숭실대학교 Soongsil University

시스템프로그래밍 2020 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.

1.1. 나는 동료의 보고서를 베끼지 않았습니다.

1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.

2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히 인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)

3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.

4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

과목	시스템프로그래밍 2020
과제명	Project#2 SIC/XE Machine Simulator 구현
담당교수	최 재 영 교 수
제출인	*전자정보공학부 20160458 김지우* 컴퓨터 학부수업 (출석번호 109번)
제출일	2020년 6월 4일

INDEX

1. 프로젝트 개요

1.1. 개발 배경 및 목적

현재 시스템 프로그래밍을 수강하며 Assembler, Loader의 동작 및 Object program을 메모리에 올리고 실행하는 절차를 배우는 중이다. 개발자라면 이러한 코드들이 어떠한 의미를 가지는지, 잘못 되었다면 어떤 부분이 잘못된 것인지 알 필요가 있다. 개발자의 꿈을 가진 사람으로써, Assembler 및 Loader를 설계하고 구현하는 것은 큰 의미가 있는 일일 것이다. 프로그램의 동작에 더 나은 이해를 갖기 위해, 이 프로젝트를 시작하게 되었다.

이번 프로젝트의 목적은 SIC/XE 머신의 동작을 수행할 시뮬레이터를 개발하는 것이다. Object code가 입력으로 들어가면, 해당 Object code에 따른 프로그램 실행의 결과가 출력으로 나타난다.

SIC/XE 명령어를 실제로 처리하는 동작을 구현해야 할 뿐만 아니라, GUI를 만들어 Ojbect Code별 모니터링을 할 수 있도록 하고, 단계별 Instruction 수행 과정과 Register를 모니터링 할 수 있도록 구현한다.

이 프로젝트를 구현함으로써 Loader와 실행하는 Simulator의 동작에 대해 더 높은 이해를 갖는 것 또한 목적이다. object program code를 실제로 올리고, 비트 단위로 읽어 opcode, nixbpe, disp 등으로 변환해봄으로써 SIC/XE Simulator의 동작을 이해한다.

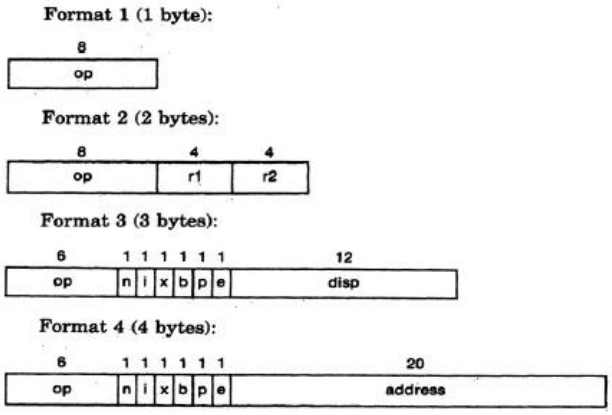
2. 배경 지식

2.1. 주제에 관한 배경지식

- **SIC/XE Machine**

SIC는 Simplified Instructional Computer의 줄임말로 일반적인 하드웨어의 특성은 다 있는 간단한 컴퓨터이다. SIC Machine의 확장형이다. SIC 머신은 Addressing Mode가 Direct mode밖에 없는 반면, SIC/XE 머신은 Relative mode를 주로 쓴다.

- **Instruction Format**



[그림 2-1] Instruction Format

SIC/XE Machine은 명령어의 형식이 4가지가 존재하고, 각각의 길이가 다르다.

1형식: opcode 1byte만 존재한다.

2형식: opcode 1byte와 레지스터 두 개 각각 4bit씩 총 2byte이다.

3형식: opcode 6bit와 nixbpe 6bit, disp(상대주소) 12bit 총 3byte이다.

4형식: opcode 6bit와 nixbpe 6bit, address(보통 절대주소) 20bit 총 4byte이다. 4형식은 다른 형식들과 다르게 코드 operator앞에 +를 붙여 4형식임을 표현한다.

- **Relocation**

Modification Record를 이용해서 메모리에서 수정해야 할 주소들을 수정한다.

- **Linking Loader**

Absolute loader와 다르게 control section으로 나누어진 프로그램이 linking되어야 한다. External Reference를 처리해야 하기 때문에 1pass로 되지 않고, 2pass로 처리 되어야 한다.

Pass 1 : External Symbol들에게 주소를 모두 할당한다(Table 만듦)

Pass 2 : 실제 Loading, Relocating, Linking을 한다.

- **Data Structures**

ESTAB(External symbol table): SYMTAB과 비슷하다. External symbol들의 이름과 주소 값을 저장해놓는다. 이 때는 실제 메모리에 올라간 절대 주소가 저장된다.

PROGADDR(program load address): 프로그램의 메모리 시작 주소를 저장해놓는다. 이 주소도 마찬가지로 메모리에 올라간 절대 주소가 저장된다.

CSADDR(control section address): 각각 control section의 시작 주소가 저장된다.

- **Addressing Mode와 nixbpe**

Direct mode: 보통 가장 많이 쓰는 형식으로 해당 주소로 가는 것을 뜻한다. n=1, i=1로 설정한다. 만약, n=0, i=0일 경우 SIC 머신 명령어임을 표현한다. operand가 MAXLEN일 경우, MAXLEN에 해당하는 주소가 target address인데 SIC/XE machine은 절대 주소보다 상대 주소가 기반이기 때문에 먼저 PC relative로 계산하고, 안 된다면 Base relative로, 그것도 안 된다면 4형식으로 확장한다. 4형식의 경우 e=1로 설정한다.

@(Indirect Addressing mode): n=1, i=0으로 설정한다. @뒤의 값의 주소로 가서, 그 주소에 있는 메모리 값을 주소로 하여 또 한 번 이동한다. 두 번 건너뛰게 된다. @뒤의 값이 Label이라면 상대 주소를 이용해서 값을 계산한다.

#(Immediate Addressing mode): n=0, i=1로 설정한다. #뒤의 값이 상수로 들어가게 된다. 이 때도 #뒤에 4096같은 값이 아니라 MAXLEN같은 Label이 들어온다면 상대 주소를 이용해서 값을 계산한다. (b 또는 p가 1, relative로 안 될 경우 마찬가지로 4형식으로 확장 후, e=1로 설정한다)

BUFFER, X(Index mode): Index mode는 Operand에서 뒤에 X가 붙게 된다. 이 경우 x=1로 설정한다.

- **Location Counter(LOCCTR)**

주소 값 할당을 도와주는 변수이다. 소스코드의 START 문장에서 시작주소로 초기화된다. 그 이후 명령어들을 읽으면서 해당하는 Byte를 LOCCTR에 누적해나간다. 현재 LOCCTR을 Label에 할당하면 그것이 Label의 주소가 된다.

- **Object Program에서의 Record**

Object Program에 적힌 Record의 뜻은 다음과 같다.

Header record: H, 프로그램(Section) 이름, 시작 주소, Section 길이

Text record: T, 해당 record의 시작주소, 해당 record의 길이를 적은 이후, 뒤에 Instruction을 object code로 변환한 것을 나열하는데 이 때 나열한 object code가 30byte를 넘어가게 되면 해당 record를 마무리하고 새로 Text record를 시작한다.

End record: E, 처음으로 실행할 명령어의 주소

Define record: D, 외부로 나갈 symbol 이름, 그 symbol의 주소(현재 Section에 기준한 주소)를 계속해서 나열한다.

Refer record: R, 외부에서 쓸 symbol 이름을 나열한다. 주소는 외부에 있어 모르므로 적지 않는다.

Modification record: M, 현재 Section에서 수정할 부분의 주소, 수정할 길이, Modification flag(+,-), 그 주소에 더하거나 혹은 뺄 Symbol 이름(=주소)

2.2. 기술적 배경지식

- **파일 입출력**

java.io.File

파일을 열기 위해 사용되는 기본이다.

boolean exists()를 이용하여 파일이 존재하는지 여부를 묻고, 없다면 boolean createNewFile()를 이용하여 파일을 생성할 수 있다. 또한 append모드인지, 새로 쓰는 모드인지도 설정할 수 있다.

java.io.FileReader

int read(char[] buf)을 이용하여 파일을 읽는 데 사용된다.

java.io.FileWriter

void write(char[] buf)를 이용하여 원하는 문장을 파일에 쓸 수 있다.

모든 File들은 close()를 이용하여 닫아 주어야 한다.

- **Format에 맞추어 출력**

java.lang.String

static String format(String format, Object... args)

String class의 str에 데이터 형식에 맞추어 데이터를 쓸 수 있다. 원하는 format과 인자를 적으면 원하는 format으로 적힌 String을 반환한다. int를 hex string으로 만드는 데 주로 사용된다.

- **Exception**

이번 프로젝트는 에러 처리를 Exception try-catch문을 이용하여 처리하였다.

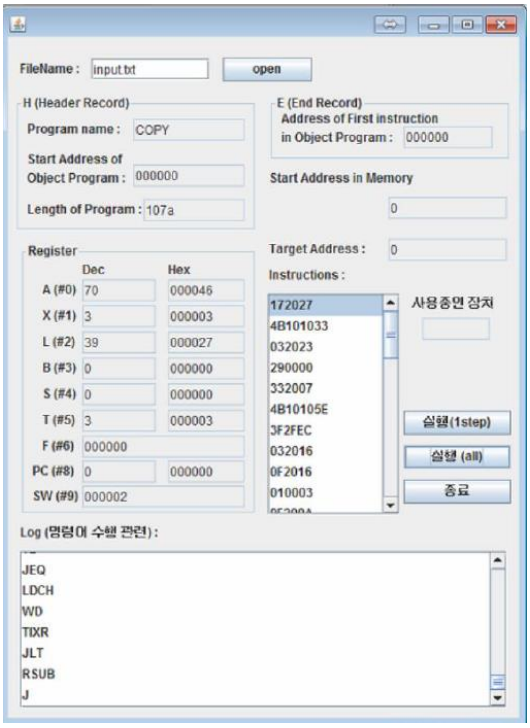
3. 시스템 설계 내용

3.1. 전체 시스템 설계 내용

이 프로젝트는 Assembler를 통해 만들어진 Object Code를 직접 메모리에 올리고, 실행할 수 있는 Loader 및 Simulator를 Java로 구현하는 프로젝트이다.

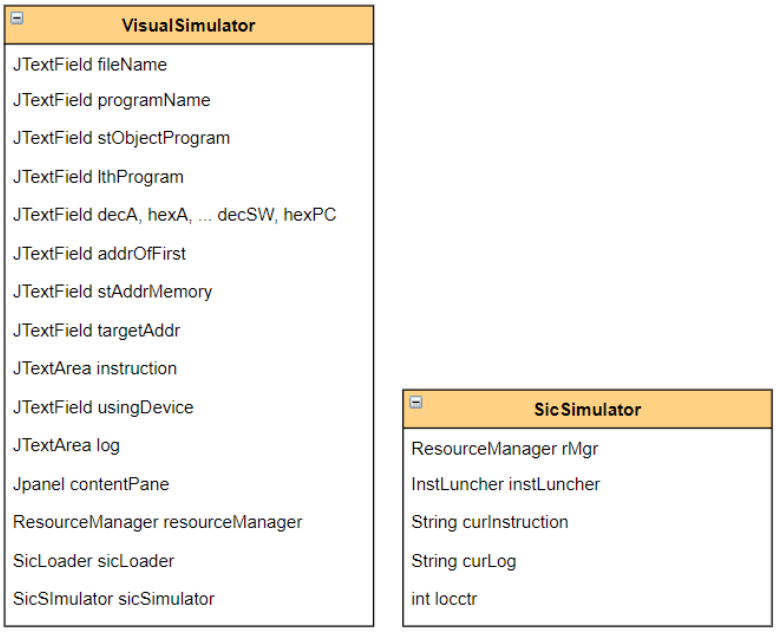
위의 2.1 배경지식에서 읽을 수 있듯이, 프로그램의 전체적인 틀은 다음과 같다. 어셈블러가 Object Code를 만들면 Loader가 그것을 읽어 메모리에 알맞게 올린다. 마지막에 start address로 가서 처음 명령어를 읽고 실행한다. Caller로 돌아가라는 명령이 나올 때까지 (끝날 때까지) 계속 명령을 실행한다.

전체 시스템을 설계하기 위해 필요한 클래스부터 설계하였다.



[그림 3-1] 명세서에 나온 GUI 예시

먼저, 명세서에 나온 GUI 예시를 보고 버튼을 누를 때마다 바뀌줄 필드들을 결정했다.



