Klausur zur Vorlesung

Grundlagen der Rechnerarchitektur / Technische Informatik (GRA/TI)

Prof. Marco Platzner
Fachgebiet Technische Informatik
Universität Paderborn

22.09.2015

- Die Bearbeitungsdauer beträgt für alle Studenten **90 Minuten.** Es sind **alle 5 Aufgaben** zu bearbeiten.
- Es sind keine Hilfsmittel zugelassen.
- Schreiben Sie nicht mit Bleistift oder Rotstift.
- Verwenden Sie kein eigenes Papier. Bei Bedarf bekommen Sie Papier bei der Klausuraufsicht.
- Schreiben Sie auf jedes Blatt (auch auf das Konzeptpapier) in Blockschrift Ihren Namen und Ihre Matrikelnummer.
- Bei mehreren präsentierten Lösungen wird die Aufgabe nicht gewertet! Streichen Sie daher bei Angabe mehrerer Lösungsansätze die nicht zu bewertenden Lösungen durch! Verwenden Sie kein Tipp-Ex.
- Abschreiben und abschreiben lassen oder Hilfe Dritter führt zum Nichtbestehen der Klausur.

Nachname: _____

Vorname: _____

Matrikelnummer: _____

Studiengang: _____

Aufkleber

Aufgabe	**1**	**2**	**3**	**4**	**5**	$\sum$
Punkte	**15**	**20**	**25**	**15**	**15**	90
Erreicht						

Aufgabe 1 (Multiple Choice) **[15 Punkte]**

Bei den folgenden Fragen können keine, eine oder mehrere Antworten richtig sein. Kreuzen Sie die richtigen Antworten deutlich an.

(a) Möglichkeiten zur Reduktion von Data Hazards sind ...

- ☐ feste Instruktionslängen
- ☐ Forwarding
- ☐ Umordnen der Instruktionen
- ☐ dynamische Sprungvorhersage

(b) In welchen Fällen ist der Branch Prediction Buffer (BPB) 2 KBit groß?

- ☐ 10 LSB der Adresse für BPB-Indizierung verwendet, 2-bit Prädiktor
- ☐ 11 LSB der Adresse für BPB-Indizierung verwendet, 1-bit Prädiktor
- ☐ 8 LSB der Adresse für BPB-Indizierung verwendet, (2,2) korrelierender Prädiktor
- ☐ 6 LSB der Adresse für BPB-Indizierung verwendet, (4,1) korrelierender Prädiktor

(c) Welche Maßnahmen reduzieren die Miss-rate eines Caches?

- ☐ höherer Grad der Assoziativität
- ☐ breitere Speicherorganisation
- ☐ größerer Cache
- ☐ Prefetching

(d) Was ist *Predication*?

- ☐ Sprungvorhersage zur Auflösung von Control Hazards
- ☐ die bedingte Ausführung von Instruktionen, abhängig von dem Wert eines Booleschen Registers
- ☐ eine Technik zur Eliminierung von bedingten Sprüngen
- ☐ eine Strategie zur Ersetzung von Cacheblöcken

(e) Auf einem Prozessor mit einer Taktfrequenz von 1 GHz werden 10^6 Instruktionen in 0,5 ms abgearbeitet. Um welche Prozessorarchitektur könnte es sich handeln?

- ☐ superskalarer Prozessor
- ☐ VLIW Prozessor
- ☐ Einzyklenimplementierung
- ☐ Mehrzyklenimplementierung

Aufgabe 2 (Instruktionssatzarchitekturen) **[20 Punkte]**

(a) Es sollen die drei Instruktionssatzarchitekturen *Stack-*, *Akkumulator-* und *Load-Store-Architektur* hinsichtlich des Speicherbedarfs und der Speicherbandbreite untersucht werden. Erstellen Sie dazu für die Anweisung

```
D = A * B + (A - C * B)
```

für jede der drei Architekturen eine Assemblercodesequenz. Bestimmen Sie den Speicherbedarf der Instruktionen und die Menge der übertragenen Daten in Bytes. Geben Sie für die Stack- und Akkumulator-Architektur außerdem die Belegung des Stacks (Abb. 1) und des Akkumulators (Abb. 3) an.

Die Operanden `A`, `B` und `C` sind 32 bit groß und liegen im Speicher. Das Ergebnis `D`, ebenfalls 32 bit groß, muss nach Berechnung in den Speicher abgelegt werden. Nehmen Sie an, dass der `opcode` einer Instruktion 1 Byte beträgt, die Speicheradressen 16 bit und die Registeradressen 4 bit lang sind. Die Instruktionslänge muss ein Vielfaches von einem Byte sein.

