## Gruppe 15 | Tobias Schoch, Luis Nothvogel

## Simulation wurde auf dem HTWG Container ausgeführt

## 20.1

Two-level page table: 2.

Ein Register für das Page Table Directory und ein 2tes Register für den Page table entry

Three-level page table: 3.

Ein Register für das Page Table Directory. Ein zweites Register für den richtigen Teil des PT und dann das 3 Register für den eigentlichen PTE

Mehrere Register da man 1 Register pro Level benötigt, um die jeweilige Adresse zu speichern. Denn es ist sicher durchaus praktisch diese Werte in einem Register zu haben, falls man diese noch einmal benötigt. Beim TLB ist es wahrscheinlich sehr praktisch, da wenn man in der gleichen Adresse nochmal unterwegs hat man diese schon vorher in einem Register und kann dann schneller drauf zugreifen.

## 20.2 Zusatz

```
lu851not@ct-bsys-ss20-15:~/z-drive/5.Semester/BSYS/Chap20$ ./paging-multilevel-translate.py -s 15
```

```
PDBR: 123  (decimal) [This means the page directory is held in this page]
```

Valid:

```
Virtual Address 016a: Translates To What Physical Address (And Fetches what Value)? Or Fault?
```

1) 0x016a -> 00000 (0) 01011 (11) 01010
   Zuerst wird die virtuelle Adresse in Binär übersetzt. Dann werden die einzelnen Teile zugeordnet. Also welches davon die PDE, PTE und der Offset ist. In unserem Beispiel nimmt der Offset 5 Bits und die VPN nimmt 10 Bits ein. Hier muss beachtet werden, dass wenn die zahl zu klein ist mit 0 aufgefüllt werden muss.
2) `page 123: 94 d8 ae 87 d5 f9 e7 e3 b7 d7 a1 c2 e6 86 7f c3 c8 9d 99 93 d3 7f ab dc 92 d1 a4 e0 cc c5 a3 b2`
   Wir schauen in Page 123 da dort der Page Directory ist (siehe Screenshot von Simulator oben). Hier schauen wir uns dann die „0" Stelle an. Diese kommt von unsere zuvor übersetzten virtuellen Adresse. Die erste 5er binär Zahl, das PDE.
3) 0x94 -> 1(valid)|0010100 (20)
   Daraus bekommen wir die oben gezeigte Adresse. Diese wird auch wieder in binär umgewandelt. Hier wieder wichtig zu beachten es müssen 7 Bits für die PFN sein und 1 Bit für das Valitbit da sein. Falls nicht muss wieder mit 0 aufgefüllt werden. Wir übersetzen die PFN in dezimal, um zu wissen in welcher Page unser PTE ist.
4) `page  20: 7f 7f 7f 7f 7f 7f 7f 7f 81 7f 7f de 7f 7f 7f 7f cf 7f 7f 91 7f ba 7f 7f 7f 7f 7f 89 7f 7f 7f 7f`
   Dann schauen wir in Page 20 an der 11 stelle nach. Diese Zahl erhalten wir durch unsere vorher übersetzte PTE Nummer.

5) 0xde -> 1(valid)|1011110 (94)
   Daraus bekommen wir die oben gezeigte Adresse. Diese wird wieder unter den gleichen Regeln, wie bei 3, übersetzt und ggfs. angepasst.

6) Physical Adress: 1011110 (94) 01010 (10) -> 0xbca
   Nun haben wir endlich die physikalische Adresse. Diese setzt sich aus unserer obigen neuen Adresse und unserem Offset zusammen. Nun gilt es nur noch zu schauen welcher Wert dort steht.
7) `page  94: 1e 00 1b 15 1a 1e 07 0a 15 08 08 07 1a 1b 18 06 1c 0f 07 0e 0d 02 11 0b 11 1c 14 0c 1a 1e 0f 1d`
   Hierfür schauen wir bei Page 94 an Eintrag 10 nach. Die 94 erhalten wir, wenn wir die Übersetzung von binär zu dezimal des PTE machen.
8) Value: 08

```
Virtual Address 016a:
  --> pde index:0x0 [decimal 0] pde contents:0x94 (valid 1, pfn 0x14 [decimal 20])
    --> pte index:0xb [decimal 11] pte contents:0xde (valid 1, pfn 0x5e [decimal 94])
      --> Translates to Physical Address 0xbca --> Value: 08
```

Invalid:

```
Virtual Address 3f62: Translates To What Physical Address (And Fetches what Value)? Or Fault?
```

1) 0x3f62 -> 01111 (15) 11011 (27) 00010
2) `page 123: 94 d8 ae 87 d5 f9 e7 e3 b7 d7 a1 c2 e6 86 7f c3 c8 9d 99 93 d3 7f ab dc 92 d1 a4 e0 cc c5 a3 b2`
3) 0xc3 -> 1(valid)|1000011 (67)
4) `page  67: 7f 7f 7f 7f 7f da 7f 7f 7f 7f 7f 7f 8d 7f 7f 8a 7f 7f 9a 7f 7f 7f 7f 7f 7f 7f 7f b6 7f 7f`
5) 0x7f -> 0(valid)|1111111 (127) -> fault
   Als Unterschied zu oben ist hier die erhalten Bit Zahl zu klein. Daher mussten wir hier eine 0 auffüllen und dadurch erhalten wir das valid Bit 0. Damit ist die Adresse nicht valid.

```
Virtual Address 3f62:
  --> pde index:0xf [decimal 15] pde contents:0xc3 (valid 1, pfn 0x43 [decimal 67])
    --> pte index:0x1b [decimal 27] pte contents:0x7f (valid 0, pfn 0x7f [decimal 127])
      --> Fault (page table entry not valid)
```

## 20.3

Ich gehe hier von den TLB als Memory Cache aus. Ein Cache muss klein sein und seine Größe effizient nutzen. Damit egal an welcher Stelle der Cache geaccesed wird, man immer gleich schnell an das gewünschte Ergebnis kommt. Durch Speicherreferenzen wird das ganze wahrscheinlich in mehr TLB Miss resultieren. Denn am Anfang ist der TLB ja noch leer, d.h es kann nur TLB misses geben. Danach wenn nicht die richtige Referenz gecached ist, resultiert dies auch nur in TLB misses. Es wird also langsamer. Außerdem muss dann immer erst in der Referenz noch geschaut werden ob das gewünschte drinsteht, das heißt ein klarer mehr

Aufwand. Dazu kommt noch dass die Spatial Locality verletzt wird, denn es kann ja sein dass zuerst eine Referenz von A im Cache ist und nebendran eine Referenz von D daneben ist. Außerdem wird dadurch die Idee des TLB ignoriert. Denn dieser wurde ja gerade deswegen eingeführt damit man diese langsamen Speicherreferenzen umgeht.