# Final Exam

**CSC 252**
**6 May 2020**
**Computer Science Department**
**University of Rochester**

**Instructor**: Yuhao Zhu
**TAs**: Daniel Busaba, Sudhanshu Gupta, Mandar Juvekar, Max Kimmelman, Weituo Kong, Jiahao Lu, Vladimir Maksimovski, Nathan Reed, Yawo Alphonse Siatitse, Yudi Yang, Shuang Zhai, Prikshet Sharma

**Name**: _____

| | |
|---|---|
| Problem 0 (2 points): | _____ |
| Problem 1 (28 points): | _____ |
| Problem 2 (20 points): | _____ |
| Problem 3 (28 points): | _____ |
| Problem 4 (24 points): | _____ |
| Total (100 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!!!**
**Have a great summer break!**

**Problem 0: Warm-up (2 Points)** What's your favourite register?

| |
|---|
| Any register is accepted. |

**Problem 1: Miscellaneous (26 points)**

**Part a) (6 points)** What is the result of adding the value $11_2$ to the value 0x40? Express your answer in octal (base 8).

| |
|---|
| 103 |

**Part b) (8 points)** Consider the following assembly program. The value 5 is contained in `rcx` and the value 2 is contained in `rdx` when execution starts.

```
          add rcx, rdx
          mov 3, r11
     .L1: dec rcx
          mov r8, r9
          cmp 0,1
          beq 0x584fc0
          inc r11
     .L2: cmp rcx, 0
          jne .L1
     .L3: add r8, r9
```

**(4 points)** What is the value stored in `r11` when `.L3` is reached?

| |
|---|
| 10 |

**(4 points)** What is an instruction that could be substituted in at `.L2` without changing the functionality of the program?

| |
|---|
| "test rcx", "and rcx, rcx", others |

**Part c) (4 points)** Give code for a C function that takes one integer as input and returns that integer multiplied by 5. You may not use the * operator and you may not use a loop of any kind. You may use at most three operators.

```
int mul5(int num) {
   return (num << 2) + num;
}
```

**Part d) (4 points)** Give two reasons the pipeline may stall during execution.

Data dependency, control dependency, cache miss, etc.

**Part e) (4 points)** Consider the following C program. Assume that it executes on a CSUG machine. Reminder: the "`gets`" function reads a string, ending with a newline, from standard input into the given buffer (it will read until it sees a newline). Assume that you (the user who is giving input to the program) can figure out the address of every function in memory (e.g., from disassembly/debugger).

```
int func1() {
    printf("hello from function 1\n");
    printf("this is a neat function\n");
    return 1;
}

int func2() {
    char answer1[100];
    printf("hello from function 2\n");
    printf("What is the airspeed velocity of an unladen swallow? ");
    gets(answer1);
    return 2 + strncmp(answer1, "African or European swallow?", 99);
}

int main(int argc, char** argv) {
    func2();
    return 0;
}
```

Which of the following is a possible output of this program? Select ALL correct answers (there may be multiple).

**A**
hello from function 2
What is the airspeed velocity of an unladen swallow?

**B**
hello from function 1
What is the airspeed velocity of an unladen swallow?
hello from function 2

**C**
hello from function 2
What is the airspeed velocity of an unladen swallow?
this is a neat function

**D**
hello from function 2
What is the airspeed velocity of an unladen swallow?
hello from function 1
this is a neat function

**E**
hello from function 2
What is the airspeed velocity of an unladen swallow?
<program crashes after user input>

| |
|---|
| A, C, D, E |

**Problem 2: Floating-Point Arithmetic (20 points)**

**Part a) (4 points)** Express 4 ⅜ in binary normalized form.

| |
|---|
| 1.00011 * 2^2 |

**Part b) (16 points)** A new IEEE-consistent floating-point representation is being developed which uses 8 bits. Given below are the binary representations of 4 floating-point numbers in this representation.

A. 1111 1011
B. 0010 0100
C. 0011 0001
D. 1011 0001

Below are 2 floating-point numbers that are the sum of some pairs of the numbers above:

1. 0000 0000
2. 0011 0100

**(4 points)** Which 2 numbers in A, B, C, and D generate the sum 2 above ? (You can do this without calculating anything) (Choose two from A, B, C, or D)

| |
|---|
| B and C |

**(4 points)** How many exponent bits are used?

| |
|---|
| 4 |

**(4 points)** How many fraction bits are used?

| |
|---|
| 3 |

**(4 points)** What is the bias?

| |
|---|
| 7 |

**Problem 3: Cache (28 points)**

You have been asked to design a byte-addressable, 4-way associative cache (meaning that each set in the cache can hold 4 cache lines). You have decided that since it is complicated to implement a Least Recently Used (LRU) policy for a set of 4 cache lines, you will use another replacement policy.

