

---

# Assembler



Prozessor-  
unabhängig

Programm in einer Hochsprache

Übersetzer (Compiler)

Assemblerprogramm

Übersetzer (Assembler)

Maschinenprogramm

```
int fact (int n)
{
    f = 1;
    i = 1;
    while (i++ < n)
        f = f * i;
    return (f);
}
```

```
LDB    -1, X
CMPB   ,X
BGE    OUT
INCB
STB    -1, X
LDA    -2 ,X
MUL
STB    -2 ,X
BRA    REPEAT
```

```
0100110 01101111 01010101
0000100 01010110 01000010
0011111 11000110 11111000
0000000 00100100 01110001
```



**Maschinencode des  
Beispielprogramms:  
"Rotate Right"**

Sp.- Adr	OPC OP-Adr.	HEX	Octal
00010000	1001 10000001	10 9 81	020 4601
00010001	1000 00000000	11 8 00	021 4000
00010010	1011 10000010	12 B 82	022 5602
00010011	1011 10000100	13 B 84	023 5604
00010100	1010 10000011	14 A 83	024 5203
00010101	1001 10000000	15 9 80	025 4600
00010110	0101 00000000	16 5 00	026 2400
00010111	1010 10000000	17 A 80	027 5200
00011000	1001 10000010	18 9 82	030 4602
00011001	1000 00000000	19 8 00	031 4000
00011010	1011 10000010	1A B 82	032 5602
00011011	1011 10000011	1B B 83	033 5603
00011100	0001 00011110	1C 1 1E	034 0436
00011101	0010 00010101	1D 2 15	035 1025
00011110	0000 00000000	1E 0 00	036 0000

**Zahlendarstellung**

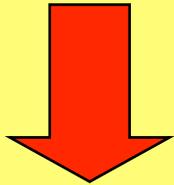
Dezimal (Basis 10)	Dual (Basis 2)	Oktal (Basis 8)	Hexadezimal (Basis 16)
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	21	10
.	.	.	.



# Assembler

---

Eingabe: Textdatei(en)  
Quellprogramm



Ausgabe: Maschinen  
Code



Assembler ist der „Compiler“ für die Maschinensprache eines Prozessors.

Unterstützt werden:

- ➔ Symbolische Befehle eines Prozessors (Mnemotechnische Codes)
- ➔ Notation für Adressierungsarten
- ➔ Festlegen von Speicheradressen für Programme und Daten
- ➔ Symbolische Speicheradressen (*Labels*)
- ➔ Durchführung von Adreßrechnungen



# Assembler

---

## Eigenschaften:

- reflektiert die Architektur des Rechners
- gibt genau an, was im Rechner geschieht, d.h. 1:1 Abbildung von Programmstatement und Maschinenoperation
- Vorhersagbarkeit (Predictability) für zeitkritische Anwendungen, z.B. in Steuerungssystemen
- Kompakter Code

Aussagen über Struktur und Leistungsfähigkeit eines Prozessors werden durch das Programmiermodell eines Rechners ermöglicht. Es erlaubt die Beurteilung einer Rechnerarchitektur vom Standpunkt eines Compiler- oder Betriebssystementwicklers. Der Assembler ermöglicht den Umgang mit dem Rechner auf der Ebene des Programmiermodells.

Das Programmiermodell umfaßt:

Befehlssatz,  
Registerorganisation,  
Speicherorganisation  
Adressierungsmodi  
Ausnahmebehandlung  
Ein/Ausgabe

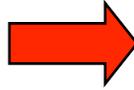


```

LOOP   LDA    25, PCR
        ADDA  LABEL
        BHI   LOOP

LABEL  some data

