Sprungbefehle

Sprungbefehle sind in Assemblersprache von elementarer Bedeutung, weil Verzweigungen und Wiederholungen mit Sprungbefehlen realisiert werden. Man unterscheidet **unbedingte** Sprungbefehle und **bedingte** Sprungbefehle.

Unbedingter Sprungbefehl - JMP

Mit dem Befehl JMP, Jump, Springe, wird ein unbedingter Sprung ausgeführt. Die Syntax ist

```
JMP SPRUNGZIEL
```

Das Sprungziel ist in der Regel eine Marke, die irgendwo im Programm erklärt ist. Eine Sprungmarke wird durch ihren Namen gefolgt von einem Doppelpunkt festgelegt:

```
JMP programmende

...
... ;weiterer Code, der niemals ausgeführt wird
...

programmende:
   mov   eax,1              ;system call number (sys_exit)
   int   0x80               ;call kernel
```

Bedingte Sprungbefehle

Bedingte Sprungbefehle sind von Bedingungen abhängig. Ist die Bedingung wahr, wird der Sprung ausgeführt, ist sie falsch, wird er nicht ausgeführt.

Es gibt viele unterschiedliche bedingte Sprungbefehle, die sich auf unterschiedliche Bedingungen beziehen. Die Bedingung ist im Namen des Befehles angedeutet und bezieht sich entweder direkt auf Flags oder auf einen vorausgegangenen Compare-Befehl (CMP).

Beispiele dafür sind:

- JC (Jump if Carry, Springe wenn Carryflag gesetzt)
- JG (Jump if greater, Springe wenn größer)
- JNE (Jump if not equal, Springe wenn nicht gleich)

Die Namen der bedingten Sprungbefehle (JXXX) sind nach folgendem Schema zusammengesetzt:

- **J** : immer erster Buchstabe, "JUMP"
- **N** : evtl. zweiter Buchstabe, "NOT", steht für die Negierung der Bedingung
- **Z,C,S,O,P** : wenn Zero-/Carry-/Sign-/Overflow-/Parityflag gesetzt
- **E** : Equal, gleich

- **A,B** : Above/Below, größer/kleiner bei vorzeichenloser Arithmetik
- **G,L** : Greater/Less, größer/kleiner bei Arithmetik mit Vorzeichen

Damit kann man eine ganze Menge von bedingten Sprüngen realisieren:

Befehl	Sprungbedingung	Sprungbed. dt.	Flags
	Direkte Abfrage von Flags		
JE|JZ	equal|zero	gleich|Null	ZF=1
JNE|JNZ	not equal|zero	ungleich|ungleich Null	ZF=0
JS	signed	Vorzeichen negativ	SF=1
JNS	not signed	Vorzeichen positiv	SF=0
JP|JPE	parity|parity even	gerade Parität	PF=1
JNP|JPO	no parity|parity odd	ungerade Parität	PF=0
JO	overflow	Überlauf	OF=1
JNO	no overflow	kein Überlauf	OF=0
JC	carry	Übertrag	(CF=1)
JNC	no carry	kein Übertrag	(CF=0)
	Arithmetik mit Vorzeichen		
JL|JNGE	less|not greater or equal	kleiner	CF ≠ OF
JLE|JNG	less or equal|not greater	kleiner oder gleich	(SF ≠ OF) oder (ZF=1)
JNL|JGE	not less|greater or equal	nicht kleiner	(SF=OF)
JG|JNLE	greater|not less or equal	größer	(SF=OF) und (ZF=0)
	Vorzeichenlose Arithmetik		
JA|JNBE	above|not below or equal	oberhalb	(CF=0) und (ZF=0)
JB|JNAE	below|not above or eq.	unterhalb	(CF=1)
JNA|JBE	not above|below or equal	nicht oberhalb	(CF=1) oder (ZF=1)
JNB|JAE	not below|above or equal	nicht unterhalb	(CF=0)

Verzweigungen und (Sprung-)Schleifen

Verzweigungen und Wiederholungsschleifen werden in Assemblersprache durch Sprungbefehle realisiert. Eine Verzweigung mit zwei Zweigen wird grundsätzlich folgendermaßen aufgebaut (die Namen der Marken sind natürlich frei wählbar):

```
cmp Operand1, Operand2
jxxx Wahr-Zweig ; Bedingter Sprungbefehl, überspringt den Falsch-Zweig
  .
  ;Falsch-Zweig, wird ausgeführt, wenn Bedingung falsch
  .
  jmp Verzweigungsende; Wenn falsch Zweig, springe zum Ende der Verzweigung

Wahr-Zweig:
  .
  ;Wahr-Zweig, wird ausgeführt, wenn Bedingung wahr
  .
Verzweigungsende:
```

```
  .
  ; Weiterer Programmablauf
```

Der Wahrzweig kann auch entfallen, dann hat man einen bedingt ausgeführten Befehlsblock.

(Sprung-)Schleifen

Die Abbruchbedingungen der Schleifen können das Erreichen eines bestimmten Zählwertes sein (Zählschleifen) oder datenabhängig formuliert werden. Die Grundkonstruktion einer (nicht abweisenden) Schleife kann folgendermaßen aussehen:

```
Initialisierung der Schleife

Schleifenstart:
   Schleifenrumpf (Befehle)
   Schleifenbedingung aktualisieren (z.B. Zählwert ändern) und auswerten,
   bedingter Sprung zu "Schleifenstart"
```

Schleifen mit Loop Befehlen

Der Loop-Befehl ist ein Spezialbefehl für die Programmierung von Schleifen (engl. Loop = Schleife). Der Loop-Befehl erniedrigt CX bzw. in 32-Bit-Segmenten ECX und springt anschließend zu einem als Operanden angegebenen Sprungziel, falls CX bzw. ECX nicht 0 ist.

Damit lassen sich sehr einfach Zählschleifen programmieren, deren Zählindex in CX/ECX geführt wird und abwärts bis auf 0 läuft.