Lab Report: Better Angles

Zhu Yunqin, PB20061372

Task 1: Read and Understand

Despcription

Translate a piece of machine code into assembly code.

Requirements

- Read the machine code in *foo.txt*.
- Write the assembly code in *translate.txt*.

Solution

1. Implement a disassembler in C++, which can translate the input machine code into assembly code.
2. Execute the disassembly program to translate *foo.txt*. Then, copy the translated text into *translate.txt*.

The disassembly program written in C++ is as follows. Please notice that only part of the instruction set are implemented.

```
#include <algorithm>
#include <bitset>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <map>

using namespace std;

// Disassemble machine code
void disassemble(const string& filename, const string& out_filename) {
  ifstream in(filename);
  if (!in) {
    cerr << "Error: cannot open file " << filename << endl;
    return;
  }
  ofstream out(out_filename);
  if (!out) {
    cerr << "Error: cannot open file " << out_filename << endl;
    return;
  }

  stringstream buf;
  string line;

  // Find .ORIG pseudo command
  while (!in.eof() && line.size() == 0) {
    getline(in, line);
    line.erase(remove_if(line.begin(), line.end(), ::isspace), line.end());
  }
  if (line != "0011000000000000") {
    cerr << "Error: cannot find .ORIG x3000 in" << filename << endl;
    exit(1);
  } else
    buf << ".ORIG x3000" << endl;

  bool is_data = false;
  uint16_t PC = 0x3000, opcode, DR, SR1, SR2, BaseR, trapvect8;
  int16_t imm5, PCoffset9, offset6;
  map<uint16_t, string> labels;
  int branch_count = 0, data_count = 0;
```

```
// Process each instruction
while (!in.eof()) {
  getline(in, line);
  line.erase(remove_if(line.begin(), line.end(), ::isspace), line.end());
  if (line.size() == 0) continue;
  if (line.size() != 16) {
    cerr << "Error: invalid line " << line << endl;
    exit(1);
  }
  PC++;
  if (is_data) {
    buf << ".FILL x" << hex << uppercase << bitset<16>(line).to_ulong()
        << endl;
    continue;
  }
  opcode = bitset<4>(line).to_ulong();
  switch (opcode) {
    case 0b0001:  // ADD
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      SR1 = bitset<3>(line.substr(7, 3)).to_ulong();
      buf << "ADD R" << DR << ", R" << SR1;
      if (line[10] == '0') {
        SR2 = bitset<3>(line.substr(13, 3)).to_ulong();
        buf << ", R" << SR2;
      } else {
        imm5 = bitset<5>(line.substr(11, 5)).to_ulong();
        if (line[11] == '1') imm5 = imm5 - 32;
        buf << ", #" << imm5;
      }
      buf << endl;
      break;

    case 0b0101:  // AND
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      SR1 = bitset<3>(line.substr(7, 3)).to_ulong();
      buf << "AND R" << DR << ", R" << SR1;
      if (line[10] == '0') {
        SR2 = bitset<3>(line.substr(13, 3)).to_ulong();
        buf << ", R" << SR2;
      } else {
        imm5 = bitset<5>(line.substr(11, 5)).to_ulong();
        if (line[11] == '1') imm5 = imm5 - 32;
        buf << ", #" << imm5;
      }
      buf << endl;
      break;

    case 0b0000:  // BR
      buf << "BR";
      if (line[4] == '1') buf << 'n';
      if (line[5] == '1') buf << 'z';
      if (line[6] == '1') buf << 'p';
      PCoffset9 = bitset<9>(line.substr(7, 9)).to_ulong();
      if (line[7] == '1') PCoffset9 = PCoffset9 - 512;
      if (labels.find(PC + PCoffset9) == labels.end())
        labels[PC + PCoffset9] = "BRANCH" + to_string(++branch_count);
      buf << ' ' << labels[PC + PCoffset9] << endl;
      break;

    case 0b1100:  // JMP && RET
      /* Since the collected pieces of code do not contain all kinds of instructions,
      I leave those unused cases in blank.*/
      buf << "JMP" << endl;
      break;

    case 0b0100:  // JSR && JSRR
```

```
      // incomplete
      buf << "JSR" << endl;
      break;

    case 0b0010:  // LD
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      buf << "LD R" << DR << ", ";
      PCoffset9 = bitset<9>(line.substr(7, 9)).to_ulong();
      if (line[7] == '1') PCoffset9 = PCoffset9 - 512;
      if (labels.find(PC + PCoffset9) == labels.end())
        labels[PC + PCoffset9] = "DATA" + to_string(++data_count);
      buf << labels[PC + PCoffset9] << endl;
      break;

    case 0b1010:  // LDI
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      buf << "LDI R" << DR << ", ";
      PCoffset9 = bitset<9>(line.substr(7, 9)).to_ulong();
      if (line[7] == '1') PCoffset9 = PCoffset9 - 512;
      if (labels.find(PC + PCoffset9) == labels.end())
        labels[PC + PCoffset9] = "DATA" + to_string(++data_count);
      buf << labels[PC + PCoffset9] << endl;
      break;

    case 0b0110:  // LDR
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      BaseR = bitset<3>(line.substr(7, 3)).to_ulong();
      buf << "AND R" << DR << ", R" << BaseR;
      offset6 = bitset<5>(line.substr(11, 5)).to_ulong();
      if (line[10] == '1') offset6 = offset6 - 64;
      buf << ", #" << offset6 << endl;
      break;

    case 0b1110:  // LEA
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      buf << "LEA R" << DR << ", ";
      PCoffset9 = bitset<9>(line.substr(7, 9)).to_ulong();
      if (line[7] == '1') PCoffset9 = PCoffset9 - 512;
      if (labels.find(PC + PCoffset9) == labels.end())
        labels[PC + PCoffset9] = "DATA" + to_string(++data_count);
      buf << labels[PC + PCoffset9] << endl;
      break;

    case 0b1001:  // NOT
      if (line.substr(10) != "111111") {
        cerr << "Error: invalid line " << line << endl;
        return;
      }
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      SR1 = bitset<3>(line.substr(7, 3)).to_ulong();
      buf << "NOT R" << DR << ", R" << SR1 << endl;
      break;

    case 0b1000:  // RTI
      // incomplete
      buf << "RTI" << endl;
      break;

    case 0b0011:  // ST
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      buf << "ST R" << DR << ", ";
      PCoffset9 = bitset<9>(line.substr(7, 9)).to_ulong();
      if (line[7] == '1') PCoffset9 = PCoffset9 - 512;
      if (labels.find(PC + PCoffset9) == labels.end())
        labels[PC + PCoffset9] = "DATA" + to_string(++data_count);
      buf << labels[PC + PCoffset9] << endl;
      break;
    case 0b1011:  // STI
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      buf << "STI R" << DR << ", ";
      PCoffset9 = bitset<9>(line.substr(7, 9)).to_ulong();
      if (line[7] == '1') PCoffset9 = PCoffset9 - 512;
      if (labels.find(PC + PCoffset9) == labels.end())
        labels[PC + PCoffset9] = "DATA" + to_string(++data_count);
      buf << labels[PC + PCoffset9] << endl;
      break;

    case 0b0111:  // STR
      DR = bitset<3>(line.substr(4, 3)).to_ulong();
      BaseR = bitset<3>(line.substr(7, 3)).to_ulong();
      buf << "STR R" << DR << ", R" << BaseR;
      offset6 = bitset<5>(line.substr(11, 5)).to_ulong();
      if (line[10] == '1') offset6 = offset6 - 64;
      buf << ", #" << offset6 << endl;
      break;

    case 0b1111:  // TRAP
      // incomplete
      trapvect8 = bitset<8>(line.substr(8)).to_ulong();
      switch (trapvect8) {
        case 0x20:
          buf << "GETC";
          break;
        case 0x21:
          buf << "OUT";
          break;
        case 0x22:
          buf << "PUTS";
          break;
        case 0x23:
          buf << "IN";
          break;
        case 0x24:
          buf << "PUTSP";
          break;
        case 0x25:
          buf << "HALT";
          // Treat following lines as data
          is_data = true;
          break;
        default:
          buf << "TRAP x" << hex << trapvect8;
          break;
      }
      buf << endl;
      break;

    case 0b1101:  // reserved
      buf << ".FILL x" << hex << uppercase << bitset<16>(line).to_ulong()
          << endl;
      break;
  }
}
buf << ".END";
in.close();

// Output with label
getline(buf, line);
out << line << endl;
PC = 0x3000;
while (!buf.eof()) {
  getline(buf, line);
  if (labels.find(PC) != labels.end()) out << labels[PC] << ' ';
```

```
    out << line << endl;
    PC++;
  }
  out.close();
}

// Main
int main(int argc, char* argv[]) {
  if (argc != 2) return 0;
  string filename(argv[1]);
  auto p = filename.find_last_of('.');
  string name = p == string::npos ? filename : filename.substr(0, p);
  string out_filename(name + ".asm");
  disassemble(filename, out_filename);
  return 0;
}
```

