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 № 2

Тема работы: «Разработка и программирование Soft-процессорных ядер с архитектурой однотактный MIPS. Часть 2»

по курсу «Системное проектирование цифровых устройств»

Выполнил:

Бригада № 10

Власов Р. В. БИВ174

Сегида Т. О. БИВ174

Принял

ассистент Американов А. А.

Оценка:

Москва 2020 г.

Оглавление

[1. Добавление периферии 3](#_Toc52025938)

[1.1. Семисегментный индикатор 3](#_Toc52025939)

[1.2. DIP-переключатель 3](#_Toc52025940)

[2. Самостоятельная работа 3](#_Toc52025941)

[2.1. Задание № 1 3](#_Toc52025942)

[2.1.1. Добавить в процессор 8-мибитный вход 4](#_Toc52025943)

[2.1.2. Добавить в систему команд процессора новую команду 4](#_Toc52025944)

[2.1.3. Реализовать дополнение байта 5](#_Toc52025945)

[2.1.4. Переработать 01\_fibonacci/ и 02\_sqrt/ 5](#_Toc52025946)

[2.1.5. Провести тестирование проекта на прототипе 6](#_Toc52025947)

[2.2. Задание № 2: 7](#_Toc52025948)

[2.2.2. Выполнить на плате программы 00\_counter, 01\_fibonacci, 02\_sqrt, 03\_ram 7](#_Toc52025949)

[2.2.3. Выполнить программы 04\_irq\_timer и 05\_exc\_ri 8](#_Toc52025950)

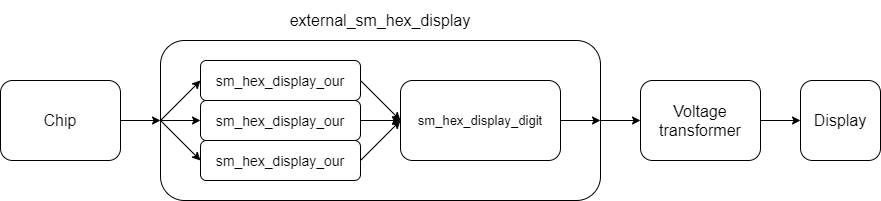
[2.2.4. Отличия 02\_irq от базовой версии процессора 11](#_Toc52025951)

[3. Выводы 12](#_Toc52025952)

[4. Список литературы 12](#_Toc52025953)

1. Добавление периферии
   1. Семисегментный индикатор

Руководствовались примерами кода из «Подключение семисегментного индикатора» из допматериалов. Встроенный семисегментный индикатор не отключали, поэтому внешний индикатор дублирует 3 младших разряда встроенного индикатора.



1. Схема подключения модулей для работы семисегментного индикатора.

Не забыть преобразователь логических уровней! GPIO выводы платы выдают 3.3В, в то время как для внешнего дисплея необходимо 2.2В (максимально допустимое значение == 2.5В).

Для управления индикатором используются GPIO[11:0].

* 1. DIP-переключатель

Изменения затронули только файл верхнего уровня – de10\_standard.v.

Для переключателя используются GPIO[35:28].

Для работы переключателя необходимо к одной стороне его контактов подключить GPIO вывод с напряжением питания 3.3В (VCC3P3), а к другой стороне контактов пины GPIO[35:28].

1. Самостоятельная работа

Вариант 1.

* 1. Задание № 1

1. Добавить в процессор 8-мибитный вход (к этому входу в файле верхнего уровня иерархии подключить DIP переключатель)
2. Добавить в систему команд процессора новую команду, которая будет загружать данные с нового входа в заданный регистр
3. Реализовать дополнение байта (8 бит) до 32-х бит нулями
4. Переработать 01\_fibonacci/ и 02\_sqrt/, чтобы загружать значение извне и выводить результат на внешнюю периферию платы
5. Провести тестирование проекта на прототипе

В данном задании используется версия процессора 00\_simple.

* + 1. Добавить в процессор 8-мибитный вход

К этому входу в файле верхнего уровня иерархии подключить DIP переключатель.

Вход – dip\_sw, добавлен в код ядра процессора (sm\_cpu) и в файл верхнего уровня процессора (sm\_top).

