Aufgabe II-1: Fragen zur Rechnerarchitektur (10 P)

a) Geben Sie an, ob die folgenden Aussagen wahr (W) oder falsch (F) sind. Falsch angekreuzte Antworten führen nicht zu Punktabzug.

W F

◯ ◯ 1) Pipelining ist eine Architekturtechnik.

◯ ◯ 2) Dynamische Sprungvorhersage ist eine Mikroarchitekturtechnik.

◯ ◯ 3) Typische Eigenschaften eines CISC-Prozessors sind die einheitliche Befehlslänge und eine hohe Anzahl an Universalregistern.

◯ ◯ 4) Die Adresse des Speicherwortes zeigt bei einem wortadressierbaren Speicher mit Little-Endian-Anordnung immer auf das höchstwertige Byte.

◯ ◯ 5) Mit der ungepackten BCD-Darstellung können bei 64 Bit Wortlänge insgesamt 8 Zeichen pro Wort kodiert werden.

◯ ◯ 6) Bei Register-Register-Architekturen gibt es spezielle Lade-/Speicherbefehle, um auf den Hauptspeicher zugreifen zu können.

◯ ◯ 7) Bei der registerindirekten Adressierung mit Verschiebung wird der Offset aus einem speziellen Indexregister zur Speicheradresse hinzu addiert.

◯ ◯ 8) Bei einer superskalaren RISC-Pipeline müssen Gegenabhängigkeiten zwischen Befehlen beachtet werden.

◯ ◯ 9) SRAM-Zellen (statischer RAM) müssen regelmäßig ausgelesen und neu beschrieben werden, um ihren Zustand zu erhalten.

◯ ◯ 10) Flash-Zellen bestehen aus einem MOS-Transistor, in welchem eine zusätzliche Elektrode im Isolator angebracht ist.

◯ ◯ 11) Beim direkt abgebildeten Cache-Speicher entspricht die Assoziativität der Anzahl an enthaltenen Cache-Blöcken ($m = n$).

◯ ◯ 12) Bei DDR3-Speicheranbindung werden acht Speicherwörter vorgeladen und in vier Taktzyklen übertragen.

b) Geben Sie drei Befehlsarten an und nennen Sie jeweils beispielhaft einen MIPS-Befehl für jede Art.

c) Es sei ein System mit 4 GB Adressraum (byteadressierbar) und 32 Bit Speicheradressen gegeben. Zur Adressierung einzelner Bits in den Registern von E/A-Geräten wird Bit-Banding eingesetzt. Der Bit-Band-Bereich belegt `0x10000000 - 0x1000FFFF`, der Bit-Band-Alias-Bereich `0x20000000 - 0x201FFFFF`.

Geben Sie die Bit-Band-Alias-Adresse für das Bit 4 im Speicherwort 0x10001002 an. (Rechnung erforderlich!)

