|  |  |  |
| --- | --- | --- |
|  |  |  |

ВВЕДЕНИЕ

В соответствии с вариантом необходимо разработать микро-ЭВМ, основанную на принстонской архитектуре (совместное хранения команд и данных в памяти). Шина адреса имеет ширину в 8 бит, шина данных - в 16 бит.

По заданию постоянное запоминающее устройство (ПЗУ) и оперативное запоминающее устройство (ОЗУ) должны иметь асинхронное и синхронное управление соответственно. Однако среда разработки AlteraQuartusII 9.1 не поддерживает асинхронное ПЗУ, поэтому было принято решение сделать оба запоминающих устройства синхронными.

Необходимо реализовать прямую и косвенную адресацию.

Микро-ЭВМ должна включать в себя 14 регистров общего назначения (РОН).

Флаговая отложенная кэш память должна иметь уровень ассоциативности равный 4. Алгоритм замещения строк – наименьшего использования. Поддерживаемые операции должны включать в себя: **HLT, NOP, MOV, JMP, JMZ, PUSH, POP, INCS, NOT, AND.**

Стек должен расти вверх и иметь объем равный 11.

Должен присутствовать децентрализованный параллельный арбитраж шин. Это значит, что необходимо реализовать отдельный блок, который ответственный за раздачу доступа к шинам. Каждое устройство, которому понадобится доступ к шине, будет запрашивать доступ и дожидаться очереди на использование.

Разработку описанного выше устройства будет проводиться с использованием специализированной среды разработки AlteraQuartusII 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, а также инкапсулировать их в отдельные блоки для дальнейшего использования

1. 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.

* 1. Функциональный состав микро-ЭВМ

После анализа варианта задания и построение структурной схемы, мною было принято решение реализовать следующие функциональные блоки:

* Устройство управления (УУ);
* Арифметико-логическое устройство (АЛУ);
* Стек;
* Регистры общего назначения (РОН);
* Оперативное запоминающее устройство (ОЗУ);
* Постоянное запоминающее устройство (ПЗУ);
* Предсказатель переходов;
* Арбитр шин;
* Контроллер прямого доступа к памяти (КПДП)
* Кэш

Далее рассмотрим подробнее каждый из блоков.

Устройство управление выполняет функции управления всеми имеющимися блоками устройства. По своей сути УУ является основным узлом вычислительной системы и имеет наибольшее количество связей с другими блоками и как результат наибольшую сложность.

Арифметико-логическое устройство занимается выполнением конкретной операции над предоставленными операндами. Оно состоит из определенного набора блоков, каждый из которых отвечает за выполнение конкретной операции. Конкретный блок выбирается в зависимости от номера команды.

Стек часто описывают на примере стопки тарелок, в которой для того чтобы добраться до самой первой тарелки, нам нужно убрать все тарелки над ней. В реализации данного устройства стек выполнен на базе регистров и счетчика, который указывает на доступный для записи регистр. Также при переполнении стека новое значение всегда будет записано поверх последнего значения.

Регистры общего назначения предназначены для хранения временных данных (энергозависимы). Внутри блока находятся четырнадцать 16-разрядных регистра.

Кэш представляет собой промежуточный буфер с быстрым доступом, содержащий информацию которая может быть запрошена с наибольшей вероятностью. Доступ к данным осуществляется быстрее, чем выборка данных из более медленной памяти.

Постоянное запоминающее устройство предназначено для хранения массива неизменяемых данных. Оно энергонезависимо.

Контроллер прямого доступа к памяти предоставляет возможность копировать некоторый блок памяти без участия процессора. КПДП переносит данные небольшими частями и имеет более низкий приоритет по сравнению с кэшем, поэтому работа КПДП не может привести к полной блокировке шин.

* 1. Разработка системы команд

При разработке системы команд, для упрощения реализации устройства управления, мною было принято решение использовать команды с фиксированной длиной. Необходимо реализовать 12 операций, поэтому не менее четырех бит будет использоваться для нумерации команды. После анализа необходимого набора команд, стало ясно, что один из операндов во всех случаях будет являться адресом регистра, другой - адресом в области памяти. С учетом того, что шина адреса по заданию - 8 бит, а количество регистров общего назначения - 14, мною был разработан следующий формат команды, представленный в таблице 1.1.

Таблица 1.1 Битовое представление команды

|  |  |  |
| --- | --- | --- |
| Биты [0..3] | Биты [4..7] | Биты [8..15] |
| Номер команды | Адрес регистра | Адрес памяти |

Таким образом битовый размер команды будет равен 16. Учитывая разрядность шины данных (16 бит), одна команда будет вычитываться за 1 такт, что существенно упростит построение ЭВМ.

С учетом разработанной структуры команды опишем все доступные команды в таблице 1.2 (архитектура системы команд) и таблице 1.3 (адресация системы команд).

Таблица 1.2 Архитектура системы команд

|  |  |  |  |
| --- | --- | --- | --- |
| Обозначение | Номер команды | Операнд 1 | Операнд 2 |
| HLT | 0000 | - | - |
| JMP | 0010 | - | addr |
| JMZ | 0011 | - | addr |
| POP | 1000 | reg | - |
| PUSH | 1001 | reg | - |
| MOV(косвен.) | 1010 | reg | addr |
| MOV | 1011 | addr | reg |
| NOT | 1100 | reg | - |
| INCS | 1101 | reg | - |
| SLL | 1110 | reg | - |
| AND | 1111 | reg | addr |

Примечания:

1. addr – прямой адрес операнда (8 бит)
2. reg – адрес регистра общего назначения (4 бита)

Таблица 1.3 Адресация системы команд

|  |  |  |  |
| --- | --- | --- | --- |
| Обозначение | Операнд 1 | Операнд 2 | Адресация |
| HLT | - | - | - |
| JMP | - | addr | прямая |
| JMZ | - | addr | прямая |
| POP | reg | - | прямая регистровая |
| PUSH | reg | - | прямая регистровая |
| MOV | reg | addr | прямая регистровая, косвенная |
| MOV | addr | reg | прямая, прямая регистровая |
| NOT | reg | - | прямая регистровая |
| INCS | reg | - | прямая регистровая |
| SLL | reg | - | прямая регистровая |
| AND | reg | addr | косвенная, прямая регистровая |

Главный преимуществом такой архитектуры команд является возможность загрузки команд за один такт. Однако такой размер команды ограничивает возможность дальнейшего расширения нашей архитектуры: возможность прямой адресации ограничена 8 битами, максимальное количество команд и регистров общего назначения равно 15 (4 бита на тип команды, 4 бита на адрес регистра).

1. РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ

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

* 1. Устройство управления

Блок управления (см. рисунок 3.1) контролирует работу всего устройства.

С него начинается работа микро-ЭВМ и он подает разрешающие работу сигналы всем устройствам процессора.



Рисунок 3.1 – Условно-графическое обозначение блока управления

Для описания входных и выходных сигналов опишем алгоритм работы устройства управления. Для старта работы всего процессора микро-ЭВМ в общем и устройства управления в частном необходимо подать тактирующий сигнал на вход clk. Устройство управления преобразует тактирующий сигнал clk в выходной сигнал \_clk, который в свою очередь тактирует все устройства процессора. Входной сигнал interrupt может прервать тактирование всех устройств (прервать выходной сигнал \_clk). Далее подается сигнал на start1 и адрес команды на PC. Выполнение команды начинается с нулевого адреса, и если не было команд переходов, то инкрементируется после выполнения команды. Выход start1 связан с блоком загрузки и декодирования команды (блок GetAndDecodeCommand). После этого работа устройства управления замораживается до прихода сигнала Ready1 (сигнал об окончании работы устройства загрузки и декодирования команды). Также устройство GetAndDecodeCommand присылает код операции на вход COP и адрес, содержащийся в декодированной команде. Затем выставляется активный сигнал на start2 – разрешающий сигнал для работы блока загрузки операндов. После подтверждения готовности (прихода сигнала Ready2), выставляется сигнал start3 – разрешающий сигнал работы АЛУ. С АЛУ приходит значение флага Z на вход ZF и сигнал об окончании работы Ready3. Start4 разрешает работу блока записи результата. Сигналы NPred и WPred контролируют работу предсказателя, который присутствует в общей схеме, однако ни на что не влияет т.к. мне не удалось реализовать конвейер.

* 1. Блок загрузки и декодирования команды



Рисунок 3.2 – Условно-графическое обозначение блока загрузки и декодирования команды

Данный блок предназначен для загрузки команды из памяти и последующего ее декодирования т.е. получения кода операции, адреса регистра, и адреса памяти.

