

TI-Probeklausur

- ❑ Am 12. Juli 2007
- ❑ Um 14.00 Uhr
- ❑ Im Audimax-Hörsaal
- ❑ Anmeldung im Tutorium



6. Übung

➤ Virtuelle Speicherverwaltung



Speicherhierarchie

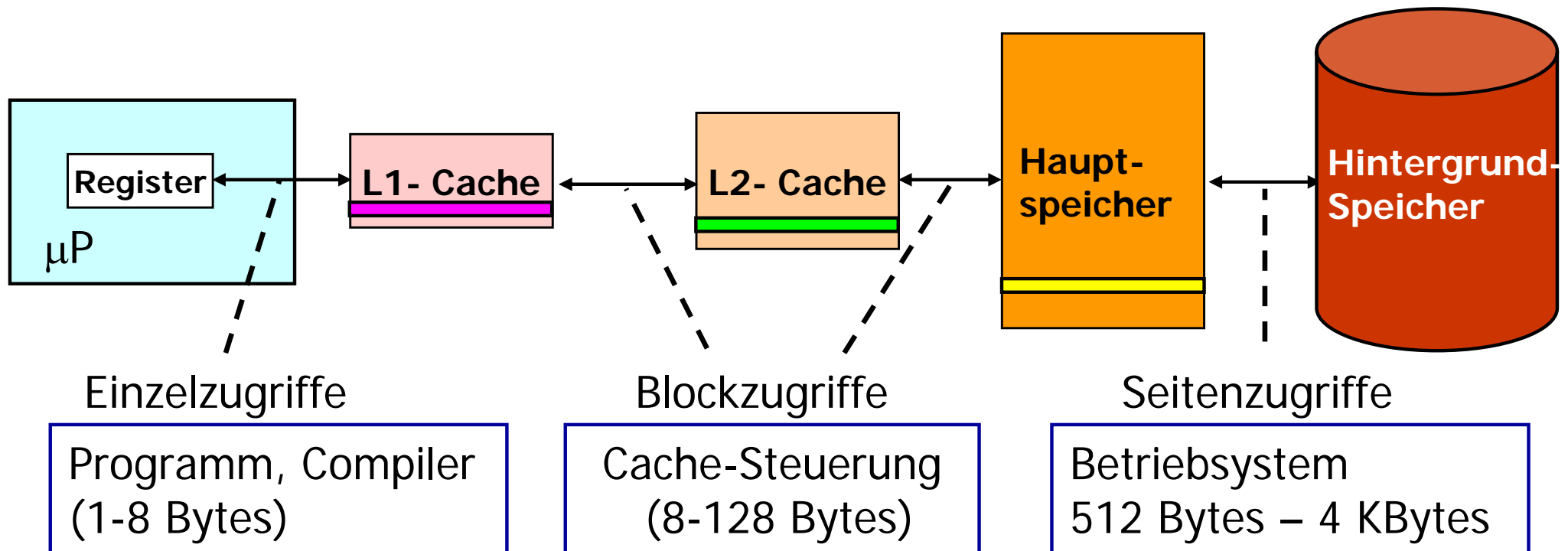
Speicherhierarchie zum Ausgleich der unterschiedlichen Zugriffszeiten der CPU und des Hauptspeichers.

2 Strategien:

- **Cache-Speicher:**
Kurze Zugriffszeiten ➡ Beschleunigung des Prozessorzugriffs
- **Virtueller Speicher:**
Vergrößerung des tatsächlich vorhandenen Hauptspeichers (z. B. bei gleichzeitiger Bearbeitung mehrerer Prozesse)

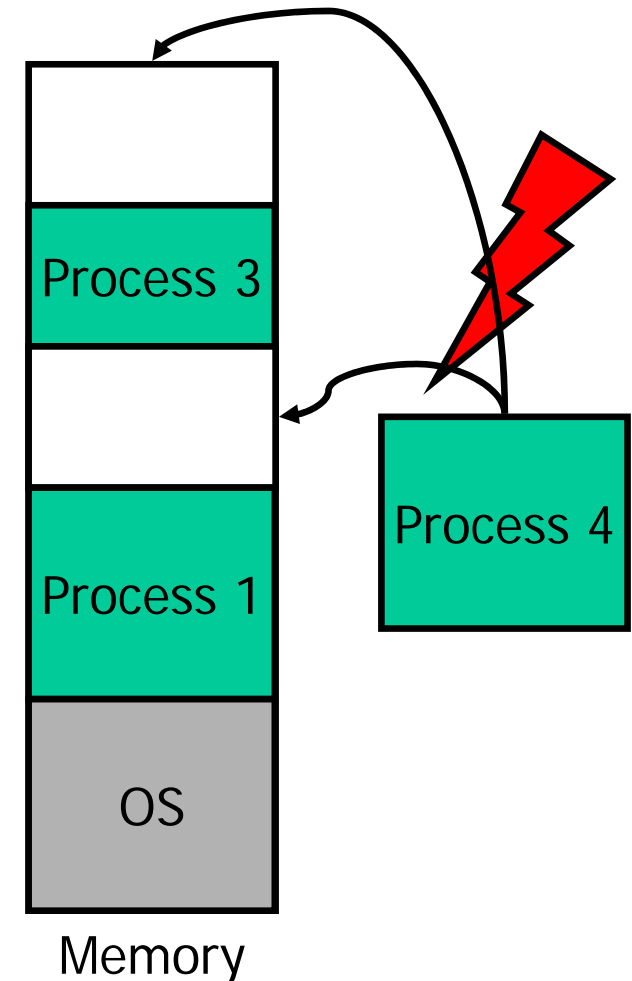
Speicherhierarchie

Daten werden nur zwischen aufeinanderfolgenden Ebenen der Speicherhierarchie kopiert.



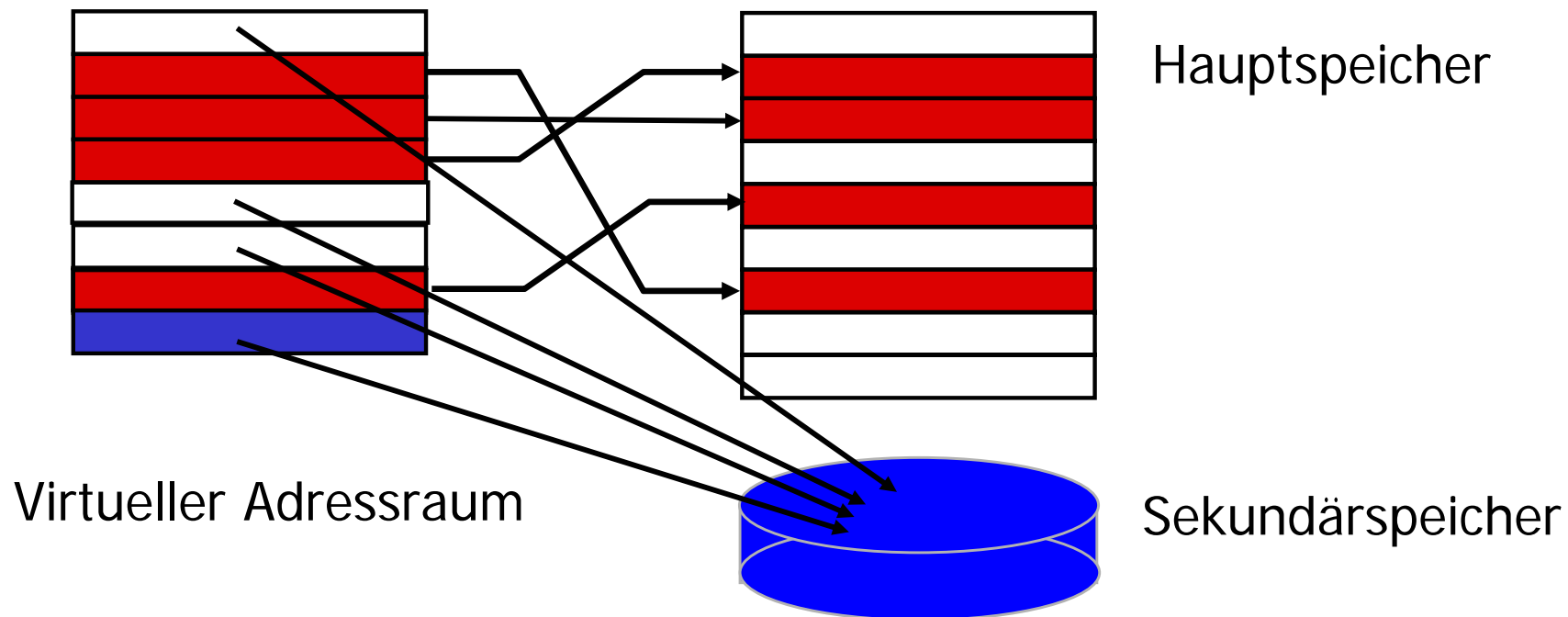
Speicherverwaltung

- ❑ Hauptspeicherkapazität ist begrenzt
 - Prozess benötigt mehr Speicher als der Hauptspeicher
 - Mehr aktive Prozesse als der Hauptspeicher "vertragen" kann.
- ❑ Effiziente Schutzmechanismen



