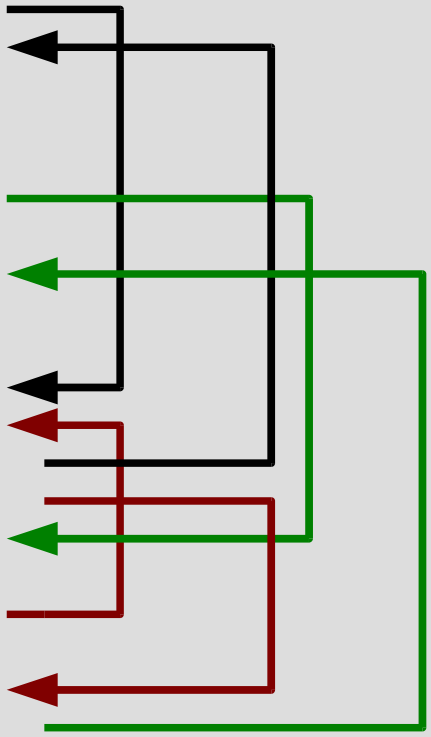
Stack, Stackpointer, Unterprogramm

```
HP:   0    *
      1    *
      2    *
      3    CAL UP1
      4    *
      5    *
      6    CAL UP2
      7    *
      ..........
UP1: 30    *
      33   RET
UP2: 40    *
      41   CAL  UP1
      42   *
      43    RET
```



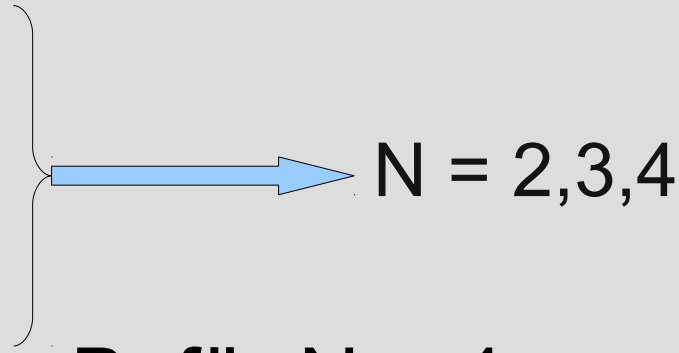
Stack, Stackpointer, UP

- **Stack** (Stapel, FIFO) wird benötigt
- UP-Ruf: Adresse des auf den Cal-Befehl folgenden Befehls wird auf den Stapel gelegt (**PUSH**)
- UP-Rücksprung: oberste Seite auf Stapel wird in PC geladen (**POP**)
- LC-1 Mikrobefehle:
  - cal adr
    Memory(SP) := PC
    SP := SP -1
    PC := adr
  - ret
    SP := SP +1
    PC := Memory(SP)

Beispiel

- Berechnung von **N! = N*(N-1)*....*3*2*1** unter Verwendung eines Unterprogramms **MUL** mit der Funktion **A:=A*B**
- Lösungsansatz:

  1! = 1 = 1
  2! = 2*1 = 2
  3! = 3*2*1 = 6
  4! = 4*3*2*1 = 24



- N! mit 1 initialisieren und z.B. für N = 4

  4! = (((1 * 4) * 3) *2)

Beispiel (2)

- Hauptprogramm:

```
L1:     LDA     N
        LDB     N!
        CAL     MUL
        MOV     N!
        LDA     N
        LDB     EINS
        SUB
        MOV     N
        SUB
        SUB
        JPS     L99
        JMP     L1
L99:    HLT
EINS: DEF     1
N:      DEF     5
N!:     DEF     1
```

Beispiel (3)

- Unterprogramm **MUL**

```
M1:     LDA   FAK1
        LDB   EINS
        SUB
        MOV   FAK1
        JPS   M99
        LDA   PROD
        LDB   FAK2
        ADD
        MOV   PROD
        JMP   M1
M99:    HLT
EINS:   DEF   1
FAK1:   DEF   5
FAK2:   DEF   2
PROD:   DEF   0
```

