

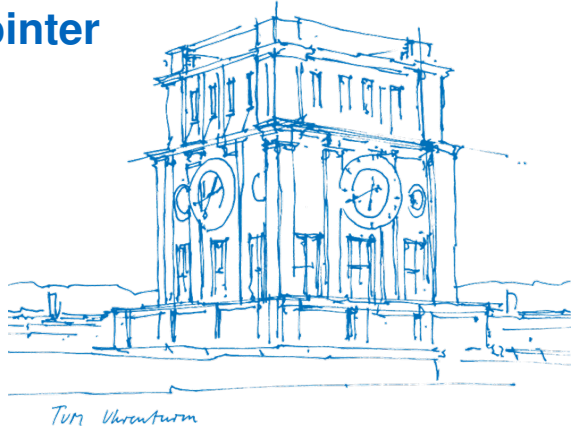
Übung 03: Sprünge und Pointer

Einführung in die Rechnerarchitektur

Niklas Ladurner

School of Computation, Information and Technology
Technische Universität München

31. Oktober 2025



Keine Garantie für die Richtigkeit der Tutorfolien.
Bei Unklarheiten/Unstimmigkeiten haben VL/ZÜ-Folien recht!

Branch-Befehle

Rücksprungadresse wird nicht gesichert,
Sprungbedingung muss erfüllt sein

- `beq: $rs1 = rs2$`
- `bne: $rs1 \neq rs2$`
- `blt(u): $rs1 < rs2$`
- `bgt(u): $rs1 > rs2$`

für Schleifen und Bedingungen (if's)

Jump-Befehle

Schreiben Rücksprungadresse in `ra` oder
angegebenes Register, springen immer

- `jal label`
- `jalr rd, offset(rs)`
- `call label`¹
- `j label` (Überschreibt `ra` nicht)

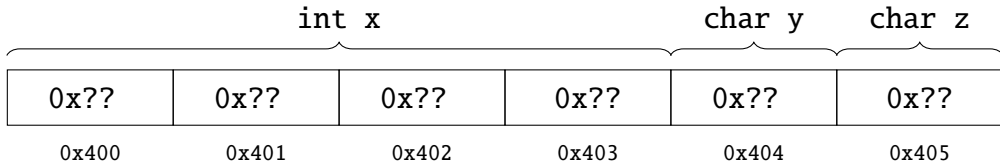
für Rekursion und Unterprogramme²

¹Pseudobefehl für `jalr` mit 32-Bit Offset

²außer `j label`, Pseudobefehl für `jal zero, label`

- 32-Bit-Architektur, d.h. Wortbreite von 4 Byte
- Adressen folglich auch 32 Bit
- Speicher ist Byte-adressierbar
- Daten liegen nacheinander im Speicher
- Adresse eines Symbols kann mittels `la rd, sym` geladen werden

```
struct myStruct{  
    int x;  
    char y;  
    char z;  
};
```



Sections und Direktiven

```
# compile-time Konstante
.equ NUM, 2748
```

```
# ro + init. Daten
.rodata
f: .word 2
```

```
# rw + init. Daten
.org 0x400
.data
arr: .byte 4, 3, 2, 1
string1: .ascii "asdf"
string2: .asciz "asdf"
```

```
# rw + uninit. Daten
.bss
```

```
a: .space 16
```

```
# globales Einstiegslabel
.globl _start
```

```
.org 0x200 # section beginnt an
↪ Adresse 0x200
```

```
.text
_start:
la a0, arr
lbu a1, 0(a0)
```

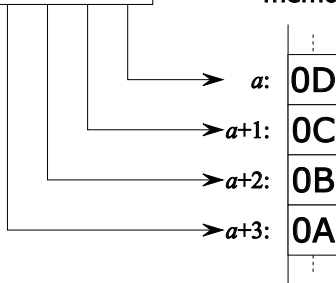
ro: read-only, **rw**: les- und schreibbar

Byte-Reihenfolge (Endianness)

one 32-bit integer

0A0B0C0D

arranged as
four bytes in
memory

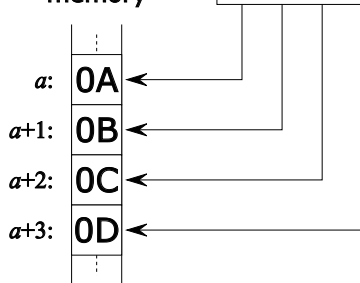


Little-endian

arranged as
four bytes in
memory

one 32-bit integer

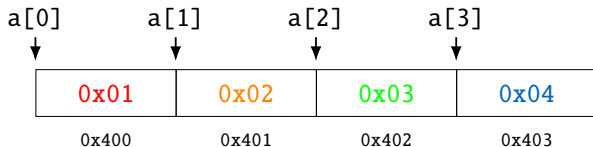
0A0B0C0D



Big-endian

Byte-Reihenfolge (Endianness)

```
.org 0x400  
.data  
  
a: .byte 0x01, 0x02, 0x03, 0x04
```

