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);
* Арифметико-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.

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

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

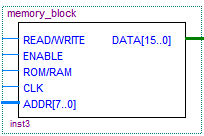


Рисунок 1. Условно-графическое обозначение блок памяти.

Как упоминалось выше блок памяти объединяет в себя 2 устройства – ПЗУ и ОЗУ.

Опишем сигналы, которые могут подаваться на данный блок:

Входные:

* READ/WRITE – выбор между чтением и записью;
* ENABLE – активация блока;
* ROM/RAM – выбор между ПЗУ и ОЗУ;
* CLK – тактирующий сигнал;
* ADDR – адрес блока памяти;
* DATA – данные для записи;

Выходные:

* DATA – данные, прочитанные из устройства

Реализация данного устройства довольно проста и представлена далее на рисунке 2. Внутри себя оно использует стандартные компоненты для Altera Quartus II 9.1 LPM\_ROM и LPM\_RAM, что соответствует ПЗУ и ОЗУ. Стоит также отметить что объединив ПЗУ и ОЗУ в один модуль мы тем самым исключили возможность одновременного их использования, однако это не требуется при реализации данного микро-ЭВМ.

Блок регистров общего назначения состоит из 16 16-разрядных регистров. Обычно данные регистры используются для хранения временных результатов.

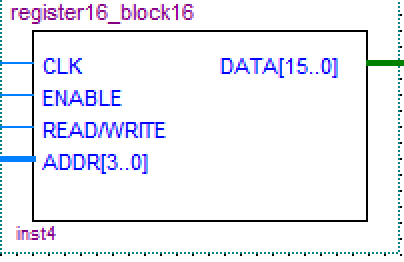


Рисунок 3. Условно-графическое обозначение блока РОН.

Опишем сигналы, входящие в данный блок:

Входные:

* ENABLE – активация блока;
* CLK – тактирующий сигнал;
* ADDR – адрес регистра;
* READ/WRITE – выбор режима чтения или записи.
* DATA – данные для записи в регистр.

Выходные:

* DATA – данные, прочитанные из устройства

Регистр начинает работу при подаче на него сигналов ENABLE и корректного адреса ADDR. Дальше в зависимости от режима READ/WRITE будет произведено чтение и выставление на шину DATA данных или запись данных в указанный регистр. Примечательно что все происходит за 1 такт, в отличии от блока памяти где требуется больше времени.

* 1. Устройство управления.

Как было сказано ранее устройство управления предназначено для организации работы всех блоков микро-ЭВМ.

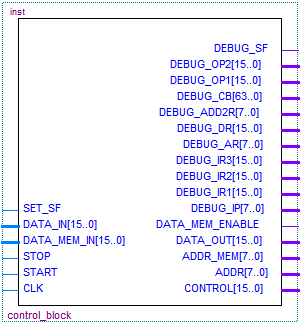


Рисунок 4. Условно-графическое обозначение устройство управления.

Опишем сигналы, входящие в данный блок:

Входные:

* ENABLE – активация блока;
* SET\_SF – установка флага S;
* DATA\_IN – вход шины данных;
* DATA\_MEM\_IN – вход шины команд;
* CLK – тактирующий сигнал;
* START – начало работы устройства;
* STOP – конец работы устройства.

Выходные:

* DATA\_OUT – выход шины команд;
* DATA\_MEM\_ENABLE – активация ПЗУ команд;
* ADDR\_MEM – адрес блока памяти в ПЗУ команд;
* ADDR – шина адреса;
* CONTROL – шина управления.

Стоит отметить что все выходы, начинающиеся с префикса DEBUG не несут в себе смысловой нагрузки и требуются только во время разработки устройства, поэтому не отражены в листинге сигналов.

Устройство управления содержит в себе различные регистры:

* IP – регистр, содержащий в себе адрес текущей исполняемой команды в блоке памяти
* IR[1-3] – регистры, используемые при выборке текущей команды из ПЗУ;
* OP[1-2] – регистры для адресов операндов;
* AR – регистр адреса, используемый для выставления адреса на шину адреса;
* DR – регистр данных, используемый для общения с внешней шиной данных;
* ADD2 – промежуточный регистр, используемый при выборки команды.

Также в устройстве управления содержится регистр флагов, представленный на рисунке 5. Данный регистр имеет соответствующие сигналы для сброса и установки флагов S, Z, J, а также для сброса всех флагов.

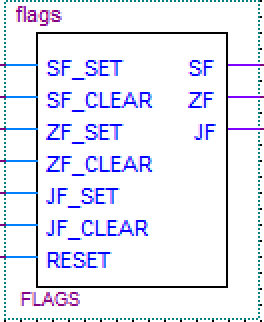


Рисунок 5. Условно-графическое обозначение регистра флагов.

