Klausur zur Vorlesung

Rechnerarchitektur

Prof. Christian Plessl
Fachgebiet Hochleistungs-IT-Systeme
Universität Paderborn

25.09.2019

- 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	**14**	**20**	**20**	**16**	**20**	90
Erreicht						

Aufgabe 1 (Multiple Choice) [14 Punkte]

Bei den folgenden Fragen können eine oder mehrere Antworten richtig sein. Kreuzen Sie die richtigen Antworten deutlich an. Für jede vollständig korrekt beantwortete Frage gibt es 2 Punkte, ansonsten 0 Punkte.

(a) Ein Computer führt ein Programm auf einer CPU mit CPI=1 aus. Welcher Beschleunigungsfaktor (t_{alt} / t_{neu}) kann durch die Verwendung einer GPU (Graphics Processing Unit) maximal erreicht werden, wenn 25% der Instruktionen auf die GPU ausgelagert werden können?

(2 Punkte)

- ☐ 4/3
- ☐ 3/4
- ☐ 1/4
- ☐ 4

(b) Welche Aussagen stimmen? Das Sprungziel des J-Befehls des MIPS Prozessors ...

(2 Punkte)

- ☐ kann eine beliebige 32-bit Instruktions-Adresse sein.
- ☐ wird implizit aus Register `$ra` gelesen.
- ☐ wird aus einem beliebigen Register (`$0-$31`) geladen.
- ☐ wird aus den höchstwertigen Bits des Programmzählers gefolgt von einer 26-bit Konstanten berechnet.

(c) Welche Instruktionen/Instruktionssequenzen laden immer den Wert 0 in Register `$1`?

(2 Punkte)

- ☐ `add $1, $1, $0`
- ☐ `add $1, $0, $0`
- ☐ `xor $1, $1, $1`
- ☐ `lui $1, 0`

(d) Welche Massnahmen reduzieren die miss-penalty eines Caches?

(2 Punkte)

- ☐ größere Blöcke
- ☐ kleine Blöcke
- ☐ höherer Grad der Assoziativität
- ☐ schmalerer Speicherbus

(e) Welche Methoden können zur Auflösung von Control Hazards eingesetzt werden?

(2 Punkte)

- ☐ forwarding
- ☐ Anhalten der Pipeline
- ☐ virtueller Speicher
- ☐ Sprungvorhersage

(f) Welche Vorteile bietet k-stufiges Pipelining im Gegensatz zu einer Mehrzyklenimplementierung mit gleich vielen Stufen?

(2 Punkte)

- ☐ Es verwendet die von Neumann statt der Harvard-Architektur.
- ☐ Die Taktrate wird um den Faktor k reduziert.
- ☐ Der Instruktionsdurchsatz ist höher.
- ☐ Die Abarbeitsungszeit der einzelnen Instruktionen verringert sich.

(g) Ersetzt man Punkt-zu-Punkt Verbindungen zwischen Komponenten durch ein Bus-System, so...

(2 Punkte)

- ☐ müssen alle Komponenten mit einem gemeinsamen Takt arbeiten.
- ☐ können neue Komponenten leichter hinzugefügt werden.
- ☐ erhöht sich die Ausfallsicherheit des Gesamtsystems.
- ☐ sinkt der Hardwareaufwand und damit die Kosten.

Aufgabe 2 (Assembler-Programming) **[20 Punkte]**

(a) Zum Auslagern von Registern wird ein Teil des Speichers einer MIPS-Architektur verwendet. Dieser Teil wird Stack (Kellerspeicher) genannt. Kreuzen Sie die zutreffenden Antworten an.

(i) In welche Richtung wird laut der aus der Vorlesung bekannten MIPS-Konvention der Stack vergrößert, um Register auszulagern?

(1 Punkt)

- ☐ Von niederwertigen Adressen zu höherwertigen Adressen.
- ☐ Von höherwertigen Adressen zu niederwertigen Adressen.

(ii) Welches Register zeigt immer auf den Anfang des Stacks?

(1 Punkt)

- ☐ `$sp`
- ☐ `$gp`
- ☐ `$fp`
- ☐ `$ra`

(b) MIPS unterstützt den Prozeduraufruf durch eine Sprunginstruktion, welche die Rücksprungadresse sichert und zur Adresse der Prozedur springt. Kreuzen Sie die zutreffenden Antworten an.

