**Содержание**

ВВЕДЕНИЕ 3

1 РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ 4

1.1 Функциональный состав микро-ЭВМ 4

1.2 Разработка системы команд 5

1.3 Взаимодействие функциональных блоков 6

2 РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ 9

2.1 Запоминающие устройства 9

2.2 Центральный процессор 10

2.2.1 Регистровый файл 12

2.2.2 Арифметико-логическое устройство 13

2.2.3 Стек 14

2.3 КЭШ данных 15

2.4 КЭШ команд 18

3 ПОЛНАЯ ВРЕМЕННАЯ ДИАГРАММА РАБОТЫ МИКРО-ЭВМ 20

ЗАКЛЮЧЕНИЕ 28

СПИСОК ЛИТЕРАТУРЫ 29

ВВЕДЕНИЕ

Разрабатываемая микро-ЭВМ имеет Гарвардскую архитектуру. Поскольку по заданию необходимо использовать ПЗУ и ОЗУ блоки памяти, было принято решение разместить команды в ПЗУ, а данные, соответственно, в ОЗУ, т.к. они могут изменяться в процесс работы. Для каждого блока памяти будут использованы отдельные шины адреса и данных, разрядностью 8 и 8 бит соответственно.

Ядро разрабатываемой ЭВМ имеет регистровый файл классического типа, содержащий 4 регистра. Регистровый файл классического типа позволяет получить доступ к любому регистру в любой момент времени и прост в реализации.

Разрабатываемая микро-ЭВМ - гарвардского типа, что подразумевает наличие двух КЭШ (команд и данных). Оба КЭШа являются множественно-ассоциативными (k = 4) с алгоритмом замещение строк “Наиболее давнего хранения”. Количество строк в банке - 4. Тип синхронизации с памятью для КЭШ данных - “сквозная без отображения”. Обе КЭШ памяти имеют схожую внутреннюю организацию, однако КЭШ команд проще, т.к. команды не изменяются в процессе работы микро-ЭВМ.

Предсказатель переходов будeт реализован отдельно, т.к. разрабатываемая микро-ЭВМ является многотактовой, но не конвеерезированной.

**1** РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ

* 1. Функциональный состав микро-ЭВМ

Основные функциональные блоки разрабатываемой микро-ЭВМ:

* блоки КЭШ
  + блок КЭШ данных
  + блок КЭШ инструкций
* блоки памяти
  + блок памяти ОЗУ
  + блок памяти ПЗУ
* блок центрального процессора

Блок центрального процессора является самым сложным с архитектурной точки зрения и, в свою очередь, состоит из следующего набора блоков:

* блок декодирования команды
* блок регистров общего назначения
* блок арифметико-логического устройства
* блок стека

Микро-ЭВМ помимо регистров общего назначения (16 шт.) располагает специальными регистрами:

* IP (instruction pointer) – регистр, обозначающий смещение следующей команды относительно кодового сегмента, его разрядность совпадает с разрядностью шины адреса (8 бит)
* IR (instruction register) – регистр, хранящий исполняемую в данный момент команду, его разрядность определена общим количеством операций микро-ЭВМ.
* FR (flag register) – регистр флагов, содержит текущее состояние процессора, используется во многих операциях как арифметических и логических, так и в управляющих и прочих (в данной микро-ЭВМ только в логических операциях АЛУ и в операциях условного перехода), его разрядность в данной микро-ЭВМ взята 16 бит.
* DR (data register) – регистр данных, используется как регистр

накопитель для промежуточных данных, его разрядность совпадает с разрядностью шины данных и равна 8 бит.

Так или иначе, специальные регистры используются в различных операциях для хранения промежуточных результатов, достоверность содержимого специальных регистров не гарантируется, специальные регистры не могут быть адресованы, их использование задано железно схемой каждой операции. Для хранения данных используйте память (RAM), или регистры общего назначения, достоверность их содержимого гарантируется, содержимое определяется только пользователем.

**1.2** Разработка системы команд

Количество команд по заданию – 11:

* M->R(чтение из памяти в регистр общего назначения)
* R->M(чтение из регистра общего назначения в память)
* ADDC
* AND
* SLL
* NOT
* PUSH
* POP
* JMP
* JNS
* HLT

Для кодирования кода операции необходимо как минимум 4 бита, т.к. количество команд – 11 (диапазон значений КОП будет от 0000 до 1011).

При таком раскладе систему команд можно будет дополнить ещё 5-ю новыми операциями (диапазон значений от 1100 до 1111).

Количество регистров общего назначения – 4 шт. Для их адресации понадобится, как минимум 2 бита (диапазон значений от 00 до 11).

Для всех вариантов обязательном для реализации видом адресации является прямая адресация память (т.е. физический адрес операнда в памяти находится непосредственно в поле команды), также присутствует непосредственная и косвенная регистровая адресация. Для типа операции было выделено 2 бита (диапазон значений от 00 до 11).

Разрядность шины адреса составляет 8 бит, соответственно для адресации участка памяти при нахождении его непосредственно в коде (а не в регистре общего назначения, к примеру) в команде будет указан полный адрес, занимающий 8 бит (адресация регистра требует меньше – всего 2 бита).

Т.к. разрядность шины данных равна 8 бит и имеет место пакетная передача данных из блока контроля КЭШа, то будет иметь место некоторая избыточность данных при чтении команд из памяти ROM. К примеру: минимальное количество бит для той же простейшей команды NOT равно 4 бита (только КОП и ничего другого), но т.к. память позволяет минимально считать только 8 бит, то последующие вычитанные 4 бита просто не используются и никак не влияют на что-либо. Можно было бы кодировать из как КОП следующей команды, но это не даёт никаких преимуществ, даже более, это усложнит схему декодирования в блоке устройства управления.

Та же ситуация со всеми остальными командами, где в таблице присутствует символ ‘X’, трактующийся как “ любое значение (не влияет на команду никоим образом, является избыточным и имеет место лишь по причине задания конкретных параметров варианта для специализированного процессора курсового проекта)”.

При проектировании системы команд было принято решение отвести под одну команду 4 байта.

Таблица с набором команд и расположением операндов представлена в таблице 1.1. (КОП – код операции, addt – тип адресации, reg – номер регистра, memory addr – адрес в памяти, X – не определено).

Таблица 1.1 – Архитектура системы команд

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название команды | 1 байт | | | 2 байт | |
| [15..12] | [11..10] | [9..8] | [7..6] | [5..0] |
| M->R | КОП | addt | reg | memory addr | |
| R->M | КОП | addt | reg | memory addr | |
| ADDC | КОП | addt | reg | reg | X |
| AND | КОП | addt | reg | reg | X |
| NOT | КОП | addt | reg | X | |
| SLL | КОП | addt | reg | reg | X |
| JMP | КОП | addt | X | memory addr | |
| JNS | КОП | addt | reg | memory addr | |
| PUSH | КОП | addt | reg | X | |
| POP | КОП | addt | reg | X | |
| HLT | КОП | addt | X | X | |

**1.3** Описание взаимодействия всех блоков микро-ЭВМ при выполнении команд программы

Главной частью схемы является блок ЦП. Данный блок соединен с блоками памяти команд (CommandMemory) и памяти данных (DataMemory). Данные блоки содержат в себе соответствующие блоки КЭШ, взаимодействующие с ОЗУ и ПЗУ, предоставляя ЦП простой и удобный интерфейс для работы и реализуя "прозрачный” доступ к командам и данным.

