СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc356903287)

[1 РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ 5](#_Toc356903288)

[1.1 Функциональный состав микро-ЭВМ 5](#_Toc356903289)

[1.2 Разработка системы команд 7](#_Toc356903290)

[1.3 Описание взаимодействия блоков микро-ЭВМ при выполнении команд программы 9](#_Toc356903291)

[2 РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ 12](#_Toc356903292)

[2.1 Блок ПЗУ 12](#_Toc356903293)

[2.2 Блок ОЗУ 13](#_Toc356903294)

[2.3 Стек 15](#_Toc356903295)

[2.4 Кэш данных 16](#_Toc356903296)

[2.5 Блок выполнения команд (АЛУ) 17](#_Toc356903297)

[2.6 Блок загрузки команд 19](#_Toc356903298)

[2.7 Блок загрузки данных 20](#_Toc356903299)

[2.8 Блок записи данных 22](#_Toc356903300)

[2.9 Устройство управления 24](#_Toc356903301)

[2.10 Арбитр 27](#_Toc356903302)

[3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ 29](#_Toc356903303)

[3.1 Блок ПЗУ 29](#_Toc356903304)

[3.2 Блок ОЗУ 30](#_Toc356903305)

[3.3 Регистры общего назначения 31](#_Toc356903306)

[3.4 Стек 31](#_Toc356903307)

[3.5 Арбитр 32](#_Toc356903308)

[3.6 Кэш данных 33](#_Toc356903309)

[3.7 Тестовая программа 35](#_Toc356903310)

[4 АНАЛИЗ И ОПТИМИЗАЦИЯ МИКРО-ЭВМ 29](#_Toc356903311)

[ЗАКЛЮЧЕНИЕ 32](#_Toc356903312)

# ВВЕДЕНИЕ

В соответствии с вариантом необходимо разработать микро-ЭВМ, основанную на гарвардской архитектуре (хранилища и каналы инструкций и данных физически разделены). Шина адреса имеет ширину 16 бит, а шина данных — 8 бит.

Постоянное запоминающее устройство (ПЗУ) и оперативное запоминающее устройство (ОЗУ) должны иметь синхронное и асинхронное управление соответственно.

Устройство должно иметь возможность работать с прямой и прямой регистровой адресацией.

Микро-ЭВМ должна включать в себя 12 регистров общего назначения (РОН).

Сквозная кэш память с отображением должна иметь уровень ассоциативности равный 2. Алгоритм замещения строк — наименьшего использования.

Поддерживаемые операции должны включать в себя: **HLT, NOP, MOV, JMP, JNZ, PUSH, POP, SUB, AND, NOTZ, SRL.**

Стек должен расти вниз и вмещать 9 слов.

Микро-ЭВМ должна включать в себя децентрализованный кольцевой арбитр.

Предсказатель переходов должен работать по схеме А5 и использовать 3-битный Local HR.

Разработка описанного выше устройства будет проводиться с использованием специализированного приложения Altera Quartus II 9.1. Опыт работы в данном приложении был получен мной в ходе выполнения лабораторных работ по курсу в течении предыдущих семестров обучения. Данная среда разработки позволяет быстро создавать отдельные модули, тестировать их, а также инкапсулировать их в отдельные символы для дальнейшего использования.

# РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ

В данном разделе описаны особенности функционального строения микро-ЭВМ, описание взаимодействия всех блоков и архитектура системы команд.

## Функциональный состав микро-ЭВМ

В соответствии с заданием необходимо включить в реализацию следующие функциональные блоки:

* Устройство управления (УУ);
* Арифметико-логическое устройство (АЛУ);
* Стек;
* Регистры общего назначения (РОН);
* Оперативное запоминающее устройство (ОЗУ);
* Постоянное запоминающее устройство (ПЗУ);
* Арбитр

Рассмотрим подробнее каждый из блоков.

Устройство управления определяет набор возможностей управления состоянием центрального процессора микро-ЭВМ. Основное его назначение — контроль за работой микро-ЭВМ, отображение текущего состояния системы в целом, а также изменение этого состояния.

Арифметико-логическое устройство выполняет необходимые операции над предоставленными ей операндами. В данном устройстве определен набор команд, которые позволяют определенным образом изменять предоставленные данные. Как правило такими командами являются арифметические и логические операции.

Блок регистров представляет из себя набор энергозависимой памяти небольшого объема, предназначенный для кратковременного хранения оперативной информации. Фактически, один регистр является набором триггеров для хранения данных и схемы для удобного управления работой этих триггеров. Блок регистров включает в себя набор регистров, каждый из которых имеет собственный адрес (имя/номер) в этом блоке. Регистр предоставляет следующие операции: чтение данных из регистра, запись данных в регистр.

Стек представляет из себя набор регистров, из которых в один момент времени виден только «верхний» — тот регистр, данные в котором наиболее новые. Для стека возможны три операции: помещение данных в стек, извлечение данных из стека, просмотр данных на вершине стека.

Постоянное запоминающее устройство предназначено для долговременного хранения неизменяемых данных. Как правило такими данными является программа работы микро-ЭВМ.

Оперативное запоминающее устройство используется для энергозависимого хранения данных. По сравнению с регистрами, ОЗУ обладает большим временем доступа и большей вместимостью, что, как правило, объясняется расположением ОЗУ отдельно от ядра микро-ЭВМ, тогда как регистры обычно находятся на том же чипе. Асинхронный протокол работы ОЗУ обеспечивает стабильность работы при разных тактовых частотах памяти и процессора. Обратной стороной является уменьшение производительности вследствие усложнения протокола взаимодействия.

Арбитраж необходим в тех случаях, когда ведущим устройством на шине может быть не одно устройство, а несколько. В таких случаях специальным образом выбирается приоритетное устройство, которое и становится ведущим на шине. Децентрализованный кольцевой арбитр является вариантом реализации схемы арбитража, при котором каждое потенциально ведущее устройство запрашивает доступ к шине у одного из узлов арбитра (закрепленного за данным ведущим устройству), и при этом узлы объединены кольцевой топологией, которая используется для определения приоритетов.

## Разработка системы команд

Согласно заданию необходимо реализовать 11 основных команд. Для разных видов адресации требуется от 4 до 16 битов на один аргумент операции (4 бита для регистровой адресации, 16 бит для прямой адресации).

Для упрощения реализации операций по обработке команд решено использовать команды фиксированной длины.

Также решено использовать формат команды, включающий в себя 2 аргумента операции, при котором один аргумент будет фиксированного типа (регистр/стек), а тип второго может изменяться (регистр/стек/адрес памяти). Некоторые команды не требуют обоих аргументов. Для упрощения формата команды неиспользуемые аргументы становятся зарезервированными областями.

С учетом поставленных требований был разработан следующий формат команд:

Таблица 1. Битовое представление команды

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Биты | 31-28 | 27-26 | 25-24 | 23-20 | 19-16 | 15-0 |
| Поле | Код операции | Зарезерв. | Тип аргумента 2 | Зарезерв. | Аргумент 1 | Аргумент 2 |

Аргумент 1 представляет из себя номер регистра либо специальное значение (0xF) для вершины стека. Аргумент 2 может иметь следующие типы (которые также являются кодами поля типа аргумента 2):

* Тип 00 — прямая регистровая адресация, поле аргумента содержит номер регистра либо код вершины стека;
* Тип 01 — прямая адресация — поле аргумента содержит адрес ОЗУ, по которому располагаются данные;
* Тип 10 — непосредственная адресация. Решено добавить данный вид адресации, так как он позволяет упростить и ускорить процесс выборки аргументов в случае, если они не изменяются в ходе работы. В частности, при использовании данного способа можно увеличить скорость работы таких инструкций, как JMP или MOV, а так же сэкономить небольшой объем ОЗУ за счет ПЗУ (располагая аргументы прямо в команде).

Архитектура системы команд описана в таблице ниже.

Таблица 2. Архитектура системы команд

| Номер | Бинарный код | Символ | Описание | Наличие аргументов |
| --- | --- | --- | --- | --- |
| 1 | 0000 | HLT | Остановка выполнения | Нет |
| 2 | 0001 | NOP | Пустая операция | Нет |
| 3 | 0010 | MOV | Перемещение данных в регистр | 1, 2 |
| 4 | 0011 | MOVM | Перемещение данных в память | 1, 2 |
| 5 | 0100 | PUSH | Помещение данных в стек | 2 |
| 6 | 0101 | POP | Извлечение данных из стека | 1 |
| 7 | 0110 | JMP | Безусловный переход на инструкцию | 2 |
| 8 | 0111 | JNZ | Переход на инструкцию по отсутствию флага Z | 2 |
| 9 | 1000 | SRL | Логический сдвиг вправо | 1, 2 |
| 10 | 1001 | SUB | Вычитание беззнаковых целых чисел | 1, 2 |
| 11 | 1010 | AND | Бинарное И | 1, 2 |
| 12 | 1011 | NOTZ | Инверсия по флагу Z | 1 |

Преимуществами данной архитектуры являются простота формата команды, отсутствие необходимости в сложном блоке декодирования команд, простота загрузки команд из-за их фиксированного размера, возможность расширения.

