숭실대학교 Soongsil University

설계(프로젝트) 보고서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다.
3. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

교과목	시스템프로그래밍 2020
프로젝트 명	JAVA언어로 SIC/XE simulator 구현하기
교과목 교수	최 재 영
제출인	전자정보공학부 학번: 20160433 성명: 김민정 (출석번호: 108)
제출일	2020년 6월 4 일

차 례

1장 프로젝트 개요

1.1 개발 배경 및 목적

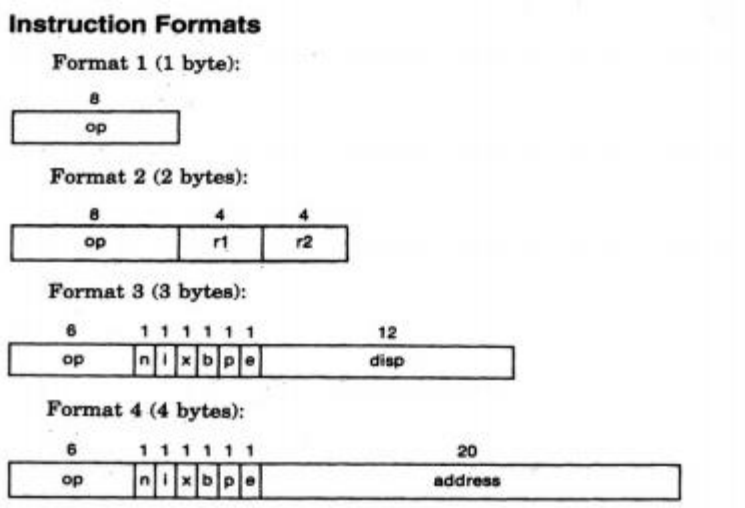
- 이 프로젝트는 objectProgram을 입력으로 받아 SIC/XE머신에서 로더와 링커의 흐름을 이해하여 그러한 기능을 하는 simulator를 만들어보는 프로젝트이다. SIC/XE머신의 링커 및 로더 역할을 하는 simulator를 개발해보면서 어셈블러에 대해 더 깊은 이해를 해볼 수 있다. 또한 object Program이 어떻게 구성되어 있는지를 다시한번 공부해보게 되고 실제로 메모리에서 object Program이 어떻게 실행되고 저장되는지를 이해하여 구현해볼 수 있다. 이를 통해 1학기에 배운 시스템프로그래밍의 전반적인 이해가 가능할 것이다. 또한 object program의 기능구현 뿐만 아니라 simulator의 GUI 개발을 통해 Swing에 대해 공부해볼 수 있는 기회도 가질 수 있다.

2장 배경 지식

2.1 주제에 관한 배경지식

1) opcode와 명령어 형식

- SIC/XE의 머신은 4종류의 형식을 지원한다.



이 때 비트 e의 값으로 3형식과 4형식을 구분한다. e=0이면 3형식을 따르고 e=1이면 4형식을 따른다.

2) 주소지정방식

- 주소지정방식에는 Absolute addressing, Indirect addressing, Immediate addressing, simple addressing, relative addressing이 존재한다.

3) object program format

- object program은 Header record, Define record, Refer record, Text record, Modification record, End record의 형식으로 되어 있는데 각각의 record들의 형식은 다음과 같다.

- Header record
  - Col. 1 H
  - Col. 2~7 Program name
  - Col. 8~13 Starting address of object program (Hex)
  - Col. 14~19 Length of object program in bytes (Hex)
- Define record
  - Col. 1 D
  - Col. 2-7 Defined external symbol name
  - Col. 8-13 Relative address of symbol within this control section
  - Col. 14-73 Repeat information in Col. 2-13 for other external symbols
- Refer record
  - Col. 1 R
  - Col. 2-7 Referred external symbol name
  - Col. 8-73 Names of other external reference symbols

- Text record
  - Col. 1 T
  - Col. 2~7 Starting address for object code in this record (Hex)
  - Col. 8~9 Length of object code in this record in bytes (Hex)
  - Col. 10~69 Object code in Hex (2 column per byte)
- Modification record *(revised)*
  - Col. 1 M
  - Col. 2-7 Starting address of the field to be modified
  - Col. 8-9 Length of the field to be modified
  - Col. 10 Modification flag (+ or -)
  - Col. 11-16 External symbol whose value is to be added to or subtracted from the indicated field
- End record
  - Col. 1 E
  - Col. 2~7 Address of first executable instruction in object (hex)

2.2 기술적 배경지식

1) 파일입출력

- 자바에서 입출력과 관련된 클래스들이 모여있는 패키지는 java.io패키지이다. 따라서 입출력 프로그램을 작성하려면 java.io패키지를 import시켜준 다음 프로그램을 작성해야 한다. 파일을 읽고쓰는 방법에는 다음과 같이 여러 방법이 존재한다. 먼저 파일을 읽는 방법이다. FileReader를 이용하여 파일을 읽을 수 있고 BufferedReader를 이용하여 파일을 읽을 수 있다. 그리고 Scanner를 이용하여 파일을 읽을 수 있고 DataInputStream을 이용하여 파일을 읽을 수 있다. 마찬가지로 FileWriter를 이용하여 파일에 데이터를 쓸 수 있고 BufferedWriter를 이용하여 파일에 데이터를 쓸 수 있다. 그리고 PrintWriter를 이용하여 파일에 데이터를 쓸 수 있고 FileoutputStream을 이용하여 파일에 데이터를 쓸 수 있다. 또한 자바에서 File I/O를 하기 위해서는 try-catch구문을 사용해야한다.

3) 비트연산

- 비트연산자에는 다음과 같은 연산자들이 있다.

&	비트단위 AND 연산
|	비트단위 OR 연산
^	비트단위 XOR 연산
~	비트단위 NOT 연산
>>	피연산자의 비트열을 오른쪽으로 이동
<<	피연산자의 비트열을 왼쪽으로 이동
>>>	피연산자의 비트열을 오른쪽으로 이동 가장 왼쪽의 비트는 항상 0으로 채워진다.

연산자 '<<'와 '>>'의 경우 시프트연산자라고도 하는데 이 연산자들의 사용방법은 다음과 같다. 'a << 2' 와 같이 사용한다면 이 연산의 의미는 다음과 같다. a의 비트열을 왼쪽으로 2칸 이동하라는 의미이다. 만일 'b >> 5'와 같이 사용했다면 이 연산의 의미는 b의 비트열을 오른쪽으로 5칸 이동하라는 의미니다.

4) 서식지정자를 활용한 입출력

- 다음과 같은 서식지정자들을 활용해서 입출력을 좀 더 편하게 할 수 있다. 자바에서는 System.out.printf(), String.format()등을 사용하여 서식지정자를 활용할 수 있다.

서식 지정자	설명
d,i	부호 있는 10진 정수
s	문자열
X	부호 없는 16진 정수(대문자)

플래그	설명
-	왼쪽 정렬
공백	양수일 때는 공백, 음수일 때는 -
0	출력하는 폭의 남는 공간에 0으로 채움

5) JAVA

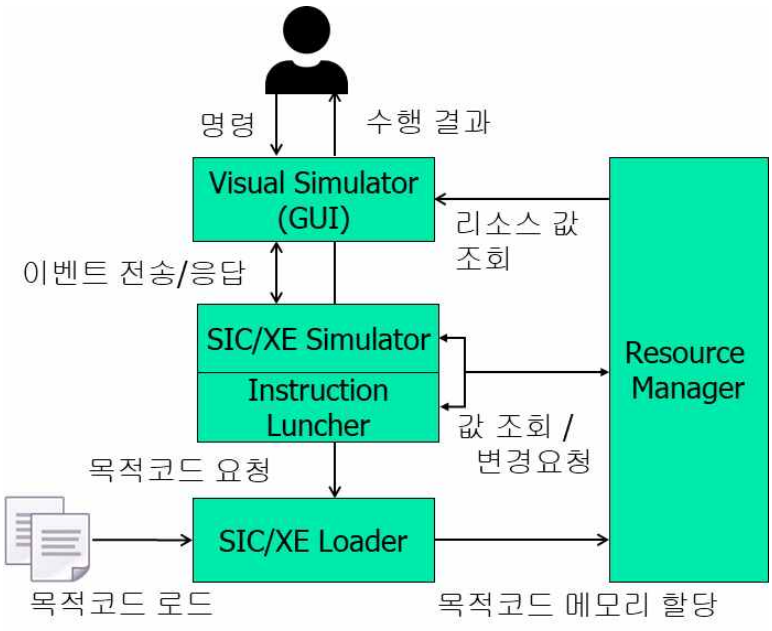
- JAVA란 자바로 지술된 프로그램 개발 및 실행을 할 수 있는 소프트웨어 모임의 총칭이다. 자바 프로그램은 운영체제나 하드웨어에 의존하지 않는 바이트코드인 추상적인 코드로 구현된다. 따라서, 자바 프로그램을 실행하기 위해서는 자바 가상머신(JVM)과 개발에 필요한 표준 라이브러리 세트와 컴파일러 환경만 맞추면 자바 프로그램은 모든 환경에서 동일하게 동작한다. 이러한 실행환경과 개발환경을 제공하는 것이 자바 플랫폼이다. JAVA는 처음부터 객체 지향 언어로 개발된 프로그래밍언어로 운영체제와는 독립적으로 실행할 수 있다. 또한 자동으로 메모리를 관리해주기 때문에 다른 언어에 비해 안정성이 높고 연산자 오버로딩을 금지하고 제네릭을 도입하여 코드의 가독성이 좋다. 그러나 자바는 자바 가상머신(JVM)에 의해 실행되기 때문에 다른 언어에 비해 실행속도가 느리고, 개발자가 일일이 예외처리를 지정해줘야한다는 불편함이 있는 언어이다.

6) Swing

- 스윙(Swing)은 자바에서 GUI를 구현하기 위해 JDK에서 기본적으로 제공하는 개발 툴킷이다. 스윙은 기존에 발표되었던 AWT(Abstract Window Toolkit)가 OS 및 윈도우 시스템의 자원을 그대로 제공하기 때문에 자바에서 지향하는 "Write Once, Run EveryWhere"를 구현하기 위해 각종 시스템에서 공통적으로 제공하는 버튼, 대화창 등만을 구현하고 표나 트리 등의 좀 더 복잡하고 다양한 그래픽 컴포넌트를 사용할 수 없는 단점을 보완하기 위하여 JDK 1.2버전부터 정식으로 JDK에 포함되었다. 스윙은 여러 환경에서 동일한 모습을 보일 수 있도록 구현되었으나 대신 해당 시스템의 고유한 모습을 보여줄 수 없다. 이러한 약점을 보완하기 위하여 Look And Feel이라는 기능을 지원하는데 이 기능을 이용하여 프로그램 전체의 UI 모습을 바꿀 수 있다.

3장 시스템 설계 내용

3.1 전체 시스템 설계 내용



3.2 모듈별 설계 내용

-기본적으로 주어진 메소드들을 이용하여 프로그램을 구현하였으나 필요에 의해 여러 메소드와 클래스변수들을 추가하였다.

1) SicLoader.java

* 클래스 : SicLoader

- 목적코드를 로드하는 클래스로 입력 프로그앰을 해석해서 메모리에 올리는 역할을 수행한다.

① void SetResourceManager(ResourceManager resourceManager)

- Loader와 프로그램을 적재할 메모리를 연결시키는 메소드이다.

② int load(File objectCode)

- object code를 읽어서 load 과정을 수행하는 메소드이다. 각 record별로 분리하여 필요한 정보들을 파싱하고 파싱한 데이터들을 resourceManager가 관리하는 메모리에 올라가도록 한다. load 과정에서 symbol table이 만들어진다. 이 자료구조 또한 resourceManager에 전달한다.

2) ResourceManager.java

* 클래스 : ResourceManager

- 컴퓨터의 가상 리소스들을 선언하고 관리하는 클래스이다. 입출력을 위한 외부장치 또는 device, 프로그램 로드 및 실행을 위한 메모리공간, 연산을 수행하는데 사용하는 레지스터공간, SYMTAB 등 simulator의 실행과정에서 사용되는 데이터들을 위한 변수들을 위한 가상 자원공간을 선언하고 이를 관리할 수 있는 함수들을 제공하는 클래스이다.

① void initializeResource()

- 메모리, 레지스터등 가상 리소스들을 초기화하는 메소드이다.

② void closeDevice()

– deviceManager가 관리하고 있는 파일 입출력 stream들을 전부 종료시키는 역할을 하는 메소드이다. 프로그램 종료버튼을 누를 때 사용된다.

③ void testDevice(String devName)

– 디바이스를 사용할 수 있는 상황인지 체크하는 메소드이다. TD 명령어를 사용했을 때 호출되는 메소드이다. 사용이 가능한 상황이라면 SW레지스터를 1로, 사용이 불가능한 상황이라면 SW레지스터를 0으로 설정한다.

④ char[] readDevice(String devName, int num)

– 디바이스로부터 원하는 개수만큼의 글자를 읽어들이는 메소드이다. RD명령어를 사용했을 때 호출되는 메소드이다. 읽어들인 글자들을 리턴값으로 반환한다.

⑤ void writeDevice(String devName, char[] data, int num)

–디바이스로 원하는 개수만큼의 글자를 출력하는 메소드이다. WD명령어를 사용했을 때 호출되는 메소드이다.

⑥ char[] getMemory(int location, int num)

– 메모리의 특정 위치에서 원하는 개수만큼의 데이터를 가져오는 메소드이다. 메모리에서 읽어들인 값들을 리턴값으로 반환한다.

⑦ void setMemory(int locate, char[] data, int num)

– 메모리의 특정 위치에 원하는 개수만큼의 데이터를 저장하는 메소드이다.

⑧ int getRegister(int regNum)

– 파라미터로 주어진 레지스터가 현재 가지고 있는 값을 리턴한다. 단 레지스터에 저장된 값은 int형이다.

⑨ String getProgName(int currentSection)

– 주어진 컨트롤섹션의 프로그램 이름을 가져오는 메소드이다.

⑩ int getProgLength(int currentSection)

– 주어진 컨트롤 =섹션의 프로그램 길이를 가져오는 메소드이다.

⑪ int getProgStart(int currentSection)

– 주어진 컨트롤섹션의 프로그램 시작주소를 가져오는 메소드이다.

⑫ int getCurrentSection()

–현재 컨트롤섹션을 가져오는 메소드이다.

⑬ int getProgTotalLen()

– 프로그램의 전체길이를 가져오는 메소드이다.

⑭ void setRegister(int regNum, int value)

– 번호에 해당하는 레지스터에 새로운 값을 입력하는 메소드이다.

⑮ void setProgName(String progName, int currentSection)

– 프로그램 이름을 저장하는 메소드이다.

⑯ void setProgLength(int progLength, int currentSection)

–프로그램 길이를 저장하는 메소드이다.

⑰ void setProgStart(int progStart, int currentSection)

– 프로그램 시작주소를 저장하는 메소드이다.

