Algorithmen und Datenstrukturen

Praktische Einführung und Programmierung

Stefan Bosse

Universität Koblenz - FB Informatik

Formale Algorithmenmodelle

ad-dpunkt 6.1

Registermaschine

Als erstes formales Modell betrachten wir die sogenannten Registermaschinen. Registermaschinen befinden sich von der Abstraktion nahe an tatsächlichen Computern und bilden daher quasi eine Art »idealisierte« Version eines Prozessors mit Maschinencode-Steuerung.

> Eine Registermaschine als Prozessor führt ein Maschinenprogramm aus, das als durchnummerierte Liste von Einzelschritten vorliegt (dies entspricht tatsächlichen Maschinenprogrammen in Rechnern). Die einzige Kontrollstruktur neben der durch die Nummerierung impliziten Sequenz bilden (bedingte) Sprünge, die es erlauben, an einer beliebigen Stelle des Programms mit der Ausführung weiterzufahren.

Registermaschine

- Daten werden in einem direkt adressierbaren Speicher gehalten, der eine Abstraktion des bekannten Hauptspeichers darstellt.
    - Als theoretisches Modell betrachten wir dabei einen beliebig großen, unbeschränkten Speicher. -
- Arithmetische Manipulationen werden ausschließlich im Akkumulatorregister des Prozessors ausgeführt.

Diese (maschinennahe) Präzisierung des Algorithmenbegriffs werden wir nun als Registermaschinen exakt definieren. Diese stellen eine relativ simple Abstraktion der programmierbaren Rechner dar, so dass wir in den Bezeichnungen auf vertraute Begriffe der technischen Informatik zurückgreifen.

Registermaschine

Eine Registermaschine besteht aus den **Registern**

`B, C0, C1, C2, …, Cn, …`

und einem Programm.

Das Register B heißt Befehlszähler, C0 heißt Arbeitsregister oder Akkumulator und jedes der Register Cn, $n \geq 1$ heißt Speicherregister.

Jedes Register enthält als Wert eine natürliche Zahl. Enthält das Register B die Zahl b und für $n \geq 0$ das Register Cn die Zahl cn, so heißt das unendliche Tupel

`(b, c0, c1, …, cn, …)`

Konfiguration der Registermaschine.

Das Programm ist eine endliche Folge von Befehlen. Durch die Anwendung eines Befehls wird die Konfiguration der Registermaschine geändert.

Def. 1. Definition einer Registermaschine

Registermaschine

Befehle bewirken eine Änderung einer Konfiguration (Zustandsänderung ↦ Imperative Algorithmen!)

`(b, c0, c1, … cn, …)`

in die neue Konfiguration

`(b′, c′0, c′1, …, c′n, …),`

geschrieben als

`(b, c0, c1, … cn, …) ↦ (b′, c′0, c′1, …, c′n, …),`

an.

Registermaschine

LOAD i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_0 = c_i$	$c'_j = c_j$ für $j \neq 0$
CLOAD i ,	$i \in \mathbb{N}$	$b' = b + 1$	$c'_0 = i$	$c'_j = c_j$ für $j \neq 0$
STORE i ,	$i \in \mathbb{N}_+$	$b' = b + 1$	$c'_i = c_0$	$c'_j = c_j$ für $j \neq i$

Def. 2. Ein- und Ausgabebefehle einer Registermaschine

Registermaschine

`ADD i,`	$i \in \mathbb{N}_+$	$b' = b+1$	$c_0' = c_0 + c_i$		$c_j' = c_j$ für $j \neq 0$
`CADD i,`	$i \in \mathbb{N}$	$b' = b+1$	$c_0' = c_0 + i$		$c_j' = c_j$ für $j \neq 0$
`SUB i,`	$i \in \mathbb{N}_+$	$b' = b+1$	$c_0' = \begin{cases} c_0 - c_i & \text{für } c_0 \geq c_i \\ 0 & \text{sonst} \end{cases}$		$c_j' = c_j$ für $j \neq 0$
`CSUB i,`	$i \in \mathbb{N}$	$b' = b+1$	$c_0' = \begin{cases} c_0 - i & \text{für } c_0 \geq i \\ 0 & \text{sonst} \end{cases}$		$c_j' = c_j$ für $j \neq 0$
`MULT i,`	$i \in \mathbb{N}_+$	$b' = b+1$	$c_0' = c_0 \cdot c_i$		$c_j' = c_j$ für $j \neq 0$
`CMULT i,`	$i \in \mathbb{N}$	$b' = b+1$	$c_0' = c_0 \cdot i$		$c_j' = c_j$ für $j \neq 0$
`DIV i,`	$i \in \mathbb{N}_+$	$b' = b+1$	$c_0' = \lfloor c_0 / c_i \rfloor$		$c_j' = c_j$ für $j \neq 0$
`CDIV i,`	$i \in \mathbb{N}_+$	$b' = b+1$	$c_0' = \lfloor c_0 / i \rfloor$		$c_j' = c_j$ für $j \neq 0$