Недостатком является относительно большая избыточность (6 бит из 32 — зарезервированные), а также ограниченный набор используемых типов аргументов для команд.

## Описание взаимодействия блоков микро-ЭВМ при выполнении команд программы

Для того чтобы устройство начало работать необходимо два условия:

1. Подача тактирующего сигнала на вход CLK;
2. П-образный сигнал на входе START.

При подаче П-образного сигнала устройство инициализирует необходимые узлы, а затем переходит к основному режиму работы. Данный режим состоит в попеременном чередовании двух этапов выполнения операций.

На первом этапе команда загружается из ПЗУ в регистр команд процессора IR. Адрес команды для загрузки указан в регистре адреса команды IP. На данном этапе блок загрузки инструкции выставляет нужные адреса на шину адреса ПЗУ, считывает байты команды в буфер, а затем записывает полную команду в регистр IR. Считывание одной команды требует 4 обращений к ПЗУ. Для минимизации повторных считываний команд можно использовать кэш.

На втором этапе три блока выполняют разные подэтапы одной команды — загрузка аргументов, выполнение, запись результатов. Примечательным является использование pull-based модели взаимодействия между блоками. В начале этапа устройство управления обращается к *последнему* блоку (записи) и ждет, пока он не выставит статус своей готовности в «1». Данный блок обращается к предыдущему блоку, ждет его готовности, считывает нужные данные, а затем выполняет свой этап команды. При этом предыдущий блок (исполнения) в свою очередь обращается к своему предыдущему блоку (чтения данных) и ждет загрузки аргументов.

На первом подэтапе блок загрузки данных загружает необходимые аргументы в соответствии с их типами, а затем передает их на второй этап. Для функционирования блок должен быть связан с блоком регистров IP/IR, блоком РОН, стеком и кэшем.

Второй подэтап — выполнение команды. На данном этапе над данными выполняется одна из операций заранее предопределенного набора. Как одна из операций, блок может просто пропускать данные на выход их блока. Следует отметить, что операции, выполняемые на данном подэтапе, необязательно должны соответствовать команде, поступившей на процессор.

Третьим подэтапом является запись данных в регистры/стек/память. По окончанию этапа в регистр адреса команды записывается новый адрес, который будет использован при загрузке команды в следующем цикле работы.

Каждый вышеописанный этап представляет собой отдельный логический блок, режим работы которого задается соседними блоками и выполняемой командой. В первую очередь, у блоков есть сигналы запроса и готовности, необходимые для реализации взаимодействия между блоками. Кроме того, блок выполняет действия согласно заранее заданному набору команд, определенная команда из которого выбирается после загрузки с помощью блока выборки команды из ПЗУ.

Блоки чтения и записи данных обращаются к ОЗУ через кэш с помощью асинхронного протокола. Строка кэша содержит в себе два 16-битных слова.

# РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ

В данном разделе описываются и обосновываются решения принятые мною при проектировании всех имеющихся устройств микро-ЭВМ.

## Блок ПЗУ

Для реализации блока ПЗУ используется стандартный модуль Altera Quartus для реализации синхронного ПЗУ  — lpm\_rom. Реализована возможность подключения нескольких блоков к шинам адреса/данных команд процессора, благодаря наличию входа DEV\_ID, который определяет, на какие 4 старших бита адреса реагирует конкретный блок ПЗУ. Это позволяет изменять объем ПЗУ в устройстве. Кроме того, сегмент поля адреса, который не используется для идентификации блока, позволяет разместить только 4 КиБ данных, но данное ограничение легко обойти, реализовав блок, «реагирующий» не на отдельный DEV\_ID, а на диапазон.

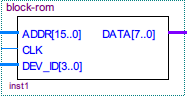


Рисунок 1. Условно-графическое обозначение блока ПЗУ

Входы и выходы блока:

1. CLK — тактовый сигнал
2. DEV\_ID — номер устройства. Данный вход устанавливает, на какие 4 старших бита адреса должна реагировать память.
3. ADDR — адрес, по которому происходит считывание данных
4. DATA — вывод данных, которые располагаются по адресу ADDR

## Блок ОЗУ

Так как Altera Quartus не предоставляет готового блока асинхронного ОЗУ, реализована схема управления, основанная на автомате Мура, с определенными состояниями простоя, принятия запроса, объявления готовности и завершения запроса, а в качестве модуля хранения данных используется стандартный модуль Altera Quartus для релизации синхронного ОЗУ — lpm\_ram\_dq.

Для совмещения адресных пространств ОЗУ и периферии старшие 4 бита адреса отдаются под номер устройства. Таким образом, можно, реконфигурируя аппаратное подключение ОЗУ/периферии к шине, задавать разные диапазоны адресов для разных блоков, а также расширять доступную память. Если к шине подключается устройство, которое может быть ведущим, необходимо подключать его через арбитр.

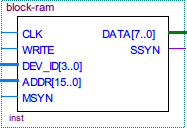


Рисунок 2. Условное графическое обозначение блока ОЗУ

Входы и выходы блока:

1. Сигнал CLK — сигнал тактовой частоты.
2. WRITE — если 1, то текущий запрос — на запись, иначе — на чтение.
3. ADDR, DATA и DEV\_ID аналогичны тем же выводам у ПЗУ.
4. MSYN/SSYN — сигналы синхронизации ведущего и ведомого устройства соответственно.

Для осуществления передачи данных ведущее устройство ждет, пока уровень SSYN не станет равным 0, затем выводит данные на вход ведомого устройства (на шину) и устанавливает MSYN=1. Далее ведомое устройство выполняет действия с данными, а ведущее ждет. Затеи ведомое устройство выведет данные на выходы и установит SSYN=1, это будет означать окончание обработки запроса. После этого ведущее устройство может снять MSYN, когда оно закончит запрос.

Здесь и далее основной принцип работы в асинхронном режиме соответствует вышеуказанному, названия выводов могут отличаться (например, REQ/RDY).

Блок регистров общего назначения состоит из 12 16-разрядных регистров. Обычно данные регистры используются для хранения временных результатов.

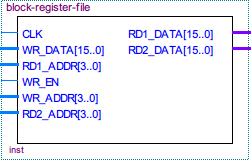


Рисунок 3. Условно-графическое обозначение блока РОН

Блок регистров содержит два выхода для чтения данных и один вход для записи данных, что позволяет одновременно считывать два аргумента. Каждый регистр реализован на триггерах.

Входы и выходы блока:

1. CLK — тактовый сигнал
2. RD1\_DATA, RD2\_DATA — первый и второй выходы чтения
3. RD1\_ADDR, RD2\_ADDR — номера регистров первого и второго выходов чтения соответственно
4. WR\_DATA — вход данных для записи
5. WR\_ADDR — номер регистра для записи
6. WR\_EN — разрешение на запись.

Запись в регистр происходит по переднему фронту CLK при WR\_EN=1.

## Стек

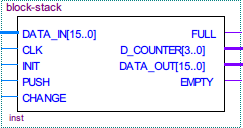


Рисунок 9. Условно-графическое обозначение стека

Стек реализован как набор регистров. Для выбора нужного регистра используется счетчик, который необходимо инициализировать перед запуском выполнения программы на микро-ЭВМ.

Входы и выходы блока:

1. CLK — тактовый сигнал
2. INIT — вход сигнала инициализации, при установке в «1» внутренний счетчик стека сбрасывается на значение, соответствующее пустому стеку.
3. DATA\_IN — вход данных для записи в стек
4. DATA\_OUT — выход значения текущей вершины стека
5. CHANGE — сигнал проведения операции помещения в стек/изъятия из стека
6. PUSH — сигнал помещения в стек, если «1» — данные будут помещаться в стек, иначе — извлекаться из него.

При CHANGE установленном в «1» по тактовому сигналу вершина стека будет сдвигаться на 1 ячейку вверх/вниз (в зависимости от PUSH).

## Кэш данных

Используется кэш со сквозной записью с отображением с коэффициентом ассоциативности 2. Одна строка кэша содержит 2 16-битных слова.

Кэш данных управляется набором конечных автоматов. Основной принцип работы заключается в том, что главный автомат кэша определяет ситуацию, в которой находится кэш (простаивание, чтение/запись, попадание/промах) и при необходимости вызывает автомат, реализующий алгоритм для работы с нужным состоянием.

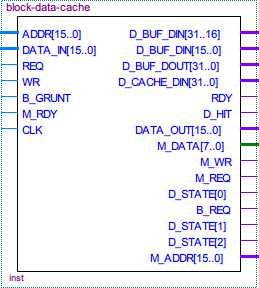


Рисунок 11. Условное графическое обозначение кэша

Входы и выходы блока:

1. CLK — тактовый сигнал
2. ADDR — запрашиваемый адрес памяти
3. DATA\_IN — данные для записи в память
4. DATA\_OUT — вывод данных кэша
5. WR — флаг выполнения записи в память
6. REQ — запрос данных из кэша/записи в кэш
7. RDY — сигнал готовности кэша
8. B\_REQ, B\_GRUNT — сигналы запроса и предоставления шины
9. MEM\_REQ, MEM\_RDY — сигналы запроса памяти и ее готовности
10. M\_WR — флаг записи в память
11. M\_DATA — вход/выход для данных с шины, к которой подключена ОЗУ
12. M\_ADDR — запрашиваемый адрес данных, выставляемый на шину
13. Выводы с префиксом «D\_» — отладочные выводы

Для чтения из памяти или записи в нее используются два соответствующих блока внутри кэша, которые отвечают за обращения к арбитру для доступа к шине и памяти.

