Санкт–Петербургский государственный университет

ГОРБУНОВ Павел Александрович

Выпускная квалификационная работа

Разработка интерактивного учебного пособия для изучения устройства и работы компьютера

Уровень образования: бакалавриат

Направление 01.03.02 «Прикладная математика и информатика»

Основная образовательная программа СВ.5005.2021 «Прикладная математика, фундаментальная информатика и программирование»

Профиль «Технологии программирования»

Научный руководитель:

доцент кафедры

технологии программирования,

к.ф.–м.н. Сергеев С.Л.

Рецензент:

генеральный директор

Петербургский Капитал,

к.ф.–м.н. Пашкевич В. Э.

Санкт-Петербург,

2025 г.

Содержание

Введение

В современном мире компьютеры имеют значительные вычислительные возможности, но при этом остаются «чёрным ящиком» для студентов, желающих изучить компьютерные науки. Это обусловлено большим количеством уровней абстракций над процессором, оперативной памятью и другими модулями компьютера, что во время обучения ведёт к формальному запоминанию материала. В данной работе предлагается одно из решений данной проблемы – создание интерактивной модели компьютера, которая упростит сложные технические концепции для базового понимания устройства компьютера, что сделает материал наглядным и осязаемым.

Постановка задачи

Цель работы

Разработать и технически обосновать интерактивную аппаратную модель компьютера, демонстрирующую базовые принципы его работы через визуализацию процессов и ручное управление компонентами.

Задачи работы

- Провести анализ существующих образовательных моделей и выявить их ограничения.
- Сформулировать требования к аппаратной реализации учебного пособия.
- Реализовать модель с возможностью ручного управления тактами и визуализацией данных.
- Провести тестирование функциональности модели и оценить её образовательную ценность.

Обзор литературы

При изучении темы устройства компьютера в первую очередь стоит рассмотреть Э.Таненбаума «Архитектура компьютера» [1]. В этих работах содержится системный подход к объяснению взаимодействия модулей и детальное описание организации компьютера: от устройства регистров до уровня операционной системы. Приводятся примеры на реальных процессорах. Для начинающих может ощутиться сложность восприятия материала из-за обилия технических деталей и отсутствия визуализации процессов. В моей работе труды Таненбаума служат теоретической базой для проектирования учебной модели.

В книге Д.Паттерсона и Дж.Хеннесси «Организация и проектирование компьютеров» [2] содержатся принципы проектирования, включая конвейерную обработку. Продемонстрированные практические примеры иллюстрируют теорию и уравновешивают её связь с реальным применением. Это подчёркивает необходимый баланс между теорией и практикой, что учтено в разработке модели.

Также стоит обратить внимание на работу Ч.Петцольда «Код» [3]. Автор поэтапно объясняет работу компьютера «с нуля»: от логических вентилей до ассемблера. Также делает упор на доступный язык и отсутствие сложное математики, что делает книгу доступной для новичков. Однако её главный недостаток в том, что читатель усваивает теорию, но не взаимодействует с «живыми» компонентами. Разрабатываемое учебное пособие устраняет этот пробел и преподносит обучающемуся осязаемый опыт.

Важным дополнением к теоретическим трудам является статья А.Акрам (Ayaz Akram) и Л.Саволха (Lina Sawalha) "A Survey of Computer Architecture Simulation Techniques and Tools" (IEEE Transactions on Education, 2019) [4]. Авторы проводят обзор современных методов и инструментов для симуляции архитектуры компьютеров, включая программные эмуляторы (например, QEMU, Gem5) и аппаратные решения на базе FPGA. В статье подчёркивается, что, несмотря на мощь цифровых симуляторов, физические модели остаются незаменимыми для начального обучения, так как обеспечивают непосредственное взаимодействие с аппаратурой. Это подтверждает необходимость разработки предложенной мною модели.

Глава 1. Обзор существующих решений и обоснование разработки

1.1 Обзор образовательных моделей

Перед тем как начать проектирование и сборку собственной модели необходимо изучить уже существующие образовательные стенды.

Во многих учебных заведениях традиционно тема про внутреннее устройство компьютера преподносится с демонстрацией физического системного блока без части корпуса, что позволяет рассмотреть отдельные модули – процессор, оперативную память, жёсткий диск и так далее. Тем не менее такой подход предоставляет ограниченную информацию – нет возможности проследить, как компьютер выполняет программу, как перемещаются данные по модулям.

У студентов МГТУ им. Н.Э. Баумана имеется возможность не только изучать теорию программирования контроллеров, но и применять знания на практике на учебном стенде ТРЭИ [5]. Такой подход способствует глубокому пониманию принципов автоматизации и дает будущим инженерам возможность реализовывать свои идеи в реальных условиях. С другой стороны, студенты изучают контроллеры как «черные ящики», не погружаясь в работу процессора, памяти, шин данных.

Интернет-портал Учтех-Профи предоставляет возможность заказать лабораторный стенд [6]. Данный стенд предназначен для теоретической и практической подготовки студентов в рамках дисциплины «Архитектура ЭВМ» и позволяет научиться проектировать вычислительные системы, писать

программы на языке VHDL, ассемблер для различных архитектур ЭВМ, а также на языках высокого уровня с ассемблерными вставками, программировать алгоритмы работы с периферийными устройствами через стандартные интерфейсы. Несмотря на широкую функциональность, стенд имеет ряд ограничений – стоимость и высокий порог входа для работы с данной моделью может подойти не для каждого студента.

Также, на интернет-портале kpolyakov.spb.ru в открытом доступе предоставлен бесплатный софт виртуального тренажёра «ЛамПанель» [7] для изучения работы процессора. Это учебная модель компьютера, управляющего ламповой панелью. Он предназначен для проведения практических занятий по теме «Процессор» в школьном курсе информатики. Тренажёр можно использовать: для изучения принципов работы компьютера (процессор, ОЗУ, ПЗУ); для начального изучения программирования на языке ассемблера; для изучения операций с целыми числами, в том числе поразрядных логических операций и сдвигов. Стоит отметить, что даже при таких обширных возможностях виртуального стенда, его абстрактность и отсутствие с ним тактильного взаимодействия может затруднить понимание материальной основы компьютера.

