**과목명: 시스템프로그래밍**

**담당 교수 명: 김 지 환**

**2분반**

**<<Project #2>>**

**서강대학교 컴퓨터학과**

**20181618**

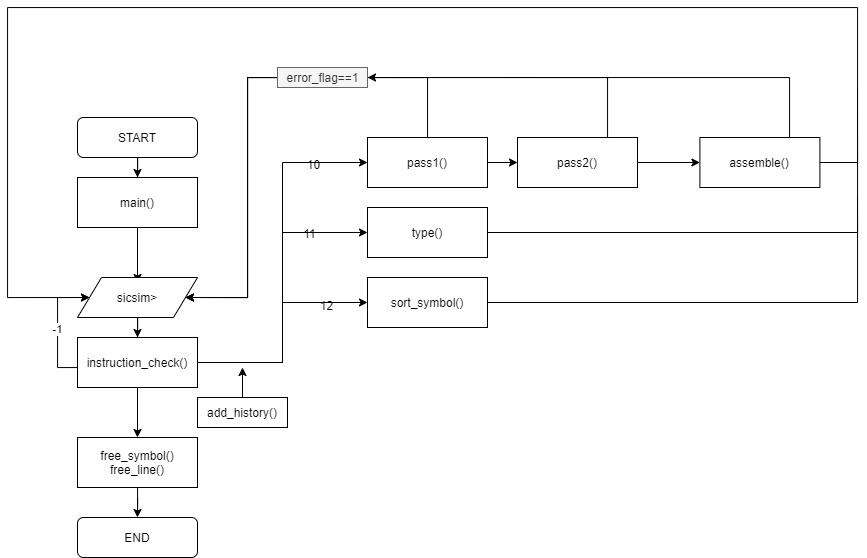
**김하늘**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. 모듈 이름 : int pass1();
      1. 기능
      2. 사용 변수
   2. 모듈 이름 : void pass2();
      1. 기능
      2. 사용 변수
   3. 모듈 이름 : int assemble();
      1. 기능
      2. 사용 변수
   4. 모듈 이름 : void type();
      1. 기능
      2. 사용 변수
   5. 모듈 이름 : int add\_symbol(char\* str, int num);
      1. 기능
      2. 사용 변수
   6. 모듈 이름 : int search\_symbol(char\* str);
      1. 기능
      2. 사용 변수
   7. 모듈 이름 : void sort\_symbol();
      1. 기능
      2. 사용 변수
   8. 모듈 이름 : char\* two\_comp(int n, int b);
      1. 기능
      2. 사용 변수
   9. 모듈 이름 : char\* biTohex(char\* buf, int b);
      1. 기능
      2. 사용 변수
   10. 모듈 이름 : char\* ascii\_str(char\* str);
       1. 기능
       2. 사용 변수
   11. 모듈 이름 : int loc\_func(char\* str1, char\* str2);
       1. 기능
       2. 사용 변수
   12. 모듈 이름 : char\* hex(int n, int b);
       1. 기능
       2. 사용 변수
   13. 모듈 이름 : int regi(char\* str);
       1. 기능
       2. 사용 변수
   14. 모듈 이름 : int op\_check(char\* str);
       1. 기능
       2. 사용변수
   15. 모듈 이름 : void obj\_reset(char\* str1, char\* str2, char\* str3, char\* str4);
       1. 기능
       2. 사용변수
   16. 모듈 이름 : void free\_symbol(); / void free\_line();
       1. 기능
       2. 사용 변수
4. **전역 변수 정의**
   1. int sym\_flag
   2. int cur\_loc
   3. int fLoc
   4. int base
   5. char filename[50]
   6. int fCnt
   7. char\* reg[]
5. **코드**
   1. 20181618.c
   2. 20181618.h
6. 프로그램 개요

프로젝트 1에서 구현한 셀에 assemble 기능을 추가하였다. SIC/XE의 asm 소스 파일을 입력받아 object 파일을 생성하고 어셈블리 과정 중 생성된 symbol을 테이블로 만들며, 어셈블 결과로 얻은 파일을 확인할 수 있는 명령어가 추가되었다. 교재의 2.2에 설명된 SIC/XE 어셈블러를 기준으로 구현하였다.

1. 프로그램 설명
   1. 프로그램 흐름도



프로그램의 Flow chart는 위와 같다. 프로젝트2 에서 추가된 기능을 위주로 표시하였다.

assemble 명령어는 파일 이름을 받아 파일이 유효한지 확인한 뒤, 해당 asm파일을 열어 pass1 기능을 수행한다. 한 줄씩 읽으며 symbol 테이블 생성 및 loc를 저장하고, 이 정보를 바탕으로 pass2 기능을 수행하여 object code를 생성한다. 이를 바탕으로 lst, obj파일을 생성하여 출력한다. instruction이 정의되지 않은 불분명한 경우, symbol 이름이 중복되는 경우, 정의되지 않은 변수명 또는 symbol이 사용될 경우, 존재하지 않는 레지스터일 경우 error flag를 체크하여 lst, obj파일을 생성을 중단하고 다시 명령어를 받는 단게로 돌아간다.

type 명령어는 파일 내용을 출력하는 기능이며, 파일이 존재하는지 확인한 뒤 출력한다.

symbol 명령어는 assemble이 성공적으로 수행되었을 때 갱신되며, 저장되어있는 symbol 이름을 내림차순으로 정렬하여 출력하였다. 또한 프로그램 실행 후 symbol과 line 정보가 한번이라도 동적할당되어 리스트가 되었다면 프로그램 종료시 free를 해주었다.

