**/\*phase3.c\*/**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "phase3\_funcs.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ana Cecilia Sánchez Ramos

SYS PROG - CSCI 3334.01

PHASE 3.

Implement the second pass of a two-pass assembler:

Read from the intermediate file (pass 1), write to

listing and object files.

Missing: object code + error msgs

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(){

build\_optab();

pass1("source.asm");

symtab\_to\_file();

pass2();

return 0;

}

**/\*phase3\_funcs.h\*/**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "phase2\_funcs.h"

//\*find the opcode from the intermediate file\*/

char\* find\_opcode(char \*line){

static char opcode[6];

int tab\_ct = 0; //<---- amount of tabs (max = 2)

int i = 0; //<---- moving

int op\_ct = 0; //<---- opcode counter

while(tab\_ct != 2){ //<--- count 2 tabs (#tabs before opcode)

if(line[i] == '\t'){

tab\_ct++;

}

i++;

}

while(line[i] != '\t'){ //<--- put opcode in string

opcode[op\_ct] = line[i];

op\_ct++;

i++;

}

opcode[op\_ct] = '\0'; //<--- null-terminate string

return opcode;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*find the operand from the intermediate file -> used to create object code

char\* find\_operand(char \*line){

static char operand[10];

int tab\_ct = 0; //<---- amount of tabs (max = 3)

int i = 0; //<---- moving

int op\_ct = 0; //<---- operand counter

while(tab\_ct != 3){ //<--- count 3 tabs (#tabs before opcode)

if(line[i] == '\t'){

tab\_ct++;

}

i++;

}

while(line[i] != '\t'){ //<--- put opcode in string

operand[op\_ct] = line[i];

op\_ct++;

i++;

}

operand[op\_ct] = '\0'; //<--- null-terminate string

return operand;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*write line to listing file

void w\_listing\_file(FILE\* output\_ptr, char\* line, char\* opcode, char\* mnemonic, char\* address){

char obj\_code[20];

int i = 0;

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

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

}

else if(strcmp(opcode,"WORD") != 0 && strcmp(opcode,"RESW") != 0 && strcmp(opcode,"RESB") != 0){

for(i; i < strlen(mnemonic); i++){

obj\_code[i] = mnemonic[i];

}

obj\_code[i] = '\0';

if(strcmp(opcode,"RSUB") == 0){

strcat(obj\_code,"0000");

}

else{

strcat(obj\_code,address);

}

fprintf(output\_ptr,"%s\t%s\n",obj\_code, line);

}

else{

fprintf(output\_ptr,"\t%s\n", line); //testing purposes

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*get object code for BYTE

void getmnem\_BYTE(char\* operand, char\* mnemonic){

int m\_ct= 0;

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

char con\_mnem[5];

for(int i = 0 ; i < strlen(operand); i++){ //get rid of apostrophes + C

if(operand[i] != '\'' && operand[i] != 'C'){

con\_mnem[m\_ct] = operand[i];

m\_ct++;

}

}

for(int i = 0; i<strlen(con\_mnem); i++){ //get hex vals of constants

sprintf(mnemonic+i\*2, "%02X", con\_mnem[i]);

}

}

else{

for(int i = 0 ; i < strlen(operand); i ++){ //if X

if(operand[i] != 'X' && operand[i] != '\''){

mnemonic[m\_ct] = operand[i];

m\_ct++;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void pass2(){

FILE \* file\_ptr;

file\_ptr = fopen("intermediate\_file.txt","r"); //<--- read from intermediate file

FILE \* output\_ptr;

output\_ptr = fopen("listing\_file.txt","w");

char line[80];

char opcode[7], mnemonic[7];

char operand[10], address[7];

char obj\_code[20];

if(file\_ptr == NULL){

printf("can't reach file\n");