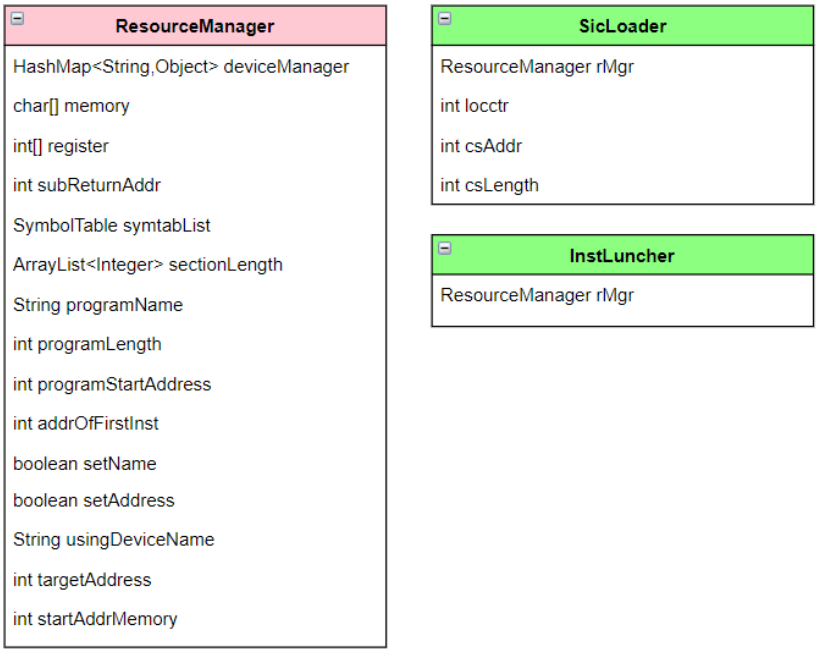
[그림 3-2] 설계한 Class-1

(1) VisualSimulator

시뮬레이터의 동작을 GUI 방식으로 모여주는 모듈이다. 실질적으로 시뮬레이터가 수행하는 작업은 SIC/XE 시뮬레이터 모듈에게 전담시킨다. SIC/XE 시뮬레이터가 동작시킨 이후 리소스 매니저 내의 값들을 읽어 사용자에게 보여준다. 버튼을 누르는 이벤트가 일어나면 이벤트를 SIC/XE Simulator에게 전달한다. 사용자는 위의 GUI 예시에서 볼 수 있듯이 File Open, 1 Step 실행, All 실행, 종료 버튼을 누를 수 있다. 한 스텝씩 진행할 때마다 register들의 값과 target address, 명령어의 start memory address, 사용하고 있는 device 등을 업데이트 해야 하는데, 업데이트 해주어야 할 TextField들을 멤버로 갖고 있다. open한 파일명, 프로그램 이름, 프로그램 전체 길이, 시작할 명령어 주소, 레지스터들의 hex와 dec값, 현재 instruction의 시작 주소와, 타깃 주소, 사용중인 device 이름을 보여주고, 실행하고 있는 instruction object code와 opcode(mnemonic)를 누적해서 보여준다. 자원을 참조하기 위해 ResourceManager를 멤버로 갖고, SicLoader와 SicSimulator를 멤버로 가져 file open시 바로 메모리에 load하고, 이후 실행 버튼을 누를 때마다 이벤트를 SicSimulator에게 전달한다.

(2) SicSimulator

이벤트를 받아 명령어의 실질적인 수행을 담당한다. ResourceManager를 컨트롤하기 위해 멤버로 갖고있다. 또한 명령어를 실제로 시행하는 함수들을 갖는 InstLuncher를 멤버로 가진다. 명령어의 실직적인 수행을 담당하기 때문에 현재 locctr을 갖고, 현재 시행하는 명령어의 Instruction object code와 그 영어 이름을 갖는 curInstruction과 curLog를 갖는다.



[그림 3-3] 설계한 Class-2

(3) ResourceManager

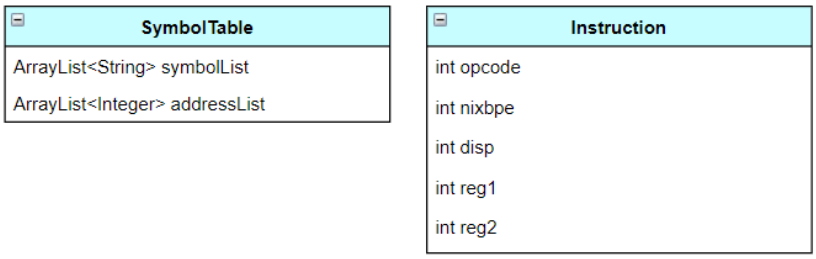
SIC/XE 머신은 실제 물리적인 하드웨어가 아니기 때문에 시뮬레이터를 구동시키기 위해서는 가상의 하드웨어 장치를 가정하여야 한다. 가상의 Device들을 이름과 매핑시켜 넣는 Device Manager를 멤버로 갖는다. 또한 object code를 올리기 위한 메모리 영역과 레지스터 영역을 담당하는 char[] memory (여기서는 64KB 사용)와 int[] register를 변수로 갖는다. section별로 Label들을 갖고 있는 SymbolTable과, section별 길이를 갖는 sectionLength, program 정보를 저장하기 위해 program name program length, program start address, First Instruction을 멤버로 갖고, name과 start address를 세팅 했는지 여부를 묻는 setName setAddress 또한 멤버로 갖는다. 한 프로그램에 H 레코드가 여러 개 있을 수 있는데, 이런 플래그가 없으면 무엇이 실제 프로그램 이름인지 알 수 없기 때문이다. 또한 현재 사용 중인 usingDeviceName과 현재 명령어의 target address, start memory address를 갖는다.

(4) SicLoader

시뮬레이터에서 가장 먼저 수행하는 모듈로써 어셈블러로 만들어진 목적 코드를 읽어 ResourceManager에 변수로 지정되어 있는 가상의 메모리 영역에 로드해주는 역할을 수행한다. 실제 메모리에 올리기 위해 ResourceManager를 멤버로 갖고, 올리는 데에 필요한 locctr와 현재 섹션 시작 주소인 csAddr, 현재 섹션의 길이인 csLength를 멤버로 갖는다.

(5) InstLuncher

SIC/XE machine이 명령어에 따라 수행할 동작을 정의한 모듈이다. SIC/XE Simulator에서 Instruction Luncher를 통해 목적 코드를 수행한다. 자원들 값을 실제로 바꿔야 하기 때문에 ResourceManager를 멤버로 갖는다.



[그림 3-4] 설계한 Class-3

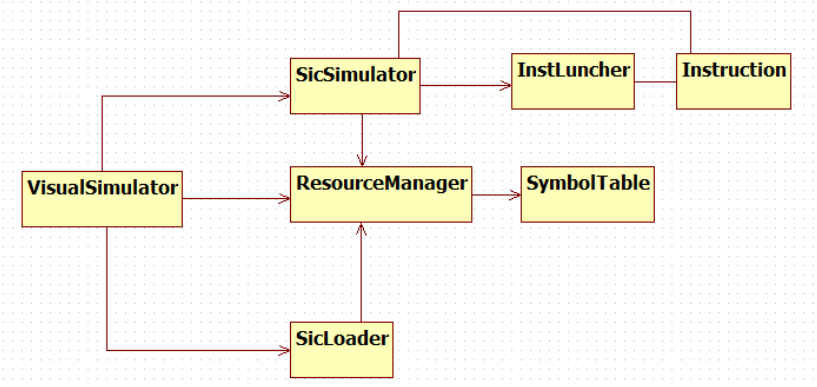
(6) SymbolTable

section별로 존재하는 label들을 저장하는 Table이다. symbol과 주소를 ArrayList로 갖는다.

(7) Instruction

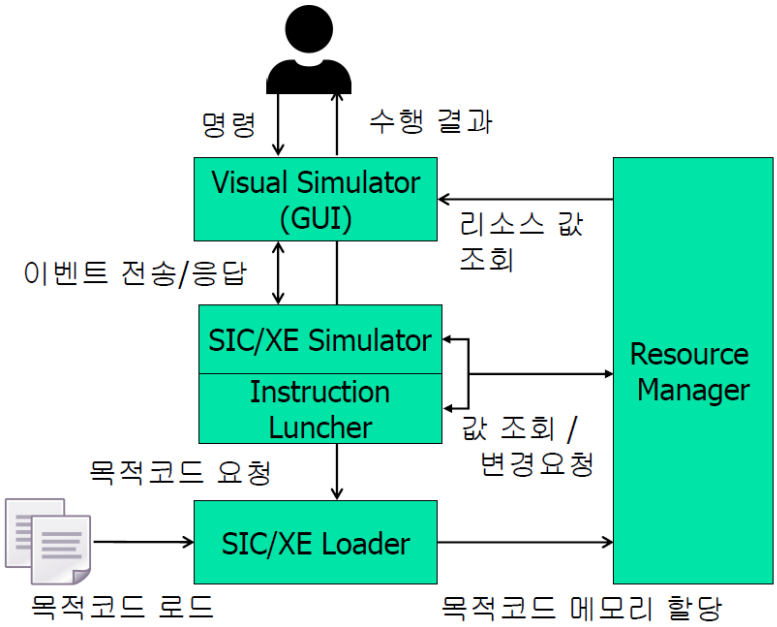
Object code의 명령어 정보를 간편하게 저장하기 위한 Class이다. opcode와 nixbpe를 갖고 있으며, 3,4형식일 경우 disp를, 2형식일 경우 reg1, reg2를 갖게 된다.

지금까지 설계한 Class들을 바탕으로 관계도를 그려보면 다음과 같다.



[그림 3-5] 설계한 Class들의 관계도

VisualSimulator는 SicSimulator, ResourceManager, SicLoader를 멤버로 갖고, SicSimulator는 ResourceManager와 InstLuncher를 멤버로 가진다. Instruction은 직접 멤버로 갖고 있지는 않지만 argument로 생성하여 주는 데 사용된다. SicLoader 또한 ResourceManager를 멤버로 가진다. ResourceManager는 SymbolTable을 멤버로 가진다.



[그림 3-6] 전체적인 시스템 흐름도

다음은 간단히 나타낸 전체적인 시스템 설계 흐름도이다. 해당 흐름도에 따라 필요한 모듈의 기능들을 아래에 작성해 보았다.

3.2. 모듈별 설계 내용

(1) VisualSimulator

(1-1) open 버튼을 눌러서 파일을 선택했을 때, Object code를 load하기위한 모듈

(1-2) one step 버튼을 눌렀을 때 모듈

(1-3) all step 버튼을 눌렀을 때 모듈

(1-4) step을 끝내고 GUI에 상태를 업데이트하기 위한 모듈

(1-5) load를 다 하고 GUI에 상태를 업데이트하기 위한 모듈

(1-6) 멈춤 버튼을 눌렀을 때 모듈

(2) SIC/XE Simulator

(2-1) one step을 눌렀을 때 실제로 동작을 수행하는 모듈

(2-2) all step을 눌렀을 때 실제로 동작을 수행하는 모듈

(2-3) log(GUI)를 추가하는 모듈

(2-4) instruction(GUI)을 추가하는 모듈

(2-5) 메모리에서 ObjectCode를 읽어 Instruction으로 만드는 모듈

(3) ResourceManager

(3-1) 메모리와 레지스터를 깨끗하게 초기화하는 모듈

(3-2) 열린 Device들을 다 닫는 모듈

(3-3) Device가 사용 가능한지 Test하는 모듈

(3-4) Device에서 data를 읽어오는 모듈

(3-5) Device에 data를 쓰는 모듈

(3-6) 메모리에서 정해진 길이만큼 데이터를 읽어오는 모듈

(3-7) 메모리에 데이터를 setting하는 모듈

(3-8) 레지스터 값을 가져오는 모듈

(3-9) 레지스터에 값을 세팅하는 모듈

(3-10) 메모리에 쓰기 위해 int를 char배열로 만들어주는 모듈

(3-11) 메모리에서 읽은 char배열을 int로 만들어주는 모듈

(4) SIC/XE Loader

(4-1) 실제로 Object Code를 메모리에 load하는 작업을 하는 모듈

(4-2) H record를 처리하는 모듈

(4-3) T record를 처리하는 모듈

(4-4) M record를 처리하는 모듈

(5) Instruction Luncher

COPY에 있는 모든 명령어를 처리하는 모듈을 만든다

(6) SymbolTable

(6-1) Symbol과 주소를 넣는 모듈

(6-2) 주소를 수정하는 모듈

(6-3) Symbol로 주소를 찾는 모듈

(7) Instruction

(7-1) Instruction을 만드는 모듈

이러한 설계 내용들을 기반으로 처음 설계해본 함수는 다음과 같다.

