6

Подсистема прерывания (IC)

Целью лабораторной работы является **разработка и внедрение подсистемы прерывания** в ранее созданный однотактный процессор с архитектурой RISC-V

- Прерывания/Исключения
- Control and Status Registers (CSR) в архитектуре RISC-V
- Реализация прерываний в архитектуре RISC-V
- Структура разрабатываемого устройства
- Пример обработки прерывания
- Верификация процессора с подсистемой прерывания

Прерывания/Исключения

С компьютером постоянно происходят события, на которые он должен реагировать, запуская соответствующие подпрограммы, например, при движении мышки нужно перерисовать ее курсор на новом месте или реагировать на подключение новой флешки и так далее. Возможность запускать нужные подпрограммы в ответ на различные события, возникающие внутри или снаружи компьютера, существенно расширяют его возможности. События, требующие внимания процессора называются **прерываниями** (interrupt). Происходящие события формируют запрос на прерывание процессору.

Система прерывания – это совокупность программно-аппаратных средств, позволяющая процессору (при получении соответствующего запроса) на время прервать выполнение текущей программы, передать управление программе обслуживания поступившего запроса, а по завершении последней продолжить прерванную программу с того места, где она была остановлена.

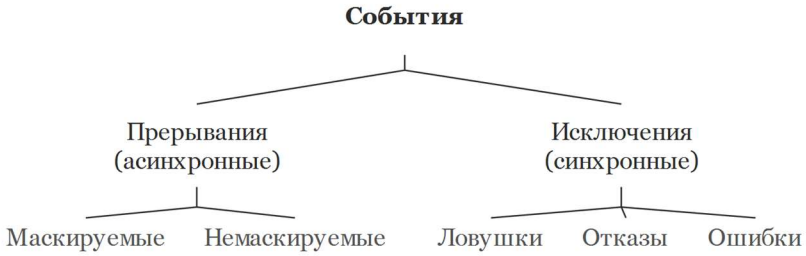
Под аппаратными событиями понимается одно из четырех состояний, возникших на соответствующем входе системы прерывания. В зависимости от конфигурации системы прерывания событием может быть либо появление 0 на входе, либо – 1, либо возникновение фронта сигнала, то есть его переключение из 0 в 1, либо появление спада, то есть переключение сигнала из 1 в 0.

Прерывания делятся на маскируемые, которые при желании можно игнорировать, и немаскируемые, которые игнорировать нельзя, например,

критический сигнал от датчика температуры, сообщающий, что в скором времени произойдет разрушение процессора.

Прерывание похоже на незапланированный вызов функции, вследствие события в аппаратном обеспечении. Программа (функция), запускаемая в ответ на прерывание, называется **обработчик прерывания**.

События могут быть не только аппаратными, но и программными – синхронными. Такие события называются **исключениями** (exception). Программа может столкнуться с состоянием ошибки, вызванным программным обеспечением, таким как неопределенная инструкция, неподдерживаемая данным процессором, в таком случаях говорят, что возникло исключение. Исключения программного обеспечения иногда называют **ловушками** (trap). К исключениям также относятся сброс, деление на ноль, переполнение и попытки считывания несуществующей памяти. Как и любой другой вызов функции, при возникновении прерывания или исключения необходимо сохранить адрес возврата, перейти к программе обработчика, выполнить свою работу, очистить после себя (не оставить никаких следов) и вернуться к программе, которую прервали.



Благодаря исключениям можно реализовать имитацию наличия каких-то аппаратных блоков программными средствами. Например, при отсутствии аппаратного умножителя, можно написать программу обработчика исключения неподдерживаемой инструкции умножения, реализующую алгоритм умножения через сложение и сдвиг. Тогда, каждый раз, когда в программе будет попадаться инструкция умножения, будет возникать исключение, приводящее к запуску обработчика, перемножающего числа и размещающего результат в нужные ячейки памяти. После выполнения обработчика управление возвращается программе, которая даже не поймет, что что-то произошло и умножитель «ненастоящий».

Современные процессоры, предусматривающие запуск операционной системы, обладают несколькими уровнями привилегий выполнения инструкций. Это значит, что существуют специальный регистр, определяющий режим, в котором в данный момент находится вычислительная машина. Наличие определенного значения в этом регистре устанавливает определенные ограничения для выполняемой в данный момент программы что-либо делать. Как правило выделяется 4 режима работы, в порядке убывания возможностей и увеличения ограничений:

машинный (machine mode), в котором можно всё;

гипервизора (hypervisor mode), который поддерживает виртуализацию машин, то есть эмуляцию нескольких машин (потенциально с несколькими операционными системами), работающих на одной физической машине;

привилегированный (supervisor mode), для операционных систем, с возможностью управления ресурсами;

пользовательский (user mode), для прикладных программ, использующих только те ресурсы, которые определила операционная система.

Переключение между этими режимами происходит с помощью еще одного вида исключений, называемого **системный вызов**, и который происходит при выполнении специальной инструкции. Для RISC-V такой инструкцией является **ecall**. Это похоже на вызов подпрограммы, но при системном вызове изменяется режим работы и управление передается операционной системе, которая, по коду в инструкции вызова определяет, что от нее хотят. Например, операционная система может предоставить данные с диска, так как запускаемая программа не имеет никакого представления о том, на какой машине ее запустили, или что используется какая-то конкретная файловая система.

