ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ И ОТЛАДКИ ПРОГРАММ MC STUDIO

ИНСТРУМЕНТЫ RISCORE32

(ПРИЛОЖЕНИЕ 1)

СИСТЕМА КОМАНД

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Система команд RISCore32 полностью соответствует архитектуре MIPS32.

1.1 Классы и форматы команд

Все команды RISCore32 являются однословными (32 бита) и выровнены на границу слова. На рис.1 представлены основные форматы команд.

op	код операции (6 бит)
rs	№ регистра-источника (5 бит)
rt	№ регистра-источника/назначения или для команд с первичным кодом операции «REGIMM» (5 бит)
immediate	непосредственный знаковый операнд (16 бит), используется для логических операндов, арифметических знаковых операндов, для смещений в адресе загруски/сохранения, для PC зависимых команд ветвления
target	адрес перехода (26 бит)
rd	№ регистра-назначения (5 бит)
sa	величина сдвига (5 бит)
funct	поле функции (6 бит), используется для определения команд с первичным кодом операции «SPECIAL»

Рисунок 1. Форматы команд RISCore

Команды RISCore делятся на следующие 5 классов:

- команды загрузки/записи (I-формат) пересылают данные между памятью и общими регистрами. Адрес памяти равен сумме регистра *base* и непосредственного 16-битового смещения *offset*;
- вычислительные команды выполняют арифметические, логические и сдвиговые операции над значениями регистров (команды R-формата). Если один из операндов *immediate*, применяется I-формат;
- команды переходов/ветвлений меняют ход выполнения программы. Адрес перехода представлен 26-битовым *target* (J-формат) либо содержимым регистра (R-тип, для диспетчеризации и возвратов). Ветвления имеют 16-битовое смещение относительно PC (I-тип). Переходы с возвратом сохраняют адрес возврата в регистре 31;

- сопроцессорные команды выполняют операции в сопроцессорах. Сопроцессорные загрузки/сохранения имеют I-формат. Команды CP0 манипулируют устройствами управления памяти и обработки исключений процессора;
- специальные команды (R-формат) выполняют различные задачи, включая пересылки данных между специальными и общими регистрами и прерывания.

1.1.1 Условные обозначения в изображении команды

Все переменные поля в формате команды (*rs, rt, immediate* и др.) обозначаются малыми буквами; мнемоники полей и подполей имени команды пишутся заглавными буквами. Для большей ясности возможны замены имен переменных полей определенных команд. Например, используется *base* вместо *rs* в формате команд загрузки и записи. Чтобы указать, что поле переменное, замененное имя пишется малыми буквами.

В описании каждой команды в разделе *Операция* описывается на языке высокого уровня выполнение команды; специальные символы, используемые при описании, приведены в таблице 1.

Таблица 1. Обозначения в операциях команд RISCore32

Символ	Значение
←	Присваивание
||	Соединение в битовую строку
x^y	Тиражирование x(обычно 1 бит) в битовую строку длиной y
$x_{Y..Z}$	Битовая выборка от y до z в битовой строке x. Обычно, изображение Little-endian. При y<x строка пуста (длина 0)
+	Сложение в дополнительном коде или с плавающей запятой
-	Вычитание в дополнительном коде или с плавающей запятой
*	Умножение в дополнительном коде или с плавающей запятой
div	Деление целых в дополнительном коде
mod	Значение по модулю в дополнительном коде
/	Деление с плавающей запятой
<	Сравнение Меньше
and	Поразрядное логическое умножение
or	Поразрядное логическое сложение
xor	Поразрядное логическое исключающее сложение
nor	Поразрядное логическое НЕ-ИЛИ
GPRLEN	Длина регистров общего назначения в битах(32 или 64)
GPR[*x*]	Общий регистр x. GPR[0] обычно содержит 0(неизменяемый)
CPR[*z,x*]	Общий регистр x сопроцессорного устройства z
CCR[*z,x*]	Управляющий регистр x сопроцессорного устройства z
COC[*z*]	Сигнал условия сопроцессорного устройства z
PC	Значение Счётчика комманд. Указывает на адрес выполняемой инструкции
BigEndianMem	Режим Big-Endian как задан при включении(0→Little,1→Big). Указывает endianess памяти и в режиме ядра.

Символ	Значение
ReverseEndian	Изменить endianess командам загрузки/записи; возможно лишь в режиме пользователя и с единичным RE битом Status рг (R300A). ReverseEndian вычисляется как (SR_{25} and User mode)
BigEndianRISCORE	Endianess для команд загрузки/записи (0→Little, 1→Big). В User режиме endianess можно изменить при SR_{25}=1. BigEndianRISCORE - это (BigEndianMem XOR ReversEndian).
T+i	Указывает временной шаг между операциями. Каждое положение внутри шага определяет исполнение операций последовательно (порядок изменяется условными и цикловыми конструкциями). Операции в T+i исполняются в командном i-такте от старта команды (если старт в j, то операции в T+i исполняются во время i + j). Порядок исполнения для двух операций двух одновременно выполняющихся команд не устанавливается.

1.1.2 Примеры изображения команды

Пример 1.

$$GPR[rt] \leftarrow immediate \ || 0^{16}$$

16 нулевых битов присоединяется справа к значению непосредственного операнда (обычно 16-битового), и 32-битовая строка помещается в GPR регистр *rt*.

Пример 2.

$$(immediate_{15})^{16} \ || immediate_{15..0}$$

Формируется значение расширенного знаком до 32 бит непосредственного операнда: знак (15-й бит) размножается на 16 позиций, результат присоединяется слева к 16 битам immediate.

1.2 Команды загрузки и записи

Все операции загрузки имеют задержку в 1 команду, так как непосредственно следующая за загрузкой команда, не может использовать значение регистра, который загружался данными из памяти. Исключение - команды LWL и LWR, у которых регистр назначения может загружаться предшествующей командой загрузки.

В таблице 2 даны операции загрузки/записи, используемые при работе с виртуальными адресами и физической памятью.

Таблица 2. Общие функциии загрузки/записи

Функция	Описание
AddressTranslation	TLB определяет физический адрес по данному виртуальному. Функция не исполняется и возникает исключение, если entry содержащей виртуальный адрес страницы не отображен в TLB.
LoadMemory	В кэш и основной памяти определяется слово по указанному физическому адресу. 2 младших бита адреса и поле Тип Выборки указывают загружаемые байты в слове данных. Если обращение кэшируемое, полное слово загружается в кэш.
StoreMemory	В кэш, буфер записи и основную память записывается слово (или часть) по данному физическому адресу. 2 младших бита адреса и поле Тип Выборки задают изменяемые байты в слове

Функция	Описание
SyncOperation	Функция упорядочивает загрузки из памяти, чтобы синхронизировать общедоступную память. Эта операция синхронизирует чтение и запись порядком, обозначенном stype, и они происходят в одинаковом порядке на всех процессорах
SignalException	Функция SignalException сообщает о условии исключения. Это действие приводит к исключению, которое прерывает команду. Псевдокод операции команды никогда не видит возврат из этого вызова функции.
NullifyCurrentInstruction	Аннулирует текущую команду. Команда прерывается. Для команд вероятного перехода нуллификация уничтожает команду в слоте задержки в течение выполнения
CoprocessorOperation	Выполняет определённую команду сопроцессора

Поле «Тип выборки» (см. таблицу 3) указывает, сколько байт в слове данных загружаются или запоминаются в памяти. Независимо от типа выборки и порядка нумерации байтов адрес указывает байт с самым маленьким байтовым адресом среди байтов в адресованном поле. Для Big-Endian это самый левый байт со знаком числа, для Little-Endian - самый незначащий правый байт.

Таблица 3. Поле Тип выборки для загрузок/записи

Мнемоника	Значение
DOUBLEWORD	двойное слово (64 бита)
WORD	слово (32 бита)
TRIPLEBYTE	три байта (24 бита)
HALFWORD	полслова (16 бит)
BYTE	байт (8 бит)

Байты, используемые в пределах двойного слова, определяются непосредственно типом выборки или 3 младшими битами адреса.

1.3 Команды перехода и ветвления

Все команды переходов и ветвлений имеют задержку точно на 1 команду. То есть, следующая непосредственно за переходом или ветвлением (и занимающая delay slot) команда всегда исполняется, пока команда назначения выбирается из памяти. В delay важно разместить не влияющую на результат вычислений команду.

Если исключение или прерывание препятствуют завершению команды в delay, в EPC регистре аппаратно устанавливается адрес команды перехода/ветвления, предшествующей прерванной команде. Если вычисления возобновляются, команда перехода/ветвления и следующая в delay будут исполнены вновь.

И так как выполнение команды перехода/ветвления может быть возобновлено после исключения или прерывания, следовательно, при запоминании возврата в команде перехода/ветвления рг31 не может использоваться как исходный регистр.

Поскольку команды должны быть выровнены на границу слова, значение регистра перехода в команде перехода по регистру должно иметь два нулевых младших бита. Иначе, позже при выборке команды назначения перехода возникнет исключение по адресу.

1.4 Команды сопроцессора

Архитектура RISCore32 обеспечивает 4 сопроцессорных устройства. Сопроцессоры - альтернативные вычислительные устройства, имеющие раздельные от RISCore32 регистровые файлы. Сопроцессоры имеют 2 регистровых пространства, каждое по 32 регистра (32-битовых). К первому относятся сопроцессорные регистры общего назначения, имеющие прямой доступ к памяти для загрузки/записи; их содержимое может передаваться между сопроцессором и процессором. Второе пространство регистров заполняют сопроцессорные управляющие регистры, их содержимое может передаваться между сопроцессором и процессором. Команды сопроцессора могут изменять регистры обоих пространств.

Обычно, принято сопроцессорный управляющий рг0 рассматривать как сопроцессорный регистр реализации и ревизии. Однако системный управляющий сопроцессор CP0 использует сопроцессорный регистр общего назначения рг15 в качестве процессорно/сопроцессорного регистра ревизии. Младший байт регистра (биты 7¸0) содержит № ревизии сопроцессорного устройства в виде ух, где у - номер старшей ревизии (7¸4 бит), а х - номер младшей ревизии в 3¸0 битах. Следующий байт (биты 15¸8) интерпретируются как описание реализации сопроцессорного устройства. Содержимое старшей половины регистра неопределено (читается 0 и при попытке записи будет 0).

1.5 Команды системного управляющего сопроцессора

На команды CP0 наложены некоторые специальные ограничения, диктуемые задачей объединения с RISCore32. Хотя команды загрузки/записи передают данные в/из сопроцессора и команды пересылок в/из управляющих регистров сопроцессора разрешены, CP0 требует некоторой защиты, так как ответственен за обработку исключений и управление памятью. Поэтому команды пересылок в/из управляющих регистров сопроцессора - это только механизм для чтения из и записи в регистры CP0.

Главная задача команд CP0 - чтение, запись, проверка TLB входов и изменение операционных режимов для подготовки возврата в режим потребителя или состояние разрешенного прерывания.

2. ОПИСАНИЕ СИСТЕМЫ КОМАНД

В данном разделе дается подробное описание выполнения каждой команды процессорного ядра RISCore32. Команды представляются в алфавитном порядке.

В конце описания каждой команды перечисляются возможные при ее выполнении исключения. Перечень команд, реализованных в RISCore32, приведен в таблицах 4 - 12.

Таблица 4. Команды Load, Store

Условное обозначение	Название
LB	Load Byte
LBU	Load Byte Unsigned
LH	Load Halfword
LHU	Load Halfword Unsigned
LW	Load Word
LWL	Load Word Left
LWR	Load Word Right
LL	Load Linked Word
LUI	Load Upper Immediate
SB	Store Byte
SH	Store halfword
SW	Store Word
SWL	Store Word Left
SWR	Store Word Right
SC	Store Conditional Word

Таблица 5. Команды ALU с непосредственным операндом

Условное обозначение	Название
ADDI	ADD Immediate
ADDIU	ADD Immediate Unsigned
SLTI	Set and Less Then Immediate
SLTIU	Set on Less Than Unsigned Immediate
ANDI	AND Immediate
ORI	OR Immediate
XORI	Exclusive OR Immediate
LUI	Load Upper Immediate

Таблица 6. Команды ALU, в которых используется три операнда

Условное обозначение	Название
ADD	Add
ADDU	Add Unsigned
CLO	Count Leading Ones
CLZ	Count Leading Zeroes
SUB	Subtract
SUBU	Subtract Unsigned
SLT	Set and Less Then
SLTU	Set on Less Than Unsigned
AND	And

Условное обозначение	Название
OR	Or
XOR	Exclusive OR
NOR	Nor

Таблица 7. Команды сдвига

Условное обозначение	Название
SLL	Shift Left Logical
SRL	Shift Right Logical
SRA	Shift Right Arithmetical
SLLV	Shift Left Logical Variable
SRLV	Shift Right Logical Variable
SRAV	Shift Right Arithmetical Variable

Таблица 8. Команды умножения и деления

Условное обозначение	Название
MUL	Multiply withregister write
MULT	Multiply
MULTU	Multiply Unsigned
MADD	Multiply-Add
MADDU	Multiply-Add Unsigned
MSUB	Multiply-Subtract
MSUBU	Multiply-Subtract Unsigned
DIV	Divide
DIVU	Divide Unsigned
MFHI	Move From HI
MFLO	Move From LO
MTHI	Move To HI
MTLO	Move To LO

Таблица 9. Команды Branch

Условное обозначение	Название
BEQ	Branch on Equal
BEQL	Branch on Equal Likely
BNE	Branch on Not Equal
BNEL	Branch on Not Equal Likely
BLEZ	Branch on Less than or Equal Zero
BLEZL	Branch on Less than or Equal Zero Likely
BGTZ	Branch on Greater Than Zero
BGTZL	Branch on Greater Than Zero Likely
BLTZ	Branch on Less Than Zero
BLTZL	Branch on Less Than Zero Likely
BGEZ	Branch on Greater Than or Equal Zero
BGEZL	Branch on Greater Than or Equal Zero Likely
BLTZAL	Branch on Less Than Zero And Link
BLTZALL	Branch on Less Than Zero And Link Likely
BGEZAL	Branch on Greater Than or Equal Zero And Link
BGEZALL	Branch on Greater Than or Equal Zero And Link Likely

Таблица 10. Команды Jump

Условное обозначение	Название
J	Jump
JAL	Jump and Link
JR	Jump Register
JALR	Jump and Link Register

Таблица 11. Специальные команды

Условное обозначение	Название
SYSCALL	System Call
BREAK	Breakpoint
ERET	Return from Exception
SYNC	Synchronize Shared Memory
TEQ	Trap if Equal
TEQI	Trap if Equal Immediate
TGE	Trap if Greater Then or Equal
TGEI	Trap if Greater Then or Equal Immediate
TGEIU	Trap if Greater Then or Equal Immediate Unsigned
TGEU	Trap if Greater Then or Equal Unsigned
TLT	Trap if Less Than
TLTI	Trap if Less Than Immediate
TLTIU	Trap if Less Than Immediate Unsigned
TLTU	Trap if Less Than Unsigned
TNE	Trap if Not Equal
TNEI	Trap if Not Equal Immediate

Таблица 12. Команды управления сопроцессором CP0

Условное обозначение	Название
MTC0	Move To CP0
MFC0	Move From CP0
MOVN	Move Conditional on Not Zero
MOVZ	Move Conditional on Zero
TLBR	Read indexed TLB entry
TLBWI	Write indexed TLB entry
TLBWR	Write Random TLB entry
TLBP	Probe TLB for matching entry

2.1 ADD – Сложение

31	25	20	15	10	5 0
SPECIAL 000000	rs	rt	rd	0 00000	ADD 100000

Формат: ADD rd, rs, rt

Описание: Содержимое регистров общего назначения *rs* и *rt* складывается. 32-битный результат помещается в регистр общего назначения *rd.*

Если сложение приводит к арифметическому переполнению в дополнительном коде, регистр-приемник при этом не изменяется.

Если целочисленное переполнение не происходит, то результат сложения записывается в общий регистр *rd.*

Ограничения: нет.

Операция:

```
T       temp      ← (GPR[rs]31||GPR[rs]31..0)      +
        (GPR[rt]31||GPR[rt]31..0)
        if temp32 ≠ temp31 then
        SignalException(IntegerOverflow)
        Else
        GPR[rd] ← temp
        Endif
```

Исключения: Integer Overflow

Примечание программисту: ADDU выполняет ту же самую операцию, но без перехвата переполнения.

2.2 ADDI - Сложение с непосредственным операндом

31	25	20	15 0
ADDI 001000	rs	rt	immediate

Формат: ADDI rt, rs, immediate

Описание: Расширенная знаком до 32 бит 16- битная величина *immediate* и значение регистра общего назначения *rs* складываются . 32-битный результат помещается в регистр общего назначения *rt.*

Если сложение приводит к арифметическому переполнению в дополнительном коде, регистр-приемник при этом не изменяется.

Если целочисленное переполнение не происходит, то результат сложения записывается в общий регистр *rd.*

Ограничения: нет.

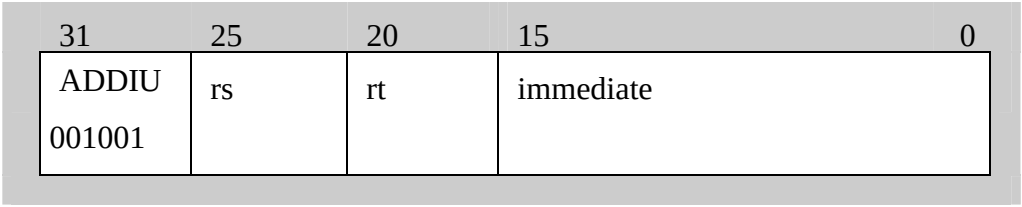
Операция:

```
T      temp       ← (GPR[rs]31||GPR[rs]31..0)       +
       sign_extend(immediate)
       if temp32 ≠ temp31 then
       SignalException(IntegerOverflow)
       Else
```

Исключения: Integer Overflow

Примечание программисту: ADDIU выполняет ту же самую операцию, но без перехвата переполнения.

2.3 ADDIU - Сложение с непосредственным операндом без знака



31	25	20	15 0
ADDIU 001001	rs	rt	immediate

Формат: ADDIU rt, rs, immediate

Описание: Расширенная знаком до 32 бит 16- битная величина *immediate* и значение регистра общего назначения *rs* складываются . 32-битный результат помещается в регистр общего назначения *rt*. Исключение «Целочисленное переполнение» не возникает ни при каких обстоятельствах.

Ограничения: нет.

Операция:

```
T       temp ← GPR[rs] + sign_extend(immediate)
        GPR[rt]← temp
```

Исключения: нет

Примечание программисту: Термин “без знака” в имени команды – не совсем верный. Эта операция относится к 32-разрядной арифметике по модулю, которая не отслеживает переполнение. Эта команда для арифметики без знака, типа адресной арифметики, или целочисленного арифметического окружения, которое игнорирует переполнение.

2.4 ADDU - Сложение без знака

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 00000	ADDU 100001	

Формат: ADDU rd, rs, rt

Описание: 32-битное слово из регистра общего назначения *rt* прибавляется к 32-битной величине в регистре общего назначения *rs*. Полученный арифметический результат помещается в регистр общего назначения *rd*. Целочисленное переполнение не происходит ни при каких условиях.

Ограничения: нет.

Операция:

```
T     temp ← GPR[rs] + GPR[rt]
      GPR[rd] ← temp
```

Исключения: нет

Примечание программисту: Термин “без знака” в имени команды – не совсем верный. Эта операция относится к 32-разрядной арифметике по модулю, которая не отслеживает переполнение. Эта команда для арифметики без знака, типа адресной арифметики, или целочисленного арифметического окружения, которое игнорирует переполнение.

2.5 AND - Празрядное логическое умножение

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 00000	AND 100100	

Формат: AND rd, rs, rt

Описание: 32-битный результат логического поразрядного умножения содержимого регистров общего назначения *rs* и *rt* помещается в регистр общего назначения *rd.*

Ограничения: нет.

Операция:

```
T: GPR[rd] ← GPR[rs] and GPR[rt]
```

Исключения: нет

2.6 ANDI - Поразрядное логическое умножение с непосредственным операндом

31	25	20	15 0
ANDI 001100	rs	rt	immediate

Формат: ANDI rt ,rs, immediate

Описание: 16-битная величина *immediate* расширяется нулями влево до 32 бит. Затем производится ее поразрядное логическое умножение с содержимым регистра общего назначения *rs*. Результат записывается в регистр общего назначения *rt.*

Ограничения: нет.

Операция:

T: GPR[rt] $\leftarrow (0)^{16}$ ||(immediate *and* GPR$[rs]_{15..0}$)

Исключения: нет

2.7 B – Безусловное ветвление

31	25	20	15 0
BEQ 000100	0	0	offset

Формат: B offset

Описание: Данное выражение является ассемблерной идиомой для обозначения безусловного ветвления. Фактически же команда интерпретируется аппаратным обеспечением как BEQ r0, r0, offset.

Для формирования PC-относительного адреса перехода 18-битное знаковое смещение (16-битная величина *offset*, смещенная влево на 2 бита) прибавляется к адресу команды, следующей за командой перехода (командой в слоте задержки).

Ограничения: Результат **НЕПРЕДСКАЗУЕМ**, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
I: target_offset ← sign_extend(offset || 0^2)
I+1: PC ← PC + target_offset
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC ± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

2.8 BAL – Ветвление с возвратом

31	25	20	15	0
REGIMM 000001	0	BGEZAL 10001	offset	

Формат: BAL rs, offset

Описание: Данное выражение является ассемблерной идиомой для обозначения безусловного ветвления. Фактически же команда интерпретируется аппаратным обеспечением как BGEZAL r0, offset.

Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2. В регистр связи r31 безусловно помещается адрес команды, находящейся после слота задержки.

