Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

РАЗРАБОТКА МИКРО-ЭВМ НА ПЛИС

БГУИР КП 1–40 02 01 110 ПЗ

Студент Р.Д. Горбач

Руководитель А.И. Стракович

МИНСК 2025

Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: КСиС. Кафедра: ЭВМ.

Специальность: 40 02 01 «Вычислительные машины, системы и сети».

Специализация: 400201-01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_\_\_\_\_ Б.В. Никульшин

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

ЗАДАНИЕ

по курсовому проекту студента

Горбача Романа Дмитриевича

**1** Тема проекта: «Проектирование Микро-ЭВМ на ПЛИС – вариант №45»

**2** Срок сдачи студентом законченного проекта: 2 мая 2025 г.

**3** Исходные данные к проекту:

Тип архитектуры: пристонская; Разрядность шин: адреса – 9 бит, данных – 15 бит; Память: ПЗУ – ассинхронная, ОЗУ – синхронная, тип адресации –прямая регистровая; Команда условного перехода: JAZ; РОН: 12; КЭШ: к – 2, алгоритм замещения строк – наиболее давнего хранеия, синхронизация с памятью – сквозная с отображением; Команды АЛУ: SUB, XOR, NAND, ROR; Арбитраж шин:децентрализованный параллельный; Стек: объем – 13, напр. роста – вверх; Схема предск. переходов: тип автомата – А4, тип шаблона – GHR | PC, размер шаблона – 4 бит; КПДП; Конвейер;

**4** Содержание пояснительной записки (перечень подлежащих разработке

вопросов):

Введение 1. Разработка общей структуры микро-ЭВМ. 1.1. Функциональный состав микро-ЭВМ. 1.2. Разработка системы команд. 1.3. Описание взаимодействия всех блоков микро-ЭВМ при выполнении команд программы. 2. Разработка основных устройств микро-ЭВМ. 3. Функциональное моделирование. Заключение. Приложения.

**5** Перечень графического материала (с точным указанием обязательных

чертежей):

**5.1** Проектирование микро-ЭВМ на ПЛИС. Схема структурная.

**5.2** Проектирование микро-ЭВМ на ПЛИС. Схема функциональная.

КАЛЕНДАРНЫЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапов  курсового проекта | Объем  этапа,  % | Срок выполнения этапа | Примечания |
| Разработка общей структуры микро-ЭВМ | 20 | 27.01 – 27.02 |  |
| Разработка основных устройств микро-ЭВМ | 50 | 27.02 – 15.04 |  |
| Функциональное моделирование | 15 | 15.04 – 22.04 |  |
| Оформление пояснительной записки и графического материала | 15 | 24.04 – 01.05 |  |
| Защита курсового проекта |  | 04.05 – 16.05 |  |

Дата выдачи задания: 25.01.2025 г.

Руководитель А.И. Стракович

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ \_\_\_\_\_\_\_\_\_\_

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc196466592)

[1 РАЗРАБОТКА ОБЩЕЙ СТРУКТРУЫ МИКРО-ЭВМ 5](#_Toc196466593)

[1.1 Функциональный состав микро-ЭВМ 5](#_Toc196466594)

[1.2 Разработка архитектуры команд процессора 5](#_Toc196466595)

[1.3 Описание взаимодействия блоков в микро-ЭВМ 7](#_Toc196466596)

[2 РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ 9](#_Toc196466597)

[2.1 Блок памяти 9](#_Toc196466598)

[3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ 12](#_Toc196466599)

[3.1 Функциональное моделирование программы для тестирования 12](#_Toc196466600)

**ВВЕДЕНИЕ**

Микропроцессорные вычислительные устройства играют ключевую роль в современной цифровой инфраструктуре. Их применение охватывает широкий спектр задач – от автоматизации бытовых приборов до сложных управляющих систем в аэрокосмической отрасли. Благодаря своей компактности, энергоэффективности и доступности, микро-ЭВМ открывают широкие возможности для создания интеллектуальных, надежных и высокопроизводительных решений, способных эффективно справляться с различными задачами в реальном времени.

С развитием цифровых технологий все большую актуальность приобретает проектирование специализированных микропроцессорных архитектур, адаптированных под конкретные задачи. Программируемые логические интегральные схемы (ПЛИС) становятся одним из наиболее гибких и универсальных инструментов для создания таких архитектур. Использование ПЛИС позволяет не только сократить сроки разработки, но и добиться высокой степени настраиваемости аппаратной части системы.

Данный курсовой проект посвящен разработке микро-ЭВМ на базе архитектуры фон Неймана (Принстонской архитектуры), реализуемой на ПЛИС. Особенностью этой архитектуры является единое адресное пространство для команд и данных, что упрощает реализацию системы и снижает её стоимость. Вместе с тем, такое решение накладывает определённые ограничения на производительность из-за необходимости разделения доступа к памяти по одной шине.

Проектирование выполняется с использованием среды Altera Quartus II 9.1, которая обеспечивает средства для создания и отладки цифровых схем, а также функциональное моделирование и симуляцию работы спроектированной микро-ЭВМ.

**1 РАЗРАБОТКА ОБЩЕЙ СТРУКТРУЫ МИКРО-ЭВМ**

**1.1 Функциональный состав микро-ЭВМ**

Так как выполнение некоторых условий задания не является строго обязательным, в проектируемой системе не предусмотрены такие элементы, как конвейер, кэш, КПДП и блок предсказания переходов. Таким образом, команды будут поступать на выполнение последовательно, причём следующая команда не будет выполняться, пока не текущая команда не закончила выполнение.

В Принстонской архитектуре память данных и память команд представляют из себя один функциональный блок. Центральный процессор (ЦП) в свою очередь состоит из нескольких более мелких логических блоков. Общая схема устройства включает в себя следующие блоки:

– блок памяти;

– микропроцессор.

Центральный процессор в свою очередь включает в себя блоки:

– устройство выборки команд;

– декодер команд;

– устройство чтения операндов;

– устройство выполнения команд;

– устройство записи результата;

– регистры общего назначения (РОН);

– устройство управления (УУ);

– промежуточные буферы.

Устройство выполнения команд также является сложным устройством и состоит из следующих блоков:

– арифметико-логическое устройство (АЛУ);

– стек;

– блок флагов.

Далее рассмотрим принцип работы каждого устройства отдельно, а также его интерфейс.

Структурная схема проектируемой микро-ЭВМ представлена в приложении ГУИР.400201.110 Э1.

**1.2 Разработка архитектуры команд процессора**

В связи с тем, что нужное количество информации невозможно переслать за один такт работы процессора, было принято решение сформировать полноценную команду из двух машинных слов по 15 бит. За первый такт пересылается код операции (КОП) и адрес первого операнда, который может быть как в общей памяти, так и в регистрах общего назначения.

Стоит отметить, что некоторые команды не требуют данных, которые принимается на отдельных тактах. Было принято решение записывать нули на место «безразличных» битов.

Для простоты реализации, каждой команде присваивается порядковый номер, позволяющий однозначно определить, какая команда поступила на выполнение. Общее количество команд по варианту равно 16. Следовательно, функционально для идентификации команды рационально использовать дешифратор на 4 бита. Но в данном случае было решено использовать дешифратор на 6 битов для упрощения работы с шиной данных. Старшие два бита в коде операции будут “безразличными”, по умолчанию они будут инициализироваться нулями.

В таблице 1.1 приведена структура команды разрабатываемого микро-ЭВМ.

Таблица 1.1 – Структура команды микро-ЭВМ

|  |  |  |
| --- | --- | --- |
| Первое слово | | |
| Биты 29-24 | Биты 23-15 | |
| КОП | Адрес операнда блока памяти  (MEM) | |
| Второе слово | | |
| Биты 14-11 | | Биты 10-0 |
| Адрес операнда в памяти общего назначения (R) | | Безразличные биты |

В таблице 1.2 представлен список команд микро-ЭВМ с соответствующими типами адресации и кодами операций.

Таблица 1.2 – Архитектура системы команд микро-ЭВМ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Команда | Первое слово | | Второе слово | |
| 1 | 2 | 3 | | 4 | |
| 1 | MOV (MEM → R1) | 000000 | MEM | R1 | XXXXXXXXXXX |
| 2 | MOV (R1 → MEM) | 000001 | MEM | R1 | XXXXXXXXXXX |
| 3 | MOV (R1 → R2) | 000010 | R1 | R2 | XXXXXXXXXXX |
| 4 | PUSH (R1) | 000011 | R1 | XXXX | XXXXXXXXXXX |
| 5 | POP (R1) | 000100 | R1 | XXXX | XXXXXXXXXXX |
| 6 | SUB(MEM, R1) → MEM | 000101 | MEM | R1 | XXXXXXXXXXX |
| 7 | SUB(R2, R1) → R2 | 000110 | R2 | R1 | XXXXXXXXXXX |
| 8 | XOR(MEM, R1) → MEM | 000111 | MEM | R1 | XXXXXXXXXXX |
| 9 | XOR(R2, R1) → R2 | 001000 | R2 | R1 | XXXXXXXXXXX |
| 10 | NAND(MEM, R1) → MEM | 001001 | MEM | R1 | XXXXXXXXXXX |
| 11 | NAND(R2, R1) → R2 | 001010 | R2 | R1 | XXXXXXXXXXX |
| 12 | ROR(MEM, R1) → MEM | 001011 | MEM | R1 | XXXXXXXXXXX |
| 13 | ROR(R2, R1) → R2 | 001100 | R2 | R1 | XXXXXXXXXXX |
| 14 | JMP(MEM) | 001101 | MEM | XXXX | XXXXXXXXXXX |
| 15 | JAZ(MEM) | 001110 | MEM | XXXX | XXXXXXXXXXX |

Продолжение таблицы 1.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 16 | HLT | 111111 | XXXXXXXXX | XXXX | XXXXXXXXXXX |
| Примечание – X обозначает безразличный бит, MEM - адрес операнда блока памяти, R1 и R2 – номера регистров из РОН. | | | | | |

**1.3 Описание взаимодействия блоков в микро-ЭВМ**

Для описания взаимодействия основных блоков микро-ЭВМ, стоит представить типовую последовательность действий, заложенную в функционал микропроцессора, а именно исполнение команд.

Память команд в Принстонской архитектуре совмещена с памятью данных. Это свойство реализует блок памяти, который сочетает в себе оперативное запоминающее устройство (ОЗУ) и постоянное запоминающее устройство (ПЗУ), тем самым алгоритм считывания команд и данных выглядит достаточно просто. На вход данного блока поступает адрес, по которому будет производиться считывание. На выходе получаем содержимое данной ячейки.

Устройство управления считает количество выполнения команд по сигналу доступа к работе считывателя команд, что фактически эквивалентно началу выполнения команды в микро-ЭВМ.