⑱ void setCurrentSection(int currentSection)

– 현재 컨트롤섹션을 저장해주는 메소드이다.

⑲ char[] intToChar(int data)

– int값을 char[] 형태로 변환해주는 메소드로 주로 레지스터와 메모리간의 데이터 교환에서 사용된다.

⑳ int byteToInt(char[] data)

- char[]값을 int 형태로 변환해주는 메소드로 주로 레지스터와 메모리간의 데이터 교환에서 사용된다.

3) SymbolTable.java

* 클래스 : SymbolTable

- symbol과 관련된 데이터와 연산을 가지고 있는 클래스이다.

① void putSymbol(String symbol, int location)

- 새로운 Symbol을 table에 추가하는 메소드이다.

② void modifySymbol(String symbol, int newLocation)

- 기존에 존재하는 symbol 값에 대해서 가리키는 주소값을 변경해주는 메소드이다.

③ void modSymbol(String symbol, int location, int size)

- modification record에 작성할 symbol들을 table에 추가해주는 메소드이다.

④ int search(String symbol)

- 인자로 전달된 symbol이 어떤 주소를 지칭하는지 알려주는 메소드이다.

⑤ String getSymbol(int index)

- index에 해당하는 symbol을 리턴해주는 메소드이다.

⑥ int getaddress(int index)

- index를 이용하여 symbol을 찾은 후 그 symbol에 해당하는 address을 리턴한다.

⑦ int getSize()

- symboltable의 크기을 리턴해주는 메소드이다.

4) InstLuncher.java

* 클래스 : InstLuncher

- instruction에 따라 동작을 수행하는 메소드를 정의하는 클래스이다. 각 instruction을 수행할 땐 이곳의 메소드들을 이용한다.

① void stl(int start, int format, int addressing)

- L 레지스터의 값을 메모리에 저장하는 메소드이다.

② void jsub(int start, int format, int addressing)

- PC에 저장되어 있는 값을 L레지스터에 저장후 메모리의 값을 PC 레지스터에 저장하는메소드이다.

③ void lda(int start, int format, int addressing)

- 메모리의 값을 읽어 A레지스터에 저장하는 메소드이다.

④ void comp(int start, int format, int addressing)

- A레지스터의 값과 메모리에 저장되어 있는 값을 비교하는 메소드이다.

⑤ void jeq(int start, int format, int addressing)

- sw레지스터가 0이라면 메모리에 저장된 값으로 이동하는 메소드이다.

⑥ void j(int start, int format, int addressing)

- 메모리에 저장된 값을 pc레지스터에 저장하는 메소드이다.

⑦ void sta(int start, int format, int addressing)

- A레지스터의 값을 메모리에 저장하는 메소드이다.

⑧ void clear(int start)

- 레지스터의 값을 0으로 초기화하는 메소드이다.

⑨ void ldt(int start, int format, int addressing)

- 메모리의 값을 읽어와 T레지스터에 저장하는 메소드이다.

⑩ String td(int start, int format, int addressing)

- 장치(파일)를 확인하는 메소드로 sw레지스터가 0이면 준비안된 것이고 0이 아니라면 준비된 것이다.

⑪ void rd(int start, int format, int addressing)

- 기기(파일)에서 한바이트 읽어와 A레지스터의 제일 오른쪽 1바이트에 저장하는 메소드이다.

⑫ void compr(int start)

- r1레지스터에 저장된 값과 r2 레지스터에 저장된 값을 비교하는 메소드이다.

⑬ void stch(int start, int format, int addressing)

- A레지스터의 제일 오른쪽 한바이트를 메모리에 저장하는 메소드이다.

⑭ void tixr(int start)

- X레지스터의 값을 1 증가시키고 r1레지스터와 비교하는 메소드이다.

⑮ void jlt(int start, int format, int addressing)

- SW 레지스터의 값이 -1이라면 메모리에 저장된 값을 PC레지스터에 저장하는 메소드이다.

⑯ void stx(int start, int format, int addressing)

- x레지스터의 값을 메모리에 저장하는 메소드이다.

⑰ void rsub()

- L레지스터에 저장된 값을 pc레지스터에 저장하는 메소드이다.

⑱ void ldch(int start, int format, int addressing)

- 메모리의 값을 A의 제일 오른쪽 한 바이트에 저장하는 메소드이다.

⑲ void wd(int start, int format, int addressing)

- A레지스터에 저장된 값 중 제일 오른쪽 한바이트를 장치(파일)에 저장하는 메소드이다.

5) SicSimulator.java

* 클래스 : SicSimulator

- simulator로서의 작업을 담당하는 클래스이다. VisualSimulator에서 사용자의 요청을 받으면 이에따라 ResourceManager에 접근하여 작업을 수행한다.

① void load(File program)

- 레지스터, 메모리 초기화 등 프로그램 load와 관련된 작업을 수행하는 메소드이다.

② void oneStep()

- 1개의 instruction이 수행된 모습을 보이는 메소드이다.

③ void allStep()
- 남은 모든 instruction이 수행된 모습을 보이는 메소드이다.
④ void addLog(String log)
- 각 단계를 수행할 때마다 관련된 기록을 남기도록 하는 메소드이다.
⑤ void addInstruction(String instruction)
-각 단계를 수행할 때 마다 수행한 명령어 code를 남기도록 하는 메소드이다.
⑥ List<String> getLogList()
- 지금까지 수행한 로그기록들의 리스트를 반환하는 메소드이다.
⑦ List<String> getInstList()
- 지금까지 수행한 code기록들의 리스트를 반환하는 메소드이다.
⑧ String getDevice()
- 현재 사용중인 장치(파일)이름을 반환하는 메소드이다.
⑨ int getAddress()
- 현재 진행중인 명령어의 주소를 반환하는 메소드이다.
⑩ String makingCode(char[] instruction, int format)
- 현재 수행중인 code를 기록으로 남기기 위해 char[]를 String으로 바꿔주는 메소드이다.

5) VisualSimulator.java

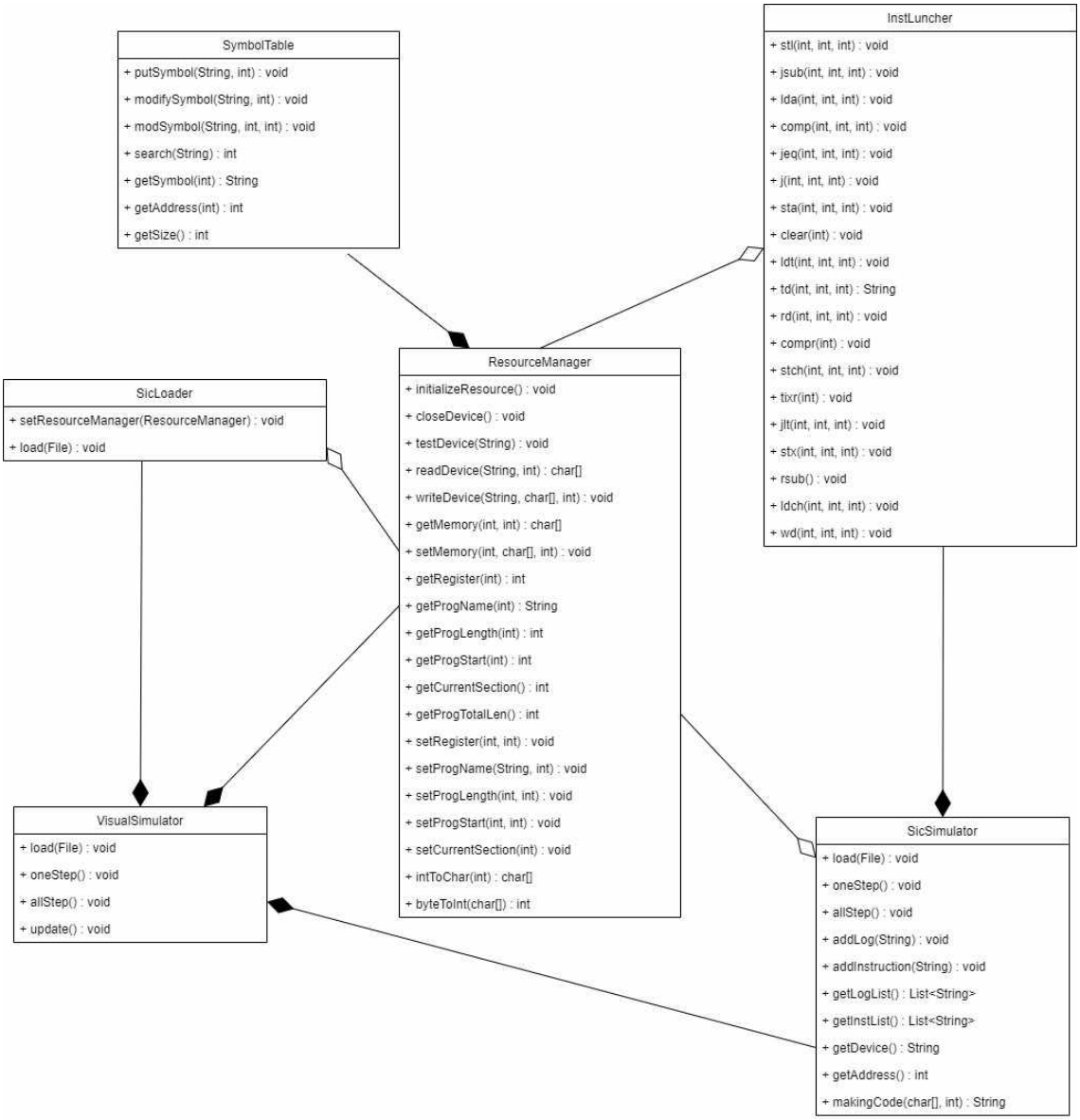
* 클래스 : VisualSimulator

- 사용자와의 상호작용을 담당하는 simulator 클래스이다. 사용자에게서 버튼 클릭등의 이벤트를 받으면 그것을 전달하고 그에따른 결과값을 화면에 업데이트하는 역할을 수행한다. 이 클래스는 사용자에게 입력을 받고 출력을 보여주는 역할만을 담당하고 실질적인 프로그램작업은 SicSimulator에서 수행된다.

① void load(File program)
- 프로그램 로드 명령을 전달하는 메소드이다.
② void oneStep()
- 하나의 명령어만 수행할 것을 SicSimulator에 요청하는 메소드이다.
③ void allStep()
- 남아있는 모든 명령어를 수행할 것을 SicSimulator에 요청하는 메소드이다.
④ void addLog(String log)
- 각 단계를 수행할 때마다 관련된 기록을 남기도록 하는 메소드이다.
⑤ void addInstruction(String instruction)
-각 단계를 수행할 때 마다 수행한 명령어 code를 남기도록 하는 메소드이다.
⑥ void update()
- 화면을 최신값으로 갱신하는 역할을 수행하는 메소드이다.

4장 시스템 구현 내용 (구현 화면 포함)

4.1 전체 시스템 구현 내용



4.2 모듈별 구현 내용

1) SicLoader.java

- 목적코드를 로드하는 모듈이다. 목적코드를 읽어들인 후 각각의 레코드별로 구분짓는다. 그러고나서 각 라인마다 필요한 정보들을 파싱하고 저장해주는 역할을 수행한다. pass1에서는 목적코드를 읽은 후 필요한 정보들을 파싱하여 저장하는 과정을 수행하고 pass2에서는 modify record에 기록되어 있는 symbol들을 찾아 해당 symbol이 들어있는 메모리의 값을 수정해주는 과정을 수행한다.

2) ResourceManager.java

- 레지스터나 메모리, device 같이 컴퓨터의 가상 리소스들을 선언하고 관리해주는 모듈이다. 이곳에서는 메모리에 값을 입력해주거나 메모리에서 값을 읽어오고, 레지스터에 값을 입력해주거나 레지스터에서 값을 읽어온다. 또한 device에 데이터를 입력해주거나 device에서 데이터를

읽어오는 역할을 하는 메소드들을 포함하는 모듈이다. 컴퓨터의 가상 리소스들에 대한 것들은 모두 이 모듈을 통해 수행된다.

3) SymbolTable.java

- 프로그램에 사용되는 Symbol들을 ArrayList로 만들어서 저장하였다. modify 해야할 Symbol들은 해당하는 Symbol들과 address, size를 각각 저장하여 pass2과정에서 이 정보들을 활용하여 메모리값을 수정해주었다.

4) InstLuncher.java

- 각각의 instruction을 수행하면서 수행되는 동작들을 이곳에 정의하였다. SicSimulator에서 해당 명령어에 해당하는 메소드들을 호출하면 해당 명령어 메소드가 호출되면서 레지스터와 메모리의 정보를 수정하는 동작이 이곳에서 수행된다.

5) SicSimulator.java

- VisualSimulator가 사용자에게 보여지는 GUI를 담당하는 곳이라면 이곳은 VisualSimulator가 보낸 이벤트를 받아 그 이벤트를 실질적으로 수행하는 모듈이다. step by step으로 명령어가 처리되며 수행되어야 하는 명령어를 받으면 먼저 해당 명령어가 어떠한 주소방식을 사용하는지를 체크한다. 또한 이 명령어가 4형식명령어인지를 구분한 다음 명령어의 instruction이 어떤 것인지를 찾는다. 수행되어야하는 명령어의 instruction을 찾으면 그 instruction이 수행되도록 해당 instruction의 메소드를 호출해준다. 또한 이곳에서는 사용자에게 출력해 줄 log를 만들어주는 동작도 수행한다.

