Beispielhafte Prüfungsaufgaben zur Vorlesung
Technische Informatik I
Gestellt im Frühjahr 2012

Die beigefügte Lösung ist ein Vorschlag. Für Korrektheit, Vollständigkeit und Verständlichkeit wird keine Verantwortung übernommen.

Aufgabe 4 : I/O-System ("Internet Radio Station") (maximal 32 Punkte)

4.1: Grundlagen (maximal 6 Punkte)

Beantworten Sie folgende Fragen mit jeweils maximal zwei Sätzen.

(a) (2 Punkte) Erklären Sie den Unterschied zwischen Polling und IO-Unterbrechung (Interrupts) in Bussystemen.

Lösungsvorschlag: Bei Polling werden Informationen über eine I/O-Einheit periodisch in einem Statusregister durch das OS abgefragt. Bei I/O Unterbrechung unterbricht die I/O-Einheit den Prozessor im laufenden Programm, falls sie seine Aufmerksamkeit benötigt.

□

(b) (2 Punkte) Was ist der Vorteil von Polling im Vergleich zur IO-Unterbrechung?

Lösungsvorschlag: Durch Polling kann unter umständen schneller auf ein Ereignis reagiert werden, da der Zeitaufwand für den Wechsel in die Interruptabhandlung wegfällt. Polling benötigt keine zusätzliche Hardwareunterstützung.

□

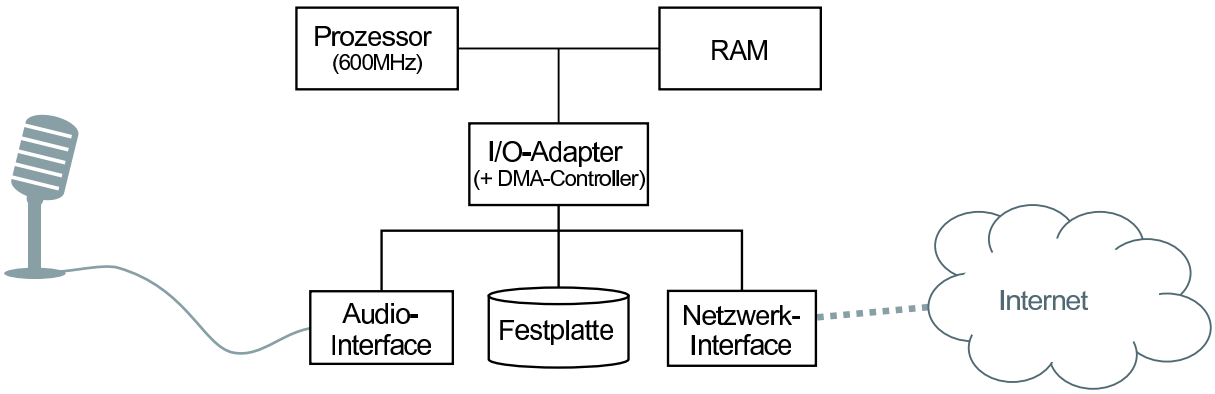
(c) (2 Punkte) Was versteht man unter speicheradressierter Ein/Ausgabe (memory mapped I/O)?

Lösungsvorschlag: Teile des Adressraumes sind den I/O-Einheiten zugewiesen. Schreiben und Lesen zu diesen Adressen bzw. Registern wird als Befehl an die I/O Einheit interpretiert.

□

4.2: Speicherplatz, Latenz, Durchsatz (maximal 26 Punkte)

Gegeben sei folgender Aufbau einer Internet Radio Station:



Ein Audiosignal wird mit einem Mikrophon aufgenommen, im Audio-Interface in ein digitales Signal umgewandelt und in einem Zwischenspeicher gespeichert. Diese Daten werden komprimiert und danach auf der lokalen Festplatte gespeichert (zu Archivzwecken). Anschliessend werden die Daten live über ein Netzwerk-Interface an Hörer im Internet weitergeleitet.

Der gesamte interne Zwischenspeicher des Audio-Interfaces wird jeweils mit einem DMA Transfer ausgelesen und in den Arbeitsspeicher geschrieben. Der Zwischenspeicher des Audio-Interfaces muss also jeweils ausgelesen werden sobald er voll ist, damit eine ununterbrochene Aufnahme gewährleistet werden kann.