В начале устройство чтения команд запрашивает доступ к началу своей работы. УУ обрабатывает запрос и решает, выдать ли разрешение на работу. Получив разрешение, устройство выборки команд получает данные из блока памяти по адресу. После считывания двух слов блок выборки команд сохраняет данные в промежуточный буфер между блоком чтения команд и декодером команд, а также посылает сигнал готовности.

Декодер формирует адрес назначения для записи результата, адреса операндов, тип операндов. Также, если полученная команда – это HLT, то именно декодер команд посылает сигнал для остановки работы ЦП. К тому же блок отвечает за условные и безусловные переходы, так как при обнаружении команды перехода, он отправляет сигналы запросов на переход в устройство управление, которое в свою очередь уже управляет осуществлением перехода посредством координации блока чтения команд.

Устройство чтения операндов регулирует доступ к памяти или регистрам общего назначения посредством шины управления. Оно посылает запрос на доступ к работе в устройство управления (УУ) и получает разрешение на работу. После разрешения на работу блок устанавливает адрес на шину адреса (ША), управляющие сигналы на шину управления (ШУ) и получает запрашиваемы данные по шине данных (ШД). Все данные блок чтения операндов сохраняет в промежуточный буфер, который находится между устройством чтения операндов и устройством выполнения команд.

Устройство выполнения команд в зависимости от полученной команды направляет данные из буфера на АЛУ или стек. В АЛУ кроме результата формируются специальные флаги SE, SF, ZF, CF, которые сохраняются в блоке флагов. Если распознанная команда – это PUSH или POP, то управление блоком переходит в стек. Промежуточный буфер получает данные из устройства выполнения команд и отправляет эти данные далее – на устройство записи результата.

Устройство записи результата, если ему выдано разрешение на работу, выставляет адрес на ША, данные для записи – на ШД и сообщает и своей готовности записать данные. УУ посылает на ШУ сигнал записи данных, если было зафиксирована готовность устройства записи результата. После этого данные могут быть сохранены либо в РОН, либо в блоке памяти.

**2 РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ**

**2.1 Блок памяти**

Так как пристанская архитектура требует использования принципа общей памяти, то было принято решение: объединить блок ПЗУ (постоянного запоминающего устройства) и блок ОЗУ (Оперативного запоминающего устройства) в один компонент. Условное графическое обозначение (УГО) блока памяти представлено на рисунке 2.1.

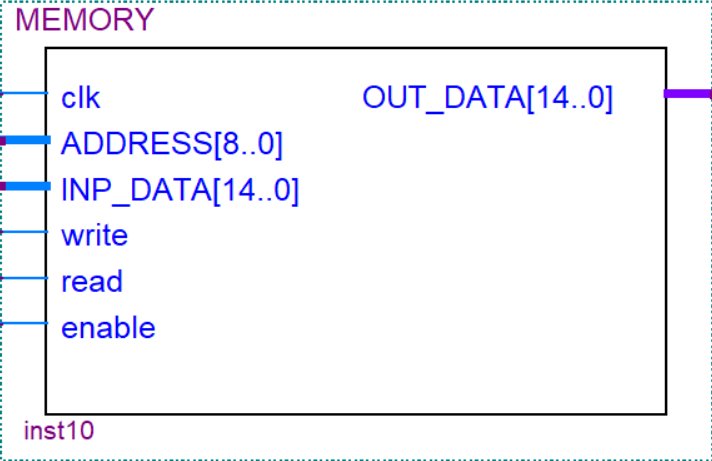


Рисунок 2.1 – УГО блока памяти

Данный блок имеет следующие входные сигналы:

– clk – тактовый сигнал;

– ADDRESS[8..0] – шина адреса для доступа к необходимой ячейке;

– INP\_DATA[14..0] – шина данных для записи информации в память;

– write – сигнал для разрешения записи в память;

– read – сигнал для разрешения чтения из памяти;

– enable – сигнал, разрешающий работу блока.

Также выходные сигналы:

– OUT\_DATA[14..0] – шина для вывода данных из памяти;

Так как ПЗУ является асинхронным, а ОЗУ синхронным, следовательно для вывода данных из ОЗУ необходимо использовать синхросигнал, в отличии от ПЗУ, где данные выходят постоянно.

Общий объем памяти равен 29 = 512, так как разрядность шины данных равна 9 битам. Для регулирования обращения к ОЗУ или ПЗУ используется старший бит адреса, если он равен единице, то обращение будет происходить к ОЗУ. Также в ПЗУ нельзя записывать данные. Для работы блока необходимо подать сигнал enable вместе с сигналом clk.

Чтение осуществляется за 2 такта. Для получение необходимых данных нужно подавать необходимый адрес на протяжении 2 тактов, а также read на протяжении 2-го такта для вывода данных.

Для записи необходимо также подавать адрес на протяжении 2 тактов, а также сигнал write.

Моделирование блока памяти представлено на рисунке 2.2.

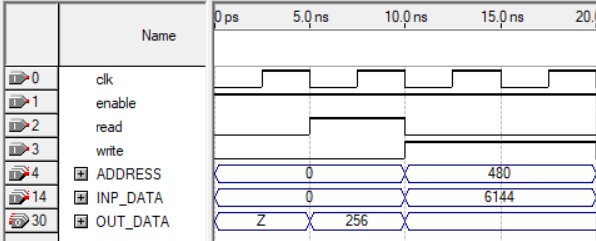


Рисунок 2.2 – Моделирование блока памяти

На моделировании представлен процесс чтения данных по 0 адресу из ПЗУ, а также запись 6144 по 480 адресу в ОЗУ.

Функциональная схема блока памяти приведена в приложении ГУИР.400201.111 Э2.1.

**2.2 Устройство управления**

Устройство управления необходимо для координации работы всей схемы, так как оно подает сигналы доступа для работы различных частей микро-ЭВМ. Условное графическое обозначение устройства управления представлено на рисунке 2.3.

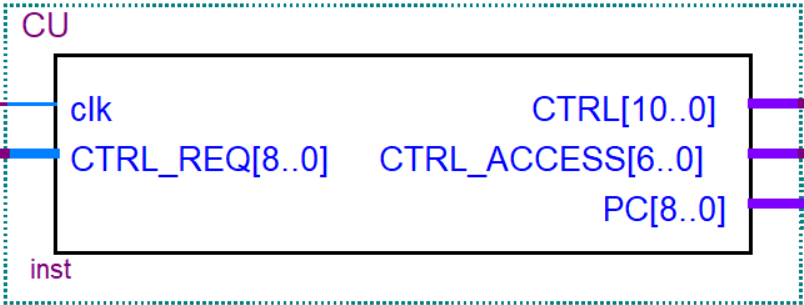


Рисунок 2.3 – УГО устройства управления

Данный блок имеет следующие входные сигналы:

– clk – тактовый сигнал;

– CTRL\_REQ[8..0] –шина запросов для доступа устройства к работе.

Также выходные сигналы:

– CTRL [10..0] –шина для управления доступом к чтению и записи данных из памяти или регистров общего назначения;

– CTRL\_ACCESS [6..0] –шина предоставления доступа к работе различных компонентов схемы;

– PC [8..0] – шина счетчика команд.

Блок получается запросы от различных компонентов микро-ЭВМ и в зависимости от условий решает предоставлять ли им доступ к работе или нет. Разрешение работы устройству для чтения команды разрешается, если получен запрос на завершение выполнения команды либо просто при отсутствии других запросов и подаче синхросигнала для старта работы схемы.

После завершения чтения команды, устройство чтения подает запрос на разрешение работы устройства для чтения необходимых операндов, после прочтения операндов подается сигнал запроса для разрешения работы устройства для выполнения необходимой операции, после выполнения операции подается сигнал, позволяющий работать устройству записи результата. После записи результата операции считается завершенной и подается сигнал чтения следующей команды.

Моделирование блока представлено на рисунке 2.4.

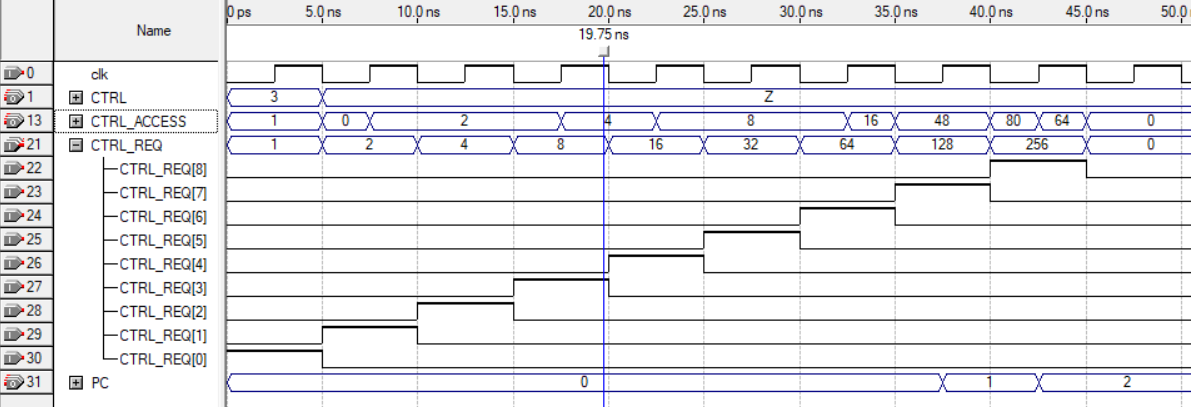


Рисунок 2.4 – Моделирование управляющего устройства

**3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ**

**3.1 Функциональное моделирование программы для тестирования**

Общая функциональная схема разработанной микро-ЭВМ представлена в приложении ГУИР.400201.110 Э2.18. Программа для тестирования приведена в таблице 3.1.