Speicherverwaltung

- ❑ Virtuelle Speicherkapazität > effektive Hauptspeicherkapazität
- ❑ Betriebssystem lagert bei Bedarf Speicherbereiche ein und aus
- ❑ Speicherverwaltungseinheiten (*memory management units, MMU*) unterstützt hardwaremäßig eindeutige Adressberechnung
- ❑ Abbildungsinformation in Übersetzungstabellen



Speicherverwaltung

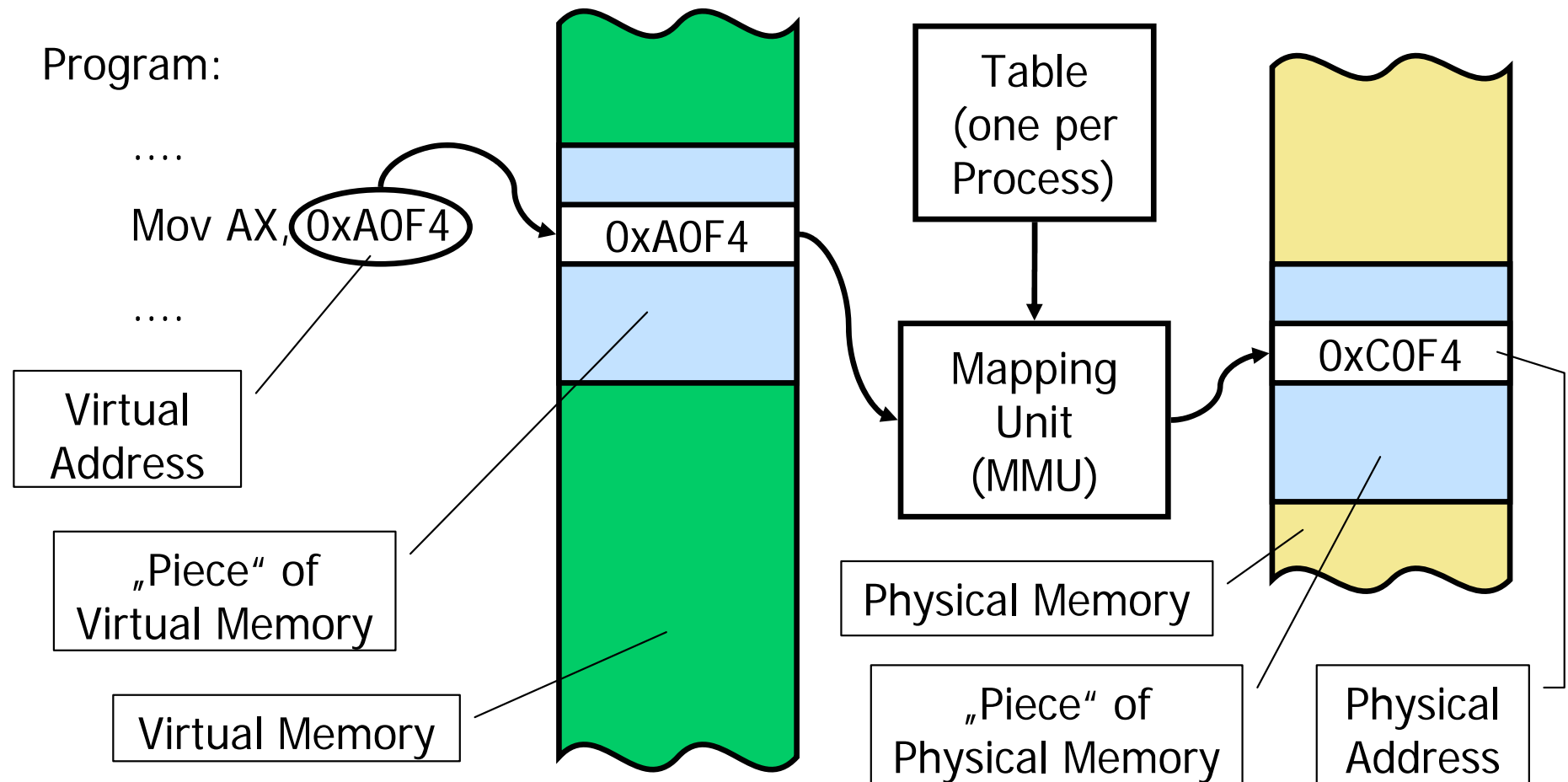
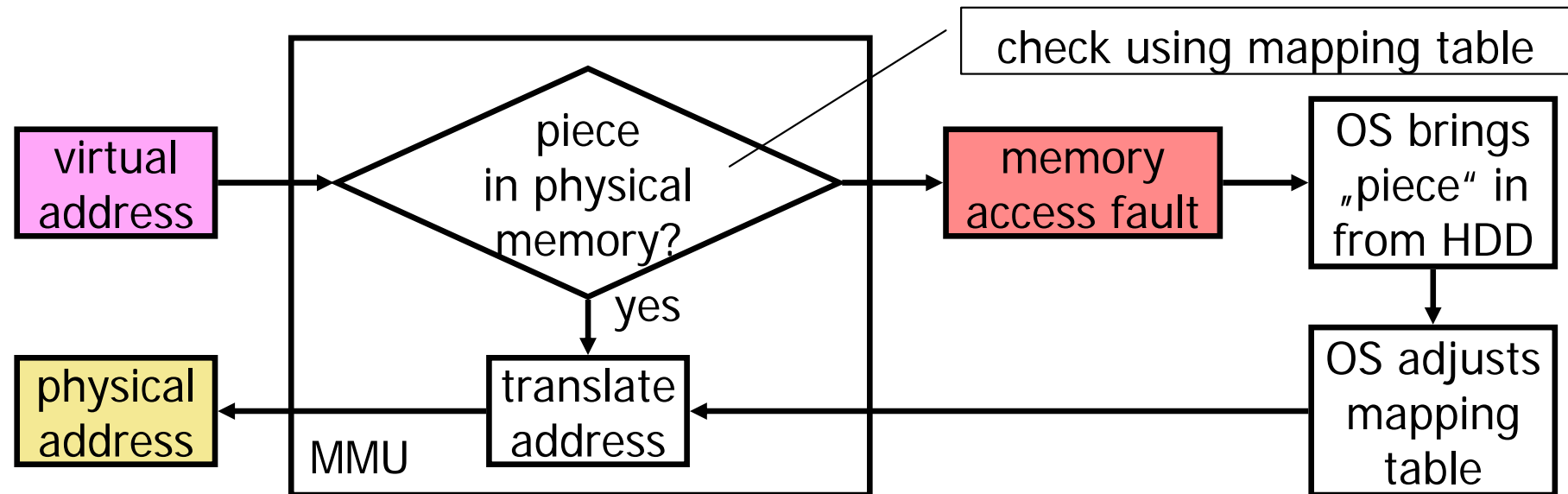
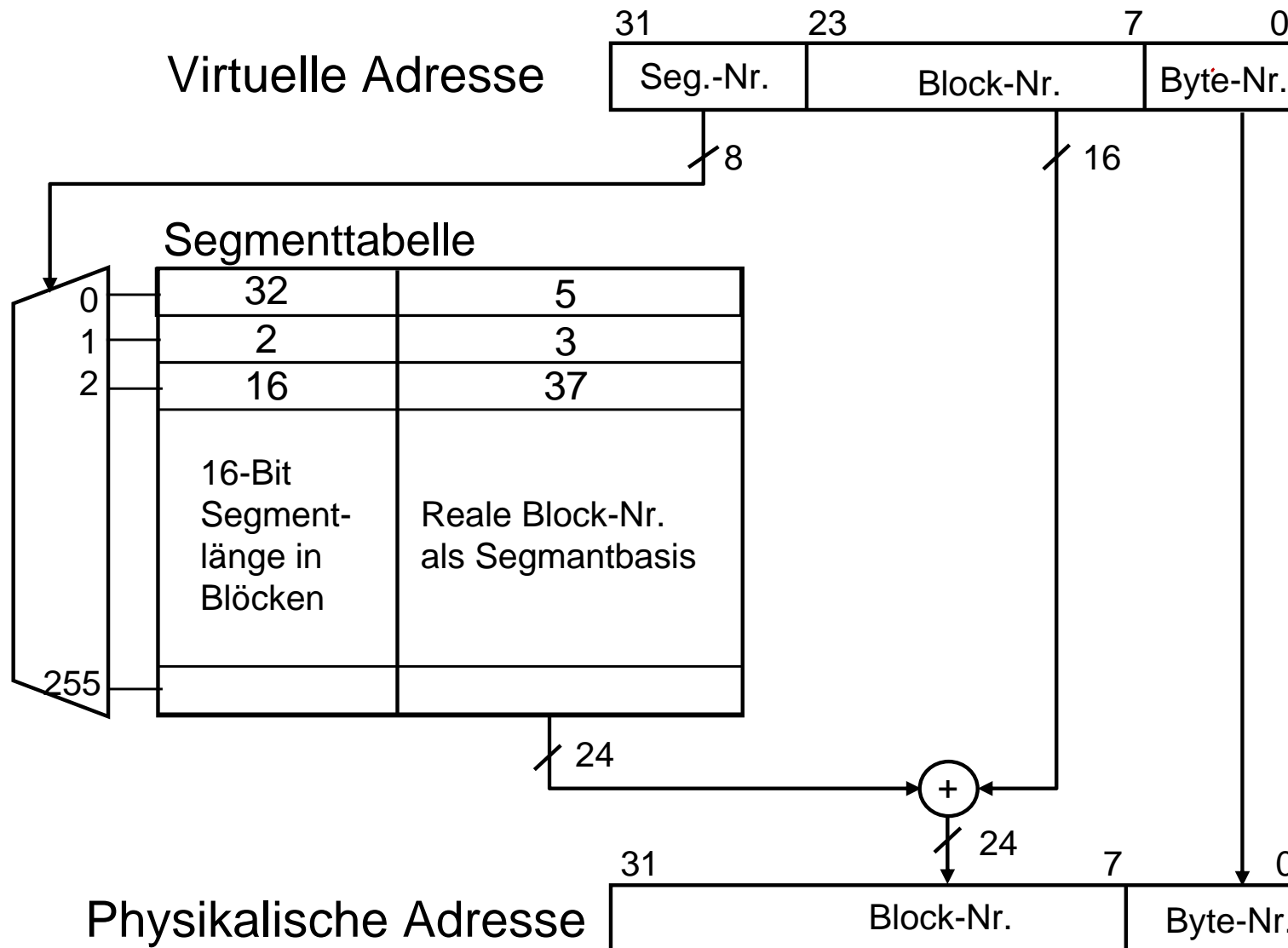


