Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: СиФО ЭВМ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

РАЗРАБОТКА МИКРО-ЭВМ НА ПЛИС

БГУИР КП 1-40 02 01 1 ПЗ

Студент: гр. 550504 Нарейко В.И.

Руководитель: Третьяков А.Г.

Минск 2018

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ……………………………………………………………………..3

РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ………………………4

РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ………………..7

ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ…………………………….....19

ЗАКЛЮЧЕНИЕ……………………………………………………………….30

**ВВЕДЕНИЕ**

Целью данного курсового проекта является реализация микро\_ЭВМ на ПЛИС. В качестве инструмента разработки используется программа Quartus II 9.1 Web Edition. В этой программе находятся все необходимые элементы для разработки схем любой сложности. Программа позволяет тестировать разрабатываемую схему при помощи .vpf диаграмм на любом этапе разработки. Также программа позволяет объединять участки схем в блоки, которые в дальнейшем можно использовать в других схемах.

Архитектура разрабатываемой схемы – Пристонская. Это значит, что команды и данные будут находится в одной памяти, в ROM. Также будет использоваться RAM для хранения промежуточных значений. В целом устройство будет выглядеть как RAM, ROM и устройство управления, которое будет включать в себе такие модули, как кэш, арифметически-логическое устройство, стек. Схема будет способна распознавать команды.

1. **РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ**

В данном разделе будут расписаны задание, структурная схема микро-ЭВМ, а также разработка архитектуры команд.

* 1. **Задание**

Вариант: 1.

Тип архитектуры: Пристонская.

Разрядность шин: ША – 8 бит, ШД – 8 бит.

Тип памяти: ПЗУ – синхр., ОЗУ – асинхр.

Тип адресации: Непосредственная.

Команда условного перехода: JS.

Количество РОН: 4.

Команды АЛУ: INCS, NOR, NAND, SLA.

Стек: 11 элементов, рост вверх.

КЭШ: k = 2, алгоритм замещения строк – по наиболее давнему хранения, синхронизация с памятью – сквозная без отображения.

Арбитраж шин: децентрализованный параллельный.

Автомат предсказания: тип – А3, 3 бит, тип шаблона – PC + GHR.

* 1. **Структурная схема микро-ЭВМ**

Структурная схема будет состоять из устройства управления CU и памяти MEMORY. Устройство управления CU будет содержать в себе блок РОН, АЛУ и стек. А память будет оперативная и постоянная. Структурная схема представлена на рисунке 1.2.1.



Рисунок 1.2.1

Работа устройства будет начинаться при подаче на вход управляющего устройства CU тактового сигнала CLK, каждый такт устройство управления будет обрабатывать и управлять внутренними компонентами (АЛУ, стек) и работой с памятью. В управляющем устройстве существует цикл обработки команд. В его начале устройство управления обращается к ПЗУ для чтения команды и операндов, затем расшифровывая команду начинается её выполнение. При выполнении команды устройство управления будет обращаться к стеку, АЛУ, регистрам. Затем начинается выполнение новой команды. Сами команды хранятся в ПЗУ в .hex-файле ROM.hex.

* 1. **Разработка архитектуры команд**

Команды, которые требуется реализовать:

MOV reg, $mem

MOV $mem, reg

PUSH reg

POP reg

JMP $mem

JS $mem

HLT

INCS reg, reg

NOR reg, reg

NAND reg, reg

SLA reg, reg

INCS reg, const

NOR reg, const

NAND reg, const

SLA reg, const

MOV reg, const

MOV $mem, const

Все команды будут состоять из трех слов. Первое слово -идентификатор команды, второе – первый операнд, третье - второй операнд см. таблицу 1.3.1:

Таблица 1.3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Команда | Идентификатор | | Первый операнд | Второй операнд |
| DEC | HEX |
| MOV reg, $mem | 01 | 01 | reg | $mem |
| MOV $mem, reg | 02 | 02 | $mem | reg |
| PUSH reg | 03 | 03 | reg | - |
| POP reg | 04 | 04 | reg | - |
| JMP $mem | 05 | 05 | $mem | - |
| JS $mem | 06 | 06 | $mem | - |
| HLT | 07 | 07 | - | - |
| INCS reg, reg | 08 | 08 | reg | reg |
| NOR reg, reg | 09 | 09 | reg | reg |
| NAND reg, reg | 10 | 0A | reg | reg |
| SLA reg, reg | 11 | 0B | reg | reg |
| INCS reg, const | 12 | 0C | reg | const |
| NOR reg, const | 12 | 0D | reg | const |
| NAND reg, const | 14 | 0E | reg | const |
| SLA reg, const | 15 | 0F | reg | const |
| MOV reg, const | 16 | 10 | reg | const |
| MOV $mem, const | 17 | 11 | $mem | const |

Так как в разрабатываемой схеме используются 8 разрядные шины адреса и данных, то любое слово команды будет занимать ровно 8 бит.

1. **РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ**

В данном разделе будут описаны основные устройства, которые

были разработаны для реализации проекта.

* 1. **Память**

Исходя из того, что была выбрана пристонская архитектура, где

память команд и данных объединена, было принято следующее решение: все так же использовать асинхронный rom для хранения команд, а синхронный ram для хранения данных, но объединить их в одно адресное пространство следующим образом: для адресации в rom использовались адреса с 0-63, а для адресации в ram – 63-127 (диапазоны были выбраны исходя из разрядности ША). Организация блока памяти представлена на рисунке 2.1.1.

Так как была выбрана Пристонская архитектура в курсовом проекте используется одна память для хранения команд и данных. В нашем случае это ROM. Также в разрабатываемом устройстве необходима память для хранения промежуточных результатов – это RAM. Объем ROM и RAM выбран максимально адресуемый это 256 слов по 8 бит. Память в общей схеме представлена на рисунке 2.1.1.

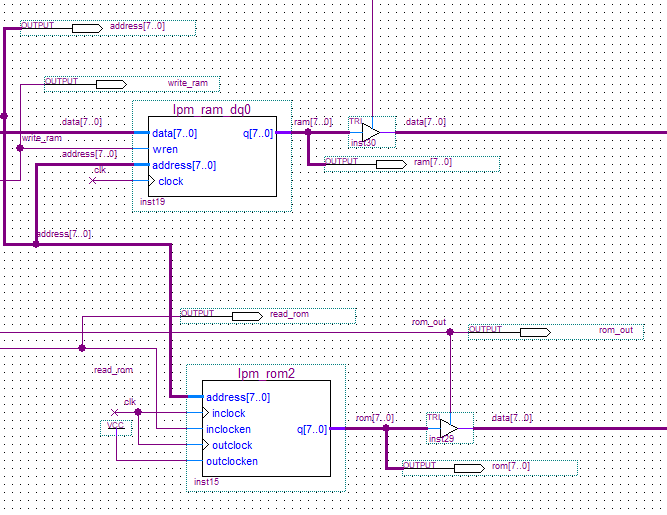


Рисунок 2.1.1

На рисунке мы видим два блока для ПЗУ и ОЗУ. К ним подключены управляющие сигналы от устройства управления, рисунки 2.1.2 и 2.1.3. Также отметим у ПЗУ наличие тактовых входов для синхронизации записи и чтения.