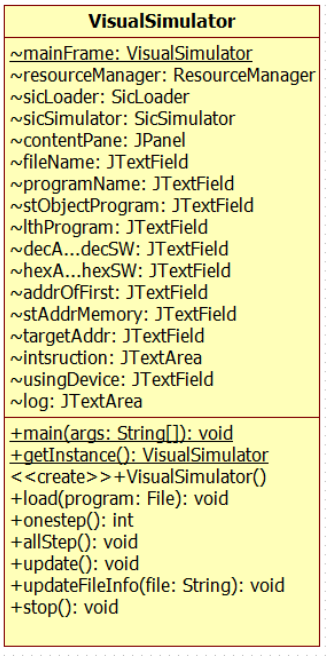
함수 기능	설계한 함수
(1) VisualSimulator	
(1-1) open 버튼을 눌러서 파일을 선택했을 때, Object code를 load하기위한 모듈	void load(File program)
(1-2) one step 버튼을 눌렀을 때 모듈	int oneStep()
(1-3) all step 버튼을 눌렀을 때 모듈	void allStep()
(1-4) step을 끝내고 GUI에 상태를 업데이트하기 위한 모듈	void update()
(1-5) load를 다 하고 GUI에 상태를 업데이트하기 위한 모듈	void updateFileInfo(String file)
(1-6) 멈춤 버튼을 눌렀을 때 모듈	void stop()
(2) SIC/XE Simulator	
(2-1) one step을 눌렀을 때 실제로 동작을 수행하는 모듈	int oneStep()
(2-2) all step을 눌렀을 때 실제로 동작을 수행하는 모듈	void allStep()
(2-3) log(GUI)를 추가하는 모듈	void addLog(String log)
(2-4) instruction(GUI)을 추가하는 모듈	void setCurInstruction(char[] arr)
(2-5) 메모리에서 ObjectCode를 읽어 Instruction으로 만드는 모듈	int[] parsing(char[] arr)
(3) ResourceManager	
(3-1) 메모리와 레지스터를 깨끗하게 초기화하는 모듈	void initializeResource()
(3-2) 열린 Device들을 다 닫는 모듈	void closeDevice()
(3-3) Device가 사용 가능한지 Test하는 모듈	void testDevice(String devName)
(3-4) Device에서 data를 읽어오는 모듈	char[] readDevice(String devName, int num)
(3-5) Device에 data를 쓰는 모듈	void writeDevice(String devName, char[] data, int num)
(3-6) 메모리에서 정해진 길이만큼 데이터를 읽어오는 모듈	char[] getMemory(int location, int num)
(3-7) 메모리에 데이터를 setting하는 모듈	void setMemory(int locate, int num)
(3-8) 레지스터 값을 가져오는 모듈	int getRegister(int regNum)
(3-9) 레지스터에 값을 세팅하는 모듈	void setRegister(int regNum, int value)
(3-10) 메모리에 쓰기 위해 int를 char배열로 만들어주는 모듈	char[] intToChar(int data)
(3-11) 메모리에서 읽은 char배열을 int로 만들어주는 모듈	int byteToInt(char[] data)

(4) SIC/XE Loader	
(4-1) 실제로 Object Code를 메모리에 load하는 작업을 하는 모듈	void load(File objectCode)
(4-2) H record를 처리하는 모듈	void headRecord(String line)
(4-3) T record를 처리하는 모듈	void textRecord(String line)
(4-4) M record를 처리하는 모듈	void modifyRecord(String line)
(5) Instruction Luncher	COPY에 있는 모든 명령어를 처리하는 모듈을 만든다
(6) SymbolTable	
(6-1) Symbol과 주소를 넣는 모듈	void putSymbol(String symbol, int address)
(6-2) 주소를 수정하는 모듈	void modifySymbol(String symbol, int newaddress)
(6-3) Symbol로 주소를 찾는 모듈	int search(String symbol)
(7) Instruction	
(7-1) Instruction을 만드는 모듈	생성자로 가능하다

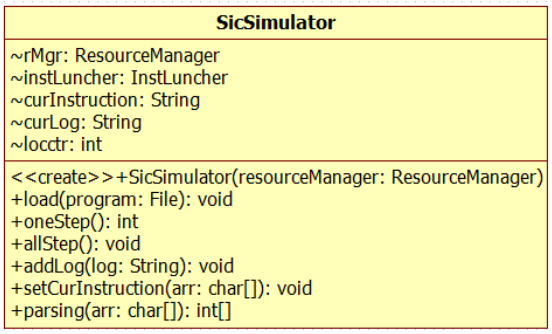
4. 시스템 구현 내용(구현 화면 포함)

4.1. 전체 시스템 구현 내용

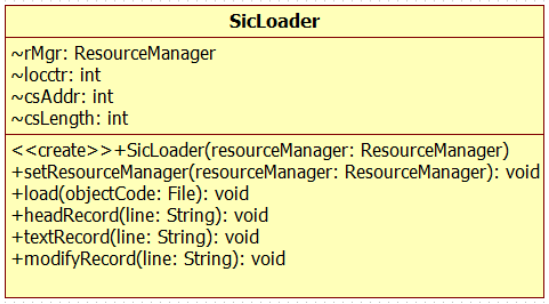
다음은 클래스 다이어그램이다. 클래스는 설계한대로 만들어졌다. 아래의 다이어그램은 StarUML을 사용하였다.



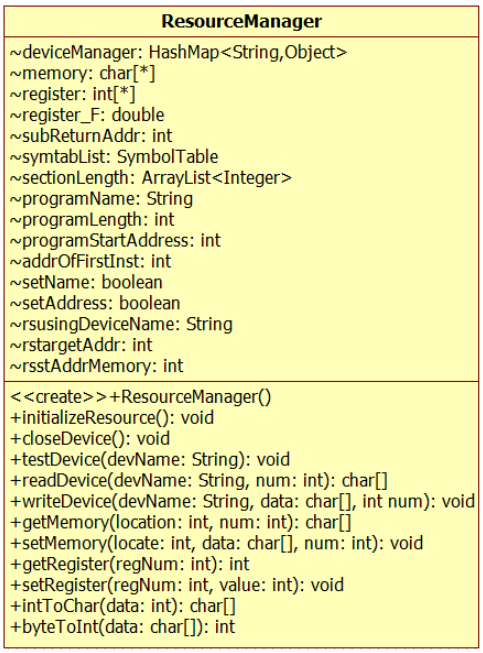
[그림 4-1] VisualSimulator.java



[그림 4-2] SicSimulator.java

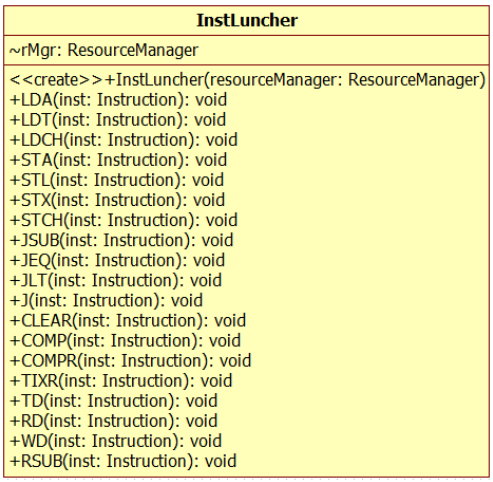


[그림 4-3] SicLoader.java

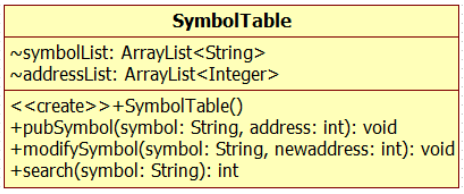


[그림 4-4] ResourceManager.java

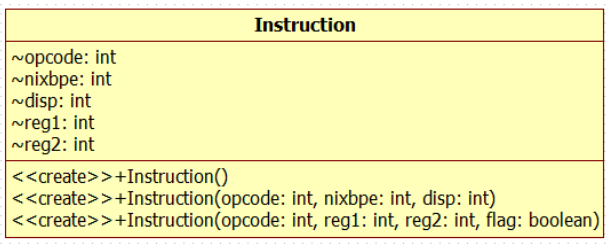
여기서 subReturnAddr라는 것이 추가되었는데, 원래 sub routine을 실행하면 돌아갈 주소를 L register에 저장한다. 그럼 원래 L register에 있던 값은 저장할 곳이 없다. 그래서 Stack을 사용해 돌아갈 주소를 넣어둔다. 원칙적으로는 그렇지만, 여기서는 Stack을 쓸 만큼 깊게 sub routine들을 여러 번 호출하지 않아서 int 변수로 두었다.



[그림 4-5] InstLuncher.java



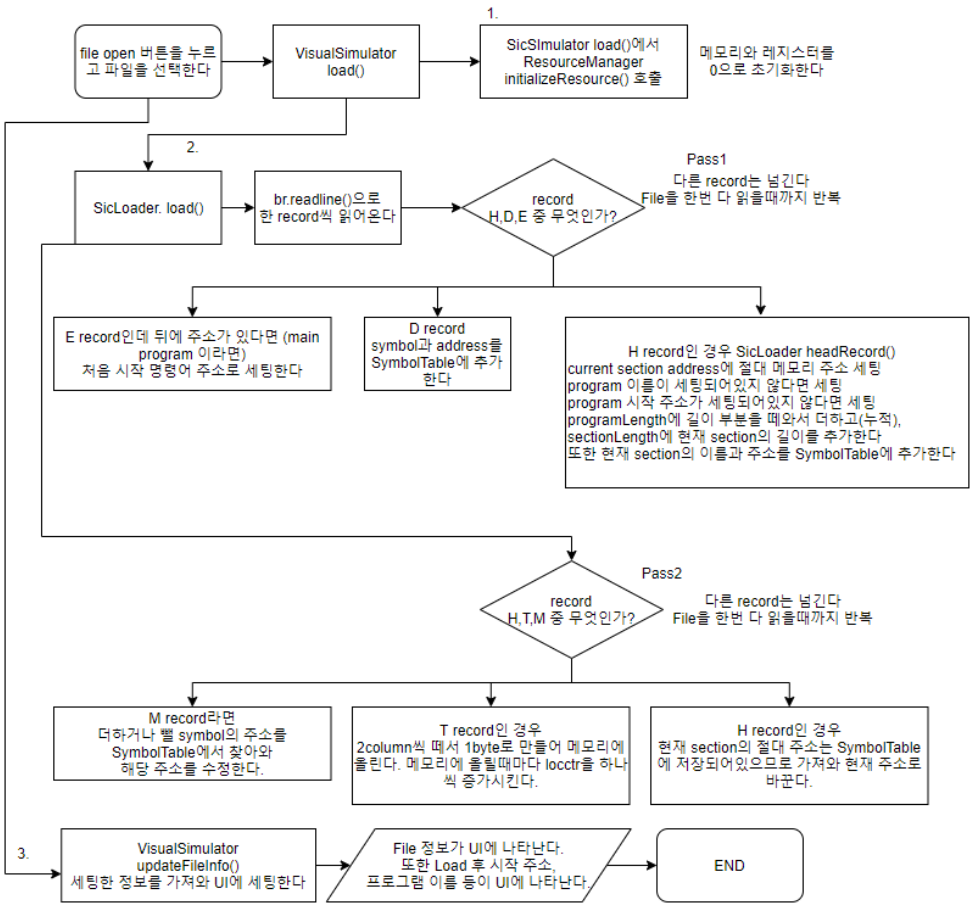
[그림 4-6] SymbolTable.java



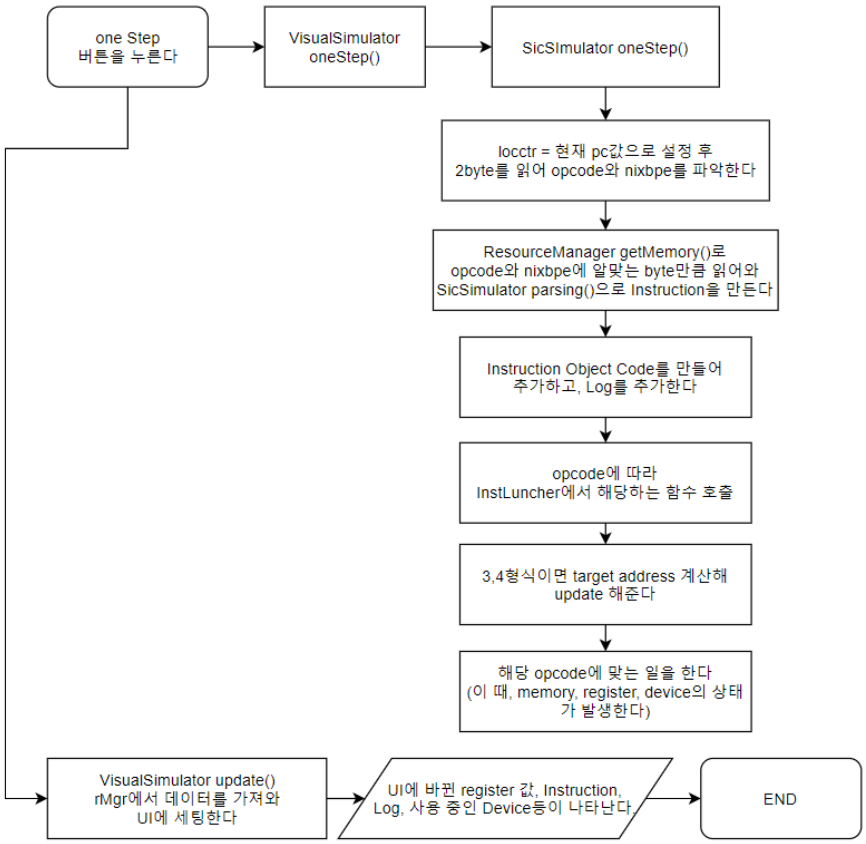
[그림 4-7] Instruction.java

원래 객체지향 설계를 위해서는 멤버변수들이 private인 것이 맞지만, 여러 번 접근하는 변수들이 많다 보니 모두 default(package) 범위로 두게 되었다. 사실 이 부분은 보안을 위해서 수정하는 것이 맞다고 생각한다.

