Midterm Exam

18-213/613 Midterm Exam (Fall 2021)

Important notes:

- This exam contains 6 questions.
- You are not required to answer all of them. Please choose to answer questions within the constraints described below.
- There is no extra credit for answering additional questions.
- Should additional questions be answered, we will count the LOWER of the options. It is to your advantage to make choices.
- This exam is an individual effort.
- You are not permitted to help others, in any way, with this exam.
- You are not permitted to release or to discuss this exam with anyone, except the course staff, until given permission to do so by the instructors (which will not occur until all students have completed the exam. There may be exceptional cases that take it late).
- You are permitted to use only the official course textbook, the official course slides, and your own personal notes.
- A simple calculator is permitted, but won't prove to be helpful (we don't think).
- **You have 90 minutes, from first exposure through submission to take this exam. Do not attempt to "peek", "check", or "test" the exam. This will start your clock.**

Answer EXACTLY ONE of these:

- Question 1: Integers
- Question 2: Floats
    - Properties
    - Special Values

Answer EXACTLY ONE of these:

- Question 3: Assembly
    - Basic control
    - Switch
- Question 4: Calling Convention, Stack Discipline

Answer ***BOTH* of these:**

- Question 5: Data
    - Structs
    - Arrays
- Question 6: Caching and Memory Access

- Fully Associative Trace
- 2-Way Set Associative Trace
- Comparative Performance
- Memory Access Time

Question 1: Integers

This question is based upon the following declaration on a **machine using 5-bit two's complement arithmetic for signed integers.**

```
int x = -13;
unsigned uy = x;
```

Fill in the empty boxes in the table below.

- **Show all digits for the "Binary" column, including any leading 0s.**
- You need not fill in entries marked with "–".
- TMax denotes the largest positive two's complement number
- TMin denotes the smallest negative two's complement number.
- Hint: Be careful with the promotion rules that C uses for signed and unsigned ints, i.e. how the C Language handles implicit casts between the types.

Expression	Decimal Representation	Binary Representatior
–	*−3*	(A)
–	(B)	*0(*
x	-	(C)
uy	(D)	-
x − uy	-	(E)
TMax + 1	(F)	-
TMin - 1	-	(G)
TMin + 1	(H)	-
TMin + TMin	-	(I)

1 2.5 points

Blank (A):

2 2.5 points

Blank (B):

3 2.5 points

Blank (C):

4 2.5 points

Blank (D):

5 2.5 points

Blank (E):

6 2.5 points

Blank (F):

TMax + TMin	(J)	-

7 2.5 points

Blank (G):

8 2.5 points

Blank (H):

9 2.5 points

Blank (I):

10 2.5 points

Blank (J):

Question 2: Floats

Part 1: Properties

- Consider the following 7-bit floating point representation based on the IEEE floating point format:
- The most significant bit is the sign bit
- The next *k=3* bits are the exponent.
- The last *n=3* bits are the significand.
- The bias is to balance the exponents in a way consistent with IEEE single and double precision floating point numbers, i.e. according to the formula and with the intuition we discussed in class.

Please answer the questions to the right.

11 2 points

2.1(A): What is the bias? (Decimal number)

12 2 points

2.1(B): What is the actual exponent, e.g. what we called “E” in class, for denormalized numbers? (Give answer in decimal).

Part 2: Special values

This question is based upon the same number format as Part I.

Fill in the blank entries in the following table. Include nothing but 0s and 1s. Include no spaces.

Description	Sign	Binary Encoding
Zero	+	0000000
Smallest Positive (nonzero)	+	(A)
Largest denormalized	-	(B)
Smallest positive normalized	+	(C)

13 2 points

2.1(C): Consider any two adjacent denormalized floating point numbers.

What is the absolute value of their difference **in base-2 binary**? Fill in the blank, **without** any unnecessary trailing 0s.:

0. ______

14 2 points

2.1(D): Consider any two adjacent normalized numbers with **a biased exponent field of** exp=010.

