# 프로젝트2

**20181259**

**조수민**

**130**

**프로젝트 #1-c**

**<가>**

## 동기/목적

sic/xe 머신의 오브젝트 코드를 기반으로 GUI 시뮬레이터를 구현했다.

오브젝트 프로그램은 일단 파싱을 가장 신경써야 했고, 1pass와 2pass에서 해야할 일과 그걸 기반으로 어떻게 오브젝트 코드를 생성하는 지 공부할 수 있었다.

GUI 시뮬레이터를 구현하면서 실제로 오브젝트 코드가 어떻게 프로그램으로 실행되는 지 공부할 수 있었다.

로더를 통해 메모리에 적재된 오브젝트 코드를 읽으며 각 레지스터를 초기화 하고, 레지스터의 값을 참조하면서 분기하거나 return 혹은 jump를 통해 프로그램을 실행하는 과정을 익혔다.

## 설계/구현 아이디어

일단 GUI 프로그래밍에 대한 이해가 없었고, 구조를 설계하는 데 시간을 많이 썼기 때문에, 다른 코드를 많이 참조했다.

출처 : https://github.com/always0ne/SIC_XE_Simulator

참조한 구조는 크게 3가지이다.

1. loader에서 프로그램을 적재하는 방식
    a. 프로그램 로드 방식을 해당 코드에서 효율적으로 작성하였다고 판단하여 로더 부분을 차용하였다. 프로그램 길이까지 전부 0으로 채우고, modification record는 테이블에

추가한 후 오브젝트코드를 다 읽은 뒤 한번에 연산한다.

2. Instruction 명령어
    a. 이건 어려워서가 아니라, 단순히 시간이 모자랐기 때문에 사용했다. 사실 단순 반복의 영역이다.
3. 하이라이팅
    a. 하이라이팅 메소드를 참고하여 작성했다.

50퍼센트 이상일 것으로 예상한다. 하지만 코드를 전부 이해한 뒤 작성하였고, 단순 노가다 부분만 구조를 그대로 썼고, 핵심 부분은 스스로 짜거나 리팩토링 하였다.

사실 손이 가는대로 구현했다면 엄청 어렵진 않았을 것 같다, 하지만 명세에서 요구한 class의 기능에 맞추어 설계하는 것이 어려웠다.

메모리의 표현방식 2가지 중 2번이 이해가 가지 않아서 1번을 구현하였다. 1번은 가상 메모리를 gui로 출력하고 그 메모리에 rdrec 혹은 wrrec을 통해 변화하는 메모리를 보여주면 된다. 실제 copy program을 흉내내기 위해 또device를 파일로 가정하고, 파일이 복사되도록 구현하였다.

resourceManager가 가장 먼저 초기화 된다. 레지스터 정보와 program name, targetAddress등의 정보를 가지고 있다.

file 추가버튼을 누르면 ObjectCodeLoader를 통해 코드가 가상 메모리에 올라간다.

Sic/xeSimulator는 시뮬레이션을 수행하기 위한 클래스이고, instructionExecutor 클래스를 통해 instruction 한 스텝을 실제로 수행한다.

instructionExecutor는 실제 명령어와 동일한 행동을 수행한다. resourceManager를 통해 각 레지스터를 초기화 하며 수행된다.

메모리는 1바이트 단위로 띄어쓰기로 쓰여진다.

