**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

**Факультет программной инженерии и компьютерной техники**

Образовательная программа Компьютерные системы и технологии

Направление подготовки (специальность)09.04.01 Информатика и вычислительная техника

КУРСОВОЙ ПРОЕКТ

По дисциплине «Проектирование систем на кристалле»

На тему: «Моделирование и сравнение потенциальных реализаций с использованием весов для оценки производительности конвейера»

Авторы: *студенты* *группы P41193*

*Евтушенко Олег Владимирович,*

*Прожирко Владислав Александрович,*

*Самойлов Владислав Романович*

Руководитель:*Березина Екатерина Михайловна,*

Курсовой проект выполнен и защищен с оценкой **\_\_\_\_**

Дата защиты **\_\_\_\_**

Санкт-Петербург

2020

**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

**Факультет программной инженерии и компьютерной техники**

**ЗАДАНИЕ**

**НА КУРСОВОЙ ПРОЕКТ**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Студенты:** | | | Евтушенко Олег Владимирович, Прожирко Владислав Александрович, | | | | | | | | |
| Самойлов Владислав Романович | | | | | | | | |
| **Группа:** | | | P41193 | | | | | | | | |
| **Руководитель:** | | | Березина Екатерина Михайловна | | | | | | | | |
|  | | |  | | | | | | | | |
| 1. **Наименование темы:** | | | | | Моделирование и сравнение потенциальных реализаций с | | | | | | |
| использованием весов для оценки производительности конвейера | | | | | | | | | | | |
|  | | | | | | | | | | | |
| 1. **Срок сдачи студентом законченной работы:** | | | | | | | 6 июля 2020 |  | | | |
| 1. **Техническое задание** | | | | |  | | | | | | |
| **3.1 Цель работы:** | | | | анализ и разработка потенциальной реализации и декодирования | | | | | | |
| команд стандартного набора инструкций RISC-V (RV32I Base Instruction Set), структуры | | | | | | | | | | | |
| памяти и работы с ней. | | | | | | | | | | | |
| **3.2 Объем работ** | | | | | | | | | | | |
| 1. Перечень элементов, которые планируется реализовать в симуляторе: | | | | | | | | | | | |
| * Ядро процессора должно содержать: | | | | | | | | | | | |
| * регистровый файл X0-X31; | | | | | | | | | | | |
| * счетчик инструкций PC; | | | | | | | | | | | |
| * декодер выбранных инструкций; | | | | | | | | | | | |
| * память размером 32Кб, адреса 0 - 0xFFFF; | | | | | | | | | | | |
| * Графический интерфейс и возможность пошагового исполнения | | | | | | | | | | | |
| * Вывод сообщений об ошибках: при ошибочном декодировании, при переполнении | | | | | | | | | | | |
| 1. Реализовать стандартный набор инструкций RV32I Base Instruction Set в симуляторе, | | | | | | | | | | | |
| кроме инструкций FENCE, CSRRW/S/C. | | | | | | | | | | | |
| 1. План на тестирование симулятора: | | | | | | | | | | | |
| * тестирование модуля работы с памятью; | | | | | | | | | | | |
| * тестирование модуля работы при декодировании; | | | | | | | | | | | |
| * тестирование модуля работы с командами. | | | | | | | | | | | |
| 1. Средства, которые планируется использовать для разработки, тестирования: | | | | | | | | | | | |
| * Visual Studio 2019; | | | | | | | | | | | |
| * RISC-V GNU toolchain; | | | | | | | | | | | |
| * Notepad++; | | | | | | | | | | | |
| * Спецификация RISC-V “The RISC-V Instruction Set Manual”. | | | | | | | | | | | |
| 1. Планирование работ: | | | | | | | | | | | |
|  |  | | | | | | | | | | |
|  | № работы | Наименование работы | | | | | | | | Срок завершения работы | |
|  | 1 | Анализ предметной области. | | | | | | | | 06.04.2020 | |
|  | 2 | Обзор существующих реализаций. | | | | | | | | 08.04.2020 | |
|  | 3 | Составление вариантов возможных решений задачи. | | | | | | | | 15.04.2020 | |
|  | 4 | Распределение задач по модулям между участниками проекта. | | | | | | | | 23.04.2020 | |
|  | 5 | Реализация модулей участниками проекта. | | | | | | | | 30.04.2020 | |
|  | 6 | Реализация связей между модулями проекта. | | | | | | | | 15.05.2020 | |
|  | 7 | Тестирование проекта. | | | | | | | | 22.05.2020 | |
|  | 8 | Сравнение потенциальных реализаций. | | | | | | | | 08.06.2020 | |
|  | 9 | Анализ результатов. | | | | | | | | 15.06.2020 | |
|  | 10 | Написание и оформление отчета. | | | | | | | | 25.06.2020 | |
|  | | | | | | | | | | | |
| **Дата выдачи задания:** | | | | 29 марта 2020 | |  | | | | | |
| Руководитель | | | | | | | | |  | | |
|  | | | | | | | | | (подпись) | | |
| Задание принял к исполнению | | | | | | | | |  | | |
|  | | | | | | | | | (подпись) | | |

### Содержание

[Содержание 2](#_Toc43998868)

[Введение 4](#_Toc43998869)

[1 Анализ предметной области 5](#_Toc43998870)

[1.1 Обзор архитектур RISC-V 5](#_Toc43998871)

[1.2 Регистры RISC-V 9](#_Toc43998872)

[2 Обзор существующих реализаций 10](#_Toc43998873)

[2.1 Venus 10](#_Toc43998874)

[2.1 Spike RISC-V ISA Simulator 11](#_Toc43998875)

[2.3 Выводы 11](#_Toc43998876)

[3 Разработка алгоритма ДЛЯ РЕШЕНИЯ задачи 13](#_Toc43998877)

[4 Распределение задач по модулям между участниками проекта и рассмотрение реализаций 16](#_Toc43998878)

[4.1 Обзор возможной программной реализации 16](#_Toc43998879)

[4.1.1 Java 16](#_Toc43998880)

[4.1.2 C/C++ 17](#_Toc43998881)

[4.1.3 C Sharp 17](#_Toc43998882)

[4.1.4 Python 18](#_Toc43998883)

[4.2 Обзор реализации пользовательского интерфейса 18](#_Toc43998884)

[4.2.1 Qt фреймворк 18](#_Toc43998885)

[4.2.2 Windows API 19](#_Toc43998886)

[4.2.3 Windows Forms 19](#_Toc43998887)

[4.2.4 Windows Presentation Foundation 20](#_Toc43998888)

[5 Реализация модулей участниками проекта 22](#_Toc43998889)

[5.1 Модуль декодирования инструкций 23](#_Toc43998890)

[5.2 Модуль реализации инструкций 26](#_Toc43998891)

[5.3 Модуль работы с памятью 27](#_Toc43998892)

[5.3.1 Операции с сегментом кода 27](#_Toc43998893)

[5.3.2 Операции с сегментами данных и стека 28](#_Toc43998894)

[5.3.3 Операции с регистрами общего назначения 28](#_Toc43998895)

[6 Реализация связей между модулями проекта 30](#_Toc43998896)

[7 Тестирование проекта 32](#_Toc43998897)

[7.1 Тестирование команд 32](#_Toc43998898)

[7.2 Тестирование режимов 49](#_Toc43998899)

[8 Сравнение потенциальных реализаций 50](#_Toc43998900)

[8.1 Реализация без конвейера 50](#_Toc43998901)

[8.2 Реализация с 5-стадийным конвейером 51](#_Toc43998902)

[8.3 Сравнение реализаций 53](#_Toc43998903)

[9 Руководство пользователя 54](#_Toc43998904)

[Заключение 56](#_Toc43998905)

[Список литературы 57](#_Toc43998906)

[Приложение А 59](#_Toc43998907)

[Приложение Б 64](#_Toc43998908)

[Приложение В 74](#_Toc43998909)

[Приложение Г 78](#_Toc43998910)

[Приложение Д 93](#_Toc43998911)

### Введение

Темой данного курсового проекта является моделирование и сравнение потенциальных реализаций с использованием весов для оценки производительности конвейера.

Цель работы: анализ, разработка и сравнение потенциальных реализаций, а также реализация и декодирование команд из стандартного набора инструкций RISC-V (RV32I Base Instruction Set), структуры памяти и работы с ней.

Основные этапы и задачи выполнения работы:

1. Анализ предметной области.
2. Обзор существующих реализаций.
3. Составление вариантов возможных решений задачи.
4. Распределение задач по модулям между участниками проекта.
5. Реализация модулей участниками проекта.
6. Реализация связей между модулями проекта.
7. Тестирование проекта.
8. Сравнение потенциальных реализаций.
9. Анализ результатов.

### 1 Анализ предметной области

RISC-V — открытая и свободная система команд (ISA — Instruction Set Architecture) и процессорная архитектура на основе концепции RISC для микропроцессоров и микроконтроллеров. Создана в 2010 году исследователями из Computer Science Division, калифорнийского университета в Беркли при непосредственном участии Дэвида Паттерсона.

Стандарт RISC-V определяет сравнительно небольшое число стандартных инструкций, около 50 штук, многие из которых были типичны еще для ранних RISC-I 1980 года. Стандартные расширения (M, A, F и D) расширяют набор на 53 инструкции, сжатый формат C определяет 34 команды. Используется 6 типов кодирования инструкций (форматов) [1].

#### 1.1 Обзор архитектур RISC-V

Базовое подмножество команд использует следующий набор регистров: специальный регистр x0 (zero), 31 целочисленный регистр общего назначения (x1 — x31), регистр счётчика команд (PC, используется только косвенно), а также множество CSR (Control and Status Registers, может быть адресовано до 4096 CSR).

Для встраиваемых применений может использоваться вариант архитектуры RV32E (Embedded) с сокращённым набором регистров общего назначения. Уменьшение количества регистров позволяет не только экономить аппаратные ресурсы, но и сократить затраты памяти и времени на сохранение/восстановление регистров при переключениях контекста.

При одинаковой кодировке инструкций в RISC-V предусмотрены реализации архитектур с 32, 64 и 128-битными регистрами общего назначения и операциями (RV32I, RV64I и RV128I соответственно).

Разрядность регистровых операций всегда соответствует размеру регистра, а одни и те же значения в регистрах могут трактоваться целыми числами как со знаком, так и без знака. Нет операций над частями регистров, нет каких-либо выделенных «регистровых пар».

Операции не сохраняют где-либо биты переноса или переполнения, что приближено к модели операций в языке программирования C. Также аппаратно не генерируются исключения по переполнению и даже по делению на 0. Все необходимые проверки операндов и результатов операций должны производиться программно [2].

Целочисленная арифметика расширенной точности (большей, чем разрядность регистра) должна явно использовать операции вычисления старших битов результата. Например, для получения старших битов произведения регистра на регистр имеются специальные инструкции.

Архитектура использует только little-endian модель — первый байт операнда в памяти соответствует наименее значащим битам значений регистрового операнда.

Для пары инструкций сохранения/загрузки регистра операнд в памяти определяется размером регистра выбранной архитектуры, а не кодировкой инструкции (код инструкции один и тот же для RV32I, RV64I и RV128I, но размер операндов 4, 8 и 16 байт соответственно

Для всех допустимых размеров операндов в памяти, меньших, чем размер регистра, имеются отдельные инструкции загрузки/сохранения младших битов регистра, в том числе для загрузки из памяти в регистр есть парные варианты инструкций, которые позволяют трактовать загружаемое значение как со знаком (старшим знаковым битом значения из памяти заполняются старшие биты регистра) или без знака (старшие биты регистра устанавливаются в 0).

Инструкции базового набора имеют длину 32 бита с выравниванием на границу 32-битного слова, но в общем формате предусмотрены инструкции различной длины (стандартно — от 16 до 192 бит с шагом в 16 бит) с выравниванием на границу 16-битного слова. Полная длина инструкции декодируется унифицированным способом из её первого 16-битного слова. Список наборов команд представлен в таблице 1.

Для наиболее часто используемых инструкций стандартизовано применение их аналогов в более компактной 16-битной кодировке (C — Compressed extension).

Операции умножения, деления и вычисления остатка не входят в минимальный набор инструкций, а выделены в отдельное расширение (M — Multiply extension). Имеется ряд доводов в пользу разделения и данного набора на два отдельных (умножение и деление).

Стандартизован отдельный набор атомарных операций (A — Atomic extension) [2].

Таблица 1.1 — Список наборов команд

|  |  |
| --- | --- |
| **Сокращение** | **Наименование** |
| **Базовые наборы** | |
| **RV32I** | Базовый набор с целочисленными операциями, 32-битный |
| **RV32E** | Базовый набор с целочисленными операциями для встраиваемых систем, 32-битный, 16 регистров |
| **RV64I** | Базовый набор с целочисленными операциями, 64-битный |
| **RV128I** | Базовый набор с целочисленными операциями, 128-битный |
| **Стандартные расширенные наборы** | |
| **M** | Целочисленное умножение и деление (Integer Multiplication and Division) |
| **A** | Атомарные операции (Atomic Instructions) |
| **F** | Арифметические операции с плавающей запятой над числами одинарной точности (Single-Precision Floating-Point) |

Таблица 1.1 — Список наборов команд (продолжение)

|  |  |
| --- | --- |
| **D** | Арифметические операции с плавающей запятой над числами двойной точности (Double-Precision Floating-Point) |
| **G** | Сокращеное обозначение для комплекта из базового и стандартного наборов команд |
| **Q** | Арифметические операции с плавающей запятой над числами четвертной точности |
| **L** | Арифметические операции над числами с фиксированной запятой (Decimal Floating-Point) |
| **C** | Сокращённые имена для команд (Compressed Instructions) |
| **B** | Битовые операции (Bit Manipulation) |
| **J** | Двоичная трансляция и поддержка динамической компиляции (Dynamically Translated Languages) |
| **T** | Транзакционная память (Transactional Memory) |
| **P** | Короткие SIMD-операции (Packed-SIMD Instructions) |
| **V** | Векторные расширения (Vector Operations) |
| **N** | Инструкции прерывания (User-Level Interrupts) |

RISC-V состоит из базового 32-разрядного целочисленного набора инструкций пользовательского уровня. ISA RV32I, она включает в себя 47 инструкций, которые могут быть сгруппированы в шесть типов:

* R-тип: регистр-регистр;
* I-тип: short immediates and loads;
* S-тип: команды загружающие значения в память;
* B-тип: условные переходы;
* U-тип: long immediates;
* J-тип: безусловные переходы.

#### 1.2 Регистры RISC-V

RISC-V имеет 32 (или 16 для встраиваемых применений) целочисленных регистра. При реализации вещественных групп команд есть дополнительно 32 вещественных регистра.

Для операций над числами в бинарных форматах плавающей запятой используется набор дополнительных 32 регистров FPU (Floating Point Unit), которые совместно используются расширениями базового набора инструкций для трёх вариантов точности: одинарной — 32 бита (F extension), двойной — 64 бита (D — Double precision extension), а также четверной — 128 бит (Q — Quadruple precision extension) [2].

### 2 Обзор существующих реализаций

В данном разделе рассмотрены несколько существующих реализаций симулятора RISC-V и сделаны выводы о дальнейшем реализуемом функционале программного решения.

#### 2.1 Venus

Venus – это симулятор набора инструкций RISC-V, реализованный в виде web-приложения с интуитивно понятным и эргономическим графическим интерфейсом. Графический интерфейс содержит:

* элементы управления симулятором;
* окна информации о выполняемой программе;
* окна информации о содержимом регистров и памяти, представляемое в различных системах счисления (hex, decimal, unsigned, ASCII).

Данный продукт создан для обучения, как заявляет разработчик, и реализует следующий функционал:

* стандартный набор инструкций RV32I;
* расширение набора команд M;
* все стандартные псевдо-инструкции RISC-V, если они преобразуются в серию поддерживаемых инструкций.

На данный момент, не поддерживаются инструкции счетчика циклов и такие команды как ebreak, fence, fence.i.

Более подробную информацию можно найти в GitHub репозитории Venus [3].

#### 2.1 Spike RISC-V ISA Simulator

Spike – функциональный симулятор программного обеспечения RISC-V ISA на языке C/C++. Реализует функциональную модель одного или нескольких хартов RISC-V, а также он моделирует систему кэша.

Данный симулятор имеет следующие основные особенности:

* Несколько ISA: RV32I /M/A/F/D/Q/C/V расширения.
* Модели с несколькими типами памяти: RVWMO и RVTSO.
* Привилегированная спецификация: машинный, супервизор, пользовательские режимы.
* Пошаговая отладка с поддержкой просмотра содержимого памяти / регистров.
* Поддержка многоядерных процессоров.
* Поддержка JTAG.

Более подробную информацию можно найти в GitHub репозитории Spike [4].

#### 2.3 Выводы

Проанализировав вышеописанные симуляторы RISC-V и возможности участников проекта, делаем вывод о реализуемом функционале программного решения.

Разрабатываемое программное решение должно содержать:

1. Минимальный набор инструкций (стандартный набор RV32I).

2. Стандартные псевдо-инструкции.

3. Пошаговая отладка с возможностью просмотра содержимого памяти и регистров общего назначения.

4. Реализация одностадийного и 5-стадийного конвейеров.

5. Графический интерфейс с элементами управления и отображения информации о выполняемой программе.

6. Реализация счетчика инструкций (Program Counter).

7. Возможность сохранения log-файла об исполняемой или выполненной программе.

8. Сохранение дампов (файлов содержимого регистров и памяти) о выполненной программе.

### 3 Разработка алгоритма и его реализация

В данном курсовом проекте рассмотрены реализации симулятора RISC-V со стандартным набор инструкций с использованием 1-стадийного и 5-стадийного конвейеров [6].

Исходя из выводов предыдущего раздела можно составить общий алгоритм работы для реализаций симулятора, который основывается на структуре 5-стадийного конвейера:

Выбор инструкции из памяти.

Декодирование инструкции.

Выполнение инструкции.

Запись/чтение из памяти.

Запись в регистр.

Данный алгоритм для реализации без конвейера выполнятся последовательно, а с использованием 5-стадийного конвейера –параллельно [7].

Для реализаций, первоначально, необходимо из программы на ассемблере получить машинный код всех выполняемых операций в программе. Для этого использована часть из RISC-V GNU toolchain компании Syntacore.

Чтобы использовать toolchain в качестве получения машинного кода написан текстовый файл, содержащий последовательность команд, предназначенных для использования командным интерпретатором. Данный файл имеет расширение .BAT.

Команды, использованные в файле представлены на рисунке 3.1.

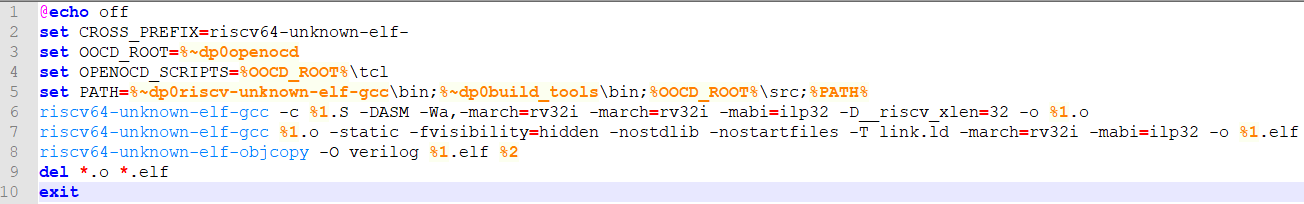


Рисунок 3.1 – Команды bat-файла

Описание команд:

1. Команда «@echo off» отключает режим вывода команд на экран.
2. Команды типа «set var=value» указывают пути для запуска тулчейна.
3. С помощью использования команды «riscv64-unknown-elf-gcc» получаем object-файл, а затем elf-файл.
4. С помощью команды «riscv64-unknown-elf-objcopy» из object-файла получаем hex-файл. Это выполняется с помощью скрипта компоновщика [8] (файл «link.ld» представлен на рисунке 3.2), чтобы описать, как разделы во входных файлах должны быть отображены в выходной hex-файл. Сегмент кода начинается с адреса «00000000» и заканчивается «00000FFF». Сегмент данных начинается с адреса «00001000» и заканчивается «00007FF0».

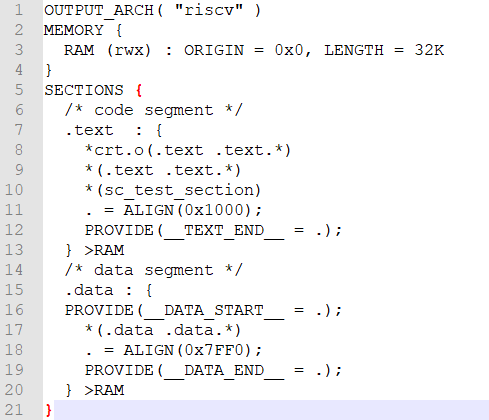


Рисунок 3.2 – Скрипт компоновщика

После получение hex-файла (файла памяти) можно начинать считывание закодированных инструкций и выполнять их по 5 стадиям конвейера.

Основываясь на вышеописанном алгоритме, разработана общая структура симулятора, которая представлена на рисунке 3.1.



Рисунок 3.1 – Общая структура работы симулятора

### 4 Распределение задач по модулям между участниками проекта и рассмотрение реализаций

Для удобства решения задачи можно использовать разделение на следующие модули:

* Модуль работы с памятью. В данном модуле необходимо реализовать запись/чтение из памяти и регистров общего назначения. Разработчик: студент Евтушенко О.В.
* Модуль декодирования инструкций. В данном модуле необходимо реализовать декодирование инструкции из стандартного набора RV32I. Разработчик: студент Самойлов В.Р.
* Модуль реализаций выполнения инструкций. В данном модуле необходимо реализовать выполнение инструкции из стандартного набора RV32I. Разработчик: студент Прожирко В.А.

#### Обзор возможной программной реализации

Обращая внимание на ограниченные временные рамки и основываясь на имеющихся знаниях и навыков студентов для выполнения курсового проекта рассмотрены следующие языки программирования:

* C/C++.
* C Sharp.
* Python.

##### 4.1.1 C/C++

C/C++ — это компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. Язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы. Имеется возможность работы с памятью на низком уровне. Для разработки интерфейса используется WinForms и WinApi.

##### 4.1.2 C Sharp

C Sharp — это язык программирования общего назначения, который следует парадигме объектно-ориентированного программирования. Предоставляет много инструментов, библиотек и IDE для реализации ПО на этом языке программирования. C# позволяет разработчику сосредоточиться на алгоритме, а не на деталях реализации, так как сложные конструкции в нём заключены в абстракции. Для разработки интерфейса используется WinForm и WPF.

##### 4.1.3 Python

Python – это высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Для разработки интерфейса ПО на Python используется PyQt и Qt Designer.

#### 4.2 Обзор реализации пользовательского интерфейса

Пользовательский интерфейс можно реализовать с помощью следующих компонентов:

* Windows API.
* Windows Forms.
* Windows Presentation Foundation.

##### 4.2.1 Windows API

Windows API (англ. application programming interfaces) — общее наименование набора базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows корпорации «Майкрософт». Windows API представляет собой множество функций, структур данных и числовых констант, следующих соглашениям языка Си. Все языки программирования, способные вызывать такие функции и оперировать такими типами данных в программах, исполняемых в среде Windows, могут пользоваться этим API. В частности, это языки C++, Java с использованием библиотеки JNA, C# при использовании атрибута DllImport("user32.dll"), Python с использованием PyWin32, Visual Basic и многие другие.

##### 4.2.2 Windows Forms

Windows Forms — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, С++, так и на VB.Net, J# и др.

##### 4.2.3 Windows Presentation Foundation

Windows Presentation Foundation (WPF) — аналог WinForms, система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая подсистема в составе .NET Framework (начиная с версии 3.0), использующая язык XAML.

В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление, включая язык XAML (eXtensible Application Markup Language).

XAML представляет собой язык декларативного описания интерфейса, основанный на XML. Также реализована модель разделения кода и дизайна, позволяющая кооперироваться программисту и дизайнеру.

Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Windows Forms, где используется GDI/GDI+. Производительность WPF выше, чем у GDI+ за счёт использования аппаратного ускорения графики через DirectX.

Для работы с WPF требуется любой .NET-совместимый язык. В этот список входит множество языков: C#, VB.NET, C++ и многие другие. Для полноценной работы может быть использована как Visual Studio, так и Expression Blend. Первая ориентирована на программирование, а вторая — на дизайн и позволяет делать многие вещи, не прибегая к ручному редактированию XAML.

### 5 Реализация модулей участниками проекта

Из-за ограниченных временных рамок для программной реализации выбран язык C Sharp, так как он прост в понимании для всех участников проекта и каждый из студентов имеет достаточно опыта работы с ним.

В качестве реализации пользовательского интерфейса выбрана Windows Presentation Foundation (WPF), так как с помощью неё можно реализовать стилизация, состояния, создание элементов управления и так далее. При этом WPF обладает высокой производительностью за счёт использования аппаратного ускорения графики, что особенно необходимо при разработке ПО с графическим интерфейсом.

Далее в разделе описана реализация каждого из модулей проекта.

Для декодирования и реализации инструкций выбраны команды из стандартного набора RV32I. Список выбранных команд [2] представлен в таблице 5.1.

Таблица 5.1 – Реализованные команды из RV32I

|  |  |
| --- | --- |
| Тип команды | Команда |
| Загрузка констант | LUI |
| AUIPC |
| Безусловный переход | JAL |
| JALR |
| Условный переход | BEQ |
| BNE |
| BLT |
| BGE |
| BLTU |
| BGEU |
| Запись в память | SB |
| SH |
| SW |

Таблица 5.1 – Реализованные команды из RV32I (Продолжение)

|  |  |
| --- | --- |
| Чтение из памяти | LB |
| LH |
| LW |
| LBU |
| LHU |
| Арифметические с константой | ADDI |
| SLTI |
| SLTIU |
| XORI |
| ORI |
| ANDI |
| SLLI |
| SRLI |
| SRAI |
| Арифметические с регистрами | ADD |
| SUB |
| SLL |
| SLT |
| SLTU |
| XOR |
| SRL |
| SRA |
| OR |
| AND |
| Вызов окружающей среды | ECALL |

#### 5.1 Модуль декодирования инструкций

Модуль предназначен для декодирования, считаных из памяти инструкций на стадии конвейера Fetch. Стадия расшифровки инструкций имеет название Decode.

Листинг модуля декодирования инструкций представлен в Приложении А.

Данный модуль реализует декодирование команд всех типов, имеющихся в архитектуре RV32I [2], представленных на Рис. 5.1.

