

Факультет Информационных технологий

Кафедра Информационных систем

Направление 09.03.03 Прикладная информатика

Профиль Прикладная информатика в экономике

**Курсовая работа**

По дисциплине: Проектирование информационных систем

**На тему: «Разработка эмулятора портативного калькулятора на собственной виртуальной машине»**

**Студента** Шорникова Егора Евгеньевича

**Группа** УСс 23.1/Б1-17

Проверил:

Зайцев М.А.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2019

# Введение.

Я, как студент, и в некотором роде молодой ученый, стремлюсь понять самую суть моей профессии (программист), мне интересно понять, как работает компьютер с точки зрения микроэлектроники и низкоуровневое программирования. Изучать что-то только в теории, для меня слишком скучно, поэтому требуется реальное железо или его эмуляция, что на данный момент вызывает множество проблем.

В настоящее время, существует ограниченное количество архитектур компьютеров, которые сами по себе очень сложны и не поддаются изучению до мельчайших подробностей студенту, вроде меня. Оперативная память в основном DDR3-4 которая работает на столь высоких частотах, что её невозможно изучить при помощи каких-то дешевых микроконтроллеров (Atmel, Stm-32 и тд), а собрать систему из ПЛИС и пытаться реализовать память на ней, слишком сложно и дорого. Современные процессоры имеют также очень высокие частоты, защиту и тысячи ножек, что делает невозможной, сборку простенькой платы для отладки и программирования на низком уровне.

В недалеком прошлом, в СССР, существовало множество простейших разновидностей персональных компьютеров, со множеством архитектур и своих уникальных возможностей, впоследствии их заменили на клоны IBM PC и широко распространенный в народе Spectrum на процессоре Z80. Студенты могли активно изучать компьютер, и даже собирать его “на коленке” в домашних условиях, за сравнительно небольшие деньги. Также, были программируемые калькуляторы серии МК, которые также, позволяли изучить себя, и были просты в программировании и использовании.

Глядя на всё это, я поставил перед собой цель, создать эмулятор собственного программируемого калькулятора на собственной виртуальной машине, со своим простейшим микропроцессором и архитектурой, с помощью которой, самые молодые умы, смогут прикоснуться к неизведанному, понять азы работы компьютера и программирования на упрощённом ассемблере. Пока что, устройство предполагается создать только в виртуальном виде, а в дальнейшем если это “зайдёт”, собрать реальную физическую копию, используя микроконтроллер для эмуляции процессора (а возможно и ПЛИС).

Задача курсовой: создать эмулятор устройства, для множества платформ, чтобы программировать было удобно где удобно и когда угодно, будь то смартфон или компьютер.

# Оглавление.

[Введение. 2](#_Toc30144067)

[Оглавление. 4](#_Toc30144068)

[Основная часть. 5](#_Toc30144069)

[Выбор платформы и языка разработки. 5](#_Toc30144070)

[Структура готового продукта. 7](#_Toc30144071)

[Язык описания алгоритмов работы виртуальной машины. 8](#_Toc30144072)

[Запись констант. 12](#_Toc30144073)

[Запись ссылок. 13](#_Toc30144074)

[Виртуальная машина 15](#_Toc30144075)

[Архитектура. 15](#_Toc30144076)

[Оперативная память. 16](#_Toc30144077)

[Дисплей. 19](#_Toc30144078)

[Внешний стек 22](#_Toc30144079)

[Финальная версия 23](#_Toc30144080)

[Сборщик кода. 24](#_Toc30144081)

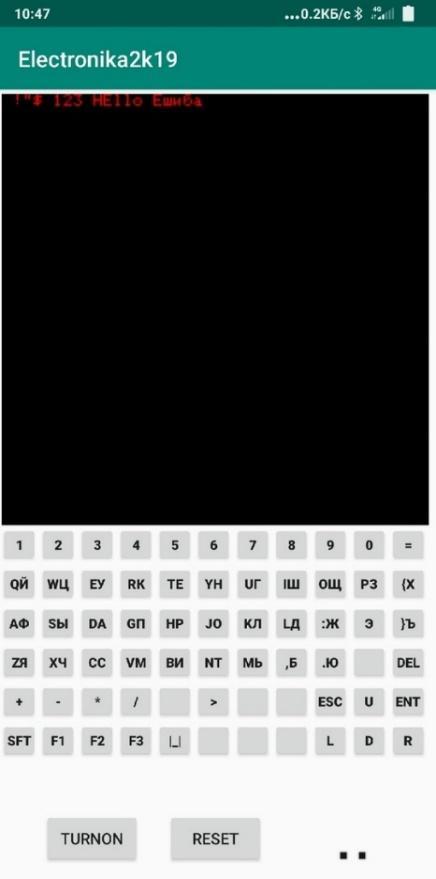
[Заключение. 25](#_Toc30144082)

