Aufgabe 1 : Assembler

(maximal 23 Punkte)

Hinweis: *Auf der letzten Seite des Prüfungsbogens finden Sie eine Übersicht von Assemblerbefehlen. Alle für diese Aufgabe benötigten Befehle sind dort enthalten.*

Die C-Funktion `num2hex` stelle eine (im Binärformat gespeicherte) vorzeichenlose, ganze Zahl im Hexadezimalformat dar. Hierzu erzeugt sie eine Zeichenfolge („C-String“), bei der ein Byte jeweils eine Ziffer (0 – f) repräsentiert. Ein solches Byte erhält man dabei aus der Addition des Wertes der Ziffer und der Konstanten 48 (`'0'`) für die Werte 0 – 9 oder 87 (`'a'-10`) für die Werte a – f. Am Ende der Zeichenfolge wird zusätzlich ein Byte des Wertes Null (`'\0'`) angehängt. Als Beispiel betrachte man die Zahl $\texttt{70c}_{hex}$ und ihre Konvertierung:

Ziffer	7	0	c	Nullzeichen
Dezimalwert des Bytes	55	48	99	0

Das folgende MIPS Assemblerprogramm sei eine Implementierung der `num2hex`-Funktion, d.h., sie konvertiert eine Zahl `n` in eine Folge von Bytes, die sie im Speicher ablegt. Die Zahl `n` wird in `$a0` übergeben und die resultierende Zeichenkette wird in das Output-Array `s` geschrieben. Der Zeiger auf das erste Byte des Arrays wird in `$a1` übergeben. Es gibt **keinen** Branch-Delay-Slot.

```
1   num2hex:    ???
2               ???
3               add   $t3,   $a1,   $zero
4   n2h_loop:   andi  $t1,   $a0,   15
5               slti  $t2,   $t1,   10
6               bne   $t2,   $zero, n2h_L1
7               addi  $t1,   $t1,   39
8   n2h_L1:     addi  $t1,   $t1,   48
9               sb    $t1,   0($a1)
10              addi  $a1,   $a1,   1
11              srl   $a0,   $a0,   4
12              bne   $a0,   $zero, n2h_loop
13              sb    $zero, 0($a1)
14              add   $a0,   $t3,   $zero
15              sub   $a1,   $a1,   $t3
16              jal   reverse
17              ???
18              ???
19              jr    $ra
```

Die Funktion `reverse` (Zeile 16) kehrt dabei die Reihenfolge einer Zeichenkette um. So wird aus der Zeichenkette „`TI 1\0`“ die neue Zeichenkette „`1 IT\0`“. Der Zeiger auf das erste Element der Zeichenkette wird in `$a0` und deren Länge in `$a1` übergeben.

(a) (6 Punkte) Das obige MIPS Assemblerprogramm ist unvollständig. Ergänzen Sie das Assemblerprogramm, indem Sie an den mit `???` markierten Stellen je eine Instruktion einfügen.

Lösungsvorschlag:

```
1     num2hex:    addi  $sp,   $sp,   -4
2                 sw    $ra,   0($sp)
                  ⋮
17                lw    $ra,   0($sp)
18                addi  $sp,   $sp,   4
```

□

(b) (12 Punkte) Übersetzen Sie das obige Assemblerprogramm (sinngemäss) in eine C-Funktion. Die `reverse`-Anweisung soll mittels eines geeigneten Funktionsaufrufs realisiert werden. Zur Vereinfachung sind die Argumente der C-Funktion bereits gegeben.

```
void num2hex( unsigned int n, char *s ) {
    ...
}
```

Lösungsvorschlag:

```
1   void num2hex( unsigned int n, char *s) {
2       unsigned int i = 0;
3       do {
4           char c = n & 15;
5           s[i++] = (c < 10) ? c+'0' : c+'a'-10;
6       } while( ( n >>= 4 ) > 0 );
7
8       s[i] = '\0';
9       reverse( s, i );
10  }
```

□

(c) (5 Punkte) Erweitern Sie das obige Assemblerprogramm so, dass die `num2hex`-Funktion vor jeden ausgegebenen String das Präfix „`0x`“ setzt.

Hinweis: Das Zeichen `'x'` entspricht einem Byte des Wertes 120.

Lösungsvorschlag: Zwischen den Zeilen 2 und 3 fügt man folgende Instruktionen ein:

```
1        addi   $t0, $zero, 48     # s[0] = '0';
2        sb     $t0, 0($a1)        #
3        addi   $t0, $zero, 120    # s[1] = 'x';
4        sb     $t0, 1($a1)        #
5        addi   $a1, $a1, 2        # s += 2;
```

□

Aufgabe 2 : Pipelining (maximal 22 Punkte)

2.1: Hazards und Verzweigungen (maximal 13 Punkte)

(a) (7 Punkte) In dieser Aufgabe betrachten wir eine fünfstufige MIPS Pipeline-Architektur mit Forwarding. Die folgenden drei Pipelinediagramme zeigen einen Programmablauf, welcher auch die Architektur charakterisiert.
Fügen Sie den Assemblerprogrammen Instruktionen hinzu, sodass diese zu den Diagrammen passen. Nennen Sie in jedem Beispiel um welche Art von Hazard es sich handelt.
Für die Diagramme wurde folgende Schreibweise verwendet:

Forwarding: ↓ ↑ Stall: – Leeren der Pipeline: *flush*

Hinweis: Auf der letzten Seite des Prüfungsbogens finden Sie eine Übersicht von Assemblerbefehlen. Alle für diese Aufgabe benötigten Befehle sind dort enthalten.

