Department Informatik
Lehrstuhl für Informatik 3 (Rechnerarchitektur)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen
08.02.2016
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 18 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!
- Offensichtlich falsche oder überflüssige Antworten können zu Punktabzug führen!
- Begründen Sie Ihre Antworten!

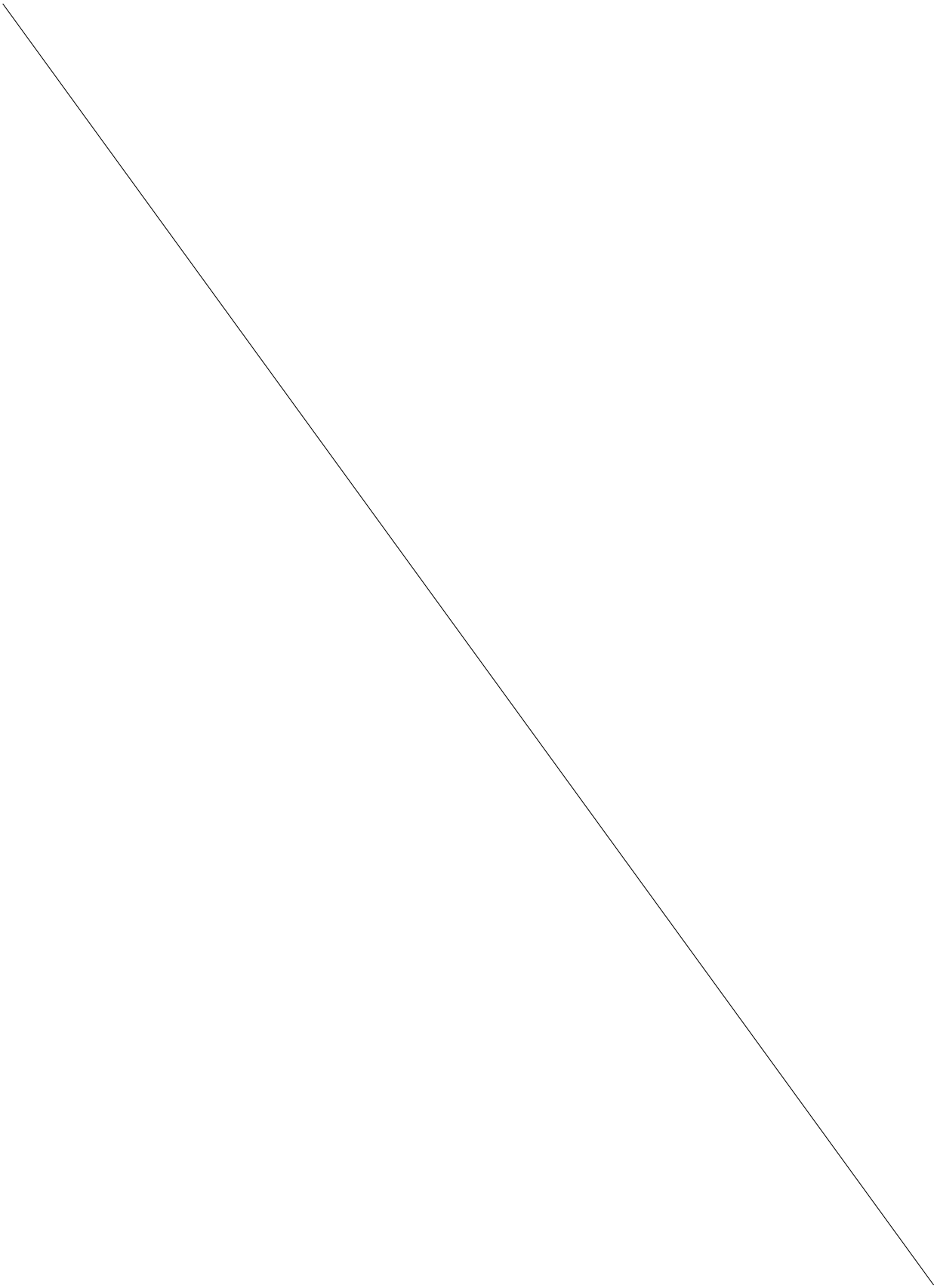
Durch meine Unterschrift bestätige ich

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

Erlangen, den 08.02.2016 .....
(Unterschrift)