Lösungsvorschläge

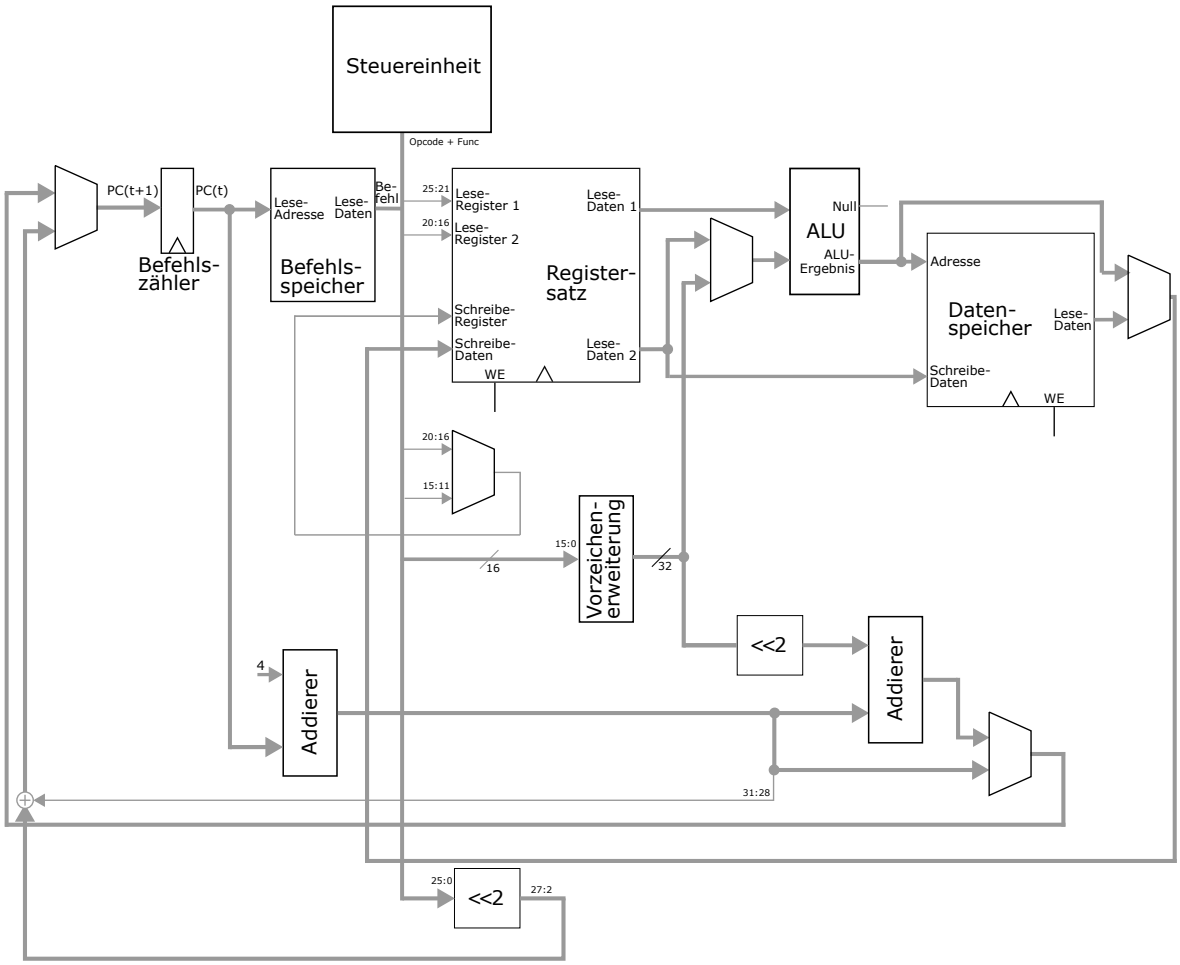
a) Die Aussagen 2, 5, 6, 8, 10 und 12 sind wahr.

b) Siehe Kurs.

c) (Rechnung in hex): 0x20000000 + (1002 * 20) + (4 * 4) = 0x20020040

Aufgabe II-2: Mikroarchitekturen (8 P)

a) Gegeben sei die unten stehende Einzyklus-Mikroarchitektur des MIPS-Prozessors, wie sie auch im Kurstext eingeführt wurde. Aus Gründen der Übersichtlichkeit sind die Steuerleitungen entfernt worden und die Steuereinheit nur vereinfacht dargestellt. Zeichnen Sie den Datenpfad für den Befehl `j` (*Jump*) deutlich in die Abbildung ein.



b) Geben Sie in Textform an, was der Befehl `jal` (*Jump and Link*) bewirkt und wie der Befehl `jr` (*Jump Register*) mit diesem zusammenhängt.

c) Gegeben sei folgender MIPS-Programmcode. Zeichnen Sie den zugehörigen Abhängigkeitsgraphen bzgl. Datenabhängigkeiten. Markieren Sie hierbei die Kanten mit (E) für eine echte Datenabhängigkeit, (A) für eine Ausgabeabhängigkeit oder (G) für eine Gegenabhängigkeit.