}

else{

fgets(line,80,file\_ptr); //<--- get headers to ignore them

while(!feof(file\_ptr)){

memset(&opcode[0],0,sizeof(opcode));

memset(&opcode[0],0,sizeof(mnemonic));

memset(&operand[0],0,sizeof(operand));

memset(&address[0],0,sizeof(address));

memset(&obj\_code[0],0,sizeof(obj\_code));

fgets(line,80,file\_ptr);

strcpy(opcode,find\_opcode(line)); //<--- get opcode

if(strcmp(opcode,"RSUB")==0){

strcpy(mnemonic,mnem\_value(opcode)); //<--- get mnemonic

}

else if(strcmp(opcode,"START") != 0 && strcmp(opcode,"END") != 0 && strcmp(opcode,"RSUB") != 0 &&

strcmp(opcode, "BYTE") != 0 && strcmp(opcode,"WORD") != 0 && strcmp(opcode,"RESW") != 0 && strcmp(opcode,"RESB") != 0){

strcpy(operand,find\_operand(line)); //<---get operand

if(strcmp(operand,"BUFFER,X") == 0){ //exception

strcpy(operand, "BUFFER");

}

strcpy(address,address\_val(operand));

strcpy(mnemonic,mnem\_value(opcode)); //<--- get mnemonic

}

else{

operand[0] = '\0';

/\*check if BYTE or WORD\*/

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

strcpy(operand,find\_operand(line)); //<---get operand & call function to get object code

getmnem\_BYTE(operand,mnemonic);

}

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

/\*call function to get objcode\*/

}

}

w\_listing\_file(output\_ptr,line, opcode, mnemonic, address);

}

}

fclose(output\_ptr);

fclose(file\_ptr);

}

**/\*phase2\_funcs.h\*/**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "hashtable.h"

void build\_optab(){

hasht = malloc(max \* sizeof(linkedList));

init\_hasht();

char\* op[][2] = {{"ADD", "18"}, {"RSUB", "4C"},

{"AND", "58"}, {"STA", "0C"},

{"COMP", "28"}, {"STCH", "54"},

{"DIV", "24"}, {"STL", "14"},

{"J", "3C"}, {"STX", "10"},

{"JEQ", "30"}, {"SUB", "1C"},

{"JGT", "34"}, {"TD", "E0"},

{"JLT", "38"}, {"TIX", "2C"},

{"JSUB", "48"}, {"WD", "DC"},

{"LDA", "00"}, {"LDCH", "50"},

{"LDL", "08"}, {"LDX", "04"},

{"MUL", "20"}, {"OR", "44"},

{"RD", "D8"}};

for(int i=0; i<25; i++){

insert(op[i][0],op[i][1]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void split\_into(char \*str, char\* label, char\* opcode, char\* operand) {

memset(&label[0],0,sizeof(label));

memset(&opcode[0],0,sizeof(opcode));

memset(&operand[0],0,sizeof(operand));

if(str[0] == '.'){ //<---comment

// ignore

}

else if(str[0] == ' ' || str[0] == '\t' || str[0] == '\n' || str[0] == '\0'){ //no label

sscanf(str, "%s %s", opcode, operand);

memset(&label[0],0,sizeof(label));

if(strcmp(opcode,"RSUB") == 0){

operand[0] = '\0';

}

}

else{

sscanf( str, "%s %s %s", label,opcode,operand);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int byte\_length(char\* operand, char\* error){

int b\_len = 0;

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

b\_len = strlen(operand) - 3;

}

else{ //<---if it's an X

b\_len = strlen(operand) - 3;

if(b\_len % 2 != 0){

strcat(error,"invalid operand! ");

} else{

b\_len = b\_len / 2;

}

}

return b\_len;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void handle\_line(int\* loc\_hexval, char\* locctr, char\* label, char\* opcode, char\* operand, char\* error){

//❤--------- check label !!! ---------❤

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

else if(lookup\_symbol(label) == 0){ //<--- symbol already exists

strcat(error,"duplicated label! ");

}else{

symtab\_insert(label, locctr);

}

//❤--------- check opcode + add to locctr !!! ---------❤

if(lookup\_opcode(opcode)==0 || strcmp(opcode,"WORD") == 0){ //add +3 to locctr

\*loc\_hexval = (int)strtol(locctr, NULL, 16);

\*loc\_hexval = (\*loc\_hexval) + 3;