Строка № 34 файла de10\_standard.v:

wire [7:0] dip\_sw = GPIO [35:28];

* + 1. Добавить в систему команд процессора новую команду

Которая будет загружать данные с нового входа в заданный регистр

Так как для выполнения инструкции требуется указать лишь регистр, в который нужно записать данные, и opcode, она будет выглядеть следующим образом:



1. Вид инструкции записи данных из DIP переключателя в регистр.

rd – register destination.

Так как необходимая инструкция не вписывается в систему R, I, J типов, opcode придется придумать самостоятельно.

Так как эта инструкция наиболее похожа на инструкцию работы с памятью[[1]](#footnote-1) по своему смыслу, opcode было принято взять по образу и подобию opcode этих инструкций. Первая цифра этого поля у всех инструкций ‘1’, в отличие от opcode остальных инструкций, поэтому первая цифра тоже будет ‘1’. Вторая цифра у всех команд работы с памятью ‘0’, этого знать достаточно, чтобы команда получила уникальный opcode, который будет следующим:

110000

Этот opcode добавляем в файл sm\_cpu.vh (есть ярлык в lab\_2/attachments/) под названием C\_RSW – read switch.



1. Пример инструкции.

То, что считается в АЛУ попадает в шину AluResult, берущую начало в АЛУ (ALU) и далее попадает в порт WD3 регистрового файла (Register file) (стр. 946 H&H). В нашем случае будем читать данные из DIP переключателя и отправлять в тот же порт WD3. Различие состоит в том, что переключатель подразумевается 8-мибитным, а регистр имеет 32 бита, но это можно обойти, дополнив нулями число, что будет сделано в следующем пункте.

dataToRegDst – добавленный управляющий сигнал, который определяет откуда данные записываются в регистр. При всех прошлых инструкциях выставляется “0”, при добавленной выставляется “1”. Управляет мультиплексором на шине AluResult, ведущим в WD3 порт регистрового файла.

regWrite – присутствующий управляющий сигнал, разрешает запись в регистр, должен быть выставлен в “1” для добавленной команды.

regDst – присутствующий управляющий сигнал, определяет откуда из команды брать адрес регистра, в который осуществляется запись, по умолчанию равен “0”, необходимо, чтобы “0” и был равен.

* + 1. Реализовать дополнение байта

(8 бит) до 32-х бит нулями.

Так как операция дополнения нулями нигде не используется, а единственное место, где ее целесообразно использовать – это дополнение данных с DIP переключателя перед записью в регистр, туда это и будет вставлено.

Так выглядит код, который выполняет дополнение нулями (дополняет 24 нулями спереди) (стр. 72 sm\_cpu):

wire [31:0] zeroExtDipSw = { {24 { 1'b0 }}, dip\_sw[7:0] };

* + 1. Переработать 01\_fibonacci/ и 02\_sqrt/

Чтобы загружать значение извне и выводить результат на внешнюю периферию платы.

01\_fibonacci/

Первое число ряда – “0” (регистр t0), в исходном коде программы:

.text

start: move $t0, $0

li $t1, 1

move $v0, $t1

fibonacci: addu $t0, $t0, $t1

move $v0, $t0

addu $t1, $t0, $t1

move $v0, $t1

beqz $0, fibonacci

Было принято решение, что первое число ряда будет браться из DIP переключателя. То есть, в дампе памяти необходимо будет заменить первую инструкцию (move) на ту, что была добавлена нами. t0 – регистр под номером 8 == 01000.

Инструкция примет вид – 11000000000010000000000000000000, что в 16-ричном формате выглядит как C0080000.

Код в 03\_fibonacci\_reworked/ уже с замененной инструкцией.

Для вывода результатов работы программы на внешний семисегментный индикатор не нужно предпринимать никаких дополнительных действий, кроме как набрать на переключателях на самой плате номер регистра v0 == 2== 00010.

Warning (10850): Verilog HDL warning at sm\_rom.v(14): number of words (8) in memory file does not match the number of elements in the address range [0:63]

При компиляции Quartus выдает предупреждение, что в файле памяти строк всего 8, а в памяти в коде под это выделено 64 строки, возможно, программа не выполнялась из-за этого в первой лабораторной. В данной программе (03\_fibonacci\_reworked/) файл памяти дополнен до 64 строк. После дополнения файла памяти при повторной компиляции предупреждение перестало выводиться.

