KIT-Fakultät für Informatik
Prof. Dr.-Ing. Uwe D. Hanebeck, Prof. Dr.-Ing. Jörg Henkel

Musterlösungen zur Klausur

Digitaltechnik und Entwurfsverfahren (TI-1)

und

Rechnerorganisation (TI-2)

am 18. August 2021, 09:00 – 11:00 Uhr

Name:	Vorname:	Matrikelnummer:
Bond	James	007

Digitaltechnik und Entwurfsverfahren (TI-1)	
Aufgabe 1	10 von 10 Punkten
Aufgabe 2	10 von 10 Punkten
Aufgabe 3	8 von 8 Punkten
Aufgabe 4	9 von 9 Punkten
Aufgabe 5	8 von 8 Punkten

Rechnerorganisation (TI-2)	
Aufgabe 6	7 von 7 Punkten
Aufgabe 7	10 von 10 Punkten
Aufgabe 8	13 von 13 Punkten
Aufgabe 9	7 von 7 Punkten
Aufgabe 10	8 von 8 Punkten

Gesamtpunktzahl:	90 von 90 Punkten

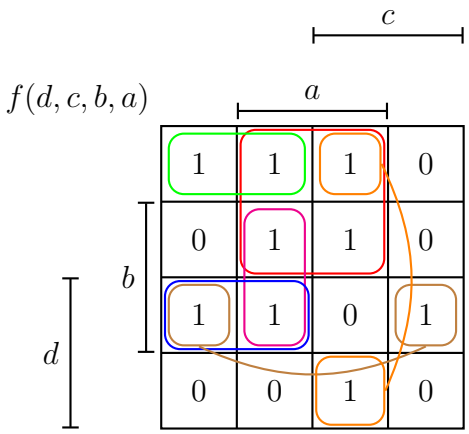
	Note: **1,0**

Aufgabe 1 *Schaltfunktionen* (10 Punkte)

1. Konjunktive Normalform (KNF): 2 P.

$$\begin{aligned} f(d,c,b,a) \;=\; & (d \vee c \vee \overline{b} \vee a) \wedge (d \vee \overline{c} \vee b \vee a) \wedge (d \vee \overline{c} \vee \overline{b} \vee a) \wedge \\ & (\overline{d} \vee c \vee b \vee a) \wedge (\overline{d} \vee c \vee b \vee \overline{a}) \wedge \\ & (\overline{d} \vee \overline{c} \vee b \vee a) \wedge (\overline{d} \vee \overline{c} \vee \overline{b} \vee \overline{a}) \end{aligned}$$

2. 4 P.



Primimplikanten:

$A: \overline{d}c\overline{b}$ $B: \overline{d}a$ $C: c\overline{b}a$

$D: \overline{c}ba$ $E: db\overline{a}$ $F: d\overline{c}b$

3. Disjunktive Minimalform von $f(d,c,b,a)$: 2 P.

$$\begin{aligned} f(d,c,b,a) &= A \vee B \vee C \vee E \vee D \\ (&= \overline{d}c\overline{b} \vee \overline{d}a \vee c\overline{b}a \vee db\overline{a} \vee \overline{c}ba\,) \end{aligned}$$

oder:

$$\begin{aligned} f(d,c,b,a) &= A \vee B \vee C \vee E \vee F \\ (&= \overline{d}c\overline{b} \vee \overline{d}a \vee c\overline{b}a \vee db\overline{a} \vee d\overline{c}b\,) \end{aligned}$$

