Department Informatik
Lehrstuhl für Informatik 3 (Rechnerarchitektur)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen
12.02.2018
Klausur zu

Grundlagen der Rechnerarchitektur und -organisation

....
Matrikelnummer Geburtsdatum

....
Name Vorname

- Es sind *keine* Hilfsmittel erlaubt!
- Legen Sie den Ausweis (mit Lichtbild!) griffbereit auf den Platz!
- Dieses Aufgabenheft umfasst 16 Seiten. Überprüfen Sie die Vollständigkeit!
- Gesondert beigelegte Blätter werden nicht bewertet.
- Schreiben Sie deutlich! Unleserliches wird nicht bewertet!
- Es darf nicht mit der Farbe rot geschrieben werden!
- Benutzen Sie dokumentenechte Stifte (kein Bleistift!)
- Offensichtlich falsche oder überflüssige Antworten können zu Punktabzug führen!
- Begründen Sie Ihre Antworten!

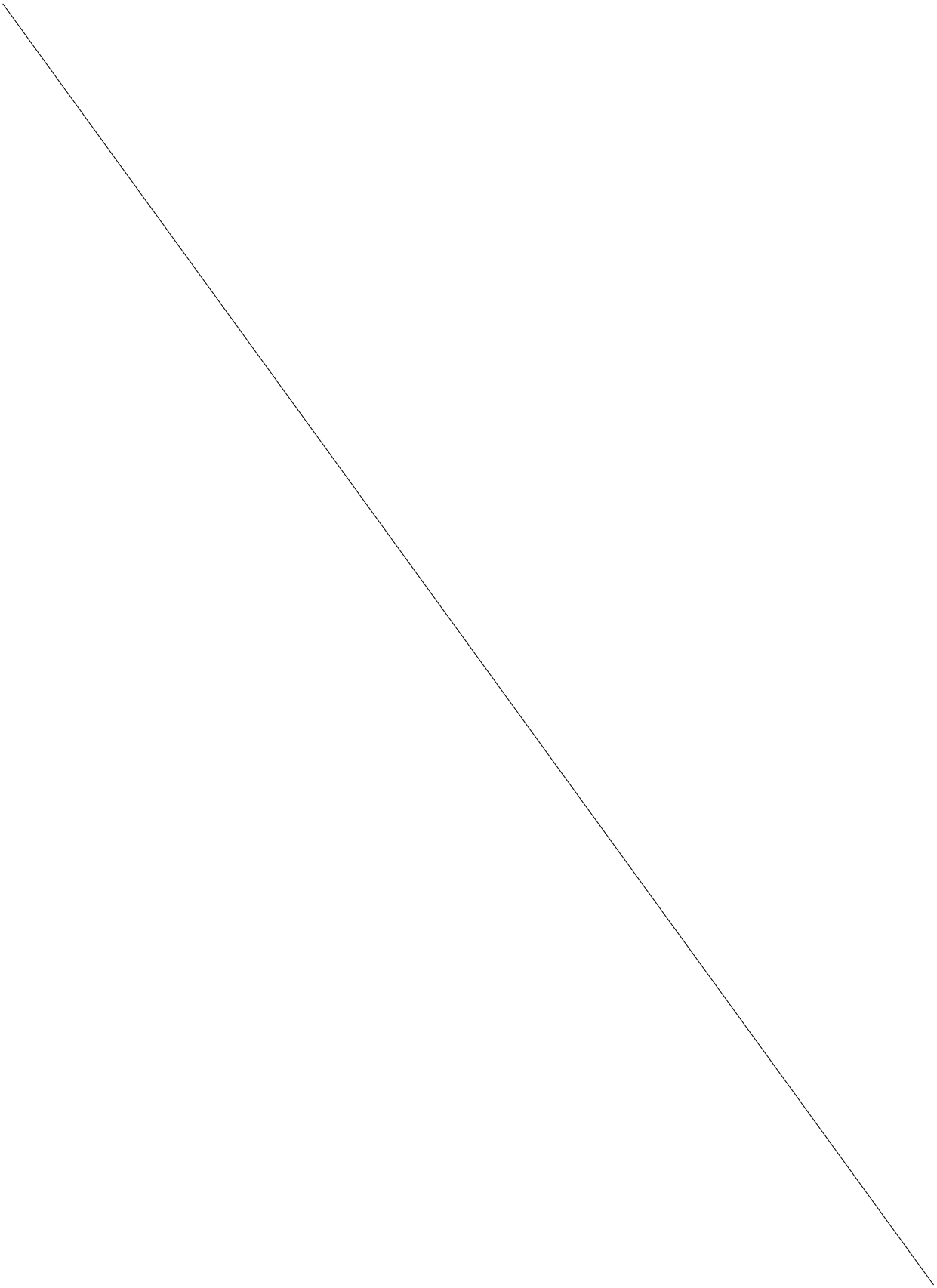
Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausurunterlagen
- die Kenntnisnahme der obigen Informationen.

