CS33: Introduction to Computer Organization
Fall 2020 Final

Name:	Karim Saraipour

Rules/Instructions:

- All of your answers go into red tables like this:

What's the answer	Your answer here

- When complete, save the exam as a PDF. (if there is a technical problem, just save as docx)
- Turn the exam in on CCLE, before 10:00pm PST (normal time), 11:30am PST (makeup time). The exam is designed for 3 hours, but we are giving you an extra 30 minutes in case you have any technical difficulties.
- This is an open notes exam. By the honor system, you may not discuss exam questions/solutions/experiences/thoughts/etc. with any person for 24 hours after the exam start time.
- Please do not alter which page each question is on. **Please do your best to keep the question boxes approximately the same size.** If choose to scan the exam, try to line it up nicely. This is to help TAs suffer less while grading.

Notes:

- There are 75 points total, but the exam is graded out of 60. (ie. the exam is pre-curved so that there are 10 extra credit points possible)
- You may ask for questions on Piazza (private posts only). Clarifications will be posted to this google drive link: so it may be a good idea to check this before the exam is over.
- If the architecture of the machine is not specified, assume that the question is being asked in the context of a 64-bit little endian x86 machine.

Finally, please follow the university guidelines in reporting academic misconduct.

You may begin once you have read the rules above.

Question 1. Linking (4 pts)

Suppose src1.c and src2.c are compiled and linked separately. Determine if the following combinations of source files would cause errors, and if not, what would get printed.

Feel free to solve this problem by compiling the source files and linking them together. If an answer is undefined, simply write “undefined” in the result box.

src1.c	**src2.c**	**Result?** **(“compile error”, “linker error” or describe output)**

`int i=1;`
`int main() {`
`  printf("%d\n",i);`
`}`
	`int i=2;`
	`int func() {`
	`  i=3;`
	`}`
		Linker Error
		2 strong symbols

`int i=1;`
`int main() {`
`  printf("%d\n",i);`
`}`
	`int i;`
	`int func() {`
	`  i=3;`
	`}`
		1

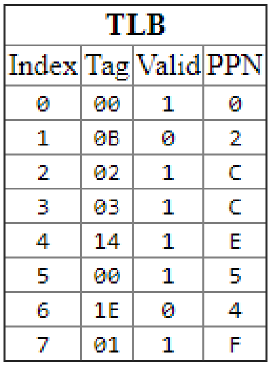
`int i;`
`int main() {`
`  printf("%d\n",i);`
`}`
	`int i;`
	`int func() {`
	`  i=3;`
	`}`
		0

`int i;`
`int main() {`
`  printf("%d\n",i);`
`}`
	`int i=2;`
	`int func() {`
	`  int i=3;`
	`}`
		2

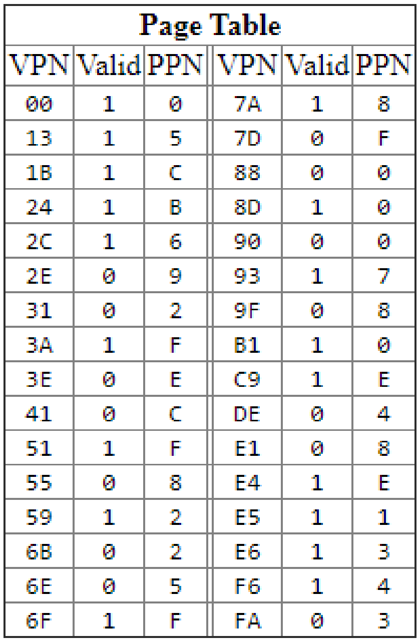
Question 2. Virtual Memory (6 pts)

Given the following details about the memory system and the states of the TLB and Caches, fill out the following information for a request for the virtual address: **0x597A0**

