| Technische Grundlagen der Informatik | | | Test 3<br>22.06.2018<br>90 Minuten<br>Gruppe A |
|---|---|---|---|
| Matrikelnr. | Nachname | Vorname | Unterschrift |
| | | | |

- Deckblatt sofort ausfüllen.
- Undeutliche oder unleserliche Antworten führen zu Punkteabzug.
- Streichen Sie Passagen, die nicht gewertet werden sollen, durch.
- Geben Sie immer den Lösungsweg an.
- Zusatzblätter werden nicht akzeptiert.

| | | |
|---|---|---|
| 1 | [8] | [ ] |
| 2 | [14] | [ ] |
| 3 | [8] | [ ] |
| 4 | [8] | [ ] |
| 5 | [10] | [ ] |
| 6 | [11] | [ ] |
| 7 | [11] | [ ] |
| 8 | [10] | [ ] |
| 9 | [10] | [ ] |
| 10 | [10] | [ ] |
| Summe | [100] | [ ] |

1. *( ____ / 8 Punkte)* Netzwerke – Datenkapselung

Sie kopieren mit dem Tool "scp" eine Datei von 8100 Byte (Nutzdaten). Sie verwenden dabei folgende Protokolle mit den in Klammern angegebenen Header-Größen:

*IPv6 (28 Bytes) – TCP (16 Bytes) – Ethernet (17 Bytes) – SFTP (19 Bytes)*

Die Transportschicht beschränkt die Größe eines übertragenen Pakets (Nutzdaten + Header) auf maximal 1024 Byte. Alle darunterliegenden Schichten unterliegen keiner weiteren Beschränkung.

(a) Ordnen Sie die angegebenen Protokolle den Layern im TCP/IP-Referenzmodell zu. In Klammer sind die jeweils entsprechenden Schichten des OSI-Modells angegeben.

| Layer | Protokoll |
|---|---|
| Application (5-7) | SFTP |
| Transport (4) | TCP |
| Internet (3) | IPv6 |
| Network (1-2) | Ethernet |

(b) In wie viele Einzelpakete muss die Transportschicht das ursprüngliche SFTP-Paket aufteilen?

$\frac{8100 + 19}{1024 - 16} = \frac{8119}{1008}$

8119 : 1008 = 8, ... ⇒ 9 Pakete
8016

2. *( ____ / 14 Punkte)* Theorie

(a) Wie heißt die Computer-Architektur, in der Programmspeicher und Datenspeicher voneinander getrennt sind?

Harvard-Architektur

(b) Wie nennt man die Lokalität ("Locality"), bei der gilt: Wird auf einen Speicherinhalt gerade zugegriffen, so ist es relativ wahrscheinlich, dass der nächste Zugriff in dessen Nachbarschaft erfolgen wird.

(Temporal locality: ..., dass der nächste Zugriff bald erfolgen wird)

Spatial Locality

(c) Welche der Strategien Write-Through, Move-Back, Copy-Back, Write-Around und Fetch-on-Write kann verwendet werden, wenn während eines Schreibzugriffes ein Miss auftritt?

Fetch on Write, Write Around

(d) Wie nennt man Hazards, bei denen Nachfolgebefehl vom Ausgang des Sprunges abhängt? Welche Maßnahmen sind dagegen effektiv?

Control Hazards, stall, prediction

(e) Welche der Abkürzungen *LIFO, LILO, FIFO, SUDO* und *FILO* beschreibt die Funktionsweise eines Stacks und wofür steht die Abkürzung?

LIFO: Last in first out

(f) Gegeben sind nachfolgende Instruktionen. Schreiben Sie jeweils rechts neben der Instruktion die verwendete Adressierungsart hin.

*Hinweis:* In der Vorlesung sind folgende Adressierungsarten betrachtet worden: Direct-Addressing Mode, Immediate Mode, Indirect-Addressing Mode, Register-Indirect Mode und Register Mode.

| | |
|---|---|
| $R1 \leftarrow R2$ | Register Mode |
| $R7 \leftarrow memory[memory[0x666]]$ | Indirekt Adressing Mode |
| $R0 \leftarrow memory[0x500]$ | Direkt-Adressing Mode |
| $R3 \leftarrow -1$ | Immediate Mode |
| $R3 \leftarrow memory[R4]$ | Register Indirekt Mode |

4. *( ____ / 8 Punkte)* Das nachfolgende Micro16-Programm soll überprüfen, ob das Datenwort, das im Register R0 abgelegt ist, gerade oder ungerade Parität hat. Ist die Anzahl der Einsen im Register R0 gerade, dann wird der Wert 0 im Register R1 abgelegt. Ansonsten wird der Wert +1 im Register R1 abgelegt.

