ECE 486: PDP-8 Instruction Set Simulator

March 12, 2015

Edward Sayers

1 Introduction

The goal of this project was to design a program to simulate the instruction set architecture of a PDP-8. All PDP-8 instructions are supported except the i/o instructions which are treated as no ops. Byte swap (BSW) and EAE instructions are also not supported because they were not found in the standard version of the PDP-8. The front panel switches can be set at run time via command line argument.

The program is written in C++ and compiled using g++ with the c++11 standard. The simulator is written in three main sections. The first two sections handle the simulation. One manages the memory and other related functions and the other handles simulation of the op codes. The last section starts the simulation and handles command line arguments.

The simulation is started from the command line and requires a memory file be specified. It accepts memory files in either hexadecimal or octal format. Exactly one of these options must be chosen or an error will be displayed. In addition to this there are options to change the trace file name and print debugging information. Starting the simulator with no options, or with incorrect options, will display help information.

2 Design

2.1 Memory

The first section of the simulator to be designed was the memory system. The memory consists of an array of 4096 12-bit words. Each word is stored as a structure containing the 12-bit word as a bitset and an access tag. The access tag is initialized to false but will be set to true when its memory location is accessed. Tagging the words this way allows memory to be printed without printing out untouched sections of memory. A public method was written to print the memory to a chosen stream.

Loading of the memory file is done by one of two methods depending of the format to be loaded. Both functions operate in a similar manner, just differing in how they interpret the memory file. The memory information taken from the file is stored into the memory array using a private method that does not log the access.

Logging of memory transactions is done through the load, store and fetch public methods. These functions utilize private methods to load and store data but also log the transaction with the appropriate type. These three methods are the only public methods that can access memory once loaded from file, so there is no danger of unlogged transactions occurring.

2.2 Simulation

The simulation takes place in two steps. First, the instruction is fetched and decoded, and then it is processed depending on its op code. Fetching the instruction is done by the memory section and the instruction is given to the decode function. The decode function takes the individual elements of the instruction apart and stores them in a structure which is passed back. The structure contains the op code, indirect flag, zero flag, offset, micro instruction, and the page the instruction was fetched from. This information is used by the next part to execute the instruction.

Each op code is handled by a different section of code via a switch statement. Micro ops are handled by a separate function.

The op codes which reference memory first calculate the effective address of the target. This is done by a function which takes the instruction data structure and returns the address. Indirection and auto incrementation is handled by this function and the statistics are updated if necessary. After the address is calculated, the data is loaded and/or stored as necessary and state of the object is updated to reflect the proper execution of the op code.

2.3 Interface

The interface processes command line flags using the getopt function. After processing all flags and updating the simulator object as necessary, the simulation is started. The simulator runs to completion and then the final contents of memory and the statistics are printed.

3 Testing

Testing was accomplished by designing PDP-8 assembly programs to target specific functionality and ensure proper operation. Short programs were written for each op code and for other portions of the program that required testing.

3.1 AND

```
            / and.as - test file for and
            / The first two and instructions should result in 0.
            / The last should result in the value 05050.
            / It should run in 26 cycles

            *010
00010 0277 AI,      0277

            *0200   / Program
00200 7300 Main,    cla cll        / clear ac and l
00201 0250          and A          / And with zero
00202 3410          dca I AI       / save
00203 1250          tad A
00204 0251          and B          / and alternating bit patterns
00205 3410          dca I AI       / save
00206 7040          cma
00207 0250          and A          / and with all 1's
00210 3410          dca I AI       / save
00211 7402          hlt

            *250    / Data
00250 5252 A,       05252
00251 2525 B,       02525
            $Main
```

3.2 TAD

```
                / tad.as - test file for tad
                / -1 + 1 should result in 0 with a carry
                / -1 - 1 should result in 07776 with a carry
                / 03777 + 03777 should result in 07776 without a carry
                / 04000 + 04000 should result in 0 with a carry
                / 04000 + 03777 should result in 07777 without a carry
                / It should run in 51 cycles

                *010    / Auto Increment
00010 0277 AI,          0277

                *0200   / Program
00200 7300 Main,        cla cll         / clear ac and l
00201 1250              tad A
00202 1251              tad B           / -1 + 1
00203 3410              dca I AI        / store
00204 7100              cll
00205 1250              tad A
00206 1250              tad A           / -1 -1
00207 3410              dca I AI        / store
00210 7100              cll
00211 1253              tad D
00212 1253              tad D           / largest + largest
00213 3410              dca I AI        / store
00214 7100              cll
00215 1252              tad C
00216 1252              tad C           / smallest + smallest
00217 3410              dca I AI
00220 7100              cll
00221 1252              tad C
00222 1253              tad D           / smallest + largest
00223 3410              dca I AI        / store
00224 7402              hlt

                *250    / Data
00250 7777 A, 07777                     / -1
00251 0001 B, 01                        / 1
00252 4000 C, 04000                     / smallest negative
00253 3777 D, 03777                     / largest positive
                $Main
```

3.3 ISZ

```
                / isz.as -test file for isz
                / Loop should run twice
                / ac should end at two
                / should run in 9 cycles

                *010    / Auto Increment
00010 0277 AI,          0277

                *0200   / Program
00200 7300 Main,        cla cll         / clear ac and l
00201 7001 Loop,        iac             / increment ac
00202 2250              isz A           / increment a until 0
00203 5201              jmp Loop        / loop back
00204 7402              hlt             / halt
                *250    / Data
00250 7776 A,           07776           / -2
                $Main
```