[Список литературы. 26](#_Toc30144083)

# Основная часть.

## Выбор платформы и языка разработки.

Цель проекта, создать кроссплатформенный эмулятор виртуальной машины, который сможет запуститься на максимально большом количестве устройств, от компьютера до мобильного телефона. Телефон, доступный мне, работает на платформе Android, которая запускает своё программное обеспечение внутри java virtual machine, поэтому изначально язык java был выбран для разработки эмулятора. Был разработан компилятор языка ассемблер в байт код, который в свою очередь запускался ядром виртуальной машины. Также был написан интерфейс для компьютера на библиотеке swing, универсальный для mac os, windows linux и интерфейс для платформы Android на Kotlin (Изображение 1). Но создав всё это, я понял, что не получил новых знаний и умений, что было целью данной курсовой работы. Поэтому начались поиски чего-то нового, языка или платформы, к которым я не прикасался во время основной работы или учёбы. Выбор пал на web разработку, а точнее интерфейс на html+css и ядро на js, что стало новым опытом для меня.



Изображение 1 : “Интерфейс на Kotlin под Android”

Направление проекта стало создание одной веб страницы, которая запускает весь код из единой js вставки, без развёртывания веб сервера и прочего, вроде nodejs и Apache.

Система должна запускаться по клику на \*.html файле и включать в себе всё необходимое.

В последствии я тестировал эмулятор на компьютере, планшете и телефоне.

## Структура готового продукта.

Физически проект состоит из основного electroniks2k19.html который запускает ядро lectroniks2k19.js, а также, стороннюю css библиотеку bootstrap для внешнего вида страницы.

Хотя по изначальной задумке сервера не планировалось, пришлось развернуться простейший http сервер на Python 3.8 HTTPServer. Это простейшая утилита, которая позволила расшить файловую систему проекта по http протоколу и по определённому порту в моей локальной сети, для отладки продукта сразу на нескольких устройствах.

Код сервера выглядит следующим образом:

*from http.server import HTTPServer, CGIHTTPRequestHandler*

*server\_address = ("192.168.68.110", 8000)*

*httpd = HTTPServer(server\_address, CGIHTTPRequestHandler)*

*httpd.serve\_forever()*

В код напрямую записывается ip адрес и протокол, и программа готова к работе.

## Язык описания алгоритмов работы виртуальной машины.

По своей сути, язык описания моей виртуальной машины, это модифицированный под нужные задачи ассемблер.

В качестве источников данных выступают регистры, прямая адресация в память и простое число, а также декларируемые статические переменные. (планируется убрать их в дальнейшем, так как их не должно быть в таком виде).

Для описания команд были использованы следующие сокращения:

RefIn от reference (анг. ссылка) и input (анг. ввод) – точка назначения, операнд или ячейка памяти, куда в результате выполнения команды записываются данные. (при этом нужно учитывать размеры цели назначения при записи).

RefOut от reference (анг. ссылка) и output (анг. вывод) – точка из которой берётся значение, для дальнейших действий.

Всего присутствует 16 команд (таблица 1).

