СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данном курсовом проекте предполагается реализовать микро-ЭВМ по принципу фон Неймана с характеристиками, указанными в листе задания.

Электронно-вычислительная машина – это комплекс технических и программных средств, предназначенных для автоматизации подготовки и решения задач пользователей. ЭВМ может при помощи вычислений производить обработку информации по определенному алгоритму. Помимо этого, программное обеспечение позволяет компьютеру хранить, принимать и искать информацию и, в дальнейшем, выводить ее на различные устройства ввода.

Схему ЭВМ предложил в 1949 году математик Джон фон Нейман, включающую в себя арифметико-логическое устройство, запоминающее устройство, устройство управления и устройства ввода-вывода информации, и с тех пор принцип устройства почти не изменился. В течение всего периода эволюции компьютерных систем прослеживается тенденция к повышению скорости обработки информации процессором, уменьшение физических размеров компонентов, росту объема памяти и повышению пропускной способности каналов ввода-вывода.

Не отрицая того факта, что одной из причин повышения производительности процессоров явился прогресс в области микроэлектроники, в частности миниатюризация электронных компонентов, следует отметить, что существенное влияние на этот процесс, особенно в последние годы, оказали новые идеи в отношении структурной организации процессора, в частности широкое использование принципов конвейерной и параллельной обработки и внедрение технологии предпочтительного выбора направления ветвления программы. Все эти идеи преследуют одну цель – максимально сократить время простоя процессора и, следовательно, увеличить его производительность.

Для разработки проекта используется система автоматизируемого проектирования (САПР) Altera Quartus II v9.1. САПР Quartus II предназначена для проектирования устройств с высокой степенью интеграции, объединяющий в себе проектирование, синтез, размещение элементов, трассировку соединений и верификацию, связь с системами проектирования других производителей.

1 РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ

При написании данной работы были использованы научная и учебно-методическая литература, справочный и практический материал и схемы микро-ЭВМ, находящиеся в открытом доступе. Исходя из полученных данных далее представлено описание основных блоков, необходимых для исправного функционирования данного проекта.

1.1 Функциональный состав микро-ЭВМ

Существуют два взгляда на построение и функционирование ЭВМ. Первый - взгляд пользователя, не интересующегося технической реализацией ЭВМ и озабоченного только получением некоторого набора функций и услуг, обеспечивающих эффективное решение его задач; второй - разработчика ЭВМ, усилия которого направлены на рациональную техническую реализацию необходимых пользователю функций [1]. В данном подразделе описывается функциональный состав разрабатываемой микро-ЭВМ.

Основной задачей реализуемого проекта является решение вычислительных задач. Процесс решения задач на компьютере – это совместная деятельность человека и ЭВМ. На долю человека приходятся этапы, связанные с творческой деятельностью – постановкой, алгоритмизацией, программированием задач и анализом результатов, а на долю персонального компьютера – этапы обработки информации в соответствии с разработанным алгоритмом [2]. Исходя из вышеперечисленного – ЭВМ должна включать необходимые для выполнения задач блоки: хранения данных с последовательным или произвольным доступом и блок выполнения операций.

Исходя из вышеописанного, набор блоков для проектируемой микро-ЭВМ следующий:

1Блок памяти. Представляет собой общее адресное пространство команд и данных (принстонская архитектура) и кэш, который взаимодействует непосредственно с блоком выполнения операций.

2Блок выполнения операций (CPU). Данный блок включает в себя устройство управления и арифметико-логическое устройство.

Ниже представлено более подробное описание каждого блока.

Блок памяти включает в себя ПЗУ с асинхронным (данные передаются в любой момент времени) процессом доступа к данным, ОЗУ с синхронным (все процессы при выполнении операций записи и чтения данных согласованы во времени с тактовой частотой центрального процессора) процессом и стека – непрерывную область основной памяти, занимающую крайние адреса и адресуемую регистрами. Особенность стека заключается в том, что данные в него помещаются и извлекаются по принципу FILO [3].

Память является «узким» местом современной ЭВМ. Для решения данной проблемы и достижения быстродействия микропроцессора используется иерархия памяти, на различных уровнях которой расположены

памяти с отличающимися временем доступа, сложностью, стоимостью и объемом. Возможность построения иерархии памяти вызвана тем, что большинство алгоритмов обращаются в каждый промежуток времени к небольшому набору данных, который может быть помещен в более быструю, но дорогую и поэтому небольшую память [4]. Для реализации иерархии памяти в данном курсовом проекте использует, как уже упоминалось ранее, основная памяти, а также следующие виды:

1Кэш-память. Высокоскоростная память произвольного доступа, используемая процессором компьютера для временного хранения информации. В данном курсовом проекте реализуется множественно-ассоциативный кэш – кэш-память делится на несколько банков, каждый из которых функционирует как кэш с прямым отображением, таким образом строка ОЗУ может быть отображена не в единственную возможную запись кэша, а в один из нескольких банков; выбор банка осуществляется на основе LRU.

2Регистры процессора. Сверхоперативная память небольшого размера, которая предназначена для временного хранения служебной информации или данных. Регистры общего назначения являются основными рабочими регистрами, предназначены для хранения операндов арифметико-логических инструкций, а также адресов или отдельных компонентов адресов ячеек памяти. Специализированные регистры – это регистр-счетчик команд, регистры флагов и регистр указателя стека, каждый из которых имеет определенное функциональное назначение [5].

Для вычислительной части ЭВМ разрабатывается два основных блока:

1Устройство управления. Предназначено для выработки управляющих сигналов, под воздействием которых происходит преобразование информации в арифметико-логическом устройстве, а также операции по записи и чтению информации в/из запоминающего устройства.

2Арифметико-логическое устройство. Блок процессора, управляемый УУ. Его предназначение – выполнение логических и арифметических преобразований над данными-операндами. Помимо вычислений, в АЛУ присутствуют биты состояния (флаги) – сигналы управления, которые направляют ЭВМ на выбор правильного пути для выполнения необходимого вычислительного процесса в зависимости от итоговых типов данных.

Взаимодействие основных блоков происходит при помощи магистрально-модульного принципа – информационная связь между устройствами компьютера осуществляется через системную шину. Системная шина, в свою очередь, представляет собой совокупность шины данных, по которой выполняется обмен данным между модулями, шины адреса, служащей для определения процессором модуля или ячейки памяти, с которой будет выполняться обмен информационными данным, шины управления служит для передачи управляющих сигналов, определяющих какой тип операции следует исполнить (запись или считывание данных и источник для получения данных, синхронизацию с микропроцессором) [6].

1.2 Разработка системы команд

Важной составной частью архитектуры ЭВМ является система команд. Несмотря на большое число разновидностей ЭВМ, на машинном уровне они имеют много общего. Система команд любой ЭВМ обязательно содержит следующие группы команд обработки информации:

1Команды передачи данных. Копируют информацию из одного места в другое.

2Арифметические операции. Основные действия – сложение и вычитание.

3Логические операции. Позволяют компьютеру производить анализ получаемой информации. После выполнения такой команды, с помощью условного перехода, ЭВМ определяет дальнейший ход выполнения программы.

4Сдвиги двоичного кода влево и вправо. Данные операции позволяют реализовывать трудные арифметические операции, например, умножение.

5Команды ввода и вывода информации для обмена с внешними устройствами.

6Команды управления, реализующие нелинейные алгоритмы. К данным относятся условный и безусловный переход, а также команды обращения к подпрограмме (переход с возвратом).

7Команды управления процессором. Примером данной команды является HLT. Команда прекращает выполнение программы и переводит процессор в состояние останова. Работа процессора возобновляется после операции запуска, а также в случае прихода немаскируемого или разрешенного маскируемого прерываний.

В данном курсовом проекте количество команд для реализации равно 25, следовательно, необходимо использовать минимум пять бит (разрядов) для обозначения кода операции; для передачи адреса, согласно разрядности шины адреса, необходимо выделить 12 бит для каждого операнда. Исходя из вышеперечисленного, постоянный размер команды состоит из четырех машинных слов данных фиксированного размера, который равен разрядности шины данных. Данный блок слов обрабатывается УУ как единое целое с помощью блоков дешифратора команд и генератора сигналов управления.

Команда состоит из 32 бит, структура которой представлена ниже:

КодОперации (31-27 биты) (28-24 биты, не влияющие на команду) Операнд1 (23-12 биты), Операнд2 (11-0 биты).

Шина адреса, согласно листу задания, определена разрядностью 12 бит, соответственно, для непосредственной адресации участка памяти в команде будет занимать все 12 битов операнда, количество РОН – 16 – для адресации регистра необходимо 3 бит операнда, размерность стека не влияет на размерность команды из-за отсутствия непосредственной адресации.

В данном курсовом проекте, на основе установленных данных, система команд будет иметь вид, представленный в таблице 1.1.

Таблица 1.1 – Кодирование операций в микро-ЭВМ

Команда	Тип адресации	Код операции
MOV reg, $mem	Прямая регистровая	00000
MOV $mem, reg	Прямая регистровая	00001
PUSH reg	Прямая регистровая	00010
POP reg	Прямая регистровая	00011
JMP $mem	Непосредственная	00100
JBZ $mem	Непосредственная	00101
CMP $mem, $mem	Прямая	00110
AND $mem, $mem	Прямая	00111
OR $mem, $mem	Прямая	01000
ROR $mem, $mem	Прямая	01001
MOV reg, reg	Прямая регистровая	01010
CMP reg, $mem	Прямая регистровая	01011
AND reg, $mem	Прямая регистровая	01100
OR reg, $mem	Прямая регистровая	01101
ROR reg, $mem	Прямая регистровая	01110
HLT	–	01111
MOV reg, si + $mem	Базовая индексная	10000
MOV si +$mem, reg	Базовая индексная	10001
MOV reg, bx + si + $mem	Базовая индексная со смещением	10010
MOV bx + si + $mem, reg	Базовая индексная со смещением	10011
JB $mem	Непосредственная	10100
JNZ $mem	Непосредственная	10101
CMP reg, reg	Прямая регистровая	10110
INC reg	Прямая регистровая	10111
DEC reg	Прямая регистровая	11000

В данном курсовом проекте необходимо разработать 16 базовых команд согласно листу задания и дополнительно 9 команд, с дополнительными типами адресации, для реализации микропрограммы сортировки массива данных.

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

Программа в ЭВМ реализуется ЦП с использованием вспомогательных блоков, представленных в приложении А, посредством последовательного исполнения образующих эту программу команд. Действия, требуемые для выборки (извлечения из основной памяти) и выполнения команды, называют циклом команды. Каждая программа состоит из отдельных машинных команд. Каждая машинная команда, в свою очередь, делится на ряд элементарных унифицированных составных частей, которые принято называть тактами. В

зависимости от сложности команды она может быть реализована за разное число тактов. Например, пересылка информации из одного внутреннего регистра процессора в другой выполняется за несколько тактов, а для перемножения двух целых чисел их требуется на порядок больше. Существенное удлинение команды происходит, если обрабатываемые данные еще не находятся внутри процессора и их приходится считывать из ОЗУ.

В общем случае цикл команды включает в себя несколько составляющих (этапов) [8]:

– выборка команды, согласно содержимому счетчика адреса команд, считывается очередная команда программы;

– формирование адреса следующей команды (счетчик команд автоматически изменяется так, чтобы в нем содержался адрес следующей команды);

– декодирование команды;

– вычисление адресов операндов;

– выборка операндов;

– исполнение операции;

– формирование признака результата, установка регистра флагов;

– запись результата.

Затем во всех случаях, за исключением команды останова или наступления прерывания, все описанные действия циклически повторяются. После выборки команды останова HLT ЭВМ прекращает обработку программы. Для выхода из этого состояния требуется либо запрос от внешних устройств, либо перезапуск машины. Следует отметить, что не все из этапов присутствуют при выполнении любой команды (зависит от типа команды), однако этапы выборки, декодирования, формирования адреса следующей команды и исполнения операции имеют место всегда.

Рассмотренный основной алгоритм работы ЭВМ позволяет шаг за шагом выполнить хранящуюся в ОЗУ или ПЗУ линейную программу. Если же требуется изменить порядок вычислений для реализации развилки или цикла, достаточно в счетчик команд занести требуемый адрес, именно так происходит условный или безусловный переход [9].

2 РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ

В данном курсовом проекте реализованы следующие основные функциональные блоки:

– запоминающие устройства, или основная память;
– устройство управления;
– арифметико-логическое устройство;
– кэш-память процессора;
– контроллер прямого доступа к памяти;
– арбитр шины.

Каждый из вышеперечисленных блоков микро-ЭВМ в подробном виде рассмотрен ниже.

2.1 Запоминающие устройства

Запоминающие устройства разделяются на ПЗУ, ОЗУ, РОН и стек предназначенные для записи, хранения и выдачи данных и отличающиеся по быстродействию и типу доступа.

Постоянные ЗУ или односторонние ЗУ предназначены только для хранения и считывания информации, которая не изменяется в процессе вычислений, например, постоянно используемые программы, различные константы и микропрограммы. Оперативные ЗУ или основная память предназначены для хранения данных и программ текущих вычислений.

Блоки ОЗУ и ПЗУ реализованы на основе принстонской архитектуры (имеют общее адресное пространство) с помощью стандартных модулей lpm_rom и lpm_ram_dq размером, который определяется разрядностью шины адреса, по 4096 байт. Условно-графические обозначение блоков памяти представлено на рисунке 2.1.

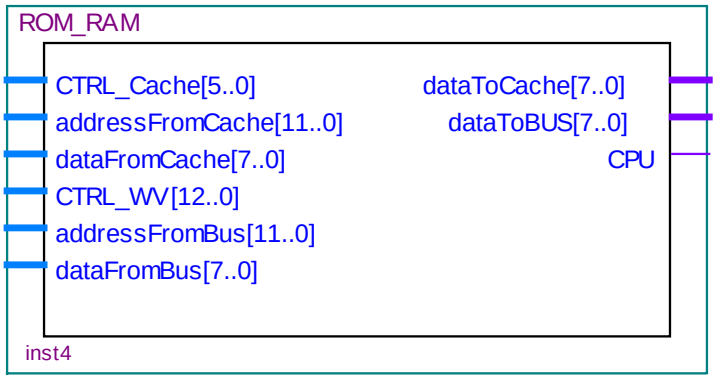

Рисунок 2.1 – Условно-графическое обозначение блока памяти

Блок памяти имеет следующие входы и выходы:

1. CTRL_Cache[5..0] – управляющие данные из кэш-памяти;
2. CTRL[12..0] – управляющий вход по шине управления;
3. addressFromCache[11..0] – адрес для чтения/записи данных из кэша;

4. addressFromBus[11..0] – адрес для чтения/записи по шине адреса;
5. dataFromCache[7..0] –данные для записи RAM из кэш-памяти;
6. dataFromBus[7..0] –данные для записи RAM по шине данных;
7. dataToCache[7..0] – полученные по адресу данные/команды для кэш;
8. dataToBus[7..0] – полученные по адресу данные/команды по шине данных.

Блок ROM_RAM получает с шины управления сигналы чтения для ROM или чтения/записи для RAM по шине управления или слово управления из кэш-памяти, с шины адреса – адрес команды или данных, на вход данных поступают значения, которые в последующем будут записаны в RAM, выход данных или команд на шину данных или напрямую в кэш-память. Дублирование шин адреса, данных и управления для кэш-памяти позволяет реализовать конвейерную оптимизацию без конфликтов на общих шинах. Данный блок работает на CLK-сигнале, поступающем из процессора, что позволяет синхронизировать операции чтения/записи данных. Для выполнения последних необходимо выделить два такта микропроцессора. Функциональная схема ПЗУ и ОЗУ представлена в приложении Б.

Более быстродействennой памятью, находящейся на кристалле микропроцессора является блок РОН. Они предназначаются для хранения ряда чисел, используемых некоторой текущей последовательностью команд программы. В данном курсовом проекте данный блок состоит из 16 регистров разрядностью 8 бит. Условно графическое обозначение блока РОН представлено на рисунке 2.2.

