СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках данного курсового необходимо реализовать микро-ЭВМ на ПЛИС согласно заданным требованиям. Краткое описание основных архитектурных свойств приведено ниже.

Архитектура. Гарвардская архитектура – одна из наиболее распространённых типов архитектуры ЭВМ на сегодняшний день. Главной её особенностью является физическое разделение команд и данных. Они хранятся в разных запоминающих устройствах. Зачастую, такую архитектуру имеет кэш «L1» в микропроцессорах. Шины для доступа к командам и данным, соответственно, тоже разные. Такой подход позволяет одновременно обмениваться необходимой информацией, следовательно, повышается быстродействие системы в целом.

Запоминающие устройства. Для хранения данных используется RAM (ОЗУ) синхронного типа. Объем памяти и ширина машинного слова отражены в листе задания. Команды хранятся в ROM (ПЗУ) синхронного типа. Выбор ширины команды будет рассмотрен далее. Синхронный тип устройства предполагает фиксированный промежуток времени между подачей данных на вход и получением результата. Устройства такого типа работают медленнее, чем устройства асинхронного типа, которые выдают результат по мере его готовности.

УУ. Работа микропроцессора осуществляется по «циклам». За один цикл процессор выполняет одну команду. Такая архитектура менее производительна, чем выполнение команд на конвейере, однако более проста в реализации.

Стэк. Аппаратный стек позволяет хранить данные по типу структуры данных «Стэк». Данное устройство повышает быстродействие некоторых алгоритмов, так как не нужно реализовывать стек в памяти.

АЛУ. Для выполнения арифметических, логических, а также сдвиговых команд микро-ЭВМ использует арифметико-логическое устройство. Данные типы команд являются основными для вычислительной машины.

РОН. Выполняя команды микро-ЭВМ должна получить входные данные команды — операнды. Некоторые команды предполагают выборку операндов из ОЗУ. Для этого они пересылаются в регистры общего назначения, где происходит их дальнейшая обработка.

Подробное описание всех узлов ЭВМ представлено в следующих разделах.

1 РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ

В данном разделе приводится функциональный состав разрабатываемой микро-ЭВМ, разработка и описание системы команд, а также описание взаимодействия всех блоков при выполнении команд программы.

1.1 Функциональный состав микро-ЭВМ

Все ЭВМ имеют определённый набор функциональных блоков. Обязательными являются запоминающие устройства и блок выполнения операций. Наличие остальных блоков зависит от их необходимости в данной модификации.

Функциональные блоки:

- центральный процессор;
- запоминающие устройства;

Структурная схема микро-ЭВМ представлена в приложении А.

Рассмотрим каждый блок подробно.

1.1.1 Центральный процессор

Для реализации ЦП было решено использовать устройство последовательного выполнения команд. Данное устройство идентифицирует этапы выполнения команд, а также управляет ими. Всего имеется 6 этапов:

- чтение команды;
- декодирования команды;
- разрешения адресов операндов;
- чтение операндов;
- выполнение команды;
- запись результата.

Управление всем циклом процессора лежит на устройстве управления (УУ). В нём находятся:

- указатель команд;
- буфер команд;
- счётчик и дешифратор цикла;
- реализация команд JMP, JNZ и HLT.
- комбинационная логика по выделению тактов цикла командам.
- комбинационная логика разрешения косвенных адресов.
- комбинационная логика управления прямым доступом к памяти.
- комбинационная логика завершения команд.

Устройство управления посылает управляющие сигналы в АЛУ, РОН, и стек. Это говорит о централизации управления.

В ЦП размещены регистры общего назначения (РОН), стек и АЛУ.

Регистры общего назначения (16 шт) доступны пользователю и имеют размер шины данных (ШД).

Стек имеет направление роста вниз. При помещении данных в стек сначала в текущий регистр помещаются данные, потом увеличивается вершина стека. При извлечении элемента из стека происходит обратное действие: уменьшение значения вершины стека и последующее чтение элемента из вершины. Таким образом, при записи данных в стек значение вершины стека уменьшается. Было принято решение не реализовывать флаг переполнения стека и флаг пустого стека. Это связано с тем, что в данной ЭВМ не предусмотрены команды, позволяющие реагировать на переполнение или попытку извлечения данных их пустого стека.

Арифметико-логическое устройство (АЛУ) содержит регистры загрузки операндов, комбинационную логику, реализовывающую команды INCS, NOTZ и NXOR, а также регистр флагов, позволяющий выполнять условные комманды (JNZ, NOTZ, INCS). В данном устройстве также нет никакой управляющей логики. Всё управление осуществляется через управляющие пины, и исходит из УУ.

1.1.2 Запоминающие устройства

В соответствии с вариантом задания, требуется реализовать архитектуру Гарвардского типа. Соответственно, память команд и данных — это физически разделённые устройства. Соответственно Данные хранятся в RAM синхронного типа. Команды в ROM синхронного типа. Объёмы запоминающих устройств определяются шириной шины адреса.

1.2 Разработка архитектуры системы команд

Для упрощения работы с командами, принято решение использовать фиксированную длину команды в формате, представленном на рисунке 1.1. Такой подход позволит избежать проблем с чтением команд из памяти и индикации каждой из них в общем потоке, а также облегчает дальнейшее расширение системы команд.

КОП	РОН	АДР1	Резерв
5 бит	4 бита	16 бит	23 бит

Рисунок 1.1 — Формат команды

Согласно варианта задания необходимо реализовать 10 команд, 1 из которых имеет 4 варианта работы с операндами (команда MOV), 4 команды – 2 варианта (NXOR, XOTZ, INCS, SRA), 4 команды – 1 вариант (JMP, JNZ, PUSH, POP) и 2 не имеют операндов (NOP, HLT).

Таким образом, фактически необходимо реализовать 18 команд. Для их кодирования потребуется $\lceil \log_2 18 \rceil = 5$ бит.

Реализуемые команды имеют до 1 операнда, причём длина операнда может варьироваться. Максимальные длины операндов соответствуют ширине

шин данных и адреса. Следовательно, для передачи операндов необходимо 16 бит.

Однако, при использовании 5 бит для КО остаётся ещё 32-18 = 14 вариантов команд, которыми можно расширить разнообразие операций.

Архитектура система команд представлена в таблице 1.1. Неиспользуемые биты отмечены знаками X.

Таблица 1.1 — архитектура системы команд

Команда	Адресация	название на схеме	Код комманды	Код регистра	адресс	резерв
			5 бит	4 бит	16 бит	23 бит
NOP		NOP	00000	XXXXXXXXXX	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXX
MOV adr, reg	прямая	MOVARD	00001	1111	1010011001101110	XXXXXXXXXXXXXXX
MOV adr, reg	косвенная	MOVARI	00010	1001	1010011110001101	XXXXXXXXXXXXXXX
MOV reg, adr	прямая	MOVRAD	00011	0010	1000101101101101	XXXXXXXXXXXXXXX
MOV reg, adr	косвенная	MOVRAI	00100	1001	1000101100101011	XXXXXXXXXXXXXXX
INCS reg	прямая регистровая	INCSDR	00101	1000	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXX
INCS adr	косвенная	INCSI	00110	XXXXXXXXXX	1101001010101010	XXXXXXXXXXXXXXX
NOTZ reg	прямая регистровая	NOTZDR	00111	1010	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXX
NOTZ adr	косвенная	NOTZI	01000	XXXXXXXXXX	1001010101010101	XXXXXXXXXXXXXXX
NXOR reg1, reg2	прямая регистровая	NXORDR	01001	1011	1010XXXXXXXXX	XXXXXXXXXXXXXXX
NXOR reg, adr	косвенная	NXORI	01010	1000	1010101010101010	XXXXXXXXXXXXXXX
SRA reg, const	прямая регистровая	SRADR	01011	1111	1001XXXXXXXXX	XXXXXXXXXXXXXXX
SRA reg, addr	косвенная	SRAI	01100	1101	1010101001010101	XXXXXXXXXXXXXXX
PUSH reg	прямая регистровая	PUSH	01101	0110	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXX
POP reg	прямая регистровая	POP	01110	0111	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXX
JMP adr	непосредственная	JMP	01111	XXXXXXXXXX	1010101101010101	XXXXXXXXXXXXXXX
JNZ adr	непосредственная	JNZ	10000	XXXXXXXXXX	1010110010110011	XXXXXXXXXXXXXXX
HLT	XXXXXXXXXXXXXXXX	HLT	10001	XXXXXXXXXX	XXXXXXXXXXXXXX	XXXXXXXXXXXXXXX

При выполнении команд необходимо отслеживать их результат. Одним из способов отслеживания является использование регистра флагов. Регистр флагов необходим и при выполнении команды JNZ, для которой нужен флаг нуля (устанавливается 1, если результат предыдущей команды равен 0).

Из команд, представленных выше, флаг нуля могут устанавливать арифметические команды (NXOR, NOTZ, SRA, INCS). Флаг переноса может установить только команда SRA.

Таким образом, разработана архитектура системы команд, заданная по варианту.

2 РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ

В данном разделе подробно рассмотрено строение основных устройств, из которых состоит микро-ЭВМ. Структурная схема микро-ЭВМ представлена в приложении Б.

2.1 Запоминающие устройства

Как уже упоминалось, для хранения данных и команд используются разные устройства. Рассмотрим их подробнее.

