University of Washington – Computer Science & Engineering

Autumn 2017 Instructor: Justin Hsia 2017-12-13

CSE351 FINAL

Last Name:	
First Name:	
Student ID Number:	
Name of person to your Left | Right	|
All work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CSE351 who haven't taken it yet. Violation of these terms could result in a failing grade. **(please sign)**	

Do not turn the page until 12:30.

Instructions

- This exam contains 14 pages, including this cover page. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The last page is a reference sheet. Please detach it from the rest of the exam.
- The exam is closed book (no laptops, tablets, wearable devices, or calculators). You are allowed two pages (US letter, double-sided) of *handwritten* notes.
- Please silence and put away all cell phones and other mobile or noise-making devices. Remove all hats, headphones, and watches.
- You have 110 minutes to complete this exam.

Advice

- Read questions carefully before starting. Skip questions that are taking a long time.
- Read *all* questions first and start where you feel the most confident.
- Relax. You are here to learn.

Question	M1	M2	M3	M4	M5	F6	F7	F8	F9	F10	Total
Possible Points	8	2	8	10	8	10	9	10	9	5	**79**

Question M1: Number Representation [8 pts]

(A) Take the 32-bit numeral **`0xC0800000`**. Circle the number representation below that has the *most negative* value for this numeral. [2 pt]

Floating Point   Sign & Magnitude   Two's Complement   Unsigned

(B) Let `float f` hold the value $\mathbf{2^{20}}$. What is the *largest power of 2* that gets rounded off when added to `f`? Answer in exponential form, not just the exponent. [2 pt]

Traffic lights display three basic colors: red (R), yellow (Y), and green (G), so we can use them to encode base 3! We decide to use the encoding 0↔R, 1↔Y, 2↔G. For example, $5 = 1\times3^1+2\times3^0$ would be encoded as **YG**. Assume each traffic light can only display one color at a time.

(C) What is the *unsigned* decimal value of the traffic lights displaying **RGYY**? [2 pt]

(D) If we have **9 bits** of binary data that we want to store, how many *traffic lights* would it take to store that same data? [2 pt]

Question M2: Design Question [2 pts]

(A) The machine code for x86-64 instructions are variable length. Name one advantage and one disadvantage of this design decision. [2 pt]

Advantage:
Disadvantage:

Question M3: Pointers & Memory [8 pts]

For this problem we are using a 64-bit x86-64 machine (**little endian**). Below is the `count_nz` function disassembly, *showing where the code is stored in memory.*

```
0000000000400536 <count_nz>:
  400536:   85 f6            testl  %esi,%esi
  400538:   7e 1b            jle    400555 <count_nz+0x1f>
  40053a:   53               pushq  %rbx
  40053b:   8b 1f            movl   (%rdi),%ebx
  40053d:   83 ee 01         subl   $0x1,%esi
  400540:   48 83 c7 04      addq   $0x4,%rdi
  400544:   e8 ed ff ff ff   callq  400536 <count_nz>
  400549:   85 db            testl  %ebx,%ebx
  40054b:   0f 95 c2         setne  %dl
  40054e:   0f b6 d2         movzbl %dl,%edx
  400551:   01 d0            addl   %edx,%eax
  400553:   eb 06            jmp    40055b <count_nz+0x25>
  400555:   b8 00 00 00 00   movl   $0x0,%eax
  40055a:   c3               retq
  40055b:   5b               popq   %rbx
  40055c:   c3               retq
```

(A) What are the values (in hex) stored in each register shown after the following x86 instructions are executed? Use the appropriate bit widths. Hint: what is the *value* stored in `%rsi`? [4 pt]

```
leal 2(%rdi, %rsi), %eax
movw (%rdi,%rsi,4), %bx
```

Register	Value (hex)
%rdi	0x 0000 0000 0040 0544
%rsi	0x FFFF FFFF FFFF FFFF
%eax	0x
%bx	0x

(B) Complete the C code below to fulfill the behaviors described in the inline comments using pointer arithmetic. Let **`char* charP = 0x400544`**. [4 pt]

```
char v1 = *(charP + _____);                       // set v1 = 0xDB

int* v2 = (int*)((____________*)charP - 2);       // set v2 = 0x400534
```

Question M4: Procedures & The Stack [10 pts]

