

Application Binary Interface

Beschreibt Schnittstelle auf Maschinencode-Ebene:

- ▶ Datentypen (Größe, Alignment, Layout)
- ▶ Calling Convention
 - ▶ Ort der Funktionsparameter
 - ▶ Ort der Rückgabewerte
 - ▶ Welche Register zu sichern sind oder überschrieben werden können
- ▶ Speicherstruktur
- ▶ Betriebssystemschnittstelle

Quiz: In welchen Registern liegt der...

C-Source	
1	long sub(long a, long b) {
2	return a - b;
3	}
(Dis-)Assembly	
1	mov rax,rdi
2	sub rax,rsi
3	ret

a
Handwritten annotations: An arrow points from 'a' to the 'rdi' operand in the first assembly instruction. Another arrow points from the 'rsi' operand in the second instruction down to the 'ret' instruction.

Rückgabewert

1.Parameter

2.Parameter

☐

eax

☐

rax

☐

rdi

☐

rsi

☐

rsi

☐

rax

☒

rax

☒

rdi

☒

rsi

Funktionsparameter

- ▶ Parameter: `rdi`, `rsi`
- ▶ Rückgabewert: `rax`, `rdx`

Quiz: Wie werden weitere Parameter übergeben?

C-Source

```
1 long add_sub_many(long a0,  
2     long a1, long a2,  
3     long a3, long a4,  
4     long a5, long a6,  
5     long a7) {  
6     return a0 - a1 + a2 - a3  
7         + a4 - a5 + a6 - a7;  
8 }
```

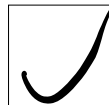
(Dis-)Assembly

```
1 sub    rdi,rsi  
2 add    rdi,rdx  
3 sub    rdi,rcx  
4 add    rdi,r8  
5 sub    rdi,r9  
6 mov    rax,rdi  
7 add    rax,qword ptr([rsp+0x8])  
8 sub    rax,qword ptr([rsp+0x10])  
9 ret
```

Stack



rdi, rsi, rdx, rcx, r8, r9



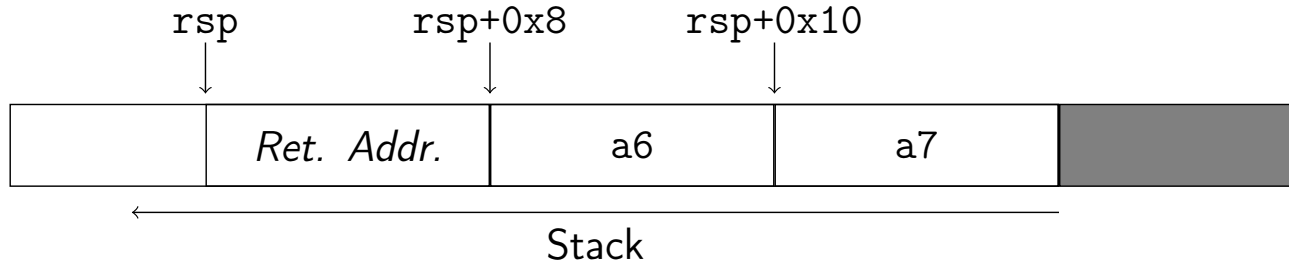
Ab Parameter 7 über den Stack



Ab Parameter 7 gibt es keine feste
Regelung mehr

Funktionsparameter

- ▶ Parameter: `rdi, rsi, rdx, rcx, r8, r9, Stack`
- ▶ Rückgabewert: `rax, rdx`



Registertypen

▶ Callee-Saved Register:

- ▶ Gehören aufrufender Funktion
- ▶ Müssen vom callee gesichert werden
- ▶ Nach Funktionsende unverändert
- ▶ `rbx`, `rbp`, `r12–r15`, `rsp`

▶ Temporäre/caller-saved Register:

- ▶ Gehören aufgerufener Funktion
- ▶ Dürfen frei verwendet werden
- ▶ Nach Funktionsende undefiniert
- ▶ `rax`, `rcx`, `rdx`, `rsi`, `rdi`, `r8–r11`

```
1 myfun:
2     push    rbx
3     mov     rbx, [rdi]
4     ...
5     pop     rbx
6     ret
```

