Федеральное государственное автономное образовательное учреждение  
высшего образования   
"Национальный исследовательский университет   
"Высшая школа экономики"

Московский институт электроники и математики им. А. Н. Тихонова

Департамент компьютерной инженерии

Отчёт

о выполнении практической работы № 3

Тема работы: «Разработка и программирование Soft-процессорных ядер с архитектурой однотактный MIPS. Часть 3»

по курсу «Системное проектирование цифровых устройств»

Выполнили:

Власов Р. В. БИВ186

Сегида Т. О. БИВ186

Принял

асс. МИЭМ НИУ ВШЭ

Американов А. А.

Оценка:

Москва 2021 г.

Оглавление

[1. Добавление памяти 3](#_Toc52308632)

[1.1. Добавление IP-блока памяти 3](#_Toc52308633)

[1.2. Добавление памяти средствами Verilog 3](#_Toc52308634)

[2. Самостоятельная работа 3](#_Toc52308635)

[2.1. Задание № 1 3](#_Toc52308636)

[2.1.1. Добавить блок памяти 3](#_Toc52308637)

[2.1.2. Добавить команды LW и SW в тракт передачи данных 3](#_Toc52308638)

[2.2. Задание № 2 5](#_Toc52308639)

[2.2.1. Разработать программу на ассемблере, которая инвертирует все нечетные числа в памяти 5](#_Toc52308640)

[2.2.2. Провести прототипирование процессора и выполняемой программы 6](#_Toc52308641)

[2.2.3. Сравнить данный вариант процессора с версией 01\_mmio 6](#_Toc52308642)

[2.3. Задание № 3 6](#_Toc52308643)

[2.3.1. Перейти в ветку проекта 03\_pipeline 6](#_Toc52308644)

[2.3.2. Выполнить программы 00\_counter, 01\_fibonacci, 02\_sqrt, 03\_ram 6](#_Toc52308645)

[2.3.3. Выполнить программы 06\_hz\_forward, 07\_hz\_stall, 08\_hz\_branc 7](#_Toc52308646)

[2.3.4. Объяснить, чем эта версия процессора отличается от базовой 9](#_Toc52308647)

[3. Выводы 10](#_Toc52308648)

[4. Список литературы 10](#_Toc52308649)

1. Добавление памяти
   1. Добавление IP-блока памяти

Результаты генерации находятся в lab\_3\schoolMIPS\board\de10\_standard\generated\.

* 1. Добавление памяти средствами Verilog

Результаты находятся в lab\_3\schoolMIPS\board\de10\_standard\written\.

1. Самостоятельная работа
   1. Задание № 1
2. Добавить в тракт передачи данных блок памяти на 32 (переключателей 5, почему бы не сделать тогда на 32) 32-хбитных слов с двумя входами/выходами (1-ый вход-выход используется для работы процессора, адресация в памяти начинается с 0 адреса, 2-ой вход-выход используется для подключения внешнего управления (адресный вход получает значение с переключателей, меняя значения которых можно просматривать содержимое памяти, выход данных не подключен, вход разрешения записи не подключен))
3. Добавить команды LW и SW в тракт передачи данных (память инициализируется hex-файлом, содержащим 32 32-битных слова)

Данное задание выполнялось на версии процессора 00\_simple.

* + 1. Добавить блок памяти

На 32 32-битных слова с двумя входами/выходами. Используется модуль ram\_2\_port, сгенерированный в предыдущей части лабораторной работы

Для дальнейшего выполнения этого пункта задания целесообразно перейти к следующему пункту.

