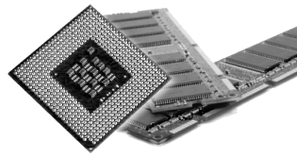
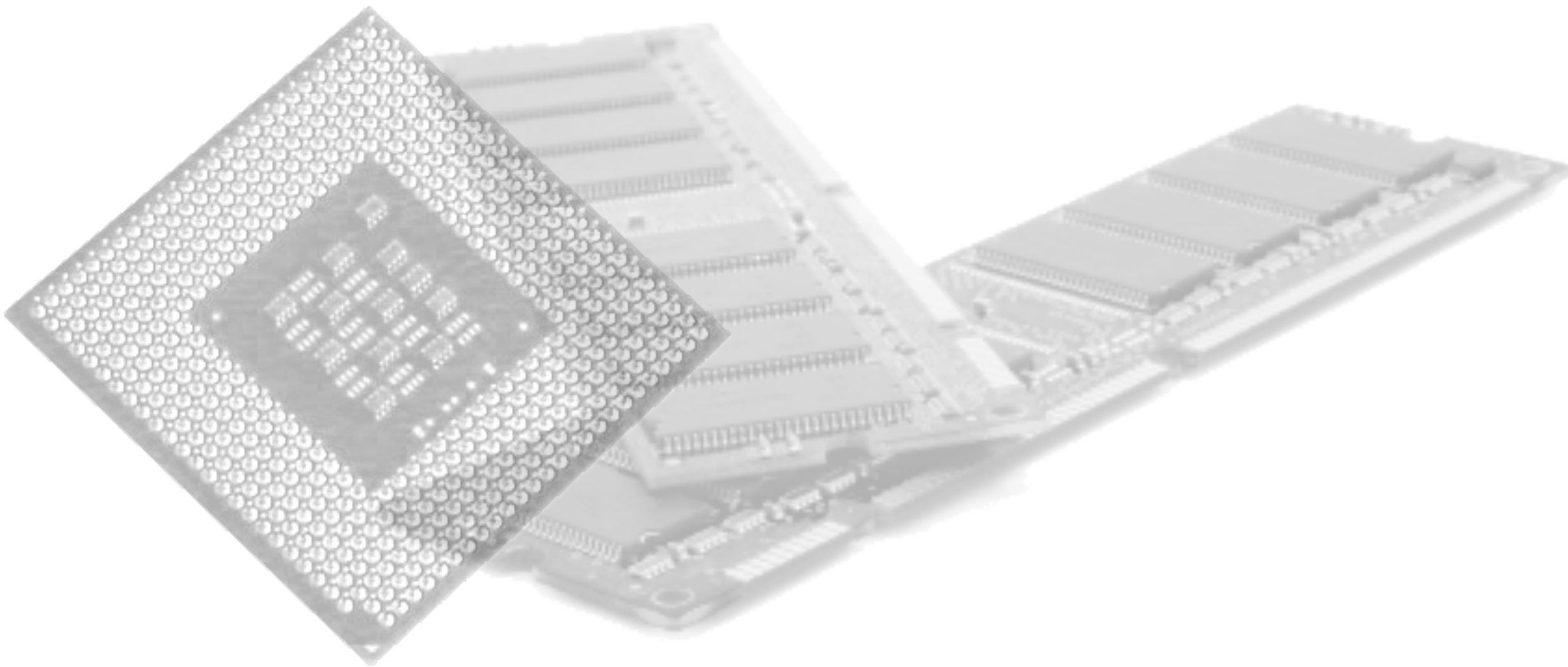
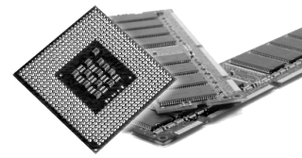


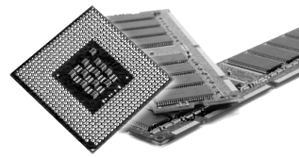
System Call





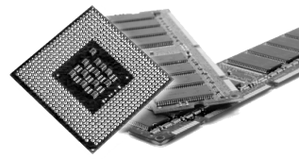
System Calls

- Funktionen des Kernel aufrufen
- z.B. zur Ein- und Ausgabe (diesmal ohne Hilfsmittel durch C-Code)
- Call Nummer in RAX
- Parameter in die Register
- syscall
- Return in RAX



syscall Mac

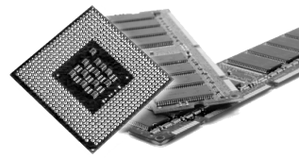
```
1   global main
2
3   section .data
4   msg:    db      "Hello, world!", 10, 0
5   .len:   equ     $-msg
6
7   section .text
8   main:
9       mov     rax, 0x2000004 ; write
10      mov     rdi, 1 ; stdout
11      mov     rsi, msg
12      mov     rdx, msg.len
13      syscall
14
15      mov     rax, 0x2000001 ; exit
16      mov     rdi, 0
17      syscall
18
```



syscall Mac

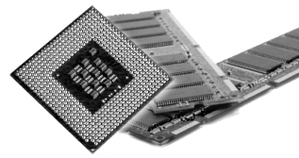
- rax: System Call Nr 4
- rdi: stdout
- rsi: text
- rdx: length
- syscall

```
1   global main
2
3   section .data
4   msg:     db        "Hello, world!", 10, 0
5   .len:    equ       $-msg
6
7   section .text
8   main:
9       mov       rax, 0x2000004 ; write
10      mov       rdi, 1 ; stdout
11      mov       rsi, msg
12      mov       rdx, msg.len
13      syscall
14
15      mov       rax, 0x2000001 ; exit
16      mov       rdi, 0
17      syscall
18
```



