Full Name: ________________________________

"On my honor as a University of Colorado at Boulder student I have neither given nor received unauthorized assistance on this work."

CSCI 2400, Fall 2017

Final Exam

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name on the front.
- Write your answers in the space provided below the problem. Show your work. If you make a mess, clearly indicate your final answer.
- Feel free to use the back of pages, but indicate that you have done so.
- This exam is CLOSED BOOK and you can use a *single page* of notes along with our reference sheets. You can not use a computer or calculator.

Problem	Possible	Score
1	12	
2	12	
3	28	
4	20	
5	25	
6	10	
Total	107	

1. **[12 Points]**

```
#include <stdlib.h>
#include <stdio.h>

void main() {

    if(fork() && fork()){
        fork();
    }

    if(fork() || fork()){
        fork();
    }
printf("Hello World\n");

}
```

Answer:

The number of printfs is **20** (full 12 points)

Partial credit:

- 3 point for Answers: $64 \quad or \quad 20 \pm 12$
- 6 points for Answers: $12, 9 \quad or \quad 20 \pm 8$
- 9 points for Answers: $15, 10 \quad or \quad 20 \pm 4$

If one number satistify two or more conditions, take the highest socre.

2. **[12 Points]**

Answer:

- 1 **Answer: 2 [+4]**
- 2 **Answer: Count1 = 3, Count2 = 0 [+4]**
- 3 **Answer: Count1 = 2, Count2 = 1 [+4]**

3. **[28 Points]** The following problem concerns the way virtual addresses are translated into physical addresses.

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- The TLB is 4-way set associative with 8 total entries.
- The L1 Cache is 2-way set associative, with a 4-byte block size and 64 total bytes.
- Virtual addresses are 13 bits wide.
- Physical addresses are 11 bits wide.
- The page size is 32 bytes.

In the following tables, **all numbers are given in hexadecimal**. The contents of the TLB and a portion of the page tables are as follows:

TLB			
Index	Tag	PPN	Valid
0	1a	06	1
	–	–	0
	15	02	1
	–	–	0
1	14	0f	1
	–	–	0
	0a	0c	1
	07	04	1

Page Table		
VPN	PPN	Present
031	000	1
0a2	00c	1
032	009	1
051	004	1
03f	00d	1
02f	00a	1
00e	008	1
02c	003	1
021	00f	1
012	00b	1
01a	00e	1
03d	002	1
006	001	1
034	006	1
017	005	1
003	007	1

Cache			
Index	Valid	Tag	Data
0	1	1C	6021130E
	1	00	DCAEB820
1	0	12	1DFE0C46
	0	0B	29E5DBF8
2	1	1F	DFFBCC85
	1	02	CB570940
3	1	08	57A84A44
	1	3C	8E85761F
4	1	0D	DF2C1CE2
	1	07	BE10CEA4
5	1	04	579C4AB6
	1	0C	A11D81A1
6	1	13	B250AE92
	1	15	7751E21A
7	0	0C	6AA3E19A
	1	09	6AC09E41

(a) **[6 Points]** Calculate the number of bits for the following elements:

VPO The virtual page offset __5__ *TLBI* The TLB index __1__

VPN The virtual page number __8__ *TLBT* The TLB tag __7__

PPO The physical page offset __5__ *PPN* The physical page number __6__

(b) **[22 Points]** (1 points each) For the given virtual addresses, indicate the TLB entry accessed and the physical address. **Indicate whether the TLB misses and whether the entry is or is not in the page table.** If the physical page number and address can not be determined, write "N/A". Then if a physical address exists indicate the cache translation parts, if its a cache hit, and a value if applicable. If any part can't be determined just write "N/A".

Virtual address: `0x0549`

(i) Address translation

Parameter	Value
VPN	0x2a
TLB Index	0x0
TLB Tag	0x15
TLB Hit? (Y/N)	Y
In Page Table? (Y/N)	N
PPN	0x2

(ii) Cache Translation

Parameter	Value
Cache Offset	0x1
Cache Index	0x2
Cache Tag	0x02
Cache Hit? (Y/N)	Y
Byte Value	0x57

Virtual address: `0x0244`

(i) Address translation

Parameter	Value
VPN	0x12
TLB Index	0x0
TLB Tag	0x09
TLB Hit? (Y/N)	N
In Page Table? (Y/N)	Y
PPN	0x0b

(ii) Cache Translation

Parameter	Value
Cache Offset	0x0
Cache Index	0x1
Cache Tag	0x0b
Cache Hit? (Y/N)	N
Byte Value	0xN/A

4. **[20 Points]**

Suppose our memory allocator uses an implicit free list with both header and footer. Assume a word size of eight bytes, and that all blocks are aligned to addresses divisible by eight. You should assume that the addresses you see span the entire heap, and that a block is marked as allocated by setting the least significant bit of the header and footer to 1. Similarly, a block is marked as free by setting the least significant bit of the header and footer to 0. Note that each row in the heap pictured below represents one eight-byte word. **Assume a best-fit placement policy.**

Address	Value
FF00	00 . . . 00 18
FF08	?? . . . ??
FF10	00 . . . 00 18
FF18	00 . . . 00 29
FF20	?? . . . ??
FF28	?? . . . ??
FF30	?? . . . ??
FF38	00 . . . 00 29
FF40	00 . . . 00 19
FF48	?? . . . ??
FF50	00 . . . 00 19
FF58	00 . . . 00 21
FF60	?? . . . ??
FF68	?? . . . ??
FF70	00 . . . 00 21
FF78	00 . . . 00 30
FF80	?? . . . ??
FF88	?? . . . ??
FF90	?? . . . ??
FF98	?? . . . ??
FFA0	00 . . . 00 30
FFA8	00 . . . 00 21
FFB0	?? . . . ??
FFB8	?? . . . ??
FFC0	00 . . . 00 21
FFC8	00 . . . 00 39
FFD0	?? . . . ??
FFD8	?? . . . ??
FFE0	?? . . . ??
FFE8	?? . . . ??
FFF0	?? . . . ??
FFF8	00 . . . 00 39

Unless clearly marked otherwise, assume all numbers are in hexadecimal!

Suppose that, after some sequence of `malloc`'s and `free`'s, the state of the heap is as you see it on the left. Then, assume that the following calls to `malloc` and `free` are made:

```
l0: void* p1 = malloc(0x08);
l1: free(0xff48);
l2: free(0xff60);
l3: free(0xffd0);
l4: void* p2 = malloc(0x30);
```

And answer the following:

(a) **[5 Points]** How much space on the heap does the smallest valid block size take up, in bytes?

Answer:
24 (or 0x18)
Partial credit: 1 point for 8 (or 0x8)

(b) **[5 Points]** What is `p1`?

Answer:
0xff08(or ff08) partical credit: 2 points for (ff00 or 0xff00)

(c) **[5 Points]** Which of the five lines above will cause the allocator to perform a 'coalesce' operation? (ie, `l0`, `l1`, `l2`, `l3`, or `l4`?)

Answer:
Line 2

(d) **[5 Points]** What is `p2`?

Answer:
0xff48 partial credit: 2 points for ff00, ffd0, 0xff00, 0xffdo; 1 point for 0xffe8 or ffe8.

Unless clearly marked otherwise, assume all numbers are in hexadecimal!

5. **[25 Points]**

Answer these questions on linking

(a) **[15 Points]** For the following code, identify the symbols listed in the symbol table of the ELF relocatable object files (.o), whether that symbol is defined or undefined, and if defined, then in which section of the corresponding ELF file that the symbol would be defined.

main.c

```c
extern void func();
int p=7;
int q;
int main()
{
        int m=50000;
        q=sqrt(m);
        func(q);
        return 0;
}
```

func.c

```c
int n=10;
int temp;
int func(int x)
{
        if(x>100)
                temp=x;
        else
                temp=-x;
        return temp;
}
```

main.o

Symbol Name	Defined/undefined	Section
func	undefined	–
p	defined	.data
q	defined	.bss
main	defined	.text

func.o

Symbol Name	Defined/undefined	Section
n	defined	.data
temp	defined	.bss
func	defined	.text

(b) **[5 Points]** For the code in Question (a), the sizes of the .text and .data sections of the .o relocatable object files are listed below. The two object files above are then linked together with the command line `ld -o p main.o func.o`. Assume the object files are combined similar to the order shown in the lecture slides and the starting address of the .text section of the unified executable object file starts at 0x8048501. What is the relocated address of p?

File+Section	Size (Byte)
main.o's .text	32
main.o's .data	8
func.o's .text	58
func.o's .data	4

Answer:

p is initialized so it is in the main.o's .data section. Then the relocation address is: starting address + main.o's .text + func.o's .text = 0x8048501 + 32 + 58 = 0x8048501 + 0x5a = 0x804855b. +4 for 0x804855b, +2 for 0x8048501 + 32 + 58 + 8 = 0x8048501 + 0x62 = 0x8048563, +1 for 0x8048501 + 32 + 58 + 8 + 4 = 0x8048501 + 0x66 = 0x8048567, otherwise 0.

(c) **[5 Points]** When the two .o files above are linked together with the command line `ld -o p main.o func.o`, the virtual addresses of the merged and relocated various subsections follow what kind of orderings, from lowest to highest addresses (circle)? (can circle more than one correct answer):

Answer:

c and e are the only two correct answers. +5 for only circling the two correct answers, +3 if 1 correct after cancellations, otherwise 0.

6. **[10 Points]** For each of the following, answer True or False:

(a) ________ For a binary number, left shift by 1 corresponds to division by 2.

(b) ________ The stack pointer in 64-bit x86 systems is stored in the %rsp register.

(c) ________ There is no difference between binary encoding of integers and floating point.

(d) ________ Each Y86 instruction can be divided into 6 stages of execution: Fetch, Decode, Execute, Memory, Write, Update PC

(e) ________ for(i=0;i$\leq$100;i++)sum+=a[0]; For the given code, cache helps on spatial locaity.

Answer:

(a) __ False __ For a binary number, left shift by 1 corresponds to division by 2.

(b) __ True __ The stack pointer in 64-bit x86 systems is stored in the %rsp register.

(c) __ False __ There is no difference between binary encoding of integers and floating point.

(d) __ True __ Each Y86 instruction can be divided into 6 stages of execution: Fetch, Decode, Execute, Memory, Write, Update PC

(e) __ False __ for(i=0;i$\leq$100;i++)sum+=a[0]; For the given code, cache helps on spatial locaity.