Prüfungsklausur Computersysteme (25211)

Teil II: Kurs 1609

WS 2016/17

Prof. Dr. W. Schiffmann

04.03.2017

Aufgabenübersicht:

Aufgabe II-1:	Fragen zu Computersystemen	(8 Punkte)
Aufgabe II-2:	Gleitkommazahlen-Darstellung	(8 Punkte)
Aufgabe II-3:	Code-Analyse	(10 Punkte)
Aufgabe II-4:	MMX-Rechenwerk	(12 Punkte)
Aufgabe II-5:	Pseudo-LRU-Verfahren	(12 Punkte)

Punkteverteilung:

	1	2	3	4	5	Summe
bearbeitet						
max. Punktzahl	8	8	10	12	12	50
erreichte Punktzahl						

Aufgabe 1: (Fragen zur Rechnerarchitektur) (8 Punkte)

Geben Sie für die folgenden Aussagen an, ob sie richtig oder falsch sind.

	richtig	falsch
a) Voraussetzung für den Einsatz von Cache-Speichern und der virtuellen Speicherverwaltung ist das **Lokalitätsprinzip typischer Programme**. (0,5 Punkte)	☐	☐
b) Bei der **LRU-Verdrängungsstrategie** der Cache-Verwaltung kann u.U. auch das zuletzt eingelagerte Datum als erstes wieder verdrängt werden. (0,5 Punkte)	☐	☐
Bei einem **superskalaren Prozessor** mit der Zuweisungsbandbreite *n* werden in jedem Takt immer genau n Befehle den Ausführungseinheiten zugewiesen. (0,5 Punkte)	☐	☐

d) Geben Sie das Hauptmerkmal für die Harvard-Architektur und den daraus folgenden wesentlichen Vorteil gegenüber der Von-Neumann-Architektur an. (2 Punkte)

Merkmal: .....

.....

Vorteil: .....

.....

e) Geben Sie für einen Cache-Speicher kurz die Bedeutung der beiden folgenden Begriffe an: (2,5 Punkte)

Cache-Kohärenz: .....

.....

.....

Cache-Konsistenz: .....

.....

.....

f) Geben Sie die beiden grundsätzlichen Verfahren zur virtuellen Speicherverwaltung an und benennen Sie die von ihnen erzeugten Adressen: (2 Punkte)

1. Verfahren: .....

Adresse: .....

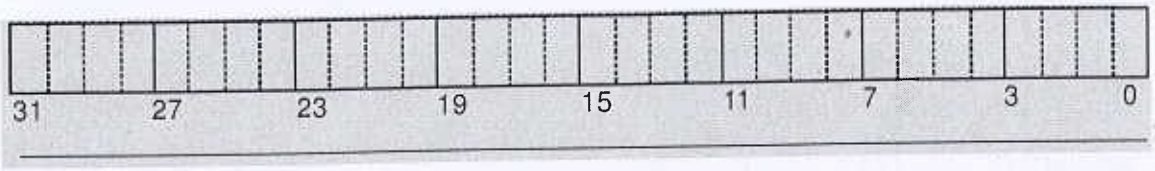
2. Verfahren: .....

Adresse: .....

Aufgabe II-2: (Gleitkommazahlen-Darstellung) (8 Punkte)

a) Die Zahl $Z1_{32}$ = \$4A5A0000 sei im 32-bit-Format nach dem IEEE-754-Standard gegeben.

i. Tragen Sie die Zahl in binärer Form in den folgenden Rahmen ein und kennzeichnen Sie darunter (auf der waagerechten Linie) die einzelnen Bitfelder und die Lage des Dezimalpunktes. (2 Punkte)



ii. Berechnen Sie den Dezimalwert $Z1_{10}$ der Zahl $Z1_{32}$. Benutzen Sie dazu die Bitfelder aus Teil i. (Den Exponenten können Sie als Dezimalzahl schreiben.) Rechnung erforderlich! (2 Punkte)

$Z1_{10} = (-1)^{\cdots} \cdot 2^{\cdots} \cdot (\ldots\ldots)_2 = \ldots\ldots$

b) Stellen Sie die Zahl $Z2_{10}$ = –0,224609375 im 32-bit-Format nach dem IEEE-754-Standard dar. Geben Sie die Zahl auch als Binär- und Hexadezimalzahl sowie Ihre Rechnung an.

$Z2_{32} = (-1)^{\cdots} \cdot 2^{\cdots} \cdot (\ldots\ldots)_2$ (2 Punkte)

$= (\ldots\ldots)_2 = \$\ldots\ldots$

Berechnung: (2 Punkte)

Benutzen Sie für Ihre Berechnungen die folgende Tabelle.