Таблица 3.1 – Программа для тестирования микро-ЭВМ

|  |  |  |  |
| --- | --- | --- | --- |
| Счетчик команд | Символьный вид | Бинарный вид | Пояснение |
| 1 | 2 | 3 | 4 |
| 00 | MOV $256, R0 | 000000100000000  000000000000000 | [$256] = -122 |
| 01 | MOV $259, R1 | 000000100000000  000000000000000 | [$259] = 1488 |
| 02 | MOV R0, R3 | 000010000000000  001100000000000 | [R0] = -122 |
| 03 | NAND R0, R3 | 001010000000000  001100000000000 | [R0] = -122,  [R3] = -122 |
| 04 | JAZ $0 | 001110000000000  000000000000000 | ZF = 0, SF = 0  Перехода не будет |
| 05 | SUB $352, R1 | 000101101100000  000100000000000 | [$352] = 56  [R1] = 1488 |
| 06 | JAZ $90 | 000111001011010  000000000000000 | ZF = 0, SF = 1  Переход будет |
| 07 | XOR $260, R1 | 000111000000100  000100000000000 | [$260] = 22  [R1] = 1488 |
| 08 | PUSH R3 | 000011000000011  000000000000000 | [R3] = -122 |
| 09 | MOV R3, $261 | 000001100000101  001100000000000 | [R3] = -122 |
| 10 | MOV $288, R4 | 000000100100000  010000000000000 | [$288] = 91 |
| 11 | MOV $289, R5 | 000000100100001  010100000000000 | [$288] = 234 |
| 12 | MOV R5, R7 | 000010000000101  011100000000000 | [R5] = 234 |
| 13 | PUSH R4 | 000011000000100  000000000000000 | [R4] = 91 |
| 14 | PUSH R7 | 000011000000111  000000000000000 | [R7] = 234 |

Продолжение таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | POP R0 | 000100000000000  000000000000000 | [TOP] = 234 |
| 16 | PUSH R3 | 000011000000011  000000000000000 | [R3] = -122 |
| 17 | JMP $144 | 001101010010000  000000000000000 | Переход будет |
| 18 | ROR $400, R1 | 001011110010000  000100000000000 | [$400] = 1,  [R1] = 1488 |
| 19 | SUB R7, R5 | 000110000000111  010100000000000 | [R7] = 234,  [R5] = 234 |
| 20 | ROR R1, R4 | 001100000000001  010000000000000 | [R1] = 1488,  [R4] = 91 |
| 21 | NAND $401, R7 | 001001110010001  011100000000000 | [$401] = -2,  [R7] = 0 |
| 22 | XOR $402, R1 | 000111110010010  000100000000000 | [$402] = -100,  [R1] = -8960 |
| 23 | MOV R7, $410 | 000001110011010  011100000000000 | [R7] = 0 |
| 24 | HLT | 11111010010000  000000000000000 | Остановка работы ЭВМ |

Дамп части памяти, которая содержит данные о выполняемых командах представлена на рисунке 3.1.

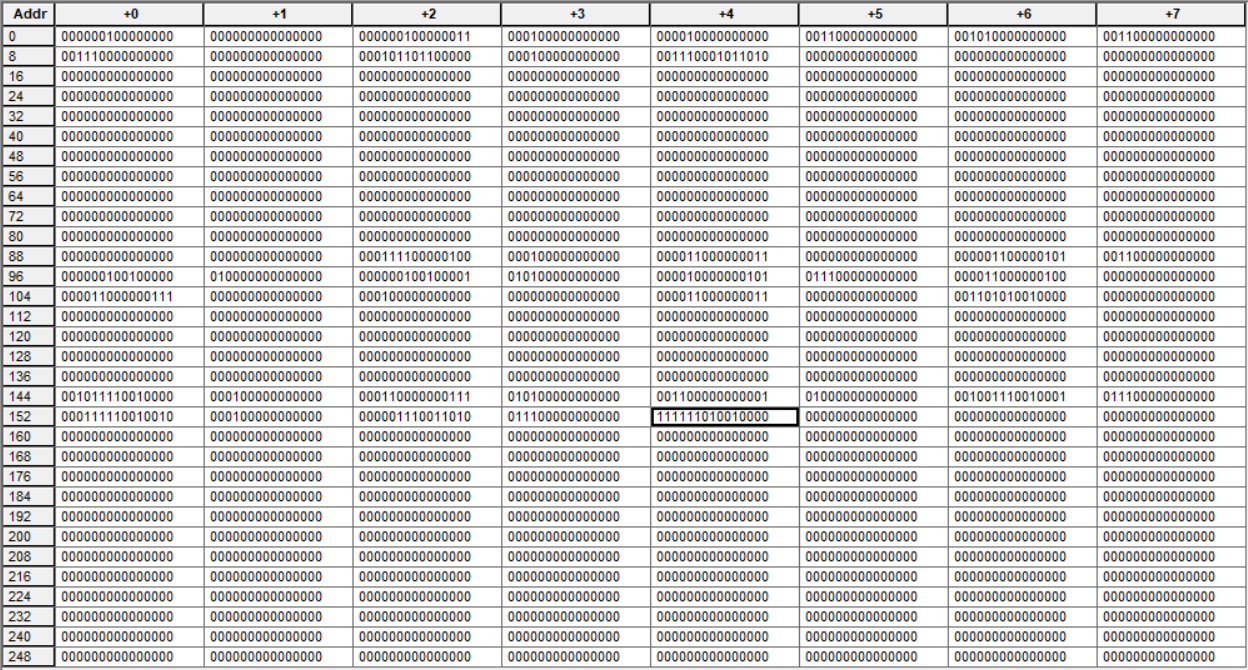


Рисунок 3.1 – Дамп части памяти, содержащую программу

Дамп части памяти, которая содержит операнды для выполнения программы, а также результаты операций программы, до и после выполнения программы представлен на рисунках 3.2 и 3.3 соответственно.

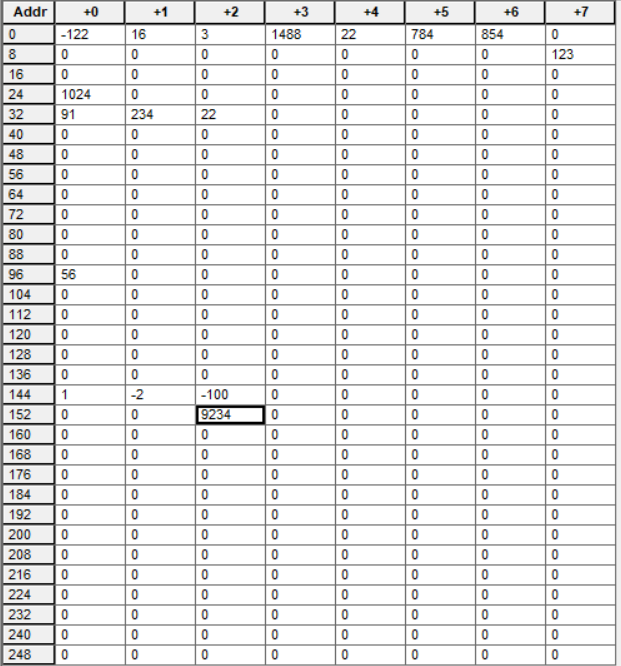


Рисунок 3.2 – Дамп части памяти с операндами и результатами до выполнения



Рисунок 3.3 – Дамп части памяти с операндами и результатами после выполнения

Результаты моделирования программы представлены на рисунках 3.4 – 3.28.