3.4 DCA

```
                / dca.as - test file for dca
                / 01, 020, and 0300 should be stored starting at address 0300
                / should run in 40 cycles


                *010    / Auto Increment Load
00010 0250 AI,          0250
                *017    / Auto Increment Store
00017 0277 AI2,         0277

                *0200   / Program
00200 7300 Main,        cla cll         / clear ac and l
00201 1410 Loop,        tad I AI        / load
00202 3417              dca I AI2       / store
00203 2250              isz A           / test
00204 5201              jmp Loop        / loop
00205 7402              hlt             / halt

                *250    / Data
00250 7775 A,           07775 -3
00251 0001              01
00252 0020              020
00253 0300              0300
                $Main
```

3.5 JMS

```
                / jms.as -test file for jms
                / should increment, rotate 2 right, compliment ac
                / should run in 18 cycles

                *0020   / Functions
00020 0000 J1,          0               / return value
00021 7001              iac             / ac++
00022 5420              jmp I J1        / return
00023 0000 J2,          0               / return value
00024 7012              rtr             / rotate right
00025 5423              jmp I J2        / return
                *0200   / Program
00200 7300 Main,        cla cll         / clear ac and l
00201 4020              jms J1          / call J1
00202 4650              jms I A         / call J2
00203 4205              jms J3          / call J3
00204 7402              hlt             / halt

00205 0000 J3,          0               / return value
00206 7040              cma             / complement
00207 5605              jmp I J3        / return

                *250    / Data
00250 0023 A,           023             / address of J2
                $Main
```

3.6 JMP

```
           / jmp.as - test file for jmp
           / should increment and rotate right ac
           / should run in 7 cycles

           *010     / Auto Increment
00010 0250 AI,      0250
           *017
00017 0277 AI2,     0277

           *0200    / Program
00200 7300 Main,    cla cll          / clear ac and l
00201 5205          jmp A            / jump to A
00202 7402          hlt              / halt
00203 7010          rar              / rotate right
00204 7402          hlt              / halt
00205 7001 A,       iac              / increment
00206 5650          jmp I B          / jump indirect
00207 7402          hlt              / hlt

           *250     / Data
00250 0203 B,       0203             / address of rar
           $Main
```

3.7 Micro ops

3.7.1 Group 1

```
           / m1.as - Tests group 1 micro ops
           / should manipulate the ac in the ways decribed below
           / should run in 9 cycles

           *0200    / Program
00200 7300 Main,    cla cll          / clear ac an l
00201 7060          cma cml          / compliment ac and l
00202 7001          iac              / increment ac
00203 7001          iac              / increment ac
00204 7010          rar              / rotate right
00205 7004          ral              / rotate left
00206 7012          rtr              / rotate two right
00207 7006          rtl              / rotate two left
00210 7402          hlt              / halt

           *0250    / Data
00250 7777 A,       07777
           $Main
```

3.7.2 Group 2 AND Subgroup

```
           / m2-and1.as - Tests 1 of  group 2 AND subgroup
           / should not skip any instructions
           / Should run in 10 cycles

           *0200    / program
00200 7340 Main,    cla cll cma      / clear ac and l, compliment ac
00201 7710          spa cla          / skip on pos then clear ac
00202 7340          cla cll cma      / should not be skipped
00203 7300          cla cll          / zero ac and inc
00204 7650          sna cla          / skip on non-zero ac
00205 7340          cla cll cma      / should not be skipped
00206 7320          cla cll cml      / clear ac and l then compliment l
00207 7430          szl              / skip on non-zero link
00210 7340          cla cll cma      / should not be skipped
00211 7402          hlt              / halt
           $Main
```

```
                / m2—and2.as – Test 2 of group 2 AND subgroup
                / Should in 8 cycles

                *0200   /program
00200 7300 Main,    cla cll          / clear ac and l
00201 7710          spa cla          / skip on pos then clear ac
00202 7340          cla cll cma      / should be skipped
00203 7301          cla cll iac      / zero ac and inc
00204 7650          sna cla          / skip on non−zero ac
00205 7340          cla cll cma      / should be skipped
00206 7300          cla cll          / clear ac and l
00207 7430          szl              / skip on zero link
00210 7340          cla cll cma      / should be skipped
00211 7410          skp              / skip always
00212 7340          cla cll cma      / should be skipped
00213 7402          hlt              / halt
                $Main
```

```
                / m2—and3.as – Test 3 of group 2 AND subgroup
                / Should run in 3 cycles

                *0200   / program
00200 7301 Main,    cla cll iac      / clear ac and l, increment ac
00201 7770          spa sna szl cla / skip on all 3
00202 7340          cla cll cma      / should be skipped
00203 7402          hlt              / halt
                $Main
```

```
                / m2—and4.as – Test 4 of group 2 AND subgroup
                / Should run in 10 cycles

                *0200   / program
00200 7340 Main,    cla cll cma      / clear ac and l, compliment ac
00201 7770          spa sna szl cla / skip on all 3
00202 7340          cla cll cma      / should not be skipped
00203 7300          cla cll          / clear ac and l
00204 7770          spa sna szl cla / skip on all 3
00205 7340          cla cll cma      / should not be skipped
00206 7321          cla cll iac cml / clear ac and l, inc ac, comp l
00207 7770          spa sna szl cla / skip on all 3
00210 7340          cla cll cma      / should not be skipped
00211 7402          hlt              / halt
                $Main
```

