Министерство цифрового развития, связи

и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра вычислительных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по дисциплине «**Архитектура ЭВМ**»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. ИС-241  «\_\_» мая 2024 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | / Кулик П.Е. / |
|  |  |  |
| Проверил:  преподаватель  «\_\_» мая 2024 г. | **\_\_\_\_\_\_\_\_\_\_\_\_** | / Майданов Ю.С. / |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2024

Оглавление

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc166422507)

[Транслятор с языка Simple Assembler 3](#_Toc166422508)

[Транслятор с языка Simple Basic 4](#_Toc166422509)

[Оформление отчета по курсовой работе 5](#_Toc166422510)

[ВЫПОЛНЕНИЕ РАБОТЫ 6](#_Toc166422511)

[БЛОК-СХЕМЫ АЛГОРИТМОВ 7](#_Toc166422512)

[ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 10](#_Toc166422513)

[console 10](#_Toc166422514)

[include 40](#_Toc166422515)

[myBigChars 43](#_Toc166422516)

[myReadKey 46](#_Toc166422517)

[mySimpleComputer 50](#_Toc166422518)

[myTerm 59](#_Toc166422519)

[simpleassembler 64](#_Toc166422520)

[simplebasic 73](#_Toc166422521)

[simplecomputer 91](#_Toc166422522)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 93](#_Toc166422523)

[ЗАКЛЮЧЕНИЕ 96](#_Toc166422524)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 97](#_Toc166422525)

# ПОСТАНОВКА ЗАДАЧИ

В рамках курсовой работы необходимо:

➢ Разработать транслятор с языка Simple Basic. Итог работы транслятора – бинарный файл с образом оперативной памяти Simple Computer, который можно загрузить в модель и выполнить;

➢ Доработать модель Simple Computer – реализовать алгоритм работы блока «L1-кэш команд и данных» и модифицировать работу контроллера оперативной памяти и обработчика прерываний таким образом, чтобы учитывался простой процессора при прямом доступе к оперативной памяти;

➢ Разработать транслятор с языка Simple Basiс. Итог работы транслятора – текстовый файл с программой на языке Simple Basic.

## Транслятор с языка Simple Assembler

Разработка программ для Simple Computer может осуществляться с использованием низкоуровневого языка Simple Assembler. Для того чтобы программа могла быть обработана Simple Computer необходимо реализовать транслятор, переводящий текст Simple Assembler в бинарный формат, которым может быть считан консолью управления. 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)

Программа транслируется по строкам, задающим значение одной ячейки памяти. Каждая строка состоит как минимум из трех полей: адрес ячейки памяти, команда (символьное обозначение), операнд. Четвертым полем может быть указан комментарий, который обязательно должен начинаться с символа точка с запятой. Название команд представлено в таблице 1. Дополнительно используется команда =, которая явно задает значение ячейки памяти в формате вывода его на экран консоли (+XXXX).

Команда запуска транслятора должна иметь вид: sat файл.sa файл.o, где файл.sa – имя файла, в котором содержится программа на Simple Assembler, файл.o – результат трансляции.

## Транслятор с языка Simple Basic

Для упрощения программирования пользователю модели Simple Computer должен быть предоставлен транслятор с высокоуровневого языка Simple Basic. Файл, содержащий программу на Simple Basic, преобразуется в файл с кодом Simple Assembler. Затем Simple Assembler-файл транслируется в бинарный формат.

В языке Simple Basic используются следующие операторы: rem, input, output, goto, if, let, end. Пример программы на Simple Basic:

10 REM Это комментарий

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.

Транслятор должен распознавания только букв верхнего регистра, то есть все символы в программе на Simple Basic должны быть набраны в верхнем регистре (символ нижнего регистра приведет к ошибке). Имя переменной может состоять только из одной буквы. Simple Basic оперирует только с целыми значениями переменных, в нем отсутствует объявление переменных, а упоминание переменной автоматически вызывает её объявление и присваивает ей нулевое значение. Синтаксис языка не позволяет выполнять операций со строками.

## Оформление отчета по курсовой работе

Отчет о курсовой работе представляется в виде пояснительной записки (ПЗ), к которой прилагается диск с разработанным программным обеспечением. В пояснительную записку должны входить:

• титульный лист;

• полный текст задания к курсовой работе;

• реферат (объем ПЗ, количество таблиц, рисунков, схем, программ, приложений, краткая характеристика и результаты работы);

• содержание:

· постановка задачи исследования;

· блок-схемы используемых алгоритмов;

· программная реализация;

· результаты проведенного исследования;

· выводы;

• список использованной литературы;

• подпись, дата.

Пояснительная записка должна быть оформлена на листах формата А4, имеющих поля. Все листы следует сброшюровать и пронумеровать.

# ВЫПОЛНЕНИЕ РАБОТЫ

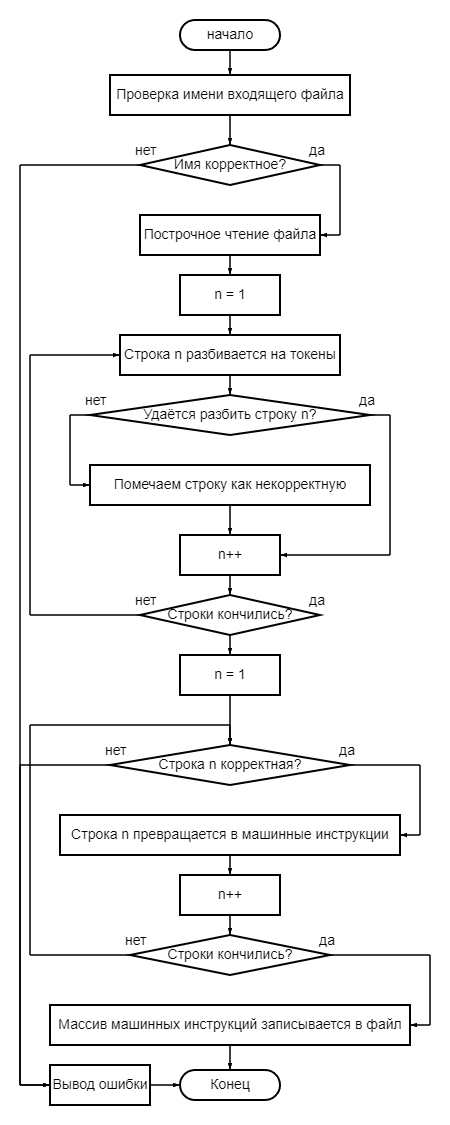
В первую очередь был реализован транслятор с языка simple assembler. Это было самой простой задачей, которая не потребовала поиска дополнительной информации, так как после реализации большей части проекта simple computer уже было понятно, как инструкции simple assembler преобразуются в машинные инструкции.

За этим последовала реализация транслятора с simple basic. Данная задача оказалась менее тривиальной, так как язык simple basic поддерживает арифметические выражения, составленные из множества операций, которые обладают определённым порядком действий. Для решения задачи правильного порядка выполнения пришлось узнать о том, что такое обратная польская нотация и реализовать её. Помимо этого, в языке simple basic могут быть как константы, так и переменные, для поддержки которых требуется построить таблицу символов, чтобы правильно расположить их в памяти и они не пересекались с инструкциями. Проблема размещения данных в памяти была решена очень просто: так как команды располагаются по порядку, начиная с нулевого адреса, то переменные и константы располагаются в обратном порядке, начиная с последнего адреса. Транслятор был написан не с первой попытки, но задача в итоге была решена.

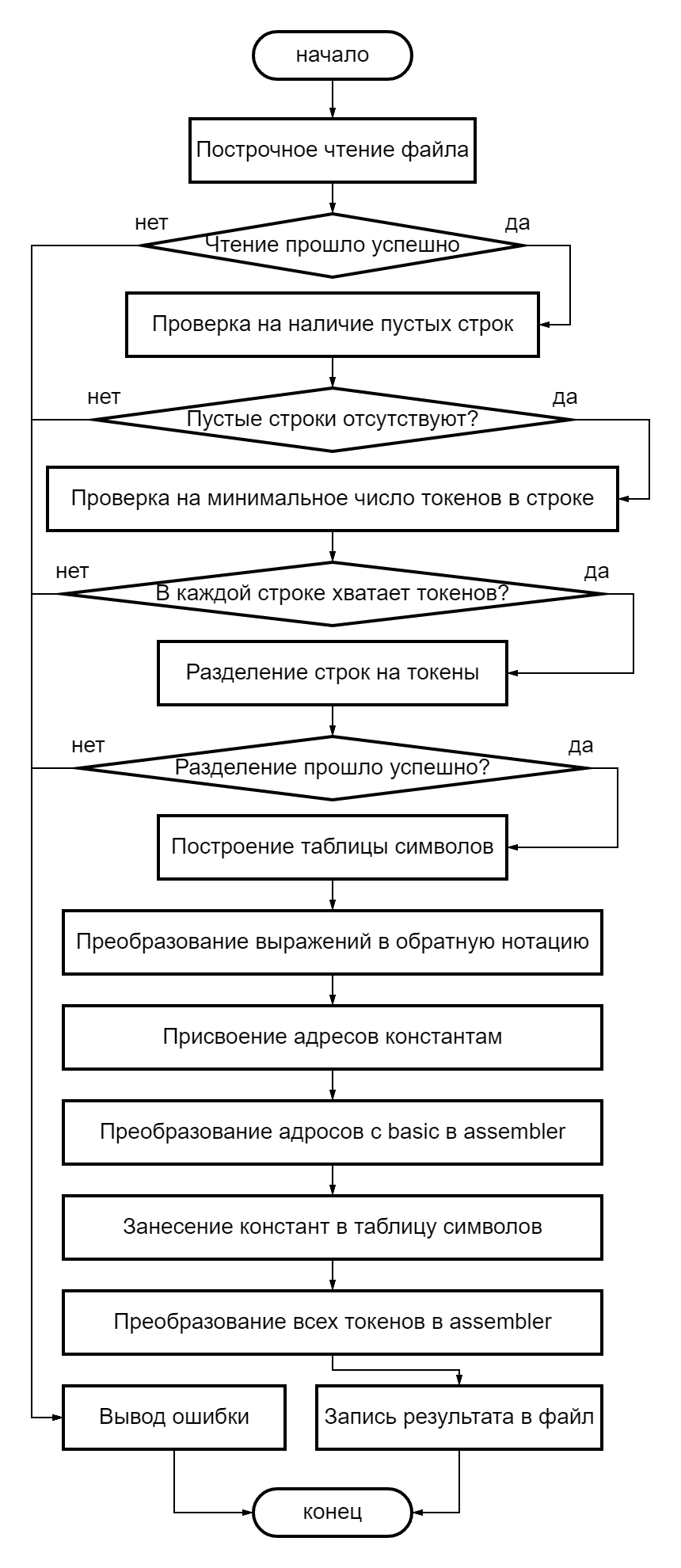
Последним был реализован блок кэша процессора. Предварительно были изучены различные алгоритмы замещения кэша, из которых был выбран LRU кэш как наиболее оптимальный, так как он хорошо работает, учитывая важность часто используемых строк и при этом лёгок в реализации. В моей реализации за каждой строкой кэша закреплена переменная downtime. При каждом обращении к памяти эта переменная обнуляется для той строки, которая была запрошена и инкрементируется для всех остальных строк.

# БЛОК-СХЕМЫ АЛГОРИТМОВ

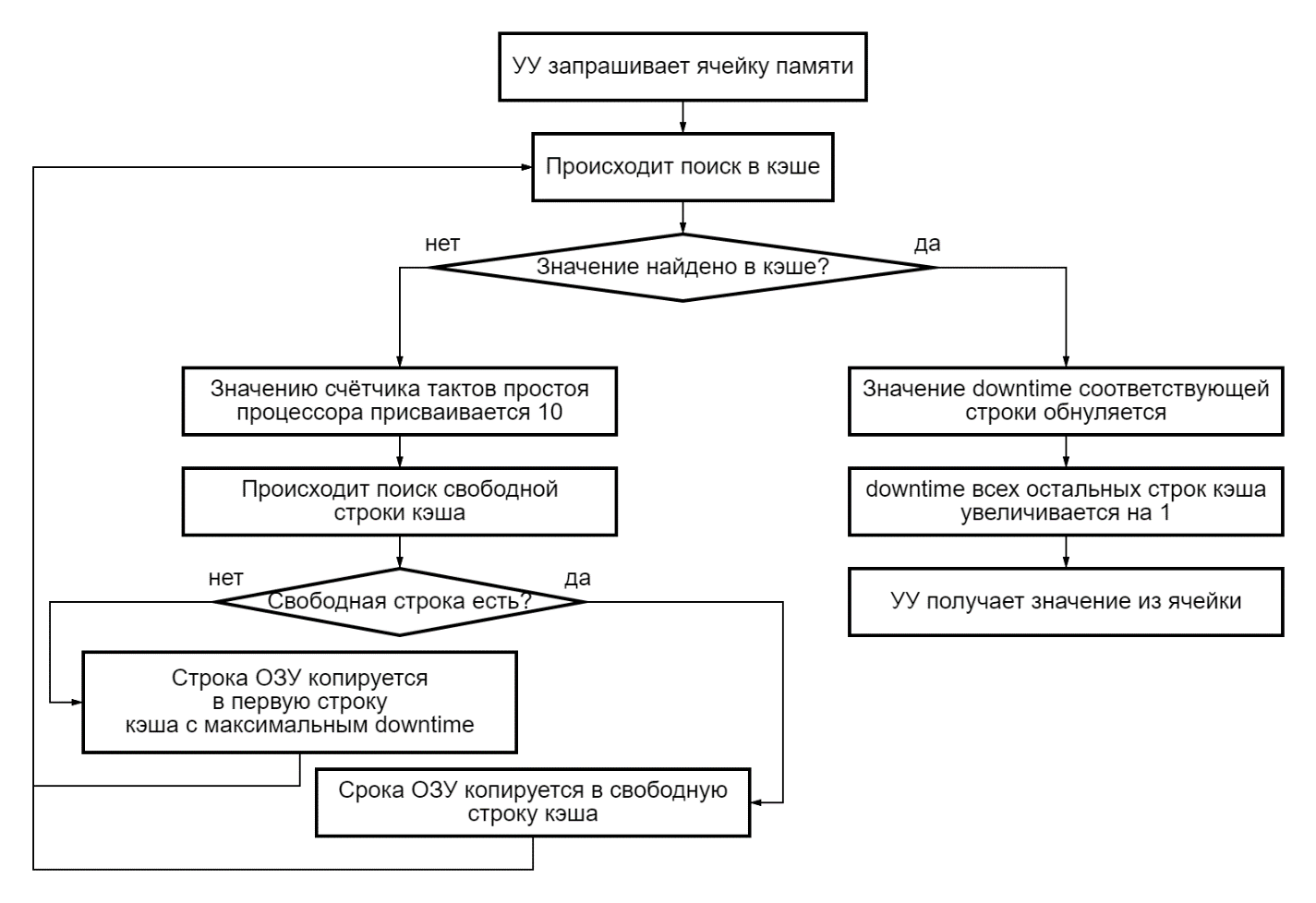
1. Транслятор с Simple Assembler



1. Транслятор с Simple Basic



1. Кэш



# ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

## console

alu.c

#include "console.h"

#include <mySimpleComputer.h>

int

ADD\_C (int cell\_number)

{

int accumulator\_value, memory\_value;

sc\_accumulatorGet (&accumulator\_value);

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return (((accumulator\_value << 17) + (memory\_value << 17)) >> 17) & 0x7FFF;

}

int

SUB\_C (int cell\_number)

{

int accumulator\_value, memory\_value;

sc\_accumulatorGet (&accumulator\_value);

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return (((accumulator\_value << 17) + ((~memory\_value + 1) << 17)) >> 17)

& 0x7FFF;

}

int

DIVIDE\_C (int cell\_number)

{

int accumulator\_value, memory\_value;

sc\_accumulatorGet (&accumulator\_value);

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

if (memory\_value == 0)

{

sc\_regSet (SC\_DIVIDING\_BY\_ZERO, 1);

sc\_regSet (SC\_THROTTLE, 1);

return accumulator\_value;

}

int accumulator\_sign = accumulator\_value >> 14;

int memory\_sign = memory\_value >> 14;

accumulator\_value = accumulator\_value << 17;

memory\_value = memory\_value << 17;

int value = (accumulator\_value / memory\_value) & 0x3FFF;

if (accumulator\_sign ^ memory\_sign)

value = value | 0x4000;

return value;

}

int

MUL\_C (int cell\_number)

{

int accumulator\_value, memory\_value;

sc\_accumulatorGet (&accumulator\_value);

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

int accumulator\_sign = accumulator\_value >> 14;

int memory\_sign = memory\_value >> 14;

if (accumulator\_sign)

accumulator\_value = ~(accumulator\_value - 1);

if (memory\_sign)

memory\_value = ~(memory\_value - 1);

int value = (accumulator\_value \* memory\_value) & 0x3FFF;

if (accumulator\_sign ^ memory\_sign)

value = ((~value & 0x3FFF) + 1) | 0x4000;

return value;

