****

**Lab YP4**

**Введение в прерывания**



**MIPSfpga 2.0. Lab YP4 – Введение в прерывания**

**1. Введение**

В данной лабораторной работе рассмотрены основы использования прерываний в микропроцессоре MIPS. В ней также показано, как с помощью прерываний освободить микропроцессор от постоянного опроса портов ввода-вывода. Это также позволяет ядру процессора увеличить количество циклов, которые можно использовать для вычислений и других операций отличных от ввода-вывода. В работе также показан пример, который может быть использован в дальнейшем в качестве основы для целого ряда задач, связанных с изучением особенностей работы процессора MIPS с прерываниями. Другая группа последующих задач может быть связана с использованием прерываний реализации параллельного выполнения программ, а также переключении контекста в операционной системе. Также в конце этой лабораторной работы перечислен ряд задач для последующего изучения обработки прерываний.

Эта лабораторная работа может также быть скомбинирована с предыдущей работой *MIPSfpga 2.0. Lab YP3 – Интегрированная периферия: пример использования датчика освещенности,* для создания студенческого проекта, демонстрирующего процедуру ввода-вывода, основанную на использовании прерываний и сравнения этого подхода для опроса портов ввода-вывода без использования прерываний.

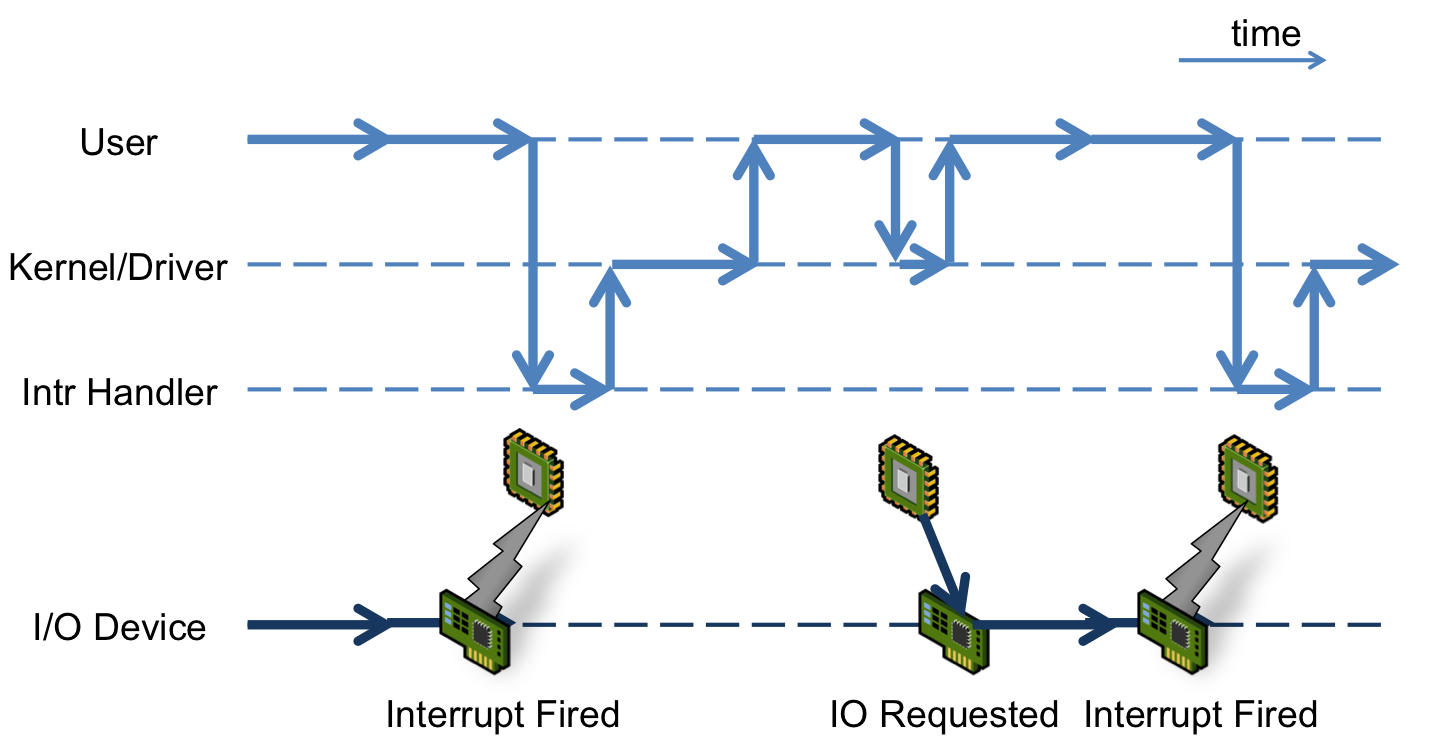
**2. Основные теоретические положения**

Прерывание – это одна из ключевых концепций в программировании и разработке компьютерных систем. Прерывание – это такое свойство микропроцессора, которое позволяет приостанавливать последовательное выполнение команд процессора для перехода по определённому адресу в программе в ответ на некоторое внешнее воздействие. Таким внешним воздействием обычно является изменение сигнала снаружи процессора. Адрес в памяти (или, в некоторых процессорах, индекс в массиве ячеек памяти), по которому происходит этот переход, называется вектором прерывания. Часть кода, которой передается контроль после вызова прерывания, называется обработчиком прерывания (ISR – Interrupt Service Routine). После того как обработчик прерывания выполнил необходимые действия контроль передается назад основной программе, которая выполнялась до поступления прерывания.

Внешним событием, вызывающим прерывание, может быть, например, тик таймера. Такие прерывания от таймера полезны для организации переключения задач в системе, где несколько задач или программ, разделяют процессорное время одного и того же процессора, который переключается между ними. Другим примером внешнего события, вызывающего прерывание, является сигнал о завершении операции ввода-вывода. Прерывания ввода-вывода могут быть использованы для разгрузки микропроцессора от постоянного опроса регистров ввода/вывода и проверки их статуса, как показано на **Рисунке 1**.

**Рисунок 1. Выполнение прерывания ввода-вывода.**

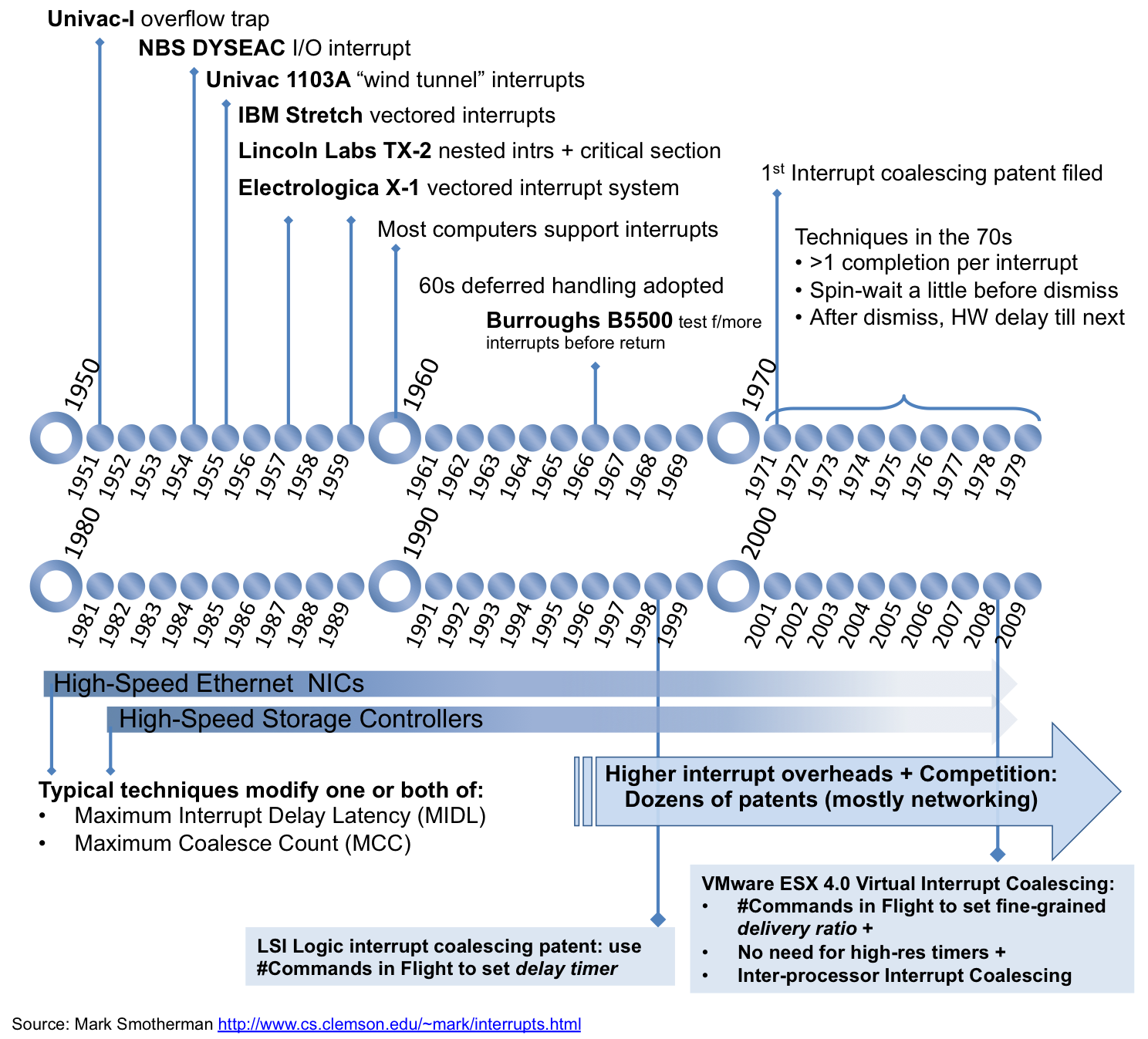
Источник: <http://virtualirfan.com/history-of-interrupts>.



