СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4

**1** РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРОЭВМ 6

**1.1** Функциональный состав микроЭВМ 6

**1.2** Архитектура системы команд 6

**2** РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРОЭВМ 7

**2.1** Центральное устройство управления 7

**2.2** Арифметико-логическое устройство 10

**2.3** Регистры общего назначения 12

**2.4** Стековая память 12

**2.5** Арбитраж шин 14

**2.6** Система предсказания переходов 15

**3** ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ 16

**3.1** Функциональное моделирование АЛУ 16

**3.2** Функциональное моделирование стека 17

**3.3** Функциональное моделирование блока РОН 17

**3.5** Функциональное моделирование арбитража 18

**3.6** Функциональное моделирование предсказателя переходов 18

**3.7** Функциональное моделирование системы в целом 19

**4** АНАЛИЗ И ОПТИМИЗАЦИЯ РАЗРАБОТАННОЙ МИКРОЭВМ 22

ЗАКЛЮЧЕНИЕ 23

СПИСОК ЛИТЕРАТУРЫ 24

ВВЕДЕНИЕ

Цель данной курсовой работы – разработка микроЭВМ согласно заданному варианту.

В общем случае, ЭВМ – это устройство или система, которая способна выполнять какие-либо заданные операции: манипулирование данными, операции ввода вывода.

При проектировании архитектуры наиболее важной частью будет являться тип этой самой архитектуры. В моём случае этим типом является гарвардская архитектура. В этом типе архитектура команды и данные хранятся раздельно. Использование такого типа памяти, как правило, позволяет выиграть в скорости по сравнению с Принстонской архитектурой. Запись в область команд невозможна в принципе. Сделано это в целях безопасности: невозможно случайно записать что-то в область команд и испортить код.

По заданию мне необходимо использовать два типа памяти ОЗУ и ПЗУ. Мной было принято решение сделать ПЗУ для команд и ОЗУ для данных, что как раз удовлетворяет всем требованиям к памяти команд, которая не может быть изменена.

Адресации в данной схеме наблюдается такие как прямая и базовая со смещением, что позволяет указать в команде адрес в памяти или номер регистра, откуда возьмем адрес, плюс смещение, и получим нужный нам адрес, где лежат данные.

Сам ход выполнения выглядит следующим образом: выборка и декодирование команд, вычисление адресов, выборка операндов, выполнение и/или запись.

Децентрализованный арбитраж шин не предполагает наличие главного арбитра, связанного со всеми ведущими устройствами, а предполагает наличие нескольких линий, ответственных за установление текущего приоритетного устройства, которое будет само следить за доступностью шины. Эта схема не идеальна, так как приоритет будет всегда фиксированным.

Предсказатель позволяет угадывать, будет ли совершен условный переход или нет. В моем случае используется схема А4, шаблоном будет служить программный счетчик. На лекциях я узнал, что это не самый лучший вариант шаблона, так лучшие показатели выдает вариант глобальная таблица шаблонов плюс программный счетчик, однако и мой вариант достаточно приемлем.

В качестве кэш используется схема 2-way associative, которая сочетает в себе достоинства кэш с полностью ассоциативным отображением и кэш с прямым отображением. Данные с 1 в старшем бите могут попасть в один сет, с 0 – в другой сет и никак иначе.

В АЛУ есть 4 команды: арифметическая команда CMP, логические команды NXOR и NOTZ, сдвиговая операция ROL.

Разработка проекта велась в Altera Quartus II 9.1. В ней возможно проектирование схем любой сложности, есть базовые примитивы, можно промоделировать работу результата с помощью временных диаграмм сигналов. Есть мощные заготовки megafunctions, благодаря которым можно легко реализовывать очень сложные схемы. Кроме того, я получил достаточно большой опыт работы в этой программе на лабораторном практикуме.

**1** РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРОЭВМ

В этом разделе описан функциональный состав микроЭВМ и архитектура системы команд.