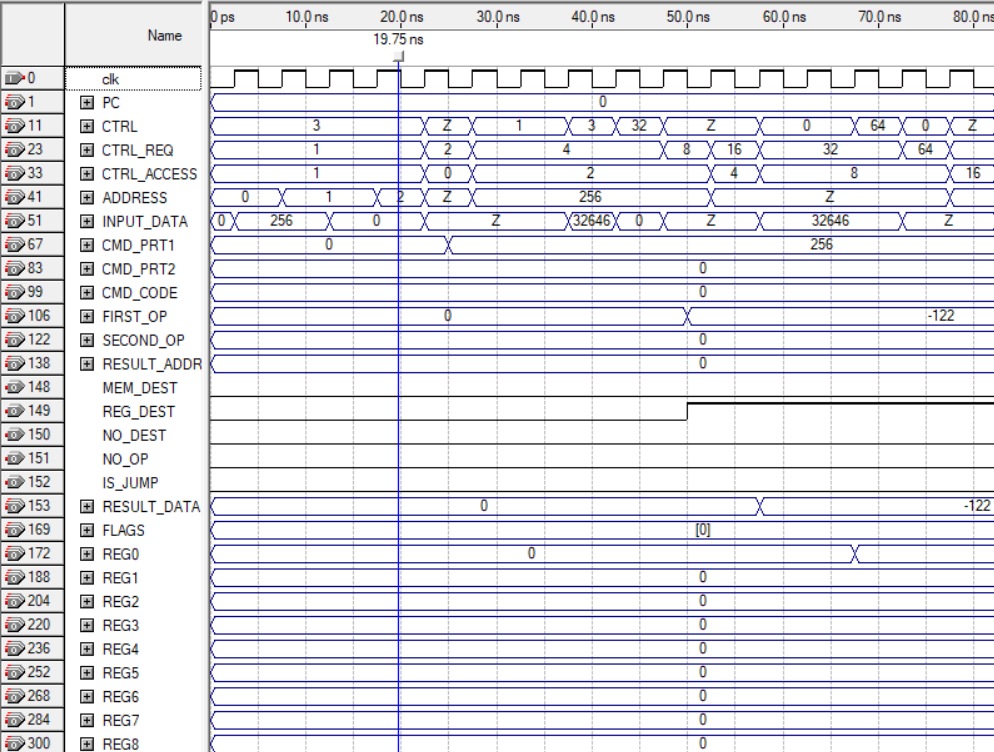


Рисунок 3.4 – Моделирование первой команды

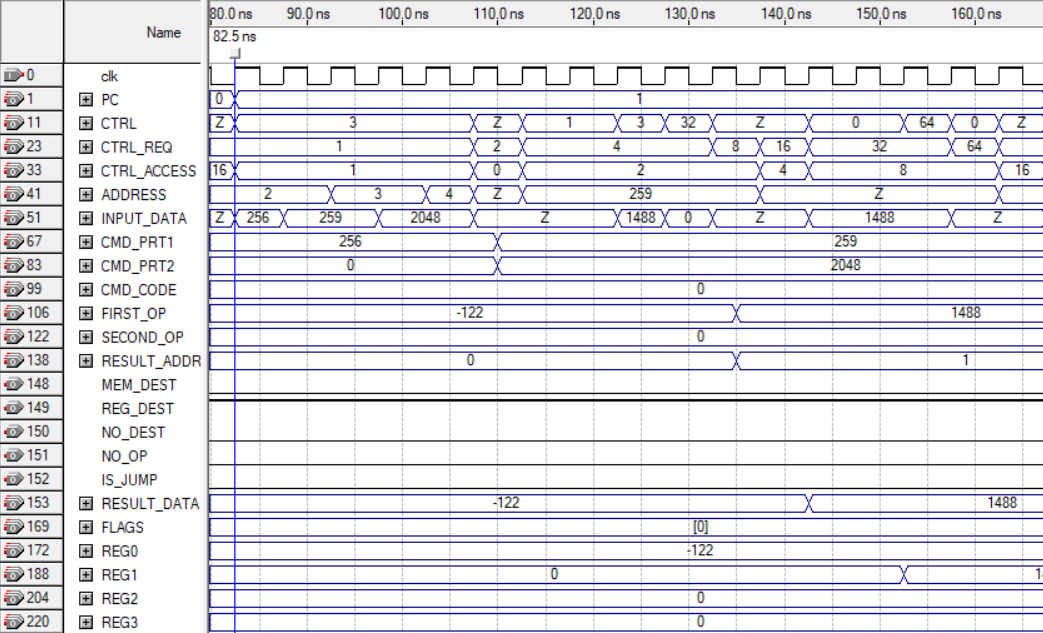


Рисунок 3.5 – Моделирование второй команды

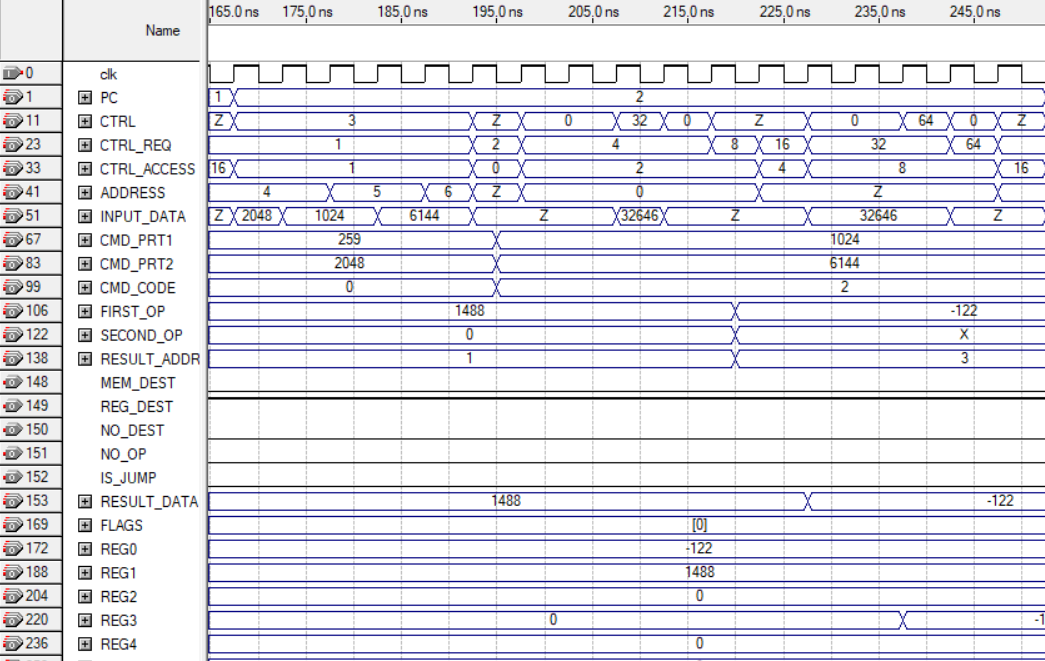


Рисунок 3.6 – Моделирование третей команды

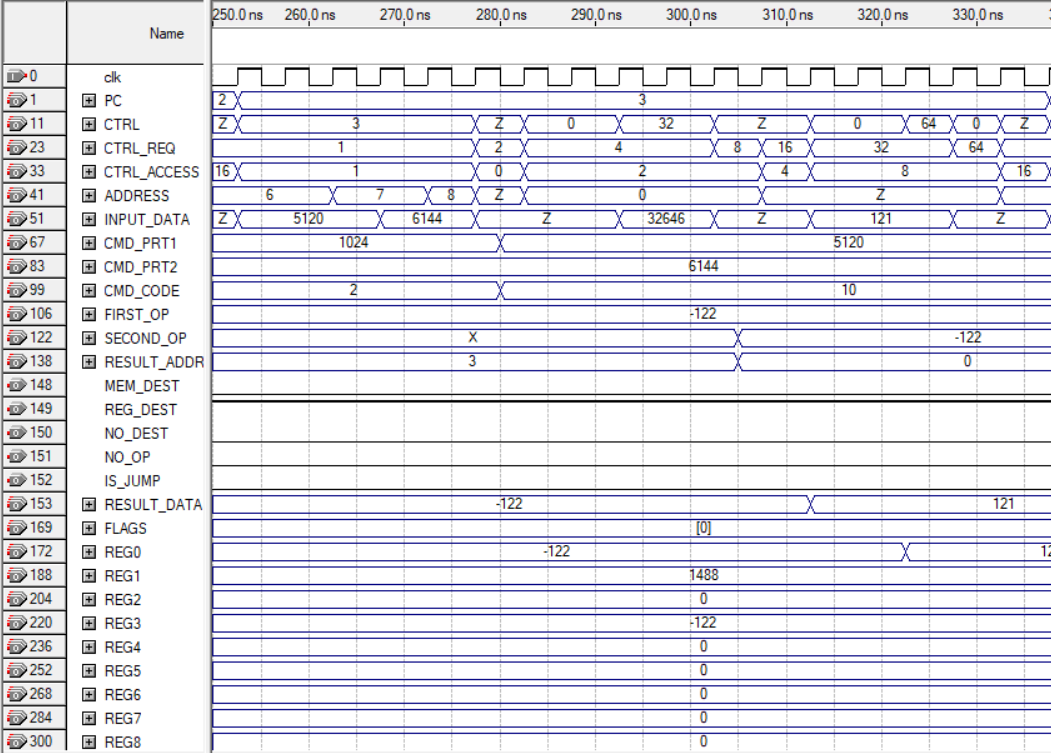


Рисунок 3.7 – Моделирование четвертой команды

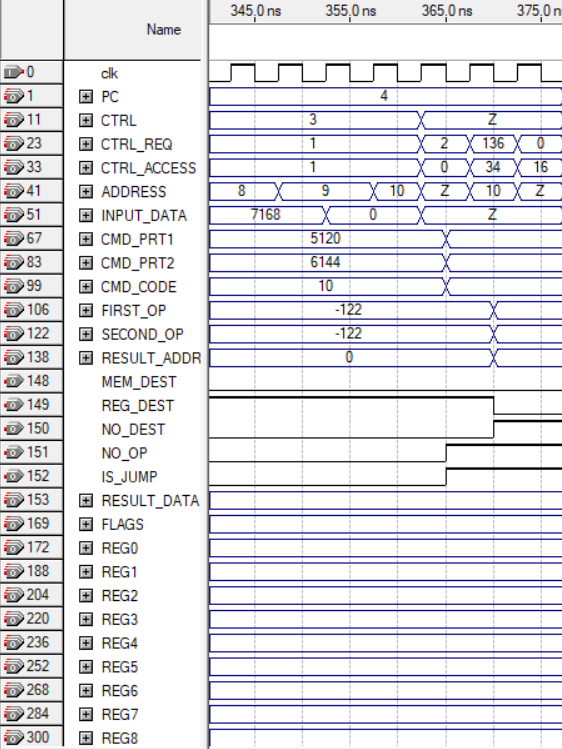


Рисунок 3.8 – Моделирование пятой команды

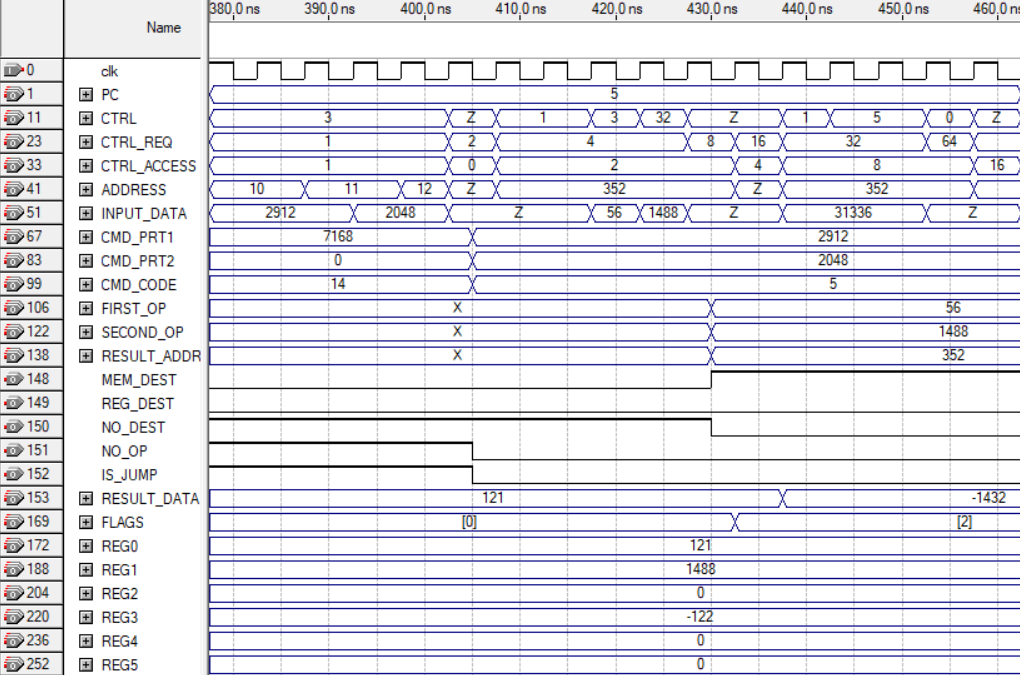


Рисунок 3.9 – Моделирование шестой команды

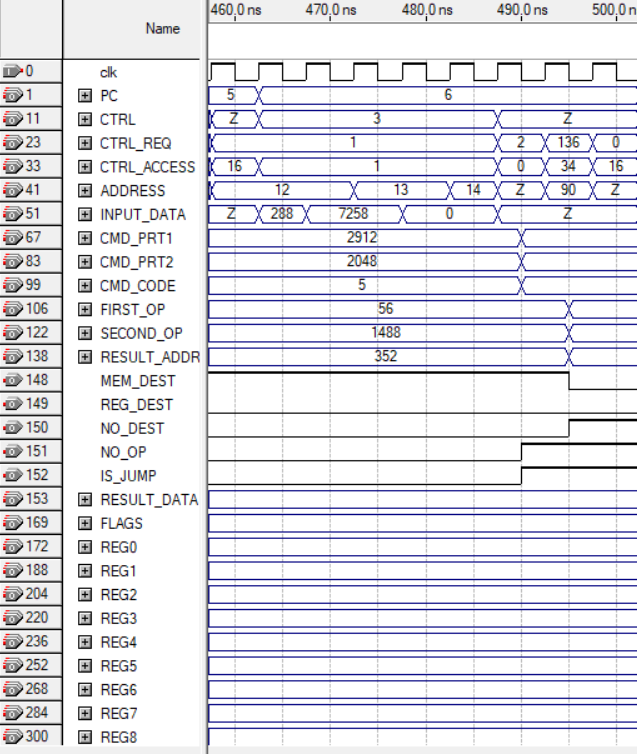


Рисунок 3.10 – Моделирование седьмой команды

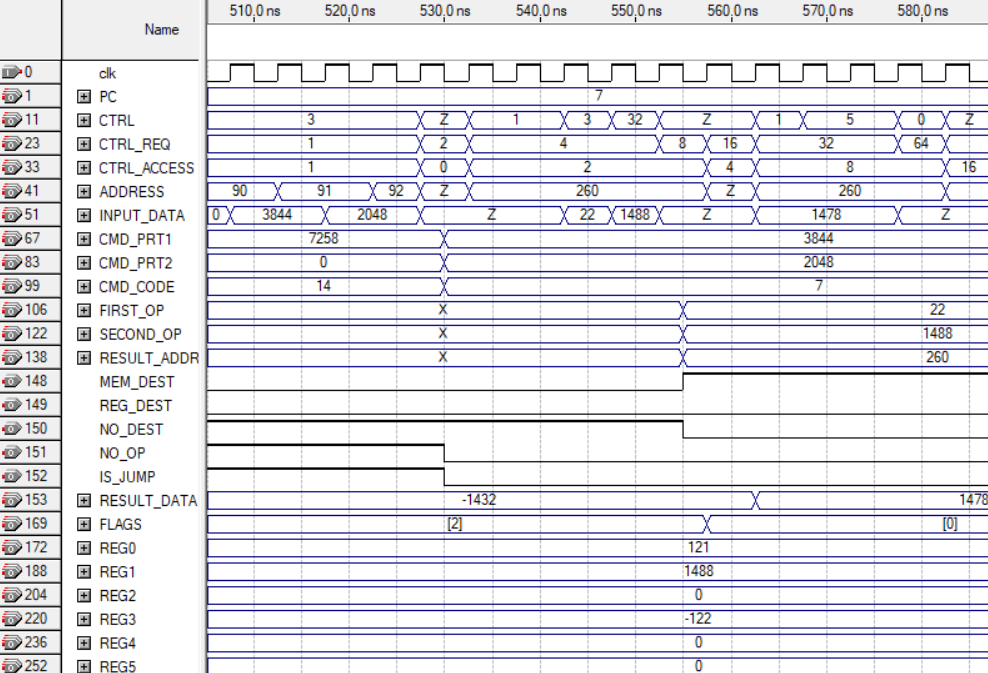
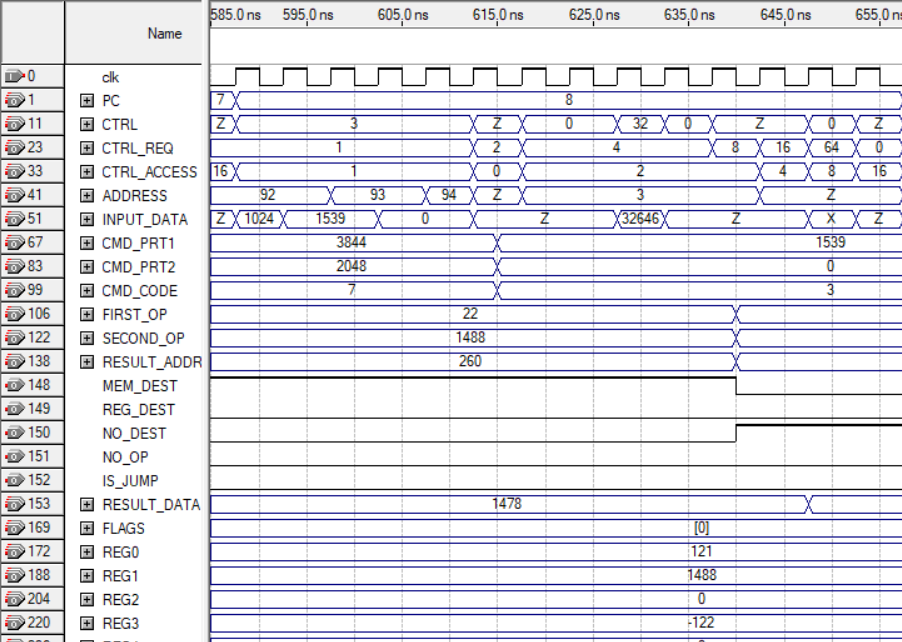


