

COMS W4115

Programming Languages and Translators

Homework Assignment 2

Prof. Stephen A. Edwards Due November 29th, 2007
Columbia University at 2:40 PM (beginning of class)

Submit solution on paper (no email). Please write your name clearly on the paper.

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

When you complete this assignment, please write down an estimate of the number of hours you spent on it. I am just collecting statistics; this will not affect your grade.

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 a good optimizing compiler would produce for the following two switch statements.

```
switch (a) {  
case 1: x = 3; break;  
case 2: x = 5; break;  
case 3: x = 8; y = 15; break;  
case 4: y = 20; break;  
case 5: z = 23; break;  
default: z = 28; break;  
}
```

```
switch (b) {  
case 1: a = 3; break;  
case 10: a = 5; break;  
case 100: b = 8 ; c = 15; break;  
case 1000: c = 20; break;  
default: c = 25; 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 C types.

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

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

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

4. Consider the following C-like program.

```
int w = 3;  
int x = 10;  
  
int incw() { return ++w; }  
int incx() { return ++x; }  
  
void foo(y, z){  
    printf("%d\n", y + y);  
    x = 1;  
    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?