Theoretische Informatik (WS04)

Prof. Dr. Raimund Seidel
Wei Ding, Dierk Johannes
Lösungsvorschlag zu Aufgabenblatt 8

Aufgabe 1

Der Vergleich mit Null und alle Befehle mit nur einem Operandenregister stehen bei einer gewöhnlichen Registermaschinen direkt zur Verfügung. Wir brauchen uns also nur mit Befehlen zu beschäftigen, die zwei Operandenregister verwenden. Deren Ausführung simulieren wir unter Verwendung zusätzlicher Register. Im einzelnen:

Wir simulieren eine *komfortable* k-Registermaschine durch eine *gewöhnliche* $k+2$-Registermaschine wie folgt:

Eine Zeile $\texttt{ADD}_{i=j+k}$`: do` x_i`=`x_j`+`x_k `goto ...` wird durch ein Programmfragment der Art

$\vdots$

$\texttt{ADD}_{i=j+k}$`: do` x_i `=` x_j `+ 0 goto` $\texttt{L0}_{i=j+k}$
$\texttt{L0}_{i=j+k}$ `: do` x_{n+1} `=` x_k `goto` $\texttt{L1}_{i=j+k}$
$\texttt{L1}_{i=j+k}$ `: if` x_k `= 0 then` $\texttt{END}_{i=j+k}$ `else` $\texttt{L2}_{i=j+k}$
$\texttt{L2}_{i=j+k}$ `: do` x_k `=` x_k `- 1 goto` $\texttt{L3}_{i=j+k}$
$\texttt{L3}_{i=j+k}$ `: do` x_i `=` x_i `+ 1 goto` $\texttt{L1}_{i=j+k}$
$\texttt{END}_{i=j+k}$`: do` x_k `=` x_{n+1} `goto ...`

ersetzt, und entsprechend x_i`=`x_j`-`x_k durch

$\vdots$

$\texttt{SUB}_{i=j-k}$`: do` x_i `=` x_j `+ 0 goto` $\texttt{L0}_{i=j-k}$
$\texttt{L0}_{i=j-k}$ `: do` x_{n+1} `=` x_k `goto` $\texttt{L1}_{i=j-k}$
$\texttt{L1}_{i=j-k}$ `: if` x_k `= 0 then` $\texttt{END}_{i=j-k}$ `else` $\texttt{L2}_{i=j-k}$
$\texttt{L2}_{i=j-k}$ `: do` x_k `=` x_k `- 1 goto` $\texttt{L3}_{i=j-k}$
$\texttt{L3}_{i=j-k}$ `: do` x_i `=` x_i `- 1 goto` $\texttt{L1}_{i=j-k}$
$\texttt{END}_{i=j-k}$`: do` x_k `=` x_{n+1} `goto ....`

Hier wird zuerst das eine Eingaberegister x_j in das Ergebnisregister geschrieben (x_i), der Inhalt des zweiten Eingaberegisters x_k in einem zusätzlichen Register x_{n+1} zwischengespeichert, und schließlich durch „gleichzeitiges“ Inkrementieren (bzw. Dekrementieren) von x_i und Dekrementieren von x_k die Summe (bzw. Differenz) gebildet. Danach wird noch der ursprüngliche Inhalt in x_k zurückgeschrieben.

Wir sehen, daß Addition und Subtraktion sehr ähnlich simuliert werden. Ebenso verhält es sich mit Multiplikation und Division. Daher beschränken wir uns auf die Division.

x_i`=`x_j `div` x_k wird durch

$\vdots$

$\texttt{DIV}_{i=j/k}$`: do` x_i `= 0 goto` $\texttt{L0}_{i=j/k}$
$\texttt{L0}_{i=j/k}$ `: do` x_{n+1} `=` x_j `+ 0 goto` $\texttt{L2}_{i=j/k}$
$\texttt{L1}_{i=j/k}$ `: if` x_j `= 0 then` $\texttt{END}_{i=j/k}$ `else` $\texttt{L2}_{i=j/k}$
$\texttt{L2}_{i=j/k}$ `: do` x_{n+2} `=` x_k `-` x_j `goto` $\texttt{L3}_{i=j/k}$

$\texttt{L3}_{i=j/k}$: if x_{n+2} = 0 then $\texttt{L4}_{i=j/k}$ else $\texttt{END1}_{i=j/k}$
$\texttt{L4}_{i=j/k}$: do x_j = x_j - x_k goto $\texttt{L5}_{i=j/k}$
$\texttt{L5}_{i=j/k}$: do x_i = x_i + 1 goto $\texttt{L2}_{i=j/k}$
$\texttt{END}_{i=j/k}$: do x_j = x_{n+1} + 0 goto ...