* 1. Функциональный состав микроЭВМ

Согласно заданию в состав микро-ЭВМ входит ряд устройств:

* ЦУУ – центральное устройство управления
* АЛУ – арифметико-логическое устройство
* Блок регистровой памяти
* Блок стековой памяти
* Блок кэш-памяти
* Система предсказания переходов
* Система арбитража шин

**1.2.** Архитектура системы команд

Таблица 1.1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Название | 1ый байт | | | | 2ой байт | | 3ий байт | | 4ый байт | |
| JMP | 0001 | | R | | Adr | | ------------------ | | ------------------ | |
| MOV Adr,Reg | 0010 | | R | | Adr | | R | Reg | ------------------ | |
| MOV Reg,Adr | 0011 | | R | | R | Reg | Adr | | ------------------ | |
| PUSH reg | 0100 | | Reg | | ------------------ | | ------------------ | | ------------------ | |
| POP reg | 0101 | | Reg | | ------------------ | | ------------------ | | ------------------ | |
| SRL | 1000 | A | | R | Adr | | Adr (Displ) | | R | Reg |
| SLL | 1001 | A | | R | Adr | | Adr (Displ) | | R | Reg |
| NAND | 1010 | A | | R | Adr | | Adr (Displ) | | R | Reg |
| XOR | 1011 | A | | R | Adr | | Adr (Displ) | | R | Reg |
| SUB | 1100 | A | | R | Adr | | Adr (Displ) | | R | Reg |

Примечания.

1. R – reserved, не используемые биты.
2. Adr – 8-разрядный адрес операнда в памяти.
3. Reg – 4-разрядный номер РОНа (всего их 12, это стоит учитывать).
4. A – 1-разрядный признак адресации для линейных команд (0 – прямая, 1 – относительная со смещением).
5. Displ – displacement, смещение относительно PC в случае А = 1.
6. 4ый байт используется только для линейных команд в случае А = 1 и сдержит номер регистра с дополнительным смещением.

Выбранная система команда имеет ряд преимуществ:

* При выборке коротких команд требуется меньше времени на этот процесс, чем при выборке длинных команд
* Так команды, занимают столько места в памяти, сколько им реально необходимо, следовательно, длина кода программы сокращается

Как недостаток стоит отметить, усложнение схемы управления программным счётчиком и выбора буферного регистра.

**2** РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРОЭВМ

В этом разделе описаны все устройства и блоки, реализующие их, с подробным описанием внутренней структуры и функционирования.

**2.1** Центральное устройство управления

Согласно заданию необходимо было реализовать ЦУУ, реализующее весь процесс обработки команд программы. Для данного варианта указаны следующие требования к архитектуре ЭВМ:

* Архитектура – гарвардская
* Разрядность шин адреса/данных – 8/8 бит
* Память – запись синхронная, чтение асинхронное (без буферизации)
* Адресация – прямая + относительная со смещением

Память команд реализована на блоке памяти ОЗУ, память данных на блоке ОЗУ. Буферы команд, регистр флагов, буферы операндов для АЛУ реализованы с помощью регистров на основе D-триггеров. Контроль этапов и циклов выполнения реализован на связках счётчик + дешифратор. Для вычисления исполнительного адреса при относительной адресации со смещением используется параллельный сумматор. Вся управляющая логика реализована с помощью логических функций. Контроль доступа к шине осуществляют буферы с тремя состояниями.

2.1.1 Блок CU

В данном блоке реализована память ЭВМ, программный счётчик, обеспечено взаимодействие с РОНами, стеком, АЛУ, дешифрация и выполнение команд, не реализуемых в АЛУ, вычисление исполнительных адресов, выборка операндов для команд и сохранение результата АЛУ.

Описание входных сигналов блока CU.

Таблица 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Ширина | Направление | Назначение |
| clk | 1 | In | Синхросигнал |

Этот блок является верхним в иерархии проекта, поэтому он только получает синхросигнал от генератора тактов, а все управляющие сигналы являются внутренними.

Описание внутренних сигналов блока CU.

Шина ST[2..0] – текущий этап обработки команды.

Таблица 2.2

|  |  |  |
| --- | --- | --- |
| Название | Назначение | Формирование |
| ST0 | Этап выборки и декодирования команды | eq0 |
| ST1 | Этап вычисления адресов операндов, их выборки и выполнения команды (для нелинейных команд) | eq1 |
| ST2 | Этап выполнения команды и сохранения результата (только для команд АЛУ) | eq2 |

Примечание. eqi – i-ый выход дешифратора счётчика этапов.