Aufgabe	1	2	3	4	5	6	7	8
max. Punktzahl	7	15	13	10	11	14	9	11
erreichte Punktzahl								

Summe	/90		
Bonus	/15		
Gesamt		Note	



Aufgabe 1: Allgemein — 7 Punkte

Welche der folgenden Aussagen sind wahr, welche falsch? Kreuzen Sie an.

Punktabzug bei falschen Antworten!

1. Ein-/Ausgabe — 2 Punkte

wahr	falsch	
☐	☐	Lesen von Isolated-I/O-Geräten erfordert andere Maschinenbefehle als aus dem RAM.
☐	☐	Der CPU-Cache soll Zugriffe auf I/O-Geräte beschleunigen.
☐	☐	Memory-Mapped-I/O-Zugriffe sind langsamer als Isolated-I/O.
☐	☐	DMA erlaubt I/O-Geräten direkten Speicherzugriff.

2. Befehlssatzarchitekturklassen — 3 Punkte

wahr	falsch	
☐	☐	Stackbasierte Architekturen besitzen keine Register.
☐	☐	Akkumulatorbasierte Architekturen besitzen genau ein Rechenregister.
☐	☐	Der Assemblerbefehl `push` ist ein eindeutiges Zeichen für eine stackbasierte Architektur.
☐	☐	ALU-Befehle haben bei stackbasierten Architekturen keine expliziten Operanden.
☐	☐	Register-Speicher-Architekturen haben im Allgemeinen eine höhere Code-Dichte als Akku-basierte.
☐	☐	Speicheradressen sind im Maschinenbefehl aufwändiger zu codieren als Register.

3. Speicherschutz — 2 Punkte

wahr	falsch	
☐	☐	Paging soll Speicherzugriffe beschleunigen.
☐	☐	Durch Paging kann Prozessen mehr Arbeitsspeicher zur Verfügung gestellt werden, als physikalisch vorhanden ist.
☐	☐	Segmentierung erlaubt die Auslagerung von Arbeitsspeicher auf der Festplatte
☐	☐	Paging erlaubt die Auslagerung von Arbeitsspeicher auf der Festplatte.

Aufgabe 2: Assembler — 15 Punkte

Gegeben sei das folgende AT&T-Assemblerprogramm für einen 32-Bit Intel-x86-Prozessor:

```
1  .text
2  ggt:
3    pushl %ebx
4    movl 8(%esp), %eax
5    movl 12(%esp), %ebx
6    cmpl %ebx, %eax          # set_flags(%eax-%ebx)
7    je LRet
8    jg LA
9  LB:
10   subl %eax, %ebx
11   jmp LNext
12 LA:
13   subl %ebx, %eax          # %eax = %eax - %ebx
14 LNext:
15   pushl %ebx
16   pushl %eax
17   call ggt
18   addl $8, %esp
19 LRet:
20   popl %ebx
21   ret
22
23 .globl main
24 main:
25   pushl $4
26   pushl $2
27   call ggt
28   addl $8, %esp
29   ret
```