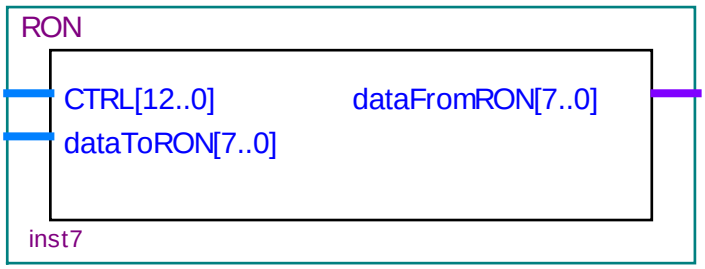

Рисунок 2.2 – Условно-графическое обозначение блока РОН

Блок регистров общего назначения имеет следующие входы и выходы:
1. CTRL[12..0] – управляющее слово полученное по шине управления;
2. dataToRON[7..0] – входные данные для записи в регистры;
3. dataFromRON[7..0] – выходные данные после операции чтения.

Блок РОН получает с шины управления команду для чтения/записи данных и регистров по адресу, который находится в этой команде. Для функционирования данного блока также необходим CLK-сигнал от МП. Время доступа к данным – один такт процессора, что в два раза меньше чем в ОЗУ или ПЗУ. Функциональная схема РОН представлена в приложении В.

Заключительным блоком типа запоминающих устройств является стек – абстрактная часть основной памяти, занимающая крайние верхние или нижние

адреса. Стек в данном курсовом проекте реализован по принципу LIFO с направление роста вниз (от старших адресов к младшим) и размерностью 11 ячеек. Условно графическое обозначение блока стека представлено на рисунке 2.3.

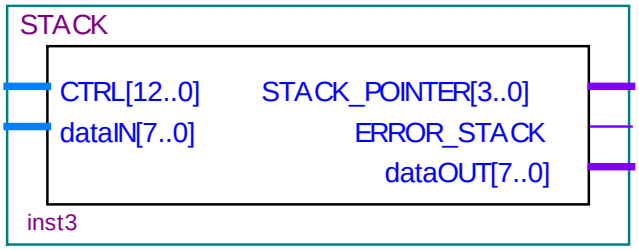

Рисунок 2.3 – Условно-графическое обозначение блока стека

Блок стека управляется УУ при помощи слов управления – посылаются основные команды PUSH и POP. В зависимости от последней выполненной команды изменяется внутренний регистр стека, называемый указателем стека, который содержит адрес текущей ячейки. Функциональная схема блока стека представлена в приложении Г.

2.2 Устройство управления

Устройство управления предназначено для выработки управляющих сигналов, под воздействием которых происходит преобразование информации в арифметико-логическом устройстве, а также операции по записи и чтению информации в/из запоминающего устройства. Данный блок отправляет на шину управления сигналы для правильной и синхронной работы остальных блоков системы.

УУ имеет внутренние регистры на которых хранится выполняемая команда и адрес следующей. Также в УУ находится главный счетчик микро-ЭВМ, который определяет частоту работы процессора и количество тактов, которые отводятся на выполнение каждой команды. В данном блоке поэтапно происходит обработка каждой поступившей команды: декодирование, получение операндов, их вычисление и формирование адреса следующей команды. Условно графическое обозначение УУ представлено на рисунке 2.4.

УУ организует работу всей системы в зависимости от этапа исполняемой команды. Данный блок непосредственно соединен с АЛУ и отправляет на последний отдельную команду управления, организует запись и хранение данных в АЛУ и, соответственно, их обработку и вывод на общую шину данных.

Отдельными элементами являются пины для организации арбитража шин – УУ, при необходимости, посылает запрос на предоставление шины и, имея самый высокий приоритет, согласно определению последовательного арбитража, в основном использует общую шину. Функциональная схема УУ представлена в приложении Д.

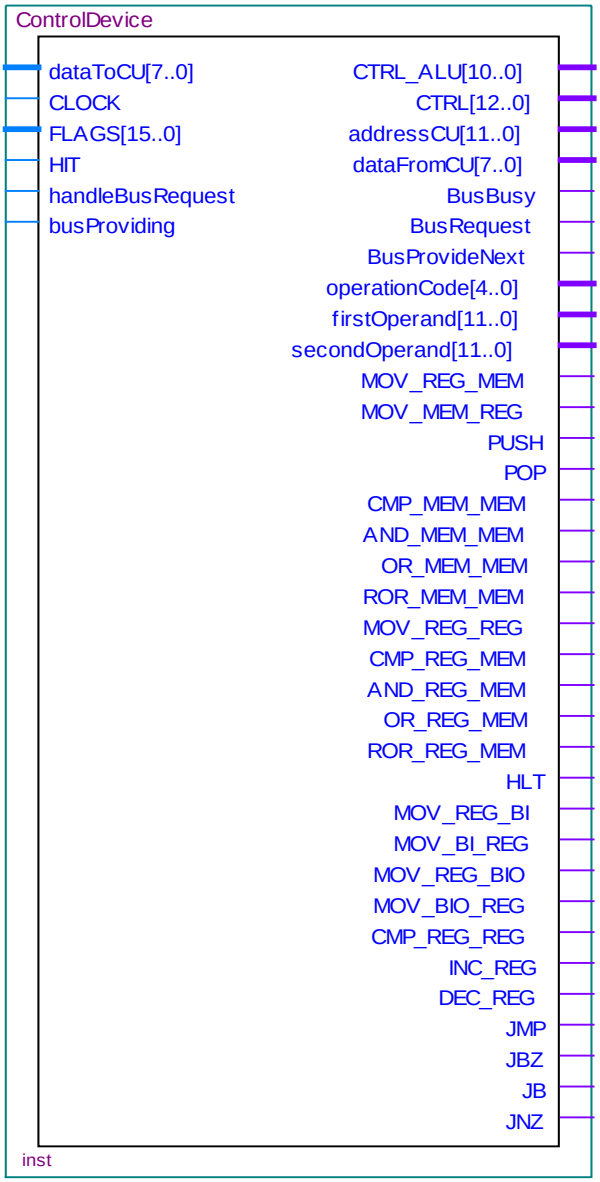

Рисунок 2.4 – Условно-графическое обозначение блока УУ

Данный блок имеет следующие пины:

1. CLOCK – вход синхросигнала;
2. dataToCU[7..0] – вход данных/команд по шине данных;
3. FLAGS[15..0] – вход регистра флагов из АЛУ;
4. HIT – сигнал из кэш-памяти о готовности предоставить данные;
5. handleBusRequest– запрос на предоставление общей шины;
6. busProviding – сигнал предоставления шины;
7. CTRL_ALU[10..0] – команда управления на для АЛУ;
8. CTRL[12..0] – команда управления на шину управления;
9. addressCU[11..0] – адрес для чтения/записи на шину адреса;
10. dataFromCU[7..0] – обработанные данные на шину данных;
11. operationCode[4..0] – вывод кода выполняемой операции;
12. firstOperand[11..0] – вывод первого операнда операции;
13. secondOperand[11..0] – вывод второго операнда операции;

14. BusBusy – сигнал о том, что УУ заняло общую шину;

15. BusRequest – сигнал запроса шины для арбитра шин;

16. BusProvideNext– сигнал для предоставления шин следующим устройсвам;

17. MOV_REG_MEM, MOV_MEM_REG, PUSH, POP, JMP, JBZ, CMP_MEM_MEM, AND_MEM_MEM, OR_MEM_MEM, ROR_MEM_MEM, MOV_REG_REG, CMP_REG_MEM, AND_REG_MEM, OR_REG_MEM, ROR_REG_MEM, HLT, MOV_REG_BI, MOV_BI_REG, MOV_REG_BIO, MOV_BIO_REG, CMP_REG_REG, INC_REG, DEC_REG, JB, JNZ – вывод выполняемой команды.

2.2.1 Центральный узел распределения управляющих сигналов

Генератор команд получает информацию о выполняемой команде из дешифратора команд после ее непосредственного получения из ROM и, в зависимости от нее, выставляет соответствующие биты на шину управления, например, для чтения из РОН или записи в стек. Данный включает в себя основной счетчик с помощью которого каждой команде выделяется необходимое количество тактов, несколько вспомогательных счетчиков для реализации работы с кэш-памятью, а также внутренние сигналы, которые позволяют установить правильный этап выполнения каждой команды в зависимости от структуры последней.

Рисунок 2.5 – Условно-графическое обозначение блока генерации команд

1. CLK – тактовый вход;

2. operandAddressTO[11..0] – адрес для записи операнда;

3. operandAddressFROM[11..0] – адрес для чтения операнда;

4. MOV_REG_MEM, MOV_MEM_REG, PUSH, POP, JMP, JBZ, CMP_MEM_MEM, AND_MEM_MEM, OR_MEM_MEM, ROR_MEM_MEM, MOV_REG_REG, CMP_REG_MEM, AND_REG_MEM, OR_REG_MEM, ROR_REG_MEM, MOV_REG_BI, MOV_BI_REG, MOV_REG_BIO, MOV_BIO_REG, CMP_REG_REG, INC_REG, DEC_REG, JB, JNZ – вывод выполняемой команды;

5. HIT – сигнал доступности данных в кэш-памяти;

6. BusBusy – сигнал, о том, что шина занята УУ;

7. START_SET – вывод сигнала установки начальных значений;

8. addressBasixEN, instructionAddressEN, instructionDecoderEN, instructionInitEN, operationRetrievalEn, executionStageEN, addressBasicEN recordingAluResultsEN, aluWorkPeriodEN – сигналы для контроля выполнения команды;

8. CTRL[12..0] – выходные данные для шины управления;

9. CTRL_ALU[10..0] – выходная данные для управления АЛУ;

10. CLEAR – сигнал очистки системы;

11. busBusy – шина еще используется УУ.

Данный блок осуществляет основную функцию УУ – управление работой всей микро-ЭВМ. Правильно организованная работа позволяет множеству устройств использовать общую шинную магистраль без перебоев и конфликтов. Условно графическое обозначение блока генерации команд представлено на рисунке 2.5.

Для увеличения производительности каждой команде отведено определенное количество тактов микропроцессора, в зависимости от выполняемой функции, например, команда безусловного перехода выполняется в 1 такт, если не считать этап получения и дешифровки команды, а команда логического ИЛИ в АЛУ – за 3 такта, ввиду необходимости получения большего числа операндов и последующей записи их обратно в память.

2.2.2 Декодер команд

Блок декодера команд преобразовывает полученные из ROM данные в код операции и, затем, в команду, а также определяет операнды. Данный блок основывается на lpm_decode, который на основании архитектуры системы команд расшифровывает код операции и устанавливает сигнал выполняемой операции, и внутреннем регистре, который хранит полученную команду до ее выполнения. Дополнительно данный блок разделяет непрерывный поток данных из ROM на первый и второй операнды

Данный блок ожидает поступления сигнала instructionDecoderEN и instructionInitEn, чтобы избежать ошибок определения КОП, и после его

поступления выполняет свою непосредственную функцию. При поступлении сигнала CLR блок сбрасывает хранящееся значение. Условно графическое обозначение блока генерации команд представлено на рисунке 2.6.

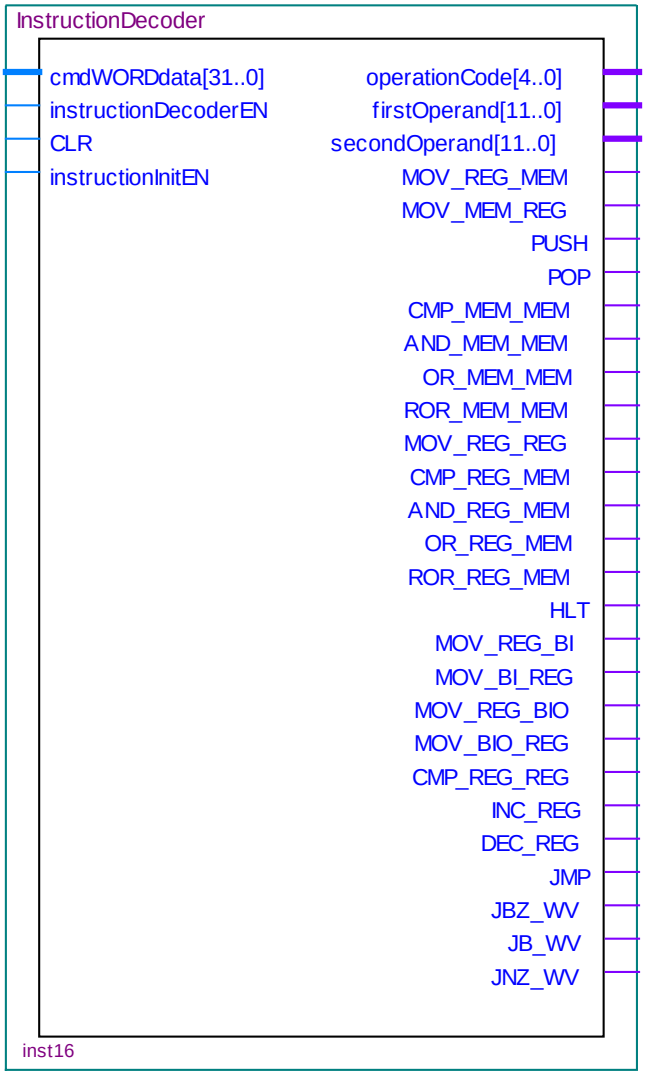

Рисунок 2.6 – Условно-графическое обозначение блока декодера команд

Блок декодера команд имеет следующие входы и выходы:

1. cmdWORDdata[31..0] – команда, полученная из ROM;
2. CLK – тактовый вход;
3. instructonDecoderEN – сигнал записи в декодер;
4. CLR – сигнал сброса внутреннего регистра;
5. instructionInitEN – сигнал для вывода расшифрованной команды;
6. operationCode[4..0] – код выполняемой операции;
7. firstOperand[11..0] – первый операнд операции;
8. secondOperand[11..0] – второй операнд операции;
9. MOV_REG_MEM, MOV_MEM_REG, PUSH, POP, JMP, JBZ, CMP_MEM_MEM, AND_MEM_MEM, OR_MEM_MEM, ROR_MEM_MEM, MOV_REG_REG, CMP_REG_MEM, AND_REG_MEM, OR_REG_MEM,

ROR_REG_MEM, HLT, MOV_REG_BI, MOV_BI_REG, MOV_REG_BIO, MOV_BIO_REG, CMP_REG_REG, INC_REG, DEC_REG, JB_WV, JNZ_WV – вывод выполняемой команды.

2.2.3 Блок обработки операндов

Блок обработки операндов отвечает за своевременную постановку адреса на шину адреса для записи или чтения в зависимости от текущей команды и этапа ее выполнения. Условно-графическое обозначение блок обработки операндов представлен на рисунке 2.7.