함수들의 자세한 기능은 다음 모듈정의 부분에서 설명하겠지만, assemble 함수의 구현은 다음과 같다.

pass1 함수에서는 asm 파일을 열어 한 줄씩 읽고, 그 라인의 명령어를 확인하여 포맷을 분류하였다. opcode.txt에 정의되어있는 명령어일 경우 hash function을 통해 검색하였고, 명렁어에 '+'가 붙어있을 경우 포맷4로 처리하였다. opcode.txt 에 존재하지 않는 START, END, BYTE, WORD, RESB, RESW, BASE일 경우 따로 처리를 해주었고, operand를 확인해서 할당된 byte수를 저장하여 loc 값을 지정하였다.

pass2 함수에선 pass1에서 구한 loc와 포맷 정보를 바탕으로 2진수 비트를 하나씩 변경시켜주었다. BYTE, WORD의 경우 opcode가 존재하지 않으므로 값을 그대로 저장해주었고, 포맷1은 opcode만을 1byte로 저장하며, 포맷2는 opcode 저장 후 레지스터 번호를 테이블에서 가져와 저장해주었다. 포맷 3, 4의 경우 어떤 방식의 addressing인지에 따라 nixbpe 비트를 변경해주어야 했다. 또한 opcode가 6bit이므로 opcode 값에 따라 비트를 바꾸어주었고, 포맷 4의 경우 e = 1을 해주었다. 명령어에 #이 있을 경우 immediate addressing이므로 i=1, @가 있을 경우 indirect addressing이므로 n=1, 배열과 X에 관한 명령어일경우 x=1로 해주었다. 그 다음 disp 계산을 해주는데, 우선 pc relative addressing으로 먼저 offset 값을 계산한다. 이 값이 [-2048, 2047]의 범위를 벗어날 경우 base relative addressing 방식으로 처리를 한다. 포맷 4의 경우 직접 주소 방식을 이용하기 때문에 pc와 b 값 모두 0bit이다. 이후 2진수 비트들을 16진수로 바꾸어 object code로 저장해주었다.

위의 과정이 성공적으로 에러 없이 수행되었다면 파일을 출력한다. lst파일에는 기존의 파일에 loc와 object code를 추가하여 출력하고, obj파일의 Text recode부분은 한 줄에 30byte를 넘기지 않도록 명세서에 나와있는 것과 같이 조정하였다.

1. 모듈 정의
   1. 모듈 이름 : **int pass1();**
      1. 기능

Pass1을 구현한 함수이다. 어셈블리 프로그램 소스 파일을 열어 한 줄씩 읽으며 instruction의 종류에 따라 fomat 또는 변수, 상수 등을 구분한 뒤, 라인별로 할당되는 메모리를 계산하여 location counter를 지정한다. 또한 symbol을 테이블에 리스트 형태로 추가한다. 이때 심볼이 중복이면 에러처리를 해주었다. 공백, 주석, BASE, END의 경우 Loc는 지정되지 않는다. error가 일어났을 경우 -1을 반환한다.

* + 1. 사용 변수

FILE\* f - file 포인터

char buf[50] - 한 줄 읽은 문자열이 저장됨

char str1[10], str2[10], str3[10], str4[10] - 토큰화된 문자열이 저장됨

char\* ptr - strtok의 반환값이 저장됨

debugCnt - 에러메세지 출력시 에러 발생한 지점 카운트

line\* temp, node - 해당 라인에 대한 정보를 저정하기 위한 line 구조체 노드

* 1. 모듈 이름 : **void pass2();**
     1. 기능

Pass2를 구현한 함수이다. pass1()함수에서 처리한 정보를 바탕으로 해당 라인에 대한 object code를 생성한다. 포맷에 따른 opcode 및 nixbpe, disp, addr의 비트를 적절하게 바꾸어주며 정의되지않은 instruction 또는 변수가 사용될 경우 에러처리를 해주었다. 공백, 주석, BASE, END 및 변수 할당에는 object code가 생성되지 않는다.

* + 1. 사용 변수

FILE\* f - file 포인터

char buf[50] - 한 줄 읽은 문자열이 저장됨

char str1[10], str2[10], str3[10], str4[10] - 토큰화된 문자열이 저장됨

char\* ptr - strtok의 반환값이 저장됨

char op[3], nixbpe[7], disp[4], addr[6] - object code 생성을 위한 2진수 비트문자열

int b\_flag - relative addressing의 offset 값이 pc register로 표현하지 못할 경우 b register를 사용하기 위한 flag

line\* cur, ne - object code를 계산할 현재 라인의 노드(cur)와 pc값을 구하기 위한 다음 줄의 노드(ne)

* 1. 모듈 이름 : **int assemble();**
     1. 기능

사용가 지정한 asm 파일을 assembly하여 Pass1, Pass2 과정을 거친 뒤 얻은 loc와 object code 정보를 바탕으로 lst 파일과 obj파일을 생성하는 함수이다. 에러가 생기면 파일을 생성하지 않는다. lst 파일에는 기존의 코드에 라인별 loc, object code가 추가되었으며, obj파일의 Text record 한 줄의 최대 byte 길이는 30을 넘기지 않는다. pass1에서 에러가 발생했을 경우, -1을 반환하여 assemble을 중단한다.

* + 1. 사용 변수

FILE\* fr, fw - asm파일을 열어 읽고 lst, obj파일을 열어 결과를 출력한다.

char buf[50] - 한 줄을 읽어 저장한다.

line\* cur - lst파일을 출력하기 위해 해당 리스트에 저장된 loc 값을 사용한다.

char obj\_buf[200] - Text recode 한 줄에 대한 정보를 저장한다. strcat을 이용해 최대 길이 또는 변수 지정 전까지의 object code가 저장된다.

int first - Text record 한 줄의 시작 주소가 저장된다.

* 1. 모듈 이름 : **void type();**
     1. 기능

type 기능을 구현한 함수이다. 파일이 존재하지 않을 경우 에러메세지를 출력하며 해당 파일의 내용 전체를 출력한다.

