Midterm Exam (Homework #6 / Homework #7)

This is a preview of the published version of the quiz

Started: Jan 13 at 10:52pm

Quiz Instructions

Instructions

- 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 160 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.
- We only expect the exam to take 70-90 minutes.
- The exam counts for the 25% "exam portion" of the midterm grade, but is reduced to counting as a "double homework" for the final grade.
- In order to make the exam an "invested but low stakes" experience, half of this exam's weight toward the final grade may be dropped as one of the two "homework drops", but the full weight can't be dropped.

Question	**Topic**	**Points**
1	Integers	10
2	Floats	15
3	Array Sizes	5
4	Array Arithmetic	5
5	Structs and Alignment	12
6	Assembly: Basic	8
7	Assembly: Switch	15
8	Assembly: Loops and Conditionals	12
9	Memory Hierarchy	5
10	Locality	3
11	Caching	10
Total:		100

Question 1

10 pts

1. Integers (10 points, 2 points per blank)

This question is based upon the following declaration on a **machine using 6-bit two’s complement arithmetic for signed integers.**

Fill in the empty boxes in the table below.

- **Show all digits for the "Binary" column, including any leading 0s. Do not add spaces, letters, annotations, groupings, units, etc.**
- You need not fill in entries marked with “--”.
- TMax denotes the largest positive two’s complement number
- TMin denotes the most negative two’s complement number.

Expression	**Decimal Representation**	**Binary Representation**
0 + 0		-
Tmin		-
-28 - 5		-
--	-7	
--		101101

2. Floats (15 points)

The floating point questions below are based upon an IEEE-like floating point format with the following specification:

- 9-bit width
- There is s = 1 sign bit
- There are k = 4 fraction bits
- Wherever rounding is necessary, round-to-even should be used. In addition, you should give the rounded value of the encoded floating point number.
- This question asks about the undecoded bits within the IEEE-like representation, answer in binary without spaces, groupings, annotations, letters, units, etc.

Question 2

1 pts

Question 2: Floats (15 points, 1 point for this part)

2(A) (1 points) What is the bias? (Answer in decimal)

Question 3

1 pts

Question 2: Floats (15 points, 1 point for this part)

2(B) (1 points) What is the exponent for denormalized numbers? (Answer in decimal)

Hint: This question asks about the actual, decoded exponent, not the bit pattern or value of the bit pattern in isolation.

Question 4

1 pts

Question 2: Floats (15 points, 1 point for this part)

2(C) (1 points) What is the maximum exponent for normalized numbers? (Answer in decimal)

Hint: This question asks about the actual, decoded exponent, not the bit pattern or value of the bit pattern in isolation.

Question 5

1 pts

Question 2: Floats (15 points, 1 point for this part)

2(D) (1 points) What exponent bit pattern is used for special values (infinity, NaN, etc)?

Hint: This question asks about the undecoded bits within the IEEE-like representation, answer in binary without spaces, groupings, annotations, letters, units, etc.

Question 6

11 pts

Question 2: Floats (15 points, 1 point for each blank in this part)

This question is based upon an IEEE-like floating point format with the following specification:

- 9-bit width
- There is s = 1 sign bit
- There are k = 4 fraction bits
- Wherever rounding is necessary, round-to-even should be used. In addition, you should give the rounded value of the encoded floating point number.
- If the question asks about the undecoded bits within the IEEE-like representation, answer in binary without spaces, groupings, annotations, letters, units, etc.
- **For the 3rd column: Answer as a fully reduced decimal fraction**, i.e. use the smallest denominator possible without a fractional numerator.

2(E-I) (1 point per blank) Fill in the following:

Value	**Binary Representation**	**Rounded Value as a reduced decimal fraction**	**Rounding *ERROR* as a reduced decimal fraction**
3/16		--	--
-9/1024		--	--
-Infinity -Inf		--	--
19/2048		**Fully reduced:** + /	**Fully reduced:** (neglect sign) /
17/2048		**Fully reduced:** +	--

		/	

Question 7

5 pts

3. Arrays Sizes (5 points, 2.5pts per part)

Consider the following definitions in an x86-64 system with 8-byte pointers and 2-byte shorts, 4-byte ints, and 8-byte longs. Answer with only a decimal number

Definition A	**Definition B**
int numbersA[5][3][2];	char *numbersB = numbersA;

3(A) (2.5 points): How many bytes are allocated to numbersA? (Write “UNKNOWN” if not knowable): Bytes

Hint: Think sizeof(); **answer with only a whole decimal number.** No units. no fractions. No weirdness.

Bytes

3(B) (2.5 points): Consider the following line of code. What is the difference in value of `numbersB` before and after the increment? (Write “UNKNOWN” if not knowable):

```
numbersB = numbersB + 1;
```

Hint: **Answer with only a whole decimal number**. No units. no fractions. No weirdness.

