Klausur zur Vorlesung

Grundlagen der Rechnerarchitektur (GRA)

Prof. Marco Platzner
Fachgebiet Technische Informatik
Universität Paderborn

27.09.2017

- 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**	**15**	**20**	**25**	**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) Wie nennt man die Konvention, nach der das niederwertigste Byte eines Wortes an der niedrigsten Byte-Adresse im Speicher abgelegt wird?

- ☐ Little Endian
- ☐ Big Endian
- ☐ CISC
- ☐ RISC

(b) Der CPI-Wert (*Cycles-Per-Instruction*), den ein ausgeführtes Programm auf einem Prozessor erreicht, wird beeinflusst durch:

- ☐ Programmiersprache
- ☐ Compiler
- ☐ Instruktionssatz des Prozessors
- ☐ Technologie des Prozessors

(c) Welche MIPS-Rate muss man für einen Prozessor mit einer Taktperiode von 1 ns und einem CPI-Wert von 0,5 angeben?

- ☐ Nicht bestimmbar, da zu wenige Parameter angegeben sind.
- ☐ 1000 MIPS
- ☐ 2000 MIPS
- ☐ 5000 MIPS

(d) Welche Maßnahmen reduzieren die Miss-penalty eines Caches?

- ☐ größerer Cache
- ☐ höherer Grad an Assoziativität
- ☐ Victim Cache
- ☐ breite Speicherorganisation

(e) Wie gross ist die durchschnittliche Rotationslatenz bei einer Festplatte mit 8000 Umdrehungen pro Minute?

- ☐ 7,75 μs
- ☐ 1,50 ms
- ☐ 3,75 ms
- ☐ 40,00 ms

Aufgabe 2 (Assembler) **[15 Punkte]**

(a) Gegeben ist eine unvollständige Implementierung der rekursiven Fibonacci-Funktion $f(n)$. Vervollständigen Sie die rekursive Implementierung. Beachten Sie dabei die in der Vorlesung behandelte Konvention für die Verwendung verschiedener Registertypen bei Funktionsaufrufen. Kommentieren Sie bitte Ihren Code.

```
f:                              # Sichern relevanter Register
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # Abbruchbedingung für die Rekursion
                                # $v0 = 1
                                # ist n < 3 ?
                                #     dann springe zu RET
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
        addi  $a0, $a0, -1      # ___________________________________
        jal   f                 # ___________________________________
        sw    $v0, 0($sp)       # ___________________________________
        addi  $a0, $a0, -1      # ___________________________________
        jal   f                 # ___________________________________
        lw    $t0, 0($sp)       # ___________________________________
        add   $v0, $v0, $t0     # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # Wiederherstellen relevanter Register
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
RET:                            # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
```

Ersatzlösung (inkorrekte Lösung bitte streichen):

```
f:                              # Sichern relevanter Register
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # Abbruchbedingung für die Rekursion
                                # $v0 = 1
                                # ist n < 3 ?
                                #     dann springe zu RET
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
        addi  $a0, $a0, -1      # ___________________________________
        jal   f                 # ___________________________________
        sw    $v0, 0($sp)       # ___________________________________
        addi  $a0, $a0, -1      # ___________________________________
        jal   f                 # ___________________________________
        lw    $t0, 0($sp)       # ___________________________________
        add   $v0, $v0, $t0     # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # Wiederherstellen relevanter Register
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
RET:                            # ___________________________________
                                # ___________________________________
                                # ___________________________________
                                # ___________________________________
```

(b) Wie viele Worte werden auf dem Stack bei der Berechnung von $f(n)$ mit $n = 3, 4, 5$ höchstens belegt? Vervollständigen Sie bitte die folgende Tabelle.

n	3	4	5
maximale Stackbelegung in Worten			

Ersatzlösung (inkorrekte Lösung bitte streichen):

n	3	4	5
maximale Stackbelegung in Worten			

(c) Die Fibonacci Funktion $f(n)$ kann auch durch einen iterativen Algorithmus berechnet werden. Welche Vorteile würde eine iterative Implementierung von $f(n)$ gegenüber der rekursiven Implementierung bieten?