(i) **Stack-Architektur**. Verfügbare Instruktionen: `push`, `pop`, `add`, `sub`, `mul`

	Assembler-Instruktion	Speicherbedarf	Übertragene Daten
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Tabelle 1: Programm in Stack-Architektur

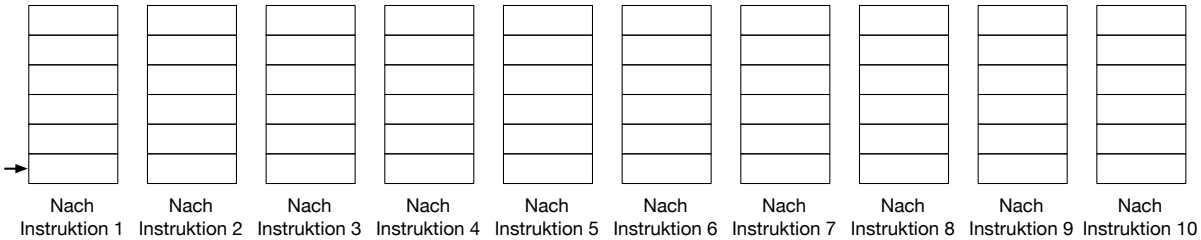


Abbildung 1: Belegung des Stacks

	Assembler-Instruktion	Speicherbedarf	Übertragene Daten
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Tabelle 2: Programm in Stack-Architektur - **Ersatzlösung, ungültige Lösung streichen!**

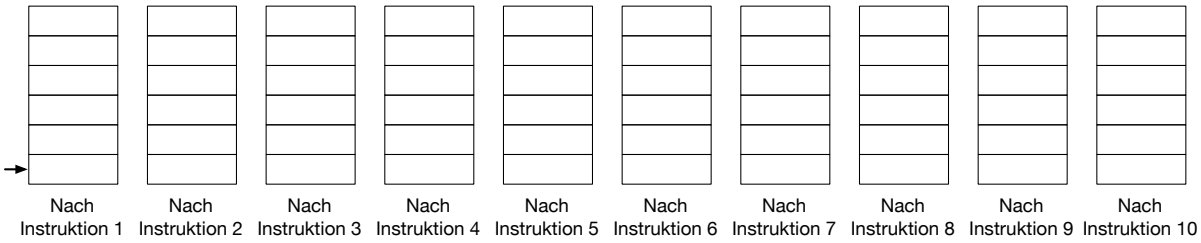


Abbildung 2: Belegung des Stacks - **Ersatzlösung, ungültige Lösung streichen!**

(ii) **Akkumulator-Architektur**. Verfügbare Instruktionen: `load`, `store`, `add`, `sub`, `mul`

	Assembler-Instruktion	Speicherbedarf	Übertragene Daten
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Tabelle 3: Programm in Akkumulator-Architektur

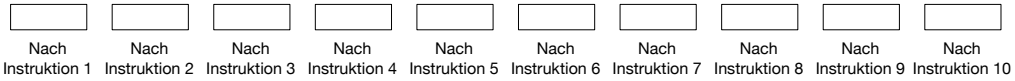


Abbildung 3: Belegung des Akkumulators

	Assembler-Instruktion	Speicherbedarf	Übertragene Daten
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Tabelle 4: Programm in Akkumulator-Architektur - **Ersatzlösung, ungültige Lösung streichen!**

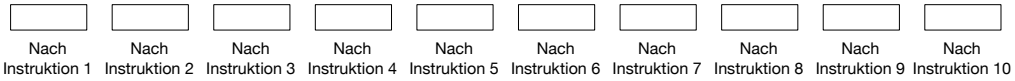


Abbildung 4: Belegung des Akkumulators - **Ersatzlösung, ungültige Lösung streichen!**

(iii) **Load-Store-Architektur**. Verfügbare Instruktionen: `lw`, `sw`, `add`, `sub`, `mul`. Nehmen Sie an, dass sie beliebig viele Register zur Verfügung haben. Verwenden Sie die aus der Vorlesung bekannte MIPS Syntax, aber die in dieser Aufgabe definierte Instruktionskodierung.

	Assembler-Instruktion	Speicherbedarf	Übertragene Daten
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Tabelle 5: Programm in Load-Store-Architektur

	Assembler-Instruktion	Speicherbedarf	Übertragene Daten
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Tabelle 6: Programm in Load-Store-Architektur - **Ersatzlösung, ungültige Lösung streichen!**

(b) Nennen Sie jeweils einen Vorteil von fester und variabler Instruktionslänge und erläutern Sie ihn kurz.