Tabelle der Potenzen von 2 für n=0,..,10 bzw. n=0,-1,...,-10

n	2^n	2^{-n}	n	2^n
0	1	1	11	2048
1	2	0,5	12	4096
2	4	0,25	13	8192
3	8	0,125	14	16384
4	16	0,0625	15	32768
5	32	0,03125	16	65536
6	64	0,015625	17	131072
7	128	0,0078125	18	262144
8	256	0,00390625	19	524288
9	512	0,001953125	20	1048576
10	1024	0,0009765625	21	2097152

Aufgabe II-3: Code-Analyse

(10 Punkte)

Gegeben sei der unten stehende DLX-Assembler-Code.

```
Zeile Marke    Anweisung

1             LB     R4, op1
2             LB     R10, lp
3             LW     R11, mask1
4             LW     R12, mask2
5             LW     R13, mask3
6             AND    R4, R4, R11
7      l1:    JAL    m2
8             SLLI   R4, R4, #0x1
9             SUBI   R10, R10, #0x1
10            BNEZ   R10, l1
11            SRLI   R4, R4, #0x8
12            SW     dest, R4
13     m1:    J      m1
14     m2:    AND    R5, R4, R12
15            SGTI   R5, R5,  #0x400
16            BEQZ   R5, m3
17            ADDI   R4, R4, #0x300
18     m3:    AND    R5, R4, R13
19            SGTI   R5, R5, #0x4000
20            BEQZ   R5, m4
21            ADDI   R4, R4, #0x3000
22     m4:    JR     R31

23     op1:   byte   0xe4
24     lp:    byte   0x8
25     dest:  word   0xdeadbeef
26     mask1: word   0xff
27     mask2: word   0xf00
28     mask3: word   0xf000
```

Hinweise:

- Immediate-Werte werden mit einem Doppelkreuz ‚#' gekennzeichnet.
- Hexadezimalzahlen werden mit einem einleitenden ‚0x' gekennzeichnet.
- Bei *Store*-Befehlen gibt der erste Operand das Ziel im Speicher, der zweite Operand die Quelle (ein Register) an.
- Bei *Load*-Befehlen gibt der erste Operand das Ziel (ein Register), der zweite Operand die Quelle im Speicher an.

a) Erklären Sie in einem Satz folgende Assembler-Anweisungen (vgl. das Beispiel am Ende dieses Aufgabenteils!).

Zeile 8: SLLI R4, R4, #0x1 (1 Punkt)

Zeile 14: AND R5, R4, R12 (1 Punkt)

Zeile 20: BEQZ R5, m4 (1 Punkt)

Beispiel:
Zeile 3: *LW R11, mask1* (*Load Word*): Diese Anweisung lädt das 32-Bit-Wort an der Speicherstelle mask1 in das Register R11.

b) Erläutern Sie, was der Befehl in Zeile 7: *JAL m2* bewirkt und wie oft dieser im Programmverlauf aufgerufen wird. Wie hängt der Befehl in Zeile 22: *JR R31* mit diesem zusammen? (2 Punkte)

c) Erklären Sie, was der Codeblock von Zeile 14 bis 22 funktional realisiert. (3 Punkte)

d) Geben Sie den Wert von R4 an, der in Zeile 12 in den Speicher geschrieben wird. Falls der Befehl mehrfach aufgerufen wird, geben Sie den zuerst geschriebenen Wert an. (1 P)

e) Beschreiben Sie in einem Satz, was das obige Programm macht. (Hinweis: Zahlendarstellungen.) (1 Punkt)

Aufgabe II-4: (MMX-Rechenwerk) (12 Punkte)

Ein MMX-Rechenwerk *(Multimedia Extension)* unterstützt gepackte Datenformate, die in einem 64-bit-Register wahlweise 8 Bytes, 4 (16-bit-)Wörter, 2 (32-bit-)Doppelwörter oder ein 64-bit-Wort (Quadword) unterbringen. Alle Werte können vorzeichenlos oder vorzeichenbehaftet sein. Negative Werte werden dabei im Zweierkomplement dargestellt. Spezielle MMX-Befehle wirken parallel auf diese Datenformate, d.h. es können z.B. durch einen einzigen Addier-Befehl zweimal 8 Bytes addiert werden. Ein Übertrag zwischen den einzelnen Werten der gepackten Daten findet dabei nicht statt. Die Datenbreite wird im Assemblerbefehl spezifiziert. Der erwähnte Addier-Befehl hat z.B. die Form: PADDX (*Parallel Add*), wobei X = B, W, D für 8 Bytes, 4 Wörter bzw. 2 Doppelwörter steht.

