|  |
| --- |
| Федеральное агентство связи Государственное образовательное учреждение высшего профессионального образования «Сибирский государственный университет телекоммуникаций и информатики»  Кафедра вычислительных систем |
| Курсовая работа по дисциплине «Архитектура ЭВМ» |
|  |
|  |

|  |
| --- |
|  |

Выполнил:  
Студент группы ИП-813  
Бурдуковский И.А.

Проверил доцент кафедры:  
Майданов Ю. С.

Новосибирск - 2020

Содержание:

1. Постановка задачи   
   2. Выполнение работы   
    2.1. Модель центрального процессора   
    2.2. Транслятор с языка Simple Assembler

2.3. Транслятор с языка Simple Basic

2.4. Пользовательская функция   
3. Пример работы Simple Computer   
4. Вывод   
5. Листинг Программы

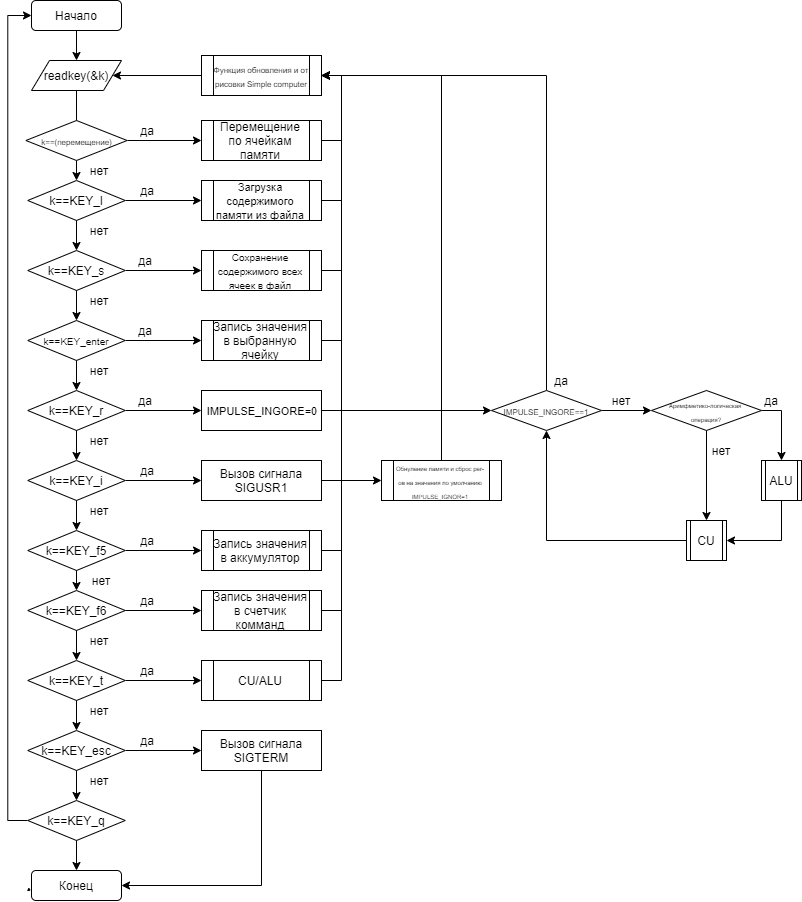
6. Список Литературы

1. Постановка задачи.

В рамках курсовой работы необходимо доработать модель *Simple Computer* так, чтобы она обрабатывала команды, записанные в оперативной памяти. Система команд представлена в таблице 1. Из пользовательских функций необходимо реализовать только одну согласно варианту задания(вариант 54). Для разработки программ требуется создать трансляторы с языков *Simple Assembler* и *Simple Basic*

1. Выполнение работы.

Блок-схемы алгоритмов программы. (SimpleComputer, Assembler, Basic translators)







2.1. Модель центрального процессора.

Для выполнения программ моделью *Simple Computer* необходимо реализовать две функции:

**int *ALU*** (*int command, int operand*) – реализует алгоритм работы арифметико-логического устройства. Если при выполнении функции возникла ошибка, которая не позволяет дальше выполнять программу, то функция возвращает -1, иначе 0;

**int *CU*** (void) – обеспечивает работу устройства управления. Является функцией обратного вызова, вызываемая системным таймером или по нажатию клавиши «t» в случае установленного флага пошаговой работы. Производит чтение из массива – модели оперативной памяти со смещением, хранящемся в регистре - указателе команд InstructionCounter и декодирует полученное значение. В результате декодирования возможны следующие исходы: - устройство управления самостоятельно обрабатывает инструкцию, если команда не требует арифметических вычислений. Возвращает «0». - устройство управления передает обработку команды в АЛУ, если команда требует арифметических вычислений над операндами. К этой группе команд относятся команды, выполняющие арифметические и логические действия. Возвращает «0». - в случае поступления некорректной команды (с недопустимым операционным кодом или операндом) устройство управления останавливает работу SimpleComputer и возвращает «-1».

