/\* File MicroC/machine.c

   A unified-stack abstract machine for imperative programs.

   sestoft@itu.dk \* 2009-10-18

   Compile like this, on ssh.itu.dk say:

      gcc -O3 -Wall machine.c -o machine

   If necessary, force compiler to use 32 bit integers:

      gcc -O3 -m32 -Wall machine.c -o machine

   To execute a program file using this abstract machine, do:

      machine <programfile> <arg1> <arg2> ...

   To get also a trace of the program execution:

      machine -trace <programfile> <arg1> <arg2> ...

\*/

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <sys/time.h>

#ifndef \_WIN32

#include <sys/resource.h>

#endif

// These numeric instruction codes must agree with MicroC/Machine.fs:

// (Use #define because const int does not define a constant in C)

#define CSTI 0

#define ADD 1

#define SUB 2

#define MUL 3

#define DIV 4

#define MOD 5

#define EQ 6

#define LT 7

#define NOT 8

#define DUP 9

#define SWAP 10

#define LDI 11

#define STI 12

#define GETBP 13

#define GETSP 14

#define INCSP 15

#define GOTO 16

#define IFZERO 17

#define IFNZRO 18

#define CALL 19

#define TCALL 20

#define RET 21

#define PRINTI 22

#define PRINTC 23

#define LDARGS 24

#define STOP 25

#define STACKSIZE 1000

// Print the stack machine instruction at p[pc]

void printInstruction(int p[], int pc)

{

  switch (p[pc])

  {

  case CSTI:

    printf("CSTI %d", p[pc + 1]);

    break;

  case ADD:

    printf("ADD");

    break;

  case SUB:

    printf("SUB");

    break;

  case MUL:

    printf("MUL");

    break;

  case DIV:

    printf("DIV");

    break;

  case MOD:

    printf("MOD");

    break;

  case EQ:

    printf("EQ");

    break;

  case LT:

    printf("LT");

    break;

  case NOT:

    printf("NOT");

    break;

  case DUP:

    printf("DUP");

    break;

  case SWAP:

    printf("SWAP");

    break;

  case LDI:

    printf("LDI");

    break;

  case STI:

    printf("STI");

    break;

  case GETBP:

    printf("GETBP");

    break;

  case GETSP:

    printf("GETSP");

    break;

  case INCSP:

    printf("INCSP %d", p[pc + 1]);

    break;

  case GOTO:

    printf("GOTO %d", p[pc + 1]);

    break;

  case IFZERO:

    printf("IFZERO %d", p[pc + 1]);

    break;

  case IFNZRO:

    printf("IFNZRO %d", p[pc + 1]);

    break;

  case CALL:

    printf("CALL %d %d", p[pc + 1], p[pc + 2]);

    break;

  case TCALL:

    printf("TCALL %d %d %d", p[pc + 1], p[pc + 2], p[pc + 3]);

    break;

  case RET:

    printf("RET %d", p[pc + 1]);

    break;

  case PRINTI:

    printf("PRINTI");

    break;

  case PRINTC:

    printf("PRINTC");

    break;

  case LDARGS:

    printf("LDARGS");

    break;

  case STOP:

    printf("STOP");

    break;

  default:

    printf("<unknown>");

    break;

  }

}

// Print current stack and current instruction

void printStackAndPc(int s[], int bp, int sp, int p[], int pc)

{

  printf("[ ");

  int i;

  for (i = 0; i <= sp; i++)

    printf("%d ", s[i]);

  printf("]");

  printf("{%d:", pc);

  printInstruction(p, pc);

  printf("}\n");

}

// Read instructions from a file, return array of instructions

int \*readfile(char \*filename)

{

  int capacity = 1, size = 0;

  int \*program = (int \*)malloc(sizeof(int) \* capacity);

  FILE \*inp = fopen(filename, "r");

  int instr;

  while (fscanf(inp, "%d", &instr) == 1)

  {

    if (size >= capacity)