Determine the absolute value of the difference **in their base-2 binary values and write it out in binary as x.y without** any unnecessary trailing 0s and **without** any unnecessary leading 0s (include a single leading or trailing zero per field, as necessary, to avoid leaving either field entirely blank.):

________(x)________ . ________(y)________

What is (x)?

______

15 2 points

2.1(E): Consider the scenario in question (D) above.

________(x)________ . ________(y)________

What is (y)?

______

16 2 points

2.1(F) Consider any two adjacent normalized numbers with **a biased exponent field of** exp=011.

Determine the absolute value of their difference **in base-2 binary and write it out as x.y without** any unnecessary trailing 0s and **without** any unnecessary leading 0s (include a single 0 per field as necessary to avoid leaving either field blank):

________(x)________ . _______(y)________

17 1 point

2.1(E): Consider the scenario in question (D) above, what is y?

_________(x)__________ .

_________(y)__________

What is (y)?

18 2 points

2.1(H) Which of the following explains the difference between your answers to (d), (e), and (f). Check **all** that apply:

- [ ] When the points on the number line are assigned to be closer in value, rounding error is reduced
- [ ] When the points on the number line are assigned to be farther apart in value, a number line can cover a larger range
- [ ] IEEE wanted the number line to span a large range but to keep the rounding error approximately proportional to the magnitude of the number
- [ ] Denormalized numbers are relatively very small in magnitude and represent only a very small portion of the range, so it makes sense for them to be equidistant.

19 3 points

Part 2, Blank (A):

20 3 points

Part 2, Blank (C):

21 3 points

Part 2, Blank (B):

Question 3: Control and Switch

22 3 points

Part 1: Control

Please consider the following assembly code and then answer the questions about it that follow:

Hint: We strongly suggest that, before answering the questions, you translate the code below into the C Language and simplify it in writing.

```
.LC0:
        .string "count: %d\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        pushq   %rbp
        movq    %rsp, %rbp
        pushq   %r13
        pushq   %r12
        pushq   %rbx
        subq    $8, %rsp
        movl    $0, %r12d
        movl    $10, %ebx
        jmp     .L2
.L5:
        movl    %ebx, %r13d
        jmp     .L3
.L4:
        addl    $1, %r12d
        addl    $1, %r13d
.L3:
        cmpl    $10, %r13d
        jle     .L4
        subl    $1, %ebx
.L2:
        testl   %ebx, %ebx
        jg      .L5
        movl    %r12d, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT

        nop
        addq    $8, %rsp
        popq    %rbx
        popq    %r12
        popq    %r13
        popq    %rbp
        ret
```

Part 2: Switch

Please consider the following assembly and memory dump:

Hint: Recall that the gdb command x/g *SOME_ADDRESS_EXPRESSION* will examine an 8-byte word starting at the given address.

22

3.1(A): How many loops are there?

23 3 points

3.1(B)How would you describe the relationship among the loop(s). Choose one:

- One loop
- Nested
- Sequential
- Two or more of the above
- None of the above

24 2 points

3.1(C): If you had to choose one C Language loop construct to represent the loop(s) above, which of the following would you choose?

- While
- Do-While
- For

25 3 points

3.1(D): How many loop control variables are there,in total?

