Universität Paderborn

Institut *Elektrotechnik und Informationstechnik*

Fachgebiet *Datentechnik*

Prof. Sybille Hellebrand

Klausur

Technische Informatik / Grundlagen der Rechnerarchitektur

6. Februar 2012

Punkteverteilung							
Aufgabe	1	2	3	4	5	6	Σ
maximale Punkte	11	10	20	20	20	9	90
erreichte Punkte							

Note:	

Aufkleber

Name:	
Matrikelnummer:	
Studienrichtung:	

Hinweise:
Für die Lösung der Klausuraufgaben sind ausschließlich die Aufgabenblätter zu verwenden. Lösungsangaben außerhalb der Aufgabenblätter („Schmierzettel“, etc.) werden bei der Bewertung nicht berücksichtigt!

Beschriften Sie jede Doppelseite mit Ihrer Matrikelnummer!

Mit Bleistift oder der Korrekturfarbe rot angefertigte Lösungen werden nicht bewertet!

Die Verwendung von „Tipp-Ex“ oder „Tintenkiller“ ist untersagt.

Es ist ein handgeschriebener DIN-A4 Zettel als Hilfsmittel zugelassen!

Es sind keine weiteren Hilfsmittel zugelassen!

Aufgabe 1: (Leistungsbewertung) 11 Punkte

a) Die Firma Limited Performance Corp. hat einen Prozessor mit einem idealen CPI-Wert von 1 entwickelt, der mit 500 MHz betrieben werden kann. Um schnelle Speicherzugriffe zu unterstützen, sollen zwei Cache-Ebenen realisiert werden (L1 und L2). Die Fehlgriffsrate für den L1-Cache sei 5 %. Bei einem Fehlgriff im L1-Cache wird auf den L2-Cache zugegriffen. Die zusätzliche Zeit dafür beträgt 20 ns. Von den notwendigen Zugriffen auf den L2-Cache sind 40 % Fehlgriffe und die Daten müssen dann aus dem Hauptspeicher geholt werden. Die zusätzliche Zeit dafür beträgt 200 ns. Wie groß ist der realistische CPI-Wert (einschließlich Speicherwartetakten)?

Bitte kreuzen Sie die richtige Lösung an.

(3 Punkte)

- ☐ 7
- ☐ 3,5
- ☐ 5
- ☐ 1,98
- ☐ keine Lösung ist richtig

b) Der Chef-Ingenieur von Limited Performance schlägt eine Verbesserung der Gleitkomma-Einheit vor. Bisher benötigen einfache Befehle (alle Befehle außer Gleitkommabefehlen) im Durchschnitt 2 Taktzyklen und Gleitkommabefehle 10 Taktzyklen. Durch die Verbesserung werden für Gleitkommabefehle nur noch 5 Taktzyklen benötigt, einfache Befehle brauchen dann jedoch 3 Taktzyklen. Außerdem muss die Zykluszeit von 40 ns auf 60 ns erhöht werden. Wie groß muss der Anteil der Gleitkommabefehle mindestens sein, damit sich die Verbesserung lohnt (d.h. insgesamt ein Speedup > 1 erreicht wird)?

Bitte kreuzen Sie den richtigen Wert an.

(3 Punkte)

- ☐ größer als 3/5
- ☐ größer als 4/5
- ☐ größer als 1/2
- ☐ größer als 12/25
- ☐ keine Lösung ist richtig

c) Die Konkurrenzfirma Murx Electronics hat eine neue CPU entwickelt, die mit 500 MHz getaktet werden kann. Die Befehle lassen sich grob in drei Klassen einteilen (vgl. Tabelle).

Befehlsklasse	CPI für die Befehlsklasse
A	1
B	2
C	3

Bei einem Testlauf mit einem Beispielprogramm werden 400 MIPS erreicht. In der folgenden Tabelle sind Benchmarkprogramme mit verschiedenen Verteilungen der Befehle auf die einzelnen Befehlsklassen gezeigt. Welche davon können für den Testlauf verwendet worden sein?