ex) 0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

# 수행 결과

## F1(Input Device)

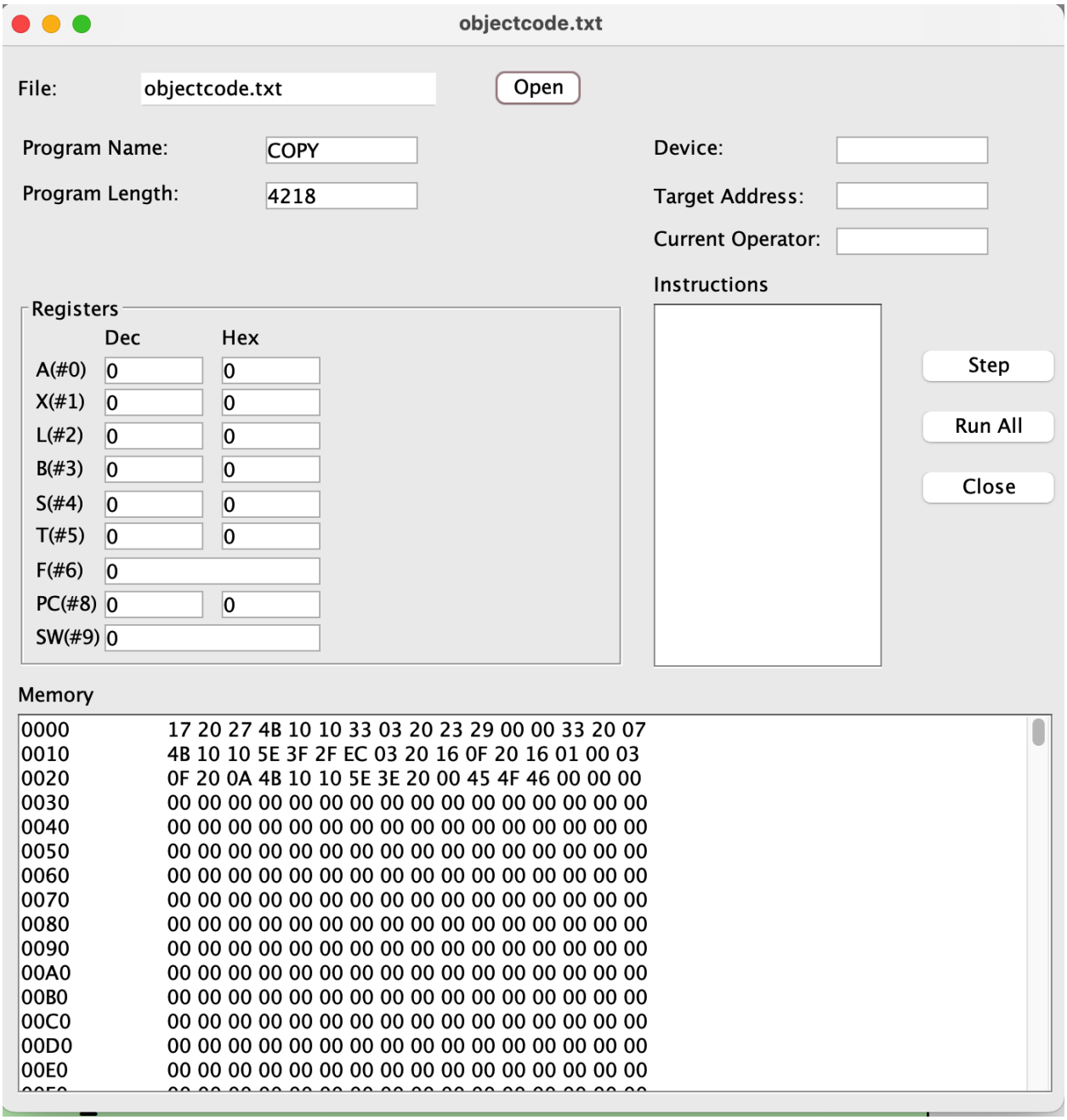
F1

```
SIC/XE Machine GUI Simulator
This file acts as an input device.
Write something in this file.
The simulator will read the input from this file
And write the output to the output device (F5), which will also be in this file.
```

