

Chapters 4

Arithmetic & Logic

Embedded Systems with ARM Cortex-M

Updated: Wednesday, February 7, 2018

Overview: Arithmetic and Logic Instructions

- Shift
 - **LSL** (logic shift left), **LSR** (logic shift right), **ASR** (arithmetic shift right), **ROR** (rotate right), **RRX** (rotate right with extend)
- Logic
 - **AND** (bitwise and), **ORR** (bitwise or), **EOR** (bitwise exclusive or), **ORN** (bitwise or not), **MVN** (move not)
- Bit set/clear
 - **BFC** (bit field clear), **BFI** (bit field insert), **BIC** (bit clear), **CLZ** (count leading zeroes)
- Bit/byte reordering
 - **RBIT** (reverse bit order in a word), **REV** (reverse byte order in a word), **REV16** (reverse byte order in each half-word independently), **REVSH** (reverse byte order in each half-word independently)
- Addition
 - **ADD**, **ADC** (add with carry)
- Subtraction
 - **SUB**, **RSB** (reverse subtract), **SBC** (subtract with carry)
- Multiplication
 - **MUL** (multiply), **MLA** (multiply-accumulate), **MLS** (multiply-subtract), **SMULL** (signed long multiply-accumulate), **SMLAL** (signed long multiply-accumulate), **UMULL** (unsigned long multiply-subtract), **UMLAL** (unsigned long multiply-subtract)
- Division
 - **SDIV** (signed), **UDIV** (unsigned)
- Saturation
 - **SSAT** (signed), **USAT** (unsigned)
- Sign extension
 - **SXTB** (signed), **SXTH**, **UXTB**, **UXTH**
- Bit field extract
 - **SBFX** (signed), **UBFX** (unsigned)
- Syntax
 - **<Operation>{<cond>}{S} Rd, Rn, Operand2**

ADD {Rd,} Rn, Op2	Add. $Rd \leftarrow Rn + Op2$	AND {Rd,} Rn, Op2	Bitwise logic AND. $Rd \leftarrow Rn \& \text{operand2}$
ADC {Rd,} Rn, Op2	Add with carry. $Rd \leftarrow Rn + Op2 + \text{Carry}$	ORR {Rd,} Rn, Op2	Bitwise logic OR. $Rd \leftarrow Rn \text{operand2}$
SUB {Rd,} Rn, Op2	Subtract. $Rd \leftarrow Rn - Op2$	EOR {Rd,} Rn, Op2	Bitwise logic exclusive OR. $Rd \leftarrow Rn \wedge \text{operand2}$
SBC {Rd,} Rn, Op2	Subtract with carry. $Rd \leftarrow Rn - Op2 + \text{Carry} - 1$	ORN {Rd,} Rn, Op2	Bitwise logic NOT OR. $Rd \leftarrow Rn (\text{NOT operand2})$
RSB {Rd,} Rn, Op2	Reverse subtract. $Rd \leftarrow Op2 - Rn$	BIC {Rd,} Rn, Op2	Bit clear. $Rd \leftarrow Rn \& \text{NOT operand2}$
MUL {Rd,} Rn, Rm	Multiply. $Rd \leftarrow (Rn \times Rm)[31:0]$	BFC Rd, #lsb, #width	Bit field clear. $Rd[(\text{width} + \text{lsb} - 1) : \text{lsb}] \leftarrow 0$
MLA Rd, Rn, Rm, Ra	Multiply with accumulate. $Rd \leftarrow (Ra + (Rn \times Rm))[31:0]$	BFI Rd, Rn, #lsb, #width	Bit field insert. $Rd[(\text{width} + \text{lsb} - 1) : \text{lsb}] \leftarrow Rn[(\text{width} - 1) : 0]$
MLS Rd, Rn, Rm, Ra	Multiply and subtract, $Rd \leftarrow (Ra - (Rn \times Rm))[31:0]$	MVN Rd, Op2	Move NOT, logically negate all bits. $Rd \leftarrow 0xFFFFFFFF \text{ EOR } Op2$
SDIV {Rd,} Rn, Rm	Signed divide. $Rd \leftarrow Rn / Rm$		
UDIV {Rd,} Rn, Rm	Unsigned divide. $Rd \leftarrow Rn / Rm$		
SSAT Rd, #n, Rm {,shift #s}	Signed saturate		
USAT Rd, #n, Rm {,shift #s}	Unsigned saturate		

