Andrew login ID: _____

Full Name: _____

CS 15-213, Fall 2005

Final Exam

Friday Dec 16, 2005

- Make sure that your exam is not missing any sheets, then write your full name and Andrew login ID on the front.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 92 points.
- This exam is OPEN BOOK. You may use any books or notes you like. You may use a calculator, but no other electronic devices are allowed. Good luck!

01 (06):	
02 (12):	
03 (08):	
04 (10):	
05 (12):	
06 (12):	
07 (12):	
08 (08):	
09 (08):	
10 (04):	
TOTAL (92):	

Problem 1. (6 points):

Address spaces. Suppose you have a computer system with:

- A 1 GB byte-addressable virtual address space,
- A 256 MB byte-addressable physical address space, and
- A virtual memory page size of 4 KB.

A. What is the minimum number of address bits needed to represent the virtual address space? _____.

B. What is the minumum number of bits needed to represent the physical address space? _____

C. What is the total number of page table entries? _____
(Express your answer in the form 2^x).

Problem 2. (12 points):

Data representation. Consider the following two 9-bit floating point representations based on the IEEE floating point format.

1. Format A
   - There is one sign bit.
   - There are $k = 5$ exponent bits. The exponent bias is 15.
   - There are $n = 3$ fraction bits.
2. Format B
   - There is one sign bit.
   - There are $k = 4$ exponent bits. The exponent bias is 7.
   - There are $n = 4$ fraction bits.

Numeric values are encoded in both of these formats as a value of the form $V = (-1)^S \times M \times 2^E$, where S is the sign bit, E is exponent after biasing, and M is the significand value. The fraction bits encode the significand value M using either a denormalized (exponent field 0) or a normalized representation (exponent field nonzero).
Below, you are given some bit patterns in Format A, and your task is to convert them to the closest value in Format B. If rounding is necessary you should *round toward* $+\infty$. In addition, give the values of numbers given by the Format A and Format B bit patterns. Give these as whole numbers (e.g., 17) or as fractions (e.g., $17/64$ or $17/2^6$).

Format A		Format B	
Bits	Value	Bits	Value
`1 01111 001`	$-9/8$	`1 0111 0010`	$-9/8$
`0 10110 011`			
`1 00111 010`			
`1 11100 000`			
`0 10111 100`			

Problem 3. (8 points):

Array indexing. Consider the source code below, where `M` and `N` are constants declared with `#define`.

```
int array1[M][N];
int array2[N][M];

int copy(int i, int j)
{
    array1[i][j] = array2[j][i];
}
```

The above code generates the following assembly code on a 64-bit Pentium:

```
  Arguments: i is in %edi, j is in %esi
copy:
  movslq  %esi,%rsi
  movslq  %edi,%rdi
  leaq    0(,%rsi,8), %rax
  leaq    (%rdi,%rdi,4), %rdx
  subq    %rsi, %rax
  addq    %rsi, %rdx
  addq    %rdi, %rax
  movl    array2(,%rax,4), %eax
  movl    %eax, array1(,%rdx,4)
  ret
```

Assuming that `sizeof(int) == 4`, what are the values of `M` and `N`?

`M` =

`N` =

Problem 4. (10 points):

Machine-level code. Consider the following function's assembly code:

```
00000000004004f8 <foo>:
  4004f8: 53                    push   %rbx
  4004f9: 89 f8                 mov    %edi,%eax
  4004fb: 83 ff 01              cmp    $0x1,%edi
  4004fe: 76 21                 jbe    400521 <foo+0x29>
  400500: b8 01 00 00 00        mov    $0x1,%eax
  400505: b9 00 00 00 00        mov    $0x0,%ecx
  40050a: ba 02 00 00 00        mov    $0x2,%edx
  40050f: 39 fa                 cmp    %edi,%edx
  400511: 77 0e                 ja     400521 <foo+0x29>
  400513: 01 c8                 add    %ecx,%eax
  400515: 89 c3                 mov    %eax,%ebx
  400517: 29 cb                 sub    %ecx,%ebx
  400519: 89 d9                 mov    %ebx,%ecx
  40051b: ff c2                 inc    %edx
  40051d: 39 fa                 cmp    %edi,%edx
  40051f: 76 f2                 jbe    400513 <foo+0x1b>
  400521: 5b                    pop    %rbx
  400522: c3                    retq
```

Please fill in the corresponding C code:

```
int foo (unsigned int x)
{
  int a, b, i;

  if(__________)
    _____________;

  a = 1;
  b = 0;

  for(_______ ; ________ ; _______)
  {
    a = _____________;
    b = _____________;
  }

  return _____;
}
```

Problem 5. (12 points):

Performance Evaluation. We saw in class how loop unrolling can be used to improve the performance of a piece of code. This problem will test your ability to analyze the performance improvements offered by this technique.

