Department Informatik
Lehrstuhl für Informatik 3 (Rechnerarchitektur)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen
02.02.2015
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 02.02.2015 .....
(Unterschrift)

Aufgabe	1	2	3	4	5	6	7
max. Punktzahl	10	12	13	17	14	11	13
erreichte Punktzahl							

Summe	/90		
Bonus	/15		
Gesamt		Note	

Aufgabe 1: Allgemein — 10 Punkte

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

Punktabzug bei falschen Antworten!

1. Assemblerprogrammierung — 4 Punkte

wahr	falsch	
< >	< >	Ein unmittelbar adressierter Operand wird normalerweise während der Befehlsholphase aus dem Arbeitsspeicher geladen.
< >	< >	Der Framepointer beschleunigt den Zugriff auf den Arbeitsspeicher.
< >	< >	CPU-Register können über Arbeitsspeicheradressen angesprochen werden.
< >	< >	Eine stackbasierte Architektur besitzt mehrere Register.
< >	< >	Zur Verwendung eines Stacks sind zwingend spezielle CPU-Instruktionen erforderlich.
< >	< >	Auf Register-Register-Architekturen können sich Operanden für arithmetische Instruktionen im Speicher befinden.
< >	< >	Eine akkumulatorbasierte Architektur besitzt mehrere Rechenregister.
< >	< >	Ein Befehl mit Register-Direkt adressiertem Operanden wird schneller ausgeführt als mit Register-Indirektem.

2. Arbeitsspeicher — 3 Punkte

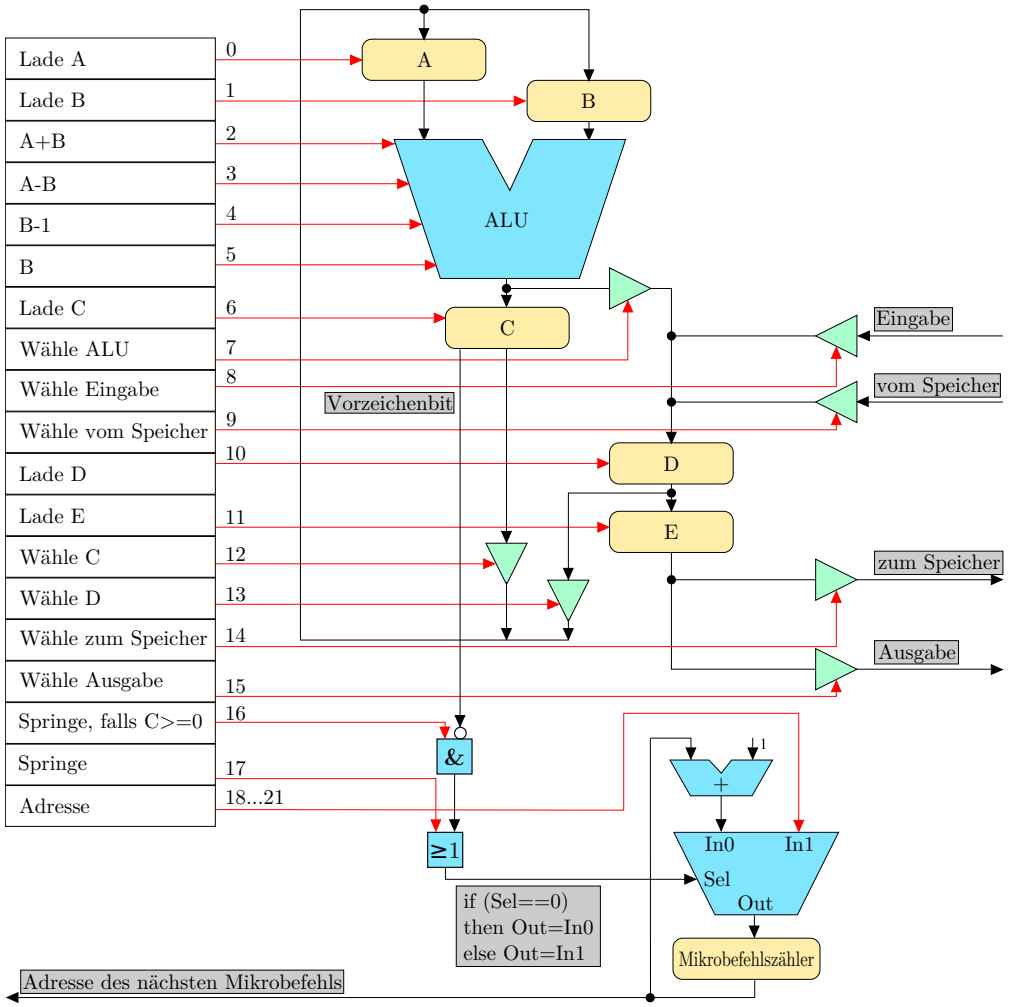
wahr	falsch	
< >	< >	Der Effektivtakt bei DDR-SDRAM ist doppelt so hoch wie bei normalem SDRAM.
< >	< >	Interleaving erhöht den Speichertakt.
< >	< >	Speicherzellen sind bei DDR3-SDRAM immer höher getaktet als bei DDR2.
< >	< >	Burst-Transfers sind nötig, um den Prefetch von DDRx-SDRAM ausnutzen zu können.
< >	< >	Benachbarte Daten sollten sich bei Interleaving möglichst auf gleichen Speicherbänken befinden, sofern sie nicht ohnehin gemeinsam geladen werden können.
< >	< >	Die Blockgröße des Caches beeinflusst die Häufigkeit von Burst-Transfers zwischen CPU und RAM.

