

# Praktische Informatik 1

## Operatoren und Ausdrücke

Thomas Röfer

- Operatoren
- Schreibweisen von Operatoren
- Arten von Operatoren
- Vorrang von Operatoren
- Typanalyse von Ausdrücken
- Musterlösung, Übungsblatt



# Typерweiterung

- Automatische Typkonvertierung (Typерweiterung)
  - byte → short → int → long → float → double
  - char →
- Entspricht „unsichtbaren“ Operatoren
  - : byte → short
  - : short → int
  - : char → int
  - : int → long
  - : long → float
  - : float → double
- Wahl der Überladung eines Operators
  - Es wird immer die Überladung eines Operators gewählt, die die wenigsten Typерweiterungen erfordert
  - Ausnahme: konstante Ausdrücke werden nach der Berechnung wieder auf den kleinsten Typ reduziert, in dem das Ergebnis noch darstellbar ist

# Typanalyse von Ausdrücken

- Fragestellung
  - Was ist der Typ jedes Teilausdrucks?
  - Welche Typerweiterungen müssen vorgenommen werden?
- Vorgehen
  - In der Auswertungsreihenfolge des Ausdrucks entsprechend Bindungsstärke, Assoziativität und Klammerung
    - Einsetzen der Typen der Variablen und Literale
    - Wähle jeweils die Überladung eines Operators, deren Operandentypen mit den wenigsten Typerweiterungen von den Typen der Operanden im Ausdruck erreicht werden können
    - Wenn nötig, füge Typerweiterungen ein
    - Ist dies nicht möglich, ist der Ausdruck nicht typkorrekt

```
int i; byte b;  
i > b ? b * 4 : i * 2
```

# Typanalyse

```
int i; byte b;
i > b ? b * 4 : i * 2
```

```
?: : boolean × α × α → α
> : α × α → boolean,
   α ∈ {int, ..., double}
* : α × α → α,
   α ∈ {int, ..., double}
```

$i : \text{int}, b : \text{byte}, 4 : \text{int}, 2 : \text{int}$

$i > b : \text{int} \times ((\text{byte} \rightarrow \text{short}) \rightarrow \text{int}) \rightarrow \text{boolean}$

$b * 4 : ((\text{byte} \rightarrow \text{short}) \rightarrow \text{int}) \times \text{int} \rightarrow \underline{\text{int}}$

$i * 2 : \text{int} \times \text{int} \rightarrow \underline{\text{int}}$

$i > b ? b * 4 : i * 2 : (\text{int} \times ((\text{byte} \rightarrow \text{short}) \rightarrow \text{int}) \rightarrow \text{boolean}) \times$   
 $((\text{byte} \rightarrow \text{short}) \rightarrow \text{int}) \times \text{int} \rightarrow \text{int}) \times$   
 $(\text{int} \times \text{int} \rightarrow \text{int}) \rightarrow \text{int}$

# Musterlösung zu Übungsblatt 5

- Aufgabe 1
  - Punkt- vor Strichrechnung heißt nicht, dass Punktoperationen immer links von Strichoperationen stehen müssen
- Aufgabe 2
  - Iteration ist was mit Schleife
  - Rekursion ist was mit Selbstaufruf
  - Das mit *if* nennt man Verzweigung

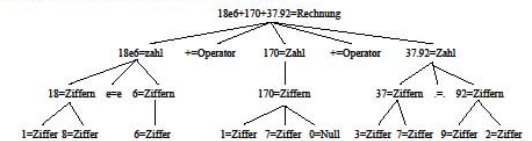
Praktische Informatik I WS 2007/08

## Übungsblatt 5 Musterlösung

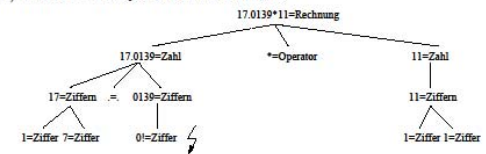
### Aufgabe 1 Grammatiken mit EBNF (40%)

Überprüft, ob die folgenden Sätze der Grammatik entsprechen. Gebt die entsprechenden Regeln an, um eure Antwort zu begründen.