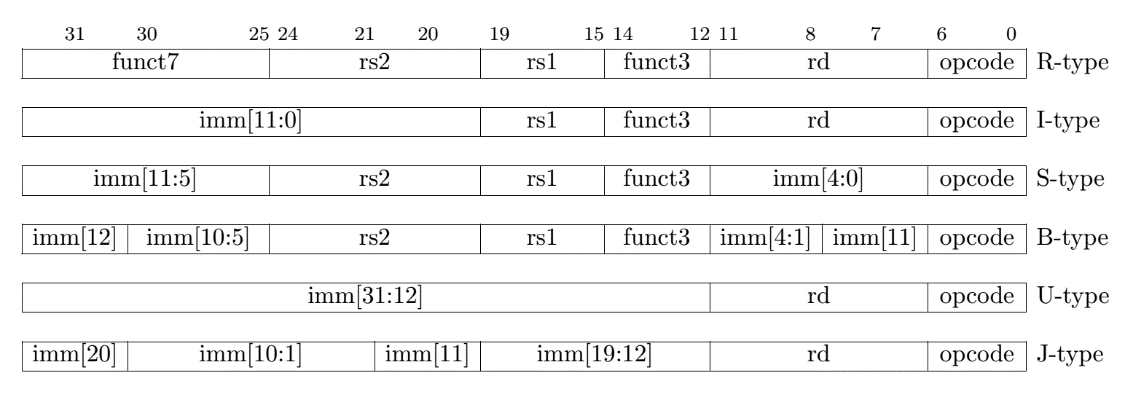


Рисунок 5.1 – Форматы инструкций RV32I

* rs1 – номер регистра, в котором находится первый операнд;
* rs2 – номер регистра, в котором находится второй операнд;
* rd – номер регистра, в который будет записан результат;
* imm – константа, передаваемая напрямую через операцию;
* opcode – код операции;
* funct3 – дополнительное поле, размером 3 бита для определения инструкции;
* funct7 – дополнительное поле, размером 7 бит для определения инструкции.

Изначально в модуль поступает набор данных длинной word (4 байта) в шестнадцатеричной системе счисления, после чего команда расшифруется и принимает вид **Command <OP1, OP2, OP3>**.

Это происходит следующим образом:

1. Команда переводится из 16сс в 2сс;
2. Последовательность разбивается на значимые поля Opcode, Rd, Funct3, Funct7 и др.;
3. Анализируется значимые поля для определения команды (путем сопоставления полученных последовательностей с файлом-базой, содержащим команды и значения полей Opcode, Funct3, Funct7 и др.)
4. Составление ассемблерного и упрощенного (без разделительных знаков между операндами для передачи внутри модулей) представления команды;
5. Передача расшифрованной команды из модуля в основную программу.

Файл-база, содержащий значащие поля каждой команды для сопоставления с полученным набором данных, представлен на Рис. 5.2.

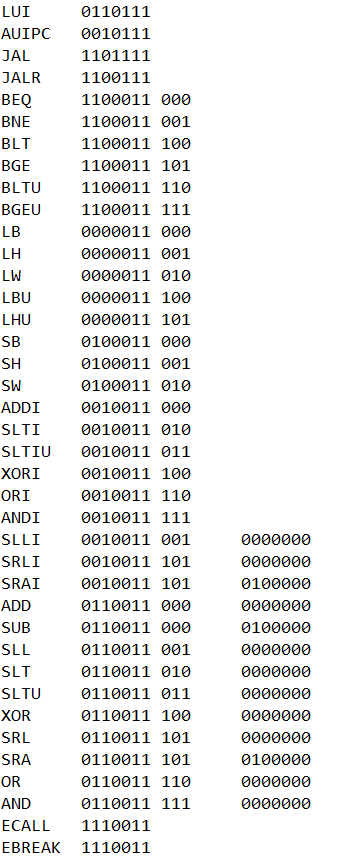


Рисунок 5.2 – Файл-база команд архитектуры RISC-V

Для удобства дальнейшей работы с командой, внутри модулей она передается в упрощенном виде — как набор данных, разделенных пробелами, т.е. команда «ADD x5, x6, x7» будет иметь вид «ADD x5 x6 x7». В интерфейс программы выводится полное представление команды со всеми разделительными символами.

Следует отметить, что в данном модуле реализована на элементарном уровне проверка корректности инструкций. Если инструкция не корректная, то выполнение программы прерывается и в консоль выводится сообщение об ошибке.

#### 5.2 Модуль реализации инструкций

Данный модуль используется для стадии Execute, а именно для выполнения инструкции на АЛУ.

Модуль содержит реализацию инструкций [2], представленных в таблице 5.1. Реализация выполнена с помощью функции Execute (), которая в качестве параметра принимает строку с названием инструкции и операндами для её выполнения.

Возвращает данная функция строку с разными значениями, которые описаны в таблице 5.2.

Таблица 5.2 – Значения возвращаемой строки

|  |  |
| --- | --- |
| **Строка** | **Значение** |
| **””** | пустая строка, если операции безусловного перехода/ ECALL (x10 = 0xB/0x1), т.е. нет записи/ чтения из памяти и записи в регистр (стадии Memory и Write back без операций) |
| **“value register”** | **value** – значение, которое необходимо записать в регистр.  а **register** – имя регистра для записи |

Таблица 5.2 – Значения возвращаемой строки (продолжение)

|  |  |
| --- | --- |
| **“address data”** | **address** – это адрес, по которому необходимо записать значение из регистра (команды SB, SH, SW)  **data –** значение, которое необходимо записать по адресу в память |
| **“value register”** | **value** – значение, которое необходимо записать в регистр.  а **register** – имя регистра для записи |
| **“ecall exit”** | сигнализирует о завершении программы |

Листинг данного модуля приведен в приложении Б.

#### 5.3 Модуль работы с памятью

Данный модуль используется для чтения и записи в память, а также в регистры общего назначения.

На стадиях Fetch, Memory, Write back необходима работа с памятью, то есть запись и чтение данных из неё. Следовательно, на данных стадиях используется модуль работы с памятью.

Память представлена в виде файла, в котором строчками начинающими с «@» помечены сегменты памяти.

Память разделена на следующие секции:

* Сегмент кода.
* Сегмент данных.
* Сегмент стека.

##### 5.3.1 Операции с сегментом кода

Сегмент кода располагается в начале файла, а именно со строки «@00000000» до начала Data-сегмента.

С сегментом кода выполняется только операция чтения инструкции по счетчику команд (PC). Чтение выполняется в главной программе (Приложение Г) с помощью функции Read\_code ().

##### 5.3.2 Операции с сегментами данных и стека

Сегмент данных располагается начиная со строки «@00001000» до сегмента стека. Сегмент стека располагается начиная со строки «@00007F00» до конца файла.

С данными сегментами выполняются операции чтения и записи.

Операция чтения выполняется с помощью соответствующих функций в зависимости от необходимого количества байт.

Функции для чтения с сегмента данных:

* Read\_data\_byte () – чтение 1 байта;
* Read\_data\_hw () – чтение 2 байт;
* Read\_data\_word () – чтение 4 байт.

В качестве параметра данные функции принимают адрес, по которому необходимо произвести операцию чтения.

В реализациях функций предусмотрен вариант с расположением данных по не выровненному адресу.

Операция записи выполняется с помощью функции Write\_data (). В качестве параметров данная функция принимает следующие:

* + - 1. Адрес, по которому необходимо выполнить запись.
      2. Значение, которое необходимо записать в память.

##### 5.3.3 Операции с регистрами общего назначения

С регистрами общего назначения можно выполнять операции чтения и записи.

Так как регистры общего назначения хранятся в словаре (Dictionary) и имеют модификатор доступа public с ключевым словом static, то запись и чтение выполняют без дополнительны функций, а просто с обращением в словарь по ключу – имени регистра.

Для реализации выбрана архитектура RV32I, следовательно регистров общего назначения – 32 [2], они представлены в таблице 5.3.

Таблица 5.3 – Регистры общего назначения

|  |  |  |
| --- | --- | --- |
| **Имя регистра** | **ABI имя** | **Описание** |
| X0 | zero | Всегда 0 |
| X1 | RA | Return address |
| X2 | SP | Stack pointer |
| X3 | GP | Global pointer |
| X4 | TP | Thread pointer |
| X5 | T0 | Temporary |
| X6 | T1 | Temporaries |
| X7 | T2 |
| X8 | S0/FP | Saved register/ Frame pointer |
| X9 | S1 | Saved register |
| X10 | A0 | Function arguments / return values |
| X11 | A1 |
| X12 | A2 | Function arguments |
| X13 | A3 |
| X14 | A4 |
| X15 | A5 |
| X16 | A6 |
| X17 | A7 |
| X18 | S2 | Saved registers |
| X19 | S3 |
| X20 | S4 |
| X21 | S5 |
| X22 | S6 |
| X23 | S7 |
| X24 | S8 |
| X25 | S9 |
| X26 | S10 |
| X27 | S11 |

Таблица 5.3 – Регистры общего назначения (продолжение)

|  |  |  |
| --- | --- | --- |
| X28 | T3 | Temporaries |
| X29 | T4 |
| X30 | T5 |
| X31 | T6 |

Листинг данного модуля приведен в приложении В.

### 6 Реализация связей между модулями проекта

Модули связываются с помощью основной программы, листинг которой представлен в приложении Г.

Модули подключаются с помощью классов:

1. Модуль декодирования с помощью включения класса Decoder.cs (приложение А) и файла Comand\_base.txt. Для того, чтобы использовать данный модуль необходимо создать экземпляр класса и вызвать функцию DecodeInstruction ().
2. Модуль реализации инструкций с помощью включения класса Alg\_operation.cs. для того, чтобы использовать данный модуль необходимо создать экземпляр класса и вызвать функцию Execute ().
3. Модуль памяти с помощью включения класса Memory.cs. Для того чтобы использовать данный модуль необходимо создать экземпляр класса и вызывать функции для работы с памятью: Read\_data\_byte (), Read\_data\_hw (), Read\_data\_word (), Write\_data (). А также обращаться к регистрам общего назначения через словарь (Dictionary) по ключу – имени регистра.

Алгоритм работы, описывает общую структуру работы симулятора, представленную на рисунке 6.1:

Выбор инструкции из памяти – функция Read\_code ().

Декодирование инструкции – функция DecodeInstruction ().

Выполнение инструкции – функция Execute ().

Запись/чтение из памяти – функции Read\_data\_byte (), Read\_data\_hw (), Read\_data\_word (), Write\_data ().

Запись в регистр – обращение к словарю (Registers).



Рисунок 6.1 – Общая структура работы симулятора

Данный алгоритм для реализации без конвейера выполнятся последовательно, а с использованием 5-стадийного конвейера –параллельно.

### 7 Тестирование проекта

Для проверки работоспособности симулятора необходимо провести тестирование каждой команды, а также двух разработанных режимов. Проект доступен на github по ссылке: https://github.com/VladislavProzhirko/Simulator\_RISC-V.git

#### 7.1 Тестирование команд

Первоначально протестированы команды занесения констант в регистр: LUI (Рисунок 7.1) и AUIPC (Рисунок 7.2).

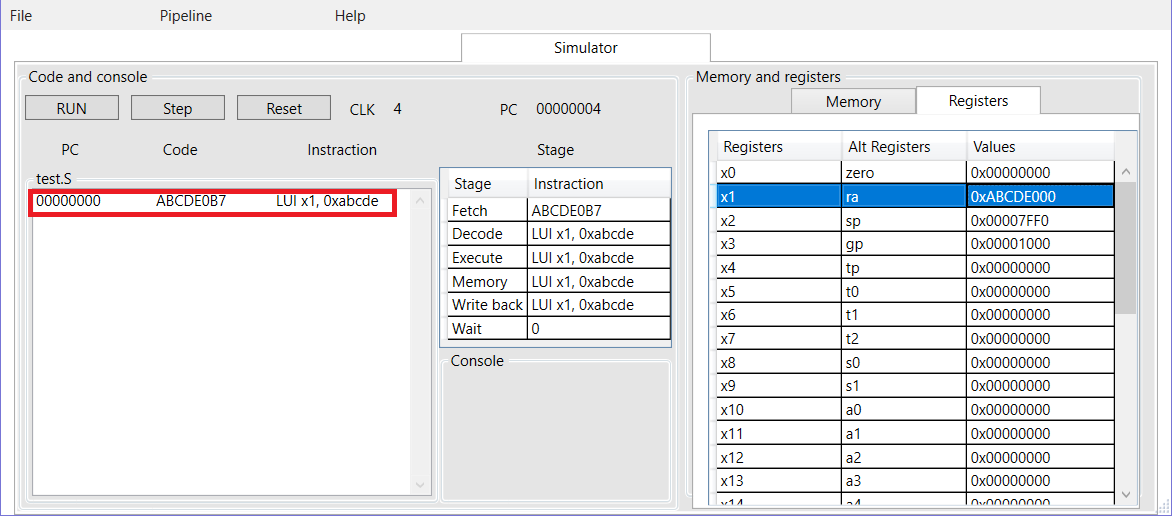


Рисунок 7.1 – Тестирование команды LUI

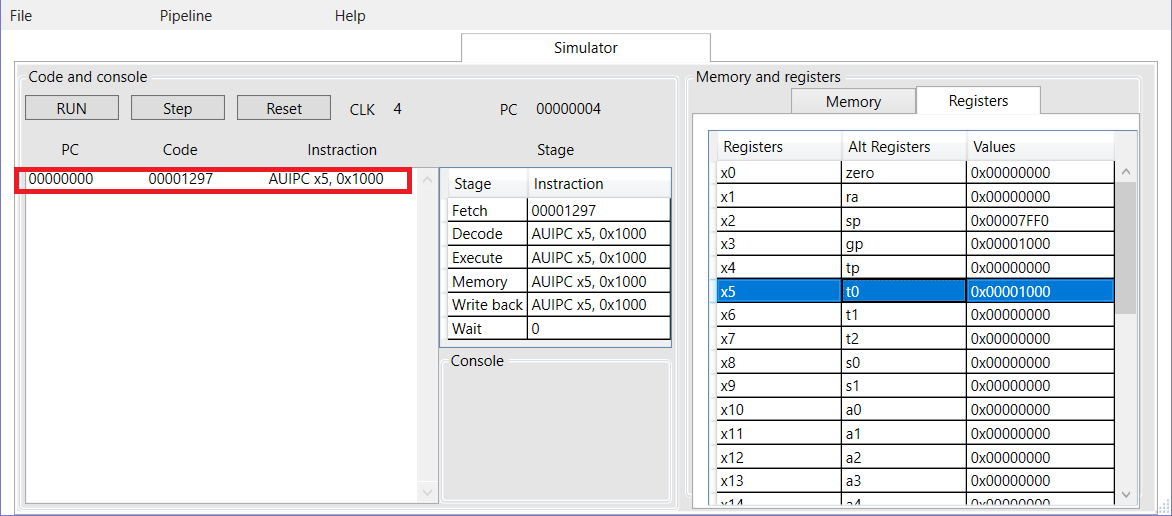


Рисунок 7.2 – Тестирование команды AUIPC

Затем протестированы команды записи/чтения из памяти: LB (Рисунок 7.3), LH (Рисунок 7.4), LW (Рисунок 7.5), LHU (Рисунок 7.6), LBU (Рисунок 7.7), SB (Рисунок 7.8), SH (Рисунок 7.9), SW (Рисунок 7.10).