Beispiel (4)

- UP soll **A := A * B** ausführen
  - HP lädt A und B in die Register A und B
  - UP liest in Register A und B und speichert Zahlen in „lokalen“ Speicherzellen ab
  - UP lädt vor Rückspriung PROD in Reg. A
  - HP entnimmt aus Reg. A das Ergebnis des UP
- Bei jedem Aufruf des UP:

```
LDA  NULL
MOV  PROD
```

- Nicht stoppen, sondern Rücksprung ins rufende Programm

Beispiel (5)

- Unterprogramm **MUL**

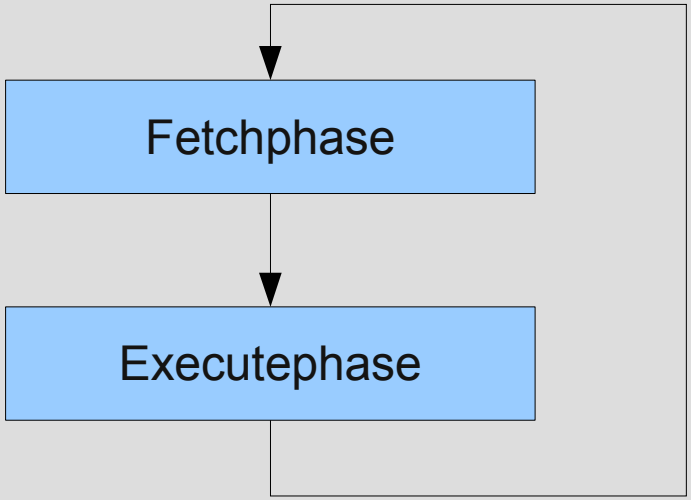
```
MUL:    MOV    FAK1
        LDA    NULL
        MOV    PROD
        ADD
        MOV    FAK2
M1:     LDA    FAK1
        .....
M99:    LDA    PROD
        RET
NULL: DEF      0            ; lokal
EINS:  DEF     0            ;  global
FAK1:  DEF     0            ;  lokal
FAK2:  DEF     0            ;
PROD: DEF      0            ;
```

Interruptsystem

- LC1 verfügt über ein rudimentäres Interruptssystem
  Reaktion auf
  - Fehlersituationen (Netzausfall, Hardwarefehler)
  - von außen veranlasste Eingabe von Daten
  - Betätigung sog. hot keys
- aber

Interruptsystem (2)

- von-Neumann: abwechselnd Fetch und Executephase

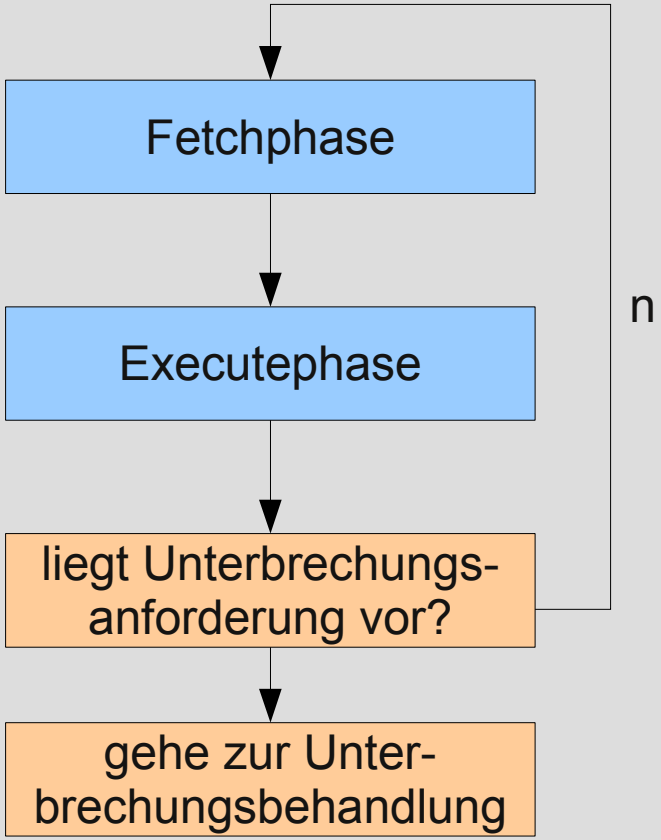


Interruptsystem (3)

