Principles in Computer Architecture I – CSE 240A (Section 631684)

CSE 240A Homework One

September 28, 2008

Only Problem Set Two will be graded. Turn in only Problem Set Two which will be due on October 16, 2008 (Thursday) at 11:00am.

1 Problem Set One

- Hennessy & Patterson (4th Ed) 1.2
- Hennessy & Patterson (4th Ed) 1.5
- Hennessy & Patterson (4th Ed) 1.7
- Hennessy & Patterson (4th Ed) 1.9
- Hennessy & Patterson (4th Ed) 1.10
- Hennessy & Patterson (4th Ed) 1.13
- Hennessy & Patterson (4th Ed) 1.14
- And Attached 5 Appendix A problems
2 Problem Set Two

1 (Performance vs. cost) (25 points)

A company which used to sell single-core processors is forced by the market to produce dual-core processors instead of single-cores. The design team is evaluating the idea of integrating two cores into one die in fabrication. You are asked to calculate the cost of the dual-core die, given the following assumptions in your calculation:

- Die cost is inversely proportional to the number of good dies per wafer.
- The number of dies per wafer is inversely proportional to die area.
- The area of a dual-core die = 2 * the area of a single-core die.

Additionally, the design team provides you the following formula on die yield:

\[
\text{Die yield} = \text{Wafer yield} \times (1 + \frac{\text{Defects per unit area} \times \text{Die area}}{\alpha})^{-\alpha}
\]

(Please use a value for $\alpha$ that is appropriate for current VLSI technology.)

You were told that previously when each die only has one core, the die yield = 90%, and the cost of each die = $50. Then, what is the cost of a dual-core die? Please show the intermediate steps in your calculation.

The cost of a dual-core die =

(Part B) Given that a due-core die is more expansive than a single-core die, The design team has come up with two ways to produce a due-core processor:

1. Integrate two cores on each die in fabrication.
2. Integrate one core on each die yet package two dies into one chip.

You are asked to evaluate these two ideas in terms of their impact on performance and power. What is your opinion and what are your reasons? Please clearly state them below.
2 (Reg-Mem ISA Pipeline)  

This question concerns the pipeline implementation of a Reg-Mem ISA. Recall that a Reg-Mem ISA allows one of the operands of ALU instructions to come from the data memory. Therefore, a simple “add” instruction can have the following possible formats:

\[
\begin{align*}
\text{add } R1, R2, R3 \\
\text{add } R1, R2, (R3) \\
\text{add } R1, R2, 100 \\
\text{add } R1, R2, (100)
\end{align*}
\]

A simple way to implement this Reg-Mem ISA is to simply reorder the 5 stages of the traditional MIPS pipeline. Now the order of the modified 5 stages are \textbf{IF ID MEM EXE WB}, which allows the ALU instructions to get one of the operands from data memory before performing ALU operations. On the other hand, since we do not want to incur extra cost for these expensive hardware components, the hardware resources of each stage in this Reg-Mem pipeline are identical to the stage with the \textbf{same} name in MIPS pipeline. Another similar point to the MIPS pipeline is that register R0 always equals 0.

(Part A) Among the following group of instructions, one of them is redundant in this particular Reg-Mem architecture implementation. Which one? Give a small example to illustrate how to replace that instruction with one or more other instructions in the group.

\begin{itemize}
\item[A.] load  \hspace{1cm}  \item[B.] store  \hspace{1cm}  \item[C.] add  \hspace{1cm}  \item[D.] multiply  \hspace{1cm}  \item[E.] branch
\end{itemize}

The redundant instruction is ( )

Example:

(Part B) Because of the limitation of hardware resources in this Reg-Mem ISA, the \textbf{data transfer} instructions (i.e. loads and stores) can only have two possible addressing modes: \textit{register indirect addressing} and \textit{absolute (direct) addressing}. Compared with the MIPS pipeline, both the change of ISA (from Reg-Reg to Reg-Mem) and the change of addressing modes will have an influence in instruction counts. For different classes of instructions (i.e. load, store, ALU, branch), please compare this Reg-Mem pipeline to the MIPS pipeline in terms of instruction counts and explain the influence this Reg-Mem pipeline will have. Give a brief explanation of your conclusion.

