##2 pass assembler pass 1

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

char symbol[10];

int address;

} syymbol;

typedef struct {

char literal[10];

int address;

} littab;

typedef struct {

char mnemonic[10];

int opcode;

} mot;

typedef struct {

char reg[10];

int code;

} registertab;

mot MOT[] = {

{"MOVER", 1}, {"MOVEM", 2}, {"ADD", 3}, {"SUB", 4}, {"MULT", 5},

{"DIV", 6}, {"BC", 7}, {"COMP", 8}, {"PRINT", 9}, {"READ", 10} ,{"SHIFT",11},{"BC",12}

};

mot POT[] = {

{"START", 1}, {"END", 2}, {"EQU", 3}, {"ORIGIN", 4}, {"LTORG", 5}

};

mot DL[] = {

{"DS", 1}, {"DC", 2}

};

registertab registers[] = {

{"AREG", 1}, {"BREG", 2}, {"CREG", 3}, {"DREG", 4}

};

int pooltab[50];

int poolcount = 0;

syymbol symtab[50];

littab litt[50];

int symcount = 0, litcount = 0, locctr = 0;

int searsys(char \*symbol) {

int i;

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

if (strcmp(symtab[i].symbol, symbol) == 0) {

return i;

}

}

return -1;

}

int searl(char \*literal) {

int i;

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

if (strcmp(litt[i].literal, literal) == 0) {

return i;

}

}

return -1;

}

int gop(char \*mnemonic, mot table[], int size) {

int i;

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

if (strcmp(table[i].mnemonic, mnemonic) == 0) {

return table[i].opcode;

}

}

return -1;

}

int grp(char \*reg) {

int i;

for (i = 0; i < 4; i++) {

if (strcmp(registers[i].reg, reg) == 0) {

return registers[i].code;

}

}

return -1;

}

void processLine(char \*line, FILE \*intermediate) {

char label[10] = "", opcode[10] = "", operand1[10] = "", operand2[10] = "";

int fields = sscanf(line, "%s %s %s %s", label, opcode, operand1, operand2);

// printf("\nCHEKC %s %s %s %d",label,opcode,operand1,locctr);

if (poolcount == 0) {

pooltab[poolcount++] = 0;

}

if (fields >= 3 && strlen(label) > 0 && strcmp(opcode, "START") != 0 && strcmp(opcode, "END") != 0 && strcmp(label, "END") != 0 && strcmp(label, "START") != 0) {

int symIndex = searsys(label);

if (symIndex == -1) {

strcpy(symtab[symcount].symbol, label);

symtab[symcount].address = locctr;

symcount++;

} else {

symtab[symIndex].address = locctr;

}

}

else if (fields>=3 && strlen(label) > 0 && strcmp(opcode, "START") != 0 && strcmp(opcode, "END") != 0 && strcmp(label, "END") != 0 && strcmp(label, "START") != 0) {

int symIndex = searsys(label);

if (symIndex == -1) {

strcpy(symtab[symcount].symbol, label);

symtab[symcount].address = locctr;

symcount++;

} else {

symtab[symIndex].address = locctr;

}

}

if (fields==3 && operand1[0] == '='){

int opcodeVal = gop(label, MOT, 11);

// printf("\nTHREE %s %s %s %d",label,opcode,operand1,locctr);

if (opcodeVal != -1) {

int regCode = grp(opcode);

int litIndex = searl(operand1);

if (litIndex == -1 && operand1[0] == '=') {

strcpy(litt[litcount].literal, operand1);

litt[litcount].address = 0;

litcount++;

litIndex = litcount - 1;

}

fprintf(intermediate, "%d\t(IS, %02d)\t(%d)\t(L, %d)\n", locctr, opcodeVal, regCode, litIndex + 1);

locctr++;

} else {

opcodeVal = gop(opcode, POT, 5);

if (opcodeVal != -1) {

fprintf(intermediate, "%d\t(AD, %02d)\t(C, %s)\n", locctr, opcodeVal, operand1);

if (strcmp(opcode, "START") == 0 || strcmp(opcode, "ORIGIN") == 0) {

locctr = atoi(operand1);

//locctr ++;

}

} else {

opcodeVal = gop(opcode, DL, 2);

if (opcodeVal != -1) {

fprintf(intermediate, "%d\t(DL, %02d)\t(C, %s)\n", locctr, opcodeVal, operand1);

locctr += atoi(operand1);

}

}

}

}

else if (fields==3 && operand1[0] != '=' && (label[0]!='B' && label[1]!='C')){

int opcodeVal = gop(label, MOT, 11);

// printf("\nFIVEEE %s %s %s %d",label,opcode,operand1,locctr);

if (opcodeVal != -1) {

int regCode = grp(opcode);

// printf("MEEEEE\n");

fprintf(intermediate, "%d\t(IS, %02d)\t(%d)\t(S, %d)\n", locctr, opcodeVal, regCode, searsys(operand1)+1);

locctr++;

} else if (gop(opcode, POT, 5)==3){

//nothing

}else {

opcodeVal = gop(opcode, POT, 5);

if (opcodeVal != -1) {

fprintf(intermediate, "%d\t(AD, %02d)\t(C, %s)\n", locctr, opcodeVal, operand1);

if (strcmp(opcode, "START") == 0 || strcmp(opcode, "ORIGIN") == 0) {

locctr = atoi(operand1);

//locctr ++;