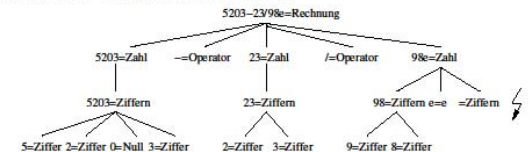
a)  $18e6+170+37.92 \rightarrow$  Entspricht der Grammatik.



b)  $17.0139*11 \rightarrow$  Entspricht nicht der Grammatik.



c)  $5203-23/98e \rightarrow$  Entspricht nicht der Grammatik.



# Übungsblatt 7

- Aufgabe 1
  - 3x Typanalyse
  - 3x Ausdruck auswerten
- Aufgabe 2
  - Von Neumann-Simulator bauen
  - Programm zum Austauschen zweier Zahlen als Testfall für Simulator implementieren
  - Weitere Tests durchführen
- Abgabe
  - 19.12.07, 15:00-15:15 vor MZH 1400 (Weihnachtsvorlesung)

Praktische Informatik I WS 2007/08

## Übungsblatt 7

Abgabe: 19.12.07

### Aufgabe 1 Typisch Ausdrücke (30%)

Gibt die Auswertungsreihenfolge und das Ergebnis der nachfolgenden Ausdrücke an und ermittelt dabei auch jeweils den Typ der (Teil-)Ausdrücke. Dabei gelte jeweils:

byte b = 2; int i = 5, j = 10, k = 20; float r = 5.0f; double d = 2.5;

- a)  $j = (\text{int}) (r * b) + i \ll j$
- b) "Hallo" + i + (j - i % (j + k))
- c)  $b + k \% (\text{byte}) r = r / d \ ? \ 3 * b : k + d$

### Aufgabe 2 Softcomputing (70%)

Schreibt einen Simulator, der die von-Neumann-Architektur nachbildet. Er soll als *uebung07* im Repository abgelegt werden. Man soll ihm eine Reihung von Registern (*int*-Werte) und den Hauptspeicher (ebenfalls eine Reihung von *int*-Werten) übergeben können. Der Instruktionszähler ist eines der Register, z.B. Register 0. Der Simulator soll das im Speicher abgelegte Programm ausführen. Die Ausführung terminiert, wenn der Instruktionszähler einen Wert erreicht, der außerhalb des Hauptspeichers liegt.

Der Simulator soll nur vier Instruktionen kennen. Diese werden durch einzelne Zahlen im Hauptspeicher repräsentiert (*int*-Werte, die ihr euch ausdenken könnt) auf die dann ihre Parameter einer nach dem anderen, ebenfalls als Zahlen, folgen. Alle Instruktionen haben zwei Parameter, d.h. die Befehle belegen jeweils drei Speicherstellen:

**MRI param1 param2.** Speichert die Zahl *param2* in das Register mit der Nummer *param1*.

**MRR param1 param2.** Kopiert den Inhalt des Registers mit der Nummer *param2* in das Register mit der Nummer *param1*.

**MRM param1 param2.** Liest den Inhalt der Speicherzelle aus, deren Adresse im Register mit der Nummer *param2* steht, und speichert den Wert im Register mit der Nummer *param1*.

**MMR param1 param2.** Schreibt den Inhalt des Registers mit der Nummer *param2* in die Speicherzelle, deren Adresse im Register mit der Nummer *param1* steht.

Zum Testen füllt ihr den Hauptspeicher mit einem Programm, das die zwei Speicherstellen miteinander vertauscht, deren Adressen beim Aufruf in den Registern 1 und 2 stehen. Euer simulierter Prozessor hat nicht mehr als vier Register. Führt den Simulator dann mit einer geeigneten Anfangsbelegung dieser Register aus.

**Frage:** Kann euer Simulator bereits Schleifen ausführen? Wenn ja, wie?

**Tipp:** Damit man nachvollziehen kann, was der Simulator macht, solltet ihr den aktuell verarbeiteten Befehl sowie den Zustand von Speicher und Registern als Text in der Konsole ausgeben. Wenn *param1* und *param2* *int*-Variablen sind, kann man z.B. schreiben: `System.out.println("MRI R" + param1 + " = " + param2);`

# Übungsblatt 7 – Aufgabe 1

## Typanalyse und TeX

- $a \% b : int \times int \rightarrow int$ 
  - `\texttt{a \% b} :`  
`$int \times int \mapsto int$`
  - `\verb|a \% b| :`  
`$int \times int \mapsto int$`
- `"A" + "B" : ...`
  - `\texttt{\dq A\dq + \dq B\dq} : \ldots`
  - `\verb|"A" + "B"| : \ldots`

