

Andrew ID:  
Full Name:

*Hint: This is an old school handwritten exam. There is no authenticated login. If we can't read your AndrewID, we won't easily know who should get credit for this exam. If we can't read either your AndrewID or Full Name, we're in real bind. Please write neatly :-)*

## 18-213/18-613, Fall 2021 Final Exam

Friday, December 10th, 2021

### Instructions:

- Make sure that your exam is not missing any sheets (check page numbers at bottom)
- Write your Andrew ID and full name on this page (and we suggest on each and every page)
- This exam is closed book and closed notes (except for 2 double-sided note sheets).
- You may not use any electronic devices or anything other than what we provide, your notes sheets, and writing implements, such as pens and pencils.
- Write your answers in the space provided for the problem.
- If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 100 points.
- The point value of each problem is indicated.
- **Good luck!**

Problem #	Scope	Max Points	Score
1	Data Representation: "Simple" Scalars: Ints and Floats	10	
2	Data Representation: Arrays, Structs, Unions, and Alignment	10	
3	Assembly, Stack Discipline, Calling Convention, and x86-64 ISA	15	
4	Caching, Locality, Memory Hierarchy, Effective Access Time	15	
5	Malloc(), Free(), and User-Level Memory Allocation	10	
6	Virtual Memory, Paging, and the TLB	15	
7	Process Representation and Lifecycle + Signals and Files	10	
8	Concurrency Control: Maladies, Semaphores, Mutexes, BB, RW	15	
<b>TOTAL</b>	<b>Total points across all problems</b>	<b>100</b>	

### Question 1: Representation: “Simple” Scalars (10 points)

#### Part A: Integers (5 points, 1 point per blank)

Assume we are running code on two machines using two’s complement arithmetic for signed integers.

- Machine 1 has 5-bit integers
- Machine 2 has 7-bit integers.

Fill in the five empty boxes in the table below when possible and indicate “UNABLE” when impossible.

	<b>Machine 1: 5-bit w/2s complement signed</b>	<b>Machine 2: 7-bit w/2s complement signed</b>
Binary representation of 18 decimal		
Binary representation of -9 decimal		
Binary representation of +Tmax		
Binary representation of -1 decimal		

**Part B: Floats (5 points, 1/2 point per blank)**

For this problem, please consider a floating point number representation based upon an IEEE-like floating point format as described below.

- Format A:
  - There are 5 bits
  - There is 1 sign bit s.
  - There are  $k = 2$  exponent bits.
  - You need to determine the number of fraction bits.
- Format B:
  - There are 7 bits
  - There is 1 sign bit s.
  - There are  $n = 3$  fraction bits.

Fill in the empty (non grayed-out) boxes as instructed.

	Format A	Format B
<b>Total Number of Bits (Decimal)</b>	<b>5</b>	<b>7</b>
<b>Number of Sign Bits (Decimal)</b>	<b>1</b>	<b>1</b>
<b>Number of Fraction Bits (Decimal)</b>		<b>3</b>
<b>Number of Exponent Bits (Decimal)</b>	<b>2</b>	
<b>Bias (Decimal)</b>		
<b>+Infinity (Binary bit pattern)</b>		
<b>100010 (Decimal value, unrounded)</b>		
<b>11011 (Decimal value, unrounded)</b>		
<b>0111001 Meaning of the bit pattern</b>		
<b>// x,y, and z are floats <math>(x+1) - ((x+2) - 1) == 0</math></b>	Circle one:  Always equal Always unequal It depends	

## Question 2: Representation: Arrays, Structs, Unions, Alignment, etc. (10 points)

### Part A: Arrays (5 points)

Consider the following definitions in an x86-64 system with 8-byte pointers and 4-byte ints:

Definition A	Definition B
<pre>int numbersA[3][2] =   {2, 4, 6, 8, 10, 12};</pre>	<pre>int **numbersB =   malloc (3*sizeof(int *));  // ... // You'll complete this code in 2(A) (3) below</pre>

**2(A)(1) (1 point):** How many bytes are allocated to `numbersA`? (Write "UNKNOWN" if not knowable).

*Hint:* Think `sizeof()`

**2(A)(2) (1 point):** How many bytes are allocated to `numbersB`? (Write "UNKNOWN" if not knowable).

*Hint:* Think `sizeof()`

**2(A)(3) (3 points)** Complete the given C language code for `numbersB` such that it fully allocates the array and initializes it such that corresponding elements of `numbersB` and `numbersA` have the same values:

Continued on next page.