Execute the disassembly program *lab3_disassembler.exe* in Windows Powershell and load *foo.txt*. The input info should be like `PS C:\> .\lab3_disassembler.exe .\foo.txt`. The output code would be stored in *foo.asm* regardless of its correctness.

Results

The machine code in *foo.txt* , i.e. *170.txt* from the zip file, was successfully translated and stored into *foo.asm*. The assembly code is shown as follows:

```

.ORIG x3000
ADD R1, R1, #1
ADD R2, R2, #1
ADD R3, R3, #2
LD R4, DATA1
NOT R5, R4
ADD R5, R5, #1
ADD R0, R0, #-2
BRnz BRANCH1
BRANCH4 ADD R0, R0, #-1
BRn BRANCH2
ADD R6, R1, R1
ADD R7, R3, R6
BRANCH3 ADD R7, R7, R4
BRzp BRANCH3
ADD R7, R7, R5
ADD R1, R2, #0
ADD R2, R3, #0
ADD R3, R7, #0
BRnzp BRANCH4
BRANCH1 BRn BRANCH5
ADD R7, R7, #1
BRANCH5 ADD R7, R7, #1
BRANCH2 HALT
DATA1 .FILL xFC00
.FILL x3A2
.FILL x4
.FILL x32
.FILL x1E
.END
```

The result has been copied into *translate.txt*.

