Intrinsic Functions for SIMD Instructions [only Cortex-M4 and Cortex-M7]

Access to dedicated SIMD instructions. More...

Functions

uint32_t **__SADD8** (uint32_t val1, uint32_t val2)
GE setting quad 8-bit signed addition. More...

uint32_t **__QADD8** (uint32_t val1, uint32_t val2)
Q setting quad 8-bit saturating addition. More...

uint32_t **__SHADD8** (uint32_t val1, uint32_t val2)
Quad 8-bit signed addition with halved results. More...

uint32_t **__UADD8** (uint32_t val1, uint32_t val2)
GE setting quad 8-bit unsigned addition. More...

uint32_t **__UQADD8** (uint32_t val1, uint32_t val2)
Quad 8-bit unsigned saturating addition. More...

uint32_t **__UHADD8** (uint32_t val1, uint32_t val2)
Quad 8-bit unsigned addition with halved results. More...

uint32_t **__SSUB8** (uint32_t val1, uint32_t val2)
GE setting quad 8-bit signed subtraction. More...

uint32_t **__QSUB8** (uint32_t val1, uint32_t val2)
Q setting quad 8-bit saturating subtract. More...

uint32_t **__SHSUB8** (uint32_t val1, uint32_t val2)
Quad 8-bit signed subtraction with halved results. More...

uint32_t **__USUB8** (uint32_t val1, uint32_t val2)
GE setting quad 8-bit unsigned subtract. More...

uint32_t **__UQSUB8** (uint32_t val1, uint32_t val2)
Quad 8-bit unsigned saturating subtraction. More...

uint32_t **__UHSUB8** (uint32_t val1, uint32_t val2)
Quad 8-bit unsigned subtraction with halved results. More...

uint32_t **__SADD16** (uint32_t val1, uint32_t val2)
GE setting dual 16-bit signed addition. More...

uint32_t **__QADD16** (uint32_t val1, uint32_t val2)
Q setting dual 16-bit saturating addition. More...

uint32_t **__SHADD16** (uint32_t val1, uint32_t val2)
Dual 16-bit signed addition with halved results. More...

uint32_t **__UADD16** (uint32_t val1, uint32_t val2)
GE setting dual 16-bit unsigned addition. More...

uint32_t **__UQADD16** (uint32_t val1, uint32_t val2)
Dual 16-bit unsigned saturating addition. More...

uint32_t **__UHADD16** (uint32_t val1, uint32_t val2)
Dual 16-bit unsigned addition with halved results. More...

uint32_t **__SSUB16** (uint32_t val1, uint32_t val2)
GE setting dual 16-bit signed subtraction. More...

uint32_t **__QSUB16** (uint32_t val1, uint32_t val2)
Q setting dual 16-bit saturating subtract. More...

uint32_t **__SHSUB16** (uint32_t val1, uint32_t val2)
Dual 16-bit signed subtraction with halved results. More...

uint32_t **__USUB16** (uint32_t val1, uint32_t val2)
GE setting dual 16-bit unsigned subtract. More...

uint32_t **__UQSUB16** (uint32_t val1, uint32_t val2)
Dual 16-bit unsigned saturating subtraction. More...

uint32_t **__UHSUB16** (uint32_t val1, uint32_t val2)
Dual 16-bit unsigned subtraction with halved results. More...

uint32_t **__SASX** (uint32_t val1, uint32_t val2)
GE setting dual 16-bit addition and subtraction with exchange. More...

uint32_t **__QASX** (uint32_t val1, uint32_t val2)
Q setting dual 16-bit add and subtract with exchange. More...

uint32_t **__SHASX** (uint32_t val1, uint32_t val2)
Dual 16-bit signed addition and subtraction with halved results. More...

uint32_t **__UASX** (uint32_t val1, uint32_t val2)
GE setting dual 16-bit unsigned addition and subtraction with exchange. More...

uint32_t **__UQASX** (uint32_t val1, uint32_t val2)
Dual 16-bit unsigned saturating addition and subtraction with exchange. More...

uint32_t **__UHASX** (uint32_t val1, uint32_t val2)
Dual 16-bit unsigned addition and subtraction with halved results and exchange. More...

uint32_t **__SSAX** (uint32_t val1, uint32_t val2)
GE setting dual 16-bit signed subtraction and addition with exchange. More...

uint32_t **__QSAX** (uint32_t val1, uint32_t val2)
Q setting dual 16-bit subtract and add with exchange. More...

uint32_t **__SHSAX** (uint32_t val1, uint32_t val2)
Dual 16-bit signed subtraction and addition with halved results. More...

uint32_t **__USAX** (uint32_t val1, uint32_t val2)
GE setting dual 16-bit unsigned subtract and add with exchange. More...

uint32_t **__UQSAX** (uint32_t val1, uint32_t val2)
Dual 16-bit unsigned saturating subtraction and addition with exchange. More...

uint32_t **__UHSAX** (uint32_t val1, uint32_t val2)
Dual 16-bit unsigned subtraction and addition with halved results and exchange. More...

uint32_t **__USAD8** (uint32_t val1, uint32_t val2)
Unsigned sum of quad 8-bit unsigned absolute difference. More...

uint32_t **__USADA8** (uint32_t val1, uint32_t val2, uint32_t val3)
Unsigned sum of quad 8-bit unsigned absolute difference with 32-bit accumulate. More...

uint32_t **__SSAT16** (uint32_t val1, const uint32_t val2)
Q setting dual 16-bit saturate. More...

uint32_t **__USAT16** (uint32_t val1, const uint32_t val2)
Q setting dual 16-bit unsigned saturate. More...

uint32_t **__UXTB16** (uint32_t val)
Dual extract 8-bits and zero-extend to 16-bits. More...

uint32_t **__UXTAB16** (uint32_t val1, uint32_t val2)
Extracted 16-bit to 32-bit unsigned addition. More...

uint32_t **__SXTB16** (uint32_t val)
Dual extract 8-bits and sign extend each to 16-bits. More...

uint32_t **__SXTB16_RORn** (uint32_t val, uint32_r rotate)
Rotate right, dual extract 8-bits and sign extend each to 16-bits. More...

uint32_t **__SXTAB16** (uint32_t val1, uint32_t val2)
Dual extracted 8-bit to 16-bit signed addition. More...

uint32_t **__SMUAD** (uint32_t val1, uint32_t val2)
Q setting sum of dual 16-bit signed multiply. More...

uint32_t **__SMUADX** (uint32_t val1, uint32_t val2)
Q setting sum of dual 16-bit signed multiply with exchange. More...

uint32_t **__SMMLA** (int32_t val1, int32_t val2, int32_t val3)
32-bit signed multiply with 32-bit truncated accumulator. More...

uint32_t **__SMLAD** (uint32_t val1, uint32_t val2, uint32_t val3)
Q setting dual 16-bit signed multiply with single 32-bit accumulator. More...

uint32_t **__SMLADX** (uint32_t val1, uint32_t val2, uint32_t val3)
Q setting pre-exchanged dual 16-bit signed multiply with single 32-bit accumulator. More...

uint64_t **__SMLALD** (uint32_t val1, uint32_t val2, uint64_t val3)
Dual 16-bit signed multiply with single 64-bit accumulator. More...

unsigned long long **__SMLALDX** (uint32_t val1, uint32_t val2, unsigned long long val3)
Dual 16-bit signed multiply with exchange with single 64-bit accumulator. More...