Прерывания, также иногда называемые аппаратными прерываниями, это термин, применяемый вместо общего термина «исключительное событие или исключение». Исключение приостанавливает регулярный поток команд и переходит к вектору обработки в ответ не только на внешний сигнал, но и на некоторые внутренние события в процессоре, которые требуют немедленной реакции на них. К таким событиям относят прежде всего ошибки: арифметические переполнения, попытка доступа к недопустимому диапазону адресов, выполнение привилегированной команды в не привилегированном режиме (режиме пользователя), ошибки шины и другие исключительные ситуации. При обнаружении некоторых из таких ошибок следует прекратить программу, вызвавшую эту ошибку, в то же время при обнаружении других ошибок необходимо вызвать обработчик исключения, который ликвидирует возникшую проблему и возвратит программу в состояние, когда она сможет работать дальше по основному алгоритму. Некоторые из этих условий, такие как попытка доступа к недопустимому диапазону адресов, не могут рассматриваться как ошибки. Это скорее часть работы механизма, реализующего виртуальную память. Существуют также так называемые программные прерывания, исключения, которые инициируются программой и используются для обращения к функциям операционной системы. Исключения также используются в интерфейсе отладки процессора. Во время этой лабораторной работы мы будем иметь дело только с "настоящими" аппаратными прерываниями. Чтобы узнать о других аспектах исключений, пожалуйста, обратитесь к документации ядра и архитектуры.

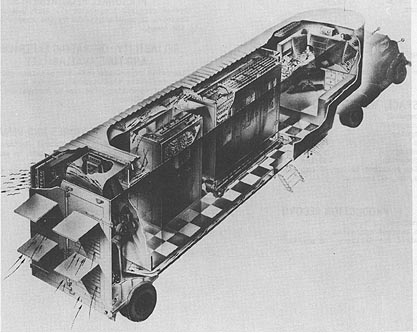
Как видно из истории компьютерной техники, использование исключений и прерываний было необходимо еще на ранней стадии развития компьютеров, как это показано на **Рисунке 2**.

**Рисунок 2. История исключений и прерываний.** Источник изображения: <http://virtualirfan.com/history-of-interrupts>.



Согласно статье *Прерывания* Марка by Maрка Смотермана (<https://people.cs.clemson.edu/~mark/interrupts.html#dyseac>), первым компьютером, который использовал прерывания ввода-вывода был DYSEAC, вторая версия SEAC (Standards Electronic Automatic Computer). Как говорит Википедия (<https://en.wikipedia.org/wiki/DYSEAC>), "DYSEAC сконструирован в Национальным бюро стандартов США для Службы связи армии США. Запущен в работу в апреле 1954 года". Согласно Смотерману, DYSEAC был возможно и первым движимым компьютером, перевозимым на двух прицепах, весом 12 и 8 тонн, как показано на **Рисунке 3**.

**Рисунок 3. DYSEAC, первый компьютер с прерываниями ввода-вывода.** Источник изображения: <http://ed-thelen.org/comp-hist>.



**3. Выполнение лабораторной работы**

Этот параграф описывает последовательность действий, необходимых для выполенения лабораторной работы. Большинство действий в этой лабораторной работе подобны тем действиям, которые производились при выполнении лабораторной работы *MIPSfpga 2.0 Lab YP1. Using MIPSfpga with Serial Loader Flow that does not require BusBlaster board and OpenOCD software*. Такие общие действия не описаны в этой разделе. Подробно описаны только те действия, которые отличаются от Lab YP1.

**3.1. Краткий обзор дополнительных материалов**

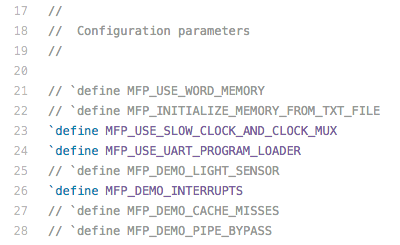
Используйте *Приложение А «Список рекомендованных источников до и во время выполнения лабораторной работы по прерываниям».* Краткий обзор списка материалов даст вам возможность понять, что именно из дополнительных материалов использовать во время выполнения работы.

**3.2. Обзор информации о прерываниях – сигналы от аппаратных средств**

Рассмотрите *MIPS32® microAptiv™ UP Integrator's Guide*, *Chapter 4: Interrupt Interface*. Это руководство включено в пакет *MIPSfpga*. Самая важная информация в этом документе относится к описанию вывода прерываний сигнала *SI\_Int*, используемого в этой лабораторной работе*.* В ходе лабораторной работы этот многобитный сигнал подключается к кнопкам на плате ПЛИС. Это позволяет генерировать прерывания простым нажатием на соответствующую кнопку.

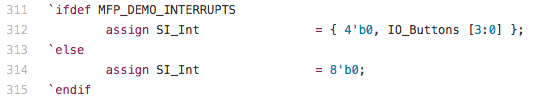
**3.3. Установка параметров конфигурации аппаратуры**

Измените конфигурационные параметры в файле *system\_rtl/mfp\_ahb\_lite\_matrix\_config.vh* так, как написано ниже:



**3.4. Обзор лабораторной работы – специфические модификации аппаратных средств**

Рассмотрим следующий фрагмент файла *system\_rtl/mfp\_system.v*:



**3.5. Подключение платы к компьютеру**

Для плат фирмы *Digilent*, таких как *Nexys4*, *Nexys4 DDR* или *Basys3*, этот шаг очевиден. Для плат фирмы *Altera/Terasic* потребуются некоторые дополнительные действия:

1. Подключите разъем преобразователя USB-to-UART к плате ПЛИС. Вы можете использовать преобразователь *FT232RL* или *PL2303TA,* которые можно приобрести в интернет-магазине. Вывод *TX* разъема преобразователя (зеленый провод для *PL2303TA*) должен быть подключен к выводу 3 правого нижнего разъема на платах Terasic DE0, DE0-CV, DE1, DE2-115 (или правого верхнего на плате DE0-Nano) и вывод *GND* (черный провод на *PL2303TA*) должен быть подключен к выводу 6 правого нижнего разъема на платах Terasic DE0, DE0-CV, DE1, DE2-115 (или правого верхнего на плате DE0-Nano). Проверьте подключение по рисунку в лабораторной работе *Lab YP1* для того, чтобы избежать короткого замыкания или других проблем подключения проводников.
2. Для преобразователя *FT232RL*: убедитесь в том, что переключатель 3.3V/5V в преобразователе *FT232RL* переключен в положение 3.3V.
3. Для некоторых плат рекомендуется использование внешнего дополнительного питания кроме того, что подается через USB разъем от компьютера. К таким платам относится *Terasic DE2-115*. Для этих плат необходимо подключить внешний источник питания.
4. Подключите плату ПЛИС к компьютеру с помощью основного загрузочного кабеля, который идет в поставке вместе с платой. Убедитесь, что вы подключили загрузочный кабель именно в нужный USB разъем, а не в разъем USB device (как например на плате *Terasic DE2-115*).
5. Убедитесь, что питание включено на плате ПЛИС (нажата кнопка питания) перед подключением UART кабеля от преобразователя USB-to-UART к компьютеру. Несоблюдение этого требования может привести к повреждению платы.
6. Подсоедините разъем USB-to-UART к плате ПЛИС.