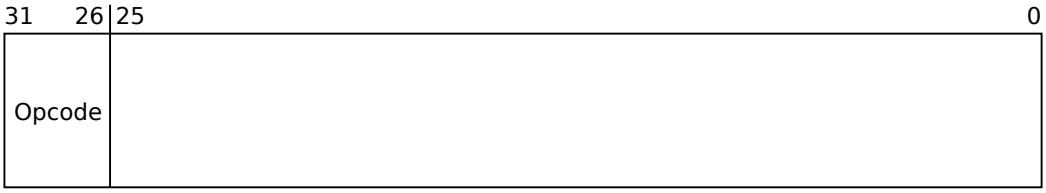
Aufgabe 3 (Prozessorentwurf) **[25 Punkte]**

In dieser Aufgabe wird die in der Vorlesung besprochene Mehrzyklenimplementierung eines MIPS-Prozessors betrachtet. Sie soll um einen Befehl `addm` erweitert werden:

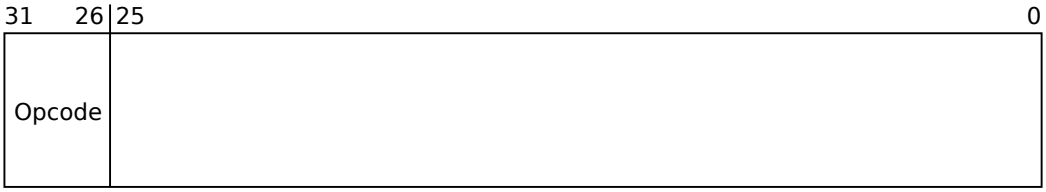
```
addm $s0, imm($s1)     # $s0 = $s0 + mem[imm + $s1]
```

(a) Bestimmen Sie einen geeigneten Instruktionstypen für `addm`. Tragen Sie in die unten stehende Abbildung die entsprechende Instruktionskodierung ein. Kennzeichnen Sie in der Abbildung anhand des obigen Beispiels mit welchen Werten die Instruktionsfelder belegt werden. Nicht benötigte Instruktionsbits können mit `X` (don't care) gekennzeichnet werden.

Instruktionstyp: _____



Ersatz, ungültige Lösung streichen!



(b) Geben Sie eine Befehlsfolge mit den in der Vorlesung besprochenen Assemblerbefehlen an, um `addm` alternativ abbilden zu können. Wie viele Taktzyklen benötigt diese Befehlsfolge?