3.7.3 Group 2 OR Subgroup

```
                / m2−1.as – Test 1 of  group 2 OR subgroup
                / Should run in 7 cycles

                *0200   / program
00200 7340 Main,    cla cll cma      / clear ac and l then compliment ac
00201 7700          sma cla          / skip on minus then clear ac
00202 7340          cla cll cma      / should be skipped
00203 7300          cla cll          / zero ac
00204 7640          sza cla          / skip on zero ac
00205 7340          cla cll cma      / should be skipped
00206 7320          cla cll cml      / complement link
00207 7420          snl              / skip on non−zero link
00210 7340          cla cll cma      / should be skipped
00211 7402          hlt              / halt
                $Main
```

```
            / m2-2.as - Test 2 of group 2 OR subgroup
            / Should run in 10 cycles

            *0200                   / start at address 0200
00200 7300 Main,   cla cll          / clear ac and l
00201 7700         sma cla          / skip on minus then clear ac
00202 7340         cla cll cma      / should not be skipped
00203 7340         cla cll cma      / compliment ac
00204 7640         sza cla          / skip on zero ac
00205 7340         cla cll cma      / should not be skipped
00206 7300         cla cll          / clear ac and l
00207 7420         snl              / skip on non-zero link
00210 7340         cla cll cma      / should not be skipped
00211 7402         hlt              / halt
            $Main
```

```
            / m2-or3.as - Test 3 of group 2 OR subgroup
            / Should run in 11 cycles

            *0200   / program
00200 7301 Main,   cla cll iac      / clear ac and l then inc
00201 7760         sma sza snl cla / skip on all 3
00202 7340         cla cll cma      / should not be skipped
00203 7300         cla cll          / clear ac and l
00204 7760         sma sza snl cla / skip on all 3
00205 7340         cla cll cma      / should be skipped
00206 7300         cla cll          / clear ac
00207 7040         cma              / complement ac
00210 7760         sma sza snl cla / skip on all 3
00211 7340         cla cll cma      / should be skipped
00212 7402         hlt              / halt
            $Main
```

```
            / m2-or3.as - Test 4 of group 2 OR subgroup
            / Should run in 3 cycles

            *0200   / program
00200 7320 Main,   cll cla cml      / compliment link
00201 7760         sma sza snl cla / skip on all 3
00202 7340         cla cll cma      / should be skipped
00203 7402         hlt              / halt
            $Main
```

3.7.4 Group 2 OSR

```
            / m2-osr.as - Tests group 2 or with switches
            / Should or accumulator with switch settings
            / Should run in 3 cycles

            *0200   / program
00200 7300 Main,   cla cll          / clear ac and l
00201 7404         osr              / or with switches
00202 7402         hlt              / halt
            $Main
```

3.8 Indirection

```
                  / ind-1.as - Indirection test 1
                  / Should copy 01, 020, 0300 to memory starting at 0300

                  *010     / Auto Increment
00010 0250 AI,             0250
                  *017
00017 0277 AI2,            0277

                  *0200    / Program
00200 7300 Main,           cla cll          / clear ac and l
00201 1410 Loop,           tad I AI         / load
00202 3417                 dca I AI2        / store
00203 2250                 isz A            / test
00204 5201                 jmp Loop         / loop
00205 7402                 hlt              / halt

                  *250     / Data
00250 7775 A,              07775
00251 0001                 01
00252 0020                 020
00253 0300                 0300
                  $Main
```

```
                  / ind-2.as - Indirection test 2
                  / Should store 07777 in 300 and 301
                  / Should complete in 13 cycles

                  *020     / indirect from zero page
00020 0301 B,              0301

                  *0200    / program
00200 7340 Main,           cla cll cma      / clear ac and l and complement ac
00201 3650                 dca I A          / Store indirectly through A
00202 7040                 cma              / Compliment ac
00203 3420                 dca I B          / Store indirectly through B
00204 7402                 hlt              / halt

                  *0250    / data
00250 0300 A,              0300


                  $Main
```

3.9 Tracefile

```
                  / tf.as - test file for the tracefile
                  / Should log the indicated memory accesses
                  / It should run in 16 cycles

                  *010     / Auto Increment
00010 0277 AI,             0277
                  *020     / Function
00020 0000 J1,             0                / return address
00021 7001                 iac              / fetch
00022 5420                 jmp I J1         / fetch, read

                  *0200    / Program
00200 7300 Main,           cla cll          / fetch
00201 0250                 and A            / fetch, read
00202 3410                 dca I AI         / fetch, read, store, store
00203 1250                 tad A            / fetch, read
00204 2250                 isz A            / fetch, read, store
00205 4020                 jms J1           / fetch, store
00206 7402                 hlt              / fetch

                  *250     / Data
00250 7776 A,              07776            / -2
                  $Main
```

