

$$\begin{aligned}
 \text{Nr. access} \dots &= 10^8 \times 0,1 + (\% \text{ Reads} \times \text{Read Miss Penalty} + \% \text{ Writes} \times \\
 &\quad \times \text{Write Miss Penalty}) + 10^8 \times 0,3 + 0,3 \times \underbrace{\text{Write Hit Penalty}}_{\text{Writes}} \\
 &= 1
 \end{aligned}$$

$$\begin{aligned}
 \text{Read Miss Penalty} &= \text{Write Miss Penalty} = \\
 &= \underbrace{2 \text{ BUS Reads}}_{\text{Allocate}} + 0,35 \cdot \underbrace{2 \text{ BUS Writes}}_{\text{Update for Main Memory}}
 \end{aligned}$$

CVRS 13

28.05.2019

$$\begin{aligned}
 b) \text{Nr. access} &= \underbrace{10^8 \text{ words/second}}_{\text{Frequency of memory references}} \times \underbrace{0,1}_{\text{Miss Rate}} \times (0,3 \text{ Write Miss Penalty} + \\
 &\quad + 0,7 \text{ Read Miss Penalty}) + 10^8 \times 0,9 \times 0,3 \text{ Write Hit Penalty}
 \end{aligned}$$

$$\text{Read Miss Penalty} = 2 \text{ BUS Reads} \quad (\text{non main updated})$$

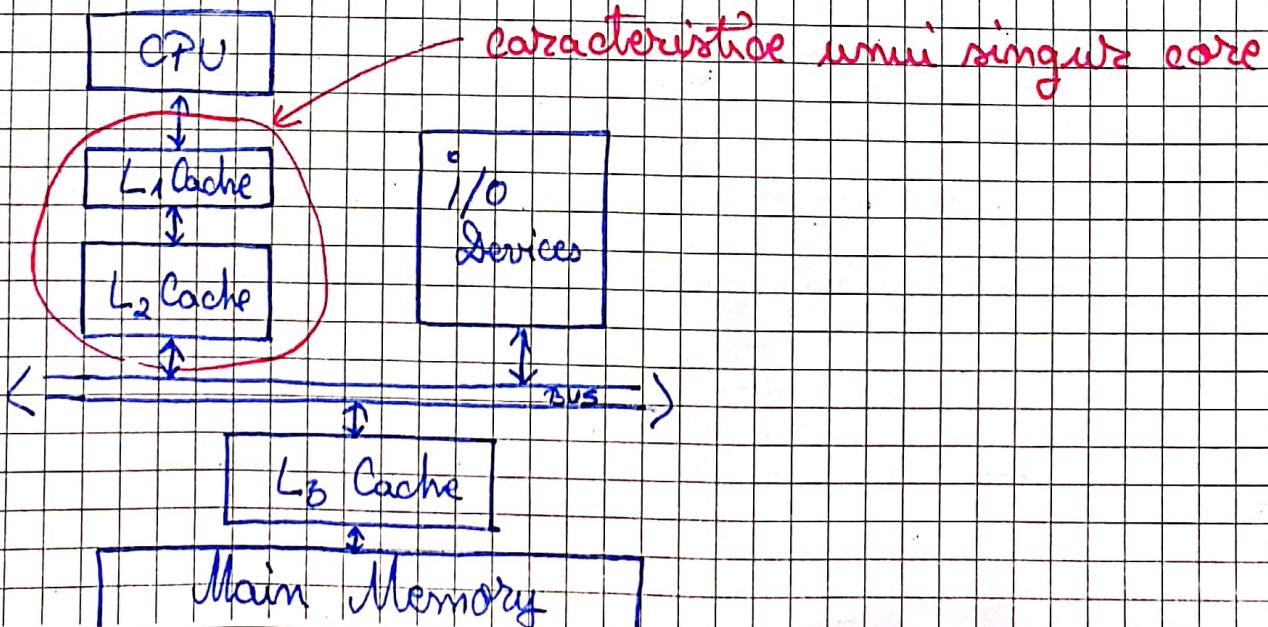
$$\begin{aligned}
 \text{Write Miss Penalty} &= 2 \text{ BUS Reads} + 1 \text{ BUS Write} = \\
 &= 3 \text{ BUS Accesses}
 \end{aligned}$$