Ниже опишем входные и выходные сигналы.

Входные:

* clk – тактирующий сигнал;
* address – адрес команды;
* WordIn – непосредственно сама команда (приходит после обращения в кэш);
* work – разрешающий работу сигнал;
* cacheReady – сигнал об окончании работы кэша;

Выходные:

* COP – код операции;
* aReg – адрес регистра (номер регистра общего назначения);
* addr – адрес памяти;
* cacheWrite – имеет постоянное низкое значение, т.к. мы производим чтение из памяти;
* cacheAddress – адрес команды, которую необходимо считать с памяти;
* request – запрос на работу с кэшем;
* ready – сигнал об окончании работы устройства;

После окончания работы блока загрузки и декодирования команды, разобранная команда отправляется в блок загрузки операндов.

* 1. Блок загрузки операндов

Блок загрузки операндов принимает разобранную команду от блока загрузки и декодирования команды. Далее, в зависимости от кода операции, необходимо загрузить операнд или операнды. Так же в данном блоке необходимо было предусмотреть косвенную адресацию т.е. повторную загрузку операнда.

Опишем входные и выходные сигналы данного блока.

Входные:

* clk – тактирующий сигнал;
* FromRON – данные с регистра общего назначения;
* FromCache – данные с памяти;
* aReg – адрес первого операнда – (номер) регистра общего назначения;
* address – адрес второго операнда – адрес в памяти
* work – разрешающий работу сигнал;
* cacheReady – сигнал об окончании работы кэша;

Выходные:

* Op1 – значение первого операнда;
* Op2 – значение второго операнда;
* cacheWrite – имеет постоянное низкое значение, т.к. мы производим чтение из памяти;
* cacheAddress – адрес опернада, который необходимо считать с памяти;
* request – запрос на работу с кэшем;
* ready – сигнал об окончании работы устройства;
* q – номер такта, выводится для отладки и демонстрации правильности работы устройства.

Данный блок спроектирован таким образом, что использует разное количество тактов для разного типа команд. Так при прямой регистровой адресации необходимо 2 такта, а при косвенной – 5.

Данные после загрузки операндов отправляются на блок АЛУ

* 1. Арифметико-логическое устройство

Арифметико-логическое устройство (см. рисунок 2.6) зависит от работы блока извлечения операндов и включается только тогда, когда операнды извлечены. Разрешающий сигнал работы АЛУ поступает после окончание работы блока выборки операндов. В данной реализации блок АЛУ контролирует логические, сдвиговые команды, а так же команды работы со стеком (PUSH, POP).

Для выполнения каждой операции существует отдельный блок. Конкретный блок выбирается при помощи дешифрации номера команды. Выходы дешифратора соединены с соответствующими входами запуска блоков. Пока блок выключен, дешифратор также находится в выключенном состоянии и на все его выходы поступает низкий уровень сигнала. При включении блока выполнения включается и дешифратор. Происходит дешифрация текущей команды и включается необходимый блок выполнения.



Рисунок 3.3 – Условно-графическое обозначение АЛУ

Входные:

* iclk – тактирующий сигнал;
* opcode – код выполняемой операции;
* op1 – значение первого операнда;
* op2 – значение второго операнда;
* stackIn – шина данных для записи в стек (используется, если пришел код операции PUSH);
* CEStart – сигнал запуска блока;

Выходные:

* result – результат работы АЛУ;
* push – сигнал для записи/чтения из стека;
* stackOut – данные полученные из стека (используется, если пришел код операции POP);
* moveWR – высокий уровень сигнала – увеличиваем или уменьшаем значение счетчика стека, низкий уровень сигнала – пишем или читаем из стека (см таблицу 2.1);
* CEReady – высокий уровень при окончании работы АЛУ;
* ZFout – значение флага Z (активный сигнал при равенству нулю результата работы АЛУ);

Ниже приведен список реализованных арифметических, логических, сдвиговых операций и операций работы со стеком:

Таблица 2.1 Cписок операций АЛУ.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Операция | Операнды | Код |
| 1 | POP | 1 | 1000 |
| 2 | PUSH | 1 | 1001 |
| 3 | INCS | 1 | 1101 |
| 4 | NOT | 1 | 1100 |
| 5 | AND | 2 | 1111 |
| 6 | SLL | 1 | 1110 |

Данные с АЛУ могут непосредственно записываться в стек или подаваться на вход блока записи результата.

* 1. Стек

Стек(см. рис. 2.5) – это запоминающий элемент, работающий по принципу LIFO (последний вошел – первый вышел). Стек состоит из одиннадцати 16-разрядных элементов. Направление роста адресов – вверх. Данное направление означает, что при занесении новых данных в стек, значение указателя на вершину стека увеличивается на единицу, при извлечении – уменьшается. При попытке записи в полностью заполненный стек, данные будут занесены в вершину стека (произойдет перетирание данных на вершине стека), при попытке чтения из пустого стека выдается значение самого первого регистра.

Доступны следующие команды:

* PUSH - при выполнении этой команды сначала увеличивается значение указателя на вершину (SP), а затем производится запись в стек по новому указателю;
* POP - при выполнении этой команды сначала извлекаются данные с вершины стека, а затем уменьшается значение указателя на вершину стека на единицу.



Рисунок 2.5 – Условно-графическое обозначение стека

Опишем сигналы блока stack.

Входные:

* clk – тактирующий сигнал;
* wordIn – данные, помещаемые в стек;
* push – высокий уровень сигнала при команде PUSH, низкий – при команде POP;
* Move/WriteRead – высокий уровень сигнала – увеличиваем или уменьшаем значение счетчика стека, низкий уровень сигнала – пишем или читаем из стека (см таблицу 2.2);

Выходные:

* wordOut – выходные данные с вершины стека;
* q – счетчик стека, указывающий на вершину;

Таблица 2.2 – Команды управления стеком

|  |  |  |
| --- | --- | --- |
| Значение Move/WriteRead | Значение push | Действие |
| 0 | 0 | POP |
| 0 | 1 | PUSH |
| 1 | 0 | Уменьшение счетчика стека на 1 |
| 1 | 1 | Увеличение счетчика стека на 1 |

* 1. Блок записи результата



Рисунок 3.5 – Условно-графическое обозначение блока записи результата

Данный блок служит для записи результата работы АЛУ либо в память (кэш), либо в регистры общего назначения.

Опишем входные и выходные сигналы.

Входные:

* clk – тактирующий сигнал;
* COP – код операции
* addrWR – адрес по которому будет осуществлена запись в память (кэш);
* aRegWR – адрес по которому будет осуществлена запись в регистр (РОН);
* data – данные для записи;
* WRstart – разрешающий работу сигнал;
* cacheReady – сигнал об окончании работы кэша;

Выходные:

* write – сигнал на запись в РОН;
* cacheWrite – сигнал на запись в кэш;
* addCache – адрес по которому будет осуществлена запись в память (кэш);
* addrReg – адрес по которому будет осуществлена запись в регистр (РОН);
* dataout – данные для записи;
* request – запрос на работу с кэшем;
* ready – сигнал об окончании работы устройства;
  1. Регистры общего назначения.

Блок регистров общего назначения (см. рисунок 3.6) состоит из четырнадцати 16-разрядных регистров. Эти регистры предназначены для хранения временных результатов т.к. представляют собой более быструю память по сравнению с ОЗУ.



Рисунок 3.6 – Условно-графическое обозначение РОН

Сигналы блока РОН:

Входные:

* clk – тактирующий сигнал;
* register\_number – адрес регистра;
* write – выбор режима чтения или записи (сигнал write активен при записи).
* data – данные для записи в регистр.

Выходные:

* result – данные, прочитанные из устройства

Блок РОН начинает свою работу при подаче на него тактирующего сигнала и корректного номера регистра register\_number. Дальше, в зависимости от сигнала write, будет произведена запись или чтение данных из указанного регистра и их выставление на шину result. Примечательно что все происходит за 1 такт в отличии от блока памяти, где требуется больше времени (тактов).

* 1. КЭШ.

Кэш представляет собой промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Синхронизация с памятью – флаговая отложенная. Алгоритм замещения строк – наименьшего хранения. Количество строк в кэш памяти – 4. Каждая строка разбита на 4 блока, каждый блок содержит 2 слова. Таким образом общий объем кэша 32 слова. Взаимодействует с АЛУ (данные на запись), блоком памяти (чтение/запись) и с арбитром, который сообщает об освобождении шины данных (может быть временно занята КПДП).



