

P-Code-Handbuch

Dieses Kapitel erklart den P-Code der PL/0-VM und bietet Beispiele je Instruktion. Die detaillierte Instruktionsliste wird im kuratierten Kapitel aus [docs/VM_INSTRUCTION_SET.md](#) aufbereitet.

Register und Speicher

- P: Program Counter
- B: Base Pointer
- T: Top-of-Stack
- I: Instruction Register

Die VM arbeitet mit einem Stack und einer separaten Code-Area.

Instruktions-Referenz

Dieses Kapitel beschreibt alle P-Code-Elemente, die die VM versteht.

Grundform

OPCODE LEVEL ARG

- **OPCODE**: Mnemonic (z. B. LIT, LOD)
- **LEVEL**: Lexikalische Ebene (statisch verschachtelte Prozeduren)
- **ARG**: Argument oder Adresse

Instruktionen

LIT

Laedt eine Konstante auf den Stack.

LOD

Laedt eine Variable von einer (LEVEL, ARG)-Adresse auf den Stack.

STO

Speichert den Wert vom Stack in eine Variable an (LEVEL, ARG).

CAL

Ruft eine Prozedur an der Zieladresse auf und legt einen neuen Stack-Frame an.

INT

Reserviert Speicher auf dem Stack fuer lokale Variablen.

JMP

Unbedingter Sprung zur Zieladresse.

JPC

Bedingter Sprung zur Zieladresse, wenn der Stack-Top 0 (false) ist.

OPR

Fuehrt Operationen aus. Die Bedeutung von **ARG** ist:

- 0: Return (Rueckkehr aus Prozedur)

- 1: Negation
- 2: Addition
- 3: Subtraktion
- 4: Multiplikation
- 5: Division (Integer)
- 6: odd-Test
- 8: Gleichheit
- 9: Ungleichheit
- 10: Kleiner
- 11: Groesser gleich
- 12: Groesser
- 13: Kleiner gleich
- 14: Read (Eingabe)
- 15: Write (Ausgabe)

Die vollstaendige Referenz wird zusaetzlich aus dem VM-Instruction-Set kuratiert eingebunden.

Beispiele je Instruktion

Dieses Kapitel zeigt fuer jede Instruktion ein einfaches Beispiel. Die Kommentare erklaeren jeweils die Wirkung auf Stack und Kontrollfluss.

LIT

```
LIT 0 5      // push 5
```

LOD

```
LOD 0 0      // load variable at level 0, offset 0
```

STO

```
LIT 0 7  
STO 0 0      // store 7 into variable 0
```

CAL

```
CAL 0 10     // call procedure at address 10
```

INT

```
INT 0 3      // reserve 3 cells for locals (plus 3 control cells)
```

JMP

```
JMP 0 20     // jump to instruction 20
```

JPC

```
LIT 0 0  
JPC 0 12     // if top is 0, jump to 12
```

OPR (Auswahl)

Return (OPR 0 0)

```
OPR 0 0      // return from procedure
```

Negation (OPR 0 1)

```
LIT 0 4
OPR 0 1      // result: -4
```

Addition (OPR 0 2)

```
LIT 0 1
LIT 0 2
OPR 0 2      // result: 3
```

Subtraktion (OPR 0 3)

```
LIT 0 10
LIT 0 3
OPR 0 3      // result: 7
```

Multiplikation (OPR 0 4)

```
LIT 0 6
LIT 0 7
OPR 0 4      // result: 42
```

Division (OPR 0 5)

```
LIT 0 10
LIT 0 3
OPR 0 5      // result: 3 (Integer)
```

odd (OPR 0 6)

```
LIT 0 5
OPR 0 6      // result: 1
```

Vergleich (OPR 0 8..13)

```
LIT 0 2
LIT 0 2
OPR 0 8      // result: 1 (gleich)
```

Read (OPR 0 14)

```
OPR 0 14      // read int from input, push on stack
```

Write (OPR 0 15)

```
LIT 0 9
OPR 0 15      // write 9 to output
```

Ausfuehrliches Mini-Programm (Summe)

```
INT 0 3      // locals: x, y, sum
LIT 0 2
STO 0 0
LIT 0 3
STO 0 1
LOD 0 0
LOD 0 1
OPR 0 2      // x + y
STO 0 2
LOD 0 2
OPR 0 15      // print sum
```

Ausfuehrliche Beispiele

In diesem Kapitel folgen groessere P-Code-Programme mit kurzen Erklaerungen.

Beispiel 1: Summe von 1 bis N

Berechnet die Summe 1..N und gibt das Ergebnis aus. Lokale Variablen: **n** (0), **i** (1), **sum** (2).

```
INT 0 3
OPR 0 14      // read n
STO 0 0
LIT 0 1
STO 0 1
LIT 0 0
STO 0 2
LOD 0 1
LOD 0 0
OPR 0 10      // i < n
JPC 0 18
LOD 0 2
LOD 0 1
OPR 0 2      // sum + i
STO 0 2
LOD 0 1
LIT 0 1
OPR 0 2      // i + 1
STO 0 1
JMP 0 7
LOD 0 2
OPR 0 15      // print sum
OPR 0 0      // return
```

Beispiel 2: Maximum von zwei Zahlen

```
INT 0 3
OPR 0 14
STO 0 0
OPR 0 14
STO 0 1
LOD 0 0
LOD 0 1
OPR 0 12      // a > b
JPC 0 12
LOD 0 0
```

```
STO 0 2
JMP 0 14
LOD 0 1
STO 0 2
LOD 0 2
OPR 0 15
OPR 0 0
```

Debugging-Tipps

- P-Code mit `--list-code` ausgeben.
- Schrittweise durch Instruktionen gehen und Stackzustand notieren.
- Fehlerhafte Sprungadressen pruefen.