2.1.1 ROM

ROM используется для хранения команд. Объём памяти можно рассчитать, исходя из ширины шин команд и адресов. По варианту память адресуется при помощи 16-ти бит. Такое количество адресов позволяет адресовать 65536 слов памяти. Также, по заданию разрядность машинного слова (разрядность шины данных) составляет 16 бит. Итого имеем 65536*16=1048576 бит на ПЗУ и столько же на ОЗУ. Однако, стандартные установленные средства Quartus II не позволяют работать с такими объемами данных, да и для реализации небольшой программы, охватывающей малый блок памяти, вполне будет достаточно небольшого количества адресов. Так, путём подбора шин адреса, были установлены оптимальные разрядности данных шин: 14 бит для шины адреса данных, и 11 бит для адресов команд. Условно-графическое изображение ROM представлено на рисунке 2.1.

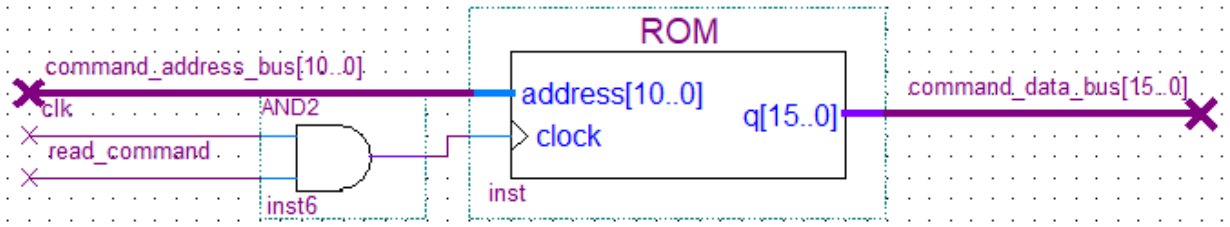

Рисунок 2.1 — УГО ROM

Входы:

- Command_address_bus[10..0] — адрес в памяти;
- clk — тактирующий сигнал

Выход command_data_bus[15..0] предназначен для вывода данных.

2.1.2 RAM

RAM используется для хранения данных. Объём памяти также уменьшен с 16-ти до 14-ти бит, в связи с недоступностью большего количества адресов. Условно-графическое изображение блока RAM представлено на рисунке 2.2.

Входы:

- address[] — адрес в памяти;
- inclock — тактирующий сигнал для записи входных данных;
- outclock — тактирующий сигнал для выдачи данных;
- we — разрешение на запись;

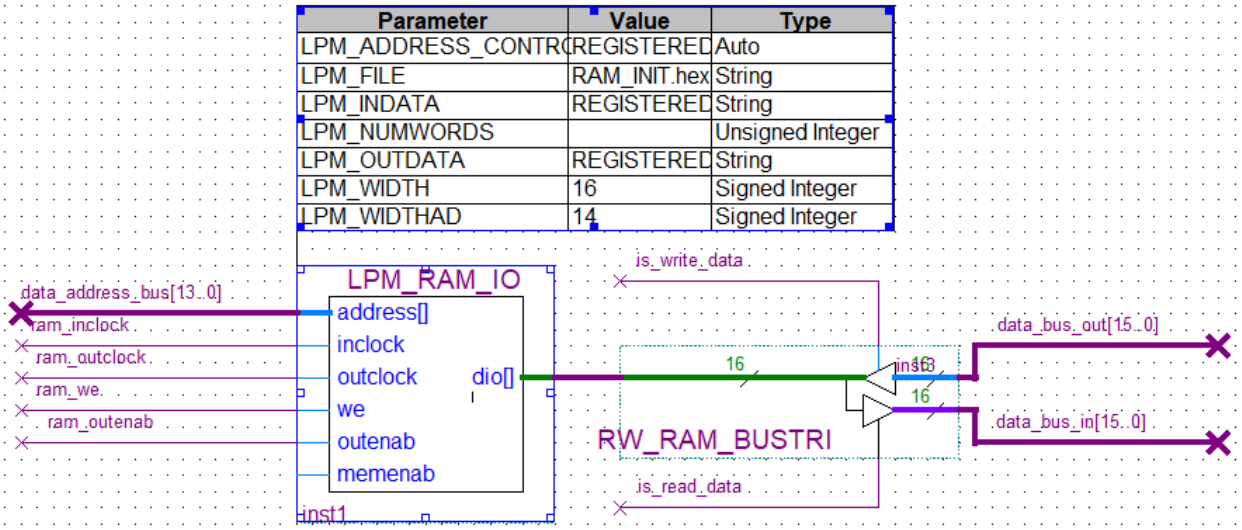

Рисунок 2.2 — УГО RAM

dio[] является двунаправленным входом-выходом для передачи данных. Направление передачи контролируется входными сигналами.

Сигналы is_read_data и is_write_data исходят из устройства управления ЦП и управляют соответственно считыванием/записью.

2.2 Центральный процессор

Центральный процессор содержит в себе:

- УУ;
- стек;
- РОН;
- АЛУ;

Структурная схема ЦП представлена в приложении В. УГО процессора представлено на рисунке 2.3. Рассмотрим каждую из составляющих процессора подробнее.

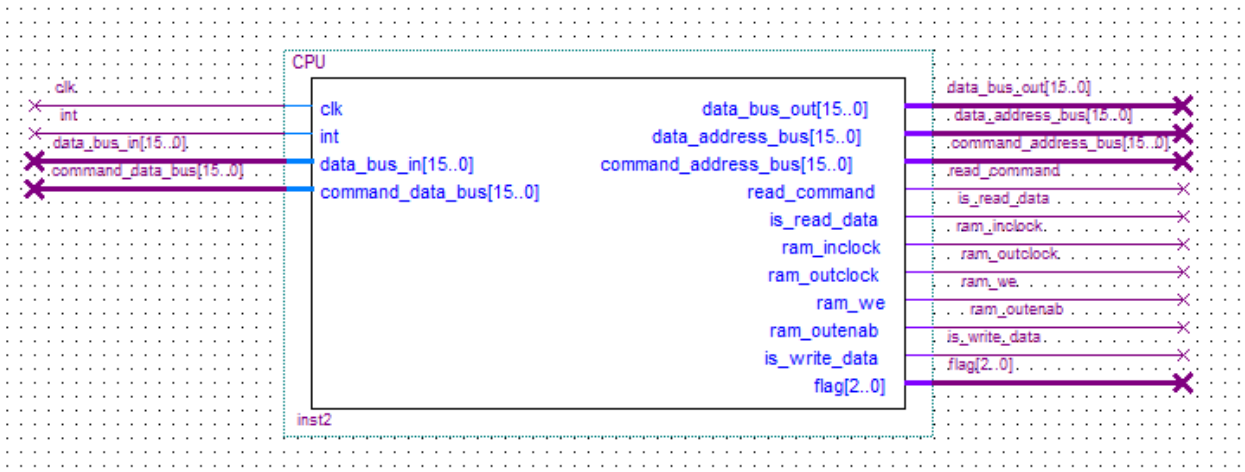

Рисунок 2.3 — УГО центрального процессора

2.2.1 Устройство управления

Устройство управления управляет шестью стадий. Все они выполняются последовательно и для одной команды за раз. Первая стадия длится 5 тактов. Все последующие стадии выполняются специфическое для каждой команды время.

Из устройства управления выходит масса выходных сигналов для управления АЛУ, Стеком, РОН, работой с памятью. УГО устройства управления представлено на рисунке 2.3. Структурная схема УУ представлена в приложении Г.

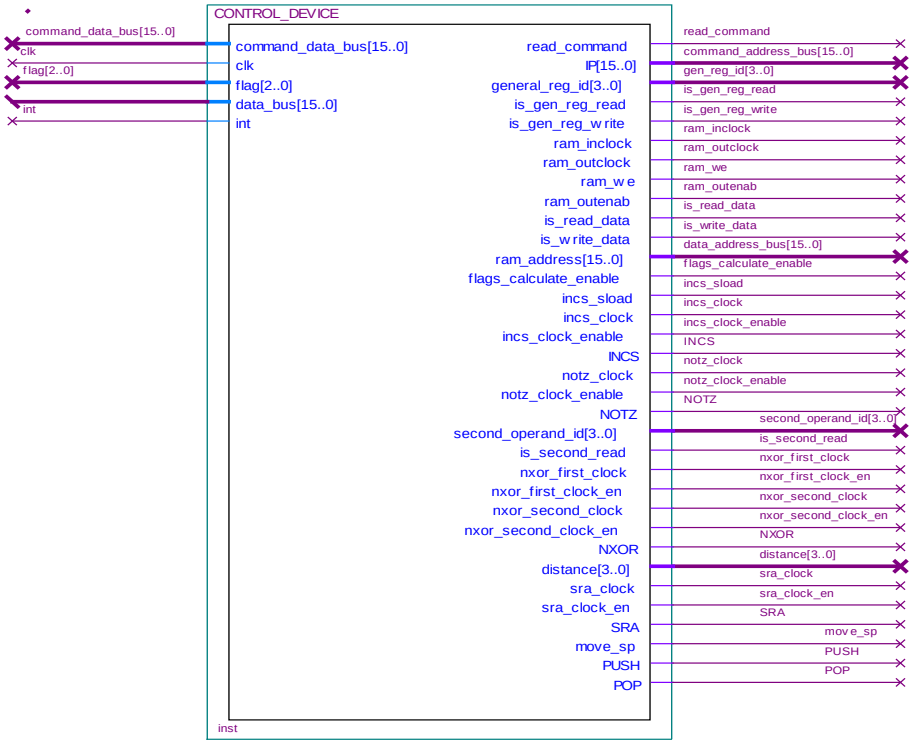

Рисунок 2.3 — УГО устройства управления

2.2.1.1 Блок выборки команды

Первой стадией цикла процессора является стадия выборки команды. Данная стадия реализована в УУ. Делается это при помощи счётчика команд, представленного на рисунке 2.4, а также буфера команд, представленного на рисунке 2.5. В блоке счётчика команд также находятся управляющий сигнал «load_new_address», реализующий инструкции JMP и JNZ. Комбинационная логика данного сигнала представлена на рисунке 2.6. Данный сигнал формируется при помощи управляющего такта «T[5]», являющегося выходом декодера цикла процессора, о котором будет говориться ниже. На рисунке 2.5 видно, что считанная команда отображается в 4 составляющие:

- COP[4..0] — код операции;
- REG[3..0] — номер регистра;
- FADDR[15..0] — адрес первого операнда;
- SDDR[15..0] — адрес второго операнда (пока не используется).