Kreuzen Sie bitte alle richtigen Antworten an:

(3 Punkte)

	Programm	Anzahl der Befehle in Klasse		
		A	B	C
☐	I	$5 * 10^9$	$1 * 10^9$	$1 * 10^9$
☐	II	$10 * 10^9$	$1 * 10^9$	$1 * 10^9$
☐	III	$9 * 10^9$	$3 * 10^9$	0
☐	IV	$10 * 10^9$	$2 * 10^9$	$2 * 10^9$
☐	keines der Programme wurde verwendet			

d) Murx Electronics möchte außerdem eine Prozessor-Variante mit Pipelining auf den Markt bringen. Der Befehlsablauf wird ähnlich wie beim MIPS-Prozessor in 5 Pipelinestufen aufgeteilt. Zur Abarbeitung einer Pipelinestufe wird 1 CPU-Zyklus benötigt. Die Analyse von Benchmarkprogrammen zeigt, dass die Wahrscheinlichkeit für einen Konflikt 25 % beträgt. Bei einem Konflikt müssen im Durchschnitt 2 Wartezyklen eingefügt werden. Wie hoch ist der asymptotische Speedup (unendliche viele Befehle) der Pipeline-Implementierung gegenüber einer Mehrzyklen-Implementierung?

Bitte kreuzen Sie alle richtigen Antworten an.

(2 Punkte)

- ☐ 5
- ☐ 0,25
- ☐ 4
- ☐ 2,5
- ☐ keine Lösung ist richtig

Aufgabe 2: (Datenpfad) 10 Punkte

a) Bei der Verarbeitung von Befehlen kann es im MIPS Datenpfad mit fünfstufiger Pipeline zu Datenhazards kommen. Um diese aufzulösen, können sowohl *Stalls* (Leerzyklen) in die Pipeline eingefügt werden als auch eine *Forwarding*-Einheit integriert werden.

Welche weitere Möglichkeit besteht meistens auch noch?

(1 Punkt)

b) Bei der Verarbeitung folgender Speicher-zu-Speicher Kopiersequenz tritt ein Datenhazard auf.

```
lw  $2, 100($5)
sw  $2, 200($6)
```

Vervollständigen Sie die zweite Zeile der Pipelinebelegung (siehe Abbildung 1) und fügen Sie die minimale Anzahl an *Stalls* (STA) in die Pipeline ein, um den Hazard aufzulösen. Begründen Sie kurz Ihre Antwort.

(3 Punkte)

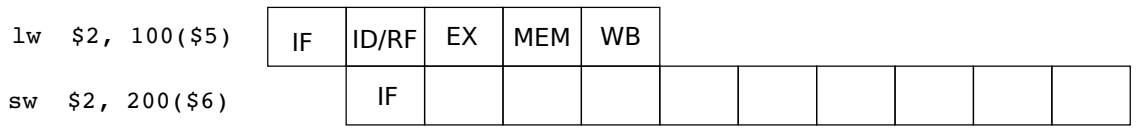

Abbildung 1: Pipelinebelegung

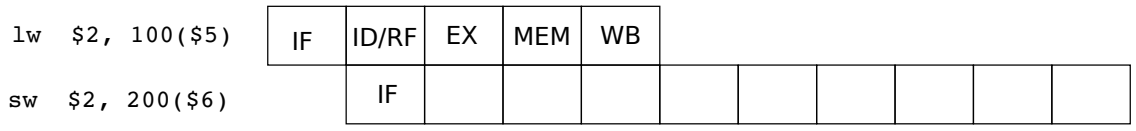

Abbildung 2: Ersatzgrafik, **ungültige Lösung streichen!**

c) Der Datenhazard, der durch die Speicher-zu-Speicher Kopiersequenz aus Aufgabenteil **b)** entstanden ist, soll nun durch eine *Forwarding*-Einheit aufgelöst werden. Modifizieren Sie den Datenpfad in Abbildung 3 entsprechend.

(Ersatzabbildung auf Seite 9, **ungültige Lösung streichen!**)

(3 Punkte)

d) Geben Sie eine mögliche *lw-sw* Instruktionssequenz an, bei der trotz der eingefügten *Forwarding*-Einheit *Stalls* notwendig sind. Begründen Sie Ihre Antwort.

