Freie Universität Berlin

Technische Informatik II
Rechnerarchitektur

3.Unterprogramme in MMIX

Matthias Dräger

E-Mail: mdraeger@mi.fu-berlin.de
www: www.matthias-draeger.info/lehre/sose2010ti2/
tinyurl.com/sose2010ti2

Zuletzt bearbeitet: 19.05.2010

Vorteile von Unterprogrammen

- Strukturierung des Programms
  - komplexe Programme werden in verständliche Teilprogramme untergliedert
- Wiederverwendung
  - Unterprogramme zur Lösung eines Problems können leicht in andere Programme integriert werden
- Verteilte Entwicklung im Team
  - Unterprogramme können unabhängig voneinander entwickelt werden
- Separate Übersetzung (z.B. in Java)
  - Programmteile werden separat übersetzt und später *gebunden*, d.h., bei einer kleinen Änderung muss nicht immer das komplette Programm übersetzt werden
- Verwendung von Programmbibliotheken (z.B. in C, Python)
  - Bibliotheken sind Sammlungen von Unterprogrammen zu einen bestimmten Anwendungsgebiet

Bedingungen von Unterprogrammen

Ein Unterprogramm sollte:

- nur eine Aufgabe lösen
- einen Mittelweg zwischen Einfachheit und Allgemeinheit darstellen
- Daten als Parameter erhalten
- globale Variablen unverändert lassen
- das Ergebnis als Rückgabewert zurückliefern

Überblick: Unterprogramme in MMIX

- beim Aufruf eines Unterprogramms steht die Rücksprungadresse* im Spezialregister rJ
- einem Unterprogramm können mehrere Parameter übergeben werden
- es können mehrere Ergebnisse zurückgeliefert werden (Besonderheit!)
- bei Rekursion oder ineinander verschachtelten Aufrufen, muss man die Rücksprungadresse zwischenspeichern

* ist die Adresse des Folgebefehls nach dem Unterprogrammaufruf

Unterprogramme mit dem Befehl GO

Name: GO (Get Address)

Spez.: GO $X,*Marke*
GO $X,$Y,$Z

Beschreibung:
Es wird die Adresse des nachfolgenden Befehls (Rücksprungadresse) in Register $X gespeichert. Anschließend wird zur Stelle *Marke* bzw. zur Adresse $Y+$Z gesprungen. Es wird absolut adressiert.

```
% unterprogramm_go.mms
        LOC  Data_Segment
        GREG @  % wird für LDA benötigt
String  BYTE "Willkommen im Unterprogramm :)",0

        LOC #100
        GREG @  % wird für GO benötigt

% Unterprogramm anlegen
ausgabe LDA  $255,String        % Adresse von String ins globale Register laden
        TRAP 0,Fputs,StdOut     % Srting ausgeben
        GO   $0,$0,0            % Rücksprung zum Hauptprogramm
                                % in $0 steht die Rücksprungadresse

% Hauptprogramm
Main    GO   $0,ausgabe         % Rücksprungadresse wird in $0 gespeichert und nach ausgabe springen
        TRAP 0,Halt,0
```

Nachteile von GO

- Parameter lassen sich nicht übergeben
- Register im Unterprogramm werden auch im aufzurufenden Programm überschrieben
- Rekursion ist dadurch nicht möglich

Lösung 1:

- Stack implemetieren und mithilfe eines Stackpointers verwalten

→ sehr kompliziert und aufwändig!

Lösung 2:

- die Befehle `PUSHJ` (push & jump) und `POP` verwalten den Stack automatisch

Die Befehle PUSHJ, PUSHGO und POP

Name: PUSHJ (Push & Jump)

Spez.: PUSHJ $X,*Marke*

Beschreibung:
Die Register $0 bis $X werden auf dem Registerstack gesichert. Die Register ab $X (also $X+1, $X+2, …) stellen die Parameter dar und werden umnummeriert, beginnend bei $0, $1, …
Register $X wird von MMIX intern benutzt und enthält die Anzahl der gesicherten Register.
Die Rücksprungadresse wird im Spezialregister rJ gespeichert und es wird zur *Marke* gesprungen.

