COMS W4115
Programming Languages and Translators
Homework Assignment 3

Prof. Stephen A. Edwards   Due July 22nd, 2011
Columbia University   at 11:59 PM your time

Submit solutions through the CVN website.

Do this assignment alone. You may consult the instructor and the TAs, but not other students.

1. For the following C array,

   ```
   int a[2][3];
   ```

   assume you are working with a 32-bit little-endian processor with the usual alignment rules (e.g., a Pentium) and

   (a) Show how its elements are laid out in memory.

   (b) Write the address expression for accessing `a[i][j]`.

   (c) Verify parts a) and b) by writing a small C program that contains and accesses such an array and looking at the assembly language output with the C compiler's -S flag (e.g., `gcc -O -S array.c`. Turn in a copy of your C program and an annotated version of the assembly listing. Make sure the assembly listing is no more than 40 lines.

2. In an assembly-language-like notation (e.g., use MIPS or a pseudocode of your own choosing), write what an optimizing compiler would produce for the following two `switch` statements.

   ```
   switch (a) {
   case 5:  x = 2; break;
   case 6:  x = 5; break;
   case 7:  x = 24; y = 11; break;
   case 8:  y = 8; break;
   case 9:  z = 3; break;
   default: z = 4; break;
   }

   switch (b) {
   case 5:    a = 18; break;
   case 73:   a = 2; break;
   case 105:  b = 7; c = 10; break;
   case 5644: c = 8; break;
   default:   c = 17; break;
   }
   ```

3. For a 32-bit little-endian processor with the usual alignment rules, show the **memory layout** and **size in bytes** of the following three C variables.

   ```
   union {
     short a;  /* 16-bit */
     struct {
       int b;  /* 32-bit */
       char c; /* 8-bit */
     } s;
   } u1;
   ```

   ```
   struct {              struct {
     short a;              int d;
     char b;               short a;
     short c;              short c;
     int d;                char b;
   } s1;                 } s2;
   ```

4. Consider the following C-like program.

   ```
   int w = 8;
   int x = 12;

   int incw() { return ++w; }
   int incx() { return ++x; }

   void foo(y, z){
     printf("%d\n", y + 1 + y);
     x = 4;
     printf("%d\n", z);
   }

   int main() {
     foo(incw(), incx()); return 0;
   }
   ```

   What does it print if the language uses

   (a) Applicative-order evaluation?

   (b) Normal-order evaluation?