Assembler-2000 Befehlsreferenz

ADD op1,op2
Addiert op1 mit op2 und speichert das Ergebnis in op1

CALL op1
Ruft die Funktion mit dem Namen op1 auf

CMP op1,op2
Vergleicht op1 mit op2(siehe Kapitel „Sprungbefehle“)

DEC op1
Verringert op1 um 1

DIV op1
Register AX wird durch op1 geteilt und speichert das Ergebnis in AX

FUNC op1
Erstellt eine Funktion mit dem Namen op1(siehe Kapitel „Funktionen“)

INC op1
Erhöht op1 um 1

INT op1
Führt Interrupt aus(siehe Kapitel „Interrupts“)

JA op1
Springt zu op1, wenn op1 von CMP größer als op2 war

JAE op1
Springt zu op1, wenn op1 von CMP größer oder gleich op2 war

JB op1
Springt zu op1, wenn op1 von CMP kleiner als op2 war

JBE op1
Springt zu op1, wenn op1 von CMP kleiner oder gleich op2 war

JE op1

Springt zu op1, wenn op1 von CMP gleich op2 war

JNE op1
Springt zu op1, wenn op1 von CMP ungleich op2 war

JMP op1
Springt zu op1

LABEL op1
Erstellt ein Label mit dem Namen op1, zu welchem via Sprungbefehl(JMP,JA,JNE …) gesprungen wird

MOV op1,op2
Kopiert den Wert von op2 und speichert ihn in op1 ab

MUL op1
Register AX wird mit op1 multipliziert und speichert das Ergebnis in AX

NEG op1
Dreht das Vorzeichen von op1 um

RET
Stellt das Ende einer Funktion da

SUB op1,op2
Subtrahiert op2 von op1 und speichert das Ergebnis in op1

Funktionen

Eine Funktion wird via FUNC erstellt. Sie dienen dazu, oft benutzten Code durch nur einen Befehl(CALL) aufrufen zu können. Somit spart man sich viel Schreibarbeit. Beispiel:

```
.data
        1
        hello         Hello World

.code
        mov ax,%hello
        call print          ;Funktion aufrufen
```

```
func print              ;Funktion erstellen
        mov ah,02h
        int 42h
        ret             ;Ende der Funktion
```

Der Befehl RET bildet das Ende der Funktionsdefinierung.

Sprungbefehle

Man unterscheidet in Assembler zwischen bedingten und unbedingten Sprüngen:

Unbedingte Sprünge:

Unbedingte Sprünge sind einfach. Es wird in einer beliebigen Zeile ein Label erstellt und via JMP wird die Programmausführung in der aktuellen Zeile abgebrochen und in der Labelzeile fortgesetzt. Beispiel:

```
        mov ax,34
        jmp label       ;Zum Label springen
        mov bx,32       ;Dieser Code wird nicht mehr ausgeführt
label test              ;Hierhin wird gesprungen
        mov cx,45
        mov ah,01h
        int 42h
```

Solche Sprünge sind mit den GOTOs in Hochsprachen zu vergleichen und sind zu vermeiden, wo es nur geht, da sie schnell zu unübersichtlichen Code führen können. Stattdessen wird zu Funktionen geraten.

Bedingte Sprünge:

Über bedingte Sprünge kann man den Status eines Register oder einer Speicherzelle prüfen und nur dann springen, wenn eine Bedingung gültig ist(Wert1 ist größer als Wert2, Wert1 ist gleich Wert2 …). Der Vergleich findet hierbei über CMP statt. Erst nach dem Aufruf von CMP kann ein bedingter Sprung ausgeführt werden. Beispiel:

```
        mov ax,34
        mov bx,45
        cmp ax,bx       ;Register AX und BX vergleichen
```

```
        jb test             ;Zu Label test springen, wenn AX kleiner ist als BX
        mov bx,32           ;Dieser Code wird nicht ausgeführt
label test                  ;Hierhin wird gesprungen
        mov ax,22
        mov ah,01h
        int 42h
```