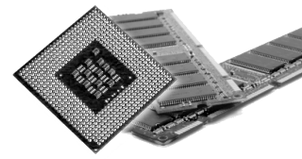
Länge des Strings

- len: equ $-msg
- $ => aktuelle Adresse
- $-msg => aktuelle Adresse - Adresse von msg => Länge von msg
- equ ersetzt jedes Vorkommen von len mit dem Integer-Wert



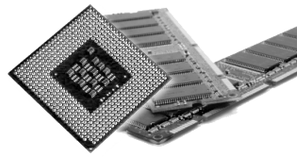
sys_write (Linux)

- Syntax: ssize_t sys_write(unsigned int fd, const char * buf, size_t count)

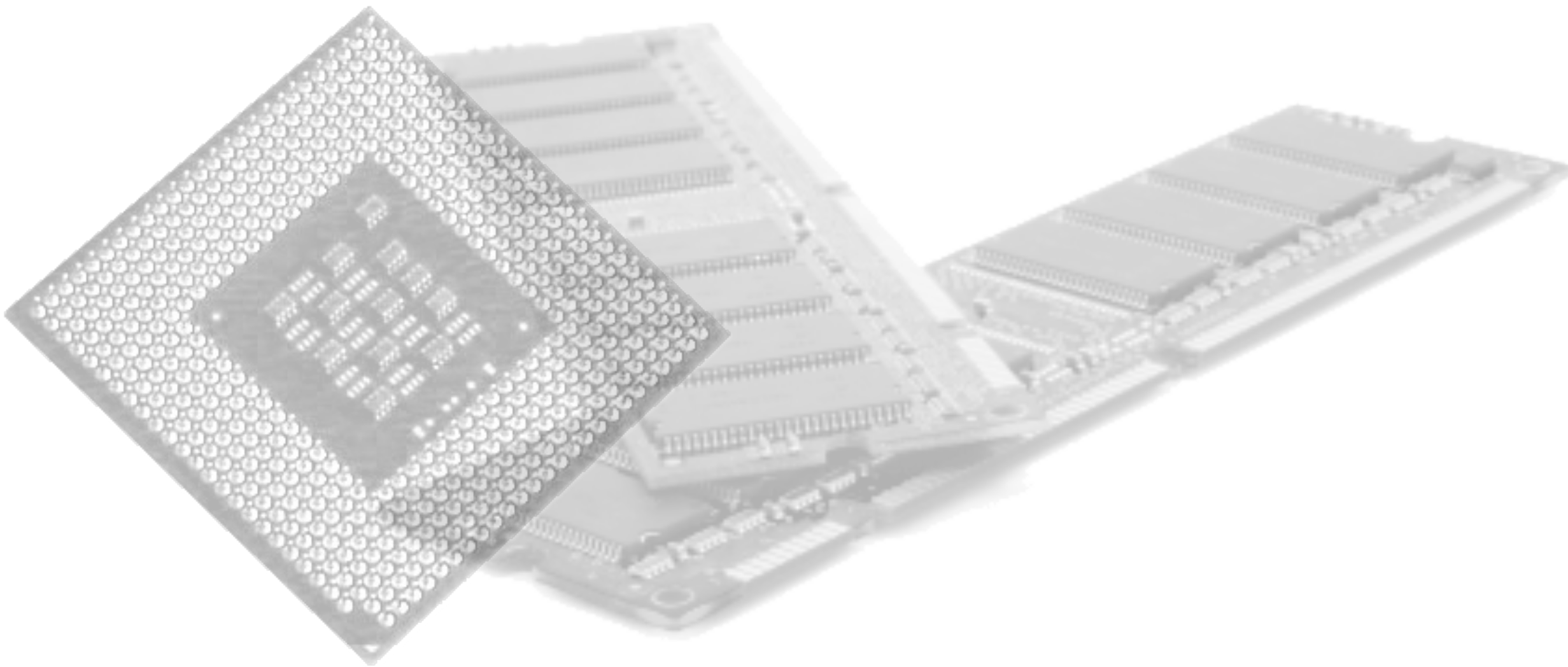


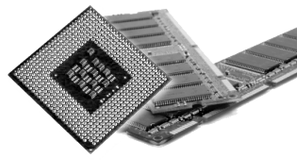
System Call Table (Linux)

rax	Name	Quelle	rbx	rcx	rdx
1	sys_exit	kernel/exit.c	int		
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs		
3	sys_read	fs/read_write.c	unsigned int	char *	size_t
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t
5	sys_open	fs/open.c	const char *	int	int
6	sys_close	fs/open.c	unsigned int		
…	…	…	…	…	…



Speicher und Strings





SASM

File Edit Build Debug Settings Help

Memory

Variable or expression	Value	Type
fmt	{0x48,0x61,0x6c,0x6c,0x6f,0x20,0x25,0x69,0x0,0x0}	Hex b 10 Address
Add variable...		Smart d Array size Address

