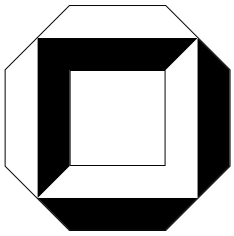


**Universität Karlsruhe**
Institut für Rechnerentwurf und Fehlertoleranz
Prof. Dr. Wolfgang Karl

# Klausur Rechnerstrukturen
# Sommersemester 2004
# Musterlösung

Aushang der Ergebnisse: ab 15. September 2004

# Musterlösung 1 (Leistungsbewertung) *10 Punkte*

a) zyklen = codesize*CPI, takt = zyklen/zeit *2P*

A: z=4.500.000, t=2.250.000.000 Hz = 2250 oder $\frac{9}{4} * 10^9$ MHz
B: z=2.500.000, t=1.125.000.000 Hz = 1250 oder $1.25 * 10^9$ MHz
→ Prozessor A hat den höheren Wert.

b) MIPS = takt/CPI *1P*
A: 1500 MIPS, B: 1250 MIPS
→ Prozessor A hat den besseren Wert.

c) Taktrate für sich alleine und auch MIPS als Verknüpfung von Takt und CPI sind nicht aussagekräftig. Der Prozessortakt alleine sagt nichts über die tatsächliche Rechenleistung aus und ist ohne die Angabe von CPI wertlos. *2P*

CPI sagt zwar etwas über die Effizienz der Architektur und somit scheint die MIPS-Angabe für einen direkten Leistungsvergleich tauglich, jedoch sind hier wichtige Parameter nicht ablesbar: MIPS können ineffektiv (lange Pipeline, hohe Taktrate) oder effektiv (kurze Pipeline, niedrige Taktrate) erzeugt worden sein.

Weiterhin fließt die Codegüte mit in die Bewertung ein, d.h. für einen fairen Vergleich muß sichergestellt sein, daß der verwendete Compiler von gleicher Qualität ist und auch mit gleichen Optimierungseinstellungen betrieben wird.

d) Anzahl Instruktionen: $(300 + 75 + 150 + 25) * 10^3 = 550.000$ *5P*

Taktzyklen: $(300 * 1 + 75 * 2 + 150 * 3 + 25 * 4) * 10^3 = 1.000.000$

Zykluszeit: $\frac{1}{4GHz} = 0.25 * 10^{-9}s = 0.25ns$

Ausführungszeit: $\#Zyklen * Zykluszeit = 1000 * 10^3 * 0.25 * 10^{-9} = 250 * 10^{-6}s = 250\mu s$ oder $\frac{1000}{4} * 10^{-6}$

CPI: $\frac{Zyklen}{Instruktionen} = \frac{1000*10^3}{550*10^3} = \frac{100}{55} = \frac{20}{11} \approx 1.82$

MIPS: $\frac{Instruktionen}{Ausfuehrungszeit*10^6} = \frac{550.000}{250} = 2200$

MFLOPS: wie MIPS, wobei Anzahl der Befehle und Ausführungszeit nur für Fließkommaberechnung
$\frac{75.000}{(75.000*2)*(0.25*10^{-9})*10^6} = \frac{1}{0.5*10^{-3}} = 2000$

# Musterlösung 2 (Prozessorarchitektur) *15 Punkte*

a) Beide Schleifen werden je 2x durchlaufen, d.h. der Schleifenkörper wird insgesamt 4x abgearbeitet. *1P*

b) *3P*

*Variante 1: Lösung für einen einzigen Branch Predictor:*

| | Inhalt Register | | Prädiktor | | |
|---|---|---|---|---|---|
| **Befehl** | `r1` | `r0` | alt | neu | Vorhersage |
| Init | 1 | 1 | — | NT | — |
| BREQ loop2 | 0 | — | NT | T | falsch |
| BREQ loop2 | -1 | — | T | NT | falsch |
| BREQ loop1 | — | 0 | NT | T | falsch |
| BREQ loop2 | 0 | — | T | T | richtig |
| BREQ loop2 | -1 | — | T | NT | falsch |
| BREQ loop1 | — | -1 | NT | NT | richtig |