Erlangen, den 12.02.2018 .....
(Unterschrift)

Aufgabe	1	2	3	4	5	6	7
max. Punktzahl	10	14	14	12	13	19	8
erreichte Punktzahl							

Summe	/90		
Bonus	/15		
Gesamt		Note	



Aufgabe 1: Verschiedenes — 10 Punkte

1. Erklären Sie die Begriffe *Befehlsarchitektur* und *Endoarchitektur*, wie sie durch Ungerer beschrieben werden. Geben Sie dazu außerdem ein Beispiel aus der Realität an, bei dem der Unterschied zwischen beiden Architekturen zum Tragen kommt. 2 Punkte

2. Nennen Sie die vier in der Vorlesung vorgestellten Klassen von Befehlssatzarchitekturen. 2 Punkte

3. Wofür steht die Abkürzung DMA? Wie funktioniert DMA? Welchen Vorteil bietet der Ansatz? 2 Punkte

4. Welche Adressierungsmodi zum Zugriff auf Peripherie gibt es? Was sind ihre Vor- und Nachteile? 2 Punkte

5. Welcher Mechanismus unterscheidet DDR2-SDRAM abgesehen von höheren Taktraten von DDR-SDRAM? 1 Punkt

6. Bei welcher Art von Multithreading spricht man von *logischen Prozessoren*? 1 Punkt

Aufgabe 2: Cache — 14 Punkte

Gegeben sei ein 4-fach assoziativer Datencache der Größe 128 Byte. Er bestehe aus insgesamt 4 Mengen. Es handelt sich um ein System mit 32 Bit Wort- und Adressbreite. Als Verdrängungsstrategie wird LRU verwendet.

1. Wie groß sind die Cachezeilen? (Mit Rechnung!) — 1 Punkt

2. Geben Sie an, wie die Adressen auf ihre für den Cache relevanten Komponenten aufgeteilt werden. Nutzen Sie dafür die folgende Tabelle. — 1 Punkt

3. Gegeben sei der folgende C-Code.

```
1  #define SIZE 16
2  // Ab Adresse 0x000
3  int a[SIZE];
4  // Ab Adresse 0x100
5  int b[SIZE] = { 0x12, 0x23, 0x34, 0x45, 0x56, 0x67, 0x78, 0x89,
6                  0x90, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77 };
7
8  void scalarvecmult(int scalar) {
9      for (int i = 0; i < SIZE; ++i) {
10         a[i] = scalar * b[i];
11     }
12 }
```

Es soll nun der Cachezustand für obigen Cache ermittelt werden, der nach der Ausführung der Funktion `scalarvecmult(1)` besteht. Vor dem Aufruf der Funktion ist der Cache im Initialzustand und alle Cachezeilen sind invalid.

Die Variablen `i` und `scalar` werden jeweils in einem Register gehalten und haben daher keinen Einfluss auf den Cache. Gleiches gilt für die Konstante `SIZE`. Es werden keine Codeoptimierungen durchgeführt. Nur der Datencache ist von Bedeutung.

Füllen Sie, wie gewohnt, die Tabelle auf der nächsten Seite aus und klammern Sie verdrängte Elemente ein. Die Daten werden in Big Endian Darstellung repräsentiert. Geben Sie in der Spalte der Daten die Werte aller in der Cachezeile enthaltenen Integerwerten an.

9 Punkte

Menge	Valid	Tag	Daten

4. Wie häufig wurde bei den Zugriffen in der obigen Situation die räumliche und wie häufig die zeitliche Lokalität ausgenutzt? 2 Punkte

5. Welche zwei Größen der Cacheorganisation können geändert werden, wenn die Cachezeilengröße erhöht, die Gesamtgröße des Caches aber gleich bleiben soll? 1 Punkt

Aufgabe 3: Codeumformungen — 14 Punkte

Gegeben sei folgende C-Funktion `function`, die auf einem MIPS32 Little Endian System ausgeführt werden soll.

```
1  int function(int *sources, int *targets, int size) {
2      int *src    = sources;
3      int *target = targets;
4      int sum     = 0;
5      while (NULL != src && NULL != target) {
6          for (int i = 0; i < size; ++i) {
7              int tmp = src[i];
8              target[i] = tmp;
9              if (tmp < 0) {
10                 continue;
11             }
12             sum += tmp;
13         }
14         target++;
15         src++;
16     }
17     return sum;
18 }
```