Устройство управления состоит из большого количества логических блоков, которые выставляют необходимые нам последовательности сигналов на шину управления. Условно-графическое обозначение такого блока представлено на рисунке 6, а пример реализации блока представлен на рисунке 7.

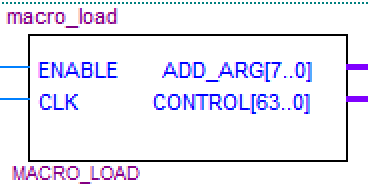


Рисунок 6. Условно-графическое обозначение логического блока устройства управления.

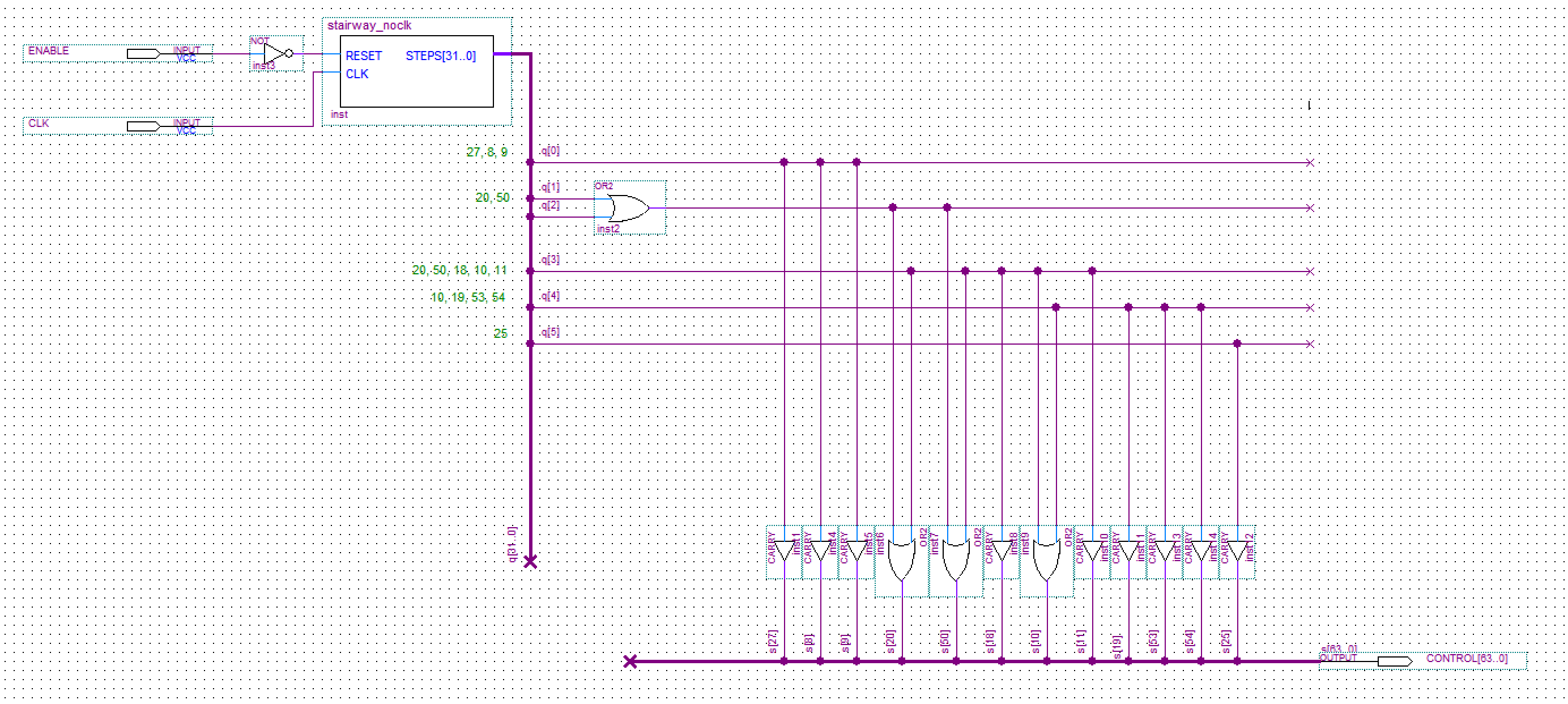


Рисунок 7. Пример внутренней реализации логического блока устройства управления.

В ходе разработки данного устройства была написана специальная программа на языке Ruby для определения необходимого количества логических элементов для построения очередного логического блока. Листинг программы и результат ее выполнения приведены ниже.

commands = <<COMM

27, 8, 9

20, 48

18, 10, 11, 20, 48

10, 19, 56

29, 8, 9

20, 19, 10, 48, 49

18, 10, 11, 20, 48

10, 19, 57

18, 10, 11, 62

27, 8, 9

20, 19, 10, 48, 49

25

COMM

puts commands.split("\n").flat\_map do |str|

str.split(",")

end.map(&:lstrip).each\_with\_object(Hash.new(0)) {|e, h| h[e] += 1}

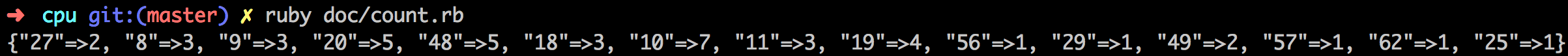


Рисунок 8. Результат выполнения работы программы.