**3.6. Запуск синтеза и конфигурация ПЛИС синтезируемой MIPSfpga системой**

Этот шаг идентичен шагу синтеза в лабораторной работе *Lab YP1*

**3.7. Переходим в папку с лабораторной работой и очищаем ее**

В системе Windows:

cd programs\04\_interrupts

00\_clean\_all.bat

В системе Linux:

cd programs/04\_interrupts

./00\_clean\_all.sh

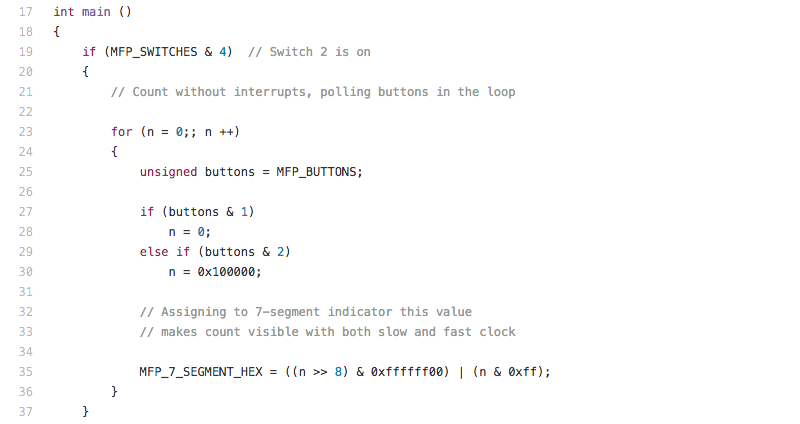
**3.8. Обзор кода лабораторной работы, не использующей прерывания**

Функция *main()* находится в файле *programs/04\_interrupts/main.c*. Эта функция выполняется после сброса и запуска последовательности загрузки. Основная функция в этой лабораторной работе просто запускает счетчик и выводит его значения на семисегментный индикатор на плате ПЛИС.

Выходное значение состоит как из старших, так и из младших битов счетчика. Это позволяет студенту наблюдать за изменениями цифр на дисплее при запуске синтезированной системы с помощью двух частот – высокой (50 МГЦ) или ультранизкой (0.75 Гц или 12.5 Гц). Для более подробной информации о переключаемых тактовых частотах см. лабораторную работу *Lab YP2.* Использование переключаемых тактовых частот позволяет проводить наблюдения за потактовой работой процессора.

В зависимости от положения переключателя switch 2, программа может запускаться как с использования прерываний так и без них. В обоих случаях счётчик сбрасывается в предопределенное значение, когда нажата кнопка 0 или кнопка 1.

Отметим, что на некоторых платах ПЛИС кнопка 0 – это системный сброс. Для таких плат можно либо делать всю лабораторную работу только с помощью кнопки 1, либо повторно синтезировать систему, подключив ее к кнопке или переключателю, отличному от кнопки 0.



**3.9. Подготовка к первому запуску программы**

Следуя процедуре, описанной в лабораторной *Lab YP1*, компилируем и линкуем программу, генерируем файл S-Record file и загружаем его в память синтезированной MIPSfpga системы на плате ПЛИС.

Для Windows:

1. cd programs\04\_interrupts
2. Запустите 02\_compile\_and\_link.bat
3. Запустите 08\_generate\_motorola\_s\_record\_file.bat
4. Запустите 11\_check\_which\_com\_port\_is\_used.bat
5. Отредактируйте 12\_upload\_to\_the\_board\_using\_uart.bat основываясь на результатах предыдущего шага – установите значение рабочего порта в строке "set a=".
6. Убедитесь, что переключатели 0 и 1 на плате ПЛИС выключены, а переключатель 2 - включен. Переключатели 0 и 1 определяют значение тактовой частоты, а переключатель 2 определяет будет ли программа использовать прерывания (переключатель 2 выключен) или не будет использовать прерывания (переключатель 2 включен). См. п. *3.8. Обзор кода лабораторной работы* . Если переключатели 0 и 1 выключены, то загрузка через UART не будет работать.
7. Запустите 12\_upload\_to\_the\_board\_using\_uart.bat

Для Linux:

Если происходит загрузка программы на плату первый раз в течение данной сессии Linux, то необходимо добавить текущего пользователя в группу Linux *dialout.* Введите пароль *root-а* при вводе следующего текста:

sudo adduser $USER dialout

su - $USER

После этого:

1. cd programs/04\_interrupts
2. Запустите ./02\_compile\_and\_link.sh
3. Запустите ./08\_generate\_motorola\_s\_record\_file.sh
4. Запустите ./11\_check\_which\_com\_port\_is\_used.sh
5. Отредактируйте ./12\_upload\_to\_the\_board\_using\_uart.sh основываясь на результатах предыдущего шага – установите значение рабочего порта в строке "set a=".
6. Убедитесь, что переключатели 0 и 1 на плате ПЛИС выключены, а переключатель 2 - включен. Переключатели 0 и 1 определяют значение тактовой частоты, а переключатель 2 определяет будет ли программа использовать прерывания (переключатель 2 выключен) или не будет использовать прерывания (переключатель 2 включен). См. п. *3.8. Обзор кода лабораторной работы*. Если переключатели 0 и 1 выключены, то загрузка через UART не будет работать.
7. Запустите ./12\_upload\_to\_the\_board\_using\_uart.sh

**3.10. Первый запуск программы**

1. Выключите переключатели 0 и 1 на плате ПЛИС, а переключатель 2 включите. Еще раз убедитесь, что переключатели 0 и 1 действительно выключены. В противном случае загрузка процессора (последовательность операций процессора до функции *main* ) будет очень долгой, т.к. эти переключатели определяют тактовую частоту процессора.
2. Сбросьте процессор. Для сброса процессора на каждой плате присутствует соответствующая кнопка, как показано ниже в таблице:

|  |  |
| --- | --- |
| **Плата** | **Кнопка сброса** |
| Digilent Basys3 | Вверх |
| Digilent Nexys4 | Выделенная кнопка сброса процессора |
| Digilent Nexys4 DDR | Выделенная кнопка сброса процессора |
| Terasic DE0 | Button/Key 0 |
| Terasic DE0-CV | Выделенная кнопка сброса процессора |
| Terasic DE0-Nano | Button/Key 0 |
| Terasic DE1 | Button/Key 0 |
| Terasic DE2-115 | Button/Key 0 |
| Terasic DE10-Lite | Button/Key 0 |

1. Убедитесь в выводе данных на семисегментный индикатор. Нажмите кнопку 1 и убедитесь в сбросе данных в счетчике.
2. Включите переключатель 1. Это изменит тактовую частоту системы с 25 МГц на 12.5 Гц / тактов в секунду. Вы увидите, что светодиод 7 начинает мигать, так как он подключен напрямую к тактовой частоте процессора. Отметьте также изменение скорости работы счетчика.
3. Нажмите кнопку 1 еще раз и заметьте как произошёл сброс счетчика.

**3.12. Обзор материалов, объясняющих атрибуты функции *interrupt***

Используйте следующий материал, чтобы понять атрибуты функции прерывания в коде: *Using the GNU Compiler Collection (GCC). 6.31.18 MIPS Function Attributes* (<http://gcc.gnu.org/onlinedocs/gcc/MIPS-Function-Attributes.html>).

**3.13. Обзор С макроса, используемого для доступа к регистрам Coprocessor 0**

Найдите файл *mips/cpu.h* в пакете Codescape компилятора GCC. Этот файл содержит набор макроопределений таких как *mips32\_getcr* и *mips32\_bicsr* , которые помогают получать доступ к регистрам сопроцессора 0 (Coprocessor 0). Эти регистры нужны для установки прерываний.

