

# 1 ISA - Instruction Set Architecture

Prozessoren mit der selben ISA sind binaer kompatibel, jedoch variierende Geschwindigkeit.

## 1.1 Befehlssatz

Drei Hauptklassen an Befehlen:

- arithmetische und logische Operationen
- Datentransfer
- Steuerung des Programmablaufs

Zusaetzlich:

- Systembefehle fuer Verwaltungsaufgaben, nur fuer das Betriebssystem
- I/O-Befehle fuer I/O-Gerte und (spezielle) Speicherbereiche

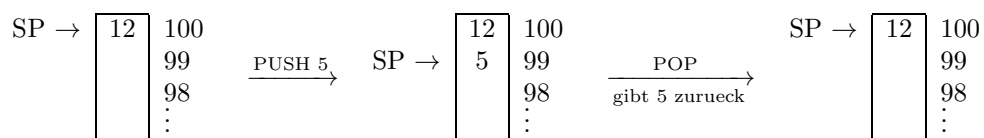
Befehle:

- Ganzzahlarithmetik (ADD, SUB, CMP, MUL, ...) - Statusregister nutzbar
- Gleitkommaarithmetik (FADD, FSUB, FNEG, FABS, ...)
- Logische Befehle (NOT, AND, XOR, ...)
- Sprungbefehle (Statusregister, CMP) - Realisierung von Schleifen/Verzweigungen
- Unterprogrammaufruf (CALL) - Realisierung von Methodenaufrufen

## 1.2 Der Stack

Eine Liste von Werten, die aufeinanderliegen. Das Ende wird durch den Stackpointer (SP) gekennzeichnet und funktioniert nach dem LIFO Prinzip: Last In First Out.

- Es koennen Werte auf den Stack gelegt werden durch PUSH
- Es koennen Werte entnommen werden durch POP



Der Stack waechst demnach nach unten im Kontrast zum Heap welcher nach oben waechst.

→ Achtung dabei koennen beide Speicherbereiche aufeinandertreffen und Error erzeugen.

### 1.3 Unterprogrammaufruf

- Implementiert bestimmte Funktionalitaet (Methodenaufruf)
- Aufruf durch Call
- nach Ende des Call-Segments: Weiterfuehrung von der Stelle des Aufrufs +1
- die Ruecksprungadresse (Ruecksprung durch RET) liegt auf dem Stack

100: Work

...

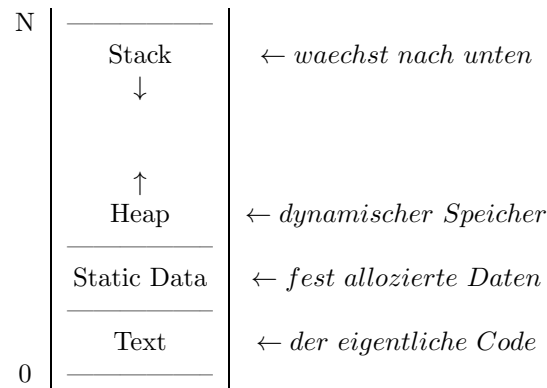
123: CALL abc  $\leftarrow$  *Sichern von 123 + 1 und Sprung zu abc*

124 : Work  $\leftarrow$  *Nach RET wird hier das Programm weitergefuehrt*

...

### 1.4 Typische Speicherverwaltung

Jedes Programm besitzt einen Adressraum welcher alle noetigen Daten des Programms enthaelt:



N = Speicheradresse

Der Heap ist ein dynamischer Speicher welcher angefragt und freigegeben werden kann. Waechst im Gegensatz zum Heap nach oben.

$\rightarrow$  *Bei Treffen von Heap und Stack : Error*

### 1.5 Datentransferbefehle

- LOAD/STORE: Transfer Speicher
- MOVE: Register
- Unterschiedliche Befehle fuer Ganzzahl- und Gleitkommaregister

## 1.6 Adressierungsarten

Jeder Maschinenbefehl muss die zu bearbeitenden Daten (Operanden) spezifizieren.

- Adressierung: Spezifizierung der Operanden eines Befehls
- Adressierungsart: Konkrete Methode zur Spezifizierung eines Operanden
- Unmittelbare Adressierung: Der Operand steht als Wert im Befehl selbst
- Register-Adressierung: Fetchen des Operanden durch Speicheradresszugriff

Modifikation einer direkten Speicheradressangabe:

- Berechnung aus mehreren Teilen
- Berechnung erfolgt zur Laufzeit
- Ermöglichung von Arrayzugriffen

Registerindirekt mit Präinkrement/Postinkrement bedeutet, dass eine im Register stehende Adresse vor ihrem Zugriff um 1, 2, 4 oder 8 Byte verschoben wird.

Registerindirekt mit Displacement (positiv und negativ) bedeutet, dass zu der im Register stehende Adresse ein konstanter Wert addiert wird. Indizierte Adressierung eines Arrays. Skalierter Zugriff möglich.

Befehlszähler-relative Adressierung besagt, dass eine effektive Adresse relativ zum aktuellen Befehlszählerstand gebildet wird. Zur Adressierung von Befehlen und Daten innerhalb des Programmcodes (Sprünge, Unterprogramme, ...). Ermöglicht positionsunabhängige Programme.

## 1.7 Ausführungsmodelle

Designentscheidung: Wo stehen Operanden bzw. das Ergebnis.

- Explizit: Operand (bzw. Zielort der Operation) wird frei gewählt
- Implizit: Adresse ist vom Befehl fest vorgegeben
- Ueberdeckte Adressierung: Zieladresse ist gleich einer der Quelladressen

Nulladressform - keine explizite Operandenspezifikation:

- Operanden: oberste Stackelemente
- Entfernung aus dem Stack
- Ergebnis wird auf den Stack gepusht.

Einadressform - Operand ist immer ein Register (AC).

Zweiadressform - Das Ergebnis überschreibt ersten Operanden (Intel IA-32).

Dreiadressform - Operanden und Ergebnis werden explizit adressiert oder Operanden und Ergebnis dürfen nur in Registern stehen (letzteres: RISC-CPU's)