**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ

1. РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ
2. РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ
3. ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ
4. АНАЛИЗ И ОПТИМИЗАЦИЯ РАЗРАБОТАННОЙ МИКРО-ЭВМ
5. ЗАКЛЮЧЕНИЕ

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

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. 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 2. 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 3. 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 3. 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.

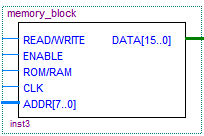


Рисунок 1. Условно-графическое обозначение блок памяти.

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.

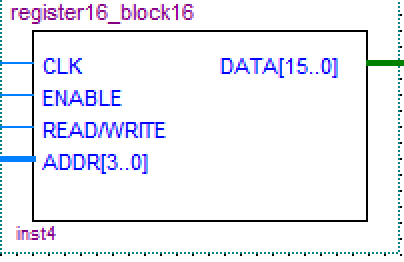


Рисунок 3. Условно-графическое обозначение блока РОН.

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.

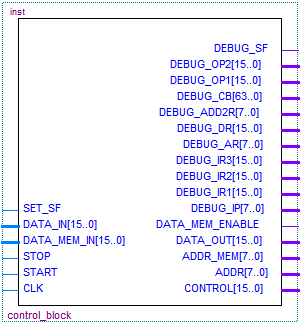


Рисунок 4. Условно-графическое обозначение устройство управления.

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.

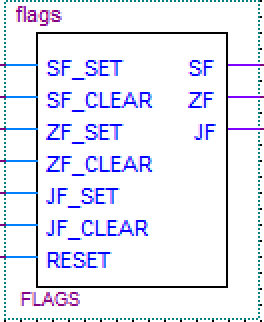


Рисунок 5. Условно-графическое обозначение регистра флагов.

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.

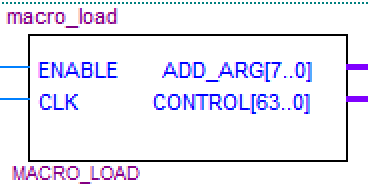


Рисунок 6. Условно-графическое обозначение логического блока устройства управления.

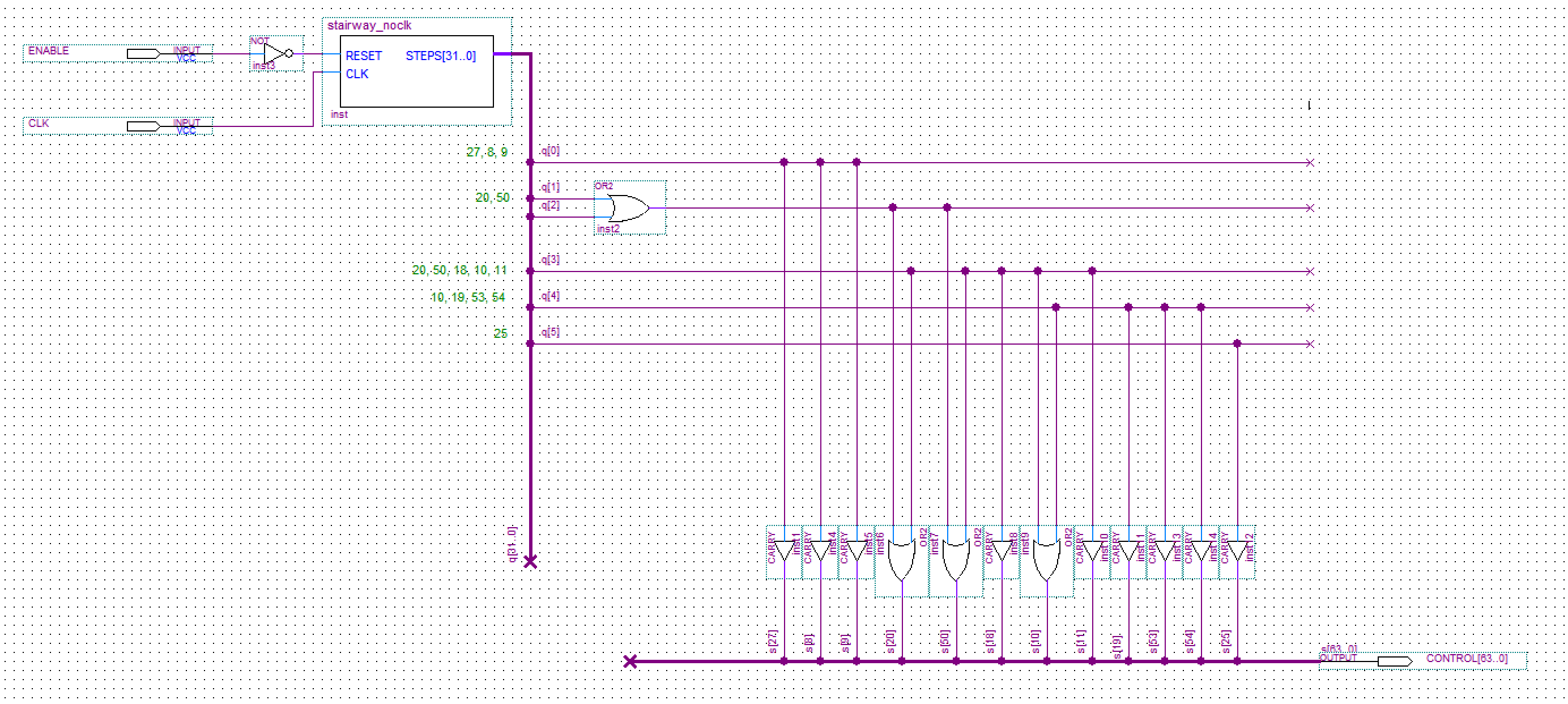


Рисунок 7. Пример внутренней реализации логического блока устройства управления.

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}

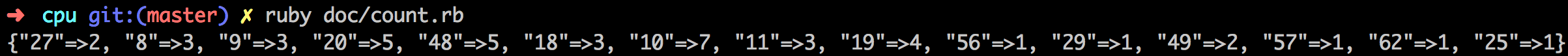


Рисунок 8. Результат выполнения работы программы.

* 1. Стек.

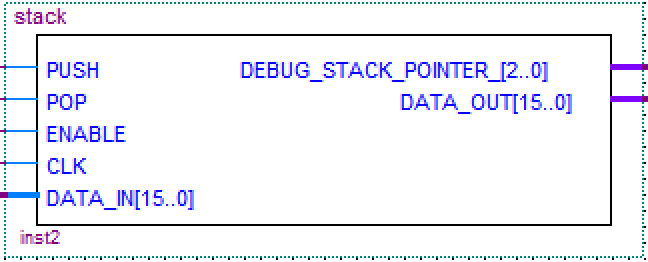


Рисунок 9. Условно-графическое обозначение стека.