Рисунок 3.7 – Условно-графическое обозначение блока кэш памяти

Рассмотрим входные и выходные сигналы.

Входные:

* clk – тактирующий сигнал;
* request – запрос на доступ к КЭШ памяти;
* ProvisionOfBus – сигнал, который сообщает об освобождении шины данных (приходит с арбитра);
* write – выбор режима чтения или записи (сигнал write активен при записи);
* address – адрес по которому необходимо провести чтение или запись;
* inWord –входные данные;
* OnMem – данные для чтения из памяти

Выходные:

* ready – сигнал об окончании работы устройства;
* word – выходное слово данных;
* OnMem – данные для записи в память;
* MemClk – сигнал тактирующий блок памяти;
* MemAddress – адрес для чтения или записи в память;
* BusRequest – запрос на доступ к шине данных
* MemW/R – выбор типа работы с памятью: запись или чтение;
  1. Запоминающее устройство.



Рисунок 3.8 – Условно-графическое обозначение блока памяти

Блок памяти объединяет в себе 2 устройства – ПЗУ и ОЗУ. Как упоминалось ранее, по заданию необходимо реализовать асинхронное ПЗУ, однако САПР QuartusII 9.1 не поддерживает асинхронное ПЗУ, поэтому я решил использовать синхронное ПЗУ на базе стандартного модуля LPM\_ROM (см. рисунок 3.4). ОЗУ (см. рисунок 2.3) реализовано мной на основе модуля LPM\_RAM\_IO.

Опишем сигналы, которые могут подаваться на блок memory.

Входные:

* clk – тактирующий сигнал;
* address – адрес блока памяти;
* RamWrite – запись в ОЗУ при активном сигнале, чтение при пассивном;
* inOutData – данные для записи в ОЗУ;

Выходные:

* inOutData – данные, прочитанные из ОЗУ или ПЗУ.

Данный блок содержит внутри себя проверку старшего бита адреса. При активном старшем бите адреса (8-ой бит) происходит работа (запись или чтение) с ОЗУ, при пассивном – c ПЗУ.

Рассмотрим более детально блоки ОЗУ и ПЗУ.



Рисунок 3.9 – Условно-графическое обозначение блока ПЗУ

Сигналы блока ПЗУ:

* clock – тактирующий сигнал;
* address – адрес блока памяти;
* q – шина данных, выдает прочитанные данные;



Рисунок 3.10 – Условно-графическое обозначение блока ОЗУ

Сигналы блока ОЗУ:

* inclock – тактирующий сигнал;
* address – адрес блока памяти;
* we – сигнал на запись, активный при записи в ОЗУ;
* outenab - сигнал на чтение из ОЗУ, активный при чтении из ОЗУ;
* dio – шина данных, передает прочитанные данные или данные для записи в зависимости от сигналов we и outenab;
  1. КПДП.

Контроллер прямого доступа к памяти (см. рисунок 3.11) реализует передачу информации между внешним хранилищем данных и внутренней памятью.

КПДП позволяет передавать данные небольшими частями в промежутки времени, когда шина не используется другими блоками.



Рисунок 3.11 – Условно-графическое обозначение КПДП

Рассмотрим входные и выходные сигналы.

Входные:

* clk – тактирующий сигнал;
* request – запрос на доступ к КПДП;
* ProvisionOfBus – сигнал, который сообщает об освобождении шины данных (приходит с арбитра);
* write – выбор режима чтения или записи (сигнал write активен при записи);
* startA – выставляется стартовый адрес для чтения или записи в память;
* count – количество слов которые будет прочитано из памяти;
* inDevice – данные которые приходят с внешнего устройства;
* MemWord – данные прочитанные из памяти;

Выходные:

* ready – сигнал об окончании работы устройства;
* MemClk – сигнал тактирующий блок памяти;
* RamAddress – адрес для чтения или записи в память;
* BusRequest – запрос на доступ к шине данных;
* RamWrite – выбор типа работы с памятью: запись или чтение;
* clkRam – тактирующий сигнал для блока памяти;
* deviceWrite – выбор типа работы с внешним устройством: запись или чтение;
* MemWord – данные для записи в память (RAM);
* OnDevice – данные для записи во внешнее устройство;

Для того чтобы КПДП не нарушал работу остальных блоков, он имеет самый низкий приоритет при обращении к памяти.

* 1. Арбитр.

В данной реализации микро-ЭВМ присутствует кэш и КПДП, которые могут работать независимо и одновременно обращаться к памяти, поэтому необходимо устройство, которое предотвратит конфликты и будет выдавать разрешение на работу с шинами строго по приоритету. Данную функцию берет на себя арбитр



Рисунок 3.12 – Условно-графическое обозначение арбитра

Сигналы арбитра:

* Req1 – более приоритетный сигнал на доступ к памяти;
* Req2 – менее приоритетный сигнал на доступ к памяти;
* pred – выходной, разрешающий доступ сигнал.

1. ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ

В этом разделе будет представлены временные диаграммы функционального моделирования разработанных модулей и всей системы в целом.

* 1. Запоминающие устройство.

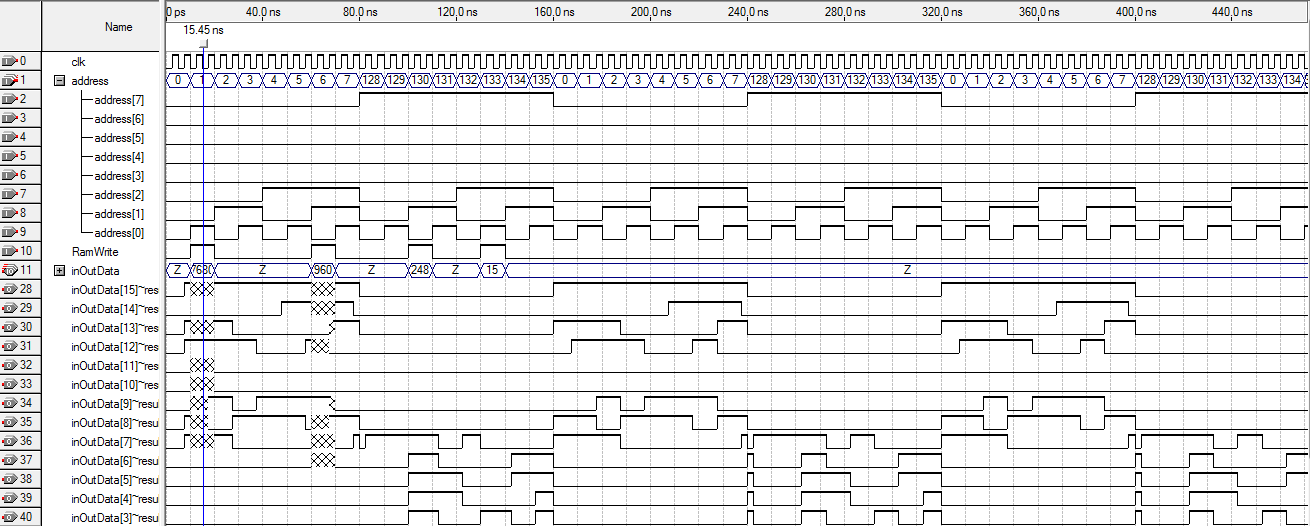


Рисунок 3.1 – Функциональное моделирование блока памяти

На рисунке 3.1 показано обращение с памятью. Можно наблюдать, что при попытке записи в ПЗУ (старший бит адреса 0), запись не происходит. При записи в ОЗУ (старший бит адреса 1) запись происходит успешно(100 ns, 130 ns - запись; 260 ns, 290 ns – чтение записанных данных).