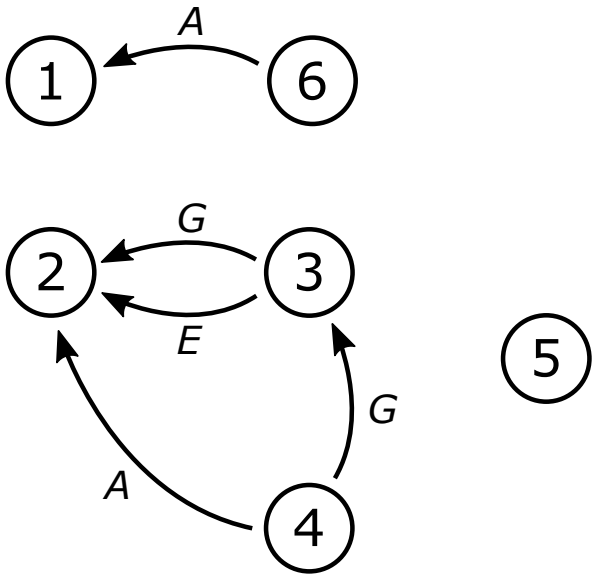
```
1          subi    $t4, $t4, 0x2
2          add     $t1, $t2, $t3
3          mul     $t2, $t1, $t5
4          sub     $t1, $t5, $t7
5          addi    $t6, $zero, 0x1
6          and     $t4, $t8, $t9
```

Lösungsvorschläge

a) Siehe Abb. 2.27 im Kurstext.

b) `jal` springt zur Zieladresse und speichert die Rücksprungadresse im Register `$31` (Unterprogrammaufruf). Mit dem Befehl `jr` kann am Ende des Unterprogramms wieder mit dem aufrufenden Programm fortgefahren werden (`jr $31`). Registerbelegungen müssen ggf. zwischengespeichert werden.

c) Eine mögliche Lösung (invertierte Pfeilrichtung ebenso korrekt):



Aufgabe II-3: Code-Analyse und IEEE-754 (20 P)

Gegeben sei der unten stehende, an die MIPS-Architektur angelehnte, Assembler-Code.

```
1  .text
2           lw      $t1, in
3           lw      $t5, masks
4           lw      $t6, maske
5           lw      $t7, maskm
6           move    $t9, $zero
7  b1:
8           and     $t2, $t1, $t6
9           srl     $t2, $t2, 0x17
10          subi    $t2, $t2, 0x7F
11          blt     $t2, $zero, end
12 b2:
13          and     $t3, $t1, $t7
14          addi    $t3, $t3, 0x800000
15          addi    $t4, $zero, 0x17
16          sub     $t4, $t4, $t2
17          srlv    $t3, $t3, $t4
18          move    $t9, $t3
19          and     $t2, $t1, $t5
20          beqz    $t2, end
21 b3:
22          not     $t3, $t3
23          addi    $t3, $t3, 1
24          move    $t9, $t3
25 end:
26          j       end
27
28 .data
29          in:     .word 0x41ca0000
30          masks:  .word 0x80000000
31          maske:  .word 0x7F800000
32          maskm:  .word 0x7FFFFF
```

Hinweise:

- Der Befehl `srlv $x, $y, $z` schiebt den Inhalt des Registers `$y` um die Anzahl an Stellen, welche in `$z` gespeichert werden, logisch nach rechts und speichert das Ergebnis im Register `$x`.
- Das Register `$zero` enthält feste den Wert 0.
- Der Wert `0x7F` entspricht 127_{10}.

a) Erklären Sie in einem Satz folgende Assembler-Anweisungen (vgl. Beispiel am Ende dieses Aufgabenteils!):
Zeile 10: `subi $t2, $t2, 0x7F`

. . . . . .

. . . . . .

Zeile 11: `blt $t2, $zero, end`

. . . . . .

. . . . . .

Zeile 19: `and $t2, $t1, $t5`

. . . . . .

. . . . . .

Beispiel: Zeile 3: `lw $t5, masks` : Load Word: Die Anweisung lädt das Wort in Register `$t5`, welches an derjenigen Stelle im Speicher steht, auf welche die Maske `masks` verweist.

b) Der Wert an der Speicherstelle `in` stellt eine Gleitkommazahl nach IEEE-754 dar. Geben Sie die Zahl im Dezimalsystem an (Umrechnung erforderlich!).

c) Erklären Sie, was die folgenden Code-Blöcke funktional realisieren (vgl. Beispiel am Ende dieses Aufgabenteils!):

Zeilen 8 – 11

. . . . . .

. . . . . .

. . . . . .

Zeilen 13 – 20

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Zeilen 22 – 24

. . . . . .

. . . . . .

. . . . . .

Beispiel: Zeilen 2 – 6: Dies ist die Initialisierung. Es werden der Eingabewert sowie drei Bit-Masken geladen. Außerdem wird das Ausgaberegister `$t9` mit 0 vorbelegt.

d) Beschreiben Sie in einem Satz, was das Programm berechnet!

