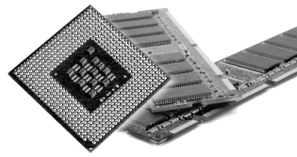
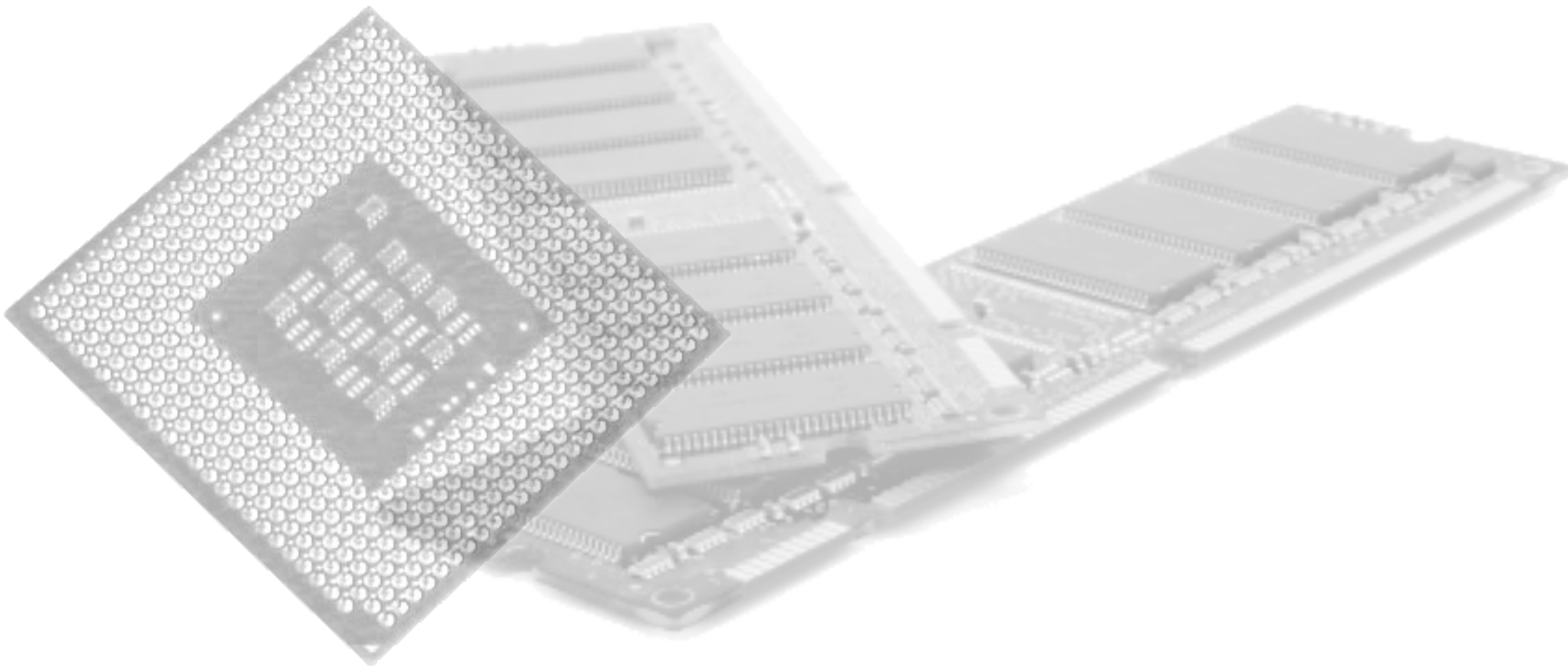
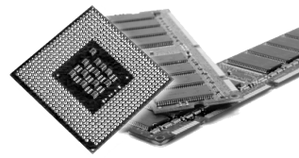