1)

1	2	3	4	5	6	7	8	
IF	ID	EX	MEM	WB				`addi $t0, $t1, 13`
	IF	ID	EX	MEM↓	WB			`...`
		IF	ID	–	↑EX	MEM	WB	`...`

Hazard-Art:

2)

1	2	3	4	5	6	7	8	
IF	ID	EX	MEM	WB				`add $t0, $t1, $t2`
	IF	ID	EX	MEM	WB			`...`
		IF	*flush*					`...`
			IF	ID	EX	MEM	WB	`...`

Hazard-Art:

3)

1	2	3	4	5	6	7	8	
IF	ID	EX	MEM	WB				`lw $t0, 0($t1)`
	IF	ID	EX↓	MEM	WB			`...`
		IF	–	↑ID	EX	MEM	WB	`...`

Hazard-Art:

Lösungsvorschlag:

1) `addi $t0, $t1, 13`
   `lw` **`$t0`**`, 0($t2)`
   `addi $t1,` **`$t0`**`, 1`
   → Datenhazard.

2) `add $t0, $t1, $t2`
   `bnez $t1,` **label**
   `lw $t0, 0($t2)`
   label: `lw $t0, 0($t1)`
   → Ablaufhazard.

3) `lw $t0, 0($t1)`
   `add` **`$t0`**`, $t1, $t2`
   `beq $t1,` **`$t0`**`, label`
   → Datenhazard.

□

(b) (2 Punkte) In dieser Aufgabe sei nun eine leicht veränderte Variante der MIPS Prozessorarchitektur gegeben: Der Zugriff auf den Instruktionsspeicher und den Datenspeicher wird über denselben Bus geführt, daher kann gleichzeitig nur auf einen der beiden Speicher zugegriffen werden. Weshalb kann es in dieser Situation zu einem strukturellen Hazard kommen?

Lösungsvorschlag: Wenn eine Speicherinstruktion in der MEM-Stufe auf den Speicher zugreift, kann gleichzeitig keine neue Instruktion gelesen werden.

□

(c) (3 Punkte) Ein Programm habe drei bedingte Verzweigungen. Im Folgenden sind die Verzweigungsentscheidungen aufgelistet, welche bei einem Programmablauf gemacht werden (T: Es wird verzweigt, N: keine Verzweigung):

Verzweigung 1:T-T-T-T
Verzweigung 2:N-N-N-N-T
Verzweigung 3:T-N-T-N-T-N

Berechnen Sie jeweils die Voraussagegenauigkeit in Prozent für folgende zwei Strategien:

- *Immer kein Sprung* (Always not taken)

  Lösungsvorschlag:

  1) 0%
  2) 80%
  3) 50%

□

- 2-Bit Prädiktor (Initialzustand ist jeweils *strongly not taken*)

  Lösungsvorschlag:

  1) N - N - T - T → 50%
  2) N - N - N - N - N → 80%
  3) N - N - N - N - N - N → 50%

□

(d) (1 Punkt) Was versteht man unter einem Warteplatz (branch delay slot)?

Lösungsvorschlag: Die Semantik der Verzweigungsinstruktion wird verändert. Die Instruktion nach der Verzweigungsinstruktion wird in jedem Fall ausgeführt.

□

2.2: Datenpfad

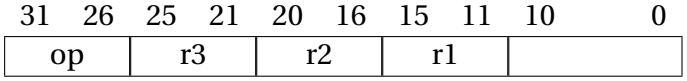
(maximal 9 Punkte)

(a) (9 Punkte) Abbildung 2 zeigt den aus der Vorlesung bekannten Datenpfad. Dort soll zusätzlich eine `seq`-Instruktion implementiert werden, die `1` ins Resultat-Register schreibt falls zwei Werte gleich sind und andernfalls `0`.

Instruktion: `seq r1, r2, r3`

Bedeutung: Setze `r1=1` falls (`r2==r3`); sonst `r1=0`

Die Instruktionskodierung sieht wie folgt aus:



1) Erweitern Sie Abbildung 2 um die `seq`-Instruktion. Sie dürfen dabei nur eine `zero extend` Einheit hinzufügen, welche analog zum `sign extend` ein *N* Bit breites Signal mit Nullen ergänzt, um ein 32 Bit breites Signal zu erhalten. Benutzen Sie dazu das Symbol aus Abbildung 1, wobei Sie *N* durch eine konkrete Bit-Anzahl ersetzen. Gegebenenfalls können Sie bereits im Datenpfad vorhandenen Elemente um weitere Eingänge erweitern.

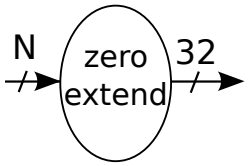


Abbildung 1: Symbol der `zero extend` Einheit mit *N* Bit breitem Eingangssignal.

2) Geben Sie die Steuersignale in der nachfolgenden Tabelle an. Falls ein Steuersignal für die Instruktion nicht relevant ist (don't care), so kennzeichnen Sie dies mit einem `x`.