The function `count_sp` counts the number of *spaces* in a `char` array (this is the recursive version of the `mystery` function from the Midterm). The function and its *disassembly* are shown below:

```
int count_sp(char* str) {
     if (*str)
          return (*str == ' ') + count_sp(str+1);
     return 0;
}
```

```
0000000000400536 <count_sp>:
  400536:  0f b6 07        movzbl (%rdi),%eax
  400539:  84 c0           testb  %al,%al
  40053b:  74 16           je     400553 <count_sp+0x1d>
  40053d:  53              pushq  %rbx
  40053e:  3c 20           cmpb   $0x20,%al
  400540:  0f 94 c3        sete   %bl
  400543:  0f b6 db        movzbl %bl,%ebx
  400546:  48 83 c7 01     addq   $0x1,%rdi
  40054a:  e8 e7 ff ff ff  callq  400536 <count_sp>
  40054f:  01 d8           addl   %ebx,%eax
  400551:  eb 06           jmp    400559 <count_sp+0x23>
  400553:  b8 00 00 00 00  movl   $0x0,%eax
  400558:  c3              retq
  400559:  5b              popq   %rbx
  40055a:  c3              retq
```

(A) The *right-most* column/portion of the disassembly is first generated as the output of which of the following? Circle one. [1 pt]

Compiler   Assembler   Linker   Loader

(B) The *left-most* column of the disassembly was generated by which of the following? [1 pt]

Compiler   Assembler   Linker   Loader

(C) Why is `%rbx` being pushed onto the stack? What is `%rbx` being used for in this function? [2 pt]

Why push:
Usage:

(D) What is the return address to `count_sp` that gets stored on the stack? Answer in hex. [1 pt]

0x

(E) Provide a call to `count_sp` that is *guaranteed* to cause a **segmentation fault**. [1 pt]

`count_sp( ____________ );`

(F) We call `count_sp(" ! ")`. Fill in the incomplete snapshot of the stack below (in hex) once this call to `count_sp` returns to `main`. For unknown words, write "garbage". [4 pt]

0x7fffffffdb68	<ret addr to main>
0x7fffffffdb60	<original rbx>
0x7fffffffdb58	
0x7fffffffdb50	
0x7fffffffdb48	
0x7fffffffdb40	
0x7fffffffdb38	
0x7fffffffdb30	
0x7fffffffdb28	
0x7fffffffdb20	

Question M5: C & Assembly [8 pts]

Answer the questions below about the following x86-64 assembly function, which *uses a struct*:

```
mystery:
.L3:    testq   %rdi, %rdi      # Line 1
        je      .L4             # Line 2
        cmpw    %si, 0(%rdi)    # Line 3
        je      .L5             # Line 4
        movq    8(%rdi), %rdi   # Line 5
        jmp     .L3             # Line 6
.L4:    movl    $0, %eax        # Line 7
        retq                    # Line 8
.L5:    movl    $1, %eax        # Line 9
        retq                    # Line 10
```

(A) What **C variable type** would `%rsi` be in the corresponding C program? [1 pt]

__________ `rsi`

(B) `%rdi` is a pointer to a struct that contains 2 fields. What is the width of the second field? [1 pt]

______ bytes

(C) Based on Line 5, give an intuitive name for the second field in the struct. [1 pt]

(D) Convert lines 1, 2, 7, and 8 into C code. Use variable names that correspond to the register names (e.g. `al` for the value in `%al`). [3 pt]

`if ( ____________ ) _______________;`

(E) Describe at a high level what you think this function *accomplishes* (not line-by-line). [2 pt]

Question F6: Caching [10 pts]

We have 64 KiB of RAM and a 2-KiB L1 data cache that is 4-way set associative with 32-byte blocks and random replacement, write-back, and write allocate policies.

(A) Calculate the TIO address breakdown: [1.5 pt]

Tag bits	Index bits	Offset bits

(B) How many management bits (bits *other* than the block data) are there in every line in the cache? [1 pt]

_______ bits

(C) The code snippet below accesses an array of `double`s. Assume `i` is stored in a register. Calculate the **Miss Rate** if the cache starts *cold*. [2.5 pt]

```
#define ARRAY_SIZE 256
double data[ARRAY_SIZE];       // &data = 0x1000 (physical addr)
for (i = 0; i < ARRAY_SIZE; i += 1)
    data[i] /= 100;
```

(D) For each of the proposed (independent) changes, write **IN** for "increased", **NC** for "no change", or **DE** for "decreased" to indicate the effect on the **Miss Rate** for the code above: [4 pt]