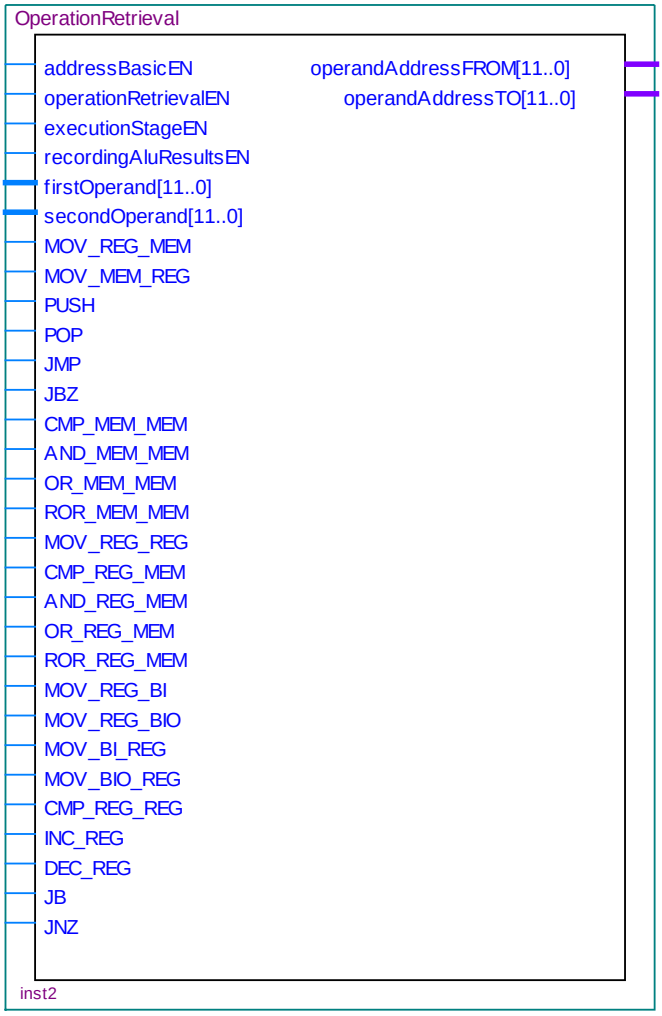

Рисунок 2.7 – Условно-графическое обозначение блока обработки операндов

Блок обработки операндов имеет следующие входы и выходы:
1. operationRetrievalEN – сигнал для чтения данных из ОП2;
2. executionStageEN – сигнал для чтения или записи данных из ОП1;
3. recordingAluResultsEN – сигнал записи данных из АЛУ;
4. addressBasicEN – сигнал получения фактического адреса операнда;
5. firstOperand[11..0] – первый операнд операции;
6. secondOperand[11..0] – второй операнд операции;

7. MOV_REG_MEM, MOV_MEM_REG, PUSH, POP, JMP, JBZ, CMP_MEM_MEM, AND_MEM_MEM, OR_MEM_MEM, ROR_MEM_MEM, MOV_REG_REG, CMP_REG_MEM, AND_REG_MEM, OR_REG_MEM, ROR_REG_MEM, HLT, MOV_REG_BI, MOV_BI_REG, MOV_REG_BIO, MOV_BIO_REG, CMP_REG_REG, INC_REG, DEC_REG, JB_WV, JNZ_WV – ввод выполняемой команды.

8. operandAddressFROM[11..0] – адрес, откуда производится чтение;

9. operandAddressTO[11..0] – адрес, куда производится чтение/запись.

2.2.4 Указатель инструкции

Блок указателя инструкции, или регистра команд, предназначен для корректной обработки команд в необходимом порядке. Данный блок задает адрес считывания данных из ПЗУ. После успешного считывания адрес инкрементируется, также может дополнительно изменяться при помощи команд условных переходов и JMP. Перед поступлением в данный блок команды условного перехода принимают значения в зависимости от текущих значений регистра флагов.

Условно – графическое обозначение блока регистра команд представлено на рисунке 2.8.

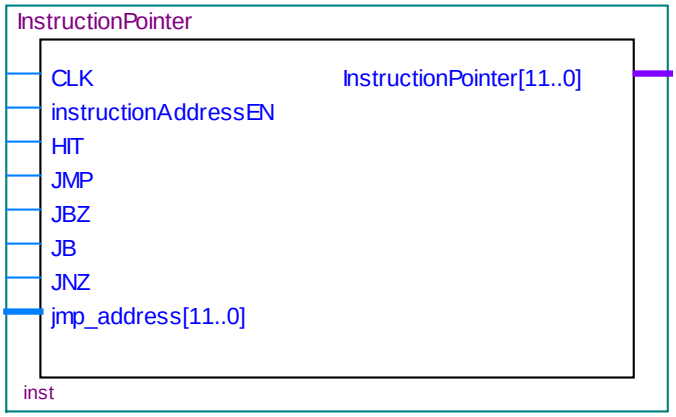

Рисунок 2.8 – Условно-графическое обозначение блока регистра команд

Блок регистра команд имеет следующие входы и выходы:

1. CLK – вход синхросигнала;

2. instructionAddressEN – сигнал о том, что данные из ячейки ROM успешно получены;

3. HIT – сигнал доступности данных в кэш-памяти;

4. JMP – команда для изменения адреса команд;

5. JBZ – команда условного перехода «если меньше нуля»;

5. JB – команда условного перехода «если меньше»;

5. JNZ – команда условного перехода «если неравно нулю»;

5. operandAddressTO[11..0] – адрес, куда производится переход;

6. InstructionPointer[11..0] – вывод адреса следующей команды.

2.3 Арифметико-логическое устройство

Арифметико-логическое устройство (АЛУ) предназначено для выполнения арифметических, логических и сдвиговых операций над операндами. Данный блок подключен непосредственно к УУ – для получения сигналов управления и передачи регистра флагов, и к шине данных, чтобы передавать обработанные данные для записи в память. Условно-графическое обозначение АЛУ представлено на рисунке 2.9.

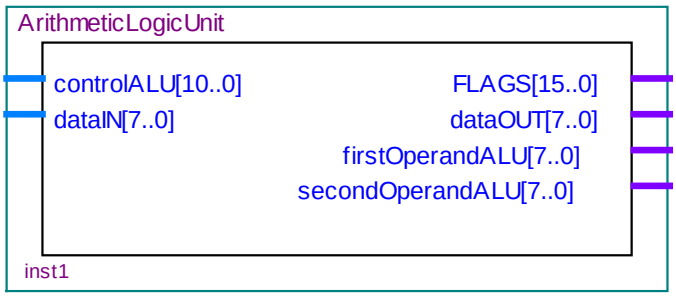

Рисунок 2.9 – Условно-графическое обозначение АЛУ

Блок АЛУ команд имеет следующие входы и выходы:

1. controlALU[10..0]–команда управления, полученная из УУ;
2. dataIN[7..0] – вход данных из шины данных;
3. FLAGS[15..0] – регистр флагов для УУ;
4. dataOUT[7..0] – выход обработанных данных на шину данных;
5. firstOperandALU[7..0] – первый операнд АЛУ;
6. secondOperandALU[7..0] – второй операнд АЛУ;

Для реализации АЛУ представлены следующие команды: CMP – сравнение двух операндов, AND – выполнение логического И над двумя операндами и запись результата в первый, OR – выполнение логического ИЛИ над двумя операндами и запись результата в первый, ROR – циклический сдвиг вправо. Каждая из команд подразумевает загрузку данных и памяти, их обработку, установку флагов состояния регистра флагов и запись результата в память. Функциональная схема АЛУ представлена в приложении Е.

2.4 Организация кэш-памяти процессора

Кэш-память представляет собой организованную в виде ассоциативного запоминающего устройства быстродействующую буферную память ограниченного объема, которая располагается между регистрами процессора и относительно медленной основной памятью и хранит наиболее часто используемую информацию совместно с ее признаками (тегами), в качестве которых выступает часть адресного кода.

В процессе работы отдельные блоки информации копируются из основной памяти в кэш-память. При обращении процессора за командой или данными сначала проверяется их наличие в КП. Если необходимая информация находится в кэше, она быстро извлекается. Это *кэш-попадание*.

Если необходимая информация в КП отсутствует (*кэш-промах*), то она выбирается из основной памяти, передается в микропроцессор и одновременно заносится в кэш-память. Повышение быстродействия вычислительной системы достигается в том случае, когда кэш-попадания реализуются намного чаще, чем кэш-промахи. Условно-графическое обозначение кэш-памяти представлено на рисунке 2.10.

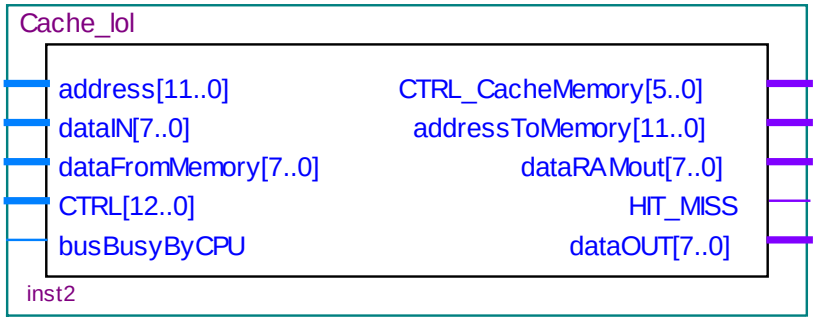

Рисунок 2.10 – Условно-графическое обозначение кэш-памяти

Блок кэш-памяти имеет следующие пины:

1. address[11..0]– адрес данных для чтения или записи;
2. dataIN[7..0] – вход данных из шины данных;
3. dataFromMemory[7..0] – данные, полученные из ПЗУ или ОЗУ;
4. CTRL[12..0] – вход для управления кэш-памятью;
5. busBusyByCPU– сигнал, что УУ использует общую магистраль;
6. CTRL_CacheMemory[5..0] – сигналы управления для памяти;
7. addressToMemory[11..0] – адрес ячейки памяти, необходимый кэшу;
8. dataRAMout[7..0] – данные для записи в RAM;
9. HIT_MISS – сигнал о кэш-попадании или промахе;
10. dataOUT[7..0] – вывод данных на шину данных.

Процессор, выполняя команду, запрашивает операнд по некоторому адресу в адресном пространстве. Кэш-контроллер проверяет, есть ли в кэш-памяти строка данных, соответствующая запрашиваемому адресу. В случае наличия такой строки, данные выводятся на шину данных и дальше используются УУ по назначению. В противном случае, кэш-памяти необходимо обратится к ОЗУ или ПЗУ и получить строку данных для последующей записи ее в банк кэша и предоставлению микропроцессору. После непосредственной обработки данных, в зависимости от типа алгоритма синхронизации с памятью, данные записывают обратно в ОЗУ. Скорость доступа к данным в кэш-памяти при попадании равно одному такту, что примерно равно доступу в РОН. Функциональная схема АЛУ представлена в приложении Ж.

2.5 Прямой доступ к памяти

В режиме прямого доступа к памяти используется специализированное устройство - контроллер прямого доступа к памяти, который перед началом

обмена программируется с помощью центрального процессора: в него передаются адреса основной памяти и количество передаваемых данных. Затем центральный процессор от контроллера прямого доступа к памяти отключается, разрешив ему работать, и до окончания обмена может выполнять другую работу. Об окончании обмена контроллер прямого доступа к памяти сообщает процессору. В этом случае участие центрального процессора косвенное. Обмен ведет контроллер прямого доступа к памяти. Условно-графическое обозначение контроллера ПДП представлено на рисунке 2.11.

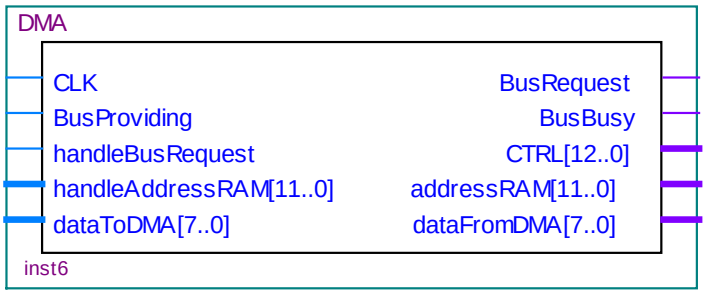

Рисунок 2.11 – Условно-графическое обозначение контроллера ПДП

Блок кэш-памяти имеет следующие пины:

1. CLK – вход синхросигнала;
2. BusProviding – вход предоставления шины от МП;
3. handleBusRequest – запрос от внешних устройств на ПДП;
4. handleAddressRAM[11..0] – адрес для получения данных;
5. dataToDMA[7..0] – данные с шины данных из ОЗУ;
6. BusRequest – сигнал запроса шины у арбитра;
7. BusBusy – сигнал использования шины контроллером;
8. CTRL[12..0] – слово управления на шину управления;
9. addressRAM[11..0] – адрес для получения данных из ОЗУ;
10. dataFromDMA[7..0] – вывод данных внешнему устройству.

ПДП разгружает процессор от обслуживания операций ввода-вывода, способствует увеличению общей производительности ЭВМ. Функциональная схема контроллера ПДП представлена в приложении И.

При работе в режиме прямого доступа к памяти контроллер ПДП выполняет следующие функции:

– принимает запрос на ПДП от внешнего устройства;

– формирует запрос арибтру на захват шин системной магистрали;

– формирует сигнал, сообщающий внешнему устройству о начале выполнения циклов ПДП;

– выдает на шину адреса системной магистрали адрес ячейки ОП, предназначенной для обмена;

– вырабатывает сигналы для управление обменом данными;

– по окончании ПДП контроллер либо организует повторение цикла ПДП, либо прекращает режим ПДП, снимая запросы на него.

2.6 Описание арбитра шины

В реальных системах на роль ведущего вправе одновременно претендовать сразу несколько из подключенных к шине устройств, однако управлять шиной в каждый момент времени может только одно из них. Чтобы исключить конфликты, шина должна предусматривать определенные механизмы арбитража запросов и правила предоставления шины одному из запросивших устройств. Решение обычно принимается на основе приоритетов претендентов.

В данном курсовом проекте арбитр шины является последовательным, следовательно, используются статические приоритеты. Основной недостаток статических приоритетов в том, что устройства, имею-щие высокий приоритет, в состоянии полностью блокировать доступ к шине устройств с низким уровнем приоритета. Однако они достаточно дешевы и просты в реализации. Условно-графическое обозначение арбитра шины представлено на рисунке 2.12.

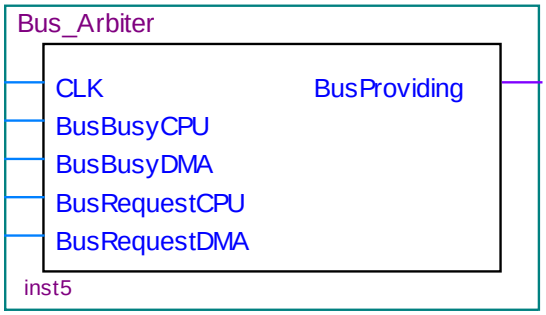

Рисунок 2.12 – Условно-графическое обозначение арбитра шины

Блок кэш-памяти имеет следующие пины:

1. CLK – вход синхросигнала;
2. BusBusyCPU – сигнал о том, что шина занята микропроцессором;
3. BusBusyDMA – сигнал о том, что шина занята контроллером ПДП;
4. BusRequestCPU – запрос от микропроцессора;
5. BusRequestDMA – запрос от контроллера КПДП;
6. BusProviding – сигнал предоставления шины одному из устройств.

Шина арбитром предоставляется в случае, если в данный момент общая магистраль свободна и одно из ведомых устройств подало запрос на захват шины. Функциональная схема арбитра шин представлена в приложении К.

3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ

В данном разделе представлена работа отдельных блоков разрабатываемой микро-ЭВМ. Для демонстрации используется искусственная отправка команд и предоставляется возможность проверить весь предполагаемый для разработки функционал блоков.

В разрабатываемом проекте для всех команд используется слово управления следующей структуры:

– CTRL[12] – сигнал очистки;
– CTRL[11] – сигнал установки начальных значений;
– CTRL[10] – сигнал работы стека;
– CTRL[9..6] – адрес регистра в РОН;
– CTRL[5] – сигнал работы РОН;
– CTRL[4] – сигнал работы ОЗУ;
– CTRL[3] – сигнал работы ПЗУ;
– CTRL[2] – сигнал записи данных;
– CTRL[1] – сигнал чтения данных;
– CTRL[0] – синхросигнал.

3.1 Функциональное моделирование отдельных блоков

Разработка данного курсового проекта началась с проектирования блока регистров общего назначения – самой быстрой памяти процессора. Данный блок получает адрес регистра и команду через шину управления. На рисунке 3.1 изображено функциональное моделирование блока РОН: представлены функции записи и получения данных из регистров.