Шина W[4..0] – текущее значение счётчика циклов, формируется на дешифраторе счётчика циклов. Счётчики этапов и циклов – фактически конечные автоматы, управляющие процессом обработки команд. Формирование сигнала инкрементирования программного счётчика: ST0∙(W0+((C2+C3+D\_0[7]+C1)∙W1)+((C2+C3+D\_07)∙W2)+(D\_0[3]∙D\_0[7]∙W3)). Формирование сигнала буферизации первого байта команды: W0∙ST0. Формирование сигнала буферизации второго байта команды: W2∙ST0∙(C1+ +D\_0[7]+C2+C3). Формирование сигнала буферизации третьего байта команды: W3∙ST0∙( D\_0[7]+C2+C3). Формирование сигнала буферизации четвёртого байта команды: W4∙ST0∙ D\_0[7]∙ D\_0[3]. Формирование сигнала decode\_end (окончание этапа выборки и декодирования команды): ST0∙(( D\_0[3]∙ D\_0[7]∙W3)+( D\_0[3]∙ ∙D\_0[7]∙W4)+(((C0+C4+C5) ∙W1)+(C1∙W2)+((C2+C3) ∙W3)))). Шина D\_0[7..0] – первый байт команды. Шина D\_1[7..0] – второй байт команды. Шина D\_2[7..0] – третий байт команды. Шина D\_3[7..0] – четвёртый байт команды. Шина C[5..0] – результат декодирования кода операции первого байта команды (D\_0[7..4]).

Таблица 2.3

|  |  |  |
| --- | --- | --- |
| Название | Назначение | Формирование |
| JMP | К. безусловного перехода | C1∙ST1 |
| MOV\_AR | К. перемещения слова из РОНа в память | C2∙ST1 |
| MOV\_RA | К. перемещения слова из памяти в РОН | C3∙ST1 |
| PUSH | К. занесения регистра в стек | C4∙ST1 |
| POP | К. извлечения регистра из стека | C5∙ST1 |
| op\_end | Окончание обработки нелинейной операции | (JMP∙W1)+(MOV\_AR∙W1)+ (MOV\_RA∙W2)+ (PUSH∙W2)+ (POP∙W3)+(C0∙W1) |
| sno | Начало обработки линейной операции в АЛУ | LOP∙W3 |
| sko | Окончание обработки линейной операции в АЛУ | АЛУ |
| MR | Адресация ячейки памяти из третьего байта команды (для чтения) | MOV\_RA∙(W0+W1) |
| MW | Адресация ячейки памяти из второго байта команды (для записи) | MOV\_AR∙W0 |
| OP\_1 | Адресация ячейки памяти (для чтения, либо сохранения), содержащей первый операнд линейной операции | (W0∙ST1∙D\_0[7])+sko |
| OP\_2d | Адресация ячейки памяти (для чтения), содержащей второй операнд линейной операции при прямой адресации | dir\_adr∙W1 |
| OP\_2r | Адресация ячейки памяти (для чтения), содержащей второй операнд линейной операции при относительной адресации со смещением | rel\_adr∙W1 |
| WE | Запись данных в память | MW+sko |
| OE | Чтение данных из памяти | RW+LOP\_SEL |
| dir\_adr | Признак прямой адресации | LOP∙! D\_0[3] |
| rel\_adr | Признак относительной адресации | LOP∙ D\_0[3] |
| LOP | Признак линейной операции | D\_0[7]∙ST1 |
| LOP\_SEL | Выборка операндов линейной операции | LOP ∙(W2+W1) |
| RR | Адресация РОНа (для чтения и записи в память) | MOV\_AR∙W0 |
| RW | Адресация РОНа (для записи слова из памяти), пропуск данных для записи, команда записи для блока РОНов | MOV\_RA∙W1 |
| RRA | Адресация РОНа (для чтения из него дополнительного смещения при относительной адресации) | rel\_adr∙W1 |
| PW | Адресация РОНа (для записи слова из стека с проверкой на наличие данных), пропуск данных для записи, команда записи для блока РОНов | POP∙W1∙!empty |
| PTS | Занесение данных в стек | PUSH∙W0 |

Формирование сигнала exe блока стека: (POP∙W0)+PTS. Формирование команды для блока РОНов: PW+RW. Формирование сигнала записи в буферный регистр первого операнда: LOP∙W1. Формирование сигнала записи в буферный регистр второго операнда: LOP∙W2. Формирование остальных сигналов очевидно и не вызывает вопросов.

2.1.2 Описание процессов обработки отдельных команд

Выборка требует столько тактов, сколько байтов в команде.