Instr.	**EX Stage Control Lines**					**MEM Stage Control Lines**				**WB Stage Control Lines**		
	RegDst	ALUOp1	ALUOp0	ALUSrc		Branch	MemRead	MemWrite		RegWrite	MemtoReg	
`seq`												

Lösungsvorschlag: Tabelle unten, für Datenpfad siehe Abbildung 3

Instr.	**EX Stage Control Lines**				**MEM Stage Control Lines**			**WB Stage Control Lines**	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	MemRead	MemWrite	RegWrite	MemtoReg
`seq`	1	0	1	0	0	x	0	1	10 (oder 2, 0x2)

□

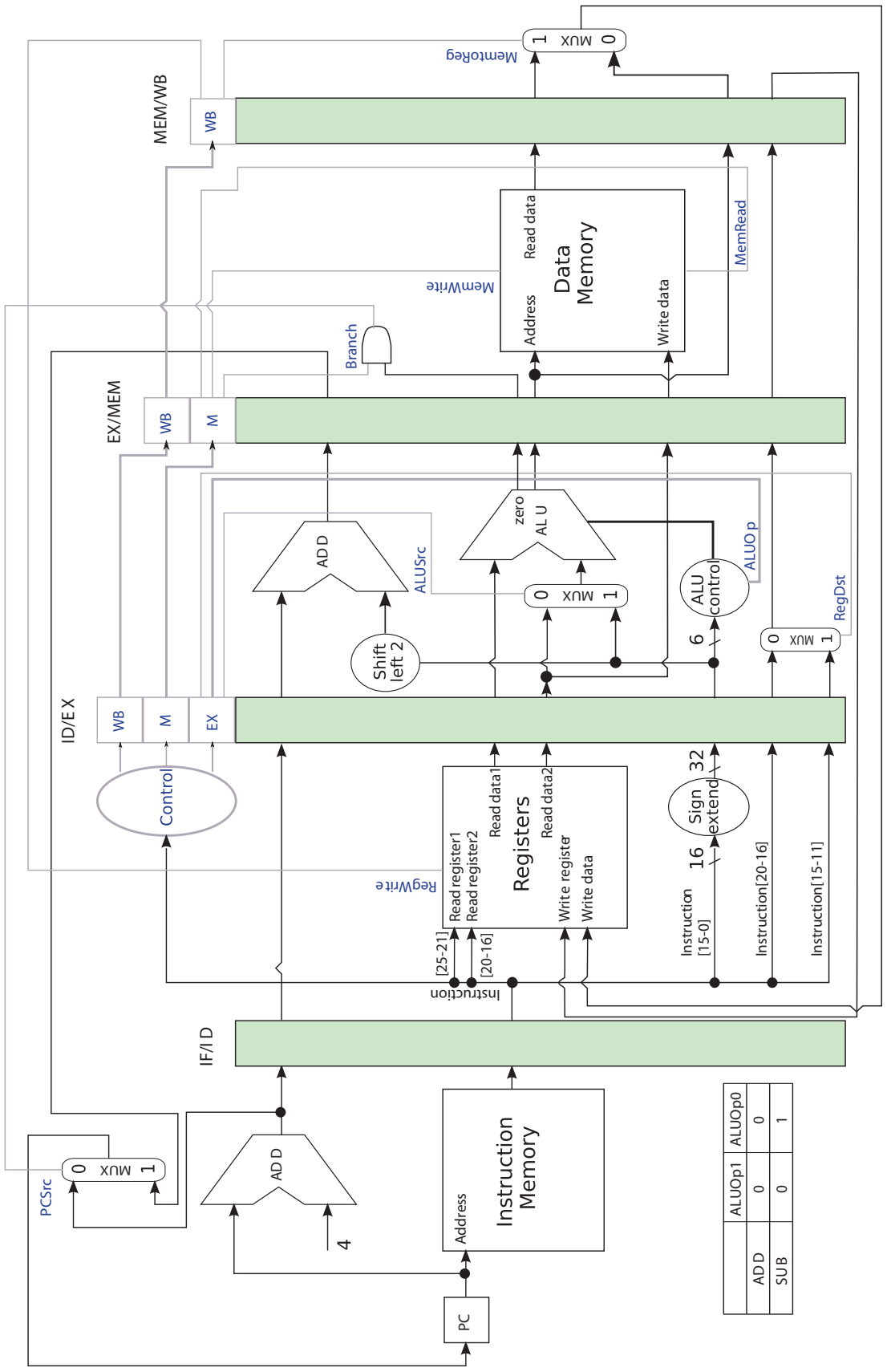


Abbildung 2: Datenpfad, Kontrolleinheiten und Steuersignale der CPU.

