SYSTEM SOFTWARE

SIC Assembler

1조

200820299 양성혁

201311753 장용성

201511779 강찬우

201511784 권신영

팀 역할 분담

Pass1

1. sic 파일 input (error flag 처리): 파일이 없을 경우 에러

2. 주소 할당과 symbol 테이블 생성

<1> 코멘트일 경우 무시

<2> 처리하는 명령어에 label이 있을 경우 symbol table 탐색

동일한 label 이름이 있을 경우 에러(error flag 처리)

동일한 label 이름이 없을 경우 symbol table에 label의 이름과 주소 저장

<3> 레이블이 없을 경우 주소 할당

optable 탐색

(명령어가 optable에 없거나 word, resw, resb, byte에 해당하지 않는 명령어인 경우 에러) (error flag 처리)

3. 중간 파일 output

Pass2

4. 중간 파일 input, 헤더 레코드 작성

5. 텍스트 레코드 작성

<1> 코멘트일 경우 무시

<2> optable 참조해서 opcode 찾기

<3> 명령어의 operand로 label이 발견되면 symbol table에서 label 확인

- 찾으면 label의 주소 값을 operand의 주소에 저장.

- 못 찾으면 operand의 주소 값에 0 저장 (error flag 처리)

<4> 명령어의 operand로 symbol의 label이 발견되지 않으면 operand에 0 저장

<5> 명령어에 BYTE 나 WORD 가 있으면 대응되는 값 저장

\* 만약 쓰고 있는 텍스트 레코드의 길이가 길어지면 현재 텍스트 레코드를 오브젝트 파일에 저장하고 새 텍스트 레코드 라인 생성

반복하다가 마지막 텍스트 레코드를 object file에 저장.

