ВВЕДЕНИЕ

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

Постоянное запоминающее устройство (ПЗУ) и оперативное запоминающее устройство (ОЗУ) должны иметь синхронное и асинхронное управление соответственно. Однако используемое ПО Altera Quartus II 9.1 не поддерживает асинхронное ОЗУ, поэтому мною было принято решение делать оба запоминающих устройства синхронными.

Устройство должно иметь возможность работать с прямой и косвенной адресацией.

Микро-ЭВМ должно включать в себя 16 регистра общего назначения (РОН).

Сквозная кэш память с отображением должна иметь уровень ассоциативности равный 4. Алгоритм замещения строк – наиболее давнего хранения.

Поддерживаемые операции должны включать в себя: **HLT, NOP, MOV, JMP, JMS, PUSH, POP, SUB, NAND, NOR, ROL, ROR**

Стек должен расти вверх и вмещать 6 слов.

Микро-ЭВМ должен включать в себя децентрализованный арбитр.

Предсказатель переходов должен работать по схеме А3 и использовать 2 бита указателя инструкций и 2 бита из регистр глобальной истории.

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

1. РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ

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

* 1. Функциональный состав микро-ЭВМ.

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

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

Рассмотрим подробнее каждый из блоков.

Устройство управление выполняет функции управления всеми имеющимися блоками устройства. Если сравнивать его с человеческим телом, то можно назвать его “мозгом” все системы, который управляет своеобразными “органами”. Соответственно как и человеческий мозг данный блок является самым сложными в реализации.

Арифметическо-логическое устройство выполняет необходимые операции над предоставленными ей операндами. Проводя аналогию с человеческим телом дальше можно назвать это устройством своеобразным “разумом” всей системы, который обрабатывает предоставленную ему информацию. По своей структуре устройство состоит из множества различных блоков, каждый из которых активируется при пода на вход соответствующего номера необходимой команды.

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

Регистры общего назначения обычно используются для хранения каких-либо временных данных, так как они энергозависимы. Продолжая проводить аналогию с человеческим телом можно сказать что они представляют собой “временную память” человека. Блок регистров представляет собой 16 регистров разрядностью 16 бит каждый.

Оперативное запоминающее устройство предназначено для хранения долговременной информации. Проводя аналогию можно смело утверждать что это “память человека”. Благодаря гарвардской архитектуре данному устройству необходимо 2 ОЗУ: для команд и для данных. Каждое из них вмещает 256 слов по 16 бит. Учитывая что каждая операция имеет длины в 48 бит, то этого достаточно для хранения 85 операций. К сожалению объем памяти нельзя увеличить в связи с ограничением на длину шины адреса. Примечательно что блоки ОЗУ и ПЗУ объединены в один логический блок с возможностью выбора необходимого блока.

* 1. Разработка системы команд.

При разработке системы команд для упрощения реализации устройства управления было принято решение использовать команды с фиксированной длиной. Учитывая разрядность шины данных (16 бит) и адреса (8 бит) один операнд будет занимать не менее 16 бит. Согласно требованиям необходимо обеспечить наличие операций с двумя операндами, таким образом команда должна как минимум содержать код операции, и адреса двух операндов. Также требуется реализовать 2 различных вида адресации, что накладывает дополнительные накладные расходы. Также необходимо обеспечить необходимый запас зарезервированных бит, на случай внесение дополнительных изменений в архитектуру. Таким образом был разработан следующий формат команды, представленный.

Таблица 1.1 – Битовое представление команды

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Биты [39..36] | Биты [39..34] | Биты [33..32] | Биты [31..16] | Биты [15..0] |
| Зарезервированные | Номер команды | Номер адресации | Операнд 1 | Операнд 2 |

Поставим каждому значению номера адресации соответствующее название в таблице 1.2:

Таблица 1.2 – Номера адресации

|  |  |
| --- | --- |
| Значение | Название |
| 00 | Прямая |
| 01 | Косвенная |
| 10 | Двойная косвенная |
| 11 | Прямая |

В таблице 1.2 можно заметить присутствие “двойной косвенной” адресации, которая нигде ранее не упомяналась. Данный тип адресации будет оговорен в дальнейшем при описании устройства управления.

Опишем архитектуру системы комманд в таблице 1.3:

Таблица 1.3 – Архитектура системы команд

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Комманды: | Количество операндов | Bin |
| 1 | HLT | 0 | 000001 |
| 2 | NOP | 0 | 000010 |
| 3 | MOV adr, reg | 2 | 000011 |
| 4 | MOV reg, adr | 2 | 000100 |
| 5 | JMP adr | 1 | 000101 |
| 6 | JMS adr | 1 | 000110 |
| 7 | PUSH adr | 1 | 000111 |
| 8 | POP adr | 1 | 001000 |
| 9 | SUB reg1, reg2 | 2 | 001001 |
| 10 | NAND reg1, reg2 | 1 | 001010 |
| 11 | NOR reg1, reg2 | 1 | 001011 |
| 12 | ROL reg | 1 | 001100 |
| 13 | ROR reg | 1 | 001101 |

Опишем очевидные преимущества и недостатки разработанной нами системы команд.

Преимущества:

* Простота чтения;
* Расширяемость;
* Не требуется устройство декодирования команды.

Недостатки:

* Значительный объем;
* Избыточность.
  1. Описание взаимодействия всех блоков микро-ЭВМ при выполнении команд программы.

Для того чтобы устройство начало работать необходимо два условия:

1. Подача тактирующего сигнала на вход clk;
2. П-образный сигнал на входе start.

С подачей П-образного сигнала на вход происходит очистка всех узлов управляющего блока от результатов прошлой работы. Это необходимо для корректной работы устройства

Дальнейшую работу устройства можно разбить на 4 этапа обработки. Рассмотрим подробнее каждый из этапов:

1. В работу данного этапа задействован блок ОЗУ и УУ. Данный этап предполагает собой загрузку команды из соответствующей ячейки памяти ОЗУ. Так как шина данных ограничена, то весь процесс происходит в 3 стадии, результат каждой из которых помещается в соответствующий регистр IR. Для вычислении на каждой стадии адреса блока памяти для загрузки в УУ имеется специальный регистр ADDR2. Eго значение суммируется со значением регистра IP и выставляет на шину адреса, после чего происходит загрузка. По окончанию стадии мы имеем в регистре IR1 – номер команды и тип адресации, а в регистрах IR2 и IR3 – адреса операндов;
2. В работу данного этапа вовлечен блок УУ, а также могут быть вовлечены блоки ОЗУ и РОН. В ходе работы данного этапа в регистры OP1 и OP2 блока УУ в соответствии с типом адресации загружаются необходимые в дальнейшем адреса операндов. Данный этап необходим из-за наличия различных типов адресации, так как каждый из них предполагает свою последовательность необходимых сигналов;
3. При работе данного этапа теоретически могут быть задействованы все блоки микро-ЭВМ. Этот этап представляет собой выполнение команды в соответствии с ее номером, а также сохранение результата операции в ПЗУ или в РОН;
4. Работа данного этапа требует только работы одного блока – УУ. Этот этап переводит наш указатель инструкций на следующую инструкцию для дальнейшего ее выполнения. Примечательно что в случае команд условного перехода этот этап не выполняется, а сразу же происходит выполнения этапа 1.

Каждый вышеописанный этап по завершению работы представляет собой

отдельный логический.

Каждый вышеописанный этап представляет собой отдельный логический блок, в котором записана последовательность сигналов выставляемых на шину управления. Шина управления имеет длину 64 бита, при этом последние 16 бит доступны снаружи, а первые 48 бит являются внутренней шиной устройства управления. В таблице 1.4 приведен список сигналов, которые могут быть выставлены на шину управления:

Таблица 1.4 – Сигналы шины управления

|  |  |  |
| --- | --- | --- |
| Номер | Внешний номер | Комманда |
| 1 | 2 | 3 |
| 0 | - | IP\_WRITE |
| 1 | - | IR1\_ENABLE |
| 2 | - | IR1\_RW |
| 3 | - | IR2\_ENABLE |
| 4 | - | IR2\_RW |
| 5 | - | IR3\_ENABLE |
| 6 | - | IR3\_RW |
| 7 | - | AR\_ENABLE\_ADDR |
| 8 | - | AR\_ENABLE\_DATA |
| 9 | - | AR\_RW |
| 10 | - | DR\_ENABLE |
| 11 | - | DR\_RW |
| 12 | - | SF\_SET |
| 13 | - | SF\_CLEAR |
| 14 | - | ZF\_SET |
| 15 | - | ZF\_CLEAR |
| 16 | - | JF\_SET |
| 17 | - | JF\_CLEAR |
| 18 | - | DB\_IN |
| 19 | - | DB\_OUT |
| 20 | - | AB\_OUT |
| 21 | - | ADD2\_IN |
| 22 | - | ADD2\_DATA\_OR\_BUS |
| 23 | - | ADD\_OUT |
| 24 | - | ADD\_PLUS\_MINUS |
| 25 | - | STEP\_FINISHED |
| 26 | - | OPERATION\_FINISHED |
| 27 | - | OP1\_ENABLE |
| 28 | - | OP1\_RW |
| 29 | - | OP2\_ENABLE |
| 30 | - | OP2\_RW |
| 31 | - | ADD\_CLR |
| 32 | - | FLAG\_RESET |
| 33 | - | HLT |
| 34-47 | - | - |

Таблица 1.4 – Сигналы шины управления (продолжение)

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 48 | 0 | REG\_ENABLE |
| 49 | 1 | REG\_RW |
| 50 | 2 | MEM\_ENABLE |
| 51 | 3 | MEM\_ROM\_RAM |
| 52 | 4 | MEM\_RW |
| 53 | 5 | STACK\_ENABLE |
| 54 | 6 | STACK\_PUSH |
| 55 | 7 | STACK\_POP |
| 56 | 8 | ALU1\_IN |
| 57 | 9 | ALU2\_IN |
| 58 | 10 | ALU\_OPER0 |
| 59 | 11 | ALU\_OPER1 |
| 60 | 12 | ALU\_OPER2 |
| 61 | 13 | SF\_SET |
| 62 | 14 | ALU\_OUT |
| 63 | 15 | ADDR\_MEM\_ENABLE |

Таким образом путем выставления соотвествующиx последовательностей команд происходит взаимодействие всех блоков.

2 РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ

В текущем разделе подробно описываются и обосновываются решения принятые мною при проектировании всех устройств микро-ЭВМ. Также здесь затрагиваются принцип работы и характеристика основных устройств.