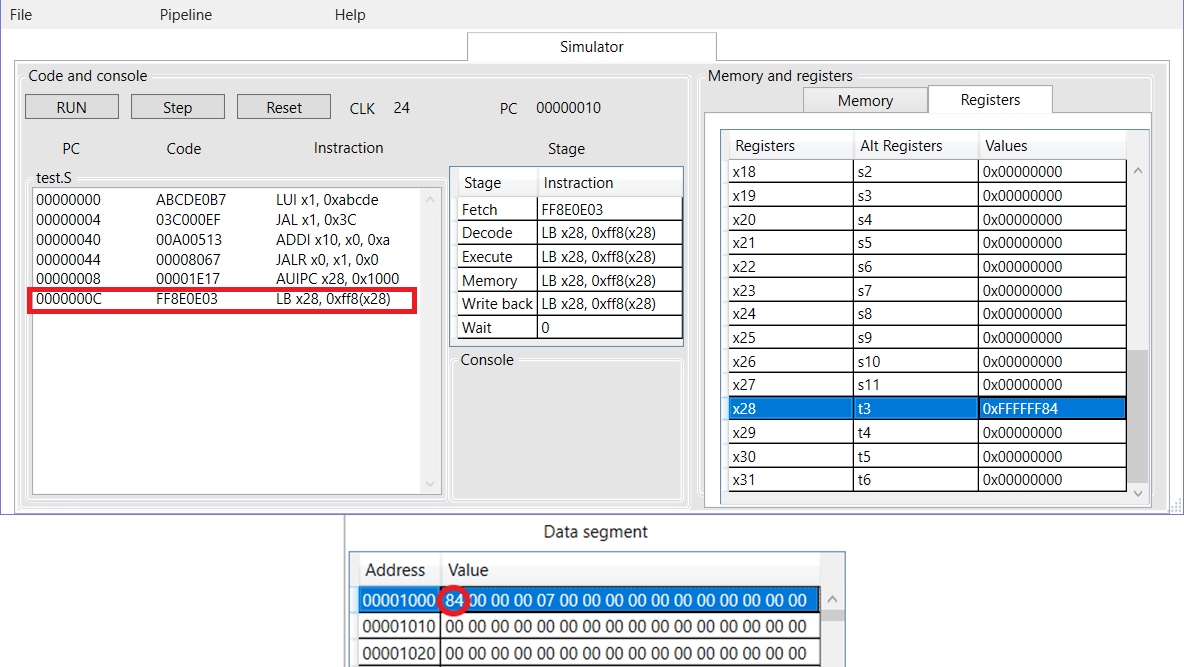


Рисунок 7.3 – Тестирование команды LB

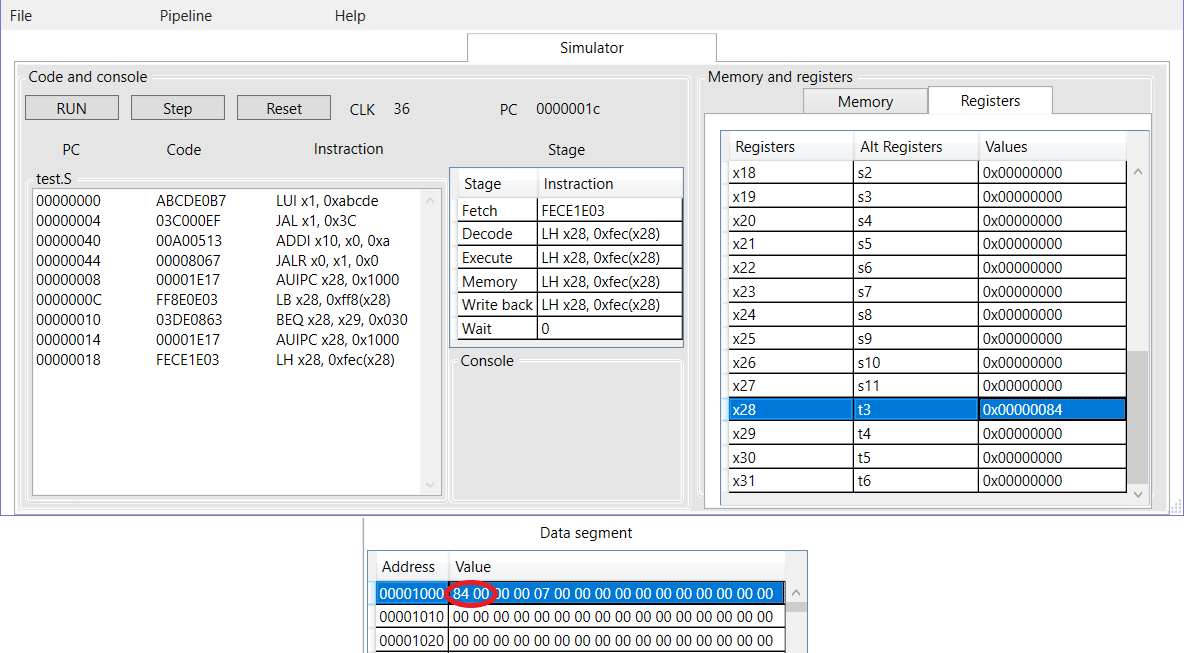


Рисунок 7.4 – Тестирование команды LH

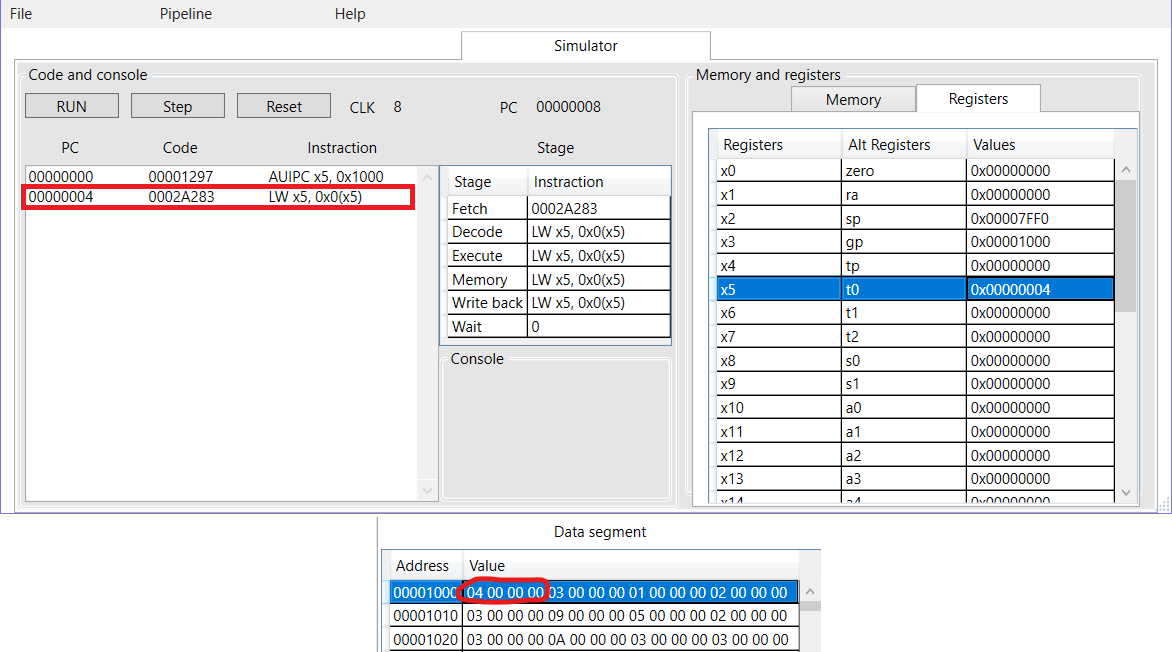


Рисунок 7.5 – Тестирование команды LW

 Рисунок 7.6 – Тестирование команды LBU

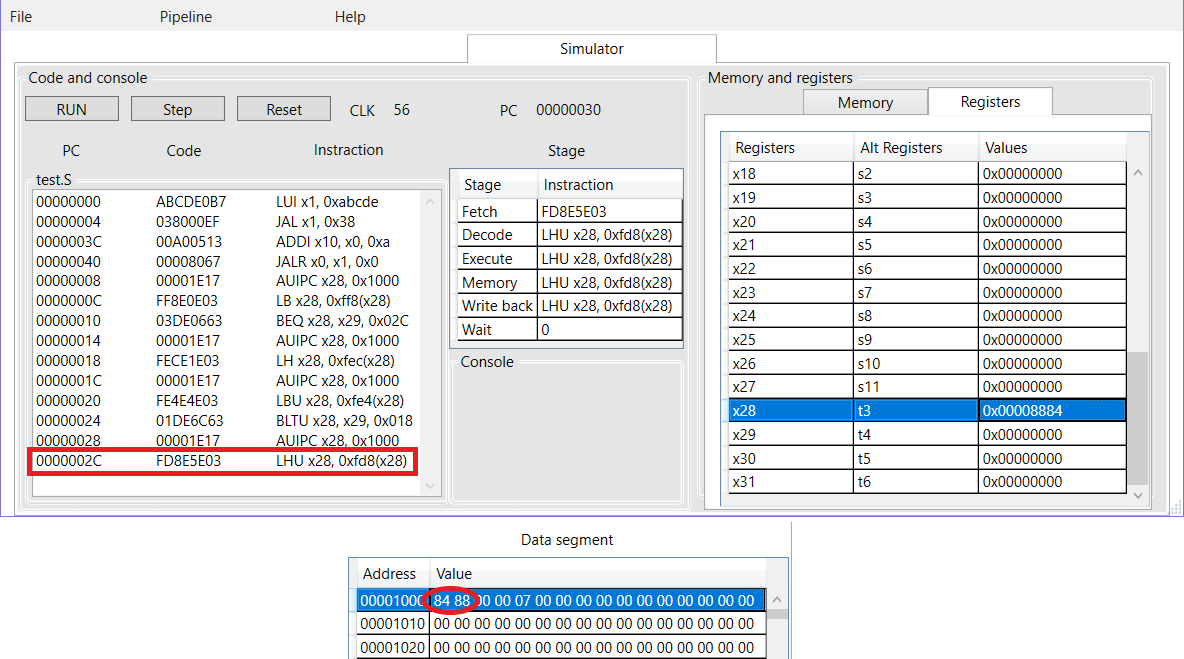


Рисунок 7.7 – Тестирование команды LHU

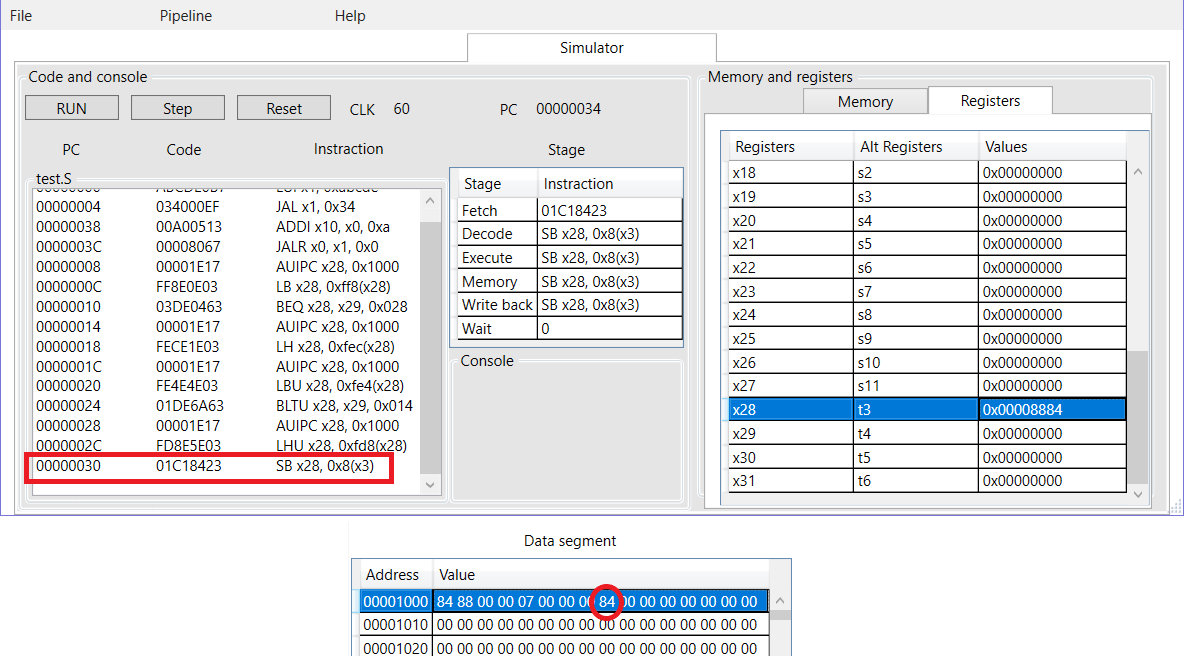


Рисунок 7.8 – Тестирование команды SB

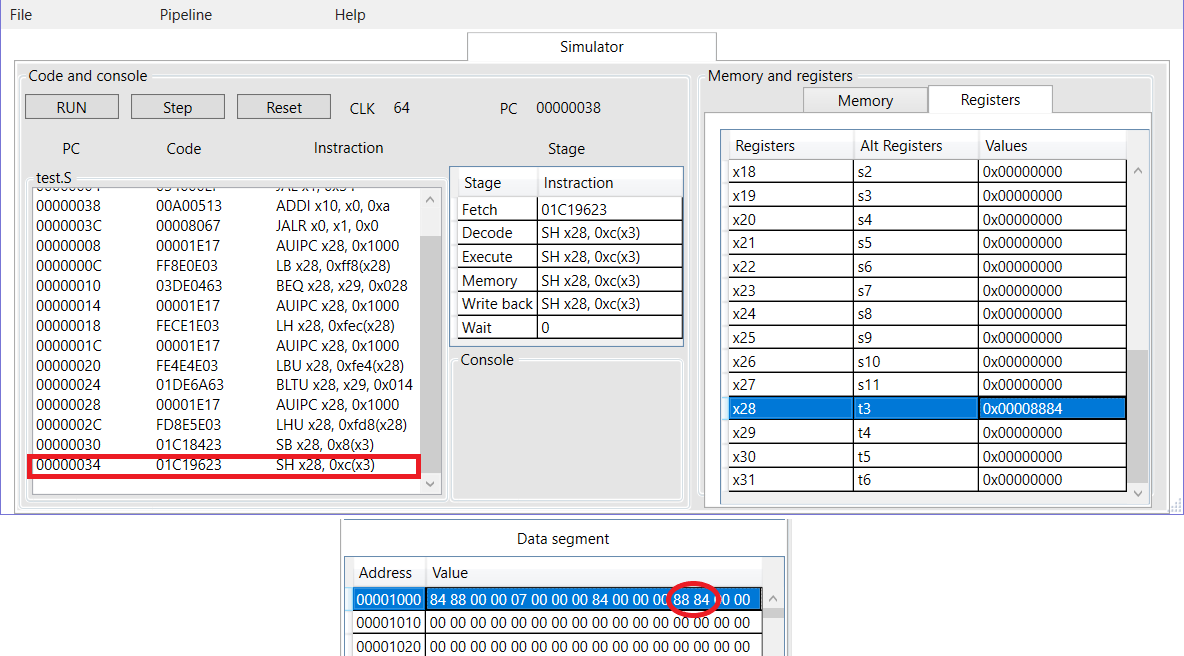


Рисунок 7.9 – Тестирование команды SH

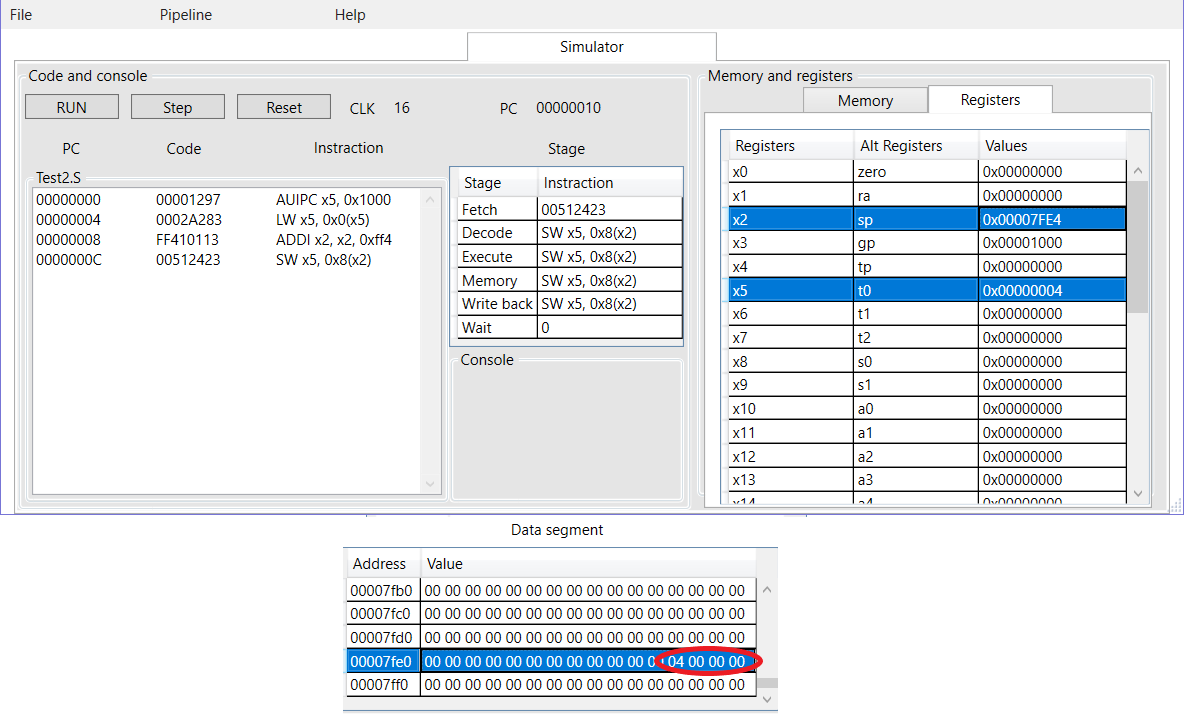


Рисунок 7.10 – Тестирование команды SH

Далее протестированы арифметические операции с константой: SLTI (Рисунок 7.11), SLTIU (Рисунок 7.12), XORI (Рисунок 7.13), ANDI (Рисунок 7.14), ORI (Рисунок 7.15), SLLI (Рисунок 7.16), SRLI (Рисунок 7.17), SRAI (Рисунок 7.18), ADDI (Рисунок 7.19).

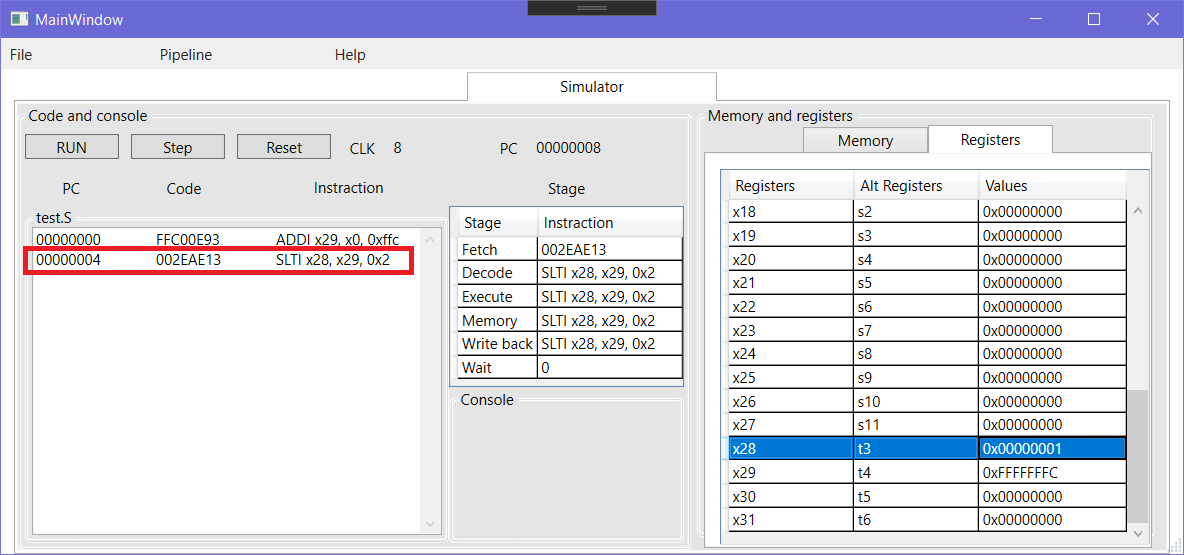


Рисунок 7.11 – Тестирование команды SLTI

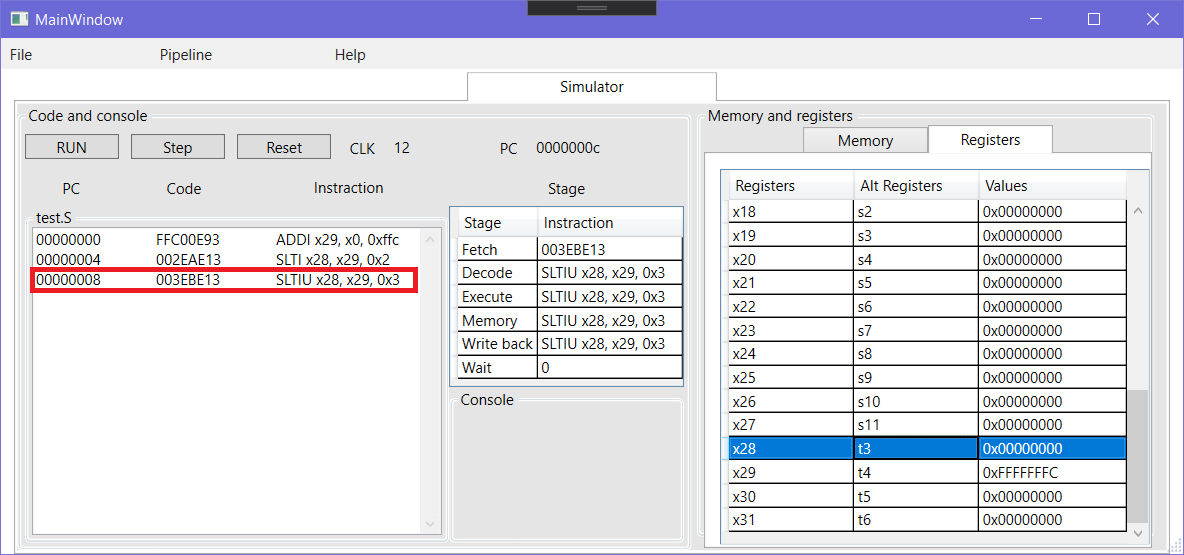


Рисунок 7.12 – Тестирование команды SLTIU

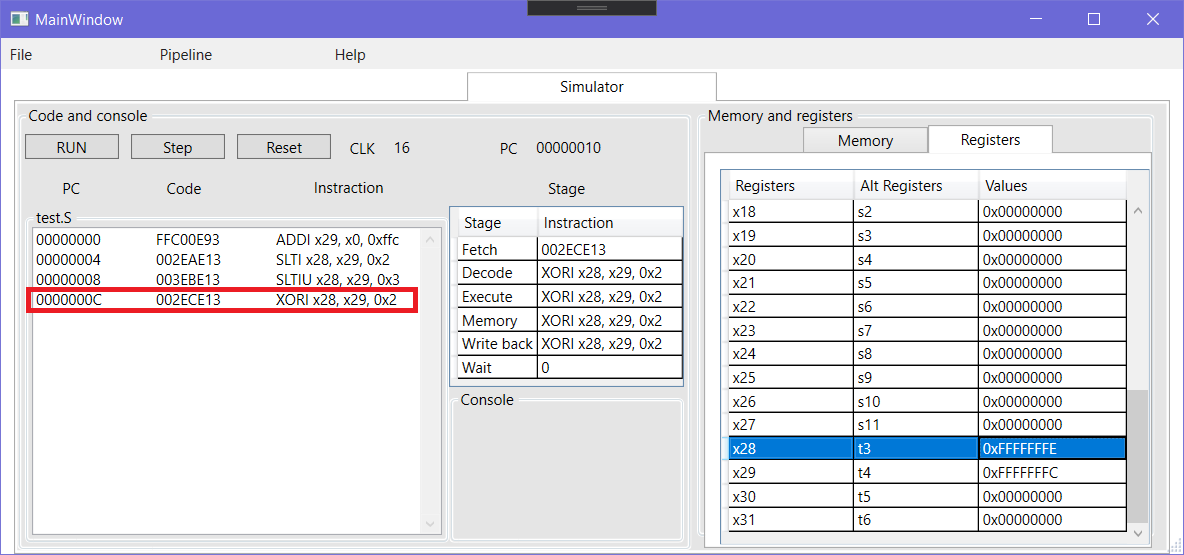


Рисунок 7.13 – Тестирование команды XORI

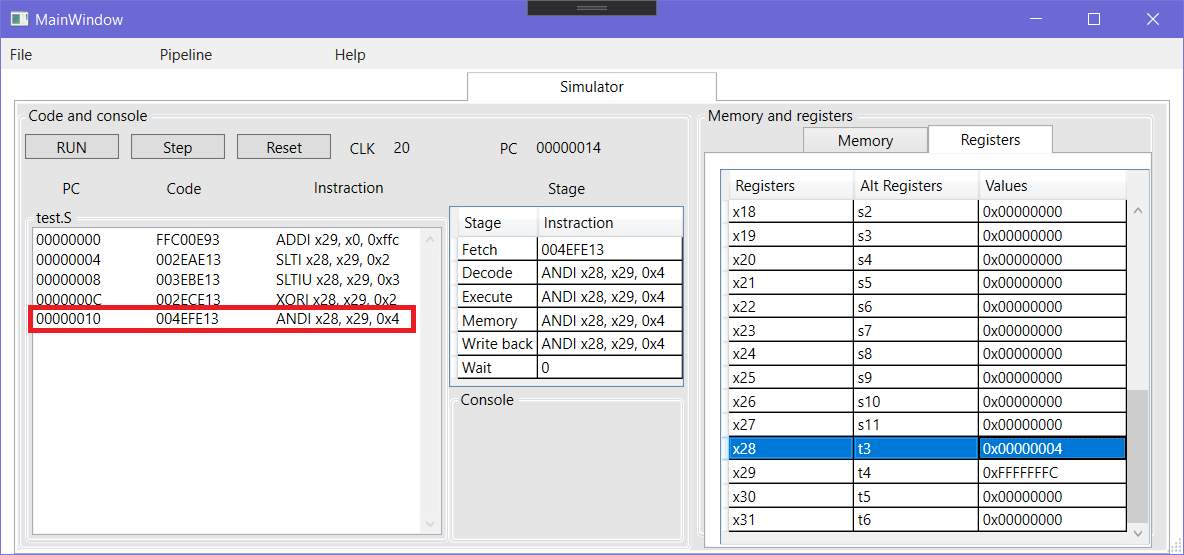


Рисунок 7.14 – Тестирование команды ANDI

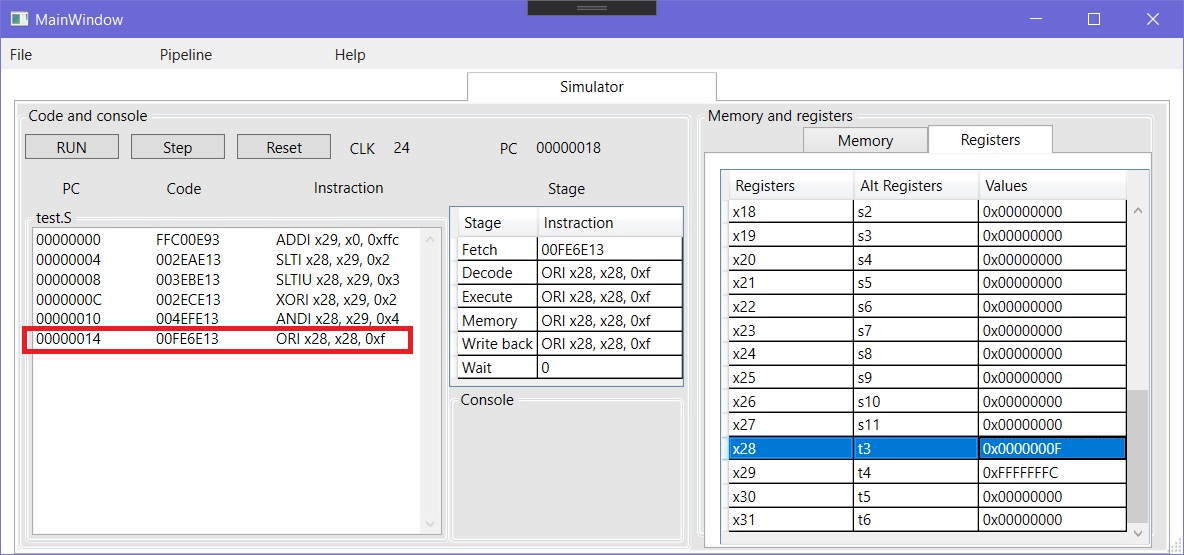


Рисунок 7.15 – Тестирование команды ORI

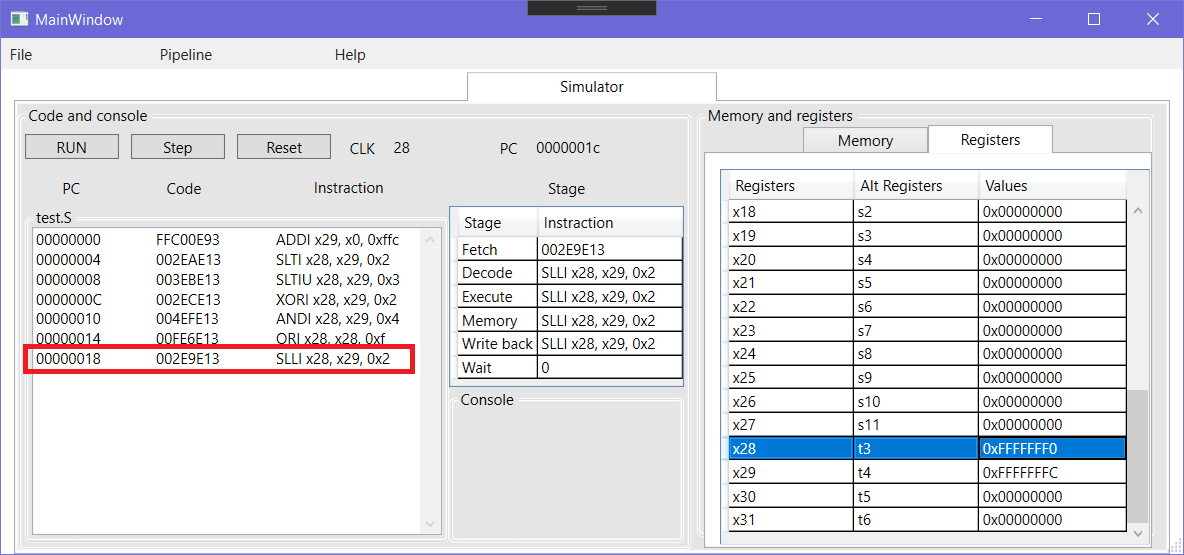


Рисунок 7.16 – Тестирование команды SLLI

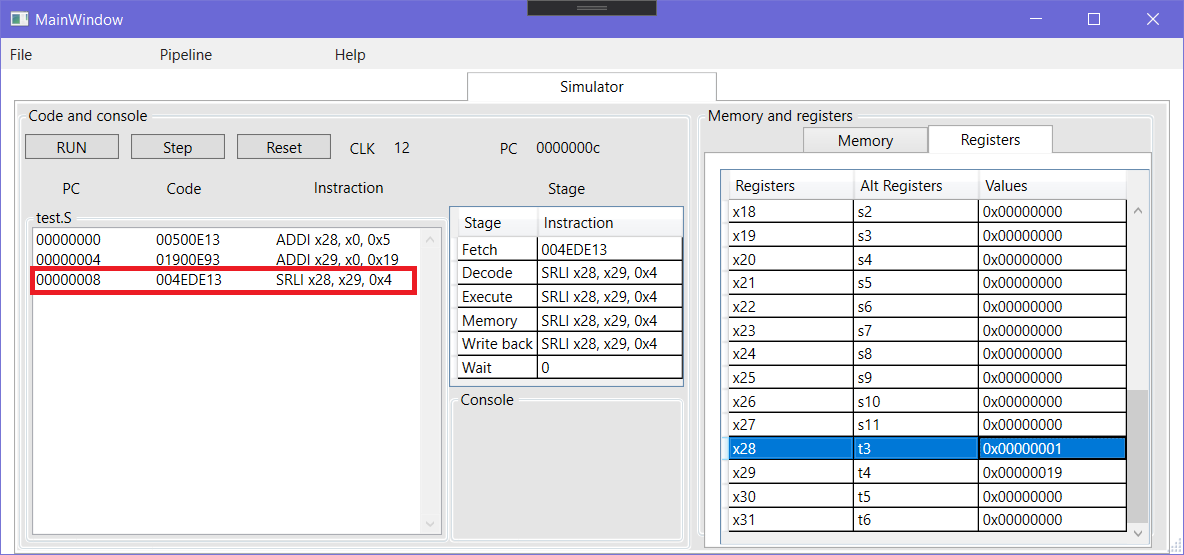


Рисунок 7.17 – Тестирование команды SRLI

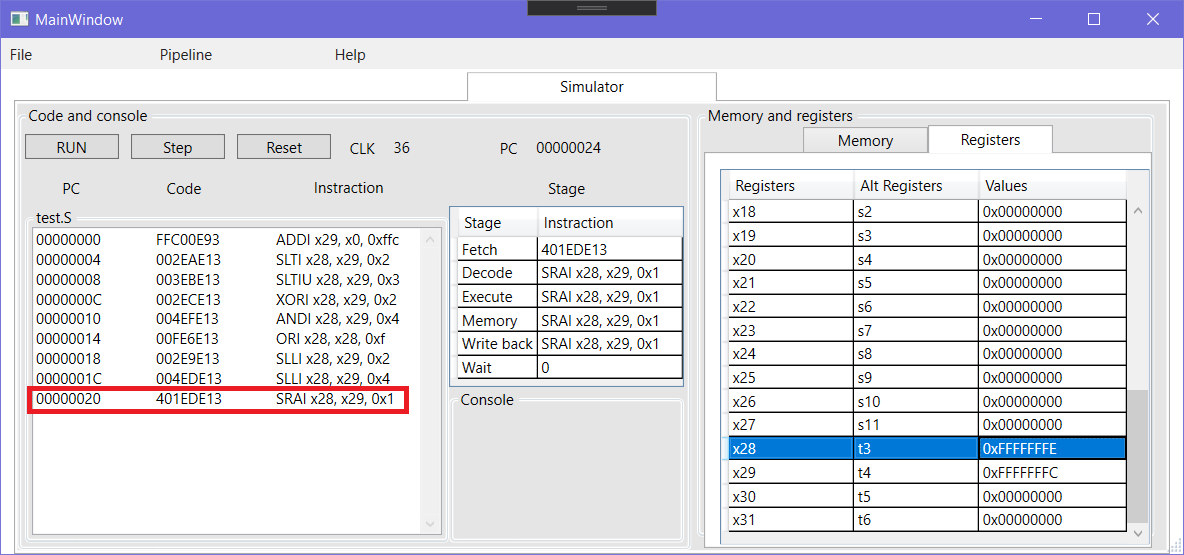


Рисунок 7.18 – Тестирование команды SRAI

**ADDI**

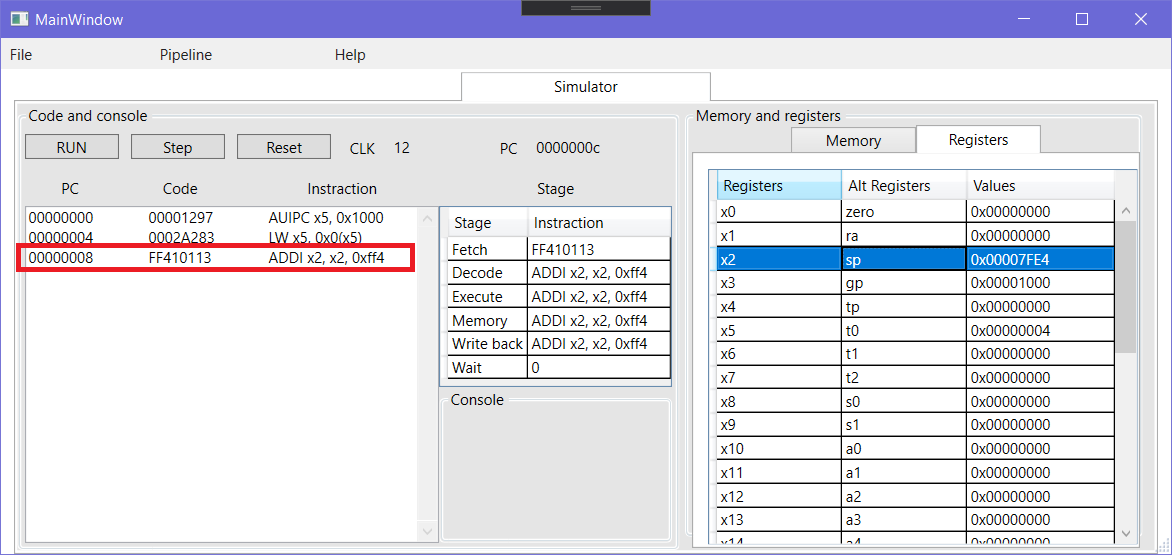


Рисунок 7.19 – Тестирование команды ADDI

Затем протестированы арифметические операции: SLT (Рисунок 7.20), SLTU (Рисунок 7.21), XOR (Рисунок 7.22), AND (Рисунок 7.23), OR (Рисунок 7.24), SLL (Рисунок 7.25), SRL (Рисунок 7.26), SRA (Рисунок 7.27), ADD (Рисунок 7.28), SUB (Рисунок 7.29).