- Polling
  - selten anfragen: „träge“ Reaktion auf Interrupt
  - häufig abgefragt: „Verschwendung von Rechenzeit
- Interrupt
  - zusätzlich zur zentralen Steuerschleife:
  - hardwaremäßiges Abfragen, ob Unterbrechungsanforderung vorliegt.

Interruptsystem (4)

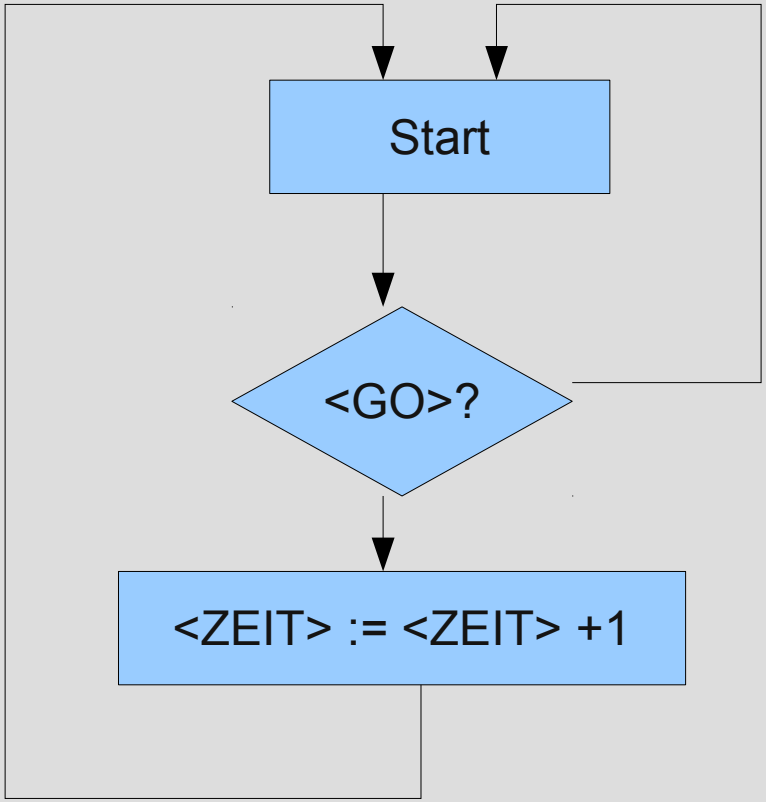
- Interrupt- oder Unterbrechungsverfahren:



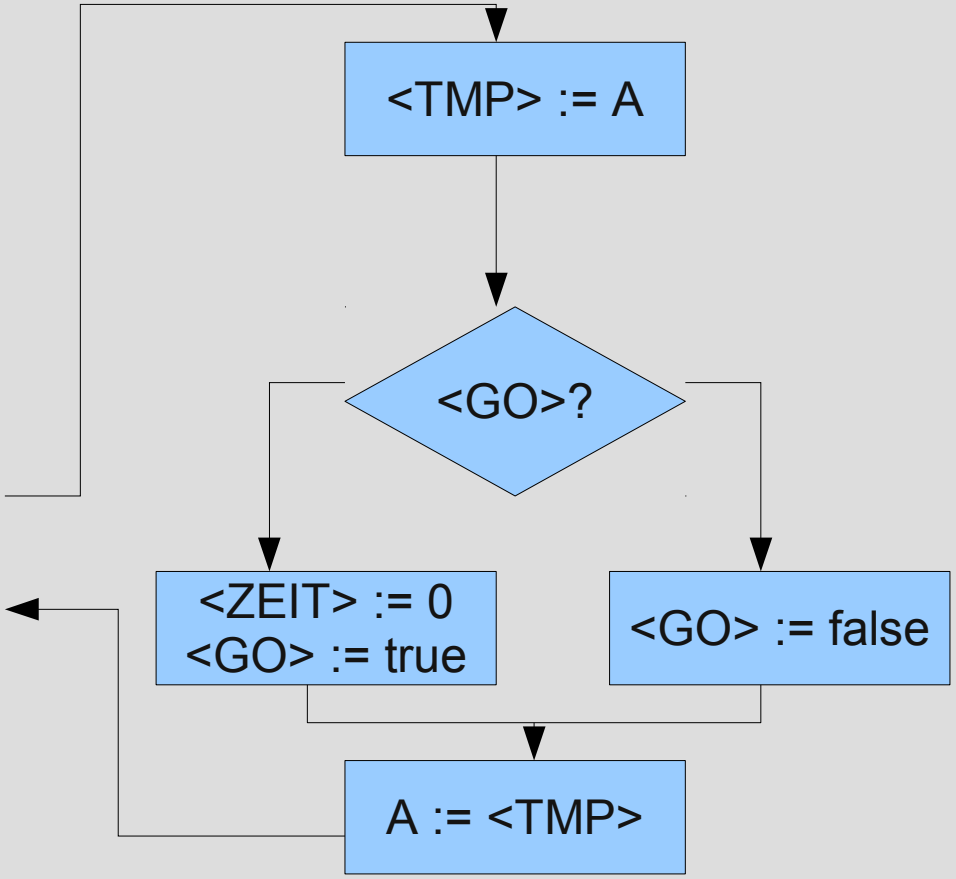
Beispiel

- Stoppuhr

Hauptprogramm



Interruptserviceroutine



Beispiel (3)