6. 엔드 레코드 작성

권신영: pass1- 1

장용성: pass1- 2, 3

강찬우: pass2- 어셈블 과정

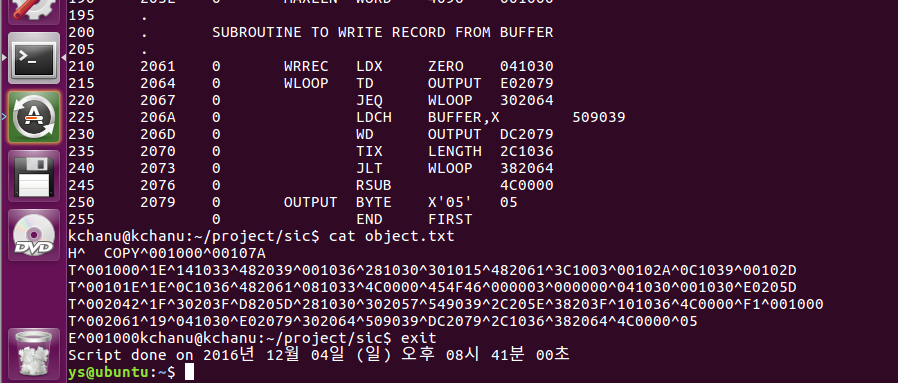
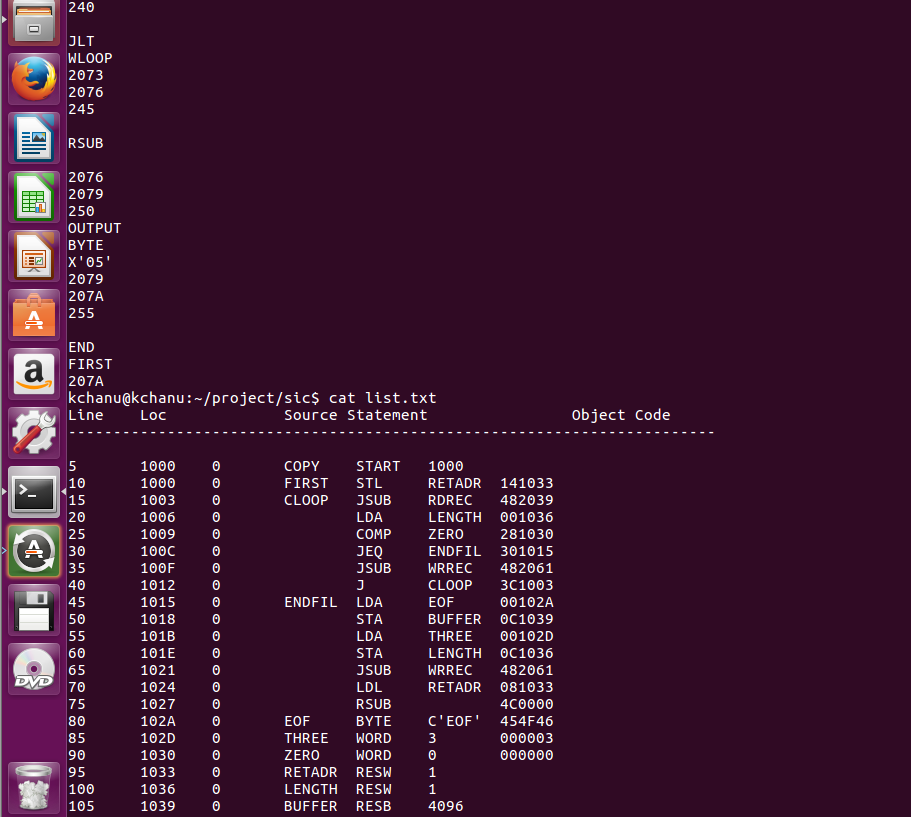
