CMPE 382/CMPUT 429 - Computer Systems and Architecture
Midterm Exam — Winter 2002

Prof. José Nelson Amaral
Computing Science Department
University of Alberta

StudentID:	Name:

CMPE 382 CMPUT429

CMPE 383/CMPUT 429 Honor Code

By turning in the exam solution for grading, I certify that I have worked all the solutions on my own, that I have not copied or transcribed solutions from a classmate, someone outside the class, or from any other source. I also certify that I have not facilitated or allowed any of my classmates to copy my own solutions. I am aware that the violation of this honor code constitutes a breach of the trust granted me by the teaching staff, compromises my reputation, and subjects me to the penalties prescribed in Section 26.1 of the University of Alberta 2001/2002 Calendar.

Edmonton, February 26, 2002.

Question 1	(a)	/10	
	(b)	/10	
	(c)	/5	/25
Question 2	(a)	/10	
	(b)	/10	/20
Question 3	(a)	/10	
	(b)	/10	
	(c)	/10	/30
Question 4	(a)	/10	
	(b)	/10	
	(c)	/5	/25
		Total	/100
		Curving	/100
		Rank	

Benchmark	L1-I Miss Rate	L1-D Miss Rate
gcc	10.6	24.1
vortex	6.9	23.1
parser	0.4	30.7
gap	6.7	24.4
Perlbmk	14.1	28.7
Eon	15.1	55.6
Crafty	12.3	27.0
Mcf	0.1	46.9
Average	**8.3**	**32.6**

Table 1: Miss Rates measured by Zhao for SPEC2000 benchmarks in the Itanium.

Question 1 (25 points):

In the Compiler Design and Optimization Laboratory at the University of Alberta Peng Zhao, a Ph.D. student, is designing a method to improve inter-procedural code placement for the Intel Itanium processor. This improvement, called *partial inlining*, consists on identifying procedures that have a small portion of frequently executed instructions, called *hot code*, and a large portion of infrequently executed instructions, called *cold code*. Once such a procedure is identified, the procedure is split into two portions. The hot portion of the code is *inlined*, *i.e.* it is inserted directly in the place of the procedure call, and the cold portion of the code is kept into a separate procedure. The inlining of hot code allows the compiler to do a better placement of the program's instructions in the memory space, and thus enable the compiler to reduce the miss rate in the instruction cache. However, all the data accesses that were executed in the original code are still executed in the optimized code.

Zhao did a preliminary measurement of the cache performance for some of the benchmarks in the SPEC 2000 suite. These benchmarks were run in an Itanium machine with an 800 MHz processor. Zhao measured the miss ratio for the Itanium L1 Instruction cache and L1 Data cache. The results of this measurement are shown in Table 1.

The Itanium has a split L1 cache with a 4-way, 16 KB L1 instruction cache, and a 4-way, 16 KB, dual ported, write through, L1 data cache. Both of these caches are organized in 32-byte blocks and have an access time of 2 cycles. The Itanium also has an L2 unified cache (instruction and data) that stores 96 KB, has 64-byte blocks, write allocates, and has a hit time of 12 cycles. The Itanium's L3 cache is a 4 MB, on-package, off-chip cache, with 64-byte blocks, that has a hit time of 20 cycles.

Zhao is interested on an early estimate for the maximum speedup that his compiler improvement can possibly deliver in this machine. The number that he is looking for is an upper bound on the speedup that he can obtain by applying partial inlining to the benchmarks of Table 1. For the purpose of this analysis, he makes the reasonable assumptions that: (1) most of the execution time is spent in loops; and (2) when executing loops, all the instruction memory references will be found at most in the 4 MB L3 cache. By concentrating in loops, he is ignoring cold misses. He also decides to simplify the memory hierarchy by ignoring the L2 cache. Thus for his initial estimate, a memory reference is either found in the L1 cache and returned in two cycles, or it is found in the L3 cache and returned in 20 cycles.

a. (10 Points) If 25% of the instructions in the Itanium access memory, using the average miss rate measured by Zhao, what is the **maximum percentage reduction in the average memory access time (AMAT)** that Zhao can obtain using the partial inlining technique? Show your calculation.

b. (10 Points) For which of the benchmarks in Table 1 Zhao can *hope* to obtain the most reduction in the AMAT? First state what characteristic the benchmark should have to yield a high reduction in AMAT, then list some of the benchmarks of Table 1 where such high reduction *could* occur.

c. (5 Points) If Zhao were to take into consideration the effect of the L2 cache, the estimate for the maximum reduction in the AMAT would be higher or lower? Explain your answer.