    {

      int \*buffer = (int \*)malloc(sizeof(int) \* 2 \* capacity);

      int i;

      for (i = 0; i < capacity; i++)

        buffer[i] = program[i];

      free(program);

      program = buffer;

      capacity \*= 2;

    }

    program[size++] = instr;

  }

  fclose(inp);

  return program;

}

// The machine: execute the code starting at p[pc]

//p:存放程序，是数组

//s：堆栈，保存全局变量、局部变量、中间计算结果。是调用栈

//pc:程序计数器，指向下一条指令的地址。是寄存器

//sp:堆栈指针。是寄存器

//bp：栈帧基指针，保存当前栈帧(stack frame)开始地址。是寄存器

//iargs：参数

int execcode(int p[], int s[], int iargs[], int iargc, int /\* boolean \*/ trace)

{

  int bp = -999; // 基指针, for local variable access

  int sp = -1;   // 栈顶指针

  int pc = 0;    // 程序计数器: next instruction

  for (;;)

  {

    if (trace)

      printStackAndPc(s, bp, sp, p, pc);

    switch (p[pc++])

    {

    case CSTI:

      s[sp + 1] = p[pc++];//把程序中的第一个压入堆栈s

      sp++; //堆栈指针更新到下一个栈帧

      break;

    case ADD:

      s[sp - 1] = s[sp - 1] + s[sp];//堆栈中上一个栈帧的值+当前栈帧的值

      sp--;//栈帧更新到上一个栈帧

      break;

    case SUB:

      s[sp - 1] = s[sp - 1] - s[sp];//堆栈中上一个栈帧的值-当前栈帧的值

      sp--;//栈帧更新到上一个栈帧

      break;

    case MUL:

      s[sp - 1] = s[sp - 1] \* s[sp];//堆栈中上一个栈帧的值\*当前栈帧的值

      sp--;//栈帧更新到上一个栈帧

      break;

    case DIV:

      s[sp - 1] = s[sp - 1] / s[sp];//堆栈中上一个栈帧的值/当前栈帧的值

      sp--;//栈帧更新到上一个栈帧

      break;

    case MOD:

      s[sp - 1] = s[sp - 1] % s[sp];//堆栈中上一个栈帧的值%当前栈帧的值

      sp--;//栈帧更新到上一个栈帧

      break;

    case EQ:

      s[sp - 1] = (s[sp - 1] == s[sp] ? 1 : 0);//比较上一个栈帧的值和当前栈帧的值，若相等，上一个栈帧的值为1，否则为0

      sp--;//栈帧更新到上一个栈帧

      break;

    case LT:

      s[sp - 1] = (s[sp - 1] < s[sp] ? 1 : 0);//比较上一个栈帧的值和当前栈帧的值，若小于，上一个栈帧的值为1，否则为0

      sp--;//栈帧更新到上一个栈帧

      break;

    case NOT:

      s[sp] = (s[sp] == 0 ? 1 : 0);//当前栈帧的值若为0则当前栈帧的值设置为1，否则设置为0

      break;

    case DUP:

      s[sp + 1] = s[sp];//复制当前栈帧

      sp++;//栈帧指针更新到下一个栈帧

      break;

    case SWAP:

    {

      int tmp = s[sp];//交换当前栈帧和上一个栈帧的值

      s[sp] = s[sp - 1];

      s[sp - 1] = tmp;

    }

    break;

    case LDI: //间接加载

      s[sp] = s[s[sp]];//当前栈帧的值作为下标，找到栈中指定的栈帧，把当前栈帧的值设置为那个指定栈帧的值

      break;

    case STI: // 存储间接，使值在顶部

      s[s[sp - 1]] = s[sp];//上一个栈帧的值作为下标，找到栈中指定的栈帧，把当前栈帧的值赋值给那个指定栈帧

      s[sp - 1] = s[sp];//上一个栈帧的值更新为当前栈帧的值

      sp--;//栈帧更新到上一个栈帧

      break;

    case GETBP:

      s[sp + 1] = bp;//下一个栈帧的值是基指针

      sp++;//栈帧更新到下一个栈帧

      break;

