과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 3. Linker / Loader>>

**서강대학교 컴퓨터공학과**

**20121578**

**김태훈**

목 차

1. 프로그램 개요 4

2. 프로그램 설명 5

2.1 프로그램 흐름도 5

3. 모듈 정의 7

3.1 모듈 이름 : main() 7

3.1.1 기능 7

3.1.2 사용 변수 7

3.2 모듈 이름: sic\_menu() 7

3.2.1 기능 7

3.2.2 사용 변수 7

3.3 모듈이름: sic\_menu\_classify() 8

3.3.1 기능 8

3.3.2 사용변수 8

3.4 모듈이름: trim\_left() 8

3.4.1 기능 8

3.4.2 사용변수 8

3.5 모듈이름: trim\_right() 8

3.5.1 기능 8

3.5.2 사용변수 8

3.6 모듈이름: err\_msg\_bundle() 8

3.5.1 기능 8

3.5.2 사용변수 8

3.7 모듈이름: opcode\_init() 9

3.5.1 기능 9

3.5.2 사용변수 9

3.8 모듈이름: opcode\_free() 9

3.5.1 기능 9

3.5.2 사용변수 9

3.9 모듈이름: opcode\_table\_print() 9

3.5.1 기능 9

3.5.2 사용변수 9

3.10 모듈이름: opcode\_find() 9

3.5.1 기능 9

3.5.2 사용변수 9

3.11 모듈이름: history\_add() 10

3.5.1 기능 10

3.5.2 사용변수 10

3.12 모듈이름: history\_free() 10

3.5.1 기능 10

3.5.2 사용변수 10

3.13 모듈이름: sic\_help() 10

3.5.1 기능 10

3.5.2 사용변수 10

3.14 모듈이름: sic\_history\_print() 10

3.5.1 기능 10

3.5.2 사용변수 10

3.15 모듈이름: sic\_dir() 10

3.5.1 기능 10

3.5.2 사용변수 10

3.16 모듈이름: mem\_hexa\_convert() 11

3.5.1 기능 11

3.5.2 사용변수 11

3.17 모듈이름: mem\_dump\_only() 11

3.5.1 기능 11

3.5.2 사용변수 11

3.18 모듈이름: mem\_dump\_first\_only() 11

3.5.1 기능 11

3.5.2 사용변수 11

3.19 모듈이름: mem\_dump\_first\_last() 11

3.5.1 기능 11

3.5.2 사용변수 12

3.20 모듈이름: mem\_edit() 12

3.5.1 기능 12

3.5.2 사용변수 12

3.21 모듈이름: mem\_fill() 12

3.5.1 기능 12

3.5.2 사용변수 12

3.22 모듈이름: mem\_reset() 12

3.5.1 기능 12

3.5.2 사용변수 12

3.23 모듈이름: asmCreate() 13

3.5.1 기능 13

3.5.2 사용변수 13

3.24 모듈이름: asmFirstPass() 14

3.5.1 기능 14

3.5.2 사용변수 14

3.25 모듈이름: asmSecondPass() 14

3.5.1 기능 14

3.5.2 사용변수 14

3.26 모듈이름: opSearch() 15

3.5.1 기능 15

3.5.2 사용변수 15

3.27 모듈이름: opcodeNumberSearch() 15

3.5.1 기능 15

3.5.2 사용변수 15

3.28 모듈이름: objFormatTwo() 15

3.5.1 기능 15

3.5.2 사용변수 15

3.29 모듈이름: objFormatThree() 15

3.5.1 기능 15

3.5.2 사용변수 15

3.30 모듈이름: objFormatFour() 16

3.5.1 기능 16

3.5.2 사용변수 16

3.31 모듈이름: objFileMaking() 16

3.5.1 기능 16

3.5.2 사용변수 16

3.32 모듈이름: symbolTableSearch() 16

3.5.1 기능 16

3.5.2 사용변수 16

3.33 모듈이름: symbolTableInsert() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.34 모듈이름: freeSymbolTable() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.35 모듈이름: printSymbolTable() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.36 모듈이름: sicType() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.37 모듈이름: printBreakPoint() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.38 모듈이름: insertBreakPoint() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.39 모듈이름: clearBreakPoint() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.40 모듈이름: searchBreakPoint() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.41 모듈이름: loaderBundle() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.42 모듈이름: loaderPass1() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.43 모듈이름: loaderPass2() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.44 모듈이름: loaderMod() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.45 모듈이름: runProgram() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.46 모듈이름: saveRegHistory() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.47 모듈이름: opcodeFormat() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.48 모듈이름: formatTwoCheck() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.49 모듈이름: readMemory() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.50 모듈이름: storeMemoryFormat3() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.51 모듈이름: searchESTAB() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.52 모듈이름: insertESTAB() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.53 모듈이름: deallESTAB() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.54 모듈이름: printESTAB() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.55 모듈이름: insertRefList() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.56 모듈이름: searchRefList() 17

3.5.1 기능 17

3.5.2 사용변수 17

3.57 모듈이름: deallRefList() 17

3.5.1 기능 17

3.5.2 사용변수 17

4. 전역 변수 정의 18

4.1 #define Macros 18

4.2 history\_prt his\_list\_front, history\_ptr his\_list\_rear 18

4.3 opcode\_ptr opcode\_table[HASH\_SIZE] 18

4.4 char mem\_arr[MEM\_ROW\_SIZE][MEM\_COL\_SIZE] 19

4.5 symbol\_ptr last\_symbol\_table[SYMBOL\_TABLE\_SIZE] 19

# 프로그램 개요

본 수업에서는 시스템 소프트웨어의 작동 원리를 이해하기 위해 SIC / XE Machine을 학습하고, 직접 구현한다. SIC (Simplified Instructional Computer) Machine은 시스템 소프트웨어 원리 이해를 위한 교육용 Machine으로, 개량 버전으로 SIC / XE (SIC Extended) Machine이 있다.

본 프로그램은 프로젝트 #1, #2에서 구현했던 SIC / XE Machine에 Linker, Loader를 추가한 프로그램이다.

본 프로그램의 소스 파일, 헤더 파일 및 리소스 파일의 구성은 다음과 같다.

1. 소스 파일

* 20121578.c : main() 함수 포함
* cmd\_ctr.c : sic\_menu() 함수 포함
* cmd\_execution.c : Memory, Hash Table 관리를 제외한 function들
* cmd\_mem\_ctr.c : Memory 관리와 관련된 function들
* cmd\_opcode.c : Opcode Hash Table과 관련된 function들
* asm\_list\_create.c : Assembler Pass 1, Pass 2 Function 포함됨
* asm\_symbol\_ctr.c : Assembler Symbol Table을 관리하는 Function들 포함됨.
* asm\_obj\_format.c : 각 Mnemonic의 Format에 따라 Object Code 생성
* asm\_obj\_create.c : .obj 파일 생성
* linkload\_load\_bp.c : Loader 및 Breakpoint 관련 함수
* linkload\_run.c : run 기능과 관련된 함수

1. 헤더 파일

* 20121578.h

1. 리소스 파일

* opcode.txt : opcode들의 information이 저장되어 있는 파일

# 프로그램 설명

## 프로그램 흐름도

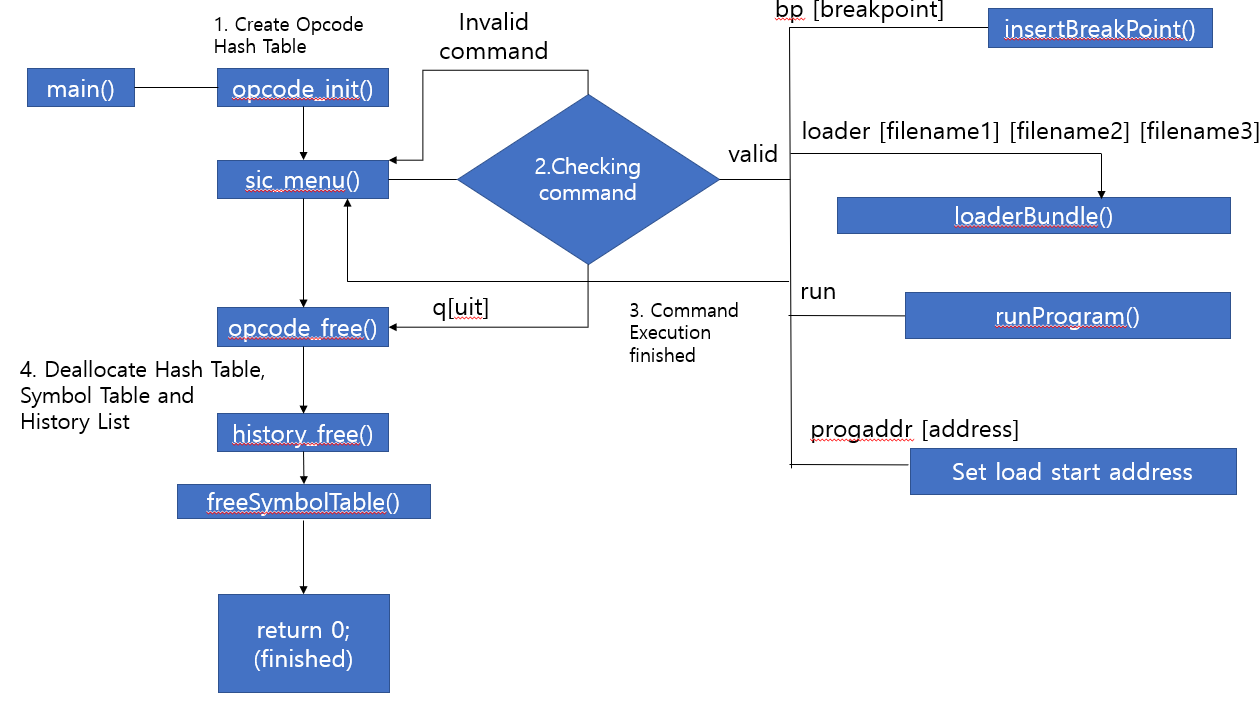


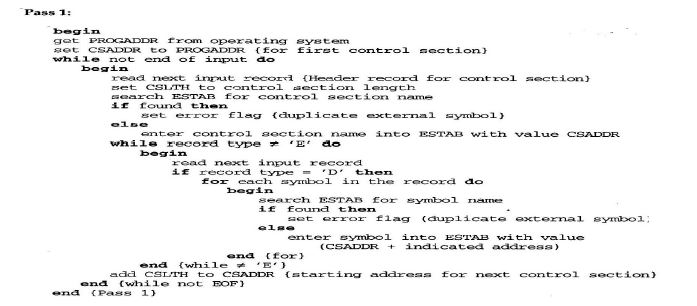
Figure 1. Program Flowchart (Project 3 추가기능)

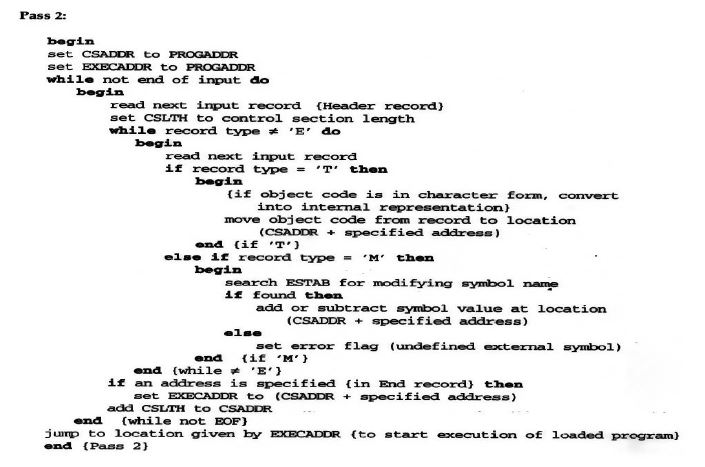
본 프로젝트에서는 4개의 기능 (bp, loader, run, progaddr) 이 추가되었다.

첫 번째로 progaddr는 loader나 run 명령어를 수행할 때의 시작 지점을 지정하는 기능이다. 프로그램 시작 시 default 값은 0x00이다.

두 번째로, loader는 입력한 object file을 memory 에 load 해주는 기능이다. 본 프로그램은 최대 3개의 object file을 입력 받아 load할 수 있다.

세 번째로 bp는run을 실행할 때의 breakpoint를 추가한다 run을 수행할 때, 프로그램은 breakpoint까지만 실행되고 멈춘다.





위의 사진은 Loader에 사용되는 Algorithm의 Pass 1, Pass 2이다.

# 모듈 정의

## 모듈 이름 : main() (20121578.c)

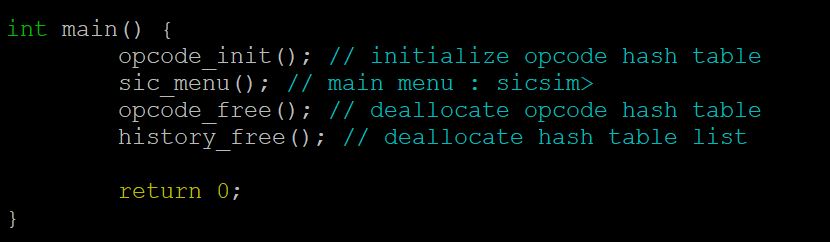


Figure 2. Code of main() function

### 기능

Opcode Table을 생성하고, SIC / XE Machine을 실행한다. SIC / XE Machine을 종료하라는 Command, 즉 q[uit]를 입력 받으면 sic\_menu()를 종료하고, opcode\_free()로 Opcode Hash Table을 Deallocate하고, history\_free()로 History List를 Deallocate 한다.

### 사용 변수

없음

## 모듈 이름: sic\_menu() (cmd\_ctr.c)

### 기능

Command를 입력 받고 Tokenize한 후에, 입력 오류를 Check한다. 입력에 오류가 있을 경우, 입력을 다시 받는다. 입력에 오류가 없을 시, sic\_menu\_classify() 함수를 실행시켜 입력한 Command에 맞는 함수를 실행한다. 오류 없이 실행되었을 경우, History List에 Command를 저장한다. sic\_menu\_classify() 에서 0을 받는 경우에는 함수를 종료한다.