Основной функция блока ЦП - управление командами и памятью. В данном блоке находится логика, управляющая работой ядра ЭВМ. Данный блок осуществляет декодирование команд, содержит механизм генерации опорных сигналов, соответствующие этапам выполнения команды (выборка, декодирование, выборка операндов, выполнение, сохранение операндов). В зависимости от стадии выполнения команды, блок управления генерирует сигналы, координирующие работу остальной части схемы ядра. При выполнении команд, требующий значения из памяти, блок управления инициирует обращение к КЭШ памяти и приостанавливает работу ЦП(в случае “промаха”) до получения данных.

Блок ЦП также включает в себя блок стека, блок регистров общего назначения, арифметико-логическое устройство, блок декодирования команды.

При моделировании работы ЭВМ можно задать только тактовый сигнал CLK. Код программы и операнды размещаются в ПЗУ и ОЗУ соответственно. Для изменения поведения микро-ЭВМ необходимо произвести изменение файлов памяти .hex, находящиеся в папке с проектом.

Алгоритм выполнения операции следующий:

* При подаче питания счетчик опорных сигналов хранит нули => на нулевом выходе декодера фазы команды висит логическая единица.
* В регистр IR заносится нулевой адрес.
* КЭШ-у команд отправляется запрос на получение команды по адресу IR (активируется сигнал READ). ЦП ждет сигнала HIT\_COMMAND от КЭШа команд, по этому сигналу инкрементируется содержимое регистра IR. Когда процессор вычитал 2 байта (длинна команды) из КЭШа команд, счетчик регистра IR приостанавливается на время выполнения команды (до получения сигнала SCL, который сигнализирует о завершении выполнения текущей команды).
* Схема переходит на стадию декодирования команды (запускается генератор опорных сигналов)
* После декодирования команды начинается стадия выборки операндов. Для разных команд данная стадия выполняется по разному:

1. **MOV** a, reg – из регистра достается операнд и выставляется на шину CPU\_DATA[7..0].
2. **MOV** reg, a – выставляется сигнал READ для КЭШа команд, адрес подается на шину DATA\_ADDR[7..0]. КЭШ команд проверяет, есть ли данные по такому адресу в КЭШ. Если есть, выставляет сигнал HIT\_COMMAND и данные отправляются на шину DATA\_BUS[7..0]. Если сигнал HIT\_COMMAND не пришел, процессор приостанавливает свое работу до получения данных.
3. **MOV** reg, reg; **AND** reg, reg; **ADDC** reg, reg; **SLL** reg,reg – из команды вынимаются номера регистров, в которых находятся операнды. Операнды по указанным адресам вынимаются из блока РОН и устанавливаются на шину DATA[7..0].
4. **MOV** reg, [reg]; **AND** reg, [reg]; **ADDC** reg, [reg]; **SLL** reg,[reg] – из команды вынимаются номера регистров, в которых находятся адреса операнд. Адреса устанавливаются на шину DATA\_ADDR[7..0], далее ЦП работает как в пункте 2.
5. **POP** - операнд вынимается из вершины стека, устанавливается на DATA[7..0] .
6. **JMP, JNS -** операнд из команды устанавливается на ADD[7..0].
7. **HLT -** при получении данной команды устройство управления перестает генерировать сигналы и блок ЦП простаивает.

* Выполнение различных команд тоже производится по разному:

1. **MOV** a, reg – подается сигнал WRITE КЭШу данных и данные с шины CPU\_DATA[7..0]. КЭШ проверяет, есть ли данные по такому адресу. Если данные по данному адресу присутствую КЭШ обновляет их, а также записываются данные в ОЗУ.
2. **MOV** reg, a – на шину данных выставляется значение по адресу a и значение с DATA\_BUS[7..0] записывается в регистр по сигналу W\_RON.
3. **AND, ADDC, SLL** – значения с выхода РОН RON\_1 и выхода РОН RON\_2 ( DATA\_BUS[7..0] в случае косвенной адресации)

поступают на входы АЛУ. Результат работы АЛУ записывается в регистр результата АЛУ.

1. **PUSH** – запись значения с выхода РОН RON\_1 на вершину стека.
2. **JMP, JNS** – запись в нового адреса IR ( в случае JNS бит регистра флагов SF должен быть установлен в 1)

Затем, алгоритм повторяется до тех пор, пока не будут выполнены все команды, предусмотренные заложенной программой.

**2** РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ

**2.1** Запоминающих устройства

По варианту, в курсовом проекте необходимо использовать 2 блока памяти :

1. постоянную память (ПЗУ);
2. оперативную (ОЗУ);

Блоки запоминающих устройств не разрабатываются, т.к. есть возможность воспользоваться уже спроектированными, предоставляемыми средством Magic Wizard.

Оба блока памяти могут “прошиваться” специальными файлами в формате .hex.

Ключевое отличие между ними в том, что блок ПЗУ во время работы ЭВМ доступен только в режиме чтения, в отличие от блока ОЗУ, данный в котором могут изменяться.

Т.к. в наборе инструкций присутствуют инструкции пересылки данных в память, то для хранения данных будет использован блок ОЗУ а для хранения инструкций (которые изменяться не будут) - блок ПЗУ.

Файл прошивки ОЗУ представлен на рис. 2.1

Файл прошивки ПЗУ представлен на рис. 2.2

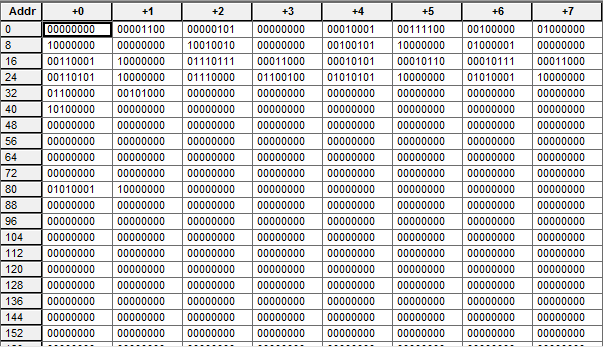


Рис. 2.1 - Содержимое файла прошивки OЗУ

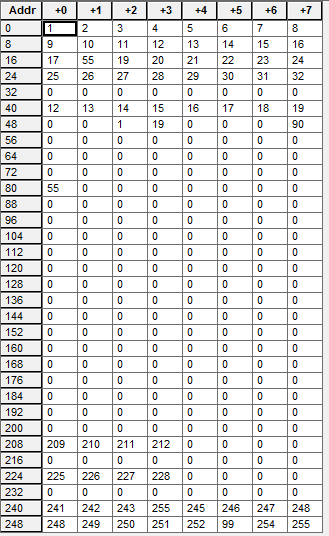


Рис. 2.2 - Содержимое файла прошивки ПЗУ

**2.2** Центральный процессор

Блок ЦП - ядро разрабатываемой микро-ЭВМ. Данный блок является ключевым и занимается выполнением инструкций, поступающий из КЭШ команд, над данными, поступающими из КЭШ данных.