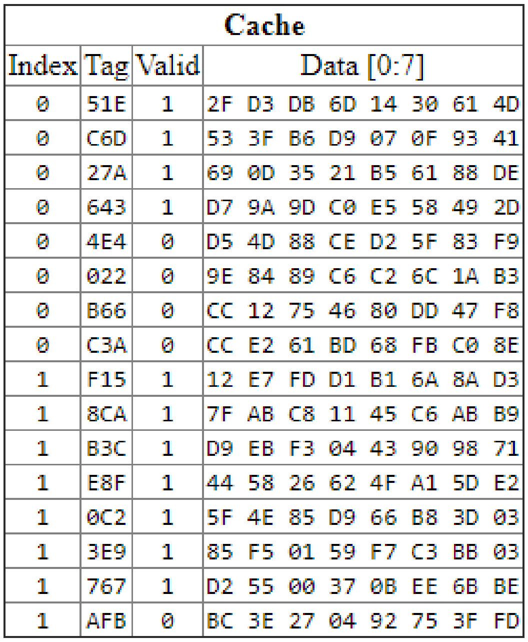
- Main memory is 64 KB (2^16 bytes), byte-addressable with a 16 bit physical address.
- Virtual address is 20 bits long
- A page of memory is 4 KB (2^12 bytes)
- Direct mapped TLB with 8 entries
- 8-way set associative cache with 16 lines and cache block of 8 bytes



TLB			
Index	Tag	Valid	PPN
0	00	1	0
1	0B	0	2
2	02	1	C
3	03	1	C
4	14	1	E
5	00	1	5
6	1E	0	4
7	01	1	F



Page Table					
VPN	Valid	PPN	VPN	Valid	PPN
00	1	0	7A	1	8
13	1	5	7D	0	F
1B	1	C	88	0	0
24	1	B	8D	1	0
2C	1	6	90	0	0
2E	0	9	93	1	7
31	0	2	9F	0	8
3A	1	F	B1	1	0
3E	0	E	C9	1	E
41	0	C	DE	0	4
51	1	F	E1	0	8
55	0	8	E4	1	E
59	1	2	E5	1	1
6B	0	2	E6	1	3
6E	0	5	F6	1	4
6F	1	F	FA	0	3



Cache			
Index	Tag	Valid	Data [0:7]
0	51E	1	2F D3 DB 6D 14 30 61 4D
0	C6D	1	53 3F B6 D9 07 0F 93 41
0	27A	1	69 0D 35 21 B5 61 88 DE
0	643	1	D7 9A 9D C0 E5 58 49 2D
0	4E4	0	D5 4D 88 CE D2 5F 83 F9
0	022	0	9E 84 89 C6 C2 6C 1A B3
0	B66	0	CC 12 75 46 80 DD 47 F8
0	C3A	0	CC E2 61 BD 68 FB C0 8E
1	F15	1	12 E7 FD D1 B1 6A 8A D3
1	8CA	1	7F AB C8 11 45 C6 AB B9
1	B3C	1	D9 EB F3 04 43 90 98 71
1	E8F	1	44 58 26 62 4F A1 5D E2
1	0C2	1	5F 4E 85 D9 66 B8 3D 03
1	3E9	1	85 F5 01 59 F7 C3 BB 03
1	767	1	D2 55 00 37 0B EE 6B BE
1	AFB	0	BC 3E 27 04 92 75 3F FD

VPN	01011001 ➔ 0x59
VPO/PPO	011110100000 ➔ 0x7A0
TLB Index	001 ➔ 0x1
TLB Tag	1011 ➔ 0x0B
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	0x2
Physical Address	0x27A0 ➔ 10011110100000
Cache Offset	0x0
Cache Index	0x0
Cache Tag	0x27A

Data	69

Question 3. Performance Analysis (7 pts)

Supposed the following code is compiled without aggressive compiler optimizations (ie. -Og like we have been using in this class so far).

```
float sum, c[N], a[N], b[N];

int func(int j) {
  for(j = 0; j < N; j+=2) {
    sum += a[j+0] * b[j+0];
    sum += a[j+1] * b[j+1];
  }
}
```

Our processor has the following characteristics:

Func Unit	Latency	Cycles/Issue	Func Unit Count in Processor
Float Multiplies	3	2	4
Float Adds	1	1	2
Loads	4	1	4
All other instructions	1	1	Infinite

