

Таблица 21. Инструкции коры-M4 (продолжение)

Мнемосхема	Операнды	Краткое описание	Флаги	Страница
LDREXB	Rt, [Rn]	Регистр загрузки, монопольный с байтом	—	3.4.8 на странице 78
LDREXH	Rt, [Rn]	Регистр загрузки, монопольный с полусловом	—	3.4.8 на странице 78
LDRH, LDRHT	Rt, [Rn, #offset]	Регистр загрузки с полусловом	—	3.4 на странице 68
LDRSB, LDRSBT	Rt, [Rn, #offset]	Регистр загрузки с байтом со знаком	—	3.4 на странице 68
LDRSH, LDRSHT	Rt, [Rn, #offset]	Регистр загрузки с полусловом со знаком	—	3.4 на странице 68
LDRT	Rt, [Rn, #offset]	Регистр загрузки со словом	—	3.4 на странице 68
LSL, LSLs	Rd, Rm, <Rs #n>	Логический сдвиг уехал	N,Z,C	3.5.3 на странице 85
LSR, LSRs	Rd, Rm, <Rs #n>	Право логического сдвига	N,Z,C	3.5.3 на странице 85
MLA	Rd, Rn, Rm, Ra	умножители накапливаются, результат на 32-биты	—	3.6.1 на странице 109
MLS	Rd, Rn, Rm, Ra	Умножьте и вычитите, 32-разрядный результат	—	3.6.1 на странице 109
MOV, MOVs	Rd, Op2	Переместиться	N,Z,C	3.5.6 на странице 88
MOVT	Rd, #imm16	Переместите вершину	—	3.5.7 на странице 90
MOVW, MOV	Rd, #imm16	Переместите 16-разрядную константу	N,Z,C	3.5.6 на странице 88
MRS	Rd, spec_reg	Переместитесь от специального регистра до общего регистра	—	3.11.6 на странице 173
MSR	spec_reg, Rm	Переместитесь от общего регистра до специального	N,Z,C,V	3.11.7 на странице 174
MUL, MULS	{Rd,} Rn, Rm	Умножьтесь, 32-разрядный результат	N,Z	3.6.1 на странице 109
MVN, MVNS	Rd, Op2	Переместитесь НЕТ	N,Z,C	3.5.6 на странице 88
NOP	—	Никакая работа	—	3.11.8 на странице 175
ORN, ORNS	{Rd,} Rn, Op2	Логичный ИЛИ НЕТ	N,Z,C	3.5.2 на странице 84
ORR, ORRS	{Rd,} Rn, Op2	Логичный ИЛИ	N,Z,C	3.5.2 на странице 84
PKHTB, PKHBT	{Rd,} Rn, Rm, Op2	Упакуйте Полуслово		3.8.1 на странице 133
POP	reglist	Вытолкайте регистры от стека	—	3.4.7 на странице 77
PUSH	reglist	Продвиньте регистры на стек	—	3.4.7 на странице 77
QADD	{Rd,} Rn, Rm	Двойное насыщение и добавляет		3.7.3 на странице 127
QADD16	{Rd,} Rn, Rm	Насыщение добавляет 16		3.7.3 на странице 127
QADD8	{Rd,} Rn, Rm	Насыщение добавляет 8		3.7.3 на странице 127
QASX	{Rd,} Rn, Rm	Насыщение добавляет и вычитает с обменом		3.7.4 на странице 128

Таблица 21. Инструкции коры-M4 (продолжение)

Мнемосхема	Операнды	Краткое описание	Флаги	Страница
QDADD	{Rd,} Rn, Rm	Насыщение добавляет		3.7.5 на странице 129
QDSUB	{Rd,} Rn, Rm	Двойное насыщение и вычитает		3.7.5 на странице 129
QSAX	{Rd,} Rn, Rm	Насыщение вычитает и добавляет с обменом		3.7.4 на странице 128
QSUB	{Rd,} Rn, Rm	Насыщение вычитает		3.7.3 на странице 127
QSUB16	{Rd,} Rn, Rm	Насыщение вычитает 16		3.7.4 на странице 128
QSUB8	{Rd,} Rn, Rm	Насыщение вычитает 8		3.7.4 на странице 128
RBIT	Rd, Rn	Обратные биты	—	3.7.4 на странице 128
REV	Rd, Rn	Обратный порядок байтов одним словом	—	3.5.8 на странице 91
REV16	Rd, Rn	Обратный порядок байтов в каждом полуслове	—	3.5.8 на странице 91
REVSH	Rd, Rn	Обратный порядок байтов в нижнем полуслове и знаке расширяется	—	3.5.8 на странице 91
ROR, RORS	Rd, Rm, <Rsl#n>	Поверните право	N,Z,C	3.5.3 на странице 85
RRX, RRXS	Rd, Rm	Вращайтесь прямо с, расширяются	N,Z,C	3.5.3 на странице 85
RSB, RSBS	{Rd,} Rn, Op2	Реверс вычитает	N,Z,C,V	3.5.1 on page 82
SADD16	{Rd,} Rn, Rm	Подписанный добавляют 16		3.5.9 на странице 92
SADD8	{Rd,} Rn, Rm	Подписанный добавляют 8		3.5.9 на странице 92
SASX	{Rd,} Rn, Rm	Подписанный добавляют и вычитают с обменом		3.5.14 на странице 97
SBC, SBCS	{Rd,} Rn, Op2	Вычитите с переносом	N,Z,C,V	3.5.1 на странице 82
SBFX	Rd, Rn, #lsb, #width	Извлечение битового поля со знаком	—	3.9.2 на странице 138
SDIV	{Rd,} Rn, Rm	Подписанный делятся	—	3.6.3 на странице 111
SEV	—	Отправьте событие	—	3.11.9 на странице 175
SHADD16	{Rd,} Rn, Rm	Деление на два со знаком добавляет 16	—	3.5.10 на странице 93
SHADD8	{Rd,} Rn, Rm	Деление на два со знаком добавляет 8	—	3.5.10 на странице 93
SHASX	{Rd,} Rn, Rm	Деление на два со знаком добавляет и вычитает с обменом	—	3.5.11 на странице 94
SHSAX	{Rd,} Rn, Rm	Деление на два со знаком вычитает и добавляет с обменом	—	3.5.11 на странице 94
SHSUB16	{Rd,} Rn, Rm	Деление на два со знаком вычитает 16	—	3.5.12 на странице 95
SHSUB8	{Rd,} Rn, Rm	Деление на два со знаком вычитает 8	—	3.5.12 на странице 95
SMLABB, SMLABT, SMLATB, SMLATT	Rd, Rn, Rm, Ra	Подписанный умножаются, накапливаются длинный (полуслова)	Q	3.6.3 на странице 111
SMLAD, SMLADX	Rd, Rn, Rm, Ra	Подписанный умножаются, накапливаются двойной	Q	3.6.4 на странице 113

Таблица 21. Инструкции коры-M4 (продолжение)

Мнемосхема	Операнды	Краткое описание	Флаги	Страница
SMLAL	RdLo, RdHi, Rn, Rm	Подписанный умножаются с, накапливаются (32 x 32 + 64), результат на 64-биты	—	3.6.2 на странице 110
SMLALBB, SMLALBT, SMLALTB, SMLALTT	RdLo, RdHi, Rn, Rm	Подписанный умножаются, накапливаются долго, полуслова	—	3.6.5 на странице 114
SMLALD, SMLALDX	RdLo, RdHi, Rn, Rm	Подписанный умножаются, накапливаются долго двойной	—	3.6.5 на странице 114
SMLAWB, SMLAWT	Rd, Rn, Rm, Ra	Подписанный умножаются, накапливаются, слово	Q	3.6.3 на странице 111
SMLS	Rd, Rn, Rm, Ra	Подписанный умножаются, вычитают двойной	Q	3.6.6 на странице 116
SMLS	RdLo, RdHi, Rn, Rm	Подписанный умножаются, вычитают долго двойной	—	3.6.6 на странице 116
SMMLA	Rd, Rn, Rm, Ra	Старшее значащее слово со знаком умножается,	—	3.6.7 на странице 118
SMMLS, SMMLR	Rd, Rn, Rm, Ra	Старшее значащее слово со знаком умножается, вычитают	—	3.6.7 на странице 118
SMMUL, SMMULR	{Rd,} Rn, Rm	Старшее значащее слово со знаком умножается	—	3.6.8 на странице 119
SMUAD	{Rd,} Rn, Rm	Со знаком двойной умножаются, добавляют	Q	3.6.9 на странице 120
SMULBB, SMULBT SMULTB, SMULTT	{Rd,} Rn, Rm	Подписанный умножаются (полуслова)	—	3.6.10 на странице 121
SMULL	RdLo, RdHi, Rn, Rm	Подписанный умножаются (32 x 32), результат на 64-биты	—	3.6.2 на странице 110
SSAT	Rd, #n, Rm {,shift #s}	Подписанный насыщают	Q	3.7.1 на странице 125
SSAT16	Rd, #n, Rm	Подписанный насыщают 16	Q	3.7.2 на странице 126
SSAX	{Rd,} Rn, Rm	Подписанный вычитают и добавляют с обменом	GE	3.5.14 на странице 97
SSUB16	{Rd,} Rn, Rm	Подписанный вычитают 16	—	3.5.13 на странице 96
SSUB8	{Rd,} Rn, Rm	Подписанный вычитают 8	—	3.5.13 на странице 96
STM	Rn{!}, reglist	Сохраните многократные регистры, инкремент после	—	3.4.6 на странице 75
STMDB, STMEA	Rn{!}, reglist	Сохраните многократные регистры, декремент прежде	—	3.4.6 на странице 75
STMFD, STMIA	Rn{!}, reglist	Сохраните многократные регистры, инкремент после	—	3.4.6 на странице 75
STR	Rt, [Rn, #offset]	Слово регистра хранилища	—	3.4 на странице 68
STRB, STRBT	Rt, [Rn, #offset]	Байт регистра хранилища	—	3.4 на странице 68

Таблица 21. Инструкции коры-M4 (продолжение)

Мнемосхема	Операнды	Краткое описание	Флаги	Страница
STRD	Rt, Rt2, [Rn, #offset]	Регистр хранилища два слова	—	3.4.2 на странице 70
STREX	Rd, Rt, [Rn, #offset]	Монопольный регистр хранилища	—	3.4.8 на странице 78
STREXB	Rd, Rt, [Rn]	Регистр хранилища монопольный байт	—	3.4.8 на странице 78
STREXH	Rd, Rt, [Rn]	Регистр хранилища монопольное полуслово	—	3.4.8 на странице 78
STRH, STRHT	Rt, [Rn, #offset]	Полуслово регистра хранилища	—	3.4 на странице 68
STRT	Rt, [Rn, #offset]	Слово регистра хранилища	—	3.4 на странице 68
SUB, SUBS	{Rd,} Rn, Op2	Вычесть	N,Z,C,V	3.5.1 на странице 82
SUB, SUBW	{Rd,} Rn, #imm12	Вычесть	N,Z,C,V	3.5.1 на странице 82
SVC	#imm	Вызов супервизора	—	3.11.10 на странице 176
SXTAB	{Rd,} Rn, Rm,{,ROR #}	Расширитель на 8 битов на 32 и добавьте	—	3.8.3 на странице 135
SXTAB16	{Rd,} Rn, Rm,{,ROR #}	Двойной расширяются на 8 битов на 16 и добавляют	—	3.8.3 на странице 135
SXTAH	{Rd,} Rn, Rm,{,ROR #}	Расширитель на 16 битов на 32 и добавьте	—	3.8.3 на странице 135
SXTB16	{Rd,} Rm {,ROR #n}	Подписанный расширяют байт 16	—	3.8.2 на странице 134
SXTB	{Rd,} Rm {,ROR #n}	Знак расширяет байт	—	3.9.3 на странице 139
SXTH	{Rd,} Rm {,ROR #n}	Знак расширяет полуслово	—	3.9.3 на странице 139
TBB	[Rn, Rm]	Табличный байт ответвления	—	3.9.8 на странице 146
TBH	[Rn, Rm, LSL #1]	Табличное полуслово ответвления	—	3.9.8 на странице 146
TEQ	Rn, Op2	Тестовая эквивалентность	N,Z,C	3.5.9 на странице 92
TST	Rn, Op2	Тест	N,Z,C	3.5.9 на странице 92
UADD16	{Rd,} Rn, Rm	Без знака добавляют 16	GE	3.5.16 на странице 99
UADD8	{Rd,} Rn, Rm	Без знака добавляют 8	GE	3.5.16 на странице 99
USAX	{Rd,} Rn, Rm	Без знака вычитают и добавляют с обменом	GE	3.5.17 на странице 100
UHADD16	{Rd,} Rn, Rm	Деление на два без знака добавляет 16	—	3.5.18 на странице 101
UHADD8	{Rd,} Rn, Rm	Деление на два без знака добавляет 8	—	3.5.18 на странице 101
UHASX	{Rd,} Rn, Rm	Деление на два без знака добавляет и вычитает с обменом	—	3.5.19 на странице 102
UHSAX	{Rd,} Rn, Rm	Деление на два без знака вычитает и добавляет с обменом	—	3.5.19 на странице 102
UHSUB16	{Rd,} Rn, Rm	Деление на два без знака вычитает 16	—	3.5.20 на странице 103
UHSUB8	{Rd,} Rn, Rm	Деление на два без знака вычитает 8	—	3.5.20 на странице 103
UBFX	Rd, Rn, #lsb, #width	Извлечение битового поля без знака	—	3.9.2 на странице 138