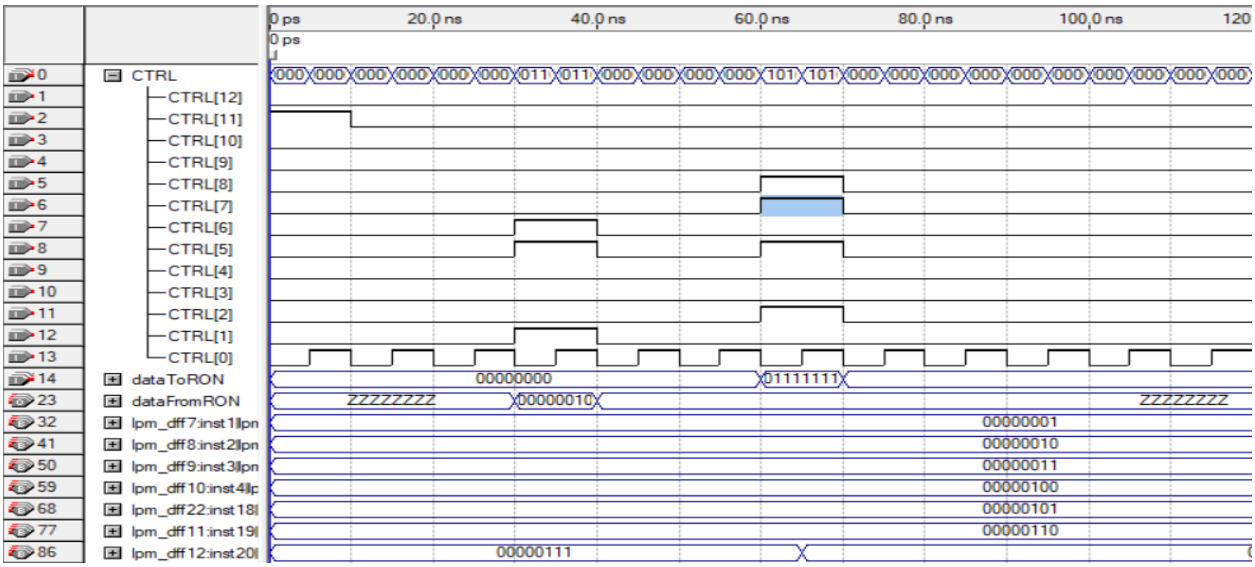

Рисунок 3.1 – Функциональное моделирование блока РОН

Следующим блоком является основная память, состоящая из общего адресного пространства (принстонская архитектура) ПЗУ и ОЗУ. Данный блок принимает от процессора слово управления, а также данные для записи и адрес

команды или выходных данных. На рисунке 3.2 изображено функциональное моделирование блока памяти: представлены функции получения команд из ПЗУ и запись/чтение данных из ОЗУ.

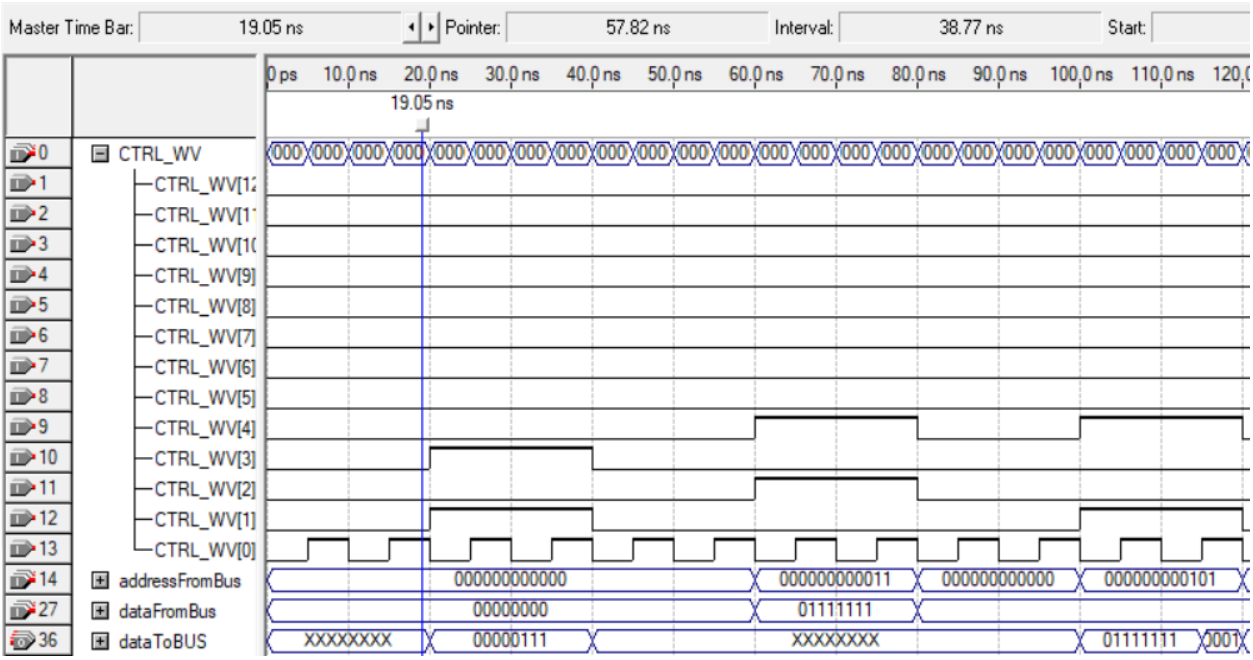

Рисунок 3.2 – Функциональное моделирование блока памяти

Для наглядной демонстрации работы на рисунке 3.3. представлены дампы памяти до и после модуляции, соответственно:

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	7	0	16	2	15	0	32	2
15	0	39	1	128	0	0	0	0
30	0	0	0	0	0	0	0	0
45	0	0	0	55	0	96	7	63
60	87	0	32	4	95	0	48	0
75	14	119	0	96	15	39	7	128
90	0	0	0	0	0	0	0	0
105	0	0	0	0	0	0	0	0
120	7	0	128	33	7	0	64	20
135	32	135	0	8	40	151	0	24

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	15	106	21	0	255	6	5	5
15	2	0	0	0	0	4	0	0
30	0	0	1	0	0	0	0	0
45	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0
105	0	0	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0
135	31	85	248	0	0	0	0	0

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	7	0	16	2	15	0	32	2
15	0	39	1	128	0	0	0	0
30	0	0	0	0	0	0	0	0
45	0	0	0	55	0	96	7	63
60	87	0	32	4	95	0	48	0
75	14	119	0	96	15	39	7	128
90	0	0	0	0	0	0	0	0
105	0	0	0	0	0	0	0	0
120	7	0	128	33	7	0	64	20
135	32	135	0	8	40	151	0	24

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	15	106	21	127	255	6	5	5
15	2	0	0	0	0	4	0	0
30	0	0	1	0	0	0	0	0
45	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0
105	0	0	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0
135	31	85	248	0	0	0	0	0

Рисунок 3.3 – Дампы памяти ПЗУ и ОЗУ до(слева) и после (справа) модуляции

Стек является частью основной памяти, но имеет другую организацию. В данном курсовом проекте стек «растет» вниз, т.е. адреса изменяются от самого большого до нижней границы памяти, выделенной под стек. В данном блоке также присутствуют модули для обработки ошибок, например, переполнение стеки или попытка извлечь значение из пустого стека. На рисунке 3.4 изображено функциональное моделирование блока стека: представлены функции получения и записи данных, а также работа в случае ошибочной команды.

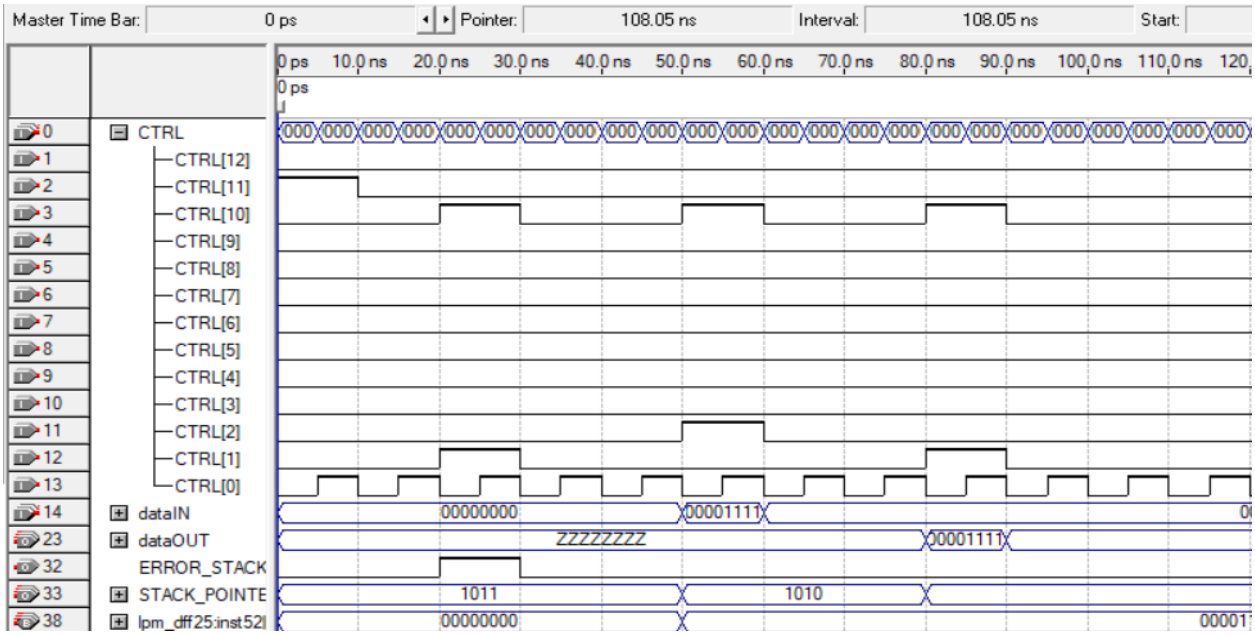

Рисунок 3.4 – Функциональное моделирование блока стека

Для моделирования УУ приведен пример получения команды MOV_REG_MEM. Результат представлен на рисунке 3.5.

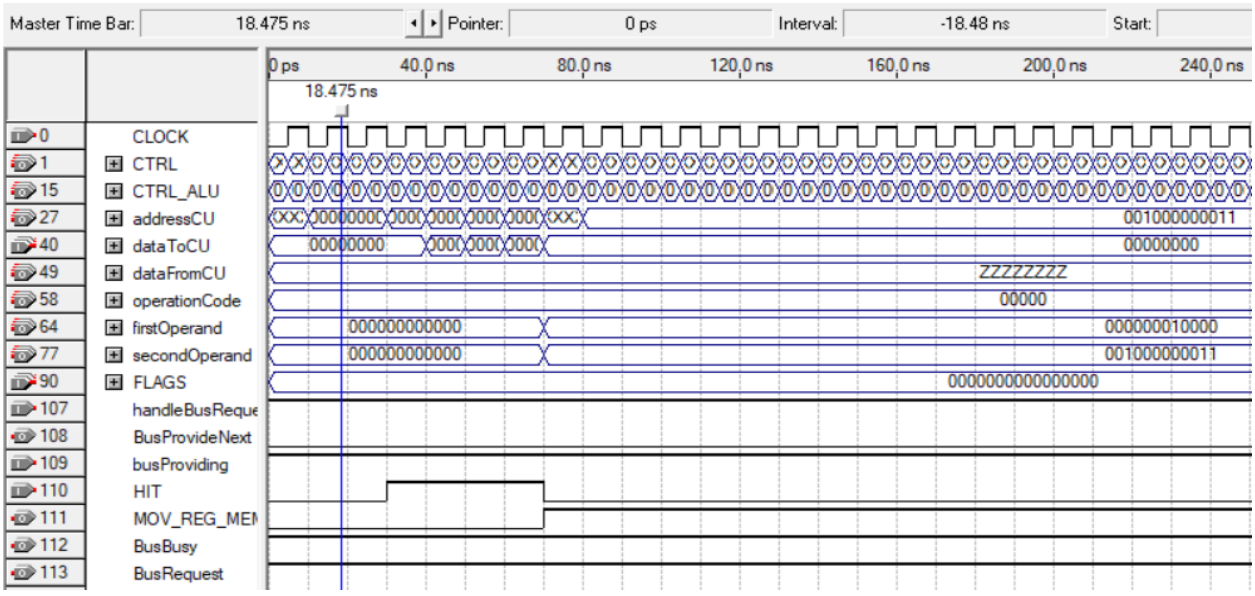

Рисунок 3.5 – Функциональное моделирование УУ

На рисунке 3.6 представлено функциональное моделирование блока АЛЕ на примере команды AND, которая включает в себя запись двух операндов во внутренние регистры и дальнейшую выдачу результата на шину данных. Для управления АЛУ устройство управления использует отдельное слово управления, которое имеет следующие биты:

– CTRL_ALU[10] – сигнал очистки;
– CTRL_ALU[9] – сигнал вывода данных;
– CTRL_ALU[8] – сигнал команды DEC;
– CTRL_ALU[7] – сигнал команды INC;
– CTRL_ALU[6] – сигнал команды ROR;
– CTRL_ALU[5] – сигнал команды OR;
– CTRL_ALU[4] – сигнал команды AND;
– CTRL_ALU[3] – сигнал команды CMP;
– CTRL_ALU[2] – сигнал записи второго операнда;
– CTRL_ALU[1] – сигнал записи первого операнда;
– CTRL_ALU[0] – синхросигнал.

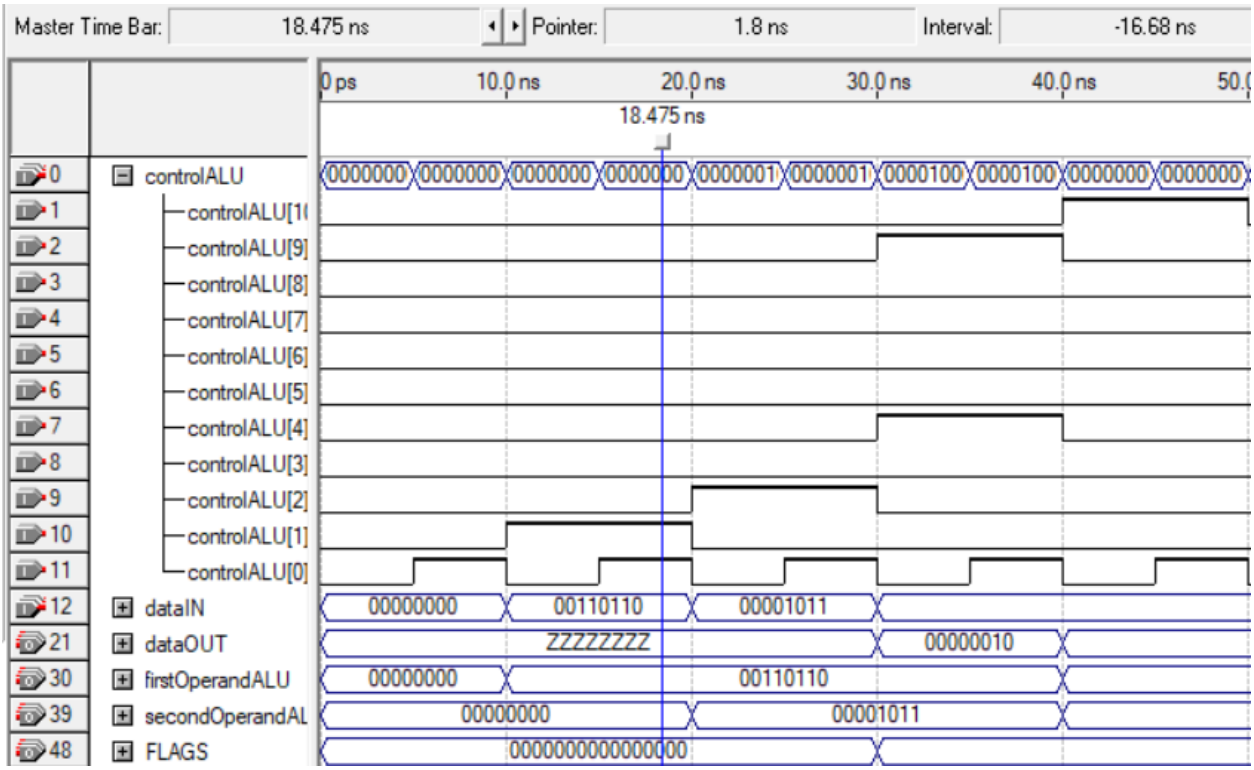

Рисунок 3.6 – Функциональное моделирование АЛУ

Следующим этапом является моделирование кэш-памяти микпроцессора. На рисунке 3.7 представлена основная функция кэш-памяти – чтение и запись данных из памяти. Основное отличие от обычного получения данных является пересылка сразу нескольких слов из ОЗУ или ПЗУ, что позволяет предугадать дальнейшее обращение микпроцессора и, следовательно, получить кэш-попадание, сократив время выполнения команды.

