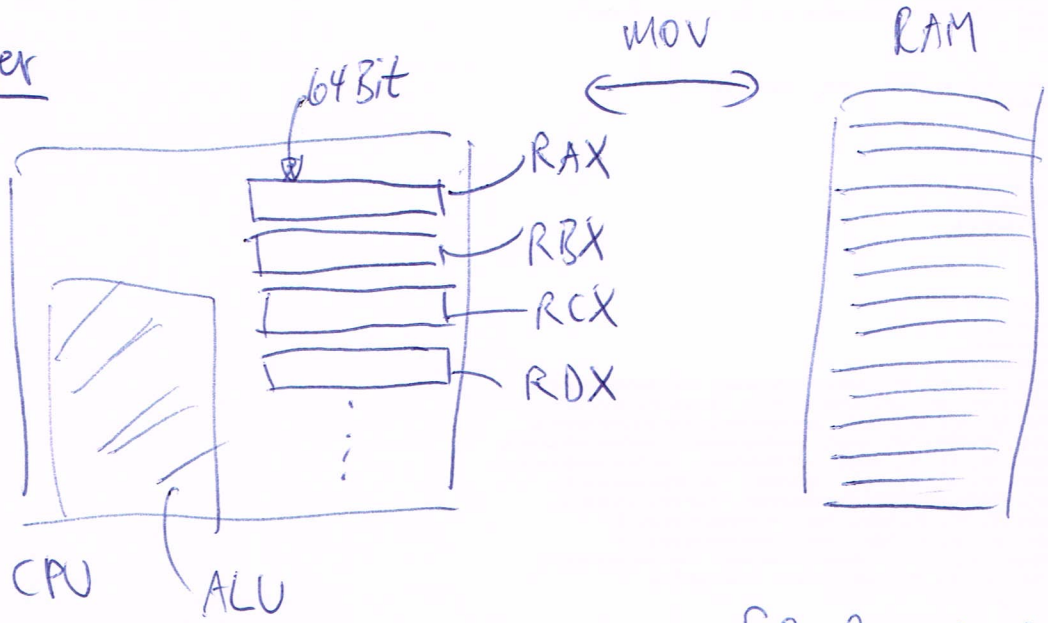
Assembler: x86_64

16.04.25
113

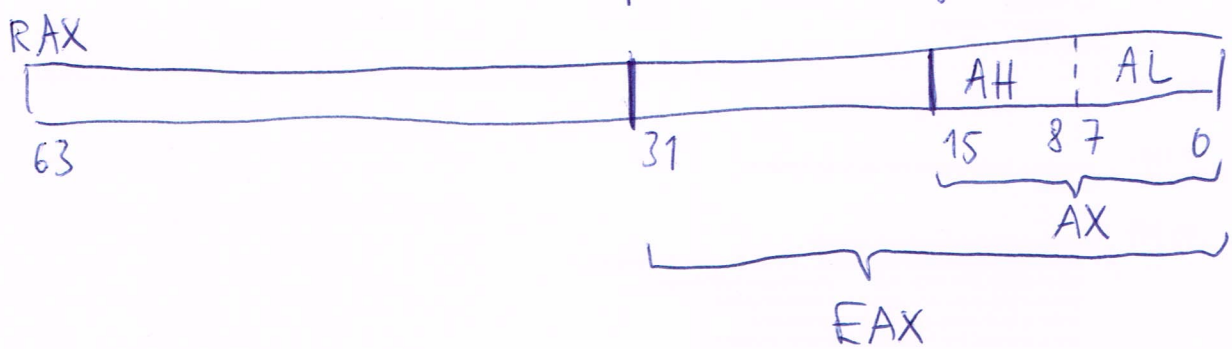
CPU: versteht Maschinensprache

Assembler (als Sprache): lesbare Repräsentation

Register



Transfer RAM → Reg.: MOV ~~RAX~~, [0x0000ABCD]
(kopiert 64 Bit Integer)



