

Ведение

Данное руководство содержит информацию для разработчиков системного ПО и приложений. Руководство предоставляет полное описание модели программирования процессоров Cortex[®]-M4 серий STM32F3 и STM32F4 Cortex[®]-M4, набора инструкций и основных периферийных устройств.

Процессоры Cortex[®]-M4 серий STM32F3 и STM32F4 представляют собой высокопроизводительные 32 битные процессоры разработанные для рынка микроконтроллеров. Они предлагают значимые преимущества для разработчиков, такие как:

- Выдающаяся производительность совмещённая с быстрой обработкой прерываний.
- Улучшенная система отладки.
- Эффективное ядро, система и память.
- Низкое потребление энергии.
- Безопасность платформы.

Справочные документы

- Документация на STM32F3 и STM32F4
- STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx и STM32F43xxx
- STM32F401xB/C и STM32F401xD/E
- STM32F37xx
- STM32F303xB/C, STM32F303x6/8, STM32F328x8 и STM32F358xC

Перечисленные документы доступны на www.st.com.

Данное руководство применимо к продуктам перечисленным в [Таблице 1](#).

Таблица.1

Тип	Обозначение
Микроконтроллеры	STM32F3 STM32F4

Содержание

1	About this document	12
1.1	Typographical conventions	12
1.2	List of abbreviations for registers	12
1.3	About the STM32 Cortex-M4 processor and core peripherals	13
1.3.1	System level interface	14
1.3.2	Integrated configurable debug	14
1.3.3	Cortex-M4 processor features and benefits summary	14
1.3.4	Cortex-M4 core peripherals	15
2	The Cortex-M4 processor	16
2.1	Programmers model	16
2.1.1	Processor mode and privilege levels for software execution	16
2.1.2	Stacks	16
2.1.3	Core registers	17
2.1.4	Exceptions and interrupts	25
2.1.5	Data types	25
2.1.6	The Cortex microcontroller software interface standard (CMSIS)	25
2.2	Memory model	27
2.2.1	Memory regions, types and attributes	28
2.2.2	Memory system ordering of memory accesses	28
2.2.3	Behavior of memory accesses	29
2.2.4	Software ordering of memory accesses	30
2.2.5	Bit-banding	31
2.2.6	Memory endianness	33
2.2.7	Synchronization primitives	33
2.2.8	Programming hints for the synchronization primitives	35
2.3	Exception model	36
2.3.1	Exception states	36
2.3.2	Exception types	36
2.3.3	Exception handlers	38
2.3.4	Vector table	39
2.3.5	Exception priorities	40
2.3.6	Interrupt priority grouping	40
2.3.7	Exception entry and return	41

2.4	Fault handling	43
2.4.1	Fault types	44
2.4.2	Fault escalation and hard faults	45
2.4.3	Fault status registers and fault address registers	46
2.4.4	Lockup	46
2.5	Power management	46
2.5.1	Entering sleep mode	47
2.5.2	Wakeup from sleep mode	47
2.5.3	External event input / extended interrupt and event input	48
2.5.4	Power management programming hints	48
3	The STM32 Cortex-M4 instruction set	49
3.1	Instruction set summary	49
3.2	CMSIS intrinsic functions	57
3.3	About the instruction descriptions	59
3.3.1	Operands	59
3.3.2	Restrictions when using PC or SP	59
3.3.3	Flexible second operand	59
3.3.4	Shift operations	61
3.3.5	Address alignment	64
3.3.6	PC-relative expressions	64
3.3.7	Conditional execution	64
3.3.8	Instruction width selection	67
3.4	Memory access instructions	68
3.4.1	ADR	69
3.4.2	LDR and STR, immediate offset	70
3.4.3	LDR and STR, register offset	72
3.4.4	LDR and STR, unprivileged	73
3.4.5	LDR, PC-relative	74
3.4.6	LDM and STM	75
3.4.7	PUSH and POP	77
3.4.8	LDREX and STREX	78
3.4.9	CLREX	79
3.5	General data processing instructions	80
3.5.1	ADD, ADC, SUB, SBC, and RSB	82
3.5.2	AND, ORR, EOR, BIC, and ORN	84

3.5.3	ASR, LSL, LSR, ROR, and RRX	85
3.5.4	CLZ	86
3.5.5	CMP and CMN	87
3.5.6	MOV and MVN	88
3.5.7	MOVT	90
3.5.8	REV, REV16, REVSH, and RBIT	91
3.5.9	SADD16 and SADD8	92
3.5.10	SHADD16 and SHADD8	93
3.5.11	SHASX and SHSAX	94
3.5.12	SHSUB16 and SHSUB8	95
3.5.13	SSUB16 and SSUB8	96
3.5.14	SASX and SSAX	97
3.5.15	TST and TEQ	98
3.5.16	UADD16 and UADD8	99
3.5.17	UASX and USAX	100
3.5.18	UHADD16 and UHADD8	101
3.5.19	UHASX and UHSAX	102
3.5.20	UHSUB16 and UHSUB8	103
3.5.21	SEL	104
3.5.22	USAD8	105
3.5.23	USADA8	106
3.5.24	USUB16 and USUB8	107
3.6	Multiply and divide instructions	108
3.6.1	MUL, MLA, and MLS	109
3.6.2	UMULL, UMAAL and UMLAL	110
3.6.3	SMLA and SMLAW	111
3.6.4	SMLAD	113
3.6.5	SMLAL and SMLALD	114
3.6.6	SMLSD and SMLSLD	116
3.6.7	SMMLA and SMMLS	118
3.6.8	SMMUL	119
3.6.9	SMUAD and SMUSD	120
3.6.10	SMUL and SMULW	121
3.6.11	UMULL, UMLAL, SMULL, and SMLAL	122
3.6.12	SDIV and UDIV	123
3.7	Saturating instructions	124
3.7.1	SSAT and USAT	125

3.7.2	SSAT16 and USAT16	126
3.7.3	QADD and QSUB	127
3.7.4	QASX and QSAX	128
3.7.5	QDADD and QDSUB	129
3.7.6	UQASX and UQSAX	130
3.7.7	UQADD and UQSUB	131
3.8	Packing and unpacking instructions	132
3.8.1	PKHBT and PKHTB	133
3.8.2	SXT and UXT	134
3.8.3	SXTA and UXTA	135
3.9	Bitfield instructions	136
3.9.1	BFC and BFI	137
3.9.2	SBFX and UBFX	138
3.9.3	SXT and UXT	139
3.9.4	Branch and control instructions	140
3.9.5	B, BL, BX, and BLX	141
3.9.6	CBZ and CBNZ	143
3.9.7	IT	144
3.9.8	TBB and TBH	146
3.10	Floating-point instructions	148
3.10.1	VABS	149
3.10.2	VADD	150
3.10.3	VCMP, VCMPE	150
3.10.4	VCVT, VCVTR between floating-point and integer	151
3.10.5	VCVT between floating-point and fixed-point	152
3.10.6	VCVTB, VCVTT	153
3.10.7	VDIV	154
3.10.8	VFMA, VFMS	154
3.10.9	VFNMA, VFNMS	155
3.10.10	VLDM	156
3.10.11	VLDR	157
3.10.12	VLMA, VLMS	158
3.10.13	VMOV immediate	158
3.10.14	VMOV register	159
3.10.15	VMOV scalar to ARM core register	159
3.10.16	VMOV ARM core register to single precision	160
3.10.17	VMOV two ARM core registers to two single precision	161

3.10.18	VMOV ARM Core register to scalar	162
3.10.19	VMRS	162
3.10.20	VMSR	163
3.10.21	VMUL	163
3.10.22	VNEG	164
3.10.23	VNMLA, VNMLS, VNMUL	165
3.10.24	VPOP	166
3.10.25	VPUSH	166
3.10.26	VSQRT	167
3.10.27	VSTM	167
3.10.28	VSTR	168
3.10.29	VSUB	169
3.11	Miscellaneous instructions	170
3.11.1	BKPT	170
3.11.2	CPS	171
3.11.3	DMB	172
3.11.4	DSB	172
3.11.5	ISB	173
3.11.6	MRS	173
3.11.7	MSR	174
3.11.8	NOP	175
3.11.9	SEV	175
3.11.10	SVC	176
3.11.11	WFE	176
3.11.12	WFI	177
4	Core peripherals	178
4.1	About the STM32 Cortex-M4 core peripherals	178
4.2	Memory protection unit (MPU)	178
4.2.1	MPU access permission attributes	180
4.2.2	MPU mismatch	181
4.2.3	Updating an MPU region	181
4.2.4	MPU design hints and tips	184
4.2.5	MPU type register (MPU_TYPER)	185
4.2.6	MPU control register (MPU_CTRL)	186
4.2.7	MPU region number register (MPU_RNR)	187
4.2.8	MPU region base address register (MPU_RBAR)	188

4.2.9	MPU region attribute and size register (MPU_RASR)	189
4.2.10	MPU register map	191
4.3	Nested vectored interrupt controller (NVIC)	193
4.3.1	Accessing the Cortex-M4 NVIC registers using CMSIS	194
4.3.2	Interrupt set-enable registers (NVIC_ISERx)	195
4.3.3	Interrupt clear-enable registers (NVIC_ICERx)	196
4.3.4	Interrupt set-pending registers (NVIC_ISPRx)	197
4.3.5	Interrupt clear-pending registers (NVIC_ICPRx)	198
4.3.6	Interrupt active bit registers (NVIC_IABRx)	199
4.3.7	Interrupt priority registers (NVIC_IPRx)	200
4.3.8	Software trigger interrupt register (NVIC_STIR)	201
4.3.9	Level-sensitive and pulse interrupts	202
4.3.10	NVIC design hints and tips	203
4.3.11	NVIC register map	204
4.4	System control block (SCB)	206
4.4.1	Auxiliary control register (ACTLR)	207
4.4.2	CPUID base register (CPUID)	208
4.4.3	Interrupt control and state register (ICSR)	210
4.4.4	Vector table offset register (VTOR)	212
4.4.5	Application interrupt and reset control register (AIRCR)	212
4.4.6	System control register (SCR)	214
4.4.7	Configuration and control register (CCR)	215
4.4.8	System handler priority registers (SHPRx)	217
4.4.9	System handler control and state register (SHCSR)	219
4.4.10	Configurable fault status register (CFSR; UFSR+BFSR+MMFSR) . . .	221
4.4.11	Usage fault status register (UFSR)	222
4.4.12	Bus fault status register (BFSR)	223
4.4.13	Memory management fault address register (MMFSR)	224
4.4.14	Hard fault status register (HFSR)	225
4.4.15	Memory management fault address register (MMFAR)	226
4.4.16	Bus fault address register (BFAR)	226
4.4.17	Auxiliary fault status register (AFSR)	227
4.4.18	System control block design hints and tips	227
4.4.19	SCB register map	228
4.5	SysTick timer (STK)	230
4.5.1	SysTick control and status register (STK_CTRL)	231
4.5.2	SysTick reload value register (STK_LOAD)	232

4.5.3	SysTick current value register (STK_VAL)	233
4.5.4	SysTick calibration value register (STK_CALIB)	234
4.5.5	SysTick design hints and tips	234
4.5.6	SysTick register map	235
4.6	Floating point unit (FPU)	236
4.6.1	Coprocessor access control register (CPACR)	237
4.6.2	Floating-point context control register (FPCCR)	237
4.6.3	Floating-point context address register (FPCAR)	239
4.6.4	Floating-point status control register (FPSCR)	239
4.6.5	Floating-point default status control register (FPDSCR)	241
4.6.6	Enabling the FPU	241
4.6.7	Enabling and clearing FPU exception interrupts	241
5	Revision history	244

List of tables

Table 1.	Applicable products	1
Table 2.	Summary of processor mode, execution privilege level, and stack usage	17
Table 3.	Core register set summary	17
Table 4.	PSR register combinations	19
Table 5.	APSR bit definitions	20
Table 6.	IPSR bit definitions	21
Table 7.	EPSR bit definitions	22
Table 8.	PRIMASK register bit definitions	23
Table 9.	FAULTMASK register bit definitions	23
Table 10.	BASEPRI register bit assignments	24
Table 11.	CONTROL register bit definitions	24
Table 12.	Ordering of memory accesses	28
Table 13.	Memory access behavior	29
Table 14.	SRAM memory bit-banding regions	31
Table 15.	Peripheral memory bit-banding regions	31
Table 16.	CMSIS functions for exclusive access instructions	35
Table 17.	Properties of the different exception types	37
Table 18.	Exception return behavior	43
Table 19.	Faults	44
Table 20.	Fault status and fault address registers	46
Table 21.	Cortex-M4 instructions	49
Table 22.	CMSIS intrinsic functions to generate some Cortex-M4 instructions	58
Table 23.	CMSIS intrinsic functions to access the special registers	58
Table 24.	Condition code suffixes	66
Table 25.	Memory access instructions	68
Table 26.	Immediate, pre-indexed and post-indexed offset ranges	71
Table 27.	label-PC offset ranges	74
Table 28.	Data processing instructions	80
Table 29.	Multiply and divide instructions	108
Table 30.	Saturating instructions	124
Table 31.	Packing and unpacking instructions	132
Table 32.	Instructions that operate on adjacent sets of bits	136
Table 33.	Branch and control instructions	140
Table 34.	Branch ranges	141
Table 35.	Floating-point instructions	148
Table 36.	Miscellaneous instructions	170
Table 37.	STM32 core peripheral register regions	178
Table 38.	Memory attributes summary	179
Table 39.	TEX, C, B, and S encoding	180
Table 40.	Cache policy for memory attribute encoding	180
Table 41.	AP encoding	181
Table 42.	Memory region attributes for STM32	184
Table 43.	Example SIZE field values	190
Table 44.	MPU register map and reset values	191
Table 45.	NVIC register summary	193
Table 46.	CMSIS access NVIC functions	194
Table 47.	IPR bit assignments	200
Table 48.	CMSIS functions for NVIC control	203

Table 49.	NVIC register map and reset values	204
Table 50.	Summary of the system control block registers	206
Table 51.	Priority grouping	213
Table 52.	System fault handler priority fields	217
Table 53.	SCB register map and reset values	228
Table 54.	System timer registers summary	230
Table 55.	SysTick register map and reset values	235
Table 56.	Cortex-M4F floating-point system registers	236
Table 57.	Effect of a Floating-point comparison on the condition flags	240
Table 58.	Document revision history	244

List of figures

Figure 1.	STM32 Cortex-M4 implementation	13
Figure 2.	Processor core registers	17
Figure 3.	APSR, IPSR and EPSR bit assignments	19
Figure 4.	PSR bit assignments	19
Figure 5.	PRIMASK bit assignments	23
Figure 6.	FAULTMASK bit assignments	23
Figure 7.	BASEPRI bit assignments	24
Figure 8.	Memory map	27
Figure 9.	Bit-band mapping	32
Figure 10.	Little-endian example	33
Figure 11.	Vector table	39
Figure 12.	Cortex-M4 stack frame layout	42
Figure 13.	ASR#3	61
Figure 14.	LSR#3	62
Figure 15.	LSL#3	62
Figure 16.	ROR #3	63
Figure 17.	RRX #3	63
Figure 18.	Subregion example	183
Figure 19.	NVIC_IPRx register mapping	200
Figure 20.	CFSR subregisters	221

1 Информация о руководстве

Данное руководство содержит информацию необходимую для разработки приложений и системного ПО. Он не несёт информации об операциях, компонентах и особенностях отладки.

Данный материал предназначен для разработчиков ПО и инженеров, а также для тех, кто не имеет опыта в работе с продукцией ARM.