}

int

NOT\_C (void)

{

int memory\_value;

sc\_accumulatorGet (&memory\_value);

return ((~(memory\_value << 17)) >> 17) & 0x7FFF;

}

int

AND\_C (int cell\_number)

{

int accumulator\_value, memory\_value;

sc\_accumulatorGet (&accumulator\_value);

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return accumulator\_value & memory\_value;

}

int

OR\_C (int cell\_number)

{

int accumulator\_value, memory\_value;

sc\_accumulatorGet (&accumulator\_value);

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return accumulator\_value | memory\_value;

}

int

XOR\_C (int cell\_number)

{

int accumulator\_value, memory\_value;

sc\_accumulatorGet (&accumulator\_value);

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return accumulator\_value ^ memory\_value;

}

int

CHL\_C (int cell\_number)

{

int memory\_value;

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return ((memory\_value << 18) >> 17) & 0x7FFF;

}

int

SHR\_C (int cell\_number)

{

int memory\_value;

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return memory\_value >> 1;

}

int

RCL\_C (int cell\_number)

{

int memory\_value;

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return ((memory\_value << 1) | (memory\_value >> 14)) & 0x3FFF;

}

int

RCR\_C (int cell\_number)

{

int memory\_value;

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return ((memory\_value >> 1) | (memory\_value << 14)) & 0x3FFF;

}

int

NEG\_C (int cell\_number)

{

int memory\_value;

if (sc\_memoryGet (cell\_number, &memory\_value) == -2)

return 1000000;

return (-(memory\_value << 17) >> 17) & 0x7FFF;

}

int

ADDC\_C (int cell\_number)

{

int accumulator\_value, memory\_value1, memory\_value2;

sc\_accumulatorGet (&accumulator\_value);

if (sc\_memoryGet (cell\_number, &memory\_value1) == -2)

return 1000000;

if (sc\_memoryGet (accumulator\_value & 0x3F, &memory\_value2) == -2)

return 1000000;

return (((memory\_value1 << 17) + (memory\_value2 << 17)) >> 17);

}

int

SUBC\_C (int cell\_number)

{

int accumulator\_value, memory\_value1, memory\_value2;

sc\_accumulatorGet (&accumulator\_value);

if (sc\_memoryGet (cell\_number, &memory\_value1) == -2)

return 1000000;

if (sc\_memoryGet (accumulator\_value & 0x3F, &memory\_value2) == -2)

return 1000000;

return (((memory\_value1 << 17) - (memory\_value2 << 17)) >> 17);

}

int

alu (int command, int operand)

{

switch (command)

{

case ADD:

return ADD\_C (operand);

case SUB:

return SUB\_C (operand);

case DIVIDE:

return DIVIDE\_C (operand);

case MUL:

return MUL\_C (operand);

case NOT:

return NOT\_C ();

case AND:

return AND\_C (operand);

case OR:

return OR\_C (operand);

case XOR:

return XOR\_C (operand);

case CHL:

return CHL\_C (operand);

case SHR:

return SHR\_C (operand);

case RCL:

return RCL\_C (operand);

case RCR:

return RCR\_C (operand);

case NEG:

return NEG\_C (operand);

case ADDC:

return ADDC\_C (operand);

case SUBC:

return SUBC\_C (operand);

}

return 0;

}

check\_terminal\_size.c

#include "console.h"

#include <myTerm.h>

#include <stdio.h>

int

check\_terminal\_size (void)

{

int rows = 0, cols = 0;

if (mt\_getscreensize (&rows, &cols))

return -1;

if (rows < 27 || cols < 108)

{

printf ("Terminal is too small\n");

printf ("Needs 27x108, but it's %dx%d\n", rows, cols);

return -1;

}

return 0;

}

console.h

#pragma once

#include <myTerm.h>

enum commands

{

NOP = 0x00,

CPUINFO = 0x01,

READ = 0x0A,

WRITE = 0x0B,

LOAD = 0x14,

STORE = 0x15,

ADD = 0x1E,

SUB = 0x1F,

DIVIDE = 0x20,

MUL = 0x21,

JUMP = 0x28,

JNEG = 0x29,

JZ = 0x2A,

HALT = 0x2B,

NOT = 0x33,

AND = 0x34,

OR = 0x35,

XOR = 0x36,

JNS = 0x37,

JC = 0x38,

JNC = 0x39,

JP = 0x3A,

JNP = 0x3B,

CHL = 0x3C,

SHR = 0x3D,

RCL = 0x3E,

RCR = 0x3F,

NEG = 0x40,

ADDC = 0x41,

SUBC = 0x42,

LOGLC = 0x43,

LOGRC = 0x44,

RCCL = 0x45,

RCCR = 0x46,

MOVA = 0x47,

MOVR = 0x48,

MOVCA = 0x49,

MOVCR = 0x4A,

ADDC2 = 0x4B,

SUBC2 = 0x4C

};

extern int cell;

extern int big[36];

void printAccumulator (void);

void printCell (int address, enum colors fg, enum colors bg);

void printCounters (void);

void printDecodedCommand (int value);

void printFlags (void);

int printTerm (int address, int input);

void printInfo (void);

void printBigCell (void);

void printCommand (void);

void CU (void);

int alu (int command, int operand);

void IRC (int signum);

void print\_all\_mem\_cells\_def (void);

void printCache (void);

int get\_font (char \*filename);

int check\_terminal\_size (void);

void draw\_boxes (void);

void print\_all\_mem\_cells\_def (void);

void default\_state (void);

void draw\_interface (void);

void running\_application ();

cu.c

#include "console.h"

#include <myReadKey.h>

#include <mySimpleComputer.h>

#include <myTerm.h>

#include <signal.h>

#include <unistd.h>

void

CPUINFO\_C (void) // 0x01

{

mt\_gotoXY (20, 79);

mt\_print ("Кулик Павел Евгеньевич, ИС241");

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

{

mt\_gotoXY (21 + i, 79);

mt\_print (" ");

}

sleep (2);

printInfo ();

}

void

READ\_C (int cell\_number) // 0x0A

{

sc\_regSet (SC\_THROTTLE, 1);

sc\_setIgnoreCache (1);

printTerm (cell\_number, 1);

sc\_setIgnoreCache (0);

rk\_mytermregime (0, 1, 0, 0, 0);

sc\_regSet (SC\_THROTTLE, 0);

}

int

WRITE\_C (int cell\_number) // 0x0B

{

if (printTerm (cell\_number, 0) == -2)

return -2;

return 0;

}

int

LOAD\_C (int cell\_number) // 0x14

{

int value;

if (sc\_memoryGet (cell\_number, &value) == -2)

return -2;

sc\_accumulatorSet (value);

return 0;

}

int

STORE\_C (int cell\_number) // 0x15

{

int value;

sc\_accumulatorGet (&value);

if (sc\_memorySet (cell\_number, value) == -2)

return -2;

return 0;

}

void

JUMP\_C (int cell\_number) // 0x28

{

sc\_icounterSet (cell\_number);

}

void

JNEG\_C (int cell\_number) // 0x29

{

int value;

sc\_accumulatorGet (&value);

if ((value >> 14) > 0)

{

sc\_icounterSet (cell\_number);

}

}

void

JZ\_C (int cell\_number) // 0x2A

{

int value;

sc\_accumulatorGet (&value);

if ((value & 0x3FFF) == 0)

{

sc\_icounterSet (cell\_number);

}

}

void

HALT\_C (void)

{

sc\_regSet (SC\_THROTTLE, 1);

}

void

JNS\_C (int cell\_number) // 0x37

{

int value;

sc\_accumulatorGet (&value);

if (((value >> 14) == 0) && (value & 0x3FFF) != 0)

{

sc\_icounterSet (cell\_number);

}

}

void

JC\_C (int cell\_number) // 0x38

{

int value = 0;

sc\_regGet (SC\_OVERFLOW, &value);

if (value)

{

sc\_icounterSet (cell\_number);

}

}

void

JNC\_C (int cell\_number) // 0x39

{

int value = 0;

sc\_regGet (SC\_OVERFLOW, &value);

if (!value)

{

sc\_icounterSet (cell\_number);

}

}

void

JP\_C (int cell\_number) // 0x3A

{

int value;

sc\_accumulatorGet (&value);

if ((value & 0x3FFF) % 2 == 0)

{

sc\_icounterSet (cell\_number);

}

}

void

JNP\_C (int cell\_number) // 0x3B

{

int value;

sc\_accumulatorGet (&value);

if ((value & 0x3FFF) % 2 != 0)

{

sc\_icounterSet (cell\_number);

}

}

int

MOVA\_C (int cell\_number) // 0x47

{

int value;

int address;

sc\_accumulatorGet (&address);

if (sc\_memoryGet (cell\_number, &value) == -2)

return -2;

if (sc\_memorySet (address & 0x7F, value) == -2)

return -2;

return 0;

}

int

MOVR\_C (int cell\_number) // 0x48

{

int value;

int address;

sc\_accumulatorGet (&address);

if (sc\_memoryGet (address & 0x7F, &value) == -2)

return -2;

if (sc\_memorySet (cell\_number, value) == -2)

return -2;

return 0;

}

int

MOVCA\_C (int cell\_number) // 0x49

{

int value;

int address\_from;

int address\_to;

sc\_accumulatorGet (&address\_from);

if (sc\_memoryGet (address\_from & 0x7F, &address\_to) == -2)

return -2;

if (sc\_memoryGet (cell\_number, &value) == -2)

return -2;

if (sc\_memorySet (address\_to & 0x7F, value) == -2)

return -2;

return 0;

}

void

CU (void)

{

int command\_number;

int memory\_value;

int sign;

int value;

int operand;

int returned;

sc\_icounterGet (&command\_number);

sc\_setIgnoreCache (0);

returned = sc\_memoryGet (command\_number, &memory\_value);

if (returned == -1)

{

sc\_regSet (SC\_OUT\_OF\_MEMORY, 1);

sc\_regSet (SC\_THROTTLE, 1);

return;

}

if (returned == -2)

return;

if (sc\_commandDecode (memory\_value, &sign, &value, &operand))

{

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

sc\_regSet (SC\_THROTTLE, 1);

return;

}

if (sc\_commandValidate (value) || sign == 1)

{

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

sc\_regSet (SC\_THROTTLE, 1);

return;

}

switch (value)

{

case NOP:

break;

case CPUINFO:

CPUINFO\_C ();

break;

case READ:

READ\_C (operand);

break;

case WRITE:

if (WRITE\_C (operand) == -2)

return;

break;

case LOAD:

if (LOAD\_C (operand) == -2)

return;

break;

case STORE:

if (STORE\_C (operand) == -2)

return;

break;

case JUMP:

JUMP\_C (operand);

break;

case JNEG:

JNEG\_C (operand);

break;

case JZ:

JZ\_C (operand);

break;

case HALT:

HALT\_C ();

break;

case JNS:

JNS\_C (operand);

break;

case JC:

JC\_C (operand);

break;

case JNC:

JNC\_C (operand);

break;

case JP:

JP\_C (operand);

break;

case JNP:

JNP\_C (operand);

break;

case MOVA:

if (MOVA\_C (operand) == -2)

return;

break;

case MOVR:

if (MOVR\_C (operand) == -2)

return;

break;

case MOVCA:

if (MOVCA\_C (operand) == -2)

return;

break;

case NOT:

if (sc\_memorySet (operand, alu (value, operand)) == -2)

return;

break;

default:

returned = alu (value, operand);

if (returned == 1000000)

return;

sc\_accumulatorSet (returned);

}

int new\_command\_number;

sc\_icounterGet (&new\_command\_number);

if (new\_command\_number == command\_number)

if (sc\_icounterSet (command\_number + 1))

sc\_regSet (SC\_THROTTLE, 1);

}

default\_state.c

#include "console.h"

#include <mySimpleComputer.h>

void

default\_state (void)

{

cell = 0;

sc\_accumulatorSet (0);

sc\_icounterSet (0);

draw\_boxes ();

sc\_setIgnoreCache (1);

sc\_memoryInit ();

sc\_setIgnoreCache (0);

sc\_regInit ();

sc\_cacheInit ();

printFlags ();

printAccumulator ();

printCounters ();

printInfo ();

}

draw\_boxes.c

#include <myBigChars.h>

#include <myTerm.h>

void

draw\_boxes (void)

{

mt\_clrscr ();

bc\_box (1, 1, 13, 59, DEFAULT, DEFAULT, " Оперативная память ", RED,

DEFAULT);

bc\_box (16, 1, 1, 59, DEFAULT, DEFAULT, " Редактируемая ячейка (формат) ",

RED, WHITE);

bc\_box (1, 62, 1, 21, DEFAULT, DEFAULT, " Аккумулятор ", RED, DEFAULT);

bc\_box (1, 85, 1, 22, DEFAULT, DEFAULT, " Регистр флагов ", RED, DEFAULT);

bc\_box (4, 62, 1, 21, DEFAULT, DEFAULT, " Счётчик команд ", RED, DEFAULT);

bc\_box (4, 85, 1, 22, DEFAULT, DEFAULT, " Команда ", RED, DEFAULT);

bc\_box (7, 62, 10, 45, DEFAULT, DEFAULT,

" Редактируемая ячейка (увеличено) ", RED, WHITE);

bc\_box (19, 1, 5, 64, DEFAULT, DEFAULT, " Кеш процессора ", GREEN, WHITE);

bc\_box (19, 67, 5, 9, DEFAULT, DEFAULT, " IN--OUT ", GREEN, WHITE);

bc\_box (19, 78, 5, 29, DEFAULT, DEFAULT, " Клавиши ", GREEN, WHITE);

}

draw\_interface.c

#include "console.h"

#include <mySimpleComputer.h>

#include <myTerm.h>

void

draw\_interface (void)

{

int value;

sc\_setIgnoreCache (1);

sc\_memoryGet (cell, &value);

print\_all\_mem\_cells\_def ();

printCell (cell, BLACK, WHITE);

printDecodedCommand (value);

printBigCell ();

printAccumulator ();

printFlags ();

printCache ();

printCounters ();

printCommand ();

sc\_setIgnoreCache (0);

mt\_gotoXY (27, 1);

}

font.c

#include <myBigChars.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

void

print\_bin (int n)

{

int j = 0;

for (int i = 0; i < 32; i++)

{

printf ("%d", (n >> i) & 1);

j++;

if (j == 8)

{

printf ("\n");

j = 0;

}

}

}

void

make\_zero (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 3; i <= 6; i++)

{

bc\_setbigcharpos (big, i, 4, 0);

bc\_setbigcharpos (big, i, 5, 0);

}

}

void

make\_one (int \*big)

{

big[0] = 0;

big[1] = 0;

for (int i = 1; i <= 8; i++)

{

bc\_setbigcharpos (big, i, 5, 1);

bc\_setbigcharpos (big, i, 6, 1);

}

bc\_setbigcharpos (big, 2, 4, 1);

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

bc\_setbigcharpos (big, 3, 4, 1);

bc\_setbigcharpos (big, 7, 4, 1);

bc\_setbigcharpos (big, 8, 3, 1);

bc\_setbigcharpos (big, 8, 4, 1);

bc\_setbigcharpos (big, 8, 7, 1);

}

void

make\_two (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 2; i <= 5; i++)

{

bc\_setbigcharpos (big, 3, i, 0);

bc\_setbigcharpos (big, 6, i + 2, 0);

}

}

void

make\_three (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 3; i <= 5; i++)

{

bc\_setbigcharpos (big, 3, i, 0);

bc\_setbigcharpos (big, 6, i, 0);

bc\_setbigcharpos (big, 4, i - 2, 0);

bc\_setbigcharpos (big, 5, i - 2, 0);

}

}

void

make\_four (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 1; i <= 4; i++)

{

bc\_setbigcharpos (big, i, 4, 0);

bc\_setbigcharpos (big, i, 5, 0);

bc\_setbigcharpos (big, 7, i + 1, 0);

bc\_setbigcharpos (big, 8, i + 1, 0);

}

}

void

make\_five (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 2; i <= 5; i++)

{

bc\_setbigcharpos (big, 3, i + 2, 0);

bc\_setbigcharpos (big, 6, i, 0);

}

}

void

make\_six (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 4; i <= 7; i++)

{

bc\_setbigcharpos (big, 3, i, 0);

}

bc\_setbigcharpos (big, 6, 4, 0);

bc\_setbigcharpos (big, 6, 5, 0);

}

void

make\_seven (int \*big)

{

big[0] = 0;

big[1] = 0;

for (int i = 2; i <= 7; i++)

{

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

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

}

for (int i = 3; i <= 8; i++)

{

bc\_setbigcharpos (big, i, 6, 1);

bc\_setbigcharpos (big, i, 7, 1);

}

}

void

make\_eight (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

bc\_setbigcharpos (big, 3, 4, 0);

bc\_setbigcharpos (big, 3, 5, 0);

bc\_setbigcharpos (big, 6, 4, 0);

bc\_setbigcharpos (big, 6, 5, 0);