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.

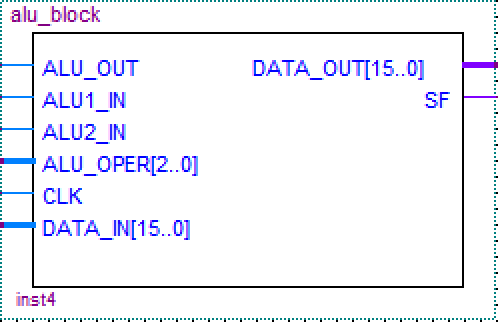


Рисунок 10. Условно-графическое обозначение АЛУ.

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 4. 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.

3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ

В данном разделе приведены временные диаграммы функционального моделирования отдельных модулей и всего устройства в целом. Во время объяснения работы модулей будут опускаться единицы измерения времени для удобства, предполагая что имеются ввиду наносекунды.

* 1. Блок ОЗУ и ПЗУ.

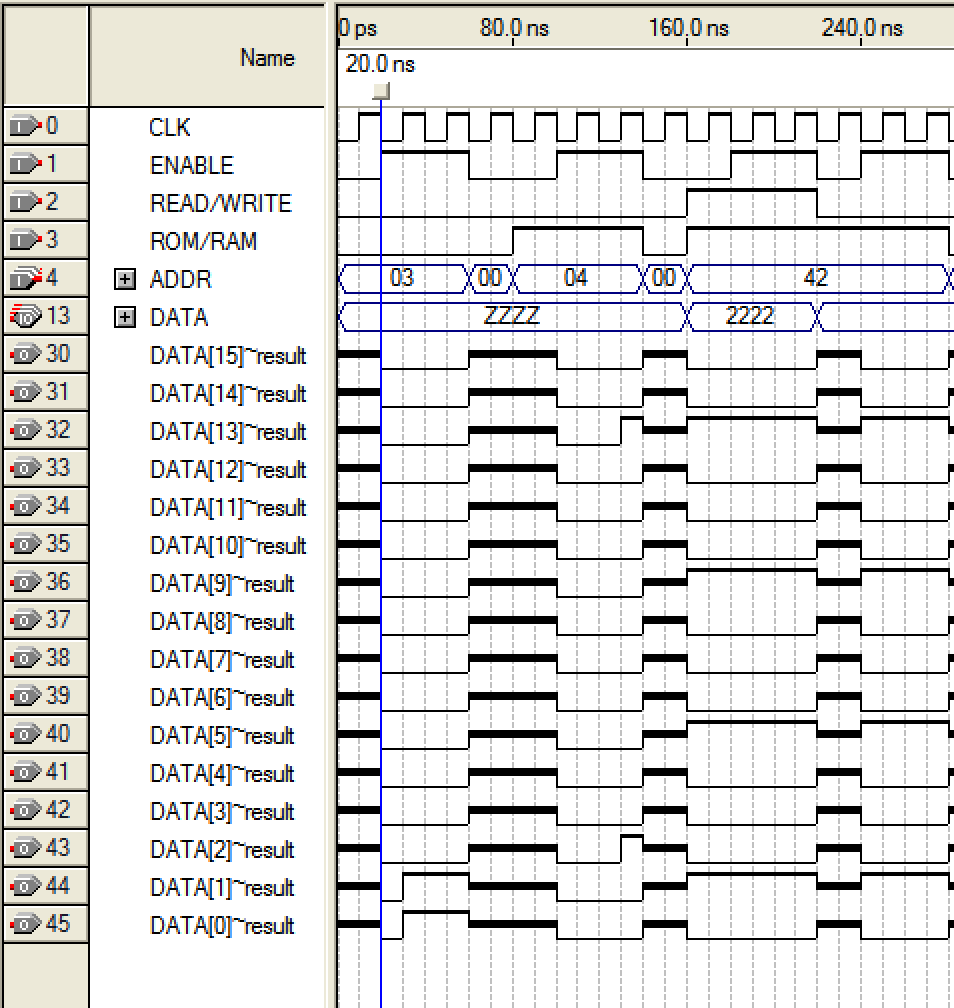


Рисунок 11. Функциональное моделирование ПЗУ и ОЗУ

В интервале 20-60 происходит попытка чтения из ПЗУ по адресу 03. В результате мы можем видеть что результат появляется на выходе DATA на интервале 30-60. Далее на интервале 100-140 происходит чтение из ОЗУ, результат которого заметен на интервале 130-140. Последний шаг – это запись в ОЗУ на интервале 160-220 по адресу 42 значения 2222 и последующая попытка чтения только что записанного значения на интервале 240-280. Как видно после чтения мы также получаем значение 2222, что означает что наше устройство функционирует правильно.

* 1. Регистры общего назначения.