2.2. Транслятор с языка Simple Assembler.

Задачей транслятора является преобразование текстового файла, содержащего программный код на языке SimpleAssembler, в бинарный файл, содержащий дамп памяти SimpleComputer (дамп содержит программу в машинном коде).

Пример программы на **Simple Assembler**:  
  
00 READ 09 ; (Ввод А)

01 READ 10 ; (Ввод В)

02 LOAD 09 ; (Загрузка А в аккумулятор)

03 SUB 10 ; (Отнять В)

04 JNEG 07 ; (Переход на 07, если отрицательное)

05 WRITE 09 ; (Вывод А)

06 HALT 00 ; (Останов)

07 WRITE 10 ; (Вывод В)

08 HALT 00 ; (Останов)

09 = +0000 ; (Переменная А)

10 = +9999 ; (Переменная В)

2.3 Транслятор с языка Simple Basic

В языке Simple Basic используются следующие операторы: REM, INPUT, PRINT, GOTO, IT, LET, END

Пример программы на **Simple Basic**:

10 REM

20 INPUT A

30 INPUT B

40 LET C = A-B

50 IF C < 0 GOTO 20

60 PRINT C

70 END

Каждая строка программы состоит из номера строки, оператора Simple Basic и параметров. Номера строк должны следовать в возрастающем порядке. Все команды за исключением команды конца программы могут встречаться в программе многократно. Simple Basic должен оперировать с целыми выражениями, включающими операции +, -, \*, и /. Приоритет операций аналогичен C. Для того чтобы изменить порядок вычисления, можно использовать скобки.

2.4 Пользовательская функция

XOR с кодом 54- Логическая операция исключающее ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)

Код функции:

case 0x54:

last\_command = 54;

if((accum^value)>0x7FFF || (accum^value)<-0x7FFF){

sc\_regSet(REG\_OPERATION\_OVERFLOW, 1);

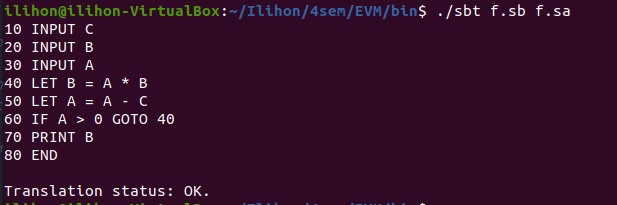
return -1;

}

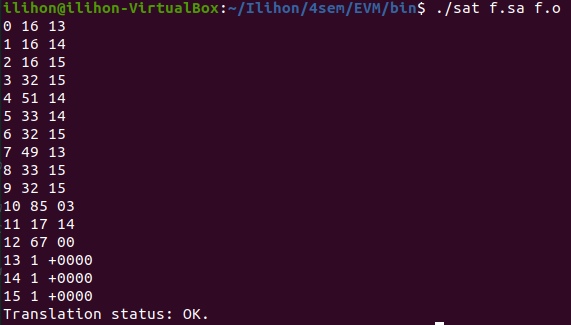
accum = accum ^ value;

break;

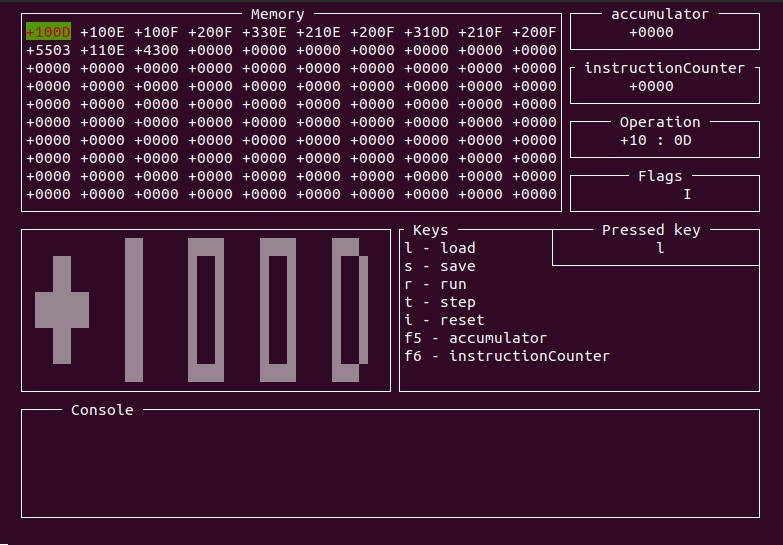
3.Пример работы Simple Computer.