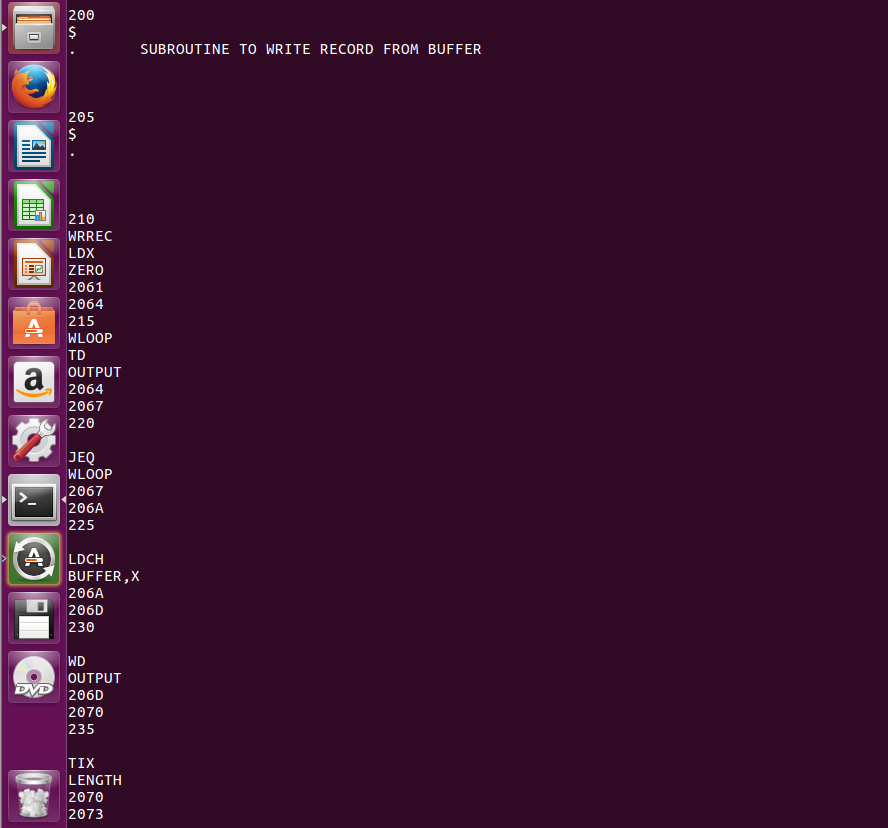
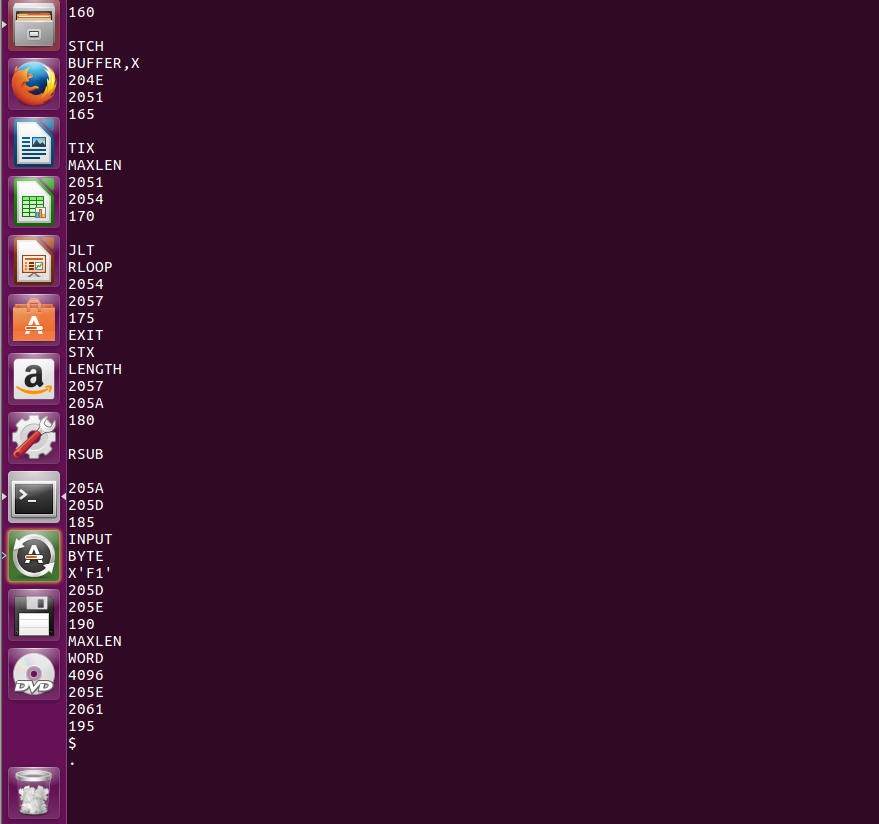
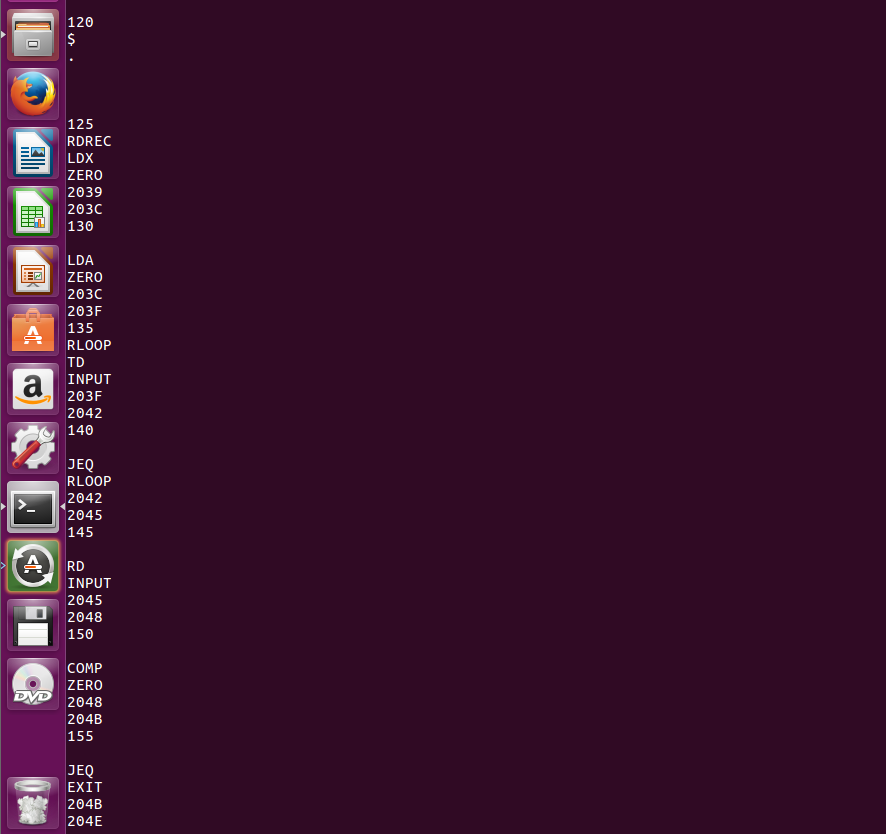
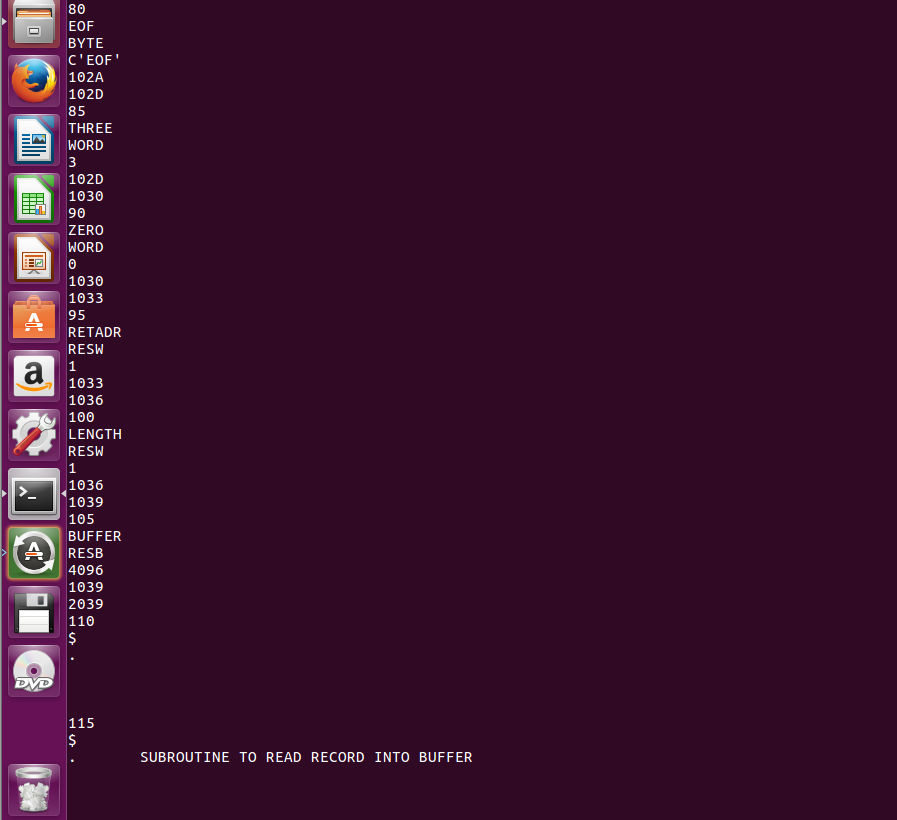
