1 Processor (25 points)

(a) (10 points) Register forwarding avoids stalls due to RAW (read-after-write) hazards. While we expect forwarding to improve processor performance to some extent, why does forwarding achieve such large improvements that it is used extensively in modern pipelines?

Please restrict your answer to 1-2 sentences.

(b) (15 points) A processor achieves an ideal CPI (cycles per instruction) of 0.5 if branches and cache misses are ignored (ideal CPI accounts for on-chip cache hit latency). Assume that (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 memory operation and the off-chip cache miss rate and penalty are 0.5% (0.005) and 500 cycles, respectively. Further assume that for half of the mispredicted branches, the refilling of the pipeline upon prediction verification occurs while an off-chip miss is underway.

Compute the real CPI which includes branch and cache effects. Answers that ignore the last condition will receive no points.

2 Memory Subsystem (25 points)

(a) (9 points) Victim caches are often used as a technique to reduce conflict misses in a cache. Why are victim caches unlikely to benefit TLB performance?

(b) (16 points) List at least two distinct advantages and two distinct disadvantages of using large virtual memory pages. Keep in mind that page size is not used to shrink the page table in physical memory.

3 Multicore (25 points)

(a) (12 points) A multicore uses a simple write invalidate protocol which encodes the number of cache block copies with three states, namely exactly-one, more-than-one, and unknown. Assume the following sequence of events: (1) Processors P1 and P2 issue reads to a cache block, (2) P1 writes to the block, and (3) P2 reads the block.

Write the state of the block in P1’s and P2’s caches after each of the above events (2 states x 3 events).

(b) (13 points) Assume that test&set lock synchronization is implemented as follows:
test&set(lock) {
    loop: ld r2, lock ;; load lock into r2
    bnez r2, loop ;; if r2 is non-zero, loop back
    st lock, #1 ;; store 1 into lock
    return
}

(i) (6 points) Explain exactly why the above code is incorrect?

(ii) (7 points) What hardware support is needed to fix the problem? You need not provide the correct code with the hardware support, simply stating the hardware support is sufficient.

4 Fundamentals (25 points)

Purdue ECE nanotechnology has developed a new technology, called FAST-LOGIC, which is 100x faster than conventional CMOS. Unfortunately, FAST-LOGIC is unreliable in that it “zeroes-out” all on-chip memory structures such as registers and caches at unpredictable times. However, because all memory structures are zeroed out, it is easy to detect when a failure has occurred (but the damage is already done). For instance, a latch initialized to all 1’s at boot time can generate an interrupt when the latch is zeroed out.

Because FAST-LOGIC is a replacement for CMOS and not for DRAM technology, main memory will still use DRAM. Note that DRAM will be much slower than FAST-LOGIC and cannot replace on-chip registers and caches.

(a) (12 points) For computer systems designed using FAST-LOGIC,

(i) (6 points) what would be your basic strategy to handle the unpredictable failures?

(ii) (6 points) what is the key trade-off involved in the strategy?

(b) (13 points) Think of the key runtime overhead incurred by your strategy when there are no failures. If applications exhibit a certain write characteristic (as in stores to memory), a certain design choice for FAST-LOGIC caches would reduce this overhead.

(i) (8 points) What write characteristic is relevant here and what does the characteristic have to be to allow overhead reduction? Unrealistic applications that never write will receive no points.

(ii) (5 points) What cache design choice will reduce the overhead?