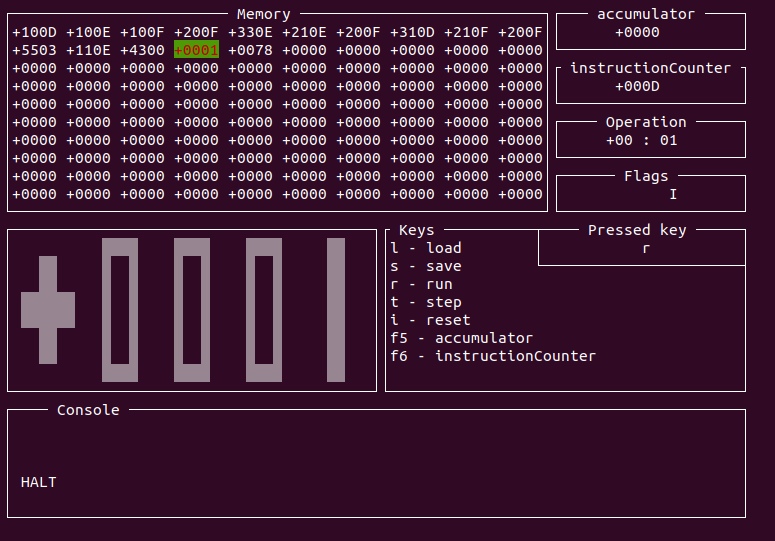
 Транслятор Basic

Транслятор Assembler



Загрузка файла f.o



Расчет факториала 5

4.Вывод.

В рамках данной курсовой работы была реализована модель Simple Computer, выполнение программ которой было осуществлено реализованными функциональными блоками центрального процессора. Для разработки программы были созданы трансляторы с языков Simple Assembler и Simple Basic.

Была разработана простейшая модель вычислительной машины, включающая в себя:  
- память  
- устройства ввода-вывода  
- устройство управления  
- арифметико-логическое устройство

5.Листинг программы

**Main.cpp**

#include "../../lib/SimpleComputer/mySimpleComputer.h"

//#include "SimpleComputerTest.h"

#include "../../lib/Terminal/myTerminal.h"

//#include "../lab3/TerminalTest.h"

#include "../../lib/BigChars/myBigChars.h"

#include "../../lib/ReadKeys/myReadKeys.h"

#include "../../lib/CellChange/SetVariables.h"

#include "../../lib/CU/CU.h"

#include "../../lib/SignalHandler/signals.h"

#include "../../lib/Computer/DisplayInterface.h"

int main()

{

sc\_memoryInit();

sc\_regInit();

//testMemory();

//testRegister();

//testCommandCoding();

//testTerminal();

/\*mt\_clrscr();

bc\_box(5,5,8,16);

int big[2] = {bc\_symbols[10][0], bc\_symbols[10][1]};

bc\_printbigchar(big,6,6,DEFAULT,DEFAULT);

bc\_setbigcharpos(big,3,1,1);

bc\_setbigcharpos(big,2,1,0);

bc\_printbigchar(big,6,6,DEFAULT,DEFAULT);

\*/

sc\_regSet(REG\_IMPULSE\_IGNORE,1);

interface();

return 0;

}

**CU.cpp**

#include "../SimpleComputer/mySimpleComputer.h"

#include "../Terminal/myTerminal.h"

#include "../BigChars/myBigChars.h"

#include "../ReadKeys/myReadKeys.h"

#include "../CellChange/SetVariables.h"

#include "../SignalHandler/signals.h"

#include "../Computer/DisplayInterface.h"

#include "CU.h"

int last\_command = 0;

void CU(){

last\_command = 0;

int value;

int command;

int operand;

sc\_memoryGet(instructionCounter, &value);

if(sc\_commandDecode(value,&command, &operand) != 0){

sc\_regSet(REG\_IMPULSE\_IGNORE, 1);

return;

}

if(operand < 0 | operand >=100){

sc\_regSet(REG\_INVALID\_COMMAND,1);

sc\_regSet(REG\_IMPULSE\_IGNORE, 1);

return;

}

if(instructionCounter>=99){

sc\_regSet(REG\_MEMORY\_OVERFLOW, 1);

sc\_regSet(REG\_IMPULSE\_IGNORE, 1);

instructionCounter = 0;

return;

}

if((command>=0x30 && command <=0x33) || command == 0x54 || command == 0x67 || command == 0x68){

if(ALU(command,operand) !=0){

sc\_regSet(REG\_IMPULSE\_IGNORE, 1);

return;

}

}

value = 0;

switch (command) {

case 0x10:

last\_command = 10;

set\_mem(operand);

break;

case 0x11:

last\_command = 11;

sc\_memoryGet(operand, &value);

mt\_gotoXY(27,4);

if (value < 0)

printf("WRITE: -%04X\n", -value);

else

printf("WRITE: +%04X\n", value);

break;

case 0x20:

last\_command = 20;

sc\_memoryGet(operand, &value);

accum = value;

break;

case 0x21:

last\_command = 21;

sc\_memorySet(operand, accum);

break;

case 0x40:

last\_command = 40;

instructionCounter = operand-1;

break;

case 0x41:

last\_command = 41;

if(accum < 0){

instructionCounter = operand-1;

}

break;

case 0x42:

last\_command = 42;

if(accum == 0){

instructionCounter = operand-1;