|  |  |  |
| --- | --- | --- |
| *Команда* | *Описание* | *Пример* |
| *MOV* | *Копирует значение из истопника данных в пункт назначения* | *MOV AL, 25; //AL становится = 25* |
| *ADD* | *Прибавляет RefOut к RefIn и записывает в RefIn. Действие команды вклиняет на флаги F0,F1,F2.* | *MOV AL, 2; //AL становится = 2* |
| *ADC* | *Вычитает RefOut из RefIn и записывает в RefIn. Действие команды вклиняет на флаги F0,F1,F2.* | *MOV AL, 10; //AL становится =10;ADC AL, 2; //AL становится = 8* |
| *MUL* | *Умножает RefOut на RefIn и записывает в RefIn. Действие команды вклиняет на флаги F0,F1,F2.2* | *MOV AL, 2; //AL становится = 3;MUL AL, 5; //AL становится = 15* |
| *INC* | *Увеличивает RefIn на 1. Действие команды вклиняет на флаги F0,F1,F2.* | *MOV AL, 1;  //AL становится = 1;INC AL;  //AL становится = 2* |
| *DEC* | *Уменьшает RefIn на 1. Действие команды вклиняет на флаги F0,F1,F2.* | *MOV AL, 1;  //AL становится = 1;INC AL;  //AL становится = 2* |
| *DIV* | *Делит RefOut на RefIn и записывает разность в регистр DL и остаток DH. Действие команды вклиняет на флаги F0,F1,F2.* | *MOV AL, 9;  //AL становится = 9;DIV AL, 5 // DL становится = 1;  DH становится = 4;* |
| *PUSH* | *Отправить значение RefOut в стек* | *MOV AL, 25; // AL становится = 25;PUSH AL;  // Помещаем в стек 25* |
| *POP* | *Взять значение из стека, и записать его в RefIn* | *MOV AL, 25; // AL становится = 25;PUSH AL;  // Помещаем в стек 25;POP AH; // Берём 25 из стека и помещаем его в AH* |
| *CALL* | *Вызов команды записанной по её имени в операнде, текущее состояние регбистов CP, SC записывается в стек.* | *CALL DRAW; // Прыгаем в функцию DRAW состояние регбистов CP, SC записывается в стек.* |
| *CALLF* | *Вызов команды записанной по её имени в операнде, флаг,чей номер записан во второй операнд, включен. текущее состояние регбистов CP, SC записывается в стек.* | *MOV AL, 20; // AL становится = 20;CP AL, AH; // F5 = false;CALLF DRAW, 5; // 5 флаг выключен, поэтому не прыгаем;MOV AH, AL; // AH становится = 20;CP AL, AH; // F5 = true;CALLF DRAW, 5; // 5 флаг включен, прыгаем в функцию DRAW* |
| *RETF* | *Возвращение из функции, которая была вызвана, если флаг,чей номер записан во второй операнд, включен* | *RETF 1;* |
| *RET* | *Возвращение из функции, которая была вызвана* | *RET;* |
| *CP* | *Сравнивает RefIn и RefOut. Действие команды вклиняет на флаги F3,F4,F5.* | *MOV AL, 20; // AL становится = 20;MOV AH, AL; // AH становится = 20;CP AL, 10; // F4 = true, остальные false;CP AL, AH; // F5 = true, остальные false* |
| *RES* | *Инициирует перезагрузку* | *RES* |
| *RESF* | *Инициирует перезагрузку,чей номер записан во второй операнд, включен.* | *RESF 1* |
| *JMP* | *Безусловный переход* | *JMP Main; //Прыгаем в функцию Main, без затрагивания стека* |
| *RAM* | *Прямая запись массива в Оперативную память, с указанием началного адреса* | *RAM 0 [2,1,1,44]; // Записывает в начальный адрес 0, 4 числа в []* |

Таблица 1 - Список команд

### Запись констант.

Константы записываются перед первой функцией программы, записывается сначала команда DECLARE затем имя константы, зачем значение переменной. Можно записать массив и строку;

DECLARE A = 55; //Создаёт константу A со значением 25

DECLARE VAL\_ARR = 45121; //Создаёт константу VAL\_ARR со значением 45121

Его можно записать в 16 битный операнд (AX,BX,CX,DX) а также можно получить одно из чисел,

MOV AL, $VAL\_ARR.[0] - AL становится =4;

DECLARE STR = “Hello Eshiba!” // Создаёт строковую переменную

Можно получить размер строки через MOV AL, $STR.SIZE - AL становится =4;

### Запись ссылок.

Числа пишутся следующим образом *MOV AL, 55; //Число 55*

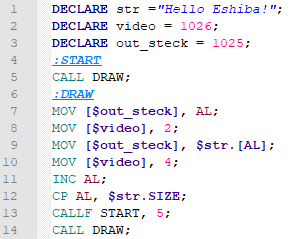
Нельзя писать в число (первым операндом).