Vervollständigen Sie das Programm in den Zeilen 00, 03, 08 und 11.

*Micro-Code:*

```
00:     R1 ← 0
01:  :loop
02:     (R0 ) if Z goto .check
03:     (R0 & 1) if ____Z____ goto .noadd
04:     R1 ← R1 +1
05:     R0 ← rsh(R0 )
06:     goto .loop
07:  :noadd
08:     R0 ← rsh(R0)
09:     goto .loop
10:  :check
11:     (R1 & 1) if Z goto even
12:  :odd
13:     R1 ← 1
14:     goto .end
15:  :even
16:     R1 ← 0
17:  :end
```

5. *( ____ / 10 Punkte)* Schreiben Sie ein Micro16-Programm, das zwei Datenworte dividi
Der Dividend liegt im Speicher an der Adresse 0xFFFE und der Divisor liegt im Register
Subtrahieren Sie dafür den Divisor so oft vom Dividenden, bis dieser negativ wird und erhöl
Sie dabei jedes mal das Ergebnisregister um 1. Am Ende des Programms soll das Ergeb
der Division im Register R1 stehen und der verbleibende Rest soll im Register R7 stehen.
können davon ausgehen, dass der Dividend und der Divisor beide positiv sind.

Micro-Code:

```
R2 ← (sh(-1)
MAR ← R2 ; rd
rd
R2 ← MBR
R0 ← ~R0
R0 ← R0+1
:loop
R2 ← R2+R0
(R2) ; ifN goto.end
R1 ← R1+1
R7 ← R2
goto.loop
:end
```

7. *( ____ / 11 Punkte)* Folgender eingerückter Text ist analog zu der Stack-Aufgabe in Übung 7:

> Gegeben ist folgendes Programm in Pseudocode-Notation. Die Ausführung startet bei `Program Start()` und endet bei `exit`. Die Variable `P` ist global und die Variable `f` ist lokal deklariert. Die Funktion `fact_rec` hat einen Übergabeparameter. Für die Verwaltung von Rücksprungadressen, lokalen Variablen und Übergabeparameter wird ein Stack verwendet. Lokale Variablen werden bei der ersten Verwendung auf den Stack gelegt und beim Verlassen der Funktion wieder vom Stack entfernt. Zuerst werden Rücksprungadressen und erst dann Übergabeparameter auf den Stack gelegt. Übergabeparameter entsprechen lokalen Variablen in der aufgerufenen Funktion.
>
> *Hinweis:* Bei Veränderung von lokalen Variablen oder Übergabeparameter werden keine Stack-Operationen durchgeführt. Der Inhalt vom RAM, in dem der Stack liegt, ändert sich allerdings.
>
> *Hinweis:* Möglicherweise werden nicht alle Zeilen der Tabelle benötigt.

Tragen Sie in die Tabelle ein:

- welche Codezeile (**C**, kann `1:` bis `9:` sein) ausgeführt wurde,
- welchen Wert die globale Variable `P` nach Exekution dieser Codezeile hat,
- welche Stack-Operationen (push(<Wert>)/pop()) ausgeführt werden und
- wie der Stack-Inhalt nach der Codezeile aussieht. Der Stack soll nach **rechts** wachsen.

```
     global P;
     function fact_rec(f) {
1:     if f > 1 then {
2:       P = P*f;
3:       f = f-1;
4:       fact_rec(f);
       }
5:     return;
     }

     Program Start() {
6:     P = 1;
7:     f = 3;
8:     fact_rec(f);
9:     exit;
     }
```

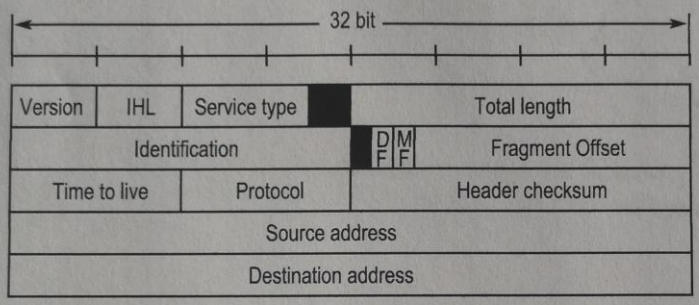
| C | P | Stack-Operationen | Stack-Inhalt (→) |
|---|---|---|---|
| 6 | 1 | | |
| 7 | 1 | push(3) | 3 |
| 8 | 1 | push(9) push(3) | 3, 9, 3 |
| 1 | P | | 3, 9, 3 |
| 2 | 3 | | 3, 9, 3 |
| 3 | 3 | | 3, 9, 3 |
| 4 | 3 | push(5) push(2) | 3, 9, 3, 5, 2 |
| 1 | 3 | | — " — |
| 2 | 6 | | — " — |
| 3 | 6 | | — " — |
| 4 | 6 | push(5) push(1) | 3, 9, 3, 5, 2, 5, 1 |
| 1 | 6 | | — " — |
| 5 | 6 | pop(1) pop(5) | 3, 9, 3, 5, 2, |
| 5 | 6 | pop(2) pop(5) | 3, 9, 3 |
| 5 | 6 | pop(9) pop(3) | 3 |
| 9 | 6 | pop(3) | |
| | | | |