Stack-Alignment

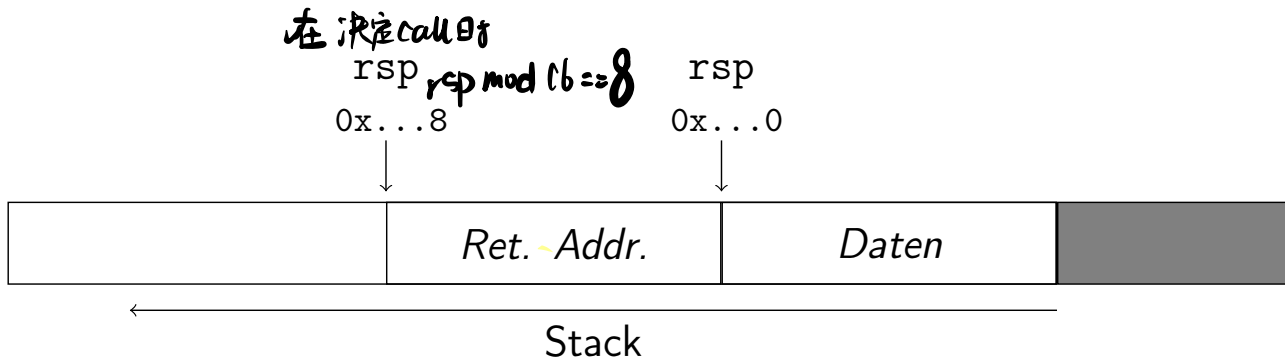
- ▶ Calling Convention stellt Anforderung an *Stack-Alignment*

- ▶ 16-Byte-Alignment vor einem Funktionsaufruf

$$\text{rsp} \% 16 == 0$$

- ▶ `call` legt weitere 8 Byte auf den Stack

$$\rightsquigarrow \text{rsp} \% 16 == 8$$



Stack-Alignment

```
1 myfun:  
2     push    rbx  
3     mov     rbx, [rdi]  
4     ...  
5     call    do_sth  
6     add     rax, rbx  
7     pop     rbx  
8     ret
```


Quiz: Wurde die Calling Convention eingehalten?

Signatur: long fun0(long x, long y)

```
1 fun0:  
2     mov rax, rdi  
3     imul rax, rdi  
4     mov r10, rax  
5     mov rax, rsi  
6     imul rax, rsi  
7     add rax, r10  
8     ret
```

☐

Ja

☐

Nein

Quiz: Wurde die Calling Convention eingehalten?

Signatur: long fun1(long x)

```
1 fun1:
2     mov r10, rdi
3     sar r10, 1
4     mov rdi, 1
5     mov rsi, 1
6     call fun0
7     mov rax, r10
8     ret
```

☐

Ja

☐

Nein

Quiz: Wurde die Calling Convention eingehalten?

Signatur: long fun2(long x)

```
1 fun2:
2     push rbp
3     mov rbp, rsp
4     sub rsp, 0x8
5     mov qword ptr [rbp-0x8], rdi
6     mov rax, qword ptr [rbp-0x8]
7     sar rax, 1
8     pop rbp
9     ret
```

☐

Ja

☐

Nein

Struct-Layout

Wie sind Felder eines Structs im Speicher angeordnet?

- ▶ Abhängig von Reihenfolge und Alignment der Felder
- ▶ Padding um Alignment der Felder sicherzustellen
- ▶ Alignment des structs ist das seines Felds mit dem größten Alignment
- ▶ Größe des structs ist ein Vielfaches davon → Padding am Ende

Quiz: Padding (1)

Welche Aussage bezüglich struct s ist korrekt?



```
1 struct s {  
2     char c;  
3     int i;  
4 };
```