uint32_t	**__SMUSD** (uint32_t val1, uint32_t val2) Dual 16-bit signed multiply returning difference. More...
uint32_t	**__SMUSDX** (uint32_t val1, uint32_t val2) Dual 16-bit signed multiply with exchange returning difference. More...
uint32_t	**__SMLSD** (uint32_t val1, uint32_t val2, uint32_t val3) Q setting dual 16-bit signed multiply subtract with 32-bit accumulate. More...
uint32_t	**__SMLSDX** (uint32_t val1, uint32_t val2, uint32_t val3) Q setting dual 16-bit signed multiply with exchange subtract with 32-bit accumulate. More...
uint64_t	**__SMLSLD** (uint32_t val1, uint32_t val2, uint64_t val3) Q setting dual 16-bit signed multiply subtract with 64-bit accumulate. More...
unsigned long long	**__SMLSLDX** (uint32_t val1, uint32_t val2, unsigned long long val3) Q setting dual 16-bit signed multiply with exchange subtract with 64-bit accumulate. More...
uint32_t	**__SEL** (uint32_t val1, uint32_t val2) Select bytes based on GE bits. More...
uint32_t	**__QADD** (uint32_t val1, uint32_t val2) Q setting saturating add. More...
uint32_t	**__QSUB** (uint32_t val1, uint32_t val2) Q setting saturating subtract. More...
uint32_t	**__PKHBT** (uint32_t val1, uint32_t val2, uint32_t val3) Halfword packing instruction. Combines bits[15:0] of *val1* with bits[31:16] of *val2* levitated with the *val3*. More...
uint32_t	**__PKHTB** (uint32_t val1, uint32_t val2, uint32_t val3) Halfword packing instruction. Combines bits[31:16] of *val1* with bits[15:0] of *val2* right-shifted with the *val3*. More...

Description

Access to dedicated SIMD instructions.

Single Instruction Multiple Data (SIMD) extensions are provided **only for Cortex-M4 and Cortex-M7 cores** to simplify development of application software. SIMD extensions increase the processing capability without materially increasing the power consumption. The SIMD extensions are completely transparent to the operating system (OS), allowing existing OS ports to be used.

SIMD Features:

- Simultaneous computation of 2x16-bit or 4x8-bit operands
- Fractional arithmetic
- User definable saturation modes (arbitrary word-width)
- Dual 16x16 multiply-add/subtract 32x32 fractional MAC
- Simultaneous 8/16-bit select operations
- Performance up to 3.2 GOPS at 800MHz
- Performance is achieved with a "near zero" increase in power consumption on a typical implementation

Examples:

Addition: Add two values using SIMD function

```
uint32_t add_halfwords(uint32_t val1, uint32_t val2)
{
  return __SADD16(val1, val2);
}
```

Subtraction: Subtract two values using SIMD function

```
uint32_t sub_halfwords(uint32_t val1, uint32_t val2)
{
  return __SSUB16(val1, val2);
}
```

Multiplication: Performing a multiplication using SIMD function

```
uint32_t dual_mul_add_products(uint32_t val1, uint32_t val2)
{
  return __SMUAD(val1, val2);
}
```

Function Documentation

uint32_t __PKHBT (uint32_t val1,
uint32_t val2,
uint32_t val3
)

Halfword packing instruction. Combines bits[15:0] of *val1* with bits[31:16] of *val2* levitated with the *val3*.

Combine a halfword from one register with a halfword from another register. The second argument can be left-shifted before extraction of the halfword. The registers PC and SP are not allowed as arguments. This instruction does not change the flags.

Parameters

val1 first 16-bit operands

val2 second 16-bit operands

val3 value for left-shifting *val2*. Value range [0..31].

Returns

the combination of halfwords.

Operation:

```
res[15:0]  = val1[15:0]
res[31:16] = val2[31:16]<<val3
```

```
uint32_t __PKHTB ( uint32_t val1,
                   uint32_t val2,
                   uint32_t val3
                 )
```

Halfword packing instruction. Combines bits[31:16] of *val1* with bits[15:0] of *val2* right-shifted with the *val3*.

Combines a halfword from one register with a halfword from another register. The second argument can be right-shifted before extraction of the halfword. The registers PC and SP are not allowed as arguments. This instruction does not change the flags.

Parameters

val1 second 16-bit operands

val2 first 16-bit operands

val3 value for right-shifting *val2*. Value range [1..32].

Returns

the combination of halfwords.

Operation:

```
res[15:0]  = val2[15:0]>>val3
res[31:16] = val1[31:16]
```

```
uint32_t __QADD ( uint32_t val1,
                  uint32_t val2
                )
```

Q setting saturating add.

This function enables you to obtain the saturating add of two integers.
The Q bit is set if the operation saturates.

Parameters

val1 first summand of the saturating add operation.

val2 second summand of the saturating add operation.

Returns

the saturating addition of val1 and val2.

Operation:

```
res[31:0] = SAT(val1 + SAT(val2))
```

```
uint32_t __QADD16 ( uint32_t val1,
                    uint32_t val2
                  )
```

Q setting dual 16-bit saturating addition.

This function enables you to perform two 16-bit integer arithmetic additions in parallel, saturating the results to the 16-bit signed integer range $-2^{15} <= x <= 2^{15} - 1$.

Parameters

val1 first two 16-bit summands.

val2 second two 16-bit summands.

Returns

- the saturated addition of the low halfwords, in the low halfword of the return value.
- the saturated addition of the high halfwords, in the high halfword of the return value.

The returned results are saturated to the 16-bit signed integer range $-2^{15} <= x <= 2^{15} - 1$

Operation:

```
res[15:0]  = val1[15:0]  + val2[15:0]
res[31:16] = val1[31:16] + val2[31:16]
```

```
uint32_t __QADD8 ( uint32_t val1,
                   uint32_t val2
                 )
```

Q setting quad 8-bit saturating addition.

This function enables you to perform four 8-bit integer additions, saturating the results to the 8-bit signed integer range $-2^{7} <= x <= 2^{7} - 1$.

Parameters

val1 first four 8-bit summands.

val2 second four 8-bit summands.

Returns

- the saturated addition of the first byte of each operand in the first byte of the return value.
- the saturated addition of the second byte of each operand in the second byte of the return value.
- the saturated addition of the third byte of each operand in the third byte of the return value.
- the saturated addition of the fourth byte of each operand in the fourth byte of the return value.

The returned results are saturated to the 16-bit signed integer range $-2^{7} <= x <= 2^{7} - 1$.

Operation:

```
res[7:0]   = val1[7:0]   + val2[7:0]
res[15:8]  = val1[15:8]  + val2[15:8]
res[23:16] = val1[23:16] + val2[23:16]
res[31:24] = val1[31:24] + val2[31:24]
```

```
uint32_t __QASX ( uint32_t val1,
                  uint32_t val2
                )
```

Q setting dual 16-bit add and subtract with exchange.

This function enables you to exchange the halfwords of the one operand, then add the high halfwords and subtract the low halfwords, saturating the results to the 16-bit signed integer range $-2^{15} <= x <= 2^{15} - 1$.

Parameters

val1 first operand for the subtraction in the low halfword, and the first operand for the addition in the high halfword.

val2 second operand for the subtraction in the high halfword, and the second operand for the addition in the low halfword.

Returns

- the saturated subtraction of the high halfword in the second operand from the low halfword in the first operand, in the low halfword of the return value.
- the saturated addition of the high halfword in the first operand and the low halfword in the second operand, in the high halfword of the return value.

The returned results are saturated to the 16-bit signed integer range $-2^{15} <= x <= 2^{15} - 1$.

Operation:

```
res[15:0]  = val1[15:0]  - val2[31:16]
res[31:16] = val1[31:16] + val2[15:0]
```

```
uint32_t __QSAX ( uint32_t val1,
                  uint32_t val2
                )
```

Q setting dual 16-bit subtract and add with exchange.

This function enables you to exchange the halfwords of one operand, then subtract the high halfwords and add the low halfwords, saturating the results to the 16-bit signed integer range $-2^{15} <= x <= 2^{15} - 1$.

Parameters

val1 first operand for the addition in the low halfword, and the first operand for the subtraction in the high halfword.

val2 second operand for the addition in the high halfword, and the second operand for the subtraction in the low halfword.

Returns

- the saturated addition of the low halfword of the first operand and the high halfword of the second operand, in the low halfword of the return value.
- the saturated subtraction of the low halfword of the second operand from the high halfword of the first operand, in the high halfword of the return value.

