Übung 12

Freitag, 19. Februar 2021 12:14

Aufgabe 2: Assembler-Code lesen

Gegeben sei folgende NASM-Funktion (Parameter stehen in rdi und rsi, der Rückgabewert steht in rax):

```
1               global function
2
3 function:     mov rax, rsi
4               add rax, rdi
5               sub rax, 1
6
7 .loop:        cmp rax, rsi
8               jbe .end
9
10              mov r8b, [rsi]
11              mov r9b, [rax]
12              mov [rax], r8b
13              mov [rsi], r9b
14
15              inc rsi
16              dec rax
17              jmp .loop
18
19 .end:        ret
```

win18_t array[] = {1,2,3,4,5,6,7,8};
function(8, array);

1. {8,2,3,4,5,6,7,1}
2. Adresse der mittleren Elements des Arrays wenn ungerade Anzahl an Elementen, sonst das letzte Element der ersten Hälfte des Arrays
3. {8,7,6,5,4,3,2,1}
4. Dreht Reihenfolge Array um

Aufgabe 1: MESI-Protokoll

Beantworten Sie folgende Fragen, geben Sie ggf. Ihre Quellen an.

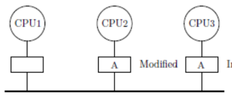
a) Was ist unter Cache Konsistenz zu verstehen?

b) Was ist unter Cache Kohärenz zu verstehen?

c) Beschreiben Sie kurz *in eigenen Worten* die möglichen Zustände im MESI-Protokoll.

d) Warum darf es im MESI-Protokoll den Zustand *Shared Modified* nicht geben?

Nehmen Sie an, die Prozessoren CPU1, CPU2 und CPU3 greifen auf einen gemeinsamen Speicher zu. Außerdem hat jede CPU einen eigenen lokalen Cache.
Wie in der Grafik ersichtlich liegt das Datum A im lokalen Cache von CPU2 und CPU3 vor. Nehmen Sie an, auf dieses Datum bezogen ist CPU2 im Zustand *Modified* und CPU3 im Zustand *Invalid*.



Zeigen Sie grafisch und erläutern Sie kurz, wie sich die Zustände aufgrund der folgenden Befehlsfolge verändern:

1. CPU3 schreibt Datum A zu A'
2. CPU1 liest A'
3. CPU1 schreibt A' zu A"

Sie finden einen LaTeX-Snippet für diese TikZ-Grafik im Anhang des Zettels.

a) -> alle Kopien eines Datums haben zu jeder Zeit den gleichen Wert

b) -> Es werde bis zu einem gewissen Grad Inkonsistenzen zugelassen
-> invalide Werte sind genau vor dem Zugriff (lesen/schreiben) zu aktualisieren
-> dadurch wird nur der neueste Inhalt einer Variable gelesen

c) (Exclusive) Modified -> Cache ist der einzige der aktuelle Inhalt eines Datums hat (ist als modifiziert)
-> andere Caches haben wenn dann invalide veraltete Kopien

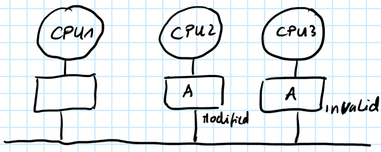
Exclusive Unmodified -> Cache hat als einziger Kopie eines Datums geladen
-> wurde noch nicht verändert/ stimmt mit Datum im Hauptspeicher überein

Shared Unmodified -> mehrere Caches haben Kopie eines Datums geladen
-> wurde noch irgendwo verändert

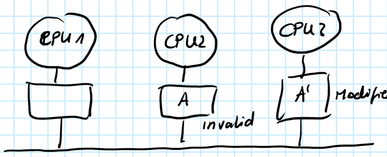
Invalid -> ein anderer Cache hat Datum verändert, deshalb ist eigene Kopie ungültig

d) - würde bedeuten, das mehrere veränderte Kopien gleichzeitig unterwegs sind
- genau dass soll ja verhindert werden
- man wüsste dann nicht welches davon man nun nimmt und in Hauptspeicher zurück-schreibt

I

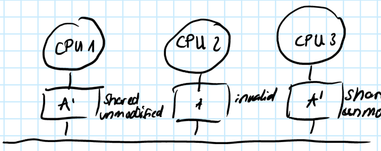


II



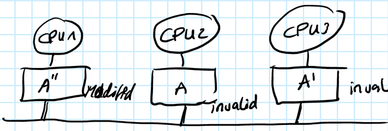
1. CPU3 schreibt Datum A zu A'
- CPU2 stoppt CPU3 kurz und schreibt A in Hauptspeicher
- CPU3 liest aus und verändert zu A'

III



2. CPU1 liest A'
- CPU3 stoppt CPU1 kurz und schreibt A' in Hauptspeicher
- CPU1 liest aus A'

IV



3. CPU1 schreibt A' zu A"
- CPU1 schreibt A' zu A"
- CPU3 snooped das und ändert zu invalid