Системы прерываний имеет ряд характеристик, которые варьируются в зависимости от их реализации. Все системы можно условно разбить на две категории: обзорные (прямые) и векторные.

В **обзорных системах прерывания** любое событие прерывания приводит к запуску одного и того же обработчика. Внутри такого обработчика прерывания определяется причина его возникновения (как правило — это число в специальном регистре), и уже в зависимости от причины запускается нужная подпрограмма. Обзорные системы аппаратно проще векторных, но требуют больше рутины и времени на обработку.

В **векторных системах прерывания** разные события приводят к запуску на исполнение разных программ обработчиков. Адрес начала обработчика прерывания называется **вектором прерывания**. В векторных системах прерывания выделяется фрагмент памяти, в котором хранятся адреса переходов на начало каждого из обработчиков. Такой участок памяти называется **таблицей векторов прерываний**.

В самом простом случае система прерывания позволяет обрабатывать только одно прерывание за раз. Существуют реализации позволяющие во время обработки прерывания «отвлекаться» на другие события. В таких системах используется система приоритетов, чтобы прерывание с более низким приоритетом не прерывало более приоритетное.

Control and Status Registers

Для поддержания работы операционной системы, виртуализации, системы прерывания и тому подобное, в архитектуре RISC-V предусмотрено использование группы регистров, под общим названием Control and Status Registers (**CSR**), обеспечивающих управление элементами процессора и доступ к статусной информации о системе. С помощью этих регистров реализуются привилегированные режимы работы процессора, хранение указателей на различные программные стеки, статус различных подсистем, регистры для обеспечения работы прерываний и многое другое.

Все регистры имеют уникальные 12-битные адреса, а их роли определены в спецификации на архитектуру RISC-V. В таблице ниже, в качестве примера, приводится фрагмент спецификации, определяющий роль некоторых из регистров. В левом столбике указан 12-битный адрес. Далее указывается в каком режиме, что можно делать с этим регистром. Далее идет название, а в правом столбике описание.

В этом примере можно увидеть регистры для сохранения адреса возврата из прерывания, адрес вектора прерывания (в идеологии RISC-V исключения называется ловушками – trap), регистры причины (cause), статус устройства для чисел с плавающей точкой, различные таймеры и это далеко не всё.

Number	Privilege	Name	Description
		User Trap Setup	
0x000	URW	ustatus	User status register.
0x004	URW	uie	User interrupt-enable register.
0x005	URW	utvec	User trap handler base address.
		User Trap Handling	
0x040	URW	uscratch	Scratch register for user trap handlers.
0x041	URW	uepc	User exception program counter.
0x042	URW	ucause	User trap cause.
0x043	URW	utval	User bad address or instruction.
0x044	URW	uip	User interrupt pending.
		User Floating-Point CSRs	
0x001	URW	fflags	Floating-Point Accrued Exceptions.
0x002	URW	frm	Floating-Point Dynamic Rounding Mode.
0x003	URW	fcsr	Floating-Point Control and Status Register (frm + fflags).
		User Counter/Timers	
0xC00	URO	cycle	Cycle counter for RDCYCLE instruction.
0xC01	URO	time	Timer for RDTIME instruction.
0xC02	URO	instret	Instructions-retired counter for RDINSTRET instruction.
0xC03	URO	hpmcounter3	Performance-monitoring counter.
0xC04	URO	hpmcounter4	Performance-monitoring counter.
		⋮	
0xC1F	URO	hpmcounter31	Performance-monitoring counter.
0xC80	URO	cycleh	Upper 32 bits of cycle, RV32 only.
0xC81	URO	timeh	Upper 32 bits of time, RV32 only.
0xC82	URO	instreth	Upper 32 bits of instret, RV32 only.
0xC83	URO	hpmcounter3h	Upper 32 bits of hpmcounter3, RV32 only.
0xC84	URO	hpmcounter4h	Upper 32 bits of hpmcounter4, RV32 only.
		⋮	
0xC9F	URO	hpmcounter31h	Upper 32 bits of hpmcounter31, RV32 only.

Для работы с регистрами CSR используются специальные инструкции **SYSTEM** (1110011) I-типа, хранящие в 12-битном поле **imm** адрес регистра, к которому будет осуществлен доступ и адреса в регистровом файле от куда будет считан или куда будет записан один из регистров CSR. В лабораторной работе «Основной дешифратор» инструкции с этим кодом операции были выставлены как **no operation**. В этой лабораторной работе ему надо будет добавить функциональности.

Так как стандартный ISA RISC-V выделяет 12-битное пространство кодирования для адресации, то в CSR может содержаться 4096 регистров. По соглашению, верхние 4 бита адреса CSR используются для кодирования доступности CSR для чтения и записи в соответствии с уровнем привилегий: два верхних бита **imm[11:10]** указывают, доступен ли регистр для чтения/записи или только для чтения, следующие два бита **imm[9:8]** кодируют самый низкий уровень привилегий, который может получить доступ к CSR.