Aufgabe 3 (Instruktionssatzarchitekturen) **[20 Punkte]**

(a) Es sollen die drei Instruktionssatzarchitekturen *Stack-*, *Speicher-Speicher-* und *Register-Register-Architektur* hinsichtlich des Speicherbedarfs und der Speicherbandbreite untersucht werden. Gegeben sei dafür folgendes Programm für eine dieser Architekturen:

```
1: add Adresse_A Adresse_I Adresse_I
2: mul Adresse_A Adresse_A Adresse_S
3: add Adresse_B Adresse_B Adresse_A
```

Für welche der Instruktionssatzarchitekturen ist dieses Programm?

☐ Stack-Architektur

☐ Speicher-Speicher-Architektur

☐ Register-Register-Architektur

(b) Was berechnet das Programm? Schreiben Sie die arithmetischen Ausdrücke hin.

1	A =
2	
3	

Tabelle 1: Programm als arithmetische Sequenz

(c) Erstellen Sie nun für jede der drei Architekturen eine Assemblercodesequenz, die das Programm implementiert, und bestimmen Sie den Speicherbedarf der Instruktionen, sowie die Menge der übertragenen Daten in Bytes. Geben Sie für die Stack- und Register-Register-Architektur außerdem die Belegung des Stacks (Abb. 1) und der Register (Abb. 3) an.

Alle auftretenden Operanden (z.B. `I`, `S`, ...) sind 32 bit groß und liegen im Speicher. Alle Ergebnisse (`A` und `B`) sind ebenfalls 32 bit groß und müssen nach Ablauf des Programms im Speicher abgelegt sein. 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`, `mul`

	Assembler-Instruktion	Speicherbedarf	Übertragene Daten
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			

Tabelle 2: Programm in Stack-Architektur

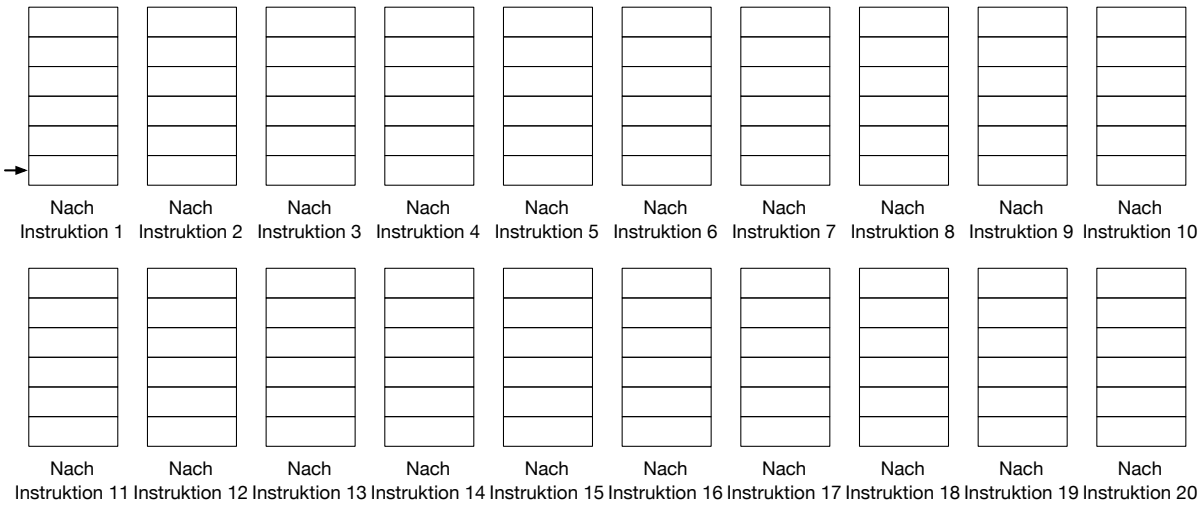


Abbildung 1: Belegung des Stacks

	Assembler-Instruktion	Speicherbedarf	Übertragene Daten
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			

