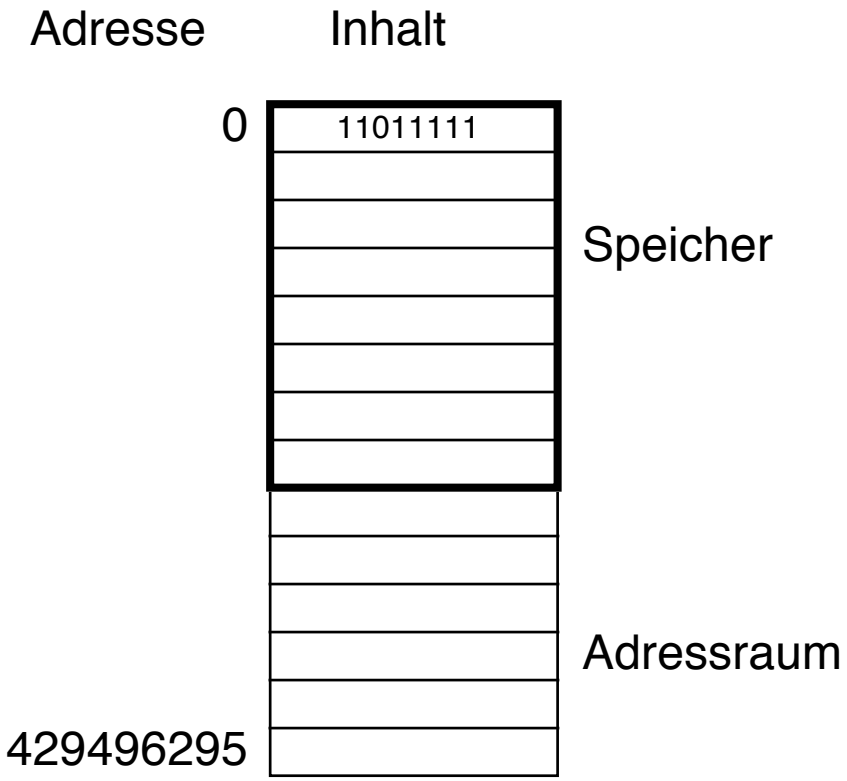
Kapitel 2: Computerarchitektur und Assembler

Themen

- Speicher
- Datenstrukturen
- Register
- Befehle
- Funktionen

Speicher



Interpretation des Inhalts ist nicht vorgegeben

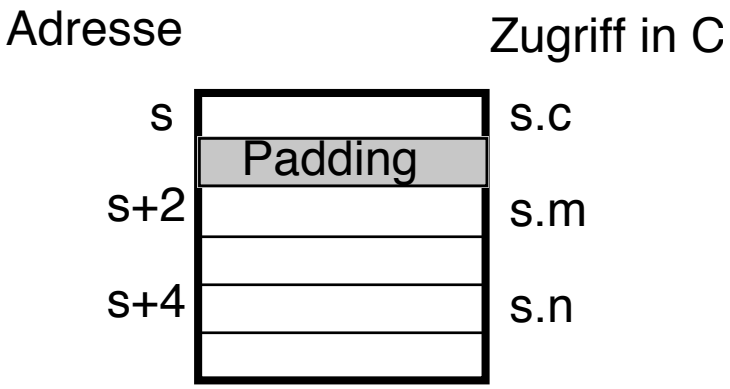
11011111: ß, −33, 223, oder ...

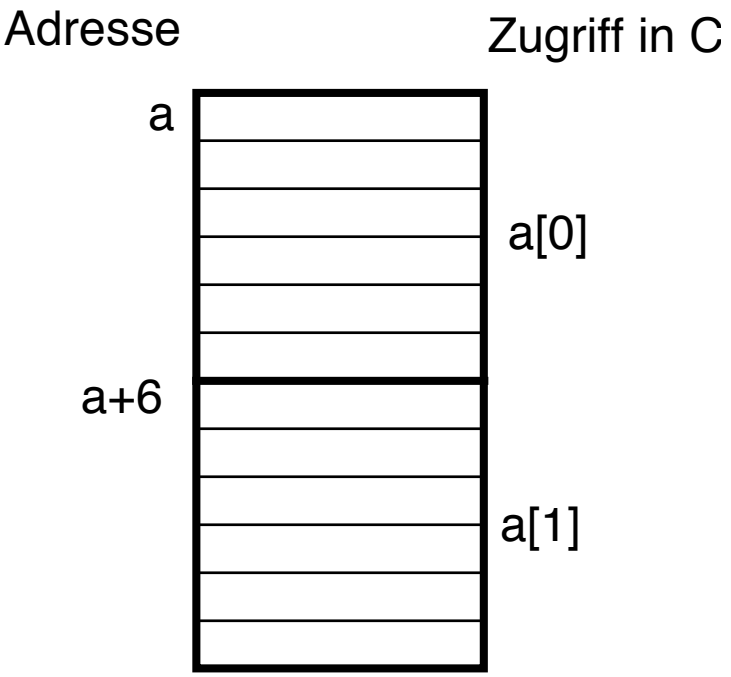
Register

- AMD64: rax–rdx, rsp, rbp, rsi, rdi, r8–r15, rip
- Zwischenspeicher für Operanden und häufig verwendete Daten
- keine indirekten Zugriffe möglich