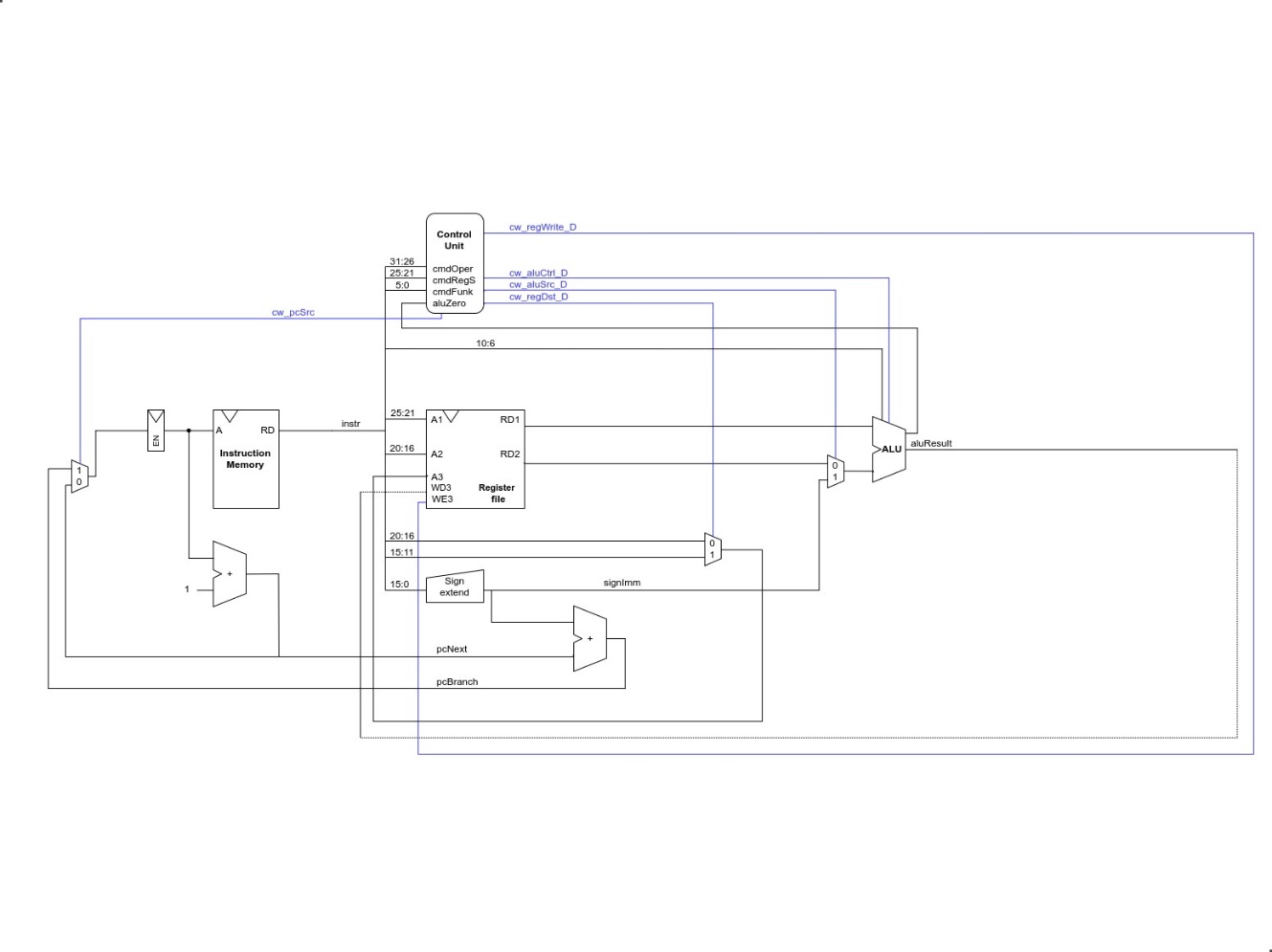
* + 1. Добавить команды LW и SW в тракт передачи данных

Память инициализируется hex-файлом, содержащим 32 32-битных слова. Данный файл находится в lab\_3\schoolMIPS\program\03\_task\_2\_1\_1\.

