HW1 Solutions

Problem 1

1. Given the parameters of Problem 6 (note that int =35% and shift=5% to fix typo in book problem), 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%	15%	1	0.15
load	25%	25%	2	0.50
branch	15%	15%	4	0.60
integer & shift	40%	47.5%	1	0.475
multiply	5%	2.5%	10	0.25
Total	**100%**	**105%**		**1.975/105% = 1.88**

There are 5% more instructions per program, the CPI is reduced by 12.5% to 1.88, and overall speedup is 2.15/1.975 = 1.089 or 8.9%.

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%	15%	1	0.15
load	25%	25%	2	0.50
branch	15%	15%	4	0.60
integer	35%	38.75%	1	0.3875
shift	5%	8.75%	4	0.35
multiply	5%	2.5%	10	0.25
Total	**100%**	**105%**		**2.2375/105% = 2.131**

Speedup is now a slowdown: 2.15/2.2375 = 0.96 or 4% slowdown, hence strength reduction is a bad idea.

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).

2.15 = (0.15+0.50+0.60+0.35+0.05x4+0.25 + (1-s)x0.075x1 + sx0.075x4

0.025 = 0.225s => s = 0.111 = 11.1%

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?

   4-cycle shifter: Time per program = 1.05 IPP x 2.2375 CPI x 1/2.0GHz = 1.17e-9

   2-cycle shifter: Time per program = 1.05 IPP x (2.2375-0.175) CPI x 1/1.9GHz = 1.13e-9

   Hence, 2-cycle shifter is a better option if strength reduction is applied.

   If there is no strength reduction (back to Problem 6):

   4-cycle shifter: Time per program = 1.00 IPP x 2.30 CPI x 1/2.0GHz = 1.150e-9 s

   2-cycle shifter: Time per program = 1.00 IPP x 2.20 CPI x 1/1.9GHz = 1.157e-9

   Hence, the 4-cycle shifter is a better option. Overall, the best choice is still strength reduction with a 2-cycle shifter @1.9GHz.

Problem 2

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.

   Since there are now 2 stages that write the register file (ID and WB), WAW hazards may also occur in addition to RAW hazards. WAW hazards exist with respect to instructions that are ahead of the load immediate in the pipeline. WAR hazards do exist since the ID register write stage is earlier than the RD register read stage (assuming a write-before-read register file). If the register file is read-before-write, the ID write occurs after the RD read, and therefore WAR hazards do not exist.

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.

   Since there are 2 stages that write the register file (ID and WB), the register file must have two write ports. Additional write ports are expensive, since they require the RF array and bitcells to be redesigned to support multiple writes per cycle. Alternatively, a banked RF design with bank conflict resolution logic could be added. However, this would require additional control logic to stall the pipeline on bank conflicts.

7. Considering the change outlined in Problem 4, redraw the pipeline interlock hardware shown in Figure 2-18 to correctly handle the load-immediate instructions.

The modified figure should show the ID stage destination latch connected to a second write port register identifier input. Further, comparators that check the ID stage destination latch against the destination latches of instructions further in the pipeline should drive a stall signal to handle WAW hazards.

Problem 3

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.

   The CPI effect of unfilled load delay slots is 0.0625. Since the L0 cache can satisfy 75% of these loads, 75% of this CPI term disappears. Hence, the CPI is reduced by .75 x .0625 = .047, resulting in a load CPI adder of .25x 0.0625 = 0.016

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.

   For the 75% of loads that hit the L0, there is no impact, and CPI is reduced by 0.047 as in Problem 13. For the 25% that miss the L0, there are now two load delay slots. 65% of the time, both delay slots can be covered, leading to no CPI adders. 10% of the time, only one slot can be covered, while 25% of the time, neither can be covered. Hence, the CPI adder for loads is now 0.25 x (0.25 x (0.10 x 1 + 0.25 x 2)) = 0.0375.

