Aufgabe 3: Mehrstufige Caches

In der Vorlesung wurde der MIPS R4000 Prozessor als Beispiel für mehrstufige Pufferspeicher vorgestellt (Folie 2.55). Der primäre Cache ist virtuell indiziert mit physikalischen Tags, der sekundäre Cache ist rein physikalisch. Beide arbeiten mit write-back/write-allocate.

a) Wenn ein Miss im primären Cache ein Zurückschreiben einer modifizierten Cacheline notwendig macht, wohin sollen die Daten geschrieben werden?

Aufgabe 4: Adressraumlayout

a) Erklären Sie die Möglichkeiten des Adressraumlayouts bei virtuell und physikalisch indizierten Caches.

b) Diskutieren Sie die dynamische Vergabe von Anfangsadressen von Segmenten bei Ihnen bekannten Cachearten.

Lösung Aufgabe 3:

a) Die Möglichkeiten die modifizierten Daten zurückzuschreiben sind in den sekundären Cache oder in den Hauptspeicher. Sie sollten in den sekundären Cache zurückgeschrieben werden, da es weder notwendig, noch wünschenswert ist die Daten direkt in den Hauptspeicher zu schreiben.

Der primäre Cache ist immer (beim R4000) ein Subset des sekundären. Deshalb enthält der sekundäre Cache bereits eine Kopie der primären Cachezeile. Ein Miss in diesem während des write-back der primären Cachezeile ist deshalb unmöglich.

Eine Schreibzugriff auf den sekundären Cache ist deutlich schneller, als auf den Hauptspeicher

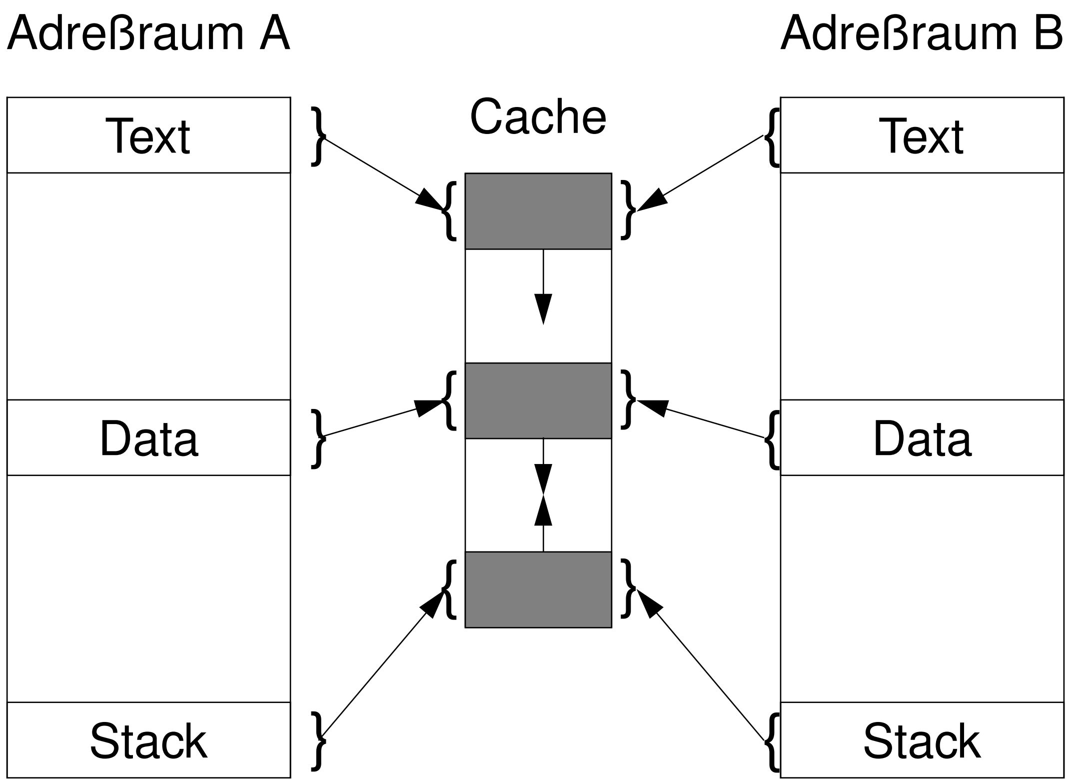
Der Prozess wird in näherer Zukunft wahrscheinlich noch mal auf die zu ersetzenden Daten zugreifen. Auch deshalb macht eine Pufferung in der zweiten Ebene Sinn.

Lösung Aufgabe 4:

a) Virtuelle Caches: Vergabe der Anfangsadressen für Text, Data/BSS und Stacksegmente, fest bzw dynamisch, (2.58 - 2.63)
Physikalische Caches: Page Coloring (2.64).

b) Wie wir gesehen haben, kann man ein Überlappen der Data-, Text- und Stacksegmente durch geschickte Wahl der Segmentsanfangsadressen unterbinden.
Betrachtet werden:
Virtuelle Caches mit Prozessschlüsseln,
virtuelle Caches mit physikalischen Tags ,
rein virtuelle Caches und
physikalische Caches.

Virtuelle Caches mit Prozessschlüsseln bzw. *virtuelle Caches mit physikalischen Tags* sind dafür gedacht Daten mehrerer Prozesse nebeneinander im Cache halten zu können. Da aber die Anfangsadressen für die einzelnen Segmente wiederum gleich sind, ergibt sich für das Mapping folgende Situation:



Der sich dabei ergebenden Verschwendung von Cachezeilen kann man durch eine *dynamische Vergabe der Anfangsadressen* begegnen.
Zur Compilezeit: Linker wählt zufällig ein Set von Anfangsadressen beim Binden.
Zur Ausführungszeit: Nur möglich, wenn keinerlei hardcodierte Adressen im Programm vorkommen.
Rein virtuelle Caches können während eines Kontextwechsels den Prozesskontext nicht “behalten”. Deshalb lohnt der Mehraufwand des Anfangsadressenlayouts nicht.
Rein physikalische Caches haben auch keinen Vorteil durch diese Methodik, weil sie eben mit physikalischen Adressen indiziert sind und die virtuellen ausser acht lassen.