02\_sqrt/

В коде программы вычисляется квадратный корень 82. Было принято решение, что число, квадратный корень которого нужно вычислять, будет задаваться DIP переключателем, для чего, его перед началом работы программы необходимо установить в требуемое положение.

Так как число для вычисления квадратного корня сохраняется в регистре a0 (4 == 00100), код инструкции изменится и примет вид – 11000000000001000000000000000000, что в 16-ричном формате == C0040000.

В данной программе также не требуется предпринимать дополнительных действий для вывода результатов работы на внешний семисегментный индикатор, кроме как набрать на переключателях на самой плате номер регистра v0 == 2== 00010.

Файл памяти данной программы также дополнен до 64 строк. Скрипты для компиляции из этих программ убраны, чтобы случайно не переписать файлы памяти. Оставлены только скрипты, чтобы скопировать файлы памяти в указанное в проекте место.

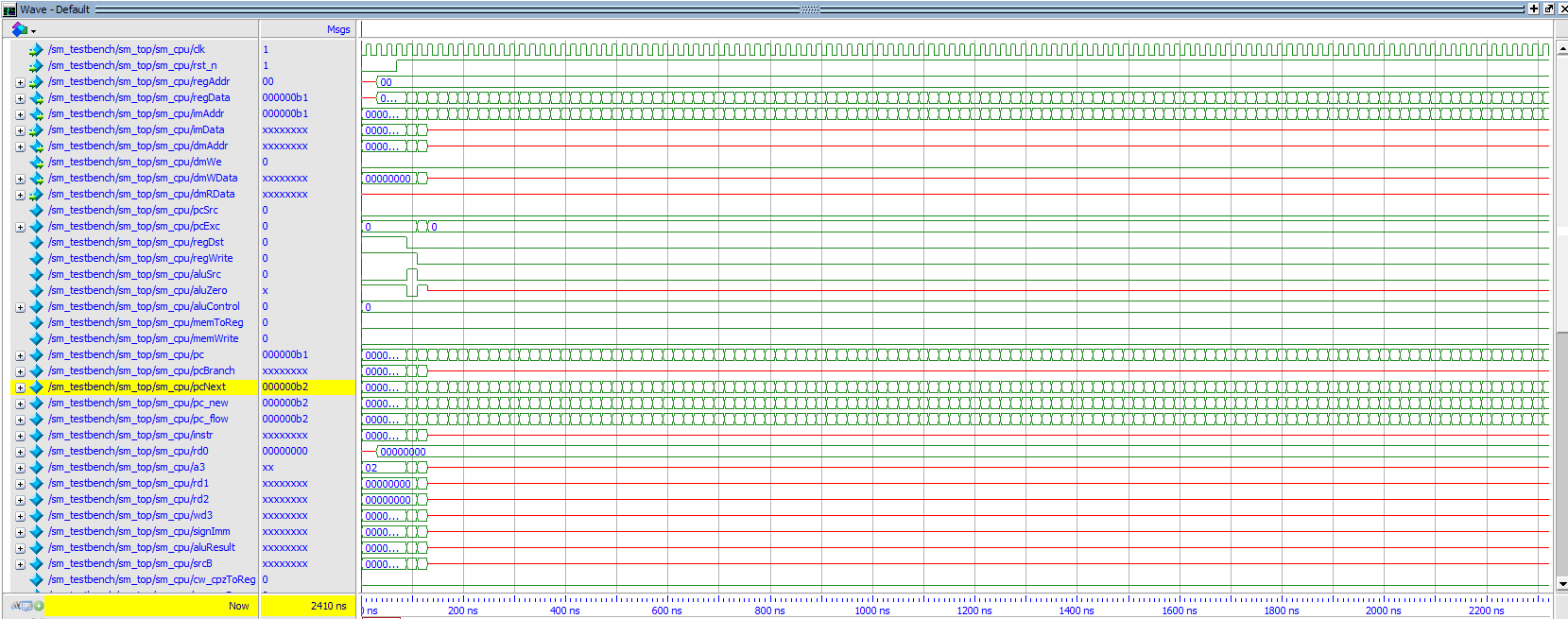
Обе программы компилируется без ошибок и предупреждений, которые могли бы относится к изменениям в коде, внесенным нами, а также файлу памяти, за исключением предупреждений о GPIO выводах, которые объявлены в файле верхнего уровня иерархии как двунаправленные (inout).