(i) Welche Sprunginstruktion wird für den Prozeduraufruf zur Verfügung gestellt?

(1 Punkt)

- ☐ `jal`
- ☐ `jr`
- ☐ `j`

(ii) Welches Register wird zur Sicherung der Rücksprungadresse genutzt?

(1 Punkt)

- ☐ `$sp`
- ☐ `$gp`
- ☐ `$fp`
- ☐ `$ra`

(c) Bei einem geschachtelten Aufruf von Prozeduren kann es zu potentiellen Konflikten bei der Nutzung der Register kommen.

Welche Register müssen laut der aus der Vorlesung bekannten MIPS-Konvention von der **aufrufenden** und **aufgerufenen** Prozedur gesichert werden, um diese Konflikte zu verhindern?

(4 Punkte)

Aufrufende Prozedur:

Aufgerufene Prozedur:

(d) Listing 3 zeigt eine lückenhafte Implementierung des rekursiven *Quick-Sort* Algorithmus. Füllen Sie die Lücken im Listing 3 entsprechend den aus der Vorlesung bekannten MIPS-Konventionen aus. (12 Punkte)

Listing 1 und Listing 2 zeigen den Pseudocode des Algorithmus. Außerdem ist in Abbildung 1 eine Illustration der Funktion `partition` gegeben.

```
void quicksort(int *A, int p, int r){
    if( p < r ){
        int q = partition(A, p, r);
        quicksort(A, p, q-1);
        quicksort(A, q+1, r);
    }
}
```

Listing 1: Pseudocode `quicksort`

```
int partition(int *A, int p, int r){
    int x = A[r];
    int i = p-1;
    for(int j = p; j < r; j++){
        if(A[j] <= x) {
            i++;
            swap(A, i, j);
        }
    }
    swap(A, i+1, r);
    return i+1;
}
```