* + 1. 사용 변수

FILE\* f - 파일을 열기위한 포인터

char buf - 한 문자씩 저장된다.

* 1. 모듈 이름 : **int add\_symbol(char\* str, int num);**
     1. 기능

문자열과 loc값 정보를 받아 해당 symbol을 리스트에 주가하는 함수이다. 중복되는 symbol이 있을 경우 에러메세지를 출력하고 -1이 반환되며 명령이 종료된다.

* + 1. 사용 변수

sym\* temp, node - 리스트 탐색 및 추가를 위한 symbol 구조체

* 1. 모듈 이름 : **int search\_symbol(char\* str);**
     1. 기능

symbol 테이블에서 일치하는 symbol을 찾아 loc값을 반환한다. 일치하는 symbol이 없을 경우 에러메세지를 출력하며 명령이 종료된다.

* + 1. 사용 변수

sym\* temp - 리스트 탐색을 위한 구조체

* 1. 모듈 이름 : **void sort\_symbol();**
     1. 기능

symbol 명령어 수행을 위해 구현한 함수이다. 기존에 저장된 symbol 리스트에서 symbol 문자열을 복사하고, 인덱스 배열을 만들어 문자열과 맵핑한 뒤 symbol 문자열의 값을 비교하며 인덱스 배열을 솔팅한다.

* + 1. 사용 변수

char\*\* temp - symbol 문자열을 동적할당하여 복사한다.

int\* tLoc - symbol의 loc값을 동적할당하여 복사한다.

int\* idx - temp와 tLoc를 맵핑하는 인덱스이다. 실제 정렬되는 배열이다.

sym\* cur - symbol 리스트의 정보를 탐색하기 위한 노드

int i, j, t, cnt - 반복문의 인덱싱 및 임시 저장 변수

* 1. 모듈 이름 : **char\* two\_comp(int n, int b);**
     1. 기능

음수인 10진수 값을 보수화하여 16진수 문자열로 반환하는 함수이다. 10진수 음수와 원하는 문자열의 길이를 받아 2진수 양수로 만들고, 보수화 한 뒤 biTohex()함수를 통해 16진수 문자열로 반환한다.

* + 1. 사용 변수

int i, idx, temp - 반복문의 인덱싱 및 임시 저장 변수

* 1. 모듈 이름 : **char\* biTohex(char\* buf, int b);**
     1. 기능

2진수 문자열을과 원하는 길이를 받아서 16진수 문자열로 반환하는 함수이다.

* + 1. 사용 변수

int i, idx, n - 반복문 인덱싱 및 임시 변수이다.

char\* str - 반환할 16진수 문자열을 동적할당하여 초기화한다.

3. 10. 모듈 이름 : **char\* ascii\_str(char\* str);**

3. 10. 1. 기능

문자열의 문자 하나하나를 16진수 아스키코드 문자로 변환하여 반환하는 함수이다.

3. 10. 2. 사용 변수

int i, n, temp - 반복문 인덱싱 및 임시변수이다.

char s[3] - strcat으로 이어붙일 정보가 저장되는 임시 문자열이다.

3. 11. 모듈 이름 : **int loc\_func(char\* str1, char\* str2);**

3. 11. 1. 기능

instruction을 분석하여 포맷 정보를 검색해 해당 라인에 할당되는 byte 크기를 반환한다. opcode에 저장된 instruction이 아닌 BYTE, WORD, RESB, RESW, START, END, BASE의 케이스도 적절하게 byte를 계산하여 반환한다. 예를들어, RESW의 경우 배정된 숫자에 3을 곱하고, RESB의 경우 그대로 정수로 바꾸어 반환한다. byte가 배정되지 않는 BASE, END, 주석, 공백 등은 음수를 반환한다.

3. 11. 2. 사용 변수

int i, n, len - 반복문 인덱싱 및 임시변수이다.

char\* ptr - strtok의 결과값이 저장된다.

char t\_str - 임시 문자열

3. 12. 모듈 이름 : **char\* hex(int n, int b);**

3. 12. 1. 기능

10진수 정수를 b길이의 16진수 문자열로 반환하는 함수이다.

3. 12. 2. 사용 변수

int i, a, temp - 반복문 인덱싱 및 임시변수

char\* str - 변환된 16진수가 저장된다.

3. 13. 모듈 이름 : **int regi(char\* str);**

3. 13. 1. 기능

레지스터 리스트에서 입력받은 레지스터를 검색하여 번호를 반환한다. 테이블에 존재하지 않을 경우 에러메세지를 출력하며 명령을 종료한다.

3. 13. 2. 사용 변수

int cnt, idx - 반복문 인덱싱 및 임시변수

3. 14. 모듈 이름 : **int op\_check(char\* str);**

3. 14. 1. 기능

opcode에 저장된 instruction이 아닌 명령어로, BYTE, WORD, RESB, RESW, BASE, START, END를 구분하여 번호를 반환한다.

3. 14. 2. 사용 변수

char\* str - 파일의 instruction이 저장되어있다.

3. 15. 모듈 이름 : **void obj\_reset(char\* str1, char\* str2, char\* str3, char\* str4);**

3. 15. 1. 기능

파일 라인별 토큰을 분리하기 위한 초기화 함수이다.

3. 15. 2. 사용 변수

int i - 반복문 인덱스

3. 16. 모듈 이름 : **void free\_symbol(); / void line\_opcode();**

3. 16. 1. 기능

symbol list, line list를 free시켜준다

3. 16. 2. 사용 변수

sym\* temp, prev

line\* temp, prev

1. 전역 변수 정의
   1. int sym\_flag

assemble이 성공적으로 실행되었을 때 1이 저장되어 이후 symbol 및 line 리스트를 free할 때 사용된다.