* 1. Стек.

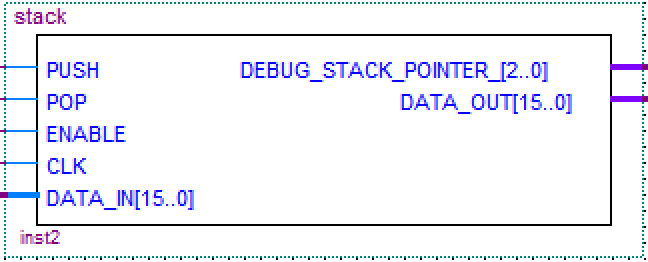


Рисунок 9. Условно-графическое обозначение стека.

Опишем сигналы, входящие в данный блок:

Входные:

* ENABLE – активация блока;
* PUSH – запись в стек;
* POP – извлечение из стека;
* DATA\_IN – вход шины данных;
* CLK – тактирующий сигнал;

Выходные:

* DATA\_OUT – выход шины данных;
  1. Арифметико-логическое устройство.

В состав АЛУ входят 2 регистра операндов, а также множество логических

блоков, соответствующих определенной операции.

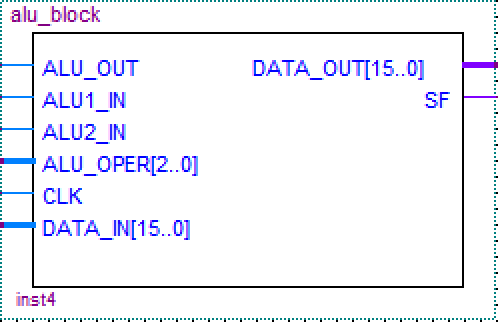


Рисунок 10. Условно-графическое обозначение АЛУ.

Опишем сигналы, входящие в данный блок:

Входные:

* ALU\_OUT – подключение внутренней шины данных к внешней;
* ALU1\_IN – запись операнда в первый регистр операндов;
* ALU2\_IN – запись операнда во второй регистр опренадов;
* ALU\_OPER – код номера операции блока АЛУ;
* DATA\_IN – вход шины данных;
* CLK – тактирующий сигнал.

Выходные:

* DATA\_OUT – выход шины данных;
* SF – выставление флага S.

Ниже представлен список номер операций:

Таблица 1.5 – Cписок операций АЛУ.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | АЛУ | Операнды | Код |
| 1 | SUB | 2 | 000 |
| 2 | NAND | 2 | 001 |
| 3 | NOR | 2 | 010 |
| 4 | ROL | 1 | 011 |
| 5 | ROR | 1 | 100 |

Для корректной работы АЛУ необходимо предварительно загрузить данные в регистре операндов. Стоит отметить что различные операции требуют наличие различного количества операндов. После загрузки операндов для получения результата нужно подать необходимый номер команды на вход ALU\_OPER соответствующий коду операции и сигнал ALU\_OUT.

Стоит отметить что появление результата на шине данных происходит сразу же после подачи сигнала ALU\_OPER.

Если во время выполнения операции SUB результатом является отрицательное число то АЛУ сгенерирует сигнал SF, который должен быть обработан устройством управления и в регистре флагов должен быть выставлен флаг S.

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.

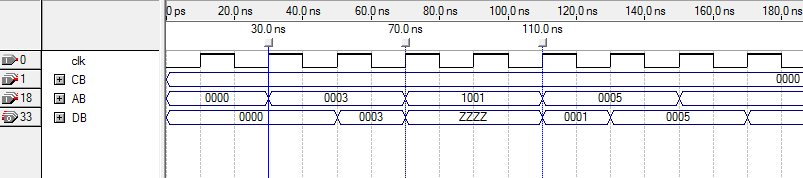


Рисунок 3.1 – Функциональное моделирование ПЗУ

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.

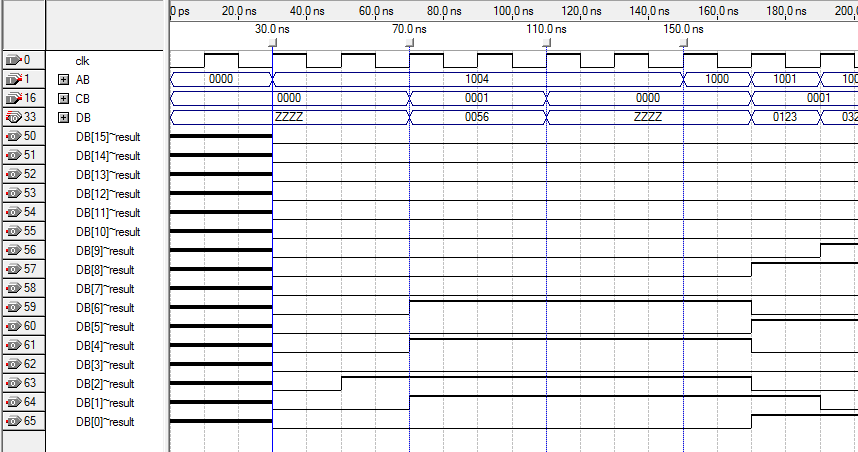


Рисунок 3.2 – Функциональное моделирование ОЗУ

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.