}

void

make\_nine (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 2; i <= 5; i++)

{

bc\_setbigcharpos (big, 6, i, 0);

}

bc\_setbigcharpos (big, 3, 4, 0);

bc\_setbigcharpos (big, 3, 5, 0);

}

void

make\_A (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 4; i <= 5; i++)

{

bc\_setbigcharpos (big, 3, i, 0);

bc\_setbigcharpos (big, 4, i, 0);

bc\_setbigcharpos (big, 7, i, 0);

bc\_setbigcharpos (big, 8, i, 0);

}

}

void

make\_B (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 4; i <= 5; i++)

{

bc\_setbigcharpos (big, 3, i, 0);

bc\_setbigcharpos (big, 6, i, 0);

bc\_setbigcharpos (big, i, 7, 0);

}

}

void

make\_C (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 4; i <= 7; i++)

{

bc\_setbigcharpos (big, 3, i, 0);

bc\_setbigcharpos (big, 4, i, 0);

bc\_setbigcharpos (big, 5, i, 0);

bc\_setbigcharpos (big, 6, i, 0);

}

}

void

make\_D (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 3; i <= 6; i++)

{

bc\_setbigcharpos (big, i, 4, 0);

}

bc\_setbigcharpos (big, 4, 5, 0);

bc\_setbigcharpos (big, 5, 5, 0);

bc\_setbigcharpos (big, 1, 7, 0);

bc\_setbigcharpos (big, 8, 7, 0);

}

void

make\_E (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 4; i <= 7; i++)

{

bc\_setbigcharpos (big, 3, i, 0);

bc\_setbigcharpos (big, 6, i, 0);

}

}

void

make\_F (int \*big)

{

big[0] = 0xffffffff;

big[1] = 0xffffffff;

for (int i = 1; i <= 8; i++)

{

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

bc\_setbigcharpos (big, i, 8, 0);

}

for (int i = 4; i <= 7; i++)

{

bc\_setbigcharpos (big, 3, i, 0);

bc\_setbigcharpos (big, 6, i, 0);

bc\_setbigcharpos (big, 7, i, 0);

bc\_setbigcharpos (big, 8, i, 0);

}

}

void

make\_plus (int \*big)

{

big[0] = 0;

big[1] = 0;

for (int i = 1; i <= 8; i++)

{

bc\_setbigcharpos (big, i, 4, 1);

bc\_setbigcharpos (big, i, 5, 1);

}

for (int i = 2; i <= 7; i++)

{

bc\_setbigcharpos (big, 4, i, 1);

bc\_setbigcharpos (big, 5, i, 1);

}

}

void

make\_minus (int \*big)

{

big[0] = 0;

big[1] = 0;

for (int i = 2; i <= 7; i++)

{

bc\_setbigcharpos (big, 4, i, 1);

bc\_setbigcharpos (big, 5, i, 1);

}

}

int

main ()

{

int \*big = malloc (36 \* sizeof (int));

if (!big)

return -1;

int index = 0;

make\_zero (big + index);

index += 2;

make\_one (big + index);

index += 2;

make\_two (big + index);

index += 2;

make\_three (big + index);

index += 2;

make\_four (big + index);

index += 2;

make\_five (big + index);

index += 2;

make\_six (big + index);

index += 2;

make\_seven (big + index);

index += 2;

make\_eight (big + index);

index += 2;

make\_nine (big + index);

index += 2;

make\_A (big + index);

index += 2;

make\_B (big + index);

index += 2;

make\_C (big + index);

index += 2;

make\_D (big + index);

index += 2;

make\_E (big + index);

index += 2;

make\_F (big + index);

index += 2;

make\_plus (big + index);

index += 2;

make\_minus (big + index);

int fd = open ("font.bin", O\_CREAT | O\_WRONLY | O\_TRUNC, 0644);

if (fd == -1)

{

printf ("1\n");

return -1;

}

if (bc\_bigcharwrite (fd, big, 18))

{

printf ("2\n");

return -1;

}

close (fd);

free (big);

}

get\_font.c

#include "console.h"

#include <myBigChars.h>

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

int

get\_font (char \*filename)

{

int fd;

fd = open (filename, O\_RDONLY);

if (fd == -1)

{

printf ("Can't open font :(\n");

return -1;

}

int count;

bc\_bigcharread (fd, big, 18, &count);

if (count != 18)

{

printf ("Something wrong with bc\_bigcharread\n");

close (fd);

return -1;

}

close (fd);

return 0;

}

IRC.c

#include <signal.h>

#include <unistd.h>

#include "console.h"

#include <mySimpleComputer.h>

void

IRC (int signum)

{

if (signum == SIGUSR1)

{

sc\_memoryInit ();

sc\_regInit ();

sc\_accumulatorInit ();

sc\_icounterSet (0);

sc\_regSet (SC\_THROTTLE, 1);

}

if (signum == SIGALRM)

{

int flag;

sc\_regGet (SC\_THROTTLE, &flag);

unsigned char tcounter;

sc\_tcounterGet (&tcounter);

if (tcounter)

{

sc\_tcounterSet (--tcounter);

sc\_setIgnoreCache (1);

print\_all\_mem\_cells\_def ();

printBigCell ();

printAccumulator ();

printFlags ();

printCounters ();

printCommand ();

printCache ();

sc\_setIgnoreCache (0);

mt\_gotoXY (27, 1);

if (!tcounter)

{

sc\_regSet (SC\_THROTTLE, 0);

flag = 0;

}

else

{

sc\_regSet (SC\_THROTTLE, 1);

return;

}

}

if (flag)

return;

CU ();

sc\_setIgnoreCache (1);

print\_all\_mem\_cells\_def ();

printBigCell ();

printAccumulator ();

printFlags ();

printCounters ();

printCommand ();

printCache ();

sc\_setIgnoreCache (0);

mt\_gotoXY (27, 1);

}

}

main.c

#include "console.h"

#include <mySimpleComputer.h>

#include <signal.h>

#include <stdio.h>

#include <sys/time.h>

#include <unistd.h>

int cell = 0;

int big[36];

int

main (int argc, char \*argv[])

{

if (argc > 1)

{

if (get\_font (argv[1]))

return 1;

}

else

{

if (get\_font ("./console/font.bin"))

return 1;

}

if (!isatty (STDIN\_FILENO))

{

printf ("Can't reach terminal\n");

return 1;

}

if (check\_terminal\_size ())

return 1;

sc\_setIgnoreCache (1);

default\_state ();

draw\_interface ();

sc\_setIgnoreCache (0);

signal (SIGALRM, IRC);

signal (SIGUSR1, IRC);

struct itimerval nval, oval;

nval.it\_interval.tv\_sec = 0;

nval.it\_interval.tv\_usec = 500000;

nval.it\_value.tv\_sec = 1;

nval.it\_value.tv\_usec = 0;

setitimer (ITIMER\_REAL, &nval, &oval);

running\_application ();

mt\_print ("\n");

mt\_gotoXY (50, 1);

}

makefile

APP\_NAME = app

SRC\_EXT = c

APP\_SOURCES = $(filter-out font.c, $(wildcard \*.$(SRC\_EXT)))

APP\_OBJECTS := $(patsubst %.$(SRC\_EXT),%.o,$(APP\_SOURCES))

DEPS = $(APP\_OBJECTS:.o=.d)

.PHONY: all

all: $(APP\_NAME) font.bin

-include $(DEPS)

$(APP\_NAME): $(APP\_OBJECTS) $(LIBS)

$(CC) $(CFLAGS) $(CPPFLAGS) $^ -o $@ $(LFLAGS)

/%.o: /%.$(SRC\_EXT)

$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@ $(LFLAGS)

font.bin: font

./font

rm font

font: font.c $(LIBS)

$(CC) $(CFLAGS) $(CPPFLAGS) $^ -o $@ $(LFLAGS)

.PHONY: clean

clean:

rm -rf $(APP\_OBJECTS) $(DEPS) $(APP\_NAME) font.d font.bin

print\_all\_mem\_cells\_def.c

#include "console.h"

void

print\_all\_mem\_cells\_def (void)

{

for (int i = 0; i < 128; i++)

printCell (i, DEFAULT, DEFAULT);

}

printAccumulator.c

#include <mySimpleComputer.h>

#include <myTerm.h>

#include <stdio.h>

void

printAccumulator (void)

{

int value;

if (sc\_accumulatorGet (&value))

{

mt\_print ("Error!\n");

return;

}

mt\_gotoXY (2, 64);

mt\_print ("sc: ");

if (value >> 14)

{

if (value & 0x3FFF)

{

mt\_print ("-");

value = (~value & 0x3FFF) + 1;

}

else

{

mt\_print ("-7F80");

mt\_print (" hex: %04X", value);

return;

}

}

else

mt\_print ("+");

mt\_print ("%02X", value >> 7 & 0b1111111);

mt\_print ("%02X", value & 0b1111111);

sc\_accumulatorGet (&value);

mt\_print (" hex: %04X", value);

}

printBigCell.c

#include "console.h"

#include <myBigChars.h>

#include <mySimpleComputer.h>

void

printBigCell (void)

{

int value;

sc\_memoryGet (cell, &value);

if (value >> 14)

{

bc\_printbigchar (&big[34], 9, 64, DEFAULT, DEFAULT);

value = ((~value & 0x3FFF) + 1) | 0x4000;

}

else

bc\_printbigchar (&big[32], 9, 64, DEFAULT, DEFAULT);

if ((value >> 14) && ((value & 0x3FFF) == 0))

{

bc\_printbigchar (&big[14], 9, 72, DEFAULT, DEFAULT);

bc\_printbigchar (&big[30], 9, 80, DEFAULT, DEFAULT);

bc\_printbigchar (&big[16], 9, 88, DEFAULT, DEFAULT);

bc\_printbigchar (&big[0], 9, 96, DEFAULT, DEFAULT);

}

else

{

bc\_printbigchar (&big[((value >> 11) & 0b111) \* 2], 9, 72, DEFAULT,

DEFAULT);

bc\_printbigchar (&big[((value >> 7) & 0b1111) \* 2], 9, 80, DEFAULT,

DEFAULT);

bc\_printbigchar (&big[((value >> 4) & 0b111) \* 2], 9, 88, DEFAULT,

DEFAULT);

bc\_printbigchar (&big[(value & 0b1111) \* 2], 9, 96, DEFAULT, DEFAULT);

}

mt\_gotoXY (17, 64);

mt\_setfgcolor (BLUE);

mt\_print ("Номер редактируемой ячейки: %03d", cell);

mt\_setfgcolor (DEFAULT);

}

printCache.c

#include "mySimpleComputer.h"

#include "myTerm.h"

#include <stdio.h>

void

printCache (void)

{

int cacheline[10];

int line\_size;

int address;

for (int line = 0; line < 5; line++)

{

address = sc\_cachelineGet (line, cacheline);

if (address != -1)

{

line\_size = address == 120 ? 8 : 10;

mt\_gotoXY (20 + line, 2);

mt\_print ("%d: "

" ",

address);

for (int i = 0; i < line\_size; i++)

{

mt\_gotoXY (20 + line, i \* 6 + 7);

if (cacheline[i] >> 14)

{

mt\_print ("-");

cacheline[i] = (~cacheline[i] & 0x3FFF) + 1;

}

else

mt\_print ("+");

mt\_print ("%02X", cacheline[i] >> 7 & 0b1111111);

mt\_print ("%02X", cacheline[i] & 0b1111111);

}

}

}

}

printCell.c

#include "mySimpleComputer.h"

#include "myTerm.h"

#include <stdio.h>

void

printCell (int address, enum colors fg, enum colors bg)

{

int value;

if (sc\_memoryGet (address, &value) == -1)

return;

mt\_setbgcolor (bg);

mt\_setfgcolor (fg);

int row = 1, col = 0;

col = address % 10;

int tmp\_address = address;

while (tmp\_address > 9)

{

tmp\_address -= 10;

row++;

}

mt\_gotoXY (row + 1, col \* 6 + 2);

if (value >> 14)

{

mt\_print ("-");

value = (~value & 0x3FFF) + 1;

}

else

mt\_print ("+");

mt\_print ("%02X", value >> 7 & 0b1111111);

mt\_print ("%02X", value & 0b1111111);

if (value >> 14 && (value & 0x3FFF) == 0)

{

mt\_gotoXY (row + 1, col \* 6 + 2);

mt\_print ("-7F80");

}

mt\_setdefaultcolor ();

}

printCommand.c

#include <mySimpleComputer.h>

#include <myTerm.h>

void

printCommand (void)

{

int value, cell\_number;

sc\_icounterGet (&cell\_number);

mt\_gotoXY (5, 90);

if (cell\_number < 0 || cell\_number > 127)

{

mt\_print ("! FF : FF");

return;

}

sc\_memoryGet (cell\_number, &value);

if (value >> 14 > 0)

mt\_print ("- ");

else

mt\_print ("+ ");

mt\_print ("%02X : %02X", (value >> 7) & 0x7F, value & 0x7F);

}

printCounters.c

#include <mySimpleComputer.h>

#include <myTerm.h>

#include <stdio.h>

void

printCounters (void)

{

int IC;

unsigned char TC;

if (sc\_icounterGet (&IC))

{

mt\_print ("Error!\n");

return;

}

if (sc\_tcounterGet (&TC))

{

mt\_print ("Error!\n");

return;

}

mt\_gotoXY (5, 63);

mt\_print ("T: %02d IC: ", TC);

mt\_gotoXY (5, 77);

if (IC >> 14)

{

if (IC & 0x3FFF)

{

mt\_print ("-");

IC = (~IC & 0x3FFF) + 1;

}

else

{

mt\_print ("-7F80");

return;

}

}

else

mt\_print ("+");

mt\_print ("%02X", IC >> 7 & 0b1111111);

mt\_print ("%02X", IC & 0b1111111);

}

printDecodedCommand.c

#include <mySimpleComputer.h>

#include <myTerm.h>

#include <stdio.h>

void

printDecodedCommand (int value)

{

mt\_gotoXY (17, 2);

mt\_print ("dec: %05d | ", value);

mt\_print ("oct: %05o | ", value);

mt\_print ("hex: %04X ", value);

mt\_print ("bin: ");

for (int i = 14; i >= 0; i--)

mt\_print ("%d", (value >> i) & 1);

mt\_print ("\n");

}

printFlags.c

#include <mySimpleComputer.h>

#include <myTerm.h>

#include <stdio.h>

void

printFlags (void)

{

int P, ZERO, M, T, E;

if (sc\_regGet (SC\_OVERFLOW, &P))

{

mt\_print ("Error!\n");

return;

}

if (sc\_regGet (SC\_DIVIDING\_BY\_ZERO, &ZERO))

{

mt\_print ("Error!\n");

return;

}

if (sc\_regGet (SC\_OUT\_OF\_MEMORY, &M))

{

mt\_print ("Error!\n");

return;

}

if (sc\_regGet (SC\_THROTTLE, &T))

{

mt\_print ("Error!\n");

return;

}

if (sc\_regGet (SC\_INVALID\_COMMAND, &E))

{

mt\_print ("Error!\n");

return;

}

mt\_gotoXY (2, 90);

if (P == 0)

mt\_print ("\_ ");

else

mt\_print ("P ");

mt\_gotoXY (2, 93);

if (ZERO == 0)

mt\_print ("\_ ");

else

mt\_print ("0 ");

mt\_gotoXY (2, 96);

if (M == 0)

mt\_print ("\_ ");

else

mt\_print ("M ");

mt\_gotoXY (2, 99);

if (T == 0)

mt\_print ("\_ ");

else

mt\_print ("T ");

mt\_gotoXY (2, 102);

if (E == 0)

mt\_print ("\_ ");

else

mt\_print ("E ");

}

printInfo.c

#include <myTerm.h>

void

printInfo ()

{

mt\_gotoXY (20, 79);

mt\_print ("l - load s - save i - reset");

mt\_gotoXY (21, 79);

mt\_print ("r - run t - step");

mt\_gotoXY (22, 79);

mt\_print ("ESC - выход");

mt\_gotoXY (23, 79);

mt\_print ("F5 - accumulator");

mt\_gotoXY (24, 79);

mt\_print ("F6 - instruction counter");

}

printTerm.c

#include <myReadKey.h>

#include <mySimpleComputer.h>

#include <myTerm.h>

#include <stdio.h>

#include <string.h>

char terms[5][10];

int

printTerm (int address, int input)

{

for (int i = 4; i > 0; i--)

strcpy (terms[i], terms[i - 1]);

int value = 0;

int returned;

if (input == 0)

{

returned = sc\_memoryGet (address, &value);

if (returned)

{

return returned;

}

if (value >> 14)

{

value = (~value & 0x3FFF) + 1;

snprintf (terms[0], 10, "%02X> -%02X%02X", address,

(value >> 7) & 0x7F, value & 0x7F);

}

else

snprintf (terms[0], 10, "%02X> +%02X%02X", address,

(value >> 7) & 0x7F, value & 0x7F);

}

else

{

int row = 20;

for (int i = 4; i >= 0; i--)

{

mt\_gotoXY (row++, 68);

mt\_print (terms[i]);