Listing 2: Pseudocode `partition`

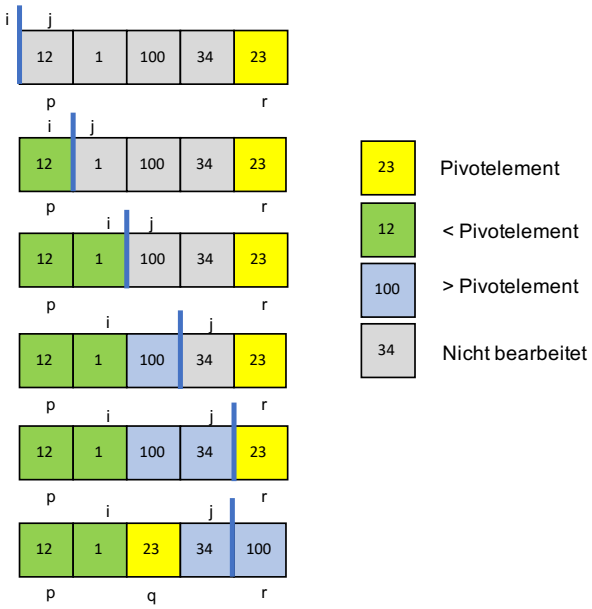


Abbildung 1: Beispieldurchlauf der Funktion `partition(A, p, r)`.

```
        .data
values: .word 12, 1, 100, 34, 23
        .text
main:   la      $a0, values         # Lade Startadresse des Arrays
        la      $a1, values         # Adresse (p) des ersten Elements
        la      $a2, values+16      # Adresse (r) des letzten Elements
        .....   qsort               # Rufe qsort auf
        addi    $v0, $zero, 10      # load system call code for exit
        syscall                     # exit the program
qsort:  addi    ..............      # Schaffe Platz für 4 Worte
        sw      ..............      # Speichere Rücksprungadresse
        sw      ..............      # Speichere Startadresse
        sw      ..............      # Speichere p
        sw      ..............      # Speichere r
        bge     $a1, $a2, done      # p >= r ? ,dann springe zu done
        .....   part                # Rufe part auf
        add     $s0, $zero, $v0     # Pos q speichern
        lw      ..............      # Lade Startadresse des Arrays
        lw      ..............      # Lade p
        addi    $a2, $s0, -4        # Neue letzte Adresse r=q-4
        .....   qsort               # Rufe qsort auf
        lw      ..............      # Lade Startadresse des Arrays
        addi    $a1, $s0, 4         # Neue erste Adresse p=q+4
        lw      ..............      # Lade letzte Adresse r
        .....   qsort               # Rufe qsort auf
done:   lw      ..............      # Hole Rücksprungadresse
        lw      ..............      # Hole Startadresse des Arrays
        lw      ..............      # Hole erste Adresse p
        lw      ..............      # Hole letzte Adresse r
        addi    ..............      # 4 Worte freigeben
        jr      ..............      # Rücksprung zum Aufruf
part:   addi    ..............      # Schaffe Platz für 1 Wort
        sw      ..............      # Speichere Rücksprungadresse
        lw      $t0, ($a2)          # Lade Pivot-Element
        addi    $t1, $a1, -4        # i = p-4
        addi    $t2, $zero, $a1     # j = p
        addi    $t3, $a2, -4        # Obere Schranke r-4 berechnen
loop:   bgt     $t2, $t3, loopend   # j > r-4 dann stop
        lw      $t4, ($t2)          # Lade elem A[j]
        addi    $t2, $t2, 4         # j = j + 4
        bgt     $t4, $t0, loop      # Springe wenn A[j] > x
        addi    $t1, $t1, 4         # i = i + 4
        lw      $t5, ($t1)          # Lade A[i]
        sw      $t4, ($t1)          # Speichere A[j] an Platz von A[i]
        sw      $t5, -4($t2)        # Speichere A[i] an Platz von A[j]
        j       loop                # Springe zu loop
loopend:addi    $t1, $t1, 4         # i = i + 4
        lw      $t5, ($t1)          # Hole A[i]
        sw      $t5, ($a2)          # Speichere A[i] an Platz von A[r]
        sw      $t0, ($t1)          # Speichere A[r] an Platz von A[i]
        addi    $v0, $t1, $zero     # Setzte Rückgabewert
        lw      ..............      # Hole Rücksprungadresse
        addi    ..............      # 1 Wort freigeben
        jr      ..............      # Rücksprung
```

Listing 3: MIPS-Assembler

```
        .data
values: .word 12, 1, 100, 34, 23
        .text
main:   la      $a0, values         # Lade Startadresse des Arrays
        la      $a1, values         # Adresse (p) des ersten Elements
        la      $a2, values+16      # Adresse (r) des letzten Elements
        .....   qsort               # Rufe qsort auf
        addi    $v0, $zero, 10      # load system call code for exit
        syscall                     # exit the program
qsort:  addi    ..............      # Schaffe Platz für 4 Worte
        sw      ..............      # Speichere Rücksprungadresse
        sw      ..............      # Speichere Startadresse
        sw      ..............      # Speichere p
        sw      ..............      # Speichere r
        bge     $a1, $a2, done      # p >= r ? ,dann springe zu done
        .....   part                # Rufe part auf
        add     $s0, $zero, $v0     # Pos q speichern
        lw      ..............      # Lade Startadresse des Arrays
        lw      ..............      # Lade p
        addi    $a2, $s0, -4        # Neue letzte Adresse r=q-4
        .....   qsort               # Rufe qsort auf
        lw      ..............      # Lade Startadresse des Arrays
        addi    $a1, $s0, 4         # Neue erste Adresse p=q+4
        lw      ..............      # Lade letzte Adresse r
        .....   qsort               # Rufe qsort auf
done:   lw      ..............      # Hole Rücksprungadresse
        lw      ..............      # Hole Startadresse des Arrays
        lw      ..............      # Hole erste Adresse p
        lw      ..............      # Hole letzte Adresse r
        addi    ..............      # 4 Worte freigeben
        jr      ..............      # Rücksprung zum Aufruf
part:   addi    ..............      # Schaffe Platz für 1 Wort
        sw      ..............      # Speichere Rücksprungadresse
        lw      $t0, ($a2)          # Lade Pivot-Element
        addi    $t1, $a1, -4        # i = p-4
        addi    $t2, $zero, $a1     # j = p
        addi    $t3, $a2, -4        # Obere Schranke r-4 berechnen
loop:   bgt     $t2, $t3, loopend   # j > r-4 dann stop
        lw      $t4, ($t2)          # Lade elem A[j]
        addi    $t2, $t2, 4         # j = j + 4
        bgt     $t4, $t0, loop      # Springe wenn A[j] > x
        addi    $t1, $t1, 4         # i = i + 4
        lw      $t5, ($t1)          # Lade A[i]
        sw      $t4, ($t1)          # Speichere A[j] an Platz von A[i]
        sw      $t5, -4($t2)        # Speichere A[i] an Platz von A[j]
        j       loop                # Springe zu loop
loopend:addi    $t1, $t1, 4         # i = i + 4
        lw      $t5, ($t1)          # Hole A[i]
        sw      $t5, ($a2)          # Speichere A[i] an Platz von A[r]
        sw      $t0, ($t1)          # Speichere A[r] an Platz von A[i]
        addi    $v0, $t1, $zero     # Setzte Rückgabewert
        lw      ..............      # Hole Rücksprungadresse
        addi    ..............      # 1 Wort freigeben
        jr      ..............      # Rücksprung
```