1. Was versteht man unter „caller-save“ und „callee-save“ in Bezug auf Register? 2 Punkte

2. Wieviel Byte auf dem Stack benötigt das Programm zur Laufzeit maximal?

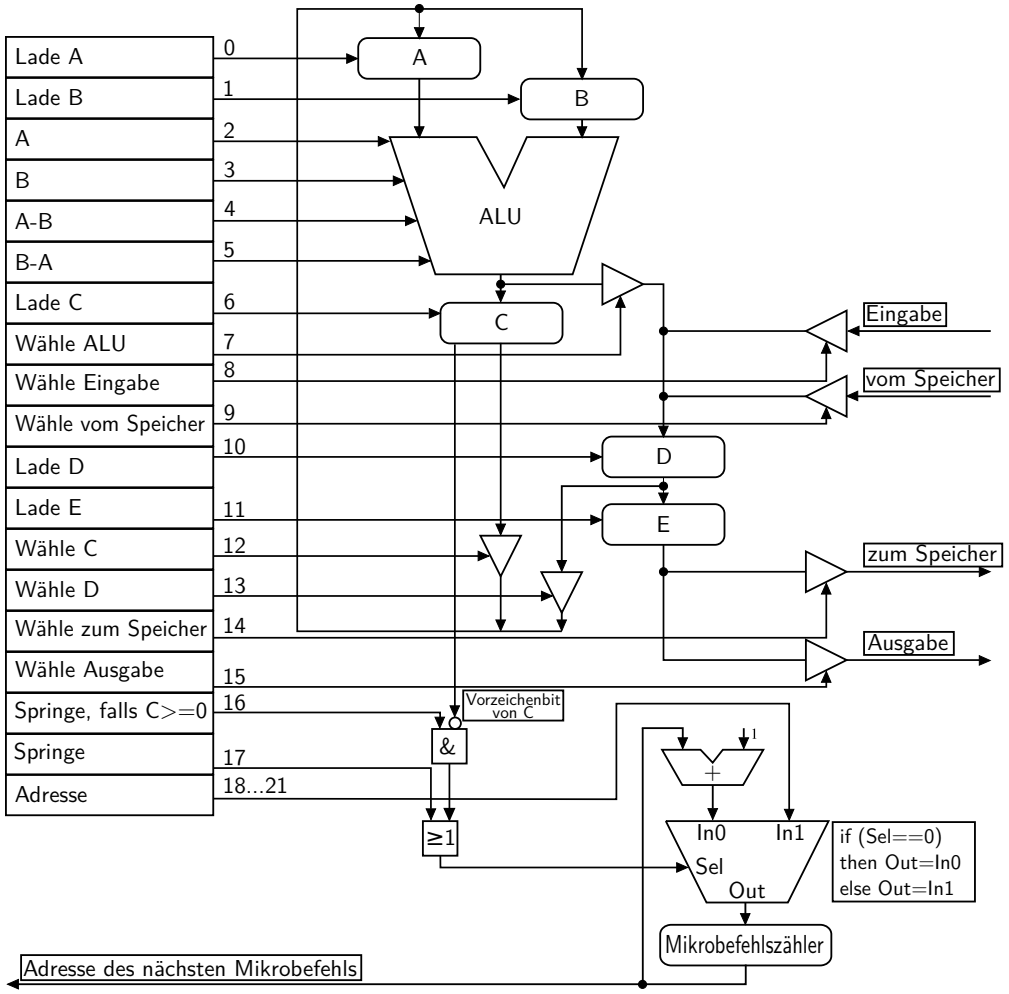
   Die Rücksprungadresse der Funktion `main` befindet sich zu Beginn auf dem Stack und soll mitgezählt werden. *(Begründung!)* 6 Punkte

3. Wie könnte der ursprüngliche C-Code ausgesehen haben? Achten Sie auf Parameter und Rückgabewerte. Alle Variablen sind vom Typ `int`. 6 Punkte

4. Welchen Zahlenwert gibt die Funktion `main` zurück? 1 Punkt

Aufgabe 3: Mikroprogrammierung — 13 Punkte

Gegeben sei der folgende Teil einer CPU:



Die CPU soll um den Makrobefehl `Reg[A] mod Eingabe` → `Reg[A]` erweitert werden. `Reg[A]` und `Eingabe` sind natürliche Zahlen echt größer Null.

