

SIC/XE Simulator의 구현

System Programming Project #2

Jaeyoung Choi

choi@ssu.ac.kr

Contents

1. 프로젝트 목표
2. 필수 구현기능
3. 프로그램 전체 구조
4. GUI 환경 구현 예제
5. 코드 예시

1. 프로젝트 목표

- SIC/XE 기계의 동작을 수행할 시뮬레이터 개발
- SIC/XE Simulator
 - ◆ 입력
 - SIC/XE 어셈블러를 통해 만들어진 object code
 - ◆ 출력
 - 해당 Object 코드에 따른 프로그램 실행의 결과
COPY Program 실행의 결과

2. 필수 구현 기능

- 교재에 나온 모든 SIC/XE 명령어 처리
- 모니터링 구현
 - ◆ Object Code 별 모니터링
 - ◆ 단계별 Instruction 수행과정
 - ◆ Register 모니터링

3.1 프로그램 전체 구조 설명

① Visual Simulator

- ◆ 사용자 인터페이스
- ◆ 이벤트를 SIE/XE Simulator에게 전달

② SIC/XE Simulator

- ◆ 명령어의 실질적인 수행
- ◆ Resource Manager 컨트롤

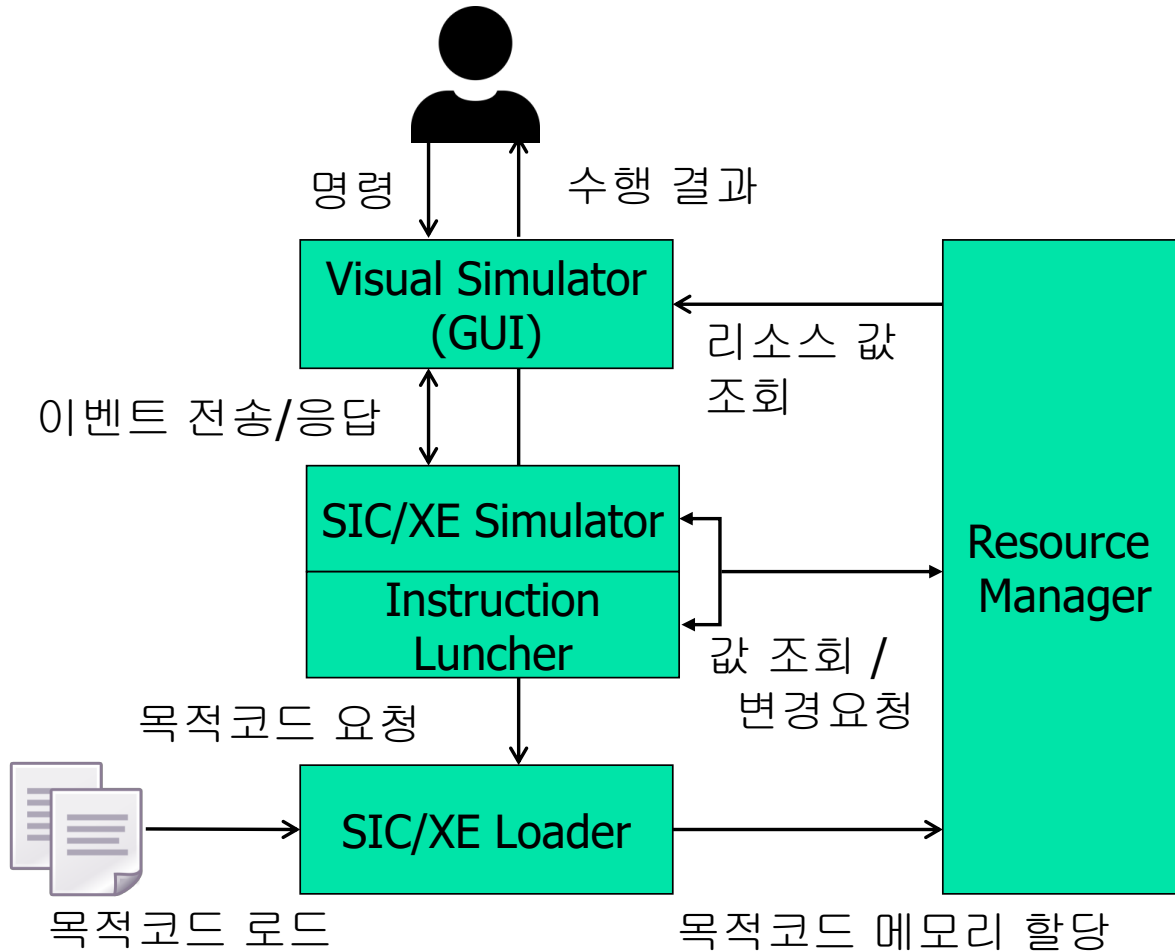
③ SIC/XE Loader

- ◆ 가장 먼저 수행되는 모듈.
- ◆ Object code를 memory 영역에 로드

④ Resource Manager

- ◆ 가상의 메모리, 레지스터, 디바이스 관리

3.2 프로그램 전체구조도



3.3 단계별 실행과정

1. SIC/XE Loader가 Object code를 메모리에 로딩
2. SIC/XE Simulator가 instruction을 가져와서 해석
3. Instruction의 요구 데이터를 Resource Manager 에게 요청
4. 받은 데이터를 이용하여 SIC/XE Simulator가 연산 수행
 - ◆ 이 과정에서 resource (register, memory, device)의 상태 값 변경 발생
5. Object code가 남아 있으면 1~4번과정 반복

4.1 GUI 환경 예제 – main 화면

The screenshot shows a GUI application window with a light blue border and standard Windows-style window controls (minimize, maximize, close) in the top right corner. The interface is organized into several sections:

- File Selection:** At the top left, there is a text field labeled "FileName :" followed by an "open" button.
- Header Record (H):** A group box containing three input fields: "Program name :", "Start Address of Object Program :", and "Length of Program :".
- End Record (E):** A group box containing two input fields: "Address of First instruction in Object Program :" and "Start Address in Memory".
- Register Table:** A table with two columns, "Dec" and "Hex", listing registers A (#0) through SW (#9). Each register name is followed by a text input field for its decimal value and another for its hexadecimal value.
- Target Address:** A single text input field.
- Instructions:** A large, empty text area for entering or viewing instructions.
- Execution Controls:** To the right of the instructions area, there is a "사용중인 장치" (Device in use) label above a text field, and three buttons: "실행 (1step)", "실행 (all)", and "종료" (End).