Name: PUSHGO (Push & Go)

Spez.: PUSHGO $X,$Y,$Z

Beschreibung:
siehe PUSHJ und GO

Name: POP

Spez.: POP X,Y

Beschreibung:
Der Befehl POP macht die von der PUSHJ ausgeführten Aktionen rückgängig.

Die Zahl X gibt an, wie viele Rückgabewerte (also Ergebnisse) es gibt, d.h. sie muss im Bereich 0 bis 255 liegen. Die Rückgabewerte müssen dabei in den Registern $0 bis $X-1 stehen.
Anschließend wird die Umnummerierung von PUSHJ rückgängig gemacht. Die Ergebnisse aus dem Unterprogramm ($0 bis $X-1) stehen nun an der Stelle $X (von PUSHJ $X,*Marke*).

Es folgt ein Rücksprung an die Adresse, die im Spezialregister rJ hinterlegt wurde. Ist der Parameter Y von POP größer als 0, so folgt der Rücksprung an die Adresse:

$$\text{Rücksprungadresse} = \text{rJ} + 4 \cdot \text{Y}$$

Beispiel 1: PUSHJ und POP

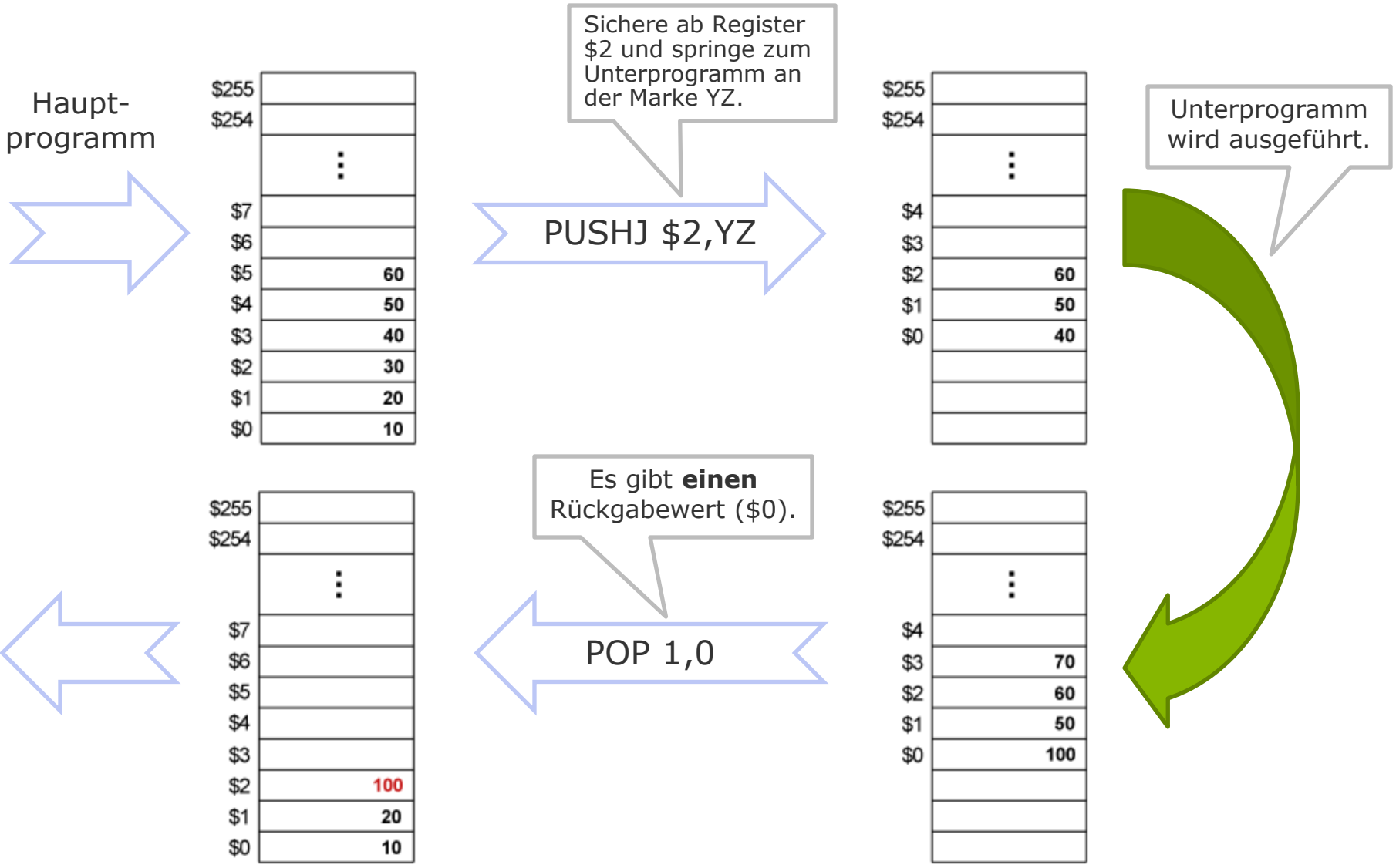
Ziel: Wir wollen das Unterprogramm YZ aufrufen und dessen Ergebnis soll danach in Register $2 stehen.

```
% bsp1-pushj.mms
        LOC #100

% Unterprogramm anlegen
YZ      SET  $3,70
        ADD  $0,$0,60           % erhöhe $0 um 60
        POP  1,0                % liefere $0 als Ergebnis zurück

% Hauptprogramm
Main    SET   $0,10             % Register mit Beispielwerten füllen
        SET   $1,20
        SET   $2,30
        SET   $3,40
        SET   $4,50
        SET   $5,60
        PUSHJ $2,YZ             % Zum Unterprogramm YZ springen
        SET   $0,$2             % Welcher Wert steht nun in $0?
        TRAP 0,Halt,0
```