**SLT**

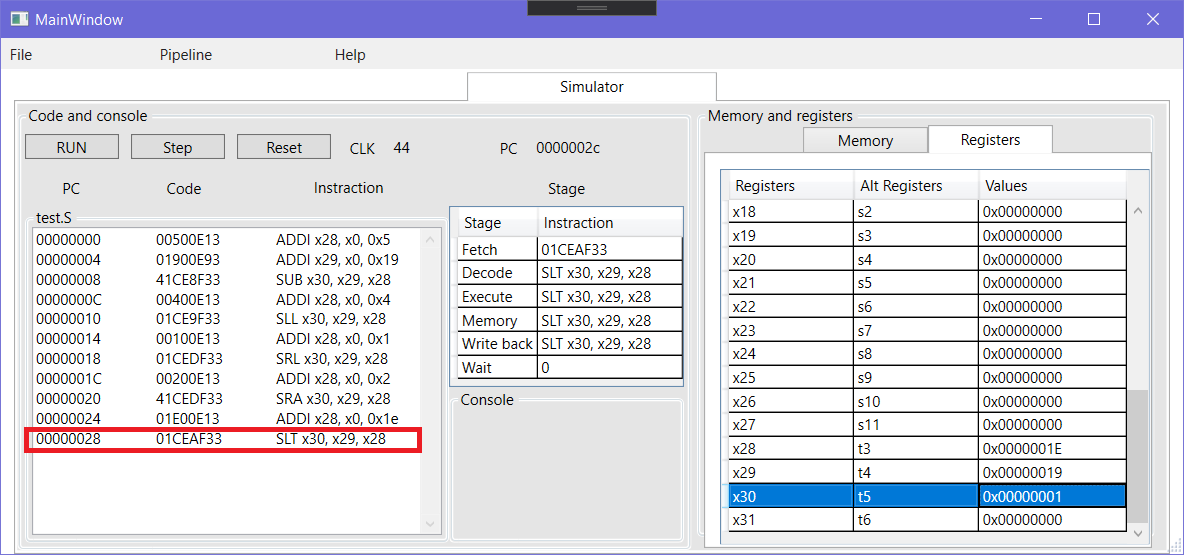


Рисунок 7.20 – Тестирование команды SLT

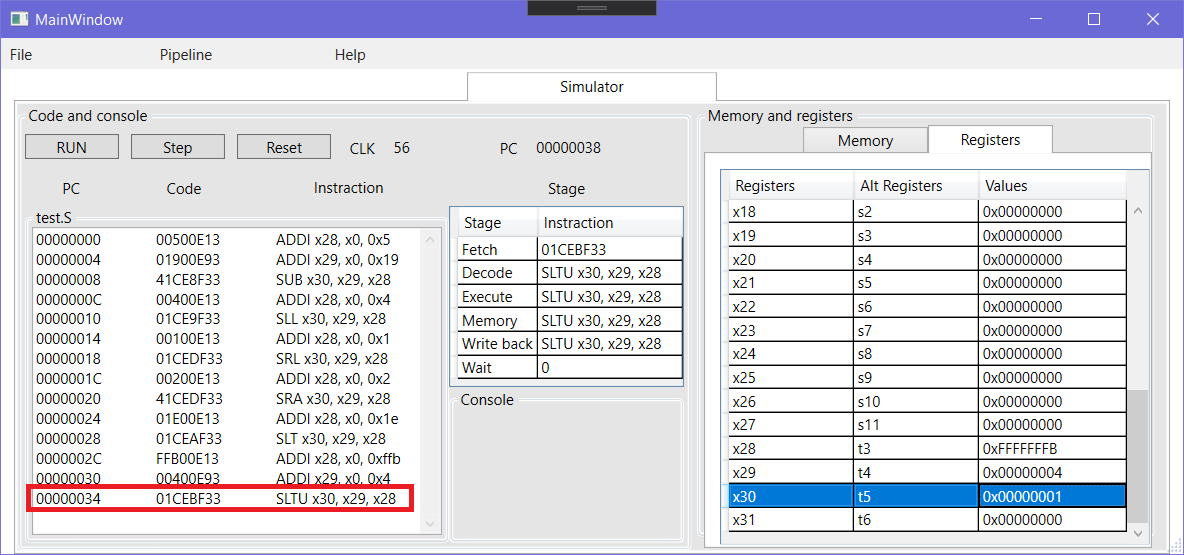


Рисунок 7.21 – Тестирование команды SLTU

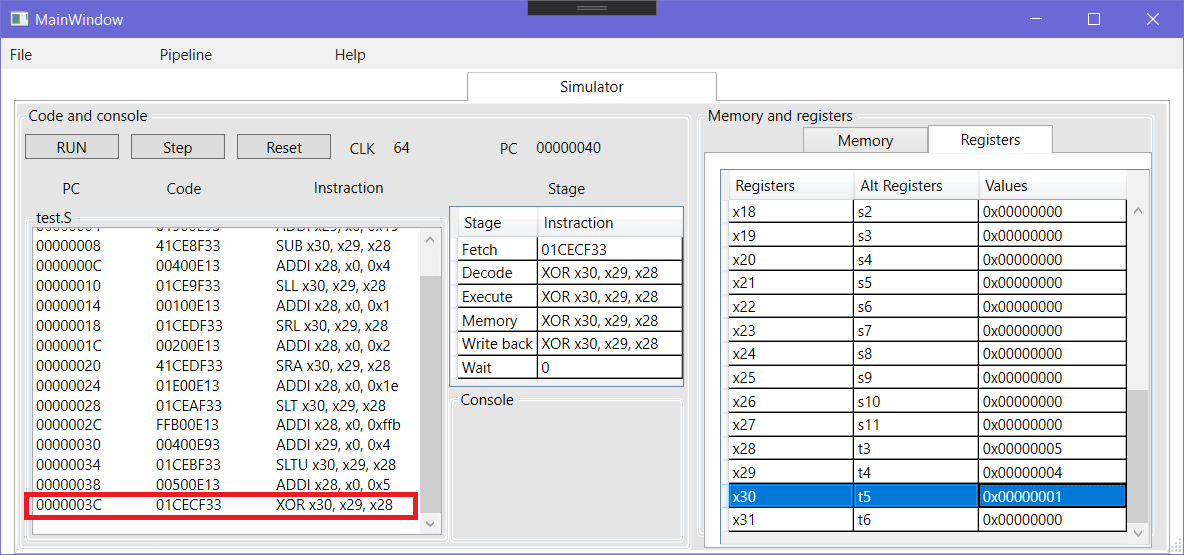


Рисунок 7.22 – Тестирование команды XOR

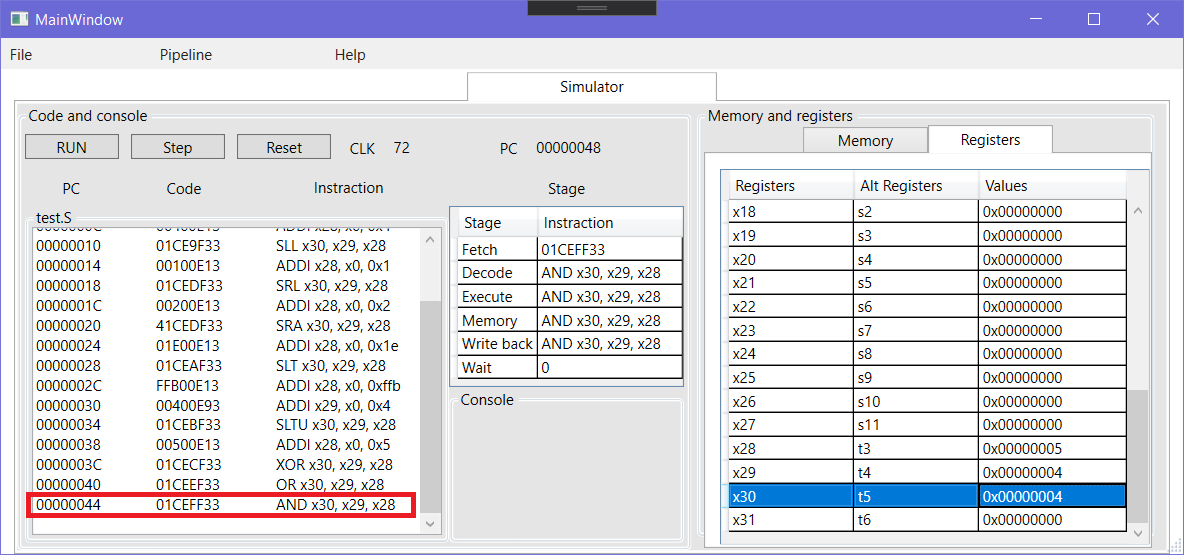


Рисунок 7.23 – Тестирование команды AND

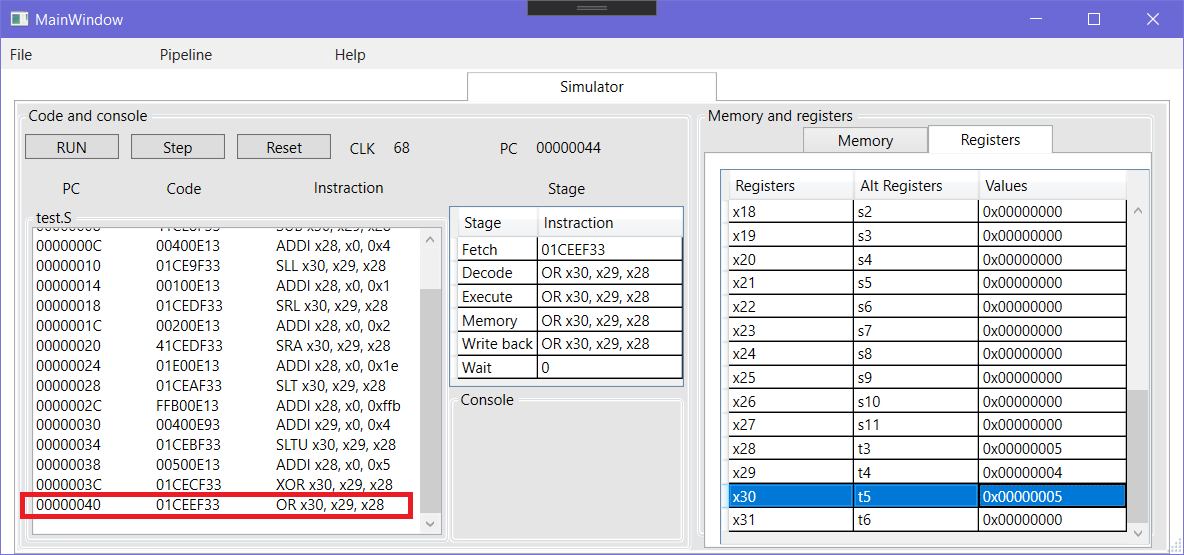


Рисунок 7.24 – Тестирование команды OR

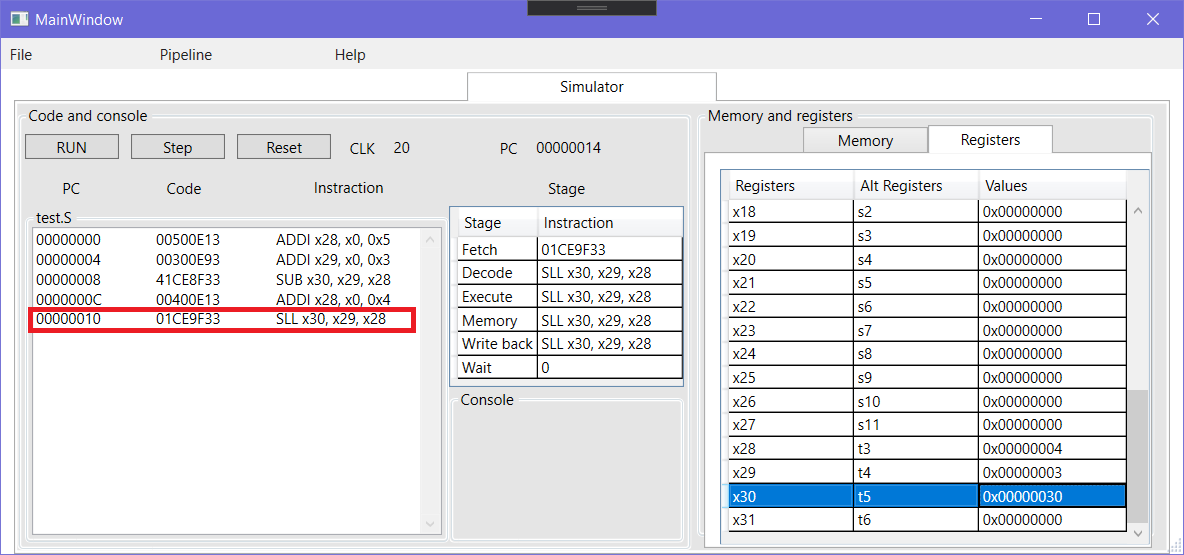


Рисунок 7.25 – Тестирование команды SLL

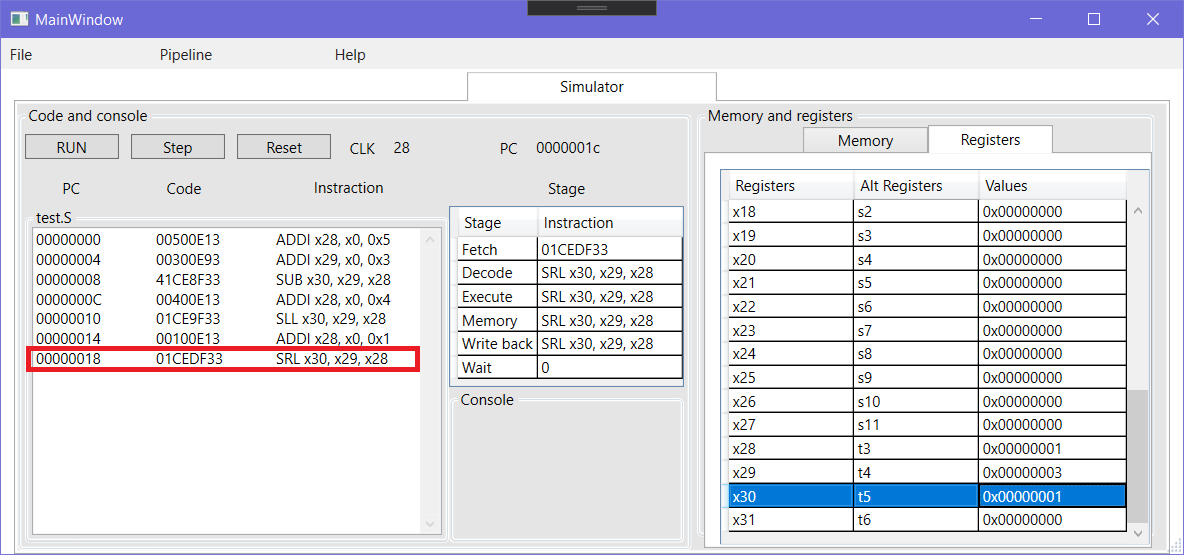


Рисунок 7.26 – Тестирование команды SRL

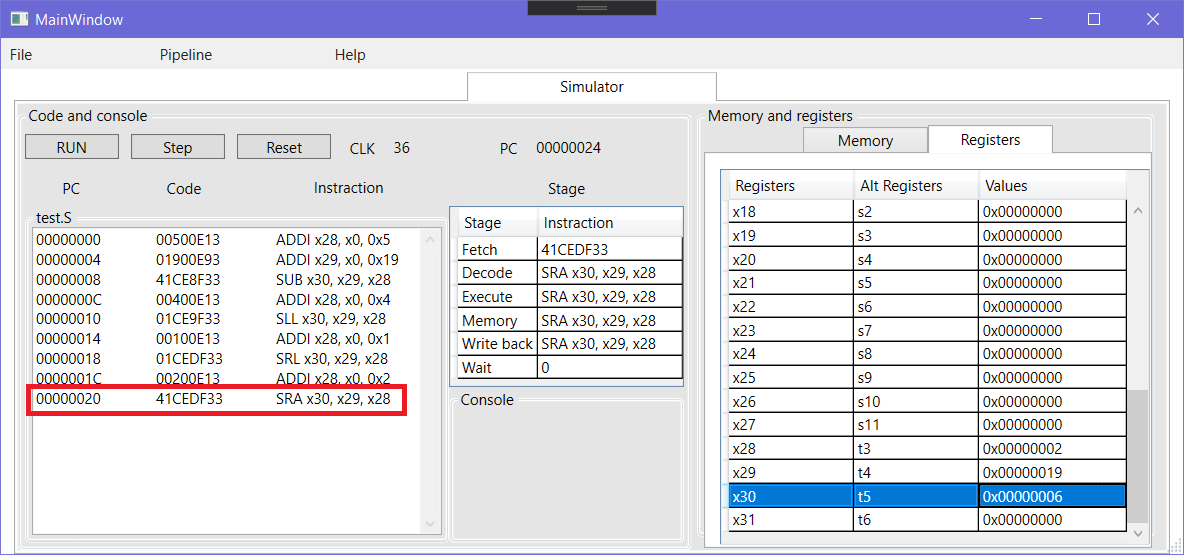


Рисунок 7.27 – Тестирование команды SRA

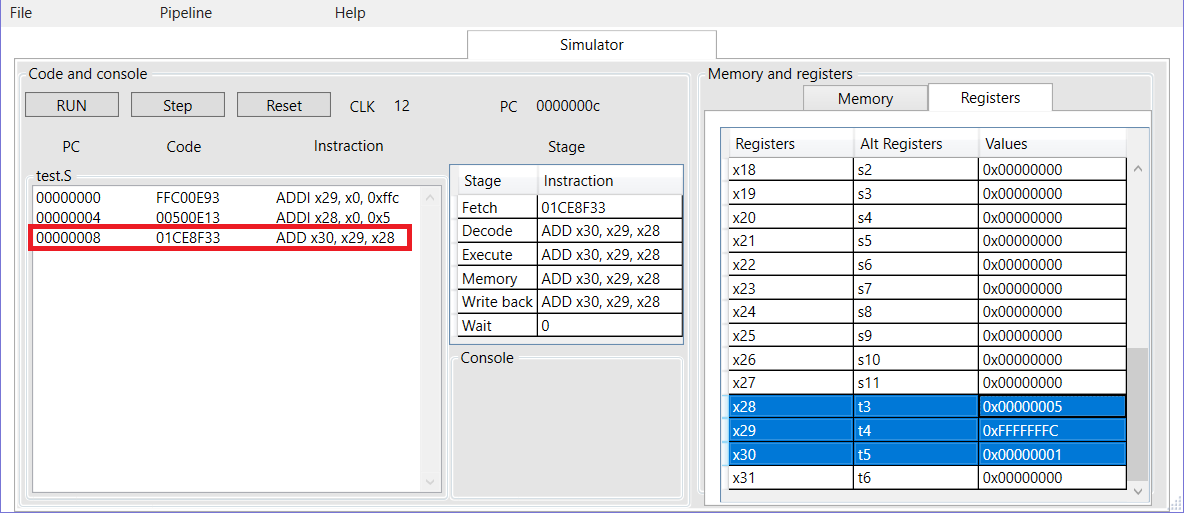


Рисунок 7.28 – Тестирование команды ADD

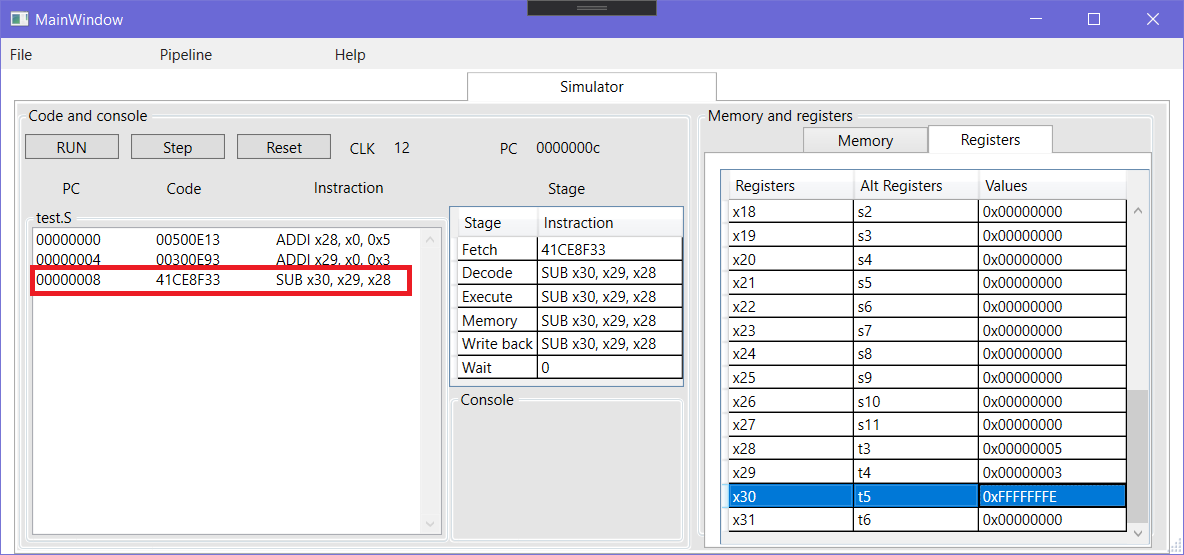


Рисунок 7.29 – Тестирование команды SUB

Затем протестированы команды безусловного перехода: JAL (Рисунок 7.30) и JALR (Рисунок 7.31), а также команды условного перехода: BGE (Рисунок 7.32), BGEU (Рисунок 7.33), BLTU (Рисунок 7.34), BLT (Рисунок 7.35), BEQ (Рисунок 7.36), BNE (Рисунок 7.37).