UMULL RdLo, RdHi, Rn, Rm	Unsigned long multiply. $RdHi, RdLo \leftarrow \text{unsigned}(Rn \times Rm)$	RBIT Rd, Rn	Reverse bit order in a word. for ($i = 0; i < 32; i++$) $Rd[i] \leftarrow Rn[31 - i]$
SMULL RdLo, RdHi, Rn, Rm	Signed long multiply. $RdHi, RdLo \leftarrow \text{signed}(Rn \times Rm)$	REV Rd, Rn	Reverse byte order in a word. $Rd[31:24] \leftarrow Rn[7:0]$, $Rd[23:16] \leftarrow Rn[15:8]$, $Rd[15:8] \leftarrow Rn[23:16]$, $Rd[7:0] \leftarrow Rn[31:24]$
UMLAL RdLo, RdHi, Rn, Rm	Unsigned multiply with accumulate. $RdHi, RdLo \leftarrow \text{unsigned}(RdHi, RdLo + Rn \times Rm)$	REV16 Rd, Rn	Reverse byte order in each half-word. $Rd[15:8] \leftarrow Rn[7:0]$, $Rd[7:0] \leftarrow Rn[15:8]$, $Rd[31:24] \leftarrow Rn[23:16]$, $Rd[23:16] \leftarrow Rn[31:24]$
SMLAL RdLo, RdHi, Rn, Rm	Signed multiply with accumulate. $RdHi, RdLo \leftarrow \text{signed}(RdHi, RdLo + Rn \times Rm)$	REVSH Rd, Rn	Reverse byte order in bottom half-word and sign extend. $Rd[15:8] \leftarrow Rn[7:0]$, $Rd[7:0] \leftarrow Rn[15:8]$, $Rd[31:16] \leftarrow Rn[7] \& 0xFFFF$

Data Movement

MOV	$Rd \leftarrow \text{operand2}$
MVN	$Rd \leftarrow \text{NOT operand2}$
MRS Rd, spec_reg	Move from special register to general register
MSR spec_reg, Rm	Move from general register to special register

```
MOV r4, r5           ; Copy r5 to r4
MVN r4, r5           ; r4 = bitwise logical NOT of r5
MOV r1, r2, LSL #3   ; r1 = r2 << 3
MOV r0, PC           ; Copy PC (r15) to r0
MOV r1, SP           ; Copy SP (r14) to r1
```

Commonly Used Arithmetic Operations

ADD {Rd,} Rn, Op2	Add. $Rd \leftarrow Rn + Op2$
ADC {Rd,} Rn, Op2	Add with carry. $Rd \leftarrow Rn + Op2 + \text{Carry}$
SUB {Rd,} Rn, Op2	Subtract. $Rd \leftarrow Rn - Op2$
SBC {Rd,} Rn, Op2	Subtract with carry. $Rd \leftarrow Rn - Op2 + \text{Carry} - 1$
RSB {Rd,} Rn, Op2	Reverse subtract. $Rd \leftarrow Op2 - Rn$
MUL {Rd,} Rn, Rm	Multiply. $Rd \leftarrow (Rn \times Rm)[31:0]$
MLA Rd, Rn, Rm, Ra	Multiply with accumulate. $Rd \leftarrow (Ra + (Rn \times Rm))[31:0]$
MLS Rd, Rn, Rm, Ra	Multiply and subtract, $Rd \leftarrow (Ra - (Rn \times Rm))[31:0]$
SDIV {Rd,} Rn, Rm	Signed divide. $Rd \leftarrow Rn / Rm$
UDIV {Rd,} Rn, Rm	Unsigned divide. $Rd \leftarrow Rn / Rm$
SSAT Rd, #n, Rm {,shift #s}	Signed saturate
USAT Rd, #n, Rm {,shift #s}	Unsigned saturate