Рисунок 3.11 – Моделирование восьмой команды



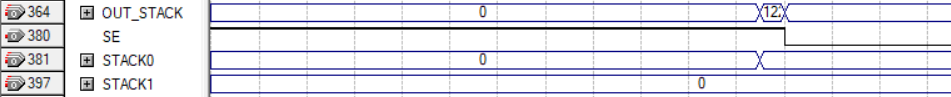


Рисунок 3.12 – Моделирование девятой команды

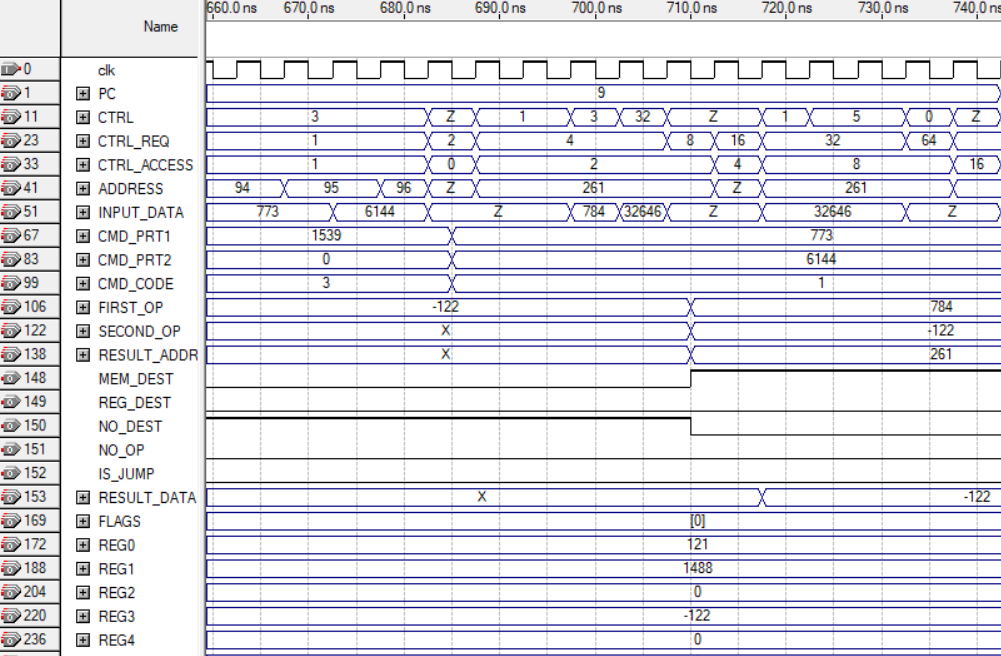


Рисунок 3.13 – Моделирование десятой команды

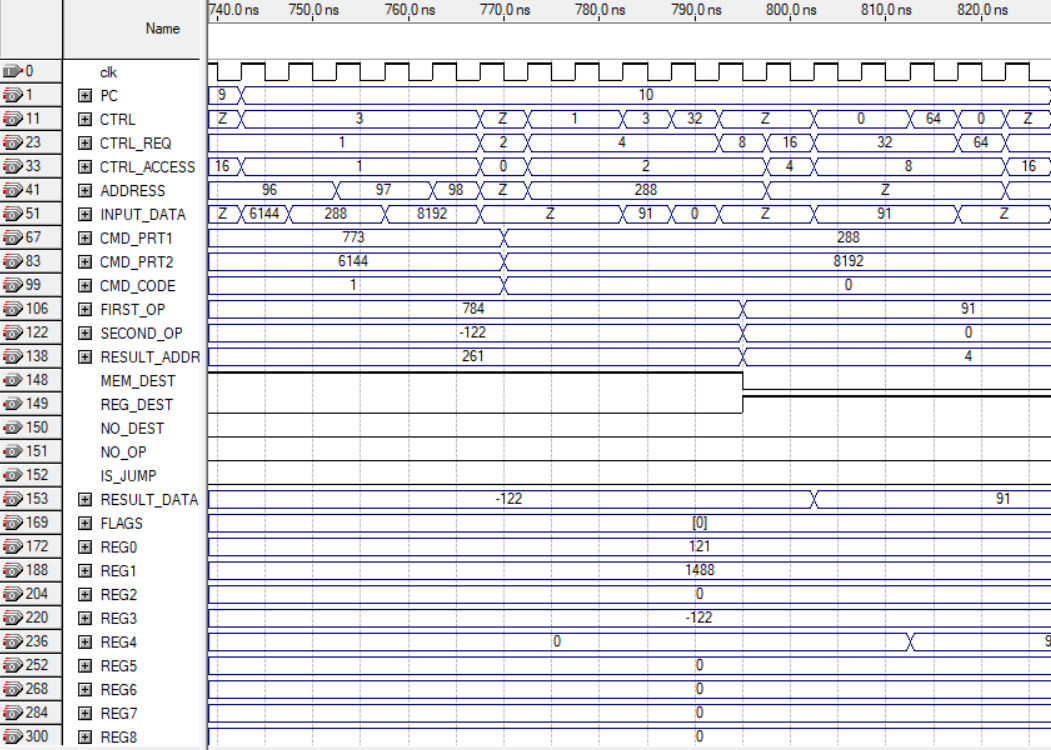


Рисунок 3.14 – Моделирование одиннадцатой команды

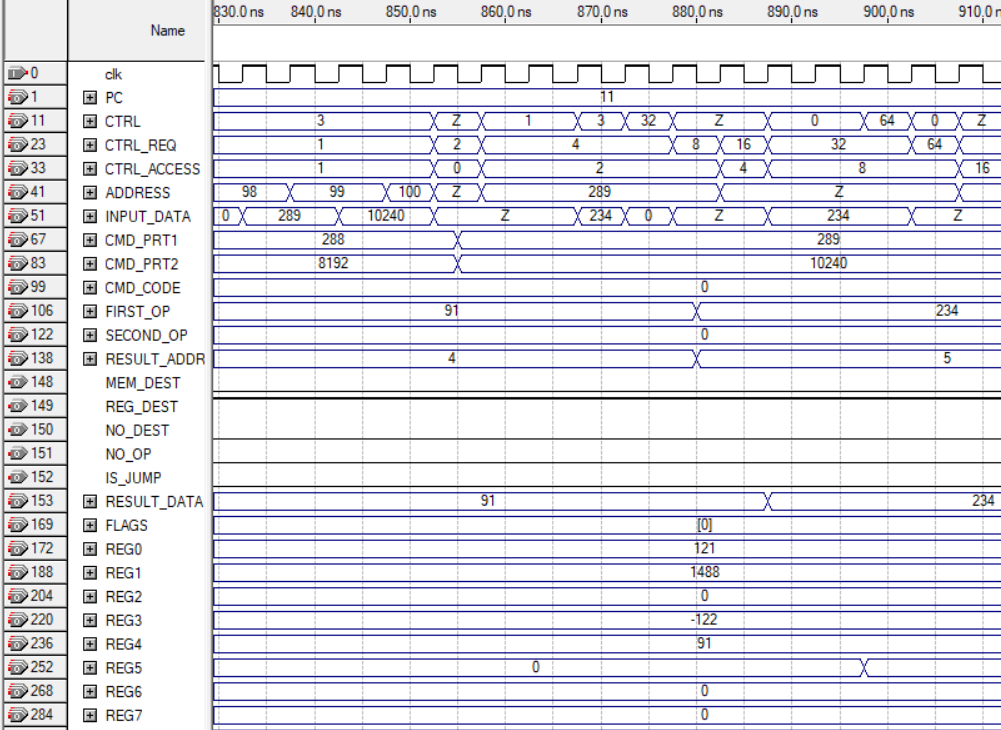


Рисунок 3.15 – Моделирование двенадцатой команды

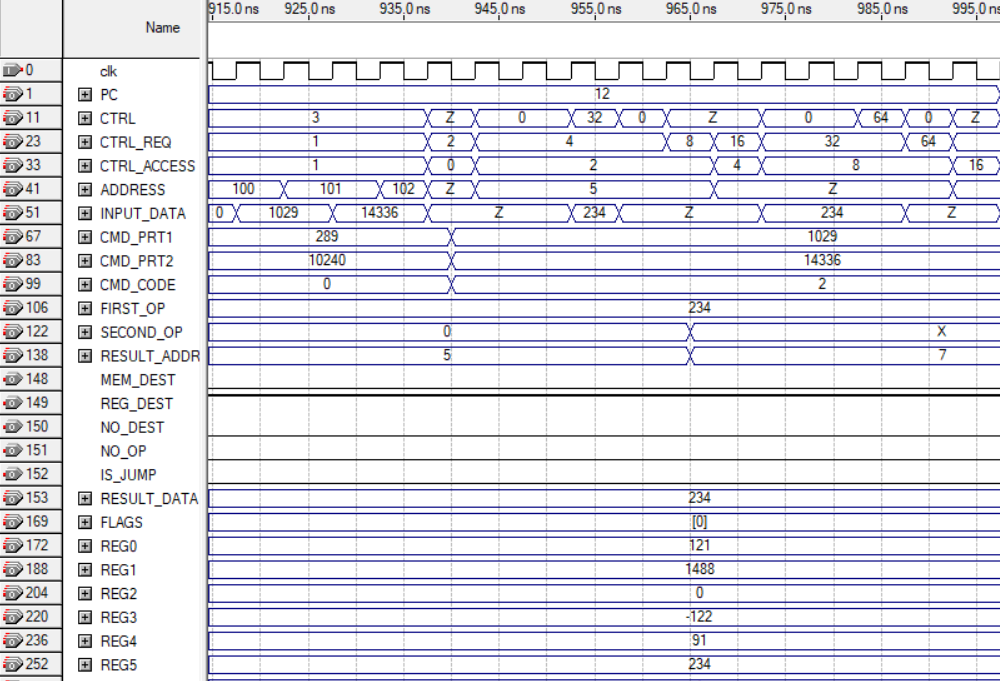
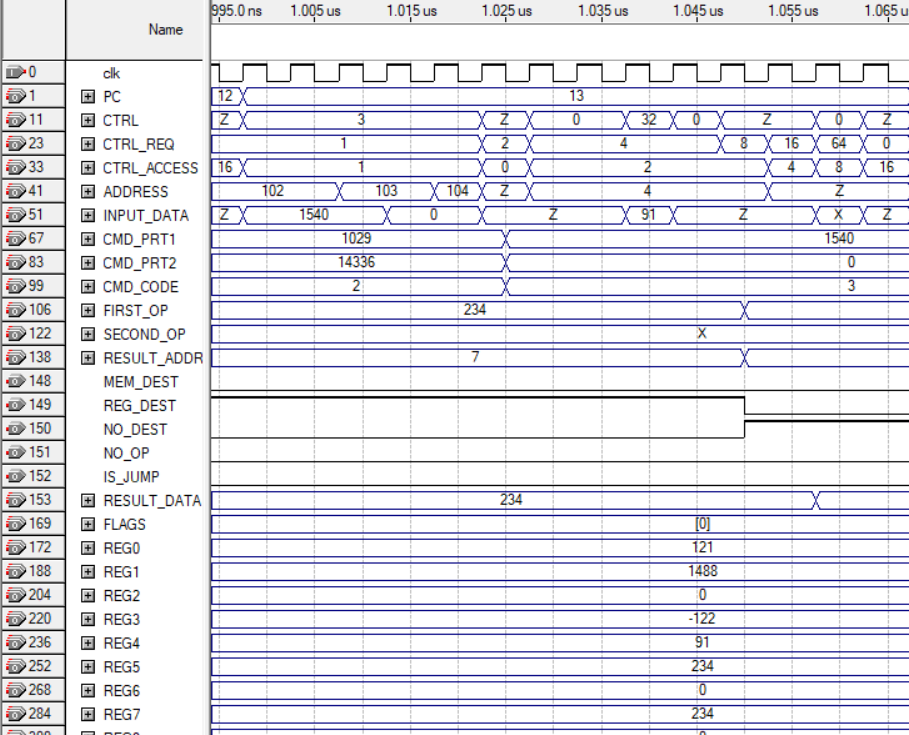


Рисунок 3.16 – Моделирование тринадцатой команды



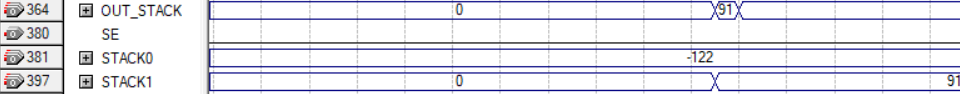
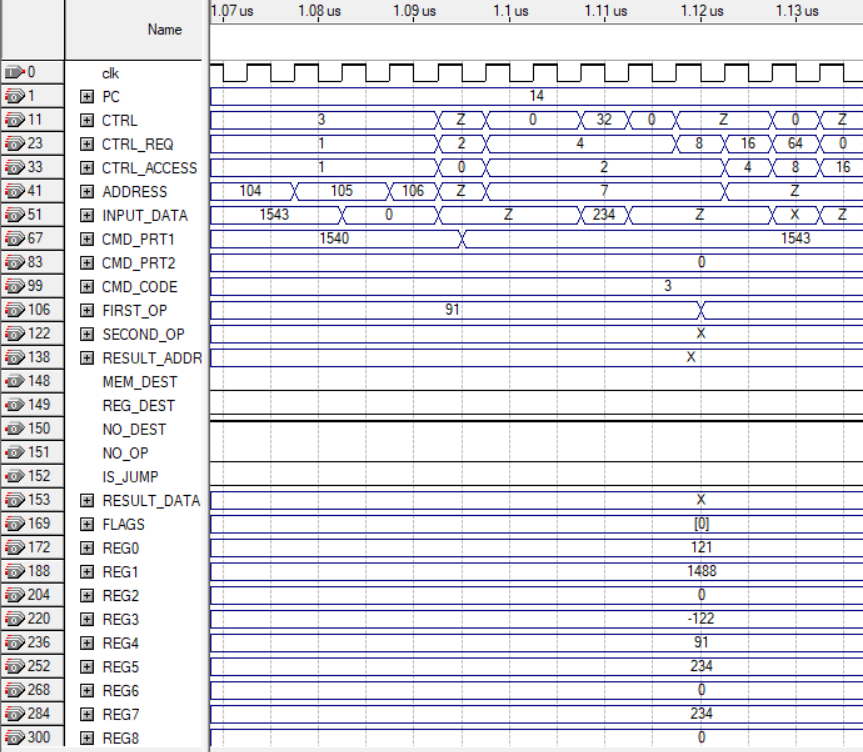


Рисунок 3.17 – Моделирование четырнадцатой команды



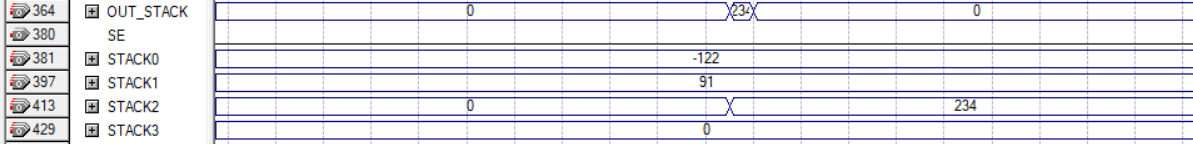
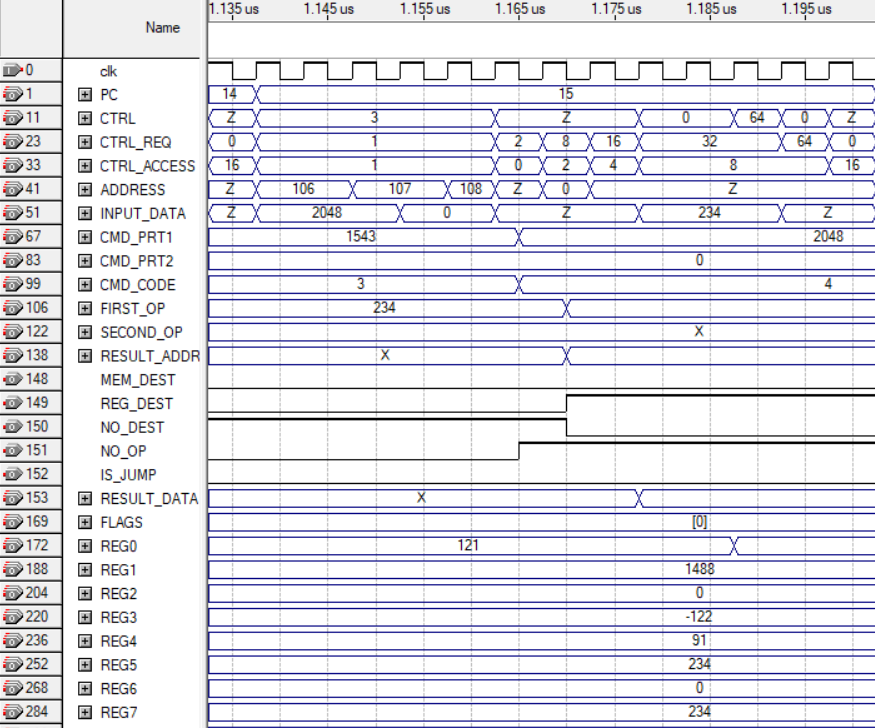


