocks of 4 bytes each, which is initially blank at the beginning of our program. Assume that we are using the LRU replacement scheme and that our system is byte addressed [so 2 offset bits].

a. In terms of N, what is the minimum number of instructions needed to cause a cache line replacement [assume all registers are initially 0, and that all memory addresses are valid]?
b. Assume N=2. Given 3 memory accesses to locations $L_1$, $L_2$, $L_3$, give a condition in terms of $L_1$, $L_2$, $L_3$ under which a replacement occurs.

**Problem C6:** Write a MIPS program for which a Most Recently Used replacement scheme is optimal. For easiness, assume a fully associative cache.

**Problem C7:** Homework 8. See Problem X3 for the ultimate in caching problems.

**Virtual Memory:**

**Problem V0:** An entry in the data cache is a cache of one block of data memory. An entry in the TLB is a cache of _____.

**Problem V1:** In class, we said that each virtual memory address is mapped to a physical address by a page table. How do we know which page table to consult? [In the lecture notes if you're stuck]

**Problem V2:** How much memory in bits would be required for a one level page table if we have a virtual address space of 32 bits, a physical address space of 29 bits, 4 KB pages, two bits of access and one valid and one dirty bit per line.

**Problem V3:** If we use the virtual address to index into the TLB, and we don't do anything to the TLB when we switch from one process to another, what problem might we encounter? What are two possible solutions to this problem?

**Problem V4:** Suppose that we now create a two-level page table for our system in problem V1. Now, we have two types of tables. The first is type is a master table which contains a list of addresses of page tables, and there is only one master table. The second table is just a page table as usual, only it now holds information about only a subset of the addresses in our virtual address space. If each page table is to contain $2^{10}$ rows, how much memory in bits will be required to store an entire page table solution?

**Problem V5:** Since the page table is implemented in software, we could in principle have a monolithic page table like in problem V2 which dynamically allocates memory as virtual addresses are used. Why would this be a bad thing?

**Problem V6**: Why don't we have tag bits on a row of the page table? Why do we need them in the TLB?

**Problem V7:** Suppose that we have a computer with 16 GB of RAM, giving us a 34 bit physical address space. Is there any reason to implement a 32 bit virtual addressing system? (In other words, do we gain anything by virtualization?). If we do implement such a table, what will be fundamentally different about our page table? Would a TLB still be useful?

**Everything Together Part 1:**

**Problem X1**: Given a program with high spatial and temporal locality, and an architecture with typical TLB, page table, and cache sizes. Rank the likelihood of the following:

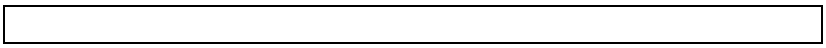
a. TLB miss, page table hit, cache hit
b. TLB miss, page table hit, cache miss
c. TLB miss, page table miss, cache miss

**Problem X2**: Given a program that accesses random memory locations, and an architecture with typical TLB, page table, and cache sizes. Rank the likelihood of the same scenarios as in **X1.**

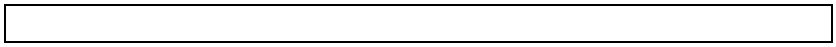
**Problem X3:**

For this problem, we have a 32 bit virtual address space and a 28 bit physical address space. We also have a direct mapped TLB with 16 entries, and no access bits. Memory is divided into 64KB pages. We have a 16,384 entry 64-way set associative write back cache which stores 16 byte blocks. Our page table is the normal one level scheme discussed in class (where each process has a page table, and that page table has one entry for each virtual address). Our system is byte addressed.

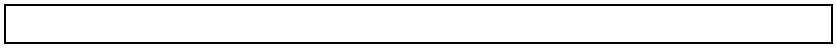
a. How many LRU bits do we need for an N-way set associative information cache?
b. Draw in the fields of an entry in the TLB below [there should be 5 fields, with a total width of 27 bits]. Specify the width of each field.



c. What is the total size of the TLB in bits?
d. Draw in the fields of an entry in the page table below [2 fields, 13 bits total]. Specify the width of each field.



e. What is the total size of the page table in bits?
f. Draw in the fields of an entry in the cache below [5 fields, 152 bits total]



g. What is the total size of the cache in bits?
h. Alright, now that we've got that out of the way, let's analyze program behavior using our fancy setup. Let's assume our system has a blank TLB, blank page table, and blank cache (so all valid bits are zero, and all of memory is 0s). Assume that after a page table miss that the data on the physical page loaded by the operating system is all 1s the first time, all 2s the second time, and so forth.

   Assume that our operating system will allocate virtual addresses in order. So the first call to a virtual memory location that is not in physical memory will be allocated to 0x0000000. So after the first page fault, we'll have a block of 65,535 repeats of the byte 0x01 starting at memory location 0x00000000. On the second page fault, we'll have a block of 65,535 repeats of the byte 0x02 starting at memory location 0x00010000. This is meant to simulate loading some information that was previously written to the disk. In this case, it was pretty boring information that was just a long sequence of the same number over and over.

   Suppose that we tell the OS that we want to execute the MAL instruction lw $t0, 0x12345678.

   1. What is the TAL equivalent of this instruction [show assembly code not binary]?
   2. Show all fields of the entry of the TLB that change after this instruction is executed.
   3. Show all fields of the entry of the page table that change after this instruction is executed. [Be careful, remember the point of the TLB…]. What is the index of this entry?
   4. Show all but the data field of the entry of the cache that change after this instruction is executed. What is the index of this entry?
   5. Does anything in memory change?
   6. What value goes into the $t0 register?
   7. The operating system translates our TAL instructions from part 1. Show what instruction actually gets executed by the processor.

i. Suppose that the next instruction that comes along is lw $t0, 0x12345680.

1. Do we have a TLB hit? A page file hit? A cache hit?
2. Does anything change in the TLB, page file, cache, or memory?

j. Suppose that the next instruction is sw $t0, 0x123456A0.
   1. Do we have a TLB hit? A page file hit? A cache hit?
   2. Does anything change in the TLB, page file, cache, or memory?

k. Suppose that the next instructions are a sequence of 5 loads:

```
lw $t0, 0x00001234
lw $t0, 0x00210F08
lw $t0, 0x00311234
lw $t0, 0x00210000
lw $t0, 0x00311238
```

   For each of the above instructions, say whether it is a TLB hit? A page file hit? A cache hit?

l. Give an example of a program that would be a TLB miss, page file miss, then cache hit.

**Problem I/O 1:** Do the I/O labs. If you feel out of time, at least be able to read and write from devices using polling.

**Problem I/O 2:** What are the advantages of interrupts over polling? Polling over interrupts?

**Problem I/O 3:** Give an example of a system with low latency and low bandwidth. A system with high latency and high bandwidth? When is each appropriate?