JMP:

* Выборка
* Занесение нового адреса в программный счётчик
* Окончание обработки

MOV\_RA:

* Выборка
* Выборка слова из памяти
* Запись слова в РОН
* Окончание обработки

MOV\_AR:

* Выборка
* Выборка слова из РОНа и запись его в память
* Окончание обработки

PUSH:

* Выборка
* Выборка слова из РОНа и занесение его в стек
* Окончание обработки

POP:

* Выборка
* Подача команды POP на стек
* Выборка слова из стека и запись в РОН
* Окончание обработки

Линейные операции:

* Выборка
* Выборка первого операнда из памяти
* Запись первого операнда в буфер, выборка второго операнда из памяти
* Запись второго операнда в буфер
* Подача сигнала sno
* При получении сигнала sko - запись результата в память, окончание обработки

**2.2** Арифметико-логическое устройство

Согласно заданию необходимо было реализовать АЛУ, реализующие заданные арифметические и логические команды. Для данного варианта указан следующий набор команд:

* Арифметические: SUB
* Логические: XOR, NAND
* Сдвиговые: SLL, SRL

Для реализации SUB использован вычитатель, с входными и выходными сигналами заёма, для XOR и NAND соответствующие логические элементы, для SRL и SLL сдвиговые регистры и счетчик количества сдвигов с компаратором и управляющей логикой. Также реализована управляющая логика для обеспечения процесса обработки команд и генерации выходных сигналов.

2.2.1 Блок ALU

Данный блок реализует арифметико-логическое устройство с заданным набором команд, генерирует сигналы для регистра флагов.

Описание входных и выходных сигналов блока ALU.

Таблица 2.4

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Ширина | Направление | Назначение |
| clk | 1 | In | Синхросигнал |
| cin | 1 | In | Входной сигнал заёма |
| SNO | 1 | In | Сигнал начала обработки |
| linear\_cop | 3 | In | Код операции |
| A | 8 | In | Первый операнд |
| B | 8 | In | Второй операнд |
| DataOut | 8 | Out | Результат операции |
| SKO | 1 | Out | Сигнал конца обработки |
| cout | 1 | Out | Выходной сигнал заёма (флаг переноса) |
| zout | 1 | Out | Сигнал флага нуля |

Описание внутренних сигналов блока ALU. Шина SH[1..0] – отделяет команды сдвига от остальных.

Таблица 2.5

|  |  |  |
| --- | --- | --- |
| Название | Назначение | Формирование |
| SH0 | Команда сдвига | eq0+eq1 |
| SH1 | Остальные команды | eq2+eq3+eq4 |

Примечание. eqi – i-ый выход дешифратора кода операции.

Шина S[1..0] – этапы обработки операций сдвига.

Таблица 2.6

|  |  |  |
| --- | --- | --- |
| Название | Назначение | Формирование |
| S0 | Занесение в регистры сдвига первого операнда, в счётчик - 3 младших разряда второго | !S1 |
| S1 | Выполнение сдвига | q[0] – выход счётчика этапов |

Сигнал zout формируется как результат сравнения результата операции с 0. В соответствии с поступившим кодом операции мультиплексор, выбирает соответствующий результат и подает его на выход DataOut. Сигнал shift\_end формируется как результат сравнения счётчика сдвигов с нулём. Формирование выходного сигнала SKO (сигнализирует ЦУУ, что обработка окончена и результат можно сохранять в память): (SH1∙S1) + (SH0∙S1∙shift\_end).

**2.3** Регистры общего назначения

Согласно заданию необходимо было реализовать блок регистровой памяти с возможностью чтения и записи данных. Для данного варианта указаны следующие параметры:

* Количество РОНов - 12
* Разрядность – 8 бит

Для реализации РОНа используется регистр на основе D-триггеров,

для выбора элемента памяти при чтении – мультиплексор, при записи – дешифратор. Для отключения от шин используются буферы с тремя состояниями.

2.3.1 Блок RONs

Данный блок реализует блок регистровой памяти с заданными параметрами, обрабатывает команды чтения/записи. Описание входных и выходных сигналов блока RONs.

Таблица 2.7

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Ширина | Направление | Назначение |
| clk\_in | 1 | In | Синхросигнал |
| read\_write | 1 | In | Команда для памяти (0 – чтение, 1 - запись) |
| ron\_num | 4 | In | Номер РОНа, для которого будет выполняться команда |
| dataIn | 8 | In | Данные для записи |
| DataOut | 8 | Out | Данные при чтении |

При чтении элемента, мультиплексор выбирает данные из нужной ячейки и выставляет их на шину данных. При записи дешифратор включает сигнал тактирования для соответствующей ячейки, и данные с шины записываются в неё по фронту синхросигнала.

**2.4** Стековая память

Согласно заданию необходимо было реализовать блок стековой памяти с реализацией команд занесения в стек и выборки из стека. Для данного варианта указаны следующие параметры:

* Количество элементов памяти - 11
* Направление роста - вниз

Для реализации элемента стековой памяти используется регистр на основе D-триггеров, указатель стека реализован на счётчике с возможностью счёта как вверх, так и вниз, выбор элемента памяти для записи осуществляет дешифратор, для чтения - мультиплексор. Также реализована управляющая логика для обработки ситуаций выборки из пустого стека и записи в заполненный стек и идентификации команд.