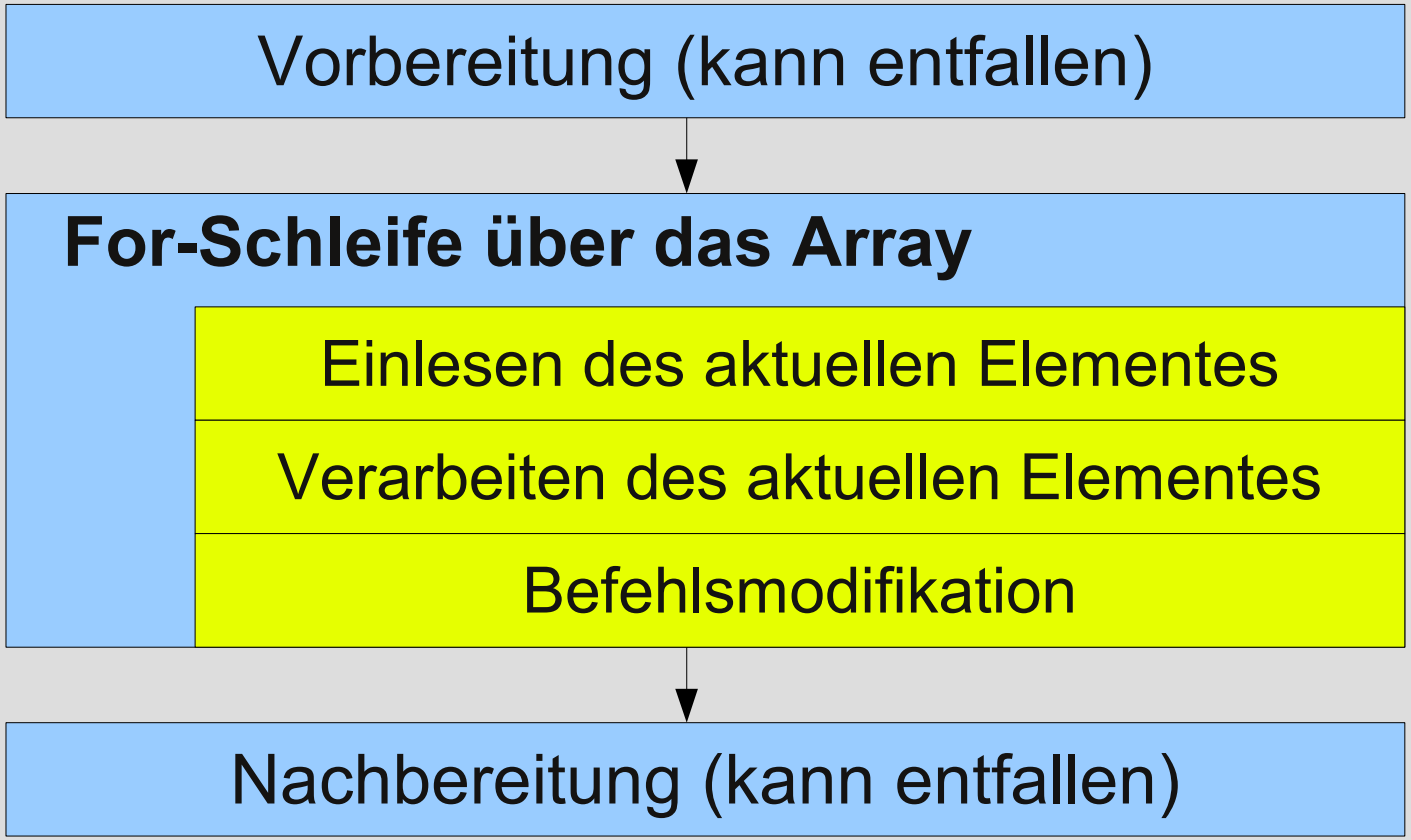
```
ISR:   MOV   TMP
       LDA   GO
       JPS   ISR1
       LDA   MINUS1
       JMP   ISR2
ISR1:  LDA   NULL
       MOV   ZEIT
       LDA   PLUS1
ISR2:  MOV   GO
       LDA   TMP
       RET
```

Beispiel (2)

```
START:  LDA   GO
        JPS   START
        LDA   ZEIT
        LDB   PLUS1
        ADD
        MOV   ZEIT
        JMP   START
;
|
```

„Softwareentwicklung mit LC1“

- Elementeweises Lesen und Verarbeiten eines Feldes



„Softwareentwicklung mit LC1“

- Aufgabe:

in der Adresse ANZ sei eine vorzeichenlose ganze Zahl n, auf der Adresse WERT eine vorzeichenbehaftete Zahl y und ab der Adresse Zahl ein Feld von n vorzeichenbehafteten Zahlen x_i (i = 1 ... n) abgelegt.

Gesucht ist die Anzahl der Elemente des Feldes, für die gilt $|x_i| = |y|$. Das Zählergebnis soll in Register A abgelegt werden.

„Softwareentwicklung mit LC1“

- Lösungsansatz

Für jedes Element x_i ist der Vergleich „$|x_i| = |y|$“ auszuführen. Abhängig vom Vergleichsergebnis ist eine Zählvariable zu inkrementieren oder nicht.

Das Zählergebnis ist in Register A abzulegen. Da A auch für andere Aufgaben zu verwenden ist, muss eine Zählvariable z im Speicher angelegt werden. Während der „Nachbehandlung“ ist dieser Wert dann in das Register A zu transportieren.

