# HW1 Assignment

## Problem 1: Solve P1.7, P1.8, P1.9, and P1.10 in Chapter 1 of the Shen/Lipasti textbook.

1. Given the parameters of Problem 6, consider a strength-reducing optimization that converts multiplies by a compile-time constant into a sequence of shifts and adds. For this instruction mix, 50% of the multiplies can be converted to shift-add sequences with an average length of three instructions. Assuming a fixed frequency, compute the change in instructions per program, cycles per instruction, and overall program speedup.

TABLE 1 CPI computation

| Type | Old Mix | New Mix | Cost | CPI |
|---|---|---|---|---|
| store | 15% | | 1 | |
| load | 25% | | 2 | |
| branch | 15% | | 4 | |
| integer & shift | 40% | | 1 | |
| multiply | 5% | | 10 | |
| **Total** | **100%** | | | |

(P1.6 for reference: A program's run time is determined by the product of instructions per program, cycles per instruction, and clock frequency. Assume the following instruction mix for a MIPS-like RISC instruction set: 15% stores, 25% loads, 15% branches, and 35% integer arithmetic, 5% integer shift, and 5% integer multiply. Given that load instructions require two cycles, branches require four cycles, integer ALU instructions require one cycle, and integer multiplies require ten cycles, compute the overall CPI.)

2. Recent processors like the Pentium 4 processors do not implement single-cycle shifts. Given the scenario of Problem 7, assume that $s = 50\%$ of the additional integer and shift instructions introduced by strength reduction are shifts, and shifts now take four cycles to execute. Recompute the cycles per instruction and overall program speedup. Is strength reduction still a good optimization?

TABLE 2 CPI computation

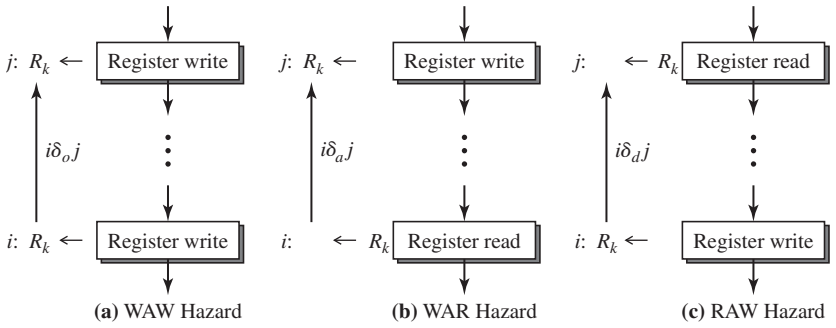
| Type | Old Mix | New Mix | Cost | CPI |
|---|---|---|---|---|
| store | 15% | | 1 | |
| load | 25% | | 2 | |
| branch | 15% | | 4 | |
| integer | 35% | | 1 | |
| shift | 5% | | 4 | |
| multiply | 5% | | 10 | |
| **Total** | **100%** | | | |

3. Given the assumptions of Problem 8, solve for the break-even ratio $s$ (percentage of additional instructions that are shifts). That is, find the value of $s$ (if any) for which program performance is identical to the baseline case without strength reduction (Problem 6).

4. Given the assumptions of Problem 8, assume you are designing the shift unit on the Pentium 4 processor. You have concluded there are two possible implementation options for the shift unit: 4-cycle shift latency at a frequency of 2 GHz, or 2-cycle shift latency at 1.9 GHz. Assume the rest of the pipeline could run at 2 GHz, and hence the 2-cycle shifter would set the entire processor's frequency to 1.9 GHz. Which option will provide better overall performance?

