

Adressierungsart: Register (explizit)

- Registernummer als Operand
- Verwendung:
Schreiben/Lesen von in Registern gespeicherten Variablen:
`register short int r; /* r liegt im Register %ax */`
`register short int s; /* s liegt im Register %bx */`
`r = s;`
- ergibt compiliert:
`movw %bx, %ax`

Rechnerorganisation: Verwendungsmöglichkeiten Adressierungsarten

Robert Tolksdorf
Freie Universität Berlin

[1] © Robert Tolksdorf, Berlin

[2] © Robert Tolksdorf, Berlin

Adressierungsart: Unmittelbar / immediate

- Operand ist Bestandteil des Befehls
- Verwendung:
Zuweisung von Konstanten an Variablen
`register long int x; /* x liegt im Register %edx */`
`x = 14;`
- ergibt compiliert:
`movl $14, %edx`
- Bei Beschränkung der Befehlsgröße (z.B. bei RISC):
Ersetzung durch zwei Befehle
`movl 0x12345678, %eax`
↓
`sethi 0x1234, %eax`
`setlo 0x5678, %eax`

[3] © Robert Tolksdorf, Berlin

Adressierungsart: direkt absolut

- Adressen als Operanden
- Verwendung:
Schreiben/Lesen von im Speicher abgelegten Variablen
`char c; /* c hat die Adresse 1245 */`
`char d; /* d hat die Adresse 1246 */`
`c = 'A'; d = c;`
- ergibt compiliert:
`movb $65, 1245`
`movb 1245, 1246`
- oder (lesbarer):
`#define c 1245`
`#define d 1246`
`movb $65, c`
`movb c, d`

[4] © Robert Tolksdorf, Berlin

Adressierungsart: Register indirekt

- Registernummer im Befehl, EA ist Registerinhalt
- Verwendung:
Schreiben/Lesen von Variablen über Zeiger, die in Registern liegen

```
register int *p; /* p liegt im Register %edx */
```

```
int x; /* x hat Adresse 1456 */
```

```
p = &x;
```

```
*p = 14;
```

```
x = *p;
```

- ergibt compiliert:

```
movl $1456, %edx
```

```
movl $14, (%edx)
```

```
movl (%edx), 1456
```

[5] © Robert Tolksdorf, Berlin

Adressierungsart: Speicher indirekt

- Speicheradresse als Operant, EA ist Speicherinhalt
- Verwendung:
Schreiben bzw. Lesen von Variablen über Pointer, die im Speicher liegen

```
short int x; /* x hat die Adresse 340 */
```

```
short int *p; /* p hat die Adresse 342 */
```

```
p = &x;
```

```
*p = 45;
```

```
x = *p;
```

- ergibt compiliert:

```
movl $340, 342;
```

```
movw $45, (342)
```

```
movw (342), 340
```

[6] © Robert Tolksdorf, Berlin

Adressierungsart: Register indirekt mit pre/post de/increment

- Vor/nach dem Zugriff auf den Operanden wird Registerinhalt dekrementiert/inkrementiert (um die Größe des Operanden)

- Verwendung: Durchlaufen eines Records

```
struct rec { long int x; short int y; char c; };
```

```
struct rec a; /* a hat die Adresse 1232 */
```

```
struct rec b; /* b hat die Adresse 344 */
```

```
a = b;
```

- ergibt compiliert

```
movl $1232, %eax
```

```
movl $344, %ebx
```

```
movl (%ebx)+, (%eax)+
```

```
movw (%ebx)+, (%eax)+
```

```
movb (%ebx)+, (%eax)+
```

[7] © Robert Tolksdorf, Berlin

Adressierungsart: Register indirekt mit Index

- EA = Basisadresse + Index * Scale-Faktor

```
short int f[5]; /* Adresse 14 */
```

```
f[3] = 300;
```