Из открытых проектов также стоит упомянуть восьмибитный компьютер Бена Итера (Ben Eater) [8], который демонстрирует работу процессора на макетных платах. Модель отличается подробной документацией, но её главный недостаток – необходимость самостоятельной сборки, которая может занять десятки часов. Тем не менее, данная модель послужила вдохновением для создания собственного тренажёра, но с некоторыми доработками в виде изменения схем отдельных модулей.

1.2 Обоснование разработки собственной модели

Каждая рассмотренная образовательная модель имеет некоторые недостатки. Классический вариант с системным блоком – это поверхностное знакомство с устройством компьютера, стенд ТРЭИ имеет закрытую архитектуру и высокий порог входа, коммерческие стенды наподобие тех, которые предлагает интернет-портал Учтех Профи, может быть финансово недоступен, виртуальный тренажёр исключает тактильное взаимодействие и наглядность, а модель Бена Итера требует самостоятельной сборки.

Таким образом, разработка собственного интерактивного учебного пособия для изучения устройства и работы компьютера, которое решает упомянутые недостатки существующих моделей, является обоснованным и будет иметь практическую ценность для образовательного процесса.

Глава 2. Разработка и тестирование аппаратной модели

2.1 Требования к модели

Будущая сконструированная модель должна решать недостатки существующих решений, обозримых в первой главе. Для этого сформулируем требования, которые будут предъявлены к конечной модели. Они должны охватывать функциональные, образовательные и экономические аспекты.

Функциональные требования включают возможность ручного ввода данных и выбор адреса в памяти через переключатели, что позволяет преодолеть абстракции и сделать процесс осязаемым. В модели необходим тактовый генератор с регулируемой частотой от 0.2 до 10 Гц. Также необходима возможность посылать «ручные» такты компьютерной модели для отслеживания и анализа выполняемых команд. Помимо всего необходима полная визуализация состояния ячеек памяти и регистров через светодиоды, а также дисплей для вывода данных в десятичном формате.

Образовательные требования ориентированы на обеспечение наглядного и практическое изучение архитектуры компьютера. Модель должна демонстрировать базовые принципы: цикл «выборка-декодирование-выполнение», работу памяти, арифметико-логического устройства, состояние управляющих сигналов. Для обеспечения прозрачности архитектуры компоненты компьютера должны быть спроектированы из логических вентилей и минимального числа интегральных схем со сложной функциональностью.

Экономические требования ограничивают стоимость компонентов учебного стенда до 10000-15000 рублей, использование широко распространённых микросхем для обеспечения ремонтопригодности.

2.2 Архитектура модели

Модель компьютера состоит из независимых модулей и системы управляющих сигналов. Модули взаимодействуют через шину данных, отправляя и считывая данные с неё, что продемонстрировано на рисунке 2.1. Синхронизирует работу всех компонентов тактовый генератор. Разрядность модели – 8 бит.

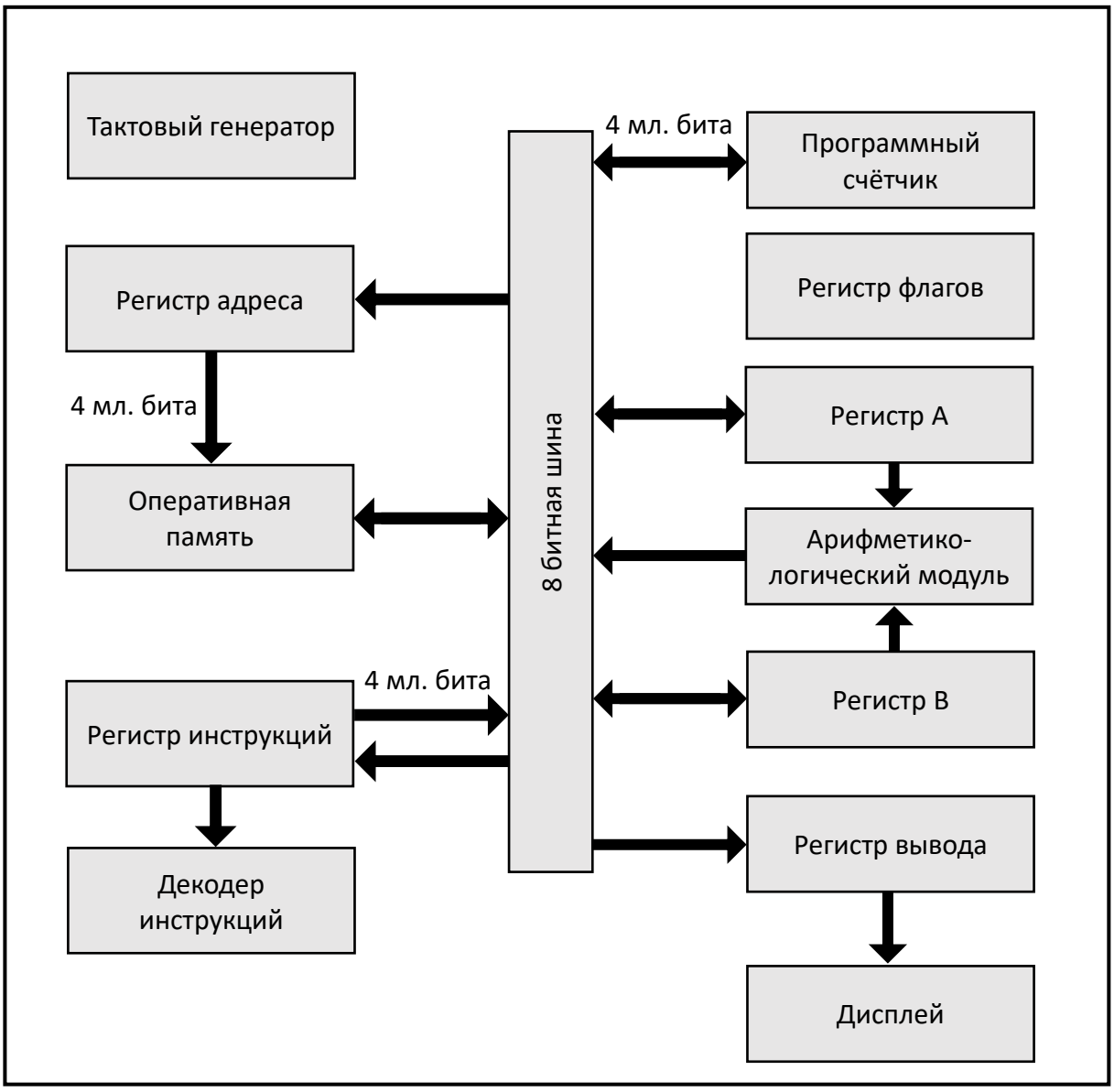

Рис. 2.1: Архитектура модели

В компьютере представлены следующие модули, базирующиеся на микросхемах серии 74 [9]:

- Тактовый генератор - компонент, выполняющий роль метронома, который синхронизирует работу всего компьютера. Его ключевая особенность в том, что можно гибко управлять частотой и режимами работы. Генератор построен на нескольких интегральных схемах NE555, которые позволяют генерировать импульсы засчёт заряда и разряда конденсаторов внутри микросхемы. Модуль имеет два режима работы, которые переключаются по кнопке. В первом режиме генерирует стабильные импульсы, частоты которых можно вручную регулировать с помощью потенциометра на 1MΩ: в крайнем положении с выключенным потенциометром получаем максимальную частоту тактов в 15 Гц, в другой крайнем положении минимальную – 0.25 Гц. С такой возможностью можно замедлить выполнение программы для детального анализа или, наоборот, ускорить для скорейшего выполнение. Во втором режиме тактовый генератор испускает импульсы только по кнопке, что удобно для отладки программ.
- Шина данных - магистраль для передачи данных между модулями. Состоит она из 8 параллельных линий, к которым подключаются все основные модули: к первой полосе подключаются первые биты регистров, оперативной памяти, ко второй полосе второй бит и т.д. Шина заземлена через резисторы на 10кОм для предотвращения неопределённого состояния.
- Регистры A и B – восьмибитные ячейки памяти, способные читать, хранить и писать данные на шину. Эти модули используются для хранения промежуточных данных во время выполнения операций. Реализованы через пару четырёхбитных регистров.

- Арифметико-логический модуль – компонент, показывающий на каждом такте сумму чисел, которые находятся в регистрах A и B. Представляет из себя два четырёхбитных сумматора, бит переполнения первого поступает как дополнительный бит на второй. Вычитание реализовано через обратный код: число в регистре B инвертируется по битам, добавляется единица и складывается с числом в регистре A.
- Регистр флагов – четырёхбитный регистр, который хранит информацию о последней операции, выполненной в арифметико-логическом модуля. В первом бите содержится флаг нуля (ZF), который выставляется в значении 1, если в АЛУ число 0, и в значении 0 в противном случае. Во втором бите находится флаг переполнения (CF), который выставляется в 1, если на последней операции в АЛУ был бит переполнения в старшем четырёхбитном сумматоре, и выставляется в 0 иначе.
- Программный счётчик – восьмибитный регистр памяти со счётчиком. Она содержит номер следующей команды, которую выполнит компьютер. При старте программы содержит нулевое значение. Непосредственно счётчик реализован через микросхему JK flip-flop.
- Регистр вывода – восьмибитный регистр, способный читать данные с шины и хранить их перед отображением на дисплее. Для экономии схемы вывода и использования одного ПЗУ вместо четырёх применяется метод мультиплексирования. Этот метод заключается в том, что последовательно на каждом такте локального генератора производится вывод единиц, на следующем такте десяток и т.д. При выводе каждого разряда активируется только один из четырёх дисплеем, на котором ожидается цифра. Частота тактов локального генератора стабильна и равно 120 Гц - каждый разряд обновляется 30 раз в секунду, что устраняет мерцание.

- Оперативная память – энергозависимая память, которая необходима для хранения кода программы и других данных во время работы компьютера. Модуль реализован с помощью двух микросхем по 8 байт каждая, что в сумме даёт 16 байт памяти. Адрес ячейки, к которой хотим обратиться, берётся из регистра адреса памяти.
- Регистр адреса памяти – четырёхбитный регистр, который временно хранит адрес ячейки памяти, к которой обращается программа при чтении или записи в память. В модели оперативная память представлена 16 ячейками по 8 бит каждая, поэтому 4 бита регистра адреса памяти как раз адресуют 2^4 = 16 ячеек.
- Регистр инструкций – восьмибитный регистр, который хранит команду, выполняемую компьютером в данный момент, а также поставляет её в декодер команд. Каждая восьмибитная команда разделена на две части: старшие четыре бита определяют код операции, действие, а младшие четыре биты – аргумент команды. Регистр считывает с шины восемь бит данных, четыре старших отправляет в декодер команд, а младшие обратно на шину для дальнейшего перемещение в другие модули.
- Декодер инструкций - модуль, который отображает четыре бита, полученных из регистра инструкций, в соответствующую команду, которая является набором управляемых сигналов для модулей компьютера. Компонент реализован на базе EEPROM, запрограммированного как ПЗУ с помощью Arduino.

2.3 Логика работы модели

Учебная модель базируется на архитектуре фон Неймана. В этом классическом цикле команд выделяется три основных этапа: выборка

команды, её декодирование и выполнение. Рассмотрим каждый этап поподробнее:

1. Выборка команды:
    - Программный счётчик передаёт номер инструкции в регистр адреса памяти и увеличивает счётчик на единицу.
    - Оперативная память по адресу возвращает данные, которые помещаются в регистр инструкций.
    - Регистр инструкций старшие четыре бита передаёт в декодер инструкций.

2. Декодирование:
    - Декодер инструкций сопоставляет четырём полученным битам определённую команду с тремя микроинструкциями, каждая из которых является набором управляющих сигналов.

3. Выполнение:
    - Счётчик декодера инструкций по порядку проходит по микроинструкциям и посылает соответствующие управляющие сигналы во все модули компьютера.
    - Счётчик декодера и тактовый генератор компьютера работают в противофазе: на такт счётчика декодера посылаются управляющие сигналы, на такт главного генератора активируются все модули компьютера и выполняют действия на основе установленных управляющих сигналов.

Каждый модуль рассчитан выполнять некоторые действия при наличии восходящего такта компьютера. Какое из доступных действий совершится,

контролируют управляющие сигналы. Доступные управляющие сигналы описаны в таблице 2.1.