(3 Punkte)

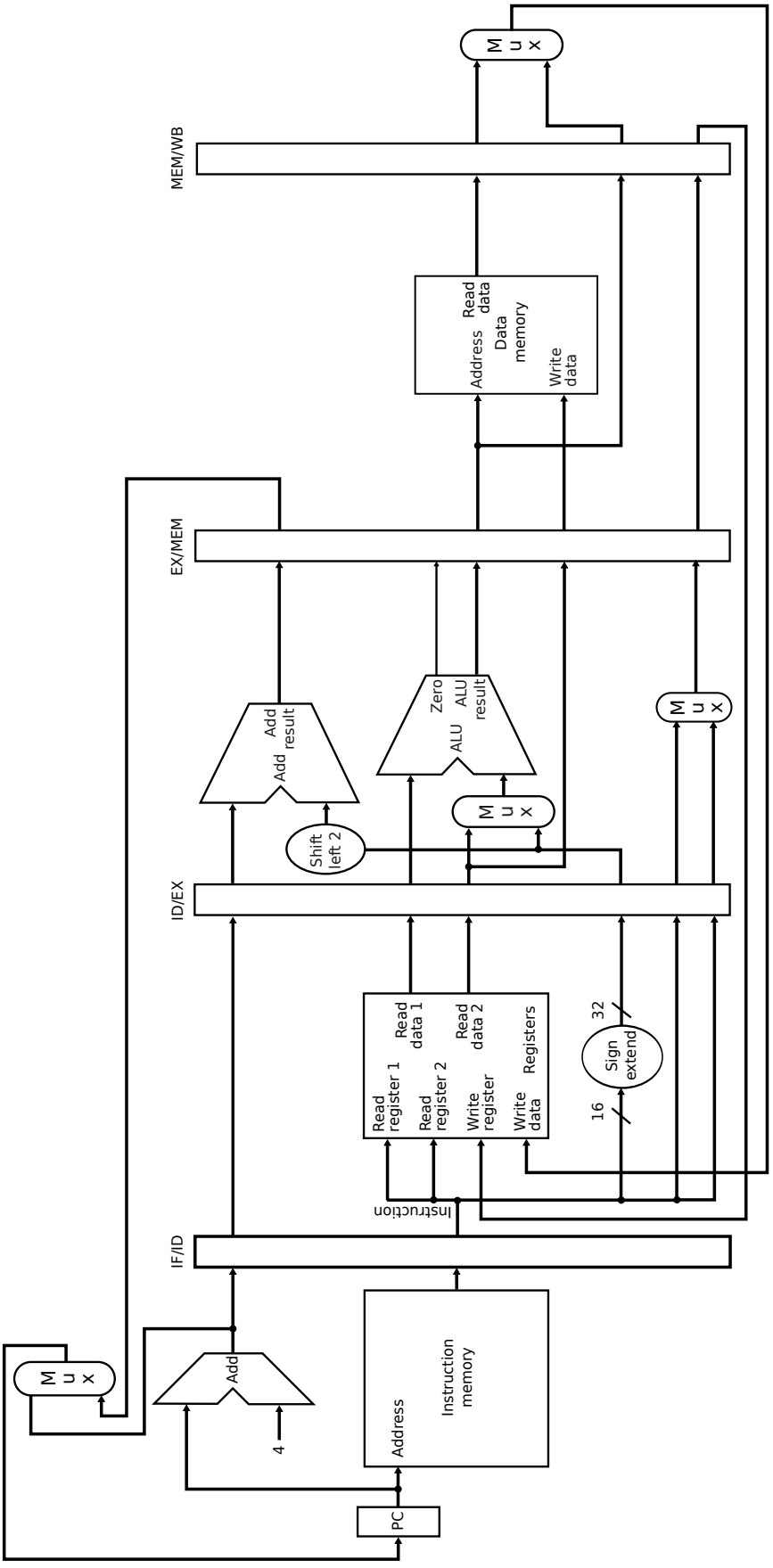

Abbildung 3: Fünfstufige Pipeline des MIPS Prozessors

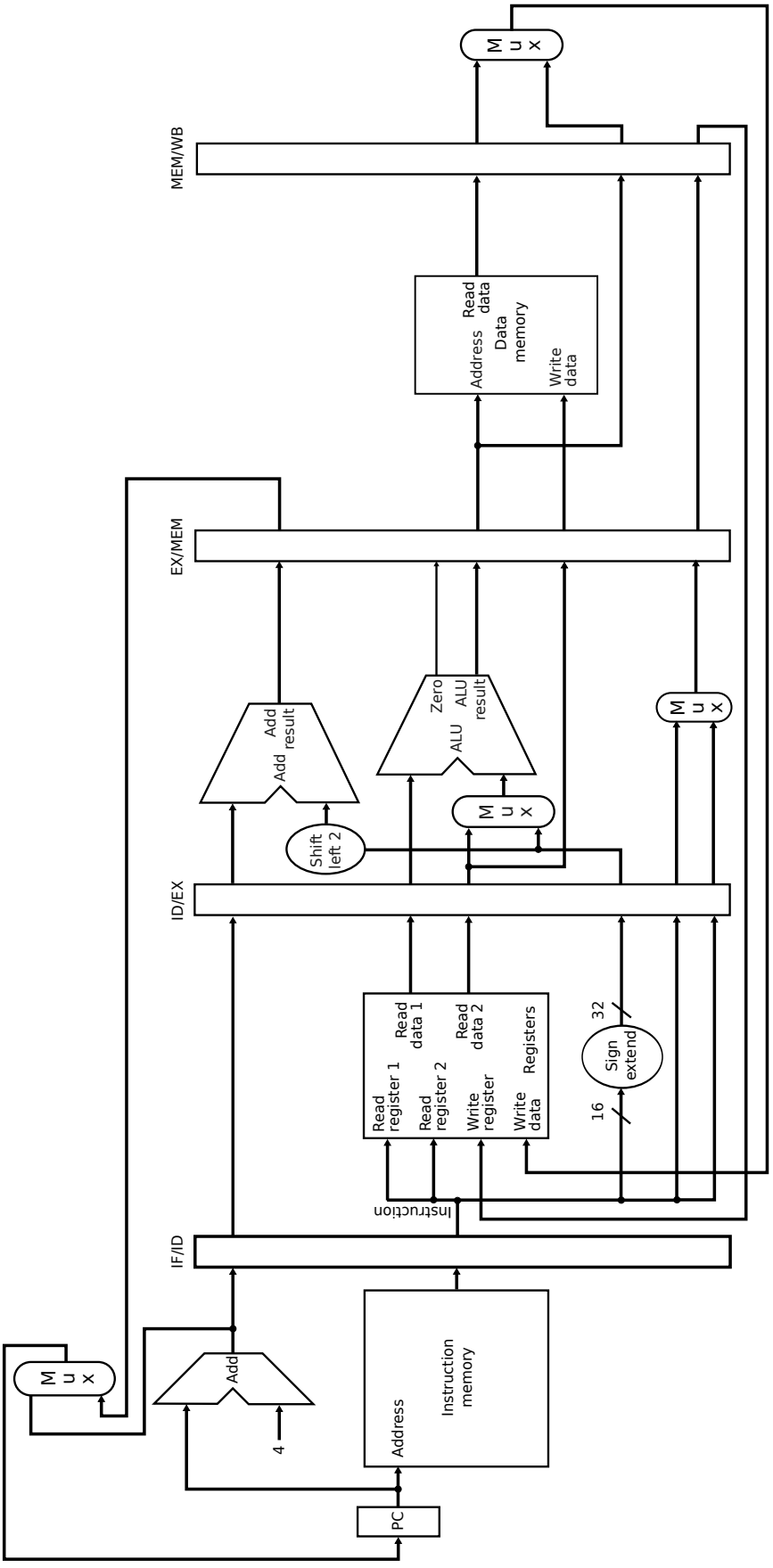

Abbildung 4: Ersatzgrafik, **ungültige Lösung streichen!**

Aufgabe 3: (Pipelining) 20 Punkte

Gegeben sei ein MIPS Prozessor mit folgenden Eigenschaften:

- 5-stufige Pipeline (IF, ID, EX, MEM, WB)
- Getrennter Speicher für Daten und Programm (**Harvard-Architektur**)
- Die Pipeline nutzt alle Möglichkeiten für **Forwarding** aus
- Static Branch Prediction (**branch not taken**, Entscheidung in der MEM-Phase)