* + 1. Провести тестирование проекта на прототипе

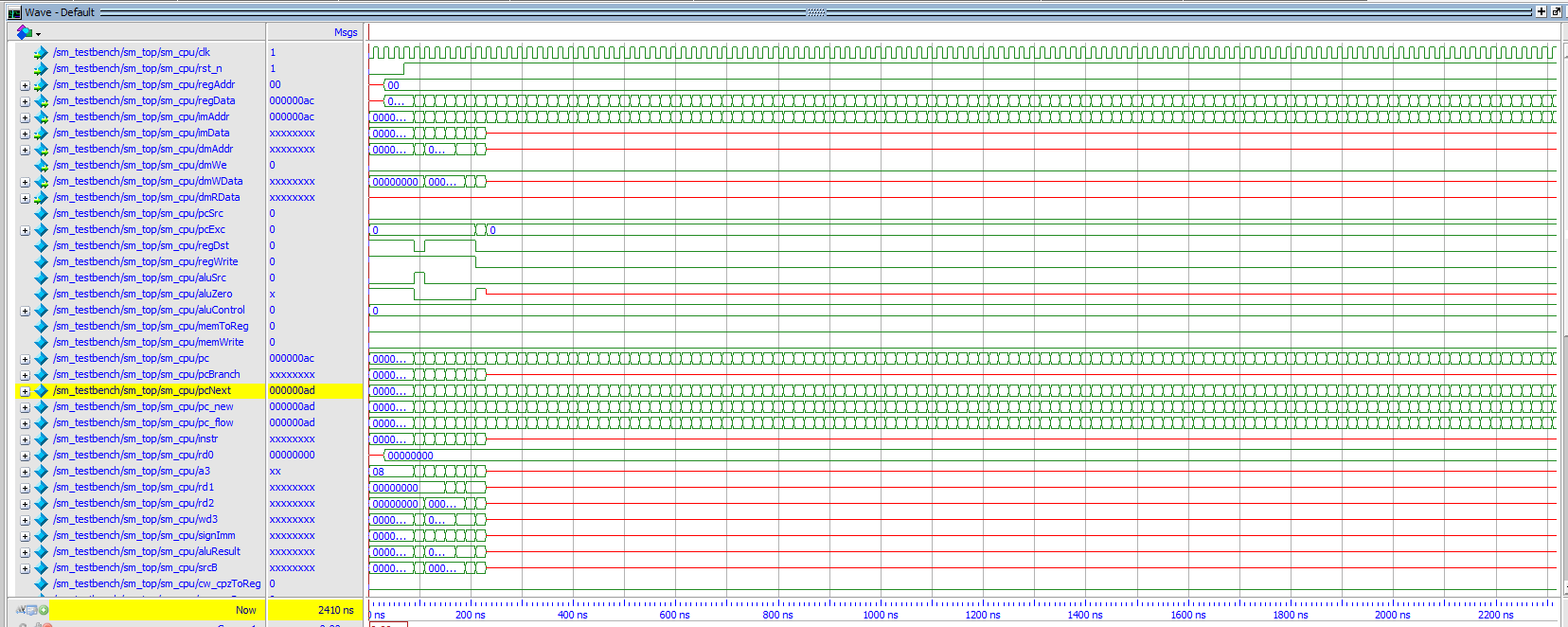
Данный этап доступен только в лаборатории, так как необходима такая периферия, как DIP переключатель, внешний семисегментный индикатор и преобразователь логических уровней (для индикатора с 3.3В в 2.2В).

* 1. Задание № 2:

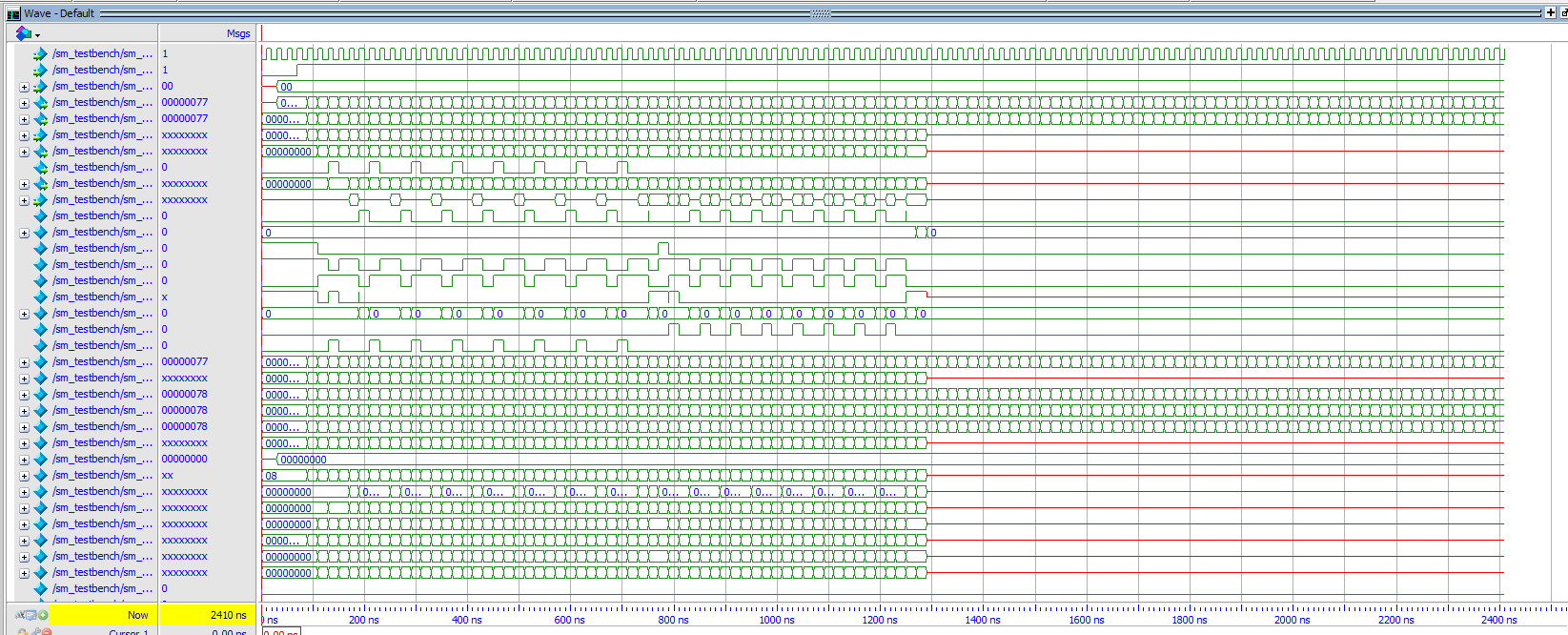
1. Перейти в ветку проекта schoolMIPS 02\_irq: <https://github.com/MIPSfpga/schoolMIPS/tree/02_irq/>
2. Выполнить на плате программы 00\_counter, 01\_fibonacci, 02\_sqrt, 03\_ram и убедиться, что они работают также, как в предыдущей лабораторной работе
3. Выполнить программы 04\_irq\_timer и 05\_exc\_ri и добавить комментарии к программам
4. Объяснить, чем эта версия процессора отличается от базовой
   * 1. Выполнить на плате программы 00\_counter, 01\_fibonacci, 02\_sqrt, 03\_ram



1. Моделирование в ModelSim процессора с программой 00\_counter.



1. Моделирование в ModelSim процессора с программой 01\_fibonacci.



1. Моделирование в ModelSim процессора с программой 03\_ram.
   * 1. Выполнить программы 04\_irq\_timer и 05\_exc\_ri

И добавить комментарии к программам

Компилировалось скриптом из проекта процессора версии 2017 года, так как в современной версии Mars не распознавал директивы.

Сопроцессор 0 отвечает за обработку исключений (6.7.2 H&H).

.set, .text, .org – директивы ассемблера.

The mtc0 (Move To Coprocessor 0) instruction moves a value from a general-purpose register to a Coprocessor 0 register.

Example:

mtc0 $v0, $12 # Copy $v0 to $12 (status) in coprocessor 0.

