Übung 9 – Lösung

C	Assembler

int function1(int par1, short par2, char par3)
 	
{
 	
return par1 + par2 + par3;
 	
}
 	
int main()
 	
{ int a = 1; return
 	
function1(a, 1, '0') + 1;
 	
}
	0: push ebp
	
	1: mov ebp,esp
	
	3: sub esp,0x0
	
	9: mov eax,DWORD PTR [ebp+0x8]
	
	c: movsx ecx,WORD PTR [ebp+0xc]
	
	10: add eax,ecx
	
	12: movsx ecx,BYTE PTR [ebp+0x10]
	
	16: add eax,ecx
	
	18: leave
	
	19: ret
	
	1a: push ebp
	
	1b: mov ebp,esp
	
	1d: sub esp,0x4
	
	23: mov eax,0x1
	
	28: mov DWORD PTR [ebp-0x4],eax
	
	2b: mov eax,0x30
	30: push eax
	
	31: mov eax,0x1

	36: push eax
	37: mov eax,DWORD PTR [ebp-0x4] 3a: push eax 3b: call 0x0 40: add esp,0xc 43: inc eax 44: leave 45: ret

Gegeben ist ein C-Programm und der von einem Compiler (Befehle: tcc -m32 -nostdlib -Wl,-Ttext,0x0 -Wl,--oformat,binary -static uebung11.c -o uebung11 objdump -b binary -mi386 -M intel -D uebung11 > uebung11.s) erzeugte x86 32bit Assembler Code (Tipp: https://c9x.me/x86/):

1. Erörtern Sie die Bedeutung der folgenden Anweisungen für die Programmausführung, sowie deren Einfluss auf den Stack:

   - push ebp — Legt den aktuellen Basepointer auf dem Stack ab (Stackpointer -= 4)

     mov ebp,esp — Überschreibt den Basepointer mit dem Stackpointer
     Bedeutung für Programmausführung: Es wird ein neuer Stackframe für eine Funktion erstellt

   - call

     Programmausführung:
     Ein Unterprogramm wird aufgerufen
     Stack: Adresse des nach der Ausführung des Unterprogramms zu verarbeitendem Befehl wird auf den Stack gelegt (Stackpointer -= 4)

- leave

  Gleichbedeutend mit mov esp,ebp pop ebp

  Stack: Der vom aktuellen Unterprogramm auf dem Stack genutzte Speicherplatz wird freigegeben. Der Stackpointer zeigt auf die Rücksprungadresse.
  Programmausführung: Der aktuelle Stackframe wird auf den der Aufruferfunktion gesetzt.

- Ret

  Stack: Lädt die Rücksprungadresse vom Stack in das IPRegister (Stackpointer +=4)

  Programmausführung: Unterprogramm wird verlassen und die Programmausführung wird an der durch den Call-Befehl auf dem Stack abgelegten Adresse fortgesetzt

2. Erstellen Sie eine Skizze des Programmstacks, die für die Funktionen „main“ und „function1“ die Position des gesicherten „eip“, des gesicherten „ebp“, der lokalen Variablen, der übergebenen Parameter, sowie den aktuellen Wert des „esp“ wiederspiegelt. Gehen Sie dabei davon aus, dass die Rücksprungadresse der Funktion „main“ an Position „0xfffffffc“ im Speicher liegt (bezeichnen Sie diese als EIP<_start>) und die Programmausführung in Zeile 16 des Assembler-Codes angehalten wurde.

FFFFFFFC	EIP<_start>	
FFFFFFF8	EBP<_start>	
FFFFFFF4	0x1 (a)	
FFFFFFF0	0x30	
FFFFFFEC	0x1	
FFFFFFE8	0x1	
FFFFFFE4	0x40 (EIP<main>)	
FFFFFFE0	0xFFFFFFFF8 (EBP<main>)	EBP|ESP

3. Wo befindet sich im gezeigten Beispiel der jeweilige Rückgabewert der

Funktionen
Im EAX-Register

4. Erläutern sie Anhand des gezeigten Codes, wie lokale Variablen einer C-Funktion in Assembler abgebildet werden. 1d: sub esp,0x4 23: mov eax,0x1 28: mov DWORD PTR [ebp-0x4],eax Sie werden auf dem Stack abgelegt

5. Erläutern sie Anhand des gezeigten Codes, wie die Übergabe von Funktionsparametern in Assembler realisiert wird.
   30: push eax
   31: mov eax,0x1
   36: push eax
   37: mov eax,DWORD PTR [ebp-0x4]
   3a: push eax
   3b: call 0x0

   Sie werden in umgekehrter Reihenfolge auf dem Stack abgelegt

6. Erläutern sie Anhand des gezeigten Codes, wie der Zugriff auf Funktionsparameter in Assembler realisiert wird und vervollständigen Sie den nachfolgenden Befehl so, dass der erste Funktionsparameter in Form eines 32bit Wertes in das Register „eax“ abgelegt wird.

   Der Zugriff erfolgt abhängig vom Inhalt des EBP-Registers. Auf diesen wird ein Offset addiert, das die Position des Parameterwertes angibt.

   mov eax, DWORD PTR [ebp+0x8]

7.
   a) Erörtern sie den folgenden Assemblercode

```
sub esp,0x0 mov eax,DWORD
PTR [ebp+0x8] movsx ecx,WORD
PTR [ebp+0xc] add eax,ecx
movsx ecx,BYTE PTR [ebp+0x10]
add eax,ecx
```

   parameter1 + parameter2 + parameter3

b) Welchem Abschnitt des C-Codes entspricht der in a) gezeigte Code?

```
return par1 + par2 + par3;
```