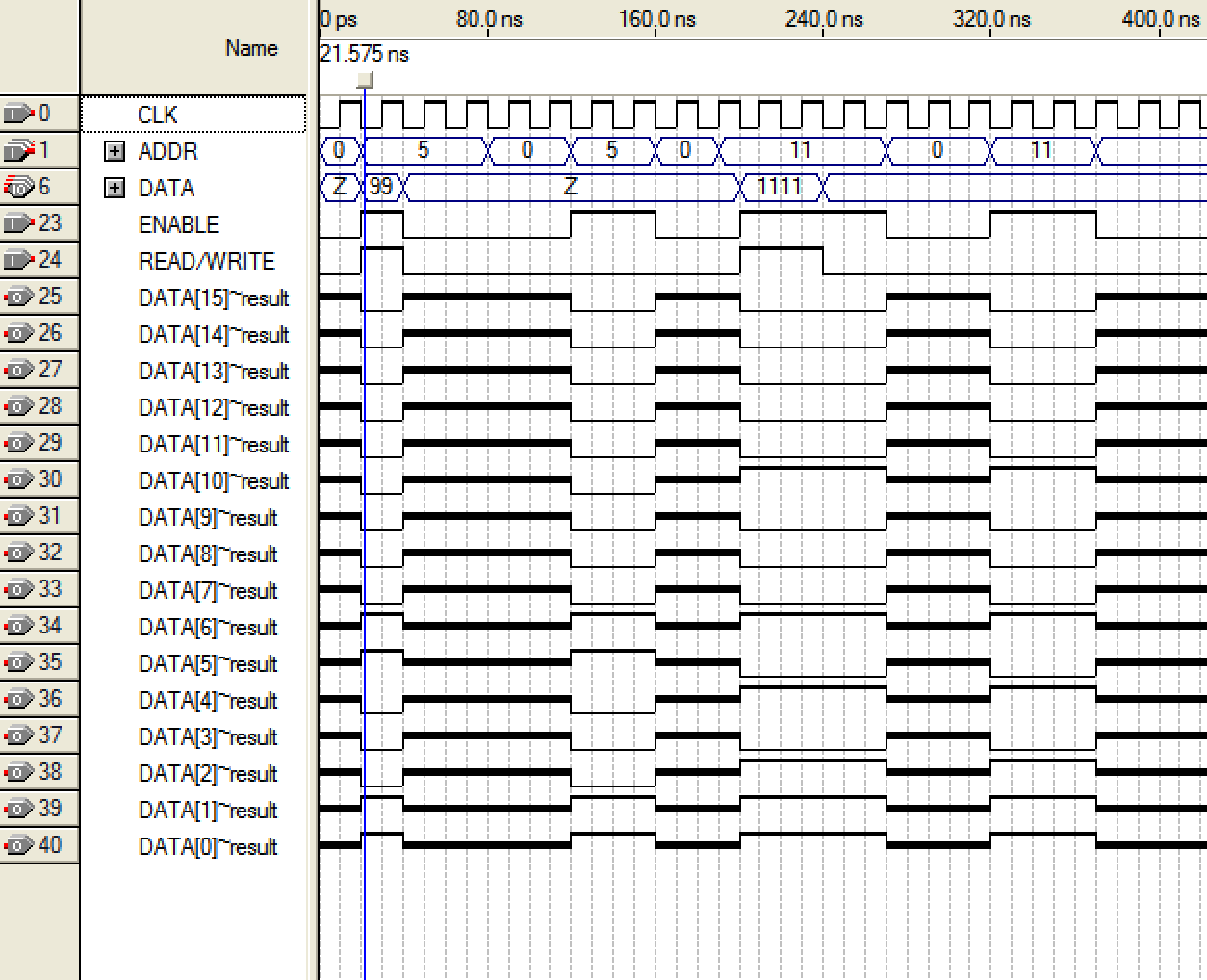


Рисунок 12. Функциональное моделирование блока РОН.

На отрезке 20-40 мы пишем в 5 регистр значение 99. Далее на отрезке 120-160 мы читаем значение из 5 регистра и убеждаемся что оно равно 99. Аналогичные действия происходят с 11 регистром на отрезка 200-240 и 320-370. Стоит отметить что операции чтения и записи происходят значительно быстрее в сравнении с блоком памяти.

* 1. Стек.

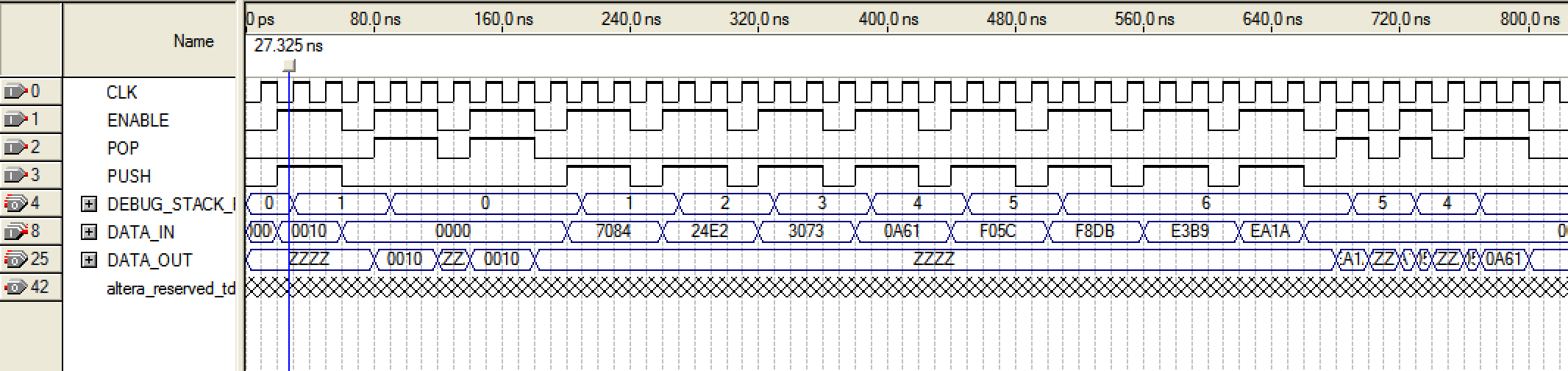


Рисунок 13. Функциональное моделирование стека.

На интервале 20-60 происходит запись в стек значения 0010. Стоит отметить что на диаграмме видно изменение указателя вершины стека. После записи на интервалах 80-120 и 140-160 происходит считывание из стека. Стоит отметить что оба результата выдают значение 0010, хотя в стек было записано всего одно значение. Дело в том что стек обладает своеобразной защитой, что будет также показано далее. На интервале 210-510 в стек происходит запись 5 значений, а 610-690 происходит 3 попытки записать значения, которые вполне успешны, хотя стек достиг своего максимального размера. Как указано было ранее новые значения просто переписывают последнее при достижении максимального размера стека. В этом мы можем убедиться на интервале 680-800.

* 1. Арифметико-логическое устройство.

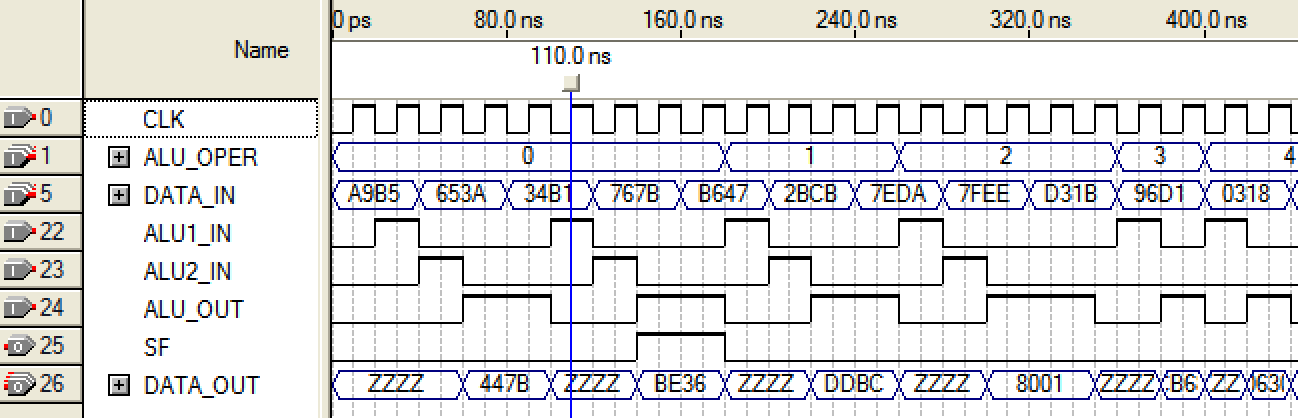


Рисунок 14. Функциональное моделирование АЛУ.