Функциональная схема блока представлена на чертеже “Блок центрального процессора. Схема комбинированная структурная”.

УГО блока ЦП представлено на рис. 2.3



Рисунок 2.3

Входами блока ЦП являются сигналы “попадания” КЭШа данных и команд. Инструкция поступает в блок ЦП по входной шине COMMAND\_BUS[7..0], данные по шине DATA\_BUS[7..0].

Главными выходами блока являются сигналы на чтение и запись КЭШ данных (CONTROL\_BUS[0], CONTROL\_BUS[1]) и чтения КЭШ команд (CONTROL\_BUS[2]), а также выходы DATA\_ADDR[7..0] и COMMAND\_ADDR[7..0] которые подают адреса на КЭШ данных и команд соответственно. Остальные выходные сигналы смысловой нагрузки не несут и необходимы для отладки микро-ЭВМ и более детализированной демонстрации работы схемы.

На схеме присутствуют следующие функциональные блоки:

* Блок регистров общего назначения
* Блок стека
* Блок АЛУ
* Блок декодера команды

Все вышеперечисленные блоки будут подробно описаны в этом разделе.

Для тестирования блока ЦП была выбрана та же последовательность инструкций, которая будет использована при проверке работоспособности микро ЭВМ. Данная последовательность сохранена в файле прошивки ROM.hex и будет является программой для выполнения. Код программы в ассемблерном и двоичном виде представлен в таблице 2.2

Таблица 2.2 – Код программы

|  |  |
| --- | --- |
| ASM | BIN |
| MOV AX, $12 | 00000000 00001100 |
| MOV BX, [AX] | 00000101 00000000 |
| MOV 60, [BX] | 00010001 00111100 |
| AND AX,BX | 00100000 01000000 |
| PUSH AX | 10000000 00000000 |
| POP CX | 10010010 00000000 |
| AND BX, [CX] | 00100101 10000000 |
| NOT BX | 01000001 00000000 |
| ADDC BX, CX | 00110001 10000000 |
| JNS 24 | 01110111 00011000 |
| ADDC BX,[CX] | 00110101 10000000 |
| JNS 100 | 00111000 01100100 |
| HLT | 10100000 00000000 |

Полная временная диаграмма работы блока ЦП представлена на рис. 2.4 - 2.6.

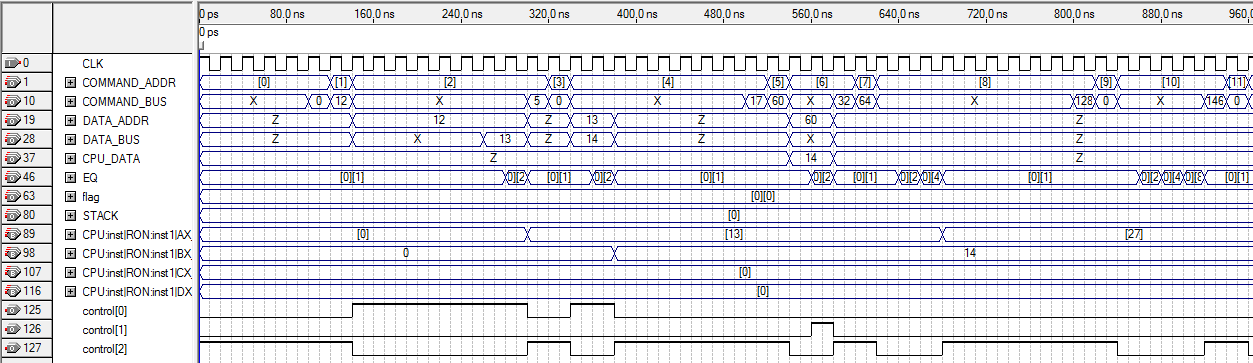


Рис. 2.4 - Временная диаграмма ч. 1

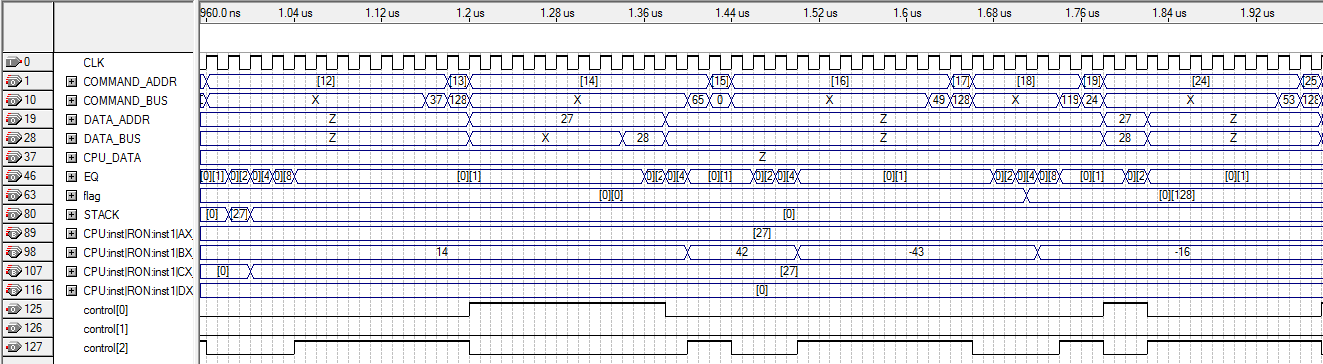


Рис. 2.5 - Временная диаграмма ч.2

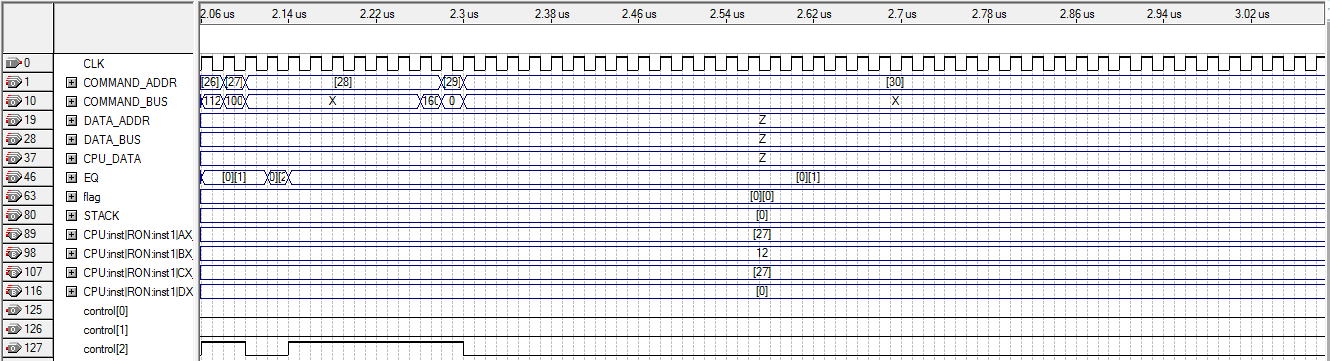


Рис. 2.6 - Временная диаграмма ч.3

**2.2.1** Регистровый файл

При реализации ядра микро-ЭВМ было принято решение использовать регистровый файл классического типа (количество регистров - 4 шт). Данный тип организации внутренних регистров ЦП прост в реализации, в любой момент времени можно получить доступ к любому регистру (в отличие от. стекового типа).