Таблица 21. Инструкции коры-M4 (продолжение)

Мнемосхема	Операнды	Краткое описание	Флаги	Страница
UDIV	{Rd,} Rn, Rm	Деление без знака	—	3.6.3 на странице 111
UMAAL	RdLo, RdHi, Rn, Rm	Без знака умножаются, накапливаются, накапливаются долго (32 x 32 + 32 +32), 64-разрядный результат	—	3.6.2 на странице 110
UMLAL	RdLo, RdHi, Rn, Rm	Без знака умножаются с, накапливаются (32 x 32 + 64), результат на 64-биты	—	3.6.2 на странице 110
UMULL	RdLo, RdHi, Rn, Rm	Без знака умножаются (32 x 32), 64-разрядный результат	—	3.6.2 на странице 110
UQADD16	{Rd,} Rn, Rm	Насыщение без знака добавляет 16	—	3.7.7 на странице 131
UQADD8	{Rd,} Rn, Rm	Насыщение без знака добавляет 8	—	3.7.7 на странице 131
UQASX	{Rd,} Rn, Rm	Насыщение без знака добавляет и вычитает с обменом	—	3.7.6 на странице 130
UQSAX	{Rd,} Rn, Rm	Насыщение без знака вычитает и добавляет с обменом	—	3.7.6 на странице 130
UQSUB16	{Rd,} Rn, Rm	Насыщение без знака вычитает 16	—	3.7.7 на странице 131
UQSUB8	{Rd,} Rn, Rm	Насыщение без знака вычитает 8	—	3.7.7 на странице 131
USAD8	{Rd,} Rn, Rm	Сумма без знака абсолютных разностей	—	3.5.22 на странице 105
USADA8	{Rd,} Rn, Rm, Ra	Сумма без знака абсолютных разностей и накапливается	—	3.5.23 на странице 106
USAT	Rd, #n, Rm {,shift #s}	Без знака насыщают	Q	3.7.1 на странице 125
USAT16	Rd, #n, Rm	Без знака насыщают 16	Q	3.7.2 на странице 126
UASX	{Rd,} Rn, Rm	Без знака добавляют и вычитают с обменом	GE	3.5.17 на странице 100
USUB16	{Rd,} Rn, Rm	Без знака вычитают 16	GE	3.5.24 на странице 107
USUB8	{Rd,} Rn, Rm	Без знака вычитают 8	GE	3.5.24 на странице 107
UXTAB	{Rd,} Rn, Rm,{,ROR #}	Вращайтесь, расширитесь на 8 битов на 32 и добавьте	—	3.8.3 на странице 135
UXTAB16	{Rd,} Rn, Rm,{,ROR #}	Вращайтесь, двойной расширяются на 8 битов на 16 и добавляют	—	3.8.3 на странице 135
UXTAH	{Rd,} Rn, Rm,{,ROR #}	Вращайтесь, без знака расширяют и добавляют полуслово	—	3.8.3 на странице 135
UXTB	{Rd,} Rm {,ROR #n}	Нуль расширяет байт	—	3.8.2 на странице 134
UXTB16	{Rd,} Rm {,ROR #n}	Без знака расширяют байт 16	—	3.8.2 на странице 134
UXTH	{Rd,} Rm {,ROR #n}	Нуль расширяет полуслово	—	3.8.2 на странице 134
VABS.F32	Sd, Sm	Абсолют с плавающей точкой	—	3.10.1 на странице 149
VADD.F32	{Sd,} Sn, Sm	С плавающей точкой добавляют	—	3.10.2 на странице 150

Таблица 21. Инструкции коры-M4 (продолжение)

Мнемосхема	Операнды	Краткое описание	Флаги	Страница
VCMPF32	Sd, <Sm #0.0>	Сравните два регистра с плавающей точкой, или один регистр с плавающей точкой и ноль	FPSCR	3.10.3 на странице 150
VCMPE.F32	Sd, <Sm #0.0>	Сравните два регистра с плавающей точкой, или один регистр с плавающей точкой и ноль с Недопустимой проверкой Работы	FPSCR	3.10.3 на странице 150
VCVT.S32.F32	Sd, Sm	Преобразуйте между с плавающей точкой и целочисленным	—	3.10.4 на странице 151
VCVT.S16.F32	Sd, Sd, #fbits	Преобразуйте между с плавающей точкой и фиксированной точкой	—	3.10.4 на странице 151
VCVTR.S32.F32	Sd, Sm	Преобразуйте между с плавающей точкой и целочисленным с округлением	—	3.10.4 на странице 151
VCVT<B H>.F32.F16	Sd, Sm	Преобразовывает значение полуточности в одинарную точность	—	3.10.5 на странице 152
VCVTT<B T>.F32.F16	Sd, Sm	Преобразовывает регистр одинарной точности в полуточность	—	3.10.6 на странице 153
VDIV.F32	{Sd,} Sn, Sm	Деление с плавающей точкой	—	3.10.7 на странице 154
VFMA.F32	{Sd,} Sn, Sm	С плавающей точкой сплавленный умножаются, накапливаются	—	3.10.8 на странице 154
VFNMA.F32	{Sd,} Sn, Sm	С плавающей точкой сплавленный инвертируют, умножаются, накапливаются	—	3.10.9 на странице 155
VFMS.F32	{Sd,} Sn, Sm	С плавающей точкой сплавленный умножаются, вычитают	—	3.10.8 на странице 154
VFNMS.F32	{Sd,} Sn, Sm	С плавающей точкой сплавленный инвертируют, умножаются, вычитают	—	3.10.9 на странице 155
VLDM.F<32 64>	Rn{!}, list	Загрузите многократные регистры расширения	—	3.10.10 на странице 156
VLDR.F<32 64>	<Dd Sd>, [Rn]	Загрузите регистр расширения из памяти	—	3.10.11 на странице 157
VLMA.F32	{Sd,} Sn, Sm	С плавающей точкой умножаются, накапливаются	—	3.10.12 на странице 158
VLMS.F32	{Sd,} Sn, Sm	С плавающей точкой умножаются, вычитают	—	3.10.12 на странице 158
VMOV.F32	Sd, #imm	Непосредственное перемещение с плавающей точкой	—	3.10.13 на странице 158
VMOV	Sd, Sm	Регистр перемещения с плавающей точкой	—	3.10.14 на странице 159
VMOV	Sn, Rt	Ядро ARM копии регистрируется к одинарной точности	—	3.10.18 на странице 162
VMOV	Sm, Sm1, Rt, Rt2	Скопируйте 2 регистра ядра ARM в 2 одинарной точности	—	3.10.17 на странице 161

Таблица 21. Инструкции коры-M4 (продолжение)

Мнемосхема	Операнды	Краткое описание	Флаги	Страница
VMOV	Dd[x], Rt	Ядро ARM копии регистрируется к скаляру	—	3.10.15 на странице 159
VMOV	Rt, Dn[x]	Скопируйте скаляр в регистр ядра ARM	—	3.10.16 на странице 160
VMRS	Rt, FPSCR	Переместите FPSCR в регистр ядра ARM или APSR	N,Z,C,V	3.10.19 на странице 162
VMSR	FPSCR, Rt	Переместитесь в FPSCR от регистра Ядра ARM	FPSCR	3.10.20 на странице 163
VMUL.F32	{Sd,} Sn, Sm	С плавающей точкой умножаются	—	3.10.21 на странице 163
VNEG.F32	Sd, Sm	С плавающей точкой инвертируют	—	3.10.22 на странице 164
VNMLA.F32	Sd, Sn, Sm	С плавающей точкой умножаются и добавляют	—	3.10.23 на странице 165
VNMLS.F32	Sd, Sn, Sm	С плавающей точкой умножают и вычитают	—	3.10.23 на странице 165
VNMUL	{Sd,} Sn, Sm	С плавающей точкой умножаются	—	3.10.23 на странице 165
VPOP	list	Вытолкайте регистры расширения	—	3.10.24 на странице 166
VPUSH	list	Продвиньте регистры расширения	—	3.10.25 на странице 166
VSQRT.F32	Sd, Sm	Вычисляет квадратный корень с плавающей точкой	—	3.10.26 на странице 167
VSTM	Rn{!}, list	Многократное хранилище регистра с плавающей точкой	—	3.10.27 на странице 167
WFE	—	Ожидайте события	—	3.11.11 на странице 176
WFI	—	Ожидайте прерывания	—	3.11.12 на странице 177

3.2 Встроенные функции CMSIS

Код ISO/IEC C не может непосредственно получить доступ к некоторым инструкциям Cortex-M4. Этот раздел описывает встроенные функции, которые могут генерировать эти инструкции, при условии CMSIS и это может быть обеспечено компилятором C. Если компилятор C не поддерживает соответствующую встроенную функцию, Вам, возможно, придется использовать встроенный ассемблер, чтобы получить доступ к некоторым инструкциям. CMSIS обеспечивает встроенные функции, перечисленные в [Таблице 22](#), чтобы генерировать инструкции ANSI к которым нельзя непосредственно получить доступ.

Таблица 22. Встроенные функции CMSIS, чтобы генерировать некоторые инструкции Cortex-M4

Инструкция	Встроенная функция CMSIS
CPSIE I	void __enable_irq(void)
CPSID I	void __disable_irq(void)
CPSIE F	void __enable_fault_irq(void)
CPSID F	void __disable_fault_irq(void)
ISB	void __ISB(void)
DSB	void __DSB(void)
DMB	void __DMB(void)
REV	uint32_t __REV(uint32_t int value)
REV16	uint32_t __REV16(uint32_t int value)
REVSH	uint32_t __REVSH(uint32_t int value)
RBIT	uint32_t __RBIT(uint32_t int value)
SEV	void __SEV(void)
WFE	void __WFE(void)
WFI	void __WFI(void)

CMSIS также обеспечивает много функций для того, чтобы они получили доступ к специальным регистрам, используя MRS и инструкции MSR (см. [Таблицу 23](#)).