Ограничение: Результат **НЕПРЕДСКАЗУЕМ**, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Общий регистр r31 не должен использоваться в качестве регистра-источника *rs*, потому что такая команда после перезапуска не приведет к такому же результату. Результат выполнения такой команды – **НЕПРЕДСКАЗУЕМ**. Это ограничение позволяет обработчику исключения возобновить выполнение программы путем перезапуска команды перехода, когда исключение возникает в слоте задержки.

Операция:

```
I: target_offset ← sign_extend(offset || 0^2)
GPR[31] ← PC + 8
I+1: PC ← PC + target_offset
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду JAL или JALR, если необходимо совершить переход на адрес за этим диапазоном.

2.9 BEQ - Ветвление по равенству

31	25	20	15	0
BEQ 000100	rs	rt	offset	

Формат: BEQ rs, rt, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2. Значения регистров общего назначения *rs* и *rt* сравниваются. Если они равны, осуществляется переход по адресу назначения с задержкой на 1 команду (после того, как выполнится команда из слота задержки).

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 02)
        condition ← (GPR[rs] = GPR[rt])
T+1:    if condition then
          PC ← PC + target_offset
        endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC ± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

2.10 BEQL - Ветвление по равенству по вероятности

31	25	20	15 0
BEQL 010100	rs	rt	offset

Формат: BEQL rs, rt, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2. Значения регистров общего назначения *rs* и *rt* сравниваются. Если они равны, осуществляется переход по адресу назначения с задержкой на 1 команду. Если условный переход не совершается, то команда в слоте задержки не выполняется.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 0^2)
        condition ← (GPR[rs] = GPR[rt])
T+1:    if condition then
        PC ← PC + target_offset
        else
        NullifyCurrentInstruction()
        endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC ± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

Рекомендуется не использовать её в ПО, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что ,если переход не происходит, мы имеем значительный штраф по

времени исполнения. Программное обеспечение должно использовать эту команду только, если существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BEQ.

2.11 BGEZ - Ветвление по соотношению «больше или равно нулю»

31	25	20	15 0
REGIMM 000001	rs	BGEZ 00001	offset

Формат: BGEZ rs, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2. Если содержимое регистра общего назначения *rs* больше или равно нулю (знаковый бит - нулевой), осуществляется переход по адресу назначения с задержкой на 1 команду.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:    target_offset ← sign_extend(offset || 0^2)
      condition ← GPR[rs] ≥ 0^GPRLEN
T+1:  if condition then
      PC ← PC + target_offset
      else
      NullifyCurrentInstruction()
      endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC ± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

2.12 BGEZAL - Ветвление по соотношению «больше или равно нулю» с возвратом

31	25	20	15	0
REGIMM 000001	rs	BGEZAL 10001	offset	

Формат: BGEZAL rs, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2. В регистр связи r31 безусловно помещается адрес команды, находящейчя после слота задержки. Если содержимое общего регистра *rs* больше или равно нулю (знаковый бит равен 0), совершается переход по адресу назначения с задержкой на 1 команду.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Общий регистр r31 не должен использоваться в качестве регистра-источника *rs*, потому что такая команда после перезапуска не приведет к такому же результату. Результат выполнения такой команды – НЕПРЕДСКАЗУЕМЫЙ. Это ограничение позволяет обработчику исключения возобновить выполнение программы путем перезапуска команды перехода, когда исключение возникает в слоте задержки.

Операция:

```
T:      target_offset ← sign_extend(offset || 0^2)
        condition ← GPR[rs] ≥ 0^GPRLEN
        GPR[31] ← PC + 8
T+1:    if condition then
        PC ← PC + target_offset
        else
        NullifyCurrentInstruction()
        endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду JAL, если необходимо совершить переход на адрес за этим диапазоном.

Выражение BAL является идиомой используемой, чтобы обозначить PC-относительный переход и связь, а JAL – абсолютный.

2.13 BGEZALL - Ветвление по соотношению «больше или равно нулю» по вероятности с возвратом

31	25	20	15	0
REGIMM 000001	rs	BGEZALL 10011	offset	

Формат: BGEZALL rs, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2. В регистр связи r31 бузуслвно помещается адрес команды, находящейся после слота задержки. Если содержимое общего регистра *rs* больше или равно нулю (знаковый бит равен 0), совершается переход по адресу назначения с задержкой на 1 команду. Если условный переход не совершается, то команда в слоте задержки не выполняется.

Ограничения: Общий регистр r31 не должен использоваться в качестве регистра-источника *rs*, потому что такая команда после перезапуска не приведет к такому же результату. Результат выполнения такой команды – НЕПРЕДСКАЗУЕМЫЙ. Это ограничение позволяет обработчику исключения возобновить выполнение программы путем перезапуска команды перехода, когда исключение возникает в слоте задержки.

Результат также НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 0^2)
        condition ← GPR[rs] ≥ 0^GPRLEN
        GPR[31] ← PC + 8
T+1:    if condition then
        PC ← PC + target_offset
        else
        NullifyCurrentInstruction()
        endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду JAL или JALR, если необходимо совершить переход на адрес вне этого диапазона.

Рекомендуется не использовать её в ПО, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что ,если переход не происходит, мы имеем значительный штраф по времени исполнения. Программное обеспечение должно использовать эту команду только, если существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BGEZAL.

2.14 BGEZL - Ветвление по соотношению «больше или равно нулю» по вероятности

31	25	20	15	0
REGIMM 000001	rs	BGELZ 00011	offset	

Формат: BGEZL rs, offset

Описание: `if rs ≥ 0 then branch_likely`

Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2. Если содержимое общего регистра *rs* больше или равно нулю (знаковый бит равен 0), совершается переход по адресу назначения с задержкой на 1 команду. Если условный переход не совершается, то команда в слоте задержки не выполняется.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 02)
        condition ← GPR[rs] ≥ 0^GPRLEN
T+1:    if condition then
        PC ← PC + target_offset
        else
        NullifyCurrentInstruction()
        endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

Рекомендуется не использовать её, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что ,если переход не происходит, мы имеем значительный штраф по времени исполнения. Программное обеспечение должно использовать эту команду только, если существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BGEZ.

2.15 BGTZ - Ветвление по соотношению «больше нуля»

31	25	20	15	0
BGTZ 000111	rs	0 00000	offset	

Формат: BGTZ rs, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2. Если содержимое общего регистра rs больше нуля (знаковый бит равен 0, а величина не равна 0), осуществляется переход по адресу назначения с задержкой на 1 команду.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 0^2)
        condition ← GPR[rs] > 0^GPRLEN
T+1:    if condition then
        PC ← PC + target_offset
        endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

2.16 BGTZL - Ветвление по соотношению «больше нуля» по вероятности

31	25	20	15	0
BGTZL 010111	rs	0 00000	offset	

Формат: BGTZL rs, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2 бита и расширенного знаком до 32 бит. Если содержимое общего регистра больше нуля (знаковый бит равен 0, а величина не равна 0), осуществляется переход по адресу назначения с задержкой на 1 команду. Если условный переход не совершается, то команда в слоте задержки не выполняется.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 0^2)
        condition ← GPR[rs] > 0GPRLEN
T+1:    if condition then
        PC ← PC + target_offset
        else
        NullifyCurrentInstruction()
        Endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

Рекомендуется не использовать её, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что, если переход не происходит, мы имеем значительный штраф по времени исполнения. Программное обеспечение должно использовать эту команду только, если

существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BGTZ.

2.17 BLEZ - Ветвление по соотношению «меньше или равно нулю»

31	25	20	15	0
BLEZ 000110	rs	0 00000	offset	

Формат: BLEZ rs,offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2 бита и расширенного знаком до 32 бит. Если содержимое общего регистра *rs* меньше или равно нулю (знаковый бит равен 1 или величина равна нулю), осуществляется переход по адресу назначения с задержкой на 1 команду.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 02)
        condition ← GPR[rs] ≤ 0^GPRLEN
T+1:    if condition then
        PC ← PC + target_offset
        Endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

2.18 BLEZL - Ветвление по соотношению «меньше или равно нулю» по вероятности

31	25	20	15	0
BLEZL 010110	rs	0 00000	Offset	

Формат: BLEZL rs, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2 бита и расширенного знаком до 32 бит. Если содержимое общего регистра *rs* меньше или равно нулю (знаковый бит равен 1 или величина равна нулю), осуществляется переход по адресу назначения с задержкой на 1 команду Если условный переход не совершается, команда в слоте задержки не выполняется.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 02)
        condition ← GPR[rs] ≤ 0^GPRLEN
T+1:    if condition then
        PC ← PC + target_offset
        else
        NullifyCurrentInstruction()
        endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона

Рекомендуется не использовать её в ПО, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что ,если переход не происходит, мы имеем значительный штраф по времени исполнения. Программное обеспечение должно использовать эту команду только, если

существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BLEZ.

2.19 BLTZ - Ветвление по соотношению «меньше нуля»

31	25	20	15	0
REGIMM 000001	rs	BLTZ 00000	Offset	

Формат: BLTZ rs,offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2 бита и расширенного знаком до 32 бит. Если содержимое общего регистра *rs* меньше нуля (знаковый бит равен 1), осуществляется переход по адресу назначения с задержкой на 1 команду.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 02)
        condition ← GPR[rs] < 0^GPRLEN
T+1:    if condition then
        PC ← PC + target_offset
        else
        NullifyCurrentInstruction()
        endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду JAL или JALR, если необходимо совершить переход на адрес вне этого диапазона.

2.20 BLTZAL - Ветвление по соотношению «меньше нуля» с возвратом

31	25	20	15	0
REGIMM 000001	rs	BLTZAL 10000	offset	

Формат: BLTZAL rs,offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2 бита и расширенного знаком до 32 бит. В регистр связи r31 безусловно помещается адрес команды, находящейчя после слота задержки. Если содержимое общего регистра *rs* меньше нуля (знаковый бит равен 1), осуществляется переход по адресу назначения с задержкой на 1 команду.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Общий регистр r31 не должен использоваться в качестве регистра-источника *rs*, потому что такая команда после перезапуска не приведет к такому же результату. Результат выполнения такой команды – НЕПРЕДСКАЗУЕМЫЙ. Это ограничение позволяет обработчику исключения возобновить выполнение программы путем перезапуска команды перехода, когда исключение возникает в слоте задержки.

Операция:

```
T:      Target_offset ← sign_extend(offset || 02)
        condition ← GPR[rs] < 0^GPRLEN
T+1:    If condition then
        PC ← PC + target_offset
        Endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду JAL или JALR, если необходимо совершить переход на адрес вне этого диапазона.

2.21 BLTZALL - Ветвление по соотношению «меньшще нуля» по вероятности с возвратом

31	25	20	15	0
REGIMM 000001	rs	BLTZALL 10010	offset	

Формат: BLTZALL rs, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2 бита и расширенного знаком до 32 бит. В регистр связи r31 бузуслвно помещается адрес команды, находящейся после слота задержки. Если содержимое общего регистра *rs* меньше нуля (знаковый бит равен 1), осуществляется переход по адресу назначения с задержкой на 1 команду. Если условный переход не совершается, то команда в слоте задержки не выполняется.

Ограничения: Общий регистр r31 не должен использоваться в качестве регистра-источника *rs*, потому что такая команда после перезапуска не приведет к такому же результату. Результат выполнения такой команды – НЕПРЕДСКАЗУЕМЫЙ. Это ограничение позволяет обработчику исключения возобновить выполнение программы путем перезапуска команды перехода, когда исключение возникает в слоте задержки.

Результат также НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      Target_offset ← sign_extend(offset || 02)
        Condition ← GPR[rs] < 0^GPRLEN
T+1:    If condition then
        PC ← PC + target_offset
        Else
        NullifyCurrentInstruction()
        Endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду JAL или JALR, если необходимо совершить переход на адрес вне этого диапазона.

Рекомендуется не использовать её, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что ,если переход не происходит, мы имеем значительный штраф по времени исполнения. Программное обеспечение должно использовать эту команду только, если существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BLTZAL.

2.22 BLTZL - Ветвление по соотношению «меньше нуля» по вероятности

31	25	20	15	0
REGIMM 000001	rs	BLTZL 00010	offset	

Формат: BLTZL rs, offset

Описание: `if rs < 0 then branch_likely`

Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2 бита и расширенного знаком до 32 бит. Если содержимое общего регистра *rs* меньше нуля (знаковый бит равен 1), осуществляется переход по адресу назначения с задержкой на 1 команду. Если условный переход не совершается, то команда в слоте задержки не выполняется.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 02)
        condition ← GPR[rs] < 0^GPRLEN
T+1:    if condition then
        PC ← PC + target_offset
        else
        NullifyCurrentInstruction()
        Endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

Рекомендуется не использовать её, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что ,если переход не происходит, мы имеем значительный штраф по времени исполнения. Программное обеспечение должно использовать эту команду только, если существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BLTZ.

2.23 BNE - Ветвление по соотношению «не равно»

31	25	20	15	0
BNE 000101	rs	rt	offset	

Формат: BNE rs, rt, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2 бита и расширенного знаком до 32 бит. Значения регистров общего назначения *rs* и *rt* сравниваются. Если они не равны, осуществляется переход по адресу назначения с задержкой на 1 команду.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 02)
        condition ← GPR[rs] ≠ 0^GPRLEN
T+1:    if condition then
        PC ← PC + target_offset
        else
        NullifyCurrentInstruction()
        endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

2.24 BNEL - Ветвление по соотношению «не равно» по вероятности

31	25	20	15	0
BNEL 010101	rs	rt	Offset	

Формат: BNEL rs, rt, offset

Описание: Адрес перехода вычисляется как сумма адреса команды в слоте задержки и 16-битного *offset*, сдвинутого влево на 2 бита и расширенного знаком до 32 бит. Значения регистров общего назначения *rs* и *rt* сравниваются. Если они не равны, осуществляется переход по адресу назначения с задержкой на 1 команду. Если условный переход не совершается, команда в слоте задержки не выполняется.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если команды ERET, DERET, WAIT или команды ветвления и перехода находятся в слоте задержки ветвления или перехода.

Операция:

```
T:      target_offset ← sign_extend(offset || 02)
        condition ← (GPR[rs] ≠ GPR[rt])
T+1:    if condition then
        PC ← PC + target_offset
        Else
        NullifyCurrentInstruction()
        Endif
```

Исключения: нет

Примечание программисту: Эта команда имеет ограниченное применение, так как она может адресовать знаковое смещение в диапазоне PC± 128 Кбайт. Используйте команду J или JR, если необходимо совершить переход на адрес вне этого диапазона.

Рекомендуется не использовать её, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что ,если переход не происходит, мы имеем значительный штраф по времени исполнения. Программное обеспечение должно использовать эту команду только, если

существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BNE.

2.25 BREAK – Точка останова

31	25	6	5	0
SPECIAL 000000	code		BREAK 001101	

Формат: BREAK

Описание: Поле code доступно для использования в качестве программных параметров, но извлекается обработчиком исключения только загрузкой содержимого слова в памяти, содержащего команду.

Ограничения: нет

Операция:

```
T:          SignalException(Breakpoint)
```

Исключения: Breakpoint

2.26 CLO - Подсчет количества ведущих единиц в слове

31	25	20	15	10	5	0
SPECIAL2 011100	rs	rt	rd	0 00000	CLO 100001	

Формат: CLO rd, rs

Описание: Выполняет подсчет количества ведущих единиц в слове. Биты 31-0 общего регистра *rs* сканируются от старшего к младшему. Количество ведущих единиц помещается в общий регистр *rd*. Если все биты 31-0 были установлены, то в регистр *rd* записывается значение 32.

Ограничения:

Программное обеспечение должно записать одинаковые значения GPR в rt и rd поля команды. Операция команды непредсказуема, если rt и rd поля команды содержат различные значения.

Операция:

```
T:      temp ← 32
        for i in 31 .. 0
          if GPR[rs]i = 0 then
        temp ← 31 - i
        break
          endif
        endfor
        GPR[rd] ← temp
```

Исключения: нет

2.27 CLZ - Подсчет количества ведущих нулей в слове

31	25	20	15	10	5	0
SPECIAL2 011100	rs	rt	rd	0 00000	CLZ 100000	

Формат: CLZ rd, rs

Описание: Выполняет подсчет количества ведущих нулей в слове. Биты 31-0 общего регистра *rs* сканируются от старшего к младшему. Количество ведущих нулей помещается в общий регистр *rd*. Если все биты 31-0 были сброшены, то в регистр *rd* записывается значение 32.

Ограничения: Программное обеспечение должно записать одинаковые значения GPR в rt и rd поля команды. Операция команды непредсказуема, если rt и rd поля команды содержат различные значения.

Операция:

```
T:      temp ← 32
        for i in 31 .. 0
          if GPR[rs]i = 1 then
        temp ← 31 - i
        break
          endif
        endfor
        GPR[rd] ← temp
```

Исключения: нет

2.28 DIV - Деление

31	25	20	15	5	0
SPECIAL 000000	rs	rt	0 00 0000 0000	DIV 011010	

Формат: DIV rs, rt

Описание: 32-битное слово из общего регистра *rs* делится на 32-битную величину из общего регистра *rt*, причем оба операнда интерпретируются как знаковые. 32-битное частное записывается в специальный регистр *LO*, а 32-битный остаток – в специальный регистр *HI*. Исключение по переполнению не происходит ни при каких обстоятельствах.

Ограничения: Если делитель rt равен нулю, то арифметическое значение результата НЕПРЕДСКАЗУЕМО.

Операция:

```
T-2:    q ← GPR[rs]31..0 div GPR[rt]31..0
T-1:    LO ← q
T:      r ← GPR[rs]31..0 mod GPR[rt]31..0
        HI ← r
```

Исключения: нет

Примечание программисту: Арифметические исключения не происходят ни при каких обстоятельствах. Если обнаружены условия переполнения или деления на ноль, и произведены некоторые действия, то за командой деления обычно используют дополнительные команды, выполняющие проверку на нулевой делитель и/или переполнение.

На некоторых процессорах операция целочисленного деления может выполняться асинхронно, другие команды процессора могут запускаться до завершения этой операции. Попытка получить значение из регистров HI или LO до того, как результаты деления будут записаны в них, блокируется, пока результаты не будут получены.

Асинхронное выполнение операции не влияет на результат, но предоставляет возможность оптимизировать производительность, планируя деление так, чтобы другие команды могли выполняться параллельно.

Если размер операндов известен, ПО должно размещать меньший операнд в регистре *rt*. Это позволит снизить время выполнение команды.

2.29 DIVU - Деление без знака

31	25	20	15	5	0
SPECIAL 000000	rs	rt	0 00 0000 0000	DIVU 011011	

Формат: DIVU rs, rt

Описание: 32-битное слово из общего регистра *rs* делится на 32-битную величину из общего регистра *rt*, причем оба операнда интерпретируются как беззнаковые. 32-битное частное записывается в специальный регистр *LO*, а 32-битный остаток – в специальный регистр *HI*. Исключение по переполнению не происходит ни при каких обстоятельствах.

Ограничения: Если делитель rt равен нулю, то арифметическое значение результата НЕПРЕДСКАЗУЕМО.

Операция:

```
T-2:    q ← (0 || GPR[rs]^31..0) div (0 || GPR[rt]^31..0)
T-1:    r ← (0 || GPR[rs]^31..0) mod (0 || GPR[rt]^31..0)
T:      LO ← sign_extend(q^31..0)
        HI ← sign_extend(r^31..0)
```

Исключения: нет

Примечание программисту: Арифметические исключения не происходят ни при каких обстоятельствах. Если обнаружены условия переполнения или деления на ноль, и произведены некоторые действия, то за командой деления обычно используют дополнительные команды, выполняющие проверку на нулевой делитель и/или переполнение.

На некоторых процессорах операция целочисленного деления может выполняться асинхронно, другие команды процессора могут запускаться до завершения этой операции. Попытка

получить значение из регистров HI или LO до того, как результаты деления будут записаны в них, блокируется, пока результаты не будут получены.

2.30 ERET – Возврат из исключения

31	25	24	5	0
COP0 010000	CO 1	0 000 0000 0000 0000 0000	ERET 011000	

Формат: ERET

Описание: Возврат из прерывания, исключения или перехвата ошибки.

Данная команда реализует программный барьер для любого изменения состояния сопроцессора 0, которое может нарушить выборку и декодирование команды по адресу PC, к которому возвращается ERET (например, изменение исполнительного ASID, режима User и режима адресации).

Ограничения: Команда не должна использоваться в слоте задержки комманд ветвления (branch) или перехода (jump), иначе результат неопределен.

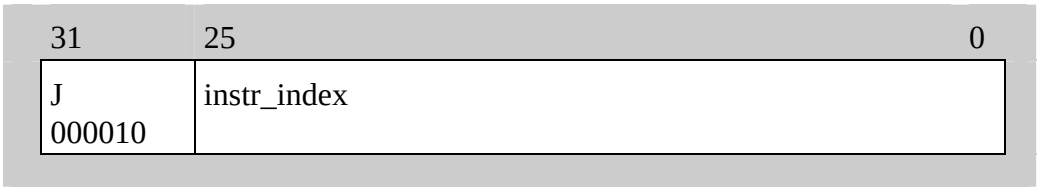
Включение команды ERET между командами LL и SC всегда приведет к сбою команды SC.

Операция:

```
T:    if StatusERL = 1 then
        temp ← ErrorEPC
        StatusERL ← 0
      else
        temp ← EPC
        StatusEXL ← 0
      endif
      if IsMIPS16Implemented() then
        PC ← temp31..1 || 0
        ISAMode ← temp0
      else
        PC ← temp
        endif
      LLbit ← 0
```

Исключения: Coprocessor Unusable

2.31 J - Переход



Формат: J target

Описание: Вычисляется адрес перехода: 26-битная *instr_index* сдвигается влево на 2 бита и дописывается до 32-х бит слева 4-мя старшими битами адреса команды в слоте задержки. Осуществляется безусловный переход по вычисленному адресу с задержкой на 1 команду (сначала выполняется команда в слоте задержки перехода).

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если в слоте задержки перехода расположены команды ERET, DERET, или WAIT.

Операция:

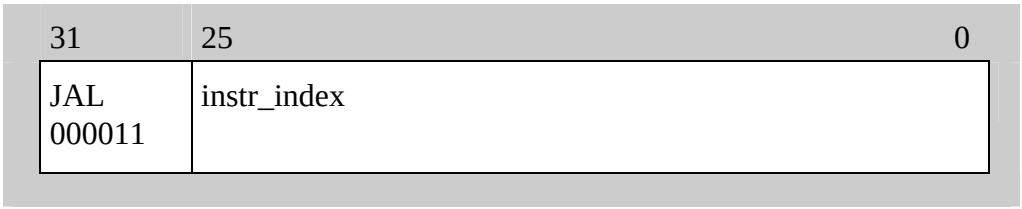
T:	temp ← instr_index
T+1:	PC ← $PC^{31..28}$ ||temp ||0^2

Исключения: нет

Примечание программисту: Формирования адреса перехода сложением PC и индексных битов вместо прибавления знакового смещения к PC выгоднее, если все адреса кода программы попадают в область 256 МБ, выравненную на границе 256 МБ. Это позволяет делать переход из любой области в любую область, что невозможно при относительном знаковом смещении.

Это определение создает следующий граничный случай: когда команда перехода находится в последнем слове из области 256 МБ, она может выполнять переход только к следующей 256 МБ области, содержащей слот задержки перехода.

2.32 JAL - Переход с возвратом



Формат: JAL target

Описание: Вычисляется адрес перехода: 26-битная *instr_index* сдвигается влево на 2 бита и дописывается до 32-х бит слева 4-мя старшими битами адреса команды в слоте задержки. Осуществляется безусловный переход по вычисленному адресу с задержкой на 1 команду. Адрес следующей за слотом задержки команды записывается в регистр связи r31.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если в слоте задержки перехода расположены команды ERET, DERET, или WAIT.

Операция:

T:	temp	←	instr_index
	GPR[31] ← PC+8		
T+1:	PC ← $PC_{31..28}$ ||temp ||0^2		

Исключения: нет

Примечание программисту: Формирования адреса перехода сложением PC и индексных битов вместо прибавления знакового смещения к PC выгоднее, если все адреса кода программы попадают в область 256 МБ, выравненную на границе 256 МБ. Это позволяет делать переход из любой области в любую область, что невозможно при относительным знаковом смещении.

Это определение создает следующий граничный случай: когда команда перехода находится в последнем слове из области 256 МБ, она может выполнять переход только к следующей 256 МБ области, содержащей слот задержки перехода.

2.33 JALR - Переход по регистру с возвратом

31	25	20	20	10	5	0
SPECIAL 000000	rs	0 00000	rd	0 00000	JALR 001001	

Формат: JALR rs или JALR rs, rd

Описание: Осуществляется безусловный переход с задержкой на 1 команду по содержащемуся в общем регистре *rs* адресу. Адрес следующей за слотом задержки команды помещается в общий регистр *rd*. По умолчанию, если не указано в ассемблерной команде, *rd* - это r31.

Регистры *rs*, *rd* должны быть разные, иначе при перезапуске команда будет выполняться по-другому. Однако, при попытке выполнения, такая команда не будет прервана, и результат ее выполнения неопределен.

Так как команды должны быть выравнены по границе слова, 2 младших бита содержимого *rs* должны быть нулевые. Иначе, при выборке команды назначения произойдет исключение по адресу.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если в слоте задержки перехода расположены команды ERET, DERET, или WAIT.

Номера регистров *rs* и *rd* не должны быть равными, потому что в этом случае команда после перпезапуска не даст тот же результат.

Операция:

```
T:      temp ← GPR[rs]
        GPR[rd] ← PC+8
T+1:    if Config1CA = 0 then
           PC ← temp
        else
           PC ← tempGPRLEN-1..1 || 0
           ISAMode ← temp0
        endif
```

Исключения: нет

Примечание программисту: Это единственная команда типа "переход и связывание", которая может выбирать регистр для запоминания адреса возврата; все другие команды связи используют GPR 31.

2.34 JR - Переход по регистру

31	25	20	5	0
SPECIAL 000000	rs	0 000 0000 0000 0000	JR 001000	

Формат: JR rs

Описание: Осуществляется безусловная передача управления с задержкой на 1 команду по содержащемуся в общем регистре *rs* адресу.

Так как команды должны быть выравнены по границе слова, 2 младших бита содержимого *rs* должны быть нулевые. Иначе, при выборке команды назначения произойдет исключение по адресу.

Ограничения: Результат НЕПРЕДСКАЗУЕМ, если в слоте задержки перехода расположены команды ERET, DERET, или WAIT.

Операция:

```
T:      temp ← GPR[rs]
T+1:    if Config1CA = 0 then
          PC ← temp
        else
          PC ← tempGPRLEN-1..1 || 0
          ISAMode ← temp0
        endif
```

Исключения: нет

Примечание программисту: Программы почти всегда используют регистр 31 для возврата из JAL, JALR или BGEZAL и поэтому регистр rs не должен использоваться в этой команде.

2.35 LB - Загрузка байта

31	25	20	15	0
LB 100000	base	rt	offset	

Формат: LB rt, offset(base)

Описание: Вычисляется исполнительный адрес: 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Байт памяти, определяемый этим адресом, выбирается, расширяется знаком до 32 бит и загружается в общий регистр *rt*.

Содержимое *rt* неопределено на время T в команде, сразу следующей за данной.

Ограничения: нет

Операция:

T:	vAddr ← $((offset_{15})^{16}$ || $offset_{15..0})$ + GPR[base] (pAddr,uncached) ← AddressTranslation(vAddr,DATA) mem ← LoadMemory(uncached,BYTE,pAddr,vAddr,DATA) byte ← $vAddr_{1..0}$ *xor* $BigEndianRISCORE^{2}$ GPR[rt] ← undefined
T+1:	GPR[rt] ← $(mem_{7+8*byte})^{24}$ ||$mem_{7+8*byte..8byte}$

Исключения: TLB Refill, TLB Invalid, Address Error

2.36 LBU - Загрузка байта без знака

31	25	20	15 0
LBU 100100	base	rt	offset

Формат: LBU rt, offset(base)

Описание: Вычисляется исполнительный адрес: 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Байт памяти, определяемый этим адресом, выбирается, расширяется нулями слева до 32 бит и загружается в общий регистр *rt*.

Содержимое *rt* неопределено на время T в команде, сразу следующей за данной.

Ограничения: нет

Операция:

```
T:    vAddr ← ((offset15)^16 || offset15..0) + GPR[base]
      (pAddr,uncached) ← AddressTranslation(vAddr,DATA)
      mem ← LoadMemory(uncached,BYTE,pAddr,vAddr,DATA)
      byte ← vAddr1..0 xor BigEndianRISCORE^2
      GPR[rt] ← undefined
T+1:  GPR[rt] ← 0^24 ||mem7+8*byte..8*byte
```

Исключения: TLB Refill, TLB Invalid, Address Error

2.37 LH - Загрузка полуслова

31	25	20	15	0
LH 100001	base	rt	offset	

Формат: LH rt, offset(base)

Описание: Вычисляется исполнительный адрес: 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Полуслово памяти, определяемое этим адресом, выбирается, расширяется знаком до 32 бит и загружается в общий регистр *rt*.

Содержимое *rt* неопределено на время T в команде, сразу следующей за данной.

Ограничения: Исполнительный адрес должен быть естественно выровненным. Ненулевой младший бит адреса вызывает исключение «Ошибка адреса».

Операция:

```
T:      vAddr ← ((offset15)^16 || offset15..0) + GPR[base]
        (pAddr,uncached) ← AddressTranslation(vAddr,DATA)
        mem ← LoadMemory(uncached,HALFWORD,pAddr,vAddr,DATA)
        byte ← vAddr1..0 xor (BigEndianRISCORE || 0)
        GPR[rt] ← undefined
T+1:    GPR[rt] ← (mem15+8*byte)^16 ||mem15+8*byte..8*byte
```

Исключения: TLB Refill, TLB Invalid, Bus Error, Address Error

2.38 LHU - Загрузка полуслова без знака

31	25	20	15 0
LHU 100101	base	rt	offset

Формат: LHU rt, offset(base)

Описание: Вычисляется исполнительный адрес: 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Полуслово памяти, определяемое этим адресом, выбирается, расширяется нулями слева до 32 бит и загружается в общий регистр *rt*. Содержимое *rt* неопределено на время T в команде, сразу следующей за данной.

Ограничения: Исполнительный адрес должен быть естественно выровненным. Ненулевой младший бит адреса вызывает исключение «Ошибка адреса».

Операция:

```
T:    vAddr ← ((offset15)^16 || offset15..0) + GPR[base]
      (pAddr,uncached) ← AddressTranslation(vAddr,DATA)
      mem ← LoadMemory(uncached,HALFWORD,pAddr,vAddr,DATA)
      byte ← vAddr1..0 xor (BigEndianRISCORE || 0)
      GPR[rt] ← undefined
T+1:  GPR[rt] ← 0^16 ||mem15+8*byte..8*byte
```

Исключения: TLB Refill, TLB Invalid, Address Error

2.39 LL - Загрузка связанного слова

31	25	20	15	0
LL 110000	base	rt	offset	

Формат: LL rt, offset(base)

Описание: Команды LL и SC обеспечивают атомарные операции «чтение-модификация-запись» (RMW) для кэшируемых областей памяти.

Вычисляется исполнительный адрес: 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Слово памяти, определяемое этим адресом, выбирается, расширяется знаком до 32 бит и загружается в общий регистр *rt*.

Это запускает RMW последовательность на текущем процессоре. Может быть только одна активная RMW последовательность на процессор.

Завершение клманды LL запускает активную RMW последовательность, замещая любую другую последовательность, которая была активна.

RMW последовательность завершается последующей SC командой, которая либо успешно завершает RMW последовательность, либо не завершает ее и происходит сбой.

Выполнение LL на одном процессоре не приводит к такому действию, которое, само по себе, привело бы к сбою при выполнении команды SC для этого же блока на другом процессоре.

За выполнением LL не обязательно должно следовать выполнение SC; программа может отказаться от RMW последовательности без попытки записи.

Содержимое *rt* неопределено на время T в команде, сразу следующей за данной.

Ограничения: Адресованное расположение должно быть кешированным, иначе результат НЕОПРЕДЕЛЕН. Исполнительный адрес должен быть естественно-выровненным. Если любой из 2 младших битов исполнительного адреса ненулевой, происходит исключение «Ошибка адреса».

Операция:

```
T:      vAddr ← sign_extend(offset) + GPR[base]
        if vAddr^1..0 ≠ 0^2 then
          SignalException(AddressError)
        endif
        (pAddr, CCA) ← AddressTranslation (vAddr,
        DATA, LOAD)
        memword ← LoadMemory (CCA, WORD, pAddr,
        vAddr, DATA)
        GPR[rt] ← memword
        LLbit ← 1
T+1:    GPR[rt] ← mem
```

Исключения: TLB Refill, TLB Invalid, Address Error, Reserved Instruction

Примечание программисту: Не существует инструкции Load Linked Word Unsigned, соответствующей инструкции Load Word Unsigned.

2.40 LUI - Загрузка непосредственного операнда

31	25	20	15	0
LUI 001111	00000	rt	immediate	

Формат: LUI rt, immediate

Описание: 16-битный непосредственный операнд *immediate*, расширенный до 32 бит справа нулями, загружается в общий регистр *rt*.

Ограничения: нет

Операция:

T: GPR[rt] ← immediate $||0^{16}$

Исключения: нет

2.41 LW - Загрузка слова

31	25	20	15 0
LW 100011	base	rt	offset

Формат: LW rt, offset(base)

Описание: Для вычисления исполнительного адреса, 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Содержимое слова памяти, определяемое исполнительным адресом, загружается в общий регистр *rt.*

Содержимое *rt* неопределено на время T в следующей команде.

Ограничения: Исполнительный адрес должен быть естественно выровненным. Ненулевой младший бит адреса вызывает исключение «Ошибка адреса».

Операция:

```
T:    vAddr ← ((offset15)^16 || offset15..0) + GPR[base]
      (pAddr,uncached) ← AddressTranslation(vAddr,DATA)
      mem ← LoadMemory(uncached,WORD,pAddr,vAddr,DATA)
      GPR[rt] ← undefined
T+1:  GPR[rt] ← mem
```

Исключения: TLB Refill, TLB Invalid, Bus Error, Address Error

2.42 LWL - Загрузка слова влево

31	25	20	15	0
LWL 100010	base	rt	offset	

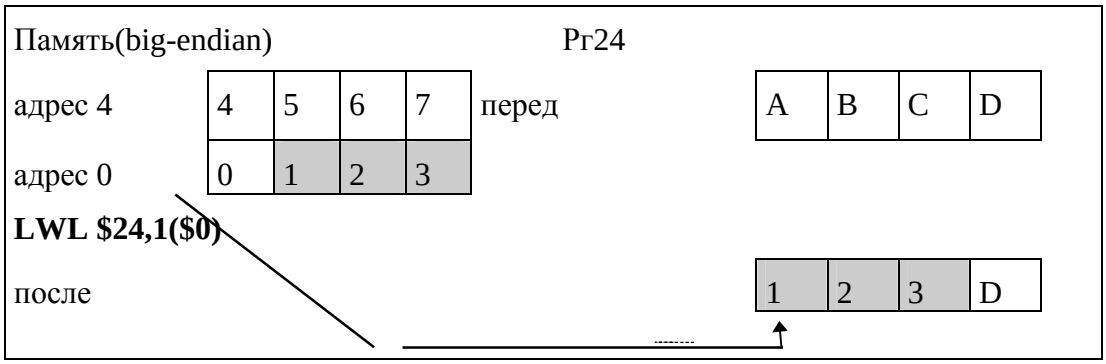
Формат: LWL rt, offset(base)

Описание: Эта команда может использоваться в комбинации с LWR для загрузки в регистр выборки 4-х последовательных байт , когда 2 байта пересекают границу двух слов. LWL заполняет левую часть регистра *rt* более значащей частью слова, LWR заполняет *rt* справа менее значащей частью слова.

Исполнительный адрес формируется сложением 16-битной *offset,* расширенной знаком до 32 бит, и содержимого общего регистра *base* и может характеризовать произвольный байт. Команда считывает байты только из слова памяти, которое содержит определенный стартовый байт. В зависимости от стартового байта будет загружено от 1 до 4 байт.

Механизм работы данной команды можно представить следующим образом.

Команда начинается с загрузки стартового байта в старший (самый левый) байт регистра. Затем выбирается следующий, менее значащащий байт памяти и загружается в менее значащий байт регистра. Этот процесс продолжается до тех пор, пока не будет достигнут самый младший байт слова памяти. Самый(е) младший(е) байт(ы) (самый(е) правый(е)) регистра не меняются.



Содержимое общего регистра *rt* игнорируется внутри процессором, так что не требуется использовать команду NOP между предшествующей командой загрузки, которая использует регистр *rt*, и последующей инстукцией LWL (LWR), также использующих этот регистр.

Исключения вследствии невыровненности адреса невозможны.

Содержимое *rt* неопределено на время T в следующей команде.

Ограничения: нет

Операция:

T:	vAddr ← $((offset_{15})^{16}$ || $offset_{15..0})$ + GPR[base]
	(pAddr,uncached) ← AddressTranslation(vAddr,DATA)
	byte ← $vAddr_{1..0}$ *xor* $BigEndianRISCORE^{2}$
	if BigEndianMem = 0 then
	pAddr ← $pAddr_{31..2}$ ||0^{2}
	endif
	mem ← LoadMemory(uncached,byte,pAddr,vAddr,DATA)

T+1:	GPR[rt] ← $mem_{7+8*byte..0}$ ||$GPR[rt]_{23-8*byte..0}$

Исключения: TLB Refill, TLB Invalid, Bus Error, Address Error

2.43 LWR - Загрузка слова вправо

31	25	20	15	0
LWR 100110	base	rt	offset	

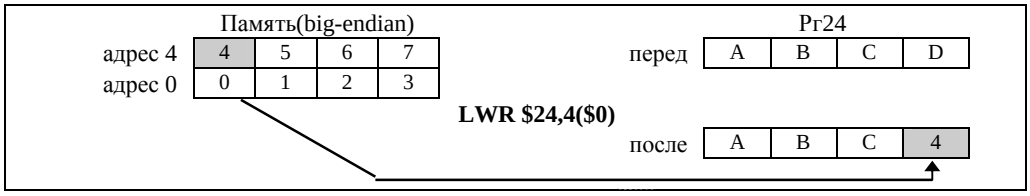
Формат: LWR rt, offset(base)

Описание: Эта команда может использоваться в комбинации с LWL для загрузки в регистр выборки 4-х последовательных байт , когда 2 байта пересекают границу двух слов. LWR заполняет правую часть регистра *rt* менее значащей частью слова, LWД заполняет *rt* слева более значащей частью слова.

Виртуальный адрес формируется сложением 16-битной *offset*, расширенной знаком до 32 бит, и содержимого общего регистра *base* и может характеризовать произвольный байт. Команда считывает байты только из слова памяти, которое содержит определенный стартовый байт. В зависимости от стартового байта будет загружено от 1 до 4 байт.

Механизм работы данной команды можно представить следующим образом.

Команда начинается с загрузки стартового байта в младший (самый правый) байт регистра. Затем выбирается следующий, более значащащий байт памяти и загружается в более значащий байт регистра. Этот процесс продолжается до тех пор, пока не будет достигнут самый старший байт слова памяти. Самый(е) старший(е) байт(ы) (самый(е) левый(е)) регистра не меняются.



Содержимое общего регистра *rt* игнорируется внутри процессором, так что не требуется использовать команду NOP между предшествующей командой загрузки, которая использует регистр *rt*, и последующей инстукцией LWL (LWR), также использующих этот регистр.

Исключения вследствии невыровненности адреса невозможны.

Содержимое *rt* неопределено на время Т в следующей команде.

Ограничения: нет

Операция:

T:	vAddr ← $((offset_{15})^{16}$ || $offset_{15..0})$ + GPR[base]
	(pAddr,uncached) ← AddressTranslation(vAddr,DATA)
	byte ← $vAddr_{1..0}$ *xor* $BigEndianRISCORE^{2}$
	if BigEndianMem = 1 then
	pAddr ← $pAddr_{31..2}$ ||0^{2}
	endif
	mem ← LoadMemory(uncached,WORD-byte,pAddr,vAddr,DATA)

T+1:	GPR[rt] ← $GPR[rt]_{31..32-8*byte}$ ||$mem_{31..8*byte}$

Исключения: TLB Refill, TLB Invalid, Bus Error, Address Error

2.44 MADD - Умножение и добавление слова к регистрам HI, LO

31	25	20	15	10	5	0
SPECIAL2 011100	rs	rt	0 00000	0 0000	MADD 000000	

Формат: MADD rs, rt

Описание: 32-битное значение общего регистра *rt* умножается на 32-битное значение регистра *rs*. Оба операнда интерпретируются как знаковые. Результат умножения – 64-битная величина. Этот результат добавляется к 64-битной величине, полученной объединением регистров HI и LO. Старшие 32 бита полученного значения записываются в регистр HI, младшие 32 бита полученного значения записываются в регистр LO. Арифметические исключения не происходят ни при каких обстоятельствах.

```
(LO,HI) ← (rs x rt) + (LO,HI)
```

Ограничения: Инструкция MADD выполняется за несколько тактов. Во время её выполнения могут выполняться другие инструкции процессора, не относящиеся к умножению. В случае попытки чтения регистров результата до окончания умножения происходит блокировка процессора до тех пор, пока результат умножения не будет получен и предоставлен для чтения.

Операция:

```
T:    temp ← (HI || LO) + (GPR[rs] * GPR[rt])
      HI ← temp63..32
      LO ← temp31..0
```

Исключения: нет

Примечание программистам: Если известен размер операндов, то желательно меньший из них расположить в общий регистр rt. Это может немного уменьшить время выполнения операции.

2.45 MADDU - Умножение и добавление слова без знака к регистрам HI, LO

31	25	20	15	10	5	0
SPECIAL 011100	rs	rt	0 00000	0 00000	MADDU 000001	

Формат: MADDU rs, rt

Описание: 32-битное значение общего регистра *rt* умножается на 32-битное значение регистра *rs*. Оба операнда интерпретируются как беззнаковые. Результат умножения – 64-битная величина. Этот результат добавляется к 64-битной величине, полученной объединением регистров HI и LO. Старшие 32 бита полученного значения записываются в регистр HI, младшие 32 бита полученного значения записываются в регистр LO. Арифметические исключения не происходят ни при каких обстоятельствах.