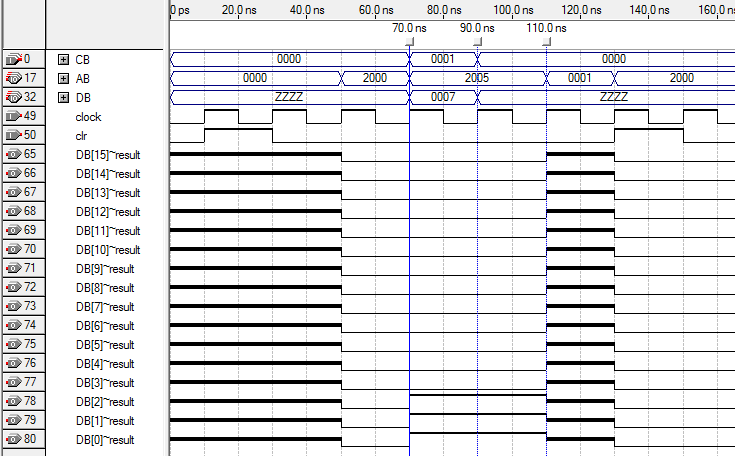


Рисунок 3.3 – Функциональное моделирование РОН

* 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.

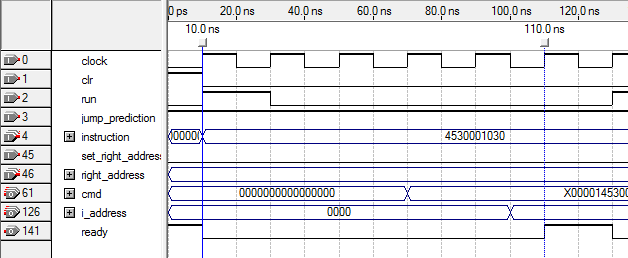


Рисунок 3.4 – Функциональное моделирование выборки инструкции

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.

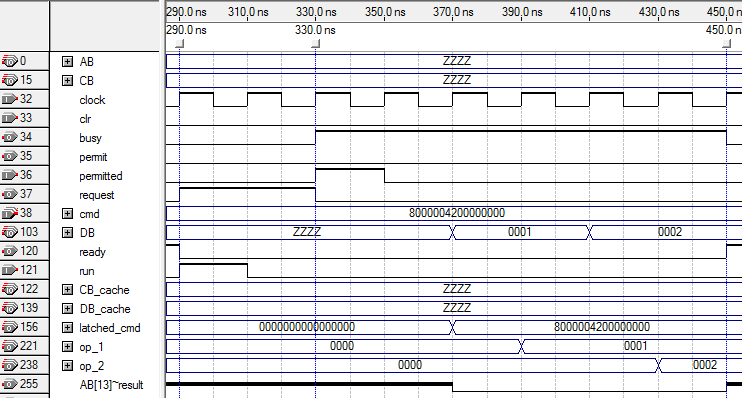


Рисунок 3.5 – Функциональное моделирование извлечения операндов

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).

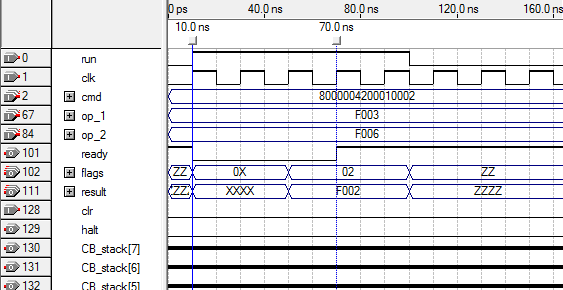
.

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.