Таблица 23. Встроенные функции CMSIS, чтобы получить доступ к специальным регистрам

Специальный регистр	Доступ	Функция CMSIS
PRIMASK	Read	uint32_t __get_PRIMASK(void)
	Write	void __set_PRIMASK(uint32_t value)
FAULTMASK	Read	uint32_t __get_FAULTMASK(void)
	Write	void __set_FAULTMASK(uint32_t value)
BASEPRI	Read	uint32_t __get_BASEPRI(void)
	Write	void __set_BASEPRI(uint32_t value)
CONTROL	Read	uint32_t __get_CONTROL(void)
	Write	void __set_CONTROL(uint32_t value)
MSP	Read	uint32_t __get_MSP(void)
	Write	void __set_MSP(uint32_t TopOfMainStack)
PSP	Read	uint32_t __get_PSP(void)
	Write	void __set_PSP(uint32_t TopOfProcStack)

3.3 Об описаниях инструкции

Следующие разделы дают больше информации об использовании инструкций:

[Операнды на странице 59](#)

[Ограничения при использовании PC или SP на странице 59](#)

[Гибкий второй операнд на странице 59](#)

[Операции сдвига на странице 61](#)

[Выравнивание адреса на странице 64](#)

[Относительные PC выражения на странице 64](#)

[Условное выполнение на странице 64](#)

[Выбор ширины инструкции на странице 67.](#)

3.3.1 Операнды

Операнд инструкции может быть регистром ARM, константой, или другим специфичным для инструкции параметром. Действие инструкций на операндах часто хранит результат в целевом регистре. Когда есть целевой регистр в инструкции, он обычно определяется перед операндами. Операнды в некоторых инструкциях гибки, они могут быть регистром или константой (см. [Гибкий второй операнд](#)).

3.3.2 Ограничения при использовании PC или SP

У многих инструкций есть ограничения на то, можно ли использовать *счетчик программы* (PC) или *указатель вершины стека* (SP) для операндов или целевого регистра. См. описания инструкции для получения дополнительной информации.

Бит [0] из любого адреса, записанного PC с BX, BLX, LDM, LDR, или инструкцией POP, должен быть 1 для корректного выполнения, потому что этот бит указывает на необходимый набор команд, и процессор Cortex-M4 только поддерживает инструкции ползунка.

3.3.3 Гибкий второй операнд

У многих общих инструкций обработки данных есть гибкий второй операнд. Он показывается как *operand2* в описаниях синтаксиса каждой инструкции. *Operand2* может быть:

[Постоянный регистр](#)

[С дополнительным сдвигом](#)

Постоянный

Вы определяете operand2 константу в форме *#constant*, где *константа* может быть:

Любая константа, которая может быть произведена, смещая 8-разрядное значение, оставленное любым числом битов в пределах 32-разрядного слова.

Любая константа формы 0x00XY00XY

любая константа формы 0xXY00XY00

любая константа формы 0xXYXYXYXY

В константах, показанных выше, X и Y, шестнадцатеричные цифры.

Кроме того, в небольшом количестве инструкций, *постоянные*, могут взять более широкий диапазон значений. Они описываются в отдельных описаниях инструкции.

Когда operand2 константа используется с инструкциями MOVs, MVNS, ANDs, ORRs, ORNs, EORs, BICs, TEQ или TST, флаг переноса обновляется к биту [31] из константы, если константа больше чем

255 то может быть произведено, смещение 8-разрядного значения. Эти инструкции не влияют на флаг переноса, если operand2 является какой-либо другой константой.

Замена инструкции

Ваш ассемблер может быть в состоянии произвести эквивалентную инструкцию в случаях где Вы определите константу, которая не разрешается. Например, ассемблер мог бы собрать инструкцию `CMR Rd, #0xFFFFFFFF` как эквивалентную инструкции `CMN Rd, #0x2`.

Регистр с дополнительным сдвигом

Регистр operand2 определяется в *Rm* форме {*сдвиг*}, где:

Rm является регистром, содержащим данные для второго операнда

Сдвиг является дополнительным сдвигом, который будет применен к *Rm*. Это может быть один из:

ASR *#n*: право Арифметического сдвига *n* биты, LSL за £1£n

32 *#n*:

Логический сдвиг оставил *n* биты, LSR за £1£n 31 *#n*:

право Логического сдвига *n* биты, ROR за £1£n 32 *#n*:

Поверните право *n* биты, RRX за £1£n 31:

Поверните правильный один бит, с расширяются-:

Если опущено, никакой сдвиг не происходит, эквивалентный

LSL #0

Если Вы опускаете сдвиг, или определяете LSL #0, инструкция использует значение в *Rm*. Если Вы определяете сдвиг, сдвиг применяется к значению в *Rm*, и получающееся 32-разрядное значение используется инструкцией. Однако, содержание в *Rm* регистра остается неизменным. Определение регистра со сдвигом также обновляет флаг переноса когда используется с определенными инструкциями. Для получения информации об операциях сдвига и как они влияют на флаг переноса, см. [операции Сдвига](#).

3.3.4 Операции сдвига

Операции сдвига регистра перемещают биты в регистр, левый или правый конкретным

количеством битов, *длина сдвига*. Сдвиг регистра может быть выполнен:

Непосредственно инструкциями ASR, LSR, LSL, ROR, и RRX. Результат пишется как целевой регистр.

Во время вычисления operand2 инструкциями, которые определяют второй операнд как регистр со сдвигом (см. [Гибкий второй операнд на странице 59](#)). Результат используется инструкцией.

Разрешенные длины сдвига зависят от типа сдвига и инструкции (см. описание инструкции или [Гибкий второй операнд](#)). Если длина сдвига 0, никакой сдвиг не происходит. Операции сдвига регистра обновляют флаг переноса кроме тех случаев, когда указанная длина сдвига 0.

Следующие подразделы описывают различные операции сдвига и как они влияют на флаг переноса. В этих описаниях *Rm* является регистром, содержащим значение, которое будет смещено, и *n* является длиной сдвига.

ASR

Арифметический сдвиг прямо n битами перемещает левые $32-n$ биты Rm регистра, направо n места, в правые $32-n$ биты результата. И он копирует исходный бит [31] из регистра в левые n биты результата (см. [рисунок 13: ASR#3 на странице 61](#)).

Можно использовать ASR $\#n$ работа, чтобы разделить значение на Rm регистра 2^n , с результатом округленным к отрицательной бесконечности.

Когда инструкция является ASRS или когда ASR $\#n$ используется в operand2 с инструкциями MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST, флаг переноса обновляется к последнему переключенному на верхний регистр биту, бит $[n-1]$, Rm регистра.

Отметьте:

1 Если n 32 или больше, все биты в результате устанавливаются в значение бита [31] из Rm .

2 Если n 32 или больше, и флаг переноса обновляется, это обновляется к значению бита [31] из Rm .



LSR

Логический сдвиг прямо n битами перемещает левые $32-n$ биты Rm регистра, направо n места, в правые $32-n$ биты результата. И это устанавливает левую руку n биты результата к 0 (см. [рисунок 14](#)).

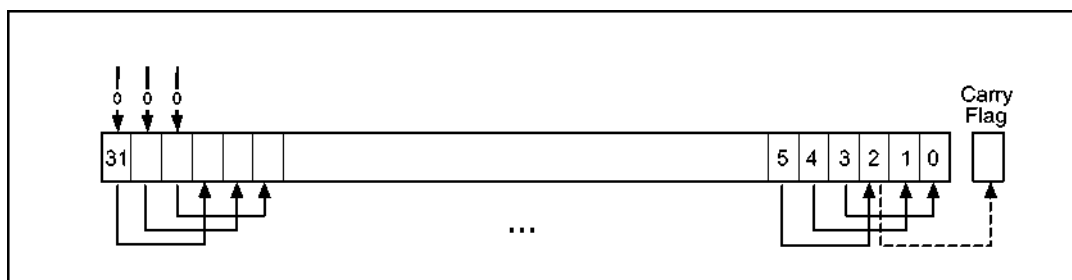
Можно использовать LSR $\#n$ работа, чтобы разделить значение на Rm регистра 2^n , если значение расценено как целое без знака.

Когда инструкция является LSRS или когда LSR $\#n$ используется в operand2 с инструкциями MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST, флаг переноса обновляется к последнему переключенному на верхний регистр биту, бит $[n-1]$, Rm регистра.

Отметьте: 1. Если n 32 или больше, то все биты в результате очищаются к 0.

2. Если n 33 или больше, и флаг переноса обновляется, это обновляется к 0.

Рисунок 14. LSR#3



LSL

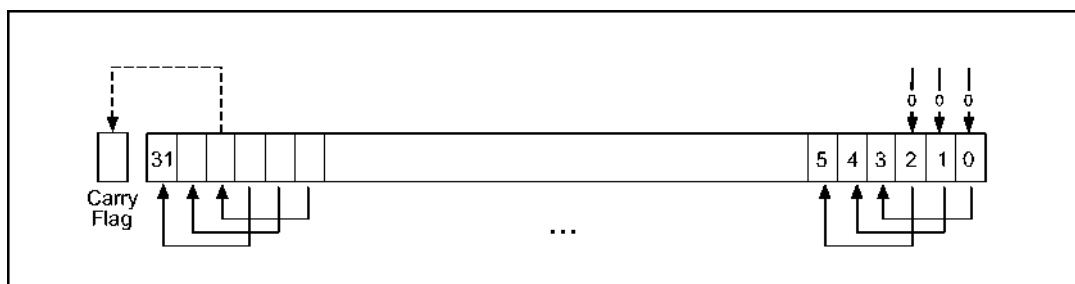
Логический сдвиг, оставленный n битами, перемещает правые $32-n$ биты Rm регистра, налево n места, в левые $32-n$ биты результата. И это устанавливает правую руку n биты результата к 0 (см. [рисунок 15: LSL#3 на странице 62](#)).

Можно использовать LSL $\#n$ работа, чтобы умножить значение в Rm регистра 2^n , если значение расценивается как целое без знака или дополнительное целое число со знаком two. Переполнение может произойти без предупреждения. Когда инструкция является LSLS или когда LSL $\#n$, с ненулевым n , используется в *operand2* с инструкции MOVs, MVNS, ANDs, ORRS, ORNS, EORS, BICS, TEQ или TST, флаг переноса обновляется к последнему переключенному на верхний регистр бит, $[32-n]$ бит, Rm регистра. Эти инструкции не влияют на флаг переноса когда используются с LSL $\#0$.

Отметьте: 1 Если n 32 или больше, то все биты в результате очищаются к 0.

2 Если n 33 или больше, и флаг переноса обновляется, это обновляется к 0.

Рисунок 15. LSL#3



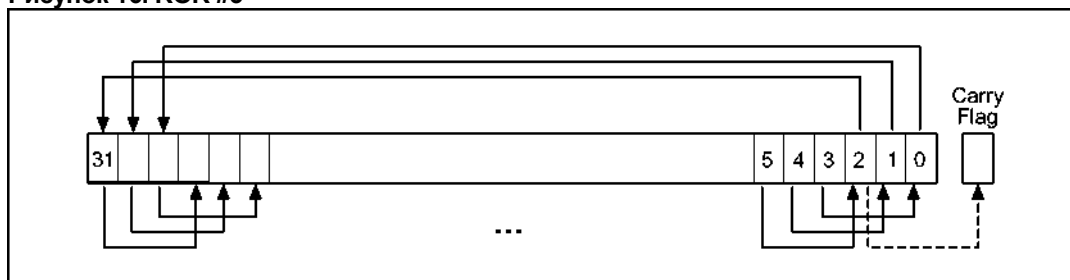
ROR

Поверните прямо на n перемещения битов левые $32-n$ биты Rm регистра, направо местами n , в правые $32-n$ биты результата. Это также перемещает правую руку n биты регистра в левую руку n биты результата (см. [рисунок 16](#)). Когда инструкция является RORS или когда ROR $\#n$ используется в *operand2* с инструкциями MOVs, MVNS, ANDs, ORRS, ORNS, EORS, BICS, TEQ или TST, флаг переноса обновляется к последнему разрядному вращению, бит $[n-1]$, Rm регистра.