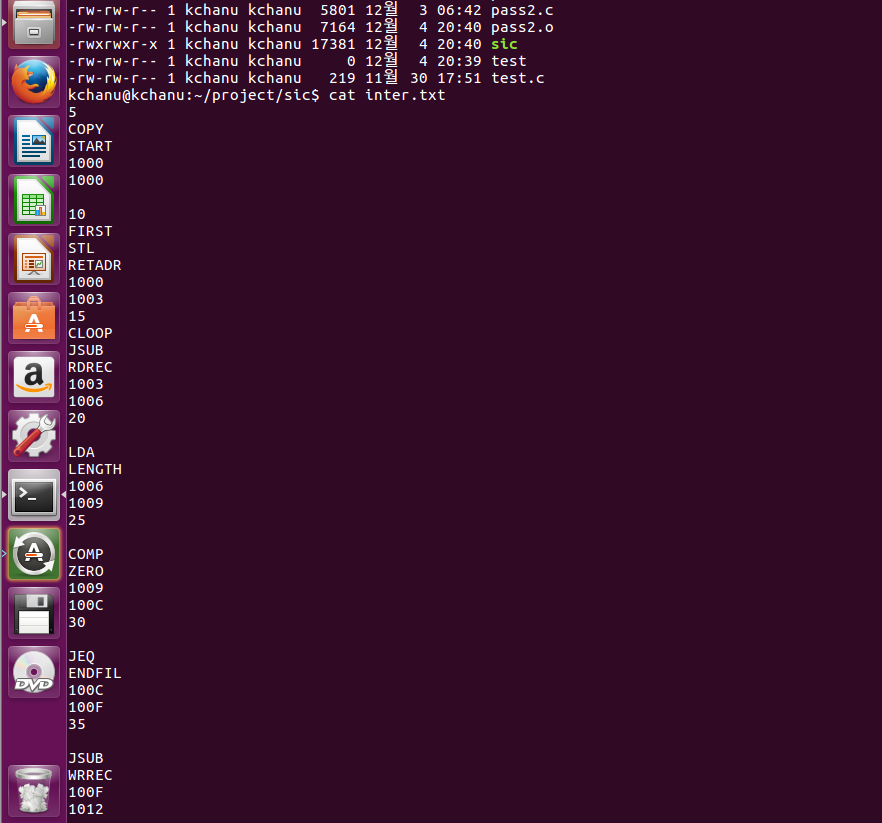
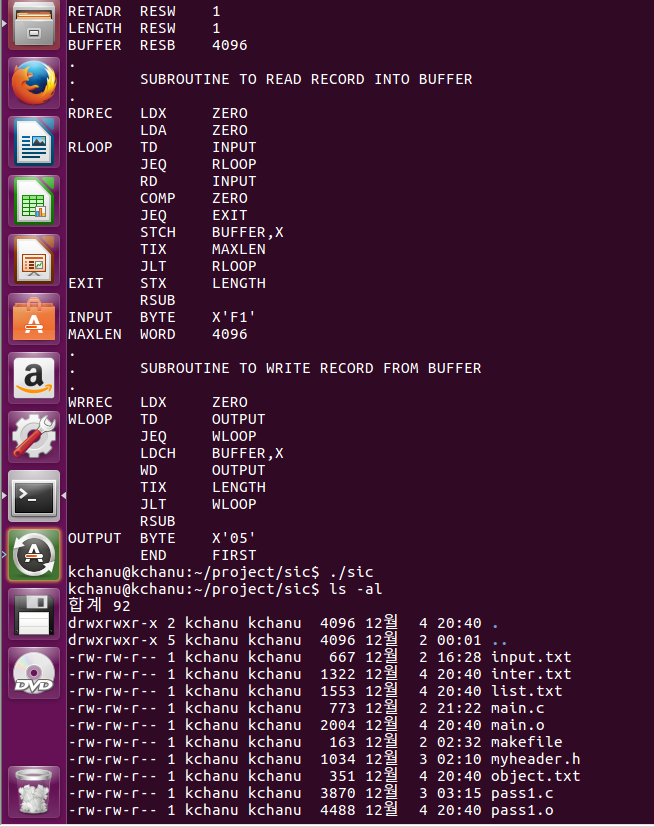
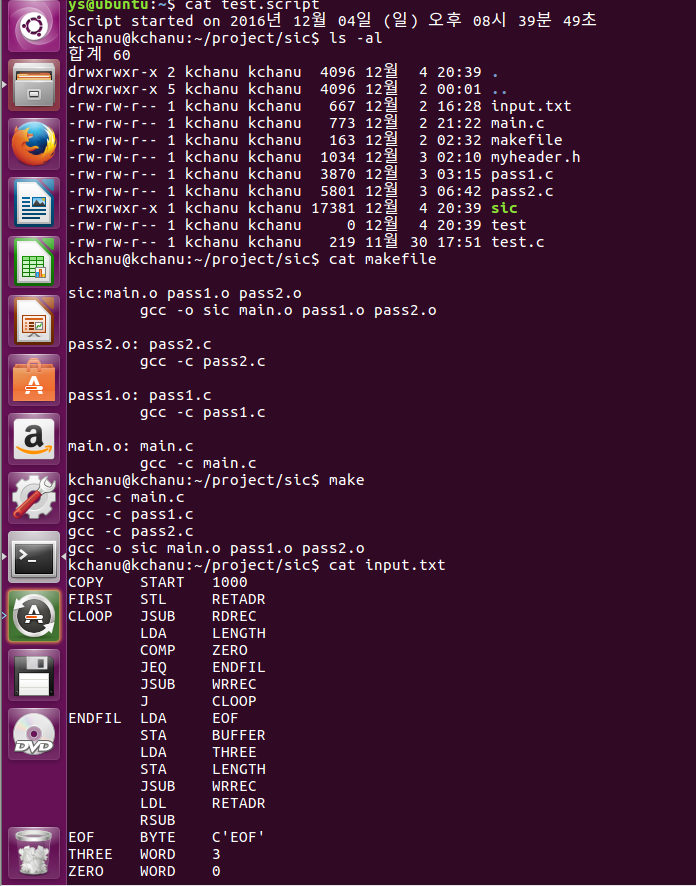
양성혁: pass2- 어셈블된 코드로 오브젝트 프로그램 작성