1.1 Условные обозначения

Условные обозначения используемые в руководстве

<i>Курсив</i>	Означает важные замечания, вводит специальную терминологию, обозначает внутренние перекрестные ссылки и цитаты.
< и >	Заменимые условия для языка assembler, когда они возникают в коде или его фрагменте. Например: LDRSB<cond> <Rt>, [<Rn>, #<offset>]
Жирный	Означает основные элементы интерфейса, такие как, названия меню. Указывает имена сигналов.
Моноширинный	Означает текст который вы можете ввести с клавиатуры, например команды, имена файлов и программ, и исходный код.
Моноширинный	Означает разрешенное сокращение для команды или опции. Вы можете ввести подчеркнутый текст вместо полного имени команды или опции.
<i>Курсивный шрифт</i>	Обозначает аргументы в моноширинном тексте, в котром аргумент должен быть изменен на конкретное значение.
Жирный шрифт	Означает ключевые слова языка, при использовании вне кода в примере

1.2 Список сокращений для регистров

Следующие сокращения используются в описании регистров

чтение/запись (rw)	В данные биты можно записать, либо считать ПО
только чтение(r)	С данных битов можно только считать ПО
только запись (w)	ПО может быть только записано в данный бит Чтение возвращает значение сброса
чтение/очистка (rc_w1)	ПО может читать, а также очистить этот бит записав 1. Запись 0 никак не влияет на значение бита
чтение/очистка (rc_w0)	ПО может как считать так и записать информацию на бит записав 0 Запись 1 не влияет на значение бита
переключение (t)	ПО может переключить этот бит только записью 1
зарезервирован (Res.)	Зарезервированный бит, должен иметь значение сброса

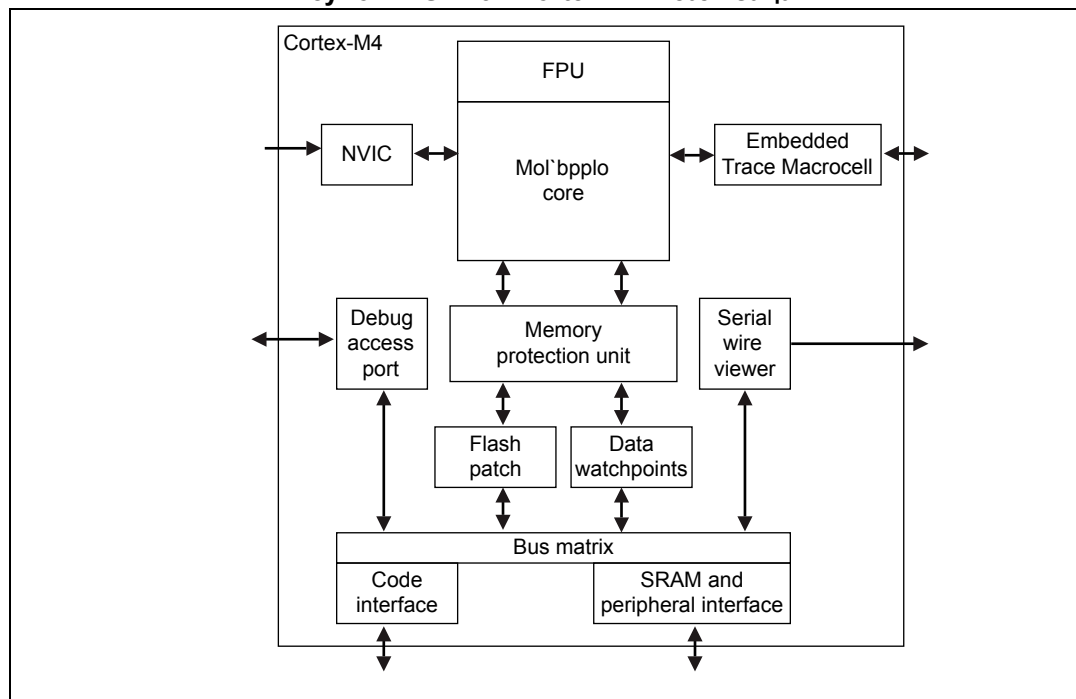
1.3 О процессоре STM32 Cortex-M4 и его основной периферии

Высокопроизводительные 32-битные процессоры Cortex-M4 разработаны специально для рынка микроконтроллера и обладают следующими преимуществами.

- выдающаяся производительность совмещенная с быстрой обработкой прерываний.
- расширенная системная отладка с возможностями трассировки
- эффективное ядро процессора, система и память
- сверхнизкая потребляемая мощность с интегрированными режимами ожидания
- безопасность платформы, с интегрированным модулем защиты памяти (MPU).

Процессор Cortex-M4 основан на высокоэффективном ядре на 3-этапной конвейерной Гарвардской архитектурой, что делает его идеальным выбором для встраиваемых приложений. Процессор обеспечивает исключительную эффективность питания через эффективную систему команд и экстенсивно оптимизированную архитектуру, обеспечивая высокопроизводительные аппаратные средства обработки включая IEEE754-совместимую одинарную точность вычисления с плавающей точкой, диапазон единственного цикла и умножения SIMD.

Рисунок 1. STM32 Cortex-M4 Реализация



Чтобы упростить разработку чувствительных к стоимости устройств, процессор Cortex-M4 реализует системные компоненты тесно связаны, чтобы уменьшить габариты процессора. Процессор Cortex-M4 реализует версию системы команд Thumb® на основе Thumb-2 технологии, гарантируя высокую плотность кода и уменьшенные требования программы к памяти. Система команд ядра -M4 обеспечивает исключительную производительность, ожидаемую от современной 32-разрядной архитектуры с высокой плотностью кода.

Процессор Cortex-M4 использует конфигурируемый встроенный контроллер прерываний (NVIC), чтобы обеспечить ведущую в отрасли производительность прерывания. NVIC включает немаскируемое прерывание (NMI) и обеспечивает до 256 уровней приоритета прерываний. Тесная интеграция ядра процессора и

NVIC обеспечивает быстрое выполнение процедур обработки прерывания (ISRs), существенно уменьшая задержку прерывания. Это достигнуто посредством аппаратной укладки регистров. Обработчики прерываний не требуют никаких ассемблерных заглушек, удаляя любой код сверху от ISRs. Объединяющая в цепочку оптимизация также значительно уменьшает издержки при переключении от одного ISR до другого.

Чтобы оптимизировать маломощные проекты, в NVIC интегрированы режимы ожидания, которые включают функцию глубокого сна, которая позволяет STM32 ввести режим STOP или STDBY.

1.3.1 Системный интерфейс

Процессор Cortex-M4 поддерживает множественные интерфейсы, используя технологию AMBA®, чтобы обеспечить высокую скорость, и доступ к памяти с минимальной задержкой.

Процессор Cortex-M4 содержит модуль защиты памяти (MPU), который обеспечивает мелко модульное управление памятью, позволяя приложениям использовать многократные уровни полномочий, отделяясь и защищая код, данные и складываться по условию задачи. Такие требования критически важны во многих встраиваемых приложениях.

1.3.2 Отладка

Процессор Cortex-M4 реализует полностью аппаратную отладку. Это обеспечивает высокую системную видимость процессора и памяти как через традиционный порт JTAG так и через 2-контактный порт Serial Wire Debug (SWD),

Для системной трассировки процессор использует (ITM) наряду с контрольными точками данных и профильным модулем. Чтобы включить простое и экономически эффективное профилирование системных событий, они генерируют (SWV) может экспортировать поток сгенерированных программным обеспечением сообщений.

Дополнительная (ETM) обеспечивает непревзойденную скорость получения трассировки инструкции в области, намного меньшей, чем традиционные модули трассировки.

1.3.3 Список особенностей и преимуществ процессора Cortex-M4

- Тесная интеграция периферийных устройств уменьшает размеры и затраты на разработку
- Объединение системы команд обеспечивают высокую плотность кода.
- IEEE754-совместимый FPU одинарной точности реализованный во всех микроконтроллерах STM32F4xxx и STM32F3xxx Cortex-M4
- Оптимизация компонентов системы питания.
- Встроенные режимы ожидания для низкого потребления энергии.
- Быстрое выполнение кода позволяет использование меньших частот
- Разделение аппаратных средств и быстрый множитель
- Детерминированная, высокоэффективная обработка прерыва
- Модуль защиты памяти (MPU) для безопасности приложений
- Обширная отладка и возможности трассировки.

1.3.4 Периферия Cortex-M4

Встроенный направленный диспетчер прерываний

Вложенный диспетчер прерываний, который поддерживает обработку прерывания времени ожидания с низкой задержкой.

Системный блок управления

Системный блок управления (SCB) - интерфейс подключения программатора к процессору. Он предоставляет информацию о внедрении и системный контроль, включая конфигурацию, контроль и сообщение системных исключений.

Системный таймер

Системный таймер, SysTick, является 24-битным таймером обратного отсчета.

Модуль защиты памяти

модуль защиты памяти (MPU) улучшает системную надежность, определяя характеристики памяти для различных областей применения. Это обеспечивает до восьми различных областей памяти и дополнительную предопределенную фоновую область памяти.

Модуль плавающей запятой

Модуль плавающей запятой (FPU) обеспечивает IEEE754- совместимость приложений

Примеры

UQADD16 R7, R4, R2; Добавляет полуслово из R4 к соответствующему полуслову в R2,

UQADD8 R4, R2, R5; Добавляет байты из R2 к соответствующим байтам R5, насыщает
; к 8ми битам, записывает к соответствующим битам R4

UQSUB16 R6, R3, R0; Вычитает полуслово R0 из соответствующего бита в
R3, записывает к 16 битам, записывает к соответствующим битам
; полуслово в R6

UQSUB8 R1, R5, R6; Добавляет байты из R1 к соответствующим байтам R5, насыщает
; записывает к 16 битам, записывает к соответствующим битам в R6
R1.

3.8 Инструкции по упаковке и распаковке данных

В таблице 31 представлены инструкции влияющие на распаковку и упаковку данных.

Таблица 31. Инструкции упаковки и распаковки данных

Сокращение	Описание	страница
PKH	Полуслово пакета	PKHBT и PKHTB на странице 133
SXTAB	Расширяет 8 битов до 32	SXTA и UXTA на странице 135
SXTAB16	Расширяет 8 битов до 16	SXTA и UXTA на странице 135
SXTAH	Расширяет 16 битов до 32	SXTA и UXTA на странице 135
SXTB	Расширяет знак	SXT и UXT на странице 134
SXTB16	Расширяет 8 битов до 16	SXT и UXT на странице 134
SXTH	Знак расширения полуслова	SXT и UXT на странице 134
UXTAB	Расширяет 8 битов до 32	SXTA и UXTA на странице 135
UXTAB16	Расширяет 8 битов до 16	SXTA и UXTA на странице 135
UXTAH	Расширяет 16 битов до 32	SXTA и UXTA на странице 135
UXTB	нулевое расширение байта	SXT и UXT на странице 134
UXTB16	Нулевое расширение 8 битов до 18	SXT и UXT на странице 134
UXTH	Нулевое расширение полуслова	SXT и UXT на странице 134

3.8.1 РКНВТ и РКНТВ

Архивация полуслова

Синтаксис

op{cond} {Rd}, Rn, Rm {, LSL #imm}

op{cond} {Rd}, Rn, Rm {, ASR #imm}

где

- *op* является одним из
РКНВТ сжатое полуслово, низ и верх со сдвигом
РКНТВ сжатое полуслово, верх и низ со сдвигом
- '*cond*' дополнительный код состояния
- '*Rd*' регистр назначения
- '*Rn*' первый регистр операнда
- '*Rm*' второй регистр операнда удерживающий сдвигаемое значение
- '*imm*' длина изменения. длина изменения зависит от указаний:
Для РКНВТ: LSL: сдвиг влево с длиной от 1 до 31, 0 означает отсутствие сдвига
For РКНТВ: ASR: арифметический сдвиг вправо с длиной от 1 до 32 сдвиг 32х битов обозначается 0b00000.

Действие

1. Записывает значение нижнего полуслова первого операнда к нижнему полуслову регистра назначения
 2. Если есть сдвиг, то сдвинутое значение второго операнда записывается в верхнее полуслово регистра назначения
-
1. Записывает значение верхнего полуслова операнда в верхнее полуслово регистра назначения
 2. Если есть сдвиг, то сдвинутое значение второго операнда записывается в нижнее полуслово регистра назначения

Ограничения

Rd не должно быть SP и не должно быть PC.

Условия

Задание не изменяет условия

Примеры

РКНВТ R3, R4, R5 LSL #0 ; записывает нижнее полуслово R4 в нижнее полуслово
; R3, записывает нижнее полуслово R5, бес сдвига в верхнее полуслово
; R3

РКНТВ R4, R0, R2 ASR #1 ; записывает R2 сдвинутое на 1 бит в нижнее полуслово
; R4, и записывает нижнее полуслово R0 в верхнее
полуслово R4

3.8.2 SXT и UXT

расширение знака и нуля

Синтаксис

`op{cond} {Rd}, Rm {, ROR #n}`

`op{cond} {Rd}, Rm {, ROR #n}`

где:

- *op* один из
 - SXTB Знак расширяет 8-битное значение на 32-битное
 - SXTH Знак расширяет 16-битное значение на 32-битное
 - SXTB16 Знак расширяет 2 8-битных значения на 2 32 битных
 - UXTB Ноль расширяет 8-битное значение на 32-битное.
 - UXTH Ноль расширяет 16-битное значение на 32-битное
 - UXTB16 Ноль расширяет 2 8-битных значения на 2 32 битных
- '*cond*' необязательный код состояния
- '*Rd*' регистр назначения
- '*Rm*' регистр содержащий расширяемое значение
- '*ROR #n*' один из
 - ROR #8 Значение из Rm повернуто вправо на 8 бит
 - ROR #16 Значение из Rm повернуто вправо на 16 бит
 - ROR #24 Значение из Rm повернуто вправо на 24 бита
 - Если ROR #n опущен поворот не производится

Действие

1. Изменить значение Rm вправо на 0, 8, 16 или 24 бита.
2. Извлечь биты из получившегося значения
 - SXTB извлекает биты[7:0] и знак распространяется на 32 бита.
 - UXTB извлекает биты[7:0] и ноль распространяется на 32 бита.
 - SXTH извлекает биты[15:0] и знак распространяется на 32 бита.
 - UXTH извлекает биты [15:0] и ноль распространяется на 32 бита.
 - SXTB16 извлекает биты[7:0] и знак распространяется на 16 битов. а также извлекает биты [23:16] и знак распространяется на 16 битов
 - UXTB16 извлекает биты[7:0] и ноль распространяется на 16 битов. а также извлекает биты [23:16] и ноль распространяется на 16 битов

Ограничения

Не использовать SP и PC.

Условия

Инструкции не влияют на условия

Примеры

SXTH R4, R6, ROR #16 ; Поворачивает R6 на 16 бит, получает нижнее полуслово
; как результат знак расширяется на 32 бита и записывается в R4
UXTB R3, R10 ; Извлекает байт с наименьшим значением из R10, ноль расширяется и
; записывается в R3.

3.8.3 SXTA и UXTA

Обозначенное и необозначенное расширение и добавление

Синтаксис

`op{cond} {Rd,} Rn, Rm {, ROR #n}`

`op{cond} {Rd,} Rn, Rm {, ROR #n}`

где:

- *op* один из
 - SXTAB Знак расширяет 8-битное значение до 32-битного
 - SXTAH Знак расширяет 16-битное значение до 32-битного
 - SXTAB16 Знак расширяет два 8-битных значения до двух 16-битных
 - UXTAB Ноль расширяет 8-битное значение до 32-битного
 - UXTAH Ноль расширяет 16-битное значение до 32-битного
 - UXTAB16 Ноль расширяет два 8-битных значения до двух 16-битных.
- '*cond*' дополнительный код состояния
- '*Rd*' регистр назначения
- '*Rn*' первый регистр операнда
- '*Rm*' регистр содержащий расширяемое значение
- '*ROR #n*' один из
 - ROR #8 Значение из Rm повернуто вправо на 8 бит
 - ROR #16 Значение из Rm повернуто вправо на 16 бит
 - ROR #24 Значение из Rm повернуто вправо на 24 бита
 - Если ROR #n отсутствует поворот не производится.