*Variante 2: Lösung für individuelle Branch Predictors:*

| | Inhalt Register | | Prädiktor 1 | | Prädiktor 2 | | |
|---|---|---|---|---|---|---|---|
| **Befehl** | `r1` | `r0` | alt | neu | alt | neu | Vorhersage |
| Init | 1 | 1 | — | NT | — | NT | — |
| BREQ loop2 | 0 | — | NT | T | — | — | falsch |
| BREQ loop2 | -1 | — | T | NT | — | — | falsch |
| BREQ loop1 | — | 0 | — | — | NT | T | falsch |
| BREQ loop2 | 0 | — | NT | T | — | — | falsch |
| BREQ loop2 | -1 | — | T | NT | — | — | falsch |
| BREQ loop1 | — | -1 | — | — | T | NT | falsch |

c) *Variante 1:* Ja, gibt es. Mit der Initialisierung auf $T$ wird zumindest die erste Fehlvorhersage vermieden. *1P*

*Variante 2:* Ja, gibt es. Mit der Initialisierung auf $T$ werden zwei Fehlvorhersagen (erstmaliger Sprung nach loop2 bzw. loop1) vermieden.

d) Ein (m,n)-Korrelationsprädiktor nutzt das Verhalten der letzten $m$ Sprünge für die Auswahl aus $2^m$ Prädiktoren. Jeder Prädiktor stellt einen $n$-Bit Prädiktor für den jeweils einzelnen Sprung dar. Die globale Vergangenheit der letzten $m$ Sprünge kann in einem $m$-Bit Schieberegister, dem Sprungverlaufsregister/Branch History Register (BHR), gespeichert werden. Der Inhalt des BHR wird als Adresse benutzt, um eine Sprungsverlaufstabelle (Pattern History Table, PHT) zu selektieren. Hierdurch kann die Sprunghistorie, d.h. Sprungmuster erkannt, werden. *3P*

Für $m = 0$ verhält sich ein Korrelationsprädiktor wie der einfache n-Bit Prädiktor.

e) Da nur Geschwindigkeit gewertet wird und der Prozessor nicht über die Möglich- *1P*

keit zur prädizierten Ausführung verfügt, bedienen wir uns des sogenannten Loop-Unrolling und kopieren einfach den Schleifenkörper 4x hintereinander. Dadurch werden die durch die Schleifenabfragen verursachten Sprünge vermieden.

f) Bei VLIW- (und auch EPIC-) Prozessoren wird die Parallelisierung statisch durch den Compiler vorgenommen. In superskalaren Prozessoren wird durch entsprechende Funktionseinheiten eine dynamische Parallelisierung erzielt. *1P*

g) Bei Precise Interrupts muß darauf geachtet werden, daß: *2P*

- alle Resultate von Befehlen, die in der Programmreihenfolge vor dem Ereignis stehen, gültig gemacht und
- die Resultate aller nachfolgenden Befehle verworfen werden.
- Das Ergebnis des verursachenden Befehls wird in Abhängigkeit der Architektur oder Art der Unterbrechung gültig gemacht oder verworfen, ohne weitere Auswirkungen zu haben.

Die Wiederherstellung der ursprünglichen Semantik erfolgt durch den Reorder-Buffer.

h) *1.5P*

Der Begriff *Datenabhängigkeit* ist wörtlich zu verstehen und beschreibt nur das Vorhandensein einer grundsätzlichen Abhängigkeit *im Befehlsstrom*. Sie kann, aber muß nicht zwingend, zu einem Pipeline-Konflikt – hier Datenkonflikt – führen: Ein solcher Konflikt tritt nur dann auf, wenn ein Operand in der Pipeline (noch) nicht verfügbar ist oder das Register bzw. der Speicherplatz, in den ein Resultat geschrieben werden soll, noch nicht zur Verfügung steht.

i) *1.5P*

Diese Arten von Datenabhängigkeiten sind nicht problemimmanent, sondern werden durch Mehrfachnutzung von Speicherplätzen (Register oder Arbeitsspeicher) hervorgerufen. Sie können durch Variablenumbenennungen entfernt werden.