Die Schritte 1 bis 6 aus Tabelle 1 werden sequenziell abgearbeitet, während sich der Audio-Zwischenspeicher parallel dazu füllt.

Schritt	Operation
1	Kopieren des Audio-Zwischenspeichers $\rightarrow$ RAM
2	Kompression
3	Kopieren der komprimierten Audio Samples von RAM $\rightarrow$ Festplatte
4	Hinzufügen von Netzwerk Metainformationen
5	Kopieren des Datenpakets (Metainformationen + komprimierte Audio Samples) von RAM $\rightarrow$ Netzwerk-Interface
6	Versenden des Datenpakets
parallel dazu	Aufnahme des Audiosignals $\rightarrow$ Audio-Zwischenspeicher

Tabelle 1: Der Gesamtablauf des Systems

Das System ist folgendermassen spezifiziert:

- Prozessor (CPU):
  - Taktfrequenz 600Mhz
- Arbeitsspeicher (RAM)
  - Benötigte CPU-Taktzyklen für einen Lesezugriff (CPU/Speicher-Bus): 4
  - Benötigte CPU-Taktzyklen für einen Schreibzugriff (CPU/Speicher-Bus): 6
  - Wortbreite (Daten): 16bit
- I/O-Adapter (Übergang vom CPU/Speicher-Bus zum I/O-Bus):
  - Benötigte CPU-Taktzyklen für einen Lesezugriff (I/O-Bus): 6
  - Benötigte CPU-Taktzyklen für einen Schreibzugriff (I/O-Bus): 10
  - Wortbreite (Daten): 16bit
- DMA-Controller (integriert in I/O-Adapter):
  - Aufsetzen eines kompletten DMA-Transfers benötigt 1150 CPU-Taktzyklen.
- Netzwerk Metainformation:
  - Hinzufügen von Metainformationen (64 Bytes) zu einem Datenpaket benötigt 816 CPU-Taktzyklen (inklusive Lesen der Daten aus dem Arbeitsspeicher und Zurückschreiben).
- Netzwerk:
  - Bandbreite des Netzwerk-Interfaces: 10Mbit/s (1Mbit/s = 1000kbit/s).
- Audio-Interface:
  - Digitalisiert das Audiosignal in stereo Samples mit einer Abtastrate von 44100Hz.
  - 1 stereo Sample $= 2 \cdot 16$ bit.
  - Interner Zwischenspeicher: 2048 Bytes.
- Kompression:
  - Anzahl Instruktionen für die Kompression von 512 stereo Samples des Audiosignals (inklusive Lesen der Samples aus dem Arbeitsspeicher und Schreiben der komprimierten Daten in den Arbeitsspeicher): 20000.
  - Audiokompression konstant auf $\frac{1}{8}$ der ursprünglichen Grösse.

(a) (6 Punkte) Wieviel Speicherplatz wird auf der Festplatte benötigt um die Radioaufzeichnung eines Tages zu archivieren?

Lösungsvorschlag: $2 \cdot 16 \cdot 44100 \cdot \frac{1}{8} \cdot 60 \cdot 60 \cdot 24 Bits = 1816.86 MBytes$

□

(b) (10 Punkte) Wie lange dauert es um den Zwischenspeicher des Audio-Interfaces zu füllen? Mit welcher Verzögerung gelangt eine Audio-Aufnahme vom Zwischenspeicher zur Festplatte?

Lösungsvorschlag:

Benötigte Schritte für lokale Speicherung:

Operation	Taktzyklen	Zeit
Audio → RAM (DMA, 512 Samples)	$1150 + (2048/2) \cdot (6 + 6)$	
Kompression	20000	
RAM → Festplatte	$1150 + (2048 \cdot \frac{1}{8}/2) \cdot (4 + 10)$	
Total	36380	$T_{local} = 60.63 \mu s$

Total Latenz: $= T_{local} = 60.63 \mu s$

□

(c) (10 Punkte) Für jeden Hörer wird ein Datenpaket mit gleichem Inhalt (jeweils die Aufnahme, die in den Audio-Zwischenspeicher gepasst hat) aber anderen Metainformationen über das Netzwerk-Interface verschickt. Für jeden Hörer müssen also Schritte 4 bis 6 aus Tabelle 1 wiederholt ausgeführt werden. Auch das geschieht sequentiell. Erst wenn das Datenpaket an einen Hörer vollständig verschickt wurde, kann das nächste vorbereitet werden.