Действие

1. Изменить значение Rm вправо на 0, 8, 16 или 24 бита.
2. Извлечь биты из получившегося значения
 - SXTB извлекает биты[7:0] и знак распространяется на 32 бита.
 - UXTB извлекает биты[7:0] и ноль распространяется на 32 бита.
 - SXTH извлекает биты[15:0] и знак распространяется на 32 бита.
 - UXTH извлекает биты [15:0] и ноль распространяется на 32 бита.
 - SXTB16 извлекает биты[7:0] и знак распространяется на 16 битов. а также извлекает биты [23:16] и знак распространяется на 16 битов
 - UXTB16 извлекает биты[7:0] и ноль распространяется на 16 битов. а также извлекает биты [23:16] и ноль распространяется на 16 битов
3. Добавляет обозначенное или нулевое значение к слову или полуслову Rn и записывает результат в Rd.

Ограничения

Не используйте SP и PC.

Условия

Указания не влияют на условия

Примеры

SXTAH R4, R8, R6, ROR #16 ; Поворачивает R6 вправо на 16 битов, получает нижнее
; полуслово, знак расширяется до 32 бит, добавляет R8, и
; записывает в R4

UXTAB R3, R4, R10 ; Извлекает нижний бит из R10 и ноль расширяется до 32
; бит добавляет R4, и записывает в R3.

3.9 Указания битового поля

В таблице 32 указаны операции для работы со смежными наборами битов в регистрах и битовых полях.

Таблица 32. Операции для работы со смежными наборами битов

Аббревиатура	Описание	страница
BFC	Битовое поле очищено	BFC и BFI на странице 137
BFI	Вставка в битовое поле	BFC и BFI на странице 137
SBFX	Вставка обознач битового поля	SBFX и UBFX на странице 138
SXTB	Расширение байта знаком	SXT и UXT на странице 139
SXTH	Знак расширяет полуслово	SXT и UXT на странице 139
UBFX	Вставка необознач битового поля	SBFX и UBFX на странице 138
UXTB	Расширение байта нулем	SXT и UXT на странице 139
UXTH	Ноль расширяет полуслово	SXT и UXT на странице 139

3.9.1 BFC и BFI

Очистка и вставка битового поля

Синтаксис

`BFC{cond} Rd, #lsb, #width`

`BFI{cond} Rd, Rn, #lsb, #width`

где:

- ‘*cond*’ Дополнительный код состояния
- ‘*Rd*’ Регистр назначения
- ‘*Rn*’ Регистр источник
- ‘*lsb*’ Позиция наименее значимого бита в битовом поле *lsb* должно быть в пределах от 0 до 31.
- ‘*width*’ Ширина битового поля и должна быть в пределах от 1 до 32-*lsb*.

Действия

BFC очищает битовое поле регистра. Очищает биты в *Rd*, начиная с нижнего бита *lsb*. другие биты в *Rd* не изменяются.

BFI копирует битовое одного регистра в другой регистра. заменяет биты ширины в *Rd*, запускающейся в позиции младшего бита *lsb* с битами ширины от *Rn*, запускающегося в бите [0]. Другие биты в *Rd*. неизменны.

Ограничения

Не используйте SP и PC.

Условия

Указания не влияют на условия

Примеры

```
BFC  R4, #8, #12      ; Очищает биты начиная с 8 до 19 (12 битов) R4 до 0
BFI  R9, R2, #8, #12  ; Переносит бит 8 на место 19 бита (12 битов) R9 ; с 0 к
                        биты 11 от R2
```

3.9.2 SBFX и UBFX

Извлечение битового поля со знаком и извлечение битового поля без знака.

Синтаксис

`SBFX{cond} Rd, Rn, #lsb, #width`

`UBFX{cond} Rd, Rn, #lsb, #width`

где:

- `cond` Дополнительный код состояния
- `'Rd'` Регистр назначения
- `'Rn'` Регистр источник
- `'lsb'` Положение наименее важного бита в битовом поле. `lsb` должно находиться в пределе от 0 до 31.
- `'width'` ширина битового поля должна быть в диапазоне от 1 до 32-`lsb`.

Действия

SBFX извлекает битовое поле из одного регистра, знак расширяется до 32 бит и результат записывается в регистр назначения

UBFX извлекает битовое поле из одного регистра, ноль расширяется до 32 бит и результат записывается в регистр назначения

Ограничения

Не используйте SP и PC.

Условия

Указания не влияют на условия

Examples

`SBFX R0, R1, #20, #4` ; Извлекает 20 к биту 23 (4 бита) из R1 знак
; расширяется до 32 бит и записывает результат в R0.

`UBFX R8, R11, #9, #10` ; Извлекает бит 9 к биту 18 (10 бит) из R11 и ноль
; расширяется до 32 бит и записывает результат в R8

3.9.3 SXT и UXT

Расширение знака и единицы

Синтаксис

SXTend{*cond*} {*Rd*,} *Rm* {, ROR #*n*}

UXTend{*cond*} {*Rd*,} *Rm* {, ROR #*n*}

где:

- 'extend' один из
 - В: расширяет 8-битное значение до 32-битного значения
 - Н: расширяет 16-битное значение до 32-битного значения
 - *cond* Дополнительный код состояния
 - 'Rd' Регистр назначения
 - 'Rn' Регистр содержащий расширяемое значение
 - ROR #*n* один из
 - ROR #8: Значение *Rm* повернуто вправо на 8бит
 - ROR #16: Значение *Rm* повернуто вправо на 16бит
 - ROR #24: Значение *Rm* повернуто вправо на 24бита.
- Если ROR #*n* отсутствует поворота не производится

Действия

1. *Rm* переместить вправо на 0, 8, 16 или 24 бита
2. Извлекает бит из результирующего значения
 - SXTB извлекает биты[7:0] и знак расширяется до 32 бит.
 - UXTB извлекает биты[7:0] и ноль расширяется до 32 бит.
 - SXTH извлекает биты[[15:0] and sign extends to 32 bits.
 - UXTH извлекает биты[[15:0] и ноль расширяется до 32 бит.

Ограничения

Не используйте SP и PC.

Условия

Указания не влияют на условия

Примеры

```
SXTH  R4, R6, ROR #16 ; Сдвигает R6 на 16 бит затем извлекает нижнее
                        ; полуслово результата и затем знак расширяется до 32 бит
                        ; и результат записывается в R4 .
UXTB  R3, R10          ; извлекает наименее важный байт R10 и ноль
                        ; расширяет его, затем записывает результат в R3
```

3.9.4 Команды перехода и команды управления

В таблице 33 перечислены команды перехода и команды управления

Команды перехода и команды управления

Сокращение	Описание	страница
B		B, BL, BX, и BLX на странице 141
BL	Ответвление со ссылкой	B, BL, BX, и BLX на странице 141
BLX	Косвенное ответвление со ссылкой	B, BL, BX, и BLX на странице 141
BX	Косвенное ответвление	B, BL, BX, и BLX на странице 141
CBNZ	Сравнить и разветвиться в случае нуля	CBZ и CBNZ на странице 143
CBZ	Сравнить и разветвиться в случае не нуля	CBZ и CBNZ на странице 143
IT	If-Then	IT на странице 144
TBB	Ответвление табличного байта	TBB и TBH на странице 146
TBH	ТОтветвление табл полуслова	TBB и TBH на странице 146

3.9.5 В, BL, BX, и BLX

Руководство по разветвлению

Синтаксис

$B\{cond\} \ label$

$BL\{cond\} \ label$

$BX\{cond\} \ Rm$

$BLX\{cond\} \ Rm$

где:

- 'B' ответвление
- 'BL' ответвление со ссылкой
- 'BX' косвенное ответвление
- 'BLX' косвенное ответвление со ссылкой.
- 'cond' Дополнительный код состояния
- 'label' выражение близкое к PC
- 'Rm' Нтubcnh gjrfpsdf.obq flhtc hfpdtncktybz. Бит[0] в *Rm* должен быть 1, но адрес разветвления создается если бит[0] 0.

Действия

Все эти действия ведут к ответвлению в *label*, или к *Rm*. в добавок

- BL и BLX указания пишут адрес следующего указания в LR (регистр связи)
- BX и BLX вызывают ошибку использования если бит[0] *Rm* является 0.

В *cond* единственная условная инструкция, которая может быть любой внутри или снаружи блока IT. Все другие команды перехода должны быть условным выражением в блоке IT и должны быть безусловными вне блока IT

Таблица 34 показывает диапазоны для различных команд перехода.

Таблица 34 диапазоны различных команд перехода.

Указание	Диапазон
B label	от -16 МВ до +16 МВ
Bcond label (Снаружи IT блока)	от -1 МВ до +1 МВ
Bcond label (Внутри IT блока)	от -16 МВ до +16 МВ
BL{cond} label	от -16 МВ до +16 МВ
BX{cond} Rm	Любое значение регистра
BLX{cond} Rm	Любое значение регистра

Ограничения

- Не используйте PC в указаниях BLX
- Для BX и BLX, бит[0] *Rm* должен быть 1 для корректного выполнения, но ответвление идет к адресу созданному битом[0] в состоянии 0
- Когда эти указания применяются внутри IT блока они должны быть последними указаниями

Vcond - единственная условная инструкция, которой не нужно находится в, блоке IT. Однако у нее есть более длинный диапазон ответвления, когда она в блоке IT.

Условия

указания не влияют на условия

Примеры

```
B      loopA ; Ответвиться к loopA
BLE    ng    ; Опционально ответить к label ng
B.W    target ; Ответвиться к адресу в пределах 16MB
BEQ     target ; Опционально ответить к адресу в пределах 16MB
BEQ.W   target ; Опционально ответить к адресу в пределах 1MB
BL      funC  ; Ответвиться со ссылкой к функции funC.

BX      LR    ; Запрос возврата от функции
BXNE    R0    ; Опционально ответить к адресу содержащемуся в R0
BLX     R0    ; Ответвиться со ссылкой к адресу содержащемуся в ; in R0
```

3.9.6 CBZ и CBNZ

Сравнение и переход по 0 и не 0

Синтаксис

CBZ *Rn*, *label*

CBNZ *Rn*, *label*

где:

- '*Rn*' регистр содержащий операнд
- '*label*' переход к назначению

Действия

Используйте CBZ или CBNZ инструкции, чтобы избежать изменения условия и сокращать количество инструкций.

CBZ *Rn*, не меняют условия. но приравниваются к:

```
CMP      Rn, #0
BEQ      label
```

CBNZ *Rn*, не меняют условия. но приравниваются к:

```
CMP      Rn, #0
BNE      label
```

Ограничения

- *Rn* должно быть в диапазоне от R0 до R7
- Назначение перехода должно быть не далее 4 - 130 байт после указания
- Эти указания не должны использоваться в IT блоке

Условия

Указания не влияют на условия

Примеры

```
CBZ      R5, target ; смещение вперед если R5 = 0
CBNZ     R0, target ; смещение вперед если R0 не равен 0
```

3.9.7 IT

If-Then состояния

Синтаксис

`IT{x{y{z}}}` *cond*

где;

- 'x' определяет переключатель условия для второй инструкции в блоке IT.
- 'y' определяет переключатель условия для третьей инструкции в блоке IT.
- 'z' определяет переключатель условия для четвертой инструкции в блоке IT
- 'cond' определяет переключатель условия для первой инструкции в блоке IT

Переключатель условия для второй, третьей и четвертой инструкции в блоке IT может быть также If: Then. Применяет состояние *cond* к указанию

E: Else. Применяет инверсное состояние *cond* к указанию

- b) возможно использовать AL (всегда) для *cond* в инструкции IT. Если это сделано, все инструкции в блоке IT должны быть безусловными, и каждый из x, y, и z должны быть T или отсутствовать, но не E.

Действия

Указание IT делает до четырех следующих указаний состояния. Условия могут быть все одинаковыми, или некоторые из них могут быть логической инверсией других. Условные инструкции после инструкции IT формируют блок IT.

Инструкции в блоке IT, включая любые отделения, должны определить условие в {*cond*} части их синтаксиса.

Ваш ассемблер мог бы быть в состоянии произвести необходимые инструкции по IT для условных инструкций автоматически, так, чтобы вы не должны были писать их сами.

А ВКРП инструкция в блоке IT всегда выполняется, даже если его условие нет

Исключения могут быть между инструкцией по IT и соответствующим блоком IT, или в пределах блока IT. Такое исключение приводит к входу в соответствующего укладчика исключения с подходящей информацией о возвращении в LR и PSR.Is

Инструкции, разработанные для использования для прибыли исключения, могут использоваться в качестве нормальных, чтобы возвратиться из исключения, и выполнение блока IT возобновляется правильно. Это - единственный способ, которым изменяющей PC инструкции разрешают ветвиться к инструкции в блоке IT.

Ограничения

Следующие условия не допустимы

- IT
- CBZ и CBNZ
- CPSID и CPSIE.

- любая инструкция, которая изменяет PC, должны или быть вне блока IT или должна быть последняя инструкция в блоке IT. Это:
 - ADD PC, PC, Rm
 - MOV PC, Rm
 - B, BL, BX, BLX
 - Любое LDM, LDR, или POP условие которое записывает PC
 - TBB и TBH
- Не переходить ни к какой инструкции в блоке IT, кроме тех случаев, когда, возвращаясь от укладчика исключения
- Все условные инструкции кроме Vcond должны быть в блоке IT. Vcond может быть либо внутри либо снаружи блока IT но если большой ряд отклонений если снаружи.
- Каждая инструкция в блоке IT должна определить кодовый суффикс условия.

Ваш ассемблер мог бы установить дополнительные ограничения для использования блоков IT, такие как запрещение использования директив ассемблера в пределах них.

Условия

Указания не влияют на условия

Example

```

ITTE    NE                ; Следующие 3 указания необязательны
ANDNE   R0, R0, R1        ; ANDNE не обновляет указания
ADDSNE  R2, R2, #1        ; ADDSNE обновляет указания
MOVEQ   R2, R3            ; Сдвиг указаний

CMP     R0, #9            ; изменить значение R0 (0-15) на ASCII
                                ; ('0'-'9', 'A'-'F')
        ITE      GT        ; следующие 2 условия необязательны
ADDGT   R1, R0, #55        ; изменить 0xA -> 'A'
ADDLE   R1, R0, #48        ; изменить 0x0 -> '0'

IT      GT                ; IT блок с одним необязательным указанием
ADDGT   R1, R1, #1        ; Условно увеличивает R1

ITTEE   EQ                ; Следующие 3 указания необязательны
MOVEQ   R0, R1            ; Опциональный сдвиг
ADDEQ   R2, R2, #10        ; Опциональное добавление
ANDNE   R3, R3, #1        ; Опционное И
BNE.W   dloop            ; Указание сдвига может быть использовано только в конце
                                ; указаний IT блока

IT      NE                ; Следующее условие опционально
ADD     R0, R0, R1        ; Ошибка синтаксиса: не используется кода состояний

```

3.9.8 TBB и TBH

Синтаксис

TBB [*Rn*, *Rm*]

TBH [*Rn*, *Rm*, LSL #1]

где:

- '*Rn*' Регистр содержащий адрес таблицы разветвления
Если *Rn* является PC, то адрес таблицы немедленно следует инструкции TBB TBH
- '*Rm*' индексный регистр. Он содержит индекс таблицы. Для таблиц полуслова, LSL #1 удваивает значение в *Rm* чтобы сформировать правильное смещение в таблицу.

Действия

Эти инструкции вызывают прямое ответвление, используя таблицу единственных байтовых смещений для TBB или смещений полуслова для TBH. *Rn* обеспечивает указатель на таблицу, и *Rm* предоставляет индекс в таблицу. Для TBB смещение ответвления - дважды значение без знака байта, возвращенного из таблицы. и для TBH смещение ответвления - дважды значение без знака полуслова, возвращенного из таблицы. Ответвление сразу происходит с адресом при том смещении от адреса байта после TBB или инструкции TBH.