사용법

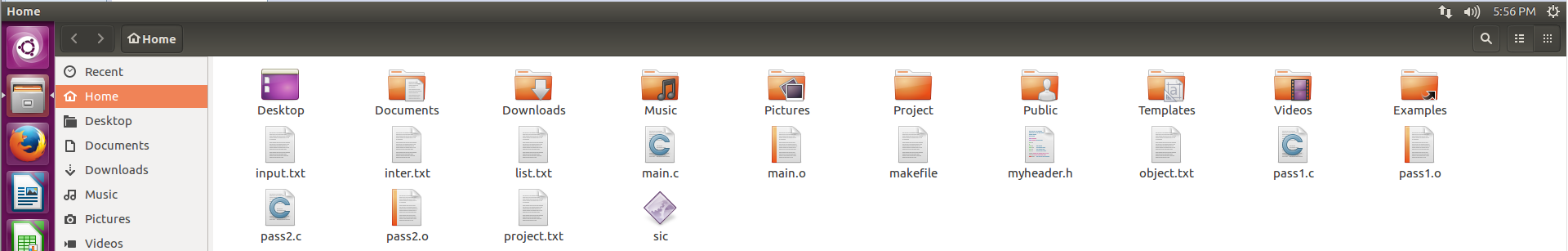
Input.txt에 SIC원시프로그램을 먼저 저장 후 sic프로그램을 실행하면 object.txt에

해당 원시프로그램의 목적프로그램이 저장됩니다.

Script file



Screen capture file



Source code

Myheader.h

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct opTab {

char mnemonic[7];

char opcode[7];

} opTab;

typedef struct symTab {

char symbol[10];

int addr;

} symTab;

typedef struct record {

char label[7], mnemonic[10], operand[10], commentLine[50];

} record;

typedef struct imline{

char line[10], label[10], mnemonic[50], operand[20], loc[7], pc[7], objcode[7];

} imline;

typedef int errorflag;

extern errorflag ef;

extern void openfile(FILE \*\* fp, char \* filename, char \* mode);

extern opTab myOpTab[26];

extern symTab mySymTab[20];

extern record rec;

extern imline line;

extern char startAddr[7];

extern int LOCCTR, SymCount;

extern int length;

extern void PASS1(void);

extern void PASS2(void);

pass1.c

#include"myheader.h"

void chk\_label();

void chk\_mnemonic();

void READ\_LINE();

symTab mySymTab[20];

int length;

char startAddr[7];

int lineNumber = 0;

int LOCCTR = 0;

int PCCTR = 0;

int symCount = 0;

record rec;

FILE \*input;

void PASS1(void)

{

int i;

FILE \*inter;

openfile(&input, "input.txt", "r");

openfile(&inter, "inter.txt", "w");

READ\_LINE();

if (strcmp(rec.mnemonic, "START") == 0) //if OPCODE = 'START'

{

strcpy(startAddr, rec.operand);//save #[OPERAND] as starting address

LOCCTR = strtoul(startAddr, NULL, 16); //initialize LOCCTR to starting address

fprintf(inter, "%d\n%s\n%s\n%s\n%X\n\n", lineNumber, rec.label, rec.mnemonic, rec.operand, LOCCTR); //write line to intermediate file

READ\_LINE();

}

else

{

strcpy(startAddr, "0");