아래는 실제 메모리에 object code를 load하는 동작과, one step 버튼을 눌렀을 때 명령어를 수행하는 동작의 아주 간단한 flow chart이다. (호출 관계 중심)

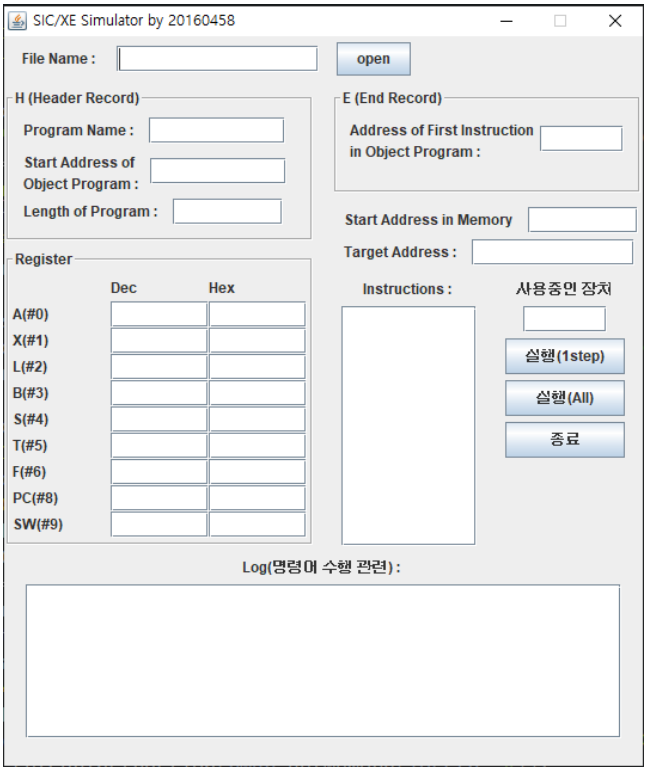


[그림 4-8] file open시 flow chart

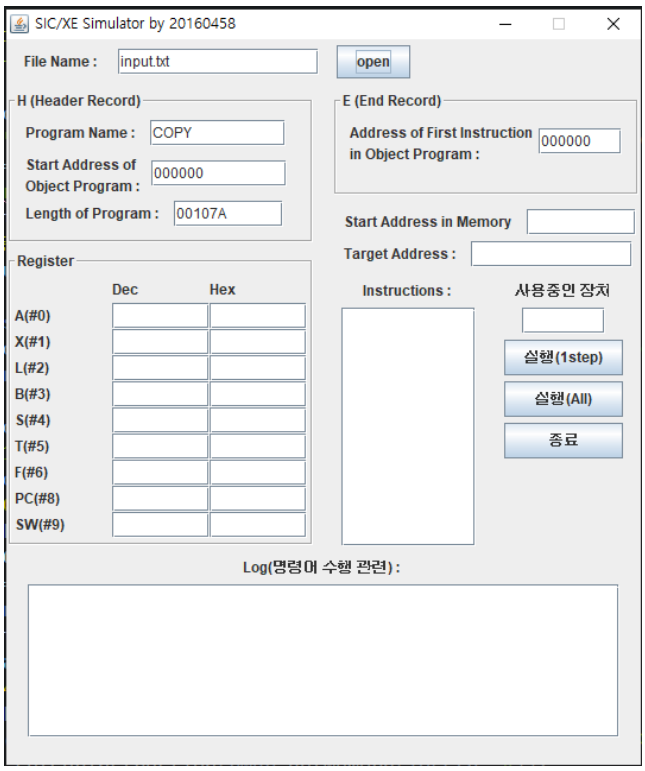


[그림 4-9] one step 버튼 눌렀을 시 flow chart

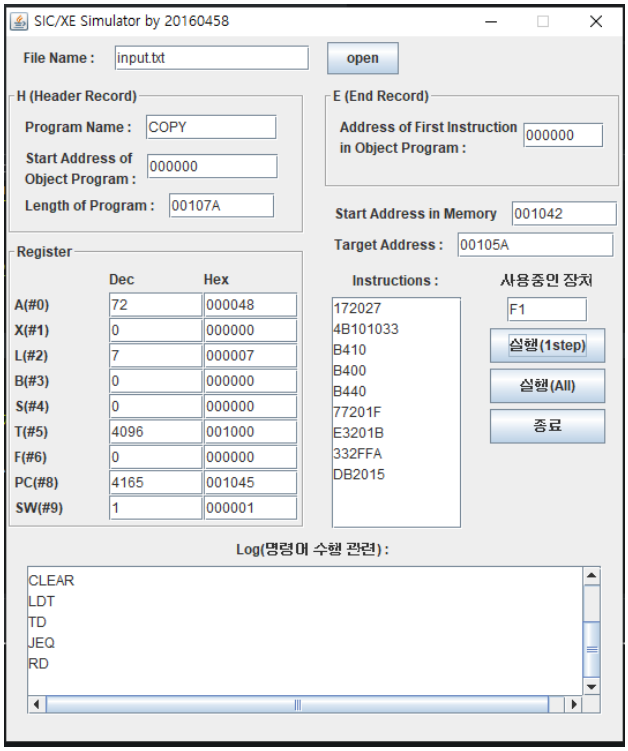
프로그램 수행 결과



[그림 4-10] 실행했을 때 GUI 초기화면

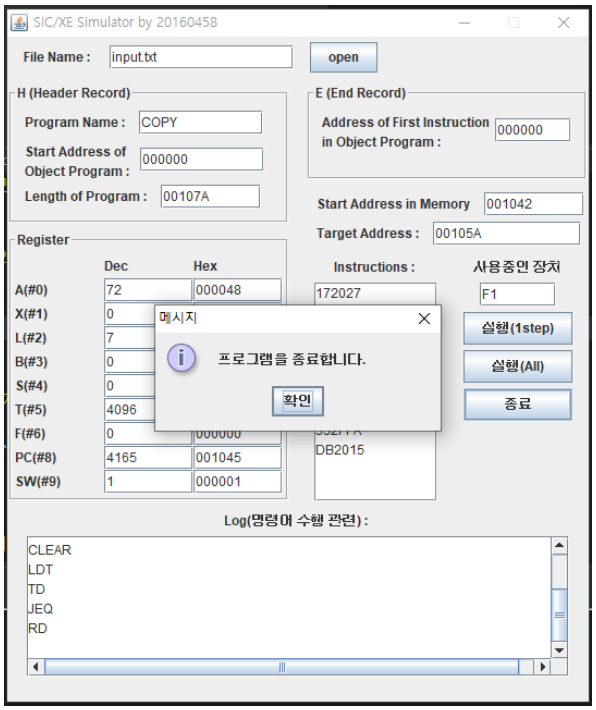


[그림 4-11] 파일을 열었을 때 Load된 후 화면

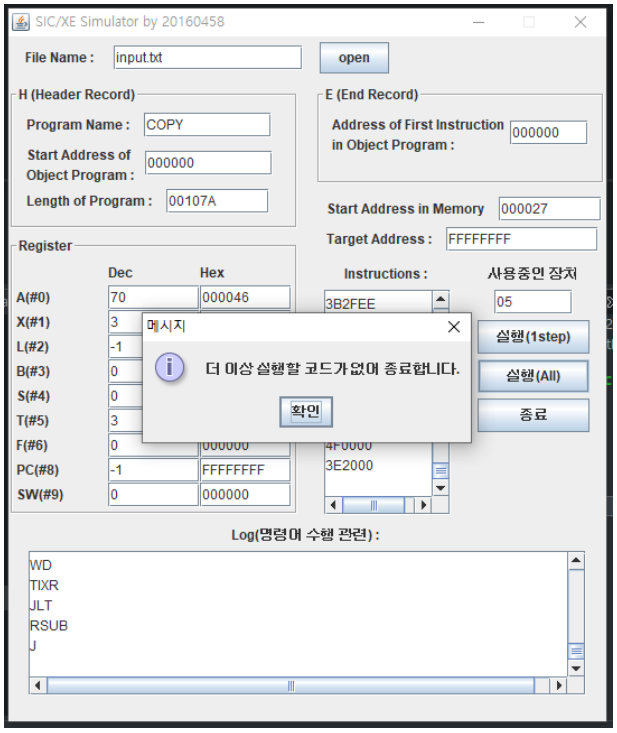


[그림 4-12] 1step씩 실행했을 때 화면

레지스터 값이 바뀌고, Instructions와 Log에 값들이 하나씩 누적되는 것을 볼 수 있다.

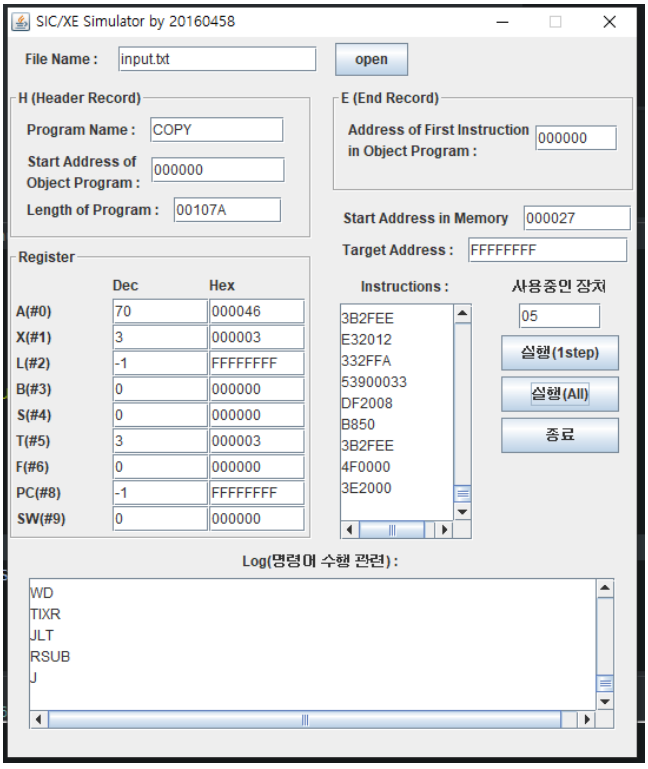


[그림 4-13] 종료를 눌렀을 때의 화면

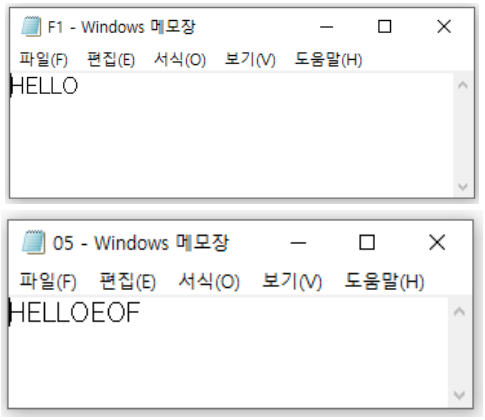


[그림 4-14] All 실행 버튼을 누르거나, 더 이상 실행할 명령어가 없는데 1 step 버튼을 눌렀을 때의 화면

더 이상 실행할 명령이 없으므로 종료한다. 여기서 종료 버튼을 누르면 UI 화면이 꺼지지는 않는다. UI화면을 끄고 싶다면 우측 상단에 x버튼을 눌러야 한다. PC 값이 -1이면 모든 코드를 실행하고 Caller(@RETADR)로 돌아왔단 뜻이다.



[그림 4-15] 모든 명령어가 실행되고 난 후의 화면



[그림 4-16] 모든 명령어가 실행되고 난 후 Device에 적힌 값들

4.2. 모듈별 구현 내용

클래스 구현 내용이다. 주석으로 동작 과정들을 모두 설명하였다.