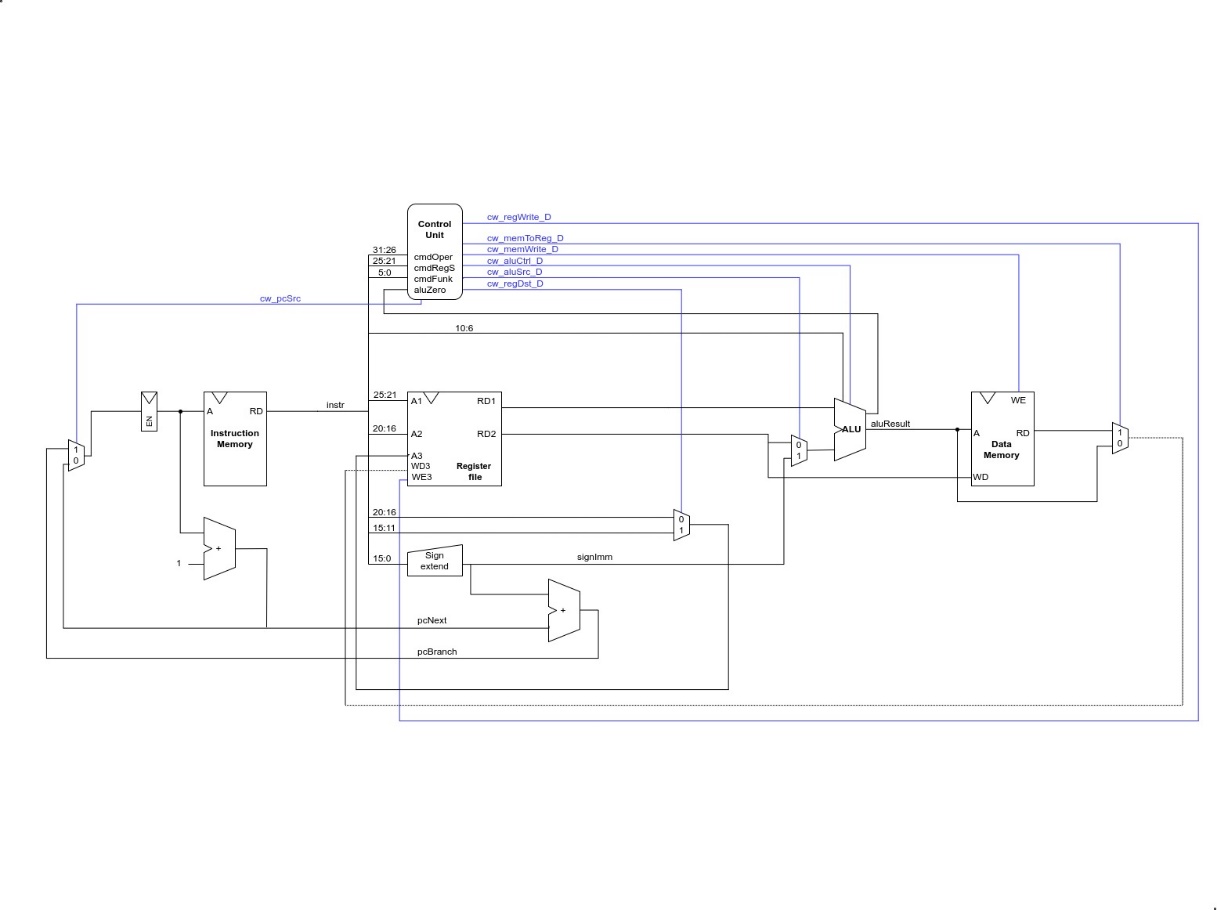
1-й вход-выход используется для работы процессора, адресация в памяти начинается с 0 адреса. Сделано также, как в книге H&H в главе 7.3.1, чтобы память была как-то задействована для работы процессора.

Чтобы упростить задачу, сравним версии процессора 00\_simple и 01\_mmio. Заметим, что вторая версия архитектурно отличается только наличием необходимой нам памяти (и реализаций, в следствие, команд lw и sw). Как видно на рисунках ниже, все, что нужно для добавления памяти в процессор это:

1. добавить саму память данных в разрыв между регистровым файлом и АЛУ (ram\_2\_port)
2. добавить мультиплексор перед входом WD3, который выбирает между выходом RD памяти и шиной AluResult
3. протянуть из порта RD2 регистрового файла шину в память данных в порт WD
4. добавить управляющий сигнал WE памяти данных (memWrite), разрешающий запись в память данных
5. добавить управляющий сигнал мультиплексору (из пункта 2 этого списка), выбирающий между памятью данных и результатом АЛУ



1. Версия процессора 00\_simple.



1. Версия процессора 01\_mmio.

address\_a – WD

data\_a – A

wren\_a – WE

q\_a – RD

добавили в sm\_cpu.vh (взято из 01\_mmio):

`define C\_LW 6'b100011 // I-type, Load Word

// Rt = memory[Rs + Immed]

`define C\_SW 6'b101011 // I-type, Store Word

// memory[Rs + Immed] = Rt

2-й вход-выход блока памяти используется для подключения внешнего управления (адресный вход получает значение с переключателей, меняя значения которых, можно просматривать содержимое памяти, выход данных не подключен, вход разрешения записи не подключен)

Чтобы не усложнять себе жизнь, приняли такое решение – переключатели, встроенные в плату, отвечавшие за выбор регистра, выводимого на семисегментные дисплеи, будут выбирать какое слово читается из памяти и выводится на семисегментный дисплей. Соответственно, на семисегментные дисплеи выводятся теперь данные из памяти данных, а не из регистров.