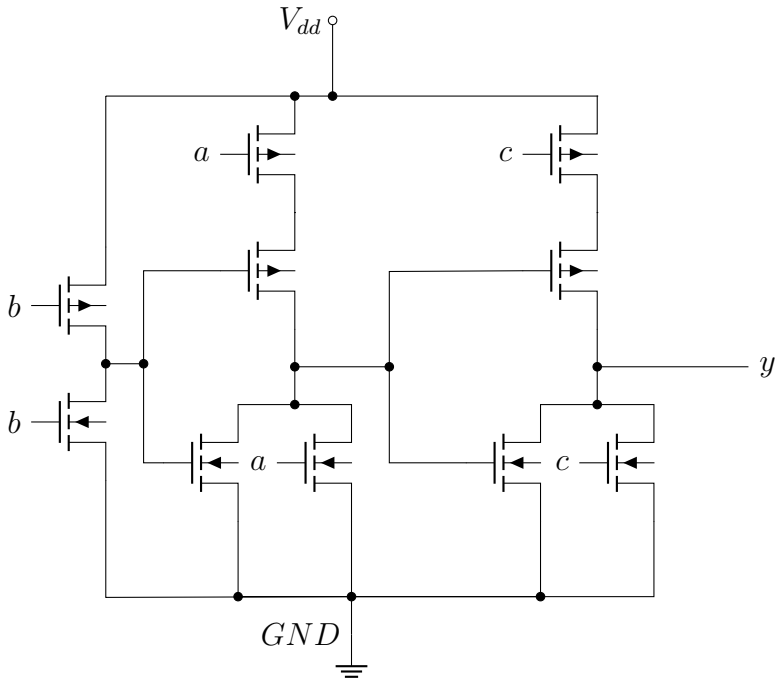
4. Zweistufige disjunktive Form von $g(c,b,a)$: 2 P.

$$\begin{aligned} g(c,b,a) &= \overline{c}b\overline{a} \vee \overline{c}ba \vee (cba \wedge cb\overline{a}) \\ &= \overline{c}b\overline{a} \vee \overline{c}ba \vee cba\overline{a} \\ &= \overline{c}b\overline{a} \vee \overline{c}ba \\ &= \overline{c}b \end{aligned}$$

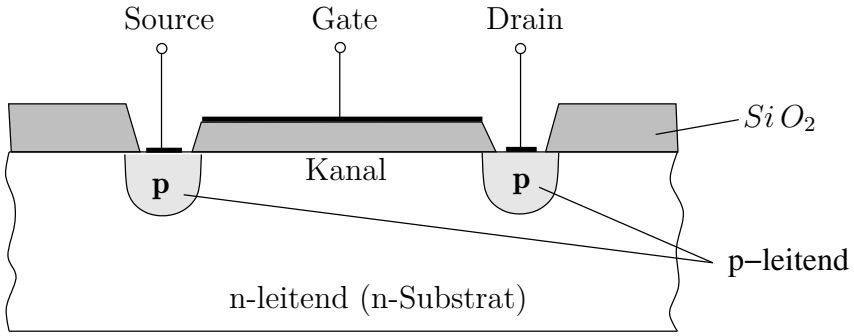
Aufgabe 2 *CMOS-Technologie* (10 Punkte)

1. Umgeformte Schaltfunktion und Transistor-Schaltbild: 5 P.

$$\begin{aligned} y &= k(c,b,a) = (a \vee \overline{b}) \wedge \overline{c} \\ &= \overline{\overline{(a \vee \overline{b}) \wedge \overline{c}}} \\ &= \overline{\overline{(a \vee \overline{b})} \vee c} \\ &= \mathrm{NOR}_2(\mathrm{NOR}_2(a,\overline{b}),c) \end{aligned}$$



2. Aufbau eines pMOS-Transistors: 3 P.



Selbstsperrender p-Kanal-MOSFET

3. Unterschied zwischen n-Kanal- und einem p-Kanal-MOSFET: 2 P.

   Der Unterschied zwischen den beiden Transistortypen besteht in der gegensätzlichen Dotierung der jeweiligen Zonen der Transistoren. Beim p-Kanal-MOSFET sind Source und Drain p-dotiert (siehe Aufgabenteil 2).

   n-Kanal-MOSFETs können eine logische Null gut und eine logische Eins schlecht durchschalten, bei p-Kanal-MOSFETs ist es umgekehrt. Daher werden n-Kanal-MOSFETs im n-Netz von CMOS-Schaltungen verwendet, um den Funktionswert Null durchzuschalten, und p-Kanal-MOSFETs im p-Netz, um den Funktionswert Eins durchzuschalten.

