Assignment 5

Submission Date: 26th April 2015

This is the last part of the assignment in which you generate code for a simulated machine called the C-machine. The accompanying files contain (i) `code.asm`—the code for the factorial program compiled to C-machine code, (ii) `machine.cc`—the code for C-machine itself, and a helpful (iii) `Makefile`. C-machine was coded by Nikhil George with some help from K. Prasanna. The initial ideas came from me.

You have to use C-machine without any changes. If you require new instructions to be added to the C-machine, you have to talk to me to and provide justification. I shall make the changes and inform everybody else.

You may design the code generator on your own. For initial ideas, you can read section 8.6 of the book. This is titled *A simple code generator*. The primary requirement is correctness. The next requirement is completeness—all features should be implemented in a general sense. For instance, your code generator should not fail because an expression was too large and you could not find registers. Going beyond these may earn you bonus points. If you design and implement a code generator that you can argue to be more efficient, that will win you bonus points. The extent of the credit will be decided by me during the grand viva.

Please start off as soon as possible. I shall keep on adding to the assignment description as usual.

The C-machine

The C-machine is a class that inherits from the class `CMachineBase` and includes the functions in C-machine code as its private member functions. Most of the complexity of the machine is in the class `CMachineBase`.

```
class CMachine: public CMachineBase
{
#include "code.asm"
};
```

You should generate your code in the file `code.asm`.

The machine has 6 general purpose registers: `eax, ebx, ecx, edx, edi, esi`. Besides, it has two registers `esp` and `ebp` with functionality similar to x386. The stack size is 1000—feel free to change it. The instructions supported by the machine appear as protected member functions in `CMachineBase` so that they are visible to the C-machine code. Here is the list of instructions: Most of them are self-explanatory.

```
void move(Reg rSrc, Reg rDes)
void move(int i, Reg r)     //move immediate int
void move(float f, Reg r)   //move immediate float

void loadi(char* mem, Reg r)
void loadf(char* mem, Reg r)

void storei(int i, char* mem)
void storei(Reg r, char* mem)
void storef(float i, char* mem)
void storef(Reg r, char* mem)

void cmpi(Reg r1, Reg r2)
void cmpi(int val1, Reg r2)
void cmpf(Reg r1, Reg r2)
void cmpf(float val1, Reg r2)

je(label)
jne(label)
jl(label)
jle(label)
jg(label)
jge(label)

void intTofloat(Reg reg)
void floatToint(Reg reg)

void addi(int i, Reg r)
void addi(Reg rSrc, Reg rDes)
void addf(float i, Reg r)
void addf(Reg rSrc, Reg rDes)
void muli(int i, Reg r)
void muli(Reg rSrc, Reg rDes)
void mulf(float i, Reg r)
void mulf(Reg rSrc, Reg rDes)
void divi(int i, Reg r)
void divi(Reg rSrc, Reg rDes)
void divf(float i, Reg r)
void divf(Reg rSrc, Reg rDes)

void pushi(int val)
void pushi(Reg r)
void pushf(float val)
```

```
void pushf(Reg r)
void popi(int i)
void popf(int i)

char* ind(Reg r)                // r = *r
char* ind(Reg r, int offset)    // r = *(r+offset)

void print_int(Reg r)
void print_int(int i)
void print_float(Reg r)
void print_float(float i)
void print_char(Reg r)
void print_char(char val)
void print_string(Reg r)
void print_string(char* val)
```