Solution:

a. The main observation is that the optimization cannot change the data access pattern, it can only reduce the instruction miss rate. Thus in the best case scenario, this optimization could reduce the instruction cache miss rate to zero.

If we follow what is stated in the question, when an instruction or data is found in the L1 cache, it is returned in 2 cycles, when it has to be fetched from the L3 cache, it is returned in 20 cycles. Thus we can compute the AMAT as:

$$\begin{aligned}\text{AMAT} &= \text{HitRate}_{\text{Instr}} \times \text{HitTime}_{\text{Instr}} + \text{MissRate}_{\text{Instr}} \times \text{MissTime}_{\text{Instr}} + \\ &\quad \text{DataAccessRate} \times [\text{HitRate}_{\text{Data}} \times \text{HitTime}_{\text{Data}} + \text{MissRate}_{\text{Data}} \times \text{MissTime}_{\text{Data}}]\end{aligned}$$

Thus we can compute the AMAT before the improvement as:

$$\begin{aligned}\text{AMAT}_{\text{before}} &= (1 - 0.083) \times 2 + 0.083 \times 20 \\ &\quad +0.25 \times [(1 - 0.326) \times 2 + 0.326 \times 20] \quad \text{cycles} \\ &= 1.834 + 1.66 + 0.25 \times [1.348 + 6.52] \quad \text{cycles} \\ &= 5.461\end{aligned}$$

The AMAT after the improvement is similar, but the miss rate of the instruction cache is reduced to zero (in the best case scenario):

$$\begin{aligned}\text{AMAT}_{\text{after}} &= 1.0 \times 2 + 0.0 \times 20 \\ &\quad +0.25 \times [(1 - 0.326) \times 2 + 0.326 \times 20] \quad \text{cycles} \\ &= 2 + 0.0 + 0.25 \times [1.348 + 6.52] \quad \text{cycles} \\ &= 3.967\end{aligned}$$

Thus the maximum percentage reduction in the AMAT is:

$$\text{MaxPercReduc} = \left(\frac{\text{AMAT}_{\text{before}} - \text{AMAT}_{\text{after}}}{\text{AMAT}_{\text{before}}}\right) \times 100 = \left(\frac{5.461 - 3.967}{5.461}\right) \times 100 = 27.3\% \quad (1)$$

b. Zhao had the most hope to reduce the AMAT for benchmarks that have a high miss rate for the instruction cache (something that he can improve) and a low miss rate for the data cache (something that he cannot improve). Benchmarks `crafty`, `Perlbmk`, `gcc`, and `eon` are good candidates.

c. Considering L2 will reduce the original AMAT because the miss penalty of L1 will be reduced, therefore the effect of the optimization will be less important. Thus, considering L2, the reduction on AMAT will be smaller.

Old Addressing	New Addressing
ADD R1, R1, R2 LW Rd, 100(R1)	LW Rd, 100(R1)(R2)
ADD R3, R3, R4 SW 0(R3), Rs	SW 0(R3)(R4), Rs

Table 2: Code transformations to use new addressing mode.

Instruction	Percentage
load	26%
store	9%
add	14%
compare	14%
cond. branch	17%
shift	4%
and	3%
or	5%
others	8%

Table 3: Percentage of instructions of each class executed.

Question 2 (20 points): Consider adding a new index addressing mode to an existing architecture. This new addressing mode determines the effective address of the memory location by adding the content of two registers and an 11-bit signed offset.

The compiler is changed so that code transformations shown in Table 2 are applied as often as possible. Because of the short offset in the new addressing mode, it is not possible to apply the transformations shown on the table to all displacement load and stores.

a. (10 Points) Use the instruction frequencies of Table 3. If the new addressing mode can be used for 10% of the loads and stores, what is the **percentage reduction** in the number of instructions on the new machine compared with the old machine.

b. (10 Points) If in order to implement the new addressing mode, the designers have to lengthen the clock cycle by 5%. If the CPI does not change with this modification, which machine will be faster, and by how much?

Solution:

a. The only instructions that are eliminated are the ADD instructions that are associated with a load or a store when the pairs ADD/LW or ADD/LW of the old addressing mode are replaced by a single load or a single store in the new one. There are 35% load and stores (26% + 9%), and in only 10% of those the modification can be applied. Thus 3.5% of the instructions are eliminated.