*printf DEMO.asm | printf32.asm

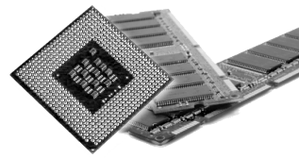
```
1   %include "io.inc"
2   section .data
3
4   zahl dd 12
5   fmt Dd "Hallo %i",0
6
7   section .text
8   global CMAIN
9   extern printf
10  CMAIN:
11      mov ebp, esp; for correct debugging
12      ;write your code here
13
14      push dword [zahl]
15      push fmt
16
17      call printf
18
19      mov esp, ebp
20
21      xor eax, eax
22      ret
```

Input

Output

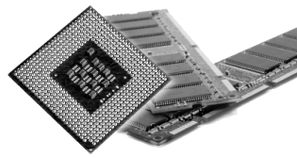
Registers

Register	Hex	Info
eax	0x8	8
ecx	0x753a49b3	1966754227
edx	0x753e1200	1967002112
ebx	0x393000	3747840
esp	0x61ff14	0x61ff14
ebp	0x61ff1c	0x61ff1c
esi	0x40126c	4199020
edi	0x40126c	4199020
eip	0x4013a2	0x4013a2 <main+18
eflags	0x246	[PF ZF IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x53	83
gs	0x2b	43



ASCII

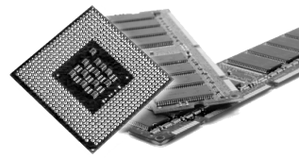
- American Standard Code for Information Interchange
- „Internationaler“ Standard zur Interpretation von Zeichen (der Tastatur)
- Nutzung von 7-Bit Speicher für Dezimalzahl
  - Enthält Steuerzeichen (NULL) am Anfang
  - Klein- und Großbuchstaben in alphabetischer Reihenfolge
  - Ziffern in natürlicher Reihenfolge



ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

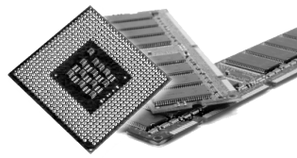


UTF-8

- Universal Character Set Transformation Format
- 8 Bit
- mit ASCII kompatibel

Beispiele für UTF-8 Kodierungen

Zeichen	Unicode	Unicode binär	UTF-8 binär	UTF-8 hexadezimal
Buchstabe *y*	U+0079	00000000 01111001	01111001	79
Buchstabe *ä*	U+00E4	00000000 11100100	11000011 10100100	C3 A4
Zeichen für eingetragene Marke ®	U+00AE	00000000 10101110	11000010 10101110	C2 AE
Eurozeichen €	U+20AC	00100000 10101100	11100010 10000010 10101100	E2 82 AC
Violinschlüssel 𝄞	U+1D11E	00000001 11010001 00011110	11110000 10011101 10000100 10011110	F0 9D 84 9E

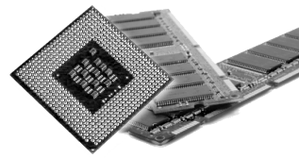


UTF-8

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Beispiele für UTF-8 Kodierungen

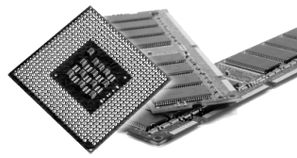
Zeichen	Unicode	Unicode binär	UTF-8 binär	UTF-8 hexadezimal
Buchstabe *y*	U+0079	00000000 01111001	**0**1111001	79
Buchstabe *ä*	U+00E4	00000000 11100100	**110**00011 **10**100100	C3 A4
Zeichen für eingetragene Marke ®	U+00AE	00000000 10101110	**110**00010 **10**101110	C2 AE
Eurozeichen €	U+20AC	00100000 10101100	**1110**0010 **10**000010 **10**101100	E2 82 AC
Violinschlüssel 𝄞	U+1D11E	00000001 11010001 00011110	**11110**000 **10**011101 **10**000100 **10**011110	F0 9D 84 9E



UTF-8

- **á** Unicode => U+00E1 (225 in Dezimal)
- 225 in Binär: **1110 0001**
- 110**00011** 10**100001** (C3 A1 in Hex)

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx



UTF-8

U+001A | Dec: 26

Unicode® character table

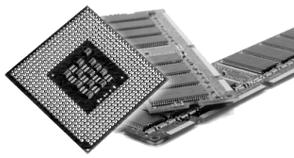
Character search

Example: heart

Most popular characters

❤ ✓ ☀ ★ ☂ ♞ ☯ ☭ ☢ € ☎ ∞ ❄ ♫ ₽

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0010	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0020		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0070	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	



SASM

File Edit Build Debug Settings Help

Memory

Variable or expression	Value	Type
fmt	{0x48,0x61,0x6c,0x6c,0x6f,0x20,0x25,0x69,0x0,0x0}	Hex b 10 Address