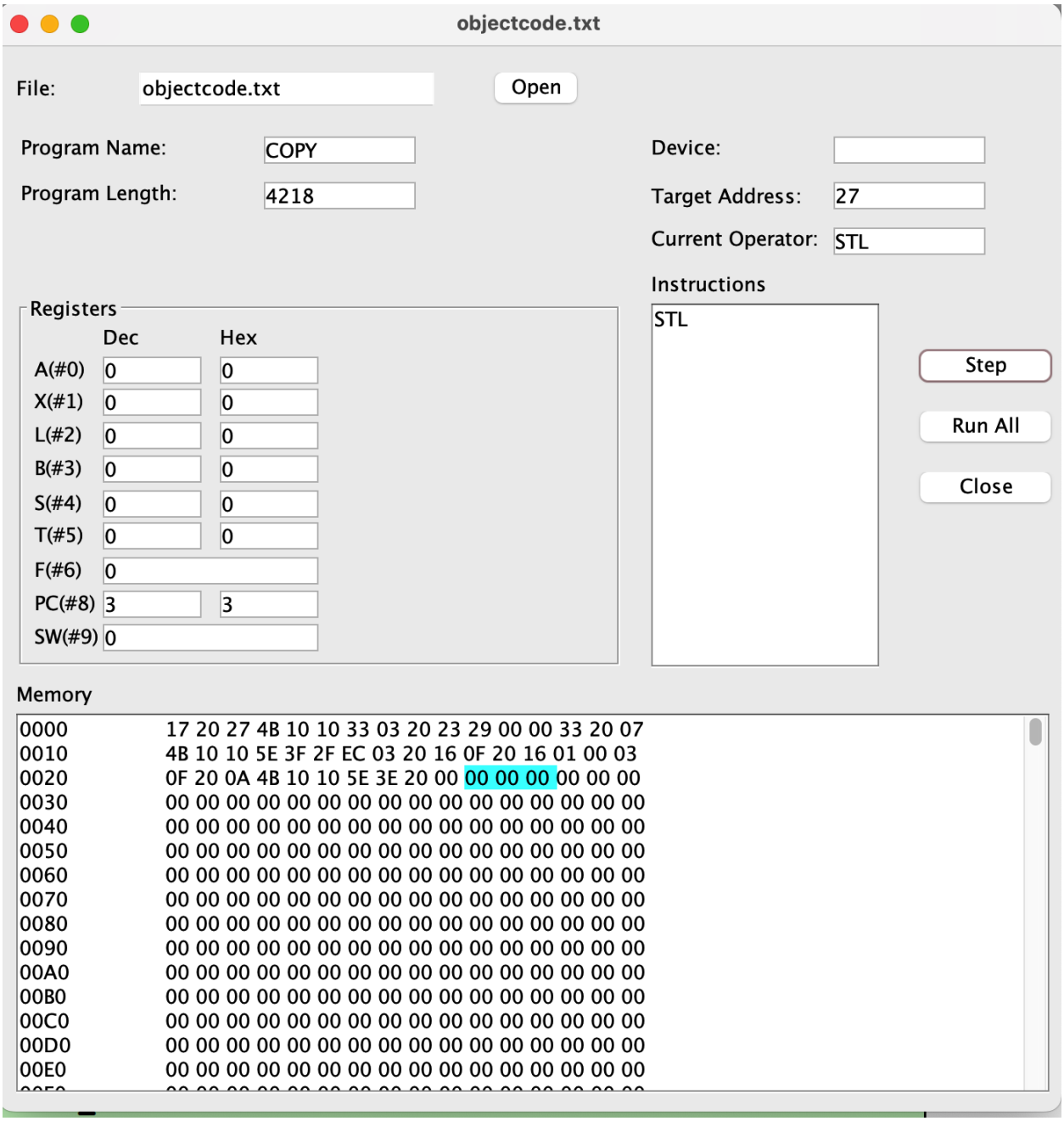
## 프로그램 실행

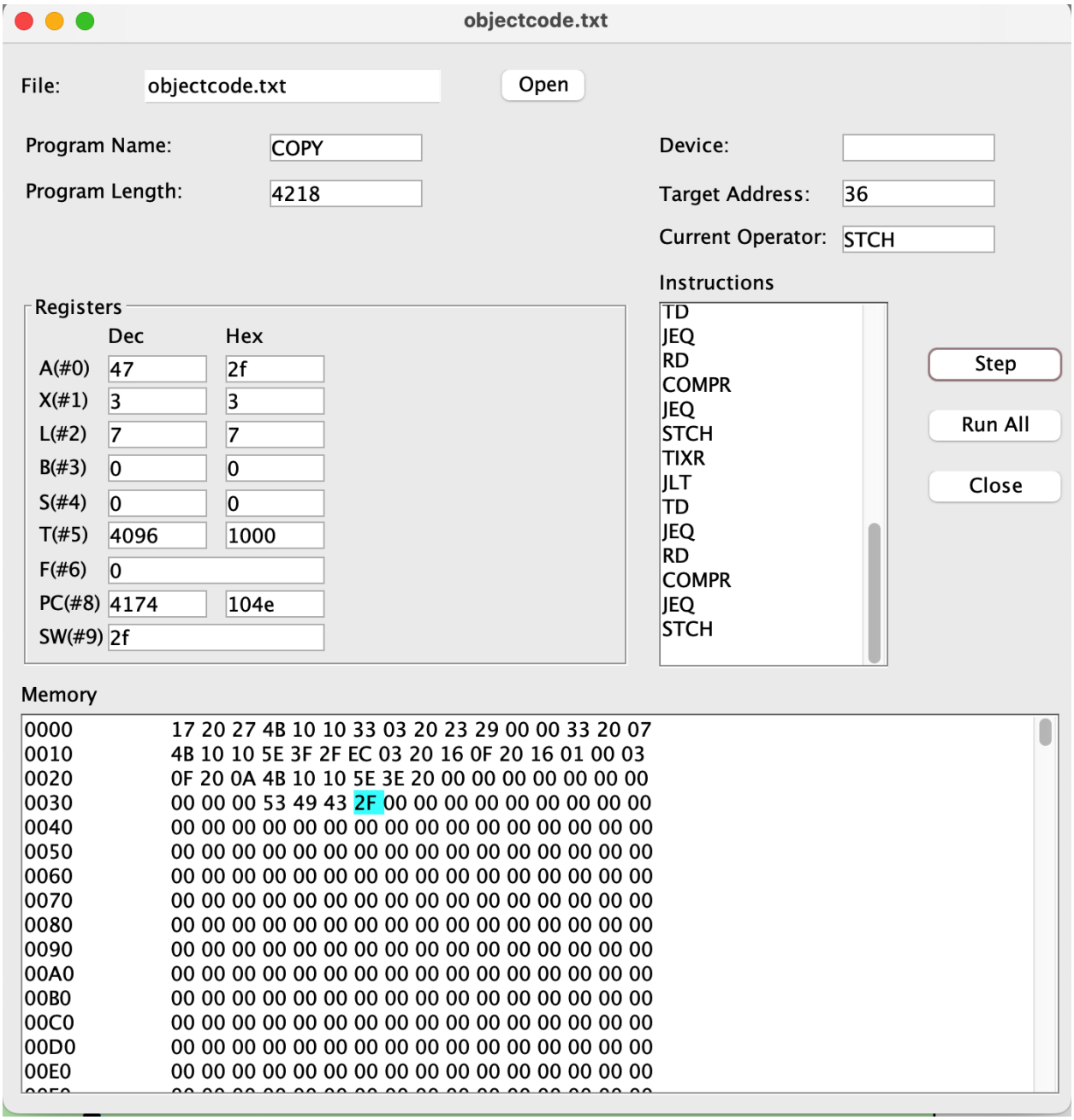

File:
Open
Program Name:
Program Length:
Device:
Target Address:
Current Operator:
Instructions
Registers
Dec
Hex
A(#0)
X(#1)
L(#2)
B(#3)
S(#4)
T(#5)
F(#6)
PC(#8)
SW(#9)
Step
Run All
Close
Memory