   Another way to reach the same conclusion, is to assume that we start with a given number of instructions, say 1000 instructions, then we have:

$$\begin{aligned}
\text{NumInstructions}_{\text{Old}} &= 1000 \quad \text{instructions} \\
\text{NumInstructions}_{\text{New}} &= 1000 - [(0.26 + 0.09) \times 0.1] \times 1000 \\
&= 965 \quad \text{instructions} \\
\text{PercReduction} &= \left(\frac{\text{NumInstructions}_{\text{Old}} - \text{NumInstructions}_{\text{New}}}{\text{NumInstructions}_{\text{Old}}} \right) \times 100 = 3.5\%
\end{aligned}$$

(2)

b. The execution time is given by:

$$\text{Time} = \text{NumInstructions} \times \text{CPI} \times \text{ClockCycle}$$

The part (b) states that $\text{ClockCycle}_{\text{New}} = 1.05 \times \text{ClockCycle}_{\text{Old}}$. Thus we have:

$$\begin{aligned} \text{Time}_{\text{Old}} &= 1000 \times \text{CPI} \times \text{ClockCycle}_{\text{Old}} \\ \text{Time}_{\text{New}} &= 965\text{CPI} \times 1.05 \times \text{ClockCycle}_{\text{Old}} \\ \text{Time}_{\text{New}} &= 1013.25 \times \text{CPI} \times \text{ClockCycle}_{\text{Old}} \end{aligned}$$

To compute speedup between two machines we divide the time of the slowest machine by the time of the faster machine, thus:

$$\begin{aligned} \text{Speedup} &= \frac{\text{Time}_{\text{New}}}{\text{Time}_{\text{Old}}} \\ &= \frac{1013.25 \times \text{CPI} \times \text{ClockCycle}_{\text{Old}}}{1000 \times \text{CPI} \times \text{ClockCycle}_{\text{Old}}} \\ &= 1.01325 \end{aligned} \tag{3}$$

Thus the **Old** machine is 1.013 times, or 1.3%, **faster** than the **New** machine.

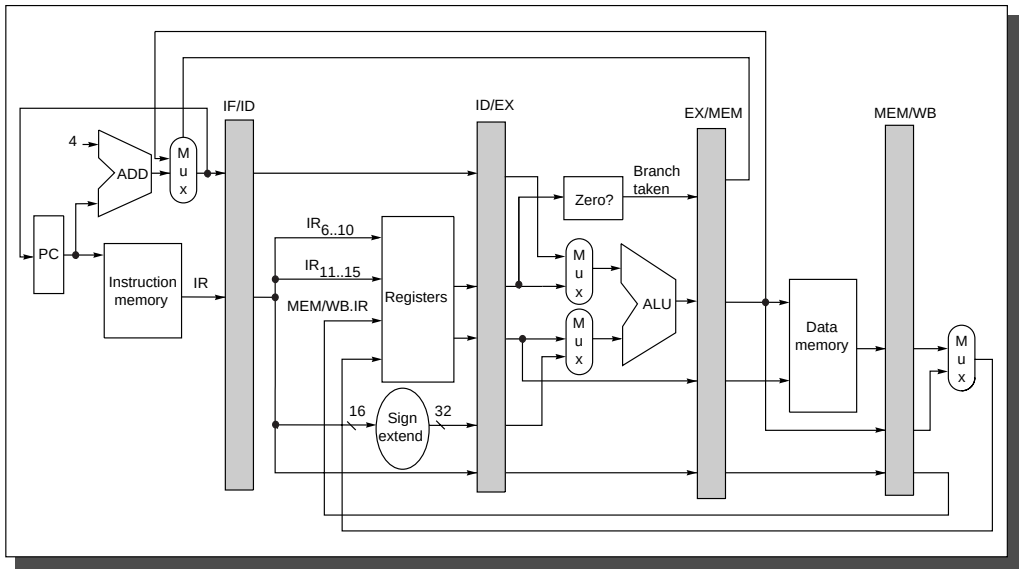

FIGURE 3.4 The datapath is pipelined by adding a set of registers, one between each pair of pipe stages.