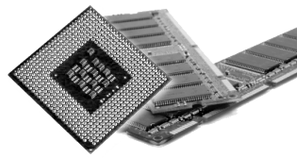
fmt = H a l l o % i

fmt = 0x48, 0x61, 0x6c, 0x6c, 0x6f, 0x20, 0x25, 0x69, 0x0, 0x0

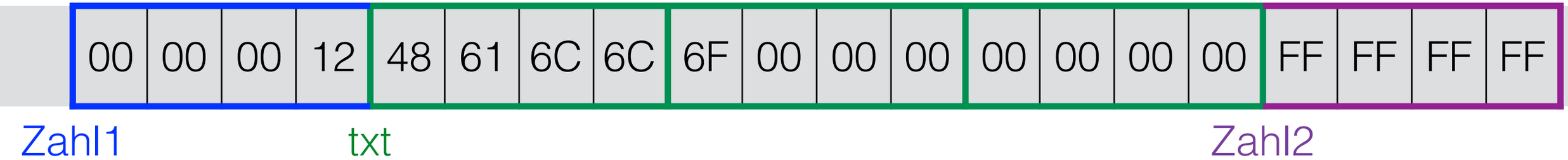
```
 4   zahl dd 12
 5   fmt Dd "Hallo %i",0
 6
 7   section .text
 8   global CMAIN
 9   extern printf
10   CMAIN:
11       mov ebp, esp; for correct debugging
12       ;write your code here
13
14       push dword [zahl]
15       push fmt
16
17       call printf
18
19       mov esp, ebp
20
21       xor eax, eax
22       ret
```

Output

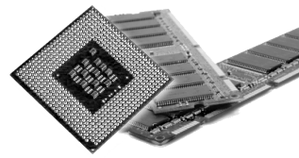
ecx	0x753a49b3	1966754227
edx	0x753e1200	1967002112
ebx	0x393000	3747840
esp	0x61ff14	0x61ff14
ebp	0x61ff1c	0x61ff1c
esi	0x40126c	4199020
edi	0x40126c	4199020
eip	0x4013a2	0x4013a2 <main+18
eflags	0x246	[PF ZF IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x53	83
gs	0x2b	43



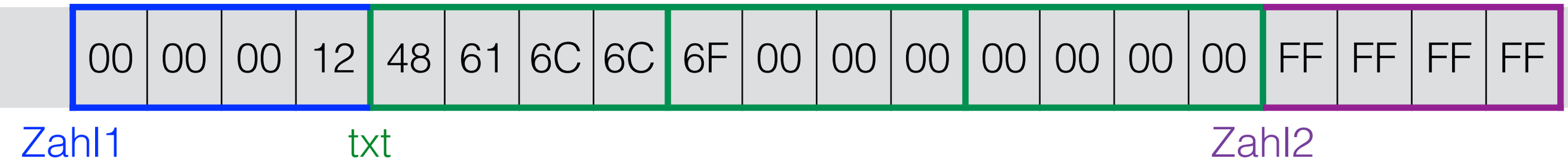
String im Speicher



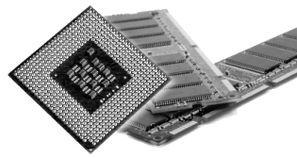
- Zahl1 DD 0x12
- txt DD “Hallo”, 0
- Zahl2 DD 0xFFFFFFFF



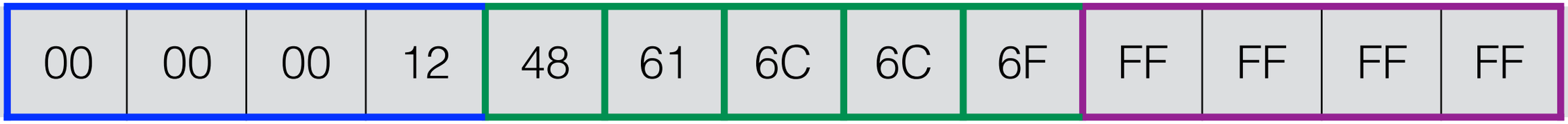
String im Speicher



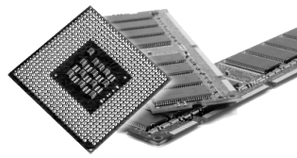
- Zahl1 steht für die Speicheradresse (es ist keine Größenangabe damit verknüpft)
- Mit **[Zahl1 + 4]** kann **txt** überschrieben werden!
- Bei printf wird als String die Adresse übergeben (nicht der Inhalt)
- PUSH txt



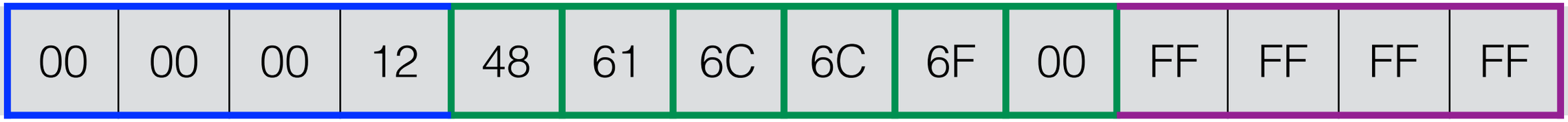
Byte String im Speicher



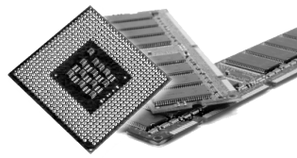
- Zahl1 DD 0x12
- txt DB “Hallo”
- Zahl2 DD 0xFFFFFFFF