Task 2: Guess

Despcription

Guess the owner of the program by the last 4 lines of the program.

Requirements

- Parse the owner's code in *foo.txt*.
- Write down the owner's student number in *id.txt*.

Solution

1. Enumerate all the pairs $(F(n), n)$ for $0 \leq n \leq 99$ so as to find the inverse mapping of F.
2. Create a hash table for F^{-1}. Then, implement a decoding program, which can convert the last 4 lines of the input machine code into decimal numbers and decode them as a student number based on the hash table.
3. Execute the decoding program to get the student number in *foo.txt*. Then, copy the output text into *id.txt*.

The decoding program written in C++ is as follows.

```
#include <algorithm>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <unordered_map>
#include <vector>

using namespace std;

void decode(const string& filename, const string& out_filename) {
  // Hash Table for the inverse mapping
  static const unordered_multimap<uint16_t, uint16_t> f = {
      {1, 0},    {1, 1},    {2, 2},    {2, 64},   {4, 3},    {6, 4},
      {6, 93},   {10, 5},   {18, 6},   {22, 43},  {30, 7},   {34, 36},
      {50, 8},   {50, 62},  {54, 23},  {66, 72},  {70, 53},  {82, 66},
      {86, 9},   {86, 67},  {98, 44},  {102, 65}, {118, 47}, {130, 80},
      {146, 10}, {150, 91}, {162, 52}, {166, 25}, {178, 14}, {178, 78},
      {182, 71}, {194, 88}, {210, 82}, {226, 60}, {242, 22}, {246, 11},
      {246, 95}, {258, 96}, {262, 61}, {274, 26}, {290, 68}, {294, 73},
      {306, 94}, {326, 21}, {338, 98}, {342, 35}, {354, 76}, {358, 33},
      {370, 38}, {386, 16}, {402, 42}, {406, 59}, {418, 12}, {418, 84},
      {438, 39}, {454, 69}, {470, 19}, {470, 83}, {482, 92}, {486, 81},
      {498, 54}, {502, 63}, {518, 29}, {530, 58}, {550, 41}, {566, 87},
      {626, 70}, {646, 77}, {658, 74}, {662, 27}, {678, 89}, {706, 24},
      {710, 13}, {710, 37}, {722, 18}, {726, 51}, {742, 17}, {742, 49},
      {754, 86}, {758, 31}, {770, 32}, {786, 90}, {790, 75}, {818, 30},
      {822, 55}, {834, 40}, {838, 85}, {850, 34}, {854, 99}, {870, 97},
      {886, 79}, {898, 48}, {902, 45}, {930, 20}, {934, 57}, {946, 46},
      {962, 56}, {978, 50}, {994, 28}, {1014, 15}};
  ifstream in(filename);
  if (!in) {
    cerr << "Error: cannot open file " << filename << endl;
    return;
  }
  ofstream out(out_filename, ios::app);
  if (!out) {
    cerr << "Error: cannot open file " << out_filename << endl;
    return;
  }
  vector<string> lines;
  while (!in.eof()) {
    string line;
    getline(in, line);
    line.erase(remove_if(line.begin(), line.end(), ::isspace), line.end());
    if (line.empty()) continue;
    lines.push_back(line);
  }
  int n = lines.size();
  string id = "PB";
  for (int i = 4; i > 0; i--) {
    auto range = f.equal_range(stoi(lines[n - i], nullptr, 2));
    // The first segment in a student number
    if (i == 4)
```

```
      while (range.first != range.second &&
             (range.first->second < 17 || range.first->second > 21))
        range.first++;
    if (range.first == range.second) {
      // Caused by invalid format
      id = "PB????????";
      break;
    } else if (distance(range.first, range.second) > 1)
      // If there are more than one possible values
      id += "??";
    else {
      if (range.first->second < 10) id += '0';
      id += to_string(range.first->second);
    }
  }
  out << "student number: " << id << endl;
}

int main(int argc, char* argv[]) {
  if (argc != 2) return 0;
  string filename(argv[1]);
  auto p = filename.find_last_of('.');
  string name = p == string::npos ? filename : filename.substr(0, p);
  string out_filename(name + ".out");
  decode(filename, out_filename);
  return 0;
}
```

Execute the decoding program *lab3_decoder.exe* in Windows Powershell and load *foo.txt*. The input info should be like `PS C:\> .\lab3_decoder.exe .\foo.txt`. The output student number would be appended to *foo.out*.