Для реализации простейшей системы прерывания на процессоре с архитектурой RISC-V достаточно реализовать 5 регистров CSR работающих в машинном, самом привилегированном режиме и 4 инструкции для работы с этими регистрами.

		Machine Trap Setup	
0x304	MRW	mie	Machine interrupt-enable register.
0x305	MRW	mtvec	Machine trap-handler base address.
0x340	MRW	mscratch	Scratch register for machine trap handlers.
0x341	MRW	mepc	Machine exception program counter.
0x342	MRW	mcause	Machine trap cause.

По адресу 0x304 должен располагаться регистр, позволяющий маскировать прерывания. Например, если на 5-ом входе системы прерывания генерируется прерывание, то процессор отреагирует на него только в том случае, если 5-ый бит регистра **mie** будет равен 1.

Регистр **mtvec** является базовым адресом обработчика прерывания. Это значит, что предусмотрена возможность реализации как обзорной (прямой), так и векторной системы прерывания. В первом случае при возникновении прерывания в program counter загружается значение **mtvec**. Во втором случае, в **program counter** загружается сумма регистра базового адреса **mtvec** и регистра причины прерывания **mcause**, который обновляется каждый раз, когда происходит прерывание, значение в нем несет информацию о том, что именно произошло в системе.

Так как обработчик прерывания будет использовать те же регистры, что и прерванная программа, то перед использованием регистрового файла, данные из него необходимо сохранить, разместив их на стеке. Стек для прерывания находится не там же, где программный стек, а адрес начала этого стека хранится в регистре **mscratch** и по сути является указателем на верхушку стека. Регистр **mepc** сохраняет адрес инструкции, во время которой произошло прерывание или исключение. Именно текущий адрес **PC**, а не следующей за ним инструкции **PC + 4** надо сохранять потому, что иногда надо продолжить с этой же инструкции, а иногда со следующей. В данном случае, чтобы сохранить простоту реализации, при возникновении сигнала прерывания **INT** результат текущей инструкции не должен быть записан, а после обработки прерывания надо продолжить с этой же инструкции по адресу **PC** (так как ее результат не был записан).

Когда процессор включается, программа первым делом должна инициализировать все требуемые CSR регистры, в частности:

задать маску прерывания **mie**,

задать адрес вектора прерывания **mtvec**,

задать адрес вершины стека прерываний **mscratch**.

Ниже приводится таблица со списком из четырех инструкций, поддержку которых требуется реализовать, для поддержания работы CSR и системы прерывания.

opcode	func3	Тип	Инструкция	Описание	Операция
1110011	000	I	mret	Возврат из прерывания	PC = mepc
1110011	001	I	csrrw rd, csr, rs1	Чтение/Запись CSR	rd = csr, csr = rs1
1110011	010	I	csrrs rd, csr, rs1	Чтение/Установка бит CSR	rd = csr, csr = csr | rs1
1110011	011	I	csrrc rd, csr, rs1	Чтение/Очистка бит CSR	rd = csr, csr = csr & ~rs1

Непосредственно с прерыванием связана только одна инструкция – **mret**, которая загружает в **PC** значение регистра **mepc**, в котором хранится адрес, на котором была прервана программа.

Остальные операции считывают значение одного из регистров CSR в регистровый файл, при этом, инструкция **csrrw** еще записывает значение из регистрового файла в CSR. Инструкция **csrrs** выполняет логическое ИЛИ между содержимым регистров CSR и регистрового файла, тем самым устанавливая в регистре CSR единицу в тех же битах, что и у считываемого регистра. Операция **csrrc** приводит к очищению битов, значения которых в считываемом из регистрового файла регистре были равны 1.

Для удобства программирования на языке ассемблера RISC-V существуют псевдоинструкции для работы с регистрами CSR.

Псевдоинструкция	Инструкция RISC-V	Описание	Операция
csrr rd, csr	csrrs rd, csr, x0	Чтение CSR	rd = csr
csrw csr, rs1	csrrw x0, csr, rs1	Запись CSR	csr = rs1

Операция логического ИЛИ нулевого регистра с содержимым регистра CSR не меняет его содержимого, поэтому при использовании инструкции **csrr** происходит только операция чтения. Подобным образом реализована псевдоинструкция **csrw**.

Реализация прерываний в архитектуре RISC-V

Процессор RISC-V может работать в одном из нескольких режимов выполнения с различными уровнями привилегий. Машинный режим – это самый высокий уровень привилегий; программа, работающая в этом режиме, может получить доступ ко всем регистрам и ячейкам памяти. M-режим является единственным необходимым режимом привилегий и единственным режимом, используемым в процессорах без операционной системы, включая многие встроенные системы.

Обработчики прерываний/исключений используют четыре специальных регистра, называемых регистрами управления и состояния (CSR), для обработки исключения: **mtvec**, **mcause**, **mepc** и **mscratch**. Регистр базового адреса вектора прерывания **mtvec**, содержит адрес кода обработчика прерывания. При возникновении прерывания или исключения процессор:

записывает причину исключения в **mcause**,

сохраняет **PC** инструкции, на которой произошло событие, в **mepc**,

переходит к обработчику прерывания, загружая в **PC** адрес, предварительно настроенный в **mtvec**.