Weiterhin sei folgendes Assemblerprogramm gegeben:

```
         .data                    # Start der Datensektion
numEle:  .word 5                  # Anzahl der Elemente in der Liste
liste:   .word 3,6,7,5,14         # Liste der Elemente

         .text                    # Start der Textsektion
main:    la   $a0, liste          # Adresse in $a0 laden
         addi $t0, $zero, 0       # $t0 mit 0 initialisieren
         lw   $s0, numEle         # Anzahl der Elemente nach $s0
test:    add  $a1, $a0, $t0       # Adresse des aktuellen Elements
         lw   $t1, 0($a1)         # Element nach $t1 laden
         and  $t2, $t1, 1         # $t1 mit 1 UND-Verknüpfen
         beq  $t2, $zero, gerade  # Wenn $t2 = 0 ist, zu gerade springen
         sll  $t1, $t1, 1         # $t1 nach links schieben
         addi $t1, $t1, 1         # Und 1 addieren
         sw   $t1, 0($a1)         # Berechneten Wert in Liste speichern
gerade:  addi $t0, $t0, 4         # $t0+4 für nächste Adresse
         addi $s0, $s0, -1        # Schleifenzähler dekrementieren
         bne  $s0, $zero, test    # Schleife wiederholen bis $s0=0

         addi $v0, $zero, 10      # Systemcall-Code für Exit laden
         syscall                  # Programm beenden
```

Die Entscheidung über einen bedingten Sprung wird bei dieser Architektur in der MEM-Phase entschieden, so dass die Static Branch Prediction bis zu 3 Befehle im Voraus in die Pipeline laden kann.
Ermitteln Sie in den folgenden Teilaufgaben die Stalls in der MIPS Pipeline für die angegebenen Codeseqenzen.

Erster Durchlauf der Schleife

Das Programm wird zum ersten Mal aufgerufen. Damit ergibt sich folgende Registerbelegung:

`$s0 = 5 ; $t0 = 0 ; $a0 = liste (Die Adresse der Daten im Speicher)`

Nehmen Sie weiter an, dass die Befehle vor der unten angegebenen Sequenz keine Stalls der Pipeline erzeugen.

a) Markieren Sie die Befehle, die in der gegebenen Konfiguration ausgeführt werden (Machen Sie ein × in den Kästchen vor den Befehlen in der folgenden Tabelle, wenn ein Befehl ausgeführt wird). (1 Punkte)

b) Füllen Sie nun die Pipeline Tabelle aus. Falls Forwarding verwendet wird, machen Sie dies kenntlich. (Ersatz Pipeline Tabelle auf Seite 13, **ungültige Lösung streichen!**) (8 Punkte)

Beachten Sie dabei die **Static Branch Prediction** und das die Entscheidung über einen bedingten Sprung erst in der **MEM-Phase** getroffen werden kann!

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t: ☒	`add $a1,$a0,$t0`	IF	ID	EX	M	WB										
☐	`lw $t1,0($a1)`															
☐	`and $t2, $t1, 1`															
☐	`beq $t2,$zero,g`															
☐	`sll $t1,$t1, 1`															
☐	`addi $t1,$t1,1`															
☐	`sw $t1,0($a1)`															
g: ☐	`addi $t0,$t0,4`															
☐	`addi $s0,$s0,-1`															

c) Wie viele Takte benötigt die Befehlsequenz? (0,5 Punkte)

Takte der gegebenen Befehlssequenz = _____

d) Wie viele Takte würde die ideale Befehlssequenz benötigen (keine Stalls, sofortige Entscheidung des bedingten Sprunges)? (0,5 Punkte)

Takte der idealen Befehlssequenz = _____

Zweiter Durchlauf der Schleife

Das Programm befindet sich nun im zweiten Durchlauf der Schleife. Damit ergibt sich folgende Registerbelegung:

`$s0 = 4 ; $t0 = 4 ; $a0 = liste (Die Adresse der Daten im Speicher)`

Nehmen Sie weiter an, dass die Befehle vor der unten angegebenen Sequenz keine Stalls der Pipeline erzeugen.

a) Markieren Sie die Befehle, die in der gegebenen Konfiguration ausgeführt werden (Machen Sie ein × in den Kästchen vor den Befehlen in der folgenden Tabelle, wenn ein Befehl ausgeführt wird). (1 Punkte)

b) Füllen Sie nun die Pipeline Tabelle aus. Falls Forwarding verwendet wird, machen Sie dies kenntlich. (Ersatz Pipeline Tabelle auf Seite 13, **ungültige Lösung streichen!**) (8 Punkte)

Beachten Sie dabei die **Static Branch Prediction** und das die Entscheidung über einen bedingten Sprung erst in der **MEM-Phase** getroffen werden kann!