Example: Add

- Unified Assembler Language (UAL) Syntax

ADD r1, r2, r3 ; r1 = r2 + r3

ADD r1, r2, #4 ; r1 = r2 + 4

- Traditional Thumb Syntax

ADD r1, r3 ; r1 = r1 + r3

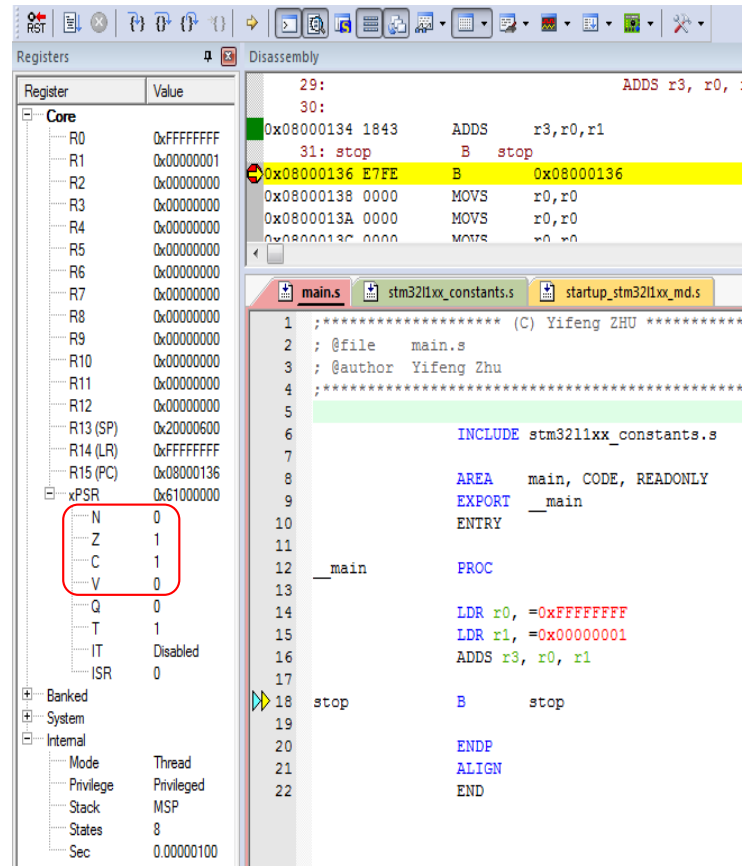
ADD r1, #15 ; r1 = r1 + 15

Example:

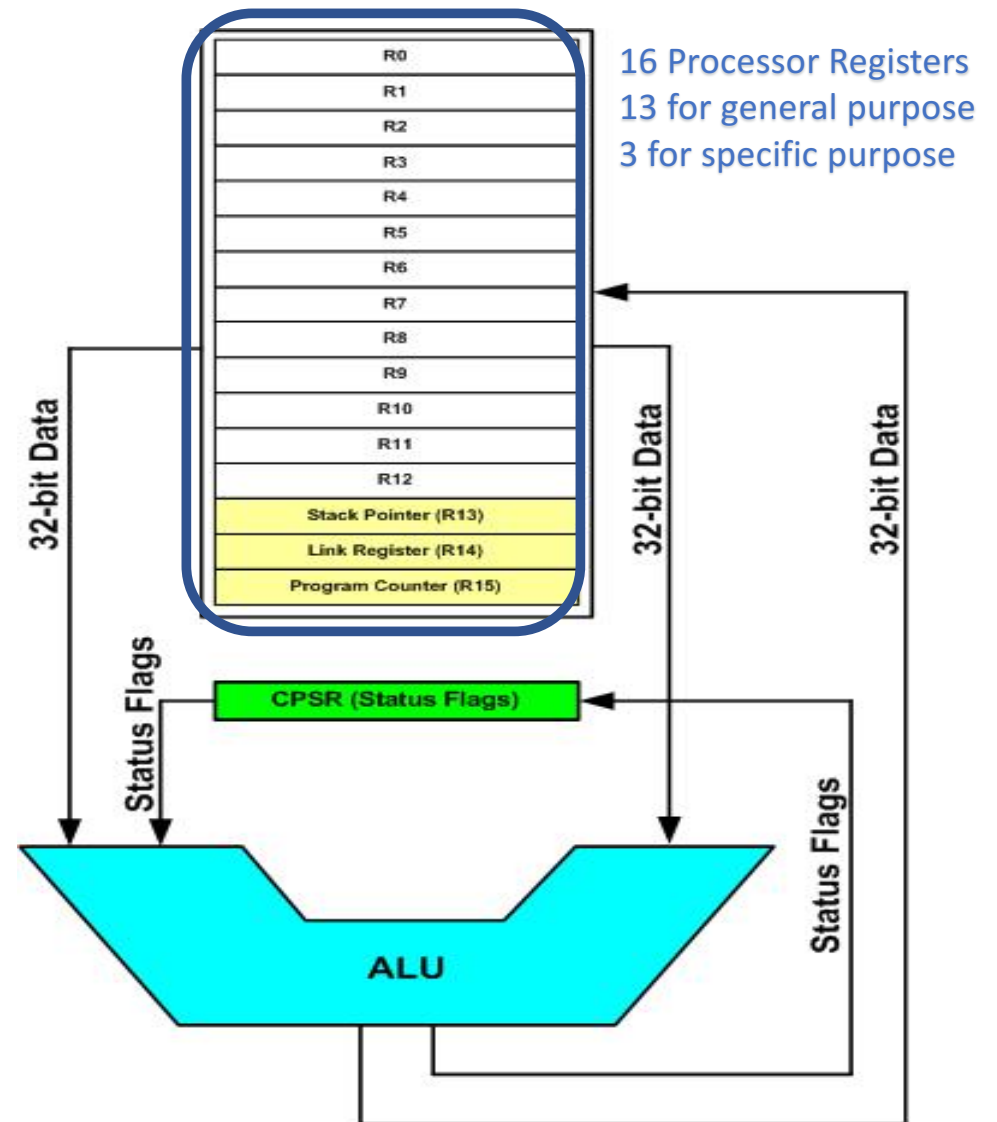
S: Set Condition Flags

```
start
    LDR    r0, =0xFFFFFFFF
    LDR    r1, =0x00000001
    ADDS   r0, r0, r1
stop    B    stop
```