The returned results are saturated to the 16-bit signed integer range $-2^{15} <= x <= 2^{15} - 1$.

Operation:

```
res[15:0]  = val1[15:0]  + val2[31:16]
res[31:16] = val1[31:16] - val2[15:0]
```

```
uint32_t __QSUB ( uint32_t val1,
                  uint32_t val2
                )
```

Q setting saturating subtract.

This function enables you to obtain the saturating subtraction of two integers.
The Q bit is set if the operation saturates.

Parameters

val1 minuend of the saturating subtraction operation.

val2 subtrahend of the saturating subtraction operation.

Returns

the saturating subtraction of val1 and val2.

Operation:

```
res[31:0] = SAT(val1 - SAT(val2))
```

```
uint32_t __QSUB16 ( uint32_t val1,
                    uint32_t val2
                  )
```

Q setting dual 16-bit saturating subtract.

This function enables you to perform two 16-bit integer subtractions, saturating the results to the 16-bit signed integer range $-2^{15} <= x <= 2^{15} - 1$.

Parameters

val1 first two 16-bit operands.

val2 second two 16-bit operands.

Returns

- the saturated subtraction of the low halfword in the second operand from the low halfword in the first operand, in the low halfword of the returned result.
- the saturated subtraction of the high halfword in the second operand from the high halfword in the first operand, in the high halfword of the returned result.

The returned results are saturated to the 16-bit signed integer range $-2^{15} <= x <= 2^{15} - 1$.

Operation:

```
res[15:0]  = val1[15:0]  - val2[15:0]
res[31:16] = val1[31:16] - val2[31:16]
```

```
uint32_t __QSUB8 ( uint32_t val1,
                   uint32_t val2
                 )
```

Q setting quad 8-bit saturating subtract.

This function enables you to perform four 8-bit integer subtractions, saturating the results to the 8-bit signed integer range $-2^7 <= x <= 2^7 - 1$.

Parameters

val1 first four 8-bit operands.

val2 second four 8-bit operands.

Returns

- the subtraction of the first byte in the second operand from the first byte in the first operand, in the first bytes of the return value.
- the subtraction of the second byte in the second operand from the second byte in the first operand, in the second byte of the return value.
- the subtraction of the third byte in the second operand from the third byte in the first operand, in the third byte of the return value.
- the subtraction of the fourth byte in the second operand from the fourth byte in the first operand, in the fourth byte of the return value.

The returned results are saturated to the 8-bit signed integer range $-2^7 <= x <= 2^7 - 1$.

Operation:

```
res[7:0]   = val1[7:0]   - val2[7:0]
res[15:8]  = val1[15:8]  - val2[15:8]
res[23:16] = val1[23:16] - val2[23:16]
res[31:24] = val1[31:24] - val2[31:24]
```

```
uint32_t __SADD16 ( uint32_t  val1,
                    uint32_t  val2
                  )
```

GE setting dual 16-bit signed addition.

This function enables you to perform two 16-bit signed integer additions.
The GE bits in the APSR are set according to the results of the additions.

Parameters

val1 first two 16-bit summands.

val2 second two 16-bit summands.

Returns

- the addition of the low halfwords in the low halfword of the return value.
- the addition of the high halfwords in the high halfword of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[15:0] >= 0 then APSR.GE[1:0] = 11 else 00
- if res[31:16] >= 0 then APSR.GE[3:2] = 11 else 00

Operation:

```
res[15:0]  = val1[15:0]  + val2[15:0]
res[31:16] = val1[31:16] + val2[31:16]
```

```
uint32_t __SADD8 ( uint32_t val1,
                   uint32_t val2
                 )
```

GE setting quad 8-bit signed addition.

This function performs four 8-bit signed integer additions. The GE bits of the APSR are set according to the results of the additions.

Parameters

val1 first four 8-bit summands.

val2 second four 8-bit summands.

Returns

- the addition of the first bytes from each operand, in the first byte of the return value.
- the addition of the second bytes of each operand, in the second byte of the return value.
- the addition of the third bytes of each operand, in the third byte of the return value.
- the addition of the fourth bytes of each operand, in the fourth byte of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[7:0] >= 0 then APSR.GE[0] = 1 else 0
- if res[15:8] >= 0 then APSR.GE[1] = 1 else 0
- if res[23:16] >= 0 then APSR.GE[2] = 1 else 0
- if res[31:24] >= 0 then APSR.GE[3] = 1 else 0

Operation:

```
res[7:0]   = val1[7:0]   + val2[7:0]
res[15:8]  = val1[15:8]  + val2[15:8]
res[23:16] = val1[23:16] + val2[23:16]
res[31:24] = val1[31:24] + val2[31:24]
```

```
uint32_t __SASX ( uint32_t  val1,
                  uint32_t  val2
                )
```

GE setting dual 16-bit addition and subtraction with exchange.

This function inserts an SASX instruction into the instruction stream generated by the compiler. It enables you to exchange the halfwords of the second operand, add the high halfwords and subtract the low halfwords.
The GE bits in the APRS are set according to the results.

Parameters

val1 first operand for the subtraction in the low halfword, and the first operand for the addition in the high halfword.

val2 second operand for the subtraction in the high halfword, and the second operand for the addition in the low halfword.

Returns

- the subtraction of the high halfword in the second operand from the low halfword in the first operand, in the low halfword of the return value.
- the addition of the high halfword in the first operand and the low halfword in the second operand, in the high halfword of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[15:0] >= 0 then APSR.GE[1:0] = 11 else 00
- if res[31:16] >= 0 then APSR.GE[3:2] = 11 else 00

Operation:

```
res[15:0]  = val1[15:0]  - val2[31:16]
res[31:16] = val1[31:16] + val2[15:0]
```

```
uint32_t __SEL ( uint32_t  val1,
                 uint32_t  val2
               )
```

Select bytes based on GE bits.

This function inserts a SEL instruction into the instruction stream generated by the compiler. It enables you to select bytes from the input parameters, whereby the bytes that are selected depend upon the results of previous SIMD instruction function. The results of previous SIMD instruction function are represented by the Greater than or Equal flags in the Application Program Status Register (APSR). The __SEL function works equally well on both halfword and byte operand function results. This is because halfword operand operations set two (duplicate) GE bits per value.

Parameters

val1 four selectable 8-bit values.

val2 four selectable 8-bit values.

Returns

The function selects bytes from the input parameters and returns them in the return value, res, according to the following criteria:

- if APSR.GE[0] == 1 then res[7:0] = val1[7:0] else res[7:0] = val2[7:0]
- if APSR.GE[1] == 1 then res[15:8] = val1[15:8] else res[15:8] = val2[15:8]
- if APSR.GE[2] == 1 then res[23:16] = val1[23:16] else res[23:16] = val2[23:16]
- if APSR.GE[3] == 1 then res[31;24] = val1[31:24] else res = val2[31:24]

```
uint32_t __SHADD16 ( uint32_t  val1,
                     uint32_t  val2
                   )
```

Dual 16-bit signed addition with halved results.

This function enables you to perform two signed 16-bit integer additions, halving the results.

Parameters

val1 first two 16-bit summands.

val2 second two 16-bit summands.

Returns

- the halved addition of the low halfwords, in the low halfword of the return value.
- the halved addition of the high halfwords, in the high halfword of the return value.

Operation:

```
res[15:0]  = val1[15:0]  + val2[15:0]  >> 1
res[31:16] = val1[31:16] + val2[31:16] >> 1
```

```
uint32_t __SHADD8 ( uint32_t val1,
                    uint32_t val2
                  )
```

Quad 8-bit signed addition with halved results.

This function enables you to perform four signed 8-bit integer additions, halving the results.

Parameters

val1 first four 8-bit summands.

val2 second four 8-bit summands.

Returns

- the halved addition of the first bytes from each operand, in the first byte of the return value.
- the halved addition of the second bytes from each operand, in the second byte of the return value.
- the halved addition of the third bytes from each operand, in the third byte of the return value.
- the halved addition of the fourth bytes from each operand, in the fourth byte of the return value.