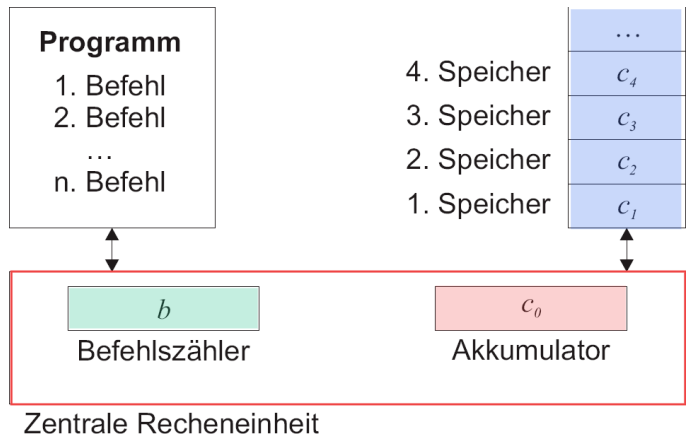
Def. 3. Arithmetische Befehle von Registermaschinen

Registermaschine

`GOTO i,`	$i \in \mathbb{N}_+$	$b' = i$		$c'_j = c_j$ für $j \geq 0$
`IF` $c_0 = 0$ `GOTO i,`	$i \in \mathbb{N}_+$	$b' = \begin{cases} i & \text{falls } c_0 = 0 \\ b+1 & \text{sonst} \end{cases}$		$c'_j = c_j$ für $j \geq 0$
`END`		$b' = b$		$c'_j = c_j$ für $j \geq 0$

Def. 4. Sprung- und Stoppbefehle einer Registermaschine

Registermaschine



Def. 5. Architektur und Aufbau einer Registermaschine

Beispiel 1

```
LOAD 1 DIV 2 MULT 2 STORE 3 LOAD 1 SUB 3 STORE 3 END
```

Bsp. 1. Ein einfaches Maschinenprogramm für die Registermaschine - eine reine lineare Sequenz

Stehen in den Registern zuerst die Zahlen

`b = 1, c0 = 0, c1 = 32, c2 = 5, c3 = 0,`

so ergibt sich diese Folge von Konfigurationen

```
(1, 0, 32, 5, 0, …) ↦ (2, 32, 32, 5, 0, …)
                    ↦ (3, 6, 32, 5, 0, …)
                    ↦ (4, 30, 32, 5, 0, …)
                    ↦ (5, 30, 32, 5, 30, …)
                    ↦ (6, 32, 32, 5, 30, …)
                    ↦ (7, 2, 32, 5, 30, …)
                    ↦ (8, 2, 32, 5, 2, …),
```

womit der »stoppende« Befehl erreicht wird.

Beispiel 1

Sind die Inhalte der Register dagegen

```
b = 1, c0 = 0, c1 = 100, c2 = 20, c3 = 0,
```

so ergibt sich folgende Folge von Konfigurationen

```
(1, 0, 100, 20, 0, …)  ↦ (2, 100, 100, 20, 0, …)
                       ↦ (3, 5, 100, 20, 0, …)
                       ↦ (4, 100, 100, 20, 0, …)
                       ↦ (5, 100, 100, 20, 100, …)
                       ↦ (6, 100, 100, 20, 100, …)
                       ↦ (7, 0, 100, 20, 100, …)
                       ↦ (8, 0, 100, 20, 0, …).
```

Allgemeiner lässt sich Folgendes feststellen. Wir betrachten eine Konfiguration, die durch

```
b = 1, c0 = 0, c1 = n, c2 = m, c3 = 0
```

gegeben ist, und nehmen an, dass $n = q \cdot m + r$ mit $0 \leq r < m$ gilt, d.h., $q = \lfloor n/m \rfloor$ ist das ganzzahlige Ergebnis der Division von n durch m und r ist der verbleibende Rest bei dieser Division.