(1) VisualSimulator

```
package SP20_simulator;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.io.File;
import java.io.IOException;

/**
 * VisualSimulator는 사용자와의 상호작용을 담당한다.<br>
 * 즉, 버튼 클릭등의 이벤트를 전달하고 그에 따른 결과값을 화면에 업데이트 하는 역할을 수행한다.<br>
 * 실제적인 작업은 SicSimulator에서 수행하도록 구현한다.
 */
public class VisualSimulator extends JFrame{

	ResourceManager resourceManager = new ResourceManager();
	SicLoader sicLoader = new SicLoader(resourceManager);
	SicSimulator sicSimulator = new SicSimulator(resourceManager);

	private static VisualSimulator mainFrame;

	static{
		mainFrame=new VisualSimulator();
	}

	private JPanel contentPane;

	/*step마다 update될 textfield*/
	JTextField fileName;
	JTextField programName;
	JTextField stObjectProgram;
	JTextField lthProgram;
	JTextField decA;
	JTextField hexA;
	JTextField decX;
	JTextField hexX;
	JTextField decL;
	JTextField hexL;
	JTextField decB;
	JTextField hexB;
	JTextField decS;
	JTextField hexS;
	JTextField decT;
	JTextField hexT;
	JTextField decF;
	JTextField hexF;
	JTextField decPC;
	JTextField hexPC;
	JTextField decSW;
	JTextField hexSW;
	JTextField addrOfFirst;
	JTextField stAddrMemory;
	JTextField targetAddr;
```

```
	JTextArea instruction;
	JTextField usingDevice;
	JTextArea log;


	/**
	 * 프로그램 로드 명령을 전달한다.
	 */
	public void load(File program) throws IOException{
		//...
		sicSimulator.load(program); //여기서 메모리 initialize
		sicLoader.load(program);
	};

	/**
	 * 하나의 명령어만 수행할 것을 SicSimulator에 요청한다.
	 */
	public int oneStep(){
		try{
			if(sicSimulator.oneStep()>0){
				update();
				return 1;
			}else{
				//program이 끝난 경우
				resourceManager.closeDevice();
				JOptionPane.showMessageDialog(contentPane, "더 이상 실행할
코드가 없어 종료합니다.");
				return -1;
			}
		}catch(Exception e){
			e.printStackTrace();
		}

		return -1;
	};

	/**
	 * 남아있는 모든 명령어를 수행할 것을 SicSimulator에 요청한다.
	 */
	public void allStep(){
		try{
		while(oneStep()>0)
			;
		}catch(Exception e){
			e.printStackTrace();
		}
	};

	/**
	 * 화면을 최신값으로 갱신하는 역할을 수행한다.
	 */
	public void update(){

		decA.setText(String.format("%d", resourceManager.register[0]));
		hexA.setText(String.format("%06X", resourceManager.register[0]));
		decX.setText(String.format("%d", resourceManager.register[1]));
		hexX.setText(String.format("%06X", resourceManager.register[1]));
		decL.setText(String.format("%d", resourceManager.register[2]));
		hexL.setText(String.format("%06X", resourceManager.register[2]));
		decB.setText(String.format("%d", resourceManager.register[3]));
		hexB.setText(String.format("%06X", resourceManager.register[3]));
```

```
                decS.setText(String.format("%d", resourceManager.register[4]));
                hexS.setText(String.format("%06X", resourceManager.register[4]));
                decT.setText(String.format("%d", resourceManager.register[5]));
                hexT.setText(String.format("%06X", resourceManager.register[5]));
                decF.setText(String.format("%d", resourceManager.register[6]));
                hexF.setText(String.format("%06X", resourceManager.register[6]));
                decPC.setText(String.format("%d", resourceManager.register[8]));
                hexPC.setText(String.format("%06X", resourceManager.register[8]));
                decSW.setText(String.format("%d", resourceManager.register[9]));
                hexSW.setText(String.format("%06X", resourceManager.register[9]));

                //target address가 바뀌는 경우에만 Target Addr 값이 바뀐다(immediate일 경우,
TA가 없으므로 바뀌지 않음)
                //Device도 마찬가지로 바뀌는 경우에만 바뀐다. 그렇지 않을 경우 유지된다.
                stAddrMemory.setText(String.format("%06X",
resourceManager.rsstAddrMemory));
                targetAddr.setText(String.format("%06X", resourceManager.rstargetAddr));
                usingDevice.setText(resourceManager.rsusingDeviceName);

                //얘네는 하나씩 항목을 추가해야함
                instruction.append(sicSimulator.curInstruction+"\n"); // instruction 내용을
붙이고
                instruction.setCaretPosition(instruction.getDocument().getLength()); //맨
아래로 스크롤한다.
                log.append(sicSimulator.curLog+"\n");
                log.setCaretPosition(log.getDocument().getLength());

        };

        /*load 후 file 정보와 program 정보를 update하는 함수*/
        public void updateFileInfo(String file){
                fileName.setText(file);
                programName.setText(resourceManager.programName);
                String stobpr = String.format("%06X", resourceManager.programStartAddress);
                stObjectProgram.setText(stobpr);
                String lthp = String.format("%06X", resourceManager.programLength);
                lthProgram.setText(lthp);
                String adf = String.format("%06X", resourceManager.addrOfFirstInst);
                addrOfFirst.setText(adf);
        }

        public void stop(){
                try{
                resourceManager.closeDevice();
                JOptionPane.showMessageDialog(contentPane, "프로그램을 종료합니다.");
                }catch(Exception e){
                        e.printStackTrace();
                }
        }

        public static VisualSimulator getInstance(){
                return mainFrame;
        }

        public static void main(String[] args) {

                EventQueue.invokeLater(new Runnable(){
                        public void run(){
                                try{
                                        VisualSimulator frame = mainFrame;
```

```
                                        frame.setVisible(true);}
                                catch(Exception e){
                                        e.printStackTrace();
                                }
                        }
                });
        }

        public VisualSimulator(){ //생성자, GUI 화면 구성
                super("SIC/XE Simulator by 20160458");
                setResizable(false);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //윈도우 종료시 강제 종료
                setBounds(200,200,540,640); //위치와 크기 설정
                contentPane=new JPanel(new BorderLayout());

                JPanel filePanel = new JPanel(new FlowLayout(FlowLayout.LEFT,15,5));
                JLabel lblFileName = new JLabel("File Name : ");
                fileName = new JTextField(15);
                JButton btnFileOpen=new JButton("open");
                filePanel.add(lblFileName);
                filePanel.add(fileName);
                filePanel.add(btnFileOpen);

                JPanel leftPanel = new JPanel(new FlowLayout());
                leftPanel.setPreferredSize(new Dimension(260,400));
                JPanel headerPanel = new JPanel(new FlowLayout(FlowLayout.LEFT,10,5));
                headerPanel.setPreferredSize(new Dimension(260,130));
                headerPanel.setBorder(new TitledBorder(new EtchedBorder(), "H (Header
Record)"));
                JLabel lblProgramName = new JLabel("Program Name : ");
                programName = new JTextField(10);
                JLabel lblStObjectProgram = new JLabel("<html>Start Address of <br>Object
Program : <html>");
                stObjectProgram = new JTextField(10);
                JLabel lblLthProgram = new JLabel("Length of Program : ");
                lthProgram = new JTextField(8);

                headerPanel.add(lblProgramName);
                headerPanel.add(programName);
                headerPanel.add(lblStObjectProgram);
                headerPanel.add(stObjectProgram);
                headerPanel.add(lblLthProgram);
                headerPanel.add(lthProgram);


                JPanel registerPanel = new JPanel(new GridLayout(10,3));
                registerPanel.setPreferredSize(new Dimension(260,250));
                registerPanel.setBorder(new TitledBorder(new EtchedBorder(), "Register"));
                JLabel lblDec = new JLabel("Dec");
                JLabel lblHex = new JLabel("Hex");
                JLabel lblA = new JLabel("A(#0) ");
                decA = new JTextField(6);
                hexA = new JTextField(6);
                JLabel lblX = new JLabel("X(#1) ");
                decX = new JTextField(6);
                hexX = new JTextField(6);
                JLabel lblL = new JLabel("L(#2)");
                decL = new JTextField(6);
                hexL = new JTextField(6);
                JLabel lblB = new JLabel("B(#3)");
                decB = new JTextField(6);
                hexB = new JTextField(6);
                JLabel lblS = new JLabel("S(#4)");
```

```
                decS = new JTextField(6);
                hexS = new JTextField(6);
                JLabel lblT = new JLabel("T(#5)");
                decT = new JTextField(6);
                hexT = new JTextField(6);
                JLabel lblF = new JLabel("F(#6)");
                decF = new JTextField(6);
                hexF = new JTextField(6);
                JLabel lblPC = new JLabel("PC(#8)");
                decPC = new JTextField(6);
                hexPC = new JTextField(6);
                JLabel lblSW = new JLabel("SW(#9)");
                decSW = new JTextField(6);
                hexSW = new JTextField(6);
                registerPanel.add(Box.createHorizontalGlue());
                registerPanel.add(lblDec);
                registerPanel.add(lblHex);
                registerPanel.add(lblA);
                registerPanel.add(decA);
                registerPanel.add(hexA);
                registerPanel.add(lblX);
                registerPanel.add(decX);
                registerPanel.add(hexX);
                registerPanel.add(lblL);
                registerPanel.add(decL);
                registerPanel.add(hexL);
                registerPanel.add(lblB);
                registerPanel.add(decB);
                registerPanel.add(hexB);
                registerPanel.add(lblS);
                registerPanel.add(decS);
                registerPanel.add(hexS);
                registerPanel.add(lblT);
                registerPanel.add(decT);
                registerPanel.add(hexT);
                registerPanel.add(lblF);
                registerPanel.add(decF);
                registerPanel.add(hexF);
                registerPanel.add(lblPC);
                registerPanel.add(decPC);
                registerPanel.add(hexPC);
                registerPanel.add(lblSW);
                registerPanel.add(decSW);
                registerPanel.add(hexSW);

                leftPanel.add(headerPanel);
                leftPanel.add(registerPanel);

                JPanel rightPanel = new JPanel(new FlowLayout());
                rightPanel.setPreferredSize(new Dimension(260,400));
                JPanel endPanel = new JPanel(new FlowLayout());
                endPanel.setPreferredSize(new Dimension(260,90));
                endPanel.setBorder(new TitledBorder(new EtchedBorder(), "E (End Record)"));
                JLabel lblAddrOfFirst = new JLabel("<html>Address of First Instruction<br>
in Object Program : <html>");
                addrOfFirst = new JTextField(6);
                endPanel.add(lblAddrOfFirst);
                endPanel.add(addrOfFirst);

                JPanel instructionPanel = new JPanel(new BorderLayout());
                instructionPanel.setPreferredSize(new Dimension(260,300));

                JPanel instNorthPanel = new JPanel(new FlowLayout(FlowLayout.LEFT,10,5));
                instNorthPanel.setPreferredSize(new Dimension(260,60));
```

```
        JLabel lblStAddrMemory = new JLabel("Start Address in Memory ");
        stAddrMemory = new JTextField(8);
        JLabel lblTargetAddr = new JLabel("Target Address : ");
        targetAddr = new JTextField(12);
        instNorthPanel.add(lblStAddrMemory);
        instNorthPanel.add(stAddrMemory);
        instNorthPanel.add(lblTargetAddr);
        instNorthPanel.add(targetAddr);

        JPanel instWestPanel = new JPanel(new FlowLayout());
        instWestPanel.setPreferredSize(new Dimension(130,250));
        JLabel lblInstructions = new JLabel("Instructions : ");
        instruction = new JTextArea(11,10);
        JScrollPane instscroll = new JScrollPane(instruction);
        instWestPanel.add(lblInstructions);
        instWestPanel.add(instscroll);

        JPanel instEastPanel = new JPanel(new FlowLayout());
        instEastPanel.setPreferredSize(new Dimension(130,250));
        JLabel lblUsingDevice = new JLabel("사용중인 장치");
        usingDevice = new JTextField(6);
        JButton btnOneStep=new JButton("실행(1step)");
        btnOneStep.setPreferredSize(new Dimension(100,30));
        JButton btnAllStep=new JButton("실행(All)");
        btnAllStep.setPreferredSize(new Dimension(100,30));
        JButton btnStop=new JButton("종료");
        btnStop.setPreferredSize(new Dimension(100,30));
        instEastPanel.add(lblUsingDevice);
        instEastPanel.add(usingDevice);
        instEastPanel.add(btnOneStep);
        instEastPanel.add(btnAllStep);
        instEastPanel.add(btnStop);


        instructionPanel.add(instNorthPanel,BorderLayout.NORTH);
        instructionPanel.add(instWestPanel,BorderLayout.WEST);
        instructionPanel.add(instEastPanel,BorderLayout.EAST);

        rightPanel.add(endPanel);
        rightPanel.add(instructionPanel);

        JPanel logPanel = new JPanel(new FlowLayout());
        logPanel.setPreferredSize(new Dimension(520,180));
        JLabel lblLog = new JLabel("Log(명령어 수행 관련) : ");
        log = new JTextArea(7,45);
        JScrollPane logscroll = new JScrollPane(log);
        logPanel.add(lblLog);
        logPanel.add(logscroll);
        contentPane.add(filePanel,BorderLayout.NORTH);
        contentPane.add(leftPanel,BorderLayout.WEST);
        contentPane.add(rightPanel,BorderLayout.EAST);
        contentPane.add(logPanel,BorderLayout.SOUTH);

        btnFileOpen.addActionListener(new ActionListener(){
                public void actionPerformed(ActionEvent event){
                        JFileChooser chooser = new JFileChooser();
                        int ret = chooser.showOpenDialog(null);
                        if(ret==JFileChooser.APPROVE_OPTION){
                                try{
                                        load(chooser.getSelectedFile());

updateFileInfo(chooser.getSelectedFile().getName());
```

```
                                        }catch(Exception e){
                                                e.printStackTrace();
                                        }
                                }
                        }
                });

                btnOneStep.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent event){
                                oneStep();
                        }
                });

                btnAllStep.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent event){
                                allStep();
                        }
                });

                /*종료 버튼은 지금 실행하는 program을 종료하는거지 이 UI를 끄는 것이 아님.
                 * UI를 끄려면 X 버튼을 눌러 나가야함.*/
                btnStop.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent event){
                                stop();
                        }
                });

                setContentPane(contentPane);
        }
}
```