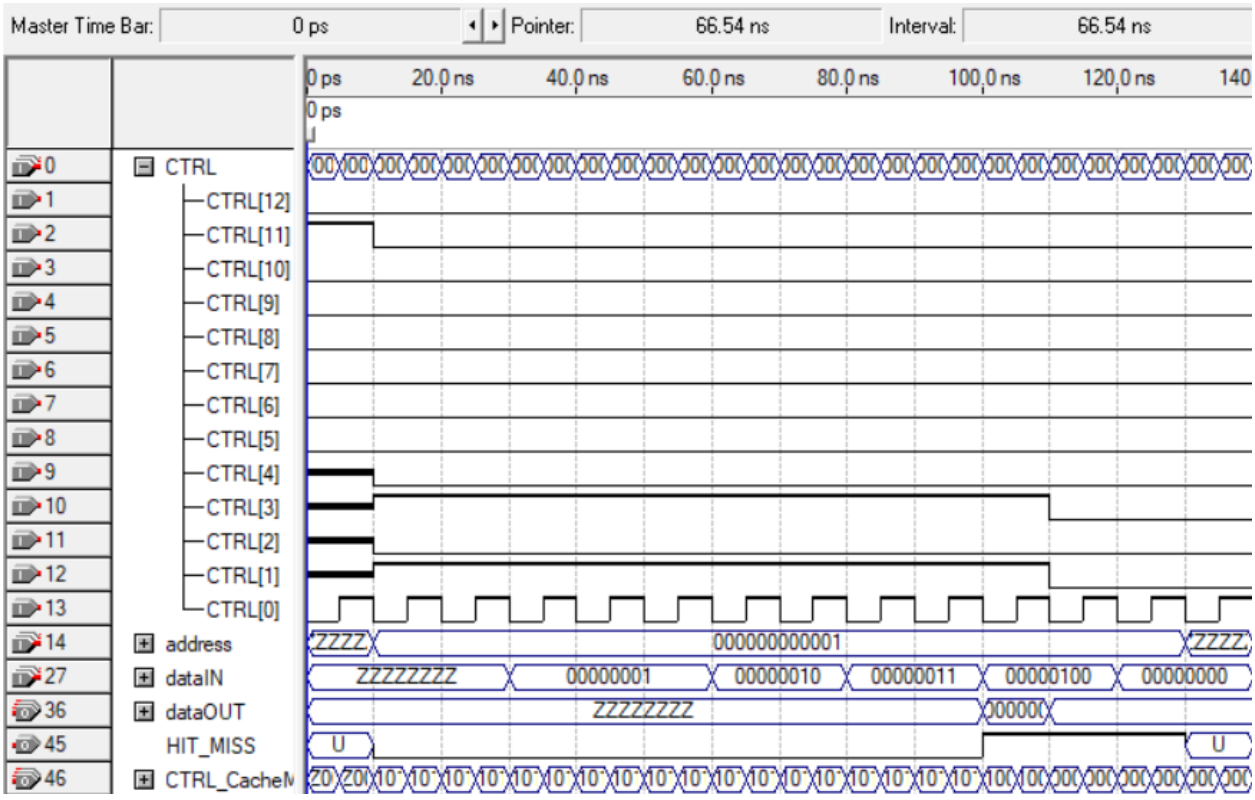

Рисунок 3.7 – Функциональное моделирование кэш-памяти

Контроллер ПДП позволяет обмениваться данными с основной памятью без непосредственного участия микропроцессора. На рисунке 3.8 представлена реализация получения данных из ОЗУ.

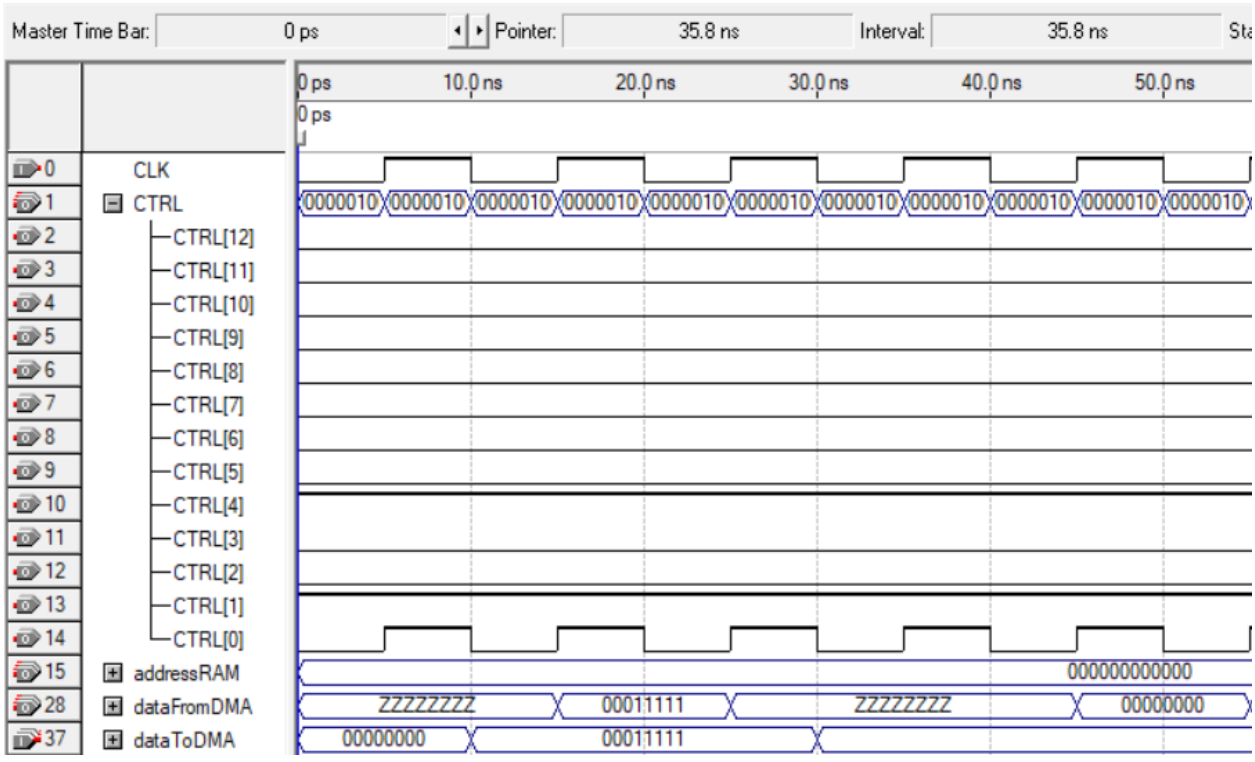

Рисунок 3.8 – Функциональное моделирование контроллера ПДП

3.2 Описание работы временной диаграммы всего устройства

Для моделирования разработанной системы используется написанная микропрограмма, включающая выполнение каждой команды, требуемой для разработки. Команды хранятся в ПЗУ и выполняются поочередно и автоматически, для этого блоки, представленные в подразделе 3.1 объединены в единую систему. Данная микропрограмма представлена в таблице 3.1.

Таблица 3.1 – Микропрограмма микро-ЭВМ

Команда	Двоичная кодировка		
	КОП	ОП1	ОП2
MOV reg, $mem	00000	0000 0000 0001	0000 0000 0010
MOV $mem, reg	00001	0000 0000 0001	0000 0000 0010
PUSH reg	00010	0000 0000 0000	–
POP reg	00011	0000 0000 0111	–
JMP	00100	0000 0001 1000	–
ROR $mem, $mem	01001	0000 0000 0100	0000 0000 0101
JBZ	00101	0000 0011 0000	–
CMP $mem, $mem	00110	0000 0000 0110	0000 0000 0111
AND $mem, $mem	00111	0000 0000 1000	0000 0000 1001
OR $mem, $mem	01000	0000 0000 1100	0000 0000 1101
MOV reg, reg	01010	0000 0000 0010	0000 0000 0100
CMP reg, $mem	01011	0000 0000 0011	0000 0000 0000
AND reg, $mem	01100	0000 0000 0100	0000 0001 0001
OR reg, $mem	01101	0000 0000 0101	0000 0000 1110
ROR reg, $mem	01110	0000 0000 0110	0000 0000 1111
HLT	01111	–	–

Представленные выше команды подразделяются на три основные категории: команды пересылки данных, команды, выполняемые блоком АЛУ и команды переходов.

Команды MOV reg, reg, MOV reg, $mem, MOV $mem, reg, PUSH, POP относятся к первой категории. При обработке данных команд не используется АЛУ, для их правильного выполнения необходимо, при помощи УУ, обеспечить своевременную передачу данных по общим шина.

Команды CMP $mem, $mem, CMP reg, $mem являются арифметическими и относятся ко второй категории – это означает, что для их выполнения необходимо работа АЛУ. Выполнение данных команд основано на получении данных по адресам, переданным в операндах, и дальнейшей передачи данных в АЛУ, где над ними выполняется операция декрементирования с последующим выставлением регистра флагов и записи обратно в память или регистр.

Команды AND $mem, $mem, AND reg, $mem, OR reg, $mem, OR $mem, $mem являются логически и относятся ко второй категории. Первый этап

выполнения данных команд аналогичен арифметическим командам – над данными выполняется операция логического И или ИЛИ.

Команды ROR $mem, $mem, ROR reg, $mem являются сдвиговыми и относятся ко второй категории. Данные команды, как следует из названия, выполняют циклический сдвиг вправо и последующую запись результата по адресу ОП1.

Команды JMP, JBZ являются командами безусловного и условного перехода, соответственно, и относятся к командам третьей категории. Данные команды позволяют изменять значение регистра команд, т.е. реализовать, например, конструкции с условием, от выполнения которого зависит дальнейшая работа программы.

Последней командой для реализации является HLT – прекращает выполнение программы и переводит процессор в состояние останова.

В соответствии с заданием была доработана АСК и реализована микропрограмма сортировки массива различной длины. Данная микропрограмма представлена в таблице 3.2. Исходные данные к программе:

– длина массива – 4;

– адрес начала массива в памяти – 0000 0010 1000.

Таблица 3.2 – Микропрограмма сортировки массива

Команда	Двоичная кодировка		
	КОП	ОП1	ОП2
MOV reg, $mem	00000	0000 0000 1000	0000 0010 0001
MOV reg, $mem	00000	0000 0000 0100	0000 0010 0100
DEC reg	11000	0000 0000 0100	–
MOV reg, $mem	00000	0000 0000 0110	0000 0010 0000
MOV reg, si + $mem	10000	0000 0000 0000	1000 0010 1000
MOV reg, [reg] + si + $mem	10010	0000 0000 0001	1000 0010 1000
CMP reg, reg	10110	0000 0000 0000	0000 0000 0001
JB	10100	0000 1010 0011	–
MOV si + $mem, reg	10001	1000 0010 1000	0000 0000 0001
MOV [reg] + si + $mem	10011	1000 0010 1000	0000 0000 0000
MOV reg, $mem	00000	0000 0000 0110	0000 0010 0001
INC reg	10111	0000 0000 1000	–
DEC reg	11000	0000 0000 0100	–
CMP reg, $mem	01011	0000 0000 0100	0000 0010 0001
JZ	10101	0000 1000 0111	–
AND reg, $mem	01100	0000 0000 0110	0000 0010 0000
JZ	10101	0000 0111 1000	–
HLT	01111	–	–

Работа спроектированной микро-ЭВМ представлена на рисунках 3.9.1 – 3.9.13.