LOCCTR = 0; //initialize LOCCTR to 0

}

PCCTR = LOCCTR;

while (strcmp(rec.mnemonic, "END") != 0) { //OPCODE < > 'END'

if(strcmp(rec.label, "$") != 0){ // if this is not a comment line

chk\_label(); //search SYMTAB for LABEL

chk\_mnemonic(); //search OPTAB for OPCODE

fprintf(inter, "%d\n%s\n%s\n%s\n%X\n%X\n", lineNumber, rec.label, rec.mnemonic, rec.operand, LOCCTR, PCCTR); //write line to intermediate file

LOCCTR = PCCTR;

}

else

{

fprintf(inter, "%d\n%s\n%s\n\n\n", lineNumber, rec.label, rec.commentLine);

}

READ\_LINE();

}

fprintf(inter, "%d\n%s\n%s\n%s\n%X\n", lineNumber, rec.label, rec.mnemonic, rec.operand, LOCCTR);

fclose(inter);

fclose(input);

}

void chk\_label()

{

int i;

if(rec.label[0] != '\0'){

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

if (strcmp(rec.label, mySymTab[i].symbol) == 0)

{

ef = 1;

return;

}

}

strcpy(mySymTab[symCount].symbol, rec.label);

mySymTab[symCount++].addr = LOCCTR;

}

}

void chk\_mnemonic() {

int i = 0, found = 0, l = 0;

for (i = 0; i < 26; i++)

{

if (strcmp(rec.mnemonic, myOpTab[i].mnemonic) == 0) //found

{

//add 3 to LOCCTR

PCCTR += 3;

found = 1;

break;

}

}

if (found==0)

{

if (strcmp(rec.mnemonic, "WORD") == 0) //OPCODE = 'WORD'

PCCTR += 3; //add 3 to LOCCTR

else if (strcmp(rec.mnemonic, "RESW") == 0) //OPCODE = 'RESW'

PCCTR += (3 \* atoi(rec.operand)); //add 3\*#[OPERAND] to LOCCTR

else if (strcmp(rec.mnemonic, "RESB") == 0) //OPCODE = 'RESB'

PCCTR += atoi(rec.operand); //add #[OPERAND] to LOCCTR

else if (strcmp(rec.mnemonic, "BYTE") == 0) //OPCODE = 'BYTE'

{

if (rec.operand[0] == 'C') //if BYTE is CHARACTER

{

while (rec.operand[l] != '\0')

l++;

PCCTR = PCCTR + l - 3;

}

else if (rec.operand[0] == 'X') //BYTE

{

while (rec.operand[l] != '\0')

l++;

PCCTR = PCCTR + (l - 3)/2;

}

else { //set error flag(invalid operation code)

ef = 2;

}

}

}

}

void READ\_LINE() {

char buffer[100];

char \* token;

strcpy(rec.label, "\0");

strcpy(rec.mnemonic, "\0");

strcpy(rec.operand, "\0");

fgets(buffer, 100, input);

if(buffer[0] == ';'){

return;

}

if (strchr(buffer, '.') != NULL){

strcpy(rec.label, "$");

strcpy(rec.commentLine, buffer);

lineNumber += 5;

return;

}

token = strtok(buffer, "\n");

token = strtok(buffer, ";");

if(token[0] != '\t'){

token = strtok(token, "\t");

strcpy(rec.label, token);

token = strtok(NULL, "\t");

}

else

token = strtok(token, "\t");

strcpy(rec.mnemonic, token);

token = strtok(NULL, "\t");

if(token != NULL){

strcpy(rec.operand, token);

}

lineNumber += 5;

}

Pass2.c

#include "myheader.h"

opTab myOpTab[26] = {

{ "ADD","0x18" },{ "AND","0x40" },

{ "COMP","0x28" },{ "DIV","0x24" },

{ "J","0x3C" },{ "JEQ","0x30" },

{ "JGT","0x34" },{ "JLT","0x38" },

{ "JSUB","0x48" },{ "LDA","0x00" },

{ "LDCH","0x50" },{ "LDL","0x08" },

{ "LDX","0x04" },{ "MUL","0x20" },

{ "OR","0x44" },{ "RD","0xD8" },

{ "RSUB","0x4C" },{ "STA","0x0C" },

{ "STCH","0x54" },{ "STL","0x14" },

{ "STSW","0xE8" },{ "STX","0x10" },

{ "SUB","0x1C" },{ "TD","0xE0" },

{ "TIX","0x2C" },{ "WD","0xDC" }

};

/\*\*\* comment check function should be extended \*\*\*/

imline line; // record structure

FILE \* object; //output program

FILE \* list; //listing file

FILE \*inter;

char filename[7];

/\*

static void openfile(FILE \*\* fp, char \* filename, char \* mode){

if((\*fp=fopen(filename,mode)) == NULL){

fprintf(stderr, "\*\*\* Opening file [%s] failed! \*\*\*\n", filename);

errorFlag = 1;

}

}

\*/

int xcheck(){

int i;

for(i=0;i<strlen(line.operand);i++){

if(line.operand[i] == ','){

return 1;

}

}

return 0;