1. Überlegen Sie sich einen Programmablauf in Pseudocode, der sich direkt als Mikroprogramm umsetzen lässt. 1 Punkt

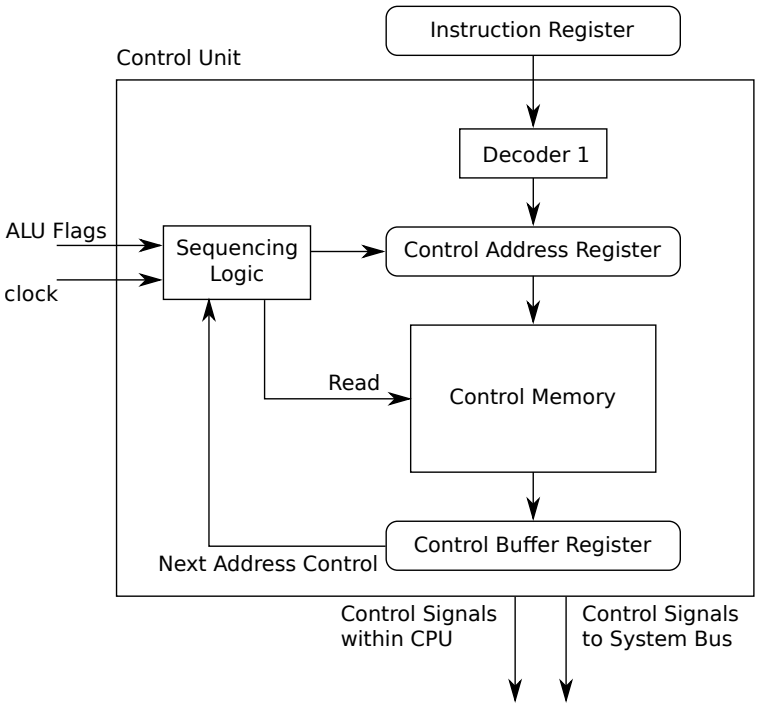
2. Implementieren Sie den Befehl als Mikroprogramm für obige CPU. Vermeiden Sie unnötige Befehle! Alle Register dürfen überschrieben werden. Verwenden Sie die folgende Tabelle und tragen Sie jeweils auch die Bedeutung (z. B. D $\rightarrow$ E) in die Spalte „Erklärung“ ein. Leer gelassene Steuerleitungen entsprechen dem Wert „0“. Das niederwertigste Bit des Sprungziels entspricht Steuerleitung 21.

   Eine Fehlerbehandlung ist nicht nötig.

   (Die Tabellenlänge entspricht nicht der erwarteten Mikroprogrammlänge!) 8 Punkte

Adr.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	Erklärung
0																							
1																							
2																							
3																							
4																							
5																							
6																							
7																							
8																							
9																							
10																							
11																							
12																							
13																							
14																							
15																							

Gegeben sei folgende Darstellung der Ablaufsteuerung einer mikroprogrammierten Architektur:



3. Nennen Sie jeweils die Aufgaben der folgenden Einheiten: 3 Punkte

- Sequencing Logic:

- Control Memory:

- Instruction Register:

4. Welche Form der Mikroprogrammierung wird hier verwendet und woran ist dies erkennbar? 1 Punkt

Aufgabe 4: Umformung — 10 Punkte

Formen Sie die folgende Funktion so in if-goto-Darstellung um, dass sie sich möglichst leicht in Assembler übersetzen lässt.

```
1  void sort(int skip, int *A, int length) {
2    for (int n = length; n > 1; n--) {
3      int i = 0;
4      while (true) {
5        if (skip == 0 && i < n-1) {
6          if (A[i] > A[i+1]) {
7            A[i] = A[i] ^ A[i+1];
8            A[i+1] = A[i+1] ^ A[i];
9            A[i] = A[i] ^ A[i+1]; } }
10       else break;
11       i++; } } }
```

Einfache Arrayzugriffe in der Form `A`[i] können direkt verwendet werden. Vergleichsoperanden und komplexere Array-Indizes müssen vorher berechnet werden. Verändern Sie dabei weder die Auswertungsreihenfolge noch die eigentlichen Operationen. Achten Sie auf die Kurzschluss-Semantik!