„Softwareentwicklung mit LC1“

- Lösungsansatz

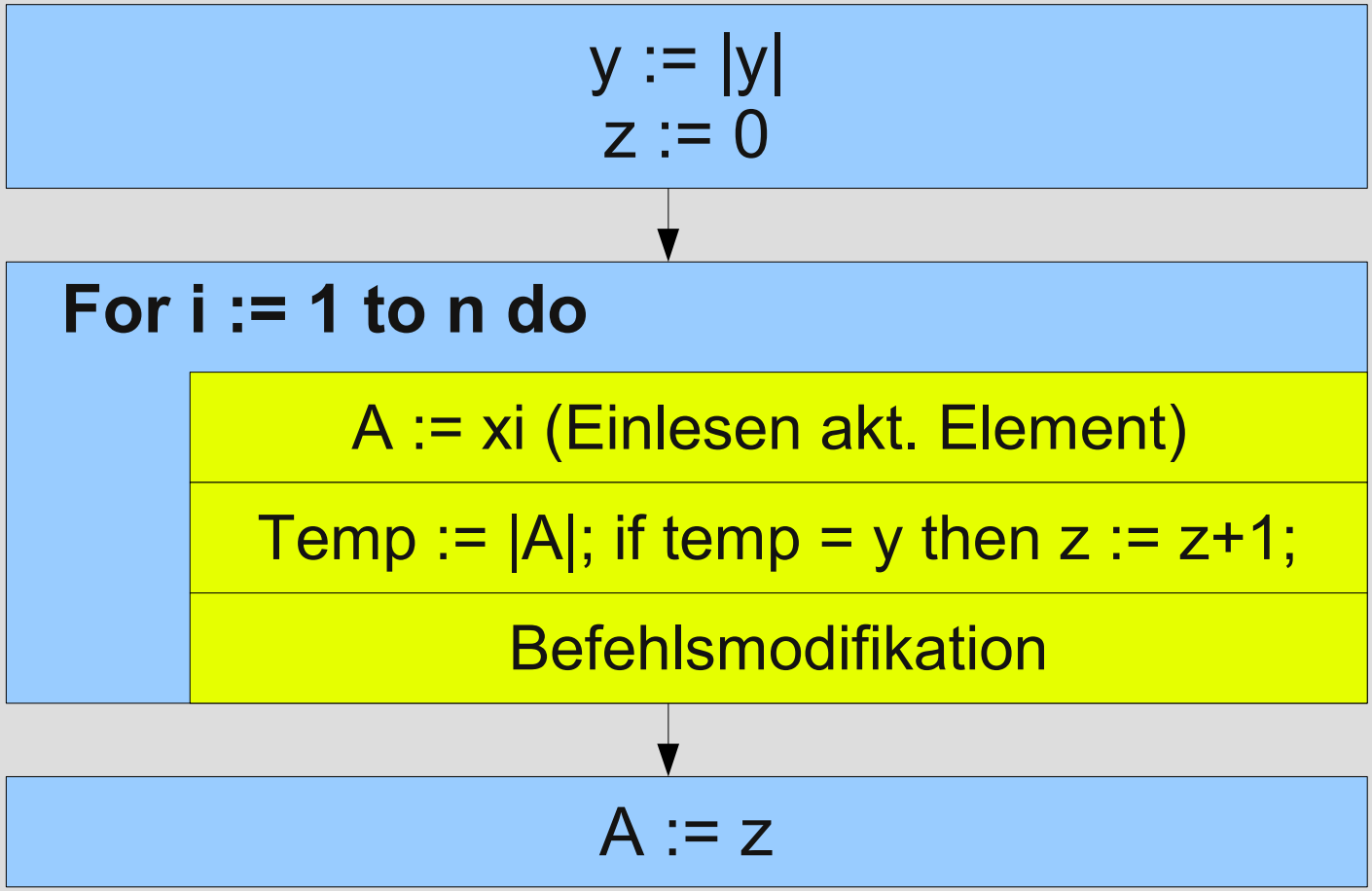
Der Wert |y| bleibt für alle Vergleiche $|x_i| = |y|$ gleich. Deshalb sollte die Betragsbildung innerhalb einer einmaligen „Vorbehandlung“ erfolgen, z.B. durch Überschreiben von y durch |y|.