8. *( ____ / 10 Punkte)* Die nachfolgend angeführte Bitfolge in hexadezimaler Notation stellt ein IP-Paket dar, das über das Ethernet Protokoll transportiert wird:

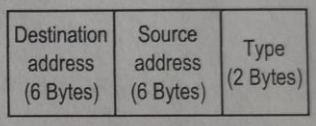
```
0057A65C7698 0A0EC667670C 0800 4 5 00 003F 35F0 0000 7F 01 8DF2 80820C6A 80821F01 4E
                                                                4749...
```

Start der Folge ist links. Den genauen Aufbau der Protokoll-Header können Sie den Grafiken entnehmen.

IP-Header:




Ethernet-Header:




(a) Markieren Sie in obiger Bitfolge den Beginn des IP-Headers klar ersichtlich.

(b) Geben Sie die *Time to live* des IP-Pakets in <u>hexadezimaler</u> Notation an.

$(7F)_{16}$

(c) Geben Sie die beiden IP-Adressen in <u>Dezimal</u>notation an.

0x80820C6A = 1000 0000 | 1000 0010 | 0000 1100 | 0110 1010 = 128.130.12.

0x80821F01 = 1000 0000 | 1000 0010 | 0001 1111 | 0000 0001 = 128.130.31.1

(d) Wie lautet das erste Byte des mittels IP transportierten *Payloads* in <u>binärer</u> Notation?

$0x4E = (0100\ 1110)_2$

10. *( ____ / 10 Punkte)* Ein Programm $P$ wird beschleunigen, indem es teilweise parallel auf einem System mit sechs Cores ausgeführt wird. Dabei wurde ein Speed-Up von $S = 3,\dot{3}$ $(= \frac{10}{3})$ erreicht. Bei der Analyse der Programmlaufzeit auf dem Multicore System ist aufgefallen, dass

- 60% des Programmes ($p_1 = 0,6$) beliebig parallelisiert werden können und
- während 10% des Programms ($p_2 = 0,1$) nur ein Core aktiv ist.

(a) Wie viele Cores $n$ sind in den restlichen 30% ($p_3 = 0,3$) des Programms in Verwendung?

$S = \frac{1}{1-p+p/c}$

$\frac{1}{0,1+\frac{0,6}{6}+\frac{0,3}{n}} = \frac{10}{3} \Rightarrow \frac{1}{\frac{2}{10}+\frac{3}{10n}} = \frac{10}{3}$

$\frac{2}{10}+\frac{3}{10n} = \frac{3}{10} \Rightarrow 2+\frac{3}{n} = 3 \quad n = 3$

$\frac{3}{n} = 1$

(b) Wie hoch liegt der theoretisch mögliche Speed-Up unter der Annahme, dass der Programmanteil $p_3$ 3 Cores verwendet und eine unbegrenzte Anzahl an Cores zur Verfügung stehen?

$S = \frac{1}{\frac{1}{10}+\frac{0,6}{10n}+\frac{1}{10}} = \frac{1}{\frac{2}{10}+\frac{0,6}{10n}}$ $\quad n \to \infty \Rightarrow \frac{1}{\frac{2}{10}} = S$

(c) Wie hoch ist die Ausnutzung der 6 Cores in Prozent, wenn während der Ausführung von

- $p_1$(60% des Programms) alle 6 Cores,
- $p_2$(10% des Programms) nur 1 Core und
- $p_3$(30% des Programms) 3 Cores

verwendet werden?

$A_z: \frac{tp}{tps}$ $\quad 1-0,9+0,6 \cdot 0,3 = 1$

$\frac{\frac{1}{6}}{\frac{2}{10}+\frac{1}{10}} = \frac{1}{6} \cdot \frac{10}{3} = \frac{10}{18} = 55,5\%$

(d) Wie kann eine Ausnutzung von 100% erreicht werden?

1 Kern nutzen

9. *( ____ / 10 Punkte)* Ein Prozessor besitzt eine Hierarchie aus zwei Caches. L1 ist 32KiB groß und als *8-Way Set Associative Cache* organisiert, L2 als *16-Way Set Associative Cache*. Die Adresslänge beträgt 64 Bit, die kleinste adressierbare Einheit ist 1 Byte. Beide Caches verwenden eine Blockgröße von 64 Byte.

