CPSC 418	Advanced Computer Architecture	Spring 1995

Homework 10 Solution

30 Mar

Problem 1 (0 Points)

1. Pipelines
   (a) definition
   Divide the processing of instructions into stages. Overlap processing of multiple instructions by sending next instruction into first stage as soon as previous instruction is done in first stage.
   (b) how does pipelining improve the performance of a microprocessor
   By overlapping processing of instructions, no single instruction finishes in any less time, but instructions are issued more frequently (with less delay between subsequent issues).
   (c) give upper and lower bounds on the number of stages in typical microprocessors' pipeline
   5 min, 12 max
   (d) why not have fewer or more stages in a pipeline?
   Fewer stages would increase clock cycle because more circuitry within each stage. More stages would means that too high of a percentage of the clock cycle would be spent in latches and other control circuitry; also, this would increase the latency of instructions — causing more data hazards and stalls.
2. Superscalar
   (a) definition
   Issue multiple instructions per cycle.
   (b) how does "superscalar-ing" improve the performance of a microprocessor
   More instructions are processed in parallel.
   (c) give upper and lower bounds on the "superscalar-edness" in typical microprocessors
   2 – 4 instructions / cycle.
   (d) why not have more or less "superscalar-edness" in a microprocessor?
   Fewer instructions per cycle would not fully utilize the functional units that can be put on a microprocessor. More instructions would require more functional units than can be put on a microprocessor; also, there are fundamental limits to the amount of instruction level parallelism in programs.
3. Caches
   (a) definition
   Store currently active code and data in small fast memory.
   (b) how do caches improve the performance of a microprocessor
   Instruction fetch and data accesses take less time.
   (c) give upper and lower bounds on the number and sizes of caches in typical microprocessors
   1 or 2 levels of cache, plus main memory, plus virtual memory.
   (d) what design decisions must be made in designing a cache system
   Cache size, associativity, replacement policy, write-scheme.

(e) why not have more or bigger caches in a microprocessor?
Given the current technology increasing the cache size would be very expensive, but not yield large performance gains. (current caches give very good performance)

4. Register renaming

(a) definition
Utilize more physical registers than architected registers.

(b) how does register renaming improve the performance of a microprocessor
Eliminates many WAR and WAW hazards in code.

(c) what design decisions must be made in designing a register renaming system
Number of physical registers, Freelist scheme.

5. Multiple / Duplicate functional units

(a) give an example of a typical configuration for a microprocessor with multiple and/or duplicate functional units
2 integer units with overlapped functionality. 2 floating-point units with disjoint functionality. 1 load/store unit. 1 branch unit.

(b) how do multiple and/or duplicate functional units improve the performance of a microprocessor
Increases the number of instructions that can be processed in parallel.

(c) what design decisions must be made in designing a microprocessor with duplicate and/or multiple functional units
Number of units and what each one does; plus design each unit.

Problem 2 (0 Points) You are considering three options for a new microprocessor design. Assuming that average performance doubles each year, decide which option results in the best price/performance ratio for the microprocessor at the time it will be introduced on the market. Assume that all of the options add the same cost to the product.

Design decisions already made:

Generic RISC instruction set (*e.g.* DLX).

Single issue. (With current design, CPI is predicted to be = 1.3, *ignoring all stalls caused by memory accesses and control hazards.*)

An on-chip 4k instruction cache: 2-way set associative with 32 byte blocks. Access time for a hit is 1 cycle. On a cache miss, the entire block must be read in from main memory before the instruction is available.

There is one dual-word read/write port on the main memory and one dual-word read port. The read/write port is connected to the load/store unit of the CPU and the read port is connected to the instruction cache. The latency for accessing a single or double word in main memory is 100 ns. A 32 byte block can be transferred to or from main memory in 180 ns.

Clock speed = 10ns / cycle.

Statically predict branches as not taken. Branch targets are calculated in the Decode (second) stage. Branch taken/not-taken determined in the Fetch Operands (third) stage.