(Hint: A "loop control variable" is a variable that is evaluated as part of a loops test /and/ which is,or can be, changed within the loop's body or by its update (if a for loop).

```
(gdb) disassemble foo
Dump of assembler code for function foo:
   0x0000000000400550 <+0>:     cmp
$0x5,%esi
   0x0000000000400553 <+3>:     ja
0x40058b <foo+59>
   0x0000000000400555 <+5>:     mov
%esi,%eax
 0x0000000000400557 <+7>:     jmpq
*0x400630(,%rax,8)
   0x000000000040055e <+14>:    xchg
%ax,%ax
   0x0000000000400560 <+16>:    add
$0x2,%edi
   0x0000000000400563 <+19>:    mov
%edi,%eax
   0x0000000000400565 <+21>:    mov
$0x55555556,%edx
   0x000000000040056a <+26>:    sar
$0x1f,%edi
   0x000000000040056d <+29>:    imul
%edx
   0x000000000040056f <+31>:    sub
%edi,%edx
   0x0000000000400571 <+33>:    mov
%edx,%eax
   0x0000000000400573 <+35>:    retq
   0x0000000000400574 <+36>:    nopl
0x0(%rax)
   0x0000000000400578 <+40>:    add
$0xa,%edi
   0x000000000040057b <+43>:    lea
0x0(,%rdi,4),%edx
   0x0000000000400582 <+50>:    mov
%edx,%eax
   0x0000000000400584 <+52>:    retq
   0x0000000000400585 <+53>:    nopl
(%rax)
   0x0000000000400588 <+56>:    and
$0x1,%edi
   0x000000000040058b <+59>:    lea
(%rdi,%rsi,1),%edx
   0x000000000040058e <+62>:    mov
%edx,%eax
   0x0000000000400590 <+64>:    retq
```

26 2 points

What is the output of the code shown?

count:

27 2 points

3.2(A): Blank (A): 0x

28 2 points

3.2(B): Blank (B): 0x

29 2 points

3.2(C): Blank (C): 0x

30 2 points

3.2(D): Blank (D): 0x

31 2 points

3.2(E): Blank (E): 0x

```
End of assembler dump.
(gdb) disassemble 0x400550 Dump of
assembler code for function foo:
0x0000000000400550 <+0>: cmp
$0x5,%esi 0x0000000000400553 <+3>:
ja 0x40058b <foo+59>
0x0000000000400555 <+5>: mov
%esi,%eax 0x0000000000400557 <+7>:
jmpq *0x400630(,%rax,8)
0x000000000040055e <+14>: xchg
%ax,%ax 0x0000000000400560 <+16>:
add $0x2,%edi 0x0000000000400563
<+19>: mov %edi,%eax
0x0000000000400565 <+21>: mov
$0x55555556,%edx 0x000000000040056a
<+26>: sar $0x1f,%edi
0x000000000040056d <+29>: imul %edx
0x000000000040056f <+31>: sub
%edi,%edx 0x0000000000400571 <+33>:
mov %edx,%eax 0x0000000000400573
<+35>: retq 0x0000000000400574
<+36>: nopl 0x0(%rax)
0x0000000000400578 <+40>: add
$0xa,%edi 0x000000000040057b <+43>:
lea 0x0(,%rdi,4),%edx
0x0000000000400582 <+50>: mov
%edx,%eax 0x0000000000400584 <+52>:
retq 0x0000000000400585 <+53>: nopl
(%rax) 0x0000000000400588 <+56>: and
$0x1,%edi 0x000000000040058b <+59>:
lea (%rdi,%rsi,1),%edx
0x000000000040058e <+62>: mov
%edx,%eax 0x0000000000400590 <+64>:
retq End of assembler dump.
```

Please fill in the switch jump table corresponding to the gdb dump above. **Do not include any leading zeros and note that the answer should be in hexadecimal without the leading 0x**, as it is given.

```
(gdb) x/6g 0x400630
0x400630:        0x____(A)______
0x___(B)______
0x400640:        0x____(C)______
0x___(D)______
0x400650:        0x____(E)______
0x___(F)______
```

32 2 points

3.2(F): Blank (F): 0x

Question 4: Stack Use and Calling

4 points

Convention

Calling Convention and Stack Discipline

The following stack and register dump is from a Linux x86-64 machine like the shark hosts. It is taken immediately AFTER a function has been called, right before the first instruction within that function has been executed. The original function was written in the C Language.

```
(gdb) info registers
rax            0x6      6
rbx            0x0      0
rcx            0x4      4
rdx            0x9      9
rsi            0x8      8
rdi            0x6      6
rbp
0x7fffffffe0b0
rsp
0x7fffffffe0b0
r8
0x7ffff7dd5060
r9
0x7fffffffe528
r10            0x4      4
r11            0x0      0
r12            0x400440 4195392
r13
0x7fffffffe1d0
r14            0x0      0
r15            0x0      0
rip            0x40053d
0x40053d <add+16>

(gdb) x/10xg 0x7fffffffe0a8
0x7fffffffe0a8:
0x00007ffff7a44900
0x00007fffffffe0f0
0x7fffffffe0b8:
0x00000000004005e9
0x00007fffffffe1d8
0x7fffffffe0c8:
0x0000000700000000
0x0000000000400600
```