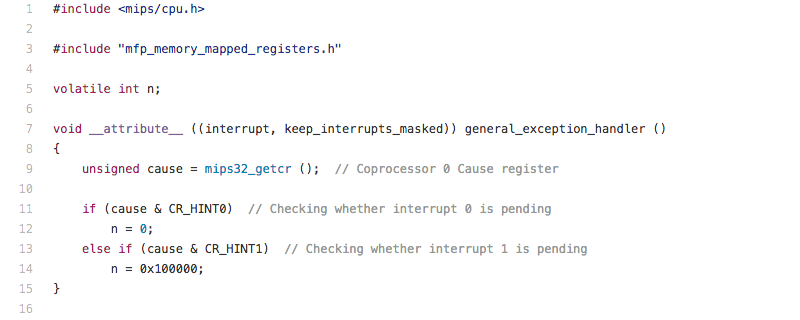
**3.14. Обзор регистров *Status* и *Cause* Coprocessor 0**

Для изучения данного вопроса можно обратиться к следующим материалам:

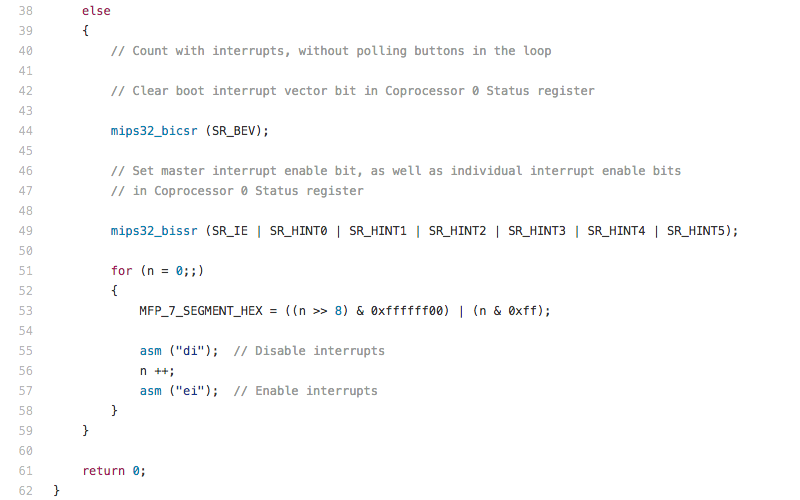
* Книга *See MIPS Run, Second Edition, by Dominic Sweetman*, *Chapter 3. Coprocessor 0: MIPS Process Control*
* *MIPS32® microAptiv™ UP Processor Core Family Software User's Manual*, *Chapter 6: CP0 Registers of the microAptiv™ UP Core*
* *MIPS® Architecture For Programmers Volume III: The MIPS32® and microMIPS32™ Privileged Resource Architecture*

**3.15. Обзор кода лабораторной работы, необходимого для использования прерываний**

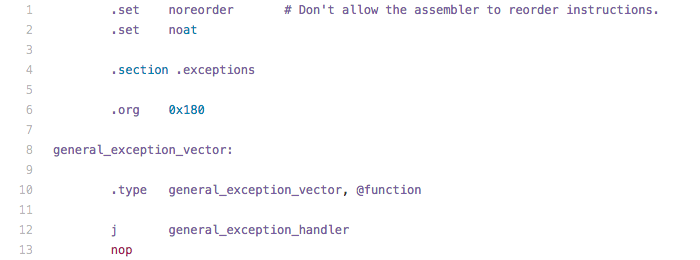
Обработчик прерываний находится в файле *programs/04\_interrupts/main.c*:



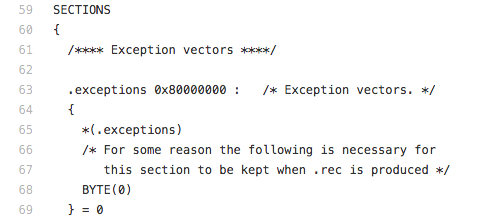
Установка регистров Coprocessor 0 для обработчика прерываний в *programs/04\_interrupts/main.c*:



Код установки вектора исключений в *programs/04\_interrupts/exceptions.S*:



Сценарий компоновщика, который включает новый раздел для вектора исключений в *programs/04\_interrupts/program.ld*:



**3.16. Запуск программы, использующей прерывания**

Еще раз выполните пункт *3.10. Первый запуск программы* с переключателем 2 в позиции «выключено». Заметили ли вы изменение скорости работы программы? Как вы можете объяснить это изменение.

**4. Домашнее задание: напишите все это на ассембелере**

Этот раздел помогает студенту понять сущность обработки прерываний на уровне сборки. Эта часть работы помогает получить четкое представление о том, как все работает.

**4.1. Дизассембирование ELF файла, созданного при выполнении работы**

Для Windows:

cd programs\04\_interrupts

04\_disassemble.bat

Для Linux:

cd programs/04\_interrupts

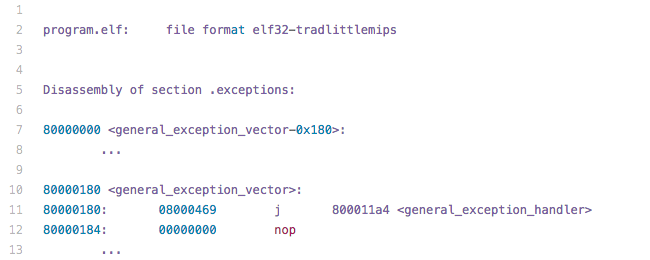
./04\_disassemble.sh

Сгенерированный файл носит название *program.dis*. Он находится в текущей директории.

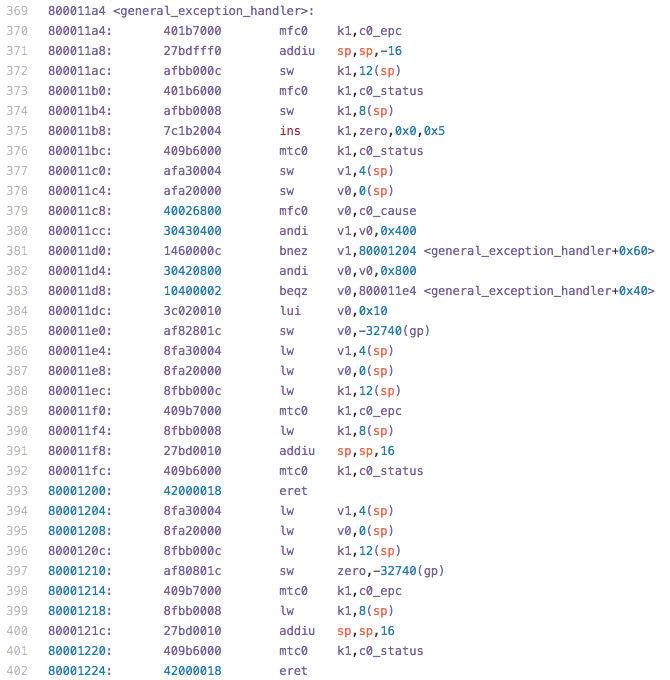
**4.2. Обзор и объяснение части кода, зависящей от прерываний**

Проанализируем часть файла *programs/04\_interrupts/program.dis.*

Код вектора прерываний:



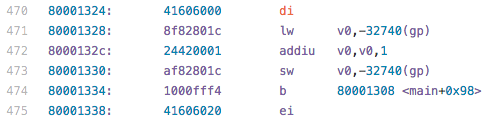
Код обработчика прерывания:



Код для установки прерываний:



Код, который защищает инкремент счетчика в основном цикле от прерываний. Такая защита осуществляется с использованием пары инструкций *di/ei*  (отключить прерывания / разрешить прерывания). Можете ли вы объяснить, почему это необходимо? Какая ошибка может произойти, если эти инструкции будут отсутствовать? Можете ли вы угадать сценарий, когда такая защита не будет работать?



**4.3. Перепишите всю лабораторию в сборке. Можете ли вы вручную написать подпрограмму обработки прерываний в сборке, которая имеет меньшее количество инструкций и работает быстрее, чем программа, сгенерированная компилятором GNU C?**

**5. Последующие проекты и упражнения**

Тема прерываний важна, и мы рекомендуем вам дополнительные упражнения, чтобы полностью ее освоить.

**5.1 Улучшения в лабораторных работах**

**5.1.1 Улучшения лабораторных работ: изменение системы тестирования для изучения прерываний с использованием моделирования Verilog без синтеза проекта**

Создайте все необходимые сценарии, чтобы показать основные сигналы, связанные с прерываниями, на временных диаграммах.

**5.1.2 Упражнение: улучшите код лабораторной работы по прерываниям, выставив значение счетчика команд (PC) на внешний семисегментный индикатор**

Создайте вариант данной лабораторной работы, которая соединяет счетчик команд процессора (РС) с внешним многоразрядным семисегментным индикатором. Это соединение должно быть мультиплексировано с обычным семисегментным индикатором и должно зависеть от того, нажат ли какой-либо выбранный переключатель или кнопка. С этой установкой, в том случае когда тактовая частота процессора переходит в медленный режим, можно будет наблюдать, как процессор входит в режим обработки прерываний. Подробнее о переключаемых тактовых частотах см. *MIPSfpga 2.0 Lab YP2. Using switchable clock to observe the CPU cycle-by-cycle*.

**5.1.3 Исследовательская задача: Какие биты регистра Coprocessor 0 должны быть выставлены на внешние светодиоды, чтобы наблюдать прерывания в действии в медленном режиме?**

Перейдите на веб-сайт компании Imagination Technologies <http://imgtec.com> и найдите там руководство *MIPS® Architecture For Programmers Volume III: The MIPS32® and microMIPS32™ Privileged Resource Architecture*. Исследуйте, какие поля регистра Coprocessor 0 интересно наблюдать, когда ядро ​​процессора получает прерывание и входит в подпрограмму обработки прерываний в режиме медленной тактовой частоты. Примером такого поля является бит EXL (Exception Level) регистра состояния (*Status*) Coprocessor 0.