}

} else {

opcodeVal = gop(opcode, DL, 2);

if (opcodeVal != -1) {

int symIndex = searsys(label);

// printf("\nHEREEE %s\n",label);

if (symIndex == -1) {

// printf("\nHEREEE %s\n",label);

strcpy(symtab[symcount].symbol, label);

symtab[symcount].address = locctr;

symcount++;

} else {

symtab[symIndex].address = locctr;

}

fprintf(intermediate, "%d\t(DL, %02d)\t %s\n", locctr, opcodeVal, operand1);

// locctr += atoi(operand1);

locctr ++;

}

}

}

}

else if (fields >= 3) {

int opcodeVal = gop(opcode, MOT, 11);

//printf("\nFOUR %s %s %s %s %d",label,opcode,operand1,operand2,locctr);

if (opcodeVal != -1) {

int regCode = grp(operand1);

int litIndex = searl(operand2);

int symIndex = searsys(operand2);

if (litIndex == -1 && operand2[0] == '=') {

strcpy(litt[litcount].literal, operand2);

litt[litcount].address = 0;

litcount++;

litIndex = litcount - 1;

}else if (symIndex == -1) {

// printf("\nHEREEE %s\n",operand1);

strcpy(symtab[symcount].symbol, operand1);

symtab[symcount].address = locctr;

symcount++;

}

litIndex = searl(operand2);

symIndex = searsys(operand2);

if(litIndex!=-1){

fprintf(intermediate, "%d\t(IS, %02d)\t(%d)\t(L, %d)\n", locctr, opcodeVal, regCode, litIndex + 1);

}else{

fprintf(intermediate, "%d\t(IS, %02d)\t(%d)\t(S, %d)\n", locctr, opcodeVal, regCode, symIndex + 1);

}

locctr++;

} else {

opcodeVal = gop(opcode, POT, 5);

if (opcodeVal != -1) {

fprintf(intermediate, "%d\t(AD, %02d)\t(C, %s)\n", locctr, opcodeVal, operand1);

if (strcmp(opcode, "START") == 0 || strcmp(opcode, "ORIGIN") == 0) {

locctr = atoi(operand1);

//locctr ++;

}

} else if (strcmp(label, "BC") == 0) {

int conditionCode = atoi(opcode);

int symIndex = searsys(operand1);

if (symIndex == -1) {

// printf("\nHEREEE %s\n",operand1);

strcpy(symtab[symcount].symbol, operand1);

symtab[symcount].address = locctr;

symcount++;

} else {

symtab[symIndex].address = locctr;

}

fprintf(intermediate, "%d\t(IS, %02d)\t(S, %d)\n", locctr, gop(label, MOT, 11), symcount);

locctr++;

}else {

opcodeVal = gop(opcode, DL, 2);

if (opcodeVal != -1) {

int symIndex = searsys(label);

// printf("\nHEREEE %s\n",label);

if (symIndex == -1) {

// printf("\nHEREEE %s\n",label);

strcpy(symtab[symcount].symbol, label);

symtab[symcount].address = locctr;

symcount++;

} else {

symtab[symIndex].address = locctr;

}

fprintf(intermediate, "%d\t(DL, %02d)\t %s\n", locctr, opcodeVal, operand1);

// locctr += atoi(operand1);

locctr ++;

}

}

}

} else if (fields == 2) {

int opcodeVal = gop(label, POT, 5);

// printf("\nTWOLO %s %s %d",label,opcode,locctr);

if (strcmp(label, "PRINT") == 0) {

int regCode = grp(opcode);

if(regCode!=-1){

fprintf(intermediate, "%d\t(IS, 09)\t(S, %d)\n", locctr, regCode);

locctr++;

}else if (searsys(opcode)!=-1){

fprintf(intermediate, "%d\t(IS, 09)\t(S, %d)\n", locctr,searsys(opcode));

locctr++;

}else if (searl(opcode)!=-1){

fprintf(intermediate, "%d\t(IS, 09)\t(L, %d)\n", locctr, searl(opcode));

locctr++;

}

}

else if (opcodeVal != -1) {

if ( strcmp(label, "ORIGIN") == 0) {

locctr = atoi(opcode);

//locctr ++;

}

else if (strcmp(label, "START") == 0 ){

locctr = atoi(opcode);

fprintf(intermediate, "%d\t(AD, %02d)\t(C, %s)\n", locctr, opcodeVal, opcode);

}

} else if (gop(label, MOT, 11)==10){

//NEEWWWWWWWW

int symIndex = searsys(opcode);

// printf("\nHEREEE %s %d\n",opcode,symIndex);

if (symIndex == -1) {

// printf("\nHEREEE %s\n",opcode);

strcpy(symtab[symcount].symbol, opcode);

symtab[symcount].address = locctr;

symcount++;

} else {

symtab[symIndex].address = locctr;

}

fprintf(intermediate, "%d\t(IS, %02d)\t(S, %d)\n", locctr, gop(label, MOT, 11), symcount);

locctr++;

}else if(gop(label, MOT, 11)!=-1){

//NEW

if(opcode[0]!='='){

int reg=0;

// printf("AKALKALAAL: %c\n ",opcode[0]);

opcodeVal = gop(label, MOT, 11);

reg=grp(opcode);

if(reg==-1){

fprintf(intermediate, "%d\t(IS, %02d)\t(S, %d)\n", locctr, opcodeVal, opcode);

locctr++;

}else{

fprintf(intermediate, "%d\t(IS, %02d)\t(S, %d)\n", locctr, opcodeVal, reg);

locctr++;

