Daten im Memory

• Alignement

− Alle Datentypen nach ihrer Grösse aligned: [vgl. sizeof(..) in C]

- char 1 Byte (»beliebige« Speicherstelle)
- short 2 Byte (adr % 2 == 0)
- int, long 4 Byte (adr % 4 == 0)
- long long 8 Byte (adr % 8 == 0)

− Grund: Einschränkungen durch Sytem− Prozessor−Bus
Effiziente Speicher Operationen

• Structs

− Alignement: Nach dem grössten Alignement aller enthaltenen Basistypen: $s\_align = max\ (\ m_1.align,\ldots,\ m_n.align)$

− Grösse: Summe der Grösse der einzelnen Basistypen
+ zusätzliche Bytes für Alignement

− Offsets: Jeder Member besitzt einen ***positiven*** Offset relativ zur struct−basis−adresse

− Beispiele:

```
struct {
   char c;
}
```

```
struct {
   char c;
   int i;
}
```

```
struct {
   short c;
   long long l;
}
```

```
struct {
   int a;
   char c;
   short s,t;
   long long l;
   char k;
}
```

• Arrays

− Betrachten als »homogene structs«
− Alignement: Alignement des MemberTypes
− Grösse: Grösse des MemberTypes multipliziert mit Anzahl Elementen: *a_size = sizeof(m.type) * #Elemente*
− Offsets: Positive Offsets relativ zur array−basis−adresse:
1D: *a[i] = a.base + (i*sizeof(m.type))*
− Beispiele:

```
char s[3];
```

```
long long l[2];
```

```
struct {char c; long long l;} a[3];
```