Федеральное государственное бюджетное образовательное учреждение «Сибирский государственный университет телекоммуникаций и информатики»

Кафедра Вычислительных систем

Курсовая работа

по дисциплине «Архитектура ЭВМ»

на тему «Обработка команд центральным процессором, транслятор с языка Simple Assembler и Simple Basic»

Выполнил: студент II курса

ИВТ, гр. ИА-831

Зарубин Максим Евгеньевич

Проверил:

Майданов Юрий Сергеевич

Новосибирск 2020

Содержание

1. Текст задания к курсовой работе

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

Обработка команд центральным процессором

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

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

int *CU* (void) – обеспечивает работу устройства управления. Обработку команд осуществляет устройство управления. Функция *CU* вызывается либо обработчиком сигнала от системного таймера, если не установлен флаг «игнорирование тактовых импульсов», либо при нажатии на клавишу *t*. Алгоритм работы функции следующий:

1. из оперативной памяти считывается ячейка, адрес которой храниться в регистре *instructionCounter*;
2. полученное значение декодируется как команда;
3. если декодирование невозможно, то устанавливаются флаги «указана неверная команда» и «игнорирование тактовых импульсов» (системный таймер можно отключить) и работа функции прекращается.
4. Если получена арифметическая или логическая операция, то вызывается функция *ALU*, иначе команда выполняется самим устройством управления.
5. Определяется, какая команда должна быть выполнена следующей и адрес еѐ ячейки памяти заносится в регистр *instructionCounter*.
6. Работа функции завершается.