## Open

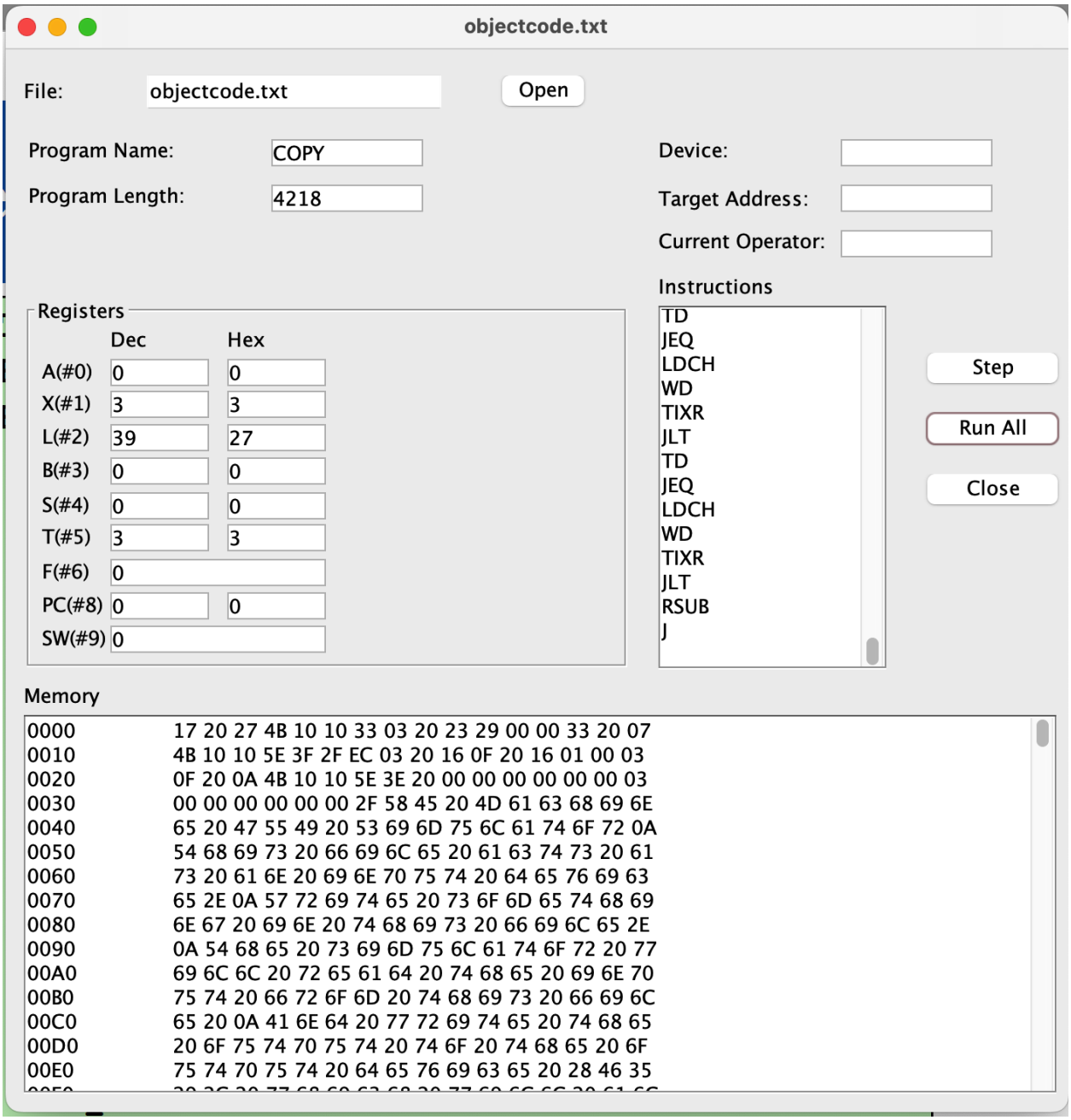

objectcode.txt
File:
objectcode.txt
Open
Program Name:
COPY
Program Length:
4218
Device:
Target Address:
Current Operator:
Instructions
Registers
Dec
Hex
A(#0) 0 0
X(#1) 0 0
L(#2) 0 0
B(#3) 0 0
S(#4) 0 0
T(#5) 0 0
F(#6) 0
PC(#8) 0 0
SW(#9) 0
Step
Run All
Close
Memory
0000 17 20 27 4B 10 10 33 03 20 23 29 00 00 33 20 07
0010 4B 10 10 5E 3F 2F EC 03 20 16 0F 20 16 01 00 03
0020 0F 20 0A 4B 10 10 5E 3E 20 00 45 4F 46 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