3. Speicherschutz — 3 Punkte

wahr	falsch	
< >	< >	Segmentierung erlaubt es, Speicher von Prozessen vor ungewolltem Zugriff zu schützen.
< >	< >	Speicherschutz kann alleine durch das Betriebssystem realisiert werden.
< >	< >	Wenn Paging verwendet wird, sind ggf. mehrere Speicherzugriffe nötig, um ein Datum aus dem Speicher zu laden.
< >	< >	Paging erlaubt das Auslagern von Arbeitsspeicherinhalt auf der Festplatte.
< >	< >	Segmentierung erlaubt das Auslagern von Arbeitsspeicherinhalt auf der Festplatte.
< >	< >	Bei Verwendung von Segmentierung entscheidet das Betriebssystem bei jedem Speicherzugriff, ob der Zugriff gewährt wird oder nicht.

Aufgabe 2: Mikroprogrammierung — 12 Punkte

Gegeben sei der folgende Teil einer CPU:



1. Erklären Sie konkret anhand der obigen Darstellung, welche Arten von Sprüngen unterstützt werden und wie sie realisiert werden. 4 Punkte

2. Schreiben Sie ein Mikroprogramm für obige CPU, das eine natürliche Zahl $n > 0$ von der Eingabe liest, schrittweise dekrementiert und jeweils genau einmal ausgibt, einschließlich der 0. In der Ausgabe soll also $n, n-1, n-2, ..., 0$ erscheinen.

   Vermeiden Sie unnötige Befehle! Eine Fehlerbehandlung ist nicht nötig.

   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”, 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																							

Aufgabe 3: Parallelverarbeitung — 13 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. `subl12`). 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! — 8 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							

3. Unterscheiden sich Hochsprachenprogramme für VLIW-basierte und superskalare Architekturen?

   Von wem wird jeweils die Parallelisierung umgesetzt? 2 Punkte

4. Erklären Sie das Prinzip von Simultanem Multithreading und welchen Vorteil es gegenüber Zeitscheiben-, bzw. Ereignisgesteuertem Multithreading besitzt. 2 Punkte

Aufgabe 4: Speicherschutz — 17 Punkte

1. Auf einem 64-Bit System soll der Arbeitsspeicher durch mehrstufiges Paging verwaltet werden. Virtuelle und physikalische Adressen seien 48 Bit breit. Die Einträge in den Tabellen umfassen jeweils 8 Byte. Eine Seite sei 4 KiB groß, ebenso die einzelnen Verwaltungstabellen.