4 Code

```
// pdp8-main.cpp
// Edward Sayers
// ECE 486: PDP-8 Instruction Set Simulator

#include <iostream>
#include <unistd.h>
#include <cctype>
#include <cstdlib>
#include "pdp8-simulator.h"

void printHelp(std::string);

int main (int argc, char **argv)
{
    int c;

    // Input flags
    bool vflg = false;
    bool oflg = false;
    bool tflg = false;
    bool sflg = false;

    std::string filename;
    std::string tracefile;
    std::string sstring;
    Pdp8::Simulator sim;

    // Process input arguments
    opterr = 0; // Turn off option error messages
    while ((c = getopt(argc, argv, "v:o:t:s:dp")) != -1)
    {
        switch(c)
        {
        case 'v': // Load from hex file
            vflg = true;
            filename = optarg;
            break;
        case 'o': // Load from oct file
            oflg = true;
            filename = optarg;
            break;
        case 't': // Tracefile name
            tflg = true;
            tracefile = optarg;
            break;
        case 's': // Set switches
            sflg = true;
            sstring = optarg;
        case 'd': // Debug
            sim.set_debug(true);
            break;
        case 'p':
            sim.set_pause(true);
            break;
        case '?': // Invalid argument
            switch (optopt)
            {
            case 'o':
            case 'v':
            case 't':
            case 's':
                std::cerr << "Option -" << (char) optopt << " requires an argument" << std::endl;
                break;
            default:
                if (std::isprint(optopt))
                    std::cerr << "Unknown option: " << (char) optopt << std::endl;
                else
                    std::cerr << "Unknown option: 0x" << std::hex << optopt << std::endl;
                printHelp(argv[0]);
                break;
            }
        default:
            printHelp(argv[0]);
            return 0;
        }
    }

    // Handle -o and -v flags
    if (oflg && vflg)
    {
        std::cerr << "Options -o and -v are mutually exclusive" << std::endl;
        printHelp(argv[0]);
        return 0;
    }
    else if (oflg | vflg)
    {
        try
        {
            int ferr = 0; // To store number of times memory is touched
            if (oflg)
            {
                ferr = sim.load_from_oct(filename);
            }
            else if (vflg)
            {
                ferr = sim.load_from_hex(filename);
            }

            if (ferr == 0)
            {
                std::cerr << "File \"" << filename << "\" does not exist or contains no data" << std::
                    endl;
                printHelp(argv[0]);
                return 0;
            }
        }
        catch (...)
        {
            std::cerr << "Unable to load \"" << filename << "\": invald file format" << std::endl;
            printHelp(argv[0]);
            return 0;
        }
    }
    else
    {
        std::cerr << "Either option -o or -v must be present" << std::endl;
        printHelp(argv[0]);
        return 0;
    }

    // Handle -t flag
    if (tflg)
    {
        sim.set_tracefile(tracefile);
    }

    if (sflg)
    {
        Pdp8::reg12 sw;
        try
        {
           sw = std::stoul(sstring, NULL, 8);
        }
        catch (...)
        {
            std::cerr << "Switch value is invalid" << std::endl;
            printHelp(argv[0]);
            return 0;
        }
        sim.set_switches(sw);
    }

    sim.start();
    std::cout << std::endl << "Final memory contents:" << std::endl;
    sim.dump_memory();
    std::cout << std::endl;
    sim.print_stats();

    return 0;
}

void printHelp(std::string filename)
{
    std::cerr << "Usage: " << filename << " [options]" << std::endl;
    std::cerr << "  options:" << std::endl;
    std::cerr << "-v <filename>     Loads a hex formated memory file" << std::endl;
    std::cerr << "-o <filename>     Loads an octal formated memory file" << std::endl;
    std::cerr << "-t <filename>     Specifies the filename for the tracefile output" << std::endl;
    std::cerr << "-s <oct number>   Sets the switches to the value in number" << std::endl;
    std::cerr << "-d                Print debug messages" << std::endl;
    std::cerr << "-p                Pause between debug messages" << std::endl;
}
```

```
// pdp8-memory.h
// Edward Sayers
// ECE 486: PDP-8 Instruction Set Simulator


#ifndef PDP8_MEMORY_H
#define PDP8_MEMORY_H

#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <stdexcept>
#include <bitset>

namespace Pdp8
{
    typedef std::bitset<12> reg12;
    typedef std::bitset<9> reg9;

    namespace mem
    {
        const int size = 4096;
        const std::string default_logfile = "tracefile";

        enum log_type
        {
            data_read = 0,
            data_write = 1,
            inst_fetch = 2,
        };
    }

    struct Word
    {
        Word();
        reg12 value;
        bool access;
    };

    class Memory
    {
    public:
        Memory();
        ~Memory();

        void store(unsigned short, reg12);
        reg12 load(unsigned short);
        reg12 fetch(unsigned short);

        int dump_memory(std::ostream&) const;
        int load_from_hex(std::string);
        int load_from_oct(std::string);

        void set_tracefile(std::string);
    private:
        const unsigned int mem_size;
        Pdp8::Word *mem;

        std::string logfile_name;
        std::ofstream logfile;

        void mem_put(unsigned short, reg12);
        reg12 mem_get(unsigned short);
        void log(unsigned short, Pdp8::mem::log_type);
    };
}

#endif // PDP8_MAIN_MEMORY_H
```

```
// pdp8-memory.cpp
