Andrew login ID:_____

Full Name:_____

CS 15-213, Fall 2006

Exam 2

Wednesday Nov 15, 2006

Instructions:

- 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 55 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. Calculators are allowed, but no other electronic devices. Good luck!

1 (11):
2 (08):
3 (08):
4 (12):
5 (08):
6 (08):
TOTAL (55):

Problem 1. (11 points):

Consider the following C function:

```
data_t psum(data_t a[], data_t b[], data_t c[], int cnt)
{
    data_t r = 0;
    int i;
    for (i = 0; i < cnt; i++) {
        /* Inner loop expression */
        r =   r  +  a[i]  *  b[i]  +  c[i]  ;
    }
    return r;
}
```

In this code, data type `data_t` can be defined to different types using a `typedef` declaration.

According to the C rules for operator precedence and associativity, the line labeled “Inner loop expression” will be computed according to the following parenthesization:

```
        r =  (r  + (a[i]  *  b[i]))+  c[i]  ;
```

In all, there are 5 different parenthesizations for this expression, They will not all compute the same result.

Imagine we run this code on a machine in which multiplication requires 7 cycles, while addition requires 5. Assume that these latencies are the only factors constraining the performance of the program. Don't worry about the cost of memory references or integer operations, resource limitations, etc.

A. For each parenthesization listed below, write down the CPE that the function would achieve. **Hint:** All of your answers will be in the set $\{5, 7, 10, 12, 14, 15, 17, 19\}$.

```
// P1.  CPE =
r = ((r  +  a[i]) *  b[i]) +  c[i]  ;

// P2.  CPE =
r =  (r  + (a[i]  *  b[i]))+  c[i]  ;

// P3.  CPE =
r =   r  +((a[i]  *  b[i]) +  c[i]) ;

// P4.  CPE =
r =  (r  +  a[i]) * (b[i]  +  c[i]) ;

// P5.  CPE =
r =   r  + (a[i]  * (b[i]  +  c[i]));
```

B. Of the parenthesizations that give the same result as the original function, which has the best CPE? Assume that addition and multiplication are associative for data type `data_t`.

Problem 2. (8 points):

This problem requires you to analyze the cache behavior of a function that sums the elements of an array A:

```
int A[2][4];

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

    for (j=0; j<4; j++) {
        for (i=0; i<2; i++) {
            sum += A[i][j];
        }
    }
    return sum;
}
```

Assume the following:

- The memory system consists of registers, a single L1 cache, and main memory.
- The cache is cold when the function is called and the array has been initialized elsewhere.
- Variables i, j, and sum are all stored in registers.
- The array A is aligned in memory such that the first two array elements map to the same cache block.
- `sizeof(int) == 4`.
- The cache is direct mapped, with a block size of 8 bytes.

A. Suppose that the cache consists of 2 sets. Fill out the table to indicate if the corresponding memory access in A will be a hit (**h**) or a miss (**m**).

A	Col 0	Col 1	Col 2	Col 3
Row 0	**m**			
Row 1				

B. What is the pattern of hits and misses if the cache consists of 4 sets instead of 2 sets?

A	Col 0	Col 1	Col 2	Col 3
Row 0	**m**			
Row 1				

Problem 3. (8 points):

This problem tests your understanding of the the cache organization and performance. Assume the following:

1. `sizeof(int) = 4`
2. Array `x` begins at memory address 0.
3. The cache is initially empty.
4. The only memory accesses are to the entries of the array `x`. All variables are stored in registers.

Consider the following C code:

```
int x[128];
int i, j;
int sum = 0;

for (i = 0; i < 64; i ++){
    j = i + 64;
    sum += x[i] * x[j];
}
```

