Exam 2

601.229 Computer Systems Fundamentals

November 3, 2025

Complete all questions.

Time: 50 minutes.

I affirm that I have completed this exam without unauthorized assistance from any person, materials, or device.

Signed: _____

Print name: _____

Date: _____

Reference

Powers of 2 ($y = 2^x$):

x	0	1	2	3	4	5	6	7	8	9	10	11	12
y	1	2	4	8	16	32	64	128	256	512	1,024	2,048	4,096

x	13	14	15	16
y	8,192	16,384	32,768	65,536

Note that in all questions concerning C:

- `uint8_t` is an 8-bit unsigned integer type
- `uint16_t` is a 16-bit unsigned integer type
- `uint32_t` is a 32-bit unsigned integer type
- `int8_t` is an 8-bit signed two's complement integer type
- `int16_t` is a 16-bit signed two's complement integer type
- `int32_t` is a 32-bit signed two's complement integer type

x86-64 registers:

Callee-saved: `%rbx`, `%rbp`, `%r12`, `%r13`, `%r14`, `%r15`

Caller-saved: `%r10`, `%r11`

Return value: `%rax`

Arguments: `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, `%r9`

Note that argument registers and return value register are effectively caller-saved.

Registers and sub-registers:

Register	Low 32 bits	Low 16 bits	Low 8 bits
`%rax`	`%eax`	`%ax`	`%al`
`%rbx`	`%ebx`	`%bx`	`%bl`
`%rcx`	`%ecx`	`%cx`	`%cl`
`%rdx`	`%edx`	`%dx`	`%dl`
`%rbp`	`%ebp`	`%bp`	`%bpl`
`%rsi`	`%esi`	`%si`	`%sil`
`%rdi`	`%edi`	`%di`	`%dil`
`%r8`	`%r8d`	`%r8w`	`%r8b`
`%r9`	`%r9d`	`%r9w`	`%r9b`
`%r10`	`%r10d`	`%r10w`	`%r10b`
`%r11`	`%r11d`	`%r11w`	`%r11b`
`%r12`	`%r12d`	`%r12w`	`%r12b`
`%r13`	`%r13d`	`%r13w`	`%r13b`
`%r14`	`%r14d`	`%r14w`	`%r14b`
`%r15`	`%r15d`	`%r15w`	`%r15b`

Stack alignment: `%rsp` must contain an address that is a multiple of 16 when any `call` instruction is executed.

Operand size suffixes: **b** = 1 byte, **w** = 2 bytes, **l** = 4 bytes, **q** = 8 bytes (Examples: `movb`, `movw`, `movl`, `movq`)

Question 1. [10 points] Consider the `combine` function, intended to combine strings together into a single string (unit test shown on right):

```
void combine(const char *s[],
             int n, char *d) {
  *d = '\0';
  for (int i = 0; i < n; ++i)
    strcat(d, s[i]);
}
```

```
const char *str[] = {
  "CSF", "is", "FUN!",
};
char out[100];
combine(str, 3, out);
assert(0 == strcmp(out, "CSFisFUN!"));
```

(a) What is the most significant inefficiency in the implementation of the `combine` function?

(b) Briefly explain how to fix the inefficiency you identified in (a).

Question 2. [10 points] A memory cache has the following characteristics:

- Its data capacity is 1,024 KB (2^{20}) bytes
- Its block size is 32 (2^5) bytes
- It has 4,096 (2^{12}) sets

(a) What is the associativity factor of this cache? (I.e., how many blocks can be stored in each set?) Explain briefly.

(b) Assume that the system has 32-bit addresses. Sketch the address format for the cache, indicating which exact bits are the offset, index, and tag.

Question 3. [10 points] Assume that a superscalar CPU has two independent fully-pipelined integer multiplication units, each of which can complete an integer multiplication in 3 cycles.

Consider the following computation:

```
e = a * b
f = a * c
g = a * d
h = e * f
i = g * f
j = h * i
```

Assume that all variables (`a` through `j`) represent CPU registers. How many cycles are required to compute the value of to be stored in `j`? Explain briefly. Recommendation: show the operations scheduled in each pipeline stage on each clock cycle.

Question 4. [15 points] Consider the following code:

```
int arr[32];
for (int i = 0; i < 32; ++i)
  arr[i] = i;
```

Assume the following:

- There is a write-back memory cache with 32 byte blocks
- The address of `arr[0]` is an exact multiple of 32
- `sizeof(int)` = 4
- Before the above code executes, the cache is completely empty
- The variable `i` is a CPU register
- All data transfers between the cache and main memory (DRAM) are in units of 4 bytes

Also, assume that there is a separate instruction cache and that the instructions of the code above are resident in the instruction cache, so no cache misses due to instruction fetches occur.

(a) For which values of `i` is it likely that cache misses will occur? Explain briefly.

(b) How many 4 byte reads from main memory (DRAM) occur when the code above is executed? Explain briefly.

(c) How many 4 byte writes to main memory (DRAM) are likely to occur when the code above is executed? Explain briefly.

Question 5. [25 points] In a cache, addresses are 8 bits, blocks are 16 bytes, there are 4 sets, and the cache is 2-way set associative.

Complete the following table. For each address in the *Request* column, indicate the tags of cached blocks after handling the request. Addresses are specified in base-2. Assume each request is a load, and that they execute sequentially (top row is the first in the sequence.) All slots are initially empty. When a block is evicted, select the victim using the LRU (Least Recently Used) replacement policy.

	Set 0		Set 1		Set 2		Set 3	
Request	Slot 0	Slot 1	Slot 0	Slot 1	Slot 0	Slot 1	Slot 0	Slot 1
	empty	empty	empty	empty	empty	empty	empty	empty
00110110								
10010011								
11010111								
01000001								
10001011								
10010110								
11001011								
10111001								
10000000								
01011100								
00010000								
11001110								

Question 6. [10 points] Consider the following x86-64 instruction:

```
movl (%rax), %edi
```

State a reason why a page fault could occur when the CPU attempts to execute this instruction.

Question 7. [10 points] Assume that the following x86-64 instruction occurs in a program:

```
movl %esi, %r10d
```

Assume that a branch in the program jumps to this instruction. Is it possible that a page fault could occur when the CPU attempts to fetch and execute the above instruction? Answer "Yes" or "No," and then either state a reason a page fault could occur, or explain why a page fault definitely cannot occur.

Question 8. [10 points] Consider the following program:

```
 1:    #include <stdio.h>
 2:
 3:    int answer = 42;
 4:
 5:    int main(void) {
 6:      printf("The answer is ");
 7:      int x;
 8:      x = answer;
 9:      printf("%d\n", x);
10:      return 0;
11:    }
```

Assume that the local variable `x` in the `main` function is allocated as a CPU register (e.g., `%esi`.)

(a) Explain why a page fault is likely to occur when line 8 is executed.

(b) Explain why in handling the page fault you explained in (a), the OS kernel's page fault handler will read data from a file. Explain which file the page fault handler will read from, and what data will be read from that file.

[Extra page for answers and/or scratch work.]

[Extra page for answers and/or scratch work.]