Question 3 (30 points):

Assume a machine that implements the pipelined datapath shown in Figure 3.4. The five stages of this pipeline are summarized in Figure 3.3, where IF is the stage in which instructions are fetched from the instruction memory, ID is the instruction decoding stage where the source registers of an instruction are read, the EX (execution) stage performs arithmetic operations or address computations, DM accesses the data memory, and the WB is the write back stage where the result of the operation is written into the register file. Consider the loop code shown in Figure 1.

a. (10) Assume that each pipeline in Figure 3.3 represents the execution of one of the instructions of the loop, starting with the first load instruction associated with the top pipeline representation in the figure. Draw, in Figure 3.3, lines between the output of a stage and the input of another stage to represent data dependences in the loop code. Indicate in the figure only dependences found among the first 5 instructions of the loop code.

b. (10) Assume that this pipeline machine allows no bypassing (not even from the WB to the ID stage), *i.e.* it is not possible to write to a register and read from the same register in the same clock cycle. In Table 4, for each one of the first 32 cycles of the schedule show what stage of which instructions is executed in that cycle.You should schedule each instruction as early as possible as long as you do not violate any data dependence in the code. Assume that branches are always predicted as not taken, and that a branch is resolved in the EX stage, *i.e.*, at the end of the EX state we know what is the next instruction to be executed. To represent your schedule in Table 4 use the notation IF, ID, EX, DM, and WB to abbreviate the name of the pipeline stages.

c. (10) You want to redesign the pipeline to allow forwarding from the output of the ALU at the end of the execution stage to the input of the ALU, and from the data memory to the input of the ALU. Draw in Figure 3.4 the new datapath lines that are required to enable these data forwarding. What is the speedup that this improvement in the design will result for the loop of Figure 1?

Solution

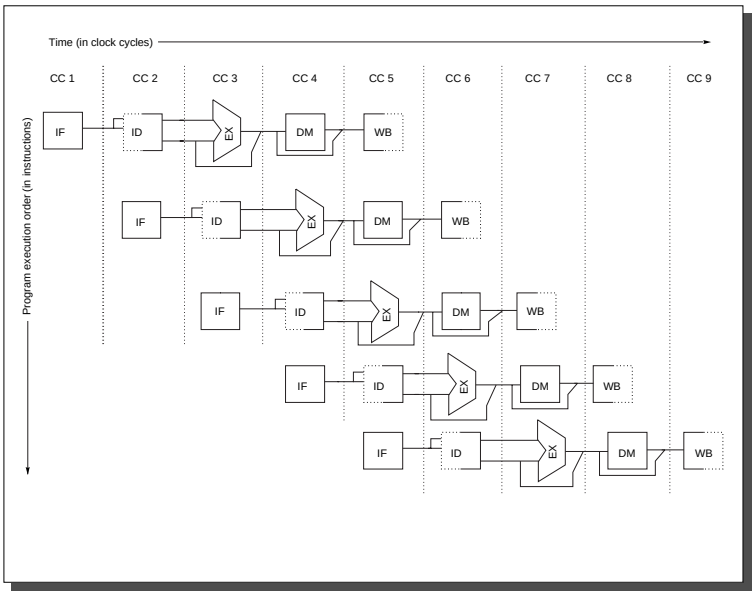

FIGURE 3.3 The pipeline can be thought of as a series of datapaths shifted in time.

```
LOOP:   LW      R1,0(R2)
        LW      R3,8(R2)
        ADD     R4, R1, R3
        SW      0(R2), R4
        SUB     R2, R2, #4
        BNEZ    R2, LOOP
        OR      R7, R8, R9
        AND     R8, R7, R2
```

Figure 1: A simple loop code.

	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
LW R1,0(R2)	IF	ID	EX	DM	WB											
LW R3,8(R2)		IF	ID	EX	DM	WM										
ADD R4, R1, R3			IF	S	S	S	ID	EX	DM	WB						
SW 0(R2), R4							IF	S	S	S	ID	EX	DM	WB		
SUB R2, R2,#4											IF	ID	EX	DM	WB	
BNEZ R2, LOOP												IF	S	S	S	ID
OR R7, R8, R9																IF
AND R8, R7, R2																
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
LW R1,0(R2)		IF	ID	EX	DM	WB										
LW R3,8(R2)			IF	ID	EX	DM	WB									
ADD R4, R1, R3				IF	S	S	S	ID	EX	DM	WB					
SW 0(R2), R4								IF	S	S	S	ID	EX	DM	WB	
SUB R2, R2,#4												IF	ID	EX	DM	WB
BNEZ R2, LOOP	EX	WB											IF	S	S	S
OR R7, R8, R9	ID															
AND R8, R7, R2	IF															

Table 4: Pipeline Schedule.