}

}else{

int litIndex = searl(opcode);

opcodeVal = gop(label, MOT, 11);

if (litIndex == -1 && opcode[0] == '=') {

strcpy(litt[litcount].literal, opcode);

litt[litcount].address = 0;

litcount++;

litIndex = litcount - 1;

}

fprintf(intermediate, "%d\t(IS, %02d)\t(L, %d)\n", locctr, opcodeVal, litIndex + 1);

locctr++;

}

}else {

opcodeVal = gop(label, DL, 2);

if (opcodeVal != -1) {

// printf("\nLELE\n");

fprintf(intermediate, "%d\t(DL, %02d)\t(C, %s)\n", locctr, opcodeVal, opcode);

locctr += atoi(opcode);

//locctr ++;

} else {

opcodeVal = gop(label, MOT, 11);

if (opcodeVal != -1) {

int regCode = grp(opcode);

fprintf(intermediate, "%d\t(IS, %02d)\t(%d)\n", locctr, opcodeVal, regCode);

}

}

}

} else if (fields == 1) {

int opcodeVal = gop(label, POT, 5);

if (opcodeVal != -1) {

//fprintf(intermediate, "%d\t(AD, %02d)\n", locctr, opcodeVal);

int i;

if (strcmp(label, "LTORG") == 0||strcmp(label, "END") == 0) {

// printf("\nZZZZZZ %d\n",opcodeVal);

for (i = pooltab[poolcount - 1]; i < litcount; i++) {

if (litt[i].address == 0) {

litt[i].address = locctr;

fprintf(intermediate, "%d\t(AD, %02d)\t%d\n", locctr, opcodeVal, litt[i].address);

locctr++;

}

}

pooltab[poolcount++] = litcount;

}

}

}else if (strcmp(label, "BC") == 0) {

int conditionCode = atoi(operand1);

if (conditionCode >= 1 && conditionCode <= 12) {

fprintf(intermediate, "%d\t(IS, %02d)\t(C, %d)\n", locctr, gop(opcode, MOT, 11), conditionCode);

locctr++;

}

}

else {

fprintf(intermediate, "%d\t%s\n", locctr, label);

locctr++;

}

}

//ad,05 LITERAL

//DL,INDEX space

//dont print label equ loop lines

//dont print origin lines

void writeTablesToFile() {

FILE \*symbolFile = fopen("symbol.txt", "w");

FILE \*literalFile = fopen("literal.txt", "w");

int i;

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

if (gop(symtab[i].symbol, MOT, 11) == -1 && gop(symtab[i].symbol, DL, 2) == -1) {

fprintf(symbolFile, "%s %d\n", symtab[i].symbol, symtab[i].address);

}

}

i=0;

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

fprintf(literalFile, "%s %d\n", litt[i].literal, litt[i].address);

}

fclose(symbolFile);

fclose(literalFile);

printf("Tables written to files successfully.\n");

}

int main() {

FILE \*input = fopen("in7.txt", "r");

FILE \*intermediate = fopen("inter.txt", "w");

//// FILE \*machineCode = fopen("machineCodee.txt", "w");

int i;

char line[50];

while (fgets(line, sizeof(line), input)) {

processLine(line, intermediate);

}

fclose(input);

fclose(intermediate);

intermediate = fopen("inter.txt", "r");

//generateMachineCode(intermediate, machineCode);

fclose(intermediate);

// fclose(machineCode);

i=0;

printf("\nSymbol Table:\n");

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

if (gop(symtab[i].symbol, MOT, 11) == -1 && gop(symtab[i].symbol, DL, 2) == -1) {

printf("%s\t%d\n", symtab[i].symbol, symtab[i].address);

}

}

i=0;

printf("Literal Table:\n");

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

printf("%s\t%d\n", litt[i].literal, litt[i].address);

}

i=0;

printf("\nPool Table:\n");

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

printf("Pool %d starts at literal %d\n", i + 1, pooltab[i]);

}

writeTablesToFile();

return 0;

}

#note if diff input given don't use "," use space instead also declare litereals in between "=' '" ex ='2' and also no ':'

#input for pass 1:

START 100

INPSTR DS 50

COUNT DC 0

READ INPSTR

MOVER AREG ='0'

MOVEM AREG COUNT

LOOP MOVER BREG INPSTR

ADD BREG AREG

COMP BREG ='0'

LTORG

BC 8 ENDLOOP

ADD AREG ='1'

MOVEM BREG COUNT

ADD BREG ='1'

LTORG

BC 7 LOOP

ENDLOOP

PRINT COUNT

END

#pass 2

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

char symbol[10];

int address;

} syymbol;

typedef struct {

char literal[10];

int address;

} littab;

typedef struct {

char mnemonic[10];

int opcode;

} mot;

typedef struct {

char reg[10];

int code;

} registertab;

mot MOT[] = {

{"MOVER", 1}, {"MOVEM", 2}, {"ADD", 3}, {"SUB", 4}, {"MULT", 5},

{"DIV", 6}, {"BC", 7}, {"COMP", 8}, {"PRINT", 9}, {"READ", 10} ,{"SHIFT",11},{"BC",12}

};

mot POT[] = {

{"START", 1}, {"END", 2}, {"EQU", 3}, {"ORIGIN", 4}, {"LTORG", 5}

};

mot DL[] = {

{"DS", 1}, {"DC", 2}

};

registertab registers[] = {

{"AREG", 1}, {"BREG", 2}, {"CREG", 3}, {"DREG", 4}

};

syymbol symtab[50];

littab litt[50];

int symcount = 0, litcount = 0, locctr = 0;

int searsys(char \*symbol) {

int i;