## Блок выполнения команд (АЛУ)

В состав данного блока входят собственно АЛУ и логика реализации асинхронного протокола работы. При выставлении сигнала запроса к блоку, выставляется запрос к блоку загрузки операндов, а затем выполняется операция, которая устанавливается управляющим входом блока.

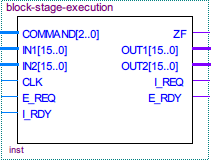


Рисунок 10. Условно-графическое обозначение АЛУ.

Входные и выходные сигналы:

1. CLK — тактовый сигнал
2. COMMAND — номер операции, которую должно выполнять АЛУ
3. IN1, IN2 — аргументы
4. E\_REQ — сигнал запроса операции АЛУ
5. E\_RDY — сигнал готовности операции АЛУ
6. I\_REQ — выход сигнала запроса загрузки данных от блока загрузки
7. I\_RDY — вход сигнала готовности от блока записи
8. ZF — значение флага Z
9. OUT1, OUT2 — выходные значения операции

Ниже представлен список операций, реализуемых АЛУ:

Таблица 3. Список операций АЛУ.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | АЛУ | Операнды | Код |
| 0 | Pass through | 2 | 000 |
| 1 | Shift Right Logical | 2 | 001 |
| 2 | Subtract | 2 | 010 |
| 3 | AND | 2 | 011 |
| 4 | NOTZ | 1 | 100 |

АЛУ работает в качестве второй стадии выполнения команды в асинхронном режиме. Следует отметить операцию Pass through (0) — отправку аргументов на выход без изменений, используется в таких командах, как MOV, JMP и прочие, не требующие вычислений, но использующие аргументы для записи данных. Код операции устанавливается устройством управления после загрузки команды.

Отдельные операции АЛУ выполнены как комбинационные схемы, что не вносит тактовой задержки в схему, однако увеличивает необходимую длительность такта из-за большой логической глубины некоторых операций, в частности, операция логического сдвига реализована схемой типа «barrel shifter».

Так как по заданию требуется реализация только одного флага состояния — ZF, регистр флагов включен в состав блока выполнения команд, так как больше нигде данный флаг состояния изменен быть не может. Однако при появлении других флагов может быть выделен отдельный блок регистров (по функционированию аналогичный блоку IP/IR) для хранения флагов.

## Блок загрузки команд

Данный блок является реализацией первого этапа обработки команды процессором. Он подключается к шинам адреса/данных команд для загрузки команд из ПЗУ.

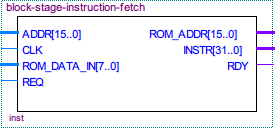


Рисунок 11. Условное графическое обозначение блока загрузки команд

Входы и выходы блока:

1. CLK — тактовый сигнал
2. REQ, RDY — сигналы запроса/готовности для реализации асинхронного протокола работы
3. ADDR — вход адреса запрашиваемой команды
4. ROM\_ADDR, ROM\_DATA\_IN — выход на шину адреса команд и вход шины данных команд
5. INSTR — выход данных команды

После установки адреса запрашиваемой команды и сигнала запроса команды по фронту тактового сигнала блок устанавливает на выход шины адреса 4 последовательных адреса, начиная с адреса запрашиваемой команды, и защелкивает в буферных регистрах 4 байта команды. После этого на выходе устанавливается сигнал готовности блока.

## Блок загрузки данных

Данный блок служит для загрузки операндов, которые будут использоваться для исполнения команды или при записи данных. Так как только для одного из аргументов может понадобиться обращение к кэшу, осуществляется одновременная загрузка обоих аргументов, где первый может размещаться в блоке регистров (используется первый канал чтения блока регистров) или в стеке, а второй — также в блоке регистров (используется второй канал чтения блока регистров), в стеке, в ОЗУ (используется кэш) или непосредственно размещается в теле команды (поле второго аргумента непосредственно соответствует аргументу).

Тип первого аргумента устанавливается по адресу чтения (благодаря тому, что вершине стека присвоен фиксированный адрес). Тип второго аргумента устанавливается флагами расположения 2 аргумента в памяти и непосредственного типа 2 аргумента (установлен может быть только один флаг из двух).

Для оптимизации количества операций блока выполнения команд реализована возможность обмена местами выходных данных. В случае установки флага первый аргумент команды отображается во втором выходе данных, а второй аргумент — в первом выходе.

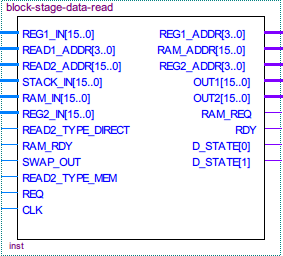


Рисунок 12. Условное графическое обозначение блока загрузки данных

Входы и выходы блока:

1. CLK — тактовый сигнал
2. REQ/RDY — сигналы запроса/готовности асинхронного протокола работы блока
3. READ1\_ADDR, READ2\_ADDR — адреса чтения первого и второго аргументов соответственно
4. READ2\_TYPE\_MEM — флаг расположения 2 аргумента в ОЗУ
5. READ2\_TYPE\_DIRECT — флаг непосредственного типа 2 аргумента
6. SWAP\_OUT — флаг обмена местами аргументов на выходе
7. OUT1, OUT2 — выходы 1 и 2 аргументов соответственно
8. REG1\_ADDR, REG2\_ADDR — выходы номеров запрашиваемых регистров для двух каналов чтения блока регистров
9. REG1\_IN, REG2\_IN — входы чтения данных регистров
10. RAM\_REQ/RAM\_RDY — сигналы запроса/готовности кэша
11. RAM\_ADDR — адрес запрашиваемых из кэша данных
12. RAM\_IN — вход запрашиваемых из кэша данных
13. STACK\_IN — вход данных вершины стека

## Блок записи данных

Данный блок является последним вызываемым блоком при обработке команды. Он записывает обработанные данные в соответствующую память, а также устанавливает значение регистра адреса команды IP на следующую команду.

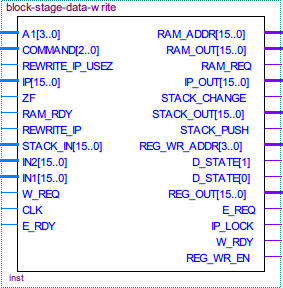


Рисунок 13. Условное графическое обозначение блока записи данных

Операции, выполняемые блоком, определяются номером команды блока записи, и флагами записи регистра адреса команды.

Таблица 4. Команды блока записи данных

| Код | Команда |
| --- | --- |
| 0 | Пустая операция |
| 1 | Запись 1 аргумента в регистр |
| 2 | Запись 1 аргумента в память по адресу, указанному во 2 аргументе |
| 3 | Запись 1 аргумента в стек (push) |
| 4 | Выгрузка аргумента из стека в регистр (pop) |

Входы и выходы блока:

1. CLK — тактовый сигнал
2. W\_REQ/W\_RDY — сигналы запроса/готовности блока для реализации асинхронного протокола работы
3. E\_REQ/E\_RDY — сигналы запроса/готовности блока выполнения команд
4. COMMAND — номер команды блока записи данных
5. REWRITE\_IP — флаг записи в регистр IP значения 1 аргумента
6. REWRITE\_IP\_USEZ — флаг учета флага Z при записи 1 аргумента в регистр IP (установка флага Z в 0 означает произведение записи аргумента)
7. ZF — значение флага Z
8. A1 — номер регистра для записи данных (соответствует 1 аргументу выполняемой команды)
9. IN1, IN2 — первый и второй входные аргументы блока
10. RAM\_REQ/RAM\_RDY — сигналы запроса/готовности кэша
11. RAM\_ADDR — адрес записи в кэш
12. RAM\_OUT — данные для записи в кэш
13. STACK\_CHANGE, STACK\_PUSH — флаги проведения операций со стеком (см. раздел 2.2)
14. STACK\_IN — данные вершины стека
15. STACK\_OUT — данные для записи в стек
16. REG\_WR\_ADDR — номер регистра для записи данных
17. REG\_WR\_EN — флаг записи в регистр
18. REG\_OUT — данные для записи в регистр
19. IP — вход адреса команды
20. IP\_OUT — выход записываемого адреса команды
21. IP\_LOCK — сигнал записи нового адреса команды

## Устройство управления

Логика управления устройством разделена на 2 блока — блок управляющего автомата и блок управления командами.

Основным блоком управления микропроцессором является управляющий конечный автомат. Данный автомат координирует работу этапов выполнения команд, а также обеспечивает запуск и остановку выполнения.

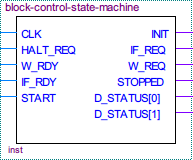


Рисунок 14. Условно-графическое обозначение  
управляющего конечного автомата.