$$\text{Write Hit Penalty} = 1 \text{ BUS Write} = 1 \text{ BUS Access}$$

$$\begin{aligned}
 \text{fazadar: Nr. access: } &10^8 \cdot 0,1 (0,3 \cdot 3 + 0,7 \cdot 2) + 10^8 \cdot 0,9 \cdot 0,3 \cdot 1 = \\
 &= 10^7 \cdot 2,3 + 2,7 \cdot 10^7 = 5 \cdot 10^7
 \end{aligned}$$

$$\% \text{ Bus Used} = \frac{10^7 \cdot 5}{10^8} = \frac{5}{100} = 5\%$$

3.9 Reducing miss rate with multi-level caches



Example: Se dă un sistem de memorie cu 2 nivele de cache

unde CPI ideal = 1,0 clock cycle.

Clock rate = 4 GHz

MM access time = 100 ns (includând Miss Penalty în MM)

Miss Rate per instruction = 2% (în L1)

De către ori va fi mai rapid procesorul dacă adăugăm L2 cache cu un timp de acces de 5 ns

și pt. hit-wri și pt. miss-wri să fie suficient de mare să reducă Miss Rate la MM la 0,5% din cazuri.

Doar cu L1:

$$\text{CPU time original} = \text{YC} \times (\text{CPI ideal} + \underbrace{\text{Memory accesses per instruction} \times \text{Miss Rate}}_{\text{Misses per instruction}} \times \text{Miss Penalty}) \times \text{CCT}$$

$$\text{Misses per instruction} = 0,2$$

$$\text{CPU time original} = \text{YC} \times (1 + 0,02 \times 400) \times 0,25 \text{ ms} = 2,25 \text{ ms} \times \text{YC}$$

$$\text{CCT} = \frac{1}{4 \cdot 10^9 \text{ Hz}} = 0,25 \text{ ns} \quad (= \frac{1}{\text{Clock Rate}})$$

$$\text{Miss Penalty} = \left\lceil \frac{100 \text{ ns}}{0,25 \text{ ns}} \right\rceil = 400 \text{ c.c. (clock cycles)}$$

↳ găsește info în MM.

L1 și L2:

$$\text{CPU time}_{\text{L2}} = \text{YC} \times [\text{CPI ideal} + (0,02 - 0,005) \times 20 + \text{info nu numerește în L1 și hit-wri în L2}] \times 0,25 \text{ ms}$$

$$+ 0,005 (400 \text{ c.c.} + 20 \text{ c.c.})] \times 0,25 \text{ ms} = 0,85 \text{ ms} \times \text{YC}$$

↑
info nu numerește nici în L1, nici în L2

$$\text{Miss Penalty}_{\text{L2}} = \left\lceil \frac{5 \text{ ns}}{0,25 \text{ ns}} \right\rceil = 20 \text{ c.c.}$$

↳ miss penalty când nu găsește în L1 și găsește în L2

$$\text{fără: } \frac{\text{CPU time orig}}{\text{CPU time 2L}} = \frac{2,25}{0,85} = 2,64$$

3.10. Virtual Memory Mechanism

Virtual Machines, probleme pe care trebuie implementate la:

- 1) Protect processes from one another
- 2) Share the limited memory among processes (VMs)

Pentru asta, programatorii folosesc overlays, dar acum se folosesc și memorii virtuale ce folosesc un spațiu de memorie virtuală (SMV) ce urmărește să fie mapeat pe adresele fizice.