2.4.1. Блок Stack

Данный блок реализует блок стековой памяти с заданными параметрами, обрабатывает команды занесения и выборки.

Описание входных и выходных сигналов блока Stack.

Таблица 2.8

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Ширина | Направление | Назначение |
| clk | 1 | In | Синхросигнал |
| push\_pop | 1 | In | Команда для стека (0 – чтение, 1 - запись) |
| exe | 1 | In | Сигнал подтверждения команды, подаётся одновременно с командой |
| DI | 8 | In | Данные для записи |
| DO | 8 | Out | Данные при чтении |
| empty | 1 | Out | Сигнал, что стек пуст |

Описание внутренних сигналов блока Stack.

Таблица 2.9

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Ширина | Назначение | Формирование |
| push | 1 | Разрешение записи и инкрементирования счётчика | exe∙cmd∙!D1 |
| pop | 1 | Разрешение декрементирования счётчика | exe∙!cmd∙!D0 |
| D | 11 | Результат дешифрации указателя стека | Номер РОНа, для которого будет выполняться команда |

Для выбора направления счёта используется входной сигнал cmd. Для записи данных в стек достаточно 1го такта, для чтения необходимо 2. Сигналы exe, cmd, D0 буферизируются, таким образом, данные пропускаются на выходной пин S\_DO на следующем такте после подачи команды. Также буферизированный сигнал D0 поступает на выход empty.

**2.5** Арбитраж шин

Согласно заданию необходимо было реализовать арбитра шины и продемонстрировать его работу. Для данного варианта указаны следующие параметры:

* Вид арбитража – децентрализованный
* Вид подключения – параллельный

Так как алгоритма смены приоритетов не было указано, выбраны статические приоритеты для устройств, использующих шину. Данный вид арбитража предполагает наличие в каждом ведущем устройстве схемы контроля доступа к шине, а тип подключения – каждое устройство подключено ко всем остальным.

2.5.1 Блок Devices

Данный блок реализует ведущие устройства в количестве 4 штук. Каждое устройство, представленное счётчиком, через индивидуальное количество тактов генерирует запрос шины и в случае предоставления шины выставляет на неё свои данные, представленные порядковым номером устройства.

Описание входных и выходных сигналов блока Devices.

Таблица 2.22

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Ширина | Направление | Назначение |
| clk | 1 | In | Синхросигнал |
| grant0 | 1 | In | Предоставление шины устройству 1 |
| grant1 | 1 | In | Предоставление шины устройству 2 |
| grant2 | 1 | In | Предоставление шины устройству 3 |
| grant3 | 1 | In | Предоставление шины устройству 4 |
| request0 | 1 | Out | Запрос шины устройством 1 |
| request1 | 1 | Out | Запрос шины устройством 2 |
| request2 | 1 | Out | Запрос шины устройством 3 |
| request3 | 1 | Out | Запрос шины устройством 4 |
| dataout0 | 2 | Out | Данные устройства 1 |
| dataout1 | 2 | Out | Данные устройства 2 |
| dataout2 | 2 | Out | Данные устройства 3 |
| dataout3 | 2 | Out | Данные устройства 4 |

Описание внутренних сигналов блока Devices.

Таблица 2.23

|  |  |  |
| --- | --- | --- |
| Название | Ширина | Назначение |
| data0 | 2 | Данные устройства 1 |
| data1 | 2 | Данные устройства 2 |
| data2 | 2 | Данные устройства 3 |
| data3 | 2 | Данные устройства 4 |

2.5.2 Блок Bus

Данный блок реализует шину, которую пытаются захватить устройства блока Devices. Для каждого устройства присутствует простейшая схема арбитража на логических элементах. Устройство из блока Devices, получившее доступ к шине выставляет на неё данные, которые получает ведомое устройство, представленное выходным пином.

Описание входных и выходных сигналов блока Bus.

Таблица 2.24

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Ширина | Направление | Назначение |
| clk | 1 | In | Синхросигнал |
| outdata | 2 | Out | Полученные данные |

Описание внутренних сигналов блока Bus.