Измените ядро ​​и системный RTL код, чтобы соединить эти регистры Coprocessor 0 с внешними светодиодами на плате ПЛИС. Некоторые поля, такие как *Status.EXL*, не требуют изменений в RTL коде ядра, поскольку исходный основной RTL код уже выводит это значение на внешний вывод *SI\_EXL*. Выставление таких полей требует только изменений в системе RTL (*system\_rtl / mfp \_ \*. V* файлы). В других полях может потребоваться добавление дополнительных портов и соединений в RTL коде ядра (*core\_rtl / m14k \_ \*. V* файлы).

Задокументируйте свои изменения и создайте сообщение на форуме MIPSfpga сайта Imagination Technologies.

**5.2 Синхронизация доступа к памяти в прерываниях**

**5.2.1 Упражнение: Наблюдайте за результатом работы программы, которая не отключает прерывания вокруг критической секции, которая обновляет ту же переменную, что и обработчик прерывания**

Закомментируйте команды сборки, которые включают и отключают прерывания (*asm ("di");* и *asm ("ei");*) вокруг команды инкремента переменной счетчика *n* в основной функции *main*  внутри файла *main.c*:

// asm ("di");

n ++;

// asm ("ei");

Откомпилируйте и перезапустите программу, используя быструю и медленную тактовые частоты. Видите ли вы видите разницу в реакции системы на прерывания? Вы можете наблюдать случаи, когда система получает прерывание и выполняет подпрограмму обработки прерывания, которая сбрасывает счетчик. Однако после возврата из прерывания счетчик не сбрасывается. Вместо этого он продолжает увеличивать свое старое значение, установленное до входа в подпрограмму обработки прерываний. Можете ли вы объяснить, что происходит? Мы рекомендуем анализировать выходные данные сборки компилятора *gcc*, выполнив *01\_compile\_c\_to\_assembly.sh* под Linux или *01\_compile\_c\_to\_assembly.bat* под Windows.

**5.2.2 Упражнение: Синхронизировать обновления переменной счетчика с помощью пары инструкций *LL/SC* (Load-Linked / Store-Conditional)**

Архитектура MIPS обеспечивает способ синхронизации изменений переменных без отключения прерываний, используя специальную пару инструкций *LL / SC* (Load-Linked / Store-Conditional). Создайте версию *main.c*, которая использует эту особенность процессора вместо пары инструкций *DI / EI*, используемых в этой работе, и упражнении 5.2.1. Если вы не знакомы с *LL / SC*, вы можете изучить следующие материалы:

• *MIPS32® microAptiv™ UP Processor Core Family Software User's Manual*, раздел 12.3, описания инструкций *LL* и *SC*. Данное руководство включено в пакет *MIPSfpga*

• Книга *See MIPS Run*, второе издание, *Dominic Sweetman*, разделы 5.8.4 *Critical Regions with Interrupts Enabled: Semaphores the MIPS Way* и 8.5.2 *Load-Linked / Store-Conditional*.

Вы можете реализовать это упражнение, используя либо функцию, написанную в сборке и вызванную из функции *main*, либо, альтернативно, используя конструкцию *asm* в C с параметрами, как описано в онлайн-документации GCC (<https://gcc.gnu.org/onlinedocs>), раздел [6.45.2 Extended Asm - Assembler Instructions with C Expression Operands](https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html)

**5.2.3 Исследование: зачем нужно ключевое слово *volatile* для переменной счетчика**

Просмотрите код файла *main.c*. Зачем нужна переменная *n* счетчика, объявляемая с ключевым словом *volatile*? Как отсутствие этого ключевого слова влияет на результат выполнения программы? Попробуйте запустить программу без *volatile* с различными уровнями оптимизации компилятора. Есть ли какие-либо изменения в поведении? Просмотрите код, созданный компилятором gcc с различными параметрами *-O*.

**5.3 Прерывания таймера**

**5.3.1 Студенческий проект: Проект с прерываниями таймера с использованием пары *Count / Compare* для пары регистров Coprocessor 0, существующих внутри ядра *MIPS microAptiv UP***

Прерывание таймера представляет собой особый тип аппаратного прерывания, который происходит регулярно, с установленной частотой, обычно в диапазоне килогерц. Такие прерывания часто используются для измерения временных интервалов и для реализации многозадачности программного обеспечения, включая параллельное выполнение задач в операционных системах.

Прочитайте документацию по аппаратным и программным средствам о встроенном прерывании таймера, используемом в ядрах MIPS, в сочетании с парой регистров *Count / Compare* Coprocessor 0. Эта функция присутствует в MIPSfpga? Если да, реализуйте лабораторную работу с использованием этой функции. В этой лабораторной работе можно, например, измерить время между нажатием какой-либо кнопки или выполнить некоторые вычисления, в то время как программа обработки прерываний опросит какой-либо вход.

**5.3.2 Студенческий проект: Лабораторная работа по прерываниям с использованием специального таймера, реализованного вне ядра *MIPS microAptiv UP***

Создайте альтернативную реализацию лабораторной работы по прерываниям таймера (Упражнение 5.3.1), не используя прерывание таймера, уже реализованного в ядре *MIPS microAptiv UP*. Чтобы создать такую реализацию, напишите Verilog модуль генерации прерываний пользовательского таймера. Затем подключите его к одному из выводов аппаратного прерывания, битам внешнего сигнала *SI\_Int*.

**5.3.3 Студенческий проект: Лабораторная работа по прерываниям таймера с использованием специального таймерного модуля, реализованного вне ядра MIPS microAptiv UP**

Расширьте студенческий проект 5.3.2, связав модуль формирования прерываний таймера не только с системными часами, сбросом и сигналом *SI\_Int*, но и с системной шиной *AHB-Lite*. Целью проекта является контроль частоты прерываний с помощью программного обеспечения, работающего на ядре процессора *MIPSfpga*. Более подробно об интерфейсе *AHB-Lite* см. в Lab YP3 - Интеграция периферийного устройства: пример датчика освещенности.

**5.3.4 Студенческий проект: вариант лабораторной работы по прерываниям таймера, которая использует несколько счетчиков и выводов прерываний**

Создайте версию студенческого проекта 5.3.3, который использует несколько счетчиков для генерации аппаратных прерываний с различными частотами на разных аппаратных выводах (биты сигнала *SI\_Int*).

**5.3.5 Проект для студентов среднего и продвинутого уровня: создать лабораторную работу, демонстрирующую многозадачность / переключение контекста**

Важным применением прерываний таймера является облегчение варианта параллельного программирования, называемого переключением задач, или переключением контекста. Переключение контекста широко используется в операционных системах, от относительно простых, таких как FreeRTOS, до сложных, таких как Linux.

Идея переключения контекста состоит в периодическом переключении между различными потоками исполнения, которые также иногда называются процессами или задачами. «Контекст» - это ссылка на набор информации, связанной с задачей, включая счетчик программ (PC) и регистры общего назначения (GPR). Переключение происходит внутри процедуры обработки прерывания таймера, которая сохраняет текущий контекст в некоторой структуре памяти (называемой в некоторых системах Process Control Block - PCB), затем восстанавливает контекст другого процесса из другой PCB и выходит из прерывания таймера в новый поток исполнения.

Создайте лабораторную работу, которая переключается между двумя различными функциями C, работающими параллельно. Для этого вам не нужно использовать какую-либо операционную систему. Просто сохраняйте и восстанавливайте все необходимые регистры внутри процедуры обработки прерываний таймера и поддерживайте иллюзию параллельного выполнения двух C-программ для конечного пользователя. Конечный пользователь будет наблюдать выходы двух программ на светодиодах или других устройствах вывода.

**5.3.6 Продвинутый студенческий проект: Портируйте операционную систему реального времени с открытым исходным кодом, такие как FreeRTOS, в MIPSfpga**

FreeRTOS - популярная операционная система реального времени, используемая в качестве примера RTOS в курсе *Connected MCU*, созданном доктором Александром Дином из Университета Северной Каролины. Курс *Connected MCU* демонстрирует работу FreeRTOS на микроконтроллере Microchip PIC32MZ, в котором используется процессорное ядро, схожее с MIPSfpga. Однако PIC32MZ использует внешний контроллер прерываний, отсутствующий в системе MIPSfpga по умолчанию, которая использует Режим Совместимости Прерываний (Interrupt Compatibility Mode). Портирование FreeRTOS на MIPSfpga должно быть подходящим проектом для аспирантов, изучающим программирование встроенных систем или компьютерную архитектуру.