Abbildung: virtuell → physikalisch

- Jeder Prozess hat seinen virtuellen Adressraum und seine Abbildungstabelle

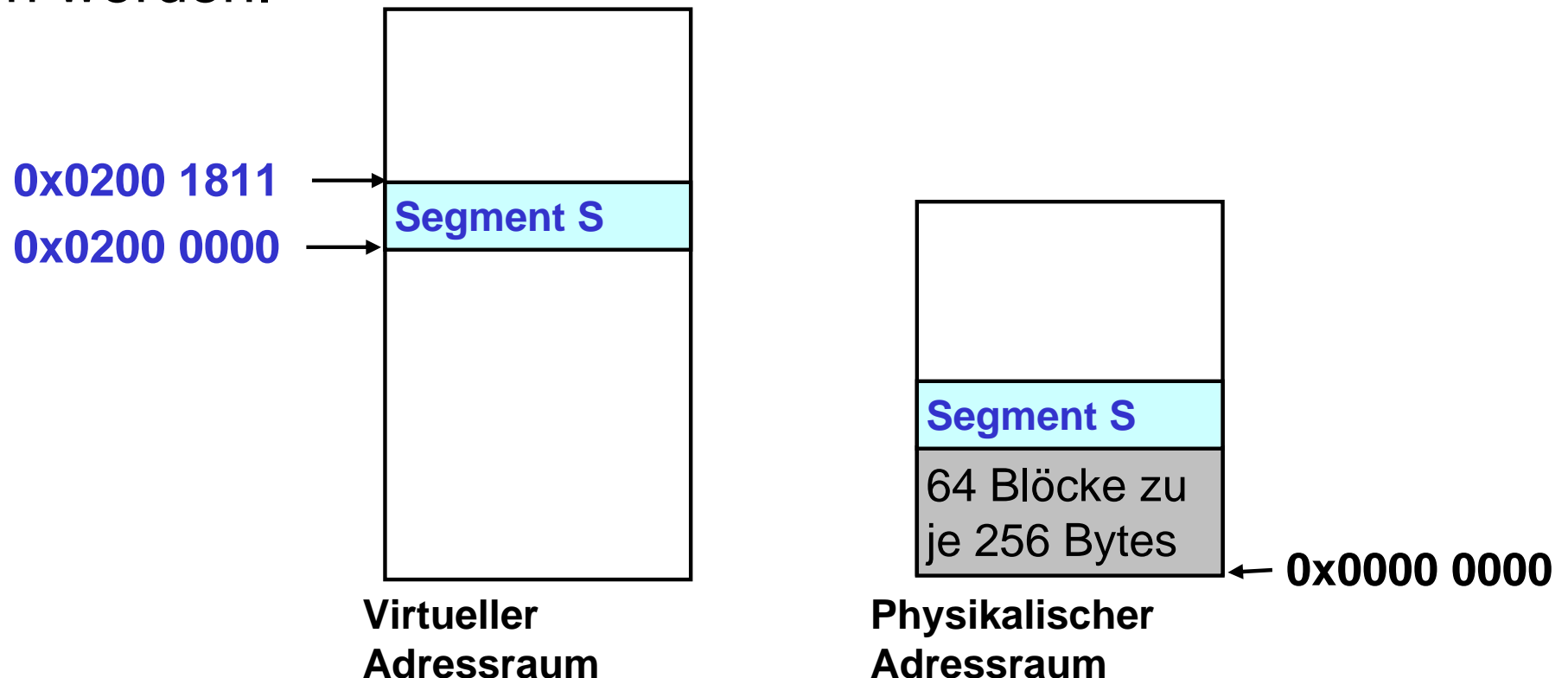


Segmentbasierte Speicherverwaltung



Aufgabe 1

Der Hauptspeicher sei bereits beginnend ab Adresse 0 mit **64 Blöcken** zu je **256 Bytes** belegt. Im Anschluss an diesen Bereich soll ein Segment **S** mit der virtuellen Anfangsadresse **02000000₁₆** und der virtuellen Endadresse **2001811₁₆** geladen werden.



