

Final Exam
CSC 252
13 December 2021
Computer Science Department
University of Rochester

Instructor: Alan Beadle

TAs: Abhishek Tyagi, Matthew DeWeese, Elana Elman, Yifan "Iven" Jiang, Yanghui "Woody" Wu, Zeyu "George" Wu

Name: _____

Problem 0 (3 points):	_____
Problem 1 (16 points):	_____
Problem 2 (14 points):	_____
Problem 3 (17 points):	_____
Problem 4 (21 points):	_____
Problem 5 (22 points)	_____
Total (93 points):	_____
Extra Credit (6 points)	_____

Remember "**I don't know**" is given 15% partial credit, but you must erase everything else. This does not apply to extra credit questions.

Your answers to all questions must be contained in the given boxes. Use spare space to show all supporting work to earn partial credit.

You have 3 hours to work.

Please sign the following. I have not given nor received any unauthorized help on this exam.

Signature: _____

GOOD LUCK!!!

Problem 0: Warm-up (3 Points)

What's the most surprising thing about computers you learned from 252?

Problem 1: Number Representation (16 points)

(3 points) Convert the decimal number 443 to hexadecimal.

(3 points) Put $5\frac{7}{16}$ into the binary normalized form.

The IEEE introduces a floating-point standard consisting of 10 bits, of which 4 bits are reserved for the exponent and 5 bits are reserved for the fraction.

(1 point) What is the bias in this standard?

(3 points) If possible, **precisely** represent the number $\frac{35}{128}$ in this 10-bit standard, writing your answer in binary. If not possible, explain why.

(3 points) If possible, **precisely** represent the number $\frac{65}{128}$ in this 10-bit standard, writing your answer in binary. If not possible, explain why.

(3 points) What is the largest denormalized value in this 10-bit format? **Write your answer in binary normalized form.**

Problem 2: Miscellaneous (14 points)

Part a) (3 points) If a program is 80% parallelizable and you have an infinite number of CPUs, what is the maximum possible speedup factor? (round to the nearest integer)

Part b) (3 points) Suppose a CPU is using a 2-bit branch predictor. The last five branches were: not taken, taken, taken, taken, not taken. What does the branch predictor predict next? If it is wrong, then what will it predict after that?

Part c) (5 points) Consider the following assembly code fragment:

```
.L1:  
dec %rax  
mov %rax, (%r11)  
inc %r11  
cmpq %rax, $0x01  
jge .L1  
sub $0x99, %rsi  
mov %rsi, %rax  
ret
```

Suppose that this code executes on a **four-stage** pipeline with the following stages: Fetch, Decode, Execute, and Mem+writeback. Assume that this CPU has an **always-wrong** branch predictor, which always makes the wrong prediction.

If the conditional branch is **taken exactly once**, how many cycles are needed to fully execute this code snippet? Assume that all data dependencies are handled by data forwarding, and assume that the branch misprediction is detected in the execute stage. **Hint: Remember that the pipeline starts empty, and the program is not done until the last instruction has finished all stages.**

Part d) (3 points) Suppose there is a C struct containing (in some unknown order) an array of 2 chars, an array of 6 integers, and a pointer to a long int. Assume the code will run on a 64-bit CSUG machine. What is the minimum possible size of this struct? **Hint:** the struct alignment/padding rules treat the elements of arrays as individual elements. You don't need to write any code.

Problem 3: Assembly Programming (17 points + 2 points extra credit)

Consider the following objdump output, produced from a C function which has been compiled with gcc:

```
0000000000001135 <foo>:
   1135: 89 7c 24 fc      mov     %edi, -0x4(%rsp)
   1139: 89 74 24 f8      mov     %esi, -0x8(%rsp)
  113d: 83 7c 24 fc 00   cml     $0x0, -0x4(%rsp)
   1142: 79 04           jns    1148 <foo+0x13>
   1144: f7 5c 24 fc      negl   -0x4(%rsp)
   1148: 83 7c 24 f8 00   cml     $0x0, -0x8(%rsp)
   114d: 79 04           jns    1153 <foo+0x1e>
   114f: f7 5c 24 f8      negl   -0x8(%rsp)
   1153: 8b 44 24 fc      mov     -0x4(%rsp),%eax
   1157: 0f af 44 24 f8   imul   -0x8(%rsp),%eax
  115c: c3             retq
```

(3 points) How many arguments does this function have, and what are their types?