Bytes

Question 8

5 pts

4.Array Arithmetic (5 points, 2.5 points per blank)

(2.5 points): Consider the following definitions as implemented on a shark machine, i.e. x86-64 with 1-byte chars, 2-byte shorts, 4-byte ints, 8-byte longs, and 8-byte pointers.

Definition A	**Definition B**

unsigned long numbersA[5][3];	unsigned long *numbersB = numbersA;

For each part below, write "UNKNOWN" (without quotes) if there is not enough information to answer the question or the answer is otherwise unknowable. **Otherwise, answer with only a whole decimal number**. No units. no fractions. No weirdness.

4(A) (2.5 points) What is the difference, i.e. number of bytes, between the addresses of numbersA[1][2] and numbersA[2][1]?

Bytes

4(B) (2.5 points) What is the difference, i.e. number of bytes, between numbersB[1][2] and numbersB[2][1]?

Bytes

5. Structs and Alignment (12 points, 2 points per part)

The struct questions below are based upon the following definition as implemented on a shark machine, i.e. x86-64 with 1-byte chars, 2-byte shorts, 4-byte ints, 8-byte longs, and 8-byte pointers.

```
struct {
  char c1;
  char c2;
  short s;
  long l;
  int i;
} exam;
```

Assume a system which requires "natural alignment" (the alignment presented in lectures), i.e. each type needs to be aligned to a multiple of its data type size.

Question 9

2 pts

5. Structs and Alignment (12 points, 2 points per part)

5(A) (2 points) What is the value of sizeof(struct exam)?

Question 10

2 pts

5. Structs and Alignment (12 points, 2 points per part)

5(B) (2 points) How many bytes of padding does the compiler introduce after *s*?

Question 11

2 pts

5. Structs and Alignment (12 points, 2 points per part)

5(C) (2 points) How many bytes of padding does the compiler introduce after l?

Question 12

2 pts

5. Structs and Alignment (12 points, 2 points per part)

5(D) (2 points) How many bytes of padding does the compiler introduce after i?

Question 13

2 pts

5. Structs and Alignment (10 points, 2 points per part)

5(E) (2 points) Which of the following field orderings minimize the amount of padding introduced by the compiler?

○ c1, c2, s, l, i

○ s, c1, c2, l, i

○ None of the above

○ c1, s, c2, i, l

○ c1, c2, s, i, l

○ All of the above

Question 14

2 pts

5. Structs and Alignment (12 points, 2 points per part)

5(F) (2 points) Assuming the fields of the struct were organized optimally by the programmer, what would be the value of sizeof (struct exam)?

Question 15

8 pts

6. Assembly-Basic (8 points, 2 points per part)

Please consider the following assembly:

```
movq   32(%rdi), %rax          # Assume that %rdi holds the starting address of an array
movq   %rax, 16(%rdi)
movq   8(%rdi), %rax
leaq   (%rax,%rax,4), %rax
salq   $2, %rax
movq   %rax, 24(%rdi)
```

6(A) (2 points) What type are the elements of the array? [Select]

6(B) (2 points) Upon what element, i.e. via what index, of the array is arithmetic being performed?
[Select]

6(C) (2 points) What arithmetic is being performed upon that element? [Select]

6(D) (2 points) Which one of the following elements of the array never has values assigned to it by this assembly code?
[Select]

7. Assembly-Switch (15 points)

Consider the following code, which was compiled from C Programming Language source code containing one switch statement and no (zero) if statements:

```
Dump of assembler code for function foo:
   0x0000000000400530 <+0>:     cmp    $0x5,%esi
   0x0000000000400533 <+3>:     ja     0x400555 <foo+37>
   0x0000000000400535 <+5>:     mov    %esi,%esi
   0x0000000000400537 <+7>:     jmpq   *0x400620(,%rsi,8)
   0x000000000040053e <+14>:    lea    0x1(%rdi),%eax
   0x0000000000400541 <+17>:    retq
   0x0000000000400542 <+18>:    shl    $0x2,%edi
   0x0000000000400545 <+21>:    lea    (%rdi,%rdi,1),%eax
   0x0000000000400548 <+24>:    retq
   0x0000000000400549 <+25>:    lea    0x3(%rdi),%eax
   0x000000000040054c <+28>:    test   %edi,%edi
   0x000000000040054e <+30>:    cmovns %edi,%eax
   0x0000000000400551 <+33>:    sar    $0x2,%eax
   0x0000000000400554 <+36>:    retq
   0x0000000000400555 <+37>:    mov    %edi,%eax
   0x0000000000400557 <+39>:    shl    $0x4,%eax
   0x000000000040055a <+42>:    retq
End of assembler dump.
```

Consider also the following dump:

```
(gdb) x/16gx 0x400610
0x400610:       0x0000000000020001      0x0000000000000000
0x400620:       0x000000000040053e      0x000000000040053e
0x400630:       0x0000000000400542      0x0000000000400545
0x400640:       0x0000000000400555      0x0000000000400549
0x400650:       0x0000003c3b031b01      0xfffffdb000000006
0x400660:       0xfffffdf000000088      0xfffffee000000058
0x400670:       0xffffff0b000000b0      0xffffff40000000c8
0x400680:       0xffffffb0000000e8      Cannot access memory at address 0x400688
```

Question 16

3 pts

7. Assembly-Switch (15 points)

7(A) (3 points) Which of the following executes for case 3?