- Log:** At the bottom, a label "Log (명령어 수행 관련):" is positioned above a large, empty text area for logging command execution.

4.2 GUI 환경 예제 – 로드 결과

FileName : ← Load button

H (Header Record)

Program name : 이름

Start Address of Object Program : 시작주소

Length of Program :

E (End Record)

Address of First instruction in Object Program :

Start Address in Memory :

Target Address :

Instructions : 사용종연 장치

실행(1step)

실행 (all)

종료

Log (명령어 수행 관련):

Register	Dec	Hex
A (#0)	<input type="text" value="0"/>	<input type="text" value="0"/>
X (#1)	<input type="text" value="0"/>	<input type="text" value="0"/>
L (#2)	<input type="text" value="0"/>	<input type="text" value="0"/>
B (#3)	<input type="text" value="0"/>	<input type="text" value="0"/>
S (#4)	<input type="text" value="0"/>	<input type="text" value="0"/>
T (#5)	<input type="text" value="0"/>	<input type="text" value="0"/>
F (#6)	<input type="text" value="0"/>	<input type="text" value="0"/>
PC (#8)	<input type="text" value="0"/>	<input type="text" value="0"/>
SW (#9)	<input type="text" value="0"/>	<input type="text" value="0"/>

4.3 GUI 환경 예제 - 실행화면

FileName :

H (Header Record)

Program name :

Start Address of Object Program :

Length of Program :

E (End Record)

Address of First Instruction in Object Program :

Start Address in Memory :

Target Address :

Register

	Dec	Hex
A (#0)	<input type="text" value="70"/>	<input type="text" value="000046"/>
X (#1)	<input type="text" value="3"/>	<input type="text" value="000003"/>
L (#2)	<input type="text" value="39"/>	<input type="text" value="000027"/>
B (#3)	<input type="text" value="0"/>	<input type="text" value="000000"/>
S (#4)	<input type="text" value="0"/>	<input type="text" value="000000"/>
T (#5)	<input type="text" value="3"/>	<input type="text" value="000003"/>
F (#6)	<input type="text" value="000000"/>	
PC (#8)	<input type="text" value="0"/>	<input type="text" value="000000"/>
SW (#9)	<input type="text" value="000002"/>	

Instructions :

172027 ← 처리중

4B101033

032023

290000

332007

4B10105E

3F2FEC

032016

0F2016

010003

0F200A

Log (명령어 수행 관련):