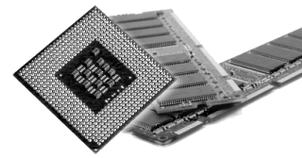
C und Assembler





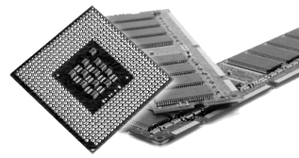
Beispiel in C

```
10 int main() {
11     int i;
12     int a;
13     a = 1;
14     i = 5;
15     i--;
16     for (i = 0;i < 5;i++) {
17         a++;
18     }
19     return 0;
20 }
```



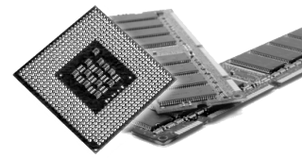
```
6   _main:
7   ## BB#0:
8     push  ebp
9     mov ebp, esp
10    sub esp, 12
11    mov dword ptr [ebp - 4], 0
12    mov dword ptr [ebp - 12], 1
13    mov dword ptr [ebp - 8], 5
14    mov eax, dword ptr [ebp - 8]
15    add eax, -1
16    mov dword ptr [ebp - 8], eax
17    mov dword ptr [ebp - 8], 0
18  LBB0_1:
19    cmp dword ptr [ebp - 8], 5
20    jge LBB0_4
21  ## BB#2:
22    mov eax, dword ptr [ebp - 12]
23    add eax, 1
24    mov dword ptr [ebp - 12], eax
25  ## BB#3:
26    mov eax, dword ptr [ebp - 8]
27    add eax, 1
28    mov dword ptr [ebp - 8], eax
29    jmp LBB0_1
30  LBB0_4:
31    xor eax, eax
32    add esp, 12
33    pop ebp
34    ret
```

```
10 int main() {
11     int i;
12     int a;
13     a = 1;
14     i = 5;
15     i--;
16     for (i = 0;i < 5;i++) {
17         a++;
18     }
19     return 0;
20 }
```



```c
int main() {
    int i;
    int a;
    a = 1;
    i = 5;
    i--;
    for (i = 0;i < 5;i++) {
        a++;
    }
    return 0;
}
```

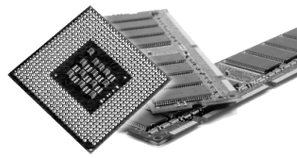
```
    main:
## BB#0:
    push  ebp
    mov ebp, esp
    sub esp, 12
    mov dword ptr [ebp - 4], 0
    mov dword ptr [ebp - 12], 1
    mov dword ptr [ebp - 8], 5
```



```
10 int main() {
11     int i;
12     int a;
13     a = 1;
14     i = 5;
15     i--;
16     for (i = 0;i < 5;i++) {
17         a++;
18     }
19     return 0;
20 }
```

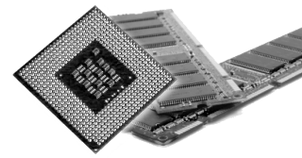
```
 6     main:
 7     ## BB#0:
 8       push  ebp
 9       mov ebp, esp
10       sub esp, 12
11       mov dword ptr [ebp - 4], 0
12       mov dword ptr [ebp - 12], 1
13       mov dword ptr [ebp - 8], 5
```

Stack sichern
Stack "leeren"
Stack mit 3 Double belegen
Null auf Stack
Variable a = 1
Variable i = 5



```c
int main() {
    int i;
    int a;
    a = 1;
    i = 5;
    i--;
    for (i = 0;i < 5;i++) {
        a++;
    }
    return 0;
}
```

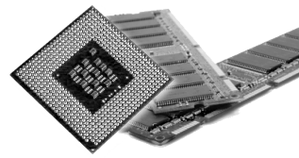
```
    mov eax, dword ptr [ebp - 8]
    add eax, -1
    mov dword ptr [ebp - 8], eax
```



```
10 int main() {
11     int i;
12     int a;
13     a = 1;
14     i = 5;
15     i--;
16     for (i = 0;i < 5;i++) {
17         a++;
18     }
19     return 0;
20 }
```