## Question 2: Representation: Arrays, Structs, Unions, Alignment, etc. (10 points)

### Part B: Structs and Alignment (5 points)

For this question please assume “Natural alignment”, in other words, please assume that each type must be aligned to a multiple of its data type size.

Please consider the following struct:

```
struct {
    char c;        // 1-byte type
    short s1;     // 2-byte type
    double d;     // 8-byte type
    short s2;
} partB;
```

**2(B)(1) (1 point):** What would you expect to be the value of the expression below?

```
sizeof(struct partB)
```

**2(B)(2) (1 point):** Why should a programmer always use the sizeof() operator in code versus computing the value themselves? Give two (2) reasons.

**2(B)(1) (2 points):** Rewrite the struct above to minimize its size after alignment-mandated padding:

```
struct {
```

```
} partB;
```

### Question 3: Assembly, Stack Discipline, Calling Convention, and x86-64 ISA

#### Part A: Loops and Calling Convention (7 points)

Consider the following code:

```
function:
.LFB0:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $32, %rsp
    movl    %edi, -20(%rbp)
    movl    %esi, -24(%rbp)
    movl    -20(%rbp), %eax
    movl    %eax, -4(%rbp)
    jmp     .L2

.L5:
    movl    $0, -8(%rbp)
    jmp     .L3

.L4:
    addl    $1, -8(%rbp)

.L3:
    movl    -8(%rbp), %eax
    cmpl   -4(%rbp), %eax
    jl     .L4
    movl    $88, %edi           # 88 is ASCII for 'X'
    call   putchar
    movl    $10, %edi          # 10 is ASCII for '\n'
    call   putchar
    addl   $1, -4(%rbp)

.L2:
    movl    -4(%rbp), %eax
    cmpl   -24(%rbp), %eax
    jl     .L5
    nop
    leave
    ret
```

**3(A)(1) (2 points):** How many loops does this function have? How do you know?

**3(A)(2) (1 points):** How many arguments does this function receive (and use)?

**Continued on next page.**

**3(A)(3) (2 points):** For each argument you listed, please indicate either (a) which **specific** register was used to pass it in, or (b) that it was sourced from the stack (**you don't need to give the address**). Please leave any extra blanks empty (*Hint: You won't need all of them*).

Argument	Specific register or "Stack"
1st	
2nd	
3rd	
4th	
5th	

Consider the following function activation. Consistent with your answer to the question above, it includes more arguments that the function actually requires. Please ignore any extra arguments.

```
function(10, 9, 8, 7, 6);
```

**3(A)(2) (2 points):** How many times does the inner-most loop run?

Hint: If the inner-most loop is nested, you may need to consider the loops in which it is nested.

**Continued on next page.**

## Part B: Conditionals (8 points)

Consider the following code:

```
(gdb) disassemble function
Dump of assembler code for function function:
0x000000000400533 <+0>:      cmp     %esi,%edi
0x000000000400535 <+2>:      jg     0x400561 <function+46>
0x000000000400537 <+4>:      cmp     $0x5,%edi
0x00000000040053a <+7>:      ja     0x400557 <function+36>
0x00000000040053c <+9>:      mov     %edi,%eax
0x00000000040053e <+11>:     jmpq   *0x400630(,%rax,8)
0x000000000400545 <+18>:     mov     $0x1,%edi
0x00000000040054a <+23>:     mov     %edi,%eax
0x00000000040054c <+25>:     imul   %edi,%eax
0x00000000040054f <+28>:     add    %edi,%eax
0x000000000400551 <+30>:     retq
0x000000000400552 <+31>:     mov     $0xffffffffec,%edi
0x000000000400557 <+36>:     mov     %edi,%eax
0x000000000400559 <+38>:     shr    $0x1f,%eax
0x00000000040055c <+41>:     add    %edi,%eax
0x00000000040055e <+43>:     sar    %eax
0x000000000400560 <+45>:     retq
0x000000000400561 <+46>:     mov     $0xffffffff,%eax
0x000000000400566 <+51>:     retq
0x000000000400567 <+52>:     mov     $0x8,%eax
0x00000000040056c <+57>:     retq
End of assembler dump.
```