}

break;

case 0x43:

last\_command = 43;

sc\_regSet(REG\_IMPULSE\_IGNORE, 1);

break;

case 0x55:

last\_command = 55;

if(accum > 0){

instructionCounter = operand-1;

}

break;

}

return;

}

int ALU(int command, int operand){

int value;

sc\_memoryGet(operand, &value);

switch(command){

case 0x30:

last\_command = 30;

if((accum+value)>0x7FFF || (accum+value)<-0x7FFF){

sc\_regSet(REG\_OPERATION\_OVERFLOW, 1);

return -1;

}

accum = accum + value;

break;

case 0x31:

last\_command = 31;

if((accum-value)>0x7FFF || (accum-value)<-0x7FFF){

sc\_regSet(REG\_OPERATION\_OVERFLOW, 1);

return -1;

}

accum = accum - value;

break;

case 0x32:

last\_command = 32;

if(value == 0){

sc\_regSet(REG\_DIVISION\_ZERO, 1);

return -1;

}

if((accum/value)>0x7FFF || (accum/value)<-0x7FFF){

sc\_regSet(REG\_OPERATION\_OVERFLOW, 1);

return -1;

}

accum = accum / value;

break;

case 0x33:

last\_command = 33;

if((accum\*value)>0x7FFF || (accum\*value)<-0x7FFF){

sc\_regSet(REG\_OPERATION\_OVERFLOW, 1);

return -1;

}

accum = accum \* value;

break;

case 0x54:

last\_command = 54;

if((accum^value)>0x7FFF || (accum^value)<-0x7FFF){

sc\_regSet(REG\_OPERATION\_OVERFLOW, 1);

return -1;

}

accum = accum ^ value;

break;

case 0x67:

last\_command = 67;

if((value<<accum)>0x7FFF || (value<<accum)<-0x7FFF){

sc\_regSet(REG\_OPERATION\_OVERFLOW, 1);

return -1;

}

accum = value<<accum;

break;

case 0x68:

last\_command = 68;

if((value>>accum)>0x7FFF || (value>>accum)<-0x7FFF){

sc\_regSet(REG\_OPERATION\_OVERFLOW, 1);

return -1;

}

accum = value>>accum;

break;

}

return 0;

}

**SimpleBasic.cpp**

#include "SimpleBasic.h"

char vars[100], asm\_str[100][100], asm\_str\_out[100][100];

int numbers\_mas[100];

char clearBackspace(char \*str)

{

char \* current = str;

char \* last = str;

for (; \*current != '\0' ; ++current)

{

if (\*current != ' ')

{

\*last = \*current;

last++;

}

}

\*last = '\0';

}

int getVarAddr(char varName, int offset)

{

int i;

for (i = 0; i < sizeof(vars); i++)

{

if (vars[i] == varName) return offset + i;

else if (strcmp((vars + i), "\0") == 0) break;

}

\*(vars + i) = varName;

return offset + i;

}

int getGotoAddr(int gotoLine)

{

int ln, num, addr;

for (int k = 0; k < sizeof(asm\_str)/sizeof(asm\_str[0]); k++)

{

sscanf(asm\_str[k], "%d %d %d", &ln, &num, &addr);

if (num == gotoLine) return addr;

}

return -1;

}

char getVarName(int varAddr, int offset)

{

if ((varAddr - offset >= 0) && (strcmp((vars + varAddr - offset), "\0") != 0)) vars + varAddr - offset;

else return '\0';

}

int SimpleBasic(int argc, char\*\* argv){

FILE \*file\_sb, \*file\_sa;

if( argc != 3)

{

printf("Usage: sbt file.sb file.sa\n ");

return -1;

}

if ((file\_sb = fopen (argv[1], "rb" )) <= 0)

{

printf("Can`t open '%s' file.\n", argv[1]);

return -1;

}

if ((file\_sa = fopen (argv[2], "wb" )) <= 0)

{

printf("Can`t create '%s' file.\n", argv[2]);

return -1;

}

for (int i = 0; i < sizeof(asm\_str)/sizeof(asm\_str[0]); i++) sprintf(asm\_str[i], "%s", "");

for (int i = 0; i < sizeof(asm\_str\_out)/sizeof(asm\_str\_out[0]); i++) sprintf(asm\_str\_out[i], "%s", "");

for (int i = 0; i < sizeof(vars); i++) strcpy(&vars[i], "\0");

int var\_offset, strnum, asm\_addr = 0, last\_strnum = 0;

int max\_strnum, goto\_addr, goto\_line;

char opr[6], str[100], ch;

char line[100];

int linenum = 1;

do

{

fgets (line, sizeof(line), file\_sb);

if(feof(file\_sb)){

break;

}

printf("%s",line);

if (strcmp(line, "\n") != 0)

{

if (sscanf(line, "%d %s", &strnum, opr) >= 2)

{

if (strnum <= last\_strnum) {

printf("Translation error 1\n");

return -1;

}

if (strcmp(opr, "REM") == 0)

{

linenum++;

continue;

}

else if (strcmp(opr, "END") == 0)