## Problem 2: Solve P2.4, P2.5, P2.6 in Chapter 2 of the Shen/Lipasti textbook.

Figure 2-15



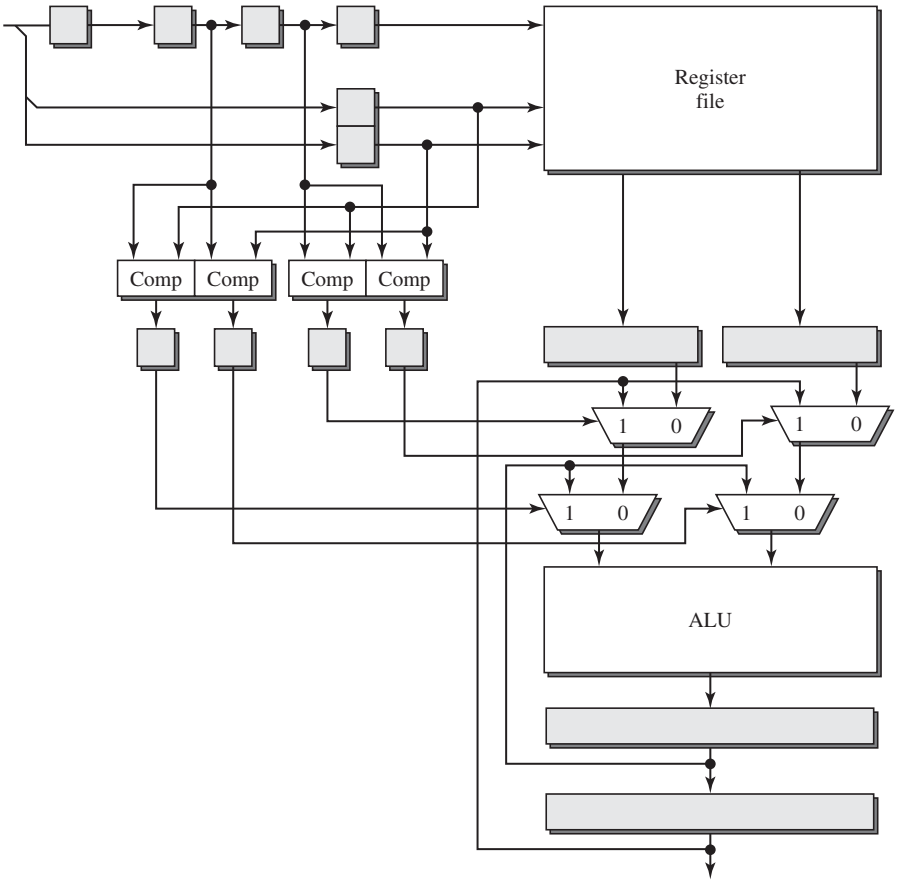

5. Consider that you would like to add a *load-immediate* instruction to the TYP instruction set and pipeline. This instruction extracts a 16-bit immediate value from the instruction word, sign-extends the immediate value to 32 bits, and stores the result in the destination register specified in the instruction word. Since the extraction and sign-extension can be accomplished without the ALU, your colleague suggests that such instructions be able to write their results into the register in the decode (ID) stage. Using the hazard detection algorithm described in Figure 2-15, identify what additional hazards such a change might introduce.
6. Ignoring pipeline interlock hardware (discussed in Problem 6), what additional pipeline resources does the change outline in Problem 4 require? Discuss these resources and their cost.
7. Considering the change outlined in Problem 4, redraw the pipeline interlock hardware shown in Figure 1-1 to correctly handle the load-immediate instructions.

## Problem 3: Solve P2.13 and P2.14 from Chapter 2 of the Shen/Lipasti textbook.

8. Given the IBM experience outlined in Section 2.2.4.3, compute the CPI impact of the addition of a level-zero data cache that is able to supply the data operand in a single cycle, but only 75% of the time. The level-zero and level-one caches are accessed in parallel, so that when the level-zero cache misses, the level-one cache returns the result in the next cycle, resulting in one load-delay slot. Assume uniform distribution of level-zero hits across load delay slots that can and cannot be filled. Show your work.

9. Given the assumptions of Problem 11, compute the CPI impact if the level-one cache is accessed sequentially, only after the level-zero cache misses, resulting in two load-delay slots instead of one. Show your work.

FIGURE 1-1 Figure 2-18: Implementation of pipeline interlock for RAW hazards involving a leading ALU instruction.