```



```

A6
8C
BB
label ?
label ?
22
F9

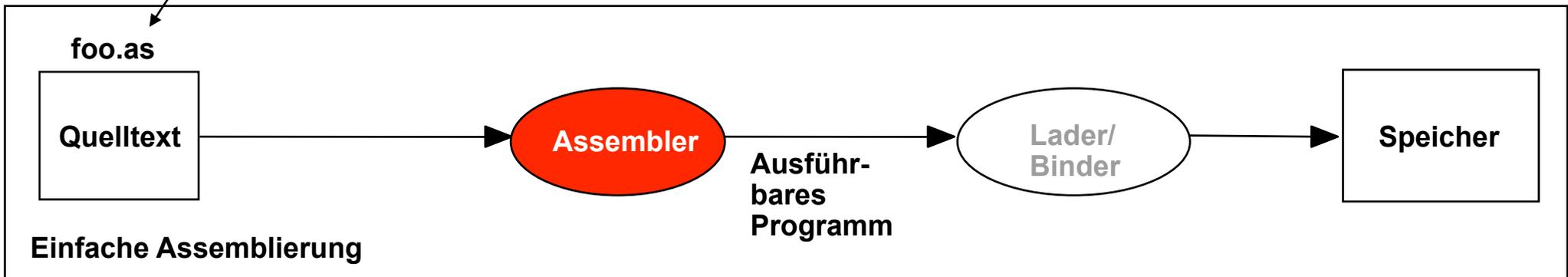
```



```

10100110
10001100
00001101
10111011
01101101
01101101
00100010
11111001

```



- Zuordnung von Namen zu Binärzahlen
- Mnemotechnische Codes
- Statische Reservierung von Speicherplatz
- Berechnung von Sprungadressen
- Erstellen der Speicherbelegung



---

## **Assemblerbefehle** (Zeilenorientiert)

**[<Label>] <Opcode> [<Operanden>] [<Kommentar>]**

**Label:** symbolische Marke die (meist) mit der aktuellen Speicheradresse verbunden wird

**Opcode:** symbolischer Maschinenbefehl oder Assemblerdirektive (Steueranweisung für den Assembler)

**Operanden:** z.B. Angaben zur Adressierungsart, Adressen

**Kommentar:** meist bis Ende der Zeile (oft eingeleitet mit speziellem Zeichen, z.B. \* oder ;)



---

## Assembler Direktiven:

Assembler Direktiven sind Statements, die nicht direkt ausführbarem Code entsprechen, sondern zur Steuerung der Übersetzung dienen.

<b>EQUATE oder DEFINE</b>	<b>name</b>	<b>EQU</b>	<b>number</b>
	<b>name</b>	<b>EQU</b>	<b>name</b>
<b>ORIGIN</b>	<b>ORG</b>	<b>name</b>	
	<b>ORG</b>	<b>number</b>	
<b>RESERVE</b>	<b>RMB</b>	<b>number</b>	
<b>DATA</b>	<b>FCB</b>	<b>number</b>	
	<b>FDB</b>	<b>number</b>	
	<b>FCC</b>	<b>string (ASCII)</b>	



# Zahlendarstellung

---

Dezimalzahlen: wie gewohnt,	z.B. <b>55</b> meist default, sonst oft <b>&amp;55</b>
Hexadezimalzahlen: mit führendem <b>\$</b> -Zeichen,	z.B. <b>\$a0</b>
Octalzahlen: mit führendem <b>@</b> Zeichen	z.B. <b>@71</b>
Binärzahlen: mit führendem <b>%</b> -Zeichen,	z.B. <b>%00101001</b>
Oft auch Berechnungen im Text möglich:	z.B. <b>\$a0+5</b>



## Konstantendefinition:

# : bezeichnet Konstanten im Programmtext (immediate) LDA #23, ADDA #\$AF

FCB, FDB, FCC: typische Verwendung: Label FCB %10101010

Label FCC „das ist eine Zeichenkette“

Zuordnung Konstante → Symbolischer name:

Assemblerdirektive EQU

Beispiel: Length EQU \$0400 Label Length wird mit Konstante \$0400 verknüpft

Length wird nun vom Assembler immer durch 400 ersetzt.