Ограничения

- *Rn* не должно быть SP
- *Rm* не должно быть SP либо PC
- Когда любая из этих инструкций используется в блоке IT, она должна быть последней

Условия

Указания не влияют на условия

Примеры

ADR.W R0, BranchTable_Byte

TBB [R0, R1] ; R1 индекс, R0 Базовый индекс таблицы ответвления Case1

; последовательность инструкции совпадает

Case2

; последовательность инструкции совпадает

Case3

; последовательность инструкции совпадает

BranchTable_Byte

DCB 0 ; Case1 вычисление смещения

DCB ((Case2-Case1)/2) ; Case2 вычисление смещения

DCB ((Case3-Case1)/2) ; Case3 вычисление смещения

TBN [PC, R1, LSL #1] ; R1 индекс PC используется как основа для таблицы
ответвления

BranchTable_H

DCI ((CaseA - BranchTable_H)/2) ; CaseA вычисление смещения

DCI ((CaseB - BranchTable_H)/2) ; CaseB вычисление смещения

DCI ((CaseC - BranchTable_H)/2) ; CaseC вычисление смещения

CaseA

; последовательность инструкции совпадает

CaseB

; последовательность инструкции совпадает

CaseC

; последовательность инструкции совпадает

3.10 Инструкции с плавающей точкой

Эти инструкции только доступны, если FPU включен и включен в системе.

Инструкции с плавающей точкой

сокращение	описание	страница
VABS	Абсолютное число с плавающей точкой	VABS на странице 149
VADD	Добавление с плавающей точкой	VADD на странице 150
VCMP	Сравнение двух регистров с плавающей точкой либо одного регистра с 0	VCMP, VCMPE на странице 150
VCMPE	Сравнение двух регистров с плавающей точкой либо одного регистра и 0 с проверкой	VCMP, VCMPE на странице 150
VCVT	Преобразование плавающей точки в целое число	VCVT, VCVTR на странице 151
VCVT	Преобразование фиксированной и плавающей	VCVT на странице 152
VCVTR	Преобразование плавающей точки и целого числа с округлением	VCVT, VCVTR на странице 151
VCVTB	Преобразовывает значение полуточности в одинарную точность	VCVTB, VCVTT на странице 153
VCVTT	Преобразовывает значение одинарной точности в полуточность	VCVTB, VCVTT на странице 153
VDIV	Деление числа с плавающей точкой	VDIV на странице 154
VFMA	Умножение числа с плавающей точкой	VFMA, VFMS на странице 154
VFNMA	Инвертированное умножение числа с плавающей точкой	VFNMA, VFNMS на странице 155
VFMS	Умножение вычитание	VFMA, VFMS на странице 154
VFNMS	Инверсное умножение вычитание	VFNMA, VFNMS на странице 155
VLDM	Загрузка многократных регистров расширения	VLDM на странице 156
VLDR	Загружает регистр расширения из памяти	VLDR на странице 157
VLMA	Умножение накопление	VLMA, VLMS на странице 158
VLMS	Умножение деление	VLMA, VLMS на странице 158
VMOV	Перемещение	VMOV на странице 158
VMOV	Регистр движения	VMOV на странице 159
VMOV	Скопировать регистр ARM одинарной точности	VMOV на странице 159
VMOV	Скопировать 2 регистра ARM одинарной точности	VMOV ARM на странице 160
VMOV	Копирует от регистра к скаляру	VMOV на странице 161
VMOV	Копирует от скаляра к регистру	VMOV ARM на странице 162

Таблица 35 указания для действий с числами с плавающей точкой

Сокращение	Описание	Страница
VMRS	От системного регистра плавающей точки переместить к регистру ядра ARM	VMRS на странице 162
VMSR	От регистра ядра ARM переместить к системному регистру плавающей точки	VMSR на странице 163
VMUL	Умножение плавающей точки	VMUL на странице 163
VNEG	Инвертирование плавающей точки	VNEG на странице 164
VNMLA	Добавление плавающей точки	VNMLA, VNMLS, VNMUL на странице 165
VNMLS	Добавление и извлечение плавающей точки	VNMLA, VNMLS, VNMUL на странице 165
VNMUL	Умножение плавающей точки	VNMLA, VNMLS, VNMUL на странице 165
VPOP	Вытолкнуть регистры смещения	VPOP на странице 166
VPUSH	Задвинуть регистры смещения	VPUSH на странице 166
VSQRT	Квадратный корень с плавающей точкой	VSQRT на странице 167
VSTM	Сохранение многократных регистров смещения	VSTM на странице 167
VSTR	Сохраняет регистры смещения в память	VSTR на странице 168
VSUB	Извлечение плавающей точки	VSUB на странице 169

3.10.1 VABS

Синтаксис

VABS{cond}.F32 Sd, Sm

где

- 'cond' дополнительный код состояния
- 'Sd, Sm' значение назначения плавающей точки и значение плавающей точки операнда

Действия

1. Принимает абсолютное значение операнда регистра с плавающей точкой.
2. Помещает результаты в целевой регистр с плавающей точкой.

Ограничения

Ограничения отсутствуют

Условия

Инструкция с плавающей точкой очищает

знаковый бит.

Примеры

VABS.F32 S4, S6

3.10.2 VADD

Добавление с плавающей точкой

Синтаксис

VADD{cond}.F32 {Sd,} Sn, Sm

где:

- 'cond' Дополнительный код состояния
- 'Sd' Значение назначения плавающей точки
- 'Sn, Sm' Назначение плавающих точек операнда

Действия

1. Добавляют значения в двух регистрах операнда с плавающей точкой.
2. Помещает результаты в целевой регистр с плавающей точкой.

Ограничения

Ограничений нет

Условия

Указания не влияют на условия

Примеры

VADD.F32 S4, S6, S7

3.10.3 VCMPE, VCMPE

Сравнивает два регистра с плавающей точкой, или один регистр с плавающей точкой и нуль.

Синтаксис

VCMPE{E}{cond}.F32 Sd, Sm

VCMPE{E}{cond}.F32 Sd, #0.0

где:

- 'cond' Дополнительный код состояния
- 'E' Если существующий, какой-либо операнд NaN вызывает Недопустимое исключение
- 'Sd' Операнд с плавающей точкой для сравнения.
- 'Sm' Операнд с плавающей точкой для сравнения

Действия

1. Сравнивает:
 - Два регистра с плавающей точкой.
 - Один регистр с плавающей точкой и один ноль
1. Записывает результат в FPSCR

Ограничения

Эта инструкция может повысить Недопустимое исключение Работы, если любой операнд - какой-либо тип NaN. Это всегда повышает Недопустимое исключение Работы, если любой операнд - сигнальный NaN.

Условия

Когда эта инструкция пишет результат во флаги FPSCR, значения обычно передаются флагам ARM последующей инструкцией VMRS

Примеры

VCMP.F32 S4, #0.0

VCMP.F32 S4, S2

3.10.4 VCVT, VCVTR Между плавающей точкой и целым числом

Преобразовывает значение в регистре с плавающей точкой до 32-разрядного целого числа.

Синтаксис

VCVT{R}{cond}.Tm.F32 Sd, Sm

VCVT{cond}.F32.Tm Sd, Sm

where:

- 'R' .
Если R определен, работа использует округляющийся режим, определенный FPSCR. Если R опущен, работа использует Раунд, по направлению к нулю округляющий режим.
- 'Tm' Тип данных операнда
S32 Обозначенный 32 битный.
U32 Необозначенный 32 битный
- 'Sd, Sm' являются целевым регистром и регистром операнда.

Действия

1. Также
 - Преобразовывает значение в регистре от значения с плавающей точкой до 32-разрядного целого числа.
2. Помещает результат во второй регистр

С плавающей точкой к целочисленной операции обычно использует Раунд, по направлению к нулю округляющий режим, но может дополнительно использовать округляющийся режим, определенный FPSCR.

Целое число к работе с плавающей точкой использует округляющийся режим, определенный FPSCR.

Ограничения

Ограничений нет

Условия

Указания не изменяют условия

3.10.5 VCVT Между плавающей и фиксированной точкой

Преобразовывает значение в регистре от с плавающей точкой до и от фиксированной точки.

```
VCVT{cond}.Td.F32 Sd, Sd, #fbits
```

```
VCVT{cond}.F32.Td Sd, Sd, #fbits
```

где:

- '*cond*' Дополнительный код состояния
- '*Td*' Тип данных фиксированной точки
 - S16 Обозначенное 16 битное значение
 - U16 Необозначенное 16 битное значение
 - S32 Обозначенное 32 битное значение
 - U32 Необозначенное 32 битное значение
- '*Sd*' Регистр назначения и регистр операнда
- '*fbits*' число дробных битов в числе фиксированной точки:
 - Если *Td* имеет значение S16 или U16, число битов должно быть в диапазоне 0-16.
 - Если *Td* имеет значение S32 или U32, число битов должно быть в диапазоне 1-32.

Действия

Также

Преобразовывает значение в регистре от фиксированной точки до с плавающей точкой. Помещает результат во второй регистр. Значения с плавающей точкой - одинарная точность. Значение фиксированной точки может быть 16-разрядным или 32-разрядным. Преобразования из значений фиксированной точки берут свой операнд от битов младшего разряда исходного регистра и игнорируют любые остающиеся биты. Преобразования со знаком в знак значений фиксированной точки - расширяют значение результата до целевой ширины регистра. Преобразования без знака в нуль значений фиксированной точки - расширяют значение результата до целевой ширины регистра. С плавающей точкой к работе фиксированной точки использует Раунд, по направлению к нулю округляющий режим. Фиксированная точка к работе с плавающей точкой использует Раунд для Самого близкого режима округления.

Ограничения

Ограничений нет

Условия

Указания не влияют на условия

3.10.6 VCVTB, VCVTT

Преобразовывает значения полуточности и значения одинарной точности.

Синтаксис

$\text{VCVT}\{y\}\{\text{cond}\}.\text{F32}.\text{F16} \text{ Sd}, \text{ Sm}$

$\text{VCVT}\{y\}\{\text{cond}\}.\text{F16}.\text{F32} \text{ Sd}, \text{ Sm}$

Где:

- 'y' Определяет, какая половина операнда регистрирует Sm

Если y имеет значение B, то нижняя половина битов [15:0], Sm Sd используется.

Если y имеет значение T, то верхняя половина битов [31:16], Sm Sd используется.

- 'cond' дополнительный код состояния
- 'Sd' Регистр назначения
- 'Sm' Регистр операнда

Действия

1. Преобразовывает значение полуточности в вершине или нижней половине точности.
2. Записывает результат в регистр одинарной точности
3. Преобразовывает значение в регистре одинарной точности к полуточности.
4. Пишет результат в вершину или нижнюю половину регистра одинарной точности, сохраняя другую половину целевого регистра.

Ограничения

Ограничений нет.

Условия

Указания не влияют на условия

3.10.7 VDIV

Делит значения плавающей точки.

Синтаксис

`VDIV{cond}.F32 {Sd,} Sn, Sm`

где:

- '*cond*' Дополнительный код состояния
- '*Sd*' Целевой регистр
- '*Sn, Sm*' Регистры операнда

Действия

1. Делит одно значение с плавающей точкой на другое значение с плавающей точкой.
2. Пишет результат в целевой регистр с плавающей точкой.

Ограничения

Ограничений нет

Условия

Указания не влияют на условия.

3.10.8 VFMA, VFMS

Синтаксис `VFMA{cond}.F32`

`{Sd,} Sn, Sm VFMS{cond}.F32`

`{Sd,} Sn, Sm`

где:

- '*cond*' Дополнительный код состояния
- '*Sd*' Регистр назначения
- '*Sn, Sm*' Регистры операнда

Действия

1. Умножает значения с плавающей точкой в регистрах операнда.
2. Накапливает результаты в целевой регистр.
3. Результат умножения не округляется перед накоплением.

VFMS указания

1. Инвертирует первый регистр операнда.
2. Умножает значения с плавающей точкой первых и вторых регистров операнда.
3. Добавляют продукты к целевому регистру.
4. Помещает результаты в целевой регистр.
5. Результат умножения не округляется перед дополнением.

Ограничения

Ограничений нет.

Условия

Указания не влияют на условия

3.10.9 VFNMA, VFNMS

Syntax

VFNMA{cond}.F32 {Sd,} Sn, Sm

VFNMS{cond}.F32 {Sd,} Sn, Sm

where:

- 'cond' дополнительный код состояния
- 'Sd' Целевой регистр
- 'Sn, Sm' Регистры операнда

Действия

1. Инвертирует первый регистр операнда с плавающей точкой.
2. Умножает первый операнд с плавающей точкой со вторым операндом с плавающей точкой.
3. Добавляет, что отрицание плавления - указывает на целевой регистр на продукт
4. Помещает результат в целевой регистр.

Результат умножения не округляется перед

дополнением.

1. Умножает первый операнд с плавающей точкой со вторым операндом с плавающей точкой.
2. Добавляет отрицание значения с плавающей точкой в целевом регистре к продукту.
3. Помещает результат в целевой регистр.

Результат умножения не округляется перед дополнением

Ограничения

Ограничений нет

Условия

Указания не влияют на условия

3.10.10 VLDM

Множественная загрузка с плавающей точкой

Синтаксис

`VLDM{mode}{cond}{.size} Rn{!}, list`

где:

- *'mode'* способ адресации:
IA: Инкремент После. Последовательные адреса запускаются в адресе, определенном в Rn. DB: Декремент Прежде. Последовательный конец адресов как раз перед адресом определен в Rn.
- *'cond'* Дополнительный код состояния
- *'Size'* спецификатор размера дополнительных данных.
- *'Rn'* базовый регистр SP может быть использован
- *'!'* команда к инструкции, чтобы записать измененное значение обратно к Rn. Это требуется если режим == DB, и дополнительное если режим == IA.
- *'list'* список регистров расширения, двойное слово или регистры однословные, разделенные запятыми и окруженные скобками.

Действия

Эта инструкция загружает множественные регистры расширения из последовательных ячеек памяти, используя адрес от регистра ядра ARM как базовый адрес.

Ограничения

- Если размер присутствует, он равен размеру в битах, 32 или 64, регистров в списке.
- базовый регистр SP может быть использован
- В системе команд ARM, если ! не определен PC может использоваться.
- список должен содержать по крайней мере один регистр.
- При использовании Декремента Перед способом адресации запись назад отмечает!, должен быть добавлен к спецификации базового регистра.

Условия

Команды не влияют на условия

3.10.11 VLDR

Загружает единственный регистр расширения из памяти

Синтаксис

```
VLDR{cond}{.64} Dd, [Rn{#imm}]  
VLDR{cond}{.64} Dd, label  
VLDR{cond}{.64} Dd, [PC, #imm]  
VLDR{cond}{.32} Sd, [Rn {, #imm}]  
VLDR{cond}{.32} Sd, label  
VLDR{cond}{.32} Sd, [PC, #imm]
```

где:

- *'cond'* Дополнительный код состояния
- *'64, 32'* спецификаторы размера дополнительных данных.
- *Dd* целевой регистр для загрузки двойного слова.
- *Sd* целевой регистр для загрузки одного слова.
- *Rn* базовый регистр. SP может использоваться.
- *imm* + или - непосредственное смещение раньше формировало адрес.
Разрешенные значения адреса - варьируются в диапазоне от 0 до 1020.
- *label* метка литерального элемента данных, который будет загружен.

Действия

Эта инструкция загружает единственный регистр расширения из памяти, используя базовый адрес от регистра ядра ARM, с дополнительным смещением.

Ограничения

Ограничений нет

Условия

Команды не влияют на условия

3.10.12 VLMA, VLMS

Умножает два значения с плавающей точкой, и накапливает или вычитает результаты.

Синтаксис

VLMA{cond}.F32 Sd, Sn, Sm

VLMS{cond}.F32 Sd, Sn, Sm

где:

- 'cond' Дополнительный код состояния
- 'Sd' целевое значение с плавающей точкой
- 'Sn, Sm' операнд значения с плавающей точкой.