14	f[0]=15
16	f[1]=3
18	f[2]=4
20	f[3]=16
22	f[4]=0

- EA = 14 + 3*2 = 20

[8] © Robert Tolksdorf, Berlin

Adressierungsart: Register indirekt mit Index

- Verwendung:
Schreiben bzw. Lesen von Elementen eines Arrays;
`long int f[100]; /* f hat die Adresse 1246 */`
`register long int *p; /* p liegt im Register %ebx */`
`register long int i; /* i liegt im Register %ecx */`
`p = &f[0];`
`p[i] = 50;`
`f[1] = p[i];`
- ergibt compiliert:
`movl $1246, %ebx`
`movl $50, (%ebx,%ecx,4)`
`movl (%ebx,%ecx,4), 1250`

[9] © Robert Tolksdorf, Berlin

Adressierungsart: Register indirekt mit Offset

- EA= Basisadresse + Offset (Displacement)
`struct rec { short int x; long int i;`
`short int y; short int z; };`
`struct rec p;`
`struct rec s; /* Adresse 14 */`
`p = &s.y`
`*p= 30;`

14	s.x=13
16	s.i=100000
18	
20	s.y=25
22	s.z=0
- EA= 14 + 6 = 20

[10] © Robert Tolksdorf, Berlin

Adressierungsart: Register indirekt mit Offset

- Verwendung:
Schreiben/Lesen von Record-Elementen über Pointer
`struct rec { long int x; char c; short int s; };`
`struct rec r; /* r hat die Adresse 3456 */`
`register struct rec *p; /* p liegt im Register %eax */`
`p = &r;`
`p->c = 'A';`
`r.s = p->s;`
- ergibt compiliert:
`movl $3456, %eax`
`movb $65, 4(%eax)`
`movw 6(%eax), 3462 /* =3456+6; mit Alignment */`

[11] © Robert Tolksdorf, Berlin

Adressierungsart: Register indirekt mit Index und Offset

- EA = Basisadresse + Index * Scale-Faktor + Offset
- Verwendung:
Schreiben/Lesen von Record-Elementen eines Arrays
`struct rec { long int x; char c; short s; };`
`struct rec f[100]; /* f hat die Adresse 452 */`
`register struct rec *p; /* p liegt im Register %ecx */`
`register long int i; /* i liegt im Register %eax */`
`p = &f[0];`
`p[i].c = 'A';`
`f[2].s = p[i].s;`
- ergibt compiliert:
`movl $452, %ecx;`
`movb $65, 4(%ecx,%eax,8)`
`movw 6(%ecx,%eax,8), 474 /* =452+2*8+6; mit Alignment */`

[12] © Robert Tolksdorf, Berlin

Adressierungsarten

- nicht alle CPUs (RISC!) können alle Adressierungsarten
- Ersatz (Beispiele):
 - Direct Address:
`movl 12345678, %eax`
↓
`movl $12345678, %eax`
`movl (%eax), %eax`
 - Memory Indirect:
`movl (12345678), %eax`
↓
`movl $12345678, %eax`
`movl (%eax), %eax`
`movl (%eax), %eax`
 - Register Indirect with Index and Displacement:
`movl 123(%eax,%ebx,1), %eax`
↓
`leal 123(%eax), %eax`
`movl (%eax,%ebx,1), %eax`

[13] © Robert Tolksdorf, Berlin

Adressierungsarten

- nicht alle Adressierungsarten mit allen Registern möglich
- nicht alle Kombinationen von Adressierungsarten (Ziel und Quelle) möglich (z.B. `movl 1232, 432`)
- Einschränkungen für die möglichen Werte bzw. die Größe von Scale-Faktor und Displacement
 - Scale-Faktor darf vielleicht nur 1, 2, 4 oder 8 betragen
 - Displacement häufig < 256
- Eventuell eigene Adressberechnung bei komplizierteren Strukturen notwendig

[14] © Robert Tolksdorf, Berlin

Adressierungsarten

- Synonyme (Beispiele):

<code>movl %eax, -(%esp)</code>	<code>pushl %eax</code>
<code>movl (%esp)+, %eax</code>	<code>popl %eax</code>
<code>movl %eflags, -(%esp)</code>	<code>pushfl</code>
<code>movl (%esp)+, %eflags</code>	<code>popfl</code>
<code>movl \$1234, %eip</code>	<code>jmp 1234</code>
<code>movl (%esp)+, %eip</code>	<code>ret</code>

[15] © Robert Tolksdorf, Berlin