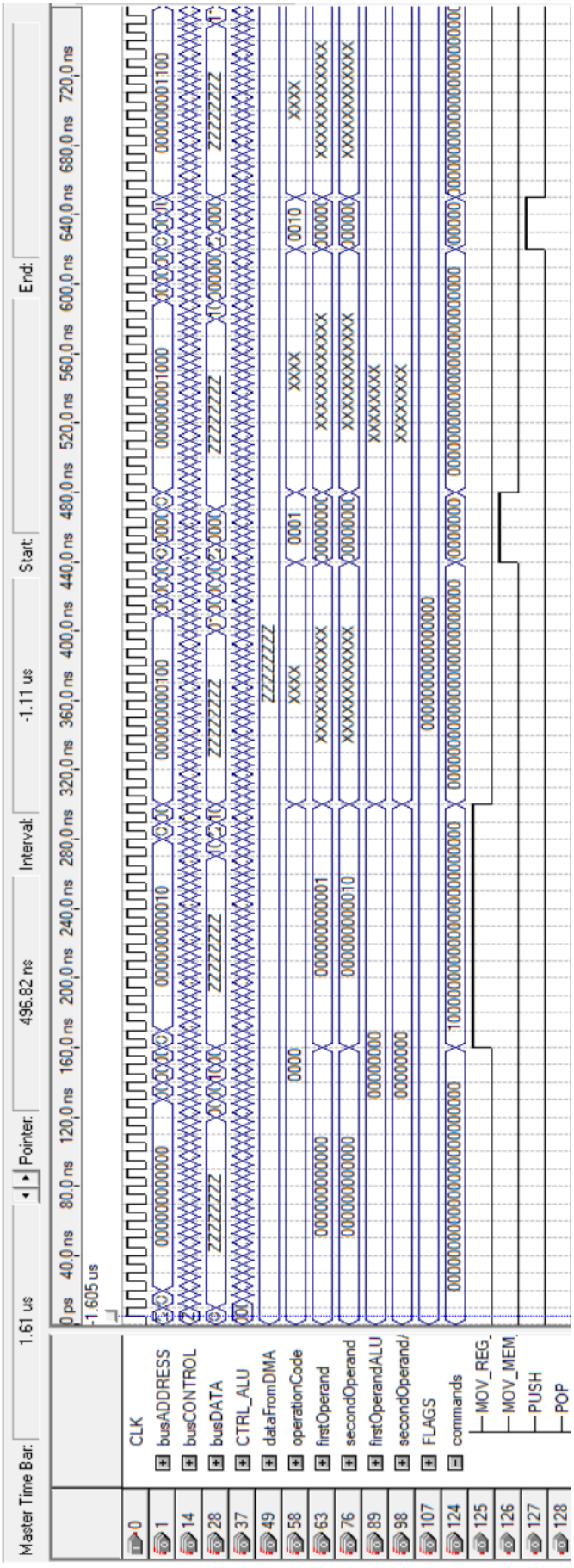

Рисунок 3.9.1 – Функциональное моделирование всех блоков системы

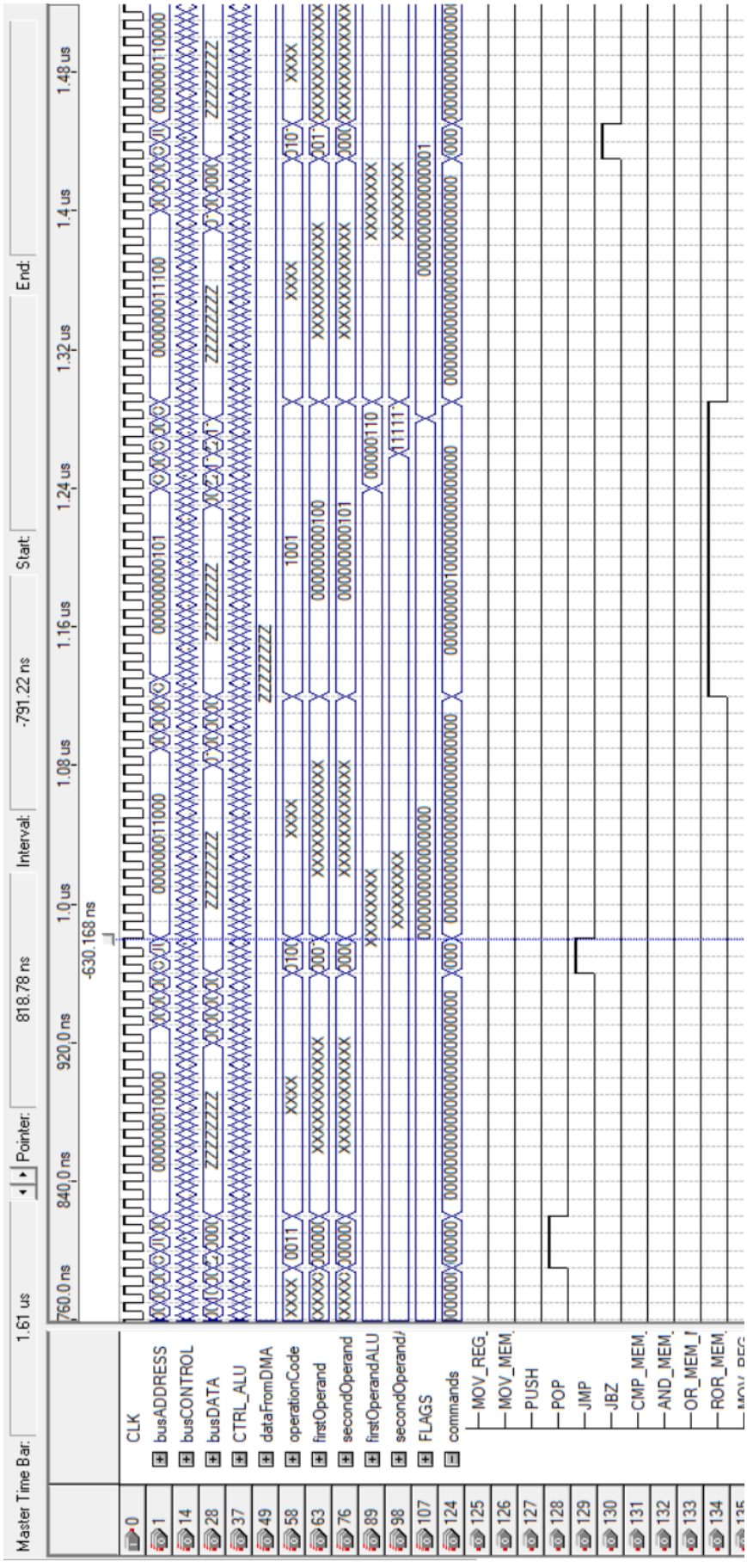

Рисунок 3.9.2 – Функциональное моделирование всех блоков системы

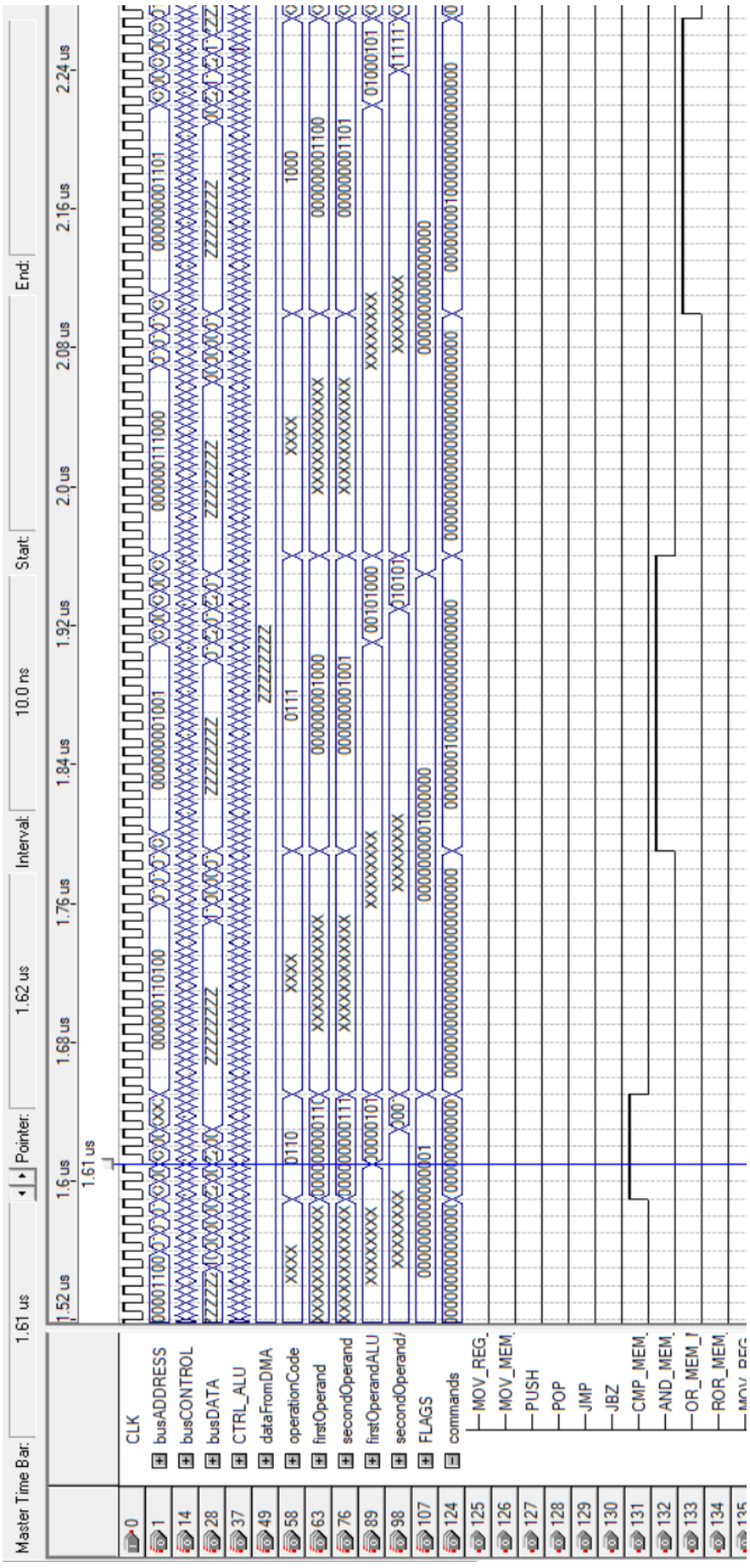

Рисунок 3.9.3 – Функциональное моделирование всех блоков системы

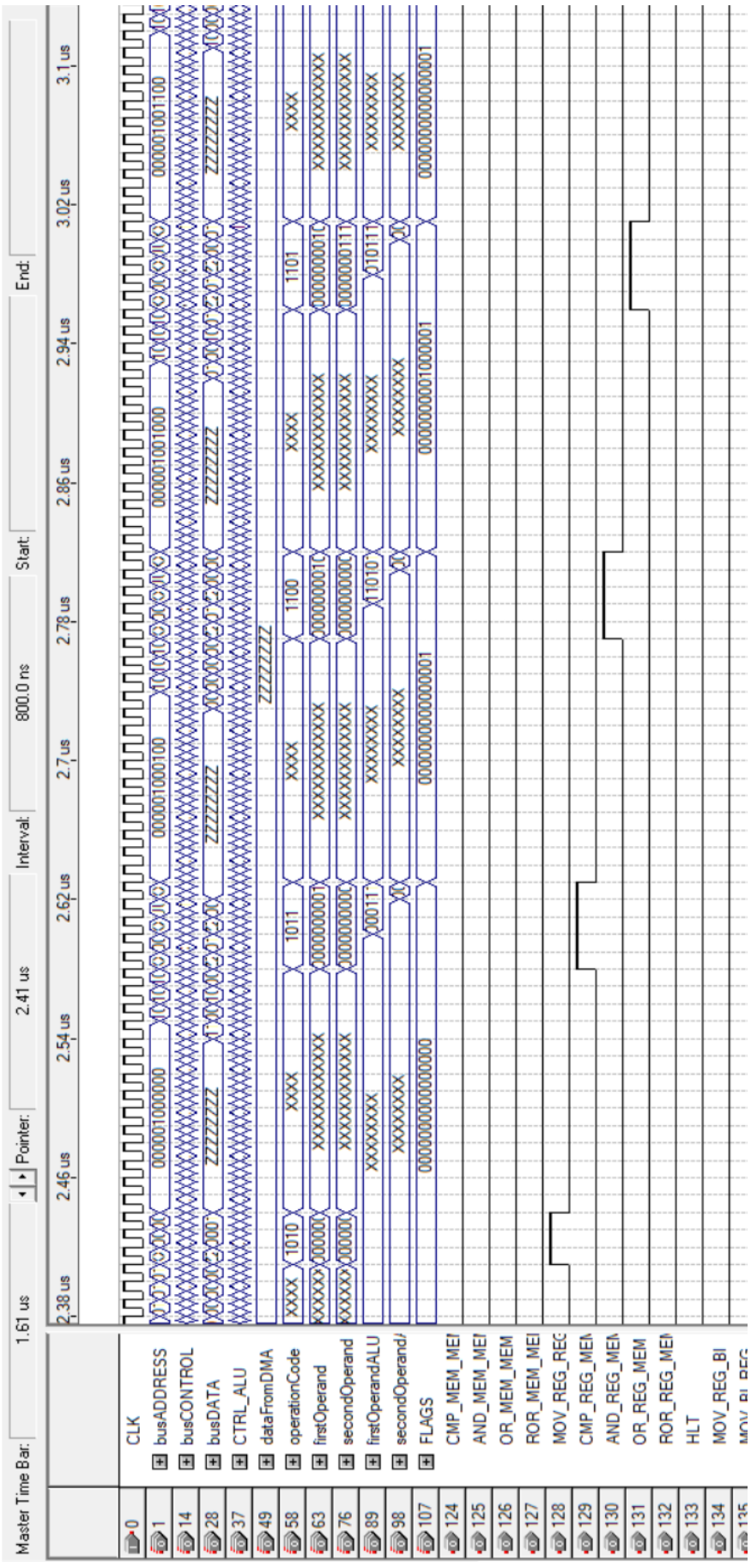

Рисунок 3.9.4 – Функциональное моделирование всех блоков системы

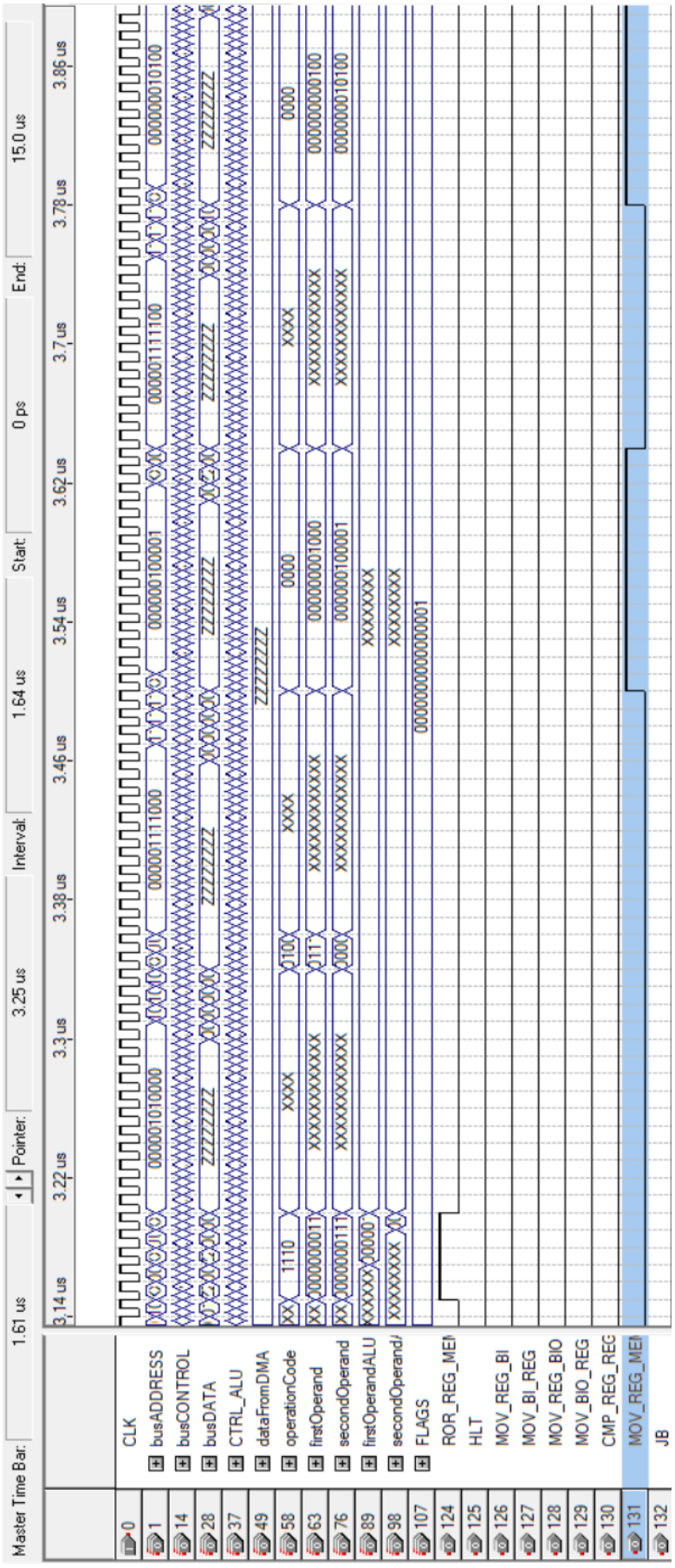

Рисунок 3.9.5 – Функциональное моделирование всех блоков системы

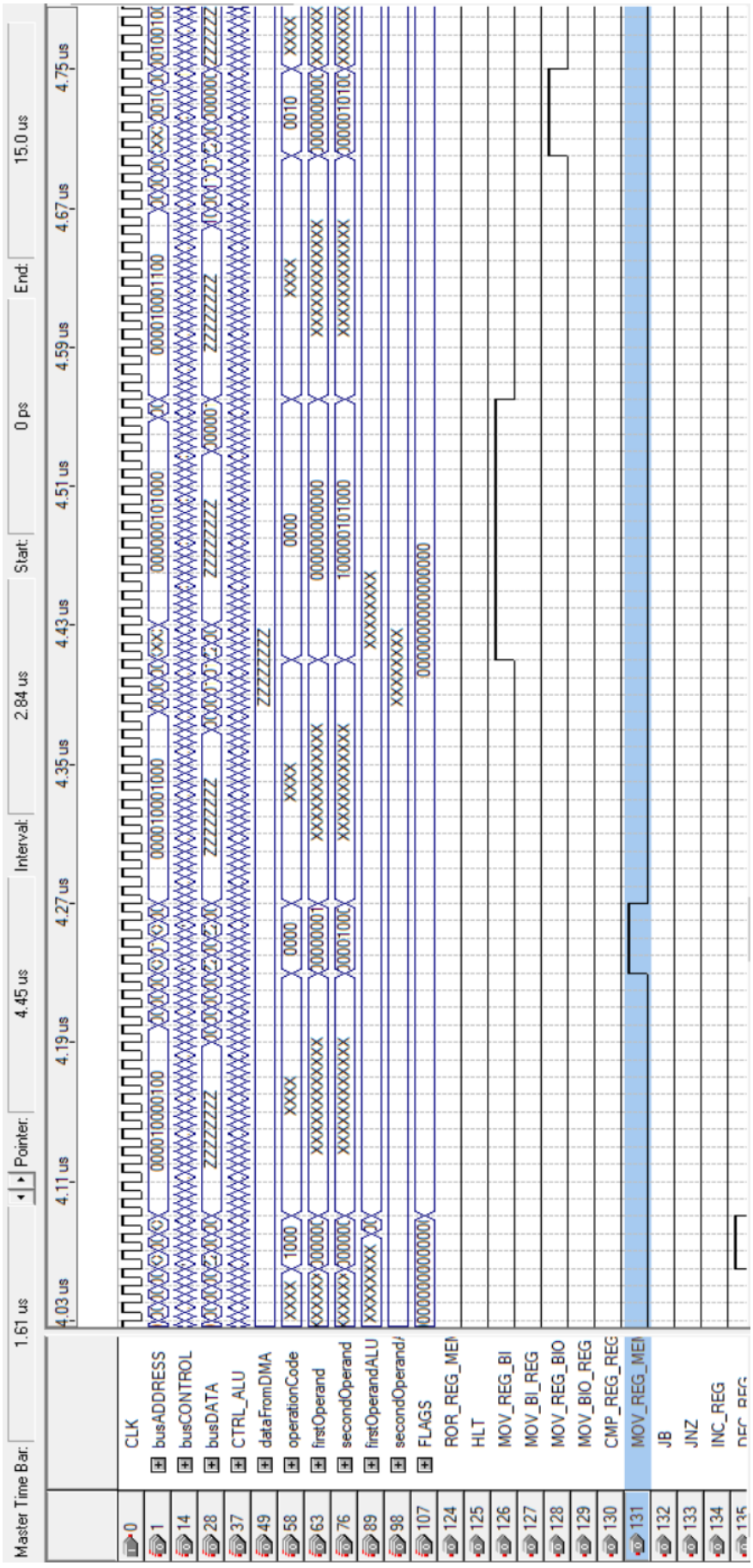

Рисунок 3.9.6 – Функциональное моделирование всех блоков системы

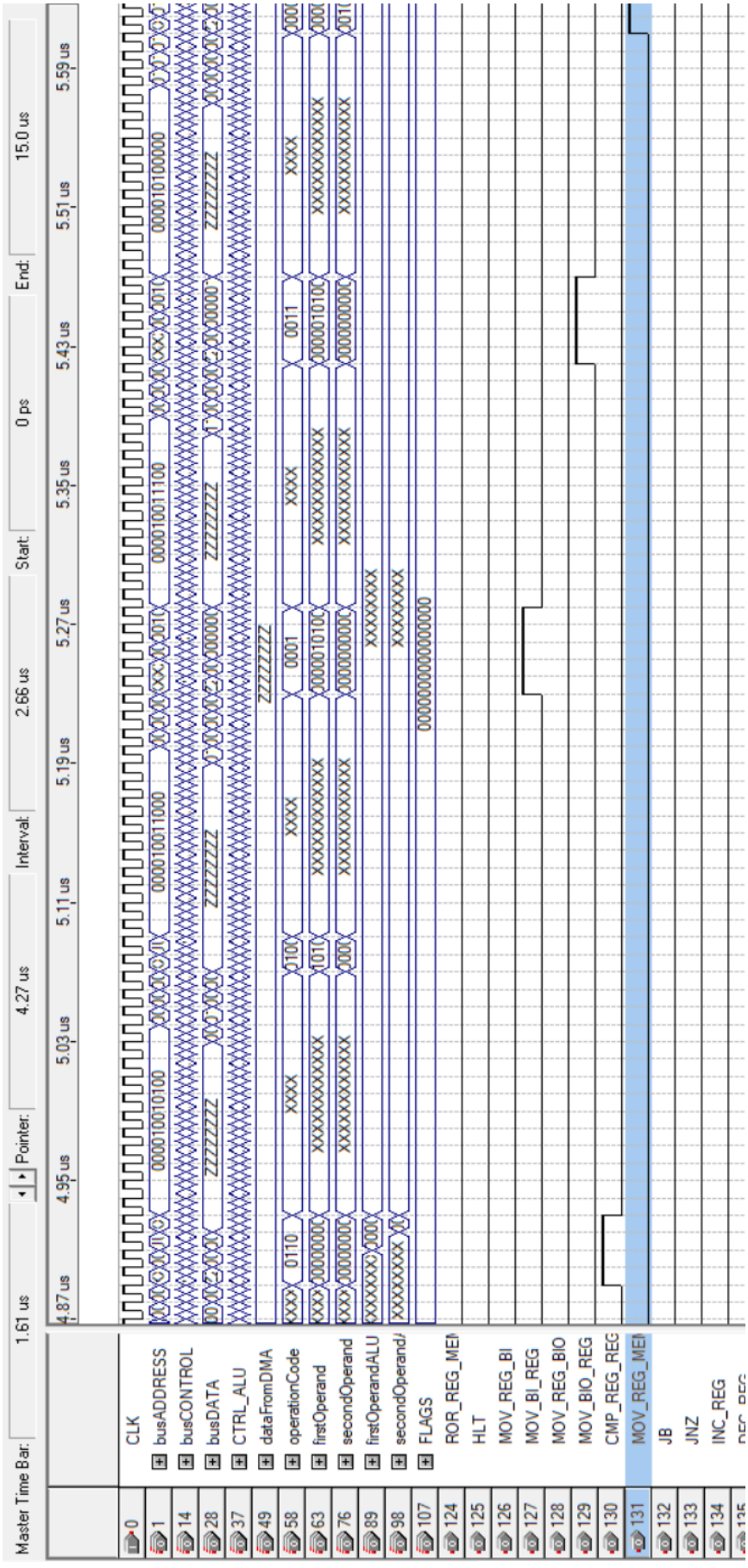

Рисунок 3.9.7 – Функциональное моделирование всех блоков системы

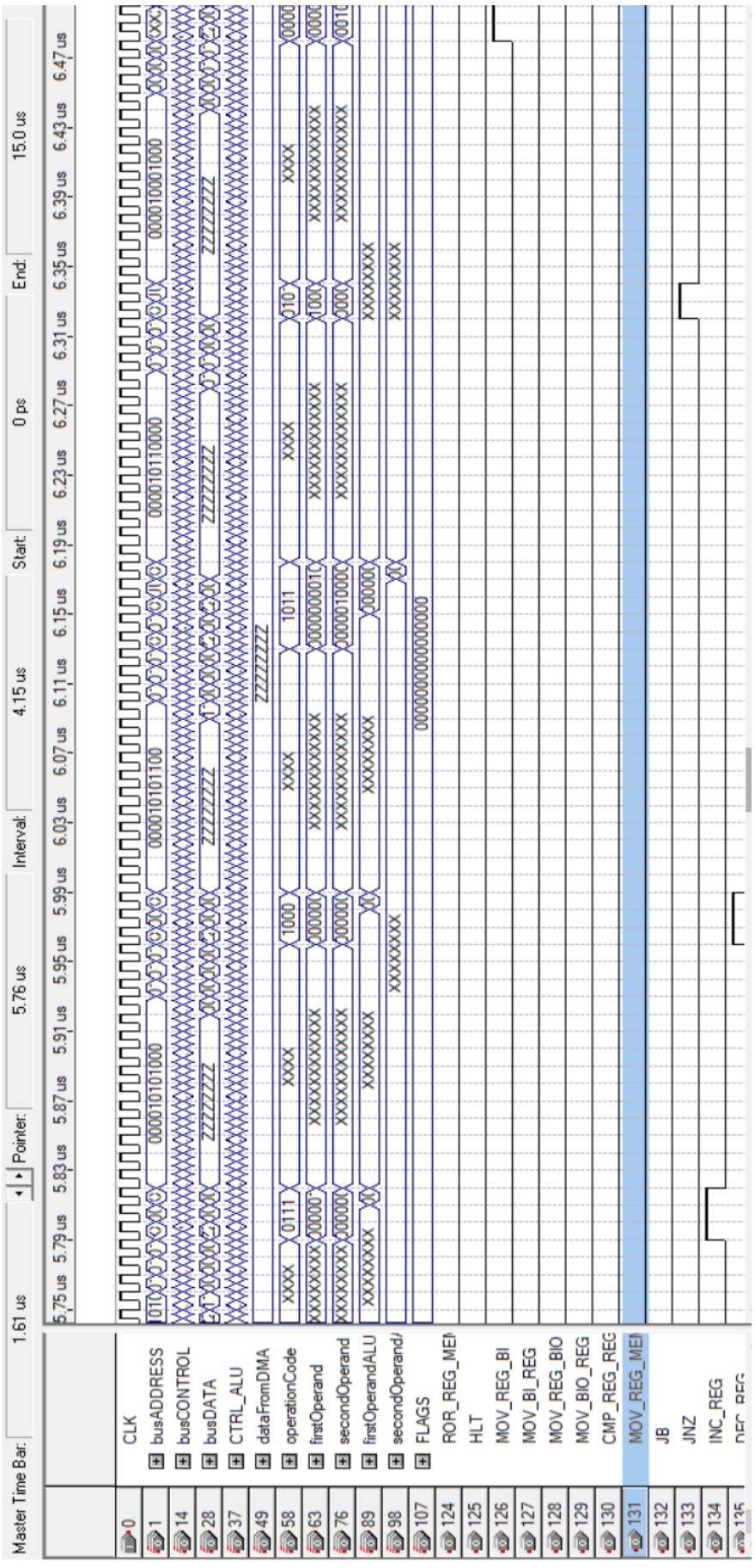

Рисунок 3.9.8 – Функциональное моделирование всех блоков системы

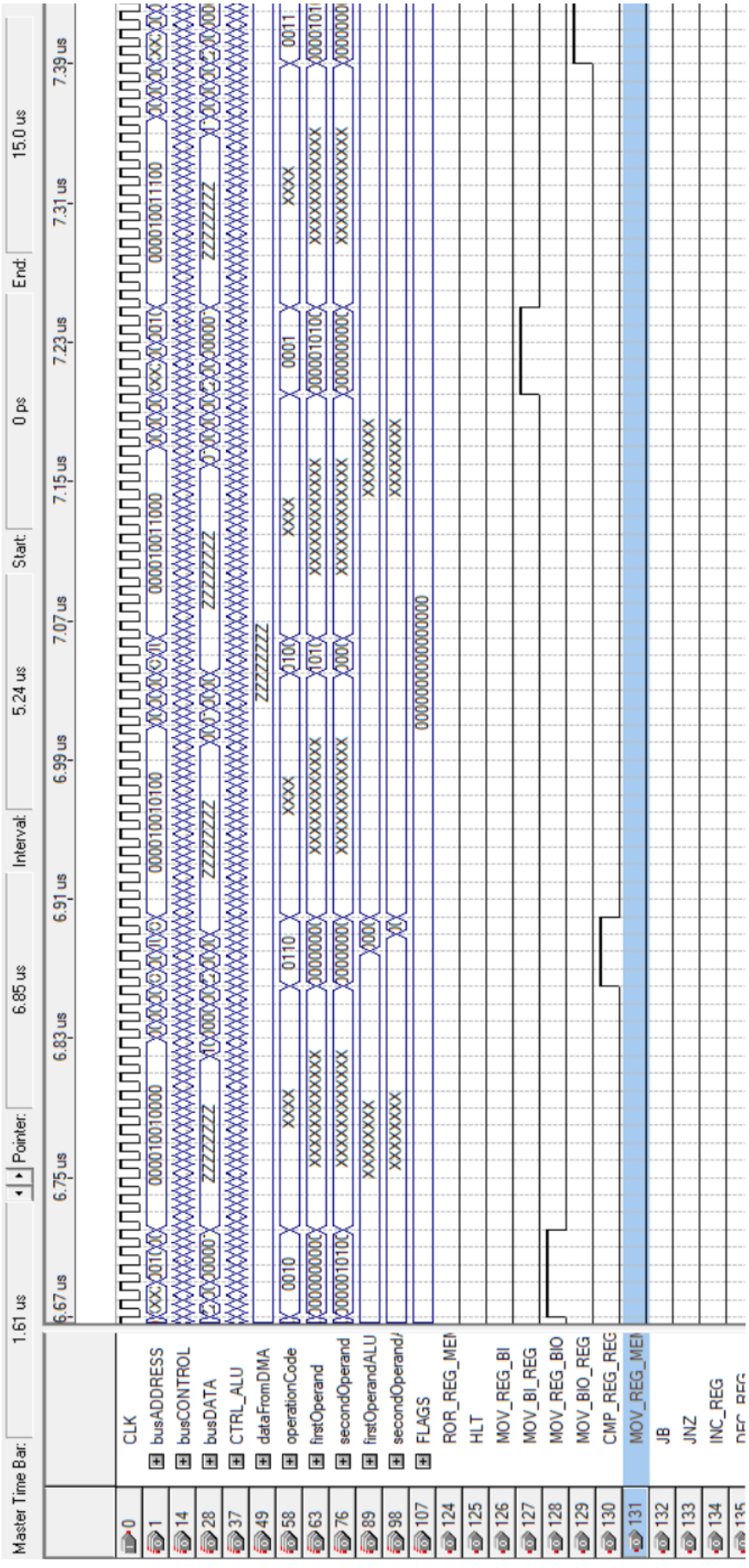

Рисунок 3.9.9 – Функциональное моделирование всех блоков системы

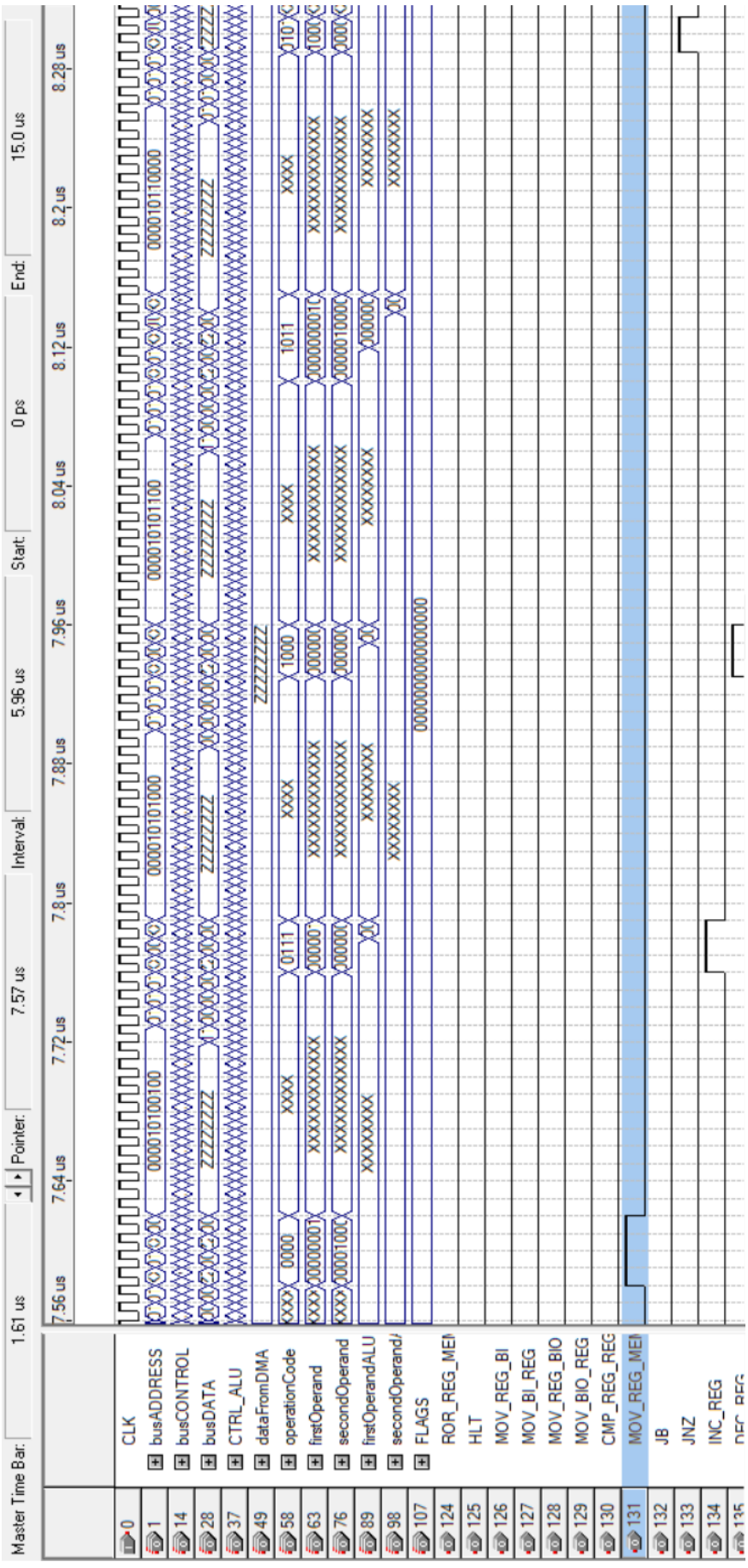

Рисунок 3.9.10 – Функциональное моделирование всех блоков системы

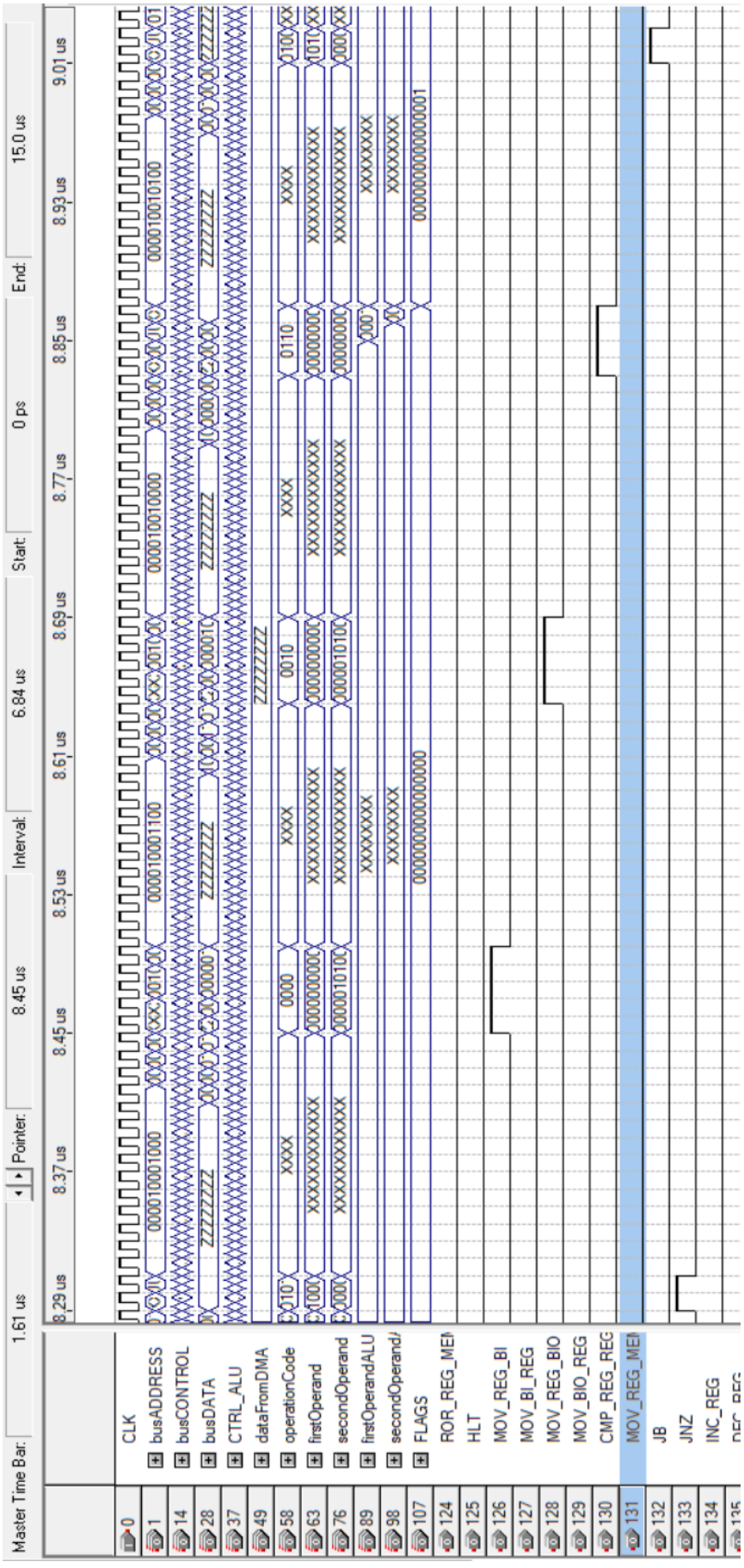

Рисунок 3.9.11 – Функциональное моделирование всех блоков системы

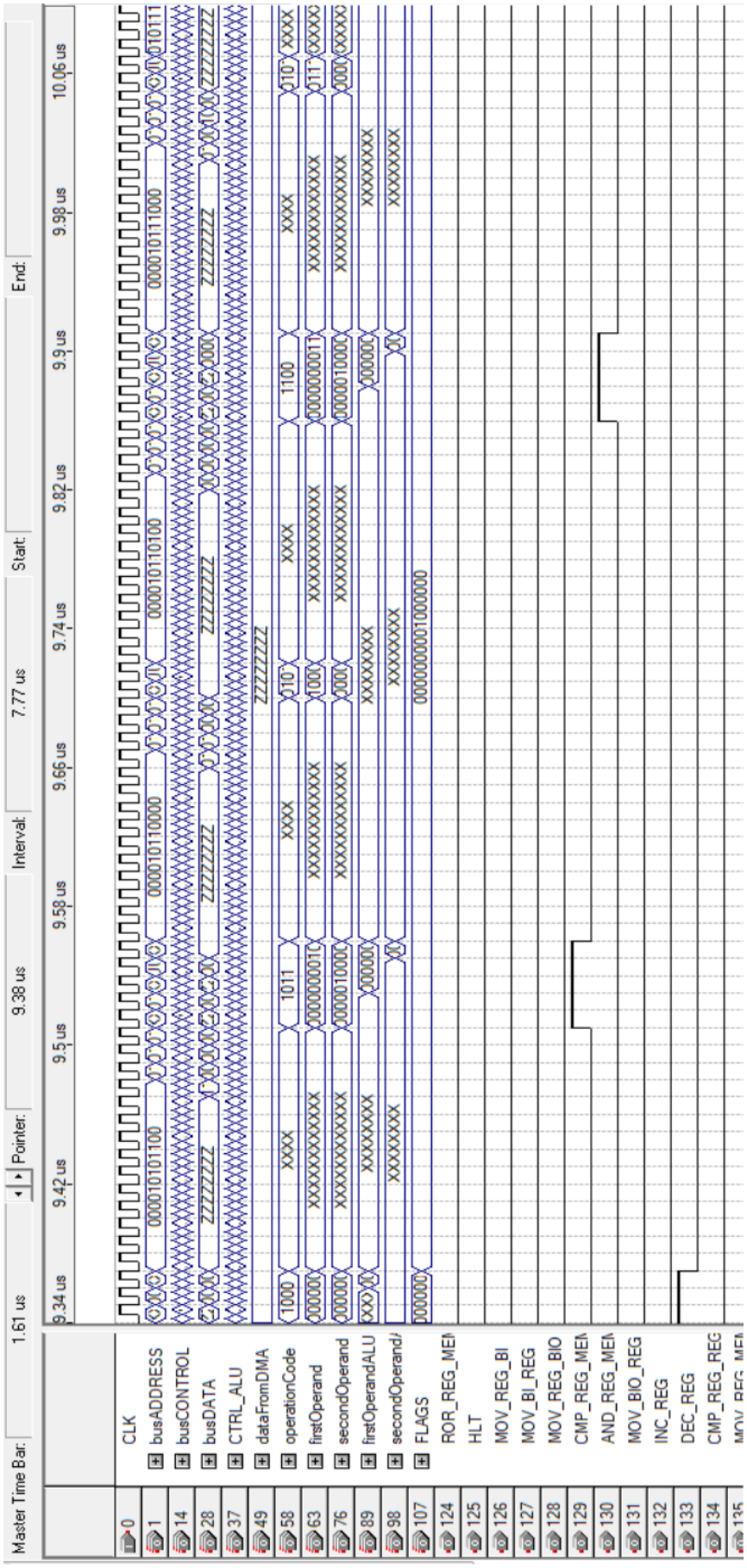

Рисунок 3.9.12 – Функциональное моделирование всех блоков системы

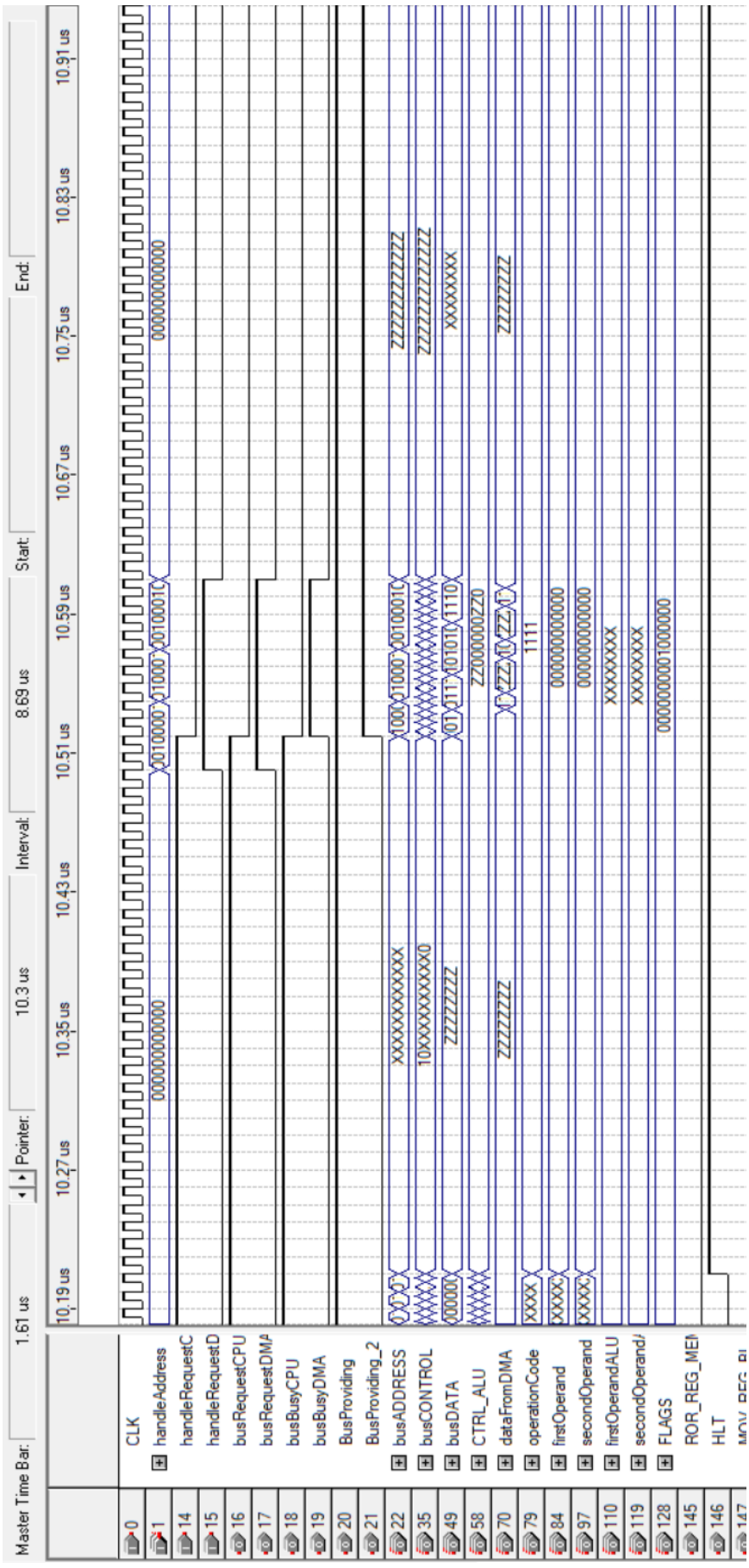

Рисунок 3.9.13 – Функциональное моделирование всех блоков системы

Свидетельством исправной работы разработанной микро-ЭВМ являются дампы памяти ОЗУ, представленные на рисунке 3.10.

Addr	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14
0	15	106	21	0	255	6	5	5	42	40	0	0	126	69	47
15	2	0	0	0	0	4	0	0	0	0	0	0	0	0	0
30	0	0	1	0	0	0	0	0	0	0	3	1	2	4	0
45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
105	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
135	31	85	248	0	0	0	0	0	0	0	0	0	0	0	0
150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
165	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
180	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
195	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Addr	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14
0	15	106	21	0	255	6	5	5	42	40	0	0	126	69	47
15	2	0	0	0	0	4	0	0	0	0	0	0	0	0	0
30	0	0	1	0	0	0	0	0	0	0	3	1	2	4	0
45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