Функциональная схема представлена на чертеже “Блок РОН. Схема комбинированная”.

УГО блока РОН представлена на рис. 2.7

Временная диаграмма блока РОН представлена на рис. 2.8



Рис. 2.7 - УГО блока РОН

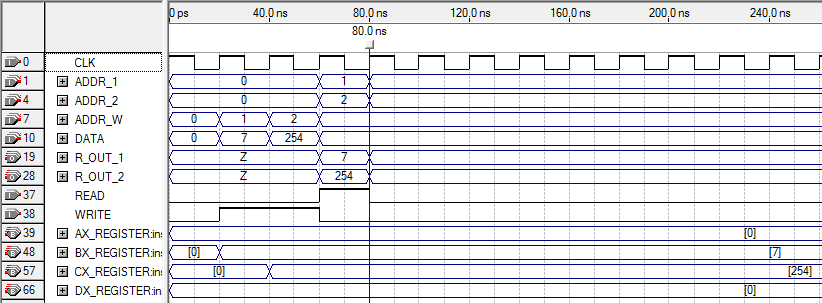


Рис. 2.8 - Временная диаграмма работы блока РОН

Принцип работы схемы прост: В схеме присутствуют 2 декодера адреса на чтение и 1 на запись. Первые открывают выходы регистров, чьи адреса поданы на входы схемы, на выход. Декодер на запись работает таким же образом, но управляет значением на входе enable каждого регистра (данная сигнальная линия разрешает запись регистра).

**2.2.2** АЛУ

Арифметико-логическое устройство – одна из основных составляющих блока ЦП. Обязанность схемы - выполнение всех арифметических и логических операций. Данное устройство является неотъемлемой частью тракта данных.

Функциональная схема блока представлена на чертеже “Блок АЛУ. Схема комбинированная ”.

УГО блока АЛУ представлено на рис. 2.9

Временная диаграмма блока РОН представлена на рис. 2.10



Рис. 2.9 - УГО блока АЛУ

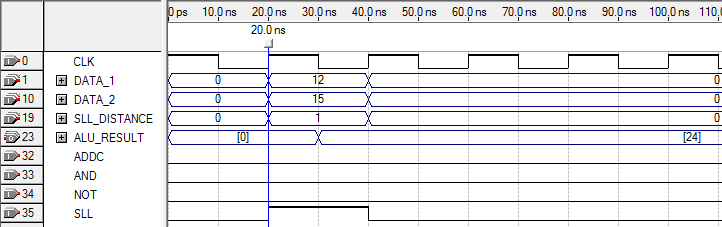


Рис. 2.10 - Временная диаграмма работы блока АЛУ

Схема имеет две входные шины для операндов (DATA\_1[7..0] и DATA\_2[7..0]), результат выполнения операции над данными с которых будет подан на выход ALU\_RESULT[7..0], в зависимости от входных управляющих сигналов, соответствующих конкретным операциям. В соответствии с вариантом, блок АЛУ способен производить операции AND, ADDC, AND и SLL.

Необходимо отметить, что данная схема является комбинационной, т.е. отрабатывает за один такт. В силу этого, на уровне блока АЛУ реализован регистр для хранения результата последней АЛУ операции. Из этого регистра на этапе сохранения данные попадают в РОН.

**2.2.3** Стек

Стек - структура данных, организованных особым образом. Добавление  и удаление элементов из стека производится только с головы стека. Голова стека - указатель на один из элементов стека.

В данном курсовом проекте в ядре ЦП аппаратно реализован регистровый файл стекового типа на 11 регистров.

Для ЦП блок стека предоставляет след. интерфейс сигнальных линий:

1. PUSH - для добавления операнда с RON\_1 в стек.

2. POP - для снятия операнда с вершины стека на DATA[7..0].

Функциональная схема стека представлена на чертеже “ Блок стека*.* Схема комбинированная ”.

УГО блока стека представлено на рис. 2.11

Временная диаграмма блока стека представлена на рис. 2.12



Рис. 2.11 - УГО блока стека

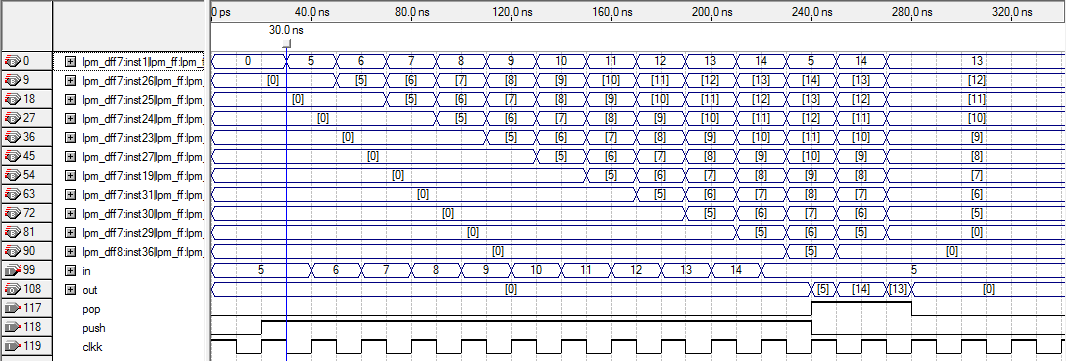


Рис. 2.12 - Временная диаграмма работы блока стека

Ни при команде POP ни при PUSH нет необходимости стирать значение из регистра, т.к. он в любом случае будет перезаписан при операции PUSH прежде, чем невалидные данные будут выданы схемой.

Красивое схемотехническое решение, примененное в блоке управления стеком, позволяет не делать задержки между операциями на стеке. Т.е. операции PUSH и POP могут идти в произвольном порядке с произвольной плотностью.

**2.3** КЭШ данных

КЭШ память - быстрая, близкая к процессору память небольшого размера. КЭШ память является статической.

В данной КЭШ памяти реализован механизм замещения строк "Наиболее давнего хранения". Данный способ является простым, с точки зрения реализации, а также эффективным. Данный алгоритм предполагает наличие счетчика в каждой строке КЭШ, который считает какое количество тактов к строке не было обращения, а при записи делает “активной” ту строку,

которая хранится в КЭШе дольше всего.

Алгоритм общения с памятью - "Сквозная запись без отображения". Согласно этому алгоритму, при поступлении сигнала от ЦП о записи данных, данные сначала пишутся в ОЗУ. После записи в ОЗУ необходимо проверить наличие в КЭШ копии данных. Если есть, необходимо пересылать блока из ОЗУ, тем самым обновив копию.

