CS 449 Summer 2019 – Final Exam

Please read through the entire examination first!

- You have **110 minutes** to complete this exam. Don't spend too much time on any one problem!
- The last page is a reference sheet. Feel free to detach it from the rest of the exam.
- The exam is CLOSED book and CLOSED notes (no summary sheets, no calculators, no mobile phones).

There are **6 problems** for a total of **67 points**. The point value of each problem is indicated in the table below. Write your answer **neatly** in the spaces provided.

Please do not ask or provide anything to anyone else in the class during the exam. Make sure to ask clarification questions early so that both you and the others may benefit as much as possible from the answers.

Good Luck!

Last Name: ___________________________

First Name: ___________________________

Pitt Student ID: ___________________________

Problem	Topic	Max Score
1	Caches	15
2	Processes	10
3	Virtual Memory	12
4	Memory Allocation	11
5	Representation	10
6	Pointers & Memory	9
TOTAL		**67**

1. Caches (15 points total)

You are using a byte-addressed machine with 64 KiB of Physical address space. You have a 2-way associative L1 data cache of total size 256 bytes with a cache block size of 16 bytes. It uses LRU replacement and write-allocate and write-back policies.

a) [2 pt] Give the number of bits needed for each of these:

Cache Block Offset:____________ Cache Tag: ______________

b) [1 pt] How many **sets** will the cache have? ______________

c) [4 pts] Assume `i` and `j` are stored in registers, and that the array `x` starts at address 0x0. Give the <u>miss</u> rate (as a fraction or a %) for the following two loops, assuming that the cache starts out empty.

```
#define LEAP 2
#define SIZE 128
int x[SIZE];
... // Assume x has been initialized to contain values.
... // Assume the cache starts empty at this point.
for (int i = 0; i < SIZE; i += LEAP) {       // Loop 1
  x[i] = x[i] + i * i;
}
for (int j = 1; j < SIZE; j += LEAP) {       // Loop 2
  x[j] = x[j] + j * 2;
}
```

<u>Miss</u> Rate for Loop 1:_____________ **<u>Miss</u>** Rate for Loop 2: _____________

d) [8 pts] For each of the changes proposed below, indicate how it would affect the **<u>miss</u> rate** of each loop above in part c) *assuming that all other factors remained the same* as they were in the original problem. Mark X in one of: "increase", "no change", or "decrease" for each loop.

Change associativity from	Loop 1:	( ) increase / ( ) no change / ( ) decrease
2-way to direct mapped:	Loop 2:	( ) increase / ( ) no change / ( ) decrease
Change `LEAP` from	Loop 1:	( ) increase / ( ) no change / ( ) decrease
2 to 4:	Loop 2:	( ) increase / ( ) no change / ( ) decrease
Change cache size from	Loop 1:	( ) increase / ( ) no change / ( ) decrease
256 bytes to 512 bytes:	Loop 2:	( ) increase / ( ) no change / ( ) decrease
Change block size from	Loop 1:	( ) increase / ( ) no change / ( ) decrease
16 bytes to 32 bytes:	Loop 2:	( ) increase / ( ) no change / ( ) decrease

2. Processes (10 points total)
The following function prints out numbers.

```
void sunny(void) {
  int x = 4;
  if (fork()) {
    x += 6;
  } else {
    x += 1;
  }
  printf("%d ", x);
  if (fork()) {
    x += 1;
  } else {
    x -= 2;
  }
  printf("%d ", x);
  fork();
  exit(0);
}
```

a. [3 pts] List 3 possible outputs of the code above:

(1) ______________________________________

(2) ______________________________________

(3) ______________________________________

b. [2 pts] What is the total number of processes created (including the original process that called **`sunny`**) by this function?

c. [1 pt] Is it possible for the numbers to appear in descending order (highest value to lowest value) in the output (Mark X)?

( ) YES / ( ) NO

d. [2 pts] The function call **`fork()`** returns something. Describe, in general, what **`fork()`** returns?

e. [2 pts] When context-switching from a process A to a process B, which elements of process B's state must be restored before process B can begin executing (Mark X):

- Contents of registers ( ) YES / ( ) NO
- Contents of L1 cache ( ) YES / ( ) NO
- Contents of PTBR ( ) YES / ( ) NO
- Contents of TLB ( ) YES / ( ) NO

3. Virtual Memory (12 points)

Assume we have a virtual memory detailed as follows:

- 8 KiB Virtual Address Space,
- 2 KiB Physical Address Space,
- a TLB with 16 entries that is 4-way set associative with LRU replacement
- 64 B page size

a) [5 pts] How many bits will be used for:

Page offset? ____________

Virtual Page Number (VPN)?___________ Physical Page Number (PPN)? __________

TLB index?_____________________ TLB tag? ___________________

b) [1 pt] How many TOTAL entries are in this page table?
(It is fine to leave your answer as powers of 2).