Consider also the following memory dump:

```
(gdb) x/10gx 0x400620
0x400620:      0x0000000000002001      0x0000000000000000
0x400630:      0x00000000000400545    0x0000000000040054a
0x400640:      0x00000000000400557    0x0000000000040054a
0x400650:      0x00000000000400567    0x00000000000400552
0x400660:      0x000000443b031b01     0xffffffffa00000007
```

Continued on next page



**(3)(B)(1) (1 points):** How many “if statements” are likely present in the C Language code from which this assembly was compiled? At what address of the assembly code shown above does each occur?

This code was compiled from C Language code containing a switch statement. **Please do not include any “if statement” present in the assembly that is likely part of the switch statement** in the original C code, i.e. do not count any “if statement” that is used to manage one or more “cases” of a “switch statement”.

**(3)(B)(2) (2 points):** What integer input values are managed by non-default cases of the switch statement? How do you know?

**(3)(B)(3) (1 point):** Is there a default case? If so, at what address does it begin? How do you know?

**(3)(B)(4) (2 points):** Which case(s), if any, share exactly the same code? How do you know?

**(3)(B)(5) (2 points):** Which case(s), if any, fall through to the next case after executing some of their own code? How do you know?

**Question 4: Caching, Locality, Memory Hierarchy, Effective Access Time (15 points)**

**Part A: Caching (8 points)**

Given a model described as follows:

- Associativity: 2-way set associative
- Total size: 512 bytes (not counting meta data)
- Block size: 32 bytes/block
- Replacement policy: Set-wise LRU
- 12-bit addresses

**4(A)(1) (1 point)** How many bits for the block offset?

**4(A)(2) (1 point)** How many bits for the set index?

**4(A)(3) (1 point)** How many bits for the tag?

**4(A)(4) (5 points, ½ point each):** For each of the following addresses, please indicate if it hits, or misses, and if it misses, if it suffers from a capacity miss, a conflict miss, or a cold miss:

Address	Circle one (per row):		Circle one (per row):			
0xA10	Hit	Miss	Capacity	Cold	Conflict	N/A
0X804	Hit	Miss	Capacity	Cold	Conflict	N/A
0X898	Hit	Miss	Capacity	Cold	Conflict	N/A
0XFDF	Hit	Miss	Capacity	Cold	Conflict	N/A
0XA00	Hit	Miss	Capacity	Cold	Conflict	N/A
0X806	Hit	Miss	Capacity	Cold	Conflict	N/A
0XCD5	Hit	Miss	Capacity	Cold	Conflict	N/A
0XA10	Hit	Miss	Capacity	Cold	Conflict	N/A
0X3DD	Hit	Miss	Capacity	Cold	Conflict	N/A
0XFC7	Hit	Miss	Capacity	Cold	Conflict	N/A

**Continued on next page.**

**Part B: Locality (4 points)**

**4(B)(1) (2 points):** Consider the following code:

```
int array[SIZE1][SIZE2];
int sum=0;
for (int outer=0; outer<SIZE1; outer+=STEP)
    for (int inner=0; inner<(SIZE2-1); inner++)
        sum += array[outer][inner] + 2*array[outer][inner+1];
```

Considering only access to “array”, as “step” increases (significantly), please mark how each type of locality would be impacted. Please also explain why in the space provided.