L1 hat eine Hit-Rate von 80 Prozent und L2 von 90 Prozent.

(a) Berechnen Sie für den L1-Cache die Länge des Tags in Bits.

(b) Der Tag des L2 Cache ist um 2 Bit kürzer als der des L1 Cache. Wie groß ist der L2 Cache?

(c) Betrachten Sie L1 und L2 als ein System und geben Sie die Hit-Rate des gesamten Cache in Prozent an.

(d) Der Prozessor arbeitet mit einer Taktrate von 2 GHz und der Cache hat die folgenden Zugriffszeiten (vorangehende Misses inkludiert):

- L1-Cache: 2 Takte
- L2-Cache: 4 Takte
- Hauptspeicher: 20 Takte

Es wird ein Programm mit 100.000 Adressanfragen ausgeführt. Wie lange dauern diese 100.000 Adressanfragen durchschnittlich. Geben Sie das Ergebnis in Mikrosekunden ($\mu$s) an.

___ / *11 Punkte)* Folgender eingerückter Text ist analog zu der Pipeline-Aufgabe in Übung 7:

Sie haben ein Programm mit bedingten Sprüngen in Pseudo-Code Notation gegeben. Der Prozessor verwendet Pipelining mit *Branch Prediction*. Die Pipeline besteht aus den vier Stufen *Instruction Fetch (F)*, *Decode (D)*, *Execute (E)* und *Memory & Write Back (M/W)*. Werte, die in *M/W* geschrieben werden, stehen im selben Takt in *D* bereits zur Verfügung.

Bedingte Sprünge werden in Stufe *D* erkannt. Ob gesprungen wird oder nicht, steht fest, wenn die Instruktion die Stufe *E* verlässt.

Wandert also eine Instruktion mit bedingtem Sprung in die Stufe *D*, so wird zunächst die nächste Instruktion in *F* geladen. Danach wird, abhängig von der Sprungvorhersage, wieder die nächste Instruktion in die Pipeline geladen (kein Sprung), oder die Instruktion in *F* wird aus der Pipeline geworfen und es wird die Instruktion an der Sprungadresse in *F* geladen (Sprung).

Verlässt die Instruktion mit bedingtem Sprung die Stufe *E* und war die Sprungvorhersage falsch, muss die Pipeline *geflushed* werden. Das bedeutet, die Stufen *F* und *D* werden geleert und *F* wird mit der nächsten korrekten Instruktion geladen.

Die Strategie zur Sprungvorhersage nimmt beim ersten Mal an, dass *nicht* gesprungen wird. Bei jedem weiteren bedingten Sprung wird angenommen, dass die Sprungentscheidung so wie beim letzten Mal ausfällt.

Zeichnen Sie den Ablauf der Pipelineverarbeitung für das gegebene Programm. Setzen Sie die Darstellung solange fort, bis alle Instruktionen vollständig abgearbeitet wurden.

Berücksichtigen Sie auch ***Read-After-Write* Data Hazards** und vermeiden Sie diese mittels *Stalling*. Gestallte Instruktionen müssen eingeklammert werden.

Programm *P*:

*L1:* P $\leftarrow$ *1*
*L2:* f $\leftarrow$ *3*
*L3:* P $\leftarrow$ *P* $*$ *f*
*L4:* f $\leftarrow$ *f* $-$ 1
*L5:* *if* (*f* $>$ 1) *goto L3*
*L6:* *NOP*
*L7:* *NOP*

| Zeit ↓ | F | D | E | M/W |
|---|---|---|---|---|
| 1 | L1 | | | |
| 2 | L2 | L1 | | |
| 3 | L3 | L2 | L1 | |
| 4 | L4 | (L3) | L2 | L1 |
| 5 | L4 | L3 | | L2 |
| 6 | L5 | L4 | L3 | |
| 7 | L6 | (L5) | L4 | L3 |
| 8 | L6 | L5 | | L4 |
| 9 | L7 | L6 | L5 | ← f=2 |
| 10 | L3 | | | L5 |
| 11 | L4 | L3 | | |
| 12 | L5 | L4 | L3 | |
| 13 | L6 | (L5) | L4 | L3 |
| 14 | L6 | L5 | | L4 |
| 15 | L3 | | L5 | ← f=1 |
| 16 | L6 | | | L5 |
| 17 | L7 | L6 | | |
| 18 | | L7 | L6 | |
| 19 | | | L7 | L6 |
| 20 | | | | L7 |
| 21 | | | | |