Ввиду выбора статической длины инструкции, шаг считывания команды одинаков для всех команд, и длится до T[5] такта цикла. Реализация сигналов управления считыванием показана на рисунке 2.7.

Данный подход можно слегка оптимизировать, анализируя на код операции первое считанное слово. Тогда инструкции можно было бы писать компактно, не занимая лишних слов в ROM. Однако это требует более сложной реализации логики обработки команды.

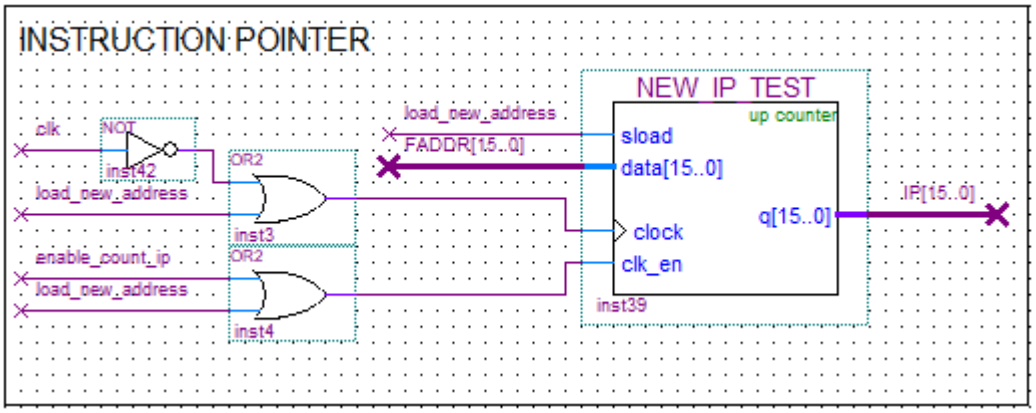

Рисунок 2.4 — Счётчик команд

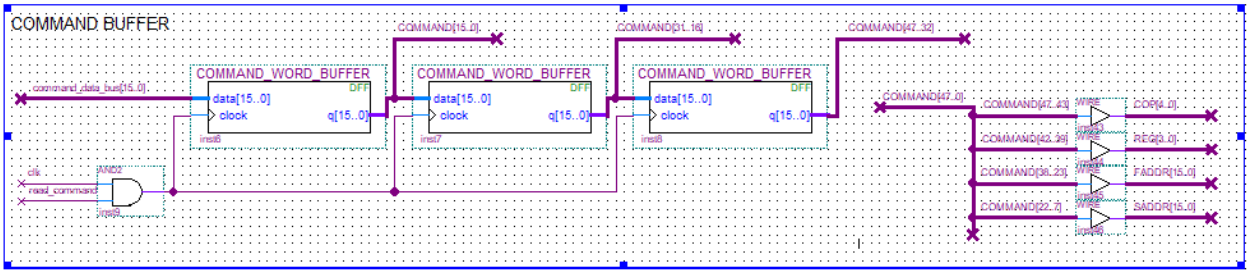

Рисунок 2.5 — буфер команд

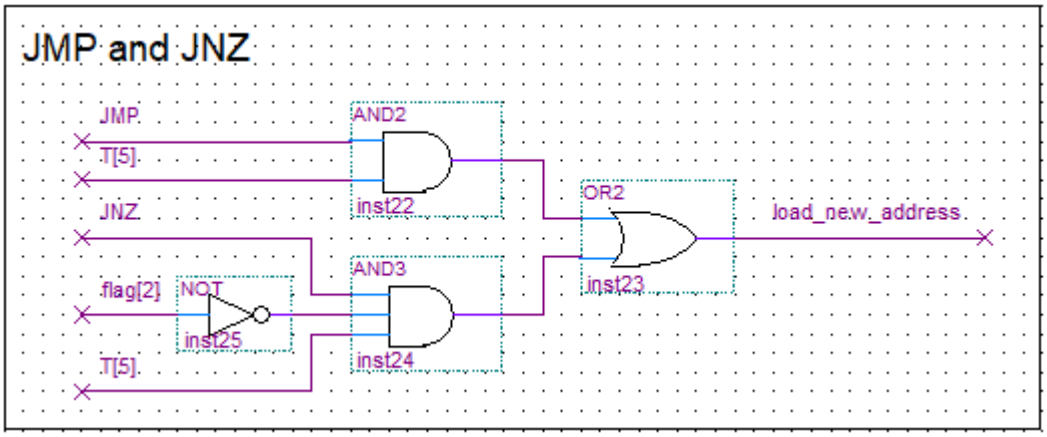

Рисунок 2.6 — реализация JMP и JNZ

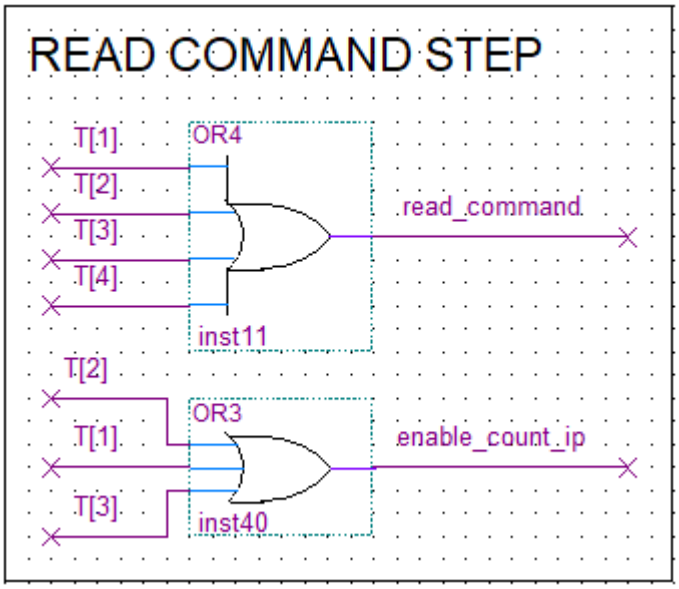

Рисунок 2.7 — Стадия считывания команды

2.2.1.2 Цикл процессора

Для управления циклом процессора, а также идентификации текущей стадии процессора используются счётчик цикла процессора и декодер цикла, представленные на рисунке 2.8. Вместе они входят в блок управления циклом процессора. Данный блок находится в УУ. Прямо в этом блоке реализована команда «HLT». В данной реализации, при считывании из ROM команды HLT этапе исполнения команды счётчик цикла процессора «замораживается», и цикл процессора приостанавливается. Возврат процессора в рабочий режим осуществляется путём поступления асинхронного прерывания «int». Также при такой реализации, регистр заморозки цикла инициализируется единичным значением. Это означает, что процессор начинает свою работу только по прибытию асинхронного прерывания.

Разные команды требуют разной длины цикла процессора для своей работы. Поэтому в счётчик цикла заведён сигнал «cycle_reset», реализация которого приведена на рисунке 2.9. Здесь происходит следующее: при достижении

определённого такта цикла процессора, который является завершающим в текущей команде, сигнал cycle_reset устанавливается в 1. Это провоцирует синхронный сброс счётчика цикла.

Цикл процессора является основным блоком управления обработкой команд процессора.