_____
_____
_____
_____
_____
_____
_____

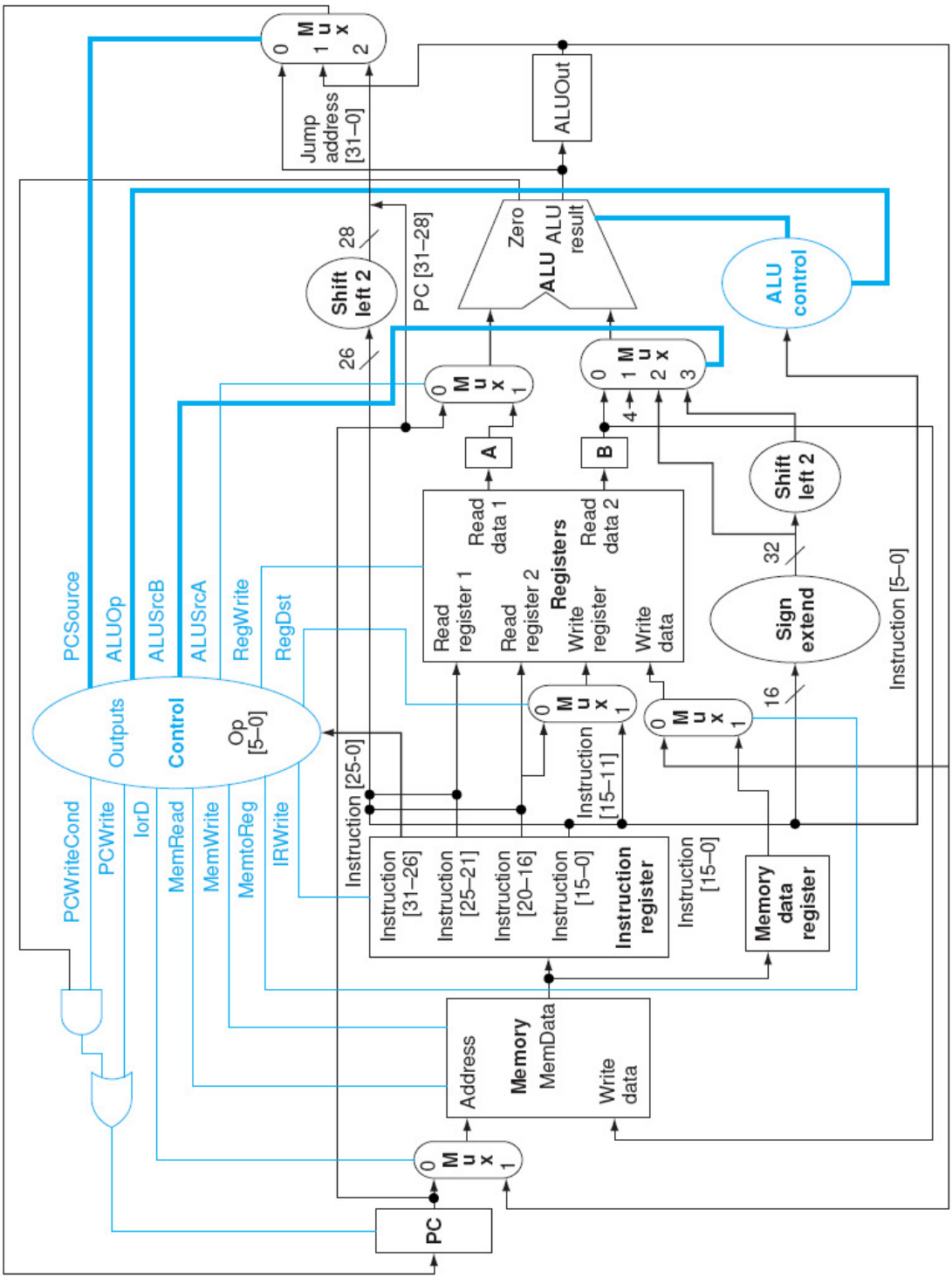
Anzahl Taktzyklen: _____

(c) Geben Sie einen Ablauf des `addm`-Befehls mit Hilfe der in der Vorlesung besprochenen Pseudocode-Notation für die verschiedenen Phasen, die `addm` durchläuft, an. Wie viele Taktzyklen benötigt `addm`?