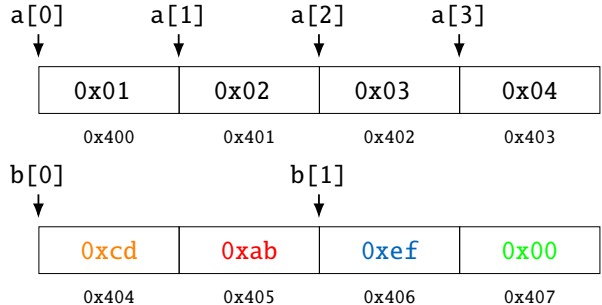


Wichtig: Innerhalb der einzelnen Bytes bleibt die Bitreihenfolge gleich!

Byte-Reihenfolge (Endianness)

```
.org 0x400
.data

a: .byte 0x01, 0x02, 0x03, 0x04
b: .half 0xabcd, 0x00ef
```



Wichtig: Innerhalb der einzelnen Bytes bleibt die Bitreihenfolge gleich!

Byte-Reihenfolge (Endianness)

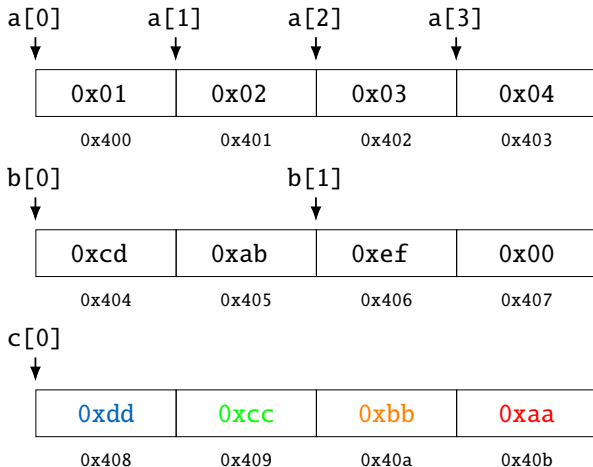
```
.org 0x400  
.data
```

```
a: .byte 0x01, 0x02, 0x03, 0x04
```

```
b: .half 0xabcd, 0x00ef
```

```
c: .word 0xaabbccdd
```

Wichtig: Innerhalb der einzelnen Bytes bleibt die Bitreihenfolge gleich!

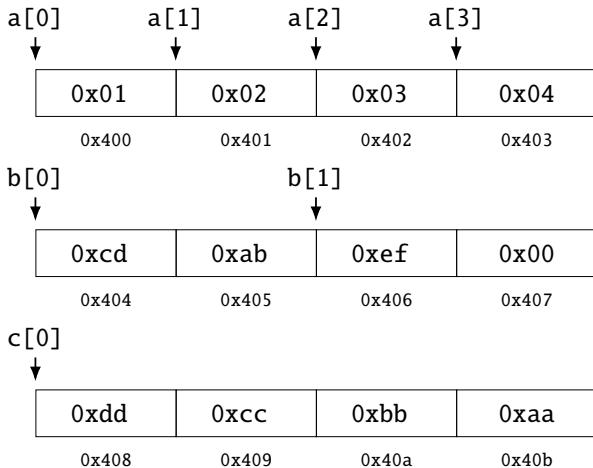


Byte-Reihenfolge (Endianness)

```
.org 0x400  
.data
```

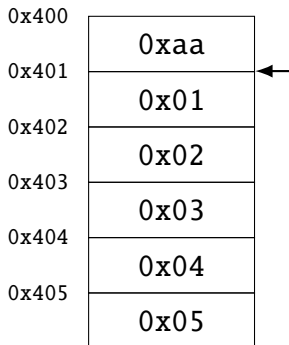
```
a: .byte 0x01, 0x02, 0x03, 0x04  
b: .half 0xabcd, 0x00ef  
c: .word 0xaabbccdd
```

Wichtig: Innerhalb der einzelnen Bytes bleibt die Bitreihenfolge gleich!



```
lw t0, 0(a0)
```

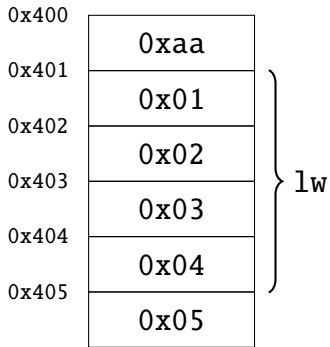
lade 4 Byte an der Adresse $a0 + 0$ Bytes
Offset in das Register $t0$