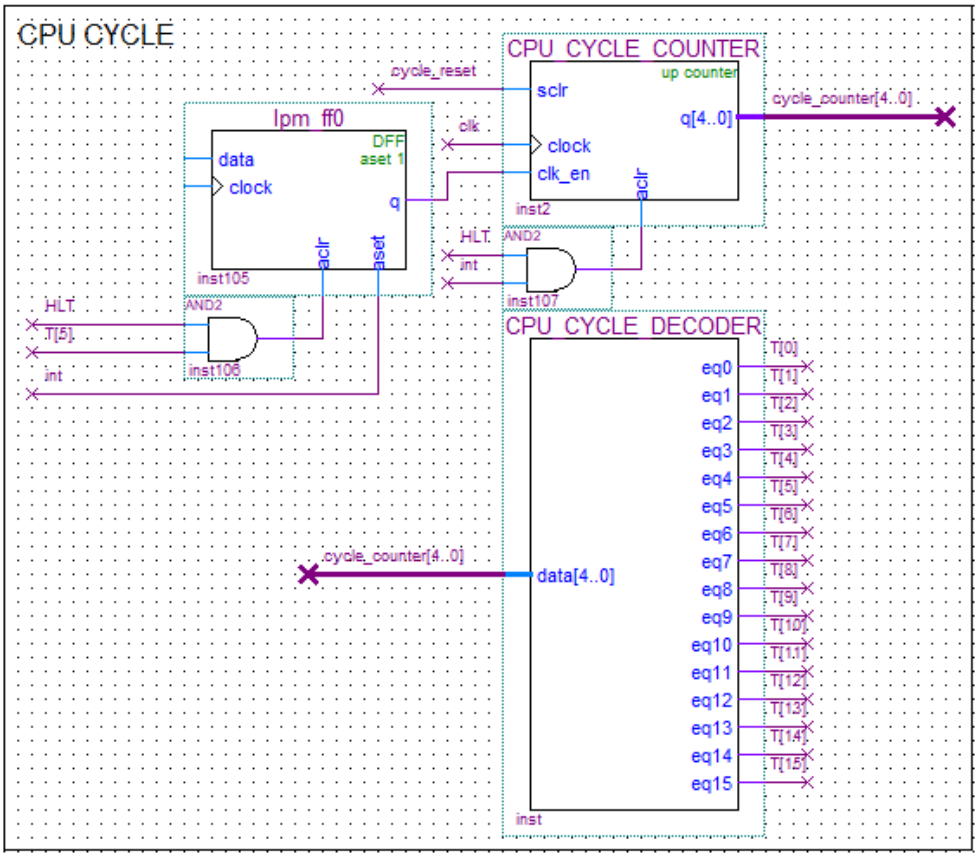

Рисунок 2.8 — Счётчик и декодер цикла процессора

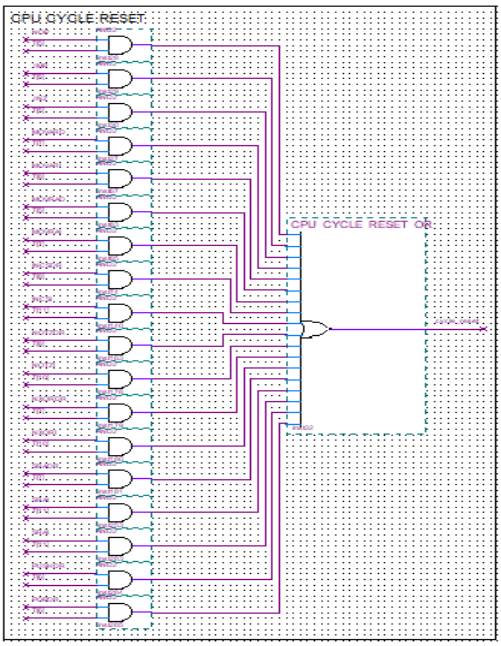

Рисунок 2.9 — Реализация сброса цикла процессора

2.2.1.3 Декодирование команды

После считывания команды на выходе дешифратора, представленного на рисунке 2.10, формируется результат идентификации команды. Данная стадия представляет собой комбинационную схему, и в функциональном моделировании результат доступен уже на T[5] такт цикла процессора. Далее результат декодирования используется для исполнения конкретных команд.

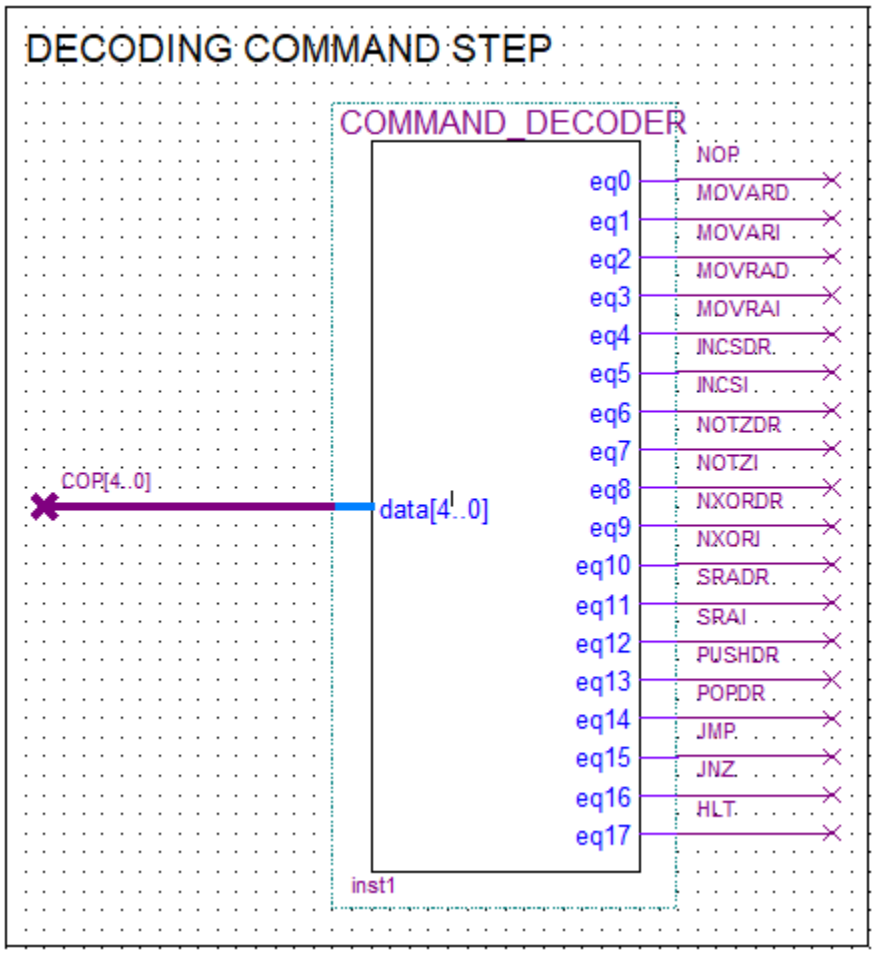

Рисунок 2.10 — Дешифратор команд

2.2.1.4 Разрешение адресов операндов

Согласно варианту, требуется реализовать большую часть команд в двух адресациях. В рамках данной реализации варианта курсового проекта, реализованы следующие команды:

- MOV addr, reg — прямая и косвенная адресация;
- MOV reg, addr — прямая и косвенная адресация;
- INCS reg; INCS addr — прямая регистровая и косвенная адресация;
- NOTZ reg; NOTZ addr — прямая регистровая и косвенная адресация;
- NXOR reg1, reg2; NXOR reg, addr — прямая регистровая и косвенная адресация;

– SRA reg, const; SRA reg, addr — прямая регистровая и косвенная адресация;
– PUSH reg — прямая регистровая адресация;
– POP reg — прямая регистровая адресация;
– JMP addr — непосредственная адресация;
– JNZ addr — непосредственная адресация.

При прямой адресации разрешения адресов не требуется. Адрес операндов подаётся сразу из поля FADDR[15..0] регистра команды. Показано это на рисунке 2.11.

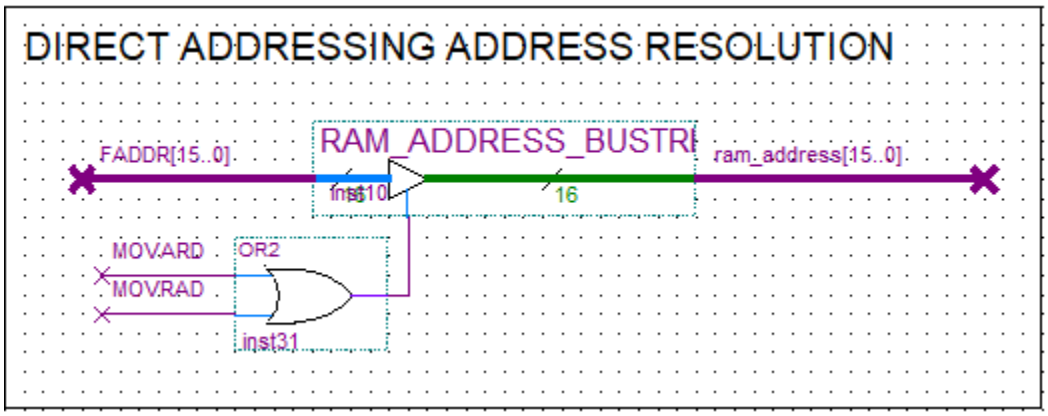

Рисунок 2.11 — Прямая адресация

Всё немного сложнее обстоит с косвенной адресацией. В поле адреса регистра команд лежит не прямой адрес операнда, а адрес места в памяти, где лежит прямой адрес. Данный тип косвенной адресации имеет «глубину» равную 1. Он же и реализован в рамках варианта данного курсового проектирования. Блок, отвечающий за разрешение косвенного адреса, находится в УУ и показан на рисунке 2.12. Решение косвенного адреса занимает порядка 3 такта. После этого на шине адреса данных установлен прямой адрес операнда.