33 4 points

4.1(A) 1st argument:

34 4 points

4.1(B) 2nd argument:

35 4 points

4.1(C) 3rd argument:

36 4 points

4.1(D): Return address: 0x

37 4 points

Number of arguments:

38 5 points

4.1(E) C Language data type for 3rd argument:

- int
- float
- long
- double
- Unknowable
- None of the above

```
0x7fffffffe0d8:
0x0000000000400440
0x0000000900000004
0x7fffffffe0e8:
0x0000000600000008
0x0000000000000000
```

Please fill in the following, or indicate that the value is not knowable from the provided trace:

Question 5: Data

Part 1: Structs

Consider the following struct as compiled on a system using "natural alignment", i.e. the size of a data type is also its alignment requirement, and where chars are 1 byte, shorts are 2 bytes, ints are 4 bytes, and longs are 8 bytes, and then answer the questions that follow:

```
struct {
  char c;
  short s;
  long l;
  int i;
} initial;
```

Please answer the questions to the right.

Part 2: Arrays

Consider the following code as compiled and executed in an environment with 4-byte integers and 8-byte pointers:

```
int array1[4][5];
int **array2;
array2 = malloc (4*sizeof(int *));
for (int row=0; row<4; row++) {
  array2[row] = malloc
(5*sizeof(int));
}
```

Please answer the questions to the right.

39 2 points

5.1(A): How many bytes of alignment does the struct as a whole require?

40 2 points

5.1(B): How many bytes of padding does the compiler add before the first (char c) field?

41 2 points

5.1(C): How many bytes of padding does the compiler add after the last (int i) field?

42 2 points

5.1(D): How many bytes of alignment does the compiler add between fields, e.g. neither at the beginning nor at the end?

43 2 points

5.1(E): How many bytes can be saved in a single instance of the struct by reorganizing the fields?

44 1 point

5.1(F): Given the reorganized struct you contemplated for (E) above, how many bytes would be saved across an array of four (4) such structs as compared to an array of four (4) of the original structs?

45 2 points

5.2(A): In total, how many bytes are allocated, directly and/or indirectly, to *array1*? If you don't have enough information to answer or if the answer isn't knowable, write “-1”.

46 2 points

5.2(B): What is the minimum number of bytes allocated **directly** to *array2*?

47 2 points

5.2(C): In total, how many bytes are allocated, directly and/or indirectly, to *array2*? If you don't have enough information to answer or if the answer isn't knowable, write “-1”.

48 2 points

5.2(D): Consider the addresses of *array1[1][1]* and *array1[3][2]*. What is the absolute difference as measured in bytes? If you don't have enough information to answer or if the answer isn't knowable, write "-1".

49 2 points

5.2(E): Consider the addresses of *array2[1][1]* and *array2[3][2]*. What is the absolute difference as measured in bytes? If you don't have enough information to answer or if the answer isn't knowable, write "-1".

50 2 points

5.2(F): If the entirety of *array1* is initialized, is the value of array1[1][6], knowable? Yes or No

- Yes
- No

51 2 points

5.2(G): If the entirety of *array2,* including the indirect components, is initialized, is the value of array2[1][6], knowable? Yes or No

- Yes
- No

Question 6: Caching and Memory Access

This question tests your understanding of cache behavior, asks you to simulate and describe the behavior of the same memory access trace on two different cache configurations, asks you some questions about the performance, and then asks you about the impact of caching upon memory access time.

