CSE 410 Final Review

Question 1: Number Representation

(A) What is the value of the char 0b 1101 1101 in decimal?

(B) What is the value of char \( z = (0xB << 7) \) in decimal?

(C) Let char \( x = 0xC0 \). Give one value (in hex) for char \( y \) that results in both signed and unsigned overflow for \( x+y \).

For the rest of this problem we are working with a floating point representation that follows the same conventions as IEEE 754 except using 8 bits split into the following vector widths:

| Sign (1) | Exponent (4) | Mantissa (3) |

(D) What is the magnitude of the bias of this new representation?

(E) Translate the floating point number 0b 1100 1110 into decimal.
Question 2: Pointers & Memory

For this problem we are using a 64-bit x86-64 machine (little endian). Below is the factorial function disassembly, showing where the code is stored in memory.

```
0000000000040052d <fact>:
  40052d:  83 ff 00  cmpl   $0, %edi
  400530:  74 05  je 400537 <fact+0xa>
  400532:  83 ff 01  cmpl $1, %edi
  400535:  75 07  jne 40053e <fact+0x11>
  400537: b8 01 00 00 00  movl $1, %eax
  40053c: eb 0d  jmp 40054b <fact+0x1e>
  40053e: 57  pushq %rdi
  40053f:  83 ef 01  subl $1, %edi
  400542: e8 e6 ff ff ff  call 40052d <fact>
  400547: 5f  popq %rdi
  400548: 0f af c7  imull %edi, %eax
  40054b: f3 c3  rep ret
```

(A) What are the values (in hex) stored in each register shown after the following x86 instructions are executed? Remember to use the appropriate bit widths.

<table>
<thead>
<tr>
<th>Register</th>
<th>Value (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>0x0000 0000 0040 052D</td>
</tr>
<tr>
<td>%rsi</td>
<td>0x0000 0000 0000 0003</td>
</tr>
</tbody>
</table>

leal (%rdi, %rsi), %eax

movb 3(%rdi,%rsi,2), %bl

(B) Complete the C code below to fulfill the behaviors described in the inline comments using pointer arithmetic. Let char* cp = 0x40052D.

```c
char v1 = *(cp + _____); // set v1 = 0x75
int* v2 = (int*)((___________*)cp + 2); // set v2 = 0x40053D
```
Question 3: The Stack

The recursive Fibonacci sequence function `fib()` and its x86-64 disassembly are shown below:

```c
int fib (int n) {
    if (n<2)
        return 1;
    else
        return fib(n-2) + fib(n-1);
}
```

```
000000000040055d <fib>:
  40055d:  55    push %rbp
  40055e:  53    push %rbx
  40055f:  89 fb  mov %edi,%ebx
  400561:  83 ff 01  cmp $0x1,%edi
  400564:  7e 16  jle 40057c <fib+0x1f>
  400566:  8d 7f fe  lea -0x2(%rdi),%edi
  400569:  e8 ef ff ff  callq 40055d <fib>
  40056e:  89 c5  mov %eax,%ebp
  400570:  8d 7b ff  lea -0x1(%rbx),%edi
  400573:  e8 e5 ff ff ff  callq 40055d <fib>
  400578:  01 e8  add %ebp,%eax
  40057a:  eb 05  jmp 400581 <fib+0x24>
  40057c:  b8 01 00 00 00  mov $0x1,%eax
  400581:  5b    pop %rbx
  400582:  5d    pop %rbp
  400583:  c3    retq
```

(A) In no more than a sentence, explain what the instruction at address 0x40055f does (in terms of the function – don’t be too literal) and why it is necessary.
(B) How much space (in bytes) does this function take up in our final executable?

(C) Calling fib(4): How many total fib stack frames are created?

(D) Calling fib(4): What is the maximum amount of memory on the stack (in bytes) used for fib stack frames at any given time?

(E) Below is an incomplete snapshot of the stack during the call to fib(4). Fill in the values of the four missing intermediate words in hex:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7fffc39b72e8</td>
<td>&lt;ret addr to main&gt;</td>
</tr>
<tr>
<td>0x7fffc39b72e0</td>
<td>&lt;original rbp&gt;</td>
</tr>
<tr>
<td>0x7fffc39b72d8</td>
<td>&lt;original rbx&gt;</td>
</tr>
<tr>
<td>0x7fffc39b72d0</td>
<td></td>
</tr>
<tr>
<td>0x7fffc39b72c8</td>
<td></td>
</tr>
<tr>
<td>0x7fffc39b72c0</td>
<td></td>
</tr>
<tr>
<td>0x7fffc39b72b8</td>
<td></td>
</tr>
<tr>
<td>0x7fffc39b72b0</td>
<td>0x1</td>
</tr>
<tr>
<td>0x7fffc39b72a8</td>
<td>0x3</td>
</tr>
</tbody>
</table>
**Question 4: C & Assembly**

We are writing the *recursive* function `search`, which takes a `char` pointer and returns the *address* of the first instance in the string of a specified `char c`, or the null pointer if not found.

**Example:** `char* p = "TeST oNe"`, then `search(p, 'N')` will return the address `p+6`.

```c
char *search (char *p, char c) {
    if (!*p)
        return 0;
    else if (*p==c)
        return p;
    return search(p+1,c);
}
```

Fill in the blanks in the x86-64 code below with the correct instructions and operands. *Remember to use the proper size suffixes and correctly-sized register names!*

```
search(char*, char):
1    movzbl _____, %eax       # get *p
2    _____ _____, %al         # conditional
3    _____ .NotFound          # conditional jump
4    _____ _____, %al         # conditional
5    _____ ______            # conditional jump
6    _____ $1, _____          # argument setup
7    _____ ______            # recurse
8    ret
   .NotFound:
9    _____ $0, %eax           # return value
10   ret
   .Found:
11   movq _____, _____        # return value
12   ret
```
**Question 5: Caching**

We have 16 KiB of RAM and two options for our cache. Both are two-way set associative with 256 B blocks, LRU replacement, and write-back policies. **Cache A** is size 1 KiB and **Cache B** is size 2 KiB.