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

if (strcmp(symtab[i].symbol, symbol) == 0) {

return i;

}

}

return -1;

}

int searl(char \*literal) {

int i;

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

if (strcmp(litt[i].literal, literal) == 0) {

return i;

}

}

return -1;

}

int gop(char \*mnemonic, mot table[], int size) {

int i;

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

if (strcmp(table[i].mnemonic, mnemonic) == 0) {

return table[i].opcode;

}

}

return -1;

}

int grp(char \*reg) {

int i;

for (i = 0; i < 4; i++) {

if (strcmp(registers[i].reg, reg) == 0) {

return registers[i].code;

}

}

return -1;

}

void readTablesFromFile() {

FILE \*symbolFile = fopen("symbol.txt", "r");

FILE \*literalFile = fopen("literal.txt", "r");

symcount = 0;

while (fscanf(symbolFile, "%s %d", symtab[symcount].symbol, &symtab[symcount].address) == 2) {

symcount++;

}

litcount = 0;

while (fscanf(literalFile, "%s %d", litt[litcount].literal, &litt[litcount].address) == 2) {

litcount++;

}

fclose(symbolFile);

fclose(literalFile);

printf("Tables read from files successfully.\n");

}

void generateMachineCode(FILE \*intermediate, FILE \*machine) {

char line[50];

int address, opcode, regCode, operand;

int litIndex, symIndex;

char l[50];

int k=0;

//char labeel[10] = "", opcodee[10] = "", operand11[10] = "", operand22[10] = "",op[10]="",po[10]="";

// int fields = sscanf(line, "%s %s %s %s %s %s", labeel, opcodee, operand11, operand22,op,po);

readTablesFromFile();

while (fgets(line, sizeof(line), intermediate)) {

//note: for registersss

if (sscanf(line, "%d\t(IS, %02d)\t(%d)\t(S, %d)", &address, &opcode, &regCode, &operand) == 4) {

symIndex = operand - 1;

if (symIndex >= 0 && symIndex < symcount) {

if(regCode){

fprintf(machine, "%d %02d %d %d\n", address, opcode, regCode, symtab[symIndex].address);

}else{

fprintf(machine, "%d %02d %d\n", address, opcode, symtab[symIndex].address);

}

} else {

fprintf(machine, "%d %02d %d %d\n", address, opcode, regCode, operand);

}

}

else if (sscanf(line, "%d\t(IS, %02d)\t(%d)\t(L, %d)", &address, &opcode, &regCode, &operand) == 4) {

litIndex = operand - 1;

if (litIndex >= 0 && litIndex < litcount) {

fprintf(machine, "%d %02d %d %d\n", address, opcode, regCode, litt[litIndex].address);

} else {

fprintf(machine, "%d %02d %d %d\n", address, opcode, regCode, operand);

}

}

//note: without

else if (sscanf(line, "%d\t(IS, %02d)\t(S, %d)", &address, &opcode, &operand) == 3) {

symIndex = operand - 1;

if (symIndex >= 0 && symIndex < symcount) {

fprintf(machine, "%d %02d %d\n", address, opcode, symtab[symIndex].address);

} else {

fprintf(machine, "%d %02d %d\n", address, opcode, operand);

}

}

else if (sscanf(line, "%d\t(IS, %02d)\t(L, %d)", &address, &opcode, &operand) == 3) {

litIndex = operand - 1;

if (litIndex >= 0 && litIndex < litcount) {

fprintf(machine, "%d %02d %d\n", address, opcode, litt[litIndex].address);

} else {

fprintf(machine, "%d %02d %d\n", address, opcode, operand);

}

}

else if (sscanf(line, "%d\t(AD, %02d)\t(C, %s)", &address, &opcode, line) == 3) {

fprintf(machine, "%d %02d %s\n", address, opcode, line);

}

else if (sscanf(line, "%d\t(DL, %02d)\t %s", &address, &opcode, line) == 3) {

fprintf(machine, "%d %02d %s\n", address, opcode, line);

}

else if (sscanf(line, "%d\t(AD, %02d)", &address, &opcode) == 2) {

// fprintf(machine, "%d %02d\n", address, opcode);

fprintf(machine, "%d %d %d\n",address,opcode,litt[k].address);

k++;

}

else if (sscanf(line, "%d\t(DL, %02d)", &address, &opcode) == 2) {

fprintf(machine, "%d %02d\n", address, opcode);

}

else if (sscanf(line, "%d\t%s", &address,&opcode, line) == 2) {

fprintf(machine, "%d %d %d\n",address,k,litt[k].address);

printf("\nliterals %d %d\n",k,litt[k].address);

k++;

}

}

}

//ad,05 LITERAL

//DL,INDEX space

//dont print label equ loop lines

//dont print origin lines

int main() {

FILE \*input = fopen("in7.txt", "r");

FILE \*machineCode = fopen("machine.txt", "w");

int i;

FILE \*intermediate = fopen("inter.txt", "r");

fclose(input);

generateMachineCode(intermediate, machineCode);

fclose(intermediate);

fclose(machineCode);

i=0;

printf("\nSymbol Table:\n");