Модуль компьютера	Управляющие сигналы
Тактовый генератор	**HLT** – остановка тактов
Регистр адреса памяти	**MI** – считать данные с шины в регистр
Оперативная память	**RI** – считать данные с шины в ячейку памяти **RO** – вывести данные из ячейки памяти на шину
Регистр инструкций	**IO** – вывести младшие четыре бита (аргумент команды) на шину **II** – считать данные с шины в регистр
Регистр A	**AI** – считать данные с шины в регистр **AO** – вывести данные из регистра на шину
Арифметико-логический модуль	**EO** – вывести данные из АЛУ на шину **SU** – активировать режим вычитания
Регистр B	**BI** – считать данные с шины в регистр
Регистр вывода	**OI** – считать данные с шины в регистр
Программный счётчик	**CE** – добавить +1 к счётчику на следующем такте **CO** – вывести номер команды на шину **J** – перепрыгнуть на номер команды с шины
Регистр флагов	**FI** – заполнить регистр флагов данными

Таблица 2.2: Управляющие сигналы модулей компьютера

2.4 Ключевые архитектурные решения

При проектировании сознательно упрощена архитектура модели по сравнению с современным компьютером, чтобы легче преподнести базовые принципы вычислительных систем для начинающих.

- Отказ от кэш-памяти – позволяет упростить процесс обращения к памяти и устранить сложные алгоритмы кэширования и виртуальной памяти в пользу прямого обращения в память по заданному адресу без промежуточных шагов.
- Выполнение команд последовательно, без конвейеризации – жертвуем производительностью ради наглядности.
- Использование микросхем 74LS245 (октальный буфер с тристабильным выходом) – данное решение позволяет всегда выводить данные с регистров, чтобы визуализировать их через светодиоды, а далее направлять в тристабильный буфер, который уже контролирует, читать или писать данные на шину.
- 8-битная разрядность – ограничение шины данных до восьми бит, вместо 32 или 64, позволяет многократно упростить схемы, сделать их менее перегруженными и визуально понятнее.
- Использование одной шины для данных и адресов – позволяет упростить конструкцию и сделать модель нагляднее.

Таким образом, проектируемая модель компьютера — это основы, которые могут быть развиты для моделирования современного компьютера.

2.5 Реализация интерфейса

Интерфейс учебного пособия спроектирован под максимальную визуализацию и интерактивность, позволяя управлять системой и наблюдать состояние в реальном времени.

1. Визуализация:

- В учебной модели каждый светодиод визуализирует бит данных, адреса или управляющий сигнал, что позволяет анализировать их состояние. Горящий светодиод означает логическую единицу в бите, не горящий – логический ноль. Светодиодами оснащены все регистры, шина данных, панель управляющих сигналов, тактовый генератор и счётчик микроинструкций декодера команд.
- В любой момент есть возможность вывести данные в десятичном формате на дисплей.

2. Управление:

- Переключатель режима тактирования позволяет выбирать режим стабильных и ручных тактов.
- Кнопка ручного такта генерирует тактовый импульс, если модуль генератора находится в ручном режиме.
- Переключатель режима работы оперативной памяти позволяет выбрать между обычным и режимом программирования. В первом случае модуль читает и пишет данные из памяти и шины. Во втором читает данные с переключателей, и по кнопке записывает данные по адресу, который тоже задаётся пользователем через переключатели.
- Кнопка сброса очищает данные в регистрах и устанавливает программный счётчик на 0.
- Переключатель в регистре вывода позволяет выбирать интерпретацию данных между знаковыми и беззнаковыми числами.

2.6 Реализация системы команд

Вычитывая команду из оперативной памяти, компьютер должен понимать, что он должен сделать. Для этого он выгружает её из оперативной инструкцию в декодер команд, тот в свою очередь ставит в соответствие инструкции управляющие сигналы. Эти сигналы активируют соответствующие модули, которые выполняют требуемое действие.

Для удобства программирования компьютера (записи команд в оперативную память) можно ввести абстракцию в виде человекочитаемых команд. Она позволяет писать программу на языке, близком к ассемблеру, не задумываясь об управляющих сигналах. Затем по таблице преобразовать эти команды в 0 и 1 и записать в оперативную память в режиме программирования.

Для данной модели был придуман язык, описанный в таблицах 2.2 и 2.3.

Название команды	**Описание команды**
FETCH	Подготовильное действие перед каждой командой. Вычитывание команды из оперативной памяти и передача в декодер инструкций.
NOP	Бездействие.
LDA	Загрузить в регистр A данные из памяти по адресу аргумента.
ADD	Загрузить в регистр B данные из памяти по адресу аргумента, сложить с тем, что в регистре A. Результат сложения записать в регистр A.
SUB	Режим вычитания ALU.
STA	Сохранить содержимое регистра A в память по адресу аргумента.
LDI	Загрузить аргумент в регистр A (аргумент четырёхбитный).
JMP	Установить значение программного счётчика значением аргумента.
JC	Установить значение программного счётчика значением аргумента, если флаг CF=1.

JZ	Установить значение программного счётчика значением аргумента, если флаг ZF=1.
OUT	Вывести значение регистра A на дисплей.
HLT	Остановить компьютер.

Таблица 2.2: *Команды языка компьютера и их описание*

Название команды	**Код команды**	**Шаг**	**Состояние флагов CF ZF**	**Состояние управляющих сигналов HLT MI | RI RO| IO II | AI AO EO SU | BI OI | CE CO | J FI**
FETCH	XXXX	000	XX	01 00 00 00 00 00 01 00
	XXXX	001	XX	00 01 01 00 00 00 10 00
NOP	0000	010	XX	00 00 00 00 00 00 00 00
	0000	011	XX	00 00 00 00 00 00 00 00
	0000	100	XX	00 00 00 00 00 00 00 00
LDA	0001	010	XX	01 00 10 00 00 00 00 00
	0001	011	XX	00 01 00 10 00 00 00 00
	0001	100	XX	00 00 00 00 00 00 00 00
ADD	0010	010	XX	01 00 10 00 00 00 00 00
	0010	011	XX	00 01 00 00 00 10 00 00
	0010	100	XX	00 00 00 10 10 00 00 01
SUB	0011	010	XX	01 00 10 00 00 00 00 00
	0011	011	XX	00 01 00 00 00 10 00 00
	0011	100	XX	00 00 00 10 11 00 00 01
STA	0100	010	XX	01 00 10 00 00 00 00 00
	0100	011	XX	00 10 00 01 00 00 00 00
	0100	100	XX	00 00 00 00 00 00 00 00
LDI	0101	010	XX	00 00 10 10 00 00 00 00
	0101	011	XX	00 00 00 00 00 00 00 00
	0101	100	XX	00 00 00 00 00 00 00 00
JMP	0110	010	XX	00 00 10 00 00 00 00 10
	0110	011	XX	00 00 00 00 00 00 00 00