```
(LO,HI) ← (rs x rt) + (LO,HI)
```

Ограничения: Инструкция MADDU выполняется за несколько тактов. Во время её выполнения могут выполняться другие инструкции процессора, не относящиеся к умножению. В случае попытки чтения регистров результата до окончания умножения происходит блокировка процессора до тех пор, пока результат умножения не будет получен и предоставлен для чтения.

Операция:

```
T:    temp ← (HI || LO) + (GPR[rs] * GPR[rt])
      HI ← temp63..32
      LO ← temp31..0
```

Исключения: нет

Примечание программистам: Если известен размер операндов, то желательно меньший из них расположить в общий rt. Это может немного уменьшить время выполнения операции.

2.46 MFC0 - Пересылка из сопроцессора 0

31	25	20	15	10 0
COP0 010000	MF 00000	rt	rd	0 000 0000 0000

Формат: MFC0 rt, rd

Описание: Содержимое регистра *rd* сопроцессора 0 загружается в общий регистр *rt.*

Ограничения: Результат НЕОПРЕДЕЛЕН, если в сопроцессоре нет регистра с номером *rd.*

Операция:

```
T:      data ← CPR[0,rd]
T+1:    GPR[rt] ← data
```

Исключения: Coprocessor Unusable, Reserved Instruction

2.47 MFHI - Пересылка из регистра HI

31	25	15	10	5 0
SPECIAL 000000	0 00 0000 0000	rd	0 00000	MFHI 010000

Формат: MFHI rd

Описание: Содержимое специального регистра *HI* загружается в общий регистр *rd.*

Ограничения: нет

Операция:

```
T:          GPR[rd] ← HI
```

Исключения: нет

2.48 MFLO - Пересылка из регистра LO

31	25	15	10	5	0
SPECIAL 000000	0 00 0000 0000	rd	0 00000	MFLO 010010	

Формат: MFLO rd

Описание: Значение регистра *LO* загружается в общий регистр *rd.*

Ограничения: нет

Операция:

```
T:          GPR[rd] ← LO
```

Исключения: нет

2.49 MOVN - Условная пересылка при неравенстве нулю

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 00000	MOVN 001011	

Формат: MOVN rs, rt, rd

Описание: `if rt ≠ 0 then rd ← rs`

Пересылает значение общего регистра rs в общий регистр rd в случае, если значение общего регистра rt не равно нулю.

Ограничения: нет

Операция:

T:	if GPR[rt] □ 0□then GPR[rd] □ GPR[rs] endif

Исключения: нет

Примечание программистам: Ненулевая величина, которая здесь проверяется, - это результат «ИСТИНА» из команд сравнения SLT, SLTI, SLTU, SLTIU.

2.50 MOVZ - Условная пересылка при равенстве нулю

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 00000	MOVZ 001010	

Формат: MOVZ rs, rt, rd

Описание: `if rt = 0 then rd ← rs`

Пересылает значение общего регистра rs в общий регистр rd, в случае, если значение общего регистра rt равно нулю.

Ограничения: нет

Операция:

T:	`if GPR[rt] = 0 then`
	`GPR[rd] ← GPR[rs]`
	`endif`

Исключения: нет

Примечание программистам: : Нулевая величина, которая здесь проверяется, - это результат «ЛОЖЬ» из команд сравнения SLT, SLTI, SLTU, SLTIU.

2.51 MSUB - Умножение и вычитание слова из регистров HI, LO

31	25	20	15		5	0
SPECIAL 011100	rs	rt	0 00000	0 00000	MSUB 000100	

Формат: MSUB rs, rt

Описание: 32-битное значение общего регистра *rt* умножается на 32-битное значение регистра *rs*. Оба операнда интерпретируются как знаковые. Результат умножения – 64-битная величина. Этот результат вычитается из 64-битной величины, полученной объединением регистров HI и LO. Старшие 32 бита полученного значения записываются в регистр HI, младшие 32 бита полученного значения записываются в регистр LO. Арифметические исключения не происходят ни при каких обстоятельствах.

```
(LO,HI) ← (rs x rt) - (LO,HI)
```

Ограничения: Инструкция MSUB выполняется за несколько тактов. Во время её выполнения могут выполняться другие инструкции процессора, не относящиеся к умножению. В случае попытки чтения регистров результата до окончания умножения происходит блокировка процессора до тех пор, пока результат умножения не будет получен и предоставлен для чтения.

Операция:

```
T:    temp ← (HI || LO) - (GPR[rs] * GPR[rt])
      HI ← temp63..32
      LO ← temp31..0
```

Исключения: ует

Примечание программистам: Если известен размер операндов, то желательно меньший из них расположить в общий регистр rt. Это может немного уменьшить время выполнения операции.

2.52 MSUBU - Умножение и вычитание слова без знака из регистров HI, LO

31	25	20	15	10	5 0
SPECIAL2 011100	rs	rt	0 00000	0 00000	MSUBU 000101

Формат: MSUBU rs, rt

Описание: 32-битное значение общего регистра *rt* умножается на 32-битное значение регистра *rs*. Оба операнда интерпретируются как, беззнаковые. Результат умножения – 64-битная величина. Этот результат вычитается из 64-битной величины, полученной объединением регистров HI и LO. Старшие 32 бита полученного значения записываются в регистр HI, младшие 32 бита полученного значения записываются в регистр LO. Арифметические исключения не происходят ни при каких обстоятельствах.

```
(LO,HI) ← (rs x rt) - (LO,HI)
```

Ограничения: Инструкция MSUBU выполняется за несколько тактов. Во время её выполнения могут выполняться другие инструкции процессора, не относящиеся к умножению. В случае попытки чтения регистров результата до окончания умножения происходит блокировка процессора до тех пор, пока результат умножения не будет получен и предоставлен для чтения.

Операция:

```
T:      temp ← (HI || LO) - (GPR[rs] * GPR[rt])
        HI ← temp63..32
        LO ← temp31..0
```

Исключения: нет

Примечание программистам: Если известен размер операндов, то желательно меньший из них расположить в общий регистр rt. Это может немного уменьшить время выполнения операции.

2.53 MTC0 - Пересылка в сопроцессор 0

31	25	20	15	10 0
COP0 010000	MT 00100	rt	rd	0 000 0000 0000

Формат: MTC0 rt, rd

Описание: Значение общего регистра *rt* загружается в регистр *rd* сопроцессора 0.

Ограничения: Результат НЕОПРЕДЕЛЕН, если в сопроцессоре нет регистра с номером *rd.*

Операция:

```
T:      data          ←          GPR[rt]
T+1:    CPR[0,rd] ← data
```

Исключения: Coprocessor Unusabke, Reserved Instruction

2.54 MTHI - Пересылка в регистр HI

31	25	20	5	0
SPECIAL 000000	rs	0 000 0000 0000 0000	MTHI 010001	

Формат: MTHI rs

Описание. Значение общего регистра *rs* загружается в специальный регистр HI.

Ограничения: Если данная команда выполняется после арифметических команд DIV, DIVU, MULT, MULTU, но перед командами MFLO или MFHI, содержимое регистра LO непредсказуемо.

Операция:

```
T-2:    HI          ←    неопределен
T-1:    HI          ←    неопределен
T:      HI ← GPR[rs]
```

Исключения: нет

2.55 MTLO - Пересылка в регистр LO

31	25	20	5	0
SPECIAL 000000	rs	0 000 0000 0000 0000	MTLO 010011	

Формат: MTLO rs

Описание: Значение общего регистра *rs* загружается в специальный регистр *LO*

Ограничения: Если данная команда выполняется после арифметических команд DIV, DIVU, MULT, MULTU, но перед командами MFLO или MFHI, содержимое регистра HI непредсказуемо.

Операция:

```
T-2:    LO          ←    неопределен
T-1:    LO          ←    неопределен
T:      LO ← GPR[rs]
```

Исключения: нет

2.56 MUL - Умножение слова и запись в регистр общего назначения

31	25	20	15	10	5	0
SPECIAL2 011100	rs	rt	rd	0 00000	MUL 100000	

Формат: MUL rd, rs, rt

Описание: `rd ← rs × rt.`

32-битное слово из общего регистра *rs* умножается на 32-битное слово из общего регистра *rt*. Оба значения интерпретируются как знаковые величины. Произведение является 64-битной величиной. Младшие 32 бита результата записываются в общий регистр *rd*. Регистры HI и LO НЕПРЕДСКАЗУЕМЫ после этой команды. Арирфметические исключения не происходят ни при каких обстоятельствах.

Ограничения: Данная команда не предоставляет возможности для записи результата в регистры HI и LO.

Операция:

```
T:      temp <- GPR[rs] * GPR[rt]
        GPR[rd] <- temp31..0
        HI <- UNPREDICTABLE
        LO <- UNPREDICTABLE
```

Исключения: нет

Примечание программисту: На некоторых процессорах операция целочисленного умножения может выполняться асинхронно, другие команды процессора могут запускаться до завершения этой операции. Попытка получить значение из регистров HI или LO до того, как результаты деления будут записаны в них, блокируется, пока результаты не будут получены.

Асинхронное выполнение операции не влияет на результат, но предоставляет возможность оптимизировать производительность, планируя умножение так, чтобы другие команды могли выполняться параллельно.

Программы, требующие распознования переполнения, должны производить проверку явным образом.

Если размер операндов известен, ПО должно размещать меньший операнд в регистре *rt*. Это позволит снизить время выполнение команды.

2.57 MULT – Умножение слова

31	25	20	15	5 0
SPECIAL 000000	rs	rt	0 00 0000 0000	MULT 011000

Формат: MULT rs, rt

Описание: 32-битное слово из общего регистра *rs* умножается на 32-битное слово из общего регистра *rt*, причем оба операнда интерпретируются как знаковые. Младшие 32 бита произведения записывается в специальный регистр *LO*, а старшие 32 бита – в специальный регистр *HI*. Арифметических исключений происходит ни при каких обстоятельствах.

Ограничения: нет

Операция:

```
T-2:    LO,HI          ←          неопределен
T-1:    LO,HI          ←          неопределен
T:      t       ←      GPR[rs]    *    GPR[rt]
        LO             ←               t31..0
        HI ← t63..32
```

Исключения: нет

Примечание программисту: На некоторых процессорах операция целочисленного умножения может выполняться асинхронно, другие команды процессора могут запускаться до завершения этой операции. Попытка получить значение из регистров HI или LO до того, как результаты деления будут записаны в них, блокируется, пока результаты не будут получены.

Асинхронное выполнение операции не влияет на результат, но предоставляет возможность оптимизировать производительность, планируя умножение так, чтобы другие команды могли выполняться параллельно.

Программы, требующие распознования переполнения, должны производить проверку явным образом.

Если размер операндов известен, ПО должно размещать меньший операнд в регистре *rt*. Это позволит снизить время выполнение команды.

2.58 MULTU - Умножение слова без знака

31	25	20	15	5 0
SPECIAL 000000	rs	rt	0 00 0000 0000	MULTU 011001

Формат: MULTU rs, rt

Описание: 32-битное слово из общего регистра *rs* умножается на 32-битное слово из общего регистра *rt*, причем оба операнда интерпретируются как беззнаковые. Младшие 32 бита произведения записывается в специальный регистр *LO*, а старшие 32 бита – в специальный регистр *HI*. Арифметических исключений происходит ни при каких обстоятельствах.

Ограничения: нет

Операция:

```
T-2:    LO,HI                   ←   неопределен
T-1:    LO,HI                   ←   неопределен
T:      t   ←   (0 || GPR[rs])*(0 || GPR[rt])
        LO                      ←   t31..0
        HI ← t63..32
```

Исключения: нет

Примечание программисту: На некоторых процессорах операция целочисленного умножения может выполняться асинхронно, другие команды процессора могут запускаться до завершения этой операции. Попытка получить значение из регистров HI или LO до того, как результаты деления будут записаны в них, блокируется, пока результаты не будут получены.

Асинхронное выполнение операции не влияет на результат, но предоставляет возможность оптимизировать производительность, планируя умножение так, чтобы другие команды могли выполняться параллельно.

Программы, требующие распознования переполнения, должны производить проверку явным образом.

Если размер операндов известен, ПО должно размещать меньший операнд в регистре *rt*. Это позволит снизить время выполнение команды.

2.59 NOP – Нет операции

31	25	20	15	10	5	0
SPECIAL 000000	00000	0	0	0	SLL 000000	

Формат: NOP

Описание: Данное выражение является ассемблерной идиомой для обозначения команды "нет операции". Фактически же команда интерпретируется аппаратным обеспечением как SLL r0, r0, 0.

Ограничения: нет

Операция: нет

Исключения: нет

2.60 NOR - Отрицание логического сложения

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 00000	NOR 100111	

Формат: NOR rd, rs, rt

Описание: Результат отрицания логического поразрядного сложения (НЕ ИЛИ) значений общих регистров *rs* и *rt* помещается в общий регистр *rd.*

Ограничения: нет

Операция:

```
T: GPR[rd] ← GPR[rs] nor GPR[rt]
```

Исключения: нет

2.61 OR - Логическое сложение

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 00000	OR 100101	

Формат: OR rd, rs, rt

Описание. Результат логического поразрядного сложения значений общих регистров *rs* и *rt* помещается в общий регистр *rd.*

Ограничения: нет

Операция:

```
T: GPR[rd] ← GPR[rs] or GPR[rt]
```

Исключения: нет

2.62 ORI - Логическое сложение с непосредственным операндом

31	25	20	15 0
ORI 001101	rs	rt	immediate

Формат: ORI rt, rs, immediate

Описание: Результат логического поразрядного сложения *immediate*, расширенного нулями влево до 32 бит, и значения общего регистра *rs* помещается в общий регистр *rt.*

Ограничения: нет

Операция:

T: GPR[rt] ← $GPR[rs]_{31..16}$ ||(immediate *or* $GPR[rs]_{15..0}$)

Исключения: Нет

2.63 SB - Сохранение байта

31	25	20	15	0
SB 101000	base	rt	offset	

Формат: SB rt, offset(base)

Описание: Формируется исполнительный адрес: 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base.* Младший байт из общего регистра *rt* записывается в память по полученному адресу.

Ограничения: нет

Операция:

```
T:      vAddr ← ((offset15)^16 || offset15..0) + GPR[base]
        (pAddr,uncached) ← AddressTranslation(vAddr,DATA)
        pAddr ← pAddrPSIZE-1..2 || (pAddr1..0 xor ReservEndian^2)
        byte ← vAddr1..0 xor BigEndianRISCORE^2
        data ← GPR[rt] 31-8*byte..0 || 0^8*byte ?
        StoreMemory(uncached,BYTE,data,pAddr,vAddr,DATA)
```

Исключения: TLB Refill, TLB Invalid, TLB Modified, Bus Error, Address Error

2.64 SC - Сохранение условного слова

31	25	20	15	0
SC 111000	base	rt	offset	

Формат: SC rt, offset(base)

Описание.

Команды LL и SC предоставляют элементарные операции для RMW последовательностей для кешированных участков памяти.

16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base* для формирования исполнительного адреса.

SC завершает RMW последовательность, начатую предшествующей командой LL, выполненной на данном процессоре, Для завершения RMW последовательности выполняется следующее:

1) младшее 32-битное слово из регистра *rt* записывается в ячейку памяти, характеризуемую выровненным исполнительным адресом;

2) единица записывается в общий регистр *rt*, что сигнализирует об успешно выполненной операции. В противном случае в память ничего не записывается, а в регистр записывается нуль, что сигнализирует об ошибке.

Если хотя бы одно из следующих событий происходит между выполнением LL и SC, команда SC дает сбой:

1) произведена когерентная запись другим процессором или выполнен модуль когерентного ввода/вывода в блок физической памяти, содержащей требуемое слово. Размер и выравнивание такого блока зависят от реализации процессора, но как минимум – это 1 слово, а как максимум – это минимальный размер страницы;

2) при выполнении команд LL/SC происходит исключение.

Если хотя бы одно из следующих событий происходит между выполнением LL и SC, команда SC может либо завершиться успешно, либо дать сбой (успех или неудача непредсказуемы). Переносимые программы не должны приводить к следующим событиям:

1) процессором, исполняющим команды LL/SC, выполняется загрузка, сохранение или упреждающая выборка из памяти;

2) выполняемые команды, начиная с LL и заканчивая SC, не расположены в пределах 2048-байтной непрерывной области виртуальной памяти. Область памяти не обязательно должна быть выровнена, в отличие от слов команд.

Для того чтобы результат выполнения команды SC был определен, следующие условия должны быть «ИСТИНОЙ»:

1) перед выполнением команды SC должна быть выполнена команда LL;

2) RMW последовательность, выполняемая без промежуточных исключений, должна использовать одинаковые адреса в командах LL и SC. Адреса являются одинаковыми, если виртуальный адрес, физический адрес и кеш-когерентный алгоритм идентичны.

Атомарная RMW последовательность предусмотрена только для кешируемых областей памяти. Экстент, для которого определение вида атомарности производится корректно, зависит от реализации системы и типа доступа к памяти (в данной области):

1) **MP атомарность.** Используется для реализации RMW последовательности между несколькими процессорами. Все обращения к участкам памяти должны выполняться при типе доступа к памяти «кешируемая когерентная».

2) **Монопроцессорная атомарность.** Используется для реализации RMW последовательности на 1 процессоре. Все обращения к участкам памяти должны выполняться при типе доступа к памяти «кешируемая когерентная» или «кешируемая некогерентная». Все обращения должны быть либо одного типа, либо другого, они не должны перемешиваться.

3) **Система ввода/вывода.** Используется для реализации RMW последовательности с когерентной системой ввода/вывода. Все обращения к участкам памяти должны выполняться при типе доступа к памяти «кешируемая когерентная». Если система ввода/вывода не использует операции с когерентной памятью, тогда RMW последовательность не может быть реализована.

```
if atomic_update then memory[base+offset] ← rt, rt ← 1 else rt ← 0
```

Ограничения: Адресуемая область памяти должна иметь тип доступа «кэшируемая когерентная» или «кэшируемая некогерентная», в противном случае результат неопределен.

Исполнительный адрес должен быть естественно-выровненным. Если один из двух младших битов адреса не равен нулю, возникает исключение «ошибка адреса».

Операция:

```
T:    vAddr ← sign_extend(offset) + GPR[base]
      if vAddr1..0 ≠ 02 then
        SignalException(AddressError)
      endif
      (pAddr, CCA)← AddressTranslation (vAddr,
      DATA, STORE)
      dataword← GPR[rt]
      if LLbit then
        StoreMemory (CCA, WORD, dataword,
      pAddr, vAddr, DATA)
      endif
      GPR[rt]← 031 || LLbit
T+1:  GPR[rt] ← mem
```

Исключения: TLB Refill, TLB Invalid, TLB Modified, Address Error, Reserved Instruction

2.65 SH - Сохраненние полуслова

31	25	20	15	0
SH 101001	base	rt	offset	

Формат: SH rt, offset(base)

Описание: Формируется исполнительный адрес: 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Младшее 16-битное полуслово из общего регистра *rt* записывается в память по полученному адресу.

Ограничения: Исполнительный адрес должен быть естественно-выровненным. Если один из двух младших битов адреса не равен нулю, возникает исключение «ошибка адреса».

Операция:

```
T:      vAddr ← ((offset15)^16 || offset15..0) + GPR[base]
        (pAddr,uncached) ← AddressTranslation(vAddr,DATA)
        pAddr ← pAddrPSIZE-1..2 ||(pAddr1..0 xor(ReservEndian ||0 ))
        byte ← vAddr1..0 xor (BigEndianRISCORE || 0 ))
        data ← GPR[rt] 31-8*byte..0 || 0^8*byte ?
        StoreMemory(uncached,HALFWORD,data,pAddr,vAddr,DA
        TA)
```

Исключения: TLB Refill, TLB Invalid, TLB Modified, Address Error

2.66 SLL - Логический сдвиг влево

31	25	20	15	10	5 0
SPECIAL 000000	00000	rt	rd	sa	SLL 000000

Формат: SLL rd, rt, sa

Описание: Значение общего регистра *rt*, сдвинутое влево на *sa* бит (младшие освободившиеся биты заполняются нулями), помещается в общий регистр *rd*.

Ограничения: нет

Операция:

```
T:          GPR[rd] ← GPR[rt]31-sa..0 ||0sa
```

Исключения: нет

Примечания программисту: Команда SLL r0, r0, 0, представленная как NOP, является идиомой, используемой, чтобы обозначить отсутствие операции.

Команда SLL r0, r0, 1, выраженная как SSNOP, является идиомой, используемой, чтобы обозначить отсутствие операции, которая вызывает прерывание выдачи на суперскалярных процессорах.

2.67 SLLV – Переменный логический сдвиг влево

31	25	20	15	10	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SLLV 000100

Формат: SLLV rd, rt, rs

Описание: Значение общего регистра *rt*, сдвинутое влево на заданную младшими 5 битами общего регистра *rs* величину (младшие освободившиеся биты заполняются нулями), помещается в общий регистр *rd.*

Ограничения: нет

Операция:

```
T:                    s                ←              GPR[rs]4..0
GPR[rd] ← GPR[rt] 31-s..0 ||0s
```

Исключения: нет

2.68 SLT - Установка по соотношению «меньше, чем»

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 00000	SLT 101010	

Формат: SLT rd, rs, rt

Описание: Значения общих регистров *rs* и *rt* сравниваются как целые со знаком. Если *rs* меньше *rt*, в общий регистр *rd* помещается 1, иначе - 0.

Арифметическое сравнение не вызывает исключение «целочисленное переполнение».

Ограничения: нет

Операция:

```
T:      if GPR[rs] < GPR[rt] then
        GPR[rd] ← 0^31 || 1
        else
         GPR[rd] ← 0^32
        endif
```

Исключения: нет

2.69 SLTI - Установка по соотношению «меньше, чем непосредственный операнд»

31	25	20	15	0
SLTI 001010	rs	rt	immediate	

Формат: SLTI rt,rs,immediate

Описание: Значения общего регистра *rs* и расширенного знаком *immediate* сравниваются как целые со знаком. Если *rs* меньше *immediate*, в общий регистр *rt* помещается 1, иначе - 0. Арифметическое сравнение не вызывает исключение «целочисленное переполнение».

Ограничения: нет

Операция:

```
T:      if GPR[rs] < (immediate15)^16 || immediate15..0 then
          GPR[rt] ← 0^31 ||1
        else
          GPR[rt] ← 0^32
        endif
```

Исключения: нет

2.70 SLTIU - Установка по соотношению «меньше, чем непосредственный операнд без знака»

31	25	20	15	0
SLTIU 001011	rs	rt	immediate	

Формат: SLTIU rt, rs, immediate

Описание: Значения общего регистра *rs* и расширенного знаком *immediate* сравниваются как целые без знака. Если *rs* меньше *immediate*, в общий регистр *rt* помещается 1, иначе - 0. Арифметическое сравнение не вызывает исключение «целочисленное переполнение».

Ограничения: нет

Операция:

```
T: if (0 || GPR[rs]) < 0 ||((immediate15)16 || immediate15..0) then
GPR[rt] ← 031 ||1
else
GPR[rt] ← 032
endif
```

Исключения: нет

2.71 SLTU - Установка по соотношению «меньше, чем» без знака

31	25	20	15	10	5 0
SPECIAL 000000	rs	rt	rd	0 00000	SLTU 101011

Формат: SLTU rd, rs, rt

Описание: Значения общих регистров *rs* и *rt* сравниваются как целые без знака . Если *rs* меньше *rt*, в общий регистр *rd* помещается 1, иначе - 0.

Арифметическое сравнение не вызывает исключение «целочисленное переполнение».

Ограничения: нет

Операция:

```
T: if (0 || GPR[rs]) < (0 || GPR[rt]) then
GPR[rd] ← 0^31 ||1
else
GPR[rd] ← 0^32
endif
```

Исключения: нет

2.72 SRA - Арифметический сдвиг вправо

31	25	20	15	10	5	0
SPECIAL 000000	0 00000	rt	rd	sa	SRA 000011	

Формат: SRA rd, rt, sa

Описание: Производится арифметический сдвиг вправо содержимого общего регистра *rt* на *sa* бит (освободившиеся биты заполняются знаковым битом исходного значения – битом 31). Результат помещается в общий регистр *rd.*

Ограничения: нет

Операция:

```
T: GPR[rd] ← (GPR[rt]31)^sa ||GPR[rt]31..sa
```

Исключения: нет

2.73 SRAV - Арифметический сдвиг вправо переменный

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 000000	SRAV 000111	

Формат: SRAV rd ,rt, rs

Описание: Производится арифметический сдвиг вправо содержимого общего регистра *rt* (освободившиеся биты заполняются знаковым битом исходного значения – битом 31). Сдвиг осуществляется на число бит, определяемое 5 младшими битами из общего регистра *rs* . Результат помещается в общий регистр *rd.*

Ограничения: нет

Операция:

```
T:                    s                    ←                    GPR[rs]4..0
GPR[rd] ← (GPR[rt]31)s ||GPR[rt] 31..s
```

Исключения: нет

2.74 SRL - Логический сдвиг вправо

31	25	20	15	10	5	0
SPECIAL 000000	00000	rt	rd	sa	SRL 000010	

Формат: SRL rd, rt, sa

Описание: Значение общего регистра *rt*, сдвинутое вправо на заданную младшими 5 битами общего регистра *rs* величину (старшие освободившиеся биты заполняются нулями), помещается в общий регистр *rd*.

Ограничения: нет

Операция:

```
T: GPR[rd] ← 0^sa ||GPR[rt]_31..sa
```

Исключения: нет

2.75 SRLV - Логический сдвиг вправо переменный

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 00000	SRLV 000110	

Формат: SRLV rd, rt, rs

Описание: Значение общего регистра *rt*, сдвинутое вправо на заданную младшими 5 битами общего регистра *rs* величину (старшие освободившиеся биты заполняются нулями), помещается в общий регистр *rd.*

Ограничения: нет

Операция:

```
T:                    s                ←                GPR[rs]4..0
GPR[rd] ← 0^s ||GPR[rt] 31..s
```

Исключения: нет

2.76 SUB – Вычитание слова

31	25	20	15	10	5 0
SPECIAL 000000	rs	rt	rd	0 000000	SUB 100010

Формат: SUB rd, rs, rt

Описание: Из содержимого регистра общего назначения *rs* и вычитается содержимое регистра общего назначения *rt* . 32-битный результат помещается в регистр общего назначения *rd.*

Если сложение приводит к арифметическому переполнению в дополнительном коде, регистр-приемник при этом не изменяется.

Если целочисленное переполнение не происходит, то результат вычитания записывается в общий регистр *rd.*

Ограничения: нет.

Операция:

```
T:      temp                                  (GPR[rs]31||GPR[rs]31..0)
        (GPR[rt]31||GPR[rt]31..0)
        if temp32 temp31 then
          SignalException(IntegerOverflow)
        else
          GPR[rd]  temp31..0
        endif
```

Исключения: Integer Overflow

Примечания программисту: команда SUBU выполняет ту же операцию, но не вызывает исключения «Целочисленное переполнение».

2.77 SUBU - Вычитание слова без знака

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 000000	SUBU 100011	

Формат: SUBU rd, rs, rt

Описание: 32-битное значение общего регистра *rt* вычитается из 32-битного значения общего регистра *rs*. 32-битный результат помещается в общий регистр *rd*.

Целочисленное переполнение не происходит ни при каких обстоятельствах.

Ограничения: нет

Операция:

```
T: GPR[rd] ← GPR[rs] - GPR[rt]
```

Исключения: нет

Примечание программисту: Термин “без знака” в имени команды – не совсем верный. Эта операция относится к 32-разрядной арифметике по модулю, которая не отслеживает переполнение. Эта команда для арифметики без знака, типа адресной арифметики, или целочисленного арифметического окружения, которое игнорирует переполнение.

2.78 SW - Сохранение слова

31	25	20	15	0
SW 101011	base	rt	offset	

Формат: SW rt, offset(base)

Описание: Для формирования исполнительного адреса 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Содержимое общего регистра *rt* записывается по полученному адресу.

Ограничения: Исполнительный адрес должен быть естественно-выровненным. Если хотя бы 1 из двух его младших битов не равен нулю, возникает исключение «Ошибка адреса».

Операция:

```
T: vAddr ← ((offset15)^16 || offset15..0) + GPR[base]
(pAddr,uncached) ← AddressTranslation(vAddr,DATA)
data ← GPR[rt]
StoreMemory(uncached,WORD,data,pAddr,vAddr,DATA)
```

Исключения: TLB Refill, TLB Invalid, TLB Modified, Address Error

2.79 SWL - Сохранение слова слева

31	25	20	15	0
SWL 101010	base	кt	offset	

Формат: SWL rt, offset(base)

Описание: Эта команда может использоваться в комбинации с SWR для сохранения содержимого регистра в 4-х последовательных байтах памяти, когда 2 байта пересекают границу двух слов. SWL сохраняет левую часть регистра *rt* в соответствующей части старшего слова памяти, SWR сохраняет правую часть регистра *rt* в соответствующей части младшего слова памяти.

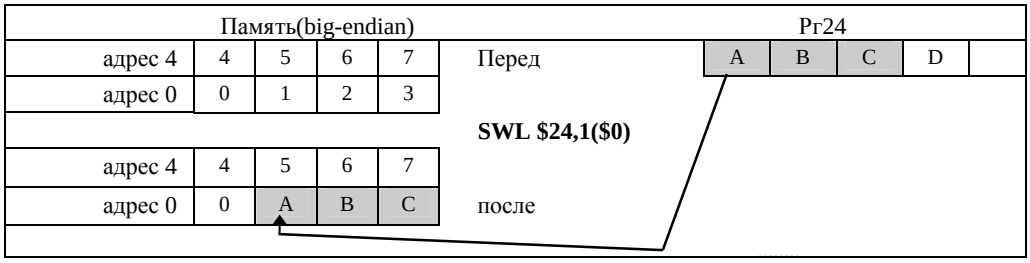
Для формирования исполнительного адреса 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Исполнительный адрес может характеризовать произвольный байт. Команда сохраняет от 1 до 4 байт в зависимости от стартового байта.

Механизм работы данной команды можно представить следующим образом.

Команда начинается с копирования старшего байта регистра в определенный стартовый байт памяти. Затем команда переходит к младшему байту регистра и сохраняет его в младшем байте слова памяти, копируя байты из регистра в память до тех пор, пока не будет достигнут самый младший байт слова в памяти.

Исключений из-за невыровненности адреса не происходит.

Ограничения: нет



Операция:

```
T: vAddr ← ((offset15)16 || offset15..0) + GPR[base]
(pAddr,uncached) ← AddressTranslation(vAddr,DATA)
pAddr ← pAddrpsize-1..2 || (pAddr1..0 xor ReverseEndian2)
byte ← vAddr1..0 xor BigEndianRISCORE2
if BigEndianMem = 0 then
pAddr ← pAddrpsize-1..2 ||02
endif
data ← 024-8*byte || GPR[rt]31..24-8*byte
StoreMemory(uncached,byte,data,pAddr,vAddr,DATA)
```

Исключения: TLB Refill, TLB Invalid, TLB Modified, Bus Error, Address Error

2.80 SWR - Сохранение слова справа

31	25	20	15	0
SWR 101110	base	rt	offset	

Формат: SWR rt, offset(base)

Описание: Эта команда может использоваться в комбинации с SWL для сохранения содержимого регистра в 4-х последовательных байтах памяти, когда 2 байта пересекают границу двух слов. SWL сохраняет левую часть регистра *rt* в соответствующей части старшего слова памяти, SWR сохраняет правую часть регистра *rt* в соответствующей части младшего слова памяти.

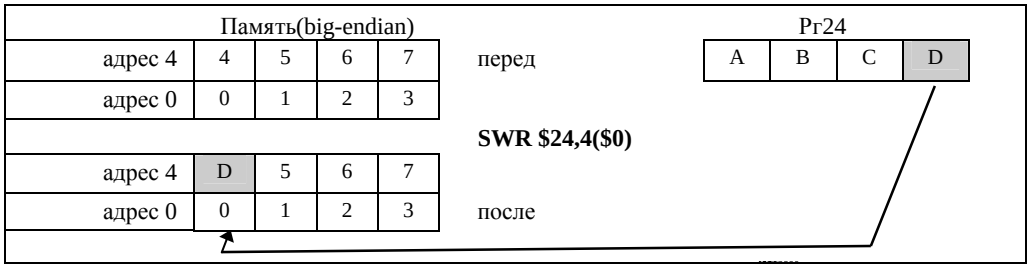
Для формирования исполнительного адреса 16-битная *offset* расширяется знаком и складывается с содержимым общего регистра *base*. Исполнительный адрес может характеризовать произвольный байт. Команда сохраняет от 1 до 4 байт в зависимости от стартового байта.

Механизм работы данной команды можно представить следующим образом.

Команда начинается с копирования младшего байта регистра в определенный стартовый байт памяти. Затем команда переходит к старшему байту регистра и сохраняет его в старшем байте слова памяти, копируя байты из регистра в память до тех пор, пока не будет достигнут самый старший байт слова в памяти.

Исключений из-за невыровненности адреса не происходит.

Ограничения: нет



Операция:

T: vAddr ← $((offset_{15})^{16} || offset_{15..0})$ + GPR[base]
(pAddr,uncached) ← AddressTranslation(vAddr,DATA)
pAddr ← $pAddr_{psize-1..2} || (pAddr_{1..0}$ *xor* $ReverseEndian^2)$
byte ← $vAddr_{1..0}$ *xor* $BigEndianRISCORE^2$
if BigEndianMem = 1 then
pAddr ← $pAddr_{psize-1..2} || 0^2$
endif
data ← $GPR[rt]_{31..8*byte} || 0^{8*byte}$
StoreMemory(uncached,byte,data,pAddr,vAddr,DATA)

Исключения: TLB Refill, TLB Invalid, TLB Modified, Bus Error, Address Error

2.81 SYNC – Синхронизация совместноиспользуемой памяти

31	25	10	5	0
SPECIAL 000000	0 00 0000 0000 0000 0	stype	SYNC 001111	

Формат: SYNC (подразумевается stype = 0).

Описание: Общее описание:

Используется для упорядочевания операций записи/загрузки.

Данная команда воздействует только на некэшированные и кэшированные когерентные операции записи/загрузки. Все операции записи/загрузки, выполняемые до команды SYNC, должны быть завершены прежде, чем будет разрешено исполняться командам записи/загрузки, которые расположены после команды SYNC.

Считается, что команды загрузки завершены, если в регистр назначения произведена запись; команды записи завершены, если записываемая величина доступна любому процессору системы.

Команда SYNC необходима, возможно, в сочетании с командой SSNOP, чтобы гарантировать, что результаты, связанные с работой с памятью, доступны при изменении режима работы. Например, команда SYNC необходима в некоторых реализациях во время перехода в режим отладки и выхода из него, чтобы гарантировать, что ошибки при работе с памятью будут обработаны верно.

Подробное описание:

Когда поле *stype* равно нулю, каждая синхронизируемая операция загрузки/записи которая выполняется в потоке команд до команды SYNC, должны быть глобально выполнены до того, как любая синхронизируемая команда загрузки/записи, находящаяся после команды SYNC, может быть выполнена. Это касается любого другого процессора или модуля ввода/вывода.

Команда SYNC не гарантирует, что команды будут выполняться в том же порядке, в каком производилась их выборка. Значения 1-31 поля *stype* зарезервированы. При таких значениях результат будет таким же, как и при нулевом значении.

Ограничения: Результат работы команды непредсказуем, если тип доступа к памяти не является «некэшированный когерентный» или «кэшированный когерентный».

Операция:

T:	`SyncOperation(stype)`

Исключения: нет

2.82 SYSCALL - Системный вызов

31	25	5 0
SPECIAL 000000	code 0000 0000 0000 0000 0000	SYSCALL 001100

Формат: SYSCALL

Описание: Возникает исключение System Call, и управление безусловно передается обработчику исключений.

Поле *code* доступно для использования в качестве программного параметра, но оно извлекается обработчиком исключения только загрузкой слова команды из памяти.

Ограничения: нет

Операция:

```
T: SystemCallException
```

Исключения: System Call

2.83 TEQ - Ловушка по равенству

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	code		TEQ 110100	

Формат: TEQ rs, rt

Описание: Если значение общего регистра *rs* равно значению общего регистра *rt*, то выполняется условная ловушка. Значения регистров интерпретируются как знаковые.

Содержимое поля code игнорируется системой и может быть использовано для кодированиия информации для системного ПО. Для получения этой информации программа должна загрузить слово команды из памяти.

Ограничения: нет

Операция:

T:	`if GPR[rs] = GPR[rt] then`
	`SignalException(Trap)`
	endif

Исключения: Trap

2.84 TEQI - Ловушка по равенству с непосредсвенным операндом

31	25	20	15	0
REGIMM 000001	rs	TEQI 01100	immediate	

Формат: TEQI rs, immediate.

Описание: Если значение общего регистра *rs* равно 16-битному *immediate*, расширенному знаком, то выполняется условная ловушка. Оба операнда интерпретируются как знакоые целые.

Ограничения: нет

Операция:

T:	`if GPR[rs] = sign_extend(immediate) then`
	`SignalException(Trap)`
	endif

Исключения: Trap

2.85 TGE - Ловушка по соотношению «больше или равно»

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	code		TGE 110000	

Формат: TGE rs, rt

Описание: Если значение общего регистра *rs* больше или равно значению общего регистра *rt*, то выполняется условная ловушка. Оба операнда интерпретируются как знаковые целые.

Содержимое поля code игнорируется системой и может быть использовано для кодированиия информации для системного ПО. Для получения этой информации программа должна загрузить слово команды из памяти.

.**Ограничения:** нет

Операция:

T:	`if GPR[rs] ≥ GPR[rt] then`
	`SignalException(Trap)`
	`endif`

Исключения: Trap

2.86 TGEI - Ловушка по соотношению «больше или равно» непосредственному операнду

31	25	20	15	0
REGIMM 000001	rs	TGEI 01000	immediate	

Формат: TGEI rs, immediate.

Описание: Если значение общего регистра *rs* больше или равно 16-битному *immediate*, расширенному знаком, то выполняется условная ловушка. Оба операнда интерпретируются как знакоые целые.

Ограничения: нет

Операция:

T:	`if GPR[rs] ≥ sign_extend(immediate) then`
	`SignalException(Trap)`
	`endif`

Исключения: Trap

2.87 TGEIU - Ловушка по соотношению «больше или равно» непосредсвенному операнду без знака

31	25	20	15 0
REGIMM 000001	rs	TGEIU 01001	immediate

Формат: TGEIU rs, immediate.

Описание: Если значение общего регистра *rs* больше или равно 16-битному *immediate*, расширенному знаком, то выполняется условная ловушка. Оба операнда интерпретируются как беззнакоые целые.

Так как *immediate* расширяется знаком до выполнения сравнения, в инструкции может быть представлено самое маленькое или самое большое беззнаковое число.

Ограничения: нет

Операция:

```
T: if (0 || GPR[rs]) ≥ (0 || sign_extend(immediate)) then
   SignalException(Trap)
   endif
```

Исключения: Trap

2.88 TGEU - Ловушка по соотношению «больше или равно» без знака

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	code		TGEU 110001	

Формат: TGEU rs, rt

Описание: Если значение общего регистра *rs* больше или равно значению общего регистра *rt*, то выполняется условная ловушка. Оба операнда интерпретируются как беззнаковые целые.

Содержимое поля code игнорируется системой и может быть использовано для кодированиия информации для системного ПО. Для получения этой информации программа должна загрузить слово команды из памяти.

Ограничения: нет

Операция:

T:	`if GPR[rs] ≥ GPR[rt] then`
	`SignalException(Trap)`
	`endif`

Исключения: Trap

2.89 TLBP - Проверка TLB на совпадающую запись

31	25	24	5	0
COP0 010000	CO 1	0 000 0000 0000 0000 0000		TLBP 001000

Формат: TLBP

Описание: В регистр Index загружается адрес записи TLB, содержимое которой совпадает с регистром *EntryHi*. Если такой записи нет, устанавливается старший бит регистра Index.

Ограничения: нет

Операция:

```
T:      index ← 1 || 0^31
for i in 0..TLBEntries-1
if (TLB[i]63..44=EntryHi31..12) and (TLB[i]8 or (TLB[i]43..38=EntryHi11..6)) then
index ← 0^18 ||i5..0 ||0^8
endif
endfor
```

Исключения: Coprocessor Unusable

2.90 TLBR - Чтение из TLB по регистру Index

31	25	24	5	0
COP0 010000	CO 1	0 000 0000 0000 0000 0000	TLBR 000001	

Формат: TLBR

Описание: В регистры EntryHi, EntryLo0, EntryLo1 и PageMask загружается содержимое записи TLB, на которую указывает регистр Index. Следует отметить, что значения, записываемые в регистры EntryHi, EntryLo0, EntryLo1 могут отличаться от значений, которые были первоначально записаны в TLB через эти же регистры. Различия могут быть в следующем:

1) У значения, возвращаемого в поле VPN2 регистра EntryHi, могут быть установлены в 0 те биты, которые были установлены в 1 в поле Mask записи TLB (самый младший бит поля VPN2 соответствует самому младшему биту поля Mask). Зависит от реализации, зарезервированы эти биты, или они обнуляются после того, как будет произведена запись TLB, а затем чтение этой записи.

2) У значения, возвращаемого в поле PFN регистров EntryLo0 и EntryLo1, могут быть установлены в 0 те биты, которые были установлены в 1 в поле Mask записи TLB (самый младший бит поля PFN соответствует самому младшему биту поля Mask). Зависит от реализации, зарезервированы эти биты, или они обнуляются после того, как будет произведена запись TLB, а затем чтение этой записи.

3) Значение, возвращаемое в бит G, регистров EntryLo0 и EntryLo1, берется только из одного бита G в записи TLB. (А при записи в TLB этот бит был результатом логического «И» двух G битов регистров EntryLo0 и EntryLo1).

Ограничения: Данная операция НЕОПРЕДЕЛЕНА, если значение регистра Index больше или равно числу записей TLB в процессоре.

Операция:

```
T:      EntryHi  ←  TLB[Index13..8]63..32
EntryLo ← TLB[Index13..8]31..0
```

Исключения: Coprocessor Unusable

2.91 TLBWI - Запись в TLB по регистру Index

31	25	24	5	0
COP0 010000	CO 1	0 000 0000 0000 0000 0000	TLBWI 000010	

Формат: TLBWI

Описание: Создается запись TLB, указанная в регистре Index. Запись производится из регистров EntryHi, EntryLo0, EntryLo1 и PageMask. Следует отметить, что значения, записываемые в TLB, могут отличаться от значений в регистрах EntryHi, EntryLo0, EntryLo1. Различия могут быть в следующем:

1) У значения, записываемого в поле VPN2 записи TLB, могут быть установлены в 0 те биты, которые были установлены в 1 в поле Mask регистра PageMask (самый младший бит поля VPN2 соответствует самому младшему биту поля Mask). Зависит от реализации, зарезервированы эти биты, или они обнуляются во время записи в TLB.

2) У значения, записываемого в поля PFN0 и PFN1 записи TLB, могут быть установлены в 0 те биты, которые были установлены в 1 в поле Mask регистра PageMask (самый младший бит поля PFN соответствует самому младшему биту поля Mask). Зависит от реализации, зарезервированы эти биты, или они обнуляются во время записи в TLB.

3) Единственный бит G в записи TLB устанавливается как результат логического «И» двух битов G из регистров EntrLo0 и EntryLo1.

Ограничения: Данная операция НЕОПРЕДЕЛЕНА, если значение регистра Index больше или равно числу записей TLB в процессоре.

Операция:

```
T: TLB[Index13..8] ← EntryHi ||EntryLo
```

Исключения: Coprocessor Unusable

2.92 TLBWR - Запись в TLB по регистру Random

31	25	24	5	0
COP0 010000	CO 1	0 000 0000 0000 0000 0000	TLBWR 000110	

Формат: TLBWR

Описание: В регистры EntryHi, EntryLo0, EntryLo1 и PageMask загружается содержимое записи TLB, на которую указывает регистр Random. Следует отметить, что значения, записываемые в регистры EntryHi, EntryLo0, EntryLo1 могут отличаться от значений, которые были первоначально записаны в TLB через эти же регистры. Различия могут быть в следующем:

1) У значения, возвращаемого в поле VPN2 регистра EntryHi, могут быть установлены в 0 те биты, которые были установлены в 1 в поле Mask записи TLB (самый младший бит поля VPN2 соответствует самому младшему биту поля Mask). Зависит от реализации, зарезервированы эти биты, или они обнуляются после того, как будет произведена запись TLB, а затем чтение этой записи.

2) У значения, возвращаемого в поле PFN регистров EntryLo0 и EntryLo1, могут быть установлены в 0 те биты, которые были установлены в 1 в поле Mask записи TLB (самый младший бит поля PFN соответствует самому младшему биту поля Mask). Зависит от реализации, зарезервированы эти биты, или они обнуляются после того, как будет произведена запись TLB, а затем чтение этой записи.

3) Значение, возвращаемое в бит G, регистров EntryLo0 и EntryLo1, берется только из одного бита G в записи TLB. (А при записи в TLB этот бит был результатом логического «И» двух G битов регистров EntryLo0 и EntryLo1).

Ограничения: Данная операция НЕОПРЕДЕЛЕНА, если значение регистра Index больше или равно числу записей TLB в процессоре.

Операция:

```
T: TLB[Random13..8] ← EntryHi ||EntryLo
```

Исключения: Coprocessor Unusable

2.93 TLT - Ловушка по соотношению «меньше, чем»

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	code		TLT 110010	

Формат: TLT rs, rt

Описание: Если значение общего регистра *rs* меньше значения общего регистра *rt*, то выполняется условная ловушка. Оба операнда интерпретируются как знаковые целые.

Содержимое поля code игнорируется системой и может быть использовано для кодированиия информации для системного ПО. Для получения этой информации программа должна загрузить слово команды из памяти.

Ограничения: нет

Операция:

T:	`if GPR[rs] < GPR[rt] then`
	`SignalException(Trap)`
	`endif`

Исключения: Trap

2.94 TLTI - Ловушка по соотношению «меньше, чем» непосредсвенный операнд

31	25	20	15	0
REGIMM 000001	rs	TLTI 01010	immediate	

Формат: TLTI rs, immediate.

Описание: Если значение общего регистра *rs* меньше 16-битного *immediate*, расширенного знаком, то выполняется условная ловушка. Оба операнда интерпретируются как знакоые целые.

Ограничения: нет

Операция:

T:	`if GPR[rs] < sign_extend(immediate) then`
	`SignalException(Trap)`
	`endif`

Исключения: Trap

2.95 TLTIU - Ловушка по соотношению «меньше, чем» непосредсвенный операнд без знака

31	25	20	15	0
REGIMM 000001	rs	TLTIU 01011	immediate	

Формат: TLTI rs, immediate.

Описание: Если значение общего регистра *rs* меньше 16-битного *immediate*, расширенного знаком, то выполняется условная ловушка. Оба операнда интерпретируются как беззнакоые целые.

Так как *immediate* расширяется знаком до выполнения сравнения, в инструкции может быть представлено самое маленькое или самое большое беззнаковое число.

Ограничения: нет

Операция:

T:	`if GPR[rs] < sign_extend(immediate) then`
	`SignalException(Trap)`
	`endif`

Исключения: Trap

2.96 TLTU - Ловушка по соотношению «меньше, чем» без знака

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	code		TLTU 110011	

Формат: TLTU rs, rt

Описание: Если значение общего регистра *rs* меньше значения общего регистра *rt*, то выполняется условная ловушка. Оба операнда интерпретируются как беззнаковые целые.

Содержимое поля code игнорируется системой и может быть использовано для кодированиия информации для системного ПО. Для получения этой информации программа должна загрузить слово команды из памяти.

Ограничения: нет

Операция:

T:	`if (0 || GPR[rs]) < (0 || GPR[rt]) then`
	`SignalException(Trap)`
	`endif`

Исключения: Trap

2.97 TNE - Ловушка по соотношению «не равно»

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	code		TNE 110110	

Формат: TNE rs, rt

Описание: Если значение общего регистра *rs* не равно значению общего регистра *rt*, то выполняется условная ловушка. Значения регистров интерпретируются как знаковые.

Содержимое поля code игнорируется системой и может быть использовано для кодированиия информации для системного ПО. Для получения этой информации программа должна загрузить слово команды из памяти.

Ограничения: нет

Операция:

T:	`if GPR[rs] ≠ GPR[rt] then`
	`SignalException(Trap)`
	`endif`

Исключения: Trap

2.98 TNEI - Ловушка по соотношению «не равно» непосредсвенному операнду

31	25	20	15 0
REGIMM 000001	rs	TNEI 01110	immediate

Формат: TNEI rs, immediate.

Описание: Если значение общего регистра *rs* равно 16-битному *immediate*, расширенному знаком, то выполняется условная ловушка. Оба операнда интерпретируются как знакоые целые.

Ограничения: нет

Операция:

T:	if GPR[rs] □□sign_extend(immediate) then
	SignalException(Trap)
	endif

Исключения: Trap

2.99 WAIT – Переход в режим ожидания

31	25	24	5	0
COP0 010000	CO 1	Implementation-Dependent Code	WAIT	

Формат: WAIT

Описание: Данная команда представляет собой зависимую от реализации операцию, обычно вызывающую переход режим сбережения энергии. Программное обеспечение может использовать биты 24:6 для передачи дополнительной информации в процессор. Процессор может использовать такую информацию как контрольную для режима пониженной энергии. Нулевое значение этих битов является их значением по умолчанию и должно быть включено в любую реализацию.

Ограничения: Действия процессора **НЕОПРЕДЕЛЕНЫ**, если данная команда находится в слоте задержки перехода или ветвления.

Операция:

```
Enter implementation dependent lower power mode
```

Исключения: Coprocessor Unusable

2.100 XOR - Логическое исключающее «ИЛИ»

31	25	20	15	10	5	0
SPECIAL 000000	rs	rt	rd	0 00000	XOR 100110	

Формат: XOR rd, rs, rt

Описание: Результат логического поразрядного исключающего «ИЛИ» значений общих регистров *rs* и *rt* помещается в общий регистр *rd.*

Ограничения: нет

Операция:

```
T: GPR[rd] ← GPR[rs] xor GPR[rt]
```

Исключения: нет

2.101 XORI - Логическое исключающее «ИЛИ» с непосредственным операндом

31	25	20	15	0
XORI 001110	rs	rt	immediate	

Формат: XORI rt, rs, immediate

Описание: Результат логического поразрядного исключающего «ИЛИ» значения *immediate*, расширенного нулями до 32 бит, и значения общего регистра *rs* помещается в общий регистр *rt.*

Ограничения: нет

Операция:

```
T: GPR[rt] ← GPR[rs] xor (016 ||immediate)
```

Исключения: нет

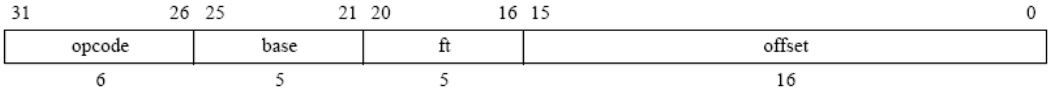
3. СИСТЕМА КОМАНД СОПРОЦЕССОРА С ПЛАВАЮЩЕЙ ТОЧКОЙ

В этом разделе рассмотрена система команд сопроцессора с плавающей точкой (сопроцессора 1).

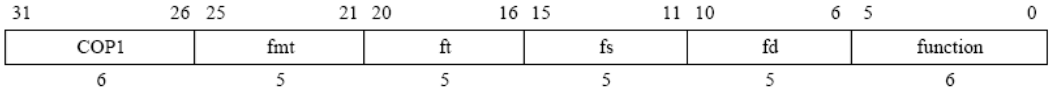
3.1 Классы и форматы команд

Каждая инструкция сопроцессора с плавающей точкой представляет собой 32-х битное выровненное слово. Ниже приведены форматы команд сопроцессора с плавающей точкой.

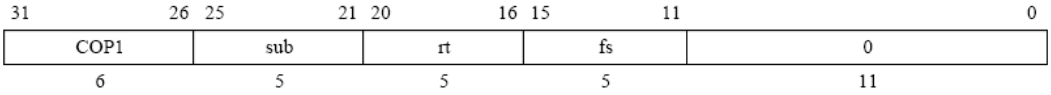
1) **I-Type (Immediate)** формат



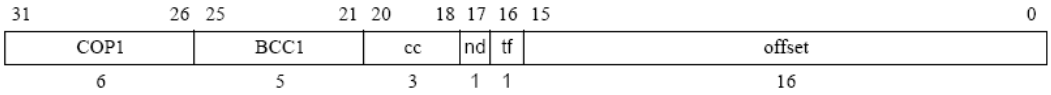
2) **R-Type (Register)** формат



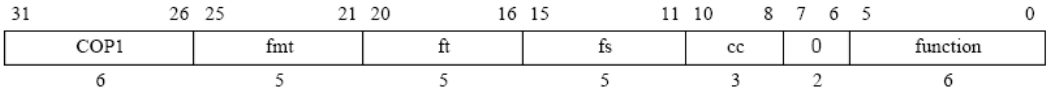
3) **Register-Immediate** формат



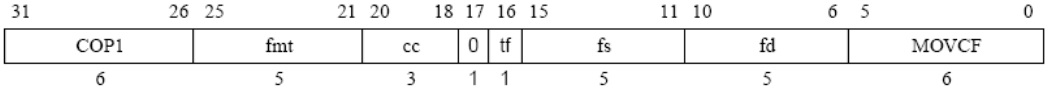
4) **Condition Code, Immediate** формат



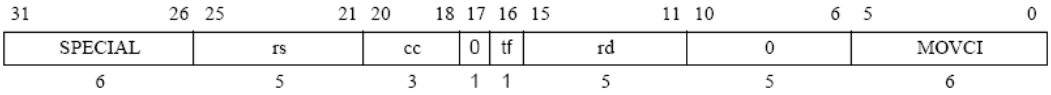
5) **Formatted FPU Compare** формат



6) **Floating Point Register Move, Conditional** формат



7) **Condition Code, Register Integer** формат



В следующей таблице приведены описания полей команд сопроцессора с плавающей точкой.

Поле	**Описание**
BC1	Субкод команды условного перехода
base	Регистр CPU: базовый адрес для вычисления адресов
COP1	Первичный код операции для команд сопроцессора 1 (в поле *op*)
cc	Спецификатор кода условия
fd	Регистр FPU: приемник (арифметические команды, команды загруки, пересылки "в") или источник (команды сохранения, пересылки "из")
fmt	Спецификатор формата приемника и/или формата операнда
fr	Регистр FPU: источник
fs	Регистр FPU: источник
ft	Регистр FPU: источник (команды сохранения, арифметические команды) или приемник (команды загрузки)
function	Спецификатор команды при частных значениях кода операции
index	Регистр CPU, содержащий компоненту адресного индекса для адресных вычислений
MOVC	Значение в поле function для условной пресылки. Существует одно значение для команды, когда op = COP1, и другое, когда op = SPECIAL
nd	Отменить задержку. Если установлено, переход является переходом по вероятности, и команда в слоте задержки не выполняется
offset	Знаковая величина, используется в адресных вычислениях
op	Первичный код операции
rd	Регистр CPU: приемник
rs	Регистр CPU: источник
rt	Регистр CPU: может быть либо источником, либо приемником
SPECIAL	Первичный код SPECIAL
sub	Субкод команды для команд сопроцессора 1 типа "immediate"
tf	True/False. Условие, полученное при выполнении команд сравнения, которые проверяется на равенство биту tf

3.1.1 Команды пересылки данных

Сопроцессор с плавающей точкой содержит два набора регистров: регистры общего назначения и управляющие регистры. Архитектура сопроцессора 1 - это архитектура загрузки/записи. Все вычисления производятся над данными, которые хранятся в общих регистрах сопроцессора. Управляющие регистры используются для управления операциями сопроцессора. Пересылка данных между регистрами и остальной системой осуществляется с помощью специально предназначенных для этого команд загрузки, записи и пересылки. Пересылаемые данные интерпретируются как неформатированные двоичные данные - преобразование форматов не производится, и, поэтому, исключения операций с плавающей запятой стандарта IEEE произойти не могут.

3.1.2 Арифметические команды

Арифметические команды работают с форматированными данными. Результаты большинства таких операций соответствуют спецификации стандарта IEEE-754 для точности - результат идентичен результату бесконечной точности, округленному в рамках определенного формата данных согласно заданному режиму округления. Округленный результат отличается от точного менее, чем на 1 элемент в наименее значимой позиции (Unit in the Least-significant Place).

3.1.3 Команды преобразования типа данных

Данные команды производят преобразования форматов с плавающей точкой и фиксированной точкой. Каждая команда преобразует операнд в одном из форматов в один результат определнного формата. Некоторые команды преобразования типа используют режим округления, заданный в управляющем регистре FCSR, другие - определяют необходимый режим округления непосредственно.

3.1.4 Команды пересылки форматированных значений

Все эти команды обеспечивают пересылку операндов между общими регистрами сопроцессора с плавающей запятой. Каждый тип операнд должен пересылаться командой, которая оперирует с операндами именно данного типа. Существует три типа команд пересылки данных:

1) безусловная пересылка;
2) условная пересылка, при которой производится проверка кода условия сопроцессора 1 на истину/ложь;
3) условная пересылка, при которой производится проверка на равенство 0 общего регистра центрального процессора.

Результат условных пересылок может быть неожиданным. Они всегда присваивают числу в регистре назначения именно тот формат, который указан в команде. Если до завершения выполнения такой команды регистр назначения не содержит операнда заданного формата, то содержимое этого регистра становится неопределенным.

3.1.5 Команды условного перехода

В системе команд сопроцессора с плавающей точкой есть команды PC-зависимых условных ветвлений, которые проверяют коды условия, выставленные командами сравнения (C.cond.fmt). Все переходы архитектурно имеют задержку в 1 команду. Когда совершается переход, то говорят, что команда, непосредственно следующая за командой ветвления, находится в слоте задержки. Эта команда выполняется до совершения перехода по заданному адресу. Команды условного перехода бывают двух видов, в зависимости от того, как они обрабатывают команду в слоте задержки в ситуации, когда переход не совершается:

1) **команды ветвления**, которые исполняют команду в слоте задержки;
2) **команды ветвления по вероятности**, которые не исполняют команду в слоте задержки, если переход не осуществляется (команда в слоте задержки отменяется - *nullify*).

Хотя команды ветвления по вероятности и включены в систему команд, настоятельно рекомендуется не использовать их в программном обеспечении, так как в будущих версиях MIPS-архитектуры планируется исключить их из системы команд.

По архитектуре MIPS32 определены 8 кодов условия для использования их в командах сравнения и ветвления. Для совместимости с предыдущими версиями ISA (Instruction Set Architecture) бит 0 кодов условия и биты 1 - 7 являются отдельными полями в регистре FCSR.

3.1.6 Смешанные команды

В системе команд архитектуры MIPS определены смешанные команды, которые осуществляют условную пересылку значений из одного общего регистра центрального процессора в другой. Условия считываются из кодов условия сопроцессора с плавающей точкой.

4. ОПИСАНИЕ СИСТЕМЫ КОМАНД СОПРОЦЕССОРА С ПЛАВАЮЩЕЙ ТОЧКОЙ

В данном разделе приводится подробное описание выполнения команд сопроцессора 1 (сопроцессора с плавающей точкой). Рассматриваются все возможные в ходе выполнения каждой команды исключения.

Команды сопроцессора с плавающей точкой

Команды пересылки данных

LDC1	Load Doubleword to Floating Point
LWC1	Load Word to Floating Point
SDC1	Store Doubleword from Coprocessor 1
SWC1	Store Word from Coprocessor 1
CFC1	Move Control Word From Floating Point
CTC1	Move Control Word To Floating Point
MFC1	Move Word From Floating Point
MTC1	Move Word To Floating Point

Арифметические команды

ABS.fmt	Floating Point Absolute Value
ADD.fmt	Floating Point Add
C.cond.fmt	Floating Point Compare
DIV.fmt	Floating Point Divide
MUL.fmt	Floating Point Multiply
NEG.fmt	Floating Point Negate
SQRT.fmt	Floating Point Square Root
SUB.fmt	Floating Point Subtract

Команды преобразования данных

CVT.D.fmt	Floating Point Convert to Double Floating Point
CVT.S.fmt	Floating Point Convert to Single Floating Point
CVT.W.fmt	Floating Point Convert to Word Fixed Point
CEIL.W.fmt	Floating Point Ceiling Convert to Word Fixed Point
FLOOR.W.fmt	Floating Point Floor Convert to Word Fixed Point
ROUND.W.fmt	Floating Point Round to Word Fixed Point
TRUNC.W.fmt	Floating Point Truncate to Word Fixed Point

Команды пересылки форматированных значений

MOV.fmt	Floating Point Move
MOVF.fmt	Floating Point Move Conditional on Point False
MOVT.fmt	Floating Point Move Conditional on Point True
MOVN.fmt	Floating Point Move Conditional on Not Zero
MOVZ.fmt	Floating Point Move Conditional on Zero

Команды условного перехода

BC1F	Branch on Floating Point False
BC1T	Branch on Floating Point True
BC1FL	Branch on Floating Point False Likely
BC1TL	Branch on Floating Point True Likely

4.1 ABS.fmt, модуль числа

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	00000	fs	fd	ABS 000101

Формат: ABS.S fd, fs

ABS.D fd, fs

Описание: `fd ← abs(fs)`

Абсолютное значение содержимого общего регистра сопроцессора с плавающей точкой (FPR) *fs* помещается в FPR *fd*. Операнд и результат – величины формата *fmt.*

Данная операция является арифметической; операнды NaN вызывают исключение Invalid Operation.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМО.

Операция:

```
StoreFPR(fd, fmt, AbsoluteValue(ValueFPR(fs, fmt)))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation, Invalid Operation

4.2 ADD.fmt, сложение

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	ft	fs	fd	ADD 000000

Формат: ADD.S fd, fs, ft

ADD.D fd, fs, ft

Описание: `fd ← fs + ft`

Значение FPR *ft* прибавляется к значению FPR *fs*. Результат вычисляется с бесконечной точностью и округляется согласно текущему режиму округления, который задается в управляющем регистре FCSR. Результат помещается в FPR *fd*. Операнды и результат операции – величины формата *fmt*. Если исключение не возникает, выставляются биты *flags* посредством операции логического «ИЛИ» над битами *cause* в управляющих регистрах.

Ограничения: Поля *fs, ft* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнды должны быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значения операндов также становятся НЕПРЕДСКАЗУЕМЫ.

Операция:

```
StoreFPR (fd, fmt, ValueFPR(fs, fmt) +fmt ValueFPR(ft, fmt))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation, Invalid Operation, Inexact, Overflow, Underflow

4.3 BC1F, ветвление по соотношению «ЛОЖЬ»

31 26	25 21	20 18	17	16	15 0
COP1 010001	BC 01000	cc	nd 0	tf 0	offset

Формат: BC1F cc, offset

Описание: `if cc = 0 then branch`

Для формирования PC-относительного исполнительного адреса назначения 18-битная знаковая величина offset (16-битное поле *offset*, сдвинутое на 2 бита влево) складывается с адресом команды в слоте задержки ветвления – следующей за командой ветвления (не с адресом самой команды ветвления!). Если бит *cc* кода условия (в управляющих регистрах сопроцессора с плавающей точкой) сброшен (FALSE), после выполнения команды, находящейся в слоте задержки, происходит переход по вычисленному ранее исполнительному адресу.

Код условия устанавливается командой сравнения C.cond.fmt.

Ограничения: Результат выполнения команды НЕПРЕДСКАЗУЕМ, если команды ветвления, перехода или команды ERET, DERET или WAIT находятся в слоте задержки перехода или ветвления

Операция:

```
I: condition ← FPConditionCode(cc) = 0
target_offset ← (offset15)GPRLEN-(16+2) || offset || 02
I+1: if condition then
PC ← PC + target_offset
endif
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation

Примечание программисту: С 18-битным знаковым смещением диапазон условного перехода составляет ±128 Кбайт. Для перехода на адрес вне этого диапазона, используйте команды J и JR.

4.4 BC1FL, ветвление по соотношению «ЛОЖЬ» по вероятности

31 26	25 21	20 18	17	16	15 0
COP1 010001	BC 01000	cc	nd 1	tf 0	offset

Формат: BC1FL cc, offset

Описание: `if cc = 0 then branch_likely`

Для формирования PC-относительного исполнительного адреса назначения 18-битная знаковая величина *offset* (16-битное поле *offset*, сдвинутое на 2 бита влево) складывается с адресом команды в слоте задержки ветвления – следующей за командой ветвления (не с адресом самой команды ветвления!). Если бит *cc* кода условия (в управляющих регистрах сопроцессора с плавающей точкой) сброшен (FALSE), после выполнения команды, находящейся в слоте задержки, происходит переход по вычисленному ранее исполнительному адресу. Если переход не совершается, команда в слоте задержки не выполняется.

Код условия устанавливается командой сравнения C.cond.fmt.

Ограничения: Результат выполнения команды НЕПРЕДСКАЗУЕМ, если команды ветвления, перехода или команды ERET, DERET или WAIT находятся в слоте задержки перехода или ветвления

Операция:

```
I: condition ← FPConditionCode(cc) = 0
target_offset ← (offset15)GPRLEN-(16+2) || offset || 02
I+1: if condition then
PC ← PC + target_offset
else
NullifyCurrentInstruction()
endif
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation

Примечание программисту: С 18-битным знаковым смещением диапазон условного перехода составляет ±128 Кбайт. Для перехода на адрес вне этого диапазона, используйте команды J и JR.

Рекомендуется не использовать её в ПО, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что, если переход не происходит, мы имеем значительный штраф по времени исполнения. Программное обеспечение должно использовать эту команду только, если существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BC1F.

4.5 BC1T, ветвление по соотношению «ИСТИНА»

31 26	25 21	20 18	17	16	15 0
COP1 010001	BC 01000	cc	nd 0	tf 1	offset

Формат: BC1T cc, offset

Описание: `if cc = 1 then branch`

Для формирования PC-относительного исполнительного адреса назначения 18-битная знаковая величина *offset* (16-битное поле *offset*, сдвинутое на 2 бита влево) складывается с адресом команды в слоте задержки ветвления – следующей за командой ветвления (не с адресом самой команды ветвления!). Если бит *cc* кода условия (в управляющих регистрах сопроцессора с плавающей точкой) установлен (TRUE), после выполнения команды, находящейся в слоте задержки, происходит переход по вычисленному ранее исполнительному адресу.

Код условия устанавливается командой сравнения C.cond.fmt.

Ограничения: Результат выполнения команды НЕПРЕДСКАЗУЕМ, если команды ветвления, перехода или команды ERET, DERET или WAIT находятся в слоте задержки перехода или ветвления

Операция:

```
I: condition ← FPConditionCode(cc) = 1
target_offset ← (offset15)GPRLEN-(16+2) || offset || 02
I+1: if condition then
PC ← PC + target_offset
endif
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation

Примечание программисту: С 18-битным знаковым смещением диапазон условного перехода составляет ±128 Кбайт. Для перехода на адрес вне этого диапазона, используйте команды J и JR.

