Your Name (first last) _____ SID _____

UC Berkeley CS61C
Fall 2018 Midterm

← Name of person on left (or aisle) _____ TA name _____ Name of person on right (or aisle) → _____

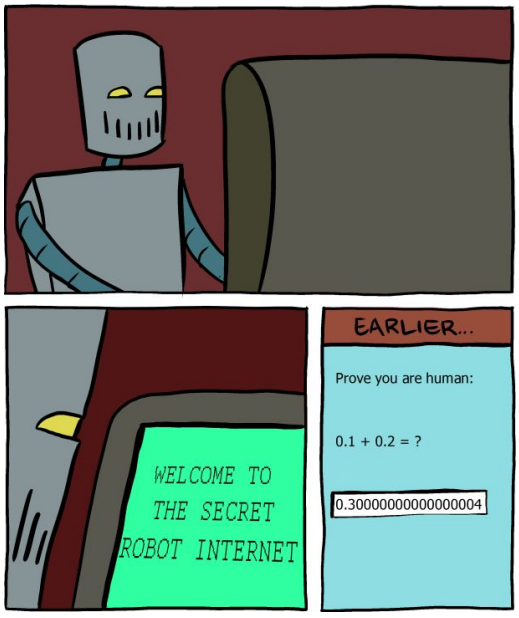
Fill in the correct circles & squares completely…like this: ● (select ONE), and ■(select ALL that apply)

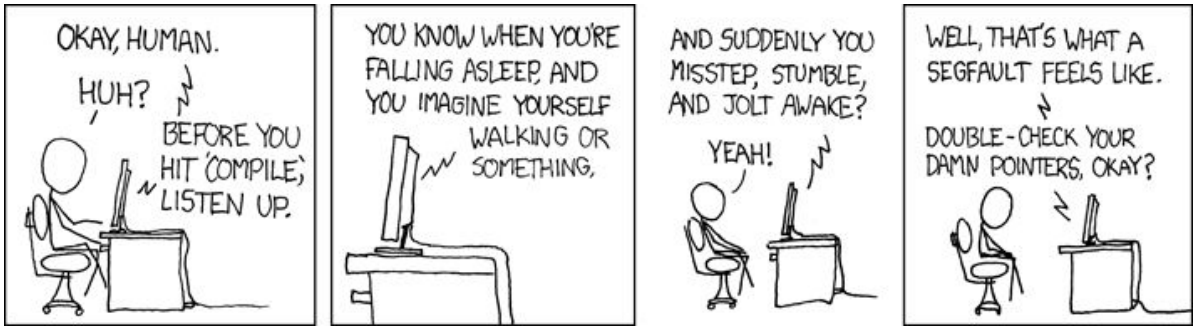
Quest-clobber questions: Q2, Q3a, Q4

Some people, when confronted with a problem, think *"I know, I'll use regular expressions."*
Now they have two problems.

You say *"I know, I'll use floating point!"*
Now you have 2.0001341678 problems.

Then you say *"I know, I'll solve it with threads!"*.
have Now problems. two you





Q1) *Float, float on…* (6 points; a,b 1pt; c,d 2pts)

Consider an 8-bit "minifloat" SEEEEEMM (1 sign bit, 5 exponent bits, 2 mantissa bits). All other properties of IEEE754 apply (bias, denormalized numbers, ∞, NaNs, etc). The bias is -15.

a) How many NaNs do we have? _____

b) What is the bit representation (in hex) of the next minifloat bigger than the minifloat represented by the hexadecimal value is **`0x3F`**? _____

c) What is the bit representation (in hex) of the encoding of -2.5? _____

d) What does **`should_be_a_billion()`** return? *(assume we always round down to 0)* _____

```
minifloat should_be_a_billion() {
      minifloat sum = 0.0;
      for (unsigned int i = 0; i < 1000000000; i++) { sum = sum + 1.0; }
      return sum;
}
```

Q2) *How can I bring you to the C of madness...* (4 points)