If the machine code does not contain valid student number with correct format, the output would be `PB???????`. If there are multiple possible values for any segment of a student number, the decoded segment would be replaced by `??`.

Results

The student number in *foo.txt* , i.e. *170.txt* from the zip file, was successfully decoded and stored into *foo.out* as `PB2003??07`, which actually stands for "PB20030807" or "PB20036207" since $F(8) = F(62) = 50$. Obviously, "PB20030807" is the right one.

The correct answer has been copied into *id.txt*.

Task 3: Optimize

Despcription

Optimize the program to improve its performance.

Requirements

- Store the assembly code in *optimize.txt*.
- Rewrite the code when it's neccessary.
- Test the given data sets $n = 24, 144, 456, 1088, 1092, 2096, 4200, 8192, 12000, 14000$.
- Evaluate the ratio of the average number of execution cycles before and after.

Solution

1. Analyze the problem and design a program with improved algorithms.
2. Write the optimized program in assembly language.
3. Write the testing program using C++ and LC3Tools API， which assembles *optimized.txt*, simulates a LC-3 machine, inputs the given test cases and checks correctness and efficiency accordingly.
4. Calculate the ratio.

The testing program written in C++ is as follows.

```cpp
#include <algorithm>
#include <chrono>
#include <iomanip>
#include <iostream>
#include <random>

#define API_VER 2
#include "console_inputter.h"
#include "console_printer.h"
#include "interface.h"

using namespace std;
using namespace lc3;

const int CASE_NUM = 10, SAMPLE_NUM = 10;
const uint16_t SAMPLE[SAMPLE_NUM] = {24,   144,  456,  1088,  1092,
                                     2096, 4200, 8192, 12000, 14000};
uint32_t print_level = 0;
bool enable_liberal_asm = false;
bool ignore_privilege = false;
uint32_t inst_limit = 1919810;
ConsolePrinter printer;
ConsoleInputter inputter;

uint16_t F(uint16_t n) {
  static uint16_t f[0x4001] = {1, 1, 2};
  return f[n] ? f[n] : f[n] = (F(n - 1) + 2 * F(n - 3)) % 1024;
}

void test(const string& filename, const string& out_filename) {
  lc3::as assembler(printer, print_level, enable_liberal_asm);
  string file = assembler.assemble(filename)->first;
  ofstream out(out_filename);
  if (!out) {
    cerr << "Error: cannot open file " << out_filename << endl;
    return;
  }
  lc3::sim simulator(printer, inputter, print_level);
  simulator.setIgnorePrivilege(ignore_privilege);
  simulator.setRunInstLimit(inst_limit);

  // Test
  uint64_t prev_count, sum = 0;
  uint16_t n, result;
  bool is_wrong = false;
  mt19937 gen(unsigned(time(0)));
  uniform_int_distribution<uint16_t> dis(0x0000, 0x4000);
  out << left;
  for (int i = 0; i < CASE_NUM; i++) {
    // Set machine state
    simulator.zeroState();
    if (!simulator.loadObjFile(file)) {
      cerr << "Error: invalid file " << filename << endl;
      out.close();
      exit(1);
    }
    n = i < SAMPLE_NUM ? SAMPLE[i] : dis(gen);
    simulator.writeReg(0, n);
    prev_count = simulator.getInstExecCount();
    // Run and check
    simulator.runUntilHalt();
    result = static_cast<uint16_t>(simulator.readReg(7));
    sum += simulator.getInstExecCount() - prev_count;
    out << "Case " << setw(4) << i + 1 << setw(8) << ("F(" + to_string(n) + ")")
        << " =    " << setw(8) << result;
    if (F(n) != result) {
      out << "Wrong answer!";
      is_wrong = true;
    }
    out << endl;
  }
  // Print result
  out << "correctness: " << (is_wrong ? "wrong" : "correct") << endl;
  out << "instruction count: " << sum << endl;
  out << "average instruction count: " << 1.0 * sum / CASE_NUM << endl;

  // Get line count
  ifstream in(filename);
  if (!in) {
    cerr << "Error: cannot open file " << filename << endl;
    exit(1);
  }
  string line;
  sum = -1;
  getline(in, line);
  while (!in.eof()) {
    getline(in, line);
    if (line != "") sum++;
  }
  out << "line count: " << sum << endl;
  out.close();
}

int main(int argc, char* argv[]) {
  if (argc != 2) return 0;
  string filename(argv[1]);
  auto p = filename.find_last_of('.');
  string name = p == string::npos ? filename : filename.substr(0, p);
  string out_filename(name + ".out");
  test(filename, out_filename);
  return 0;
}
```

Execute the testing program *lab3_tester.exe* in Windows Powershell and load *fib.asm*. The input and output info would have the following format.

```
PS C:\> .\lab3_tester.exe .\fib.asm
attempting to assemble .\fib.asm into .\fib.obj
assembly successful
Case 1   F(24)    =    ...
Case 2   F(144)   =    ...
Case 3   F(456)   =    ...
Case 4   F(1088)  =    ...
Case 5   F(1092)  =    ...
Case 6   F(2096)  =    ...
Case 7   F(4200)  =    ...
Case 8   F(8192)  =    ...
Case 9   F(12000) =    ...
Case 10  F(14000) =    ...
correctness: ...
instruction count: ...
average instruction count: ...
line count: ...
```

If the assembly program goes wrong in any testing case, the corresponding output would be like

```
Case ...   F(...)    =    ...   Wrong answer!
```

Accordingly, we are able to evaluate the correctness and the efficiency of the assembly program.