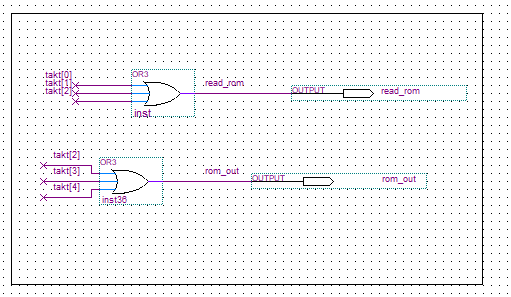


Рисунок 2.1.2

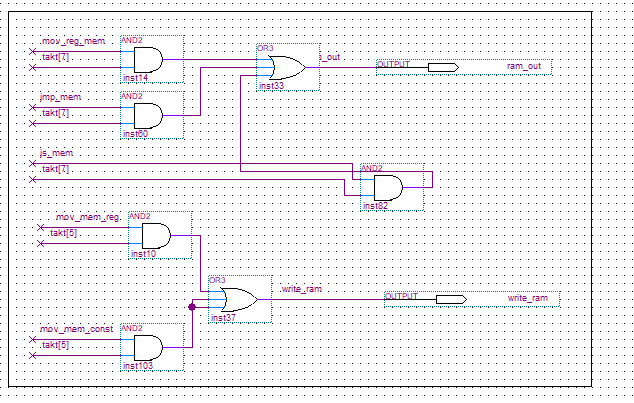


Рисунок 2.1.3

* 1. **АЛУ**

АЛУ будет выполнять 4 различных команд: инкремент, побитовый ИЛИ с инверсией, побитовый И с инверсией и арифметический сдвиг влево.

Инкремент реализован при помощи сумматора с статической единицей в качестве второго операнда, рисунок 2.2.1.

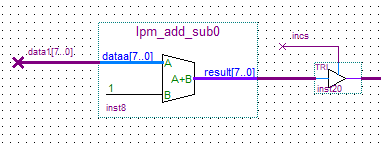


Рисунок 2.2.1

Побитовый ИЛИ с инверсией представлен на рисунке 2.2.2.

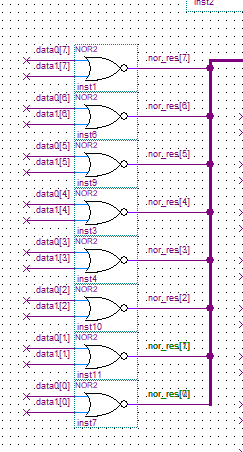


Рисунок 2.2.2

Побитовый И с инверсией представлен на рисунке 2.2.3.

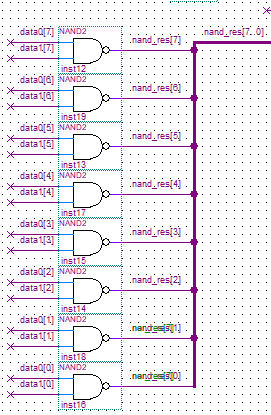


Рисунок 2.2.3

Арифметический сдвиг реализован при помощи прибавления операнда к самому себе же, рисунок 2.2.4

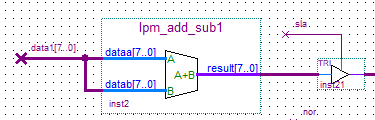


Рисунок 2.2.4

Флаг Sign Flag всегда равен старшему биту результата, см. рисунок 2.2.5

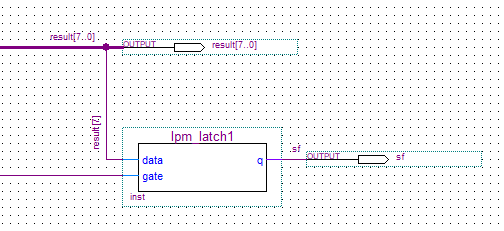


Рисунок 2.2.5

* 1. **Стек**

Стек состоит из 12 элементов. Выбор элемента происходит при помощи указателя ST, рисунок 2.3.1

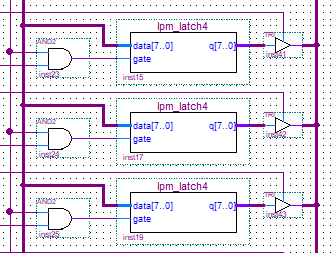


Рисунок 2.3.1

Так как стек по варианту растет вверх, требуется реализовать

правильный инкремент SP. Если на вход блока стека пришла команда push, требуется проверить, а не заполнен ли стек полностью. Если заполнен, то инкрементацию выполнять не надо. Также требуется выполнить проверку при подаче команды pop. Реализация Stack Pointer рисунок 2.3.2

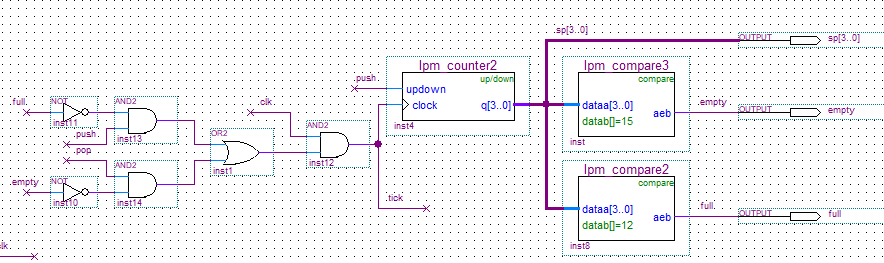


Рис. 2.3.2

* 1. **РОН**

Блок РОН состоит из 4 регистров и управляющих флагов. Выбор регистра происходит при помощи декодирования 2 битного сигнала num[1..0], см. рисунок 2.4.1