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 A)
01  READ  10  ; (Ввод B)
02  LOAD  09  ; (Загрузка A в аккумулятор)
03  SUB  10  ; (Отнять B)
04  JNEG  07  ; (Переход на 07, если отрицательное)
05  WRITE  09  ; (Вывод A)
06  HALT  00  ; (Останов)
07  WRITE  10  ; (Вывод B)
08  HALT  00  ; (Останов)
09  =  +0000  ; (Переменная A)
10  =  +9999  ; (Переменная B)
```

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 ее объявление и присваивает ей нулевое значение. Синтаксис языка не позволяет выполнять операций со строками.

2. Архитектура Вычислительной машины Simple Computer

Архитектура *Simple Computer* включает следующие функциональные блоки:

- оперативную память;
- внешние устройства;
- центральный процессор.

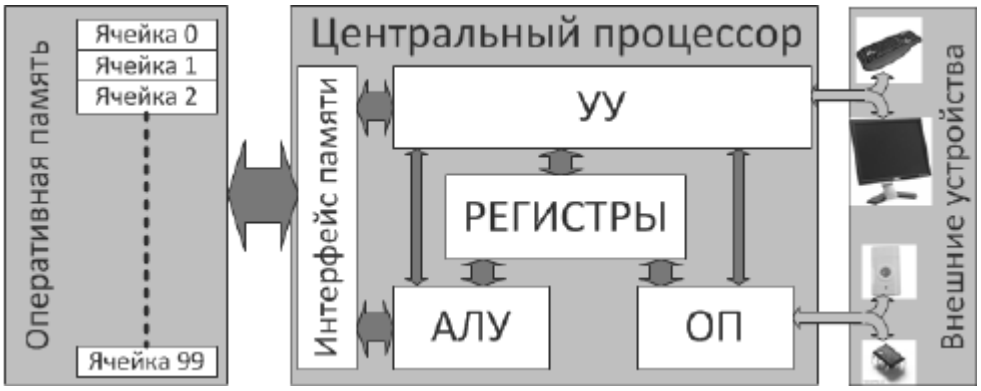


Рисунок 1 – Архитектура вычислительной машины Simple Computer

Оперативная память

Оперативная память – это часть *Simple Computer*, где хранятся программа и данные. Память состоит из ячеек (массив), каждая из которых хранит 15 двоичных разрядов. Ячейка – минимальная единица, к которой можно обращаться при доступе к памяти. Все ячейки последовательно пронумерованы целыми числами. Номер ячейки является еѐ адресом и задается 7-миразрядным числом.

Внешние устройства

Внешние устройства включают: клавиатуру и монитор, используемые для взаимодействия с пользователем, системный таймер, задающий такты работы *Simple Computer* и кнопку «*Reset*», позволяющую сбросить *Simple Computer* в исходное состояние.

Центральный процессор

Выполнение программ осуществляется центральным процессором *Simple Computer*. Процессор состоит из следующих функциональных блоков:

- регистры (аккумулятор, счетчик команд, регистр флагов);
- арифметико-логическое устройство (АЛУ);
- управляющее устройство (УУ);
- обработчик прерываний от внешних устройств (ОП);
- интерфейс доступа к оперативной памяти.

Регистры являются внутренней памятью процессора. Центральный процессор *Simple Computer* имеет: аккумулятор, используемый для временного хранения данных и результатов операций, счетчик команд, указывающий на адрес ячейки

памяти, в которой хранится текущая выполняемая команда и регистр флагов, сигнализирующий об определѐнных событиях. Аккумулятор имеет разрядность 15 бит, счетчика команд – 7 бит. Регистр флагов содержит 5 разрядов: переполнение при выполнении операции, ошибка деления на 0, ошибка выхода за границы памяти, игнорирование тактовых импульсов, указана неверная команда.

Арифметико-логическое устройство (англ. arithmetic and logic unit, *ALU*) — блок процессора, который служит для выполнения логических и арифметических преобразований над данными. В качестве данных могут использоваться значения, находящиеся в аккумуляторе, заданные в операнде команды или хранящиеся в оперативной памяти. Результат выполнения операции сохраняется в аккумуляторе или может помещаться в оперативную память. В ходе выполнения операций АЛУ устанавливает значения флагов «деление на 0» и «переполнение».

Управляющее устройство (англ. control unit, *CU*) координирует работу центрального процессора. По сути, именно это устройство отвечает за выполнение программы, записанной в оперативной памяти. В его функции входит: чтение текущей команды из памяти, ее декодирование, передача номера команды и операнда в АЛУ, определение следующей выполняемой команды и реализации взаимодействий с клавиатурой и монитором. Выбор очередной команды из оперативной памяти производится по сигналу от системного таймера. Если установлен флаг «игнорирование тактовых импульсов», то эти сигналы устройством управления игнорируются. В ходе выполнения операций устройство управления устанавливает значения флагов «указана неверная команда» и «игнорирование тактовых импульсов».