}

void assemble(int \*x, int \*operval, int \*i){

sprintf(line.objcode, "%02X%04X", (unsigned int)strtoul(myOpTab[\*i].opcode, NULL, 16), (\*x) \* 32768 + (\*operval));

}

int wtext(){

/\*

while(1){

rdmnemonic(fint);

}

l=len;-

fscanf(fint,"%s%s%s%s",add,label,mne,operand);

fprintf(fobj,"T^00%s^",add);

pos=ftell(fobj);

fprintf(fobj,"--");\*/

int i, j;

int optablen = sizeof(myOpTab) / sizeof(myOpTab[0]);

int symtablen = sizeof(mySymTab) / sizeof(mySymTab[0]);

int operval;

int x;

int maxTextLen = 31;

int curTextLen = 0;

int textlenloc;

char operand[10];

char \*tmp;

//initialize first text record

fprintf(object, "T^");

for(i=strlen(line.loc);i<=6;i++){

fprintf(object, "0");

}

fprintf(object, "%s^", line.loc);

textlenloc = ftell(object);

fprintf(object, "00");

while(strcmp(line.mnemonic,"END")!=0){ //while OPCODE != 'END' do

curTextLen = 0;

for(i=0;i<optablen;i++){ //search OPTAB for OPCODE

if(strcmp(line.mnemonic,myOpTab[i].mnemonic)==0){ //if found then

j = i;

if(line.operand != NULL){ //if there is a symbol in OPERAND field then

strcpy(operand, line.operand); //check if using x register

if((x = xcheck()) == 1){

tmp = strtok(operand, ",");

strcpy(operand, tmp);

}

operval = 0;

for(i=0;i<symtablen;i++){ //search SYMTAB for OPERAND

if(strcmp(operand,mySymTab[i].symbol)==0){ //if found then

curTextLen += 3;

operval = mySymTab[i].addr; //store symbol value as operand address

break;

}

}

if(i == symtablen){

printf("No symbol founded! : %s", operand);

ef = 3;

return;

}

} else {

operval = 0; //store 0 as operand address

}

assemble(&x, &operval, &j); //assemble the object code instruction

break;

}

}

if((strcmp(line.mnemonic,"BYTE")==0)||(strcmp(line.mnemonic,"WORD")==0)){ //else if OPCODE = 'BYTE' or 'WORD' then

if(strcmp(line.mnemonic,"WORD")==0){

curTextLen += 3;

} else {

curTextLen += 1;

}

if(line.operand[0] == 'C'){

char tmp[4];

for(i=2;line.operand[i] != '\'';i++){

tmp[i-2] = line.operand[i];

}

sprintf(line.objcode, "%02X%02X%02X", tmp[0], tmp[1], tmp[2]);

}

else if(line.operand[0] == 'X'){

line.objcode[0] = line.operand[2];

line.objcode[1] = line.operand[3];

line.objcode[2] = '\0';

}

else{

sprintf(line.objcode, "%06X", atoi(line.operand));

}

}

if(strcmp(line.mnemonic,"RESW")==0||strcmp(line.mnemonic,"RESB")==0){

while(strcmp(line.mnemonic,"RESW")==0||strcmp(line.mnemonic,"RESB")==0){

strcpy(line.objcode, "");

wlist();

rdline();

printf("sdf");

}

}

if(maxTextLen < curTextLen){ //if object code will not fit into the current Text record then

fprintf(object, "^%s", line.objcode);

/\* fseek, ftell\*/

fseek(object, textlenloc, SEEK\_SET);

fprintf(object, "%02X", curTextLen);

fseek(object, 0, SEEK\_END);

fprintf(object, "\nT^");

for(i=strlen(line.pc);i<=6;i++){

fprintf(object, "0");

}

fprintf(object, "%s^", line.loc);

textlenloc = ftell(object);

fprintf(object, "00");

curTextLen = 0;

}

fprintf(object, "^%s", line.objcode);

if(strcmp(line.label,"$") == 0){ //if this is not a comment line continue;

fprintf(list, "%s\t%s\n", line.line, line.mnemonic);

rdline();

continue;

}/\*

if(strcmp(line.mnemonic, "$") == 0){

fprintf(list, "%s\t%s\n", line.mnemonic, line.operand);

rdline();

continue;

}\*/

wlist();

rdline();

}

strcpy(line.objcode, "");

strcpy(line.loc, "");

}

void wend(){

fprintf(object, "\nE^%06X", (unsigned int)strtoul(startAddr,NULL,16));

}

void wheader(){

fprintf(object, "H^%6s^%06X^%06X\n", filename, (unsigned int)strtoul(startAddr,NULL,16), length);

}

int initlist(){

fprintf(list, "Line\tLoc\t\tSource Statement\t\tObject Code\n");

fprintf(list, "------------------------------------------------------------------------\n\n");

}

int wlist(){

fprintf(list, "%s\t%s\t0\t%s\t%s\t%s\t%s\n", line.line, line.loc, line.label, line.mnemonic, line.operand, line.objcode);