. . . . . .

. . . . . .

. . . . . .

e) Geben Sie an, was der Befehl `srl` in Zeile 9 macht. Vergleichen Sie diesen Befehl mit dem (recht ähnlichen) MIPS-Befehl `sra`. Worin liegen die Gemeinsamkeiten bzw. Unterschiede? Könnte in Zeile 9 auch der Befehl `sra` verwendet werden, ohne die Programmfunktion zu verändern?

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Lösungsvorschläge

a) Zeile 10: `subi $t2, $t2, 0x7F`: Subtract immediate: Vom Wert in Register `$t2` wird `0x7F` bzw. 127_{10} subtrahiert und das Ergebnis in `$t2` zurück geschrieben..
Zeile 11: `blt $t2, $zero, end`: Branch on less than: Bedingter Sprung zur Marke `end`, falls das Register `$t2` kleiner als 0 ist.
Zeile 19: `and $t2, $t1, $t5`: Bitwise AND: Verknüpft die Werte der Register `$t1` und `$t5` logisch mit UND und speichert das Ergebnis in `$t2`.

b) Der Zahl lautet im Dezimalsystem $25,25_{10}$.

Umrechnung:
Hex nach Bin: `41ca0000` $\rightarrow$ $01000001110010100000000000000000_2$
Vorzeichenbit: 0
Exponent: $10000011_2 = 131_{10} \rightarrow 131 - 127 = 4$
Mantisse: $1{,}1001010\ldots0_2 \rightarrow 11001{,}010\ldots0_2$
Bin nach Dec: 25,25

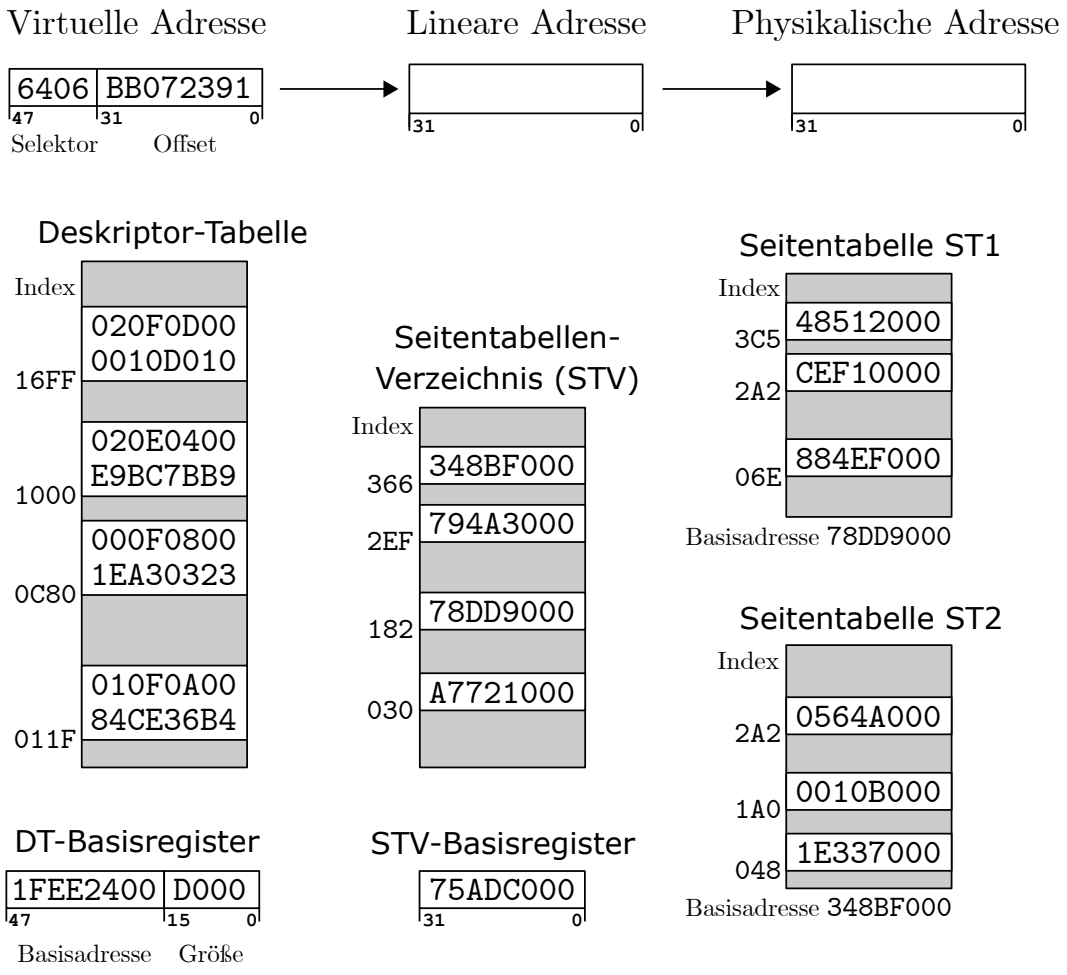
c) Zeilen 8 – 11:
Der versch. Exponent wird mit `maske` ausgeschnitten und an das LSB verschoben. Anschließend wird der Exponent berechnet, indem `0x7F` bzw. 127_{10} subtrahiert werden. Ist der Exponent kleiner als 0, dann wird das Programm beendet (Ergebnisregister `t9` ist 0).
Zeilen 13 – 20:
Die Mantisse wird ausgeschnitten und die implizite 1 wird hinzugefügt. Anschließend wird die Mantisse so oft verschoben, bis das LSB der Stelle 2^0 entspricht. Ist das Vorzeichenbit gleich 0 (positive Zahl), wird das Programm beendet, nachdem das Ergebnis in `t9` geschrieben wurde.
Zeilen 22 – 24:
Handelt es sich um eine negative Zahl (Vorzeichenbit gleich 1), dann wird zusätzlich noch die entsprechende Zweierkomplementdarstellung berechnet.