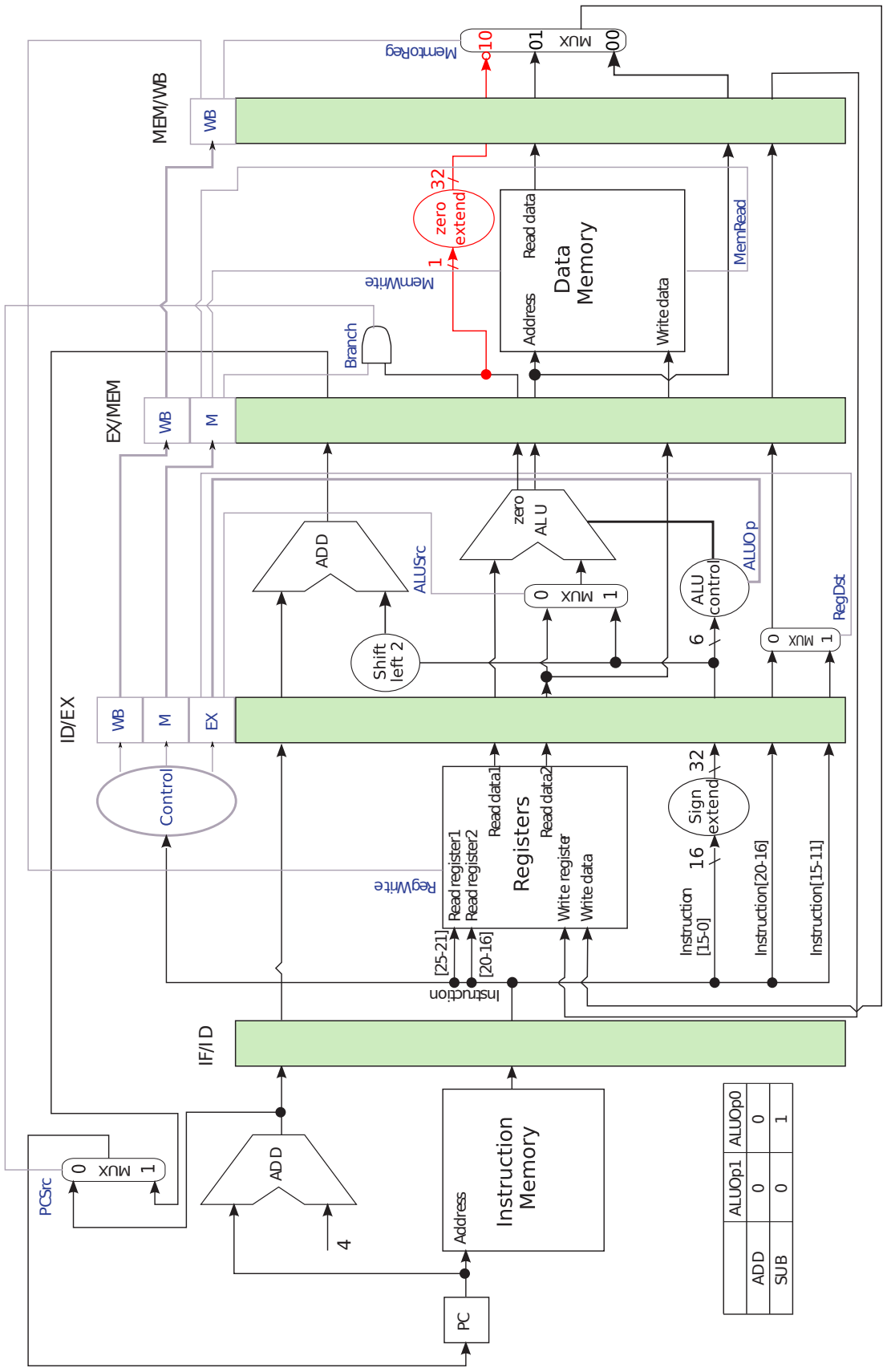


Abbildung 3: Lösung zu Datenpfad, Kontrolleinheiten und Steuersignale der CPU für die `seq`-Instruktion.

Aufgabe 3 : Cache (maximal 18 Punkte)

3.1: Cache Grundlagen (maximal 13 Punkte)

(a) (2 Punkte) Erklären Sie mindestens 3 Ziele, welche mit einer hierarchischen Speicherarchitektur erreicht werden sollen.

Lösungsvorschlag: Z.B.: Möglichst viel Speicherplatz, kostengünstig, kurze Zugriffszeit, hohe Bandbreite, nicht-flüchtig

□

(b) (2 Punkte) Nennen Sie je einen Vor- und Nachteil der folgenden Ersetzungsstrategien:

- Zufällig (random):

  Lösungsvorschlag: Vorteil: Einfache Implementierung, benötigt keine Speicherung von zuätzlicher Information
  Nachteil: In-effizienter im Vergleich mit LRU.

  □

- Am längsten unbenutzt (Least-Recently-Used):

  Lösungsvorschlag: Vorteil: Üblicherweise geringere Missrate als random ersetzen
  Nachteil: Komplexe Implementation

  □

(c) (2 Punkte) Gegeben ist ein ”quad-core”-Prozessor, wobei jeder der vier Prozessorkerne seinen eigenen L1 Cache hat. Ein ”Write-invalidate for Write-through”-Protokoll stellt die Kohärenz der Caches sicher. Erklären Sie, was im Cache des ersten Prozessorkerns vorgeht, wenn der vierte Prozessorkern in seinen Cache schreibt.

