시스템 소프트웨어와 실습 과제

2016110056 불교학부 박승원
2016년 9월 21일

```
//1번 문제 어셈블리어 코드.
   start 1000
first lda seven
   sta alpha
   lda two
   add incr
   sta beta
   lda gamma
   sub two
   sta delta
   ldch charx
   stch cha
seven word 7
two word 2
alpha resw 1
beta resw 1
gamma word 10
delta resw 1
incr word 3
cha resb 1
charx byte 78
   end first
```

```
//1번 문제 컴파일 후의 목적화일
start 1000
data 101e
end 1035
1000 00101e
1003 0c1024
1006 001021
1009 181030
100c 0c1027
100f 00102a
1012 1c1021
1015 0c102d
1018 e41034
101b e81033
101e 000007
1021 000002
1024 000000
1027 000000
102a 000010
102d 000000
1030 000003
1033 00
1034 78
1035 00
```

```
//2번 문제 어셈블리어 파일
   start 1000
first lda data1
   add data2
   sta sum
   lda data1
   sub data2
   sta diff
data1 word 8
data2 word 7
sum resw 1
diff resw 1
   end first
```

```
//2번 문제 컴파일 후의 목적 화일
start 1000
data 1012
end 101e
1000 001012
1003 181015
1006 0c1018
1009 001012
100c 1c1015
100f 0c101b
1012 000008
1015 000007
1018 000000
101b 000000
101e 00
```

```
1번 문제 심볼 테이블 주소
alpha : 1024
beta : 1027
cha : 1033
charx : 1034
data_begin : 101e
delta : 102d
end : 1035
first : 1000
gamma : 102a
incr : 1030
seven : 101e
start : 1000
two : 1021

1번 문제 실행 전후의 메모리 데이터 영역
```

```
000007 000002 000000 000000 000010 000000 000003 0078
000007 000002 000007 000005 000010 00000e 000003 7878
seven two  alpha beta gamma delta incr cha charx

2번  문제  실행  전후의  메모리  데이터  영역
000008 000007 000000 000000
000008 000007 00000f 000001
data1 data2 sum diff
```

직접 수업시간에 들은 내용을 토대로 컴파일러와 실행인터프리터를 구현해 보았다. SIC시뮬레이터와 동일한 결과를 얻었다. 그 소스 코드를 첨부한다.

```
//sic/xe simulator 헤더파일입니다.
#include<vector>
#include<functional>
#include<string>
#include<map>

struct Register//24 bit
{
   unsigned char opcode;
   short address;//1 bit(direct0, indexed 1) + 15 bit address
   Register& operator=(int n);
   operator int();
};

//typedef char byte;//byte c'z'
//typedef char[3] word;//상수 정의 five(addr) word 5

class SIC
{
public:
   Register A, X, L, PC, SW;
   //pseudo instruction
   void start();
   void end();
   void resw();//1word space allocate
   void resb();//alpha resb 2; 2byte alloca

   //load and store
   void LDA(short address);
   void LDX(short);
   void STA(short address);
   void STX(short);

   //arithmetic
   void ADD(short);
   void SUB(short);
   void MUL(short);
   void DIV(short);

   //comparison
   void COMP(short);

   //conditional jump
   void JLT(short);
   void JEQ(short);
   void JGT(short);

   //subroutine linkage
   void JSUB(short);
   void RSUB(short);

   //IO. test divice. read write device
   void TD(short);
   void RD(short);
   void WD(short);

   //char handle
   void LDCH(short);
   void STCH(short);

protected:
   unsigned char memory[32768];//2**15
   bool is_opcode(std::string s);
   int fetch(short addr) const;

   std::map<std::string, unsigned char> op_table = {
      {"lda", 0x00}, {"ldx", 0x04}, {"sta", 0x0c}, {"stx", 0x10},
      {"add", 0x18}, {"sub", 0x1c}, {"mul", 0x20}, {"div", 0x24},
      {"comp", 0x28}, {"jeq", 0x30}, {"jgt", 0x34}, {"jlt", 0x38},
      {"jsub", 0x48}, {"rsub", 0x4c},