# Musterlösung 3 (Speicherhierarchie/Caches) *10 Punkte*

a) *3P*

**direkt zugeordnet / direct-mapped:** feste und eindeutige Zuordnung von Speicher- zu Cache-Adressen, darum sehr einfache Hardware-Struktur und sehr schnell. Die feste Zuordnung wirkt sich jedoch nachteilig bei der Cache-Verdrängung aus.

**vollassoziativ / fully associative:** freie Zuordnung von Speicher- zu Cache-Adressen. Extrem flexibel, aber auch sehr aufwendig, da pro Cache-Zeile ein eigener Vergleicher vorhanden sein muß.

**n-fach satzassoziativ / n-way set associative:** Mittelweg zwischen den beiden Erstgenannten: Es wird hardwaremäßig lediglich eine Zuordnung von Adressen zu sogenannten Sets getroffen. Die einzelnen Zeilen innerhalb eines Sets können jedoch vollassoziativ belegt werden. Geringfügig höherer Hardwareaufwand als direkt zugeordnet, jedoch in der Praxis ähnlich flexibel wie vollassoziativ.

b) Es handelt sich hierbei um den sogenannten Trace Cache. Im Unterschied zum Instruktions-Cache, welcher die Befehlsfolgen statisch, d.h. in der vom Compiler erzeugten Abfolge, speichert, wird im Trace-Cache die dynamische – d.h. von den internen Einheiten spekulativ erzeugte – Befehlsfolge ("traces") abgelegt. *2P*

c) $t_A = r_H * t_H + r_M * t_M = r_H * t_H + (1 - r_H) * t_M$ *1P*
$\rightarrow t_A = (3 * 0.9 + 0.1 * 100)ns = 12.7ns$

d) $t_{A1} = (0.8 * 2.5 + (1 - 0.8) * 50)ns = 12ns$ *2P*

$t_{A2} = r_{H1} * t_{H1} + r_{M1} * (r_{H2} * t_{H2} + r_{M2} * t_{Hm})$
$\rightarrow t_{A2} = (0.8 * 2.5 + 0.2 * (0.25 * 16 + 0.75 * 50))ns = 10.3ns$

e) Hit rate erhöhen bzw. miss rate vermindern. Kann geschehen durch Veränderung der Cache-Parameter (Organisation, Blockgröße, Assoziativität: bedingt ggf. höhere Komplexität), Vergrößerung des Cache Speichers (ggf. Verlangsamung der Zugriffszeit), Einfügen einer weiteren Cache-Stufe (teuer) oder programmtechnische Maßnahmen (aufwendig). *1P*

f) Die Geschwindigkeit von Elektronen ist endlich: Soll der Cache in nur einem Taktzyklus zugreifbar sein, müssen die Elektronen diesen auch schnell genug durcheilen können. Ist dies aufgrund der Größe nicht möglich, müssen Wartezyklen eingefügt werden, was zur Verlangsamung des Zugriffes führt. *1P*

# Musterlösung 4 (Cache-Kohärenz und Parallelverarbeitung) *15 Punkte*

a) Die Cache-Zeile wird um zwei Zustandsbits erweitert. Diese Bits kodieren die 4 Zustände des MESI-Protokolls. *1P*

b) *4P*

- Invalid (I), die Cache Zeile ist ungültig. Bei Zugriff auf diese Zeile, muß die Cache-Zeile mit dem betreffenden Speicherblock geladen werden. Bei Lesezugriffen Übergang auf S oder E, je nachdem, ob der Block gegenwärtig gemeinsam oder exklusiv genutzt wird. Bei Write-Miss Wechsel nach M und Ausgabe von Invalidate.
- Shared Unmodified (S), Speicherblock existiert als (unveränderte) lokale und entfernte Kopie. Bei Write-Hit Änderung der Zeile und Wechsel in M sowie Ausgabe des Invalidate-Signales. Andere Caches, bei denen diese Zeile ebenfalls im Zustand S ist, kennzeichnen diese als ungültig (I)
- Exclusive Unmodified (E), Speicherblock existiert nur als lokale Kopie, kein externer Buszugriff. Bei Schreibzugriff Wechsel nach M. Andere Caches sind nicht betroffen.
- Exclusive Modified (M), Speicherblock wurde nach dem Laden verändert und existiert nur als lokale Kopie. Bei entferntem Zugriff auf diesen Block (Snoop Hit), muß der Block zurückkopiert werden und der Zustand wechselt nach S oder I. Der entfernt zugreifende Prozessor wird per Retry informiert, daß zunächst ein Zurückschreiben erforderlich ist.