}

mt\_gotoXY (24, 68);

mt\_print ("%02X< ", address);

mt\_gotoXY (24, 72);

rk\_readvalue (&value, 1000);

sc\_setIgnoreCache (1);

sc\_memorySet (address, value);

if (value >> 14)

{

if (value & 0x3FFF)

{

value = (~value & 0x3FFF) + 1;

snprintf (terms[0], 10, "%02X< -%02X%02X", address,

(value >> 7) & 0x7F, value & 0x7F);

}

else

{

snprintf (terms[0], 10, "%02X< -7F80", address);

}

}

else

{

snprintf (terms[0], 10, "%02X< +%02X%02X", address,

(value >> 7) & 0x7F, value & 0x7F);

}

}

int row = 20;

for (int i = 4; i >= 0; i--)

{

mt\_gotoXY (row++, 68);

mt\_print (terms[i]);

}

return 0;

}

running\_application.c

#include "console.h"

#include <myReadKey.h>

#include <mySimpleComputer.h>

#include <myTerm.h>

void

running\_application ()

{

rk\_mytermsave ();

int value;

int running = 1;

enum keys key;

int throttle = 1;

unsigned char tc = 0;

draw\_interface ();

while (running)

{

rk\_mytermregime (0, 1, 0, 0, 0);

sc\_regGet (SC\_THROTTLE, &throttle);

sc\_tcounterGet (&tc);

if (throttle && tc == 0)

{

if (key)

draw\_interface ();

key = 0;

rk\_readkey (&key);

}

if (key == key\_ESC)

running = 0;

if (key == key\_RIGHT)

{

cell++;

if (cell % 10 == 0)

cell -= 10;

if (cell == 128)

cell = 120;

sc\_setIgnoreCache (1);

print\_all\_mem\_cells\_def ();

printCell (cell, BLACK, WHITE);

sc\_setIgnoreCache (0);

}

if (key == key\_DOWN)

{

cell += 10;

if (cell > 127)

{

cell -= 130;

if (cell < 0)

cell += 10;

}

sc\_setIgnoreCache (1);

print\_all\_mem\_cells\_def ();

printCell (cell, BLACK, WHITE);

sc\_setIgnoreCache (0);

}

if (key == key\_LEFT)

{

cell--;

if ((cell + 1) % 10 == 0)

{

cell += 10;

if (cell == 129)

cell -= 2;

}

sc\_setIgnoreCache (1);

print\_all\_mem\_cells\_def ();

printCell (cell, BLACK, WHITE);

sc\_setIgnoreCache (0);

}

if (key == key\_UP)

{

cell -= 10;

if (cell < 0)

{

cell += 130;

if (cell > 127)

cell -= 10;

}

sc\_setIgnoreCache (1);

print\_all\_mem\_cells\_def ();

printCell (cell, BLACK, WHITE);

sc\_setIgnoreCache (0);

}

if (key == key\_ENTER)

{

mt\_gotoXY (2 + cell / 10, 2 + (cell % 10) \* 6);

mt\_print (" ");

mt\_gotoXY (2 + cell / 10, 2 + (cell % 10) \* 6);

sc\_setIgnoreCache (1);

if (!rk\_readvalue (&value, 100))

sc\_memorySet (cell, value);

print\_all\_mem\_cells\_def ();

printCell (cell, BLACK, WHITE);

sc\_setIgnoreCache (0);

}

if (key == key\_F5)

{

mt\_gotoXY (2, 68);

mt\_print (" ");

mt\_gotoXY (2, 68);

if (!rk\_readvalue (&value, 100))

sc\_accumulatorSet (value);

printAccumulator ();

}

if (key == key\_F6)

{

mt\_gotoXY (5, 77);

mt\_print (" ");

mt\_gotoXY (5, 77);

if (!rk\_readvalue (&value, 100))

sc\_icounterSet (value);

printCounters ();

}

if (key == key\_L)

{

rk\_mytermrestore ();

mt\_gotoXY (26, 1);

mt\_print ("Введите имя файла для загрузки: ");

char filename[128];

ssize\_t size;

size = read (STDIN\_FILENO, filename, 127);

filename[size - 1] = '\0';

mt\_gotoXY (26, 1);

mt\_print ("%\*c", 108, ' ');

mt\_gotoXY (26, 1);

if (sc\_memoryLoad (filename))

{

mt\_print ("Не удаётся загрузить память из файла \"%s\"",

filename);

sleep (2);

}

else

{

mt\_print ("Память из файла \"%s\" успешно загружена", filename);

sleep (2);

}

mt\_gotoXY (26, 1);

mt\_print ("%\*c", 108, ' ');

rk\_mytermregime (0, 0, 1, 0, 0);

}

if (key == key\_S)

{

rk\_mytermrestore ();

mt\_gotoXY (26, 1);

mt\_print ("Введите имя файла для сохранения: ");

char filename[128];

ssize\_t size;

size = read (STDIN\_FILENO, filename, 127);

filename[size - 1] = '\0';

mt\_gotoXY (26, 1);

mt\_print ("%\*c", 108, ' ');

mt\_gotoXY (26, 1);

if (sc\_memorySave (filename))

{

mt\_print ("Не удаётся сохранить память в файл \"%s\"", filename);

sleep (2);

}

else

{

mt\_print ("Память успешно сохранена в файл \"%s\"", filename);

sleep (2);

}

mt\_gotoXY (26, 1);

mt\_print ("%\*c", 108, ' ');

rk\_mytermregime (0, 0, 1, 0, 0);

}

if (key == key\_I)

{

default\_state ();

}

if (key == key\_R)

{

sc\_regSet (SC\_THROTTLE, 0);

key = 0;

}

if (key == key\_T)

{

CU ();

}

}

rk\_mytermrestore ();

}

## include

myBigChars.h

#pragma once

#include <myTerm.h>

int bc\_strlen (char \*str);

int bc\_printA (char \*str);

int bc\_box (int x1, int y1, int x2, int y2, enum colors box\_fg,

enum colors box\_bg, char \*header, enum colors header\_fg,

enum colors header\_bg);

int bc\_setbigcharpos (int \*big, int x, int y, int value);

int bc\_getbigcharpos (int \*big, int x, int y, int \*value);

int bc\_printbigchar (int \*big, int x, int y, enum colors, enum colors);

int bc\_bigcharwrite (int fd, int \*big, int count);

int bc\_bigcharread (int fd, int \*big, int need\_count, int \*count);

myReadKey.h

#pragma once

#include <string.h>

#include <termios.h>

#include <unistd.h>

extern struct termios current, backup;

enum keys

{

key\_UNDEFINED,

key\_0,

key\_1,

key\_2,

key\_3,

key\_4,

key\_5,

key\_6,

key\_7,

key\_8,

key\_9,

key\_A,

key\_B,

key\_C,

key\_D,

key\_E,

key\_F,

key\_plus,

key\_minus,

key\_L,

key\_S,

key\_I,

key\_R,

key\_T,

key\_F5,

key\_F6,

key\_ESC,

key\_ENTER,

key\_UP,

key\_RIGHT,

key\_DOWN,

key\_LEFT

};

int rk\_readkey (enum keys \*key);

int rk\_mytermsave (void);

int rk\_mytermregime (int regime, int vtime, int vmin, int echo, int sigint);

int rk\_mytermrestore (void);

int rk\_readvalue (int \*value, int timeout);

mySimpleComputer.h

#define GET\_BIT\_VALUE(REGISTER, NUMBER) ((REGISTER >> (NUMBER - 1)) & 1)

#define SET\_BIT\_ZERO(REGISTER, NUMBER) \

REGISTER = (REGISTER & (~(1 << (NUMBER - 1))))

#define SET\_BIT(REGISTER, NUMBER) REGISTER = (REGISTER | (1 << (NUMBER - 1)))

#define SC\_OVERFLOW 16

#define SC\_DIVIDING\_BY\_ZERO 8

#define SC\_OUT\_OF\_MEMORY 4

#define SC\_INVALID\_COMMAND 2

#define SC\_THROTTLE 1

int sc\_accumulatorGet (int \*value);

int sc\_accumulatorInit (void);

int sc\_accumulatorSet (int value);

int sc\_commandDecode (int value, int \*sign, int \*command, int \*operand);

int sc\_commandEncode (int sign, int command, int operand, int \*value);

int sc\_commandValidate (int command);

int sc\_icounterGet (int \*value);

int sc\_icounterInit (void);

int sc\_icounterSet (int value);

int sc\_memoryGet (int address, int \*value);

int sc\_memoryInit (void);

int sc\_memoryLoad (char \*filename);

int sc\_memorySave (char \*filename);

int sc\_memorySet (int adress, int value);

int sc\_regGet (int sc\_register, int \*value);

int sc\_regInit (void);

int sc\_regSet (int sc\_register, int value);

int sc\_cacheGet (int address, int \*value);

int sc\_cacheSet (int address, int value);

int sc\_cacheInit (void);

int sc\_tcounterSet (unsigned char value);

int sc\_tcounterGet (unsigned char \*value);

int sc\_tcounterInit (void);

int sc\_cachelineGet (int line\_number, int \*cacheline);

int sc\_setIgnoreCache (int value);

myTerm.h

#pragma once

enum colors

{

BLACK,

RED,

GREEN,

YELLOW,

BLUE,

PURPLE,

CYAN,

WHITE,

DEFAULT

};

int mt\_clrscr (void);

int mt\_delline (void);

int mt\_getscreensize (int \*rows, int \*cols);

int mt\_gotoXY (int x, int y);

int mt\_setbgcolor (enum colors color);

int mt\_setcursorvisible (int value);

int mt\_setdefaultcolor (void);

int mt\_setfgcolor (enum colors color);

int mt\_print (char \*format, ...);

## myBigChars

bc\_bigcharread.c

#include <unistd.h>

int

bc\_bigcharread (int fd, int \*big, int need\_count, int \*count)

{

if (!big)

return -1;

if (!count)

return -1;

ssize\_t size = need\_count \* 2 \* sizeof (int);

\*count = read (fd, big, size) / 2 / sizeof (int);

if (\*count != need\_count)

return -1;

return 0;

}

bc\_bigcharwrite.c

#include <unistd.h>

int

bc\_bigcharwrite (int fd, int \*big, int count)

{

if (!big)

return -1;

ssize\_t size = count \* sizeof (int) \* 2;

if (write (fd, big, size) != size)

return -1;

return 0;

}

bc\_box.c

#include <myBigChars.h>

#include <myTerm.h>

int

bc\_box (int x1, int y1, int x2, int y2, enum colors box\_fg, enum colors box\_bg,

char \*header, enum colors header\_fg, enum colors header\_bg)

{

if (!header)

return -1;

mt\_setbgcolor (box\_bg);

mt\_setfgcolor (box\_fg);

mt\_gotoXY (x1, y1);

bc\_printA ("l");

mt\_gotoXY (x1, ++y1);

for (int i = 0; i < y2; i++)

{

bc\_printA ("q");

mt\_gotoXY (x1, ++y1);

}

bc\_printA ("k");

mt\_gotoXY (++x1, y1);

for (int i = 0; i < x2; i++)

{

bc\_printA ("x");

mt\_gotoXY (++x1, y1);

}

bc\_printA ("j");

mt\_gotoXY (x1, --y1);

for (int i = 0; i < y2; i++)

{

bc\_printA ("q");

mt\_gotoXY (x1, --y1);

}

bc\_printA ("m");

mt\_gotoXY (--x1, y1);

for (int i = 0; i < x2; i++)

{

bc\_printA ("x");

mt\_gotoXY (--x1, y1);

}

mt\_setbgcolor (header\_bg);

mt\_setfgcolor (header\_fg);

mt\_gotoXY (x1, y1 + (y2 - bc\_strlen (header)) / 2 + 1);

mt\_print ("%s", header);

mt\_setdefaultcolor ();

return 0;

}

bc\_getbigcharpos.c

#include <mySimpleComputer.h>

int

bc\_getbigcharpos (int \*big, int x, int y, int \*value)

{

int bit\_number = x < 5 ? y + (x - 1) \* 8 : y + (x - 5) \* 8;

if (x < 5)

\*value = GET\_BIT\_VALUE (big[0], bit\_number);

else

\*value = GET\_BIT\_VALUE (big[1], bit\_number);

return 0;

}

bc\_printA.c

#include <myTerm.h>

int

bc\_printA (char \*str)

{

if (!str)

return -1;

mt\_print ("\e(0%s\e(B", str);

return 0;

}

bc\_printbigchar.c

#include <myBigChars.h>

#include <myTerm.h>

int

bc\_printbigchar (int \*big, int x, int y, enum colors fg, enum colors bg)

{

if (!big)

return -1;

mt\_setfgcolor (fg);

mt\_setbgcolor (bg);

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

int value;

mt\_gotoXY (x + i, y + j);

bc\_getbigcharpos (big, i + 1, j + 1, &value);

if (value)

bc\_printA ("a");

else

mt\_print (" ");

}

}

mt\_setdefaultcolor ();

return 0;

}

bc\_setbigcharpos.c

#include <mySimpleComputer.h>

int

bc\_setbigcharpos (int \*big, int x, int y, int value)

{

int bit\_number = x < 5 ? y + (x - 1) \* 8 : y + (x - 5) \* 8;

if (x < 5)

{

if (value)

SET\_BIT (big[0], bit\_number);

else

SET\_BIT\_ZERO (big[0], bit\_number);

}

else

{

if (value)

SET\_BIT (big[1], bit\_number);

else

SET\_BIT\_ZERO (big[1], bit\_number);

}

return 0;

}

bc\_strlen.c

int

bc\_strlen (char \*str)

{

int length = 0;

while (\*str != '\0')

{

length++;

if ((\*str & 128) > 0)

str += 2;

else

str++;

}

return length;

}

makefile

LIB\_PATH = libmyBigChars.a

SRC\_EXT = c

APP\_SOURCES = $(wildcard \*.$(SRC\_EXT))

LIB\_OBJECTS := $(patsubst %.$(SRC\_EXT),%.o,$(APP\_SOURCES))

DEPS = $(LIB\_OBJECTS:.o=.d)

.PHONY: all

all: $(LIB\_PATH)

-include $(DEPS)

$(LIB\_PATH): $(LIB\_OBJECTS)

ar rcs $@ $^

/%.o: /%.$(SRC\_EXT)

$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean

clean:

rm -rf $(LIB\_OBJECTS)

rm -rf $(DEPS)

rm -rf $(LIB\_PATH)

## myReadKey

makefile

LIB\_PATH = libmyReadKey.a

SRC\_EXT = c

APP\_SOURCES = $(wildcard \*.$(SRC\_EXT))

LIB\_OBJECTS := $(patsubst %.$(SRC\_EXT),%.o,$(APP\_SOURCES))

DEPS = $(LIB\_OBJECTS:.o=.d)

.PHONY: all

all: $(LIB\_PATH)

-include $(DEPS)

$(LIB\_PATH): $(LIB\_OBJECTS)

ar rcs $@ $^

/%.o: /%.$(SRC\_EXT)

$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean

clean:

rm -rf $(LIB\_OBJECTS)

rm -rf $(DEPS)

rm -rf $(LIB\_PATH)

rk\_mytermregime.c

#include <myReadKey.h>

int

rk\_mytermregime (int regime, int vtime, int vmin, int echo, int sigint)

{

struct termios term;

if (tcgetattr (STDIN\_FILENO, &term))

return -1;

if (regime)

term.c\_lflag |= ICANON;

else

term.c\_lflag &= ~ICANON;

if (echo)

term.c\_lflag |= ECHO;

else

term.c\_lflag &= ~ECHO;

if (sigint)

term.c\_lflag |= ISIG;

else

term.c\_lflag &= ~ISIG;

term.c\_cc[VTIME] = vtime;

term.c\_cc[VMIN] = vmin;

return tcsetattr (STDIN\_FILENO, TCSANOW, &term);

}

rk\_mytermrestore.c

#include <myReadKey.h>

int

rk\_mytermrestore (void)

{

return tcsetattr (STDIN\_FILENO, TCSANOW, &backup);

}

rk\_mytermsave.c

#include <myReadKey.h>

struct termios backup;

int

rk\_mytermsave (void)

{

return tcgetattr (STDOUT\_FILENO, &backup);

}

rk\_readkey.c

#include <myReadKey.h>

int

rk\_readkey (enum keys \*key)

{

char buf[16];

ssize\_t n;

n = read (STDIN\_FILENO, buf, 15);

if (n == 0)

return -1;

buf[n] = '\0';

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

\*key = key\_L;

else if (strcmp (buf, "s") == 0)

\*key = key\_S;

else if (strcmp (buf, "i") == 0)

\*key = key\_I;

else if (strcmp (buf, "r") == 0)

\*key = key\_R;

else if (strcmp (buf, "t") == 0)

\*key = key\_T;

else if (strcmp (buf, "\e[15~") == 0)

\*key = key\_F5;

else if (strcmp (buf, "\e[17~") == 0)

\*key = key\_F6;

else if (strcmp (buf, "\e") == 0)

