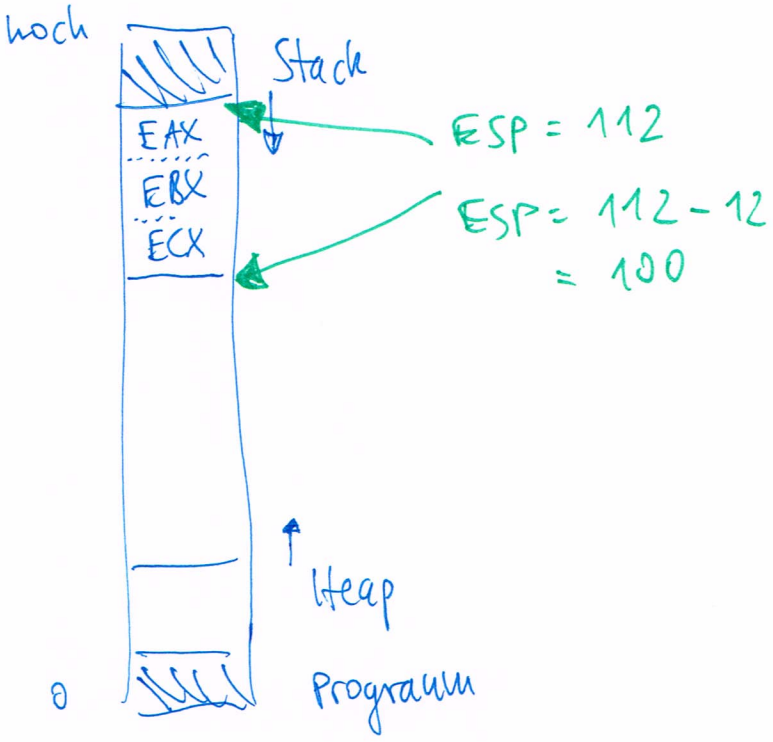
RA 24.10.20
1/3



		Quelle ziel
mov eax, eax	89 c0	11 000 000
mov eax, ebx	89 d8	11 011 000
mov eax, ecx	89 c8	11 001 000
mov eax, edx	89 d0	11 010 000
eax, esi	89 f0	11 110 000
eax, edi	89 f8	11 111 000
eax, ebp	89 e8	11 101 000
eax, esp	89 e0	11 100 000
mov ecx, edx	89 d1	11 010 001

eax = 000
ebx = 011
ecx = 001
edx = 010
esi = 110
edi = 111
ebp = 101
esp = 100

RA 24.10. 3/3

3.4 Lösungshinweise zu den Lernkontrollfragen

1. Übergabe in Registern vs. Übergabe auf dem Stack
   a. Vorteil Register: schneller Mechanismus, kein Speicherzugriff.
   b. Nachteile Register: Anzahl der Register (und damit der Argumente) ist begrenzt, Implementierung rekursiver Funktionen ist damit schwieriger; Register nehmen nur 32-Bit-Werte auf.
   c. Vorteil Stack: Beliebig viele und beliebig komplexe Argumente (z. B. ganze Datenstrukturen) möglich.
   d. Nachteil Stack: Speicherzugriffe nötig, Funktionsaufruf wird dadurch langsamer.
2. Die Antwort ist 100: Der Stack wächst „nach unten", also zu den kleineren Adressen hin.
3. Das lässt sich mit `jmp eax` erreichen: Der `jmp`-Befehl springt an die angegebene Adresse, und das bedeutet (implizit), diesen Wert ins Register EIP zu schreiben.
4. Die erste Frage ist hier, ob EIP *vor* oder *nach* Ausführung des Befehls gemeint ist, denn der Wert ändert sich ja während der Ausführung. Wir nehmen an, dass der Wert *nach* der Ausführung gemeint ist, also die Adresse, an der der folgende Befehl steht.

   Hier muss man eine Assembler-Funktion schreiben, die man z. B. `get_eip` nennen könnte: Sie wird mit `call` angesprungen und liest dann mit pop die Rücksprungadresse vom Stack. Diese Rücksprungadresse ist der Wert von EIP (nach Rückkehr aus der Funktion). Die Lösung ist also:

```
call get_eip
...
get_eip: pop eax     ; Rücksprungadresse auslesen
         push eax    ; und zurück schreiben
         ret
```