Lösung 1.1

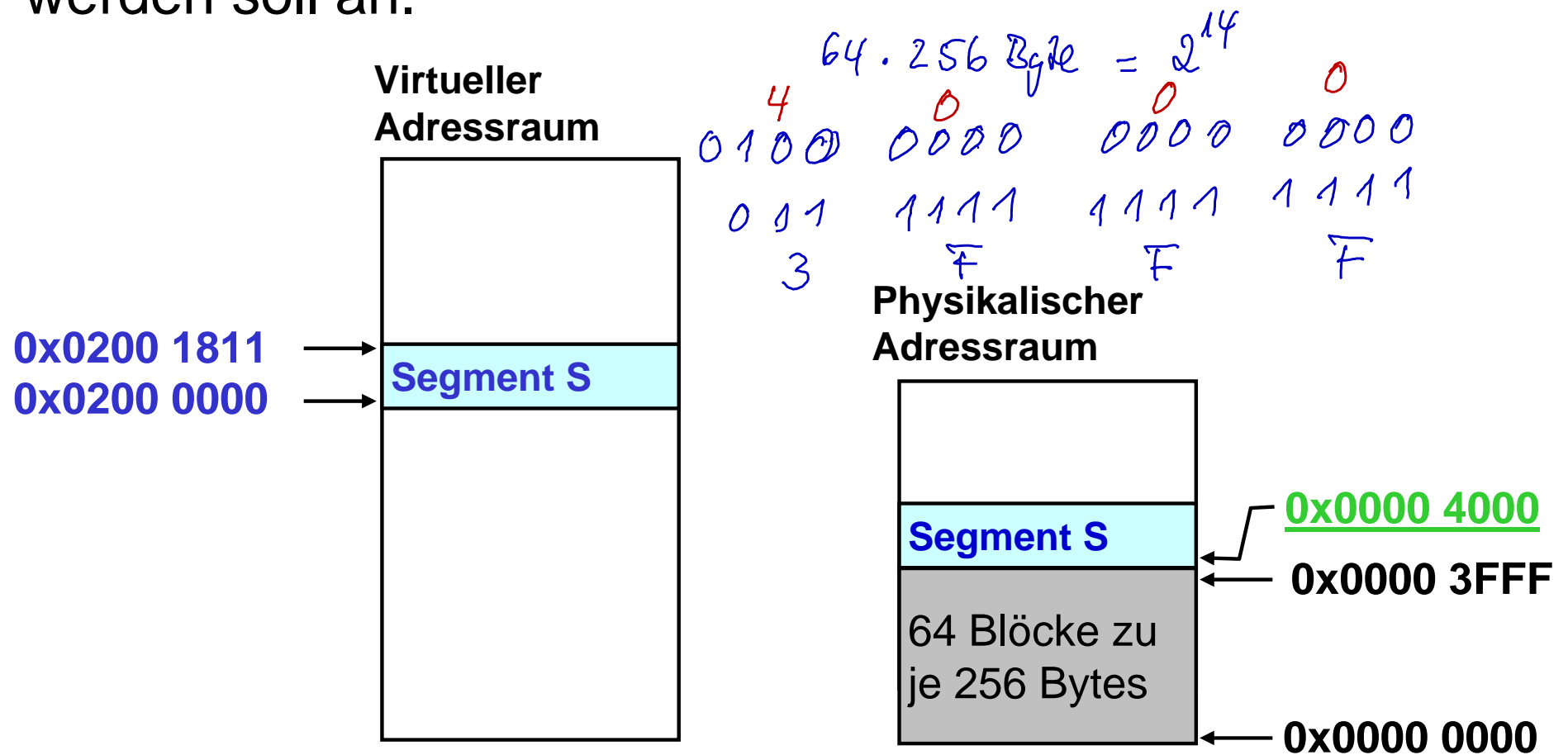
64 Blöcken zu je **256 Bytes** im Hauptspeicher ab Adresse 0

Im Anschluß daran: Segment **S** mit der virtuellen Anfangsadresse **02000000₁₆** und Endadresse **02001811₁₆**

1. Geben Sie die Ladeadresse des Segmentes **S** unter der Bedingung, dass der Hauptspeicher lückenlos gefüllt werden soll an.

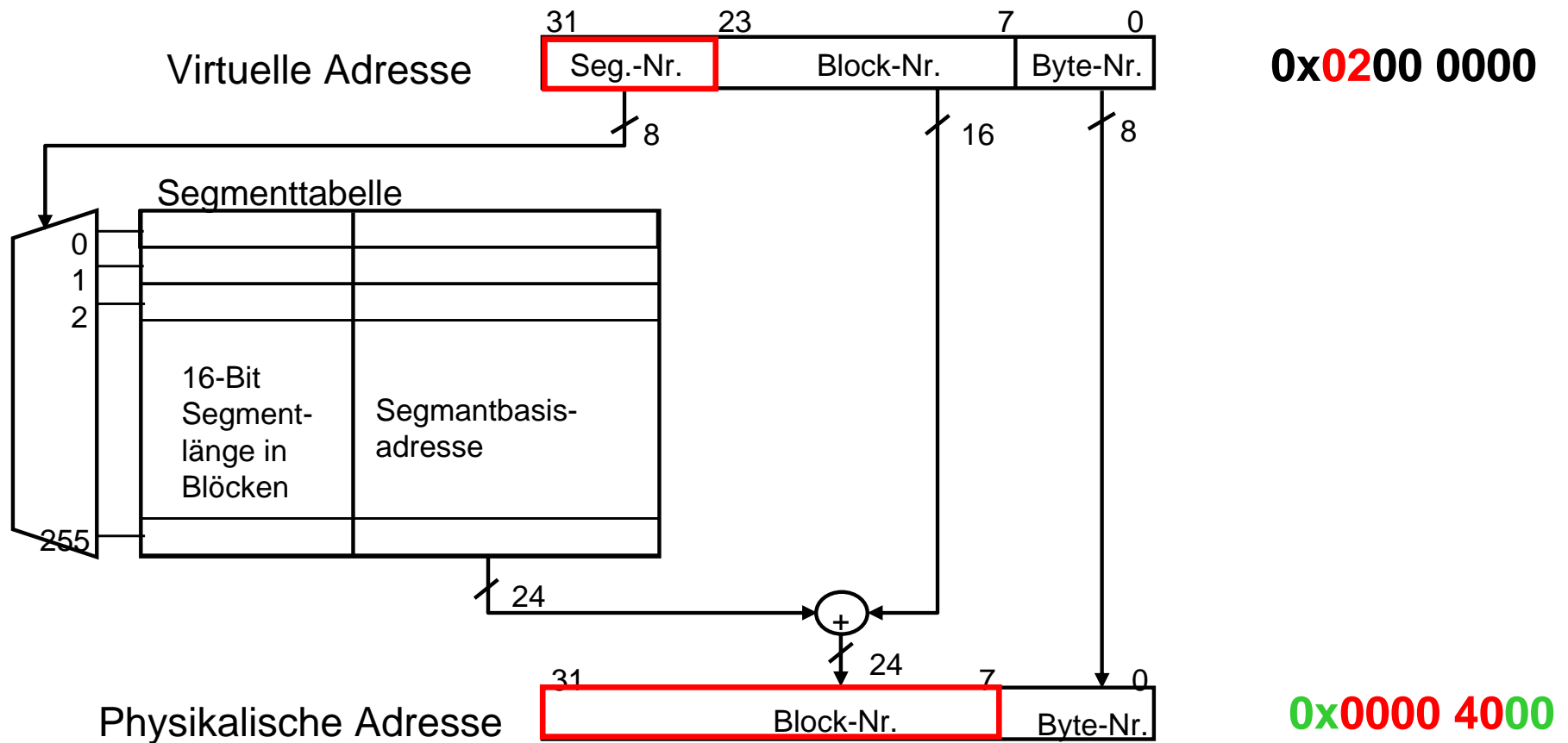
Lösung 1.1

1. Geben Sie die **Ladeadresse** des Segmentes **S** unter der Bedingung, dass der Hauptspeicher lückenlos gefüllt werden soll an.