### 사용 변수

-char input\_str[256] : Command를 입력 받는 변수

-char history\_str[256] : History List에 저장할 Command

-char copy\_str[256], char \*check\_str : Command 입력 error checking을 위한 문자열

-char \*input\_arr[5] : Tokenize된 Command를 저장하는 char pointer 배열

-char \*check\_arr[5] : Command 입력 error checking을 위한 char pointer 배열

-const char \*cmd\_tok : 공백을 Tokenize하기 위한 Token

-const char \*cmd\_comma\_tok : comma와 공백을 Tokenize하기 위한 Token

-const char \*comma\_only\_tok : comma만을 Tokenize하기 위한 Token. 입력 error checking에

사용됨.

-int i : 반복문을 실행시키기 위한 변수

-int cmd\_num : Tokenize된 Command의 문자열 개수

-int exec\_res : 함수 실행 결과. sic\_menu\_classify() 에서 결과값을 받는다.

-int input\_check\_num : input\_str으로입력 받은 Command의 길이

-int cur\_mem\_ctr : du[mp]를 다시 실행했을 때의 시작 주소 값

## 모듈이름: sic\_menu\_classify() (cmd\_ctr.c)

### 기능

입력 받은 Command에 맞는 함수를 실행한다. 만약 입력 받은 Command가 본 프로그램에서 유효하지 않은 Command거나 정상적으로 실행되지 않았을 경우 -1을 return하고, q[uit]를 입력 받았을 경우 0을 return 한다. hi[story]를 입력 받고 정상적으로 실행되었을 경우 2를 return한다. 나머지 Case에 정상적으로 실행되었을 경우 1을 return 한다.

### 사용변수

1. Parameters

-char \*\*input\_arr : Tokenize한 Command

-char \*history\_str : History List에 저장할 Command

-int \*cur\_mem\_ctr : du[mp] 를 다시 실행했을 때 시작 주소를 저장하는 변수

1. Local Variables

-int du\_start, du\_end : du[mp] 의 시작, 끝 주소

-int edit\_addr, edit\_val :e[dit] 의 주소와 ASCII Code 값

-int fill\_start, fill\_end, fill\_val : f[ill]의 시작, 끝 주소와 ASCII Code 값

## 모듈이름: trim\_left() (cmd\_ctr.c)

### 기능

Parameter로 받은 문자열의 왼쪽 공백을 없앤다.

### 사용변수

char \*tok\_num : 공백을 없앨 문자열

## 모듈이름: trim\_right() (cmd\_ctr.c)

### 기능

Parameter로 받은 문자열의 오른쪽 공백을 없앤다.

### 사용변수

1. Parameter

-char \*tok\_num : 공백을 없앨 문자열

1. Local Variable

-char \*end : 문자열의 오른쪽 끝

## 모듈이름: err\_msg\_bundle() (20121578.c)

### 기능

특정 ERROR 상황에 맞는 error message를 출력한다.

### 사용변수

int err\_state : ERROR 상황을 구분하는 변수

## 모듈이름: opcode\_init() (cmd\_opcode.c)

### 기능

opcode.txt 파일을 읽고 Opcode Hash Table을 생성한다.

이 때 Opcode Hash Table의 Index 값은 0~19 이며, Index 값은 mnemonic의 문자들의 ASCII Code 값의 합을 20으로 나눈 나머지이다.

### 사용변수

FILE \*op\_txt : opcode.txt 파일을 가리키는 file pointer

int op\_num\_reader :opcode number를 읽는 변수

char op\_cmd\_reader : opcode mnemonic을 읽는 변수

char op\_mode\_reader : opcode의 mode를 읽는 변수

int op\_cmd\_len : mnemonic의 길이

int op\_table\_idx : mnemonic의 Opcode Hash Table index

opcode\_ptr new\_node : Opcode Hash Table의 node

int i : 반복문을 실행시키기 위한 변수

## 모듈이름: opcode\_free() (cmd\_opcode.c)

### 기능

Opcode Hash Table의 모든 node들을 Deallocate한다.

### 사용변수

int i : 반복문을 실행시키기 위한 변수

opcode\_ptr de\_node, temp\_node : Deallocation을 위한 변수

## 모듈이름: opcode\_table\_print() (cmd\_opcode.c)

### 기능

Opcode Hash Table을 출력한다.

### 사용변수

int i : 반복문을 실행시키기 위한 변수

opcode\_ptr temp\_node : Opcode Hash Table 탐색을 위한 변수

## 모듈이름: opcode\_find() (cmd\_opcode.c)

### 기능

opcode mnemonic에서, 입력 받은 mnemonic을 Opcode Hash Table에서 찾는다. 만약 찾으면 mnemonic의 opcode를 출력한 후 0을 return하고, 찾지 못하면 -1을 return한다.

### 사용변수

1. Parameters

-char \*op\_mnec : Opcode Hash Table에서 찾으려 하는 mnemonic

-int mnec\_len : mnemonic의 길이

1. Local Variables

-int i : 반복문을 실행시키기 위한 변수

-int op\_idx : 찾으려는 mnemonic이 있는 Opcode Hash Table의 index

-opcode\_ptr temp\_node : Opcode Hash Table 탐색을 위한 pointer 변수

## 모듈이름: history\_add() (cmd\_execution.c)

### 기능

입력 받은 Command가 정상적으로 종료되면, History List에 입력 받은 Command를 저장한다. 이 때 처음에 입력 받았던 형태 그대로 저장한다.

### 사용변수

1. Parameter

-char \*history\_input : History List에 새로 추가할 Command string

1. Local Variable

- history\_ptr new\_history : History List에 새로 추가할 node

## 모듈이름: history\_free() (cmd\_opcode.c)

### 기능

History List에 연결되어 있는 모든 node들을 Deallocate한다.

### 사용변수

-history\_ptr de\_node, temp\_node : Deallocation에 사용되는 변수들

## 모듈이름: sic\_help() (cmd\_execution.c)

### 기능

본 프로그램에서 입력할 수 있는 Command들을 모두 출력한다. 일종의 도움말 기능이다.

### 사용변수

없음

## 모듈이름: sic\_history\_print() (cmd\_execution.c)

### 기능

현재 History List에 저장된 Command들을 출력한다. History List에는 정상적으로 실행된 Command만 저장되며, 입력한 Command 그대로 출력한다.

### 사용변수

int his\_num : History List의 순서를 저장하고 출력하는 변수

history\_ptr temp\_his : History List의 가장 앞을 가리키는 pointer 변수

## 모듈이름: sic\_dir() (cmd\_execution.c)

### 기능

현재 디렉토리에 있는 파일들을 출력한다.

### 사용변수

Local Variables

- DIR \*cur\_dir : 디렉토리를 가리키는 변수

- struct dirent \*f\_point : Dirent 구조체

- struct stat s\_buf : 디렉토리의 상태를 저장하는 변수

## 모듈이름: mem\_hexa\_convert() (cmd\_mem\_ctr.c)

### 기능

문자열의 형태로 입력된 16진수를 int 형으로 형 변환해준다. 만약 0~9나 A(a)~F(f) 이외의 문자가 입력 되었을 시, 잘못된 입력으로 처리한다.

### 사용변수

1. Parameters

-char \*num\_str : 변환할 16진수 형태의 문자열

2) Local Variables

-int i,j : 반복문을 실행시키기 위한 변수

-int num\_len : num\_str의 길이

-int cur\_idx : 현재 변환 중인 자리 수

-int num\_res : int형으로 형 변환된 16진수 형태의 문자열

-int hexa\_start : 16진수의 자리 수. 초기 값은 1이다.

## 모듈이름: mem\_dump\_only() (cmd\_mem\_ctr.c)

### 기능

10줄의 메모리에 할당되어 있는 값을 출력한다. 만약 처음으로 du[mp] 를 입력했거나, 마지막으로 출력된 주소가 0xFFFFF이면 처음부터, 만약 마지막으로 출력된 주소에 10줄 (160byte) 을 더했을 때 0xFFFFF를 넘으면 0xFFFFF까지, 그렇지 않으면 10줄을 출력한다.

### 사용변수

1. Parameters

-int \*cur\_mem\_ctr : 다시 du[mp]를 실행했을 때의 시작 주소 값을 저장하는 변수

1. Local Variables

-int du\_start, du\_end : 출력을 시작할 주소 값과 끝낼 주소 값

## 모듈이름: mem\_dump\_first\_only() (cmd\_mem\_ctr.c)

### 기능

입력 받은 시작 주소부터 10줄까지 할당되어 있는 값을 출력한다.

### 사용변수

1. Parameters

-int du\_start : 시작 주소

-int \*cur\_mem\_ctr : 다시 du[mp]를 실행했을 때의 시작 주소 값을 저장하는 변수

1. Local Variables

-int du\_end : 끝 주소. 만약 시작 주소부터 10줄(160byte)이 0xFFFFF을

넘으면 du\_end==0xFFFFF이다.

## 모듈이름: mem\_dump\_first\_last() (cmd\_mem\_ctr.c)

### 기능

입력 받은 시작 주소부터 끝 주소까지 할당되어 있는 값들을 출력한다.

### 사용변수

1. Parameters

-int du\_start, du\_end : 시작 주소와 끝 주소

-int \*cur\_mem\_ctr : 다시 du[mp]를 실행했을 때의 시작 주소 값을 저장하는 변수

1. Local Variables

-int i,j : 반복문을 실행시키기 위한 변수

-int start\_line, end\_line : 시작 주소의 Memory index와 끝 주소의 Memory index

-int cur\_line : 현재 Line의 시작 주소 값

-int line\_bound, line\_bound\_end : 시작 줄과 끝 줄의 시작 지점과 끝 지점

## 모듈이름: mem\_edit() (cmd\_mem\_ctr.c)

### 기능

입력 받은 Memory의 주소의 ASCII Code 값을 입력 받은 값으로 변경한다.

### 사용변수

1. Parameters

-int edit\_mem : 값을 변경할 Memory 주소

-int edit\_val : 변경할 ASCII Code 값

1. Local Variables

-int edit\_row, edit\_col : Memory 배열의 row, column

## 모듈이름: mem\_fill() (cmd\_mem\_ctr.c)

### 기능

입력 받은 시작 주소부터 끝 주소까지 입력 받은 특정한 ASCII Code 값으로 변경한다.

### 사용변수

1. Parameters

- int fill\_start : Memory 시작 주소

- int fill\_end : Memory 끝 주소

- int fill\_val : 변경할 ASCII Code 값

1. Local Variables

- int i,j : 반복문을 실행시키기 위한 변수

- int start\_line, end\_line, line\_bound : Memory 시작 row, 끝 row, 시작 row와 끝 row의 분기점

## 모듈이름: mem\_reset() (cmd\_mem\_ctr.c)

### 기능

Memory 전체를 전부 0으로 변경시킨다. 즉, 프로그램 시작 시의 상태로 되돌려놓는다.

### 사용변수

int i, j : 반복문을 실행시키기 위한 변수

## 모듈이름: asmCreate() (asm\_lst\_create.c)

### 기능

입력 받은 .asm 파일을 찾고 Assemble한다. asmFirstPass(), asmSecondPass(), objFileMaking()함수를 포함한다.

### 사용변수

int start\_locctr, int end\_locctr : 시작 주소, 끝 주소를 저장하는 변수

int res\_flag : Pass 1, Pass 2, Object File Making의 정상 실행 여부를 저장하는 변수

int nobase\_flag : NOBASE directive 사용 여부

## 모듈이름: asmFirstPass() (asm\_lst\_create.c)

### 기능

Assembler의 Pass 1 Algorithm에 따라, Symbol Table을 생성한다.

### 사용변수

1. Parameter

-char \*asm\_file\_name : .asm 파일 이름

1. Local Variable

-FILE \*asm\_fp : .asm 파일 포인터

-int cur\_locctr : 현재 Location Counter

-int cur\_line : 현재 Line number

-int cur\_fmt : mnemonic Format 저장하는 변수

-char asm\_reader[129] : .asm 파일을 읽는 char 배열

-char \*sym, \*mnemonic, \*operand\_1 : Symbol, Mnemonic, Operand를 저장

-const char \*asm\_token : Tokenizing을 위한 배열

-int byte\_len : BYTE directive를 위한 변수

## 모듈이름: asmSecondPass() (asm\_lst\_create.c)

### 기능

Assembler의 Pass 2 Algorithm에 따라 .lst 파일을 생성한다. ERROR가 있을 시 현재 Line을 return하여 예외처리한다.

### 사용변수

1. Parameter

-char \*asm\_file\_name : .asm 파일 이름

-int \*start\_locctr : 시작 주소

-int \*end\_locctr : 끝 주소

1. Local Variable

-int cur\_line : 현재 Line

-int cur\_locctr : 현재 LOCCTR

-int base\_num : BASE number

-int cur\_fmt : 현재 Format

-int byte\_len : BYTE directive에 사용되는 변수

-int obj\_code : Object Code

-int opcode\_num : Mnemonic의 opcode

## 모듈이름: opSearch() (asm\_symbol\_ctr.c)

### 기능

Parameter로 받은 mnemonic의 Format number를 Opcode Hash Table에서 찾고, Format number를 return한다.

### 사용변수

1. Parameter

- char \*op\_str : Mnemonic

2) Local Variables

- int op\_idx : Mnemonic의 index number

- opcode\_ptr search\_op : Opcode Hash Table을 Search하기 위한 변수

## 모듈이름: opcodeNumberSearch() (asm\_symbol\_ctr.c)

### 기능

Parameter로 받은 mnemonic의 Opcode number를 Opcode Hash Table에서 찾고, opcode number를 return한다.

### 사용변수

1. Parameter

- char \*op\_str : Mnemonic

2) Local Variables

- int op\_idx : Mnemonic의 index number

- opcode\_ptr search\_op : Opcode Hash Table을 Search하기 위한 변수

## 모듈이름: objForamtTwo() (asm\_obj\_format.c)

### 기능

Format 2의 Object Code를 생성한다.

### 사용변수

1. Parameter

- int opcode\_num : Mnemonic의 Opcode number

- char \*operand\_1 : operand 1 (register 1 or n)

- char \*operand\_2 : operand 2 (register 2 or n)

2) Local Variables

- int obj : Format 2 의 Object Code

## 모듈이름: objFormatThree() (asm\_obj\_format.c)

### 기능

Format 3의 Object Code를 생성한다. 만약 PC relative, BASE relative 모두 사용이 불가능하면 Format 4로 Object Code를 생성한다.