(2) SicSimulator

```
package SP20_simulator;

import java.io.File;
import java.io.IOException;

/**
 * 시뮬레이터로서의 작업을 담당한다. VisualSimulator에서 사용자의 요청을 받으면 이에 따라
 * ResourceManager에 접근하여 작업을 수행한다.
 *
 * 작성중의 유의사항 :
 *  1) 새로운 클래스, 새로운 변수, 새로운 함수 선언은 얼마든지 허용됨. 단, 기존의 변수와 함수들을
삭제하거나 완전히 대체하는 것은 지양할 것.
 *  2) 필요에 따라 예외처리, 인터페이스 또는 상속 사용 또한 허용됨.
 *  3) 모든 void 타입의 리턴값은 유저의 필요에 따라 다른 리턴 타입으로 변경 가능.
 *  4) 파일, 또는 콘솔창에 한글을 출력시키지 말 것. (채점상의 이유. 주석에 포함된 한글은 상관 없음)
 *
 *  + 제공하는 프로그램 구조의 개선방법을 제안하고 싶은 분들은 보고서의 결론 뒷부분에 첨부
바랍니다. 내용에 따라 가산점이 있을 수 있습니다.
 */
public class SicSimulator {
        ResourceManager rMgr;
        InstLuncher instLuncher;
        String curInstruction;
        String curLog;
        int locctr;

        public SicSimulator(ResourceManager resourceManager) {
                // 필요하다면 초기화 과정 추가
                this.rMgr = resourceManager;
                this.instLuncher=new InstLuncher(resourceManager);
                locctr=0; //start address로 초기화해야함
        }

        /**
         * 레지스터, 메모리 초기화 등 프로그램 load와 관련된 작업 수행.
         * 단, object code의 메모리 적재 및 해석은 SicLoader에서 수행하도록 한다.
         */
        public void load(File program) {
                /* 메모리 초기화, 레지스터 초기화 등*/
                //resourceManager 생성할때 initialize함수를 통해 쓰레기값 없도록 초기화한다.
                rMgr.initializeResource();
        }

        /**
         * 1개의 instruction이 수행된 모습을 보인다.
         * @return 프로그램 끝을 알릴시 -1, 아니면 1을 리턴한다
         */
        public int oneStep() throws IOException {
                //어차피 명령어대로 실행하는 거기 때문에 data(05,F1등을 명령어로 읽을 일은 없음)
                locctr = rMgr.register[8]; //현재 pc값 = locctr update
                //locctr 업데이트 후 그 다음부터 opcode를 가져옴
                if(locctr<0){
```

```
                        //종료 해야함
                        return -1;
                }
            rMgr.rsstAddrMemory=locctr;
                char[] ornopcode = rMgr.getMemory(locctr, 2);//2byte만 가져온다 (opcode,
e확인 때문)
                int[] arr;

                String opcode = String.format("%02X",ornopcode[0]&252);
                //1111 1100로 ni를 떼고 진짜 opcode만 가져온다
                switch(opcode){
                case "00"://LDA
                        addLog("LDA");
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3)); //3byte가져온다
                        rMgr.register[8]+=3; //pc값 변경
                        instLuncher.LDA(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                case "74"://LDT
                        addLog("LDT");
                        if((ornopcode[1] & 16) == 16){ //xbpe 0000 으로 여기서 e는 16이다
                                setCurInstruction(rMgr.getMemory(locctr, 4));
                                arr=parsing(rMgr.getMemory(locctr, 4)); //e==1, 4형식인
경우

                                rMgr.register[8]+=4;
                        }else{
                                setCurInstruction(rMgr.getMemory(locctr, 3));
                                arr=parsing(rMgr.getMemory(locctr, 3)); //3형식인 경우
                                rMgr.register[8]+=3;
                        }
                        instLuncher.LDT(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                case "50"://LDCH
                        addLog("LDCH");
                        if((ornopcode[1] & 16) == 16){
                                setCurInstruction(rMgr.getMemory(locctr, 4));
                                arr=parsing(rMgr.getMemory(locctr, 4));
                                rMgr.register[8]+=4;
                        }else{
                                setCurInstruction(rMgr.getMemory(locctr, 3));
                                arr=parsing(rMgr.getMemory(locctr, 3));
                                rMgr.register[8]+=3;
                        }
                        instLuncher.LDCH(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                case "0C"://STA
                        addLog("STA");
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                        instLuncher.STA(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                case "14"://STL
                        addLog("STL");
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                        instLuncher.STL(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                case "10"://STX
```

```
                addLog("STX");
                if((ornopcode[1] & 16) == 16){
                        setCurInstruction(rMgr.getMemory(locctr, 4));
                        arr=parsing(rMgr.getMemory(locctr, 4));
                        rMgr.register[8]+=4;
                }else{
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                }
                instLuncher.STX(new Instruction(arr[0],arr[1],arr[2]));
                break;
        case "54":STCH
                addLog("STCH");
                if((ornopcode[1] & 16) == 16){
                        setCurInstruction(rMgr.getMemory(locctr, 4));
                        arr=parsing(rMgr.getMemory(locctr, 4));
                        rMgr.register[8]+=4;
                }else{
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                }
                instLuncher.STCH(new Instruction(arr[0],arr[1],arr[2]));
                break;
        case "48":JSUB
                addLog("JSUB");
                if((ornopcode[1] & 16) == 16){
                        setCurInstruction(rMgr.getMemory(locctr, 4));
                        arr=parsing(rMgr.getMemory(locctr, 4));
                        rMgr.register[8]+=4;
                }else{
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                }
                instLuncher.JSUB(new Instruction(arr[0],arr[1],arr[2]));
                break;
        case "30":JEQ
                addLog("JEQ");
                setCurInstruction(rMgr.getMemory(locctr, 3));
                arr=parsing(rMgr.getMemory(locctr, 3));
                rMgr.register[8]+=3;
                instLuncher.JEQ(new Instruction(arr[0],arr[1],arr[2]));
                break;
        case "38":JLT
                addLog("JLT");
                setCurInstruction(rMgr.getMemory(locctr, 3));
                arr=parsing(rMgr.getMemory(locctr, 3));
                rMgr.register[8]+=3;
                instLuncher.JLT(new Instruction(arr[0],arr[1],arr[2]));
                break;
        case "3C":J
                addLog("J");
                setCurInstruction(rMgr.getMemory(locctr, 3));
                arr=parsing(rMgr.getMemory(locctr, 3));
                rMgr.register[8]+=3;
                instLuncher.J(new Instruction(arr[0],arr[1],arr[2]));
                break;
        case "B4":CLEAR(2형식)
                addLog("CLEAR");
                setCurInstruction(rMgr.getMemory(locctr, 2));
                arr=parsing(rMgr.getMemory(locctr, 2));
                rMgr.register[8]+=2;
```

```
                        instLuncher.CLEAR(new Instruction(arr[0],arr[1],arr[2],true));
                        break;
                case "28"://COMP
                        addLog("COMP");
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                        instLuncher.COMP(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                case "A0"://COMPR
                        addLog("COMPR");
                        setCurInstruction(rMgr.getMemory(locctr, 2));
                        arr=parsing(rMgr.getMemory(locctr, 2));
                        rMgr.register[8]+=2;
                        instLuncher.COMPR(new Instruction(arr[0],arr[1],arr[2],true));
                        break;
                case "B8"://TIXR
                        addLog("TIXR");
                        setCurInstruction(rMgr.getMemory(locctr, 2));
                        arr=parsing(rMgr.getMemory(locctr, 2));
                        rMgr.register[8]+=2;
                        instLuncher.TIXR(new Instruction(arr[0],arr[1],arr[2],true));
                        break;
                case "E0"://TD
                        addLog("TD");
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                        instLuncher.TD(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                case "D8"://RD
                        addLog("RD");
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                        instLuncher.RD(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                case "DC"://WD
                        addLog("WD");
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                        instLuncher.WD(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                case "4C"://RSUB
                        addLog("RSUB");
                        setCurInstruction(rMgr.getMemory(locctr, 3));
                        arr=parsing(rMgr.getMemory(locctr, 3));
                        rMgr.register[8]+=3;
                        instLuncher.RSUB(new Instruction(arr[0],arr[1],arr[2]));
                        break;
                }

                return 1;
        }

        /**
         * 남은 모든 instruction이 수행된 모습을 보인다.
         */
        public void allStep() {
                try{
                while(oneStep()>0);
                rMgr.closeDevice();
                }catch(Exception e){
```

```
                        e.printStackTrace();
                }
        }

        //현재 instruction Log를 visualSimulator에서 볼 수 있도록을 setting한다
        public void addLog(String log) {
                curLog=log;
        }

        //현재 instruction object code를 visualSimulator에서 볼 수 있도록을 setting한다
        public void setCurInstruction(char[] arr){
                String str="";
                if(arr.length==2){
                        str+=String.format("%02X", (int)arr[0]);
                        str+=String.format("%02X", (int)arr[1]);
                }else if(arr.length==4){
                        str+=String.format("%02X", (int)arr[0]);
                        str+=String.format("%02X", (int)arr[1]);
                        str+=String.format("%02X", (int)arr[2]);
                        str+=String.format("%02X", (int)arr[3]);
                }else{
                        //3형식
                        str+=String.format("%02X", (int)arr[0]);
                        str+=String.format("%02X", (int)arr[1]);
                        str+=String.format("%02X", (int)arr[2]);
                }

                curInstruction=str;
        }

        /**
         * 참조시 편한 Instuction을 만들기위해 parsing한다
         * int[0] opcode
         * int[1] nixbpe
         * int[2] disp
         * 2형식인 경우 int[1]=reg1, int[2]=reg2
         * 를 만들어 반환한다
         */
        public int[] parsing(char[] arr){
                int[] retarr=new int[3];
                if(arr.length==2){
                        retarr[0]=arr[0];//B4 12
                        retarr[1]=(arr[1]&240)>>4;//1111 0000 reg1, right shift
4해주어야한다
                        retarr[2]=(arr[1]&15);//0000 1111 reg2

                }else if(arr.length==4){
                        retarr[0]=(arr[0]&252);//1111 1100 opcode
                        retarr[1]=(arr[0]&3); //ni들어감
                        retarr[1] <<=4; //xbpe 들어갈 자리 만듦
                        int xbpe = (arr[1] >> 4); //4만큼 right shift 해주어야한다
                        retarr[1] |= xbpe;
                        retarr[2]=(arr[1]&15); //0000 1111 disp가 먼저 들어감
                        retarr[2] <<=8; //1byte는 8이므로 8만큼
                        retarr[2] |= arr[2];
                        retarr[2] <<=8;
                        retarr[2] |= arr[3]; //int는 4byte이므로 괜찮다
                }else{
```

```
                        //length==3인 경우
                        retarr[0]=(arr[0]&252);//1111 1100 opcode
                        retarr[1]=(arr[0]&3); //ni들어감
                        retarr[1] <<=4; //xbpe 들어갈 자리 만듦
                        int xbpe = (arr[1] >> 4); //4만큼 right shift 해주어야한다
                        retarr[1] |= xbpe;
                        retarr[2]=(arr[1]&15); //0000 1111 disp가 먼저 들어감
                        retarr[2] <<=8; //1byte는 8이므로 8만큼
                        retarr[2] |= arr[2];
                }

                //**F로 시작하면 음수 처리 해줘야함!!!**
                if(((retarr[2]>>11)&1)==1){
                        // bit시작이 1로 시작하는 음수일 경우, 앞에 FF를 처리해주어야 한다
                        long i1 = Long.parseLong("FFFFFF000",16);
                        retarr[2]|=(int)i1; //bit or로 앞부분도 다 FF처리 해주기

                }

                return retarr;
        }
}
```