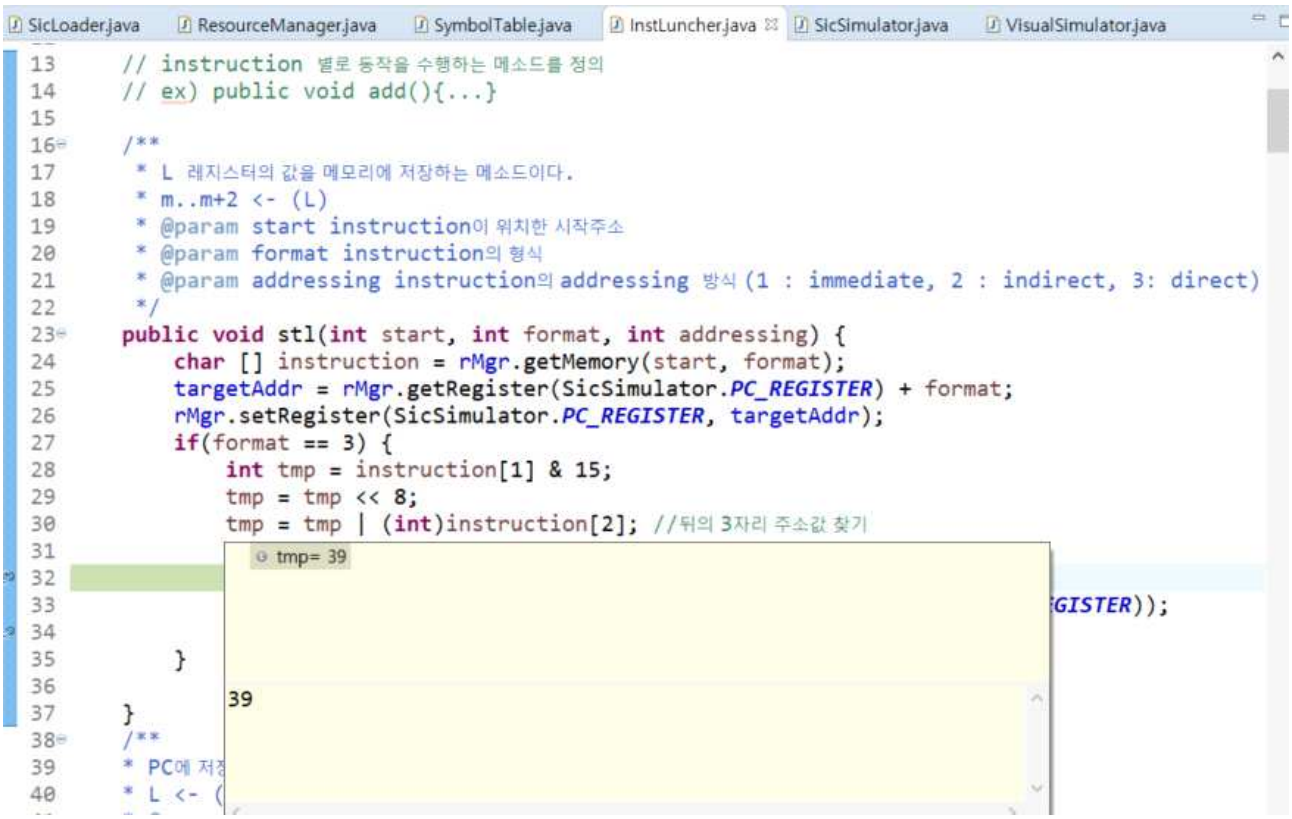
6) VisualSimulator.java

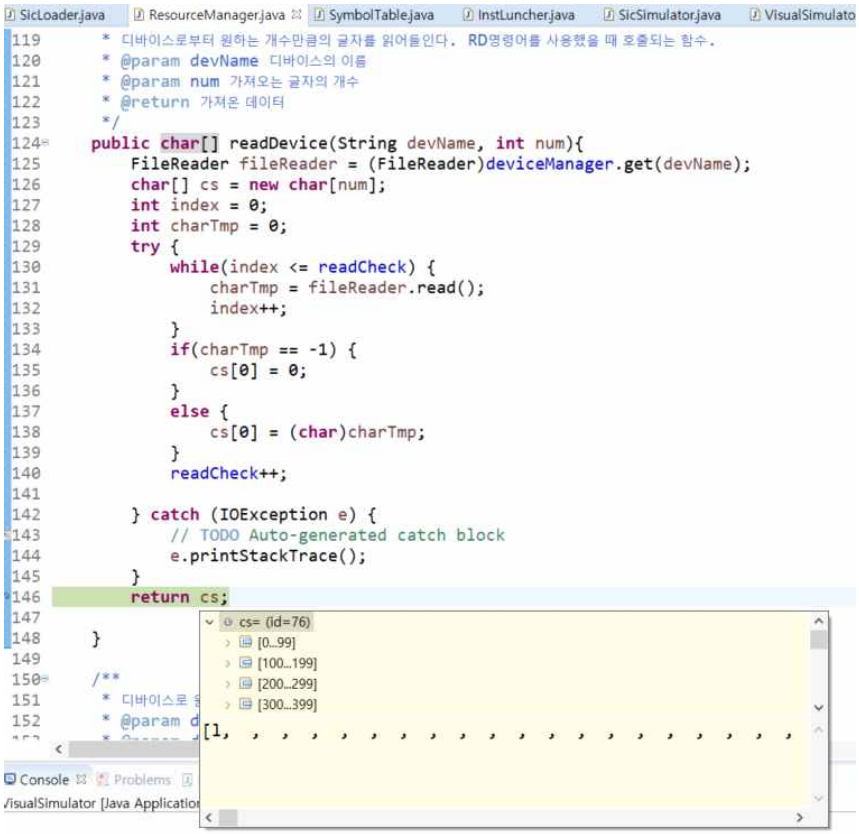
- 사용자에게 보여지는 GUI를 담당하는 곳이다. 사용자의 입력을 이곳에서 받아들인 후 다른 모듈들에게 그 입력을 전달하고 다른 모듈들에서 입력에 해당되는 동작들을 수행한다. 그 후 바뀐 레지스터나 메모리의 정보들을 이곳에서 새로 업데이트하여 그 정보들을 사용자에게 보여준다.

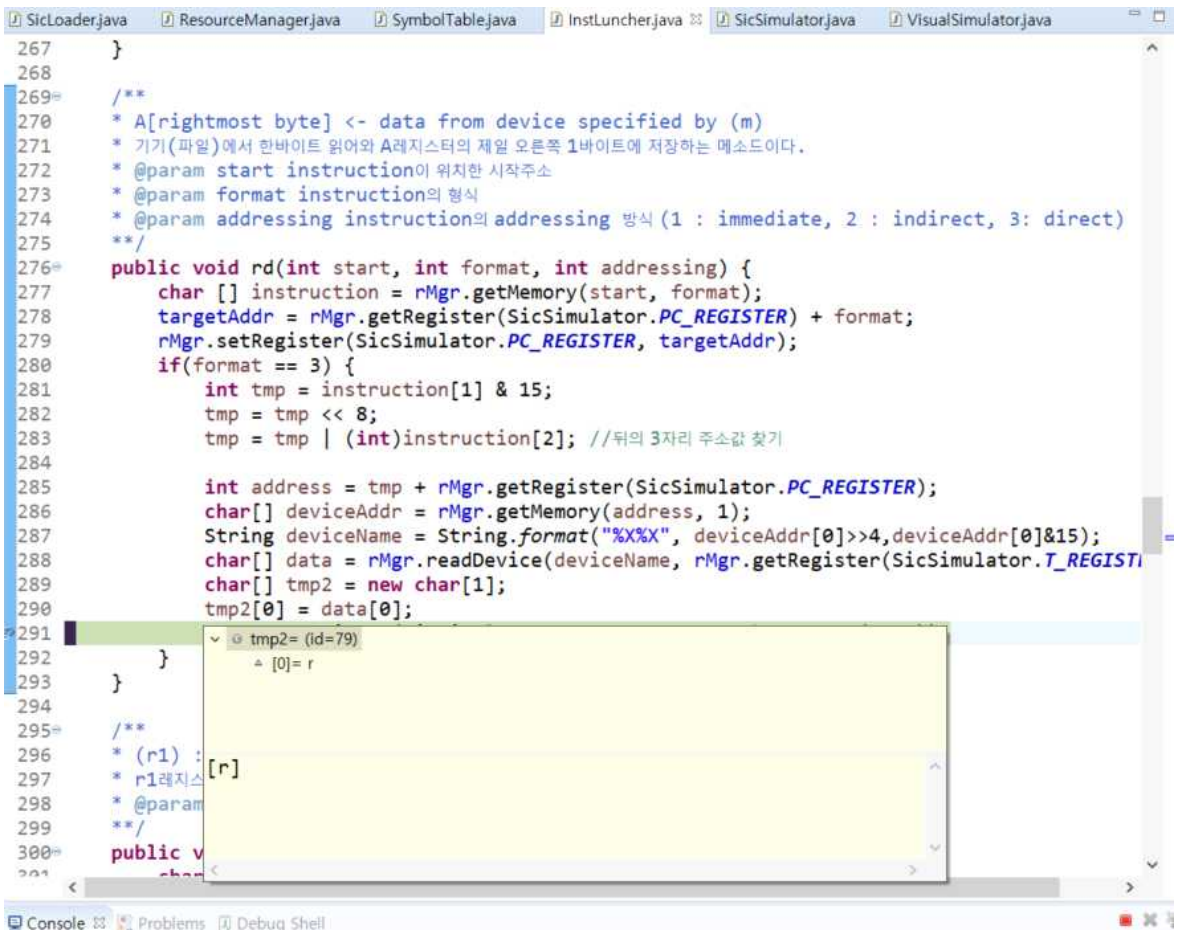
4.3 기존에 생성해놓은 F1 파일 내용

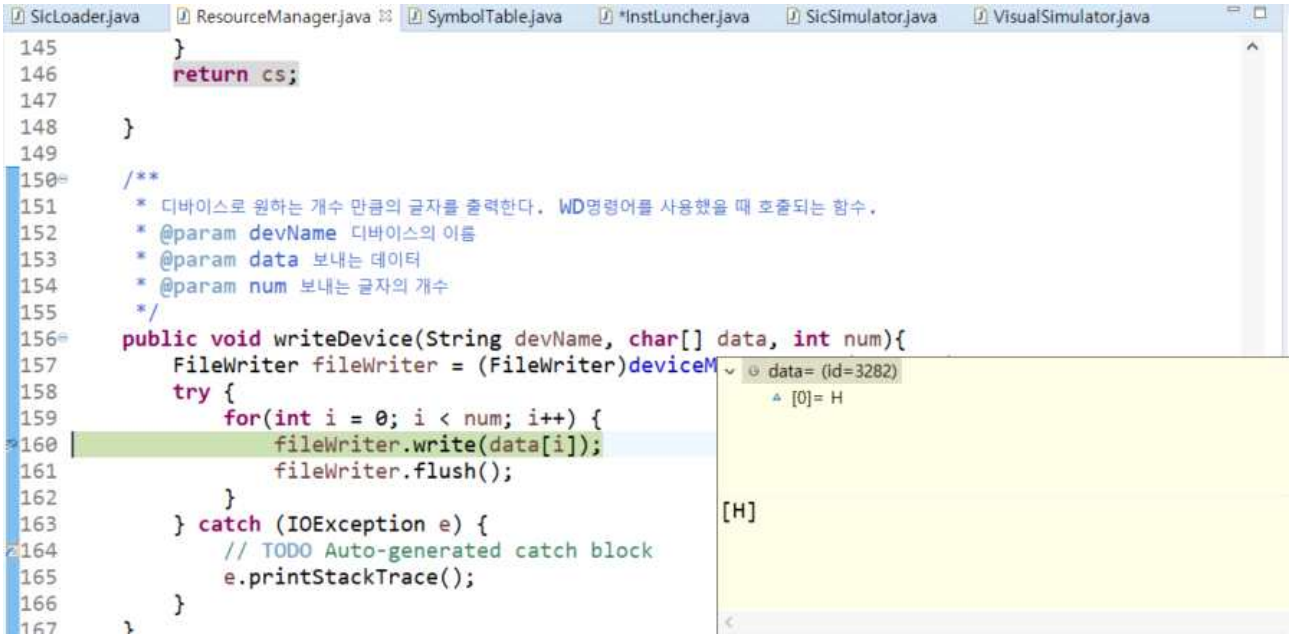
4.4 디버깅 과정

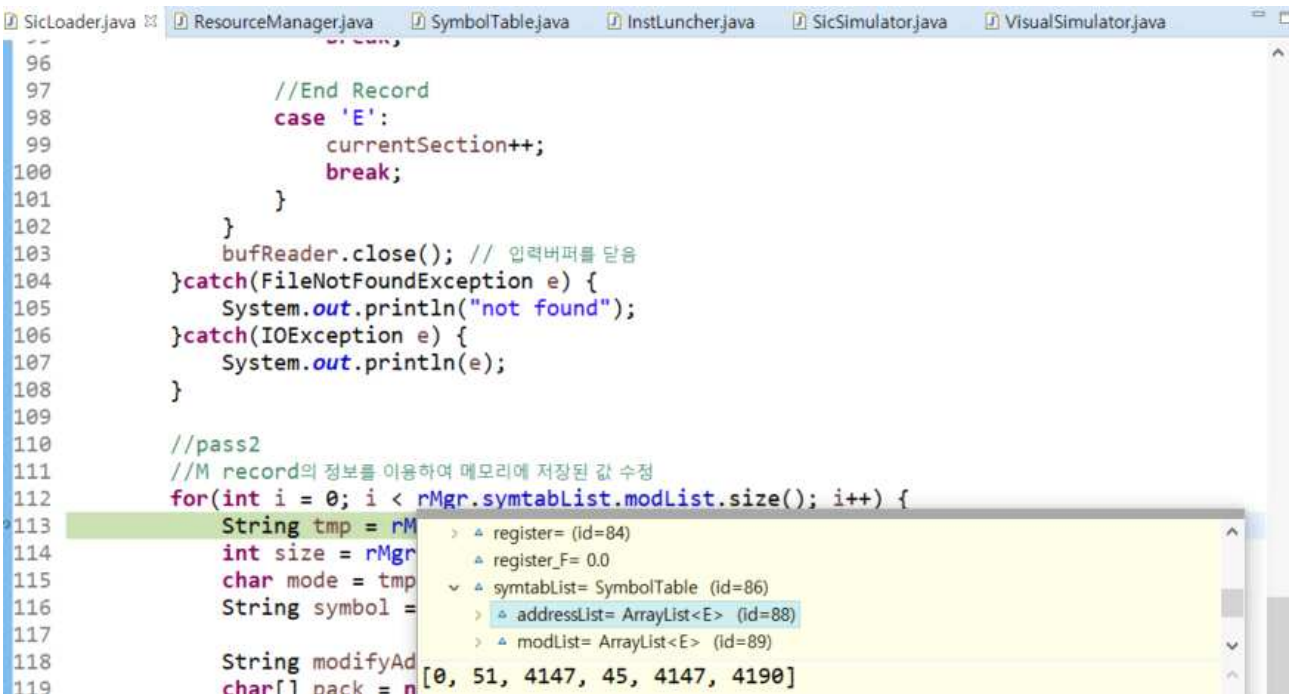
-프로그램을 구현하면서 디버깅했던 내용들 중 일부화면들을 첨부하였다.