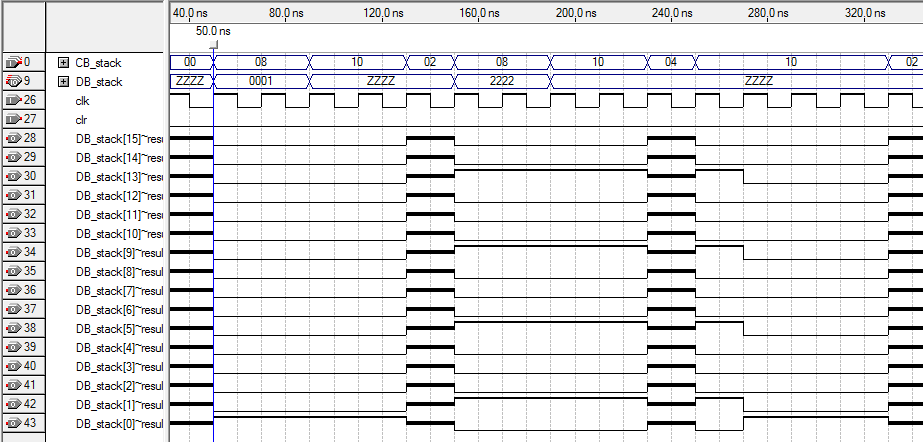


Рисунок 3.7 – Функциональное моделирование стека

* 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.

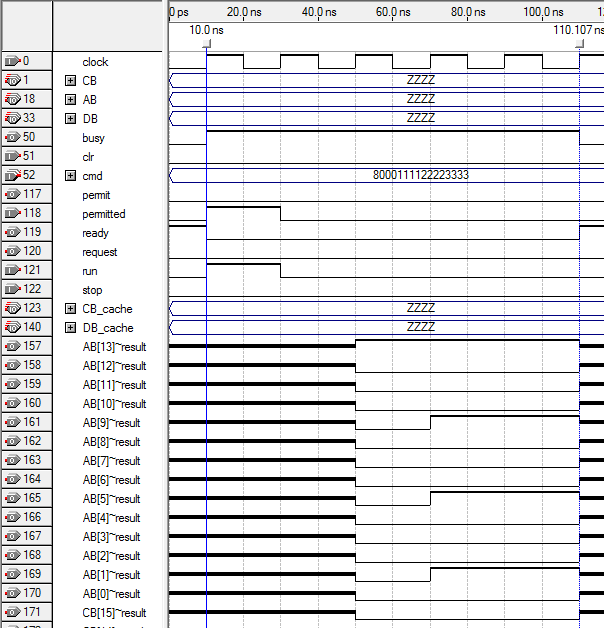


Рисунок 3.8 – Функциональное моделирование блока записи

* 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.

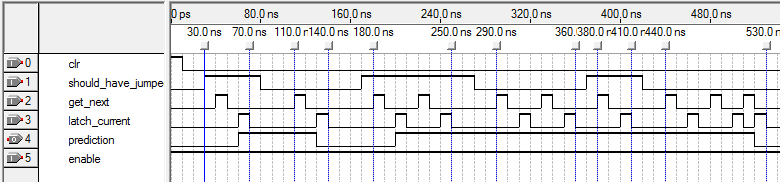


Рисунок 3.9 – Функциональное моделирование автомата А5

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.

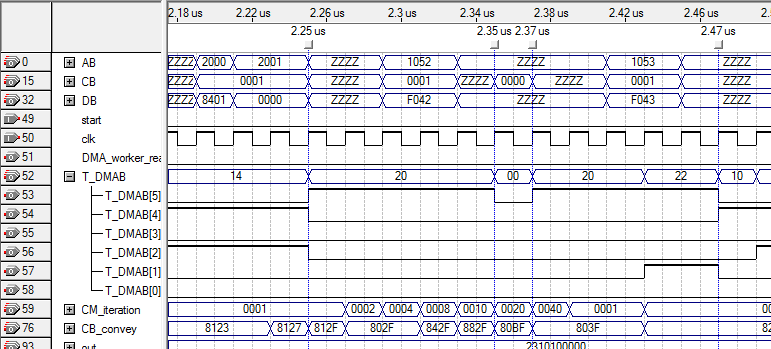


Рисунок 3.10 – Функциональное моделирование КПДП

* 1. Арбитр.

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

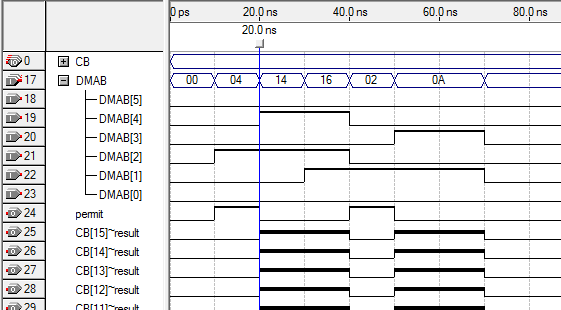


Рисунок 3.11 – Функциональное моделирование арбитра

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.