1. What program optimization has been manually applied to this code? (1pt)	**Loop Unrolling**
2. Assuming the above hardware characteristics, what is the latency bound on CPE? (please consider one "element" to be one multiply and accumulate) (2pt)	**4**
3. Assuming the above hardware characteristics, what is the throughput bound on CPE? (only consider floating multiplies, adds and loads) (2pt)	**1**
4. In terms of N, roughly how many cycles does this program take on an out-of-order processor with no other bottlenecks other than instruction latency and throughput? (1pt)	**1N ➔ 1 perfect out-order**
5. In as few words as possible, what optimization can you perform to improve the performance? (1pt)	**different accumalators Such as sum1, sum2, … , sumN**

Question 4. Cache Design (6pts)

Suppose you are in charge of designing the next generation of Unintel CPUs, and you are working on the cache design. The current proposal is to use a single direct mapped cache, with a total capacity of 1KB, and a block size of 32B. Your job is to decrease the *cache miss rate* as much as possible.

For each program, choose the optimization that *minimizes the cache miss rate* for that program:

Code	Possible Optimizations	Best Optimization (list either A, B,or C)

`#define N 65536`
`int a[N], b[N], c[N];`
`...`
`for(int i = 0; i < N; ++i) {`
`  c[i] = a[i] * b[i];`
`}`
	A. Increase Cache Capacity to 32KB
	B. Increase Cache Block size to 128B
	C. Increase the Associativity to 4-Way
		B

`#define N 256`
`int a[N], b[N][N], c[N];`
`...`
`for(int j = 0; j < N; ++j) {`
`  for(int i = 0; i < N; ++i)`
`{`
`    c[i] += a[i] * b[j][i];`
`  }`
`}`
	A. Increase Cache Capacity to 32KB
	B. Increase Cache Block size to 128B
	C. Increase the Associativity to 4-Way
		A

`#define N 256`
`int a[N], b[N], c[N];`
`...`
`for(int i = 0; i < N; ++i) {`
`  c[i] = a[i] * b[i];`
`}`
	A. Increase Cache Capacity to 2KB
	B. Increase Cache Block size to 64B
	C. Increase the Associativity to 4-Way
		C

Question 5. Forking Around (7pts)

Here’s a really forked up program:

```
int main () {
  if (fork() == 0) {
    if (fork() == 0) {
      printf("O");
    }
    else {
      pid_t pid; int status;
      if ((pid = wait(&status)) > 0) {
        printf("D");
      }
    }
  }
  else {
    printf("E");
    exit(0);
  }
  printf("R");
  return 0;
}
```

1. List all possible program outputs (just put a space between answers if there is more than one, leave empty if nothing can be printed, **5pts**):

EORDR E EOR

2. Will there be any zombies after this program runs? (yes or no, **2pts**)

No, the original parent dies

Question 6. Multiple Choice (16 pts)

For the following multiple choice questions, **select all that apply**. Ie. if none of the answers are correct, simply leave the question blank. (2pts each, no partial credit)

1. What is the difference (or differences) between a TLB and on-chip cache?
    a. The TLB is direct-mapped, while caches are set associative.
    b. The TLB is indexed by the virtual address, while caches are indexed by the physical address.
    c. The TLB stores virtual-to-physical address translations, while caches store data.
    d. The TLB can be slow, but caches need to be fast.
    e. The TLB stores instructions, while caches store data.

2. Say we have two mutexes, implemented with binary semaphores, and two threads which access them. Which of the following can cause *deadlock*:
    a. The threads lock the mutexes in the same order.
    b. The threads lock the mutexes in a different order.
    c. The threads unlock the mutexes in the same order.
    d. The threads unlock the mutexes in a different order.

3. Which stack protection techniques are vulnerable to return-oriented programming?
    a. Stack Canaries
    b. Address space layout randomization
    c. Limiting Executable Code Regions

4. In malloclab, several students implemented an optimization where small blocks would be allocated at the beginning of a free block, and large blocks would be allocated at the end of a free block. In what way was this useful on some traces?
    a. It increases the memory utilization due to less internal fragmentation.
    b. It increases the memory utilization due to more coalescing opportunities.
    c. It increases the throughput due to smaller free lists.
    d. It increases the throughput due to better temporal locality in caches.