Options:

small on chip caches Upgrade the 4k instruction cache to 8k and add an 8k data cache. Both caches are 2-way set associative and have 32 byte blocks. The data cache is write-through and at any time 50% of the blocks are dirty. Access latency = 1 cycle for hits.

branch prediction Add a branch-target buffer (BTB) for conditional branches only. Buffer hit rate = 85%; when a miss occurs the branch is predicted as not-taken and execution proceeds as without the BTB. Prediction correctness for branches that hit in BTB = 90%. The additional control circuitry required for the branch-target buffer will cause the clock cycle to increase to 12ns. Additional development time = 3 months.

register-renaming Adding register renaming can cut the average CPI (not including memory or control stalls) from 1.3 to 1.15. Additional development time = 6 months.

From Fig 8.12:	*8k, 2 way set assoc cache*	*%Miss = 0.069*
	4k, 2 way set assoc cache	*%Miss = 0.095*

	%LD	*18*	*load instructions*
	%ST	*8*	*store instructions*
From Fig C.4:	*%L/S*	*26*	*load and store instructions*
	%COND	*11*	*conditional branch instructions*
	%UNCOND	*2*	*unconditional branch instructions*

From table of instruction distributions, we calculate for memory accesses:

%Write	*18 / 26 = 0.692*
%Read	*8 / 26 = 0.308*

From Fig 3.22: *%Taken* *0.53* *percentage of conditional branches that are taken*

From problem statement:

number of cycles to transfer one or two words between L1 cache and Main Memory:

$$\begin{aligned} T_M &= 100ns \ / \ 10(ns/cycle) \\ &= 10 \ cycles \end{aligned}$$

number of cycles to transfer block between L1 cache and Main Memory:

$$\begin{aligned} T_{MB} &= 180ns \ / \ 10(ns/cycle) \\ &= 18 \ cycles \end{aligned}$$

Average access time for instructions:

$$\begin{aligned} TA_{inst} &= (\%Hit)(T_{L1}) + (\%Miss)(T_{MB}) \\ &= (1 - 0.095)(\ 1\ ) + (0.095)(18) \\ &= 2.62 \end{aligned}$$

Average access time for data:

$$\begin{aligned} TA_{data} &= T_M \ (no\ data\ cache\ on\ base\ machine) \\ &= 10 \end{aligned}$$

Memory stalls per instruction:

$$\begin{aligned} MemStalls &= (TA_{inst} - 1) + (\%L/S)(TA_{data} - 1) \\ &= (2.62 - 1) + (0.26)(10 - 1) \\ &= 3.95 \end{aligned}$$

Branch stalls per instruction:

> *Statically predict branches as not taken, if mispredict get two cycle penalty, because taken/not-taken determined in third stage. Branch target determined in second stage, so get one penalty cycle for unconditional branches.*

$$\begin{aligned} BraStalls &= (\%COND)(\%TAKEN)(2) + (\%UNCOND)(1) \\ &= (0.11)(0.53)(2) + (0.02)(1) \\ &= 0.14 \end{aligned}$$

CPI of base machine:

$$\begin{aligned} CPI &= IdealCPI_{pipe} + MemStalls + BraStalls \\ &= 1.3 + 3.95 + 0.14 \\ &= 5.39 \end{aligned}$$

Nanoseconds per instruction:

$$\begin{aligned} NSPI &= (5.39)(10) \\ &= 53.9\ ns/inst \end{aligned}$$

Cache Option

*We will assume that the processor has write-buffer (*NOTE: this information should have been given in the problem statement*)*

Average access time for instructions:

$$\begin{aligned} TA_{inst} &= (\%Hit)(T_{L1}) + (\%Miss)(T_{MB}) \\ &= (1 - 0.069)(\ 1\ ) + (0.069)(18) \\ &= 2.17 \end{aligned}$$

Average access time for data:

$$\begin{aligned} TA_{data} &= (\%Read)(\%Hit)(T_{L1}) + (\%Read)(\%Miss)(T_{MB}) + (\%Write)(T_{L1}) \\ &= (0.692)(1 - 0.069)(1) + (0.692)(0.069)(18) + (0.308)(1) \\ &= 1.43 \end{aligned}$$

Memory stalls per instruction:

$$\begin{aligned} MemStalls &= (TA_{inst} - 1) + (\%L/S)(TA_{data} - 1) \\ &= (2.17 - 1) + (0.26)(1.43 - 1) \\ &= 1.28 \end{aligned}$$

BraStalls = 0.14 (same as with base machine).