Results

Version 1

- **Basic idea: exponentiating by squaring**
  For any $0 \leq n \leq 16384$, let $f(n)$ be the congruence class $[F(n)]_{1024}$. In the ring $\mathbb{Z}/n\mathbb{Z}$, we have the recurrence relation $f(n) = f(n-1) + 2f(n-3)$, or in matrix form

$$\begin{pmatrix} f(n+2) \\ f(n+1) \\ f(n) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 10 \end{pmatrix} \cdot \begin{pmatrix} f(n+1) \\ f(n) \\ f(n-1) \end{pmatrix}$$

  Let $\mathbf{x}_n = \begin{pmatrix} f(n+2) \\ f(n+1) \\ f(n) \end{pmatrix}, \mathbf{A} = \begin{pmatrix} [1] & [0] & [2] \\ [1] & [0] & [0] \\ [0] & [1] & [0] \end{pmatrix}$. Then we have the general formula

$$\mathbf{x}_n = \mathbf{A}^n \mathbf{x}_0$$

  Therefore, what we need to do is to calculate the nth power of $\mathbf{A}$ by means of exponentiating by squaring and multiply $\mathbf{x}_0$ with $\mathbf{A}^n$. Let the exponent n be written as

$$n = \sum_{i=0}^{14} 2^i b_i$$

  Then we have the equation

$$\mathbf{x}_n = \left(\prod_{i=0}^{14} b_i \mathbf{A}^{2^i} \right) \mathbf{x}_0$$

  To optimize the performance to the fullest, it is reasonable to calculate and store every 2^ith power of $\mathbf{A}$ by other means in advance, and later load them into register for matrix multiplications.
  The optimized program is supposed to run in $O(\log(n))$ time, faster than the original *foo.txt* that runs in time of $O(n)$.

Sadly, owing to the deadline, I have not implemented this idea.

Version 2

- **Basic idea: cycle detection**
  When calculating the 2^ith power of $\mathbf{A}$ described in the fisrt version, we noticed that

$$\mathbf{A}^{14} = \mathbf{A}^{13} = \cdots = \mathbf{A}^8 = \begin{pmatrix} [197] & [634] & [590] \\ [807] & [414] & [634] \\ [317] & [490] & [414] \end{pmatrix}$$

  This fact indicates that $\mathbf{x}_n$ and $F(n)$ must continue periodically after finite iterations.
  Actually, it is not hard to prove that $\mathbf{x}_n$ must have a pre-period μ and a period λ, such that for any $i > 0$,

$$\mathbf{x}_{\mu+i} = \mathbf{x}_{\mu+i+\lambda}$$

  Since there are 1024 possible congruence classes for each component of $\mathbf{x}_n$ as $F(n)$, $F(n+1)$ or $F(n+2)$, there are finitely 1024^3 possible combinations for $\mathbf{x}_n$, which means that the sequence $\mathbf{x}_n$ must reappear after at most 1024^3 iterations, according to the pigeonhole principle. Given the recurrence relation, we know that the sequence must repeat periodically.
  The task, then, is to find the minimal μ and λ for the sequence $\mathbf{x}_n$ through cycle detection algorithms such as Floyd's tortoise and hare, i.e., using two pointers which move through the sequence at different speeds to detect the cycle. After some easy coding, we have the results that

$$\mu = 19,\ \lambda = 128$$

  Based on the results, we simply convert the first 148 elements of the sequence into `.FILL` pseudo commands, and complete the assembly code with the following branch structure:

$$F(n) = \begin{cases} F((n-20)\ mod\ 128 + 20), & n \geq 20 \\ F(n), & n < 20 \end{cases}$$