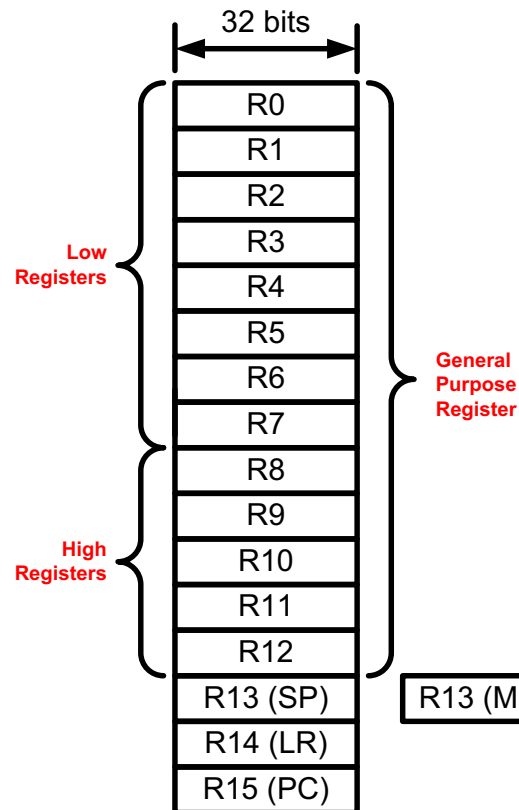
- For most instructions, we can add a suffix **S** to update the NZCV bits of the APSR register.
- In this example, the Z and C bits are set.



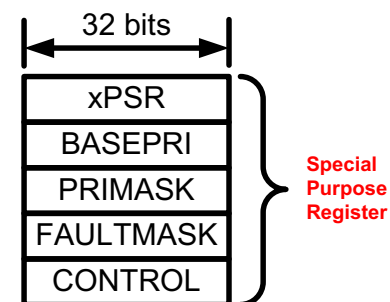
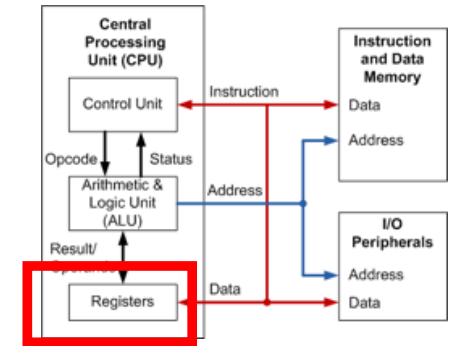
ARM Register and ALU



Processor Registers

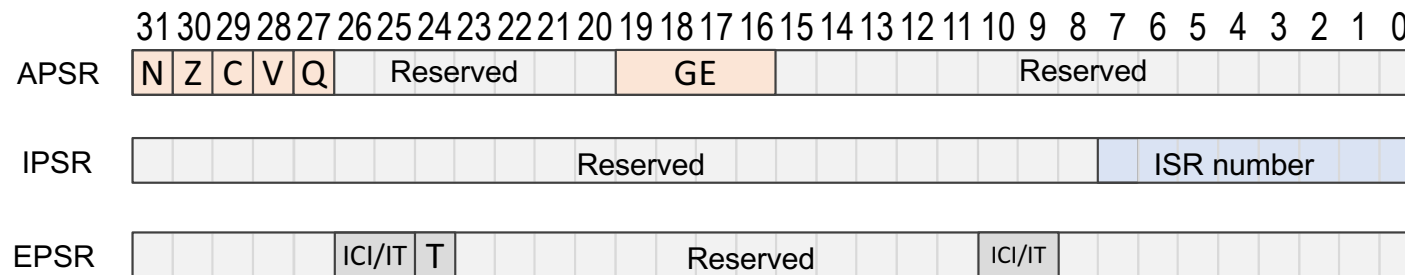


- ▶ Fastest way to read and write
- ▶ Registers are within the processor chip
- ▶ A register stores 32-bit value
- ▶ Cortex M (STM32L) has
 - ▶ **R0-R12**: 13 general-purpose registers
 - ▶ **R13**: Stack pointer (Shadow of MSP or PSP)
 - ▶ **R14**: Link register (LR)
 - ▶ **R15**: Program counter (PC)
 - ▶ Special registers (xPSR, BASEPRI, PRIMASK, etc.)- more later



Program Status Register

- Application PSR (**APSR**), Interrupt PSR (**IPSR**), Execution PSR (**EPSR**)