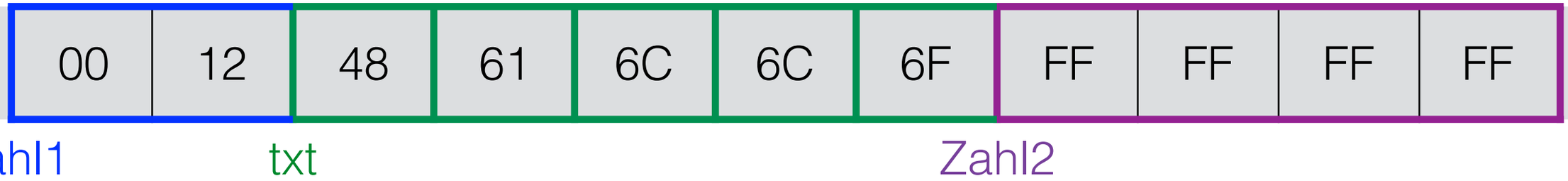
Byte String im Speicher



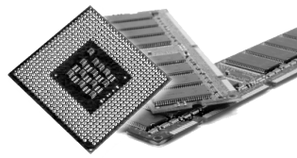
- Zahl1 DD 0x12
- txt DB “Hallo”,0
- Zahl2 DD 0xFFFFFFFF



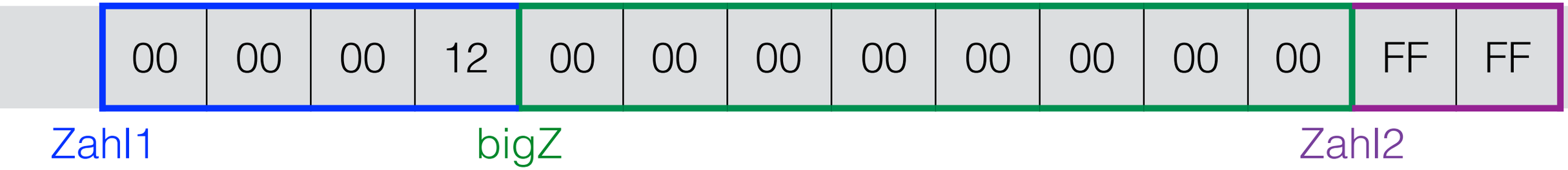
Falsche Größenangabe



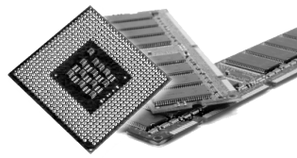
- Zahl1 DW 0x12
- PUSH **DWORD** Zahl1
- **ACHTUNG: ANFANG VON txt WIRD AUCH MIT GESCHRIEBEN**



Quad Word im Speicher



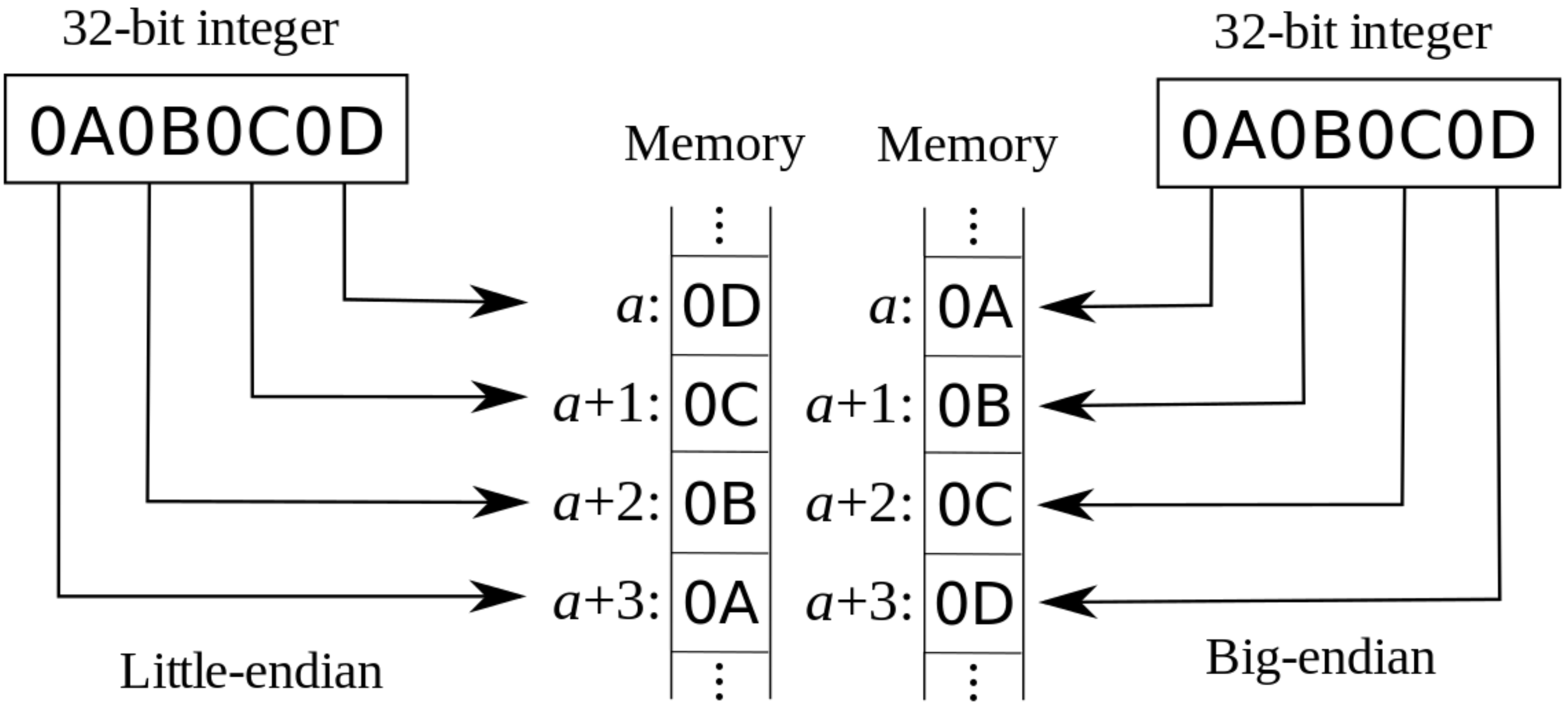
- Zahl1 DD 0x12
- bigZ DQ 0
- Zahl2 DW 0xFFFF