Tabelle 3: Programm in Stack-Architektur - **Ersatz, ungültige Lösung streichen!**

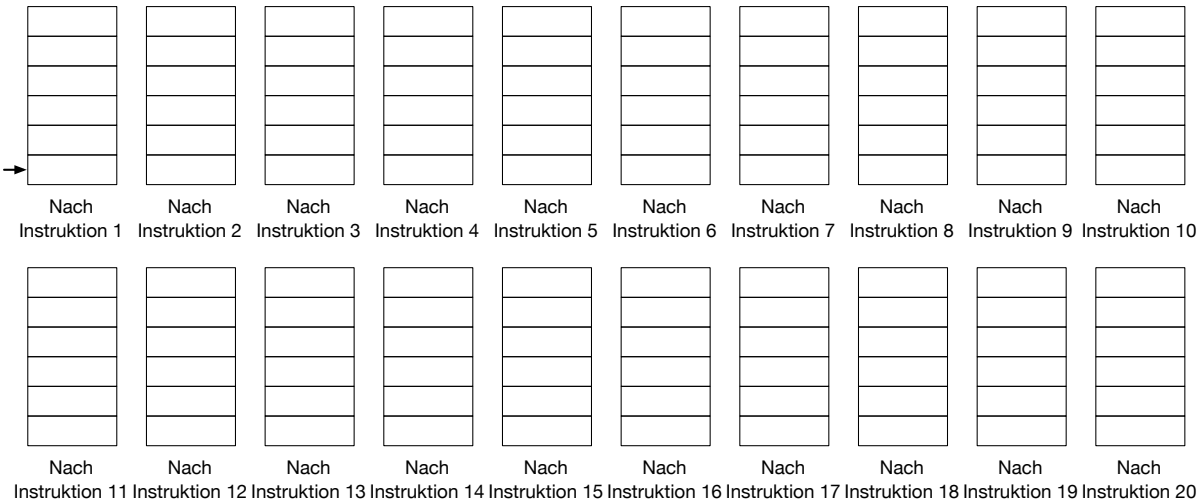


Abbildung 2: Belegung des Stacks - **Ersatz, ungültige Lösung streichen!**

(ii) **Register-Register-Architektur**. Verfügbare Instr.: `load`, `store`, `add`, `mul`

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

Tabelle 4: Programm in Register-Register-Architektur

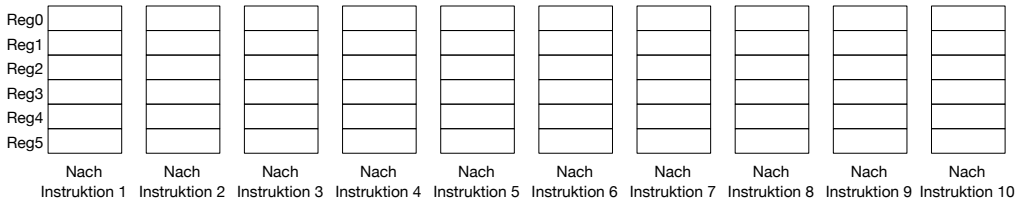


Abbildung 3: Belegung der Register

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

Tabelle 5: Programm in Register-Register-Architektur - **Ersatz, ungültige Lösung streichen!**

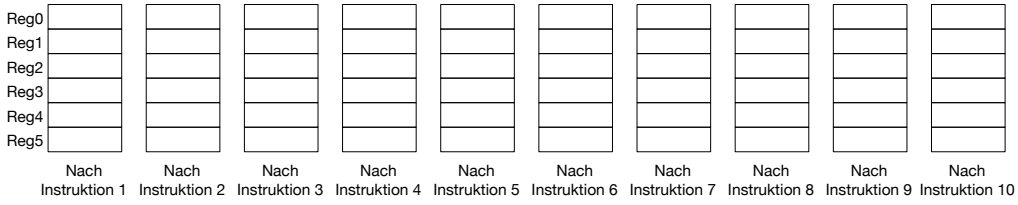


Abbildung 4: Belegung der Register - **Ersatz, ungültige Lösung streichen!**

(iii) **Speicher-Speicher-Architektur**. Verfügbare Instruktionen: `add`, `mul`.