	0110	100	XX	00 00 00 00 00 00 00 00
JC	0111	010	0X	00 00 00 00 00 00 00 00
	0111	010	1X	00 00 10 00 00 00 00 10
	0111	011	XX	00 00 00 00 00 00 00 00
	0111	100	XX	00 00 00 00 00 00 00 00
JZ	1000	010	X0	00 00 00 00 00 00 00 00
	1000	010	X1	00 00 10 00 00 00 00 10
	1000	011	XX	00 00 00 00 00 00 00 00
	1000	100	XX	00 00 00 00 00 00 00 00
OUT	1110	010	XX	00 00 00 01 00 01 00 00
	1110	011	XX	00 00 00 00 00 00 00 00
	1110	100	XX	00 00 00 00 00 00 00 00
HLT	1111	010	XX	10 00 00 00 00 00 00 00
	1111	011	XX	00 00 00 00 00 00 00 00
	1111	100	XX	00 00 00 00 00 00 00 00

Таблица 2.3: *Соответствие шага каждой команды машинному коду*

2.7 Тестирование

В процессе сборки учебного тренажёра тестировался каждый модуль для обеспечения корректной работы.

Первым тестировался тактовый генератор на факт того, что он стабильно переключается между автоматическим и ручным режимом. Ручные такты не должны были генерировать больше одного такта за одно нажатие кнопки (такая проблема решалась в процессе сборки).

Далее тестировались регистры – производились действия по проверке того, что они корректно считывают данные с шины, успешно их запоминают, могут передавать обратно на шину. Тестировался сценарий передачи данных между регистрами через шину. При этом управляющие сигналы контролировались вручную, т.к. модуля декодера команд не было.

Регистры флагов должны корректно заполнятся, когда в ALU находился ноль или случалось переполнение.

Особое внимание было уделено тестированию арифметико-логического юнита (ALU). Вручную помещался ноль в регистр A, в регистр B единица, ALU считала сумму, помещала результат в регистр A. На каждом шаге ALU должно было выводить число на единицу больше предыдущего, пока не переполнится и не начнёт заново с нуля (проблема, что 79+1=16 решалась в процессе сборки).

По похожей схеме тестировался дисплей – он должен был корректно отобразить все 256 значений как в знаковом, так и в беззнаковом режиме.

Оперативная память тестировалась следующим образов: в ячейку с адресом 0 помещались данные X, затем переключался на другой адрес, в него записывались другие данные и при возвращение на адрес 0 там должны быть те же данные X.

Нажатие кнопки сброса должно сбросить программный счётчик на ноль и очистить данные в регистрах.

Тестирование запрограммированного EEPROM проще проводить, когда оно через Arduino Nano подключено к ПК. В интерфейсе программного обеспечения Arduino IDE проверяется соответствие кода каждой команды к соответствующим наборам управляющих сигналов путём чтения данных с EEPROM.

Выводы

Созданная интерактивная 8-битная модель компьютера демонстрирует базовые принципы архитектуры фон Неймана. Основные результаты включают: реализацию открытой архитектуры с ручным управлением тактами, визуализацию данных через светодиоды и 7-сегментный дисплей, а также режим программирования памяти через переключатели. Тестирование подтвердило корректность работы всех модулей, включая регистры, АЛУ и память. Поставленная задача решена в полном объёме: модель устраняет ключевые недостатки существующих аналогов — высокую стоимость и отсутствие тактильного взаимодействия. При этом ограничения, такие как отсутствие поддержки кэширования и конвейеризации, сужают её применение в продвинутых курсах. Тем не менее для начального уровня обучения она обеспечивает эффективный баланс между наглядностью и сложностью.

Заключение

В ходе работы была разработана интерактивное учебное пособие, позволяющее изучить устройство и принципы работы компьютера через тактильное взаимодействие. Модель прошла тестирование и полностью готова к использованию в учебном процессе. Её основное преимущество — прозрачность процессов за счёт визуализации и ручного управления. Такой процесс превращает абстракции в осязаемый опыт и открывает новые возможности для изучения более сложных концепций.

Список литературы

[1] Таненбаум, Э. // Архитектура компьютера. — 6-е изд. — СПб.: Питер, 2014.

[2] Паттерсон, Д. / Хеннеси, Дж. // Организация и проектирование компьютеров. — М.: Лаборатория Базовых Знаний, 2020.

[3] Петцольд, Ч. // Код: Тайный язык информатики. — М.: Русская Редакция, 2001.

[4] Интернет-портал " IEEE Explore": сайт. — URL: https://ieeexplore.ieee.org/abstract/document/8718630 (дата обращения: 10.04.2025). — Текст: электронный.

[5] Практикум на учебном стенде ТРЭИ: метод. указания / МГТУ им. Н.Э. Баумана. — М.: Изд-во МГТУ, 2022.

[6] Интернет-портал "Учтех-Профи": сайт. — URL: https://labstand.ru/catalog/vychislitelnaya-tehnika/laboratornyj-stend-arhitektura-evm (дата обращения: 10.04.2025). — Текст: электронный.

[7] Тренажёр «ЛамПанель» // Образовательный портал К. Полякова. — URL: https://kpolyakov.spb.ru/prog/lamp.htm (дата обращения: 10.04.2025). — Текст: электронный.

[8] Официальный сайт Бена Итера: сайт. — URL: https://eater.net/8bit (дата обращения: 19.09.2024). — Текст: электронный.

[9] Официальный сайт "SYC Electronica": сайт. — URL: https://www.sycelectronica.com.ar/semiconductores/ (дата обращения: 24.11.2024). — Текст: электронный.