{

if (sscanf(line, "%d %s %s", &strnum, opr, str) != 2){

printf("Translation error 2\n");

return -1;

}

sprintf(asm\_str[asm\_addr], "%d %d %d %s", linenum, strnum, asm\_addr, "HALT\n");

}

else if (strcmp(opr, "INPUT") == 0)

{

if (sscanf(line, "%d %s %c %s", &strnum, opr, &ch, str) != 3){

printf("Translation error 3\n");

return -1;

}

sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", linenum, strnum, asm\_addr, "READ", ch, "\n");

}

else if (strcmp(opr, "LET") == 0)

{

char \*s = strstr(line, "LET") + 4;

clearBackspace(s);

char ch1, ch2, ch3, ch4;

int d;

if (sscanf(s, "%c=%d",&ch1,&d) == 2){

sprintf(asm\_str[asm\_addr], "%d %d %d %s %d%s", linenum, strnum, asm\_addr, "LOAD", d, "\n");

asm\_addr++;

sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", -1, -1, asm\_addr, "STORE", ch1, "\n");

asm\_addr++;

last\_strnum = strnum;

linenum++;

continue;

}

if (sscanf(s, "%c=%c%c%c%s", &ch1, &ch2, &ch3, &ch4, str) != 4){

printf("Translation error 4\n");

return -1;

}

sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", linenum, strnum, asm\_addr, "LOAD", ch2, "\n");

asm\_addr++;

if (ch3 == '+') sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", -1, -1, asm\_addr, "ADD", ch4, "\n");

else if (ch3 == '-') sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", -1, -1, asm\_addr, "SUB", ch4, "\n");

else if (ch3 == '\*') sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", -1, -1, asm\_addr, "MUL", ch4, "\n");

else if (ch3 == '/') sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", -1, -1, asm\_addr, "DIVIDE", ch4, "\n");

else{

printf("Translation error 5\n");

return -1;

}

asm\_addr++;

sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", -1, -1, asm\_addr, "STORE", ch1, "\n");

}

else if (strcmp(opr, "IF") == 0)

{

char \*s = strstr(line, "IF") + 2;

clearBackspace(s);

char ch1, ch2, ch3;

if (sscanf(s, "%c%c%cGOTO%d%s", &ch1, &ch2, &ch3, &goto\_line, str) != 4){

printf("Translation error 6\n");

return -1;

}

sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", linenum, strnum, asm\_addr, "LOAD", ch1, "\n");

asm\_addr++;

if ((ch2 == '<') && (ch3 == '0')) sprintf(asm\_str[asm\_addr], "%d %d %d %s %d%s", -1, -1, asm\_addr, "JNEG", goto\_line, "\n");

else if ((ch2 == '>') && (ch3 == '0')) sprintf(asm\_str[asm\_addr], "%d %d %d %s %d%s", -1, -1, asm\_addr, "JNS", goto\_line, "\n");

else if ((ch2 == '=') && (ch3 == '0')) sprintf(asm\_str[asm\_addr], "%d %d %d %s %d%s", -1, -1, asm\_addr, "JZ", goto\_line, "\n");

else{

printf("Translation error 7\n");

return -1;

}

}

else if (strcmp(opr, "GOTO") == 0)

{

if (sscanf(line, "%d %s %d %s", &strnum, opr, &goto\_line, str) != 3){

printf("Translation error 8\n");

return -1;

}

sprintf(asm\_str[asm\_addr], "%d %d %d %s %d%s", linenum, strnum, asm\_addr, "JUMP", goto\_line, "\n");

}

else if (strcmp(opr, "PRINT") == 0)

{

if (sscanf(line, "%d %s %c %s", &strnum, opr, &ch, str) != 3){

printf("Translation error 9\n");

return -1;

}

sprintf(asm\_str[asm\_addr], "%d %d %d %s %c%s", linenum, strnum, asm\_addr, "WRITE", ch, "\n");

}

else{

printf("Translation error 11\n");

return -1;

}

asm\_addr++;

last\_strnum = strnum;

}

else{

printf("Translation error 12\n");

return -1;

}

}

linenum++;

}

while (!feof(file\_sb));

fclose (file\_sb);

max\_strnum = strnum;

var\_offset = asm\_addr;

for (int i = 0; i < sizeof(asm\_str)/sizeof(asm\_str[0]); i++)

{

if (strcmp(asm\_str[i], "") != 0)

{

sscanf(asm\_str[i], "%d %d %d %s", &linenum, &strnum, &asm\_addr, opr);

if ((strcmp(opr, "READ") == 0) || (strcmp(opr, "WRITE") == 0) ||

(strcmp(opr, "LOAD") == 0) || (strcmp(opr, "STORE") == 0) ||

(strcmp(opr, "ADD") == 0) || (strcmp(opr, "SUB") == 0) ||

(strcmp(opr, "DIVIDE") == 0) || (strcmp(opr, "MUL") == 0))