1. Wie viele Byte werden auf dem Stackframe für die Funktion `function` benötigt, wenn alle Variablen und Registerinhalte, die bei einem Funktionsaufruf von `function` auftreten, auf den Stack gelegt werden und die Stackframe-Definition aus der Vorlesung angewandt wird? Geben Sie dabei auch an, wofür wie viele Platz gebraucht wird. 2 Punkte

2. Nennen Sie zwei Möglichkeiten, einer Funktion Parameter zu übergeben. 1 Punkt

3. Formen Sie die Funktion `function` in die aus der Vorlesung bekannte If-Goto-Darstellung um, von der aus sie sich leicht in Assemblercode überführen lässt.

   Achten Sie darauf, dass alle Variablendefinitionen am Beginn der Funktion stehen und keine zusammengesetzten Bedingungen mehr bestehen. Führen Sie außerdem keine Optimierungen auf dem Code aus und achten Sie auf die Kurzschlusssemantik. 8 Punkte

Platz für If-Goto-Darstellung

4. Implementieren Sie in MIPS32-Assembler einen Arrayzugriff der Form `tmp = src[i]`. Dabei liege `tmp` an `0($sp)`, `i` an `4($sp)` und `src` an `8($sp)`. Die temporären Register stehen zur freien Verfügung. 3 Punkte

Aufgabe 4: Pipelining — 12 Punkte

Gegeben sei folgende Pipeline mit fünf Stufen. Sie verfüge über keine Mechanismen wie Sprungvorhersage, spekulative Ausführung, Forwarding oder Ähnlichem. Um eine korrekte Ausführung zu garantieren wird stattdessen die jeweilige Pipelinestufe angehalten bis der Konflikt behoben wurde.

Befehl Holen	**Operanden Holen**	**Befehl Ausführen**	**Speicherzugriff**	**Ergebnis Schreiben**
Wird immer jeden Takt durchgeführt	Befehlszähler setzen; Warten bei Hazards; Operanden lesen			Das Ergebnis steht erst nach dieser Stufe zur Verfügung

1. Nennen und erklären Sie zwei Arten von Hazards, die bei obiger Pipeline auftreten können. 2 Punkte

2. Wie hoch ist der asymptotische Speedup der obigen Pipeline (mit Herleitung!)? Warum wird er in der Realität nicht erreicht? 2 Punkte

3. Es soll nun das folgende Programm auf der Pipeline ausgeführt werden. Schreiben Sie, wie aus der Übung bekannt, in welcher Stufe sich welcher Befehl zu jedem Takt befindet. Nutzen Sie dafür die Tabelle auf der nächsten Seite. Um eine eindeutige Zuordnung der Instruktionen zu ermöglichen, sollen zudem die Zeilennummern der Instruktionen an die Mnemonics in der Tabelle angehängt werden. Außerdem sollen die Werte der Register `$a0`, `$t0` und `$v0` zu jedem Takt angegeben werden. (Hinweis: `$a1` wird auch geändert, in der Tabelle aber nicht aufgeführt!)

   Zellen in denen sich momentan kein Befehl, bzw. ein `nop` befindet, können leer gelassen werden. Alle Instruktionen am Ende des Programmes entsprechen auch `nop`s. Jeder Befehl muss alle Pipelinestufen durchlaufen.

   Der Einstiegspunkt in das Programm liegt beim Label `_start`.

