Department Informatik
Lehrstuhl für Informatik 3 (Rechnerarchitektur)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen
17.09.2014
Klausur zu

Grundlagen der Rechnerarchitektur und -organisation

.....	.....	.....	.....
Matrikelnummer	Geburtsdatum	Vorname	Name

- 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 17.09.2014 .....
(Unterschrift)

Ich bin damit einverstanden, dass mein Prüfungsergebnis der Klausur unter Angabe der Matrikelnummer veröffentlicht wird.

Erlangen, den 17.09.2014 .....
(Unterschrift)

Aufgabe	1	2	3	4	5	6	7
max. Punktzahl	9	11	17	16	10	13	14
erreichte Punktzahl							

Summe	/90		
Bonus	/15		
Gesamt		Note	

Aufgabe 1: Allgemein — 9 Punkte

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

Punktabzug bei falschen Antworten!

1. Speicher — 2 Punkte

wahr	falsch	
< >	< >	Shift-Operationen sind leicht in Hardware zu realisieren.
< >	< >	Auf einem 64-Bit-Rechner müssen *alle* Variablen auf durch 8 teilbare Adressen ausgerichtet werden.
< >	< >	Die Verwendung von Alignment kann Speicherzugriffe beschleunigen.
< >	< >	Bei Little-Endian steht das niederwertigste Bit an der niedrigsten Adresse.

2. Assemblerprogrammierung — 3 Punkte

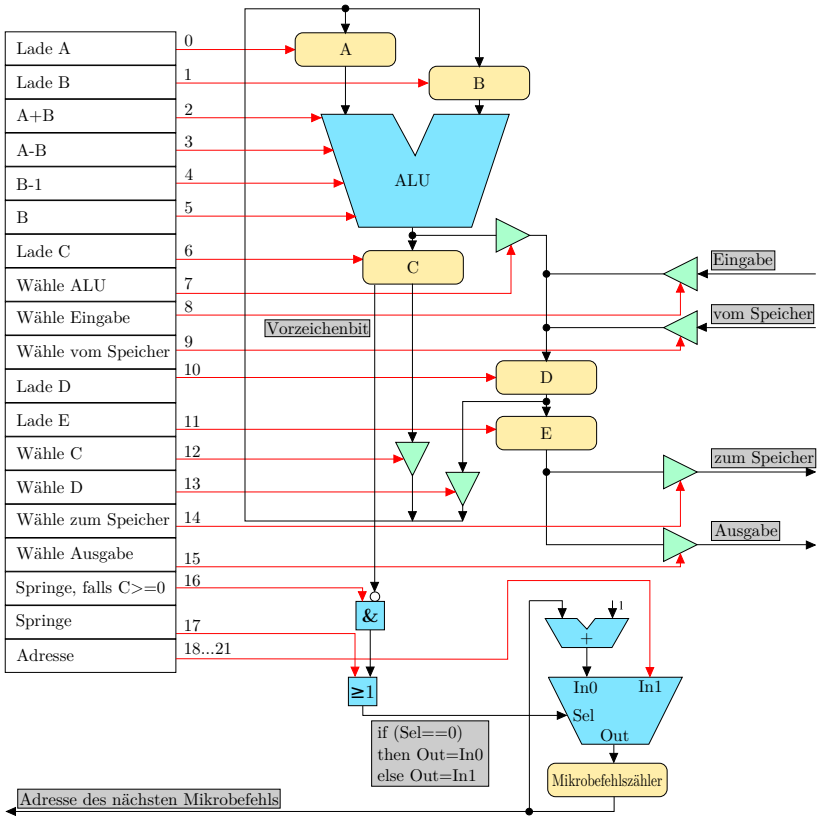
wahr	falsch	
< >	< >	Alle Kontrollstrukturen können auf `if-goto`-Befehle abgebildet werden.
< >	< >	Unbedingte Sprungbefehle auf der *x86*-Architektur werten das Flags-Register aus.
< >	< >	Parameterübergabe über Register ist schneller als über den Stack.
< >	< >	Der Stack eines Prozesses ist ein Teil des Hauptspeichers.
< >	< >	Zur Realisierung von Unterprogrammaufrufen ist zwingend ein Stack erforderlich.
< >	< >	Beim Betreten einer Interrupt Service Routine muss die CPU immer alle Universalregister sichern.