Входы и выходы блока:

1. CLK — тактовый сигнал
2. START — сигнал запуска
3. HALT\_REQ — сигнал запроса остановки
4. INIT — выход инициализации, на данный выход при запуске микро-ЭВМ подается единичный сигнал, который сбрасывается в 0 при начале работы, что позволяет инициализировать различные блоки перед работой.
5. STOPPED — сигнал того, что процессор остановлен
6. IF\_REQ — выход запроса для блока загрузки инструкций
7. IF\_RDY — сигнал готовности блока загрузки инструкций
8. W\_REQ — выход запроса для блока записи — с данного запроса начинается выполнение загруженной команды.
9. W\_RDY — вход готовности блока записи, означает конец выполнения текущей инструкции.

Изначально блок находится в состоянии простоя. Когда на вход подается сигнал START, блок вызывает сначала блок чтения инструкций из ПЗУ, а затем блок записи, что приведет к чтению аргументов, выполнению операции АЛУ, записи данных в память и обновлению регистра IP. В случае, если на вход поступает команда остановки, после выполнения инструкции блок переходит в состояние останова, из которого процесс выполнения может быть продолжен повторной отправкой сигнала START. Если команды остановки не поступает, блок переходит к этапу чтения инструкции и далее выполняет следующую инструкцию.

Блок управления командами устанавливает режимы работы блоков с помощью установки нужных команд и флагов в зависимости от текущей выполняемой команды.

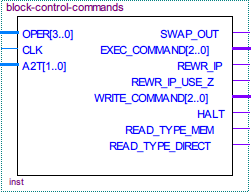


Рисунок 15. Условное графическое отображение блока управления командами

На вход блока подаются тактовый сигнал, код операции и тип второго аргумента команды. Выходы соответствуют управляющим входам блоков загрузки и записи данных, а также выполнения команд. Соответствие состояний входа и выхода приведено в таблице ниже. Обновление состояния выходов происходит через 1 такт после смены входов, по фронту тактового сигнала.

Таблица 5. Команды блоков в зависимости от исполняемой команды

| Код операции | Команда | READ\_TYPE\_MEM | READ\_TYPE\_DIRECT | SWAP\_OUT | EXEC\_COMMAND | WRITE\_COMMAND | REWR\_IP | REWR\_IP\_USE\_Z | HALT |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0000 | HLT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0001 | NOP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0010 | MOV | + | + | 1 | 0 | 1 | 0 | 0 | 0 |
| 0011 | MOVM | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 |
| 0100 | PUSH | + | + | 1 | 0 | 3 | 0 | 0 | 0 |
| 0101 | POP | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| 0110 | JMP | + | + | 0 | 0 | 0 | 1 | 0 | 0 |
| 0111 | JNZ | + | + | 0 | 0 | 0 | 1 | 1 | 0 |
| 1000 | SRL | + | + | 0 | 1 | 1 | 0 | 0 | 0 |
| 1001 | SUB | + | + | 0 | 2 | 1 | 0 | 0 | 0 |
| 1010 | AND | + | + | 0 | 3 | 1 | 0 | 0 | 0 |
| 1011 | NOTZ | + | + | 0 | 4 | 1 | 0 | 0 | 0 |

Запись «+» означает зависимость выходных флагов от типа второго аргумента команды.

## Арбитр

Арбитр используется для определения приоритетного ведущего устройства на системной шине и построен по децентрализованной кольцевой схеме. Принцип арбитража заключается в передаче маркера по кольцу моделей арбитража. Если ведущее устройство запрашивает доступ к шине и у модуля арбитра на входе установлен сигнал наличия маркера, то арбитр переходит в состояние захваченного маркера и выдает разрешение использования шины, в ином случае (маркера нет), арбитр остается в состоянии простоя. Если арбитр получает маркер и ведущему устройству не требуется шина, он передает маркер дальше по цепочке.

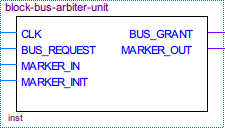


Рисунок 16. Условное графическое обозначение модуля арбитража

Входы и выходы блока:

1. CLK — тактовый сигнал
2. BUS\_REQUEST — сигнал запроса шины
3. BUS\_GRANT — сигнал подтверждения шины
4. MARKER\_INIT — начальное значение маркера (при инициализации арбитра)
5. MARKER\_IN — флаг входящей передачи маркера
6. MARKER\_OUT — флаг исходящей передачи маркера

Флаг начального значение маркера нужен для определения того модуля арбитра, у которого маркер будет в начале процесса арбитража.

# ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ

В данном разделе приведены временные диаграммы функционального моделирования отдельных модулей и всего устройства в целом.

Следует отметить, что в проектируемом устройстве конечные автоматы фактически изменяют (защелкивают) свое состояние по фронту тактового сигнала, но изменяют выходы по его спаду, что создает задержку в 1 такт между соседними состояниями и зависимыми состояниями разных автоматов, но также увеличивают стабильность работы путем создания фаз «установки/использования» сигналов и снижения помех переключения.

## Блок ПЗУ

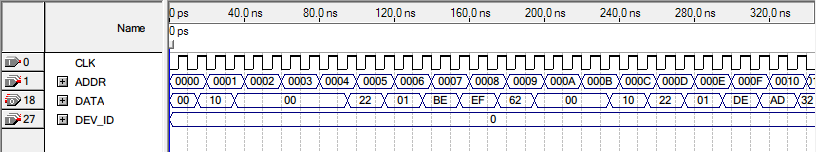


Рисунок 12. Функциональное моделирование ПЗУ

В ПЗУ по адресу 0x0004 располагается инструкция 0x2201BEEF. Из диаграммы функционального моделирования видно, что между выставлением адреса и получением данных присутствует задержка в 1,5 такта. Это связано с тем, что блок lpm\_ram\_dq защелкивает в буфер адрес по фронту тактового сигнала, а затем через 1 такт выставляет данные. Таким образом, между выставлением адреса на шину и считыванием данных должно проходить 2 такта (при условии, что и выставление адреса, и запись данных в буфер происходят по спаду тактового сигнала).

## Блок ОЗУ

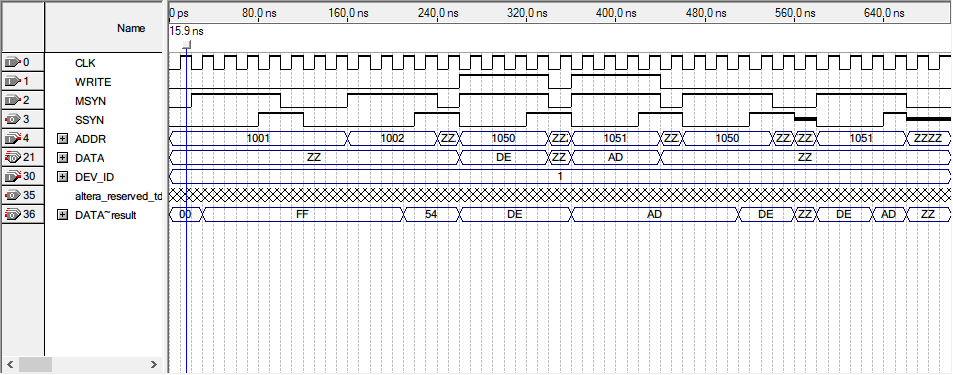


Рисунок 13. Функциональное моделирование ОЗУ

Из диаграммы функционального моделирования видно, как работает асинхронный протокол обмена. Сигналы ADDR и WRITE выставляются ведущим устройством одновременно с выставлением MSYN в «1», так как фактическое обновление состояния конченого автомата, управляющего работой ОЗУ, происходит по фронту тактового сигнала, а все обновления выходных сигналов происходят по спаду тактового сигнала.

Так как старшие 4 бита шины адреса определяют подключенное устройство, на диаграмме видны моменты, где отсутствие установленных старших 4 бит адреса отключает ОЗУ от шины, что приводит к неопределенному уровню сигнала.

Средняя задержка между запросом данных из памяти и готовностью памяти составляет 4 такта (следует учитывать, что данная задержка выражается в тактах сигнала тактовой частоты памяти, которая может отличаться от частоты процессора).

## Регистры общего назначения

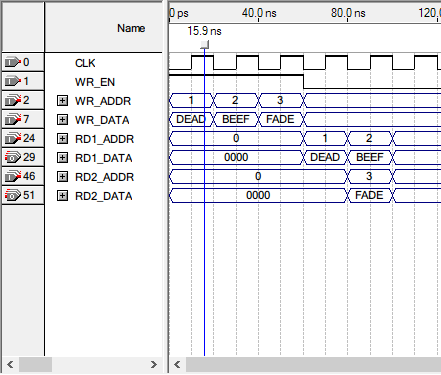


Рисунок 14. Функциональное моделирование блока РОН.

В течение первых трех тактов в регистры записываются 2-байтовые значение, далее они считываются. Так как у блока регистров есть два порта чтения данных, демонстрируется одновременное чтение сразу из обоих портов.