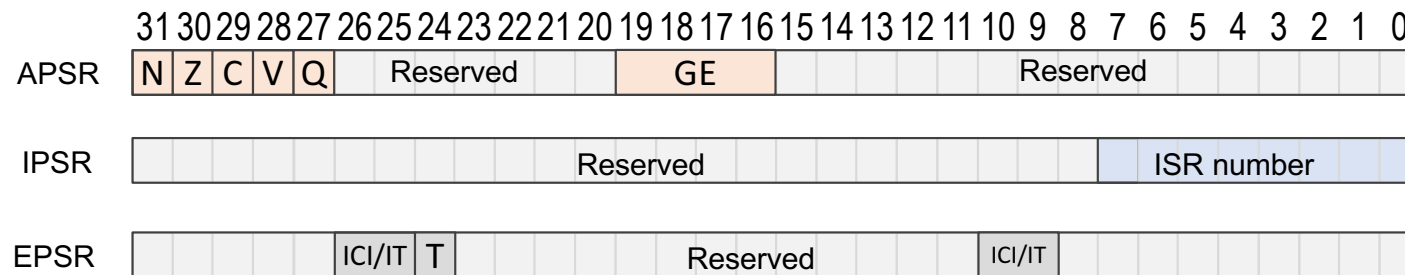


Note:

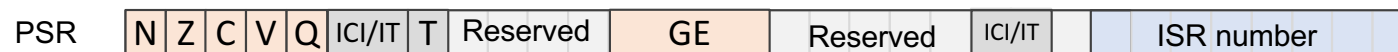
- GE flags are only available on Cortex-M4 and M7

Program Status Register

- Application PSR (**APSR**), Interrupt PSR (**IPSR**), Execution PSR (**EPSR**)



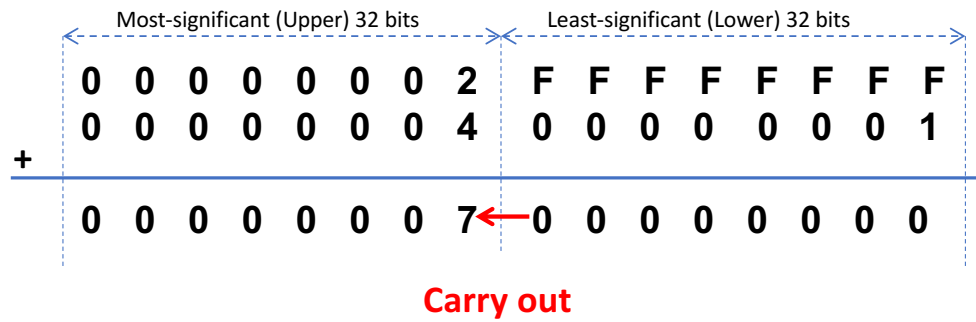
Combine them together into one register (**PSR**)



Note:

- GE flags are only available on Cortex-M4 and M7
- Use PSR in code

Example: 64-bit Addition



- A register can only store 32 bits
- A 64-bit integer needs two registers
- Split 64-bit addition into two 32-bit additions

Example: 64-bit Addition

```
start
; C = A + B
; Two 64-bit integers A (r1,r0) and B (r3, r2).
; Result C (r5, r4)
; A = 00000002FFFFFFFF
; B = 0000000400000001
LDR  r0, =0xFFFFFFFF ; A's lower 32 bits
LDR  r1, =0x00000002 ; A's upper 32 bits
LDR  r2, =0x00000001 ; B's lower 32 bits
LDR  r3, =0x00000004 ; B's upper 32 bits

; Add A and B
ADDS r4, r2, r0 ; C[31..0] = A[31..0] + B[31..0], update Carry
ADC  r5, r3, r1 ; C[64..32] = A[64..32] + B[64..32] + Carry

stop B stop
```

Example: 64-bit Subtraction

```
start
; C = A - B
; Two 64-bit integers A (r1,r0) and B (r3, r2).
; Result C (r5, r4)
; A = 00000002FFFFFFFF
; B = 0000000400000001
LDR  r0, =0xFFFFFFFF ; A's lower 32 bits
LDR  r1, =0x00000002 ; A's upper 32 bits
LDR  r2, =0x00000001 ; B's lower 32 bits
LDR  r3, =0x00000004 ; B's upper 32 bits

; Subtract B from A
SUBS r4, r0, r2 ; C[31..0]= A[31..0] - B[31..0], update Carry
SBC  r5, r1, r3 ; C[64..32]= A[64..32] - B[64..32] - Carry

stop  B  stop
```