c) Voraussetzung für das Erkennen von externen Speicherzugriffen ist eine Einheit zur Busüberwachung, dem sogenannten *Bus Snooping*. *1P*

d) siehe nächste Seite (vgl. Foliensatz 2004/17, S.17) *4P*

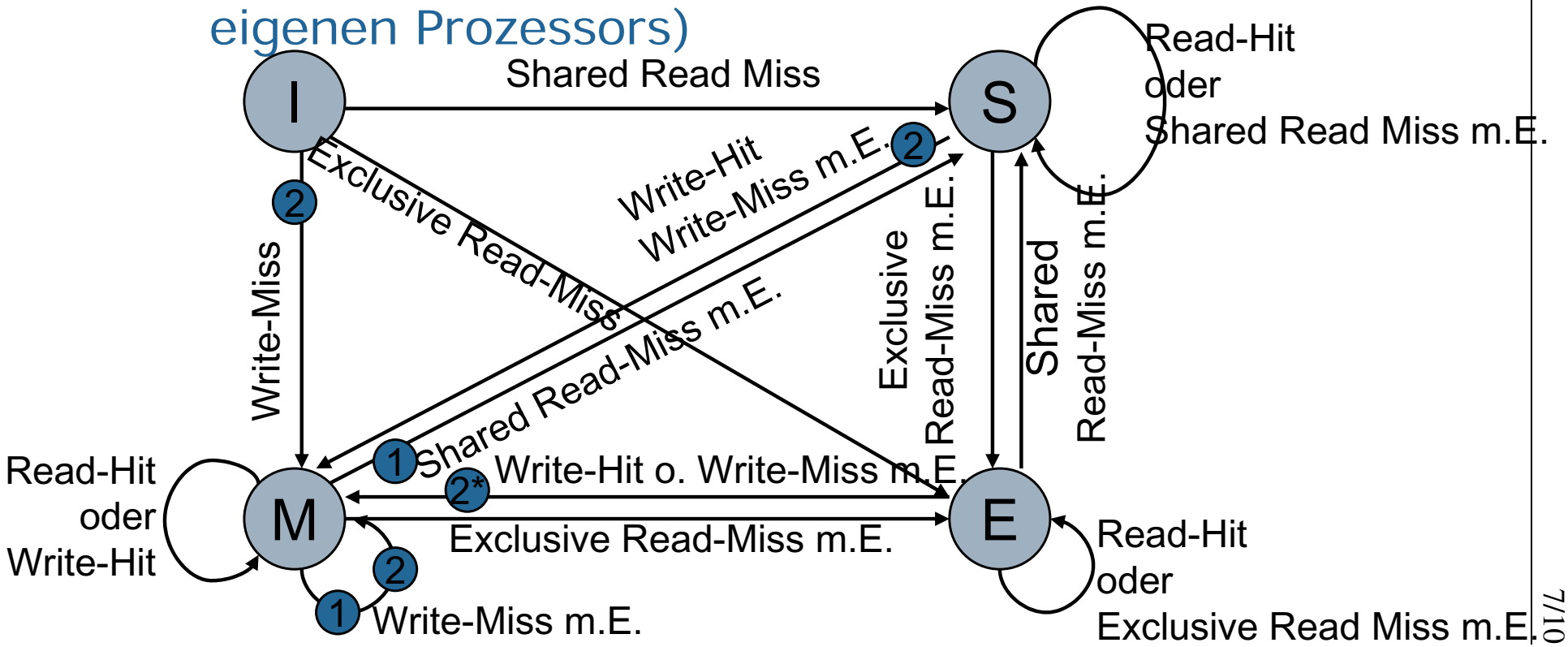
e) *2P*

- Shared Memory: implizite Kommunikation, semantisch ähnlich zur seriellen Programmierung, OpenMP
- Message Passing: Explizite Kommunikation im Quellcode, PVM&MPI