ersetzt. Hier wird wie oben der Wert von x_j nach x_{n+1} „gerettet“, um ihn am Ende wiederherzustellen, und in Register x_{n+2} wird in jedem Schleifendurchlauf auf $x_j < x_k$ geprüft (genau dann ist nämlich $x_k - x_j > 0$). Ist dies der Fall, so wird x_j um x_k erniedrigt und das Ergebnis hochgezählt. Der Sonderfall $x_j = 0$ wird gleich zu Beginn abgefangen, in diesem Fall ist das Ergebnis $= 0$

Im wesentlichen genauso wird aus x_i=x_j mod x_k
⋮
$\texttt{MOD}_{i=j \bmod k}$: do x_{n+1} = x_j + 0 goto $\texttt{L0}_{i=j \bmod k}$
$\texttt{L0}_{i=j \bmod k}$: do x_{n+2} = x_k - x_j goto $\texttt{L1}_{i=j \bmod k}$
$\texttt{L1}_{i=j \bmod k}$: if x_{n+2} = 0 then $\texttt{L2}_{i=j \bmod k}$ else $\texttt{END1}_{i=j \bmod k}$
$\texttt{L2}_{i=j \bmod k}$: do x_j = x_j - x_k goto $\texttt{L0}_{i=j \bmod k}$
$\texttt{END1}_{i=j \bmod k}$: do x_i = x_j + 0 goto $\texttt{END2}_{i=j \bmod k}$
$\texttt{END2}_{i=j \bmod k}$: do x_j = x_{n+1} goto ...

Aufgabe 2

Seien S1, S2, ... die Register der Registermaschine mit indirekter Adressierung und R1, ..., R6 die Register einer komfortablen 6-Registermaschine.

Codiere die Registerinhalte von S1, S2, ... wie in der Vorlesung mit Hilfe von Primzahlen. Wir betrachten die Menge der Primzahlen $P = \{p_1, p_2, \ldots\}$ und verwenden für den Inhalt von Register S_i die Primzahl p_i.

- $\texttt{S}_i = x \in \mathbb{N} \Rightarrow p_i^x$ teilt R1
- p_j teilt R1 nicht $\Rightarrow \texttt{S}_j = 0$

Also: $\texttt{R1} = \prod_{p_i \in P} p_i^{x_i}$, wobei x_j der Inhalt von Register j ist und p_j die j-te Primzahl ist. Dies ist möglich, da zu jedem Zeitpunkt nur endlich viele Register $\neq 0$ sind.

Nun müssen wir die einzelnen Befehle der Registermaschine mit indirekter Adressierung simulieren. Der Übersicht halber Zerlegen wir das Problem in kleinere Stücke: Für goto ?? muss jeweil ein entsprechendes Label des aufrufenden Programms eingesetzt werden.

Theoretische Informatik (WS04)

Prof. Dr. Raimund Seidel
Wei Ding, Dierk Johannes
Lösungsvorschlag zu Aufgabenblatt 8

Folgende Programmsequenz `IS_PRIME` berechnet `R3` $= \begin{cases} 1 & \text{falls } \texttt{R2} \in P \\ 0 & \text{falls } \texttt{R2} \notin P \end{cases}$. Dabei wird Register `R3` beginnend bei 2 hochgezählt, bis es `R2` teilt. Ist dann `R2 = R3`, so ist `R2` eine Primzahl.

```
          R3 = 2
TOP       R4 = R2 mod R3
          if (R4 = 0) then TEILT else NEXT
NEXT      R3 = R3+1
          goto TOP
TEILT     R4 = R2-R3
          if (R4 = 0) then JA else NEIN
JA        R3 = 1
          goto ??
NEIN      R3 = 0
          goto ??
```

Folgende Programmsequenz (`NEXT_PRIME`) berechnet
`R2` $= \min\{p \in P \mid p >$ `R2`$\}$ also die nächst größere Primzahl. Dabei wird `R2` so lange erhöht, bis es eine Primzahl ist.

```
START     R2 = R2+1
          IS_PRIME
          if (R3 = 0) then START else OK
OK        goto ??
```

Folgende Programmsequenz (`LOG`) berechnet `R3` $=$ `S`$_i$, wobei `R2` $= p_i$ ist. Dazu zählen wir, wie oft wir `R1` durch p_i teilen können. Wir berechnen also den Exponenten von p_i in $\prod_{p_i \in P} p_i^{x_i}$, dies erklärt die Namenswahl.

```
          R5 = R1
          R3 = 0
MOD       R4 = R5 mod R2
          if (R4 = 0) then DIV else END
DIV       R3 = R3+1
          R5 = R5 div R2
          goto MOD
END       goto ??
```

In der Vorlesung haben wir gesehen, daß wir alle angegebenen Operationen mit Inkrement- und Dekrement-Operationen simulieren können. Daher können wir uns auf diese beschränken.