SicLoader.java
ResourceManager.java
SymbolTable.java
InstLuncher.java
SicSimulator.java
VisualSimulato
119 * 디바이스로부터 원하는 개수만큼의 글자를 읽어들인다. RD명령어를 사용했을 때 호출되는 함수.
120 * @param devName 디바이스의 이름
121 * @param num 가져오는 글자의 개수
122 * @return 가져온 데이터
123 */
124 public char[] readDevice(String devName, int num){
125 FileReader fileReader = (FileReader)deviceManager.get(devName);
126 char[] cs = new char[num];
127 int index = 0;
128 int charTmp = 0;
129 try {
130 while(index <= readCheck) {
131 charTmp = fileReader.read();
132 index++;
133 }
134 if(charTmp == -1) {
135 cs[0] = 0;
136 }
137 else {
138 cs[0] = (char)charTmp;
139 }
140 readCheck++;
141
142 } catch (IOException e) {
143 // TODO Auto-generated catch block
144 e.printStackTrace();
145 }
146 return cs;
147
148 }
149
150 /**
151 * 디바이스로
152 * @param
cs= (id=76)
[0...99]
[100...199]
[200...299]
[300...399]
[1, ,
Console
Problems
/isualSimulator [Java Applicatio


SicLoader.java
ResourceManager.java
SymbolTable.java
InstLuncher.java
SicSimulator.java
VisualSimulator.java
267 }
268
269 /**
270 * A[rightmost byte] <- data from device specified by (m)
271 * 기기(파일)에서 한바이트 읽어와 A레지스터의 제일 오른쪽 1바이트에 저장하는 메소드이다.
272 * @param start instruction이 위치한 시작주소
273 * @param format instruction의 형식
274 * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
275 **/
276 public void rd(int start, int format, int addressing) {
277 char [] instruction = rMgr.getMemory(start, format);
278 targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
279 rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
280 if(format == 3) {
281 int tmp = instruction[1] & 15;
282 tmp = tmp << 8;
283 tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기
284
285 int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
286 char[] deviceAddr = rMgr.getMemory(address, 1);
287 String deviceName = String.format("%X%X", deviceAddr[0]>>4,deviceAddr[0]&15);
288 char[] data = rMgr.readDevice(deviceName, rMgr.getRegister(SicSimulator.T_REGIST
289 char[] tmp2 = new char[1];
290 tmp2[0] = data[0];
291
292 }
293 }
294
295 /**
296 * (r1) :
297 * r1레지스
298 * @param
299 **/
300 public v
tmp2= (id=79)
[0]= r
[r]
Console
Problems
Debug Shell


SicLoader.java
ResourceManager.java
SymbolTable.java
*InstLuncher.java
SicSimulator.java
VisualSimulator.java
145 }
146 return cs;
147
148 }
149
150 /**
151 * 디바이스로 원하는 개수 만큼의 글자를 출력한다. WD명령어를 사용했을 때 호출되는 함수.
152 * @param devName 디바이스의 이름
153 * @param data 보내는 데이터
154 * @param num 보내는 글자의 개수
155 */
156 public void writeDevice(String devName, char[] data, int num){
157 FileWriter fileWriter = (FileWriter)deviceM
158 try {
159 for(int i = 0; i < num; i++) {
160 fileWriter.write(data[i]);
161 fileWriter.flush();
162 }
163 } catch (IOException e) {
164 // TODO Auto-generated catch block
165 e.printStackTrace();
166 }
data= (id=3282)
[0]= H
[H]


SicLoader.java
ResourceManager.java
SymbolTable.java
InstLuncher.java
SicSimulator.java
VisualSimulator.java
96
97 //End Record
98 case 'E':
99 currentSection++;
100 break;
101 }
102 }
103 bufReader.close(); // 입력버퍼를 닫음
104 }catch(FileNotFoundException e) {
105 System.out.println("not found");
106 }catch(IOException e) {
107 System.out.println(e);
108 }
109
110 //pass2
111 //M record의 정보를 이용하여 메모리에 저장된 값 수정
112 for(int i = 0; i < rMgr.symtabList.modList.size(); i++) {
113 String tmp = rM
114 int size = rMgr
115 char mode = tmp
116 String symbol =
117
118 String modifyAd
119 char[] pack = n
register= (id=84)
register_F= 0.0
symtabList= SymbolTable (id=86)
addressList= ArrayList<E> (id=88)
modList= ArrayList<E> (id=89)
[0, 51, 4147, 45, 4147, 4190]

SicLoader.java | ResourceManager.java | SymbolTable.java | InstLuncher.java | SicSimulator.java | VisualSimulator.java

```
368
369     }
370
371     /**
372     * PC <- m if CC set to <(-1)
373     * SW 레지스터의 값이 -1이라면 메모리에 저장된 값을 PC레지스터에 저장하는 메소드이다.
374     * @param start instruction이 위치한 시작주소
375     * @param format instruction의 형식
376     * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
377     **/
378     public void jlt(int start, int format, int addressing) {
379         char [] instruction = rMgr.getMemory(start, format);
380         targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
381         rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
382         if(format == 3) {
383             if(rMgr.getRegister(SicSimulator.SW_REGISTER) == -1) {
384                 int tmp = instruction[1] & 15;
385                 tmp = tmp << 8;
386                 tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기
387                 if((instruction[1] & 15) == 15) {//F로 시작하는 주소의 경우 앞부분을 다 F로 채워넣기..
388                     tmp = tmp |((0xFFFFF)<<12);
389                 }
390                 int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
391                 rMgr
392             }
393         }
394     }
395
396     /**
397     //m .. m+2 <- (X
```

address= 4156

4156

SicLoader.java | ResourceManager.java | SymbolTable.java | *InstLuncher.java | SicSimulator.java | VisualSimulator.java

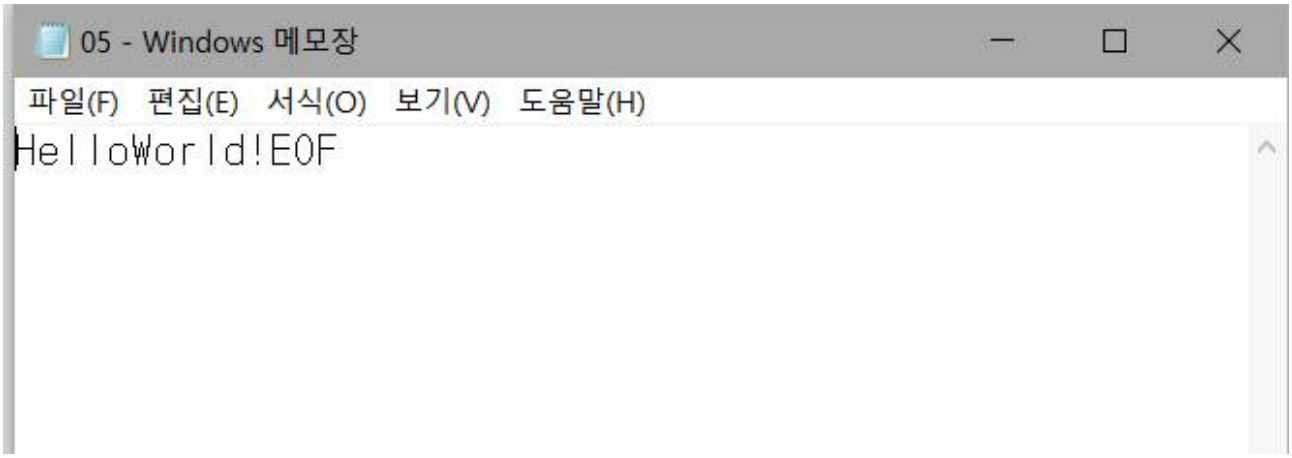
```
147     }
148
149     /***
150     * PC <- m
151     * 메모리에 저장된 값을 pc레지스터에 저장하는 메소드이다.
152     * @param start instruction이 위치한 시작주소
153     * @param format instruction의 형식
154     * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
155     */
156     public void j(int start, int format, int addressing) {
157         char [] instruction = rMgr.getMemory(start, format);
158         targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
159         rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
160         if(format == 3) {
161             int tmp = instruction[1] & 15;
162             tmp = tmp << 8;
163             tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기
164             if((instruction[1] & 15) == 15) {//F로 시작하는 주소의 경우 앞부분을 다 F로 채워넣기..
165                 tmp = tmp |((0xFFFFF)<<12);
166             }
167             int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
168             if(addressing == 2){ //indirect addressing이라면
169                 address = rMgr.byteToInt(rMgr.getMemory(address, 3));
170             }
171             rMgr
172         }
173     }
174
175     /**
176     * m..m+2 <-
177     * A레지스터의 값
```

address= 0

0

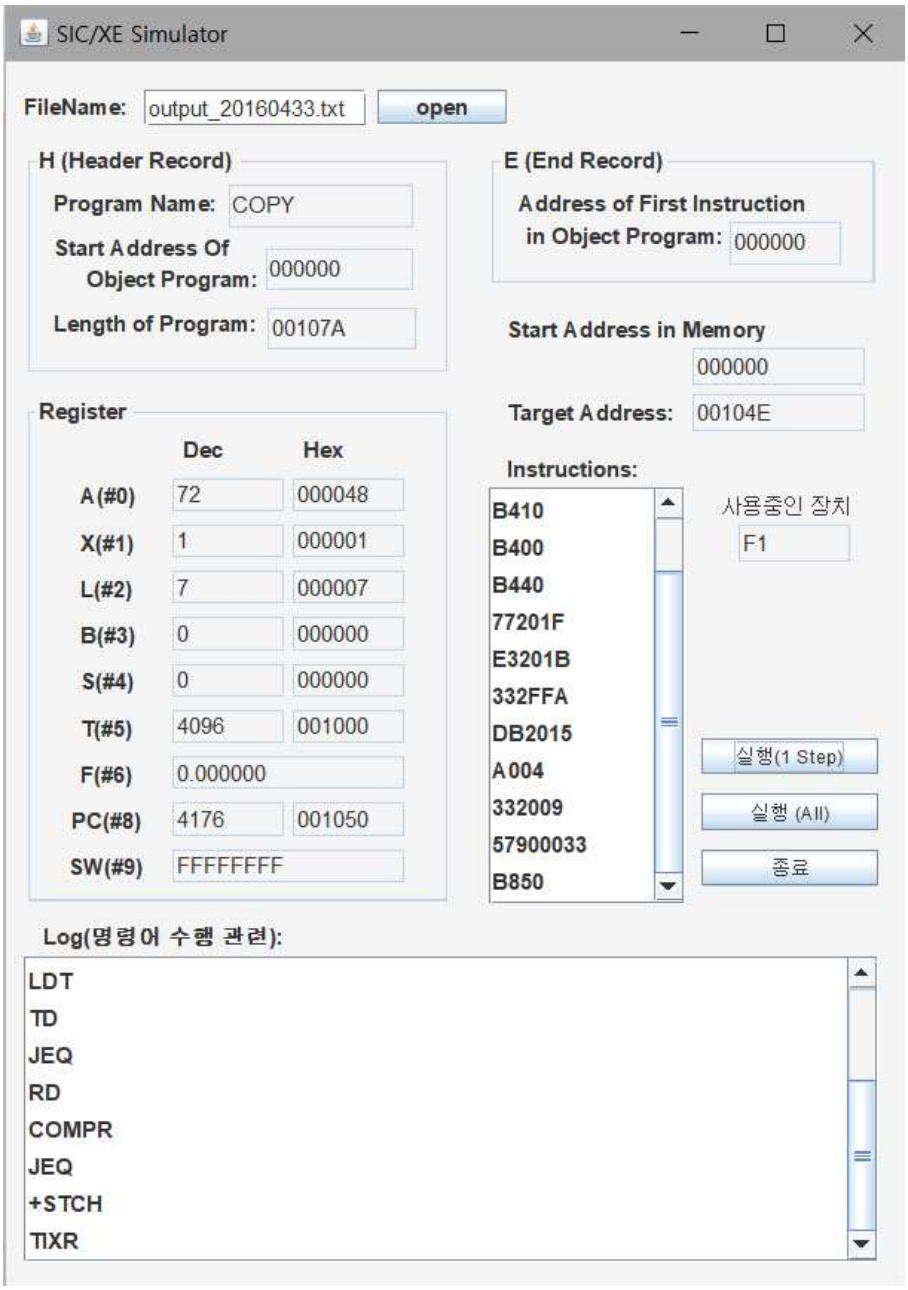
4.5 수행 결과

1) 05의 내용

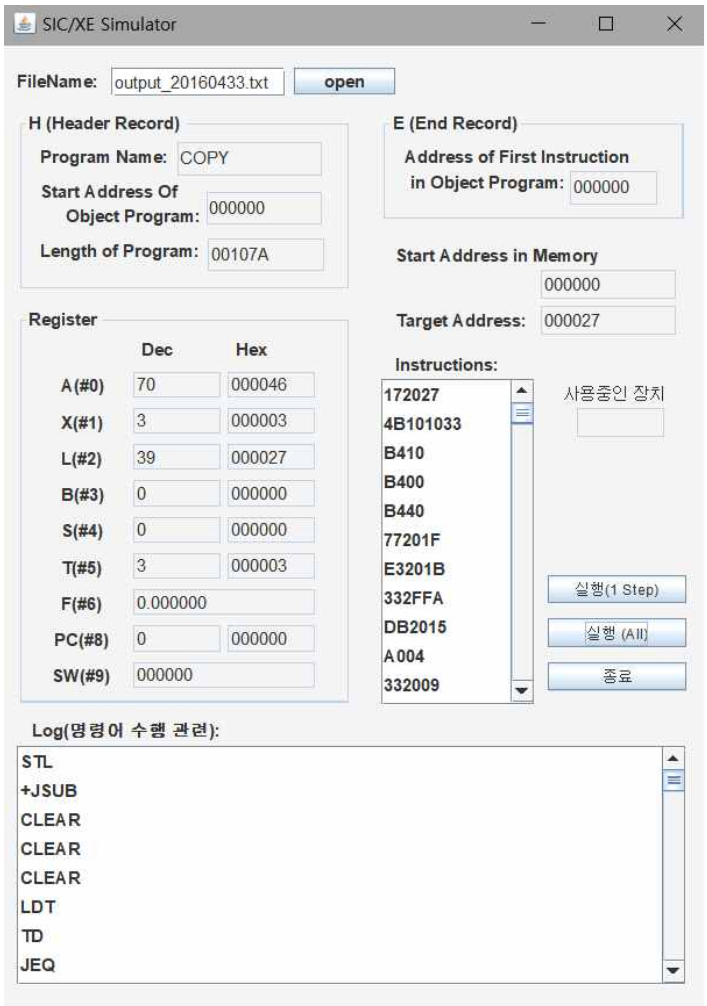


2) simulator 초기 화면

3) simulator 중간과정



4) simulator 프로그램 수행을 모두 끝낸 화면



5장 기대효과 및 결론

-이번 프로젝트는 이전 프로젝트로 만든 결과인 object Program을 input파일로 하여 프로그램을 실행하는 simulator를 Java를 이용하여 만드는 프로젝트였다. 먼저 Swing을 이용하여 GUI를 만들었다. 그리고나서 object Program을 읽은 후 각각의 record에 맞추어 읽고 명령어의 형식을 파악한 후에 그 형식에 맞추어 object code를 분리하였다. 명령어를 단계별로 진행하면서 step-by-step별로 레지스터와 메모리의 값을 변경해보는 작업까지 해야하는 프로젝트였어서 처음에 어려움을 겪었었다. 이전 프로젝트들을 진행하면서 어셈블러를 object code로 변환하는 과정은 많이 익숙해졌다고 생각했는데 이번 프로젝트를 하면서 그게 아니었다는 것을 알 수 있었다. 오히려 거꾸로 다시 변환하는 과정을 해보니 object Program에 대해 정말 확실하게 알 수 있었던 것 같다. 이번 프로젝트로 만든 결과물은 정말 COPY 프로그램만 수행할 수 있는 프로그램이다. 시간이 된다면 대부분의 상황에서 simulator 수행이 가능하게 만들고 싶어서 프로그램을 크게 바꾸지 않고 몇몇 부분만 추가하면 가능하게끔 프로그램을 구성하였지만 이번 프로젝트 제출 기한 내에는 불가능하여 거기까지는 만들어내지 못하였다. 추후 몇몇 기능들을 추가하여 좀 더 완벽한 assembler simulator를 만들어보고 싶다. 그래도 지금까지 만든 결과물로도 object code, loader에 대해 많은 이해를 할 수 있었던 프로젝트였기 때문에 이해를 돕는다는 측면에서는 만족스러운 프로젝트였다.