* 1. Устройство управления

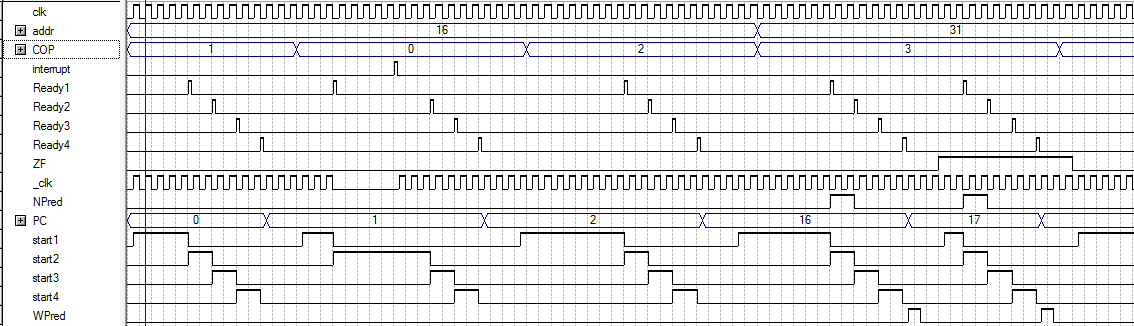
****

Рисунок 3.2 – Функциональное моделирование блока устройства управления

На рисунке 3.2 вручную выставляются сигналы, которые должны приходить от этапов выполнения команды. На нем можно наблюдать, что когда команда с кодом 2 (JMP) завершается, то происходит переключение счетчика команд на указанный адрес ПЗУ. По приходу сигнала *Ready1*, при выполнении команды 3 (JMZ) происходит подача сигнала *NPred* на предсказатель, а по окончанию исполнения команды происходит подача сигнала *WPred* на предсказатель, и в зависимости от значения флага Z происходит (Z = 1) или нет (Z = 0) условный переход.

* 1. Блок декодирования команды.

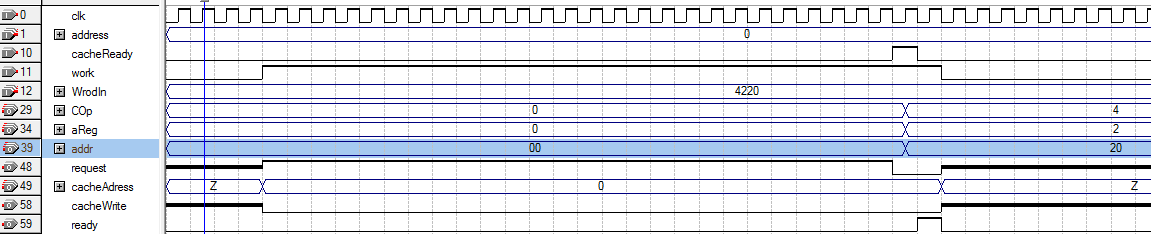


Рисунок 3.3 – Функциональное моделирование блока декодирования команды

На рисунке 3.3 видно, что происходит обращение в кэш за командой, и по приходу сигнала готовности от кэш-памяти происходит декодирование команды на код операции, адрес регистра и адрес в памяти (ОЗУ).

* 1. Блок исполнения команды.

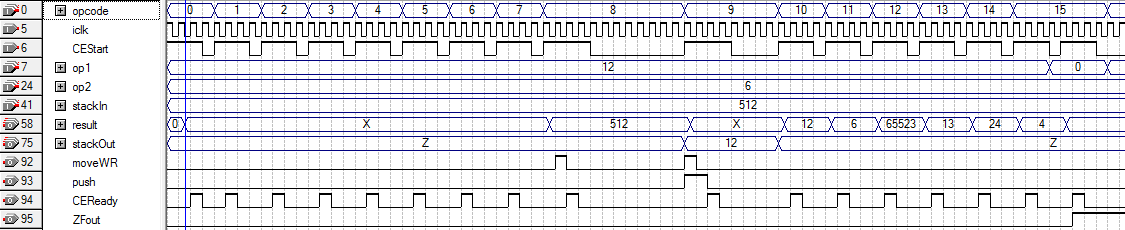


Рисунок 3.4 – Функциональное моделирование блока исполнения команды

На рисунке 3.4 перебираются возможные команды, которые могут поступить на этап исполнения команд. Команды 8 (POP) и 9 (PUSH) выполняются дольше остальных, так как требуются такты на обращение со стеком. При получении нулевого результата в арифметических операциях (команды 12-15) происходит установка флага *ZFout* в единичное значение.

* 1. Блок выборки операндов**.**

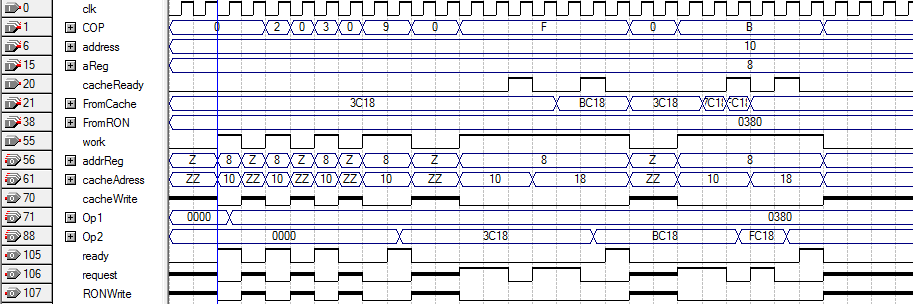


Рисунок 3.5 – Функциональное моделирование блока выборки операндов

На рисунке 3.5 видно, что при различных командах требуется различное время выполнение. При команде POP сигнал готовности выдается сразу, так как не требуется выборка операндов. При команде AND происходит двойное обращение в кэш, так как используется косвенная адресация.

* 1. Блок записи результата**.**

На рисунке 3.6 показаны ситуации записи в основную память и в регистр. В случае записи в память ожидается готовность выполнения записи, выдавая запрос в кэш. Запись в регистр происходит за один такт. Если команда не требует записи, то сигнал готовности выдается сразу.

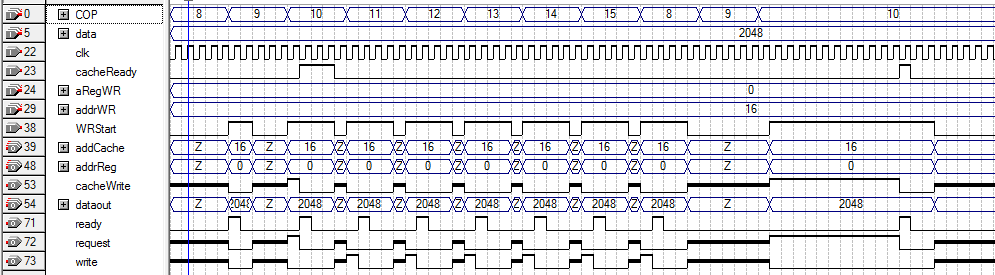


Рисунок 3.6 Функциональное моделирование блока записи результата

* 1. Блок регистров.

На рисунке 3.7 видно, что при единичном сигнале *write* происходит запись в регистр, в последующем там уже находится записанное значение. Запись происходит по фронту тактирующего сигнала.

* 1. Стек

На рисунке 3.8 видно, что при единичном сигнале *Move/WriteRead* происходит изменение вверх или вниз SP, в зависимости от сигнала *push*. Когда стек пуст или полон изменение SP не происходит, а когда *push* в единице происходит перезапись элемента на вершине стека.

* 1. Кэш-память процессора**.**

Чтобы показать выгрузку в память и совместную работу с КПДП было произведено совместное моделирование этих блоков на рисунке 3.9. При промахе возникающем в кэш, происходит загрузка данных из памяти, если блок данных в кэш был пустой (60 -110 ns). Потом происходит запись данных, если установлен сигнал записи *write* (110-120 ns). Далее выдается сигнал готовности. Если в замещаемом блоке были изменения, то этот блок сначала записывается в память (550-620 ns), только потом загружается новая порция данных из основной памяти (640-670 ns). Состояния ОЗУ до модуляции и после представлены ниже.