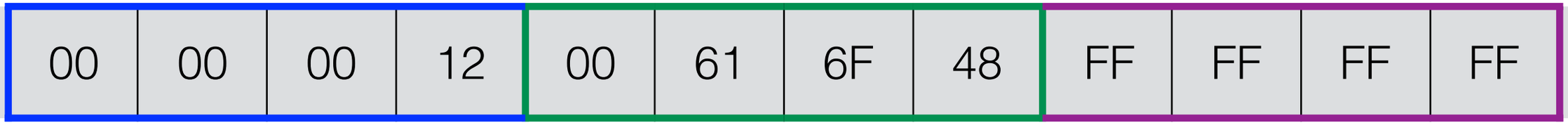
Quad Word im Speicher

- bigZ mit Daten befüllen in 32 Bit:
- mov [bigZ], eax
- mov [bigZ + 4], edx

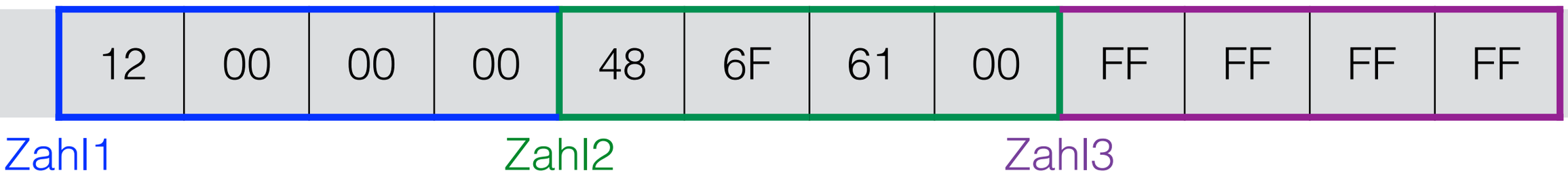
Big and Little Endian



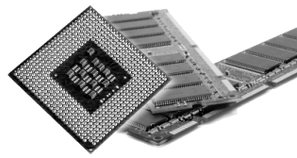
Big and Little Endian



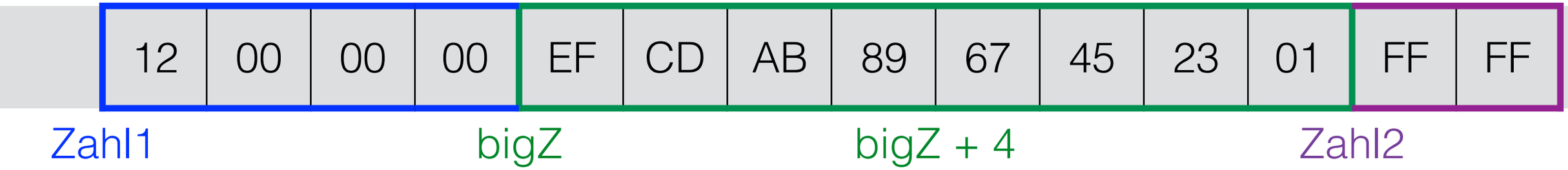
- Speicher mit Little Endian (x86):



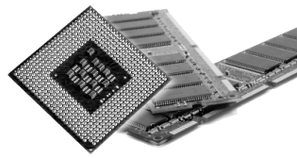
- Die Werte der Zahlen verändert sich nicht!



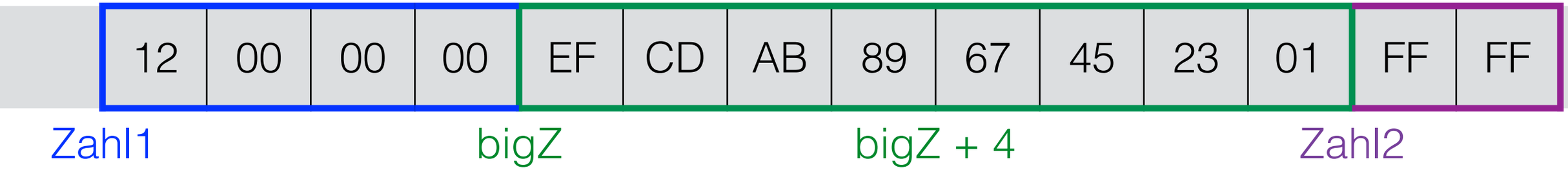
Quad Word (Little Endian) im Speicher



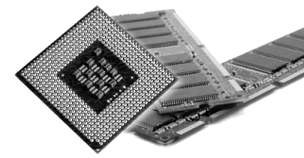
- eax: 0x89ABCDEF
- edx: 0x01234567
- mov [bigZ], eax
- mov [bigZ + 4], edx