Beispiel 1

Dann ergibt sich immer die Folge

```
(1, 0, n, m, 0, …)  ↦ (2, n, n, m, 0, …)
                    ↦ (3, q, n, m, 0, …)
                    ↦ (4, q · m, n, m, 0, …)
                    ↦ (5, q · m, n, m, q · m, …)
                    ↦ (6, n, n, m, q · m, …)
                    ↦ (7, r, n, m, q · m, …)
                    ↦ (8, r, n, m, r, …).
```

Diese Berechnungsfolge der Registermaschine M1 berechnet also den ganzzahligen Rest der Division zweier natürlicher Zahlen.

Beispiel 2

```
1. CLOAD 1
2. STORE 3
3. LOAD 2
4. IF c0 = 0 GOTO 12
5. LOAD 3
6. MULT 1
7. STORE 3
8 .LOAD 2
9. CSUB 1
10 .STORE 2
11. GOTO 4
12. END
```

Bsp. 2. Mit Kontrollbefehlen und Verzweigung - jetzt werden die Adressen wichtig!

Stackmaschine

Bisher gabe es eine klassische von-Neumann Rechnerarchitektur bei der Registermaschine. Häufig werden aber Stackmaschinen als Virtuelle Maschinen verwendet.

Meistens gibt es keine reinen Stackmaschinen sondern eine Mischung aus einer Stack- und Registermaschinen.

Registermaschine
: Alle Berechnungen laufen über einen Registerspeicher

Stackmaschine
: Alle Berechnungen laufen über einen oder mehrere Stack(Stapel)speicher

Stackmaschine

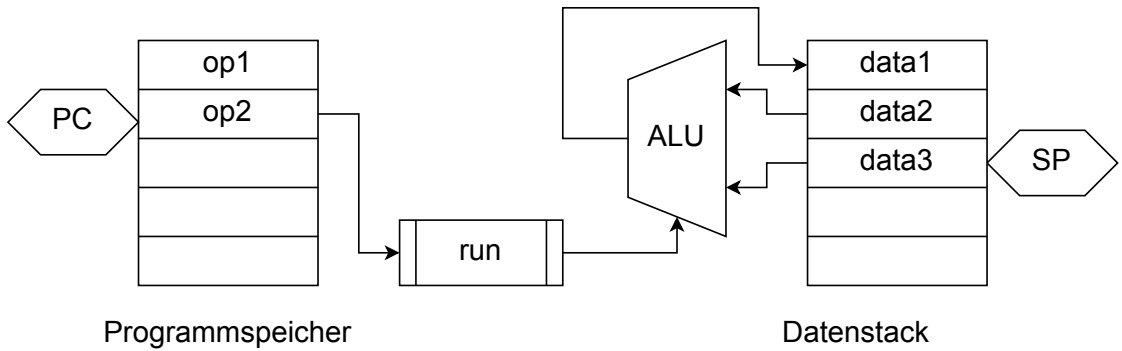


Abb. 1. Prinzipielle Architektur einer Stackmaschine mit einem Programmspeicher, Programmzähler PC und Stackzeiger SP

Stackmaschine

Die *Continuum* VM ist ein Beispiel für eine Stackmaschine.

> Continuum ist eine virtuelle JavaScript-Maschine, die in JavaScript implementiert ist.

- Sie setzt Bytecode aus Quellcode zusammen und führt ihn in einer ES6-Laufzeitumgebung aus.
- Ein Assembler wird verwendet, um den AST in Bytecode und statische Skriptinformationen zu konvertieren.
- Ein **Codeobjekt** enthält den kompilierten Bytecode eines Skripts.
  - Es enthält auch die meisten statischen Semantiken, die von der Quelle abgeleitet wurden, z. B. gebundene Namen, Deklarationen, Importe / Exporte usw.

Vorteil: Codeobjekte können eine portable Form serialisierbar sein, so dass Code einmal kompiliert und dann in dieser Form weiterverteilt und ausgeführt werden kann.