Lösungsvorschlag: Wenn der vierte Prozessor-Kern in seinen Cache schreibt, wird das Datum auf Grund der Write-Through Policy auch im Hauptspeicher aktualisiert. Wegen dem Snoopy-Protokol überwachen die anderen Kerne alle Zugriffe auf dem Memory-Bus. Falls nun der Tag im Cache von Kern 1 ”valid” ist, wird dieser auf”invalid” gesetzt.

□

(d) (3 Punkte) Treffen die folgenden Aussagen zu? Kreuzen Sie die richtigen Antworten an. Eine korrekte Antwort gibt 0.5 Punkte. Für eine falsche Antwort werden 0.5 Punkte abgezogen. Die Aufgabe gibt kein negatives Total.

- Beim durchgängingen Schreiben (Write-through) von Cache-Updates wird nur in die oberste Cache-Ebene geschrieben.

  ☐ richtig ☐ falsch

  Lösungsvorschlag: Falsch. Es wird sowohl in den Cache als auch in die unteren Speicherebenen geschrieben (0.5P)

  ☐

- Die Perfomance eines Cache ist weniger kritisch bei schnelleren Prozessoren, da die hohe Taktrate die langen Speicherzugriffszeiten kompensiert.

  ☐ richtig ☐ falsch

  Lösungsvorschlag: Falsch, bei einer grossen Prozessor-Speicher Geschwindigkeiten Lücke ist der Cache besonders wichtig. (0.5P)

  ☐

- Die örtliche Lokalität bei Speicherzugriffen besagt, dass nach einem Zugriff auf einen Adressbereich mit hoher Wahrscheinlichkeit der nächste Zugriff auf eine Adresse in unmittelbarer Nachbarschaft erfolgt.

  ☐ richtig ☐ falsch

  Lösungsvorschlag: Richtig. (0.5P)

  ☐

- Bei der direkten Abbildung von Speicherblöcken in den Cache wird bei einem Miss zufällig ein Cache-Block ersetzt.

  ☐ richtig ☐ falsch

  Lösungsvorschlag: Falsch, bei der direkten Abbildung ist die Speicher zu Cache Abbildung fix. (0.5P)

  ☐

- Das Dirty-Bit einer Cache-Zeile zeigt an, ob der jeweilige Cache-Block beim Ersetzen zurückkopiert werden muss.

  ☐ richtig ☐ falsch

  Lösungsvorschlag: Richtig. (0.5P)

  ☐

- Ein vollassoziativer Cache hat einen kleineren Speicher-Overhead im Vergleich zu einem Cache mit direkter Abbildung.

  ☐ richtig ☐ falsch

Lösungsvorschlag: Falsch, der Tag ist z.B. overhead und deshalb ist ein grösserer Overhead notwendig. (0.5P)

□

(e) (4 Punkte) Betrachten Sie eine Speicherhierarchie mit einem teilassoziativen L1-Cache und einem Hauptspeicher. Die mittlere Zugriffszeit eines Programms hängt von der Trefferrate (Hit-Rate) und Fehlzugriffsrate (Miss-Rate) beim Cache, sowie von der Miss-Strafe ab. Geben Sie an, bei welchen der angegebenen Massnahmen man eine Erhöhung oder eine Senkung der **Miss-Strafe** (in Zeiteinheiten) für den L1 Cache erwarten würde oder ob es keine direkte Abhängigkeit zur Miss-Strafe gibt. Begründen Sie jeweils ihre Antwort!

Hinweis: Jede Design-Alternative soll individuell berücksichtigt werden (alle übrigen Designparameter bleiben jeweils unverändert).

Massnahmen:

- Vergrösserung der Blockgrösse:

  Lösungsvorschlag: Erhöhung, da mehr Daten aus dem Hauptspeicher geholt werden müssen.

  □

- Erweiterung durch eine zweite Cacheebene (L2-Cache):

  Lösungsvorschlag: Senkung, es besteht die Möglichkeit aus dem L2 Cache Daten zu holen anstatt aus dem langsameren Hauptspeicher.

  □

- Ersetzung durch einen grösseren Cache:

  Lösungsvorschlag: Keine Abhängigkeit, wirkt sich auf Miss-Rate aber nicht auf die Strafe aus.

  □

- Verwendung eines Hauptspeichers mit mehreren parallel zugreifbaren Speicherbänken (interleaved memory organization):

  Lösungsvorschlag: Senkung. Durch das parallele Lesen wird die Speicherzugriffszeit verkleinert.

  □

3.2: Cache Status

(maximal 5 Punkte)

Gegeben ist ein direkt abgebildeter Cache mit einer 32-Bit-Adresse und Byte-adressiertem Speicher. Die Adresseinteilung in Tag, Index und Byte-Offset ist in Abbildung 4 gegeben. Gehen Sie zusätzlich davon aus, dass ein Wort 4-Byte lang ist und der Cache eine auf Zurückkopieren (Write back) basierende Ersetzungsstrategie verwendet. Beantworten Sie anhand der Abbildung folgende Fragen.

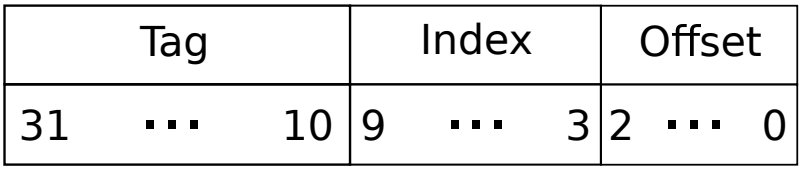


Abbildung 4: Adresseneinteilung in Tag, Index und Byte-Offset mit den Bit-Positionen.