После перехода по адресу в **mtvec** обработчик считывает регистр **mcause**, чтобы проверить, что вызвало прерывание или исключение, и реагирует соответствующим образом (например, считывая клавиатуру при аппаратном прерывании).

После выполнения программы обработчика прерывания возвращение в программу, выполняется командой возврата **mret**, которая помещает в **PC** значение регистра **mepc**. Сохранение **PC** инструкции при прерывании в **mepc** аналогично использованию **ra** для хранения обратного адреса во время инструкции **jal**. Обработчики прерываний должны использовать программные регистры (**x1**–**x31**) для своей работы, поэтому они используют память, на которую указывает **mscratch**, для хранения и восстановления этих регистров.

Обработчики прерываний используют специальные инструкции, которые называются привилегированными, потому что они получают доступ к CSR. Они являются частью базового набора инструкций **RV32I**. Регистры **mepc** и **mcause** не являются частью программных регистров RISC-V (**x1**–**x31**), поэтому обработчик прерываний должен переместить эти регистры специального назначения (CSR) в регистровый файл для чтения и работы с ними.

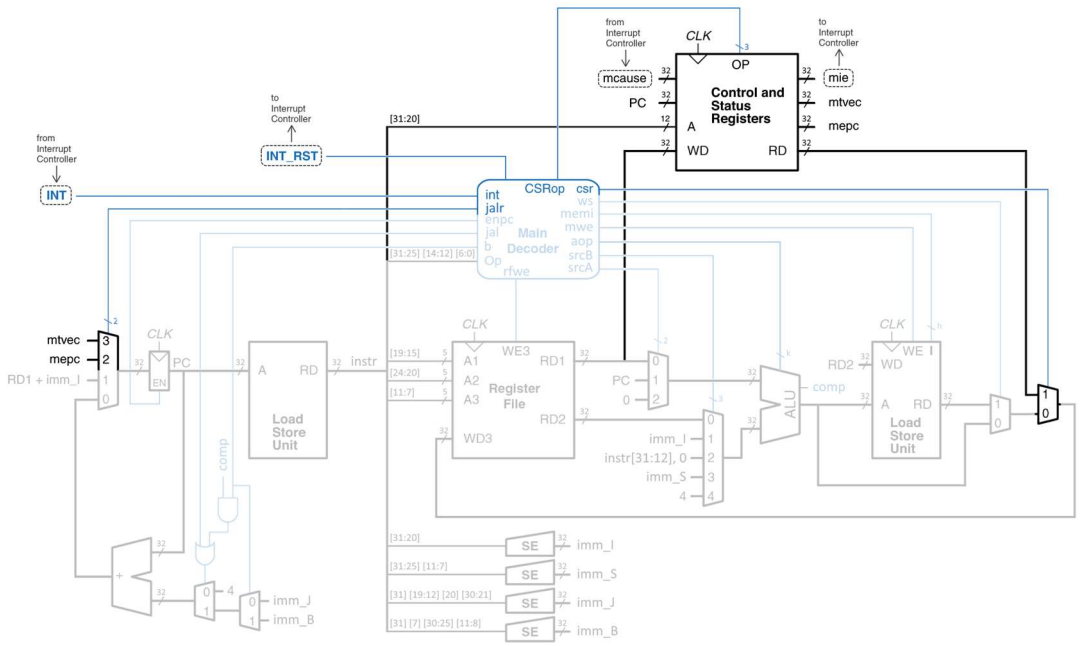
Обновление содержимого регистра причины **mcause** обеспечивает контроллер прерывания. Контроллер прерывания – это блок процессора, обеспечивающий взаимодействие с устройствами, запрашивающими прерывания, формирование кода причины прерывания для процессора, маскирование прерываний, а также, в других реализациях, может реагировать на прерывания в соответствии с приоритетом и тому подобное.

Каждое периферийное устройство, которое может сгенерировать прерывание, подключается к контроллеру прерывания по одной из 32 пар проводов: запрос на прерывание (**int_req**) и прерывание обслужено (**int_fin**). Например, подключили клавиатуру к 7-ой паре. Когда на клавиатуру нажимают, код этой клавиши попадает в буферный регистр с дополнительным управляющим битом, выставленным в единицу, который подключен к входу запроса на прерывание. Если прерывание не замаскированно, то есть в данном примере 7-ой бит регистра **mie** выставлен в 1, то контроллер прерывания сгенерирует соответствующий код причины (например, 7 или 7*4, если реализуется векторная система прерывания [по желанию студента]). Кроме этого, контроллер прерывания выдаст сигнал **INT** прямо в устройство управления процессора, чтобы оно узнало, что произошло прерывание и разрешило обновить содержимое регистра причины **mcause**, сохранило адрес прерванной инструкции в **mepc** и загрузило в **pc** вектор прерывания **mtvec**.

Когда будет выполняться инструкция **mret**, устройство управления подаст сигнал контроллеру прерывания, чтобы тот, в свою очередь, направил его в виде сигнала «прерывание обслужено» для соответствующего устройства. После этого периферийное устройство обязано снять сигнал запроса прерывания хотя бы на один такт. В нашем примере сигнал «прерывание обслужено» может быть подключен непосредственно к сбросу буферного регистра клавиатуры.

