# Введение.

Я, как студент, и в некотором роде молодой учёный, 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 дешёвых 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.

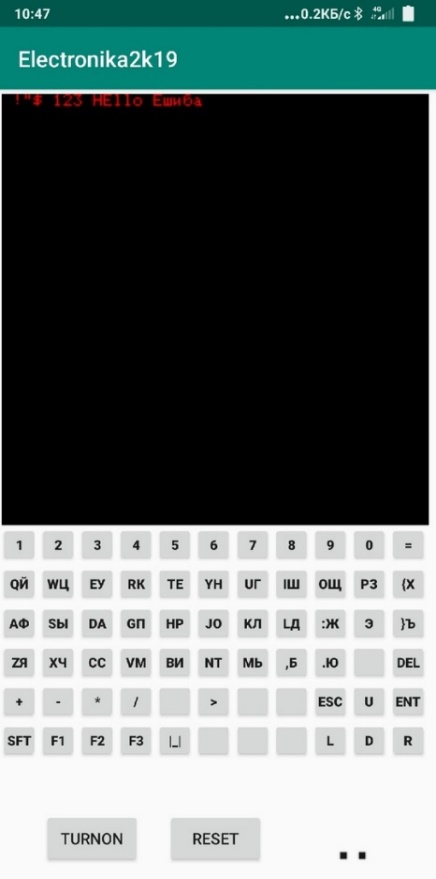
В недалёком прошлом, в СССР, существовало множество простейших разновидностей персональных компьютеров, со множеством архитектур и своих уникальных возможностей, в последствии их заменили на клоны IBM PC и широко распространённый в народе Spectrum на процессоре Z80. Студенты могли активно изучать компьютер, и даже собирать его “на коленке” в домашних условиях, за сравнительно небольшие деньги. Также, были программируемы 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.

## 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 2). 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 2 : “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.

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 10 команд:

MOV – Копирует значение из истопника данных в пункт назначения, мнемоника: MOV RefIn, RefOut

Пример:

*MOV AL, 25; //AL становится = 25*

ADD – Прибавляет RefOut к RefIn и записывает в RefIn. Действие команды вклиняет на флаги F0,F1,F2.

Пример:

*MOV AL, 2; //AL становится = 2*

*ADD AL, 25; //AL становится = 27*

ADC - Вычитает RefOut из RefIn и записывает в RefIn. Действие команды вклиняет на флаги F0,F1,F2.

Пример:

*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

Пример:

*MOV AL, 2; //AL становится = 3*

*MUL AL, 5; //AL становится = 15*

DIV – Делит RefOut на RefIn и записывает разность в регистр DL и остаток DH. Действие команды вклиняет на флаги F0,F1,F2.

Пример:

MOV AL, 9; *//AL становится = 9*

DIV AL, 5 // DL *становится* = 1; DH *становится* = 4;

INC – Увеличивает RefIn на 1. Действие команды вклиняет на флаги F0,F1,F2.

Пример:

MOV AL, 1; *//AL становится = 1*

INC AL; *//AL становится = 2*

DEC – Уменьшает RefIn на 1. Действие команды вклиняет на флаги F0,F1,F2.

Пример:

MOV AL, 1; *//AL становится = 1*

DEC AL; *//AL становится = 0*

СP – Сравнивает RefIn и RefOut. Действие команды вклиняет на флаги F3,F4,F5.

F3 – флаг включен, если результат операции “>”

F4 - флаг включен, если результат операции “<”

F5 - флаг включен, если результат операции “=”

Пример:

*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*

CALL – Вызов команды записанной по её имени в операнде, текущее состояние регбистов CP, SC записывается в стек.

Пример:

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.

Пример:

*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 состояние регбистов CP, SC записывается в стек*

POP – Взять значение из стека, и записать его в RefIn

Пример:

*MOV AL, 25; // AL становится = 25*

*PUSH AL; // Помещаем в стек 25*

*POP AH; // Берём 25 из стека и помещаем его в AH*

PUSH – Отправить значение RefOut в стек

Пример:

*MOV AL, 25; // AL становится = 25*

*PUSH AL; // Помещаем в стек 25*

### 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*

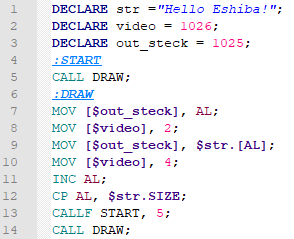
Нельзя писать в число (первым операндом).

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 (рисунок)



Изображение – 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.

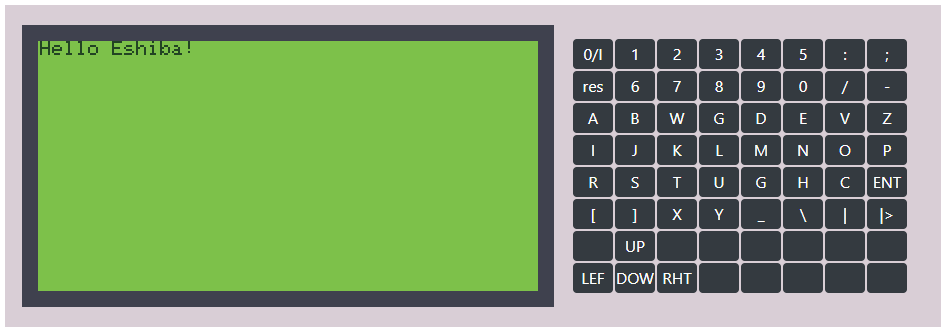
Программа стартует с функции START и сразу же прыгает в функцию DRAW.

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)



Изображение – 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)

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 (рисунок)



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.

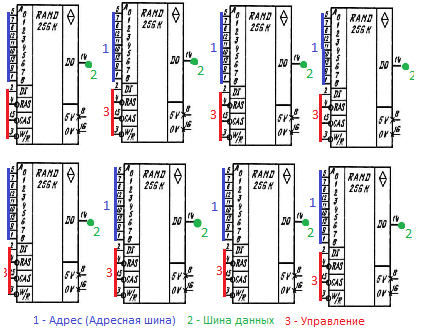
Память имеет 4 управляющих ноги, WR – чтение, запись, RAS – ножка выборки строки памяти, CAS – выбор столбца, DI – информационный вход

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 (картинка), также память требует регенерации (если к ячейке не обращаться в течении определённого времени, данные исчезают).



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



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 определённым образом, что бы каждый из выходов направлялся на определённую линию. Схема подключения показана на (картинка).

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