// Edward Sayers
// ECE 486: PDP-8 Instruction Set Simulator

#include "pdp8-memory.h"


//
// Public functions
//

// Constructor for struct Word
Pdp8::Word::Word()
{
    access = false;
}

// Constructor for class Memory
Pdp8::Memory::Memory() : mem_size(Pdp8::mem::size)
{
    mem = new Pdp8::Word[mem_size];

    set_tracefile(Pdp8::mem::default_logfile);
}

// Destructor for class Memory
Pdp8::Memory::~Memory()
{
    delete [] mem;
    logfile.close();
}

// Display all memory locations that have been previously accessed
// INPUT: ostream to display on
// OUTPUT: number of memory locations displayed
int Pdp8::Memory::dump_memory(std::ostream& out) const
{
    int count = 0;  // To count lines displayed

    // Setup stream format
    out << std::oct << std::setfill('0') << std::right;

    for (unsigned int i = 0; i < mem_size; ++i)
    {
        if (mem[i].access)
        {
            out << std::setw(4) << i << ": ";
            out << std::setw(4) << mem[i].value.to_ulong() << std::endl;
            ++count;
        }
    }

    return count;
}


// Load memory from hex file
// INPUT: file name to load from
// OUTPUT: number of memory locations loaded
int Pdp8::Memory::load_from_hex(std::string filename)
{
    std::ifstream file (filename);  // Open filename as file
    std::string line;               // String for getline
    int count = 0;                  // Count of memory locations
    unsigned short address = 0;     // Current address

    Pdp8::Word *new_mem = new Pdp8::Word[mem_size];

    if (file.is_open())
    {
        while (std::getline(file, line))
        {

            if (line.length()) // Ignore blank lines
            {
                // Lines starting with "@" specify a starting address
                if (line.compare(0, 1, "@") == 0)
                {
                    unsigned long conv; // Converted string
                    try
                    {
                        conv = std::stoul(line.substr(1), NULL, 16);
                    }
                    catch (...)
                    {
                        // If conversion fails, delete new_mem and rethrow
                        file.close();
                        delete [] new_mem;
                        throw;
                    }

                    // Check that requested address is valid
                    if (conv < mem_size)
                    {
                        address = conv;
                    }
                    else
                    {
                        file.close();
                        delete [] new_mem;
                        throw std::out_of_range("Array out of bounds");
                    }
                }
                else
                {
                    unsigned long conv;

                    // Check that address is valid
                    if (address >= mem_size)
                    {
                        file.close();
                        delete [] new_mem;
                        throw std::out_of_range ("Array out of bounds");
                    }

                    try
                    {
                        conv = std::stoul(line, NULL, 16);
                    }
                    catch (...)
                    {
                        // If conversion fails, delete new_mem and rethrow
                        file.close();
                        delete [] new_mem;
                        throw;
                    }

                    new_mem[address].value = conv;
                    new_mem[address++].access = true;
                    ++count;
                }
            }
        }
        file.close();
    }

    delete [] mem;
    mem = new_mem;

    return count;
}
```

```
// Load memory from octal file
// INPUT: file name to load from
// OUTPUT: number of memory locations loaded
int Pdp8::Memory::load_from_oct(std::string filename)
{
    std::ifstream file (filename);  // Open filename as file
    std::string line1, line2;       // String for getline
    int count = 0;                  // Count of memory locations
    unsigned short address = 0;     // Current address

    Pdp8::Word *new_mem = new Pdp8::Word[mem_size];

    if (file.is_open())
    {
        while (std::getline(file, line1) && std::getline(file, line2))
        {
            unsigned long conv1, conv2;

            try
            {
                conv1 = std::stoul(line1, NULL, 8);
                conv2 = std::stoul(line2, NULL, 8);
            }
            catch (...)
            {
                file.close();
                delete [] new_mem;
                throw;
            }

            bool is_address = conv1 & (1 << 6);

            // remove upper bits
            conv1 &= ~((~0u) << 6);
            conv2 &= ~((~0u) << 6);

            unsigned short value = (conv1 << 6) | conv2;

            if (is_address)
            {
                if (value >= mem_size)
                {
                    file.close();
                    delete [] new_mem;
                    throw std::out_of_range ("Array out of bounds");
                }
                else
                {
                    address = value;
                }
            }
            else
            {
                if (address >= mem_size)
                {
                    file.close();
                    delete [] new_mem;
                    throw std::out_of_range("Array out of bounds");
                }
                else
                {
                    new_mem[address].value = value;
                    new_mem[address++].access = true;
                    ++count;
                }