(3 points) Write one line of C code to call this function with the right number of arguments, **at least one of which is negative**. Remember to assign the return value to a variable of an appropriate type. Then write what the actual return value will be.

(2 points extra credit) If this function had one more argument, where would the value of that argument be stored when the function is called? Be very specific for any credit.

Part b) (7 points) Some students wrote a CSC252 project in assembly. They discover right before submission that the machine used for grading has a faulty **ret** instruction. They decide to rewrite the project without using **ret**.

(3 points) On a **correct 64-bit machine**, which register(s) does **ret** update? Explain how the new register values are determined.

(4 points) Which **two** instructions could they combine to replace the faulty **ret**? Provide **just the names** of the instructions in the order they are used.

Part c) (4 points) Write a few lines of x86_64 assembly code to conditionally call a function `bar()` if the values contained in `rcx` and `rdx` are equal. Make up arbitrary values for any addresses needed. You may use labels like `“.L1:”`.

Problem 4: Cache (21 points + 1 point extra credit)

You have been asked to design a byte-addressable, 2-way associative cache. Each cache line will hold 2 bytes. There will be 16 sets, and you will use an LRU replacement policy.

Unfortunately, you can only afford to have up to 720 bits in the cache. **Hint: Your cache will use exactly this number of bits.** This includes both data and overhead bits. You have decided to use a write-through policy to save some bits.

(2 points) How many bits are needed **per set** to implement the LRU policy?

(2 points) How many bits of an address are used for the set index? How many bits of an address are used for the line offset?

(2 points) What is the total number of overhead bits in the cache?

(3 points) How many tag bits does each cache **line** have?

(3 points) What is the maximum amount of physical memory this machine can have? (In bytes)

Assume that the machine runs only one program, which will generate the following memory access sequence:

Access	Address
1	0000 0101 1000
2	0000 0111 1001
3	0000 1111 0001
4	0000 0001 1001
5	0000 0111 1000
6	0000 1111 0000
7	0000 0101 1000
8	0000 0101 1001

(3 points) Assuming the cache is initially empty, how many cache misses will the program generate?

(3 points) How many valid bits will be set to 1 (valid) after the program runs?

(3 points) If the cache used a random replacement policy instead of LRU, what is the maximum number of misses that could happen in the above trace?

(1 point extra credit) How many total bits did you save by using a write-through design and what are those bits called?

Problem 5: Virtual Memory (22 points + 3 points extra credit)

You are building a byte-addressable machine with virtual memory support. The virtual address space is 128 KB (2^{17} bytes). The physical memory size is 32KB. The system uses a one-level page table.

Part a) Basic organization

(4 points) Assume the page size is 512 bytes. How many bits are used for the physical page number (PPN), and how many for the virtual page number (VPN)?

(2 points) What is the size of each PTE, assuming that the only overhead in the PTE is one valid bit and one dirty bit?

(4 points) Suppose a program attempts to read a value from memory, but the PTE for the page containing that value has a valid bit of 0. Describe the events that might happen as a result of this. **Be especially sure to mention any possible changes to the page table.** You may assume there is no TLB for this question.

(3 points) Suppose the system has a TLB with a fixed number of entries. What effect would decreasing the page size have on the TLB hit rate? Why? Explain your answer to receive ANY credit.

(3 points extra credit) Assuming the original 512 byte page size, how much memory could be saved by splitting the page table into multiple levels?

Part b) Performance

Suppose you build the machine described above, with a 512 byte page size and a 2-entry TLB that uses LRU replacement.

Consider the C code below. Assume:

1. A 1-level page table translation scheme with the page table starting empty, and
2. arr is aligned such that it starts at the start of a page.