Quad Word (Little Endian) im Speicher



- Beim Auslesen (komplette 64 Bit):
- bigZ = 01 23 45 67 89 AB CD EF

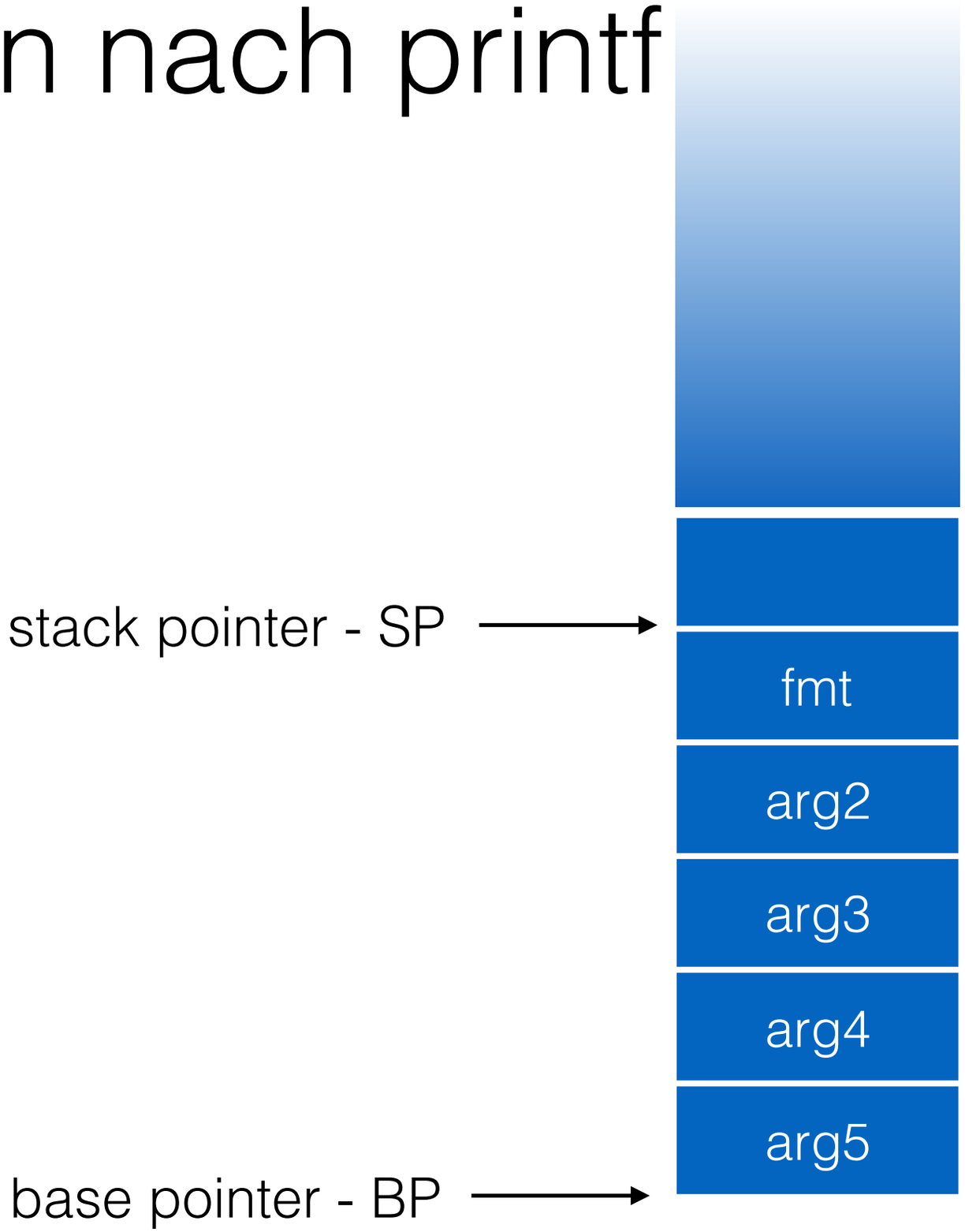


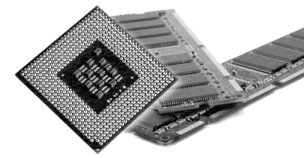
Vorteil Little Endian

- Zahl: 0x00000048
- Little Endian: 48 00 00 00
- Big Endian: 00 00 00 48
- Zugriff mit 16 Bit (anstelle von 32 Bit): mov ax, [zahl]
- Little Endian: 48 00 => ax = 0x0048
- Big Endian: 00 00 => ax = 0x0000

Stack leeren nach printf

- push arg5
- push arg4
- push arg3
- push arg2
- push fmt
- call printf
- **pop eax**
- **...**