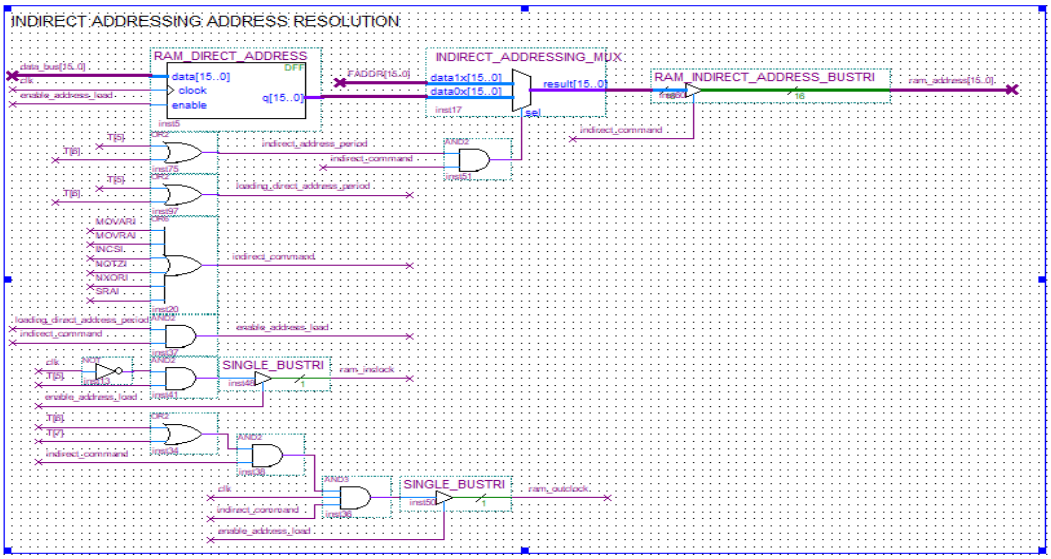

Рисунок 2.12 — Решение косвенного адреса

2.2.1.5 Исполнение команды

Стадия исполнения специфична для каждой из команд. Команды условного и безусловного переходов, а также команда HLT и NOP исполняются в пределах устройства управления. Все остальные команды управляют внешними по отношению к УУ блоками: Стэк, АЛУ, РОН. На рисунке 2.13 показаны реализации стадии исполнения команды INCS двух типов адресации. На рисунке 2.14 показана реализация стадии исполнения команды NOTZ также двух типов адресации.

Реализации стадий исполнения других команд можно найти в устройстве управления. Они помечены прямоугольником с названием команды в левом верхнем углу.

Что касается команды MOV, то её реализация не полностью соответствует выделенному в УУ блоку, так как основная часть этой команды есть пересылка данных из RAM или в RAM. Об управлении доступом к памяти рассказано ниже.

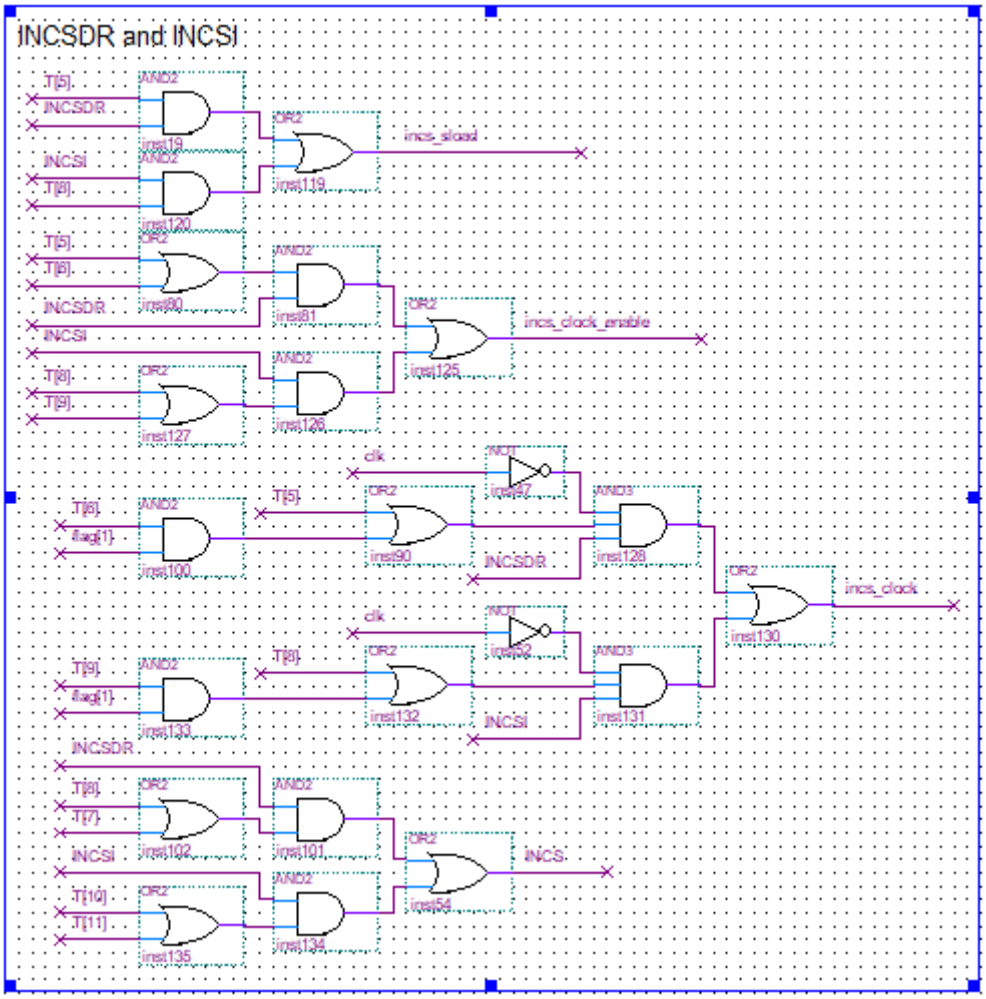

Рисунок 2.13 — Реализация команды INCS

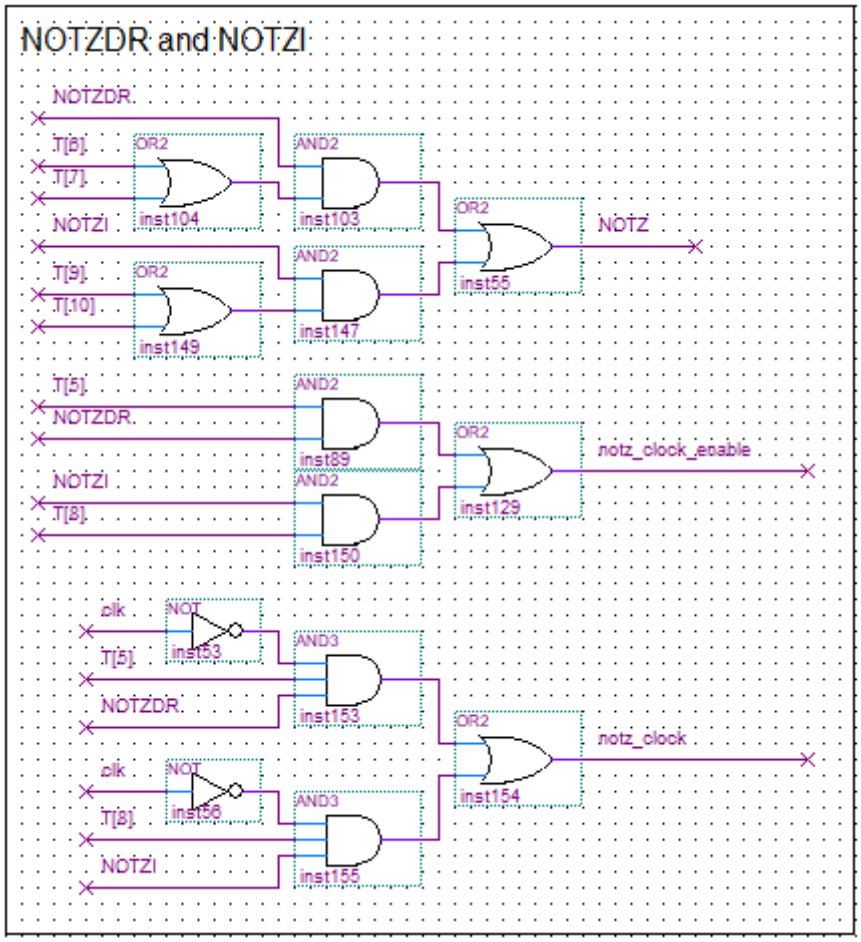

Рисунок 2.14 — Реализация команды NOTZ

2.2.1.6 Доступ к памяти

Устройство управления координирует работу всех действий с памятью. Для этого в нём выделены специальные блоки, показанные на рисунках 2.15—2.17. Эти сигналы управляют чтением/записью из/в RAM, а также управляют доступностью шины данных для считывания и записи данных в RAM. Ввиду того, что многие команды нуждаются в считывании/записи данных из/в RAM, данные блоки представляют собой массивные блоки комбинационной логики. Принятие решения о доступе к памяти осуществляется на основе:

- Типа текущей команды;
- Такта цикла процессора, начиная с которого текущая команда нуждается во взаимодействии с RAM.

Все управляющие блоки доступом к памяти находятся в УУ.

