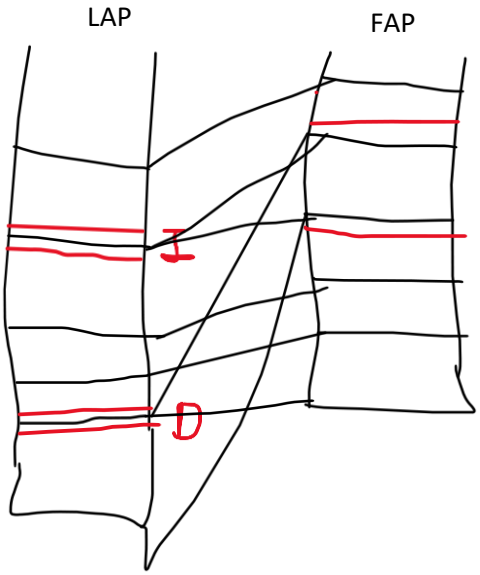
IOS - cvičení 2

- **Určete a zdůvodněte počet TLB miss, které mohou v nejhorším případě nastat při zpracování dosud nenačtené instrukce o velikosti: 4B, která čte 4B dat z paměti (uvažujme stránky o velikosti 4kiB)**



Může se stát, že I i data D jsou na rozhraní stránek

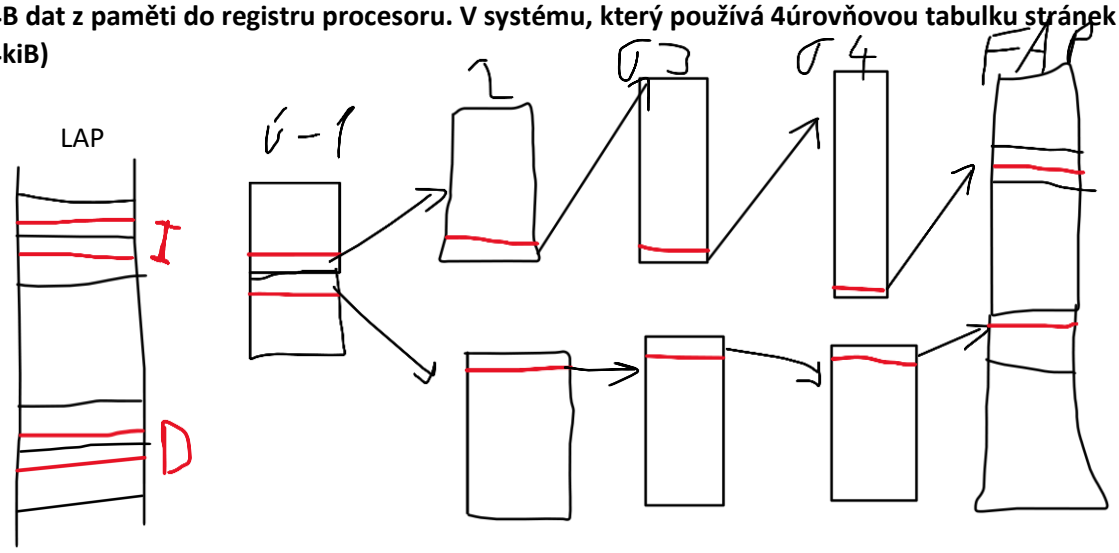
Pro I a data může tím pádem dojít k 2 přístupů do TLB a tím pádem k 2 TBL miss

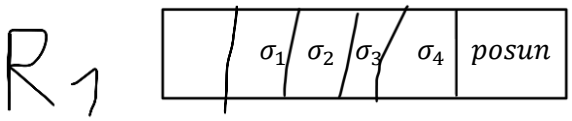
Celkově 4 TLB miss

- Instrukce navazují bezprostředně po provedení předchozí instrukce, která byla v paměti (LAP) před ní.
  - Stánky obsahující instrukci I_p jsou v TLB (nebude TLB miss)
  - I může zasahovat do jedné další stránky (max 1 TLB miss po instrukci)
  - Celkem 1+2=3

- Instrukce provádí kopii 4B dat v rámci LAP.
  - U instrukce max 2 TLB miss
  - U dat:
    - 2 TLB miss pro načtení
    - 2 TLB miss pro zápis
    - Protože obě místa mohou být na rozhraní stránek

- **Jaký je maximální počet přístupů do RAM při provádění dosud nenačtené instrukce I o velikosti 4B, která čte 4B dat z paměti do registru procesoru. V systému, který používá 4úrovňovou tabulku stránek (velikost stránky 4kiB)**
-





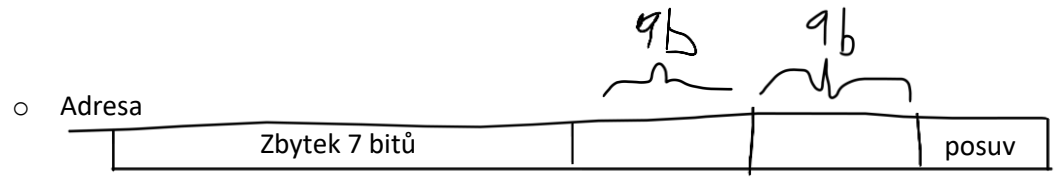
Sigma 1 až 4 = 1

R2 je stejné ale s PI

PI 1 až 4 = 0

- Instrukce I i data D mohou být na rozhraní stránek
  - V obou případech je třeba číst obsah 2 různých rámců FAP
  - Celkem 4
- Překlad stránek pro I (analogicky pro data D) může být na konci a začátku dvou tabulek 4úrovně
  - Celkem 2 čtení (krát 2)
- Pro třetí a druhou úroveň obdobně
  - Celkem 2+2 čtení (krát 2)
- V první úrovni čtu z jednoho rámce 2 řádky najednou
  - Celkem 1 čtení (krát 2)
- Celkem 7+7+4=18