Отметьте: 1 Если n 32, то значение результата - то же самое как значение в Rm , и если флаг переноса обновленный, то обновляется к биту $[31]$ из Rm .

2 ROR с длиной сдвига, n , больше чем 32 - то же самое как ROR с длиной сдвига $n-32$.

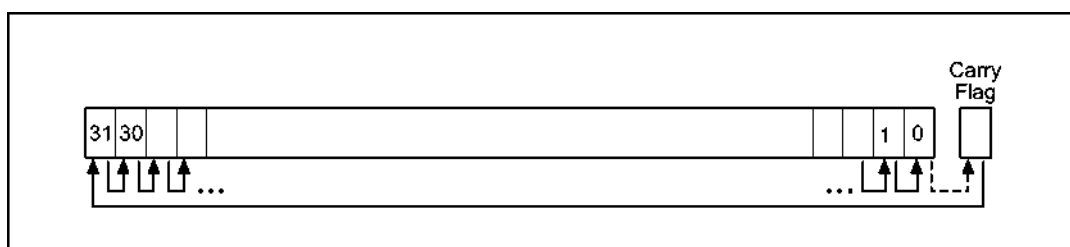
Рисунок 16. ROR #3



RRX

Вращаются прямо с, расширяют перемещения битов *Rm* регистра направо на один бит. И копирует флаг переноса в бит [31] из результата (см. [рисунок 17](#)). Когда инструкция является RRXS или когда RRX используется в operand2 с инструкциями MOVS, MVNS, ANDS, ORRS, ORNS, EORS, BICS, TEQ или TST, флаг переноса обновляется к биту [0] из *Rm* регистра.

Рисунок 17. RRX #3



3.3.5 Выравнивание адреса

Выровненный доступ является работой, где выровненный адрес используется для слова, двойного слова, или многократный доступ слова, или где выровненный адрес используется для доступа полуслова. Доступы байта всегда выровненные.

Процессор Cortex-M4 поддерживает не выровненный доступ только для следующих инструкций:

LDR, LDRT, LDRH, LDRHT, LDRSH, LDRSHT

STR, STRT, STRH, STRHT

Вся другая загрузка и инструкции хранилища генерируют исключение отказа использования, если они выполняют не выровненный доступ, и поэтому их доступы должны быть выровненным адресом. Для получения дополнительной информации об отказах использования см., [Отказ обработки на странице 43](#).

Не выровненные доступы обычно медленней, чем выровненные доступы. Кроме того, некоторая память области не может поддерживать не выровненные доступы. Поэтому, ARM рекомендует, чтобы программисты гарантировали, что доступы выровненные. Чтобы избежать случайной генерации не выровненных доступов, используйте бит UNALIGN_TRP в конфигурации и регистре команд, чтобы захватить все не выровненные доступы, см. [Конфигурацию и регистр команд \(CCR\) на странице 215](#).

3.3.6 Относительные PC выражения

Относительное PC выражение или *метка* являются символом, который представляет адрес инструкции или

литеральные данные. Это представляется в инструкции как значение PC плюс или минус числовое смещение. Ассемблер вычисляет необходимое смещение от метки и адреса текущей команды. Если смещение является слишком большим, ассемблер производит ошибку.

Для B, BL, CBNZ, и инструкций CBZ, значение PC является адресом текущей команды плюс четыре байта.

Для всех других инструкций, которые используют метки, значение PC является адресом текущей команды плюс четыре байта, с битом [1] из результата, очищенного к 0, чтобы сделать это словом - выровненным.

Ваш ассемблер мог бы разрешить другие синтаксисы для относительных PC выражений, таких как метка плюс или минус число, или выражение формы [PC, #number].

3.3.7 Условное выполнение

Большинство инструкций обработки данных может дополнительно обновить флаги условия в *приложении*

регистра состояния программы (APSR) согласно результату работы (см., [состояние Прикладной программы регистрации на странице 20](#)). Некоторые инструкции обновляют все флаги, и немного обновляют подмножество. Если флаг не обновляется, исходное значение сохраняется. См. описания инструкции для флагов, на которые они влияют.

Можно выполнить инструкцию условно, основанную на наборе флагов условия в другой инструкции: Сразу после инструкции, которая обновила флаги после любого числа прошедших инструкций, которые не обновили флаги

Условное выполнение доступно при использовании условных переходов или добавлении суффиксов кода условия к инструкциям. См. [Таблицу 24: суффиксы Кода условия на странице 66](#) для списка суффиксов, чтобы добавить к инструкциям, чтобы сделать их условными инструкциями. Суффикс кода условия позволяет процессору протестировать условие, основанное на флагах. Если тест условия условной инструкции перестал работать, инструкция:

Не выполняется не пишет значения в его целевой регистр не влияет ни на один из флагов не генерирует исключения

Условные инструкции, за исключением условных переходов, должны быть в блоке инструкции If-then. См. [IT на странице 144](#) для получения дополнительной информации и ограничений при использовании инструкции IT. В зависимости от поставщика ассемблер мог бы автоматически вставить инструкцию IT, если у Вас есть условные инструкции вне блока IT.

Используйте инструкции CBZ И CBNZ, чтобы сравнить значение регистра против нуля и ответвление по результату. Этот раздел описывает: [Флаги условия суффиксы кода условия на странице 66](#)

Флаги условия

APSR содержит флаги следующего условия:

N: Набор к 1, когда результат работы отрицателен, иначе очищенный к 0

Z: Набор к 1, когда результатом работы является нуль, иначе очищенный к 0

C: Набор к 1, когда работа приводит к переносу, иначе очищенный к 0.

V: Набор к 1, когда работа вызывает переполнение, иначе очищенный к 0.

Для получения дополнительной информации о APSR см., [состояние Программы регистрации на странице 18](#). Перенос происходит:

Если результат дополнения больше чем или равен 2^{32} **ЕСЛИ** результат вычитания положителен или нуль **КАК** результат встроенной работы многорегистрового циклического сдвигового устройства в перемещении или логической инструкции
Переполнение происходит, если знак результата не соответствует, знаку результата работы, выполняемый в бесконечной точности, например:

Если добавление двух отрицательных величин приводит к положительному значению

ЕСЛИ добавление двух положительных значений приводит к отрицательной величине

ЕСЛИ вычитание положительного значения от отрицательной величины генерирует положительное значение

ЕСЛИ вычитание отрицательной величины от положительного значения генерирует отрицательную величину.

Сравнить операции идентичны вычитанию, для CMP, или добавления, для CMN, за исключением того, что результат отбрасывается. См. описания инструкции для получения дополнительной информации.

Большинство инструкций обновляет флаги состояния, только если суффикс S определяется. См. инструкцию описания для получения дополнительной информации.

Суффиксы кода условия

У инструкций, которые могут быть условным выражением, есть дополнительный код условия, показанный в синтаксисе описания как {cond}. Условное выполнение требует предыдущей инструкции IT. Инструкция с кодом условия выполняется, если флаги кода условия в APSR удовлетворяют указанному условию. [Таблица 24](#) показывает коды условия, чтобы использовать. Можно использовать условное выполнение с инструкцией IT, чтобы сократить количество ответвления инструкции в коде. [Таблица 24](#) также показывает отношение между суффиксами кода условия и N, Z, C, и V флаги.

Таблица 24. Суффиксы кода условия

Суффикс	Флаги	Значение
EQ	Z = 1	Равный
NE	Z = 0	Не равный
CS or HS	C = 1	Выше или то же самое, без знака ≥
CC or LO	C = 0	Ниже, без знака <
MI	N = 1	Отрицательный
PL	N = 0	Положительный или нуль
VS	V = 1	Переполнение
VC	V = 0	Никакое переполнение
HI	C = 1 and Z = 0	Выше, без знака >
LS	C = 0 or Z = 1	Ниже или то же самое, ≤ без знака
GE	N = V	Больше чем или равный, подписанный ≥
LT	N != V	Меньше чем, подписанный <
GT	Z = 0 and N = V	Больше чем, подписанный >
LE	Z = 1 and N != V	Меньше чем или равный, ≤ со знаком
AL	Может иметь любое значение	Всегда. Это - значение по умолчанию, когда никакой суффикс не определяется.

[Определенный пример 1: Абсолютное значение](#) показывает использование условной инструкции, чтобы найти абсолютное значение числа. R0 = ABS (R1).

Определенный пример 1: Абсолютное значение

MOVSR0, R1; R0 = R1, устанавливая флаги

MI IT; Инструкция IT для отрицательного условия RSBMIR0,

R1, #0; Если отрицание, R0 = -R1

Определенный пример 2: Сравните и обновите значения использования условных инструкций к обновлению значение R4, если значение со знаком R0 и R2 больше чем R1 и R3 соответственно.

Определенный пример 2: Сравните и обновите значение

CMP R0, R1; сравните R0 и R1, устанавливая
ITT флагов GT; Инструкция IT для двух условий GT

CMPGT R2, R3; если 'больше чем', сравните R2 и R3, устанавливая флаги
MOVGT R4, R5; если все еще 'больше чем', сделайте R4 = R5

3.3.8 Выбор ширины инструкции

Есть много инструкций, которые могут генерировать 16-разрядное кодирование или 32-разрядное кодирование в зависимости от операндов и места назначения определяется регистр. Для некоторых из этих инструкций можно вызвать размер конкретной инструкции при использовании суффикса ширины инструкции. Суффикс.W вызывает 32-разрядное кодирование инструкции. Суффикс.N вызывает 16-разрядное кодирование инструкции. Если Вы определяете суффикс ширины инструкции, и ассемблер не может генерировать инструкцию кодируя требуемой ширины, это генерирует ошибку. В некоторых случаях могло бы быть необходимо определить суффикс.W, например если операнд метка инструкции или литеральных данных, как в случае команд перехода. Это - потому что ассемблер не мог автоматически генерировать кодирование правильного размера. Чтобы использовать суффикс ширины инструкции, поместите его сразу после мнемоники команды и кода условия. *Определенный пример 3: выбор ширины Инструкции* показывает инструкции с суффиксом ширины инструкции.

Определенный пример 3: выбор ширины Инструкции

Метка BCS.W; создает 32-разрядную инструкцию даже для короткого
ответвления ADDS.W R0, R0, R1; создает 32-разрядную инструкцию даже при том,
что то же самое
; работа может быть сделана 16-разрядной инструкцией

3.4 Инструкции доступа памяти

Таблица 25 показывает инструкции доступа памяти:

Таблица 25. Инструкции доступа памяти

Мнемосхема	Краткое описание	См.
ADR	Загрузите относительный PC адрес	<i>ADR на странице 69</i>
CLREX	Очистите монопольный	<i>CLREX на странице 79</i>
LDM(mode)	Загрузите многократные регистры	<i>LDM и STM на странице 75</i>
LDR(type)	Регистр загрузки, используя непосредственное смещение	<i>LDR и STR, непосредственное смещение на странице 70</i>
LDR(type)	Регистр загрузки, используя регистр смещается	<i>LDR и STR, регистр смещается на странице 72</i>
LDR(type)T	Регистр загрузки с непривилегированным доступом	<i>LDR и STR, непривилегированный на странице 73</i>
LDR	Регистр загрузки, используя относительный PC адрес	<i>LDR, родственник PC на странице 74</i>
LDRD	Двойной регистр загрузки	<i>LDR и STR, непосредственное смещение на странице 70</i>
LDREX(type)	Монопольный регистр загрузки	<i>LDREX и STREX на странице 78</i>
POP	Вытолкните регистры от стека	<i>PUSH и POP на странице 77</i>
PUSH	Продвиньте регистры на стек	<i>PUSH и POP на странице 77</i>
STM(mode)	Сохраните многократные регистры	<i>LDM и STM на странице 75</i>
STR(type)	Регистр хранилища, используя непосредственное смещение	<i>LDR и STR, непосредственное смещение на странице 70</i>
STR(type)	Регистр хранилища, используя регистр смещается	<i>LDR и STR, регистр смещается на странице 72</i>
STR(type)T	Регистр хранилища с непривилегированным доступом	<i>LDR и STR, непривилегированный на странице 73</i>
STREX(type)	Монопольный регистр хранилища	<i>LDREX и STREX на странице 78</i>

3.4.1 ADR

Загрузите относительный PC адрес.