* 1. int cur\_loc

현재 loc값이 저장되어있다.

* 1. int fLoc

asm파일의 가장 끝의 Loc값이 저장되어있다.

* 1. int base

BASE로 지정된 symbol의 loc값이 저장되어있다.

* 1. char filename[50]

assemble, type 명령어의 파일 이름이 저장되어있다.

* 1. int fCnt

파일의 최종 라인 번호가 저장되어있다.

* 1. char\* reg[]

레지스터가 저장된 문자열 배열로, {"A", "X", "L", "B", "S", "T", "F", "PC", "SW"} 으로 초기화되어있다.

* 1. sym\* symbol, line\* l\_head

asm 파일이 정상적으로 assemble 되었을 경우 심볼과 라인에 대한 정보가 저장된 리스트의 헤드이다.

1. 코드
   1. 20181618.h

#ifndef \_20181618

#define \_20181618

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys /stat.h>

#include <unistd.h>

#include <dirent.h>

//----------- symbol의 정보가 저장되는 구조체 ----------

typedef struct sym{

char name[15]; // symbol

int loc;

struct sym \*next;

}sym;

// ---------- line별 정보가 저장되는 구조체 -------------

typedef struct line{

int l\_num, loc, fmat, modi;

//fmat은 포멧번호, 0이면 byte word, -1이면 기재x, -2면 주석

char obj[10]; // object code

struct line \*next;

}line;

//==========PASS1===========

// pass1을 구현한 함수이다.

// 라인별로 loc를 계산하여 저장한다.

//==========================

int pass1();

//==============PASS2===============

// pass2을 구현한 함수이다.

// 라인별로 object code를 생성하여 저장한다.

//===================================

void pass2();

//=======Assemble 함수=========

// pass1, pass2를 통해 얻은 정보로

// .lst , .obj 파일 생성

//===========================

int assemble();

//=========type 함수=========

// 파일이 존재하는지 확인한 후, 출력

//=========================

void type();

//=========symbol 추가 함수=========

// 리스트에 symbol을 추가한다.

// symbol이 중복될경우 -1을 반환한다.

//===============================

int add\_symbol(char \* str, int num);

//============symbol->loc===========

// symbol과 대응되는 주소를 리스트에서 찾아 반환한다

// 존재하지 않을 경우 -1을 반환한다.

//==================================

int search\_symbol(char \* str);

//======symbol 정렬 뒤 출력======

// symbol 테이블을 내림차순으로 출력.

//===========================

void sort\_symbol();

//=========2's compliment========

// 음수인 10진수 offset과 길이를 받아서

// 보수화한 뒤 16진수 문자열로 반환한다.

//===============================

char \* two\_comp(int n, int b);

//=====2진수 문자열->16진수 문자열=======

// 2진수 문자열을 16진수 문자열 길이를 받아

// 16진수 문자열로 반환한다.

//===============================

char \* biTohex(char \* buf, int b);

//=========아스키코드 문자열=========

// 문자열을 받아와 문자 하나하나에 대응하는

// 아스키코드 값을 다시 문자열로 반환한다.

//=============================

char \* ascii\_str(char \* str);

//===========loc 값 계산 =============

// 해당 라인에 할당된 byte수를 게산하여 반환

//=================================

int loc\_func(char \* str1, char \* str2);

//=========10진수->16진수===========

// 10진수와 길이를 받아 16진수 문자열 반환

//==============================

char \* hex(int n, int b);

//========레지스터 검색========

// 레지스터 테이블에서 번호 반환

//=========================

int regi(char \* str);

//=====opcode table에 없는 명령어 예외처리=======

int op\_check(char \* str);

//========문자열 초기화==========

void obj\_reset(char \* str1, char \* str2, char \* str3, char \* str4);

int sym\_flag =0; // symbol 성공적으로 저장되면 1

int cur\_loc =0; // 현재 loc값 저장

int fLoc =0; // 최종 loc값

int base =0; // base로 지정된 loc

char filename[50];

int fCnt =0; // 파일 끝 번호

char \* reg[] = {"A", "X", "L", "B", "S", "T", "F", "PC", "SW"};

#endif

* 1. 20181618.c

#include <stdio.h >

#include <string.h >

#include <stdlib.h >

#include <sys /types.h >

#include <sys /stat.h >

#include <unistd.h >

#include <dirent.h >

#include "20181618.h“

//==========PASS1===========

// pass1을 구현한 함수이다.

// 라인별로 loc를 계산하여 저장한다.

//==========================

int pass1(){

FILE\* f = fopen(filename, "r");

if(f ==NULL){

printf(" file dosen't exist!\n");

return -1;

}

char buf[50], str1[10], str2[10], str3[10], str4[10];

char \* ptr;

int cnt =1;

int result =0;

int debugCnt =0, start\_loc;

error\_flag=0;

cur\_loc =0;

fLoc =0;

base =0;

fCnt =0;

line\* temp = l\_head;

while(!feof(f)){

fgets(buf, sizeof(buf), f);

debugCnt++;

if(buf[0] =='\n') continue; // 빈 줄

line\* node = (line \*)malloc(sizeof(line));

node->l\_num = cnt ++;

node->modi =0;

node->next =NULL; // 해당 줄에 대한 노드 생성

if(buf[0] =='.'){

node->loc =-2;

temp->next = node;

temp = temp ->next;

continue;

}

if(buf[0] ==' '){ // symbol 없는 줄

////////////////////////////////////line 토큰 나누기

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

str1[0] ='\0';

ptr = strtok(buf, " ");

strcpy(str2, ptr);

ptr = strtok(NULL, " ");

if(ptr !=NULL){

strcpy(str3, ptr);