* 1. Запоминающие устройства. Функциональный состав ПЗУ и ОЗУ.

ПЗУ (см. рисунок 2.1) реализовано мной на основе стандартного модуля Altera Quartus II 9.1 – LPM\_ROM. Этот модуль поддерживает синхронный режим работы и полностью подходит для интеграции в микро-ЭВМ.

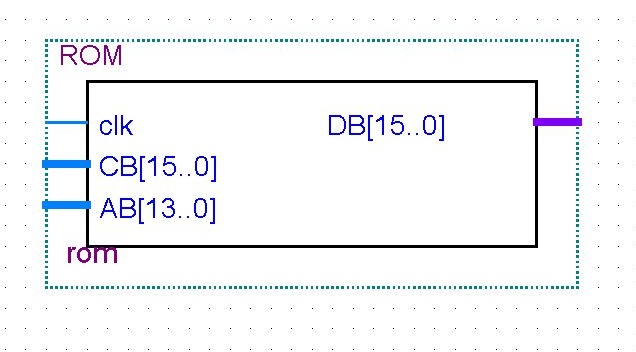


Рисунок 2.1 – Условно-графическое обозначение блока ПЗУ

Здесь и далее я решил использовать следующие обозначения сигналов в списках:

* Входной сигнал.
* Выходной сигнал.
* Двунаправленный сигнал.

Сигналы блока ПЗУ:

* *clk* – тактирующий сигнал.
* *CB* – контрольная шина.
* *AB* – адресная шина, передает адрес слова для чтения.
* *DB* – шина данных, передает прочитанные данные.

Как упоминалось ранее, по заданию необходимо реализовать асинхронное ОЗУ, однако САПР Quartus II не поддерживает асинхронное ОЗУ, поэтому я решил использовать синхронное ОЗУ на базе стандартного модуля LPM\_RAM\_IO. На рисунке 2.2 изображено условно-графическое обозначение (УГО) блока ОЗУ.

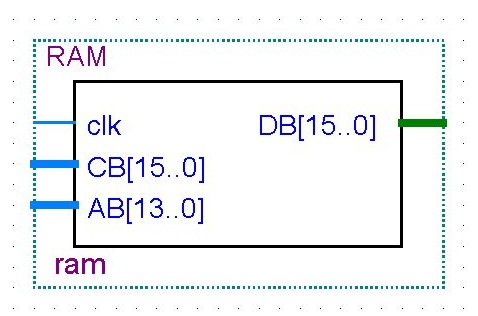


Рисунок 2.2 – Условно-графическое обозначение блока ОЗУ

Сигналы блока ОЗУ:

* *clk* – тактирующий сигнал.
* *CB* – контрольная шина.
* *AB* – адресная шина, передает адрес слова для чтения или записи.
* *DB* – шина данных, передает прочитанные данные или данные для записи.

Каждый из этих модулей содержит внутри себя проверку двух старших бит адреса. Если эти биты совпадают с битовым представлением номера модуля, то модуль активируется и отвечает на запрос (производить чтение или запись).

Важной особенностью работы обоих модулей является неспособность выполнить чтение или запись за один такт. Минимальное время выполнения любой операции – два такта. Во время первого такта модуль устанавливает адрес и только во время второго такта выполняется операция, относящаяся к установленному адресу. Это необходимо учитывать при разработке микропрограмм других модулей.

* 1. Регистры общего назначения.

Блок регистров общего назначения (см. рисунок 2.3) состоит из 24 16-разрядных регистров. Эти регистры предназначены для хранения временных результатов т.к. представляют собой более быструю память по сравнению с ОЗУ.

Особенностью данной реализации является то, что нулевой регистр зарезервирован и содержит значение регистра флагов. Это предоставляет пользователю возможность считать регистр флагов и использовать его значение в программах.

В дополнение к 24 доступным регистрам имеется ещё 2 дополнительных регистра: регистр данных и регистра адреса.

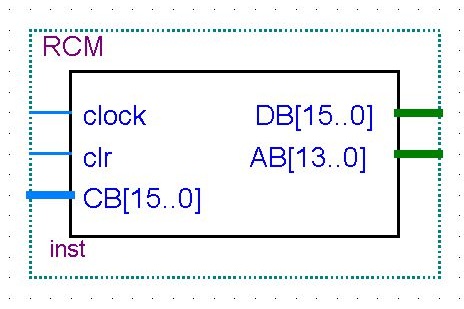


Рисунок 2.3 – Условно-графическое обозначение РОН

Сигналы блока РОН:

* clock – тактирующий сигнал.
* clr – сигнал сброса. Сбрасывает состояние всех внутренних элементов в исходное.
* CB – контрольная шина.
* DB – шина данных.
* AB – шина адреса.

Регистр данных работает автоматически и сохраняет в себя любое значение, которое считывается из любого источника информации. По умолчанию при записи регистр данных не выставляет сохраненное значение на шину данных. Для того, чтобы использовать сохраненное значение необходимо выставить третий бит контрольной шины в единицу.

Регистр адреса полностью управляется первым и вторым битами контрольной шины. Первый бит позволяет выставить значение, сохраненное в регистре адреса на адресную шину. Второй бит позволяет записать в регистр адреса значение, находящееся на шине данных.

* 1. Блок выборки команд.

Блок выборки команд (см. рисунок 2.4) считывает команду (инструкцию) из памяти инструкций, защелкивает считанную команду внутри себя и переводит указатель инструкций на следующую инструкцию. Этот блок представляет собой всю первую стадию конвейера.

Переход на следующую инструкцию не обязательно означает переход на следующую по счету инструкцию. Если в текущей стадии конвейера была прочитана инструкция прыжка, то следующей инструкцией может быть инструкция, расположенная по адресу прыжка.

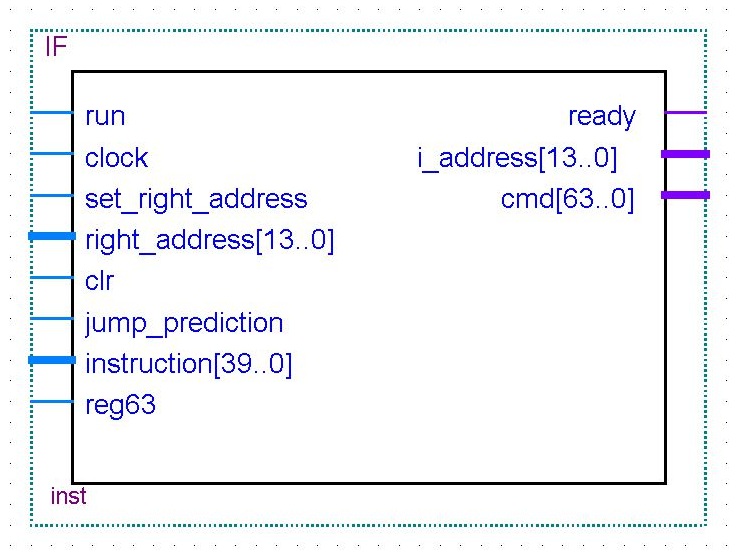


Рисунок 2.4 – Условно-графическое обозначение блока выборки команд

Сигналы блока выборки команд:

* *run* – сигнал запуска блока.
* *clock* – тактирующий сигнал.
* *set\_right\_address* – сигнал, который инициирует подмену текущего адреса на значение, поступающее по шине *right\_address*.
* *right\_address* – верный адрес перехода.
* *clr* – сигнал сброса. Сбрасывает состояние всех внутренних элементов в исходное.
* *jump\_prediction* – результат работы предсказателя переходов.
* *instruction* – полная считанная инструкция.
* *reg63* – сигнал включения/выключения последующих стадий конвейера.
* *ready* – сигнал готовности.
* *i\_address* – шина адреса считываемой команды.
* *cmd* – выходные данные, которые проходят через промежуточные регистры и направляются во вторую стадию конвейера.

Выбор конкретной ветви исполнения зависит от результата работы предсказателя. Однако адрес начала отвергнутой ветви исполнения не теряется, а сохраняется (записывается в промежуточные регистры конвейера) на место второго операнда в инструкции прыжка. Дополнительно сохраняется и текущее предсказание перехода. Таким образом когда инструкция прыжка окажется на стадии выполнения (вторая стадия конвейера) я смогу проверить верное ли было предсказание и исправить неверное предсказание.

Представим алгоритм работы блока в более наглядной форме:

1. Выставить адрес инструкции на адресную шину инструкций и подождать один такт.
2. Считать инструкцию и защелкнуть её.
3. Сменить адрес, содержащийся в указателе инструкций на следующий. Здесь возможно несколько вариантов.
   1. Если прочитанная инструкция не является прыжком, то увеличить текущее значение указателя инструкций на единицу.
   2. Если прочитанная инструкция является прыжком, то изменить значение указателя инструкций в соответствии с предсказанием и записать адрес начала отвергнутой ветви исполнения как второй операнд.
4. Завершить работу и выдать сигнал о готовности.

Данная микро-ЭВМ основан на гарвардской архитектуре, поэтому блок извлечения команд имеет отдельную шину данных и адреса, которые связаны с блоком памяти инструкций. Блок памяти инструкций представляет из себя стандартный элемент LPM\_ROM, поэтому, как упоминалось выше, чтение одной инструкции занимает два такта.

Выходная шина данного блока проходит через промежуточные регистры конвейера и направляется в блок извлечения операндов.

* 1. Блок извлечения операндов.