Assume that multiplication has a latency of 7 cycles and addition has a latency of 5 cycles.

A. Alice has written the code below to compute the dot product of two vectors, computing one element per iteration.

```
data_t dot_prod(data_t A[], data_t B[], int size)
{
    data_t result = 0;
    int i;
    for (i = size-1; i >= 0; i--) {
      result = result + (A[i] * B[i]);
    }
    return result;
}
```

What is the optimal CPE achieved by the code above? Assume that there are an unlimited number of functional units.

CPE = _____

B. Now suppose Alice unrolls the loop, computing two elements per iteration. What is the resulting optimal CPE? Once again, assume that there are an unlimited number of functional units.

```
/* Unroll 2x */
data_t dot_product2(data_t A[], data_t B[], int size)
{
    data_t result = 0;
    int i;
    /* Unroll by 2X */
    for (i = size-1; i >= 1; i -= 2) {
      data_t t1 = A[i] * B[i];
      data_t t2 = A[i-1] * B[i-1];
      result = result + (t1 + t2);
    }

    /* Finish off remaining element(s) */
    for (; i >= 0; i -= 1) {
        result = result + (A[i] * B[i]);
    }
    return result;
}
```

CPE = _____

C. By what factor would Alice need to unroll the loop to get an optimal CPE of 0.5? Once again, assume that there are an unlimited number of functional units.

Unroll by _____

D. One of the reasons why we get diminishing returns from unrolling on a real CPU is that there are a limited number of functional units, and therefore, a limit to how many operations we can perform in parallel. So Alice is very excited to learn that the CS department is getting a new machine with one million floating point units. In anticipation, she unrolls her dot product code by 10,000. Give one reason why her new code may actually perform worse than her old code (which unrolled by 2).

You may assume that memory latencies and cache sizes on the new machine are the same as on the old one.

_____

Problem 6. (12 points):

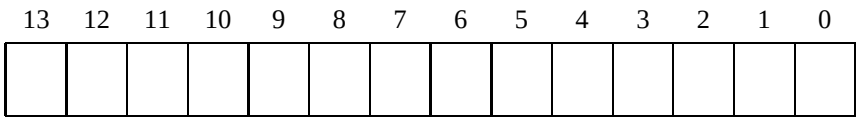
Cache memories. This problem requires you to analyze both high-level and low-level aspects of caches. You will be required to perform part of a a cache translation, determine individual hits and misses, and analyze overall cache performance.

- Memory is byte addressable
- Physical addesses are 14 bits wide
- The cache is direct-mapped with a 16 byte block-size and 4 sets
- sizeof(int) = 4 bytes

A. The following question will deal with a 5×5 int matrix $arr[5][5]$. Assume that the array has already been initialized.

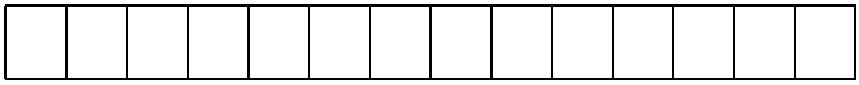
(a) The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

`CO` The block offset within the cache line
`CI` The set index
`CT` The cache tag



(b) Given the address of $arr[0][0]$ has value **0x2A6C**, perform a cache address translation to determine the block offset and set index for the first item in the array.