Синтаксис

ADR {cond} Резерфорд, метка

где:

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),

'Резерфорд' является целевым регистром

'метка' является относительным PC выражением (см. [относительные PC выражения на странице 64](#)),

Работа

ADR определяет адрес, добавляя непосредственное значение к PC и пишет результат в целевой регистр. ADR производит позиционно-независимый код, потому что адрес относителен PC. Если Вы используете ADR, чтобы генерировать целевой адрес для BX или инструкции BLX, следует гарантировать бит [0] из адреса, который Вы генерируете, устанавливается to1 для корректного выполнения.

Значения *метки* должны быть в пределах диапазона -4095 к 4095 от адреса в PC.

Отметьте:

Вам, возможно, придется использовать суффикс *W*, чтобы получить максимальный диапазон смещения или генерировать адреса, которые не выравниваются словом (см. [выбор ширины Инструкции на странице 67](#)).

Ограничения

Резерфорд не должен быть ни SP, ни PC.

Флаги условия

Эта инструкция не изменяет флаги.

Примеры

```
ADR R1, TextMessage; запишите значение адреса расположения, маркированного
как
; TextMessage к R1
```

3.4.2 LDR и STR, непосредственное смещение

Загрузка и хранилище с непосредственным смещением, предварительно индексированным непосредственным смещением, или постиндексированный непосредственным смещением.

Синтаксис

```
ор {тип} {cond} Rt, [Rn {#offset}]; непосредственное смещение
ор {тип} {cond} Rt, [Rn, #offset]!; предварительно индексированный
ор {тип} {cond} Rt, [Rn], #offset; постиндексированный
орD {cond} Rt, Rt2, [Rn {#offset}]; непосредственное смещение, два слова
орD {cond} Rt, Rt2, [Rn, #offset]!; предварительно индексированный, два слова
орD {cond} Rt, Rt2, [Rn], #offset; постиндексированный, два слова
```

где:

'ор' является любой LDR (регистр загрузки) или STR (регистр хранилища)

'тип' является одним из следующего:

B: Байт без знака, нуль расширяется на 32 бита загрузок:

байт Со знаком, знак расширяется на 32 бита (LDR только)

H: полуслово Без знака, нуль расширяет до 32 битов на загрузках

SH: полуслово Со знаком, знак расширяется на 32 бита (LDR только)

:- Опустите, для слова

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),

'Rt' является регистром, чтобы загрузить или сохранить

'Rn' является регистром, на котором базируется адрес памяти

'смещение' является смещением от Rn. Если смещение опускается, адрес является содержанием

Rn 'Rt2' является дополнительным регистром, чтобы загрузить или сохранить для операций с двумя словами

Работа

Инструкции LDR загружают один или два регистра значением из памяти. Хранилище инструкций STR одно или два значения регистра к памяти. Загрузка и инструкции хранилища с непосредственным смещением могут использовать следующие режимы адресации:

Адресация смещения

Значение смещения добавляется или вычитается из адреса, полученного из регистра Rn . Результат используется в качестве адреса для доступа памяти. Регистр Rn неизменен. Синтаксис ассемблера для этого режима: $[Rn, \#offset]$.

Предварительно индексированная адресация

Значение смещения добавляется или вычитается из адреса, полученного из регистра Rn . Результат используется в качестве адреса для доступа памяти и записывается обратно в регистр Rn . Синтаксис ассемблера для этого режима: $[Rn, \#offset]!$

Постиндексированная адресация

Адрес, полученный из регистра Rn , используется в качестве адреса для доступа памяти. Значение смещения добавляется или вычитается из адреса, и записывается обратно в регистр Rn . Синтаксис ассемблера для этого режима: $[Rn], \#offset$.

Значение, чтобы загрузиться или сохранить может быть байтом, полусловом, словом, или двумя словами. Байты и полуслова могут быть подписаны или без знака (см. [выравнивание Адреса на странице 64](#)).

Таблица 26 показывает диапазон смещений для непосредственных, предварительно индексированных и постиндексированных форм.

Таблица 26. Непосредственные, предварительно индексированные и постиндексированные диапазоны смещения

Тип инструкции	Непосредственное смещение	Предварительно индексированный	Постиндексированный
Слово, полуслово, подписанное полуслово, байт, или подписанный байт	-255 to 4095	-255 to 255	-255 to 255
Два слова	Множественный из 4 в диапазоне-1020 к 1020	Множественный из 4 в диапазоне-1020 к 1020	Множественный из 4 в диапазоне-1020 к 1020

Ограничения

Для инструкций загрузки

- Rt может быть SP, или PC для слова загружается только
- Rt должен отличаться от $Rt2$ для загрузок с двумя словами
- Rn должен отличаться от Rt и $Rt2$ в предварительно индексированных или постиндексированных формах

Когда Rt является PC, одним словом загружают

инструкцию

- бит [0] из загруженного значения должен быть 1 для корректного выполнения
- Ответвление происходит с адресом, создаваемым, изменяя бит [0] из загруженного значения к 0
- Если инструкция является условным выражением, это должна быть последняя инструкция в блоке IT

Для инструкций хранилища

- Rt может быть SP для хранилищ слова только
- Rt не должен быть PC
- Rn не должен быть PC
- Rn должен отличаться от Rt и $Rt2$ в предварительно индексированных или постиндексированных формах

Флаги условия

Эти инструкции не изменяют флаги.

Примеры

```
LDR R8, [R10] ; загрузки R8 от адреса в R10.
LDRNE R2, [R5, #960]!; загрузки (условно) R2 от слова
; На 960 байтов выше адреса в R5, и; инкременты R5 960.
STR R2, [R9, #const-struct]; константа-struct является оценкой выражения
; к константе в диапазоне 0-4095. STRH R3, [R4], #4; Сохраните R3
как данные полуслова в адрес в
; R4, затем постепенно увеличьте R4 4 LDRD R8, R9, [R3, #0x20];
Загрузите R8 из слова на 32 байта выше
; адрес в R3, и загрузка R9 от word 36; байты выше адреса в R3
STRD R0, R1, [R8], #-16; Сохраните R0, чтобы адресоваться в R8, и сохранить R1
к
; слово на 4 байта выше адреса в R8; и затем постепенно уменьшите R8 16.
```

3.4.3 LDR и STR, регистр смещения

Загрузка и хранилище с регистром смещаются.

Синтаксис

```
op {тип} {cond} Rt, [Rn, Комната {LSL #n}]
```

где:

'op' является любой LDR (регистр загрузки) или STR (регистр хранилища)

'тип' является одним из следующего:

B: Байт без знака, нуль расширяется на 32 бита загрузок

: байт Со знаком, знак расширяется на 32 бита (LDR только)

H: полуслово Без знака, нуль расширяет до 32 битов на загрузках

SH: полуслово Со знаком, знак расширяется на 32 бита (LDR только)

:- Опустите, для слова

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),

'Rt' является регистром, чтобы загрузить или сохранить

'Rn' является регистром, на котором базируется адрес памяти

'Rm' является регистром, содержащим значение, которое будет использоваться в качестве смещения 'LSL

#n' является дополнительным сдвигом, с n в диапазоне от 0 до 3

Работа

Инструкции LDR загружают регистр значением из памяти. Инструкции STR хранят регистр значения в памяти. Адресом памяти загрузки или хранилища является смещение от регистра Rn. Смещение определяется Rm регистра и может быть смещено оставленное на 3 бита, используя LSL. Значение, чтобы загрузить или сохранить может быть байтом, полусловом, или словом. Для инструкций загрузки байты и полуслова могут быть подписаны или без знака (см. [выравнивание Адреса на странице 64](#)).

Ограничения

В этих инструкциях:

Rn не должен быть PC Rm не должна быть ни SP, ни PC Rt может быть

SP только для загрузок слова и хранилищ слова Rt может быть PC

только для загрузок слова

Когда Rt является PC, одним словом загружают инструкцию:

Бит [0] из загруженного значения должен быть 1 для корректного выполнения, и ответвление происходит с этим выровненным полусловом адрес

Если инструкция является условным выражением, это должна быть последняя инструкция в блоке IT.

Флаги условия

Эти инструкции не изменяют флаги.

Примеры

```
STR R0, [R5, R1]; значение хранилища R0 в адрес, равный
                  ; сумма R5 и R1
LDRSB R0, [R5, R1, LSL #1]; считайте значение байта из адреса, равного
                           ; сумма R5 и два раза R1, знак расширял это
                           ; к слову оценивают и помещенный это в R0
STR R0, [R1, R2, LSL #2]; хранилища R0 к адресу, равному сумме R1
                           ; и четыре раза R2
```

3.4.4 LDR и STR, непривилегированный

Загрузка и хранение с непривилегированным доступом.

Синтаксис

op {тип} T {cond} Rt, [Rn {#offset}]; непосредственное смещение

где:

'op' является любой LDR (регистр загрузки) или STR (регистр хранилища)

'тип' является одним из следующего:

B: Байт без знака, нуль расширяется на 32 бита загрузки

H: полуслово Без знака, нуль расширяет до 32 битов на загрузках

SH: полуслово Со знаком, знак расширяется на 32 бита (LDR только)

:- Опустите, для слова

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),

'Rt' является регистром, чтобы загрузить или сохранить

'Rn' является регистром, на котором базируется адрес памяти

'смещение' является смещением от Rn и может быть от 0 до 255. Если смещение опускается, адрес значение в Rn.

Работа

Они загружают и хранят инструкции, выполняют ту же самую функцию как доступ памяти инструкции с непосредственным смещением (см. [LDR и STR, непосредственное смещение на странице 70](#)). Различие - в том, что у этих инструкций есть только непривилегированный доступ даже когда использующийся в привилегированном программном обеспечении.

Когда используются в непривилегированном программном обеспечении, эти инструкции ведут себя точно таким же образом как нормальные инструкции доступа памяти с непосредственным смещением.

Ограничения

В этих инструкциях:

Rn не должен быть PC

Rt не должен быть ни SP, ни PC.

Флаги условия

Эти инструкции не изменяют флаги.

Примеры

STRBTEQ R4, [R7] ; условно сохраните младший значащий байт в R4 к адресу в R7, с непривилегированным доступом

LDRHT R2, [R2, #8]; загрузите значение полуслова из адреса, равного сумме R2 и 8 в R2, с непривилегированным доступом

3.4.5 LDR, родственник PC

Регистр загрузки из памяти.

Синтаксис

LDR {тип} {cond} Rt, метка

LDRD {cond} Rt, Rt2, метка; загрузите два слова

где:

'Тип' является одним из следующего:

B: Байт без знака, нуль расширяется на 32 бита:

байт Со знаком, знак расширяется на 32 бита

H: полуслово Без знака, знак расширяется на

SH на 32 бита: полуслово Со знаком, знак расширяется на 32 бита

:- Опустите, для слова

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#))

'Rt' является регистром, чтобы загрузить или сохранить

'Rt2' является вторым регистром, чтобы загрузить или сохранить

'метка' является относительным PC выражением (см. [относительные PC выражения на странице 64](#)),

Работа

LDR загружает регистр значением от относительного PC адреса памяти.

Адрес памяти определяется меткой или смещением от PC. Значение, чтобы загрузиться или сохранить может быть байтом, полусловом, или словом. Для инструкций загрузки байты и полуслова могут быть подписаны или без знака (см. [выравнивание Адреса на странице 64](#)).