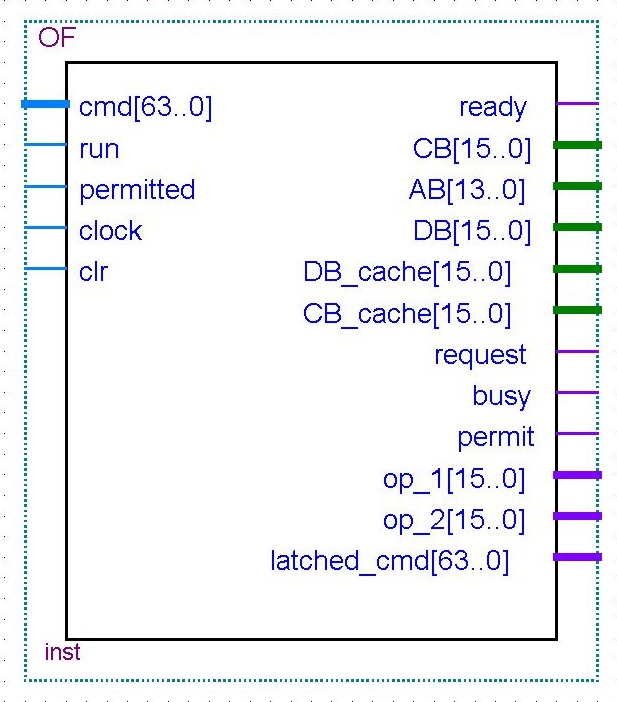


Рисунок 2.5 – Условно-графическое обозначение блока извлечения операндов

Сигналы блока извлечения операндов:

* *cmd* – входные данные из промежуточных регистров конвейера.
* *run* – сигнал запуска.
* *permitted* – сигнал получения разрешения на взаимодействие с шинами.
* *clock* – тактирующий сигнал.
* *clr* – сигнал сброса. Сбрасывает состояние всех внутренних элементов в исходное.
* *ready* – сигнал готовности.
* *CB* – контрольная шина.
* *AB* – шина адреса.
* *DB* – шина данных.
* *DB\_cache* – шина данных кэша.
* *CB\_cache* – контрольная шина кэша.
* *request* – сигнал запроса на взаимодействие с шинами.
* *busy* – сигнал, показывающий, что блок получил разрешение и сейчас работает с шинами.
* *permit* – сигнал, при помощи которого блок разрешает работу с шинами следующему блоку.
* *op1* – первый извлеченный операнд.
* *op2* – второй извлеченный операнд.
* *latched\_cmd* – защелкнутые входные данные.

Вторая стадия конвейера (извлечение операндов и выполнение) начинается с запуска этого блока. Блок извлечения операндов (см. рисунок 2.5) занимается получением определенного количества операндов в зависимости от операции. Так как в командах содержатся дополнительные биты, сигнализирующие о количестве операндов, то блоку не требуется декодировать номер операции для того, чтобы узнать количество операндов.

При включении блок действует по следующему алгоритму:

1. Выставляет запрос на доступ к шинам.
2. Ожидает получения разрешения.
3. Выполняет микропрограмму загрузки искусственно изменяя её длину в зависимости от дополнительных битов команды.
4. Подает сигнал ready.

Каждый блок, которому необходимо получать доступ к шинам, имеет внутри себя специальный блок проверки и получения доступа DMA\_check, который показан на рисунке 2.6.

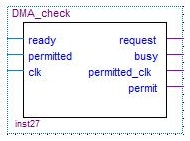


Рисунок 2.6 – Условно-графическое обозначение блока проверки и получения доступа к шинам

Сигналы блока проверки и получения доступа:

* *ready* – после завершения работы основного блока на этот вход поступает высокий уровень сигнала, чтобы передать доступ другому блоку.
* *permitted* – сигнал разрешения работы с шинами.
* *clk* – тактовый сигнал.
* *request* – сигнал запроса на работу с шинами.
* *busy* – сигнал показывает, что блок захватил управление и использует шины.
* *permitted\_clk* – тактовый сигнал, который присутствует только если разрешение получено. Если разрешения нет, то подается постоянный низкий уровень сигнала.
* *permit* – сигнал передачи разрешения следующему блоку.

Основной идеей этого блока является выход permitted\_clk. Именно он служит тактовым сигналом для всего блока, который должен контролироваться арбитром шин. Таким образом, пока разрешение не получено, блок будет искусственно задерживаться из-за отсутствия тактового сигнала.

Последовательность работы блока проверки и получения доступа:

1. На вход ready поступает низкий уровень сигнала. Это означает, что контролируемое устройство начало работу и требует доступа к шинам.
2. На выход request выставляется высокий уровень сигнала.
3. Блок ожидает поступления сигнала permitted.
4. После получения сигнала permitted блок выставляет сигнал busy в высокий уровень и сбрасывает сигнал request. На выход permitted\_clk начинает поступать тактовый сигнал.
5. Блок ожидает высокого уровня сигнала ready, что будет означать о завершении работы блока.
6. После получения сигнала ready блок сбрасывает сигнал busy, чтобы арбитр мог сгенерировать разрешения для другого блока.

Из-за специфики работы блока проверки и получения доступа в основной блок не нужно вносить никаких изменений. Достаточно добавить блок DMA\_check и сделать выходной сигнал permitted\_clk тактирующим для всего блока.

После получения разрешения в дело вступает микропрограмма блока выборки операндов. Микропрограмма состоит из двух очень похожих частей. Первая часть содержит микрокоманды, извлекающие первый операнд, вторая часть – микрокоманды, извлекающие второй операнд. В зависимости от значения дополнительных бит команды изменяется длина микропрограммы (см. таблицу 2.1).

Таблица 2.1 – Зависимость микропрограммы от количества операндов

|  |  |
| --- | --- |
| Количество операндов | Микропрограмма |
| 0 | Микропрограмма имеет длину 0. На выход мгновенно подается сигнал готовности |
| 1 | Длина микропрограммы равна длине только первой части. Загружается только первый операнд |
| 2 | Выполняется вся микропрограмма. Загружаются оба операнда |

Такая реализация позволяет загружать нужное количество операндов и тратить только минимально необходимое количество тактов.

* 1. Блок выполнения.

Блок выполнения (см. рисунок 2.7), так же как и блок извлечения операндов находится во второй стадии конвейера. Он зависит от работы блока извлечения операндов и включается только тогда, когда операнды извлечены. Сигналом запуска для этого блока служит сигнал готовности блока извлечения операндов.

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 дешифраций текущей команды и включается необходимый блок выполнения.

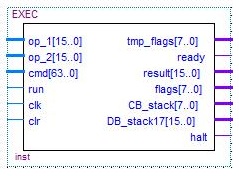


Рисунок 2.7 – Условно-графическое обозначение блока выполнения

Сигналы блока выполнения:

* *op\_1* – значение первого операнда.
* *op\_2* – значение второго операнда.
* *cmd* – защелкнутое значение промежуточного регистра между первой и второй стадией.
* *run* – сигнал запуска блока.
* *clr* – сигнал сброса. Сбрасывает состояние всех внутренних элементов в исходное.
* *clk* – тактирующий сигнал.
* *tmp\_flags* – отладочный сигнал. Данные на шине временных флагов.
* *ready* – сигнал готовности блока.
* *result* – результат операции.
* *flags* – флаги операции.
* *CB\_stack* – отладочный сигнал. Контрольная шина стека.
* *DB\_stack17* – отладочный сигнал. Шина данных стека.
* *halt* – сигнал остановки всего устройства.

После того как блок выполнения конкретной команды отработал, он оставляет на выходных шинах значения результата и флагов. Эти значения подаются на выход, пока на вход run блока выполнения операции подается высокий уровень сигнала. Эти сигналы должны подаваться до конца выполнения стадии конвейера, чтобы сохранить результат в промежуточные регистры между второй и третьей стадией.

Также блок выполнения содержит временный регистр флагов и стек. Каждый из них внесен внутрь блока выполнения для ускорения работы устройства.

Напомню, что в нулевой регистр общего назначения переносятся значения флагов доступные программисту. Запись значения флагов осуществляет блок записи на третьей стадии конвейера. Если мы будем использовать значение этого регистра для выполнения операций, то в работе устройства появятся значительные задержки. Если операций зависит от регистра флагов, то произойдет конфликт чтения-записи (операция зависит от регистра флагов поэтому его необходимо прочитать, но блок записи ещё не успел записать туда верные значения). Для разрешения конфликта придется останавливать первую и вторую стадию конвейера и ожидать завершения работы блока записи. Значительное количество конфликтов резко уменьшит производительность блока и всей системы в целом.

Поэтому я принял решение ввести временный регистр флагов в устройство выполнения. Этот регистр сохраняет значения флагов предыдущей операции и предоставляет эти значения для следующей команды. Он позволяет выполнять команды, зависящие от флагов, не ожидая записи и чтения регистра флагов из нулевого регистра общего назначения. Если программист будет использовать регистр флагов (нулевой РОН) как операнд, то конфликт чтения-записи все же произойдет. Точно так же оно произойдет, если как операнд будет использоваться результат работы прошлой операции, поэтому значения флагов также можно считать результатом работы операции и учитывать это при написании программ.

* 1. Стек.

Я решил включить стек (см. рисунок 2.8) в блок выполнения по двум причинам:

1. Стек управляется исключительно двумя блоками PUSH и POP. Оба блока находятся в блоке выполнения.
2. При подключении стека на общие шины блок выполнения будет обязан их использовать и конкурировать за право доступа с остальными блоками. Это замедлит работу устройства в целом.

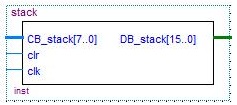


Рисунок 2.8 – Условно-графическое обозначение стека

Сигналы стека:

* *CB\_stack* – контрольная шина стека.
* *clr* – сигнал сброса. Сбрасывает состояние всех внутренних элементов в исходное.
* *clk* – тактирующий сигнал.
* *DB\_stack* – шина данных стека.

Стек полностью контролируется блоками, отвечающими за выполнение операций PUSH и POP. По контрольной шине передаются сигналы управления стеком, а по шине данных передаются данные и значения флагов.

* 1. Блок записи.

Блок записи (см. рисунок 2.9) находится в третьей стадии конвейера и отвечает за запись результатов в память. Для нормального функционирования этому блоку необходим доступ к общим шинам, поэтому он, так же как и блок извлечения операндов, содержит блок проверки и получения доступа к шинам.

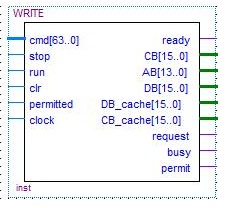


Рисунок 2.9 – Условно-графическое обозначение блока записи

Сигналы блока записи:

* *cmd* – значение промежуточного регистра конвейера между второй и третьей стадией.
* *stop* – сигнал остановки блока. Если подается высокий уровень сигнала, то блок не может быть запущен.
* *permitted* – сигнал разрешения на доступ к шинам.
* *run* – сигнал запуска блока.
* *clr* – сигнал сброса. Сбрасывает состояние всех внутренних элементов в исходное.
* *clock* – тактирующий сигнал.
* *ready* – сигнал готовности блока.
* *CB* – контрольная шина.
* *AB* – шина адреса.
* *DB* – шина данных.
* *DB\_cache* – шина данных кэша.
* *CB\_cache* – контрольная шина кэша.
* *request* – сигнал запроса на доступ к шинам.
* *busy* – устройство захватило доступ к шинам и использует его.
* *permit* – сигнал передачи разрешения на доступ к шинам другому устройству.