Прямая ссылка на память *MOV AL, [1025]* - взять число из ячейки памяти и записать в операнд

Прямая ссылка на память *MOV [1025], [AL]* – взять адрес из операнда AL и записать в ячейку памяти 1025.

2.2. Первая программа - Hello world.

По канонам программирования первой программой сталла hello world (Изображение 2)



Изображение 2 – Hello World

Первым делом идёт объявление констант:

Цель программы вывод строки поэтому объявляем строку str содержащую “hello Eshiba!”.

Затем указываем адреса video команд и выходного стека.

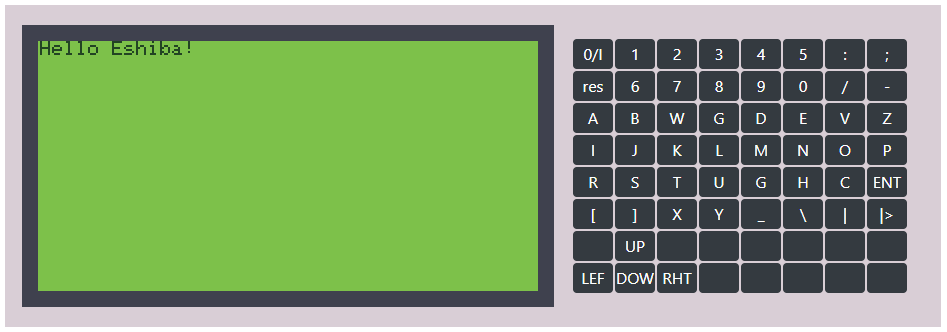
Программа стартует с функции START и сразу же прыгает в функцию DRAW.

Draw должна записать каждый символ на дисплей, выводя каждый символ друг за другом.

Для этого нужно получить номер сивола и поместить его во внешний стек (MOV [$out\_steck], AL), а затем вызвать 2 функцию видео контроллера, что бы записать положение сивола по оси X (MOV [$video], 2), затем получаем сивол мз переменной str и отправляем его в видео контроллер с помощью стека (MOV [$out\_steck], $str.[AL]) и команды 4 (MOV [$video], 4), проверяем вывелена ли вся строка, и если нет запускаем DRAW заново.

CP AL, $str.SIZE; CALLF START, 5;

Результат выполнения программы строка на дисплее (Изображение 3)



Изображение 3 – вывод на дисплей программы Hello World

## Виртуальная машина

## Архитектура.

Для организации памяти между всеми устройствами, я разделил 16 битную шину адреса на шину 12 бит и шину 4 бит. Малая шина предназначена для выборки устройства, всего может быть 16 устройств 0 устройство ROM в текущем эмуляторе не используется, его адреса 0:4095 , следующим идёт RAM память 4096:8191, Далее идёт видеоконтроллер 8192:12287 и в заключении идёт всё остальное (клавиатура, терминальный доступ к пк и тд.)

Всего реализовано 4 устройства, и можно написать ещё множество модулей, но не в данной курсовой.

В основе данной виртуальной машины лежит симулятор простейшего, 8 битного процессора, который имеет 16 битную адресную шину, и 8 битную шину данных. Присутствуют 3 системных регистра PC – Program counter (указывает на адрес текущей функции), CS – steck counter (счётчик стека, указывает на верхний адрес стека), SC – stroke counter (счётчик команд, указывает на адрес текущей команды).

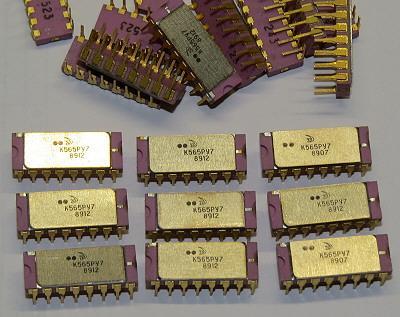
Также существует специализированный регистр, который в настоящем процессоре занимается регенерацией оперативной памяти, и используется программистами для реализации рандомных функций. В нашем случае, на регистре генерируется рандомное значение, а сам он состоит из двух частей (2x - 8 битных регистра) RL,RH которые в свою очередь объединяются в 16 битный регистр RX.

Пользовательских регистров 8 штук AL, AH, BL, BH, CL, CH, DH, DL которые объединяются в AX, BX, CX, DX (Таблица 2)

Цель виртуального процессора - организация менеджмента информации между модулями виртуальной машины, передача данных по 8 битной шине данных.