Wieviele Hörer kann dieses System maximal kontinuierlich beliefern?

Lösungsvorschlag: Zeit um den Audio-Zwischenspeicher zu füllen:

$T_{audio} = 512 Samples / 44100 \frac{Samples}{s} = 11.61 ms.$

Der sequenzielle Ablauf darf maximal T_{audio} dauern, da sonst der Zwischenspeicher überläuft.

Netzwerk Part:

Operation	Taktzyklen	Zeit
Metainformationen hinzufügen	816	
RAM → NI	$1150 + (64 + 256)/2 \cdot (4 + 10)$	
Total	4206	$T_{net} = 7.01 \mu s$

Versendezeit: $T_{transmit} = (64 + 256) \cdot 8/10e6 = 256\mu s$

Benötigte Zeit ohne Netzwerk: T_{local} aus (b)

Verbleibende Zeit: $T_{rest} = T_{audio} - T_{local}$

$T_{rest}/(T_{net} + T_{transmit}) = 43.91$, also können maximal **43** Hörer beliefert werden.

□

Aufgabe 5 : Assemblerprogrammierung (maximal 25 Punkte)

Ein *binärer Suchalgorithmus* bestimmt die Position eines gegebenen Elementes in einem sortierten Array. Betrachten Sie als Beispiel das folgende 24-elementige sortierte Array a:

```
a = [2,3,8,10,16,21,35,42,43,50,64,69,70,77,82,83,84,90,96,99,100,105,111,120]
```

Es wird angenommen, dass das erste Element im Array an der Position 0 steht. Demzufolge würde bei Eingabe von 42 ein binärer Suchalgorithmus als Ergebnis 7 liefern, da das Element 42 im Array a an der Position 7 steht.

Das folgende MIPS Assemblerprogramm implementiert einen solchen binären Suchalgorithmus. Existiert das zu suchende Element im Array a, gibt die Funktion `bsearch` dessen Position im Register `$v0` zurück. Konnte das Element nicht gefunden werden, enthält `$v0` nach Ausführung von `bsearch` den Wert `-1`. Die Funktion `bsearch` erwartet drei Eingabeparameter in den Registern `$a0`, `$a1` und `$a2`. (*Hinweis:* Die Instruktion `la $t1,a` in Zeile 12 lädt die Basisadresse des Arrays a in das Register `$t1`.)

```
(01) bsearch:  addi  $sp,$sp,-4
(02)           sw    $ra,0($sp)
(03)
(04)           addi  $v0,$zero,-1
(05)           slt   $t0,$a2,$a1
(06)           bne   $t0,$zero,L2
(07)
(08)           sub   $v0,$a2,$a1
(09)           srl   $v0,$v0,1
(10)           add   $v0,$v0,$a1
(11)           sll   $t0,$v0,2
(12)           la    $t1,a
(13)           add   $t0,$t0,$t1
(14)           lw    $t1,0($t0)
(15)           beq   $t1,$a0,L2
(16)
(17)           slt   $t0,$t1,$a0
(18)           beq   $t0,$zero,L1
(19)
(20)           addi  $a1,$v0,1
(21)           jal   bsearch
(22)           j     L2
(23)
(24) L1:       addi  $a2,$v0,-1
(25)           jal   bsearch
(26)
(27) L2:       lw    $ra,0($sp)
(28)           addi  $sp,$sp,4
```

```
(29)            jr    $ra
```

(a) (12 Punkte) Mit Hilfe des obigen MIPS Assemblerprogramms soll die Position des Elementes 42 im Array `a` (siehe oben) bestimmt werden. Dazu wird die Funktion `bsearch` mit den initialen Parameterwerten

```
$a0 = 42        # Element, dessen Position bestimmt werden soll
$a1 = 0         # Erste Position im Array a
$a2 = 23        # Letzte Position im Array a
```

aufgerufen. Vervollständigen Sie Tabelle 1 mit den Parameterwerten bei jedem weiteren *rekursiven* Aufruf von `bsearch` bis das Ergebnis (Position 7) feststeht.