?:  $boolean \times \alpha \times \alpha \rightarrow \alpha$   
 $+, -, *, /, \% : \alpha \times \alpha \rightarrow \alpha,$   
 $\alpha \in \{int, \dots, double\}$   
 $+ : String \times \alpha \rightarrow String,$   
 $\alpha \in \{long, double\}$   
 $+ : \alpha \times String \rightarrow String,$   
 $\alpha \in \{long, double\}$   
 $<<, >>, <<< : \alpha \times \alpha \rightarrow \alpha,$   
 $\alpha \in \{int, long\}$   
 $(type) : \alpha \rightarrow type$



# Übungsblatt 7 – Aufgabe 1

## Auswertung eines Ausdrucks

•  $\text{int } i = \overset{3}{2}; \text{ double } d = 3.0;$

•  $\underline{i}++ * (i + 9 / d)$

$$\underline{2} * (i + 9 / d)$$

$$2.0 * (\underline{i} + 9 / d)$$

$$2.0 * (\underline{3} + 9 / d)$$

$$2.0 * (3.0 + \underline{9} / d)$$

$$2.0 * (3.0 + 9.0 / \underline{d})$$

$$2.0 * (3.0 + \underline{9.0 / 3.0})$$

$$2.0 * (\underline{3.0 + 3.0})$$

$$2.0 * 6.0$$

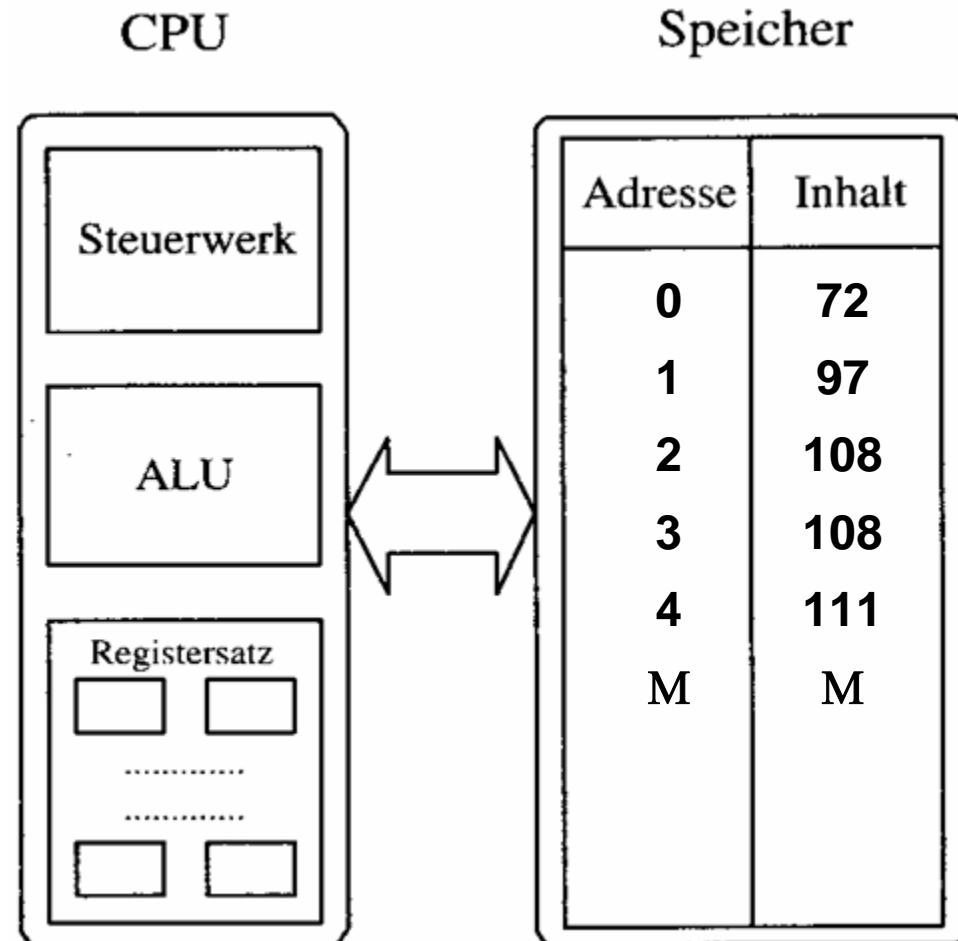
$$12.0$$



# Übungsblatt 7 – Aufgabe 2

## von Neumann Architektur

- Steuerwerk
  - Führt Programme aus
- Arithmetisch-logische Einheit (ALU)
  - Rechnet
- Registersatz
  - Speichert Daten im Prozessor
- Jede Speicherzelle hat
  - eine Adresse
  - einen Inhalt
- Programm und Daten im selben Speicher



# Übungsblatt 7 – Aufgabe 2

## Instruktionszyklus einer CPU

- Fetch