JEQ

LDCH

WD

TIXR

JLT

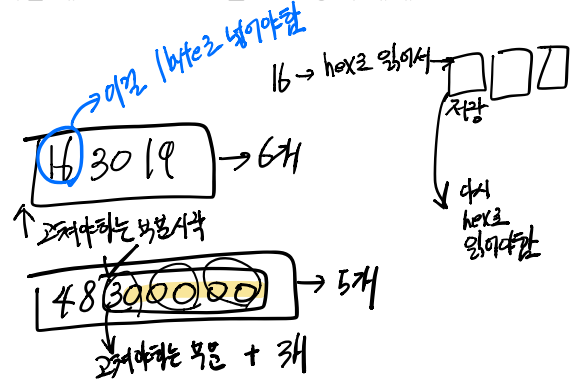
RSUB

J

5.1 코드 예시 – GUI 이벤트 처리

```
JButton btnRunstep = new JButton("run(1step)"); // 버튼 생성
btnRunstep.addActionListener(new ActionListener() { // 버튼이 눌렸을 때의 작동방식 선언. 여기서는 임시 ActionListener 생성
    public void actionPerformed(ActionEvent argo) { // ActionListener의 구체적인 내용 작성
        oneStep(); // 실제 작업은 oneStep() 함수에서 정의할 것.
        update(); // 작업이 끝나면 visualSimulator 화면 업데이트
    }
});
```

```
btnOpen.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent argo) {
        JFileChooser chooser = new JFileChooser(); // open할 파일을 선택할 때 JfileChooser를 쓰는 방식 예제
        int ret = chooser.showOpenDialog(null);
        if( ret == JFileChooser.APPROVE_OPTION){
            try {
                initialize();
                load(chooser.getSelectedFile());
                update();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
});
```



5.2 코드 예시 – 로더 입력처리

// 로드하는 데이터의 태그별로 처리방법을 결정한다.

```
switch(line.charAt(o)) {  
    case 'H':  
        //H와 관련된 정보를 Resource manager를 통해서 저장  
        //필요한 변수와 함수는 자유롭게 선언 가능  
        rMgr.setProgname(line.substring(1, 7), currentSection);  
        rMgr.setProgLength(line.substring(13,19), currentSection);  
        rMgr.setStartADDR(currentSection);  
  
        // SYMTAB 등록  
        rMgr.symtabList.putSymbol(  
            line.substring(1,7), Integer.parseInt(rMgr.getStartADDR(currentSection), 16));  
        ...  
        break;  
    Case 'T':  
    Case 'M':  
    ...  
}
```

5.3 코드 예시 – instruction 처리(1)

◆ LDA 예시

```
// instruction은 instruction 정보를 가지고 있는 임의의 클래스  
If(instruction.opcode==0){    // LDA opcode와 비교  
    instluncher.lda(instruction) // InstLuncher에서 LDA 메소드 호출  
}
```

5.4 코드 예시 – instruction 처리(2)

// InstLuncher의 LDA 메소드

```
public void LDA(Instruction instruction) {
```

```
    // Target Address 계산
```

```
    switch(instruction.getFlag(bFlag|pFlag)){           // instruction의 b,p bit 정보를 불러온다.
```

```
        case 0:
```

```
            if(instruction.getFlag(eFlag)){
```

```
                // TA=address(4byte format)
```

```
            }else{
```

```
                // TA=disp (3byte format)
```

```
            }
```

```
            break;
```

```
        case bFlag: // TA= (B) + disp
```

```
            break;
```

```
        case pFlag: // TA= (PC) + disp
```

```
            break;
```

```
    }
```

... (다음장에서 계속)

5.5 코드 예시 – instruction 처리(3)

```
switch(instruction.getFlag(nFlag|iFlag) { //instruction의 n, i bit정보를 확인한다.
    case 0:
        case nFlag|iFlag : // n,i 모두 0이거나 모두 1인 경우 simple addressing
            if(e_flag == 1)
                //0번 레지스터에 Target address로부터 16진수 값을 로드
                //과제로 주어지는 getMemory()함수는 char[] 타입이므로 Integer.parseInt가
                //적용되지 않는다. 각자의 방법으로 Integer로 변환한다.
                rMgr.setregister( 0,(Integer.parseInt (getMemory(TA,5),16)));
            else
                rMgr.setregister( 0, (Integer.parseInt(getMemory(TA, 3), 16)));
            break;
        case nFlag : // indirect addressing
            break;
        case iFlag : // immediate addressing
            break;
    }
}
```