}

if(str3[strlen(str3)-1] ==','){

str3[strlen(str3)-1] ='\0';

ptr = strtok(NULL, " ");

strcpy(str4, ptr);

}

////////////////////////////////////line 정보 저장

if(op\_check(str2)==5 || op\_check(str2)==6)

node->loc =-1; // BASE, END의 경우 loc x

else

node->loc = cur\_loc;

result = loc\_func(str2, str3);

if(result >=0){

cur\_loc += result; // loc 누적 갱신

}

if(op\_check(str2)==1 || op\_check(str2)==2){

node->fmat =0; // BYTE, WORD는 포맷 없음

}

else if(op\_check(str2)>=3){

if(op\_check(str2)==6) node ->modi =-1;

node->fmat =-1;

// BASE, START, END, RESB, RESW는 object code 없음

}

else{

node->fmat = result;

}

temp->next = node;

temp = temp ->next;

}

else{

////////////////////////////////////line 토큰 나누기

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

ptr = strtok(buf, " ");

strcpy(str1, ptr);

ptr = strtok(NULL, " ");

if(ptr !=NULL){

strcpy(str2, ptr);

}

ptr = strtok(NULL, " ");

if(ptr !=NULL){

strcpy(str3, ptr);

}

if(str3[strlen(str3)-1] ==','){

str3[strlen(str3)-1] ='\0';

ptr = strtok(NULL, " ");

strcpy(str4, ptr);

}

if(op\_check(str2)==7){ // 시작주소 초기화

cur\_loc = hexTodec(str3, 0, 0);

start\_loc = cur\_loc;

}

if(op\_check(str2)==5 || op\_check(str2)==6)

node->loc =-1; // BYTE, WORD는 포맷 없음

else

node->loc = cur\_loc;

result = loc\_func(str2, str3);

if(result >=0){

cur\_loc += result; // loc 누적 갱신

if(add\_symbol(str1, cur\_loc -result)==-1){

error\_flag =1; // symbol 추가 및 중복 체크

}

}

if(op\_check(str2)==1 || op\_check(str2)==2){

node->fmat =0;// BYTE, WORD는 포맷 없음

}

else if(op\_check(str2)>=3){

node->fmat =-1;

// BASE, START, END, RESB, RESW는 object code 없음

}

else{

node->fmat = result;

}

temp->next = node;

temp = temp ->next;

}

if(error\_flag ==1){

printf("[pass1] error in line %d\n", (debugCnt)\*5);

break;

} // error 메세지 출력

}

fLoc = cur\_loc -start\_loc; // 파일의 총 길이

fCnt = cnt -1; // 파일의 끝 줄

fclose(f);

if(error\_flag ==1) return -1;

return 0;

}

//==============PASS2===============

// pass2을 구현한 함수이다.

// 라인별로 object code를 생성하여 저장한다.

//===================================

void pass2(){

FILE\* f = fopen(filename, "r");

char buf[50], temp[20], str1[10], str2[10], str3[10], str4[10];

char \* ptr;

char op[3], nixbpe[7], disp[4], addr[6];

int pc =0;

int i, len, flag, b\_flag, offset, debugCnt =0, t = fCnt;

line\* cur = l\_head; // 현재 라인 정보

line\* ne =NULL; // 다음 라인 정보

while(!feof(f)){

fgets(buf, sizeof(buf), f);

debugCnt++;

if(buf[0] =='\n') continue; // 빈 줄

if(buf[0] =='.'){

fCnt--;

cur = cur ->next;

continue; // 주석

}

cur = cur ->next;

fCnt--;

if(fCnt >1){

ne = cur ->next;

while(ne ->loc <0){

ne = ne ->next;

}

pc = ne ->loc;

} // 현재 줄 기준 program counter 지정

else{

pc =0;

} // 마지막 줄의 program counter는 0

if(fCnt ==0){

cur->loc =-1;

strcpy(cur->obj,"");

continue;

}

////////////////////////////////////line 토큰 나누기

for(int i =0;i <10;i ++){

str1[i]='\0'; str2[i]='\0';

str3[i]='\0'; str4[i]='\0';

}

if(buf[0] ==' '){ // symbol 없는 줄

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

str1[0] ='\0';

ptr = strtok(buf, " ");

strcpy(str2, ptr);

ptr = strtok(NULL, " ");

if(ptr !=NULL) strcpy(str3, ptr);

if(str3[strlen(str3)-1] ==','){

str3[strlen(str3)-1] ='\0';

ptr = strtok(NULL, " ");

strcpy(str4, ptr);

}

}

else{ // symbol 있는 줄

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

ptr = strtok(buf, " ");

strcpy(str1, ptr);

ptr = strtok(NULL, " ");

if(ptr !=NULL) strcpy(str2, ptr);

ptr = strtok(NULL, " ");

if(ptr !=NULL) strcpy(str3, ptr);

if(str3[strlen(str3)-1] ==','){

str3[strlen(str3)-1] ='\0';

ptr = strtok(NULL, " ");

strcpy(str4, ptr);

}

}

////////////////////////////////////obj 코드 생성

obj\_reset(op, nixbpe, disp, addr);

flag =0;

b\_flag =0; // base 플래그

if(strcmp(str2, "BASE")==0){

base = search\_symbol(str3);

continue;

} // BASE의 기준 심볼 loc값 저장

else if(cur ->loc <0) continue; // object code 정보 x

if(cur ->fmat ==0){ // BYTE, WORD

len = strlen(str3);

if(strcmp(str2, "BYTE")==0){

if(str3[0] =='C'){ // 캐릭터 문자

i=2;

while(i <len -1){

str3[i-2] = str3[i];

i++;

}

str3[i-2] ='\0';

strcpy(cur->obj, ascii\_str(str3));

// 해당 문자의 16진수 아스키코드 값이 그대로 저장한다.