  The optimized program is supposed to run in $O(1)$ time, much faster than the original *foo.txt* that runs in time of $O(n)$.
- **Assembly code**
  The optimized assembly code, stored in *optimized.txt*, is shown as follows:

```
.ORIG x3000
```

```
LEA R1, MAP
ADD R2, R0, #-10
ADD R2, R2, #-10
BRn OK
LD R3, BITMASK
AND R0, R2, R3
ADD R0, R0, #10
ADD R0, R0, #10
OK ADD R1, R1, R0
LDR R7, R1, #0
HALT
BITMASK .FILL #127
MAP .FILL #1
.FILL #1
.FILL #2
.FILL #4
.FILL #6
.FILL #10
.FILL #18
.FILL #30
.FILL #50
.FILL #86
.FILL #146
.FILL #246
.FILL #418
.FILL #710
.FILL #178
.FILL #1014
.FILL #386
.FILL #742
.FILL #722
.FILL #470
.FILL #930
.FILL #326
.FILL #242
.FILL #54
.FILL #706
.FILL #166
.FILL #274
.FILL #662
.FILL #994
.FILL #518
.FILL #818
.FILL #758
.FILL #770
.FILL #358
.FILL #850
.FILL #342
.FILL #34
.FILL #710
.FILL #370
.FILL #438
.FILL #834
.FILL #550
.FILL #402
.FILL #22
.FILL #98
.FILL #902
.FILL #946
.FILL #118
.FILL #898
.FILL #742
.FILL #978
.FILL #726
.FILL #162
.FILL #70
.FILL #498
```

```
.FILL #822
.FILL #962
.FILL #934
.FILL #530
.FILL #406
.FILL #226
.FILL #262
.FILL #50
.FILL #502
.FILL #2
.FILL #102
.FILL #82
.FILL #86
.FILL #290
.FILL #454
.FILL #626
.FILL #182
.FILL #66
.FILL #294
.FILL #658
.FILL #790
.FILL #354
.FILL #646
.FILL #178
.FILL #886
.FILL #130
.FILL #486
.FILL #210
.FILL #470
.FILL #418
.FILL #838
.FILL #754
.FILL #566
.FILL #194
.FILL #678
.FILL #786
.FILL #150
.FILL #482
.FILL #6
.FILL #306
.FILL #246
.FILL #258
.FILL #870
.FILL #338
.FILL #854
.FILL #546
.FILL #198
.FILL #882
.FILL #950
.FILL #322
.FILL #38
.FILL #914
.FILL #534
.FILL #610
.FILL #390
.FILL #434
.FILL #630
.FILL #386
.FILL #230
.FILL #466
.FILL #214
.FILL #674
.FILL #582
.FILL #1010
.FILL #310
.FILL #450
.FILL #422
```

```
.FILL #18
.FILL #918
.FILL #738
.FILL #774
.FILL #562
.FILL #1014
.FILL #514
.FILL #614
.FILL #594
.FILL #598
.FILL #802
.FILL #966
.FILL #114
.FILL #694
.FILL #578
.FILL #806
.FILL #146
.FILL #278
.FILL #866
.FILL #134
.FILL #690
.FILL #374
.FILL #642
.FILL #998
.FILL #722
.FILL #982
.END
```

- **Output**

foo.txt:

```


Case 1   F(24)    =    706
Case 2   F(144)   =    642
Case 3   F(456)   =    66
Case 4   F(1088)  =    2
Case 5   F(1092)  =    290
Case 6   F(2096)  =    898
Case 7   F(4200)  =    322
Case 8   F(8192)  =    514
Case 9   F(12000) =    258
Case 10  F(14000) =    898
correctness: correct
instruction count: 561728
average instruction count: 56172.8
line count: 28
```

optimized.txt:

```


Case 1   F(24)    =    706
Case 2   F(144)   =    642
Case 3   F(456)   =    66
Case 4   F(1088)  =    2
Case 5   F(1092)  =    290
Case 6   F(2096)  =    898
Case 7   F(4200)  =    322
Case 8   F(8192)  =    514
Case 9   F(12000) =    258
Case 10  F(14000) =    898
correctness: correct
instruction count: 110
average instruction count: 11
line count: 160
```

- **Assessment**

  - Correctness: ✓(before) -> ✓(after)
  - Number of lines: 28(before) -> 160(after)
  - Average number of instructions: 56172.8(before) -> 11(after)
  - Ratio of instructions: $\frac{11}{56172.8} \approx 0.0001958$
- **Analysis**
  The program in *optimized.txt* correctly gets the value of $F(n)$ for all 10 cases. Compared with *foo.txt*, the average number of instructions is extraordinarily reduced.

Bonus

The following Windows PowerShell script automatically disassembles all the *.txt* files provided by TA, tests them, decodes the student numbers and sorts them by line count in ascending order.

```
Get-ChildItem .\zips\*\*.txt |
ForEach-Object {
    $Pg = Get-Item $_.FullName.Replace('.txt', '.asm')
    .\lab3_disassembler.exe $_.FullName
    .\lab3_tester.exe $Pg.FullName
    .\lab3_decoder.exe $_.FullName
    $Out = Get-Content $_.FullName.Replace('.txt', '.out')
    $Props = @{
        LineCount   = 0
        InstCount   = "???"
        Correctness = "invalid"
        StudentID   = "PB????????"
    }
    if ($Out.Count -gt 1) {
        $Props.StudentID = $Out[-1] -replace 'student number: '
        $Props.LineCount = $Out[-2] -replace 'line count: '
        $Props.InstCount = $Out[-3] -replace 'average instruction count: '
        $Props.Correctness = $Out[-5] -replace 'correctness: '
    }
    else {
        $Props.StudentID = $Out -replace 'student number: '
        $Props.LineCount = (Get-Content $Pg.FullName).Count - 2
    }
    $Pg | Add-Member -NotePropertyMembers $Props -PassThru
} |
Sort-Object -Property Correctness, LineCount, InstCount |
Format-Table -Property Name, StudentID, Correctness, LineCount, InstCount
```