На интервале 0-180 дважды проверяется команда SUB. В первом случае очевидно первый операнд больше второго и флаг SF не выставляется, а во втором ситуация противоположная и мы можем заметить выставленный флаг. Далее по очереди проверяется каждая из 5 команд АЛУ.

* 1. Тестовая программа.

Для нормального функционирования устройства нам необходимо заполнить наши ПЗУ и ОЗУ соответствующими данными. Ниже представлены данные, хранящиеся в устройстве.

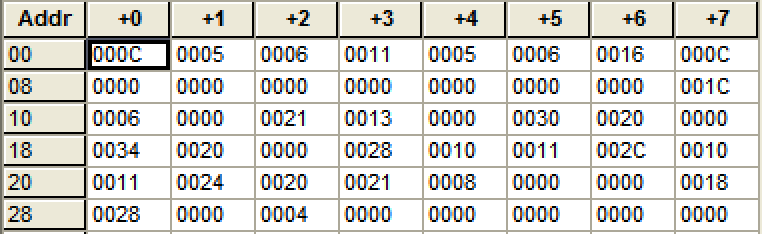


Рисунок 15. Содержимое ПЗУ команд.

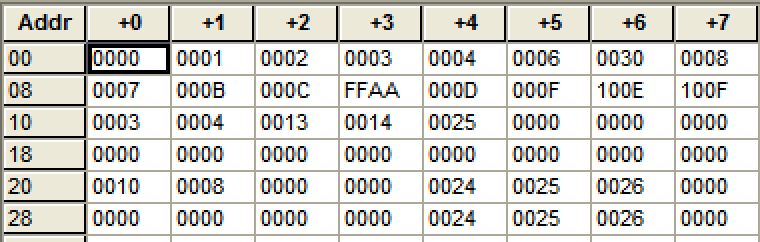


Рисунок 16. Содержимое ПЗУ данных.

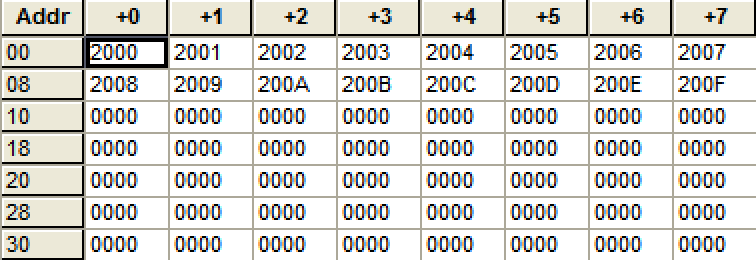


Рисунок 17. Содержимое ОЗУ (до моделирования).

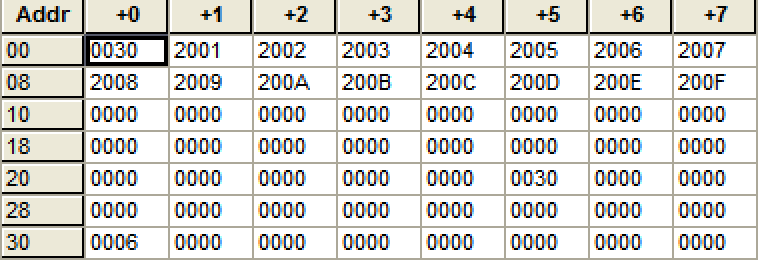


Рисунок 18. Содержимое ОЗУ (после моделирования).

Таблица 5. Тестовая программа.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Комманда | Адресация | Оп.1 | Оп.2 | Hex | Hex | Hex |
| 1 | MOV adr, reg | Прямая | 5 | 6 | 000C | 0005 | 0006 |
| 2 | MOV reg, adr | Косвенная | 5 | 6 | 0011 | 0005 | 0006 |
| 3 | JMP adr | Косвенная x2 | 12 | 0 | 0016 | 000C | 0000 |
| 4 | PUSH adr | Прямая | 6 | 0 | 001C | 0006 | 0000 |
| 5 | POP adr | Косвенная | 19 | 0 | 0021 | 0013 | 0000 |
| 6 | ROL reg | Прямая | 32 | 0 | 0030 | 0020 | 0000 |
| 7 | ROR reg | Прямая | 32 | 0 | 0034 | 0020 | 0000 |
| 8 | NAND reg, reg | Прямая | 16 | 17 | 0028 | 0010 | 0011 |
| 9 | NOR reg, reg | Прямая | 16 | 17 | 002C | 0010 | 0011 |
| 10 | SUB reg, reg | Прямая | 32 | 33 | 0024 | 0020 | 0021 |
| 11 | NOP | Прямая | 0 | 0 | 0008 | 0000 | 0000 |
| 12 | JMS adr | Прямая | 40 | 0 | 0018 | 0028 | 0000 |
| 13 | HLT | Прямая | 0 | 0 | 0004 | 0000 | 0000 |

Проанализируем работу программы. Первые 2 команды последовательно копирую сначала значения из памяти в регистра, потом из регистра в памяти. Результаты записи в память можно видеть на рисунке 18. Потом осуществляется безусловный переход. Далее проверяется работа стека, путем помещения туда значения и считывания его оттуда. Далее проверяется работа каждой из команд АЛУ. Далее выполняется команда ожидания и условного перехода. Если условие условного перехода будет выполнено ( флаг S=1 ) тогда программа начнет свое выполнение заново, если нет – то устройство остановит свою работу.