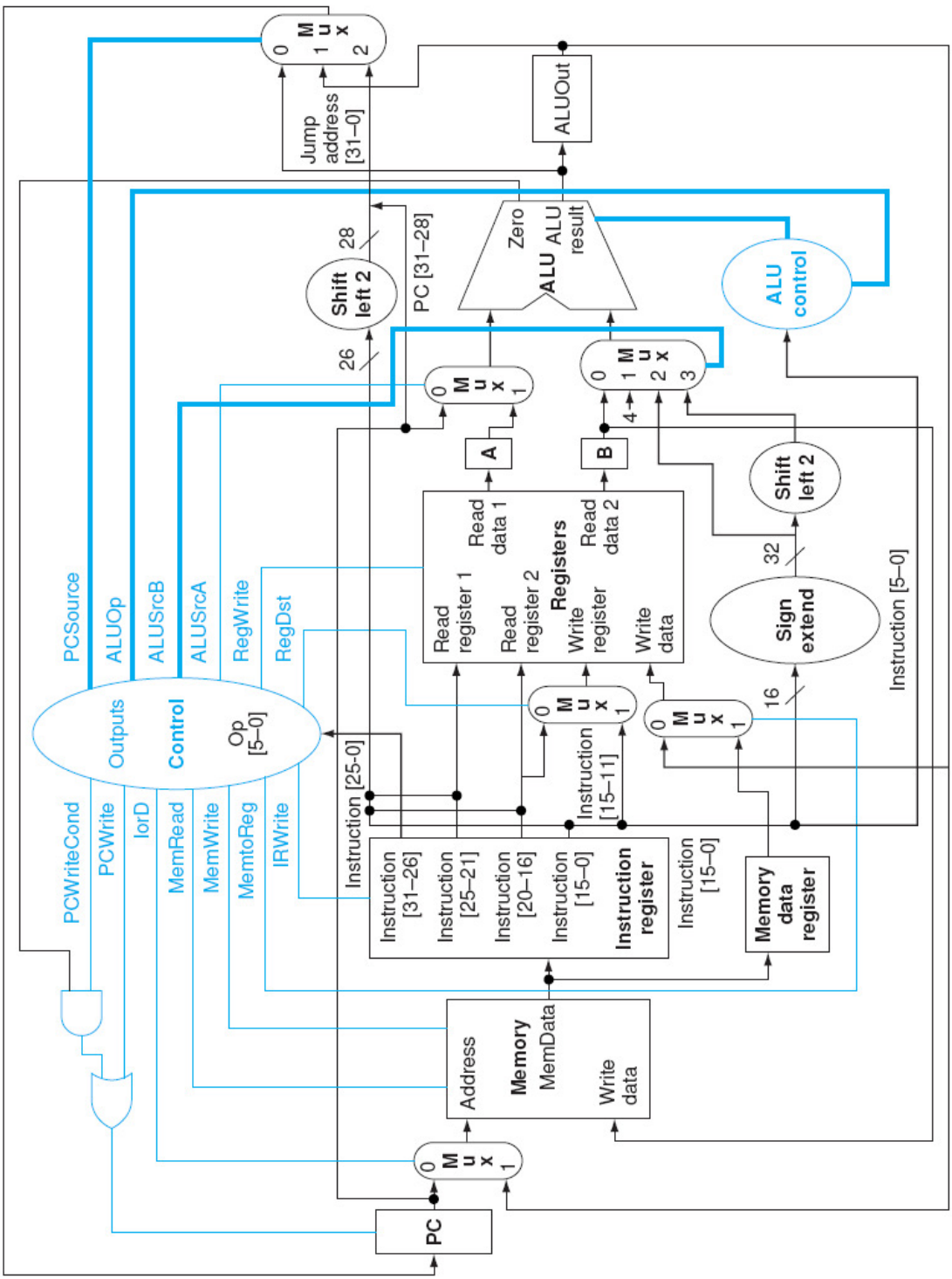
```
IF: IR <= Memory[PC];
    PC <= PC + 4;

ID: A <= Reg[IR[25:21]];
    B <= Reg[IR[20:16]];
    ALUOut <= PC + (sign-extend(IR[15:0]) << 2);
```

Anzahl Taktzyklen: _____

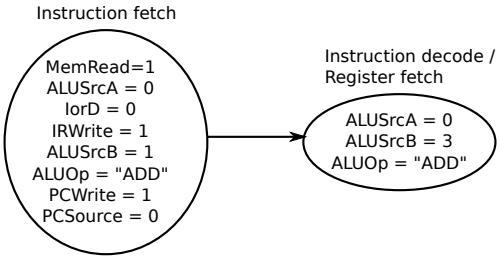
(d) Erweitern Sie den Datenpfad und die Steuersignale der Mehrzyklenimplementierung, um `addm` ausführen zu können.



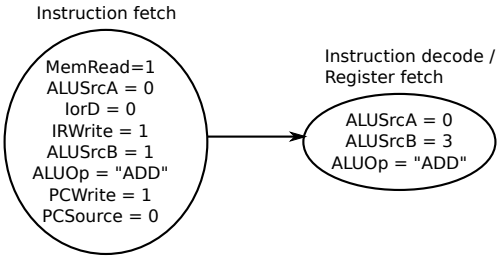
Ersatz, ungültige Lösung streichen!



(e) Vervollständigen Sie den Automaten der Steuereinheit für den `addm`-Befehl.



Ersatzautomat, ungültige Lösung streichen!



(f) Nun soll die relative Performance zwischen einer klassischen Implementierung ohne Erweiterung und der Implementierung mit `addm` bestimmt werden, wobei beide Architekturen mit der gleichen Taktfrequenz betrieben werden. Dazu sei ein passendes Benchmark-Programm gegeben.

Mehrzyklenimplementierung A ohne Erweiterung: Das Programm hat 10.000 Instruktionen.

Instruktion	Häufigkeit	Anzahl Takte
`add`	20%	
Sonstige R-Typ	20%	
`lw`	30%	
`sw`	20%	
Branch (`j` und `beq`)	10%	

Mehrzyklenimplementierung B mit `addm`: Das Programm hat 9.000 Instruktionen.

Instruktion	Häufigkeit	Anzahl Takte
`add`	15%	
`addm`	10%	
Sonstige R-Typ	20%	
`lw`	25%	
`sw`	20%	
Branch (`j` und `beq`)	10%	

Geben Sie die relative Performance zwischen den Implementierungen A und B an.