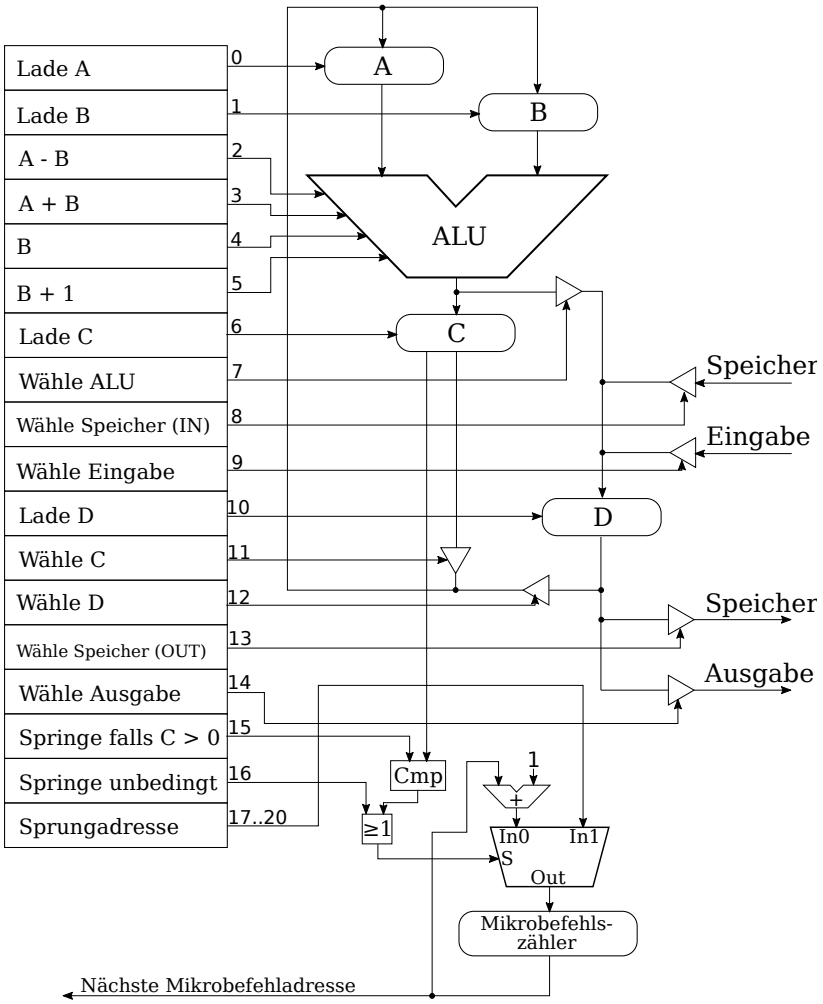
   Die Anzahl der benötigten Takte entspricht nicht der Tabellenlänge! 8 Punkte

```
1  remainder:
2      move $v0, $a0
3      subu $a0, $a0, $a1
4      slt  $t0, $a0, $zero
5      beq  $t0, $zero, remainder
6      jr   $ra
7
8  _start:
9      li $a0, 2
10     li $a1, 3
11     jal remainder
```

Takt	BH	OH	BA	MEM	ES	`$a0`	`$t0`	`$v0`
0	li9					?	?	?
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								
24								

Aufgabe 5: Mikroprogrammierung — 13 Punkte

Es sei folgendes Mikroprogrammwerk gegeben.



1. Implementieren Sie ein Mikroprogramm, das eine Operation „Count“ anbietet. Diese Operation erhält eine Zahl, die größer oder gleich null ist, von der Eingabe und zählt dann von null bis zu dieser Zahl, wobei jede Zahl (inklusive der null) hintereinander in der richtigen Reihenfolge auf der Ausgabe ausgegeben wird. Beispielsweise sollen die Zahlen *0, 1* und *2* ausgegeben werden, wenn eine *2* von der Eingabe gelesen wird. Der Inhalt aller Register ist zu Beginn null.

   Nutzen Sie die folgende Tabelle, um das Programm zu implementieren. Leer gelassene Felder entsprechen dem Wert null. Bei Sprüngen muss die Adresse explizit angegeben werden. Die 20. Steuerleitung ist das niederwertigste Bit. Vermeiden Sie unnötige Befehle und Sprünge.

   Tragen Sie zu jedem Befehl auch eine Erklärung dazu ein (wie bekannt aus der Übung, z.B. A-B → C).

   (Die Tabellenlänge muss nicht der Programmlänge entsprechen!) — 8 Punkte

Adr.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Erklärung
0																						
1																						
2																						
3																						
4																						
5																						
6																						
7																						
8																						
9																						
10																						

2. Woher stammt die initiale Adresse des Mikroprogramms? 1 Punkt

3. Bei welcher Art von Befehlssatzarchitekturen werden Mikroprogrammwerke bevorzugt eingesetzt? 1 Punkt

4. Welchen Bestandteil des von Neumann'schen Befehlszyklus müsste man mikroprogrammiert abarbeiten, um Instruktionen beliebiger Befehlssatzarchitekturen unterstützen zu können? 2 Punkte

5. Welchen Vorteil hat ein dynamisch ladbares Mikroprogramm gegenüber einem fest verdrahteten? 1 Punkt

Aufgabe 6: Paging — 19 Punkte

Um auf einem 32-Bit-System dennoch mehr als 2^{32} Byte adressieren zu können, wird oftmals eine MMU verwendet, die 32 Bit lange virtuelle Adresse auf längere physikalische Adressen abbilden kann.