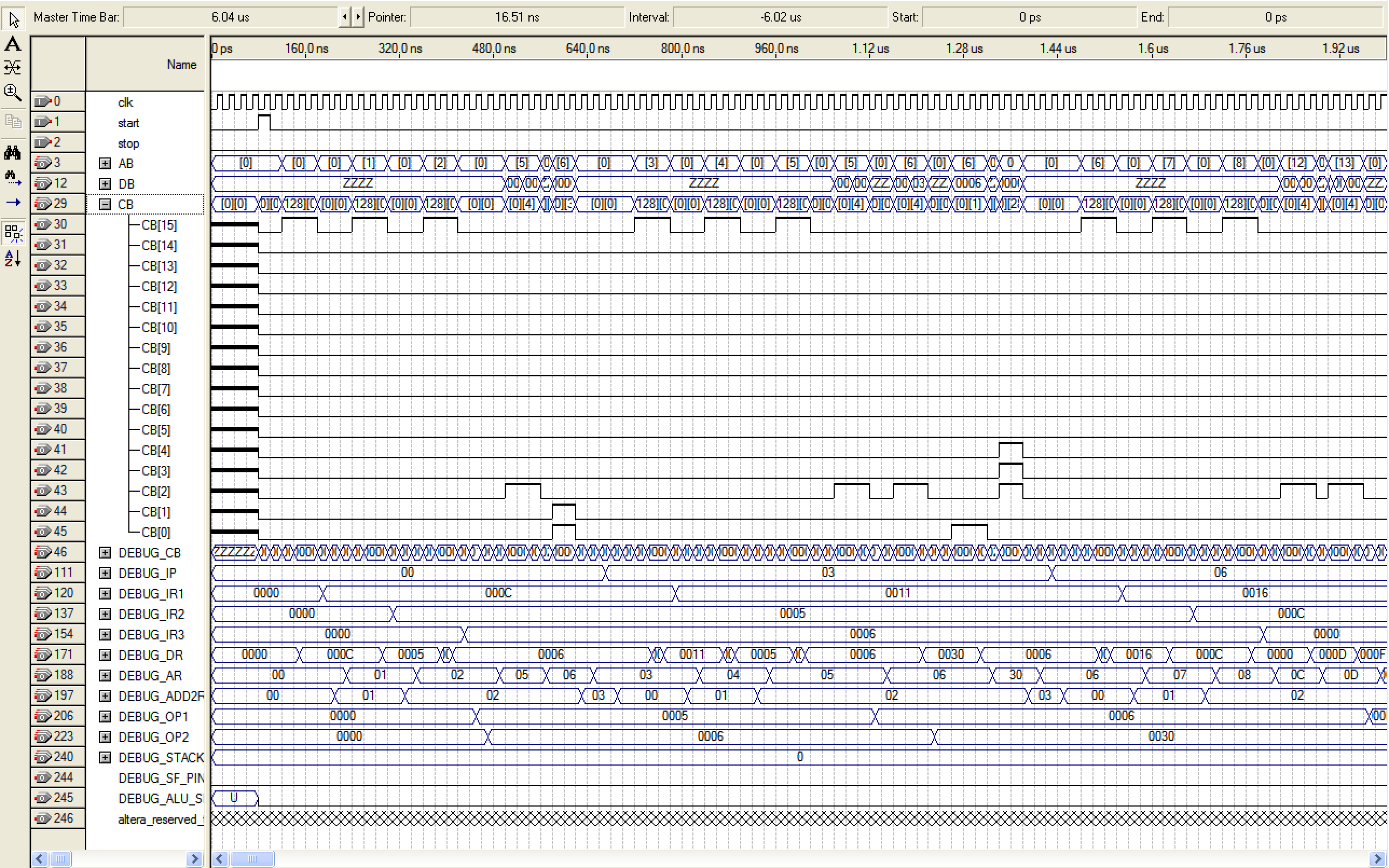


Рисунок 19. Функциональное моделирование работы микро-ЭВМ.

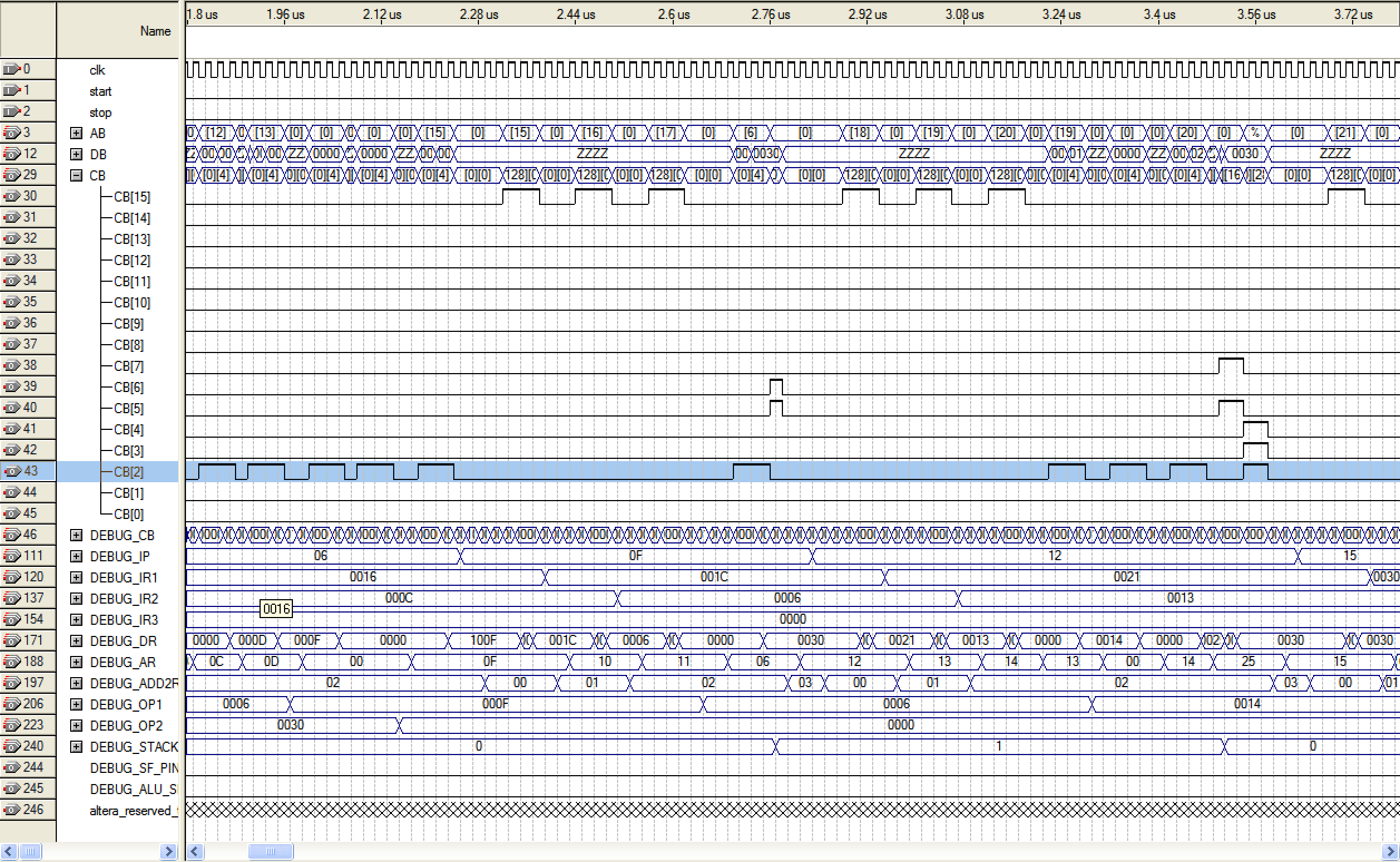


Рисунок 20. Функциональное моделирование работы микро-ЭВМ(продолжение).