Listing 4: MIPS-Assembler **(Ersatz! Ungültige Lösung streichen!)**

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

Gegeben sei folgendes Programm[1]

```
1  a = 1
2
3  for ( i = 0; i < 4; i++ )
4    if ( a == 1 )          // Verzweigung B1
5         // ...
6
7    if ( i > 1 )           // Verzweigung B2
8         a = 0
9         // ...
10
11   if ( i % 2 == 1)       // Verzweigung B3
12        // ...
```

(a) Beschreiben Sie das tatsächliche Sprungverhalten der Anweisungen B1, B2 und B3. Tragen Sie hierzu ein, ob für die jeweilige Iteration i die Verzweigung gewählt wird (T := taken) oder nicht (N := not taken). Hinweis: eine Verzweigung wird gewählt, wenn die Bedingung in der `if`-Abfrage zutrifft. (4 Punkte)

Iteration i	0	1	2	3
Entscheidung				

Tabelle 1: **Verzweigung B1**

Iteration i	0	1	2	3
Entscheidung				

Tabelle 2: Ersatztabelle

Iteration i	0	1	2	3
Entscheidung				

Tabelle 3: **Verzweigung B2**

Iteration i	0	1	2	3
Entscheidung				

Tabelle 4: Ersatztabelle

Iteration i	0	1	2	3
Entscheidung				

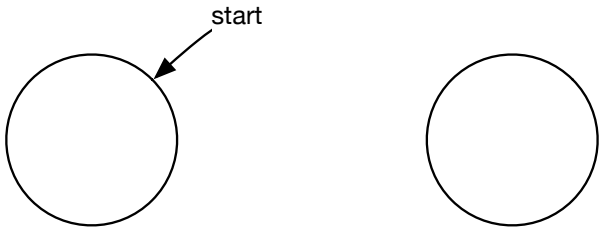
Tabelle 5: **Verzweigung B3**

Iteration i	0	1	2	3
Entscheidung				

Tabelle 6: Ersatztabelle

[1] % bezeichnet die Modulo-Operation. Diese berechnet den Rest r einer Division von a geteilt durch b. Beispiel: $42 \% 9 = 6$, da $42 = 9 * 4 + 6$.

(b) Zeichnen Sie den Automatengraphen eines 1-Bit Prädiktors. Wie lautet die Sprungvorhersage Ihres 1-Bit Prädiktors für die folgende Sequenz? (6 Punkte)



