

**COMPUTER ARCHITECTURE AND SYSTEMS — Ph.D. Qualifying Exam Fall 2005**

## 1. Number representation (8 points)

Consider a sign and magnitude floating point representation that involves an 8-bit exponent, and a 11-bit significand. Assume that the ordering of components in the representation is (from most significant to least significant) the sign bit, followed by the exponent, followed by the significand. Follow the convention of writing binary numbers with the most significant bit at the left end of the number, the least significant bit on the right end, etc.

- Give the binary representation of the number -78.25 assuming normalized representation, an implicit leading 1 for the significand, and no bias in the exponent. (4 points)
- Give the binary representation of the number -78.25 assuming normalized representation, an implicit leading 1 for the significand, and a bias of 127 in the exponent. (3 points)
- Why is the bias added to the exponent in floating point standards such as IEEE 754? (1 point)

## 2. Low-level programming (12 points)

Consider an assembler that supports the following instructions and directives. Next to each instruction is a pseudocode / register transfer description of what the instruction performs. Here, Ra and Rb can be any one of eight 16-bit general-purpose registers, R0–R7. The symbol B represents a 1-bit special-purpose register. The symbol “label” represents an arbitrary alphanumeric string that contains up to six characters. If R is a register, then [R] represents that value stored in R. Similarly, MEM[X] represents the value stored in memory at memory location X.

```

ADD Ra, Rb      ; [Ra] + [Rb] → Ra
SUB Ra, Rb      ; [Ra] - [Rb] → Ra
TST Ra          ; if ([Ra] < 0) 0 → B, else 1 → B
BRN label       ; if ([B] == 0) then branch to the instruction
                 ; associated with "label"
JMP label       ; Branch unconditionally to the instruction
                 ; associated with "label"
LOA Ra, Rb      ; This is a load instruction with register
                 ; indirect addressing: MEM[[Ra]] → Rb
LOA #I, Ra      ; This is a load instruction with immediate
                 ; addressing: I → Ra
STO Ra, Rb      ; This is a store instruction with register
                 ; indirect addressing: [Ra] → MEM[[Rb]]
NEG Ra          ; Negation: -[Ra] → Ra
label:          ; Associate "label" with the next instruction
                 ; in the program

```

- Using only the programming features shown above, write a program that multiplies the value stored in memory location 1000 to the value stored in memory location 2000, and stores the result in memory location 3000. You may assume that the numbers to be multiplied are non-negative. Furthermore, overflow handling is not required. (8 points)
- Let  $x$  denote the multiplicand (the value stored in memory location 1000), and let  $y$  denote the multiplier (the value stored in memory location 2000). Suppose that a load instruction takes 2 cycles to execute, a store instruction also takes 2 cycles to execute, and all other instructions take 1 cycle each to execute. Derive an expression for the total number of execution cycles required by your program in terms of  $x$  and  $y$ . (4 points)