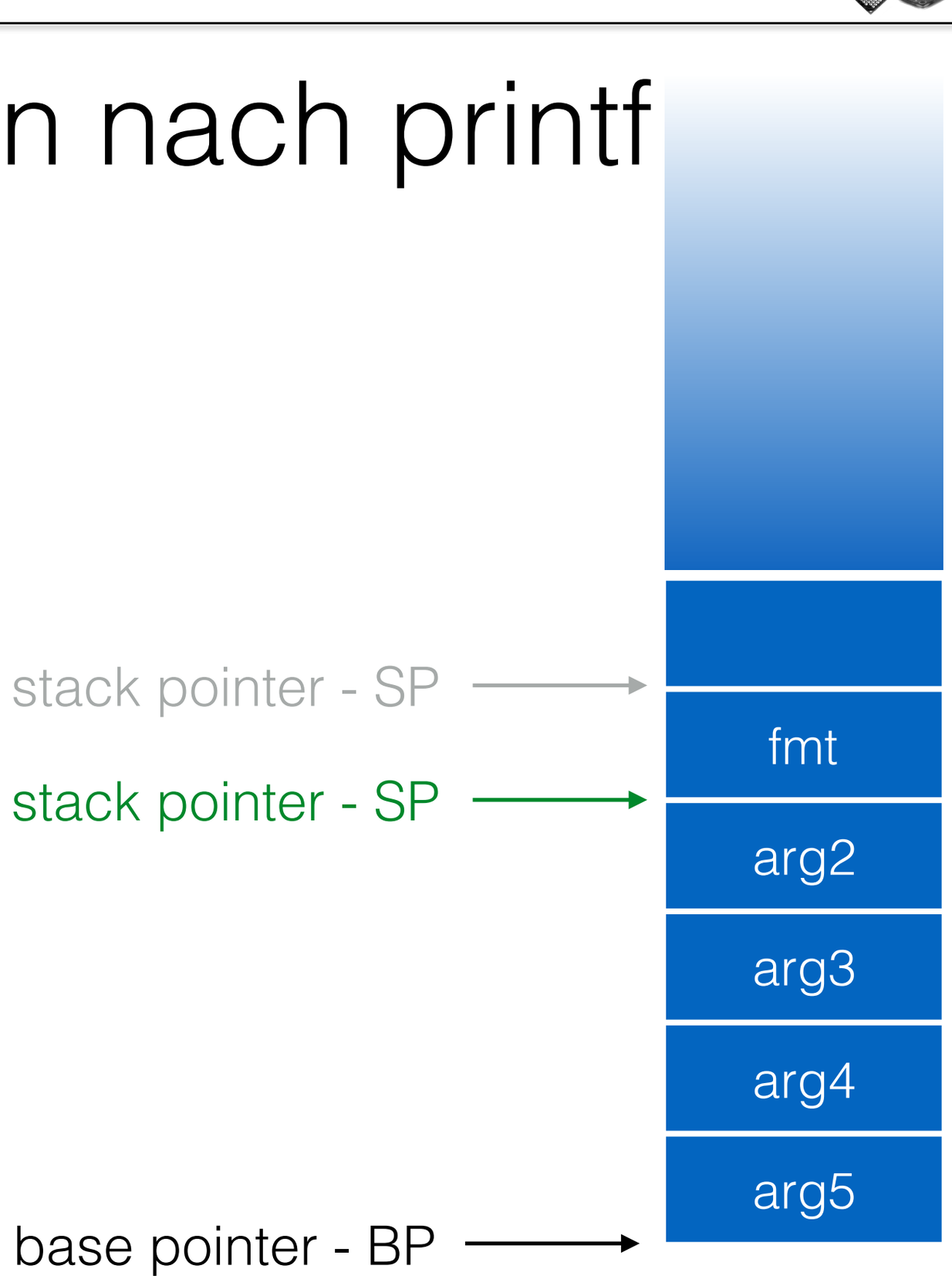


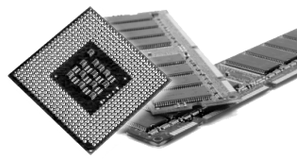


Stack leeren nach printf

- push arg5
- push arg4
- push arg3
- push arg2
- push fmt
- call printf
- **pop eax**
- **...**

nach pop:

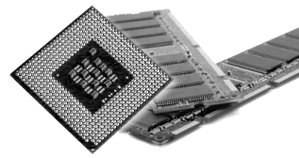




Stack leeren nach printf

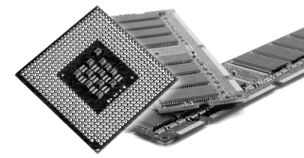
- push arg5
- push arg4
- push arg3
- push arg2
- push fmt
- call printf
- **mov esp, ebp**

stack pointer - SP ⟶ base pointer - BP ⟶



printf und Register

- printf verwendet die Register (z.B. eax, ecx, …)
- => Inhalt vorher sichern
- mov ecx, 5
- Schleife:
  - push txt
  - call printf <- ecx wird verändert
  - mov esp, ebp
- loop Schleife <- Endet nicht nach 5 mal



printf und Register

- mov ecx, 5
- Schleife:
  - **push ecx**
  - push txt
  - call printf
  - pop eax
  - **pop ecx**
- loop Schleife