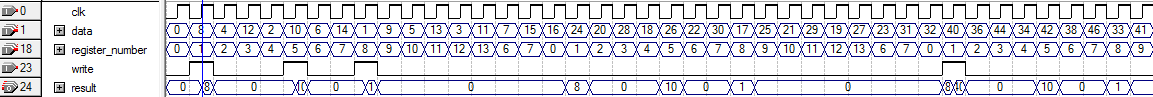


Рисунок 3.7 Функциональное моделирование блока регистров общего назначения

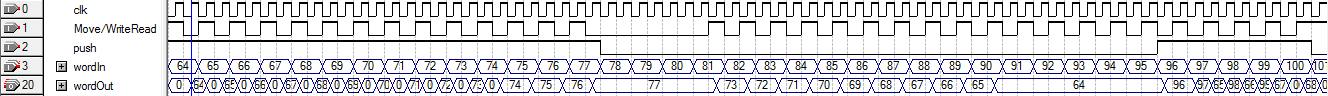


Рисунок 3.8 Функциональное моделирование блока стека

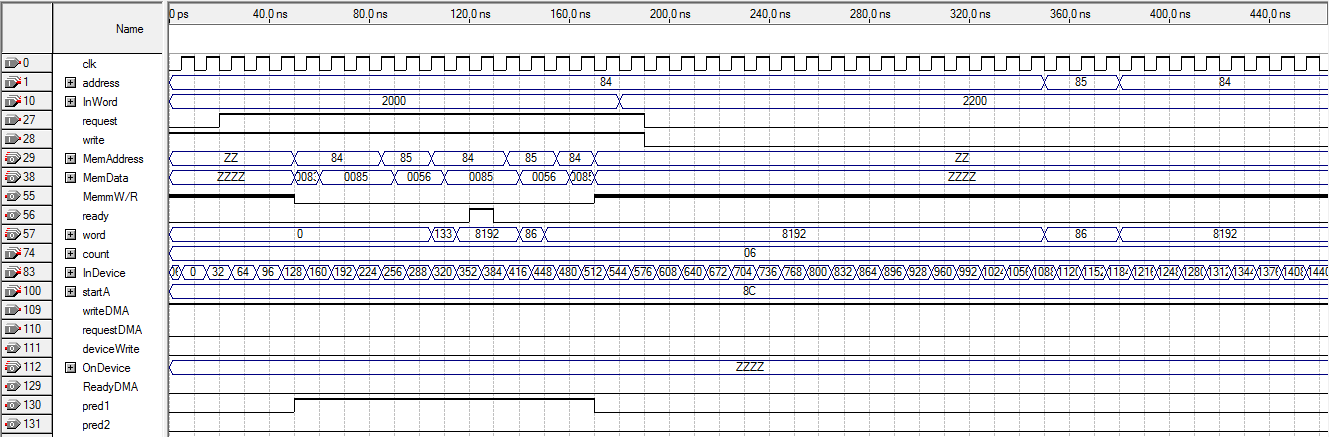


Рисунок 3.9 Совместное функциональное моделирование блока *кэш* (часть 1)

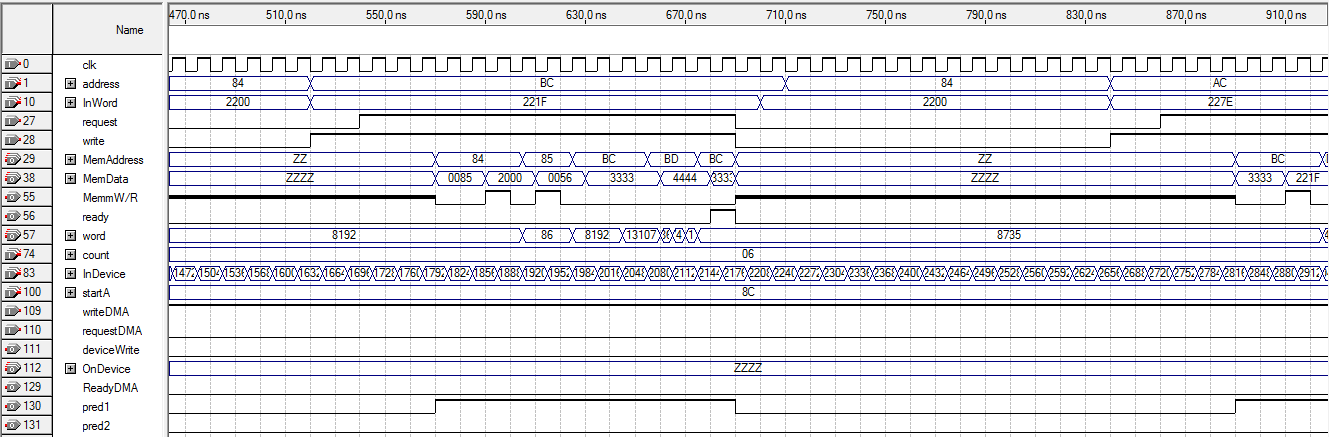


Рисунок 3.9 Совместное функциональное моделирование блока *кэш* (часть 2)

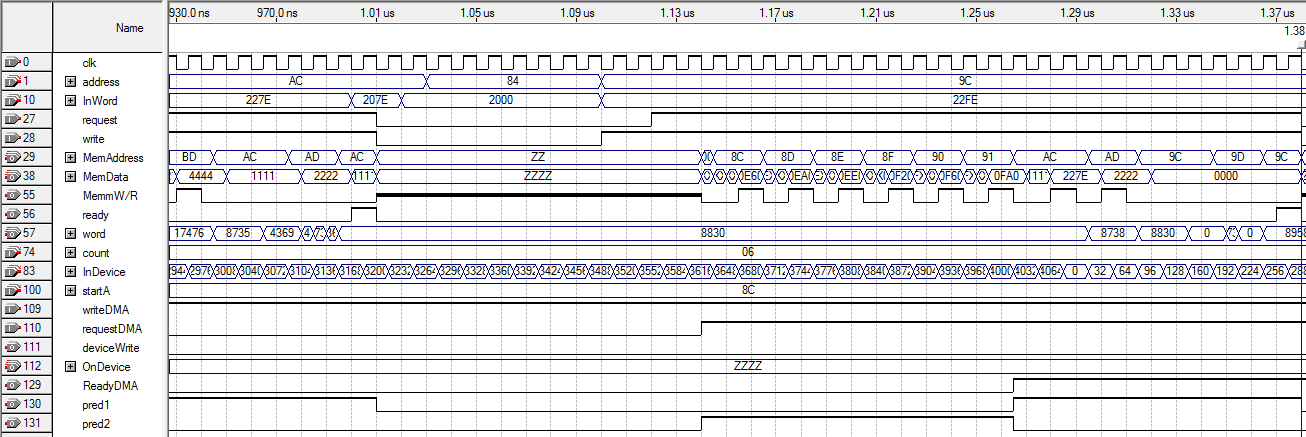
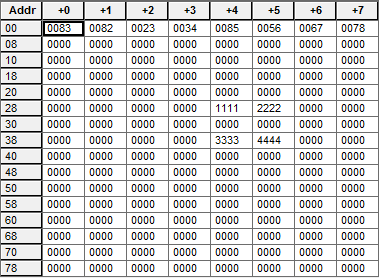


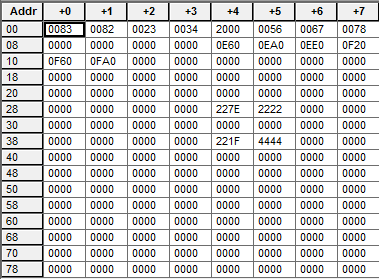
Рисунок 3.9 Совместное функциональное моделирование блока *кэш* (часть 3)

Состояния RAM.

До моделирования:



После моделирования:

****

* 1. Система предсказания переходов**.**

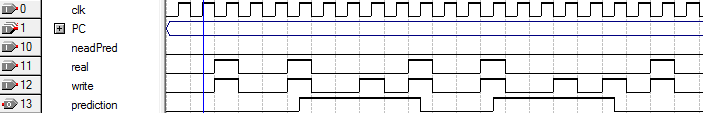


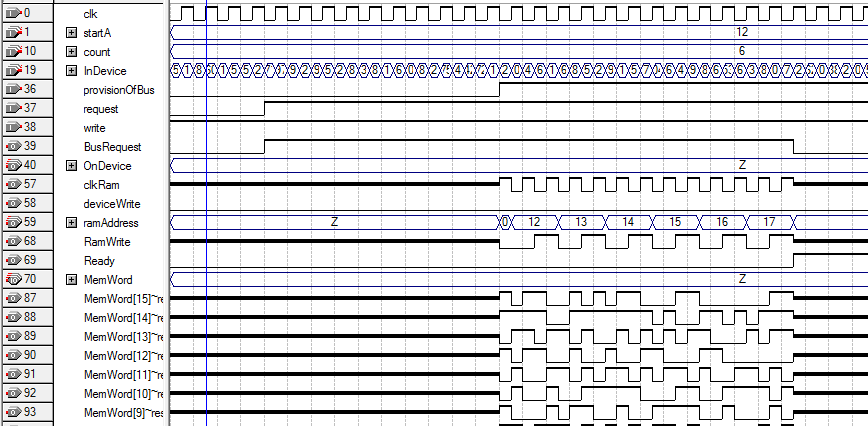
Рис. 3.10 Функциональное моделирование блока предсказания переходов

На рисунке 3.10 представлено функциональное моделирование блока предсказания переходов. На старте автомат в состоянии 00 и выдает предсказания, что перехода нет, далее основываясь на сигнале *real* и выходе *prediction* можно придти к выводу, что автомат работает правильно. Переходы по моделированию:

00->01->11->10->01->11->10->00->01

Предсказание делается на основе значения старшего бита. Для тестирования одного автомата смены шаблона не производится (сигнал *needPred* = 00).