'метка' должна быть в пределах ограниченного диапазона текущей команды. [Таблица 27](#) показывает возможное смещения между меткой и PC. Вам, возможно, придется использовать суффикс.W, чтобы получить максимальный диапазон смещения (см. [выбор ширины Инструкции на странице 67](#)).

Таблица 27. PC метки смещал диапазоны

Тип инструкции	Диапазон смещения
Слово, полуслово, подписанное полуслово, байт, подписанный байт	-4095 to 4095
Два слова	-1020 to 1020

Ограничения

В этих инструкциях:

Rt2 не должен быть ни SP, ни PC

Rt должен отличаться от Rt2

Rt может быть SP или PC только для загрузок слова

когда Rt является PC, одним словом загружают инструкцию: бит [0] из загруженного значения должен быть 1 для корректного выполнения, и ответвления выровненного полусловом адреса. Если инструкция является условным выражением, это должна быть последняя инструкция в блоке IT.

Флаги условия

Эти инструкции не изменяют флаги.

Примеры

```
LDR R0, LookUpTable; загрузите R0 словом данных от адреса  
                      ; маркированный как LookUpTable LDRSB  
R7, localdata; загрузите значение байта из маркированного  
адреса  
                ; как localdata, знак расширяет это до  
                ; слова; значение, и помещенный это в R7
```

3.4.6 LDM и STM

Загружает сохраняет многократные регистры.

Синтаксис

op {addr_mode} {cond} Rn{!}, переслюда

где:

'op' является любой LDM (загрузите многократный регистр), или STM (хранят многократный регистр),

'addr_mode' является любым следующим:

IA: Инкрементный адрес после каждого доступа (это - значение по умолчанию),

DB: Декрементный адрес перед каждым доступом

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#))

'Rn' является регистром, на котором базируются адреса памяти

'!' дополнительный суффикс обратной записи.

Если! присутствует, заключительный адрес, который загружается из сохраненных записей обратно в Rn.

'reglist' является списком одного или более регистров, которые будут загружены или сохранены, включены в фигурные скобки может содержать диапазоны регистра должно быть разделено от запятой, если содержит больше чем один регистр или диапазон регистра (см. [Примеры на странице 76](#)).

LDM и LDMFD являются синонимами для LDMIA. LDMFD обращается к его использованию для того, чтобы вытолкать данные от убывающих стеков.

LDMEA является синонимом для LDMDB, и обращается к его использованию для того, чтобы вытолкать данные от пустого возрастающего стека.

STM и STMEA являются синонимами для STMIA. STMEA обращается к его использованию для того, чтобы продвинуть данные на пустые возрастающие стеки.

STMFD является синонимом для STMDB, и обращается к его использованию для того, чтобы продвинуть данные на полное убывание стеков

Работа

Инструкции LDM загружают, регистры в reglist со значениями слова из памяти адреса основанными на Rn.

Инструкции STM хранят значения слова в регистрах в к базируемым адресам памяти на Rn.

Для LDM, LDMIA, LDMFD, STM, STMIA, и STMEA адреса памяти, используемые для доступы в 4-байтовых интервалах в пределах от Rn к Rn + 4 * (n-1), где n является числом

регистры в *reglist*. Доступы происходят в порядке увеличения чисел регистра, с самым низким пронумерованным регистром, используя самый низкий адрес памяти и регистр самого большого количества, используя самый высокий адрес памяти. Если суффикс обратной записи определяется, значением $Rn + 4 * (n-1)$ записывается обратно к Rn .

Для LDMDb, LDMEa, STMDB, и STMFD адреса памяти используются для доступов в 4-байтовых интервалах в пределах от Rn к $Rn - 4 * (n-1)$, где n является числом регистров в *reglist*. Доступы происходят в порядке сокращения чисел регистра, с самым высоким пронумерованным регистром, используя самый высокий адрес памяти и самый низкий регистр числа, используя самый низкий адрес памяти. Если суффикс обратной записи определяется, значение, $Rn - 4 * (n)$ записывается обратно к Rn .

PUSH и инструкции POP могут быть выражены в этой форме (см. [PUSH и POP](#) для детали).

Ограничения

В этих инструкциях:

Rn не должен быть PC *reglist* не должен содержать SP

В любой инструкции STM *reglist* не должен содержать PC

В любой инструкции LDM *reglist* не должен содержать PC,

если содержит LR *reglist* не должен содержать Rn ,

если Вы определяете суффикс обратной записи Когда PC находится в *reglist* в инструкции LDM:

Бит [0] из значения, загруженного в PC, должен быть 1 для корректного выполнения, и ответвления происходит с выровненным полусловом адресом.

Если инструкция является условным выражением, должна быть последняя инструкция в блоке IT

Флаги условия

Эти инструкции не изменяют флаги.

Примеры

LDM R8, {R0, R2, R9} ; LDMIA является синонимом для LDM

STMDB R1!, {R3-R6, R11, R12}

Неправильные примеры

STM R5!, {R5, R4, R9}; значение, сохраненное для R5, является непредсказуемым LDM R2, {}
; в списке должен быть по крайней мере один регистр

3.4.7 PUSH и POP

Продвиньте регистры, и вытолкайте регистры от полно убывающего стека. PUSH и POP синонимы для STMDB и LDM (или LDMIA) с памятью адресуются для доступа, основанного на SP, и с заключительным адресом для доступа, записанного обратно к SP. PUSH и POP являются привилегированной мнемоникой в этих случаях.

Синтаксис

ПРОДВИНЬТЕ {cond} *reglist*

POP {cond} *reglist*

где:

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)), 'reglist' является непустым списком регистров (или диапазоны регистра), включенный

в фигурные скобки. Запятые должны разделить списки регистра или диапазоны (см. [Примеры на странице 76](#)).

Работа

ПРОДВИНЬТЕ регистры хранилищ на стеке в порядок сокращения чисел регистра, с самого высокого пронумерованного регистра, используя самую высокую память адреса и самый низкий пронумерованный регистр, используя самый низкий адрес памяти.

POP загружает регистры из стека в порядке увеличения чисел регистра, с самого низкого пронумерованного регистра, используя самую низкую память адреса и самый высокий пронумерованный регистр, используя самый высокий адрес памяти.

ПРОДВИНЬТЕ использованное значение в регистре SP минус четыре как самый высокий адрес памяти,

POP использует значение регистра SP в качестве самого низкого адреса памяти, реализовывая полный - убывающий стек. На завершении, ПРОДВИНЬТЕ обновленный регистр SP указав на расположение самого низкого значения хранилища, POP обновляет регистр SP, чтобы указать на расположение самого высокого загруженного расположения.

Если инструкция POP включает PC в свой *reglist*, ответвление к этому расположению выполняется

когда инструкция POP завершилась. Бит [0] из значения, считанного из PC, используется, чтобы обновить APSR T-bit. Этот бит должен быть 1, чтобы гарантировать корректную работу. См. [LDM и STM на странице 75](#) для получения дополнительной информации.

Ограничения

В этих инструкциях:

'*reglist*' не должна содержать SP

ДЛЯ инструкции PUSH *reglist* не должно содержать PC

ДЛЯ инструкции POP *reglist* не должно содержать PC, если содержит LR.

Когда PC находится в *reglist* инструкции POP: бит [0] из значения, загруженного в PC, должен быть 1 для корректного выполнения, и ответвление происходит с этим выровненным полусловом адресом. Если инструкция является условным выражением, это должна быть последняя инструкция в блоке IT.

Флаги условия

Эти инструкции не изменяют флаги.

Примеры

ПРОДВИНЬТЕ {R0, R4-R7}; Продвиньте R0, R4, R5, R6, R7 на стек

ПРОДВИНЬТЕ {R2, LR} ; Продвиньте R2 и регистр ссылки на стек

POP {R0, R6, PC}; Вытолкайте r0, r6 и PC от стека, затем перейдите к новому PC.

3.4.8 LDREX и STREX

Загрузка и монопольный регистр хранилища.

Синтаксис

LDREX {cond} Rt, [Rn {#offset}]

STREX {cond} Резерфорд, Rt, [Rn {#offset}] LDREXB {cond} Rt,

[Rn] STREXB {cond} Резерфорд, Rt, [Rn] LDREXH {cond} Rt, [Rn]

STREXH {cond} Резерфорд, Rt, [Rn]

где:

'*cond*' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)), 'Резерфорд'

является целевым регистром для возвращенного состояния

'*Rt*' является регистром, чтобы загрузить или сохранить

'*Rn*' является регистром, на котором базируется адрес памяти

'*Смещение*' является дополнительным смещением, применяется к значению *Rn*. Если *смещение* опускается, адрес значение в *Rn*.

Работа

LDREX, LDREXB, и LDREXH загружают слово, байт, и полуслово соответственно от адреса памяти.

STREX, STREXB, и STREXH пытаются сохранить слово, байт, и полуслово соответственно адресу памяти. Адрес, используемый в любой монопольной хранилищем инструкции, должен быть тем же самым как адрес в последний раз выполняемой монопольной загрузкой инструкции. У значения, сохраненного монопольным хранилищем инструкций, должен также быть тем же самым что и размер данных как значение, загруженное предыдущей монопольной загрузкой инструкции. Это означает, что программное обеспечение должно всегда использовать загрузку - монопольная инструкция и соответствующая монопольная хранилищем инструкция, чтобы выполнить работу синхронизации, смотри [примитивы Синхронизации на странице 33](#).

Если монопольная хранилищем инструкция выполняет хранилище, она пишет 0 в его целевой регистр. Если это не выполняет хранилище, пишет 1 в его целевой регистр. Если монопольная хранилищем инструкция запишет 0 в целевой регистр, то гарантируется, что никакой другой процесс в системе не получит доступ к расположению памяти между монопольной загрузкой и монопольным хранилищем инструкций.

По причинам производительности, сохраните число инструкций между соответствующей загрузкой - монопольная и монопольная хранилищем инструкция к минимуму.

Отметьте: Результат выполнения монопольной хранилищем инструкции к адресу, который отличается от этого используемый в предыдущей монопольной загрузкой инструкции непредсказуемо.

Ограничения

В этих инструкциях:

Не используйте PC не используйте SP для *Rd* и *Rt*

ДЛЯ STREX Резерфорд должен отличаться и от *Rt* и от *Rn*

значение смещения должно быть кратным числом четыре в диапазоне 0-1020

Флаги условия

Эти инструкции не изменяют флаги.

Примеры

```
MOV P1, #0x1 LDREX R0, [LockAddr] CMP R0, #0 ITT EQ STREXEQ R0, R1, [LockAddr] CMPEQ R0, #0
попытка BNE
; инициализируйте 'блокировку взятая' попытка значения; загрузите значение
блокировки; действительно ли блокировка свободна?; инструкция IT для STREXEQ и
CMPEQ; попытайтесь требовать блокировки; это успешно выполнялось?; нет -
попробовали еще раз; да - у нас есть блокировка
```

3.4.9 CLREX

Очистить монопольный.

Синтаксис

CLREX {cond}

где: 'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),

Работа

Используйте CLREX, чтобы сделать следующий STREX, STREXB, или инструкцию STREXH пишет 1 в ее место назначения регистр и сбой, чтобы выполнить хранилище. Полезно в коде обработчика

исключением вызвать отказ хранилища, монопольного, если исключение происходит между загрузкой монопольная инструкция и соответствующим хранилищем монопольная инструкция в работе синхронизации. См. [примитивы Синхронизации на странице 33](#) для получения дополнительной информации.

Флаги условия

Эти инструкции не изменяют флаги.

Примеры

CLREX

3.5 Общие инструкции обработки данных

Таблица 28 показывает инструкции обработки данных.

