Sungkyunkwan University
School of Information and Communication Engineering

CSE2003: System Programming
Final Exam. (Spring 2009)

3:00PM - 5:00PM, June 17, 2009. Instructor: Jin-Soo Kim

Student ID: _____

Name: _____

Q1 (25)		Q5 (60)	
Q2 (30)		Q6 (50)	
Q3 (30)		Q7 (60)	
Q4 (45)			
		Total (300)	

1. Write the full name of the following acronym. (25 points)

(1) GNU ( )

(2) DRAM ( )

(3) ELF ( )

(4) MMU ( )

(5) ISA ( )

2. Which of the followings are the characteristics of CISC? Choose all that apply. (30 points. Do not take guesses. Each wrong answer will be penalized with -5 points.)

A. Fixed length encodings

B. Registers are used for procedure arguments and return addresses

C. A large number of instructions

D. Special flags are set as a side effect of instructions and then used for conditional branch testing.

E. Some instructions with long execution times

F. Arithmetic and logical operations only use register operands

G. Multiple formats for specifying operands

3. Generating position-independent code (PIC) is based on the following interesting fact: No matter where we load an object module (including shared object modules) in memory, the data segment is always allocated immediately after the code segment. Thus, the *distance* between any instruction in the code segment and any variable in the data segment is a run-time constant, independent of the absolute memory locations of the code and data segments. In order to exploit this fact, it is necessary to obtain the current value of the PC (program counter) at run time. Write x86 assembly code which puts the current value of the PC (= the contents of the **`%eip`** register) into the **`%ebx`** register. Note that the **`%eip`** register cannot be the source operand of the **`movl`** instruction. (30 points, no partial credit)

4. 3M decides to make Post-Its by printing yellow squares on white pieces of paper. As part of the printing process, they need to set the CMYK (cyan, magenta, yellow, black) value for every point in the square. 3M hires you to determine the efficiency of the following algorithms on a machine with a 2048-byte direct-mapped data cache with 32-byte blocks. You are given the following definitions and assumptions: (45 points)

```
struct point_color {
    int  c;
    int  m;
    int  y;
    int  k;
};
struct point_color square[16][16];
int i, j;
```

Assume the following:

- **sizeof(int) == 4**
- **square** begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array **square.** Variables **i** and **j** are stored in registers.

(1) Determine the cache performance of the following code:

```
for (i = 0; i < 16; i++) {
  for (j = 0; j < 16; j++)  {
    square[i][j].c = 0;
    square[i][j].m = 0;
    square[i][j].y = 1;
    square[i][j].k = 0;
  }
}
```

A. What is the total number of memory writes?

B. What is the total number of memory writes that miss in the cache?

C. What is the miss rate?

(2) Determine the cache performance of the following code:

```
for (i = 0; i < 16; i++) {
  for (j = 0; j < 16; j++)  {
    square[j][i].c = 0;
    square[j][i].m = 0;
    square[j][i].y = 1;
    square[j][i].k = 0;
  }
}
```

A. What is the total number of memory writes?

B. What is the total number of memory writes that miss in the cache?

C. What is the miss rate?

(3) Determine the cache performance of the following code:

```
for (i = 0; i < 16; i++)
  for (j = 0; j < 16; j++)
    square[i][j].y = 1;

for (i = 0; i < 16; i++) {
  for (j = 0; j < 16; j++)  {
    square[i][j].c = 0;
    square[i][j].m = 0;
    square[i][j].k = 0;
  }
}
```

A. What is the total number of memory writes?

B. What is the total number of memory writes that miss in the cache?

C. What is the miss rate?

5. Consider the following C program "**print.c**". What is the output of this program when it is executed on the IA-32/Linux system? Fill in each blank with a correct value and explain why. (60 points)

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

typedef unsigned char *ptr;
struct node {
    char        c;
    int         i;
    double      d;
    char        s[5];
} x;

void printb (ptr start, int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%.2x ", start[i]);
    printf("\n");
}
```

```
int main (void)
{
    x.c = '0';
    x.i = 0xcafebabe;
    x.d = 1.0;
    strcpy (x.s, "012");

    printf("0: ascii('0') = %.2x\n", x.c);
    printf("1: ");
    printb((ptr) &x.i, 1);
    printf("2: ");
    printb((ptr) &x.s[1], 3);
    printf("3: %d\n", sizeof(struct node));
    printf("4: %d\n", (ptr) &x - (ptr) &x.c);
    printf("5: %d\n", (ptr) &x.i - (ptr) &x.c);
    printf("6: %d\n", (ptr) &x.d - (ptr) &x.c);
    printf("7: %d\n", (ptr) &x.s[0] - (ptr) &x.c);
    printf("8: %d\n", (ptr) (&x+1) - (ptr) &x.s[0]);
}
```

```
linux> gcc -o print print.c
linux> ./print

0: ascii('0') = 30

1: _____

2: _____

3: _____

4: _____

5: _____

6: _____

7: _____

8: _____
```

6. A program consists of two files written in C language, "**foo.c**" and "**bar.c**", as shown below. When is each of the following values determined? Choose among four possible answers: A. compile time, B. static link time, C. dynamic link time, and D. run time. We assume that the function **printf()** belongs to a shared library. (50 points)

(1) The value of the underlined expression in **bar.c** (line #17)

(2) The address of the variable **ep**

(3) The address of the variable **m**

(4) The address of the variable **e**

(5) The address of the variable **z**

(6) The initial value of the variable **ep**

(7) The initial value of the variable **e**

(8) The address of the function **bar()**

(9) The address of the function **printf()**

(10) The address of the function **foo()**

```
1: /* foo.c */
2: #define MAX(a,b) (((a)>(b))? (a):(b))
3:
4: int e = MAX(9,15);
5:
6: int foo (int n)
7: {
8:     return n+1;
9: }
```

```
 1: /* bar.c */
 2: #include <stdio.h>
 3:
 4: extern int foo (int n);
 5: extern int e;
 6:
 7: int *ep = &e;
 8: int m;
 9:
10: int bar (int n)
11: {
12:     return n*n;
13: }
14: int main (void)
15: {
16:     int z;
17:     m = (1 << 24) - 1;
18:     z = bar(4) & m;
19:     z += foo(*ep);
20:     printf("%x %d\n", &z, z);
21: }
```

7. In the BMP image file format, each pixel is represented by three bytes: the first byte for blue, the second for green, and the third for red color. In addition, there is a restriction in the BMP format that the number of bytes occupied by each row should be a multiple of 4. We want to implement a function **bytes_per_row()** which returns the number of bytes occupied by each row in the given image. The function **bytes_per_row()** takes the width of the image as an argument. Complete the function **bytes_per_row()** in x86 assembly language, whose skeleton is given below. (60 points, no partial credit)

Restrictions:

(1) Do not use any integer multiplication or division instruction because it's an expensive operation.
(e.g., **imull, mull, idiv, div,** etc.)

(2) The code to be inserted should be less than or equal to 3 instructions long.

(3) The final result should be stored in the **%eax** register.

```
#include <stdio.h>

int bytes_per_row (int width);

void display (int width)
{
    int r;

    r = bytes_per_row (width);
    printf ("Bytes per row = %d\n",
        r);
}
```

```
.text
    .align  4
.globl  bytes_per_row
    .type bytes_per_row,@function

bytes_per_row:
    pushl   %ebp
    movl    %esp, %ebp

    movl    8(%ebp), %edx     # width is in %edx

    ____________________________

    ____________________________

    ____________________________

    movl    %ebp, %esp
    popl    %ebp
    ret
```