3. Speicherschutz — 4 Punkte

wahr	falsch	
< >	< >	Paging und Segmentierung schließen sich gegenseitig aus.
< >	< >	Durch Segmentierung oder Paging kann mehr Speicher zur Verfügung gestellt werden, als physikalisch vorhanden ist.
< >	< >	Mit Hilfe der MMU können Daten vor unerlaubtem Zugriff geschützt werden.
< >	< >	Die MMU rechnet physikalische Adressen in Segmente um.
< >	< >	Die Seitengröße beim Paging hängt in erster Linie von der Adressbusbreite ab.
< >	< >	Eine virtuelle Adresse ist nur gültig, wenn in der Seitentabelle das *Present-Bit* gesetzt ist.
< >	< >	Mehrstufige Seitentabellen sorgen für eine effizientere Speicherausnutzung.
< >	< >	Bei der Verwendung mehrstufiger Seitentabellen kann die Berechnung der physikalischen Adresse besonders schnell erfolgen.

Aufgabe 2: Mikroprogrammierung — 11 Punkte

1. Gegeben sei der folgende Teil einer CPU:



Schreiben Sie ein Mikroprogramm für obige CPU, das so lange (positive und negative) Werte von der Eingabe liest und in den Speicher schreibt, bis der Wert '0' gelesen wurde. Vermeiden Sie unnötige Befehle!

Verwenden Sie die folgende Tabelle und tragen Sie jeweils auch die Bedeutung (z.B. D → E) ein.

Leer gelassene Steuerleitungen entsprechen dem Wert „0", bei Sprüngen muss die Zieladresse explizit angegeben werden! Das niederwertigste Bit des Sprungziels entspricht Steuerleitung 21.

Hinweis:
Überlegen Sie sich zunächst den Ablauf in Pseudocode (wird nicht bewertet)! 8 Punkte

(Die Tabellenlänge entspricht nicht der erwarteten Mikroprogrammlänge!)

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																							

2. Was unterscheidet eine mikroprogrammierte von einer „fest-verdrahteten“ Ablaufsteuerung der Befehlsausführung eines Prozessors? 2 Punkte

3. Warum sind bedingte und unbedingte Sprünge auf Mikrobefehlsebene nötig? 1 Punkt

Aufgabe 3: Speicherverwaltung — 17 Punkte

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

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

   Folgendes soll für ein Programm erfüllt sein:

   - im virtuellen Adressbereich `0x00 00 70 00` bis `0x00 00 7F FF` soll schreibgeschützter Programmcode stehen,
   - ab der virtuellen Adresse `0x00 40 00 00` soll der Video-Speicher der VGA-Karte eingeblendet sein,
   - im virtuellen Adressbereich `0x00 AC 00 00` bis `0x00 AC FF FF` sollen die Programmdaten gespeichert sein,
   - alle anderen Bereiche sollen Zugriffsfehler auslösen.

   Physikalisch befinde sich der Video-Speicher an den Adressen `0x00 0B 80 00` bis `0x00 0B FF FF`, freier Arbeitsspeicher sei ab der physikalischen Adresse `0x00 00 10 00` ausreichend vorhanden.

   Skizzieren Sie eine mögliche Page-Tabelle, die obige Bedingungen erfüllt! Achten Sie auf die korrekte Vergabe der Bits für die Flags!

   (Bearbeitung auf der nächsten Seite!) — 12 Punkte

Fortsetzung von Aufgabe 3:

2. Was passiert, wenn die CPU auf die virtuelle Adresse `0x00 00 78 88` schreibend zugreift? Begründen Sie Ihre Antwort! 1 Punkt

3. Welche Informationen werden in einem TLB für obige Konfiguration gespeichert? 1 Punkt

4. Ist der *arithmetische* Aufwand für die Bestimmung der physikalischen Adresse bei Segmentierung oder Paging größer? Begründen Sie! 2 Punkte

5. Wie viel Byte physikalischen Speicher kann eine CPU mit N-Bit breiten Speicheradressen maximal direkt adressieren, wenn Daten byteweise adressierbar sein sollen? 1 Punkt