CPI of machine with cache option:

$$\begin{aligned} CPI &= IdealCPI_{pipe} + MemStalls + BraStalls \\ &= 1.3 + 1.28 + 0.14 \\ &= 2.72 \end{aligned}$$

Nanoseconds per instruction:

$$NSPI = (2.72)(10) = 27.2\ ns/inst$$

Machine with cache is n% faster than base machine:

$$n = (NSPI_{Base} - NSPI) \; / \; NSPI = (53.9 - 27.2) \; / \; 27.2 = 98\%$$

Branch Target Buffer Option

Assume that branches that miss in the BTB are predicted as not-taken. (NOTE: this information should have been given in the problem statement)

Stalls per conditinal branch instruction:

$$CondStalls = (\%Hit)(\%PredOk)(0) + (\%Hit)(1 - \%PredOk)(2) + (\%Miss)(\%NotTaken)(0) + (\%Miss)(\%Taken)(2)$$
$$= (0.85)(0.90)(0) + (0.85)(0.10)(2) + (0.15)(0.47)(0) + (0.15)(0.53)(2)$$
$$= 0.33$$

Branch stalls per instruction:

$$BraStalls = (\%COND)(CondStalls) + (\%UNCOND)(1) = (0.11)(0.33) + (0.02)(1) = 0.056$$

MemStalls = 3.95 (same as with base machine).

CPI of machine with BTB:

$$CPI = IdealCPI_{pipe} + MemStalls + BraStalls = 1.3 + 3.95 + 0.056 = 5.31$$

Nanoseconds per instruction:

$$NSPI = (5.31)(12) = 63.7\ ns/inst$$

Performance relative to other machines at time of introduction:

Option is n% faster than Base $= (TimeBase - TimeOption)/(TimeOption)$

In order to keep up with the market, the performance of the machine with the options must be $(2^{ScheduleDelay})$ *faster than the base machine.*

We can compare the performance of the various options by exponentially scaling down the time per instruction for the base machine.

$$
\begin{aligned}
NSPI_{Base}' &= (NSPI_{Base})/(2^{ScheduleDelay}) \\
&= 53.9 \ / \ (2^{0.25}) \\
&= 45.3
\end{aligned}
$$

At time of release, machine with BTB will be n% faster than $NSPI_{Base}'$*:*

$$
\begin{aligned}
n &= (NSPI_{Base}' - NSPI) \ / \ NSPI \\
&= (45.3 - 63.7) \ / \ 63.7 \\
&= -0.29\%
\end{aligned}
$$

Register Renaming Option

MemStalls = 3.95 (same as with base machine).

BraStalls = 0.14 (same as with base machine).

CPI of machine with register renaming:

$$
\begin{aligned}
CPI &= IdealCPI\_RR_{pipe} + MemStalls + BraStalls \\
&= 1.15 + 3.95 + 0.14 \\
&= 5.24
\end{aligned}
$$

Nanoseconds per instruction:

$$
\begin{aligned}
NSPI &= (5.24)(10) \\
&= 52.4 \ ns/inst
\end{aligned}
$$

$$
\begin{aligned}
NSPI_{Base}' &= (NSPI_{Base})/(2^{ScheduleDelay}) \\
&= 53.9 \ / \ (2^{0.5}) \\
&= 38.1
\end{aligned}
$$

At time of release, machine with Register Renaming will be n% faster than $NSPI_{Base}'$*:*

$$
\begin{aligned}
n &= (NSPI_{Base}' - NSPI) \ / \ NSPI \\
&= (38.1 - 52.4) \ / \ 52.4 \\
&= -0.27\%
\end{aligned}
$$

Conclusion

The best option is the cache enhancement. The next best option is Register Renaming, and then the Branch Target Buffer.

Problem 3 (0 Points) What are the key differences between scoreboards, reservations stations, and register renaming?

Scoreboards stall an instruction in the instruction pipe until the functional unit is available. Also, both operands for an instruction are fetched at the same time (when the latter one becomes valid). The first difference causes structural hazards and the second one causes WAW and WAR hazards.

The key difference between reservation stations and register renaming is that in reservations stations, instructions are identified by the reservation station that the instruction resided in while waiting for its operands and functional unit. In register renaming, instructions are identified by the physical register that the result will be written to. This difference separates the queuing of instructions before execution from the identification of results, which simplifies the design and eliminates some stalls caused by instructions waiting for reservation stations to become available (see handout on reservation stations).