CS33: Introduction to Computer Organization
Fall 2020 Final

Name:	
UID:	

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 15 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.

Note: Question4 eliminated due to ambiguity.

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;`
	`void func() {`
	`  i=3;`
	`}`
		linker error

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

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

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

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 **in hex** for a one byte request for the virtual address: **0xD2A7E7**

- Main memory is 1 MB (2^20 bytes), byte-addressable with a 20 bit physical address.
- Virtual address is 24 bits long
- A page of memory is 64 KB (2^16 bytes)
- 8-way set associative TLB with 16 entries
- 4-way set associative cache with 16 lines and cache block of 8 bytes

TLB			
Index	Tag	Valid	PPN
0	69	1	2
0	48	0	A
0	04	0	8
0	06	1	C
0	05	1	A
0	53	0	C
0	38	1	8
0	24	1	9
1	2E	1	C
1	66	0	6
1	01	1	3
1	07	1	F
1	02	1	5
1	04	1	9
1	00	1	0
1	1C	1	E

Page Table					
VPN	Valid	PPN	VPN	Valid	PPN
01	0	8	A1	0	8
07	1	3	A7	1	C
0C	1	3	A9	0	B
24	1	F	B5	0	2
3E	0	A	B9	0	8
49	0	D	BB	0	F
4F	1	1	C6	1	A
56	1	1	CD	1	6
5D	1	C	CF	1	1
66	0	6	D2	1	2
6A	1	5	D7	0	0
72	1	8	E4	0	C
7B	0	8	EA	1	9
90	1	A	EF	0	D
94	0	5	F5	1	C
9B	0	A	F7	1	C

Cache			
Index	Tag	Valid	Data [0:7]
0	0F95	1	44 DC 07 94 BB 1C EC 3F
0	5177	1	88 76 F8 3E 21 93 9F 0D
0	153F	0	74 82 2C 15 71 B9 8B 12
0	1B1E	0	C8 5D 07 8C A2 43 08 6D
1	6406	1	6C 65 41 55 EE F5 9B D3
1	08AE	1	2B F4 C1 1B F8 77 61 94
1	52BD	0	5F E3 B5 13 E7 60 DA B7
1	6676	0	58 5D 18 DE 2F 2C 88 1B
2	2CC1	1	CE 75 DE 35 8A E5 91 23
2	46A0	0	81 69 99 3D F1 8D 52 92
2	1101	0	A6 0A CB 06 E3 17 EC 75
2	7626	0	00 E9 AF C7 91 15 13 1A
3	4E9D	1	8B AD 28 00 19 42 C5 CE
3	3593	1	7A 5F 5E 61 7F A2 49 3A
3	6041	1	36 92 15 65 69 AE 25 D8
3	5227	0	94 7E C7 26 14 4A 82 E9

VPN	0xD2 1101 0010
VPO/PPO	0xA7E7
TLB Index	0
TLB Tag	0x69 110 1001
TLB Hit? (Y/N)	1
Page Fault? (Y/N)	N
PPN	0x2 0010
Physical Address	0x2A7E7 0010 1010 0111 1110 0111
Cache Offset	0x7 111
Cache Index	0x0 00
Cache Tag	0x153F 001 0101 0011 1111
Data	Cache miss

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, 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)	**1**
3. Assuming the above hardware characteristics, what is the throughput bound on CPE? (only consider floating multiplies, adds and loads) (2pt)	**1/2**
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)	**N**
5. In as few words as possible, what optimization can you perform to improve the performance? (1pt)	**Separate Accumulators** **Or** **Change associativity**

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 OERDR OREDR ORDER ORDRE

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

no

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 do not prevent return-oriented programming attacks?
    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 x84_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.	Either “” or “e”
6.	b
7.	b
8.	d,g,h

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, vertical 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	ELF	LOCK	CACHE
LINK	CYCLE	CANARY	KILL
BRANCH	PAGE	BUS	FLOAT
BYTE	LEAK	LIBRARY	KERNEL
PIPELINE	FAULT	EXIT	TABLE

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 sem;

void *meet(void *personID)
{


    sem_wait(&sem);

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

    sem_post(&sem);

  return NULL;
}

void main()
{
  sem_init(&sem, 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)

   Race on shared variable “number_of_people_meeting”. They can both read the same number.

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)

Alignment

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. (3pts)

Have to do two address translations to access one cache line.

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

Size of both is a power of 2, so they divide evenly.

Question 10. Another Phase? (8pts)

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

```
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 = 115783711918063280247642 5216.000000
- Ucomiss compares two floating point values, it works similarly to cmp. You should be able to ignore jp.
- string_cpy is the same as strcpy; but these versions make the code easier to read.

1. What string will diffuse this phase? (4pts)	"cool"
2. What string will enter the secret phase? (4pts)	"0000000000000000ZYX"