

Tilldelningar: `a = b;`

Tilldelningar görs alltid via något register, grundregel, load/store-operationer anpassade efter datatyp....

Exempel:

Följande deklarationer är gjorda på global nivå, visa hur deklarationerna uttrycks i assemblyspråk.

```
char    c1,c2;
short   s1,s2;
int      i1,i2;
```

visa också hur följande tilldelningssatser kan kodas i ARM-v6 assemblyspråk:

```
c1 = c2;
s1 = s2;
i1 = i2;
```

Vi löser på tavlan...

```
@ c1 = c2;
    LDR    R3,=c2
    LDRB   R3,[R3]
    LDR    R2,=c1
    STRB   R3,[R2]

@ s1=s2;
    LDR    R3,=s2
    LDRH   R3,[R3]
    LDR    R2,=s1
    STRH   R3,[R2]

@ i1=i2;
    LDR    R3,=i2
    LDR    R3,[R3]
@ Alternativt
    LDR    R3,i2

    LDR    R2,=i1
    STR    R3,[R2]

.ALIGN
i1:    .SPACE    4
i2:    .SPACE    4
c1:    .SPACE    1
c2:    .SPACE    1
s1:    .SPACE    2
s2:    .SPACE    2
.GLOBL    i1,i2,c1,c2,s1,s2
```

Exempel:

```
char f(void);
```

```
int  a,b;
```

Visa en kodsekvens, i ARM assemblerspråk, som evaluerar följande uttrycks värde till register R0.

```
f() + a * ~(b & 4);
```

Vi löser på tavlan...

Anm: Man kan använda den alternativa adresseringsmetoden:

```
LDR    R3, variabel
```

i stället för sekvensen:

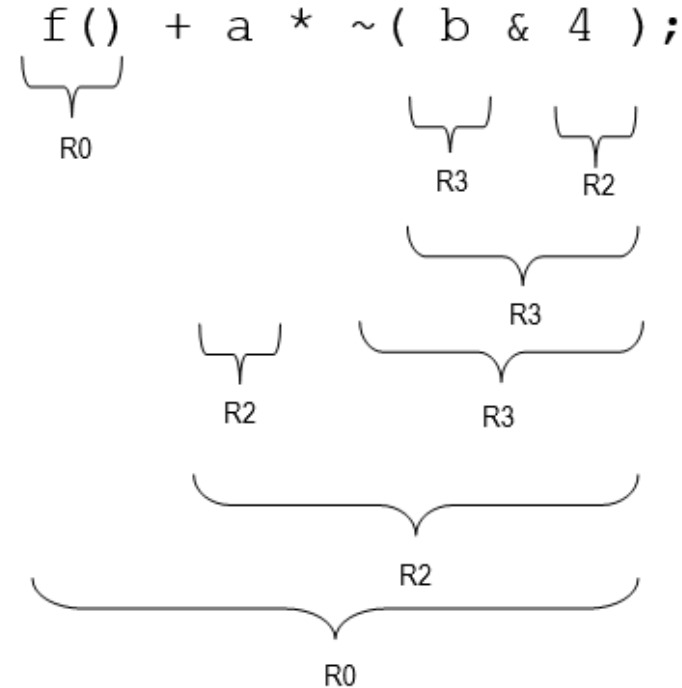
```
LDR    R3,=variabel
```

LDR R3, [R3]

Detta gäller dock INTE för BYTE och HALFWORD, dvs:

~~LDRH Rx, variabel~~~~LDRB Rx, variabel~~

finns inte!



BL	f
LDR	R3, b
MOV	R2, #4
AND	R3, R3, R2
MVN	R3, R3
LDR	R2, a
MUL	R2, R2, R3
ADD	R0, R0, R2