### 사용변수

1. Parameter

- int opcode\_num : Mnemonic의 opcode number

- char \*operand\_1 : operand 1 (symbol)

- char \*operand\_2 : operand 2 (index)

- int cur\_locctr : 현재 LOCCTR

- int base\_num : BASE number

2) Local Variables

- int obj : Object Code

- int const\_flag : 상수/Symbol 구분

- int sym\_locctr : Symbol의 LOCCTR

- int disp : Displacement를 저장하는 변수

- int cur\_pc : 현재 Program Counter 위치

## 모듈이름: objFormatFour() (asm\_obj\_format.c)

### 기능

Format 4 의 Object Code를 생성한다..

### 사용변수

1. Parameter

- int opcode\_num : Mnemonic의 opcode number

- char \*operand\_1 : operand 1 (symbol)

- char \*operand\_2 : operand 2 (index)

- int cur\_locctr : 현재 LOCCTR

2) Local Variables

- int obj : Object Code

- int const\_flag : 상수/Symbol 구분

- int sym\_locctr : Symbol의 LOCCTR

## 모듈이름: objFileMaking() (asm\_obj\_create.c)

### 기능

.lst 파일을 읽어서 .obj 파일을 생성한다.

### 사용변수

1. Parameter

- char \*lst\_file\_name : .lst 파일 이름

- int locctr\_start : 시작 location counter

- int locctr\_end : location counter 끝

2) Local Variables

- char obj\_line[62] : .obj 파일의 한 줄

- const char \*obj\_token : Tokenizing을 위한 문자열

- int first\_locctr : 시작 location counter

- int loc\_plus : 다음 location counter

- int \*mod\_record : Modification해야 하는 위치를 저장하는 int 배열

- int byte\_len : BYTE directive에 사용되는 변수

## 모듈이름: symbolTableSearch() (asm\_symbol\_ctr.c)

### 기능

Parameter로 넘겨받은 symbol을 Symbol Table에서 찾아서, Symbol의 Location Counter를 return한다.

### 사용변수

1. Parameter

-char \*sym\_str : Symbol의 이름

2) Local Variable

-symbol\_ptr sy\_search : Symbol Table을 search하기 위한 pointer

-int table\_idx : Symbol의 Index (Symbol의 첫 글자)

## 모듈이름: symbolTableInsert() (asm\_symbol\_ctr.c)

### 기능

Symbol Table에 새 symbol을 내림차순으로 저장한다.

### 사용변수

1. Parameter

-int sym\_locctr : Symbol의 LOCCTR

-char \*sym\_str : Symbol의 이름

2) Local Variable

-symbol\_ptr added\_symbol : Symbol Table에 새로 추가되는 Node

-symbol\_ptr search\_front, search\_rear: Symbol Table을 search하기 위한 pointer

-int table\_idx : Symbol의 Index (Symbol의 첫 글자)

## 모듈이름: freeSymbolTable() (asm\_symbol\_ctr.c)

### 기능

Symbol Table을 Deallocation한다.

### 사용변수

symbol\_ptr temp\_node, deall\_node : Symbol Table을 Deallocation하기 위한 변수

## 모듈이름: printSymbolTable() (cmd\_mem\_ctr.c)

### 기능

Symbol Table에 저장된 Symbol들을 내림차순으로 출력한다.

### 사용변수

int i : 반복문을 실행시키기 위한 변수

symbol\_ptr print\_node : Symbol Table을 search하기 위한 변수

## 모듈이름: sicType() (cmd\_execution.c)

### 기능

입력 받은 파일의 내용을 출력한다.

### 사용변수

1. Parameter

-char \*type\_fname : 출력할 file의 이름

1. Local Variable

-char res : 출력할 문자

## 모듈이름: printBreakPoint() (linkload\_load\_bp.c)

### 기능

현재 저장되어 있는 Breakpoint들을 오름차순으로 출력한다.

### 사용변수

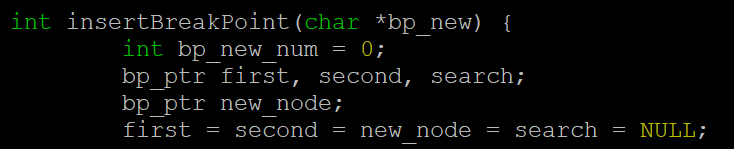
bp\_ptr iter\_bp : Breakpoint list를 탐색하기 위한 변수

## 모듈이름: insertBreakPoint() (linkload\_load\_bp.c)

### 기능

입력 받은 새로운 Breakpoint를 Breakpoint list에 삽입한다.

### 사용변수



## 모듈이름: clearBreakPoint() (linkload\_load\_bp.c)

### 기능

모든 Breakpoint를 삭제한다.

### 사용변수

bp\_ptr deal\_bp : Deallocate할 node를 가리키는 변수

## 모듈이름: searchBreakPoint() (linkload\_load\_bp.c)

### 기능

현재 Breakpoint를 bp\_idx 값에 따라return한다.

### 사용변수

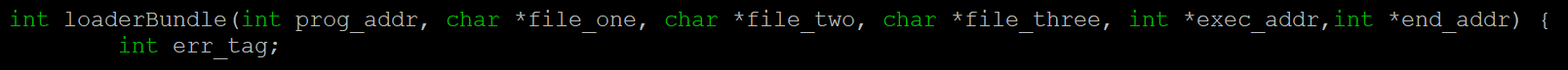
bp\_ptr search\_bp : 현재 Breakpoint

## 모듈이름: loaderBundle() (linkload\_load\_bp.c)

### 기능

Loader Pass 1, Pass 2함수를 실행한다.

### 사용변수

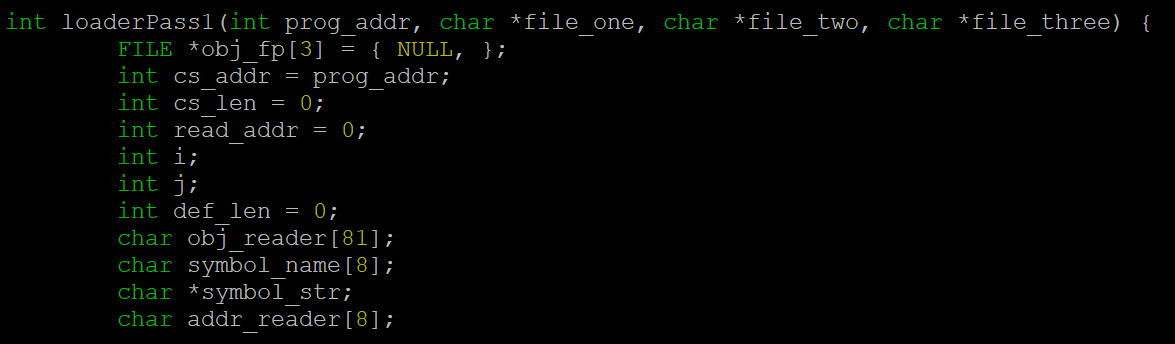


## 모듈이름: loaderPass1() (linkload\_load\_bp.c)

### 기능

Loader 알고리즘의 Pass 1을 실행한다. ERROR가 발생할 경우 Error Message를 출력한다.

### 사용변수

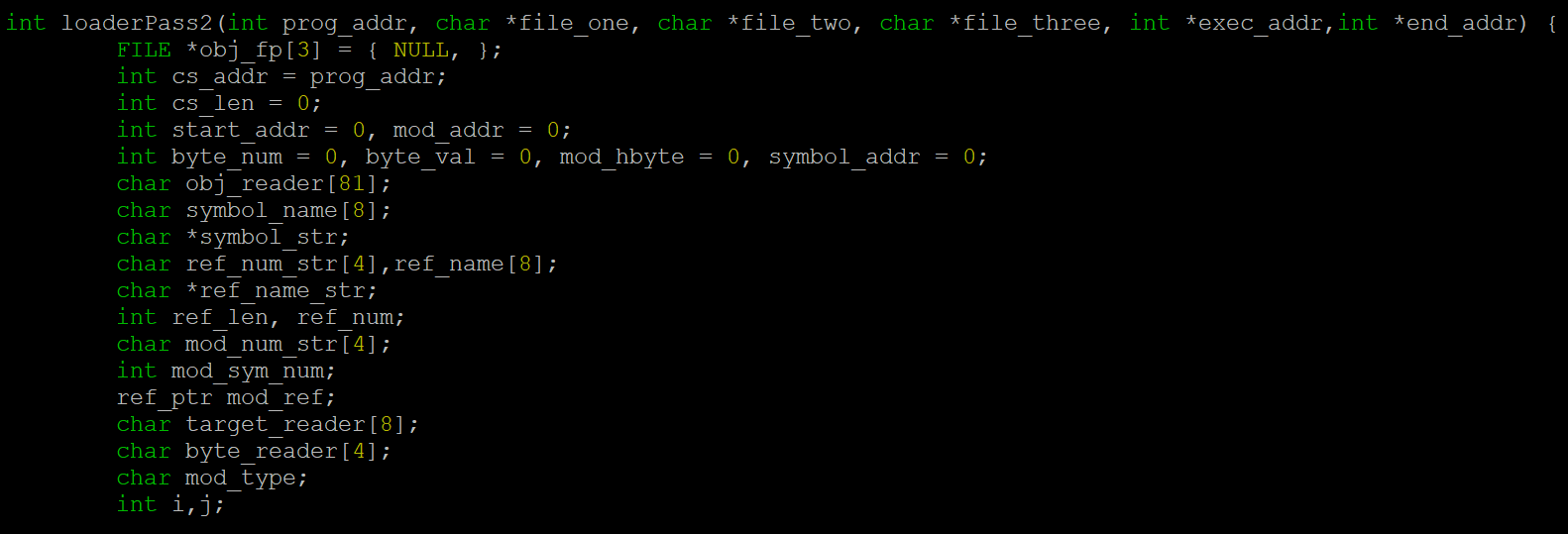


## 모듈이름: loaderPass2() (linkload\_load\_bp.c)

### 기능

Loader 알고리즘의 Pass 2를 실행한다. ERROR가 발생할 경우 Error Message를 출력한다.

### 사용변수

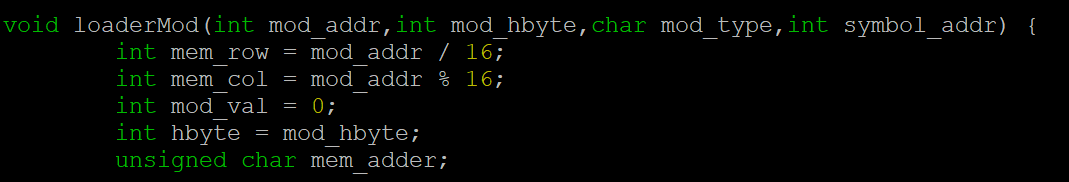


## 모듈이름: loaderMod() (linkload\_load\_bp.c)

### 기능

Loader Pass 2에서, Modification Record의 내용에 맞게 Memory를 Modification한다.

### 사용변수

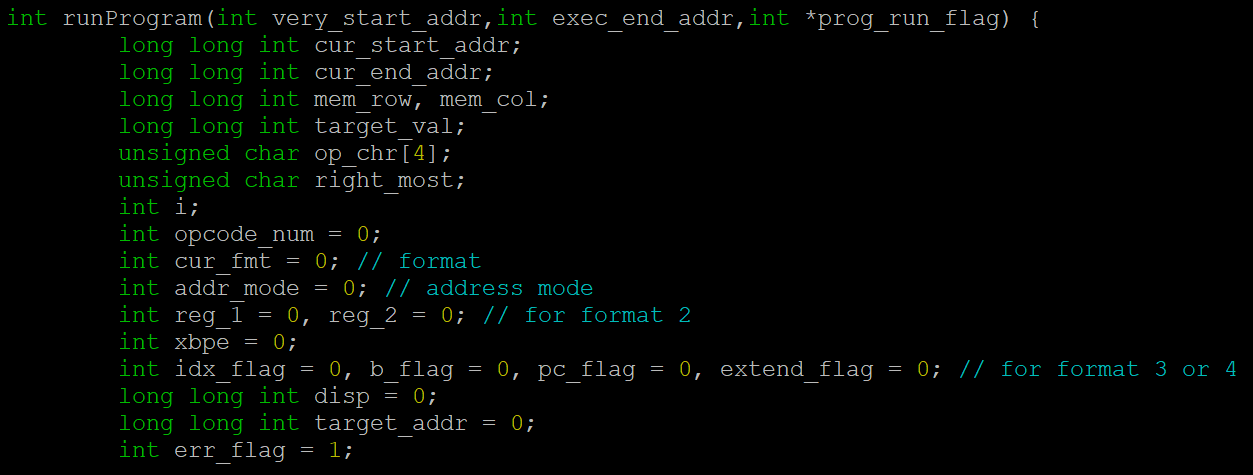


## 모듈이름: runProgram() (linkload\_run..c)

### 기능

Memory에 Load된 프로그램을 실행한다.

### 사용변수



## 모듈이름: saveRegHistory() (linkload\_run.c)

### 기능

실행 전의 Register들의 값을 저장한다.

### 사용변수

## 모듈이름: opcodeFormat() (linkload\_run.c)

### 기능

Opcode number를 Parameter로 받아서, 해당 Opcode의 Format을 Return한다.

### 사용변수

int opcode\_num : opcode number

## 모듈이름: formatTwoCheck() (linkload\_run.c)

### 기능

Format 2에서, 유효한 Register Number가 들어왔는지 Check한다.