The output is as follows.

```

Name    StudentID  Correctness LineCount InstCount
----    ---------  ----------- --------- ---------
6.asm   PB2015??66 correct     17        25981.2
75.asm  PB19??05?? correct     18        30310.4
120.asm PB200307?? correct     18        30310.4
64.asm  PB20????65 correct     18        34638.6
66.asm  PB20????40 correct     18        34638.6
62.asm  PB20051036 correct     18        34646.6
112.asm PB20??1632 correct     19        34633.6
193.asm PB19??03?? correct     19        34639.6
136.asm PB2006??26 correct     19        34639.6
147.asm PB20??1651 correct     19        34639.6
74.asm  PB20??1652 correct     19        38969.8
168.asm PB20??1631 correct     20        25972.2
180.asm PB20??1655 correct     20        30305.4
107.asm PB???????? correct     20        38968.8
102.asm PB20??1697 correct     20        38977.8
157.asm PB20??1642 correct     21        25972.2
123.asm PB2006??72 correct     21        30299.4
```

```
51.asm  PB2006??88 correct     21        30313.4
15.asm  PB2015???? correct     21        34625.6
142.asm PB20??1554 correct     21        34625.6
125.asm PB???????? correct     21        34625.6
79.asm  PB2003??66 correct     21        38962.8
78.asm  PB20?????? correct     22        25974.2
126.asm PB20??1626 correct     22        30298.4
37.asm  PB20?????? correct     22        30299.4
166.asm PB20?????? correct     22        30300.4
122.asm PB20??1686 correct     22        30300.4
8.asm   PB2015???? correct     22        30301.4
31.asm  PB20051046 correct     22        34624.6
135.asm PB???????? correct     22        34624.6
192.asm PB20????96 correct     22        34625.6
49.asm  PB2003???? correct     22        34626.6
98.asm  PB20????70 correct     22        34626.6
67.asm  PB20??16?? correct     22        34626.6
72.asm  PB???????? correct     22        34627.6
183.asm PB???????? correct     22        34633.6
2.asm   PB2006??56 correct     22        34640.6
152.asm PB20??1696 correct     22        39016.8
14.asm  PB20??1672 correct     23        30299.4
195.asm PB20??1669 correct     23        30299.4
106.asm PB20??1620 correct     23        30300.4
5.asm   PB20511877 correct     23        30301.4
10.asm  PB2006??74 correct     23        34627.6
196.asm PB20??1589 correct     23        34627.6
83.asm  PB20?????? correct     23        34627.6
164.asm PB20??1660 correct     23        38953.8
35.asm  PB20051086 correct     23        38954.8
141.asm PB2006??23 correct     23        38954.8
19.asm  PB1905???? correct     23        69135.3
121.asm PB20??1644 correct     24        30300.4
191.asm PB2005???? correct     24        30300.4
113.asm PB2005???? correct     24        34624.6
145.asm PB2005??69 correct     24        34625.6
30.asm  PB2006??07 correct     24        34626.6
146.asm PB???????? correct     24        34628.6
41.asm  PB20??1665 correct     24        38948.8
172.asm PB2003??98 correct     24        38954.8
176.asm PB18??0691 correct     24        38956.8
29.asm  PB???????? correct     24        56124.9
20.asm  PB2006??63 correct     24        64747.4
156.asm PB20?????? correct     25        30301.4
92.asm  PB20??1653 correct     25        34624.6
21.asm  PB2005??59 correct     25        34625.6
0.asm   PB2005??59 correct     25        34625.6
7.asm   PB20??1685 correct     25        34625.6
178.asm PB2005??77 correct     25        34626.6
1.asm   PB20??1654 correct     25        34628.6
119.asm PB2006??10 correct     25        38952.8
58.asm  PB20??1666 correct     25        43283
103.asm PB20151805 correct     25        51864.8
16.asm  PB20????07 correct     25        56193
155.asm PB20611816 correct     26        30299.4
181.asm PB20??1627 correct     26        34626.6
118.asm PB20611836 correct     26        34627.6
17.asm  PB2006??55 correct     26        34627.6
87.asm  PB20??1689 correct     26        34627.6
110.asm PB???????? correct     26        38948.8
160.asm PB2006??68 correct     26        38951.8
177.asm PB18??1696 correct     26        43280
109.asm PB20030770 correct     26        43285
117.asm PB18071566 correct     26        47519.4
158.asm PB20??16?? correct     26        51933.4
63.asm  PB20051052 correct     26        56171.8
23.asm  PB20??1679 correct     27        34627.6
```

```
104.asm PB20??1680 correct     27        34714.4
154.asm PB20??1690 correct     27        38947.8
11.asm  PB20????16 correct     27        38949.8
138.asm PB2015???? correct     27        38949.8
101.asm PB20??1634 correct     27        38950.8