Problem 4

10. Given the example code in Problem 1, 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.

    The first iteration accesses memory location &B[0], &C[1], and &A[0]. Unfortunately, since the arrays are consecutive in memory, these locations are exactly 8KB (1024 x 8B per double) apart. Hence, in a two-way set-associative cache they conflict, and the access to A[0] will evict B[0]. In the second iteration, the access to B[1] will evict C[1], and so on. However, since the access to C is offset by 1 double (8 bytes), in the eighth iteration (when i=7) it will access C[8], which does not conflict with B[7]. Hence, B[7] will hit, as will A[7]. In the ninth iteration, C[10] will also hit, but now B[8] and A[8] will again conflict, and no hits will result. Hence, there are three hits every eight iterations, leading to a total number of hits of floor(1000/8)*3 = 375 hits. The number of misses is 3000-375 = 2625, for an overall miss rate of 87.5% misses per reference.

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	7	7	18	512x(18 tag + 1 valid) = 9728 bits	64KB
D-cache	6	8	22	512x(22+1 valid + 1 dirty) = 12,288 bits	32KB
TLB	12	5	15	128x(15 tag + v) = 2048 bits	128x(24+d+r +3p) = 3712 bits

Problem 5

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?

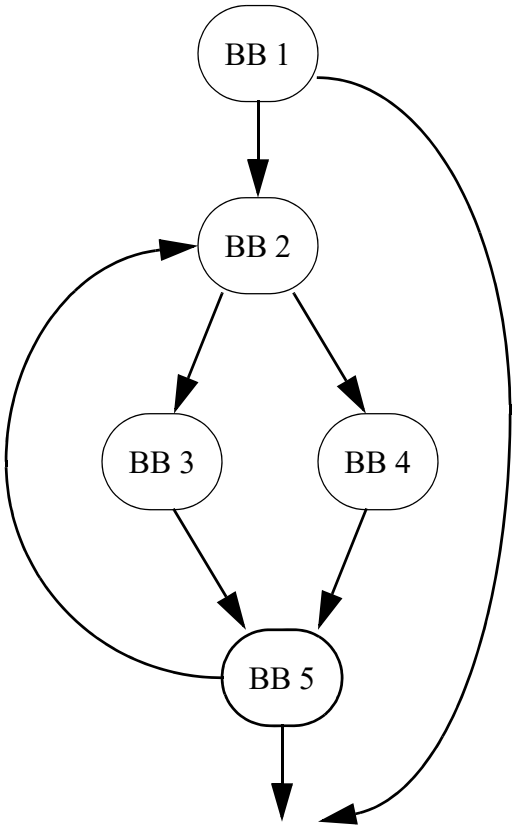
    For maximum throughput, each pipeline stage will contain four inflight instructions. Since registers are allocated at decode, and freed at retire, each instruction holds a rename register for a minimum of five cycles. Hence 4x5 = 20 rename registers are needed at minimum, assuming no data dependences or cache misses that would cause instructions to stall. Adding five extra stages would increase the minimum to 40 registers. Of course, the students should understand that this is a minimum that assumes no data dependences or cache misses. At the same time, it also assumes throughput of 4 IPC. Since few processors achieve 4 IPC on real programs due to data dependences, control dependences, and cache misses, this "minimum" may in fact be sufficient. The only reliable way to determine the right number of rename registers is a sensitivity study using detailed simulation of a range of rename registers to find the knee in the performance curve.