첨부 프로그램 소스파일

1) SicLoader.java

```
package SP20_simulator;

import java.io.*;

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

	public SicLoader(ResourceManager resourceManager) {
		// 필요하다면 초기화
		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){
		//pass1
		try {
			// 입력 스트림 생성
			FileReader fileReader = new FileReader(objectCode);
			// 입력 버퍼 생성
			BufferedReader bufReader = new BufferedReader(fileReader);
			String line = "";

			int currentSection = 0;
			int loc = 0;
```

```
                while((line = bufReader.readLine()) != null){ //파일을 끝까지 읽기 전까지 한줄씩 읽음
                    if (line.length() == 0) {//읽은 문자열이 빈 라인이라면 다음 라인으로 넘기기
                        continue;
                    }

                    //라인의 첫번째 문자로 그 라인을 판단
                    switch(line.charAt(0)) {

                    //Header Record
                    //프로그램 이름, 시작주소, 프로그램 길이  저장
                    case 'H':

rMgr.setProgName(line.substring(1,7), currentSection);
                        loc = Integer.parseInt(line.substring(7, 13),16);
                        rMgr.setProgStart(loc, currentSection);

rMgr.setProgLength(Integer.parseInt(line.substring(13,19),16), currentSection);
                        if(currentSection == 0) {

rMgr.symtabList.putSymbol(line.substring(1,7).trim(), loc);
                        }
                        else {

rMgr.symtabList.putSymbol(line.substring(1,7).trim(), +
rMgr.getProgStart(currentSection));
                        }
                        break;

                    //Define Record
                    //정의된 심볼들을 symbol table에 저장
                    case 'D':
                        for(int i = 0; i < ((line.length() - 1) / 12); i++){

rMgr.symtabList.putSymbol(line.substring(12*i + 1, 12*i + 7).trim(), Integer.parseInt(line.substring(12*i + 7, 12*i + 13),16));
                        }
                        break;

                    //Text Record
                    case 'T':
                        int currentAddr = Integer.parseInt(line.substring(1, 7), 16) + rMgr.getProgStart(currentSection);
                        int lineLength = Integer.parseInt(line.substring(7, 9), 16);
                        char[] pack = new char[lineLength];
                        for(int i = 0; i < lineLength; i++){
```

```
                                        pack[i] = (char) (0 |
Integer.parseInt(line.substring(2*i + 9, 2*i + 11),16));
                                    }
                                    rMgr.setMemory(currentAddr,
pack, lineLength);
                                    break;

                                //Modification Record
                                //pass2에서 수정할 정보 저장
                                case 'M':
                                    int modLocation =
Integer.parseInt(line.substring(1, 7), 16) +
rMgr.getProgStart(currentSection);
                                    int modSize =
Integer.parseInt(line.substring(7, 9), 16);
                                    String modSymbol =
line.substring(9, line.length());

rMgr.symtabList.modSymbol(modSymbol, modLocation, modSize);
                                    break;

                                //End Record
                                case 'E':
                                    currentSection++;
                                    break;
                                }
                        }
                        bufReader.close(); // 입력버퍼를 닫음
                    }catch(FileNotFoundException e) {
                        System.out.println("not found");
                    }catch(IOException e) {
                        System.out.println(e);
                    }

                    //pass2
                    //M record의 정보를 이용하여 메모리에 저장된 값 수정
                    for(int i = 0; i < rMgr.symtabList.modList.size(); i++) {
                        String tmp = rMgr.symtabList.modList.get(i);
                        int size = rMgr.symtabList.modSize.get(i);
                        char mode = tmp.charAt(0);
                        String symbol = tmp.substring(1);

                        String modifyAddr = new String();
                        char[] pack = new char[3];
                        if(size == 5) {
                            modifyAddr = String.format("%05X",
rMgr.symtabList.search(symbol));
                            char[] temp =
rMgr.getMemory(rMgr.symtabList.modLocationList.get(i), 1);
                            int temp2 = temp[0];
                            temp2 = temp2 >>> 4;
                            temp2 = temp2 << 4;
                            pack[0] = (char)(temp2 |
Integer.parseInt(modifyAddr.substring(0,1),16));

                            for(int j = 1; j < 3; j++){
```

```
                                        pack[j] = (char) (0 |
Integer.parseInt(modifyAddr.substring(2*j - 1, 2*j + 1),16));
                                }
                        }
                        else if(size == 6) {
                                modifyAddr = String.format("%06X",
rMgr.symtabList.search(symbol));
                                if(mode == '-') {
                                        char[] temp =
rMgr.getMemory(rMgr.symtabList.modLocationList.get(i), 3);
                                        int tmpChar = temp[0];
                                        String tmp2 =
String.format("%02X", tmpChar);
                                        tmpChar = temp[1];
                                        tmp2 += String.format("%02X",
tmpChar);
                                        tmpChar = temp[2];
                                        tmp2 += String.format("%02X",
tmpChar);

                                        tmpChar =
Integer.parseInt(tmp2,16);
                                        int tmpNew =
Integer.parseInt(modifyAddr,16);
                                        modifyAddr =
String.format("%06X", tmpChar - tmpNew);
                                }
                                        for(int j = 0; j < 3; j++){
                                                pack[j] = (char) (0 |
Integer.parseInt(modifyAddr.substring(2*j, 2*j + 2),16));
                                        }
                                }

rMgr.setMemory(rMgr.symtabList.modLocationList.get(i), pack, 3);
                        }

        };

}
```

2) ResourceManager.java

```
package SP20_simulator;

import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;



/**
 * ResourceManager는 컴퓨터의 가상 리소스들을 선언하고 관리하는 클래스이다.
```

```
 * 크게 네가지의 가상 자원 공간을 선언하고, 이를 관리할 수 있는 함수들을 제공한다.<br><br>
 *
 * 1) 입출력을 위한 외부 장치 또는 device<br>
 * 2) 프로그램 로드 및 실행을 위한 메모리 공간. 여기서는 64KB를 최대값으로 잡는다.<br>
 * 3) 연산을 수행하는데 사용하는 레지스터 공간.<br>
 * 4) SYMTAB 등 simulator의 실행 과정에서 사용되는 데이터들을 위한 변수들.
 * <br><br>
 * 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.<br>
         * 즉, 'F1'이라는 디바이스는 'F1'이라는 이름의 파일을 의미한다.<br>
         * 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.
         * <br><br>
         * 변형된 형태로 사용하는 것 역시 허용한다.<br>
         * 예를 들면 key값으로 String대신 Integer를 사용할 수 있다.
         * 파일 입출력을 위해 사용하는 stream 역시 자유로이 선택, 구현한다.
         * <br><br>
         * 이것도 복잡하면 알아서 구현해서 사용해도 괜찮습니다.
         */
        HashMap<String,Object> deviceManager = new HashMap<String,Object>();
        char[] memory = new char[65536]; // String으로 수정해서 사용하여도 무방함.
        int[] register = new int[10];
        double register_F;

        SymbolTable symtabList = new SymbolTable();
        // 이외에도 필요한 변수 선언해서 사용할 것.
        List<String> progNameList = new ArrayList<>(); //프로그램 이름 저장
        List<Integer> progLengthList = new ArrayList<>(); //프로그램 길이 저장
        List<Integer> progStartList = new ArrayList<>(); //프로그램 시작주소 저장

        int currentSection = 0; //현재 섹션
        int readCheck = 0;

        /**
         * 메모리, 레지스터등 가상 리소스들을 초기화한다.
         */
        public void initializeResource(){
                for(int i = 0; i < memory.length; i++) {
```

```
                memory[i] = '0';
            }
            for(int i = 0; i < register.length; i++) {
                register[i] = 0;
            }
            register_F = 0;
        }

        /**
         * deviceManager가 관리하고 있는 파일 입출력 stream들을 전부 종료시키는 역할.
         * 프로그램을 종료하거나 연결을 끊을 때 호출한다.
         */
        public void closeDevice() {
            Iterator<String> keys = deviceManager.keySet().iterator();
            while(keys.hasNext()) {
                String key = keys.next();
                Object stream = deviceManager.get(key);

                if(stream instanceof FileReader) {
                    try {
                        ((FileReader)stream).close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
                else if(stream instanceof FileWriter) {
                    try {
                        ((FileWriter)stream).close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }

        /**
         * 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) {
            try {
                File file = new File(devName);
                if(devName.equals("F1")) {
                    FileReader fileReader = new FileReader(file);
                    deviceManager.put(devName, fileReader);
                    setRegister(9,1);
                }
                else if(devName.equals("05")) {
                    FileWriter fileWriter = new FileWriter(file, true); //append 모드로 flie write
                    deviceManager.put(devName, fileWriter);
```

```
                        setRegister(9,1);
                }
            }catch(FileNotFoundException e) {
                setRegister(9,0);
            }catch(IOException e) {
                e.printStackTrace();
            }

        }

        /**
         * 디바이스로부터 원하는 개수만큼의 글자를 읽어들인다. RD명령
어를 사용했을 때 호출되는 함수.
         * @param devName 디바이스의 이름
         * @param num 가져오는 글자의 개수
         * @return 가져온 데이터
         */
        public char[] readDevice(String devName, int num){
            FileReader fileReader =
(FileReader)deviceManager.get(devName);
            char[] cs = new char[num];
            int index = 0;
            int charTmp = 0;
            try {
                while(index <= readCheck) {
                    charTmp = fileReader.read();
                    index++;
                }
                if(charTmp == -1) {
                    cs[0] = 0;
                }
                else {
                    cs[0] = (char)charTmp;
                }
                readCheck++;

            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return cs;

        }

        /**
         * 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){
            FileWriter fileWriter =
(FileWriter)deviceManager.get(devName);
            try {
                for(int i = 0; i < num; i++) {
```

```
                        fileWriter.write(data[i]);
                        fileWriter.flush();
                }
        } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
}

/**
 * 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[] data = new char[num];

        for(int i = 0; i < num; i++) {
                data[i] = memory[location + i];
        }

        return data;

}

/**
 * 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];

}

/**
 * 주어진 컨트롤섹션의 프로그램 이름을 가져오는 함수
 * @param currentSection 프로그램 이름을 알고싶은 컨트롤 섹션
 */
public String getProgName(int currentSection){
        return progNameList.get(currentSection);
}
```

```
/**
 * 주어진 컨트롤섹션의 프로그램 길이를 가져오는 함수
 * @param currentSection 프로그램 길이를 알고싶은 컨트롤 섹션
 */
public int getProgLength(int currentSection){
	return progLengthList.get(currentSection);
}

/**
 * 주어진 컨트롤섹션의 프로그램 시작주소를 가져오는 함수
 * @param currentSection 프로그램 길이를 알고싶은 컨트롤 섹션
 */
public int getProgStart(int currentSection){
	return progStartList.get(currentSection);
}

/**
 * 현재 컨트롤섹션을 가져오는 함수
 */
public int getCurrentSection(){
	return currentSection;
}

/**
 * 프로그램의 전체 길이를 가져오는 함수
 */
public int getProgTotalLen() {
	int len = 0;
	for(int i = 0; i< progLengthList.size(); i++) {
		len += progLengthList.get(i);
	}
	return len;
}


/**
 * 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;
}

/**
 * 프로그램 이름을 저장하는 함수
 * @param progName 프로그램 이름
 * @param currentSection 프로그램 이름에 해당하는 컨트롤 섹션
 */
public void setProgName(String progName, int currentSection){
	progNameList.add(currentSection, progName);
}

/**
```

```
	 * 프로그램 길이를 저장하는 함수
	 * @param progLength 프로그램 길이
	 * @param currentSection 프로그램 길이에 해당하는 컨트롤 섹션
	 */
	public void setProgLength(int progLength, int currentSection){
		progLengthList.add(currentSection, progLength);
	}

	/**
	 * 프로그램 시작주소를 저장하는 함수
	 * @param progStart 프로그램 시작주소
	 * @param currentSection 프로그램 길이에 해당하는 컨트롤 섹션
	 */
	public void setProgStart(int progStart, int currentSection){
		int address = progStart;

		if(currentSection != 0) { //첫번째 섹션이 아니라면 이전 섹션의 주소를 더해줌
			address += progStartList.get(currentSection - 1) + progLengthList.get(currentSection - 1);
		}
		progStartList.add(currentSection, address);
	}

	/**
	 * 현재 컨트롤섹션을 저장하는 함수
	 * @param currentSection 저장할 컨트롤섹션
	 */
	public void setCurrentSection(int currentSection){
		this.currentSection = currentSection;
	}

	/**
	 * 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. int값을 char[]형태로 변경한다.
	 * @param data
	 * @return
	 */
	public char[] intToChar(int data){
		String tmp = "";
		char[] charData = new char[3];

		tmp = String.format("%06X", data);

		for(int i = 0; i < 3; i++) {
			charData[i] = (char)(Integer.parseInt(tmp.substring(2*i,2*i+2),16));
		}

		return charData;
	}

	/**
	 * 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. char[] 값을 int형태로 변경한다.
	 * @param data
```

```
	 * @return
	 */
	public int byteToInt(char[] data){
		int intData = 0;

		String tmp = new String();
		for(int i = 0; i < data.length; i++){
			tmp += String.format("%02X", (int)data[i]);
		}
		intData = Integer.parseInt(tmp,16);
		return intData;
	}
}
```

3) SymbolTable.java

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

/**
 * symbol과 관련된 데이터와 연산을 소유한다.
 */
public class SymbolTable {
	ArrayList<String> symbolList;
	ArrayList<Integer> addressList;
	// 기타 literal, external 선언 및 처리방법을 구현한다.
	ArrayList<String> modList;
	ArrayList<Integer> modLocationList;
	ArrayList<Integer> modSize;

	public SymbolTable() {
		symbolList = new ArrayList<>();
		addressList = new ArrayList<>();
		modList = new ArrayList<>();
		modLocationList = new ArrayList<>();
		modSize = new ArrayList<>();
	}