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

if (gop(symtab[i].symbol, MOT, 11) == -1 && gop(symtab[i].symbol, DL, 2) == -1) {

printf("%s\t%d\n", symtab[i].symbol, symtab[i].address);

}

}

i=0;

printf("Literal Table:\n");

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

printf("%s\t%d\n", litt[i].literal, litt[i].address);

}

return 0;

}

#inputs for pass 2 are symbol,literal table and intermediate code, I will provide example input for previous pass 1 input

#symbol table:

INPSTR 102

COUNT 101

LOOP 114

ENDLOOP 109

#literal table:

='0' 108

='1' 113

#intermediate code:

100 (AD, 01) (C, 100)

100 (DL, 01) 50

101 (DL, 02) 0

102 (IS, 10) (S, 2)

103 (IS, 01) (1) (L, 1)

104 (IS, 02) (1) (S, 2)

105 (IS, 01) (2) (S, 1)

106 (IS, 03) (2) (S, 0)

107 (IS, 08) (2) (L, 1)

108 (AD, 05) 108

109 (IS, 07) (S, 9)

110 (IS, 03) (1) (L, 2)

111 (IS, 02) (2) (S, 2)

112 (IS, 03) (2) (L, 2)

113 (AD, 05) 113

114 (IS, 07) (S, 9)

115 (IS, 09) (S, 1)

#macro processor pass 1:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_MNT 50

#define MAX\_MDT 100

#define MAX\_ALA 10

typedef struct {

char name[10];

int mdtIndex;

} MNTEntry;

typedef struct {

char definition[50];

} MDTEntry;

MNTEntry MNT[MAX\_MNT];

MDTEntry MDT[MAX\_MDT];

int mntCount = 0, mdtCount = 0;

void storeMacroDefinition(char \*line) {

strcpy(MDT[mdtCount++].definition, line);

}

void processMacroDefinition(FILE \*input, FILE \*nonMacroFile) {

char line[100], macroName[10];

while (fgets(line, sizeof(line), input)) {

if (strstr(line, "MACRO")) {

fscanf(input, "%s", macroName);

strcpy(MNT[mntCount].name, macroName);

MNT[mntCount].mdtIndex = mdtCount;

mntCount++;

fgets(line, sizeof(line), input);

while (fgets(line, sizeof(line), input)) {

if (strstr(line, "MEND")) {

storeMacroDefinition(line);

break;

}

storeMacroDefinition(line);

}

} else {

fprintf(nonMacroFile, "%s", line);

}

}

}

void saveTables() {

FILE \*mntFile = fopen("MNT.txt", "w");

FILE \*mdtFile = fopen("MDT.txt", "w");

int i;

if (!mntFile || !mdtFile) {

printf("Error opening file for saving tables.\n");

return;

}

i=0;

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

fprintf(mntFile, "%s\t%d\n", MNT[i].name, MNT[i].mdtIndex);

}

i=0;

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

fprintf(mdtFile, "%s", MDT[i].definition);

}

fclose(mntFile);

fclose(mdtFile);

}

int main() {

FILE \*input = fopen("mput3.txt", "r");

FILE \*nonMacroFile = fopen("IC.txt", "w");

if (!input || !nonMacroFile) {

printf("Error opening file.\n");

return 1;

}

processMacroDefinition(input, nonMacroFile);

fclose(input);

fclose(nonMacroFile);

saveTables();

return 0;

}

#input for pass1:

MACRO

DTWO &ARG1, &ARG2

AIF &ARG3 EQ 4 LABEL1

MOVER AREG, &ARG1

ADD AREG, &ARG2

AGO ENDLABEL

LABEL1

MOVER AREG, 1

ENDLABEL

MEND

MACRO

MULTIPLY &ARG3

MOVER BREG, &ARG3

MUL BREG, 2

DTWO BREG, 5

MEND

START 100

MULTIPLY 4

END

#pass 2 code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

typedef struct {

char name[10];

int mdtIndex;

}mntet;

typedef struct {

char definition[50];

}mdtet;

mntet mnt[50];

mdtet mdt[100];

char ALA[10][10];

char bl[10][10];

int xc[10];

int mntCount = 0, mdtCount = 0;

int alaIndex = 0, c = 0, f = 0;

void trimWhitespace(char \*str) {

char \*end;

while (isspace((unsigned char)\*str)) str++;

end = str + strlen(str);

while (end > str && isspace((unsigned char)\*(end - 1))) end--;

\*end = '\0';

}

int ca(int i) {

int z;

for (z = 0; z < alaIndex; z++) {

if (strcmp(bl[z], ALA[z]) != 0) {

xc[z] = i;

return z;

} else if (xc[z] == i) {

return z;

}

}

}

void lt() {

FILE \*mntFile = fopen("MNT.txt", "r");

FILE \*mdtFile = fopen("MDT.txt", "r");

while (fscanf(mntFile, "%s %d", mnt[mntCount].name, &mnt[mntCount].mdtIndex) != EOF) {

mntCount++;

}

while (fgets(mdt[mdtCount].definition, sizeof(mdt[mdtCount].definition), mdtFile)) {

trimWhitespace(mdt[mdtCount].definition);

mdtCount++;

}

fclose(mntFile);

fclose(mdtFile);

}

int evaluateCondition(char \*condition) {

char arg[10], op[3], value[10];

int i;

sscanf(condition, "&%s %s %s", arg, op, value);

//printf("sfds%s %s %s\n",arg,op,value);

//printf("sfds%s\n",condition);

i=0;