52 1 point

Blank (A)

Part 1: 2-Way Set-Associative Cache

Given the following information, please fill in the table below. **If no set bits are decoded, fill in 0 for the set number.**

The cache configuration for Part-1 is described as follows:

- 2-way set-associative (E=2)
- Address with = 6 bits
- Block size = 8 bytes
- 32byte total cache size

Time	Mem Addr (Hex)	Set (Decimal)	Tag (Binary)	Hit/Miss (H/M)	Ty Mi (C Co Ca N/
0	0x1A	(A)	(B)	(C)	(D
2	0X2A				(E)
3	0X05				
4	0X0A	(F)	(G)	(H)	(I)
5	0X23				
6	0X16				(J)
6	0X00				(K

Part 2: Fully-Associative Cache

Given the following information, please fill in the table below. **If no set bits are decoded, fill in 0 for the set #.**

- Fully associative (All cache lines in same set)
- Address with = 6 bits
- 3 tag bits
- 32byte total cache size

Time	Mem Addr (Hex)	Set (Decimal)	Tag (Binary)	Hit/Miss (H/M)	Ty Mi (C Co

53 1 point

Blank (B):

54 1 point

Blank (C)

- (H)it
- (M)iss

55 1 point

Blank (D)

- Cold
- Conflict
- Capacity
- N/A

56 1 point

Blank (E)

- Cold
- Conflict
- Capacity
- N/A

57 1 point

Blank (F)

58 1 point

Blank (G):

					Ca N/
0	0x1A	(A)	(B)	(C)	(D)
2	0X2A				
3	0X05				(E)
4	0X0A	(F)	(G)	(H)	(I)
5	0X23				(J)
6	0X16				
6	0X00				(K)

Part 3: Comparison

Please answer the question to the right.

Part 4: Memory Access

Consider a memory system with the following properties:

- Level 1 cache: SRAM, 10nS access tie
- Main memory: DRAM, 100nS access time.
- Cache hit rate: 95%

Please answer the questions to the right.

59 1 point

Blank (H)

- (H)it
- (M)iss

60 1 point

Blank (I)

- Cold
- Conflict
- Capacity
- N/A

61 1 point

Blank (J)

- Cold
- Conflict
- Capacity
- N/A

62 1 point

Blank (K)

- Cold
- Conflict
- Capacity
- N/A

63 1 point

Blank (A)

64 1 point

Blank (B):

65 1 point

Blank (C)

- (H)it
- (M)iss

66 1 point

Blank (D)

- Cold
- Conflict
- Capacity
- N/A

67 1 point

Blank (E)

- Cold
- Conflict
- Capacity
- N/A

68 1 point

Blank (F)

69 1 point

Blank (G):

70 1 point

Blank (H)

- (H)it
- (M)iss

71 1 point

Blank (I)

- Cold
- Conflict
- Capacity
- N/A

72 1 point

Blank (J)

- Cold
- Conflict
- Capacity
- N/A

73 1 point

Blank (K)

- Cold
- Conflict
- Capacity
- N/A

74 1 point

6.3: Did either cache configuration perform better for the given traces than the other? If so, how do you know

- They performed equally well for the given trace
- It isn't possible to know, given the traces provided
- The cache configuration in Part 1 had fewer hits than the cache configuration Part 2, so the cache configuration in Part 2 performed better.
- The cache configuration in Part 1 had fewer misses than the cache configuration in Part 2, so the cache configuration in Part 1 performed better.
- None of the above

75 1 point

6.4(A): What is the cache miss rate?
Fill in the blank: ___________%.

76 1 point

6.4(B): What is the cache miss penalty (in nS)?
Fill in the blank: ________nS.

77 1 point

6.4(C): What is the average access time to the nearest 0.01 nS?
Fill in the blank: ________ nS.

78 0 points

Feel free to provide us any feedback, comments, or notes here. For example, if you made any assumptions, etc. If you do, after the dust has settled (grades are back), please ping one of us and let us know that we should take a look. Remember -- grades can be adjusted at any time. And, we are humans, just like you. We're happy to discuss anything with you. Thanks!