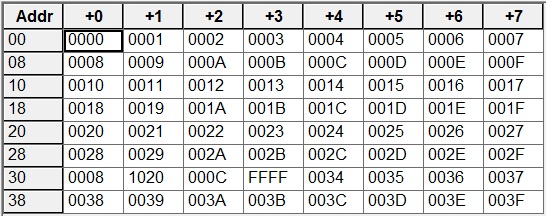


Рисунок 3.12 – Содержимое ОЗУ

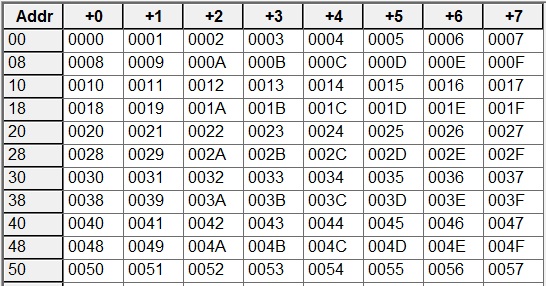


Рисунок 3.13 – Содержимое ПЗУ

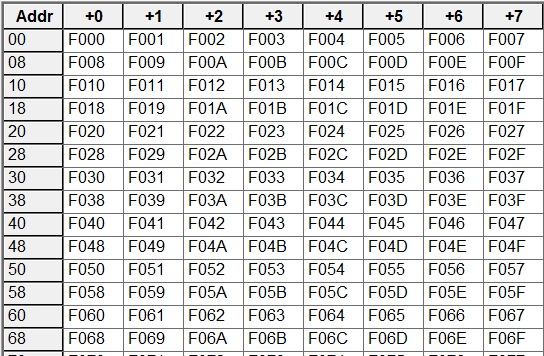


Рисунок 3.14 – Содержимое внешнего устройства

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.

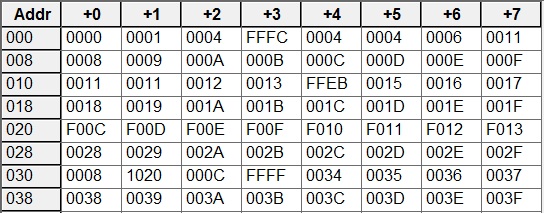


Рисунок 3.15 – Содержимое ОЗУ после выполнения программы

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.

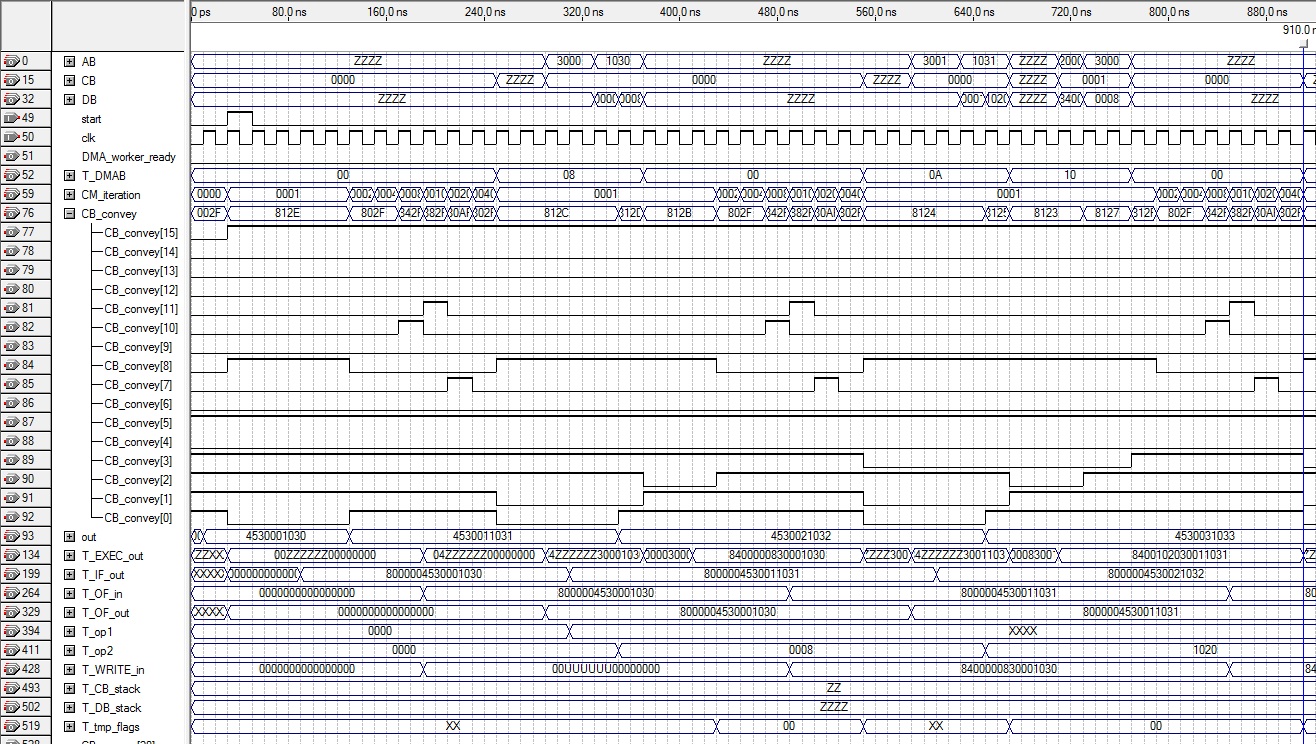


Рисунок 3.16 – Функциональное моделирование работы микро-ЭВМ (часть 1)