## Step


objectcode.txt
File:
objectcode.txt
Open
Program Name:
COPY
Program Length:
4218
Device:
Target Address:
27
Current Operator:
STL
Instructions
STL
Registers
Dec
Hex
A(#0) 0 0
X(#1) 0 0
L(#2) 0 0
B(#3) 0 0
S(#4) 0 0
T(#5) 0 0
F(#6) 0
PC(#8) 3 3
SW(#9) 0
Step
Run All
Close
Memory
0000 17 20 27 4B 10 10 33 03 20 23 29 00 00 33 20 07
0010 4B 10 10 5E 3F 2F EC 03 20 16 0F 20 16 01 00 03
0020 0F 20 0A 4B 10 10 5E 3E 20 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


objectcode.txt
File:
objectcode.txt
Open
Program Name:
COPY
Program Length:
4218
Device:
Target Address:
36
Current Operator:
STCH
Instructions
TD
JEQ
RD
COMPR
JEQ
STCH
TIXR
JLT
TD
JEQ
RD
COMPR
JEQ
STCH
Step
Run All
Close
Registers
Dec
Hex
A(#0) 47 2f
X(#1) 3 3
L(#2) 7 7
B(#3) 0 0
S(#4) 0 0
T(#5) 4096 1000
F(#6) 0
PC(#8) 4174 104e
SW(#9) 2f
Memory
0000 17 20 27 4B 10 10 33 03 20 23 29 00 00 33 20 07
0010 4B 10 10 5E 3F 2F EC 03 20 16 0F 20 16 01 00 03
0020 0F 20 0A 4B 10 10 5E 3E 20 00 00 00 00 00 00 00
0030 00 00 00 53 49 43 2F 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

## Run All




## 05(Output Device)

```
05
1  SIC/XE Machine GUI Simulator
2  This file acts as an input device.
3  Write something in this file.
4  The simulator will read the input from this file
5  And write the output to the output device (F5), which will also be in this file.
```

# 결론 및 보충할 점

초기에 계획한 바는 2 pass loader를 통해 유동적인 linking을 수행하고 싶었다.

하지만 난이도가 역량에 비해 너무 높아지고, 시험기간과 예비군으로 인해 오히려 규모를 줄였음에도 지각으로 제출하게 되었다.

이전의 과제는 전부 제너럴하게 실행될 수 있는 프로그램을 구현하고자 했다. 즉, 주어진 input 외에 다른 "상식적인 영역의" input 또한 에러 없이 실행되게 하는 것이 목적이었다. 하지만 이번 과제는 일반적인 input은 많이 실행할 수 없고, 주어진 input에 맞추어 설계하게 되었다. 무엇보다 시간이 가장 큰 문제였고, 시뮬레이터이기 때문에 명령어의 행동을 일일히 구현해야 일반적인 목적으로 사용할 수 있기 때문에 규모가 너무 컸다. 이 외에는 몇가지 쉽지만 하지 않은 사항들이 있다.

input 디바이스가 없을 때 에러, 그리고 output device가 없을 때 생성하는 기능

input에 있는 모든 명령어의 immediate 처리 (지금은 immediate가 있는 operator만 처리)

gui에 instruction 출력

등이 그것이다.