Operation:

```
res[7:0]   = val1[7:0]   + val2[7:0]  >> 1
res[15:8]  = val1[15:8]  + val2[15:8] >> 1
res[23:16] = val1[23:16] + val2[23:16] >> 1
res[31:24] = val1[31:24] + val2[31:24] >> 1
```

```
uint32_t __SHASX ( uint32_t val1,
                   uint32_t val2
                 )
```

Dual 16-bit signed addition and subtraction with halved results.

This function enables you to exchange the two halfwords of one operand, perform one signed 16-bit integer addition and one signed 16-bit subtraction, and halve the results.

Parameters

val1 first 16-bit operands.

val2 second 16-bit operands.

Returns

- the halved subtraction of the high halfword in the second operand from the low halfword in the first operand, in the low halfword of the return value.
- the halved subtraction of the low halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

Operation:

```
res[15:0]  = (val1[15:0]  - val2[31:16]) >> 1
res[31:16] = (val1[31:16] - val2[15:0] ) >> 1
```

```
uint32_t __SHSAX ( uint32_t val1,
                   uint32_t val2
                 )
```

Dual 16-bit signed subtraction and addition with halved results.

This function enables you to exchange the two halfwords of one operand, perform one signed 16-bit integer subtraction and one signed 16-bit addition, and halve the results.

Parameters

val1 first 16-bit operands.

val2 second 16-bit operands.

Returns

- the halved addition of the low halfword in the first operand and the high halfword in the second operand, in the low halfword of the return value.
- the halved subtraction of the low halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

Operation:

```
res[15:0]  = (val1[15:0]  + val2[31:16]) >> 1
res[31:16] = (val1[31:16] - val2[15:0] ) >> 1
```

```
uint32_t __SHSUB16 ( uint32_t val1,
                     uint32_t val2
                   )
```

Dual 16-bit signed subtraction with halved results.

This function enables you to perform two signed 16-bit integer subtractions, halving the results.

Parameters

val1 first two 16-bit operands.

val2 second two 16-bit operands.

Returns

- the halved subtraction of the low halfword in the second operand from the low halfword in the first operand, in the low halfword of the returned result.
- the halved subtraction of the high halfword in the second operand from the high halfword in the first operand, in the high halfword of the returned result.

Operation:

```
res[15:0]  = val1[15:0]  - val2[15:0]   >> 1
res[31:16] = val1[31:16] - val2[31:16]  >> 1
```

```
uint32_t __SHSUB8 ( uint32_t val1,
                    uint32_t val2
                  )
```

Quad 8-bit signed subtraction with halved results.

This function enables you to perform four signed 8-bit integer subtractions, halving the results.

Parameters

val1 first four 8-bit operands.

val2 second four 8-bit operands.

Returns

- the halved subtraction of the first byte in the second operand from the first byte in the first operand, in the first bytes of the return value.
- the halved subtraction of the second byte in the second operand from the second byte in the first operand, in the second byte of the return value.
- the halved subtraction of the third byte in the second operand from the third byte in the first operand, in the third byte of the return value.
- the halved subtraction of the fourth byte in the second operand from the fourth byte in the first operand, in the fourth byte of the return value.

Operation:

```
res[7:0]   = val1[7:0]   - val2[7:0]   >> 1
res[15:8]  = val1[15:8]  - val2[15:8]  >> 1
res[23:16] = val1[23:16] - val2[23:16] >> 1
res[31:24] = val1[31:24] - val2[31:24] >> 1
```

```
uint32_t __SMLAD ( uint32_t val1,
                   uint32_t val2,
                   uint32_t val3
                 )
```

Q setting dual 16-bit signed multiply with single 32-bit accumulator.

This function enables you to perform two signed 16-bit multiplications, adding both results to a 32-bit accumulate operand.
The Q bit is set if the addition overflows. Overflow cannot occur during the multiplications.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

val3 accumulate value.

Returns

the product of each multiplication added to the accumulate value, as a 32-bit integer.

Operation:

```
p1 = val1[15:0]  * val2[15:0]
p2 = val1[31:16] * val2[31:16]
res[31:0] = p1 + p2 + val3[31:0]
```

```
uint32_t __SMLADX ( uint32_t val1,
                    uint32_t val2,
                    uint32_t val3
                  )
```

Q setting pre-exchanged dual 16-bit signed multiply with single 32-bit accumulator.

This function enables you to perform two signed 16-bit multiplications with exchanged halfwords of the second operand, adding both results to a 32-bit accumulate operand.
The Q bit is set if the addition overflows. Overflow cannot occur during the multiplications.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

val3 accumulate value.

Returns

the product of each multiplication with exchanged halfwords of the second operand added to the accumulate value, as a 32-bit integer.

Operation:

```
p1 = val1[15:0]  * val2[31:16]
p2 = val1[31:16] * val2[15:0]
res[31:0] = p1 + p2 + val3[31:0]
```

```
uint64_t __SMLALD ( uint32_t val1,
                    uint32_t val2,
                    uint64_t val3
                  )
```

Dual 16-bit signed multiply with single 64-bit accumulator.

This function enables you to perform two signed 16-bit multiplications, adding both results to a 64-bit accumulate operand. Overflow is only possible as a result of the 64-bit addition. This overflow is not detected if it occurs. Instead, the result wraps around modulo 2^{64}.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

val3 accumulate value.

Returns

the product of each multiplication added to the accumulate value.

Operation:

```
p1 = val1[15:0]  * val2[15:0]
p2 = val1[31:16] * val2[31:16]
sum = p1 + p2 + val3[63:32][31:0]
res[63:32] = sum[63:32]
res[31:0]  = sum[31:0]
```

```
unsigned long long __SMLALDX ( uint32_t          val1,
                               uint32_t          val2,
                               unsigned long long val3
                             )
```

Dual 16-bit signed multiply with exchange with single 64-bit accumulator.

This function enables you to exchange the halfwords of the second operand, and perform two signed 16-bit multiplications, adding both results to a 64-bit accumulate operand. Overflow is only possible as a result of the 64-bit addition. This overflow is not detected if it occurs. Instead, the result wraps around modulo2^{64}.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

val3 accumulate value.

Returns

the product of each multiplication added to the accumulate value.

Operation:

```
p1 = val1[15:0]  * val2[31:16]
p2 = val1[31:16] * val2[15:0]
sum = p1 + p2 + val3[63:32][31:0]
res[63:32] = sum[63:32]
res[31:0] = sum[31:0]
```

```
uint32_t __SMLSD ( uint32_t val1,
                   uint32_t val2,
                   uint32_t val3
                 )
```

Q setting dual 16-bit signed multiply subtract with 32-bit accumulate.

This function enables you to perform two 16-bit signed multiplications, take the difference of the products, subtracting the high halfword product from the low halfword product, and add the difference to a 32-bit accumulate operand.
The Q bit is set if the accumulation overflows. Overflow cannot occur during the multiplications or the subtraction.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

val3 accumulate value.

Returns

the difference of the product of each multiplication, added to the accumulate value.

Operation:

```
p1 = val1[15:0]  * val2[15:0]
p2 = val1[31:16] * val2[31:16]
res[31:0] = p1 - p2 + val3[31:0]
```

```
uint32_t __SMLSDX ( uint32_t val1,
                    uint32_t val2,
                    uint32_t val3
                  )
```

Q setting dual 16-bit signed multiply with exchange subtract with 32-bit accumulate.

This function enables you to exchange the halfwords in the second operand, then perform two 16-bit signed multiplications. The difference of the products is added to a 32-bit accumulate operand. The Q bit is set if the addition overflows. Overflow cannot occur during the multiplications or the subtraction.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

val3 accumulate value.

Returns

the difference of the product of each multiplication, added to the accumulate value.