\*key = key\_ESC;

else if (strcmp (buf, "\n") == 0)

\*key = key\_ENTER;

else if (strcmp (buf, "\e[A") == 0)

\*key = key\_UP;

else if (strcmp (buf, "\e[C") == 0)

\*key = key\_RIGHT;

else if (strcmp (buf, "\e[B") == 0)

\*key = key\_DOWN;

else if (strcmp (buf, "\e[D") == 0)

\*key = key\_LEFT;

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

\*key = key\_0;

else if (strcmp (buf, "1") == 0)

\*key = key\_1;

else if (strcmp (buf, "2") == 0)

\*key = key\_2;

else if (strcmp (buf, "3") == 0)

\*key = key\_3;

else if (strcmp (buf, "4") == 0)

\*key = key\_4;

else if (strcmp (buf, "5") == 0)

\*key = key\_5;

else if (strcmp (buf, "6") == 0)

\*key = key\_6;

else if (strcmp (buf, "7") == 0)

\*key = key\_7;

else if (strcmp (buf, "8") == 0)

\*key = key\_8;

else if (strcmp (buf, "9") == 0)

\*key = key\_9;

else if (strcmp (buf, "a") == 0)

\*key = key\_A;

else if (strcmp (buf, "b") == 0)

\*key = key\_B;

else if (strcmp (buf, "c") == 0)

\*key = key\_C;

else if (strcmp (buf, "d") == 0)

\*key = key\_D;

else if (strcmp (buf, "e") == 0)

\*key = key\_E;

else if (strcmp (buf, "f") == 0)

\*key = key\_F;

else if (strcmp (buf, "+") == 0)

\*key = key\_plus;

else if (strcmp (buf, "-") == 0)

\*key = key\_minus;

return 0;

}

rk\_readvalue.c

#include <myReadKey.h>

#include <myTerm.h>

#include <stdlib.h>

int

rk\_readvalue (int \*value, int timeout)

{

rk\_mytermregime (0, timeout, 0, 0, 0);

char buf[16] = "";

int is\_completed = 0;

int n\_symbol = 0;

while (!is\_completed)

{

enum keys key = key\_UNDEFINED;

if (rk\_readkey (&key))

return -1;

if (key == key\_ESC)

return -1;

if (n\_symbol == 0)

{

if (key == key\_plus)

{

buf[0] = '+';

mt\_print ("+");

n\_symbol++;

}

else if (key == key\_minus)

{

buf[0] = '-';

mt\_print ("-");

n\_symbol++;

}

}

else

{

if (key >= key\_0 && key <= key\_9)

{

buf[n\_symbol] = key - key\_0 + '0';

mt\_print ("%c", key - key\_0 + '0');

n\_symbol++;

}

else if (key >= key\_A && key <= key\_F)

{

buf[n\_symbol] = key - key\_A + 'A';

mt\_print ("%c", key - key\_A + 'A');

n\_symbol++;

}

}

key = key\_UNDEFINED;

if (n\_symbol == 5)

is\_completed++;

}

buf[5] = '\0';

int sign = buf[0] == '+' ? 0 : 1;

int right\_value = strtol (&buf[3], NULL, 16);

buf[3] = '\0';

int left\_value = strtol (&buf[1], NULL, 16);

if (sign && right\_value > 127 && left\_value > 126)

{

\*value = 0b100000000000000;

return 0;

}

right\_value = right\_value > 127 ? 127 : right\_value;

left\_value = left\_value > 127 ? 127 << 7 : left\_value << 7;

\*value = 0;

\*value |= (sign << 14) | right\_value | left\_value;

if (sign)

\*value = ((~(\*value - 1) & 0x3FFF) | (sign << 14));

if (sign && !right\_value && !left\_value)

\*value = 0;

return 0;

}

## mySimpleComputer

makefile

LIB\_PATH = libmySimpleComputer.a

SRC\_EXT = c

APP\_SOURCES = $(wildcard \*.$(SRC\_EXT))

LIB\_OBJECTS := $(patsubst %.$(SRC\_EXT),%.o,$(APP\_SOURCES))

DEPS = $(LIB\_OBJECTS:.o=.d)

.PHONY: all

all: $(LIB\_PATH)

-include $(DEPS)

$(LIB\_PATH): $(LIB\_OBJECTS)

ar rcs $@ $^

/%.o: /%.$(SRC\_EXT)

$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean

clean:

rm -rf $(LIB\_OBJECTS)

rm -rf $(DEPS)

rm -rf $(LIB\_PATH)

sc\_accumulatorGet.c

#include "sc\_variables.h"

#include <stdio.h>

int

sc\_accumulatorGet (int \*value)

{

if (value == NULL)

return -1;

\*value = SC\_ACCUMULATOR;

return 0;

}

sc\_accumulatorInit.c

#include "sc\_variables.h"

int

sc\_accumulatorInit (void)

{

SC\_ACCUMULATOR = 0;

return 0;

}

sc\_accumulatorSet.c

#include "sc\_variables.h"

int

sc\_accumulatorSet (int value)

{

if (value < 0 || value > 32767)

return -1;

SC\_ACCUMULATOR = value;

return 0;

}

sc\_cacheGet.c

#include "sc\_variables.h"

#include <stdio.h>

int

sc\_cacheGet (int address, int \*value)

{

if (address < 0 || address >= SC\_MEMARR\_SIZE || value == NULL)

return -1;

int cacheline\_address;

for (int i = 0; i < SC\_CACHE\_SIZE; i++)

if (cache[i].address != -1)

cache[i].downtime++;

for (int i = 0; i < SC\_CACHE\_SIZE; i++)

{

if (cache[i].address != -1)

{

cacheline\_address = address - cache[i].address;

if (cacheline\_address >= 0 && cacheline\_address < 10)

{

cache[i].downtime = 0;

\*value = cache[i].line[cacheline\_address];

return 0;

}

}

}

return -1;

}

sc\_cacheInit.c

#include "sc\_variables.h"

int

sc\_cacheInit (void)

{

for (int i = 0; i < SC\_CACHE\_SIZE; i++)

{

cache[i].address = -1;

cache[i].downtime = 0;

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

{

cache[i].line[j] = 0;

}

}

return 0;

}

sc\_cachelineGet.c

#include "sc\_variables.h"

#include <stdio.h>

int

sc\_cachelineGet (int line\_number, int \*cacheline)

{

if (cacheline == NULL)

return -1;

if (line\_number < 0 || line\_number > 4)

return -1;

int line\_size = cache[line\_number].address == 120 ? 8 : 10;

if (cache[line\_number].address != -1)

for (int i = 0; i < line\_size; i++)

cacheline[i] = cache[line\_number].line[i];

return cache[line\_number].address;

}

sc\_cacheSet.c

#include "sc\_variables.h"

#include <myTerm.h>

#include <stdio.h>

int

sc\_cacheSet (int address, int value)

{

if (address < 0 || address >= SC\_MEMARR\_SIZE)

return -1;

int i;

int max\_downtime = 0;

int displacement = 0;

int line\_size = 10;

int hit = 0;

for (i = 0; i < SC\_CACHE\_SIZE; i++)

{

if (cache[i].address == -1)

{

break;

}

if (cache[i].downtime > max\_downtime)

{

max\_downtime = cache[i].downtime;

}

if (cache[i].address == (address - (address % 10)))

{

hit = 1;

break;

}

}

if (hit)

{

cache[i].line[address % 10] = value;

return 0;

}

else

SC\_TCOUNTER = 10;

if (i == SC\_CACHE\_SIZE)

{

displacement = 1;

for (i = 0; i < SC\_CACHE\_SIZE; i++)

if (cache[i].downtime == max\_downtime)

break;

}

line\_size = cache[i].address == 120 ? 8 : 10;

if (displacement)

for (int j = 0; j < line\_size; j++)

{

SC\_MEMARR[cache[i].address + j] = cache[i].line[j];

}

cache[i].address = address - (address % 10);

line\_size = cache[i].address == 120 ? 8 : 10;

for (int j = 0; j < line\_size; j++)

{

cache[i].line[j] = SC\_MEMARR[cache[i].address + j];

}

return -2;

}

sc\_commandDecode.c

#include <stdio.h>

int

sc\_commandDecode (int value, int \*sign, int \*command, int \*operand)

{

if (sign == NULL)

return -1;

if (command == NULL)

return -1;

if (operand == NULL)

return -1;

if (value < 0 || value > 32767)

return -1;

int mask = 0x7f;

\*sign = value >> 14;

\*command = (value >> 7) & mask;

\*operand = value & mask;

return 0;

}

sc\_commandEncode.c

#include "mySimpleComputer.h"

#include <stdio.h>

int

sc\_commandEncode (int sign, int command, int operand, int \*value)

{

if (sign != 0 && sign != 1)

return -1;

if (sc\_commandValidate (command))

return -1;

if (operand < 0 || operand > 127)

return -1;

if (value == NULL)

return -1;

sign <<= 14;

command <<= 7;

\*value = operand;

\*value |= command;

\*value |= sign;

return 0;

}

sc\_commandValidate.c

int

sc\_commandValidate (int command)

{

if (command < 0 || command > 32767)

return 1;

int mask = 0x7f;

command = (command >> 7) & mask;

if (command != 0 && command != 1 && command != 10 && command != 20

&& command != 21 && command != 30 && command != 31 && command != 32

&& command != 33 && command != 40 && command != 41 && command != 42

&& command != 43 && command < 51 && command > 67)

return 1;

return 0;

}

sc\_icounterGet.c

#include "sc\_variables.h"

#include <stdio.h>

int

sc\_icounterGet (int \*value)

{

if (value == NULL)

return -1;

\*value = SC\_ICOUNTER;

return 0;

}

sc\_icounterInit.c

#include "sc\_variables.h"

int

sc\_icounterInit (void)

{

SC\_ICOUNTER = 0;

return 0;

}

sc\_icounterSet.c

#include "sc\_variables.h"

int

sc\_icounterSet (int value)

{

if (value < 0 || value > 127)

return -1;

SC\_ICOUNTER = value;

return 0;

}

sc\_memoryGet.c

#include "sc\_variables.h"

#include <mySimpleComputer.h>

#include <stdio.h>

int

sc\_memoryGet (int address, int \*value)

{

if (address < 0 || address >= SC\_MEMARR\_SIZE || value == NULL)

return -1;

if (SC\_IGNORE\_CACHE)

{

\*value = SC\_MEMARR[address];

return 0;

}

if (sc\_cacheGet (address, value))

{

sc\_cacheSet (address, \*value);

return -2;

}

return 0;

}

sc\_memoryInit.c

#include "sc\_variables.h"

int

sc\_memoryInit (void)

{

for (int i = 0; i < SC\_MEMARR\_SIZE; i++)

SC\_MEMARR[i] = 0;

return 0;

}

sc\_memoryLoad.c

#include "sc\_variables.h"

#include <stdio.h>

#include <string.h>

int

sc\_memoryLoad (char \*filename)

{

if (filename == NULL)

return -1;

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

if (file == NULL)

return -2;

int tmp\_SC\_MEMARR[SC\_MEMARR\_SIZE] = { 0 };

if (fread (tmp\_SC\_MEMARR, sizeof (int), SC\_MEMARR\_SIZE, file)

!= SC\_MEMARR\_SIZE)

{

fclose (file);

return -3;

}

memcpy (SC\_MEMARR, tmp\_SC\_MEMARR, sizeof (int) \* SC\_MEMARR\_SIZE);

fclose (file);

return 0;

}

sc\_memorySave.c

#include "sc\_variables.h"

#include <stdio.h>

int

sc\_memorySave (char \*filename)

{

if (filename == NULL)

return -1;

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

if (file == NULL)

return -1;

if (fwrite (SC\_MEMARR, sizeof (\*SC\_MEMARR), SC\_MEMARR\_SIZE, file)

!= SC\_MEMARR\_SIZE)

{

fclose (file);

return -1;

}

fclose (file);

return 0;

}

sc\_memorySet.c

#include "sc\_variables.h"

#include <mySimpleComputer.h>

int

sc\_memorySet (int address, int value)

{

if (address < 0 || address >= SC\_MEMARR\_SIZE || value < 0 || value > 32767)

return -1;

if (SC\_IGNORE\_CACHE)

{

SC\_MEMARR[address] = value;

return 0;

}

if (sc\_cacheSet (address, value))

return -2;

return 0;

}

sc\_regGet.c

#include "sc\_variables.h"

#include <stdio.h>

int

sc\_regGet (int sc\_register, int \*value)

{

if (value == NULL)

return -1;

if (sc\_register != SC\_THROTTLE && sc\_register != SC\_INVALID\_COMMAND

&& sc\_register != SC\_OUT\_OF\_MEMORY && sc\_register != SC\_DIVIDING\_BY\_ZERO

&& sc\_register != SC\_OVERFLOW)

return -1;

\*value = SC\_FLAGS & sc\_register;

if (\*value > 0)

\*value = 1;

return 0;

}

sc\_regInit.c

#include "sc\_variables.h"

int

sc\_regInit (void)

{

SC\_FLAGS = 1;

return 0;

}

sc\_regSet.c

#include "sc\_variables.h"

int

sc\_regSet (int sc\_register, int value)

{

if (sc\_register != SC\_THROTTLE && sc\_register != SC\_INVALID\_COMMAND

&& sc\_register != SC\_OUT\_OF\_MEMORY && sc\_register != SC\_DIVIDING\_BY\_ZERO

&& sc\_register != SC\_OVERFLOW)

return -1;

switch (value)

{

case 1:

SC\_FLAGS |= sc\_register;

return 0;

case 0:

SC\_FLAGS &= ~sc\_register;

return 0;

}

return -1;

}

sc\_setIgnoreCache.c

#include "sc\_variables.h"

int

sc\_setIgnoreCache (int value)

{

if (value != 0 && value != 1)

return -1;

SC\_IGNORE\_CACHE = value;

return 0;

}

sc\_tcounterGet.c

#include "sc\_variables.h"

#include <stdio.h>

int

sc\_tcounterGet (unsigned char \*value)

{

if (value == NULL)

return -1;

\*value = SC\_TCOUNTER;

return 0;

}

sc\_tcounterInit.c

#include "sc\_variables.h"

#include <stdio.h>

int

sc\_tcounterInit (void)

{

SC\_TCOUNTER = 0;

return 0;

}

sc\_tcounterSet.c

#include "sc\_variables.h"

int

sc\_tcounterSet (unsigned char value)

{

SC\_TCOUNTER = value;

return 0;

}

sc\_variables.c

#include "sc\_variables.h"

int SC\_MEMARR[SC\_MEMARR\_SIZE];

cacheline cache[SC\_CACHE\_SIZE];

int SC\_ACCUMULATOR;

int SC\_ICOUNTER;

unsigned char SC\_TCOUNTER;

int SC\_FLAGS;

int SC\_IGNORE\_CACHE;

sc\_variables.h

#pragma once

#define SC\_MEMARR\_SIZE 128

#define SC\_CACHE\_SIZE 5

#define SC\_OVERFLOW 16

#define SC\_DIVIDING\_BY\_ZERO 8

#define SC\_OUT\_OF\_MEMORY 4

#define SC\_INVALID\_COMMAND 2

#define SC\_THROTTLE 1

extern int SC\_MEMARR[SC\_MEMARR\_SIZE];

extern int SC\_ACCUMULATOR;

extern int SC\_ICOUNTER;

extern unsigned char SC\_TCOUNTER;

extern int SC\_FLAGS;

extern int SC\_IGNORE\_CACHE;

typedef struct sc\_cache\_line

{

int address;

int downtime;

int line[10];

} cacheline;

extern cacheline cache[SC\_CACHE\_SIZE];

## myTerm

colors.h

enum colors

{

BLACK,

RED,

GREEN,

YELLOW,

BLUE,

PURPLE,

CYAN,

WHITE,

DEFAULT

};

makefile

LIB\_PATH = libmyTerm.a

SRC\_EXT = c

APP\_SOURCES = $(wildcard \*.$(SRC\_EXT))

LIB\_OBJECTS := $(patsubst %.$(SRC\_EXT),%.o,$(APP\_SOURCES))

DEPS = $(LIB\_OBJECTS:.o=.d)

.PHONY: all

all: $(LIB\_PATH)

-include $(DEPS)

$(LIB\_PATH): $(LIB\_OBJECTS)

ar rcs $@ $^

/%.o: /%.$(SRC\_EXT)

$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean

clean:

rm -rf $(LIB\_OBJECTS) $(DEPS) $(LIB\_PATH)

mt\_clrscr.c

#include <string.h>

#include <unistd.h>

int

mt\_clrscr (void)

{

const char \*str = "\E[H\E[2J\E[0;0H";

ssize\_t bytes\_written = write (STDOUT\_FILENO, str, strlen (str));

if (bytes\_written == -1)

return -1;

return 0;

}

mt\_delline.c

#include <string.h>

#include <unistd.h>

int

mt\_delline (void)

{

const char \*esc = "\E[M";

if (write (STDOUT\_FILENO, esc, strlen (esc)) == -1)

return -1;

return 0;

}

mt\_getscreensize.c

#include <sys/ioctl.h>

#include <unistd.h>

int

mt\_getscreensize (int \*rows, int \*cols)

{

struct winsize ws;

if (ioctl (STDOUT\_FILENO, TIOCGWINSZ, &ws))

return -1;

\*rows = ws.ws\_row;

\*cols = ws.ws\_col;

return 0;

}

mt\_gotoXY.c

#include "mt\_itoa.h"

#include <string.h>

#include <unistd.h>

int

mt\_gotoXY (int x, int y)

{

char buf[32];

char \*ptr = buf;

const char \*esc = "\033[";

const char \*sep = ";";

const char \*end = "H";

strcpy (ptr, esc);

ptr += strlen (esc);

char x\_str[16];

mt\_itoa (x, x\_str);

strcpy (ptr, x\_str);

ptr += strlen (x\_str);

\*ptr++ = sep[0];

char y\_str[16];

mt\_itoa (y, y\_str);

strcpy (ptr, y\_str);

ptr += strlen (y\_str);

\*ptr++ = end[0];

if (write (STDOUT\_FILENO, buf, ptr - buf) == -1)

{

return -1;

}

return 0;

}

mt\_itoa.c

#include <string.h>

void

mt\_itoa (int n, char \*buf)

{

int i = 0;

if (n == 0)

{

buf[i++] = '0';

}

else

{

while (n != 0)

{

int digit = n % 10;

buf[i++] = digit + '0';

n /= 10;

}

}

buf[i] = '\0';

int len = strlen (buf);

for (int j = 0; j < len / 2; ++j)

{

char temp = buf[j];

buf[j] = buf[len - j - 1];

buf[len - j - 1] = temp;

}

}

mt\_itoa.h

void mt\_itoa (int n, char \*buf);

mt\_print.c

#include <stdarg.h>

#include <stdio.h>

#include <unistd.h>

int

mt\_print (char \*format, ...)

