1. Multiple Choices

Q1. Why could automatic function call inlining sometimes be bad for performance?
(a) It introduces extra dynamic instructions, potentially adding to the critical path length.
(b) It introduces extra static instructions, putting more strain on the instruction cache.
(c) Fewer call instructions reduce the instruction-level parallelism.
(d) Trick question, it is never harmful for performance.

Q2. What of the following are reason(s) why multi-threaded code might go slower than expected?
(a) More contention for data in the cache.
(b) Overhead of synchronizing shared variables.
(c) Overhead of starting/stopping threads.
(d) Space overhead from .bss/.data sections.

Q4. When you print the address of a variable from C, what kind of address is that?
(a) Physical Address
(b) Virtual Address
(c) Depends on the context
(d) None of the above

Q5. For a floating point number, what would be an effect of allocating more bits to the exponent part by taking them from the fraction part?
(a) You could represent fewer numbers, but they could be much larger.
(b) You could represent the same numbers, but with more decimal places.
(c) You could represent both larger and smaller numbers, but with less precision.
(d) Some previously representable numbers would now round to infinity

Q6.

Which of the following is the best justification for using the middle bits of an address as the set index into a cache rather than the most significant bits?

(a) Indexing with the most significant bits would necessitate a smaller cache than is possible with middle-bit indexing, resulting in generally worse cache performance.
(b) It is impossible to design a system that uses the most significant bits of an address as the set index.
(c) The process of determining whether a cache access will result in a hit or a miss is faster using middle-bit indexing.
(d) A program with good spatial locality is likely to make more efficient use of the cache with middle-bit indexing than with high-bit indexing.

Answers: 1. b, 2. abc, 4. b, 5. c, 6. d,

2. Integer Puzzles

True or False? If false, give an explanation as to why.

```
int x = foo();
int y = bar();
unsigned ux = x
unsigned uy = y
```

if x < 0, then x * x > 0	
ux > -1	
If x >= 0, then -x <= 0	
If x <= 0, then -x >= 0	
x >> 3 == x/8	
if x & 7 = 7, then (x << 30) < 0	

False, overflow is possible.
False, ux = 0, 1
True
False, -TMIN == TMIN
False, negative numbers truncate towards 0
True, x_1 = 1, left shift 30, x_{31} = 1

3.

a.
```
int bitAnd(int x, int y) {
        return ~((~x) | (~y));
}
```

b.
```
int isPositive(int x) {
        return !(!(x)) & !(x >> 31);
}
```

c.
```
int byteSwap(int x, int n, int m) {
        int maskNth, maskMth;

        maskNth = ((x >> (n << 3)) & 0xFF);
        maskMth = ((x >> (m << 3)) & 0xFF);

        x ^= (maskNth << (n << 3));
        x ^= (maskMth << (m << 3));

        maskNth = maskNth << (m << 3);
        maskMth = maskMth << (n << 3);

        return (x | maskNth | maskMth);
}
```

Fill out the following for an access to virtual address: 0x27CB

VPN:	0x13E
PPO/VPO:	0x0B
TLB Index:	0x0
TLB Tag:	0x9F
TLB Hit?:	Y
Page Fault?:	N
PPN:	0x0DDE
Physical Address:	0x1BBCB
Cache Offset (CO/BO):	0x3
Cache Index:	0x0
Cache Tag:	0x3779
Cache Hit?:	N
Cache Data:	-

4.

Fill out the following for an access to virtual address: 0x3E78

VPN:	0x1F3
PPO/VPO:	0x18
TLB Index:	0x1
TLB Tag:	0xF9
TLB Hit?:	N
Page Fault?:	N
PPN:	0x0E6D
Physical Address:	0x1CDB8
Cache Offset (CO/BO):	0x0
Cache Index:	0x0
Cache Tag:	0x39B7
Cache Hit?:	Y
Cache Data:	0xFD

5. No, we cannot have deadlock.
First start by removing “a” from consideration, as this is not a resource the two threads are both using. Next look at what is left in Thread 1. If Thread 1 locks b, it will immediately unlock it, holding onto no other relevant resources. If c is available, it would then use c and immediately unlock it.
If thread 2 locks c immediately, we’ll get a similar situation where following this either thread 1 or thread 1 immediately lock and then unlock b. Thread 2 can then finish c, and thread 1 will be free to finish all.

6. Assembly

Solution: “1 7”

7. Stack overflow

a.

48 89 ee c3 … 56 padding bytes … 16 3b 4e 57 ff ff ff ff e4 23 b0 04 00 00 00 00

Before padding is instruction mv rbp to rsi followed by return, first pointer after padding is location of input aka 0xffffffff574e3b52 - 0x3C, and last pointer is call to Print_String_Exit

(Really any answer that is properly encoded and puts original rdi in rsi before returning to Print_String_Exit)

b.

60 padding byte … 18 a1 2e 05 00 00 00 00 02 e3 e2 e2 00 00 00 00 23 31 01 42 00 00 00 00 e4 23 b0 04 00 00 00 00

The functions contain gadgets that do the following:
good_function: mv rdi rsp
bad_function: mv rdx rsi
that_function: mv rsp rsi
some_function: mv rcx rdx
this_function: mv rdi rcx
f_function: mv rdi rsi

The string that does rdi->rcx->rdx->rsi through gadgets and then returns to Print_String_Exit is correct

Using rdi->rsp->rsi is not valid cause you need rsp to do the rop chaining

Trying rdi->rsi using f_function is not valid because there are non-nop bytes btw mv and retq

c.

Answers that mention implementing code that checks the length of input or canaries are acceptable answers since they prevent stack overflow from occurring. Random stack offsets is not an acceptable answer as the solution for b can still be used to hack the code.

8.

A.

- Multiplications: 2 * n
- Additions: n

B.

We can see that the performance-limiting computation here is the repeated computation of the expression xpwr = x * xpwr. This requires a floating point multiplication (5 clock cycles), and

the computation for one iteration cannot begin until the one for the previous iteration has completed. The updating of result only requires a floating-point addition (3 clock cycles) between successive iterations.

9. Linking

Answer:

1. (linker error)
2. (no error or warning, prints out 0 twice.
3. 2 3
4. (linker error)
5. (compiler warning)1 1

Notes:

1. static makes the symbol local to the source file (module, or compilation unit) where it is declared
2. extern denotes that the symbol should defined in another (i.e., external) source file, and we are currently referencing that same external symbol