Operation:

```
p1 = val1[15:0]  * val2[31:16]
p2 = val1[31:16] * val2[15:0]
res[31:0] = p1 - p2 + val3[31:0]
```

```
uint64_t __SMLSLD ( uint32_t val1,
                    uint32_t val2,
                    uint64_t val3
                  )
```

Q setting dual 16-bit signed multiply subtract with 64-bit accumulate.

This function It enables you to perform two 16-bit signed multiplications, take the difference of the products, subtracting the high halfword product from the low halfword product, and add the difference to a 64-bit accumulate operand. Overflow cannot occur during the multiplications or the subtraction. Overflow can occur as a result of the 64-bit addition, and this overflow is not detected. Instead, the result wraps round to modulo2^{64}.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

val3 accumulate value.

Returns

the difference of the product of each multiplication, added to the accumulate value.

Operation:

```
p1 = val1[15:0]  * val2[15:0]
p2 = val1[31:16] * val2[31:16]
res[63:0] = p1 - p2 + val3[63:0]
```

```
unsigned long long __SMLSLDX ( uint32_t          val1,
                               uint32_t          val2,
                               unsigned long long val3
                             )
```

Q setting dual 16-bit signed multiply with exchange subtract with 64-bit accumulate.

This function enables you to exchange the halfwords of the second operand, perform two 16-bit multiplications, adding the difference of the products to a 64-bit accumulate operand. Overflow cannot occur during the multiplications or the subtraction. Overflow can occur as a result of the 64-bit addition, and this overflow is not detected. Instead, the result wraps round to modulo 2^{64}.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

val3 accumulate value.

Returns

the difference of the product of each multiplication, added to the accumulate value.

Operation:

```
p1 = val1[15:0]  * val2[31:16]
p2 = val1[31:16] * val2[15:0]
res[63:0] = p1 - p2 + val3[63:0]
```

```
uint32_t __SMMLA ( int32_t val1,
                   int32_t val2,
                   int32_t val3
                 )
```

32-bit signed multiply with 32-bit truncated accumulator.

This function enables you to perform a signed 32-bit multiplications, adding the most significant 32 bits of the 64-bit result to a 32-bit accumulate operand.

Parameters

val1 first operand for multiplication.

val2 second operand for multiplication.

val3 accumulate value.

Returns

the product of multiplication (most significant 32 bits) is added to the accumulate value, as a 32-bit integer.

Operation:

```
p = val1 * val2
res[31:0] = p[61:32] + val3[31:0]
```

```
uint32_t __SMUAD ( uint32_t  val1,
                   uint32_t  val2
                 )
```

Q setting sum of dual 16-bit signed multiply.

This function enables you to perform two 16-bit signed multiplications, adding the products together. The Q bit is set if the addition overflows.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

Returns

the sum of the products of the two 16-bit signed multiplications.

Operation:

```
p1 = val1[15:0]  * val2[15:0]
p2 = val1[31:16] * val2[31:16]
res[31:0] = p1 + p2
```

```
uint32_t __SMUADX ( uint32_t  val1,
                    uint32_t  val2
                  )
```

Q setting sum of dual 16-bit signed multiply with exchange.

This function enables you to perform two 16-bit signed multiplications with exchanged halfwords of the second operand, adding the products together.
The Q bit is set if the addition overflows.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

Returns

the sum of the products of the two 16-bit signed multiplications with exchanged halfwords of the second operand.

Operation:

```
p1 = val1[15:0]  * val2[31:16]
p2 = val1[31:16] * val2[15:0]
res[31:0] = p1 + p2
```

```
uint32_t __SMUSD ( uint32_t val1,
                   uint32_t val2
                 )
```

Dual 16-bit signed multiply returning difference.

This function enables you to perform two 16-bit signed multiplications, taking the difference of the products by subtracting the high halfword product from the low halfword product.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

Returns

the difference of the products of the two 16-bit signed multiplications.

Operation:

```
p1 = val1[15:0]  * val2[15:0]
p2 = val1[31:16] * val2[31:16]
res[31:0] = p1 - p2
```

```
uint32_t __SMUSDX ( uint32_t val1,
                    uint32_t val2
                  )
```

Dual 16-bit signed multiply with exchange returning difference.

This function enables you to perform two 16-bit signed multiplications, subtracting one of the products from the other. The halfwords of the second operand are exchanged before performing the arithmetic. This produces top * bottom and bottom * top multiplication.

Parameters

val1 first 16-bit operands for each multiplication.

val2 second 16-bit operands for each multiplication.

Returns

the difference of the products of the two 16-bit signed multiplications.

Operation:

```
p1 = val1[15:0]  * val2[31:16]
p2 = val1[31:16] * val2[15:0]
res[31:0] = p1 - p2
```

```
uint32_t __SSAT16 ( uint32_t        val1,
                    const uint32_t  val2
                  )
```

Q setting dual 16-bit saturate.

This function enables you to saturate two signed 16-bit values to a selected signed range.
The Q bit is set if either operation saturates.

Parameters

val1 two signed 16-bit values to be saturated.

val2 bit position for saturation, an integral constant expression in the range 1 to 16.

Returns

the sum of the absolute differences of the following bytes, added to the accumulation value:

- the signed saturation of the low halfword in *val1*, saturated to the bit position specified in *val2* and returned in the low halfword of the return value.
- the signed saturation of the high halfword in *val1*, saturated to the bit position specified in *val2* and returned in the high halfword of the return value.

Operation:

```
Saturate halfwords in val1 to the signed range specified by the bit position in val2
```

```
uint32_t __SSAX ( uint32_t  val1,
                  uint32_t  val2
                )
```

GE setting dual 16-bit signed subtraction and addition with exchange.

This function enables you to exchange the two halfwords of one operand and perform one 16-bit integer subtraction and one 16-bit addition.
The GE bits in the APSR are set according to the results.

Parameters

val1 first operand for the addition in the low halfword, and the first operand for the subtraction in the high halfword.

val2 second operand for the addition in the high halfword, and the second operand for the subtraction in the low halfword.

Returns

- the addition of the low halfword in the first operand and the high halfword in the second operand, in the low halfword of the return value.
- the subtraction of the low halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[15:0] >= 0 then APSR.GE[1:0] = 11 else 00
- if res[31:16] >= 0 then APSR.GE[3:2] = 11 else 00

Operation:

```
res[15:0]  = val1[15:0]  + val2[31:16]
res[31:16] = val1[31:16] - val2[15:0]
```

```
uint32_t __SSUB16 ( uint32_t val1,
                    uint32_t val2
                  )
```

GE setting dual 16-bit signed subtraction.

This function enables you to perform two 16-bit signed integer subtractions.
The GE bits in the APSR are set according to the results.

Parameters

val1 first two 16-bit operands of each subtraction.

val2 second two 16-bit operands of each subtraction.

Returns

- the subtraction of the low halfword in the second operand from the low halfword in the first operand, in the low halfword of the return value.
- the subtraction of the high halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If

- res is the return value, then:
- if res[15:0] >= 0 then APSR.GE[1:0] = 11 else 00
- if res[31:16] >= 0 then APSR.GE[3:2] = 11 else 00

Operation:

```
res[15:0]  = val1[15:0]  - val2[15:0]
res[31:16] = val1[31:16] - val2[31:16]
```

```
uint32_t __SSUB8 ( uint32_t val1,
                   uint32_t val2
                 )
```

GE setting quad 8-bit signed subtraction.

This function enables you to perform four 8-bit signed integer subtractions.
The GE bits in the APSR are set according to the results.

Parameters

val1 first four 8-bit operands of each subtraction.

val2 second four 8-bit operands of each subtraction.