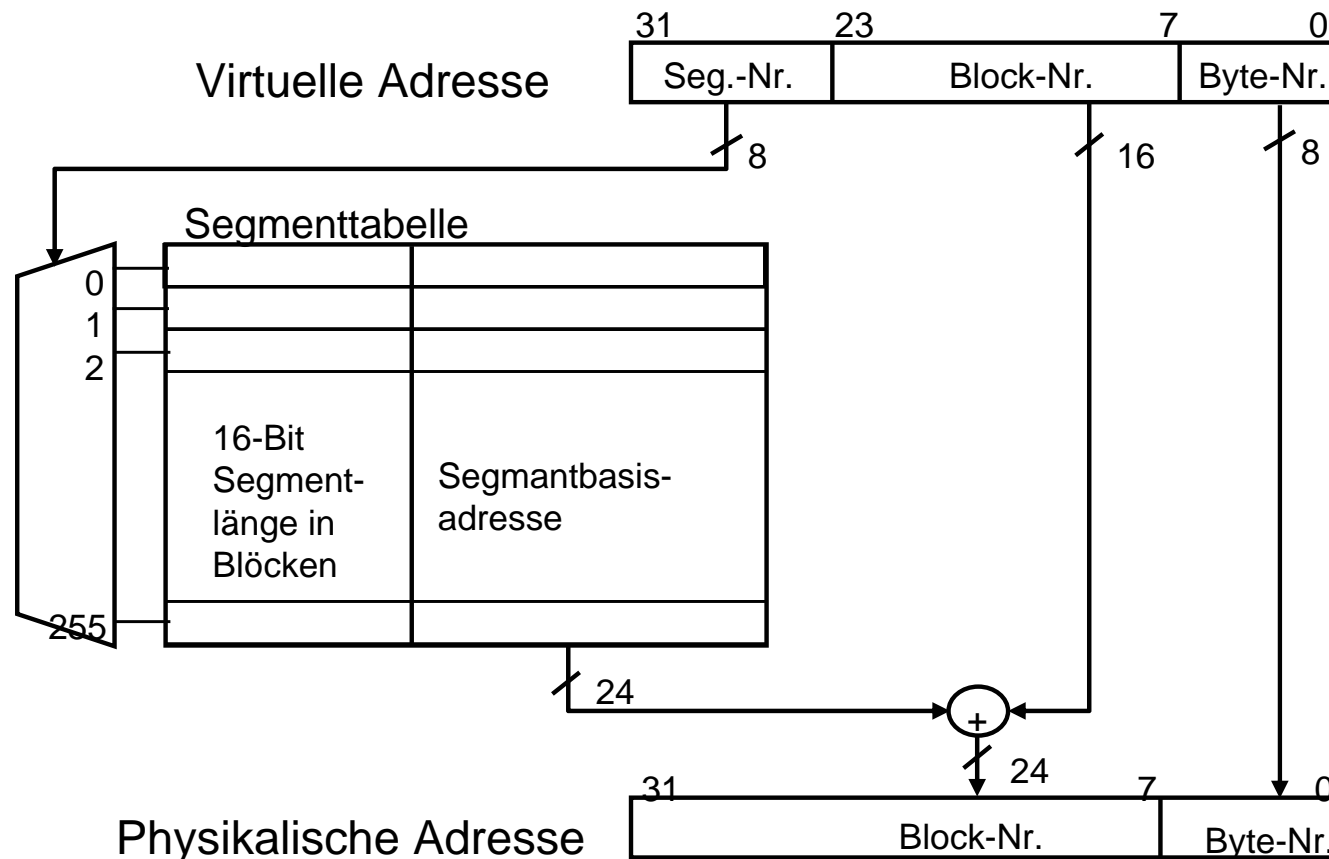
Lösung 1.2

2. An welcher Position im Registerspeicher der MMU muss die physikalische Blocknummer als Abbildungsinformation abgelegt werden und welchen Wert hat sie?



Lösung 1.3

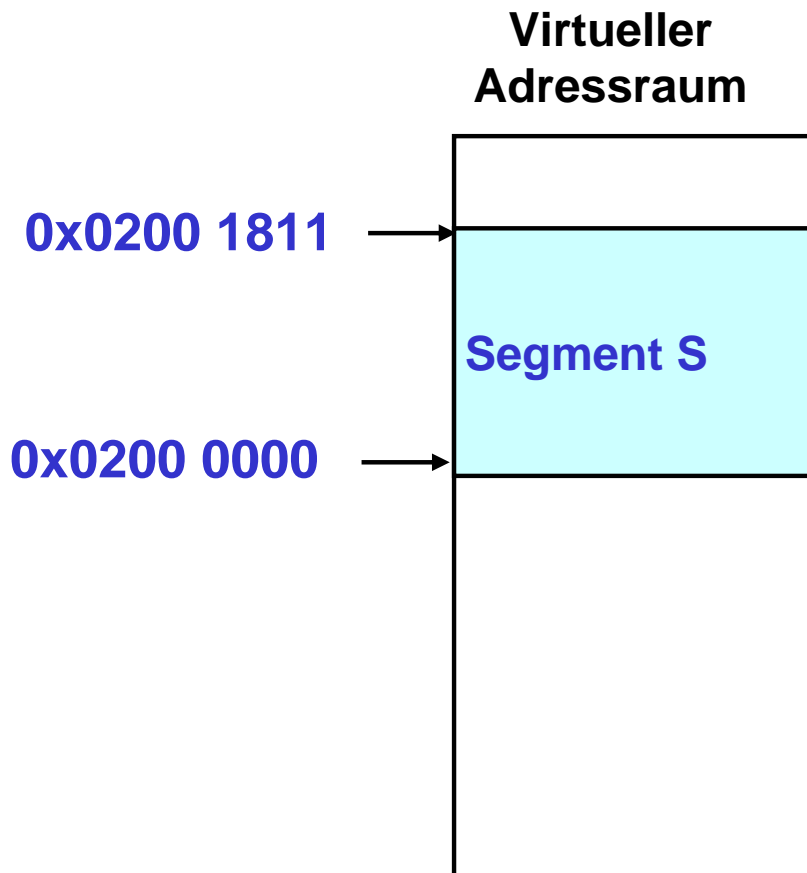
3. Welchen Wert hat die für die Überprüfung von Segmentüberschreitungen einzutragende Blockanzahl?



Wie groß ist
Segment SI
in Blöcke?

Lösung 1.3

3. Welchen Wert hat die für die Überprüfung von Segmentüberschreitungen einzutragende Blockanzahl?



• Segment S ist

$$0x200\ 1811 - 0x200\ 0000 =$$

$$0x\ 1812\ \text{Byte}$$

• 1 Block ist 256 Byte groß

⇒ Segment S ist

$$\frac{0x1812}{256} = \frac{0x1812}{0x100} = 0x18,12$$

Blöcke groß

Lösung 1.3

Segment S benötigt 19_{16} Blöcke (25_{10} Blöcke):

- Block 0 bis Block 18_{16} (24_{10}) sind voll
- Block 19_{16} (25_{10}) ist nur teilweise belegt (enthält nur 12 Bytes)

➔ Für eine Überprüfung von Segmentüberschreitung muss man für Segment S den Wert 25_{10} eintragen

Aufgabe 2

Gegeben sei eine Speicherverwaltungseinheit (MMU) mit einer Seitengröße von 4 KByte, 8 virtuellen Seiten und 4 physikalische Seiten. Die Seitentabelle ist wie folgt belegt:

Virtuelle Seiten-Nr.	Physikalische Seiten-Nr.
0	3
1	1
2	-
3	-
4	2
5	-
6	0
7	-

Lösung 2.1

1. Wie groß ist physikalische und der virtuelle Adressraum?

Physikalischer Adressraum:

$$4 \text{ Seiten} \times 4 \text{ KByte} = 16 \text{ KByte}$$

Virtueller Adressraum:

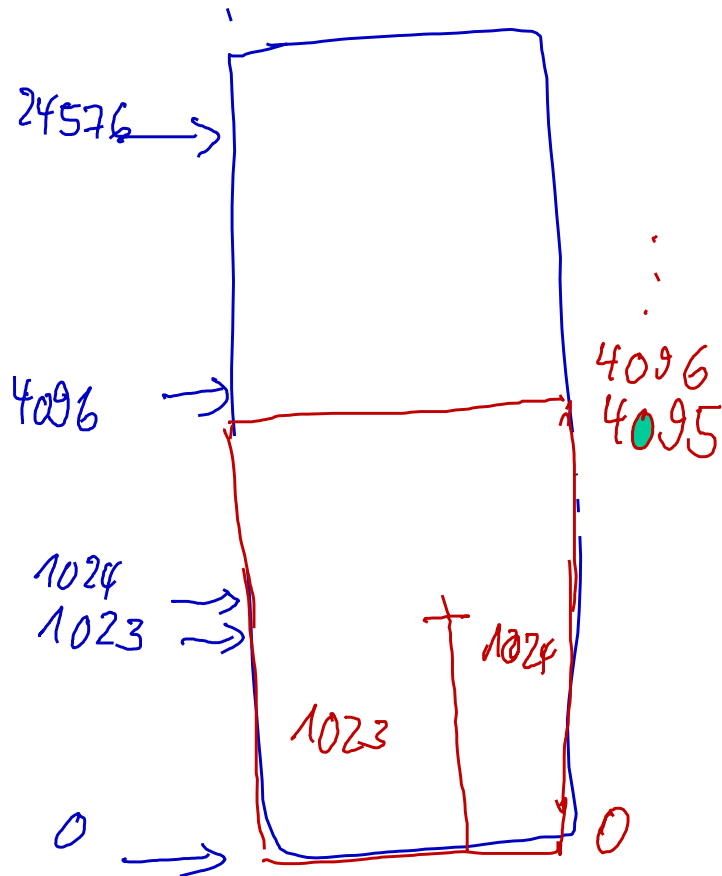
$$8 \text{ Seiten} \times 4 \text{ KByte} = 32 \text{ KByte}$$

Lösung 2.2

2. Ermitteln Sie die physikalischen Adressen zu den folgenden virtuellen Adressen:

0, 1023, 1024, 4096, 24576

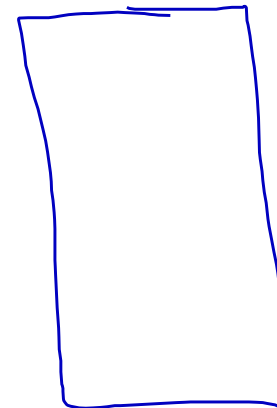
8 Seiten



Virtuell
Adressraum

1 Seite ist 4KByte groß
 \rightarrow 4096 Byte

4 Seiten



$$(\text{address}) \text{ div } (\text{Seitengröße}) = \text{Seitennummer}$$

$$(\text{address}) \text{ mod } (\quad) = \text{Offset}$$

Lösung 2.2

2. Ermitteln Sie die physikalischen Adressen zu den folgenden virtuellen Adressen:

0, 1023, 1024, 4096, 24576



Lösung 2.

2. Ermitteln Sie die physikalischen Adressen für die folgenden virtuellen Adressen:

2000 (4096)
②

0, 1023, 1024, 4096, 24576

Virtuelle Seiten-Nr.	Physikalische Seiten-Nr.
0	3
1	1
2	-
3	-
4	2
5	-
6	0
7	-

Virtuelle		Physikalische	
Adresse	Seiten-Nr.	Seiten-Nr.	Adresse
0	0 (0)	3	12288
1023	0 (1023)	3	13311
1024	0 (1024)	3	13312
4096	1 (0)	1	4096
24576	6 (0)	0	0

4096
- 3
12288
1023
-
13311

Seiten-nr. (offset)



Lösung 2.

2. Ermitteln Sie die Physikalischen Adressen für die folgenden virtuellen Adressen:

0, 1023, 1024, 4096, 24576

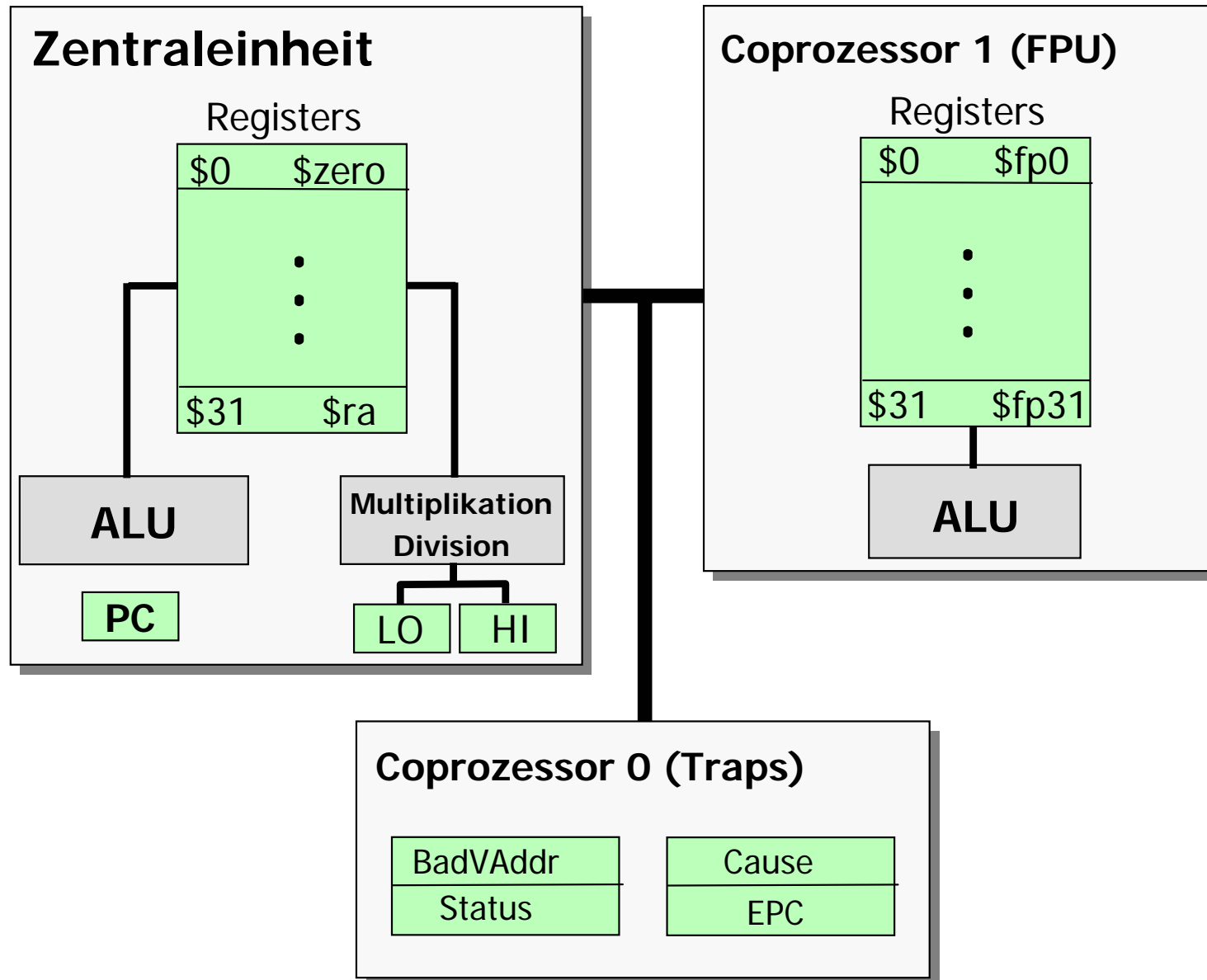
Virtuelle Seiten-Nr.	Physikalische Seiten-Nr.
0	3
1	1
2	-
3	-
4	2
5	-
6	0
7	-

Virtuelle		Physikalische	
Adresse	Seiten-Nr.	Seiten-Nr.	Adresse
0	0	3	12288
1023	0	3	13311
1024	0	3	13312
4096	1	1	4096
24576	6	0	0

□ MIPS-Assembler

- Befehlssatz
- Pseudobefehle
- Programmiertechniken
- Assemblerprogrammierung
- Stackprogrammierung
- Unterprogramm-Aufruf