(3) ResourceManager

```
package SP20_simulator;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.io.FileReader;
import java.io.FileWriter;



/**
 * ResourceManager는 컴퓨터의 가상 리소스들을 선언하고 관리하는 클래스이다.
 * 크게 네가지의 가상 자원 공간을 선언하고, 이를 관리할 수 있는 함수들을 제공한다.
 *
 * 1) 입출력을 위한 외부 장치 또는 device
 * 2) 프로그램 로드 및 실행을 위한 메모리 공간. 여기서는 64KB를 최대값으로 잡는다.
 * 3) 연산을 수행하는데 사용하는 레지스터 공간.
 * 4) SYMTAB 등 simulator의 실행 과정에서 사용되는 데이터들을 위한 변수들.
 *
 * 2번은 simulator위에서 실행되는 프로그램을 위한 메모리공간인 반면,
 * 4번은 simulator의 실행을 위한 메모리 공간이라는 점에서 차이가 있다.
 */
public class ResourceManager{
        /**
         * 디바이스는 원래 입출력 장치들을 의미 하지만 여기서는 파일로 디바이스를 대체한다.
         * 즉, 'F1'이라는 디바이스는 'F1'이라는 이름의 파일을 의미한다.
         * deviceManager는 디바이스의 이름을 입력받았을 때 해당 이름의 파일 입출력 관리 클래스를
리턴하는 역할을 한다.
         * 예를 들어, 'A1'이라는 디바이스에서 파일을 read모드로 열었을 경우, hashMap에 <"A1",
scanner(A1)> 등을 넣음으로서 이를 관리할 수 있다.
         * 변형된 형태로 사용하는 것 역시 허용한다.
         * 예를 들면 key값으로 String대신 Integer를 사용할 수 있다.
         * 파일 입출력을 위해 사용하는 stream 역시 자유로이 선택, 구현한다.
         * 이것도 복잡하면 알아서 구현해서 사용해도 괜찮습니다.
         */
        HashMap<String,Object> deviceManager = new HashMap<String,Object>();
        char[] memory = new char[65536]; // String으로 수정해서 사용하여도 무방함. 64KB
        int[] register = new int[10]; //regsiter 10개
        double register_F;
        int subReturnAddr; //stack 대신에 쓰는, sub routine을 누적해서 부를 때 L reg값을
저장해놓음

        SymbolTable symtabList; //section이름 - 주소, label-주소를 저장한다
        ArrayList<Integer> sectionLength; //section별 length를 저장한다
        String programName; //주 program 이름
        int programLength; //섹션을 다 합친 program 총 길이
```

```
        int programStartAddress; //program 시작 주소
        int addrOfFirstInst; //E record에서 나오는 첫 시작 주소

        boolean setName; //prgram 이름 세팅했는가?
        boolean setAddress; //시작 주소 세팅했는가?

    String rsusingDeviceName; //현재 쓰고있는 device의 이름
        int rstargetAddr; //현재 명령어의 target address를 저장
        int rsstAddrMemory; //현재 명령어의 시작 주소를 저장

        public ResourceManager(){
                symtabList = new SymbolTable();
                sectionLength = new ArrayList<Integer>();
                programLength = 0;
                setName = false;
                setAddress = false;
        }

        /**
         * 메모리, 레지스터등 가상 리소스들을 초기화한다.
         */
        public void initializeResource(){
                //멤버 변수에 쓰레기값이 남아있지 않도록 초기화

                for(int i=0 ; i<65536 ; i++){
                        memory[i]=0;
                }

                for(int i=0 ; i<10 ; i++){
                        register[i]=0;
                }

                register[2]=-1; //최종 return address에 -1넣기(종료 시그널)

        }

        /**
         * deviceManager가 관리하고 있는 파일 입출력 stream들을 전부 종료시키는 역할.
         * 프로그램을 종료하거나 연결을 끊을 때 호출한다.
         */
        public void closeDevice() throws IOException {
                Iterator<Object> iter=deviceManager.values().iterator();
                while(iter.hasNext()){
                        Object i = iter.next();
                        if(i.getClass().getSimpleName().equals("FileReader")){
                                ((FileReader)i).close(); //read나 write로 파일을 연 경우,
close를 해주어야 한다.
                        }else{
                                //FileWriter
                                ((FileWriter)i).flush();
                                ((FileWriter)i).close();
                        }
                }

                deviceManager.clear(); //모든 device를 지운다.
        }
```

```
	/**
	 * 디바이스를 사용할 수 있는 상황인지 체크. TD명령어를 사용했을 때 호출되는 함수.
	 * 입출력 stream을 열고 deviceManager를 통해 관리시킨다.
	 * @param devName 확인하고자 하는 디바이스의 번호,또는 이름
	 */
	public void testDevice(String devName) throws IOException {

		if(!deviceManager.containsKey(devName)){
			//device가 없다면 새로 생성해 넣는다
			File file = new File("./"+devName);
			if(!file.exists())
				file.createNewFile();

			deviceManager.put(devName, file); //먼저 file을 넣어놓는다.
		}
	}

	/**
	 * 디바이스로부터 원하는 개수만큼의 글자를 읽어들인다. RD명령어를 사용했을 때 호출되는
함수.
	 * @param devName 디바이스의 이름
	 * @param num 가져오는 글자의 개수
	 * @return 가져온 데이터
	 */
	public char[] readDevice(String devName, int num) throws IOException{

		Object o = deviceManager.get(devName);
		FileReader fr;
		if(o.getClass().getSimpleName().equals("File")){
			//아직 read 모드로 파일을 열지 않았을 때.
			//열어서 새로 넣어놓는다.
			fr = new FileReader((File)o);
			deviceManager.replace(devName, fr);
			o = fr;
		}
		fr = (FileReader)o;
		char[] cbuf = new char[num];
		if(fr.read(cbuf)==-1)
			return null;
		return cbuf;

	}

	/**
	 * 디바이스로 원하는 개수 만큼의 글자를 출력한다. WD명령어를 사용했을 때 호출되는 함수.
	 * @param devName 디바이스의 이름
	 * @param data 보내는 데이터
	 * @param num 보내는 글자의 개수
	 */
	public void writeDevice(String devName, char[] data, int num) throws IOException{
		Object o = deviceManager.get(devName);
		FileWriter fw;
		if(o.getClass().getSimpleName().equals("File")){
			//아직 read 모드로 파일을 열지 않았을 때.
			//열어서 새로 넣어놓는다.
			fw = new FileWriter((File)o);
```

```
                        deviceManager.replace(devName, fw);
                        o = fw;
                }
                fw = (FileWriter)o;
                fw.write(data);
        }

        /**
         * 메모리의 특정 위치에서 원하는 개수만큼의 글자를 가져온다.
         * @param location 메모리 접근 위치 인덱스
         * @param num 데이터 개수
         * @return 가져오는 데이터
         */
        public char[] getMemory(int location, int num){
                char[] arr = new char[num];
                for(int i=0;i<num;i++){
                        arr[i]=memory[location+i];
                }
                return arr;

        }

        /**
         * 메모리의 특정 위치에 원하는 개수만큼의 데이터를 저장한다.
         * @param locate 접근 위치 인덱스
         * @param data 저장하려는 데이터
         * @param num 저장하는 데이터의 개수
         */
        public void setMemory(int locate, char[] data, int num){
                for(int i=0;i<num;i++){
                        memory[locate+i]=data[i];
                }
        }

        /**
         * 번호에 해당하는 레지스터가 현재 들고 있는 값을 리턴한다. 레지스터가 들고 있는 값은
문자열이 아님에 주의한다.
         * @param regNum 레지스터 분류번호
         * @return 레지스터가 소지한 값
         */
        public int getRegister(int regNum){
                return register[regNum];
        }

        /**
         * 번호에 해당하는 레지스터에 새로운 값을 입력한다. 레지스터가 들고 있는 값은 문자열이
아님에 주의한다.
         * @param regNum 레지스터의 분류번호
         * @param value 레지스터에 집어넣는 값
         */
        public void setRegister(int regNum, int value){
                register[regNum]=value;
        }

        /**
         * 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. int값을 char[]형태로 변경한다.
```

```
         * @param data
         * @return
         */
        public char[] intToChar(int data){
                /**만약 음수라면 06안에 들어가지 않으므로 3byte안에 들어가도록 설정해야함**/
                if(data<0){
                        data &= (int)Math.pow(2, 24) -1;//FFFFFF(3byte)만 남게
                }
                String str = String.format("%06X",data); //무조건 3byte 반환
                char[] buf = new char[3];
                buf[0]=(char)Integer.parseInt(str.substring(0,2),16);
                buf[1]=(char)Integer.parseInt(str.substring(2,4),16);
                buf[2]=(char)Integer.parseInt(str.substring(4,6),16);
                return buf;
        }

        /**
         * 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. char[]값을 int형태로 변경한다.
         * @param data
         * @return
         */
        public int byteToInt(char[] data){ //char[]로 변경했다
                String str = "";
                for(char i : data){
                        str+=String.format("%02X", (int)i); //이렇게 02로 해주어야 길이가
맞게 잘 변형된다!
                        //Integer.toHexString을 하면 00이 안나오고 0이 나와서 다른 수가
되버림..ㅠ
                }
                //만약 앞이 F면 음수인데 음수러 변경되도록 처리해줘야함
                //그냥 FFFFFF는 -1이아니라 +1677...로 나옴
                //이 경우 FF를 덧붙여준다.
                if(data.length>=3 && str.charAt(0)=='F')
                        str = "FF"+str;
                return (int)Long.parseLong(str,16);
                //FFFFFFFF은 parseInt를 쓰면 에러가 난다. Long으로 해주어야 -1이 제대로 나온다.
        }
}
```

(4) SicLoader

```
package SP20_simulator;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.File;

/**
 * SicLoader는 프로그램을 해석해서 메모리에 올리는 역할을 수행한다. 이 과정에서 linker의 역할 또한 수행한다.
 * SicLoader가 수행하는 일을 예를 들면 다음과 같다.
 * - program code를 메모리에 적재시키기
 * - 주어진 공간만큼 메모리에 빈 공간 할당하기
 * - 과정에서 발생하는 symbol, 프로그램 시작주소, control section 등 실행을 위한 정보 생성 및 관리
 */
public class SicLoader {
	ResourceManager rMgr;

	int locctr; //section의 locctr
	int csAddr; //current section address
	int csLength; //현재 current section의 length

	public SicLoader(ResourceManager resourceManager) {
		// 필요하다면 초기화
		locctr=0;
		csAddr=0;
		csLength=0;
		setResourceManager(resourceManager);
	}

	/**
	 * Loader와 프로그램을 적재할 메모리를 연결시킨다.
	 * @param rMgr
	 */
	public void setResourceManager(ResourceManager resourceManager) {
		this.rMgr=resourceManager;
	}

	/**
	 * object code를 읽어서 load과정을 수행한다. load한 데이터는 resourceManager가 관리하는 메모리에 올라가도록 한다.
	 * load과정에서 만들어진 symbol table 등 자료구조 역시 resourceManager에 전달한다.
	 * @param objectCode 읽어들인 파일
	 */
	public void load(File objectCode) throws IOException{
		//program code를 메모리에 적재
		//pass1
		//메모리에 올라가는 주소는 current section address + locctr로 이루어진다
		BufferedReader br=new BufferedReader(new FileReader(objectCode));
		String line;
		while((line=br.readLine())!=null){
			switch(line.charAt(0)){
```

```
                        case 'H':
                                headRecord(line);
                                break;
                        case 'D':
                                int i=1; //D record 제외하고 하나씩 symtab에 넣는다
                                while(i<line.length()){
                                        rMgr.symtabList.putSymbol(line.substring(i, i+6),
Integer.parseInt(line.substring(i+6, i+12),16)); //hexstring to int
                                        i+=12;
                                }
                                break;
                        case 'E':
                                if(line.length()>1){
                                        //E말고 뒤에 주소가 있다면
                                        //Address of First Instruction in Object Program을
설정

        rMgr.addrOfFirstInst=Integer.parseInt(line.substring(1, 7),16);
                                }
                                break;
                                //남은 record들은 일단 넘긴다
                        }
                }
                br.close();

                //pass2
                br=new BufferedReader(new FileReader(objectCode));
                while((line=br.readLine())!=null){
                        switch(line.charAt(0)){
                        case 'H':
                                csAddr = rMgr.symtabList.search(line.substring(1, 7));
                                break;
                        case 'T':
                                textRecord(line);
                                break;
                        case 'M':
                                modifyRecord(line);
                                break;
                        }
                }
                br.close();
        }

        public void headRecord(String line){
                csAddr += csLength; //'이전' csLength를 누적하면 현재 cs의 시작 주소가 된다
                locctr = 0;
                //Program Name 설정(안되어있다면 설정)
                if(!rMgr.setName){
                        rMgr.programName=line.substring(1,7);
                        rMgr.setName=true;
                }
                //Start Address of Object Program 설정(안되어있다면 설정)
                if(!rMgr.setAddress){

        rMgr.programStartAddress=Integer.parseInt(line.substring(7,13),16);
                        rMgr.setAddress=true;

                }
                //section length 저장하고 더하기
                rMgr.programLength += Integer.parseInt(line.substring(13,19),16);
```

```
                rMgr.sectionLength.add(Integer.parseInt(line.substring(13,19),16));
                csLength = Integer.parseInt(line.substring(13,19),16); //csLength update

                //해당 Section Program symbolTable에 넣기
                rMgr.symtabList.putSymbol(line.substring(1,7), csAddr);
        }

        public void textRecord(String line){
                //이거는 그냥 2자리씩(1byte씩) 메모리에 올리기만 하는것이므로 음수 신경 안써도 됨
                locctr = Integer.parseInt(line.substring(1, 7),16); //hexstring to int
                int recordLength = Integer.parseInt(line.substring(7,9),16);
                int length=0;
                int i=9; //여기부터 2char를 1byte로 만들어서 char[] memory에 올려야 한다.
                while(length < recordLength){
                        char x = (char)Integer.parseInt(line.substring(i,i+2),16);
//16진수였으므로 표시를 해준다
                        rMgr.memory[csAddr+locctr+length]=x;
                        length++; //byte length는 하나 늘어남 1byte늘어난 것이므로
                        i+=2; //i는 2씩 늘어난다
                }
                locctr+=recordLength;
        }

        public void modifyRecord(String line){
                int origin = Integer.parseInt(line.substring(1, 7),16); //바꿀 address
                //바꿀 길이는 05나 06이나 3byte를 가져와야하는건 마찬가지이므로 계산하지 않았다.
                int changeDisp = rMgr.symtabList.search(line.substring(10));
                String str = "";

                for(int i=0; i<3;i++){
                        str += String.format("%02X",(int)rMgr.memory[csAddr+origin+i]);
//hex로 받아야함, 6개 되도록 맞춰서!
                        //1byte씩 가져와서 다시 합침
                }

                int originDisp = Integer.parseInt(str,16);

                if(line.charAt(9)=='+')
                        originDisp += changeDisp;
                else
                        originDisp -= changeDisp;

                //다시 1byte씩 올린다, int(10진수)상태에서는 1byte씩 올리기 어려우므로 다시
hexString으로 바꾼다
                str = String.format("%06X", originDisp);//**여섯자리가 안될수도 있음** 고로
빈공간 0으로 채우기
                int j=0;
                for(int i=0; i<3;i++){

        rMgr.memory[csAddr+origin+i]=(char)Integer.parseInt(str.substring(j, j+2),16);
                        //i는 하나씩, j는 2char=>1byte로 만들어야하므로 2씩 늘려줌으로써
1byte씩 저장한다
                        j+=2;
                }
        }

}
```