Рисунок 3.18 – Моделирование пятнадцатой команды



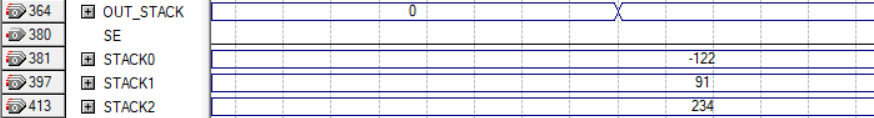
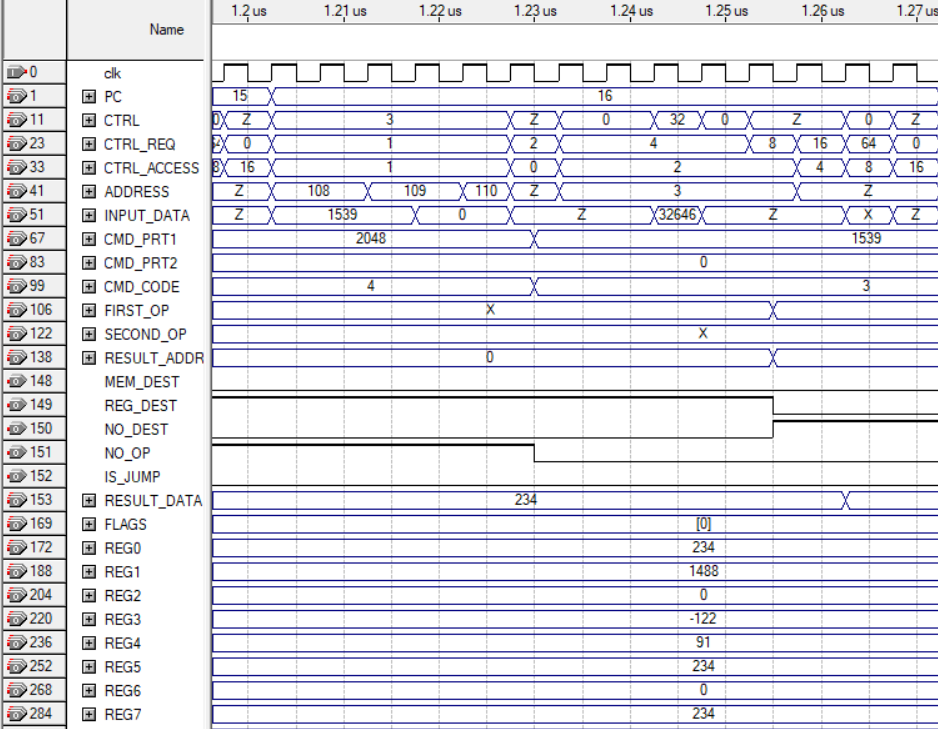