Aufbau des MIPS-Prozessors



adresse : anfangsadresse von feld +
($\$t_0$) array [0 ... 12]

feld: • space 52

13 Element (integers)

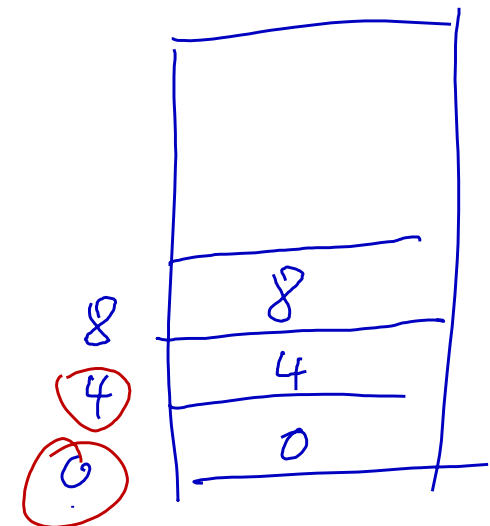
52 Bytes 4 Byte

feld[i] = i

li $\$t_0, 0$

sw $\$t_0, \text{feld}(\$t_0)$

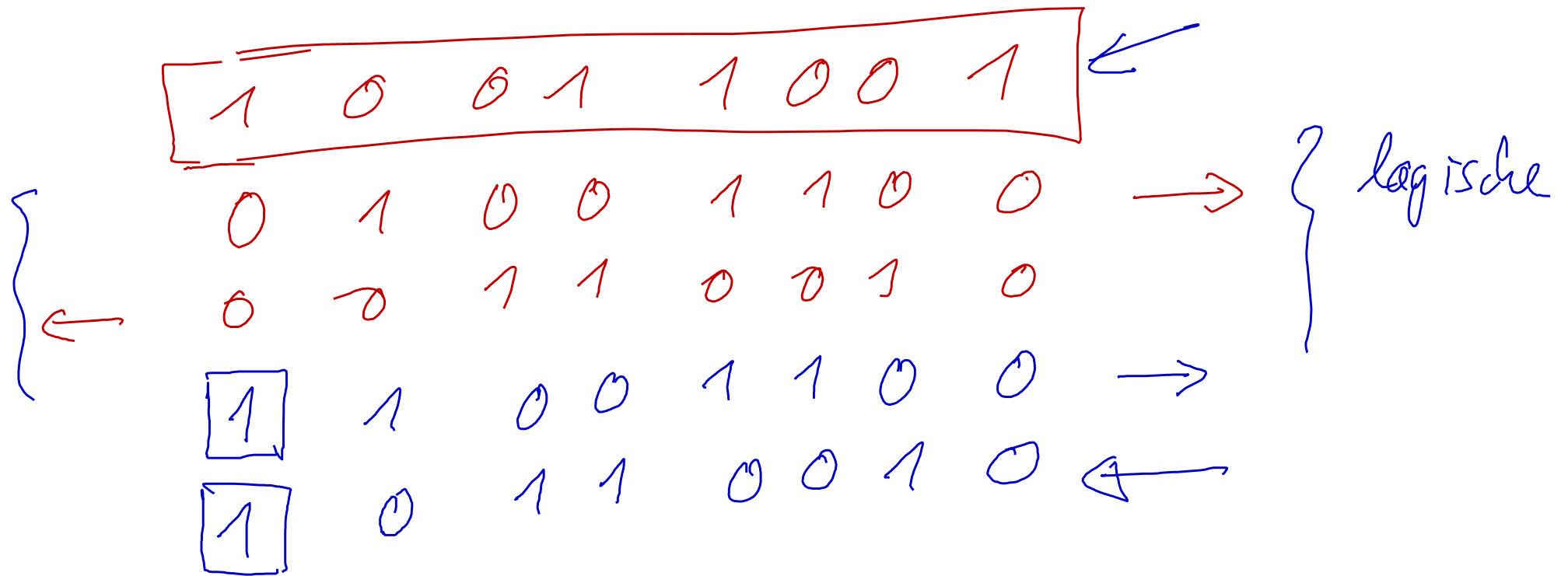
addi $\$t_0, \$t_0, 4$



Befehlssatz

Logische Befehle

- logisches AND `and rd,rs,rt` `andi rd,rs,imm`
- logisches NOR `nor rd,rs,rt`
- logische Invertierung `not,rdest,rsrc`
- logisches XOR `xor rd,rs,rt` `xori rd,rs,imm`
- logisches OR `or rd,rs,rt` `ori rd,rs,imm`
- bitweise Rotieren `rol/ror rdest,rsrc1,rsrc2`
- bitweise Schieben
`sll rd,rs,imm (imm = distance)`
`sllv rd,rs,rt`
`sra rd,rs,imm (imm = distance)`
`srlv rd,rs,rs`



Befehlssatz

Befehle zum Laden von Konstanten

- Laden einer Konstante in ein Register

`li rdest, imm`

`lui rdest, imm`

li \$t0, 2007 li \$t1, 0x100

Vergleichsbefehle

- Vergleich zweier Register

`slt/sltu` `rd,rs,rt`

`slti/sltiu` `rd,rs,imm`

`seq rdest, rsrc1, rsrc2`

`sge/sgeu rdest, rsrc1, rsrc2`

`sgt/sgtu rdest, rsrc1, rsrc2`

`sle/sleu rdest, rsrc1, rsrc2`

`sne rdest, rsrc1, rsrc2`

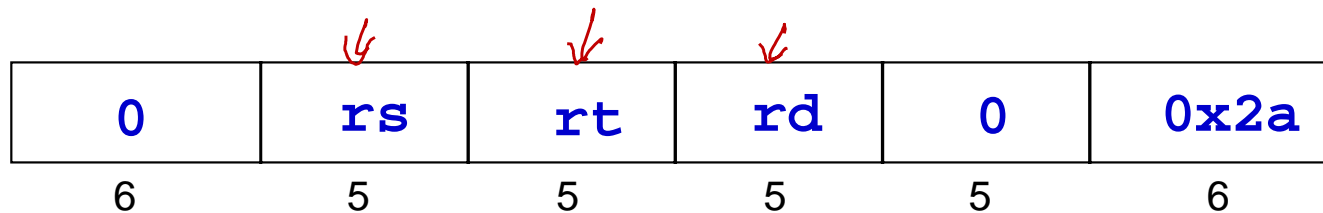
- Vergleich eines Registers mit Null

Vergleichsbefehle

```
if ( $s1 < $s2 ) then  
    $t0 = 1  
else  
    $t0 = 0  
    }   slt $t0, $s1, $s2
```

slt rd, rs, rt

slt: set less than



Befehlssatz

Kontrollflussbefehle

- unbedingtes Verzweigen zu einer Adresse
`j target b label`
- unbedingtes Verzweigen zu einer Adresse und sichern der nachfolgenden Adresse (für Unterprogramme)
`jal target`
- Verzweigen, wenn Bedingungs-Flag eines Coprozessors wahr/falsch ist
`bczt label bczf label`
- Verzweigen, wenn ein Register größer/kleiner als ein anderes Register ist

<code>bgt rsrc1,rsrc2,label</code>	<code>bgtu rsrc1,rsrc2,label</code>
<code>blt rsrc1,rsrc2,label</code>	<code>bltu rsrc1,rsrc2,label</code>

(auch `bge`, `bgeu`, `ble`, `bleu`)

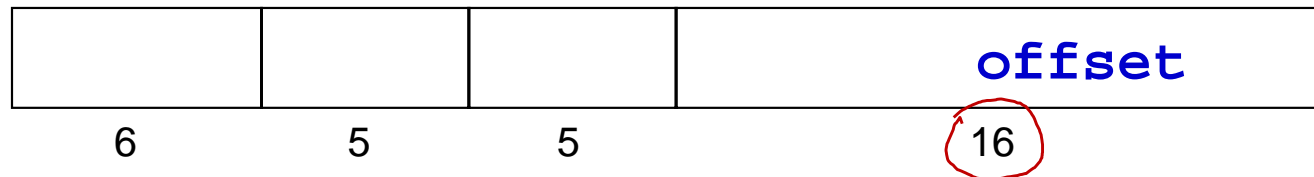
Befehlssatz

Kontrollflussbefehle

- Verzweigen, wenn zwei Register gleich/ungleich sind
`beq rs,rt,label` `bnq rs,rt,label`
- Verzweigen, wenn ein Register größer/kleiner als Null ist
`bgtz rs, label` `bltz rs,label`
(auch `bgez`, `blez`)
- Verzweigen, wenn ein Register gleich/ungleich Null ist
`beqz rsrc, label` `bnez rs,label`