	/**
	 * 새로운 Symbol을 table에 추가한다.
	 * @param symbol : 새로 추가되는 symbol의 label
	 * @param address : 해당 symbol이 가지는 주소값
	 * <br><br>
	 * 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) {
		if(symbolList.contains(symbol) == false) {
			symbolList.add(symbol);
			addressList.add(address);
		}
	}
```

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

	/**
	 * modification record에 작성할 symbol들을 table에 추가한다.
	 * @param symbol : modify될 symbol의 label
	 * @param location : 해당 symbol의 location
	 * @param size : modify될 바이트의 크기
	 */
	public void modSymbol(String symbol, int location, int size) {
		modList.add(symbol);
		modLocationList.add(location);
		modSize.add(size);
	}

	/**
	 * 인자로 전달된 symbol이 어떤 주소를 지칭하는지 알려준다.
	 * @param symbol : 검색을 원하는 symbol의 label
	 * @return symbol이 가지고 있는 주소값. 해당 symbol이 없을 경
우 -1 리턴
	 */
	public int search(String symbol) {
		int address = 0;

		if(symbolList.contains(symbol)) {
			for (int i = 0; i < symbolList.size(); i++) {
				if (symbol.equals(symbolList.get(i))){
					address = addressList.get(i);
					break;
				}
			}
		}
		else {
			address = -1;
		}

		return address;
	}

	/**
	 * index에 해당하는 symbol을 리턴한다.
	 * @param index : 리턴할 symbol의 인덱스
	 * @return : symbol
	 *
	 */
	public String getSymbol(int index) {
		return symbolList.get(index);
```

```
	}

	/**
	 * index를 이용하여 symbol을 찾은 후 그 symbol에 해당하는 address을 리턴한다.
	 * @param index : 리턴할 address의 인덱스
	 * @return : address
	 *
	 */
	public int getaddress(int index) {
		return addressList.get(index);
	}

	/**
	 * symboltable의 크기을 리턴한다.
	 * @return : symboltable의 크기
	 *
	 */
	public int getSize() {
		return symbolList.size();
	}


}
```

4) InstLuncher.java

```
package SP20_simulator;

// instruction에 따라 동작을 수행하는 메소드를 정의하는 클래스

public class InstLuncher {
    ResourceManager rMgr;
    int targetAddr = 0;

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

    // instruction 별로 동작을 수행하는 메소드를 정의
    // ex) public void add(){...}

    /**
     * L 레지스터의 값을 메모리에 저장하는 메소드이다.
     * m..m+2 <- (L)
     * @param start instruction이 위치한 시작주소
     * @param format instruction의 형식
     * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
     */
    public void stl(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
```

```
		rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
		if(format == 3) {
			int tmp = instruction[1] & 15;
			tmp = tmp << 8;
			tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기

			int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
			char[] data = rMgr.intToChar(rMgr.getRegister(SicSimulator.L_REGISTER));
			rMgr.setMemory(address, data, 3);
		}

	}
	/**
	* PC에 저장되어 있는 값을 L레지스터에 저장후 메모리의 값을 PC레지스터에 저장하는 메소드이다.
	* L <- (PC); PC <- (m)
	* @param start instruction이 위치한 시작주소
	* @param format instruction의 형식
	* @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
	*/
	public void jsub(int start, int format, int addressing) {
		char [] instruction = rMgr.getMemory(start, format);
		targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
		rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);

		if(format == 4) {
			rMgr.setRegister(SicSimulator.L_REGISTER, rMgr.getRegister(SicSimulator.PC_REGISTER));
			int tmp = instruction[1] & 15;
			tmp = tmp << 8;
			tmp = tmp | (int)instruction[2];
			tmp = tmp << 8;
			tmp = tmp | (int)instruction[3];
			rMgr.setRegister(SicSimulator.PC_REGISTER, tmp);
		}

	}
	/**
	* A <- (m..m+2)
	* 메모리의 값을 읽어 A레지스터에 저장하는 메소드이다.
	* @param start instruction이 위치한 시작주소
	* @param format instruction의 형식
	* @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
	*/
	public void lda(int start, int format, int addressing) {
		char [] instruction = rMgr.getMemory(start, format);
		targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
		rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
		if(format == 3) {
```

```
                if(addressing == 1) { // immediate addressing일 때
                        int tmp = instruction[1];
                        tmp = tmp << 8;
                        tmp = tmp | (int)instruction[2];
                        rMgr.setRegister(SicSimulator.A_REGISTER,
tmp);

                }
                else if(addressing == 3) {
                        int tmp = instruction[1] & 15;
                        tmp = tmp << 8;
                        tmp = tmp | (int)instruction[2]; //뒤의 3자리 주
소값 찾기

                        int address = tmp +
rMgr.getRegister(SicSimulator.PC_REGISTER);
                        char[] data =  rMgr.getMemory(address, 3);
                        rMgr.setRegister(SicSimulator.A_REGISTER,
rMgr.byteToInt(data));
                }
        }
    }

    /**
    * (A):(m..m+2)
    * A레지스터의 값과 메모리에 저장되어 있는 값을 비교하는 메소드이
다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate,
2 : indirect, 3: direct)
    */
    public void comp(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) +
format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 3) {
                if(addressing == 1) { // immediate addressing일 때
                        int tmp = instruction[1];
                        tmp = tmp << 8;
                        tmp = tmp | (int)instruction[2];

                        int dif =
rMgr.getRegister(SicSimulator.A_REGISTER) - tmp;
                        if(dif == 0) {
                        rMgr.setRegister(SicSimulator.SW_REGISTER, 0);
                }
                else if(dif > 0) {
                        rMgr.setRegister(SicSimulator.SW_REGISTER, 1);
                        }
                else {
                        rMgr.setRegister(SicSimulator.SW_REGISTER,
-1);
                }
                }
```

```
        }
    }

    /**
    *  PC <-m if CC set to =(0)
    * sw레지스터가 0이라면 메모리에 저장된 값으로 이동하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
    */
    public void jeq(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 3) {
            if(rMgr.getRegister(SicSimulator.SW_REGISTER) == 0) {
                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기
                if((instruction[1] & 15) == 15) {//F로 시작하는 주소의 경우 앞부분을 다 F로 채워넣기..
                    tmp = tmp |((0xFFFFF)<<12);
                }
                int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
                rMgr.setRegister(SicSimulator.PC_REGISTER, address);
            }
        }
    }

    /***
    * PC <- m
    * 메모리에 저장된 값을 pc레지스터에 저장하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
    */
    public void j(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 3) {
            int tmp = instruction[1] & 15;
            tmp = tmp << 8;
            tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기
            if((instruction[1] & 15) == 15) {//F로 시작하는 주소의 경우 앞부분을 다 F로 채워넣기..
                tmp = tmp |((0xFFFFF)<<12);
```

```
                }
                int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
                if(addressing == 2){ //indirect addressing이라면
                        address = rMgr.byteToInt(rMgr.getMemory(address, 3));
                        }
                rMgr.setRegister(SicSimulator.PC_REGISTER, address);
        }
    }

    /**
    * m..m+2 <- (A)
    * A레지스터의 값을 메모리에 저장하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
    */
    public void sta(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 3) {
                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기

                int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
                char[] data = rMgr.intToChar(rMgr.getRegister(SicSimulator.A_REGISTER));
                rMgr.setMemory(address, data, 3);
        }
    }
    /**
    * r1 <- 0
    * 레지스터의 값을 0으로 초기화하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    */
    public void clear(int start) {
        char [] instruction = rMgr.getMemory(start, 2);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + 2;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        int regNum = instruction[1];
        regNum = regNum >>> 4;
        rMgr.setRegister(regNum, 0);
    }

    /**
    * T <- (m..m+2)
    * 메모리의 값을 읽어와 T레지스터에 저장하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
```

```
    * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
    */
    public void ldt(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 4) {
                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2];
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[3];
                char[] data = rMgr.getMemory(tmp, 3);
                rMgr.setRegister(SicSimulator.T_REGISTER, rMgr.byteToInt(data));

        }
        else {
                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기

                int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
                char[] data =  rMgr.getMemory(address, 3);
                rMgr.setRegister(SicSimulator.T_REGISTER, rMgr.byteToInt(data));
        }
    }

    /**
    //Test device specified by (m)
    //장치(파일)를 확인하는 메소드로 sw레지스터가 0이면 준비안된 것. 0이 아니라면 준비된 것.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
    * @return 확인하고 있는 장치의 이름
    */
    public String td(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 3) {
                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기

                int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
```

```
                char[] deviceAddr = rMgr.getMemory(address, 1);
                String deviceName = String.format("%X%X",
deviceAddr[0]>>4,deviceAddr[0]&15);
                rMgr.testDevice(deviceName);
                return deviceName;
        }
        return null;
    }

    /**
    * A[rightmost byte] <- data from device specified by (m)
    * 기기(파일)에서 한바이트 읽어와 A레지스터의 제일 오른쪽 1바이트에
저장하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate,
2 : indirect, 3: direct)
    **/
    public void rd(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) +
format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 3) {
                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾
기
                int address = tmp +
rMgr.getRegister(SicSimulator.PC_REGISTER);
                char[] deviceAddr = rMgr.getMemory(address, 1);
                String deviceName = String.format("%X%X",
deviceAddr[0]>>4,deviceAddr[0]&15);
                char[] data = rMgr.readDevice(deviceName,
rMgr.getRegister(SicSimulator.T_REGISTER));
                char[] tmp2 = new char[1];
                tmp2[0] = data[0];
                rMgr.setRegister(SicSimulator.A_REGISTER,
rMgr.byteToInt(tmp2));
        }
    }

    /**
    * (r1) : (r2)
    * r1레지스터에 저장된 값과 r2 레지스터에 저장된 값을 비교하는 메소
드이다.
    * @param start instruction이 위치한 시작주소
    **/
    public void compr(int start) {
        char [] instruction = rMgr.getMemory(start, 2);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + 2;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);

        int regNum1 = instruction[1] >>> 4;
        int regNum2 = instruction[1] & 15;
```

```
        int dif = rMgr.getRegister(regNum1) - rMgr.getRegister(regNum2);
        if(dif == 0) {
                rMgr.setRegister(SicSimulator.SW_REGISTER, 0);
        }
        else if(dif > 0) {
                rMgr.setRegister(SicSimulator.SW_REGISTER, 1);
                }
        else {
                rMgr.setRegister(SicSimulator.SW_REGISTER, -1);
        }
    }

    /**
    * m <- (A)[rightmost byte]
    * A레지스터의 제일 오른쪽 한바이트를 메모리에 저장하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
    **/
    public void stch(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 4) {
                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2];
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[3];

                int tmpData = rMgr.getRegister(SicSimulator.A_REGISTER) & 255;
                char[] data = rMgr.intToChar(tmpData);
                char[] tmp2 = new char[1];
                tmp2[0]= data[2];
                rMgr.setMemory(tmp + rMgr.getRegister(SicSimulator.X_REGISTER), tmp2, 1);
        }
    }

    /**
    * X <- (X) + 1; (X):(r1)
    * X레지스터의 값을 1 증가시키고 r1레지스터와 비교하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    **/
    public void tixr(int start) {
        char [] instruction = rMgr.getMemory(start, 2);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + 2;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);

        rMgr.setRegister(SicSimulator.X_REGISTER, rMgr.getRegister(SicSimulator.X_REGISTER)+1);
```

```
        int regNum = instruction[1];
        regNum = regNum >>> 4;
        int dif = rMgr.getRegister(SicSimulator.X_REGISTER) - rMgr.getRegister(regNum);
        if(dif == 0) {
                rMgr.setRegister(SicSimulator.SW_REGISTER, 0);
        }
        else if(dif > 0) {
                rMgr.setRegister(SicSimulator.SW_REGISTER, 1);
                }
        else {
                rMgr.setRegister(SicSimulator.SW_REGISTER, -1);
        }

    }

    /**
    * PC <- m if CC set to <(-1)
    * SW 레지스터의 값이 -1이라면 메모리에 저장된 값을 PC레지스터에 저장하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
    **/
    public void jlt(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 3) {
                if(rMgr.getRegister(SicSimulator.SW_REGISTER) == -1) {
                        int tmp = instruction[1] & 15;
                        tmp = tmp << 8;
                        tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기
                        if((instruction[1] & 15) == 15) {//F로 시작하는 주소의 경우 앞부분을 다 F로 채워넣기..
                                tmp = tmp |((0xFFFFF)<<12);
                        }
                        int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
                        rMgr.setRegister(SicSimulator.PC_REGISTER, address);
                }
        }
    }

    /**
    //m .. m+2 <- (X)
    //x레지스터의 값을 메모리에 저장하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
    **/
```

```
    public void stx(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) +
format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 4) {
                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2];
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[3];

                char[] data =
rMgr.intToChar(rMgr.getRegister(SicSimulator.X_REGISTER));
                rMgr.setMemory(tmp, data, 3);


        }

    }
    /**
    //PC <- (L)
    //L레지스터에 저장된 값을 pc레지스터에 저장하는 메소드이다.
    **/
    public void rsub() {
        rMgr.setRegister(SicSimulator.PC_REGISTER,
rMgr.getRegister(SicSimulator.L_REGISTER));
    }
    /**
    * (A)[rightmost byte] <- (m)
    * 메모리의 값을 A의 제일 오른쪽 한 바이트에 저장하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate,
2 : indirect, 3: direct)
    **/
    public void ldch(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) +
format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 4) {
                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2];
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[3];

                char[] data = rMgr.getMemory(tmp +
rMgr.getRegister(SicSimulator.X_REGISTER), 1);
                rMgr.setRegister(SicSimulator.A_REGISTER,
rMgr.byteToInt(data));
        }

    }
```

```
    /**
    * Device specified by (m) <- (A)[rightmost byte]
    * A레지스터에 저장된 값 중 제일 오른쪽 한바이트를 장치(파일)에 저장하는 메소드이다.
    * 메모리의 값을 A의 제일 오른쪽 한 바이트에 저장하는 메소드이다.
    * @param start instruction이 위치한 시작주소
    * @param format instruction의 형식
    * @param addressing instruction의 addressing 방식 (1 : immediate, 2 : indirect, 3: direct)
    **/
    public void wd(int start, int format, int addressing) {
        char [] instruction = rMgr.getMemory(start, format);
        targetAddr = rMgr.getRegister(SicSimulator.PC_REGISTER) + format;
        rMgr.setRegister(SicSimulator.PC_REGISTER, targetAddr);
        if(format == 3) {

                int tmp = instruction[1] & 15;
                tmp = tmp << 8;
                tmp = tmp | (int)instruction[2]; //뒤의 3자리 주소값 찾기

                int address = tmp + rMgr.getRegister(SicSimulator.PC_REGISTER);
                char[] deviceAddr = rMgr.getMemory(address, 1);
                String deviceName = String.format("%X%X", deviceAddr[0]>>4,deviceAddr[0]&15);
                int regData = rMgr.getRegister(SicSimulator.A_REGISTER);
                char[] data = rMgr.intToChar(regData);
                char[] tmp2 = new char[1];
                tmp2[0] = data[2];
                rMgr.writeDevice(deviceName,tmp2,1);
                }
    }

}
```