При доступе к шинам блок записи имеет более низкий приоритет, чем блок чтения для того, чтобы запись происходила параллельно с выполнением операции, а извлечение операндов происходило параллельно с извлечением инструкции. Такое распределение по времени даст наибольший прирост производительности.

Блок записи имеет следующий алгоритм работы:

1. Выставляет запрос на доступ к шинам.
2. Ожидает получения разрешения.
3. Выполняет микропрограмму записи результата и флагов.
4. Подает сигнал ready.
   1. Блок управления.

Блок управления (см. рисунок 2.10) контролирует работу всего устройства и разрешает возникающие конфликты.

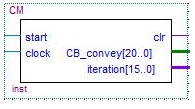


Рисунок 2.10 – Условно-графическое обозначение блока управления

Сигналы блока управления:

* *start* – сигнал запуска устройства.
* *clock* – тактирующий сигнал.
* *clr* – сигнал сброса. Сбрасывает состояние всех внутренних элементов в исходное. Подается в первую половину такта при запуске.
* *CB\_convey* – шина управляющих и информационных сигналов конвейера.
* *iteration* – отладочная шина. Показывает текущую итерацию конвейера.

С самого начала разработки было ясно, что контрольный модуль будет иметь значительное количество сигналов для управления конвейером и опроса его состояния. Для того, чтобы наглядно упростить схему и упростить отладку я принял решение объединить все эти сигналы в одну шину. В таблице 2.2 представлено детальное описание сигналов.

Таблица 2.2 – Сигналы шины CB\_convey

|  |  |
| --- | --- |
| Бит | Значение |
| 0 | Блок чтения инструкций завершил выполнение |
| 1 | Блок извлечения операндов завершил выполнение |
| 2 | Блок выполнения команды завершил выполннение |
| 3 | Блок записи завершил выполнение |
| 4 | Присутствует конфликт чтения-записи |
| 5 | Предсказание на прошлой стадии конвейера было верное |
| 6 | Остановить устройство после завершения текущей стадии конвейера |
| 7 | Защелкнуть результат проверки на конфликт чтения-записи |
| 8 | Запустить все блоки конвейера |
| 9 | Запустить только блок записи |
| 10 | Перезаписать промежуточные регистры конвейера (сохранить результат работы текущей стадии) |
| 11 | Сменить регистры конвейера |
| 12 | Исправить адрес извлечения инструкций, который был испорчен из-за неправильного предсказания |
| 13 | Остановить вторую стадию конвейера (извлечение операндов и выполнение операции) |
| 14 | Остановить блок записи |
| 15 | Самый старший бит в промежуточных регистрах. Отвечает за включение или выключение стадий. При низком уровне сигнала устройство остановится после завершения обработки команды. |
| 16 | Вычислить следующее предсказание |
| 17 | Защелкнуть текущее предсказание |

Блок управления основан на счетчике и дешифраторе, который указывает, какой этап сейчас выполняется. Если хотя бы один блок в конвейере ещё занят, в текущий этап вставляются дополнительные такты ожидания. Когда все блоки завершили свою работу – осуществляется переход к следующему этапу работы блока управления. Ниже описаны все этапы работы блока управления:

1. Запустить все блоки конвейера, если нет конфликтов чтения-записи.
2. Запустить блок записи, если есть конфликт чтения записи. Вычислить следующее предсказание.
3. Если предыдущий прыжок был с неверным предсказанием, то исправить адрес и сохранить флаг «неверный прыжок».
4. Перезаписать промежуточные регистры, если нет конфликтов чтения-записи. Защелкнуть предсказание.
5. Сменить промежуточные регистры, если нет конфликтов чтения-записи. Остановить блок записи, если есть конфликт чтения-записи.
6. Остановить вторую и третью стадии конвейера, если выставлен флаг «неверный прыжок». Перезаписать текущее значение конфликта чтения-записи.
   1. Предсказатель

Предсказатель переходов (см. рисунок 2.11) необходим для систем, использующих конвейер и вычисляющих реальное значение перехода не последовательно. Если извлечение инструкций и выполнение операций происходят параллельно, то мы должны извлекать следующую инструкцию до того, как узнаем какая ветвь исполнения верная. Безусловно, можно не использовать предсказатель переходов, а просто считывать следующую по порядку инструкцию, но это значительно понизит производительность всей системы. Именно из-за сложности и длительности операции очистки конвейера от инструкций, которые не должны исполняться применяются предсказатели. Они пытаются минимизировать количество ошибочно выбранных ветвей исполнения и тем самым уменьшают количество очисток конвейера.

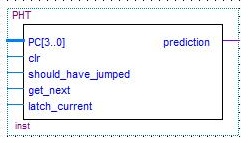


Рисунок 2.11 – Условно-графическое обозначение предсказателя

Сигналы блока управления:

* *PC* – последние четыре бита указателя инструкций.
* *clr* – сигнал сброса. Сбрасывает состояние всех внутренних элементов в исходное. Подается в первую половину такта при запуске.
* *should\_have\_jumped* – должен ли был произойти прыжок. Проверенное значение.
* *get\_next* – вычислить следующее предсказание.
* *latch\_current* – защелкнуть текущее предсказание.
* *prediction* – текущее предсказание.

Предсказания этого блока основаны на таблице шаблонов истории. Эта таблица состоит из автоматов А5 (см. рисунок 2.12), которые изменяют своё состояние в зависимости от того произошел реально переход или не произошел. Конкретный автомат в таблице выбирается в зависимости от значения последних 4 битов указателя инструкций.

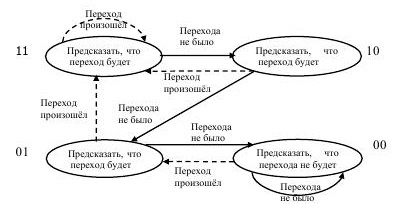


Рисунок 2.12 – Состояния и схема переходов автомата А5

* 1. КПДП.

Контроллер прямого доступа к памяти (см. рисунок 2.13) 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.

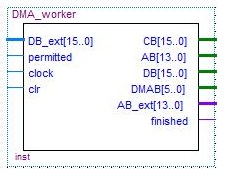


Рисунок 2.13 – Условно-графическое обозначение КПДП

Сигналы КПДП:

* *DB\_ext* – шина данных внешнего устройства.
* *permitted* – сигнал разрешения на захват шин.
* *clock* – тактовый сигнал.
* *clr* – сигнал сброса. Сбрасывает состояние всех внутренних элементов в исходное. Подается в первую половину такта при запуске.
* *CB* – контрольная шина.
* *AB* – адресная шина.
* *DB* – шина данных.
* *DMAB* – шина для взаимодействия с арбитром шин.
* *AB\_ext* – шина адреса внешнего устройства.
* *finished* – отладочный сигнал. КПДП окончил передачу одного блока данных.

Для того чтобы КПДП не нарушал работу остальных блоков, он имеет самый низкий приоритет доступа к шинам. В моей реализации КПДП имеет 4 регистра конфигурации, которые описаны в таблице 2.3.

Таблица 2.2 – Сигналы шины CB\_convey

|  |  |
| --- | --- |
| Адрес регистра | Значение |
| 0x3000 | Количество слов, которые нужно передать |
| 0x3001 | Адрес назначения |
| 0x3002 | Адрес источника |
| 0x3003 | Запустить КПДП – выставить все биты в 1. Отключить ПКДП – выставить биты в 0. |

Алгоритм работы КПДП:

1. Ожидать запуска (все биты регистра 0х3003 равны 1).
2. Выставить запрос на доступ к шинам.
3. После получения запроса выставить на внешнюю шину адреса адрес источника, а на внутреннюю шину адреса адрес назначения. Ожидать один такт.
4. Соединить внешнюю и внутреннюю шины данных и произвести копирование одного слова.
5. Освободить шины и передать управление следующему блоку.
6. Если скопировано меньше слов, чем указано в регистре 0х3000, то перейти к пункту 2, иначе к пункту 7.
7. Очистить регистр 0х3003.
   1. Арбитр.

В данной реализации микро-ЭВМ присутствует несколько устройств, которые могут работать независимо и использовать общие шины, поэтому необходимо устройство, которое предотвратит конфликты и будет выдавать разрешение на работу с шинами строго по приоритету. В данном случае используется последовательный централизованный арбитраж. Это значит, что в схеме присутствует отдельное устройство, которое берет на себя задачу по выдаче разрешений на доступ к шине.

Среда разработки Altera Quartus II 9.1 не поддерживает возможности создания монтажного или, поэтому я решил использовать отдельные связи для каждого контролируемого устройства и уже в блоке арбитра (см. рисунок 2.14) объединять контакты через логическое или.

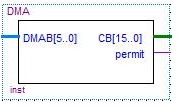


Рисунок 2.14 – Условно-графическое обозначение арбитра

Сигналы арбитра:

* *DMAB* – шина для связи контролируемых устройств и арбитра.
* *CB* – контрольная шина.
* *permit* – выдача разрешения.

Битовое представление шины DMAB представлено в таблице 2.3. Эта шина используется для взаимодействия арбитра и всех контролируемых устройств.

Таблица 2.2 – Сигналы шины DMAB

|  |  |
| --- | --- |
| Бит | Значение |
| 0 | Запрос на доступ от блока извлечения операндов |
| 1 | Запрос на доступ от блока записи |
| 2 | Запрос на доступ от внешнего устройства |
| 3 | Сигнал занятости шины от блока извлечения операндов |
| 4 | Сигнал занятости шины от блока записи |
| 5 | Сигнал занятости шины от внешнего устройства |

Последовательный арбитраж означает, что разрешение выданное арбитром будет последовательно передаваться от устройства к устройству, пока сигнал не дойдет до блока, который запрашивал разрешение. Последовательность подключения напрямую соотносится с приоритетом устройства. Если устройство в цепочке находится ближе к арбитру, то его приоритет выше, так как оно сможет получить разрешение раньше других.

1. ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ

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

* 1. Постоянное запоминающее устройство.

Постоянное запоминающее устройство предоставляет возможность чтения записанных данных. Память ПЗУ отображена на адреса 0х0000 – 0х0FFF. Заполнена память следующим образом: если ADDR – это адрес ячейки, то значение в этой ячейке также равно ADDR. На рисунке 3.1 продемонстрировано чтение из ПЗУ, а также реакция ПЗУ на адрес, не принадлежащий к его диапазону адресов.

