Pre-Indexed Adressing

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

Befehle: STR, LDR

Bedingung: EQ, NE, CC, CS, …

Typen: -, H, B (nur LDR: SH, SB)

Beispiele:

```
LDR       R1, [R0]        ;   Wer aus Adresse [R0] in R1 laden (Adresse mod 4)
LDRNESH   R0, [R1, R2]    ;   Bei Z = 0 → Adresse R1 + R2 (vorzeichenrichtig)
```

Bedeutung des Rufzeichens: Wert von R1 + R2 wird in R1 gespeichert

Post-Indexed-Adressing

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

Befehle: STR, LDR (Bedingungen & Typen s. Pre-Indexed Adressing)

Beispiele:

R2: 0x2000, R3: 0x100

a) `LDRB  R1, [R2], #0x10` ;[R2] in R1 laden, dann R2 = R2 + 0x10 = 0x2010

b) `LDR  R1, [R2], #-4` ;0x2000 mod 4 = 0 ([R2] in R1), $R2_{neu}$ = R2 - 4 = 0x1FFC

c) `LDRH  R2, [R2], R3` ;0x2000 mod 2 = 0, $R2_{neu}$ = R2 + R3 = 0x2100

d) `LDR  R1, [R2], -R3, LSR#4` ;$R2_{neu}$ = R2 – (R3 / 16) = 0x1FF0

$R2_{neu}$ wird nach dem Befehl (bzw. dem Zugriff) in R2 geschrieben

R1: 0x7F38, R4: 0x19, R5: 0x7F35

e) `STR  R5, [R1], -R4, LSL#1` ;R5 mod 4 = 0, $R1_{neu}$ = R1 – 2 * R4 = 0x7F06

f) `STRH  R5, [R1]`

| Adresse | Wert e) | Wert f) |
|---|---|---|
| 0x7F38 | 35 | 35 |
| …39 | 7F | 7F |
| …3A | 00 | XX |
| …3B | 00 | XX |

Programmbeispiel:
R2: 0x3002

```
        MOV     R0, #3
loop    LDRSH   R1, [R2], #2
        SUBS    R0, R0, #1
        BNE     loop
```

| Adresse: | Wert: |
|---|---|
| 0x3002 | 0x3F |
| …03 | 0xF2 |
| …04 | 0x8A |
| …05 | 0x3B |
| …06 | 0x10 |
| …07 | 0x7F |
| …08 | 0x3C |

Was steht nach dem Ablauf des Programms in R1?

Analyse:
R0 = 3
R1 → 0xFFFFF23F, $R2_{neu}$ = 0x3004
R0 = 3 – 1 = 2 (Z = 0)
R1 → 0x00003B8A, $R2_{neu}$ = 0x3006
R0 = 2 – 1 = 1 (Z = 0)
R1 → 0x00007F10, $R2_{neu}$ = 0x3008
R0 = 1 – 1 = 0
END

<u>Direkte Adressierung</u>

```
        LDR     R1, #0x32090004     ;FUNKTIONIERT NICHT!

        MOV     R5, #0x40
```

Befehl übersetzt in HEX-Code (Gesamtbefehl 32 Bit):

$$0x\underbrace{E3A}_{\text{Befehl}}\ \underbrace{0}_{\text{Register}}\underbrace{5}\ \underbrace{040}_{\text{Wert}}$$

<u>Laden von Speicheradressen in Registern</u>

a) Über MOV Befehl

```
        MOV     R1, 0x32000000      (relevante Zahlen entsprechen 8 bit)
        LDR     R0, [R1]
```

b) Load-Befehl
In einer Speicherstelle steht die Adresse, die in das Register geschrieben werden soll. Diese Speicherstelle wird über den Load-Befehl geladen. Dabei wird der PC (R15) als Register zur Adressbildung verwendet. Soll auf Daten im Speicher zugegriffen werden, muss die Adresse dieser Daten in einem Register geladen werden. Vorteil der Verwendung von Labeln → Programm wird lesbarer und man muss sich als Programmierer nicht um die Adressbildungen kümmern.

```
        LDR     R1, =mydata         (= ist Adressangabe)
        LDR     R2, [R1]

mydata  DCD     0x04030201          ; Belegen des Speichers mit Daten
```

Assembler-Beispiel „Finde Kleinsten“

Registerzuweisung:

R0 → index, R1 → kleinster, R2 → Speicheradresse des ersten Arrayelements, R3 → aktuelles Arrayelement

```
        MOV     R0, #10
        MOV     R1, #127
        LDR     R2, =myArray
loop    LDRSB   R3, [R2], #1
        CMP     R3, R1          ;       R3 – R1 → CCF
        MOVLT   R1, R3
        SUBS    R0, R0, #1
        BNE     loop
```

Stack-Befehle

Stack arbeitet nach LIFO-Prinzip (Last In – First Out) → Was zuerst abgelegt wird, wird als letztes wieder ausgegeben. Zur Verwendung des Stacks dient die Stackspitze (→ Stack Pointer).

```
        STMFD   R13!, {R0}      ; Stackpointer mit !, {} zeigen Register, die gesichert werden
                                  Sollen

        LDMFD   R13!, {R0}      ; gesicherten Wert auf R13 in R0 schreiben
```

Aufruf von Unterprogrammen (Beispiel)

```
        LDR     R0, =myArray
        MOV     R1, #10
        BL      Finde_kleinsten     → Branch Link → Folgeadresse ins Link-Register
```

**Assembler Direktiven**

Symboldefinitionen mit der EQU-Direktive

- Dient der Lesbarkeit des Programms
- Häufig auftretende konstante Werte anpassen muss nur an einer Stelle passieren
- Befehlsform: Symbol EQU Expression
- Beispiele:

```
Konstante1  EQU  12
Reg1        EQU  R3
```

Speicheradressierung

- Bereitstellen von Speicher
- Befehlsform: Label SPACE Anzahl
- Beispiele:
  Time SPACE 8 (→ für Time werden 8 Byte reserviert)
  Count SPACE 4 (→ für Count werden 3 Byte reserviert)
- ALIGN → bei ARM-Modus ALIGN = 4 (Thumb-Modus → ALIGN = 2)
- Speicherausrichtung entscheidet über ALIGN
- Reservierung des Speichers nur für die benannten Labels möglich

Speicherzuordnung mit der AREA Direktive

- Befehlsform: _AREA Segment-Name Speicherklasse{, Option}
- Legt ein Speichersegment an
- Name frei wählbar
- Klasse: ROM oder RAM
- Beispiele:
  _AREA MyProg, CODE, READONLY, ALIGN = 4
  _AREA MyData, READWRITE, ALIGN = 2

END

Am Ende eines Assemblerprogramms muss die Direktive END programmiert werden. END benötigt mindestens ein Leerzeichen.
Befehlsform: _END