„Softwareentwicklung mit LC1“

- Lösungsansatz

Die Verarbeitung der aktuellen Elemente besteht dann aus der Betragsbildung $|x_i|$, dem Vergleich $|x_i| = |y|$ und dem ergebnisabhängigen Inkrementieren von z.

„Softwareentwicklung mit LC1“



„Softwareentwicklung mit LC1“

- Lösung:
- Datendeklarationen, Konstanten, Variablen

```
NULL:  DEF  0
EINS:  DEF  1
ANZ:   DEF  5
TEMP: DEF  0   ;|XI|
ZZ:      DEF  0   ; z
WERT: DEF  -12; Y
ZAHL:  DEF 12 ; X1
          DEF -8 ; X2
          DEF -12; X3
...
```

„Softwareentwicklung mit LC1“

- Lösung:
- FOR-Schleife

```
M10: LDA     ANZ
     LDB     EINS
     SUB
     MOV     ANZ
     JPS     M90  ; IF A < 0 GOTO M90
;    Lesen des aktuellen Elementes
;    Verarbeiten des aktuellen Elementes
;    Befehlsmodifikation
     JMP M10
M90: ; Nachbehandlung
```

„Softwareentwicklung mit LC1“

- Lösung:
- Lesen, Befehlsmodifikation

```
M20:    LDA ZAHL
; Verarbeiten des aktuellen Elementes
M80:    LDA M20
        LDB EINS
        ADD
        MOV M20
```

„Softwareentwicklung mit LC1“

- Lösung:
- Vorbehandlung

```
LDA     NULL
LDB     WERT
SUB
JPS     M10     ; dann war Wert > 0!
MOV     WERT
....
```

„Softwareentwicklung mit LC1“

- Lösung:
- Verarbeitung

```
; Betragsbildung |Xi|
M20:    LDA  ZAHL
        MOV TEMP
        MAB
        LDA  NULL
        SUB
        JPS M21
        MOV TEMP
```

„Softwareentwicklung mit LC1“

- Lösung:
- Verarbeitung

```
; Vergleich
M21:    LDA  TEMP
        LDB  WERT
        SUB
        JPS   M80  ; A <> B !!!
         LDB  EINS    ; wenn erg = 0, then erg-1 neg!
         SUB
         JPS  M22
         JMP  M80
```