Aufgabe 3 *Laufzeiteffekte* (8 Punkte)

1. Totzeitmodell: 4 P.

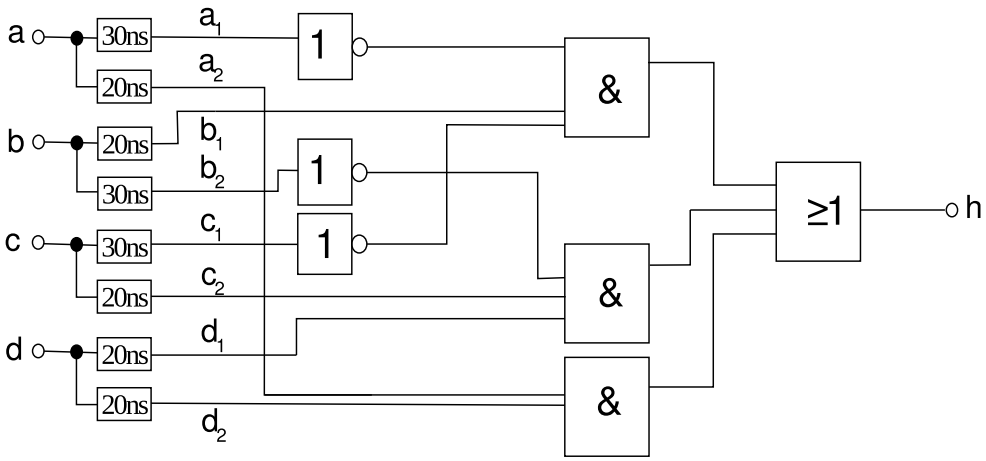
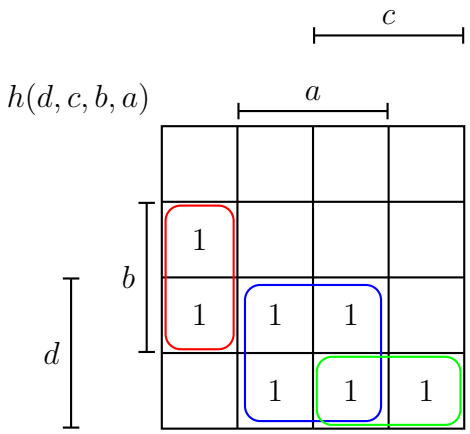


Abbildung 1: Totzeitmodell von $h(d, c, b, a)$

2. KV-Diagramm: 2 P.



3. Realisierung, die frei von allen statischen Struktuhasards: 2 P.

$$h(d, c, b, a) \quad = \quad da \vee \overline{c}b\overline{a} \vee dc\overline{b} \vee \mathbf{d\overline{c}b}$$

Begründung: Satz von Eichelberger.

Die Realisierung einer Schaltfunktion als die Disjunktion aller Primimplikanten ist frei von allen statischen Strukturhazards. Deshalb muss der in der angegeben Realisierung fehlende Primiplikant $\mathbf{d\,\overline{c}\,b}$ aufgenommen werden.

Aufgabe 4 *Schaltwerke* (9 Punkte)

1. Automatentyp: Mealy-Automat **1 P.**

   Begründung: die Ausgabe hängt sowohl vom Zustand als auch von der Eingabe ab.

2. Ansteuerfunktion: **2 P.**

$$\begin{aligned} t^t &= (x \wedge q)^t \vee (\overline{x} \wedge \overline{q})^t \\ &= x^t \leftrightarrow q^t \end{aligned}$$

   Zustandsübergangsgleichung:

$$\begin{aligned} q^{t+1} &= (q \wedge \overline{t})^t \vee (\overline{q} \wedge t)^t \\ &= q^t \overline{(x^t \leftrightarrow q^t)} \vee \overline{q}^t (x^t \leftrightarrow q^t) \\ &= q^t (x^t \overline{q}^t \vee \overline{x}^t q^t) \vee \overline{q}^t (x^t q^t \vee \overline{x}^t \overline{q}^t) \\ &= \overline{x}^t q^t \vee \overline{x}^t \overline{q}^t \\ &= \overline{x} \end{aligned}$$

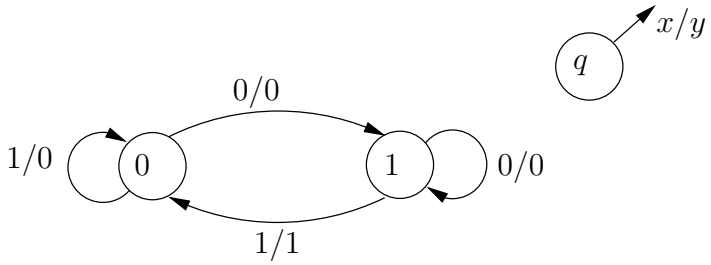
   Ausgabefunktion:

$$y^t = x^t q^t$$

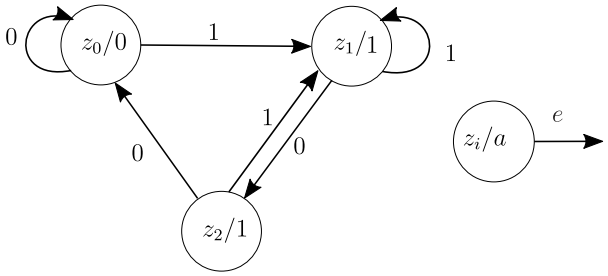
3. Automatengraph des Schaltwerks: **1 P.**
   Kodierte Ablauftabelle:

q^t	x^t	q^{t+1}	y^t
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

$\Rightarrow$



4. Automatengraph mit minimaler Anzahl Zustände: **3 P.**



5. Zustandsübergangsgleichungen: **2 P.**

q_1^t	q_0^t	e^t	q_1^{t+1}	q_0^{t+1}
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	–	–
1	1	1	–	–

$$\begin{aligned} q_1^{t+1} &= \overline{q}_1^t q_0^t e^t \\ q_0^{t+1} &= e^t \end{aligned}$$

Aufgabe 5 *Rechnerarithmetik* (8 Punkte)

1. Die Basen s und r: 1 P.

$$1 \cdot r^1 + 4 \;=\; 1 \cdot s^2 + 3 \cdot s^1 + 2 \;\rightarrow\; r = s^2 + 3 \cdot s - 2$$

Es existieren unendlich viele Lösungen:

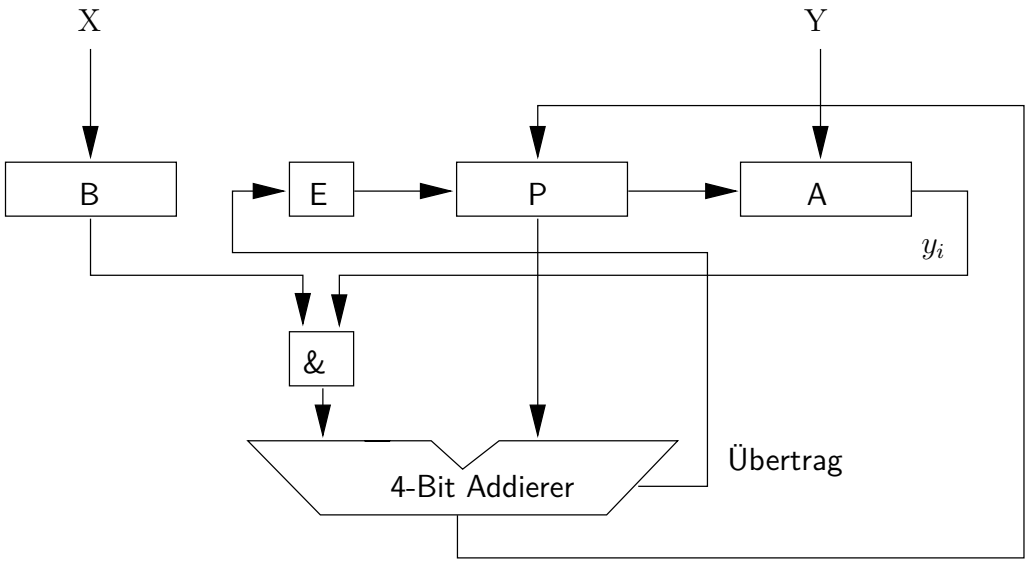
s	4	5	6	...
r	26	38	52	...

3 P.

2. (a) 14 binären Stellen: $2^{14} - 1 \;= 2^{10} \cdot 2^4 - 1 = 1024 \cdot 16 - 1 = 16384 - 1 = 16383$
   (b) 6 oktalen Stellen: $8^6 - 1 = (2^3)^6 - 1 = 2^{18} - 1 = 1024 \cdot 256 - 1 = 262143$
   (c) 4 hexadezimalen Stellen: $16^4 - 1 = 16^2 \cdot 16^2 - 1 = 256 \cdot 256 - 1 = 65535$

3. Ausnahmeregel für die Null im IEEE-754-Standard: Liegt an der Darstellung einer normalisierten Zahl, bei der die führende 1 nur implizit dargestellt wird. Auch wenn die Mantisse 0 ist, steht eine 1 vor dem Dezimalpunkt. 1 P.