Stackmaschine

- Ein **Skriptobjekt** enthält alle statischen Informationen zu einem Codeabschnitt: die angegebenen Optionen, die Originalquelle, den AST, den Bytecode und den kompilierten Thunk (Operationssatz, ISA), der den Bytecode ausführt.
- Skripte enthalten keine Laufzeitinformationen, sodass sie zwischen Realms portierbar sind und mehrmals ausgeführt werden können.

Das besondere von Continuum ist die enge Kopplung von Bytecode Instruktionen (die innerhalbe der VM ausgeführt werden) mit externen JavaScript Funktionen und Daten (z.B. Ein- und Ausgabe).

Stackmaschine

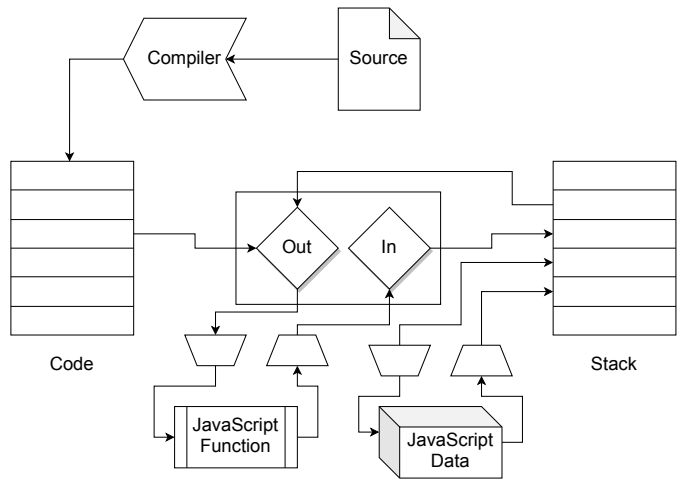


Abb. 2. Prinzipielle Architekturder Continuum VM

Stackmaschine

Befehlssatz

```
ADD, AND, ARRAY, ARG, ARGS, ARGUMENTS, ARRAY_DONE, BINARY, BINDING,
CALL, CASE, CLASS_DECL, CLASS_EXPR, COMPLETE, CONST, CONSTRUCT, DEBUGGER,
DEFAULT, DEFINE, DUP, ELEMENT, ENUM, EXTENSIBLE, EVAL, FLIP, FUNCTION,
GET, GET_GLOBAL, HAS_BINDING, HAS_GLOBAL, INC, INDEX, INTERNAL_MEMBER,
ITERATE, JUMP, JEQ_NULL, JEQ_UNDEFINED, JFALSE, JLT, JLTE, JGT, JGTE, JNEQ_NULL,
JNEQ_UNDEFINED, JTRUE, LET, LITERAL, LOG, LOOP, MEMBER, METHOD, MOVE, NATIVE_CALL,
NATIVE_REF, OBJECT, OR, POP, POPN, PROPERTY, PROTO, PUT, PUT_GLOBAL, REF, REFSYMBOL,
REGEXP, REST, RETURN, ROTATE, SAVE, SCOPE_CLONE, SCOPE_POP, SCOPE_PUSH, SPREAD,
SPREAD_ARG,    SPREAD_ARRAY, STRING, SUPER_ELEMENT, SUPER_MEMBER, SWAP, SYMBOL,
TEMPLATE, THIS, THROW, TO_OBJECT, UNARY, UNDEFINED, UPDATE, VAR, WITH, YIELD
```

Def. 6. Der ISA Befehlsatz von Continuum: Doch schon umfangreicher als bei rein formalen Modellen und Maschinen!

Stackmaschine

```
ADD(context)
---------------------------------------------------------------------------
context.stack[context.sp - 1] += context.ops[context.ip][0]

DUP(context)
---------------------------------------------------------------------------
context.stack[context.sp] = context.stack[context.sp++ - 1]

LITERAL(context)
---------------------------------------------------------------------------
context.stack[context.sp++] = context.ops[context.ip][0]

JUMP(context)
---------------------------------------------------------------------------
context.ip = context.ops[context.ip][0]

JTRUE(context)
---------------------------------------------------------------------------
isTrue(context.stack[--context.sp])? context.ip = context.ops[context.ip][0]
```

Def. 7. Wirkung von einzelnen Bytecode Befehlen

Abstrakte Maschine

TODO

Markov Algorithmen

TODO

Church’sche These

TODO

Endliche Automaten

TODO

AuD DP 17.4.1/17.4.2