a) Die Addition mit vorzeichenbehafteten Werten kann wahlweise in einer der beiden folgenden Formen geschehen:

- **mit Abschneiden** (Befehl PADD, *Wrap Around*):
  Ein (Teil-)Ergebnis, das nicht ins vorgegebene Datenformat passt, verursacht einen Über- oder Unterlauf im darstellbaren Zahlenbereich und wird auf die durch die Datenlänge definierte Anzahl von (niederwertigen) Bits abgeschnitten (modulo-2^n-Rechnung, n = 8, 16, 32, 64);.
- **mit Sättigung** (Befehl PADDS, PADDUS, *Saturation*):
  Das Ergebnis wird bei einem Überlauf auf den größten darstellbaren Wert aufgerundet bzw. bei einem Unterlauf auf den kleinsten Wert abgerundet. Dabei können die Zahlen als vorzeichenlos (U) oder vorzeichenbehaftet angesehen werden.

Welchen Wert hat das Ergebnisregister R1 nach Ausführung der folgenden Befehle mit der Eingabe? Geben Sie die Rechnungen an.

R1 = $835A7F346AE303D2, R2 = $12F0A234AF061707

(Diese Eingabe soll für jeden Befehl erneut verwendet werden.)

1. PADDB R1, R2 (R1 := R1 + R2 mit *Wrap Around*) (3 Punkte)

2. PADDUSW R1, R2 (R1 := R1 + R2 vorzeichenlos, mit Sättigung) (3 Punkte)

3. PADDSD R1, R2 (R1 := R1 + R2 vorzeichenbehaftet, mit Sättigung) (2 Punkte)

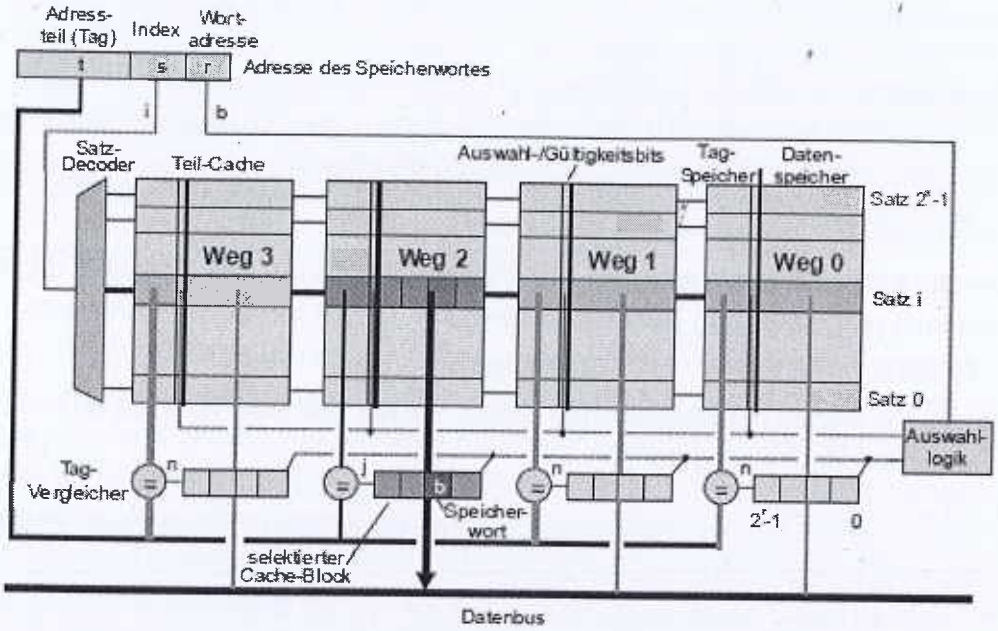
b) Geben Sie für den Eingabewert R1 = $87565432F23400F5 das Ergebnis des folgenden Befehls zum arithmetischen Rechtsschieben an:

PSRAW R1, #5 (Ergebnis in R1),

in dem #5 die Anzahl der Bitpositionen angibt, um die (parallel) verschoben werden soll. Geben Sie den Rechenweg an. (4 Punkte)

Aufgabe II-5: (Pseudo-LRU-Verfahren) (12 Punkte)

In Kapitel 3 des Kurses und in der Einsendeaufgabe EA3 wurde der *4-fach satzassoziative Cache* (auch als *4-way set-associative Cache* bezeichnet) beschrieben. Das folgende Bild zeigt – als Wiederholung – den Aufbau des Caches. Da der Speicher byte-adressierbar sei, stimmt die Wortadresse mit der Byte-Adresse überein.