4.6 BC1TL, ветвление по соотношению «ИСТИНА» по вероятности

31 26	25 21	20 18	17	16	15 0
COP1 010001	BC 01000	cc	nd 1	tf 1	offset

Формат: BC1TL cc, offset

Описание: `if cc = 1 then branch_likely`

Для формирования PC-относительного исполнительного адреса назначения 18-битная знаковая величина *offset* (16-битное поле *offset*, сдвинутое на 2 бита влево) складывается с адресом команды в слоте задержки ветвления – следующей за командой ветвления (не с адресом самой команды ветвления!). Если бит *cc* кода условия (в управляющих регистрах сопроцессора с плавающей точкой) установлен (TRUE), после выполнения команды, находящейся в слоте задержки, происходит переход по вычисленному ранее исполнительному адресу. Если переход не совершается, команда в слоте задержки не выполняется.

Код условия устанавливается командой сравнения C.cond.fmt.

Ограничения: Результат выполнения команды НЕПРЕДСКАЗУЕМ, если команды ветвления, перехода или команды ERET, DERET или WAIT находятся в слоте задержки перехода или ветвления.

Операция:

```
I: condition ← FPConditionCode(cc) = 1
target_offset ← (offset15)GPRLEN-(16+2) || offset || 02
I+1: if condition then
PC ← PC + target_offset
else
NullifyCurrentInstruction()
endif
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation

Примечание программисту: С 18-битным знаковым смещением диапазон условного перехода составляет ±128 Кбайт. Для перехода на адрес вне этого диапазона, используйте команды J и JR.

Рекомендуется не использовать её в ПО, поскольку в будущих версиях планируется исключить эту инструкцию из MIPS архитектуры.

Некоторые реализации часто предполагают, что переход будет происходить и организуют конвейер таким образом, что, если переход не происходит, мы имеем значительный штраф по времени исполнения. Программное обеспечение должно использовать эту команду только, если существует очень высокая вероятность (98 % или больше) перехода. Если переход маловероятен, или если вероятность перехода неизвестна, то рекомендуется использовать команду BC1T.

4.7 C.cond.fmt, сравнение

31 26	25 21	20 15	15 11	10 8	7	6	5 4	3 0
COP1 010001	fmt	ft	fs	cc	0	A 0	FC 11	cond

Формат: C.cond.S cc, fs, ft

C.cond.D cc, fs, ft

Описание: `cc` ← `fs` *compare_cond* `ft`

Содержимое FPR *fs* сравнивается с содержимым FPR *ft*. Операнды – величины в формате *fmt*. Сравнение является точным и не вызывает переполнения и потери значимости.

Если сравнение, определяемое битами $cond_{2...0}$, имеет значение «ИСТИНА» для операндов команды, то результат принимает значение «ИСТИНА». В противном случае, результат – «ЛОЖЬ». Если не возникает исключение, результат записывается в код условия *cc* («ИСТИНА» 1, «ЛОЖЬ-0»).

Если один из операндов является SNaN или бит $cond_3$ установлен и хотя бы один из операндов является QNaN, возникает условие исключения Invalid Operation, и в управляющем регистре FCSR выставляется флаг Invalid Operation. Если в управляющем регистре FCSR установлен *enable* бит Invalid Operation, результат не записывается и незамедлительно происходит исключение Invalid Operation. В противном случае, булев результат записывается в код условия *cc*.

Существует четыре взаимоисключающих отношения упорядочения для сравнения чисел с плавающей точкой – одно отношение всегда принимает значение «ИСТИНА», а остальные – «ЛОЖЬ». Стандартными отношениями являются «больше, чем», «меньше, чем» и «равно». Кроме того, стандарт IEEE для чисел с плавающей точкой определяет отношение «неупорядочено», которое принимает значение «ИСТИНА», когда хотя бы один из операндов является NaN. Величина NaN при сравнении с любым значением, включая себя саму, дает результат «неупорядочено». Операции сравнения игнорируют знак нуля, т.е. +0 равняется -0.

Условие сравнения – логический предикат отношений упорядочения, таких как «меньше, чем или равно», «равно», «не меньше, чем» или «неупорядочено или равно». Сравнение различается между 16 предикатами сравнения. Булев результат команды получается замещением булевых значений каждого отношения упорядочения на два числа с плавающей точкой в выражении.

Например, если отношение «равно» истинно, то все четыре вышеупомянутых предиката приводят к результату «ИСТИНА». Если отношение «неупорядочено» истинно, тогда только последний предикат, «неупорядочено или равно», даст результат «ИСТИНА».

Логическое отрицание результата сравнения позволяет проводить восемь различных сравнений для проверки 16 предикатов, как показано ниже в таблице 1. В каждом случае, команда compare проверяет истинность первого предиката. Если первый предикат истинен, результат принимает значение «ИСТИНА», как показано в графе «если предикат истинен», и второй предикат должен быть ложен и наоборот.

Истинность второго предиката – это логическое отрицание результата команды. После команды сравнения проверка на истинность первого предиката может быть выполнена командой BC1T, а истинность второго предиката может быть проверена командой BC1F.

Таблица 13. Операции сравнения FPU без исключений для специальных операндов

Команда	Предикат сравнения					Результат сравнения *cc*		Команда	
Мнемоника условия (*cond*)	Имя или предикат и его логическое отрицание (аббревиатура)	Значения отношения				Если предикат – «ИСТИНА»	Исключение «Invalid Operation», если есть величина QNaN?	Поле условия (*cond*)	
		>	<	=	?			3	2…0
F	«ЛОЖЬ»	F	F	F	F	F	Нет	0	0
	«ИСТИНА» (T)	T	T	T	T				
UN	неупорядочено	F	F	F	T	T			1
	упорядочено (OR)	T	T	T	F	F			
EQ	равно	F	F	T	F	T			2
	не равно (NEQ)	T	T	F	T	F			
UEQ	неупорядочено или равно	F	F	T	T	T			3
	упорядочено или больше, чем или меньше, чем (OGL)	T	T	F	F	F			
OLT	упорядочено или меньше, чем	F	T	F	F	T			4
	неупорядочено или больше, чем или равно (UGE)	T	F	T	T	F			
ULT	неупорядочено или меньше, чем	F	T	F	T	T			5
	упорядочено или больше, чем или равно (OGE)	T	F	T	F	F			
OLE	упорядочено или меньше, чем или равно	F	T	T	F	T			6
	неупорядочено или больше, чем (UGT)	T	F	F	T	F			
ULE	неупорядочено или меньше, чем или равно	F	T	T	T	T			7
	упорядочено или больше, чем (OGT)	T	F	F	F	F			

Условные обозначения: ? – неупорядочено; > - больше, чем; < - меньше, чем; = - равно.

В таблице 2 приведен набор других восьми операций сравнения, характеризуемых значением бита $cond_3$ равным 1, и проверяющих те же 16 условий. Для этих дополнительных сравнений возникает условие исключения Invalid Operation, если хотя бы один из операндов является NaN,

включая QNaN. Если перехват этого исключения активирован в управляющем регистре FCSR, происходит исключение Invalid Operation.

Таблица 14. Операции сравнения FPU с исключениями для специальных операндов (QNaN).

Команда	Предикат сравнения					Результат сравнения cc		Команда	
Мнемоника условия (cond)	Имя или предикат и его логическое отрицание (аббревиатура)	Значения отношения				Если предикат – «ИСТИНА»	Исключение «Invalid Operation», если есть величина QNaN?	Поле условия (cond)	
		>	<	=	?			3	2…0
SF	сигнальная «ЛОЖЬ»	F	F	F	F	F	Да	1	0
	сигнальная «ИСТИНА» (ST)	T	T	T	T				
ngle	не больше, чем или меньше, чем или равно	F	F	F	T	T			1
	больше, чем или меньше, чем или равно (GLE)	T	T	T	F	F			
seq	сигнальное равно	F	F	T	F	T			2
	сигнальное неравно (SNE)	T	T	F	T	F			
ngl	не больше, чем или меньше, чем	F	F	T	T	T			3
	больше, чем или меньше, чем (GL)	T	T	F	F	F			
lt	меньше, чем	F	T	F	F	T			4
	не меньше, чем (NLT)	T	F	T	T	F			
nge	не больше, чем или равно	F	T	F	T	T			5
	больше, чем или равно (GE)	T	F	T	F	F			
le	меньше, чем или равно	F	T	T	F	T			6
	не меньше, чем или равно (NLE)	T	F	F	T	F			
ngt	не больше, чем	F	T	T	T	T			7
	больше, чем (GT)	T	F	F	F	F			

Условные обозначения: ? – неупорядочено; > - больше, чем; < - меньше, чем; = - равно.

Ограничения: Поля *fs* и *ft* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнды должны быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значения операндов также становятся НЕПРЕДСКАЗУЕМЫ.

Операция:

```
    if SNaN(ValueFPR(fs, fmt)) or SNaN(ValueFPR(ft, fmt)) or
    QNaN(ValueFPR(fs, fmt)) or QNaN(ValueFPR(ft, fmt)) then
    less ← false
    equal ← false
    unordered ← true
    if (SNaN(ValueFPR(fs,fmt)) or SNaN(ValueFPR(ft,fmt))) or
    (cond3  and  (QNaN(ValueFPR(fs,fmt))  or  QNaN(ValueFPR(ft,fmt))))
then
    SignalException(InvalidOperation)
    endif
    else
    less ← ValueFPR(fs, fmt) <fmt ValueFPR(ft, fmt)
    equal ← ValueFPR(fs, fmt) =fmt ValueFPR(ft, fmt)
    unordered ← false
    endif
    condition ← (cond2 and less) or (cond1 and equal)
    or (cond0 and unordered)
    SetFPConditionCode(cc, condition)
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation, Invalid Operation

Примечание программисту: Команды сравнения чисел с плавающей точкой, включая команды сравнения, которые получают в качестве операнда значения SNaN, порождают условие исключения Invalid Operation. Операции сравнения, которые порождают условие исключения Invalid Operation для значений QNaN, в дополнение к SNaN, позволяют создавать простую программную модель, когда значения типа NaN являются ошибками. Используя такие сравнения, в программу не нужно добавлять излишний код для проверки на QNaN, что приводит к отноше-

нию «неупорядочено». Вместо этого, программа вызывает исключение и позволяет обработчику исключений работать с ошибками тогда, когда они возникают. Например, рассмотрим сравнение, когда мы хотим определить, равны ли два числа, но для которых отношение «неупорядочено» будет ошибкой:

```
# сравнение с избыточными проверками на QNaN
c.eq.d $f2,$f4      # проверить на равенство
nop
bc1t L2             # равно
c.un.d $f2,$f4      # не равно, но, возможно, неупорядочены
bc1t ERROR          # отношение «неупорядочено» - переход на обработку ошибок
# здесь размещен код для случая «не равно»
...
# здесь размещен код для случая «равно»
L2:
# ------------------------------------------------------------
# сравнение с использованием операций, которые сигнализируют о значениях QNaN
c.seq.d $f2,$f4     # проверить на равенство
nop
bc1t L2             # равно
nop
# здесь размещен код для случая не «неупорядочено»
...
# здесь размещен код для случая «не равно»
...
# здесь размещен код для случая «равно»
L2:
```

4.8 CEIL.W.fmt, преобразование в формат с фиксированной точкой округлением в сторону +∞

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	00000	fs	fd	CEIL.W 001110

Формат: CEIL.W.S fd, fs

CEIL.W.D fd, fs

Описание: `fd ← convert_and_round(fs)`

Число в FPR *fs* в формате *fmt* преобразуется в 32-битное число с фиксированной точкой и округляется в сторону + бесконечности (режим округления 2). Результат помещается в FPR *fd*.

Когда исходное число является бесконечностью, NaN или округляется до целого числа вне диапазона -2^{31} – $2^{31}-1$, результат не может быть правильно представлен, возникает условие исключения Invalid Operation, и в управляющем регистре FCSR выставляется соответствующий бит в поле *flags*. Если в регистре FCSR выставлен соответствующий бит в поле *enables*, результат в регистр *fd* не записывается и незамедлительно возникает исключение Invalid Operation. В противном случае, в регистр *fd* помещается значение по умолчанию, $2^{31}-1$.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операции: *fs* – для типа *fmt*, *fd* – для 32-битного слова с фиксированной точкой, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМЫМ.

Операция:

```
StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Invalid Operation, Unimplemented Operation, Inexact, Overflow

4.9 CFC1, пересылка управляющего слова из сопроцессора 1

31 25	21 20	15 11	10 0
COP1 010001	CF 00010	rt	fs	0 000 0000 0000

Формат: CFC1 rt, fs

Описание: `rt ← FP_Control[fs]`

32-битное слово из управляющего регистра *fs* сопроцессора с плавающей точкой копируется в регистр общего назначения *rt*.

Ограничения: В сопроцессоре с плавающей точкой определены несколько управляющих регистров. Результат НЕПРЕДСКАЗУЕМ, если управляющего регистра *fs* не существует.

Операция:

```
if fs = 0 then
temp ← FIR
elseif fs = 25 then
temp ← 024 || FCSR31..25 || FCSR23
elseif fs = 26 then
temp ← 014 || FCSR17..12 || 05 || FCSR6..2 || 02
elseif fs = 28 then
temp ← 020 || FCSR11.7 || 04 || FCSR24 || FCSR1..0
elseif fs = 31 then
temp ← FCSR
else
temp ← UNPREDICTABLE
endif
GPR[rt] ← temp
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: нет

4.10 CTC1, пересылка управляющего слова в сопроцессор 1

31 26	25 21	20 16	15 11	10 0
COP1 010001	CT 00110	rt	fs	0 000 0000 0000

Формат: CTC1 rt, fs

Описание: `FP_Control[fs] ← rt`

Младшее слово из общего регистра *rt* копируется в управляющий регистр *fs* сопроцессора с плавающей точкой.

Ограничения: В сопроцессоре с плавающей запятой определены несколько управляющих регистров. Результат операции НЕПРЕДСКАЗУЕМ, если регистра *fs* не существует.

Операция:

```
temp ← GPR[rt]31..0
if fs = 25 then
if temp31..8 ≠ 024 then
UNPREDICTABLE
else
FCSR ← temp7..1 || FCSR24 || temp0 || FCSR22..0
endif
elseif fs = 26 then
if temp22..18 ≠ 0 then
UNPREDICTABLE
else
FCSR ← FCSR31..18 || temp17..12 || FCSR11..7 ||
temp6..2 || FCSR1..0
endif
elseif fs = 28 then
if temp22..18 ≠ 0 then
UNPREDICTABLE
else
```

```
    FCSR ← FCSR31..25 || temp2 || FCSR23..12 || temp11..7
    || FCSR6..2 || temp1..0
    endif
    elseif fs = 31 then
    if temp22..18 ≠ 0 then
    UNPREDICTABLE
    else
    FCSR ← temp
    endif
    else
    UNPREDICTABLE
    endif
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation, Invalid Operation, Division-by-zero, Inexact, Overflow, Underflow

4.11 CVT.D.fmt, преобразование в формат с плавающей точкой, тип double

31	25	20	15	10	5	0
COP1 010001	fmt	00000	fs	fd	CVT.D 100001	

Формат: CVT.D.S fd, fs

CVT.D.W fd, fs

Описание: `fd ← convert_and_round(fs)`

Число в FPR *fs* в формате *fmt* преобразуется в число с плавающей точкой типа *double* и округляется согласно текущему режиму округления, выставленному в управляющем регистре FCSR. Результат помещается в FPR *fd*. Если формат *fmt* является *single* или *word*, то результат всегда является точным.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операции: *fs* – для типа *fmt*, *fd* – для числа с плавающей точкой типа *double*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМЫМ.

Операция:

```
StoreFPR (fd, D, ConvertFmt(ValueFPR(fs, fmt), fmt, D))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Invalid Operation, Unimplemented Operation, Inexact

4.12 CVT.S.fmt, преобразование в формат с плавающей точкой, тип single

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	00000	fs	fd	CVT.S 100000

Формат: CVT.S.D fd, fs

CVT.S.W fd, fs

Описание: `fd ← convert_and_round(fs)`

Число в FPR *fs* в формате *fmt* преобразуется в число с плавающей точкой типа *single* и округляется согласно текущему режиму округления, выставленному в управляющем регистре FCSR. Результат помещается в FPR *fd*.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операции: *fs* – для типа *fmt*, *fd* – для числа с плавающей точкой типа *single*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМЫМ.

Операция:

```
StoreFPR(fd, S, ConvertFmt(ValueFPR(fs, fmt), fmt, S))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Invalid Operation, Unimplemented Operation, Inexact, Overflow, Underflow

4.13 CVT.W.fmt, преобразование в формат с фиксированной точкой

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	00000	fs	fd	CVT.W 100100

Формат: CVT.W.S fd, fs

CVT.W.D fd, fs

Описание: `fd ← convert_and_round(fs)`

Число в FPR *fs* в формате *fmt* преобразуется в 32-битное число с фиксированной точкой и округляется согласно текущему режиму округления, выставленному в управляющем регистре FCSR. Результат помещается в FPR *fd*.

Когда исходное число является бесконечностью, NaN или округляется до целого числа вне диапазона -2^{31} – $2^{31}-1$, результат не может быть правильно представлен, возникает условие исключения Invalid Operation, и в управляющем регистре FCSR выставляется соответствующий бит в поле *flags*. Если в регистре FCSR выставлен соответствующий бит в поле *enables*, результат в регистр *fd* не записывается и незамедлительно возникает исключение Invalid Operation. В противном случае, в регистр *fd* помещается значение по умолчанию, $2^{31}-1$.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операции: *fs* – для типа *fmt*, *fd* – для числа с плавающей точкой типа *single*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМЫМ.

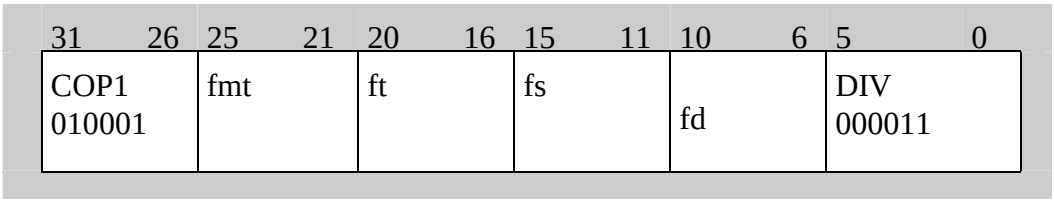
Операция:

```
StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Invalid Operation, Unimplemented Operation, Inexact, Overflow

4.14 DIV.fmt, деление



Формат: DIV.S fd, fs, ft

DIV.D fd, fs, ft

Описание: `fd ← fs / ft`

Число в FPR *fs* делится на число в FPR *ft*. Результат вычисляется с бесконечной точностью и округляется согласно текущему режиму округления, который задается в управляющем регистре FCSR. Результат помещается в FPR *fd*. Операнды и результат операции – величины формата *fmt*.

Ограничения: Поля *fs*, *ft* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнды должны быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значения операндов также становятся НЕПРЕДСКАЗУЕМЫ.

Операция:

```
StoreFPR (fd, fmt, ValueFPR(fs, fmt) / ValueFPR(ft, fmt))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Inexact, Invalid Operation, Unimplemented Operation, Division-by-zero, Overflow, Underflow

4.15 FLOOR.W.fmt, преобразование в формат с фиксированной точкой округлением в сторону -∞

31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	00000	fs	fd	FLOOR.W 001111

Формат: FLOOR.W.S fd, fs

FLOOR.W.D fd, fs

Описание: `fd ← convert_and_round(fs)`

Число в FPR *fs* в формате *fmt* преобразуется в 32-битное число с фиксированной точкой и округляется в сторону - бесконечности (режим округления 3). Результат помещается в FPR *fd*.

Когда исходное число является бесконечностью, NaN или округляется до целого числа вне диапазона -2^{31} – $2^{31}-1$, результат не может быть правильно представлен, возникает условие исключения Invalid Operation, и в управляющем регистре FCSR выставляется соответствующий бит в поле *flags*. Если в регистре FCSR выставлен соответствующий бит в поле *enables*, результат в регистр *fd* не записывается и незамедлительно возникает исключение Invalid Operation. В противном случае, в регистр *fd* помещается значение по умолчанию, $2^{31}-1$.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операции: *fs* – для типа *fmt*, *fd* – для 32-битного слова с фиксированной точкой, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМ.

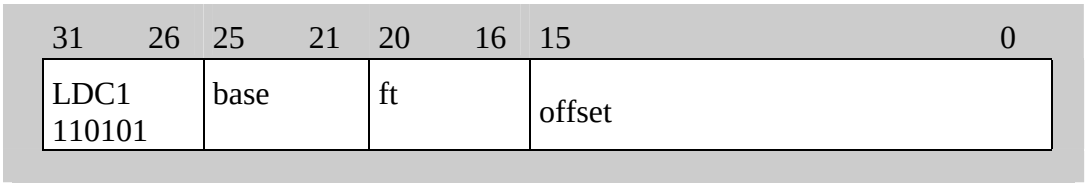
Операция:

```
StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Invalid Operation, Unimplemented Operation, Inexact, Overflow

4.16 LDC1, загрузка двойного слова



Формат: LDC1 ft, offset(base)

Описание: `ft ← memory[base+offset]`

64-битное двойное слово из ячейки памяти, определяемой выровненным исполнительным адресом, выбирается и помещается в FPR *ft*. Для формирования исполнительного адреса 16-битная знаковая *offset* прибавляется к содержимому регистра общего назначения *base*.