{

sscanf(asm\_str[i], "%d %d %d %s %c", &linenum, &strnum, &asm\_addr, opr, &ch);

int var\_addr = getVarAddr(ch, var\_offset);

sprintf(asm\_str\_out[i], "%02d %s %02d%s", asm\_addr, opr, var\_addr, "\n");

int d;

if(sscanf(asm\_str[i], "%d %d %d %s %d", &linenum, &strnum, &asm\_addr, opr, &d) == 5){

if(strcmp(opr,"LOAD")==0){

numbers\_mas[var\_addr] = d;

}

}

}

else if ((strcmp(opr, "JUMP") == 0) || (strcmp(opr, "JNEG") == 0) ||

(strcmp(opr, "JNS") == 0) || (strcmp(opr, "JZ") == 0))

{

sscanf(asm\_str[i], "%d %d %d %s %d", &linenum, &strnum, &asm\_addr, opr, &goto\_line);

int goto\_addr = getGotoAddr(goto\_line);

if (goto\_addr == -1){

printf("Translation error 13\n");

return -1;

}

sprintf(asm\_str\_out[i], "%02d %s %02d%s", asm\_addr, opr, goto\_addr, "\n");

}

else if (strcmp(opr, "HALT") == 0)

{

sscanf(asm\_str[i], "%d %d %d %s", &linenum, &strnum, &asm\_addr, opr);

sprintf(asm\_str\_out[i], "%02d %s%s", asm\_addr, opr, " 00\n");

}

}

else

{

if (i >= var\_offset) {

if (getVarName(i, var\_offset) != '\0')

sprintf(asm\_str\_out[i], "%02d %c %s", i, '=', "+0000\n");

if(numbers\_mas[i]!=0){

sprintf(asm\_str\_out[i], "%02d %c +%04X%s", i, '=',numbers\_mas[i],"\n");

}

}

}

}

for (int i = 0; i < sizeof(asm\_str)/sizeof(asm\_str[0]); i++)

fwrite (asm\_str\_out[i], 1 , strlen(asm\_str\_out[i]), file\_sa);

fclose (file\_sa);

printf("Translation status: OK.\n");

return 0;

}

**SimpleAssembler.cpp**

#include "SimpleAssembler.h"

#include "../SimpleComputer/mySimpleComputer.h"

int CommandAnalis(char \*command){

if(strcmp(command,"READ") == 0)return 0x10;

if(strcmp(command,"WRITE") == 0)return 0x11;

if(strcmp(command,"LOAD") == 0)return 0x20;

if(strcmp(command,"STORE") == 0)return 0x21;

if(strcmp(command,"ADD") == 0)return 0x30;

if(strcmp(command,"SUB") == 0)return 0x31;

if(strcmp(command,"DIVIDE") == 0)return 0x32;

if(strcmp(command,"MUL") == 0)return 0x33;

if(strcmp(command,"JUMP") == 0)return 0x40;

if(strcmp(command,"JNEG") == 0)return 0x41;

if(strcmp(command,"JZ") == 0)return 0x42;

if(strcmp(command,"HALT") == 0)return 0x43;

if(strcmp(command,"XOR") == 0)return 0x54;

if(strcmp(command,"JNS") == 0)return 0x55;

if(command[0] == '=')return 1;

return -1;

}

int SimpleAssembler(int argc, char\*\* argv){

FILE \*file\_sa, \*file\_o;

if( argc != 3)

{

printf("Usage: sat file.sa file.o\n ");

return -1;

}

if ((file\_sa = fopen (argv[1], "rb" )) <= 0)

{

printf("Can`t open '%s' file.\n", argv[1]);

return -1;

}

sc\_memoryInit();

char line[100],ch;

char strComm[10],operand\_str[10];

int value,address,command,operand;

do

{

fgets(line,sizeof(line),file\_sa);

if(feof(file\_sa)){

break;

}

if(sscanf(line,"%d %s %s",&address,strComm,operand\_str)<3)

{

printf("Translation error 1\n");

return -1;

}

command = CommandAnalis(strComm);

printf("%d %d %s\n",address,command,operand\_str);

if(command != -1)

{

if(command != 1)

{

if(sscanf(operand\_str, "%d", &operand) !=1 || sc\_commandEncode(command,operand,&value)==-1 || sc\_memorySet(address,value)==-1)

{

printf("Translation error 2\n");

return -1;

}

}

else {

if(sscanf(operand\_str, "%c%04X", &ch, &value)!=2){

printf("Translation error 3\n");

return -1;

}

if(ch == '-'){

value = -value;

}

if(sc\_memorySet(address,value)==-1)

{

printf("Translation error 4\n");

return -1;

}

}

}

else

{

printf("Translation error 5\n");

return -1;

}

}

while(!feof(file\_sa));

fclose(file\_sa);

if (sc\_memorySave(argv[2]) == -1)

{

printf("Can`t create '%s' file.\n", argv[2]);

return -1;

}

printf("Translation status: OK.\n");

return 0;

}

**DisplayInterface.cpp**

#include "../SimpleComputer/mySimpleComputer.h"