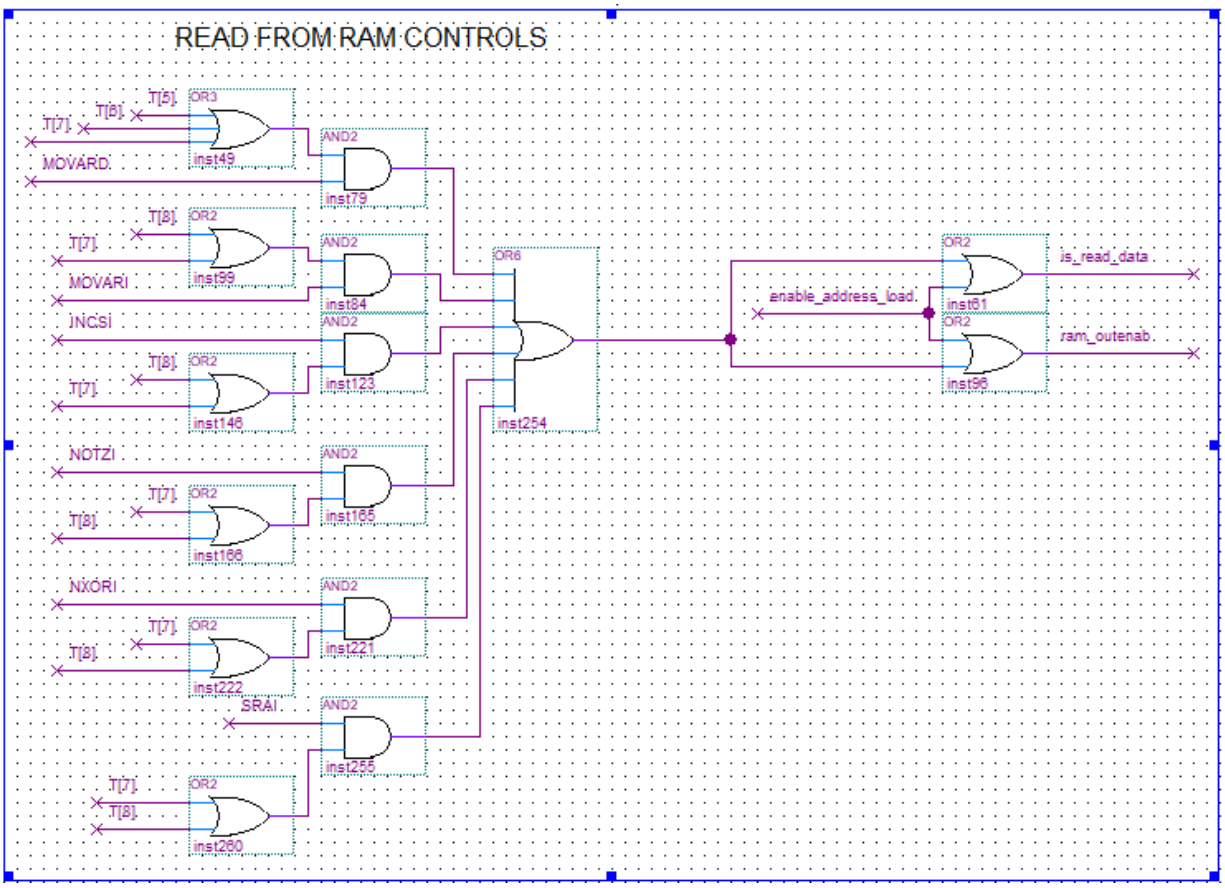

Рисунок 2.15 — Управление доступом считывания из памяти

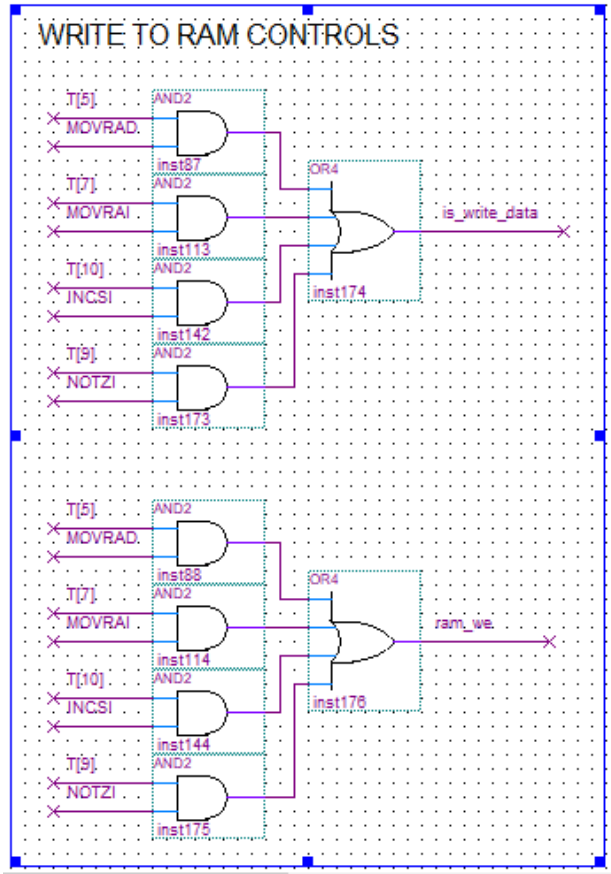

Рисунок 2.16 — Управление записью в память

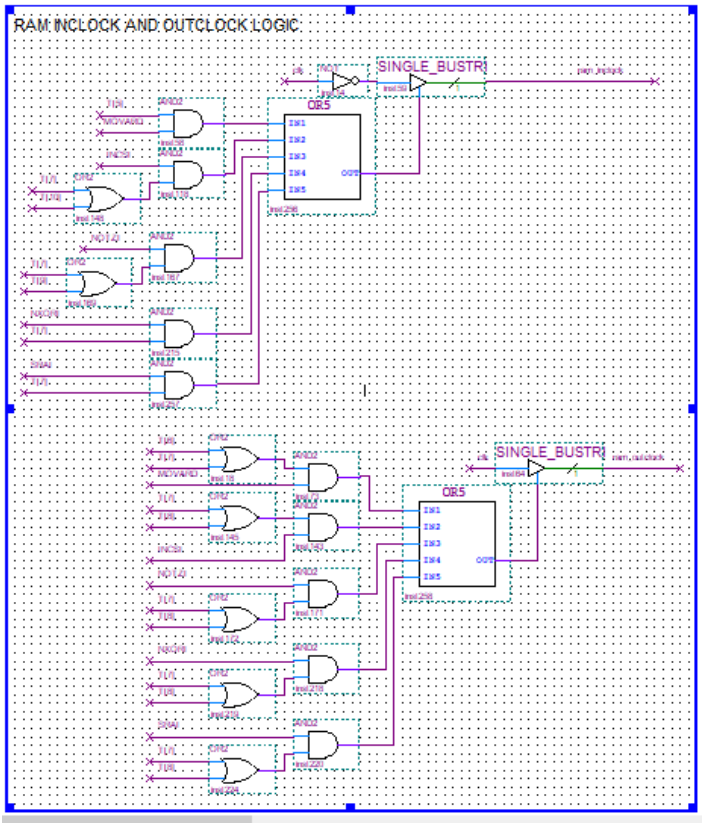

Рисунок 2.17 — сигналы считывания и записи в память

2.2.1.7 Доступ к регистрам

Многие команды работают с РОН. Поэтому было решено обобщить сигналы работы с РОН. На рисунках 2.18—2. Показаны сигналы работы с РОН.

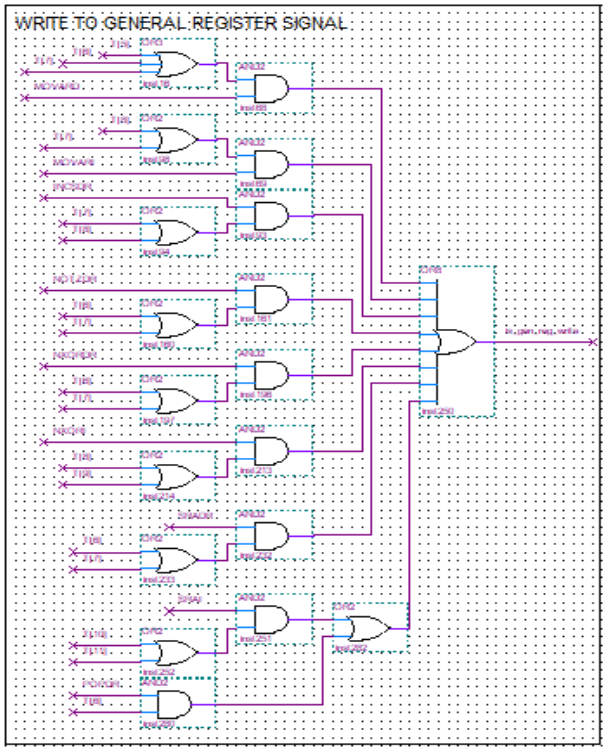

Рисунок 2.18 — Сигнал записи в регистр общего назначения

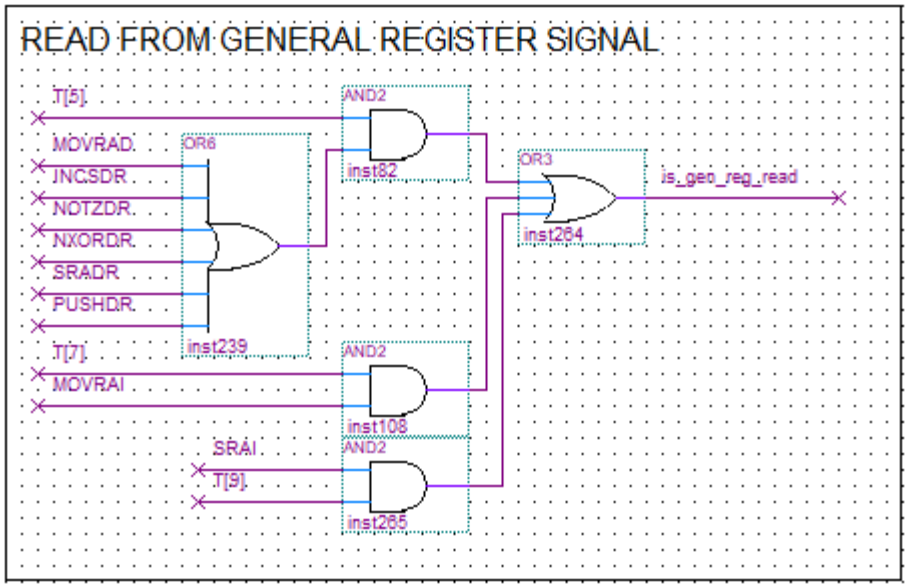

Рисунок 2.19 — Сигнал считывания из регистра общего назначения

2.2.2 Регистры общего назначения

Для реализации основных вычислительных команд требуется продолжительное время доступа к данным. Для обеспечения такого доступа, используются регистры общего назначения. Согласно варианту, требуется реализовать 16 регистров общего назначения. Все регистры названы в стиле архитетуры «x86»: al, ah, bl, bh, … , hl, hh. В приложении Д показана реализация данного блока.