Aufgabe 4: Cache — 16 Punkte

Gegeben seien die folgenden Hochsprachenfunktionen:

```
1  void vert(const double in[], double out[], size_t size) {
2    for (int i = 0; i < size; ++i) {
3      for (int j = 0; j < size; ++j) {
4        out[j * size + i] = in[j * size + i] * 0.5;
5      }
6    }
7  }
8
9  void hort(const double in[], double out[], size_t size) {
10   for (int i = 0; i < size; ++i) {
11     for (int j = 0; j < size; ++j) {
12       out[i * size + j] = in[i * size + j] * 0.5;
13     }
14   }
15 }
```

1. Erklären Sie, welche von beiden Funktionen auf einem Rechner mit einem Cache mit hinreichend großer Blockgröße wahrscheinlich schneller ausgeführt werden wird!

   Die Arrays `in, out` sind jeweils deutlich größer als der Cache. 2 Punkte

2. Warum ist es generell sinnvoll, getrennte Caches für Befehle und Daten zu verwenden? 1 Punkt

3. Was würde im Hinblick auf das Caching passieren, wenn die Elemente der Arrays nicht korrekt ausgerichtet gespeichert wären? 1 Punkt

4. Überlegen Sie sich eine allgemeine Funktion f, die der (korrekt ausgerichteten) Anfangsadresse eines Arrayelements `addr_i` den Cache-Block, bzw. die Menge zuordnet, wo das Datum im Cache gespeichert wird.

   Folgendes gilt für den Cache:
   Assoziativitatsgrad n, Gesamtgröße Nutzdaten in Byte s, Cache-Blockgröße (Cache-Zeile) in Byte b.

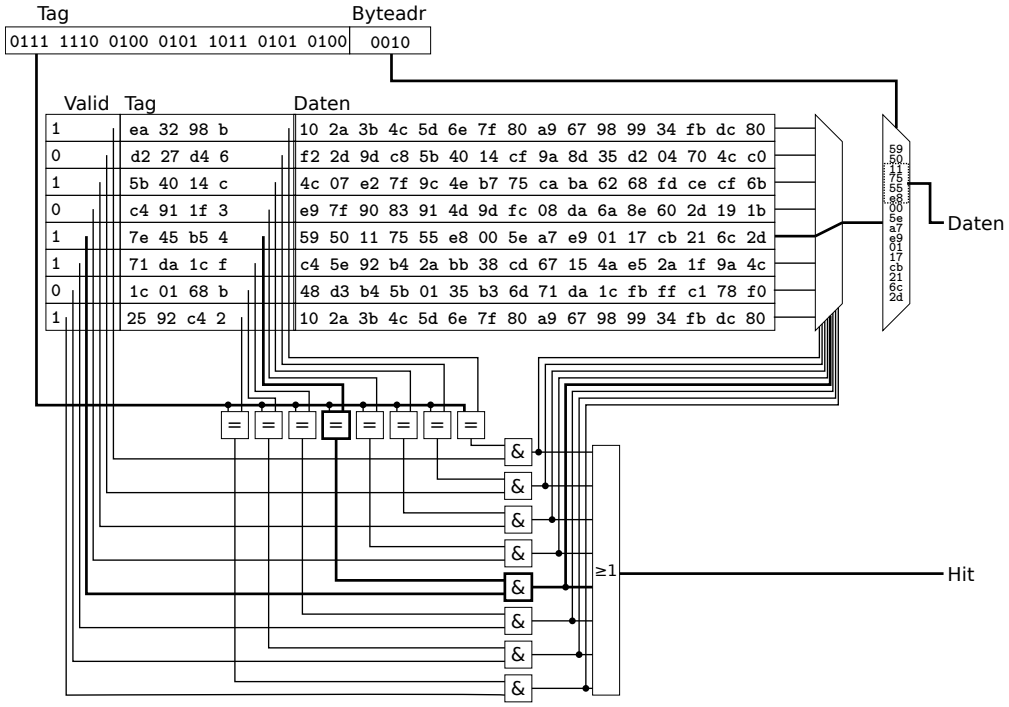
   Für n, s, b sind Werte von Zweierpotenzen.