	$a0	$a1	$a2
1. Aufruf	42	0	23
2. Aufruf			
3. Aufruf			
4. Aufruf			
5. Aufruf			

Tabelle 1: Parameterwerte bei jedem Aufruf von `bsearch`.

Lösungsvorschlag:

	$a0	$a1	$a2
1. Aufruf	42	0	23
2. Aufruf	42	0	10
3. Aufruf	42	6	10
4. Aufruf	42	6	7
5. Aufruf	42	7	7

Tabelle 2: Parameterwerte bei jedem Aufruf von `bsearch`.

□

(b) (13 Punkte) Übersetzen Sie das Assemblerprogramm in eine C-Funktion. Zur Vereinfachung sind die Argumente sowie der Rückgabetyp der C-Funktion bereits vorgegeben (siehe unten). Gemäss der obigen Beschreibung erfolgt die Parameterübergabe unter Verwendung der Argumentregister `$a0`, `$a1` und `$a2`. Nehmen Sie an, dass das Array `a` im C-Programm als globale Variable in der Form

```
int a[24] = {2,3,8,...,111,120};
```

deklariert ist. Führen Sie gegebenenfalls lokale Hilfsvariablen ein.

```
int bsearch(int toFind, int start, int end) {









}
```

Lösungsvorschlag:

```
int bsearch(int toFind, int start, int end) {
    if (start > end) {
        return -1;
    }

    int mid = start + (end - start)/2;
    if (a[mid] == toFind) {
        return mid;
    } else if (a[mid] < toFind) {
        return bsearch(toFind, mid + 1, end);
    } else {
        return bsearch(toFind, start, mid - 1);
    }
}
```

□

Aufgabe 6 : Cache (maximal 33 Punkte)

6.1: Cache Grundlagen (maximal 9 Punkte)

(a) (1 Punkt) Wie hat sich der Geschwindigkeitszuwachs bei CPUs im Gegensatz zu den Speicherzugriffszeiten in den letzten 30 Jahren qualitativ entwickelt (ein Satz)?

Lösungsvorschlag: Die Geschwindigkeit von den Speichern hat viel weniger stark zugenommen, als die Geschwindigkeit von CPUs.

□

(b) (3 Punkte) Erklären Sie die zwei Arten der Lokalität von Speicherabfragen (je ein Satz) und beschreiben Sie in Pseudocode ein kurzes Programm, welches beide Arten der Lokalität hinsichtlich des Datenzugriffs aufweist.

Lösungsvorschlag: Zeitliche Lokalität: Gleiche Adresse wird innerhalb kurzer Zeit mehrfach referenziert. Örtliche Lokalität: Beieinander gelegene Adressen werden innerhalb kurzer Zeit referenziert.

```
for(int i = 1; i < 100; i++)
{
  a[i] = i;
  a[i-1] = i;
}
```

□

(c) (3 Punkte) Erklären Sie wo ein Speicherblock platziert werden kann bei den folgenden Cache-Implementierungsalternativen: direkte Abbildung, teilassoziativ und vollassoziativ. (je ein Satz)

Lösungsvorschlag: direkte Abbildung: Speicherblock kann nur an einem bestimmten Ort im Cache platziert werden. teilassoziativ: Speicherblock kann an mehreren Stellen im Cache platziert werden. vollassoziativ: Speicherblock kann an einer beliebigen Stelle im Cache platziert werden.

□

(d) (2 Punkte) Warum werden Cache-Update Mechanismen benötigt? Erklären Sie den Unterschied zwischen den Cache-Update Mechanismen Zurückkopieren (write back) und durchgängiges Schreiben (write through) (je ein Satz).

Lösungsvorschlag: Cache-Update Mechanismen braucht es um mit der Inkonsistenz zwischen Cache-Speicher und Hauptspeicher umzugehen. Zurückkopieren: Daten werden

erst in den Hauptspeicher geschrieben, wenn der entsprechende Datensatz im Cache ersetzt wird und auf diese Adresse geschrieben wurden. durchgängiges Schreiben: Jedes mal wenn Daten im Cache geschrieben werde, werden diese Änderungen sofort auch im Hauptspeicher aktualisiert.