      {"rd", 0xd8}, {"wd", 0xdc}, {"td", 0xe0},
      {"ldch", 0xe4}, {"stch", 0xe8}
   };

   std::map<unsigned char, std::function<void(short)>> po_table = {
      {0x00, std::bind(&SIC::LDA, this, std::placeholders::_1)},
      {0x04, std::bind(&SIC::LDX, this, std::placeholders::_1)},
      {0x0c, std::bind(&SIC::STA, this, std::placeholders::_1)},
      {0x10, std::bind(&SIC::STX, this, std::placeholders::_1)},
      {0x18, std::bind(&SIC::ADD, this, std::placeholders::_1)},
      {0x1c, std::bind(&SIC::SUB, this, std::placeholders::_1)},
      {0x20, std::bind(&SIC::MUL, this, std::placeholders::_1)},
      {0x24, std::bind(&SIC::DIV, this, std::placeholders::_1)},
      {0x28, std::bind(&SIC::COMP, this, std::placeholders::_1)},
      {0x30, std::bind(&SIC::JEQ, this, std::placeholders::_1)},
      {0x34, std::bind(&SIC::JGT, this, std::placeholders::_1)},
      {0x38, std::bind(&SIC::JLT, this, std::placeholders::_1)},
      {0x48, std::bind(&SIC::JSUB, this, std::placeholders::_1)},
      {0x4c, std::bind(&SIC::RSUB, this, std::placeholders::_1)},
      {0xd8, std::bind(&SIC::RD, this, std::placeholders::_1)},
      {0xdc, std::bind(&SIC::WD, this, std::placeholders::_1)},
      {0xe0, std::bind(&SIC::TD, this, std::placeholders::_1)},
      {0xe4, std::bind(&SIC::LDCH, this, std::placeholders::_1)},
      {0xe8, std::bind(&SIC::STCH, this, std::placeholders::_1)}
   };

private:
};
```

```
//SIC/XE 구현부
```

```
#include"sic.h"
using namespace std;

void SIC::LDA(short addr)
{
   A = fetch(addr);
}

void SIC::STA(short addr)
{
   memory[addr] = A.opcode;
   memory[addr + 1] = A.address >> 8;
   memory[addr + 2] = A.address & 255;
}

bool SIC::is_opcode(string s)
{
   return op_table.find(s) != op_table.end();
}

Register& Register::operator=(int n)
{
   address = n & 65535;
   n <<= 8;
   n >>= 24;
   opcode = n;
   return *this;
}

Register::operator int()
{
   int tmp = opcode;
   tmp <<= 16;
   tmp |= address;
   return tmp;
}

int SIC::fetch(short addr) const
{
   int r = memory[addr];
   r <<= 16;
   int k = memory[addr + 1];
   k <<= 8;
   r |= k;
   r |= memory[addr + 2];
   return r;
}

void SIC::LDX(short addr)
{
   X = fetch(addr);
}
void SIC::STX(short addr)
{
   memory[addr] = X.opcode;
   memory[addr + 1] = X.address >> 8;
   memory[addr + 2] = X.address & 255;
}
void SIC::ADD(short addr)
{
   int n = fetch(addr);
   int a = A;
   A = a + n;
}
void SIC::SUB(short addr)
{
   int n = fetch(addr);
   int a = A;
   A = a - n;
}

void SIC::LDCH(short addr)
{
   A = memory[addr];
}

void SIC::STCH(short addr)
{
   memory[addr] = A.address & 255;
}

void SIC::MUL(short addr)
{
   int n = fetch(addr);
   int a = A;
   A = a * n;
}
void SIC::DIV(short addr)
{
   int n = fetch(addr);
   int a = A;
   A = a / n;
}
void SIC::COMP(short addr) {}
void SIC::JEQ(short addr) {}
void SIC::JGT(short addr) {}
void SIC::JLT(short addr) {}
void SIC::JSUB(short addr) {}
void SIC::RSUB(short addr) {}
void SIC::RD(short addr) {}
void SIC::WD(short addr) {}
void SIC::TD(short addr) {}
```

```
// 컴파일러   헤더파일 .  소오스를  .o로  만든다 .