Kontrollflussbefehle

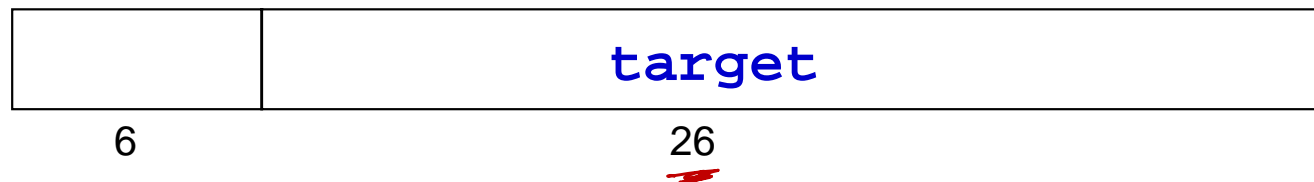
Branch



Offset: 16-bit, vorzeichenbehaftet

➔ $2^{15}-1$ Befehle vorwärts und 2^{15} rückwärts

jump



26-bit Adress-Feld

Kontrollflussbefehle

Einstufige Verzweigung:

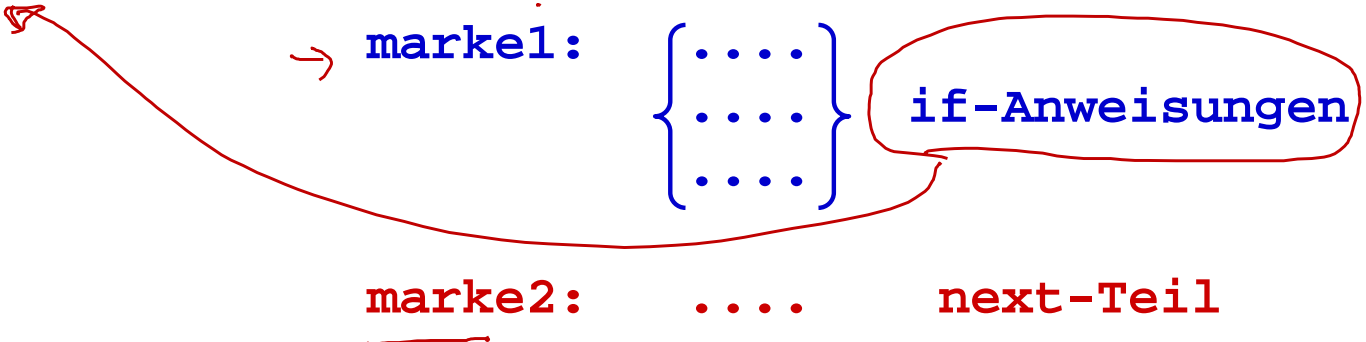
Eine einstufige Verzweigung wird durch einen bedingten Sprung realisiert

if-Anweisung

```
if ( register_s1 == 0 )  
{  
    if-Anweisungen  
}  
next-Teil;
```

Assembler-Code

```
beqz $s1, marke1  
j    marke2  
  
marke1: { ..... } if-Anweisungen  
marke2: ..... next-Teil
```



Kontrollflussbefehle

Zweistellige Verzweigung:

if-else-Anweisung

```
if (register_s1 == 0)
{
    if-Anweisungen ←
}
else
{
    else-Anweisungen ↑
}
next;
```

Assembler-Code

→ beqz \$s1, marke1

$\left\{ \begin{array}{c} \dots \\ \dots \\ \dots \end{array} \right\}$ else-Anweisungen

j marke2

⇒ marke1: $\left\{ \begin{array}{c} \dots \\ \dots \\ \dots \end{array} \right\}$ if-Anweisungen

marke2: next-Teil

Kontrollflussbefehle

Bedingte Verzweigung

`bne $t0, $t1, Label`

`beq $t0, $t1, Label`

```
if (i==j)
    h = i + j;
```

```
bne $s0, $s1, Label
add $s3, $s0, $s1
```

```
→ Label:    .h... i    j
```

Kontrollflussbefehle

Unbedingte Verzweigung

`j label`

<code>if (i!=j)</code>		<code>beq \$s4, \$s5, Lab1</code>
<code> <i>h=i+j;</i></code>	\rightarrow	<code><i>add \$s3, \$s4, \$s5</i></code>
<code>else</code>		<code><u>j Lab2</u></code>
<code> <i>h=i-j;</i></code>	\rightarrow	<code>Lab1: <i>sub \$s3, \$s4, \$s5</i></code>
		<code>Lab2: ...</code>

next \rightarrow

Befehlssatz

Lade- und Speicherbefehle

- Laden einer Adresse
`la rdest, address`

- Laden und Speichern eines Bytes, Halbwortes, Wortes und Doppelwort

<code>lb <u>rt</u>, <u>address</u></code>	<code>sb <u>rt</u>, <u>address</u></code>
<code>lbu <u>rt</u>, <u>address</u></code>	
<code>lh <u>rt</u>, <u>address</u></code>	<code>sh <u>rt</u>, <u>address</u></code>
<code>lhu <u>rt</u>, <u>address</u></code>	
<code><u>lw</u> <u>rt</u>, <u>address</u></code>	<code><u>sw</u> <u>rt</u>, <u>address</u></code>
<code>ld <u>rt</u>, <u>address</u></code>	<code>sd <u>rt</u>, <u>address</u></code>

- Laden und Speichern von Koprozessor-Registern
`lwcz rt, address` `swcz rt, address` (z=1, FPU)

Befehlssatz

Lade- und Speicherbefehle:

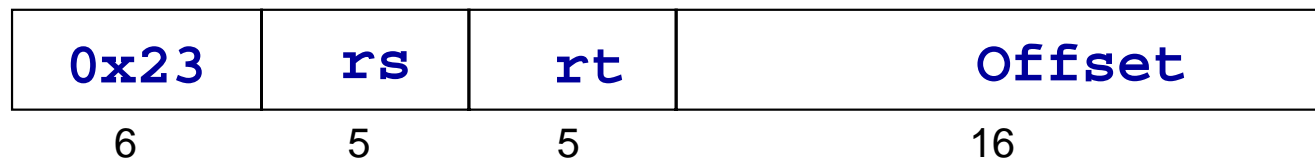
- Laden und Speichern von/an nicht-ausgerichtete Adressen
 - `lwl rt, address`
 - `lwr rt, address`
 - `ulh rdest, address` `ush rsrc, address`
 - `ulhu rdest, address`
 - `ulw rdest, address` `usw rsrc, address`
 - `ulhu rdest, address`

Lade-und Speicherbefehle

- Laden/Speichern von Bytes, Halbwörter und Wörter

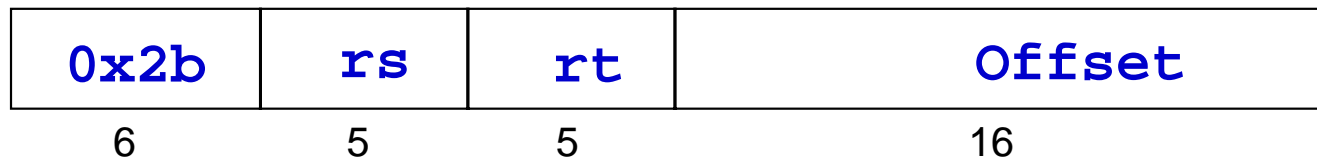
lw rt, address

Lade das 32-Bit Wort an der Adresse **address** ins Register **rt**



sw rt, address

Speichere das 32-Bit Wort im Register **rt** an der Adresse **address**



Beispiel

Lade- und Speicher-Befehle :

*Aufzugsadresse von A ist
in \$s3*

C-Code:

\$s2
`A[8] = h + A[8];`

MIPS code:

`lw $t0, 32($s3)` *# \$t0 = A[8]*
`add $t0, $s2, $t0` *# \$t0 = \$t0 + h*
`sw $t0, 32($s3)` *# \$t0 → A[8]*

Unterscheid zwischen **lb** und **lbu**

```
.data
result: .word 0x89abcdef 0x79abcdef

        .text          111 1000 1001
                        0 111
# Start des Hauptprogrammes

        .globl main
main:
    ...
    lbu $a0, result
    # $a0 enthaelt 0x00000089 0x00000079
    ...
    lb $a0, result
    # $a0 enthaelt 0xffffffff89 0x00000079
    ...
    jr $ra
```