Spatial	Decrease	Increase	Unaffected
Temporal	Decrease	Increase	Unaffected

**4(B)(2) (2 points):** Consider the following code:

```
int array[ROWS][COLS];
int sum=0;
for (int row=0; index<ROWS; row++)
    for (int col=0; col<(COLS-1); col++)
        sum += array[row][col]+ array[row][col+1];
```

Imagine an array extremely large in all dimensions, an int size of 4 bytes, and a cache block size of 16 bytes. To the nearest whole percent or simple fraction, what would you expect the miss rate for accesses to “array” to be? Why?

**Continued on next page.**

**Part C: Memory Hierarchy and Effective Access Time (3 points)**

Imagine a system with a DRAM-based main memory layered beneath an super-fast cache.

- The DRAM has a 100nS access time.
- The effective access time is 15nS.
- The miss rate is 10%.
- In the event of a miss, memory access time and cache access time do not overlap: They occur 100% sequentially, one after the other.

**FOR SIMPLICITY, AVOID COMPLEX CALCULATION AND LEAVE YOUR ANSWER AS A SIMPLE FRACTION**

What is the super-fast cache access time?

CACHE\_ACCESS\_TIME=

### Question 5: Malloc(), Free(), and User-Level Memory Allocation (10 points)

Consider the following code series of malloc's and free's:

```
ptr1 = malloc(4);
ptr2 = malloc(8);
ptr3 = malloc(4);
free(ptr1);
free(ptr3);
ptr4 = malloc(8);
ptr5 = malloc(12);
free(ptr4);
free(ptr2);
ptr6 = malloc(32);
```

And a malloc implementation as below:

- Explicit list
- First-fit (search starts at head each time)
- Headers of size 8 bytes
- Footer size of 8-bytes
- Every block is always constrained to have a size a multiple of 8 (In order to keep payloads aligned to 8 bytes).
- A first-fit allocation policy is used.
- If no unallocated block of a large enough size to service the request is found, sbrk is called for the smallest multiple of 8 that can service the request.
- The heap is unallocated until it grows in response to the first malloc.
- Constant-time coalescing is employed.

NOTE: You do NOT need to simplify any mathematical expressions. Your final answer may include multiplications, additions, and divisions.

**4(A) (2 points)** After the given code sample is run, how many total bytes have been requested via sbrk? In other words, how many bytes are allocated to the heap? Draw a figure showing the heap and where each ptr is located.

Continued on next page.

**4(B) (2 points)** How many of those bytes are used for currently allocated blocks (vs currently free blocks), including internal fragmentation and header information?

**4(C)(2 points)** How much internal fragmentation is there due to padding (Answer in bytes)? (*Hint: Free blocks have no internal fragmentation*).

**4(D)(2 points)** How much internal fragmentation is there due to headers and footers (Answer in bytes)? (*Hint: Free blocks have no internal fragmentation*).

**4(E)(2 points)** Imagine that the user wrote a 14-character string to the buffer allocated ptr5. What would be the most likely result? And why? Circle the most likely result and then explain below.

- A. It would be correct
- B. It would be incorrect code, but would likely work correctly in this environment
- C. In this environment, it will likely compile and run, but die of a SEGV or similar runtime memory error
- D. In this environment, it will likely compile and run, but could generate incorrect results or crash later on

**Your explanation:**

**6. Virtual Memory, Paging, and the TLB (15 points)**

This problem concerns the way virtual addresses are translated into physical addresses. Imagine a system has the following parameters:

- Virtual addresses are 16 bits wide.
- Physical addresses are 16 bits wide.
- The page size is 256 bytes.
- The TLB is 2-way set associative with 16 total entries.
- A single level page table is used

**Part A: Interpreting addresses**

**6(A)(1) (1 points):** Please label the diagram below showing which bit positions are interpreted as each of the PPO and PPN. Leave any unused entries blank.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPN/ PPO																

**6(A)(2) (1 points):** Please label the diagram below showing which bit positions are interpreted as each of the VPO and VPN (top line) and each of the TLBI and TLBT (bottom line). Leave any unused entries blank.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VPO/ VPN																
TLBI/ TLBT																

**6(A)(3) (1 points):** How many entries exist within each page table? *Hint:* This is the same as the total number of pages within each virtual address space.