// itoa(\*loc\_hexval,locctr,16);

sprintf(locctr, "%x", \*loc\_hexval); //converting loc decimal to hex

}

else if(strcmp(opcode,"RESB") == 0){ //add + #OPERAND to locctr

\*loc\_hexval = (int)strtol(locctr, NULL, 16);

int operand\_hexval = (int)strtol(operand, NULL, 10);

\*loc\_hexval = (\*loc\_hexval) + operand\_hexval;

// itoa(\*loc\_hexval,locctr,16);

sprintf(locctr, "%x", \*loc\_hexval); //converting loc decimal to hex

}

else if(strcmp(opcode,"RESW") == 0){ //add + 3 \* #OPERAND to locctr

\*loc\_hexval = (int)strtol(locctr, NULL, 16);

int operand\_hexval = (int)strtol(operand, NULL, 16);

\*loc\_hexval = (\*loc\_hexval) + (3 \* operand\_hexval);

// itoa(\*loc\_hexval,locctr,16);

sprintf(locctr, "%x", \*loc\_hexval); //converting loc decimal to hex

}

else if(strcmp(opcode, "BYTE") == 0){ //find length of constant in bytes & add length to locctr

if(operand[0] == 'C' || operand[0] == 'X'){

if(operand[1] == '\'' && operand[strlen(operand)-1] == '\''){

\*loc\_hexval = (int)strtol(locctr, NULL, 16);

\*loc\_hexval = (\*loc\_hexval) + (byte\_length(operand, error));

// itoa(\*loc\_hexval,locctr,16);

sprintf(locctr, "%x", \*loc\_hexval); //converting loc decimal to hex

}else{

strcat(error, "invalid operand! ");

}

}

else{

strcat(error, "invalid operand! ");

}

}

else{

strcat(error,"invalid opcode! ");

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void program\_length(int\* start, char\* start\_char, int\* end, char\* end\_char, int\* loc\_hexval){

\*start = (int)strtol(start\_char, NULL, 16);

\*end = (int)strtol(end\_char, NULL, 16);

\*end = (\*loc\_hexval)-(\*start);

//itoa(\*end, end\_char, 16);

sprintf(end\_char, "%x", \*end); //converting loc decimal to hex

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*----- P A S S 1 -----\*

void pass1(char \* src\_file){

//❤--------- VARIABLES ---------❤

char locctr[7], st\_address[7], end\_address[7];

locctr[0] = '\0', st\_address[0] = '\0', end\_address[0] = '\0';

int loc\_hexval = 0 ,int\_st = 0, int\_end = 0;

char line[80], label[7], opcode[7], operand[10], error[90];

error[0] = '\0';

FILE \* file\_ptr; //<---- file we read from

FILE \* output\_ptr; //<-----output file

//❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤

symbol\_table = malloc(symtab\_max \* sizeof(symtab\_linkedList));

build\_symtab();

if((file\_ptr = fopen(src\_file,"r")) == NULL){ //source file can't be opened

printf("can't reach file\n");

}

else{

if((output\_ptr = fopen("intermediate\_file.txt","w")) == NULL){

printf("can't open intermediate file\n"); //intermediate file can't be opened

}

else{

fprintf(output\_ptr,"%s\t%s\t%s\t%s\t%s\n", "LOCCTR","LABEL","OPCODE","OPERAND","ERRORS"); //intermediate file headers

fgets(line,80,file\_ptr); //store first string in line

split\_into(line, label, opcode, operand);

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

strcpy(st\_address, operand); //set starting address + locctr

strcpy(locctr,operand);

loc\_hexval = (int)strtol(locctr, NULL, 16); //<---- hex value to decimal integer

int\_st = (int)strtol(st\_address,NULL,16);

fprintf(output\_ptr,"%x\t%s\t%s\t%s\t%s\n", loc\_hexval, label, opcode, operand, error);

fgets(line,80,file\_ptr); //read next line

}

while(!feof(file\_ptr)){

memset(&error[0],0,sizeof(error));

split\_into(line, label, opcode, operand);

if(label[0]=='\0' && opcode[0]=='\0' && operand[0]=='\0'){}

else if(line[0] != '.'){ //<----- not a comment

fprintf(output\_ptr,"%x\t%s\t%s\t%s\t%s\n", loc\_hexval, label, opcode, operand, error); //<---write to intermediate file

if(strcmp(opcode,"END") != 0){

handle\_line(&loc\_hexval, locctr, label, opcode, operand, error);