Problem 6

14. Identify basic blocks:

BB#	1	2	3	4	5	6	7	8	9
Instr. #s	1-6	7-9	10-11	12	13-17				

15. Draw the control flow graph for this benchmark.



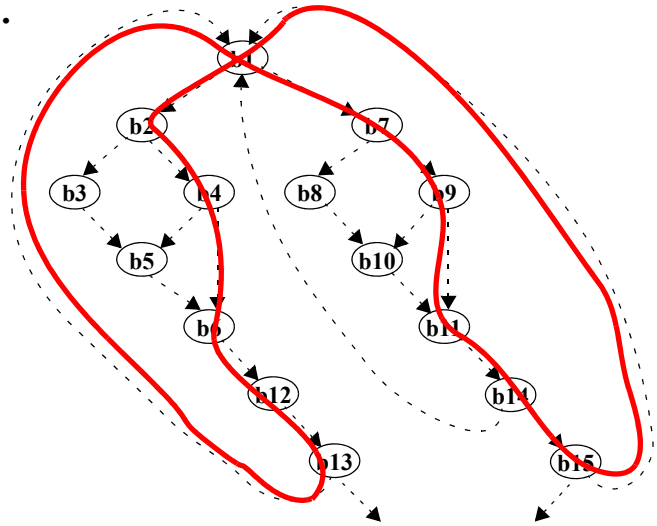
16. Assume that an one-bit (history bit) state machine (see above) is used as the prediction algorithm for predicting the execution of the two branches in this loop. Indicate the predicted and actual branch directions of the b1 and b2 branch instructions for each iteration of this loop. Assume initial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

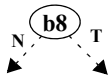
	8	9	10	11	12	20	29	30	31
b1 predicted:	__N	____T	___N	____T	___N	____T	___T	___N	____T__
b1 actual:	__T	___N	____T	___N	____T	___T	___N	____T	___N__
b2 predicted:	__N	____N	____N	____T	____N	____N	____T	____N	____T__
b2 actual:	__N	____N	____T	____N	____N	____T	____N	____T	___N__

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.





<u>Circle one</u>:

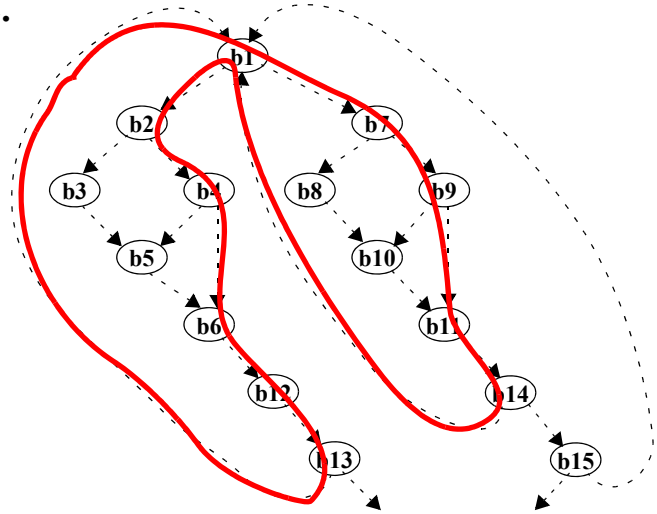
Bimodal

Local

(Gselect)

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

2.



<u>Circle one</u>:

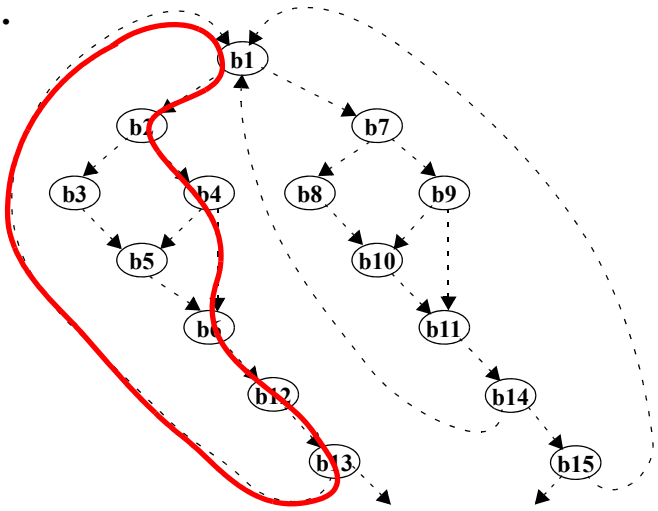
Bimodal

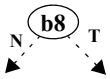
(Local)

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.