Example: Long Multiplication

UMULL RdLo, RdHi, Rn, Rm	Unsigned long multiply. $\text{RdHi, RdLo} \leftarrow \text{unsigned}(\text{Rn} \times \text{Rm})$
SMULL RdLo, RdHi, Rn, Rm	Signed long multiply. $\text{RdHi, RdLo} \leftarrow \text{signed}(\text{Rn} \times \text{Rm})$
UMLAL RdLo, RdHi, Rn, Rm	Unsigned multiply with accumulate. $\text{RdHi, RdLo} \leftarrow \text{unsigned}(\text{RdHi, RdLo} + \text{Rn} \times \text{Rm})$
SMLAL RdLo, RdHi, Rn, Rm	Signed multiply with accumulate. $\text{RdHi, RdLo} \leftarrow \text{signed}(\text{RdHi, RdLo} + \text{Rn} \times \text{Rm})$

```
UMULL r3, r4, r0, r1    ; r4:r3 = r0 × r1, r4 = MSB bits, r3 = LSB bits
SMULL r3, r4, r0, r1    ; r4:r3 = r0 × r1
UMLAL r3, r4, r0, r1    ; r4:r3 = r4:r3 + r0 × r1
SMLAL r3, r4, r0, r1    ; r4:r3 = r4:r3 + r0 × r1
```

Example: Short Multiplication and Division

`; MUL: Signed multiply`

`MUL r6, r4, r2 ; r6 = LSB32(r4 × r2)`

`; UMUL: Unsigned multiply`

`UMUL r6, r4, r2 ; r6 = LSB32(r4 × r2)`

`; MLA: Multiply with accumulation`

`MLA r6, r4, r1, r0 ; r6 = LSB32(r4 × r1) + r0`

`; MLS: Multiply with subtract`

`MLS r6, r4, r1, r0 ; r6 = LSB32(r4 × r1) - r0`

Bitwise Logic

AND {Rd,} Rn, Op2	Bitwise logic AND. $Rd \leftarrow Rn \& \text{operand2}$
ORR {Rd,} Rn, Op2	Bitwise logic OR. $Rd \leftarrow Rn \mid \text{operand2}$
EOR {Rd,} Rn, Op2	Bitwise logic exclusive OR. $Rd \leftarrow Rn \wedge \text{operand2}$
ORN {Rd,} Rn, Op2	Bitwise logic NOT OR. $Rd \leftarrow Rn \mid (\text{NOT operand2})$
BIC {Rd,} Rn, Op2	Bit clear. $Rd \leftarrow Rn \& \text{NOT operand2}$
BFC Rd, #lsb, #width	Bit field clear. $Rd[(\text{width}+\text{lsb}-1):\text{lsb}] \leftarrow 0$
BFI Rd, Rn, #lsb, #width	Bit field insert. $Rd[(\text{width}+\text{lsb}-1):\text{lsb}] \leftarrow Rn[(\text{width}-1):0]$
MVN Rd, Op2	Move NOT, logically negate all bits. $Rd \leftarrow 0xFFFFFFFF \text{ EOR } Op2$

Example: **AND** r2, r0, r1

32 bits

r0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
r1	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
r2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bit-wise Logic **AND**

Example: ORR r2, r0, r1

32 bits

r0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
r1	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
r2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bit-wise Logic OR

Example: BIC r2, r0, r1

Bit Clear

$$r2 = r0 \& \text{NOT } r1$$

Step 1:

r1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
NOT r1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0

Step 2:

r0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
NOT r1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	
<hr/>																															
r2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	

Example: BFC and BFI

- Bit Field Clear (BFC) and Bit Field Insert (BFI).
- Syntax
 - BFC Rd, #lsb, #width
 - BFI Rd, Rn, #lsb, #width
- Examples:
 - BFC R4, #8, #12**
; Clear bit 8 to bit 19 (12 bits) of R4 to 0
 - BFI R9, R2, #8, #12**
; Replace bit 8 to bit 19 (12 bits) of R9 with bit 0 to bit 11 from R2.

Bit Operators (&, |, ~) vs
Boolean Operators (&&, ||, !)

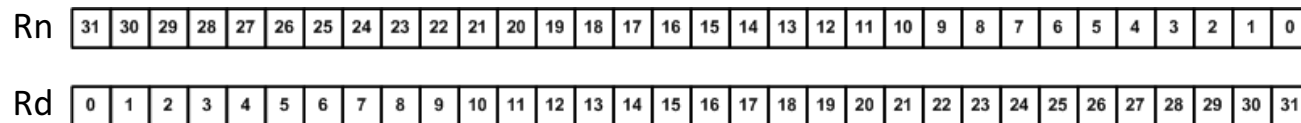
A && B	Boolean and	A & B	Bitwise and
A B	Boolean or	A B	Bitwise or
!B	Boolean not	~B	Bitwise not

- The Boolean operators perform word-wide operations, not bitwise.
- For example,
 - “0x10 & 0x01” = 0x00, but “0x10 && 0x01” = 0x01.
 - “~0x01” = 0xFFFFFFFF, but “!0x01” = 0x00.