# Problem 4: Solve P3.3 and P3.13 from Chapter 3 of the Shen/Lipasti textbook.

10. Given the example code in Problem 1 (below), and assuming a virtually-addressed two-way set associative cache of capacity 8KB and 64 byte blocks, compute the overall miss rate (number of misses divided by number of references). Assume that all variables except array locations reside in registers, and that arrays A, B, and C are placed consecutively in memory.

    ```
    double A[1024], B[1024], C[1024];
    for(int i=0;i<1000;i += 2) {
        A[i] = 35.0 * B[i] + C[i+1];
    }
    ```

11. Consider a processor with 32-bit virtual addresses, 4KB pages and 36-bit physical addresses. Assume memory is byte-addressable (i.e. the 32-bit VA specifies a byte in memory).

    ❑L1 instruction cache: 64 Kbytes, 128 byte blocks, 4-way set associative, indexed and tagged with virtual address.

    ❑L1 data cache: 32 Kbytes, 64 byte blocks, 2-way set associative, indexed and tagged with physical address, write-back.

❑4-way set associative TLB with 128 entries in all. Assume the TLB keeps a dirty bit, a reference bit, and 3 permission bits (read, write, execute) for each entry.

Specify the number of offset, index, and tag bits for each of these structures in the table below. Also, compute the total size in number of bit cells for each of the tag and data arrays.

| Structure | Offset bits | Index bits | Tag bits | Size of tag array | Size of data array |
|---|---|---|---|---|---|
| I-cache | | | | | |
| D-cache | | | | | |
| TLB | | | | | |

## Problem 5: Solve P4.5, P4.8 from Chapter4 of the Shen/Lipasti textbook.

12. One idea to eliminate the branch misprediction penalty is to build a machine that executes both paths of a branch. In a 2-3 paragraph essay, explain why this may or may not be a good idea.
13. In an in-order pipelined processor, pipeline latches are used to hold result operands from the time an execution unit computes them until they are written back to the register file during the writeback stage. In an out-of-order processor, rename registers are used for the same purpose. Given a four-wide out-of-order processor TYP pipeline, compute the minimum number of rename registers needed to prevent rename register starvation from limiting concurrency. What happens to this number if frequency demands force a designer to add five extra pipeline stages between dispatch and execute, and five more stages between execute and retire/writeback?

## Problem 6: Solve P5.1, P5.2, P5.7, P5.14 from Chapter 5 of the Shen/Lipasti textbook.

14. Identify basic blocks:

| BB# | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Instr. #s | | | | | | | | | |

15. Draw the control flow graph for this benchmark.

16. Assume that an one-bit (history bit) state machine (see above) is used as the prediction algorithm for predicting the execution of the two branches in this loop. Indicate the predicted and actual branch directions of the b1 and b2 branch instructions for each iteration of this loop. Assume initial state of 0, i.e., NT, for the predictor.

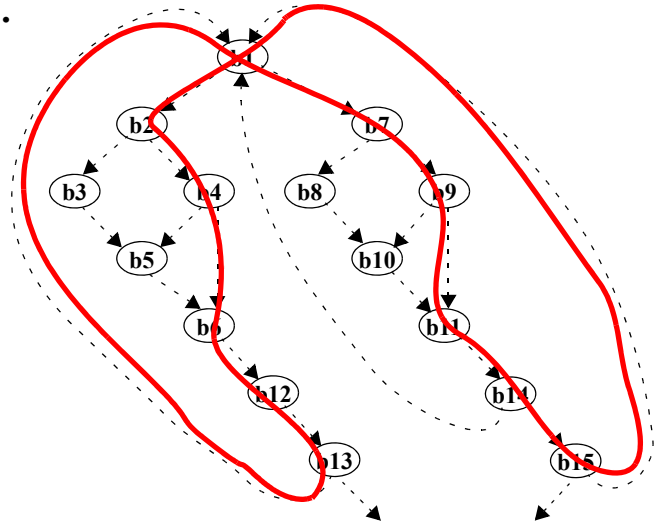
| | 8 | 9 | 10 | 11 | 12 | 20 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|
| b1 predicted: | ____ | ____ | ____ | ____ | ____ | ____ | ____ | ____ | ____ |
| b1 actual: | ____ | ____ | ____ | ____ | ____ | ____ | ____ | ____ | ____ |

