COMS W4995-02
Languages for Embedded System Design
Homework 1

Prof. Stephen A. Edwards
Columbia University

Assigned September 9, 2002
Due September 18, 2002

You may submit the solutions either on paper or electronically, but not both. If you submit it electronically, send a single file that is text, PostScript, PDF, or Word. Don't send multiple files or an archive such as tar or zip. I just print out whatever you submit and read it; I don't try to run the programs.

Make sure your name appears at the beginning of the file you send.

I want the paper or electronic versions at the beginning of class (4:10 PM EDT) on the due date. This applies to both on-campus and CVN students.

1. **(5 points)** Book, Exercise 1-1: What is nondeterminism? How might nondeterminism arise? (give two examples) What are the advantages of nondeterminism in a software language? The disadvantages?

2. **(5 points)** Book, Exercise 6-2: Name two reasons RISC machines have largely replaced CISC processors. Name two reasons why you might still prefer a CISC processor.

3. **(30 points)** There are often many different ways to implement the same functionality in assembly language. We will illustrate this using the two C compilers available on the cunix.columbia.edu cluster: cc from Sun and gcc from the GNU project. While both produce code for the SPARC, they can produce different, although equally correct, results.

   More information about the SPARC instruction set can be found at www.sparc.org (look at the V8 architecture standard).

   Create a file containing the following C program. (You can type it manually or copy it from my cunix.columbia.edu account: `~se2007/hw1`.)

```
int euclid(int m, int n)
{
  int r;
  while ( (r = m % n) != 0 ) {
    m = n;
    n = r;
```

```
  }
  return n;
}
```

Ask the C compiler to produce assembly code with and without optimization:

```
cc -S hw1-3a.c
mv hw1-3a.s hw1-3a.sun.s
cc -O -S hw1-3a.c
mv hw1-3a.s hw1-3a.sun-O.s
gcc -S hw1-3a.c
mv hw1-3a.s hw1-3a.gcc.s
gcc -O -S hw1-3a.c
mv hw1-3a.s hw1-3a.gcc-O.s
```

Compare the four versions of the program. How does the output differ? What instructions have the two compilers chosen? Have the two compilers ordered instructions differently? What effect does the -O flag have on the output? Does it seem that one compiler does a better job optimizing than the other?

Add the following main function in a file called hw1-3b.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int count;
  int i, j;
  if (argc != 2) {
    fprintf(stderr,"Usage: %s max\n", argv[0]);
    return 1;
  }

  count = atoi(argv[1]);

  for ( i = 2 ; i < count ; i++ )
    for ( j = 2 ; j < count ; j++ )
      euclid(i,j);
  return 0;
}
```

Compile the two together and time the result

```
cc -o hw1-3.sun hw1-3a.c hw1-3b.c
time ./hw1-3.sun 500
```

Adjust the number of iterations (500 in this example) so it takes between 1 and 2 seconds. The goal here is to run the program run long enough to be easily measured, but no longer.

Compare the time it takes for that same number of iterations under all four combinations of compilers and optimizations. Run each a few times and average the result to get more accurate numbers. Report the times you measure.

Which compiler/optimization flag produced the fastest code? Can you see from the assembly source why this is?

(Hand in annotated assembly language listings as part of your answer.)

4. **(20 points)** Book, Exercise 7-5: Write a small C program that exposes function argument evaluation order. Compile it with Sun cc and gcc with and without optimization. Report the evaluation order for all four combinations. Are they all the same? Does optimization change the order? Hand in your test program.

5. **(20 points)** Compare the assembly code generated for the small C program

```
int compare1(int a, int b)
{
  if (a && b) return 1;
  else return 0;
}

int compare2(int a, int b)
{
  if (a & b) return 1;
  else return 0;
}
```

Which version uses the bitwise operator? The short-circuit operator? Which would be faster?

(Hand in an annotated assembly language listing as part of your answer.)

6. **(20 points)** Book: Exercise 8-1: How would you achieve the effect of C++ reference arguments in C? Would both the calling function and the callee have to be changed? Give an implementation of a function that swaps its two arguments in C and C++ and show how it is called.