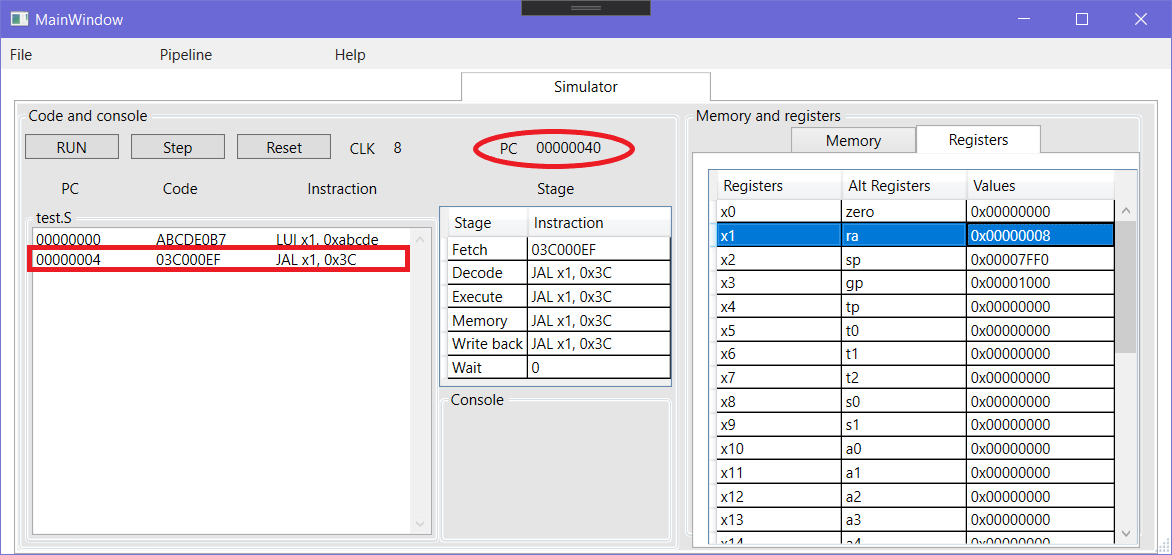


Рисунок 7.30 – Тестирование команды JAL

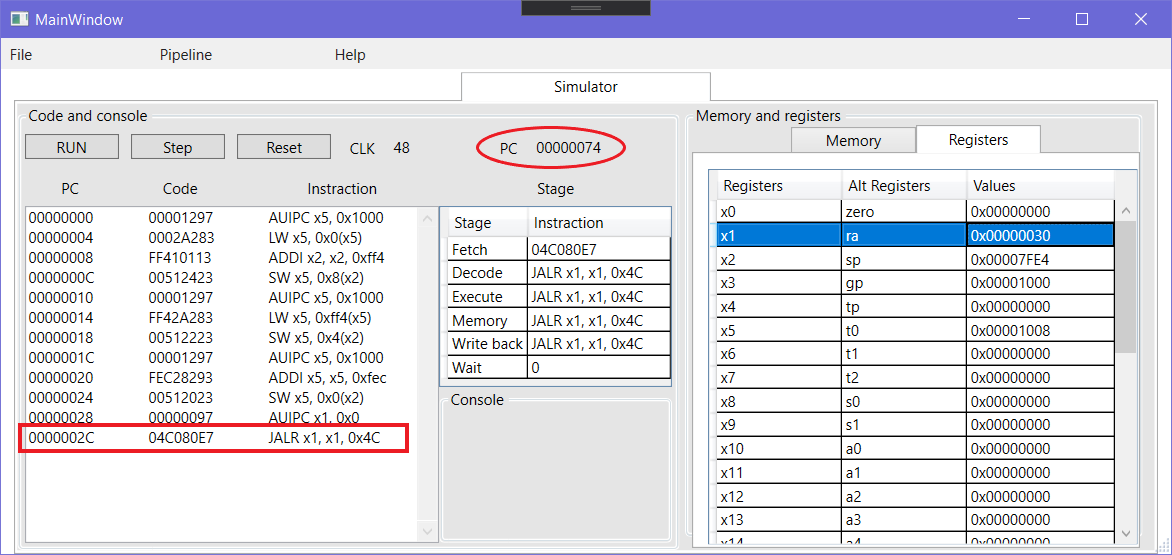


Рисунок 7.31 – Тестирование команды JALR

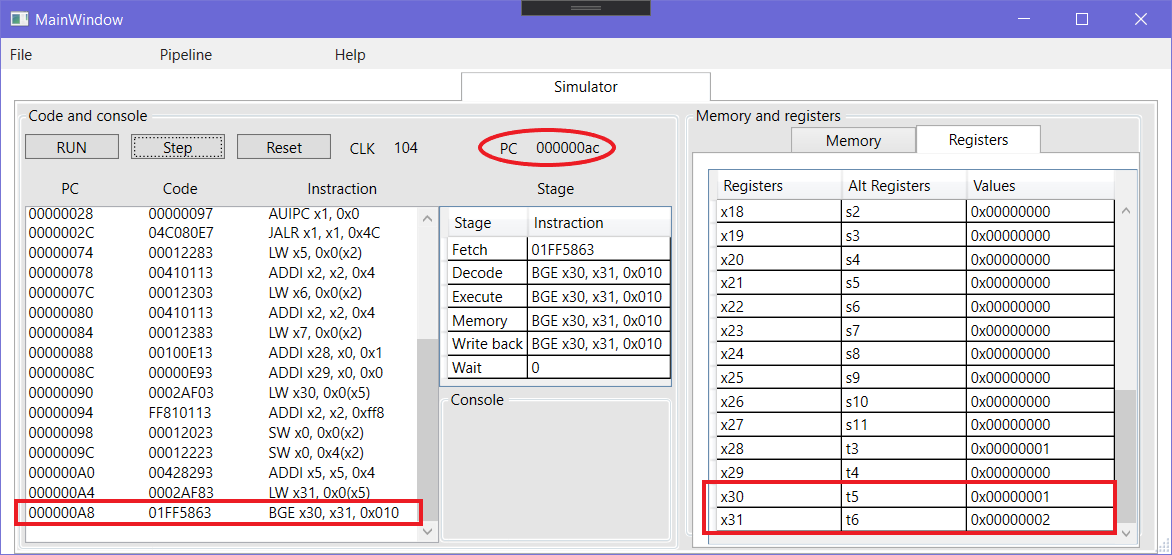
**B**

Рисунок 7.32 – Тестирование команды BGE

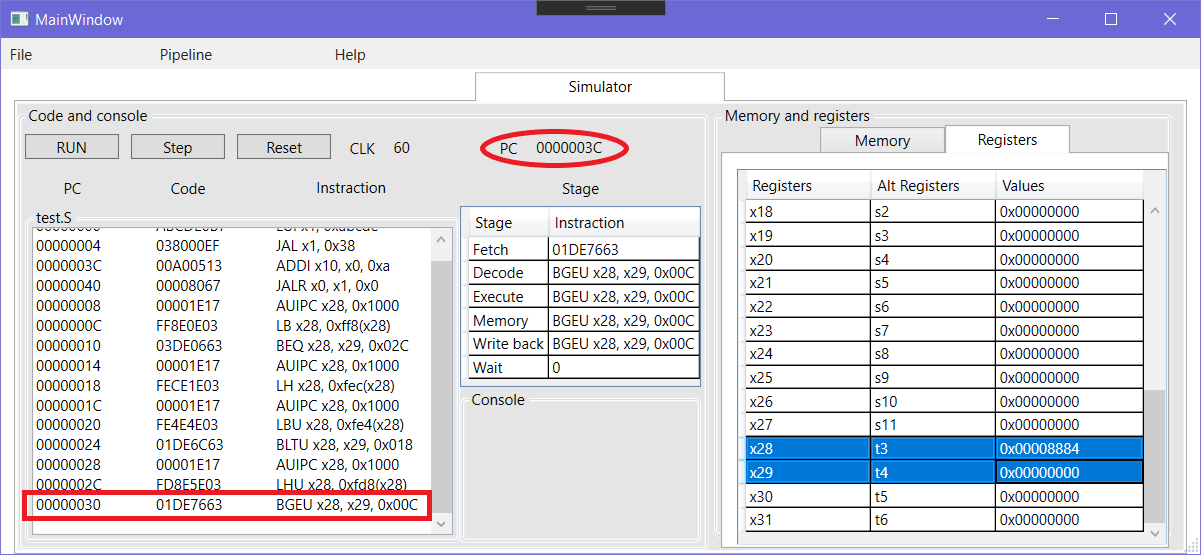


Рисунок 7.33 – Тестирование команды BGEU



Рисунок 7.34 – Тестирование команды BGEU

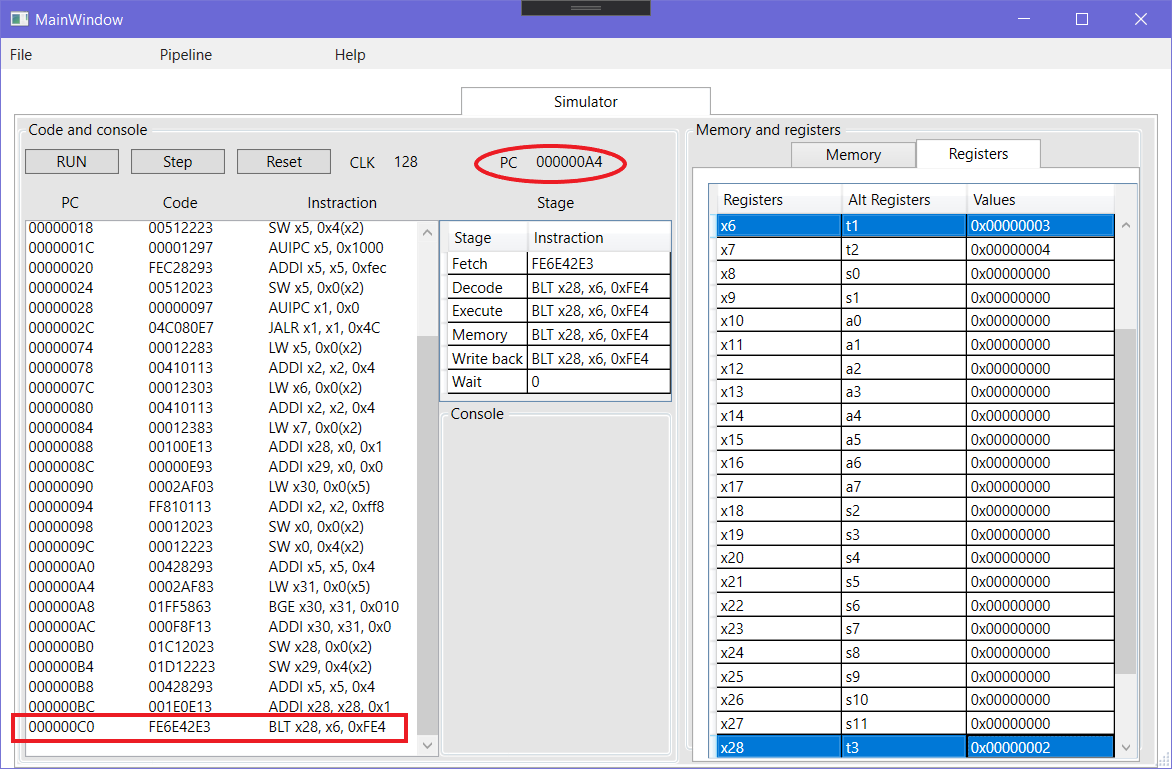


Рисунок 7.35 – Тестирование команды BLT

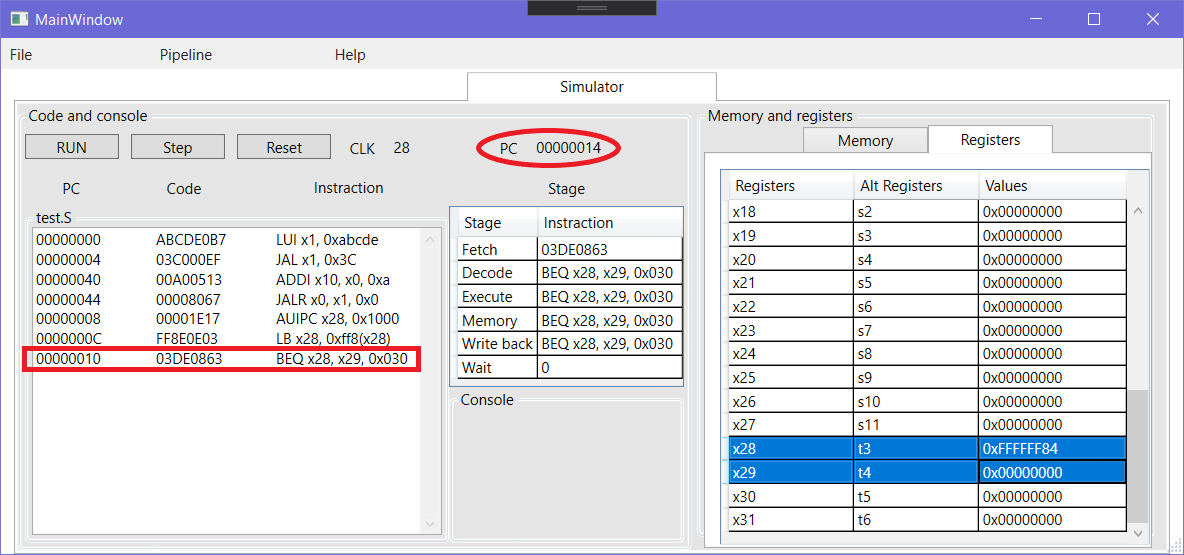


Рисунок 7.36 – Тестирование команды BEQ

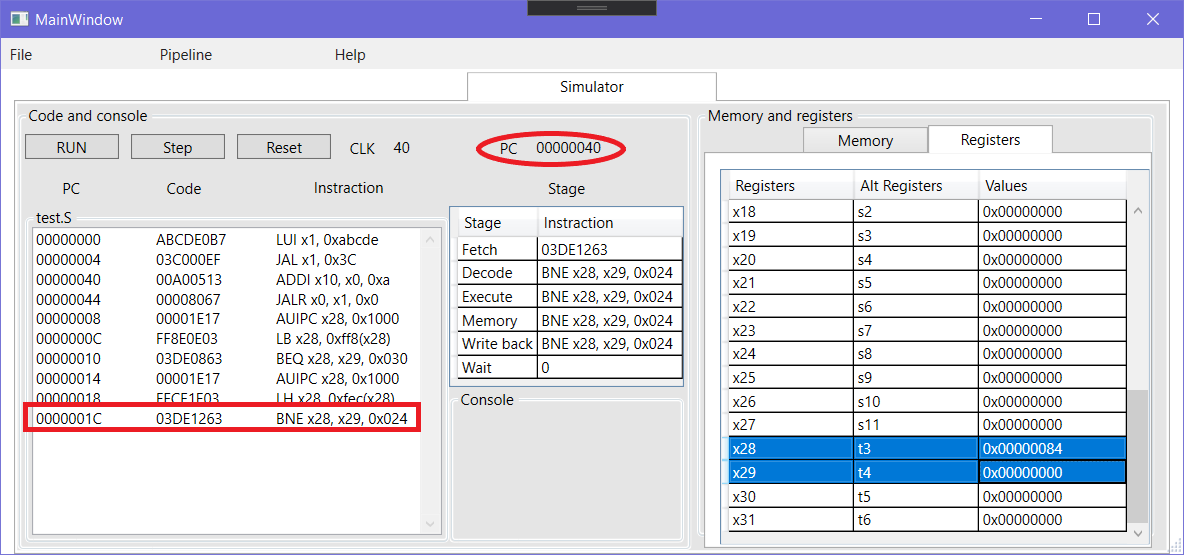


Рисунок 7.37 – Тестирование команды BNE

Далее протестирована команда ECALL с разными параметрами: x10 = 0xB (Рисунок 7.38), x10 = 0x1 (Рисунок 7.39), x10 = 0xA (Рисунок 7.40).



Рисунок 7.38 – Тестирование команды ECALL (x10 = 0xB)

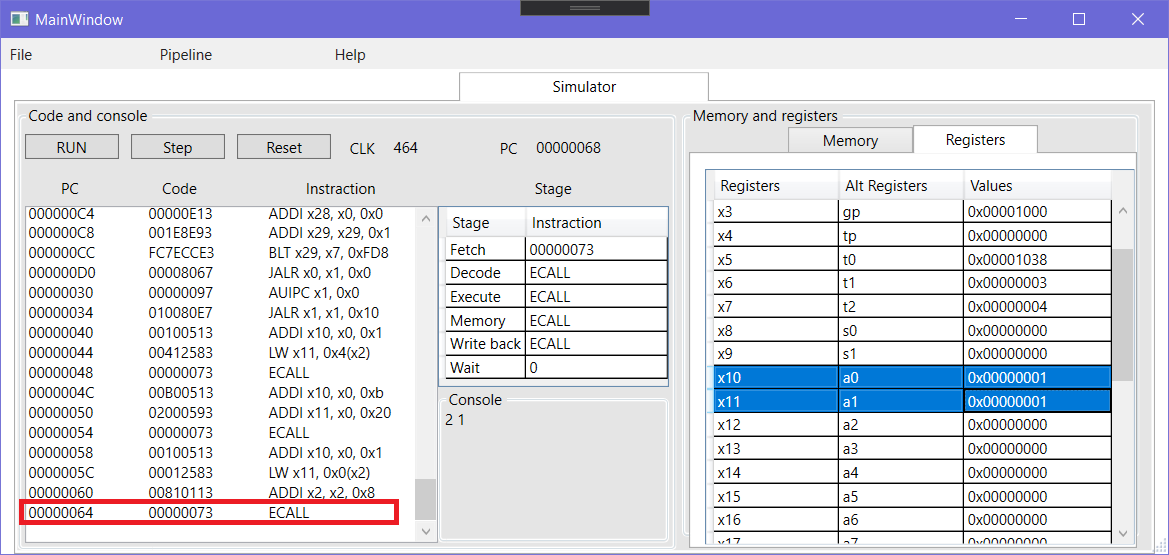


Рисунок 7.39 – Тестирование команды ECALL (x10 = 0x1)

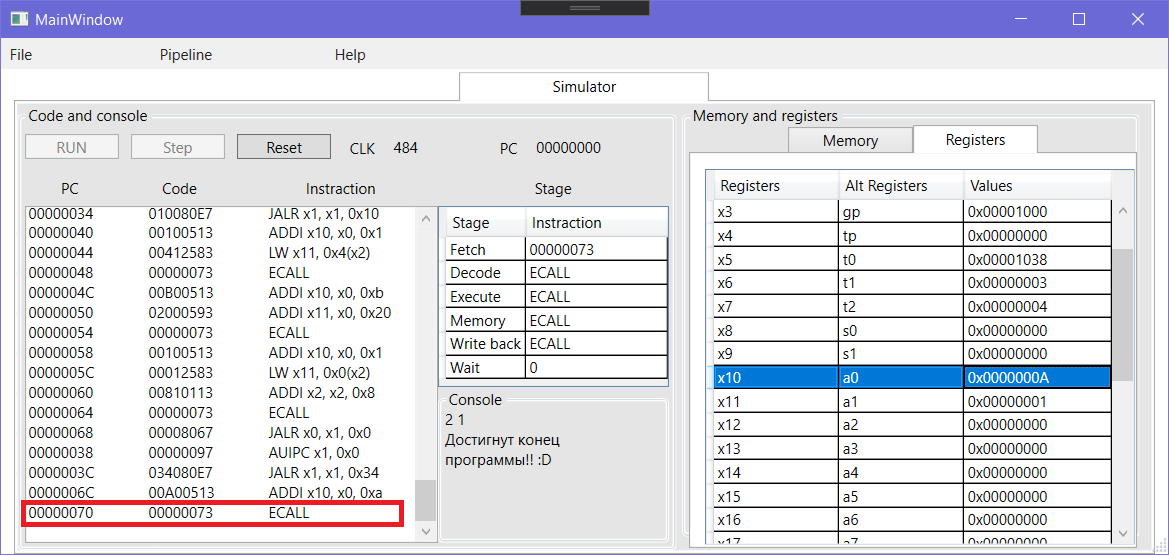


Рисунок 7.40 – Тестирование команды ECALL (x10 = 0xA)

#### 7.2 Тестирование режимов

В данном разделе протестированы режимы работы: без конвейера (Рисунок 7.41) и с 5-стадийным конвейером (Рисунок 7.42).

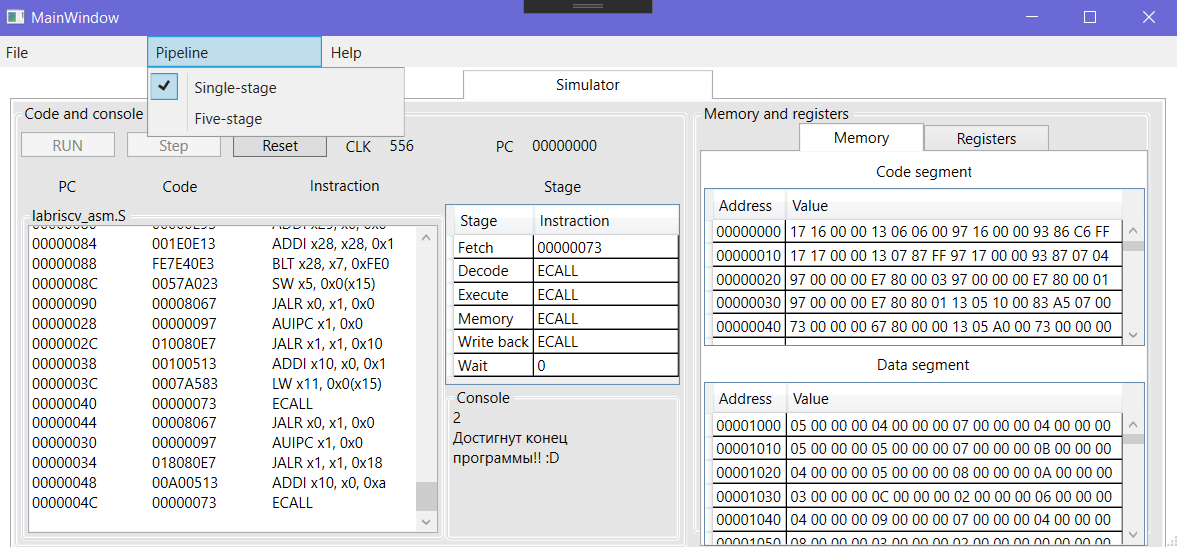


Рисунок 7.41 – Тестирование режима без конвейера

С 5-ти стадийным конвейером

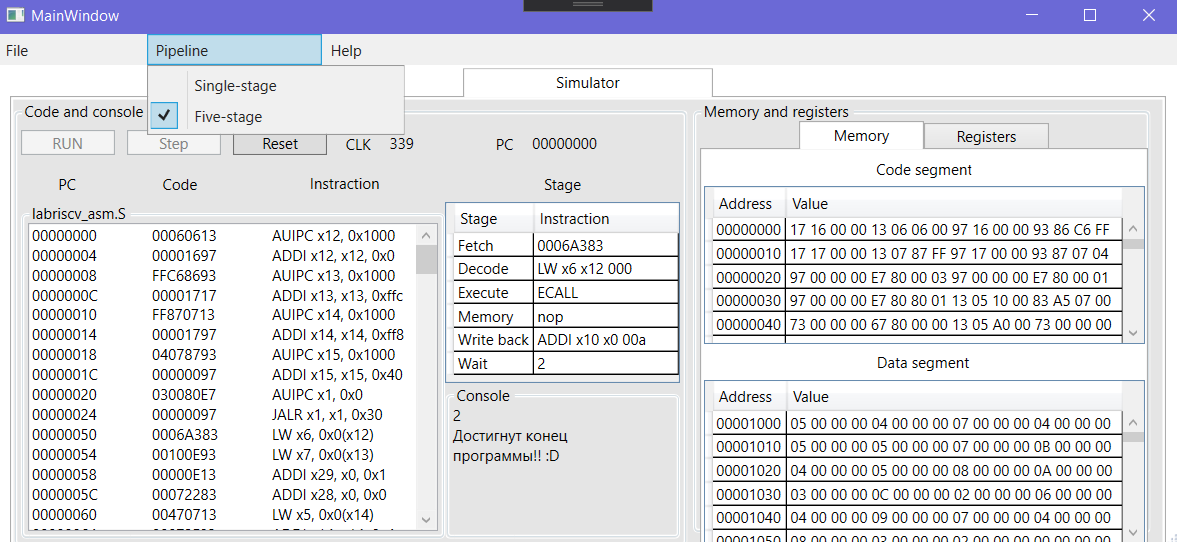


Рисунок 7.42 – Тестирование режима с 5-стадийным конвейером

### 8 Сравнение потенциальных реализаций

#### 8.1 Реализация без конвейера

В реализации без конвейера предположено, что одна инструкция выполняется приблизительно 4 такта. С учётом этого протестированы несколько программ (листинг представлен в приложении Д).

Результат первой программы представлен на рисунке 8.1. Программа выполняется за 556 условных тактов.

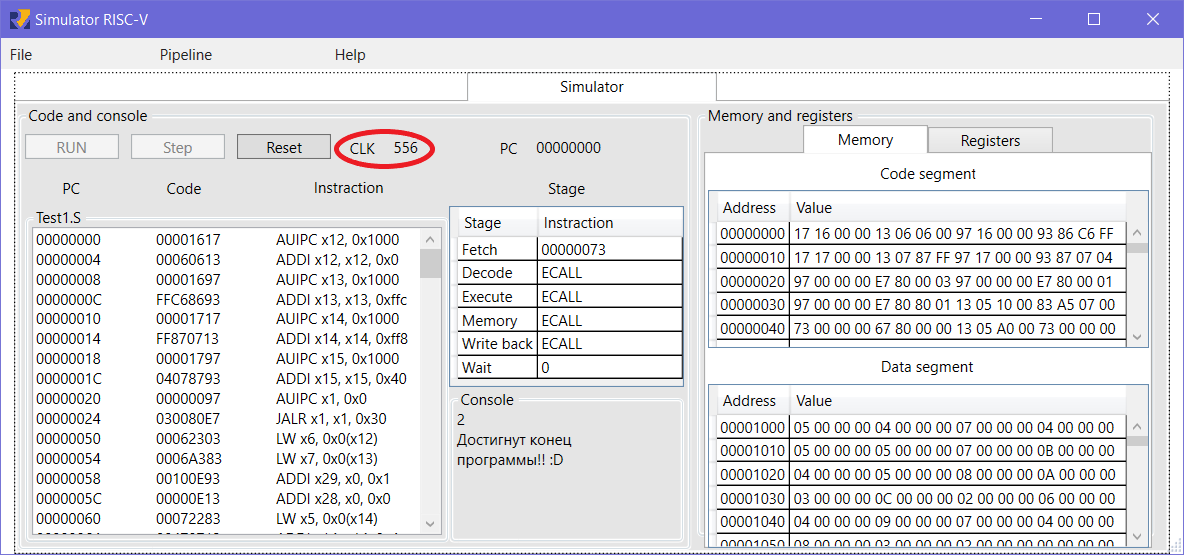


Рисунок 8.1 – Результаты программы Test1.S

Результат второй программы представлен на рисунке 8.2. Программа выполняется за 484 условных тактов.



Рисунок 8.2 – Результаты программы Test2.S

Результат третьей программы представлен на рисунке 8.3. Программа выполняется за 596 условных тактов.

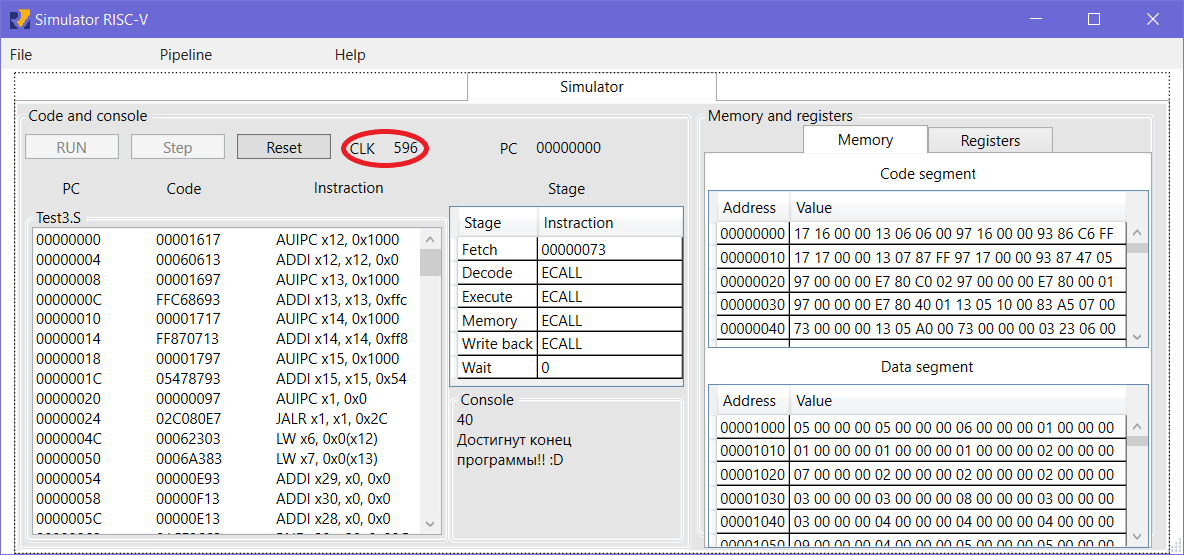


Рисунок 8.3 – Результаты программы Test3.S

Результат четвертой программы представлен на рисунке 8.4. Программа выполняется за 32 условных тактов.

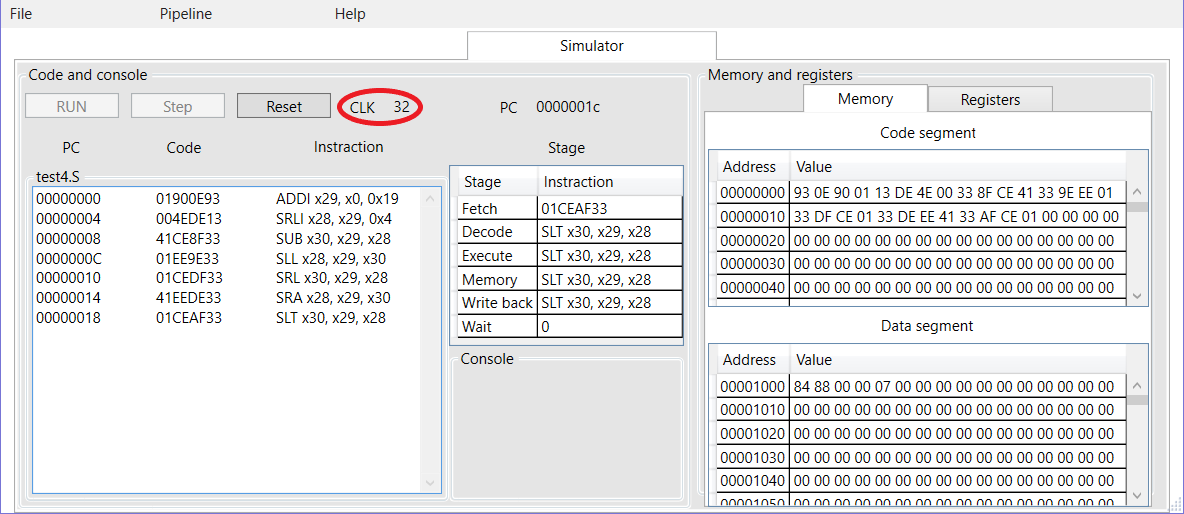


Рисунок 8.4 – Результаты программы test4.S

#### 8.2 Реализация с 5-стадийным конвейером

В реализации с конвейером одна инструкция обрабатывается на одной стадии за 1 такт. При возникновении конфликта по данным конвейер ждёт завершение выполнения команды, с которой возник конфликт. В зависимости от того на какой стадии – Write back или Memory – находится команда, конвейер ждёт, соответственно, 1 такт или 2 такта. После этого конвейер продолжает свою работу.