   Hinweis:
   Überlegen Sie sich zunächst, wie viele Blöcke und wie viele Mengen der Cache besitzt.

   4 Punkte

$f(\text{addr\_i}) =$

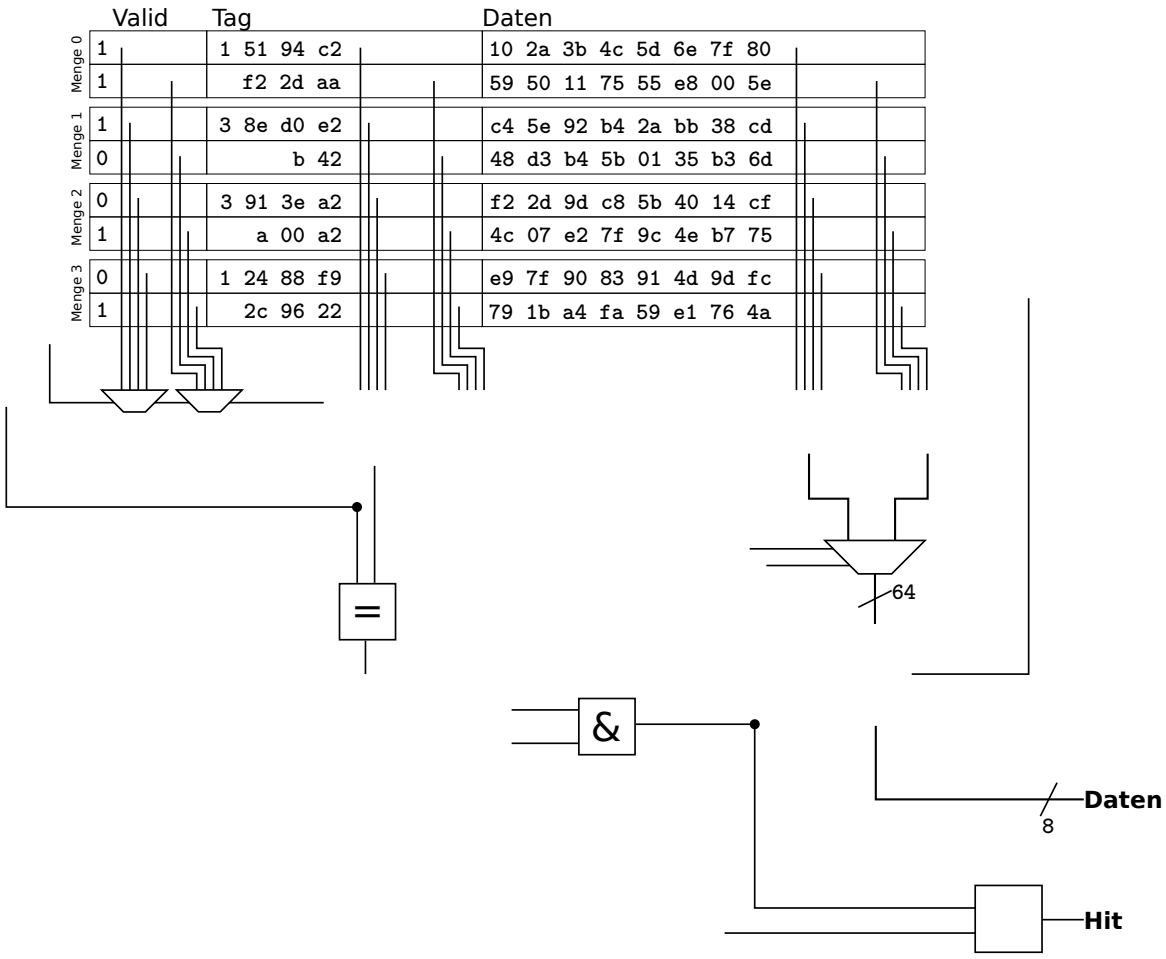
5. Welche Organisationsform besitzt folgender Cache? Begründen Sie! 2 Punkte



6. Vervollständigen Sie die Auslese-Logik (Daten-, Hitsignal) für den folgenden 2-fach-Assoziativ-cache.

   Verwenden Sie dazu einfache Logikgatter (Und &, Oder ≥1), Vergleicher = und Multiplexer ▷.