## Оперативная память.

Виртуальная оперативная память построена на базе реально существующем отечественно модуле памяти К565РУ7 (Изображение 4)



Изображение 4 - ОЗУ К565РУ7

Память создана на базе американского модуля в во второй половине 80-х годов. Данный модуль, был выбран из за своей простоты в конструкции и использовании, память может быть подключена к любому современному микроконтроллеру и имеет всего 16 ножек, 2 из которых питание 5v, а 14 логических.

Память имеет 4 управляющих ноги, WR – чтение, запись, RAS – ножка выборки строки памяти, CAS – выбор столбца, DI – информационный вход

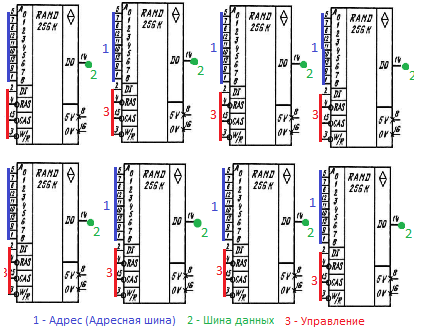
Информационный выход – D0

Память представляет из себя матрицу из 256x256 ячеек по 1 биту, 65536 бит в целом. Для того что бы управлять памятью, нужно совмещать комбинации управляющих сигналов (Таблица 3), также память требует регенерации (если к ячейке не обращаться в течении определённого времени, данные исчезают).



Таблица 3 - Таблица истинности ОЗУ

Так как в данном случае архитектура 8 битная, и шина данных также 8 бит, мы берём 8 модулей памяти и объединяем их шины данных и управления, а каждый выходной пин подключаем к разному адресу шины данных, последовательно. Таким образом, мы получаем память 65536 Байт



Изображение 5 - Схема подключения

Об общей архитектуре компьютера написано выше, шина селекта данных в конкретном (каждом) модуле у нас выходи 8 бит, её напрямую направляем на каждый модуль, для обеспечения их синхронной (одинаковой) работы.

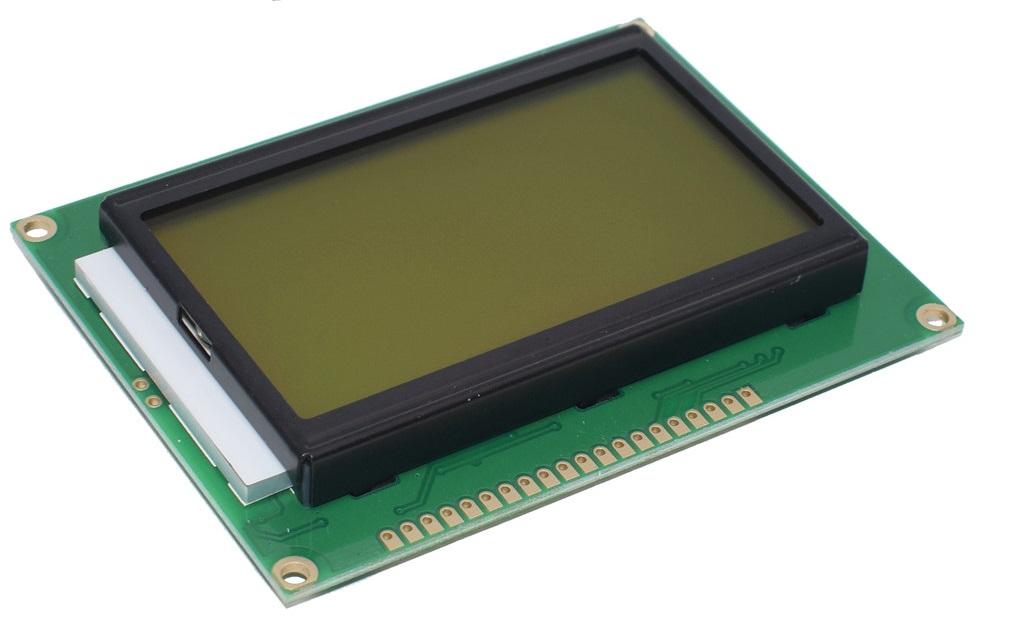
Шины управления также дублируются, а вот выходные пины данных распределяются по шине данных определенным образом, чтобы каждый из выходов направлялся на определённую линию. Схема подключения показана на (Изображение 5).