* 1. КПДП**.**



Рисунке 3.11 Функциональное моделирование блока КПДП

На рисунке 3.11 происходит запрос к КПДП на запись данных из ОЗУ. Пока не придет разрешение от арбитра на работу с основной памятью, КПДП не начинает свою работу. После прихода сигнала на разрешения работы с шиной (*provisionOfBus*) КПДП последовательно записывает 6 слов начиная с 12 адреса, значения для стартового адреса и количество слов берется с входных шин *startA* и *count* соответственно.

* 1. Система арбитража шин**.**

На рисунке 3.12 представлены все случае, которые могут быть представлены в арбитре. Запрос от устройства, которому принадлежит арбитр, приходит в качестве сигнала *Req1*. Приоритет при одновременно приходе сигналов *Req1* и *Req2* зависит от внешнего подключения арбитров. Пока одно устройство не закончит свою работу второму сигнал разрешения на работу с шиной выдан не будет.

E:\Woker\BSUIR\4 курс\2 семестр\курсач\ars_17\arbitr.PNG

Рисунок 3.12 Функциональное моделирование блока арбитра

* 1. Моделирование всей системы

В качестве тестовой программы для демонстрации работы микро-ЭВМ использовался следующий набор команд:

|  |  |  |
| --- | --- | --- |
| Символьный вид | Двоичное представление | Шестнадцатеричный вид |
| mov #1, [$81] | 1011 0001 10000001 | b181 |
| mov #2, [$80] | 1011 0010 10000000 | b280 |
| push #1 | 1001 0001 00000000 | 9100 |
| pop #3 | 1000 0011 00000000 | 8300 |
| not #3 | 1100 0011 00000000 | c300 |
| incs #2 | 1101 0010 00000000 | d200 |
| sll #1 | 1110 0001 00000000 | e100 |
| mov $83, #1 | 1010 0001 10000011 | a183 |
| and #1, [$84] | 1111 0001 10000100 | e184 |
| jmp $10 | 0010 0000 00010000 | 2010 |
| … до 0x10 адреса все заполнено 0x0000 | 0000 0000 00000000 | 0000 |
| push #3 | 1001 0011 00000000 | 9300 |
| pop #5 | 1000 0101 00000000 | 8500 |
| jnz $1 | 0011 0001 00000001 | 3101 |
| pop #3 | 1000 0011 00000000 | 8300 |
| hlt | 0000 0000 00000000 | 0000 |

Адреса в памяти имеют условное обозначение $X, а номера регистров – #X, косвенная адресация – [$X]. Такая последовательность команд содержит в себе все поддерживаемые команды микро-ЭВМ.

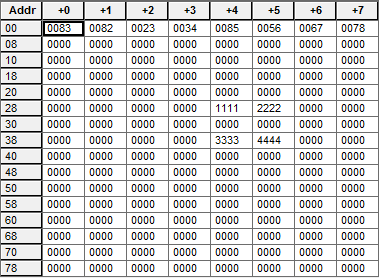


Рисунок 3.13 Первоначальное состояние ОЗУ

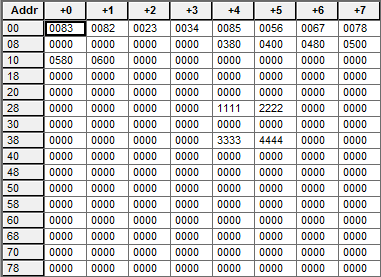


Рисунок 3.14 Состояние ОЗУ после симуляции

В процессе работы микро-ЭВМ (рис. 3.15) КПДП совершает свою транзакцию (140ns - 250 ns). Для этого, когда к нему приходит запрос (60ns) он ожидает разрешения от арбитра, и после получения разрешения (130ns) начинает свою работу, совершая полную транзакцию, 6 слов.

Результат работы КПДП можно увидеть из сравнения первоначального (рис. 3.13) и результирующего (рис. 3.14) состояния памяти ОЗУ.

Программа записанная на ПЗУ представлена на рисунке 3.14. Для проверки команд перехода она была разделена по памяти таким образом, чтобы продолжалась с точки перехода.

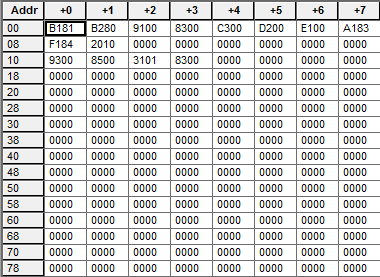


Рисунок 3.14 Содержимое ПЗУ

Изначально происходит считывание команды по нулевому адресу ПЗУ и ее декодирование(60-130ns). После этого происходит считывание операндов для реализации команды (260-460 ns), учитывая, что для команды MOV reg, addr используется косвенная адресация, то обращение в память происходит дважды. Далее происходит выполнение команды (транслирование операнда на шину результата). После чего происходит запись результата (490-510 ns). На следующем такте происходит изменения значения счетчика регистров. Работа с предсказателем и возвращение на старт цикла. Так как кэш хранит в блоке 2 слова, то обращение в основную память происходит каждые 2 слова (четное значение счетчика команд). Наблюдая изменение значения шины *COp* можно судить о правильности и последовательности считывания команд из ПЗУ. Шина *start* объединяет в себе стартовые сигналы для этапов выполнения команды: 8 - выборка и декодирование команды, 4 - выборка операндов, 2 - выполнение команды, 1- запись результата. Наблюдая шину *result* в момены *start* = 1 можно увидеть результаты выполнения очередной команды. Наблюдая шины *Op1* и *Op2* в моменты *start* = 2 можно увидеть результаты выборки операндов. При выполнении второй команды (550-750 ns) обращения в основную память не происходит, так как все данные уже находятся в кэш-памяти. При этом видно(650 ns) что реализуется косвенная адресация для операнда. При выполнении команды JMP (2.36- 2.48 us) можно наблюдать, что считчик команд сменил свое значения с 9(шестнадцатеричное) на 10(шестнадцатеричное), что свидетельствует об успешном выполнении команды. После команды JMZ (2.87 - 3.07 us) перехода не произошло, так как *ZFout* не был установлен в 1. При дешифрации команды HLT (3.36us) процессор останавливает свою работу до прихода внешнего прерывания(3.45us). Так как код команды HLT равен 0 и после программы идут нулевые ячейки, то после прихода прерывания и дешифрации очередной команды происходит очередная остановка процессора.

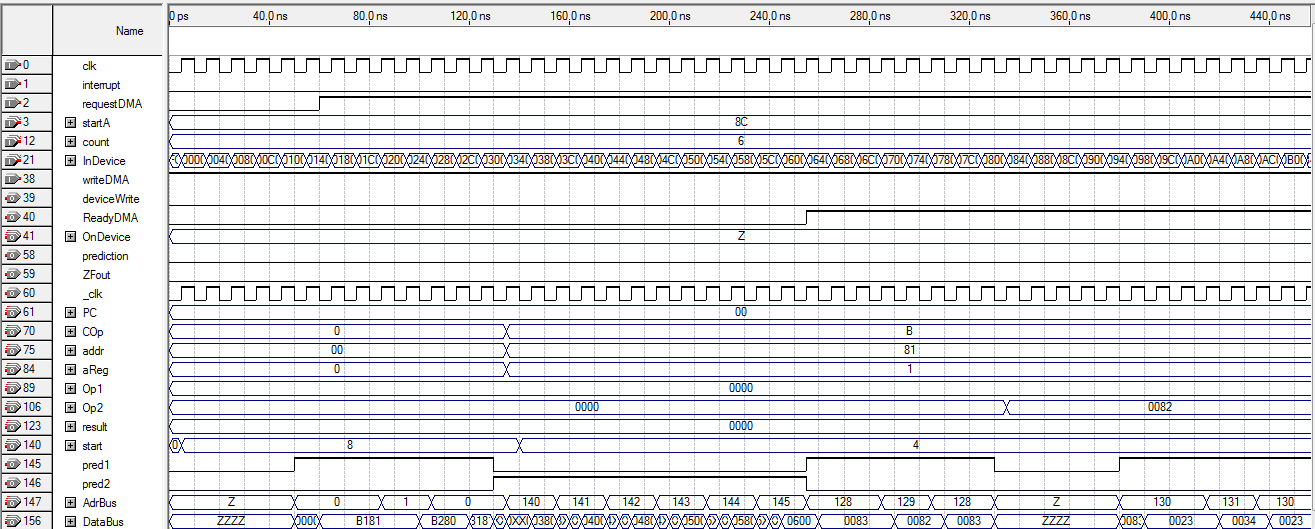


Рис. 3.15 Временная диаграмма работы микро-ЭВМ (часть 1)