## Стек

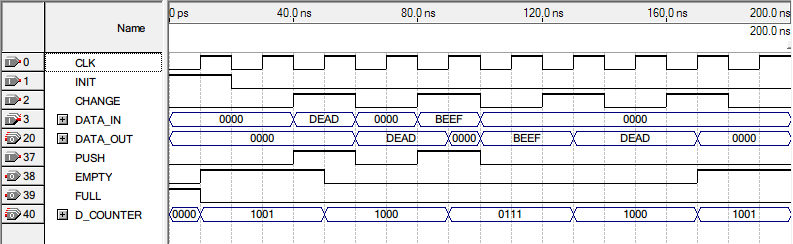


Рисунок 13. Функциональное моделирование стека.

Из диаграммы функционального моделирования можно видеть, как по фронту тактового сигнала меняется состояние стека, при этом выходы EMPTY и FULL отображают соответственно состояния пустого и заполненного стека. Момент изменения состояния выходов стека не соответствует принятому в проекте алгоритму обновления состояния блоков (изменение выводов по спаду сигнала), но, так как при записи в стек чтение из него выполняться не может (на одном и том же этапе одной и той же команды), данный факт не повлияет на общую работу схемы.

## Арбитр

Проведено моделирование для случая двух узлов арбитра в кольце арбитража. При наличии маркера приоритет получает то устройство, которое принимает маркер в момент принятия решения арбитра. Можно увидеть, как узлы обмениваются маркерами при отсутствии запросов, а также меняют приоритетные устройства при наличии нескольких последовательных запросов от обоих устройств сразу, что выравнивает шансы устройств занять шину.

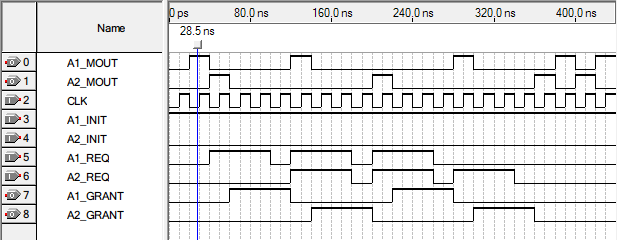


Рисунок 14. Функциональное моделирование работы арбитра

## Кэш данных

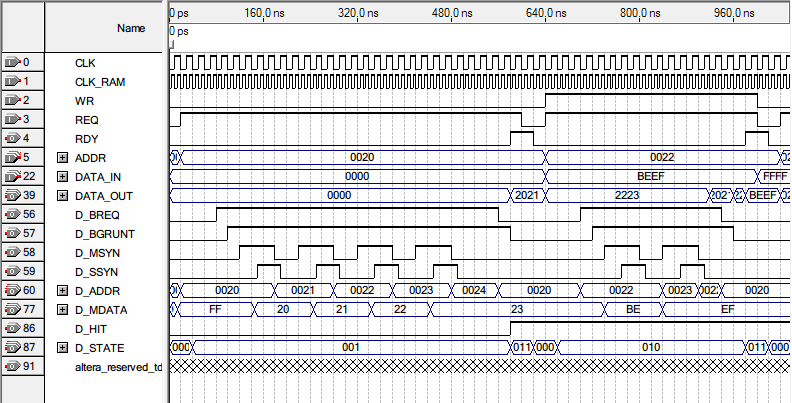


Рисунок 15. Функциональное моделирование кэша данных.

Промах чтения и попадание записи

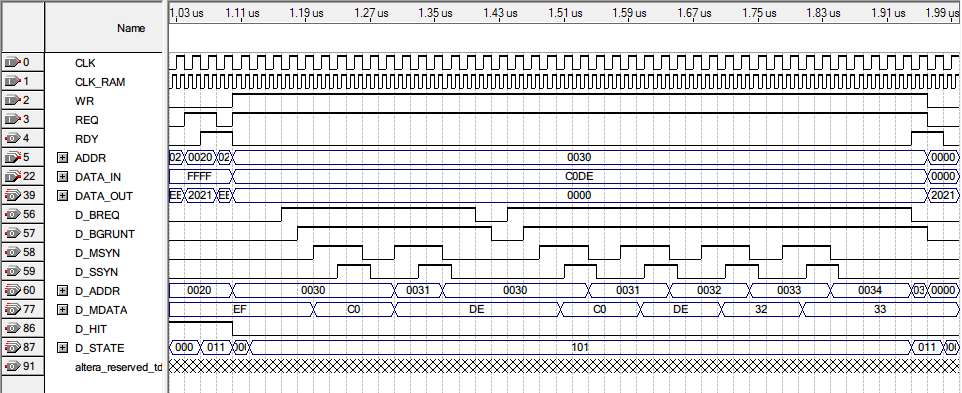


Рисунок 16. Функциональное моделирование кэша данных.

Попадание чтения и промах записи

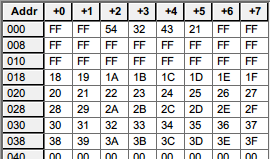


Рисунок 17. Дамп ОЗУ (до моделирования)

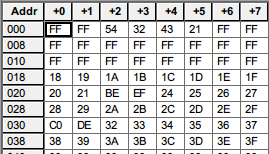


Рисунок 18. Дамп ОЗУ (после моделирования)

Легко видеть, что запросы к памяти занимают довольно много тактов из-за необходимости запроса шины и загрузки 2/4 байт данных из ОЗУ по асинхронному протоколу. Недостатком кэша является то, что при промахе записи не только записываются 2 байта в память, но и считываются все 4 байта памяти, соответствующие 2 16-битным словам, а не одно, которого нет в кэше. Объясняется это несовершенством реализации блока считывания строки кэша из памяти, который может считывать только все 4 байта, но не отдельные 2.

## Тестовая программа

Для тестирования работы всей микро-ЭВМ в целом необходима тестовая программа, которая позволяет продемонстрировать работоспособность основных функций устройства. Ниже приведен листинг подобной программы.

Таблица 6. Листинг тестовой программы

(H — номер регистра/стек, (0x1010) — адрес памяти)

| Адрес | Команда | Код опер. | Арг. 1 | Арг. 2 | Машинный код |
| --- | --- | --- | --- | --- | --- |
| 0x0000 | NOP | 0x1 | — | — | 0x10000000 |
| 0x0004 | MOV 1H, 0xBEEF | 0x2 | 0x1 | 0xBEEF | 0x2201BEEF |
| 0x0008 | JMP 0x0010 | 0x6 | — | 0x0010 | 0x62000010 |
| 0x000C | MOV 1H, 0xDEAD | 0x2 | 0x1 | 0xDEAD | 0x2201DEAD |
| 0x0010 | MOVM 1H, 0x0000 | 0x3 | 0x1 | 0x0000 | 0x32010000 |
| 0x0014 | MOV 2H, (0x0002) | 0x2 | 0x2 | (0x0002) | 0x21020002 |
| 0x0018 | MOV 3H, (0x0004) | 0x2 | 0x3 | (0x0004) | 0x21030004 |
| 0x001C | SUB 2H, 3H | 0x9 | 0x2 | 0x3 | 0x90020003 |
| 0x0020 | MOVM 2H, 0x0006 | 0x3 | 0x2 | 0x0006 | 0x32020006 |
| 0x0024 | PUSH 2H | 0x4 | — | 0x2 | 0x40000002 |
| 0x0028 | JNZ 0x0034 | 0x7 | — | 0x0034 | 0x72000034 |
| 0x002C | MOV 2H, 0xC0DE | 0x2 | 0x2 | 0xC0DE | 0x2202C0DE |
| 0x0030 | JMP 0x0038 | 0x6 | — | 0x0038 | 0x62000038 |
| 0x0034 | MOV 2H, 0xB00B | 0x2 | 0x2 | 0xB00B | 0x2202B00B |
| 0x0038 | MOVM 2H, 0x0008 | 0x3 | 0x2 | 0x0008 | 0x32020008 |
| 0x003C | POP 2H | 0x5 | 0x2 |  | 0x50020000 |
| 0x0040 | MOVM 2H, 0x0010 | 0x3 | 0x2 | 0x0010 | 0x32020010 |
| 0x0044 | MOV 1H, 0x8000 | 0x2 | 0x1 | 0x8000 | 0x22018000 |
| 0x0048 | MOV 2H, 0x0008 | 0x2 | 0x2 | 0x0004 | 0x22020004 |
| 0x004C | SRL 1H, 2H | 0x8 | 0x1 | 0x2 | 0x80010002 |
| 0x0050 | MOVM 1H, 0x0012 | 0x3 | 0x3 | 0x0012 | 0x32010012 |
| 0x0054 | MOV 2H, 0xA5A5 | 0x2 | 0x2 | 0xA5A5 | 0x2202A5A5 |
| 0x0058 | SUB 1H, 1H | 0x9 | 0x1 | 0x1 | 0x90010001 |
| 0x005C | NOTZ 2H | 0xB | 0x2 | — | 0xB0020000 |
| 0x0060 | MOVM 2H, 0x0014 | 0x3 | 0x2 | 0x0014 | 0x32020014 |
| 0x0064 | MOV 3H, 0xF0F0 | 0x2 | 0x3 | 0xF0F0 | 0x2203F0F0 |
| 0x0068 | AND 2H, 3H | 0xA | 0x2 | 0x3 | 0xA0020003 |
| 0x006C | MOVM 2H, 0x0016 | 0x3 | 0x2 | 0x0016 | 0x32020016 |
| 0x0070 | HLT | 0x 0 | — | — | 0x00000000 |