The policy you chose to use instead is a variant of a replacement policy called Not Most Recently Used (NMRU). This policy guarantees that the most recently used cache line in each set will *not* be replaced, and instead, some other cache line is selected for replacement.

The way you will implement this policy is by assigning an index to each line in the set (either 0, 1, 2, or 3), and keeping track of the index of the most recently used cache line in each set with MRU bits that are cache overhead (the only overhead in the cache will be valid, tag, and MRU bits). If the cache set is not full, new cache lines will be placed in the set at the lowest free index. If the cache set is full, the table below indicates the replacement policy.

| Index of most recently used cache line | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Index of cache line that will be replaced next | 3 | 0 | 1 | 2 |

You took notes of all the rest of the specifications of your cache on a whiteboard, but you accidentally left it next to an open window when it rained, and some of your notes washed away. Here are your remaining notes:

- the size of each cache line is 4 bytes
- the number of overhead bits required for each cache set is 26
- the cache will hold a total of 512 cache lines.

**Part a) (4 points)** How many bits per set are needed to implement the NMRU policy?

| 2 |
|---|

**Part b) (4 points)** How many bits are needed for a tag?

| 5 |
|---|

**Part c) (4 points)** What is the total physical memory on the machine that you are designing this cache for?

| |
|---|
| 16 KB |

**Part d) (16 points)** Now we want to compare memory access behavior under two different policies: NMRU and LRU. Assume that both of them have the same cache configuration as previous questions. The only difference is the replacement policy. Given the following sequence of memory accesses, please indicate whether a particular memory access will result in a cache hit or miss (Please fill in **Hit** or **Miss**).

Assume that the cache is initially empty. The hit/miss behaviors of the first seven accesses under both replacement policies are given.

| **Address** | **NMRU** | **LRU** |
|---|---|---|
| `0000 1010 0100 1001` | Miss | Miss |
| `0001 1000 0110 0100` | Miss | Miss |
| `0001 0010 0100 1010` | Miss | Miss |
| `0000 1100 0100 1001` | Miss | Miss |
| `0001 0010 0110 0111` | Miss | Miss |
| `0000 1010 0110 0101` | Miss | Miss |
| `0001 1000 0100 1010` | Miss | Miss |
| `0001 1110 0100 1011` | Miss | Miss |
| `0000 1010 0100 1011` | Hit | Miss |
| `0000 1100 0100 1000` | Miss | Hit |
| `0001 1000 0100 1011` | Miss | Hit |

**Problem 4: Virtual Memory (24 points)**

Assume a system which has the following characteristics:

1. Virtual address space is 64 KB and is byte addressable
2. Physical RAM is 16 KB and is byte addressable
3. Page size is 256 Byte
4. One level page table, where each page table entry contains a valid bit, a dirty bit, and the physical page number
5. Integer is 32 bits
6. PTBR is `0x2F5C`
7. There is a data TLB that stores two page table entries

**Part a) (4 points)** What is the size of each page table entry?

| |
|---|
| 6 (PPN) + 1 (valid) + 1 (dirty) = 8 bits |

**Part b) (4 points)** What would be the size of the page table?

| |
|---|
| 256 * 8 bits = 256 bytes |

**Part c) (4 points)** Does the system use a write-through policy or a write-back policy for writes to pages?

| |
|---|
| Write back |

**Part d) (12 points)** Consider the following C program:

```
void square(int a[16][16]) {
  for (int j=0; j < 16; j++) {
    for (int i=0; i < 16; i++) {
      a[i][j] = a[i][j] * a[i][j];
    }
  }
}
```

Suppose that the virtual address of matrix 'a' is `0x0300`. Assume the data TLB is empty when the code starts execution. Assume the following layout for each page table entry:

| Valid bit | Dirty bit | Physical page number |
|---|---|---|

The table below shows a part of the main memory before the code executes.

| **Address** | **Data** |
|---|---|
| 2F5B | 8B |
| 2F5C | 8C |
| 2F5D | CD |
| 2F5E | 8E |
| 2F5F | CF |
| 2F60 | C0 |

**(4 points)** To read the data in `a[3][10]`, what physical memory addresses are accessed by the program?

1. PTBR + VPN = 2F5F
2. 0F00 + ((16 * 3) + 10) * 4 = 0FE8

**(4 points)** How many data TLB misses does executing `square()` incur?

64

**(4 points)** How can you improve the program to have fewer TLB misses? Write both the technique and the new number of misses.

Switch the loop order. New misses = 4