{

int buffer\_size = 128;

char buffer[buffer\_size];

size\_t length = 0;

va\_list arguments;

va\_start (arguments, format);

length = vsnprintf (buffer, buffer\_size, format, arguments);

va\_end (arguments);

if (write (STDOUT\_FILENO, buffer, length))

return -1;

return 0;

}

mt\_setbgcolor.c

#include "colors.h"

#include <string.h>

#include <unistd.h>

int

mt\_setbgcolor (enum colors color)

{

ssize\_t bytes\_written;

char \*esc = NULL;

switch (color)

{

case BLACK:

esc = "\E[48;5;0m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case RED:

esc = "\E[48;5;1m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case GREEN:

esc = "\E[48;5;2m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case YELLOW:

esc = "\E[48;5;3m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case BLUE:

esc = "\E[48;5;4m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case PURPLE:

esc = "\E[48;5;5m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case CYAN:

esc = "\E[48;5;6m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case WHITE:

esc = "\E[48;5;7m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case DEFAULT:

esc = "\E[0m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

}

if (bytes\_written == -1)

{

return -1;

}

return 0;

}

mt\_setcursorvisible.c

#include <string.h>

#include <unistd.h>

int

mt\_setcursorvisible (int value)

{

ssize\_t bytes\_written;

char \*esc;

switch (value)

{

case 0:

esc = "\E[?25l";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case 1:

esc = "\E[?12;25h";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

}

if (bytes\_written == -1)

{

return -1;

}

return 0;

}

mt\_setdefaultcolor.c

#include <string.h>

#include <unistd.h>

int

mt\_setdefaultcolor (void)

{

const char \*esc = "\E[0m";

if (write (STDOUT\_FILENO, esc, strlen (esc)) == -1)

return -1;

return 0;

}

mt\_setfgcolor.c

#include "colors.h"

#include <string.h>

#include <unistd.h>

int

mt\_setfgcolor (enum colors color)

{

ssize\_t bytes\_written;

char \*esc = NULL;

switch (color)

{

case BLACK:

esc = "\E[38;5;0m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case RED:

esc = "\E[38;5;1m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case GREEN:

esc = "\E[38;5;2m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case YELLOW:

esc = "\E[38;5;3m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case BLUE:

esc = "\E[38;5;4m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case PURPLE:

esc = "\E[38;5;5m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case CYAN:

esc = "\E[38;5;6m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case WHITE:

esc = "\E[38;5;7m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

case DEFAULT:

esc = "\E[0m";

bytes\_written = write (STDOUT\_FILENO, esc, strlen (esc));

break;

}

if (bytes\_written == -1)

{

return -1;

}

return 0;

}

## simpleassembler

cell\_number\_error.cpp

#include "simpleassembler.h"

#include <iostream>

void

cell\_number\_error (code\_line &line, int line\_number)

{

std::cerr << "Error in line " << line\_number << ":\n"

<< line.line << std::endl;

std::cerr << "Uncorrect address: " << line.address << std::endl;

std::cerr << "Line must be like:" << std::endl;

std::cerr << "<address> <command> <operand>;<comment (optional)>"

<< std::endl;

}

cell\_value\_error.cpp

#include "simpleassembler.h"

#include <iostream>

void

cell\_value\_error (code\_line &line, int line\_number)

{

std::cerr << "Error in line " << line\_number << ":\n"

<< line.line << std::endl;

std::cerr << "Uncorrect operand: " << line.operand << std::endl;

std::cerr << "Line must be like:" << std::endl;

std::cerr << "<address> <command> <operand>;<comment (optional)>"

<< std::endl;

}

convert\_code\_lines\_to\_memory.cpp

#include "simpleassembler.h"

#include <iostream>

int \*

convert\_code\_lines\_to\_memory (code\_line \*code\_lines)

{

int \*memory = new int[128];

for (int i = 0; i < 128; i++)

memory[i] = 0;

int cell\_number;

int command;

int operand;

for (int i = 0; i < code\_line::counter; i++)

{

if (code\_lines[i].empty\_line)

{

std::cout << code\_lines[i].line << std::endl;

continue;

}

if (code\_lines[i].error\_line)

{

default\_error\_output (code\_lines[i], i + 1);

return NULL;

}

try

{

cell\_number = std::stoi (code\_lines[i].address);

}

catch (const std::invalid\_argument &e)

{

default\_error\_output (code\_lines[i], i + 1);

return NULL;

}

if (cell\_number < 0 || cell\_number > 127)

{

cell\_number\_error (code\_lines[i], i + 1);

return NULL;

}

command = convert\_string\_to\_command (code\_lines[i].command);

if (command == -1)

{

unknownown\_command\_error (code\_lines[i], i + 1);

return NULL;

}

try

{

operand = std::stoi (code\_lines[i].operand);

}

catch (const std::invalid\_argument &e)

{

default\_error\_output (code\_lines[i], i + 1);

return NULL;

}

if (command == EQ)

{

if (operand < -0x7F80 || operand > 0x7F7F)

{

cell\_value\_error (code\_lines[i], i + 1);

return NULL;

}

if (code\_lines[i].operand.length () != 5)

{

cell\_value\_error (code\_lines[i], i + 1);

return NULL;

}

memory[cell\_number]

= convert\_string\_to\_cell\_value (code\_lines[i].operand);

continue;

}

else

{

if (operand < 0 || operand > 127)

{

default\_error\_output (code\_lines[i], i + 1);

return NULL;

}

}

memory[cell\_number] = 0;

memory[cell\_number] |= (command << 7) | operand;

}

return memory;

}

convert\_string\_to\_cell\_value.cpp

#include "simpleassembler.h"

#include <cstring>

#include <string>

int

convert\_string\_to\_cell\_value (std::string str)

{

char buf[128];

int value;

strcpy (buf, str.c\_str ());

int sign = buf[0] == '+' ? 0 : 1;

int right\_value = strtol (&buf[3], NULL, 16);

buf[3] = '\0';

int left\_value = strtol (&buf[1], NULL, 16);

if (sign && right\_value > 127 && left\_value > 126)

{

value = 0b100000000000000;

return value;

}

right\_value = right\_value > 127 ? 127 : right\_value;

left\_value = left\_value > 127 ? 127 << 7 : left\_value << 7;

value = 0;

value |= (sign << 14) | right\_value | left\_value;

if (sign)

value = ((~(value - 1) & 0x3FFF) | (sign << 14));

if (sign && !right\_value && !left\_value)

value = 0;

return value;

}

convert\_string\_to\_command.cpp

#include "simpleassembler.h"

#include <string>

int

convert\_string\_to\_command (std::string command)

{

if (command == "NOP")

return NOP;

else if (command == "CPUINFO")

return CPUINFO;

else if (command == "READ")

return READ;

else if (command == "WRITE")

return WRITE;

else if (command == "LOAD")

return LOAD;

else if (command == "STORE")

return STORE;

else if (command == "ADD")

return ADD;

else if (command == "SUB")

return SUB;

else if (command == "DIVIDE")

return DIVIDE;

else if (command == "MUL")

return MUL;

else if (command == "JUMP")

return JUMP;

else if (command == "JNEG")

return JNEG;

else if (command == "JZ")

return JZ;

else if (command == "HALT")

return HALT;

else if (command == "NOT")

return NOT;

else if (command == "AND")

return AND;

else if (command == "OR")

return OR;

else if (command == "XOR")

return XOR;

else if (command == "JNS")

return JNS;

else if (command == "JC")

return JC;

else if (command == "JNC")

return JNC;

else if (command == "JP")

return JP;

else if (command == "JNP")

return JNP;

else if (command == "CHL")

return CHL;

else if (command == "SHR")

return SHR;

else if (command == "RCL")

return RCL;

else if (command == "RCR")

return RCR;

else if (command == "NEG")

return NEG;

else if (command == "ADDC")

return ADDC;

else if (command == "SUBC")

return SUBC;

else if (command == "LOGLC")

return LOGLC;

else if (command == "LOGRC")

return LOGRC;

else if (command == "=")

return EQ;

return -1;

}

convert\_strings\_to\_code\_line.cpp

#include "simpleassembler.h"

#include <cstring>

#include <string>

#include <vector>

code\_line \*

convert\_strings\_to\_code\_line (std::vector<std::string> lines)

{

code\_line \*code\_lines = new code\_line[lines.size ()];

code\_line::counter = lines.size ();

char char\_string\_for\_line[128];

char \*char\_string\_for\_token;

for (size\_t i = 0; i < lines.size (); i++)

{

code\_lines[i].line = lines[i];

code\_lines[i].empty\_line = false;

code\_lines[i].error\_line = false;

strcpy (char\_string\_for\_line, lines[i].c\_str ());

char\_string\_for\_token = strtok (char\_string\_for\_line, " ");

if (char\_string\_for\_token == NULL)

{

code\_lines[i].empty\_line = true;

continue;

}

code\_lines[i].address = char\_string\_for\_token;

char\_string\_for\_token = NULL;

char\_string\_for\_token = strtok (NULL, " ");

if (char\_string\_for\_token == NULL)

{

code\_lines[i].error\_line = true;

continue;

}

code\_lines[i].command = char\_string\_for\_token;

char\_string\_for\_token = NULL;

char\_string\_for\_token = strtok (NULL, " ");

if (char\_string\_for\_token == NULL)

{

code\_lines[i].error\_line = true;

continue;

}

code\_lines[i].operand = char\_string\_for\_token;

char\_string\_for\_token = NULL;

strtok (NULL, " ");

char\_string\_for\_token = strtok (NULL, ";");

if (char\_string\_for\_token == NULL)

continue;

code\_lines[i].comment = char\_string\_for\_token;

}

return code\_lines;

}

default\_error\_output.cpp

#include "simpleassembler.h"

#include <iostream>

void

default\_error\_output (code\_line &line, int line\_number)

{

std::cerr << "Error in line " << line\_number << ":\n"

<< line.line << std::endl;

std::cerr << "Line must be like:" << std::endl;

std::cerr << "<address> <command> <operand>;<comment (optional)>"

<< std::endl;

}

main.cpp

#include "simpleassembler.h"

#include <cstring>

#include <fstream>

#include <iostream>

#include <string>

#include <vector>

int code\_line::counter = 0;

int

main (int argc, char \*argv[])

{

if (argc != 3)

{

std::cerr << "Usage: " << argv[0] << " <input file> <output file>"

<< std::endl;

return 1;

}

char input\_file\_name[256];

char \*input\_file\_name\_p;

strcpy (input\_file\_name, argv[1]);

input\_file\_name\_p = strtok (input\_file\_name, ".");

input\_file\_name\_p = strtok (NULL, ".");

if (strcmp (input\_file\_name\_p, "sa") != 0)

{

std::cerr << "Uncorrect input file extension!" << std::endl;

std::cerr << "It must be like: <filename>.sa" << std::endl;

return 1;

}

std::vector<std::string> lines = read\_file (argv[1]);

code\_line \*code\_lines = convert\_strings\_to\_code\_line (lines);

int \*memory = convert\_code\_lines\_to\_memory (code\_lines);

delete[] code\_lines;

if (memory == NULL)

return 1;

if (write\_memory\_to\_file (memory, 128, argv[2]))

{

delete[] memory;

return 1;

}

return 0;

}

makefile

APP\_NAME = sat

SRC\_EXT = cpp

CC = g++

CFLAGS = -Wall -Wextra -Werror

CPPFLAGS = -MMD

APP\_SOURCES = $(wildcard \*.$(SRC\_EXT))

APP\_OBJECTS := $(patsubst %.$(SRC\_EXT),%.o,$(APP\_SOURCES))

DEPS = $(APP\_OBJECTS:.o=.d)

.PHONY: all

all: $(APP\_NAME)

-include $(DEPS)

$(APP\_NAME): $(APP\_OBJECTS)

$(CC) $(CFLAGS) $(CPPFLAGS) $^ -o $@

%.o: %.$(SRC\_EXT)

$(CC) $(CFLAGS) $(CPPFLAGS) -c $< -o $@

.PHONY: clean

clean:

rm -rf $(APP\_OBJECTS) $(DEPS) $(APP\_NAME)

read\_file.cpp

#include "simpleassembler.h"

#include <fstream>

#include <string>

#include <vector>

std::vector<std::string>

read\_file (char \*filename)

{

std::ifstream file (filename);

std::vector<std::string> lines;

std::string line;

while (std::getline (file, line))

{

lines.push\_back (line);

}

file.close ();

return lines;

}

simpleassembler.h

#pragma once

#include <string>

#include <vector>

enum commands

{

NOP = 0x00,

CPUINFO = 0x01,

READ = 0x0A,

WRITE = 0x0B,

LOAD = 0x14,

STORE = 0x15,

ADD = 0x1E,

SUB = 0x1F,

DIVIDE = 0x20,

MUL = 0x21,

JUMP = 0x28,

JNEG = 0x29,

JZ = 0x2A,

HALT = 0x2B,

NOT = 0x33,

AND = 0x34,

OR = 0x35,

XOR = 0x36,

JNS = 0x37,

JC = 0x38,

JNC = 0x39,

JP = 0x3A,

JNP = 0x3B,

CHL = 0x3C,

SHR = 0x3D,

RCL = 0x3E,

RCR = 0x3F,

NEG = 0x40,

ADDC = 0x41,

SUBC = 0x42,

LOGLC = 0x43,

LOGRC = 0x44,

RCCL = 0x45,

RCCR = 0x46,

MOVA = 0x47,

MOVR = 0x48,

MOVCA = 0x49,

MOVCR = 0x4A,

ADDC2 = 0x4B,

SUBC2 = 0x4C,

EQ = 0x4D

};

class code\_line

{

public:

static int counter;

bool empty\_line;

bool error\_line;

std::string line;

std::string address;

std::string command;

std::string operand;

std::string comment;

};

std::vector<std::string> read\_file (char \*filename);

code\_line \*convert\_strings\_to\_code\_line (std::vector<std::string> lines);

bool validate\_code\_lines (code\_line \*code\_lines);

void default\_error\_output (code\_line &line, int line\_number);

void unknownown\_command\_error (code\_line &line, int line\_number);

void cell\_number\_error (code\_line &line, int line\_number);

void cell\_value\_error (code\_line &line, int line\_number);

int convert\_string\_to\_command (std::string command);

int convert\_string\_to\_cell\_value (std::string str);

int \*convert\_code\_lines\_to\_memory (code\_line \*code\_lines);

int write\_memory\_to\_file (int \*memory, int size, std::string filename);

unknown\_command\_error.cpp

#include "simpleassembler.h"

#include <iostream>

void

unknownown\_command\_error (code\_line &line, int line\_number)

{

std::cerr << "Error in line " << line\_number << ":\n"

<< line.line << std::endl;

std::cerr << "Unknown command: " << line.command << std::endl;

std::cerr << "Line must be like:" << std::endl;

std::cerr << "<address> <command> <operand>;<comment (optional)>"

<< std::endl;

}

validate\_code\_lines.cpp

#include "simpleassembler.h"

bool

validate\_code\_lines (code\_line \*code\_lines)

{

for (int i = 0; i < code\_line::counter; i++)

{

if (code\_lines[i].error\_line)

return true;

}

return false;

}

write\_memory\_to\_file.cpp

#include <fstream>

#include <iostream>

#include <string>

int

write\_memory\_to\_file (int \*memory, int size, std::string filename)

{

std::ofstream outputFile (filename, std::ios::binary);

if (outputFile.is\_open ())

{

outputFile.write (reinterpret\_cast<const char \*> (memory),

sizeof (int) \* size);

outputFile.close ();

std::cout << "Массив успешно записан в файл в бинарном виде."

