# Übersicht der wichtigsten Assemblerbefehle für ARM®

Autoren: Markus Bommer, Dr.-Ing. Wolfgang Heenes
Version: 0.3
Datum: 21. April 2021

## Operanden, Statusflags und Registersatz

Die vier Operanden sind:

- Direkter Wert (I)
- Register (R)
- Speicher (M)
- Adresse (PTR)

Bei den Befehlen ist angegeben, welche Operanden zulässig sind. Die Befehle sind von der Struktur Befehl Ziel, Quelle, Quelle. Zu jedem Befehl sind Beispiele angegeben.
Außerdem werden für jeden Befehl die Statusflags angegeben. Bei den Flags wird nur noch der erste Buchstabe angegeben. Falls ein Befehl ein Statusbit fest setzt, ist dies via Zuweisung des Flags dargestellt.

- C Übertragsflag (Carryflag)
- Z Nullflag (Zeroflag)
- N Vorzeichenflag (Negativflag)
- V Überlaufflag (Overflowflag)

| | |
|---|---|
| r0 | Argument / return value / temporäre Variablen |
| r1 - r3 | Argument / temporäre Variablen |
| r4 - r11 | gesicherte Variablen |
| r12 | temporäre Variablen |
| r13 (sp) | Stack Pointer |
| r14 (lr) | Link Register |
| r15 (pc) | Program Counter |

## Condition Codes (CC)

Condition Codes sind Suffixes von Befehlen, die Bedingungen anhand der Statusflags definieren. Diese Bedingungen müssen erfüllt sein, damit die Anweisung ausgeführt wird.

| Suffix | Flags | Bedeutung |
|---|---|---|
| EQ | Z gesetzt | Gleich |
| NE | Z ungesetzt | Nicht gleich |
| CS oder HS | C gesetzt | >= für unsigned |
| CC oder LO | C ungesetzt | < für unsigned |
| MI | N gesetzt | Negative |
| PL | N ungesetzt | Positiv oder Null |
| VS | V gesetzt | Overflow |
| VC | V ungesetzt | Kein Overflow |
| HI | C gesetzt und Z ungesetzt | > für unsigned |
| LS | C ungesetzt oder Z gesetzt | <= für unsigned |
| GE | N und V sind gleich | >= für signed |
| LT | N und V sind ungleich | < für signed |
| GT | Z ungesetzt, N und V sind gleich | > für signed |
| LE | Z gesetzt, N und V sind ungleich | <= für signed |

## Transportbefehle

| Befehl | Operanden | Statusbits | Beispiel und dessen Wirkung | |
|---|---|---|---|---|
| `LDR(CC)` | R, M | Keine | `LDR r1,adr_v1` | r1 := Adresse von v1 |
| | | | `LDR r3,[r1, #4]` | r3 := r1[1] |
| | | | `LDR R0, [R1, R2, LSL #4]` | r0 := r1[(r2 « 4)] |
| | | | `LDR R0, [R1, #4]!` | r1 += 4; r0 := *r1 |
| | | | `LDR R0, [R1], R2` | r0 = *r1; r1 += r2 |
| `STR(CC)` | R, M | Keine | `STR r1,adr_v1` | Adresse von v1 := r1 |
| | | | `STR r3,[r1, #4]` | r1[1] := r3 |
| | | | `STR R0, [R1, R2, LSL #4]` | r1[(r2 « 4)] := r0 |
| `MOV(CC)(S)` | R, I | N,Z | `MOV r1, #26` | r1 := 26 |
| `POP(CC)` | {R} oder {R,R,...} | Keine | `POP r1` | Legt den Inhalt vom Stack in das Register r1 |
| `PUSH(CC)` | {R} oder {R,R,...} | Keine | `PUSH r1` | Legt den Inhalt des Registers r1 auf den Stack |

Bedeutung der Symbole: *r1 : Pointer auf r1

## Arithmetische Befehle

Hervorgehoben: Nutzung von Condition Codes (CC) und Suffix S zum Setzen von Statusflags beim ADD-Befehl.