1. ADDIU

Add immediate unsigned (no overflow)

|  |  |
| --- | --- |
| Description: | Adds a register and a sign-extended immediate value and stores the result in a register |
| Operation: | $t = $s + imm; advance\_pc (4); |
| Syntax: | addiu $t, $s, imm |
| Encoding: | 0010 01ss ssst tttt iiii iiii iiii iiii |

Код программы 04\_irq\_timer с небольшими комментариями:

.set TimerPeriod, 0x31 # объявление переменной

.text

.org 0x0 # устанавливается, что адресация кода начинается с 0x0[[2]](#footnote-2)

init: li $t0, TimerPeriod # timer init[[3]](#footnote-3)

mtc0 $t0, $11 # cp0.Compare = 0x20

mtc0 $0, $9 # cp0.Count = 0;

addiu $t0, $0, 0x8001 # exceptions init 8001

mtc0 $t0, $12 # cp0.Status = {IE = 1, IM7 = 1}

li $v0, 0x0 # reset bug detecting value

end: addiu $t1, $0, 0x1 # в t1 записывается 0х1

addu $v0, $v0, $t1 # к v0 прибавляется 1

subu $v0, $v0, $t1 # из v0 вычитается 1

addu $v0, $v0, $t1

subu $v0, $v0, $t1

addu $v0, $v0, $t1

subu $v0, $v0, $t1

addu $v0, $v0, $t1

subu $v0, $v0, $t1

b end # while(1);[[4]](#footnote-4)

.org 0x100 # устанавливается, что адресация кода начинается с 0x100

exception: li $t0, TimerPeriod # timer reset

mtc0 $t0, $11 # cp0.Compare = 0x20

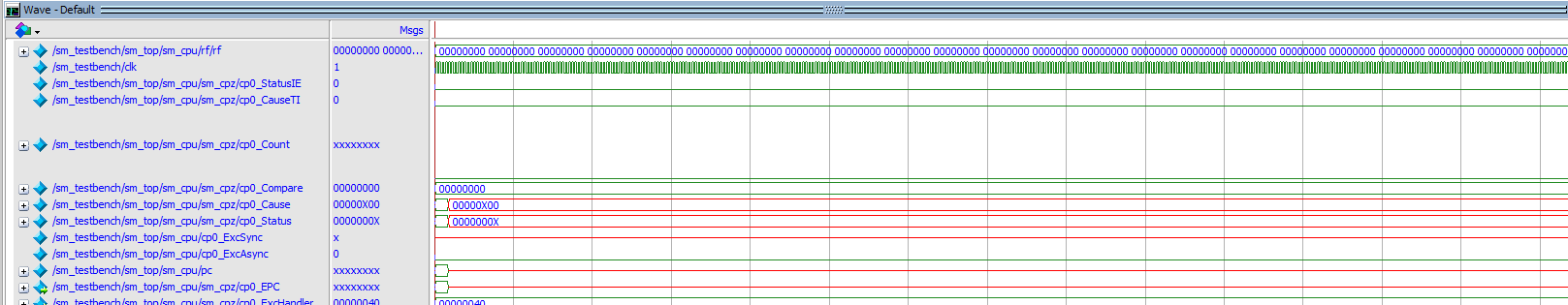
mtc0 $0, $9 # cp0.Count = 0;

# nop

# nop

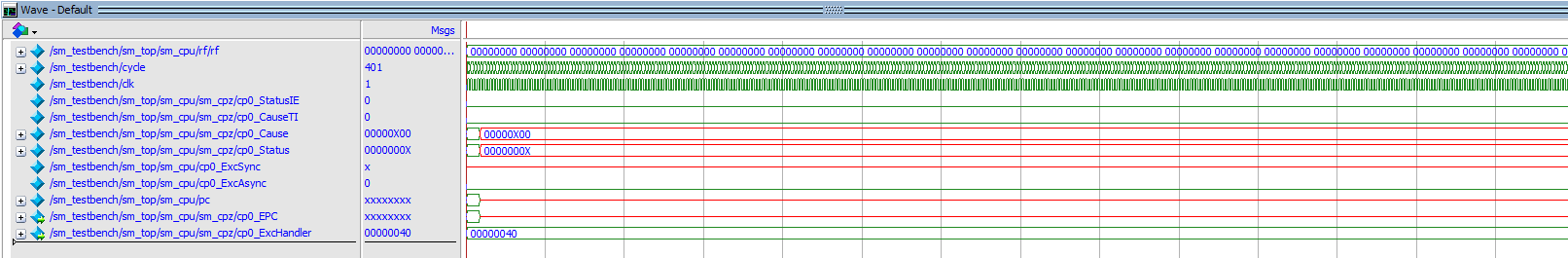
# nop

eret # eret[[5]](#footnote-5)



1. Моделирование работы программы 04\_irq\_timer.

irq – interrupt request. Наверное подразумевается, что этот код ждет (блок end) пока его прервут, после прерывания отрабатывает код блока exception, а далее выполнение продолжается с места прерывания (eret).