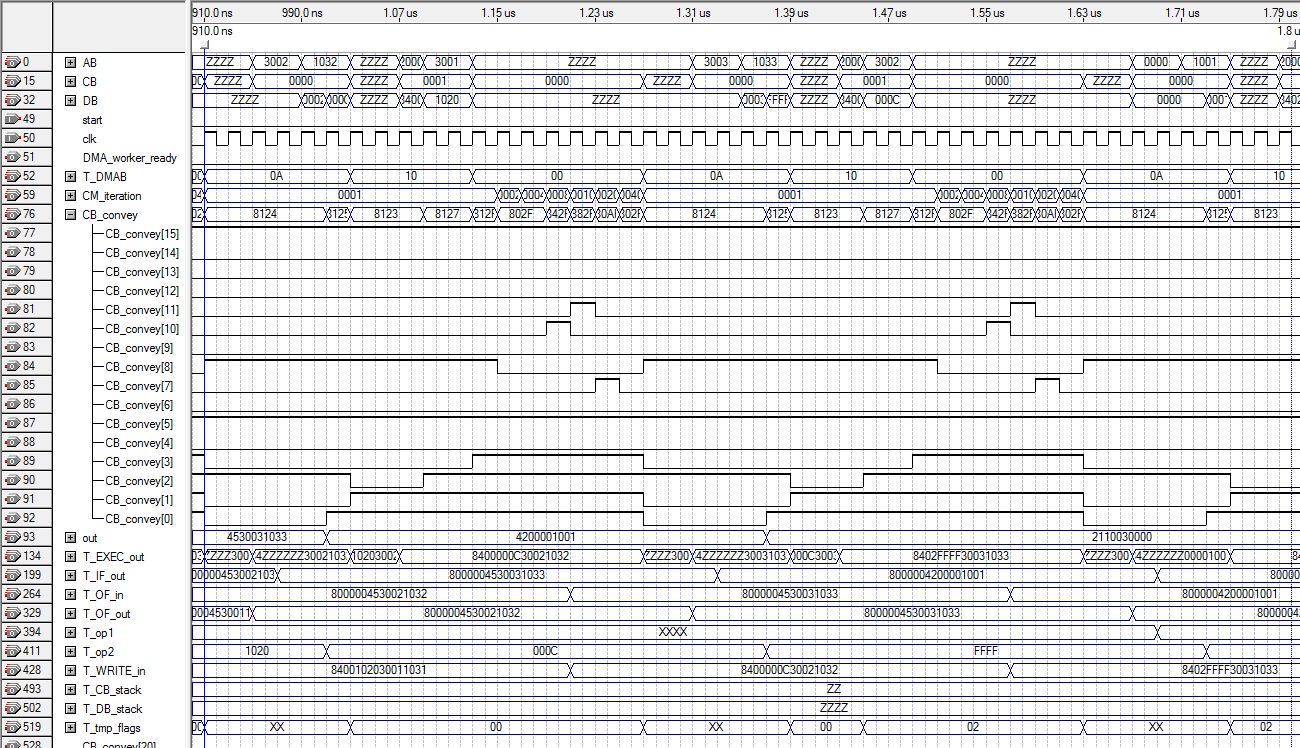


Рисунок 3.17 – Функциональное моделирование работы микро-ЭВМ (часть 2)

