Only Problem Set Two will be graded. Turn in only Problem Set Two before November 11, 2008, 11:00am.

1 Problem Set One

- Hennessy & Patterson (4th Ed) 2.3
- Hennessy & Patterson (4th Ed) 2.5
- Hennessy & Patterson (4th Ed) 2.6
- Hennessy & Patterson (4th Ed) 2.10
- Hennessy & Patterson (4th Ed) 2.12
- Hennessy & Patterson (4th Ed) 3.1
- Hennessy & Patterson (4th Ed) 3.2(a)(b)(c)(d)(e)
2 Problem Set Two

1 (Register Renaming) (30 points)

Instead of using a reorder buffer (ROB), one alternative approach for hardware speculation is the explicit use of an extended set of physical registers combined with register renaming. These physical registers are allocated in the issue stage to eliminate WAW and WAR hazards. Fundamentally, the architectural destination register of each instruction is mapped to a free physical register in the extended register set. A hardware renaming table is used to keep track of the mapping between two register sets.

(Part A) The following piece of code is considered for register renaming. Assume your hardware has a pool of 64 temporary registers (called T registers T0 through T63). Additionally, assume at the initiation of the given code, T0 through T6 have been occupied. You are asked to perform register renaming for the code segment below, that is, to replace each destination register with the next available T register. (As a hint, the renaming of the first instruction has already been done for you.) Furthermore, please specify the status of the renaming table after renaming the SUBD instruction by filling in the T registers you have used.

```
LD    F2, 0(Rx)
LD    F4, 0(Ry)
MULTD F5, F0, F2
ADDD F6, F0, F2
DIVD F8, F5, F4
ADDD F9, F0, F4
SUBD F2, F5, F6
```

Renamed code segment:

```
LD    T7, 0(Rx)
```

Status of renaming table after renaming the SUBD instruction:

```
<table>
<thead>
<tr>
<th>Architectural registers</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F8</th>
<th>F9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical registers</td>
<td>T4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Unlike the rather straightforward allocation strategy, the **deallocation** of physical registers is more complicated: a physical register can only be freed up if it no longer corresponds to an architectural register and no further uses of the physical register are outstanding. On the other hand, the deallocation of physical registers should also be performed as early as possible, since instruction fetch will be stalled if no free physical registers are available. The remaining part of the question explores deallocation strategies. *You do not have to worry about exception behavior throughout.*

(Part B) The simplest approach that a set of engineers could come up with is “checking the source registers of all the in-flight instructions”. The manager is concerned about whether this checking works or whether it will end up in a product recall. Could you help the manager figure out the correctness of this approach? Give a brief explanation for your decision.

(Part C) Leaving the correctness of the “fully checking” mechanism aside, its high cost motivates the exploration of cheaper techniques. Consequently, another group has suggested that for an **in-order** commit processor, a physical register can be deallocated once a subsequent instruction which writes the same architectural register has been committed. Please comment on the correctness of this technique. If you think it is not correct, please provide a reasoning as to why. Otherwise, please provide a discussion of any possible shortcomings.

(Part D) The company is discussing how to handle physical register deallocation for **out-of-order** commit processors. One suggestion is to just use the deallocation strategy in (Part C), that is, to free up a physical register when a subsequent instruction writing the same architectural register has been committed. Please comment on the correctness of this technique. If you think it is not correct, please provide a reasoning as to why. Otherwise, please provide a discussion of any possible shortcomings.

(Part E) Another engineer is arguing that for out-of-order commit processors, the compiler should provide hints to the hardware in physical register deallocation. He has furthermore suggested two possible hints:

- Marking the last use of each destination architectural register.
- Providing a count of the subsequent use for each destination architectural register.

Would these hints be helpful in developing a cheaper deallocation strategy that works correctly under any conditions? Would these hints be helpful for some DSP codes with no branches? For each hint, if you think the hint is helpful, please give your corresponding deallocation strategy. Otherwise, please give your reasoning as to why it is not helpful.
2 (Software pipelining) Consider the following three loops. Each one of them reads an array value, adds a constant value to it, and subsequently stores it into another array location. A natural technique to parallelize such loops and execute them on a VLIW architecture is to apply software pipelining. Please assume that no memory disambiguation hardware is present and that load and store instructions can be executed in parallel but that their actual order in accessing the memory is undefined. For each of the loops below, please identify if software pipelining could be applied. (Assume that all these loops are coded as a sequence of the three load, add, and store instructions, followed by the loop overhead code.) Please explain your answers.

```c
for (i=0;i<100;i++)
```

```c
for (i=0;i<100;i++)
```

```c
for (i=0;i<100;i++)
```
3 (Simultaneous Multithreading)  (18 points)

One critical issue in computer architecture is the limitation of instruction-level parallelism, which motivates the use of the multithreading technique to allow multiple threads to share the functional units of a single processor in an overlapping fashion. Recall that to permit this sharing, the processor needs to add more hardware to preserve the independent state of each thread.

(Part A) There is a raging debate going on within a company that is trying to adopt the SMT technique for a new processor they are designing. Being newcomers to the block, their business strategy is to undercut the competition particularly on price (of course at no impact on performance!). The engineers are involved in heated discussions trying the figure out the feasibility & desirability of adopting shared structures and those in the camp of shared structures are arguing further whether the shared structure is the only way to go, whether the shared structure is neutral performance-wise or whether the shared structure will impose a stiff performance degradation on the system. Of course, this being a low-cost provider, no additional functionality or hardware is envisioned in the case of the shared structure. Here are the four possibilities:

A The shared structure will not work, period.
B The shared structure not only will work but is the only way to go; anything else will result in incorrectly functioning processors.
C The shared structure will work and it does not make a difference performance-wise whether the structure is shared or not.
D The shared structure will work but the performance will degrade stiffly.

For the following 5 cases, decide which of the 4 options above holds and provide a brief reasoning.

Instruction fetch queue:

Register renaming table:

L1 cache:

Load/store queue:

Reorder buffer:

(Part B) One of the limiting factors on an SMT is the bandwidth of getting instructions into the fetch queue. While giving preference to the preferred thread(s) is beneficial in terms of the latency of the preferred thread, it may impact the ability of the processor to utilize fully the processing units. Why?

If your preferred thread(s) were straightline DSP code with little branch behavior & highly analyzable memory accesses, which instruction fetch queue loading policy will you suggest?

For a preferred application with heavy conditional utilization & pointer chasing, which instruction fetch queue loading policy would you suggest?