Рисунок 3.19 – Моделирование шестнадцатой команды



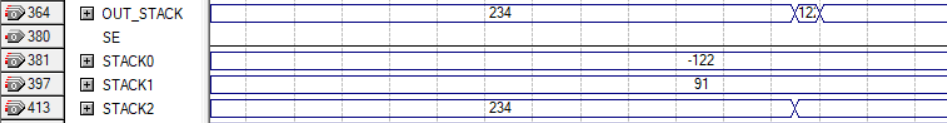


Рисунок 3.20 – Моделирование семнадцатой команды

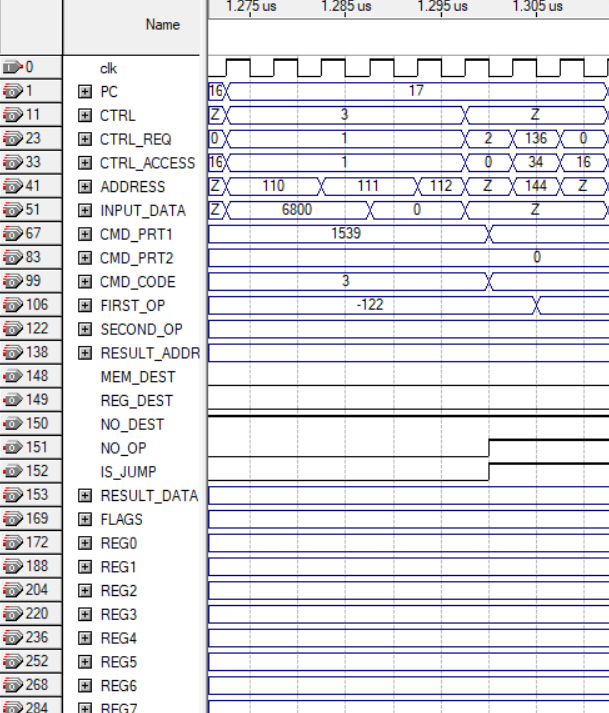


Рисунок 3.21 – Моделирование восемнадцатой команды

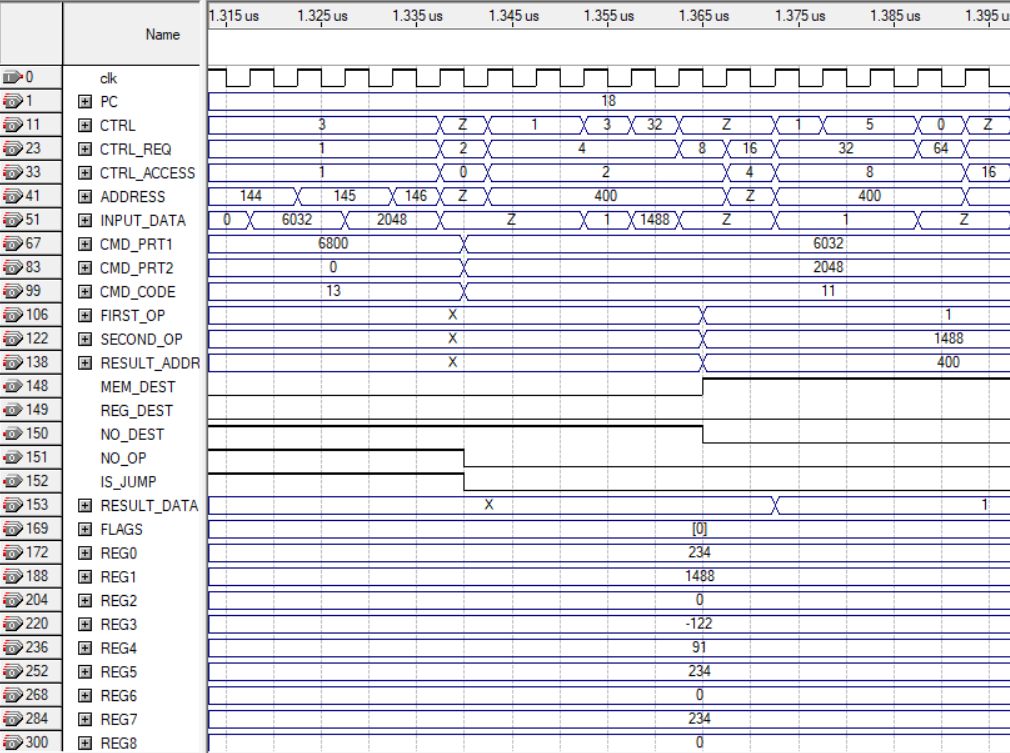


Рисунок 3.22 – Моделирование девятнадцатой команды

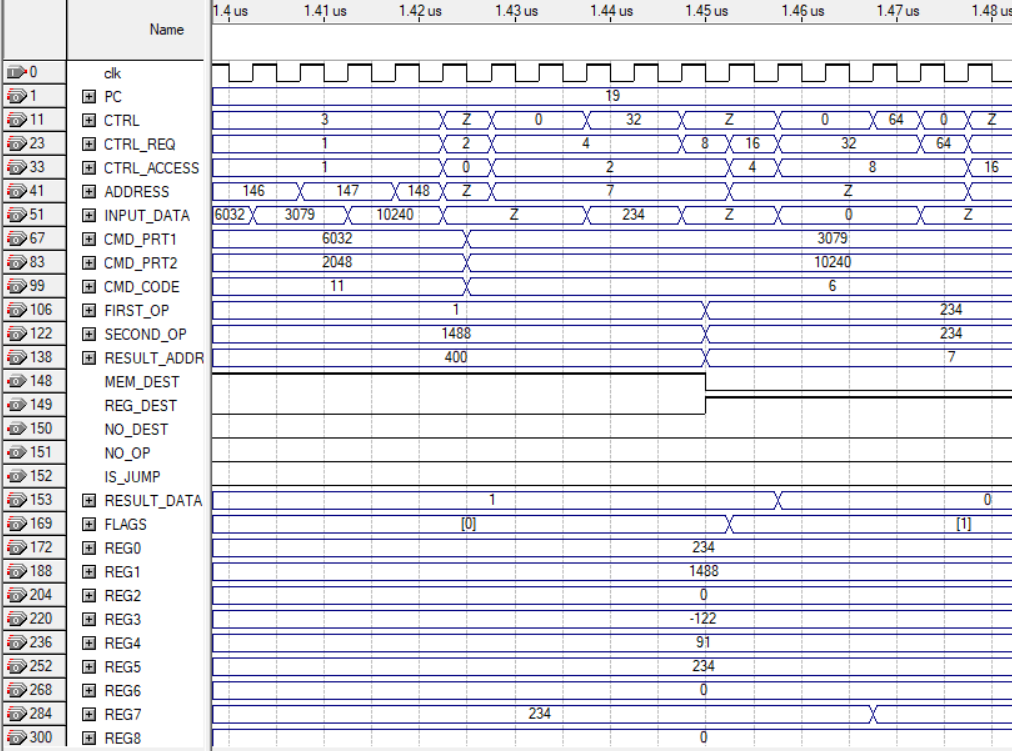


Рисунок 3.23 – Моделирование двадцатой команды

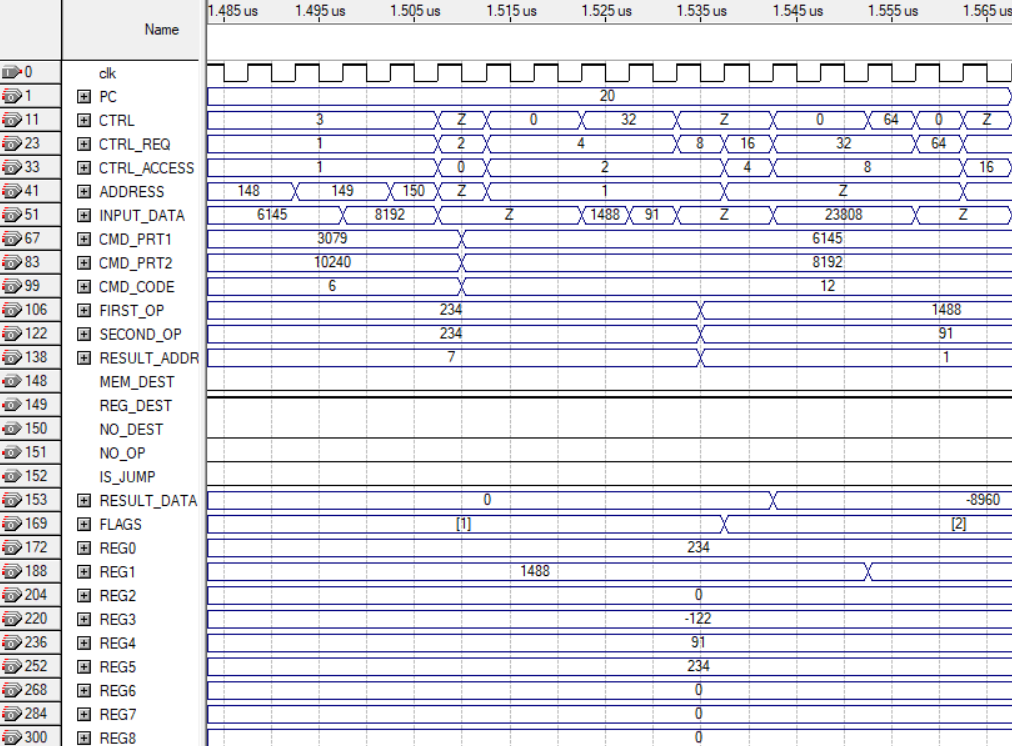