Приложение

А. Фото модели с указанием компонентов

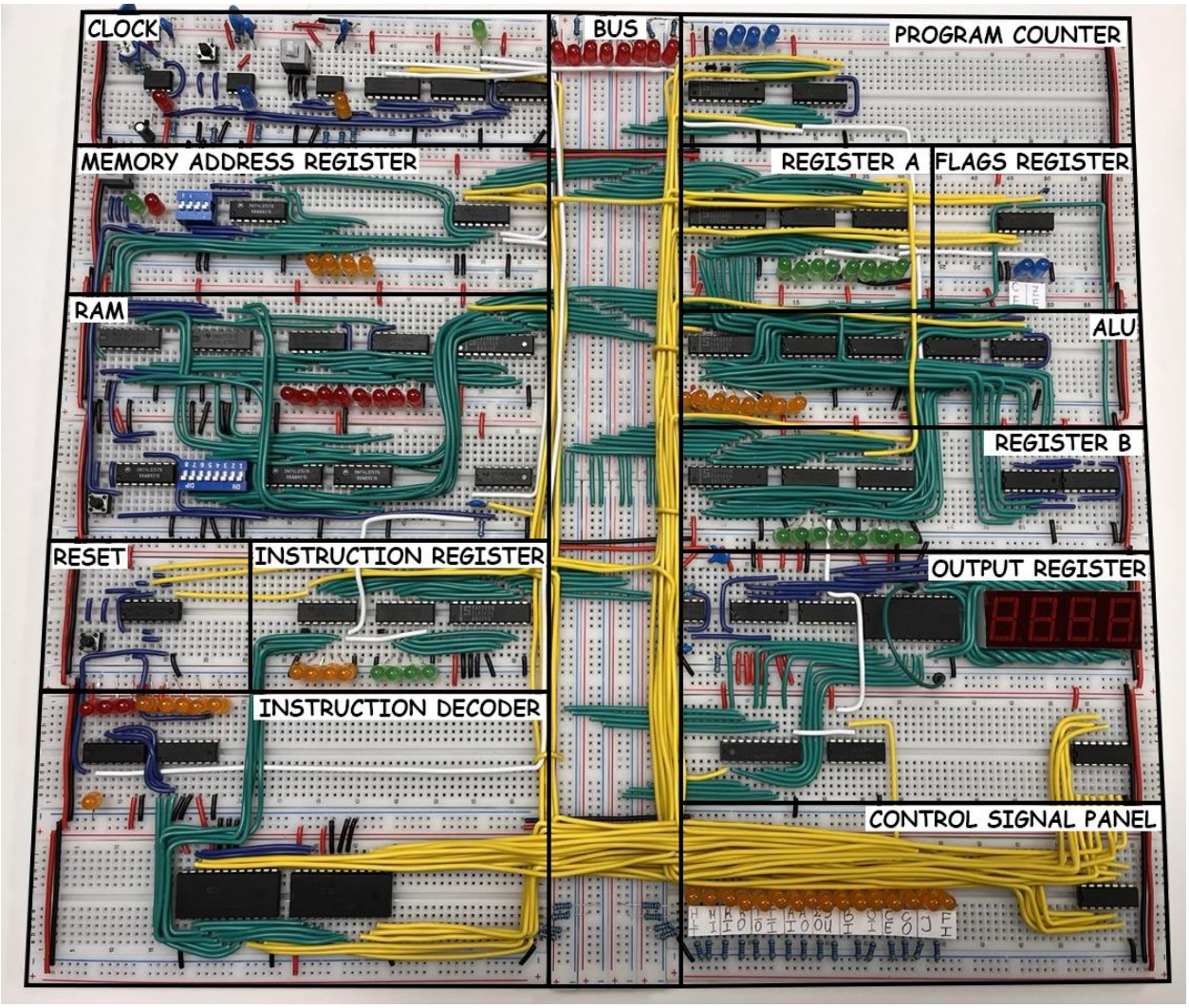

Рис. 3.3: Модель компьютера с подписанными модулями

Б. Инструкция по взаимодействию с моделью компьютера

1. Подготовка к работе

1.1. Код программы:

- Напишите программу на языке ассемблера данного компьютера
- По таблице переведите команды в машинный код

1.2. Питание:

- Подключите блок питания +5V к разъёму на плате
- Убедитесь, что загорелся светодиод «POWER» возле ОЗУ

1.3. Инициализация:

- Нажмите кнопку «RESET» для обнуления регистров и программного счётчика

2. Программирование – заполните ОЗУ в режиме “Program” машинным кодом программы

2.1. Выбор адреса:

- Установите 4-битный адрес ячейки памяти с помощью DIP-переключателей «A0-A3»

2.2. Ввод данных:

- Установите 8-битное значение на DIP-переключателях «D0-D7»

2.3. Запись в память:

- Нажмите кнопку «WRITE»
- Убедитесь, что зелёный светодиод «WRITE_OK» мигнул

3. Выполнение программы – доступно два режима работы

3.1. Ручной режим:

- Установите переключатель «CLK_MODE» в положение «MANUAL»
- Нажимайте кнопку «BEAT» для пошагового выполнения команд

3.2. Автоматический режим:

- Установите переключатель «CLK_MODE» в положение «AUTO»
- Регулируйте частоту тактового генератора потенциометром «CLK_SPEED»

4. Визуализация данных

4.1. Шина данных:

- Светодиоды «BUS» отображают текущие данные

4.2. Регистры:

- Светодиоды групп «REG_A», «REG_B» и другие отображают их состояние

4.3. Дисплей:

- 7-сегментный дисплей показывает результат в десятичном формате

В. Примеры тестовых программ с пошаговым выполнением

Программа 1. Сложение двух чисел и вывод результата на дисплей

- Нужно сложить числа 28 и 14 и вывести результат на дисплей. В память положим эти два числа, по очереди загрузим в регистр A и B, сложим, сохраним результат в регистр A и выведем его на дисплей.

- Для начала напишем код на языке ассемблера

  LDA 14 ; загрузить в регистр A число из памяти по адресу 14

  ADD 15 ; загрузить в регистр B число из памяти по адресу 15 и сложить с числом в регистре B. Результат сохранить в регистр A

  OUT ; вывести на дисплей содержимое регистра A

  HLT ; закончить выполнение программы

- Соответствующий машинный код (адрес: код команды) поместим в ОЗУ и запустим компьютер

  0000: 0001 1110

  0001: 0010 1111

  0010: 1110 0000

  0011: 1111 0000

  1110: 0001 1100

  1111: 0000 1110

- Управляющие сигналы, которые посылаются, перечислены в таблице 3.1

Шаг	LDA 14	ADD 15	OUT	HLT
000	CO MI	CO MI	CO MI	CO MI
001	RO II CE	RO II CE	RO II CE	RO II CE
010	IO MI	IO MI	AO OI	HLT
011	RO AI	RO BI		
100		EO AI		