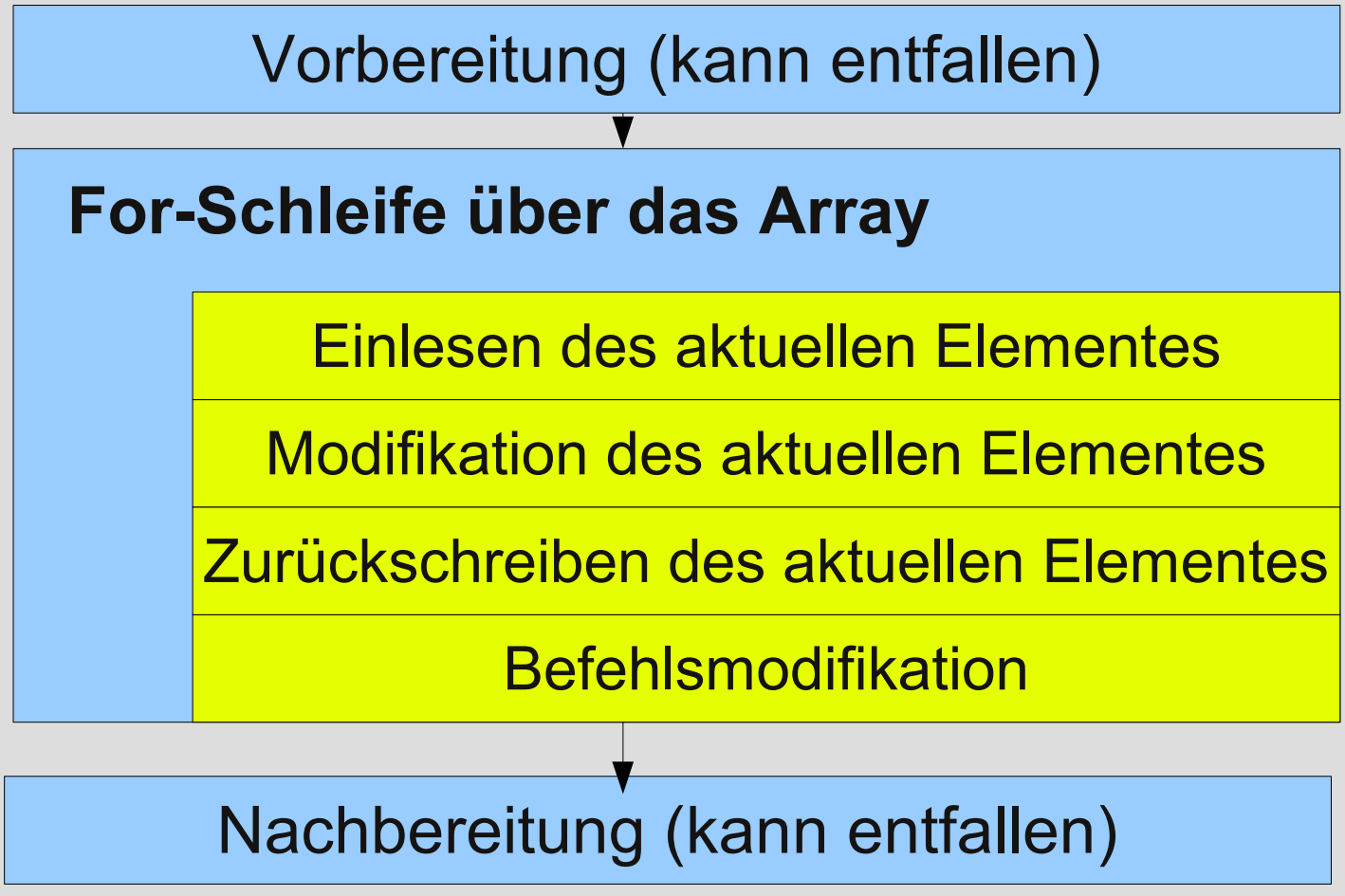
„Softwareentwicklung mit LC1“

- Lösung:
- Verarbeitung

```
; Inkrementieren
M22:    LDA ZZ
        LDB EINS
        ADD
        MOV ZZ
```

„Softwareentwicklung mit LC1“

- Elementeweises Modifizieren eines Feldes am Ort



„Softwareentwicklung mit LC1“

- Beispielaufgabe:

Bilden des Betrages jedes Elementes x_i eines Feldes aus n Elementen

- Lösungsansatz:
  - Vorbereitung und Nachbehandlung entfallen
  - Modifikation beschränkt sich auf die Operation
    - Xi = |Xi|

„Softwareentwicklung mit LC1“

- Lösung: Modifikation

```
M20:  LDA ZAHL
      JPS  M21
      JMP M30     ; oder besser JMP M80
M21:  MAB
      LDA NULL
      SUB
M30:  MOV ZAHL
M80:  ; Befehlsmodifikation  (Lade- und
      ; Rückschreibbefehl modifizieren!)
```

„Softwareentwicklung mit LC1“

- Lösung: Befehlsmodifikation

```
M80:   LDB   EINS    ; B := 1
       LDA   M20     ; A := <M20>
       ADD           ; A := A + B
       MOV   M20     ; <M20> := A
       LDA   M30     ; A := <M30>
       ADD           ; A := A + B
       MOV   M30     ; <M30> := A
       JMP   M10     ; GOTO M10
```