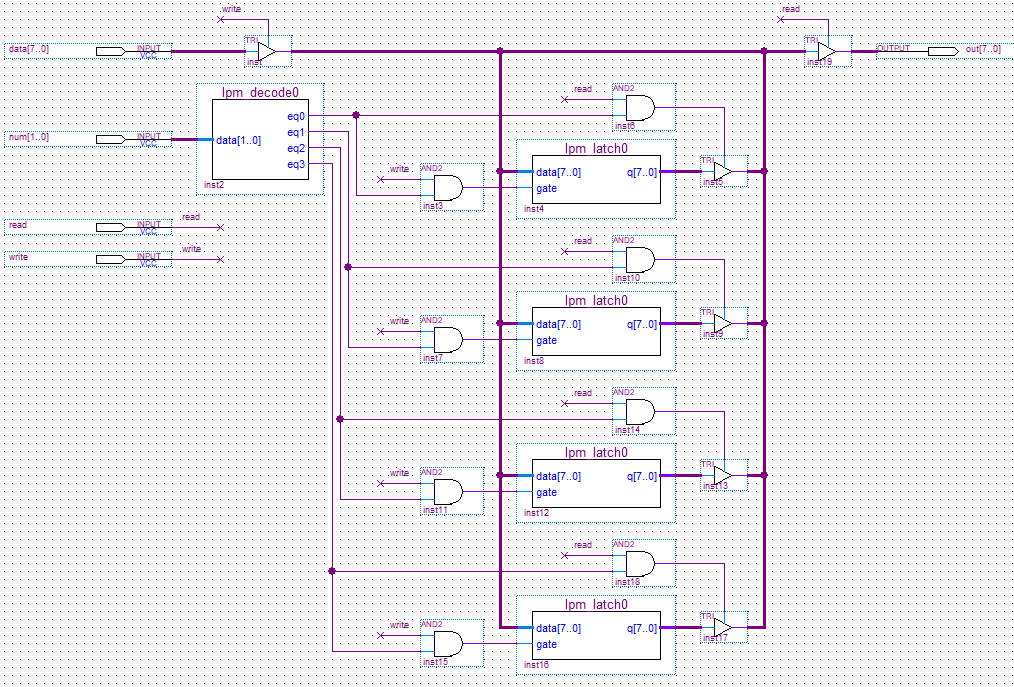


Рисунок 2.4.1

* 1. **CU**

Управляющее устройство начинает свою работу получая такты от входа clk. Сигнал clk увеличивает счетчик, который формирует бесконечный цикл из тактов. Каждый пробег по этому циклу это выполнение одной команды. Реализация тактирующего блока представлена на рисунке 2.5.1

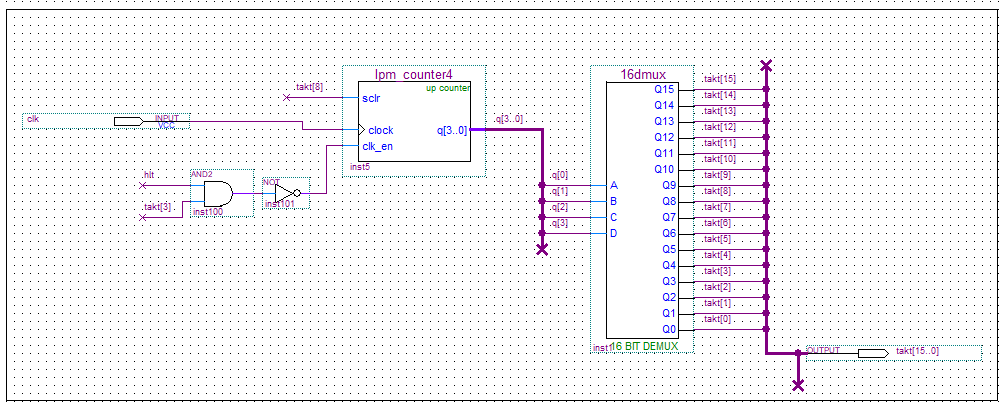


Рисунок 2.5.1

Для того чтобы устройству знать какую команду запустить следующей реализован указатель команды IP, рисунок 2.5.2

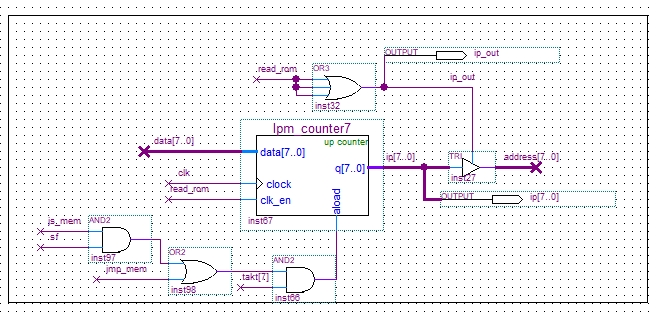


Рисунок 2.5.2

В начале цикла команда и операнды считываются из ПЗУ и записываются в регистры, чтобы в дальнейшем использовать их. Считанный идентификатор команды дешифруется чтобы установить флаг нужно команды, рисунок 2.5.3. Реализована система сигналов для управления операндами, рисунок 2.5.4