`CI = _____`
`CO = _____`



(c) For each element in the matrix, label the diagram below with the set index that it will map to.

arr	col 0	col 1	col 2	col 3	col 4
row 0					
row 1					
row 2					
row 3					
row 4					

B. The following questions will also deal with the 5×5 matrix arr and the cache defined at the beginning of the problem. Assume that the cache begins cold and then the following five cells in the matrix are accessed, in the order they are listed:

```
arr[0][0] arr[1][1] arr[2][2] arr[3][3] arr[4][4]
```

The following four cells are then accessed, in order. You need to determine if each access will be a hit (**H**) or a miss (**M**). Remember that the cache is not cold but is setup as it would be after accessing the elements above. Also remember that the elements below are also being accessed in order and the contents of the cache should change following a miss. **This part will be graded based on the set mapping in (c) above so even if you aren't confident in your mapping you can still get full credit for this section (assuming your mapping is rational).**

```
arr[1][2] = _______

arr[0][0] = _______

arr[4][0] = _______

arr[0][3] = _______
```

The following region can be used as scrap space:

C. The following questions will deal with an $N \times N$ int matrix arr except that $N = 1024$. Assuming i, j, and sum are all stored in registers, analyze the cache performance of the following piece of code:

```
#define N 1024

int arr[N][N];

int arr_sum()
{
  int i, j;
  int sum = 0;

  for(i=0; i<N; i++){
    for(j=0; j<N; j++){
      sum += arr[i][j];
    }
  }

  return sum;
}
```

Assume that the matrix is aligned so that arr[0][0] is the first element in a cache block. For each of the following caches, calculate the **miss percentage** for the function above.

(a) Direct-mapped, 16 byte block-size, 4 sets

_____%

(b) 2-way set associative, 8 byte block-size, 4 sets

_____%

(c) Direct-mapped, 32 byte block-size, 1 set

_____%

Problem 7. (12 points):

Process control.

A. How many lines of output does the following function print. Give your answer as a function of n. Assume $n \geq 1$.

```
void foo(int n) {
    int i;

    for (i = 0; i < n; i++) {
        fork();
    }
    printf("Hi there!\n");
}
```

Number of lines of output = _____

B. List all of the possible output sequences for the following program:

```
int main() {
    if (fork() == 0) {
        printf("a");
    }
    else {
        printf("b");
        waitpid(-1, NULL, 0);
    }
    printf("c");
    exit(0);
}
```

Possible output sequences:

C. Consider the following program:

```
pid_t pid;
int counter = 0;

void handler1(int sig)
{
    counter += 2;
    printf("counter = %d\n", counter);
    fflush(stdout);   /* Flush the printed string to stdout */
    kill(pid, SIGUSR1);
}

void handler2(int sig)
{
    counter -= 2;
    printf("counter = %d\n", counter);
    exit(0);
}

int main() {
    signal(SIGUSR1, handler1);
    if ((pid = fork()) == 0) {
        signal(SIGUSR1, handler2);
        kill(getppid(), SIGUSR1); /* Send SIGUSR1 to parent */
        while(1) {};
    }
    else {
        waitpid(-1, NULL, 0);
        counter += 2;
        printf("counter = %d\n", counter);
    }
    exit(0);
}
```

What is the output of this program?

Problem 8. (8 points):

Storage allocators.

This problem tests your understanding of pointer arithmetic, pointer dereferencing, and malloc implementation.

Harry Q. Bovik has implemented a simple explicit-list allocator. You may assume that his implementation follows the usual restrictions that you had to comply with in L6, such as the 8-byte alignment rule.

The following is a description of Harry's block structure:

HDR	PAYLOAD	FTR

- HDR - Header of the block (4 bytes)
- PAYLOAD - Payload of the block (arbitrary size)
- FTR - Footer of the block (4 bytes)

The size of the **payload** of each block is stored in the header and the footer of the block. Since there is an 8-byte alignment requirement, the least significant of the 3 unused bits is used to indicate whether the block is free (0) or allocated (1). .

For the first part of the problem, you can assume that:

- sizeof(int) == 4 bytes
- sizeof(char) == 1 bytes
- sizeof(short) == 2 bytes
- sizeof(long) == 4 bytes
- The size of any pointer (e.g. char*) is 4 bytes.

Note that he is working on a 32-bit machine. Also, assume that the block pointer `bp` points to the first byte of the payload.

A. Harry realizes that he can utilize more memory by exploiting the internal structure of his block. He notices that his malloc may allocate more memory than the requested size for various reasons, one of which is to comply with 8-byte alignment rule.

Your task is to help Harry figure out and indicate which of the following codes will cause `malloc_hack()` to return the actual size of payload.