4. Serieller Multiplizierer nach der PPS-Methode: 3 P.



Aufgabe 6 *Die Programmiersprache C* (7 Punkte)

1. Implementierung `addTwo(int *array, int n)`: 2 P.

```
void addTwo(int *array, int n)
{
    for(int i = 0; i < n; ++i)
    {
        *(array+i) = *(array+i) + 2;
    }
}
```

2. Implementierung `calcSum(int *array, int n)`: 2 P.

```
void calcSum(int *array, int n)
{
    int sum = 0;
    for(int i = 0; i < n; ++i)
    {
        sum += *(array+i);
    }
    *array = sum;
}
```

3. Implementierung `revArr(int *array, int n)`: 3 P.

```
void revArr(int *array, int n)
{
    int tmp;
    for(int i = 0; i < n/2; ++i)
    {
        tmp = *(array+i);
        *(array+i) = *(array+(n-1)-i);
        *(array+(n-1)-i) = tmp;
    }
}
```

Aufgabe 7 *MIPS-Assembler* (10 Punkte)

1. MIPS steht für: Microprocessor without Interlocked Pipeline Stages [1 P.]

2. Anzahl Bits für ein Befehlswort: [1 P.]
   Das MIPS-Befehlsformat sieht eine Breite von 32 Bit für ein Befehlswort vor.

3. Unterschied Maschinensprache und Assemblersprache: [2 P.]
   Maschinensprache ist eine Repräsentation von Anweisungen, die für einen Mikroprozessor unmittelbar verständlich sind.

   Assemblersprache ist eine symbolische Repräsentation der Maschinensprache, die für den Menschen verständlich und anschaulich ist.

4. Die 2 niedrigstwertigen Bits einer Wortadresse haben den Wert 0. [1 P.]

5. Laden von `0xF03D 0909` ins Register `$s0`: [2 P.]

```
lui  $s0, 0xF03D   # load upper immediate
ori  $s0, 0x0909
```

   oder auch

```
lui  $s0, 0xF03D
addi $s0, $s0, 0x0909
```

6. Inhalte der Zielregister: [3 P.]

Befehl	Zielregister = Wert (z.B. `$s6 = 0x0000 F00A`)
`ori $s1, $zero, 0x2021`	`$s1 = 0x0000 2021`
`sll  $s2, $s1, 1`	`$s2 = 0x0000 4042`
`slti $s3, $s2, 0x4043`	`$s3 = 0x0000 0001`
`sub  $s4, $s3, $s2`	`$s4 = 0xFFFF BFBF`

Aufgabe 8 *Pipelining* (13 Punkte)

1. Datenabhängigkeiten: 5 P.

   - Echte Abhängigkeiten *(True Dependence)*

     $S_1 \rightarrow S_2$ ($\$t0$) $S_2 \rightarrow S_3$ ($\$t1$) $S_3 \rightarrow S_4$ ($\$t2$)
     $S_2 \rightarrow S_4$ ($\$t1$)

   - Gegenabhängigkeiten *(Anti-Dependence)*:

     $S_1 \rightarrow S_2$ ($\$t1$) $S_2 \rightarrow S_3$ ($\$t2$) $S_2 \rightarrow S_4$ ($\$t0$)
     $S_1 \rightarrow S_3$ ($\$t2$)

   - Ausgabe-Abhängigkeiten *(Output Dependence)*:

     $S_1 \rightarrow S_4$ ($\$t0$)

2. Belegung der Register nach Ablauf des Programms und Zustand der Pipeline: 4 P.

Takt	IF	DE	OF	EX	WB	$t0	$t1	$t2
1	S1	—	—	—	—	3	6	8
2	S2	S1	—	—	—	3	6	8
3	S3	S2	S1	—	—	3	6	8
4	S4	S3	S2	S1	—	3	6	8
5	—	S4	S3	S2	S1	14	6	8
6	—	—	S4	S3	S2	14	-5	8
7	—	—	—	S4	S3	14	-5	12
8	—	—	—	—	S4	48	-5	12