d) Das Programm gibt den ganzzahligen Wert einer in IEEE-Darstellung angegebenen Zahl im Zweierkomplement aus (Eingabe über die Marke `in`, Ausgabe über Register `$t9`).

e) Der Befehl `srl` verschiebt den Registerwert logisch nach rechts (0en werden von links eingefügt), während bei `sra` der Wert arithmetisch nach rechts geschoben wird (Vorzeichenbit bleibt entsprechend erhalten). In Zeile 9 wird der verschobene Exponent um 23 Stellen nach rechts verschoben. Da dieser laut IEEE-754 immer positiv ist, haben `srl` und `sra` die gleiche Wirkung (0en werden von links eingefügt). Der Befehl `sra` könnte hier ebenso verwendet werden, ohne das Programm funktional zu ändern.

Aufgabe II-4: Virtuelle Speicherverwaltung (12 P)

Betrachten Sie die folgende Abbildung zur virtuellen Speicherverwaltung der IA-32-Architektur (32-Bit-Adressen, 4 kByte Seiten, keine Adressraumerweiterung). Für die Umrechnung der virtuellen Adresse in die physikalische Adresse wird zunächst eine Segmentierung und anschließend eine mehrstufige Seitenverwaltung durchgeführt. Die relevanten Einträge in der Deskriptor-Tabelle sowie den Seitentabellen bzw. dem Seitentabellenverzeichnis sind hierfür eingetragen. Jeder Tabelleneintrag wird mit seinem Index angegeben. Außerdem sind die Basisadressen der Seiten sowie die Inhalte des Deskriptortabellen-Basisregisters und des Seitentabellenverzeichnis-Basisregisters angegeben. Alle Zahlenangaben (außer die Bitbezeichner der Register) sind in hexadezimaler Form angegeben.



a) Die Einträge in der Deskriptor-Tabelle sind jeweils 8 Byte groß. Die unteren 32 Bit enthalten die Basisadresse des Segments, die oberen 32 Bit enthalten jeweils ein Kontrollbyte und die entsprechende Segmentlänge. Die virtuelle Adresse besteht aus 48 Bit, die sich in den 16 Bit großen Selektor sowie den 32 Bit großen Offset unterteilen. Die oberen 13 Bit des Selektors werden als Index in die Deskriptor-Tabelle verwendet. Führen Sie die Schritte zur Berechnung der linearen Adresse durch und tragen Sie diese in das entsprechende Feld der Abbildung ein (Rechnung erforderlich!).