○ lea 0x1(%rdi),%eax

○ mov %edi,%eax

○ shl $0x2,%edi

○ lea 0x3(%rdi),%eax

○ shl $0x4,%eax

○ None of the above

○ lea (%rdi,%rdi,1),%eax

Question 17

3 pts

7. Assembly-Switch (15 points)

7(B) (3 points) Which integer input values are managed by non-default cases of the switch statement? Check all that apply.

□ 1

□ 2

□ None of the above

□ 4

□ 3

□ Other value(s) in addition to those above

□ 5

Question 18

3 pts

7. Assembly-Switch (15 points)

7(C) (3 points) If there is a default case, at what address, in hex, does the begin?

- If there isn't a default case, write NONE.
- When writing an address, please do not include any leading 0s, prefixes or suffixes, or any spaces, and please write any letters in either all upper or all lower case, not mixed case.

Your answer: [blank]

Question 19

3 pts

7. Assembly-Switch (15 points)

7(D) (3 points) Which of the following case(s), if any, consist of exactly the same code as least one other case (no extra code, no code missing)? Check all that apply. [exact_same]

□ 3

□ 2

□ 0

□ 1

□ 5

□ None of the above

□ 4

Question 20

3 pts

7. Assembly-Switch (15 points)

7(E) (3 points) Which case(s), if any, fall through to the next case *after executing some of their own code*?

□ None of the above

□ 4

□ 0

□ 3

□ 5

□ 2

□ 1

8. Loops and Conditionals (12 points)

Consider the following code, under the assumption that it was compiled in the same environment using the same "shark machine" toolset you've used all semester:

```
(gdb) disassemble loop
Dump of assembler code for function loop:
   0x000000000040059d <+0>:     push   %rbp
   0x000000000040059e <+1>:     mov    %rsp,%rbp
   0x00000000004005a1 <+4>:     push   %r14
   0x00000000004005a3 <+6>:     push   %r13
   0x00000000004005a5 <+8>:     push   %r12
   0x00000000004005a7 <+10>:    push   %rbx
   0x00000000004005a8 <+11>:    sub    $0x20,%rsp
   0x00000000004005ac <+15>:    mov    %edi,-0x34(%rbp)
   0x00000000004005af <+18>:    mov    %esi,-0x38(%rbp)
   0x00000000004005b2 <+21>:    mov    %edx,-0x3c(%rbp)
   0x00000000004005b5 <+24>:    mov    $0x0,%r13d
   0x00000000004005bb <+30>:    mov    $0x0,%r14d
   0x00000000004005c1 <+36>:    mov    $0x0,%ebx
   0x00000000004005c6 <+41>:    mov    $0x0,%r12d
   0x00000000004005cc <+47>:    mov    $0x64,%eax
   0x00000000004005d1 <+52>:    cmpl   $0x64,-0x34(%rbp)
   0x00000000004005d5 <+56>:    cmovge -0x34(%rbp),%eax
   0x00000000004005d9 <+60>:    mov    %eax,-0x34(%rbp)
   0x00000000004005dc <+63>:    mov    $0x64,%eax
   0x00000000004005e1 <+68>:    cmpl   $0x64,-0x38(%rbp)
   0x00000000004005e5 <+72>:    cmovge -0x38(%rbp),%eax
   0x00000000004005e9 <+76>:    mov    %eax,-0x38(%rbp)
   0x00000000004005ec <+79>:    mov    $0x0,%ebx
   0x00000000004005f1 <+84>:    jmp    0x40062c <loop+143>
   0x00000000004005f3 <+86>:    mov    $0x0,%r13d
   0x00000000004005f9 <+92>:    mov    $0x0,%r12d
   0x00000000004005ff <+98>:    jmp    0x400620 <loop+131>
   0x0000000000400601 <+100>:   lea    -0x24(%rbp),%rax
   0x0000000000400605 <+104>:   mov    %rax,%rsi
   0x0000000000400608 <+107>:   mov    $0x400710,%edi
   0x000000000040060d <+112>:   mov    $0x0,%eax
   0x0000000000400612 <+117>:   callq  0x4004a0 <__isoc99_scanf@plt>
   0x0000000000400617 <+122>:   mov    -0x24(%rbp),%eax
   0x000000000040061a <+125>:   add    %eax,%r13d
   0x000000000040061d <+128>:   add    %ebx,%r12d
   0x0000000000400620 <+131>:   cmp    -0x38(%rbp),%r12d
   0x0000000000400624 <+135>:   jl     0x400601 <loop+100>
   0x0000000000400626 <+137>:   add    %r13d,%r14d
   0x0000000000400629 <+140>:   add    $0x1,%ebx
   0x000000000040062c <+143>:   cmp    -0x34(%rbp),%ebx
   0x000000000040062f <+146>:   jl     0x4005f3 <loop+86>
   0x0000000000400631 <+148>:   cmp    $0x63,%r14d
   0x0000000000400635 <+152>:   jg     0x40063e <loop+161>
   0x0000000000400637 <+154>:   mov    $0x64,%eax
   0x000000000040063c <+159>:   jmp    0x400641 <loop+164>
   0x000000000040063e <+161>:   mov    %r14d,%eax
   0x0000000000400641 <+164>:   add    $0x20,%rsp
   0x0000000000400645 <+168>:   pop    %rbx
   0x0000000000400646 <+169>:   pop    %r12
   0x0000000000400648 <+171>:   pop    %r13
   0x000000000040064a <+173>:   pop    %r14
   0x000000000040064c <+175>:   pop    %rbp
   0x000000000040064d <+176>:   retq
End of assembler dump.
```