128.asm PB20????50 correct     27        38953.8
82.asm  PB18????03 correct     27        38953.8
151.asm PB20????29 correct     27        38961.8
189.asm PB20??16?? correct     27        43275
108.asm PB20??06?? correct     27        43276
89.asm  PB20511882 correct     27        43277
162.asm PB1906???? correct     27        47610.2
59.asm  PB20??1643 correct     27        51940.4
26.asm  PB20??1648 correct     28        34626.6
114.asm PB18????54 correct     28        34628.6
129.asm PB20??16?? correct     28        34634.6
57.asm  PB20??1699 correct     28        34657.6
77.asm  PB20????05 correct     28        38950.8
65.asm  PB2006??86 correct     28        38954.8
39.asm  PB20?????? correct     28        38965.8
144.asm PB20??1698 correct     28        43274
131.asm PB???????? correct     28        51844.6
170.asm PB2003??07 correct     28        56172.8
91.asm  PB2003??99 correct     29        34625.6
185.asm PB20??15?? correct     29        34625.6
159.asm PB20??1625 correct     29        34628.6
194.asm PB18?????? correct     29        34633.6
69.asm  PB20?????? correct     29        34650.6
4.asm   PB2006??54 correct     29        39322.8
50.asm  PB206118?? correct     29        43314
70.asm  PB???????? correct     29        47518.4
28.asm  PB???????? correct     30        38948.8
12.asm  PB20??1586 correct     30        43285
53.asm  PB20??1694 correct     30        51844.6
85.asm  PB2005??81 correct     30        51846.6
18.asm  PB20??1646 correct     30        51933.4
45.asm  PB20030574 correct     30        56080
93.asm  PB20?????? correct     30        60406.2
163.asm PB2015??38 correct     30        60684.6
94.asm  PB2015??96 correct     31        43279
133.asm PB19??1640 correct     31        51843.6
22.asm  PB20??0380 correct     31        51845.6
43.asm  PB20??0392 correct     31        51845.6
105.asm PB20??1650 correct     31        56171.8
132.asm PB20?????? correct     31        60685.6
174.asm PB20????10 correct     32        38947.8
40.asm  PB2003??33 correct     32        38958.8
173.asm PB20611833 correct     32        43317
169.asm PB20611826 correct     32        43317
76.asm  PB20??1630 correct     32        43317
71.asm  PB2006???? correct     32        47603.2
96.asm  PB2006???? correct     32        60407.2
97.asm  PB200510?? correct     32        64823.2
32.asm  PB20????31 correct     33        23089.8
84.asm  PB???????? correct     33        25972.6
73.asm  PB???????? correct     33        34633.6
81.asm  PB19????63 correct     33        38979.8
140.asm PB20??1628 correct     33        43280
134.asm PB20??1661 correct     33        43283
127.asm PB20??1661 correct     33        43283
111.asm PB20??1691 correct     33        43289
116.asm PB20??16?? correct     33        47612.2
13.asm  PB20??16?? correct     33        60402.2
186.asm PB20????69 correct     34        56174.8
184.asm PB20??0381 correct     34        60455.1
3.asm   PB2006??39 correct     34        77667.1
46.asm  PB20051094 correct     35        23087.8
```

```
100.asm PB2015??87 correct     36        18770.8
153.asm PB20?????? correct     36        25971.2
179.asm PB???????? correct     36        56267.6
88.asm  PB2003???? correct     36        60507
188.asm PB2006??60 correct     36        73481.6
68.asm  PB2006??61 correct     37        73482.6
137.asm PB19??1643 correct     39        43114.8
130.asm PB2006??57 correct     40        43115.8
139.asm PB20511861 invalid     18        ???
99.asm  PB18????30 invalid     21        ???
148.asm PB20?????? invalid     22        ???
56.asm  PB20????07 invalid     26        ???
55.asm  PB19????56 invalid     26        ???
9.asm   PB20511898 invalid     26        ???
33.asm  PB20??1688 invalid     31        ???
95.asm  PB19?????? invalid     32        ???
80.asm  PB201550?? invalid     33        ???
167.asm PB20??1624 invalid     33        ???
44.asm  PB???????? invalid     34        ???
61.asm  PB20??1622 invalid     35        ???
171.asm PB20??1692 invalid     36        ???
187.asm PB20??1687 invalid     36        ???
165.asm PB20511897 invalid     37        ???
42.asm  PB???????? wrong       26        201
24.asm  PB20??1641 wrong       30        61017.4
52.asm  PB???????? wrong       34        30975.4
182.asm PB???????? wrong       35        1.91981e+06
27.asm  PB???????? wrong       36        1.91981e+06
115.asm PB???????? wrong       36        1.91981e+06
38.asm  PB???????? wrong       43        59509.5
```

The author's program is found as *123.txt*.