Aufgabe 4 (Branch Delay Slots) **[15 Punkte]**

Gegeben sei eine Architektur mit einem Delay Branch Slot. Formen Sie die drei Assemblerprogramme so um, dass der Delay Slot stets ausgenutzt wird.

(a)

```
01:     add  $3, $2, $1       01:_____________________________________
02:     add  $6, $1, $4       02:_____________________________________
03:     add  $2, $5, $3       03:_____________________________________
04:     add  $5, $4, $3       04:_____________________________________
05:     beq  $5, $zero, L1    05:_____________________________________
06:     nop                   06:_____________________________________
...                              _____________________________________
99: L1: ...                      _____________________________________
```

Ersatzlösung, ungültige Lösung streichen!

```
01:     add  $3, $2, $1       01:_____________________________________
02:     add  $6, $1, $4       02:_____________________________________
03:     add  $2, $5, $3       03:_____________________________________
04:     add  $5, $4, $3       04:_____________________________________
05:     beq  $5, $zero, L1    05:_____________________________________
06:     nop                   06:_____________________________________
...                              _____________________________________
99: L1: ...                      _____________________________________
```

(b)

```
01:     lw   $1, 0($a0)       01:_____________________________________
02: L1: add  $1, $1, $1       02:_____________________________________
03:     blt  $1, $2, L1       03:_____________________________________
04:     nop                   04:_____________________________________
                              05:_____________________________________
                              06:_____________________________________
```

Ersatzlösung, ungültige Lösung streichen!

```
01:     lw   $1, 0($a0)       01:_____________________________________
02: L1: add  $1, $1, $1       02:_____________________________________
03:     blt  $1, $2, L1       03:_____________________________________
04:     nop                   04:_____________________________________
                              05:_____________________________________
                              06:_____________________________________
```

(c)

```
01:      add  $1, $2, $3       01:_____________________________________
02:      beq  $1, $5, L1       02:_____________________________________
03:      nop                    03:_____________________________________
04:      add  $1, $3, $4       04:_____________________________________
05:      add  $2, $7, $8       06:_____________________________________
06: L1:  sll  $1, $1, 1        05:_____________________________________
                                  _____________________________________
                                  _____________________________________
```

Ersatzlösung, ungültige Lösung streichen!

```
01:      add  $1, $2, $3       01:_____________________________________
02:      beq  $1, $5, L1       02:_____________________________________
03:      nop                    03:_____________________________________
04:      add  $1, $3, $4       04:_____________________________________
05:      add  $2, $7, $8       06:_____________________________________
06: L1:  sll  $1, $1, 1        05:_____________________________________
                                  _____________________________________
                                  _____________________________________
```

(c) Welche Einschränkungen gibt es bei (b) und (c) bezüglich der Verwendung von Registern nach den Codesequenzen?

Aufgabe 5 (Vergleich von Caches) **[15 Punkte]**

Für diese Aufgabe sei ein 32-bit Prozessor mit einem (byte-adressierten) Adressraum von 1 GB (= 2^{30} Bit) und einem 25 KB großen Cache gegeben, für den Sie verschiedene Versionen mit folgenden Charakteristika untersuchen sollen:

Cache	**Assoziativität**	**Blockgröße**
C1	4-fach satzassoziativ	1 Wort
C2	5-fach satzassoziativ	1 Wort
C3	4-fach satzassoziativ	16 Worte
C4	8-fach satzassoziativ	16 Worte

(a) Der Cache C1 hat einen Overhead von 36%, wobei dieser wie folgt definiert ist:

$$\text{Overhead} = 1 - \frac{\text{Nutzdaten}}{\text{Gesamtgrösse}}$$

Wie verändert sich der Overhead wenn man stattdessen Variante C2 einsetzt und das System ansonsten unverändert lässt?

Hinweis: Wenn Sie diese Aufgabe durch Nachdenken statt Rechnen bearbeiten, zählt die Lösung nur mit Begründung!

(b) Berechnen Sie die Größe der Nutzdaten und die Gesamtgröße von Variante C3. Schreiben Sie die Formel für den Overhead mit diesen Daten hin.

Hinweis: Sie müssen den Overhead nicht ausrechnen!

(c) Was sind Vor- und Nachteile von Version C3 gegenüber C1, und von C4 gegenüber C3?

Konzeptpapier: Falls der Platz unter den einzelnen Aufgaben nicht ausreicht, können Sie diese Seiten für Zwischenrechnungen nutzen. Bitte Lösung und Lösungsweg eindeutig mit der Aufgabennummer markieren!