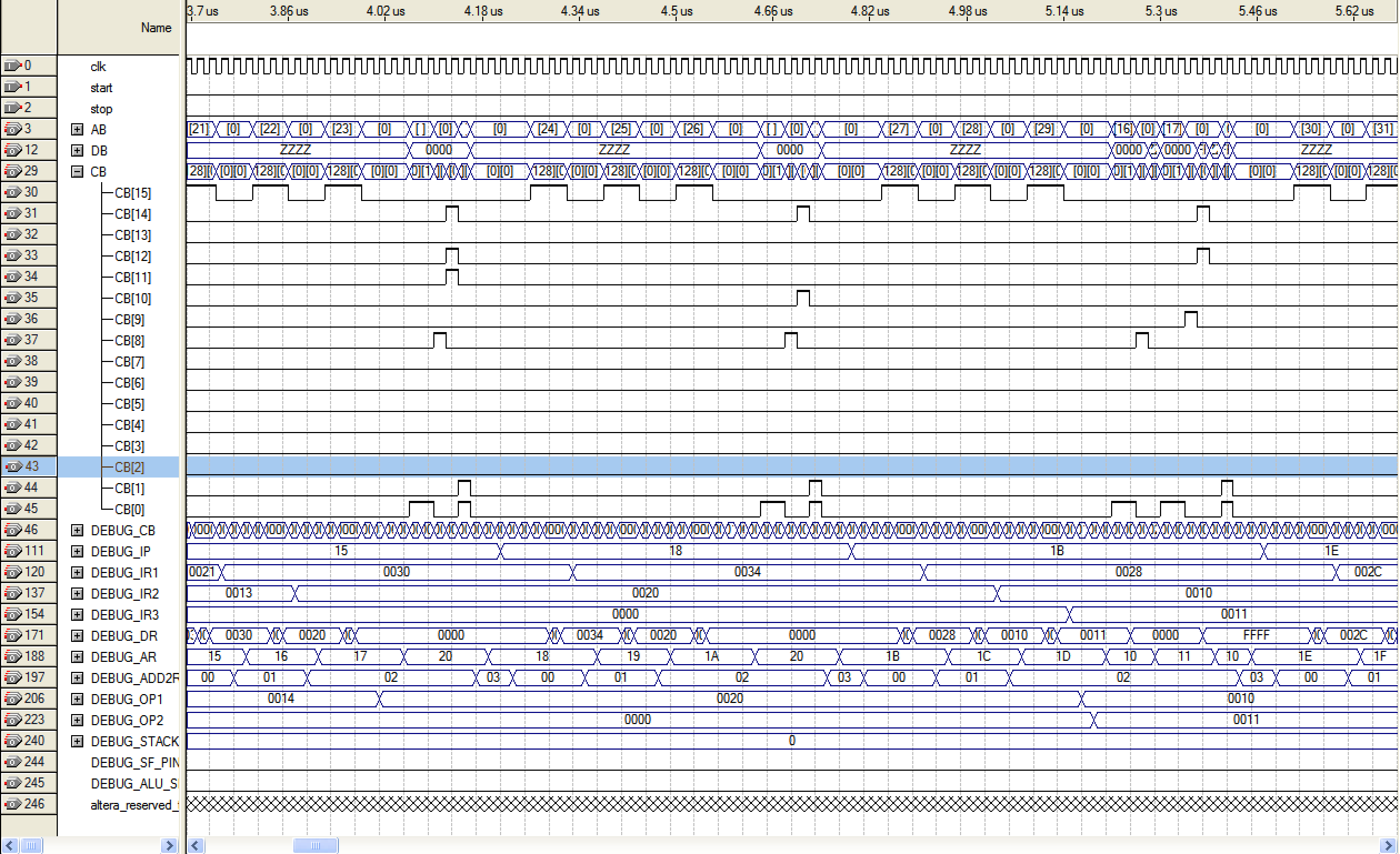


Рисунок 21. Функциональное моделирование работы микро-ЭВМ(продолжение).