Команда NOP по 0x000 служит для проверки собственно команды NOP. Команды (по адресам) 0x0004—0x0010 служат для проверки MOV, JMP и MOVM. В случае, если они работают правильно, в ОЗУ по адресу 0x0000 (здесь и далее — 16-битное значение находится в указанном байте и следующем, при этом младший байт значения находится в старшем байте ОЗУ) будет находится значение 0xBEEF. Команды 0x0014—0x0020 служат для проверки вычитания — если оно работает, то по адресу 0x0006 в ОЗУ будет значение 0x1111. Команды 0x0020, 0x0028—0x0038 служат для проверки команды JNZ, если она работает, то по адресу 0x0008 в ОЗУ будет значение B00B. Команды 0x0024 и 0x003C-0x0040 служат для проверки операций стека, при этом в ОЗУ должно оказаться значение 0x1111 по адресу 0x0010. Команды 0x0044—0x0050 — проверка SRL, в ОЗУ — 0x0800 по адресу 0x0012. Команды 0x0054—0x0060 — проверка NOTZ, в ОЗУ — 0x5A5A по адресу 0x0014. Команды 0x0054—0x006C — проверка AND, в ОЗУ — 0x5050 по адресу 0x0016. Проверка HLT по адресу 0x0070 — процессор должен остановить работу программы.

Некоторые из диапазонов тестовых команд пересекаются, так как данные одной тестовой последовательности можно использовать в другой.

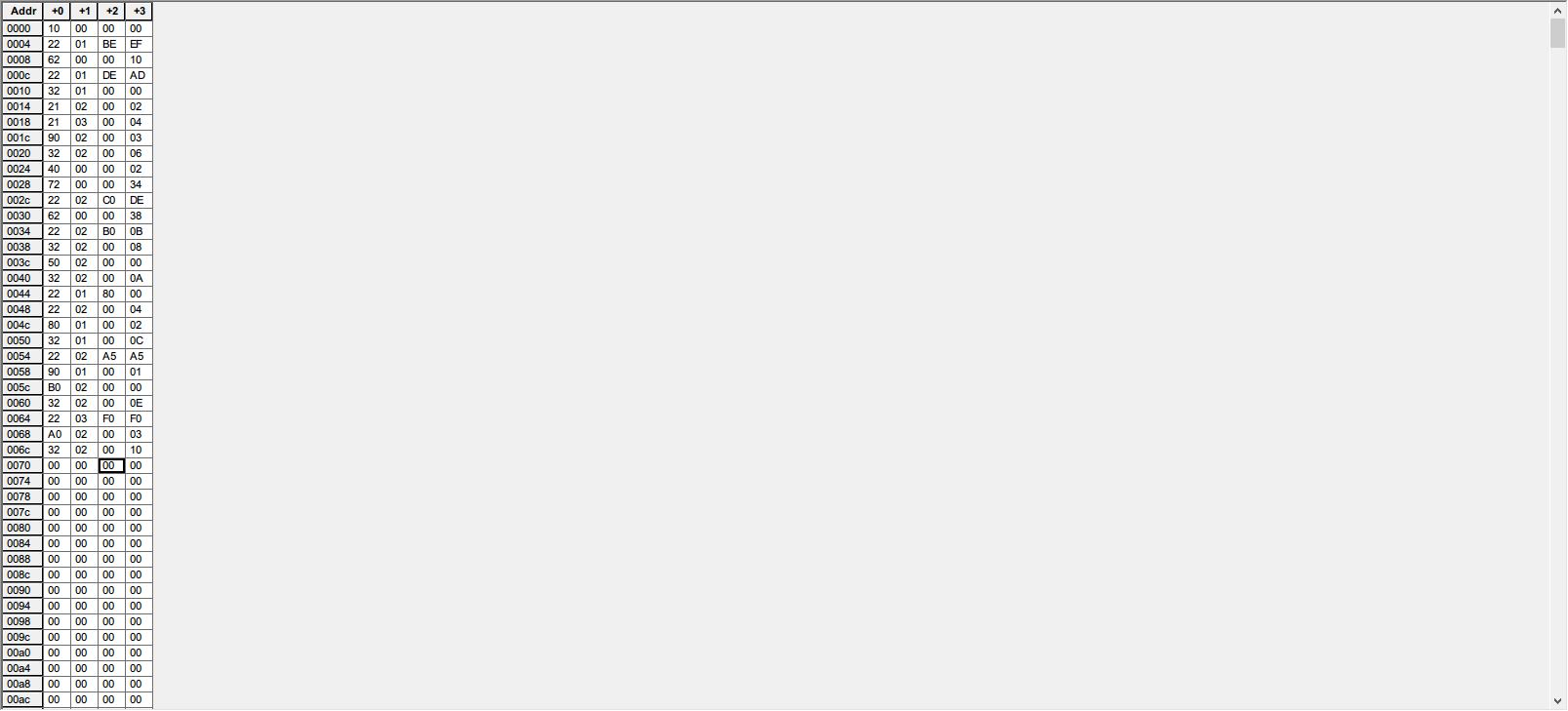
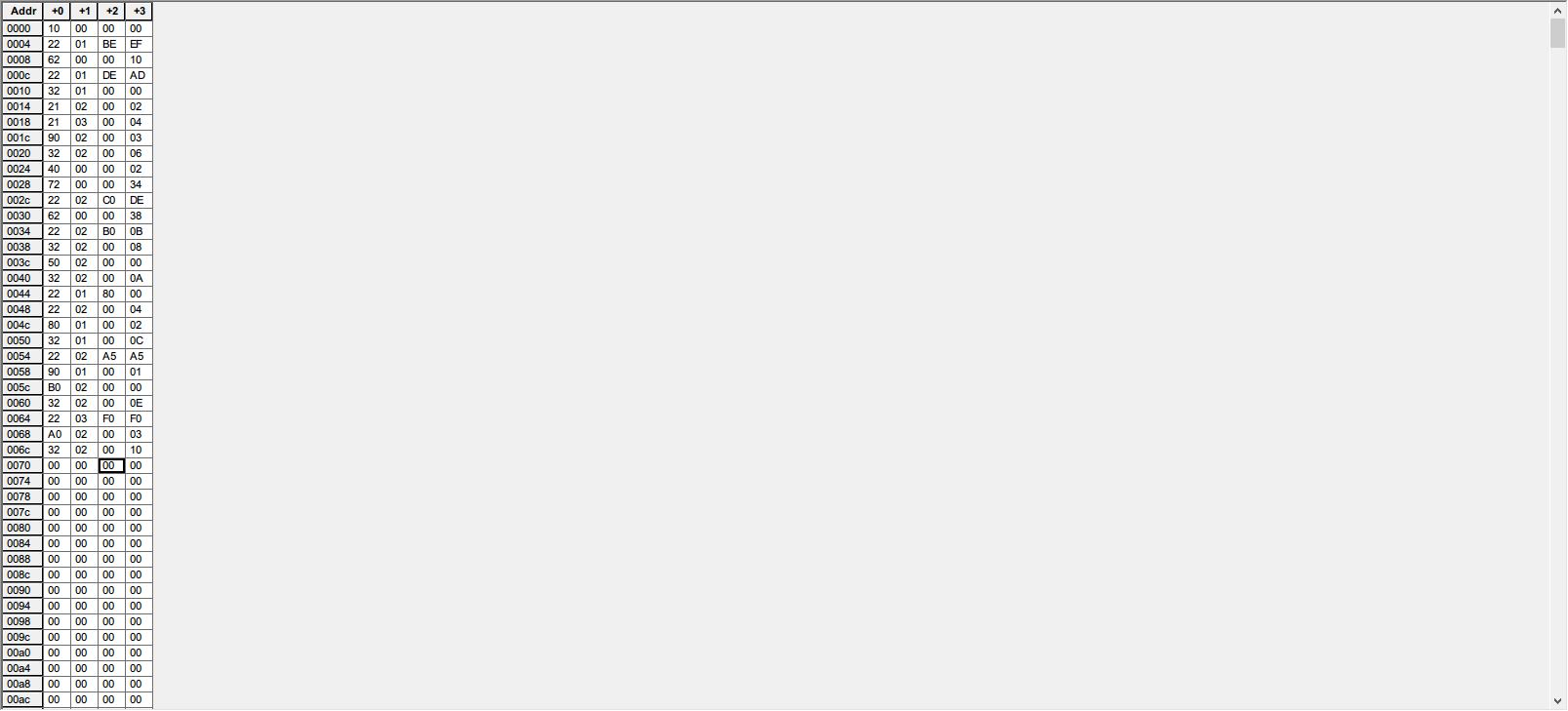
 

Рисунок 17. Дамп программы в ПЗУ

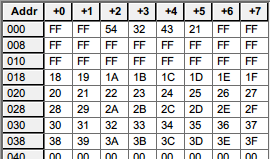


Рисунок 18. Дамп данных в ОЗУ (до моделирования)