#include"sic.h"

class Compiler : public SIC
{
public:
   Compiler(std::string filename);

protected:
```

```
   void make_sym_table();
   void make_obj_code();
   bool is_symbol(std::string s);
   void create_object(std::string file);
   void fill_instructions(std::string file);
   void show_sym_table();

   std::map<std::string, short> sym_table;
   std::vector<std::array<std::string, 3>> instructions;
   std::vector<std::pair<short, std::vector<unsigned char>>> obj_code;
};
```

```
// 컴파일러  구현부 .
#include<iostream>
#include<iomanip>
#include<string>
#include<fstream>
#include"compiler.h"
using namespace std;

Compiler::Compiler(string file)
{
   fill_instructions(file);
   make_sym_table();
   make_obj_code();
   file.back() = 'o';
   create_object(file);
   show_sym_table();
}

void Compiler::show_sym_table()
{
   for(auto& a : sym_table) cout << a.first << " : " << hex << a.second <<
       endl;
}

void Compiler::create_object(string file)
{
   ofstream f(file);
   f << "start " << hex << setfill('0') << setw(4) << sym_table["start"] <<
       endl;
   f << "data " << hex << setfill('0') << setw(4) << sym_table["data_begin"]
       << endl;
   f << "end " << hex << setfill('0') << setw(4) << sym_table["end"] << endl;
   for(auto& a : obj_code) {
      f << hex << setfill('0') << setw(4) << a.first << ' ' ;
      for(auto& b : a.second) f << hex << setfill('0') << setw(2) << +b;
      f << endl;
   }
}

void Compiler::make_obj_code()
{
   short addr = sym_table["start"];
   bool started = false;
   for(auto& a : instructions) {
      if(addr < sym_table["data_begin"]) {
         if(started) {
            short two_part;
            if(is_symbol(a[2])) two_part = sym_table[a[2]];
            else two_part = stoi(a[2], nullptr, 16);
            obj_code.push_back({addr, {+op_table[a[1]], two_part >> 8,
                two_part & 255}});
            addr += 3;
         }
         if(a[1] == "start") started = true;
      } else if(addr == sym_table["end"]) {
         obj_code.push_back({addr, {0}});
         break;
      } else {
         int k = stoi(a[2], nullptr, 16);
         if(a[1] == "word") {
            obj_code.push_back({addr, {k >> 16, (k >> 8) & 255, k & 255}});
            addr += 3;
         } else if(a[1] == "byte") {
            obj_code.push_back({addr, {k}});
            addr++;
         } else if(a[1] == "resb") {
            vector<unsigned char> v;
            v.resize(k);
            obj_code.push_back({addr, v});
            addr += k;
         } else if(a[1] == "resw") {
            vector<unsigned char> v;
            v.resize(3 * k);
            obj_code.push_back({addr, v});
            addr += 3 * k;
         }
      }
   }
}

bool Compiler::is_symbol(string s)
{
   return sym_table.find(s) != sym_table.end();
}

void Compiler::make_sym_table()
{//심볼  테이블을  만든다 .