(a) (1 Punkt) Wie viele Wörter befinden sich in einem Cache-Block?

Lösungsvorschlag: 3 Bits Offset $\Rightarrow \frac{2^3 Bytes}{4Bytes/Word} = 2\,Words$

□

(b) (1 Punkt) Wie viele Cache-Blöcke besitzt die gesamte Cache-Architektur?

Lösungsvorschlag: 7 Bits Index $\Rightarrow 2^7 Bits = 128\,Blocks$

□

(c) (3 Punkte) Berechnen Sie die Grösse des Caches in Anzahl Bits.

Lösungsvorschlag: $L = 32,\ M = 3,\ N = 10$
$\Rightarrow C = (1 + 1 + (L - N) + 8 * 2^M) * 2^{(N-M)} = (2 + 22 + 64) * 2^7 = 11264\,\text{Bits}$

□

Aufgabe 4 : Virtueller Speicher (maximal 27 Punkte)

4.1: Generelle Funktionsweise (maximal 5 Punkte)

(a) (1 Punkt) Nennen Sie zwei Gründe, weshalb virtueller Speicher verwendet wird.

Lösungsvorschlag: 1. Bengrenzter Speicherplatz, Virtueller Speicher kann mehr Speicher zur verfügung stellen als physikalisch vorhanden ist, zb. wenn mehrere Prozesse parallel laufen, 2. Isolation von Speicher

□

(b) (2 Punkte) Erläutern sie, was ein Translation Lookaside Buffer (TLB) ist und weshalb ein solcher verwendet wird (2-3 Sätze).

Lösungsvorschlag: Der TLB ist ein schneller lookup Puffer nahe am Prozessor um virtuelle Adressen in physikalische Adressen umzuwandeln. Dieser Puffer vermeidet den Umweg über die Seitentabelle im Hauptspeicher und beschleunigt deshalb die Umwandlung falls die gesuchte Seitennummer bereits im TLB ist.

□

(c) (2 Punkte) Wir betrachten ein System mit einer gegebenen physikalischen Speichergrösse. Die Seitengrösse ist konfigurierbar und ist entweder 4 kB oder 1 MB. Finden Sie je ein Argument, das für die eine oder die andere Konfiguration spricht.

Lösungsvorschlag: 4 kB: Speicher kann feingranularer verteilt werden, was zu besserer Speicherausnützung führt. / Beim Ersetzen einer Seite müssen weniger Daten kopiert werden. 1 MB: Kleinere Seitentabelle / Weniger Page/TLB misses da mit weniger Einträgen mehr Speicher verwaltet werden kann.

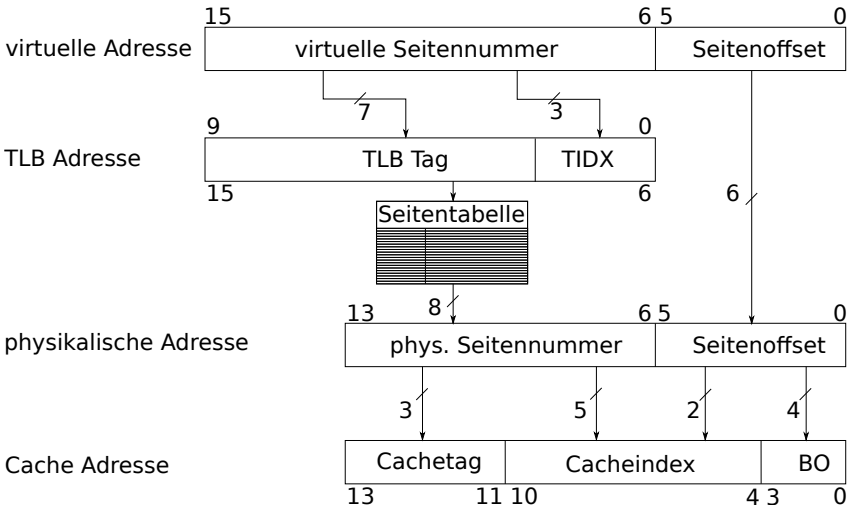
□

4.2: Adressaufteilung (maximal 6 Punkte)

Gegeben ist ein System mit einem virtuellen Adressraum von 16 Bit (Bits 0 bis 15), wobei der physikalische Speicher nur 2^{14} Byte speichern kann. Die Adressierung im virtuellen und im physikalischen Speicher erfolgt byteweise. Das System besitzt einen direkt abgebildeten TLB mit 8 Zeilen und die Seitengrösse beträgt 2^6 Byte. Auch besitzt das System einen 2-fach-assoziativ ausgeführten Cache mit 2^7 Zeilen und einer Daten-Blockgrösse von 16 Byte.

(a) (4 Punkte) Vervollständigen Sie die untenstehende Tabelle. Geben Sie an, welche Bits für eine Kennzahl verwendet werden (Spalte 2) und ob diese sich auf die virtuellen Adresse (V) oder die physikalischen Adresse (P) beziehen (Spalte 3).