}

else{ // 16진수

i=2;

while(i <len -1){

cur->obj[i -2] = str3[i];

i++;

} // 16진수 그대로 저장한다.

}

}

else{ // WORD

strcpy(cur->obj, hex(atoi(str3), 6));

} // 10진수를 16진수로 바꾸어 저장한다.

}

else if(cur ->fmat ==1){

strcpy(cur->obj, hash\_function(str2));

} // 포맷1. opcode를 저장한다.

else if(cur ->fmat ==2){

strcpy(cur->obj, hash\_function(str2));

cur->obj[2] = decTohec(regi(str3));

cur->obj[3] = decTohec(regi(str4));

} // 포맷2. opcode와 register를 저장한다.

else if(cur ->fmat >=3){

strcpy(temp, "");

ptr = strtok(str2, "+");

strcpy(str2, ptr); // 포맷4의 '+' 제거

strcpy(cur->obj, hash\_function(str2));

if(cur ->obj[1]=='0'){

op[0]='0'; op[1]='0';

}

else if(cur ->obj[1]=='4'){

op[1]='1';

}

else if(cur ->obj[1]=='8'){

op[0]='1';

}

else if(cur ->obj[1]=='C'){

op[0]='1'; op[1]='1';

}

op[2] ='\0';

cur->obj[1]='\0';

// opcode(6bit) 정보 저장

if(str3[0] =='#'){ // immediate addressing

nixbpe[1] ='1'; // i=0

ptr = strtok(str3, "#");

strcpy(str3, ptr);

if(str3[0]>='0'&& str3[0]<='9'){

flag =1;

} // [#숫자]와 같이 숫자가 그대로 저장

}

else if(str3[0]=='@'){ // indirect addressing

nixbpe[0] ='1'; // n=1

ptr = strtok(str3, "@");

strcpy(str3, ptr);

}

else{ // simple addressing

nixbpe[0] ='1';

nixbpe[1] ='1';

}

if(strcmp(str4, "X")==0) nixbpe[2] ='1'; // x=1

if(cur ->fmat ==4) nixbpe[5] ='1'; // e=1

if(cur ->fmat ==3){ // displace 및 address

if(flag ==1){

strcpy(disp, hex(atoi(str3), 3));

} // [#10진수숫자]는 16진수로 변환하여 그대로 저장

else{

offset = search\_symbol(str3) - pc;

// 현재 operand의 loc와 pc의 차이값

if(offset >=-2048 && offset <=2047){ // pc

if(offset >=0){ // 양수

strcpy(disp, hex(offset, 3));

}

else{ // 음수. 보수화 단계를 거친다.

strcpy(disp, two\_comp(offset, 3));

}

}

else{ // base

b\_flag=1;

offset = search\_symbol(str3) - base;

// 현재 operand의 loc와 base loc의 차이값

if(strcmp(str3, "")==0) {

strcpy(disp, "000");

flag=1;

} // operand가 없을 경우

else strcpy(disp, hex(offset, 3));

}

}

}

else if(cur ->fmat ==4){ // 포맷4의 경우 operand의 loc값을 바로 저장

if(flag ==1){ // [#숫자]

strcpy(addr, hex(atoi(str3), 5));

}

else{

strcpy(addr, hex(search\_symbol(str3),5));

if(strcmp(str3, "")==0) strcpy(addr, "00000");

}

}

if(flag !=1 && cur ->fmat ==4){

cur->modi =1;

} // modification의 경우

if(flag ==1 || cur ->fmat ==4){

nixbpe[3] ='0'; // b=0

nixbpe[4] ='0'; // p=0

}

else if(b\_flag ==0){

nixbpe[4] ='1'; // p=1

}

else{

nixbpe[3] ='1'; // b=1

}

strcpy(temp, op);

strcat(temp, nixbpe);

strcat(cur->obj, biTohex(temp, 2));

// [op + nixbpe] 12bit 저장

if(cur ->fmat ==3){

strcat(cur->obj, disp);

}

else{

strcat(cur->obj, addr);

}

// [displace] 12bit or [address] 20bit 저장

}

else{

strcpy(cur->obj, "");

} // object code 없는 경우

if( ==1){

printf("[pass2] error in line %d\n", (debugCnt)\*5);

break;

}

}

fCnt=t;

fclose(f);

}

//=======Assemble 함수=========

// pass1, pass2를 통해 얻은 정보로

// .lst , .obj 파일 생성

//===========================

int assemble(){

FILE\* fr;

FILE\* fw;

char t\_filename[50];

char code\_name[10];

char \* ptr;

if(pass1()==0) pass2();

if(error\_flag ==1) return -1;

fr = fopen(filename, "r");

ptr = strtok(filename, ".");

strcpy(filename, ptr);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*lst파일 생성

sprintf(t\_filename, "%s.lst", filename);

fw = fopen(t\_filename, "w");

char buf[50], str1[10], str2[10], str3[10];

line\* cur = l\_head;

while(!feof(fr)){

fgets(buf, sizeof(buf), fr);

if(buf[0] =='\n') continue; // 빈 줄

if(buf[0] =='.'){

fprintf(fw, "%-5d", (cur ->l\_num)\*5);

fprintf(fw, "%s", buf);

cur = cur ->next;

continue; // 주석

}

cur = cur ->next;

////////////////////////////////////line 토큰 나누기

for(int i =0;i <10;i ++){

str1[i]='\0'; str2[i]='\0';

str3[i]='\0';