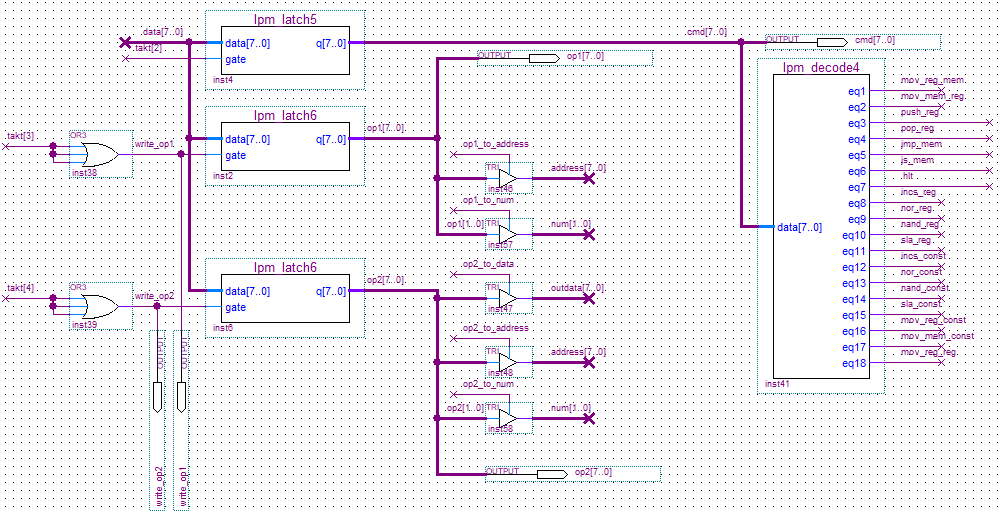


Рисунок 2.5.3

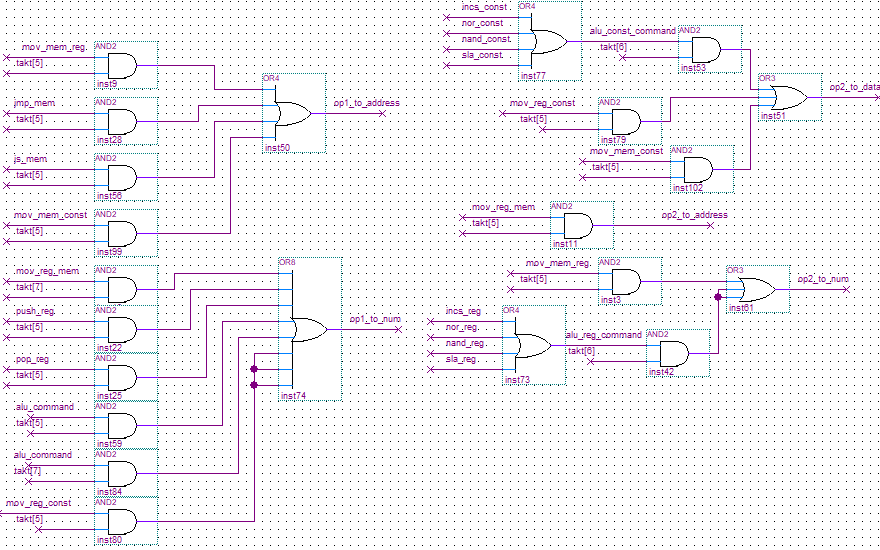


Рисунок 2.5.4

Для управления внутренними блоками АЛУ, стек, РОН реализованы системы управляющих сигналов рисунок 2.5.5, 2.5.6, 2.5.7 соответственно.

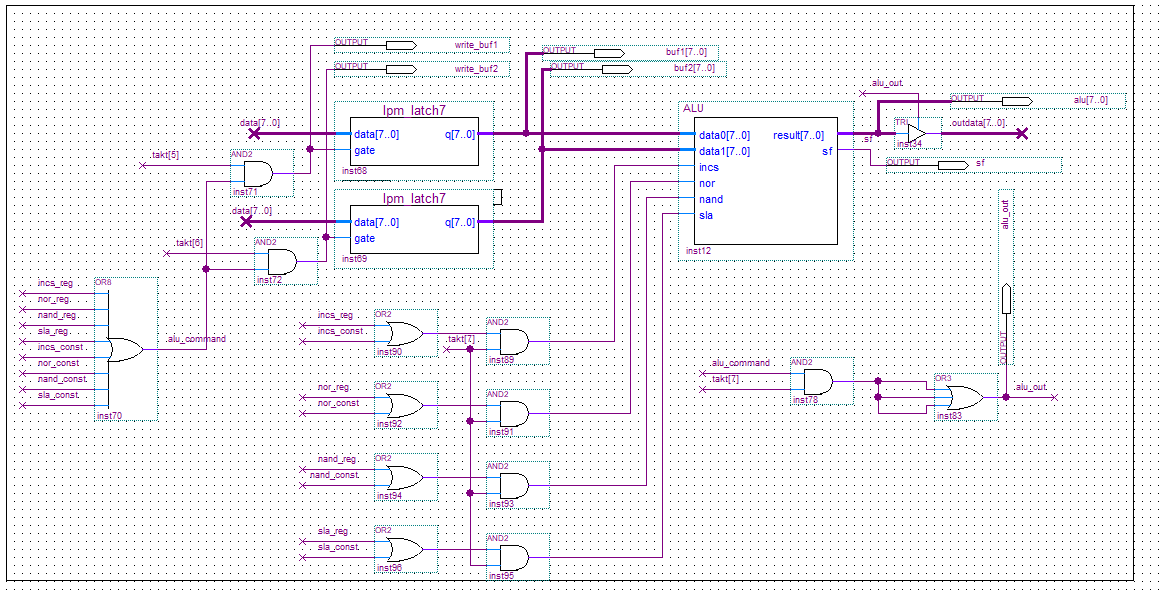


Рисунок 2.5.5

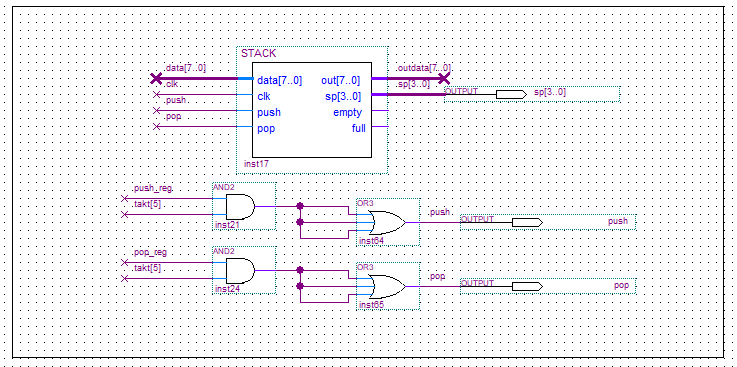


Рисунок 2.5.6

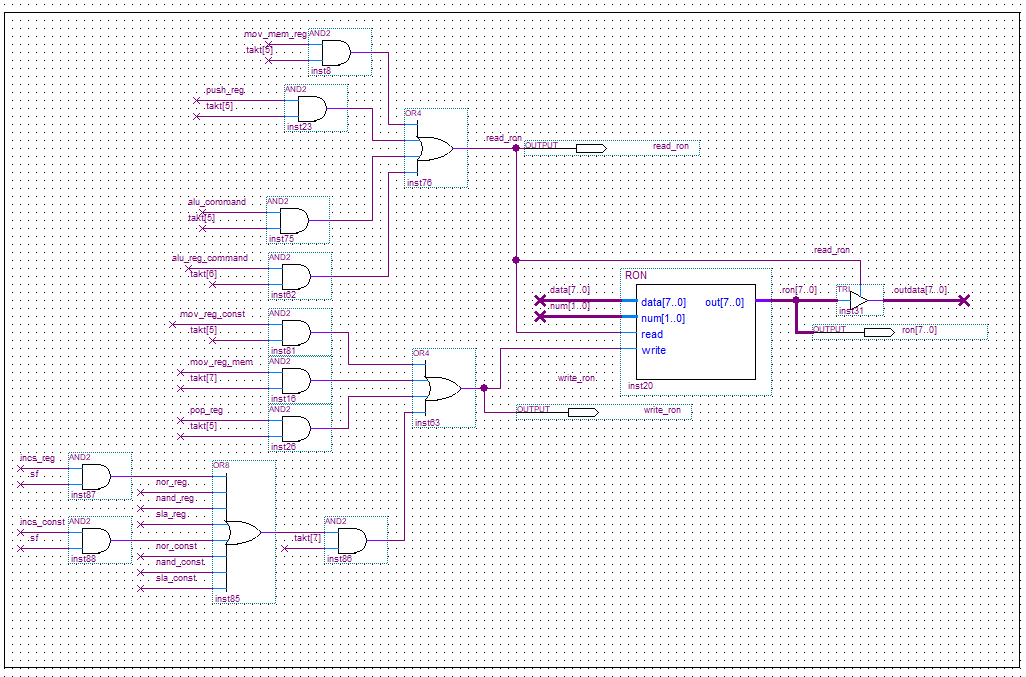


Рисунок 2.5.7

Для обращения к внешней памяти также предусмотрена система сигналов, рисунок 2.5.8.