Таблица 2.25

|  |  |  |
| --- | --- | --- |
| Название | Ширина | Формирование |
| r0 | 1 | Соответствует сигналу request0 блока Devices |
| r1 | 1 | Соответствует сигналу request1 блока Devices |
| r2 | 1 | Соответствует сигналу request2 блока Devices |
| r3 | 1 | Соответствует сигналу request3 блока Devices |
| g0 | 1 | r0 |
| g1 | 1 | !r0∙r1 |
| g2 | 1 | !r0∙!r1∙r2 |
| g3 | 1 | !r0∙!r1∙!r2∙r3 |

Сигналы g0,g1,g2,g3 – сигналы предоставления шины 1му, 2му, 3му, 4му устройствам соответственно. Функционирование системы вопросов не вызывает, поэтому детально не рассматривается.

**2.6** Система предсказания переходов

Согласно заданию необходимо было реализовать систему для предсказания условных переходов и продемонстрировать её работу. Для данного варианта указаны следующие параметры:

* Тип автомата – А2
* Количество бит для указания элементов PHT – 3
* Шаблон – сложение РС и GHR по модулю 2

Для реализации автомата А2 необходим сдвиговый регистр размерностью 2 бит. Так как для адресации элементов PHT доступно 3 бит, следовательно, в PC и GHR используются младшие 3 бит, и в таблице PHT будет 8 элементов.

2.6.1. Блок Predictor

Данный блок реализует всю систему предсказания переходов, имитирует поступление команды перехода и результат её выполнения, а также делает предсказание, на основании состояния автомата, выбранного элемента PHT.

Описание входных и выходных сигналов блока Predictor.

Таблица 2.26

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Ширина | Направление | Назначение |
| clk | 1 | In | Синхросигнал |
| PC | 3 | In | Младшие разряды счётчика команд |
| result | 1 | In | Результат выполнения команды УП |
| prediction | 1 | Out | Предсказание на основе состояния автомата |

В схеме присутствует 3-разрядный сдвиговый регистр GHR, при получении результата команды УП он сдвигается влево и на освободившееся место записывается результат выполнения последней команды УП (1 – переход был, 0 – перехода не было). Для выбора элемента PHT сигналы PC[2..0] и GHR[2..0] складываются по модулю 2 и формируют сигнал PHT\_ADR[2..0]. Сигнал PHT\_ADR[2..0] с помощью мультиплексора выбирает соответствующий сдвиговый регистр (автомат А2) и на основе его состояния (сигнал ST[1..0]) делает предсказание с помощью логической функции !ST0∙!ST1. Также, выбрав дешифратором нужный элемент PHT в него заносится результат выполнения предыдущей команды УП таким же образом, как и в случае с GHR.

**3** ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ

В этом разделе я покажу моделирование разработанных блоков, а именно временные диаграммы, сгенерированные в программе Quartus 9.1.

**3.1** Функциональное моделирование АЛУ

Временная диаграмма теста арифметико-логического устройства представлена на рисунке 3.1.

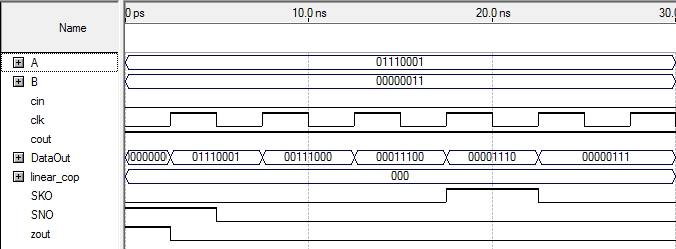


Рисунок 3.1 – Временная диаграмма АЛУ

С помощью линии linear\_cop задается типа операции. Линия cin отвечает за доставку флага заема для операции SUB. Линия cout показывает возникновение заема. Шины A[7..0] и B[7..0] нужны для передачи операндов. По выходной шине DataOut[7..0] выдается результат команд, входная линия SNO и выходная линия SKO соответственно показывают сигнал начала и конца обработки, выходная линия zout – флаг нуля.

Здесь виден результат выполнения операции SRL.

**3.2** Функциональное моделирование стека

Временная диаграмма работы стека представлена на рисунке 3.2.

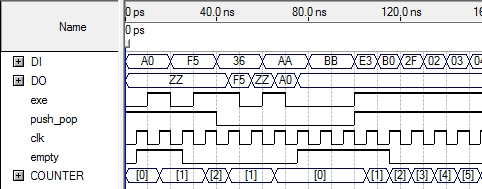


Рисунок 3.2 – Временная диаграмма работы стека.

Здесь видно, как при включении exe в стек заносятся 2 значения и после извлекаются. После этого производится операция push до полного заполнения стека.

На рисунке 3.3 показано продолжение моделирования.