Use `float` instead ______

Half the cache size ______

Split the loop body into: ______
```
data[i] /= 10;
data[i] /= 10;
```

No-write allocate ______

(E) Assume it takes 100 ns to get a block of data from main memory. If our L1 data cache has a hit time of 2 ns and a miss rate of 3%, what is the average memory access time (AMAT)? [1 pt]

_______ ns

Question F7: Processes [9 pts]

(A) The following function prints out four numbers. In the following blanks, list three possible outcomes: [3 pt]

```
void concurrent(void) {
   int x = 3, status;
   if (fork()) {
      if (fork() == 0) {
         x += 2;
         printf("%d",x);
      } else {
         wait(&status);
         wait(&status);
         x -= 2;
      }
   }
   printf("%d",x);
   exit(0);
}
```

(1) ____________________

(2) ____________________

(3) ____________________

(B) For the following examples of exception causes, write "**N**" for intentional or "**U**" for unintentional from the perspective of the user process. [2 pt]

System call _______ Hardware failure _______

Segmentation fault _______ Mouse clicked _______

(C) Briefly define a **zombie** process. Name a process that can *reap* a zombie process. [2 pt]

Zombie process:
Reaping process:

(D) In the following blanks, write "**Y**" for yes or "**N**" for no if the following need to be updated when **execv** is run on a process. [2 pt]

Page table _______ PTBR _______ Stack _______ Code _______

Question F8: Virtual Memory [10 pts]

Our system has the following setup:

- 20-bit virtual addresses and 64 KiB of RAM with 256-B pages
- A 4-entry TLB that is fully associative with LRU replacement
- A PTE contains bits for valid (V), dirty (D), read (R), write (W), and execute (X)

(A) Compute the following values: [4 pt]

Page offset width ________   # of physical pages ________

# of virtual pages ________   TLBI width ________

(B) Briefly explain why we make physical memory **write-back** and **fully-associative**. [2 pt]

Write-back:
Fully-associative:

(C) The TLB is in the state shown when the following code is executed. The code eventually causes a **protection fault**. What are the values of the variables when the fault occurs? [4 pt]

```
long *p = 0x7F080;
for (int i = 0; 1; i++) {
   *p += 1;
    p += 4;
}
```

p = 0x____________

i = ______

TLBT	PPN	Valid	R	W	X
0x7F0	0xC3	1	1	1	0
0x7F2	0x3D	1	1	0	0
0x004	0xF4	1	1	0	1
0x7F1	0x42	1	1	1	0

Question F9: Memory Allocation [9 pts]

(A) In a free list, what is a **footer** used for? Be specific. Why did we not need to use one in allocated blocks in Lab 5? [2 pt]

Footer:
Lab 5:

(B) We are designing a dynamic memory allocator for a **64-bit computer** with **4-byte boundary tags** and **alignment size of 4 bytes**. Assume a footer is always used. Answer the following questions: [4 pt]

Maximum tags we can fit into the header (ignoring size): ________ tags

Minimum block size if we implement an *explicit* free list: ________ bytes

Maximum block size (leave as expression in powers of 2): ____________ bytes

(C) Consider the C code shown here. Assume that the `malloc` call succeeds and `foo` is stored in memory (not just in a register). Fill in the following blanks with ">" or "<" to compare the *values* returned by the following expressions just before `return 0`. [3 pt]

```
#include <stdlib.h>
int ZERO = 0;
char* str = "cse351";

int main(int argc, char *argv[]) {
    int *foo = malloc(8);
    free(foo);
    return 0;
}
```

`&foo` _____ `&ZERO`

`&str` _____ `ZERO`

`&main` _____ `str`

Question F10: C and Java [5 pts]

For this question, use the following Java object definition and C struct definition. Assume addresses are all 64-bits.

```
public class RentalJ {
   String addr;
   short  rooms;
   float  rent;
   int[]  zip;

   public void info() {
      System.out.println("Rental at "+addr);
   }
}
public class Apt extends RentalJ {
   int roommates;
   public int occupants() {
      return roommates+1;
   }
}
```

```
struct RentalC {
   char* addr;
   short rooms;
   float rent;
   int   zip[5];
};
```

(A) How much memory, in bytes, does an instance of `struct RentalC` use? How many of those bytes are *internal* fragmentation and *external* fragmentation? [3 pt]

sizeof(struct RentalC)	Internal	External

(B) How much *longer*, in bytes, are the following for `Apt` than for `RentalJ`? Assume the Java instance fields are aligned to 4 bytes. [2 pt]

Instance:	
Virtual method table (vtable):	

This page purposely left blank