            }
        }
        file.close();
    }
    delete [] mem;
    mem = new_mem;
    return count;
}



// Store value in memory and log
// INPUT: address and value
// OUTPUT: logged as a write
void Pdp8::Memory::store(unsigned short address, Pdp8::reg12 value)
{
    mem_put(address, value);
    log(address, Pdp8::mem::data_write);
}

// Load value from memory and log
// INPUT: address
// OUTPUT: value returned and logged as a read
Pdp8::reg12 Pdp8::Memory::load(unsigned short address)
{
   Pdp8::reg12 rv = mem_get(address);
   log(address, Pdp8::mem::data_read);
   return rv;
}

// Fetch instruction from memory and log
// INPUT: address
// OUTPUT: instruction returned and logged as a fetch
Pdp8::reg12 Pdp8::Memory::fetch(unsigned short address)
{
    Pdp8::reg12 rv = mem_get(address);
    log(address, Pdp8::mem::inst_fetch);
    return rv;
}

// Set file name of trace file
// INPUT: filename
// OUTPUT: none
void Pdp8::Memory::set_tracefile(std::string filename)
{
    logfile.close();
    logfile_name = filename;
    logfile.open(filename, std::ios::app);
}

//
// Private functions
//

// Put value into memory
// INPUT: address and value
// OUTPUT: None
void Pdp8::Memory::mem_put(unsigned short address, Pdp8::reg12 value)
{
    // Check that address is valid
    if (address >= mem_size)
    {
        throw std::out_of_range ("Array out of bounds");
        return;
    }

    // Insert value at address
    mem[address].value = value;
    mem[address].access = true;
}

// Get value from memory
// INPUT: address
// OUTPUT: value returned
Pdp8::reg12 Pdp8::Memory::mem_get(unsigned short address)
{
    if (address >= mem_size)
    {
        throw std::out_of_range ("Array out of bounds");
        return 0;
    }

    return mem[address].value.to_ulong();
}

// Log memory access
// INPUT: address and type
// OUTPUT: none
void Pdp8::Memory::log(unsigned short address, Pdp8::mem::log_type type)
{

    if (logfile.is_open())
    {
        logfile << type << " " << std::oct << address << std::endl;
    }
}
```

```
// pdp8-simulator.h
// Edward Sayers
// ECE 486: PDP-8 Instruction Set Simulator


#ifndef PDP8_SIMULATOR_H
#define PDP8_SIMULATOR_H

#include "pdp8-memory.h"

namespace Pdp8
{
    namespace Sim
    {
        const int num_ops = 8;

        // Data structure to store statistics
        struct Stats
        {
            int clocks;
            int ops[num_ops];
        };


        // Enumeration of op codes
        enum Ops
        {
            AND = 0,
            TAD = 1,
            ISZ = 2,
            DCA = 3,
            JMS = 4,
            JMP = 5,
            IOT = 6,
            OPR = 7,
        };

        // Data structure for decoded instructions
        struct Inst
        {
           Pdp8::Sim::Ops    op;      // Op code
           bool              ind;     // Indirect flag
           bool              zero;    // Zero flag
           unsigned short    offset;  // Page offset
           reg9              micro;   // Micro coded instructions
           unsigned short    page;    // Page
        };

    };
    class Simulator
    {
    public:
        Simulator();
        ~Simulator();

        // Wrappers for memory functions
        int load_from_hex(std::string);
        int load_from_oct(std::string);
        void set_tracefile(std::string);
        int dump_memory(std::ostream& = std::cout) const;

        // Print statistics
        void print_stats(std::ostream& = std::cout) const;

        // Start simulation
        void start(unsigned short = 0200);

        // Functions to change parameters
        void set_debug(bool);
        void set_pause(bool);
        void set_switches(Pdp8::reg12);
    private:
        // State
        Memory *  memory;    // Pointer to memory class
        Sim::Stats stats;    // Statistic struct
        reg12 ac;            // Accumulator
        reg12 pc;            // Program Counter
        bool  l;             // Link Register
        reg12 switches;      // State of front panel switches
        bool debug;          // Debug flag
        bool pause;          // Pause flag

        // private functions
        Sim::Inst decode(reg12, unsigned short);
        bool process_instruction();
        bool process_micro(Pdp8::reg9);
        unsigned short get_address(Pdp8::Sim::Inst);
        void print_debug(void);
    };
};
#endif
```

```
// pdp8-simulator.cpp
// Edward Sayers
// ECE 486: PDP-8 Instruction Set Simulator

#include "pdp8-simulator.h"

//
// Public Functions
//


// Constructor for Simulator
Pdp8::Simulator::Simulator()
{
    // Create memory
    memory = new Memory;

    // Zero out stats
    stats.clocks = 0;
    for (int i = 0; i < Sim::num_ops; ++i)
    {
        stats.ops[i] = 0;
    }

    // Switchs, ac and l initialized to 0
    switches = 0;
    ac = 0;
    l = 0;

    // Debug and pause off by default
    debug = false;
    pause = false;
}

// Destructor for simulator
Pdp8::Simulator::~Simulator()
{
    delete memory;
}

// Load simulator memory from hex file
// INPUT: filename to load
// OUTPUT: Number of memory locations loaded
int Pdp8::Simulator::load_from_hex(std::string filename)
{
    return memory->load_from_hex(filename);
}

// Load simulator memory from oct file
// INPUT: filename to load
// OUTPUT: Number of memory locations loaded
int Pdp8::Simulator::load_from_oct(std::string filename)
{
    return memory->load_from_oct(filename);
}

// Set name of tracefile output
// INPUT: filename
// OUTPUT: None
void Pdp8::Simulator::set_tracefile(std::string filename)
{
    memory->set_tracefile(filename);
}

// Dump memory contents to a stream
// INPUT: Stream to use (cout default)