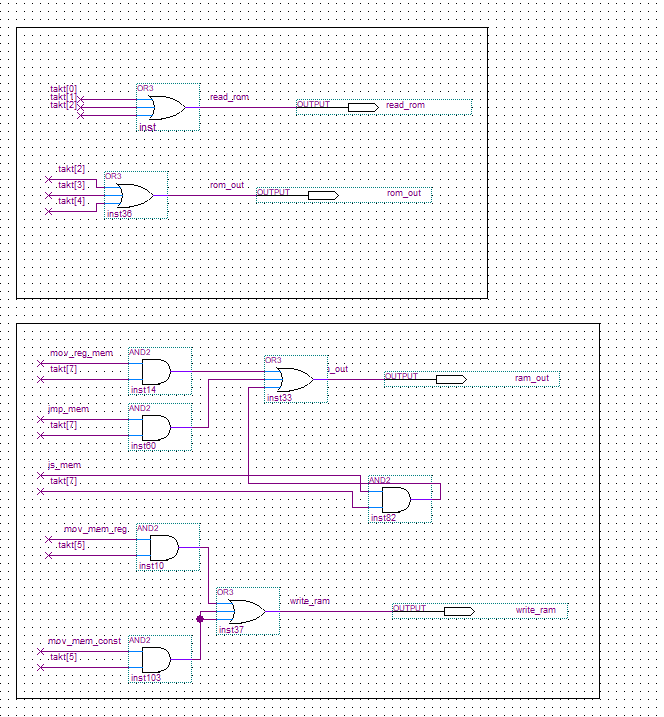


Рисунок 2.5.8

1. **ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ**

Результат функционального моделирования блока АЛУ:

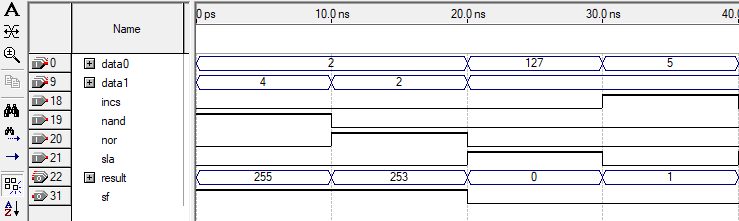


Рисунок 3.1

Результат функционального моделирования блока стек:

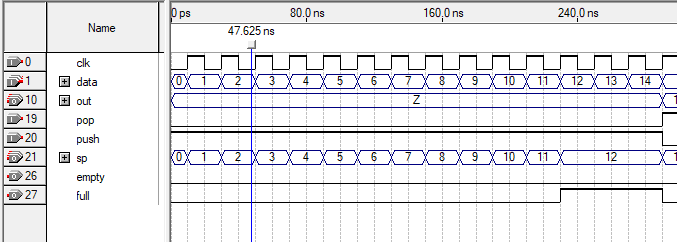


Рисунок 3.2

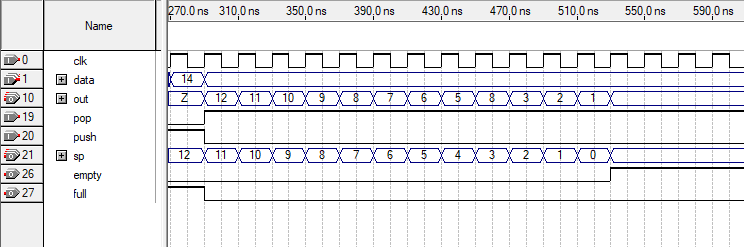


Рисунок 3.3

Результат функционального моделирования блока РОН:

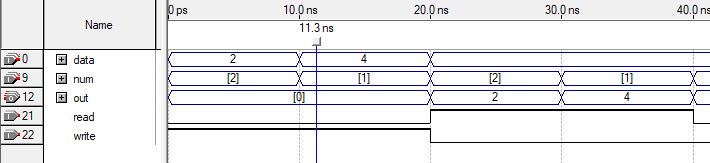


Рисунок 3.4

Код реализуемой программы:

Таблица 3.1 – программа микро-ЭВМ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | HEX | | | BIN |
| Mov r0, $3 | 01 | 00 | 03 | 00000001 00000000 00000011 |
| Mov $5, r0 | 02 | 05 | 00 | 00000010 00000101 00000000 |
| Push r0 | 03 | 00 | 00 | 00000011 00000000 00000000 |
| Pop r1 | 04 | 01 | 00 | 00000100 00000001 00000000 |
| Mov $6, r1 | 02 | 06 | 01 | 00000010 00000110 00000001 |
| Jmp $4 | 05 | 04 | 00 | 00000101 00000100 00000000 |
| Js $0 | 06 | 00 | 00 | 00000110 00000000 00000000 |
| Mov r0, $0 | 01 | 00 | 00 | 00000001 00000000 00000000 |
| Sla r1, r0 | 0B | 01 | 00 | 00001011 00000001 00000000 |
| Mov $7, r1 | 02 | 07 | 01 | 00000010 00000111 00000001 |
| Js $1 | 06 | 01 | 00 | 00000110 00000001 00000000 |
| Incs r0, r1 | 08 | 00 | 01 | 00001000 00000000 00000001 |
| Mov $8, r0 | 02 | 08 | 00 | 00000010 00001000 00000000 |
| Nand r0, r1 | 0A | 00 | 01 | 00001010 00000000 00000001 |
| Nand r0, 5 | 0E | 00 | 05 | 00001110 00000000 00001001 |
| Mov $9, r0 | 02 | 09 | 00 | 00000010 00001001 00000000 |
| Sla r0, F3 | 0F | 00 | F3 | 00001111 00000000 11110011 |
| Incs r1, r0 | 08 | 01 | 00 | 00001000 00000001 00000000 |
| Nor r0, r1 | 09 | 00 | 01 | 00001001 00000000 00000001 |
| Nor r0, 4 | 0D | 00 | 04 | 00001101 00000000 00000100 |
| Mov $10, r0 | 02 | 0A | 00 | 00000010 00001010 00000000 |
| Mov $11, 50 | 11 | 0B | 50 | 00010001 00001011 01010000 |
| Mov r0, FF | 10 | 00 | FF | 00010000 00000000 11111111 |
| Nor r0, r0 | 09 | 00 | 00 | 00001001 00000000 00000000 |
| Mov 12, r0 | 02 | 0C | 00 | 00000010 00001100 00000000 |
| Incs r0, 0 | 0C | 00 | 00 | 00001100 00000000 00000000 |
| Mov 12, r0 | 02 | 0C | 00 | 00000010 00001100 00000000 |
| Hlt | 07 | 00 | 00 | 00000111 00000000 00000000 |

Общее моделирование:

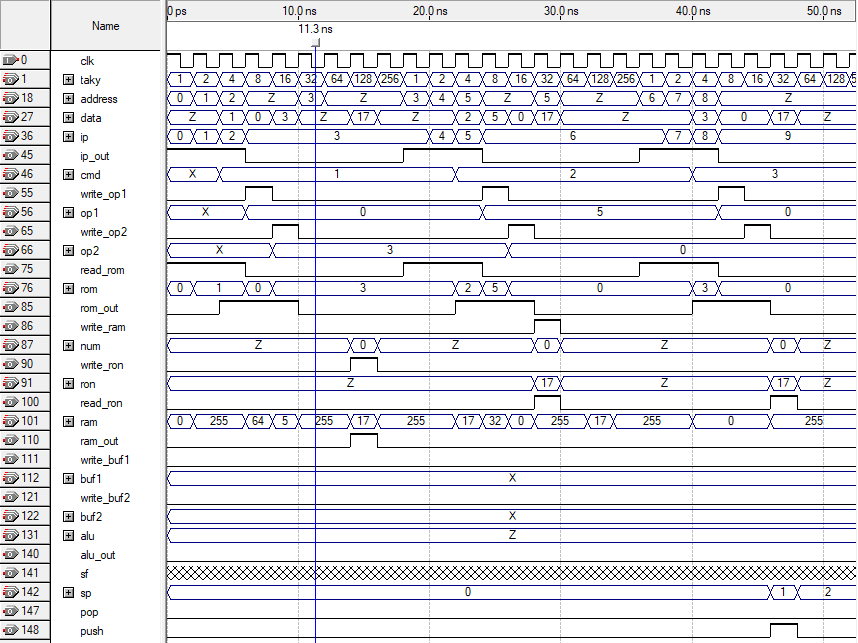


Рисунок 3.5

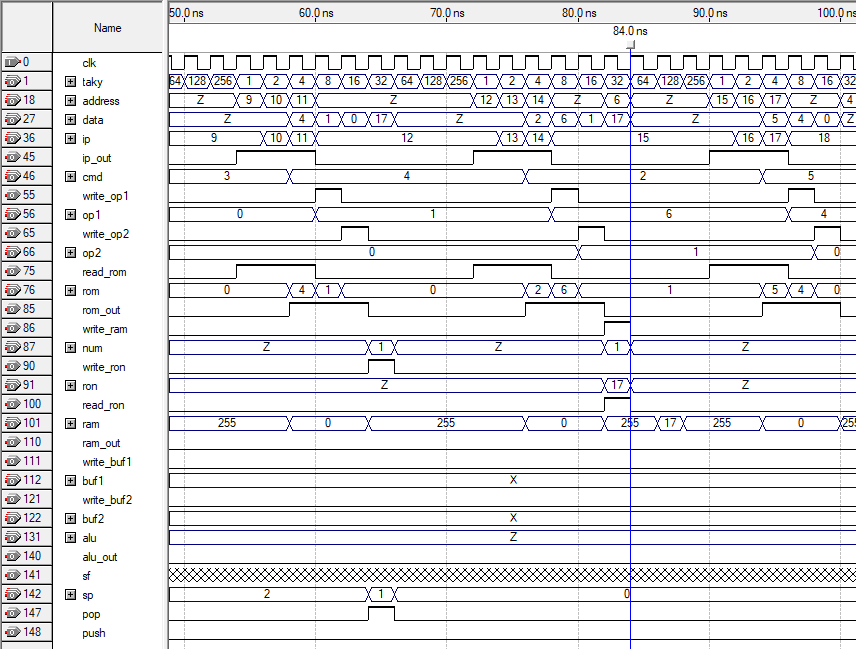


Рисунок 3.6

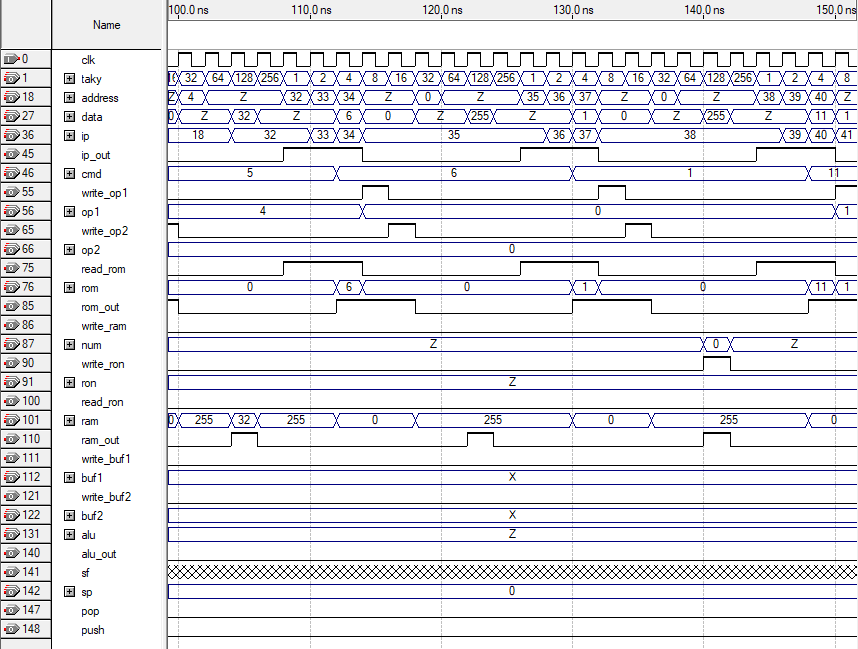
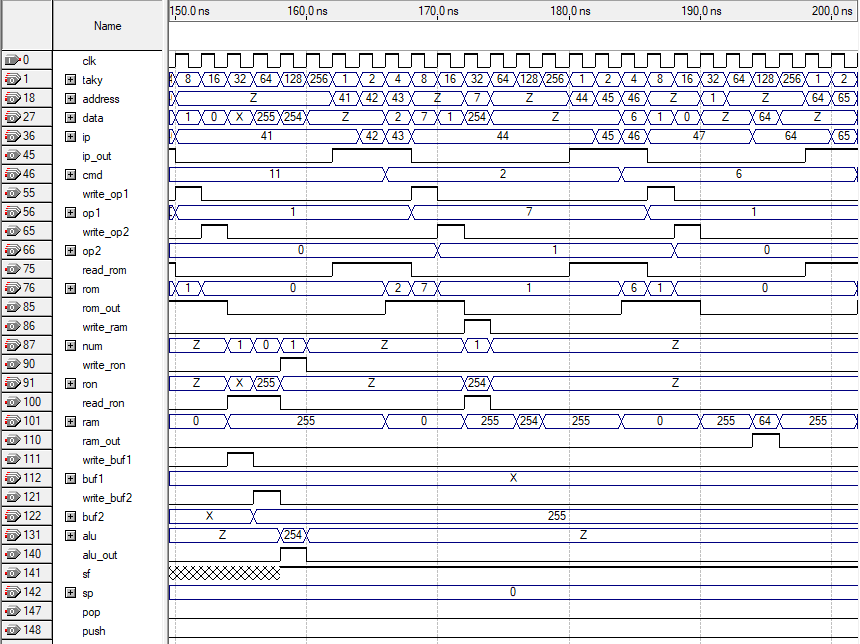