}

if(buf[0] ==' '){ // symbol 없는 줄

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

str1[0] ='\0';

ptr = strtok(buf, " ");

strcpy(str2, ptr);

ptr = strtok(NULL, " ");

if(ptr !=NULL) strcpy(str3, ptr);

if(str3[strlen(str3)-1] ==','){

str3[strlen(str3)] =' ';

str3[strlen(str3)+1] ='\0';

ptr = strtok(NULL, " ");

strcat(str3, ptr);

}

}

else{ // symbol 있는 줄

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

ptr = strtok(buf, " ");

strcpy(str1, ptr);

ptr = strtok(NULL, " ");

if(ptr !=NULL) strcpy(str2, ptr);

ptr = strtok(NULL, " ");

if(ptr !=NULL) strcpy(str3, ptr);

if(str3[strlen(str3)-1] ==','){

str3[strlen(str3)] =' ';

str3[strlen(str3)+1] ='\0';

ptr = strtok(NULL, " ");

strcat(str3, ptr);

}

if(op\_check(str2)==7) strcpy(code\_name, str1);

// code 이름 저장

}

////////////////////////////////////출력

fprintf(fw, "%-5d", (cur ->l\_num)\*5);

fprintf(fw, "%s ", hex(cur ->loc, 4));

fprintf(fw, "%-10s", str1);

fprintf(fw, "%-10s", str2);

fprintf(fw, "%-15s", str3);

fprintf(fw, "%-15s\n", cur ->obj);

}

fclose(fw);

fclose(fr);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*obj 파일 생성

sprintf(t\_filename, "%s.obj", filename);

fw = fopen(t\_filename, "w");

int first =0, len;

char obj\_buf[200];

cur = l\_head ->next;

fprintf(fw, "H%-6s%s%s\n", code\_name, hex(cur ->loc,6), hex(fLoc, 6));

cur=cur ->next;

while(cur ->next !=NULL){

strcpy(obj\_buf, "");

len=0;

if(cur ->modi ==-1) break; // END의 경우 종료한다

else if(strcmp(cur ->obj, "") ==0){

cur = cur ->next;

continue;

} // 주석

fprintf(fw, "T%-6s", hex(cur ->loc, 6));

first = cur ->loc;

while(1){

if(cur ->modi ==-1) break; // END

if(cur ->loc <0){

cur = cur ->next;

continue;

} // 주석 및 loc가 배정되지 않은 줄

if(strcmp(cur ->obj, "") ==0){

break;

}

len = cur ->loc -first;

strcat(obj\_buf, cur->obj);

cur = cur ->next;

if(len >25) break; // 한 줄에 25 ~ 30byte 출력

}

len = cur ->loc - first;

if(cur ->modi ==-1){

len = fLoc - first;

}

fprintf(fw, "%s%s\n", hex(len, 2), obj\_buf);

}

cur = l\_head ->next;

while(cur !=NULL){

if(cur ->modi ==1){

fprintf(fw, "M%s05\n", hex(cur ->loc +1, 6));

}

cur = cur ->next;

} // modification 출력

fprintf(fw, "E%s\n",hex((l\_head ->next)->loc, 6));

fclose(fw);

return 0;

}

//=========type 함수=========

// 파일이 존재하는지 확인한 후, 출력

//=========================

void type(){

FILE\* f =NULL;

char buf;

f = fopen(filename, "r");

if(f ==NULL) printf(" file doesn't exist\n"); //파일이 존재하지 않을경우 error

else{

while(fscanf(f, "%c", &buf) != EOF) printf("%c",buf);

printf("\n");

}

}

//=========symbol 추가 함수=========

// 리스트에 symbol을 추가한다.

// symbol이 중복될경우 -1을 반환한다.

//===============================

int add\_symbol(char \* str, int num){

sym\* temp = symbol;

sym\* node = (sym \*)malloc(sizeof(sym));

strcpy(node->name, str);

node->loc = num;

node->next =NULL;

while(temp ->next !=NULL){

temp = temp ->next;

if(strcmp(temp ->name, node ->name)==0) return -1; // 중복되는 경우

}

temp->next = node;

return 0;

}

//======symbol 정렬 뒤 출력======

// symbol 테이블을 내림차순으로 출력.

//===========================

void sort\_symbol(){

char \*\* temp;

int \* tLoc;

int \* idx;

sym\* cur = symbol;

int i,j,t, cnt =0;

while(cur ->next !=NULL){

cur=cur ->next;

cnt++;

}// 전체 심볼 수 체크

temp = (char \*\*)malloc(sizeof(char \*)\*cnt);

tLoc = (int \*)malloc(sizeof(char \*)\*cnt);

idx = (int \*)malloc(sizeof(int)\*cnt);

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

temp[i] = (char \*)malloc(sizeof(char)\*20);

idx[i]=i;

} // 동적할당

cur = symbol;

i=0;

while(cur ->next !=NULL){

cur=cur ->next;

strcpy(temp[i], cur->name);

tLoc[i] = cur ->loc;

i++;

}// 심볼 문자열 카피

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

for(j =i +1;j <cnt;j ++){

if(strcmp(temp[idx[i]], temp[idx[j]])<=0){

t=idx[i]; idx[i]=idx[j]; idx[j]=t;

}

}

} // 문자열과 인덱스 배열을 맵핑하여 문자열의 크기를 비교한다.

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

printf("\t%s\t%s\n", temp[idx[i]], hex(tLoc[idx[i]], 4));

}

free(temp);

free(idx);

free(tLoc); //동적할당한 배열 free

}

//============symbol->loc===========

// symbol과 대응되는 주소를 리스트에서 찾아 반환한다

// 존재하지 않을 경우 -1을 반환한다.

//==================================

int search\_symbol(char \* str){

sym\* temp = symbol ->next;

if(strcmp(str, "")==0) return 0;

while(temp !=NULL){

if(strcmp(temp ->name, str)==0) return temp ->loc;

temp = temp ->next;

}

error\_flag=1;

return -1;

}

//=========2's compliment========

// 음수인 10진수 offset과 길이를 받아서

// 보수화한 뒤 16진수 문자열로 반환한다.