// OUTPUT: Number of memory locations printed
int Pdp8::Simulator::dump_memory(std::ostream& out) const
{
    return memory->dump_memory(out);
}
```

```
// Print statistics
// INPUT: Stream to use (cout default)
// OUTPUT: None
void Pdp8::Simulator::print_stats(std::ostream& out) const
{
    int insts = 0; // Instruction total

    // Total instructions from each mnemonic
    for (int i = 0; i < Pdp8::Sim::num_ops; ++i)
        insts += stats.ops[i];

    // Print statistics
    out << std::dec;
    out << "Clocks    : " << stats.clocks << std::endl;
    out << "AND Ops   : " << stats.ops[Pdp8::Sim::AND] << std::endl;
    out << "TAD Ops   : " << stats.ops[Pdp8::Sim::TAD] << std::endl;
    out << "ISZ Op    : " << stats.ops[Pdp8::Sim::ISZ] << std::endl;
    out << "DCA Ops   : " << stats.ops[Pdp8::Sim::DCA] << std::endl;
    out << "JMS Ops   : " << stats.ops[Pdp8::Sim::JMS] << std::endl;
    out << "JMP Ops   : " << stats.ops[Pdp8::Sim::JMP] << std::endl;
    out << "I/O Ops   : " << stats.ops[Pdp8::Sim::IOT] << std::endl;
    out << "Micro Ops: " << stats.ops[Pdp8::Sim::OPR] << std::endl;
    out << "Total     : " << insts  << std::endl;
}

// Start Simulation
// INPUT: Address to start at
// OUTPUT: None
void Pdp8::Simulator::start(unsigned short pc_start)
{
    bool halt;        // halt condition
    pc = pc_start;  // set pc to starting addess

    // Process instructions until halt is encountered
    // If debug is true, print before and after every instruction
    do
    {
        print_debug();
        halt = process_instruction();
    } while (!halt);
    print_debug();
}

// Sets debug mode
// INPUT: True for on, false for off
// OUTPUT: None
void Pdp8::Simulator::set_debug(bool value)
{
    debug = value;
}

// Sets pause mode
// INPUT: True for on, false for off
// OUTPUT: None
void Pdp8::Simulator::set_pause(bool value)
{
    pause = value;
}

// Sets switches
// INPUT: value of switches
// OUTPUT: None
void Pdp8::Simulator::set_switches(Pdp8::reg12 sw)
{
    switches = sw;
}

//
// Private functions
// Decode instruction
// INPUT: 12 bit instruction to decode
// OUPTUP: instruction decoded in struct
Pdp8::Sim::Inst Pdp8::Simulator::decode(Pdp8::reg12 inst, unsigned short pc)
{
    Pdp8::Sim::Inst rv; // return value

    // Seperate instruction into parts
    rv.op       = static_cast<Pdp8::Sim::Ops>(inst.to_ulong() >> 9); // Op code
    rv.ind      = inst[8];                  // indirect flag
    rv.zero     = inst[7];                  // zero flag
    rv.offset   = inst.to_ulong() & 0x7Fu;  // page offset
    rv.micro    = inst.to_ulong() & 0x1FF;  // micro instructions
    rv.page     = pc >> 7;                  // page

    return rv;
}

// Proccess instruction
// INPUT: None
// OUTPUT: True if not halted, false if halted
bool Pdp8::Simulator::process_instruction()
{
    Pdp8::Sim::Inst inst;           // decoded instruction
    unsigned short  mar;            // address buffer
    Pdp8::reg12     mbr;            // data buffer
    std::bitset<13> lac;            // l+ac for ac arithmetic
    bool            halt = false;   // halt flag, defaults to start

    // Fetch and decode instruction
    inst = decode(memory->fetch(pc.to_ulong()), pc.to_ulong());

    stats.ops[inst.op] += 1;    // add to appropriate instruction count
    pc = pc.to_ulong() + 1;     // increment pc

    // Switch on op code
    switch (inst.op)
    {
    case (Pdp8::Sim::AND):
        // calculate address and load value
        mar = get_address(inst);
        mbr = memory->load(mar);

        // and ac with value from memory
        ac &= mbr;

        // update stats
        stats.clocks += 2;
        break;
    case (Pdp8::Sim::TAD):
        // calculate address and load value
        mar = get_address(inst);
        mbr = memory->load(mar);

        // Combine l and ac
        lac = ac.to_ulong();
        lac[12] = static_cast<unsigned int>(l);

        // add value from memory to lac
        lac = lac.to_ulong() + mbr.to_ulong();

        // update ac and l to reflect new value
        ac = lac.to_ulong() & 0xFFF;
        l = lac[12];

        // update stats
        stats.clocks += 2;
        break;
    case (Pdp8::Sim::ISZ):
        // calculate address and load value
        mar = get_address(inst);
        mbr = memory->load(mar);

        // increment and store
        mbr = mbr.to_ulong() + 1;
        memory->store(mar, mbr);

        // increment pc if result was 0
        if (mbr.to_ulong() == 0)
        {
            pc = pc.to_ulong() + 1;
        }

        // update stats
        stats.clocks += 2;
        break;
    case (Pdp8::Sim::DCA):
        // calculate address and store ac there
        mar = get_address(inst);
        memory->store(mar, ac);

        // clear ac
        ac = 0;

        // update statse
        stats.clocks += 2;
        break;
    case (Pdp8::Sim::JMS):
        // calculate address and store return value there
        mar = get_address(inst);
        memory->store(mar, pc);

        // set pc to instruction after address
        pc = mar + 1;

        // update statistics
        stats.clocks += 2;
        break;
    case (Pdp8::Sim::JMP):
        // calculate address
        mar = get_address(inst);

        // set pc to jump target
        pc = mar;

        // update stats
        stats.clocks += 1;
        break;
    case (Pdp8::Sim::IOT):
        // Print message if IOT instruction is encountered
        std::cerr << std::setfill('0') << "0" << std::oct;
        std::cerr << std::setw(4) << std::oct << pc.to_ulong() - 1;
        std::cerr << ": IOT Instructions are not supported";
        std::cerr << std::endl << std::dec;
