UCLA Computer Science 33 (Fall 2015)
Midterm 2
99 points, 99 minutes, open book, open notes.
Questions are equally weighted (11 min. each).
Use a separate sheet of paper for each answer.
Put a big problem number at each sheet's top.
Turn in your sheets in increasing numeric order.

Name:______________________ Student ID:___________

1	2	3	4	5	6	7	8	9	sum

1a (5 minutes). List in increasing numeric order the mathematical values of all the 'float' values whose stored fractions contain only 0 bits. If you cannot order them all numerically, state why and order them as best you can. You need not list each exact number if the overall pattern is obvious; just give the general mathematical form of the Nth number in the list.

1b (6 minutes). Likewise, except list 'float' values whose fractions contain only 1 bits.

2 (11 minutes). Define a C function with signature 'int divrte (int, int);' that takes two arguments X and Y, and returns X divided by Y, rounding the mathematically-correct result to the nearest int value, and with ties rounded to an even integer. For example, divrte(-3, 2) returns -2, and divrte(20, 8) returns 2. If Y is zero or if the result would overflow, your function can have any behavior you like. Use x86-64 floating-point arithmetic to implement your function with as few operations as you can.

3. Consider the following implementations of a dot-product function. Recall that the dot product of two vectors (A[1], ..., A[n]) and (B[1], ..., B[n]) is the sum of the values A[i]*B[i] for all i in 1, ..., n.

```
double
dota (double const *a, double const *b,
      long n)
{
  double r = 0;
  for (long i = 0; i < n; i++)
    r += a[i] * b[i];
  return r;
}

void
dotb (double const *a, double const *b,
      long n, double *result)
{
  *result = 0;
  for (long i = 0; i < n; i++)
    *result += a[i] * b[i];
}
```

3a (4 minutes). Give the machine-level calling conventions for dota and dotb.

3b (7 minutes). Which function is likely to be more efficient, and why can't the compiler optimize it to be nearly as efficient as the other one? Explain.

4. Suppose we have a machine with three levels of caches: L1, L2, and L3.

4a (6 minutes). Must all three levels use the same cache line size, or is it OK if the different levels use arbitrary cache line sizes, or do the levels' cache line sizes affect each other in some way? Briefly explain.

4b (5 minutes). Briefly explain why L1 caches are smaller and are typically direct-mapped, whereas L2 caches are typically larger and have multiple lines per set, and L3 caches are typically larger yet.

5 (11 minutes). Suppose we use gprof to profile two single-threaded implementations A and B of the same application. At the machine-language level implementation A has 1000 little functions, whereas implementation B has just 10 larger functions. The two implementations both take approximately 100 CPU seconds. For which implementation is gprof likely to produce more-useful information for optimizing the application? Briefly justify your answer.

6 (11 minutes). Suppose you use a simple reassociation transformation to rewrite your application's kernel, but discover that this does not improve overall performance on your particular x86-64 platform, even though you have compiler optimization enabled. Give a plausible reason this might occur.

7. The standard C function memcpy(DEST, SRC, N) copies N bytes from SRC to DEST and returns DEST. It has undefined behavior if SRC and DEST overlap. Suppose it is implemented as follows:

```
        memcpy:
                xorl    %ecx, %ecx
                testq   %rdx, %rdx
                movq    %rdi, %rax
                je      .L7
        .L5:
                movzbl  (%rsi,%rcx), %r8d
                movb    %r8b, (%rax,%rcx)
                addq    $1, %rcx
                cmpq    %rcx, %rdx
                jne     .L5
        .L7:
                rep ret
```

7a (5 minutes). Explain what this implementation does when SRC and DEST overlap.

7b (6 minutes). Explain any performance difference this implementation has when SRC and DEST overlap, as opposed to when they do not overlap.

8 (11 minutes). Consider the following x86-64 functions, which take signed integer arguments. Although the SETO instruction is not listed in the book, it is a standard SET instruction that sets the destination to OF; that is, SETO is to OF as SETS is to SF.

```
        f:
                leaq    (%rdi,%rsi), %rax
                cmpq    %rax, %rsi
                setg    %al
                shrq    $63, %rdi
                xorl    %edi, %eax
                ret

        g:
                addq    %rdi, %rsi
                seto    %al
                ret
```

8a (6 minutes). Suppose we call these functions with all arguments equal to their minimum possible values. Describe what each instruction does, in both functions.

8b (5 minutes). Compare and contrast the behavior and performance of these two functions in general.

9 (11 minutes). Why are local automatic variables typically not shared between threads? Briefly give a counterexample, showing how and why it might be useful to share a local automatic variable between two or more threads.