____

3. (cont.) The current contents of the TLB and (partial) Page Table are shown below:

TLB

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	-	0	07	00	1	06	-	0	1F	03	1
1	00	0B	1	0A	-	0	0C	03	1	01	0F	1
2	07	-	0	0C	02	1	0F	01	1	0B	-	0
3	01	1C	1	0C	01	1	04	01	0	1A	01	1

Page Table (only first 16 of the PTEs are shown)

VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid	VPN	PPN	Valid
00	03	1	04	-	0	08	07	1	0C	0F	1
01	0B	1	05	0F	1	09	-	0	0D	-	0
02	03	1	06	-	0	0A	01	1	0E	06	1
03	03	1	07	1C	1	0B	08	1	0F	0A	1

c) [6 pts] Determine the physical address, TLB miss or hit, and whether there is a page fault for the following virtual address accesses (write "Y" or "N" for yes or no, respectively, in the TLB Miss? And Page Fault? columns). If you can't determine the PPN and/or physical address and/or TLB miss and/or Page Fault, simply write ND (for non-determinable) in the appropriate entry in the table.

Virtual Address	VPN (give bits)	TLBT (give bits)	TLBI (give bits)	PPN (give bits)	Physical Address (give bits)	TLB Miss?	Page Fault?
0x1306							
0x0C62							
0x02C3							

4. Memory Allocation (9 points total)

```
1     #include <stdlib.h>
2     float pi = 3.14;
3
4     int main(int argc, char *argv[]) {
5       int year = 2019;
6       int* happy = malloc(sizeof(int*));
7       happy++;
8       free(happy);
9     return 0;
10    }
```

A) [3 pts] Consider the C code shown above. Assume that the `malloc` call succeeds and `happy` and `year` are stored in memory (not in a register). Fill in the following blanks with "<" or ">" or "UNKNOWN" to compare the *values* returned by the following expressions just before `return 0`.

&year____________**&main**

happy____________**&happy**

&pi ____________**happy**

B) [4 pts] The code above has two memory-related errors. Use the line numbers in the code to describe what the errors are and where they occur.

Error #1:

Error #2:

C) [2 pts] (Not related to code at top of page) Give one advantage that next fit placement policy has over a first fit placement policy in an implicit free list implementation.

5. Representation (10 points)

a) [4 pts] Consider the **`signed char x = 0b 1000 0110`**

i. What is the value of **`x`**? You may answer as the sum of powers of 2.

ii. Evaluate each of the following expressions:

`x & (x >> 4)`	**`~x`**	**`x ^ 0xC2`**
`0b`____________	**`0b`**____________	**`0b`**____________

b) [3 pts] What 32-bit bit pattern would be used in IEEE 754 floating point to represent the decimal value -1 (e.g. in a C float)?

______ S (1 bit) ____________ E (8 bits) ________________________ M (23 bits)

c) [3 pts] On a 64-bit word machine, you are given the following array declaration in C:

```
double x[8][2]
```

If `x` starts at address 0, what will the expression `&(x[2][4])` evaluate to? If "unknown" or "cannot be guaranteed", state that. Otherwise give your answer as a single number in **<u>decimal</u>**.

6. Pointers & Memory (9 points)

We are using a 64-bit x86-64 machine (**little endian**). Below is the `husky` function disassembly, *showing where the code is stored in memory*. Hint: read the questions before reading the assembly!

```
0000000000400507 <husky>:
  400507:       48 83 fe 02             cmp    $0x2,%rsi
  40050b:       7f 05                   jg     400512 <husky+0xb>
  40050d:       48 8d 04 7f             lea    (%rdi,%rdi,2),%rax
  400511:       c3                      retq
  400512:       48 83 ec 08             sub    $0x8,%rsp
  400516:       48 83 ee 01             sub    $0x1,%rsi
  40051a:       e8 e8 ff ff ff          callq  400507 <husky>
  40051f:       48 83 c4 08             add    $0x8,%rsp
  400523:       c3                      retq
```

a) [4 pts] 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.*

Instruction	Register	Value (in hex)
	`%rax`	`0x0000 0000 0040 050d`
	`%rsi`	`0x0000 0000 0000 0010`
`movswl 4(%rsi,%rax), %ecx`	`%rcx`	
`leaw (%rsi,%rsi,2), %di`	`%di`	

b) [4 pts] Complete the C code below to fulfill the behaviors described in the inline comments using pointer arithmetic. Let **`short* shortP = 0x400514`**

```
short v1 = shortP[________];                     // set v1 = 0x048d

long* v2 = (long*) ((__________*)shortP + 3); // set v2 = 0x400520
```

c) [1 pt] **`husky`** is a recursive function. What address is put on the stack when **`husky`** calls itself. Give the exact address:
