Additional Study Questions (CS33 Final)

What is this document? This page has lots of questions developed by authors of the textbook to be used as exam questions. I've tried to narrow down to questions that would be relevant for our course. They are pretty good, but some of them are specific to x86-32bit, so be careful. I've put the answers after each question in white, so to see the answers, just highlight the text after the question.

Problem 9. (16 points):

This problem tests your understanding of exceptional control flow in C programs. Assume we are running code on a Unix machine. The following problems all concern the value of the variable `counter`.

Part I (6 points)

```
int counter = 0;

int  main()
{
    int i;

    for (i = 0; i < 2; i ++){
        fork();
        counter ++;
        printf("counter = %d\n", counter);
    }

    printf("counter = %d\n", counter);
    return 0;
}
```

A. How many times would the value of `counter` be printed: _____

B. What is the value of `counter` printed in the first line? _____

C. What is the value of `counter` printed in the last line? _____

Answer:

Problem 7. (14 points):

Consider a direct mapped cache of size 64K with block size of 16 bytes. Furthermore, the cache is write-back and write-allocate. You will calculate the miss rate for the following code using this cache. Remember that `sizeof(int) == 4`. Assume that the cache starts empty and that local variables and computations take place completely within the registers and do not spill onto the stack.

A. Now consider the following code to copy one matrix to another. Assume that the `src` matrix starts at address 0 and that the `dest` matrix follows immediately follows it.

```
void copy_matrix(int dest[ROWS][COLS], int src[ROWS][COLS])
{
    int i, j;

    for (i=0; i<ROWS; i++) {
        for (j=0; j<COLS; j++) {
            dest[i][j] = src[i][j];
        }
    }
}
```

1. What is the cache miss rate if `ROWS = 128` and `COLS = 128`?
   Miss rate = _____%
2. What is the cache miss rate if `ROWS = 128` and `COLS = 192`?
   Miss rate = _____%
3. What is the cache miss rate if `ROWS = 128` and `COLS = 256`?
   Miss rate = _____%

Answer:

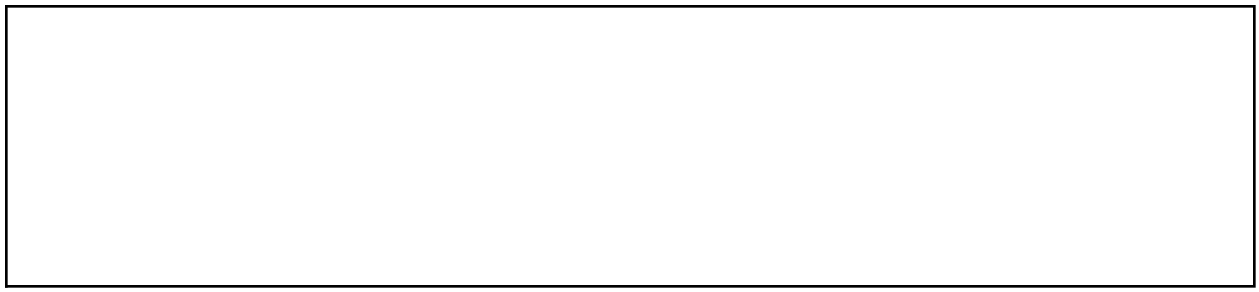
B. Now consider the following two implementations of a horizontal flip and copy of the matrix. Again assume that the `src` matrix starts at address 0 and that the `dest` matrix follows immediately follows it.

```
void copy_n_flip_matrix1(int dest[ROWS][COLS], int src[ROWS][COLS])
{
    int i, j;

    for (i=0; i<ROWS; i++) {
        for (j=0; j<COLS; j++) {
            dest[i][COLS - 1 - j] = src[i][j];
        }
    }
}
```

1. What is the cache miss rate if `ROWS = 128` and `COLS = 128`?
   Miss rate = _____%
2. What is the cache miss rate if `ROWS = 128` and `COLS = 192`?
   Miss rate = _____%

```
void copy_n_flip_matrix2(int dest[ROWS][COLS], int src[ROWS][COLS])
{
    int i, j;

    for (j=0; j<COLS; j++) {
        for (i=0; i<ROWS; i++) {
            dest[i][COLS - 1 - j] = src[i][j];
        }
    }
}
```

1. What is the cache miss rate if `ROWS = 128` and `COLS = 128`?
   Miss rate = _____%
2. What is the cache miss rate if `ROWS = 192` and `COLS = 128`?
   Miss rate = _____%



Problem 6. (6 points):

The following table gives the parameters for a number of different caches, where m is the number of physical address bits, C is the cache size (number of data bytes), B is the block size in bytes, and E is the number of lines per set. For each cache, determine the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b).

Cache	m	C	B	E	S	t	s	b
1.	32	1024	4	4				
2.	32	1024	4	256				
3.	32	1024	8	1				
4.	32	1024	8	128				
5.	32	1024	32	1				
6.	32	1024	32	4				

Problem 1. (20 points):

We are running programs on a machine with the following characteristics:

- Values of type `int` are 32 bits. They are represented in two's complement, and they are right shifted arithmetically. Values of type `unsigned` are 32 bits.
- Values of type `float` are represented using the 32-bit IEEE floating point format, while values of type `double` use the 64-bit IEEE floating point format.

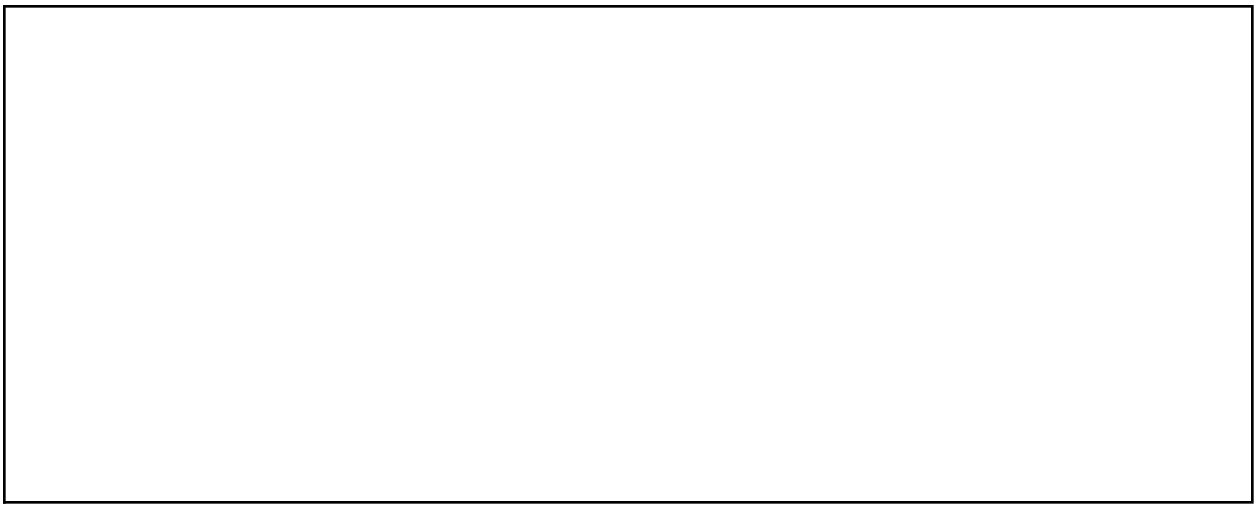
We generate arbitrary values `x`, `y`, and `z`, and convert them to other forms as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to other forms */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
double   dx = (double) x;
double   dy = (double) y;
double   dz = (double) z;
```

For each of the following C expressions, you are to indicate whether or not the expression *always* yields 1. If so, circle "Y". If not, circle "N". You will be graded on each problem as follows:

- If you circle no value, you get 0 points.
- If you circle the right value, you get 2 points.
- If you circle the wrong value, you get −1 points (so don't just guess wildly).

Expression	Always True?
`(x<y) == (-x>-y)`	Y N
`((x+y)<<4) + y-x == 17*y+15*x`	Y N
`~x+~y+1 == ~(x+y)`	Y N
`ux-uy == -(y-x)`	Y N
`(x >= 0) || (x < ux)`	Y N
`((x >> 1) << 1) <= x`	Y N
`(double)(float) x == (double) x`	Y N
`dx + dy == (double) (y+x)`	Y N
`dx + dy + dz == dz + dy + dx`	Y N
`dx * dy * dz == dz * dy * dx`	Y N



Problem 5. (15 points):

The following problem concerns the way virtual addresses are translated into physical addresses.

- The memory is byte addressable, and memory accesses are to 1-byte {**not 4-byte**} words.
- Virtual addresses are 18 bits wide.
- Physical addresses are 12 bits wide.
- The page size is 512 bytes.
- The TLB is 8-way set associative with 16 total entries.
- The cache is 2-way set associative, with a 4-byte line size and 32 total entries.

In the following tables, **all numbers are given in hexadecimal**. The contents of the TLB and the page table for the first 32 pages, and the cache are as follows:

TLB			
Index	Tag	PPN	Valid
0	55	6	0
	48	F	1
	00	A	0
	32	9	1
	6A	3	1
	56	1	0
	60	4	1
	78	9	0
1	71	5	1
	31	A	1
	53	F	0
	87	8	0
	51	D	0
	39	E	1
	43	B	0
	73	2	1

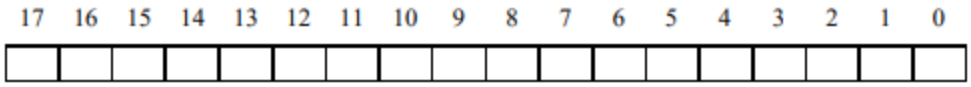
Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
000	7	0	010	1	0
001	5	0	011	3	0
002	1	1	012	3	0
003	5	0	013	0	0
004	0	0	014	6	1
005	5	0	015	5	0
006	2	0	016	7	0
007	4	1	017	2	1
008	7	0	018	0	0
009	2	0	019	2	0
00A	3	0	01A	1	0
00B	0	0	01B	3	0
00C	0	0	01C	2	0
00D	3	0	01D	7	0
00E	4	0	01E	5	1
00F	7	1	01F	0	0

2-way Set Associative Cache												
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	7A	1	09	EE	12	64	00	0	99	04	03	48
1	02	0	60	17	18	19	7F	1	FF	BC	0B	37
2	55	1	30	EB	C2	0D	0B	0	8F	E2	05	BD
3	07	1	03	04	05	06	5D	1	7A	08	03	22

Part 1

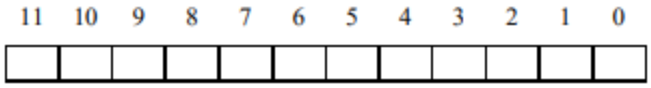
1. The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following. (If a field doesn't exist, don't draw it on the diagram.)

   VPO The virtual page offset
   VPN The virtual page number
   TLBI The TLB index
   TLBT The TLB tag



2. The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

   PPO The physical page offset
   PPN The physical page number
   CO The Cache Block Offset
   CI The Cache Index
   CT The Cache Tag



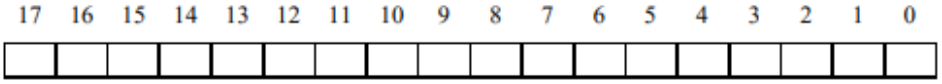
Part 2

For the given virtual addresses, indicate the TLB entry accessed and the physical address. Indicate whether the TLB misses and whether a page fault occurs.

If there is a cache miss, enter "-" for "Cache Byte Returned." If there is a page fault, enter "-" for "PPN" and leave part C blank.

Virtual address: `0x1A9F4`

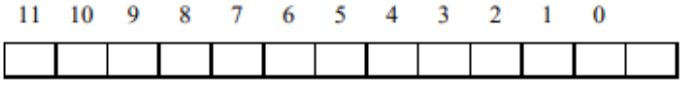
1. Virtual address format (one bit per box)



2. Address translation

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

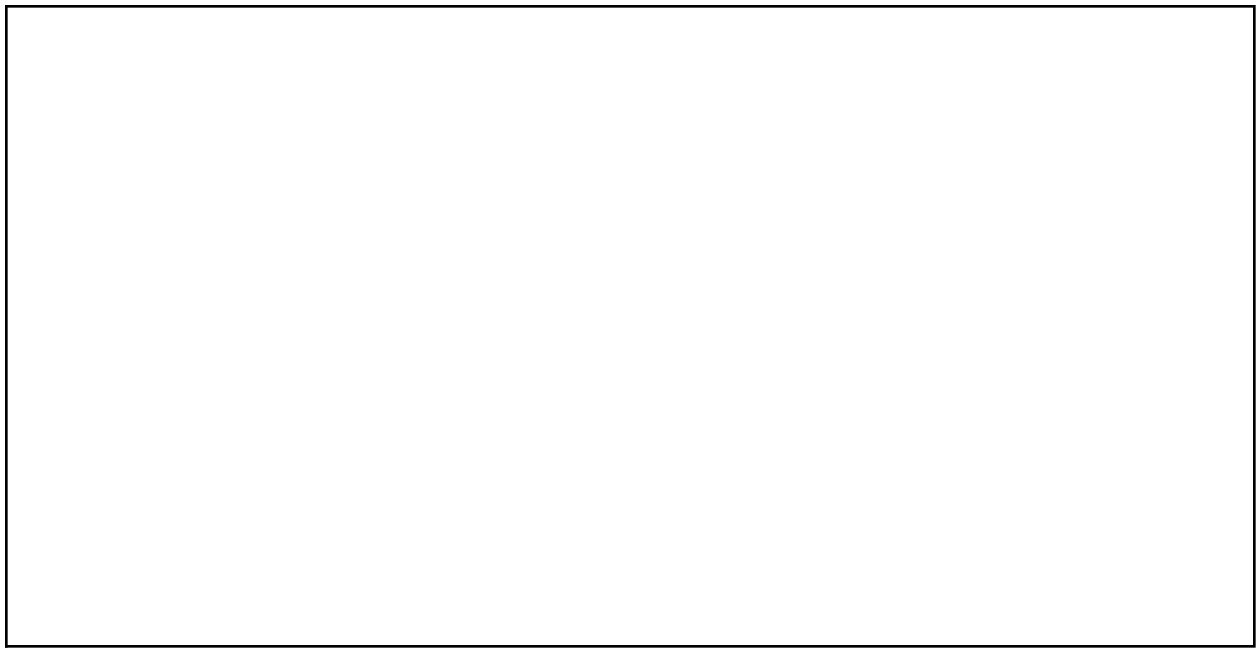
3. Physical address format (one bit per box)



4. Physical memory reference

Parameter	Value
Block Offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Value of Cache Byte Returned	0x

Answers:



Problem 6. (10 points):

This question deals with various aspects of the implementation of dynamic memory allocators.

Memory utilization

Suppose your memory allocator uses a simple implicit list, and each block in the heap has a 1 word header and footer. Assume that the heap has a unused word at the start of the heap to enforce double-word alignment, followed by a prologue block consisting of only a header and footer. At the end of the heap is a 1 word epilogue header. The allocator's minimum allocation unit is 8 bytes. The total size of the heap is 2048 bytes.

A. If the heap is full (this means a request of `malloc(1)` will fail), what is worst case memory utilization of the heap?

B. Briefly (no more than 2 sentences), describe the difference between internal and external fragmentation.

Free list strategies

Consider a heap with N blocks, where $M \leq N$ are allocated. For the following questions, express your answer in O-notation:

A. What is the worst case running time of first fit allocation with explicit free lists?

B. What is the worst case running time of first fit allocation with implicit free lists?

C. What is the worst case running time of best fit allocation with explicit free lists?

Coalescing strategies

Immediate coalescing is the strategy we have seen where a block is coalesced with its neighbor(s) immediately after it is freed. Another possible strategy is *lazy* or deferred coalescing, where free blocks are not coalesced immediately. Usually, coalescing is done when a allocation request cannot be fulfilled.

For the following two questions, express your answer in O-notation. Borrowing from the previous section, there are N blocks in the heap, where $M \leq N$ are allocated. Assume the allocator uses some form of explicit free lists.

A. What is the worst case running time of coalescing using an immediate strategy?

B. What is the worst case running time of coalescing using a lazy strategy?

For the following situations, describe whether an allocator using immediate or lazy coalesing (coalescing done when an allocation request fails) would be more appropriate. If the choice of coalescing strategy has negligible performance impact, write "does not matter".

D. The allocator allocates only two types of structures that have fixed sizes, and expects heavy reuse patterns.

E. The allocator must operate in real time. This means the allocator makes hard performance guarantees that its operations will not take more than some fixed specified amount of time.

F. The allocator will alternately allocate and free blocks where the next allocation request is 1/2 the size of the last request.

Answers:

Problem 7. (9 points):

Part 1

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

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

#define NTHREADS 20

void *thread(void *vargp)
{
  static int cnt = 0;

  cnt++;
  printf("%d\n", cnt);
}

int main ()
{
  int i;
  pthread_t tid;

  for(i = 0; i < NTHREADS; i++)
  {
    pthread_create(&tid, NULL, thread, NULL);
  }

  pthread_exit(NULL);

}
```

What are the maximal guarantees you can make about the output of the above program? Check all that apply.
Note: Checking all of the boxes implies that the output is

```
1
2
3
...
20
```

☐ 20 numbers will be printed.
☐ The numbers lie in the range 1 through 20, inclusive.
☐ There are no duplicate numbers printed.
☐ The numbers will be printed in ascending order.

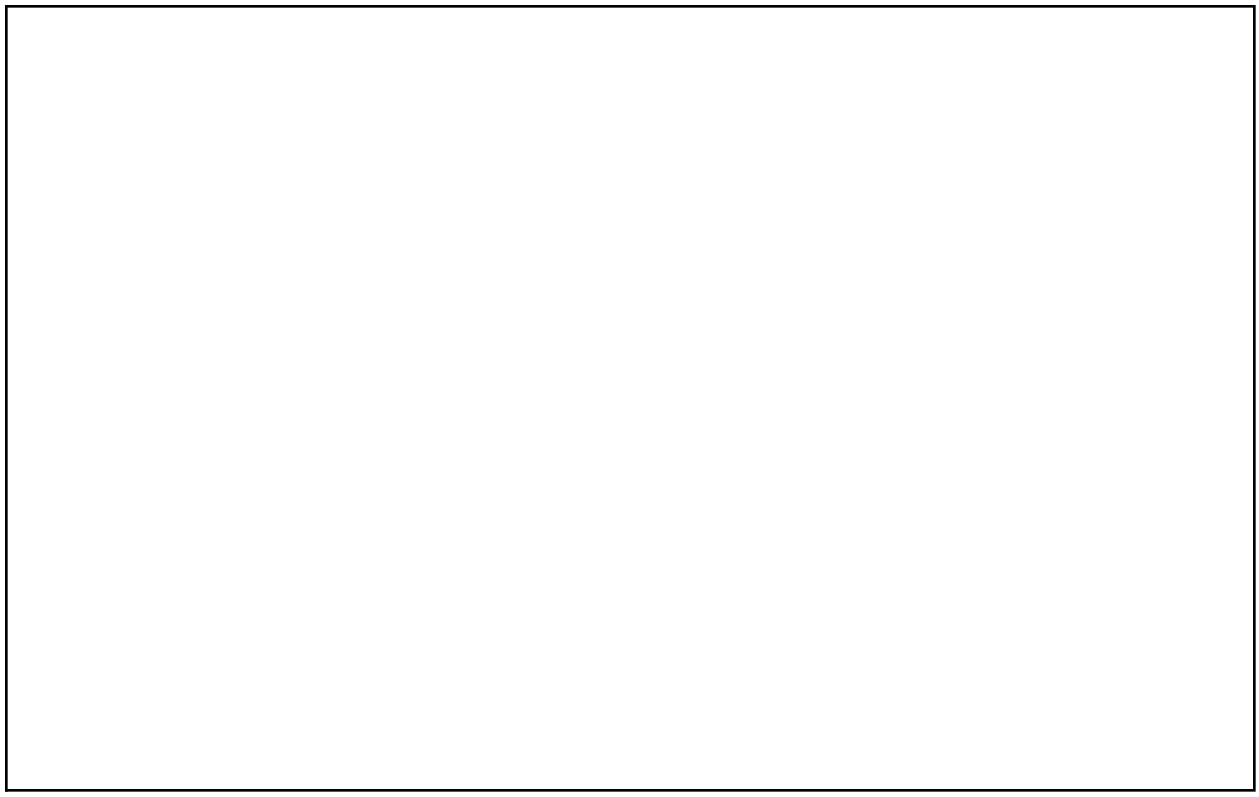
Using mutexes, modify the code to guarantee that the output of the program is

```
1
2
3
...
20
```

The numbers 1 through 20 printed sequentially in ascending order.
Here are the mutex operations:

```
int pthread_mutex_init(pthread_mutex *mutex, const pthread_mutexattr_t *mutexattr);
int pthread_mutex_lock(pthread_mutex_t *mutex));
int pthread_mutex_unlock(pthread_mutex_t *mutex));
int pthread_mutex_destroy(pthread_mutex_t *mutex));
```

Answers:



Problem 1. (18 points):

Multiple choice questions on a variety of stimulating and refreshing topics.

To receive credit, you must write your answer for each question in the following table:

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	-	-
								-	-

1. Each thread has its own _____.

   (a) Heap
   (b) Stack
   (c) Global values
   (d) Text data

2. Simply decreasing the size of block headers used internally by malloc:

   (a) Decreases internal fragmentation
   (b) Increases internal fragmentation
   (c) Decreases external fragmentation
   (d) Increases external fragmentation

3. Which of the following sentences about reader-writer locks is not true?

   (a) Many readers can hold the same rwlock at the same time
   (b) Two writers cannot hold the same rwlock at the same time
   (c) Many readers and exactly one writer can hold the same rwlock at the same time
   (d) An rwlock can be used as a mutex

4. Which of the following is the correct ordering (left-to-right) of a file's compilation cycle (a filename with no extension is an executable):

   (a) `foo.c` → `foo.o` → `foo.s` → `foo`
   (b) `foo` → `foo.s` → `foo.o` → `foo.c`
   (c) `foo.c` → `foo.s` → `foo` → `foo.o`
   (d) `foo.c` → `foo.s` → `foo.o` → `foo`

5. Suppose an `int` A is stored at virtual address `0xff987cf0`, while another `int` B is stored at virtual address `0xff987d98`. If the size of a page is `0x1000` bytes, then A's physical address is numerically less than B's physical address.

   (a) Always true
   (b) Always false
   (c) Sometimes true, sometimes false
   (d) Not enough information

6. Assuming no errors, which one of the following functions returns exactly once?

   (a) `fork()`
   (b) `execve()`
   (c) `exit()`
   (d) `longjmp()`
   (e) `waitpid()`

7. On a 64-bit system, which of the following C expressions is equivalent to the C expression `(x[2] + 4)[3]`? Assume x is declared as `int **x`.

   (a) `*((*(x + 16)) + 28)`
   (b) `*((*(x + 2)) + 7)`
   (c) `**(x + 28)`
   (d) `*(((*x) + 2) + 7)`
   (e) `(**(x + 2) + 7)`

11. In an x86-64 system, how many integers can be stored in a cache line if your cache is 4KB, is 4-way set-associative, and contains 4 sets?

    (a) 8
    (b) 16
    (c) 32
    (d) 64
    (e) 128

12. Which types of locality are leveraged by virtual memory?

    (a) Spatial locality
    (b) Temporal locality
    (c) Prime locality
    (d) Both (a) and (b)
    (e) Both (b) and (c)

13. Which of the following is not a section of an ELF file?

    (a) `.text`
    (b) `.static`
    (c) `.rodata`
    (d) `.data`
    (e) `.bss`

14. Choose the true statement.

    (a) All thread-safe functions are reentrant.
    (b) Some reentrant functions are not thread safe.
    (c) It is never a good idea to use persistent state across multiple function calls.
    (d) It is impossible to have a race condition between two threads as long as they have no shared state.
    (e) All functions which call non-thread-safe functions are themselves not thread safe.

15. We use dynamic memory because:

    (a) The heap is significantly faster than the stack.
    (b) The stack is prone to corruption from buffer overflows.
    (c) Storing data on the stack requires knowing the size of that data at compile time.
    (d) None of the above.

17. Which of the following is true about races?

(a) A race occurs when correctness of the program depends on one thread reaching point a before another thread reaches point b.

(b) Exclusive access to all shared resources eliminates race conditions.

(c) Race conditions are the same as deadlocks.

(d) All race conditions occur inside loops, since that is the only way we can interleave processes.

(more)

18. Consider the following two blocks of code, which are contained in *separate files*:

```
/* main.c */
int i = 0;
int main() {
    foo();
    return 0;
}
```

```
/* foo.c */
int i = 1;
void foo() {
    printf("%d", i);
}
```

What will happen when you attempt to compile, link, and run this code?

(a) It will fail to compile.

(b) It will fail to link.

(c) It will raise a segmentation fault.

(d) It will print "0".

(e) It will print "1".

(f) It will sometimes print "0" and sometimes print "1".

Problem 2. (6 points):

Floating point encoding. In this problem, you will work with floating point numbers based on the IEEE floating point format. We consider two different 6-bit formats:

Format A:

- There is one sign bit s.
- There are $k = 3$ exponent bits. The bias is $2^{k-1} - 1 = 3$.
- There are $n = 2$ fraction bits.

Format B:

- There is one sign bit s.
- There are $k = 2$ exponent bits. The bias is $2^{k-1} - 1 = 1$.
- There are $n = 3$ fraction bits.

For formats A and B, please write down the binary representation for the following (use round-to-even). Recall that for denormalized numbers, $E = 1 - \text{bias}$. For normalized numbers, $E = e - \text{bias}$.

Value	Format A Bits	Format B Bits
One	`0 011 00`	`0 01 000`
Three		
7/8		
15/8		

Problem 3. (6 points):

Arrays. Consider the C code below, where H and J are constants declared with `#define`.

```
int array1[H][J];
int array2[J][H];

void copy_array(int x, int y) {
    array2[x][y] = array1[y][x];
}
```

Suppose the above C code generates the following x86-64 assembly code:

```
# On entry:
#    %edi = x
#    %esi = y
#
copy_array:
    movslq  %esi,%rsi
    movslq  %edi,%rdi
    movq    %rdi, %rax
    salq    $4, %rax
    subq    %rdi, %rax
    addq    %rsi, %rax
    leaq    (%rsi,%rsi,4), %rsi
    leaq    (%rdi,%rsi,2), %rsi
    movl    array1(,%rsi,4), %edx
    movl    %edx, array2(,%rax,4)
    ret
```

What are the values of `H` and `J`?

`H =`

`J =`

Problem 8. (10 points):

Exceptional control flow. Consider the following C program. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
int main()
{
    int val = 2;

    printf("%d", 0);
    fflush(stdout);

    if (fork() == 0) {
        val++;
        printf("%d", val);
        fflush(stdout);
    }
    else {
        val--;
        printf("%d", val);
        fflush(stdout);
        wait(NULL);
    }
    val++;
    printf("%d", val);
    fflush(stdout);
    exit(0);
}
```

For each of the following strings, circle whether (Y) or not (N) this string is a possible output of the program. You will be graded on each sub-problem as follows:

- If you circle no answer, you get 0 points.
- If you circle the right answer, you get 2 points.
- If you circle the wrong answer, you get -1 points (so don't just guess wildly).

A. `01432` Y N

B. `01342` Y N

C. `03142` Y N

D. `01234` Y N

E. `03412` Y N

Problem 9. (12 points):

Address translation. This problem concerns the way virtual addresses are translated into physical addresses. Imagine a system has the following parameters:

- Virtual addresses are 20 bits wide.
- Physical addresses are 18 bits wide.
- The page size is 1024 bytes.
- The TLB is 2-way set associative with 16 total entries.

The contents of the TLB and the first 32 entries of the page table are shown as follows. **All numbers are given in hexadecimal**.

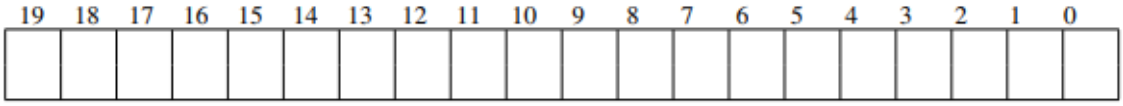
TLB			
Index	Tag	PPN	Valid
0	03	C3	1
	01	71	0
1	00	28	1
	01	35	1
2	02	68	1
	3A	F1	0
3	03	12	1
	02	30	1
4	7F	05	0
	01	A1	0
5	00	53	1
	03	4E	1
6	1B	34	0
	00	1F	1
7	03	38	1
	32	09	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
000	71	1	010	60	0
001	28	1	011	57	0
002	93	1	012	68	1
003	AB	0	013	30	1
004	D6	0	014	0D	0
005	53	1	015	2B	0
006	1F	1	016	9F	0
007	80	1	017	62	0
008	02	0	018	C3	1
009	35	1	019	04	0
00A	41	0	01A	F1	1
00B	86	1	01B	12	1
00C	A1	1	01C	30	0
00D	D5	1	01D	4E	1
00E	8E	0	01E	57	1
00F	D4	0	01F	38	1

Part 1

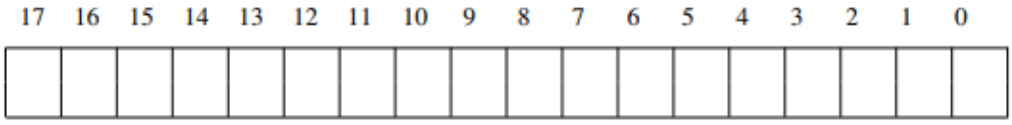
1. The diagram below shows the format of a virtual address. Please indicate the following fields by labeling the diagram:

   VPO The virtual page offset
   VPN The virtual page number
   TLBI The TLB index
   TLBT The TLB tag



2. The diagram below shows the format of a physical address. Please indicate the following fields by labeling the diagram:

   PPO The physical page offset
   PPN The physical page number

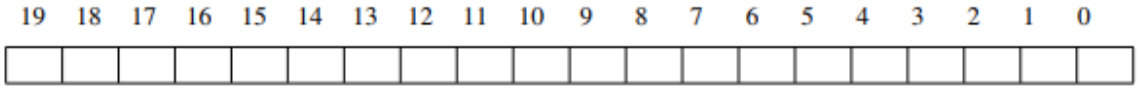


Part 2

For the given virtual addresses, please indicate the TLB entry accessed and the physical address. Indicate whether the TLB misses and whether a page fault occurs. If there is a page fault, enter "-" for "PPN" and leave the physical address blank.

Virtual address: `078E6`

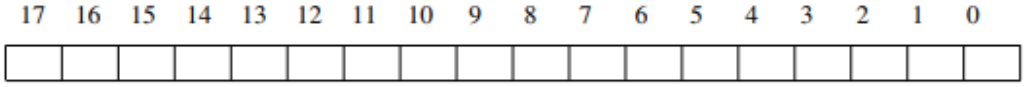
1. Virtual address (one bit per box)



2. Address translation

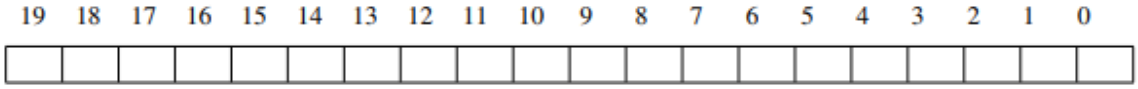
Parameter	Value	Parameter	Value
VPN	0x	TLB Hit? (Y/N)	
TLB Index	0x	Page Fault? (Y/N)	
TLB Tag	0x	PPN	0x

3. Physical address(one bit per box)



Virtual address: `04AA4`

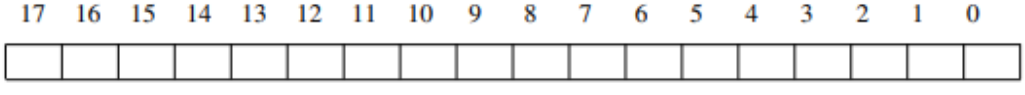
1. Virtual address (one bit per box)



2. Address translation

Parameter	Value	Parameter	Value
VPN	0x	TLB Hit? (Y/N)	
TLB Index	0x	Page Fault? (Y/N)	
TLB Tag	0x	PPN	0x

3. Physical address(one bit per box)



Problem 10. (10 points):

Concurrency, races, and synchronization. Consider a simple concurrent program with the following specification: The main thread creates two peer threads, passing each peer thread a unique integer *thread ID* (either 0 or 1), and then waits for each thread to terminate. Each peer thread prints its thread ID and then terminates.

Each of the following programs attempts to implement this specification. However, some are incorrect because they contain a race on the value of `myid` that makes it possible for one or more peer threads to print an incorrect thread ID. Except for the race, each program is otherwise correct.

You are to indicate whether or not each of the following programs contains such a race on the value of `myid`. You will be graded on each subproblem as follows:

- If you circle no answer, you get 0 points.
- If you circle the right answer, you get 2 points.
- If you circle the wrong answer, you get -1 points (so don't just guess wildly).

A. **Does the following program contain a race on the value of `myid`?** Yes No

```
void *foo(void *vargp) {
    int myid;
    myid = *((int *)vargp);
    Free(vargp);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i, *ptr;

    for (i = 0; i < 2; i++) {
        ptr = Malloc(sizeof(int));
        *ptr = i;
        Pthread_create(&tid[i], 0, foo, ptr);
    }
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

B. Does the following program contain a race on the value of `myid`? Yes No

```
void *foo(void *vargp) {
    int myid;
    myid = *((int *)vargp);
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i;

    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], NULL, foo, &i);
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
}
```

C. Does the following program contain a race on the value of `myid`? Yes No

```
void *foo(void *vargp) {
    int myid;
    myid = (int)vargp;
    printf("Thread %d\n", myid);
}

int main() {
    pthread_t tid[2];
    int i;

    for (i = 0; i < 2; i++)
        Pthread_create(&tid[i], 0, foo, i);
    Pthread_join(tid[0], 0);
    Pthread_join(tid[1], 0);
}
```

Problem 1. (10 points):

General systems topics. Write your answer for each question in the following table:

1	2	3	4	5	6	7	8	9	10

1. Consider a direct-mapped cache memory. Which one of the following statements is true?
   (a) The cache has 1 line per set.
   (b) The cache has 1 word per block.
   (c) The cache has 1 set per cache.
   (d) None of the above.
2. Which one of the following statements about cache memories is true:
   (a) Larger caches are more susceptible to capacity misses than smaller caches.
   (b) Caches with lower associativity are more susceptible to conflict misses than those with higher associativy.
   (c) Caches with higher associativity are more susceptile to cold misses than those with lower associativey.
   (d) None of the above
3. Which one of the following is NOT contained in an ELF executable file?
   (a) Machine code
   (b) Global variables
   (c) User stack
   (d) Symbol table
4. Assuming no errors, which one of the following statements about `fork` is true?
   (a) Called once, returns once.
   (b) Called once, returns twice.
   (c) Called once, returns never.
   (d) Called twice, returns once.
   (e) None of the above.

5. Assuming no errors, which one of the following statements about `execve` is true?

   (a) Called once, returns once.
   (b) Called once, returns twice.
   (c) Called once, returns never.
   (d) Called twice, returns once.
   (e) None of the above.

6. Which one of the following statements about processes is false?

   (a) The operating system kernel runs as its own separate process.
   (b) Each process shares the CPU with other processes.
   (c) Each process has its own private address space.
   (d) The environment for a process is stored on the stack.

7. What happens if the parent of a zombie child terminates?

   (a) The zombie child becomes a wraith and is never reaped.
   (b) The zombie child is reaped by the init process.
   (c) The zombie child is reaped by the process with the nearest PID.
   (d) None of the above.

9. Which one of the following statements is NOT true of storage allocators?

   (a) In the best case, coalescing with boundary tags is linear in the number of free blocks.
   (b) Seglists typically approximate best fit search.
   (c) Payloads must be aligned to some boundary.
   (d) Explicit lists are typically faster than implicit lists.
   (e) None of the above.

10. Which one of the following addresses is 8-byte aligned?

    (a) 1110110101110111_2
    (b) 1110110101110100_2
    (c) 1110110101110000_2
    (d) 1110110101110110_2
    (e) None of the above

Problem 4. (10 points):

Structure access. Consider the following data structure declarations:

```
struct data {
    long x;
    char str[16];
};
```

```
struct node {
    struct data d;
    struct node *next;
};
```

Below are given four C functions and four x86-64 code blocks. Next to each of the x86-64 code blocks, write the name of the C function that it implements.

```
int alpha(struct node *ptr) {
    return ptr->d.x;
}
```

```
char *beta(struct node *ptr) {
    ptr = ptr->next;
    return ptr->d.str;
}
```

```
char gamma(struct node *ptr) {
    return ptr->d.str[7];
}
```

```
long *delta(struct node *ptr) {
    struct data *dp =
        (struct data *) ptr;
    return &dp->x;
}
```

```
char *epsilon(struct node *ptr) {
    return &ptr->d.str[2];
}
```

_____
```
movsbl  15(%rdi),%eax
ret
```

_____
```
movq    (%rdi), %rax
ret
```

_____
```
movq    24(%rdi), %rax
addq    $8, %rax
ret
```

_____
```
movq    %rdi, %rax
ret
```

_____
```
leaq    10(%rdi), %rax
ret
```

Problem 7. (6 points):

Caches. In this problem you will estimate the miss rates for some C functions. Assumptions:

- 16-way set associative L1 cache ($E = 16$) with a block size of 32 bytes ($B = 32$).
- N is very large, so that a single row or column cannot fit in the cache.
- `sizeof(int) == 4`
- Variables `i`, `k`, and `sum` are stored in registers.
- The cache is cold before each function is called.

Part A (3 points)

```
int sum1(int A[N][N], int B[N][N])
{
    int i, k, sum = 0;

    for (i = 0; i < N; i++)
        for (k = 0; k < N; k++)
            sum += A[i][k] + B[k][i];
    return sum;
}
```

Circle the closest miss rate for `sum1`:

- 1/16
- 1/8
- 1/4
- 1/2
- 9/16
- 1

Part B (3 points)

```
int sum2(int A[N][N], int B[N][N])
{
    int i, k, sum = 0;

    for (i = 0; i < N; i++)
        for (k = 0; k < N; k++)
            sum += A[i][k] + B[i][k];
    return sum;
}
```

Circle the closest miss rate for `sum2`:

- 1/16
- 1/8
- 1/4
- 1/2
- 9/16
- 1

Problem 8. (10 points):

Exceptional control flow.

A. Consider the following C program. Assume the program executes to completion and that `fork`, `waitpid`, and `printf` always succeed.

```
int main() {
    pid_t pid;
    int sum = 0;

    if ((pid = fork()) == 0)
        sum += 10;
    else {
        waitpid(pid, NULL, 0);
        sum -= 5;
    }
    sum += 20;

    if (pid > 0)
        printf("Parent: sum=%d\n", sum);
    else
        printf("Child:  sum=%d\n", sum);
    return 0;
}
```

Show the output of this program:

```
Child:  sum=_________

Parent: sum=_________
```

B. Now consider the same program as in Part A, but with the call to `waitpid` removed. Assume the program executes to completion and that `printf` always succeeds. **Make no assumptions about the results of the other function calls.**

List all of the possible outputs of such a program. Each blank box holds the complete output from one execution of the program. Some blank boxes may be left unused.

C. Consider the C program below. Assume the program runs to completion and that all functions return normally.

```
int main ()
{
    if (fork() == 0) {
        if (fork() == 0) {
            printf("9");
            exit(1);
        }
        else
            printf("5");
    }
    else {
        pid_t pid;
        if ((pid = wait(NULL)) > 0) {
            printf("3");
        }
    }
    printf("0");
    return 0;
}
```

List four possible outputs for this program:

1. _____

2. _____

3. _____

4. _____

Problem 9. (9 points):

Imagine a system with the following attributes:

- The system has 1MB of virtual memory
- The system has 256KB of physical memory
- The page size is 4KB
- The TLB is 2-way set associative with 8 total entries.

The contents of the TLB and the first 32 entries of the page table are given below. **All numbers are in hexadecimal**.

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

A. Warmup Questions

(a) How many bits are needed to represent the virtual address space? _____

(b) How many bits are needed to represent the physical address space? _____

(c) How many bits are needed to represent a page table offset? _____

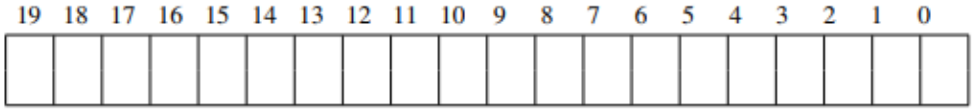
B. Virtual Address Translation I

Please step through the following address translation. Indicate a page fault by entering '-' for Physical Address.

Virtual address: `0x1F213`

Parameter	Value	Parameter	Value
VPN	0x	TLB Hit? (Y/N)	
TLB Index	0x	Page Fault? (Y/N)	
TLB Tag	0x	Physical Address	0x

Use the layout below as scratch space for the virtual address bits. To allow us to give you partial credit, clearly mark the bits that correspond to the VPN, TLB index (TLBI), and TLB tag (TLBT).



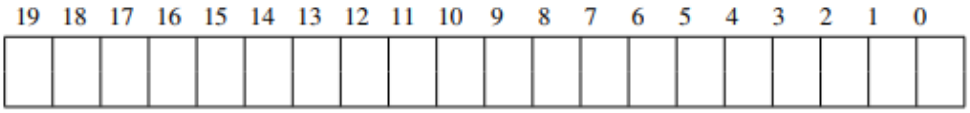
C. Virtual Address Translation II

Please step through the following address translation. Indicate a page fault by entering '-' for Physical Address.

Virtual address: `0x14213`

Parameter	Value	Parameter	Value
VPN	0x	TLB Hit? (Y/N)	
TLB Index	0x	Page Fault? (Y/N)	
TLB Tag	0x	Physical Address	0x

Use the layout below as scratch space for the virtual address bits. To allow us to give you partial credit, clearly mark the bits that correspond to the VPN, TLB index (TLBI), and TLB tag (TLBT).



Problem 1. (24 points):

Multiple choice.

Write the correct answer for each question in the following table:

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15	16	17	18	19	20	21	22	23	24

1. Which of the following is a legitimate difference between IA-32 and x86-64?

   (a) Buffer overflow exploits are impossible under x86-64.

   (b) IA-32 has caller- and callee-saved register conventions, while x86-64 does not.

   (c) Under x86-64, any instructions that take 32-bit operands are illegal.

   (d) None of the above.

2. Which of the following is the best justification for using the middle bits of an address as the set index into a cache rather than the most significant bits?

   (a) Indexing with the most significant bits would necessitate a smaller cache than is possible with middle-bit indexing, resulting in generally worse cache performance.

   (b) It is impossible to design a system that uses the most significant bits of an address as the set index.

   (c) The process of determining whether a cache access will result in a hit or a miss is faster using middle-bit indexing.

   (d) A program with good spatial locality is likely to make more efficient use of the cache with middle-bit indexing than with high-bit indexing.

4. Which of the following is not a benefit of virtual memory?

   (a) It allows the virtual address space to be larger than the physical address space

   (b) No process can accidentally access the memory of another process

   (c) The TLB is more effective since without it dereferencing a virtual address now requires two or more memory accesses

   (d) Different processes can have overlapping virtual address spaces without conflict

10. Which of the following are copied on fork and preserved on exec?

    (a) Global variables.
    (b) File descriptor tables.
    (c) Open file entry structs.
    (d) None of the above.

12. What section of memory holds the assembly for `printf`?

    (a) Stack
    (b) Kernel memory
    (c) Shared libraries
    (d) Heap

13. Every thread has its own _____

    (a) Heap
    (b) Global values
    (c) Stack
    (d) Text data

16. What is the function of the TLB?

    (a) Caches data
    (b) Caches instructions
    (c) Caches translation of virtual addresses
    (d) Translates physical addresses to virtual addresses

17. What is distinctive about superscalar processors?

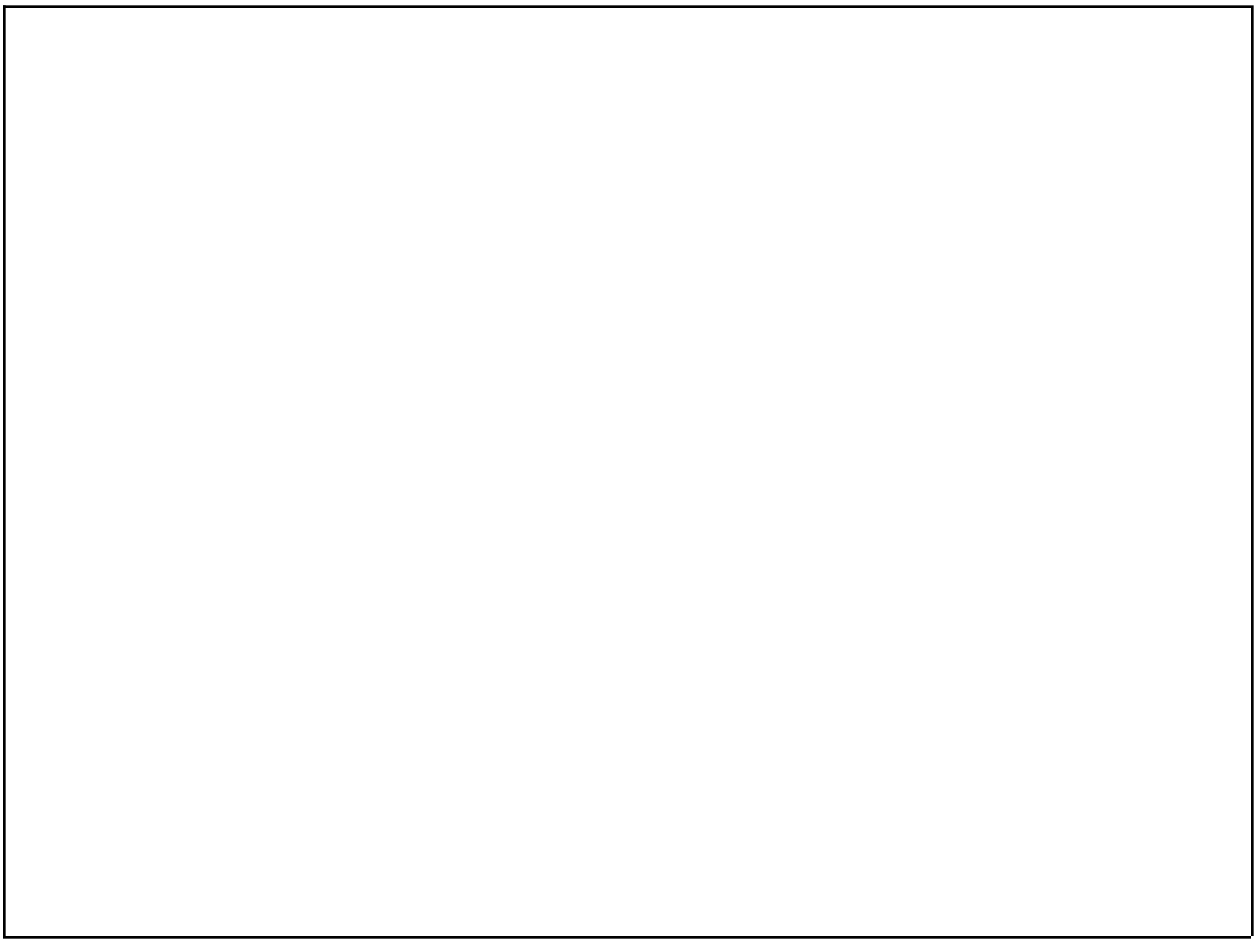
    (a) Can run at frequencies over 3.5GHz
    (b) Can address over 4GB of memory
    (c) Can perform more than one instruction per cycle
    (d) Can have more than 2 levels of cache
    (e) Have more than one core per processor

23. A 256-byte 4-way set associative cache with 16 byte blocks has

    (a) 4 sets
    (b) 16 sets
    (c) 64 sets
    (d) No sets

24. Imagine a floating point format with no sign bit, one exponent bit, and one fraction bit. Which of the following is not a number?

    (a) 00
    (b) 01
    (c) 10
    (d) 11
    (e) None of the above



Problem 7. (14 points):

Deadlocks and Dreadlocks

Two threads (X and Y) access shared variables A and B protected by mutex_a and mutex_b respectively. Assume all variable are declared and initialized correctly.

```
Thread X                    Thread Y
P(&mutex_a);                P(&mutex_b);
A += 10;                    B += 10;
P(&mutex_b);                P(&mutex_a);
B += 20;                    A += 20;
V(&mutex_b);                V(&mutex_a);
A += 30;                    B += 30;
V(&mutex_a);                V(&mutex_b);
```

A. Show an execution of the threads resulting in a deadlock. Show the execution steps as follows

Thread X	**Thread Y**
$P(\&mutex\_a)$	
$A+=10$	
$P(\&mutex\_b)$	
	$P(\&mutex\_b)$
...	
	...

Answer:

B. There are different approaches to solve the deadlock problem. Modify the code above to show **two** approaches to prevent deadlocks. You can declare new mutex variables if required. Do not change the order or amount of the increments to A and B. Rather, change the locking behavior around them. The final values of A and B must still be guaranteed to be incremented by 60.

Answer:

Problem 1. (20 points):

Short answer and multiple choice questions on a variety of stimulating and refreshing topics.

2. Which of the following is NOT a universal property of reader-writer locks?

   (a) Readers can only look at a shared item; writers can also modify it.

   (b) If a writer has access to the item, then no other thread also has access.

   (c) Any number of readers can read the item at the same time.

   (d) A writer waiting for an RW lock will get preference over subsequent read requests.

3. Starvation (in relation to threads) refers to:

   (a) A thread waiting for a lock indefinitely.

   (b) A semaphore that gets locked but the thread never unlocks it after use.

   (c) A thread is spawned but never joins the main thread when finished.

   (d) A process fails to spawn a new thread because it's hit the maximum number of threads allowed.

4. How does x86 assembly store the return value when a function is finished?

   (a) The `ret` instruction stores it in a special retval register.

   (b) By convention, it is always in `%eax`.

   (c) It is stored on the stack just above the `(%ebp)` of the callee.

   (d) It is stored on the stack just above all the arguments to the function.

5. In IEEE floating point, what would be an effect of allocating more bits to the exponent part by taking them from the fraction part?

   (a) You could represent fewer numbers, but they could be much larger.

   (b) You could represent the same numbers, but with more decimal places.

   (c) You could represent both larger and smaller numbers, but with less precision.

   (d) Some previously representable numbers would now round to infinity

6. Consider the following two blocks of code, found in *separate files*:

```
/* main.c */
int i=0;
int main()
{
  foo();
  return 0;
}
```

```
/* foo.c */
int i=1;
void foo()
{
  printf(''%d'', i);
}
```

What will happen when you attempt to compile, link, and run this code?

(a) It will fail to compile.

(b) It will fail to link.

(c) It will raise a segmentation fault.

(d) It will print "0".

(e) It will print "1".

(f) It will sometimes print "0" and sometimes print "1".

7. Which of the following is an example of external fragmentation?

(a) A malloc'ed block needs to be padded for alignment purposes.

(b) A user writes data to a part of the heap that isn't the payload of a malloc'ed block.

(c) There are many disjoint free blocks in the heap.

(d) A user malloc's some heap space and never frees it.

9. Which of the following is FALSE concerning x86-64 architecture?

   (a) A `double` is 64 bits long.
   (b) Registers are 64 bits long.
   (c) Pointers are 64 bits long.
   (d) Pointers point to locations in memory that are multiples of 64 bits apart.

10. Consider the following block of code:

```
int main()
{
  int a[213];
  int i;
  //int j = 15;
  for(i = 0; i < 213; i++)
  a[i] = i;
  return 0;
  a[0] = -1;
}
```

Which of the following instances of 'bad style' is present?

   (a) Dead code.
   (b) Magic numbers.
   (c) Poor indentation.
   (d) All of the above.

11. Consider the following structure declarations on a 64-bit Linux machine.

```
struct RECORD {
    long value2;
    double value;
    char tag[3];
};

struct NODE {
    int ref_count;
    struct RECORD record;
    union {
        double big_number;
        char string[12];
    } mix;
};
```

Also, a global variable named `my_node` is declared as follows:

```
struct NODE my_node;
```

If the address of `my_node` is `0x6008e0`, what is the value of `&my_node.record.tag[1]` ?

(a) `0x6008f8`
(b) `0x6008fa`
(c) `0x6008f9`
(d) `0x6008f5`
(e) `0x6008f1`

12. With reference to the previous question, what is the size of `my_node` in bytes ?

(a) 48
(b) 44
(c) 40
(d) 42
(e) 50

13. Which of the following x86 instructions can be used to add two registers and store the result without overwriting either of the original registers?

(a) `mov`
(b) `lea`
(c) `add`
(d) None of the above

14. Which of these uses of caching is not crucial to program performance?

    (a) Caching portions of physical memory
    (b) Caching virtual address translations
    (c) Caching virtual addresses
    (d) Caching virtual memory pages
    (e) None of the above (that is, they are all crucial)

17. In malloclab, we provided code for an implicit list allocator (the naive implementation). Many students improved this code by creating an explicit linked list of free blocks. Which of the following reason(s) explain(s) why an explicit linked list implementation has better performance?

    I. Immediate coalescing when freeing a block is significantly faster for an explicit list

    II. The implicit list had to include every block in the heap, whereas the explicit list could just include the free blocks, making it faster to find a suitable free block.

    III. Inserting a free block into an explicit linked list is significantly faster since the free block can just be inserted at the front of the list, which takes constant time.

    (a) I only.
    (b) II only.
    (c) III only.
    (d) II and III only.
    (e) All I, II and III.

18. Suppose a local variable `int my_int` is declared in a function named `func`. Which of the following is considered safe in C?

    (a) `func` returns `&my_int` and the caller dereferences the returned pointer.
    (b) `func` returns `&my_int` and the caller prints the returned pointer to the screen
    (c) `func` sets the value of a global variable to `&my_int` and returns. The global variable is unchanged up to the point another function dereferences the global variable.
    (d) None of the above

Problem 7. (7 points):

Cache memories. In this problem, we will consider the performance of the cache. You can make the following assumptions:

- There's only one level of cache.
- Block size is 4 bytes.
- The cache has 4 sets.
- Each cache set has two lines.
- Replacement policy is LRU.

Consider the following function which sets a 4×4 square in the upper left corner of an array to zero. You should assume that only operations involving `array` change the cache, that `array[0][0]` is at address `0x1000000`, and that the cache is empty when `clear4x4` is called.

```
#define LENGTH 8

void clear4x4(char array[LENGTH][LENGTH]){
    int row, col;
    for(col = 0; col < 4; col++){
        for(row = 0; row < 4; row++){
            array[row][col] = 0;
        }
    }
}
```

A. (3 pts) How many cache misses will there by when `clear4x4` is called?

Number of cache misses: _____

B. (3 pts) If `LENGTH` is changed to 16 how many cache misses will `clear4x4` have when called?

Number of cache misses: _____

C. (1 pt) If `LENGTH` is changed to 17, will calling `clear4x4` have a larger, smaller, or equal number of cache misses than when `LENGTH` is 16? Circle the correct answer.

- 16×16 will have MORE misses than 17×17.
- 16×16 and 17×17 will have an EQUAL number of MISSES.
- 17×17 will have MORE misses than 16×16.

Problem 8. (6 points):

Processes vs. threads. This problem tests your understanding of the some of the important differences between processes and threads. Consider the following C program:

```
#include "csapp.h"

/* Global variables */
int cnt;
sem_t mutex;


/* Helper function */
void *incr(void *vargp)
{
    P(&mutex);
    cnt++;
    V(&mutex);
    return NULL;
}
```

```
int main()
{
    int i;
    pthread_t tid[2];

    sem_init(&mutex, 0, 1); /* mutex=1 */

    /* Processes */
    cnt = 0;
    for (i=0; i<2; i++) {
        incr(NULL);
        if (fork() == 0) {
            incr(NULL);
            exit(0);
        }
    }
    for (i=0; i<2; i++)
        wait(NULL);
    printf("Procs:   cnt = %d\n", cnt);

    /* Threads */
    cnt = 0;
    for (i=0; i<2; i++) {
        incr(NULL);
        pthread_create(&tid[i], NULL, incr, NULL);
    }
    for (i=0; i<2; i++)
        pthread_join(tid[i], NULL);
    printf("Threads: cnt = %d\n", cnt);

    exit(0);
}
```

A. What is the output of this program?

```
Procs:   cnt = ___

Threads: cnt = ___
```