1. Моделирование работы программы 05\_exc\_ri.

Код программы 05\_exc\_ri с небольшими комментариями:

#define c0\_status $12 # define[[6]](#footnote-6)

#define c0\_EPC $14

#define c0\_cause $13

.text

.org 0x0

init: addiu $t0, $0, 0x1

li $v0, 0x0

nop

nop

addu $v0, $v0, $t0

subu $v0, $v0, $t0

mfhi $0 # reserved instruction[[7]](#footnote-7)

addu $v0, $v0, $t0

subu $v0, $v0, $t0

nop

nop

b init # while(1);

.org 0x100

exception: move $t1, $v0 # save v0

mfc0 $v0, c0\_status # load cp0.Status

mfc0 $v0, c0\_cause # load cp0.Cause

mfc0 $v0, c0\_EPC # load cp0.EPC

addiu $v0, $v0, 0x1 # increment EPC to skip error

mtc0 $v0, c0\_EPC # save cp0.EPC

move $v0, $t1 # restore v0

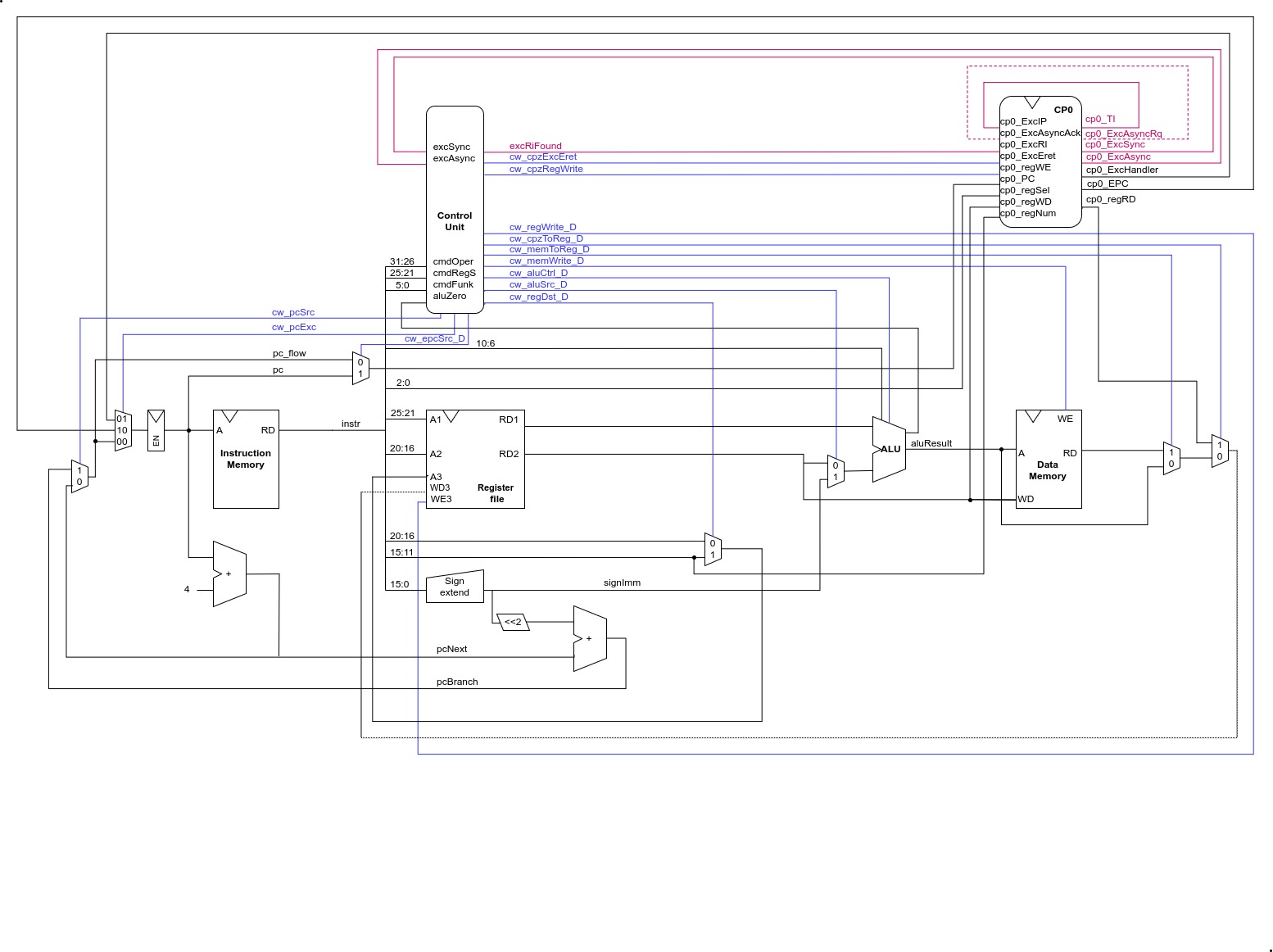
nop

nop

eret

exc – exception. Блок init работает в цикле, но, наверное (а может и нет), mfhi вызывает исключение, так как ни деления, ни умножения не было произведено. Бросается исключения, отрабатывает блок exception, и работа программы продолжается со следующей инструкции (addiu $v0, $v0, 0x1 # increment EPC to skip error), чтобы избежать повторного вызова исключения.

* + 1. Отличия 02\_irq от базовой версии процессора



1. Схема процессора версии 02\_irq.

Его отличия от базовой версии – data memory, system timer, interrupts and exceptions (CP0 coprocessor).

Data memory – в базовой версии памяти данных нет, есть только память инструкций.

System timer – присутствует регистр системного времени.

Interrupts and exceptions – добавлена поддержка прерываний и исключений.

1. Выводы

Во время выполнения данной лабораторной работы мы познакомились с версией 02\_irq процессора schoolMIPS, которая вводит поддержку памяти данных, системного времени, а также исключений и прерываний. Получили навыки расширения функциональности процессора добавлением команд. Получили навыки работы с платой DE10-Standard и работы данной платы с периферийными устройствами.

