Computer Science 2021
Spring 2015
Quiz 1 (solutions)
March 2nd, 2015
Time Limit: 50 minutes, 3:35pm-4:25pm

- This exam contains 6 pages (including this cover page) and 4 questions. Once we tell you to start, please check that no pages are missing.
- Before starting the exam, you can fill out your name and other information of this page, but don't open the exam until you are directed to start. Don't put any of your answers on this page.
- You may use any textbooks, notes, or printouts you wish during the exam, but you may not use any electronic devices: no calculators, smart phones, laptops, etc.
- You may ask clarifying questions of the instructor or TAs, but no communication with other students is allowed during the exam.
- Please read all questions carefully before answering them. Remember that we can only grade what you write on the exam, so it's in your interest to show your work and explain your thinking.
- Students often find that the quiz questions vary in difficulty. Your best strategy is usually to skim over all the questions, and then start working on the ones that look easiest. We also suggest that you leave time at the end to attempt every question, since we can't give you any partial credit if you leave a question blank.
- By signing below you certify that you agree to follow the rules of the exam, not to share exam material with other students before their exams, and that the answers on this exam are your own work only.

The exam will end promptly at 4:25pm. Good luck!

Your name (print): _____

Your UMN email/X.500: _____@umn.edu

Sign and date: _____

Question	Points	Score
1	19	
2	32	
3	30	
4	19	
Total:	100	

1. (19 points) Floating point.

   Suppose that each of the decimal numbers or computations is converted to or performed using IEEE single-precision floating point ("`float`" in C). For each number write "E" if it can be represented exactly, or "A" if it can only be approximated inexactly. (1 point each)

   (a) __**E**__ 3.5

   (b) __**A**__ 36.2

   (c) __**A**__ $\sqrt{8}$

   (d) __**E**__ $\sqrt{49}$

   The following procedure takes a double-precision floating point number in IEEE format and prints out information about what category of number it is. (Unlike the rest of the exam, this is x86-64 code, because it is convenient to use a 64-bit `long` to hold the 52-bit significand of a double.) Fill in the missing code so that it performs this classification correctly. (3 points per blank)

```
void classify_double(double f)
{
   /* Unsigned value ul has the same bit pattern as f */
   unsigned long ul = *(unsigned long *) &f;
   /* Split u into the different parts */
   long sign = (ul >> 63) & 0x1;       // The sign bit

   long exp  = (ul >> 52) & 0x7FF;     // The exponent field

   long frac = ul & 0x3FFFFFFFFFFFF;  // The fraction field
   /* The remaining expressions should be written in terms of the values of sign, exp,
   and frac. The parts of the conditions in comments are optional. */

   if ( sign == 0 && exp == 0 && frac == 0 )
     printf("Positive zero\n");

   else if ( /* sign == 1 && */ exp == 0 && frac == 0 )
     printf("Negative zero\n");

   else if ( exp == 0 /* && frac != 0 */ )
     printf("Nonzero, denormalized\n");

   else if ( exp == 0x7FF )
     printf("NaN or +/- infinity\n");

   else if ( sign == 0 /* && exp != 0 && exp != 0x7FF */ )
     printf("Positive normalized\n");
   else /* sign == 1 && exp != 0 && exp != 0x7FF */
     printf("Negative normalized\n");
}
```

2. (32 points) Assembly language.

Consider the following assembly code for a function with a loop:

```
func1:
        pushl   %ebp
        movl    %esp, %ebp
        movl    8(%ebp), %eax
        testl   %eax, %eax
        je      .L5
        movl    $0, %edx
        testl   %eax, %eax
        js      .L3
.L4:
        addl    $1, %edx
        addl    %eax, %eax
        jns     .L4
.L3:
        movl    12(%ebp), %ecx
        movl    %edx, (%ecx)
        jmp     .L2
.L5:
        movl    $32, %eax
.L2:
        popl    %ebp
        ret
```

Based on the assembly code above, fill in the blanks below in its corresponding C source code. You may only use the source-level C variables `x`, `p`, and `n` in your expressions: don't use register names!

```
int func1(int x, int *p)
{
    int n = ______ 0 _______;

    if (_______ x == 0 _______)

        return _________ 32 __________;


    while (______ x >= 0 _____) {


          ______ x <<= 1 ________;              ________ n++ _____;

    }

    ________ *p = n __________;          return ________ x ________;
}
```

(If you're having trouble remembering what all those IA32 instructions do, we've included a short summary table on the last page of the quiz.)