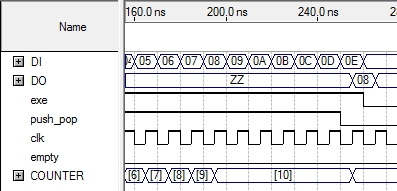


Рисунок 3.3 – Временная диаграмма работы стека.

Здесь видно, как стек полностью заполняется и лишние данные игнорируются. Затем извлекается последнее значение в стеке.

**3.3** Функциональное моделирование блока РОН

Временная диаграмма работы блока регистров общего назначения представлена на рисунке 3.4

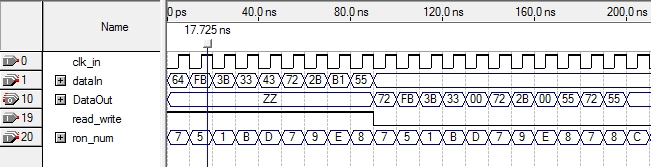


Рисунок 3.4 – Временная диаграмма блока РОН

Блок РОН представлен 12 регистрами (согласно заданию). На данной диаграмме видно, как по очереди происходит запись в разные регистры, затем из них производится считывание.

Сигналом read\_write управляется режим работы (0 - чтение, 1 - запись).

В целом блок достаточно тривиален и дополнительно объяснения не требует.

**3.4** Функциональное моделирование арбитража

Временная диаграмма работы арбитража представлена на рисунке 3.5. В этой схеме есть эмуляция четырех устройств, которые имитируют некоторую вычислительную деятельность. Шину занимает устройство с наивысшим приоритетом.

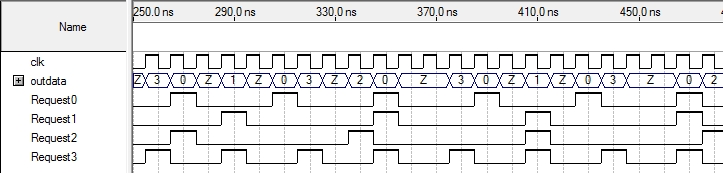


Рисунок 3.5 – Временная диаграмма работы арбитража

**3.5** Функциональное моделирование предсказателя переходов

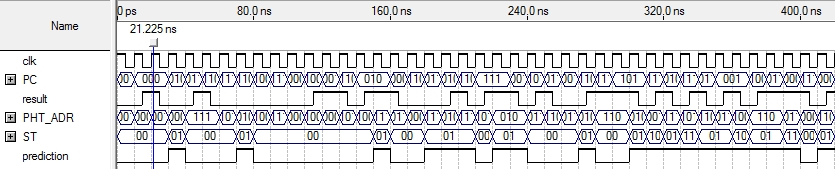


Рисунок 3.6 – Временная диаграмма работы предсказателя переходов

На рисунке 3.6 можно увидеть, как на линии prediction появляются предсказания. В данном случае предсказатель 2 раза предположил, что перехода не будет. Но в дальнейшем точность увеличивается. Вообще должно довольно много времени, прежде чем накопится достаточно истории для того, чтобы совершить точное предсказание. Это называется «холодный старт». В реальных системах используются дополнительные оптимизации работы предсказателя для более точной работы.

**3.6** Функциональное моделирование системы в целом

Процессор представляет собой, как бы, черный ящик. Мы подаем что-то на вход и ожидаем что-то на выходе. Поэтому сама по себе временная диаграмма почти ничего не покажет. На рисунке 3.7 представлена временная диаграмма работы всей системы.

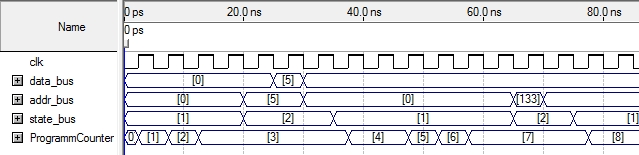


Рисунок 3.7 – Временная диаграмма работы всей системы

Процессор сам выполняет весь код, поданный на его вход, извне требуется минимальное вмешательство.

Программный код, выполняемый процессором, находится в памяти программ. Содержимое памяти программ показано на рисунке 3.8.