Но в данной курсовой, память всего лишь виртуальная, по её модели создан класс, который содержит объект K565PY7 включающий в себя матрицу из 256\*256 байт. Управление реализовано при помощи функций Read() и Write(). Также переменную RAM\_LIST, которая отвечает за страницы (строки памяти), а в функции чтения и записи передаётся переменная ADDR симулирующая адрес выбора столбца.

Логический адрес выборки страницы памяти = 4096

Ячейки находятся в адресе 4100 – 4355

## Дисплей.

Дисплей виртуального устройства может быть любым, но в данном проекте , я попытался создать подобие современного китайского модуля ЖК-дисплея 12864 256X128 (Изображение 6) в паре с микроконтроллером, который управляет логику его работы и снижает затраты виртуального процессора на вычисление и требование к пзу, для хранение символов

Изображение 6 -.Модуль 12864

Дисплеи данного модуля, выпускаются в нескольких расцветках, и всегда монохромные. Для задания нужных цветов, я создал отдельные переменные (цвет выключенного дисплея, цвет включенного дисплея с погашенной точкой и цвет включенного дисплея с зажженной точкой).

VIDEO\_COLOR\_TURNOFF = "#878C76";

VIDEO\_COLOR\_TURNON = "#7DC14A";

VIDEO\_COLOR\_TURNACTIVE = "#224423";

Модуль имеет 256 точек по ширине и 128 по высоте. Объём памяти дисплея 4 096‬ байт, при этом каждый байт отвечает за 8 точек на дисплее.

Список команд дисплея -> очистка, записать байт, считать байт. Но в моём случае виртуальной машине необходимо общаться с микрокро контроллером, который подключен к контроллеру дисплея. В микроконтроллер можно передать команду по определённому адресу, и он начнёт выполнять набор команд, забирая данные из внешнего стека.

Команда 1 - Закрасить пиксел (X,Y, VALUE - если > 0 то закрашиваем , если меньше то пустая точка)

Команда 2 - Положение курсора по X, для строковых команд (X)

Команда 3 - Положение курсора по Y, для строковых команд (Y)

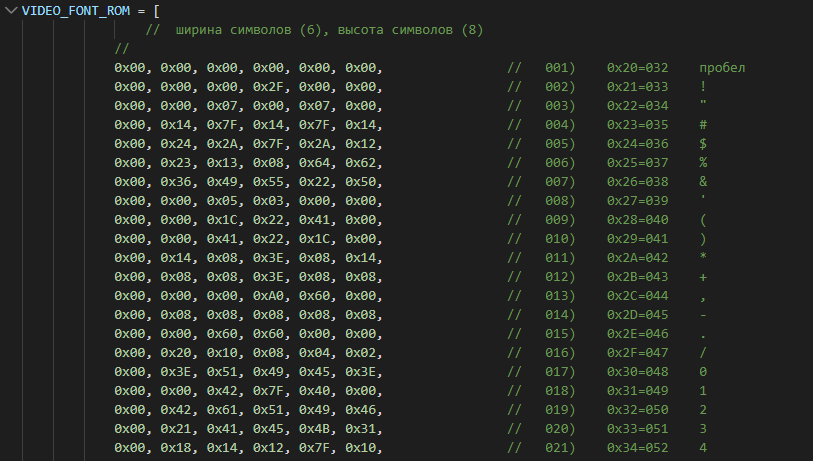
Команда 4 - Отрисовать символ (VALUE)

Команда 5 - не использована

Команда 6 - Reset, очистка дисплея

Команды 2,3 устанавливают переменные высот и ширины в переменных микроконтроллера из которых происходит подсчёт адреса нужного байта для закрашивания определённой точки а команда 1 помещает значение в переменную VALUE из которой также, путём считывания данных из дисплея (для получения текущего состояния точек входящих в нужный байт) рассчитывается новое значение байта и записывается в дисплей.

Перед разбором команды 4, стоит отметить, что в дисплейном модуле (в памяти микроконтроллера) присутствует ПЗУ, хранящая “знакогенератор” в которым лежат спрайты (картинки) каждого символа, который мы будем отрисовывать. В реальности, это массив байтов Изображение 7



Изображение 7 - Бинарный файл знакогенератора

Команда 4 отрисовывает символ на дисплее, она берёт число из внешнего стека. Это число, обозначает порядковый номер символа в нашем знакогенераторе.