Aufgabe 5: Parallelverarbeitung — 11 Punkte

Gegeben sei eine Architektur mit einer fünfstufigen Pipeline, bestehend aus *Befehl holen und dekodieren* (BH-BD), *Operanden holen* (OH), *Befehl ausführen Teil 1* (BA1) und *Teil 2* (BA2) und *Ergebnis sichern* (ES).

1. Wie hoch ist der theoretisch maximale Speedup dieser Pipeline?
   (Keine Berechnung/Begründung nötig.) — 1 Punkt

2. Führen Sie das folgende Programm auf obiger CPU aus:

```
1  Lstart:
2    movl $2, %eax          % Reg[eax]=2
3    movl $1, %ecx
4  Lcond:
5    cmpl %eax, %ecx        % Flags(Reg[ecx] - Reg[eax])
6    je Lend
7    jg Lgreater
8  Lless:
9    subl %ecx, %eax        % Reg[eax] = Reg[eax] - Reg[ecx]
10   jmp Lcond
11 Lgreater:
12   subl %eax, %ecx
13   jmp Lcond
14 Lend:
```

   Die Pipeline verfüge über keine erweiterten Mechanismen wie Sprungvorhersage, spekulative Ausführung, Forwarding, etc. Um dennoch ein korrektes Ergebnis zu garantieren, soll statt dessen die Ausführung weiterer Instruktionen so lange verzögert werden, bis kein Konflikt mehr vorliegt, bzw. das Sprungziel definitiv bekannt ist, allerdings auch nicht länger.

   Tragen Sie in der Tabelle auf der nächsten Seite in jeder Stufe *die Instruktion und die Zeile* des dort gerade ausgeführten Befehls ein (z.B. `movl2`). Geben Sie außerdem den aktuellen Registerinhalt von `eax` und `ecx` an, soweit bekannt. Felder mit `nop` können leer gelassen werden.

   Bei Sprungbefehlen soll der Befehlszähler so früh wie möglich aktualisiert werden. *Unbedingte Spünge* werden bereits in `BH-BD` ausgewertet. Der Befehl `cmpl` aktualisiert das Statusregister zur Sprungauswertung erst in der Phase *ES*, *bedingte Sprünge* werten es in der *OH*-Phase aus.

   Jeder Befehl muss die gesamte Pipeline durchlaufen. Lassen Sie am Ende die Pipeline leer laufen.

   Die Tabellenlänge entspricht nicht der erwarteten Ausführungszeit! — 10 Punkte

Takt	BH-BD	OH	BA1	BA2	ES	`eax`	`ecx`
0	movl2					?	?
1	movl3	movl2				?	?
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							

Aufgabe 6: Paging — 14 Punkte

Eine CPU biete eine Memory-Management-Unit mit folgenden Eigenschaften:

- zweistufige Adresstabellen
- je 1024 Einträge zu je 4 Bytes in den Tabellen
  - Bit 31-12: höherwertige Bits der physikalischen Adresse
  - Bit 11-4: unbenutzt
  - Bit 3: Cache-disable-Bit
  - Bit 2: Execute-enable-Bit
  - Bit 1: Write-enable-Bit
  - Bit 0: Present-Bit
- Pages zu je 4 KiB Größe

Folgendes soll für ein Programm gelten:

- im virtuellen Adressbereich `0x00 03 00 00` bis `0x00 03 ff ff` kann auf nicht ausführbare Daten zugegriffen werden,
- im virtuellen Adressbereich `0x00 40 00 00` bis `0x00 4f ff ff` ist der Speicher der Grafikkarte eingeblendet,
- im virtuellen Adressbereich `0x00 e0 00 00` bis `0x00 e0 0f ff` kann auf schreibgeschützten Programmcode zugegriffen werden,
- alle anderen Bereiche sollen Zugriffsfehler auslösen.

Freier Arbeitsspeicher ist ab der physikalischen Adresse `0x00 c0 00 00` verfügbar, die Grafikkarte ist ab Adresse `0x00 01 e0 00` eingeblendet.