For each of the proposed solutions listed below, fill in the blank with either **C** for correct, or **I** for incorrect.

```
/* malloc_hack1 returns the actual size of payload.
   bp is pointing to the first byte of a block
   returned from Harry's malloc() */

int malloc_hack(void *bp)
{
        return ____________________________ & ~0x7;
}


/* A. */
   *(int *)((int *)bp - 1)          ________

/* B. */
   *(int *)((char *)bp - 1)         ________

/* C. */
   *(int *)((char **)bp - 1)        ________

/* D. */
   *(char *)((int)bp - 1)           ________

/* E. */
   *(long *)((long *)bp - 1)        ________

/* F. */
   *(int *)((int)bp - 4)            ________

/* G. */
   *(int *)((short)bp - 2)          ________

/* H. */
   *(short *)((int *)bp - 1)        ________
```

B. Harry now wants to port his `malloc_hack` to 64-bit machine. On 64-bit machines,

- sizeof(long) == 8 bytes
- The size of any pointer (e.g. char*) is 8 bytes.

and the other types remain the same. As before, for each proposed solution, fill in the blank either **C** for correct, or **I** for incorrect.

```
int malloc_hack(void *bp)
{
        return ____________________________ & ~0x7;
}


/* A. */
   *(int *)((int *)bp - 1)          ________

/* B. */
   *(int *)((char *)bp - 1)         ________

/* C. */
   *(int *)((char **)bp - 1)        ________

/* D. */
   *(char *)((int)bp - 1)           ________

/* E. */
   *(long *)((long *)bp - 1)        ________

/* F. */
   *(int *)((int)bp - 4)            ________

/* G. */
   *(int *)((short)bp - 2)          ________

/* H. */
   *(short *)((int *)bp - 1)        ________
```

Problem 9. (8 points):

Concurrency and sharing. Consider a concurrent C program with two threads, where each thread executes the following line of code:

```
/* Increment the shared global variable cnt */
cnt = cnt + 1;
```

Suppose that this line of C code compiles to the following assembly language instructions:

```
movl cnt,%eax   # L: Load cnt
inc  %eax       # U: Update cnt
movl %eax,cnt   # S: Store cnt
```

At runtime, the operating system kernel will choose some ordering of these instructions. Since we are not explicitly synchonizing the threads, some of these orderings will produce the correct value for cnt and others will not.

Let L_i denote the execution of instruction L by thread i; similarly for U_i and S_i. Each of the sequences shown below gives a possible ordering of the instructions when the two threads execute. Assuming that `cnt` is initially zero, what is the value of `cnt` in memory after each of the sequences completes?

A. cnt=0; $L_1, U_1, S_1, L_2, U_2, S_2$ `cnt ==` _____

B. cnt=0; $L_1, U_1, L_2, S_1, U_2, S_2$ `cnt ==` _____

C. cnt=0; $L_2, U_2, S_2, L_1, U_1, S_1$ `cnt ==` _____

D. cnt=0; $L_1, L_2, U_1, S_1, U_2, S_2$ `cnt ==` _____

Problem 10. (4 points):

Thread synchronization. Consider the following classic producer-consumer system, where producer threads add items to the rear a shared queue and consumer threads remove items from the front of that queue.

```
/* Initialize semaphores */
mutex = 1;
slots = N;
items = 0;
```

```
Producer Thread Routine

while (1) {
    P(slots);
    P(mutex);
    [Insert item at rear]
    V(mutex);
    V(items);
}
```

```
Consumer Thread Routine

while (1) {
    P(items);
    P(mutex);
    [Remove item from front]
    V(mutex);
    V(slots);
}
```

We would like to add another kind of thread to this system, called a *reader thread*, that simply reads the front item in the queue without removing it. Add the appropriate synchronization statments that will allow reader threads to safely read the front item in the shared buffer. Your solution should work for an arbitrary number of reader threads. For this solution it is OK to limit the number of concurrent read operations to one. It is also OK to peek at an empty queue. (Hint: this is not a hard problem.)

```
Reader Thread Routine: Peek without removing

/* Initialize additional semaphore (if any) here */

__________ = ________;


while (1) {

    _________________;   /* Add synchronization statement here */

    [Read operation: Peek at front item without removing]

    _________________;  /* Add synchronization statement here */
}
```