Структура разрабатываемого устройства

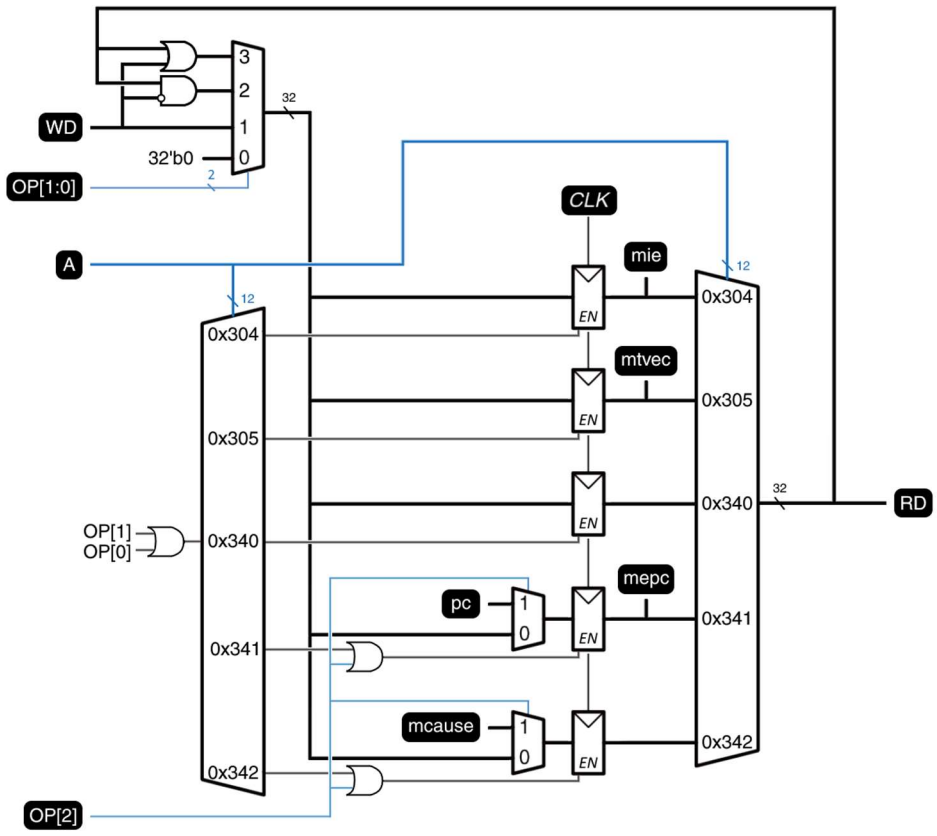
В рамках лабораторной работы необходимо реализовать поддержку обработки аппаратных прерываний. Для этого необходимо реализовать для два аппаратных блока: **Control and Status Registers** и контроллер прерываний (**Interrupt Controller**). Интеграция первого из двух блоков потребует небольших модификаций в основном дешифраторе и тракте данных. Подключение **Control and Status Registers** демонстрируется на рисунке ниже. В серых тонах изображены блоки не претерпевшие изменения.



Блок **Control and Status Registers** имеет 12-битный вход **A** для указания адреса регистра, к которому будет произведено обращение. Этот вход подключен к старшим 12 битам инструкции, что соответствует положению в ней поля **imm** инструкции типа **I**. Вход **WD** (write data – записываемые данные) подключается к первому порту чтения из регистрового файла **RD1**. На входе данных порта записи регистрового файла **WD3** появился мультиплексор позволяющий считывать данные из CSR.

3-битный вход **OP** определяет операцию, которая будет производиться над содержимым CSR по адресу **A**. Им управляет основной дешифратор. Остальные входы и выходы подключены непосредственно ко входам или выходам соответствующих регистров.

Рассмотрим один из вариантов организации блока **Control and Status Registrs**. Основную часть схемы занимают мультиплексор, обеспечивающий дешифрацию адреса и подачу на выход **RD** значения соответствующего регистра, и демультиплексор дешифрующий адрес и передающий сигнал разрешения на запись **enable** (EN) на тот же регистр.

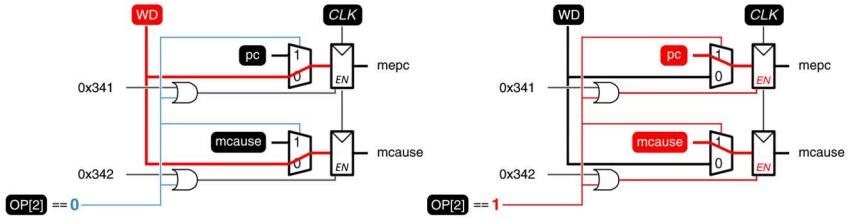


Для реализации мультиплексора на языке описания аппаратуры Verilog HDL можно воспользоваться конструкцией **case** внутри блока **always @ (*)**. Для реализации демультиплексора также можно использовать **case**, но ввиду того, что он управляет значением внутри регистров (слева от знака равно стоят регистры), то его необходимо описывсать внутри блока **always @ (posedge clk)**.

