Лабораторная работа 2

1. Цель

Ознакомиться с возможностями настройки архитектуры системы команд, обработкой исключений/прерываний и принципами работы сборщика программ.

2. Задание

1. Необходимо добавить копию репозитория SCR1 https://github.com/syntacore/scr1 в ваш личный аккаунт на GitHub (который вы заводили для предыдущих лабораторных). Это будет ваш рабочий репозиторий, в котором вы сможете делать любые изменения с проектом. Копирование производится командой fork https://help.github.com/en/articles/fork-a-repo .
2. В репозитории создать ветку с именем lab_scr1_sim. В этой ветке вы будете коммитить все изменения проекта для данной лабораторной. Убедитесь, что проект собирается и все тесты проходят в симуляции до внесения изменений в проект. В качестве программы-симулятора мы рекомендуем использовать Verilator, но по желанию вы можете использовать любую из поддерживаемого списка.
3. Отредактировать список тестов, чтобы на выполнение остался только тест по варианту задания. Как выбрать отдельные тесты для симуляции описано в SCR1 User Manual, раздел 5.2. Test subset. Запустить симуляцию для Verilator и убедиться, что проходит только выбранный тест.
4. В соответствии с вариантом задания модифицировать обработку исключений trap_vector в файле ./sim/tests/common/riscv_macros.h
5. Установить в файле ./src/includes/scr1_arch_description.svh параметры ядра Reset Vector и Trap Vector в соответствии с вариантом задания.
6. Изменить linker-скрипт ./sim/tests/common/link.ld и участвующие в сборке файлы программы для корректного запуска теста с новыми значениями Reset Vector и Trap Vector.

7. Сохранить в директории ./results: результат симуляции (test_results.txt), дизассемблер теста (*.dump), трейс лог MPRF (trace_mprf_diff_.log), трейс лог CSR (trace_csr_.log).
8. Создать в директории ./lab_scr1_sim файл с отчетом по проделанной README.md.
9. Закоммитить все произведенные изменения в ветке lab_scr1_sim.

3. Варианты заданий

Номер варианта	ФИО	Вид исключения	Тест	Reset Vecotor	Trap Vector	Обработчик
1	Баранец Максим	Illegal instruction	isa/ rv32mi/ illegal.S	0x400	0x200	Вывод строки «illegal»
2	Криворотова Полина	Breakpoint	isa/ rv32mi/ sbreak.S	0x1000	0x840	Вывод строки «BP»
3	Стручинский Артем	Illegal instruction	isa/ rv32mi/ illegal.S	0x2000	0x1880	Вывод строки «illexc»
4	Бондаренко Михаил	Breakpoint	isa/ rv32mi/ sbreak.S	0x800	0x6c0	Вывод строки «break»
5	Смирнов Александр	Instruction address misaligned	isa/ rv32mi/ ma_fetch.S	0xA000	0x8c0	Вывод строки «misalign»
6	Мирошниченко Владислав	Environment call from M-mode	isa/ rv32mi/ scall.S	0xB000	0xA880	Вывод строки «envcall»
7	Тищук Богдана	Индивидуальное задание	-----	-----	-----	-----

4. Пример выполнения

Пример выполнения доступен в https://github.com/v-crys/scr1. Смотрите ветку lab_scr1_sim. Полезно посмотреть изменения по коммиту.

Задача: Обработать исключение IllegalInstruction выводом строки "My illegal instructin is detected". Настроить ресет вектор и вектор обработки прерываний на 0x1000 и 0x0500 соответственно. Проверить работу программы на примере isa/rv32mi/illegal.S.

```
cd <WORK_LIB>
git clone https://github.com/<Ваш github ник>/scr1 # клонирование
cd ./scr1 # переходим в папку с проектом
git submodule update --init --recursive # обновляем сабрепо с тестами
git checkout -b lab_scr1_sim # создаем ветку для выполнения лабораторной
 и переходим в нее

code . # Выполняем лабораторную работу в visual studio code

git add . # индексируем все измения
git commit -am "add lab" # создаем коммит
git push origin lab_scr1_sim # пушим коммит на сервер
```

В Visual Studio Code делаем следующее:

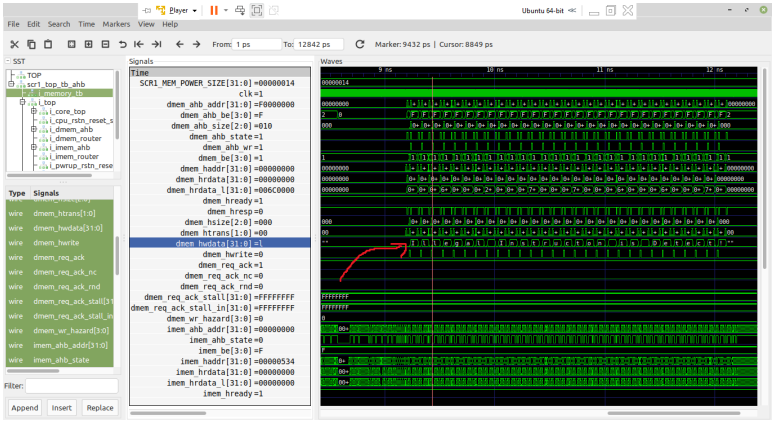
- Создаем дирректорию lab_scr1_sim и файл README.md в ней. В файле описываем поставленную задачу.
- Модифицируем список тестов в файле …/scr1/sim/tests/riscv_isa/rv32_tests.inc
- В терминале прописываем путь до тулчейна (с лабораторной 1) PATH=…/riscv-tools/riscv-gcc-10.2.0-gbbc9263-210318T1412/bin:$PATH
- Пробуем запустить симуляцию: make TARGETS="riscv_isa"
- Модифицируем вектора прирываний и ресетов в процессоре. …/scr1/src/includes/scr1_arch_description.svh меняем SCR1_ARCH_RST_VECTOR и SCR1_ARCH_MTVEC_BASE
- Модифицируем скрипты линковщика для правильной сборки проекта.
    - Для этого заходим в файл sim/tests/common/link.ld и корректируем в нем смещение кода