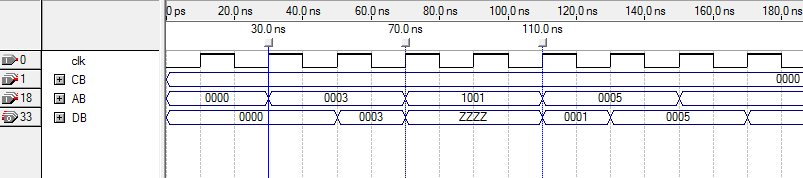


Рисунок 3.1 – Функциональное моделирование ПЗУ

В интервале 30-70 наносекунд (далее я буду опускать единицы измерения времени) виден процесс чтения из ПЗУ по адресу 0х0003. Здесь хорошо видна задержка в один такт между выставлением адреса и получением данных. Аналогично работает ОЗУ. Именно из-за этой особенности я вставлял дополнительный такт задержки при чтении-записи.

На отрезке 70-110 видно, что, если на выставлен адрес, не принадлежащий диапазону адресов ПЗУ, то блок не засоряет внешние шины.

* 1. Оперативное запоминающее устройство.

Оперативное запоминающее устройство очень похоже на ПЗУ с тем лишь отличием, что оно позволяет, как считывать данные, так и записывать их. Память ОЗУ отображена на адреса 0х1000 – 0х1FFF. На рисунке 3.2 продемонстрированы оба процесса.

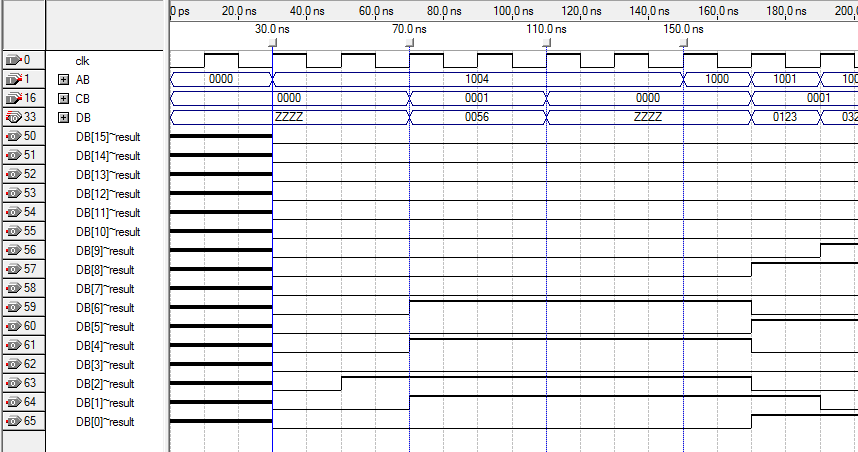


Рисунок 3.2 – Функциональное моделирование ОЗУ

На отрезке 0-30 видно, что, если на выставлен адрес, не принадлежащий диапазону адресов ОЗУ, то блок не засоряет внешние шины.

В интервале 30-70 наносекунд виден процесс чтения из ОЗУ по адресу 0х1004. После этого в интервале 70-110 происходит запись значения 0х56 по адресу 0х1004. Затем в интервале 110-150 мы снова читаем по адресу 0х1004 и подтверждаем, что значение 0х56 успешно записалось.

* 1. Регистры общего назначения.

Регистры общего назначения отличаются тем, что при работе с ними достаточно одного такта для чтения и одного такта для записи. Они отображены на адреса 0х2000 – 0х2018, где помещается ровно 24 регистра.

На рисунке 3.3 продемонстрированы операции чтения и записи данных в РОН.

В начале симуляции выставляется сигнал clr, что приводит к очистке всех регистров. После этого в промежутке 70-90 в пятый регистр записывается значение 7. В интервале 90-110 мы убеждаемся, что значение было успешно записано.

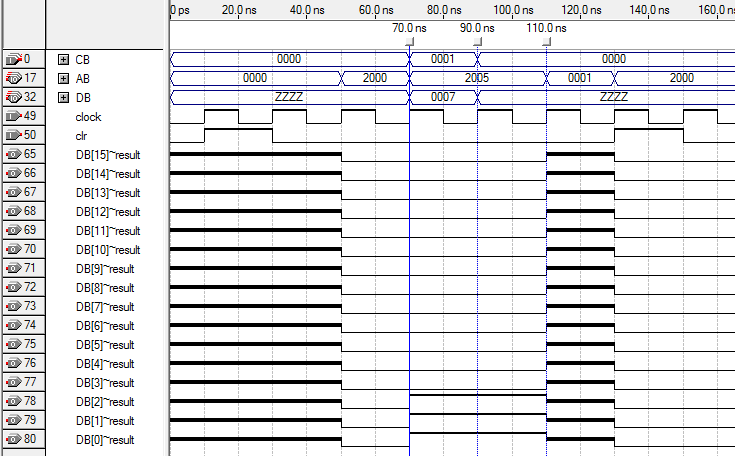


Рисунок 3.3 – Функциональное моделирование РОН

* 1. Блок выборки инструкций.

Блок выборки инструкций считывает следующую инструкцию (в зависимости от предсказания, если прошлая инструкция была прыжком) и подает результат на промежуточные регистры конвейера. На рисунку 3.4 представлено чтение очередной инструкции.

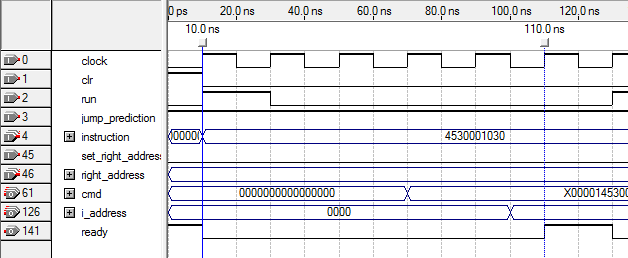


Рисунок 3.4 – Функциональное моделирование выборки инструкции

Можно отметить, что блок начинает свою работу, когда поступает сигнал run. Полный цикл работы блока занимает 5 тактов.

* 1. Блок извлечения операндов.

Блок извлечения операндов занимается считыванием необходимого количества операндов из памяти. На рисунке 3.5 можно увидеть результат функционального моделирования извлечения операндов.

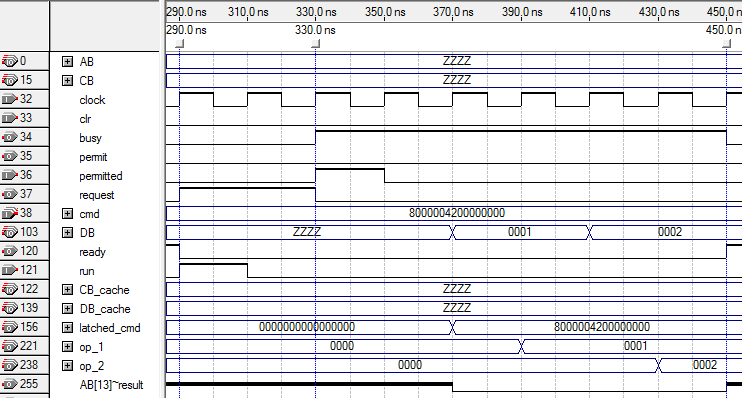


Рисунок 3.5 – Функциональное моделирование извлечения операндов

Блок начинает свою работу после подачи сигнала run. После этого блок выставляет запрос на доступ к шинам и ожидает получения доступа. Когда доступ получен (получен сигнал permitted), блок начинает выполнение микропрограммы. В данном случае мы имеем операцию AND (старший байт 0x42). Это значит, что необходимо загружать оба операнда. В ходе работы блока мы видим, что был считан первый и второй операнд.

После получения разрешения на доступ к шинам блоку требуется до 6 тактов, чтобы полностью выполнить микропрограмму извлечения операндов.

* 1. Блок выполнения.

Блок выполнения отвечает за выбор конкретной операции и ее исполнение. В данном случае мы имеем операцию AND (старший байт 0x42). Операнды равны 0хF003 и 0xF006. Это значит, что результат должен быть равен 0xF002 и должен быть выставлен первый бит регистра флагов (отрицательный результат).

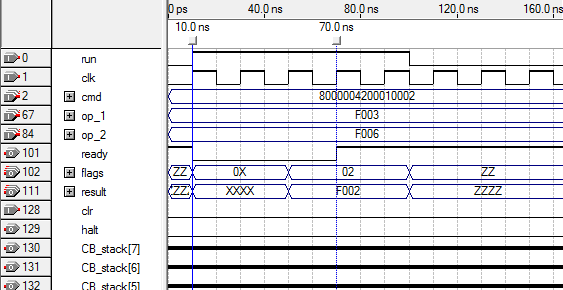
.

Рисунок 3.6 – Функциональное моделирование выполнения операции

Как мы видим, операция AND была исполнена полностью правильно. В таблице 3.1 описан регистр флагов, чтобы можно было проверить результат выполнения. На выполнение было затрачено 3 такта. В общем случае выполнение может занимать до 6 тактов (операции PUSH и POP).

Таблица 3.1 – Биты регистра флагов

|  |  |
| --- | --- |
| Бит регистра флагов | Значение |
| 0 | Результат равен нулю |
| 1 | Результат меньше нуля |
| 2 | Керри бит |
| 3 | Бит переполнения |
| 4 | Стек пуст, ошибка чтения |
| 5 | Стек полон, ошибка записи |

* 1. Стек.

Стек расположен внутри блока выполнения операций и имеет свои шину данных и контрольную шину. В таблице 3.2 представлены все значения контрольной шины стека. Блок стека основан на модуле LPM\_RAM\_IO, отсюда следует, что операции чтения или записи будут иметь задержку в один такт (см. пункт 3.1).

Таблица 3.2 – Контрольная шина стека

|  |  |
| --- | --- |
| Номер бита | Значение |
| 0 | - |
| 1 | Увеличить указатель стека |
| 2 | Уменьшить указатель стека |
| 3 | Записать данные в стек |
| 4 | Прочитать данные из стека |
| 5 | Прочитать регистр флагов из стека |

На рисунке 3.7 изображен результат функционального моделирования записи значения в стек, увеличения указателя, записи значения, уменьшения указателя, чтения первого записанного значения.