Daher auch in Befehlen verwendbar, z.B. LDX #Length+\$8

## Reservierung von Variablen:

RMB: Reserve Memory Byte(s) typische Verwendung Label RMB 2  
Array RMB 256



# Beispiel Codierung einer Siebensegmentanzeige

SP. Adr	OPC	OP	SP.M	MNE.	OP	Kommentar
					ORG \$A000	
					MMSEG EQU \$FF00	
A000	3F				CLRB	setzt den Fehlercode : löschen des Displays
A001	96	A0			LDA \$A0	Wert in den Acc. A holen
A003	81	09			CMPA #9	ist der Wert eine Ziffer 0...9 ?
A005	22	05			BHI DONE	wenn A > 9 bleibt der Fehlercode gesetzt
A007	8E	AF00			LDX #SSG	Lade X mit dem Beginn der Conversionsliste "SSG"
A00A	E6	86			LDB A, X	Lade 7-Segment Muster in Acc A
A00C	D7	FF00	DONE		STB MMSEG	Abspeichern auf ein Ausgabegerät
A010	39				RTS	Return from Subroutine
AF00			SSG	FCB	\$3F, \$06, \$5B, \$4F, \$66	
AF05				FCB	\$6D, \$7D, \$07, \$7F, \$6F	
FF00			MMSEG	.....	.....	Adresse des Ausgabegeräts

**Im Assembler-Statement: 'ADD \$69' bezeichnet '\$69' eine Adresse.  
Ein unmittelbarer Operand wird durch '#' gekennzeichnet: ADD #\$69 !**



**Macros:** spezifizieren eine Folge von **Assembler Statements**, die bei der **Assemblierung** in den Code "expandiert" werden.

### **Bedingte Assemblierung:**

**if CONDITION = TRUE then expand code  
else endif**

**IF CONDITION**

•

- **Code wird zwischen IF und ENDIF in das  
Maschinenprogramm eingefügt**

•

**ENDIF**



---

Beispiel für bedingte Assemblierung:

**HasFloppy EQU 1**

...