* 1. Задание № 2

1. Разработать программу на ассемблере, которая инвертирует все нечетные числа в памяти (5 вариант)
2. Провести прототипирование процессора и выполняемой программы
3. Сравнить данный вариант процессора с версией 01\_mmio (данный вариант – тот, что разработали мы)

Чтение и память данных осуществляется, начиная с нулевого адреса (их всего 32).

* + 1. Разработать программу на ассемблере, которая инвертирует все нечетные числа в памяти

Код программы находится в lab\_3\schoolMIPS\program\04\_task\_2\_2\_1\main.S.

Код программы для запуска в Mars находится в lab\_3\schoolMIPS\program\04\_task\_2\_2\_1\main\_for\_mars.S.



1. Память для теста программы в Mars.



1. Память после работы программы в Mars.

Как видно, программа инвертирует только нечетные значения в памяти.

* + 1. Провести прототипирование процессора и выполняемой программы

Данный пункт выполним только в лаборатории.

* + 1. Сравнить данный вариант процессора с версией 01\_mmio

Дополненная нами версия отличается лишь тем, что память данных является двухпортовой, и второй порт используется для внешнего чтения из памяти. Извне (с переключателей на плате (4..0) приходит адрес памяти для чтения, и на семисегментный дисплей выводится прочитанное из памяти слово.