2.2.3 Арифметико-логическое устройство

Для реализации арифметических, логических, а также сдвиговых операций процессор использует АЛУ. Этот блок очень быстр, использует собственные регистры для хранения операндов, а также регистр флагов, на основании которого выполняются все условные команды. Реализация данного блока представлена в приложении Е.

2.2.4 Стек

Аппаратный стек в рамках данной реализации, сделан на основе регистров, счётчика «Stack Pointer» дешифратора текущего положения указателя стека, и выходного мультиплексора. Реализация аппаратного стека показана в приложении Ж.

3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ

В данном разделе представлено функциональное моделирование всех блоков по отдельности и всего устройства в целом.

3.1 Центральный процессор

3.1.1 Стек

На рисунке 3.1 представлена временная диаграмма работы стека.

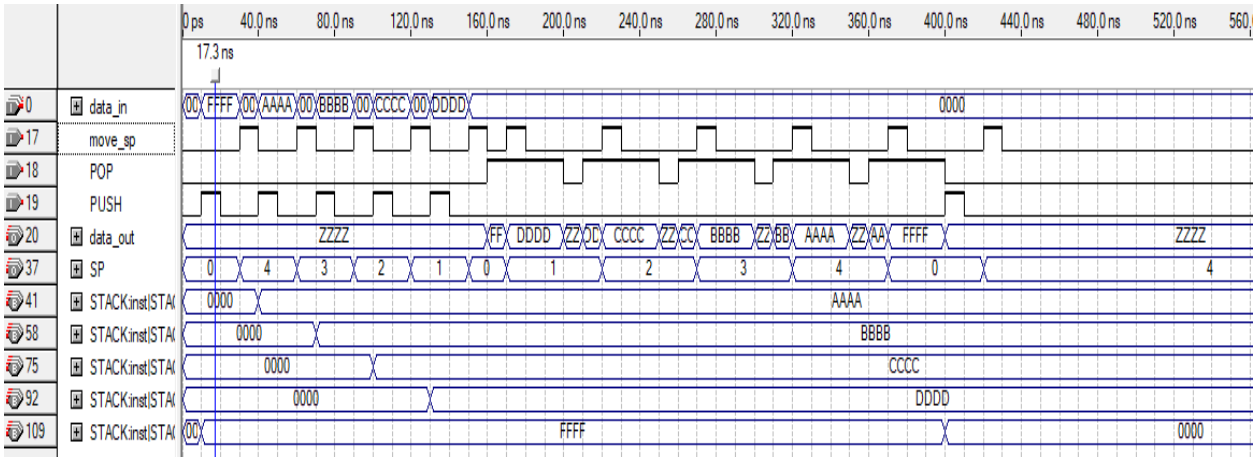

Рисунок 3.1 — Временная диаграмма работы стека

3.1.2 АЛУ

На рисунке 3.2 Представлена временная диаграмма работы АЛУ.

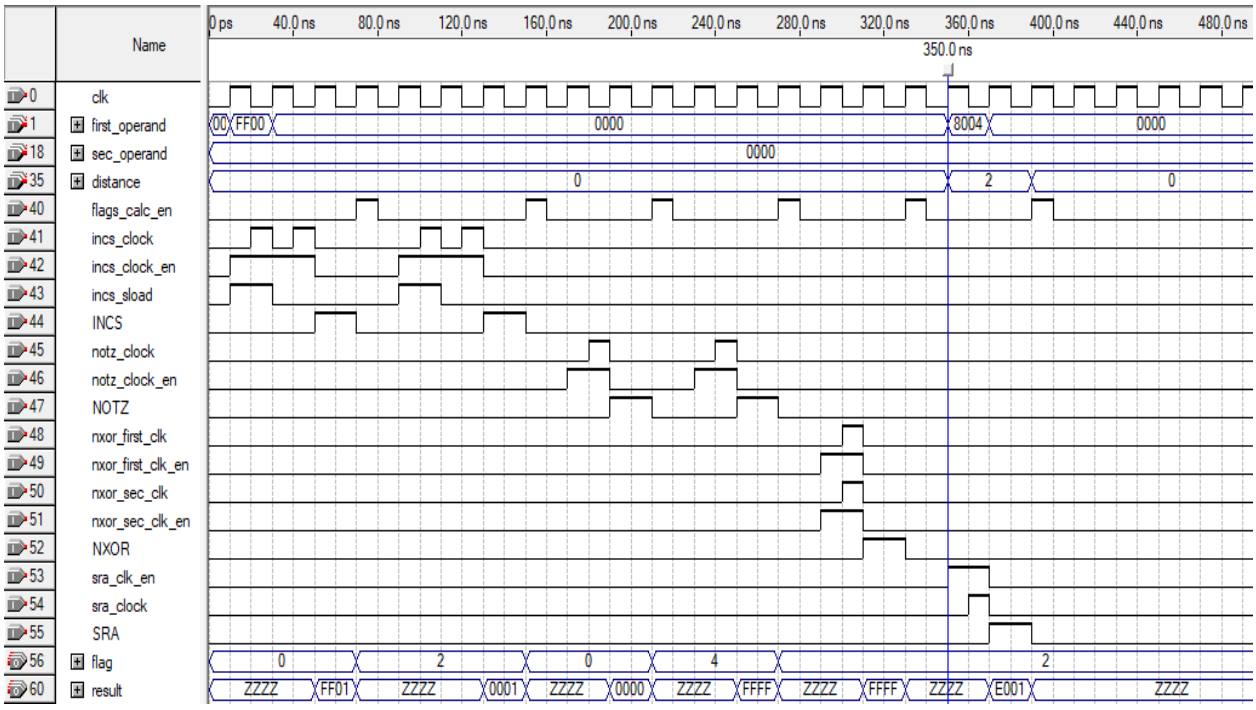

Рисунок 3.2 — Временная диаграмма работы АЛУ

3.1.3 РОН

На рисунке 3.3 представлена временная диаграмма работы РОН.

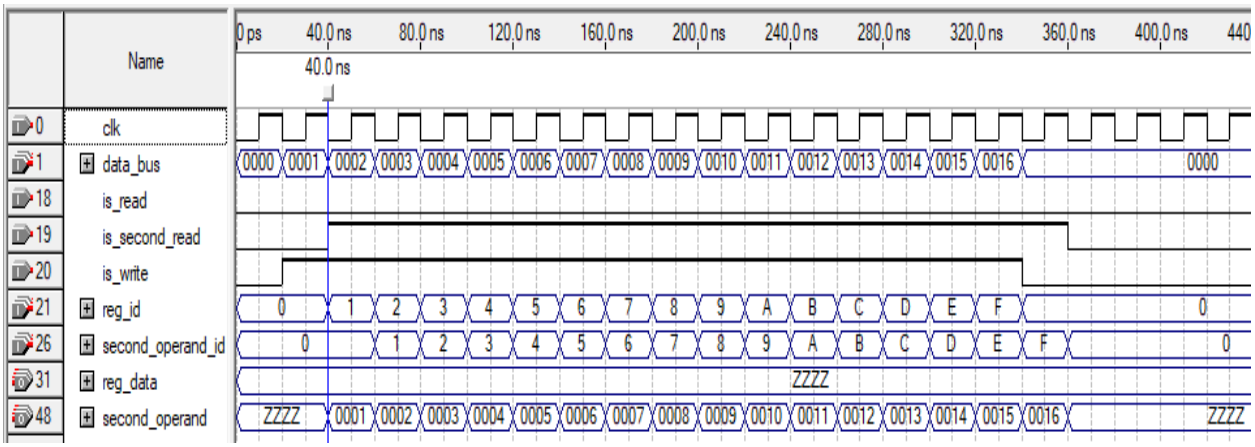

Рисунок 3.3 — Временная диаграмма работы РОН

3.2 Всё устройство

Моделирование работы всего устройства будем проводить при выполнении программы, представленной в таблице 4.1 Номера регистров обозначаются символами al - hh, перед косвенным адресом стоит символ $.