Размер шины адреса - 8 бит.

ADDR[1..0] – смещение слова в строке.

ADDR[3..2] – выбор множества, в которое будет записана строка.

ADDR[7..4] – тег.

В какой “банк” будет записана строка определяется алгоритмом замещения НДХ.

Функциональная схема КЭШ представлена на чертеже “КЭШ данных. Схема комбинированная”.

УГО блока КЭШа данных представлено на рис. 2.13

Временная диаграмма блока КЭШа данных представлена на рис. 2.14



Рис. 2.13 - УГО блока КЭШа данных

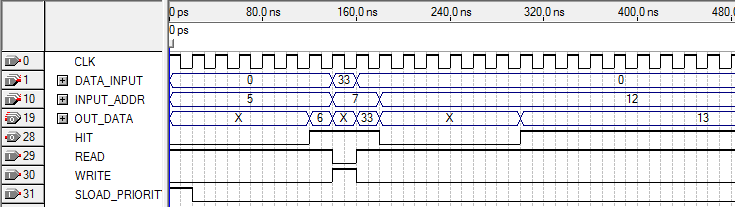


Рис. 2.14 - Временная диаграмма блока КЭШа данных

На схеме КЭШ присутствуют блоки SetCache, декодеры и счетчики. Декодеры необходимы для выбора множества КЭШ, строка в котором должна быть записана\считана. Счетчик нужен для генерации адресов, при промахе.

Необходимо отметить, что SetCache определяет в какую именно строку множества будет производится запись/чтение. Данный блок инкапсулирует в себе 4 строки КЭШа (LineCache), 4 регистра тэгов(по 1 на каждую строку) и 4 счетчика для реализации алгоритма замещения НДХ. Если по тэгу входного адреса попадания не произошло, генерируется сигнал MISS и происходит чтение данных из памяти в КЭШ.

 Функциональная схема SetaCache представлена на чертеже “Блок множества КЭШ. Схема комбинированная структурная”

УГО блока SetCache представлено на рис. 2.15



Рис. 2.15 - УГО блока SetCaсhe.

Блок SetCaсhe в свою очередь состоит из строк (LineCache). На уровне строки КЭШа реализован алгоритм распределения входных. В какое именно слово будут записаны данные определяется двумя младшими битами входного адреса.

Функциональная схема LineCache представлена на чертеже “Блок строки КЭШ. Схема комбинированная”

УГО блока LineCache представлено на рис. 2.16



Рис. 2.16 - УГО блока LineCache

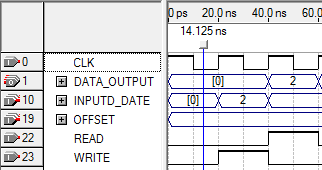


Рис. 2.17 - Временная диаграмма блок LineCache

КЭШ память работает в режимах **Чтения** и **Записи**.

При чтении:

1. На вход КЭШ памяти поступает адрес и сигнал READ

2. Из части адреса, который хранит номер строки, считываются [3..2] биты для выбора SetCache.

3. На схеме SetCache 4 старших бита адреса являются тэгом и алгоритмом замещения определяется строка в КЭШе, с которой будут читаться данные.

4. На схему LineCache поступают 2 младших бита адреса, декодер выбирает слово которое поступает на выход схемы.

При записи:

1. На вход КЭШ памяти поступает адрес и сигнал WRITE

2. Если по данному адресу происходит “попадание”, строка перезаписывается.

На рис. 2.18 представлена временная диаграмма работы КЭШ данных при промахе, попадании и записи.

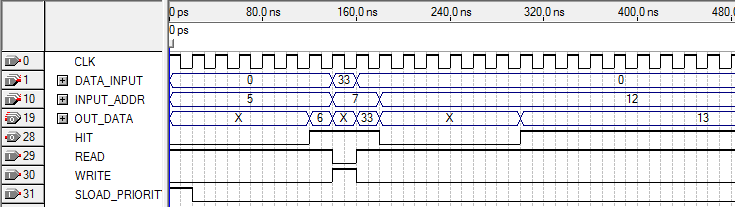


Рис. 2.18 - Временная диаграмма блока КЭШ данных

**2.4** КЭШ команд

Блок КЭШ команд по структуре является аналогичным блоку КЭШ данных. На верхнем уровне находятся SetCache, внутри - LineCache. Количество множеств, уровень ассоциативности, количество слов в строке такое же, как и в описанной выше схеме.

КЭШ команд проще, т.к. данные, находящиеся в нем, только для чтения. Это значит, что для КЭШ команд не нужно организовывать механизм общения с памятью, кроме как для получения блока данных.

УГО блока КЭШ команд представлено на рис. 2.19

Временная диаграмма работы КЭШа команд представлена на рис. 2.20



Рис. 2.19 - УГО блока КЭШ команд

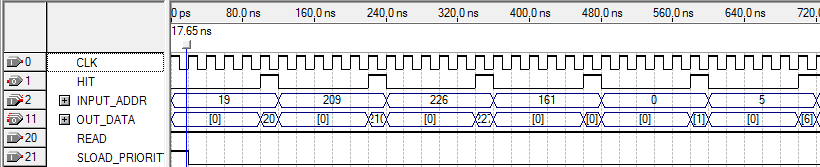


Рис. 2.20 - Временная диаграмма блока КЭШ команд

**3** ПОЛНАЯ ВРЕМЕННАЯ ДИАГРАММА ФУНКЦИОНАЛЬНОГО МОДЕЛИРОВАНИЯ РАБОТЫ МИКРО-ЭВМ И ЕЁ АНАЛИЗ

Последовательность команд, выбранная для моделирования системы представлена в таблице 3.1

Таблица 3.1 – последовательность команд

|  |  |
| --- | --- |
| ASM | BIN |
| MOV AX, $12 | 00000000 00001100 |
| MOV BX, [AX] | 00000101 00000000 |
| MOV 60, [BX] | 00010001 00111100 |
| AND AX,BX | 00100000 01000000 |
| PUSH AX | 10000000 00000000 |
| POP CX | 10010010 00000000 |
| AND BX, [CX] | 00100101 10000000 |
| NOT BX | 01000001 00000000 |
| ADDC BX, CX | 00110001 10000000 |
| JNS 24 | 01110111 00011000 |
| ADDC BX,[CX] | 00110101 10000000 |
| JNS 100 | 00111000 01100100 |
| SLL BX, [CX] | 10100000 00000000 |
| SCL BC, [CX] | 01010001 10000000 |
| JMP 40 | 01100000 00101000 |
| HLT | 00000000 00000000 |