SMV se fragmentează în:

Segments (variable size)

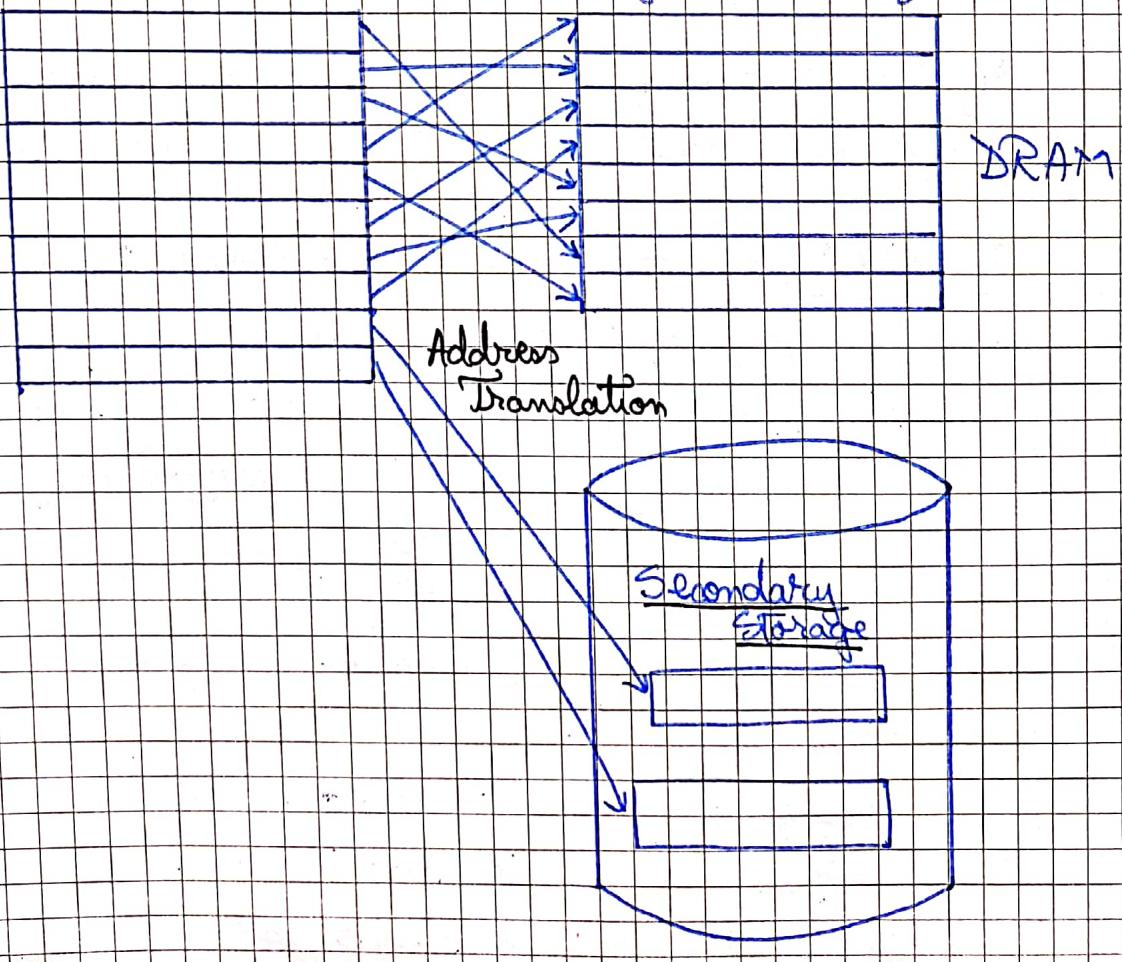
Pages (fixed size)

↑ cel care ne interesează mai mult

> Mecanismul de memorie virtuală paginat:

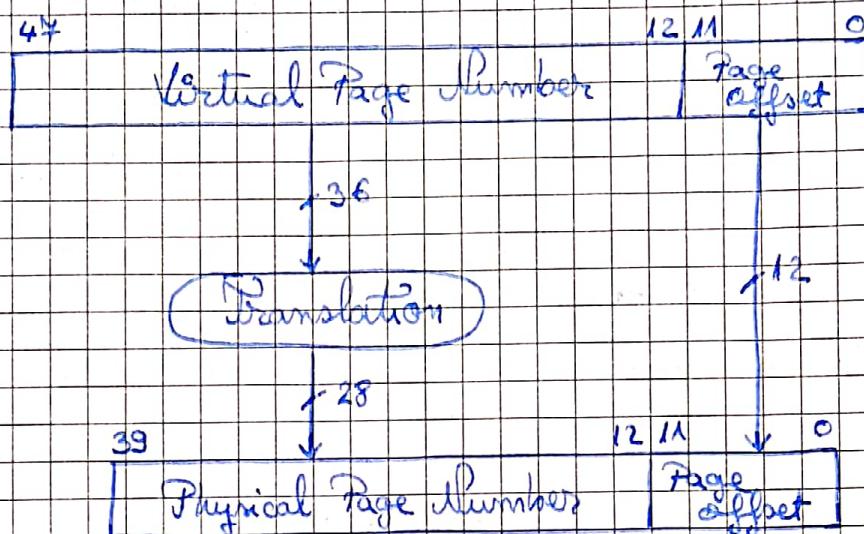
Virtual Memory

Physical Memory



• Exemplu pt. Address Translation pe ARM V8:

64 Bit Address word
16 MSB are unused

$$\begin{array}{r} 64- \\ 16 \\ \hline 48- \\ 12 \\ \hline 26 \end{array}$$


Page size = $2^{12} B = 4 KiB$ ($B = \text{Byte}$) ← pe 12 biti exprimă adresa unui byte

Physical memory size = $2^{40} B = 1 TiB = 2^{28} \text{ pages} = 256 MiPages$

Virtual memory size = $2^{48} B = 256 TiB = 2^{36} \text{ pages} = 64 GiPages$

Paginiile virtuale se mapează în memoria fizică dacă cînd procesul este activ (pentru CPU la un moment dat)