Interrupts

Interrupts sind BIOS oder Betriebssystemfunktionen, welche man über den Befehl INT nutzen kann. In Assembler-2000 kann man weder auf die Funktionen des BIOS oder des genutzten Betriebssystems zugreifen. Stattdessen werden die Funktionen eines virtuellen BIOS und eines virtuellen Betriebssystems genutzt. Das hat den Vorteil, dass man ein Programm nicht für jedes Betriebssystem umschreiben muss. Die Funktionen des virtuellen BIOS unterscheiden sich nicht von denen eines echten BIOS, allerdings ist der Funktionsumfang eingeschränkt. Ein Interrupt wird ausgeführt, nachdem eine bestimmte Funktion geladen wurde. Dies tut man mit dem Befehl MOV. Danach führt man mit INT entweder ein BIOS-Interrupt(int 10h) oder ein Betriebssysteminterrupt(int 42h) auf. Beispiel:

```
.data
        1
        hello       Hello World

.code
        mov ax,%hello
        mov ah,02h          ;Funktion 02h laden(Zeichenkette schreiben)
        int 42h             ;Betriebssysteminterrupt
        mov ah,03h          ;Funktion 03h laden(Warten, bis AL abgelaufen ist)
        mov al,2000         ;2000 Millisekunden soll gewartet werden
        int 42h             ;Betriebssysteminterrupt
```

Meist wird der Wert eines anderen Registers benötigt, um eine Funktion ordnungsgemäß durchzuführen. Alle BIOS- und Betriebssysteminterrupts können in den Dateien „BIOS-Interrupts.pdf“ und „OS-Interrupts.pdf“ nachgeschlagen werden.

Umgang mit Speicherzellen

Speicherzellen können beliebige Werte gegeben werden, seien es Zahlenwerte oder Zeichenketten. Mit ihnen lässt sich arbeiten, als seien sie Register, allerdings muss vorher im Codesegment „.data“ gesagt werden, in welcher Speicherzelle der Wert liegt. Danach lässt sich im Code damit arbeiten. Beispiel:

```
.data
        1
        ergeb       $900h       ;Definieren der Speicherzelle 900 als „ergeb“

.code
        mov [ergeb],45          ;Arbeiten wie mit Registern
        add [ergeb],34
```

Um mit ihnen zu arbeiten, muss der Speicherzelle einen „Kosenamen“ gegeben werden(hier „ergeb“). Damit der Compiler diese Kosenamen richtig erkennt, dürfen sie nur 5 Zeichen lang sein, nicht mehr und nicht weniger.

Arbeiten mit Zeichenketten

Zeichenketten sind Werte, die Buchstaben oder Sätze enthalten. Sie werden im .data Teil des Codes definiert. Danach lassen sie sich über die Funktion 02h des virtuellen Betriebssystems anzeigen. Beispiel:

```
.data
        1
        hello       Hello World

.code
        mov ax,%hello           ;Beim Verschieben dient ein % als Präfix
        mov ah,02h              ;Zeichenkette ausgeben
        int 42h                 ;Betriebssysteminterrupt
```

Genauso wie bei den Speicherzellen, dürfen die „Kosenamen“ der Zeichenketten nicht mehr oder weniger als 5 Zeichen enthalten.

Compileroptionen

Eine Liste der verfügbaren Optionen lässt sich im Terminal(Linux) oder in der DOS-Eingabeaufforderung(Windows/DOS) mithilfe von asc –help

anzeigen.

Aufruf des Compilers: asc [DATEI] [OPTION]
Beispiel Linux: asc test -d
Beispiel Windows: asc.exe test -d

Folgende Optionen sind gültig:

```
-h, --help      Diese Hilfe anzeigen
-v, --version   Versionsnummer anzeigen
-d              Die erstellte 'output.bas' nicht löschen
-l, --license   Lizenzhinweis lesen
```