Returns

- the subtraction of the first byte in the second operand from the first byte in the first operand, in the first bytes of the return value.
- the subtraction of the second byte in the second operand from the second byte in the first operand, in the second byte of the return value.
- the subtraction of the third byte in the second operand from the third byte in the first operand, in the third byte of the return value.
- the subtraction of the fourth byte in the second operand from the fourth byte in the first operand, in the fourth byte of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[8:0] >= 0 then APSR.GE[0] = 1 else 0
- if res[15:8] >= 0 then APSR.GE[1] = 1 else 0
- if res[23:16] >= 0 then APSR.GE[2] = 1 else 0
- if res[31:24] >= 0 then APSR.GE[3] = 1 else 0

Operation:

```
res[7:0]   = val1[7:0]   - val2[7:0]
res[15:8]  = val1[15:8]  - val2[15:8]
res[23:16] = val1[23:16] - val2[23:16]
res[31:24] = val1[31:24] - val2[31:24]
```

```
uint32_t __SXTAB16 ( uint32_t val1,
                     uint32_t val2
                   )
```

Dual extracted 8-bit to 16-bit signed addition.

This function enables you to extract two 8-bit values from the second operand (at bit positions [7:0] and [23:16]), sign-extend them to 16-bits each, and add the results to the first operand.

Parameters

val1 values added to the zero-extended to 16-bit values.

val2 two 8-bit values to be extracted and zero-extended.

Returns

the addition of *val1* and *val2*, where the 8-bit values in val2[7:0] and val2[23:16] have been extracted and sign-extended prior to the addition.

Operation:

```
res[15:0]  = val1[15:0]  + SignExtended(val2[7:0])
res[31:16] = val1[31:16] + SignExtended(val2[23:16])
```

uint32_t __SXTB16 (uint32_t val)

Dual extract 8-bits and sign extend each to 16-bits.

This function enables you to extract two 8-bit values from an operand and sign-extend them to 16 bits each.

Parameters

val two 8-bit values in val[7:0] and val[23:16] to be sign-extended.

Returns

the 8-bit values sign-extended to 16-bit values.

- sign-extended value of val[7:0] in the low halfword of the return value.
- sign-extended value of val[23:16] in the high halfword of the return value.

Operation:

```
res[15:0]  = SignExtended(val[7:0]
res[31:16] = SignExtended(val[23:16]
```

uint32_t __SXTB16_RORn (uint32_t val,
uint32_r rotate
)

Rotate right, dual extract 8-bits and sign extend each to 16-bits.

This function enables you to rotate an operand by 8/16/24 bit, extract two 8-bit values and sign-extend them to 16 bits each.

Parameters

val two 8-bit values in val[7:0] and val[23:16] to be sign-extended.

rotate number of bits to rotate val. Only 8,16 and 24 are accepted

Returns

the 8-bit values sign-extended to 16-bit values.

- sign-extended value of val[7:0] in the low halfword of the return value.
- sign-extended value of val[23:16] in the high halfword of the return value.

Operation:

```
val        = Rotate(val, rotate)
res[15:0]  = SignExtended(val[7:0])
res[31:16] = SignExtended(val[23:16])
```

```
uint32_t __UADD16 ( uint32_t val1,
                    uint32_t val2
                  )
```

GE setting dual 16-bit unsigned addition.

This function enables you to perform two 16-bit unsigned integer additions.
The GE bits in the APSR are set according to the results.

Parameters

val1 first two 16-bit summands for each addition.

val2 second two 16-bit summands for each addition.

Returns

- the addition of the low halfwords in each operand, in the low halfword of the return value.
- the addition of the high halfwords in each operand, in the high halfword of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[15:0] >= 0x10000 then APSR.GE[0] = 11 else 00
- if res[31:16] >= 0x10000 then APSR.GE[1] = 11 else 00

Operation:

```
res[15:0]  = val1[15:0]  + val2[15:0]
res[31:16] = val1[31:16] + val2[31:16]
```

```
uint32_t __UADD8 ( uint32_t  val1,
                   uint32_t  val2
                 )
```

GE setting quad 8-bit unsigned addition.

This function enables you to perform four unsigned 8-bit integer additions. The GE bits of the APSR are set according to the results.

Parameters

val1 first four 8-bit summands for each addition.

val2 second four 8-bit summands for each addition.

Returns

- the halved addition of the first bytes from each operand, in the first byte of the return value.
- the halved addition of the second bytes from each operand, in the second byte of the return value.
- the halved addition of the third bytes from each operand, in the third byte of the return value.
- the halved addition of the fourth bytes from each operand, in the fourth byte of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[7:0] >= 0x100 then APSR.GE[0] = 1 else 0
- if res[15:8] >= 0x100 then APSR.GE[1] = 1 else 0
- if res[23:16] >= 0x100 then APSR.GE[2] = 1 else 0
- if res[31:24] >= 0x100 then APSR.GE[3] = 1 else 0

Operation:

```
res[7:0]   = val1[7:0]   + val2[7:0]
res[15:8]  = val1[15:8]  + val2[15:8]
res[23:16] = val1[23:16] + val2[23:16]
res[31:24] = val1[31:24] + val2[31:24]
```

```
uint32_t __UASX ( uint32_t val1,
                  uint32_t val2
                )
```

GE setting dual 16-bit unsigned addition and subtraction with exchange.

This function enables you to exchange the two halfwords of the second operand, add the high halfwords and subtract the low halfwords.
The GE bits in the APSR are set according to the results.

Parameters

val1 first operand for the subtraction in the low halfword, and the first operand for the addition in the high halfword.

val2 second operand for the subtraction in the high halfword and the second operand for the addition in the low halfword.

Returns

- the subtraction of the high halfword in the second operand from the low halfword in the first operand, in the low halfword of the return value.
- the addition of the high halfword in the first operand and the low halfword in the second operand, in the high halfword of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[15:0] >= 0 then APSR.GE[1:0] = 11 else 00
- if res[31:16] >= 0x10000 then APSR.GE[3:2] = 11 else 00

Operation:

```
res[15:0]  = val1[15:0]  - val2[31:16]
res[31:16] = val1[31:16] + val2[15:0]
```

```
uint32_t __UHADD16 ( uint32_t val1,
                     uint32_t val2
                   )
```

Dual 16-bit unsigned addition with halved results.

This function enables you to perform two unsigned 16-bit integer additions, halving the results.

Parameters

val1 first two 16-bit summands.

val2 second two 16-bit summands.

Returns

- the halved addition of the low halfwords in each operand, in the low halfword of the return value.
- the halved addition of the high halfwords in each operand, in the high halfword of the return value.

Operation:

```
res[15:0]  = val1[15:0]  + val2[15:0]   >> 1
res[31:16] = val1[31:16] + val2[31:16]  >> 1
```

```
uint32_t __UHADD8 ( uint32_t val1,
                    uint32_t val2
                  )
```

Quad 8-bit unsigned addition with halved results.

This function enables you to perform four unsigned 8-bit integer additions, halving the results.

Parameters

val1 first four 8-bit summands.

val2 second four 8-bit summands.

Returns

- the halved addition of the first bytes in each operand, in the first byte of the return value.
- the halved addition of the second bytes in each operand, in the second byte of the return value.
- the halved addition of the third bytes in each operand, in the third byte of the return value.
- the halved addition of the fourth bytes in each operand, in the fourth byte of the return value.

Operation:

```
res[7:0]   = val1[7:0]   + val2[7:0]   >> 1
res[15:8]  = val1[15:8]  + val2[15:8]  >> 1
res[23:16] = val1[23:16] + val2[23:16] >> 1
res[31:24] = val1[31:24] + val2[31:24] >> 1
```

```
uint32_t __UHASX ( uint32_t val1,
                   uint32_t val2
                 )
```

Dual 16-bit unsigned addition and subtraction with halved results and exchange.

This function enables you to exchange the halfwords of the second operand, add the high halfwords and subtract the low halfwords, halving the results.

Parameters

val1 first operand for the subtraction in the low halfword, and the first operand for the addition in the high halfword.

val2 second operand for the subtraction in the high halfword, and the second operand for the addition in the low halfword.

Returns

- the halved subtraction of the high halfword in the second operand from the low halfword in the first operand.
- the halved addition of the high halfword in the first operand and the low halfword in the second operand.