Der Befehl `incr(x[i])` kann damit folgendermaßen simuliert werden: Wir suchen die i-te Primzahl und multiplizieren sie zu R1.


```
          R2 = 1
          R6 = i
N         NEXT_PRIME
          R6 = R6-1
          if (R6 = 0) then DONE else N
DONE      R1 = R1*R2
```

Der Befehl `decr(x[i])` wird folgendermaßen simuliert: Wir suchen die i-te Primzahl und teilen `R1` durch diese, wenn dies ohne Rest geht.

```
          R2 = 1
          R6 = i
N         NEXT_PRIME
          R6 = R6-1
          if (R6 = 0) then CHECK else N
CHECK     R6 = R1 mod R2
          if (R6 = 0) then DIVID else DONE
DIVID     R1 = R1 div R2
DONE
```

Der Befehl `incr(x[x[i]])` wird folgendermaßen simuliert: Wir suchen die i-te Primzahl und schreiben dann in `R6` den Inhalt von S_i (mit `LOG`). Danach suchen wir die `x[i]`-te Primzahl und multiplizieren sie zu `R1`.

```
          R2 = 1
          R6 = i
N1        NEXT_PRIME
          R6 = R6-1
          if (R6 = 0) then GO else N1
GO        LOG
          R2 = 1
          R6 = R3
N2        NEXT_PRIME
          R6 = R6-1
          if (R6 = 0) then DONE else N
DONE      R1 = R1*R2
```

Der Befehl `decr(x[x[i]])` wird folgendermaßen simuliert:

```
          R2 = 1
          R6 = i
N1        NEXT_PRIME
          R6 = R6-1
```

```
          if (R6 = 0) then GO else N1
GO        LOG
          R2 = 1
          R6 = R3
N2        NEXT_PRIME
          R6 = R6-1
          if (R6 = 0) then CHECK else N2
CHECK     R6 = R1 mod R2
          if (R6 = 0) then DIVID else DONE
DIVID     R1 = R1 div R2
DONE
```

Um $\mathtt{S}_i$ auf Null zu testen, müssen wir überprüfen, ob sich `R1` durch p_i teilen lässt:

```
          R2 = 1
          R3 = i
N         NEXT_PRIME
          R3 = R3-1
          if (R3 = 0) then MOD else N
MOD       R3 = R1 mod R3
          if (R3 = 0) then NEIN else JA
JA        R3 = 0
          goto ??
NEIN      R3 = 1
          goto ??
```

Um $\mathtt{S}_{S_i}$ auf Null zu testen, müssen wir zunächst den Index $j = \mathtt{S}_i$ berechnen und dann überprüfen, ob sich `R1` durch p_j teilen lässt:

```
          R2 = 1
          R3 = i
N1        NEXT_PRIME
          R3 = R3-1
          if (R3 = 0) then GO else N1
GO        LOG
          R3 = i
N         NEXT_PRIME
          R3 = R3-1
          if (R3 = 0) then MOD else N
MOD       R3 = R1 mod R3
          if (R3 = 0) then NEIN else JA
JA        R3 = 0
          goto ??
```

```
NEIN      R3 = 1
          goto ??
```

