0x05 – Zeiger/Pointer – Hintergrund Stack

Push und Pop auf einem (rückwärts wachsendem) Stack (Prinzip, 16 Bit):

Die Variable (=Register) SP (stack pointer) bezeichnet eine Position im Speicher, an der Daten auf dem Stack abgelegt (push, SP wächst rückwärts) bzw. von dort wiedergeholt (pop, SP steigt) werden. Die Variable IP (instruction pointer) bezeichnet die nächste Position im Programm, an der ein Befehl ausgeführt wird. A ist eine beliebige Variable.

Asm-Programm	Befehl	IP danach	SP danach	A danach
		1	0x2304	
1	A=0x1234	2	-	0x1234
2	push A	3	0x2302	-
3	A=0x5678	4	-	0x5678
4	push A	5	0x2300	-
5	A=0x9abc	6		0x9abc
6	pop A	7	0x2302	0x5678
7	pop A	8	0x2304	0x1234
8	...			

Speicher 0x2300-0x2307	+0x00	+0x02	+0x04	+0x06
"	0x0000	0x0000	0x0000	0x0000
"	0x0000	0x0000	0x0000	0x0000
"	0x0000	0x0000	0x1234	0x0000
	0x0000	0x0000	0x1234	0x0000
	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x5678	0x1234	0x0000

0x05 – Zeiger/Pointer – Hintergrund lokale Variablen

Lokale Variablen auf dem Stack (Prinzip 16 Bit):

Die Variable (=Register) BP bezeichnet eine Position im Speicher, zu der relativ lokale Variablen abgelegt werden. BP wird zu Beginn mit SP initialisiert und danach wächst SP (-=), damit künftige Calls die Daten nicht zerstören. Das muss am Ende wieder korrigiert werden (+=). Die Speicheradressen von n1 und n2 sind hier: 0x2304 und 0x2302.

Rückgabewerte werden z.B. in der Variablen A (=Register) zurückgegeben.

	C++ Programm	Asm-Programm
1	{	BP = SP (0x2306)
2		SP -= 0x06
3	int n1 = 0x1234	[BP-2] = n1
4	int n2 = 0x5678	[BP-4] = n2
5	return 0xaabb	A = 0xaabb
		SP += 0x06
6	}	ret

Speicher 0x2300-0x2307	+0x00	+0x02	+0x04	+0x06
"	0x0000	0x0000	0x0000	0x0000
"	0x0000	0x0000	0x0000	0x0000
"	0x0000	0x0000	0x1234	0x0000
	0x0000	0x5678	0x1234	0x0000
	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x5678	0x1234	0x0000
"	0x0000	0x1234	0x1234	0x0000

0x05 – Zeiger/Pointer – Hintergrund lokale Variablen

Lokale Variablen auf dem Stack (konkret, 32 bzw. 64 Bit):

C++ vs. Assembler

```
// Argument x und lokale Variablen auf dem Stack
int f(int x) {
    int n1=0x11223344, n2=0x55667788, n3=0x6699cc
    g();
    return 0x12345678;
}
```

```
0x10b759f60 <+0>:  55                    pushq  %rbp
0x10b759f61 <+1>:  48 89 e5              movq   %rsp, %rbp
0x10b759f64 <+4>:  48 83 ec 10           subq   $0x10, %rsp
```

```
        int n1=0x11223344, n2=0x55667788, n3=0x6699ccff;

0x10b759f6b <+11>: c7 45 f8 44 33 22 11  movl   $0x11223344, -0x8(%rbp)
0x10b759f72 <+18>: c7 45 f4 88 77 66 55  movl   $0x55667788, -0xc(%rbp)
0x10b759f79 <+25>: c7 45 f0 ff cc 99 66  movl   $0x6699ccff, -0x10(%rbp)
```

```
0x10b759f85 <+37>: b8 78 56 34 12        movl   $0x12345678, %eax
0x10b759f8a <+42>: 48 83 c4 10           addq   $0x10, %rsp
0x10b759f8e <+46>: 5d                    popq   %rbp
0x10b759f8f <+47>: c3                    retq
```

0x05 – Zeiger/Pointer – Hintergrund

Der Code zeigt ausserdem, dass lokale Variablen

- eine absolute Position im Speicher haben (Speicheradresse), auch wenn sie relativ angesprochen werden.
- einen zufälligen Wert besitzen (eben der an ihrer Position auf dem Stack), wenn sie nicht initialisiert werden.
- keine Typinformation im Speicher besitzen – es sind immer nur Bytes, die dem Typ entsprechend interpretiert werden.
- hintereinander liegen, wenngleich das nicht sein muss.
- illegal verändert werden können, wenn man nur die Speicheradresse einer kennt.

Zeiger bzw. Pointer sind i.d.R. Speicheradressen auf andere Variablen! Über Zeiger kann man diese lesen und auch verändern.

0x05 – Übungen

’Copper View’

Variablen über Zeiger modifizieren.

- Legen Sie eine ’int’- und eine ’float’-Variable an und initialisieren Sie sie mit beliebigen Werten.
- Legen Sie zwei Zeiger an und initialisieren Sie sie so, dass sie jeweils auf diese beiden Variablen zeigen.
- Verändern Sie die Werte der Variablen mit Hilfe der Zeiger.
- Geben Sie jeweils den Zeiger, den Wert auf den er zeigt und die zugehörige Variable aus.

Erweiterung:

- Legen Sie einen Zeiger auf den Zeiger auf ’int’ an, initialisieren Sie ihn mit der Adresse Ihres ’int’-Zeigers und geben Sie den Wert der ’int’-Variablen hierüber aus.

0x05 – Übungen

’Kengate’

Swap.

- Schreiben Sie eine Funktion ’swap’, die zwei übergebene ’double’-Zahlen tauscht. Implementieren Sie die Funktionen einmal über Zeiger und einmal über Referenzen.

Erweiterung:

- Schreiben Sie eine weitere Funktion ’swap_ptr’, die zwei übergebene ’double’-Zeiger tauscht (Zeiger und Referenzen).
- Testen Sie Ihren Code entsprechend.

0x05 – Übungen

’McAllen Spring’

Illegal – beachten Sie auch das Snippet zum Thema…

- Versuchen Sie, durch Verwendung von Zeigern einzelne Bytes von lokalen ’int’-Variablen zu manipulieren. Nutzen Sie dafür casts in ’char*’.
- Verändern Sie einzelne Bytes eines Texts der Form: char* p=”huhu”;
  Was passiert?

Erweiterung:

- Versuchen Sie, über Zeiger-Manipulationen lokale Variablen gezielt auf dem Stack zu verändern, ohne diese direkt zu adressieren.
  Tipp: Ermitteln Sie zunächst die Adresse einer lokalen Variablen. Die anderen liegen ’in der Nähe’.

0x05 – Übungen

Hausübung

- Überlegen Sie sich, wofür Sie Zeiger verwenden könnten? Gibt es Anwendungsfälle, in denen Referenzen nicht möglich sind, aber Zeiger durchaus?

Selbstkontrolle

- Ich habe alle Codes und Übungsthemen verstanden.
- Ich weiß, was ein Zeiger ist.
- Ich kann Zeiger initialisieren, verwenden und mit ihnen rechnen.
- Ich kann sowohl Zeiger als auch Referenzen für ’call-by-reference’ verwenden.
- Ich kenne spezielle Ausdrücke wie ’nullptr’ oder ’void*’.
- Ich weiß, was Felder und Indexzugriffe mit Zeigern zu tun haben.