105	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
135	31	85	248	0	0	0	0	0	0	0	0	0	0	0	0
150	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
165	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рисунок 3.9 – Дампы памяти ОЗУ до (сверху) и после (снизу) моделирования

ЗАКЛЮЧЕНИЕ

За время работы над курсовым проектом была спроектирована и разработана микро-ЭВМ в соответствии с листом задания, позволяющая выполнять основные вычислительные функции, например, сравнение двух операндов или пересылка данных между памятью.

Данный проект реализован при помощи среды разработки Quartus Altera II, что позволило провести проверку работоспособности и отладку систему без использования физических компонентов, следовательно, без существенных материальных затрат. Данная среда разработки позволяет гибко работать с подобными проектами, следовательно, в дальнейшем имеется возможность дорабатывать и улучшать разработанный проект.

Разработанная микро-ЭВМ будет востребована в компаниях, которые занимаются разработкой высокопроизводительных систем, например, Intel и AMD. Исходя из предсказаний специалистов, а также стремительному росту всевозможной автоматизации повседневных вещей или быстрому развитию вычислительной техники, для которых необходим надежный управляющий элемент, данный проект будет актуален продолжительное количество времени.

Проект разработан в соответствии с поставленными задачами и исходными данными, весь функционал реализован в полном объеме. Микро-ЭВМ была разработана поэтапно, начиная простейшей пересылкой данных между памятью и РОН и заканчивая добавления новых блоков, таких как арбитр шин и контроллер ПДП. Разработанная архитектура позволит в дальнейшем добавлять новые алгоритмы, компоненты и модули. Согласно функциональным схемам и графикам моделирования работы системы демонстрируется, что команды выполняются правильно.

Гибкость проекту, как уже упоминалось ранее, придает реализация в среде разработки Quartus, что позволит другим инженерам и разработчикам вносить свои изменения в данную вычислительную систему.

В дальнейшем планируется добавление конвейеризации операций. Данное функция позволит существенно увеличить производительность у однопоточной микро-ЭВМ за счет разделения подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры Производительность при этом возрастает, благодаря тому что одновременно на различных ступенях конвейера выполняется несколько команд

