Wdh. Pre-Indexed Adressing

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

Befehle: LDR, STR

Bedingungen: z.B. EQ, NE, CS, CC

Typ: -, H, B (für LDR zusätzlich: SH, SB)

Pre-Indexed: Adresse wird vor dem Laden verändert (! Bewirkt Speicherung der geänderten Adresse)

Beispiel:

```
LDRMISH   R0,[R3, -R2, LSL#4] !   ; R3 – (2^4 *R2) → R3neu (Adresse mod 2 = 0?)
```

Post-Indexed Adressing

Befehlsform: Befehl{Bedingung}{Typ} R_d, [R_m]{, index}

→Befehle, Bedingungen & Typen s. Pre-Indexed Adressing

Beispiel:

```
STRNEB    R3, [R2], R4, LSR#2     ; R2neu = R2 + (R4 / 2^2), wenn Z = 0
```

Stack-Befehle

```
STMFD     R13!, {R0, R2 – R7}
…
LDMFD     R13!, {R0, R2 – R7}
```

SPACE – Direktive

```
Mein_Stack  SPACE  1024
Top_Stack          ;      1024 Byte werden reserviert (Stackpointer zeigt auf 0x40000400)
```

0x40000000 → Beginn RAM-Speicher bei LPC2148

Adressierung

```
MOV   R1, #4200     ;NUR 8 Bit Werte (für relevante Werte)
LDR   R2, [R1], #2

LDR   R0,=daten
LDR   R2, [R0, R3]
```

Speicher-Initialisierung

```
Daten  DCD   0x12345678, 0x2357, 0x223FAB, 0     (ALIGN = 4)
```

Weitere Beispiele:

```
Message     DCB   „Press a key to continue“, 0
Tabelle     DCW   0x24AC, 0x12CD, 0x13, 0
Tab         DCW   2,3,4,5,“;“,0
```

Speicherzuordnung (AREA)

```
AREA   Mein_Prog     CODE, READONLY, ALIGN=4
AREA   MeineDaten    DATA, READWRITE, ALIGN=2   (ALIGN =2 für 16 Bit-Werte)
```

Prozedur-Deklaration mit FUNCTION

```
Meine   FUNCTION            ; Name ist frei wählbar
…
ENDFUNC
```

Importieren und Exportieren mit IMPORT/EXPORT

```
EXPORT      Symbolname
IMPORT      Symbolname
```

Symbolname ist das entsprechende Label.
Beispiel:
In Datei 1

```
FileOne.s
 EXPORT     meineDaten
Mydata      DCB     1,2,3,4,5,0
```

In Datei 2

```
FileTwo.s
 IMPORT     meineDaten
LDR         R1, =mydata
```

Analyse des Beispiels mit Assembler Direktiven (3.8)

```
R0 = 25, R3 = 14, R4 = 1
R3 = R3 – 1 = 13 + CCF
R4 = 12                             ; Wdh. Solange, bis Wert gefunden wurde
Wenn CMP R0, R4 zu Z = 1 führt, Sprung zu InTa
LDR     R2, = OutputCodeTab         ; Inhalt von Output-Tabelle in Speicher ab Adresse R2
RSB     R3, R3, #Elemzahl           ; R3 = Elemzahl – R3 = 14 – 2 = 12
LDRB    R1, [R2, R3]                ; Wert der OutputCodeTab bestimmen
                                      Pre-Index → R2 (start) + 12 (Elementzahl gesucht)
                                      ➔ Korrekter Wert aus OutputCodeTab
```

Einführung in C

Präprozessor

```
#include      <LPC214x.h>;   Einbinden von Datei-Textersetzung
#define       M 100;         Konstante M = 100 initialisieren
#define       MAKRO Ausdruck;
```

Bedingte Übersetzung

```
#ifdef  symbol
…[Code]
#endif
```

Beispielprogramm

Deklaration VOR main, damit main alles kennt (globale Variablen). Definition kann später erfolgen. Das Attribut volatile verhindert Compileroptimierung.

```
char    keyPressed;
long    count = 0;
…
while(keyPressed != ‚x‘){        //Compileroptimierung → while(1)
        Count++;
}
volatile char keyPressed;
```

Umwandeln von Typen

Explizit: x = (float)(3 + 7) ; wenn int i=3 und int j=7 → x wird dennoch zu float

Implizit: int a = 3; float i=1,23; f = f + a; → f wird zu float

Typenumwandlung – Beispiel

```
Float   pi = 3.14;
Int     i = (int)pi         ; i = 3 (Ganzzahlanteil wird extrahiert)
```

If-Statement

- Ja/Nein Abfrage mit Folge-Statements
- Verschachtelung möglich (Wenn ja, und anschließend ja, dann…)

Switch-Statement

- Definition von verschiedenen Fällen (cases), die eintreten können
- Jeder Fall hat ein Folge-Statement, welches beim Eintreten ausgeführt wird
- Break-Befehl führt zum Abbruch der Abfrage nach dem Durchlauf eines Statements

For-Schleife

- Schleife wird mit Laufvariable initialisiert
- Läuft, bis End-Bedingung erreicht ist

While-Schleife

- Kopfgesteuert: Prüfen der Bedingung zu Beginn der Schleife
- Fußgesteuert (do … while): ein Durchlauf, erst dann prüfen der Bedingung

Pointer

- Anlegen mit <Typ>*<name> → Bsp. int*iptr
- Pointer arbeiten wie Arrays
- Beispiel:

```
Int numbers[] = {1,2,3,4};
Int*iptr;
Iptr = numbers;        Pointer zeigt auf erstes Arrayelement
*iptr = 5;             Verändern des Inhalts zu 5 (* = Inhaltsoperator)
Iptr++;                Zeigen auf 2. Arrayelement (Typ von Basiselement)
*iptr = 7;             zweites Arrayelement erhält den Wert 7
Iptr = &x              & = Adressoperator → iptr erhält Adresse x
```