}

int rdline(){

char \*tmp;

if(fgets(line.line, 10, inter) != NULL){

fgets(line.label, 10, inter);

fgets(line.mnemonic, 50, inter);

fgets(line.operand, 10, inter);

fgets(line.loc, 10, inter);

fgets(line.pc, 10, inter);

}

if((tmp = strstr(line.line, "\n"))!= NULL) \*tmp = '\0';

if((tmp = strstr(line.label, "\n"))!= NULL) \*tmp = '\0';

if((tmp = strstr(line.mnemonic, "\n"))!= NULL) \*tmp = '\0';

if((tmp = strstr(line.operand, "\n"))!= NULL) \*tmp = '\0';

if((tmp = strstr(line.loc, "\n"))!= NULL) \*tmp = '\0';

if((tmp = strstr(line.pc, "\n"))!= NULL) \*tmp = '\0';

}

void PASS2(void){

openfile(&inter, "inter.txt", "r");

openfile(&object, "object.txt", "w");

openfile(&list, "list.txt", "w");

rdline();

initlist();

strcpy(line.objcode, "");

if(strcmp(line.mnemonic, "START") == 0){

wlist();

strcpy(filename, line.label);

rdline();

}

//write Header record to object program

wheader();

//initialize first Text record

//init\_record();

//write text record and write listing

wtext(); // listing ( object )

//write end record and write listing

wend();

wlist();

fclose(inter);

fclose(object);

fclose(list);

//pass2 end

}

Main.c

#include "myheader.h"

errorflag ef = 0;

void openfile(FILE \*\* fp, char \* filename, char \* mode){

if((\*fp=fopen(filename,mode)) == NULL){

fprintf(stderr, "\*\*\* Opening file [%s] failed! \*\*\*\n", filename);

ef = 4;

}

}

int errorCheck(void) {

if (ef){

printf("pass stopped by ef [");

switch(ef){

case 1:

printf("1:duplicate symbol]\n");

return 1;

case 2: printf("2:invalid operation]\n");

return 2;

case 3: printf("3;undefined symbol]\n");

return 3;

default:

printf("unknown error]\n");

return 4;

}

}

return 0;

}

int main(void){

PASS1();

if(errorCheck())

return -1;

length = LOCCTR - strtoul(startAddr, NULL, 16); //save (LOCCTR - starting address) as program length

PASS2();

if(errorCheck())

return -2;

return;

}

Input

COPY START 1000

FIRST STL RETADR

CLOOP JSUB RDREC

LDA LENGTH

COMP ZERO

JEQ ENDFIL

JSUB WRREC

J CLOOP

ENDFIL LDA EOF

STA BUFFER

LDA THREE

STA LENGTH

JSUB WRREC

LDL RETADR

RSUB

EOF BYTE C'EOF'

THREE WORD 3

ZERO WORD 0

RETADR RESW 1

LENGTH RESW 1

BUFFER RESB 4096

.

. SUBROUTINE TO READ RECORD INTO BUFFER

.

RDREC LDX ZERO

LDA ZERO

RLOOP TD INPUT

JEQ RLOOP

RD INPUT

COMP ZERO

JEQ EXIT

STCH BUFFER,X

TIX MAXLEN

JLT RLOOP

EXIT STX LENGTH

RSUB

INPUT BYTE X'F1'

MAXLEN WORD 4096

.

. SUBROUTINE TO WRITE RECORD FROM BUFFER

.

WRREC LDX ZERO

WLOOP TD OUTPUT

JEQ WLOOP

LDCH BUFFER,X

WD OUTPUT

TIX LENGTH

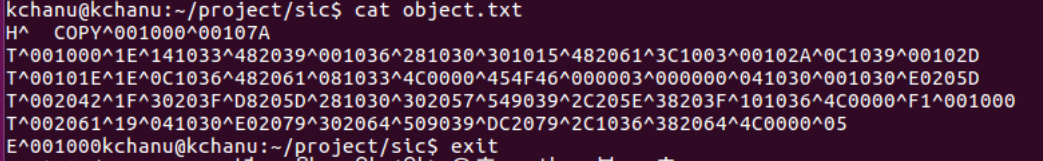
JLT WLOOP

RSUB

OUTPUT BYTE X'05'

END FIRST

Output



학기 말 프로젝트 후기

개인이 혼자 하기에는 힘들 수 있는 프로젝트 였지만 팀 프로젝트로서 팀원과의 협력을 통해서 대략 3주에 걸쳐서 완성을 했습니다. 먼저 역할 분담을 한 뒤 각자의 역할을 수행했으며 수행 중 모르는 것은 서로에게 물어보며 진행했습니다. 일주일에 최소 한번씩은 직접 모여서 작업을 했고 그 외 시간에는 카카오톡을 통해서 진행과정 및 수정내용을 토의했습니다. 팀원들 모두가 참여하는 팀 프로젝트였고 개인적인 프로그래밍이 아닌 여러 사람과 공동으로 작업하는 프로그래밍이었고 사회에서 느낄 경험을 미리 간략하게 경험 할 수 있는 뜻 깊은 시간이었습니다.