   Bestimmen Sie die Tiefe der Tabellenhierarchie rechnerisch! 4 Punkte

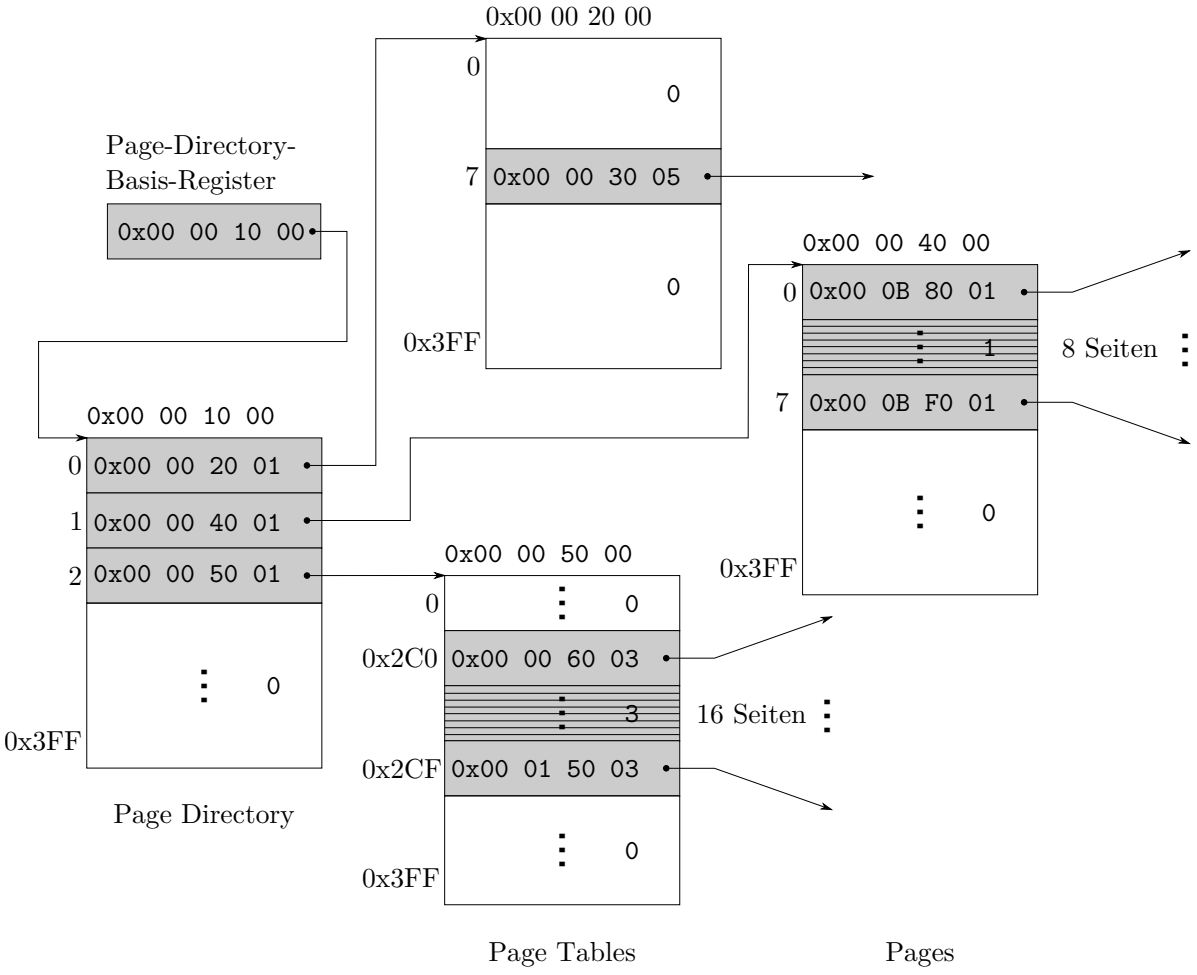
2. Wie könnten auf einem System mit 4 KiB großen Pages und zweistufiger Seitenverwaltung (z.B. Intel x86, siehe unten) sog. *Huge-Pages* realisiert werden, also Seiten, die einen langen zusammenhängenden (virtuellen/physikalischen) Adressbereich von 4 MiB verwalten?