- **Jaký maximální počet úrovní TS pro hypotetickou architekturu s 64bitovou fyzickou adresou, 4kiB stránkami, kde jedna dílčí tabulka stránek libovolné úrovně bude mít velikost jedné stránky. V každé položce TS bude 10 řídicích bitů.**



  - Máme 4kiB stránky 4kiB = 2^{12}B a tedy potřebujeme 12b na posun
  - Zbývá 52b na implementaci stránkování
  - Jedna položka každé tabulky stránek bude potřebovat 52bitů na adresu začátku rámce (posledních 12 bitů adresy je 0) a 10b na příznaky Celkem 62 bitů zaokrouhlíme na celé „B“ (64 bitů = 8 bajtů)
  - Stránka obsahuje $2^{12}/2^3=2^9$ řádků (9 bitů adresy)
  - Celkově máme 52/9 zaokrouhleno dolů = 5 úrovní tabulky stránek
  - Zbylých 7 bitů se nepoužívá

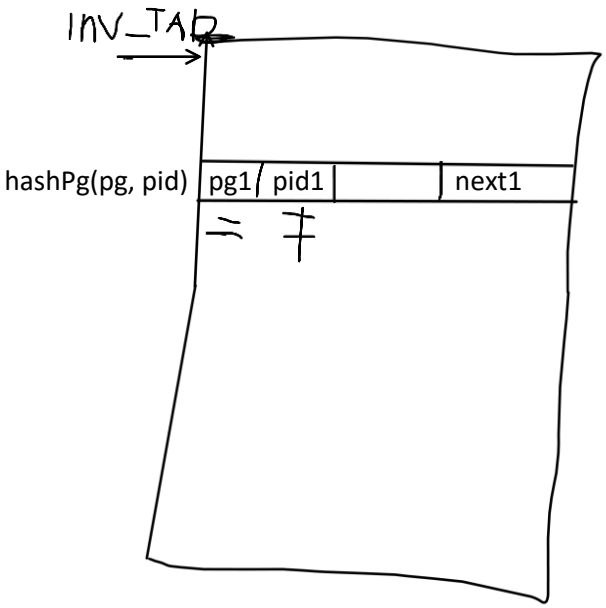
- **Uvažujme semafory s operacemi lock a unlock a následující fragmenty kódu:**
    - **P1:**
        - //Běžná sekce
        - lock(S1)
        - lock(S2)
        - //VS
        - unlock(S1)
        - unlock(S2)
        - //Běžná sekce
    - **P2:**
        - …
        - lock(S2)
        - lock(S1)
        - …
        - unlock(S1)
        - unlock(S2)
    - **Zapište Cofmanovy podmínky deadlocku a rozhodněte, jestli platí/neplatí/nelze platnost určit**

- Procesy používají zdroje s výlučným přístupem
    - Ano, platí, používáme semafory
- Proces vrací zdroje sám, po ukončení jejich používání
    - Ano, platí, protože předpokládáme standardní semafory
- Proces může žádat o zdroj i když už nějaký vlastní
    - Ano, platí, protože P1 zamyká S2 přesto že vlastní S1
- Vzniká cyklická závislost procesů a zdrojů
    - U P1 je S2 závislá na S1
    - U P2 je S1 závislá na S2

- **Implementujte pomocí semaforů a pomocných čítačů monitor s 1 čekací podmínkou a operací notify(), bez priority vstupu**

```
struct sMonitor{
   semaphore sIn, sCond;
   int cCond;
}
void Init(*m) {
   m.sIn = sem_init(1);
   m.sCond = sem_init(0);
   m.cCond = 0;
}
void Enter(*m) {
   lock(m.sIn);
}
void Leave(*m) {
   unlock(m.sIn);
}
void Wait(*m) {
   m.cCond++;
   unlock(m.sIn);
   lock(m.sCond);
   lock(m.sIn);
}
void Notify(*m) {
   if(m.cCond > 0) {
      m.cCond--;
      unlock(m.sCond);
   }
}
```

- **Implementujte v pseudokódu překlad logické adresy na adresu rámců přes inverzní tabulku stránek kombinovanou s hashovaním. Řádky TS budou instance datové struktury inv_tab_item, do které doplňte položky dle potřeby. Předpokládejme, že kolizní seznam překladových položek pro hashování jsou přímo v tabulce, nikoliv externě**
    - **Pro hashování máme funkci hashPg(…).**
    - **Na začátek tabulky odkazuje ukazatel inv_tab**



- pg_num page2frame (pg, pid, *inv_tab, …) {
  int fr=hashPg(pg, pid);
  while(pg != inv_tab[fr].pg || pid != inv_tab[fr].pid) {
  fr = inv_tab[fr].next;
  if(fr==-1) return -1;
  }
  return fr;