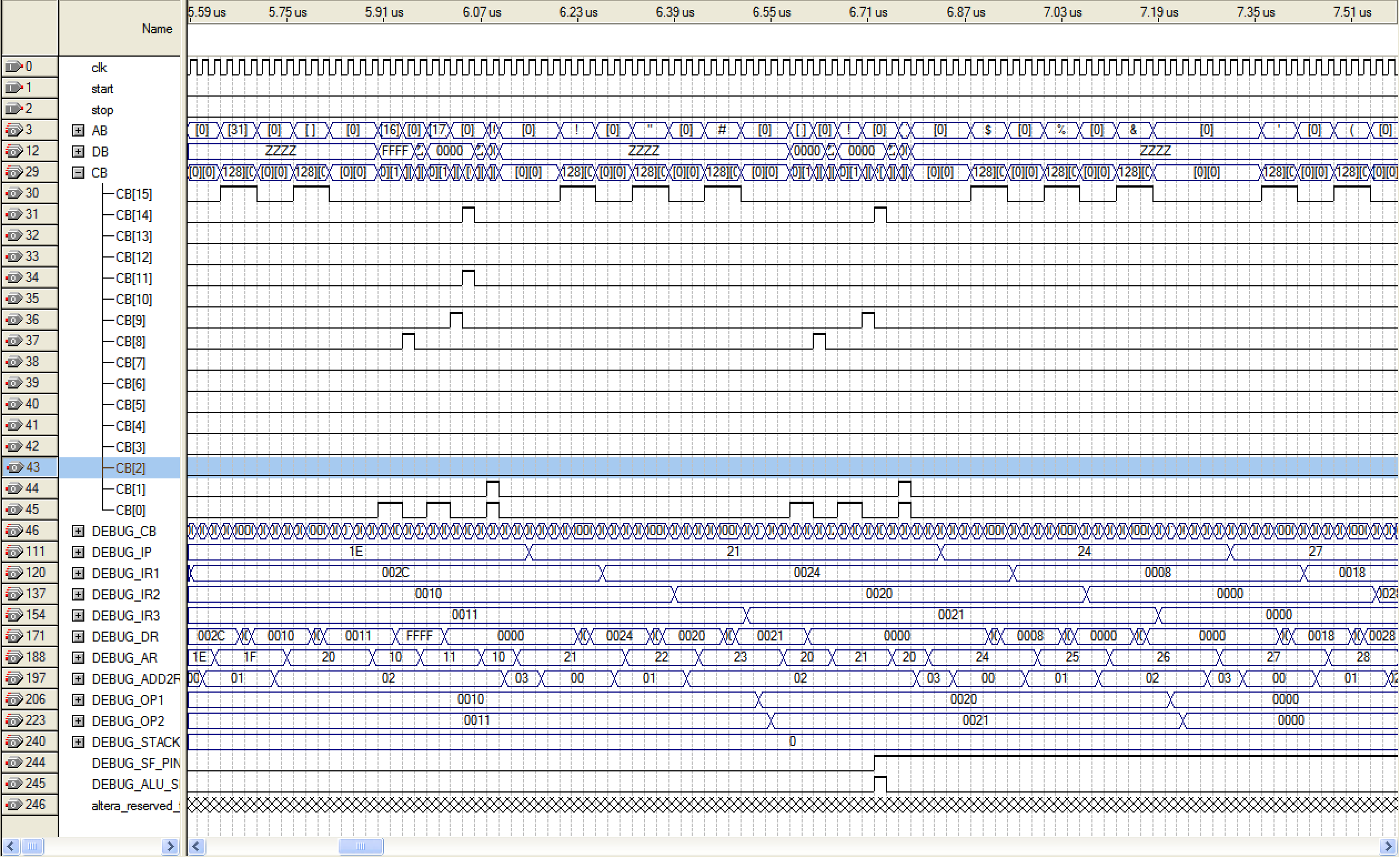


Рисунок 22. Функциональное моделирование работы микро-ЭВМ(продолжение).

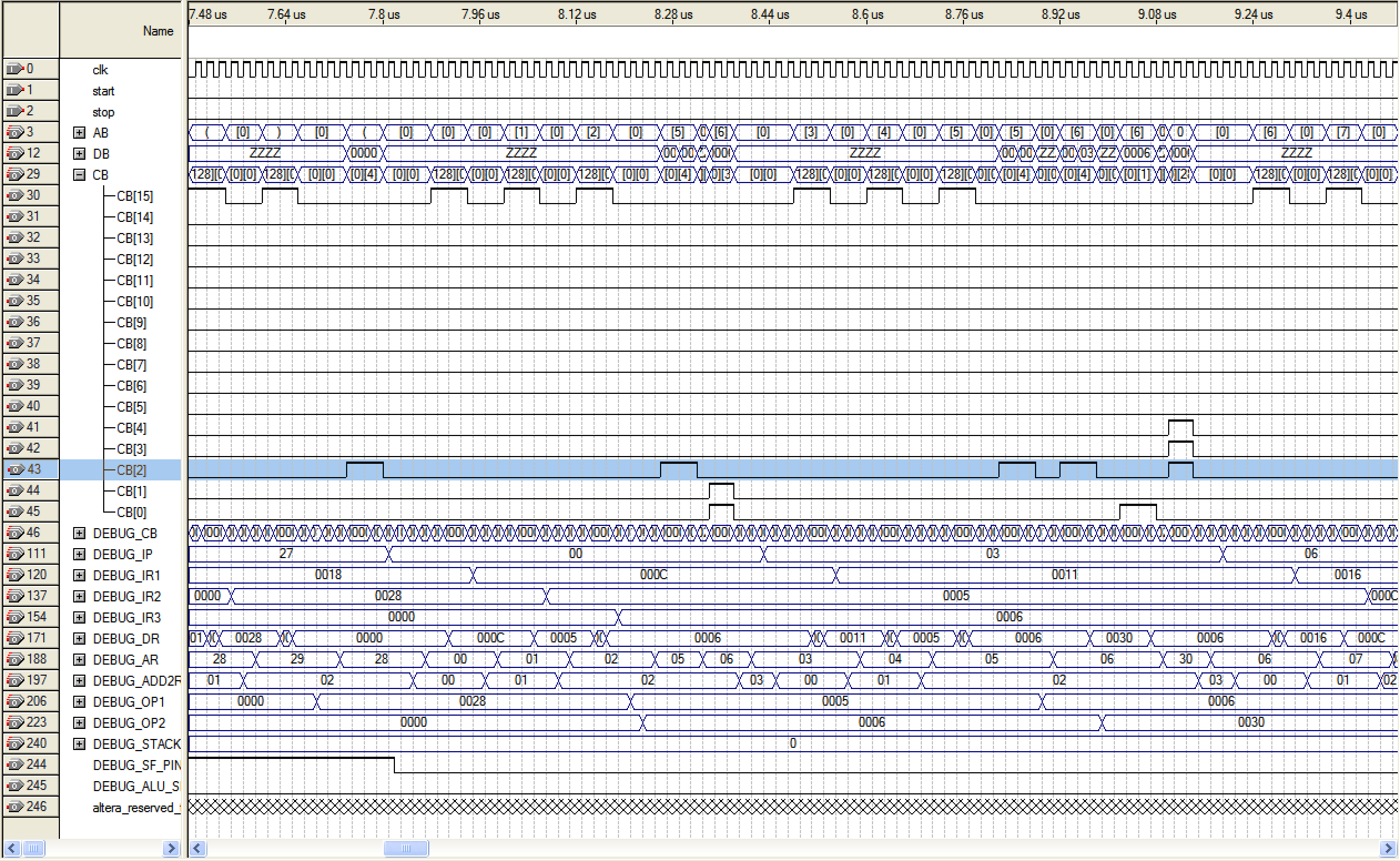


Рисунок 23. Функциональное моделирование работы микро-ЭВМ(продолжение).