Tatsächlich	T	N	N	T	N	T	T	T	N	N	T	T
Prädiktion												

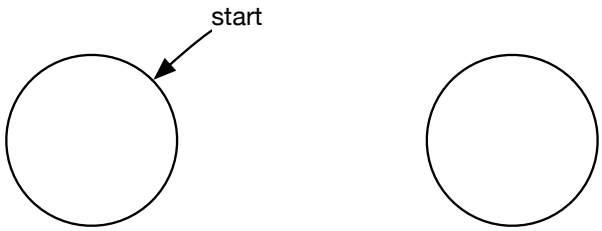


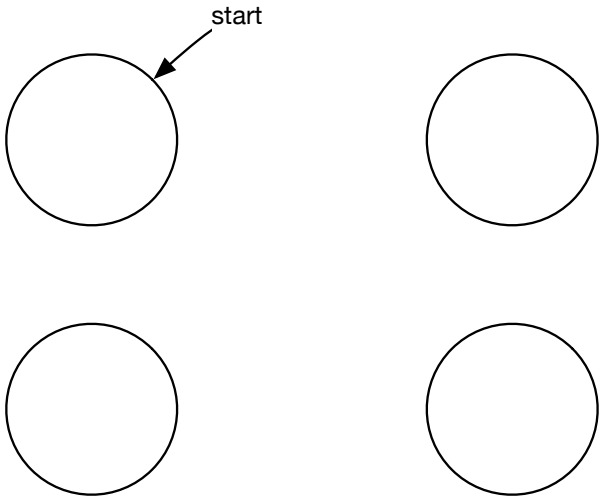
Abbildung 2: Ersatz

Tatsächlich	T	N	N	T	N	T	T	T	N	N	T	T
Prädiktion												

Tabelle 7: Ersatztabelle

(c) Wie hoch ist die Anzahl an Treffern für die Sequenz mit dem 1-Bit Prädiktor?

(d) Zeichnen Sie den Automatengraphen eines 2-Bit Prädiktors. Wie lautet die Sprungvorhersage Ihres 2-Bit Prädiktors für die gleiche Sequenz? (8 Punkte)



Tatsächlich	T	N	N	T	N	T	T	T	N	N	T	T
Prädiktion												
Folgezustand												

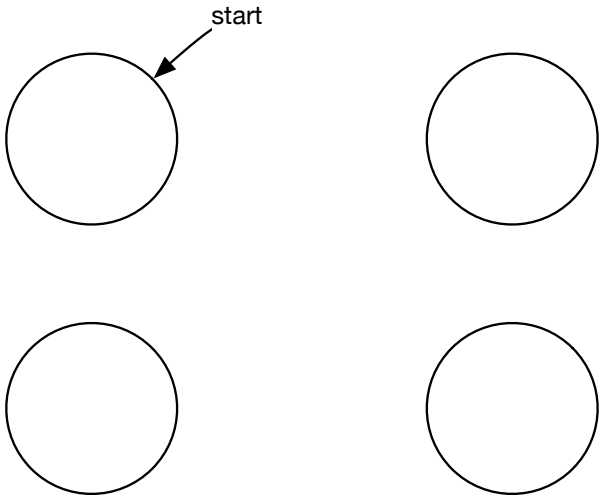


Abbildung 3: Ersatz

Tatsächlich	T	N	N	T	N	T	T	T	N	N	T	T
Prädiktion												
Folgezustand												

Tabelle 8: Ersatztabelle

(e) Wie hoch ist die Anzahl an Treffern für die Sequenz mit dem 2-Bit Prädiktor?

(f) Geben Sie eine Begründung für die erzielten Trefferquoten der beiden Prädiktoren. Wie kann man das Ergebnis verbessern? (2 Punkte)

Aufgabe 4 (I/O Arbitrierung) **[16 Punkte]**

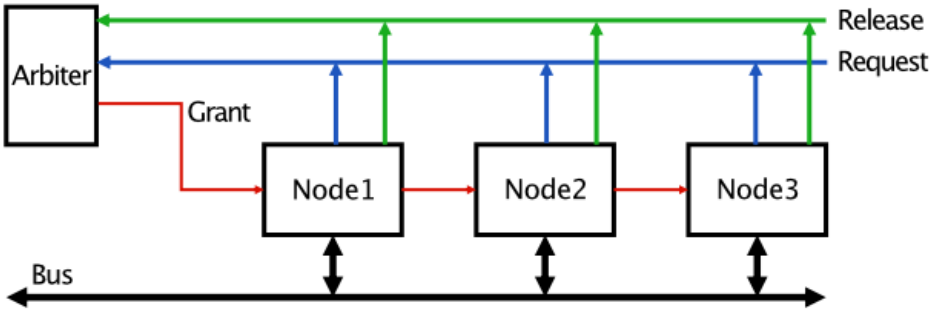


Abbildung 4: Bus mit Arbiter und drei Teilnehmern

Abbildung 4 zeigt drei Teilnehmer, die an einen gemeinsamen Bus angeschlossen sind. Damit nicht mehrere Teilnehmer gleichzeitig Daten auf den Bus legen, wird der Buszugriff arbitriert. Hierzu ist jeder Teilnehmer an je eine gemeinsame Release- und eine Request-Leitung angeschlossen. Der Grant-Ausgang des Arbiters wird von Teilnehmer zu Teilnehmer weitergereicht.