Обработчик прерываний реагирует на сигналы от системного таймера и кнопки «Reset». При поступлении сигнала от кнопки «Reset» состояние процессора сбрасывается в начальное (значения всех регистров обнуляется и устанавливается флаг «игнорирование сигналов от таймера»). При поступлении сигнала от системного таймера, работать начинает устройство управления.

Система команд Simple Computer

Получив текущую команду из оперативной памяти, устройство управления декодирует ее с целью определить номер функции, которую надо выполнить и операнд. Формат команды следующий

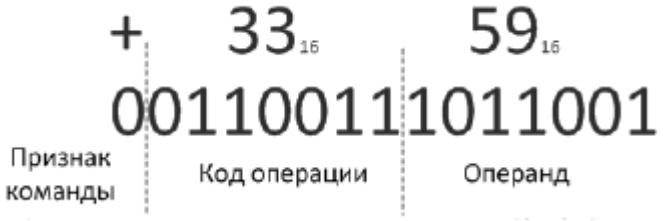


Рисунок 2 – Формат команды центрального процесса Simple Computer

Таблица команд центрального процессора Simple Computer

Операция		Значение
Обозначение	Код	
Операции ввода/вывода		
READ	10	Ввод с терминала в указанную ячейку памяти с контролем переполнения
WRITE	11	Вывод на терминал значение указанной ячейки памяти
Операции загрузки/выгрузки в аккумулятор		
LOAD	20	Загрузка в аккумулятор значения из указанного адреса памяти
STORE	21	Выгружает значение из аккумулятора по указанному адресу памяти
Арифметические операции		
ADD	30	Выполняет сложение слова в аккумуляторе и слова из указанной ячейки памяти (результат в аккумуляторе)
SUB	31	Вычитает из слова в аккумуляторе слово из указанной ячейки памяти (результат в аккумуляторе)
DIVIDE	32	Выполняет деление слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе)
MUL	33	Вычисляет произведение слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе)
Операции передачи управления		
JUMP	40	Переход к указанному адресу памяти
JNEG	41	Переход к указанному адресу памяти, если в аккумуляторе находится отрицательное число
JZ	42	Переход к указанному адресу памяти, если в аккумуляторе находится ноль
HALT	43	Останов, выполняется при завершении работы программы

Консоль управления

Консоль управления содержит следующие области:

1. Memory – содержимое оперативной памяти Simple Computer.
2. Accumulator – значение, находящееся в аккумуляторе;
3. instructionCounter – значение регистра «счетчик команд»;

4. Operation – результат декодирования операции;
5. Flags – состояние регистра флагов («P» - переполнение при выполнении операции,

«O» - ошибка деления на 0, «M» - ошибка выхода за границы памяти, «T» - игнорирование

тактовых импульсов, «E» - указана неверная команда);

6. Cell – значение выделенной ячейки памяти в области Memory (используется для редактирования);
7. Keys – подсказка по функциональным клавишам;
8. Input/Otput – область, используемая *Simple Computer* в процессе выполнения программы для ввода информации с клавиатуры и вывода еѐ на экран.

Содержимое ячеек памяти и регистров центрального процессора выводится в декодированном виде. При этом, знак «+» соответствует значению 0 в поле «признак команды», следующие две цифры – номер команды и затем операнд в шестнадцатеричной системе счисления.

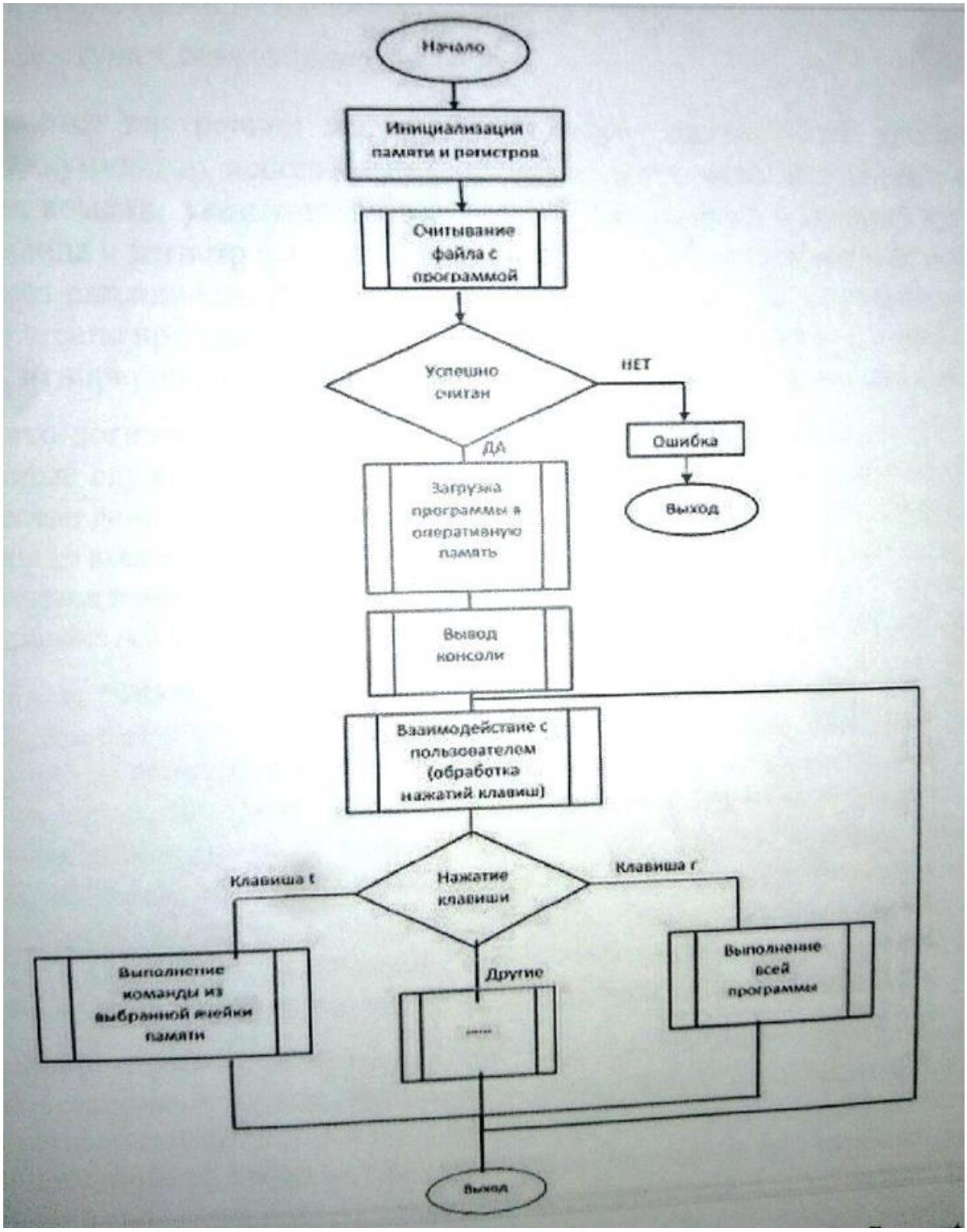
Пользователь имеет возможность с помощью клавиш управления курсора выбирать ячейки оперативной памяти и задавать им значения. Нажав клавишу *F5*, пользователь может задать значение аккумулятору, *F6* – регистру «счетчик команд». Сохранить содержимое памяти (в бинарном виде) в файл или загрузить его обратно пользователь может, нажав на клавиши «*l*», «*s*» соответственно (после нажатия в поле *Input/Output* пользователю предлагается ввести имя файла). Запустить программу на выполнение (установить значение флага «игнорировать такты таймера» в 0) можно с помощью клавиши *r*. В процессе выполнения программы, редактирование памяти и изменение значений регистров недоступно. Чтобы выполнить только текущую команду пользователь может нажать клавишу *t*. Обнулить содержимое памяти и задать регистрам значения «по умолчанию» можно нажав на клавишу *i*.

3. Проделанная работа

- ✓ Постановка задачи исследования