### 사용변수

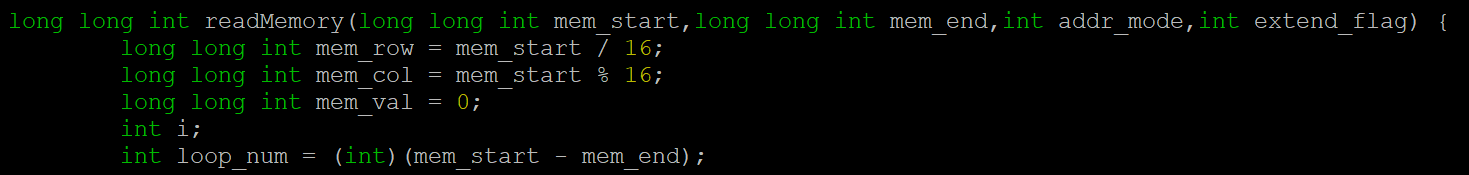
int reg\_1, int reg\_2 : 입력 받은 Register 번호

## 모듈이름: readMemory() (linkload\_run.c)

### 기능

Load하는 Opcode일 경우, Register에 Memory에 있는 값을 LOAD한다.

### 사용변수

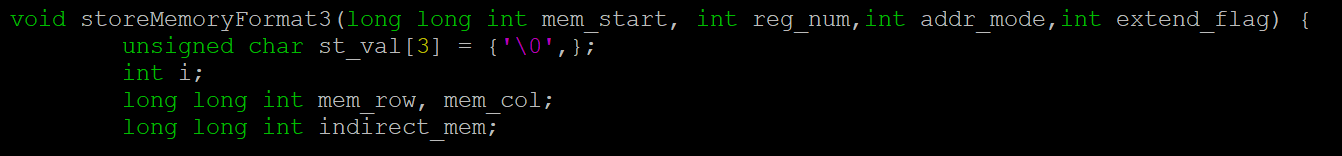


## 모듈이름: storeMemoryFormat3() (linkload\_run.c)

### 기능

Memory에 Register 값을 Store한다.

### 사용변수

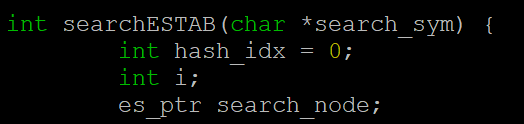


## 모듈이름: searchESTAB() (linkload\_load\_bp.c)

### 기능

Parameter로 받은 symbol을 External Symbol Table에서 찾고, Table에 있으면 Address를 Return한다.

### 사용변수

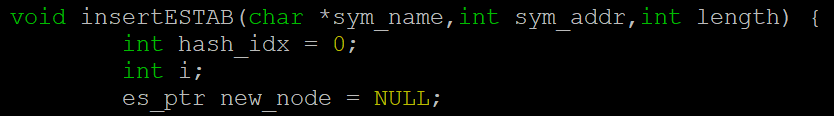


## 모듈이름: insertESTAB() (linkload\_load\_bp.c)

### 기능

External Symbol Table에 새로운 Symbol을 Insert한다.

### 사용변수



## 모듈이름: deallESTAB() (linkload\_load\_bp.c)

### 기능

External Symbol Table을 Deallocate한다.

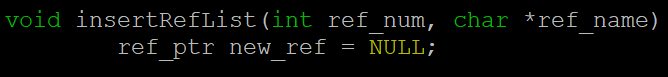
### 사용변수

## 모듈이름: printESTAB() (linkload\_load\_bp.c)

### 기능

External Symbol Table을 Print한다.

### 사용변수

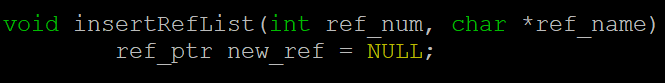


## 모듈이름: insertRefList() (linkload\_load\_bp.c)

### 기능

Reference Symbol List에 새로운 External Reference Symbol을 삽입한다.

### 사용변수



## 모듈이름: searchRefList() (linkload\_load\_bp.c)

### 기능

Symbol Number를 Parameter로 받아서, Reference Symbol List에서, External Reference Symbol을 Search한다.

### 사용변수



## 모듈이름: deallRefList() (linkload\_load\_bp.c)

### 기능

Reference Symbol List를 Deallocate한다.

### 사용변수

# 전역 변수 정의

## #define Macros

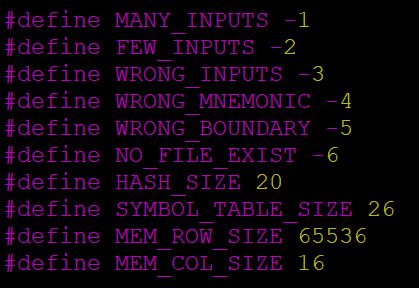


Figure 3. #define Macros

* MANY\_INPUTS ~ WRONG\_BOUNDARY : err\_msg\_bundle() 에서 쓰인다.
* HASH\_SIZE : Opcode Hash Table의 Size이다.
* MEM\_ROW\_SIZE, MEM\_COL\_SIZE : Memory Size
* SYMBOL\_TABLE\_SIZE : Symbol Table의 Size 이다.

## history\_ptr his\_list\_front, his\_list\_rear

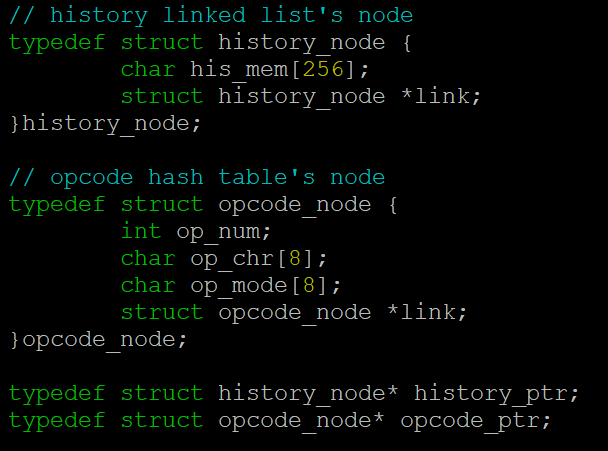


Figure 4. history\_node, opcode\_node

* history\_ptr his\_list\_front : History List의 가장 앞의 주소를 가리키는 pointer 변수
* history\_ptr his\_list\_rear : History List의 가장 뒤쪽 주소를 가리키는 pointer 변수

## opcode\_ptr opcode\_table[HASH\_SIZE]

Opcode Hash Table 변수이다. 이 변수에 Opcode Hash Table node가 저장된다.

## char mem\_arr[MEM\_ROW\_SIZE][MEM\_COL\_SIZE]

SIC / XE Machine의 Memory 전역 변수이다.

## symbol\_ptr last\_symbol\_table[SYMBOL\_TABLE\_SIZE]

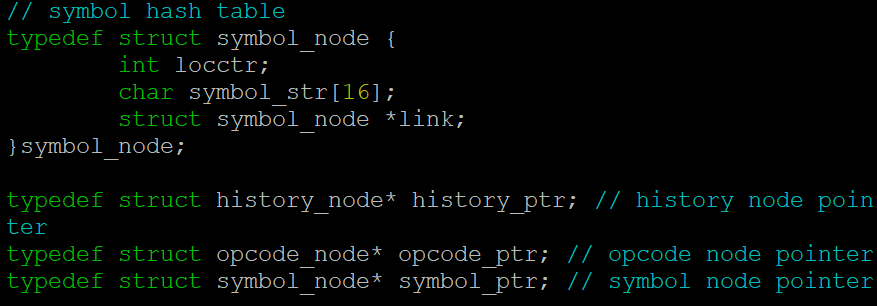


Figure 5. symbol\_ptr definition

Symbol을 저장하는 Table이다.

# 코드

/\* 20121578.h \*/

#include <stdio.h>

#include <stdio\_ext.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <dirent.h>

#include <sys/stat.h>

#define MANY\_INPUTS -1

#define FEW\_INPUTS -2

#define WRONG\_INPUTS -3

#define WRONG\_MNEMONIC -4

#define WRONG\_BOUNDARY -5

#define NO\_FILE\_EXIST -6

#define HASH\_SIZE 20

#define SYMBOL\_TABLE\_SIZE 26

#define MEM\_ROW\_SIZE 65536

#define MEM\_COL\_SIZE 16

// history linked list's node

typedef struct history\_node {

char his\_mem[256];

struct history\_node \*link;

}history\_node;

// opcode hash table's node

typedef struct opcode\_node {

int op\_num;

char op\_chr[8];

char op\_mode[8];

struct opcode\_node \*link;

}opcode\_node;

// symbol hash table

typedef struct symbol\_node {

int locctr;

char symbol\_str[16];

struct symbol\_node \*link;

}symbol\_node;

typedef struct history\_node\* history\_ptr; // history node pointer

typedef struct opcode\_node\* opcode\_ptr; // opcode node pointer

typedef struct symbol\_node\* symbol\_ptr; // symbol node pointer

// global variables (history, opcode hash table)

history\_ptr his\_list\_front;

history\_ptr his\_list\_rear;

opcode\_ptr opcode\_table[HASH\_SIZE];

// global variables (memory)

char mem\_arr[MEM\_ROW\_SIZE][MEM\_COL\_SIZE];

// global variables (symbol table)

symbol\_ptr last\_symbol\_table[SYMBOL\_TABLE\_SIZE];

//symbol\_ptr temp\_symbol\_front;

// This functions are used for the manipulation of list, table and memory.

void opcode\_init();

void opcode\_table\_print();

int opcode\_find(char\*,int);

void opcode\_free();

void history\_add(char\*);

void history\_free();

void mem\_reset();

void mem\_dump\_only(int\*);

void mem\_dump\_first\_only(int,int\*);

void mem\_dump\_first\_last(int,int,int\*);

void mem\_edit(int,int);

void mem\_fill(int,int,int);

int mem\_hexa\_convert(char\*);

// This functions are used for the main menu of SIC.

void sic\_menu();

int sic\_menu\_classify(char\*\*,char\*,int\*);

char \*trim\_left(char\*);

char \*trim\_right(char\*);

// This functions are used for the execution of command.

void sic\_help();

void sic\_history\_print();

int sic\_dir();

// This functions are used for project 2, creating .lst file and .obj file.

int sicType(char\*);

int asmCreate(char\*);

int asmFirstPass(char\*);

int asmSecondPass(char\*,int\*,int\*,int\*);

int opSearch(char\*);

// for creating objcode for .lst file

int objFormatTwo(int,char\*,char\*);

int objFormatThree(int,char\*,char\*,int,int,int);

int objFormatFour(int,char\*,char\*,int);

int opcodeNumberSearch(char\*);

int objFileMaking(char\*,int,int);

// This functions are used for project 2, used for symbol.

void freeSymbolTable();

void printSymbolTable();

int symbolTableSearch(char\*);

void symbolTableInsert(int, char\*);

// This function is used for the notice of error.

void err\_msg\_bundle(int);

/\* 20121578.c \*/

#include "20121578.h"

int main() {

opcode\_init(); // initialize opcode hash table

sic\_menu(); // main menu : sicsim>

opcode\_free(); // deallocate opcode hash table

history\_free(); // deallocate hash table list

freeSymbolTable(); // deallocate symbol table list

return 0;

}

void err\_msg\_bundle(int err\_state) {

/\*

This function contains various types of error messages in error cases.

\*/

if (err\_state == FEW\_INPUTS) {

printf("ERROR : Not enough inputs\n");

return;

}

if (err\_state == MANY\_INPUTS) {

printf("ERROR : Too many inputs\n");

return;

}

if (err\_state == WRONG\_INPUTS) {

printf("ERROR : Wrong input. Please input correctly.\n");

return;

}

if (err\_state == WRONG\_MNEMONIC) {

printf("ERROR : Wrong mnemonic. Please input valid mnemonic.\n");

return;

}

if (err\_state == WRONG\_BOUNDARY) {

printf("ERROR : Wrong Boundary. Please input valid boundary.\n");

return;

}

}

/\* cmd\_ctr.c \*/

#include "20121578.h"

char\* trim\_left(char \*tok\_num) {

while (isspace(\*tok\_num)) {

tok\_num++;

}

return tok\_num;

}

// erase the left-sided space that include spacebar and tap.

char\* trim\_right(char \*tok\_num) {

char \*end;

end = tok\_num + strlen(tok\_num) - 1;

while (end > tok\_num && isspace(\*end)) {

end--;

}

\*(end + 1) = 0;

return tok\_num;

} // erase the right-sided space that include spacebar and tap.

void sic\_menu() {

char input\_str[256], history\_str[256], copy\_str[256];

char \*check\_str;

char \*input\_arr[5], \*check\_arr[5]; // input : used for menu. check : used for checking input error

const char \*cmd\_tok = " \t\n"; // tokens for the first tokenize

const char \*cmd\_comma\_tok = " ,\t\n"; // tokens for du[mp], e[dit], f[ill]

const char \*comma\_only\_tok = ","; // tokens for the comma check

int i,cmd\_num,exec\_res,input\_check\_num;

int cur\_mem\_ctr = 0;

/\* exec\_res : used for distinguishing the state of result

cmd\_num : the number of tokens

input\_check\_run : integer that used for checking the error of input \*/

while (1) {

for (i = 0; i < 5;i++) {

input\_arr[i] = check\_arr[i] = NULL;

}

\_\_fpurge(stdin); // clean the buffer

check\_str = NULL;

cmd\_num = 0;

/\* initialization \*/

printf("sicsim> ");

fgets(input\_str,256,stdin); // input

input\_str[strlen(input\_str) - 1] = '\0';

strcpy(copy\_str, input\_str);

strcpy(history\_str,input\_str);

input\_arr[cmd\_num++] = strtok(input\_str, cmd\_tok);

//input\_arr[cmd\_num++] = strtok(NULL, cmd\_tok);

while (cmd\_num < 5) {

input\_arr[cmd\_num] = strtok(NULL, cmd\_comma\_tok);

if (input\_arr[cmd\_num] == NULL) {

break;

}

cmd\_num++;

}

// tokenize the command

// input error checking part (start)

if (input\_arr[0] == NULL) {

continue;

} // no input

else if (input\_arr[4] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

continue;

} // too many inputs

else {

check\_str = copy\_str;

check\_str = trim\_left(check\_str);

check\_str = trim\_right(check\_str);

input\_check\_num = strlen(check\_str);

if (check\_str[input\_check\_num - 1] == ',') {

err\_msg\_bundle(WRONG\_INPUTS);

input\_check\_num = -1;

}

// wrong token (last letter is comma)

if (input\_check\_num == -1) {

continue;