(A) Calculate the TIO address breakdown for **Cache B**:

<table>
<thead>
<tr>
<th>Tag bits</th>
<th>Index bits</th>
<th>Offset bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(B) The code snippet below accesses an integer array. Calculate the Miss Rate for **Cache A** if it starts cold.

```c
#define LEAP 4
#define ARRAY_SIZE 512
int nums[ARRAY_SIZE]; // &nums = 0x0100 (physical addr)
for (i = 0; i < ARRAY_SIZE; i+=LEAP)
  nums[i] = i*i;
```

(C) For each of the proposed (independent) changes, write MM for “higher miss rate”, NC for “no change”, or MH for “higher hit rate” to indicate the effect on **Cache A** for the code above:

- Direct-mapped  ____
- Increase block size  ____
- Double LEAP  ____
- Write-through policy  ____

(D) Assume it takes 200 ns to get a block of data from main memory. Assume **Cache A** has a hit time of 4 ns and a miss rate of 4% while **Cache B**, being larger, has a hit time of 6 ns. What is the worst miss rate **Cache B** can have in order to perform as well as **Cache A**?  ____
Question 6: Programs, processes, and processors (oh my!)

(a) Consider the following C code on the left (running on Linux), then give one possible output of running it. Assume that `printf` flushes its output immediately.

```
void oz() {
    char * name = "toto\n";
    printf("dorothy\n");
    if (fork() == 0) {
        name = "wizard\n";
        printf("scarecrow\n");
        fork();
        printf("tinman\n");
        exit(0);
        printf("witch\n");
    } else {
        printf("lion\n");
    }
    printf(name);
}
```

(b) "Pay no attention to the man behind the curtain." We have seen several different mechanisms used to create illusions or abstractions for running programs:

- A. Context switch
- B. Virtual memory
- C. Virtual method tables (vtables)
- D. Caches
- E. Timer interrupt
- F. Stack discipline
- G. None of the above, or impossible.

For each of the following, indicate which mechanism above (A-F) enables the behavior, or G if the behavior is impossible or untrue.

(i) ______ Allows operating system kernel to run to make scheduling decisions.

(ii) ______ Prevents buffer overflow exploits.

(iii) ______ Allows multiple instances of the same program to run concurrently.

(iv) ______ Lets programs use more memory than the machine has.

(v) ______ Makes recently accessed memory faster.

(vi) ______ Multiple processes appear to run concurrently on a single processor.

(vii) ______ Enables programs to run different code depending on an object's type.

(viii)______ Allows an x86-64 machine to execute code for a different ISA.
(c) Give an example of a synchronous exception, what could trigger it, and where the exception handler would return control to in the original program.

Page fault: triggered by access to virtual address not in memory, returns to the instruction that caused the fault.

Trap: used to for syscalls to do something protected by the kernel, returns to after the calling instruction.

(d) In what way does address translation (virtual memory) help make exec fast? Explain in less than 2 sentences. Hint: it may help to write down what happens during exec.

Address translation is a form of indirection, it allows us to implement fork without copying the whole process's memory, and exec without loading the whole program into memory at once.

(e) Which of the following can a running process determine, assuming it does not have access to a timer? (check all that apply)

- Its own process ID
- Size of physical memory
- Size of the virtual address space
- L1 cache associativity
- When context switches happen

(f) For each of the following, fill in what is responsible for making the decision: hardware ("HW"), operating system ("OS"), or program ("P").

(i) ______ Which physical page a virtual page is mapped to.

(ii) ______ Which cache line is evicted for a conflict in a set-associative cache.

(iii) ______ Which page is evicted from physical memory during a page fault.

(iv) ______ Translation from virtual address to physical address.

(v) ______ Whether data is stored in the stack or the heap.

(vi) ______ Data layout optimized for spatial locality
Question 7: Virtual Memory

Our system has the following setup:

- 24-bit virtual addresses and 512 KiB of RAM with 4 KiB pages
- A 4-entry TLB that is fully associative with LRU replacement
- A page table entry contains a valid bit and protection bits for read (R), write (W), execute (X)

(A) Compute the following values: [2 pt]

- Page offset width ________
- PPN width ________

- Entries in a page table ________
- TLB width ________

(B) Briefly explain why we make the page size so much larger than a cache block size.

(C) Fill in the following blanks with “A” for always, “S” for sometimes, and “N” for never if the following get updated during a page fault.

- Page table ________
- Swap space ________
- TLB ________
- Cache ________

(D) The TLB is in the state shown when the following code is executed. Which iteration (value of i) will cause the protection fault (segfault)? Assume sum is stored in a register.

Recall: the hex representations for TLBT/PPN are padded as necessary.

```c
long *p = 0x7F0000, sum = 0;
for (int i = 0; i < 1; i++) {
    if (i%2)
        *p = 0;
    else
        sum += *p;
    p++;
}
```

<table>
<thead>
<tr>
<th>TLBT</th>
<th>PPN</th>
<th>Valid</th>
<th>R</th>
<th>W</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7F0</td>
<td>0x31</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0x7F2</td>
<td>0x15</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0x004</td>
<td>0x1D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0x7F1</td>
<td>0x2D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

i = ________