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

Tabelle 6: Programm in Speicher-Speicher-Architektur

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

Tabelle 7: Programm in Speicher-Speicher-Architektur - **Ersatz, ungültige Lösung streichen!**

Aufgabe 4 (Pipelining) [**25 Punkte**]

(a) Ein Prozessorhersteller entwirft eine neue Prozessorgeneration mit 10 Pipelinestufen. Wie hoch ist der zu erwartende Speedup gegenüber einer Mehrzyklenimplementierung, in der jeder Befehl des Testprogramms (9991 Instruktionen) alle 10 Taktphasen durchläuft?

Speedup: _____

(b) Im folgenden soll ein Prozessor mit 5-stufiger Pipeline (IF, ID, EX, MEM, WB) betrachtet werden. Der Registerfilezugriff erfolgt im Ganztaktverfahren. Der Prozessor verwendet eine Harvard-Architektur. Die Pipeline verwendet kein Forwarding.

Folgendes Assembler Codefragment soll auf dem Prozessor ausgeführt werden:

```
0:  sub  $t2, $t1, $t3
1:  and  $t4, $t2, $t5
2:  or   $t4, $t4, $t2
3:  add  $t9, $t4, $t2
```

(i) Geben Sie die Pipelinebelegung – für das oben gezeigte Codefragment – für den Fall an, dass die Pipeline bei einem auftretenden Konflikt angehalten wird. Füllen Sie hierzu das Diagramm in Abbildung 5 aus.

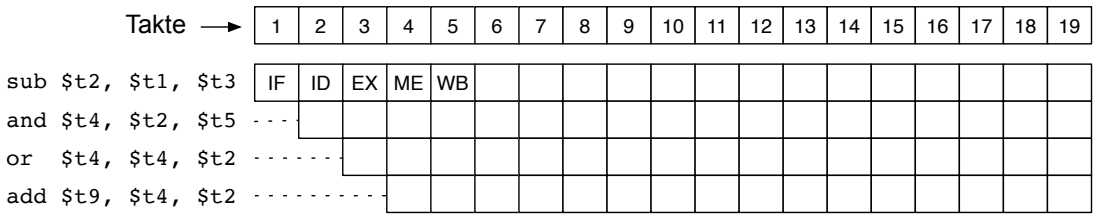


Abbildung 5: Pipelinebelegung für (i)

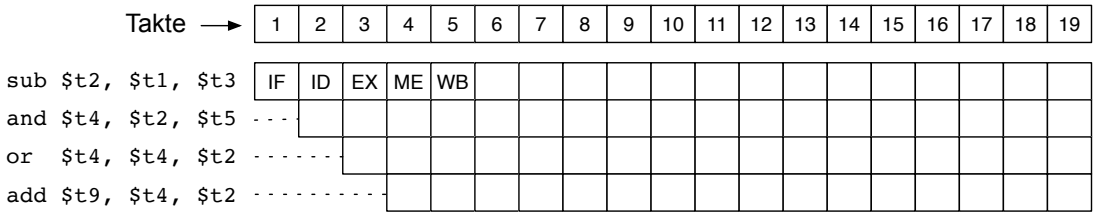


Abbildung 6: **Ersatzdiagramm:** Pipelinebelegung für (i)

(ii) Wie verändert sich die Pipelinebelegung wenn Forwarding verwendet wird? Füllen Sie hierzu das Diagramm in Abbildung 7 aus. Machen Sie das Forwarding mit Hilfe von Pfeilen deutlich.