On the quest, you saw **`mystery`**, which should really have been called **`is_power_of_2`**, since it took in an unsigned integer and returned 1 when the input was a power of 2 and 0 when it was not. Well, it turns out we can write that in one line! What should the blanks be so that it works correctly? *(Hint: start at `iii` and `iv` and think about how the bit pattern of two related numbers is special if N is a power of 2)*

```
int is_power_of_2(unsigned int N) { return (N != 0) ___  (N ____ (N ___ ____)); }
/* ^ is bitwise xor, ~ is bitwise not */            i       ii     iii  iv
```

i				ii				iii	iv
◯ `|`	◯ `|!`	◯ `&`	◯ `&!`	◯ `|`	◯ `|!`	◯ `&`	◯ `&!`	◯ `-`	◯ `0`
◯ `||`	◯ `||!`	◯ `&&`	◯ `&&!`	◯ `||`	◯ `||!`	◯ `&&`	◯ `&&!`	◯ `+`	◯ `1`
◯ `^`	◯ `^!`	◯ `~`	◯ `~!`	◯ `^`	◯ `^!`	◯ `~`	◯ `~!`	◯ `*`	◯ `2`

Q3) *Cache money, dollar bills, y'all.* (18 points; a-c 2pts d-g 3pts)

We have a 32-bit machine, and a 4 KiB direct mapped cache with 256B blocks. We run the following code from scratch, with the cache initially cold, to add up the values of an uninitialized array to see what was there.

```
uint8_t addup() {
    uint8_t A[1024], sum = 0; // 8-bit unsigned
    touch(A);
    for (int i = 0; i < 1024; i++) { sum += A[i]; }
    return sum - 1;
}
```

```
void touch(uint8_t *A) {
    // Touch random location
    // in A between first and
    // last elements, inclusive
    A[random(0, 1023)] = 0;
}   // e.g.,random(0,2)⇒ 0,1,or 2
```

a) Assume **sum** has the smallest possible value after the loop. *What would **addup** return?* _____

b) Let **A = 0x100061C0**. *What cache index is **A[0]**?* _____

c) Let **A = 0x100061C0**. If the cache has a hit at **i=0** in the loop, what is the *maximum value **random** could have returned*? _____

For d and e, assume we don't know where **A** is, and we run the code from scratch again.

d) What's the *fewest number of cache misses* caused by the loop? _____

e) What's the *most number of cache misses* caused by the loop? _____

f) If we change to a fully associative LRU cache, how would c, d, e's values change? (select ONE per col)

c: ◯up ◯down ◯same	d: ◯up ◯down ◯same	e: ◯up ◯down ◯same

g) When evaluating your code's performance, you find an AMAT of 4 cycles. Your L1 cache hits in 2 cycles and it takes 100 cycles to go to main memory. What is the *L1 hit rate*? _____%

Q4) ***RISC-V business: I'm in a CS61C midterm & I'm being chased by Guido the killer pimp...*** **(14 points)**

a) Write a function in RISC-V code to return *0 if the input 32-bit float = ∞*, else a non-zero value. The input and output will be stored in **a0**, as usual.
(If you use 2 lines=3pts. 3 lines=2 pts)

```
isNotInfinity: _____________________________
               _____________________________
               _____  a0, ______, ______
               ret
```

(the rest of the question deals with the code on the right)
Consider the following RISC-V code run on a 32-bit machine:

```
done: li a0, 1
      ret
fun:  beq a0, x0, done
      addi sp, sp, -12
      addi a0, a0, -1
      sw ra, 8(sp)
      sw a0, 4(sp)
      sw s0, 0(sp)
      jal fun
      mv s0, a0
      lw a0, 4(sp)
      jal fun
      add a0, a0, s0
      lw s0, 0(sp)
      lw ra, 8(sp)
      addi sp, sp, 12
      ret
```

b) What is the hex value of the machine code for the underlined instruction labeled **fun**? (choose ONE)