Operation:

```
res[15:0]  = (val1[15:0]  - val2[31:16]) >> 1
res[31:16] = (val1[31:16] + val2[15:0] ) >> 1
```

```
uint32_t __UHSAX ( uint32_t val1,
                   uint32_t val2
                 )
```

Dual 16-bit unsigned subtraction and addition with halved results and exchange.

This function enables you to exchange the halfwords of the second operand, subtract the high halfwords and add the low halfwords, halving the results.

Parameters

val1 first operand for the addition in the low halfword, and the first operand for the subtraction in the high halfword.

val2 second operand for the addition in the high halfword, and the second operand for the subtraction in the low halfword.

Returns

- the halved addition of the high halfword in the second operand and the low halfword in the first operand, in the low halfword of the return value.
- the halved subtraction of the low halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

Operation:

```
res[15:0]  = (val1[15:0]  + val2[31:16]) >> 1
res[31:16] = (val1[31:16] - val2[15:0] ) >> 1
```

```
uint32_t __UHSUB16 ( uint32_t val1,
                     uint32_t val2
                   )
```

Dual 16-bit unsigned subtraction with halved results.

This function enables you to perform two unsigned 16-bit integer subtractions, halving the results.

Parameters

val1 first two 16-bit operands.

val2 second two 16-bit operands.

Returns

- the halved subtraction of the low halfword in the second operand from the low halfword in the first operand, in the low halfword of the return value.
- the halved subtraction of the high halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

Operation:

```
res[15:0]  = val1[15:0]  - val2[15:0]   >> 1
res[31:16] = val1[31:16] - val2[31:16]  >> 1
```

uint32_t __UHSUB8 (uint32_t val1,
uint32_t val2
)

Quad 8-bit unsigned subtraction with halved results.

This function enables you to perform four unsigned 8-bit integer subtractions, halving the results.

Parameters

val1 first four 8-bit operands.

val2 second four 8-bit operands.

Returns

- the halved subtraction of the first byte in the second operand from the first byte in the first operand, in the first bytes of the return value.
- the halved subtraction of the second byte in the second operand from the second byte in the first operand, in the second byte of the return value.
- the halved subtraction of the third byte in the second operand from the third byte in the first operand, in the third byte of the return value.
- the halved subtraction of the fourth byte in the second operand from the fourth byte in the first operand, in the fourth byte of the return value.

Operation:

```
res[7:0]   = val1[7:0]   - val2[7:0]    >> 1
res[15:8]  = val1[15:8]  - val2[15:8]   >> 1
res[23:16] = val1[23:16] - val2[23:16]  >> 1
res[31:24] = val1[31:24] - val2[31:24]  >> 1
```

uint32_t __UQADD16 (uint32_t val1,
uint32_t val2
)

Dual 16-bit unsigned saturating addition.

This function enables you to perform two unsigned 16-bit integer additions, saturating the results to the 16-bit unsigned integer range $0 < x < 2^{16} - 1$.

Parameters

val1 first two 16-bit summands.

val2 second two 16-bit summands.

Returns

- the addition of the low halfword in the first operand and the low halfword in the second operand, in the low halfword of the return value.
- the addition of the high halfword in the first operand and the high halfword in the second operand, in the high halfword of the return value.

The results are saturated to the 16-bit unsigned integer range $0 < x < 2^{16} - 1$.

Operation:

```
res[15:0]  = val1[15:0]  + val2[15:0]
res[31:16] = val1[31:16] + val2[31:16]
```

```
uint32_t __UQADD8 ( uint32_t val1,
                    uint32_t val2
                  )
```

Quad 8-bit unsigned saturating addition.

This function enables you to perform four unsigned 8-bit integer additions, saturating the results to the 8-bit unsigned integer range $0 < x < 2^8 - 1$.

Parameters

val1 first four 8-bit summands.

val2 second four 8-bit summands.

Returns

- the halved addition of the first bytes in each operand, in the first byte of the return value.
- the halved addition of the second bytes in each operand, in the second byte of the return value.
- the halved addition of the third bytes in each operand, in the third byte of the return value.
- the halved addition of the fourth bytes in each operand, in the fourth byte of the return value.

The results are saturated to the 8-bit unsigned integer range $0 < x < 2^8 - 1$.

Operation:

```
res[7:0]   = val1[7:0]   + val2[7:0]
res[15:8]  = val1[15:8]  + val2[15:8]
res[23:16] = val1[23:16] + val2[23:16]
res[31:24] = val1[31:24] + val2[31:24]
```

```
uint32_t __UQASX ( uint32_t val1,
                   uint32_t val2
                 )
```

Dual 16-bit unsigned saturating addition and subtraction with exchange.

This function enables you to exchange the halfwords of the second operand and perform one unsigned 16-bit integer addition and one unsigned 16-bit subtraction, saturating the results to the 16-bit unsigned integer range $0 <= x <= 2^{16} - 1$.

Parameters

val1 first two 16-bit operands.

val2 second two 16-bit operands.

Returns

- the subtraction of the high halfword in the second operand from the low halfword in the first operand, in the low halfword of the return value.
- the subtraction of the low halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

The results are saturated to the 16-bit unsigned integer range $0 <= x <= 2^{16} - 1$.

Operation:

```
res[15:0]  = val1[15:0]  - val2[31:16]
res[31:16] = val1[31:16] + val2[15:0]
```

```
uint32_t __UQSAX ( uint32_t val1,
                   uint32_t val2
                 )
```

Dual 16-bit unsigned saturating subtraction and addition with exchange.

This function enables you to exchange the halfwords of the second operand and perform one unsigned 16-bit integer subtraction and one unsigned 16-bit addition, saturating the results to the 16-bit unsigned integer range 0 <= x <= 2^{16} - 1.

Parameters

val1 first 16-bit operand for the addition in the low halfword, and the first 16-bit operand for the subtraction in the high halfword.

val2 second 16-bit halfword for the addition in the high halfword, and the second 16-bit halfword for the subtraction in the low halfword.

Returns

- the addition of the low halfword in the first operand and the high halfword in the second operand, in the low halfword of the return value.
- the subtraction of the low halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

The results are saturated to the 16-bit unsigned integer range 0 <= x <= 2^{16} - 1.

Operation:

```
res[15:0]  = val1[15:0]  + val2[31:16]
res[31:16] = val1[31:16] - val2[15:0]
```

```
uint32_t __UQSUB16 ( uint32_t val1,
                     uint32_t val2
                   )
```

Dual 16-bit unsigned saturating subtraction.

This function enables you to perform two unsigned 16-bit integer subtractions, saturating the results to the 16-bit unsigned integer range 0 < x < 2^{16} - 1.

Parameters

val1 first two 16-bit operands for each subtraction.

val2 second two 16-bit operands for each subtraction.

Returns

- the subtraction of the low halfword in the second operand from the low halfword in the first operand, in the low halfword of the return value.
- the subtraction of the high halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

The results are saturated to the 16-bit unsigned integer range 0 < x < 2^{16} - 1.

Operation:

```
res[15:0]  = val1[15:0]  - val2[15:0]
res[31:16] = val1[31:16] - val2[31:16]
```

```
uint32_t __UQSUB8 ( uint32_t val1,
                    uint32_t val2
                  )
```

Quad 8-bit unsigned saturating subtraction.

This function enables you to perform four unsigned 8-bit integer subtractions, saturating the results to the 8-bit unsigned integer range $0 < x < 2^8 - 1$.

Parameters

val1 first four 8-bit operands.

val2 second four 8-bit operands.

Returns

- the subtraction of the first byte in the second operand from the first byte in the first operand, in the first bytes of the return value.
- the subtraction of the second byte in the second operand from the second byte in the first operand, in the second byte of the return value.
- the subtraction of the third byte in the second operand from the third byte in the first operand, in the third byte of the return value.
- the subtraction of the fourth byte in the second operand from the fourth byte in the first operand, in the fourth byte of the return value.

The results are saturated to the 8-bit unsigned integer range $0 < x < 2^8 - 1$.