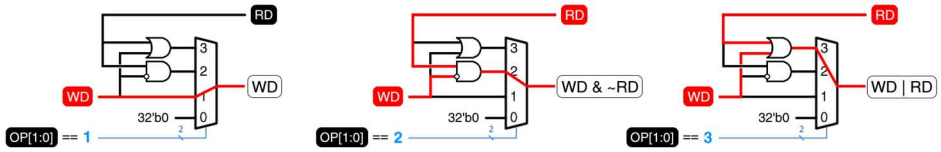
Мультиплексоры, располагаемые на входах регистров **mepc** и **mcause** нужны, чтобы при возникновении сигнала прерывания сразу же разрешить

обновить значение этих регистров значением **PC**, на котором произошло прерывание и кодом причины происходящего сейчас прерывания.

Старший бит 3-битного сигнала **OP** управляет обновлением информации в регистрах **mepc** и **mcause**. На картинке ниже демонстрируется перемещение данных в случаях, когда сейчас не момент возникновения прерывания **OP[2] == 0**, и момент возникновения прерывания **OP[2] == 1** (сигнал приходит от устройства управления). Обратите внимание, что входы **enable** регистров подключены по схеме ИЛИ с сигналом начала обработки прерывания.



Два младших бита сигнала **OP**, в свою очередь, управляют мультиплексором 4 в 1, который выбирает, что именно будет записано в регистр CSR: либо это просто значение из регистрового файла, либо результат операции логического ИЛИ между регистром из CSR и значением из регистрового файла, либо операция И с инверсией второго операнда. На рисунке ниже продемонстрировано движение данных при разных значениях управляющего сигнала.



Обратите внимание, что неважно чем будет занят нулевой вход этого мультиплексора, потому что сигнал, подаваемый на вход демультиплексора, который, в свою очередь, направляет его на вход **enable** одного из регистров, это результат логического ИЛИ между теми же управляющими битами **OP[1]** и **OP[0]**. Это значит, когда **OP[1:0] == 00**, то на все входы **enable** приходит **OP[1] | OP[0] == 0**, а значит не важно, что будет на входах регистров, так как в них не будет произведена запись.

Опишите блок **Control and Status Registers** отдельным verilog-модулем, после чего интегрируйте его в существующий модуль процессора. Процессор претерпит следующие изменения:

1. Мультиплексор на входе **PC** расширится до 4 входов, чтобы иметь возможность загружать в него значение адреса возврата и вектор прерывания;

2. Сигнал **jalr** стал 2-битным, что влечет соответствующие изменения в основном дешифраторе;

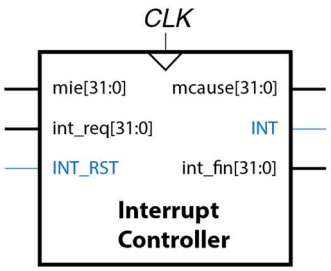
3. Появился еще один мультиплексор на входе **WD3** регистрового файла, управление которым также требует изменений в основном дешифраторе;

4. У процессора появляется два новых входа: **INT** (сигнал о том, что произошло прерывание) и **mcause** (код причины прерывания), и два новых выхода: **INT_RST** (прерывание обработано) и **mie** (маска прерывания);

5. Основной дешифратор должен реагировать на возникновение сигнала прерывания **INT** организуя необходимые сервисные действия;

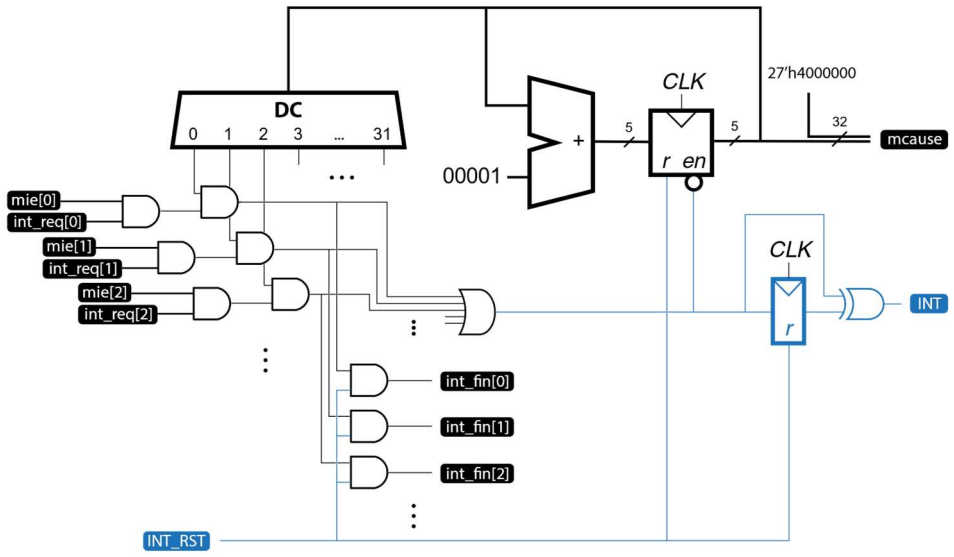
6. Основной дешифратор должен поддерживать выполнение описанных ранее инструкций для работы с регистрами CSR, выдавая **INT_RST == 1** при выполнении инструкции **mret**.