# 주요 소스코드

## ResourceManager.java

```
import java.io.*;
import java.util.HashMap;

public class ResourceManager {
    private static final int MEMORY_SIZE = 0x100000;  // 1MB
    byte[] memory = new byte[MEMORY_SIZE];
    int[] registers = new int[10];  // A, X, L, B, S, T, F, PC,
    HashMap<String, Object> deviceManager = new HashMap<>();
```

```
    String programName = "";
    int startAddr;
    int programLength;
    int currentMemory;

    String currentOperator;
    String targetAddress;
    String device;
    String log = "";

    String currentDevice = "";

    public void setDevice() {
        File file = new File("F1");
        try {
            deviceManager.put("F1", new FileInputStream(file));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        File file2 = new File("05");
        try {
            deviceManager.put("05", new FileOutputStream(file2)
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    HashMap<String, Integer> symtabList = new HashMap<>();

    public void setSymtabList(String name, int address) {
        if (symtabList.containsKey(name) == true) {
            return;
        }
        symtabList.put(name, address);

    }
```

```
    public int getSymtabList(String name) {
        if (symtabList.containsKey(name) == false) {
            return -1;
        }
        return symtabList.get(name);
    }

    int changedMemorySize = -1;
    int changedMemoryAddr = -1;

    public void initializeResource() {
        for (int i = 0; i < MEMORY_SIZE; i++) {
            memory[i] = 0;
        }
        for (int i = 0; i < registers.length; i++) {
            registers[i] = 0;
        }
        deviceManager.clear();
        programName = "";
        startAddr = 0;
        programLength = 0;
        currentMemory = 0;
        changedMemorySize = -1;
        changedMemoryAddr = -1;
    }

    public void setMemory(int address, byte[] data, int length)
        if (address < 0 || address + length > MEMORY_SIZE) {
            throw new IllegalArgumentException("Invalid memory a
        }
        System.arraycopy(data, 0, memory, address, length);
        changedMemoryAddr = address;
        changedMemorySize = length;
    }
```

```
    public byte[] getMemory(int address, int length) {
        if (address < 0 || address + length > MEMORY_SIZE) {
            throw new IllegalArgumentException("Invalid memory a
        }
        byte[] data = new byte[length];
        System.arraycopy(memory, address, data, 0, length);
        return data;
    }

    public void setRegister(int index, int value) {
        if (index < 0 || index >= registers.length) {
            throw new IllegalArgumentException("Invalid registe
        }
        registers[index] = value;
    }

    public int getRegister(int index) {
        if (index < 0 || index >= registers.length) {
            throw new IllegalArgumentException("Invalid registe
        }
        return registers[index];
    }

    public void TD(String name) {
        currentDevice = name;
        if (deviceManager.containsKey(name)) {
            registers[9] = 1;
        } else {
            registers[9] = 0;
        }
    }

    public byte RD() {
        if (currentDevice.equals("")) {
            registers[9] = -1;
            return 0;
```

```
        }
        InputStream inputStream = (InputStream) deviceManager.ge
        if (inputStream == null) {
            registers[9] = -1;
            return 0;
        }

        byte data = 0;
        try {
            int readData = inputStream.read();
            if (readData == -1) {
                registers[9] = 1; // End of file
            } else {
                data = (byte) readData;
                registers[9] = 0; // Successful read
            }
        } catch (Exception e) {
            registers[9] = -1; // Error during read
        }

        return data;
    }

    public byte WD() {
        if (currentDevice.equals("")) {
            registers[9] = -1;
            return 0;
        }
        OutputStream outputStream = (OutputStream) deviceManager
        if (outputStream == null || !(outputStream instanceof Ou
            registers[9] = -1;
            return 0;
        }

        byte data = (byte) registers[0]; // Assuming register A
        try {
```

```
            outputStream.write(data);
            outputStream.flush(); // Ensure the data is written
            registers[9] = 0; // Successful write
        } catch (Exception e) {
            registers[9] = -1; // Error during write
        }

        return data;
    }

    public void setCurrentDevice(String deviceName) {
        if (deviceManager.containsKey(deviceName)) {
            currentDevice = deviceName;
        } else {
            throw new IllegalArgumentException("Device not found
        }
    }

    public byte[] intToByte(int data) {
        String dataString = String.format("%06X", data);
        byte[] ret = new byte[3];
        ret[0] = (byte) Integer.parseInt(dataString.substring(0,
        ret[1] = (byte) Integer.parseInt(dataString.substring(2,
        ret[2] = (byte) Integer.parseInt(dataString.substring(4,
        return ret;
    }

    public int byteToInt(byte[] data) {
        return Integer.parseInt(String.format("%02X%02X%02X", da
    }
}
```