}

// recieve input again

if (!strcmp(input\_arr[0], "du") || !strcmp(input\_arr[0], "dump") || !strcmp(input\_arr[0], "e") || !strcmp(input\_arr[0], "edit") || !strcmp(input\_arr[0], "f") || !strcmp(input\_arr[0], "fill")

|| !strcmp(input\_arr[0],"opcode") || !strcmp(input\_arr[0],"assemble") || !strcmp(input\_arr[0],"type")) {

if (input\_arr[1] != NULL) {

cmd\_num = 0;

check\_arr[cmd\_num++] = strtok(check\_str, cmd\_tok);

while (cmd\_num < 5) {

check\_arr[cmd\_num] = strtok(NULL, comma\_only\_tok);

if (check\_arr[cmd\_num] == NULL) {

break;

}

check\_arr[cmd\_num] = trim\_left(check\_arr[cmd\_num]);

check\_arr[cmd\_num] = trim\_right(check\_arr[cmd\_num]);

cmd\_num++;

} // tokenize by comma

for(i=input\_check\_num-1;i>=0;i--){

if(check\_str[i] == ','){

err\_msg\_bundle(WRONG\_INPUTS);

input\_check\_num = -1;

break;

}

}

if(input\_check\_num == -1){

continue;

} // if there are many commas between commands

for (i = 0; i < 5; i++) {

if (input\_arr[i] != NULL && check\_arr[i] != NULL) {

if (strcmp(input\_arr[i], check\_arr[i])) {

err\_msg\_bundle(WRONG\_INPUTS);

input\_check\_num = -1;

break;

}

}

else if (input\_arr[i] == NULL && check\_arr[i] == NULL) {

break;

}

else {

err\_msg\_bundle(WRONG\_INPUTS);

input\_check\_num = -1;

break;

}

} // checking if there is comma between the memory number

if (input\_check\_num == -1) {

continue;

} // if there is no comma between the memory number

}

} // du[mp], e[dit], f[ill] : comma checking

} // checking

// input error checking part (end)

exec\_res = sic\_menu\_classify(input\_arr,history\_str,&cur\_mem\_ctr); // execution of command

if (exec\_res == -1 || exec\_res == 2) {

continue;

} // error or hi[story] command

else if (exec\_res == 0) {

break;

} // quit

else {

history\_add(history\_str);

} // save the history (need to add)

/\*

exec\_res == -1 : error

exec\_res == 0 : quit

exec\_res == 1 : executed normally

exec\_res == 2 : hi[story] executed normally

\*/

}

}

int sic\_menu\_classify(char\*\* input\_arr,char\* history\_str, int \*cur\_mem\_ctr) {

/\*

return -1 : error

return 0 : only quit

return 1 : executed normally

return 2 : hi[story] executed normally

\*/

int du\_start, du\_end; // dump start addr, dump end addr

int edit\_addr, edit\_val; // address that edited and value

int fill\_start, fill\_end, fill\_val; // fill start addr, fill end addr, value

du\_start = du\_end = edit\_addr = edit\_val = 0;

fill\_start = fill\_end = fill\_val = 0;

if (!strcmp(input\_arr[0], "h") || !strcmp(input\_arr[0], "help")) {

if (input\_arr[1] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

sic\_help();

return 1; // executed normally

}

else if (!strcmp(input\_arr[0], "d") || !strcmp(input\_arr[0], "dir")) {

if (input\_arr[1] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

return sic\_dir(); // execute the function

}

else if (!strcmp(input\_arr[0], "q") || !strcmp(input\_arr[0], "quit")) {

if (input\_arr[1] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

return 0; // quit : return 0

}

else if (!strcmp(input\_arr[0], "hi") || !strcmp(input\_arr[0], "history")) {

if (input\_arr[1] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

history\_add(history\_str);

// hi[story] is also command so it must be added in history list.

sic\_history\_print();

// print history list

return 2;

}

else if (!strcmp(input\_arr[0], "du") || !strcmp(input\_arr[0], "dump")) {

if (input\_arr[3] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

if (input\_arr[1] == NULL) {

mem\_dump\_only(cur\_mem\_ctr);

} // dump (only)

else {

du\_start = mem\_hexa\_convert(input\_arr[1]);

// convert dump start addr string to integer

if (du\_start == -1) {

return -1;

} // if there is wrong addr

else if (du\_start > 0xfffff) {

err\_msg\_bundle(WRONG\_BOUNDARY);

return -1;

}

// boundary checking

if (input\_arr[2] == NULL) {

mem\_dump\_first\_only(du\_start,cur\_mem\_ctr);

} // dump (start only)

else {

du\_end = mem\_hexa\_convert(input\_arr[2]);

// convert dump end addr string to integer

if (du\_end == -1) {

return -1;

} // if there is wrong addr

if (du\_start > du\_end || du\_end > 0xfffff || du\_start > 0xfffff) {

err\_msg\_bundle(WRONG\_BOUNDARY);

return -1;

} // if there is wrong boundary

mem\_dump\_first\_last(du\_start, du\_end, cur\_mem\_ctr);

} // dump (start, end)

} // dump

return 1;

// need to add

}

else if (!strcmp(input\_arr[0],"e") || !strcmp(input\_arr[0],"edit")) {

if (input\_arr[3] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

else if (input\_arr[2] == NULL) {

err\_msg\_bundle(FEW\_INPUTS);

return -1;

}

edit\_addr = mem\_hexa\_convert(input\_arr[1]);

// convert edit addr string to integer

if (edit\_addr == -1) {

return -1;

} // if there is wrong addr

else if (edit\_addr > 0xfffff) {

err\_msg\_bundle(WRONG\_BOUNDARY);

return -1;

} // if there is wrong boundary

edit\_val = mem\_hexa\_convert(input\_arr[2]);

// convert edit val string to integer (ASCII code number)

if (edit\_val == -1) {

return -1;

}

// if there is wrong input

else if (edit\_val > 0x7f) {

err\_msg\_bundle(WRONG\_BOUNDARY);

return -1;

}

// if there is wrong ASCII code

mem\_edit(edit\_addr, edit\_val);

// edit the memory

return 1;

}

else if (!strcmp(input\_arr[0],"f") || !strcmp(input\_arr[0],"fill")) {

if (input\_arr[3] == NULL) {

err\_msg\_bundle(FEW\_INPUTS);

return -1;

}

/\*

convert fill start addr and input, boundary error checking

\*/

fill\_start = mem\_hexa\_convert(input\_arr[1]);

if (fill\_start == -1) {

return -1;

}

else if (fill\_start > 0xfffff) {

err\_msg\_bundle(WRONG\_BOUNDARY);

return -1;

}

/\*

convert fill end addr and input, boundary error checking

\*/

fill\_end = mem\_hexa\_convert(input\_arr[2]);

if (fill\_end == -1) {

return -1;

}

else if (fill\_end > 0xfffff) {

err\_msg\_bundle(WRONG\_BOUNDARY);

return -1;

}

/\*

boundary checking if end addr is lower than start addr

\*/

if (fill\_start > fill\_end) {

err\_msg\_bundle(WRONG\_BOUNDARY);

return -1;

}

/\*

convert fill value and input error checking

\*/

fill\_val = mem\_hexa\_convert(input\_arr[3]);

if (fill\_val == -1) {

return -1;

}

else if (fill\_val > 0x7f) {

err\_msg\_bundle(WRONG\_BOUNDARY);

return -1;

}

mem\_fill(fill\_start, fill\_end, fill\_val);

// f[ill]

return 1;

}

else if (!strcmp(input\_arr[0],"reset")) {

if (input\_arr[1] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

mem\_reset(); // reset the memory

return 1;

}

else if (!strcmp(input\_arr[0],"opcode")) {

if (input\_arr[1] == NULL) {

err\_msg\_bundle(FEW\_INPUTS);

return -1;

}

else if (input\_arr[2] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

if (opcode\_find(input\_arr[1],strlen(input\_arr[1])) == 0) {

return 1;

} // if opcode is in hash table, opcode\_find returns 0.

else {

return -1;

} // else

}

else if (!strcmp(input\_arr[0],"opcodelist")) {

if (input\_arr[1] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

opcode\_table\_print(); // print opcode hash table

return 1;

}

else if (!strcmp(input\_arr[0],"assemble")) {

if (input\_arr[1] == NULL) {

err\_msg\_bundle(FEW\_INPUTS);

return -1;

}

else if (input\_arr[2] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

return asmCreate(input\_arr[1]);

// execute asmCreate function

} // command 'assemble'

else if (!strcmp(input\_arr[0],"type")) {

if (input\_arr[1] == NULL) {

err\_msg\_bundle(FEW\_INPUTS);

return -1;

}

else if (input\_arr[2] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

return sicType(input\_arr[1]);

} // command 'type'

else if (!strcmp(input\_arr[0],"symbol")) {

if (input\_arr[1] != NULL) {

err\_msg\_bundle(MANY\_INPUTS);

return -1;

}

printSymbolTable(); // print symbol table.

return 1;

} // command 'symbol'

else {

err\_msg\_bundle(WRONG\_INPUTS);

return -1;

} // the others

}

/\* cmd\_execution.c \*/

#include "20121578.h"

void sic\_help() {

printf("\nh[elp]\n");

printf("d[ir]\n");

printf("q[uit]\n");

printf("hi[story]\n");

printf("du[mp] [start, end]\n");

printf("e[dit] address, value\n");

printf("f[ill] start, end, value\n");

printf("reset\n");

printf("opcode mnemonic\n");

printf("opcodelist\n");

printf("assemble filename\n");

printf("type filename\n");

printf("symbol\n\n");

} // h[elp]

void sic\_history\_print() {

int his\_num = 1;

history\_ptr temp\_his = his\_list\_front;

while (temp\_his != NULL) {

printf("%d %s\n",his\_num,temp\_his->his\_mem);

his\_num++;

temp\_his = temp\_his->link;

} // print command history list

}

void history\_add(char \*history\_input) {

history\_ptr new\_history = NULL;

new\_history = (history\_ptr)malloc(sizeof(history\_node));

if (new\_history == NULL) {

printf("Memory Allocation Error!\n");

exit(-1);

}

strcpy(new\_history->his\_mem,history\_input);

new\_history->link = NULL;

// make new history node

if (his\_list\_front == NULL) {

his\_list\_front = his\_list\_rear = new\_history;

} // first history

else {

his\_list\_rear->link = new\_history;

his\_list\_rear = new\_history;

} // the others

}

int sicType(char \*type\_fname) {

FILE \*type\_fp = NULL;

char res;

type\_fp = fopen(type\_fname,"r");

if (type\_fp == NULL) {

err\_msg\_bundle(NO\_FILE\_EXIST);

return -1;

}

while (1) {

res = fgetc(type\_fp);

if (res == EOF) {

break;

}

printf("%c",res);

}

return 1;

}

int sic\_dir() {

DIR \*cur\_dir = NULL; // read the directory's name

struct dirent \*f\_point = NULL;

struct stat s\_buf; // save the directory's status

cur\_dir = opendir("."); // open current directory

if(!cur\_dir){

printf("ERROR : Cannot Open Current Directory\n");

return -1;

}

while((f\_point = readdir(cur\_dir)) != NULL){ // read the directory

lstat(f\_point->d\_name,&s\_buf); // read the status of directory

if(S\_ISDIR(s\_buf.st\_mode)){

printf("%s/\t",f\_point->d\_name);

} // S\_ISDIR : the function that check if status is directory

else if((S\_IEXEC & s\_buf.st\_mode) != 0){

printf("%s\*\t",f\_point->d\_name);

} // S\_ISEXEC : constant that check if status is execute file

else{

printf("%s\t",f\_point->d\_name);

} // others

}

printf("\n");

closedir(cur\_dir); // close the current directory

return 1;

}

/\* cmd\_mem\_ctr.c \*/

#include "20121578.h"

void mem\_reset() {

int i, j;

for (i = 0; i < MEM\_ROW\_SIZE; i++) {

for (j = 0; j < MEM\_COL\_SIZE; j++) {

mem\_arr[i][j] = 0;

}

}

} // reset the memory

int mem\_hexa\_convert(char \*num\_str) {

int i,j;

int num\_len = strlen(num\_str);

int cur\_idx = 0;

int num\_res = 0;

int hexa\_start = 1;

for (i = num\_len - 1; i >= 0;i--) {

if(cur\_idx < 5){

if (num\_str[i] >= 48 && num\_str[i] <= 57) {

num\_res += hexa\_start \* (num\_str[i] - 48);

} // char 0~9

else if (num\_str[i] >= 65 && num\_str[i] <= 70) {

num\_res += hexa\_start \* (num\_str[i] - 55);

} // char A~F

else if (num\_str[i] >= 97 && num\_str[i] <= 102) {

num\_res += hexa\_start \* (num\_str[i] - 87);

} // char a~f

else {

err\_msg\_bundle(WRONG\_INPUTS);

return -1;

} // the others

hexa\_start \*= 16;

cur\_idx++;

}

else{

for(j=0; j < num\_len - cur\_idx ; j++){

if(num\_str[j] != '0'){

err\_msg\_bundle(WRONG\_BOUNDARY);

return -1;

}

}

// for the case like 0000000000000000000fff

}

}

return num\_res; // return converted address or ASCII code

}

void mem\_dump\_only(int \*cur\_mem\_ctr) {

int du\_start, du\_end;

du\_start = \*cur\_mem\_ctr;

/\*

if (du\_start == 0xfffff) {

du\_start = 0;

}

\*/

if (du\_start + 159 > 0xfffff) {

du\_end = 0xfffff;

} // dump ends at FFFFF if du\_start + 159 over FFFFF

else {

du\_end = du\_start + 159;

}

mem\_dump\_first\_last(du\_start, du\_end, cur\_mem\_ctr);

}

void mem\_dump\_first\_only(int du\_start, int \*cur\_mem\_ctr) {

int du\_end;

if (du\_start + 159 > 0xfffff) {

du\_end = 0xfffff;

} // dump ends at FFFFF if du\_start + 159 over FFFFF

else {

du\_end = du\_start + 159;

}

mem\_dump\_first\_last(du\_start,du\_end,cur\_mem\_ctr);