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

if (strcmp(ALA[i], value) == 0 && strcmp(op, "EQ") == 0) {

return 1;

}

}

return 0;

}

void sag(char \*line, FILE \*output) {

char resultLine[100] = "";

int cnt = 0;

int foundSubstitution = 0;

int i;

// printf("sfl: '%s'\n", line);

i=0;

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

char formalParam[6];

char \*argPos;

sprintf(formalParam, "&ARG%d", i + 1);

argPos = strstr(line, formalParam);

if (argPos != NULL) {

if (f == 0) {

strcpy(bl[c], ALA[c]);

xc[c] = i + 1;

cnt = 0;

c++;

f = 1;

} else {

cnt = ca(i + 1);

strcpy(bl[c], ALA[c]);

c++;

}

// printf("subt '%s' with '%s' %d\n", formalParam, ALA[cnt], cnt);

strncat(resultLine, line, argPos - line);

strcat(resultLine, ALA[cnt]);

strcat(resultLine, argPos + strlen(formalParam));

strcpy(line, resultLine);

}

}

/\* printf("ALA:\n");

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

printf("ALA[%d]: '%s'\n", j, ALA[j]);

}\*/

if (!foundSubstitution) {

strcpy(resultLine, line);

}

//printf("Fls: '%s'\n", resultLine);

fprintf(output, "%s\n", resultLine);

}

int isMacro(char \*name) {

int i;

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

if (strcmp(name, mnt[i].name) == 0) {

return 1;

}

}

return 0;

}

int findMDT(char \*macroName) {

int i;

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

if (strcmp(mnt[i].name, macroName) == 0) {

return mnt[i].mdtIndex;

}

}

return -1;

}

int findLabelInMDT(char \*label) {

int i;

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

if (strncmp(mdt[i].definition, label, strlen(label)) == 0 &&

(mdt[i].definition[strlen(label)] == ' ' || mdt[i].definition[strlen(label)] == '\0')) {

return i;

}

}

return -1;

}

void expandMacro(char \*macroName, char \*line, FILE \*output) {

char actualArgs[100];

char \*arg;

int mdtptr;

int labelFlag = 0,cf=0;

strcpy(actualArgs, line + strlen(macroName));

trimWhitespace(actualArgs);

arg = strtok(actualArgs, ", ");

while (arg != NULL && alaIndex < 10) {

int isMacro;

trimWhitespace(arg);

isMacro = findMDT(arg);

if (isMacro == -1) {

strcpy(ALA[alaIndex++], arg);

} else {

printf("Skipping macro name: '%s' from being added to ALA.\n", arg);

}

arg = strtok(NULL, ", ");

}

mdtptr = findMDT(macroName);

labelFlag = 0,cf=0;

while (mdtptr != -1 && strcmp(mdt[mdtptr].definition, "MEND") != 0) {

char command[10],t[20],t1[20];

int fields;

sscanf(mdt[mdtptr].definition, "%s", command);

fields=sscanf(mdt[mdtptr].definition,"%s %s ",t,t1);

//printf("Processing mdt[%d]: '%s' %d\n", mdtptr, mdt[mdtptr].definition,fields);

//printf("Processing mdt[%d]: '%s' \n", mdtptr, mdt[mdtptr].definition);

if(fields<2 && labelFlag!=1 && cf!=1){

return;

}

else if (labelFlag == 1) {

labelFlag = 0;

}else if(cf ==1){

cf=0;

} else if (strcmp(command, "AIF") == 0) {

char condition[50], label[10],c1[50],c2[50];

sscanf(mdt[mdtptr].definition, "AIF %s %s %s %s", condition,c1,c2, label);

// printf("\ndasd%s %s %s %s %s",condition,label,c1,c2);

strcat(condition," ");

strcat(condition,c1);

strcat(condition," ");

strcat(condition,c2);

// printf("\ncondition:%s\n",condition);

if (evaluateCondition(condition)) {

printf("Condition '%s' is true, jumping to label '%s'.\n", condition, label);

mdtptr = findLabelInMDT(label);

if (mdtptr == -1) {

printf("Error: Label '%s' not found in mdt.\n", label);

break;

}

cf=1;

continue;

}

} else if (strcmp(command, "AGO") == 0) {

char label[10];

sscanf(mdt[mdtptr].definition, "AGO %s", label);

printf("Jumping to label '%s'.\n", label);

mdtptr = findLabelInMDT(label);

labelFlag = 1;

if (mdtptr == -1) {

printf("Error: Label '%s' not found in mdt.\n", label);

break;

}

continue;

} else if (strcmp(command, "ANOP") == 0) {

printf("ANOP encountered, skipping this line.\n");

mdtptr++;

continue;

} else {

char nmn[10];

sscanf(mdt[mdtptr].definition, "%s", nmn);

if (findMDT(nmn) != -1) {

printf("nested macro: '%s'\n", nmn);

expandMacro(nmn, mdt[mdtptr].definition, output);

} else {

sag(mdt[mdtptr].definition, output);

}

}

mdtptr++;

}

if (strcmp(mdt[mdtptr].definition, "MEND") == 0) {

printf("Finished macro: '%s'\n", macroName);

} else {

printf("mend missing !! macro '%s'.\n", macroName);

}

}

int main() {

FILE \*input = fopen("ic.txt", "r");