Ограничения: Если младшие 3 бита исполнительного адреса не равны 0 (адрес не выровнен по границе двойного слова), возникает исключение Address Error.

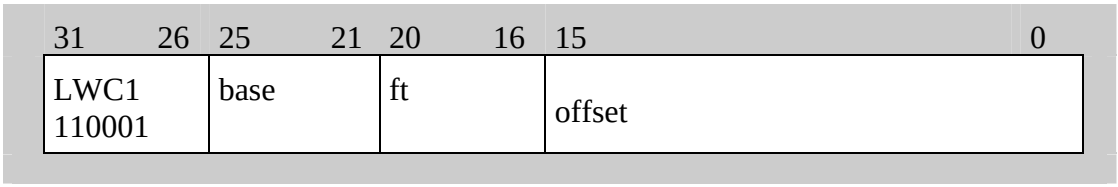
Операция:

```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr2..0 ≠ 03 then
SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, LOAD)
memdoubleword ← LoadMemory(CCA, DOUBLEWORD, pAddr, vAddr, DATA)
StoreFPR(ft, UNINTERPRETED_DOUBLEWORD, memdoubleword)
```

Исключения: Coprocessor Unusable, Reserved Instruction, TLB Refill, TLB Invalid, Address Error

Исключения в операциях с плавающей точкой: нет

4.17 LWC1, загрузка слова



Формат: LWC1 ft, offset(base)

Описание: `ft ← memory[base+offset]`

32-битное слово из ячейки памяти, определяемой выровненным исполнительным адресом, выбирается и помещается в FPR *ft*. Для формирования исполнительного адреса 16-битная знаковая *offset* прибавляется к содержимому регистра общего назначения *base*.

Ограничения: Если младшие 2 бита исполнительного адреса не равны 0 (адрес не выровнен по границе слова), возникает исключение Address Error.

Операция:

```
/* mem is aligned 64 bits from memory. Pick out correct bytes. */
vAddr ← sign_extend(offset) + GPR[base]
if vAddr1..0 ≠ 02 then
SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation (vAddr, DATA, LOAD)
memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
StoreFPR(ft, UNINTERPRETED_WORD,
memword)
```

Исключения: TLB Refill, TLB Invalid, Address Error, Reserved Instruction, Coprocessor Unusable

Исключения в операциях с плавающей точкой: нет

4.18 MFC1, пересылка слова из сопроцессора 1

31 26	25 21	20 16	15 11	10 0
COP1 010001	MF 00000	rt	fs	0 000 0000 0000

Формат: MFC1 rt, fs

Описание: `rt ← fs`

Содержимое FPR *fs* загружается в общий регистр *rt.*

Ограничения: нет

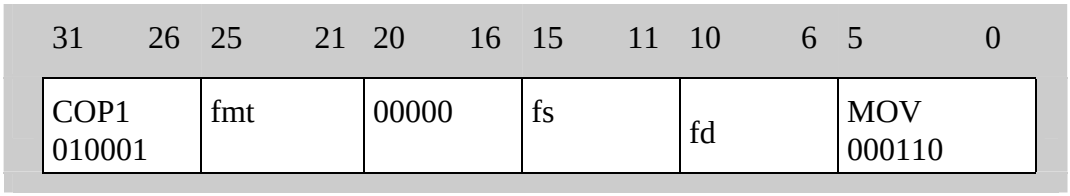
Операция:

```
data ← ValueFPR(fs, UNINTERPRETED_WORD)
GPR[rt] ← data
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: нет

4.19 MOV.fmt, пересылка



31 26	25 21	20 16	15 11	10 6	5 0
COP1 010001	fmt	00000	fs	fd	MOV 000110

Формат: MOV.S fd, fs

MOV.D fd, fs

Описание: `fd ← fs`

Число из FPR *fs* помещается в FPR *fd*. Источник и приемник – числа в формате *fmt*. Операция пересылки не является арифметической и не вызывает исключений стандарта IEEE-754.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМО.

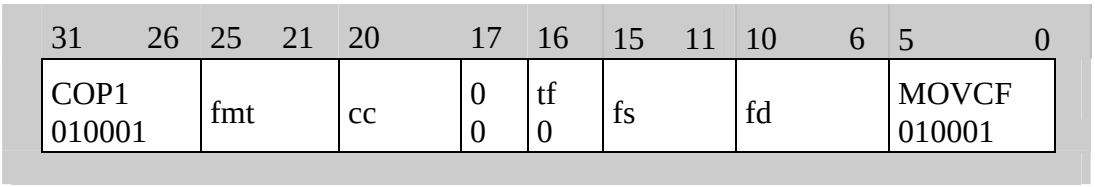
Операция:

```
StoreFPR(fd, fmt, ValueFPR(fs, fmt))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation

4.20 MOVF.fmt, условная пересылка по соотношению «ЛОЖЬ»



31 26	25 21	20 17	16	15 11	10 6	5 0
COP1 010001	fmt	cc	0 0	tf 0	fs	fd	MOVCF 010001

Формат: MOVF.S fd, fs, cc

MOVF.D fd, fs, cc

Описание: `if cc = 0 then fd ← fs`

Если код условия, определяемый полем *cc*, равен 0, то число из FPR *fs* помещается в FPR *fd*. Источник и приемник – числа в формате *fmt*.

Если код условия неравен 0, то пересылка не происходит, и FPR *fd* сохраняет свое предыдущее значение в формате *fmt*. Если регистр *fd* не содержал никакого числа в формате *fmt* или ранее не использованного числа из команд загрузки или пересылки, которое можно интерпретировать в формате *fmt*, то значение регистра *fd* становится НЕПРЕДСКАЗУЕМЫМ.

Операция пересылки не является арифметической и не вызывает исключений стандарта IEEE-754.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМО.

Операция:

```
if FPConditionCode(cc) = 0 then
StoreFPR(fd, fmt, ValueFPR(fs, fmt))
else
StoreFPR(fd, fmt, ValueFPR(fd, fmt))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation

4.21 MOVN.fmt, условная пересылка при неравенстве нулю

31	25	20	15	10	5	0
COP1 010001	fmt	rt	fs	fd	MOVN 010011	

Формат: MOVN.S fd, fs, rt

MOVN.D fd, fs, rt

Описание: `if rt ≠ 0 then fd ← fs`

Если содержимое общего регистра *rt* не равно 0, то число из FPR *fs* помещается в FPR *fd*. Источник и приемник – числа в формате *fmt*.

Если общий регистр *rt* содержит 0, то пересылка не происходит, и FPR *fd* сохраняет свое предыдущее значение в формате *fmt*. Если регистр *fd* не содержал никакого числа в формате *fmt* или ранее не использованного числа из команд загрузки или пересылки, которое можно интерпретировать в формате *fmt*, то значение регистра *fd* становится НЕПРЕДСКАЗУЕМЫМ.

Операция пересылки не является арифметической и не вызывает исключений стандарта IEEE-754.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМО.

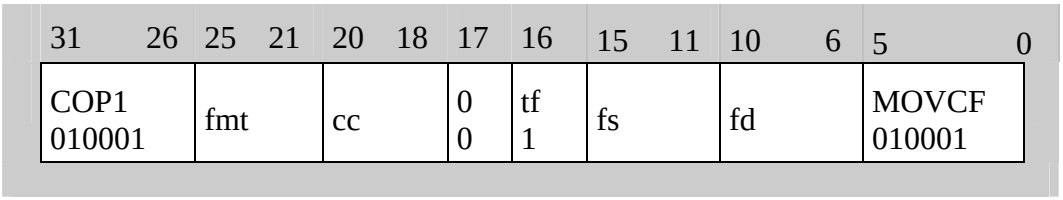
Операция:

```
if GPR[rt] ≠ 0 then
StoreFPR(fd, fmt, ValueFPR(fs, fmt))
else
StoreFPR(fd, fmt, ValueFPR(fd, fmt))
endif
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation

4.22 MOVT.fmt, условная пересылка по соотношению «ИСТИНА»



31 26	25 21	20 18	17 16	15 11	10 6	5 0
COP1 010001	fmt	cc	0 0	tf 1	fs	fd	MOVCF 010001

Формат: MOVT.S fd, fs, cc

MOVT.D fd, fs, cc

Описание: `if cc = 1 then fd ← fs`

Если код условия, определяемый полем *cc*, равен 1, то число из FPR *fs* помещается в FPR *fd*. Источник и приемник – числа в формате *fmt*.

Если код условия неравен 1, то пересылка не происходит, и FPR *fd* сохраняет свое предыдущее значение в формате *fmt*. Если регистр *fd* не содержал никакого числа в формате *fmt* или ранее не использованного числа из команд загрузки или пересылки, которое можно интерпретировать в формате *fmt*, то значение регистра *fd* становится НЕПРЕДСКАЗУЕМЫМ.

Операция пересылки не является арифметической и не вызывает исключений стандарта IEEE-754.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМО.

Операция:

```
if FPConditionCode(cc) = 0 then
StoreFPR(fd, fmt, ValueFPR(fs, fmt))
else
StoreFPR(fd, fmt, ValueFPR(fd, fmt))
endif
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation

4.23 MOVZ.fmt, условная пересылка при равенстве нулю

31	25	20	15	10	5	0
COP1 010001	fmt	rt	fs	fd	MOVZ 010010	

Формат: MOVZ.S fd, fs, rt

MOVZ.D fd, fs, rt

Описание: `if rt = 0 then fd ← fs`

Если содержимое общего регистра *rt* не равно 0, то число из FPR *fs* помещается в FPR *fd*. Источник и приемник – числа в формате *fmt*.

Если общий регистр *rt* содержит 0, то пересылка не происходит, и FPR *fd* сохраняет свое предыдущее значение в формате *fmt*. Если регистр *fd* не содержал никакого числа в формате *fmt* или ранее не использованного числа из команд загрузки или пересылки, которое можно интерпретировать в формате *fmt*, то значение регистра *fd* становится НЕПРЕДСКАЗУЕМЫМ.

Операция пересылки не является арифметической и не вызывает исключений стандарта IEEE-754.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМО.

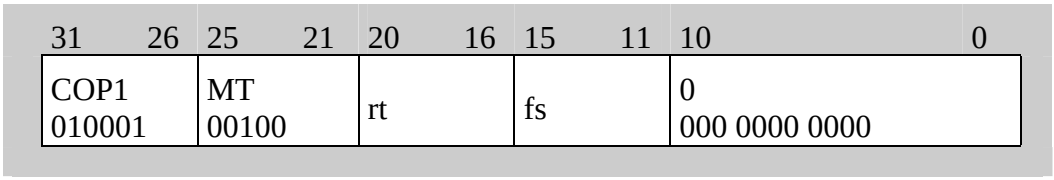
Операция:

```
if GPR[rt] = 0 then
StoreFPR(fd, fmt, ValueFPR(fs, fmt))
else
StoreFPR(fd, fmt, ValueFPR(fd, fmt))
endif
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation

4.24 MTC1, пересылка слова в сопроцессор 1



31 26	25 21	20 16	15 11	10 0
COP1 010001	MT 00100	rt	fs	0 000 0000 0000

Формат: MTC1 rt, fs

Описание: `fs ← rt`

Слово из общего регистра *rt* помещается в младшую часть (младшие 32 бита) FPR *fs*.

Ограничения: нет

Операция:

```
data ← GPR[rt]31..0
StoreFPR(fs, UNINTERPRETED_WORD, data)
```

Исключения: Coprocessor Unusable

Исключения в операциях с плавающей точкой: нет

4.25 MUL.fmt, умножение

31	25	20	15	10	5	0
COP1 010001	fmt	ft	fs	fd	MUL 000010	

Формат: MUL.S fd, fs, ft

MUL.D fd, fs, ft

Описание: `fd ← fs × ft`

Число в FPR *fs* умножается на число в FPR *ft*. Результат вычисляется с бесконечной точностью и округляется согласно текущему режиму округления, который задается в управляющем регистре FCSR. Результат помещается в FPR *fd*. Операнды и результат операции – величины формата *fmt*.

Ограничения: Поля *fs*, *ft* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнды должны быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значения операндов также становятся НЕПРЕДСКАЗУЕМЫ.

Операция:

```
StoreFPR (fd, fmt, ValueFPR(fs, fmt) ×fmt ValueFPR(ft, fmt))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Inexact, Unimplemented Operation, Invalid Operation, Overflow, Underflow

4.26 NEG.fmt, отрицание

31	25	20	15	10	5	0
COP1 010001	fmt	00000	fs	fd	NEG 000111	

Формат: NEG.S fd, fs

NEG.D fd, fs

Описание: `fd ← –fs`

Число из FPR *fs* берется с противоположным знаком и помещается в FPR *fd*. Операнды и результат операции – величины формата *fmt*. Операция является арифметической, величины NaN вызывают исключение Invalid Operation.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМО.

Операция:

```
StoreFPR(fd, fmt, Negate(ValueFPR(fs, fmt)))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Unimplemented Operation, Invalid Operation

4.27 ROUND.W.fmt, преобразование в формат с фиксированной точкой округлением до ближайшего или четного

31	25	20	15	10	5 0
COP1 010001	fmt	00000	fs	fd	ROUND.W 001100

Формат: ROUND.W.S fd, fs

ROUND.W.D fd, fs

Описание: `fd ← convert_and_round(fs)`

Число в FPR *fs* в формате *fmt* преобразуется в 32-битное число с фиксированной точкой и округляется до ближайшего четного (режим округления 0). Результат помещается в FPR *fd*.

Когда исходное число является бесконечностью, NaN или округляется до целого числа вне диапазона -2^{31} – $2^{31}-1$, результат не может быть правильно представлен, возникает условие исключения Invalid Operation, и в управляющем регистре FCSR выставляется соответствующий бит в поле *flags*. Если в регистре FCSR выставлен соответствующий бит в поле *enables*, результат в регистр *fd* не записывается и незамедлительно возникает исключение Invalid Operation. В противном случае, в регистр *fd* помещается значение по умолчанию, $2^{31}-1$.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операции: *fs* – для типа *fmt*, *fd* – для 32-битного слова с фиксированной точкой, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМ.

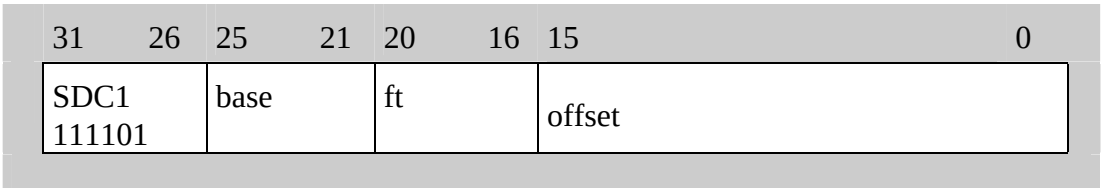
Операция:

```
StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Invalid Operation, Unimplemented Operation, Inexact, Overflow

4.28 SDC1, сохранение двойного слова



Формат: SDC1 ft, offset(base)

Описание: `memory[base+offset] ← ft`

64-битное двойное слово из FPR *ft* сохраняется в ячейке памяти, которая определяется выровненным исполнительным адресом. Для формирования исполнительного адреса 16-битная знаковая *offset* прибавляется к содержимому общего регистра *base*.

Ограничения: Если младшие 3 бита исполнительного адреса не равны 0 (адрес не выровнен по границе двойного слова), возникает исключение Address Error.

Операция:

```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr2..0 ≠ 03 then
SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
datadoubleword ← ValueFPR(ft, UNINTERPRETED_DOUBLEWORD)
StoreMemory(CCA, DOUBLEWORD, datadoubleword, pAddr, vAddr, DATA)
```

Исключения: Coprocessor Unusable, Reserved Instruction, TLB Refill, TLB Invalid, TLB Modified, Address Error

Исключения в операциях с плавающей точкой: нет

4.29 SQRT.fmt, извлечение корня

31	25	20	15	10	5 0
COP1 010001	fmt	00000	fs	fd	SQRT 000100

Формат: SQRT.S fd, fs

SQRT.D fd, fs

Описание: `fd ← SQRT(fs)`

Квадратный корень из число в FPR *fs* вычисляется с бесконечной точностью и округляется согласно текущему режиму округления, который задается в управляющем регистре FCSR. Результат помещается в FPR *fd*. Операнд и результат операции – величины формата *fmt.* Если операнд равен -0, то результат операции – также -0.

Ограничения: Если число в FPR *fs* меньше 0, то возникает исключение Invalid Operation.

Поля *fs* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМО.

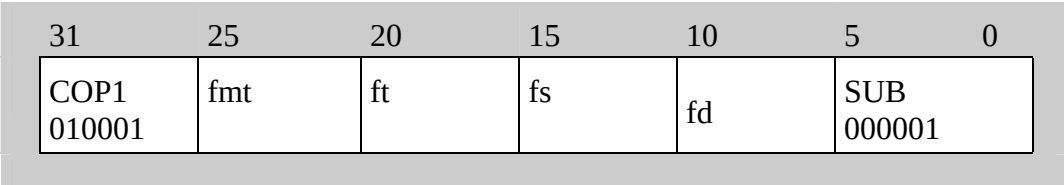
Операция:

```
StoreFPR(fd, fmt, SquareRoot(ValueFPR(fs, fmt)))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Invalid Operation, Inexact, Unimplemented Operation

4.30 SUB.fmt, вычитание



31	25	20	15	10	5	0
COP1 010001	fmt	ft	fs	fd	SUB 000001	

Формат: SUB.S fd, fs, ft

SUB.D fd, fs, ft

Описание: `fd ← fs - ft`

Значение FPR *ft* вычитается из значения FPR *fs*. Результат вычисляется с бесконечной точностью и округляется согласно текущему режиму округления, который задается в управляющем регистре FCSR. Результат помещается в FPR *fd*. Операнды и результат операции – величины формата *fmt*.

Ограничения: Поля *fs*, *ft* и *fd* должны определять регистры FPR, подходящие для операндов типа *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнды должны быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значения операндов также становятся НЕПРЕДСКАЗУЕМЫ.

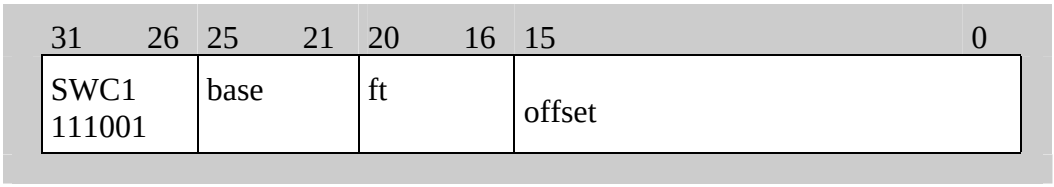
Операция:

```
StoreFPR (fd, fmt, ValueFPR(fs, fmt) –fmt ValueFPR(ft, fmt))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Inexact, Overflow, Underflow, Invalid Op, Unimplemented Op

4.31 SWC1, сохранение слова



Формат: SWC1 ft, offset(base)

Описание: `memory[base+offset] ← ft`

32-битное слово из FPR *ft* сохраняется в ячейке памяти, которая определяется выровненным исполнительным адресом. Для формирования исполнительного адреса 16-битная знаковая *offset* прибавляется к содержимому общего регистра *base*.

Ограничения: Если младшие 2 бита исполнительного адреса не равны 0 (адрес не выровнен по границе слова), возникает исключение Address Error.

Операция:

```
vAddr ← sign_extend(offset) + GPR[base]
if vAddr1..0 ≠ 03 then
SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
dataword ← ValueFPR(ft, UNINTERPRETED_WORD)
StoreMemory(CCA, WORD, dataword, pAddr, vAddr, DATA)
```

Исключения: Coprocessor Unusable, Reserved Instruction, TLB Refill, TLB Invalid, TLB Modified, Address Error

Исключения в операциях с плавающей точкой: нет

4.32 TRUNC.W.fmt, преобразование в формат с фиксированной точкой округлением в сторону нуля

31	25	20	15	10	5	0
COP1 010001	fmt	00000	fs	fd	TRUNC.W 001101	

Формат: TRUNC.W.S fd, fs

TRUNC.W.D fd, fs

Описание: `fd ← convert_and_round(fs)`

Число в FPR *fs* в формате *fmt* преобразуется в 32-битное число с фиксированной точкой и округляется в сторону 0 (режим округления 1). Результат помещается в FPR *fd*.

Когда исходное число является бесконечностью, NaN или округляется до целого числа вне диапазона -2^{31} – $2^{31}-1$, результат не может быть правильно представлен, возникает условие исключения Invalid Operation, и в управляющем регистре FCSR выставляется соответствующий бит в поле *flags*. Если в регистре FCSR выставлен соответствующий бит в поле *enables*, результат в регистр *fd* не записывается и незамедлительно возникает исключение Invalid Operation. В противном случае, в регистр *fd* помещается значение по умолчанию, $2^{31}-1$.

Ограничения: Поля *fs* и *fd* должны определять регистры FPR, подходящие для операции: *fs* – для типа *fmt*, *fd* – для 32-битного слова с фиксированной точкой, иначе, результат НЕПРЕДСКАЗУЕМ.

Операнд должен быть в формате *fmt*, иначе, результат НЕПРЕДСКАЗУЕМ, и значение операнда также становится НЕПРЕДСКАЗУЕМ.

Операция:

```
StoreFPR(fd, W, ConvertFmt(ValueFPR(fs, fmt), fmt, W))
```

Исключения: Coprocessor Unusable, Reserved Instruction

Исключения в операциях с плавающей точкой: Invalid Operation, Unimplemented Operation, Inexact, Overflow