5) SicSimulator.java

```
package SP20_simulator;

import java.io.File;
import java.util.*;

/**
 * 시뮬레이터로서의 작업을 담당한다. VisualSimulator에서 사용자의 요청을 받으면 이에 따라
 * ResourceManager에 접근하여 작업을 수행한다.
 *
 * 작성중의 유의사항 : <br>
 *  1) 새로운 클래스, 새로운 변수, 새로운 함수 선언은 얼마든지 허용됨. 단, 기존의 변수와 함수들을 삭제하거나 완전히 대체하는 것은 지양할 것.<br>
 *  2) 필요에 따라 예외처리, 인터페이스 또는 상속 사용 또한 허용
```

```
된.<br>
 *  3) 모든 void 타입의 리턴값은 유저의 필요에 따라 다른 리턴 타입으로 변경 가능.<br>
 *  4) 파일, 또는 콘솔창에 한글을 출력시키지 말 것. (채점상의 이유. 주석에 포함된 한글은 상관 없음)<br>
 *
 * <br><br>
 *  + 제공하는 프로그램 구조의 개선방법을 제안하고 싶은 분들은 보고서의 결론 뒷부분에 첨부 바랍니다. 내용에 따라 가산점이 있을 수 있습니다.
 */
public class SicSimulator {
	ResourceManager rMgr;
	InstLuncher inst;

	public static final int A_REGISTER = 0;
	public static final int X_REGISTER = 1;
	public static final int L_REGISTER = 2;
	public static final int B_REGISTER = 3;
	public static final int S_REGISTER = 4;
	public static final int T_REGISTER = 5;
	public static final int F_REGISTER = 6;
	public static final int PC_REGISTER = 8;
	public static final int SW_REGISTER = 9;

	private List<String> instructionsList = new ArrayList<>();
	private List<String> logList = new ArrayList<>();

	String currentDev = "";
	int addr = 0;
	boolean pEnd = false;

	public SicSimulator(ResourceManager resourceManager) {
		// 필요하다면 초기화 과정 추가
		this.rMgr = resourceManager;
		inst = new InstLuncher(rMgr);
	}

	/**
	 * 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*/
		rMgr.initializeResource();
	}

	/**
	 * 1개의 instruction이 수행된 모습을 보인다.
	 */
	public void oneStep() {
		addr = rMgr.getRegister(PC_REGISTER);
		if(pEnd) {//모든 instruction이 수행을 마쳤음을 의미
			return;
		}
```

```
char [] bytes = rMgr.getMemory(addr,2);
int temp = bytes[0];
int opcode = temp;
int addressing = 0;
boolean form_4 = false;

// indirect addressing인지 immediate addressing인지 판
단

if((temp & 3) == 3) {
        opcode -= 3;
        addressing = 3;
}
else if((temp & 2) == 2){
        opcode -= 2;
        addressing = 2;
}
else if((temp & 1) == 1){
        opcode -= 1;
        addressing = 1;
}

//두번째 바이트를 이용하여 확장된 4형식 명령어인지 체
크

temp = (bytes[1] >>> 4);
if((temp & 1) == 1) {
        form_4 = true;
}
String code = new String();
char [] tmpInst = new char[2];

switch(opcode) {
case 0x14://3,4
        addLog("STL");
        if(form_4) {
                logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                code = makingCode(instruction, 4);
                addInstruction(code);
                inst.stl(rMgr.getRegister(PC_REGISTER),
4, addressing);
        }
        else {
                char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                code = makingCode(instruction, 3);
                addInstruction(code);
                inst.stl(rMgr.getRegister(PC_REGISTER),
3, addressing);
        }
        break;

case 0x48: //3,4
        addLog("JSUB");
```

```
                        if(form_4) {
                                logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                                char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                                code = makingCode(instruction, 4);
                                addInstruction(code);

inst.jsub(rMgr.getRegister(PC_REGISTER), 4, addressing);
                        }
                        else {
                                char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                                code = makingCode(instruction, 3);
                                addInstruction(code);

inst.jsub(rMgr.getRegister(PC_REGISTER), 3, addressing);
                        }

                        break;

                case 0x00://3,4
                        addLog("LDA");
                        if(form_4) {
                                logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                                char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                                code = makingCode(instruction, 4);
                                addInstruction(code);
                                inst.lda(rMgr.getRegister(PC_REGISTER),
4, addressing);
                        }
                        else {
                                char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                                code = makingCode(instruction, 3);
                                addInstruction(code);
                                inst.lda(rMgr.getRegister(PC_REGISTER),
3, addressing);
                        }
                        break;

                case 0x28://3,4
                        addLog("COMP");
                        if(form_4) {
                                logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                                char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                                code = makingCode(instruction, 4);
                                addInstruction(code);

inst.comp(rMgr.getRegister(PC_REGISTER), 4, addressing);
                        }
                        else {
```

```
                        char [] instruction = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                        code = makingCode(instruction, 3);
                        addInstruction(code);

inst.comp(rMgr.getRegister(PC_REGISTER), 3, addressing);
                }
                break;

        case 0x30://3,4
                addLog("JEQ");
                if(form_4) {
                        logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));
                        char [] instruction = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                        code = makingCode(instruction, 4);
                        addInstruction(code);
                        inst.jeq(rMgr.getRegister(PC_REGISTER), 4, addressing);
                }
                else {
                        char [] instruction = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                        code = makingCode(instruction, 3);
                        addInstruction(code);
                        inst.jeq(rMgr.getRegister(PC_REGISTER), 3, addressing);
                }
                break;

        case 0x3C://3,4
                addLog("J");
                if(form_4) {
                        logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));
                        char [] instruction = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                        code = makingCode(instruction, 4);
                        addInstruction(code);
                        inst.j(rMgr.getRegister(PC_REGISTER), 4, addressing);
                }
                else {
                        char [] instruction = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                        code = makingCode(instruction, 3);
                        addInstruction(code);
                        inst.j(rMgr.getRegister(PC_REGISTER), 3, addressing);
                }
                if(rMgr.getRegister(PC_REGISTER) == 0){
                        pEnd = true;
                }
                break;
```

```
case 0x0C://3,4
	addLog("STA");
	if(form_4) {
		logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));
		char [] instruction = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
		code = makingCode(instruction, 4);
		addInstruction(code);
		inst.sta(rMgr.getRegister(PC_REGISTER), 4, addressing);
	}
	else {
		char [] instruction = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
		code = makingCode(instruction, 3);
		addInstruction(code);
		inst.sta(rMgr.getRegister(PC_REGISTER), 3, addressing);
	}
	break;

case 0xB4://2
	addLog("CLEAR");
	tmpInst = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 2);
	code = makingCode(tmpInst, 2);
	addInstruction(code);
	inst.clear(rMgr.getRegister(PC_REGISTER));
	break;

case 0x74://3,4
	addLog("LDT");
	if(form_4) {
		logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));
		char [] instruction = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
		code = makingCode(instruction, 4);
		addInstruction(code);
		inst.ldt(rMgr.getRegister(PC_REGISTER), 4, addressing);
	}
	else {
		char [] instruction = rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
		code = makingCode(instruction, 3);
		addInstruction(code);
		inst.ldt(rMgr.getRegister(PC_REGISTER), 3, addressing);
	}
	break;

case 0xE0://3,4
	addLog("TD");
	if(form_4) {
```

```
                        logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                        code = makingCode(instruction, 4);
                        addInstruction(code);
                        currentDev =
inst.td(rMgr.getRegister(PC_REGISTER), 4, addressing);
                    }
                    else {
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                        code = makingCode(instruction, 3);
                        addInstruction(code);
                        currentDev =
inst.td(rMgr.getRegister(PC_REGISTER), 3, addressing);
                    }
                    break;

                case 0xD8://3,4
                    addLog("RD");
                    if(form_4) {
                        logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                        code = makingCode(instruction, 4);
                        addInstruction(code);
                        inst.rd(rMgr.getRegister(PC_REGISTER),
4, addressing);
                    }
                    else {
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                        code = makingCode(instruction, 3);
                        addInstruction(code);
                        inst.rd(rMgr.getRegister(PC_REGISTER),
3, addressing);
                    }
                    break;

                case 0xA0://2
                    addLog("COMPR");
                    tmpInst =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 2);
                    code = makingCode(tmpInst, 2);
                    addInstruction(code);
                    inst.compr(rMgr.getRegister(PC_REGISTER));
                    break;

                case 0x54://3,4
                    addLog("STCH");
                    if(form_4) {
                        logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                        char [] instruction =
```

```
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                        code = makingCode(instruction, 4);
                        addInstruction(code);

inst.stch(rMgr.getRegister(PC_REGISTER), 4, addressing);
                }
                else {
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                        code = makingCode(instruction, 3);
                        addInstruction(code);

inst.stch(rMgr.getRegister(PC_REGISTER), 3, addressing);
                }
                break;

        case 0xB8://2
                addLog("TIXR");
                tmpInst =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 2);
                code = makingCode(tmpInst, 2);
                addInstruction(code);
                inst.tixr(rMgr.getRegister(PC_REGISTER));
                break;

        case 0x38://3,4
                addLog("JLT");
                if(form_4) {
                        logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                        code = makingCode(instruction, 4);
                        addInstruction(code);
                        inst.jlt(rMgr.getRegister(PC_REGISTER),
4, addressing);
                }
                else {
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                        code = makingCode(instruction, 3);
                        addInstruction(code);
                        inst.jlt(rMgr.getRegister(PC_REGISTER),
3, addressing);
                }
                break;

        case 0x10://3,4
                addLog("STX");
                if(form_4) {
                        logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                        code = makingCode(instruction, 4);
                        addInstruction(code);
```

```
                        inst.stx(rMgr.getRegister(PC_REGISTER),
4, addressing);
                    }
                    else {
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                        code = makingCode(instruction, 3);
                        addInstruction(code);
                        inst.stx(rMgr.getRegister(PC_REGISTER),
3, addressing);
                    }
                    break;

                case 0x4C://3,4
                    addLog("RSUB");
                    tmpInst =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                    code = makingCode(tmpInst, 3);
                    addInstruction(code);
                    inst.rsub();
                    currentDev = "";
                    break;

                case 0x50://3,4
                    addLog("LDCH");
                    if(form_4) {
                        logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                        code = makingCode(instruction, 4);
                        addInstruction(code);

inst.ldch(rMgr.getRegister(PC_REGISTER), 4, addressing);
                    }
                    else {
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                        code = makingCode(instruction, 3);
                        addInstruction(code);

inst.ldch(rMgr.getRegister(PC_REGISTER), 3, addressing);
                    }
                    break;

                case 0xDC://3,4
                    addLog("WD");
                    if(form_4) {
                        logList.set(logList.size()-1, "+" +
logList.get(logList.size()-1));
                        char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 4);
                        code = makingCode(instruction, 4);
                        addInstruction(code);
                        inst.wd(rMgr.getRegister(PC_REGISTER),
4, addressing);
```

```
                        }
                        else {
                                char [] instruction =
rMgr.getMemory(rMgr.getRegister(PC_REGISTER), 3);
                                code = makingCode(instruction, 3);
                                addInstruction(code);
                                inst.wd(rMgr.getRegister(PC_REGISTER),
3, addressing);
                        }
                        break;
                }
        }

        /**
         * 남은 모든 instruction이 수행된 모습을 보인다.
         */
        public void allStep() {
                while(!pEnd){
                        oneStep();
                }
        }

        /**
         * 각 단계를 수행할 때 마다 관련된 기록을 남기도록 한다.
         */
        public void addLog(String log) {
                logList.add(log);
        }
```

다.

```
        /**
         * 각 단계를 수행할 때 마다 수행한 명령어 code를 남기도록 한
         */
        public void addInstruction(String instruction) {
                instructionsList.add(instruction);
        }

        /**
         * 지금까지 수행한 로그기록들의 리스트를 반환하는 메소드이다.
         * @return 지금까지 수행한 로그 기록이 저장된 리스트
         */
        public List<String> getLogList(){
                return logList;
        }

        /**
         * 지금까지 수행한 code기록들의 리스트를 반환하는 메소드이다.
         * @return 지금까지 수행한 code 기록이 저장된 리스트
         */
        public List<String> getInstList(){
                return instructionsList;
        }

        /**
         * 현재 사용중인 장치(파일)이름을 반환하는 메소드이다.
         * @return 현재 사용중인 장치(파일)이름
```

```
	 */
	public String getDevice() {
		return currentDev;
	}

	/**
	 * 현재 진행중인 명령어의 주소를 반환하는 메소드이다.
	 * @return 현재 진행중인 명령어의 주소
	 */
	public int getAddress() {
		return addr;
	}

	/**
	 * 현재 수행중인 code를 기록으로 남기기 위해 char[]를 String으로 바꿔주는 메소드이다.
	 * @param instruction String으로 바꾸려고 하는 code
	 * @param format instruction의 형식
	 * @return String으로 바꾼 code
	 */
	public String makingCode(char[] instruction, int format) {
		int tmp = instruction[0];
		String code = String.format("%02X", tmp);
		tmp = instruction[1];
		code += String.format("%02X", tmp);
		if(format == 4) {
			tmp = instruction[2];
			code += String.format("%02X", tmp);
			tmp = instruction[3];
			code += String.format("%02X", tmp);
		}
		else if(format == 3) {
			tmp = instruction[2];
			code += String.format("%02X", tmp);
		}
		return code;
	}
}
```