Schematische Darstellung zu Beispiel 1



Beispiel 2: PUSHJ und POP

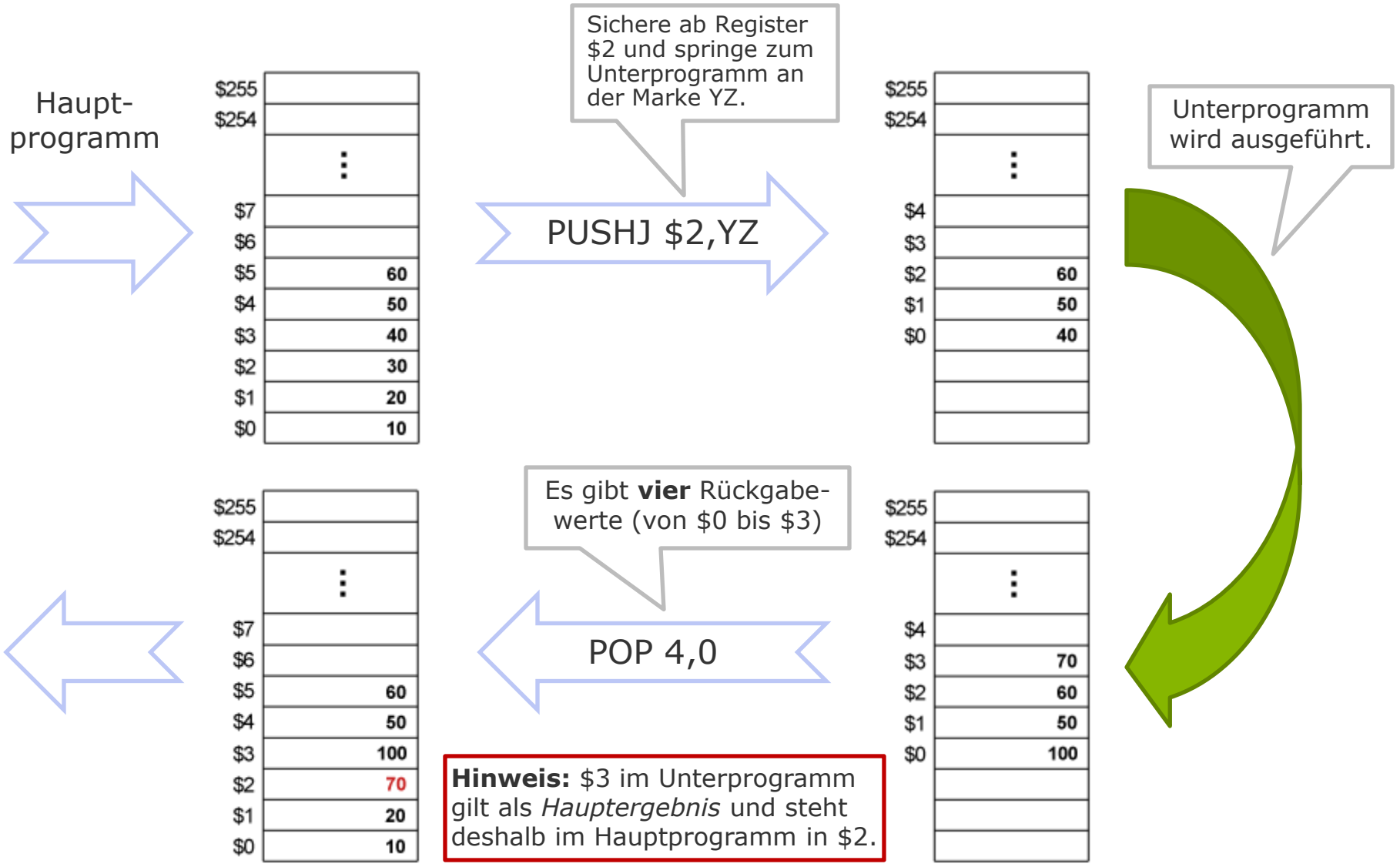
Ziel: Wir wollen das Unterprogramm YZ aufrufen und es sollen vier Ergebnisse zurück geliefert werden.

```
% bsp2-pushj.mms
        LOC #100

% Unterprogramm anlegen
YZ      SET  $3,70
        ADD  $0,$0,60           % erhöhe $0 um 60
        POP  4,0                % liefere 4 Ergebnisse zurück ($0 - $3)
                                % das Hauptergebnis ist also $3

% Hauptprogramm
Main    SET   $0,10             % Register mit Beispielwerten füllen
        SET   $1,20
        SET   $2,30
        SET   $3,40
        SET   $4,50
        SET   $5,60
        PUSHJ $2,YZ             % Zum Unterprogramm YZ springen
                                % Das Hauptergebnis steht in $2.
        SET   $0,$2             % Welcher Wert steht nun in $0?
        TRAP 0,Halt,0
```

Schematische Darstellung zu Beispiel 2



Wiederholung: Organisation des Hauptspeichers

Wo werden die Daten von Registern bei Unterprogrammausrufen in MMIX zwischengespeichert/ausgelagert?

Segment	Symbol	Startadresse	Zweck
Textsegment		#0 bis #FF	Interruptroutinen
Textsegment		#100	Programm und statische Daten
Datensegment	Data_Segment	#2000 0000 0000 0000	veränderliche Daten (z.B. Arrays, Texte, …)
Poolsegment	Pool_Segment	#4000 0000 0000 0000	Kommunikation zwischen Programm und Betriebssystem
Stacksegment	Stack_Segment	#6000 0000 0000 0000	Auslagerung der Inhalte von Registern bei Unterprogrammaufrufen