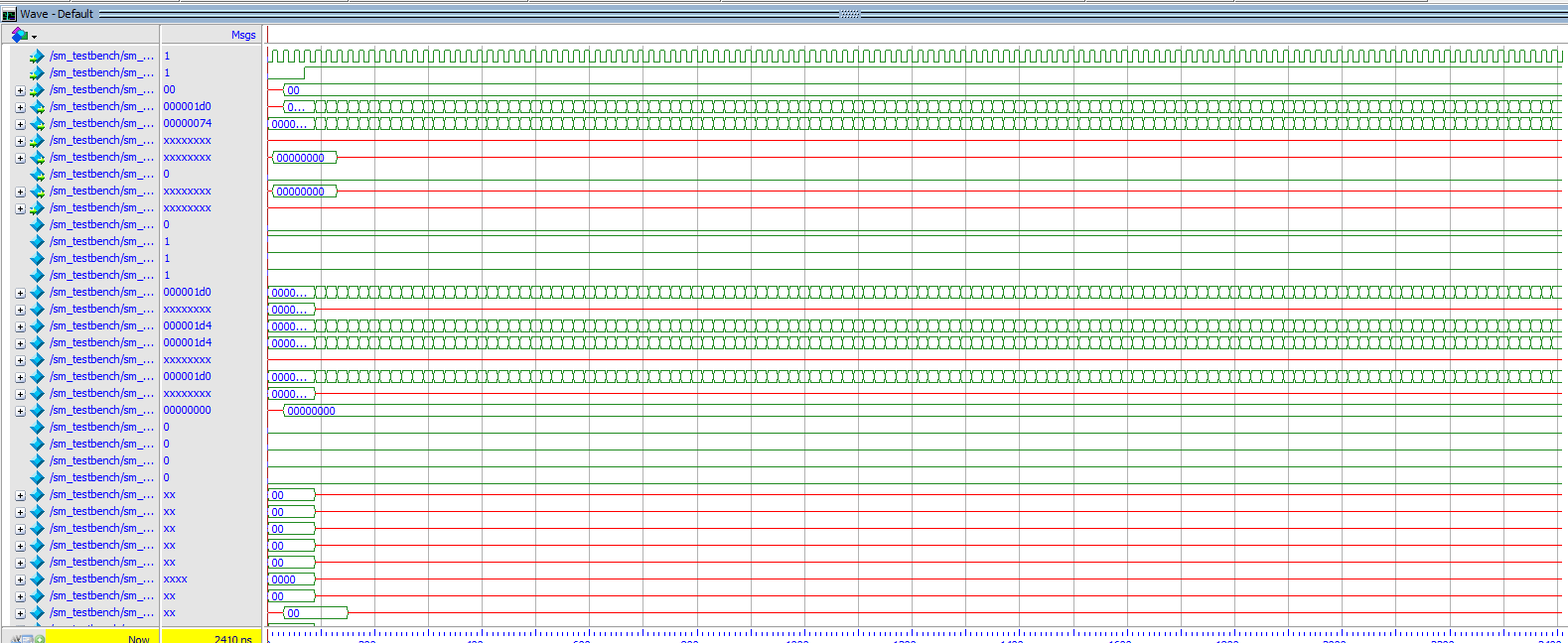
* 1. Задание № 3

1. Перейти в ветку проекта 03\_pipeline
2. Выполнить программы 00\_counter, 01\_fibonacci, 02\_sqrt, 03\_ram и убедиться, что они работают также, как в предыдущей лабораторной работе
3. Выполнить программы 06\_hz\_forward, 07\_hz\_stall, 08\_hz\_branc и добавить комментарии к программам
4. Объяснить, чем эта версия процессора отличается от базовой
   * 1. Перейти в ветку проекта 03\_pipeline

Просто клонируется проект.

* + 1. Выполнить программы 00\_counter, 01\_fibonacci, 02\_sqrt, 03\_ram

И убедиться, что они работают также, как в предыдущей лабораторной работе



1. Симуляция процессора с программой 00\_counter.

Остальные программы дают аналогичный результат, поэтому их симуляция в отчете не приводится.

* + 1. Выполнить программы 06\_hz\_forward, 07\_hz\_stall, 08\_hz\_branc

И добавить комментарии к программам.

06\_hz\_forward:

.text

init: li $t0, 1

start: addu $t1, $t0, $t0 # forward both args from M to E stage

addu $t2, $t0, $t0 # forward both args from W to E stage

addu $t3, $t0, $t0 # no forwarding

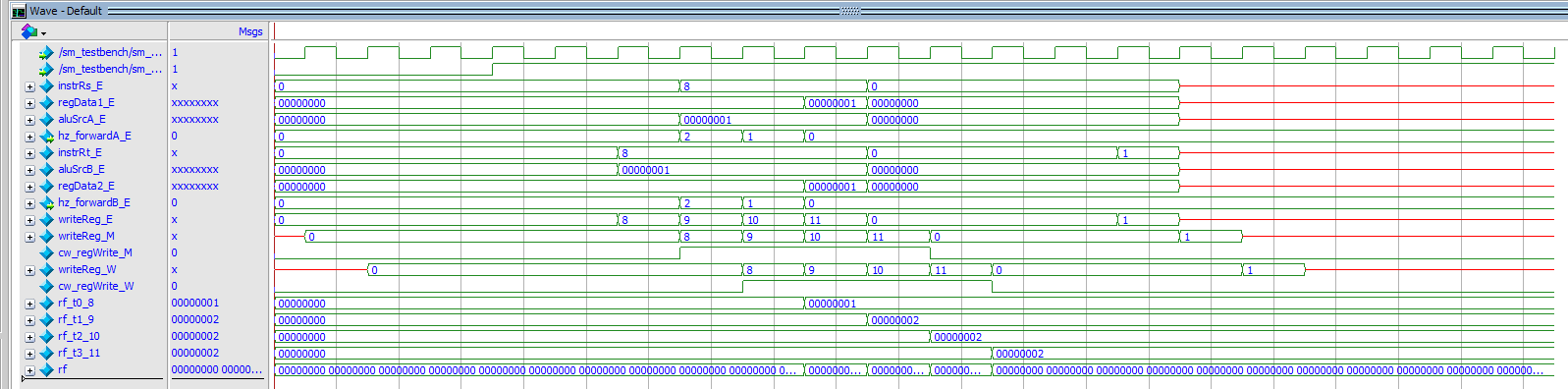
nop

nop

nop

nop

end: b end



1. Симуляция процессора с программой 06\_hz\_forward.

Сначала в $t0 записывается 1, потом в $t1, $t2, $t3 записывается $t0 + $t0, причем аргументы (то есть $t0) берутся с разных стадий, то есть в теории, в $t1, $t2 и $t3 могут быть записаны разные значения.