□

6.2: Cache Konfiguration (maximal 18 Punkte)

(a) (6 Punkte) Gegeben ist ein byteadressierter Hauptspeicher. In Abbildung 1 ist das Diagramm eines Caches gegeben. Beantworten Sie anhand dieses Diagramms die folgenden Fragen:

- Was für einer Cache-Architektur entspricht die Darstellung?
- Was ist die Grösse eines Wortes?
- Wie gross ist die benutzte Blockgrösse?
- Wie viele Cache-Einträge kann man mit dieser Konfiguration indexieren?

Lösungsvorschlag:

- Es ist ein 4-fach Assoziativ Cache.
- Die Wortgrösse ist 4 Bytes.
- Blockgrösse beträgt $2^4 = 16$ Worte.
- Es können $2^8 = 256$ Cache-Einträge indexiert werden.

□

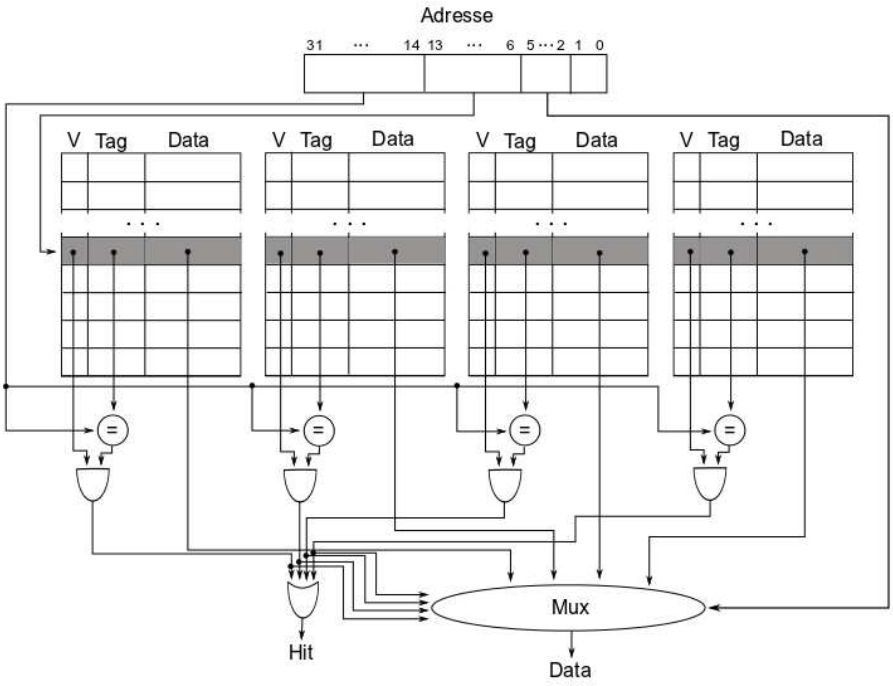


Abbildung 1: Diagramm einer Cache-Architektur.

(b) (2 Punkte) Berechnen Sie für die in Abbildung 1 gegebene Cache-Architektur die effektive Cache-Grösse in Bit.

Lösungsvorschlag: 8 Bit Index ergibt für den 4-fach Assoziativ Cache $4 \cdot 256$ Cache-Einträge mit einem Valid-Feld von 1 Bit, einem Tag von 18 Bit und Daten von $16 * 32 = 512$ Bit:

$$C = 4 \cdot 256 \cdot (1 + 18 + 512) = 543744Bit \tag{1}$$

□

(c) (2 Punkte) Berechnen Sie die effektive Cache-Grösse für einen Cache mit direkter Abbildung, wobei die gleiche Blockgrösse, Adresslänge und Anzahl Bits für den Index vorhanden sind wie bei der obigen Architektur. Berücksichtigen Sie aber zusätzlich noch das Dirty-Bit.

