Thomas Merkel

Zusammenfassung Assembler

Der Assembler

Ein Assembler ist genaugenommen ein Compiler, der den Code eines „Assemblerprogramms“ in Maschinensprache, d. h. Nullen und Einsen übersetzt.

Statt „000001011110100000000011“ schreiben zu müssen, kann der Programmierer die Assembleranweisung:

```
add ax,1000
```

symbolische Bezeichnung	Maschinencode
add ax	00000101
1000 (dez.)	0000001111101000

(Zusätzlich vertauscht der Assembler noch die Reihenfolge der Bytes des Offsets)

```
00000101  11101000  00000011
add ax    low Byte  high Byte
```

Assembler Beispiele

Ein Assembler kann eigentlich sehr wenig, nämlich nur das, was der Prozessor direkt versteht. Die ganzen schönen Konstrukte höherer Programmiersprachen, die dem Programmierer erlauben, seine Algorithmen in verständliche, (ziemlich) fehlerfreie Programme zu übertragen fehlen:

- keine komplexen Anweisungen
- keine komfortablen for, while, reapeat-until Schleifen, sondern fast nur gotos
- keine strukturierten Datentypen
- keine Unterprogramme mit Parameterübergabe

Beispiel: Addition

C Anweisung

```
summe = a + b + c + d;

summe = a;
summe = summe + b;
summe = summe + c;
summe = summe + d;
```

Assembler Anweisung

```
mov eax,[a]
add eax,[b]
add eax,[c]
add eax,[d]
```

Beispiel: Verzweigung

C Anweisung

```
if (a == 4711 ) { ... } else { ... }
```

Assembler Anweisung

```
           cmp eax,4711
           jne ungleich
gleich:    ...
           jmp weiter
ungleich:  ...
weiter:    ...
```

Beispiel: Zählschleife

C Anweisung

```
for( i=0; i<100; i++) {
    summe = summe + a:
}
```

Assembler Anweisung

```
            mov ecx,100
schleife:   add eax,[a]
            loop schleife
```

Register

Ein Register ist ein winziges Stückchen Hardware innerhalb des Prozessors, das beim 80386 und höher bis zu 32 Bits, also 32 Ziffern im Bereich 0 und 1 speichern kann.

Allgemeine Register	
Name	**Bemerkung**
aex	allgemein verwendbar, spezielle Bedeutung bei Arithmetik
ebx	allgemein verwendbar
ecx	allgemein verwendbar, spezielle Bedeutung bei Schleifen
edx	allgemein verwendbar
ebp	Basepointer
esi	Quelle für Stringoperationen
edi	Ziel für Stringoperationen
esp	Stackpointer

Segmentregister	
Name	**Bemerkung**
cs	Codesegemet
ds	Datasegment
ss	Stacksegement
es	beliebiges Segment
fs	beliebiges Segment
gs	beliebiges Segment

Sonstige Register	
Name	**Bemerkung**
eip	Instruction Pointer
ef	Flags

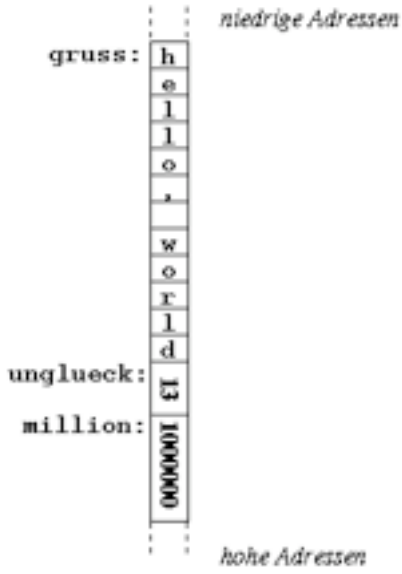
Speicher

Meistens reichen die Register nicht aus, um ein Problem zu lösen. In diesem Fall muss auf den Hauptspeicher des Computers zugegriffen werden, der erheblich mehr Information speichern kann. Für den Assemblerpogrammierer sieht der Hauptspeicher wie ein riesiges Array von Registern aus, die je nach Wunsch 8, 16 oder 32 Bits "breit" sind.

In einem Assemblerprogramm können Variablen angelegt werden, indem einer Speicheradresse ein Label zugeordnet und dabei Speicherplatz in der gewünschten Größe reserviert wird.

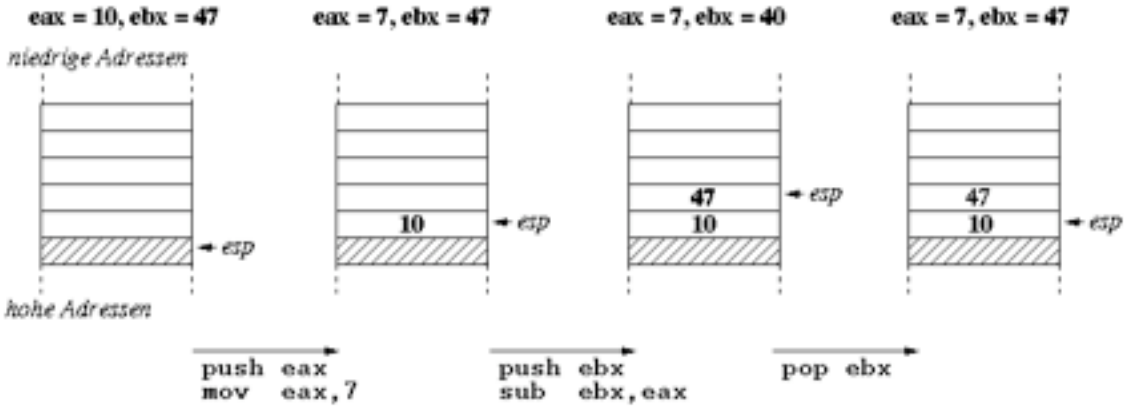
```
[SECTION .data]
gruss:       db 'hello, world'
unglueck:    dw 13
million:     dd 1000000


[SECTION .text]
             mov ax,[million]
             ...
```



Stack

Nicht immer will man sich ein neues Label ausdenken, nur um kurzfristig mal den Wert eines Registers zu speichern, beispielsweise, weil man das Register für eine bestimmte Anweisung benötigt, den alten Wert aber nicht verlieren möchte.
Der Stack ist eigentlich nichts weiter als ein Stück des Hauptspeichers, nur dass dort nicht mit festen Adressen gearbeitet wird, sondern die zu sichernden Daten einfach immer oben drauf geschrieben (*push*) bzw. von oben heruntergeholt werden (*pop*).



Adressierungsarten

Beim *mov* Befehl sind (u. a.) folgende Formen möglich, wobei der erste Operand stets das Ziel und der zweite stets die Quelle der Kopieraktion angeben:

- Registeradressierung: Der Wert eines Registers wird in ein anderes übertragen.
  ```
  mov ebx,edi
  ```
- Unmittelbare Adressierung: Die Konstante wird in das Register übertragen.
  ```
  mov ebx,1000
  ```
- Direkte Adressierung: Der Wert der an der angegebenen Speicherstelle steht, wird in das Register übertragen.
  ```
  mov ebx,[1000]
  ```
- Register-Indirekte Adressierung: Der Wert, der an der Speicherstelle steht, die durch das zweite Register bezeichnet wird, wird in das erste Register übertragen.
  ```
  mov ebx,[eax]
  ```
- Basis-Register Adressierung: Der Wert, der an der Speicherstelle steht, die sich durch die Summe des Inhalts des zweiten Registers und der Konstanten ergibt, wird in das erste Register übertragen.
  ```
  mov eax,[10+esi]
  ```