Таблица 3.1: *Управляющие сигналы каждого шага команд*

- После остановки компьютера на дисплее будет выведена ожидаемая сумма – 42

Программа 2. Генерация и вывод чисел Фибоначчи на дисплей:

- Нужно генерировать числа Фибоначчи и последовательно выводить их на дисплей

- Для начала напишем код на языке ассемблера

  LDI 0x1 ; в регистр A записать 1

  STA 0xE ; по адресу 14 сохранить содержимое регистра A (1)

  LDI 0x0 ; в регистр A записать 0

  OUT ; выведем содержимое регистра A на дисплей (0, 1)

  ADD 0xE ; поместить в регистр B содержимое ячейки памяти по адресу 14 (1), ALU выдаст сумму (2), которая сохранится в регистр A

  STA 0xF ; по адресу 15 сохранить содержимое регистра A (2)

  LDA 0xE ; загрузить в регистр A число из памяти по адресу 14 (1)

  STA 0xD ; по адресу 13 сохранить содержимое регистра A (1)

  LDA 0xF ; загрузить в регистр A число из памяти по адресу 15 (2)

STA 0xE ; по адресу 14 сохранить содержимое регистра A (2)

LDA 0xD ; загрузить в регистр A число из памяти по адресу 13 (1)

JC 0x0 ; установить программный счётчик на 0, если было переполнение в АЛУ

JMP 0x3 ; установить программный счётчик на 3

- Соответствующий машинный код (адрес: код команды) поместим в ОЗУ и запустим компьютер

  0000: 0101 0001

  0001: 0100 1110

  0010: 0101 0000

  0011: 1110 0000

  0100: 0010 1110

  0101: 0100 1111

  0110: 0001 1110

  0111: 0100 1101

  1000: 0001 1111

  1001: 0100 1110

  1010: 0001 1101

  1011: 0111 0000

  1100: 0110 0011

- Компьютер будет циклично выводить на дисплей первые 12 чисел Фибоначчи

Г. Код микропрограммирования EEPROM для дисплея на языке C

Код также доступен в репозитории https://github.com/Darnelo-Inc/8bit

```
#define SHIFT_DATA 2
#define SHIFT_CLK 3
#define SHIFT_LATCH 4
#define EEPROM_D0 5
#define EEPROM_D7 12
#define WRITE_EN 13

void setAddress(int address, bool outputEnable) {
  shiftOut(SHIFT_DATA, SHIFT_CLK, MSBFIRST, (address >>
8) | (outputEnable ? 0x00 : 0x80));
  shiftOut(SHIFT_DATA, SHIFT_CLK, MSBFIRST, address);

  digitalWrite(SHIFT_LATCH, LOW);
  digitalWrite(SHIFT_LATCH, HIGH);
  digitalWrite(SHIFT_LATCH, LOW);
}

byte readEEPROM(int address) {
  for(int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1)
{
    pinMode(pin, INPUT);
  }
  setAddress(address, true);
  byte data = 0;

  for(int pin = EEPROM_D7; pin >= EEPROM_D0; pin -= 1)
{
    data = (data << 1) + digitalRead(pin);
  }

  return data;
}

void writeEEPROM(int address, byte data) {
  setAddress(address, false);

  for(int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1)
{
    pinMode(pin, OUTPUT);
  }
```

```
  for(int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1)
{
    digitalWrite(pin, data & 1);
    data = data >> 1;
  }

  digitalWrite(WRITE_EN, LOW);
  delayMicroseconds(1);
  digitalWrite(WRITE_EN, HIGH);
  delay(10);
}

void printContents() {
  for(int base = 0; base <= 255; base += 16) {
    byte data[16];

    for(int offset = 0; offset <= 15; offset += 1) {
      data[offset] = readEEPROM(base + offset);
    }

    char buffer[80];
    sprintf(buffer, "%03x:  %02x %02x %02x %02x %02x
%02x %02x %02x   %02x %02x %02x %02x %02x %02x %02x
%02x",
            base, data[0], data[1], data[2], data[3],
data[4], data[5], data[6], data[7],
            data[8], data[9], data[10], data[11],
data[12], data[13], data[14], data[15]);

    Serial.println(buffer);
  }
}

byte data[] = { 0x01, 0x4f, 0x12, 0x06, 0x4c, 0x24,
0x20, 0x0f, 0x00, 0x04, 0x08, 0x60, 0x31, 0x42, 0x30,
0x38 };

void setup() {
  pinMode(SHIFT_DATA, OUTPUT);
  pinMode(SHIFT_CLK, OUTPUT);
  pinMode(SHIFT_LATCH, OUTPUT);

  digitalWrite(WRITE_EN, HIGH);
  pinMode(WRITE_EN, OUTPUT);
```

```
  Serial.begin(57600);

  Serial.print("Erasing EEPROM");
  for(int address = 0; address <= 2047; address += 1) {
    writeEEPROM(address, 0xff);
  }
  Serial.println(" done");

  Serial.print("Programming EEPROM");
  for(int address = 0; address <= 15; address += 1) {
    writeEEPROM(address, data[address]);
  }
  Serial.println(" done");

  Serial.println("Reading EEPROM");
  printContents();
}
```