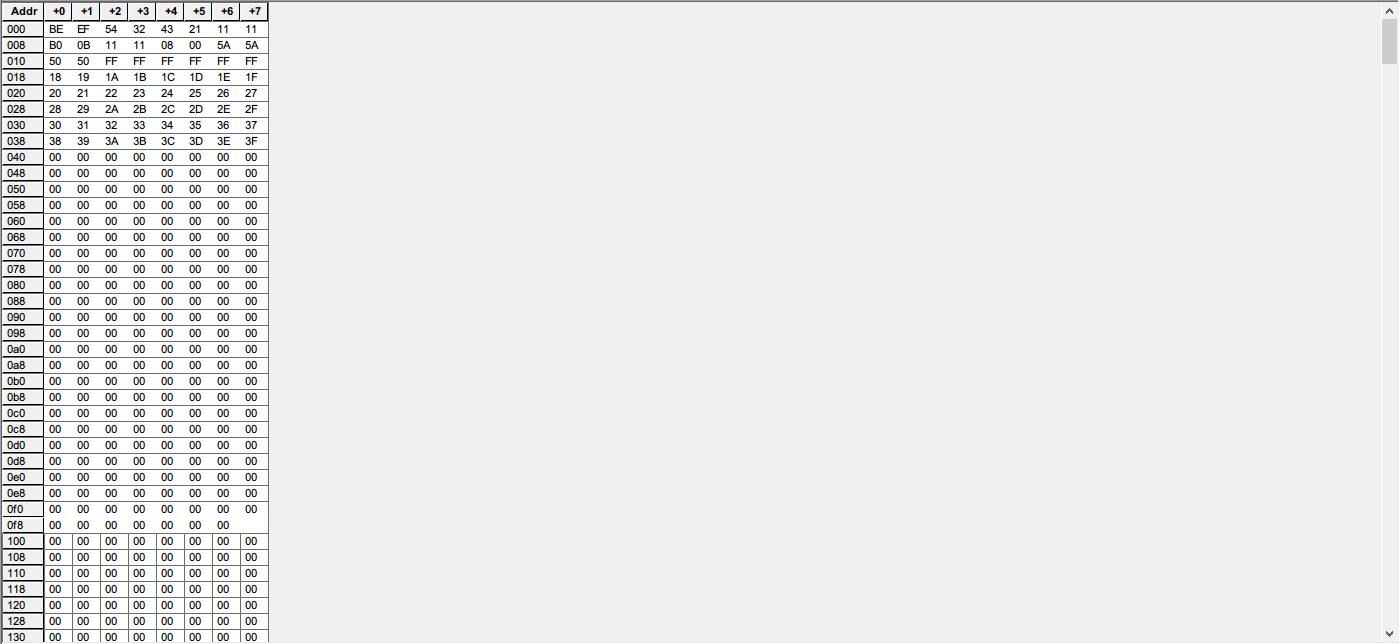


Рисунок 19. Дамп данных в ОЗУ (после моделирования)

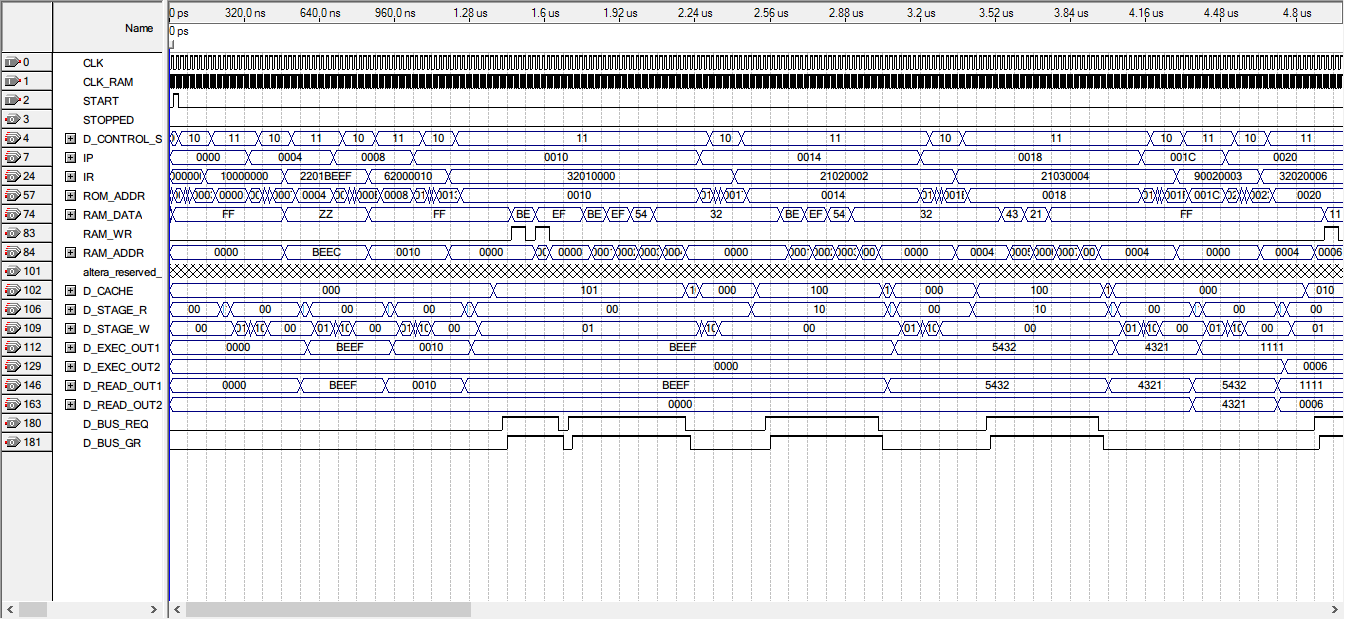


Рисунок 20. Функциональное моделирование работы микро-ЭВМ

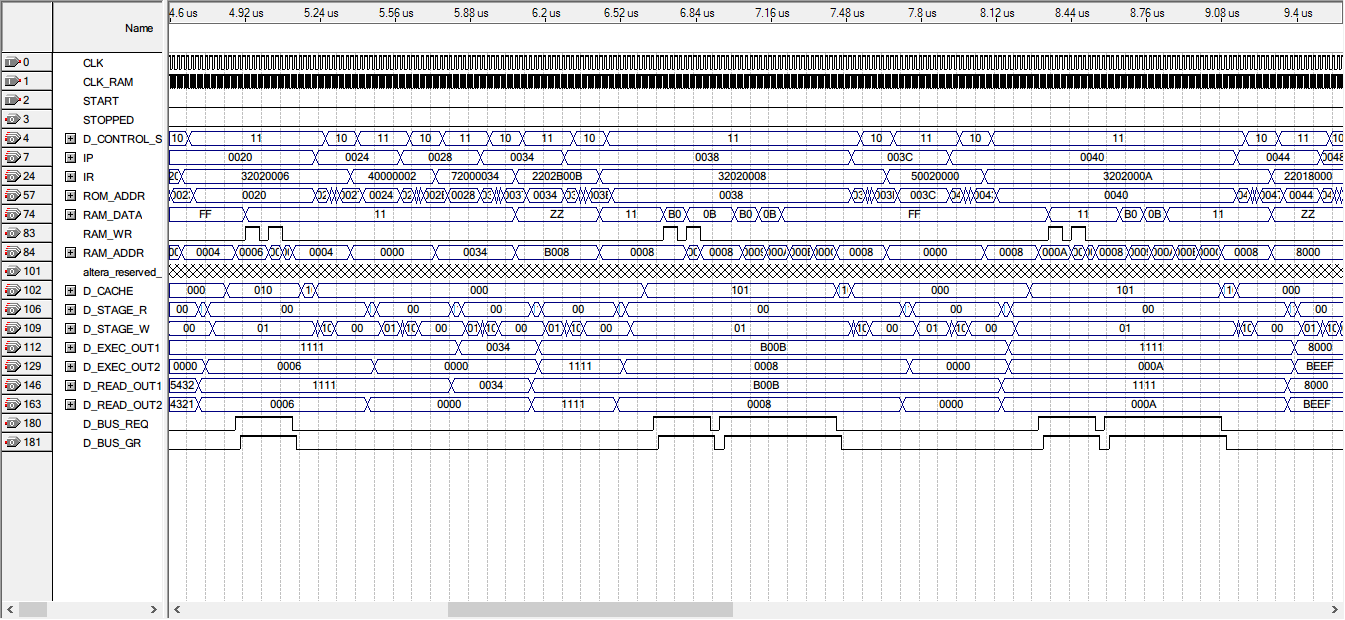


Рисунок 21. Функциональное моделирование работы микро-ЭВМ(продолжение).