   Anzahl der Takte: 8 Takte

3. Belegung der Register bei sequentieller Bearbeitung des Programms: 1 P.

$t0	$t1	$t2
72	6	12

4. Behebung der Pipelinekonflikte durch Einfügen von NOP-Befehlen: 3 P.

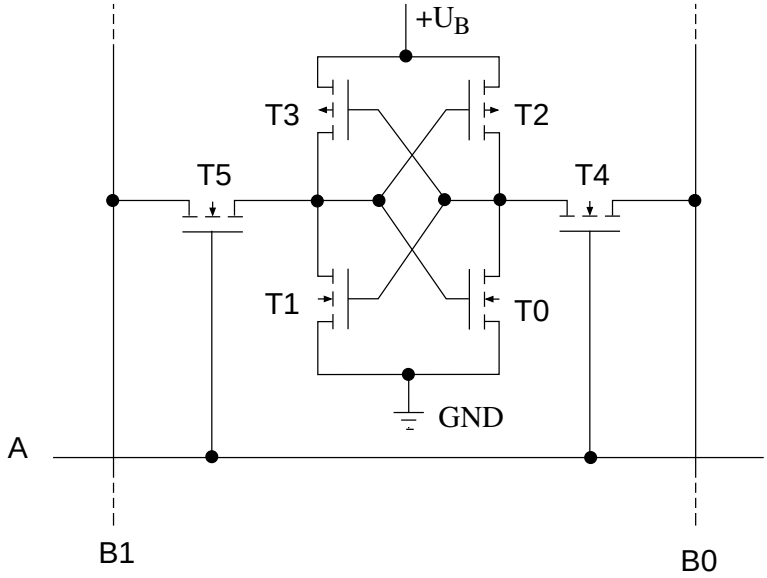
```
S1:  add $t0, $t1, $t2
NOP
NOP
S2:  sub $t1, $t0, $t2
NOP
NOP
S3:  add $t2, $t1, $t1
NOP
NOP
S4:  mul $t0, $t1, $t2s
```

Anzahl der Takte: 14 Takte

Aufgabe 9 *Speicherbausteine* (7 Punkte)

1. Problem: 1 P.
   Immer größer werdende Lücke zwischen Verarbeitungsgeschwindigkeit von Prozessoren und Zugriffsgeschwindigkeit der DRAM-Speicherchips des Hauptspeichers.

2. Aufbau 1-Bit Speicherzelle eines statischen RAM-Bausteins (SRAM):: 3 P.



3. Zugriffszeit *(access time)*: 1 P.

   Die maximale Zeitdauer, die vom Anlegen einer Adresse an den Speicher bis zur Ausgabe der gewünschten Daten vergeht.

   Zykluszeit *(cycle time)*: 1 P.

   Die minimale Zeitdauer, die zwischen zwei aufeinanderfolgenden Aufschaltungen von Adressen auf den Speicher vergehen muss.

4. Magnetische Speicher in einer Speicherhierarchie: In der unteren Ebene (Speichermedien mit großer Kapazität) 1 P.

Aufgabe 10 *Virtuelle Speicherverwaltung* (8 Punkte)

1. Unterteilung der virtuellen Adresse: 1 P.

2. Physikalische Adressen: 4 P.

Virtuelle		Physikalische	
Adresse	Seitennummer	Seitennummer	Adresse
512	0	1	$1 \cdot 4096 + 512 = \mathbf{4608}$
4095	0	1	$1 \cdot 4096 + 4095 = \mathbf{8191}$
4097	1	3	$3 \cdot 4096 + 1 = \mathbf{12289}$
4198	1	3	$3 \cdot 4096 + 102 = \mathbf{12390}$
8191	1	3	$3 \cdot 4096 + 4095 = \mathbf{16383}$
8192	2	–	*page fault*
8400	2	–	*page fault*
0	0	1	$1 \cdot 4096 + 0 = \mathbf{4096}$

3. Eine Beschleunigung der Adressumsetzung durch den *TLB* wird erst beim zweiten Zugriff auf eine Seite und solange die entsprechenden Einträge aus dem Seitentabellen-Verzeichnis und der Seitentabelle aus dem TLB nicht verdrängt wurden erreicht. 1 P.

4. Breite des *Tags*: 2 P.

   Seitengröße ist 4 KiByte ⇒ Byte-Offset ist 12 Bit breit.

   Der Tag ist dann $(n - 12)$ Bits breit