5. After a fork(), to access which datastructures should the resulting two processes synchronize?
    a. Heap
    b. Stack
    c. Registers
    d. Globals
    e. Program Code

6. What’s the purpose of the calling convention?
    a. Enables virtual memory.
    b. Helps enable separate compilation.
    c. Improves external fragmentation.
    d. Lowers the cost of creating threads.

7. Which of the following statements about C datatypes for x64_64 is true?
    a. A float can represent any number a double can represent.
    b. A double can represent any number an int can represent.
    c. A char can represent any number a short can represent.
    d. A long can represent any number a float can represent.

8. Which of the following do not necessarily involve exceptional control flow:
    a. Context switch
    b. Killing a process
    c. Page Fault
    d. Function Call
    e. Segmentation Fault
    f. Timer Interrupt
    g. Data Cache Miss
    h. TLB Miss (in x86_64)

Multiple Choice Question Number	Write your answers here: (eg: a,b,d)
1.	B, C
2.	B
3.	B, C
4.	B, C
5.	A, B, D, E
6.	-- (blank)
7.	B
8.	D, H, G

Question 7. Jumbled Metaphors (5 pts)

As you reach the later stages of the exam, you may notice your mind becoming a jumbled pile of mixed metaphors. Sort yourself out by finding 10 words related to this course in the mess of letters below.

Rules: Words must be contiguous, and they may be forwards or backwards and either horizontal or diagonal. One example is given "thread", but don't count this one! Don't list extra words, or we won't grade the question. :)

E	L	B	A	T	R	H	C	N	A	R	B
H	A	L	I	N	K	I	L	L	P	Y	E
E	E	X	I	T	D	A	E	R	H	T	R
G	K	E	R	N	E	L	P	A	L	C	L
A	X	K	A	A	L	I	B	R	A	R	Y
P	C	N	A	P	A	E	N	T	B	M	E
E	C	T	I	E	T	Y	B	H	U	R	I
F	L	A	L	E	L	P	F	L	S	I	B
A	K	B	C	A	F	L	O	A	T	C	M
U	L	H	H	H	A	E	I	F	A	D	O
L	K	P	I	P	E	L	I	N	E	X	Z
T	C	E	I	A	B	A	A	M	L	C	T
N	O	I	I	L	N	R	T	E	L	F	Y
L	L	N	U	E	Y	E	C	Y	C	L	E

Please List 10 Other Words Here:

ZOMBIE	FAULT
FLOAT	KERNEL
KILL	PIPELINE
LIBRARY	ELF
EXIT	CYCLE

Question 8. Soulmate Simulator (9pts)

Suppose we want to create a soulmate simulator, where we randomly compare *exactly* two people to see if they should be soulmates. For fun, let's represent each person as a **thread**. We will call the **"meet"** function with many threads, and they will "**mingle**" to see if they are soulmates. *The **only** important thing for this problem, is that there should only be two threads in the mingle() function at one time.*

Wrong Code

```
int number_of_people_meeting=0;
void *meet(void *personID)
{
    //This loop tries to ensure
    // there's only one thread waiting
    while(number_of_people_meeting >1) {
      // do nothing
    }
    number_of_people_meeting++;
    //This is where we check for soulmates
    //between threads.
    mingle();
    //We want at most 2 threads here!
    number_of_people_meeting--;

  return NULL;
}

void main()
{


  pthread_t t[N];
  for (i=0; i < N; i++) // make threads meet
    pthread_create(&t[i], 0,
                  meet, (void *)i);
}
```

Your Code

```

sem_t mutex_q8;

void *meet(void *personID)
{


    //This is where we check for soulmates
    //between threads.
    sem_wait(&mutex_q8);
    mingle();
    sem_post(&mutex_q8);
    //We want at most 2 threads here!


  return NULL;
}

void main()
{
  sem_init(&mutex_q8, 0, 2);

  pthread_t t[N];
  for (i=0; i < N; i++) // make threads meet
    pthread_create(&t[i], 0,
                  meet, (void *)i);
}
```

1. Examine the "Wrong Code" Above. In as few words as possible, why can't it guarantee that only 2 threads are in the mingle function at the same time? (3pts)

   number_of_people_meeting (a global) could cause data races, thus multiple threads can access it at once, therefore it theoretically possible more than 2 people to mingle()

2. Using a counting semaphore, complete the code on the right to guarantee that only 2 threads are calling mingle at the same time. (6pts)

Question 9: Inopportune Overlap (7 pts)

Think about the following two cases of “overlap” between aspects of the virtual memory system.