Datenstrukturen

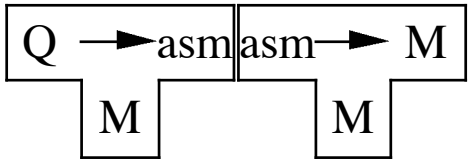
```
struct {
  char c;
  short m;
  short n;
}s, a[2];
```





Maschinencode und Assemblercode

- **Maschinencode:** `01001000 10000011 11000111 00001000`
- **Assemblercode:** `addq $8, %rdi`



Speicherzugriffe

```
leaq  -4(%ebp), %rax          /* rax = ebp-4          */
leaq  (%rdi,%rsi), %rax       /* rax = rdi+rsi        */
leaq  8(%rdi,%rsi), %rax      /* rax = rdi+rsi+8      */
leaq  4(,%rdi,8), %rax        /* rax = rdi*8+4        */
leaq  2(%rax,%rax,2), %rax    /* rax = rax+rax*2+2    */

/* a[i].m++; */

leaq  (%rdi,%rdi), %rax       /* rax = rdi+rdi        */
incw  a+2(%rax,%rax,2)        /* *(a+2+rax+rax*2)++   */
```

Rechenbefehle

```
/* b = a*(a-1)/2; */

    leaq   -1(%rdi), %rax    /* rax = rdi-1  */
    imulq  %rdi,%rax         /* rax *= rdi   */
    sarq   $1,%rax           /* rax = rax>>1 */
```

Kontrollfluss

```
/* if (a!=b) b--; */

   cmp  %rcx,%rdi      /* flags = rdi - rcx          */
   je   ziel           /* if (rdi==rcx) goto ziel */
   subq $1, %rdi       /* rdi--                      */
ziel:
```

Beispiel

```
long vsum (long x[], long n) {
   long s = 0, *p = x;
   for (;p<x+n; p++)
      s += *p;
   return s;
}

/* x/p in rdi, n in rsi, s in rax, x+n in rcx      */
  movq  $0, %rax          /* rax = 0               */
  leaq  (%rdi,%rsi,8), %rcx /* rcx = rdi+rsi*8     */
  cmpq  %rcx, %rdi        /* flags = rdi - rcx     */
  jae   exit              /* if (rdi>=rcx) goto exit*/
loop:
  addq  (%rdi), %rax      /* rax += *(rdi)         */
  addq  $8, %rdi          /* rdi += 8              */
  cmpq  %rcx, %rdi        /* flags = rdi - rcx     */
  jb    loop              /* if (rdi<rcx) goto loop */
exit:
```

Funktionen

- **Aufrufkonventionen zwischen *Caller* und *Callee***
  - **Parameterübergabe**
  - **Wer sichert welche Register**

- **Einsprunglabel**

```
/* long f (long a) {return a*(a-1)/2;} */

   .globl f
f:                              /* Funktionsname    */
    leaq   -1(%rdi), %rax       /* rax = rdi-1      */
    imulq  %rdi,%rax            /* rax *= rdi       */
    sarq   $1,%rax              /* rax = rax>>1     */
    ret                         /* return to caller */
```

Funktionen mit *Activation Record*

- **Sichern der *callee-saved*-Register**
- **Sichern der *caller-saved*-Register**

```
g:                      /*  Funktionsname g, a in rdi, b in rsi   */

  pushq %rsi            /*  caller-saved Register rsi (b) sichern */
  call  f               /*  f aufrufen                            */
  popq  %rsi            /*  rsi wiederherstellen                  */
  addq  %rsi, %rax      /*  rsi zu Ergebnis von f addieren        */
  ret                   /*  Rücksprung                            */
```

Zusammenfassung Computerarchitektur und Assembler

- Speicher und Register
- Adressarithmetik für Datenstrukturen
- Maschinencode und Assemblercode
- Rechnen, Speicherzugriffe, Kontrollfluss
- Assembleranweisungen
- Aufrufkonventionen