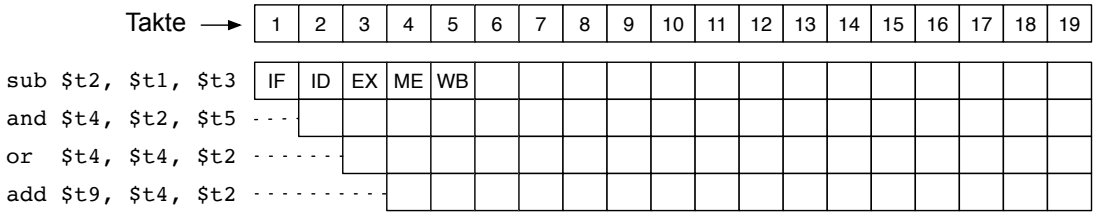


Abbildung 7: Pipelinebelegung für (ii)

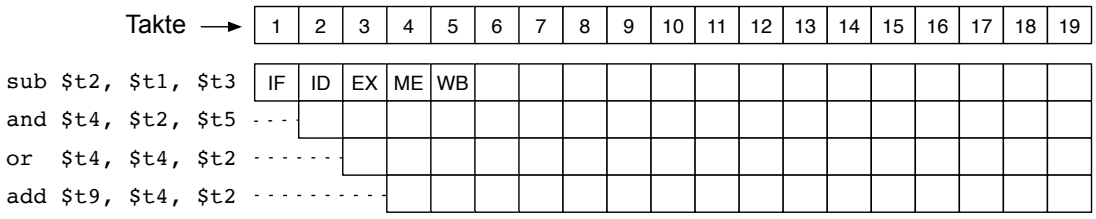


Abbildung 8: **Ersatzdiagramm:** Pipelinebelegung für (ii)

(c) Nun soll das folgende Codefragment auf dem Prozessor ohne Forwarding ausgeführt werden. Der Registerfilezugriff findet im Ganztaktverfahren statt. Die Zeile 1: liegt im Befehlsspeicher an der Adresse `0x00200010`.

```
1:  addi  $t1, $zero, 20
2:  addi  $t2, $zero, 22
3:  addi  $t3, $zero, 11
4:  sub   $t4, $t1, $t1
5:  add   $t5, $t2, $t3
```

Wie sind die Busbelegungen an den mit den Buchstaben (a-p) gekennzeichneten Stellen (siehe Abbildung 9) innerhalb der einzelnen Pipelinestufen, zu dem Zeitpunkt in dem der Befehl der Zeile 5 die IF Stufe erreicht hat?

Geben Sie die Busbelegung in Hexadezimal- oder Binärdarstellung an. Füllen Sie hierzu die Tabelle 8 aus.

Hinweis:

add:

opcode						rs					rt				
0	0	0	0	0	0										

rd					shamt					funct					
					0	0	0	0	0	1	0	0	0	0	0

sub:

opcode						rs					rt				
0	0	0	0	0	0										

rd					shamt					funct					
					0	0	0	0	0	1	0	0	0	1	0

addi:

opcode						rs					rt				
0	0	1	0	0	0										

immediate															

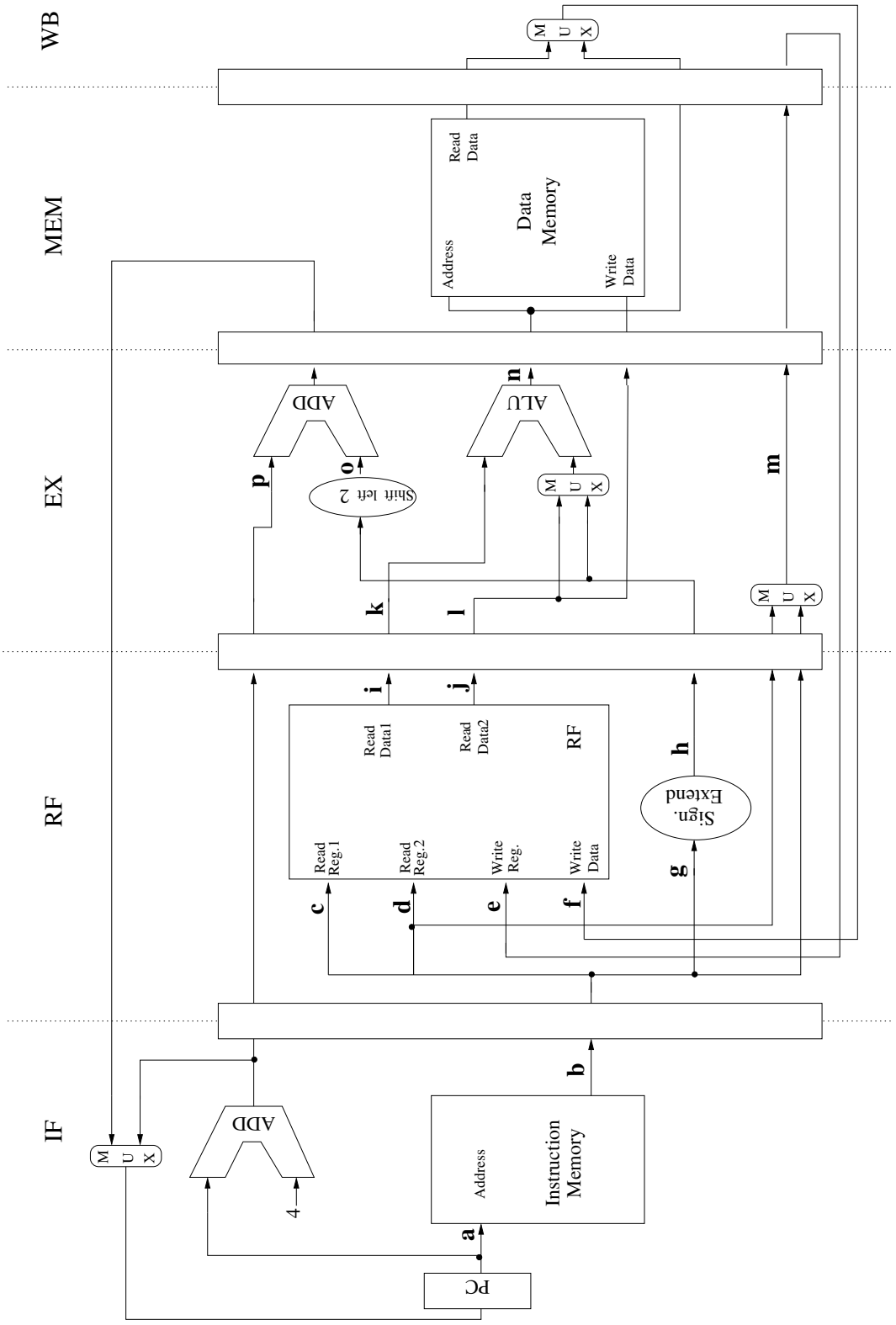


Abbildung 9: Pipeline-Datenpfad

a: _____

b:

opcode						rs					rt				
0	0	0	0	0	0										

rd					shamt					funct					
					0	0	0	0	0	1	0	0	0	0	0

c: _____

d: _____

e: _____

f: _____

g:

opcode						rs					rt				
0	0	0	0	0	0										

rd					shamt					funct					
					0	0	0	0	0	1	0	0	0	1	0

h: _____

i: _____

j: _____

k: _____

l: _____

m: _____

n: _____

o: _____

p: _____

Tabelle 8: Busbelegung für **(c)**

a: _____

b:

opcode						rs					rt				
0	0	0	0	0	0										
rd						shamt					funct				
						0	0	0	0	0	1	0	0	0	0	0

c: _____

d: _____

e: _____

f: _____

g:

opcode						rs					rt				
0	0	0	0	0	0										
rd						shamt					funct				
						0	0	0	0	0	1	0	0	0	1	0