Рассмотрим один из способов реализации простейшего контроллера прерываний с циклическим опросом. Представленный ниже контроллер имеет три входа и три выхода: 32-битный вход на который подается содержимое регистра маски **mie;** 32-битный вход для 32 разных сигналов запроса на прерывание **int_req;** однобитный вход **INT_RST** передаваемый на один из 32 выход **int_fin** – «прерывание обработано», для которого закончилась работа обработчика прерывания; выход **INT** выдающий единицу в течении такта, когда произошло прерывание; **mcause** – код причины прерывания.



В основе контроллера прерывания с циклическим опросом, схема которого изображена на картинке далее, лежит счетчик (регистр + сумматор). Выход счетчика одновременно является кодом причины прерывания **mcause** и подается на вход дешифратора **DC**. Дешифратор выдает 1 только на том выходе, код которого подается на его вход. Например, при подаче на вход дешифратора числа 17, на его 17-ом выходе будет 1, а на остальных – 0.

Каждый выход **n** дешифратора **DC** логически умножается на сигнал запроса прерывания **int_req[n]** от какого-нибудь из периферийных устройств и соответствующий бит маски прерывания **mie[n]**. Счетчик работает по кругу до тех пор пока не наткнется на незамаскированное прерывание, то есть, когда значение счетчика будет равняться **n** и **int_req[n] == 1** и **mie[n] == 1**, на n-ом входе логического ИЛИ появится 1, значит и на его выходе появится 1, которое запретит запись в регистр счетчика, зафиксировав тем самым значение кода причины, а благодаря схеме из триггера и Исключающего ИЛИ на выходе, сигнал **INT** станет единицей только на один такт.



Когда обработчик прерывания закончит свою работу исполнение инструкции **mret** приведет к формированию сигнала **INT_RST** на входе контроллера прерывания. За счет логического И этот сигнал попадет только на тот выход «прерывание обработано» **int_fin[n]** для которого обрабатывалось прерывание, и на вход сброса регистра счетчика.

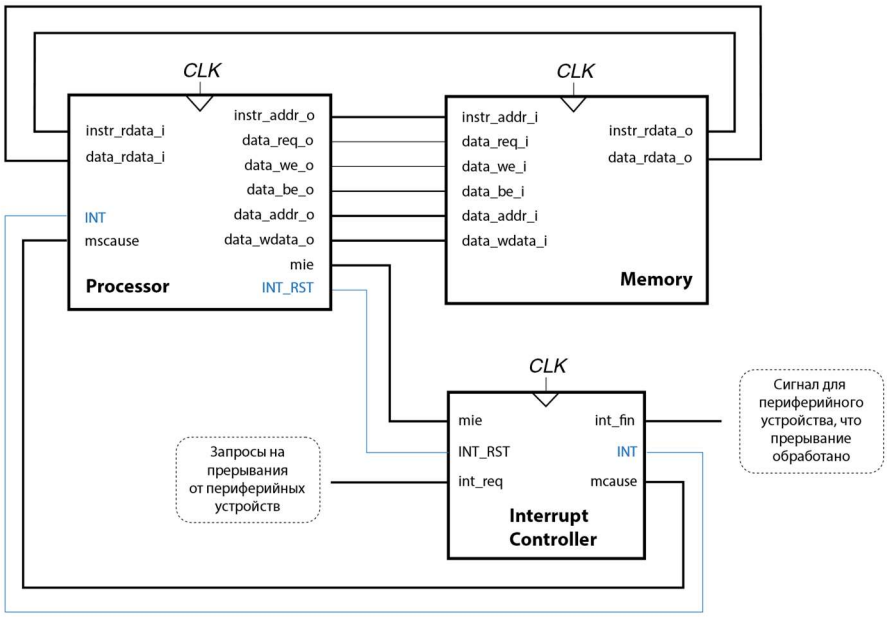
Именно такая реализация контроллера прерываний не позволяет изменять маску прерывания, из прерывания обработка которого происходит в данный момент. Однако, при желании схему можно легко модифицировать, чтоб устранить этот нюанс. Также немного доработав тракт данных, можно реализовать не обзорную, а векторную систему прерывания, что значительно увеличит скорость обработки прерывания, так как не надо будет тратить время на проверку кода причины прерывания с последовательным подбиранием подходящей программы обработчика.

Еще одной особенностью данной реализации является отсутствие возможности совершить еще одно прерывание во время обработки прерывания. Как говорилось ранее, это потребует значительной переработки и введения концепции приоритета прерывания.

Разрабатываемое устройство обрабатывает только аппаратные прерывания, но с помощью доработки основного дешифратора и регистров CSR можно реализовать поддержку обработки исключений.

В лабораторной работе требуется разработать блок **Control and Status Registers**, который необходимо интегрировать в однотактный процессор RISC-V немного доработав его микроархитектру, разработать блок контроллера прерывания (**Interrupt Controller**) и подключить его к существующей системе процессор-память, как это продемонстрировано на рисунке далее. Пояснение работы интерфейса между памятью и процессором приводится в прошлой лабораторной работе. Вся эта структура должна быть описана в файле самого верхнего уровня **miriscv_top.sv**.