}

else{ //<--- +3 to locctr when END is reached

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

strcat(error,"END does not need a label! ");

}

loc\_hexval = (int)strtol(locctr, NULL, 16);

loc\_hexval = (loc\_hexval) + 3;

// itoa(loc\_hexval,locctr,16);

sprintf(locctr, "%x", loc\_hexval); //converting loc decimal to hex\*/

}

}

fgets(line,80,file\_ptr);

}

}

fclose(output\_ptr);

}

program\_length(&int\_st, st\_address, &int\_end, end\_address, &loc\_hexval); //<--- calculate length of program

printf("File has been successfully assembled! \n");

fclose(file\_ptr);

}

**/\*hashtable.h\*/**

#include<stdio.h>

#include<stdlib.h>

#include <string.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\* O P C O D E T A B L E \*\*\*\*\*\*\*\*\*\*\*\*\*//

typedef struct node {

char opcode[7];

char mnemonic[7];

struct node \*next;

}node;

typedef struct linkedList { //stores a list at each index

struct node \*head;

struct node \*tail;

}linkedList;

linkedList \*hasht = NULL;

int max = 26;

int hashFunction(char\* opcode){ //returns index where we'll store item

unsigned int wordVal = 0;

for(int i = 0; i < strlen(opcode); i++){

wordVal = 37 \* wordVal + (opcode[i] - 48); //convert opcode -> number(key)

}

return wordVal % max;

}

//prototypes

void init\_hasht();

void insert(char\* opc, char\* mnem);

int lookup\_opcode(char\* str);

//definitions

void insert(char\* opc, char\* mnem) {

int index = hashFunction(opc);

node \*list = hasht[index].head;

node \*item = malloc(sizeof(struct node)); //item to be added

strcpy(item->opcode,opc);

strcpy(item->mnemonic,mnem);

item->next = NULL;

if (list == NULL) //linked list is empty

{

hasht[index].head = item;

hasht[index].tail = item;

}

else { //add to end of list

hasht[index].tail->next = item;

hasht[index].tail = item;

}

}

int lookup\_opcode(char\* str){

int index = hashFunction(str);

node \*temp = hasht[index].head;

while(temp!=NULL){

if(strcmp(str,temp->opcode) == 0){

return 0;

}

temp = temp->next;

}

return 1;

}

char\* mnem\_value(char\* str){

int index = hashFunction(str);

node \*temp = hasht[index].head;

while(temp!=NULL){

if(strcmp(str,temp->opcode) == 0){

return temp->mnemonic;

}

temp = temp->next;

}

return "nop";

}

void init\_hasht(){ //initialize hash symbol\_table

int i = 0;

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

{

hasht[i].head = NULL;

hasht[i].tail = NULL;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//\*\*\*\*\*\*\*\*\*\*\*\*\*\* S Y M B O L T A B L E \*\*\*\*\*\*\*\*\*\*\*\*\*//

typedef struct symtab\_node {

char label[10];

char address[7];

struct symtab\_node \*next;

}symtab\_node;

typedef struct symtab\_linkedList { //stores a list at each index

struct symtab\_node \*head;

struct symtab\_node \*tail;

}symtab\_linkedList;

symtab\_linkedList \*symbol\_table = NULL;

int symtab\_max = 501;

int symtab\_hashFunction(char\* label){ //returns index where we'll store item

unsigned int wordVal = 0;

for(int i = 0; i < strlen(label); i++){

wordVal = 37 \* wordVal + (label[i] - 48); //convert opcode -> number(key)

}

return wordVal % symtab\_max;

}

//prototypes

void build\_symtab();

void symtab\_insert(char\* lbl, char\* addr);

void symtab\_display();