<< std::endl;

}

else

{

std::cerr << "Ошибка открытия файла для записи." << std::endl;

return -1;

}

return 0;

}

## simplebasic

check\_code\_lines\_empty.cpp

#include "simplebasic.h"

#include <iostream>

#include <vector>

int

check\_code\_lines\_empty (vector<code\_line> &code\_lines)

{

int i = 0;

for (auto &cl : code\_lines)

{

i++;

if (cl.is\_empty ())

{

cerr << "Error in " << i << " line" << endl;

cerr << "This is an empty line" << endl;

return 1;

}

}

return 0;

}

check\_code\_lines\_spaces.cpp

#include "simplebasic.h"

#include <iostream>

#include <vector>

int

check\_code\_lines\_spaces (vector<code\_line> &code\_lines)

{

int i = 0;

for (auto &cl : code\_lines)

{

i++;

if (!cl.is\_there\_any\_spaces ())

{

cerr << "Error in " << i << " line" << endl;

cerr << "Code line must have at least line number and command"

<< endl;

cerr << "This is an empty line" << endl;

return 1;

}

}

return 0;

}

code\_lines\_to\_tokens.cpp

#include "simplebasic.h"

#include <iostream>

#include <vector>

int

code\_lines\_to\_tokens (vector<code\_line> &code\_lines)

{

int i = 0;

for (auto &cl : code\_lines)

{

i++;

if (cl.split ())

{

cerr << "Error in " << i << " line" << endl;

cerr << cl.basic\_code << endl;

cerr << "This is an invalid code line" << endl;

return 1;

}

}

return 0;

}

convert\_basic\_code\_line\_to\_assembler.cpp

#include "simplebasic.h"

string

convert\_basic\_code\_line\_to\_assembler (code\_line &cl, map<char, int> &table,

map<int, int> &addresses)

{

string result = "";

int address = cl.assembler\_line\_number;

if (cl.first\_command == "INPUT")

{

result += int\_to\_address (address);

result += " READ ";

result += int\_to\_address (table[cl.operand[0]]);

result += '\n';

}

else if (cl.first\_command == "OUTPUT")

{

result += int\_to\_address (address);

result += " WRITE ";

result += int\_to\_address (table[cl.operand[0]]);

result += '\n';

}

else if (cl.first\_command == "GOTO")

{

result += int\_to\_address (address);

result += " JUMP ";

result += int\_to\_address (addresses[stoi (cl.operand)]);

result += '\n';

}

else if (cl.first\_command == "LET")

{

result += convert\_let\_to\_assembler (address, cl.postfix\_expression,

cl.expression\_result, table);

}

else if (cl.first\_command == "IF")

{

result += convert\_if\_to\_assembler (cl, table, addresses);

}

else if (cl.first\_command == "END")

{

result += int\_to\_address (address);

result += " HALT ";

result += "00\n";

}

return result;

}

convert\_basic\_code\_lines\_addresses\_to\_assembler.cpp

#include "simplebasic.h"

map<int, int>

convert\_basic\_code\_lines\_adresses\_to\_assembler (vector<code\_line> &code\_lines)

{

map<int, int> addresses;

int address = 0;

for (auto &cl : code\_lines)

{

addresses[cl.line\_number] = address;

cl.assembler\_line\_number = address;

if (cl.first\_command == "LET")

{

address += count\_commands (cl.first\_command, cl.postfix\_expression);

continue;

}

if (cl.first\_command == "LET")

{

address += count\_commands (cl.second\_command, cl.postfix\_expression);

continue;

}

if (cl.first\_command == "IF")

{

address += count\_commands (cl.first\_command, cl.first\_expression);

address += count\_commands (cl.second\_command, cl.second\_expression);

continue;

}

address += count\_commands (cl.first\_command, cl.first\_expression);

}

return addresses;

}

convert\_basic\_code\_lines\_to\_assembler.cpp

#include "simplebasic.h"

string

convert\_basic\_code\_lines\_to\_assembler (vector<code\_line> &code\_lines,

map<char, int> &table,

map<int, int> &addresses,

map<char, pair<int, int> > &constants)

{

string result = "";

for (auto &cl : code\_lines)

{

result += convert\_basic\_code\_line\_to\_assembler (cl, table, addresses);

}

for (auto c = constants.rbegin (); c != constants.rend (); ++c)

{

result += int\_to\_address (c->second.first);

result += " = ";

result += int\_to\_sc\_number (c->second.second);

result += "\n";

}

return result;

}

convert\_if\_to\_assembler.cpp

#include "simplebasic.h"

string

convert\_if\_to\_assembler (code\_line &cl, map<char, int> &table,

map<int, int> &addresses)

{

string result = "";

int address = cl.assembler\_line\_number;

int left;

int right;

char left\_operand;

char right\_operand;

char sign;

int jump\_to;

for (auto &ad : addresses)

{

if (ad.second > address)

{

jump\_to = ad.second;

break;

}

}

tie (left, right) = is\_there\_numbers\_in\_if (cl.first\_expression);

if (left != 100000)

{

result += int\_to\_address (address++);

result += " = ";

result += int\_to\_sc\_number (left);

}

if (right != 100000)

{

result += int\_to\_address (address++);

result += " = ";

result += int\_to\_sc\_number (right);

}

result += int\_to\_address (address++);

result += " LOAD ";

if (left != 100000)

{

if (right != 100000)

result += int\_to\_address (address - 3);

else

result += int\_to\_address (address - 2);

}

else

{

left\_operand += cl.first\_expression[0];

result += int\_to\_address (table[left\_operand]);

}

result += "\n";

result += int\_to\_address (address++);

result += " SUB ";

if (right != 100000)

result += int\_to\_address (address - 3);

else

{

right\_operand

= cl.first\_expression[cl.first\_expression.find\_first\_of ("<>=") + 1];

result += int\_to\_address (table[right\_operand]);

}

result += "\n";

sign = cl.first\_expression[cl.first\_expression.find\_first\_of ("<>=")];

switch (sign)

{

case ('<'):

result += int\_to\_address (address++);

result += " JZ ";

result += int\_to\_address (jump\_to);

result += "\n";

result += int\_to\_address (address++);

result += " JNS ";

result += int\_to\_address (jump\_to);

result += "\n";

break;

case ('>'):

result += int\_to\_address (address++);

result += " JZ ";

result += int\_to\_address (jump\_to);

result += "\n";

result += int\_to\_address (address++);

result += " JNEG ";

result += int\_to\_address (jump\_to);

result += "\n";

break;

case ('='):

result += int\_to\_address (address++);

result += " JNEG ";

result += int\_to\_address (jump\_to);

result += "\n";

result += int\_to\_address (address++);

result += " JNS ";

result += int\_to\_address (jump\_to);

result += "\n";

break;

}

if (cl.second\_command == "LET")

{

result += convert\_let\_to\_assembler (address, cl.postfix\_expression,

cl.expression\_result, table);

}

else if (cl.second\_command == "INPUT")

{

result += int\_to\_address (address);

result += " READ ";

result += int\_to\_address (table[cl.operand[0]]);

result += '\n';

}

else if (cl.second\_command == "OUTPUT")

{

result += int\_to\_address (address);

result += " WRITE ";

result += int\_to\_address (table[cl.operand[0]]);

result += '\n';

}

else if (cl.second\_command == "GOTO")

{

result += int\_to\_address (address);

result += " JUMP ";

result += int\_to\_address (addresses[stoi (cl.operand)]);

result += '\n';

}

return result;

}

convert\_infix\_to\_postfix.cpp

#include "simplebasic.h"

void

convert\_infix\_to\_postfix (vector<code\_line> &code\_lines,

map<char, pair<int, int> > &constants)

{

for (auto &cl : code\_lines)

{

if (cl.first\_command == "LET")

{

cl.split\_expression ();

cl.postfix\_expression

= infix\_to\_postfix (cl.first\_expression, constants);

}

if (cl.second\_command == "LET")

{

cl.split\_expression ();

cl.postfix\_expression

= infix\_to\_postfix (cl.second\_expression, constants);

}

}

}

convert\_let\_to\_assembler.cpp

#include "simplebasic.h"

string

convert\_let\_to\_assembler (int address, string &expression,

string &expression\_result, map<char, int> &table)

{

int n = 0;

string left;

string right;

char sign;

string result = "";

for (auto c : expression)

{

if (isalpha (c))

n++;

}

if (n == 1)

{

result += int\_to\_address (address++);

result += " LOAD ";

result += int\_to\_address (table[expression[0]]);

result += "\n";

result += int\_to\_address (address++);

result += " STORE ";

result += int\_to\_address (table[expression\_result[0]]);

result += "\n";

return result;

}

n--;

for (int i = 0; i < n; i++)

{

tie (left, right, sign)

= get\_one\_operation (expression, expression\_result);

result += int\_to\_address (address++);

result += " LOAD ";

result += int\_to\_address (table[left[0]]);

result += "\n";

result += int\_to\_address (address++);

switch (sign)

{

case ('+'):

result += " ADD ";

break;

case ('-'):

result += " SUB ";

break;

case ('\*'):

result += " MUL ";

break;

case ('/'):

result += " DIVIDE ";

break;

}

result += int\_to\_address (table[right[0]]);

result += "\n";

result += int\_to\_address (address++);

result += " STORE ";

result += int\_to\_address (table[expression\_result[0]]);

result += "\n";

}

return result;

}

count\_commands.cpp

#include <string>

using namespace std;

int

count\_commands (string &command, string &expression)

{

bool inNumber = false;

int count = 0;

bool there\_is\_expression = false;

if (command == "REM")

return 0;

if (command == "GOTO" || command == "INPUT" || command == "OUTPUT"

|| command == "END")

return 1;

if (command == "IF")

{

for (char c : expression)

{

if (isdigit (c))

{

if (!inNumber)

{

inNumber = true;

count++;

}

}

else

{

inNumber = false;

}

}

return count + 4;

}

if (command == "LET")

{

for (char c : expression)

{

if (isalpha (c))

count += 3;

if (c == '+' || c == '-' || c == '\*' || c == '/')

there\_is\_expression = true;

}

if (there\_is\_expression)

return count - 3;

else

return 2;

}

return 0;

}

get\_one\_operation.cpp

#include "simplebasic.h"

#include <string>

#include <tuple>

using namespace std;

tuple<string, string, char>

get\_one\_operation (string &expression, string &expression\_result)

{

bool there\_is\_expression = false;

for (char c : expression)

{

if (c == '+' || c == '-' || c == '\*' || c == '/')

{

there\_is\_expression = true;

break;

}

}

if (!there\_is\_expression)

return make\_tuple ("", "", 0);

size\_t position = expression.find\_first\_of ("+-\*/");

char sign = expression[position];

string right\_operand = take\_operand (expression);

string left\_operand = take\_operand (expression);

position = expression.find\_first\_of ("+-\*/");

expression[position] = expression\_result[0];

return make\_tuple (left\_operand, right\_operand, sign);

}

get\_symbols\_from\_expression.cpp

#include <map>

#include <string>

using namespace std;

int

get\_symbols\_from\_expression (map<char, int> &table, string &expression,

int address)

{

for (char c : expression)

if (isupper (c))

if (table.count (c) == 0)

table[c] = address--;

return address;

}

give\_addresses\_to\_constants.cpp

#include <map>

#include <utility>

using namespace std;

void

give\_addresses\_to\_constants (map<char, int> &table,

map<char, pair<int, int> > &constants)

{

auto last\_address = table.end ();

--last\_address;

int address = last\_address->second - 1;

for (auto &c : constants)

c.second.first = address--;

}

infix\_to\_postfix.cpp

#include "simplebasic.h"

#include <map>

#include <stack>

#include <string>

#include <utility>

using namespace std;

string

infix\_to\_postfix (string &expression, map<char, pair<int, int> > &constants)

{

std::stack<char> s;

std::stack<int> s\_prec;

int increment = 0;

int prec = 0;

string result;

char constant = 'a';

int exist = false;

int int\_number;

string number = "";

if (constants.size () != 0)

{

auto last\_constant = constants.end ();

--last\_constant;

constant = last\_constant->first + 1;

}

for (char c : expression)

{

if (isdigit (c))

{

number += c;

continue;

}

if (number != "")

{

int\_number = stoi (number);

for (auto &c : constants)

{

if (c.second.second == int\_number)

{

constant = c.first;

exist = true;

}

}

if (!exist)

{

if (constants.size () != 0)

{

auto last\_constant = constants.end ();

--last\_constant;

constant = last\_constant->first + 1;

}

constants[constant] = make\_pair (0, int\_number);

}

result += constant;

exist = false;

number = "";

}

prec = precedence (c);

switch (prec)

{

case 0:

result += c;

break;

case 3:

increment += 2;

break;

case 4:

increment -= 2;

break;

default:

prec += increment;

if (s.size () == 0)

{

s.push (c);

s\_prec.push (prec);

}

else if (s\_prec.top () >= prec)

{

while (s.size () > 0)

{

result += s.top ();

s.pop ();

s\_prec.pop ();

}

s.push (c);

s\_prec.push (prec);

}

else

{

s.push (c);

s\_prec.push (prec);

}

break;

}

}

if (number != "")

{

int\_number = stoi (number);

for (auto &c : constants)

{

if (c.second.second == int\_number)

{

constant = c.first;

exist = true;

}

}

if (!exist)

{

if (constants.size () != 0)

{

auto last\_constant = constants.end ();

--last\_constant;

constant = last\_constant->first + 1;

}

constants[constant] = make\_pair (0, int\_number);

}

result += constant++;

exist = false;

number = "";

}

while (s.size () > 0)

{

result += s.top ();

s.pop ();

s\_prec.pop ();

}

return result;

}

int\_to\_address.cpp

#include <iomanip>

#include <sstream>

#include <string>

using namespace std;

string

int\_to\_address (int address)

{

ostringstream oss;

oss << setfill ('0') << setw (2) << address;

return oss.str ();

}

int\_to\_sc\_number.cpp

#include <iomanip>

#include <sstream>

#include <string>

using namespace std;

string

int\_to\_sc\_number (int value)

{

stringstream stream;

stream << hex << uppercase << setfill ('0') << setw (4) << value;

if (value < 0)

{

return '-' + stream.str ();

}

return '+' + stream.str ();

}

is\_there\_numbers\_in\_if.cpp

#include <string>

#include <utility>

using namespace std;

pair<int, int>

is\_there\_numbers\_in\_if (string &expression)

{

int left = 100000;

int right = 100000;

int index = 0;

string expression\_copy = expression;

if (expression\_copy[0] == '-' || isdigit (expression\_copy[0]))

{

left = stoi (expression\_copy);

}

index = expression\_copy.find\_first\_of ("><=");

expression\_copy.erase (0, index);

if (expression[0] == '-' || isdigit (expression[0]))

{

left = stoi (expression\_copy);

}

return make\_pair (left, right);

}

main.cpp

#include "simplebasic.h"

#include <fstream>

#include <iostream>

int

main (int argc, char \*argv[])

{

if (argc != 3)

{

std::cerr << "Usage: " << argv[0] << " <input file> <output file>"

<< std::endl;

return 1;

}

vector<code\_line> code\_lines = read\_file (argv[1]);

if (check\_code\_lines\_empty (code\_lines))

return 1;

if (check\_code\_lines\_spaces (code\_lines))

return 1;

if (code\_lines\_to\_tokens (code\_lines))

return 1;

map<char, int> table = make\_symbolic\_table (code\_lines);

map<char, pair<int, int> > constants;

convert\_infix\_to\_postfix (code\_lines, constants);

give\_addresses\_to\_constants (table, constants);

map<int, int> addresses

= convert\_basic\_code\_lines\_adresses\_to\_assembler (code\_lines);

merge\_tables (table, constants);

string result = convert\_basic\_code\_lines\_to\_assembler (code\_lines, table,

addresses, constants);

ofstream out (argv[2]);

if (!out.is\_open ())

{

cerr << "Can't open file \"" << argv[2] << "\"" << endl;

return 1;

}

out << result;

out.close ();

cout << "Success!" << endl;

}

make\_symbolic\_table.cpp

#include "simplebasic.h"

#include <map>

#include <string>

#include <vector>

map<char, int>

make\_symbolic\_table (vector<code\_line> &code\_lines)

{

int address = 127;

map<char, int> table;

string first\_expression;

string second\_expression;

for (auto &cl : code\_lines)

{

if (cl.first\_command == "REM" || cl.first\_command == "GOTO"

|| cl.first\_command == "OUTPUT")

continue;

if (cl.first\_command == "INPUT" || cl.second\_command == "INPUT")

{

table[cl.operand[0]] = address--;

continue;