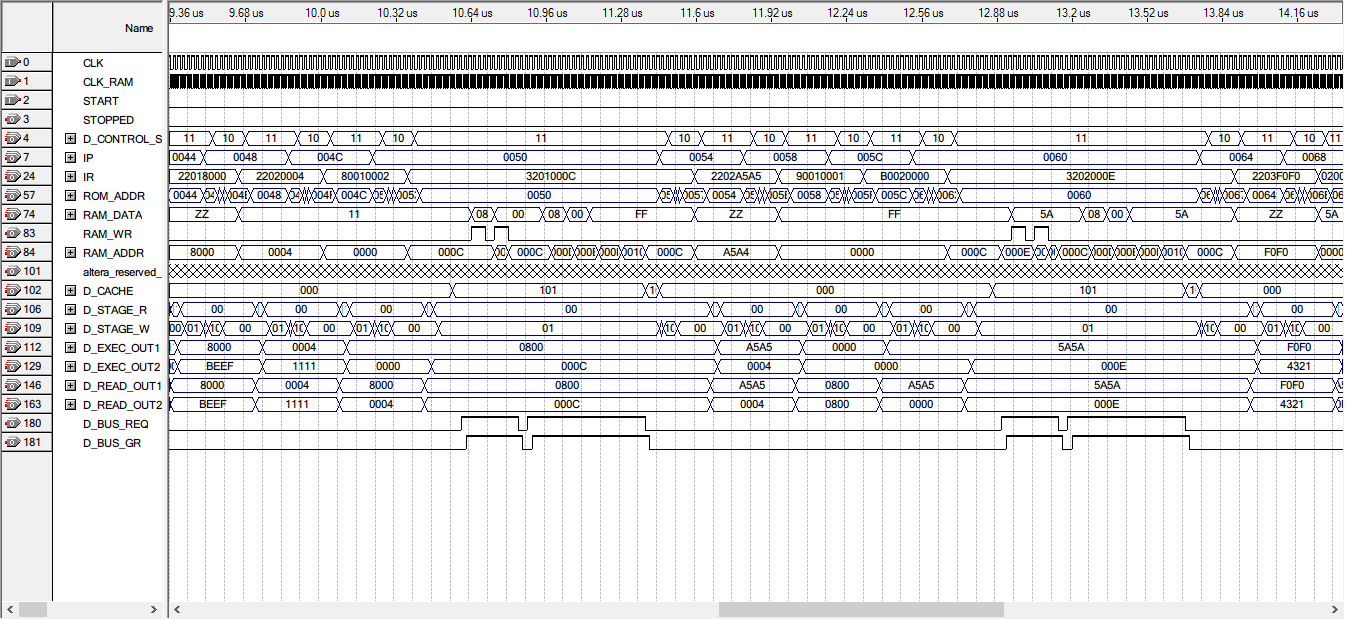


Рисунок 22. Функциональное моделирование работы микро-ЭВМ (продолжение)

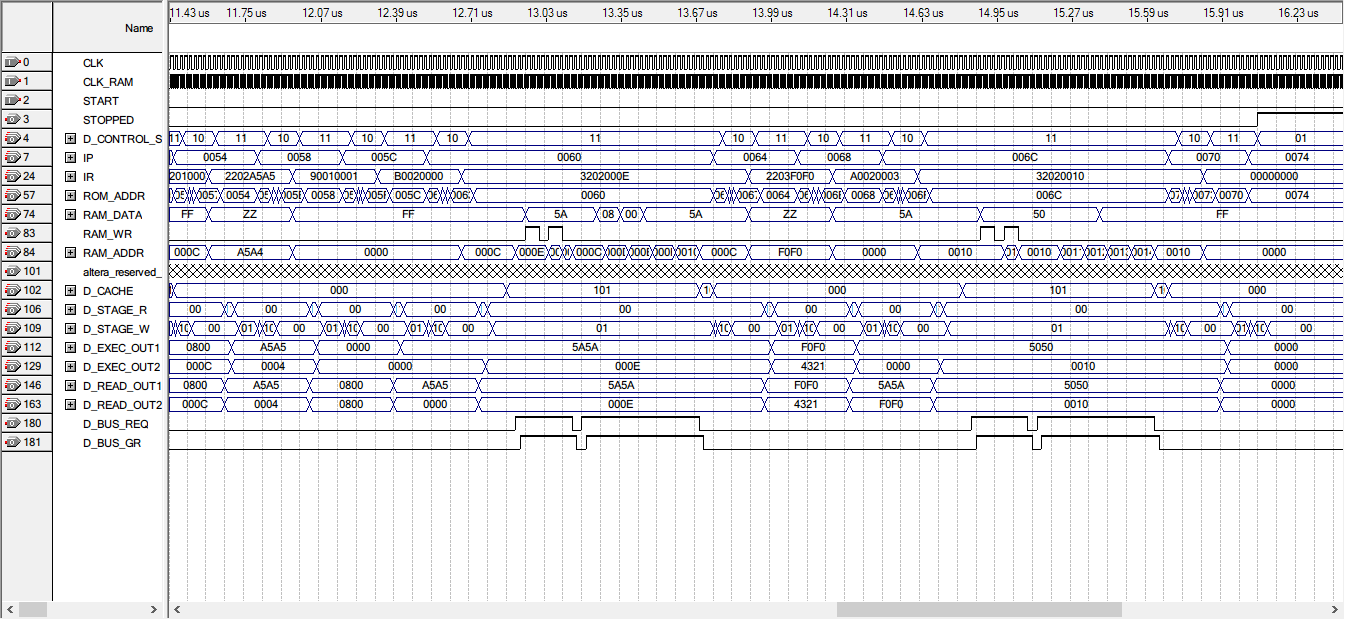


Рисунок 23. Функциональное моделирование работы микро-ЭВМ(продолжение)

# АНАЛИЗ И ОПТИМИЗАЦИЯ МИКРО-ЭВМ

В ходе работы была спроектирована микро-ЭВМ, способная выполнять 11 команд с возможностью использования прямой, прямой регистровой и непосредственной адресации.

Анализ диаграмм функционального моделирования показывает, что разработанная микро-ЭВМ плохо оптимизирована, так как даже самая короткая команда NOP занимает 19 тактов. Самая длинная команда в моделировании занимает 61 такт. Для оптимизации производительности можно принять следующие шаги:

* Конвейеризация процессора. Данный метод будет наиболее эффективным, так как архитектура уже частично приспособлена для конвейеризации — отдельные этапы возможного конвейера выполнены отдельными блоками, асинхронный протокол позволяет гибко управлять процессом. При введении 4 стадий конвейера (согласно текущим этапам выполнения команд) можно добиться пропускной способности до 3-4 раз превышающей текущую. Однако внедрение конвейеризации потребует значительного усложнения логики управления как процессором, так и отдельных блоков. В частности, необходимо будет добавлять логику разрешения конфликтов при чтении/записи в регистры, стек, кэш.
* Оптимизация обращений к ОЗУ. Наибольшее время требуется для загрузки/записи данных в ОЗУ, что объясняется использованием 16-битных слов при 8-битной шине данных, а также несовершенством кэша и асинхронным протоколом памяти. Наиболее эффективным решением является расширение шины данных до 16 бит, что, однако, является серьезным архитектурным изменением и не приемлемо по заданию.
* Оптимизация загрузки команд. Загрузка команд требует четырехкратного обращения к ПЗУ, что в некоторых случаях приводит к тому, что загрузка требует больше времени, чем исполнение команды (в частности, NOP и HLT). Оптимальными методами решения будет кэширование команд либо изменение формата команд. Как уже упоминалось, команды спроектированной ЭВМ имеют статический размер, и в некоторых случаях чрезмерную избыточность. Введение изменяемого размера команд уменьшит число обращений к ПЗУ (из 12 команд 7 не требуют обоих операндов). Кэширование также позволит сократить число обращений к ПЗУ.
* До 4 тактов уходит на выполнение промежуточных переходов конечных автоматов в каждом блоке цикла выполнения команд. В то время как асинхронный протокол позволяет гибко варьировать тайминги взаимодействия блоков, он довольно сильно сказывается на пропускной способности коротких операций. Оптимизация автоматов либо частичный отказ от их использования в некоторых блоках позволит сократить время ожидания готовности блоков, однако снизит гибкость логики блоков, а также повысит сложность поддержки архитектуры, потому данный вариант является менее предпочтительным.
* КПДП. Спроектированное устройство не включает в себя КПДП, что требует использования процессора для пересылки данных между периферийными устройствами. Так как архитектура поддерживает арбитраж шин, реализация КПДП позволит снизить нагрузку на процессор при работе с периферией и ОЗУ, особенно с учетом использования гарвардской архитектуры, что позволяет выполнять команды без обращения к системной шине вообще (при использовании регистровой/непосредственной адресации для всех аргументов).

# ЗАКЛЮЧЕНИЕ

В ходе работы реализована полноценная микро-ЭВМ, основными характеристиками которой являются:

* Гарвардская архитектура
* 16-битная адресация, 8-битные шины данных данных/команд
* Синхронное ПЗУ и асинхронное ОЗУ
* 12 16-битных РОН
* Поддержка прямой, регистровой и непосредственной адресации
* Наличие аппаратного стека на 9 16-битных слов
* Кэширование данных
* Асинхронная системная шина с арбитражем
* Модульное построение схемы
* Отсутствие конвейеризации выполнения команд

Кроме того, данная микро-ЭВМ обладает внутренней модульной архитектурой, возможностью расширения системы команд за счет использования резервных полей команд, комбинирования команд блоков и достаточно простого ввода в аппаратную архитектуру новых операций, возможностью настройки объема памяти и ее расположения в адресном пространстве, а также имеет перспективу в оптимизации производительности.