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. MIPSfpga/schoolMIPS Wiki [Электронный ресурс]. – URL: <https://github.com/MIPSfpga/schoolMIPS/wiki>
4. DE10-Standard User Manual [Текст] Terasic inc – 133c.
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. Использование GPIO выводов платы De1-SoC [Электронный ресурс]. – URL: <https://drive.google.com/file/d/0B2DyhVuZZ3BFWFU2YXJjOEFRTVU/view>
8. 14.22mm (0.56INCH) THREE DIGIT NUMERIC DISPLAY [Текст] Kingbright – 4c.
9. \_О семисегментном индикаторе (Подключение семисегментного индикатора) [Электронный ресурс]. – URL:  
   <https://drive.google.com/drive/folders/0B2DyhVuZZ3BFYzFpdHNZMzFqUjg>
10. Хабр Q&A [Электронный ресурс]. – URL: <https://qna.habr.com/q/70174>
11. Mips coprocessor 0 [Электронный ресурс]. – URL: <http://www.it.uu.se/education/course/homepage/os/vt18/module-1/mips-coprocessor-0/>
12. Plasma - most MIPS I(TM) opcodes [Электронный ресурс]. – URL: <https://opencores.org/projects/plasma/opcodes>

1. https://opencores.org/projects/plasma/opcodes [↑](#footnote-ref-1)
2. <https://qna.habr.com/q/70174> [↑](#footnote-ref-2)
3. The li instruction loads a specific numeric value into that register. [↑](#footnote-ref-3)
4. b == jump [↑](#footnote-ref-4)
5. ERET is the R10000 instruction for returning from an interrupt, exception, or error trap. Unlike a branch or jump instruction, ERET does not execute the next instruction. [↑](#footnote-ref-5)
6. Вроде бы тоже объявление переменной [↑](#footnote-ref-6)
7. mfhi и mflo – загружают результаты умножения в регистры общего назначения (hi загружает старшие биты, lo – нижние) [↑](#footnote-ref-7)