**SOURCE CODE**

/\*main.c\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "command\_funcs.c"

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

//Cecy SÃ¡nchez

//CSCI3334.01

//Phase #2

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

int main(){

char ans[100]; // string that stores the whole line

char comm[10], param1[10], param2[20]; //command + parameters

int ans\_len, n; //string length + number of parameters

int tru = 1;

printf("Welcome!\n");

printf("Here's the available commands:\n\n");

menu();

while(1){

printf("\nPlease enter a command + parameters: ");

fgets(ans, 100, stdin); /\* read in a line \*/

ans\_len = strlen(ans) - 1; /\* get rid of trailing newline character \*/

if (ans[ans\_len] == '\n'){

ans[ans\_len] = '\0';

}

breakLine(ans, comm, param1, param2, &n);

tru = strcmp(comm, "exit");

if(tru == 0 && param1[0] == '\0' && param2[0] == '\0'){

exit(0);

}

else if (tru == 0 && param1[0] != '\0'){

printf("No need for parameters! check \"help\" for more info\n");

}

else{

check(comm,param1,param2);

}

}

return 0;

}

/\*command\_funcs.h\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "phase2\_funcs.h"

//\*FUNCTION-> display available commands\*/

void menu(){

printf("load <filename> \nexecute\ndebug\ndump <start> <end>\nhelp\nassemble <filename>\ndirectory\nexit\n");

}

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

//\*FUNCTION-> break up string input\*/

void breakLine(char \*str, char \*comm, char \*param1, char \*param2, int \*n\_words) {

int str\_ct = 0, comm\_ct = 0, p1\_ct = 0, p2\_ct = 0;

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

memset(&param1[0],0,sizeof(param1));

memset(&param2[0],0,sizeof(param2));

\*n\_words = 0;

while(str[str\_ct] == ' '){ //check for spaces before command

str\_ct++;

}

while(str[str\_ct] != '\0' && str[str\_ct] != ' ') { //find command

comm[comm\_ct] = str[str\_ct];

str\_ct++;

comm\_ct++;

if(str[str\_ct] == '\0' || str[str\_ct] == ' '){ //null-terminate it

comm[comm\_ct] = '\0';

comm\_ct++;

(\*n\_words)++;

}

}

while(str[str\_ct] == ' '){

str\_ct++;

}

// find 1st parameter + store it

if(str[str\_ct] != '\0' && str[str\_ct] !=' '){

while(str[str\_ct] != '\0' && str[str\_ct]!=' '){

param1[p1\_ct] = str[str\_ct];

p1\_ct++;

str\_ct++;

if(str[str\_ct] == '\0' || str[str\_ct] == ' '){

param1[p1\_ct] = '\0';

p1\_ct++;

(\*n\_words)++;

}

}

}

else { param1[0]='\0'; }

while(str[str\_ct] == ' '){

str\_ct++;

}

//find 2nd parameter + store it

if(str[str\_ct] != '\0' && str[str\_ct] !=' ') {

while(str[str\_ct] != '\0' && str[str\_ct] != ' '){

param2[p2\_ct] = str[str\_ct];

p2\_ct++;

str\_ct++;

if(str[str\_ct] == '\0' || str[str\_ct] == ' '){

param2[p2\_ct] = '\0';

p2\_ct++;

(\*n\_words)++;

}

}

}

else{ param2[0]='\0'; }

}

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

//\*FUNCTION-> assemble function\*/

void assemble\_f(char \* src\_file){

build\_optab(); //build opcode table

pass1(src\_file);

symtab\_to\_file();

}

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

//\*FUNCTION-> check user input\*/

void check(char \*comm, char \*param1, char \*param2){

char \* commands[8]; //create array to store commands

char\* input;

int cmp;

int index;

//❤--------- commands ---------❤

commands[0] = "load";

commands[1] = "execute";

commands[2] = "debug";

commands[3] = "dump";

commands[4] = "help";

commands[5] = "assemble";

commands[6] = "directory";

//❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤❤

for(int i = 0 ; i < 8; i++){ /\*loop through array to find command\*/

cmp=strcmp(comm,commands[i]);

if(cmp == 0) {

input = comm;

index = i;

break;

}

}

/\*check each case\*/

if(index == 0){ //load

if(param1[0] != '\0' && param2[0] == '\0'){

printf("command has been reached!\n");

}

else if(param1[0] == '\0'){

printf("You're missing parameters, check \"help\" for more info\n");

}

else{

printf("Too many parameters! check \"help\" for more info\n");

}

}

else if(index == 1 || index == 2){ //execute or debug

if(param1[0] == '\0' && param2[0] == '\0'){

printf("command has been reached!\n");

}

else{

printf("No need for parameters! check \"help\" for more info\n");

}

}

else if(index == 3){ //dump

if(param1[0] != '\0' && param2[0] != '\0'){

printf("command has been reached!\n");