  - Hole Befehl, dessen Adresse im Befehlszähler steht, in das Instruktionsregister
- Increment
  - Erhöhe den Befehlszähler, damit er danach auf die nächste auszuführende Instruktion verweist
- Decode
  - Dekodiere die Instruktion, damit klar wird, was zu tun ist
- Fetch Operands
  - Falls nötig, hole die Operanden aus den im Befehl bezeichneten Stellen im Speicher
- Execute
  - Führt die Instruktion aus, ggf. durch die ALU
  - Bei einem Sprung wird ein neuer Wert in den Befehlszähler geschrieben
- Loop
  - Beginne von vorne

# Übungsblatt 7 – Aufgabe 2

Register      Inhalt

R0	
R1	5
R2	3
R3	5

MR1 R1, 5  
MR R3, R1

MRM R2, (R1)

MMR (R2), R1

Adresse      Inhalt

0	
1	
2	
3	5
4	
5	3
6	
7	

# Übungsblatt 7 – Aufgabe 2

## Beispiel: (Andere) Befehle

- MRI param1, param2
  - Speichert die Zahl *param2* in das Register mit der Nummer *param1*
- SUB param1, param2
  - Zieht das Register mit der Nummer *param2* vom Register mit der Nummer *param1* ab
  - Das Ergebnis wird im Register mit der Nummer *param1* gespeichert
- MUL param1, param2
  - Multipliziert das Register mit der Nummer *param1* mit dem Register mit der Nummer *param2*
  - Das Ergebnis wird im Register mit der Nummer *param1* gespeichert
- JNZ param1, param2
  - Wenn das Register mit der Nummer *param1* ungleich 0 ist, wird das Programm an Adresse *param2* fortgesetzt