h: _____

i: _____

j: _____

k: _____

l: _____

m: _____

n: _____

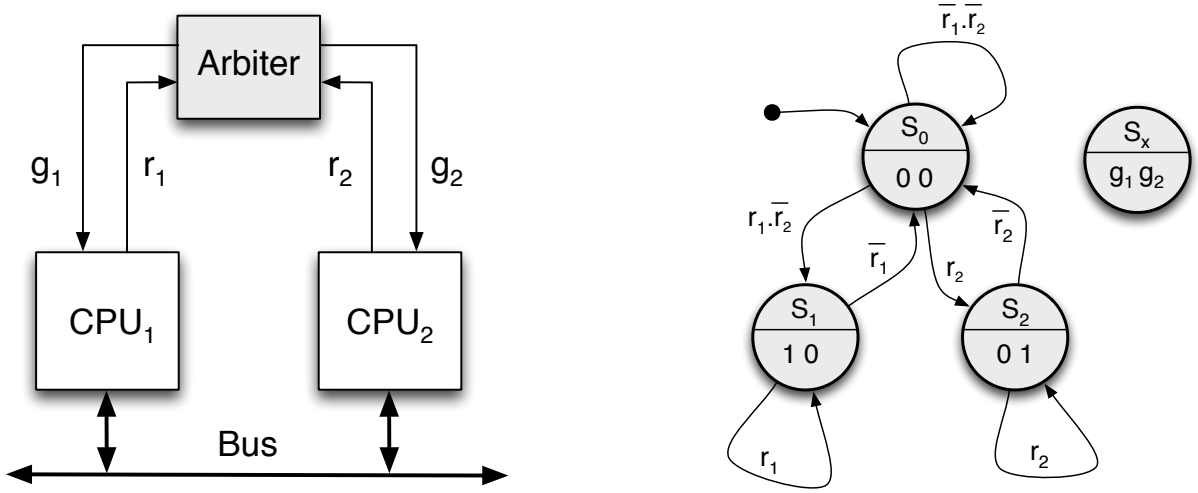
o: _____

p: _____

Tabelle 9: **Ersatztabelle:** Busbelegung für **(c)**

Aufgabe 5 (Arbitrierung) **[15 Punkte]**

Abbildung 10(a) zeigt ein System mit zwei CPUs, die an einen gemeinsamen Bus angeschlossen sind. Da beide CPUs Busmaster werden können, wird der Buszugriff durch einen zentralen Arbiter gesteuert. Die CPUs fordern über request-Signale (r_1=1 bzw. r_2=1) den Bus an. Falls einer CPU der Buszugriff gewährt wird, antwortet der Arbiter mit einem grant-Signal (g_1=1 bzw. g_2=1). Gibt eine CPU den Bus wieder frei, wird das entsprechende request Signal wieder auf 0 gesetzt, worauf der Arbiter das entsprechende grant-Signal auch wieder auf 0 setzt.



(a) Blockdiagramm (b) Zustandsautomat für den Arbiter

Abbildung 10: System mit zwei CPUs und zentralem Arbiter

(a) Abbildung 10(b) zeigt den Moore-Zustandsautomat des zentralen Arbiters. Die Ausgabesignale (g_1,g_2) sind in den Zuständen angegeben. Begründen Sie, ob dieser Arbiter den CPUs unterschiedliche Prioritäten zuweist. Wenn ja, welche der CPUs hat die höhere Prioriät?

(b) Skizzieren Sie ein Blockdiagramm für ein System mit zwei CPUs und einem gemeinsamen Bus, das *Daisy-Chain* Arbitrierung verwendet. Zeichnen Sie alle Steuersignale ein und benennen Sie diese. Die Arbitrierung soll dieselben Prioritäten für die CPUs implementieren wie die zentrale Arbitrierung in Abbildung 10(b).

(c) Entwerfen Sie einen zentralen Busarbiter, der keine festen Prioritäten vergibt, sondern "fair" ist. Ein "fairer" Arbiter weist den CPUs abwechselnd den Bus zu, falls sie ihn gleichzeitig anfordern. Geben Sie den Moore-Zustandsautomat Ihres "fairen" Arbiters an.

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!

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!