Рассмотрим на примере символ ‘\*’ в нашей кодировке это число 11;

Так как каждый символ занимает 6 байт памяти, чтобы получить первый байт от 11, нужно 11\*6=66. Далее получаем все 6 байт от символа

0x00, 0x14, 0x08, 0x3E, 0x08, 0x14, // 011) 0x2A=042 \*

Отрисовка у нас происходит сверху вниз, и выглядит так (Изображение 8):

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x00 | 00000000 |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0x14 | 00010100 |  |  | 1 |  |  |  |  |  |  |
| 0x08 | 00001000 |  |  | 2 |  |  |  |  |  |  |
| 0x3E | 00111110 |  |  | 3 |  |  |  |  |  |  |
| 0x08 | 00001000 |  |  | 4 |  |  |  |  |  |  |
| 0x14 | 00010100 |  |  | 5 |  |  |  |  |  |  |
|  |  |  |  | 6 |  |  |  |  |  |  |
|  |  |  |  | 7 |  |  |  |  |  |  |
|  |  |  |  | 8 |  |  |  |  |  |  |

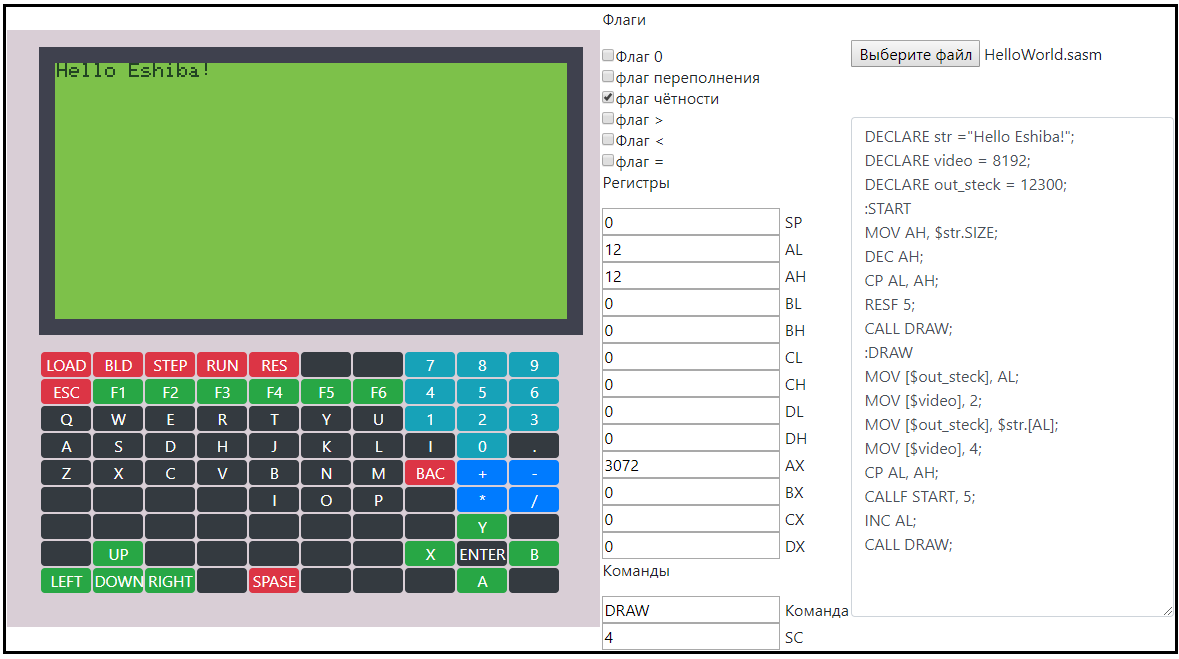
Изображение 8 - Отрисовка символа

## Внешний стек

Для связи всех устройств и организации передачи данных, без прерывания процессора и организации параллельного доступа к озу, был создан внешний стек, который в реальности будет организован на fpga, которая в свою очередь будет организовывать все передачи данных в системе. Любое устройство может занести 1 байт во внешний стек, а также может забрать данные из него. Внешний стек имеет свой определённый адрес. Например, для записи числа на экран, процессор заносит высоту,ширину и сам символ последовательно в стек, и указывает нужные команды в адрес команд дисплея. Также стек можно использовать как более быструю память для буферизации данных, он не требует регенерации как озу, и в целом более надежный. А также будет реализована возможность считать данные из стека, при отладке внешним устройством, когда данный калькулятор будет создан в реальности.