**5.3.7 Проект для студентов среднего и продвинутого уровня: создайте лабораторную работу, демонстрирующую переход из пользовательского режима в режим ядра при входе в подпрограмму обработки прерываний**

Режим работы ядра процессора - одна из самых важных концепций в информатике, особенно в разработке операционных систем. Различные ЦП имеют несколько разных режимов с разными названиями: пользователь, ядро, привилегированный, суперпользователь и другие. Архитектура MIPS32 поддерживает пользовательский режим, режим ядра и режим супервизора. Согласно книге *See MIPS Run*, второе издание, *Dominic Sweetman*, режим супервизора в MIPS - это историческое недоразумение, функция, добавленная по просьбе клиента, но практически не использовавшаяся в реальной системе. Поэтому в реальной работе используются только режим пользователя и режим ядра.

В последних версиях архитектуры MIPS32 есть несколько дополнительных режимов: режим root-а и гостевой режим. Эти режимы связаны с аппаратной поддержкой виртуализации CPU, которую не следует путать с функцией виртуальной памяти. Виртуальная память присутствует даже в ядрах процессоров без аппаратной поддержки виртуализации, включая ядро MIPS microAptiv UP, используемое в MIPSfpga. Однако аппаратная поддержка виртуализации отсутствует в MIPSfpga, поэтому мы не касаемся здесь режима root-а и гостевого режима.

Процессор в режиме ядра имеет полный доступ ко всем ресурсам своей архитектуры, он может выполнять все команды, получать доступ ко всем ячейкам памяти и ко всем регистрам системы Coprocessor 0. Этот режим используется для запуска драйверов операционной системы и устройств.

Процессор в пользовательском режиме может выполнять только подмножество команд, которое не имеет доступа к регистрам системного сопроцессора и контролирует доступ к ячейкам памяти. Этот режим используется для запуска прикладных программ, которые могут быть менее надежными, чем операционная система.

Когда происходит прерывание или, как правило, возникает исключение, процессор переключает свой режим с режима пользователя на режим ядра ​​и переходит к обработчику исключения, части кода, которая обрабатывает исключение. Когда обрабатывается исключение, операционная система выполняет следующие необходимые действия:

* Если код пользователя пытался получить доступ к адресу в памяти, к которому он не должен иметь доступа, операционная система может прервать исполнение кода пользователя, сохраняя в памяти всю остальную часть системы (как операционную систему, так и память другого приложения) без изменений.
* Если исключение вызвано прерыванием системного таймера, программа обслуживания прерывания может переключить контекст задачи / процесса и вернуться из исключения в другую пользовательскую задачу / процесс.
* Если прерывание пришло от устройства ввода-вывода, процедура обслуживания прерывания может выполнять некоторые действия, связанные с доступом к отображаемым в памяти регистрам ввода-вывода. Примером такого действия является получение байта, переданного по протоколу SPI, и помещение его в буфер, с которым впоследствии может работать программа пользователя.
* Существуют специальные инструкции в пользовательском режиме, которые намеренно вызывают исключения, заставляя систему выполнить некоторые действия в режиме ядра. В MIPS такая инструкция называется *syscall*.
* Наконец, исключение может быть вызвано незаконной инструкцией или инструкцией, которая не разрешена в пользовательском режиме. Обработчик исключений может попытаться эмулировать эту инструкцию вместо завершения пользовательского кода.

Создайте лабораторную работу, используя MIPSfpga, которая демонстрирует, как процессор входит в пользовательский режим. Эта лабораторная работа может отображать поле *KSU* («ядро / супервизор / пользователь») в регистре состояния Coprocessor 0, подключая его к светодиодам на плате FPGA и запуская систему с использованием медленного тактового сигнала 0,75 Гц или 12,5 Гц.

Главной задачей при создании этой лабораторной работы является необходимость изменить код загрузчика. В текущем загрузочном коде имеется сброс в режим ядра, который должен быть изменен на пользовательский. Текущий загрузочный код также не инициализирует сопоставление виртуальных адресов для пользовательского режима. Просмотрите документацию по буферу трансляции переводов (TLB) и блоку управления памятью и измените загрузочный код так, чтобы можно было выполнить новою лабораторною работу.

**5.4 Специальные виды прерываний, функции прерываний и прерывания, подобные исключениям**

**5.4.1 Студенческий проект и исследование: лабораторная работа по немаскируемым прерываниям (NMI)**

Изучите информацию о немаскируемых прерываниях (NMI) в руководствах *MIPS32® microAptiv™ UP Processor Core Family Software User's Manual* и *MIPS32® microAptiv™ UP Integrator's Guide*, включенных в пакет *MIPSfpga*.

Создайте лабораторную работу, которая демонстрирует немаскируемые прерывания (NMI). Исследуйте с помощью Интернета историю применения NMI на разных компьютерах. Почему эта функция была необходима в прошлом? Насколько полезна эта функция для современных приложений?

**5.4.2 Исследование: оцените полезность и применение режима векторных прерываний (VI) в архитектуре MIPS**

Ядро MIPS microAptiv UP, используемое в MIPSfpga, поддерживает три режима прерываний: режим совместимости прерываний (Interrupt Compatibility), режим векторных прерываний (Vectored Interrupt – VI) и режим использования внешнего контроллера прерываний (External Interrupt Controller – EIC). До сих пор в лабораторной работе и упражнениях использовался только режим совместимости прерываний. Альтернативный режим векторных прерываний (VI) добавляет в обработчик прерываний возможность использования приоритета и векторных прерываний. В режиме векторных прерываний также можно назначить набор теневых регистров общего назначения (General Purpose Register – GPR) для использования во время обработки прерываний.

Изучите документацию и создайте лабораторную работу, демонстрирующую работу процессора в режиме векторных прерываний. Сколько тактовых циклов сохраняет этот режим при обработке прерывания? Когда имеет смысл использовать этот режим? Рассматривайте очень маломощные / низкочастотные приложения, время отклика прерываний, использование теневых регистров регистров общего назначения (не присутствует в MIPSfpga).

Какой режим прерывания используется в Linux? Можете ли вы предложить объяснение, почему?

**5.4.3 Исследование: использование бита *IV* регистра Cause для Coprocessor 0**

Прочитайте о бите *IV* бит регистра *Причины (Cause)* Coprocessor 0. Зачем нужен такой вариант прерывания? Какая проблема решается этим вариантом? Предложите возможные ответы.

**5.4.4 Исследование: назначение регистра *EBase* Coprocessor 0**

Прочитайте о регистре *EBase* Coprocessor 0 и попытайтесь объяснить его необходимость. Это возможно только для многопроцессорных систем? Как бы вы использовали его в однопроцессорной системе?

**5.4.5 Расширенный студенческий проект: построение внешнего контроллера прерываний (EIC)**

Режим внешнего прерывания (EIC) является еще одним режимом прерывания, поддерживаемым ядром MIPS microAptiv UP, используемым в MIPSfpga. Режим EIC переопределяет способ обработки прерываний, чтобы обеспечить полную поддержку внешнего контроллера прерываний, а также поддержку приоритетов и векторов прерываний. Режим EIC используется в микроконтроллерах на основе MIPS от Microchip Technology, включая некоторые микроконтроллеры семейства Microchip PIC32MZ, построеные на базе ядра MIPS microAptiv UP, того же ядра, что и в MIPSfpga.

Функциональность внешнего контроллера прерываний, созданного Microchip, описана в документации к программному обеспечению Microchip, в курсах, проводимых Microchip во время конференции Microchip Master Conference, а также в курсе *Connected MCU*, созданном д-ром Alexander Dean из Университета штата Северная Каролина и спонсируемом компаниями Microgiver и Digilent.

MIPSfpga позволяет вам создать альтернативный внешний контроллер прерываний со своим собственным планированием прерываний и приоритетов и сравнить ваше решение с решением от Microchip Technology. Это значительный исследовательский проект, который можно комбинировать с исследованиями операционных систем реального времени (RTOS).

**5.4.6 Студенческий проект: создайте лабораторную работу, демонстрирующую неточное исключение Bus Error**

Все прерывания являются исключениями, вызванными изменениями внешних сигналов, но не все исключения, вызванные изменениями внешних сигналов, фактически являются прерываниями. Существует исключение, называемое Bus Error, которое инициируется специальным откликом шины AHB-Lite. Это исключение также примечательно тем, что оно не всегда точно, то есть может произойти с *EPC*, указывающим на другую команду.