☐

offsetof(struct s, i) == 1
sizeof(struct s) == 5

☐

offsetof(struct s, i) == 2
sizeof(struct s) == 6

☒

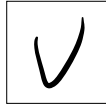
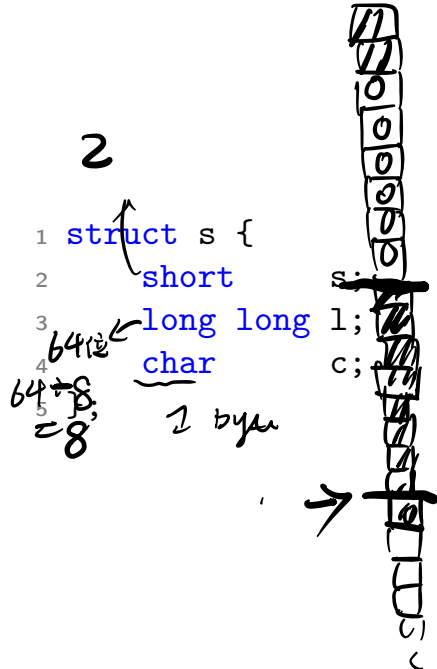
offsetof(struct s, i) == 4
sizeof(struct s) == 8

☐

offsetof(struct s, i) == 8
sizeof(struct s) == 16

Quiz: Padding (2)

Welche Aussage bezüglich struct s ist korrekt?



`offsetof(struct s, c) == 16`
`sizeof(struct s) == 24`



`offsetof(struct s, c) == 10`
`sizeof(struct s) == 11`



`offsetof(struct s, c) == 16`
`sizeof(struct s) == 17`



`offsetof(struct s, c) == 10`
`sizeof(struct s) == 16`

Structs als Funktionsparameter

Wie werden structs als Funktionsparameter übergeben?

- ▶ Größer als 16 Byte oder Felder die nicht aligned sind:
 - ▶ Übergabe via Stack
- ▶ Sonst:
 - ▶ Aufteilen in max. 2 Teile mit jeweils 64 Bit
 - ▶ Diese werden als Int-Parameter behandelt
 - ▶ Mehrere benachbarte Felder können zusammengefasst werden

Structs als Funktionsparameter – Beispiele

► 2 64-Bit Integer

```
1 void func(struct { uint64_t a; uint64_t b; } param);  
2 // a -> rdi, b -> rsi
```

► 2 32-Bit Integer

```
1 void func(struct { uint32_t a; uint32_t b; } param);  
2 // a -> rdi[31:0], b -> rdi[63:32]
```

► mehrere zusammengefasste Felder

```
1 void func(struct { uint16_t a; uint16_t b; uint8_t c; } param);  
2 // a -> rdi[15:0], b -> rdi[31:16], c -> rdi[39:32]
```

► struct größer als 16 Byte

```
1 void func(struct { uint64_t a; uint64_t b; uint64_t c; } param);  
2 // a -> [rsp], b -> [rsp + 8], c -> [rsp + 16]
```


Structs als Rückgabewerte

- ▶ Ort wird wie bei der Parameterübergabe bestimmt
- ▶ Register:
 - ▶ `rax` und `rdx` für die 64-Bit-Blöcke
- ▶ Stack:
 - ▶ *Caller* reserviert ausreichend Speicher
 - ▶ Pointer darauf wird in `rdi` "verdeckt" übergeben
 - ▶ Funktion gibt den Pointer in `rax` zurück

Structs als Rückgabewerte – Beispiele

```
1 struct ComputeRes { uint64_t a, b, c; };  
2 struct ComputeRes compute(int param);  
3 // verhält sich wie:  
4 struct ComputeRes* compute(struct ComputeRes* retval, int param);
```

Quiz: Structs als Rückgabewerte

Wo werden `s.a` und `s.b` zurückgegeben?

```
1 struct s {  
2     int a;  
3     long b;  
4 };  
5  
6 struct s foo(void);
```

☐

`rax, edx`

☐

`eax, rdx`

☐

`[rax], [rax + 4]`

☐

`[rax], [rax + 8]`

Calling Convention: Zusammenfassung

- ▶ Parameter: `rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`, Stack
- ▶ Rückgabewert: `rax`, `rdx` (bei 128-Bit)
- ▶ Callee-saved: `rbx`, `rbp`, `rsp`, `r12–r15`
 - ▶ Gleicher Wert nach Funktionsende, ggf. sichern und wiederherstellen
- ▶ Temporär/caller-saved: `rax`, `rcx`, `rdx`, `rsi`, `rdi`, `r8–r11`
 - ▶ Frei verwendbar, Wert nach Funktionsende unbekannt