    case GETSP:

      s[sp + 1] = sp;//下一个栈帧的值是堆栈指针

      sp++;//栈帧更新到下一个栈帧

      break;

    case INCSP:

      sp = sp + p[pc++];//堆栈指针指向程序中pc寄存器下一个的内容

      break;

    case GOTO:

      pc = p[pc];//pc寄存器的值 为 程序中pc寄存器指向的元素的值

      break;

    case IFZERO:

      pc = (s[sp--] == 0 ? p[pc] : pc + 1);//当前栈帧的值若为0，pc就指向程序中pc寄存器指向的元素，否则pc寄存器的值+1。堆栈指针更新为上一个

      break;

    case IFNZRO:

      pc = (s[sp--] != 0 ? p[pc] : pc + 1);//当前栈帧的值若不为0，pc就指向程序中pc寄存器指向的元素，否则pc寄存器的值+1。堆栈指针更新为上一个

      break;

    case CALL:

    {

      int argc = p[pc++];//参数

      int i;

      for (i = 0; i < argc; i++)   // 为返回地址腾出空间

        s[sp - i + 2] = s[sp - i]; // 和旧的基指针

      s[sp - argc + 1] = pc + 1;

      sp++;

      s[sp - argc + 1] = bp;

      sp++;

      bp = sp + 1 - argc;

      pc = p[pc];

    }

    break;

    case TCALL:

    {

      int argc = p[pc++]; // Number of new arguments

      int pop = p[pc++];  // Number of variables to discard

      int i;

      for (i = argc - 1; i >= 0; i--) // 放弃变量

        s[sp - i - pop] = s[sp - i];

      sp = sp - pop;

      pc = p[pc];

    }

    break;

    case RET:

    {

      int res = s[sp];

      sp = sp - p[pc];

      bp = s[--sp];

      pc = s[--sp];

      s[sp] = res;

    }

    break;

    case PRINTI:

      printf("%d ", s[sp]);

      break;

    case PRINTC:

      printf("%c", s[sp]);

      break;

    case LDARGS:

    {

      int i;

      for (i = 0; i < iargc; i++) //Push命令行参数

        s[++sp] = iargs[i];

    }

    break;

    case STOP:

      return 0;

    default:

      printf("Illegal instruction %d at address %d\n", p[pc - 1], pc - 1);

      return -1;

    }

  }

}

// Read program from file, and execute it

int execute(int argc, char \*\*argv, int /\* boolean \*/ trace)

{

  int \*p = readfile(argv[trace ? 2 : 1]);          // program bytecodes: int[]

  int \*s = (int \*)malloc(sizeof(int) \* STACKSIZE); // stack: int[]

  int iargc = trace ? argc - 3 : argc - 2;

  int \*iargs = (int \*)malloc(sizeof(int) \* iargc); // program inputs: int[]

  int i;

  for (i = 0; i < iargc; i++) // Convert commandline arguments

    iargs[i] = atoi(argv[trace ? i + 3 : i + 2]);

// Measure cpu time for executing the program

#ifndef \_WIN32

  struct rusage ru1, ru2;

  getrusage(RUSAGE\_SELF, &ru1);

#endif

  int res = execcode(p, s, iargs, iargc, trace); // Execute program proper

#ifndef \_WIN32

  getrusage(RUSAGE\_SELF, &ru2);

  struct timeval t1 = ru1.ru\_utime, t2 = ru2.ru\_utime;

  double runtime = t2.tv\_sec - t1.tv\_sec + (t2.tv\_usec - t1.tv\_usec) / 1000000.0;

  printf("Used %7.3f cpu seconds\n", runtime);

#endif

  return res;

}

// Read code from file and execute it

int main(int argc, char \*\*argv)

{

  if (argc < 2)

  {

    printf("Usage: machine [-trace] <programfile \*.out> <arg1> ...\n");

    return -1;

  }

  else

  {

    int trace = argc >= 3 && 0 == strncmp(argv[1], "-trace", 7);

    return execute(argc, argv, trace);

  }

}