   int addr = 0;
   bool data_begin = false;
   for(auto& a : instructions) {
      if(a[0] != "") sym_table[a[0]] = addr;
      if(a[1] == "start") {
         addr = stoi(a[2], nullptr, 16);
         sym_table["start"] = addr;
      }
      else if(a[1] == "end") sym_table["end"] = addr;
      else if(is_opcode(a[1])) addr += 3;
      else {
         if(!data_begin) {
            sym_table["data_begin"] = addr;
```

```
            data_begin = true;
         }
         if(a[1] == "word") addr += 3;
         else if(a[1] == "byte") addr++;
         else if(a[1] == "resb") addr += stoi(a[2]);
         else if(a[1] == "resw") addr += 3 * stoi(a[2]);
      }
   }
}

void Compiler::fill_instructions(string file)
{
   ifstream f(file);
   char c;
   string com[3];
   int n = 0;
   bool sp_flag = false;
   while(f >> noskipws >> c) {
      if(c != '\n') {
         if(c == '\t' || c == ' ') {
            if(!sp_flag) n++;
            sp_flag = true;
         } else {
            com[n] += c;
            sp_flag = false;
         }
      } else {
         instructions.push_back({com[0], com[1], com[2]});
         n = 0;
         for(int i=0; i<3; i++) com[i].clear();
      }
   }
}
```

```
// 컴파일러가 만든 .o를 실행하는 인터프리터 헤더파일
#include"sic.h"

class Interpreter : public SIC
{
public:
   Interpreter(std::string file);
   void show_mem();

protected:
   int start, end, data_begin;
   void load_to_memory(std::string file);
   void execute();

private:
};
```

```
// 인터프리터 구현부 .
#include<fstream>
#include<iostream>
#include<iomanip>
#include"interpreter.h"
using namespace std;

Interpreter::Interpreter(string file)
{
   load_to_memory(file);
   show_mem();
   while(PC < data_begin) execute();
   show_mem();
}

void Interpreter::load_to_memory(string file)
{
   ifstream f(file);
   string s;
   char c[5];
   int addr, mnemonic;
   f >> s >> hex >> start >> s >> hex >> data_begin >> s >> hex >> end;
   addr = start;
   while(f >> s >> s) {
      for(int i=0; i<s.size(); i+=2) {
         string n;
         n += s[i];
         n += s[i+1];
         memory[addr++] = stoi(n, nullptr, 16);
      }
   }
   PC = start;
}

void Interpreter::show_mem()
{
   for(int i = data_begin, j = 0; i < end; i++, j++) {
      cout << setfill('0') << setw(2) << hex << +memory[i];
      if(j % 3 == 2) cout << ' ';
   }
   cout << endl;
}

void Interpreter::execute()
{
   short operand = memory[PC + 1];
   operand <<= 8;
   operand |= memory[PC + 2];
   po_table[memory[PC]](operand);
   PC.address += 3;
}
```

```
#include<iostream>
#include"compiler.h"
using namespace std;

int main(int argc, char** argv)
{
   if(argc < 2) {
      cout << "usage : " << argv[0] << "[file to compile]" << endl;
```

```
      return 0;
   }
   Compiler cp(argv[1]);
}
```

```
#include<iostream>
#include"interpreter.h"
using namespace std;

int main(int c, char** v)
{
   if(c < 2) {
      cout << "usage : " << v[0] << " [file to run .o]" << endl;
      return 0;
   }
   Interpreter inter(v[1]);
}
```

소감

SIC 에뮬레이터의 연습을 통해 컴퓨터의 작동원리를 파악할 수 있었다. 에뮬레이터는 굉장히 어려운 프로그래밍으로 생각했고, 하드웨어적인 지식이 많아야 하는 것으로 알고 있었는데, 직접 컴파일러를 작성해 보니 에뮬레이터가 생각보다 쉽게 구현이 되는 것에 놀랐다. 하드웨어가 단순한 원리에 의해서 움직인다는 교수님의 말씀을 이해할 수 있었다.

수업을 통해 정확한 지식을 얻고 작성해 보니, 어렴풋이 알고 있던 것들이 명확해졌고, 매우 성취감이 컸다. 지금은 아직 조건분기 등의 구현과 문자 데이터의 처리등은 만들지 않았지만, 수업의 진도 대로 차근차근 만들어 보도록 해야겠다. 클래스를 만들며 수업을 들으니 SIC의 작동원리를 구체적으로 이해할 수 있게 되는 것 같다.