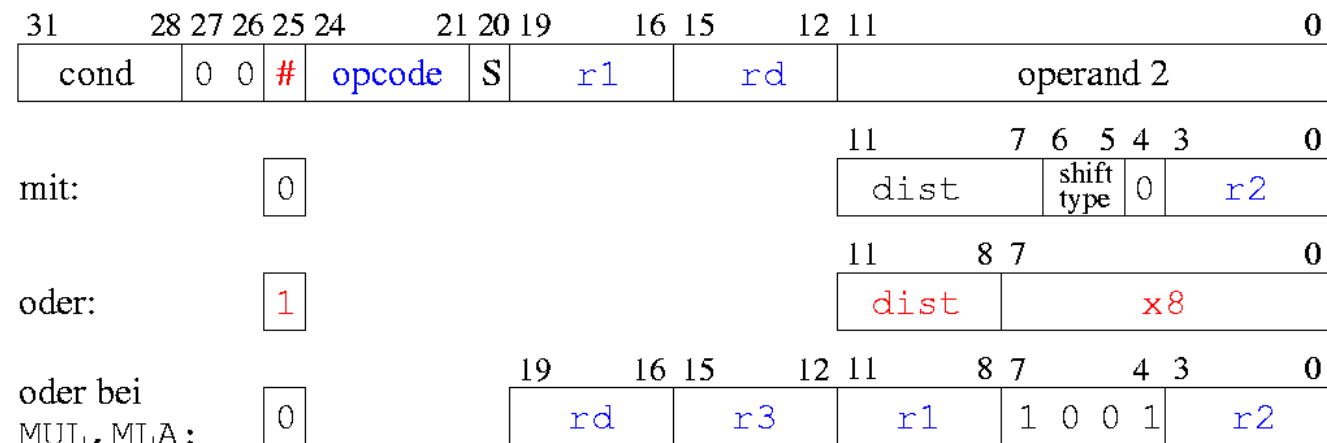


# ARM: Befehlssatz

Befehle zur Verarbeitung von Daten („data processing“):

- Register/Register-Befehle: `<op> <rd>, <r1>, <r2>`  
(Achtung! Andere Interpretation:  $\langle rd \rangle \leftarrow \langle r1 \rangle \langle op \rangle \langle r2 \rangle$ )
- Transport-Befehl: `MOV <rd>, <r2>`  
(Achtung! Andere Interpretation:  $\langle rd \rangle \leftarrow \langle r2 \rangle$ )
- statt `<r2>` ist auch eine Konstante `#x` zur unmittelbaren Adressierung möglich: `<op> <rd>, <r1>, #x` bzw. `MOV <rd>, #x`
- auch 1 Befehl mit 4 Registeroperanden: `<op> <rd>, <r1>, <r2>, <r3>`

Instruktionsformat:



# ARM: Befehlssatz (Forts.)

---

- Auswahl einiger „*data processing*“ Instruktionen:

ADD	<i>Add</i>	$rd \leftarrow r1 + r2$
ADC	<i>Add with carry</i>	$rd \leftarrow r1 + r2 + C$
SUB	<i>Subtract</i>	$rd \leftarrow r1 - r2$
SBC	<i>Subtract with carry</i>	$rd \leftarrow r1 - r2 + C - 1$
MUL	<i>Multiply</i>	$rd \leftarrow r1 \times r2$
MLA	<i>Multiply and accumulate</i>	$rd \leftarrow r1 \times r2 + r3$
AND	<i>Logical AND</i>	$rd \leftarrow r1 \text{ AND } r2$
ORR	<i>Logical OR</i>	$rd \leftarrow r1 \text{ OR } r2$
EOR	<i>Exclusive OR</i>	$rd \leftarrow r1 \oplus r2$
BIC	<i>Logical AND NOT („Bit Clear“)</i>	$rd \leftarrow r1 \text{ AND } (\text{NOT } r2)$
CMP	<i>Compare</i>	set cc on $r1 - r2$
CMN	<i>Compare negated</i>	set cc on $r1 + r2$
TST	<i>Test</i>	set cc on $r1 \text{ AND } r2$
TEQ	<i>Test on equivalence</i>	set cc on $r1 \oplus r2$
MOV	<i>Move register</i>	$rd \leftarrow r2$
MVN	<i>Move register negated</i>	$rd \leftarrow \text{NOT } r2$

# ARM: Befehlssatz (Forts.)

---

- einige Besonderheiten:

- Programmierer kann bei jeder „*data processing*“ Instruktion das **Suffix S** anfügen, wenn die Status-Bits im CPSR modifiziert werden sollen

Beispiel: `ADDS r5,r7,r9`

- die Instruktionen `<op> <rd>,<r1>,<r2>` und `MOV <rd>,<r2>` sind **mit Schiebe-Operationen kombinierbar**, die das Register `<r2>` **vor** Ausführung der Operation `<op>` bzw. `MOV` mit einem der Befehle `LSL`, `LSR`, `ASL`, `ASR` oder `ROR` **um eine Distanz  $dist \in [0,\dots,31]$**  schieben

Beispiele: `ADD r2,r3,r4,LSL #8` ;  $r2 \leftarrow r3 + r4 * 256$

`MOV r0,r1,ASR #4` ;  $r0 \leftarrow r1 / 16$

- bei unmittelbarer Adressierung muß gelten:  $x = x8 \times 2^{2 \times dist}$  (mit 8-Bit Zahl  $x8$ )
- **jede** Instruktion kann **bedingt** ausgeführt werden, indem ein **Prädikat** `cc`  $\in \{EQ, NE, CS(= HS), CC(= LO), MI, PL, VS, VC, HI, LS, GT, LT, GE, LE, AL(\text{always}), NV(\text{never})\}$  als Suffix angehängt wird