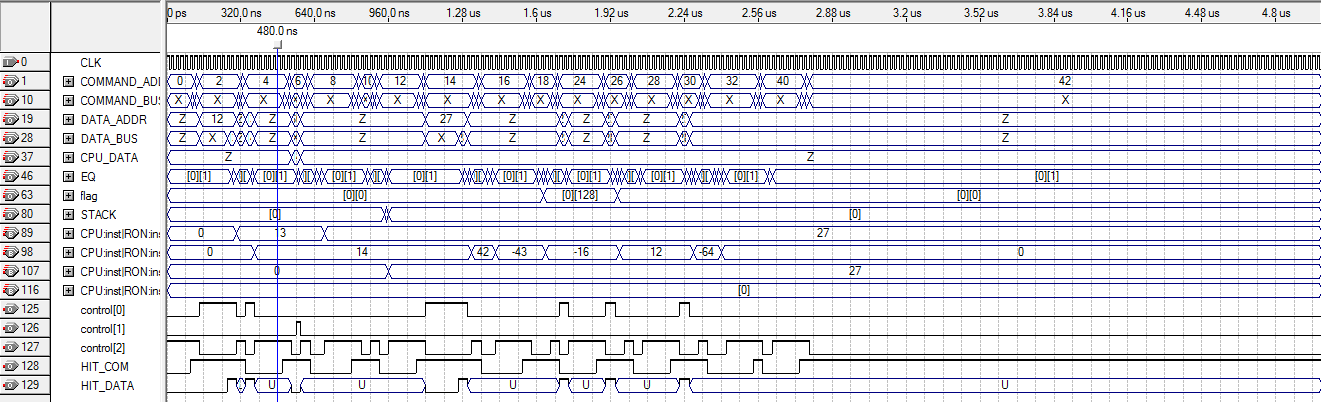


Рис. 3.1 – Временная диаграмма работы микро-ЭВМ в уменьшенном масштабе

На рис. 3.2 – 3.11 представлены отдельные участки временной диаграммы с пояснениями.

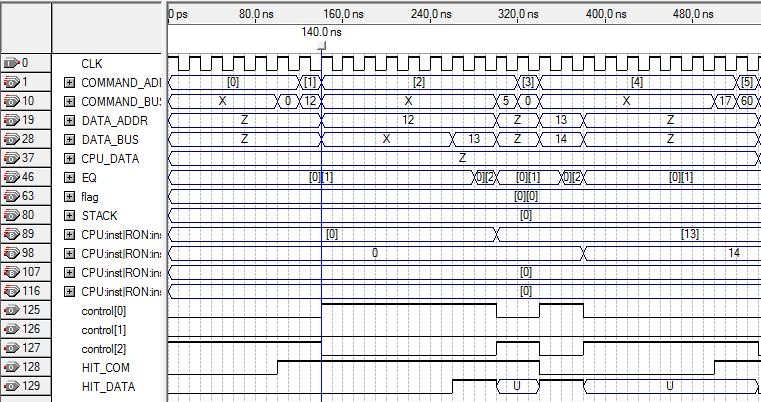


Рис. 3.2 – Выполнение команд MOV AX, $12 и MOV BX, [AX]

На рисунке 3.2 предоставлено выполнение команд MOV AX, $12 и

MOV BX, [AX].

ЦП устанавливает сигнал чтения для КЭШа команд (control[2]) и адрес на COMMAND\_ADDR[7..0]. В КЭШ команд данных по такому адресу нет, поэтому HIT\_COM = ‘0” и выполняется пересылка данных из ROM в КЭШ данных (7 тактов). Когда данные были вычитаны в КЭШ, сигнал HIT\_COM устанавливается в 1 и происходит пересылка команды в ЦП. После декодирования команды ЦП вставляет сигнал чтения из КЭШ данных control[0], в КЭШ данных нет данных по этому адресу, происходит пересылка, а ЦП приостанавливает генерацию опорных сигнал до получения сигнала HIT\_DATA. Далее происходит пересылка данных с шины DATA\_BUS[7..0] в регистр AX.

Далее происходит выполнение 2-ой команды MOV BX, [AX] ( 00000101 00000000). Так как размерность строки КЭШ 4 байта, при чтении первой команды из ROM в КЭШ вычиталась и вторая команда, поэтому

при установке ЦП сигнала control[2] команды сразу же попадают на COMMAND\_DATA[7..0]. На DATA\_ADDR[7..0] устанавливается содержимое регистра AX (косвенная адресация) и ЦП ждет сигнала HIT\_DATA от КЭШа данных. Далее происходит запись данных хранящихся по адресу AX в регистр BX.

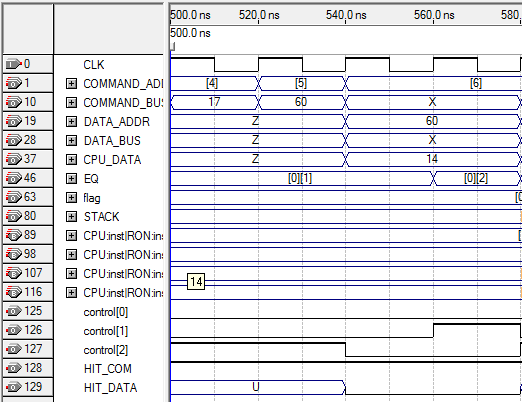


Рис. 3.3 – Выполнение команды MOV $60, BX

На шину DATA\_ADDR[7..0] устанавливается адрес памяти, в который происходит запись, на шину CPU\_DATA[7..0] выставляются данные из регистра BX. ЦП устанавливает сигнал control[1] (запись в RAM).

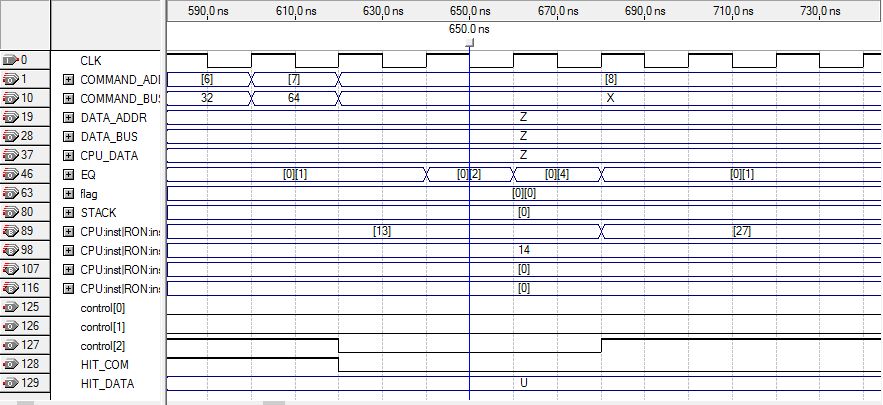


Рис. 3.4 – Выполнение команды AND AX, BX

Данные из регистров AX и BX поступают на входы АЛУ, по первому синхросигналу происходит вычисление суммы, по второму синхросигналу данные сохраняются в регистре AX.