Действия

Умеожение накопление с плавающей точкой:

1. Умножает два значения с плавающей точкой.
2. Добавляют результаты к целевому значению с плавающей точкой.

Умножение извлечение с плавающей точкой

1. Умножает два значения с плавающей точкой.
2. Добавляют результаты к целевому значению с плавающей точкой.

Помещает результаты в целевой регистр.

Ограничения

Ограничений нет

Условия

Инструкции не влияют на условия

3.10.13 VMOV непосредственное

Непосредственное перемещение с плавающей точкой

Синтаксис

VMOV{cond}.F32 Sd, #imm

где:

- 'cond' Дополнительный код состояния
- 'Sd' место назначения ответвления
- 'imm' константа с плавающей точкой.

Действия

Эта инструкция копирует постоянное значение в регистр с плавающей точкой.

Ограничения

ограничений нет

Условия

Инструкции не влияют на условия

3.10.14 VMOV Регистр

Копирует содержание одного регистра другому.

Синтаксис

VMOV{cond}.F64 Dd, Dm

VMOV{cond}.F32 Sd, Sm

где:

- 'cond' Дополнительный код состояния
- 'Dd' целевой регистр, для работы двойного слова.
- 'Dm' исходный регистр, для работы двойного слова.
- 'Sd' целевой регистр, для работы одного слова.
- 'Sm' исходный регистр, для работы одного слова.

Действия

Копирует содержание одного регистра другому.

Ограничения

Ограничений нет

Условия

Указания не влияют на условия

3.10.15 VMOV скаляр к регистру ядра ARM

Передает одно слово двойного слова регистр с плавающей точкой к регистру ядра ARM.

Синтаксис

VMOV{cond} Rt, Dn[x]

где:

- 'cond' Дополнительный код состояния
- 'Rt' целевой регистр ядра ARM.
- 'Dn' 64-разрядный регистр двойного слова.
- 'x' Определяет, какую половину двойного слова регистрируют, чтобы использовать:
Если x = 0, Использовать нижнюю если x = 1, использовать верхнюю половину

Действия

Эта инструкция передает одно слово от верхней или более низкой половины двойного слова регистр с плавающей точкой к регистру ядра ARM.

Ограничения

Rt не может быть PC или SP.

Условия

Команды не влияют на условия

3.10.16 VMOV ARM базовый регистр к одинарной точности

Передаёт регистр одинарной точности и от регистра ядра ARM.

Синтаксис

VMOV{cond} Sn, Rt

VMOV{cond} Rt, Sn

где:

- 'cond' Дополнительный код состояния
- 'Sn' Регистр одинарной точности с плавающей точкой
- 'Rt' регистр ядра ARM.

Действия

- Содержание одинарной точности регистрируется к регистру ядра ARM.
- Содержание ядра ARM регистрируется к регистру одинарной точности.

Ограничения

Rt не могут быть PC или SP.

Условия

Команды не влияют на условия

3.10.17 VMOV два ядра ARM регистрируются к двум одинарным точностям

Передаёт две последовательно пронумерованных одинарных точности регистрируются к и от двух регистров ядра ARM.

Синтаксис

VMOV{cond} Sm, Sm1, Rt, Rt2

VMOV{cond} Rt, Rt2, Sm, Sm

where:

- 'cond' Дополнительный код состояния
- 'Sm' первый регистр одинарной точности.
- 'Sm1' второй регистр одинарной точности (следующий регистр одинарной точности после Sm)
- 'Rt' регистр ядра ARM, от которого Sm передан или.
- 'Rt2' регистр ядра ARM, от которого Sm1 передан или.

Действия

1. Содержание двух последовательно пронумерованных одинарных точностей регистрируется к двум ядрам ARM Регистра
2. Содержание двух ядер ARM регистрируется к паре регистров одинарной точности.

Ограничения

- Регистры с плавающей точкой должны быть непрерывными, один за другим.
- Регистры ядра ARM не должны быть непрерывными.
- Rt не может быть PC или SP.

Условия

Указания не влияют на условия

3.10.18 VMOV ARM Базовый регистр к скаляру

Передаёт одно слово к регистру с плавающей точкой от регистра ядра ARM.

Синтаксис

`VMOV{cond}{.32} Dd[x], Rt`

где:

- *'cond'* Дополнительный код состояния
- 32 спецификатор размера дополнительных данных.
- *Dd[x]* место назначения, где [x] определяет, которая половина двойного слова передана,

Если $x = 0$, извлекается нижняя половина, Если $x = 1$, извлекается верхняя половина

- *Rt* - исходный регистр ядра ARM.

Действия

Эта инструкция передаёт одно слово верхней или нижней половине двойного слова регистра с плавающей точкой от регистра ядра ARM.

Ограничения

Rt не может быть PC или SP.

Условия

Указания не влияют на условия

3.10.19 VMRS

Перемещает слово из регистра с плавающей точкой в регистр ядра ARM

Синтаксис

`VMRS{cond} Rt, FPSCR`

`VMRS{cond} APSR_nzcv, FPSCR`

where:

- *'cond'* Дополнительный код состояния
- *'Rt'* - целевой регистр ядра ARM. Этот регистр может быть R0-R14.
- *'APSR_nzcv'* Передаёт флаги с плавающей точкой флагам APSR.

Действия

1. Копирует значение FPSCR к регистру общего назначения.
2. Копирует значение флаговых битов FPSCR к APSR N, Z, C, и V флагов.

Ограничения

Rt не могут быть PC или SP.

Условия

Эти инструкции дополнительно изменяют флаги: N, Z, C, V

3.10.20 VMSR

Перемещает из ядра ARM к регистру с плавающей точкой.

Синтаксис

VMSR{cond} FPSCR, Rt

где

- 'cond' Дополнительный код состояния
- 'Rt' - регистр общего назначения, который будет передан FPSCR.

Действия

Эта инструкция перемещает значение регистра общего назначения к FPSCR.

Ограничения

Rt не может быть PC или SP.

Условия

Эта команда обновляет FPSCR.

3.10.21 VMUL

Умножение с плавающей точкой

Синтаксис

VMUL{cond}.F32 {Sd,} Sn, Sm

where:

- 'cond' Дополнительный код состояния
- 'Sd' - целевое значение с плавающей точкой
- 'Sn, См' является операндом значения с плавающей точкой.

Действия

1. Умножает два значения с плавающей точкой.
2. Помещает результаты в целевой регистр.

Ограничения

Ограничений нет

Условия

Указания не влияют на условия

3.10.22 VNEG

Инвертирование с плавающей точкой

Синтаксис

VNEG{cond}.F32 Sd, Sm

where:

- 'cond' Дополнительный код состояния
- 'Sd' - целевое значение с плавающей точкой
- 'Sn, См' является операндом значения с плавающей точкой.

Действия

1. Инвертирует значение с плавающей точкой.
2. Помещает результаты во второй регистр с плавающей точкой.
3. Инструкция с плавающей точкой инвертирует знаковый бит.

Ограничения

Ограничений нет

Условия

Указания не влияют на условия

3.10.23 VNMLA, VNMLS, VNMUL

Умножение с плавающей точкой с отрицанием, сопровождаемым, добавлением или вычитанием

Синтаксис

`VNMLA{cond}.F32 Sd, Sn, Sm`

`VNMLS{cond}.F32 Sd, Sn, Sm`

`VNMUL{cond}.F32 {Sd,} Sn, Sm`

где:

- ‘cond’ Дополнительный код состояния
- ‘Sd’ - целевое значение с плавающей точкой
- ‘Sn, См’ является операндом значения с плавающей точкой.

Действия

1. Умножает два значения регистра с плавающей точкой.
2. Adds the negation of the floating-point value in the destination register to the negation
Добавляет отрицание значения с плавающей точкой в целевом регистре
3. Записывает результат обратно к целевому регистру.

Инструкция VNMLS:

1. Умножает два значения регистра с плавающей точкой.
2. Добавляет отрицание значения с плавающей точкой в целевом регистре к результату.
3. записывает результат обратно к целевому регистру.

Инструкция VNMUL:

1. Умножает вместе два значения регистра с плавающей точкой.
2. Пишет отрицание результата к целевому регистру.

Ограничения

Ограничений нет

Условия

Указания не влияют на условия

3.10.24 VPOP

Регистр расширения с плавающей точкой

Синтаксис

VPOP{cond}{.size} list

где:

- 'cond' Дополнительный код состояния
- спецификатор размера дополнительных данных. Если существует, он должен быть равен размеру в битах, 32 или 64, регистров в списке.
- 'list' список регистров расширения, которые будут загружены как список ~ двойное слово или регистры однословные, разделенные запятыми и окруженные скобками.

Действия

Эта инструкция загружает многократные последовательные регистры расширения из стека.

Ограничения

Список должен содержать по крайней мере один регистр и не больше чем шестнадцать регистров.

Условия

Указания не влияют на условия

3.10.25 VPUSH

Регистр расширения с плавающей точкой

Синтаксис

VPUSH{cond}{.size} list

где

- 'cond' Дополнительный код состояния
- 'size' спецификатор размера дополнительных данных. Если существует, он должен быть равен размеру в битах, 32 или 64, регистров в списке.
- 'list' список регистров расширения, которые будут загружены как список ~ двойное слово или регистры однословные, разделенные запятыми и окруженные скобками.

Действия

Эта инструкция загружает многократные последовательные регистры расширения из стека.

Ограничения

Список должен содержать по крайней мере один регистр и не больше чем шестнадцать регистров.

Условия

Указания не влияют на условия

3.10.26 VSQRT

Квадратный корень с плавающей точкой.

Синтаксис

`VSQRT{cond}.F32 Sd, Sm`

где:

- *'cond'* Дополнительный код состояния
- *'Sd'* целевое значение с плавающей точкой
- *'Sm'* операндовое значение с плавающей точкой.

Действия

1. Вычисляет квадратный корень значения в регистре с плавающей точкой.
2. Пишет результат в другой регистр с плавающей точкой.

Ограничения

Ограничений нет

Условия

Указания не влияют на условия

3.10.27 VSTM

Множественное умножение с плавающей точкой.

Синтаксис

`VSTM{mode}{cond}{.size} Rn{!}, list`

где:

- *'mode'* способ адресации:
Инкремент IA После. Последовательные адреса запускаются в адресе, определенном в Rn. Это - значение по умолчанию и может быть опущено.
Декремент DB Прежде. Последовательный конец адресов как раз перед адресом определен в Rn.
- *'cond'* Дополнительный код состояния
- спецификатор размера дополнительных данных. Если существует, он должен быть равен размеру в битах, 32 или 64, регистров в списке.
- *'Rn'* - базовый регистр. SP может использоваться.
- *'!'* функция, которая заставляет инструкцию записывать измененное значение обратно к Rn.
- *'List'* - список регистров расширения, которые будут сохранены как список последовательно пронумерованного двойного слова или регистры однословные, разделенные запятыми и окруженные скобками.

Действия

Эта инструкция хранит множественные регистры расширения к последовательным ячейкам памяти, используя базовый адрес от регистра ядра ARM.

Ограничения

- Список должен содержать минимум 1 регистр
- Если список содержит регистры двойного слова, то он не должен содержать больше чем 16 регистров.

Условия

Указания не влияют на условия

3.10.28 VSTR

Хранение с плавающей точкой

Синтаксис

VSTR{cond}{.32} Sd, [Rn{, #imm}]

VSTR{cond}{.64} Dd, [Rn{, #imm}]

где

- 'cond' Дополнительный код состояния
- '32, 64' спецификаторы размера дополнительных данных.
- 'Sd' исходный регистр для однословного хранилища
- 'Dd' исходный регистр для хранилища двойного слова.
- 'Rn' базовый регистр. SP может использоваться.
- или - непосредственное смещение раньше формировало адрес. Значения - сеть хранилищ 4 в диапазоне 0-1020. imm может быть опущен, означая смещение +0.

Действия

Эта инструкция хранит единственный регистр расширения к памяти, используя адрес от регистра ядра ARM, с дополнительным смещением, определенным в imm.

Ограничения

Ограничения - использование PC для Rn, осуждается.

Условия

Указания не влияют на условия

3.10.29 VSUB

Вычитание с плавающей точкой

Синтаксис

`VSUB{cond}.F32 {Sd,} Sn, Sm`

где

- ‘*cond*’ дополнительный код состояния
- ‘*Sd*’ - целевое значение с плавающей точкой
- ‘*Sn, Sm*’ является операндом значения с плавающей точкой.

Действия

1. Вычитает одно значение с плавающей точкой из другого значения с плавающей точкой.
2. Помещает результаты в целевой регистр с плавающей точкой.

Ограничения

Ограничений нет

Условия

Указания не влияют на условия

3.11 Остальные инструкции

Таблица 36 остальные инструкции

сокращение	описание	страница
BKPT	контрольная точка	BKPT на странице 170
CPSID	Изменение состояние процессора,	CPS на странице 171
CPSIE	Включить прерывания	CPS на странице 171
DMB	Барьер памяти данных	DMB на странице 172
DSB	Барьер синхронизации данных	DSB на странице 172
ISB	Барьер синхронизации команд	ISB на странице 173
MRS	Сдвиг от специального регистра	MRS на странице 173
MSR	Сдвиг к специальному регистру	MSR на странице 174
NOP	Нет действий	NOP на странице 175
SEV	Отправка события	SEV на странице 175
SVC	Вызов супервайзора	SVC на странице 176
WFE	Ожидание события	WFE на странице 176
WFI	Ожидание прерывания	WFI на странице 177

3.11.1 BKPT

контрольная точка.

Синтаксис

BKPT #imm

где:

- 'imm' оценка выражения к целому числу в диапазоне 0-255 (8-разрядное значение).

Действия

Инструкция BKPT заставляет процессор вводить состояние Отладки. Отладчики могут использовать это, чтобы исследовать системное состояние, когда инструкция в определенном адресе достигнута.

imm проигнорирован процессором. При необходимости отладчик может использовать его, чтобы хранить дополнительную информацию

Инструкция BKPT может быть помещена в блоке IT, но она выполняется безоговорочно, незатронутый условием, определенным инструкцией IT.

Условия

Условия не изменяют команды

Пример

ВКРТ 0xAB ; Инструкция ВКРТ может быть помещена в блоке IT, но она выполняется безоговорочно, незатронутая условием, определенным инструкцией IT.

3.11.2 CPS

Изменение состояния процессора.

Синтаксис

CPSeffect iflags

где

- *'effect'* является одним из
IE: Очищает регистр особого назначения.
ID: Устанавливает регистр особого назначения.
- *'iflags'* последовательность одного или более флагов:
i: Указать или очистить PRIMASK.
f: Указать или очистить FAULTMASK.

Действие

CPS изменяет PRIMASK и специальные значения регистра FAULTMASK. Смотрите регистры маски Исключения на странице 22 для получения дополнительной информации об этих регистрах.

Ограничения :

- Используйте CPS только из привилегированного программного обеспечения, он не имеет
- никакого эффекта, если используется в непривилегированном программном обеспечении
CPS не может быть условным выражением и так не должен использоваться в блоке IT.

Условия

Указания не влияют на условия

Examples

CPSID i ; Отключает прерывания, и конфигурируемые обработчики отказа (устанавливает PRIMASK) ,
CPSID f ; Отключает прерывания, и все обработчики отказа (устанавливает FAULTMASK)
CPSIE i ; Включает прерывания, и конфигурируемые обработчики отказа (очищает PRIMASK)
CPSIE f ; Включает прерывания, и все обработчики отказа (очищает FAULTMASK)

3.11.3 DMB

Барьер памяти данных.