**if HasFloppy**

**LDA #... \* wird nur assembliert**

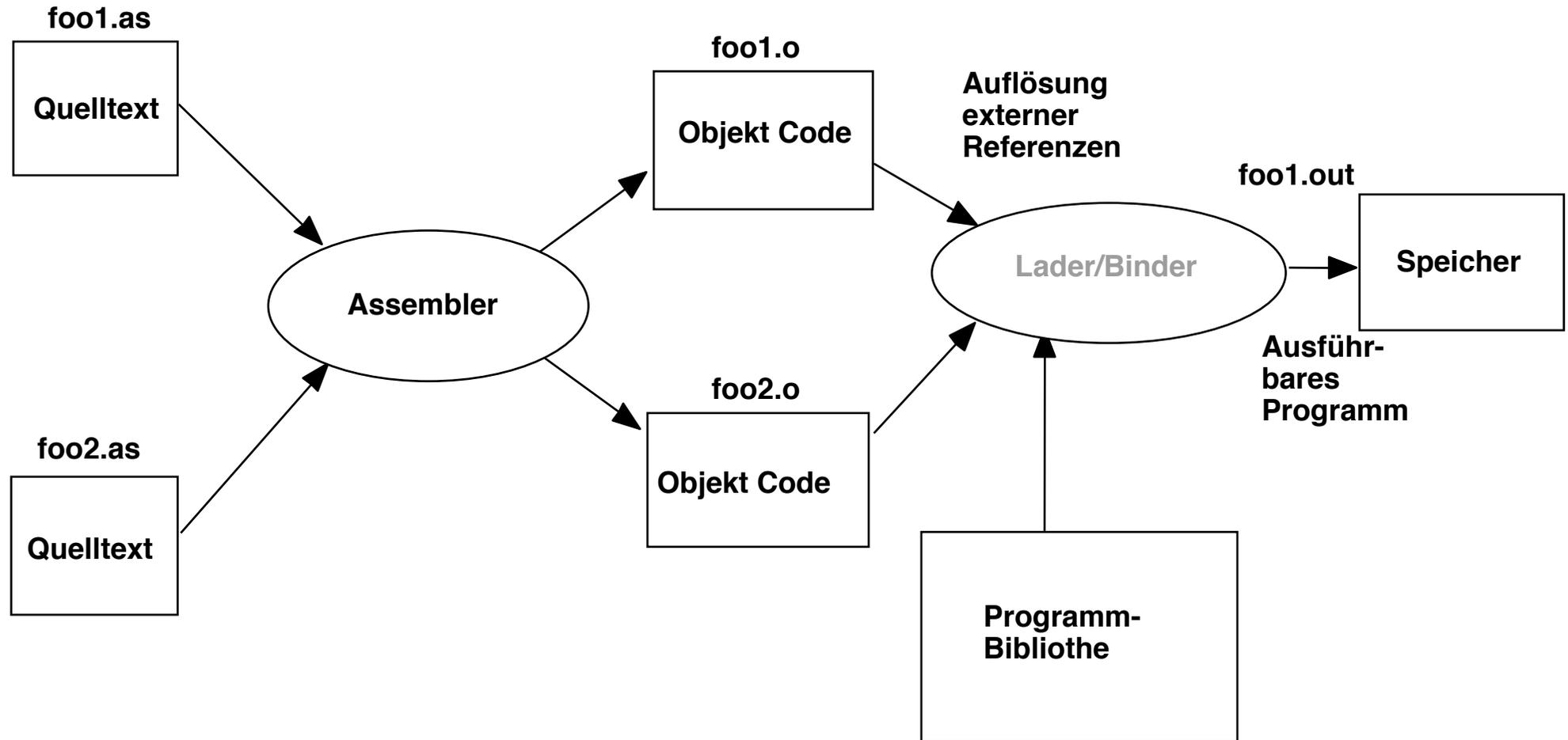
**... \* wenn HasFloppy ungleich Null**

**endif**



# Assemblieren und Binden mehrerer Assemblermoduln

---



---

## Binder

Zusammenfügen getrennt übersetzter und assemblierter Softwareteile  
insbesondere Einbinden von Bibliotheken

## Lader

Laden eines Programms in den Hauptspeicher  
Verwaltung und Berücksichtigung von Relokationsinformation  
Programm verschiebbar im Hauptspeicher  
Bindemodul ortsunabhängig



---

## Direktiven des Linkers:

**EXTERNAL (import)**  
**ENTRY (export)**

**EXT name**  
**ENT name**



## fac.c

```
main ()
{
    printf ("die Fakultät von 5 ist: %d\n", fact (5));
}

int fact (int n)
{
    if (n < 1)
        return (1);

    else
        return (n * fact (n-1));
}
```

gcc -S fac.c

## fac.s

```
.file "fac.c"
.version "01.01"
gcc2_compiled.:
.section .rodata
.LC0:
.string "die Fakultät von 5 ist: %d\n"
.text
.align 16
.globl main
.type main,@function
main:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    addl $-8,%esp
    addl $-12,%esp
    pushl $5
    call fact
    addl $16,%esp
    movl %eax,%eax
    pushl %eax
    pushl $.LC0
    call printf
    addl $16,%esp
.L2:
    movl %ebp,%esp
    popl %ebp
    ret
.Lfel:
.size main,.Lfel-main
.align 16
.globl fact
.type fact,@function
fact:
    pushl %ebp
    movl %esp,%ebp
    subl $8,%esp
    cmpl $0,8(%ebp)
    jg .L4
    movl $1,%eax
    jmp .L3
    jmp .L5
.p2align 4,,7
.
.
.
.
```



# as -a fac.s > fac1.s

```

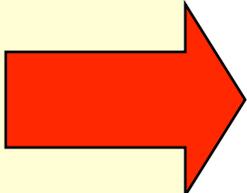
GAS LISTING fac.s                                     page 1
1
2
3
4 gcc2_compiled.:
5 .section .rodata
6 .LC0:
7 0000 64696520
8 46616B75
9 6C74E474
10 20766F6E
11 20352069
12
13 .text
14
15 .align 16
16
17 .globl main
18 .type main,@function
19
20 main:
21 pushl %ebp
22 movl %esp,%ebp
23 subl $8,%esp
24 addl $-8,%esp
25 addl $-12,%esp
26 pushl $5
27 call fact
28
29 addl $16,%esp
30 movl %eax,%eax
31 pushl %eax
32 pushl $.LC0
33
34 call printf
35
36 addl $16,%esp
37
38 .L2:
39 movl %ebp,%esp
40 popl %ebp
41 ret
42
43 .Lf1:
44
45 .size main,.Lf1-main
46 .align 16
47
48 .globl fact
49 .type fact,@function
50
51 fact:
52 pushl %ebp
53 movl %esp,%ebp
54 subl $8,%esp
55 cmpl $0,8(%ebp)
56 jg .L4
57 movl $1,%eax
58
59 jmp .L3
60 jmp .L5
61 .p2align 4,,7
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

# od -x fac.o