a. In the first five instructions the dependences are from the two loads to the ADD (R1, and R3), and from the ADD to the SW (R4). Thus, you needed to the following lines:

- from the output of DM in the first pipeline [LW R1,0(R2)] to one of the ALU inputs in the third pipeline [ADD R4, R1, R3].
- from the output of DM in the first pipeline [LW R3,8(R2)] to the other ALU inputs in the third pipeline [ADD R4, R1, R3].
- from the output of the ALU in the third pipeline [ADD R4, R1, R3] to the lower input of the ALU in the fourth pipeline [SW 0(R2), R4].

b. See Table 4.

c. For the new dapath lines, see Fig. 3.20, page 161 of the Textbook. The original loop executes in 17 cycles (number of cycles between the IF of the first load in the first iteration, and the IF of the same load in the second iteration. The forwarding that you allow in your new design will wliminate the stalls in cycles 05, 07, 08, 09, 12, 13, 14 (note that the stalls in cycles 03 and 04 cannot be eliminated by forwarding because the value needed is not yet available in those cycles. Thus the new pipeline structure will require $17 - 7 = 10$ cycles to execute. And the speedup will be:

$$\text{Speedup} = \frac{\text{Time}_{\text{Old}}}{\text{Time}_{\text{New}}} = \frac{17}{10} = 1.7 \tag{4}$$

Field	Address Bit Field
Tag	63-16
Cache Index	15-7
Sector Offset	6-5
Word Offset	4-3

Table 5: Address Fields used in the L1 Instruction Cache of the POWER4.

Question 4 (25 points):

In this question we will study the design decision for the L1 Instruction Cache of the new IBM POWER4 microprocessor. This processor has a direct mapped 64KB instruction cache with 128-byte blocks, each block divided into 32-byte sector. We will assume that this division of the blocks into sectors is equivalent to the traditional division of blocks into sub-blocks presented in our textbook as a technique to reduce miss penalty. Therefore we assume that this cache is capable of sub-block (or sector in IBM's terminology) placement. In other words, there is a valid bit associated with each sector to allow some sectors to be valid and some to be invalid within the same block. Likewise, we assume that there is a dirty bit associated with each sector to allow some sectors to be dirty while others are clean within a block.

a. (5 Points) Assuming 64 bit instruction words, how many words can be stored in this instruction cache?

b. (10 Points) With the division of blocks into sectors, we have a sector offset to determine in which sector of the block the instruction can be found, and a word offset to find the word within the sector. Complete Table 5 with the bits in a 64 bit instruction address that are used for tag, index, and offset (use the notation X-Y to indicate a bit field that starts in bit X and ends in bit Y, inclusive). The address bits are number from 63 to 0. The most significant bit is 63 and the least significant bit is 0.

c. (10 Points) The technique of dividing a cache block into sub-blocks is used to reduce the storage required for tags in a cache without requiring a large amount of data to be brought into the cache for each miss. If we compare this IBM design with an alternative design in which we use a cache with the same capacity (64 KB) but use blocks of 32-bytes instead (without sub-dividing these blocks), how much larger is the area required to store the tags of this alternative design when compared with the POWER4 L1 cache? Explain.

Solution:

a. Is it a 64 KB cache, and each word has 64 bits = 8 bytes, thus we have:

$$\text{NumWords} = \frac{CacheSize}{WordSize} = \frac{64Kbytes}{8bytes} = 8Kwords = 8 \times 1024words \tag{5}$$

b. Each block has 128 bytes, and the cache stores 64 Kbytes, therefore there are $(64K/128) = 512 = 2^9$ blocks in the cache, and we need 9 address bits to index these 512 blocks.

Each block sector has 32 bytes, and each block has 128 bytes, thus there are $(128/32) = 4 = 2^2$ sectors in each block, and we need 2 address bits to find the sector inside the block.

Each sector has 32 bytes, and each word has 64 bits = 8 bytes, thus there are $((32/8) = 4 = 2^2$ words in each sector, and we need 2 address bits to index the word within the sector.

Each word has $8 = 2^3$ bytes, thus we need three address bits to index the byte within a word.

c. If the cache size and the cache associativity do not change, I will still need the same number of bits for each tag. However with the alternative design the sectors in the IBM design become blocks, and I will need one tag for each block. Because there is 4 sectors in each block in the original design, I will need **four** times as many tags, and thus four times as much area in the processor to store this tags when compared with the original area dedicated for tag storage

.