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t:	☒ `add $a1,$a0,$t0`	IF	ID	EX	M	WB										
	☐ `lw $t1,0($a1)`															
	☐ `and $t2, $t1, 1`															
	☐ `beq $t2,$zero,g`															
	☐ `sll $t1,$t1, 1`															
	☐ `addi $t1,$t1,1`															
	☐ `sw $t1,0($a1)`															
g:	☐ `addi $t0,$t0,4`															
	☐ `addi $s0,$s0,-1`															

c) Wie viele Takte benötigt die Befehlsequenz? (0,5 Punkte)

Takte der gegebenen Befehlssequenz = _____

d) Wie viele Takte würde die ideale Befehlssequenz benötigen (keine Stalls, sofortige Entscheidung des bedingten Sprunges)? (0,5 Punkte)

Takte der idealen Befehlssequenz = _____

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t: ☒	`add $a1,$a0,$t0`	IF	ID	EX	M	WB										
☐	`lw $t1,0($a1)`															
☐	`and $t2, $t1, 1`															
☐	`beq $t2,$zero,g`															
☐	`sll $t1,$t1, 1`															
☐	`addi $t1,$t1,1`															
☐	`sw $t1,0($a1)`															
g: ☐	`addi $t0,$t0,4`															
☐	`addi $s0,$s0,-1`															

Abbildung 5: Ersatz Pipeline Tabelle erster Durchlauf, **ungültige Lösung streichen!**

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t: ☒	`add $a1,$a0,$t0`	IF	ID	EX	M	WB										
☐	`lw $t1,0($a1)`															
☐	`and $t2, $t1, 1`															
☐	`beq $t2,$zero,g`															
☐	`sll $t1,$t1, 1`															
☐	`addi $t1,$t1,1`															
☐	`sw $t1,0($a1)`															
g: ☐	`addi $t0,$t0,4`															
☐	`addi $s0,$s0,-1`															

Abbildung 6: Ersatz Pipeline Tabelle zweiter Durchlauf, **ungültige Lösung streichen!**

Aufgabe 4: (Assembler) 20 Punkte

Gegeben sei folgende Assembler-Funktion **funct**. Die Funktion bekommt als Parameter zwei positive Zahlen in **$a0** und **$a1** übergeben und schreibt das Ergebnis in das Register **$v0**.

```
1          .text
2  main:   addi    $a0,    $zero,  3
3          addi    $a1,    $zero,  4
4          j       funct0
5
6          addi    $v0,    $zero,  10      # Programm beenden
7          syscall                         # Systemaufruf
8
9  funct0: addi    $v0,    $zero,  0
10 funct1: addi    $sp,    $sp,    8
11         sw      $ra,    4($sp)
12         beq     $a1,    $zero,  exit
13
14         sll     $a0,    $a0,    1       # sll = shift left logical
15         add     $v0,    $v0,    $a0
16
17         addi    $t0,    $v0,    0       # $v0 sichern
18         sw      $a0,    0($sp)          # $a0 sichern
19
20         addi    $a0,    $v0,    0
21         addi    $v0,    $zero,  1       # Code für print integer
22         syscall                         # Systemaufruf
23         addi    $a0,    $zero,  ' '
24         addi    $v0,    $zero,  11      # Code für print character
25         syscall                         # Systemaufruf
26
27         addi    $v0,    $t0,    0       # $v0 laden
28         lw      $a0,    $sp             # $a0 laden
29
30         addi    $a1,    $a1,    -1
31         jal     funct1
32
33 exit:   lw      $ra,    0($sp)
34         addi    $sp,    $sp,    -8
35         j       $ra
```

a) Das oben abgebildete Program ist so nicht lauffähig und muss korrigiert werden. Finden Sie die 6 zu ändernden Zeilen und schreiben sie diese um (geänderte korrekte Zeilen geben Minuspunkte).