   Teilen Sie die Adresse in die entsprechenden Bereiche auf und geben Sie deren jeweilige Breite an! 6 Punkte



Aufgabe 5: Umformung — 10 Punkte

Konvertieren Sie die folgende Hochsprachen-Funktion in semantisch äquivalenten Hochsprachen-Code, so dass sich dieser möglichst leicht in Assembler-Code für eine akkumulatorbasierte Architektur umwandeln lässt!

Wandeln Sie dazu die Kontrollstrukturen in `if-goto`-Darstellung um.

Arithmetische Ausdrücke müssen der Form `akku = akku <op> operand;` genügen,
`akku` ist eine Variable, die das Akkumulatorregister repräsentiert,
`op` $\in \{+, -, *, /, <, >, <=, >=\}$,
`operand` kann eine Variable oder eine Konstante sein.

Vergleiche erfolgenden immer zwischen dem Akkumulator und einer Variablen/Konstante. Arrayzugriffe können direkt übermommen werden. Wertzuweisungen sollen nicht „wegoptimiert“ werden!

```
1
2  int test(int x, int n)
3  {
4      int akku;
5
6      int ret = 0;
7      int i;
8      for (i = 0; i < n; i++) {
9          int diff2;
10         diff2 = (wert[i] - x) * (wert[i] - x);
11         if (100 < diff2) {
12             ret = 1;
13             break;
14         }
15     }
16     return ret;
17 }
```

(Bearbeitung auf der nächsten Seite.)

Fortsetzung von Aufgabe 5:

Aufgabe 6: Assembler — 13 Punkte

1. Erklären Sie, *ob* und ggf. *wie* sich die folgenden Bestandteile eines Hochsprachenprogramms in einem fertig übersetzten und ausführbaren Maschinenprogramm (ohne extra Debuginformationen) wiederfinden. 5 Punkte

- Sprungmarken (Labels)

- Funktionsaufrufe mit Parametern

- Globale Variablen

- Lokale Variablen (auf dem Stack)

- Kommentare

2. Was versteht man unter einem Framepointer und wozu wird er verwendet? 2 Punkte

3. Welche zwei Möglichkeiten gibt es für ein (Assembler-)Programm, Ein-/Ausgabegeräte zu adressieren? Sind dazu immer spezielle Instruktionen nötig? 2 Punkte

4. Wie und warum unterscheidet sich die Code-Dichte bei RISC- und CISC-basierenden Prozessoren? 2 Punkte

5. Warum ist es für eine bestimmte Klasse von Algorithmen problematisch, wenn die Rücksprungadresse in einem Register gespeichert wird? Welche sind das und wie könnte das Problem gelöst werden? 2 Punkte

Aufgabe 7: Prozessorarchitekturen — 14 Punkte

1. Leiten Sie den asymptotischen Speedup für eine s-stufige Pipeline her! 2 Punkte

2. Welchen Vorteil hat eine superskalare Architektur gegenüber einer VLIW-basierten? Warum? 2 Punkte

3. Welche Arten von Multithreading wurden behandelt? Nennen Sie je einen Vorteil! 3 Punkte

4. Was muss ein Anwendungsprogrammierer tun, um Multithreading nutzen zu können? 1 Punkt

Gegeben sei das folgende Programmfragment (AT&T-Syntax):

```
1 | movl $1, %eax
2 | movl $2, %ecx
3 | movl $3, %edx
4 | addl %edx, %ecx
5 | addl %ecx, %eax
```

5. Ermitteln Sie den tatsächlichen Speedup einer Architektur mit einer einfachen Pipeline ohne Sprungvorhersage und Forwarding mit den Stufen

   1. Befehl holen und decodieren
   2. Operanden holen
   3. Ausführen
   4. Rückschreiben

   bei Ausführung des obigen Programmcodes gegenüber einer streng sequentiellen Befehlsausführung ohne Pipelining.

   Die Pipeline ist zu Beginn leer. Das Programmfragment ist fertig, wenn der letzte Befehl die Pipeline verlässt. 5 Punkte

6. Warum ist der Speedup nicht höher? 1 Punkt

Zusätzlicher Platz