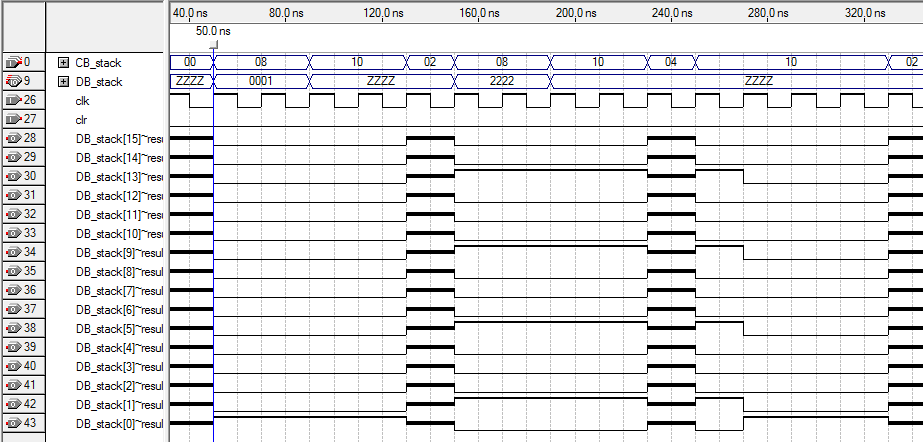


Рисунок 3.7 – Функциональное моделирование стека

* 1. Блок записи.

Блок записи занимается выгрузкой результата операции и регистра флагов в память. На рисунке 3.8 видно, как происходит эта операция. Блок начинает свою работу сразу после получения сигнала run. После этого блок запрашивает разрешение на доступ к шинам. Когда доступ получен, блок начинает выгрузку значений в память. По шине cmd мы можем определить, что результат операции равен 0х1111, адрес выгрузки операции равен 0х2222, регистр флагов равен 0х00.

Для записи значения в регистр достаточно одного такта, поэтому запись в регистр флагов производится в промежутке времени 50-70. Для того, чтобы гарантировать, что результат точно запишется верно, необходимо производить запись не менее двух тактов (из-за одного такта задержки в ПЗУ и ОЗУ). Поэтому запись результата происходит в интервале 70-110.

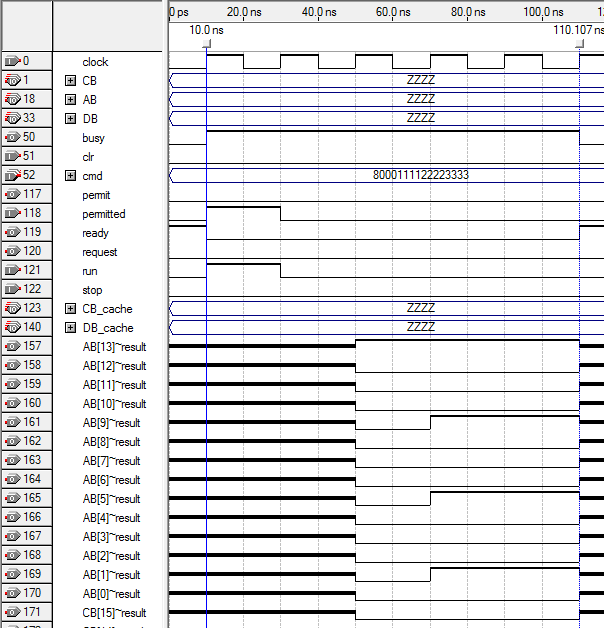


Рисунок 3.8 – Функциональное моделирование блока записи

* 1. Предсказатель.

В данной микро-ЭВМ предсказатель использует автоматы А5. На рисунке 3.9 показан результат модуляции автомата А5, который используется в построении таблицы шаблонов истории.

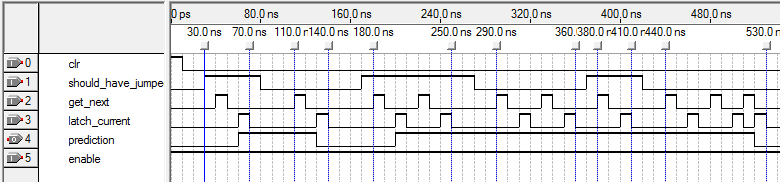


Рисунок 3.9 – Функциональное моделирование автомата А5

Сначала автомат находится в состоянии 00. В промежутке 30-70 автомат переходит в состояние 01. После того, как переход не произошел, автомат возвращается в состояние 00 на промежутке 110-140.

На промежутке 180-250 произошло два перехода, что переводит автомат в состояние 11. На промежутке 290-360 оба перехода не произошли, и это перевело автомат в состояние 01.

В интервале 410-440 произошел переход, и автомат снова вернулся в состояние 11. Для того чтобы вывести автомат в состояние 00 потребовалось 3 не произошедших перехода в промежутке 440-530.

* 1. КПДП.

На рисунке 3.10 показана работа КПДП. КПДП осуществляет перенос слов на выделенных интервалах. Сигналы шины T\_DMAB подробно описаны в таблице 2.2. КПДП завершает свою работу после переноса каждого слова, чтобы другой блок с более высоким приоритетом мог получить управление.

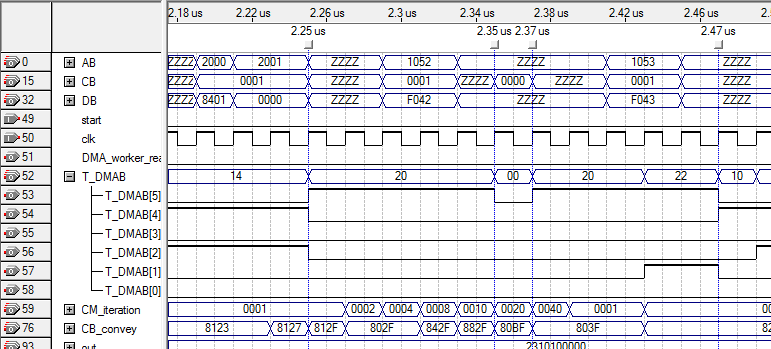


Рисунок 3.10 – Функциональное моделирование КПДП

* 1. Арбитр.

В моей реализации используется последовательный централизованный арбитраж. На рисунке 3.11 представлен результат функционального моделирования арбитра.

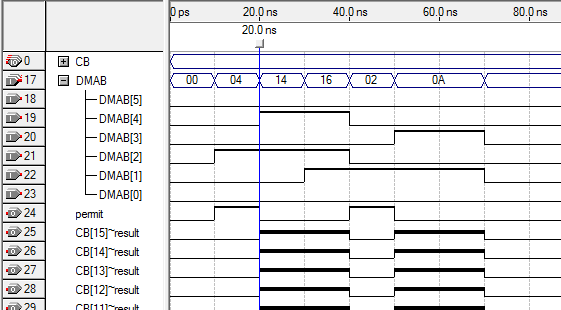


Рисунок 3.11 – Функциональное моделирование арбитра

На рисунке 3.11 видно, что блок арбитра обнуляет контрольную шину, если ни один блок не взаимодействует с ней. Это сделано для того, чтобы исключить неопределенный уровень сигнала на контрольной шине.

Так как используется последовательный арбитраж, то при модуляции схемы арбитра невозможно смоделировать случай конкуренции запросов. Это реализуется за счет последовательного подключения контролируемых блоков.

* 1. Тестовая программа.

Для того чтобы полностью представлять себе все значения, которые использует программа представим начальное заполнение блоков ОЗУ (см. рисунок 3.12) и ПЗУ (см. рисунок 3.13). На рисунке 3.14 представлены данные, хранящиеся во внешнем устройстве.

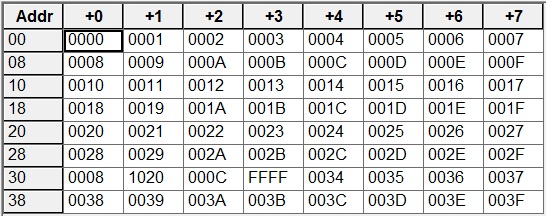


Рисунок 3.12 – Содержимое ОЗУ

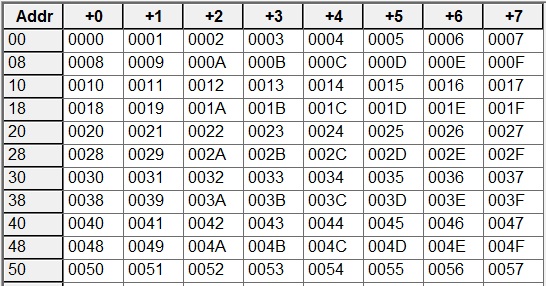


Рисунок 3.13 – Содержимое ПЗУ

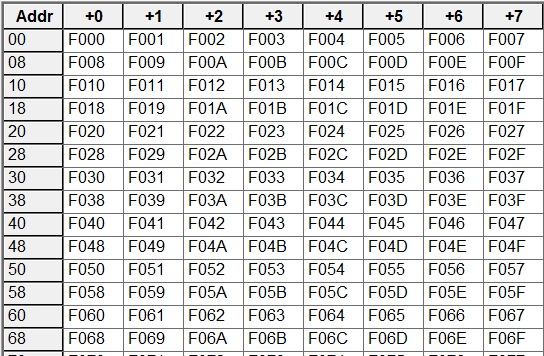


Рисунок 3.14 – Содержимое внешнего устройства

Таблица 3.3 – Тестовая программа

|  |  |  |
| --- | --- | --- |
| Адрес | Псевдокоманда | Шестнадцатеричный вид |
| 0 | MOV DMA[0], RAM[30] | 45 3000 1030 |
| 1 | MOV DMA[1], RAM[31] | 45 3001 1031 |
| 2 | MOV DMA[2], RAM[32] | 45 3002 1032 |
| 3 | MOV DMA[3], RAM[33] | 45 3003 1033 |
| 4 | AND RCM[1], ROM[10] | 42 2001 0010 |
| 5 | NOTZ RAM[3] | 21 1003 0000 |
| 6 | NOTZ RAM[3] | 21 1003 0000 |
| 7 | INCS RAM[10] | 23 1010 0000 |
| 8 | INCS RAM[10] | 23 1010 0000 |
| 9 | ROL RAM[2] | 24 1002 0000 |
| A | MOV RCM[3], RAM[4] | 45 2003 1004 |
| B | MOV RAM[5], RCM[3] | 45 1005 2003 |
| C | PUSH RAM[10] | 27 1010 0000 |
| D | POP RAM[7] | 18 1007 0000 |
| E | JMP 0x10 | 96 0010 0000 |
| F | - | - |
| 10 | AND RCM[1], RAM[1] | 42 2001 1001 |
| 11 | NOTZ RAM [14] | 21 1014 0000 |
| 12 | JBZ 0x20 | 9A 0020 0000 |
| 13 | - | - |
| … | - | - |
| 20 | HLT | 19 0000 0000 |

Проанализируем работу программы, представленной в таблице 3.3. Первые 4 команды производят конфигурацию КПДП. Количество слов для копирования выставляется в 8. Адрес назначения устанавливается в 0х1020 (RAM[20]). Адрес источника устанавливается в 0x000С. Все биты третьего конфигурационного регистра выставляются в единицу, что приводит к запуску КПДП. В содержимом ОЗУ после выполнения программы мы можем видеть 8 скопированных слов из адреса 0х000С, находящихся по адресу 0x1020 (RAM[20]).