        break;
    case (Pdp8::Sim::OPR):
        // handle micro opts in fuction
        halt = process_micro(inst.micro);

        // update stats
        stats.clocks += 1;
        break;
    default:
        break;
    }

    // return halt value
    return halt;
}

// Process a micro op
// INPUT: Micro op to be prossessed
// OUTPUT: True if halt, false otherwise
bool Pdp8::Simulator::process_micro(Pdp8::reg9 micro)
{
    bool halt = false; // halt value defaults to false

    if (micro[8] == 0) // Group 1
    {
        // set flags from instruction
        bool iac = micro[0];
        bool bsw = micro[1];
        bool ral = micro[2];
        bool rar = micro[3];
        bool cml = micro[4];
        bool cma = micro[5];
        bool cll = micro[6];
        bool cla = micro[7];

        // Clear accumulator if cla bit set
        if (cla)
            ac = 0;

        // Clear link if cll bit set
        if (cll)
            l = 0;

        // Complement accumulator if cma bit set
        if (cma)
            ac = ~ac;

        // Complement link if cml bit set
        if (cml)
            l = ~l;


        // combine l and ac
        std::bitset<13> lac = ac.to_ulong();
        lac[12] = static_cast<unsigned int>(l);

        // Increment l/ac if iac bit set
        if (iac)
        {
            // increment combined register
            lac = lac.to_ulong() + 1;

        }


        // handle rotations
        if (bsw)
        {
            if (ral && !rar)
            {
                // rotate left 2
                std::bitset<13> nlac = lac << 2;
                nlac[1] = lac[12];
                nlac[0] = lac[11];

                lac = nlac;
            }
            else if (rar && !ral)
            {
                // rotate right 2
                std::bitset<13> nlac = lac >> 2;
                nlac[12] = lac[1];
                nlac[11] = lac[0];

                lac = nlac;
            }
        }
        else
        {

            if (ral && !rar)
            {
                // rotate left
                std::bitset<13> nlac = lac << 1;
                nlac[0] = lac[12];

                lac = nlac;
            }
            else if (rar && !ral)
            {
                // rotate right
                std::bitset<13> nlac = lac >> 1;
                nlac[12] = lac[0];

                lac = nlac;
            }
        }
        // set l and ac from result
        l = lac[12];
        ac = lac.to_ulong() & 0xFFFu;
    }
    else if (micro[0] == 0) // Group 2
    {
        // set flags from instruction
        bool hlt = micro[1];
        bool osr = micro[2];
        bool rev = micro[3];
        bool snl = micro[4];
        bool sza = micro[5];
        bool sma = micro[6];
        bool cla = micro[7];

        bool skip = false;  // skip flag

        // Skip if bit 8 is true and all conditions in first block  are true
        // Or, skip if any of the other conditions is true
        if (rev)
        {
            if ((!sma || (sma && ac[11] == 0)) &&
                (!sza || (sza && ac != 0))      &&
                (!snl || (snl && l == 0)))
            {
                skip = true;
            }
        }
        else if (sma && ac[11] != 0)
        {
            skip = true;
        }
        else if (sza && ac == 0)
        {
            skip = true;
        }
        else if (snl && l != 0)
        {
            skip = true;
        }

        // skip next instruction if skip is true
        if (skip)
            pc = pc.to_ulong()  +  1;

        // Clear accumulator if cla bit is set
        if (cla)
            ac = 0;

        // OR accumulator with switches if osr bit is set
        if (osr)
            ac |= switches;
        // Return true if halt bit is set
        if (hlt)
            halt = true;

    }
    else // Group 3
    {
        // print message if a group 3 micro opt is encountered
        std::cerr << std::setfill('0') << "0" << std::oct;
        std::cerr << std::setw(4) << pc.to_ulong() - 1;
        std::cerr << ": Group 3 micro ops are not supported";
        std::cerr << std::endl << std::dec;
    }

    return halt;
}

// Get effective address
// INPUT: Instruction to get address from
// OUTPUT: Address
unsigned short Pdp8::Simulator::get_address(Pdp8::Sim::Inst inst)
{
    unsigned short page;
    unsigned short addr;

    // Set page
    if (inst.zero)
        page = inst.page;
    else
        page = 0;

    addr = (page << 7) | inst.offset;

    // Get indirect address
    if(inst.ind)
    {
        if (addr >= 010 && addr <= 017)
        {
            Pdp8::reg12 ival = memory->load(addr);
            ival = ival.to_ulong() + 1;
            memory->store(addr, ival);
            stats.clocks += 2;
            addr = ival.to_ulong();
        }
        else
        {
            addr = memory->load(addr).to_ulong();
        }
    stats.clocks += 1;
    }

    // Get data from memory
    return addr;
}

// Print debug information
// INPUT: None
// OUTPUT: Debug information
void Pdp8::Simulator::print_debug(void)
{
    // Do nothing if debug not set
    if (!debug)
        return;

    std::cerr << std::setfill('0');
    std::cerr << "PC: " << pc << " (0";
    std::cerr << std::setw(4) << std::oct << pc.to_ulong() << ") ";
    std::cerr << "AC: " << ac << " (0";
    std::cerr << std::setw(4) << std::oct << ac.to_ulong() << ") ";
    std::cerr << "L: ";
    if (1)
        std::cerr << "1";
    else
        std::cerr << "0";

    std::cerr << std::dec << " Clock: " << stats.clocks;

    // Handle pause
    if (pause)
        std::cin.get();
    else
        std::cerr << std::endl;
}
```