Beispiele mit PUSHJ und POP

Wiederholung: Summenberechnung

In dem folgenden Beispiel wird die Summe berechnet: $\text{res} = \sum_{i=1}^{n} i$

```
% sum-while.mms
% Es wird die Summe von 1..n berechnet.

        LOC #100

n       IS  $0
i       IS  $1              % Laufvariable
res     IS  $2              % Ergebnis speichern
tmp     IS  $3              % speichert Ergebnis von CMP

Main    SET n,6             % setze n auf Endwert 6
        SET i,1             % initialisiere Zählvariable
        SET res,0           % Ergebnis mit 0 initialisieren

loop    ADD res,res,i       % rechne: res = res + i
        ADD i,i,1           % erhöhe i um 1
        CMP tmp,i,n         % vergleiche i mit n
        % wenn i > n, dann hat tmp den Wert 1
        % Hinweis: tmp kann wegen CMP nur den Wert -1,0 oder 1 haben
        % prüfen, ob tmp nicht positiv ist -> zum Schleifenanfang springen
        BNP tmp,loop
        TRAP 0,Halt,0       % beende Programm
```

Summenberechnung im Unterprogramm

Ziel: Berechnung der Summe von 1 bis n im Unterprogramm.

```
% sum-pushj-while.mms
        LOC #100
%
%  Dieses Unterprogramm berechnet die Summe der Zahlen 1..sum_n
%
sum_n   IS  $0
i       IS  $1                  % Laufvariable
res     IS  $2                  % Ergebnis speichern
tmp     IS  $3                  % speichert Ergebnis von CMP

sum     SET i,1                 % initialisiere Zählvariable
        SET res,0               % Ergebnis mit 0 initialisieren
loop    ADD res,res,i           % rechne: res = res + i
        ADD i,i,1               % erhöhe i um 1
        CMP tmp,i,sum_n         % vergleiche i mit sum_n
        BNP tmp,loop            % springe solange zu loop, wie i <= sum_n
        SET $0,res              % speichere das Ergebnis in $0
        POP 1,0                 % gib Wert von $0 zurück

% Das Hauptprogramm
n       IS  $1
Main    SET n,6                 % setze n auf Endwert 6
        PUSHJ $0,sum            % springe zum Unterprogramm sum und übergebe $1
        % hier könnte noch eine Ausgabe von $0 folgen...
        TRAP 0,Halt,0           % beende Programm
```

Rekursive Summenberechnung im Unterprogramm

Ziel: Rekursive Berechnung der Summe von 1 bis n im Unterprogramm.

```
% sum-pushj-rek.mms
        LOC #100
%
%  Dieses Unterprogramm berechnet rekursiv die Summe der Zahlen 1..sum_n
%
sum_n   IS  $0                  % Laufvariable n wird mit jedem Rekursionsschritt verringert
adr_rJ  IS  $1                  % zum Zwischenspeichern der Rücksprungadresse
res     IS  $2                  % Ergebnis vom Rekursionsschritt
tmp     IS  $3                  % speichert Ergebnis von CMP

sum     SUB tmp,sum_n,1         % (n-1) für den nächsten Rekursionsschritt vorbereiten
        BNP tmp,sum_end         % falls tmp < 1, beende Rekursion
        GET adr_rJ,rJ           % Adresse der Rücksprungadresse sichern (WICHTIG!!)
        PUSHJ res,sum           % Unterprogramm sum rekursiv aufrufen
        ADD sum_n,sum_n,res     % ergebnis = n + res
        PUT rJ,adr_rJ           % Rücksprungadresse zurückspeichern
sum_end POP 1,0                 % Wert von $0 zurückliefern

% Das Hauptprogramm
n       IS  $1
Main    SET n,6                 % setze n auf Endwert 6
        PUSHJ $0,sum            % springe zum Unterprogramm sum und übergebe $1
        % hier könnte noch eine Ausgabe von $0 folgen...
        TRAP 0,Halt,0           % beende Programm
```

Zusammenfassung

- Was sind Unterprogramme?
- Vorteile von Unterprogrammen
- Unterprogramme in MMIX
  - mittels GO
  - mit den Befehlen PUSHJ und POP
  - rekursive Programmierung