Eine solche MMU sei im Folgenden gegeben, die eine 32 Bit virtuelle Adresse auf eine 36 Bit physikalische Adresse abbilden kann. Es werden 4 KiB große Tabellen und Seiten verwendet. Die Adressumsetzung erfolgt zweistufig.

Die Bits der 4 Byte großen Zeile einer Seitentabelle sind dabei wie folgt aufgeteilt:

31–8	7–4	3	2	1	0
A	I	C	X	W	P

Dabei stehen die Buchstaben für:

P: Present-Bit

W: Write-Enable-Bit

X: Execute-Enable-Bit

C: Cache-Disable-Bit

I: Diese Bits können ignoriert werden (0 setzen!)

A: Die höherwertigen Adressbits der physikalischen Adresse

Bei der Seitentabelle der ersten Stufe haben die Bit W, C und X keine Bedeutung und müssen, genauso wie die Bits unter I auf 0 gesetzt werden. Die gespeicherten physikalischen Adressbits ergeben die höherwertigen 24 Bits der 36 Bit großen physikalischen Adressbusbreite.

1. Erstellen Sie eine Seitenhierarchie, mit deren Hilfe die folgenden Adressraumbereiche abgebildet werden können. Nutzen Sie dafür die Tabellen auf der nächsten Seite. Tragen Sie dazu in der ersten Spalte die Zeilennummer und in die zweite Spalte den Inhalt der Zeile jeweils in hexadezimaler Darstellung ein. Zeilen, deren Present-Bit 0 ist, müssen nicht angegeben werden.

   Achten Sie darauf, dass die Zugriffsbits für den gegebenen Einsatzzweck korrekt gesetzt sind und setzen Sie das *Page Directory Basis Register* (PDBR, beinhaltet die oberen 32 Bit der physikalischen Adresse) auf einen sinnvollen Wert. 10 Punkte

virtuelle Adressen	**physikalische Adressen**	**Einsatzzweck**
`0x20000 - 0x21fff`	`0x100000000 - 0x100001fff`	Code
`0xe55000 - 0xe55fff`	`0xb8000 - 0xb8fff`	Grafikkartenframebuffer
`0xffca8000 - 0xffcaafff`	`0xba64a000 - 0xba64cfff`	Daten
`0xffffe000 - 0xffffffff`	`0xe243000 - 0xe244fff`	Stack

PDBR:

Adresse: `0xf0000000`

Adresse: `0xf0001000`

Adresse: `0xf0002000`

Adresse: `0xf0003000`

Adresse: `0xf0004000`

Adresse: `0xf0005000`

2. Wie viel zusätzlicher Speicher (in Byte) wird von obiger MMU gegenüber dem einstufigen Paging benötigt, wenn der komplette virtuelle Adressraum auf physikalische Adressen abgebildet werden soll? 1 Punkt

3. Warum nutzt man dennoch mehrstufige Seitenhierarchien? 1 Punkt

4. Warum kann ein Programm nicht einfach eine neue Tabellenhierarchie laden und damit auf eigentlich verbotene Adressbereiche zugreifen? 1 Punkt

5. Was ist ein TLB und welchen Zweck hat er? 1 Punkt

6. Nennen und erklären Sie die unterschiedlichen Arten der Fragmentierung. Bei welchen Speicherschutzmechanismen treten sie jeweils auf? 3 Punkte

7. Wo befindet sich die MMU relativ zu den Caches? Begründen Sie Ihre Antwort! 2 Punkte

Aufgabe 7: Speicherlayout — 8 Punkte

Gegeben sei folgender C-Code, welcher auf einem MIPS32 Little Endian System unter Verwendung der MIPS32 SystemV ABI zum Einsatz kommen soll.

```
1 struct extension {
2     char letter;
3     int  count;
4     char format;
5     int  page;
6 };
7 struct extension instance = { 71, 256, 20, 27 };
```

1. Geben Sie jedes einzelne Byte eines Speicherauszugs von `instance` in hexadezimaler Schreibweise an. 3 Punkte

2. Wie viele Byte benötigt `struct extension array[8]`, wenn die Elemente des `struct extension` so umsortiert werden, dass der Speicherverbrauch auf dem gegebenen System minimal wird. Geben Sie sowohl das umsortierte `struct extension` als auch den Rechenweg an! 2 Punkte

3. Warum werden Daten im Speicher ausgerichtet? Welchen Nachteil hätte man bei Verwendung des obigen `struct extension`, wenn man nicht auf korrekte Ausrichtung geachtet hätte (und damit der SystemV ABI widersprochen hätte)? 2 Punkte

4. Worauf muss man beim Datenaustausch mit der Peripherie oder anderen Rechnern achten? 1 Punkt

Zusätzlicher Platz