Continued on net page.

**Part B: Hits and Misses (12 points)**

Shown below are a **partial** TLB and **partial** page table.

**TLB:**

Index	Tag	PPN	Valid	Scratch space for you
0	0x0A	0x42	1	
0	0x1B	0x23	1	
1	0x14	0X12	1	
1	0x11	0X45	0	
2	0x05	0xA3	0	
2	0x0A	0X78	1	
3	0x09	0X56	1	
3	0x02	0X24	0	
4	0x08	0x25	0	
4	0x10	0x26	1	

**Page Table:**

Index/VPN	PPN	Valid	Scratch space for you
3	0X56	1	
23	0xA3	1	
96	0X42	1	
140	0X12	0	

For each address shown below, please indicate if it is a TLB Hit or Miss, whether or not it is a page fault, or if either can't be determined from the information provided.

Additionally, if knowable from the information provided, please provide the valid PPN

Virtual Address	TLB Hit or Miss?	Page Fault? Yes or No	PPN If Knowable
0x0344	Hit Miss Not knowable	Yes No Not knowable	
0x1744	Hit Miss Not knowable	Yes No Not knowable	
0x8022	Hit Miss Not knowable	Yes No Not knowable	
0x8C42	Hit Miss Not knowable	Yes No Not knowable	



## Question 7: Process Representation and Lifecycle + Signals and Files (10 points)

### Part A (3 points):

Please consider the following code:

```
void main(){
    fork()
    printf ("A"); fflush(stdout);
    if (!fork()) {
        printf ("C"); fflush(stdout);
    } else {
        wait(NULL);
        printf ("E"); fflush(stdout);
    }
    printf ("F"); fflush(stdout);
}
```

**7(A)(1) (1 points):** Give one possible output string

**7(A)(2) (1 points):** Give one output string that has the correct output characters (and number of each character), but in an impossible order.

**7(A)(3) (1 points):** Why can't the output you provided in 7(A)(2) be produced? Specifically, what constraint(s) from the code does it violate?

**Continued on next page.**

**Part B (3 points):**

Please consider the following code:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char* argv[]){
    char buffer[4] = "abc";

    // Assume "file.txt" exists but is initially empty

    int fd0 = open("file.txt", O_RDWR);
    int fd1 = 0;
    int fd2 = open("file.txt", O_RDWR);

    read(fd0, buffer, 1);
    dup2(fd0, fd1);

    read(fd2, buffer+1, 2);
    write(fd0, buffer, 3);

    read(fd2, buffer, 1);
    write(fd1, buffer, 1);

    return 0;
}
```

**7(B)(1) (1 points):** What is the content of the output file after this code completes?

**7(B)(2) (1 points):** How many entries are there in the system-wide open file table related to this code?

**Continued on next page.**

**7(B)(3) (1 points):** For each listed file descriptor variable, identify the file descriptor table entry pointed to by each file descriptor variable. Name the file descriptor entries FT1, FT2, FT3, FT4, etc.

File descriptor variable	File table entry, e.g. FT1, FT2, FT3, FT4
fd0	
fd1	
fd2	

**Continued on next page**

### Part C (4 points):

Please consider the following code:

```
#include <stdio.h>
#include <wait.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>

int count = 0;

void inthandler(int sig){
    count = 0;

    printf("SIGINT received\n");
    return;
}

void childhandler(int sig) {
    int status;

    wait(&status);
    count += WEXITSTATUS(status);

    return;
}

void main() {
    pid_t pid; // pid of child process

    signal(SIGINT, inthandler);
    signal(SIGCHLD, childhandler);

    pid = fork();
    if(!pid){
        kill(getppid(), SIGINT);
        exit(5); // Exit status is 5
    }

    sleep(5);