}

void mem\_dump\_first\_last(int du\_start, int du\_end, int \*cur\_mem\_ctr) {

int i,j;

int start\_line, end\_line, cur\_line;

int line\_bound, line\_bound\_end;

start\_line = (du\_start / 16); // start line for the print

end\_line = (du\_end / 16); // end line for the print

cur\_line = (du\_start / 16) \* 16; // current line for the memory's start addr

if(start\_line == end\_line){

printf("%05X ",cur\_line);

line\_bound = du\_start % 16;

line\_bound\_end = du\_end % 16;

for(j=0;j<line\_bound;j++){

printf(" ");

} // blank

for(j=line\_bound;j<=line\_bound\_end;j++){

printf("%02X ",mem\_arr[start\_line][j]);

} // char

for(j=line\_bound\_end+1;j<16;j++){

printf(" ");

} // blank

printf(" ; ");

for(j=0;j<16;j++){

if(j<line\_bound || j >line\_bound\_end){

printf(".");

}

else if(mem\_arr[start\_line][j] >= 0x20 && mem\_arr[start\_line][j] <= 0x7E){

printf("%c",mem\_arr[start\_line][j]);

}

else{

printf(".");

}

}

printf("\n");

if(du\_end == 0xfffff){

\*cur\_mem\_ctr = 0;

}

else{

\*cur\_mem\_ctr = du\_end + 1;

}

return;

}

for (i = start\_line; i <= end\_line; i++) {

printf("%05X ",cur\_line);

if (i == start\_line) {

line\_bound = (du\_start % 16);

for (j = 0; j < line\_bound;j++) {

printf(" ");

} // blank

for (j = line\_bound; j < 16;j++) {

printf("%02X ",mem\_arr[i][j]);

}

// mem

printf(" ; ");

for (j = 0; j < 16;j++) {

if (j < line\_bound) {

printf(".");

}

else if (mem\_arr[i][j] >= 0x20 && mem\_arr[i][j] <= 0x7E) {

printf("%c",mem\_arr[i][j]);

}

else {

printf(".");

}

}

// character

// only between character 20 ~ 7E can be printed

}

else if (i == end\_line) {

line\_bound = (du\_end % 16);

// mem

for (j = 0; j <= line\_bound;j++) {

printf("%02X ", mem\_arr[i][j]);

}

// blank

for (j = line\_bound + 1; j < 16; j++) {

printf(" ");

}

printf(" ; ");

// character

// only between character 20 ~ 7E can be printed

for (j = 0; j < 16;j++) {

if (j >= line\_bound) {

printf(".");

}

else if (mem\_arr[i][j] >= 0x20 && mem\_arr[i][j] <= 0x7E) {

printf("%c", mem\_arr[i][j]);

}

else {

printf(".");

}

}

}

else {

for (j = 0; j < 16;j++) {

printf("%02X ",mem\_arr[i][j]);

}

printf(" ; ");

// mem

for (j = 0; j < 16;j++) {

if (mem\_arr[i][j] >= 0x20 && mem\_arr[i][j] <= 0x7E) {

printf("%c", mem\_arr[i][j]);

}

else {

printf(".");

}

}

// character

}

cur\_line += 16;

printf("\n");

}

if (du\_end == 0xfffff) {

\*cur\_mem\_ctr = 0;

}

else {

\*cur\_mem\_ctr = du\_end + 1;

}

}

void mem\_edit(int edit\_mem, int edit\_val) {

int edit\_row, edit\_col;

edit\_row = edit\_mem / 16;

edit\_col = edit\_mem % 16;

mem\_arr[edit\_row][edit\_col] = edit\_val;

// edit the value of entered address

}

void mem\_fill(int fill\_start, int fill\_end, int fill\_val) {

int i,j;

int start\_line, end\_line, line\_bound;

start\_line = fill\_start / 16;

end\_line = fill\_end / 16;

for (i = start\_line; i <= end\_line;i++) {

if (i == start\_line) {

line\_bound = fill\_start % 16;

for (j = line\_bound; j < 16;j++) {

mem\_arr[i][j] = fill\_val;

}

} // if now is first line, edit from the start address.

else if (i == end\_line) {

line\_bound = fill\_end % 16;

for (j = 0; j <= line\_bound;j++) {

mem\_arr[i][j] = fill\_val;

}

} // if now is last line, edit before the end address.

else {

line\_bound = 16;

for (j = 0; j < line\_bound;j++) {

mem\_arr[i][j] = fill\_val;

}

} // edit all value of current line.

}

}

/\* cmd\_opcode.c \*/

#include "20121578.h"

void opcode\_init() {

FILE \*op\_txt = NULL;

int op\_num\_reader;

char op\_cmd\_reader[8];

char op\_mode\_reader[8];

int i, op\_cmd\_len, op\_table\_idx;

opcode\_ptr new\_node = NULL;

op\_txt = fopen("opcode.txt", "r"); // open opcode.txt file

if (op\_txt == NULL) {

printf("File open error!\n");

exit(-1);

}

while (fscanf(op\_txt,"%x %s %s",&op\_num\_reader,op\_cmd\_reader,op\_mode\_reader) != EOF) {

op\_table\_idx = 0;

op\_cmd\_len = strlen(op\_cmd\_reader);

for (i = 0; i < op\_cmd\_len;i++) {

op\_table\_idx += op\_cmd\_reader[i];

}

op\_table\_idx %= 20;

/\*

the index of mnemonic is the mod of mnemonic's letters' ASCII code's sum.

\*/

new\_node = (opcode\_ptr)malloc(sizeof(opcode\_node));

if (new\_node == NULL) {

printf("Memory Allocation Error!\n");

exit(-1);

}

new\_node->op\_num = op\_num\_reader;

strcpy(new\_node->op\_chr,op\_cmd\_reader);

strcpy(new\_node->op\_mode,op\_mode\_reader);

new\_node->link = NULL;

// create the node in hash table

if (opcode\_table[op\_table\_idx] == NULL) {

opcode\_table[op\_table\_idx] = new\_node;

}

else {

new\_node->link = opcode\_table[op\_table\_idx];

opcode\_table[op\_table\_idx] = new\_node;

}

// add the node in hash table

}

fclose(op\_txt);

}

void opcode\_table\_print() {

int i;

opcode\_ptr temp\_node;

for (i = 0; i < HASH\_SIZE;i++) {

printf("%d : ",i);

if (opcode\_table[i] == NULL) {

printf("\n");

continue;

}

temp\_node = opcode\_table[i];

while (temp\_node->link != NULL) {

printf("[%s, %02X] -> ",temp\_node->op\_chr,temp\_node->op\_num);

temp\_node = temp\_node->link;

}

printf("[%s, %02X]\n", temp\_node->op\_chr, temp\_node->op\_num);

}

// print opcode hash table

}

int opcode\_find(char \*op\_mnec, int mnec\_len) {

int i;

int op\_idx = 0;

opcode\_ptr temp\_node;

for (i = 0; i < mnec\_len;i++) {

op\_idx += op\_mnec[i];

}

op\_idx %= 20;

// calculate the index

temp\_node = opcode\_table[op\_idx];

while (temp\_node != NULL) {

if (!strcmp(temp\_node->op\_chr,op\_mnec)) {

printf("opcode is %02X\n",temp\_node->op\_num);

return 0;

}

temp\_node = temp\_node->link;

}

// searching entered mnemonic and if found, print the opcode.

err\_msg\_bundle(WRONG\_MNEMONIC);

return -1;

// if cannot found, return -1.

}

void opcode\_free() {

int i;

opcode\_ptr de\_node, temp\_node;

for (i = 0; i < HASH\_SIZE;i++) {

temp\_node = opcode\_table[i];

de\_node = opcode\_table[i];

while (temp\_node != NULL) {

temp\_node = temp\_node->link;

free(de\_node);

de\_node = temp\_node;

}

}

// deallocate mnemonic hash table

}

void history\_free() {

history\_ptr de\_node, temp\_node;

de\_node = his\_list\_front;

temp\_node = his\_list\_front;

while (temp\_node != NULL) {

temp\_node = temp\_node->link;

free(de\_node);

de\_node = temp\_node;

}

// deallocate command history list

}

/\* asm\_lst\_create.c \*/

#include "20121578.h"

int asmFirstPass(char \*asm\_file\_name) {

FILE \*asm\_fp = NULL;

int cur\_locctr = 0, cur\_line = 5, cur\_fmt;

char asm\_reader[129];

char \*sym, \*mnemonic, \*operand\_1;

const char \*asm\_token = " ,\t\n"; // token for assembler

unsigned int i = 0;

int byte\_len = 0;

sym = mnemonic = operand\_1 = NULL;

asm\_fp = fopen(asm\_file\_name,"r");

if (asm\_fp == NULL) {

printf("ERROR : Assembly File Open Error.\n");

return -1;

}

// open file

if (asm\_file\_name[strlen(asm\_file\_name) - 3] != 'a' || asm\_file\_name[strlen(asm\_file\_name) - 2] != 's' ||

asm\_file\_name[strlen(asm\_file\_name) - 1] != 'm') {

printf("ERROR : You should assemble .asm file\n");

return -2;

}

// if entered file isn't .asm file

while (fgets(asm\_reader,128,asm\_fp) != NULL) {

if (asm\_reader[0] == '.') {

cur\_line += 5;

continue;

} // case : comment

else {

asm\_reader[strlen(asm\_reader) - 1] = '\0';

trim\_left(asm\_reader);

sym = strtok(asm\_reader,asm\_token);

mnemonic = strtok(NULL,asm\_token);

if (strcmp(mnemonic,"START")) {

cur\_line -= 5;

fseek(asm\_fp, 0, SEEK\_SET);

break;

} // if there is no directive 'START', set the first locctr to 0.

operand\_1 = strtok(NULL,asm\_token);

cur\_locctr = strtol(operand\_1, NULL, 16);

break;

}

} // find the directive 'START' and set the first location counter

while (fgets(asm\_reader,128,asm\_fp) != NULL) {

cur\_line += 5;

sym = mnemonic = operand\_1 = NULL;

if (asm\_reader[0] == '.') {

cur\_line += 5;

continue;

} // case : comment

else {

asm\_reader[strlen(asm\_reader) - 1] = '\0';

trim\_left(asm\_reader);

sym = strtok(asm\_reader, asm\_token);

mnemonic = strtok(NULL, asm\_token);

operand\_1 = strtok(NULL, asm\_token);

if (sym[0] == '+') {

cur\_locctr += 4;

continue;

} // if format 4

else {

for (i = 0; i < strlen(sym); i++) {

if ((sym[i] >= 65 && sym[i] <= 90) || (sym[i] >= 97 || sym[i] <= 122)) {

continue;

}

else {

printf("ERROR : Invalid symbol name");

return cur\_line;

}

} // check the symbol name

cur\_fmt = opSearch(sym);

if (cur\_fmt != -1) {

cur\_locctr += cur\_fmt;

continue;

} // if there is mnemonic, add the format number and go on to the next line.

if (symbolTableSearch(sym) != -1) {

printf("ERROR : You already put same symbol in symbol list.\n");

return cur\_line;

}

else if (!strcmp("BASE", sym) || !strcmp("NOBASE", sym)) {

continue;

}

else if (!strcmp("END", sym)) {

break;

}

else {

symbolTableInsert(cur\_locctr, sym);

} // put the symbol in symbol table

if (!strcmp("BYTE", mnemonic)) {

byte\_len = 0;

if (operand\_1[0] == 'C') {

for (i = 2; i < strlen(operand\_1); i++) {

if (operand\_1[i] == 39) {

continue;

}

byte\_len++;

}

cur\_locctr += byte\_len;

} // character

else if (operand\_1[0] == 'X') {

for (i = 2; i < strlen(operand\_1); i++) {

if (operand\_1[i] == 39) {

continue;

}

byte\_len++;

}

if (byte\_len % 2 == 0) {

cur\_locctr += byte\_len / 2;

}

else {

cur\_locctr += (byte\_len / 2) + 1;

}

}

else {

printf("ERROR : Directive BYTE can save only hexadecimal number or string.\n");

return cur\_line;

}

} // case 'BYTE'

else if (!strcmp("WORD", mnemonic)) {

cur\_locctr += 3;

} // case 'WORD' (one integer constant)

else if (!strcmp("RESW", mnemonic)) {

cur\_locctr += strtol(operand\_1, NULL, 10) \* 3;

} // case 'RESW'

else if (!strcmp("RESB", mnemonic)) {

cur\_locctr += strtol(operand\_1, NULL, 10);

} // case 'RESB'

else {

if (mnemonic[0] == '+') {

++mnemonic;

cur\_fmt = opSearch(mnemonic);

if (cur\_fmt == -1) {

printf("ERROR : Invalid mnemonic.\n");

return cur\_line;

}

else {

cur\_locctr += 4;

}

} // format 4

else {

cur\_fmt = opSearch(mnemonic);

if (cur\_fmt == -1) {

printf("ERROR : Invalid mnemonic.\n");

return cur\_line;

}

else {

cur\_locctr += cur\_fmt;

}

} // others

} // ordinary cases

}

}

}

fclose(asm\_fp);

return 1;

}

int asmSecondPass(char \*asm\_file\_name,int \*start\_locctr,int \*end\_locctr,int \*nobase\_flag) {

FILE \*asm\_fp = NULL, \*lst\_fp = NULL;

char \*lst\_file\_name = NULL;

int cur\_line = 5;

int cur\_locctr = 0, base\_num = 0, cur\_fmt;

char asm\_reader[129], temp\_reader[129];

char \*sym, \*mnemonic, \*operand\_1, \*operand\_2;

const char \*asm\_token = " ,\t\n"; // token for assembler

int byte\_len, obj\_code, opcode\_num;

unsigned int i;

sym = mnemonic = operand\_1 = operand\_2 = NULL;

asm\_fp = fopen(asm\_file\_name, "r");

if (asm\_fp == NULL) {

printf("ERROR : Assembly File Open Error.\n");

return -1;

}

lst\_file\_name = (char\*)malloc(sizeof(char) \* (strlen(asm\_file\_name) + 1));

strcpy(lst\_file\_name, asm\_file\_name);

lst\_file\_name[strlen(lst\_file\_name) - 1] = 't';

lst\_file\_name[strlen(lst\_file\_name) - 2] = 's';

lst\_file\_name[strlen(lst\_file\_name) - 3] = 'l';

lst\_fp = fopen(lst\_file\_name, "w");

if (lst\_fp == NULL) {

printf("ERROR : List file open error\n");

return -1;