//===============================

char \* two\_comp(int n, int b){

char str[4];

char buf[13];

int i, idx,temp =0;

n \*=-1;

strcpy(str,hex(n, b));

idx =0;

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

temp = hexTodec("", str[i], 1);

buf[idx++] = ((temp /8)+1)%2 +'0';

temp -= (temp /8)\*8;

buf[idx++] = ((temp /4)+1)%2 +'0';

temp -= (temp /4)\*4;

buf[idx++] = ((temp /2)+1)%2 +'0';

temp -= (temp /2)\*2;

buf[idx++] = (temp +1)%2 +'0';

} // 2진수

buf[b\*4 -1] +=1;

for(i =b \*4 -1;i >=0;i --){

if(buf[i]>'1'){

buf[i] -=2;

buf[i-1] +=1;

}

} // 보수화

return biTohex(buf, b);

}

//=====2진수 문자열->16진수 문자열=======

// 2진수 문자열을 16진수 문자열 길이를 받아

// 16진수 문자열로 반환한다.

//===============================

char \* biTohex(char \* buf, int b){

char \* str =malloc(sizeof(char)\*4);

int i, idx =0;

int n;

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

n = (buf[idx ++]-'0')\*8;

n += (buf[idx ++]-'0')\*4;

n += (buf[idx ++]-'0')\*2;

n += (buf[idx ++]-'0');

str[i] = decTohec(n);

}

str[i] ='\0';

return str;

}

//=========아스키코드 문자열=========

// 문자열을 받아와 문자 하나하나에 대응하는

// 아스키코드 값을 다시 문자열로 반환한다.

//=============================

char \* ascii\_str(char \* str){

int i;

int temp;

int len = strlen(str);

char s[3];

char \* a\_str =malloc(sizeof(char)\*(len \*2 +1));

a\_str[0] ='\0';

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

temp = str[i];

strcpy(s, hex(temp, 2));

strcat(a\_str, s);

}

a\_str[len\*2] ='\0';

return a\_str;

}

//===========loc 값 계산 =============

// 해당 라인에 할당된 byte수를 게산하여 반환

//=================================

int loc\_func(char \* str2, char \* str3){

int i =0;

int len = strlen(str3);

int n = op\_check(str2);

char \* ptr;

char tstr[10];

if(n ==1){ // BYTE

i=2;

while(i <len -1){

tstr[i-2] = str3[i];

i++;

}

if(str3[0] =='C'){ // 문자 하나당 1byte

return strlen(tstr);

}

else if(str3[0] =='X'){ // 16진수

return (strlen(tstr)+1)/2;

}

}

else if(n ==2){ // WORD

return 3;

}

else if(n ==3){ // RESB

return atoi(str3);

}

else if(n ==4){ // RESW

return atoi(str3)\*3;

}

else if(n ==5 || n ==6){ //END, BASE

return -1;

}

else if(n ==7){ // START

return 0;

}

else{

//str2에 +면 4형식

//나머진 해쉬검색해서 나오는 값 type만큼 더함

if(str2[0] =='+'){

ptr = strtok(str2, "+");

strcpy(str2, ptr);

if(strcmp(hash\_function(str2), "\0") !=0) return 4;

}

int key = (str2[0]-'A')%21;

hash\* temp = h\_table[key];

while(temp !=NULL){

if(strcmp(str2, temp ->mnemonic) ==0){

return temp ->type;

}

temp = temp ->next;

}

error\_flag=1;

return -2; // 아무것도 반환되지않으면 없는 instruction이므로 error

}

return -2;

}

//=========10진수->16진수===========

// 10진수와 길이를 받아 16진수 문자열 반환

//==============================

char \* hex(int n, int b){

int i, temp, a;

char \* str =malloc(sizeof(char)\*(b +1));

if(n <0){

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

str[i] =' ';

}

str[i] ='\0';

return str;

}

a=1;

for(i =0;i <b -1;i ++){

a \*=16;

}

temp = n;

for(i =0;i <b -1;i ++){

str[i] = decTohec(temp /a);

temp -= (temp /a)\*a;

a /=16;

}

str[i] = decTohec(temp);

str[i+1] ='\0';

return str;

}

//========레지스터 검색========

// 레지스터 테이블에서 번호 반환

//=========================

int regi(char \* str){

int cnt =0;

int idx =0;

while(cnt <10){

if(strcmp(reg[idx], str)==0) return cnt;

if(cnt ==6) cnt +=2;

else cnt +=1;

idx++;

}

if(strcmp(str, "")==0) return 0;

error\_flag =1;

return 0; // 저장된 레지스터가 아닐경우

}

//=====opcode table에 없는 명령어 예외처리=======

int op\_check(char \* str){

if(strcmp(str, "BYTE")==0) return 1;

else if(strcmp(str, "WORD")==0) return 2;

else if(strcmp(str, "RESB")==0) return 3;

else if(strcmp(str, "RESW")==0) return 4;

else if(strcmp(str, "BASE")==0) return 5;

else if(strcmp(str, "END")==0) return 6;

else if(strcmp(str, "START")==0) return 7;

else return 0;

}

//========문자열 초기화==========

void obj\_reset(char \* op, char \* nixbpe, char \* disp, char \* addr){

int i;

for(i =0;i <3;i ++){

op[i] ='0';

}

op[6] ='\0';

for(i =0;i <6;i ++){

nixbpe[i] ='0';

}

nixbpe[6] ='\0';

for(i =0;i <3;i ++){

disp[i] ='0';

}

disp[3] ='\0';

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

addr[i] ='0';

}

addr[5] ='\0';

return ;

}