(10 Punkte)

Lösung:	Ersatz (**ungültige Lösung streichen!**):
Zeile_____:________________________	Zeile_____:________________________
Zeile_____:________________________	Zeile_____:________________________
Zeile_____:________________________	Zeile_____:________________________
Zeile_____:________________________	Zeile_____:________________________
Zeile_____:________________________	Zeile_____:________________________
Zeile_____:________________________	Zeile_____:________________________

b) Was macht das Programm in den Zeilen **10 - 11** und weswegen sind diese Schritte notwendig?

(2 Punkte)

____

____

c) Die Funktion gibt während der Berechnung Zwischenergebnisse aus. Wie sieht die komplette Ausgabe nach Beendigung des Programms aus?

(5 Punkte)

____

____

d) Welche Funktion wird in **funct** berechnet? Geben Sie den Inhalt des Registers **$v0** nach Beendigung der Funktion in Abhängigkeit von **$a0** und **$a1** an.

(3 Punkte)

Aufgabe 5: (Speicherverwaltung) 20 Punkte

Gegeben sei ein Rechnersystem, welches zur effizienteren Nutzung mit einem *virtuellen Speicher* arbeitet. Zur Speicherverwaltung wird eine **MMU** (**M**emory **M**anagement **U**nit) eingesetzt, die Segmentierung mit Seitenverwaltung einsetzt. Das Prinzip der MMU ist in Abbildung 7 skizziert. Der Speicher hat eine Byteadressierung.

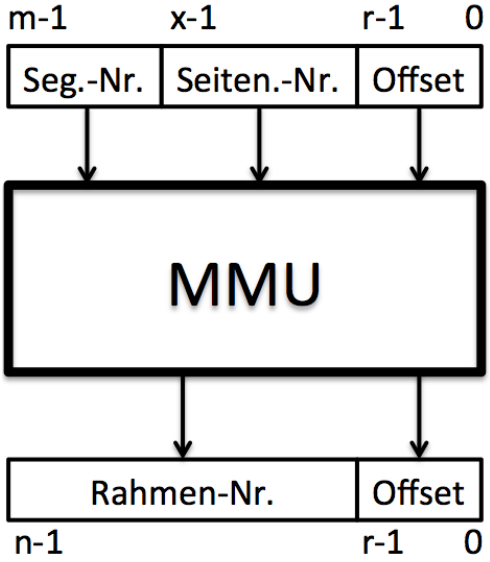

Abbildung 7: Adressberechnung

a) Wie groß ist der maximal verfügbare virtuelle Adressraum des Systems (in Byte)?
(1 Punkt)

---

b) Wie viele virtuelle Seiten stehen dem System maximal zur Verfügung?
(1 Punkt)

---

c) Wie groß ist der maximal zur Verfügung stehende physikalische Speicher des Systems (in Byte)?
(1 Punkt)

Gehen Sie im folgenden von den Werten $m = 16$, $x = 13$ und $r = 10$ aus.

d) Gegeben seien die folgenden virtuellen m-Bit Speicheradressen $0xAC35$ und $0xE4AF$. Bestimmen Sie anhand der folgenden Segment- und Seitentabellen die dazugehörigen physikalischen Adressen. Tragen Sie hierzu die Segmentnummer, Seitennummer und die physikalische Adresse in Tabelle 1 ein. Benutzen Sie die hexadezimale Schreibweise. (Ersatztabelle auf Seite 20, **ungültige Lösung streichen!**)

(7 Punkte)