07\_hz\_stall:

.text

init: li $t0, 1

write: sw $t0, 0x0 ($0)

nop

nop

start: lw $t1, 0x0 ($0)

addu $v0, $t1, $t1 # pipleline should be stalled

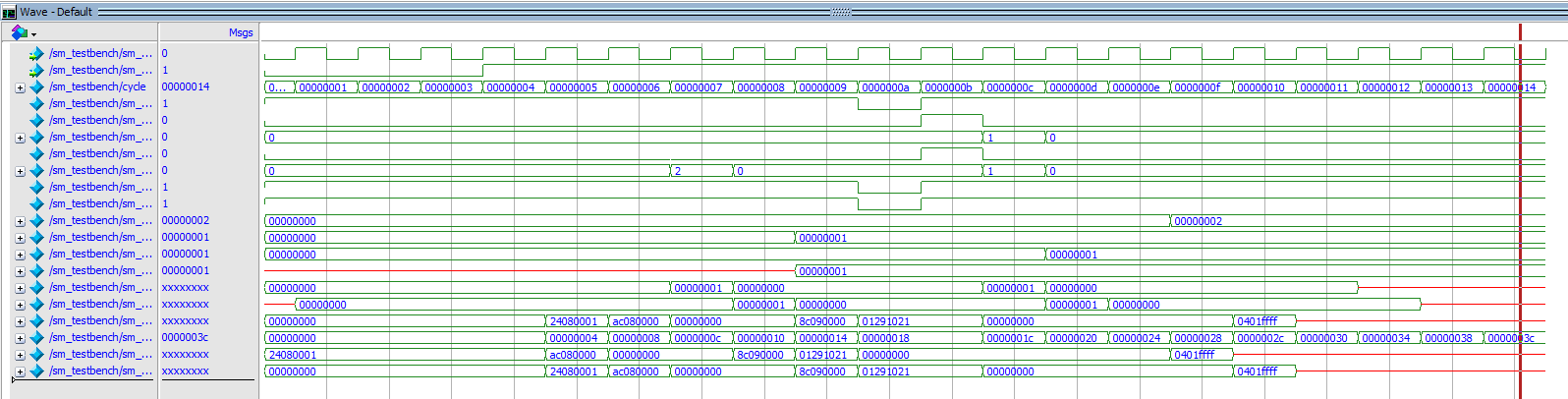
nop

nop

nop

nop

end: b end



1. Симуляция процессора с программой 07\_hz\_stall.

К тому моменту, как процессор начнет выполнять команду “addu $v0, $t1, $t1”, предыдущая команды еще не успеет выполниться, следовательно, операнды для выполнения этой команды будут неактуальными. Следует подождать 2 такта после lw прежде, чем выполнять addu.

08\_hz\_branch:

.text

init: li $t0, 1

sw $t0, 0x0 ($0)

check: bne $t0, $0, w1

addu $v0, $t0, $t0 # should be flushed from E-stage (with pipeline stall)

addu $v0, $t0, $t0 # unreacheble

addu $v0, $t0, $t0 # unreacheble

nop # unreacheble

w1: li $t1, 3

li $t2, 3

beq $t1, $t2, w2

addu $v0, $t0, $t0 # should be flushed from E-stage (with pipeline stall)

addu $v0, $t0, $t0 # unreacheble

addu $v0, $t0, $t0 # unreacheble

nop # unreacheble

w2: lw $t1, 0x0 ($0)

bne $t1, $t2, w3

addu $v0, $t0, $t0 # should be flushed from E-stage (with pipeline stall)