```

0000000 457f 464c 0101 0001 0000 0000 0000 0000
0000020 0001 0003 0001 0000 0000 0000 0000 0000
0000040 0150 0000 0000 0000 0034 0000 0000 0028
0000060 000b 0008 0000 0000 0000 0000 0000 0000
0000100 8955 83e5 08ec c483 83f8 f4c4 056a fce8
0000120 ffff 83ff 10c4 c089 6850 0000 0000 fce8
0000140 ffff 83ff 10c4 ec89 c35d b68d 0000 0000
0000160 8955 83e5 08ec 7d83 0008 097f 01b8 0000
0000200 eb00 eb1e 831c f4c4 458b 4808 e850 fffc
0000220 ffff c483 8910 89c0 0fc2 55af 8908 ebd0
0000240 8900 5dec 8dc3 2674 8d00 27bc 0000 0000
0000260 0008 0000 0000 0000 0001 0000 3130 302e
0000300 0031 0000 6964 2065 6146 756b 746c 74e4
0000320 7620 6e6f 3520 6920 7473 203a 6425 000a
0000340 4700 4343 203a 4728 554e 2029 2e32 3539
0000360 332e 3220 3030 3031 3133 2035 5328 5375
0000400 2945 0000 732e 6d79 6174 0062 732e 7274
0000420 6174 0062 732e 7368 7274 6174 0062 742e
0000440 7865 0074 722e 6c65 742e 7865 0074 642e
0000460 7461 0061 622e 7373 2e00 6f6e 6574 2e00
0000500 6f72 6164 6174 2e00 6f63 6d6d 6e65 0074
0000520 0000 0000 0000 0000 0000 0000 0000 0000
*
```



# as -o fac.o fac.s

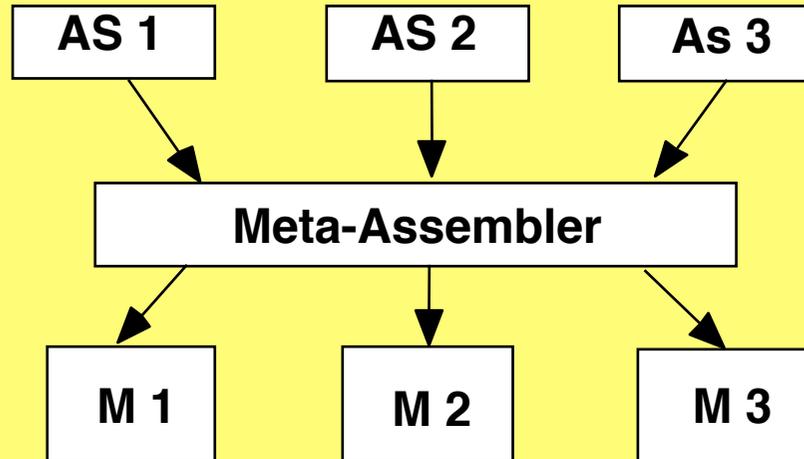


# Assembler Typen

**Cross-Assembler:** läuft auf einer (meist leistungsfähigen) Wirtsmaschine  
prodiziert Code für eine andere Zielmaschine  
Wirtsmaschine mit guter Textverarbeitung/Filesystem/Simulator etc.  
Zielmaschine z.B. "embedded Computer"

**Resident Assembler:** Wirtsmaschine und Zielmaschine sind gleich

**Meta-Assembler:** Assembler für verschiedene Instruktionssätze



**Maschinenunabhängiger Binärcode:** Die Hardware/Firmware emuliert einen einheitlichen Architektur-unabhängigen Maschinenbefehlssatz

## Nachteile von Assembler gegenüber einer höheren Programmiersprache:

---

- **Maschinennahe Abstraktionsebene**
  - einfache Datentypen
  - keine Modularisierung auf der Sprachebene
  - wenige Überprüfungen der Korrektheit möglich
- **Portierbarkeit, Lesbarkeit, Fehlermöglichkeiten, Wartbarkeit**
- **Asm.Progr. wird z.B bei RISC-Prozessoren immer schwieriger (delayed branching, umsordieren v. Instr. bei Datenabhängigkeiten)**

**Assembler ist (meist) keine Alternative zu einer problemorientierten Hochsprache sondern dient zur Beurteilung einer Rechnerarchitektur und elementarer Programmierkonstrukte.**

---