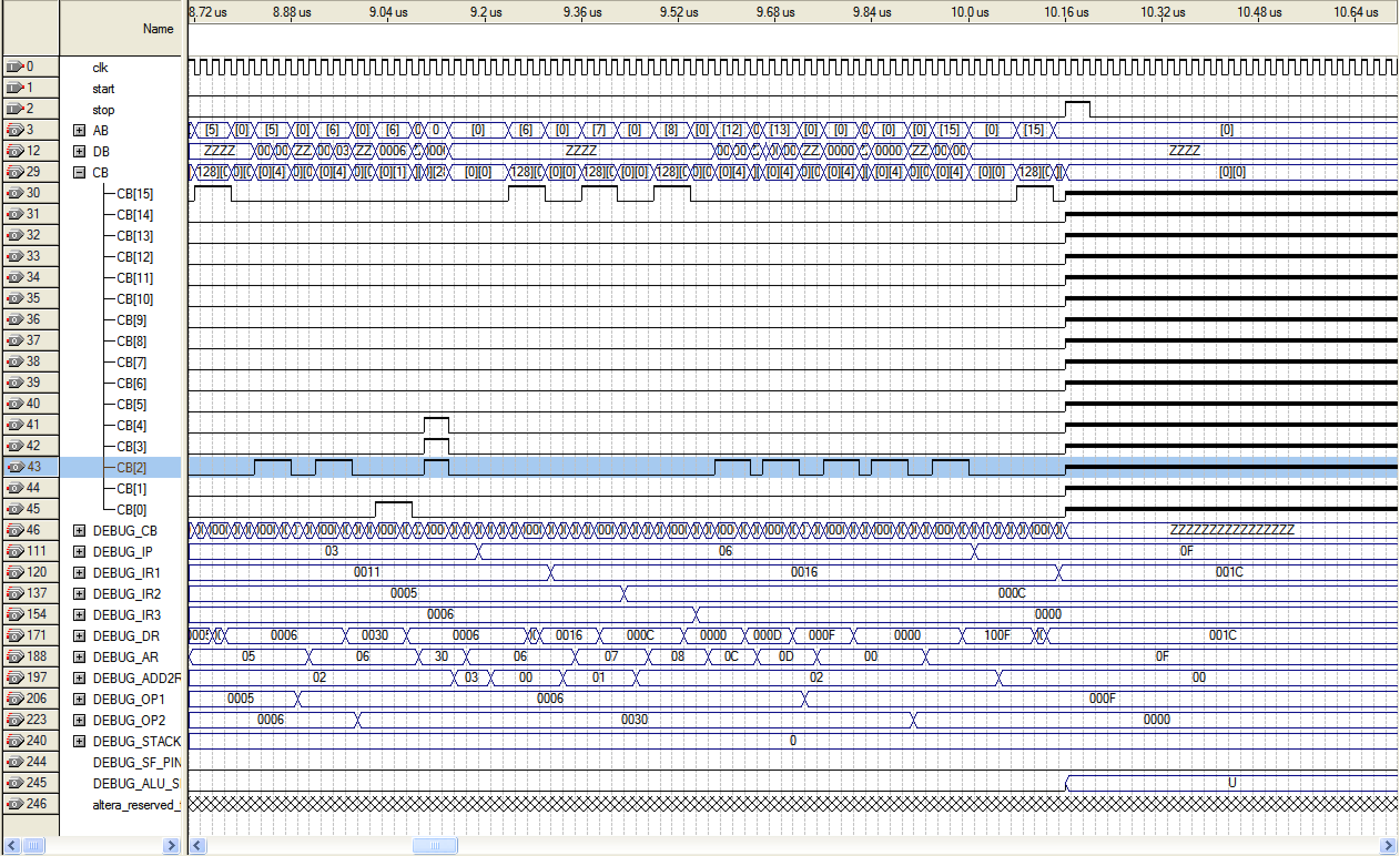


Рисунок 24. Функциональное моделирование работы микро-ЭВМ(продолжение).

1. АНАЛИЗ И ОПТИМИЗАЦИЯ РАЗРАБОТАННОЙ МИКРО-ЭВМ

В ходе работы было создано микро-ЭВМ, способное выполнять 13 различных команд, с возможностью выбора одного из 3 предложенных способов адресации.

Как видно из диаграмм выше самые затратные стадии выполнение каждой из команд – это стадия загрузки команды и стадия загрузки операндов. Стадия загрузки команды занимает в среднем 20 тактов, а стадия загрузки операндов занимает от 2 до 19 тактов в зависимости.

В целом самая длинная команда MOV adr, reg при выборе косвенной адресации длится 49 тактов, а самая короткая JMP adr при выборе прямой адресации 28 тактов.

Как видно из вышеперечисленного пространство для оптимизации в данном устройстве имеется с лихвой. Давайте рассмотрим некоторые шаги, которые могу ускорить работу данного микро-ЭВМ:

* **Использование переменной длины команд**. Как видно самым длительными является этап выборки команды из памяти. Все дело в том что при проектировании нами была выбрана фиксированная длина команд, в связи чем мы вынуждены трижды грузить из памяти блок в 16 бит. Что является не оптимальным решением, так как многие команды не требует 2 операндов или вовсе не требуют операндов. Использование переменной длины команд могло бы сократить количество необходимых тактов до 12 и даже до 6 тактов. Однако стоит иметь ввиду что это значительно усложнит внутреннюю структуру устройства управления.
* **Использование кэш-памяти.** Как известно обращение к памяти является самой медленной операцией. Ввод в микро-ЭВМ кэша помог бы значительно ускорить выполнение операций чтения и записи. Стоит отметить что из-за гарвардской архитектуры в нашем микро-ЭВМ присутствует два ПЗУ, однако использование кэш-памяти для ПЗУ команд не имеет смысла, так как обычно считывание команд происходит последовательно, исключая разве что случаи при выполнении команд условного и безусловного перехода.
* **Использование конвейера команд.** Сейчас при работе устройства все инструкции выполняются последовательно, таким образом многие устройства простаивают большую часть времени. Ввод конвейера частично решил бы данную проблему, однако потребовал бы значительного усложнения структуры микро-ЭВМ.
* **Совмещение сигнала конца команды с предыдущим сигналом.** Дело в том что сейчас каждый этап оканчивается специальной командой STEP\_FINISHED на который тратится дополнительный такт. Так как устройство содержит 4 этап, то целых 4 такта процессорного времени тратятся на простой переход между фазами.
* **Использование контроллера прямого доступа в памяти.** В данной архитектуре системы команда нету команды которая копирует значение из ячейки памяти в другую ячейку памяти. Для этого приходится использовать 2 команды: MOV adr, reg и MOV reg, adr. Это в свою очередь займет порядка 64 тактов. Данную проблему можно решить путем ввода дополнительной команды MOV adr, adr или же вводом контроллера прямого доступа в память. Он позволит разгрузить устройство управления.

ЗАКЛЮЧЕНИЕ

В ходе работы удалось реализовать полноценную микро-ЭВМ. При разработке возникало множество конфликтов и задач, к которым приходилось находить творческий подход. Конечно реализованная микро-ЭВМ не может конкурировать на рынку существующих, однако она с достоинством может занять место в качестве обучающего пособия для абитуриентов и студентов, увлекающимися проектированием ЭВМ. Еще раз перечислим основные характеристики:

1. Гарвардская архитектура;
2. 2 ПЗУ объемом 256 слов;
3. ОЗУ объемом 256 слов;
4. 16 регистров общего назначения;
5. Стек, размером 6 слов;
6. 13 различных команд.

Также в архитектуру ЭВМ были заложены возможности расширения, что позволяет продолжить работу над устройством в дальнейшем.