#include "../Terminal/myTerminal.h"

#include "../BigChars/myBigChars.h"

#include "../ReadKeys/myReadKeys.h"

#include "../CellChange/SetVariables.h"

#include "../CU/CU.h"

#include "../SignalHandler/signals.h"

#include "DisplayInterface.h"

int interface(){

int impulse\_status = 0;

char buf[128];

enum keys key = KEY\_nothing;

cout<<endl;

signal(SIGUSR1, reset\_signal\_handler);

signal(SIGALRM, signal\_handler);

int x=instructionCounter % 10;

int y=instructionCounter / 10;

int address=y\*10 + x;

mt\_clrscr();

while(key != KEY\_q) {

print\_interface(address, key);

cout<<endl;

stdin = freopen(nullptr, "r", stdin);

rk\_readkey(&key);

printf("\033(B");

if(sc\_regGet(REG\_IMPULSE\_IGNORE, &impulse\_status)==0 && impulse\_status==1){

switch(key){

case KEY\_enter:

set\_mem(address);

break;

case KEY\_l:

mt\_gotoXY(27,5);

printf("Input filename: ");

scanf("%s", buf);

if(strcmp(buf,"abort")==0)

break;

sc\_memoryLoad(buf);

sc\_regSet(REG\_IMPULSE\_IGNORE, 1);

x = instructionCounter % 10;

y = instructionCounter / 10;

last\_command = 0;

break;

case KEY\_s:

mt\_gotoXY(27,5);

printf("Input filename: ");

scanf("%s", buf);

if(strcmp(buf,"abort")==0)

break;

sc\_memorySave(buf);

break;

case KEY\_f5:

set\_accum();

break;

case KEY\_f6:

set\_instructionCounter();

x = instructionCounter % 10;

y = instructionCounter / 10;

break;

case KEY\_up:

y--;

if(y==-1)

y = 9;

break;

case KEY\_down:

y++;

if(y==10)

y = 0;

break;

case KEY\_left:

x--;

if(x==-1)

x = 9;

break;

case KEY\_right:

x++;

if(x==10)

x = 0;

break;

case KEY\_t:

sc\_regInit();

sc\_regSet(REG\_IMPULSE\_IGNORE, 1);

signal\_handler(SIGALRM);

x = instructionCounter % 10;

y = instructionCounter / 10;

break;

}

}

switch (key) {

case KEY\_q:

mt\_gotoXY(27,5);

printf("Poweroff SimpleComputer...\n");

break;

case KEY\_esc:

raise(SIGTERM);

break;

case KEY\_i:

raise(SIGUSR1);

x = instructionCounter % 10;

y = instructionCounter / 10;

break;

case KEY\_r:

int value;

if(sc\_regGet(REG\_IMPULSE\_IGNORE, &value)==0 && value == 1){

sc\_regInit();

signal\_handler(SIGALRM);

}

else {

sc\_regSet(REG\_IMPULSE\_IGNORE,1);

alarm(0);

}

x = instructionCounter % 10;

y = instructionCounter / 10;

break;

}

address = x+(y\*10);

}

mt\_gotoXY(32,0);

return 0;

}

void print\_interface(int address, enum keys key){

mt\_clrscr();

print\_term\_boxes();

print\_mem(address);

PrintBigChars(instructionCounter);

print\_accum();

print\_instruction\_Counter(instructionCounter);

print\_operation(instructionCounter);

print\_reg();

mt\_gotoXY(16,70);

print\_key(key);

mt\_gotoXY(29,5);

print\_last\_command();

mt\_gotoXY(32,0);

cout<<endl;

return;

}

void print\_last\_command(){

int value;

int command;

int operand;

switch (last\_command) {

case 10:

printf("READ\n");

break;

case 11:

sc\_memoryGet(instructionCounter-1, &value);

if(sc\_commandDecode(value,&command, &operand) == 0){

sc\_memoryGet(operand, &value);

if (value < 0)

printf("WRITE: -%04X\n", -value);

else

printf("WRITE: +%04X\n", value);

}

break;

case 20:

if (accum < 0)

printf("LOAD: -%04X\n", -accum);

else

printf("LOAD: +%04X\n", accum);

break;

case 21:

sc\_memoryGet(instructionCounter-1, &value);

if(sc\_commandDecode(value,&command, &operand) == 0){

if (accum < 0)

printf("STORE: -%04X to %02X memmory cell\n", -accum, operand);

else

printf("STORE: +%04X to %02X memmory cell\n", accum, operand);

}

break;

case 30:

printf("ADD\n");

break;

case 31:

printf("SUB\n");

break;

case 32:

printf("DIVIDE\n");

break;

case 33:

printf("MUL\n");

break;

case 40:

printf("JUMP to %02X instructionCounter\n", instructionCounter);

break;

case 41:

printf("JNEG\n");

break;

case 42:

printf("JZ\n");

break;

case 43:

printf("HALT\n");

break;

case 54:

printf("XOR\n");

break;

case 55:

printf("JNS\n");

break;

case 67:

printf("LOGLC\n");

break;

case 68:

printf("LOGRC\n");

break;