Реализовывать в контроллере прерывания возможность подключения 32 устройств не обязательно и обработки 5 сигналов на запрос прерывания вполне достаточно для этих лабораторных работ.



Пример обработки прерывания

Ниже представлен пример программы и обработчика прерывания. Программа начинается с инициализации начальных значений регистров управления, указателя на верхушку стека и глобальную область данных, после чего уходит в бесконечный цикл ничего не делая, до тех пор, пока не произойдет прерывание.

Для данного примера представим, что к контроллеру прерываний подключено две условные кнопки: одна к 5-ому входу прерывания, вторая – к 19-ому. Обработчик прерывания сначала сохраняет значения используемых регистров на стек → проверяет регистр причины, чтобы запустить необходимую подпрограмму для конкретного прерывания → выполняет полезную задачу, связанную с этим прерывание → восстанавливает значения регистрового файла → возвращает управление прерванной программе. Если бы система прерывания была векторная, то рутина со считыванием кода причины отсутствовала.

Для примера пусть прерывание 5 будет прибавлять число 3 к некоторой глобальной переменной, а прерывание 19 будет делить это же число пополам.

Пример находится на следующей странице.

Верификация процессора с подсистемой прерывания

Для верификации полученного решения необходимо написать свою программу (творческое задание), которая что-либо делала бы в основном цикле работы и при этом имела возможность обработки нескольких прерываний (количество на свое усмотрение). Так же потребуется создать тестовое окружение, формирующее для полученной системы сигналы: сброса, тактирования и прерывания. Последний включает в себя формирование запросов на прерывание и соответствующую реакцию на получения сигнала «прерывание обслужено».

Проверка правильной работы, как и демонстрация этого преподавателю, будет происходить либо с использованием временных диаграмм, сопоставляя происходящие на них с ожиданиями, либо автоматизировано с выводом результатов в терминал (на усмотрение студента). Первый подход потребует более красноречивого описания процессов, происходящих на этих диаграммах. На практике временные диаграммы используются как вспомогательный инструмент и не могут с достаточной точностью гарантировать правильную работу устройства. Автоматический анализ реакции внутренних блоков процессора на внешние сигналы – более предпочтительный подход.

При выполнении любой сложной умственной работы, например интеграции системы прерывания в однотактный процессор, главное – не торопиться. Последовательный, неспешный анализ новой информации и регулярное возвращение к этим вопросам – лучшая стратегия.

```
                # Инициализируем начальные значения регистров
    li    sp, 0xFFFFFFFC        # устанавливаем указатель на верхушку стека
    li    gp, 0x10000000        # устанавливаем указатель на глобальные данные

    li    t0, 0x00080020        # подготавливаем маску прерывания для 5 и 19 входов
    csrw  mie, t0               # загружаем маску в регистр маски
    la    t0, interrupt         # аналогично li загружает число, в данном случае - адрес
    csrw  mtvec, t0             # устанавливаем вектор прерывания
    li    t0, 0xEFFFFFFC        # готовим адрес верхушки стека прерывания
    csrw  mscratch, t0          # загружаем в указатель на верхушку стека прерывания

    li    t0, 1                 # начальное значение глобальной переменной
    sw    t0, 0(gp)             # загружаем переменную в память

    li    t1, 0                 # начальное значение, чтобы в симуляции не было xxx
    li    t2, 0                 # начальное значение, чтобы в симуляции не было xxx

while:                          # бесконечный цикл, аналогичный while (1);
    beq x0, x0, while           # ничего не делаем

                # ОБРАБОТЧИК ПРЕРЫВАНИЯ
                # Сохраняем используемые регистры на стек
interrupt:
    csrrw t0, mscratch, t0      # меняем местами mscratch и t0
    sw    t1, 0(t0)             # сохраняем t1 на стек mscratch
    sw    t2, 4(t0)             # сохраняем t2 на стек mscratch

                # Проверяем регистр причины и на 5-ое прерывание
    csrr  t1, mcause            # t1 = mcause
    li    t2, 5                 # t2 = 5 (код одного из прерываний)
    bne   t1, t2, nineteen      # если это не 5 прерывание, то проверяем 19
                # Обработчик 5-го прерывания
    lw    t2, 0(gp)             # загружаем переменную из памяти
    addi  t2, t2, 3             # прибавляем к значению 3
    sw    t2, 0(gp)             # возвращаем переменную в память
    j     done                  # идем возвращать регистры и на выход

nineteen:       # Проверяем на 19-ое прерывание
    li    t2, 19                # t2 = 19 (код другого прерывания)
    bne   t1, t2, done          # если не 19-ое, то выходим
                # Обработчик 19-го прерывания
    lw    t2, 0(gp)             # загружаем переменную из памяти
    srli  t2, t2, 1             # делим число пополам сдвигом вправо
    sw    t2, 0(gp)             # возвращаем переменную в память
    j     done                  # идем возвращать регистры и на выход

                # Возвращаем регистры на места и выходим
done:
    lw    t1, 0(t0)             # возвращаем t1 со стека
    lw    t2, 4(t0)             # возвращаем t2 со стека
    csrrw t0, mscratch, t0      # меняем обратно местами t0 и mscratch
    mret                        # возвращаем управление программе (pc = mepc)
```