FILE \*output = fopen("output.txt", "w");

int j;

char line[100];

lt();

while (fgets(line, sizeof(line), input)) {

trimWhitespace(line);

if (strlen(line) > 0) {

char macroName[10];

sscanf(line, "%s", macroName);

if (isMacro(macroName)) {

expandMacro(macroName, line, output);

} else {

fprintf(output, "%s\n", line);

}

}

}

j=0;

printf("ALA:\n");

for (j = 0; j < 10; j++) {

printf("ALA[%d]: '%s'\n", j, ALA[j]);

}

fclose(input);

fclose(output);

return 0;

}

#input for pass 2 is mdt,mnt,intermedite code, I will provide example inputs based on pass 1 input

#mdt

AIF &ARG3 EQ 4 LABEL1

MOVER AREG, &ARG1

ADD AREG, &ARG2

AGO ENDLABEL

LABEL1

MOVER AREG, 1

ENDLABEL

MEND

MOVER BREG, &ARG3

MUL BREG, 2

DTWO BREG, 5

MEND

#mnt

DTWO 0

MULTIPLY 8

#intermediate code

START 100

MULTIPLY 4

END

#lexical code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_TOKEN\_LENGTH 100

typedef enum {

KEYWORD, IDENTIFIER, NUMBER, OPERATOR, DELIMITER, UNKNOWN

} TokenType;

const char \*keywords[] = {

"auto", "break", "case", "char", "const", "continue", "default", "do", "double", "else",

"enum", "extern", "float", "for", "goto", "if", "int", "long", "register", "return",

"short", "signed", "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned",

"void", "volatile", "while"

};

const char operators[] = "+-\*/=%<>&|^!";

const char delimiters[] = "(),;{}[]";

isKeyword(const char \*str) {

int i;

for (i = 0; i < sizeof(keywords) / sizeof(keywords[0]); i++) {

if (strcmp(str, keywords[i]) == 0)

return 1;

}

return 0;

}

isOperator(char ch) {

int i;

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

if (ch == operators[i])

return 1;

}

return 0;

}

isDelimiter(char ch) {

int i;

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

if (ch == delimiters[i])

return 1;

}

return 0;

}

void printToken(TokenType type, const char \*token) {

const char \*typeStr;

switch (type) {

case KEYWORD:

typeStr = "Keyword";

break;

case IDENTIFIER:

typeStr = "Identifier";

break;

case NUMBER:

typeStr = "Number";

break;

case OPERATOR:

typeStr = "Operator";

break;

case DELIMITER:

typeStr = "Delimiter";

break;

default:

typeStr = "Unknown";

break;

}

printf("Token: %-10s : Value: %s\n", typeStr, token);

}

void analyze(const char \*src) {

char token[MAX\_TOKEN\_LENGTH];

int i = 0, j = 0;

while (src[i] != '\0') {

if (isspace(src[i])) {

i++;

continue;

}

if (isalpha(src[i])) {

j = 0;

while (isalnum(src[i])) {

token[j++] = src[i++];

}

token[j] = '\0';

if (isKeyword(token))

printToken(KEYWORD, token);

else

printToken(IDENTIFIER, token);

}

else if (isdigit(src[i])) {

j = 0;

while (isdigit(src[i])) {

token[j++] = src[i++];

}

token[j] = '\0';

printToken(NUMBER, token);

}

else if (isOperator(src[i])) {

token[0] = src[i++];

token[1] = '\0';

printToken(OPERATOR, token);

}

else if (isDelimiter(src[i])) {

token[0] = src[i++];

token[1] = '\0';

printToken(DELIMITER, token);

}

else {

token[0] = src[i++];

token[1] = '\0';

printToken(UNKNOWN, token);

}

}

}

int main() {

FILE \*file = fopen("in.c", "r");

char buffer[1024];

char sourceCode[1024] = "";

if (!file) {

printf("Error opening file.\n");

return 1;

}

while (fgets(buffer, sizeof(buffer), file)) {

strcat(sourceCode, buffer);

}

fclose(file);

printf("Lexical Analysis:\n\n");

analyze(sourceCode);

return 0;

}

#input

int a=5;

int c=7;

print(c\*a);

//code to print only literal table

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct {

char literal[10];

int address;

} LiteralEntry;

typedef struct {

int pool\_start;

} PoolEntry;

LiteralEntry littab[100];

int littab\_index = 0;

PoolEntry pooltab[100];

int pooltab\_index = 0;

int locctr = 0;

void addLiteral(char \*literal) {

int i;

for (i = 0; i < littab\_index; i++) {

if (strcmp(littab[i].literal, literal) == 0) {

return;

}

}

strcpy(littab[littab\_index].literal, literal);

littab[littab\_index].address = -1;

littab\_index++;

}

void processLTORG() {

int i;

if (pooltab\_index == 0 || pooltab[pooltab\_index - 1].pool\_start < littab\_index) {

pooltab[pooltab\_index].pool\_start = littab\_index;

pooltab\_index++;

}

for (i = pooltab[pooltab\_index ].pool\_start; i < littab\_index; i++) {

if (littab[i].address == -1) {

littab[i].address = locctr++;

}

}

}

void processLineForLiterals(char \*label,char \*mnemonic, char \*operand) {

// printf("%s %s %s %d\n",label,mnemonic,operand,locctr);

if (operand[0] == '=') {

addLiteral(operand);

locctr++;

}

else{

locctr++;

}

if (strcmp(label, "LTORG") == 0) {

processLTORG();

}

}

void finalizeLiterals(int \*locctr) {

processLTORG(locctr);