Question 21

3 pts

8. Loops and Conditionals (12 points)

8(A) (3 points) How many loops are in the code?

○ 0

○ 4 or more

○ 1

○ 3

○ 2

Question 22

3 pts

8. Loops and Conditionals (12 points)

8(B) (3 points) What is the relationship between/among the loop(s)?

○ They are all nested

○ There is only one loop, so there is no relationship between or among loops

○ Nested and one after another

○ One after another

Question 23

3 pts

8. Loops and Conditionals (12 points)

8(C) (3 points) Which of the following are true? Check all that apply.

□ The loop control variable (the variable used to test whether to loop again or exit the loop) of one loop is used by or within another loop

□ Two or more loops have a stopping value in common, e.g. progress up to or down to the same number.

□ Two or more loops have a loop control variable in common

Question 24

3 pts

8. Loops and Conditionals (12 points)

8(D) (3 points) How many times is the ?-operator likely used in the source C Language code?

○ 0

○ 3

○ 1

○ 4 or more

○ 2

Question 25

5 pts

9. Memory Hierarchy (5 points)

Your goal is to design a memory system with an average access time of 1.1nS or less.

You are given the following:

- L1 cache with an access time of 1ns and a hit rate of 99%
- L2 cache with an access time of 6ns
- Main memory with an access time of 106ns

The access times for L2 and Main memory are end-to-end times, i.e., the L2 time includes the time taken to check the L1 and the Main memory time includes the time taken to check the L1 and L2.

What is the maximum permissible L2 cache miss rate, expressed as a percentage, e.g. 0 for 0% or 10 for 10%, or 12 for 12%, or 50 for 50%, or 92 for 92%. Please enter only a 1- or 2-digit number with**out** the % -sign. Do not enter fractions, values less than 1, etc. Round **up** to the nearest percent.

Question 26

3 pts

10. Locality (3 points)

Consider a cache with 8 sets, 2 lines/set, and a block size of 16 bytes on a system with 4-byte ints.

What is the maximum stride (index step) size while sequentially accessing a 1D int array to maintain a cache miss rate of no more than 42%?

11. Caching (10 points)

Given a model described as follows:

- Number of sets: 8
- Total size: 64 bytes (not counting meta data)
- Block offset bits: 2
- Replacement policy: Set-wise LRU
- 8-bit addresses

Question 27

1 pts

11. Caching (10 points)

11(A) (1 point) How many lines per set?

Question 28

1 pts

11. Caching (10 points)

11(B) (1 point) How many bytes per block?

Question 29

8 pts

11. Caching (10 points)

11(C) (8 points, 0.5 points each blank): Consider the following memory access trace, which is in order and begins at the beginning of time. For each of the following memory accesses, 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:

Question Number	**Address**	**Hit or Miss?** **Circle one (per row):**	**Miss Type?** **Circle one (per row)**
11(C)(1)	0xA2	[Select]	[Select]
11(C)(2)	0xD0	[Select]	[Select]
11(C)(3)	0XD7	[Select]	[Select]
11(C)(4)	0X92	[Select]	[Select]
11(C)(5)	0XD3	[Select]	[Select]
11(C)(6)	0XB2	[Select]	[Select]
11(C)(7)	0XA1	[Select]	[Select]
11(C)(8)	0X92	[Select]	[Select]

Quiz saved at 10:53pm

Submit Quiz