1. **Overlap of Variables -> Cache Lines:** It’s possible to imagine that a single primitive variable (eg. int,float,char) could “straddle” two different cache lines. (ie. if it starts at the end of one cache line, and is long enough to proceed into the next cache line).
   This complicates the hardware, because a single access to a variable from the CPU has to combine the information from multiple cache lines (yuk!). However, unless you are messing about with pointer arithmetic, this does not happen in C.
   In as few words as possible, what aspect of the C language prevents a single primitive variable (ie. int, float, char, pointer, etc.) from being mapped to multiple cache lines? (2pts)

None of the primitive data types are larger than 8 bytes. Furthermore, caches are based off addresses; therefore, you could ensure the beginning and ending addresses are multiples of 8. Also padding and alignment helps

2. **Overlap of Cache Lines -> Pages:** It’s possible to imagine that one cache line could “straddle” two different virtual memory pages.

   2.1. In as few words as possible, list one reason why, if this was possible, it would complicate the hardware or software for address translation. (2pts)

Performance will plummet. Multilevel page tables (or accessing page tables in general) would be extremely expensive if you needed to access two pages for 1 line. Also, page faults might become more common

   2.2 However, cache line’s don't straddle virtual memory pages in real systems, why? (2pts)

Caches and pages are always aligned to their respective sizes. The system has pages that are much larger than the size of caches. Also, the (page size/cache size) should be a whole number(a power of 2).

Question 10. Another Phase? (8pts)

OMG another bomblab phase? Using instructions we never learned in class? 🤦

```
00000000005858a8 <string_cmp>:  # compares two strings, returns zero if equal
#... asm omitted…                                          (same as strcmp)
00000000005858b6 <string_cpy>:  # copies string (same as strcpy)
#... asm omitted...
00000000005858c4 <phase_defused>: #prints phase_defused message
#... asm omitted...
00000000005858de <explode_bomb>: #prints explode_bomb message
#... asm omitted...

00000000005858f8 <phase_11>:
  5858f8:       48 83 ec 18             sub    $0x18,%rsp
  5858fc:       48 89 fe                mov    %rdi,%rsi
  5858ff:       48 8d 7c 24 08          lea    0x8(%rsp),%rdi
  585904:       e8 ad ff ff ff          callq  5858b6 <string_cpy>
  585909:       f3 0f 10 44 24 08       movss  0x8(%rsp),%xmm0
  58590f:       0f 2e 05 12 17 09 00    ucomiss 0x91712(%rip),%xmm0
                                                #Addr: 0x617028
  585916:       7a 16                   jp     58592e <phase_11+0x36>
  585918:       75 14                   jne    58592e <phase_11+0x36>
  58591a:       b8 00 00 00 00          mov    $0x0,%eax
  58591f:       e8 a0 ff ff ff          callq  5858c4 <phase_defused>
  585924:       bf 00 00 00 00          mov    $0x0,%edi
  585929:       e8 22 df 00 00          callq  593850 <exit>
  58592e:       b8 00 00 00 00          mov    $0x0,%eax
  585933:       e8 a6 ff ff ff          callq  5858de <explode_bomb>
  585938:       48 83 c4 18             add    $0x18,%rsp
  58593c:       c3                      retq

000000000058593d <main>:
... asm omitted...

000000000058595a <secret_phase>:
```

Some helpful things:

- (gdb) print *(float*)0x617028
  $3 = 115783711918063280247642521 6.000000
- Ucomiss compares two floating point values, it works similarly to cmp.
- string_cmp is the same as the standard strcmp, and string_cpy is the same as strcpy, but I'm using my own versions b/c it makes the code easier to read.

1. What string will diffuse this phase? (4pts)	000000000000000000000000 00000000005858c4

2. What string will enter the secret phase? (4pts)	000000000000000000000000000000000058595a