}

if (cl.first\_command == "IF")

{

address = get\_symbols\_from\_expression (table, cl.first\_expression,

address);

}

if (cl.first\_command == "LET")

address = get\_symbols\_from\_expression (table, cl.first\_expression,

address);

if (cl.second\_command == "LET")

address = get\_symbols\_from\_expression (table, cl.second\_expression,

address);

}

return table;

}

makefile

APP\_NAME = sbt

SRC\_EXT = cpp

CC = g++

CFLAGS = -Wall -Wextra -Werror

CPPFLAGS = -MMD

APP\_SOURCES = $(wildcard \*.$(SRC\_EXT))

APP\_OBJECTS := $(patsubst %.$(SRC\_EXT),%.o,$(APP\_SOURCES))

DEPS = $(APP\_OBJECTS:.o=.d)

.PHONY: all

all: $(APP\_NAME)

-include $(DEPS)

$(APP\_NAME): $(APP\_OBJECTS)

$(CC) $(CFLAGS) $(CPPFLAGS) $^ -o $@

%.o: %.$(SRC\_EXT)

$(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@

.PHONY: clean

clean:

rm -rf $(APP\_OBJECTS) $(DEPS) $(APP\_NAME)

merge\_tables.cpp

#include <map>

#include <utility>

using namespace std;

void

merge\_tables (map<char, int> &table, map<char, pair<int, int> > &constants)

{

for (auto &c : constants)

{

table[c.first] = c.second.first;

}

}

precedence.cpp

int

precedence (char op)

{

if (op == '+' || op == '-')

return 1;

if (op == '\*' || op == '/')

return 2;

if (op == '(')

return 3;

if (op == ')')

return 4;

return 0;

}

read\_file.cpp

#include "simplebasic.h"

#include <fstream>

#include <iostream>

#include <vector>

vector<code\_line>

read\_file (string filename)

{

ifstream file (filename);

if (!file.is\_open ())

{

cerr << "Can't open file \"" << filename << "\"" << endl;

exit (1);

}

string line;

vector<code\_line> code\_lines;

while (getline (file, line))

{

code\_line cl;

cl.basic\_code = line;

code\_lines.push\_back (cl);

}

file.close ();

return code\_lines;

}

simplebasic.h

#pragma once

#include <map>

#include <sstream>

#include <string>

#include <vector>

using namespace std;

class code\_line

{

public:

string basic\_code;

string first\_command;

string second\_command;

string first\_expression;

string second\_expression;

string operand;

string postfix\_expression;

string expression\_result;

int line\_number;

int assembler\_line\_number;

code\_line ()

{

basic\_code = "";

first\_command = "";

second\_command = "";

first\_expression = "";

second\_expression = "";

operand = "";

postfix\_expression = "";

expression\_result = "";

line\_number = 0;

assembler\_line\_number = 0;

}

bool

is\_empty ()

{

return basic\_code.empty ();

}

bool

is\_there\_any\_spaces ()

{

return basic\_code.find (' ') != string::npos;

}

bool

token\_is\_command (string token)

{

return token == "REM" || token == "INPUT" || token == "OUTPUT"

|| token == "GOTO" || token == "IF" || token == "LET"

|| token == "END";

}

int

split ()

{

istringstream iss (basic\_code);

string token;

iss >> token;

if (token.empty ())

return 1;

line\_number = stoi (token);

iss >> token;

if (!token\_is\_command (token))

return 1;

first\_command = token;

if (first\_command == "REM")

{

return 0;

}

else if (first\_command == "INPUT" || first\_command == "OUTPUT"

|| first\_command == "GOTO")

{

iss >> token;

if (token.empty ())

return 1;

operand = token;

return 0;

}

else if (first\_command == "IF")

{

iss >> token;

while (!token\_is\_command (token))

{

first\_expression += token;

iss >> token;

}

second\_command = token;

if (second\_command == "INPUT" || second\_command == "OUTPUT"

|| second\_command == "GOTO")

{

iss >> token;

if (token.empty ())

return 1;

operand = token;

return 0;

}

else if (second\_command == "LET")

{

token = "";

iss >> token;

while (!token.empty ())

{

second\_expression += token;

token = "";

iss >> token;

}

return 0;

}

}

else if (first\_command == "LET")

{

token = "";

iss >> token;

while (!token.empty ())

{

first\_expression += token;

token = "";

iss >> token;

}

return 0;

}

else if (first\_command == "END")

{

return 0;

}

return 1;

}

void

split\_expression ()

{

if (first\_command == "LET")

{

expression\_result += first\_expression[0];

first\_expression.erase (0, 2);

}

if (second\_command == "LET")

{

expression\_result += second\_expression[0];

second\_expression.erase (0, 2);

}

}

};

vector<code\_line> read\_file (string filename);

int check\_code\_lines\_empty (vector<code\_line> &code\_lines);

int check\_code\_lines\_spaces (vector<code\_line> &code\_lines);

int code\_lines\_to\_tokens (vector<code\_line> &code\_lines);

int get\_symbols\_from\_expression (map<char, int> &table, string &expression,

int address);

map<char, int> make\_symbolic\_table (vector<code\_line> &code\_lines);

int precedence (char op);

string infix\_to\_postfix (string &expression,

map<char, pair<int, int> > &constants);

void convert\_infix\_to\_postfix (vector<code\_line> &code\_lines,

map<char, pair<int, int> > &constants);

void give\_addresses\_to\_constants (map<char, int> &table,

map<char, pair<int, int> > &constants);

int count\_commands (string &command, string &expression);

map<int, int>

convert\_basic\_code\_lines\_adresses\_to\_assembler (vector<code\_line> &code\_lines);

string int\_to\_address (int address);

string take\_operand (string &expression);

tuple<string, string, char> get\_one\_operation (string &expression,

string &expression\_result);

void merge\_tables (map<char, int> &table,

map<char, pair<int, int> > &constants);

string convert\_let\_to\_assembler (int address, string &expression,

string &expression\_result,

map<char, int> &table);

pair<int, int> is\_there\_numbers\_in\_if (string &expression);

string int\_to\_sc\_number (int value);

string convert\_if\_to\_assembler (code\_line &cl, map<char, int> &table,

map<int, int> &addresses);

string convert\_basic\_code\_line\_to\_assembler (code\_line &cl,

map<char, int> &table,

map<int, int> &addresses);

string convert\_basic\_code\_lines\_to\_assembler (

vector<code\_line> &code\_lines, map<char, int> &table,

map<int, int> &addresses, map<char, pair<int, int> > &constants);

take\_operand.cpp

#include <string>

using namespace std;

string

take\_operand (string &expression)

{

size\_t right\_position = expression.find\_first\_of ("+-\*/");

size\_t left\_position = right\_position - 1;

string operand

= expression.substr (left\_position, right\_position - left\_position);

expression.erase (left\_position, right\_position - left\_position);

return operand;

}

## simplecomputer

makefile

export CFLAGS = -Wall -Wextra -Werror

export CPPFLAGS = -I$(PWD)/include -L$(PWD)/mySimpleComputer -L$(PWD)/myTerm -L$(PWD)/myBigChars -L$(PWD)/myReadKey -MMD

export CC = gcc

SUBDIRS = myTerm mySimpleComputer myBigChars myReadKey console

LFLAGS = -lmySimpleComputer -lmyTerm -lmyBigChars -lmyReadKey

LIBS = $(PWD)/mySimpleComputer/libmySimpleComputer.a $(PWD)/myTerm/libmyTerm.a $(PWD)/myBigChars/libmyBigChars.a $(PWD)/myReadKey/libmyReadKey.a

export LFLAGS

export LIBS

all: $(SUBDIRS)

$(SUBDIRS):

@$(MAKE) --no-print-directory -C $@

clean:

@for dir in $(SUBDIRS); do \

$(MAKE) --no-print-directory -C $$dir clean; \

done

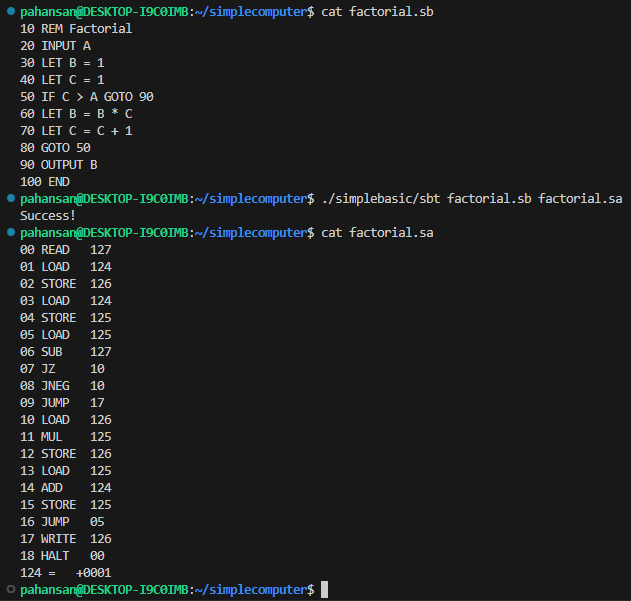
.PHONY: all clean $(SUBDIRS)

format:

find . -type f -name \*.[ch] | xargs clang-format --style GNU -i –verbose

# РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

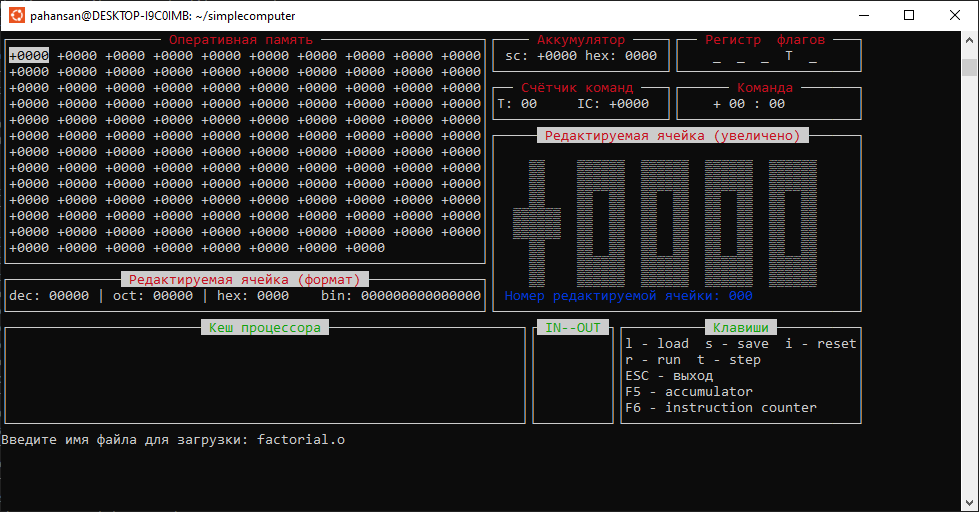
Трансляция с языка simple basic в simple assembler

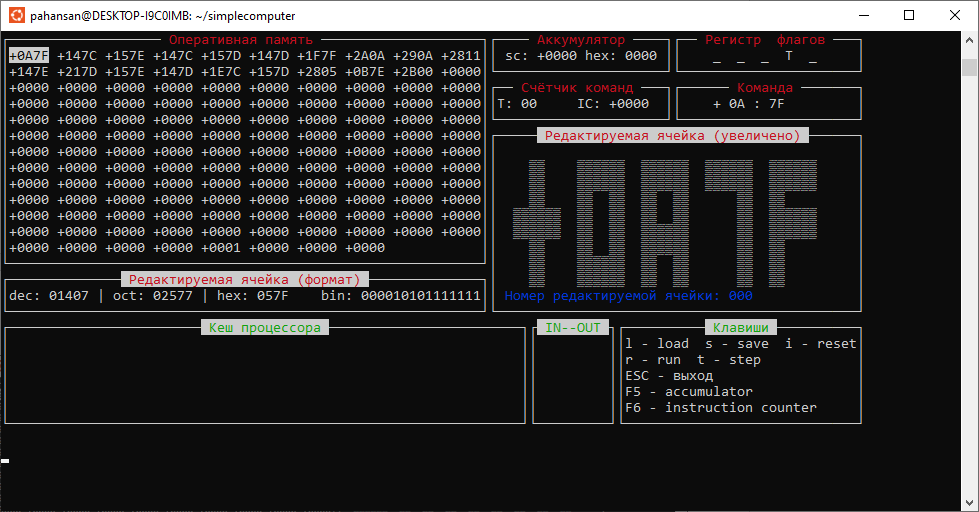


Трансляция с языка simple assembler в образ оперативной памяти



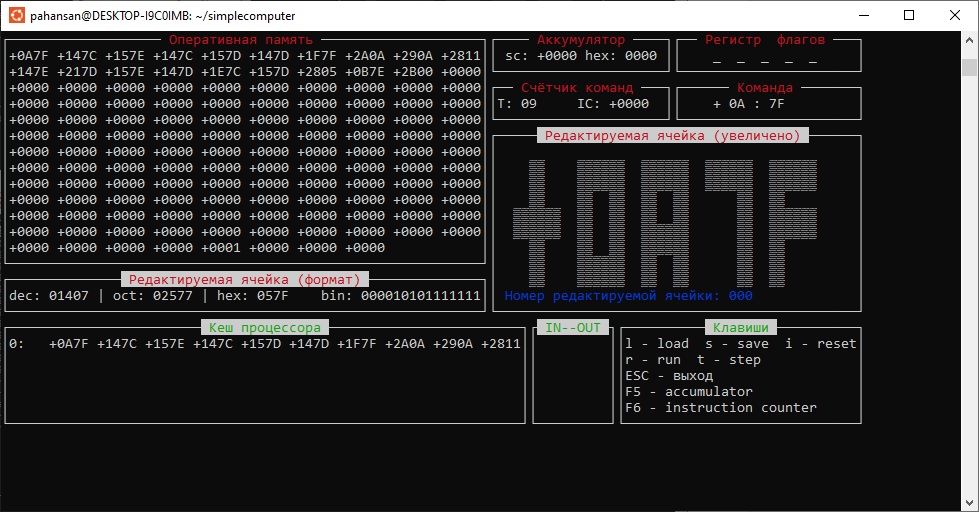
Загрузка образа оперативной памяти



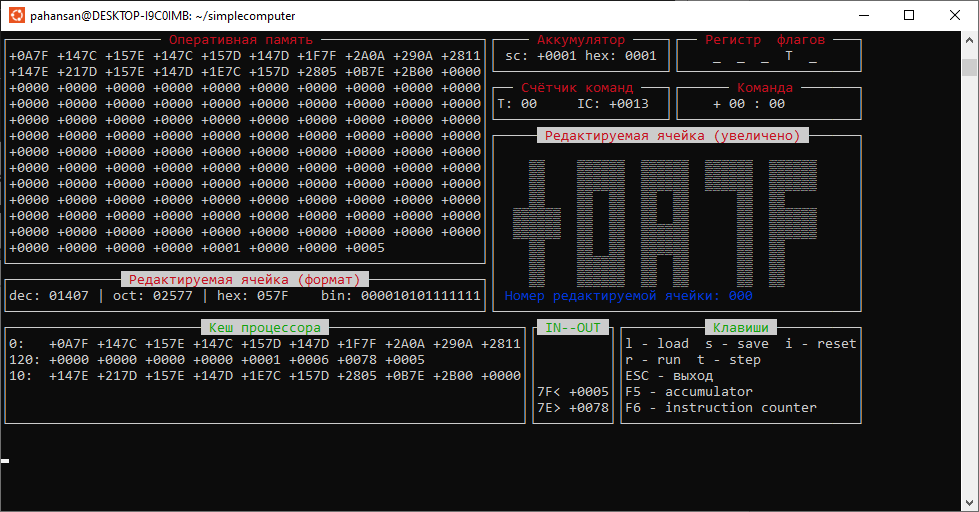


Память успешно загрузилась. Программа, которая записана в памяти, может вычислить значение факториала от 0 до 7.

При запуске программы произошёл cache miss, поэтому строка из оперативной памяти была загружена в кэш процессора и счётчик тактов простоя установился в 10.



Результат выполнения программы



Программа правильно вычислила факториал 5, запрошенный пользователем. 78 в шестнадцатеричной системе счисления – это 120 в десятичной. Видно также, что все данные, которые потребовались в ходе выполнения программы, были загружены из оперативной памяти в кэш процессора.

# ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была доработано модель Simple Computer таким образом, чтобы имитировать работу кэша процессора при выполнении программ. В модели был сымитирован алгоритм замещения кэша LRU, при котором в случае переполнения кэша из него вытесняется самая невостребованная строка.

Помимо работы кэша также были реализованы 2 транслятора: транслятор с языка simple assembler, который позволяет превратить программу на simple assembler в образ оперативной памяти simple computer, и транслятор simple basic, который превращает программу, написанную на языке более высокого уровня simple basic в программу на языке simple assembler.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Мамойленко С.Н., Молдованова О.В. ЭВМ и периферийные устройства: Учебное пособие. – Новосибирск: СибГУТИ, 2012. – 106 с.