Lösungsvorschlag:

$$C = 256 \cdot (1 + 18 + 512 + 1) = 136192Bit \tag{2}$$

□

(d) (2 Punkte) Gegeben ist ein vollassoziativer Cache für einen 16-bit Prozessor mit folgenden Eigenschaften:

- Byteadressierung
- Wortgrösse von 2 Byte
- Blockgrösse von 4 Worten
- 16 Cache-Zeilen

Geben Sie die Aufteilung der 16-bit Adresse in Tag, Word-Offset und Byte-Offset an:

Lösungsvorschlag: Das unterste Bit ist für den Byte-Offset, die nächsten zwei Bits für den Word-Offset und der Rest (13 Bits) für den Tag.

□

(e) (6 Punkte) Für diese Aufgabe gilt der gleiche Cache wie in Aufgabe (d). Nehmen Sie an, dass er Cache leer ist und eine Sequenz von Leseoperationen ausgeführt wird. Schreiben Sie in Tabelle 1 den Zustand der Cache-Zeile nach dem jeweiligen Speicherzugriff auf und geben Sie an ob es einen Fehlzugriff (Miss) gegeben hat oder nicht (Hit).

Sequenz:

- Lesen 0x42
- Lesen 0x44
- Lesen 0x46
- Lesen 0x48
- Lesen 0x40

Lösungsvorschlag:

Hex Adresse	Binäre Adresse			Hit/ Miss
	Tag	Word-Offset	Byte-Offset	
0x21	1000	01	0	Miss
0x22	1000	10	0	Hit
0x23	1000	11	0	Hit
0x24	1001	00	0	Miss
0x20	1000	00	0	Hit

Tabelle 1: Füllen Sie die Cache-Einträge für die gegebene Sequenz aus.

□

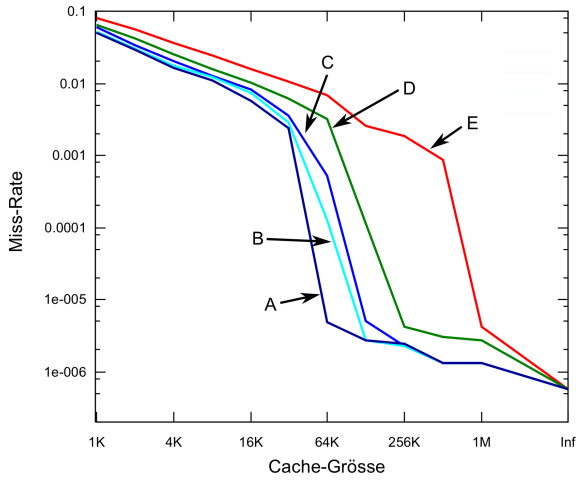


Abbildung 2: Miss-Rate vs. Cache-Grösse für eine bestimmte Menge von Programmen (SPEC CPU2000).

6.3: Einfluss der Cache-Grösse auf die Miss-Rate (maximal 6 Punkte)

Beantworten Sie die unten stehenden Fragen anhand Abbildung 2, die die Abhängigkeit zwischen Miss-Rate und Cache-Grösse bei der Ausführung einer bestimmten Menge von Programmen (SPEC CPU2000) zeigt.

(a) (3 Punkte) Ordnen Sie den 5 Linien in Abbildung 2 folgende Cache-Implementierungen zu:

- direkte Abbildung:
- 2-fach Assoziativ:
- 4-fach Assoziativ:
- 8-fach Assoziativ:
- Vollassoziativ:

Lösungsvorschlag:

- direkte Abbildung:E
- 2-fach Assoziativ: D
- 4-fach Assoziativ: C
- 8-fach Assoziativ: B
- Vollassoziativ: A

□

(b) (3 Punkte) Erklären Sie den generellen Trend der Abhängigkeit zwischen Miss-Rate und Cache-Grösse (ein Satz) und begründen Sie, weshalb die Unterschiede zwischen den Cache-Implementierung abnehmen für sehr kleine und sehr grosse Cache-Grössen (je ein Satz).

Lösungsvorschlag:

Mit zunehmender Cache-Grösse nimmt die Miss-Rate ab. Bei sehr kleinen Cache-Grössen macht es die Cache-Implementierung fast keinen Unterschied, da praktisch keine Daten im Cache platz haben. Da kann man noch so eine gute Strategie haben. Bei sehr grossen Cache-Grössen ist die Implementierung ebenso nicht so entscheidend, da auch bei einer schlechten oder simplen Strategie, es im Cache immer Platz für Daten hat.

□