# Aktualisierungsstrategie und Cache-Kohärenz

## Zustandsgraph des MESI-Protokolls

- Zustandsübergänge, die durch die lokalen Schreib- und Lesezugriffe ausgelöst werden (Zugriffe des eigenen Prozessors)




m.E.: mit Ersetzung

1 Cache-Zeile wird in den Hauptspeicher zurückkopiert. (Line-Flush)

2 Cache-Zeilen in den anderen Caches mit gleicher Blockadresse werden invalidiert. (Line Clear)

f) Diskonnektivität: $d = \frac{N}{e} \rightarrow \frac{15}{5} = 3$ *1.5P*
Kosteneffektivität: Verbindungsgrad*max(Diameter,d)=$3 * max(2, 3) = 9$

g) Die Bisektionsbreite gibt die Belastbarkeit eines Netzwerkes an; je größer, umso belastbarer ist das Netz. *1.5P*

Konnektivität definiert Zusammenhang (Grad der Vernetzung) und somit die Ausfallsicherheit; je größer, umso ausfallsicherer ist ein Netz.

# Musterlösung 5 (Fehlertoleranz) *10 Punkte*

a) Permanente (permanent), sporadische (intermittent), und vorübergehende (transient) Fehler *1.5P*

b) Entwurfsfehler, Implementationsfehler, Bedienungsfehler bzw. Fehler zur Laufzeit/Verschleiß, Wartungsfehler *1.5P*

c) Badewannenkurve: starker Ausfall zu Beginn (Initialausfälle bei Inbetriebnahme aufgrund von Fertigungsfehlern o.ä.), langsam ansteigende Senke (Alterungseffekt) und schließlich starker Anstieg (Altersausfälle) *1P*

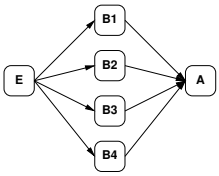
d) Im ersten Fall teilt sich die Bremskraft 80:20 zwischen den beiden Kreisläufen auf und ergibt sich somit zu $0.8 * \phi(K1) + 0.2 * \phi(K2)$. *3P*

Im zweiten Fall teilen sich je eine Vorder- und Hinterradbremse die Bremskraft, d.h. es gilt $\frac{0.8+0.2}{2} * (\phi(K1) + \phi(K2)) = 0.5 * (\phi(K1) + \phi(K2))$

Bei Ausfall von K1 kann im ersten Fall das Kfz nicht mehr sicher zum Stehen gebracht werden. Im zweiten Fall kann das Kfz bei Ausfall eines Bremskreises immer sicher zum Stehen gebracht werden.

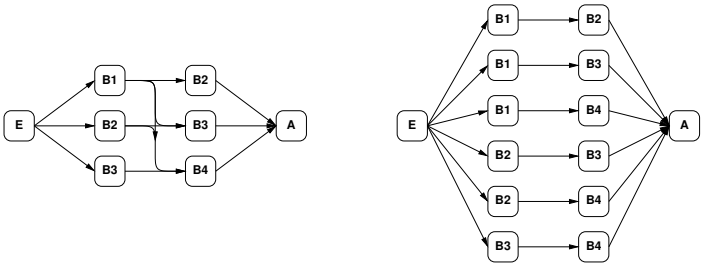
e) *2P*

*Variante 1: unabhängige Bremsen, eine reicht aus für Stillstand:*




S=B1 or B2 or B3 or B4

*Variante 2: unabhängige Bremsen, 2-aus-4 System:*




S=B1B2 or B1B3 or B1B4 or B2B3 or B2B4 or B3B4

f) *Variante 1:* *1P*

$$\phi(S) = 1 - (1 - 0.99)^4 = 0.99999999$$

*Variante 2:*

$$\phi(S) = \sum_{k=2}^{4} \binom{4}{k} * 0.99^k * (1 - 0.99)^{(4-k)}$$