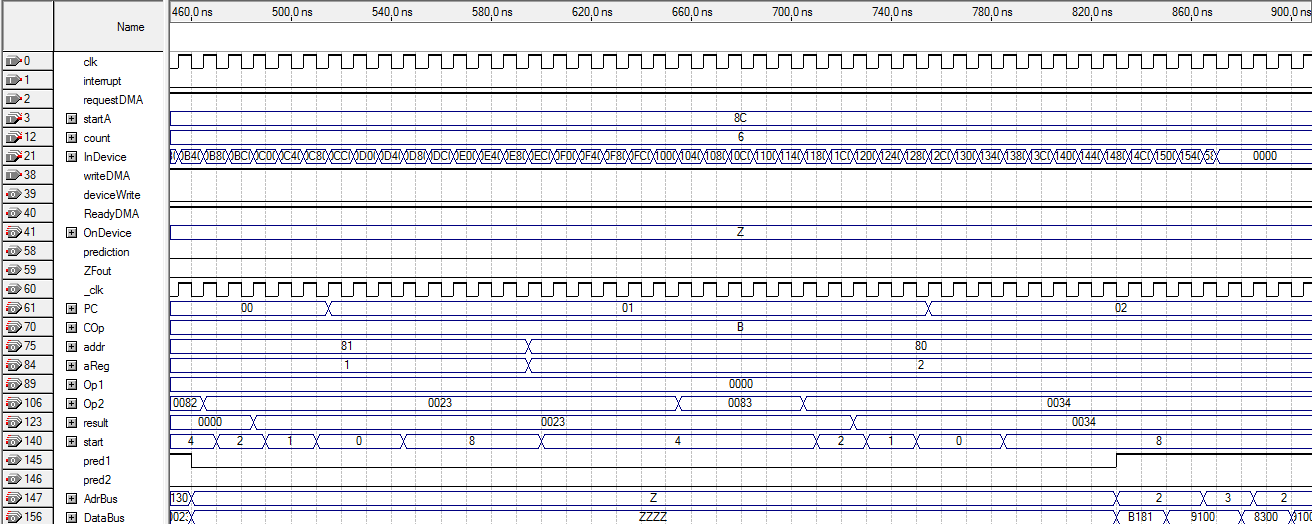


Рис.3.15 Временная диаграмма работы микро-ЭВМ (часть 2)

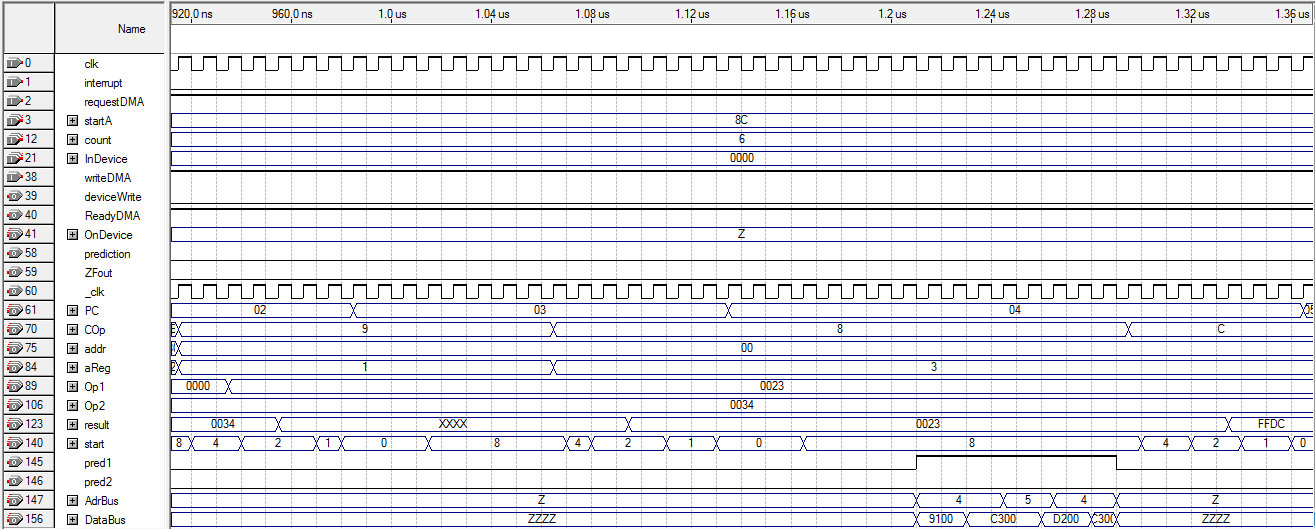


Рис. 3.15 Временная диаграмма работы микро-ЭВМ (часть 3)

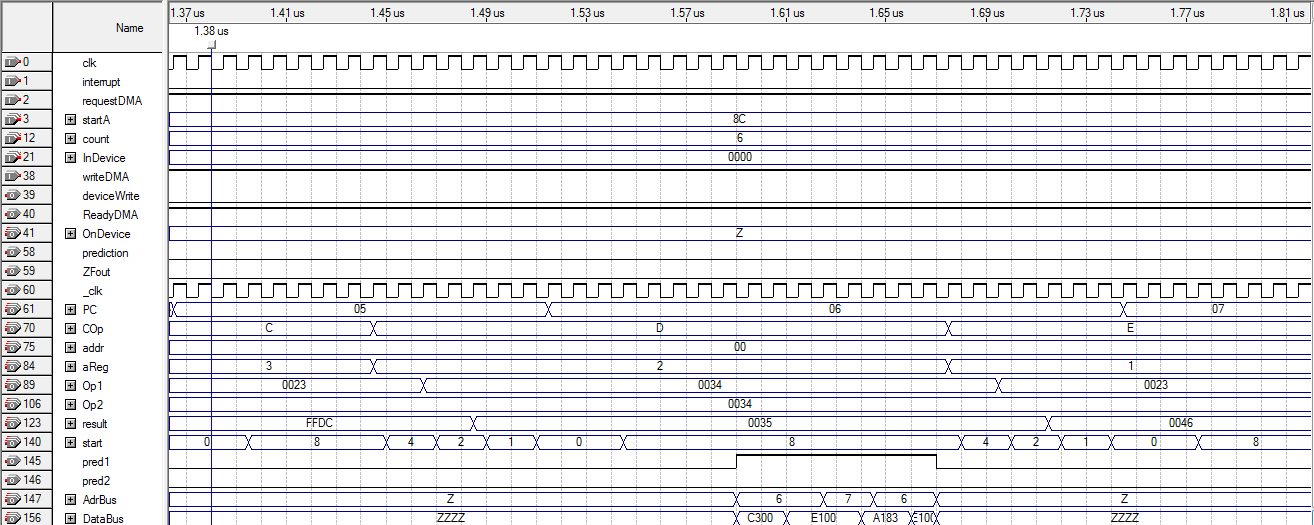


Рис. 3.15 Временная диаграмма работы микро-ЭВМ (часть 4)