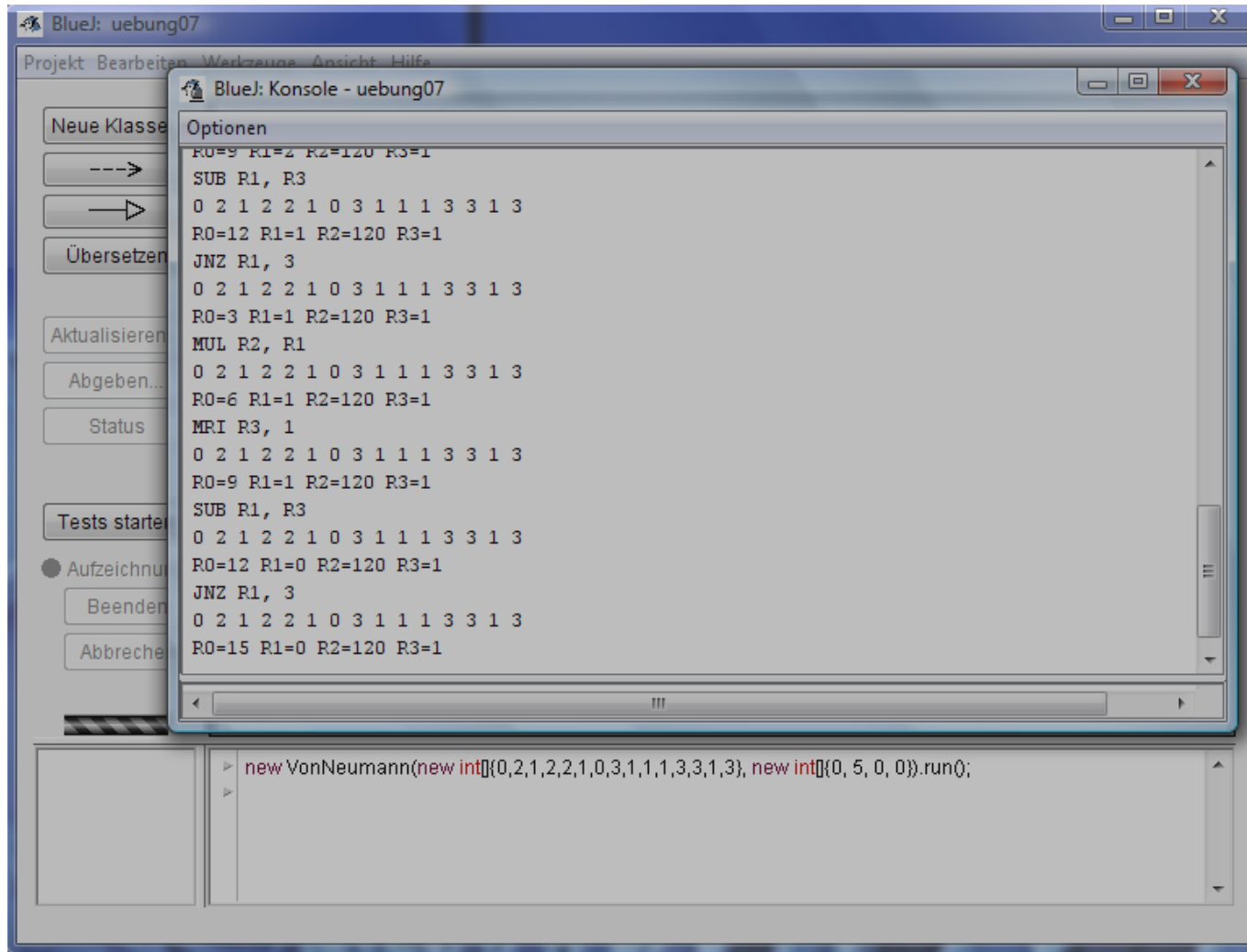
# Übungsblatt 7 – Aufgabe 2

Beispiel: R1!  $\rightarrow$  R2

0 0 2 1 MRI R2, 1  
3 2 2 1 MUL R2, R1  
6 0 3 1 MRI R3, 1  
9 1 1 3 SUB R1, R3  
12 3 1 3 JNZ R1, 3

MRI 0  
SUB 1  
MUL 2  
JNZ 3

# Übungsblatt 7 – Aufgabe 2



```
BlueJ: uebung07
Projekt Bearbeiten Werkzeuge Ansicht Hilfe

Neue Klasse
--->
-->
Übersetzen
Aktualisieren
Abgeben...
Status
Tests starten
Aufzeichnung
Beenden
Abbrechen

Options
R0=9 R1=2 R2=120 R3=1
SUB R1, R3
0 2 1 2 2 1 0 3 1 1 1 3 3 1 3
R0=12 R1=1 R2=120 R3=1
JNZ R1, 3
0 2 1 2 2 1 0 3 1 1 1 3 3 1 3
R0=3 R1=1 R2=120 R3=1
MUL R2, R1
0 2 1 2 2 1 0 3 1 1 1 3 3 1 3
R0=6 R1=1 R2=120 R3=1
MRI R3, 1
0 2 1 2 2 1 0 3 1 1 1 3 3 1 3
R0=9 R1=1 R2=120 R3=1
SUB R1, R3
0 2 1 2 2 1 0 3 1 1 1 3 3 1 3
R0=12 R1=0 R2=120 R3=1
JNZ R1, 3
0 2 1 2 2 1 0 3 1 1 1 3 3 1 3
R0=15 R1=0 R2=120 R3=1

new VonNeumann(new int[]{0,2,1,2,2,1,0,3,1,1,1,3,3,1,3}, new int[]{0, 5, 0, 0}).run();
```

## Übungsblatt 7 – Aufgabe 2

- Sonderfall
  - Was passiert, wenn eine Instruktion ausgeführt werden soll, die es nicht gibt?
- Programm in Maschinensprache
  - Austauschen von zwei Speicherstellen
  - Nur 4 Register (inkl. Instruktionszähler)!
- Frage
  - Kann man mit den vier Befehlen schon Schleifen realisieren?