Beispiel: `CMPS r0,r1`

`ADDEQ r2,r3,r4` ; **if** ( $r0==r1$ )  $r2 \leftarrow r3 + r4$

`SUBNE r2,r3,r4` ; **else**  $r2 \leftarrow r3 - r4$

# ARM: Befehlssatz (Forts.)

## Sprungbefehle:



- hier wird ausschließlich die **relative Adressierung** mit 24-Bit Distanzen (im Zweierkomplement) eingesetzt; Instruktionsformat:



(da alle Instruktionen in **einem** 32-Bit Befehlswort zu kodieren sind, kann es keine absoluten Sprungadressen geben !)

- zwei Sprungbefehle:
  - die Instruktion **B** („*Branch*“, mit **L=0**) lädt den PC mit  $PC + 4 * offset$
  - die Instruktion **BL** („*Branch with Link*“, mit **L=1**) kopiert zusätzlich die Adresse der folgenden Instruktion in das „*link register*“ r14 ( $\Rightarrow$  Unterprogrammaufruf)
- alle Sprungbefehle können mit Bedingungen versehen werden; fehlt die Bedingung, so wird das Suffix **AL** (*always*) angenommen

- Beispiel:

CMPS r1, r9		doadd: ADD r1, r1, #1024
BLLT doadd		MOV PC, r14
...		

# ARM: Befehlssatz (Forts.)

---

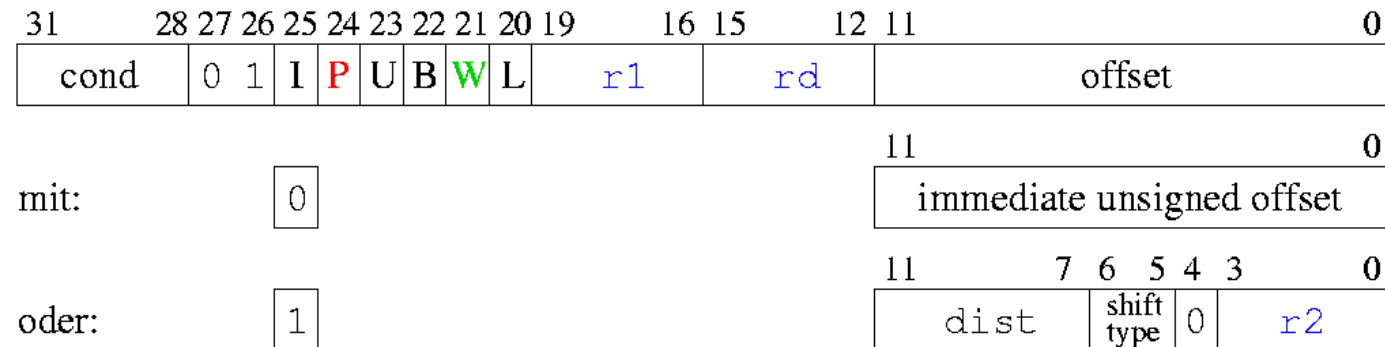
Befehle zum Laden/Speichern von Daten aus/in Speicher:

- zwei Instruktionen: `LDR rn, <ea>` und `STR rn, <ea>`
- zur Bildung der effektiven Adresse ist nur eine **indizierte Adressierung** mit beliebigem Indexregister r0 bis r15 möglich
- zusätzlich kann ein **konstanter 12-Bit Offset** oder ein (mittels Angabe eines Schiebepfeils skalierbarer) **Registeroffset** addiert/subtrahiert werden
- Offset kann entweder **vor** Berechnung der effektiven Adresse („*pre-indexed*“) oder **nach** Berechnung der effektiven Adresse („*post-indexed*“) zum Indexregister addiert/subtrahiert werden
- bei „*pre-indexed with write-back*“ (Notation mit „!*!*“) wird die berechnete effektive Adresse in das Indexregister geschrieben
- Beispiele:

<code>LDR r0, [r1, #8]</code>	;	ea=r1+8,	r1 unverändert
<code>LDR r0, [r1, #8]!</code>	;	ea=r1+8,	r1←r1+8
<code>STR r0, [r1], #16</code>	;	ea=r1,	r1←r1+16
<code>STR r5, [r2], -r3</code>	;	ea=r2,	r2←r2-r3

# ARM: Befehlssatz (Forts.)

- Instruktionsformat von LDR / STR:



Instruktionsbits:

*pre-indexing* (P=1) / *post-indexing* (P=0)

*add offset* („up“, U=1) / *subtract offset* (U=0)

*transfer byte* (B=1) / *transfer 32-bit word* (B=0)

nur bei *pre-indexing* : *write back* (W=1) / *no write back* (W=0)

*load* (L=1) / *store* (L=0)

- bei fehlender Angabe eines Indexregisters wird eine pc-relative Adresse (*pre-indexed*) mit einem 12-Bit Offset generiert (sofern möglich)
- weitere Beispiele:
 

LDR r0, [r1, r2, LSL #2]!	; ea=r1+r2*4, r1←r1+r2*4
STR r1, [r6], -r4, ASR #4	; ea=r6, r6←r6-r4/16
LDR r3, x	; ea=pc+(x-pc), pc unverändert
STREQ r5, y	; ea=pc+(y-pc), pc unverändert