Ganz ausgeschrieben sähe der Befehl `decr(x[x[i]])` also folgendermaßen aus:

```
          R2 = 1
          R6 = i
N1        R2 = R2+1
          R3 = 2
TOP1      R4 = R2 mod R3
          if (R4 = 0) then TEILT1 else NEXT1
NEXT1     R3 = R3+1
          goto TOP1
TEILT1    R4 = R2-R3
          if (R4 = 0) then JA1 else NEIN1
JA1       R3 = 1
          goto W1
NEIN1     R3 = 0
W1        if (R3 = 0) then N1 else OK1
OK1       R6 = R6-1
          if (R6 = 0) then GO else N1
GO1       R5 = R1
          R3 = 0
MOD       R4 = R5 mod R2
          if (R4 = 0) then DIV else END
DIV       R3 = R3+1
          R5 = R5 div R2
          goto MOD
END       goto R2 = 1
          R6 = R3
N2        R2 = R2+1
          R3 = 2
TOP2      R4 = R2 mod R3
          if (R4 = 0) then TEILT2 else NEXT2
NEXT2     R3 = R3+1
          goto TOP2
TEILT2    R4 = R2-R3
          if (R4 = 0) then JA2 else NEIN2
JA2       R3 = 1
          goto W2
NEIN2     R3 = 0
W2        if (R3 = 0) then N2 else OK2
OK2       R6 = R6-1
```

```
          if (R6 = 0) then CHECK else N2
CHECK     R6 = R1 mod R2
          if (R6 = 0) then DIVID else DONE
DIVID     R1 = R1 div R2
DONE
```

Zusatzaufgabe

Wir berechnen die in der Rekursion benötigten Werte sukzessive. Die (zweidimensionale) Tabelle speichern wir in linearer Reihenfolge. Dieser Lösungsvorschlag speichert sie unter Verwendung von Primzahlen in nur einem Register. Dies ist aber nicht nötig. Es können beispielsweise die Register $x_{11}, x_{12}, \ldots$ dazu verwendet werden und mittels indirekter Adressierung auf sie zugegriffen werden. In jedem Schleifendurchlauf wird *ein* neuer Wert der Ackermann-Funktion berechnet. Beachte, daß jeder Schleifendurchlauf mit den Eingabeargumenten (i, n) startet.

START
// In x_0 und x_1 speichere die Eingaben i und n
$x_0 = i$;
$x_1 = n$;
// In x_2 und x_3 speichere die aktuellen Argumente i_t und n_t
3 $x_2 = i$;
$x_3 = n$;
// Sei $B_{(i+n)(i+n+1)/2+n} = A(i,n)$. Berechne in x_4 den Suffix von B
5 $x_4 = x_2 + x_3$;
$x_4 = x_4 \times (x_4 + 1)/2 + x_3$;
// In x_5 speichere $\prod_{j=0}^{\infty} P_i^{B_j}$, dabei ist P_j die $j-te$ Primzahl.
7 **if** $(x_2 = 0\ AND\ x_3 = 1)$ **then** $x_5 = x_5 \times P_{x_4}^2$
else if$(x_2 = 0)$ **then** $x_5 = x_5 \times P_{x_4}^{x_3+2}$
else if$(x_3 = 0)$ **then** $x_5 = x_5 \times P_{x_4}$
else
{
// Entscheiden, ob $A(i_t, n_t - 1)$ schon zur Verfügung steht
$x_6 = (x_2 + x_3)(x_2 + x_3 - 1)/2 + x_t - 1$;
if $(x_5\ mod\ P_{x_6}! = 0)$ //$A(i_t, n_t - 1)$ nicht von Hand
{
// Wechseln zur Berechnung $A(i_t, n_t - 1)$
$x_3 = x_3 - 1$;
goto 5;
}
else // $A(i_t, n_t - 1)$ von Hand
{
// Berechne $A(i_t, n_t - 1)$, speiche in x_8

```
8                 if (x5 mod P_{x6}! = 0)
                  then
                  {
                    x7 = x5 div P_{x6};
                    x8 = x8 + 1;
                    goto 8
                  } // Speichere A(i_t, n_t − 1) in x8
                  // Entscheiden, ob A(i_t − 1, A(i_t, n_t − 1)) schon zur Verfügung steht
                  else
                  {
                    x9 = (x2 − 1 + x8) × (x2 + x8)/2 + x8;
                    if (x5 mod x9 == 0) then //A(i_t − 1, A(i_t, n_t − 1)) schon von Hand
                    {
                      // Berechne A(i_t − 1, A(i_t, n_t − 1)), speiche in x10
9                     if(x5 mod x9 == 0)
                      then
                      {
                            x5 = x5 div x9;
                            x10 = x10 + 1;
                            goto 9
                      } // Speichere A(i_t − 1, A(i_t, n_t − 1)) in x10
                    // Speiche A(i_t, n_t)
10                  if(x10! = 0)
                    {
                      x5 = x5 × P_{x6};
                      x10 = x10 − 1
                      goto 10;
                    }
                    }
                    else
                    {
                      x2 = x2 − 1;
                      x3 = x8;
                      goto 5;
                    }
                  }
                }
              }
        if(x2! = x0 OR x3! = x1) then goto 3 else return x5;
END
```