(5) InstLuncher

```
package SP20_simulator;

import java.io.IOException;

/* instruction에 따라 동작을 수행하는 메소드를 정의하는 클래스
*/
public class InstLuncher {
    ResourceManager rMgr;


    public InstLuncher(ResourceManager resourceManager) {
        this.rMgr = resourceManager;
    }

    // instruction 별로 동작을 수행하는 메소드를 정의
    // A0 X1 L2 B3 S4 T5 F6 PC8 SW9
    //n=i=1인 경우 무조건 p=1이므로 다른 처리는 하지 않았다.
    //copy에서만 돌아가도록 짰다

    public void LDA(Instruction inst){
        if((inst.nixbpe&48)==48){
                //n=i=1
                //target address 설정
                int ta = inst.disp + rMgr.register[8]; //disp+pc=TA
                rMgr.rstargetAddr=ta;
                //3byte(word) A reg에 저장
                char[] arr = rMgr.getMemory(ta, 3);
                int value = rMgr.byteToInt(arr);
                rMgr.register[0]=value;

        }else if((inst.nixbpe&16)==16){
                //i=1, target address 설정 필요 X
                rMgr.register[0]=inst.disp;
        }
    }


    public void LDT(Instruction inst){
        int ta;
        if((inst.nixbpe & 1)==1){
                //4형식
                        //target address 설정
                        ta = inst.disp; //ta가 절대값임(absolute, direct)
        }else{
                //3형식
                ta = inst.disp + rMgr.register[8]; //disp+pc=TA
        }
        //3byte(word) T reg에 저장
        rMgr.rstargetAddr=ta;
                char[] arr = rMgr.getMemory(ta, 3);
                int value = rMgr.byteToInt(arr);
                rMgr.register[5]=value;

    }

    public void LDCH(Instruction inst){
```

```
        int ta;
        if((inst.nixbpe & 1)==1){
                //4형식
                        //target address 설정
                        ta = inst.disp; //ta가 절대값임(absolute, direct)
        }else{
                //3형식
                ta = inst.disp + rMgr.register[8]; //disp+pc=TA
        }
        //1byte 읽어 A reg에 저장
        if((inst.nixbpe & 8)==8){
                //X가 있는 경우
                ta += rMgr.register[1];//X reg값을 TA에 더한다(index)
        }
        rMgr.rstargetAddr=ta;
                char[] arr = rMgr.getMemory(ta, 1);
                int value = rMgr.byteToInt(arr);
                rMgr.register[0]=value;
    }

    public void STA(Instruction inst){
        int ta = inst.disp + rMgr.register[8];
        rMgr.rstargetAddr=ta;
        char[] data=rMgr.intToChar(rMgr.register[0]); // A regi
        rMgr.setMemory(ta, data, 3);
    }

    public void STL(Instruction inst){
        int ta = inst.disp + rMgr.register[8];
        rMgr.rstargetAddr=ta;
        char[] data=rMgr.intToChar(rMgr.register[2]); //L regi
        rMgr.setMemory(ta, data, 3);
    }

    public void STX(Instruction inst){
        int ta;
        if((inst.nixbpe & 1)==1){
                //4형식
                        //target address 설정
                        ta = inst.disp; //ta가 절대값임(absolute, direct)
        }else{
                //3형식
                ta = inst.disp + rMgr.register[8]; //disp+pc=TA
        }
        //TA에 X에 있는 3byte(word)저장
        rMgr.rstargetAddr=ta;
        char[] data = rMgr.intToChar(rMgr.register[1]);
        rMgr.setMemory(ta, data, 3);
    }

    public void STCH(Instruction inst){
        int ta;
        if((inst.nixbpe & 1)==1){
                //4형식
                        //target address 설정
                        ta = inst.disp; //ta가 절대값임(absolute, direct)
        }else{
```

```
                //3형식
                ta = inst.disp + rMgr.register[8]; //disp+pc=TA
        }

        if((inst.nixbpe & 8)==8){
                //X가 있는 경우
                ta += rMgr.register[1];
        }
        //TA에 A에 있는 1byte(word)저장
        //char[] data = rMgr.intToChar(rMgr.register[0]); //이 함수가 무조건 3byte반환해서 쓸
수 없었다
        rMgr.rstargetAddr=ta;
        char[] data = new char[1];
        data[0]=(char)rMgr.register[0];
        rMgr.setMemory(ta, data, 1);
    }

    public void JSUB(Instruction inst){
        //PC값 변경
        rMgr.subReturnAddr=rMgr.register[2];
        rMgr.register[2]=rMgr.register[8]; //돌아올 주소(Lreg)에 PC값 저장

        if((inst.nixbpe & 1)==1){
        //4형식
                rMgr.register[8] = inst.disp; //relative가 아니므로 target address계산할 필요
없음
                rMgr.rstargetAddr=inst.disp;
        }else{
        //3형식
                int ta = inst.disp + rMgr.register[8]; //disp+pc=TA
                rMgr.register[8] = ta;
                rMgr.rstargetAddr=ta;
        }
    }

    public void JEQ(Instruction inst){
        //PC값 변경
        if(rMgr.register[9]!=0)
                return; //같지 않다면 돌아간다

        //같은 경우
        int ta = inst.disp + rMgr.register[8]; //disp+pc=TA
        rMgr.rstargetAddr=ta;
                rMgr.register[8] = ta;
    }

    public void JLT(Instruction inst){
        //PC값 변경
        if(rMgr.register[9]>=0)
                return; //작지 않다면 돌아간다

        //작은 경우
        int ta = inst.disp + rMgr.register[8]; //disp+pc=TA
        rMgr.rstargetAddr=ta;
                rMgr.register[8] = ta;
    }
```

```
public void J(Instruction inst){
    //PC값 변경
    if((inst.nixbpe&48)==48){
            //n=i=1
            //target address 설정
            int ta = inst.disp + rMgr.register[8]; //disp+pc=TA
            rMgr.rstargetAddr=ta;
            rMgr.register[8] = ta;

    }else if((inst.nixbpe&32)==32){
            //@=1 indirect
            int ta = inst.disp + rMgr.register[8]; //disp+pc=TA
            char[] arr = rMgr.getMemory(ta, 3);
            ta = rMgr.byteToInt(arr); //indirect이므로 이 주소로 가야한다
            rMgr.rstargetAddr=ta;
            rMgr.register[8] = ta; //target address로 pc값 변경
    }
}

public void CLEAR(Instruction inst){
    //2형식
    rMgr.register[inst.reg1]=0;//해당 register를 clear한다
}

public void COMP(Instruction inst){
    //target과 A와 값 비교 후 SW reg 세팅
    int ta;
    int value=0;
    if((inst.nixbpe&48)==48){
            //n=i=1
            //target address 설정
            ta = inst.disp + rMgr.register[8]; //disp+pc=TA
            rMgr.rstargetAddr=ta;
            char[] arr = rMgr.getMemory(ta, 3);
            value = rMgr.byteToInt(arr);


    }else if((inst.nixbpe&16)==16){
            //i=1, target address 설정 필요 X
            value = inst.disp;
    }
    rMgr.register[9]=rMgr.register[0]-value;
}

public void COMPR(Instruction inst){
    //2형식, reg1과 reg2를 비교한다.
    rMgr.register[9]=rMgr.register[inst.reg1]-rMgr.register[inst.reg2];
}

public void TIXR(Instruction inst){
    //2형식
    rMgr.register[1] += 1; //X하나 증가
    rMgr.register[9]=rMgr.register[1]-rMgr.register[inst.reg1];//reg1과 비교
    //reg1값이 들어가는게 아니라 reg1 index가 들어가는 거임. reg1이 X인지 A인지를 알려주는 것
    //SW에 (bit로 저장되지는 않음) X가 작으면 -1 X가 크면 1 X랑 같으면 0이 저장된다
}
```

```
    public void TD(Instruction inst) throws IOException{
        //Test Device
        //test할 device 이름은 한 byte만 읽어서 온다.
        int ta = inst.disp + rMgr.register[8]; //ta 설정
        rMgr.rstargetAddr=ta;
        char[] buf = rMgr.getMemory(ta,1);
        //16진수로 고쳐야함
        String devName = String.format("%02X",(int)buf[0]);
        rMgr.rsusingDeviceName=devName;
        rMgr.testDevice(devName);
        rMgr.register[9]=1; //test 준비가 되었으니 sw를 1로 만든다
    }

    public void RD(Instruction inst) throws IOException{
        //Read Device
        int ta = inst.disp + rMgr.register[8]; //ta 설정
        rMgr.rstargetAddr=ta;
        char[] buf = rMgr.getMemory(ta,1);
        String devName = String.format("%02X",(int)buf[0]);
        rMgr.rsusingDeviceName=devName;
        buf = rMgr.readDevice(devName, 1); //char 하나만 읽는다
        if(buf==null)
                rMgr.register[0]=0; //끝나면 null을 return하므로 0을 register에 넣는다
        else
                rMgr.register[0]=(int)buf[0];
    }

    public void WD(Instruction inst) throws IOException{
        //Write Device
        int ta = inst.disp + rMgr.register[8]; //ta 설정
        rMgr.rstargetAddr=ta;
        char[] buf = rMgr.getMemory(ta,1);
        String devName = String.format("%02X",(int)buf[0]);
        rMgr.rsusingDeviceName=devName;
        buf = new char[1];
        buf[0] = (char)rMgr.register[0]; //A reg 값을 그대로 넣는다.

        rMgr.writeDevice(devName, buf, 1); //A register 값 중 끝에 1byte만 파일에 쓴다
    }

    public void RSUB(Instruction inst){
        //L값->PC값에 저장
        rMgr.register[8]=rMgr.register[2]; //caller로 돌아감
        rMgr.register[2]=rMgr.subReturnAddr; //caller가 또 돌아갈 주소 값 복구
    }


}
```

(6) SymbolTable

```
package SP20_simulator;
import java.util.ArrayList;
import java.util.StringTokenizer;

/**
 * symbol과 관련된 데이터와 연산을 소유한다.
 * section 별로 하나씩 인스턴스를 할당한다.
 */
public class SymbolTable {
	ArrayList<String> symbolList;
	ArrayList<Integer> addressList;
	// 기타 literal, external 선언 및 처리방법을 구현한다.

	public SymbolTable(){
		symbolList = new ArrayList<String>();
		addressList = new ArrayList<Integer>();
	}


	/**
	 * 새로운 Symbol을 table에 추가한다.
	 * @param symbol : 새로 추가되는 symbol의 label
	 * @param address : 해당 symbol이 가지는 주소값
	 *
	 * 주의 : 만약 중복된 symbol이 putSymbol을 통해서 입력된다면 이는 프로그램 코드에 문제가
있음을 나타낸다.
	 * 매칭되는 주소값의 변경은 modifySymbol()을 통해서 이루어져야 한다.
	 */
	public void putSymbol(String symbol, int address){
		StringTokenizer tk = new StringTokenizer(symbol, " "); //혹시나 띄어쓰기가
들어온 경우, 없애기 위해
		symbolList.add(tk.nextToken());
		//이번엔 중복된 symbol이 있어도 가능해야 한다.
		//section별로 table을 하나씩 만드는 것이 아니라 table이 하나이기 때문(조교님이
말씀하심)
		addressList.add(address);
	}

	/**
	 * 기존에 존재하는 symbol 값에 대해서 가리키는 주소값을 변경한다.
	 * @param symbol : 변경을 원하는 symbol의 label
	 * @param newaddress : 새로 바꾸고자 하는 주소값
	 */
	public void modifySymbol(String symbol, int newaddress) {
		int index = symbolList.indexOf(symbol);
		if(index>=0)
			addressList.set(index, newaddress);
	}

	/**
	 * 인자로 전달된 symbol이 어떤 주소를 지칭하는지 알려준다.
	 * @param symbol : 검색을 원하는 symbol의 label
```

```
         * @return symbol이 가지고 있는 주소값. 해당 symbol이 없을 경우 -1 리턴
         */
        public int search(String symbol) {
                StringTokenizer tk = new StringTokenizer(symbol, " "); //혹시나 띄어쓰기가
들어온 경우, 없애기 위해
                int address = 0;
                int index = symbolList.indexOf(tk.nextToken());
                if(index<0)
                        address = -1;
                else
                        address = addressList.get(index);
                return address;
        }


}
```

(7) Instruction

```
package SP20_simulator;

/* Instruction 정보를 저장하는 Class */
public class Instruction {
        int opcode;
        int nixbpe;
        int disp;
        int reg1;
        int reg2;

        public Instruction(){
                this.opcode=0;
                this.nixbpe=0;
                this.disp=0;
        }

        public Instruction(int opcode, int nixbpe, int disp){
                //3,4형식
                this.opcode = opcode;
                this.nixbpe = nixbpe;
                this.disp = disp;
        }

        public Instruction(int opcode, int reg1, int reg2, boolean flag){
                //2형식
                this.opcode = opcode;
                this.reg1=reg1;
                this.reg2=reg2;
        }

}
```

5. 기대효과 및 결론

지금까지 프로젝트1과 2를 진행하면서, 두루뭉실했던 SIC/XE 머신의 동작을 잘 알게 되었다. 어떻게 소스 코드를 읽고, token 단위로 parsing하고, 기계어 코드를 만드는지 읽는 과정부터 출력하는 과정, 또 거기서 나아가 Object Code를 읽고 메모리에 올리고, 그것을 실행하는 동작까지 구현해볼 수 있는 좋은 기회였다. 머릿속으로 어렴풋이 생각만하고 있었던 동작 과정이 생각보다 더 복잡하단 것을 알았다.

아쉬웠던 점은, 프로젝트1과 마찬가지로 COPY 프로그램에 국한된 구현을 하다 보니 COPY 프로그램에만 잘 돌아가는 프로그램을 짰다는 것이다. 그러나 다른 명령어들까지 구현하기에 시간이 부족했다. 또, 원래는 Class의 멤버 변수는 private으로 선언하는 것이 대부분인데, 접근을 쉽게 하기 위해 default로 선언했다는 점이 고쳐야 할 부분 같다. get 함수 등을 만들지 않고 direct로 값을 바꾸는 것이 좋은 코드는 아닌 것 같다.

16진수를 생각하고 처리하는 부분이 제일 시간을 많이 들인 것 같다. char 배열을 하나씩 읽어와 합쳐서 int로 만들고, int를 또 char 배열로 만들어 하나씩 넣는데 음수로 만들어 주는 것 같은 생각하지도 못한 케이스들이 있어서 디버깅하는 데 애를 꽤나 먹었다.

UI를 처음으로 구현해보면서 당황스러웠지만 구현된 결과물을 보니 한 스텝마다 결과가 바로 나와서 이해도 쉽고 결과물이 눈에 보이니 뿌듯함도 배가 되는 것 같다. 프로젝트 1에서의 Assembler의 동작에 이어 Loader, Simulator까지 구현하면서 소스 코드부터 실행까지의 구현을 모두 손으로 해보았으니 이해가 더욱이 잘 되는 것 같다.