}

// make list file

while (fgets(asm\_reader, 128, asm\_fp) != NULL) {

if (asm\_reader[0] == '.') {

cur\_line += 5;

continue;

}

else {

asm\_reader[strlen(asm\_reader) - 1] = '\0';

strcpy(temp\_reader, asm\_reader);

sym = strtok(temp\_reader, asm\_token);

mnemonic = strtok(NULL, asm\_token);

if (strcmp("START",mnemonic)) {

cur\_line -= 5;

\*start\_locctr = -1;

fseek(asm\_fp, 0, SEEK\_SET);

break;

}

operand\_1 = strtok(NULL, asm\_token);

operand\_2 = strtok(NULL, asm\_token);

cur\_locctr = strtol(operand\_1, NULL, 16);

\*start\_locctr = cur\_locctr;

fprintf(lst\_fp, "%4d %04X %s\n", cur\_line, cur\_locctr, asm\_reader);

break;

}

} // find the start location. if there is no START, initialize location to 0.

while (fgets(asm\_reader, 128, asm\_fp) != NULL) {

sym = mnemonic = operand\_1 = operand\_2 = NULL;

cur\_line += 5;

if (asm\_reader[0] == '.') {

cur\_line += 5;

continue;

}

else {

asm\_reader[strlen(asm\_reader) - 1] = '\0';

strcpy(temp\_reader, asm\_reader);

sym = strtok(temp\_reader, asm\_token);

if (sym[0] == '+' || !strcmp(sym,"BASE") || !strcmp(sym, "END") || !strcmp(sym,"NOBASE") || opSearch(sym) != -1) {

mnemonic = sym;

sym = NULL;

}

else {

mnemonic = strtok(NULL, asm\_token);

}

// read symbol and mnemonic

operand\_1 = strtok(NULL, asm\_token);

operand\_2 = strtok(NULL, asm\_token);

if (!strcmp(mnemonic, "BASE") || !strcmp(mnemonic, "NOBASE")) {

fprintf(lst\_fp, "%4d %-28s ", cur\_line, asm\_reader);

}

// for directive BASE

else if (strcmp(mnemonic, "END") == 0) {

fprintf(lst\_fp, "%4d %-28s ", cur\_line, asm\_reader);

\*end\_locctr = cur\_locctr;

fprintf(lst\_fp, "\n");

break;

}

// for directive END

else {

fprintf(lst\_fp, "%4d %04X %-28s ", cur\_line, cur\_locctr, asm\_reader);

}

// other cases

if (!strcmp(mnemonic, "WORD")) {

cur\_locctr += 3;

obj\_code = strtol(operand\_1, NULL, 10);

fprintf(lst\_fp, "%06X\n", obj\_code);

continue;

} // for directive WORD

else if (!strcmp(mnemonic,"RESB")) {

cur\_locctr += strtol(operand\_1, NULL, 10);

fprintf(lst\_fp, "\n");

continue;

} // for directive RESB

else if (!strcmp(mnemonic,"RESW")) {

cur\_locctr += strtol(operand\_1, NULL, 10) \* 3;

fprintf(lst\_fp, "\n");

continue;

} // for directive RESW

else if (!strcmp(mnemonic,"BYTE")) {

byte\_len = 0;

if (operand\_1[0] == 'C') {

for (i = 2; i < strlen(operand\_1);i++) {

if (operand\_1[i] == 39) {

continue;

}

fprintf(lst\_fp, "%02X", operand\_1[i]);

byte\_len++;

}

if(byte\_len > 0x1E){

printf("ERROR : BYTE is too long\n");

remove(lst\_file\_name);

return cur\_line;

} // if BYTE is too long (constraint : 0x1E)

cur\_locctr += byte\_len;

fprintf(lst\_fp, "\n");

continue;

} // BYTE character

else if (operand\_1[0] == 'X') {

for (i = 2; i < strlen(operand\_1); i++) {

if (operand\_1[i] == 39) {

continue;

}

fprintf(lst\_fp, "%C", operand\_1[i]);

byte\_len++;

}

if(byte\_len > 60){

printf("ERROR : BYTE is too long\n");

remove(lst\_file\_name);

return cur\_line;

} // if BYTE is too long (constraint : 0x1E)

if (byte\_len % 2 == 0) {

cur\_locctr += (byte\_len / 2);

}

else {

cur\_locctr += (byte\_len / 2) + 1;

}

fprintf(lst\_fp, "\n");

continue;

} // BYTE hexadecimal

else {

printf("ERROR : The directive BYTE only create character or hexadecimal integer\n");

return cur\_line;

}

}

else if (!strcmp(mnemonic,"BASE")) {

base\_num = symbolTableSearch(operand\_1);

if (base\_num == -1) {

printf("ERROR : There is no symbol for BASE\n");

remove(lst\_file\_name);

return cur\_line;

}

\*nobase\_flag = 0;

// find symbol BASE in symbol table

fprintf(lst\_fp, "\n");

continue;

}

else if (!strcmp(mnemonic,"NOBASE")) {

fprintf(lst\_fp, "\n");

continue;

}

else if (!strcmp(mnemonic,"RSUB")) {

if(operand\_1 != NULL){

printf("mnemonic RSUB doesn't need operand\n");

remove(lst\_file\_name);

return cur\_line;

} // RSUB doesn't need other operand.

obj\_code = 0;

obj\_code += 0x4C; obj\_code += 3;

obj\_code <<= 16;

fprintf(lst\_fp, "%06X\n", obj\_code);

cur\_locctr += 3;

continue;

} // for mnemonic 'RSUB'

else {

obj\_code = 0;

if (mnemonic[0] == '+') {

cur\_fmt = 4;

++mnemonic;

} // format 4

else {

cur\_fmt = opSearch(mnemonic);

} // format 1,2,3

opcode\_num = opcodeNumberSearch(mnemonic);

if(opcode\_num == 0xC0 || opcode\_num == 0xC4 || opcode\_num == 0xF4 || opcode\_num == 0xC8 ||

opcode\_num == 0xF0 || opcode\_num == 0xF8){

if(operand\_1 != NULL){

printf("mnemonic %s doesn't need operand\n",mnemonic);

remove(lst\_file\_name);

return cur\_line;

}

}

// for mnemonics that do not need operand

// 'FLOAT', 'FIX', 'HIO', 'NORM', 'SIO', 'TIO'

switch (cur\_fmt) {

case 1: fprintf(lst\_fp, "%02X\n", opcode\_num); break;

case 2: obj\_code = objFormatTwo(opcode\_num, operand\_1, operand\_2); fprintf(lst\_fp, "%04X\n", obj\_code); break;

case 3: obj\_code = objFormatThree(opcode\_num, operand\_1, operand\_2, cur\_locctr, base\_num, \*nobase\_flag); fprintf(lst\_fp, "%06X\n", obj\_code); break;

case 4: obj\_code = objFormatFour(opcode\_num, operand\_1, operand\_2, cur\_locctr); fprintf(lst\_fp, "%08X\n", obj\_code); break;

default: printf("ERROR : Invalid opcode\n"); return cur\_line;

} // By format, execute the function.

if (obj\_code == -1) {

printf("ERROR : Object Code Error\n");

remove(lst\_file\_name);

return cur\_line;

}

cur\_locctr += cur\_fmt; // accumulate location counter

}

// ordinary case

}

}

fclose(asm\_fp);

fclose(lst\_fp);

free(lst\_file\_name);

return 1;

}

int asmCreate(char \*asm\_file\_name) {

int start\_locctr = 0, end\_locctr = 0;

int res\_flag;

int nobase\_flag = 1; // 1 : NOBASE, 0 : BASE

freeSymbolTable();

res\_flag = asmFirstPass(asm\_file\_name); // Pass 1

if (res\_flag == 1) { // if Pass 1 executed normally,

res\_flag = asmSecondPass(asm\_file\_name,&start\_locctr,&end\_locctr,&nobase\_flag);

// Pass 2

if (res\_flag == 1) {

res\_flag = objFileMaking(asm\_file\_name, start\_locctr, end\_locctr);

} // if Pass 2 is executed normally, create .obj file

}

if (res\_flag == 1) {

return 1;

}

else if (res\_flag == -1) {

printf("Plase check the existance of file\n");

return -1;

} // if there is no file

else if (res\_flag == -2) {

printf("Please assemble .asm file\n");

return -1;

} // if user try to assemble other form of file

else {

printf("Please check the assembly source code Line %d\n",res\_flag);

freeSymbolTable(); // deallocate symbol table

return -1;

} // information of error line

}

/\* asm\_obj\_create.c \*/

#include "20121578.h"

int objFileMaking(char \*lst\_file\_name,int locctr\_start,int locctr\_end) {

FILE \*lst\_fp = NULL, \*obj\_fp = NULL; // file pointer

char \*obj\_fname = NULL, \*lst\_fname = NULL;

char lst\_reader[129], temp\_reader[129];

char obj\_line[62];

char \*cur\_line, \*cur\_locctr, \*sym, \*mnemonic, \*op\_1, \*op\_2, \*objcode; // token

const char \*obj\_token = " ,\t\n"; // cutter

int first\_locctr = locctr\_start;

int h\_byte\_num = 0; // half byte number

int cur\_loc = locctr\_start; // current location counter

int loc\_plus; // next loc

int symbol\_flag; // format 4 and symbol flag

int \*mod\_record = NULL; // modification record

int mod\_num = 0; // the number of modification record

unsigned int i;

int byte\_len, j;

obj\_fname = (char\*)malloc(sizeof(char) \* (strlen(lst\_file\_name) + 1));

lst\_fname = (char\*)malloc(sizeof(char) \* (strlen(lst\_file\_name) + 1));

strcpy(obj\_fname, lst\_file\_name); strcpy(lst\_fname, lst\_file\_name);

obj\_fname[strlen(obj\_fname) - 3] = 'o'; obj\_fname[strlen(obj\_fname) - 2] = 'b'; obj\_fname[strlen(obj\_fname) - 1] = 'j';

lst\_fname[strlen(obj\_fname) - 3] = 'l'; lst\_fname[strlen(obj\_fname) - 2] = 's'; lst\_fname[strlen(obj\_fname) - 1] = 't';

lst\_fp = fopen(lst\_fname, "r");

obj\_fp = fopen(obj\_fname, "w");

if (lst\_fp == NULL) {

printf("ERROR : List file open error\n");

return -1;

} // open the list file

if (obj\_fp == NULL) {

printf("ERROR : Object file creation error\n");

return -1;

} // make the object file

if (locctr\_start != -1) {

fgets(lst\_reader, 128, lst\_fp);

lst\_reader[strlen(lst\_reader) - 1] = '\0';

cur\_line = strtok(lst\_reader, obj\_token);

cur\_locctr = strtok(NULL, obj\_token);

sym = strtok(NULL, obj\_token);

fprintf(obj\_fp, "H%-6s%06X%06X\n", sym, locctr\_start, locctr\_end - locctr\_start);

// write the Header

}

else {

fprintf(obj\_fp, "HTEMPSP%06X%06X\n", 0, locctr\_end);

locctr\_start++;

first\_locctr = cur\_loc = 0;

}

memset(obj\_line, '\0', 62);

while (fgets(lst\_reader, 128, lst\_fp) != NULL) {

cur\_line = cur\_locctr = sym = mnemonic = op\_1 = op\_2 = objcode = NULL;

symbol\_flag = 0;

loc\_plus = 0;

lst\_reader[strlen(lst\_reader) - 1] = '\0';

strcpy(temp\_reader, lst\_reader);

cur\_line = strtok(temp\_reader, obj\_token);

cur\_locctr = strtok(NULL, obj\_token);

if (strcmp(cur\_locctr, "BASE") == 0 || strcmp(cur\_locctr, ".") == 0 || strcmp(cur\_locctr, "NOBASE") == 0) {

continue;

} // directive BASE and comment

else if (strcmp(cur\_locctr, "END") == 0) {

fprintf(obj\_fp, "T%06X%02X%s\n", first\_locctr, h\_byte\_num, obj\_line);

for (j = 0; j < mod\_num; j++) {

fprintf(obj\_fp, "M%06X05\n", mod\_record[j]);

}

fprintf(obj\_fp, "E%06X\n", locctr\_start);

break;

} // directive END

else {

sym = strtok(NULL, obj\_token);

if (sym[0] == '+' || symbolTableSearch(sym) == -1) {

mnemonic = sym;

sym = NULL;

}

else {

mnemonic = strtok(NULL, obj\_token);

}

// find label and mnemonic

if (mnemonic[0] == '+') {

symbol\_flag = 0;

loc\_plus = 4;

op\_1 = strtok(NULL, obj\_token);

op\_2 = strtok(NULL, obj\_token);

if (strlen(op\_2) < 8) {

objcode = strtok(NULL, obj\_token);

}

else {

objcode = op\_2;

op\_2 = NULL;

}

if (op\_1[0] == '#' || op\_1[0] == '@') {

op\_1++;

}

for (i = 0; i < strlen(op\_1); i++) {

if (op\_1[i] < 48 || op\_1[i] > 57) {

symbol\_flag = 1;

}

}

if (symbol\_flag == 1) {

if (mod\_record == NULL) {

mod\_record = (int\*)malloc(sizeof(int));

}

else {

mod\_record = (int\*)realloc(mod\_record, sizeof(int) \* (mod\_num + 1));

}

mod\_record[mod\_num++] = cur\_loc + 1;

}

// symbol

} // format 4. If the modification is needed, save the location counter.

else if (strcmp(mnemonic, "BYTE") == 0) {

byte\_len = 0;

op\_1 = strtok(NULL, obj\_token);

for (i = 2; i < strlen(op\_1) - 1; i++) {

byte\_len++;

}

if (op\_1[0] == 'C') {

loc\_plus = byte\_len;

} // for characeter

else {

if (byte\_len % 2 == 0) {

loc\_plus = (byte\_len / 2);

}

else {

loc\_plus = (byte\_len / 2) + 1;