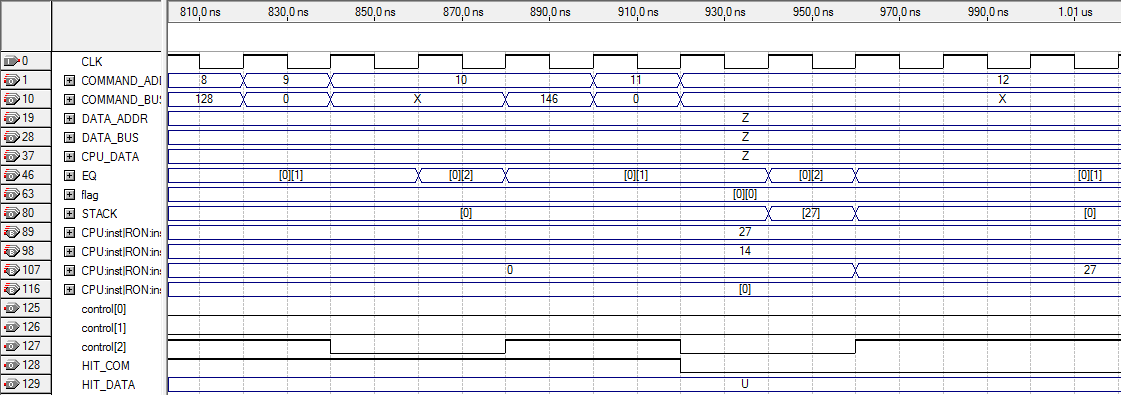
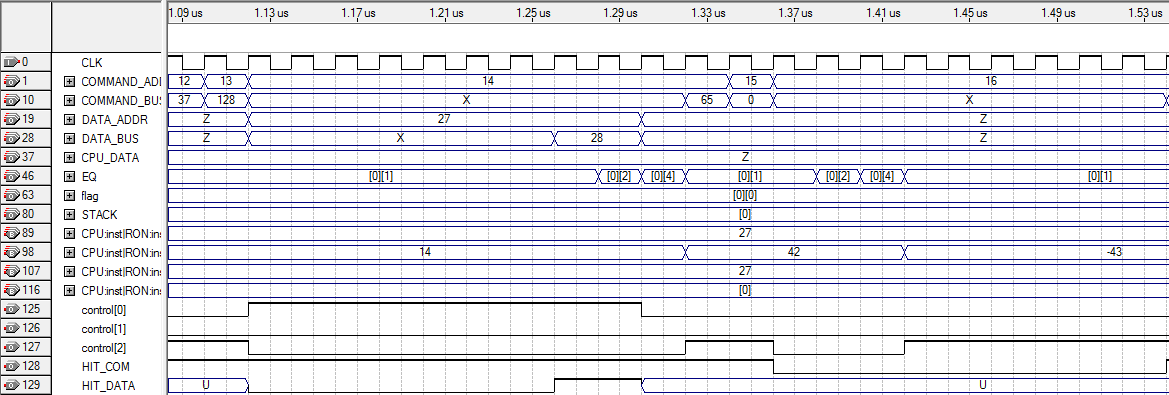


Рис. 3.5 – Выполнение команд PUSH AX и POP BX

Данные из регистра AX помещаются на вершину стека, далее данные из вершины стека помещаются в регистр BX.



Риc. 3.6 – Выполнение команд AND BX,[CX] и NOT BX

MOV BX, [CX]: содержимое регистра CX поступает на шину DATA\_ADDR[7..0] и процессор переходит в режим ожидания операнда (сигнала HIT\_DATA). Когда приходит сигнал HIT\_DATA на входы АЛУ подаются содержимое DATA\_BUS[7..0] и регистра BX. Результат сохраняется в регистре BX.

NOT BX: содержимое регистра BX поступает на входу АЛУ и выполняется инвертирование, по следующему синхросигналу содержимое с выхода АЛУ записывается в регистр BX.

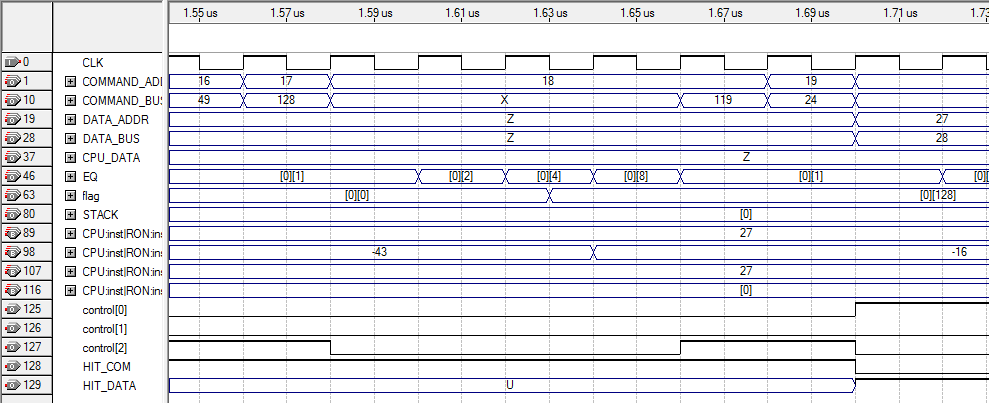


Рис. 3.7 – Выполнение команды ADDC BX, CX

ADDC BX, CX: содержимое регистров BX и CX поступает на входы АЛУ, также на вход АЛУ поступает бит СF регистра флагов. Происходит вычисление суммы. Результат сохраняется в регистре BX. Также перезаписывается регистр флагов.

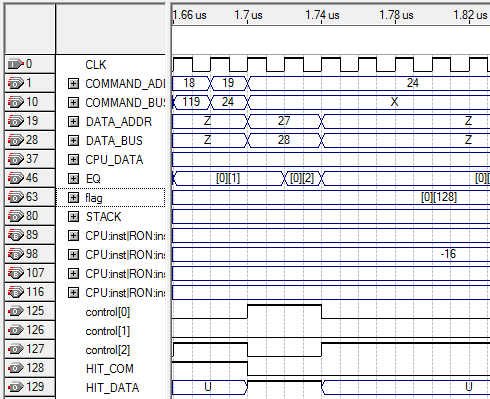


Рис. 3.8 – Выполнение команды JNS 24

На вход регистра IP подается содержимое адресной части команды и так как флаг SF = 1 (последняя операция АЛУ имела отрицательный результат), генерируется сигнал асинхронной загрузки регистра LOAD.

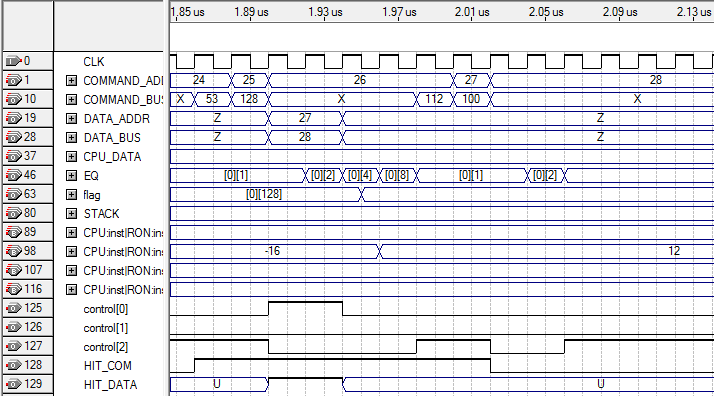


Рис. 3.9 – Выполнение команды ADDC BX,[CX] и JNS 100

ADDC BX, [CX]: на входы АЛУ поступают содержимое регистра BX и данные по адресу содержимого регистра CX, также на вход АЛУ поступает бит СF регистра флагов. Происходит вычисление суммы. Результат сохраняется в регистре BX. Также перезаписывается регистр флагов.