(a) Welches aus der Vorlesung bekannte Arbitrierungskonzept wird in Abbildung 4 verwendet? Ist die Arbitrierung fair? Begründen Sie Ihre Antwort in maximal drei Sätzen. (4 Punkte)

(b) Für die Arbitrierung soll nun ein zentraler Arbiter verwendet werden, an den pro Teilnehmer jeweils eine Request- und eine Grant-Leitung angeschlossen ist. Eine Release-Leitung wird hierfür nicht mehr benötigt. Die Funktionsweise der Arbitrierung soll unverändert bleiben.
Geben Sie den Automatengraphen eines Moore-Automaten, der den zentralen Arbiter beschreibt, an. Die Eingänge des Automaten sind die Request-Leitungen $(r_1, r_2, r_3) = (Req_{Node1}, Req_{Node2}, Req_{Node3})$; die Ausgänge des Automaten die Grant-Leitungen $(g_1, g_2, g_3) = (Grant_{Node1}, Grant_{Node2}, Grant_{Node3})$. (12 Punkte)

Aufgabe 5 (Datenpfad) **[20 Punkte]**

Gegeben sei folgende Befehlsfolge in einem Speicher:

```
0x10000000:   sw      $t0,    0($a0)
0x10000004:   lw      $t0,    4($a1)
0x10000008:   addi    $t1,    $a2,    2
0x1000000C:   add     $t2,    $t3,    $zero
0x10000010:   sub     $v0,    $t4,    $t3
```

(a) Die obige (konfliktfreie) Befehlsfolge wird von einem Prozessor verarbeitet, dessen Aufbau in Abb. 5 zu sehen ist (kein Forwarding, Halbtaktverfahren, Harvard Architektur). Geben Sie die Belegung der Steuerleitungen zu dem Zeitpunkt an, wo die Pipeline komplett gefüllt ist, der erste Befehl (`sw`) also die WB Phase erreicht hat. Tragen Sie hierfür in Tabelle 9 den aktuellen Wert der Steuerleitung ein, sowie für welchen Befehl sie gerade genutzt wird, beziehungsweise in welchem Befehlskontext sie gerade interpretiert werden muss. (12 Punkte)

Tabelle 9: Steuerleitungen bei gefüllter Pipeline

Steuerleitung	Wert	verantwortliche Instruktion / Kontext
PCSrc		
RegWrite		
ALUOperation		
ALUSrc		
RegDst		
MemWrite		
MemRead		
MemToReg		

Tabelle 10: Steuerleitungen (Ersatz, **ungültige Lösungen streichen**)

Steuerleitung	Wert	verantwortliche Instruktion / Kontext
PCSrc		
RegWrite		
ALUOperation		
ALUSrc		
RegDst		
MemWrite		
MemRead		
MemToReg		