b) Geben Sie an, wie viele Einträge die Deskriptortabelle maximal enthalten kann und wie groß diese dann wird (Rechnung erforderlich!).

c) Die höchstwertigen 10 Bit der linearen Adresse werden als Index für einen Eintrag in dem Seitentabellen-Verzeichnis verwendet. Der entsprechende Eintrag enthält die 32-Bit lange Basisadresse einer Seitentabelle. Die mittleren 10 Bit der linearen Adresse werden dann als Index für die ausgewählte Seitentabelle verwendet. Die Einträge der Seitentabelle sind ebenfalls 32 Bit lang und enthalten die Basisadressen der einzelnen Seiten. Berechnen Sie die physikalische Adresse und tragen Sie diese in das entsprechende Feld in die Abbildung ein. Falls Sie die lineare Adresse aus Aufgabenteil a) nicht berechnen konnten, dann wählen Sie `60AA2D63` als lineare Adresse (Rechnung erforderlich!).

d) Bei der Seitenverwaltung der IA-32-Architektur wird ein mehrstufiges Verfahren angewandt (STV $\rightarrow$ ST $\rightarrow$ Seite). Begründen Sie, wo die Vor- und Nachteile eines solchen Verfahrens liegen. Wie kann die Berechnung beschleunigt werden?

Lösungsvorschläge

a) Index berechnen: `0x6406` = `0110 0100 0000 0|110` → `0C80`
Segment-Basisadresse lautet: `0x1EA30323`
Addition mit Offset ergibt lineare Adresse: `0x1EA30323` + `0xBB072391` = `0xD9AA26B4`

b) Ein 13-Bit-Selektor erlaubt $2^{13} = 8192$ Einträge. Multipliziert mit 8 Byte pro Eintrag ergeben sich 64 kByte für die Deskriptor-Tabelle.

c) Index STV: oberen 10 Bit der lin. Add.: `1101 1001 10` → `366`.
Der Eintrag im STV enthält die ST-Basisadresse `0x348BF000` → ST2 ausgewählt.
Index ST2: mittlere 10 Bit der lin. Add.: `01 1010 0010` → `2A2`.
Der Eintrag in ST2 enthält die Seiten-Basisadresse `0x0564A000`.
Zusammen mit den unteren 12 Bit (Offset) ergibt sich die phy. Adr.: `0x0564A6B4`

Alternative mit lin. Adresse `0x60AA2D63`:
Index STV: `0110 0000 10` → `182`.
ST-Basisadresse `0x78DD9000` → ST1 ausgewählt.
Index ST1: `10 1010 0010` → `2A2`.
Seiten-Basisadresse: `0xCEF1000`.
Phy. Adresse: `0xCEF1D63`.

d) Vorteile: Wird nur eine Seitentabelle verwendet, welche die Basisadressen aller Seiten enthält, muss diese dauerhaft im Speicher gehalten werden. Bei einem 4 GB großen Adressraum, 4 KB großen Seiten und 4 Byte je Eintrag würde die Seitentabelle 4 MB belegen (bei Markteinführung IA-32 sehr viel!). Unterteilung auf STV und ST erlaubt es, die ST ebenfalls als Seiten zu betrachten und ein- bzw. auslagern zu können. Lediglich die STV muss dauerhaft im Hauptspeicher gehalten werden.

Nachteile: Um die phy. Adresse zu berechnen, muss mehrfach auf den Hauptspeicher zugegriffen werden (Eintrag im STV, Eintrag in ST, Eintrag in Seite). Liegen die Seiten nicht im Cache, sodass die Zugriffe in den Hauptspeicher bzw. sogar in den Hintergrundspeicher gehen, so verzögert sich die Adressberechnung entsprechend.

Beschleunigt werden kann die Adressberechnung durch einen zusätzlichen Cache, den *Translation Lookaside Buffer*, TLB. Dieser enthält die oberen 20 Bit der zuletzt berechneten physikalischen Adressen und die entsprechenden oberen 20 Bit der linearen Adressen. Bei einem Cache-Hit kann die Adressberechnung entfallen, da die oberen 20 Bit der phy. Adresse direkt zur Verfügung stehen. Die unteren 12 Bit (Offset) müssen nicht gespeichert werden, da diese bei lin. und phy. Adresse identisch sind.