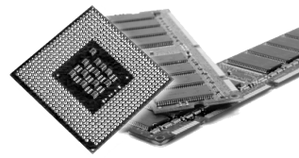
```
14     mov eax, dword ptr [ebp - 8]
15     add eax, -1
16     mov dword ptr [ebp - 8], eax
```

i in eax schreiben
i-- rechnen
eax wieder in i schreiben



```
18   LBB0_1:
19     cmp dword ptr [ebp - 8], 5
20     jge LBB0_4
21   ## BB#2:
22     mov eax, dword ptr [ebp - 12]
23     add eax, 1
24     mov dword ptr [ebp - 12], eax
25   ## BB#3:
26     mov eax, dword ptr [ebp - 8]
27     add eax, 1
28     mov dword ptr [ebp - 8], eax
29     jmp LBB0_1
30   LBB0_4:
```

```
10 int main() {
11     int i;
12     int a;
13     a = 1;
14     i = 5;
15     i--;
16     for (i = 0;i < 5;i++) {
17         a++;
18     }
19     return 0;
20 }
```



i mit 5 vergleichen
wenn es größer oder gleich ist -> Sprung

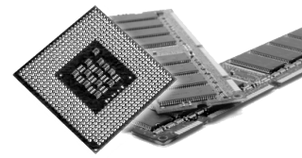
```
18  LBB0_1:
19    cmp dword ptr [ebp - 8], 5
20    jge LBB0_4
21  ## BB#2:
22    mov eax, dword ptr [ebp - 12]
23    add eax, 1
24    mov dword ptr [ebp - 12], eax
25  ## BB#3:
26    mov eax, dword ptr [ebp - 8]
27    add eax, 1
28    mov dword ptr [ebp - 8], eax
29    jmp LBB0_1
30  LBB0_4:
```

```
10 int main() {
11     int i;
12     int a;
13     a = 1;
14     i = 5;
15     i--;
16     for (i = 0;i < 5;i++) {
17         a++;
18     }
19     return 0;
20 }
```

a++ rechnen

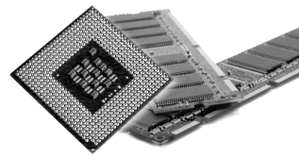
i++ rechnen

nach oben springen



```c
int main() {
    int i;
    int a;
    a = 1;
    i = 5;
    i--;
    for (i = 0;i < 5;i++) {
        a++;
    }
    return 0;
}
```

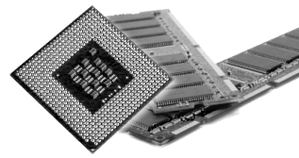
```
LBB0_4:
    xor eax, eax
    add esp, 12
    pop ebp
    ret
```



```c
int main() {
    int i;
    int a;
    a = 1;
    i = 5;
    i--;
    for (i = 0;i < 5;i++) {
        a++;
    }
    return 0;
}
```

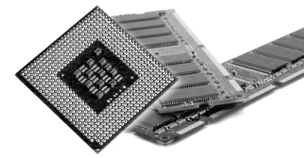
```
LBB0_4:
    xor eax, eax
    add esp, 12
    pop ebp
    ret
```

Rückgabewert in eax: eax Nullsetzen
Stack wieder leeren: esp + 12
Basepointer wiederherstellen