◯0xFE050EEA ◯0xFE050EE3 ◯0xFE050CE3 ◯0xFE050FE3 ◯0xFE050EFA ◯0xFE050FEA

c) What is the one-line C disassembly of **fun** *with* recursion, and generates the same # of function calls:

```
uint32_t fun(uint32_t a0) { return ____________________________________________ }
```

d) What is the one-line C disassembly of **fun** that has *no* recursion (i.e., see if you can optimize it):

```
uint32_t fun(uint32_t a0) { return ____________________________________________ }
```

e) Show the call and the return value for the *largest possible value* returned by (d) above:

```
fun(__________) ⇒ _______________
```

Q5) ***What in the world is that funky smell? Oh, it's potpourri!*** **(18 points; a-e 2pts)**

a) What's the ideal speedup of a program that's 90% parallel run on an ∞-core machine? _____

b) How many times faster is the machine in (a) than a 9-core machine in the ideal case? _____

c) What was NOT something companies were doing (yet) to reap PUE benefits? (select ALL that apply)
- ▢ Do away with air conditioners
- ▢ Turn the servers completely off (not in an idle state) when not in use.
- ▢ Elevate cold aisle temperatures
- ▢ Have a UPS (Uninterruptible Power Supply) for the building in case of a power outage
- ▢ Pack the servers in freight containers to control air flow

d) What was NOT something Dave Patterson talked about in his Turing talk? (select ALL that apply)
- ▢ Dennard scaling is going strong
- ▢ Machine learning researchers are pushing the boundaries of architecture
- ▢ Some researchers have found that floating point has too much range, so they made their own floats
- ▢ VLIW (Very Long Instruction Word) architectures are an exciting new area of research
- ▢ Quantum computers are at least a decade off from solving the global thirst for computation

e) The value in memory pointed to by `x1` is 10. After two cores run the following code concurrently...

`lw x2,0(x1)`
`addi x2,x2,1`
`sw x2,0(x1)`
	`lw x3,0(x1)`
	`add x3,x3,x3`
	`sw x3,0(x1)`

...on shared memory, what are possible values in the shared memory location? (select ALL that apply)

▢10 ▢11 ▢12 ▢13 ▢14 ▢15 ▢16 ▢17 ▢18 ▢19 ▢20 ▢21 ▢22

Q5 continued) ***What in the world is that funky smell? Oh, it's potpourri!*** **(18 points; f 2pts; g,h 3pts)**

f) The final machine code bits for **beq** are known after: ◯compiler ◯assembler ◯linker ◯loader
This stage handles forward references: ◯compiler ◯assembler ◯linker ◯loader
This stage reads a dynamic library (DLL): ◯compiler ◯assembler ◯linker ◯loader

g) All 61C students were asked to record the time they spent learning C, and we wrote some Spark code to calculate the AVERAGE time every student spent learning C. (select ONE per box)

```
>>> CRDD = sc.parallelize([("Ana", 10), ("Sue", 50), ("Ana", 20)])

>>> def C_init(L):
        return (L[0], _____ii_____)

>>> def C_sum(A, B):
        return (A[0] + B[0], A[1] + B[1])

>>> def C_avg(L):
        return [ (L[0], L[1][0]/L[1][1]) ]

>>> CRDD._____i_____(C_init).reduceByKey(C_sum)._____iii_____(C_avg).collect()
[("Ana", 15), ("Sue", 50)]
```

i)◯**map** ◯**flatMap** ◯**reduce** ◯**reduceByKey**	**ii)**◯**1** ◯**L[1]** ◯**(L[1],1)** ◯**(L[0],L[1])**	**iii)** ◯**map** ◯**flatMap** ◯**reduce** ◯**reduceByKey**

h) Mark all **necessary** conditions to convert this code to SIMD for a ≈4x boost. (select ALL that apply)

```
void shift_vector( int *X, int n, int s ) {  for(int i=0; i < n; i++)  X[i] += s;   }
```

▢ Loop needs to be unrolled ▢ **i++** needs to become **i+=16** ▢ It needs to use 128-bit registers
▢ The CPU must have multiple cores ▢ There needs to exist a tail case, when **n** is not divisible by 4
▢ **x[i] += s** needs to change to **x[i] = x[i] + s** ▢ **i** needs to be declared as an **unsigned int**
▢ SIMD instructions need to be added