После этого выполняется операция логического И между операндами RCM[1] и ROM[10]. При старте все регистры обнуляются, поэтому значение RCM[1] равно 0, а значит и результат будет равен 0. Эта операция приводит к выставлению флага Z.

Пятая команда выполняет операцию NOTZ над операндом RAM[3]. Учитывая, что по адресу RAM[3] располагалось значение 3, результатом будет значение 0xFFFC. Результат не является нулем, поэтому флаг Z будет сброшен.

Шестая команда дублирует пятую и это значит, что последовательное выполнение этих команд приведет к конфликту чтения-записи. В моей реализации микро-ЭВМ присутствует механизм разрешения таких конфликтов, поэтому команда будет успешно выполнена. В данном случае команда NOTZ не должна изменить операнд потому, что флаг Z сброшен. Если бы механизм разрешения конфликтов чтения-записи не был предусмотрен, то эта операция прочитала бы значение 3, а не результат предыдущей операции (0хFFFC), и записала бы его в RAM[3]. Однако механизм разрешения конфликтов приостановил первую и вторую стадию конвейера для того, чтобы результат успел попасть в память. Подтверждение этому мы видим на рисунке 3.15, где представлен результат работы программы.

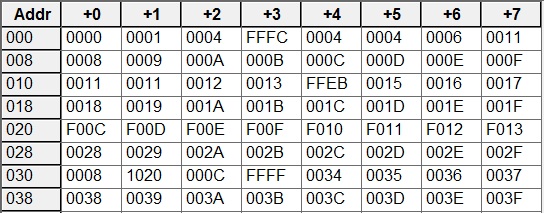


Рисунок 3.15 – Содержимое ОЗУ после выполнения программы

Результат предыдущей команды меньше нуля, поэтому флаг S будет выставлен. Выполнение операции INCS RAM[10] приведет к увеличению этого слова на единицу. Так как результатом будет число 0хB, то после выполнения флаг S будет сброшен. Поэтому повторное выполнение команды INCS никак не повлияет на содержимое памяти. Доказательством может служить значение значение 0x11 по адресу RAM[11].

Девятая инструкция приведет к сдвигу влево слова по адресу RAM[2]. Результатом будет число 4, что мы и видим по адресу RAM[2].

Десятая и одиннадцатая инструкции копируют слово RAM[4] в RAM[5] использую регистр RCM[3] как промежуточное хранилище. На рисунке 3.15 мы видим, что операции прошли успешно: значение RAM[5] равно значению RAM[4].

Двенадцатая и тринадцатая инструкции выполняют аналогичные действия. Копируется слово из RAM[11] в RAM[7], используя стек, как промежуточное хранилище. Опять же подтверждением служит равенство слов RAM[11] и RAM[7].

После этого происходит безусловный прыжок на инструкцию по адресу 0x10. Инструкция по адресу 0x10 аналогична четвертой и результатом её будет ноль. Флаг Z будет выставлен. Инструкция 0х11 инвертирует слово RAM[14] и выставит флаг S.

Инструкция 0x12 является условным прыжком. Так как флаг S выставлен, то предыдущий результат был меньше нуля, а значит, условный переход сработает и выполнение перейдет на адрес 0x20.

По адресу 0x20 находится команда остановки. Устройство закончит выполнение текущей стадии конвейера и завершит работу.

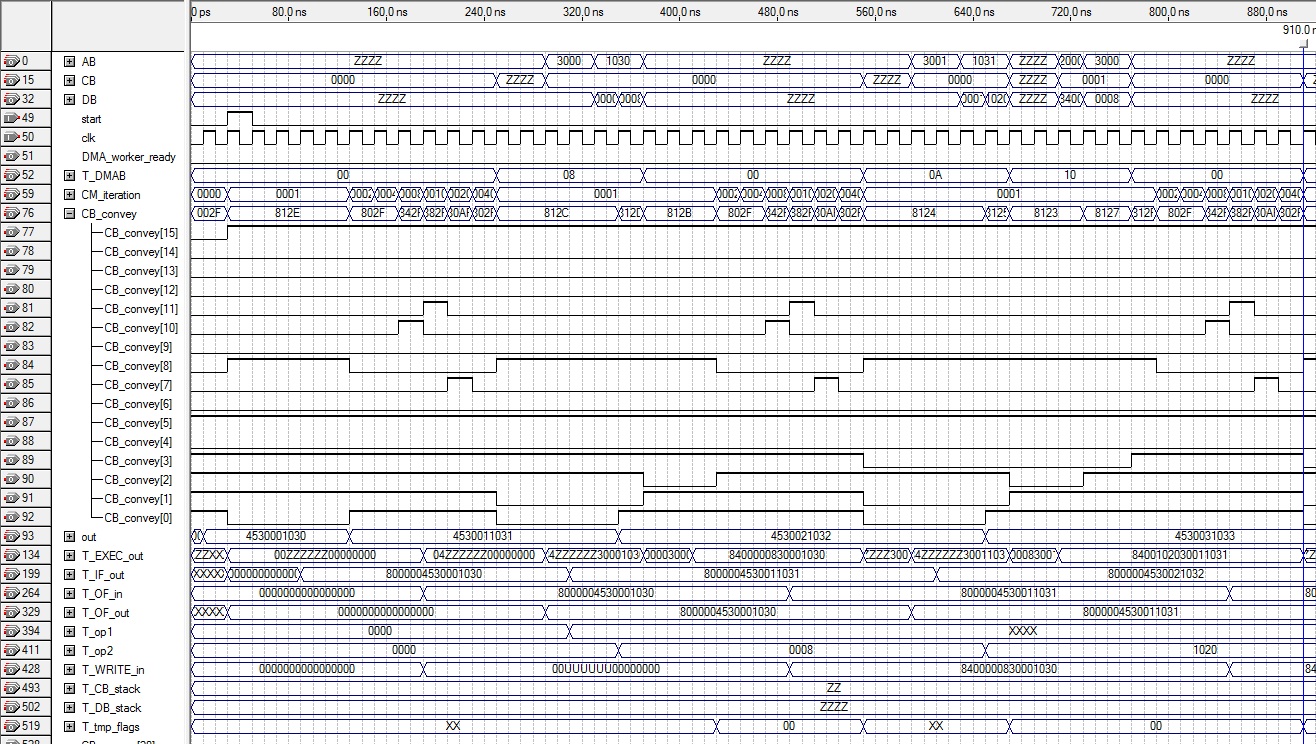


Рисунок 3.16 – Функциональное моделирование работы микро-ЭВМ (часть 1)

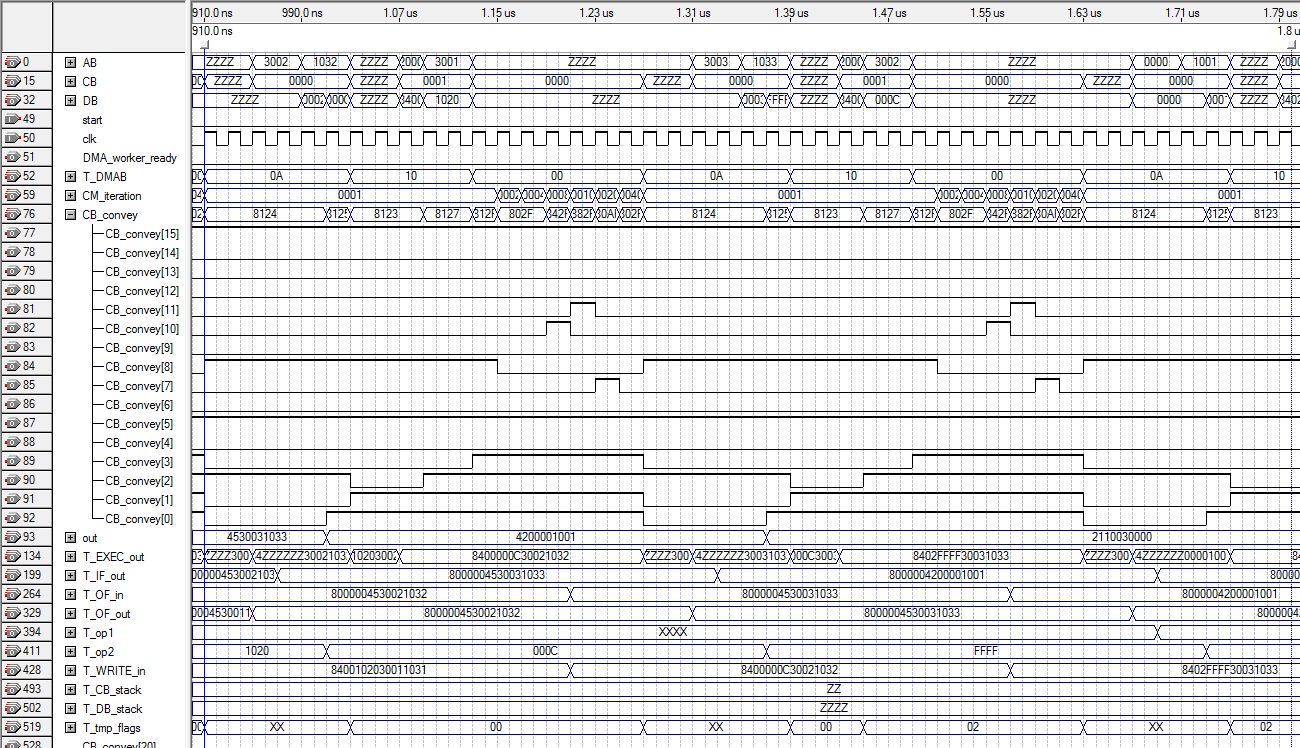


Рисунок 3.17 – Функциональное моделирование работы микро-ЭВМ (часть 2)