//definitions

void symtab\_insert(char\* lbl, char\* addr) {

int index = symtab\_hashFunction(lbl);

symtab\_node \*list = symbol\_table[index].head;

symtab\_node \*item = malloc(sizeof(struct symtab\_node)); //item to be added

strcpy(item->label,lbl);

strcpy(item->address,addr);

item->next = NULL;

if (list == NULL) { //linked list is empty

symbol\_table[index].head = item;

symbol\_table[index].tail = item;

}

else { //add to end of list

symbol\_table[index].tail->next = item;

symbol\_table[index].tail = item;

}

}

void build\_symtab(){ //initialize hash symbol\_table

int i = 0;

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

symbol\_table[i].head = NULL;

symbol\_table[i].tail = NULL;

}

}

int lookup\_symbol(char\* str){

int index = symtab\_hashFunction(str);

symtab\_node \*temp = symbol\_table[index].head;

while(temp!=NULL){

if(strcmp(str,temp->label) == 0){

return 0;

}

temp = temp->next;

}

return 1;

}

char\* address\_val(char\* str){

int index = symtab\_hashFunction(str);

symtab\_node \*temp = symbol\_table[index].head;

while(temp!=NULL){

if(strcmp(str,temp->label) == 0){

return temp->address;

}

temp = temp->next;

}

return "nop";

}

void symtab\_to\_file(){

FILE \* file\_ptr;

if((file\_ptr = fopen("symtab\_file.txt","w")) == NULL){

printf("can't open intermediate file\n"); //file can't be opened

}else{

fprintf(file\_ptr,"%s | %s \n\n","LABEL", "ADDRESS"); //<---write to file

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

symtab\_node \*temp = symbol\_table[i].head;

if (temp != NULL) {

while (temp != NULL) {

fprintf(file\_ptr,"%s \t %s \n",temp->label, temp->address); //<---write to file

temp = temp->next;

}

}

}

}

fclose(file\_ptr);

}

**SOURCE FILE + INTERMEDIATE FILE + SYMBOL TABLE**

Source

COPY START 1000 COPY FILE FROM INPUT TO OUTPUT

FIRST STL RETADR SAVE RETURN ADDRESS

CLOOP JSUB RDREC READ INPUT RECORD

LDA LENGTH TEST FOR EOF (LENGTH = 0)

COMP ZERO

JEQ ENDFIL EXIT IF EOF FOUND

JSUB WRREC WRITE OUTPUT RECORD

J CLOOP LOOP

ENDFIL LDA EOF INSERT END OF FILE MARKER

STA BUFFER

LDA THREE SET LENGTH = 3

STA LENGTH

JSUB WRREC WRITE EOF

LDL RETADR GET RETURN ADDRESS

RSUB RETURN TO CALLER

EOF BYTE C'EOF'

THREE WORD 3

ZERO WORD 0

RETADR RESW 1

LENGTH RESW 1 LENGTH OF RECORD

BUFFER RESB 4096 4096-BYTE BUFFER AREA

.

. SUBROUTINE TO READ RECORD INTO BUFFER

.

RDREC LDX ZERO CLEAR LOOP COUNTER

LDA ZERO CLEAR A TO ZERO

RLOOP TD INPUT TEST INPUT DEVICE

JEQ RLOOP LOOP UNTIL READY

RD INPUT READ CHARACTER INTO REGISTAR A

COMP ZERO TEST FOR END OF RECORD (X'00')

JEQ EXIT EXIT LOOP IF EOF

STCH BUFFER,X STORE CHARCTER IN BUFFER

TIX MAXLEN LOOP UNLESS MAX LENGTH

JLT RLOOP HAS BEEN REACHED

EXIT STX LENGTH SAVE RECORD LENGTH

RSUB RETURN TO CALLER

INPUT BYTE X'F1' CODE FOR INPUT DEVICE

MAXLEN WORD 4096

.

. SUBROUTINE TO WRITE RECORD FROM BUFFER

.