3. (30 points) Integer representations.

   Suppose we have a very small microprocessor with a 5-bit word size. A pattern of bits in one of its registers might be interpreted either as a signed (two's complement) integer or as an unsigned integer. Each row of the following table shows how one bit pattern can be interpreted as a signed or unsigned number. Given the various kinds of descriptions in the left column, your job is to fill in all the remaining table entries (we've done "three" as an example, and filled in a few other obvious ones). Remember that UMin, UMax, TMin and TMax refer to the smallest (Min) and largest (Max) representable unsigned (U) and signed (T) values. Some values will be repeated.

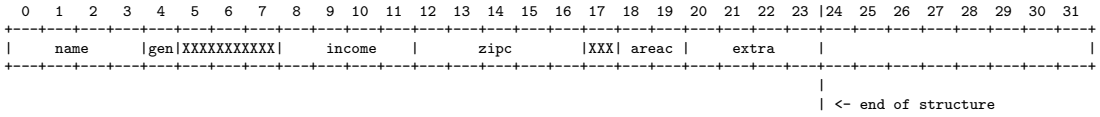
Description	Binary	Decimal (signed)	Decimal (unsigned)
zero	00000	0	0
three	00011	3	3
negative one	11111	-1	31
decimal 12	01100	12	12
unsigned 24	11000	-8	24
binary 01110	01110	14	14
binary 10101	10101	-11	21
UMin	00000	0	0
UMax	11111	-1	31
TMin	10000	-16	16
TMax	01111	15	15
TMax + 1	10000	-16	16
TMax + TMax	11110	-2	30
TMin + TMin	00000	0	0
-TMax	10001	-15	17
-TMin	10000	-16	16

4. (19 points) Structured data.

   Consider the following definition of a data structure to record information about people living in Minnesota:

```
struct info {
    char *name;
    char gend; /* 'M', 'F', etc. */
    float income;
    char zipc[5]; /* "55455", etc. */
    short areac; /* 612, etc. */
    union more_info { int bus_stop; char *rural_route; } extra;
};
```

   (a) Using the template below (which allows a maximum of 32 bytes, each labeled by its offset), indicate the memory layout of an structure of type `info`. Mark off and label the bytes that represent each element of the `struct`, treating the union `extra` as one element (so 6 in total). Cross hatch any areas that are allocated but not used (to satisfy alignment). Also separately indicate the right-hand boundary of the data structure with a labeled vertical line. (You may abbreviate names to any unique prefix.) (15 points)



   (b) How many bytes in total are allocated for an object of type `info`? (4 points)

   24 bytes.

For your reference, here are the size and alignment rules for common data types on Linux IA32, which we're assuming for this problem:

Type	Size (bytes)	Alignment (bytes)
`char`	1	1
`short`	2	2
`int`	4	4
pointer	4	4
`float`	4	4
`double`	8	4

IA32 assembly-language (AT&T format) quick reference table:

Instructions	
`add` X, Y	add $X + Y$ and store in Y, set flags based on result
`je` L	jump to L if ZF is set
`jmp` L	jump to L, always
`jne` L	jump to L if ZF is not set
`jns` L	jump to L if SF is not set
`js` L	jump to L if SF is set
`mov` X, Y	copy value X to location Y
`push` X	push X onto the stack
`pop` X	pop top element of stack and put in X
`shr` X	logical shift X right one position
`test` X, Y	Compute $X\&Y$, set flags based on result
Flags	
ZF	ZF is set if result is zero
SF	SF is set if result has sign bit set
Addressing modes	
(R)	mem[reg[R]]
D(R)	mem[D + reg[R]]
Size suffixes	
b	8-bit byte
w	16-bit value
l	32-bit value