    printf("count = %d\n", count);
    exit(0); // Exit status is 0
}
```

**Continued on next page.**

**7(C)(1) (2 points):** What are the possible output(s) of the program?

**7(C)(2) (1 points):** There a critical (problematic shared) resource (variable)? What is it?

**7(C)(3) (1 points):** What is the critical resource shared between or among?

## Question #8: Concurrency Control: Maladies, Semaphores, Mutexes, BB, RW (15 points)

Consider the goal of writing a concurrent program to achieve the following

- The main thread creates 100 threads, waits for each thread to terminate
- One the main program terminates, it prints the value of global variable *sum*.
- Each thread increments the value of global variable *cnt* by 1, and adds the value of *cnt* to *sum*.
- Both *sum* and *cnt* have the initial value of 0.

Each of the following 5 programs represent an attempt at a correct solution, but some suffer from concurrency-related problem(s).

- Please write CORRECT for each correct solution.
- Please write INCORRECT and describe the CONCURRENCY problem(s) for each incorrect attempt at a solution.

All the programs have the following global variable definitions:

```
volatile int sum = 0;
volatile int cnt = 0;
sem_t mutex1, mutex2;
```

### 8(A)(1) Attempt #1 (3 points)

```
void *foo_1(void *vargp) {
    cnt += 1;
    sum += cnt;
}
int main() {
    pthread_t threads[ 100 ];
    int i;
    for (i = 0; i < 100; i++)
        pthread_create(&threads[ i ], NULL, foo_1, NULL);
    for (i = 0; i < 100; i++)
        pthread_join(threads[ i ], NULL);
    printf("%d", sum);
}
```

**Write your response below:**

**Continued on next page.**

**8(A)(2) Attempt #2 (3 points)**

```
void foo_2(void *vargp) {
    sem_wait(&mutex1);
    cnt += 1;
    sum += cnt;
    sem_post(&mutex1);
}

int main() {
    sem_init(&mutex1, 0, 1);
    pthread_t threads[ 100 ];
    int i;
    for (i = 0; i < 100; i++)
        pthread_create(&threads[ i ], NULL, foo_2, NULL);
    for (i = 0; i < 100; i++)
        pthread_join(threads[ i ], NULL);
    printf("%d", sum);
}
```

**Write your response below:**

**Continued on next page.**

**8(A)(2) Attempt #3 (3 points)**

```
void foo_3(void *vargp) {
    sem_wait(&mutex1);
    cnt += 1;
    sem_post(&mutex1);
    sem_wait(&mutex2);
    sum += cnt;
    sem_post(&mutex2);
}
int main() {
    sem_init(&mutex1, 0, 1);
    sem_init(&mutex2, 0, 1);
    pthread_t threads[ 100 ];
    int i;
    for (i = 0; i < 100; i++)
        pthread_create(&threads[ i ], NULL, foo_3, NULL);
    for (i = 0; i < 100; i++)
        pthread_join(threads[ i ], NULL);
    printf("%d", sum);
}
```

**Write your response below:**

**Continued on next page.**



**8(A)(2) Attempt #4 (3 points)**

```
void foo_4(void *vargp) {
    sem_wait(&mutex1);
    cnt += 1;
    sem_wait(&mutex2);
    sem_post(&mutex1);
    sum += cnt;
    sem_post(&mutex2);
}
int main() {
    sem_init(&mutex1, 0, 1);
    sem_init(&mutex2, 0, 1);
    pthread_t threads[ 100 ];
    int i;
    for (i = 0; i < 100; i++)
        pthread_create(&threads[ i ], NULL, foo_4, NULL);
    for (i = 0; i < 100; i++)
        pthread_join(threads[ i ], NULL);
    printf("%d", sum);
}
```

**Write your response below:**

**Continued on next page.**

**8(A)(2) Attempt #5 (3 points)**

```
void foo_5(void *vargp) {
    cnt += 1;
    sum += cnt;
    sem_post(&mutex1);
}
int main() {
    sem_init(&mutex1, 0, 1);
    pthread_t threads[ 100 ];
    int i;
    for (i = 0; i < 100; i++) {
        sem_wait(&mutex1);
        pthread_create(&threads[ i ], NULL, foo_5, NULL);
    }
    for (i = 0; i < 100; i++)
        pthread_join(threads[ i ], NULL);
    printf("%d", sum);
}
```

**Write your response below:**