   Wie könnte hierbei die virtuelle Adresse in Seitenindex und Offset im Vergleich zur normalen Seitengröße von 4 KiB aufgeteilt werden? 4 Punkte

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: Execute-Enable-Bit (EE)
  - Bit 1: Write-Enable-Bit (WE)
  - Bit 0: Present-Bit (P)
- Pages zu je 4 KiB Größe

Der virtuelle Adressraum ist wie folgt aufgebaut. Gültige Einträge sind grau hinterlegt. Bei Bereichen ist jeweils der erste und letzte Eintrag gegeben, dazwischenliegende Einträge haben die gleichen Flags und liegen sequentiell im Speicher.



3. Überlegen Sie sich für die folgenden Codeabschnitte, ob bei bzw. im Anschluß an ihre Ausführung Exceptions auftreten. Geben Sie ggf. Art und Ursache an. 4 Punkte

a)
```
call 0x00 AC 2C 1C
```

b)
```
movl $0x00 40 20 F0, %eax
movl $5, (%eax)
```

c)
```
movl $0x00 40 20 F0, %eax
movl (%eax), %ecx
```

d)
```
incl 0x00 AD 00 00
```

4. Warum wird das Present-Bit nicht im TLB gespeichert? 1 Punkt

5. Erklären Sie das grundlegende Prinzip von Segmentierung! 4 Punkte

Aufgabe 5: Cache — 14 Punkte

1. Was versteht man unter räumlicher und zeitlicher Lokalität typischer Programme? 2 Punkte

2. Eine CPU besitze einen 64 KiB großen 4-fach Assoziativcache mit einer Blockgröße von 64 Byte. Beim wievielten Speicherzugriff findet frühestens eine Verdrängung aus dem Cache statt, wenn durch die Ersetzungsstrategie das am längsten nicht verwendete Datum einer Menge verdrängt wird (LRU) und der Cache zu Beginn leer ist?

   Wie nennt sich die Art des Cache Misses, die dabei auftritt? 2 Punkte

3. Gegeben sei folgende Adressaufteilung:

   Welche Organisationsform liegt bei den folgenden Cachegrößen (nur Cacheblöcke, keine Verwaltungsinformationen) vor?

   Begründen Sie Ihre Antwort rechnerisch! 4 Punkte

a) 2^{20} Byte

b) 4 MiB

4. 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. Es wird jeweils der Eintrag verdrängt, der am längsten nicht verwendet wurde (LRU).

   Ein Programm liest nacheinander *jeweils 4 Byte* von den folgenden Adressen:

   ```
   0x00 cd 10 50
   0x00 cd 10 58
   0x00 cd 10 60
   0x01 00 00 00
   0x00 cd 10 54
   0x01 00 00 10
   0x00 cd 10 78
   0x00 cd 10 74
   ```

   Der relevante Speicherbereich hat folgenden Inhalt:

   ```
   0x00 cd 10 50: ca 01 6b 7f ff 00 10 bc  bf 41 45 e8 00 a7 56 00
   0x00 cd 10 60: e8 00 a4 22 ff ff 50 bf  40 b0 c7 00 a3 05 21 21
   0x00 cd 10 70: 57 41 56 42 55 43 54 41  89 55 53 fd 89 48 48 f3
   ...
   0x01 00 00 00: a8 64 ff ff 89 48 e8 df  a9 1c ff ff 8b 48 ed 3d
   0x01 00 00 10: 32 4b e1 75 fb c1 11 09  7b 80 2e 01 94 0f ff 10
   ...
   ```

   Welchen Inhalt hat der Cache nach Ausführung des Programms?