Ознакомьтесь с неточными исключениями и ошибками шины в документации и создайте лабораторную работу, демонстрирующую эту функцию. Полезные главы для чтения:

* *MIPS32® microAptiv™ UP Integrator's Guide*, *Table 2.3 Signal Descriptions for m14k\_cpu Level*, описание сигнала *HRESP* : *"Передача ответа. Когда он принимает значение нуля (LOW), сигнал HRESP индицирует, что статус передачи OKAY. Когда он принимает значение единицы (HIGH), сигнал HRESP индицирует, что статус передачи ERROR."*. Это руководство включено в пакет *MIPSfpga*.
* *MIPS32® microAptiv™ UP Processor Core Family Software User's Manual*, раздел *5.8.15 Bus Error Exception - Instruction Fetch or Data Access*. Обратите внимание «*Ошибка шины возникает при запросе критичных инструкций или данных которые не находятся в кеше в данный момент. Остальные ошибки шины могут быть не определены. Это может быть запись или чтение некритичных (не нужных именно сейчас) данных при пакетном обращении.»* Это руководство включено в пакет *MIPSfpga*.
* Книга *See MIPS Run*, второе издание, *Dominic Sweetman*, раздел *5.1 Precise Exceptions* и *Table 3.2 ExcCode Values: Different Kinds of Exceptions*.

**5.5 Эффекты прерываний в микроархитектуре процессора**

**5.5.1 Расширенный студенческий проект: создайте лабораторную работу, демонстрирующую влияние прерываний на конвейерную обработку процессора**

Создайте все необходимые сценарии, чтобы отображать основные сигналы, связанные с прерыванием, во время симуляции в симуляторе Verilog. Покажите на диаграммах, как прерывание очищает конвейер процессора. На каком этапе конвейера используется ожидающее прерывание?

Подключите сигналы управления конвейером к внешним светодиодам на плате ПЛИС и продемонстрируйте, как прерывание очищает конвейер процессора от команд, которые не будут завершаться из-за прерывания.

Если вам нужна дополнительная информация о работе конвейера в процессоре MIPSfpga, вы можете просмотреть *MIPSfpga 2.0. Lab YP6 - The first glance into pipelining*.

**5.6 Переделайте лабораторную работу по использованию датчика освещенности, сделав ее управляемой с помощью прерываний**

Выполнив всю эту лабораторную работу давайте вернемся к лабораторной работе, использующей датчик освещенности *MIPSfpga 2.0. Lab YP3 - Integrating a peripheral: the light sensor example* и повторно реализуем ее с использованием подхода, управляемого прерываниями. Измените модуль интерфейса датчика света, чтобы он выдавал прерывание при изменении измеренного значения. Подключите вывод прерывания к сигналу *SI\_Int* ядра MIPS microAptive UP. Измерьте улучшение производительности системы, которое возникает при разгрузке ввода-вывода в подпрограмму обслуживания прерываний.

**Приложение A. Список рекомендованных материалов для обзора до и во время выполнения лабораторной работы по прерываниям**

* Книга *See MIPS Run*, второе издание, *Dominic Sweetman*. *Chapter 5. Exceptions, Interrupts, and Initialization*, sections 5.1-5.8. Если у вас нет этой книги, смотрите *Приложение Б* к этой лабораторной работе, содержащее наиболее важные фрагменты и книги. Разделы из книги рекомендуемые к изучению:
  + *Chapter 5. Exceptions, Interrupts, and Initialization*, sections 5.1-5.8
  + *Chapter 3. Coprocessor 0: MIPS Process Control*, sections:
    - *3.3.1 Status Register (SR)*
    - *3.3.2 Cause Register*
    - *3.3.3 Exception Restart Address (EPC) Register*
    - *3.3.5 Count/Compare Registers: The On-CPU Timer*
    - *3.3.8 EBase and IntCtl: Interrupt and Exception Setup*
    - *3.3.9 SRSCtl and SRSMap: Shadow Register Setup*
    - *3.3.10 Load-Linked Address (LLAddr) Register*
  + *Chapter 8. Complete Guide to the MIPS Instruction Set*, section *8.5.2 Load-Linked / Store-Conditional*
* *MIPS32® microAptiv™ UP Integrator's Guide*, *Chapter 4: Interrupt Interface*. Это руководство включено в пакет *MIPSfpga*.
* *MIPS32® microAptiv™ UP Processor Core Family Software User's Manual*, соответствующие разделы из:
  + *Chapter 5: Exceptions and Interrupts in the microAptiv™ UP Core*
  + *Chapter 6: CP0 Registers of the microAptiv™ UP Core*
  + раздел 12.3, описывающий инструкции *LL* и *SC*

Это руководство включено в пакет *MIPSfpga*.

* Соответсвующие разделы из руководства *MIPS® Architecture For Programmers Volume III: The MIPS32® and microMIPS32™ Privileged Resource Architecture*. Это руководство можно загрузить с сайта компании Imagination Technologies <http://imgtec.com>

**Приложение Б. Выдержки из книги *See MIPS Run,* второе издание*, Dominic Sweetman* об обработке прерываний.**

**Раздел 5. Исключения, прерывания и инициализация**

В архитектуре MIPS прерывания, ловушки, системные вызовы и все остальное, что может нарушить нормальный поток выполнения программы, называется исключениями и обрабатываются одним механизмом. Что это за события?

* *Внешние события:* некоторые события за пределами центрального процессора - то есть приходящие от некоторого реального входного сигнала, «проводника». Это прерывания. (Примечание. В настоящее время существуют некоторые более незаметные непрерываемые внешние события, такие как ошибки шины, появляющиеся при чтении, просто предположим, что они являются особым видом прерываний). Прерывания используются для того, чтобы обратить внимание процессора на некоторое внешнее событие: это такая особенность ОС, которая может обслуживать более одного события за раз. Прерывания - это единственные исключения, возникающие из-за чего-то независимого от обычного потока команд процессора. Поскольку вы не можете избежать прерываний, просто проявляя осторожность при написании кода, то должны быть некоторые программные механизмы для подавления эффекта прерываний, в ситуациях, когда это необходимо.
* *Исключения трансляции памяти.* Они случаются, когда адрес необходимо перевести, но реальный перевод недоступен для оборудования или, возможно, происходит запись на защищенную от записи страницу памяти.
* ОС должна решить, является ли такое исключение ошибкой или нет. Если исключение является признаком того, что прикладная программа выходит за пределы допустимого адресного пространства, то это будет исправлено ​​путем завершения прикладной программы для защиты остальной системы. Более распространенные исключения трансляции памяти могут быть использованы для инициализации такой достаточно сложной функций операционной системы, как система виртуальной памяти с подкачкой страниц по требованию или же просто, как расширение пространства, доступного для стека.
* *Другие необычные условия в работе ядра, требующие исправления*: заметным среди них являются условия, являющиеся результатом инструкций с плавающей запятой, где аппаратніе средства не можут справиться с трудной и редкой комбинацией операций и операндов и необходимо использование программного эмулятора. Эта категория условий не может біть четко описана, так как разные ядра имеют разные представления о том, что они хотят исправить. Неравномерная загрузка может быть ошибкой в ​​одной системе и тем, что будет обрабатываться программным обеспечением на другом.
* *Програмно или аппаратно обнаруженные ошибки*: включает несуществующие инструкции, инструкции, которые являются незаконными на уровне привилегий пользователя, инструкции сопроцессора, выполненные с отключенным флагом SR, переполнением целочисленного значения, ошибками выравнивания адресов и обращениями за пределами *kuseg* в пользовательском режиме.
* *Проблемы с целостностью данных*. Многие процессоры MIPS постоянно проверяют данные на шине или данные, поступающие из кэша, для проверки на четность или исправления ошибок в слове целиком. Ошибки кеша или четности порождают исключение в тех процессорах, которые поддерживают проверку данных.
* *Системные вызовы и ловушки*: это инструкции, цель которых - генерировать распознаваемые исключения. Они используются для создания программных средств безопасным способом (системные вызовы, условные ловушки, установленные при написани кода и точки останова).

**5.3 Вектора исключений: где начинается обработка исключений**

. . . . . . . . . .

Вот что делает процессор MIPS, когда он получает запрос на исключение:

1. Он устанавливает **EPC** для указания места перезапуска.
2. Он устанавливает **SR(EXL)**, который переводит процессор в режим ядра (высокий уровень привилегий) и отключает прерывания.
3. Регистр **Cause** установлен так, что программа может видеть причину исключения. В исключениях адресов также устанавливается **BadVAddr**. Исключения системы управления памятью также устанавливают некоторые из регистров MMU; Более подробная информация приведена в главе 6.
4. Затем ЦП начинает выборку инструкций из точки входа исключения, а все остальные действия зависят от программы.

. . . . . . . . . .