Синтаксис

`DMB{cond}`

где; 'cond' Дополнительный код состояния

Действия

DMB действует как барьер памяти данных. Это гарантирует, что все явные доступы памяти, которые появляются, в порядке программы, перед инструкцией DMB, завершены перед любыми явными доступами памяти, которые появляются, в порядке программы, после инструкции DMB. DMB не влияет на упорядочивание или выполнение инструкций, которые не получают доступ к памяти.

Условия

Указания не влияют на условия

Примеры

`DMB ;` Барьер памяти данных.

3.11.4 DSB

Барьер синхронизации данных.

Синтаксис

`DSB{cond}`

где; 'cond' Дополнительный код состояния

Действия

DSB действует как специальный барьер памяти синхронизации данных. Инструкции, которые прибывают после DSB в порядке программы, не выполняются, пока инструкция DSB не завершается. Инструкция DSB завершается, когда все явные доступы памяти перед нею завершаются.

Условия

Указания не влияют на условия

Примеры

`DSB ;` Барьер синхронизации данных.

3.11.5 ISB

Барьер синхронизации инструкции.

Синтаксис

`ISB{cond}`

где; 'cond' Дополнительный код состояния

Действия

ISB действует как барьер синхронизации инструкции. Это сбрасывает конвейер процессора, так, чтобы все инструкции после ISB были выбраны от кэша или памяти снова, после того, как инструкция ISB была завершена.

Условия

Указания не влияют на условия

Примеры

`ISB` ; Барьер синхронизации инструкции.

3.11.6 MRS

Переместите содержание специального регистра к регистру общего назначения.

Синтаксис

`MRS{cond} Rd, spec_reg`

где;

- 'cond' Дополнительный код состояния
- 'Rd' целевой регистр
- 'spec_reg' может быть: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK, или CONTROL.

Действия

Используйте MRS в сочетании с MSR, поскольку часть последовательности "чтение изменяет запись" для обновления PSR, например чтобы очистить флаг Q. В коде подкачки процесса состояние модели программистов выгружаемого процесса должно быть сохранено, включая соответствующее содержание PSR. Точно так же состояние загружаемого процесса должно также быть восстановлено. Эти операции используют MRS в сохраняющей состоянии последовательности инструкции и MSR в восстанавливающей состоянии последовательности инструкции.

BASEPRI_MAX - псевдоним BASEPRI, когда используется с инструкцией MRS.

Ограничения

Rd не должен быть SP либо PC.

Условия

Указания не влияют на условия

Примеры



MRS R0, PRIMASK ; прочитайте значение PRIMASK и запишите его значение в R0

3.11.7 MSR

Перемещение содержания регистра общего назначения в указанный специальный регистр.

Синтаксис

MSR{*cond*} *spec_reg*, *Rn*

где

- '*cond*' Дополнительный код состояния
- '*Rn*' исходный регистр.
- '*spec_reg*' может быть: APSR, IPSR, EPSR, IEPSR, IAPSR, EAPSR, PSR, MSP, PSP, PRIMASK, BASEPRI, BASEPRI_MAX, FAULTMASK, или CONTROL.

действия

доступ к регистру в MSR зависит от уровня полномочий. Непривилегированное программное обеспечение может только получить доступ к APSR, видеть Таблицу 5: APSR укусил определения на странице 20.

В непривилегированных записях программного обеспечения к битам освобожденного или режима выполнения в PSR проигнорированы.

-
- *Rn* ненулевое, и текущее значение BASEPRI 0
Rn ненулевое и меньше, чем текущее значение BASEPRI.

Ограничения

Rn не должен быть SP либо PC.

Условия

Эта инструкция обновляет условия на основе значения *Rn*.

Примеры

MSR CONTROL, R1 ; Считайте значение R1 и запишите его в Регистр команд

3.11.8 NOP

Бездействие

Синтаксис

`NOP{cond}`

где:

- 'cond' Дополнительный код состояния

Действия

бездействие только для указанных целей .. Процессор мог бы удалить его из конвейера, прежде чем это достигнет стадии выполнения.

Используйте NOP для дополнения, например чтобы поместить следующие инструкции в 64-разрядную границу.

Условия

Указания не влияют на условия

Примеры

`NOP` ; Бездействие

3.11.9 SEV

Отправка события

Синтаксис

`SEV{cond}`

где:

- 'cond' Дополнительный код состояния

действия

SEV - инструкция подсказки, которая заставляет событие быть сообщенным ко всем процессорам в многопроцессорной системе. Это также устанавливает локальный регистр события в 1.

Условия

Указания не влияют на условия

Примеры

`SEV` ; Отправка события

3.11.10 SVC

Вызов оператора

Синтаксис

`SVC{cond} #imm`

где;

- 'cond' Дополнительный код состояния
- 'imm' оценка выражения к целому числу в диапазоне 0-255 (8-разрядное значение).

Действия

Инструкция SVC вызывает исключение SVC. imm проигнорирован процессором. При необходимости это может быть получено обработчиком исключений, чтобы определить, какую службу требуют.

Условия

Указания не влияют на условия

Примеры

`SVC 0x32` ; Вызов оператора (обработчик может извлечь непосредственное значение, определив местоположение его через ПК),

3.11.11 WFE

Ожидайте События. WFE - инструкция подсказки.

Синтаксис

`WFE{cond}`

Где 'cond' Дополнительный код состояния

Действия

Если регистр события 0, WFE приостанавливает выполнение.

- Исключение, если не замаскировано маской исключения регистрируется или текущий приоритетный уровень
- Запрос Записи Отладки, если Отладка включена
- Событие, сообщенное периферийным устройством или другим процессором в многопроцессорной системе, используя инструкцию SEV.

Если регистр события равняется 1, WFE очищает его к 0 и сразу возвращается.

Условия

Указания не влияют на условия.

Примеры

`WFE` ; Ожидание события

3.11.12 WFI

Ожидание прерывания

Синтаксис

`WFI { cond }`

где:

- 'cond' Дополнительный код состояния

Действия

WFI - инструкция подсказки, которая приостанавливает выполнение

- Исключение
- Запрос Записи Отладки, независимо от того, включена ли Отладка.

Условия

Указания не влияют на условия.

Примеры

`WFI ; Ожидание прерывания`

4 Базовые периферийные устройства

4.1 О периферии STM32 Cortex-M4

Адресс	Устройство	Местоположение
0xE000E010-0xE000E01F	Системный таймер	Table 55 на странице 235
0xE000E100-0xE000E4EF	Вложенный контроллер векторного прерывания	Table 49 на странице 204
0xE000ED00-0xE000ED3F	Системный блок управления	Table 53 на странице 228
0xE000ED88-0xE000ED8B	Управление доступом сопроцессора сопроцессора для операций с плавающей точкой	Table 56 на странице 236
0xE000ED90-0xE000EDB8	Модуль защиты памяти	Table 44 на странице 191
0xE000EF00-0xE000EF03	Вложенный контроллер векторного прерывания	Table 49 на странице 204
0xE000EF30-0xE000EF44	Сопроцессор для операций с плавающей точкой	Table 56 на странице 236

В описаниях регистра,

- Тип регистра описан следующим образом:
 - RW: Чтение и запись
 - RO: Только чтение
 - WO: Только запись
- Требуемое полномочие дает уровень полномочий, требуемый получить доступ к регистру, следующим образом:
 - Привилегированный: Только привилегированное программное обеспечение может получить доступ к регистру.
 - Непривилегированный: И непривилегированное и привилегированное программное обеспечение может получить доступ к регистру.

4.2 Модуль защиты памяти (MPU)

В этом разделе описываются Модуль защиты памяти (MPU), который реализован в некоторых микроконтроллерах STM32. Обратитесь к соответствующей таблице данных устройства, чтобы видеть, присутствует ли MPU в типе STM32, Вы используете.

MPU делит карту памяти на многие области и определяет расположение, размер, права доступа и атрибуты памяти каждой области. Он поддерживает:

- Независимые параметры настройки атрибута для каждой области
- Перекрывающиеся области
- Экспорт памяти приписывает системе.

Атрибуты памяти влияют на поведение доступов памяти к области.

- Восемь отдельных областей памяти, 0-7
- Фоновая область памяти.

Когда области памяти накладываются, доступ к памяти затронут атрибутами области с самым большим количеством. Например, атрибуты для области 7 имеют приоритет по атрибутам любой области, которая перекрывает область 7.

Фоновая область памяти имеет те же атрибуты доступа к памяти как карта памяти по умолчанию, но доступна из привилегированного программного обеспечения

The Cortex-M4 MPU карта памяти объединена. Это означает доступы инструкции, и у доступов данных есть те же параметры настройки области.

Если программа получает доступ к ячейке памяти, которая запрещена MPU, процессор генерирует отказ управления памятью. Это вызывает исключение отказа и могло бы вызвать завершение процесса в среде ОС.

В среде ОС ядро может обновить область MPU, устанавливающую динамично на основе процесса, который будет выполняться. Как правило, встроенный ОС использует MPU для защиты памяти.

Конфигурация областей MPU основывается на типах памяти,

Таблица 38. Сводка атрибутов памяти

Тип памяти	разделение	Другие атрибуты	Описание
Строго упорядоченный	-	-	Все доступы к Строго упорядоченной памяти происходят в порядке программы. Все Строго упорядоченные области, как предполагается, совместно используются.
Устройство	Доступный	-	Периферийные устройства с отображенной памятью это несколько долей процессоров.
	Недоступный	-	Периферийные устройства с отображенной памятью, которые использует только единственный процессор.
Normal	Доступный	Некэшируемый Запишите - через кэшируемую кэшируемую обратную запись	Нормальная память, которая совместно использована несколькими процессорами.
	недоступный	Некэшируемый Запишите - через кэшируемую кэшируемую обратную запись	Нормальная память используемая одним процессором

4.2.1 MPU Атрибуты разрешения доступа

В этом разделе описываются атрибуты права доступа MPU. Биты права доступа, TEX, C, B, S, AP, и XN, регистра MPU_RASR, управляют доступом к соответствующей области памяти. Если доступ сделан к области памяти без требуемых полномочий, то MPU генерирует отказ разрешения.

Таблица 39. Шифрование TEX, C, B, и S

TEX	C	B	S	Тип памяти	совместное пользование	Другие атрибуты	
b000	0	0	x ⁽¹⁾	Строго упорядоченный	+	-	
		1	x ⁽¹⁾	Устройство	+	-	
	1	0	0	Нормально	-	Внешняя и внутренняя запись	
					1		+
		1	0	Нормально	-	Внешняя и внутренняя обратная запись	
					1		+
b001	0	0	0	Нормально	-	Внешний и внутренний некаэшируемый.	
			1		+		
		1	x ⁽¹⁾	Реверсивное шифрование		-	
	1	0	x ⁽¹⁾	Реализация определяет атрибуты.		-	
		1	1	0	Нормально	-	Внешняя и внутренняя обратная запись
						1	
b010	0	0	x ⁽¹⁾	Устройство	-	Несовместно используемое устройство.	
		1	x ⁽¹⁾	Зарезервированное кодирование			-
	1	x ⁽¹⁾	x ⁽¹⁾	Зарезервированное кодирование		-	
b1BB	A	A	0	Нормально	-	Кэшируемая память	
			1		+		

1. MPU игнорирует значение fo этого бита.

2. Посмотрите Таблицу 40 для кодирования битов BB и AA.

Таблица 40 показывает, что стандарты кэша для кодировок атрибута памяти со значением TEX находится в диапазоне 4-7.

Таблица 40. Стандарты кэша для кодирования атрибута памяти

Шифрование, AA или BB	Соответствующие стандарты кэша
00	Некашируемый
01	Запись назад, выделение записи и чтения
10	Запись сквозь, без выделения записи
11	Запись назад, без выделения записи

Таблица 41 показывает кодировки AP, которые определяют права доступа для привилегированного и непривилегированного программного обеспечения.

Таблица 41. Шифрование AP

AP[2:0]	Привилегированные полномочия	Непривилегированные полномочия	Описание
000	Нет доступа	Нет доступа	Все доступы генерируют отказ разрешения
001	RW	Нет доступа	Доступ только из привилегированного программного обеспечения
010	RW	RO	Записи непривилегированным программным обеспечением генерируют отказ разрешения
011	RW	RW	Полный доступ
100	Непредсказуемо	Непредсказуемо	Зарезервирован
101	RO	Нет доступа	Чтения только привилегированным программным обеспечением
110	RO	RO	Только для чтения, привилегированным или непривилегированным программным обеспечением
111	RO	RO	Только для чтения, привилегированным или непривилегированным программным обеспечением

4.2.2 Несоответствие MPU

Когда доступ нарушает полномочия MPU, процессор генерирует отказ управления памятью, посмотрите Раздел 2.1.4: Исключения и прерывания на странице 25. MMFSR указывает причину отказа. Посмотрите Раздел 4.4.15: адресный регистр отказа управления памятью (MMFAR) на странице 226 для получения дополнительной информации.

4.2.3 Обновление области MPU

Чтобы обновить атрибуты для области MPU, обновите MPU_RNR, MPU_RBAR и регистры MPU_RASR. Вы можете программировать каждый регистр отдельно или использовать запись многократного слова, чтобы программировать все эти регистры. Вы можете использовать MPU_RBAR и псевдонимы MPU_RASR, чтобы программировать до четырех областей, одновременно используя инструкцию STM. Обновление области MPU, используя отдельные слова

Простой код, чтобы сконфигурировать одну область:

```

; R1 = Нмер области
; R2 = размер/включить
; R3 = Атрибуты
; R4 = Адрес
LDR R0,=MPU_RNR ; 0xE000ED98, Регистр числа области MPU
STR R1, [R0, #0x0] ; Нмер области
STR R4, [R0, #0x4] ; Базовый адрес области
STRH R2, [R0, #0x8] ; Размер области и включение
STRH R3, [R0, #0xA] ; атрибут области

```

Отключите область прежде, чем записать новые параметры настройки области в MPU, если Вы ранее включили изменяемую область. Например:

```

; R1 = Нмер области

```

```

; R2 =размер/включить
; R3 = Атрибуты
; R4 = Адрес

LDR R0, =MPU_RNR ; 0xE000ED98, Регистр числа области MPU
STR R1, [R0, #0x0] ; Номер области
BIC R2, R2, #1 ; Отключение
STRH R2, [R0, #0x8] ; Размер области и включение
STR R4, [R0, #0x4] ; Базовый адрес области
STRH R3, [R0, #0xA] ; Атрибут области
ORR R2, #1 ; Включение
STRH R2, [R0, #0x8] ; Размер области и включение

```

Программное обеспечение должно использовать инструкции барьера памяти:

- Прежде чем MPU устанавливаются, если могли бы быть выдающиеся передачи памяти, такие как буферизованные записи, которые могли бы быть затронуты изменением в параметрах настройки MPU
- После того, как MPU устанавливаются, если это включает передачи памяти, которые должны использовать новые параметры настройки MPU.

Однако инструкции барьера памяти не требуются, если процесс установки MPU запускается, вводя обработчик исключений или сопровождается возвратом исключения, потому что запись исключения и исключение возвращают поведение барьера памяти причины механизма.

Для программного обеспечения не нужны никакие инструкции барьера памяти во время установки MPU, потому что это получает доступ к MPU через PPB, который является Строго упорядоченной областью памяти.

Например, если Вы хотите, чтобы все поведение доступа к памяти сразу вступило в силу после последовательности программирования, используйте инструкцию DSB и инструкцию ISB:

- DSB требуется после изменения параметров настройки MPU, такой как в конце контекстного переключения.
- ISB требуется, если код, который программирует область MPU или области, введен, используя ответвление или вызов. Если последовательность программирования введена, используя возврат из исключения, или беря исключение, то Вы не требуете ISB.