Таблица 3.1 – Исходный код программы

Адрес команды	Символьное кодирование	Двоичный код	Шестнадцатеричный код
0	mov [0], al	0000100000…0	08000…0
3	mov [1], ah	00001000100000000000000010…0	088000800…0
6	mov $2, bl	00010001000000000000000100…0	110001000…0
9	mov $3, bh	00010001100000000000000110…0	118001800…0
12	incs $4	00110000000000000000001000…0	3000020…0
15	incs al	0010100000…0	2800…0
18	mov al, [12]	00011000000000000000011000…0	18000600…0
21	incs $5	00110000000000000000001010…0	30000280…0
24	notz ah	0011100010…0	38800…0
27	incs bl	0010100100…0	29000…0
30	notz $6	01000000000000000000001100…0	400003000…0
33	nxor bh, al	01001001100000…0	49800…0
36	nxor bh, $7	01010001100000000000001110…0	518003800…0
39	Mov [8], ch	00001010100000000000010000…0	0A8004000…0
42	Sra ch, 2	010110101001000…0	5A90…0
45	Sra ch, $9	01100010100000000000010010…0	628004800…0
48	Push al	0110100000…0	68000…0

Продолжение таблицы 3.1

51	Pop cl	0111001000…0	72000…0
54	Mov [10], al	00001000000000000000001010…0	08000500…0
57	Nxor al, ch	01001000001010…0	48280…0
60	jnz 69	10000000000000000010001010…0	800022800…0
63	Hlt	100010…0	8800
66	Jmp 75	01111000000000000010010110…0	78002580
69	Nxor hh, $11	01010111100000000000010110…0	57800580
72	Jnz 63	10000000000000000001111110…0	80001F80

Функциональное моделирование данной программы представлено на рисунке 3.4. На рисунке 3.5 представлен рабочий фрагмент файла инициализации RAM. На рисунке 3.6 представлен фрагмент содержимого памяти RAM после функционального моделирования.

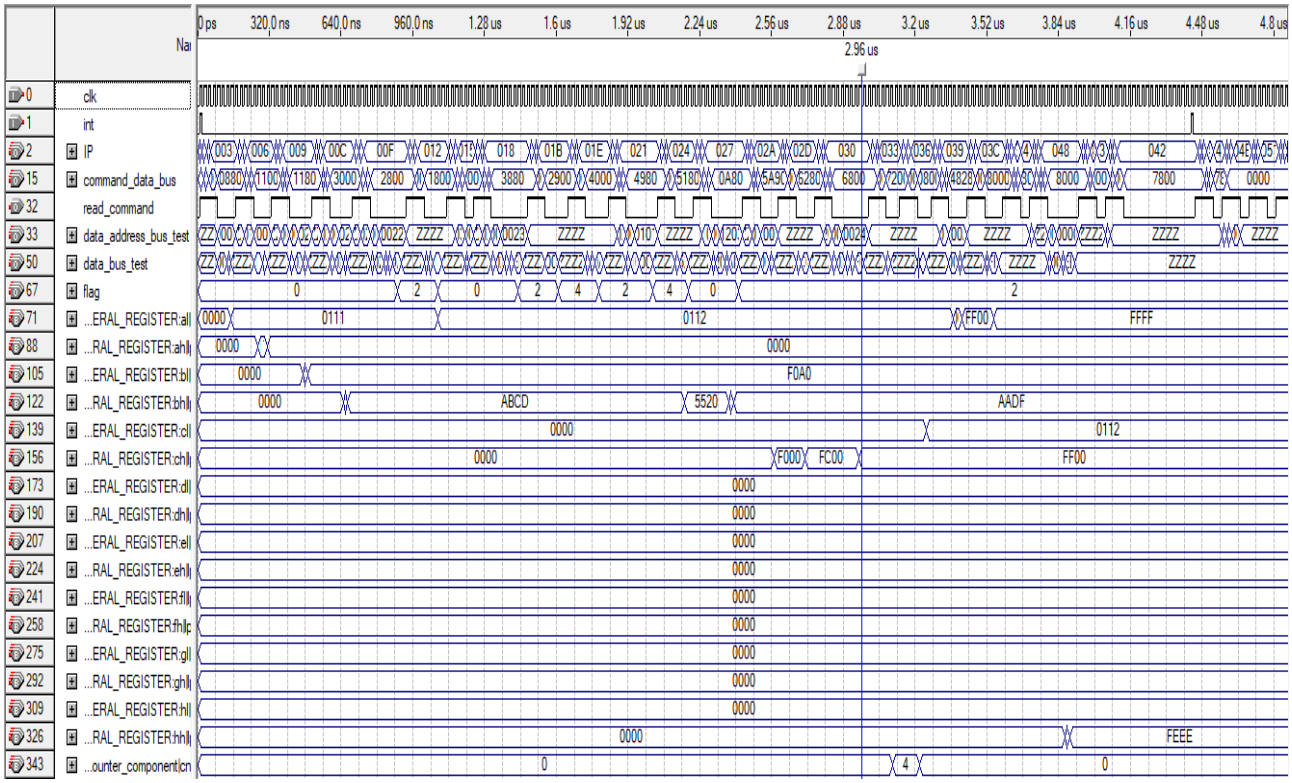

Рисунок 3.4 — Функциональное моделирование программы

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0000	0111	0000	0020	0021	0022	0023	0101	1203
0008	F000	0024	FF00	0000	0000	0000	0000	0000
0010	0000	0000	0000	0000	0000	0000	0000	0000
0018	0000	0000	0000	0000	0000	0000	0000	0000
0020	F0A0	ABCD	CDAB	EEEE	0002	0000	0000	0000
0028	0000	0000	0000	0000	0000	0000	0000	0000
0030	0000	0000	0000	0040	0000	0000	0000	0000
0038	0000	0000	0000	0000	0000	0000	0000	0000
0040	0000	0000	0000	0000	0000	0000	0000	0000
0048	0000	0000	0000	0000	0000	0000	0000	0000
0050	0000	0000	0000	0000	0000	0000	0000	0000
0058	0000	0000	0000	0000	0000	0000	0000	0000
0060	0000	0000	0000	0000	0000	0000	0000	0000
0068	0000	0000	0000	0000	0000	0000	0000	0000
0070	0000	0000	0000	0000	0000	0000	0000	0000

Рисунок 3.5 — Начальное состояние RAM

|CPU_WITH_MEMORY\|lpm_ram_io:inst1\|altram:sram\|altsyncram:ram_bl

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0000	0111	0000	0020	0021	0022	0023	0101	1203
0008	F000	0024	FF00	0000	0112	0000	0000	0000
0010	0000	0000	0000	0000	0000	0000	0000	0000
0018	0000	0000	0000	0000	0000	0000	0000	0000
0020	F0A0	ABCD	CDAB	EEEE	0002	0000	0000	0000
0028	0000	0000	0000	0000	0000	0000	0000	0000
0030	0000	0000	0000	0040	0000	0000	0000	0000
0038	0000	0000	0000	0000	0000	0000	0000	0000
0040	0000	0000	0000	0000	0000	0000	0000	0000
0048	0000	0000	0000	0000	0000	0000	0000	0000
0050	0000	0000	0000	0000	0000	0000	0000	0000
0058	0000	0000	0000	0000	0000	0000	0000	0000
0060	0000	0000	0000	0000	0000	0000	0000	0000
0068	0000	0000	0000	0000	0000	0000	0000	0000

Рисунок 3.6 — Состояние RAM после моделирования

Во время функционального моделирования были обнаружены проблемы интеграции команд. На T[5] такте цикла, буфер команд содержал в себе значения такое, что COP[4..0] = 10001. Эта ситуация обрабатывалась как команда HLT, хотя на исполнении была совершенно другая команда. Проблема была устранена запаздыванием исполнения команды HLT на пол такта. В итоге не было потеряно время лишний такт простоя для этой команды.

Этот случай показывает важность интеграционного тестирования системы.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проектирования была смоделирована работоспособная микро-ЭВМ. Данное устройство в архитектурном и функциональном плане соответствует минимальной поставленной задаче, описанной в листе задания:

- Тип архитектуры — Гарвардская;
- ПЗУ — синхронное;
- ОЗУ — синхронное;
- Типы адресации — прямая, прямая регистровая и косвенная;
- Команды условного перехода — JMP и JNZ;
- Арифметические команды: INCS;
- Логические команды: NOTZ, и NXOR;
- Сдвиговые команды: SRA;
- Стековое ЗУ — объёмом 5 слов с направлением роста вниз;
- Регистры Общего Назначения — 16 штук.

Реализованная микро-ЭВМ имеет базовый рабочий функционал, и может быть модифицирована, с целью повышения производительности. Следующие шаги могут быть предприняты:

- Минимизация количества тактов на каждую стадию цикла процессора;
- Разбиение команд на стадии и модификация исполнения команд до конвейерного типа;
- Добавление предсказателя условных переходов к конвейеру команд, для эффективного исполнения условных переходов без простоя стадий конвейера;
- Введение в систему кэш-памяти команд и данных;
- Замена базовой логики доступа к памяти на полноценный контроллер (КПДП);
- Усовершенствование архитектуры системы команд:
    - Расширение функционала АЛУ.
    - Добавление новых команд.
    - Добавление иных типов адресации.

ЛИТЕРАТУРА

[1]. У. Столлингс, Структурная организация и архитектура компьютерных систем/ У. Столлингс. 5-е изд. – М.: "Вильямс ", 2001. Пер. с англ. – 892 с .

[2]. Э. Таненбаум, Архитектура компьютерных систем / Э. Таненбаум. 4- е изд. – М.: " ПИТЕР ", 2002. Пер. с англ. – 698 с.

[3]. Р. Грушвицкий, Проектирование систем на микросхемах программируемой логики / Р. Грушвицкий. – Спб.: "Питер", 2002. – 608 с .

[4]. Е. Угрюмов, Цифровая схемотехника. - М.: " С – Петербург ", 2001 - 518 стр.

[5]. Documentation: Quartus II Development Software — [Электронный ресурс] — источник данных: http://www.altera.com/literature/lit-qts.jsp