}

} // for integer

objcode = strtok(NULL, obj\_token);

} // directive byte

else if (strcmp(mnemonic, "RESW") == 0) {

op\_1 = strtok(NULL, obj\_token);

loc\_plus = 3 \* strtol(op\_1, NULL, 10);

cur\_loc += loc\_plus;

continue;

} // directive RESW

else if (strcmp(mnemonic, "RESB") == 0) {

op\_1 = strtok(NULL, obj\_token);

loc\_plus = strtol(op\_1, NULL, 10);

fprintf(obj\_fp, "T%06X%02X%s\n", first\_locctr, h\_byte\_num, obj\_line);

first\_locctr = cur\_loc + loc\_plus;

cur\_loc += loc\_plus;

h\_byte\_num = 0;

memset(obj\_line, '\0', 62);

continue;

// go to the next line

} // directive RESB

else if (strcmp(mnemonic, "WORD") == 0) {

op\_1 = strtok(NULL, obj\_token);

objcode = strtok(NULL, obj\_token);

loc\_plus = 3;

} // directive WORD

else if (strcmp(mnemonic, "RSUB") == 0) {

objcode = strtok(NULL, obj\_token);

loc\_plus = 3;

} // mnemonic RSUB

else if (!strcmp(mnemonic, "SHIFTL") || !strcmp(mnemonic, "SHIFTR")) {

op\_1 = strtok(NULL, obj\_token);

op\_2 = strtok(NULL, obj\_token);

objcode = strtok(NULL, obj\_token);

loc\_plus = opSearch(mnemonic);

}

else {

op\_1 = strtok(NULL, obj\_token);

op\_2 = strtok(NULL, obj\_token);

if (!strcmp(op\_2, "X") || !strcmp(op\_2, "A") || !strcmp(op\_2, "L") || !strcmp(op\_2, "PC") || !strcmp(op\_2, "SW") || !strcmp(op\_2, "B") || !strcmp(op\_2, "S") || !strcmp(op\_2, "T") || !strcmp(op\_2, "F")) {

objcode = strtok(NULL, obj\_token);

}

else {

objcode = op\_2;

op\_2 = NULL;

}

loc\_plus = opSearch(mnemonic);

} // mnemonic

if (h\_byte\_num + loc\_plus > 0x1E) {

fprintf(obj\_fp, "T%06X%02X%s\n", first\_locctr, h\_byte\_num, obj\_line);

first\_locctr = cur\_loc;

cur\_loc += loc\_plus;

h\_byte\_num = loc\_plus;

memset(obj\_line, '\0', 62);

strcpy(obj\_line, objcode);

} // write the tail

else {

strcat(obj\_line, objcode);

h\_byte\_num += loc\_plus;

cur\_loc += loc\_plus;

}

}

}

printf(" output file : [%s], [%s]\n", lst\_fname, obj\_fname); // print the result

printf(" last line : %s\n", cur\_line); // for erase the warning

free(mod\_record);

free(lst\_fname);

free(obj\_fname);

fclose(lst\_fp);

fclose(obj\_fp);

return 1;

}

/\* asm\_obj\_format.c \*/

#include "20121578.h"

int objFormatTwo(int opcode\_num,char \*operand\_1, char \*operand\_2) {

int obj = opcode\_num;

obj <<= 4;

if (operand\_1 != NULL) {

if(opcode\_num == 0xB0){

obj += strtol(operand\_1,NULL,16);

} // special case : mnemonic 'SVC'

if (strcmp(operand\_1, "A") == 0) {

obj += 0;

}

else if (strcmp(operand\_1, "X") == 0) {

obj += 1;

}

else if (strcmp(operand\_1, "L") == 0) {

obj += 2;

}

else if (strcmp(operand\_1, "PC") == 0) {

obj += 8;

}

else if (strcmp(operand\_1, "SW") == 0) {

obj += 9;

}

else if (strcmp(operand\_1, "B") == 0) {

obj += 3;

}

else if (strcmp(operand\_1, "S") == 0) {

obj += 4;

}

else if (strcmp(operand\_1, "T") == 0) {

obj += 5;

}

else if (strcmp(operand\_1, "F") == 0) {

obj += 6;

}

// ordinary cases

else {

printf("ERROR : Wrong register\n");

return -1;

}

// if there is invalid name of register

} // find the first register

else{

printf("ERROR : Please input at least one register\n");

return -1;

}

// format 2 needs at least one register.

obj <<= 4;

if (operand\_2 != NULL) {

if(opcode\_num == 0xB4 || opcode\_num == 0xB8 || opcode\_num == 0xB0){

printf("ERROR : Only 1 register needed\n");

return -1;

} // special case : mnemonic 'CLEAR', 'TIXR', 'SVC'

if(opcode\_num == 0xA4){

obj += strtol(operand\_2,NULL,16);

return obj;

} // special case : mnemonic 'SHIFTL'

if (strcmp(operand\_2, "A") == 0) {

obj += 0;

}

else if (strcmp(operand\_2, "X") == 0) {

obj += 1;

}

else if (strcmp(operand\_2, "L") == 0) {

obj += 2;

}

else if (strcmp(operand\_2, "PC") == 0) {

obj += 8;

}

else if (strcmp(operand\_2, "SW") == 0) {

obj += 9;

}

else if (strcmp(operand\_2, "B") == 0) {

obj += 3;

}

else if (strcmp(operand\_2, "S") == 0) {

obj += 4;

}

else if (strcmp(operand\_2, "T") == 0) {

obj += 5;

}

else if (strcmp(operand\_2, "F") == 0) {

obj += 6;

}

// ordinary cases

else {

printf("ERROR : Wrong register\n");

return -1;

}

// if there is invalid name

} // find the second register

else if(opcode\_num == 0x90 || opcode\_num == 0xA0 || opcode\_num == 0x9C || opcode\_num == 0x98 ||

opcode\_num == 0xAC || opcode\_num == 0xA4 || opcode\_num == 0x94){

printf("ERROR : r2 is needed\n");

return -1;

}

// for the registers that operand\_2 is needed

// 'ADDR', 'COMPR', 'DIVR', 'MULR', 'RMO', 'SHIFTL', 'SUBR'

return obj;

}

int objFormatThree(int opcode\_num, char \*operand\_1, char \*operand\_2, int cur\_locctr, int base\_num, int nobase\_flag) {

int obj = opcode\_num;

int const\_flag = 0;

int sym\_locctr, disp = 0;

int cur\_pc = cur\_locctr + 3;

unsigned int i;

if (operand\_1[0] == '#') {

obj += 1;

++operand\_1;

} // immediate addressing

else if (operand\_1[0] == '@') {

obj += 2;

++operand\_1;

} // indirect addressing

else {

obj += 3;

} // simple addressing

obj <<= 4;

for (i = 0; i < strlen(operand\_1); i++) {

if (operand\_1[i] < 48 || operand\_1[i] > 57) {

const\_flag = 1;

break;

} // not number

}

if (const\_flag == 1) {

sym\_locctr = symbolTableSearch(operand\_1);

// find the symbol in symbol table

if (sym\_locctr == -1) {

printf("ERROR : Invalid symbol\n");

return -1;

}

// if cannot found, return -1.

}

else {

sym\_locctr = strtol(operand\_1, NULL, 10);

}

// if operand 1 is constant

if (operand\_2 != NULL) {

if (!strcmp(operand\_2, "X")) {

obj += 8;

} // indexded mode

else {

printf("ERROR : Invalid operand\n");

return -1;

}

}

if (const\_flag == 0) { // constant

if (sym\_locctr > 4095) {

return objFormatFour(opcode\_num, operand\_1, operand\_2, cur\_locctr);

} // if we cannot use format 3

else {

obj <<= 12;

obj += sym\_locctr;

return obj;

}

}

else {

disp = sym\_locctr - cur\_pc;

if (disp >= -2048 && disp <= 2047) {

obj += 2; obj <<= 12;

if (disp >= 0) {

obj += disp;

}

else {

disp &= 0x00000FFF;

obj += disp;

}

return obj;

} // pc relatives

else if (nobase\_flag == 0) {

disp = sym\_locctr - base\_num;

if (disp >= 0 && disp <= 4095) {

obj += 4; obj <<= 12; obj += disp;

return obj;

}

else {

return objFormatFour(opcode\_num, operand\_1, operand\_2, cur\_locctr);

} // if we cannot use both PC relative and BASE relative, use format 4.

} // base relative

else {

return objFormatFour(opcode\_num, operand\_1, operand\_2, cur\_locctr);

} // if we cannot use both PC relative and BASE relative, use format 4.

}

}

int objFormatFour(int opcode\_num, char \*operand\_1, char \*operand\_2, int cur\_locctr) {

int obj = opcode\_num;

int sym\_locctr, const\_flag = 0;

unsigned int i;

if (operand\_1[0] == '#') {

obj += 1;

++operand\_1;

} // immediate addressing

else if (operand\_1[0] == '@') {

obj += 2;

++operand\_1;

} // indirect addressing

else {

obj += 3;

}

obj <<= 4;

for (i = 0; i < strlen(operand\_1); i++) {

if (operand\_1[i] < 48 || operand\_1[i] > 57) {

const\_flag = 1;

break;

} // not number

}

if (const\_flag == 1) {

sym\_locctr = symbolTableSearch(operand\_1);

if (sym\_locctr == -1) {

printf("ERROR : Invalid symbol\n");

return -1;

}

}

else {

sym\_locctr = strtol(operand\_1, NULL, 10);

}

if (operand\_2 != NULL) {

if (!strcmp(operand\_2, "X")) {

obj += 8;

} // indexded mode

else {

printf("ERROR : Invalid operand\n");

return -1;

}

}

obj += 1; // for the bit 'e'

obj <<= 20;

obj += sym\_locctr; // put the address

return obj;

}

/\* asm\_symbol\_ctr.c \*/

#include "20121578.h"

int opSearch(char \*op\_str) {

int i;

int op\_idx = 0;

int op\_len = strlen(op\_str);

opcode\_ptr search\_op = NULL;

for (i = 0; i < op\_len;i++) {

op\_idx += op\_str[i];

}

op\_idx %= 20;

search\_op = opcode\_table[op\_idx];

// calculate the index of opcode hash table and initialize

while (search\_op != NULL) {

if (!strcmp(op\_str,search\_op->op\_chr)) {

if (!strcmp(search\_op->op\_mode,"1")) {

return 1;

} // format 1

else if (!strcmp(search\_op->op\_mode,"2")) {

return 2;

} // format 2

else {

return 3;

} // format 3 or 4

} // return the mode of mnemonic

search\_op = search\_op->link;

}

return -1;

// if there is no mnemonic

}

int opcodeNumberSearch(char \*op\_str) {

int i;

int op\_idx = 0;

int op\_len = strlen(op\_str);

opcode\_ptr search\_op = NULL;

for (i = 0; i < op\_len; i++) {

op\_idx += op\_str[i];

}

op\_idx %= 20;

search\_op = opcode\_table[op\_idx];

// calculate the index of opcode hash table

while (search\_op != NULL) {

if (!strcmp(op\_str, search\_op->op\_chr)) {

return search\_op->op\_num;

} // if opcode is found, return opcode.

search\_op = search\_op->link;

}

return -1;

// if there is no mnemonic

}

int symbolTableSearch(char \*sym\_str) {

symbol\_ptr sy\_search;

int table\_idx = sym\_str[0] - 65;

sy\_search = last\_symbol\_table[table\_idx];

while (sy\_search != NULL) {

if (!strcmp(sy\_search->symbol\_str,sym\_str)) {

return sy\_search->locctr;

} // if symbol is in the table, return the location counter of symbol.

sy\_search = sy\_search->link;

}

return -1;

// if there is no symbol, return -1.

}

void symbolTableInsert(int sym\_locctr, char\* sym\_str) {

symbol\_ptr added\_symbol = NULL;

symbol\_ptr search\_front = NULL, search\_rear = NULL;

int table\_idx = sym\_str[0] - 65; // initialize the index of table

added\_symbol = (symbol\_ptr)malloc(sizeof(symbol\_node));

if (added\_symbol == NULL) {

printf("Memory Allocation Error\n");

exit(-1);

}

strcpy(added\_symbol->symbol\_str, sym\_str);

added\_symbol->locctr = sym\_locctr;

added\_symbol->link = NULL;

// create new node of Symbol Table

/\*

SYMBOL\_TABLE\_SIZE == 26

(The number of Alphabet)

\*/

search\_front = last\_symbol\_table[table\_idx];

if (last\_symbol\_table[table\_idx] == NULL) {

last\_symbol\_table[table\_idx] = added\_symbol;

} // if this index is empty

else if (strcmp(added\_symbol->symbol\_str, search\_front->symbol\_str) > 0) {

added\_symbol->link = last\_symbol\_table[table\_idx];

last\_symbol\_table[table\_idx] = added\_symbol;

}

else {

search\_rear = search\_front->link;

while (search\_rear != NULL) {

if (strcmp(added\_symbol->symbol\_str, search\_rear->symbol\_str) > 0) {

search\_front->link = added\_symbol;

added\_symbol->link = search\_rear;

return;

}

search\_front = search\_front->link;

search\_rear = search\_rear->link;

}

search\_front->link = added\_symbol;

}

// By descending order, put new symbol to the symbol table.

}

void freeSymbolTable() {

int i;

symbol\_ptr temp\_node = NULL, deall\_node = NULL;

for (i = 0; i < SYMBOL\_TABLE\_SIZE;i++) {

temp\_node = last\_symbol\_table[i];

while (temp\_node != NULL) {

deall\_node = temp\_node;

temp\_node = temp\_node->link;

free(deall\_node);

}

last\_symbol\_table[i] = NULL;

}

// deallocation of symbol table

}

void printSymbolTable() {

int i;

symbol\_ptr print\_node = NULL;

// By descending order, print the symbol.

// Before printing, symbol is put by descending order.

// So, print symbol from 'Z'

for (i = SYMBOL\_TABLE\_SIZE-1; i >= 0;i--) {

print\_node = last\_symbol\_table[i];

while (print\_node != NULL) {

printf("\t%s\t%04X\n",print\_node->symbol\_str,print\_node->locctr);

print\_node = print\_node->link;

}

}

}