Обновление области MPU, используя многословные записи

Вы можете программировать непосредственно использующие записи многословные, в зависимости от того, как информация разделена. Рассмотрите следующее перепрограммирование:

```

; R1 = Номер области
; R2 = адрес
; R3 = размер, атрибуты в одном

LDR R0, =MPU_RNR ; 0xE000ED98, Регистр числа области MPU
STR R1, [R0, #0x0] ; Номер области
STR R2, [R0, #0x4] ; Базовый адрес области
STR R3, [R0, #0x8] ; Атрибут области, размер и включение

```

Используйте инструкцию STM, чтобы оптимизировать это:

```

; R1 = Номер области
; R2 = адрес
; R3 = размер, атрибуты в одном

LDR R0, =MPU_RNR ; 0xE000ED98, Регистр числа области MPU
STM R0, {R1-R3} ; Атрибут области, размер и включение

```

Вы можете сделать это в двух словах для предварительно упакованной информации. Это означает, что RBAR содержит требуемое число области и имел ДОПУСТИМЫЙ набор битов к 1,

```

; R1 = адрес и число области в одном
; R2 =размер и атрибуты в одном
LDR R0, =MPU_RBAR ; 0xE000ED9C, Базовый регистр Области MPU
STR R1, [R0, #0x0] ;Базовый адрес области и число области,
объединенное с ДОПУСТИМЫМ (укусил 4), набор к 1
STR R2, [R0, #0x4] ; Атрибут области, размер и включают

```

Используйте инструкцию STM, для оптимизации:

```

; R1 = адрес и число области в одном
; R2 = размер и атрибуты в одном
LDR R0, =MPU_RBAR ; 0xE000ED9C, Базовый регистр Области MPU
STM R0, {R1-R2} ; Базовый адрес области, число области и ДОПУСТИМЫЙ бит, и
Атрибут Области, Измерение и Включение

```

Подобласти

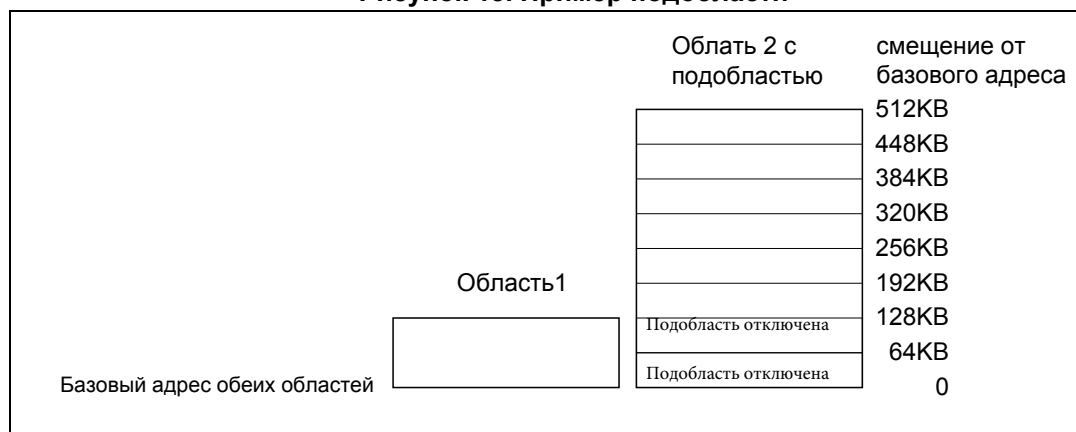
Области 256 байтов или больше разделено на восемь подобластей равного размера. Установите соответствующий бит в поле SRD RASR. Младший значащий бит SRD управляет первой подобластью, и старший значащий бит управляет последней подобластью. Отключение подобласти означает другую область, перекрывающую отключенные соответствия диапазона вместо этого. Если никакая другая включенная область не перекрывает отключенную подобласть, MPU выпускает отказ.

Области 32, 64, и 128 байтов не поддерживают подобласти. С областями этих размеров, Вы должны установить поле SRD в 0x00, иначе поведение MPU Непредсказуемо.

Пример использования SRD:

Две области с тем же перекрытием базового адреса. Область каждый - 128 КБ, и область два, составляет 512 КБ. Чтобы гарантировать атрибуты от области, каждый применяется к first128KB области, устанавливал поле SRD для области два к b00000011 отключать первые две подобласти, поскольку данные показывают.

Рисунок 18. Пример подобласти



4.2.4 Полезные советы для проектирования MPU

Чтобы избежать неожиданного поведения, отключите прерывания прежде, чем обновить атрибуты области, к которой могли бы получить доступ обработчики прерываний. Гарантируйте использованию программного обеспечения выровненные доступы корректного размера к доступу регистры MPU:

- За исключением RASR, использовать выровненные доступы слова
- Для RASR использовать байт или выровненное полуслово или доступы слова.

Процессор не поддерживает невыровненные доступы к регистрам MPU.

Во время установки MPU, и если MPU был ранее запрограммирован, отключает неиспользованные области, чтобы препятствовать тому, чтобы любые предыдущие параметры настройки области влияли на новую установку MPU.

Рекомендуемая конфигурация MPU

У системы микроконтроллера STM32 есть только единственный процессор, таким образом, Вы должны запрограммировать MPU следующим образом:

Таблица 42. Область памяти для STM32

Области памяти	TEX	C	B	S	Тип памяти и свойства
FLASH память	b000	1	0	0	Нормальная память, Необщая, запись - через
Внутренняя SRAM	b000	1	0	1	Нормальная память, Общая, запись - через
Внешняя SRAM	b000	1	1	1	Нормальная память, Общая запись -назад
Периферия	b000	0	1	1	Память устройства, Общая

В реализациях STM32 shareability и стратегические атрибуты кэша не влияют на поведение системы. Однако использование этих параметров настройки для областей MPU может сделать код приложения более переносимым. Данные значения для типичных ситуаций.

Заметка: Атрибуты MPU не влияют на доступы данных DMA к адресным пространствам памяти/периферийных устройств. поэтому, чтобы защитить области памяти от непреднамеренных доступов DMA, MPU должен управлять доступом SW/ЦП к регистрам DMA.

4.2.5 MPU регистр (MPU_TYPER)

Адрес смещен: 0x00

Значение сброса: 0x0000 0800

Требуемое полномочие: Привилегированный

Регистр MPU_TYPER указывает, присутствует ли MPU, и если так, сколько областей это поддерживает.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
зарезервированный								IREGION[7:0]							
								г	г	г	г	г	г	г	г
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DREGION[7:0]								зарезервированный							Раздельно
г	г	г	г	г	г	г									г

Биты 31:24 **Зарезервированы**

Биты 23:16 **IREGION[7:0]**: Число областей инструкции MPU.

Эти биты указывают число поддерживаемых областей инструкции MPU.

Всегда содержит 0x00. Карта памяти MPU объединена и описана полем DREGION.

Биты 15:8 **DREGION[7:0]**: Число областей данных MPU.

Эти биты указывают число поддерживаемых областей данных MPU.

0x08: Восемь областей MPU

0x00: MPU, не существующий

Биты 7:1 **Зарезервированы**

Бит 0 **ОТДЕЛЬНЫЙ**: Отдельный флаг.

Этот бит указывает поддержку объединенной или отдельной инструкции и карт памяти данных:

0 = Объединенный

1 = Отдельный

4.2.6 Регистр команд MPU (MPU_CTRL) Адрес

смещал: 0x04
Значение сброса: 0x0000 0000
Требуемое полномочие: Привилегированный
Регистр MPU_CTRL:

- Включает MPU
- Включает фоновую область памяти карты памяти по умолчанию
- Включает использование MPU когда в твердом отказе, Немаскируемое прерывание (NMI), и FAULTMASKпередаваемые обработчики.

Когда ВКЛЮЧАЮТ, и PRIVDEFENA оба установлены в 1:

- [Для привилегированных доступов карта памяти по умолчанию как описана в Разделе 2.2:](#) Модель памяти на странице 27. Любой доступ привилегированным программным обеспечением, которое не адресует включенную область памяти, ведет себя, как определено картой памяти по умолчанию.
- Любой доступ непривилегированным программным обеспечением, которое не адресует включенную область памяти вызывает отказ управления памятью.

XN и Строго упорядоченные правила всегда применяются к Системному Пространству Управления независимо от значения ВКЛЮЧАТЬ бита.

То, когда ВКЛЮЧАТЬ бит установлен в 1, по крайней мере одна область карты памяти должна быть позволена для системы функционировать, установлено в 1. Если PRIVDEFENA установлен в 1, и никакие области не включены, то только привилегированное программное обеспечение может работать.

Когда ENABLE бит установлен в 0, система использует карту памяти по умолчанию. У этого есть те же атрибуты памяти, как будто MPU не реализован, посмотрите Таблицу 13: поведение Доступа к памяти на странице 29. Карта памяти по умолчанию применяется к доступам и из привилегированного и из непривилегированного программного обеспечения.

Когда MPU включен, доступы к Системному Пространству Управления и таблице векторов всегда разрешаются. Другие области доступны на основе областей и установлен ли PRIVDEFENA в 1.

Если HFNMIENA не установлен в 1, MPU не включен, когда процессор выполняет обработчик для исключения с приоритетом –1 или –2. Эти приоритеты только возможны при обработке твердого отказа или исключения NMI, или когда FAULTMASK включен. Урегулирование HFNMIENA укусило к 1, включает MPU при работе с этими двумя приоритетами.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Зарезервирован															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Зарезервирован													PRIVDEFENA	HFNMIENA	EN ABLE
													rw	rw	rw



Биты 31:3 Зарезервированы, вызываются аппаратными средствами к 0.

Бит 2 PRIVDEFENA: Включает привилегированный доступ программного обеспечения к карте памяти по умолчанию.

0: Если MPU включен, отключает использование карты памяти по умолчанию. Любой доступ к памяти к расположению, не покрытому любой включенной областью, вызывает отказ.

1: Если MPU включен, включает использование карты памяти по умолчанию как фоновая область памяти для привилегированных доступов программного обеспечения.

Примечание: Когда включено, фоновая область памяти действует, как будто это - область номер-1. Любая область ~ определена и включена, имеет приоритет над этой картой по умолчанию. Если MPU отключен, процессор игнорирует этот бит.

Бит 1 HFNMIENA: Включает работу MPU во время твердого отказа, NMI и обработчиков FAULTMASK.

Когда MPU включен:

0: MPU отключен во время твердого отказа, NMI и обработчиков FAULTMASK, независимо от значения ВКЛЮЧАТЬ бита

1: MPU включен во время твердого отказа, NMI и обработчиков FAULTMASK.

Примечание: Когда MPU отключен, если этот бит установлен в 1, поведение непредсказуемо.

Бит 0 ENABLE: включает MPU

0: MPU отключен

1: MPU включен

4.2.7 Регистр числа области MPU (MPU_RNR)

Адрес смещения: 0x08

Значение сброса: 0x0000 0000

Требуемое полномочие: Привилегированный

Регистр MPU_RNR выбирает, на какую область памяти ссылаются регистры MPU_RASR и MPU_RBAR.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Зарезервирован															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Зарезервирован								REGION[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

Биты 31:8 Зарезервированный, вызванный аппаратными средствами к 0.

Биты 7:0 ОБЛАСТЬ [7:0]: область MPU

Эти биты указывают область MPU, на которую ссылаются регистры MPU_RASR и MPU_RBAR. MPU поддерживает 8 областей памяти, таким образом, разрешенные значения этого поля 0-7. Обычно, Вы пишете требуемое число области в этот регистр прежде, чем получить доступ

MPU_RBAR или MPU_RASR. Однако, Вы можете изменить число области при записи в регистр MPU_RBAR с ДОПУСТИМЫМ набором битов к 1, видеть индексный регистр области MPU (MPU_RBAR). Эта запись обновляет значение поля REGION.

4.2.8 MPU индексный регистр области (MPU_RBAR)

Адрес смещения: 0x0C

Значение сброса: 0x0000 0000

Требуемое полномочие: Привилегированный

Регистр MPU_RBAR определяет базовый адрес области MPU, выбранной регистром MPU_RNR, и может обновить значение регистра MPU_RNR.

Запишите в регистр MPU_RBAR с ДОПУСТИМЫМ набором битов к 1, чтобы изменить текущее число области и обновить регистр MPU_RNR.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
ADDR[31:N]...																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
...ADDR[31:N]											VALID	REGION[3:0]				
											rw	rw	rw	rw	rw	

Биты 31:N **ADDR[31:N]**: Поле базового адреса области

Значение N зависит от размера области.

Размер области, как определено полем SIZE в MPU_RASR, определяет значение N:

$N = \text{Log2}(\text{Размер области в битах})$,

Если размер области сконфигурирован к 4 Гбайт в регистре MPU_RASR, нет никакого допустимого поля ADDR. В этом случае область занимает полную карту памяти, и базовый адрес - 0x00000000.

Базовый адрес выровненный к размеру области. Например, область на 64 Кбайта должна быть выровненная на кратном числе 64 Кбайт, например, в 0x00010000 или 0x00020000.

Биты N-1:5, Зарезервированы, вызванный аппаратными средствами к 0.

Бит 4 **VALID** допустимое число области MPU

апись:

0: Регистр MPU_RNR, не измененный, и процессор:

- Обновляет базовый адрес для области определен в MPU_RNR
- Игнорирует значение поля REGION

1 Процессор

- обновляет значение MPU_RNR к значению поля REGION
- обновляет базовый адрес для области, определенной в поле REGION.

Чтение

Всегда читайте как нуль.

Биты 3:0 **REGION [3:0]**: поле области MPU

Для поведения на записях см. описание поля **VALID**.

На чтениях, возвращает текущее число области, как определено регистром MPU_RNR.

4.2.9 Атрибут области MPU и регистр размера (MPU_RASR)

Адрес смещал: 0x10

Значение сброса: 0x0000 0000

Требуемое полномочие: Привилегированный

Регистр MPU_RASR определяет размер области и атрибуты памяти области MPU, определенной MPU_RNR, и включает ту область и любые подобласти.

MPU_RASR - доступное слово использования или доступы полуслова:

- Старшее значащее полуслово содержит атрибуты области
- Младшее значащее полуслово содержит размер области, и область и подобласть включают биты.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			XN	Reserved	AP[2:0]			Зарезервирован		TEX[2:0]			S	C	B
			rw		rw	rw	rw			rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRD[7:0]								Зарезервирован		SIZE					ENABLE
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

Биты 31:29 Зарезервированы, вызваны аппаратными средствами к 0.

Бит 28 XN: доступ Инструкции отключает бит:

0: Вызовы команды включали

1: Вызовы команды отключены.

Бит 27 Зарезервирован, вызван аппаратными средствами к 0.

Биты 26:24 AP [2:0]: Право доступа

Для получения информации о праве доступа посмотрите Раздел 4: Базовые периферийные устройства Для описания кодирования битов AP относятся к Таблице 41 на странице 181.

Биты 23:22 Зарезервированный, вызванный аппаратными средствами к 0.**Биты 21:19 TEX [2:0]: атрибут памяти***Поскольку описание кодирования битов TEX относится к Таблице 39 на странице 180***Бит 18 S: атрибут Совместно используемой памяти****Бит 17 C: атрибут памяти****Бит 16 B: атрибут памяти**

Биты 15:8 SRD: Подобласть отключает биты.

Для каждого бита в этом поле:

0: соответствующая подобласть включена

1: соответствующая подобласть отключена

Посмотрите Подобласти на странице 183 для получения дополнительной информации.

Размеры области 128 байтов и меньше не поддерживает подобласти. При записи атрибутов для такой области запишите поле SRD как 0x00.

Биты 7:6 Зарезервированный, вызванный аппаратными средствами к 0.

Биты 5:1 РАЗМЕР: Размер области защиты MPU.

Минимальное разрешенное значение 3 (b00010), посмотрите значения полей РАЗМЕРА для получения дополнительной информации.

Бит 0 **ENABLE**: Включает бит области

Значения полей SIZE

Поле SIZE определяет размер области памяти MPU, определенной MPU_RNR register следующим образом:

$$(\text{Размер области в битах}) = 2^{(\text{SIZE}+1)}$$

Самый маленький разрешенный размер области 32B, соответствуя значению РАЗМЕРА 4. Таблица 43 дает значения РАЗМЕРА в качестве примера с соответствующим размером области и значением N в MPU_RBAR.