addu $v0, $t0, $t0 # unreacheble

addu $v0, $t0, $t0 # unreacheble

nop # unreacheble

w3: nop # v0 should be 0 here

nop

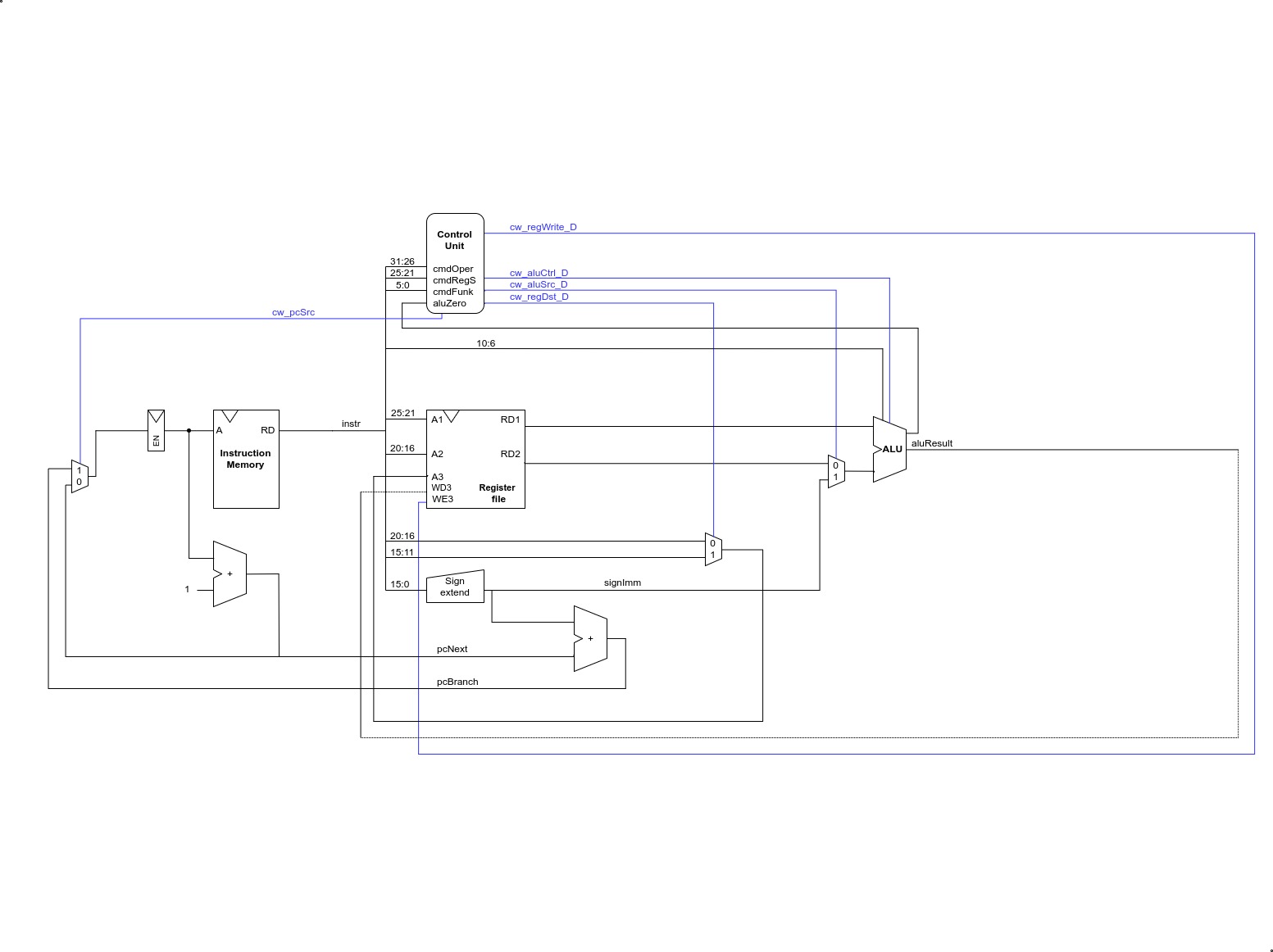
nop

nop

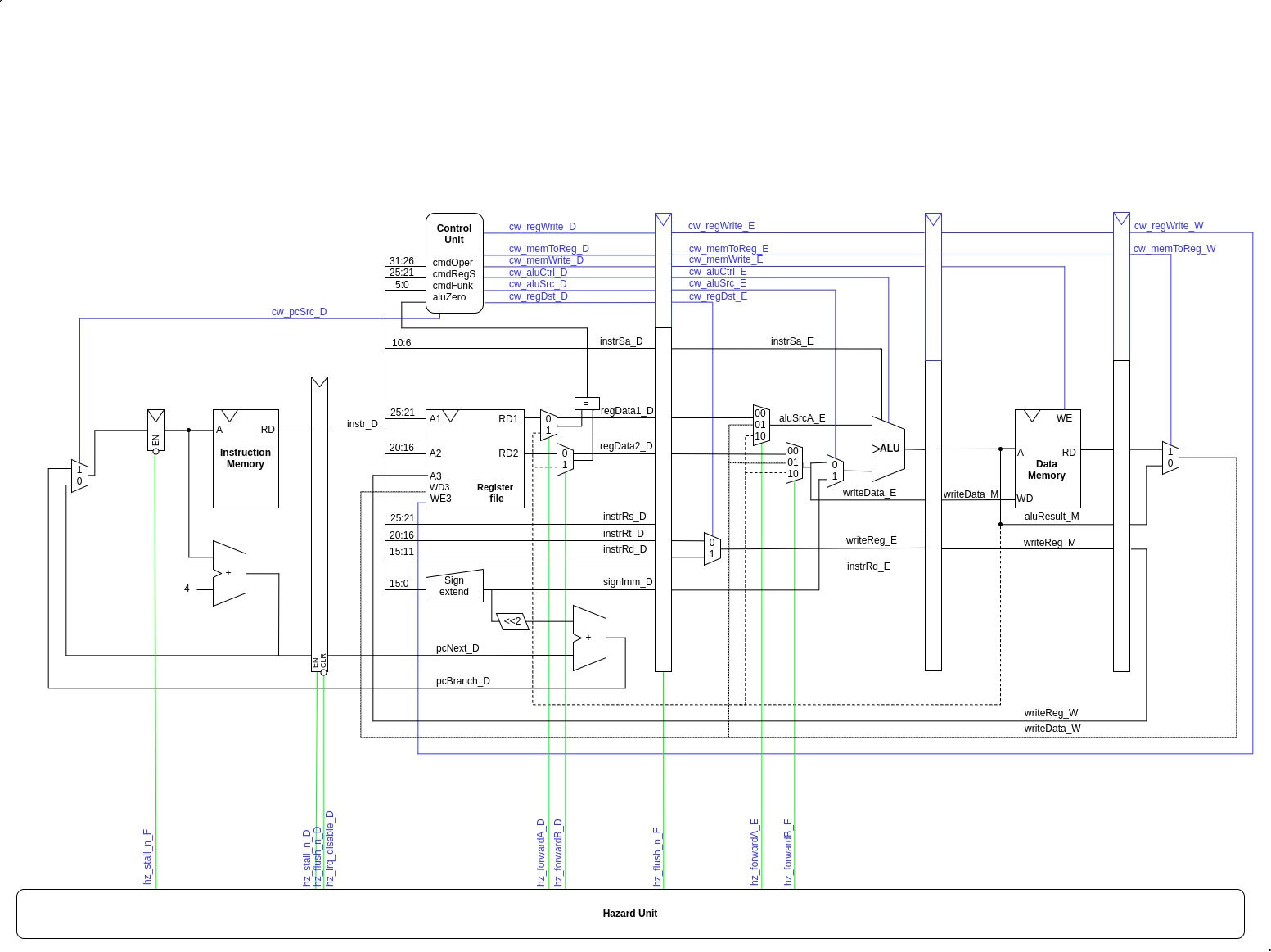
end: b end

В каждом блоке есть недостижимые инструкции, потому что процессор не успевает выполнить инструкции перед условным переходом. Если бы он успевал их выполнить (он был бы без конвейера, или там были бы еще инструкции, к примеру “nop”), то недостижимые в текущей ситуации инструкции были бы достижимыми.

* + 1. Объяснить, чем эта версия процессора отличается от базовой



1. Версия процессора 00\_simple.



1. Версия процессора 03\_pipeline.

03\_pipeline - the pipelined version of the simplest core with data memory.

В данной версии присутствует память данных, а также процессор является конвейерным, то есть следующая операция инструкции начинает выполняться до окончания выполнения текущей.

1. Выводы

Во время выполнения данной лабораторной работы мы вновь добавляли поддержку инструкций в процессор, а также RAM, которая необходима для выполнения работы данных инструкций. Разработали программу на