## Финальная версия

По ходу работы, проект множество раз менялся как внутри, так и визуально. Конечная программа выглядит как показано на рисунке 9.



Цифры и кнопки логических операций, а также функциональные кнопки, стали выделены цветом, а также сгруппированы в удобном месте.

Рядом с самим калькулятором, появились индикаторы, отображающие состояние регистров и команду, в которой находится процессор. Также появилась прямая загрузка sasm файлов, хранящих код программы. В дальнейшем это пригодится для загрузки команд из внешних компиляторов.

## Сборщик кода.

Так как идея проекта в том, что я не хочу запускать http сервер, то возможностях nodejs тоже пришлось забыть. Проект сложный, простой структурой кода в 1 файл, обойтись не удалось. Всего в проекте около 10 файлов, и их как то нужно одновременно прогрузить так, чтобы объекты кода знали друг о друге и находились в одной программной среде. В nodejs это решается созданием модулей и подключением их в основной файл, а я решил пойти другим путем.

Для того что бы файлы знали друг о друге, я просто объединяю их в 1, а заодно убираю ненужные комментарии и элементы новой строки. Был написан сборщик js файлов на языке java. Сборщик запускается из командной строки, командой :

java -jar electroniks2k19-jscore.jar > output.log

В команде запуска, я напрямую обращаюсь к jar файлу, и говорю ему, что System.out (системный вывод) должен осуществляться в файлы output.log. В файле системного вывода будут записываться ошибки, если они не возникают, а также список файлов, которые вошли в сборку. Сборщик смотрит в папку srcjs, которая должна лежать в одной директории с ним, он считывает все имеющиеся файлы, а затем считывает файл ignored, где требуется указать имена файлов которые не будут загружены. По итогу, убираются все комментарии написанные чрез ‘//’ и файл собирается в папке на уровень выше чем сборщик, для того чтобы оказаться рядом с нужным \*.html (таким образом я упростил сборку во время разработки, мне достаточно запустить \*.bat (\*.sh) файл для сборки, и новое ядро соберется).

# Заключение.

Задача проекта, была в создании простейшей виртуальной машины, и реализация калькулятора, который можно запрограммировать, на ней. Программный код самого калькулятора в итоге состоит лишь из одной программы Hello world, но цели написать операционную систему или что то в этом роде, собственно не было.

В дальнейшем, вне курсового проекта я обязательно напишу компилятор с++ и простейшую прошивку, а возможно и частично портирую операционную систему от МК или вовсе MS DOS.

Цель проекта я считаю достигнутой, виртуальная машина вариативна и легко программируется, а главное запускаться как на телефоне так и на компьютере, везде где есть интернет браузер Chrome.

Также я ставил перед собой задачу по самообучению языку java script, что я также выполнил. Базовые знания js получены, и началось углублённое изучение языка, вне курсового проекта. Ядро калькулятора на js, в целом функционирует, но имеет недостатки, такие как декларирование переменных и объектов в корне среды, без использования классов и каких то закрытых хранилищ, отсутствует реализация вывода информации и сохранения данных пользователя, но присутствует ввод файла. Не реализована загрузка бинарных файлов, хотя компилятор и примерное ядро на java имеются.

Ядро калькулятора гибкое, и возможно создать любую визуальную оболочку поверх него, а также, путём добавления дополнительных шрифтов, можно добавить языки, или вовсе спрайты для игр, и по одному алгоритму реализовать их вывод. Так что образовательное-развлечение в проекте, может найти каждый, а сам исходный код открыт на GitHub.

# Список литературы.

1. Документация js / <https://developers.google.com/maps/Documentation/javascript/tutorial>

2. Wikipedia / <https://ru.wikipedia.org/wiki/Электроника_МК-90>

3. Программирование в машинных кодах и на языке ассемблера 1993г ./ Изд. «Инфорком»

4. Микропроцессорный комплект Z80. Книга 1 1990г. / Изд. «Центр»

5. Intel: взгляд изнутри 1999г. / Тим Джексон