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1]. Функциональная и структурная организация ЭВМ [Электронный ресурс] – Электронные данные. – Режим доступа: https://infopedia.su/1xa4a5.html – Дата доступа 20.02.2021.

[2]. Основные этапы решения задач на ЭВМ [Электронный ресурс] – Электронные данные. – Режим доступа: https://studopedia.ru/5_29957_osnovnie-etapi-resheniya-zadach-na-evm.html – Дата доступа 23.02.2021.

[3]. Классификация и иерархическая структура памяти ЭВМ. Основная память ЭВМ. ОЗУ. ПЗУ. СОЗУ [Электронный ресурс] – Электронные данные. – Режим доступа: https://skarlupka.ru/articles.php?id=46 – Дата доступа 01.03.2021.

[4]. Иерархия памяти вычислительных систем [Электронный ресурс] – Электронные данные. – Режим доступа: https://studref.com/660837/informatika/ierarhiya_pamyati_vychislitelnyh_sistem – Дата доступа 05.03.2021.

[5]. Регистры процессора. Как уже упоминалось, внутренние регистры процессора представляют собой сверхоперативную память небольшого размера [Электронный ресурс] – Электронные данные. – Режим доступа: https://studopedia.ru/3_71506_registri-protsessora.html – Дата доступа 10.03.2021.

[6]. Магистрально-модульный принцип архитектуры компьютера [Электронный ресурс] – Электронные данные. – Режим доступа: https://spravochnick.ru/informacionnye_tehnologii/arhitektura_kompyutera_struktura/magistralno-modulnyy_princip_arhitektury_kompyutera/ – Дата доступа 15.03.2021.

[7]. Цикл выполнения команды [Электронный ресурс] – Электронные данные. – Режим доступа: https://studopedia.ru/22_15166_tsikl-vipolneniya-komandi.html – Дата доступа 25.03.2021.

[8]. Основной цикл работы ЭВМ [Электронный ресурс] – Электронные данные. – Режим доступа: https://vuzlit.ru/1035877/osnovnoy_tsikl_raboty – Дата доступа 26.03.2021.

Балашов, Е.П. Микро- и мини-ЭВМ: справочное пособие / Е.П. Балашов, В.Л. Григорьев, Г.А. Петров – М.: Энергоатомиздат, 2019. – 376 с.

Самаль, Д.И. Структурная и функциональная организация ЭВМ: лабораторный практикум / Д.И. Самаль, В.В. Колонов – Минск: БГУИР, 2011. – 43 с.

Самаль, Д.И. Структурная и функциональная организация ЭВМ. Повышение производительности центрального процессора: лабораторный практикум / Д.И. Самаль, В.В. Колонов – Минск: БГУИР, 2010. – 38 с.