6) VisualSimulator.java

```
package SP20_simulator;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;
import javax.swing.border.TitledBorder;
import javax.swing.filechooser.FileNameExtensionFilter;

import java.io.File;

/**
 * VisualSimulator는 사용자와의 상호작용을 담당한다.<br>
```

```
 * 즉, 버튼 클릭등의 이벤트를 전달하고 그에 따른 결과값을 화면에 업데
이트 하는 역할을 수행한다.<br>
 * 실제적인 작업은 SicSimulator에서 수행하도록 구현한다.
 */
@SuppressWarnings("serial")
public class VisualSimulator extends JFrame {
	ResourceManager resourceManager;
	SicLoader sicLoader;
	SicSimulator sicSimulator;

	String filePath;

	JList<String> instList; //명령어 리스트
	JList<String> logList; //로그리스트
	JTextField txtProgramName = new JTextField(); //H 프로그램이름
	JTextField txtObjectProgram = new JTextField(); //H 프로그램 시
작주소
	JTextField txtLengthOfProgram = new JTextField(); //H 프로그램
길이
	JTextField txtInObjectProgram = new JTextField(); //E 프로그램
시작주소
	JTextField txtADec = new JTextField();
	JTextField txtAHex = new JTextField();
	JTextField txtXDec = new JTextField();
	JTextField txtXHex = new JTextField();
	JTextField txtLDec = new JTextField();
	JTextField txtLHex = new JTextField();
	JTextField txtBDec = new JTextField();
	JTextField txtBHex = new JTextField();
	JTextField txtSDec = new JTextField();
	JTextField txtSHex = new JTextField();
	JTextField txtTDec = new JTextField();
	JTextField txtTHex = new JTextField();
	JTextField txtF = new JTextField();
	JTextField txtPCDec = new JTextField();
	JTextField txtPCHex = new JTextField();
	JTextField txtSW = new JTextField();
	JTextField txtStart = new JTextField(); //start Address in Memory
	JTextField txtTarget = new JTextField(); //Target Address
	JTextField txtDevice = new JTextField(); //사용중인 장치

	JScrollPane scrollPaneInst;
	JScrollPane scrollPane;

	public VisualSimulator() { //프레임 만들기 및 초기화
		super("SIC/XE Simulator");

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

		setBounds(100 , 100 , 500 , 700);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setLayout(null); //절대 위치를 사용하기 위한 설정

		//파일오픈 부분
```

```
JLabel fileName = new JLabel("FileName:");
fileName.setBounds(10, 20, 100, 10);
add(fileName);
JTextField inputText = new JTextField();
inputText.setBounds(75,16,120,20);
add(inputText);

JButton btnOpen = new JButton("open"); // 파일 오픈버튼 생성
btnOpen.setBounds(200, 16, 70, 18);
add(btnOpen);

//H부분
JPanel panelH = new JPanel();
panelH.setBorder(new TitledBorder(null,"H (Header Record)"));
panelH.setBounds(10,45,230,125);
add(panelH);
panelH.setLayout(null);

JLabel lbProgramName = new JLabel("Program Name:");
lbProgramName.setBounds(16,22,100,20);
panelH.add(lbProgramName);

txtProgramName.setBounds(110,22,100,23);
txtProgramName.setEditable(false);
panelH.add(txtProgramName);

JLabel lbStartAddressOf = new JLabel("Start Address Of");
lbStartAddressOf.setBounds(16,46,100,20);
panelH.add(lbStartAddressOf);
JLabel lbObjectProgram = new JLabel("Object Program:");
lbObjectProgram.setBounds(33,63,100,20);
panelH.add(lbObjectProgram);

txtObjectProgram.setBounds(130,56,80,23);
txtObjectProgram.setEditable(false);
panelH.add(txtObjectProgram);

JLabel lbLengthOfProgram = new JLabel("Length of Program:");
lbLengthOfProgram.setBounds(16,87,120,20);
panelH.add(lbLengthOfProgram);

txtLengthOfProgram.setBounds(131,88,80,23);
txtLengthOfProgram.setEditable(false);
panelH.add(txtLengthOfProgram);

//E부분
JPanel panelE = new JPanel();
panelE.setBorder(new TitledBorder(null,"E (End Record)"));
panelE.setBounds(260,45,210,77);
add(panelE);
```

```
panelE.setLayout(null);

JLabel lbAddressOfFirstInstruction = new JLabel("Address of First Instruction");
lbAddressOfFirstInstruction.setBounds(16,22,180,20);
panelE.add(lbAddressOfFirstInstruction);

JLabel lbInObjectProgram = new JLabel("in Object Program:");
lbInObjectProgram.setBounds(20,40,120,20);
panelE.add(lbInObjectProgram);

txtInObjectProgram.setBounds(130,42,60,23);
txtInObjectProgram.setEditable(false);
panelE.add(txtInObjectProgram);

//Register부분
JPanel panelReg = new JPanel();
panelReg.setBorder(new TitledBorder(null,"Register"));
panelReg.setBounds(10,180,230,275);
add(panelReg);
panelReg.setLayout(null);

JLabel lbDec = new JLabel("Dec");
lbDec.setBounds(85,20,30,20);
panelReg.add(lbDec);
JLabel lbHex = new JLabel("Hex");
lbHex.setBounds(150,20,30,20);
panelReg.add(lbHex);
JLabel lbA = new JLabel("A(#0)");
lbA.setBounds(30,45,50,20);
panelReg.add(lbA);
JLabel lbX = new JLabel("X(#1)");
lbX.setBounds(30,70,50,20);
panelReg.add(lbX);
JLabel lbL = new JLabel("L(#2)");
lbL.setBounds(30,95,50,20);
panelReg.add(lbL);
JLabel lbB = new JLabel("B(#3)");
lbB.setBounds(30,120,40,20);
panelReg.add(lbB);
JLabel lbS = new JLabel("S(#4)");
lbS.setBounds(30,145,40,20);
panelReg.add(lbS);
JLabel lbT = new JLabel("T(#5)");
lbT.setBounds(30,170,40,20);
panelReg.add(lbT);
JLabel lbF = new JLabel("F(#6)");
lbF.setBounds(30,195,40,20);
panelReg.add(lbF);
JLabel lbPC = new JLabel("PC(#8)");
lbPC.setBounds(25,220,50,20);
panelReg.add(lbPC);
JLabel lbSW = new JLabel("SW(#9)");
lbSW.setBounds(24,245,50,20);
panelReg.add(lbSW);
```

```
txtADec.setBounds(80,45,60,18);
txtADec.setEditable(false);
panelReg.add(txtADec);

txtAHex.setBounds(145,45,60,18);
txtAHex.setEditable(false);
panelReg.add(txtAHex);

txtXDec.setBounds(80,70,60,18);
txtXDec.setEditable(false);
panelReg.add(txtXDec);

txtXHex.setBounds(145,70,60,18);
txtXHex.setEditable(false);
panelReg.add(txtXHex);

txtLDec.setBounds(80,95,60,18);
txtLDec.setEditable(false);
panelReg.add(txtLDec);

txtLHex.setBounds(145,95,60,18);
txtLHex.setEditable(false);
panelReg.add(txtLHex);

txtBDec.setBounds(80,120,60,18);
txtBDec.setEditable(false);
panelReg.add(txtBDec);

txtBHex.setBounds(145,120,60,18);
txtBHex.setEditable(false);
panelReg.add(txtBHex);

txtSDec.setBounds(80,145,60,18);
txtSDec.setEditable(false);
panelReg.add(txtSDec);

txtSHex.setBounds(145,145,60,18);
txtSHex.setEditable(false);
panelReg.add(txtSHex);

txtTDec.setBounds(80,170,60,18);
txtTDec.setEditable(false);
panelReg.add(txtTDec);

txtTHex.setBounds(145,170,60,18);
txtTHex.setEditable(false);
panelReg.add(txtTHex);

txtF.setBounds(80,195,125,18);
txtF.setEditable(false);
panelReg.add(txtF);

txtPCDec.setBounds(80,220,60,18);
txtPCDec.setEditable(false);
```

```
panelReg.add(txtPCDec);

txtPCHex.setBounds(145,220,60,18);
txtPCHex.setEditable(false);
panelReg.add(txtPCHex);

txtSW.setBounds(80,245,125,18);
txtSW.setEditable(false);
panelReg.add(txtSW);

//Start Address in Memory
JLabel lbStart = new JLabel("Start Address in Memory");
lbStart.setBounds(270,135,170,20);
add(lbStart);

txtStart.setEditable(false);
txtStart.setBounds(370,155,93,20);
add(txtStart);
JLabel lbTarget = new JLabel("Target Address:");
lbTarget.setBounds(270,180,100,20);
add(lbTarget);

txtTarget.setBounds(370,180,93,20);
txtTarget.setEditable(false);
add(txtTarget);

//instruction
JLabel lbInstructions = new JLabel("Instructions:");
lbInstructions.setBounds(270,210,100,20);
add(lbInstructions);
JTextArea txtInstructions = new JTextArea();
scrollPaneInst = new JScrollPane(txtInstructions); //스크
```

롤 추가

```
scrollPaneInst.setBounds(260,230,106,225);
add(scrollPaneInst);
instList = new JList<>();
scrollPaneInst.setViewportView(instList);

JLabel lbDevice = new JLabel("사용중인 장치");
lbDevice.setFont(new Font("고딕",Font.PLAIN,12));
lbDevice.setBounds(385,230,100,20);
add(lbDevice);

txtDevice.setBounds(395,250,60,20);
txtDevice.setEditable(false);
add(txtDevice);

//실행버튼
JButton btn1Step = new JButton("실행(1 Step)");
btn1Step.setFont(new Font("고딕",Font.PLAIN,10));
btn1Step.setEnabled(false);
btn1Step.setBounds(375, 365, 95, 20);
btn1Step.addActionListener(new ActionListener() { //실행
```

(1 Step)버튼을 눌렀을 때

```
    public void actionPerformed(ActionEvent arg0) {
        oneStep();
```

```
                update();
            }
        });
        add(btn1Step);

        JButton btnStep = new JButton("실행 (All)");
        btnStep.setFont(new Font("고딕",Font.PLAIN,10));
        btnStep.setEnabled(false);
        btnStep.setBounds(375, 395, 95, 20);
        btnStep.addActionListener(new ActionListener() { //실행
(All Step)버튼을 눌렀을 때
            public void actionPerformed(ActionEvent arg0) {
                allStep();
                update();
            }
        });
        add(btnStep);

        btnOpen.addActionListener(new ActionListener() { //오픈
버튼을 눌렀을 때
            public void actionPerformed(ActionEvent arg0) {
                JFileChooser chooser = new
JFileChooser(); // open할 파일을 선택할 때 JfileChooser를 쓰는 방식 예
제
                chooser.setDialogTitle("열기");
                FileNameExtensionFilter filter = new
FileNameExtensionFilter("Object File (*.obj)", "txt", "obj");
                chooser.setFileFilter(filter);
                int ret = chooser.showOpenDialog(null);
                if( ret ==
JFileChooser.APPROVE_OPTION){

                    filePath =
chooser.getSelectedFile().getPath();
                    String file =
chooser.getSelectedFile().getName();
                    inputText.setText(file);

                    try {

load(chooser.getSelectedFile());
                        update();

btn1Step.setEnabled(true);

btnStep.setEnabled(true);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
                else {//사용자가 창을 강제로 닫았거나
취소버튼을 누른 경우

JOptionPane.showMessageDialog(null,"Not Choose
File","WARNING",JOptionPane.WARNING_MESSAGE);
```

```
                    return;
                }
            }
        });

        JButton btnExit = new JButton("종료");
        btnExit.setFont(new Font("고딕",Font.PLAIN,10));
        btnExit.setBounds(375, 425, 95, 20);
        btnExit.addActionListener(new ActionListener() { //종료버
튼을 눌렀을 때
            public void actionPerformed(ActionEvent arg0) {
                resourceManager.closeDevice();
                System.exit(0);
            }
        });
        add(btnExit);

        //Log부분
        JLabel lbLog = new JLabel("Log(명령어 수행 관련):");
        lbLog.setBounds(20,457,150,30);
        add(lbLog);

        JTextArea txtLog = new JTextArea();
        scrollPane = new JScrollPane(txtLog); //스크롤 추가
        scrollPane.setBounds(10,483,460,165);
        add(scrollPane);
        logList = new JList<>();
        scrollPane.setViewportView(logList);
        setVisible(true);
    }

    /**
     * 프로그램 로드 명령을 전달한다.
     */
    public void load(File program){
        //...
        sicSimulator.load(program);
        sicLoader.load(program);
    };

    /**
     * 하나의 명령어만 수행할 것을 SicSimulator에 요청한다.
     */
    public void oneStep(){
        sicSimulator.oneStep();
    };

    /**
     * 남아있는 모든 명령어를 수행할 것을 SicSimulator에 요청한다.
     */
    public void allStep(){
        sicSimulator.allStep();
    };

    /**
```

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

		//log 출력
		String[] logStringList = sicSimulator.getLogList().toArray(new String[sicSimulator.getLogList().size()]);
		logList.setListData(logStringList);

		//명령어 리스트 출력
		String[] instStringList = sicSimulator.getInstList().toArray(new String[sicSimulator.getInstList().size()]);
		instList.setListData(instStringList);

		//스크롤 위치 아래로 조정

scrollPaneInst.getVerticalScrollBar().setValue(scrollPaneInst.getVerticalScrollBar().getMaximum());

scrollPane.getVerticalScrollBar().setValue(scrollPane.getVerticalScrollBar().getMaximum());

		//현재 실행중인 프로그램 섹션 이름 출력

txtProgramName.setText(resourceManager.getProgName(resourceManager.getCurrentSection()));

		//현재 실행중인 프로그램의 시작주소
		String start = "";
		if(resourceManager.getCurrentSection() == 0){
			start = String.format("%06X", resourceManager.getProgStart(resourceManager.getCurrentSection()));
			txtObjectProgram.setText(start);
			txtInObjectProgram.setText(start);
		}
		else {
			start = String.format("%06X", resourceManager.getProgStart(resourceManager.getCurrentSection()) - resourceManager.getProgLength(resourceManager.getCurrentSection()-1));
			txtObjectProgram.setText(start);
			txtInObjectProgram.setText(start);
		}

		//프로그램의 전체 길이

txtLengthOfProgram.setText(String.format("%06X",resourceManager.getProgTotalLen()));

		//레지스터 정보
		txtADec.setText(String.format("%d", resourceManager.getRegister(SicSimulator.A_REGISTER)));
		txtAHex.setText(String.format("%06X", resourceManager.getRegister(SicSimulator.A_REGISTER)));
```

```
txtXDec.setText(String.format("%d",
resourceManager.getRegister(SicSimulator.X_REGISTER)));
txtXHex.setText(String.format("%06X",
resourceManager.getRegister(SicSimulator.X_REGISTER)));

txtLDec.setText(String.format("%d",
resourceManager.getRegister(SicSimulator.L_REGISTER)));
txtLHex.setText(String.format("%06X",
resourceManager.getRegister(SicSimulator.L_REGISTER)));

txtBDec.setText(String.format("%d",
resourceManager.getRegister(SicSimulator.B_REGISTER)));
txtBHex.setText(String.format("%06X",
resourceManager.getRegister(SicSimulator.B_REGISTER)));

txtSDec.setText(String.format("%d",
resourceManager.getRegister(SicSimulator.S_REGISTER)));
txtSHex.setText(String.format("%06X",
resourceManager.getRegister(SicSimulator.S_REGISTER)));

txtSDec.setText(String.format("%d",
resourceManager.getRegister(SicSimulator.S_REGISTER)));
txtSHex.setText(String.format("%06X",
resourceManager.getRegister(SicSimulator.S_REGISTER)));

txtTDec.setText(String.format("%d",
resourceManager.getRegister(SicSimulator.T_REGISTER)));
txtTHex.setText(String.format("%06X",
resourceManager.getRegister(SicSimulator.T_REGISTER)));

txtF.setText(String.format("%f",
resourceManager.register_F));

txtPCDec.setText(String.format("%d",
resourceManager.getRegister(SicSimulator.PC_REGISTER)));
txtPCHex.setText(String.format("%06X",
resourceManager.getRegister(SicSimulator.PC_REGISTER)));

txtSW.setText(String.format("%06X",
resourceManager.getRegister(SicSimulator.SW_REGISTER)));

//start Address in Memory
txtStart.setText(String.format("%06X",
resourceManager.getProgStart(resourceManager.getCurrentSection())));

//Target Address
txtTarget.setText(String.format("%06X",
sicSimulator.getAddress()));

//사용중인 장치
txtDevice.setText(sicSimulator.getDevice());

};

public static void main(String[] args) {
new VisualSimulator();
```

```
        }
}
```