Lösungsvorschlag:



Kennzahl	Bits	bezogen auf
Seitenoffset	0 bis 5	V, P
virtuelle Seitennummer	6 bis 15	V
TLB-Tag	9 bis 15	V
TLB-Index	6 bis 8	V
physikalische Seitennummer	6 bis 13	P
Cache-Byte-Offset	0 bis 3	P
Cache-Index	4 bis 10	P
Cache-Tag	11 bis 13	P

□

(b) (2 Punkte) An wievielen Orten im Cache kann ein Datum, das durch eine virtuelle Adresse angegeben ist, stehen?

Lösungsvorschlag: An $2 \cdot 2^5$ verschiedenen Stellen, da 5 Bits des Cache Index von der physikalischen Seitennummer abhängen und sich nicht direkt aus der virtuellen Adresse ergeben. Zudem wird jede Cachezeile doppelt geführt.

□

4.3: Hit- und Missraten (maximal 6 Punkte)

(a) (6 Punkte) Gegeben ist ein System mit einem virtuellen Speicher der Grösse $V = 2^{32}$ Byte und einer Seitengrösse $S = 2^{10}$ Byte. Bei Zugriffen auf zufällige Adressen, welche gleichverteilt sind, weist das System nach langer Zeit eine durchschnittliche TLB Missrate $m = \frac{63}{64}$ und eine durchschnittliche Seitenfehlerrate $p = 0.75$ auf. Berechnen Sie E, die Anzahl der Seiteneinträge der TLB, und G, die Grösse des physikalischen Speichers in Byte. Geben Sie eine vollständige Berechnung an, verwenden Sie dazu wenn nötig folgende zusätzliche Symbole:

- Anzahl der virtuellen Seiten E_V
- Anzahl der physikalischen Seiten E_P
- TLB-Hitrate h
- Seitenhitrate b

Lösungsvorschlag:

$$E_V = \frac{V}{S} = \frac{2^{32}}{2^{10}} = 2^{22} \qquad h = \frac{E}{E_V}$$

$$m = 1 - h \qquad \Rightarrow m = 1 - \frac{E}{E_V} \qquad \Rightarrow E = (1-m) \cdot E_V = \left(1 - \frac{63}{64}\right) \cdot 2^{22} = 2^{16}$$

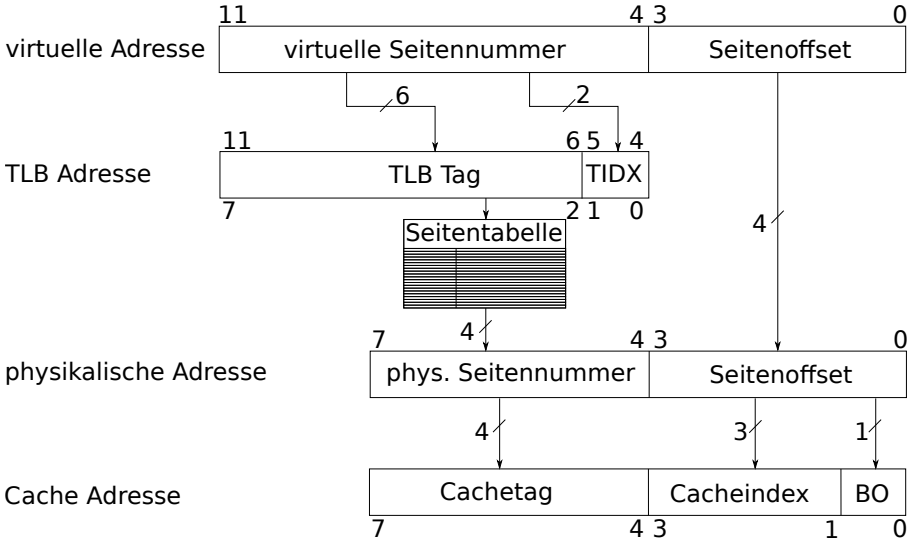
$$E_P = \frac{G}{S} \qquad b = \frac{E_P}{E_V} \qquad p = 1 - b = 1 - \frac{E_P}{E_V}$$

$$G = S \cdot E_V \cdot \big(1 - p\big) = 2^{10} \cdot 2^{22} \cdot (1 - 0.75) = 2^{30}$$

□

4.4: Virtuelle Adressierung, TLB und Cache (maximal 10 Punkte)

Gegeben ist ein System mit folgender Adressaufteilung:



Im initialen Zustand ist der Inhalt des TLB, ein Ausschnitt der Seitentabelle und der Cache wie folgt gegeben (alle Angaben sind im Hexadezimal Format):

TLB:

Index	Tag	PPN	Valid	Tag	PPN	Valid
0x0	0x1f	0x4	0x1	0x13	0xa	0x0
0x1	0x02	0xf	0x1	0x0c	0xa	0x1
0x2	0x01	0x1	0x0	0x00	0x5	0x1
0x3	0x0a	0xf	0x1	0x0e	0x0	0x1

Ausschnitt der Seitentabelle:

VPN	PPN	Valid
0x40	0xc	0x0
0x41	0x1	0x1
0x42	0x0	0x1
0x43	0x4	0x0
0x44	0x5	0x1
0x45	0xf	0x0
0x46	0xc	0x0
0x47	0x0	0x0