Reverse Order

RBIT Rd, Rn	Reverse bit order in a word. for (i = 0; i < 32; i++) Rd[i] ← Rn[31– i]
REV Rd, Rn	Reverse byte order in a word. Rd[31:24] ← Rn[7:0], Rd[23:16] ← Rn[15:8], Rd[15:8] ← Rn[23:16], Rd[7:0] ← Rn[31:24]
REV16 Rd, Rn	Reverse byte order in each half-word. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:24] ← Rn[23:16], Rd[23:16] ← Rn[31:24]
REVSH Rd, Rn	Reverse byte order in bottom half-word and sign extend. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:16] ← Rn[7] & 0xFFFF

RBIT Rd, Rn



Example:

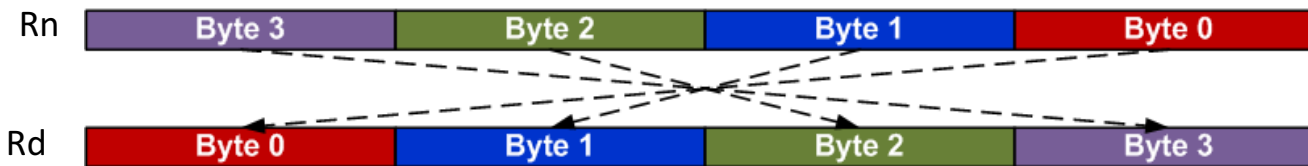
```

LDR  r0, =0x12345678 ; r0 = 0x12345678
RBIT r1, r0           ; Reverse bits, r1 = 0x1E6A2C48
    
```

Reverse Order

RBIT Rd, Rn	Reverse bit order in a word. for (i = 0; i < 32; i++) Rd[i] ← RN[31– i]
REV Rd, Rn	Reverse byte order in a word. Rd[31:24] ← Rn[7:0], Rd[23:16] ← Rn[15:8], Rd[15:8] ← Rn[23:16], Rd[7:0] ← Rn[31:24]
REV16 Rd, Rn	Reverse byte order in each half-word. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:24] ← Rn[23:16], Rd[23:16] ← Rn[31:24]
REVSH Rd, Rn	Reverse byte order in bottom half-word and sign extend. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:16] ← Rn[7] & 0xFFFF

REV Rd, Rn



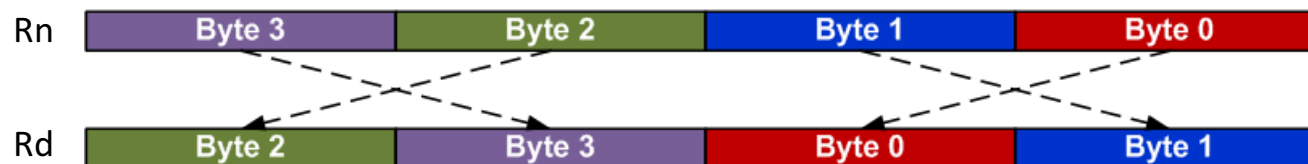
Example:

```
LDR R0, =0x12345678
REV R1, R0      ; R1 = 0x78563412
```


Reverse Order

RBIT Rd, Rn	Reverse bit order in a word. for (i = 0; i < 32; i++) Rd[i] ← RN[31– i]
REV Rd, Rn	Reverse byte order in a word. Rd[31:24] ← Rn[7:0], Rd[23:16] ← Rn[15:8], Rd[15:8] ← Rn[23:16], Rd[7:0] ← Rn[31:24]
REV16 Rd, Rn	Reverse byte order in each half-word. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:24] ← Rn[23:16], Rd[23:16] ← Rn[31:24]
REVSH Rd, Rn	Reverse byte order in bottom half-word and sign extend. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:16] ← Rn[7] & 0xFFFF

REV16 Rd, Rn



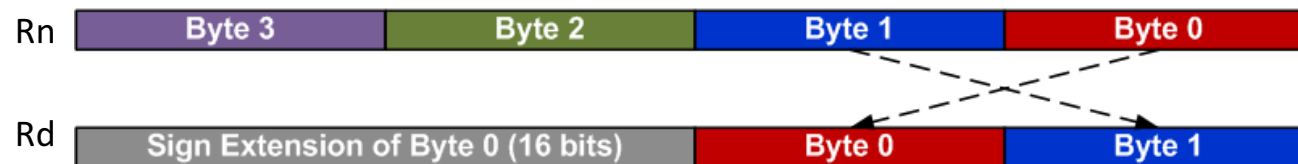
Example:

```
LDR R0, =0x12345678
REV16 R2, R0 ; R2 = 0x34127856
```