Необходимо доработать модель Simple Computer так, чтоб она выполняла команды, записанные в оперативной памяти (для чего создается обработчик команд). Для управления и визуализации модели требуется создать консоль. Для разработки программ нужно создать трансляторы с языков Simple Assembler и Simple Basic.

- ✓ Блок схема



✓ Программная реализация

Для разработки простейшей вычислительной машины Simple Computer были разработаны следующие функции:

- **sc_memory.c, sc_command.c, sc_register.c**

int sc_memoryInit () – инициализирует оперативную память Simple Computer, задавая всем еѐ ячейкам нулевые значения;

int sc_memorySet (int address, int value) – задает значение указанной ячейки памяти как value;

int sc_memoryGet (int address, int * value) – возвращает значение указанной ячейки памяти в value;

int sc_memorySave (char * filename) – сохраняет содержимое памяти в файл в бинарном виде;

int sc_memoryLoad (char * filename) – загружает из указанного файла содер-жимое оперативной памяти (используя функцию read или fread);

int sc_regInit (void) – инициализирует регистр флагов нулевым значением;

int sc_regSet (int register, int value) – устанавливает значение указанного регистра флагов;

int sc_regGet (int register, int * value) – возвращает значение указанного флага;

int sc_commandEncode (int command, int operand, int * value) – кодиру-ет команду с указанным номером и операндом и помещает результат в value;

int sc_commandDecode (int value, int * command, int * operand) – деко-дирует значение как команду Simple Computer.

- **mt.c**

int mt_clrscr (void)- производит очистку и перемещение курсора в левый верхний угол экрана;

int mt_gotoXY (int, int) - перемещает курсор в указанную позицию;

int mt_getscreensize (int * rows, int * cols) - определяет размер экрана терминала

int mt_setfgcolor (enum colors) - устанавливает цвет последующих выводимых символов;

int mt_setbgcolor (enum colors) - устанавливает цвет фона последующих выводимых символов.

- **bc.c**

int bc_printA (char * str) - выводит строку символов с использованием дополнительной кодировочной таблицы;

int bc_box(int x1,int y1,int x2,int y2)-выводит на экран псевдографическую рамку;

int bc_printbigchar (int [2], int x, int y, enum color, enum color) - выводит на экран "большой символ" размером восемь строк на восемь столбцов;

int bc_setbigcharpos (int * big, int x, int y, int value) - устанавливает значение знакоместа "большого символа" в строке x и столбце y в значение value;

int bc_getbigcharpos(int * big, int x, int y, int *value) - возвращает значение позиции в "большом символе" в строке x и столбце y;

int bc_bigcharwrite (int fd, int * big, int count) - записывает заданное число "больших символов" в файл. Формат записи определяется пользователем;

int bc_bigcharread (int fd, int * big, int need_count, int * count) считывает из файла заданное количество "больших символов".

- **rk.c**

int rk_readkey (enum keys *) - анализирующую последовательность символов и возвращающую первую клавишу, которую нажал пользователь;

int rk_mytermsave (void) - сохраняет текущие параметры терминала;

int rk_mytermrestore (void) - восстанавливает сохранѐнные параметры терминала.

int rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint) - переключает терминала между режимами.

- **cpu.c**

void CU( ) - обеспечивает работу устройства управления.
int ALU(int command, int operand) - реализует алгоритм работы арифметико-логического устройства.

- **print.c**

void printLine(int ctr) – рисует линию цвета фона, закрашивая необходимую область.
void refreshGuiSt(int position) – выполняет отрисовку консоли Simple Computer.
void refreshGui(int position) – перерисовывает изменчивые области консоли Simple Computer.
void printAccum( ) – выводит в значение аккумулятора.