С учётом этого протестированы такие же программы, как и в первой реализации для дальнейшего сравнения реализаций.

Результат первой программы представлен на рисунке 8.5. Программа выполняется за 339 условных тактов.

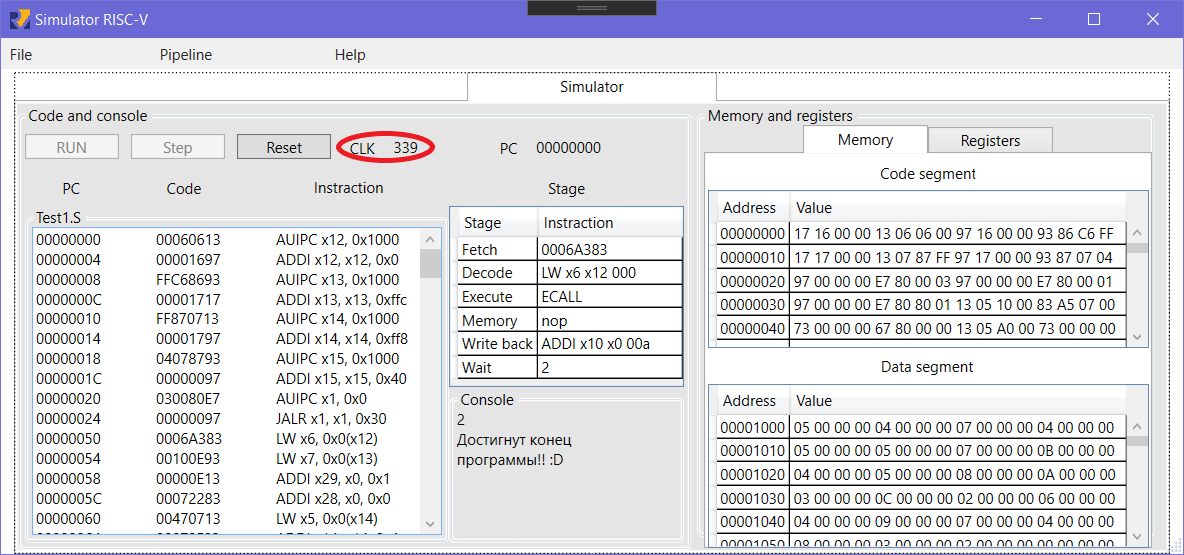


Рисунок 8.5 – Результаты программы Test1.S с конвейером

Результат второй программы представлен на рисунке 8.6. Программа выполняется за 270 условных тактов.

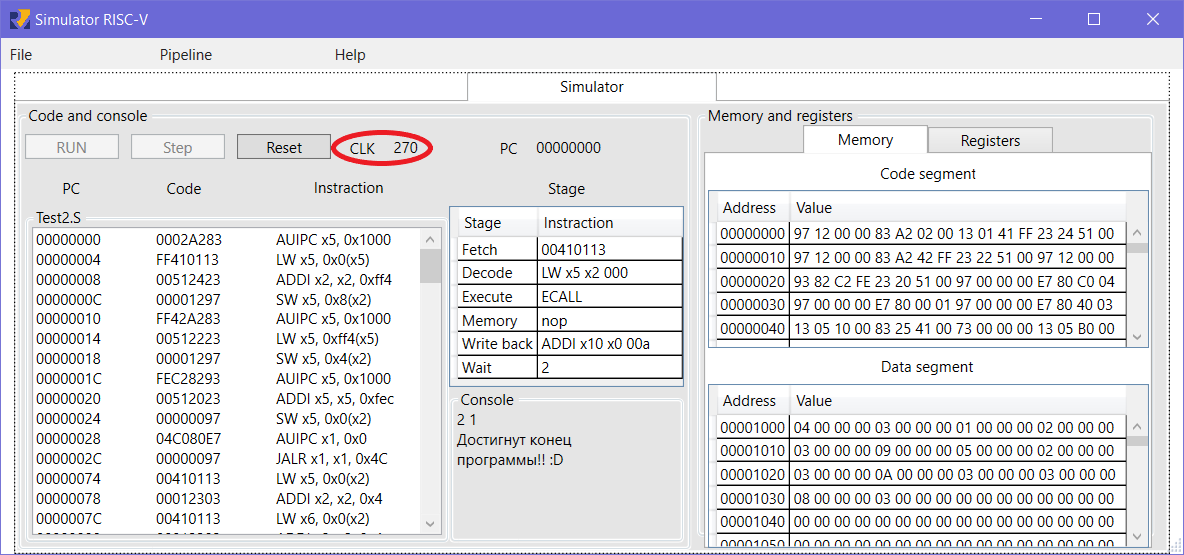


Рисунок 8.6 – Результаты программы Test2.S с конвейером

Результат третьей программы представлен на рисунке 8.7. Программа выполняется за 410 условных тактов.

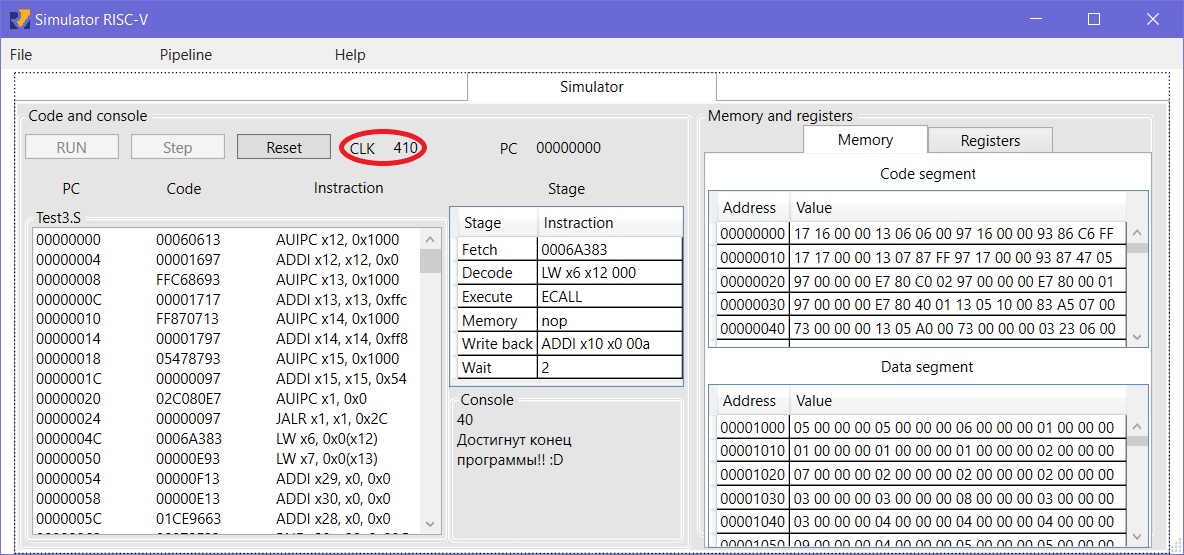


Рисунок 8.7 – Результаты программы Test3.S с конвейером

Результат четвертой программы представлен на рисунке 8.8. Программа выполняется за 24 условных тактов.

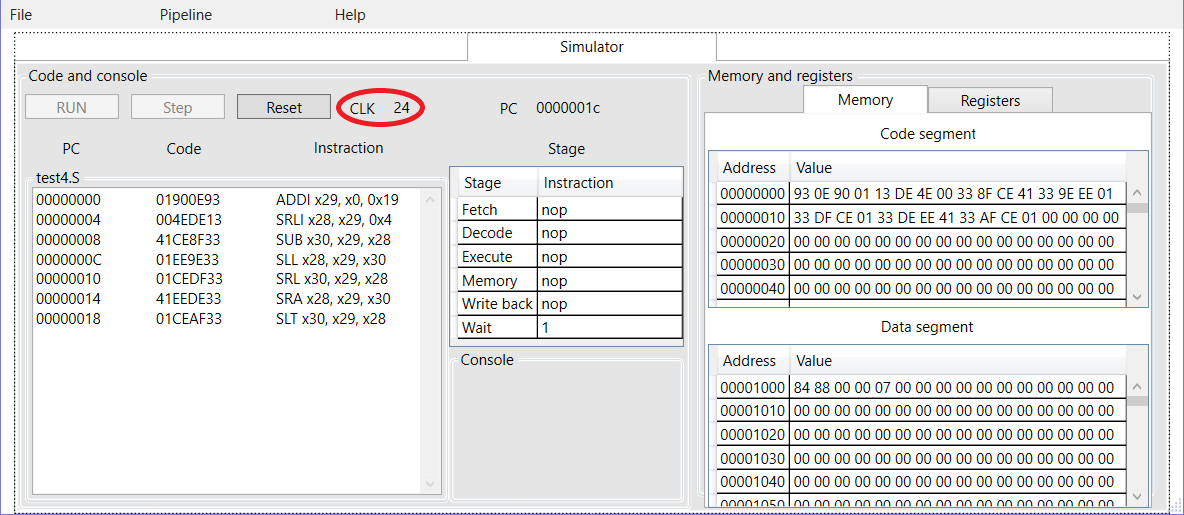


Рисунок 8.8 – Результаты программы test4.S с конвейером

#### 8.3 Сравнение реализаций

Для удобства сравнения реализаций результаты программ занесены в таблицу 8.1.

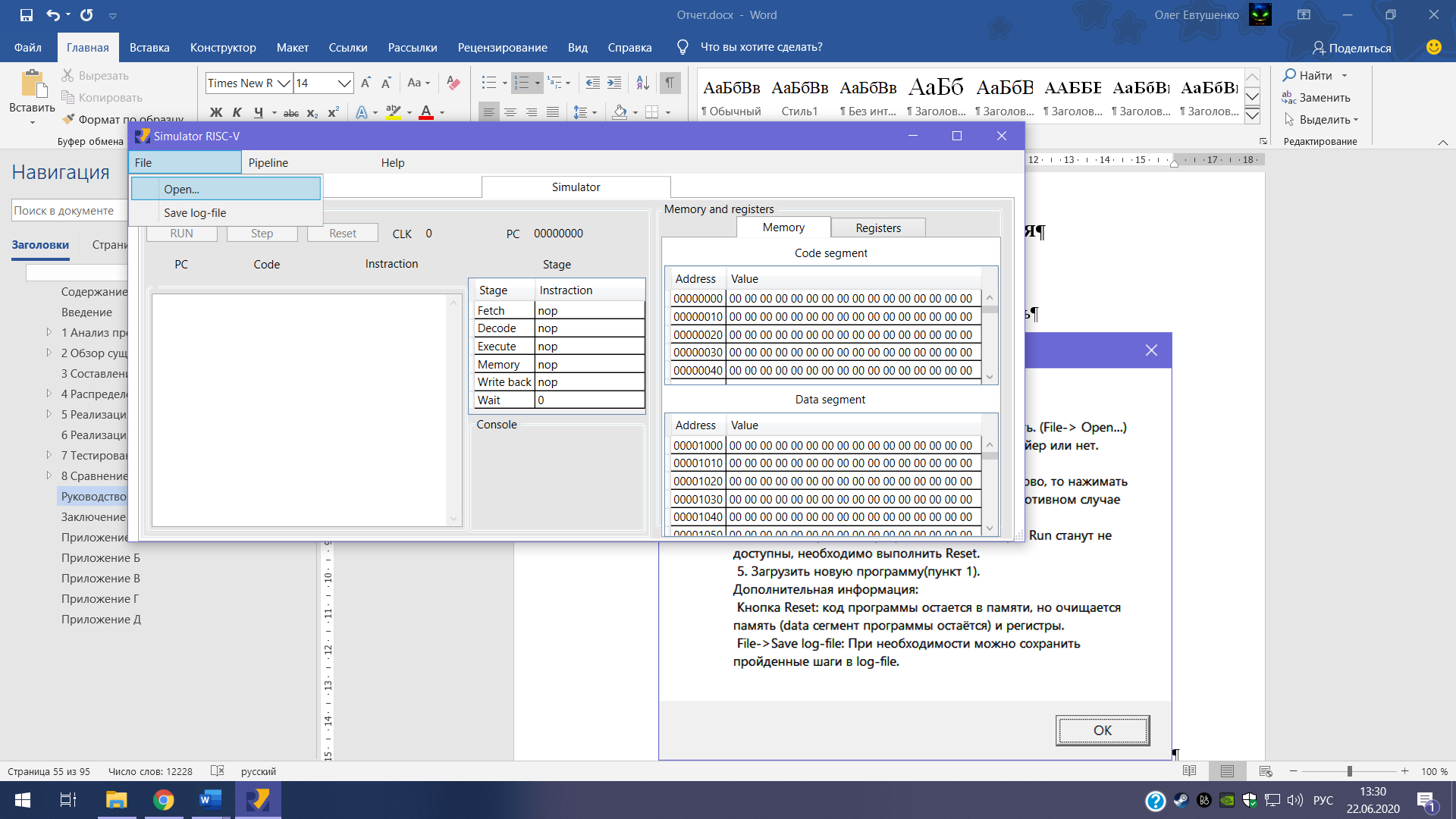
Таблица 8.1 – Результаты тестирования программ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название программы | CLK без конвейера | CLK с конвейером | Отношение результатов |
| 1 | Test1.S | 556 | 339 | 1,64 |
| 2 | Test2.S | 484 | 270 | 1,79 |
| 3 | Test3.S | 596 | 410 | 1,45 |
| 4 | test4.S | 32 | 24 | 1,33 |

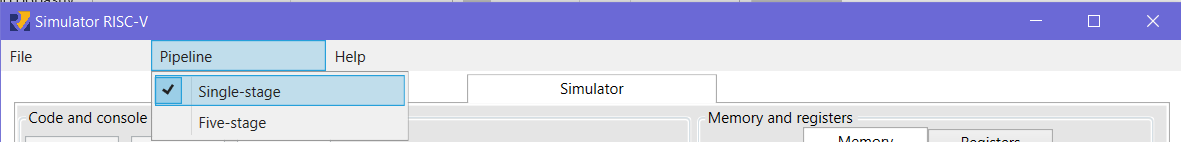
Исходя из результатов можно сделать вывод, что при использовании 5-стадийного конвейера скорость выполнения увеличивается в среднем приблизительно в 1,55 раз. Это достигается за счёт того, что при использовании конвейера, инструкции обрабатываются параллельно на 5 стадиях.

### 9 Руководство пользователя

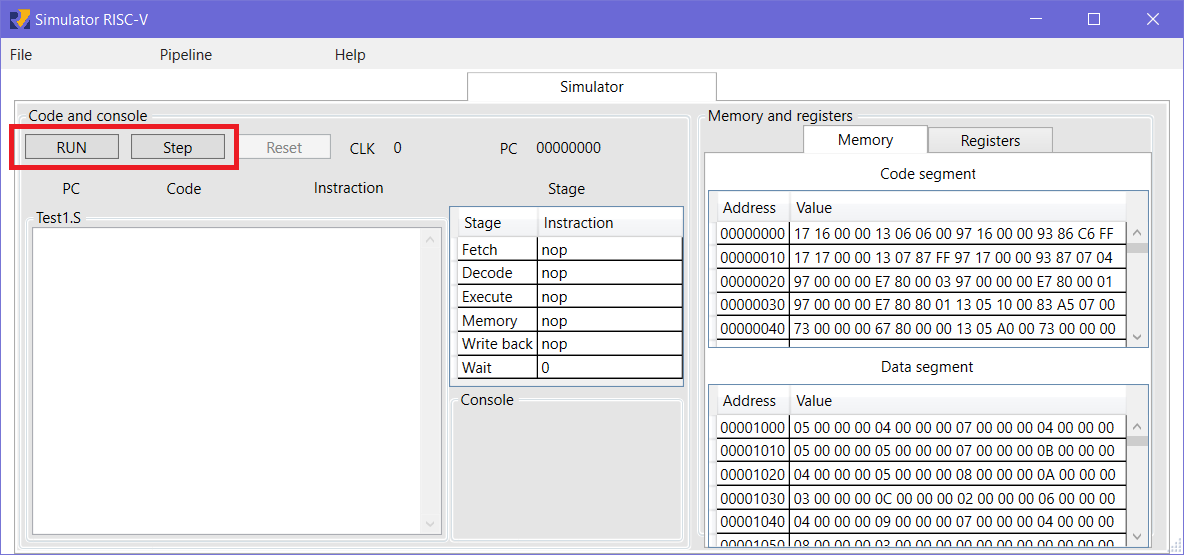
1. Выбрать файл формата «**.S**» для записи в память.



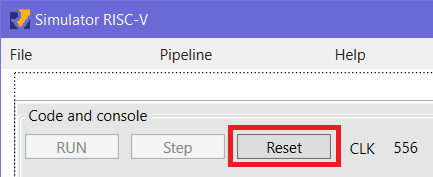
1. Выбрать режим работы с 5-стадийным конвейером или нет.



1. Если необходимо выполнить программу пошагово, то нажимать на кнопку **Step** до завершения программы, в противном случае нажать на кнопку **Run**.



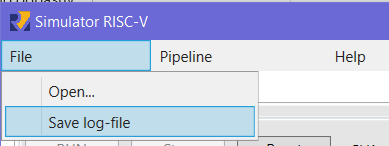
1. После завершения программы кнопки Step и Run станут недоступны, необходимо нажать на кнопку **Reset**.



1. Для загрузки новой программы вернутся в пункт 1, для выполнения текущей вернутся в пункт 2.

Кнопка Reset: код программы остается в памяти, но очищается память с данными (изначальные данные с программой остаются) и регистры.

Если необходимо, то можно сохранить пройденные шаги.



### Заключение

В ходе работы выполнен анализ, разработка и сравнение потенциальных реализаций, а также реализация и декодирование команд из стандартного набора инструкций RISC-V (RV32I Base Instruction Set), структуры памяти и работы с ней.

В результате сравнения реализаций можно сделать вывод, что реализация с 5-стадийным конвейером является более производительнее.

### Список литературы

1. Елизаров Г.С., Корнеев В.В., Тарасов И.Е., Советов П.Н. Основные тенденции развития архитектур специализированных многоядерных процессоров. ОБЗОР // Изв. вузов. Электроника. – 2018. – Т. 23. – № 2. – С. 161–172.
2. Andrew Waterman1, Krste Asanovi ́. The RISC-V Instruction Set Manual Volume I: Unprivileged ISA //SiFive Inc. CS Division, EECS Department, University of California, Berkeley. 2019.
3. Venus GitHub [Электронный ресурс]. Режим доступа <https://github.com/kvakil/venus> (15.04.2020).
4. Spike GitHub [Электронный ресурс]. Режим доступа <https://github.com/riscv/riscv-isa-sim> (25.04.2020).
5. Hennessy, John L.; Patterson, David A. (2011). Computer Architecture, A Quantitative Approach (5th ed.). Morgan Kaufmann. ISBN 978-0123838728.
6. Шилдт Г. Java. Полное руководство // Диалектика, 2019 – 1488 c.
7. Прата Стивен. Язык программирования C++. Лекции и упражнения // Диалектика, 2018 – 1244 c.
8. Васильев А. Н. Программирование на C# для начинающих. Основные сведения // Эксмо, 2018 – 592 c.
9. Бэрри Пол. Изучаем программирование на Python // Эксмо, 2017 – 624 c.
10. Krste Asanović David A. Patterson. Instruction Sets Should Be Free: The Case For RISC-V //Electrical Engineering and Computer Sciences University of California at Berkeley: UCB/EECS-2014-146, 2014.
11. П. Н. Советов. Итеративный подход с использованием компилятора для синтеза и моделирования проблемно-ориентированного набора команд //International Journal of Open Information Technologies ISSN: 2307-8162 vol. 7, no.10, 2019.
12. Computer Organization and Design: The Hardware/Software Interface, 5th Edition, David A. Patterson, John L. Hennessy. Morgan Kaufmann, 2013. — 800 p.
13. Linker Scripts [Электронный ресурс]. Режим доступа: <https://sourceware.org/binutils/docs/ld/Scripts.html> (25.05.2020).

### Приложение А

**ЛИСТИНГ Decoder.cs**

using System;

using System.IO;

namespace Simulator\_RISCV

{

class Decoder

{

class AsmInstruction //дешифрованная инструкция

{

public string Inst { get; set; } = "";

public string Op1 { get; set; } = "";

public string Op2 { get; set; } = "";

public string Op3 { get; set; } = "";

public string Full\_Comand { get; set; } = "";

public string Asm\_Comand { get; set; } = "";

public void Clear\_Inst()

{

Inst = "";

Full\_Comand = "";

Asm\_Comand = "";

Op1 = "";

Op2 = "";

Op3 = "";

}

}

class DeshInstruction //класс для определения команды

{

public string Opcode { get; set; } = "";

public string Funct3 { get; set; } = "";

public string Funct7 { get; set; } = "";

}

public int DecodeInstruction(string Instruction, out string Full\_Comand, out string Asm\_Comand)

{

DeshInstruction DeshInst = new DeshInstruction();

AsmInstruction AsmInst = new AsmInstruction();

string s;

//отделяем "0х" от кода команды, переводим в 2сс из 16сс

Instruction = Convert.ToString(Convert.ToInt32(Instruction.Trim('0', 'x').ToUpper(), 16), 2).PadLeft(32, '0');

DeshInst.Opcode = Instruction.Substring(25);

DeshInst.Funct3 = Instruction.Substring(17, 3);

DeshInst.Funct7 = Instruction.Substring(0, 7);

AsmInst.Clear\_Inst();//очищаем поля от предыдущей инструкции

string[] inst;

StreamReader ReadInstName = new StreamReader("Comand\_base.txt");

while ((s = ReadInstName.ReadLine()) != null)//определяем команду исходя из расшифрованных Opcode и Funct3

{

inst = s.Split('\t');

if (inst[1] == DeshInst.Opcode)

{

if (inst.Length >= 3)

{

if (inst.Length == 3)

{

if (inst[2] == DeshInst.Funct3)

{

AsmInst.Inst = inst[0];

break;

}

}

else

{

if (inst[3] == DeshInst.Funct7)

{

AsmInst.Inst = inst[0];

break;

}

}

}

else

{

AsmInst.Inst = inst[0];

break;

}

}

}

ReadInstName.Close(); // закрываем файл для освобождения памяти

switch (AsmInst.Inst)

{

case "LUI":

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(20, 5), 2), 10);//RD

AsmInst.Op2 = Convert.ToString(Convert.ToInt32(Instruction.Substring(0, 20), 2), 16);

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", 0x" + AsmInst.Op2;

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2;

break;

}

case "AUIPC":

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(20, 5), 2), 10);//выделение регистра Rd

AsmInst.Op2 = Instruction.Substring(0, 20).PadRight(32, '0');

AsmInst.Op2 = Convert.ToString(Convert.ToInt32(AsmInst.Op2, 2), 16);//imm[0:19]

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", 0x" + AsmInst.Op2;

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2;

break;

}

case "JAL":

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(20, 5), 2), 10);

AsmInst.Op2 = Instruction.Substring(0, 20);

char[] str1 = AsmInst.Op2.Substring(10, 8).ToCharArray();

char[] str2 = AsmInst.Op2.Substring(1, 10).ToCharArray();

Array.Reverse(str1);

Array.Reverse(str2);

string out2 = new string(str2);

string out1 = new string(str1);

AsmInst.Op2 = Instruction.Substring(0, 1) + out2 + Instruction.Substring(10, 1) + out1;

str1 = AsmInst.Op2.ToCharArray();

Array.Reverse(str1);

AsmInst.Op2 = new string(str1);

AsmInst.Op2 = Convert.ToString(Convert.ToInt32(AsmInst.Op2, 2), 16).ToUpper();

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", 0x" + AsmInst.Op2;

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2;

break;

}

case "JALR":

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(20, 5), 2), 10);

AsmInst.Op2 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(12, 5), 2), 10);

AsmInst.Op3 = Convert.ToString(Convert.ToInt32(Instruction.Substring(0, 12), 2), 16).ToUpper();

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", " + AsmInst.Op2 + ", 0x" + AsmInst.Op3;

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2 + " " + AsmInst.Op3.PadLeft(8, '0');

break;

}

case "ECALL":

{

if (Instruction.Substring(0, 12) == "000000000000")

{

AsmInst.Full\_Comand = AsmInst.Inst;

AsmInst.Asm\_Comand = AsmInst.Inst;

}

break;

}

case "EBREAK":

{

if (Instruction.Substring(0, 12) == "000000000001")

{

AsmInst.Full\_Comand = AsmInst.Inst;

AsmInst.Asm\_Comand = AsmInst.Inst;

}

break;

}

}

if ((AsmInst.Inst == "BEQ" || AsmInst.Inst == "BNE" || AsmInst.Inst == "BLT" || AsmInst.Inst == "BGE" ||

AsmInst.Inst == "BLTU" || AsmInst.Inst == "BGEU") && AsmInst.Full\_Comand == "")

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(12, 5), 2), 10);

AsmInst.Op2 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(7, 5), 2), 10);

AsmInst.Op3 = Instruction.Substring(0, 1) + Instruction.Substring(24, 1) + Instruction.Substring(1, 6) + Instruction.Substring(20, 4);

AsmInst.Op3 = (Convert.ToInt32(AsmInst.Op3, 2) << 1).ToString("X");

if (AsmInst.Op3.Length > 3)

AsmInst.Op3 = AsmInst.Op3.Substring(1, 3);

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", " + AsmInst.Op2 + ", 0x" + AsmInst.Op3.PadLeft(3, '0');

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2 + " " + AsmInst.Op3.PadLeft(3, '0');

}

if ((AsmInst.Inst == "LB" || AsmInst.Inst == "LH" || AsmInst.Inst == "LW" || AsmInst.Inst == "LBU" ||

AsmInst.Inst == "LHU") && AsmInst.Full\_Comand == "")

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(20, 5), 2), 10);//RD

AsmInst.Op2 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(12, 5), 2), 10);//RS1

AsmInst.Op3 = Convert.ToString(Convert.ToInt32(Instruction.Substring(0, 12), 2), 16);//Imm

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", 0x" + AsmInst.Op3 + "(" + AsmInst.Op2 + ")";

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2 + " " + AsmInst.Op3.PadLeft(3,'0');

}

if ((AsmInst.Inst == "SB" || AsmInst.Inst == "SH" || AsmInst.Inst == "SW") && AsmInst.Full\_Comand == "")

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(7, 5), 2), 10);//RS1

AsmInst.Op2 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(12, 5), 2), 10);//RS2

AsmInst.Op3 = Instruction.Substring(0, 7) + Instruction.Substring(20, 5);