Case 1

1. Assume your cache is a 256-byte direct-mapped data cache with 8-byte cache blocks. What is the cache **miss rate**? (2 pts)

miss rate = _____%

2. If the cache were twice as big, what would be the miss rate? (1 pts)

miss rate = _____%

Case 2

1. Assume your cache is 256-byte 2-way set associative using an LRU replacement policy with 8-byte cache blocks. What is the cache miss rate? (3 pts)

miss rate = _____%

2. Will larger **cache size** help to reduce the miss rate? (Yes / No) (1 pt)

3. Will larger **cache line** help to reduce the miss rate? (Yes / No) (1 pt)

Problem 4. (12 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 16 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	16	13	1
	1B	2D	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0
4	06	08	1
	0F	19	1
5	0A	09	1
	1F	20	1
6	02	13	0
	18	12	1
7	0C	0B	0
	1E	24	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	2D	0
05	13	1	15	1B	0
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	1
0C	01	1	1C	1E	0
0D	15	1	1D	3E	1
0E	0C	0	1E	27	1
0F	14	0	1F	18	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) What is the total number of page table entries? _____

B. Virtual Address Translation I

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

Virtual address: `0xFAA3F`

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).

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

(Please go to the next page for part C)

C. Virtual Address Translation II

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

Virtual address: `0x162A4`

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).

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Problem 5. (8 points):

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 - 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).

This problem consists of two parts.

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

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

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

Part One. Your task is to help Harry compute the correct payload size (using the function `get_payload_size()`), by indicating which of the following implementations of the `GET_HDR` macro are corect. For each of the proposed solutions listed below, fill in the blank with either **C** for correct, or **I** for incorrect.

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

#define GET_HDR(p)       ??
#define GET_SIZE(p)      (GET_HDR(p) & ~0x7)

int get_payload_size(void *bp)
{
    return (int)(GET_SIZE(bp));
}

/* (1) */
   #define GET_HDR(p)  (*(int *)((int *)(p) - 1))    _________

/* (2) */
   #define GET_HDR(p)  (*(int *)((char *)(p) - 1))   _________

/* (3) */
   #define GET_HDR(p)  (*(int *)((char **)(p) - 1))  _________

/* (4) */
   #define GET_HDR(p)  (*(char *)((int)(p) - 1))     _________

/* (5) */
   #define GET_HDR(p)  (*(long *)((long *)(p) - 1)   _________

/* (6) */
   #define GET_HDR(p)  (*(int *)((int)(p) - 4))      _________

/* (7) */
   #define GET_HDR(p)  (*(int *)((short)(p) - 2))    _________

/* (8) */
   #define GET_HDR(p)  (*(short *)((int *)(p) - 1))  _________
```

Part Two. Harry now wants to port his `GET_HDR` macro to a 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 blanks with either **C** for correct, or **I** for incorrect.

```
#define GET_HDR(p)   ??
#define GET_SIZE(p)  (GET_HDR(p) & ~0x7)

int get_payload_size(void *bp)
{
    return (int)(GET_SIZE(bp));
}

/* (1) */
   #define GET_HDR(p)  (*(int *)((int *)(p) - 1))    ________

/* (2) */
   #define GET_HDR(p)  (*(int *)((char *)(p) - 1))   ________

/* (3) */
   #define GET_HDR(p)  (*(int *)((char **)(p) - 1))  ________

/* (4) */
   #define GET_HDR(p)  (*(char *)((int)(p) - 1))     ________

/* (5) */
   #define GET_HDR(p)  (*(long *)((long *)(p) - 1)   ________

/* (6) */
   #define GET_HDR(p)  (*(int *)((int)(p) - 4))      ________

/* (7) */
   #define GET_HDR(p)  (*(int *)((short)(p) - 2))    ________

/* (8) */
   #define GET_HDR(p)  (*(short *)((int *)(p) - 1))  ________
```

Problem 6. (8 points):

This question will test your understanding of Unix process control and signals. Consider the following C program. For space reasons, we are not checking error return codes. You can assume that all functions return normally.

```
int val = 3;
void Exit(int val) {
    printf("%d", val);
    exit(0);
}

void usr1_handler(int sig) {
    Exit(val);
}

int main() {
    int pid;

    signal(SIGUSR1, usr1_handler);

    if ((pid = fork()) == 0) {
        setpgid(0, 0);

        if (fork())
            Exit(val + 1);
        else
            Exit(val - 1);
    }

    kill(-pid, SIGUSR1);
}
```

List all the outputs possible from this program:

_____ _____ _____ _____ _____

_____ _____ _____ _____ _____

_____ _____ _____ _____ _____