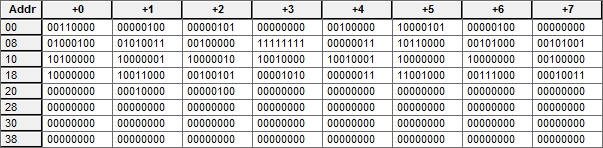


Рисунок 3.8 – Содержимое памяти программ

В таблице 3.1 дано описание того, что означает каждая ячейка

Таблица 3.1 – Описание содержимого памяти программ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Числовая запись команды | | | | Мнемоническая запись |
| 1-й байт | 2-й байт | 3-й байт | 4-й байт |
| 00110000 | 00000100 | 00000101 |  | mov R4,5h |
| 00000000 |  |  |  | пустая\ошибочная команда |
| 00100000 | 10000101 | 00000100 |  | mov 85h,R4 |
| 00000000 |  |  |  | пустая\ошибочная команда |
| 01000100 |  |  |  | push R4 |
| 01010011 |  |  |  | pop R3 |
| 00100000 | 11111111 | 00000011 |  | mov FFh,R3 |
| 10110000 | 00101000 | 00101001 |  | XOR 28h,29h |
| 10100000 | 10000001 | 10000010 |  | NAND 81h, 82h |
| 10010000 | 10010001 | 10000000 |  | SLL 91h,80h |
| 10000000 | 00100000 | 10000000 |  | SRL 20h,80h |
| 10011000 | 00100101 | 00001010 | 00000011 | SLL 25h,PC+Ah+R3 |
| 11001000 | 00111000 | 00010011 | 00000000 | SUB 38h,PC+13h+R0 |
| 00010000 | 00000100 |  |  | JMP 4h |

Теперь покажем содержимое памяти данных до и после выполнения программы на рисунках 3.9 и 3.10.

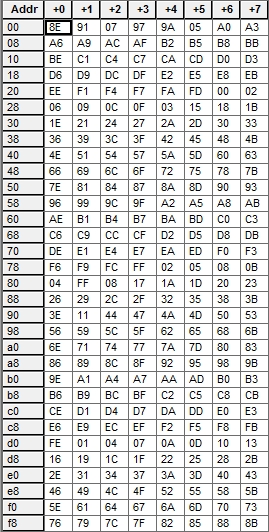


Рисунок 3.9 – Содержимое памяти данных до выполнения программы

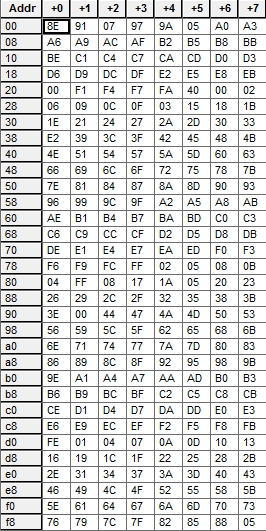


Рисунок 3.10 – Содержимое памяти данных после выполнения программы

# 4 АНАЛИЗ И ОПТИМИЗАЦИЯ РАЗРАБОТАННОЙ МИКРОЭВМ

Оптимизация, которую можно было бы сделать в данном проекте – это конвейеризация.

Конвейеризация любого процесса подразумевает его разделение на

примерно одинаковые по длительности этапы и позволяет выполнять

эти этапы одновременно и параллельно для нескольких объектов, над которыми производится конвейеризируемый процесс.

Как правило, цикл выполнения центральным процессором любой

команды состоит из следующих этапов:

– выборка команды;

– декодирование команды;

– вычисление адресов операндов;

– выборка операндов;

– исполнение команды;

– запись результата.

Несмотря на гипотетическую возможность выравнивания большинства из приведенных этапов по времени, на практике все этапы выполняются различное количество тактов.

Сама по себе реализация данного проекта очень плохо смогла бы лечь на схему конвейеризации, так как многие блоки не подразумевают параллельного чтения и записи.

ЗАКЛЮЧЕНИЕ

В данном курсовом проекте я реализовал микроЭВМ. Я получил достаточно хорошие знания в этой области. Сама ЭВМ вышла достаточно простой, однако свои функции она выполняет. Несмотря на то, что в данном проекте нет КПДП и некоторые блоки выполнены в стороне от основных блоков, все же данная ЭВМ способна функционировать правильно, вычислять, сохранять и загружать данные из памяти, работать со стеком, АЛУ, регистрами общего назначения и делать это достаточно быстро.

Кроме того, мной получены знания и о других аспекта работы ЭВМ благодаря реализации остальных блоков.

СПИСОК ЛИТЕРАТУРЫ

Столлингс, У. Структурная организация и архитектура компьютерных систем. 5-е изд. Пер. с англ. – М.: «Вильямс», 2001 – 892 стр.

Таненбаум, Э. Архитектура компьютерных систем. 4-е изд. Пер. с англ. – М.: «ПИТЕР», 2002 – 698 стр.