Wdh. Bedingte Befehlsausführung

R1: 0x71, R2: 0x73

```
CMP     R2, R4          ;   R2 – R4 (→ N = 1, C = 0, Z = 0, V = 0)
ADDNE   R0, R2, R4      ;   R0 = R2 – R4 → wird ausgeführt, weil Z = 0
SUBCS   R1, R4, R2      ;   wird nicht ausgeführt, weil C = 0
```

Wdh. Barrel Shifter

```
RSBNES  R0, R1, R2, LSR R3  ;   R0 = R2/2^R3 – R1, wenn Z = 0 + CCF
ADC     R2, R2, R1          ;   R2 = R2 + R1 + C
ORR     R1, R5, R3          ;   R1 = R5 OR R3
```

Daten-Verschiebebefehle

- Allg. Befehlsform: Befehl{Bedingung}{S} R_d, Operand2 (R_d → Zielregister, Operand2 → Register oder Konstante)
- Befehle: MOV, MVN (move, move not)
- Bedingungen: z.B. EQ, NE, MI

Beispiele:

R0 = 0x1, R1 = 0x2

```
MOV     R1, R0          ;   Inhalt von R0 nach R1 schieben
MOV     R1, R0, LSL#2   ;   R1 = 2^2 * R0 = 0x4
MOV     R0, #0x33       ;   R0 = 0x33
```

➔ Max. 8 Bit in Operand2, sonst kann der Befehl nicht ausgeführt werden

```
MVN     R0, #0x1        ;   R0 = 0xFFFFFFFE XOR 0x1
```

Vergleichs- und Testbefehle

- Allg. Befehlsform: Befehl{Bedingung}{S} R_d, Operand2
- Befehle: CMP, CMN, TST, TEQ

Beispiele:

R0 = 0x1, R1 = 0x2

```
CMP     R0, R1          ;   R0 – R1 (CCF: N = 1, Z = 0, V = 0, C = 0)
CMP     R1, #0x2        ;   R1 – 0x2 (CCF: N = 0, Z = 1, V = 0, C = 0)
CMN     R0, R1          ;   R0 + R1 (CCF: N = 0, Z = 0, V = 0, C = 0)
TST     R0, R1          ;   R0 AND R1 (CCF: N = 0, Z = 1, V = 0, C = 0)
CMP     R0, R1, LSL #2  ;   1 – 8 = -7 (CCF: N = 1, Z = 0, V = 0, C = 0)
```

Multiplikationsbefehle

Der ARM7 besitzt eine MAC (multiply accumulate unit), es ist somit möglich, ein Ergebnis einer Multiplikation anschließend zu addieren.

a) Multiplikation mit 32-Bit-Ergebnis

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

b) Multipliaktion mit 64-Bit-Ergebnis
Nutzen von 2 Registern für 64-Bit-Zahl → R_{dlo} (untere 32 Bit) und R_{dhi} (obere 32 Bit)

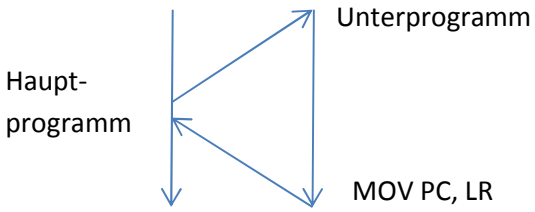
UMULL	R0, R1, R2, R3	;	R0, R1 = R2 * R3

Sprungbefehle

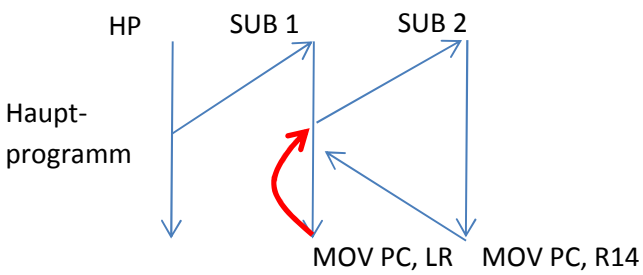
- Allgemeine Befehlsform: Befehl{Bedingung} Label (Label = Sprungmarke)
- Befehle: B (branch), BL (branch link)
- Branch für Schleiefn innerhalb eines Programms
- BL für den Aufruf von Unterprogrammen
- Beispiel:

```
CMP   R1, 0x10    ;   R1 – 0x10 (CCF setzen)
BEQ   val_ok      ;   wenn R1 = 0x10, zu val_ok springen
```

- Branch Link bewirkt:
  - PC (R15) → erhält Adresse von Befehl 1 des Unterprogramms
  - Rücksprungadresse wird im Link-Register (R14) abgelegt
- Nach dem Durchlaufen des Unterprogramms muss der Inhalt des Link-Registers in den PC geladen werden (→ aktiv ausführen, keine Automatik)
- Schematische Darstellung des Ablaufs für ein Unterprogramm



- Schematische Darstellung für mehrere Unterprogramme



LR enthält Daten von SUB1, nicht jene des HPs → Endlosschleife in SUB1

➔ Um diesen Rücksprung durchführen zu könne, muss der Inhalt des LR vorher im Stack gesichert werden und später wieder geladen werden

Beispielprogramm:

```
        MOV     R0, #0xF2
        MOV     R0, R0, LSL #24
        MOV     R1, #0x7B
        MOV     R1, R1, LSL #10
        MOV     R2, #2
        MOV     R3, #0
loop    CMP     R2, #0
        UMLA    R3, R4, R0, R1
        SUB     R2, R2, #1
        BNE     loop
```

Analyse:

```
R0 = … 1111 0010
R0 = 1111 0010 0000 0000 0000 0000 0000 0000 (F200 0000)
R1 = … 0111 1011
R1 = 0000 0000 0000 0001 1110 1100 0000 0000 (0001 EC00)
R2 = … 0000 0010
R3 = … 0000 0000
R4 = … 0000 0000
CMP     R2, #0              ;   2 – 0 = 2 (N = 0, C = 0, Z = 0, V = 0)
UMLAL   R3, R4, R0, R1      ;   R3, R4 = 0001 D118 0000 0000
SUB     R2, R2, #1          ;   R2 = R2 – 1
BNE     loop                ;   läuft, bis R2 den Wert Null enthält
```

<u>Beispiel Summenberechnung (Umwandlung C in Assembler)</u>

- Umwandeln von for-Schleife aus C-Programm in while-Schleife

```
Summe = 0;
K = 1;
While (k < (N+1))       {
        Summe = Summe + K;
        K++;
}
```

- Wahl der Variablen

Summe → R0, N → R1, K → R2

- Formulieren des Programmablaufs

```
        MOV     R0, #0
        MOV     R1, #12
        MOV     R2, #1
loop    CMP     R2, R1
        ADDGE   R0, R0, R2
        ADDGE   R2, R2, #1
        BNE     loop
```