Segmenttabelle		
Nr.	V	Zeiger
0	0	1
1	0	2
2	0	2
3	0	3
4	1	3
5	1	2
6	0	2
7	1	3

Seitentabelle 1		
Nr.	V	Zeiger
0	1	3
1	1	2
2	1	2
3	0	2
4	0	3
5	0	1
6	1	1
7	0	1

Seitentabelle 2		
Nr.	V	Zeiger
0	1	3
1	0	2
2	0	1
3	1	3
4	0	3
5	0	3
6	0	1
7	1	2

Seitentabelle 3		
Nr.	V	Zeiger
0	0	1
1	0	1
2	1	1
3	1	3
4	0	3
5	1	1
6	0	2
7	1	3

Virt. Adresse	Segmentnr.	Seitennr.	Physikalische Adresse
$0xAC35$			
$0xE4AF$			

Tabelle 1: Übersetzungstabelle

e) Der Prozess *gr.a* greift auf eine Seitentabelle mit 5 virtuellen Seiten zu und habe 2 Rahmen **A** und **B** im Hauptspeicher zur Verfügung. Während der Ausführung treten nun mehrere Speicherzugriffe auf, wobei *n*-*r* einen Lesezugriff und *n*-*w* einen Schreibzugriff auf die *n*-te virtuelle Seite bedeuten. Als Ersetzungsstrategie wird LRU (**L**east **R**ecently **U**sed) eingesetzt.

Zusätzlich wird in der Seitentabelle ein **Dirty Bit** hinzugefügt, welches auf 1 gesetzt wird, wenn ein Wort auf eine Seite geschrieben wurde, und so lange 1 bleibt, bis diese Seite zurückgeschrieben wird, sonst 0.

Geben Sie in der Tabelle 2 die Speicherbelegung des Hauptspeichers an. Bestimmen Sie für jeden Befehl

- das **Dirty-Bit** von n (zu Beginn des Prozesses mit 0 initalisiert),
- ordnen Sie die Seiten den jeweiligen Hauptspeicheradressen zu (**A** oder **B**),
- geben Sie an, ob ein Seitenfehler (Spalte **Fehler**) vorhanden ist und
- welche Seite zurückgeschrieben werden muss (Spalte **WB**).

Sie können davon ausgehen, dass die Daten erst beim Auslagern in den Sekundärspeicher zurückgeschrieben werden und dieser beliebig groß ist. Werden keine Daten zurückgeschrieben oder befinden sich keine Daten im Hauptspeicher, markieren Sie die Zelle mit "-". Der Befehl $1-r$ wurde bereits ausgeführt.
(Ersatztabelle auf Seite 20, **ungültige Lösung streichen!**)

(10 Punkte)

Befehl	Dirty-Bit	Speicheradressen		Fehler	WB
		A	B		
$1-r$	0	1	-	ja	-
$1-w$					
$2-r$					
$3-w$					
$1-r$					
$3-r$					
$5-w$					
$4-r$					

Tabelle 2: Speicherbelegung Hauptspeicher

Virt. Adresse	Segmentnr.	Seitennr.	Physikalische Adresse
$0xAC35$			
$0xE4AF$			

Tabelle 3: Ersatz Übersetzungstabelle, **ungültige Lösung streichen!**

Befehl	Dirty-Bit	Speicheradressen		Fehler	WB
		A	B		
$1-r$	0	1	-	ja	-
$1-w$					
$2-r$					
$3-w$					
$1-r$					
$3-r$					
$5-w$					
$4-r$					

Tabelle 4: Ersatz Speicherbelegung Hauptspeicher, **ungültige Lösung streichen!**

Aufgabe 6: (Maschinenstrukturen) 9 Punkte

Auf einem Rechnersystem mit **Akkumulator-Architektur** soll die Berechnung

$$c = k \cdot a + b$$

ausgeführt werden. Nach Beendigung des Programms soll sich c im Speicher befinden. Gehen Sie davon aus, dass die Variablen a und b zu Beginn im Speicher liegen und k konstant ist. Die zur Verfügung stehenden Assemblerinstruktionen sind {`add, load, store, sub`}.

a) Wie lautet der Programmcode der Berechnung und wie viele Befehle werden benötigt (in Abhängigkeit von k)?

(4 Punkte)

b) Die Berechnung kann alternativ auch auf einem Rechnersystem mit **Stack-Architektur** ausgeführt werden. Hierzu stehen die Assemlerinstruktionen {`add, pop, push, sub`} zur Verfügung. Welche Architektur ist für die gegebene Berechnung bezüglich Anzahl der benötigten Befehle besser geeignet? Begründen Sie Ihre Antwort.

(5 Punkte)