(Part C) One danger of this particular 5-stage Reg-Mem pipeline is the potential violation of precise exceptions. Recall that in the case of the traditional 5-stage MIPS pipeline, the handling of exceptions is delayed to the \textbf{WB} stage to ensure all the exceptions are taken in order. However, this simple mechanism can not provide precise exceptions in this Reg-Mem pipeline. Please give a code fragment to illustrate this potential violation and explain clearly the problem (i.e. exception will happen at which stage of which instruction, and which instruction will change the state of the processor improperly).
3 (Exceptions in Scoreboard) (40 points)

The fundamental rule of exception handling is that the instructions issued after the faulting instructions should not be allowed to change the processor into an **unrecoverable state** when an exception occurs. One simple way to ensure this result is through *precise exceptions*, which disallows any subsequent instructions’ writing their results if a previous instruction has not finished execution. Obviously, the same semantically correct result that precise exceptions deliver can be accomplished by other means. We are asking you to explore these concepts within the context of the scoreboard implementation. Recall that each instruction undergoes **four steps** in the scoreboard: *issue, read operands, execution, and write results*. In this process, the three types of hazards, **WAW, RAW** and **WAR** hazards, are checked at the *issue, read operands, and write results* stages, respectively.

For the rest of the problem, you can use the following code fragment to analyze the various conditions.

ADD.D F2, F6, F8  
DIV.D F0, F2, F4  
SUB.D F4, F6, F10  
MUL.D F6, F8, F2  
ADD.D F0, F2, F8

(Part A) For the above code fragment, assume the **DIV.D** instruction, which typically has a latency of 40 cycles, may cause an exception during its execution. In this situation, is any of the following three instructions, **SUB.D, MUL.D, ADD.D**, allowed to enter the *write results* stage? If you think an instruction cannot write its result, can it be issued if the corresponding functional unit is free? Give your reasons for each instruction.

1. **SUB.D**

2. **MUL.D**

3. **ADD.D**
(Part B) The traditional scoreboard handles \textit{WAW, RAW} and \textit{WAR} hazards in the following ways:

\textbf{WAW:} at \textit{issue} stage, an instruction can be issued if no other active instruction has the same destination register.

\textbf{RAW:} at \textit{read operands} stage, the functional unit can proceed to read the operands if no earlier issued active instruction is going to write any of the source operands.

\textbf{WAR:} at \textit{write results} stage, a completing instruction cannot be allowed to write its result when there is an earlier issued active instruction that has not read its operands and one of the operands is the same register as the result of the completing instruction.

We want to incorporate exception handling into the scoreboard. However, \textit{we do not want to unnecessarily delay the issue of instructions}. Given this condition, does any of the above three rules need to be modified in order to handle exceptions in a scoreboard? If you think a rule needs to be modified, please specify the corresponding new hazard checking mechanism. Otherwise, please give a reasoning as to why no modification is needed.

(Part C) It seems that the incorporation of exception handling introduces significant performance overhead for the traditional scoreboard. To minimize this overhead, a clever engineer has suggested to insert several \textit{“exception flags”} for each instruction. Fundamentally, an exception flag indicates whether an exception condition has been cleared, even though the corresponding instruction hasn’t finished its execution yet.

Using these \textit{“exception flags”}, do you think the conventional hazard checking mechanism, presented on the last page, needs to be modified? \textit{Again, we do not want to unnecessarily delay the issue of instructions}. If you think a rule needs to be modified, please specify the corresponding new hazard checking mechanism. Otherwise, please give a reasoning as to why no modification is needed.