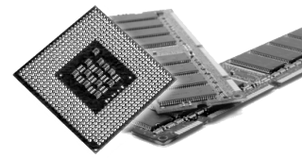
```
6   _main:
7   ## BB#0:
8     push  ebp
9     mov ebp, esp
10    sub esp, 12
11    mov dword ptr [ebp - 4], 0
12    mov dword ptr [ebp - 12], 1
13    mov dword ptr [ebp - 8], 5
14    mov eax, dword ptr [ebp - 8]
15    add eax, -1
16    mov dword ptr [ebp - 8], eax
17    mov dword ptr [ebp - 8], 0
18  LBB0_1:
19    cmp dword ptr [ebp - 8], 5
20    jge LBB0_4
21  ## BB#2:
22    mov eax, dword ptr [ebp - 12]
23    add eax, 1
24    mov dword ptr [ebp - 12], eax
25  ## BB#3:
26    mov eax, dword ptr [ebp - 8]
27    add eax, 1
28    mov dword ptr [ebp - 8], eax
29    jmp LBB0_1
30  LBB0_4:
31    xor eax, eax
32    add esp, 12
33    pop ebp
34    ret
```

```
10 int main() {
11     int i;
12     int a;
13     a = 1;
14     i = 5;
15     i--;
16     for (i = 0;i < 5;i++) {
17         a++;
18     }
19     return 0;
20 }
```



Optimierungen

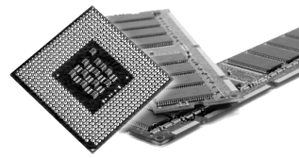
- gcc
  - -O0 (Beispiel zuvor)
  - -O1:
  - -O2:
  - -O3:



Optimierungen

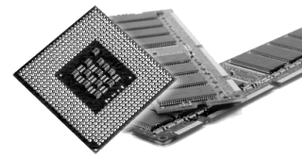
- gcc
  - -O0 (Beispiel zuvor)
  - -O1:
  - -O2:
  - -O3:

```
6   _main:
7   ## BB#0:
8     push  ebp
9     mov ebp, esp
10    xor eax, eax
11    pop ebp
12    ret
```

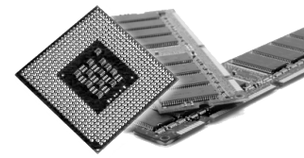


```c
#include "stdio.h"

int main() {
    int i;
    int a;
    a = 1;
    i = 5;
    i--;
    for (i = 0;i < 5;i++) {
        a++;
    }
    printf ("%d",a);
    return 0;
}
```

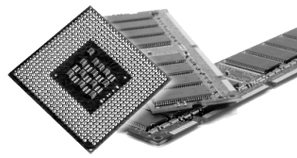


```
6   _main:                                  ## @main
7   ## BB#0:
8     push  ebp
9     mov ebp, esp
10    sub esp, 8
11    call  L0$pb
12  L0$pb:
13    pop eax
14    sub esp, 8
15    lea eax, [eax + L_.str-L0$pb]
16    push  6
17    push  eax
18    call  _printf
19    add esp, 16
20    xor eax, eax
21    add esp, 8
22    pop ebp
23    ret
24
25    .section  __TEXT,__cstring,cstring_literals
26  L_.str:                                 ## @.str
27    .asciz  "%d"
```



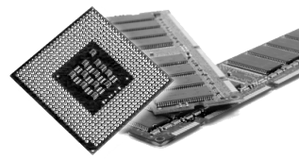
Andere Assembler

- z.B. 64 bit AT&T Syntax
  - mov $0x05, %rax
  - $ -> Konstante
  - % -> Register
  - Quelle, Ziel
  - usw.
- Bsp. Xcode mit macOS