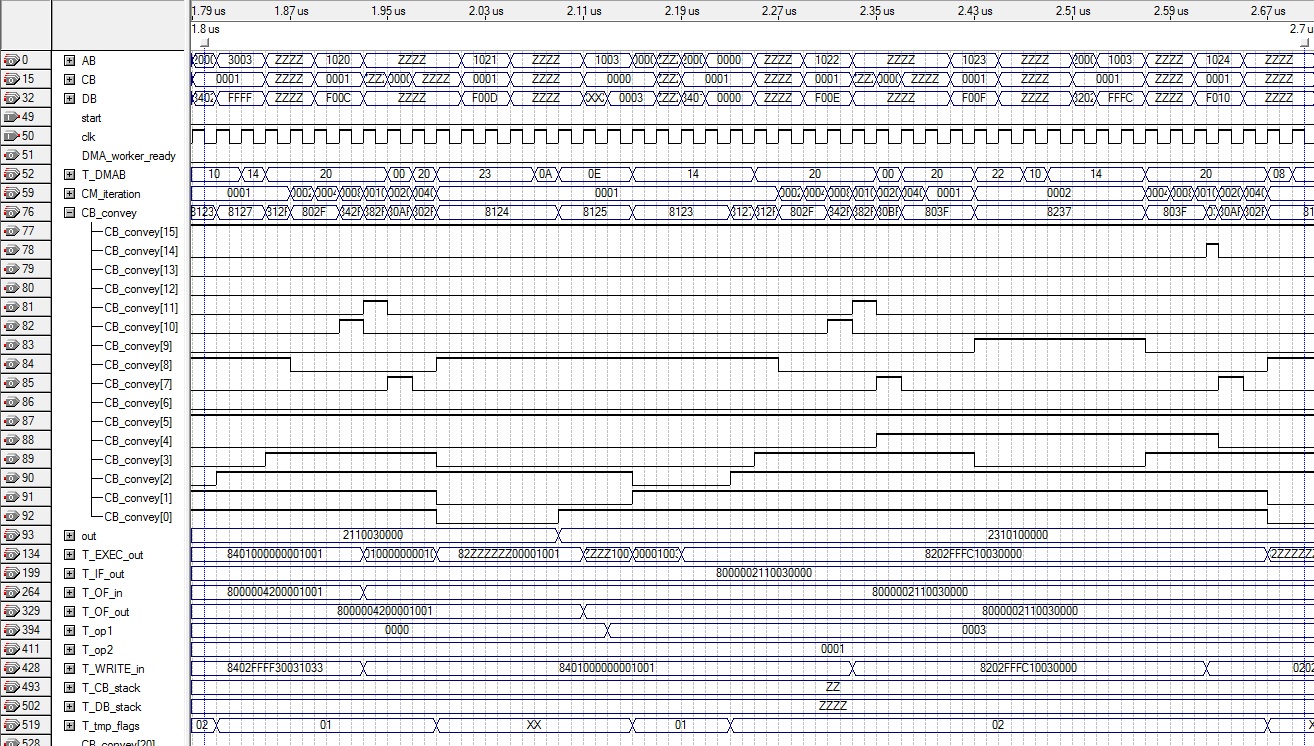
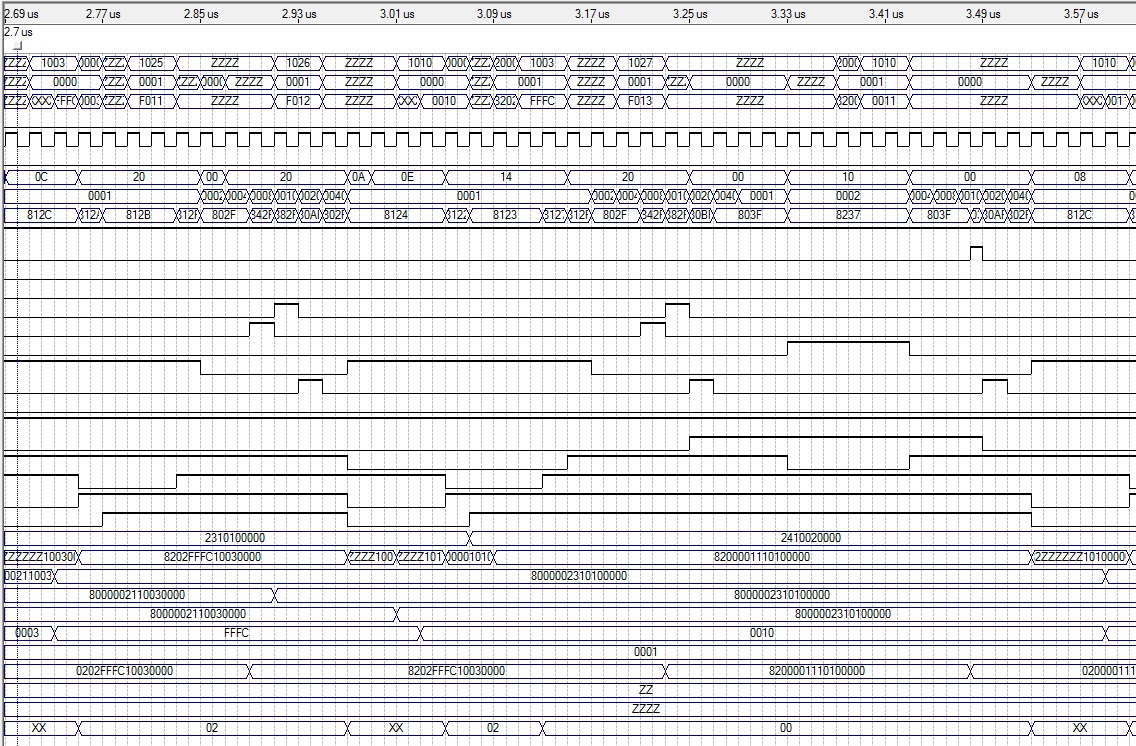


Рисунок 3.18 – Функциональное моделирование работы микро-ЭВМ (часть 3)

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.