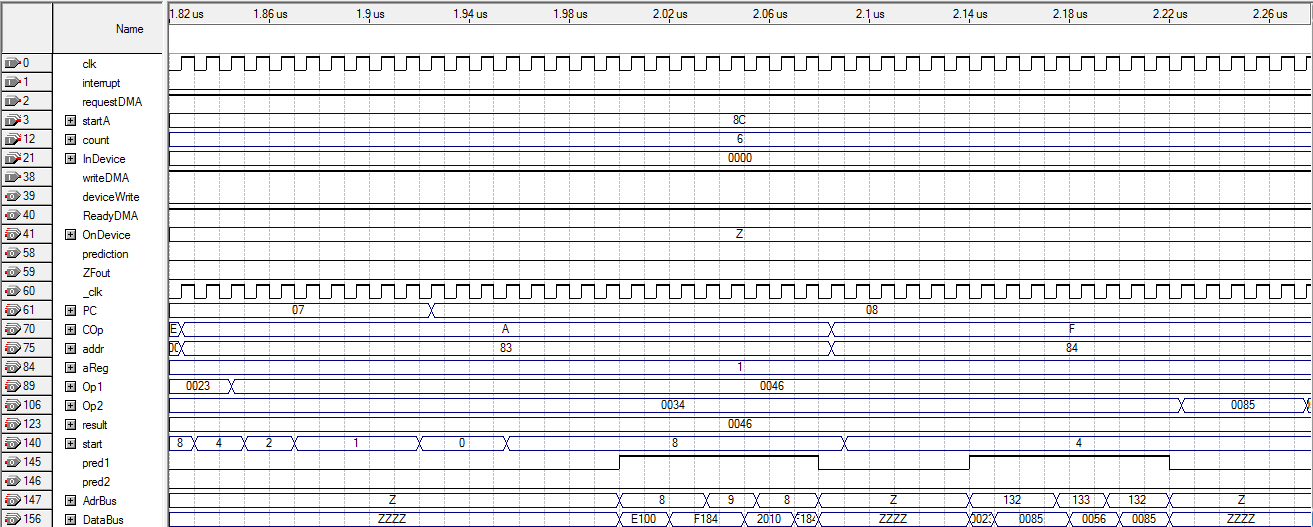


Рис. 3.15 Временная диаграмма работы микро-ЭВМ (часть 5)

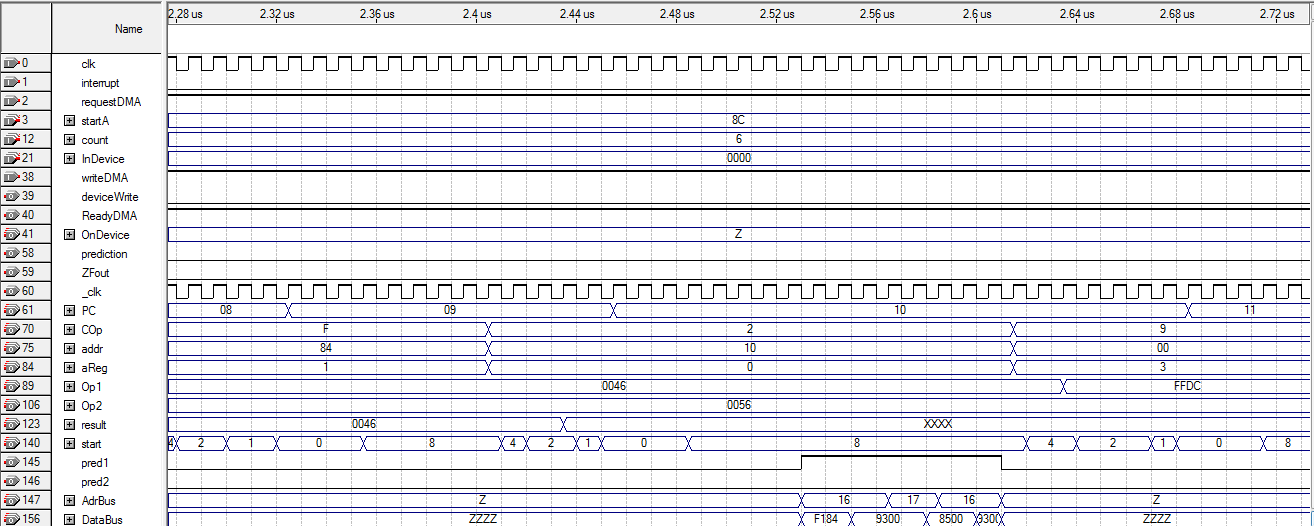


Рис. 3.15 Временная диаграмма работы микро-ЭВМ (часть 6)

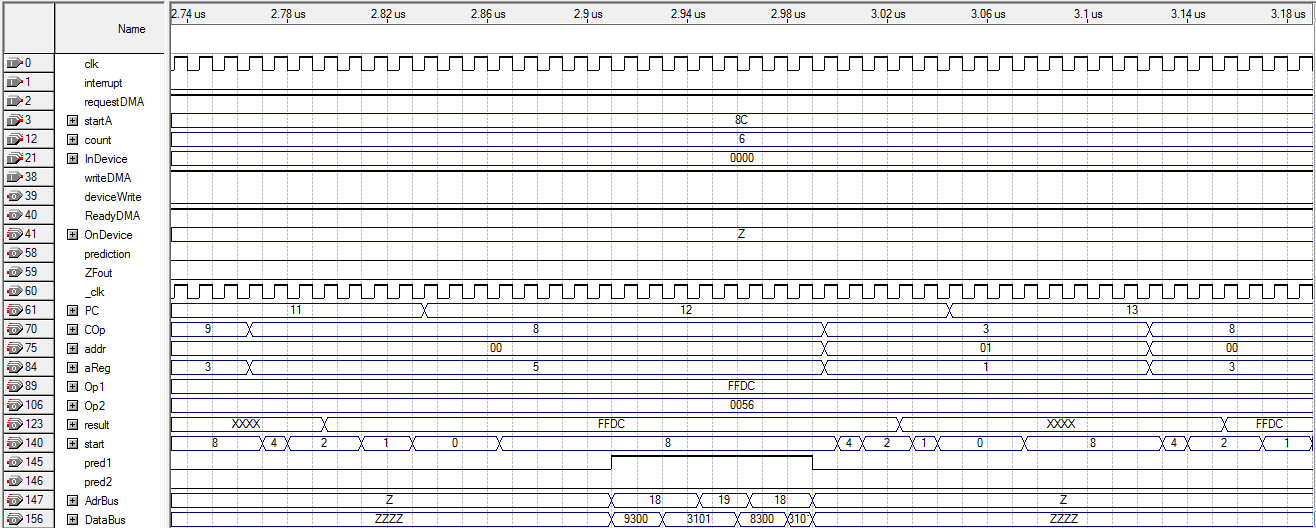


Рис. 3.15 Временная диаграмма работы микро-ЭВМ (часть 7)

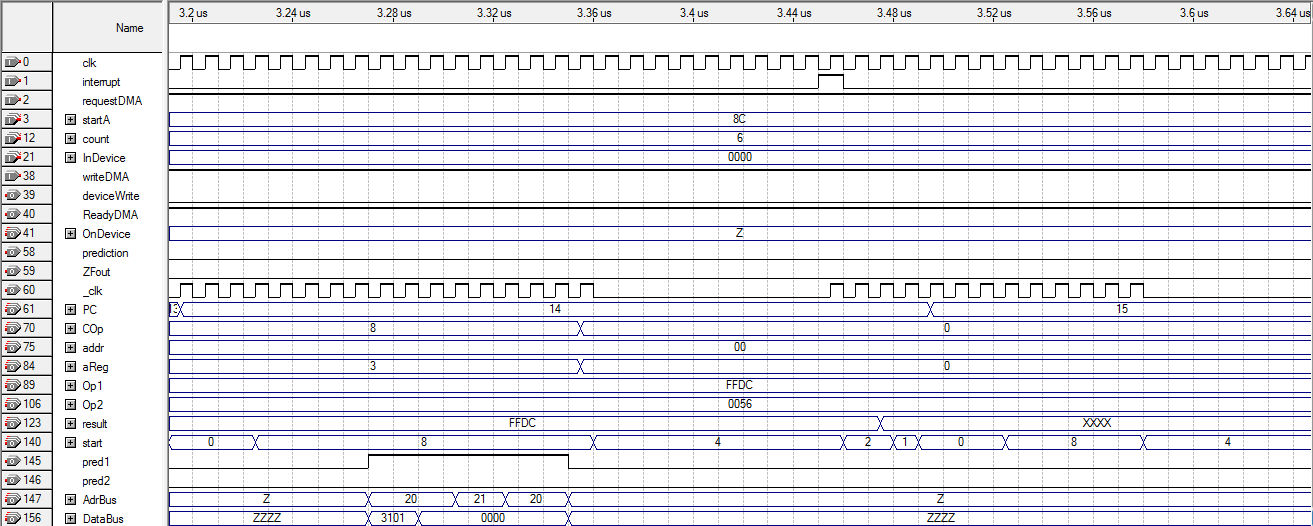


Рис. 3.16 Временная диаграмма работы микро-ЭВМ (часть 8)

ЗАКЛЮЧЕНИЕ

В ходе работы удалось реализовать простейшую микро-ЭВМ, выполняющую базовые команды. При разработке возникало множество конфликтов и задач, решаемых с трудом. Конечно, реализованная микро-ЭВМ не является выдающейся и полной, отсутствуют некоторые блоки, указанные в задании. Однако считаю, что цель обучения курсу СиФО ЭВМ была достигнута.