$a0 = 0x00000401$

```
lw t0, 0(a0)
```

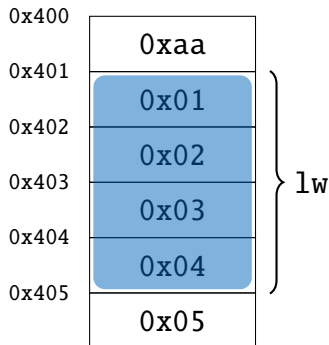
lade 4 Byte an der Adresse a0 + 0 Bytes
Offset in das Register t0



a0 = 0x00000401

```
lw t0, 0(a0)
```

lade 4 Byte an der Adresse a0 + 0 Bytes
Offset in das Register t0



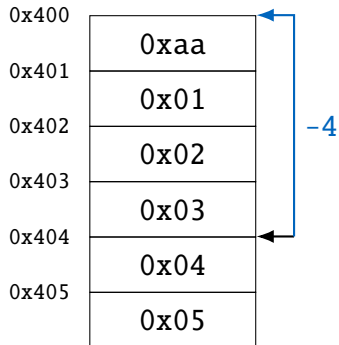
a0 = 0x00000401

t0 = 0x04030201

³byte: 1 Byte, half-word: 2 Byte, word: 4 Byte, Suffix unsigned

`lb t1, -4(a2)`

lade 1 Byte (sign-extended) an der Adresse
a2 - 4 Bytes Offset in das Register t1



a2 = 0x00000404

`lb t1, -4(a2)`

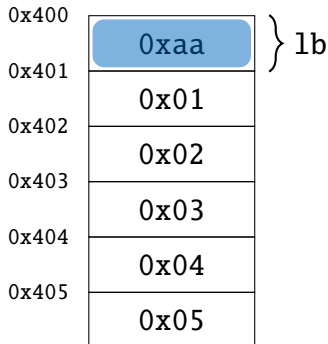
lade 1 Byte (sign-extended) an der Adresse
a2 - 4 Bytes Offset in das Register t1

0x400	0xaa	} 1b
0x401	0x01	
0x402	0x02	
0x403	0x03	
0x404	0x04	
0x405	0x05	

`a2 = 0x00000404`

`lb t1, -4(a2)`

lade 1 Byte (sign-extended) an der Adresse
a2 - 4 Bytes Offset in das Register t1



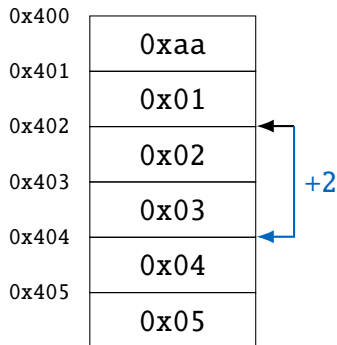
`a2 = 0x00000404`

`t1 = 0xffffffff`

³byte: 1 Byte, half-word: 2 Byte, word: 4 Byte, Suffix **u**nsigned


```
sh t3, 2(t4)
```

speichere die unteren 2 Byte von t3 an der
Adresse t4 + 2 Bytes Offset

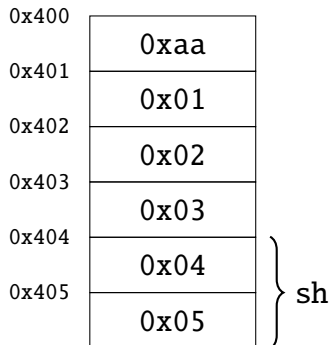


t4 = 0x00000402

t3 = 0xaabbccdd

```
sh t3, 2(t4)
```

speichere die unteren 2 Byte von t3 an der
Adresse t4 + 2 Bytes Offset

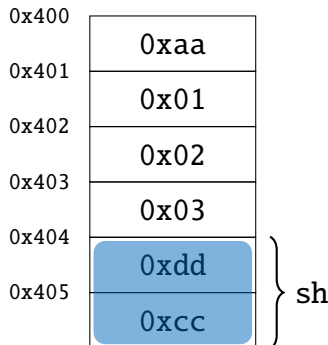


t4 = 0x00000402

t3 = 0xaabbccdd

```
sh t3, 2(t4)
```

speichere die unteren 2 Byte von t3 an der
Adresse t4 + 2 Bytes Offset



t4 = 0x00000402

t3 = 0xaabbccdd

³byte: 1 Byte, half-word: 2 Byte, word: 4 Byte, Suffix unsigned

Fragen?

- Zulip: „ERA Tutorium – Mi-1600-3“ bzw. „ERA Tutorium – Fr-1500-1“
- ERA-Moodle-Kurs
- ERA-Artemis-Kurs
- Übersicht an RISC-V-Instruktionen
- GNU as directives

Übung 03: Sprünge und Pointer

Einführung in die Rechnerarchitektur

Niklas Ladurner

School of Computation, Information and Technology
Technische Universität München

31. Oktober 2025