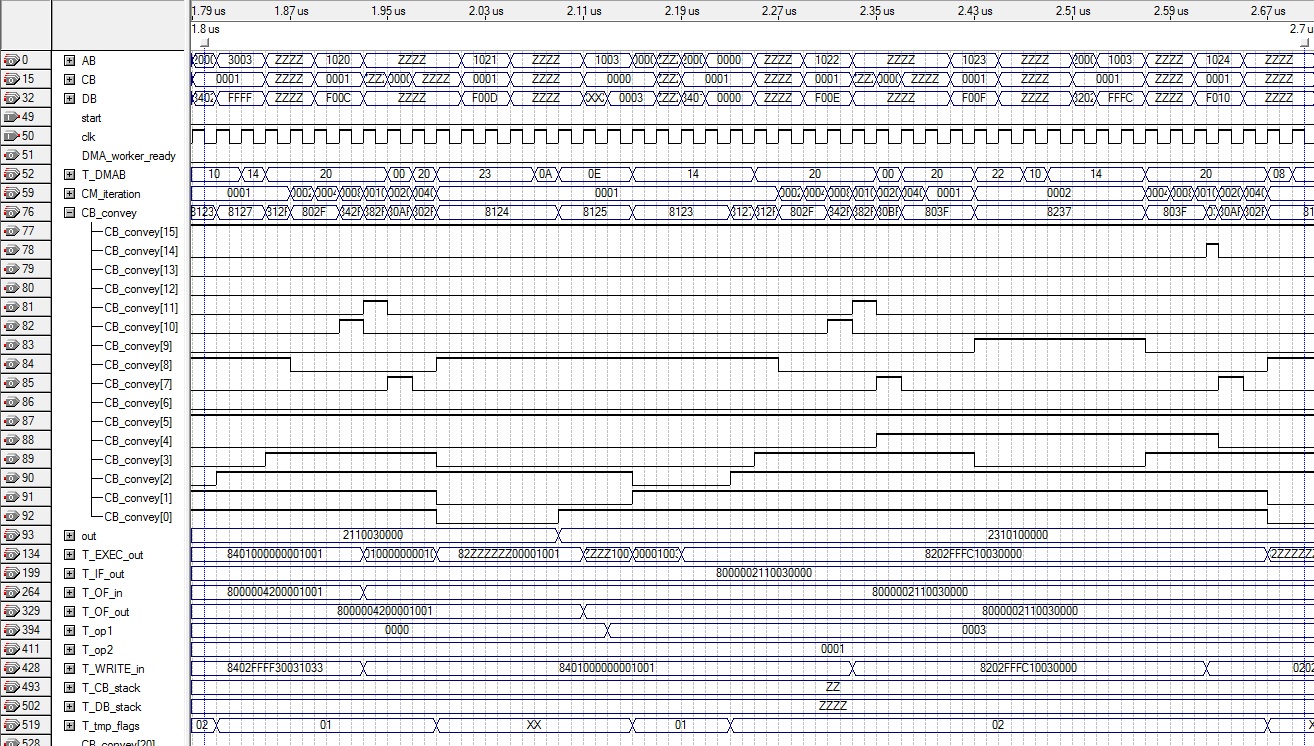
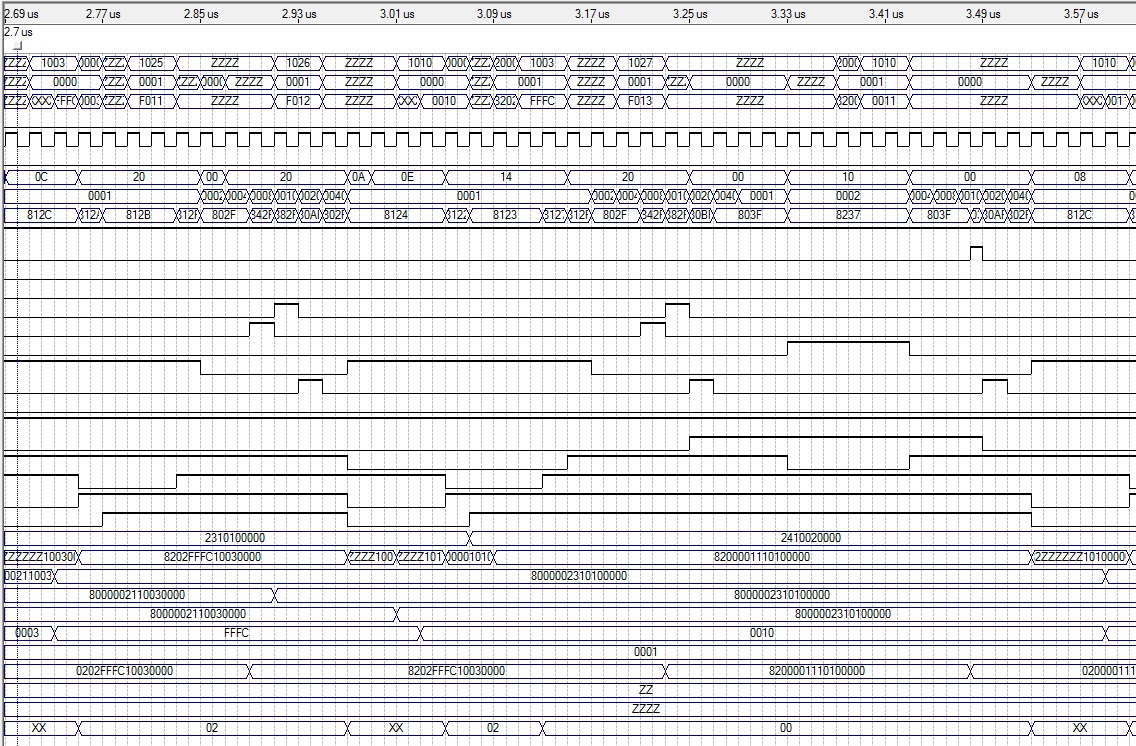


Рисунок 3.18 – Функциональное моделирование работы микро-ЭВМ (часть 3)

Рисунок 3.19 – Функциональное моделирование работы микро-ЭВМ (часть 4)

1. АНАЛИЗ И ОПТИМИЗАЦИЯ РАЗРАБОТАННОЙ МИКРО-ЭВМ

Главный прирост производительности в моей микро-ЭВМ дает трехуровневый конвейер. Все стадии конвейера были разработаны таким образом, чтобы занимать примерно одинаковое время и тем самым минимизировать простой. В идеальном случае при выполнении достаточно длинной программы трехуровневый конвейер должен ускорять работу устройства в три раза, однако, на деле не всё так просто.

Во-первых, между блоками существует зависимость из-за использования только одной общей шины данных и одной шины адреса. Поэтому блок извлечения операндов не может работать параллельно с блоком записи результата.

Во-вторых, некоторые последовательности операций могут генерировать конфликт чтения-записи. Разрешение таких конфликтов занимает значительное количество тактов процессора и резко замедляет работу всего устройства.

В-третьих, во время работы устройства некоторые условные переходы могут быть предсказаны неверно. Это повлечет за собой необходимость очистки конвейера и исправления адреса, что также является затратной операцией. Частично эту проблему решает предсказатель переходов. Он уменьшает количество ошибочных предсказаний, тем самым увеличивая общую скорость работы устройства.

В-четвертых, невозможно выровнять все стадии конвейера так, чтобы они занимали абсолютно одинаковое количество тактов. Поэтому будут присутствовать такие участки времени, когда работает только одна стадия конвейера, а все остальные уже готовы.

В ходе работы я исследовал возможные компоновки блоков в стадии и выбрал оптимальный из них. При исследовании я учитывал, что время работы каждого блока равно:

* Чтение инструкции (блок 1) – 5 тактов
* Извлечение операндов (блок 2) – 6 тактов
* Выполнение операции (блок 3) – 4 такта
* Запись результата (блок 4) – 6 тактов

Также я учитывал, что блок извлечения операндов и блок записи результата не могут работать параллельно по описанным выше причинам. В таблицах 3.4, 3.5 и 3.6 представлены варианты распределения блоков по стадиям конвейера. Так как самые длительные операции это извлечение операндов и запись результата, то, как видно из таблиц, перестановка блоков между стадиями не изменяет общую длительность. При любой группировке мы получаем длительность стадии 12 тактов. При последовательной обработке эти же действия заняли бы 21 такт. Отсюда получаем, что при отсутствии конфликтов чтения-записи и неверных предсказаний конвейер увеличивает производительность устройства примерно в 2 раза.

Таблица 3.4 – Объединение блоков в стадии (вариант 1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Стадия | Блок | Такты | | | | | | | | | | | |
| 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 3 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 4 |  |  |  |  |  |  |  |  |  |  |  |  |

Таблица 3.5 – Объединение блоков в стадии (вариант 2)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Стадия | Блок | Такты | | | | | | | | | | |  |
| 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 4 |  |  |  |  |  |  |  |  |  |  |  |  |

Таблица 3.6 – Объединение блоков в стадии (вариант 3)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Стадия | Блок | Такты | | | | | | | | | | |  |
| 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 3 |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |

Для реализации я выбрал второй вариант распределения из-за того, что в нем операции распределены наиболее равномерно.

Временный регистр флагов в блоке выполнения команды ускорил работу команд, зависящих от состояний флагов. Это избавило устройство управления от необходимости останавливать часть конвейера, когда выполняется операция, зависящая от состояния флагов.

КПДП также оказывает значительное влияние на производительность устройства, так как позволяет переносить данные из внешней памяти, не затрагивая устройство управления. Когда одновременно выполняется программа и происходит копирование данных из внешней памяти, то основные шины заняты около 80% времени, что свидетельствует о полном использовании ресурсов микро-ЭВМ.

ЗАКЛЮЧЕНИЕ

В ходе работы я реализовал полноценную микро-ЭВМ с использованием конвейера. При разработке я решал задачи по синхронизации блоков, продумывал алгоритмы устранения конфликтов чтения-записи и ошибочных переходов. Большое количество времени было потрачено на оптимизацию отдельных блоков и всей схемы в целом. В результате я создал микро-ЭВМ со следующими характеристиками:

1. Гарвардская архитектура.
2. ПЗУ объемом 0х1000 слов.
3. ОЗУ объемом 0х1000 слов.
4. 24 регистра общего назначения.
5. Стек, размером 8 слов.
6. Предсказатель переходов
7. КПДП
8. Конвейер

Набор команд реализованной микро-ЭВМ может быть расширен на 6 инструкций. Присутствует достаточное количество свободных адресов в общем адресном пространстве, на которые можно отобразить память любых синхронных подключенных устройств.