}

void processFileForLiterals(const char \*filename) {

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

int i;

char line[100];

printf("Processing file for literal table: %s\n", filename);

while (fgets(line, sizeof(line), file)) {

char label[20] = "", mnemonic[20] = "", operand[20] = "";

sscanf(line, "%s %s %s", label, mnemonic, operand);

if (strcmp(label, "START") == 0 || strcmp(label, "ORIGIN") == 0) {

locctr = atoi(mnemonic);

} else {

processLineForLiterals(label,mnemonic, operand);

}

}

fclose(file);

finalizeLiterals(&locctr);

printf("\nLiteral Table:\n");

for (i = 0; i < littab\_index; i++) {

printf("Literal: %s, Address: %d\n", littab[i].literal, littab[i].address);

}

printf("\nPool Table:\n");

for (i = 0; i < pooltab\_index; i++) {

printf("Pool %d starts at literal %d\n", i + 1, pooltab[i].pool\_start);

}

}

int main() {

const char \*filename = "in7.txt";

processFileForLiterals(filename);

return 0;

}

//input for it:

START 100

INPSTR DS 50

COUNT DC 0

READ INPSTR

MOVER AREG ='0'

MOVEM AREG COUNT

LOOP

MOVER BREG INPSTR

ADD BREG AREG

COMP BREG ='0'

LTORG

BC 8 ENDLOOP

ADD AREG ='1'

MOVEM BREG COUNT

ADD BREG ='1'

LTORG

BC 7 LOOP

ENDLOOP

PRINT COUNT

END

//code for symbol table:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct {

char mnemonic[10];

int opcode;

} mot;

typedef struct {

char symbol[10];

int address;

int defined;

} symtab\_entry;

typedef struct {

char reg[10];

int code;

} registertab;

mot MOT[] = {

{"MOVER", 1}, {"MOVEM", 2}, {"ADD", 3}, {"SUB", 4}, {"MULT", 5},

{"DIV", 6}, {"BC", 7}, {"COMP", 8}, {"PRINT", 9}, {"READ", 10}, {"SHIFT", 11}, {"STOP", 12}

};

mot POT[] = {

{"START", 1}, {"END", 2}, {"EQU", 3}, {"ORIGIN", 4}, {"LTORG", 5}

};

mot DL[] = {

{"DS", 1}, {"DC", 2}

};

registertab registers[] = {

{"AREG", 1}, {"BREG", 2}, {"CREG", 3}, {"DREG", 4}

};

symtab\_entry symtab[100];

int symtab\_index = 0;

int gop(char \*mnemonic, mot table[], int size) {

int i;

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

if (strcmp(table[i].mnemonic, mnemonic) == 0) {

return table[i].opcode;

}

}

return -1;

}

int findSymbol(char \*symbol) {

int i;

for (i = 0; i < symtab\_index; i++) {

if (strcmp(symtab[i].symbol, symbol) == 0) {

return i;

}

}

if(gop(symbol, MOT, 11)!=-1){

return 1;

}else if(gop(symbol, POT, 4)!=-1){

return 1;

}else if(gop(symbol, DL, 2)!=-1){

return 1;

}else if((gop(symbol, registers, 4))!=-1){

return 1;

}

return -1;

}

void addSymbol(char \*symbol, int address, int defined) {

int index = findSymbol(symbol);

if (index == -1) {

strcpy(symtab[symtab\_index].symbol, symbol);

symtab[symtab\_index].address = address;

symtab[symtab\_index].defined = defined;

symtab\_index++;

}

}

void processLine(char \*label, char \*mnemonic, char \*operand, int \*locctr,int fields) {

int i;

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

int index = findSymbol(label);

if (index == -1) {

addSymbol(label, \*locctr, 1);

} else {

if (!symtab[index].defined) {

symtab[index].address = \*locctr;

symtab[index].defined = 1;

}

}

}

for (i = 0; i < sizeof(POT) / sizeof(POT[0]); i++) {

if (strcmp(POT[i].mnemonic, label) == 0) {

if (strcmp(label, "START") == 0) {

\*locctr = atoi(mnemonic);

} else if (strcmp(label, "END") == 0) {

return;

} else if (strcmp(label, "LTORG") == 0) {

\*locctr += 1;

}

return;

}

}

for (i = 0; i < sizeof(DL) / sizeof(DL[0]); i++) {

if (strcmp(DL[i].mnemonic, mnemonic) == 0) {

if (strcmp(mnemonic, "DS") == 0) {

\*locctr += 1;

} else if (strcmp(mnemonic, "DC") == 0) {

\*locctr += 1;

}

return;

}

}

for (i = 0; i < sizeof(MOT) / sizeof(MOT[0]); i++) {

if (strcmp(MOT[i].mnemonic, mnemonic) == 0) {

\*locctr += 1;

return;

}

}

}

void processFile(const char \*filename) {

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

char line[100];

int locctr = 0,i;

while (fgets(line, sizeof(line), file)) {

char label[20] = "", mnemonic[20] = "", operand[20] = "";

int fields= sscanf(line, "%s %s %s", label, mnemonic, operand);

processLine(label, mnemonic, operand, &locctr,fields);

}

fclose(file);

printf("\nSymbol Table:\n");

for (i = 0; i < symtab\_index; i++) {

printf("%s\t%d\n", symtab[i].symbol, symtab[i].address);

}

}

int main() {

const char \*filename = "in7.txt";

processFile(filename);

return 0;

}

//use same input as before