Wdh. Bedingte Befehlsausführung

MOV, MVN (Befehl{Bedingung}{S} R_d, Operand2)
Verschieben von max. 8 Bit in Konstanten (Register können komplett verschoben werden)

Wdh. Vergleichs- und Testbefehle

CMP, CMN, TST, TEQ (Befehl{Bedingung} R_d, Operand2)

Wdh. Multiplikationsbefehle

a) 32-Bit Ergebnis

Befehl{Bedingung}{S}	MUL	R_d, R_m, R_s	;	$R_d = R_m * R_s$
	MLA	R_d, R_m, R_s, R_n	;	$R_d = R_n + (R_m * R_s)$

Bsp.: MLAEQS R0, R1, R2, R5 ; R0 = R5 + (R1 * R2), wenn Z = 1 + CCF

b) 64-Bit Ergebnis

Befehl{Bedingung}{S} R_{dlo}, R_{dhi}, R_m, R_s ; R_{dlo}, $R_{dhi} = R_m * R_s$
Befehle: UMULL, SMULL, UMLAL, SMLAL

Bsp.: SMLALNE R2, R0, R4, R6 ; R2, R0 = R2, R0 + R4 * R6

Fortsetzung Summenberechnung

Umsetzung while-Schleifen Index: $k < (N + 1) \rightarrow k \leq N$ (für ganze Zahlen)
Wahl der Register: R0 → summe, R1 → N, R2 → k
Programmcode:

```
        MOV   R0, #0
        MOV   R1, #12
        MOV   R2, #1
loop    CMP   R2, R1
        ADDLS R0, R0, R2
        ADD   R2, R2, #1
        BLS   loop
```

Kompaktere Darstellungen für kompakteren Code:

- Schleifen-Bedingungen am Ende abfragen (do, while-Schleife in C) spart einen Sprungbefehl
- Variable k kann eingespart werden, wenn N runtergezählt wird
- CMP kann entfallen → durch CCF kann geprüft werden, ob N bereits Null ist (Subtraktion von Eins und durch SUBS werden CCF gesetzt)

C-Programm	Assembler

summe = 0;
do {
summe = summe + N;
N = N – 1;
} while (N > 0);
	MOV R0, #0
	MOV R1, #12
	loop ADD R0, R0, R1
	SUBS R1, R1, #1
	BNE loop

Load- und Storebefehle

- Kommunikation zwischen Speicher und Programm nur per Load-/Storebefehle
- ARM Prozessor unterstützt:
  - Byte Zugriffe (8 Bit)
  - Halfword-Zugriffe (16 Bit), Bedingung → Adresse mod 2 = 0
  - Word-Zugriffe (32 Bit), Bedingung → Adresse mod 4 = 0
- Speicherordnung:
  - Little Endian (LSB auf niederwertigster Speicherstelle)
  - Big Endian (MSB auf niederwertigster Speicherstelle)

Bsp.:
Repräsentation von 16-Bit- und 32-Bit-Zahlen im Speicher am Beispiel einer 32-Bit Zahl (0x12345678)

Hex	Binär
0x12	0001 0010
0x34	0011 0100
0x56	0101 0110
0x78	0111 1000

Big Endian:

0	1	2	3
0001 0010	0011 0100	0101 0110	0111 1000
1 2	3 4	5 6	7 8

Little Endian:

0	1	2	3
0111 1000	0101 0110	0011 0100	0001 0010
7 8	5 6	3 4	1 2

Load-/Store-Konzept

Befehlsform: Befehl{Bedingung}{Typ} R_d, [R_m] ; [] meint, dass es eine Adresse ist

Befehle: LDR, STR

Typ: B (Byte), H (Halfword), - (Word), nur bei Load: SB (signed Byte), SH (signed Halfword)

Beispiele:

(1) LDR R1, [R2] ; 32 Bit ab Adresse in R2 in R1 laden

(2a/b) LDRSH R0, [R1] ; 16 Bit mit Vorzeichen aus [R1] in R0 laden

Adresse:	Wert (2a)	Wert (2b)
0x2000	0x80	0x8D
0x2001	0x1F	0x8F
0x2002	0x27	0x00
0x2003	0xAD	0x00

Ergebnisse:

(2a) R1 = 0x00001F8D

(2b) R1 = 0xFFFF8F8D

(3) R1 = 0x7ABCE612, R2 = 0x2000

STRH R1, [R2]

	00	01	02	03	04	05	06
0x2000	12	E6	X	X	X	X	X

<u>Adressierungsarten für Load-/Storebefehle</u>

Erweiterung der allgemeinen Befehlsform:

Befehl{Bedingung}{Typ} R_n, [R_m {, index}]

Beispiel:

(1) LDRCCH R1, [R0, #5] ;

Vor dem Laden wird zur Adresse noch 5 addiert, z.B. bei R0 = 0x2000 würde hier 0x2005 geladen. Die Verschiebung kann positiv oder negativ sein und hat max. 12 Bit.

(2) R2 = 0x2000

LDR R1, [R2, #-4]

0x2000 – 0x4 = 0x1FFC mod 4 = 0

(3) LDRB R1, [R2, #0x200]

0x2000 + 0x200 = 0x2200

(4) R2 = 0x2000, R3 =0x403

LDRB R1, [R2, R3] ; Zugriffsadresse 0x2403

(5) LDR R1, [R2, -R3, LSR#4] ; Zugriffsadresse R2 – (R3 / 2^4)

0x2000 – 0x40 = 0x1FC0 mod 4 = 0

<u>Pre- Indexed Adressing</u>

Allgemeine Befehlsform: Befehl{Bedingung}{Typ} R_n, [R_m {, index}] {!}

- Adresse wird evtl. vor dem Zugriff schon verändert
- Zugriffsadresse setzt sich zusammen aus:
  - Basis-Anteil R_m (wird nur gelesen, aber nicht verändert)
  - {,index} → Register oder Konstante und evtl. Barrel-Shifter-Anweisung
  - R_n → Register, in welches geladen oder gespeichert wird
  - {!} → Veränderung des Basisanteils (Bsp. [R_m, #2] !)