8086 CPU: AX, BX, CX, DX
80386 CPU: EAX, EBX, ECX, EDX
x86_64 RAX, RBX, RCX, RDX

Rechnen: Zwei Ints addieren

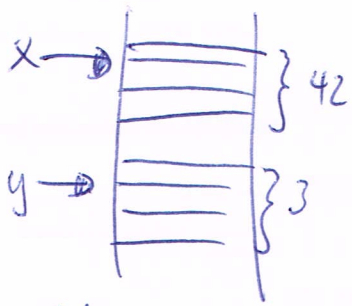
```
mov eax, 0b1000.0110.0010.1001
mov ebx, 0b0000.0101.1111.1011
                1 1 1 1 1 1 1
add eax, ebx    ; entspr.  eax = eax + ebx
                ;          eax += ebx
```

⇒ eax: [1000.1100.0010.0100]

Zwei Int-Variablen (aus RAM) addieren

C:
```
int x = 42;
int y = 3;
int z = x+y;
```

Assemb:




```
mov eax, [....] ; Adresse von x
mov ebx, [...]  ; Adresse von y
add eax, ebx
```


```
mov [...], eax ; Adresse von z
```

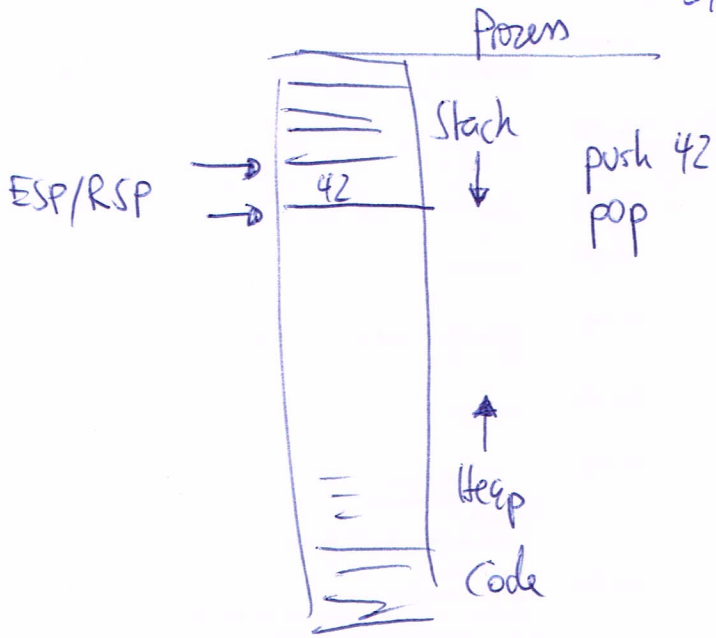
Dank CISC auch so:
```
mov eax, [...] ; Adr. von x
add eax, [...] ; Adr. von y
```

manches geht nicht:
```
add [...], [...]  ; zwei Adressen
```
↯

Beim Int-Addieren: EAX, EBX (32-bittig)
auch auf 64-Bit-CPU!

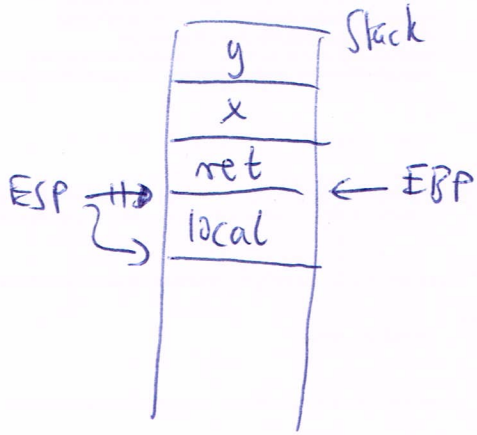
Standard-Register

EAX	RAX	
EBX	RBX	
ECX	RCX	
EDX	RDX	
EDI	RDI	Index-Reg.
ESI	RSI	
ESP	RSP	Stack Ptr.
EBP	RBP	Base Ptr.



[RSP+4]

```
Funktion sum(int):
    int test (int x, int y) {
        int local;
        local = x+y;
        print...
    }
```



[EBP-4]: local
[EBP+4]: x
[EBP+8]: y