| Befehl | Operanden | Statusbits | Beispiel und dessen Wirkung | |
|---|---|---|---|---|
| `ADD(CC)(S)` | R, R, R/I | N, C, Z, V | `ADD r1, r2, #7` | r1 := r2 + 7 (setzt keine Flags) |
| | | | **`ADDEQ r1,r2, r3`** | r1 := r2 + r3 (Ausführung nur bei Flag Z=1) (setzt keine Flags) |
| | | | **`ADDS r1,r2, r3`** | r1 := r2 + r3 (setzt Flags) |
| | | | **`ADDEQS r1,r2, r3`** | r1 := r2 + r3 (Ausführung nur bei Flag Z=1) (setzt Flags) |
| `ADC(CC)(S)` | R, R, R/I | N, C, Z, V | `ADC r1, r2, r3` | r1 := r2 + r3 + C |
| `MUL(CC)(S)` | R, R, R | N,Z | `MUL r1, r2,r3` | r1 := r2 * r3 speichert nur least significant 32 bit in r1 |
| `SUB(CC)(S)` | R, R, R/I | N, C, Z, V | `SUB r1, r2,r3` | r1=r2 - r3 |

## Logische Befehle

Bedeutung der Symbole:

- ! : not
- & : and
- | : or
- ⊕ : xor

| Befehl | Operanden | Statusbits | Beispiel und dessen Wirkung | |
|---|---|---|---|---|
| `AND(CC)(S)` | R, R, R/I | N,Z | `AND r1, r2, #0xF9` | r1 = r2 & 0xF9 |
| `ASR(CC)(S)` | R, R, R/I(1-32) | C, Z, N | `ASR r1, r1, #8` | Arithmetischer Shift von r1 nach rechts um 8 Stellen |
| `EOR(CC)(S)` | R, R, R/I | N, Z | `EOR r1, r2, r3` | r1 := r2 ⊕ r3 |
| `LSL(CC)(S)` | R, R, R/I | N, C, Z | `LSL r1, r2 , #4` | r1 := Logischer Shift von r2 nach links um 4 Stellen |
| `LSR(CC)(S)` | R, R, R/I | N, C, Z | `LSR r1, r2 , #4` | r1 := Logischer Shift von r2 nach rechts um 4 Stellen |
| `MVN(CC)(S)` | R, R/I | N, Z | `MVN r1, r2` | r1 := !r2 |
| `ORR(CC)(S)` | R, R, R/I | N,Z | `ORR r1, r2, r3` | r1 := r2 \| r3 |

## Kontroll- und Sprungbefehle

| Befehl | Operanden | Statusbits | Beispiel und dessen Wirkung | |
|---|---|---|---|---|
| `B(CC)` | PTR | Keine | `B loopA` | Springt zu label loopA |
| `BL(CC)` | PTR | Keine | `BL loopA` | Speichert nachfolgende Befehlsadresse in LR und springt zu loopA |
| `BLX(CC)` | PTR/R | Keine | `BLX loopA` | Wie BL und kann Instruction Set zwischen A32 und T32 wechseln |
| `BX(CC)` | PTR/R | Keine | `BX lr` | Springt an Adresse lr und kann Instruction Set zwischen A32 und T32 wechseln |
| `CMP(CC)` | R, R/I | N, C, Z, V | `CMP r1 , #3` | Zieht 3 von Wert in r1 ab, updatet Statusflags, Wert r1 unverändert |
| `CMN(CC)` | R, R/I | N, C, Z, V | `CMN r1 , #3` | Addiert 3 zu dem Wert in r1, updatet Statusflags, Wert r1 unverändert |
| `NOP` | | Keine | `NOP` | Nulloperation/Verzögerung |

## Referenz

Die Befehle sind dem Reference Guide[1] entnommen.

[1] `http://www.heenes.de/ro/material/arm/arm_instruction_set_reference_guide.pdf`