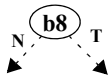
b2 predicted: ____ ____ ____ ____ ____ ____ ____ ____ ____

b2 actual: ____ ____ ____ ____ ____ ____ ____ ____ ____

17. Below is the control flow graph of a simple program. The CFG is annotated with three different execution trace paths. For each execution trace circle which branch predictor (bimodal, local, or Gselect) will *best* predict the branching behavior of the given trace. More than one predictor may perform equally well on a particular trace. However, you are to use each of the three predictors *exactly once* in choosing the best predictors for the three traces. *Circle your choice* for each of the three traces and add. (*Assume each trace is executed many times and every node in the CFG is a conditional branch. The branch history register for the* local, global, *and* Gselect *predictors is limited to 4 bits.)*

**1.**




**Circle one:**

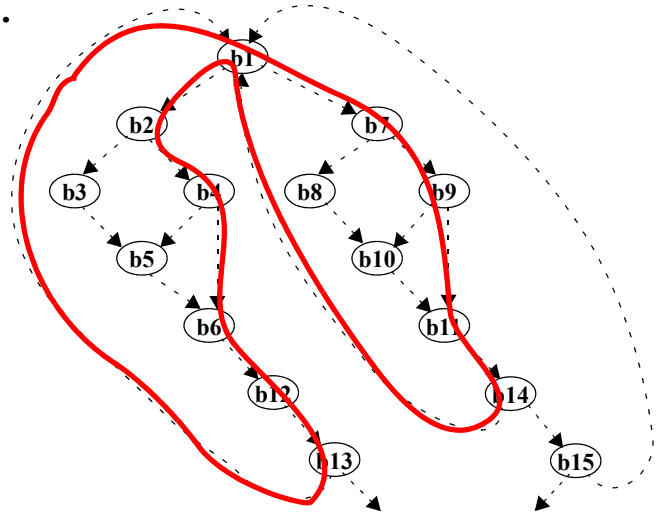
**Bimodal**

**Local**

**Gselect**

Identical global history at **b13** and **b15**, so the PC is need to differentiate them.

**2.**





**Circle one:**

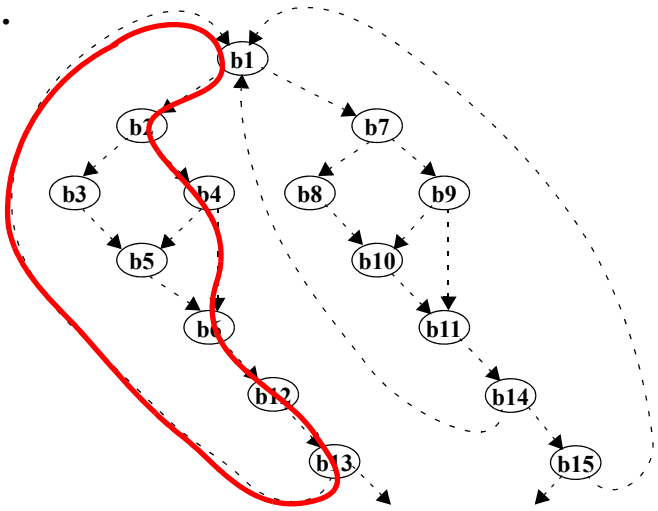
**Bimodal**

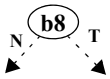
**Local**

**Gselect**

Identical global history at **b1**, so global history doesn't work. The local history of **b1** shows it alternates taken and not taken.

**3.**




**<u>Circle one</u>:**

**Bimodal**

**Local**

**Gselect**

All the branches in this trace have a constant behavior, so bimodal predicts well.