Таблица 28. Инструкции обработки данных

Мнемосхема	Краткое описание	См.
ADC	Добавьте с переносом	<i>ADD, ADC, SUB, SBC, и RSB на странице 82</i>
ADD	Добавить	<i>ADD, ADC, SUB, SBC, и RSB на странице 82</i>
ADDW	Добавить	<i>ADD, ADC, SUB, SBC, и RSB на странице 82</i>
AND	Логичный И	<i>И, OPP, EOR, BIC, и ORN на странице 84</i>
ASR	Право арифметического сдвига	<i>ASR, LSL, LSR, ROR, и RRX на странице 85</i>
BIC	Чистый бит	<i>И, OPP, EOR, BIC, и ORN на странице 84</i>
CLZ	Подсчитайте начальные нули	<i>CLZ на странице 86</i>
CMN	Сравните отрицательный	<i>CMP и CMN на странице 87</i>
CMP	Сравниться	<i>CMP и CMN на странице 87</i>
EOR	Монопольный ИЛИ	<i>И, OPP, EOR, BIC, и ORN на странице 84</i>
LSL	Логический сдвиг уехал налево	<i>ASR, LSL, LSR, ROR, и RRX на странице 85</i>
LSR	логический сдвиг уехал направо	<i>ASR, LSL, LSR, ROR, и RRX на странице 85</i>
MOV	Переместиться	<i>MOB и MVN на странице 88</i>
MOVT	Переместите вершину	<i>MOVT на странице 90</i>
MOVW	Переместите 16-разрядную константу	<i>MOB и MVN на странице 88</i>
MVN	Не перемещать	<i>MOB и MVN на странице 88</i>
ORN	Логичный ИЛИ НЕТ	<i>И, OPP, EOR, BIC, и ORN на странице 84</i>
ORR	Логичный ИЛИ	<i>И, OPP, EOR, BIC, и ORN на странице 84</i>
RBIT	Обратные биты	<i>БЕРСИЯ, REV16, REVSH, и RBIT на странице 91</i>
REV	Обратный порядок байтов одним словом	<i>БЕРСИЯ, REV16, REVSH, и RBIT на странице 91</i>
REV16	Обратный порядок байтов в каждом полуслове	<i>БЕРСИЯ, REV16, REVSH, и RBIT на странице 91</i>
REVSH	Обратный порядок байтов в нижнем полуслове и знаке расширяется	<i>БЕРСИЯ, REV16, REVSH, и RBIT на странице 91</i>
ROR	Поверните право	<i>ASR, LSL, LSR, ROR, и RRX на странице 85</i>
RRX	Вращайтесь прямо с, расширяются	<i>ASR, LSL, LSR, ROR, и RRX на странице 85</i>
RSB	Реверс вычитает	<i>ADD, ADC, SUB, SBC, и RSB на странице 82</i>
SADD16	Подписанный добавляют 16	<i>SADD16 и SADD8 на странице 92</i>
SADD8	Подписанный добавляют 8	<i>SADD16 и SADD8 на странице 92</i>
SASX	Подписанный добавляют и вычитают с обменом	<i>SASX и SSAX на странице 97</i>
SSAX	Подписанный вычитают и добавляют с обменом	<i>SASX и SSAX на странице 97</i>
SBC	Вычитите с переносом	<i>ADD, ADC, SUB, SBC, и RSB на странице 82</i>

Таблица 28. Инструкции обработки данных (продолжение)

Мнемосхема	Краткое описание	См
SHADD16	Деление на два со знаком добавляет 16	<i>SHADD16 и SHADD8 на странице 93</i>
SHADD8	Деление на два со знаком добавляет 8	<i>SHADD16 и SHADD8 на странице 93</i>
SHASX	Деление на два со знаком добавляет и вычитает с обменом	<i>SHASX и SHSAX на странице 94</i>
SHSAX	Деление на два со знаком вычитает и добавляет с обменом	<i>SHASX и SHSAX на странице 94</i>
SHSUB16	Деление на два со знаком вычитает 16	<i>SHSUB16 и SHSUB8 на странице 95</i>
SHSUB8	Деление на два со знаком вычитает 8	<i>SHSUB16 и SHSUB8 на странице 95</i>
SSUB16	Подписанный вычитают 16	<i>SSUB16 и SSUB8 на странице 96</i>
SSUB8	Подписанный вычитают 8	<i>SSUB16 и SSUB8 на странице 96</i>
SUB	Вычесть	<i>ADD, ADC, SUB, SBC, и RSB на странице 82</i>
SUBW	Вычесть	<i>ADD, ADC, SUB, SBC, и RSB на странице 82</i>
TEQ	Тестовая эквивалентность	<i>SADD16 и SADD8 на странице 92</i>
TST	Тест	<i>SADD16 и SADD8 на странице 92</i>
UADD16	Без знака добавляют 16	<i>UADD16 и UADD8 на странице 99</i>
UADD8	Без знака добавляют 8	<i>UADD16 и UADD8 на странице 99</i>
UASX	Без знака добавляют и вычитают с обменом	<i>UASX и USAX на странице 100</i>
USAX	Без знака вычитают и добавляют с обменом	<i>UASX и USAX на странице 100</i>
UHADD16	Деление на два без знака добавляет 16	<i>UHADD16 и UHADD8 на странице 101</i>
UHADD8	Деление на два без знака добавляет 8	<i>UHADD16 и UHADD8 на странице 101</i>
UHASX	Деление на два без знака добавляет и вычитает с обменом	<i>UHASX и UHSAX на странице 102</i>
UHSAX	Деление на два без знака вычитает и добавляет с обменом	<i>UHASX и UHSAX на странице 102</i>
UHSUB16	Деление на два без знака вычитает 16	<i>UHSUB16 и UHSUB8 на странице 103</i>
UHSUB8	Деление на два без знака вычитает 8	<i>UHSUB16 и UHSUB8 на странице 103</i>
USAD8	Сумма без знака абсолютных разностей	<i>USAD8 на странице 105</i>
USADA8	Сумма без знака абсолютных разностей и накапливается	<i>USADA8 на странице 106</i>
USUB16	Без знака вычитают 16	<i>USUB16 и USUB8 на странице 107</i>
USUB8	Без знака вычитают 8	<i>USUB16 и USUB8 на странице 107</i>

3.5.1 ADD, ADC, SUB, SBC, и RSB

Добавьте, добавьте с переносом, вычитите, вычитите с переносом, и реверс вычитает.

Синтаксис

ор {S} {cond} {Резерфорд}, Rn, Operand2

ор {cond} {Резерфорд}, Rn, #imm12; ADD и SUB только

где:

'ор' является одним из:

ADD: Добавьте

ADC: Добавьте с переносом

SUB: Вычитите

SBC: Вычитите с переносом

RSB: Реверс вычитает

'S' является дополнительным суффиксом. Если S определяется, флаги кода условия обновляются на результат работы (см. [Условное выполнение на странице 64](#)),

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),

'Резерфорд' является целевым регистром. Если Резерфорд опускается, целевым регистром является

Rn 'Rn' является регистром, содержащим первый операнд

'Operand2' является гибким вторым операндом (см. [Гибкий второй операнд на странице 59](#) для детали опций).

'imm12' является любым значением в диапазоне 0-4095

Работа

Инструкция ADD добавляет значение *operand2* или *imm12* к значению в *Rn*. Инструкция ADC добавляет значения в *Rn* и *operand2*, вместе с флагом переноса. Инструкция SUB вычитает значение *operand2* или *imm12* от значения в *Rn*. Инструкция SBC вычитает значение *operand2* от значения в *Rn*. Если флаг переноса

очистить, результат уменьшается. Инструкция RSB вычитает значение в *Rn* от значения *operand2*. Это полезно из-за широкого диапазона опций для *operand2*.

Используйте ADC и SBC, чтобы синтезировать многословную арифметику (см. [арифметические примеры Многословные на странице 83](#) и [ADR на странице 69](#)). ADDW эквивалентен синтаксису ADD, который использует *imm12* операнд. SUBW эквивалентен к синтаксису SUB, который использует *imm12* операнд.

Ограничения

В этих инструкциях:

Operand2 не должен быть ни SP, ни *Резерфордом* PC, может быть SP только в ADD и SUB, и только со следующими дополнительными ограничениями:

- *Rn* должен также быть SP
- Любой переключается на нижний регистр, *operand2* должен быть ограничен максимумом тремя битами, используя LSL

Rn может быть SP только в ADD, и СУБ-Ройд может быть PC только в ADD {cond}

PC, PC, инструкция *Rm* где:

- Не следует определять суффикс S
- *Rm* не должен быть ни PC, ни SP

- Если инструкция является условным выражением, это должна быть последняя инструкция в блоке IT. За исключением ADD {cond} PC, PC, инструкция Rm, Rn может быть PC только в ADD и SUB, и только со следующими дополнительными ограничениями:
- Не следует определять суффикс S
- Второй операнд должен быть константой в диапазоне от 0 до 4095

Отметьте: 1 При использовании PC для дополнения или вычитания, биты [1:0] PC

округляются к b00 прежде, чем выполнить вычисление, делая базовый адрес для выровненного словом вычисления.

2 Если Вы хотите генерировать адрес инструкции, необходимо скорректировать базисуемую константу на значении PC. ARM рекомендует, чтобы Вы использовали инструкцию ADR вместо ADD или SUB с Rn, равным PC, потому что Ваш ассемблер автоматически вычисляет корректную константу для инструкции ADR.

Когда Резерфорд является PC в ADD {cond} PC, PC, инструкция Rm:

Бит [0] из значения, записанного PC, игнорируется ответвление происходит с адресом, создаваемым, вызывая бит [0] из того значения к 0

Флаги условия

Если S определяется, эти инструкции обновляют N, Z, C и V флагов согласно результату.

Примеры

ADD R2, R1, R3

R8, R6, #240 ; устанавливает флаги на результате RSB R4, R4, #1280 ; вычитает содержание R4 с 1280 ADCHI R11, R0, R3 ; только выполняемый, если C отмечают набор и четкий флаг Z

Арифметические примеры многословные

Определенный пример 4: 64-разрядное дополнение показывает две инструкции, которые добавляют 64-разрядное целое число содержащийся в R2 и R3 к другому 64-разрядному целому числу, содержавшемуся в R0 и R1, и месте результат в R4 и R5.

Определенный пример 4: 64-разрядное дополнение

ADDS R4, R0, R2 ; добавьте младший значащий ADC слов R5, R1, R3 ; добавьте старшие значащие слова с переносом

Значения многословные не должны использовать последовательные регистры. *Определенный пример 5: 96-разрядное вычитание* показывает инструкции, которые вычитают 96-разрядное целое число, содержавшееся в R9, R1, и R11 от другого содержавшегося в R6, R2, и R8. Пример хранит результат в R6, R9, и R2.

Определенный пример 5: 96-разрядное вычитание

R6, R6, R9 ; вычитите младшие значащие слова SBCS R9, R2, R1 ; вычитите средние слова с переносом SBC R2, R8, R11 ; вычитите старшие значащие слова с переносом

3.5.2 И, ORP, EOR, BIC, и ORN

Логичный И, ИЛИ, монополюсный ИЛИ, чистый бит, и ИЛИ НЕТ.

Синтаксис

op {S} {cond} {Резерфорд}, Rn, Operand2

где:

'op' является одним из:

И: Логичный И

ORP: Логический EOR

ИЛИ или набора битов:

Логичный монопольный ИЛИ

BIC: Логичный И НЕ или бит очищают ORN:

Логичный ИЛИ НЕТ

'S' является дополнительным суффиксом. Если S определяется, флаги кода условия обновляются на результат работы (см. [Условное выполнение на странице 64](#)).

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),

'Резерфорд' является целевым регистром

'Rn' является регистром, содержащим первый операнд

'Operand2' является гибким вторым операндом (см. [Гибкий второй операнд на странице 59](#) для опций).