## InstructionExecutor - execute()

```
public void execute(int opcode, int format, byte[] data) {

        setInstruction(opcode);

        resourceManager.currentOperator = currentInstruction.nam
        resourceManager.targetAddress = Integer.toHexString(fetc

        //data 16진수로 이어서 출력
        for(int i = 0; i < data.length; i++) {
            resourceManager.log += String.format("%02X", data[i]
        }
        int nixbpe = fetchNixbpe(data);

        boolean isImmediate = (nixbpe & 0x30) >> 4 == 1;

        boolean isExtended = (nixbpe & 0x01) == 1;
        if (format == 2) {
            executeFormat2(data);
        } else {
            int targetAddress;
            if (isExtended) {
                resourceManager.setRegister(8, resourceManager.g
                targetAddress = fetchOperandAddress(data, 4);
            } else {
                resourceManager.setRegister(8, resourceManager.g
                targetAddress = fetchOperandAddress(data, 3);
            }
            executeFormat3or4(targetAddress, format, isImmediate
        }
    }

    private void executeFormat2(byte[] data) {
        // Implement format 2 instruction execution logic here
        String name = currentInstruction.name;
```

```
        int r1 = (data[1] & 0xF0) >> 4;
        int r2 = data[1] & 0x0F;


        switch (name) {
            case "CLEAR":
            resourceManager.setRegister(8, resourceManager.getRe
                resourceManager.setRegister(r1, 0);
                break;
            case "TIXR":
            resourceManager.setRegister(8, resourceManager.getRe
                resourceManager.setRegister(1, resourceManager.g
                resourceManager.setRegister(9, resourceManager.g
                break;
            case "COMPR":
            resourceManager.setRegister(8, resourceManager.getRe
                resourceManager.setRegister(9, resourceManager.g
                break;
            default:
                break;
        }
    }

    private void executeFormat3or4(int data, int format, boolean

        String name = currentInstruction.name;

        int numOperands = currentInstruction.numOperands;


        switch (name) {
            case "LDA":
                LDA(data, isImmediate);
                break;
            case "LDT":
                LDT(data);
```

```
                    break;
                case "LDCH":
                    LDCH(data, (numOperands == 2));
                    break;
                case "STCH":
                    STCH(data, (numOperands == 2));
                    break;
                case "STX":
                    STX(data);
                    break;
                case "STA":
                    STA(data);
                    break;
                case "STL":
                    STL(data);
                    break;
                case "COMP":
                    COMP(data, isImmediate);
                    break;
                case "JEQ":
                    JEQ(data);
                    break;
                case "JLT":
                    JLT(data);
                    break;
                case "JSUB":
                    JSUB(data);
                    break;
                case "J":
                    J(data, format);
                    break;
                case "RSUB":
                    RSUB();
                    break;
                case "TD":
                    TD(data);
```

```
                break;
            case "RD":
                RD();
                break;
            case "WD":
                WD();
                break;
            default:
                break;
        }
    }
```

## SicXeSimulator - oneStep()

```
    public void oneStep() {
        int pc = resourceManager.getRegister(8); // Assuming PC

        byte instruction = resourceManager.getMemory(pc, 1)[0];
        int opcode = instruction & 0xFC; // Extract the first 6

        if (instExecutor.validateOpcode(opcode)) {
            int formatTemp = instExecutor.getInstructionFormat(

            if (formatTemp == 2) {
                instExecutor.execute(opcode, formatTemp, resourc
                return ;
            }

            int nixbpe = instExecutor.fetchNixbpe(resourceManage
            boolean format4 = (nixbpe & 0x01) != 0;
            // Execute the instruction
            int format = format4 ? 4 : 3;
            instExecutor.execute(opcode, format, resourceManager
```

```
        } else {
            resourceManager.setRegister(8, pc + 1); // Increment
        }
        // Continue with your instruction execution logic
    }
```