**5.5 Возврат из исключения**

Возврат назад контроля программе и изменение (если требуется) уровня доступа с уровня ядра на уровень с более низкими привилегиями должен выполняться одновременно («атомарно» на жаргоне информатики). Это было бы брешью в системе безопасности процессора, если бы вы запустили хотя бы одну инструкцию кода приложения на уровне привилегий ядра. С другой стороны, попытка запустить инструкцию ядра с привилегиями пользователя также привела бы к фатальному исключению.

У процессоров MIPS есть инструкция, **eret**, которая выполняет всю эту работу. Она очищает бит **SR (EXL)** и возвращает управление в адрес, хранящийся в **EPC**.

**5.8.1 Ресурсы прерываний в процессорах MIPS**

У процессоров MIPS есть набор из восьми независимых бит прерывания в регистре **Cause** (Примечание: они не так независимы, если вы используете режим EIC, см. Раздел 5.8.5.). В большинстве процессоров вы обнаружите, что пять или шесть из них – это сигналы от внешней логики в CPU, в то время как два из них являются чисто программными. Встроенный счетчик / таймер (из регистров **Count** и **Compare**, описанных в разделе 3.3.5) будет подключен к одному из них. Иногда можно разделить прерывание счетчика / таймера с внешним устройством, но очень часто это не самая лучшая идея.

Активный уровень любого входного сигнала регистрируется в каждом цикле и вызывает исключение, если оно включено.

На готовность процессора реагировать на прерывание влияют значения битов SR.

Имеются три соответствующих поля:

* Бит разрешения глобального разрешения **SR (IE)** должен быть установлен в 1, или прерывание не будет обслуживаться.
* Биты **SR (EXL)** (уровень исключения) и **SR (ERL)** (уровень ошибки) будут блокировать прерывания, если они установлены (поскольку один из них будет установлен сразу же после появления любого исключения).
* Регистр состояния также имеет восемь отдельных битов маски прерывания **SR (IM)**, по одному для каждого бита прерывания в регистре **Cause**. Каждый бит **SR (IM)** должен быть установлен в 1, чтобы разрешить соответствующее прерывание, и программы могли точно определить, какие прерывания могут произойти, а какие нет.

Чтобы узнать, какие входы прерывания активны, необходимо изучить значение регистра **Cause**. Обратите внимание, что это точно «текущие уровни» и необязательно соответствуют шаблону сигнала, который вызвал исключение прерывания в первую очередь. Активные входные уровни в регистре **Cause(IP)** и масках в **SR (IM)** удобно выровнять по одному и тому же положению бит, в том случае, если вы хотите потом использовать для них функции «И». Программные прерывания находятся на самых низких уровнях приоритета, а аппаратные прерывания расположены в порядке возрастания.

В плане архитектуры процессора все прерывания равны. (Примечание. Это не совсем верно для векторного прерывания и режима «EIC», описанного в разделе 5.8.5, но эти режимы используются не слишком часто). Когда происходит исключение или прерывание, более старый процессор использует «общую» точку входа-исключения, хотя процессоры MIPS 32/64 и некоторые другие современные процессоры предлагают необязательную отдельную точку входа исключения, зарезервированную для прерываний, что позволяет сэкономить несколько циклов. Вы можете выбрать эту функцию с помощью бита IV в регистре **Cause**.

Обработка прерывания начинается после того, как мы получили исключение и после анализа бита **Cause(ExcCode)** обнаружили, что это аппаратное прерывание. Анализируя состояние **Cause** (**IP**), мы можем увидеть, какое прерывание активно, и, таким образом, какое устройство сигнализирует нам. Вот обычная последовательность обработки прерывания:

* Выполните логическую операцию «И» с полем **IP** регистра **Cause** и маской прерываний в **SR(IM)**, чтобы получить битовую карту активных разрешенных запросов прерываний. В результате вы можете обнаружить, что прерывание было вызвано больше чем одним событием.
* Выберите одно активное прерывание для обработки. Большинство ОС назначают разные входы для фиксированных приоритетов и в первую очередь работают с самым высоким приоритетом, но все это решается на уровне программного обеспечения.
* Вам нужно сохранить старые биты маски прерывания в **SR (IM)**, но вы, вероятно, уже сохранили весь регистр **SR** в основной процедуре обработки исключения.
* Измените состояние **SR (IM)**, чтобы гарантировать, что текущее прерывание и все прерывания, которые ваше программное обеспечение считает с равными или меньшими приоритетами, запрещены.
* Если вы еще не сделали это в основной процедуре исключения, сохраните контекст (пользовательские регистры и т. д.), необходимый для обработки вложенных исключений.
* Теперь измените состояние вашего процессора на соответствующее более высокоуровневой части обработчика прерываний, где обычно допускаются некоторые вложенные прерывания и исключения.

В любом случае установите бит разрешения **SR (IE)** для разрешения прерываний более высокого приоритета. Вам также необходимо изменить поле уровня привилегий на уровне процессора **SR(KSU)**, чтобы поддерживать процессор в режиме ядра, когда вы очищаете уровень исключения и, конечно же, очищаете сам **SR (EXL)**, чтобы выйти из режима исключения и сделать доступными изменения, сделанные в регистре состояния.

* Вызовите вашу процедуру прерывания.
* При возврате вам нужно будет снова отключить прерывания, чтобы вы могли восстановить значения регистров до входа в прерывание и возобновить выполнение прерванной задачи. Для этого вы установите **SR (EXL)**. Но на практике вы, скорее всего, будете делать это неявно, во время восстановления старого значение регистра **SR**, прежде чем попасть в вашу последовательность обработки возврата из исключения.

При внесении изменений в **SR** необходимо соблюдать осторожность в отношении изменений, влияние которых задерживается из-за работы конвейера «опасностей CP0». См. раздел 3.4 для больших подробностей и описания того, как программа работает в подобных ситуациях.

**Приложение В. Выдержки из статьи *Использование коллекции компиляторов GNU (GCC). 6.31.18 Атрибуты функции MIPS***

<http://gcc.gnu.org/onlinedocs/gcc/MIPS-Function-Attributes.html>

Эти атрибуты функции поддерживаются MIPS back end:

interrupt

Используйте этот атрибут, чтобы указать, что указанная функция является обработчиком прерывания. Когда этот атрибут присутствует компилятор будет генерировать последовательности входа и выхода функций, более подходящие для использования в обработчике прерываний. Для атрибута прерывания поддерживается также необязательный аргумент, который позволяет описать режим прерывания. По умолчанию GCC предполагает, что используется режим внешнего прерывания (EIC), это можно явно установить с помощью eic. Когда прерывания не маскируются, запрошенный уровень приоритета прерывания (Interrupt Priority Level – IPL) копируется в текущую IPL, и это имеет эффект только при разрешении прерываний с более высоким приоритетом. Для использования векторного режима прерывания используйте аргумент vector = [sw0 | sw1 | hw0 | hw1 | hw2 | hw3 | hw4 | hw5], что изменит поведение поддержки маскированных прерываний, и GCC организует маскирование всех прерываний от sw0 до вектора прерываний с указанным номером.

Для изменения поведения обработчика прерываний вы можете использовать следующие атрибуты:

use\_shadow\_register\_set

Предположим, что обработчик использует набор теневых регистров вместо основных регистров общего назначения. Дополнительный аргумент intstack используется, чтобы указать, что набор теневых регистров содержит допустимый указатель стека.

keep\_interrupts\_masked

Прерывания маскируются для всей функции. Без этого атрибута GCC пытается разблокировать прерывания для максимально возможного количества функций.

use\_debug\_exception\_return

Для возврата используйте инструкцию deret. Обработчики прерываний, у которых нет этого атрибута, возвращаются с использованием eret.

Вы можете использовать любую комбинацию этих атрибутов, как показано ниже:

void \_\_attribute\_\_ ((interrupt)) v0 ();

void \_\_attribute\_\_ ((interrupt, use\_shadow\_register\_set)) v1 ();

void \_\_attribute\_\_ ((interrupt, keep\_interrupts\_masked)) v2 ();

void \_\_attribute\_\_ ((interrupt, use\_debug\_exception\_return)) v3 ();

void \_\_attribute\_\_ ((interrupt, use\_shadow\_register\_set,

keep\_interrupts\_masked)) v4 ();

void \_\_attribute\_\_ ((interrupt, use\_shadow\_register\_set,

use\_debug\_exception\_return)) v5 ();

void \_\_attribute\_\_ ((interrupt, keep\_interrupts\_masked,

use\_debug\_exception\_return)) v6 ();

void \_\_attribute\_\_ ((interrupt, use\_shadow\_register\_set,

keep\_interrupts\_masked,

use\_debug\_exception\_return)) v7 ();

void \_\_attribute\_\_ ((interrupt("eic"))) v8 ();

void \_\_attribute\_\_ ((interrupt("vector=hw3"))) v9 ();