Работа

И, EOR, и инструкции ORPA выполняют поразрядно И, монопольные ИЛИ, и ИЛИ операции на значениях в Rn и operand2. Инструкция BIC выполняет операцию И на битах в Rn с дополнениями соответствующие биты в значении operand2. Инструкция ORN выполняет ИЛИ работа на битах в Rn с дополнениями соответствующие биты в значении operand2.

Ограничения

Не используйте или SP или PC.

Флаги условия

Если S определяется, эти инструкции:

Обновите флаги N и Z согласно результату

МОЖЕТ обновить флаг C во время вычисления operand2 (см. [Гибкий второй операнд на странице 59](#))

Не влияет на V флаги

Примеры

```
И R9, R2, #0xFF00
```

```
ORREQ R2, R0, R5 ANDS R9, R8, #0x19 EORS
```

```
R7, R11, #0x18181818 BIC R0, R1, #0xab ORN
```

```
R7, R11, R14, ROR #4 ORNS R7, R11, R14,
```

```
ASR #32
```

3.5.3 ASR, LSL, LSR, ROR, и RRX

Право арифметического сдвига, логический сдвиг уехал, право логического сдвига, поверните право, и вращайтесь прямо ,расшириться.

Синтаксис

ор {S} {cond} Резерфорд, Комната, PTC

ор {S} {cond} Резерфорд, Комната, #n RRX {S} {cond} Резерфорд, Комната

где:

'ор' является одним из:

ASR: LSL права Арифметического сдвига
Логический сдвиг оставил
LSR: право Логического сдвига
ROR: Поверните право

'S' является дополнительным суффиксом. Если S определяется, флаги кода условия обновляются на результат работы (см. [Условное выполнение на странице 64](#)),

'Резерфорд' является целевым регистром

'Rm' является регистром, содержащим значение, которое будет смещено

'PTC' является регистром, содержащим длину сдвига, чтобы примениться к Rm значениям.

Только наименее существенный байт используется и может быть в диапазоне от 0 до 255.

'n' является длиной сдвига. Диапазон длин сдвига зависит от инструкции следующим образом:

ASR: длина Сдвига от 1 до 32

LSL: длина Сдвига от 0 до 31

LSR: длина Сдвига от 1 до 32

ROR: длина Сдвига от 1 до 31

Отметьте: МОВС-Рруд, Rm является привилегированным синтаксисом для ЛСЛС-Рруд, Rm, #0.

Работа

ASR, LSL, LSR, и ROR перемещают биты в Rm регистра налево или прямо числом из мест, определенных постоянным n или PTC регистра. RRX перемещает биты в Rm регистра направо 1. Во всех этих инструкциях результат пишется Резерфорду, но значение в Rm регистра остается неизменным. Для получения дополнительной информации на том, какой результат сгенерирован различными инструкциями (см. [операции Сдвига на странице 61](#)).

Ограничения

Не используйте или SP или PC.

Флаги условия

Если S определяется:

Эти инструкции обновляют флаги N и Z согласно результату флаг C обновляется к последнему переключенному на верхний регистр биту, кроме тех случаев, когда длина сдвига 0 (см. [Операции сдвига на странице 61](#)).

Примеры

ASR R7, R8, #9

LSLS R1, R2, #3 LSR R4, R5, #6 ROR R4, R5, R6 RRX R4, R5

; арифметический сдвиг прямо на 9 битов

; логический сдвиг уехал на 3 бита с обновлением флага; логический сдвиг прямо на 6 битов; вращайтесь прямо значением в нижнем байте R6; вращайтесь прямо с, расширяются

3.5.4 CLZ

Подсчитывает начальные нули.

Синтаксис

CLZ {cond} Резерфорд, Комната

где:

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),

'Резерфорд' является целевым регистром

'Rm' является регистром операнда

Работа

Инструкция CLZ считает число начальных нулей в значении Rm и возвращает результат в *Резерфорде*. Значение результата 32, если никакие биты не устанавливаются в исходном регистре, и нуле, если бит [31] устанавливается.

Ограничения

Не используйте или SP или PC.

Флаги условия

Эта инструкция не изменяет флаги.

Примеры

```
CLZ R4, R9
CLZNE R2, R3
```

3.5.5 CMP и CMN

Сравнить и сравнить отрицательный.

Синтаксис

```
CMP {cond} Rn, Operand2
CMN {cond} Rn, Operand2
```

где:

'*cond*' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),
'*Rn*' является регистром, содержащим первый операнд
'*Operand2*' является гибким вторым операндом (см. [Гибкий второй операнд на странице 59](#)) для опций.

Работа

Эти инструкции сравнивают значение в регистре с *operand2*. Они обновляют условие флагов на результате, но не пишут результат в регистр. Инструкция CMP вычитает значение *operand2* от значения в *Rn*. Это - то же самое как инструкция SUBS, за исключением того, что результат отбрасывается. Инструкция CMN добавляет значение *operand2* к значению в *Rn*. Это - то же самое как Инструкция ADDS, за исключением того, что результат отбрасывается.

Ограничения

В этих инструкциях:

Не используйте PC

Operand2 не должен быть SP

Флаги условия

Эти инструкции обновляют N, Z, C и V флагов согласно результату.

Примеры

```
CMP R2, R9
CMN R0, #6400 SP CMPGT, R7, LSL #2
```

3.5.6 MOV и MVN

Переместить и не перемещать.

Синтаксис

MOB {S} {cond} Резерфорд, Operand2

MOB {cond} Резерфорд, #imm16 MVN {S} {cond} Резерфорд, Operand2

где:

'S' является дополнительным суффиксом. Если S определяется, флаги кода условия обновляются на результат работы (см. [Условное выполнение на странице 64](#)).

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)),

'Резерфорд' является целевым регистром

'Operand2' является гибким вторым операндом (см. [Гибкий второй операнд на странице 59](#)) для опций.

'imm16' является любым значением в диапазоне 0-65535

Работа

Инструкция MOB копирует значение *operand2* в *Резерфорд*. Когда *operand2* в инструкции MOB

является регистром со сдвигом кроме LSL #0, привилегированный синтаксис является

соответствующей инструкцией сдвига:

ASR {S} {cond} Резерфорд, Rm, #n является привилегированным синтаксисом для MOB {S} {cond} Резерфорд, Rm, ASR #n

LSL {S} {cond} Резерфорд, Rm, #n является привилегированным синтаксисом для MOB {S} {cond} Резерфорд, Rm, LSL #n если n != 0

LSR {S} {cond} Резерфорд, Rm, #n является привилегированным синтаксисом для MOB {S} {cond} Резерфорд, Rm, LSR #n

ROR {S} {cond} Резерфорд, Rm, #n является привилегированным синтаксисом для MOB {S} {cond} Резерфорд, Rm, ROR #n

RRX {S} {cond} Резерфорд, Rm является привилегированным синтаксисом для MOB {S} {cond} Резерфорд, Rm, RRX

Кроме того, инструкция MOB разрешает дополнительные формы *operand2* как синонимы для инструкций сдвига:

MOB {S} {cond} Резерфорд, Rm, PTC ASR является синонимом для ASR {S} {cond} Резерфорд, Rm, PTC

MOB {S} {cond} Резерфорд, Rm, PTC LSL является синонимом для LSL {S} {cond} Резерфорд, Rm, PTC

MOB {S} {cond} Резерфорд, Rm, PTC LSR является синонимом для LSR {S} {cond} Резерфорд, Rm, PTC

MOB {S} {cond} Резерфорд, Rm, PTC ROR является синонимом для ROR {S} {cond} Резерфорд, Rm, PTC

См. [ASR, LSL, LSR, ROR, и RRX на странице 85](#). Инструкция MVN принимает значение *operand2*, выполняет поразрядное логическое НЕ работа на значение, и места результат в *Резерфорд*.

Отметьте: Инструкция MOVW обеспечивает ту же самую функцию как MOB, но ограничивается использованию операнд *imm16*.

Ограничения

Можно использовать SP и PC только в инструкции MOB, со следующими ограничениями:

Второй операнд должен быть регистром без сдвига

не следует определять суффикс S

Когда *Резерфорд* является PC в инструкции MOB:

Бит [0] из значения, записанного PC, игнорируется ответвление происходит с адресом, создаваемым, вызывая бит [0] из того значения к 0.

Отметьте: Хотя возможно использовать *MOB* в качестве команды перехода, ARM строго рекомендует использование из *BX* или инструкции *BLX*, чтобы перейти для мобильности программного обеспечения к набору команд ARM.

Флаги условия

Если S определяется, эти инструкции:

Обновите флаги N и Z согласно результату

МОЖЕТ обновить флаг C во время вычисления *operand2* (см. [Гибкий второй операнд на странице 59](#)).

Не влияет на V флаги

Пример

```
MOVS R11, #0x000B
```

```
MOB P1, #0xFA05 MOVS R10, R12 MOB P3, #23 MOB P8, SP MVNS R2, #0xF
```

; запишите значение 0x000B к R11, флаги обновляются

; запишите значение 0xFA05 к R1, флаги, не обновленные; запишите значение в R12 к R10, флаги обновляются; запишите значение 23 к R3; запишите значение указателя вершины стека к R8; запишите значение 0xFFFFFFFF0 (поразрядная инверсия 0xF); к R2 и флагам обновления

3.5.7 MOVT

Переместить вершину.

Синтаксис

```
MOVT {cond} Резерфорд, #imm16
```

где:

'*cond*' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#))

'*Резерфорд*' является целевым регистром

'*imm16*' является 16-разрядной непосредственной константой

Работа

MOVT пишет 16-разрядное непосредственное значение, *imm16*, к главному полуслову, *Резерфорд* [31:16], целевой регистр. Запись не влияет на *Резерфорд* [15:0]. MOB, пара инструкции MOVT позволяет Вам генерировать любую 32-разрядную константу.

Ограничения

Резерфорд не должен быть ни SP, ни PC.

Флаги условия

Эта инструкция не изменяет флаги.

Примеры

```
MOVT R3, #0xF123 ; запишите 0xF123 в верхнее полуслово R3,
```

; более низкое полуслово и APSR неизменны

3.5.8 REV REV16, REVSH, и RBIT

Обратные байты и обратные биты.

Синтаксис

op {cond} Резерфорд, Rn

где:

'op' является одним из:

REV: Обратный порядок байтов одним словом

REV16: Обратный порядок байтов в каждом полуслове независимо

REVSH: Обратный порядок байтов в нижнем полуслове, и знак расширяются на RBIT на 32

бита: Инвертируйте разрядный порядок в 32-разрядном слове

'cond' является дополнительным кодом условия (см. [Условное выполнение на странице 64](#)), 'Резерфорд' является целевым регистром

'Rn' является регистром, содержащим операнд

Работа

Используйте эти инструкции, чтобы изменить порядок байтов

данных:

REV: Преобразовывает также:

- 32-разрядные данные с обратным порядком байтов в данные с прямым порядком байтов
- или 32-разрядные данные с прямым порядком байтов в данные с обратным порядком байтов.

REV16: Преобразовывает также:

- 16-разрядные данные с обратным порядком байтов в данные с прямым порядком байтов
- или 16-разрядные данные с прямым порядком байтов в данные с обратным порядком байтов.

REVSH: Преобразовывает также:

- 16-разрядные данные с обратным порядком байтов со знаком в 32-разрядные данные с прямым порядком байтов со знаком
- 16-разрядные данные с прямым порядком байтов со знаком в 32-разрядные данные с обратным порядком байтов со знаком

Ограничения

Не используйте или SP или PC.

Флаги условия

Эти инструкции не изменяют флаги.

Примеры

REV R3, R7 ; обратный порядок байтов имеющий значение в R7 и записи это к R3

REV16 R0, R0; обратный порядок байтов каждого 16-разрядного полуслова в

R0 REVSH R0, R5; инвертируйте Полуслово Со знаком REVHS R3, R7; реверс с

Выше или То же самое условие

RBIT R7, R8 ; инвертируйте разрядный порядок имеющий значение в R8 и результате записи к R7