Рисунок 3.7

 Рисунок 3.8

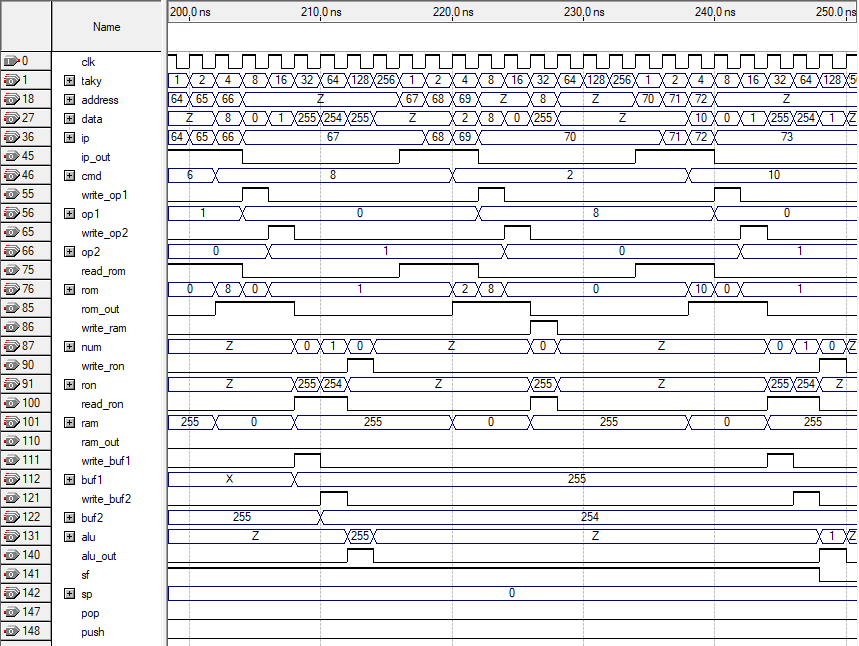


Рисунок 3.9

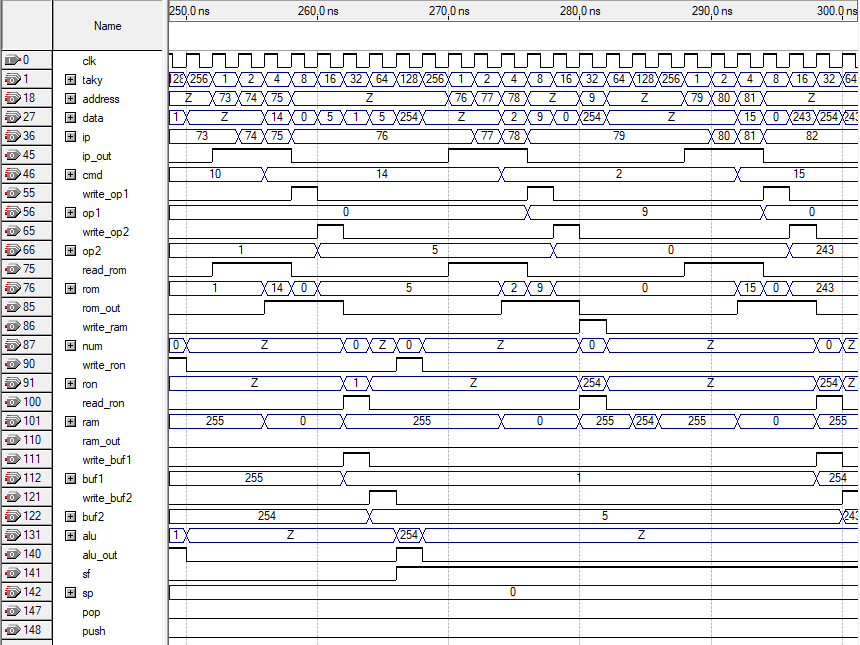


Рисунок 3.10

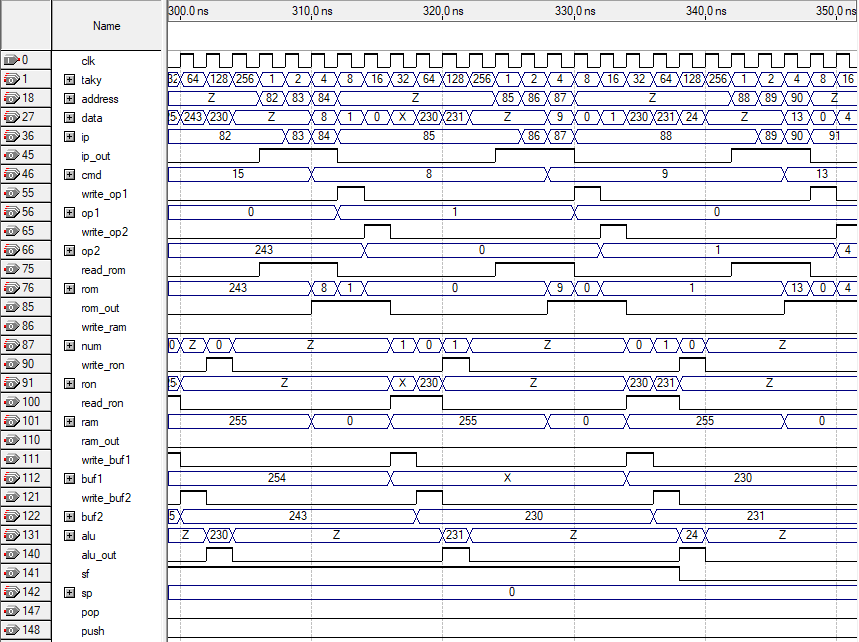


Рисунок 3.11

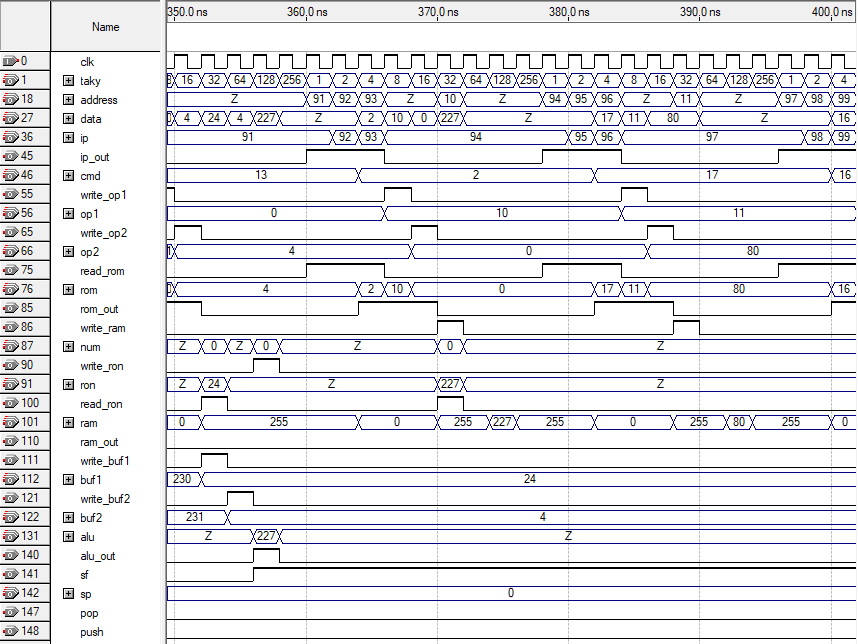


Рисунок 3.12

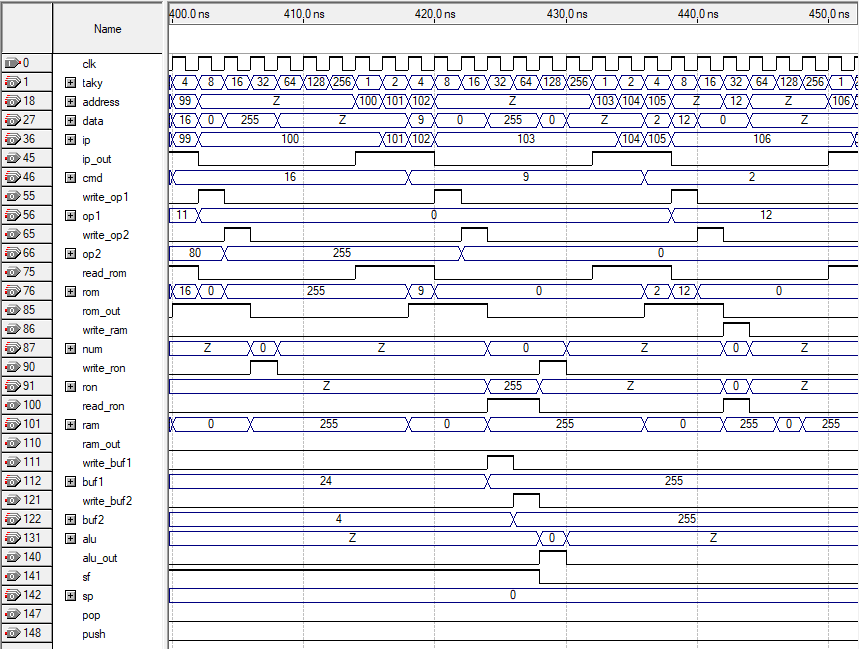


Рисунок 3.13

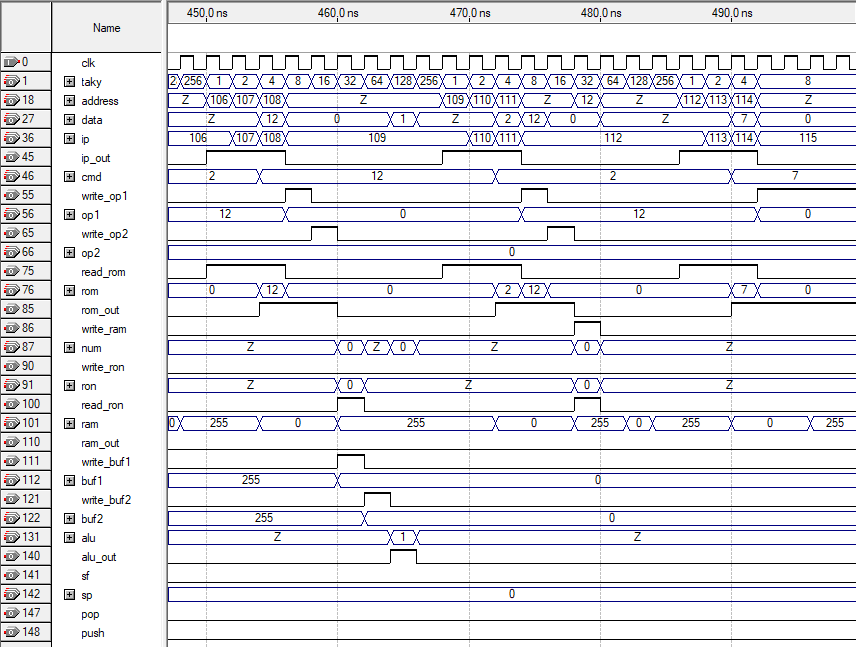


Рисунок 3.14

Дамп памяти ПЗУ:

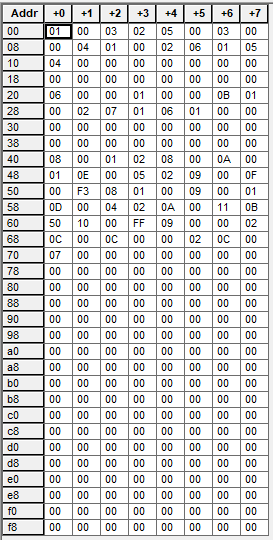


Рисунок 3.15

Дамп памяти ОЗУ до моделирования:

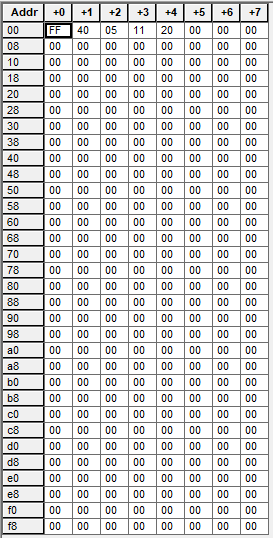


Рисунок 3.16

Дамп памяти ОЗУ после моделирования:

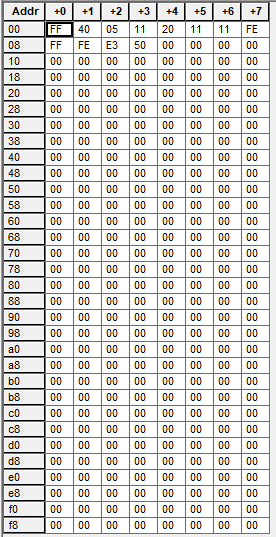


Рисунок 3.17

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения задания была спроектирована микро-ЭВМ на

ПЛИС, которая выполняет все заявленные команды. Микро-ЭВМ легко управляется путем добавления кода описанных команд в память ПЗУ. Спроектированная система может быть расширена путем добавления

конвейера и предсказателя переходов, контроллера прямого доступа к памяти (КПДП), КЭШ и арбитража.