AsmInst.Op3 = Convert.ToString(Convert.ToInt32(AsmInst.Op3, 2), 16);

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", 0x" + AsmInst.Op3 + "(" + AsmInst.Op2 + ")";

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2 + " " + AsmInst.Op3;

}

if ((AsmInst.Inst == "ADDI" || AsmInst.Inst == "SLTI" || AsmInst.Inst == "SLTIU" || AsmInst.Inst == "XORI" ||

AsmInst.Inst == "ORI" || AsmInst.Inst == "ANDI") && AsmInst.Full\_Comand == "")

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(20, 5), 2), 10);//RD

AsmInst.Op2 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(12, 5), 2), 10);//RS1

AsmInst.Op3 = Convert.ToString(Convert.ToInt32(Instruction.Substring(0, 12), 2), 16);//Imm

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", " + AsmInst.Op2 + ", 0x" + AsmInst.Op3;

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2 + " " + AsmInst.Op3;

}

if ((AsmInst.Inst == "SLLI" || AsmInst.Inst == "SRLI" || AsmInst.Inst == "SRAI") && AsmInst.Full\_Comand == "")

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(20, 5), 2), 10);//RD

AsmInst.Op2 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(12, 5), 2), 10);//RS1

AsmInst.Op3 = Convert.ToString(Convert.ToInt32(Instruction.Substring(7, 5), 2), 16);//shamt

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", " + AsmInst.Op2 + ", 0x" + AsmInst.Op3;

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2 + " " + AsmInst.Op3/;

}

if ((AsmInst.Inst == "ADD" || AsmInst.Inst == "SUB" || AsmInst.Inst == "SLL" || AsmInst.Inst == "SLT" ||

AsmInst.Inst == "SLTU" || AsmInst.Inst == "XOR" || AsmInst.Inst == "SRL" || AsmInst.Inst == "SRA" ||

AsmInst.Inst == "OR" || AsmInst.Inst == "AND") && AsmInst.Full\_Comand == "")

{

AsmInst.Op1 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(20, 5), 2), 10);//RD

AsmInst.Op2 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(12, 5), 2), 10);//RS1

AsmInst.Op3 = "x" + Convert.ToString(Convert.ToInt32(Instruction.Substring(7, 5), 2), 10);//RS2

AsmInst.Asm\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + ", " + AsmInst.Op2 + ", " + AsmInst.Op3;

AsmInst.Full\_Comand = AsmInst.Inst + " " + AsmInst.Op1 + " " + AsmInst.Op2 + " " + AsmInst.Op3;

}

if (AsmInst.Full\_Comand == "" || AsmInst.Asm\_Comand == "")

{

Full\_Comand = "";

Asm\_Comand = "Ошибка декодирования";

return 1;

}

else

{

Full\_Comand = AsmInst.Full\_Comand;

Asm\_Comand = AsmInst.Asm\_Comand;

return 0;

}

}

}

}

### Приложение Б

**ЛИСТИНГ Alg\_operation.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Text;

namespace Simulator\_RISCV

{

class Alg\_operation : INotifyPropertyChanged

{

public bool Stage;

string pc;

string Name\_op { get; set; } = "";

string Op1 { get; set; } = "";

string Number1 { get; set; } = "";

string Op2 { get; set; } = "";

string Number2 { get; set; } = "";

string Op3 { get; set; } = "";

public string PC

{

get { return pc; }

set

{

pc = value;

RaisePropertyChanged("PC");

}

}

string console;

public string Console

{

get { return console; }

set

{

console = value;

RaisePropertyChanged("Console");

}

}

int clk;

public int CLK

{

get { return clk; }

set

{

clk = value;

RaisePropertyChanged("CLK");

}

}

public static Dictionary<string, string> Data\_seg

{

get; set;

}

public event PropertyChangedEventHandler PropertyChanged;

public void RaisePropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

public Alg\_operation()

{

PC = "00000000";

Stage = true;

}

public void Reg\_init()

{

int i = 0;

while (i < 31)

{

if (i == 2)

Memory.Registers["x2"][1] = "0x00007FF0";

else

if(i == 3)

Memory.Registers["x3"][1] = "0x00001000";

else

Memory.Registers["x" + i][1] = "0x00000000";

i++;

}

}

public string Comand\_Real(string inst)

{

uint inp1;

uint inp2;

try

{

Name\_op = inst.Split(' ')[0].ToLower();

Op1 = inst.Split(' ')[1];

Op2 = inst.Split(' ')[2];

Op3 = inst.Split(' ')[3];

}

catch (Exception)

{

Op3 = "";

}

switch (Name\_op)

{

case ("lui"):

Number1 = (Op2 + "000").PadLeft(8, '0');

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

if (Op1 != "x0")

return "0x" + Number1.ToUpper() + " " + Op1;

return "";

case ("auipc"):

Number1 = Convert.ToString(Convert.ToUInt32(PC, 16) + Convert.ToInt32(Op2, 16), 16).PadLeft(8, '0');

PC = Convert.ToString((Convert.ToUInt32(PC, 16) + 4), 16).PadLeft(8, '0');

if (Op1 != "x0")

return "0x" + Number1.ToUpper() + " " + Op1;

return "";

case ("jal"):

Number1 = (Convert.ToUInt32(Memory.Registers[Op2][1], 16) + Convert.ToUInt32(PC, 16)).ToString("X");

PC = Number1.PadLeft(8, '0');

if (Op1 != "x0")

return "0x" + Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0').ToUpper() + " " + Op1;

return "";

case ("jalr"):

Number1 = (Convert.ToUInt32(Memory.Registers[Op2][1], 16) + Convert.ToUInt32(Op3, 16)).ToString("X");

Number2 = "0x" + Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0').ToUpper();

PC = Number1.PadLeft(8, '0');

if (Op1 != "x0")

return Number2 + " " + Op1;

return "";

case ("beq"):

Number1 = Memory.Registers[Op1][1].Substring(2, 8);

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Number1 == Number2)

PC = (Convert.ToUInt32(PC, 16) + Convert.ToUInt32(Op3, 16)).ToString("X").PadLeft(8, '0');

else

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "";

case ("bne"):

Number1 = Memory.Registers[Op1][1].Substring(2, 8);

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Number1 != Number2)

PC = (Convert.ToUInt32(PC, 16) + Convert.ToUInt32(Op3, 16)).ToString("X").PadLeft(8, '0');

else

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "";

case ("blt"):

Number1 = Memory.Registers[Op1][1].Substring(2, 8);

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Op3[0] > '8')

Op3 = Op3.PadLeft(8, 'F');

if (String.Compare(Number1, Number2) < 0)

PC = (Convert.ToUInt32(PC, 16) + Convert.ToUInt32(Op3, 16)).ToString("X").PadLeft(8, '0');

else

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "";

case ("bge"):

Number1 = Memory.Registers[Op1][1].Substring(2, 8);

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (String.Compare(Number1, Number2) > 0 || String.Compare(Number1, Number2) == 0)

PC = (Convert.ToUInt32(PC, 16) + Convert.ToUInt32(Op3, 16)).ToString("X").PadLeft(8, '0');

else

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "";

case ("bltu"):

Number1 = Memory.Registers[Op1][1].Substring(2, 8);

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Number1[0] >= '8')

{

inp1 = Convert.ToUInt32(Number1, 16);

inp1 = ~inp1 + 1;

}

else

{

inp1 = Convert.ToUInt32(Number1, 16);

}

if (Number2[0] >= '8')

{

inp2 = Convert.ToUInt32(Number2, 16);

inp2 = ~inp2 + 1;

}

else

{

inp2 = Convert.ToUInt32(Number2, 16);

}

if (inp1 < inp2)

{

PC = (Convert.ToUInt32(PC, 16) + Convert.ToUInt32(Op3, 16)).ToString("X").PadLeft(8, '0');

}

else

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "";

case ("bgeu"):

Number1 = Memory.Registers[Op1][1].Substring(2, 8);

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Number1[0] >= '8')

{

inp1 = Convert.ToUInt32(Number1, 16);

inp1 = ~inp1 + 1;

}

else

{

inp1 = Convert.ToUInt32(Number1, 16);

}

if (Number2[0] >= '8')

{

inp2 = Convert.ToUInt32(Number2, 16);

inp2 = ~inp2 + 1;

}

else

{

inp2 = Convert.ToUInt32(Number2, 16);

}

if (inp1 >= inp2)

{

PC = (Convert.ToUInt32(PC, 16) + Convert.ToUInt32(Op3, 16)).ToString("X").PadLeft(8, '0');

}

else

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "";

case ("lb"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);//rs1

Number2 = Convert.ToString(Convert.ToUInt32(Number2, 16) + Convert.ToUInt32(Op3, 16), 16).PadLeft(8, '0');

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number2.ToUpper() + " " + Op1;

case ("lh"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Convert.ToString(Convert.ToUInt32(Number2, 16) + Convert.ToUInt32(Op3, 16), 16).PadLeft(8, '0');

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number2.ToUpper() + " " + Op1;

case ("lw"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);//rs1

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8, 'f');

Number2 = Convert.ToString(Convert.ToUInt32(Number2, 16) + Convert.ToUInt32(Op3, 16), 16).PadLeft(8, '0');

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number2.ToUpper() + " " + Op1;

case ("lbu"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);//rs1

Number2 = Convert.ToString(Convert.ToUInt32(Number2, 16) + Convert.ToUInt32(Op3, 16), 16).PadLeft(8, '0');

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number2.PadLeft(8, '0').ToUpper() + " " + Op1;

case ("lhu"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Convert.ToString(Convert.ToUInt32(Number2, 16) + Convert.ToUInt32(Op3, 16), 16).PadLeft(8, '0');

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number2.ToUpper() + " " + Op1;

case ("sb"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Convert.ToString(Convert.ToUInt32(Number2, 16) + Convert.ToUInt32(Op3, 16), 16).PadLeft(8, '0');

Number1 = Memory.Registers[Op1][1].Substring(8, 2);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number2 + " " + Number1;

case ("sh"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Convert.ToString(Convert.ToUInt32(Number2, 16) + Convert.ToUInt32(Op3, 16), 16).PadLeft(8, '0');

Number1 = Memory.Registers[Op1][1].Substring(6, 4);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return Number2 + " " + Number1;

case ("sw"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Convert.ToString(Convert.ToUInt32(Number2, 16) + Convert.ToUInt32(Op3, 16), 16).PadLeft(8, '0');

Number1 = Memory.Registers[Op1][1].Substring(2, 8);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return Number2 + " " + Number1;

case ("addi"):

Op2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Op3.Length == 3)

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8, 'F');

Number1 = Convert.ToString(Convert.ToUInt32(Op2, 16) + Convert.ToUInt32(Op3, 16), 16).PadLeft(8, '0');

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number1.ToUpper() + " " + Op1;

case ("slti"):

Op2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8,'f');

if (String.Compare(Op2, Op3.PadLeft(8, '0')) < 0)

Number1 = "00000001";

else

Number1 = "00000000";

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number1.ToUpper() + " " + Op1;

case ("sltiu"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Number2 == "00000000")

Number1 = "00000001";

else

{

if (Number2[0] >= '8')

{

inp1 = Convert.ToUInt32(Number2, 16);

inp1 = ~inp1 + 1;

}

else

{

inp1 = Convert.ToUInt32(Number2, 16);

}

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8, 'F');

else

Op3 = Op3.PadLeft(8, '0');

if (inp1 < Convert.ToUInt32(Op3))

Number1 = "00000001";

else

Number1 = "00000000";

}

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number1.ToUpper() + " " + Op1;

case ("xori"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

if (Op3 == "ffffffff")

{

Number2 = (~Convert.ToUInt32(Number2, 16)).ToString("X");

return "0x" + Number2.ToUpper() + " " + Op1;

}

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8, 'F');

else

Op3 = Op3.PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number2, 16) ^ Convert.ToUInt32(Op3, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case ("ori"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8, 'F');

else

Op3 = Op3.PadLeft(8, '0');

PC = Convert.ToString((Convert.ToUInt32(PC, 16) + 4), 16).PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number2, 16) | Convert.ToUInt32(Op3, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case ("andi"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8, 'F');

else

Op3 = Op3.PadLeft(8, '0');

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number2, 16) & Convert.ToUInt32(Op3, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case ("slli"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8, 'F');

PC = Convert.ToString((Convert.ToUInt32(PC, 16) + 4), 16).PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number2, 16) << Convert.ToInt32(Op3, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case ("srli"):

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8, 'F');

PC = Convert.ToString((Convert.ToUInt32(PC, 16) + 4), 16).PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number2, 16) >> Convert.ToInt32(Op3, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case ("srai"):///

Number2 = Memory.Registers[Op2][1].Substring(2, 8);

if (Op3[0] >= '8')

Op3 = Op3.PadLeft(8, 'F');

Number1 = Convert.ToString(Convert.ToUInt32(Number2, 16) >> Convert.ToInt32(Op3, 16), 16);

if (Number2[0] >= '8')

{

Number1 = Convert.ToString(Convert.ToUInt32(Number1, 16), 2).PadLeft(32, '0').Substring(Convert.ToInt32(Op3, 16), 32 - Convert.ToInt32(Op3, 16)).PadLeft(32, '1');

Number1 = Convert.ToString(Convert.ToUInt32(Number1, 2), 16).PadLeft(8, '0');

}

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number1 + " " + Op1;

case ("add"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

try

{

Number1 = Convert.ToString(checked(Convert.ToUInt32(Number1, 16) + Convert.ToUInt32(Number2, 16)), 16);

//Записать в память

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number1.PadLeft(8, '0').ToUpper() + " " + Op1;

}

catch (Exception)

{

return "Возникло переполнение при выполнении оперции сложения";

}

case ("sub"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

try

{

Number1 = Convert.ToString(checked(Convert.ToUInt32(Number1, 16) - Convert.ToUInt32(Number2, 16)), 16);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number1.PadLeft(8, '0').ToUpper() + " " + Op1;

}

catch (Exception)

{

return "Возникло переполнение при выполнении оперции сложения";

}

case ("sll"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number1, 16) << Convert.ToInt32(Number2, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case ("slt"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

if (String.Compare(Number1, Number2) < 0)

return "0x00000001" + " " + Op1;

else

return "0x00000000" + " " + Op1;

case ("sltu"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

if (Number1 == "00000000")

Number1 = "00000001";

else

{

if (Number1[0] >= '8' && Number1[0] <= 'f')

{

inp1 = Convert.ToUInt32(Number1, 16);

inp1 = ~inp1 + 1;

}

else

{

inp1 = Convert.ToUInt32(Number1, 16);

}

if (Number2[0] >= '8' && Number2[0] <= 'f')

{

inp2 = Convert.ToUInt32(Number2, 16);

inp2 = ~inp2 + 1;

}

else

{

inp2 = Convert.ToUInt32(Number2, 16);

}

if (inp1 < inp2)

Number1 = "00000001";

else

Number1 = "00000000";

}

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number1 + " " + Op1;

case ("xor"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number1, 16) ^ Convert.ToUInt32(Number2, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case ("srl"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

PC = Convert.ToString((Convert.ToUInt32(PC, 16) + 4), 16).PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number1, 16) >> Convert.ToInt32(Number2, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case ("sra"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

Number1 = Convert.ToString(Convert.ToUInt32(Number1, 16) >> Convert.ToInt32(Number2, 16), 16);

if (Number2[0] >= '8')

{

Number1 = Convert.ToString(Convert.ToUInt32(Number1, 16), 2).PadLeft(32, '0').Substring(Convert.ToInt32(Op3, 16), 32 - Convert.ToInt32(Op3, 16)).PadLeft(32, '1');

Number1 = Convert.ToString(Convert.ToUInt32(Number1, 2), 16).PadLeft(8, '0');

}

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

return "0x" + Number1 + " " + Op1;

case ("or"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number1, 16) | Convert.ToUInt32(Number2, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case ("and"):

Number1 = Memory.Registers[Op2][1].Substring(2, 8);

Number2 = Memory.Registers[Op3][1].Substring(2, 8);

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

Number1 = (Convert.ToUInt32(Number1, 16) + Convert.ToUInt32(Number2, 16)).ToString("X").PadLeft(8, '0').ToUpper();

return "0x" + Number1 + " " + Op1;

case "ecall":

Number1 = Memory.Registers["x10"][1].Substring(2, 8);

if (Convert.ToInt32(Number1, 16) != 10)

{

PC = Convert.ToString(Convert.ToUInt32(PC, 16) + 4, 16).PadLeft(8, '0');

switch (Convert.ToInt32(Number1, 16))

{

case 1:

{

Number2 = Memory.Registers["x11"][1].Substring(2, 8);

Console += Convert.ToInt32(Number2, 16).ToString();

return "";

}

case 11:

{

Number2 = Memory.Registers["x11"][1].Substring(8, 2);

byte buf = Convert.ToByte(Convert.ToInt32(Number2, 16));

Console += Encoding.GetEncoding(1251).GetString((new byte[] { buf }));

return "";

}

default:

{

Console += "\necall don't worked with a0 = " + Number1;

return "ecall exit";

}

}

}

else

{

Console += "\nДостигнут конец программы!! :D";

return "ecall exit";

}

default:

Console += "\nДостигнут конец программы!! :D";

return "";

}

}

}

}

### Приложение В

**ЛИСТИНГ MEMORY.CS**

using System;

using System.Collections.Generic;

namespace Simulator\_RISCV

{

class Memory

{

public string Begin\_stack\_mem { get; } = "00007FF0";

public string Begin\_data\_mem { get; } = "00001000";

public string Code\_pointer { get; } = "00000000";

public static Dictionary<string, string[]> Registers

{

get; set;

}

public Memory()

{

Registers = new Dictionary<string, string[]>

{

{ "x0", new string[] {"zero", "0x00000000"} },

{ "x1", new string[] { "ra", "0x00000000" } },

{ "x2", new string[] { "sp", "0x00007FF0" } },

{ "x3", new string[] { "gp", "0x00001000" } },

{ "x4", new string[] { "tp", "0x00000000" } },

{ "x5", new string[] { "t0", "0x00000000" } },

{ "x6", new string[] { "t1", "0x00000000" } },

{ "x7", new string[] { "t2", "0x00000000" } },

{ "x8", new string[] { "s0", "0x00000000" } },

{ "x9", new string[] { "s1", "0x00000000" } },

{ "x10", new string[] { "a0", "0x00000000" } },

{ "x11", new string[] { "a1", "0x00000000" } },

{ "x12", new string[] { "a2", "0x00000000" } },

{ "x13", new string[] { "a3", "0x00000000" } },

{ "x14", new string[] { "a4", "0x00000000" } },

{ "x15", new string[] { "a5", "0x00000000" } },

{ "x16", new string[] { "a6", "0x00000000" } },

{ "x17", new string[] { "a7", "0x00000000" } },

{ "x18", new string[] { "s2", "0x00000000" } },

{ "x19", new string[] { "s3", "0x00000000" } },

{ "x20", new string[] { "s4", "0x00000000" } },

{ "x21", new string[] { "s5", "0x00000000" } },

{ "x22", new string[] { "s6", "0x00000000" } },

{ "x23", new string[] { "s7", "0x00000000" } },

{ "x24", new string[] { "s8", "0x00000000" } },

{ "x25", new string[] { "s9", "0x00000000" } },

{ "x26", new string[] { "s10", "0x00000000" } },

{ "x27", new string[] { "s11", "0x00000000" } },

{ "x28", new string[] { "t3", "0x00000000" } },

{ "x29", new string[] { "t4", "0x00000000" } },

{ "x30", new string[] { "t5", "0x00000000" } },

{ "x31", new string[] { "t6", "0x00000000" } }

};

}

public string Read\_data\_byte(string address)

{

string row = address.Substring(2, 7) + "0";

int offset = Convert.ToInt32(address, 16) % 16;

string[] buf = Alg\_operation.Data\_seg[row].Split(' ');

return buf[offset];

}

//funct load word to memory

public string Read\_data\_hw(string address)

{

string row = address.Substring(2, 7) + "0";

int offset = Convert.ToInt32(address, 16) % 16;

string[] buf = Alg\_operation.Data\_seg[row].Split(' ');

if (offset != 15)

return buf[offset + 1] + buf[offset];

else

return Alg\_operation.Data\_seg[(Convert.ToInt32(row,16) + 16).ToString("X").PadLeft(8, '0')].Split(' ')[0] + buf[offset];

}

//funct load word to memory

public string Read\_data\_word(string address)

{

string row = address.Substring(2, 7) + "0";

int offset = Convert.ToInt32(address, 16) % 16;

string[] buf = Alg\_operation.Data\_seg[row.ToLower()].Split(' ');

switch (offset)

{

case 13:

return Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ')[0] + buf[offset + 2] + buf[offset + 1] + buf[offset];

case 14:

return Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ')[1] + Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ')[0] + buf[offset + 1] + buf[offset];

case 15:

return Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ')[2] + Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ')[1] + Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ')[0] + buf[offset];

default:

return buf[offset + 3] + buf[offset + 2] + buf[offset + 1] + buf[offset];

}

}

//funct store to memory

public void Write\_data(string address, string data)

{

string row = address.Substring(0, 7) + "0";

int offset = Convert.ToInt32(address, 16) % 16;

string[] buf = Alg\_operation.Data\_seg[row].Split(' ');

switch (data.Length)

{

case 2:

{

buf[offset] = data;

Alg\_operation.Data\_seg[row] = String.Join(" ", buf);

break;

}

case 4:

{

if (offset != 15)

{

buf[offset + 1] = data.Substring(2, 2);

buf[offset] = data.Substring(0, 2);

Alg\_operation.Data\_seg[row] = String.Join(" ", buf);

}

else

{

buf[offset] = data.Substring(2, 2);

Alg\_operation.Data\_seg[row] = String.Join(" ", buf);

string[] buf2 = Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ');

buf2[0] = data.Substring(0, 2);

Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')] = String.Join(" ", buf2);

}

break;

}

case 8:

{

switch (offset)

{

case 13:

{

buf[offset + 2] = data.Substring(2, 2);

buf[offset + 1] = data.Substring(4, 2);

buf[offset] = data.Substring(6, 2);

Alg\_operation.Data\_seg[row] = String.Join(" ", buf);

string[] buf2 = Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ');

buf2[0] = data.Substring(0, 2);

Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')] = String.Join(" ", buf2);

break;

}

case 14:

{

buf[offset + 1] = data.Substring(4, 2);

buf[offset] = data.Substring(6, 2);

Alg\_operation.Data\_seg[row] = String.Join(" ", buf);

string[] buf2 = Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ');

buf2[0] = data.Substring(2, 2);

buf2[1] = data.Substring(0, 2);

Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')] = String.Join(" ", buf2);

break;

}

case 15:

{

buf[offset] = data.Substring(6, 2);

Alg\_operation.Data\_seg[row] = String.Join(" ", buf);

string[] buf2 = Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')].Split(' ');

buf2[0] = data.Substring(4, 2);

buf2[1] = data.Substring(2, 2);

buf2[2] = data.Substring(0, 2);

Alg\_operation.Data\_seg[(Convert.ToInt32(row, 16) + 16).ToString("X").PadLeft(8, '0')] = String.Join(" ", buf2);

break;

}

default:

{

buf[offset + 3] = data.Substring(0, 2);

buf[offset + 2] = data.Substring(2, 2);

buf[offset + 1] = data.Substring(4, 2);

buf[offset] = data.Substring(6, 2);

Alg\_operation.Data\_seg[row] = String.Join(" ", buf);

break;

}

}

break;

}

}

}

}

}

### Приложение Г

**ЛИСТИНГ Program.cs**

using System;

using System.Windows;

using System.Diagnostics;

using System.IO;

using Microsoft.Win32;

using System.Collections.Generic;

namespace Simulator\_RISCV

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public Dictionary<string, string> Code\_seg

{

get; set;

}

static public Dictionary<string, string> Stage\_conv

{

get; set;

}

public int wait;

string PC { get; set; }

string result\_ALU;

string result\_MEM;

string prev\_result\_ALU;

string prev\_result\_MEM;

string prev\_Decode;

static public StreamReader Reader { get; set; }

string Full, Asm, Prev\_Asm;

string instruction;

Alg\_operation Prov { get; set; }

Decoder Decoder { get; set; }

Memory Mem { get; set; }

public MainWindow()

{

Stage\_conv = new Dictionary<string, string>

{

{ "Fetch", "nop"},

{ "Decode", "nop"},

{ "Execute", "nop"},

{ "Memory", "nop"},

{ "Write back", "nop"},

{ "Wait", "0"}

};

InitializeComponent();

}

private void Window\_Initialized(object sender, EventArgs e)

{

DirectoryInfo dirInfo = new DirectoryInfo(Environment.CurrentDirectory + @"\src\data");

if (!dirInfo.Exists)

dirInfo.Create();

run\_btn.IsEnabled = step\_btn.IsEnabled = reset\_btn.IsEnabled = false;

Prov = new Alg\_operation();

Decoder = new Decoder();

Prov.Console = "";

Single.IsChecked = Prov.Stage = true;

Prov.CLK = 0;

DataContext = Prov;

grid\_stage.ItemsSource = Stage\_conv;

Code\_seg = new Dictionary<string, string>();

Alg\_operation.Data\_seg = new Dictionary<string, string>();

Set\_default();

Write\_reg();

grid\_code.ItemsSource = Code\_seg;

grid\_data.ItemsSource = Alg\_operation.Data\_seg;

data\_register.ItemsSource = Memory.Registers;

instr.Text = "";

Mem = new Memory();

}

void Set\_default()

{

int i = 0;

while (i < 256)

{

Code\_seg.Add(Convert.ToString(i \* 16, 16).PadLeft(8, '0'), "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ");

i++;

}

while (i < 2048)

{

Alg\_operation.Data\_seg.Add(Convert.ToString(i \* 16, 16).PadLeft(8, '0'), "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ");

i++;

}

}

private void Step\_btn\_Click(object sender, RoutedEventArgs e)

{

Prov.CLK += 4;

Single.IsEnabled = Five.IsEnabled = false;

reset\_btn.IsEnabled = true;

instruction = Read\_code();

Stage\_conv["Fetch"] = instruction;

if (instruction == "00000000")

{

run\_btn.IsEnabled = false;

step\_btn.IsEnabled = false;

Prov.Console = "Error PC:" + Prov.PC;

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

Write\_reg();

Write\_mem();

return;

}

if (Decoder.DecodeInstruction(instruction, out Full, out Asm) == 1)

{

Prov.Console = Asm + " " + instruction;

run\_btn.IsEnabled = false;

step\_btn.IsEnabled = false;

Prov.Console = "Error PC:" + Prov.PC;

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

Write\_reg();

Write\_mem();

return;

}

else

{

Stage\_conv["Write back"] = Stage\_conv["Memory"] = Stage\_conv["Execute"] = Stage\_conv["Decode"] = Asm;