   Es genügt, jeweils das erste und letzte Byte der Daten im Cache-Block einzutragen. 6 Punkte

Menge	Valid	Tag	Daten
0			
1			

Aufgabe 6: Speicherverwaltung — 11 Punkte

1. Nennen Sie einen Vorteil und einen Nachteil von Alignment! 2 Punkte

2. Wie viel Platz braucht folgende Datenstruktur auf einer 32-Bit-Architektur, wenn korrektes Alignment verwendet wird? Arrays werden nur gemäß ihres Typs ausgerichtet.

   Ändern Sie dabei nicht die Reihenfolge der Variablen! Verwenden Sie die folgende Tabelle. Tragen Sie auch die Füllbytes explizit ein. 4 Punkte

```
1 struct student {
2   char[11] name;          //char: 1 Byte
3   char[12] vorname;
4   short    versuch;       //short: 2 Byte
5   char     geschlecht;
6   int      matnr;         //int: 4 Byte
7   char     bestanden;
8   float    note;          //float: 4 Byte
9 };
```

Variable	erstes Byte	letztes Byte
name	0	10
vorname	11	

3. Wie viele obiger Studenten lassen sich in 80 KiB abspeichern? 1 Punkt

4. Was versteht man unter *Little-Endian* (LE) und *Big-Endian* (BE)? 2 Punkte

5. Ab der Adresse `0x12` stehen folgende Bytes sequentiell im Speicher: 0xFC, 0x13, 0xCB, 0x01.

   Welchem *hexadezimalen* 4-Byte Integer-Wert entspricht dies jeweils in LE- und BE-Darstellung?
   2 Punkte

Aufgabe 7: Assembler — 13 Punkte

Gegeben sei der folgende Linux-x86-ABI konforme AT&T-Assemblercode.

```
1  .data
2  LEN:
3    .long 12
4  i:
5    .long 0
6  .text
7  Lcond:
8    movl i, %eax
9    cmpl LEN, %eax
10   jl Lbody
11   jmp Lend
12 Lbody:
13   call func1
14   incl i
15   jmp Lcond
16 Lend:
17 ...
```

1. Um welche Befehlssatzarchitektur-Klasse handelt es sich und woran ist dies erkennbar?

2 Punkte

2. Konvertieren Sie den gegebenen Assemblercode für eine Register-Register-Architektur. Verwenden Sie das folgende Befehlsformat:

Opcode	Operand 1	Operand 2	Operand 3	Erklärung
`add,sub`	Rd	Rs	Rt/C	Rd = Rs + Rt/C
`loadr`	Rt	Rs		Rt = Memory[Rs]
`storer`	Rt	Rs		Memory[Rs] = Rt
`loadm`	Rt	MemAddr		Rt = Memory[MemAddr]
`storem`	Rt	MemAddr		Memory[MemAddr] = Rt
`loadi`	Rd	C		Rd = C
`blt`	Rs	Rt	Label	if (Rs<Rt) Befehlszähler=Label
`b`	Label			Befehlszähler=Label
`call`	Label			Label()

`Rd, Rs, Rt` entsprechen den (Caller-Save) Registern R0, R1, ...
`C` ist eine unmittelbar adressierte Konstante (Immediate Operand)
`MemAddr` ist eine ausreichend breite Speicheradresse, bzw. Variablenname

6 Punkte

Fortsetzung von Aufgabe 2.

3. Wie viele Sprünge müssen beim gegebenen Programm pro Schleifeniteration, abgesehen vom `call`, ausgeführt werden? 1 Punkt

4. Beschreiben Sie, wie die Reihenfolge der Instruktionen geändert werden müsste, um die Anzahl zu reduzieren! (Es ist nicht nötig, das geänderte Programm anzugeben.) 2 Punkte

5. Warum muss `i` immer wieder ins Register `eax` geladen werden?
   Wodurch könnte das vermieden werden? (Keine nebenläufigen Prozesse.) 2 Punkte

Zusätzlicher Platz