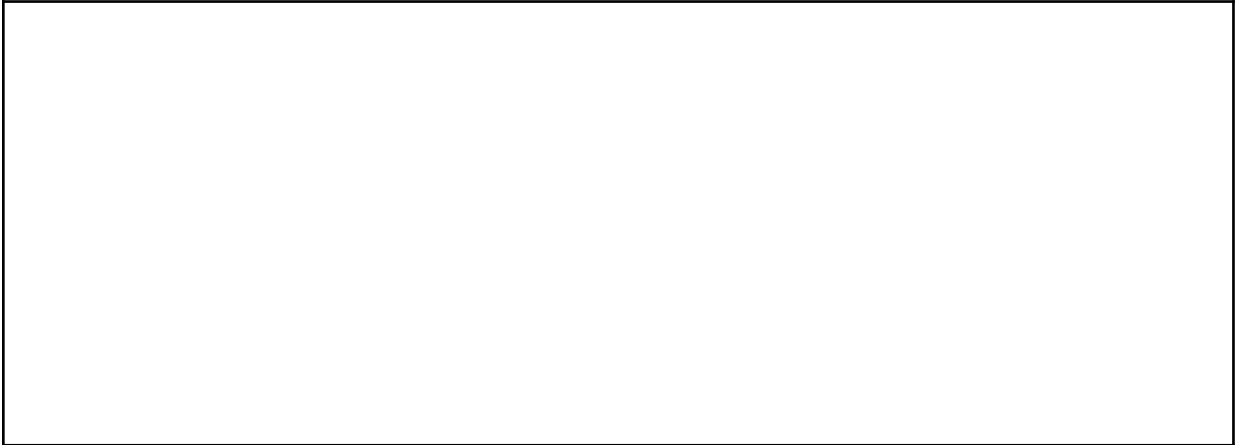
```
void func() {
    char arr[3000];
    for(int i = 0; i < 8; i++) {
        if(rand_0_1() == 1) {
            printf("%c\n", arr[(i << 8)]);
        } else {
            printf("%c\n", arr[i]);
        }
    }
}
```

rand_0_1() is a function that returns either 0 or 1, at random.

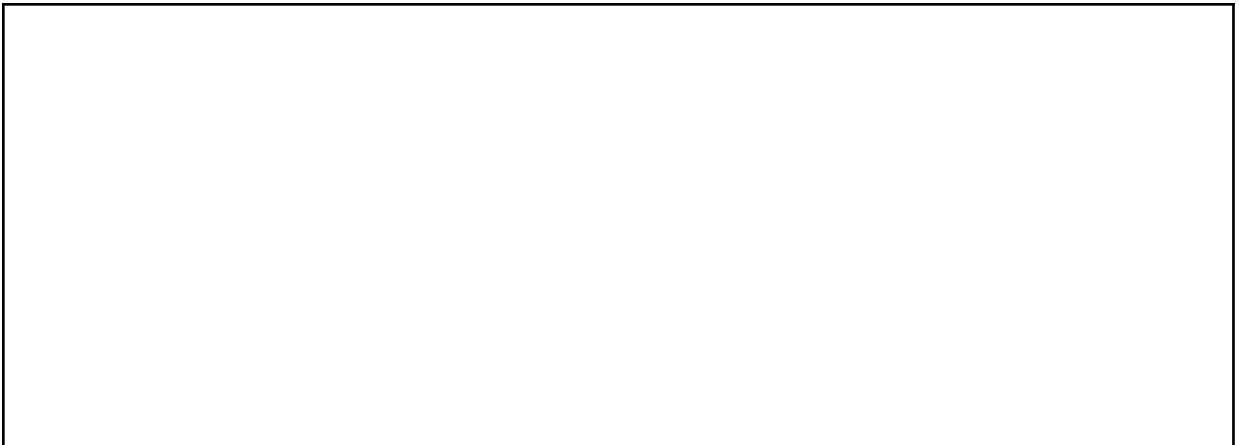
For the following 2 questions, only count page faults generated by accessing elements of arr.

(3 points) What is the **maximum** number of page faults that this code could possibly generate? **Explain.**

(3 points) What is the **minimum** number of page faults that this code could possibly generate? **Explain.**

A large, empty rectangular box with a black border, intended for the student to write their answer to the first question.

(3 points) What is the **maximum** number of TLB misses that this code could possibly generate? **Explain.**

A large, empty rectangular box with a black border, intended for the student to write their answer to the second question.