Ergebnisse dürfen hexadezimal angegeben werden.

1. Zeichnen Sie eine Tabellenhierarchie, die obige Anforderungen erfüllt!

   (Bearbeitung auf der nächsten Seite!) — 10 Punkte

Fortsetzung von Aufgabe 6.1.

2. Was muss beim Prozesswechsel mit dem PDBR und dem TLB gemacht werden, um den Speicherschutz nicht zu gefährden? 1 Punkt

3. Welche Informationen speichert ein TLB für obige Konfiguration? 1 Punkt

4. Warum ist es für bestimmte Zwecke sinnvoll, denselben physikalischen Speicher in mehreren Prozessen einzublenden? Geben Sie ein Beispiel! 2 Punkte

Aufgabe 7: Peripherie — 9 Punkte

1. Erklären Sie stichpunktartig die Adressierungsmodi Memory-Mapped I/O (eingeblendeter Speicher) und Isolated I/O (isolierte Ein-/Ausgabe). 2 Punkte

2. Welche drei Formen der Datenübertragung zwischen Ein-/Ausgabegeräten und der CPU oder dem Arbeitsspeicher wurden behandelt? Erklären Sie jeweils kurz das Prinzip. 6 Punkte

3. Was versteht man unter dem Burst-Modus bei Datenübertragungen? Was ist der Vorteil? 1 Punkt

Aufgabe 8: Cache — 11 Punkte

Eine 32-Bit CPU verwende einen 4-fach assoziativen Cache mit insgesamt 8 Blöcken. Jeder Block umfasse 8 Byte. Der Cache sei zu Beginn leer.

Ein Programm liest nacheinander *jeweils 4 Byte* von den folgenden Adressen:
`0xc, 0x20, 0x60, 0x1a8, 0x1bc, 0x7c, 0x1c, 0x8`

Der relevante Speicherbereich hat folgenden Inhalt:

```
0x00 00 00 00: 34 2f 6d a4 56 96 9b ec  a5 f0 73 51 a2 4b 39 5a
0x00 00 00 10: 95 14 0f e0 66 48 de 8e  4a b7 cb 38 77 f2 44 f8
0x00 00 00 20: 00 a4 d5 31 38 32 81 f4  b1 c3 de 33 e0 9f fb ff
0x00 00 00 30: 2e 54 60 9d aa 85 ba ad  28 ad 92 d7 b7 10 1f 21
...
0x00 00 00 60: 48 e6 87 60 a5 bf 20 f6  e6 24 cd 4b 40 79 c7 c8
0x00 00 00 70: ee dd d7 23 2d ab ab e4  6c 8c bd 8f cc 10 1f 7b
...
0x00 00 01 a0: 82 e1 30 15 b7 0c ac 90  e3 44 7b 6b ef e4 00 63
0x00 00 01 b0: 61 9f 2a c4 b6 35 77 a6  aa 92 0c c7 e1 55 a5 eb
0x00 00 01 c0: 22 18 8b 5b d9 6a 0c ad  de f4 db bb b4 43 ee 5d
0x00 00 01 d0: e2 cb 16 8e 32 ec f3 15  72 9a e7 bd be 0b 9d 42
```

1. Geben Sie die Aufteilung der Adressen in die Cache-relevanten Bestandteile an. 2 Punkte

2. Tragen Sie den Inhalt des Caches nach obigen Zugriffen in die Tabelle auf der nächsten Seite ein. Es wird jeweils der Eintrag verdrängt, der am längsten nicht verwendet wurde (LRU).

   Nummerieren Sie die Mengen entsprechend deren Index! Es genügt, jeweils das erste und letzte Byte der Daten im Cache-Block einzutragen. 8 Punkte

Menge	Block	Valid	Tag	Daten
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			

3. Wie viele Conflict Misses treten bei obigen Zugriffen auf? 1 Punkt

Zusätzlicher Platz