CE-4
August 2010 QE

1 Processor (22 points)

(a) (10 points) Modern processors employ register renaming to remove WAR and WAW hazards. With increasing clock speeds, the rename stage may not fit within the cycle time. To address this issue, one idea may be to pipeline the rename stage. Explain the key architectural problem due to which this idea is likely to degrade performance.

Please restrict your answer to 4-5 sentences.

Pipelined renaming will incur stalls due to RAW hazards which are common.

(b) (12 points) A processor runs a suite of applications which, averaged over time, keep utilized about 80% of the 10 MSHRs at the data cache (MSHRs are miss status holding registers used in lock-up free caches).

Assume that the applications achieve an **ideal** CPI (cycles per instruction) of 0.5 if branch mispredictions and cache misses are ignored (the ideal CPI accounts for both correct branch predictions and cache hits). Assume that in the applications (1) every 6th instruction is a branch and the branch misprediction rate and penalty are 10% and 24 cycles, respectively, and (2) every 5th instruction is a load and the data cache miss rate and penalty are 1% (0.01) and 500 cycles, respectively.

Compute the **real** CPI which includes branch and cache effects. Ignore any other issue not covered above.

Hint: What is a key effect of multiple misses being held in MSHRs?

8 MSHRs are utilized on average => every 8 misses see 1 miss latency => real CPI =

$0.5 + 0.1*1/6*24 + 1/5*0.01*500*1/8 = 0.9+0.125 = 1.025$

Write in Exam Book Only

2 Memory Subsystem (28 points)

(a) (8 points) What key property of memory accesses is exploited by coalescing line buffers (or FIFOs) to improve the L1 bandwidth? Generic properties such as temporal/spatial locality will not receive any credit.

Many L1 accesses in the same cycle go to the same cache block.

(b) (20 points) Consider data sharing among multiple processes achieved via virtual memory where the processes' virtual pages that are supposed to be shared point to the same physical page. Now, our goal is to look up the L1 without waiting for the TLB. Note that the shared data need **not** have the same virtual address in different processes.

(i) (6 points) What is the problem with using the virtual address to index into the L1 to achieve our goal? The virtual address synonym problem is **not** relevant to the above data sharing problem.

Different virtual addresses will have different indices for the same shared data.

(ii) (8 points) To achieve our goal, describe a simple, software-only solution that does **not** constrain the virtual memory mapping (i.e., the mapping from virtual address to physical address) in any way. Also, your solution should not change the L1 cache in any way. Your solution **may** incur some performance loss.

Flush the L1 cache upon context switch.

(6 points) Why would this solution work for non-SMT processors but not for SMT processors (SMT refers to simultaneous multithreading)?

In non-SMT processors, only one process can run at a time whereas in SMT processors, two or more data-sharing processes may run together in which case there is a potential context switch at every instruction fetch, making the above solution useless.

3 Multicore (22 points)

(a) (12 points) Assume a bus-based multicore that uses the standard three-state protocol (Invalid, Modified, Shared). The following are **all** the bus transactions for block *A* occuring one after another over some time period: a *bus read* from processor P7 (event #1), a *bus upgrade for write* from processor P1 (event #2), and a *bus read* by processor P2 (event #3).

(i) (6 points) What is the state of the block *A* in P1's and P7's caches just before event #1?

P1 - Shared

P7 - Invalid

(ii) (6 points) What is the state of the block *A* in all the processors' caches after event #3?

P1 - Shared

P2 - Shared

other processors - Invalid

(b) (10 points) Consider the following code:

```
                              a = b = 0;    /*  initial values  */
```

processor #1	**processor #2**
`a = 1;`	`b = 1;`
`print b;`	`print a;`
`a = 5;`	

If processor #1 prints 0, what value(s) printed by processor #2 is (are) sequentially consistent? You must list **all** such values.

1 and 5 are SC.

4 Fundamentals (28 points)

Assume that a commonly-used business application employs multiple, parallel processes for high performance (these are OS-visible processes). While increasing the number of processes increases parallelism, each process consumes a non-trivial amount of physical memory which is limited in size.

(a)

(i) (5 points) What is the key performance trade-off for this application?

More processes increase parallelism but put pressure on memory which will cause thrashing and decrease performance when memory capacity is exceeded.

(ii) (8 points) Based on this trade-off, how would you optimize the system performance?

Spawn new processes as long as memory capacity is not exceeded.

(b) (15 points) Experiments find that most of the data (e.g., > 80%) in the processes are read-only and hence, identical across the processes; the rest of the data are written to and different across the processes. Further assume that the data addresses are identical across the processes.

(i) (6 points) Based on the above observation, what would be your basic strategy to improve the system performance beyond the trade-off point in part (a) above?

Use shared virtual memory and have the processes' virtual pages at the same virtual address point to the same physical page. This will allow more processes without consuming memory.

(ii) (9 points) How would you handle read-write data? Assume that read-write data **cannot** be identified statically (i.e., read-only and read-write data cannot be separated statically).

Start off all pages to be shared and read-only. Upon a write, throw an fault and create a private copy in physical memory for the writing process .