

COMPUTER ARCHITECTURE & SYSTEMS—Ph.D. Qualifying Exam Spring 2009

PRELIMINARY: Questions 1, 2, and 3 refer to the following two programs, which both do the same thing—the second program is just a modified (“loop-unrolled”) version of the first.

r1: points to the beginning of vector A
r2: points just past the end of vector A
r3 and higher: used as scratch registers

PROGRAM FOO:

```
top:  lb    r3, r1, 0    // load byte into r3 from memory location (r1+0)
      lb    r4, r1, 1    // load byte into r4 from memory location (r1+1)
      sb    r3, r1, 1    // store byte from r3 into memory location (r1+1)
      sb    r4, r1, 0    // store byte from r4 into memory location (r1+0)
      addi  r1, r1, 2    // increment A pointer by 2
      bne  r1, r2, top   // branch to "top" if A pointer is still valid
      exit                               // terminate program if A pointer is beyond end
```

PROGRAM BAR:

```
top:  lb    r3, r1, 0    // load byte into r3 from memory location (r1+0)
      lb    r4, r1, 1    // load byte into r4 from memory location (r1+1)
      lb    r5, r1, 2    // load byte into r5 from memory location (r1+2)
      lb    r6, r1, 3    // load byte into r6 from memory location (r1+3)
      lb    r7, r1, 4    // load byte into r7 from memory location (r1+4)
      lb    r8, r1, 5    // load byte into r8 from memory location (r1+5)
      lb    r9, r1, 6    // load byte into r9 from memory location (r1+6)
      lb    r10, r1, 7   // load byte into r10 from memory location (r1+7)
      sb    r3, r1, 1    // store byte from r3 into memory location (r1+1)
      sb    r4, r1, 0    // store byte from r4 into memory location (r1+0)
      sb    r5, r1, 3    // store byte from r5 into memory location (r1+3)
      sb    r6, r1, 2    // store byte from r6 into memory location (r1+2)
      sb    r7, r1, 5    // store byte from r7 into memory location (r1+5)
      sb    r8, r1, 4    // store byte from r8 into memory location (r1+4)
      sb    r9, r1, 7    // store byte from r9 into memory location (r1+7)
      sb    r10, r1, 6   // store byte from r10 into memory location (r1+6)
      addi  r1, r1, 8    // increment A pointer by 8
      bne  r1, r2, top   // branch to "top" if A pointer is still valid
      exit                               // terminate program if A pointer is beyond end
```

(i). Application Behavior & Performance (6 points total)

A. (2pts) What do programs FOO and BAR do (i.e., what function/operation do they perform)?

B. (4pts) Compare the performance of the two loops (i.e., “A is X% faster than B”), as measured on a single-cycle, non-pipelined machine (one in which all instructions take 1 cycle).

(ii). Pipelines (7 points total)

A. (2pts) What is pipelining? What is the benefit of pipelining a microprocessor?

B. (5pts) Calculate the performance of FOO and BAR on a pipelined processor. **Memory-load operations** have a latency of **6 cycles**—there must be *five* intervening cycles between a load instruction and a following dependent instruction. You may assume perfect branch prediction. Give the running times for the two programs in the number of *cycles* required to execute. Assume the vector length is exactly *two* million bytes. You may ignore the cost of the “exit” instruction at the end of execution.

(iii). Caches (7 points total)

A. (4pts) Illustrate the workings of a processor cache; draw a diagram of a two-way set associative cache, including the tag and data arrays and the input address. Be sure to indicate set selection, tag match, way selection, word selection, and cache hit/miss determination.

B. (3pts) Consider the following cache parameters: *cache size*, *cache block size*, *degree of associativity*, and *replacement strategy*. Which of these will have the greatest effect on the **data-cache** performance of programs FOO and BAR, and why?