Prov.PC = Prov.PC.ToUpper();

instr.Text += Prov.PC + "\t" + instruction + "\t" + Asm + "\n";

result\_ALU = Prov.Comand\_Real(Full);

PC = Prov.PC;

if (result\_ALU == "ecall exit")

{

step\_btn.IsEnabled = run\_btn.IsEnabled = false;

reset\_btn.IsEnabled = Single.IsEnabled = Five.IsEnabled = true;

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

Write\_reg();

Write\_mem();

return;

}

if (result\_ALU != "")

{

result\_MEM = Load\_store(result\_ALU, Stage\_conv["Memory"]);

}

//stage WB

if (result\_MEM != "")

Memory.Registers[result\_MEM.Split(' ')[1]][1] = result\_MEM.Split(' ')[0];

}

Write\_reg();

Write\_mem();

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

}

private void Run\_btn\_Click(object sender, RoutedEventArgs e)

{

while (true)

{

Prov.CLK += 4;

Single.IsEnabled = Five.IsEnabled = false;

reset\_btn.IsEnabled = true;

instruction = Read\_code();

Stage\_conv["Fetch"] = instruction;

if (instruction == "00000000")

{

run\_btn.IsEnabled = false;

step\_btn.IsEnabled = false;

Prov.Console = "Error PC:" + Prov.PC;

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

break;

}

if (Decoder.DecodeInstruction(instruction, out Full, out Asm) == 1)

{

Prov.Console = Asm + " " + instruction;

run\_btn.IsEnabled = false;

step\_btn.IsEnabled = false;

Prov.Console = "Error PC:" + Prov.PC;

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

break;

}

else

{

Stage\_conv["Write back"] = Stage\_conv["Memory"] = Stage\_conv["Execute"] = Stage\_conv["Decode"] = Asm;

Prov.PC = Prov.PC.ToUpper();

instr.Text += Prov.PC + "\t" + instruction + "\t" + Asm + "\n";

result\_ALU = Prov.Comand\_Real(Full);

PC = Prov.PC;

if (result\_ALU == "ecall exit")

{

step\_btn.IsEnabled = run\_btn.IsEnabled = false;

reset\_btn.IsEnabled = Single.IsEnabled = Five.IsEnabled = true;

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

break;

}

if (result\_ALU != "")

{

result\_MEM = Load\_store(result\_ALU, Stage\_conv["Memory"]);

}

//stage WB

if (result\_MEM != "")

Memory.Registers[result\_MEM.Split(' ')[1]][1] = result\_MEM.Split(' ')[0];

}

}

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

Write\_reg();

Write\_mem();

}

private void Step\_btn\_Click\_5(object sender, RoutedEventArgs e)

{

Prov.CLK++;

Single.IsEnabled = Five.IsEnabled = false;

reset\_btn.IsEnabled = true;

if (wait != 0)

{

if (wait == 1)

{

Prov.PC = Prov.PC.ToUpper();

instr.Text += Prov.PC + "\t" + prev\_Decode + "\t" + Prev\_Asm + "\n";

result\_ALU = Prov.Comand\_Real(Stage\_conv["Execute"]);

if (result\_ALU == "ecall exit")

{

step\_btn.IsEnabled = run\_btn.IsEnabled = false;

reset\_btn.IsEnabled = Single.IsEnabled = Five.IsEnabled = true;

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

return;

}

if (Stage\_conv["Execute"][0] == 'B' || Stage\_conv["Execute"][0] == 'J')

{

Stage\_conv["Fetch"] = Stage\_conv["Decode"] = "nop";

PC = Prov.PC;

}

}

else // wait 2 steps

{

Stage\_conv["Write back"] = Stage\_conv["Memory"];

Stage\_conv["Memory"] = "nop";

Memory.Registers[result\_MEM.Split(' ')[1]][1] = result\_MEM.Split(' ')[0];

result\_MEM = "";

}

wait--;

Stage\_conv["Wait"] = wait.ToString();

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

}

else

{

instruction = Read\_code();

if (instruction == "00000000")

instruction = "nop";

Stage\_conv["Write back"] = Stage\_conv["Memory"];

Stage\_conv["Memory"] = Stage\_conv["Execute"];

Stage\_conv["Execute"] = Stage\_conv["Decode"];

Stage\_conv["Decode"] = Stage\_conv["Fetch"];

Stage\_conv["Fetch"] = instruction;

if (Stage\_conv["Decode"] != "nop")

{

Prev\_Asm = Asm;

prev\_Decode = Stage\_conv["Decode"];

if (Decoder.DecodeInstruction(instruction, out Full, out Asm) == 1)

{

Prov.Console = Asm + "\t" + instruction;

run\_btn.IsEnabled = false;

step\_btn.IsEnabled = false;

Prov.Console = "Error PC:" + Prov.PC;

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

Write\_reg();

Write\_mem();

return;

}

Stage\_conv["Decode"] = Full;

}

prev\_result\_ALU = result\_ALU;

PC = Convert.ToString(Convert.ToInt32(PC, 16) + 4, 16).PadLeft(8, '0');

if (Stage\_conv["Execute"] != "nop")

{

string[] buf1 = Stage\_conv["Execute"].Split(' ');

if (buf1[0].ToUpper() == "ECALL")

{

Array.Resize(ref buf1, 3);

buf1[1] = "x10";

buf1[2] = "x11";

}

if (prev\_result\_ALU != "" && prev\_result\_ALU.Split(' ')[1][0] == 'x')

{

int i = 2;

if (buf1[0] == "SB" || buf1[0] == "SH" || buf1[0] == "SW" || buf1[0][0] == 'B' || buf1[0][0] == 'E')

i--;

for (; i < buf1.Length; i++)

{

if (buf1[i][0] == 'x')

if (buf1[i] == prev\_result\_ALU.Split(' ')[1])

{

wait = 2;

Stage\_conv["Wait"] = wait.ToString();

break;

}

}

}

if (wait != 2)

if (prev\_result\_MEM != "" && prev\_result\_MEM.Split(' ')[1][0] == 'x')

{

int i = 2;

if (buf1[0] == "SB" || buf1[0] == "SH" || buf1[0] == "SW" || buf1[0][0] == 'B' || buf1[0][0] == 'E')

i--;

for (; i < buf1.Length; i++)

{

if (buf1[i][0] == 'x')

if (buf1[i] == prev\_result\_MEM.Split(' ')[1])

{

wait = 1;

Stage\_conv["Wait"] = wait.ToString();

break;

}

}

}

if (wait == 0)

{

Prov.PC = Prov.PC.ToUpper();

instr.Text += Prov.PC + "\t" + prev\_Decode + "\t" + Prev\_Asm + "\n";

result\_ALU = Prov.Comand\_Real(Stage\_conv["Execute"]);

if (result\_ALU == "ecall exit")

{

step\_btn.IsEnabled = run\_btn.IsEnabled = false;

reset\_btn.IsEnabled = Single.IsEnabled = Five.IsEnabled = true;

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

return;

}

if (Stage\_conv["Execute"][0] == 'B' || Stage\_conv["Execute"][0] == 'J')

{

Stage\_conv["Fetch"] = Stage\_conv["Decode"] = "nop";

PC = Prov.PC;

}

}

}

//stage memory

prev\_result\_MEM = result\_MEM;

if (Stage\_conv["Memory"] != "nop" && prev\_result\_ALU != "")

{

result\_MEM = Load\_store(prev\_result\_ALU, Stage\_conv["Memory"]);

}

//stage WB

if (Stage\_conv["Write back"] != "nop" && prev\_result\_MEM != "")

Memory.Registers[prev\_result\_MEM.Split(' ')[1]][1] = prev\_result\_MEM.Split(' ')[0];

Write\_reg();

Write\_mem();

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

}

}

private void Run\_btn\_Click\_5(object sender, RoutedEventArgs e)

{

Single.IsEnabled = Five.IsEnabled = false;

while (true)

{

Prov.CLK++;

if (wait != 0)

{

if (wait == 1)

{

Prov.PC = Prov.PC.ToUpper();

instr.Text += Prov.PC + "\t" + prev\_Decode + "\t" + Prev\_Asm + "\n";

result\_ALU = Prov.Comand\_Real(Stage\_conv["Execute"]);

if (result\_ALU == "ecall exit")

{

reset\_btn.IsEnabled = Single.IsEnabled = Five.IsEnabled = true;

step\_btn.IsEnabled = run\_btn.IsEnabled = false;

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

Write\_reg();

Write\_mem();

return;

}

if (Stage\_conv["Execute"][0] == 'B' || Stage\_conv["Execute"][0] == 'J')

{

Stage\_conv["Fetch"] = Stage\_conv["Decode"] = "nop";

PC = Prov.PC;

}

}

else // wait 2 steps

{

Stage\_conv["Write back"] = Stage\_conv["Memory"];

Stage\_conv["Memory"] = "nop";

Memory.Registers[result\_MEM.Split(' ')[1]][1] = result\_MEM.Split(' ')[0];

result\_MEM = "";

}

Stage\_conv["Wait"] = wait.ToString();

wait--;

}

else

{

//добавить проверку на конец кода

instruction = Read\_code();

if (instruction == "00000000")

instruction = "nop";

Stage\_conv["Write back"] = Stage\_conv["Memory"];

Stage\_conv["Memory"] = Stage\_conv["Execute"];

Stage\_conv["Execute"] = Stage\_conv["Decode"];

Stage\_conv["Decode"] = Stage\_conv["Fetch"];

Stage\_conv["Fetch"] = instruction;

if (Stage\_conv["Decode"] != "nop")

{

Prev\_Asm = Asm;

prev\_Decode = Stage\_conv["Decode"];

if (Decoder.DecodeInstruction(instruction, out Full, out Asm) == 1)

{

Prov.Console = Asm + "\t" + instruction;

run\_btn.IsEnabled = false;

step\_btn.IsEnabled = false;

Prov.Console = "Error PC:" + Prov.PC;

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

Write\_reg();

Write\_mem();

return;

}

Stage\_conv["Decode"] = Full;

}

prev\_result\_ALU = result\_ALU;

PC = Convert.ToString(Convert.ToInt32(PC, 16) + 4, 16).PadLeft(8, '0');

if (Stage\_conv["Execute"] != "nop")

{

string[] buf1 = Stage\_conv["Execute"].Split(' ');

if (buf1[0].ToUpper() == "ECALL")

{

Array.Resize(ref buf1, 3);

buf1[1] = "x10";

buf1[2] = "x11";

}

if (prev\_result\_ALU != "" && prev\_result\_ALU.Split(' ')[1][0] == 'x')

{

int i = 2;

if (buf1[0] == "SB" || buf1[0] == "SH" || buf1[0] == "SW" || buf1[0][0] == 'B' || buf1[0][0] == 'E')

i--;

for (; i < buf1.Length; i++)

{

if (buf1[i][0] == 'x')

if (buf1[i] == prev\_result\_ALU.Split(' ')[1])

{

wait = 2;

Stage\_conv["Wait"] = wait.ToString();

break;

}

}

}

if (wait != 2)

if (prev\_result\_MEM != "" && prev\_result\_MEM.Split(' ')[1][0] == 'x')

{

int i = 2;

if (buf1[0] == "SB" || buf1[0] == "SH" || buf1[0] == "SW" || buf1[0][0] == 'B' || buf1[0][0] == 'E')

i--;

for (; i < buf1.Length; i++)

{

if (buf1[i][0] == 'x')

if (buf1[i] == prev\_result\_MEM.Split(' ')[1])

{

Stage\_conv["Wait"] = wait.ToString();

wait = 1;

break;

}

}

}

if (wait == 0)

{

Prov.PC = Prov.PC.ToUpper();

instr.Text += Prov.PC + "\t" + prev\_Decode + "\t" + Prev\_Asm + "\n";

result\_ALU = Prov.Comand\_Real(Stage\_conv["Execute"]);

if (result\_ALU == "ecall exit" || result\_ALU == "Возникло переполнение при выполнении оперции сложения")

{

step\_btn.IsEnabled = run\_btn.IsEnabled = false;

reset\_btn.IsEnabled = Single.IsEnabled = Five.IsEnabled = true;

grid\_stage.Items.Refresh();

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

Write\_reg();

Write\_mem();

return;

}

if (Stage\_conv["Execute"][0] == 'B' || Stage\_conv["Execute"][0] == 'J')

{

Stage\_conv["Fetch"] = Stage\_conv["Decode"] = "nop";

PC = Prov.PC;

}

}

}

//stage memory

prev\_result\_MEM = result\_MEM;

if (Stage\_conv["Memory"] != "nop" && prev\_result\_ALU != "")

result\_MEM = Load\_store(prev\_result\_ALU, Stage\_conv["Memory"]);

//stage WB

if (Stage\_conv["Write back"] != "nop" && prev\_result\_MEM != "")

Memory.Registers[prev\_result\_MEM.Split(' ')[1]][1] = prev\_result\_MEM.Split(' ')[0];

}

}

}

string Read\_code()

{

if (String.Compare(PC, "00001000") < 0)

{

string row = PC.Substring(0, 7) + "0";

string[] buf = Code\_seg[row.ToLower()].Split(' ');

int offset = Convert.ToInt32(PC, 16) % 16;

return buf[offset + 3] + buf[offset + 2] + buf[offset + 1] + buf[offset];

}

else return "";

}

private void MenuItem\_Click(object sender, RoutedEventArgs e)

{

string filePath;

Prov.Reg\_init();

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "asm files (\*.S)|\*.S";

openFileDialog.RestoreDirectory = true;

if (openFileDialog.ShowDialog() == true)

{

filePath = openFileDialog.FileName;

Process pr = Process.Start(@"toolchain\create\_hex.bat", filePath.Substring(0, filePath.Length - 2) + " " + Environment.CurrentDirectory + @"\src\data");

while (!pr.HasExited) { }

try

{

Reader = new StreamReader(@"src\data\memory.hex");

}

catch (Exception)

{

Prov.Console = "File:" + openFileDialog.SafeFileName + " not corrected";

return;

}

string buf;

Reader.ReadLine();

int i = 0;

while (i < 256)

{

Code\_seg[Convert.ToString(i \* 16, 16).PadLeft(8, '0')] = "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ";

i++;

}

while (i < 2048)

{

Alg\_operation.Data\_seg[Convert.ToString(i \* 16, 16).PadLeft(8, '0')] = "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ";

i++;

}

i = 0;

while (true)

{

buf = Reader.ReadLine();

if (buf != Code\_seg[Convert.ToString(i \* 16, 16).PadLeft(8, '0')])

{

Code\_seg[Convert.ToString(i \* 16, 16).PadLeft(8, '0')] = buf;

i++;

}

else break;

}

i = 256;

grid\_code.Items.Refresh();

while (true)

{

if (Reader.ReadLine()[0] == '@')

break;

}

while (true)

{

buf = Reader.ReadLine();

if (buf != Alg\_operation.Data\_seg[Convert.ToString(i \* 16, 16).PadLeft(8, '0')])

{

Alg\_operation.Data\_seg[Convert.ToString(i \* 16, 16).PadLeft(8, '0')] = buf;

i++;

}

else break;

}

for (; i < 32; i++)

{

if (i == 2)

Memory.Registers["x" + i][1] = "0x00007FF0";

else

if (i == 3)

Memory.Registers["x" + i][1] = "0x00001000";

else

Memory.Registers["x" + i][1] = "0x00000000";

}

Reader.Close();

result\_ALU = result\_MEM = instr.Text = Prov.Console = prev\_result\_MEM = prev\_result\_ALU = Prev\_Asm = prev\_Decode = "";

wait = Prov.CLK = 0;

Stage\_conv["Write back"] = Stage\_conv["Memory"] = Stage\_conv["Execute"] = Stage\_conv["Decode"] = Stage\_conv["Fetch"] = "nop";

Stage\_conv["Wait"] = "0";

Prov.PC = PC = "00000000";

run\_btn.IsEnabled = step\_btn.IsEnabled = true;

grid\_data.Items.Refresh();

data\_register.Items.Refresh();

Reader.Close();

}

}

void Write\_reg()

{

DirectoryInfo dirInfo = new DirectoryInfo(Environment.CurrentDirectory + @"\src\data");

if (!dirInfo.Exists)

dirInfo.Create();

using (StreamWriter fs = new StreamWriter(@"src/data/register.hex"))

{

foreach (var buf in Memory.Registers)

{

fs.WriteLine(buf.Key + "\t\t" + buf.Value[1]);

}

}

}

private void Reset\_btn\_Click(object sender, RoutedEventArgs e)

{

Prov.PC = PC = "00000000";

Stage\_conv["Write back"] = Stage\_conv["Memory"] = Stage\_conv["Execute"] = Stage\_conv["Decode"] = Stage\_conv["Fetch"] = "nop";

Stage\_conv["Wait"] = "0";

int i = 1;

for (; i < 32; i++)

{

if (i == 2)

Memory.Registers["x" + i][1] = "0x00007FF0";

else

if (i == 3)

Memory.Registers["x" + i][1] = "0x00001000";

else

Memory.Registers["x" + i][1] = "0x00000000";

}

i = 256;

while (i < 2048)

{

Alg\_operation.Data\_seg[Convert.ToString(i \* 16, 16).PadLeft(8, '0')] = "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ";

i++;

}

try

{

Reader = new StreamReader(@"src\data\memory.hex");

}

catch (Exception)

{

Prov.Console = "File:" + @"src\data\memory.hex" + " not find";

return;

}

string buf;

Reader.ReadLine();

i = 0;

while (true)

{

if (Reader.ReadLine()[0] == '@')

break;

i++;

}

while (true)

{

buf = Reader.ReadLine();

if (buf != Alg\_operation.Data\_seg[Convert.ToString(i \* 16, 16).PadLeft(8, '0')])

{

Alg\_operation.Data\_seg[Convert.ToString(i \* 16, 16).PadLeft(8, '0')] = buf;

i++;

}

else break;

}

Reader.Close();

result\_ALU = result\_MEM = instr.Text = Prov.Console = prev\_result\_MEM = prev\_result\_ALU = Prev\_Asm = prev\_Decode = "";

wait = Prov.CLK = 0;

grid\_stage.Items.Refresh();

data\_register.Items.Refresh();

run\_btn.IsEnabled = step\_btn.IsEnabled = true;

instr.Text = "";

}

private void Single\_Click(object sender, RoutedEventArgs e)

{

step\_btn.Click += Step\_btn\_Click;

run\_btn.Click += Run\_btn\_Click;

step\_btn.Click -= Step\_btn\_Click\_5;

run\_btn.Click -= Run\_btn\_Click\_5;

Prov.Stage = Single.IsChecked = true;

Five.IsChecked = false;

}

private void Five\_Click(object sender, RoutedEventArgs e)

{

step\_btn.Click += Step\_btn\_Click\_5;

run\_btn.Click += Run\_btn\_Click\_5;

step\_btn.Click -= Step\_btn\_Click;

run\_btn.Click -= Run\_btn\_Click;

Prov.Stage = Single.IsChecked = false;

Five.IsChecked = true;

}

private void Save\_log\_Click(object sender, RoutedEventArgs e)

{

DirectoryInfo dirInfo = new DirectoryInfo(Environment.CurrentDirectory + @"\log");

if (!dirInfo.Exists)

dirInfo.Create();

using (StreamWriter fs = new StreamWriter(@"log/result\_log.log", true))

{

fs.Write(instr.Text);

}

}

private void Guide\_Click(object sender, RoutedEventArgs e)

{

string text = "Пример выполнения:\n 1. Выбрать файл формата .S для записи в память. (File-> Open…)\n 2. Выбрать режим работы 5 - стадийный конвейер или нет. (Pipeline->Single / Five - Stage)\n"

+ " 3. Если Вы хотите выполнить программу пошагово, то нажимать на кнопку Step до завершения программы, в противном случае нажать на кнопку Run.\n"

+ " 4. После завершения программы кнопки Step и Run станут не доступны, необходимо выполнить Reset.\n"

+ " 5. Загрузить новую программу(пункт 1).\n"

+ "Дополнительная информация:\n"

+ " Кнопка Reset: код программы остается в памяти, но очищается память (data сегмент программы остаётся) и регистры.\n"

+ " File->Save log-file: При необходимости можно сохранить пройденные шаги в log-file.";

MessageBox.Show(text, "Guide", MessageBoxButton.OK, MessageBoxImage.Information);

}

private void About\_Click(object sender, RoutedEventArgs e)

{

string text = "Данный симулятор разработан в качестве курсового проекта. \n" +

"Contact:\nEvtushenko Oleg: evtushenko.mai.ru@mail.ru\nProzhirko Vladislav: \nSamoylov Vladislav: ";

MessageBox.Show(text, "About", MessageBoxButton.OK, MessageBoxImage.Question);

}

string Load\_store(string data, string instr)

{

string[] buf = data.Split(' ');

string buf2 = instr.Split(' ')[0];

if (buf2 == "SB" || buf2 == "SH" || buf2 == "SW")

{

Mem.Write\_data(buf[0], buf[1]);

return "";

}

else

switch (buf2)

{

case "LB":

{

if (buf[1] != "x0")

{

buf[0] = Mem.Read\_data\_byte(buf[0]);

if (buf[0][0] >= '8')

return "0x" + buf[0].PadLeft(8, 'F').ToUpper() + " " + buf[1];

else

return "0x" + buf[0].PadLeft(8, '0').ToUpper() + " " + buf[1];

}

else

return "";

}

case "LH":

{

if (buf[1] != "x0")

{

buf[0] = Mem.Read\_data\_hw(buf[0]);

if (buf[0][0] >= '8')

return "0x" + buf[0].PadLeft(8, 'F').ToUpper() + " " + buf[1];

else

return "0x" + buf[0].PadLeft(8, '0').ToUpper() + " " + buf[1];

}

else

return "";

}

case "LW":

{

if (buf[1] != "x0")

{

buf[0] = Mem.Read\_data\_word(buf[0]);

return "0x" + buf[0].ToUpper() + " " + buf[1];

}

else

return "";

}

case "LHU":

{

if (buf[1] != "x0")

{

buf[0] = Mem.Read\_data\_hw(buf[0]);

return "0x" + buf[0].PadLeft(8, '0').ToUpper() + " " + buf[1];

}

else

return "";

}

case "LBU":

{

if (buf[1] != "x0")

{

buf[0] = Mem.Read\_data\_byte(buf[0]);

return "0x" + buf[0].PadLeft(8, '0').ToUpper() + " " + buf[1];

}

else

return "";

}

default:

{

if (buf2[0] != 'B')

return data;

else

return "";

}

}

}

void Write\_mem()

{

DirectoryInfo dirInfo = new DirectoryInfo(Environment.CurrentDirectory + @"\src\data");

if (!dirInfo.Exists)

dirInfo.Create();

using (StreamWriter fs = new StreamWriter(@"src/data/memory.hex"))

{

fs.WriteLine("@00000000");

foreach (var buf in Code\_seg)

{

fs.WriteLine(buf.Value);

}

fs.WriteLine("@00001000");

foreach (var buf in Alg\_operation.Data\_seg)

{

fs.WriteLine(buf.Value);

}

}

}

}

}

### Приложение Д

**ЛИСТИНГ ПРОГРАММ ДЛЯ ТЕСТИРОВАНИЯ**

**Листинг Test1.S**

.data

N:

.word 5

M:

.word 4

matrix:

.word 7, 4, 5, 5, 7

.word 11, 4, 5, 8, 10

.word 3, 12, 2, 6 ,4

.word 9, 7, 4, 8, 3

result:

.word 0

.text

main:

la a2, N

la a3, M

la a4, matrix

la a5, result

call process

call print

call exit

print:

addi a0, x0, 1

lw a1, 0(a5)

ecall

ret

exit:

addi a0, x0, 10

ecall

process:

lw t1, 0(a2) # N

lw t2, 0(a3) # M

addi t4, x0, 1 # row pointer

addi t3, x0, 0 # row element number

lw t0, 0(a4) # t0-минимальный элемент.

addi a4, a4, 4 # increment array pointer

cycle:

lw t5, 0(a4) # load value from array

blt t0, t5, cont

add t0, x0, t5

cont:

addi a4, a4, 4 # increment array pointer

addi t4, t4, 1

blt t4, t1, cycle

addi t4, x0, 0

addi t3, t3, 1

blt t3, t2, cycle

sw t0, 0(a5) # save result

ret

**Листинг Test2.S**

.data

N:

.word 4

M:

.word 3

matrix:

.word 1, 2, 3

.word 9, 5, 2

.word 3, 10, 3

.word 3, 8, 3

.text

main:

lw t0, N

addi sp, sp, -12

sw t0, 8(sp)

lw t0, M

sw t0, 4(sp)

la t0, matrix

sw t0, 0(sp)

call process

call res

call exit

res:

addi a0, x0, 1

lw a1, 4 (sp)

ecall

addi a0, x0, 11 # print\_char ecall

addi a1, x0, 32

ecall

addi a0, x0, 1

lw a1, 0 (sp)

addi sp,sp, 8

ecall

ret

exit:

addi a0, x0, 10

ecall

process:

lw t0, 0(sp) # arr pointer

addi sp,sp, 4

lw t1, 0(sp) # M

addi sp, sp, 4

lw t2, 0(sp) # N

addi t3, x0, 1 # row pointer

addi t4, x0, 0 # row element number

lw t5, 0(t0) # max

addi sp, sp, -8

sw x0, 0(sp) # Imax

sw x0, 4(sp) # Jmax

addi t0, t0, 4

loop:

lw t6, 0(t0) # load value from array

bge t5, t6, cont

addi t5, t6, 0

sw t3, 0(sp)

sw t4, 4(sp)

cont:

addi t0, t0, 4 # increment array pointer

addi t3, t3, 1

blt t3, t1, loop

addi t3, x0, 0

addi t4, t4, 1 # increment row element number

blt t4, t2, loop

ret

**Листинг Test3.S**

.data

N:

.word 5

M:

.word 5

matrix:

.word 6, 1, 1, 1, 1

.word 2, 7, 2, 2, 2

.word 3, 3, 8, 3, 3

.word 4, 4, 4, 9, 4

.word 5, 5, 5, 5, 10

result:

.word 0

.text

main:

la a2, N

la a3, M

la a4, matrix

la a5, result

call process

call print

call exit

print:

addi a0, x0, 1 # print\_int ecall

lw a1, 0(a5)

ecall

exit:

addi a0, x0, 10

ecall

process:

lw t1, 0(a2) # N

lw t2, 0(a3) # M

addi t4, x0, 0 # row pointer

addi t5, x0, 0 # result

addi t3, x0, 0 # column pointer

cycle:

bne t4, t3, next

lw t6, 0(a4) # load value from array

add t5, t5, t6 # count sum

next:

addi a4,a4,4

addi t3, t3, 1

blt t3, t1, cycle

addi t4, t4, 1

addi t3, x0, 0 # column pointer

blt t4, t2, cycle

sw t5, 0(a5) # save result to resulting array

ret