In dieser Aufgabe sollen Sie das **Pseudo-LRU-Verfahren** genauer untersuchen, das im Kapitel 3 kurz angesprochen wurde. Bei diesem Verfahren werden jedem Cache-Block drei Bits (B2, B1, B0) zugewiesen, die zur Realisierung des Verfahrens herangezogen werden.

In der folgenden Tabelle wird links dargestellt, wie die Bits B2, B1, B0 bei jedem Schreib- oder Lesezugriff auf einen Eintrag k in Abhängigkeit vom adressierten Weg (*Way*) modifiziert werden. Der Wert ‚u' zeigt dabei an, dass das entsprechende Bit ‚unverändert' bleibt. Rechts in der Tabelle ist gezeigt, wie der Controller aus dem aktuellen Zustand der Bits B2, B1, B0 nach einem fehlgeschlagenen Zugriffsversuch *(Read/Write Miss)* den Weg bestimmt, in dessen Eintrag k die neuen Daten eingelagert werden. Ggf. müssen vorher die alten Daten aus diesem Eintrag k verdrängt werden, um dem neuen Eintrag aus dem Hauptspeicher Platz zu machen. Der Wert ‚d' steht darin für ‚irrelevant' *(don't care)*. Nach der Einlagerung werden die Bits B2, B1, B0 ebenfalls, wie in der linken Tabelle angegeben, geändert. Bei einem *Read Hit* werden die Bits B2, B1, B0 dem Weg entsprechend, in dem das Datum gefunden wurde, nach dem linken Teil der Tabelle gesetzt.

Tabelle: Die Bedeutung der Bits B2 – B0

	Modifikation			Einlagerung/ Verdrängung			
Weg	B2	B1	B0	B2	B1	B0	Weg
3	1	1	u	0	0	d	3
2	1	0	u	0	1	d	2
1	0	u	1	1	d	0	1
0	0	u	0	1	d	1	0

a) In einem Programm werde auf eine Folge von Variablen A – F unter Hauptspeicheradressen zugegriffen, die alle vom Cache-Controller in denselben Cache-Einträgen k abgelegt

werden müssen. Vervollständigen Sie die folgende Tabelle der aktuellen Inhalte der Cache-Einträge k und der momentanen Zustände der Bits B2, B1, B0. 0(u) bzw. 1(u) sollen darin anzeigen, dass das entsprechende Bit unverändert im ‚0'-Zustand bzw. ‚1'-Zustand bleibt. (6 Punkte)

Zugriff	Variable	*Hit/Miss*	Weg 3	Weg 2	Weg 1	Weg 0	B2	B1	B0
0	–		(frei)	(frei)	(frei)	(frei)	0	0	0
1	A	*Miss*	**A**	(frei)	(frei)	(frei)	1	1	0(u)
2	B	*Miss*	A	(frei)	**B**	(frei)	0	1(u)	1
3	A	*Hit*	A	(frei)	B	(frei)			
4	C	*Miss*							
5	D								
6	A								
7	E								
8	C								
9	F								
10	B								
11	C								

b) Für den 4-fach satzassoziativen Cache nach Teilaufgabe a) seien die 32-bit-Adressen in die drei Bitfelder *Tag*, Index, Byteadresse eingeteilt: (3 Punkte)

Tag: Bits 31 – 16 Index: Bits 15 – 5 Byteadresse: Bits 4 – 0

Berechnen Sie daraus die folgenden Größen:

- Anzahl der Bytes pro Cache-Block *(Line Length)*: .....
- Anzahl der Einträge pro Weg: .....
- Kapazität der einzelnen Cache-Wege in kB: .....
- Gesamtkapazität des Caches in kB: .....

c) Welche der Speicherzellen mit den folgenden Adressen A1, A2, A3 werden im selben Eintrag k, wenn u.U. auch in verschiedenen Wegen, des Caches abgelegt? (3 Punkte)

A1 = $CBC39AB1, A2 = $46359AA5, A3 = $D965D236?

Welche Nummer trägt dieser Eintrag? (Kurze Begründung)

Tragen Sie *Tag*, Index und Byteadresse in hexadezimaler Darstellung in die folgende Tabelle ein.

Speicheradresse	*Tag*	Index	Byteadresse
$CBC39AB1			
$46359AA5			
$D965D236			

Nummer des Eintrags: .....