Denumiri: page → granule Memory Management Unit

page fault → MMU exception

↳ cînd pagina nu e în MMU trebuie adusă de pe disc prim mecanismul de swapping

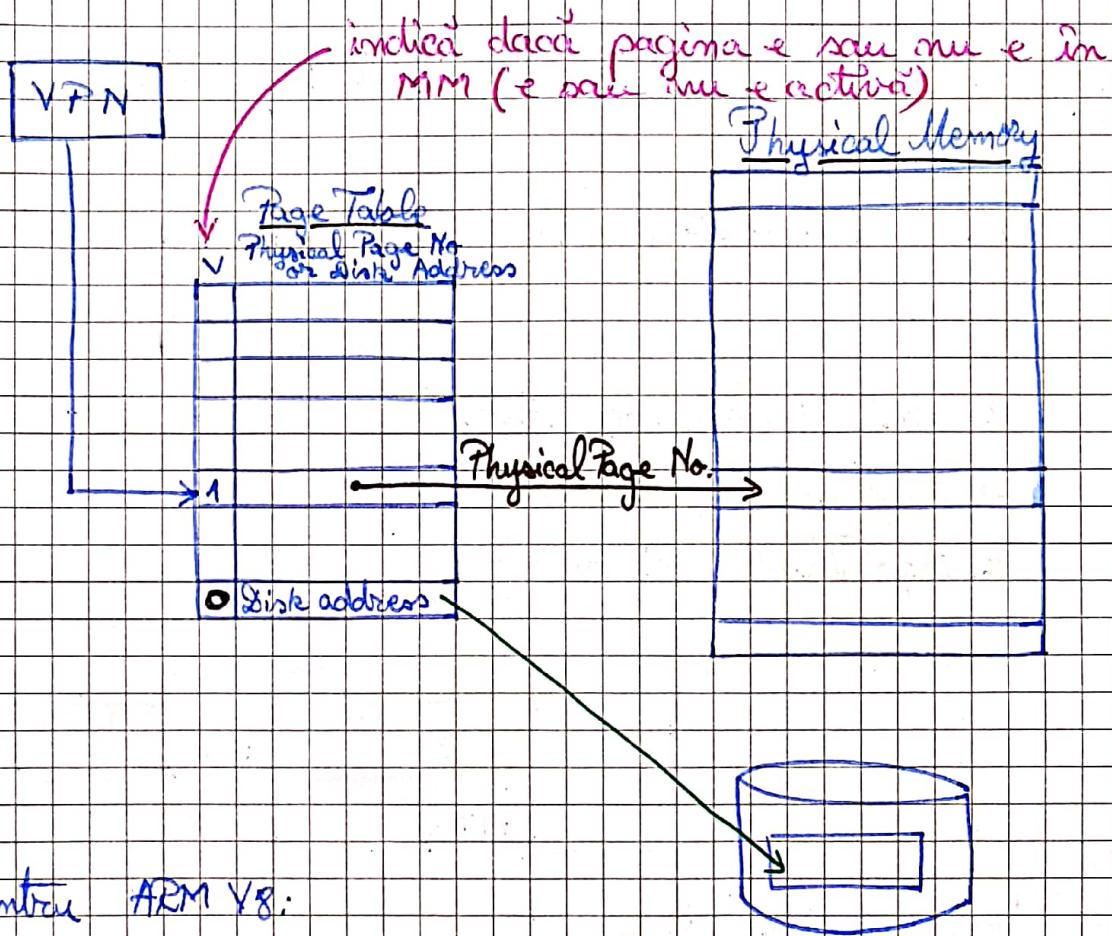
Problema lăsată în considerare pt. construirea unui MMU: Misses cost a lot! (Secondary storage is 10000x slower than MM). Rezolvare:

① relatively big pages: $4 KiB \div 64 KiB$

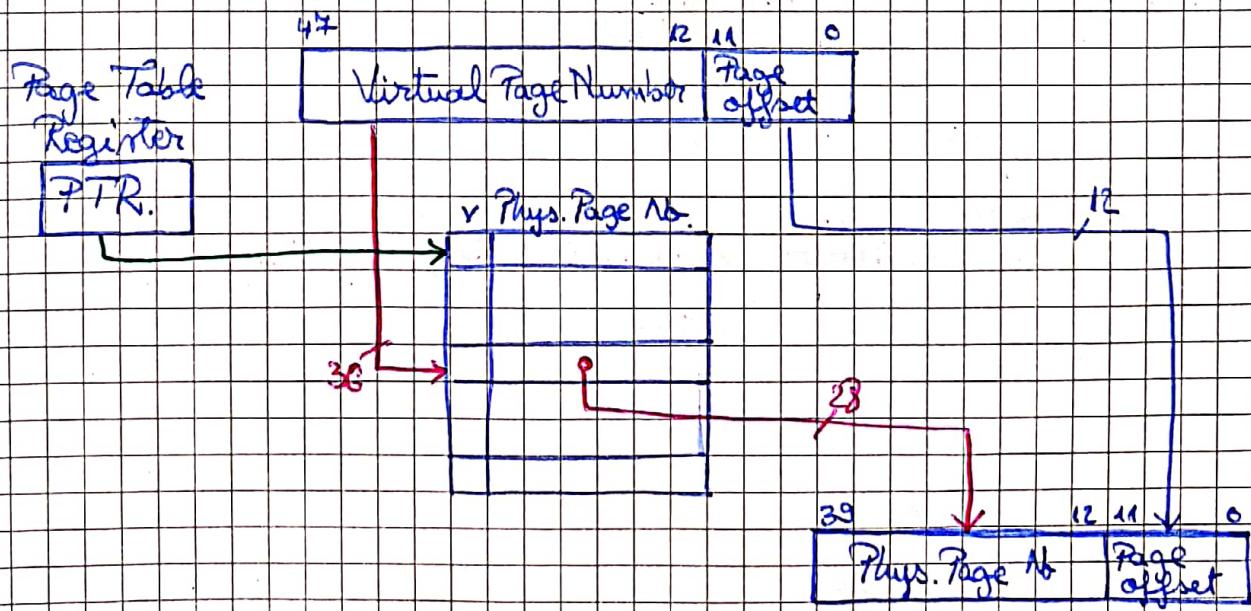
Servers → $32 KiB \div 64 KiB$

Embedded → $1 KiB \div 4 KiB$

- ② $\text{WIT} \leftarrow \text{impossibile} \rightarrow \text{Folosim doar WB}$
- ③ Replacement is decided in software (nu mai folosește metoda random) — approximate LRU
- ④ Iată că permită să cauți tag-uri \rightarrow
 \Rightarrow Folosim doar full-associativity mapping cu Page Tables



♥ Pentru ARM V8:



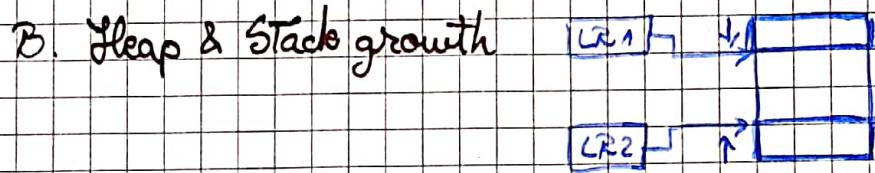
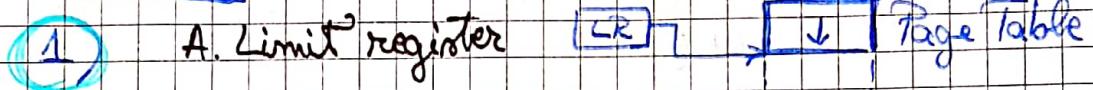
Problems: ① Page Table Size

ARM - 1 PT Entry = 64 bits = $2^{32} B$

$$2^{36} \times 2^3 = 2^{39} = 0.5 TiB$$

② Page Table Read (Page Table is placed in the Physical Memory)

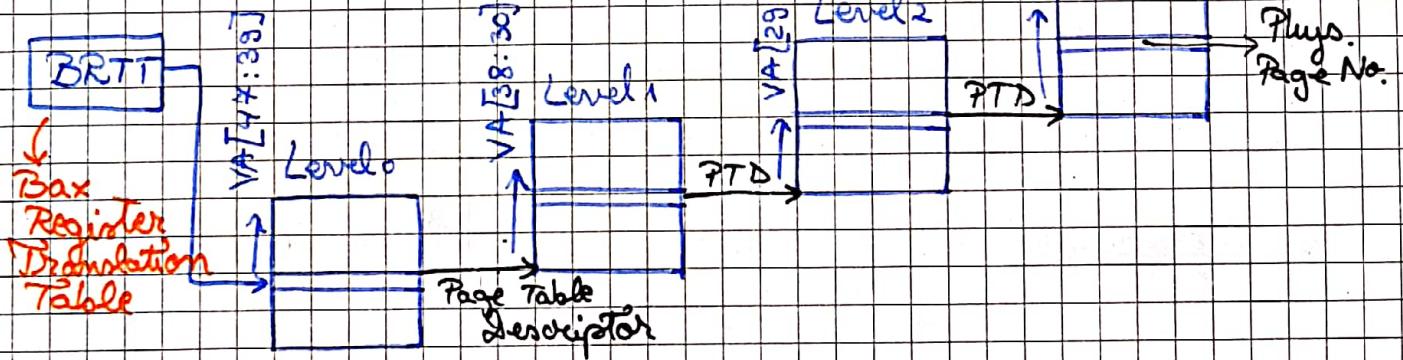
Solutions:



C. Hash (inverse page table)

D. Page The Page Table

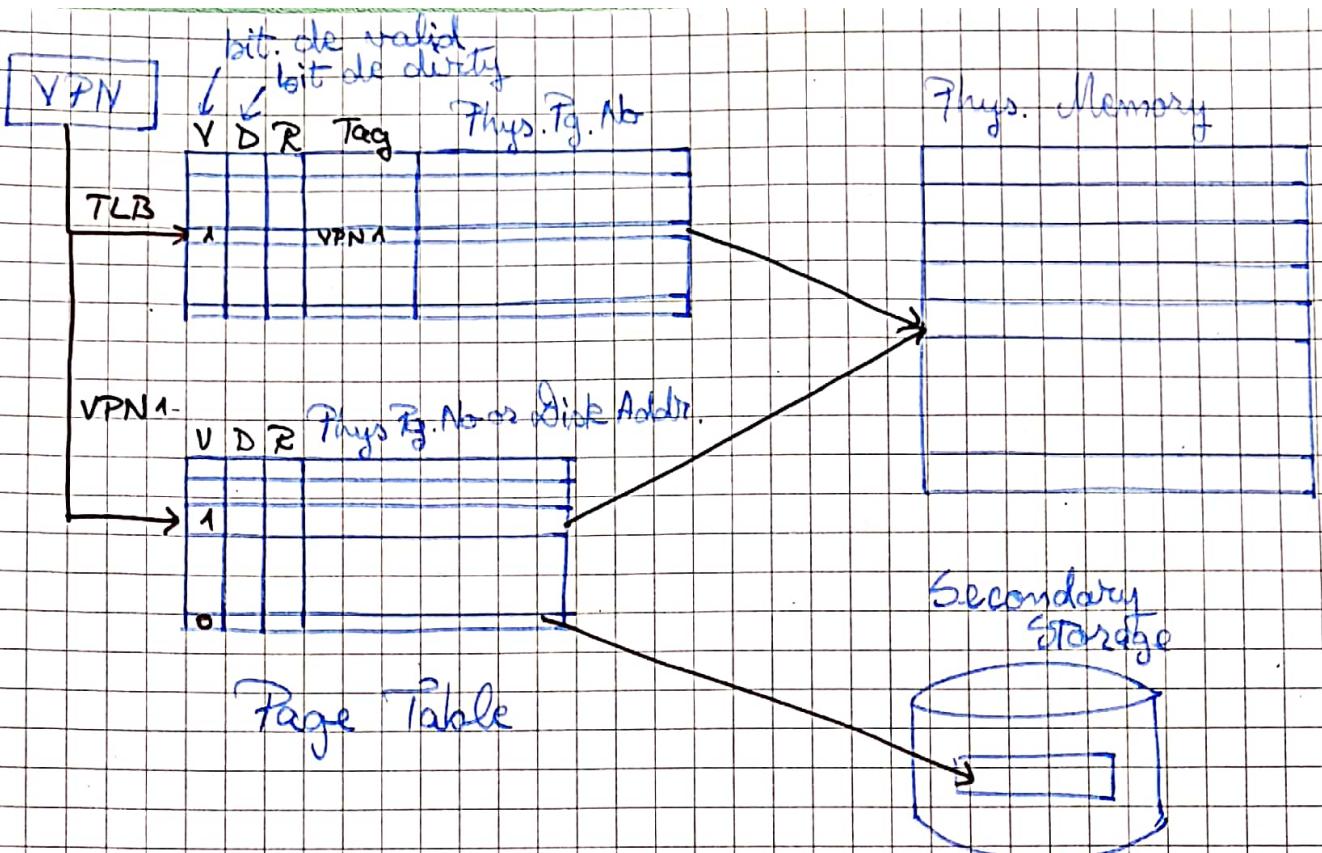
E. Page Table Hierarchy



$$2^9 \times 2^2 B = 2^{12} B = 4 KiB$$

② Translation Lookaside Buffer (TLB)

- Cache the Page Table



Fast MATH