Table 43. Пример значения поля SIZE

Значение SIZE	Размер области	Значение N ⁽¹⁾	Заметки
b00100 (4)	32B	5	Минимальный разрешенный размер
b01001 (9)	1KB	10	-
b10011 (19)	1MB	20	-
b11101 (29)	1GB	30	-
b11111 (31)	4GB	b01100	Максимальный разрешенный размер

4.2.10 Карта регистров MPU

Таблица 44. Карта регистров MPU

порядок	Регистр	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0x00	MPU_TYPER	Reserved								IREGION[7:0]								DREGION[7:0]								Зарезервировано								SEPARATE			
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0				
0x04	MPU_CTRL	Зарезервировано																												PRIVDEFENA	HFNMIENA	ENABLE					
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x08	MPU_RNR	Зарезервировано																						REGION[7:0]													
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					
0x0C	MPU_RBAR	ADDR[31:N]...																												VALID	REGION[3:0]						
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x10	MPU_RASR	Зарезервировано		XN		Зарезервировано		AP[2:0]		Зарезервировано		TEX[2:0]		S		C		B		SRD[7:0]						Зарезервировано		SIZE				ENABLE					
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x14	MPU_RBAR_A1 ⁽¹⁾	ADDR[31:N]...																												VALID	REGION[3:0]						
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x18	MPU_RASR_A1 ⁽²⁾	Зарезервировано		XN		Зарезервировано		AP[2:0]		Зарезервировано		TEX[2:0]		S		C		B		SRD[7:0]						Зарезервировано		SIZE				ENABLE					
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x1C	MPU_RBAR_A2 ⁽¹⁾	ADDR[31:N]...																												VALID	REGION[3:0]						
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x20	MPU_RASR_A2 ⁽²⁾	Зарезервировано		XN		Зарезервировано		AP[2:0]		Зарезервировано		TEX[2:0]		S		C		B		SRD[7:0]						Reserved		SIZE				ENABLE					
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

Таблица 44. Карта регистров MPU

порядок	Регистр	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1C	MPU_RBAR_A3 ⁽¹⁾	ADDR[31:N]...																										VALID		REGION[3:0]			
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x20	MPU_RASR_A3 ⁽²⁾	Зарезервировано		XN		Зарезервировано		AP[2:0]		Зарезервировано		TEX[2:0]		S		C		B		SRD[7:0]				Зарезервировано		SIZE			ENABLE				
	Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

- 1. Псевдоним регистра MPU_RBAR
- 2. Псевдоним регистра MPU_RASR



4.3 Встроенный контроллер векторного прерывания (NVIC)

В этом разделе описываются встроенный контроллер векторного прерывания(NVIC) и регистры, которые его используют. Поддержки NVIC:

- До 81 прерывания (номер прерывания зависит от типа устройства STM32),
- Программируемый приоритетный уровень 0-15 для каждого прерывания. Более высокий уровень соответствует более низкому приоритету, таким образом, уровень 0 - самый высокий приоритет прерывания
- Уровневое и импульсное обнаружение сигналов прерывания
- Динамическое изменение приоритетов прерываний
- Группировка приоритета оценивает в приоритетные и подприоритетные поля группы
- Объединение в цепочку хвоста прерывания
- Внешнее Немаскируемое прерывание (NMI)

Процессор автоматически складывает свое состояние на записи исключения и не складывает это состояние на выходе исключения без инструкции наверху. Это обеспечивает низкую обработку исключений задержки. Аппаратная реализация регистров NVIC:

Таблица. NVIC сводка регистров

Адрес	Имя	Тип	Требуемые полномочия	Значение сброса	Описание
0xE000E100-0xE000E10B	NVIC_ISER0-NVIC_ISER2	RW	привилегированный	0x00000000	Таблица 4.3.2: набор Прерывания - включает регистры (NVIC_ISERx) на странице 195
0xE000E180-0xE000E18B	NVIC_ICER0-NVIC_ICER2	RW	привилегированный	0x00000000	Таблица 4.3.3: четкое Прерывание - включает регистры (NVIC_ICERx) на странице 196
0xE000E200-0xE000E20B	NVIC_ISPR0-NVIC_ISPR2	RW	привилегированный	0x00000000	Таблица 4.3.4: Прервите ожидающие набор регистры (NVIC_ISPRx) на странице 197
0xE000E280-0xE000E29C	NVIC_ICPR0-NVIC_ICPR2	RW	привилегированный	0x00000000	Таблица 4.3.5: незаконченные регистры (NVIC_ICPRx) на странице 198
0xE000E300-0xE000E31C	NVIC_IABR0-NVIC_IABR2	RW	привилегированный	0x00000000	Таблица 4.3.6: активные разрядные регистры прерывания (NVIC_IABRx) на странице 199
0xE000E400-0xE000E503	NVIC_IPR0-NVIC_IPR20	RW	привилегированный	0x00000000	Таблица 4.3.7: приоритетные регистры Прерывания (NVIC_IPRx) на странице 200
0xE000EF00	STIR	WO	Конфигурируемый	0x00000000	Таблица 4.3.8: триггерный регистр прерываний программного обеспечения (NVIC_STIR) на странице 201

4.3.1

Доступ к NVIC регистрам Cortex-M4 используя CMSIS

Функции CMSIS включают мобильность программного обеспечения между различными процессорами профиля Cortex-M4. Чтобы получить доступ к регистрам NVIC при использовании CMSIS, используйте следующие функции:

Таблица 46. Доступ CMSIS функции NVIC

CMSIS функция)	Описание
void NVIC_EnableIRQ(IRQn_Type IRQn)	Включает прерывание или исключение.
void NVIC_DisableIRQ(IRQn_Type IRQn)	Отключает прерывание или исключение.
void NVIC_SetPendingIRQ(IRQn_Type IRQn)	Устанавливает незаконченное состояние прерывания или исключения к 1.
void NVIC_ClearPendingIRQ(IRQn_Type IRQn)	Очищает незаконченное состояние прерывания или исключения к 0.
uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)	Читает незаконченное состояние прерывания или исключения. Эта функция возвращает ненулевое значение, если незаконченное состояние установлено в 1.
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)	Устанавливает приоритет прерывания или исключения с конфигурируемым приоритетным уровнем к 1.
uint32_t NVIC_GetPriority(IRQn_Type IRQn)	Читает приоритет прерывания или исключения с конфигурируемым приоритетным уровнем. Эта функция возвращает текущий приоритетный уровень.

1. Входной параметр IRQn является числом IRQ,

4.3.2 Набор прерывания - включает регистры
(NVIC_ISERx) смещение Адреса: 0x00 - 0x0B
Значение сброса: 0x0000 0000
Требуемое полномочие: Привилегированный

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETENA[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETENA[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

Биты 31:0 SETENA: набор Прерывания - включает биты.

Запись:

- 0: Без эффекта
- 1: Включает прерывания

Чтение

- 0: Прерывания отключены
- 1: Прерывания включены

Если незаконченное прерывание включено, NVIC активирует прерывание на основе своего приоритета. Если прерывание не включено, утверждая, что его сигнал прерывания изменяет состояние прерывания на ожидание, но NVIC никогда не активирует прерывание, независимо от его приоритета.

4.3.3 Четкое прерывание - включает регистры (NVIC_ICERx)

Адрес смещения: 0x00 -
0x0B значение Сброса: 0x0000 0000 Требуемое полномочие:
Привилегированные регистры ICER0-ICER2 отключают прерывания и
показывают, какие прерывания включены.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLRENA[31:16]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLRENA[15:0]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Биты 31:0 CLRENA: четкое Прерывание - включает биты.
Запись
0; Нет эффекта
1: Отключение прерывания
Чтение
0: Прервите отключено
1: Прерывание включено.



4.3.4 Ожидающие регистры прерывания (NVIC_ISPRx)

Адрес смещения: 0x00 - 0x0B

Значение сброса: 0x0000 0000

Требуемое полномочие: Привилегированный

Регистры ISPR0-ISPR2 вызывают прерывания в незаконченное состояние и показывают, какие прерывания находятся на рассмотрении.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETPEND[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETPEND[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

Биты 31:0 SETPEND: Прервите ожидающие набор биты

Запись

0; Нет эффекта

1: Изменения прерывают состояние к ожиданию

Чтение

0: Прерывание не ожидает

1: Прерывание находится на рассмотрении

Запись 1 к ISPR укусила соответствие прерыванию, которое находится на рассмотрении:

– нет эффекта

Запись 1 к ISPR укусила соответствие заблокированному прерыванию:

– устанавливает состояние того прерывания к ожиданию.

4.3.5 Регистры чистого прерывания (NVIC_ICPRx)

Адрес смещения: 0x00 - 0x0B
Значение сброса: 0x0000 0000
Требуемое полномочие: Привилегированный
Регистры ICPR0-ICPR2 удаляют незаконченное состояние из прерываний и показывают, какие прерывания находятся на рассмотрении.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLRPEND[31:16]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLRPEND[15:0]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Биты 31:0 **CLRPEND**: Биты чистого прерывния

Запись:

- 0: Нет эффекта
- 1: Удаляет незаконченное состояние прерывания

Чтение:

- 0: Прерывание не ожидает
- 1: Прерывание находится на рассмотрении

Запись 1 к ICPR укусила, не влияет на активное состояние соответствующего прерывания.



4.3.6 Активные разрядные регистры прерывания(NVIC_IABRx)

Адрес смещения: 0x00 - 0x0B

Значение сброса: 0x0000 0000

Требуемое полномочие: Привилегированный

Регистры IABR0-IABR2 указывают, какие прерывания активны. Разрядные присвоения:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ACTIVE[31:16]															
г	г	г	г	г	г	г	г	г	г	г	г	г	г	г	г
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACTIVE[15:0]															
г	г	г	г	г	г	г	г	г	г	г	г	г	г	г	г

Биты 31:0 **ACTIVE**:Прерывает активные значения

0 Прерывание неактивно

1: Активное прерывание

Маленькие чтения как 1, если состояние соответствующего прерывания активно или активно и незакончено.

4.3.7 Приоритетные регистры прерывания (NVIC_IPRx)

Адрес смещения: 0x00 - 0x0B

Значение сброса: 0x0000 0000

Требуемое полномочие: Привилегированный

Регистры NVIC_IPR0-IPR80 обеспечивают 8-разрядное приоритетное поле для каждого прерывания. Эти регистры доступны для байта. Каждый регистр содержит четыре приоритетных поля, ту карту к четырем элементам в приоритетном IP массива прерывания CMSIS [0] к IP[67], как показано в рисунке 19.

Рисунок 19. Отображение регистра NVIC_IPRx



Таблица 47. IPR присвоения бита

Биты	имя	Функция
[31:24]	Приоритет, байтовое смещение 3	Каждое приоритетное поле содержит приоритетное значение, 0-255. Чем ниже значение, тем больше приоритет соответствующего прерывания. Процессор реализует только биты [7:4] каждого поля, биты [3:0] чтение как ноль, и проигнорируйте записи.
[23:16]	Приоритет, байтовое смещение 2	
[15:8]	Приоритет, байтовое смещение 1	
[7:0]	Приоритет, байтовое смещение 0	

Посмотрите, что набор Прерывания - включает регистры (NVIC_ISERx) на странице 195 для получения дополнительной информации о приоритетном массиве прерывания, который обеспечивает представление программного обеспечения приоритетов прерывания. Найдите число IPR и байтовое смещение для прерывания N следующим образом:

- Соответствующее число IPR, M, дано $M = N \text{ DIV } 4$
- Байтовое смещение требуемого поля приоритета в этом регистре - $\text{MOD } 4 \text{ N}$, где:
 - байтовое смещение 0 относится, чтобы зарегистрировать биты [7:0]
 - байтовое смещение 1 относится, чтобы зарегистрировать биты [15:8]
 - байтовое смещение 2 относится, чтобы зарегистрировать биты [23:16]
 - байтовое смещение 3 относится, чтобы зарегистрировать биты [31:24].

4.3.8 Триггерный регистр прерываний программного обеспечения (NVIC_STIR)

Адрес смещения:: 0xE00

Reset value: 0x0000 0000

Требуемое полномочие: То, когда USERSETMPEND укусил в SCR, установлено в 1, непривилегированное программное обеспечение может получить доступ к STIR., видеть Раздел 4.4.6: системный регистр команд (SCR). Только привилегированное программное обеспечение может включить непривилегированный доступ к STIR.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Зарезервировано															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Зарезервировано							INTID[8:0]								
							w	w	w	w	w	w	w	w	w

Биты 31:9 Зарезервированы, должны быть сохранены очищенными.

Bits 8:0 INTID Software generated interrupt ID

Биты 31:9 Зарезервированы, должны быть записаны к STIR., чтобы генерировать (SGI). Значение, которое будет записано, является ID Прерывания требуемого SGI в диапазоне 0-239. Например, значение 0x03 определяет прерывание IRQ3.cleared.

4.3.9 Чувствительные к уровню и импульсные прерывания

Чувствительные к уровню и импульсные прерывания interruptSTM32 и чувствительны к уровню и чувствительны к импульсу. Импульсные прерывания также описаны как запускаемые фронтом прерывания. ts

Чувствительное к уровню прерывание считается утверждаемым до периферийного устройства deasserts сигнал прерывания. Обычно это происходит, потому что ISR получает доступ к периферийному устройству, заставляя его очистить запрос на прерывание. Импульсное прерывание - сигнал прерывания, выбранный синхронно на нарастающем фронте часов процессора. Гарантировать NVIC обнаруживает прерывание, периферийное устройство должно утверждать сигнал прерывания по крайней мере для одного такта, во время которого NVIC обнаруживает импульс и фиксирует прерывание.

Когда процессор вводит ISR, он автоматически удаляет незаконченное состояние из прерывания, посмотрите Аппаратное и программное управление прерываний. Для чувствительного к уровню прерывания, если сигнал не deasserted перед возвратами процессора из ISR, прерывание становится ожиданием снова, и процессор должен выполнить свой ISR снова. Это означает, что периферийное устройство может содержать сигнал прерывания, утверждаемый, пока этому больше не нужно обслуживание.

Аппаратное и программное управление прерываний

Cortex-M4 фиксирует все прерывания. Прерывание ввода-вывода становится ожиданием по одной из следующих причин:

- NVIC обнаруживает, что сигнал прерывания BYCOK, и прерывание не активно
- NVIC обнаруживает нарастающий фронт на сигнале прерывания
- Программное обеспечение пишет в соответствующий бит регистра ожидания набора прерывания, посмотрите

Незаконченное прерывание остается ожидать до одного из следующего:

- Процессор вводит ISR для прерывания. Это изменяет состояние прерывания от ожидания до активного. Тогда:
 - Для чувствительного к уровню прерывания, когда процессор возвращается из ISR, NVIC выбирает сигнал прерывания. Если сигнал утверждается, состояние изменений прерывания в ожидании, которое могло бы заставить процессор сразу повторно входить в ISR. Иначе, состояние прерывания изменяется на неактивный.
 - Для импульсного прерывания NVIC продолжает контролировать сигнал прерывания, и если это пульсируется состояние изменений прерывания в ожидании и активный. В этом случае, когда процессор возвращает из ISR состояние изменений прерывания в ожидании, которое могло бы заставить процессор сразу повторно входить в ISR. Если сигнал прерывания не пульсируется, в то время как процессор находится в ISR, когда процессор возвращает из ISR состояние изменений прерывания в неактивном.
- Программное обеспечение пишет в соответствующий бит регистра четкого ожидания прерывания.

Для чувствительного к уровню прерывания, если сигнал прерывания все еще утверждается, не изменяется состояние прерывания. Иначе, состояние прерывания изменяется на неактивный.

Для импульсного прерывания состояние прерывания изменяется на:

 - Неактивный, если состояние находилось на рассмотрении
 - Активный, если состояние было активно и незакончено.