VPN	PPN	Valid
0x48	0xa	0x0
0x49	0xe	0x0
0x4a	0xf	0x1
0x4b	0x6	0x1
0x4c	0x7	0x1
0x4d	0xd	0x0
0x4e	0x8	0x1
0x4f	0x9	0x0

Cache:

Index	Tag	Valid	Byte[0x0]	Byte[0x1]	Tag	Valid	Byte[0x0]	Byte[0x1]
0x0	0x0	0x1	0x96	0x35	0x1	0x1	0x4d	0x79
0x1	0x7	0x1	0x3b	0xd8	0xf	0x1	0x32	0x3a
0x2	0xa	0x0	0x2c	0x70	0x2	0x0	0x50	0xec
0x3	0x2	0x1	0x6e	0x30	0x1	0x1	0xe7	0xfa
0x4	0x3	0x0	0x1d	0x42	0x5	0x0	0x69	0x98
0x5	–	–	–	–	0x9	0x1	0x43	0x9a
0x6	0xa	0x1	0x1e	0x4c	–	–	–	–
0x7	0x3	0x1	0x82	0x16	0x1	0x1	0xcd	0x08

HINWEIS: Die folgenden Aufgabenstellungen lassen sich unabhängig voneinander lösen.

(a) (5 Punkte) Das System befindet sich im initialen Zustand. Es wird nun die Adresse `0x46f` gelesen. Füllen Sie die folgenden Tabellen vollständig mit **Hexadezimalzahlen** aus. Wenn ein Parameter nicht ermittelbar ist, markieren Sie die entsprechende Zelle mit einem "-".

Lösungsvorschlag: `0x46f = 0b 0100 0110 1111`

Parameter	Wert
VPN	0x46
Seitenoffset	0xf
TLB-Index	0x2
TLB-Tag	0x11
TLB-Miss (j/n)	j
Seitenfehler (j/n)	j

Parameter	Wert
physikalische Adresse	-
PPN	-
Byteoffset	0x1 or -
Cacheindex	0x7 or -
Cachetag	-
Cache-Miss (j/n)	-
resultierendes Datenbyte	-

(b) (5 Punkte) Das System befindet sich im initialen Zustand. Es wird nun an die Adresse `0x4c2` geschrieben. Füllen Sie die folgenden Tabellen vollständig mit **Hexadezimalzahlen** aus. Wenn ein Parameter nicht ermittelbar ist, markieren Sie die entsprechende Zelle mit einem "-".

Lösungsvorschlag: `0x4c2 = 0b 0100 1100 0010`

Parameter	Wert
VPN	0x4c
Seitenoffset	0x2
TLB-Index	0x0
TLB-Tag	0x13
TLB-Miss (j/n)	j
Seitenfehler (j/n)	n

Parameter	Wert
physikalische Adresse	0x72
PPN	0x7
Byteoffset	0x0
Cacheindex	0x1
Cachetag	0x7
Cache-Miss (j/n)	n
resultierendes Datenbyte	0x3b

Anhang

MIPS-Instruktionssatz

Zusammenfassung einiger MIPS-Assemblerinstruktionen, die in den Aufgaben 1 und 2 benötigt werden könnten.

Instruktion	Bedeutung
`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`
`addi $s1, $s2, 100`	`$s1 = $s2 + 100`
`addiu $s1, $s2, 100`	`$s1 = $s2 + 100` (unsigned)
`addu $s1, $s2, $s3`	`$s1 = $s2 + $s3` (unsigned)
`and $s1, $s2, $s3`	`$s1 = $s2 & $s3` (bitweises Und)
`andi $s1, $s2, 100`	`$s1 = $s2 & 100` (bitweises Und)
`beq $s1, $s2, 25`	Wenn `($s1 == $s2)`, verzweige zu `PC + 4 + 100`
`bne $s1, $s2, 25`	Wenn `($s1 != $s2)`, verzweige zu `PC + 4 + 100`
`j label`	Springe zu `label`
`jal label`	Springe zu `label` und setze `$ra`
`jr $s1`	Springe zu `$s1`
`li $s1, 100`	`$s1 = 100`
`lui $s1, 100`	`$s1 = 100 * 2`^{16}
`lw $s1, 100($s2)`	`$s1 = Speicher[$s2 + 100]`
`move $s1, $s2`	`$s1 = $s2`
`mul $s1, $s2, $s3`	`$s1 = $s2 * $s3`
`nop`	No operation
`sll $s1, $s2, 10`	`$s1 = $s2 << 10`
`slt $s1, $s2, $s3`	Setze `$s1=1` falls `($s2 < $s3)`; sonst `$s1=0`
`slti $s1, $s2, 100`	Setze `$s1=1` falls `($s2 < 100)`; sonst `$s1=0`
`sltu $s1, $s2, $s3`	Setze `$s1=1` falls `($s2 < $s3)` (unsigned); sonst `$s1=0`
`srl $s1, $s2, 10`	`$s1 = $s2 >> 10`
`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`
`sb $s1, 100($s2)`	`Speicher[$s2 + 100] = $s1` (ein Byte)
`sw $s1, 100($s2)`	`Speicher[$s2 + 100] = $s1` (ein Wort)