Operation:

```
res[7:0]   = val1[7:0]   - val2[7:0]
res[15:8]  = val1[15:8]  - val2[15:8]
res[23:16] = val1[23:16] - val2[23:16]
res[31:24] = val1[31:24] - val2[31:24]
```

```
uint32_t __USAD8 ( uint32_t val1,
                   uint32_t val2
                 )
```

Unsigned sum of quad 8-bit unsigned absolute difference.

This function enables you to perform four unsigned 8-bit subtractions, and add the absolute values of the differences together, returning the result as a single unsigned integer.

Parameters

val1 first four 8-bit operands for the subtractions.

val2 second four 8-bit operands for the subtractions.

Returns

- the subtraction of the first byte in the second operand from the first byte in the first operand.
- the subtraction of the second byte in the second operand from the second byte in the first operand.
- the subtraction of the third byte in the second operand from the third byte in the first operand.
- the subtraction of the fourth byte in the second operand from the fourth byte in the first operand.

The sum is returned as a single unsigned integer.

Operation:

```
absdiff1  = val1[7:0]   - val2[7:0]
absdiff2  = val1[15:8]  - val2[15:8]
absdiff3  = val1[23:16] - val2[23:16]
absdiff4  = val1[31:24] - val2[31:24]
res[31:0] = absdiff1 + absdiff2 + absdiff3 + absdiff4
```

```
uint32_t __USADA8 ( uint32_t val1,
                    uint32_t val2,
                    uint32_t val3
                  )
```

Unsigned sum of quad 8-bit unsigned absolute difference with 32-bit accumulate.

This function enables you to perform four unsigned 8-bit subtractions, and add the absolute values of the differences to a 32-bit accumulate operand.

Parameters

val1 first four 8-bit operands for the subtractions.

val2 second four 8-bit operands for the subtractions.

val3 accumulation value.

Returns

the sum of the absolute differences of the following bytes, added to the accumulation value:

- the subtraction of the first byte in the second operand from the first byte in the first operand.
- the subtraction of the second byte in the second operand from the second byte in the first operand.
- the subtraction of the third byte in the second operand from the third byte in the first operand.
- the subtraction of the fourth byte in the second operand from the fourth byte in the first operand.

Operation:

```
absdiff1  = val1[7:0]   - val2[7:0]
absdiff2  = val1[15:8]  - val2[15:8]
absdiff3  = val1[23:16] - val2[23:16]
absdiff4  = val1[31:24] - val2[31:24]
sum       = absdiff1 + absdiff2 + absdiff3 + absdiff4
res[31:0] = sum[31:0] + val3[31:0]
```

```
uint32_t __USAT16 ( uint32_t       val1,
                    const uint32_t val2
                  )
```

Q setting dual 16-bit unsigned saturate.

This function enables you to saturate two signed 16-bit values to a selected unsigned range.
The Q bit is set if either operation saturates.

Parameters

val1 two 16-bit values that are to be saturated.

val2 bit position for saturation, and must be an integral constant expression in the range 0 to 15.

Returns

the saturation of the two signed 16-bit values, as non-negative values.

- the saturation of the low halfword in *val1*, saturated to the bit position specified in *val2* and returned in the low halfword of the return value.
- the saturation of the high halfword in *val1*, saturated to the bit position specified in *val2* and returned in the high halfword of the return value.

Operation:

```
Saturate halfwords in val1 to the unsigned range specified by the bit position in
       val2
```

```
uint32_t __USAX ( uint32_t val1,
                  uint32_t val2
                )
```

GE setting dual 16-bit unsigned subtract and add with exchange.

This function enables you to exchange the halfwords of the second operand, subtract the high halfwords and add the low halfwords.
The GE bits in the APSR are set according to the results.

Parameters

val1 first operand for the addition in the low halfword, and the first operand for the subtraction in the high halfword.

val2 second operand for the addition in the high halfword, and the second operand for the subtraction in the low halfword.

Returns

- the addition of the low halfword in the first operand and the high halfword in the second operand, in the low halfword of the return value.
- the subtraction of the low halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[15:0] >= 0x10000 then APSR.GE[1:0] = 11 else 00
- if res[31:16] >= 0 then APSR.GE[3:2] = 11 else 00

Operation:

```
res[15:0]  = val1[15:0]  + val2[31:16]
res[31:16] = val1[31:16] - val2[15:0]
```

```
uint32_t __USUB16 ( uint32_t  val1,
                    uint32_t  val2
                  )
```

GE setting dual 16-bit unsigned subtract.

This function enables you to perform two 16-bit unsigned integer subtractions.
The GE bits in the APSR are set according to the results.

Parameters

val1 first two 16-bit operands.

val2 second two 16-bit operands.

Returns

- the subtraction of the low halfword in the second operand from the low halfword in the first operand, in the low halfword of the return value.
- the subtraction of the high halfword in the second operand from the high halfword in the first operand, in the high halfword of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[15:0] >= 0 then APSR.GE[1:0] = 11 else 00
- if res[31:16] >= 0 then APSR.GE[3:2] = 11 else 00

Operation:

```
res[15:0]  = val1[15:0]  - val2[15:0]
res[31:16] = val1[31:16] - val2[31:16]
```

```
uint32_t __USUB8 ( uint32_t val1,
                   uint32_t val2
                 )
```

GE setting quad 8-bit unsigned subtract.

This function enables you to perform four 8-bit unsigned integer subtractions. The GE bits in the APSR are set according to the results.

Parameters

val1 first four 8-bit operands.

val2 second four 8-bit operands.

Returns

- the subtraction of the first byte in the second operand from the first byte in the first operand, in the first bytes of the return value.
- the subtraction of the second byte in the second operand from the second byte in the first operand, in the second byte of the return value.
- the subtraction of the third byte in the second operand from the third byte in the first operand, in the third byte of the return value.
- the subtraction of the fourth byte in the second operand from the fourth byte in the first operand, in the fourth byte of the return value.

Each bit in APSR.GE is set or cleared for each byte in the return value, depending on the results of the operation.

If *res* is the return value, then:

- if res[8:0] >= 0 then APSR.GE[0] = 1 else 0
- if res[15:8] >= 0 then APSR.GE[1] = 1 else 0
- if res[23:16] >= 0 then APSR.GE[2] = 1 else 0
- if res[31:24] >= 0 then APSR.GE[3] = 1 else 0

Operation:

```
res[7:0]   = val1[7:0]   - val2[7:0]
res[15:8]  = val1[15:8]  - val2[15:8]
res[23:16] = val1[23:16] - val2[23:16]
res[31:24] = val1[31:24] - val2[31:24]
```

```
uint32_t __UXTAB16 ( uint32_t val1,
                     uint32_t val2
                   )
```

Extracted 16-bit to 32-bit unsigned addition.

This function enables you to extract two 8-bit values from one operand, zero-extend them to 16 bits each, and add the results to two 16-bit values from another operand.

Parameters

val1 value added to the zero-extended to 16-bit values.

val2 two 8-bit values to be extracted and zero-extended.

Returns

the 8-bit values in *val2*, zero-extended to 16-bit values and added to *val1*.

Operation:

```
res[15:0]  = ZeroExt(val2[7:0]   to 16 bits) + val1[15:0]
res[31:16] = ZeroExt(val2[31:16] to 16 bits) + val1[31:16]
```

uint32_t __UXTB16 (uint32_t val)

Dual extract 8-bits and zero-extend to 16-bits.

This function enables you to extract two 8-bit values from an operand and zero-extend them to 16 bits each.

Parameters

val two 8-bit values in val[7:0] and val[23:16] to be sign-extended.

Returns

the 8-bit values zero-extended to 16-bit values.

- zero-extended value of val[7:0] in the low halfword of the return value.
- zero-extended value of val[23:16] in the high halfword of the return value.

Operation:

```
res[15:0]  = ZeroExtended(val[7:0]  )
res[31:16] = ZeroExtended(val[23:16])
```