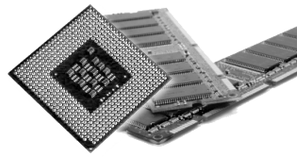
```
63  Ltmp0:
64      .loc    38 15 7 prologue_end    ## /Users/sebi/Documents/
            Data/CinAssemblerAnschauen/CinAssemblerAnschauen/
            main.cpp:15:7
65      movl    $0, -24(%rbp)
66      .loc    38 16 7                 ## /Users/sebi/Documents/
            Data/CinAssemblerAnschauen/CinAssemblerAnschauen/
            main.cpp:16:7
67      movl    $5, -20(%rbp)
68      .loc    38 17 6                 ## /Users/sebi/Documents/
            Data/CinAssemblerAnschauen/CinAssemblerAnschauen/
            main.cpp:17:6
69      movl    -20(%rbp), %edi
70      addl    $-1, %edi
71      movl    %edi, -20(%rbp)
```

```
9   int main(int argc, const char * argv[]) {
10      int i;
11      int a;
12      a = 0;
13      i = 5;
14      i--;
15      for (i = 0;i < 5;i++) {
16          a++;
17      }
18      return 0;
19  }
```



```
72  Ltmp1:
73      .loc    38 18 12                        ##
            Data/CinAssemblerAnschauen/CinA
            main.cpp:18:12
74      movl    $0, -20(%rbp)
75  LBB0_1:
        Header: Depth=1
76  Ltmp2:
77      .loc    38 18 18 is_stmt 0 discrimi
            sebi/Documents/Data/CinAssemble
            CinAssemblerAnschauen/main.cpp:18:18
78      cmpl    $5, -20(%rbp)
79  Ltmp3:
80      .loc    38 18 5 discriminator 1 ## /Users/sebi/Documents/
            Data/CinAssemblerAnschauen/CinAssemblerAnschauen/
            main.cpp:18:5
81      jge LBB0_4
82  ## BB#2:                                ##    in Loop:
        Header=BB0_1 Depth=1
83  Ltmp4:
84      .loc    38 19 10 is_stmt 1      ## /Users/sebi/Documents/
            Data/CinAssemblerAnschauen/CinAssemblerAnschauen/
            main.cpp:19:10
85      movl    -24(%rbp), %eax
86      addl    $1, %eax
87      movl    %eax, -24(%rbp)
```

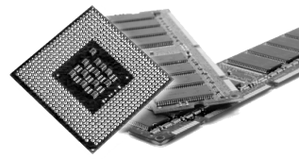
```
9   int main(int argc, const char * argv[]) {
10      int i;
11      int a;
12      a = 0;
13      i = 5;
14      i--;
15      for (i = 0;i < 5;i++) {
16          a++;
17      }
18      return 0;
19  }
```



```
 9  int main(int argc, const char * argv[]) {
10      int i;
11      int a;
12      a = 0;
13      i = 5;
14      i--;
15      for (i = 0;i < 5;i++) {
16          a++;
17      }
18      return 0;
19  }
```

```
88  Ltmp5:
89  ## BB#3:
        Header=BB0_1 Depth=1
90      .loc    38 18 23 discriminator 2 ##
            Documents/Data/CinAssemblerAnsc
            CinAssemblerAnschauen/main.cpp:18:23
91      movl    -20(%rbp), %eax
92      addl    $1, %eax
93      movl    %eax, -20(%rbp)
94      .loc    38 18 5 is_stmt 0 discriminator 2 ## /Users/sebi/
            Documents/Data/CinAssemblerAnschauen/
            CinAssemblerAnschauen/main.cpp:18:5
95      jmp LBB0_1
```

```
75  LBB0_1:
        Header: Depth=1
76  Ltmp2:
77      .loc    38 18 18 is_stmt
            sebi/Documents/Data/C
            CinAssemblerAnschauen
78      cmpl    $5, -20(%rbp)
```



```
 9  int main(int argc, const char * argv[]) {
10      int i;
11      int a;
12      a = 0;
13      i = 5;
14      i--;
15      for (i = 0;i < 5;i++) {
16          a++;
17      }
18      return 0;
19  }
```

```
96  Ltmp6:
97  LBB0_4:
98      .loc    38 0 5 discriminator 2   ## /Users/sebi/Documents/
            Data/CinAssemblerAnschauen/CinAssemblerAnschauen/
            main.cpp:0:5
99      xorl    %eax, %eax
100     .loc    38 21 5 is_stmt 1        ## /Users/sebi/Documents/
            Data/CinAssemblerAnschauen/CinAssemblerAnschauen/
            main.cpp:21:5
101     popq    %rbp
102     retq
```