ассемблере, которая использует данные инструкции, а также может подтвердить, в случае своего успешного выполнения, что предыдущие пункты лабораторной работы выполнены верно. Начали знакомство с конвейерной версией процессора schoolMIPS.

1. Список литературы
2. SPDS\_Lab\_2\_2020 [Электронный ресурс]. – URL: [https://docs.google.com/document/d/1WuYORvrBLINq-EuFRP1P-HyPyK4WiaPP\_O5PQ25Yehc/edit#heading=h.7wjwmyn0vwth](https://docs.google.com/document/d/1WuYORvrBLINq-EuFRP1P-HyPyK4WiaPP_O5PQ25Yehc/edit%23heading=h.7wjwmyn0vwth)
3. DE10-Standard User Manual [Текст] Terasic inc – 133c.
4. MIPSfpga/schoolMIPS Wiki [Электронный ресурс]. – URL: <https://github.com/MIPSfpga/schoolMIPS/wiki>
5. Цифровая схемотехника и архитектура компьютера [Текст]/ Дэвид М. Хэррис и Сара Л. Хэррис – 1627с.
6. Цифровой синтез [Текст]: практический курс / Панчул Ю., Романов А. / Романов А. – 500с.
7. Introduction to Intel® FPGA IP Cores [Электронный ресурс]. – URL: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_intro_to_megafunctions.pdf>
8. MIPS Instruction Reference [Электронный ресурс]. – URL: <http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>