Д. Код микропрограммирования EEPROM для декодера на языке C

Код также доступен в репозитории https://github.com/Darnelo-Inc/8bit

```
#define SHIFT_DATA 2
#define SHIFT_CLK 3
#define SHIFT_LATCH 4
#define EEPROM_D0 5
#define EEPROM_D7 12
#define WRITE_EN 13

#define HLT 0b1000000000000000  // Halt clock
#define MI  0b0100000000000000  // Memory address
register in
#define RI  0b0010000000000000  // RAM data in
#define RO  0b0001000000000000  // RAM data out
#define IO  0b0000100000000000  // Instruction register
out
#define II  0b0000010000000000  // Instruction register
in
#define AI  0b0000001000000000  // A register in
#define AO  0b0000000100000000  // A register out
#define EO  0b0000000010000000  // ALU out
#define SU  0b0000000001000000  // ALU subtract
#define BI  0b0000000000100000  // B register in
#define OI  0b0000000000010000  // Output register in
#define CE  0b0000000000001000  // Program counter
enable
#define CO  0b0000000000000100  // Program counter out
#define J   0b0000000000000010  // Jump (program
counter in)
#define FI  0b0000000000000001  // Flags in

#define FLAGS_Z0C0 0
#define FLAGS_Z0C1 1
#define FLAGS_Z1C0 2
#define FLAGS_Z1C1 3

#define JC  0b0111
#define JZ  0b1000

uint16_t UCODE_TEMPLATE[16][8] = {
{MI|CO,RO|II|CE,0,    0,    0,          0,0,0},// NOP
{MI|CO,RO|II|CE,IO|MI,RO|AI,0,          0,0,0},// LDA
```

```
{MI|CO,RO|II|CE,IO|MI,RO|BI,EO|AI|FI,   0,0,0},// ADD
{MI|CO,RO|II|CE,IO|MI,RO|BI,EO|AI|SU|FI,0,0,0},// SUB
{MI|CO,RO|II|CE,IO|MI,AO|RI,0,          0,0,0},// STA
{MI|CO,RO|II|CE,IO|AI,0,    0,          0,0,0},// LDI
{MI|CO,RO|II|CE,IO|J, 0,    0,          0,0,0},// JMP
{MI|CO,RO|II|CE,0,    0,    0,          0,0,0},// JC
{MI|CO,RO|II|CE,0,    0,    0,          0,0,0},// JZ
{MI|CO,RO|II|CE,0,    0,    0,          0,0,0},// 1001
{MI|CO,RO|II|CE,0,    0,    0,          0,0,0},// 1010
{MI|CO,RO|II|CE,0,    0,    0,          0,0,0},// 1011
{MI|CO,RO|II|CE,0,    0,    0,          0,0,0},// 1100
{MI|CO,RO|II|CE,0,    0,    0,          0,0,0},// 1101
{MI|CO,RO|II|CE,AO|OI,0,    0,          0,0,0},// OUT
{MI|CO,RO|II|CE,HLT,  0,    0,          0,0,0},// HLT
};

uint16_t ucode[4][16][8];

void initUCode() {
  // ZF = 0, CF = 0
  memcpy(ucode[FLAGS_Z0C0], UCODE_TEMPLATE,
sizeof(UCODE_TEMPLATE));

  // ZF = 0, CF = 1
  memcpy(ucode[FLAGS_Z0C1], UCODE_TEMPLATE,
sizeof(UCODE_TEMPLATE));
  ucode[FLAGS_Z0C1][JC][2] = IO|J;

  // ZF = 1, CF = 0
  memcpy(ucode[FLAGS_Z1C0], UCODE_TEMPLATE,
sizeof(UCODE_TEMPLATE));
  ucode[FLAGS_Z1C0][JZ][2] = IO|J;

  // ZF = 1, CF = 1
  memcpy(ucode[FLAGS_Z1C1], UCODE_TEMPLATE,
sizeof(UCODE_TEMPLATE));
  ucode[FLAGS_Z1C1][JC][2] = IO|J;
  ucode[FLAGS_Z1C1][JZ][2] = IO|J;
}

/*
 * Output the address bits and outputEnable signal
using shift registers.
 */
void setAddress(int address, bool outputEnable) {
```

```
  shiftOut(SHIFT_DATA, SHIFT_CLK, MSBFIRST, (address >>
8) | (outputEnable ? 0x00 : 0x80));
  shiftOut(SHIFT_DATA, SHIFT_CLK, MSBFIRST, address);

  digitalWrite(SHIFT_LATCH, LOW);
  digitalWrite(SHIFT_LATCH, HIGH);
  digitalWrite(SHIFT_LATCH, LOW);
}


/*
 * Read a byte from the EEPROM at the specified
address.
 */
byte readEEPROM(int address) {
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1)
{
    pinMode(pin, INPUT);
  }
  setAddress(address, /*outputEnable*/ true);

  byte data = 0;
  for (int pin = EEPROM_D7; pin >= EEPROM_D0; pin -= 1)
{
    data = (data << 1) + digitalRead(pin);
  }
  return data;
}


/*
 * Write a byte to the EEPROM at the specified address.
 */
void writeEEPROM(int address, byte data) {
  setAddress(address, /*outputEnable*/ false);
  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1)
{
    pinMode(pin, OUTPUT);
  }

  for (int pin = EEPROM_D0; pin <= EEPROM_D7; pin += 1)
{
    digitalWrite(pin, data & 1);
    data = data >> 1;
  }
```

```
  digitalWrite(WRITE_EN, LOW);
  delayMicroseconds(1);
  digitalWrite(WRITE_EN, HIGH);
  delay(10);
}


/*
 * Read the contents of the EEPROM and print them to
the serial monitor.
 */
void printContents(int start, int length) {
  for (int base = start; base < length; base += 16) {
    byte data[16];
    for (int offset = 0; offset <= 15; offset += 1) {
      data[offset] = readEEPROM(base + offset);
    }

    char buf[80];
    sprintf(buf, "%03x:  %02x %02x %02x %02x %02x %02x
%02x %02x   %02x %02x %02x %02x %02x %02x %02x %02x",
            base, data[0], data[1], data[2], data[3],
data[4], data[5], data[6], data[7],
            data[8], data[9], data[10], data[11],
data[12], data[13], data[14], data[15]);

    Serial.println(buf);
  }
}


void setup() {
  initUCode();

  pinMode(SHIFT_DATA, OUTPUT);
  pinMode(SHIFT_CLK, OUTPUT);
  pinMode(SHIFT_LATCH, OUTPUT);
  digitalWrite(WRITE_EN, HIGH);
  pinMode(WRITE_EN, OUTPUT);
  Serial.begin(57600);

  Serial.print("Programming EEPROM");

  // Program the 8 high-order bits of microcode into
the first 128 bytes of EEPROM
```

```
  for (int address = 0; address < 1024; address += 1) {
    int flags       = (address & 0b1100000000) >> 8;
    int byte_sel    = (address & 0b0010000000) >> 7;
    int instruction = (address & 0b0001111000) >> 3;
    int step        = (address & 0b0000000111);

    if (byte_sel) {
      writeEEPROM(address,
ucode[flags][instruction][step]);
    } else {
      writeEEPROM(address,
ucode[flags][instruction][step] >> 8);
    }

    if (address % 64 == 0) {
      Serial.print(".");
    }
  }

  Serial.println(" done");


  // Read and print out the contents of the EERPROM
  Serial.println("Reading EEPROM");
  printContents(0, 1024);
}
```