WRREC LDX ZERO CLEAR LOOP COUNTER

WLOOP TD OUTPUT TEST OUTPUT DEVICE

JEQ WLOOP LOOP UNTIL READY

LDCH BUFFER,X GET CHARCTER FROM BUFFER

WD OUTPUT WRITE CHARACTER

TIX LENGTH LOOP UNTIL ALL CHARACTERS

JLT WLOOP HAVE BEEN WRITTEN

RSUB RETURN TO CALLER

OUTPUT BYTE X'05' CODE FOR OUTPUT DEVICE

END FIRST

Intermediate

LOCCTR LABEL OPCODE OPERAND ERRORS

1000 COPY START 1000

1000 FIRST STL RETADR

1003 CLOOP JSUB RDREC

1006 LDA LENGTH

1009 COMP ZERO

100c JEQ ENDFIL

100f JSUB WRREC

1012 J CLOOP

1015 ENDFIL LDA EOF

1018 STA BUFFER

101b LDA THREE

101e STA LENGTH

1021 JSUB WRREC

1024 LDL RETADR

1027 RSUB

102a EOF BYTE C'EOF'

102d THREE WORD 3

1030 ZERO WORD 0

1033 RETADR RESW 1

1036 LENGTH RESW 1

1039 BUFFER RESB 4096

2039 RDREC LDX ZERO

203c LDA ZERO

203f RLOOP TD INPUT

2042 JEQ RLOOP

2045 RD INPUT

2048 COMP ZERO

204b JEQ EXIT

204e STCH BUFFER,X

2051 TIX MAXLEN

2054 JLT RLOOP

2057 EXIT STX LENGTH

205a RSUB

205d INPUT BYTE X'F1'

205e MAXLEN WORD 4096

2061 WRREC LDX ZERO

2064 WLOOP TD OUTPUT

2067 JEQ WLOOP

206a LDCH BUFFER,X

206d WD OUTPUT

2070 TIX LENGTH

2073 JLT WLOOP

2076 RSUB

2079 OUTPUT BYTE X'05'

207a END FIRST

Symbol table

LABEL | ADDRESS

RDREC 2039

MAXLEN 205e

ENDFIL 1015

THREE 102d

ZERO 1030

INPUT 205d

EXIT 2057

RETADR 1033

RLOOP 203f

FIRST 1000

EOF 102a

WLOOP 2064

BUFFER 1039

WRREC 2061

LENGTH 1036

OUTPUT 2079

CLOOP 1003

**LISTING FILE**

1000 COPY START 1000

141033 1000 FIRST STL RETADR

482039 1003 CLOOP JSUB RDREC

001036 1006 LDA LENGTH

281030 1009 COMP ZERO

301015 100c JEQ ENDFIL

482061 100f JSUB WRREC

3C1003 1012 J CLOOP

00102a 1015 ENDFIL LDA EOF

0C1039 1018 STA BUFFER

00102d 101b LDA THREE

0C1036 101e STA LENGTH

482061 1021 JSUB WRREC

081033 1024 LDL RETADR

4C0000 1027 RSUB

454F46 102a EOF BYTE C'EOF'

102d THREE WORD 3

1030 ZERO WORD 0

1033 RETADR RESW 1

1036 LENGTH RESW 1

1039 BUFFER RESB 4096

041030 2039 RDREC LDX ZERO

001030 203c LDA ZERO

E0205d 203f RLOOP TD INPUT

30203f 2042 JEQ RLOOP

D8205d 2045 RD INPUT

281030 2048 COMP ZERO

302057 204b JEQ EXIT

541039 204e STCH BUFFER,X

2C205e 2051 TIX MAXLEN

38203f 2054 JLT RLOOP

101036 2057 EXIT STX LENGTH

4C0000 205a RSUB

F1 205d INPUT BYTE X'F1'

205e MAXLEN WORD 4096

041030 2061 WRREC LDX ZERO

E02079 2064 WLOOP TD OUTPUT

302064 2067 JEQ WLOOP

501039 206a LDCH BUFFER,X

DC2079 206d WD OUTPUT

2C1036 2070 TIX LENGTH

382064 2073 JLT WLOOP

4C0000 2076 RSUB

05 2079 OUTPUT BYTE X'05'

207a END FIRST

207a END FIRST