JNS 100: На вход регистра IP подается содержимое адресной части команды и так как флаг SF = 0 (последняя операция АЛУ имела положительный результат), не генерируется сигнал асинхронной загрузки регистра LOAD, соответственно “прыжка” не происходит.

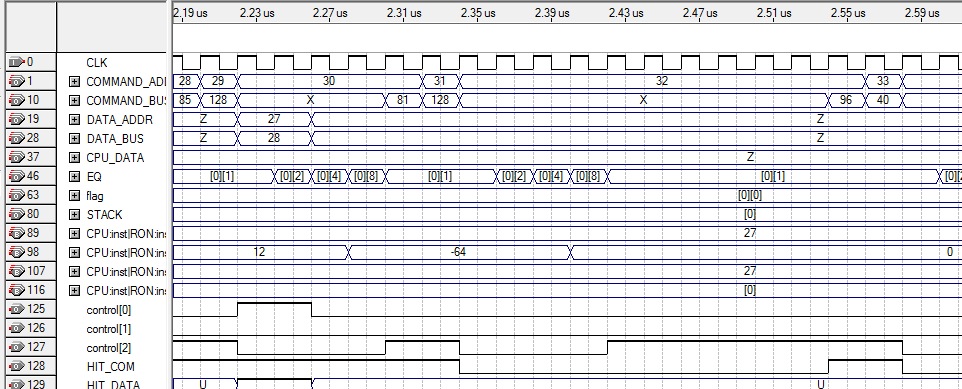


Рис. 3.10 – Выполнение команды SLL BX,CX и SLL BX,[CX]

SLL BX, CX: на входы АЛУ поступают содержимое регистра BX и CX. Происходит вычисление сдвига. Результат сохраняется в регистре BX. Также перезаписывается регистр флагов.

SLL BX, [CX]: на входы АЛУ поступают содержимое регистра BX и данные по адресу содержимого регистра CX. Происходит вычисление сдвига. Результат сохраняется в регистре BX. Также перезаписывается регистр флагов.

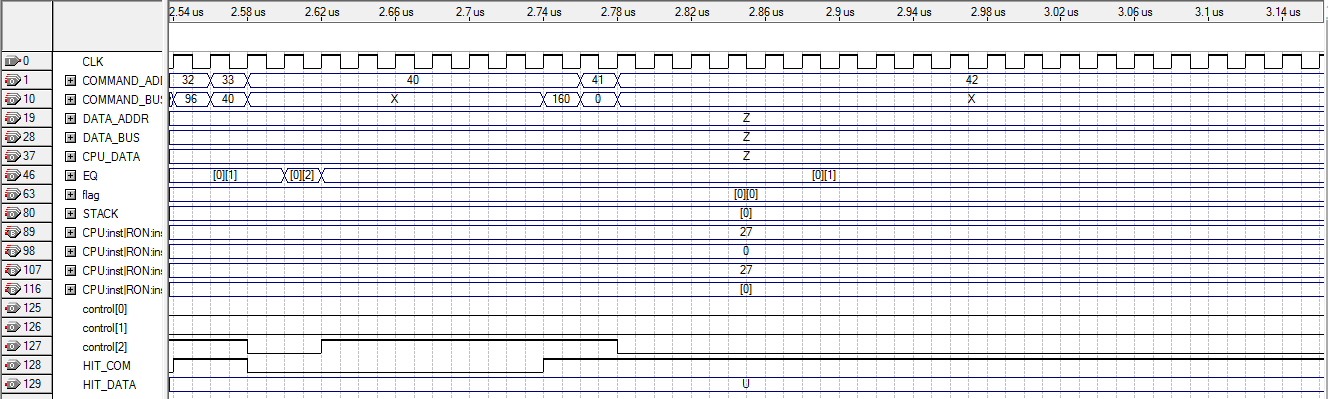


Рис. 3.11 – Выполнение команды JMP 40 и HLT

В финальной части моделирования выполняется команда безусловного перехода JMP 40 – прыжок на 40 адрес, по которому расположена команда HLT – команда остановки процессора, после генератор опорных сигналов в блоке центрального процессора останавливается.

Чтобы убедится, что пересылка числа 7 в ОЗУ прошла успешно, необходимо посмотреть дампы памяти до моделирования и после моделрования.

Содержимое ОЗУ до моделирования работы микро-ЭВМ представлено на рис. 3.12.

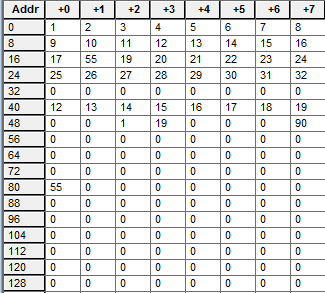


Рис. 3.12 – содержимое ОЗУ до моделирования

Содержимое ОЗУ после моделирования работы микро-ЭВМ представлено на рис. 3.13

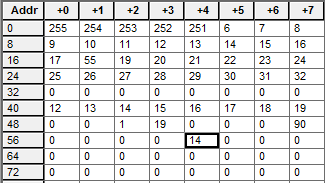


Рис. 3.13 – содержимое ОЗУ после моделирования.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта успешно разработана микро-ЭВМ с Гарвардской архитектурой, иерархической подсистемой памяти и RISC набором из 11 команд, использующих прямую регистровую и косвенную регистровую адресацию.

Разработана иерархическая модель памяти в виде 4 регистров РОН, КЭШа, ОЗУ и ПЗУ.

Разработан универсальный алгоритм остановки ЦП после выполнения всех инструкций программы (JMP на нулевой адрес, по которому всегда расположена команда останова ЦП).

Разработанная архитектура микро-ЭВМ является расширяемой как с точки зрения увеличения количества инструкций, так и с точки зрения реализации дополнительных возможностей (добавления КПДП, добавления механизма аппаратных прерываний).

СПИСОК ЛИТЕРАТУРЫ

1. Самаль Д.И - Структурная и функциональная организация ЭВМ. -БГУиР, кафедра ЭВМ, 2013.
2. Сергиенко\_А.М. – VHDL Для Проектирования Вычислительных.Устройств, 2003
3. Поляков\_А.К. – Языки VHDL и VERILOG в Проектировании Цифровой Аппаратуры, Москва СОЛОМОН-Пресс, 2003
4. И.И. Глецевич, В.А. Прытков, А.В. Отвагин - Методические указания по дипломному проектированию для студентов специальности 40 02 01 «Вычислительные машины, системы и сети» всех форм обучения, Минск, 2013.
5. Documentation: Quartus II Development Software – режим доступа: <http://www.altera.com/literature/lit-qts.jsp>.