void printBoxes( ) – рисует все рамки.
void printCounter( ) – выводит значение счетчика команд.
void printKeys(int x, int y) – выводит пояснение по активным кнопкам.
void printLabels( ) – выводит все подписи.
void printOperation(int position) – выводит результат декодирования операции ячейки, адресуемой счетчиком команд
void printFlags(int x, int y) – выводит регистр флагов.
int printMcell(int pos) – рисует большими буквами содержимое ячейки памяти, на которую указывает курсор.
void printMemory(int x, int y, int position) – выводит содержимое памяти.

- **handlers.c**

void timerHand(int sig) – обработчик сигнала.
void ursignalHand(int sig) – ставит консоль в исходное положение, сбрасывая все настройки.
void ursignalHand2(int sig) – останавливает таймер и активирует флаг игнорирования тактовых импульсов.
void killHand(int sig) – деактивирует все сигналы, очищает экран.

- **change.c**

void frequencyGenerator (int status) – запускает или останавливает таймер.
void setSignals( ) – считывает набор сигналов.
void signalsRestore( ) – возвращает сигналы в начальное состояние.
void setIgnoreAlarm( ) – задает параметры игнорирования сигналов.
void restoreIgnoreAlarm( ) – возвращает параметры игнорирования сигналов в исходное состояние.
void setEchoRegime( ) – устанавливает режим, который отображает все символы по мере их набора.
void restoreEchoRegime( ) – убирает режим, который отображает все символы по мере их набора.
int changeCell(int pos) – обрабатывает введенное пользователем значение ячейки оперативной памяти.
int changeAccumulator(int pos) - обрабатывает введенное пользователем значение аккумулятора.
int changeInstructionPointer(int pos) - обрабатывает введенное пользователем значение счетчика команд int scanNum(int *plusFlag, int *n) – считывает введенное пользователем значение.
int memorySave(int position) – выполняет сохранение оперативной памяти в файл.
int memoryLoad(int position) – выполняет загрузку из файла в оперативную память.

- **command.c**

int commandHandler(int command, int operand) – выполняет команду, соответствующую поданному коду.

- **write.c**

void writeChar(int fd, const char* str) – выводит строку символов в файл.
int writeInt(int std, int num, int radix, int znac) – выводит число в файл.
int swriteInt(char* buff, int num, int radix, int znac) – считывает число в строку.

- **readInt.c**

int myPow(int a, int b) – возводит число a в степень b.
int sreadInt(char* buffers, int* num, int radix) – переводит строку в число.

- **asm.c**

int asmTrans(int argc, char** argv) – транслирует файл с языка simple assembler на язык simple computer.
int testArgv(char *argv[]) – выполняет проверку открытых файлов.
int asmCommand(char *str) – возвращает код команды.

int strToCommand(char* ptr, int* value) – осуществляет преобразование строки в значение.
int parsingLine(char* str, int* addres, int* value) – обрабатывает строку, получая из нее адрес и значение.

- **basic.c**

volatile int keywordCode(char *str) – возвращает ключ команды строки.
int testArgvB(char *argv[]) - выполняет проверку открытых файлов
int testFile(char* filename) - выполняет проверку файла на совместимость с simple basic.
int basicTrans(int argc, char *argv[]) - Транслирует файл с языка simple basic на язык simple assembler.
int parsingLineB(char* str, int output) – преобразовывает simple basic в simple assembler.
int ifoperation(int output, char* op, int let) – считывает знак и преобразовывает его в набор команд simple assembler.

- **main.c**

int main( ) – осуществляет работу simple computer.

4. Результаты проведенного исследования

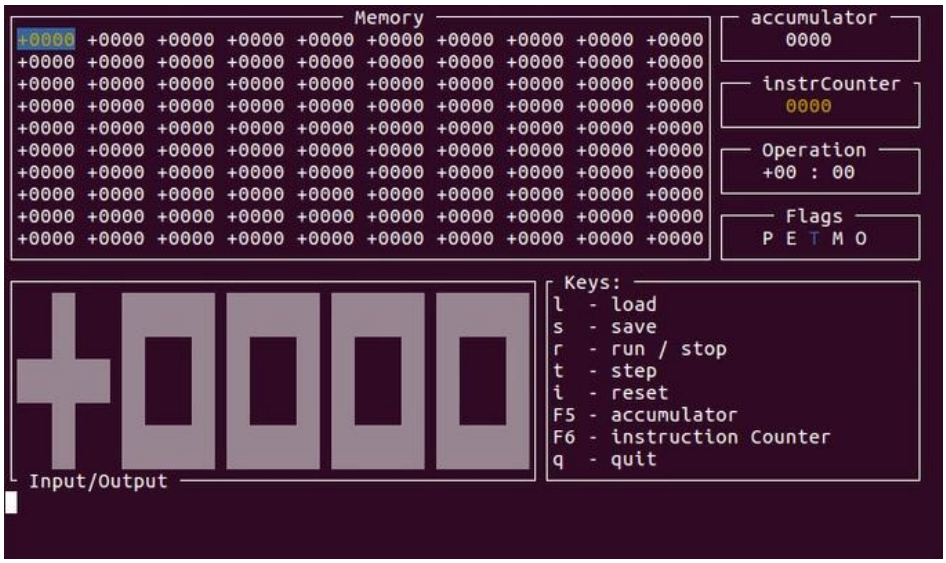


Рис.3. Начальная стадия simple computer

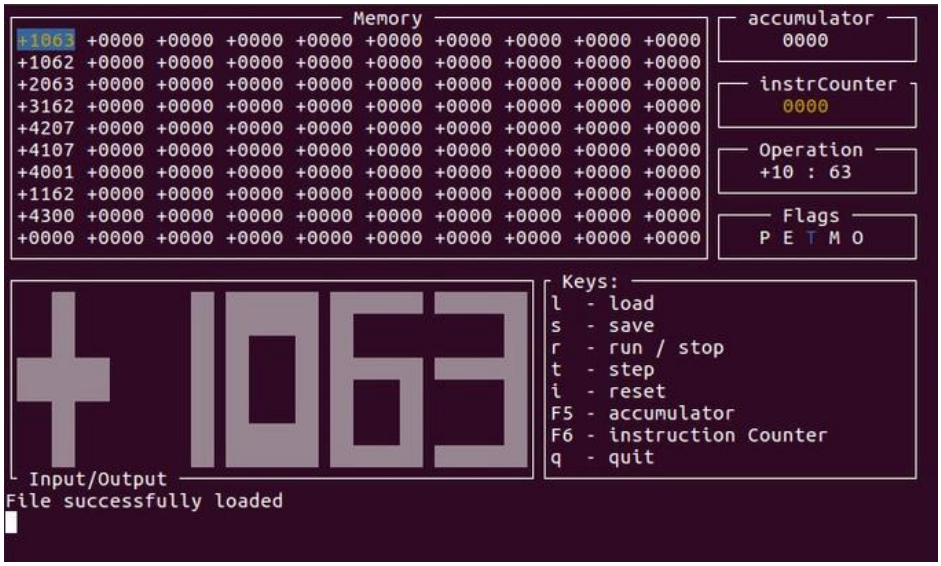


Рис.4. Simple computer после загрузки в него программы

Текст программы на simple basic:

```
00 REM REM1
10 REM REM2
20 INPUT A
30 INPUT B
40 IF A > B GOTO 30
50 OUTPUT B
60 END
```

Текст программы на simple assembler:

```
00 READ 99
01 READ 98
02 LOAD 99
03 SUB 98
04 JZ 07
05 JNEG 07
06 JUMP 01
07 WRITE 98
06 JUMP 01
08 HALT 00
```

Выводы

1. Для разработки приложения simple computer потребовалось узнать о не каноничном режиме работы терминала, описать множество функций, которые обрабатывают созданную оперативную память, задают разные режимы терминала, обрабатывают сигналы и др.
2. Для реализации трансляции с simple assembler и simple basic необходимо несколько функций с множеством условных переходов и проверок.
3. Была разработана модель простейшей вычислительной машины simple computer.

Список используемой литературы

1. Организация ЭВМ и систем. Практикум // С.Н. Мамойленко, Новосибирск: ГОУ ВПО «СибГУТИ», 2005 г., URL:
2. Архитектура компьютера. 4-е изд. // Э. Танненбаум. – СПб.: Питер, 2003.
3. Организация ЭВМ. 5-е изд. / К. Хамахер, З. Вранешич, С. Заки. – СПб.: Питер; Киев: Изда-тельская группа BHV, 2003.
4. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: учебник для ВУЗов. – СПб.: Питер, 2004.