}

else if(param1[0] == '\0' || (param1[0] !='\0' && param2[0] =='\0')){

printf("You're missing parameters, check \"help\" for more info\n");

}

else{

printf("Too many parameters! check \"help\" for more info\n");

}

}

else if(index == 4){ //help

if(param1[0] == '\0' && param2[0] == '\0'){

menu();

}

else{

printf("No need for parameters! check \"help\" for more info\n");

}

}

else if(index == 5){ //assemble

if(param1[0] != '\0' && param2[0] == '\0'){

//printf("command has been reached!\n");

assemble\_f(param1);

}

else if(param1[0] == '\0'){

printf("You're missing parameters, check \"help\" for more info\n");

}

else{

printf("Too many parameters! check \"help\" for more info\n");

}

}

else if(index == 6) { //directory

if(param1[0]=='\0' && param2[0] == '\0'){

system("ls");

}

else{

printf("No need for parameters! check \"help\" for more info\n");

}

}

else{

printf("Sorry, not a valid command!!! check \"help\" for more info\n");

}

}

/\*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

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

}

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

}

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);

}

**ASM SOURCE FILES + SYMBOL TABLES + INTERMEDIATE FILES**

**/\*no errors\*/**

/\*source file\*/

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

/\*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

/\*intermediate file\*/

LOCCTR LABEL OPCODEOPERAND ERRORS

1000 COPY START 1000

1003 FIRST STL RETADR

1006 CLOOP JSUB RDREC

1009 LDA LENGTH

100c COMP ZERO

100f JEQ ENDFIL

1012 JSUB WRREC

1015 J CLOOP

1018 ENDFIL LDA EOF

101b STA BUFFER

101e LDA THREE

1021 STA LENGTH

1024 JSUB WRREC

1027 LDL RETADR

102a RSUB

102d EOF BYTE C'EOF'

1030 THREE WORD 3

1033 ZERO WORD 0

1036 RETADRRESW 1

1039 LENGTHRESW 1

2039 BUFFER RESB 4096

203c RDREC LDX ZERO

203f LDA ZERO

2042 RLOOP TD INPUT

2045 JEQ RLOOP

2048 RD INPUT

204b COMP ZERO

204e JEQ EXIT

2051 STCH BUFFER,X

2054 TIX MAXLEN

2057 JLT RLOOP

205a EXIT STX LENGTH

205d RSUB

205e INPUT BYTE X'F1'

2061 MAXLENWORD 4096

2064 WRREC LDX ZERO

2067 WLOOP TD OUTPUT

206a JEQ WLOOP

206d LDCH BUFFER,X

2070 WD OUTPUT

2073 TIX LENGTH

2076 JLT WLOOP

2079 RSUB

207a OUTPUTBYTE X'05'

207d END FIRST

**/\*errors\*/**

/\*source file\*/

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)

LDA LENGTH TEST FOR EOF (LENGTH = 0)

COMP ZERO

JEQ ENDFIL EXIT IF EOF FOUND

JSUB WRREC WRITE OUTPUT RECORD

J CLOOP LOOP

EOF BYTE CEOF'

THREE WORD 3

THREE WORD 0

RETADR RESW 1

LENGTH RESW 1 LENGTH OF RECORD

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')

BUFFER RESB

BUFFER RESB AAA

.

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

OUTPUT BYTE X'5' CODE FOR OUTPUT DEVICE

AAKS END FIRST

/\*symbol table\*/

LABEL | ADDRESS

THREE 1018

RETADR 101e

RLOOP 1024

FIRST 1000

EOF 1018

WLOOP 1033

BUFFER 1030

WRREC 1030

LENGTH 1021

OUTPUT 1042

CLOOP 1003

/\*intermediate file\*/

LOCCTR LABEL OPCODEOPERAND ERRORS

1000 COPY START 1000

1003 FIRST STL RETADR

1006 CLOOP JSUB RDREC

1009 LDA LENGTH

100c LDA LENGTH

100f COMP ZERO

1012 JEQ ENDFIL

1015 JSUB WRREC

1018 J CLOOP

1018 EOF BYTE CEOF' invalid operand!

101b THREE WORD 3

101e THREE WORD 0 duplicated label!

1021 RETADRRESW 1

1024 LENGTHRESW 1

1027 RLOOP TD INPUT

102a JEQ RLOOP

102d RD INPUT

1030 COMP ZERO

1030 BUFFER RESB

1030 BUFFER RESB AAA duplicated label!

1033 WRREC LDX ZERO

1036 WLOOP TD OUTPUT

1039 JEQ WLOOP

103c LDCH BUFFER,X

103f WD OUTPUT

1042 TIX LENGTH

1043 OUTPUTBYTE X'5' invalid operand!

1046 AAKS END FIRST END does not need a label!