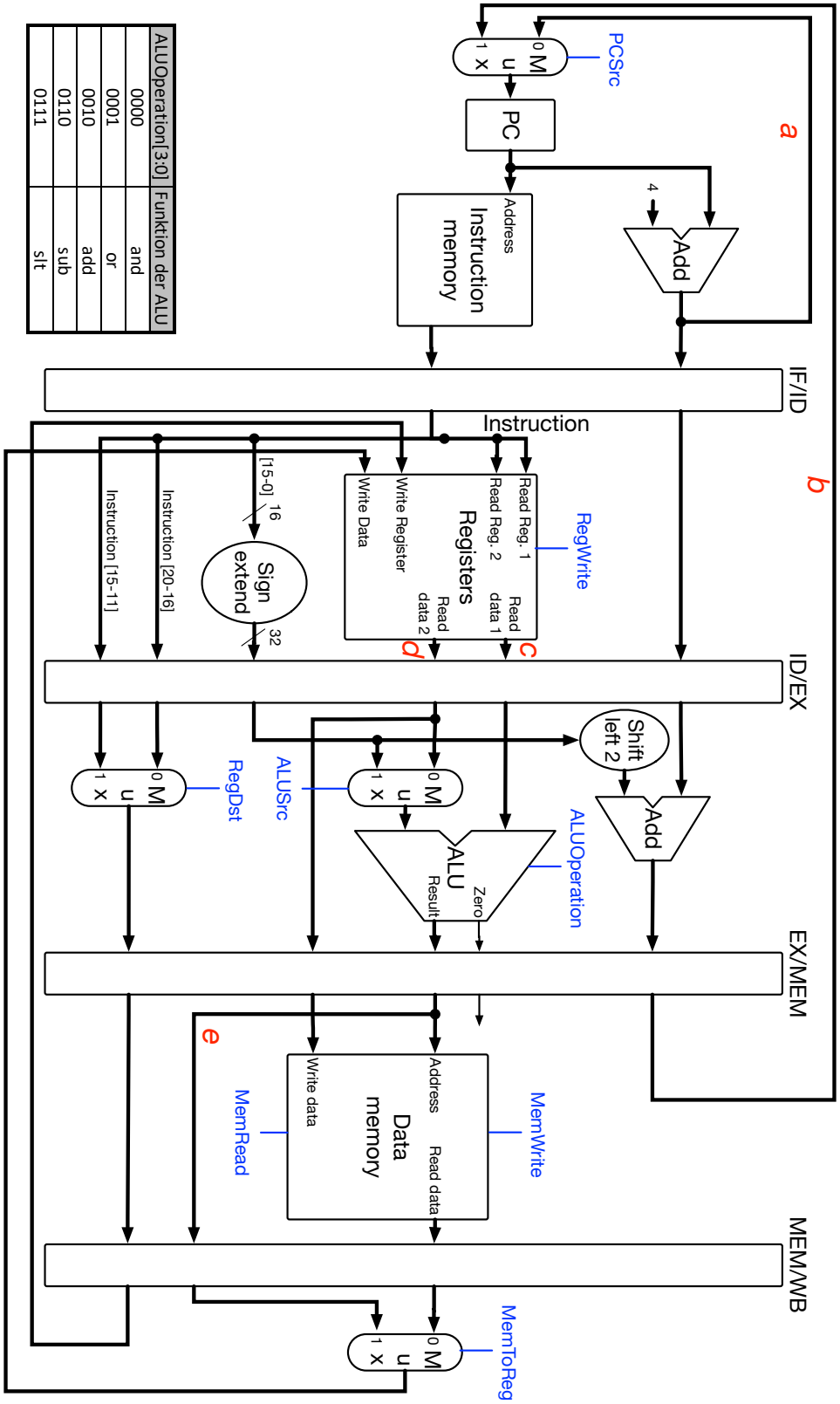


Abbildung 5: Datenpfad mit Pipelining

(b) Geben Sie nun zum selben Takt wie in Teilaufgabe a), also bei voller Pipeline, die konkreten Belegungen der in Abbildung 5 gekennzeichneten Stellen des Datenpfades in Tabelle 11 an. (8 Punkte)

Hinweis: Sie können sich zur besseren Übersicht in Abbildung 5 die aktuellen Instruktionen auf die Linien über den Pipelinestages schreiben.

Nehmen Sie folgende Werte vor Bearbeitung der ersten Instruktion an:

Register	Adresse	Inhalt
$v0	00010_2	42_{10}
$a0	00101_2	200_{10}
$a1	00110_2	0x1004001C
$a2	00111_2	65_{10}
$t0	01000_2	52_{10}
$t1	01001_2	0
$t2	01010_2	0
$t3	01011_2	225_{10}
pc	—	0x10000000

Tabelle 11: Datenleitungen

Stelle	Wert
a	
b	
c	
d	
e	

Tabelle 12: Datenleitungen (Ersatz, **ungültige Lösungen streichen**)

Stelle	Wert
a	
b	
c	
d	
e	

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!