Reverse Order

RBIT Rd, Rn	Reverse bit order in a word. for (i = 0; i < 32; i++) Rd[i] ← RN[31– i]
REV Rd, Rn	Reverse byte order in a word. Rd[31:24] ← Rn[7:0], Rd[23:16] ← Rn[15:8], Rd[15:8] ← Rn[23:16], Rd[7:0] ← Rn[31:24]
REV16 Rd, Rn	Reverse byte order in each half-word. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:24] ← Rn[23:16], Rd[23:16] ← Rn[31:24]
REVSH Rd, Rn	Reverse byte order in bottom half-word and sign extend. Rd[15:8] ← Rn[7:0], Rd[7:0] ← Rn[15:8], Rd[31:16] ← Rn[7] & 0xFFFF

REVSH Rd, Rn



Example:

```
LDR R0, =0x33448899
REVSH R1, R0 ; R0 = 0xFFFF9988
```

Sign and Zero Extension

SXTB {Rd,} Rm {,ROR #n}	Sign extend a byte. $Rd[31:0] \leftarrow \text{Sign Extend}((Rm \text{ ROR } (8 \times n))[7:0])$
SXTH {Rd,} Rm {,ROR #n}	Sign extend a half-word. $Rd[31:0] \leftarrow \text{Sign Extend}((Rm \text{ ROR } (8 \times n))[15:0])$
UXTB {Rd,} Rm {,ROR #n}	Zero extend a byte. $Rd[31:0] \leftarrow \text{Zero Extend}((Rm \text{ ROR } (8 \times n))[7:0])$
UXTH {Rd,} Rm {,ROR #n}	Zero extend a half-word. $Rd[31:0] \leftarrow \text{Zero Extend}((Rm \text{ ROR } (8 \times n))[15:0])$

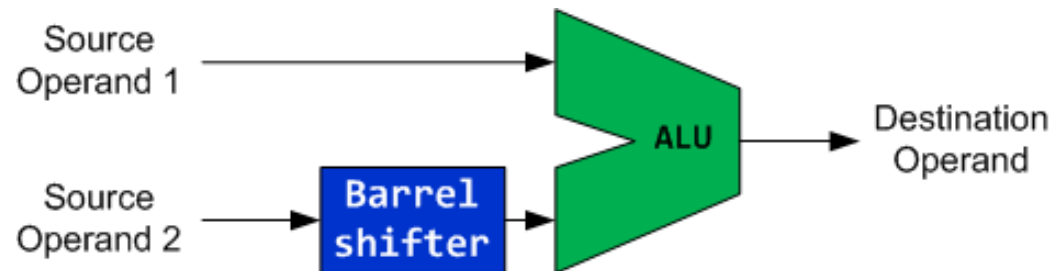
```
LDR R0, =0x55AA8765
SXTB R1, R0    ; R1 = 0x00000065
SXTH R1, R0    ; R1 = 0xFFFF8765
UXTB R1, R0    ; R1 = 0x00000065
UXTH R1, R0    ; R1 = 0x00008765
```

Sign and Zero Extension

```
signed int_8  a = -1;    // a signed 8-bit integer,  a = 0xFF
signed int_16 b = -2;    // a signed 16-bit integer, b = 0xFFFE
signed int_32 c;         // a signed 32-bit integer

c = a;                  // sign extension required, c = 0xFFFFFFFF
c = b;                  // sign extension required, c = 0xFFFFFFFFFE
```

Barrel Shifter



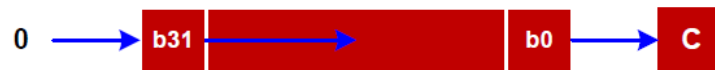
- The second operand of ALU has a special hardware called **Barrel shifter**
- Example:
`ADD r1, r0, r0, LSL #3 ; r1 = r0 + r0 << 3 = 9 × r0`

The Barrel Shifter

Logical Shift Left (**LSL**)



Logical Shift Right (**LSR**)



Rotate Right Extended (**RRX**)



Arithmetic Shift Right (**ASR**)



Rotate Right (**ROR**)



Why is there rotate right but no rotate left?

Rotate left can be replaced by a rotate right with a different rotate offset.

Barrel Shifter

- Examples:

- **ADD** r1, r0, r0, **LSL** #3

- ; r1 = r0 + r0 << 3 = r0 + 8 × r0

- **ADD** r1, r0, r0, **LSR** #3

- ; r1 = r0 + r0 >> 3 = r0 + r0/8 (unsigned)

- **ADD** r1, r0, r0, **ASR** #3

- ; r1 = r0 + r0 >> 3 = r0 + r0/8 (signed)

- Use Barrel shifter to speed up the application

- ADD** r1, r0, r0, **LSL** #3 <=> **MOV** r2, #9 ; r2 = 9
 - MUL** r1, r0, r2 ; r1 = r0
 - * 9