- в файле sim/tests/common/riscv_macros.h также корректируем смещение
- в случае, если направильно настроили, то тест перестанет проходить. Для отладки полезно открывать dump и в нем смотреть адреса, по которым расположено начало программа и trap вектор scr1/build/verilator_AHB_MAX_imc_IPIC_1_TCM_1_VIRQ_1_TRACE_0/illegal.dump
- **Между перезапусками не забываем выполнять make clean**

- После того, как правильно настроили вектора модифицируем обработчик прерывания

После завершения работы необходимо установить sudo apt install gtkwave. Собрать проект в режиме генерации wave форм: make run_verilator_wf TARGETS="riscv_isa" TRACE=1. После этого можно открыть wave форму в дирректории scr1/build/verilator_wf_AHB_MAX_imc_IPIC_1_TCM_1_VIRQ_1_TRACE_0 командой gtkwave ./simx.vcd. Также будет создан трейслог. Набор файлов с логами как описано в задаче поместить в папку lab_scr1_sim.

Как можем видеть, в память по определенному адресу была записано предложение. Тестовое окружение отловило запись по этому адресу и вывело результат в консоль:



Удалить папку build. Закоммитить изменения.

5. Дополнительная информация

5.1. Настройка симуляции SCR1 для Verilator

Задание выполнять на своей личной рабочей машине с Unix-подобной системой (в т.ч. это может быть виртуальная машина). На рабочей машине необходимо установить следующее программное обеспечение:

- GNU make версии 4.0 или выше https://www.gnu.org/software/make/
- Открытый симулятор Verilator версии 4.0 или выше https://www.veripool.org/wiki/verilator (**Необходимо собрать из исходников самый последний, apt install не использовать**)
- RISC-V GNU toolchain https://syntacore.com/page/products/sw-tools
- Клонировать ваш рабочий репозиторий SCR1 на рабочую машину
- Ознакомиться с описанием ./README.md раздел «Simulation quick start guide», выполнить описанную в нем настройку окружения и запуск симуляции для Verilator:
  - настроить пути для RISC-V toolchain
  - настроить пути для Verilator
  - клонировать RISC-V ISA, RISC-V Compliance и Coremark тесты и настроить переменные среды,
  - запустить симуляцию для Verilator и убедиться, что тесты успешно проходят.
  - Между перезапусками рекомендуется делать очистку директории build, используя команду make clean
- Добавим вывод сообщения по обработке события

5.2. Документация на SCR1

- SCR1 User Manual https://github.com/syntacore/scr1/blob/master/docs/scr1_um.pdf
- SCR1 External Architecture Specification https://github.com/syntacore/scr1/blob/master/docs/scr1_eas.pdf

5.3. Git

Полная документация по работе с Git на русском языке https://git-scm.com/book/ru/v2

Пример работы с гитом (создание репо и загрузка его на сервер):

```
sudo apt-get install git
git config --global user.name "FIRST_NAME LAST_NAME"
git config --global user.email "EMAIL"

cd work_dir
git init
git add .
git commit -am "my first commit"
git remote add origin http....<link for repo>
git push origin master
```

5.4. RISC-v

Спецификация The RISC-V Instruction Set Manual (Unprivileged), доступную по ссылке: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

Руководство по ассемблеру RISC-V Assembly Programmer’s Manual находится по ссылке: https://github.com/riscv/riscv-asm-manual/blob/master/riscv-asm.md

5.5. Диррективы ассемблера

Директива	Аргументы	Описание
.text		секция .text (секция кода)
.data		секция .data (секция данных)
.string	"string"	строка
.asciiz	"string"	строка (алиас для .string)
.byte	expression [,expression]*	8-битные данные

Директива	Аргументы	Описание
.2byte	expression [,expression]*	16-битные данные
.half	expression [,expression]*	16-битные данные
.short	expression [,expression]*	16-битные данные
.4byte	expression [,expression]*	32-битные данные
.word	expression [,expression]*	32-битные данные
.long	expression [,expression]*	32-битные данные
.zero		0

5.6. Метки

Символьные:

```
loop:
j loop
```

Переход на метку назад (backward):

```
1:
j 1b
```

Переход на метку вперед (forward):

```
j 1f
1:
```

Пример загрузки адреса:

```
.data
```

```
a: .word 1
.text
main:
la a0, a
```

5.7. Linker Scripts

Документация по linker-скриптам: https://sourceware.org/binutils/docs/ld/Scripts.html