}

return;

}

void print\_accum(){

mt\_gotoXY(4,71);

if(accum<0){

printf("-");

printf("%04X",~accum+1);

}else{

printf("+");

printf("%04X",accum);

}

return;

}

void print\_instruction\_Counter(int address){

mt\_gotoXY(7,71);

if(address<0 || address>99){

sc\_regSet(REG\_MEMORY\_OVERFLOW,1);

return;

}

printf("+%04X",address);

return;

}

void print\_operation(int address){

int value=0;

int result[4];

sc\_memoryGet(address,&value);

mt\_gotoXY(10,70);

if(value<0){

printf("-");

value = -value;

}else{

printf("+");

}

memory\_convert(value, result);

printf("%X%X : %X%X",result[0],result[1],result[2],result[3]);

return;

}

void print\_reg(){

mt\_gotoXY(13,71);

int value=0;

for(int i=1;i<6;i++){

sc\_regGet(i,&value);

if(value==1){

switch (i) {

case REG\_OPERATION\_OVERFLOW:

printf("P ");

break;

case REG\_DIVISION\_ZERO:

printf("0 ");

break;

case REG\_MEMORY\_OVERFLOW:

printf("M ");

break;

case REG\_IMPULSE\_IGNORE:

printf("I ");

break;

case REG\_INVALID\_COMMAND:

printf("E ");

break;

}

}

else{

printf(" ");

}

}

return;

}

void print\_mem(int address){

int value=0;

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

mt\_gotoXY(i+4,4);

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

if(i\*10 + j == address){

mt\_setbgcolor(GREEN);

}

if(i\*10 + j == instructionCounter){

mt\_setfgcolor(RED);

}

sc\_memoryGet (i\*10 + j, &value);

if(value<0){

printf("-");

value = -value;

}else{

printf("+");

}

printf("%04X",value);

mt\_setbgcolor(DEFAULT);

mt\_setfgcolor(DEFAULT);

if(j!=9)

printf(" ");

}

}

return;

}

void print\_term\_boxes(){

bc\_box(3,3,10,59);

mt\_gotoXY(3,28);

cout <<" Memory ";

bc\_box(3,64,1,20);

mt\_gotoXY(3,68);

cout <<" accumulator ";

bc\_box(6,64,1,20);

mt\_gotoXY(6,65);

cout <<" instructionCounter ";

bc\_box(9,64,1,20);

mt\_gotoXY(9,69);

cout <<" Operation ";

bc\_box(12,64,1,20);

mt\_gotoXY(12,71);

cout <<" Flags ";

bc\_box(15,3,8,40);

bc\_box(15,45,8,39);

bc\_box(15,62,1,22);

mt\_gotoXY(15,62);

bc\_printA("w");

mt\_gotoXY(17,85);

bc\_printA("u");

mt\_gotoXY(15,46);

cout << " Keys ";

mt\_gotoXY(15,67);

cout << " Pressed key ";

mt\_gotoXY(16,46);

cout <<"l - load";

mt\_gotoXY(17,46);

cout <<"s - save";

mt\_gotoXY(18,46);

cout <<"r - run";

mt\_gotoXY(19,46);

cout <<"t - step";

mt\_gotoXY(20,46);

cout <<"i - reset";

mt\_gotoXY(21,46);

cout <<"f5 - accumulator";

mt\_gotoXY(22,46);

cout <<"f6 - instructionCounter";

bc\_box(25,3,5,81);

mt\_gotoXY(25,8);

cout <<" Console ";

return;

}

void PrintBigChars(int address){

int result[4];

int value=0;

int otriz =0;

sc\_memoryGet (address, &value);

if(value<0){

value = -value;

otriz =1;

}

memory\_convert(value, result);

int big[2];

if(otriz == 1){

ToBigChar(17,big);

}else{

ToBigChar(16,big);

}

int x=16,y=12;

bc\_printbigchar(big,x,y-8,DEFAULT,DEFAULT);

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

ToBigChar(result[i],big);

bc\_printbigchar(big,x,y+8\*i,DEFAULT,DEFAULT);

}

return;

}

void memory\_convert(int mem, int \*conv){

int n=3;

for(int i=n;i>=0;i--){

conv[i]=mem % 16;

mem = mem / 16;

}

}

6.Список литературы

1. Организация ЭВМ и систем. Практикум // С.Н. Мамойленко, Новосибирск: ГОУ ВПО «СибГУТИ», 2005 г.

2. Архитектура компьютера. 4-е изд. // Э. Танненбаум. – СПб.: Питер, 2003.

3. Организация ЭВМ. 5-е изд. / К. Хамахер, З. Вранешич, С. Заки. – СПб.: Питер; Киев: Издательская группа BHV, 2003.

4. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: учебник для ВУЗов. – СПб.: Питер, 2004.