Рисунок 3.24 – Моделирование двадцать первой команды

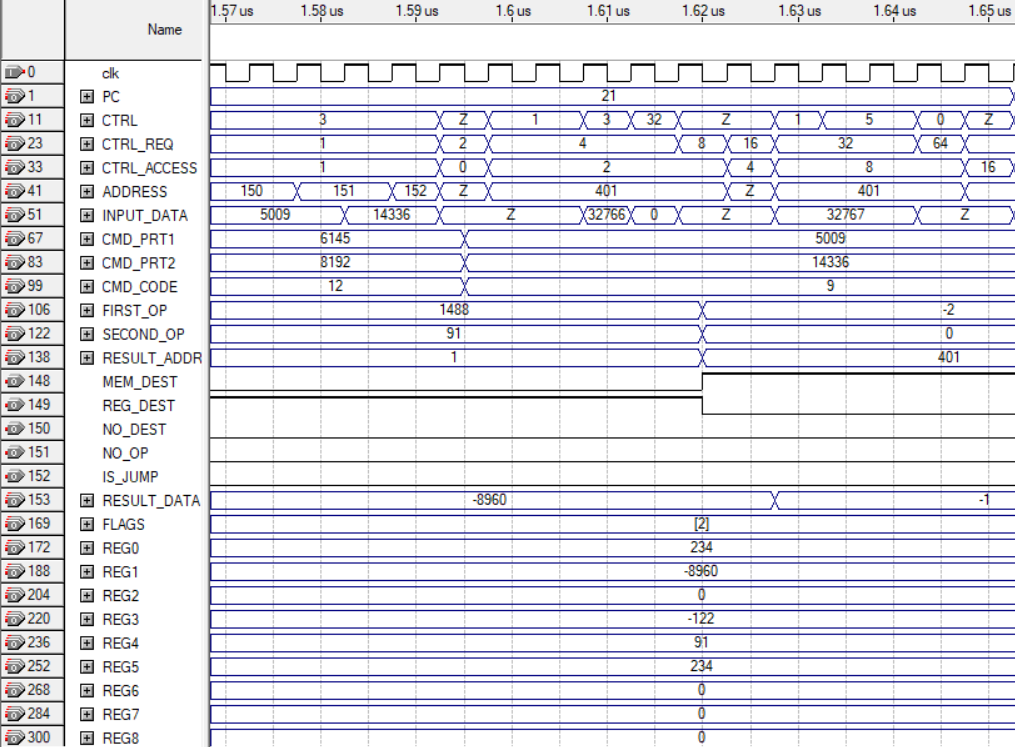


Рисунок 3.25 – Моделирование двадцать второй команды

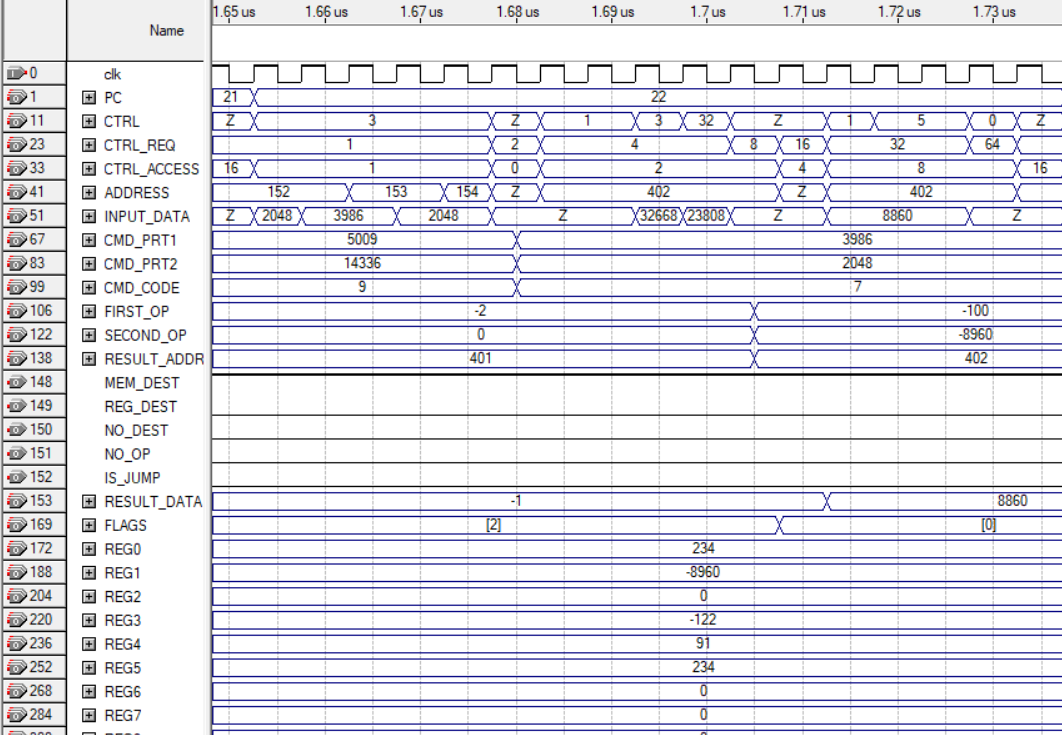


Рисунок 3.26 – Моделирование двадцать третей команды

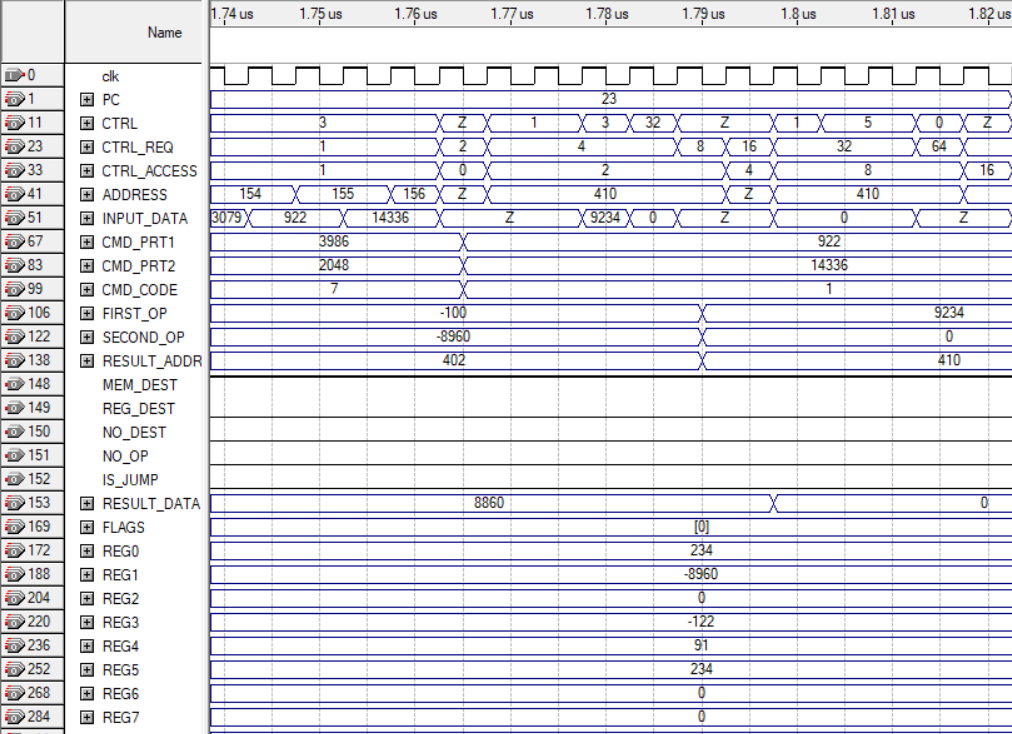


Рисунок 3.27 – Моделирование двадцать четвертой команды

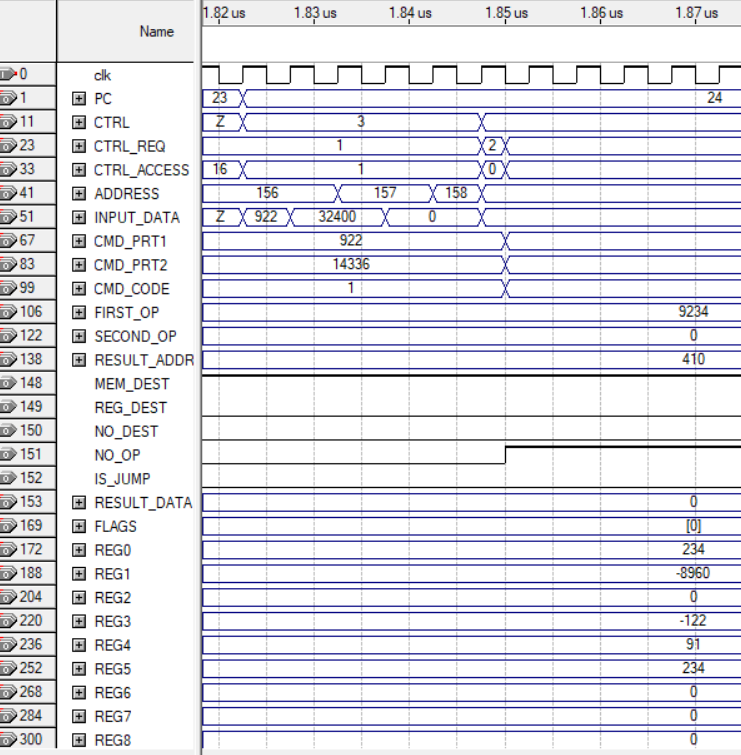


Рисунок 3.28 – Моделирование двадцать пятой команды