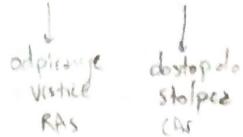


PREDPOMNILNIK

Pri meri 16-page mode način dostopa DRAM 2Gb DDR2 je povpreč. čas dostopa:

$$t_c = (40 + 16 \cdot 1,75) / 16 = 3,75 \text{ ns}$$



Razlogi: hitrost

2 dodajajoči predp. L1, L2, L3 izmenjujemo verjetnost pogreške in imamo manj dostopov do GP

SRAM, na
CPE

8.5.1 Osnovni principi delovanja PP

(206) Ustvarjava se podkadrožica GP. Zanemira se na lokalnost naslovov. Bez te te je PP neuporaben.

PP je lahko homogen ali uhomogen (ločen PP za vrstice in operande)

verjetnost zaledka... H

↳ daje naslov do katerega dostopamo v PP

$$H = \frac{N_p}{N} = \frac{N_p}{N_g + N_p}$$

obizajno $H > 0.95$ (zelo dobro)

N ... št. vseh dostopov

N_p ... št. informacije pri dostopu v PP

N_g ... št. pogrešek (informacije pri dostopu v PP)

$$t_a = t_{ap} + (1-H) t_{ag}$$

t_a ... povpreč. čas dostopa

t_{ap} ... čas dostopa do PP → redno potrebno

t_{ag} ... čas dostopa do GP (Audi pri pogreški. Zakaj? da naredi predo upe optok pričelo)

predponiščeni blok - vez sosednjih besed, ki se ob pogreški prenese v PP

če je širina podatkovnih puti do PP enaka širini bloka, je t_{ap} Vendar ni vedno tako, zato:

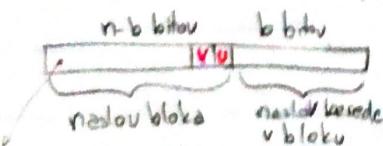
$$t_a = t_{ap} + (1-H) t_{ag}$$

t_{ap} ... čas za prenos bloka
pogrešna beseda
mrežna penality
10-100 up

→ PP je lahko neviden (transparenten). -danes besed je ne

→ Brzje (za V/I napravo nevidno. - zato imenuje uncacheable predmeti)

→ PP mora uselovati naslove, ki povedo, katerim besedam GP ustrezata njegova trenutna vrednost



→ PP zato sestavljen je iz 2 delov: kontrolni in pomnilniški

next page

- razdeljen na enake enake velikosti
4 besed BLOKI / PP-vrstice
poravnati: 8-100 pp-besed

V - veljavni bit: ce je V=0 → useljava neveljavna. V takem bloku ni nikoli zadetek. Postopek je ce je V=1

V - umazani bit: na 0 se postavi do prenosu bloka v PP. Ce pride do pisarnje v blok, se postavi na 1.

v PP bloku je 2^b sosednjih pomnilniških besed

Kontrolni del PP vsebuje informacijo, ki ENOLICNO opisuje vrak bloka.

najmanj) ~~vrak~~ naslov bloka, včasih tudi

V in U bit (prednje stran)

naslov pove, kateri del GP je preslikan v PP oz. je v bloku.

Naslovi vsek blokov skupaj določajo podmnožico GP, ki je trenutno v PP.

→ Če pride do zgrešitev, se mora blok v PP zamenjati. Zamenjava bloka. Če se je vsebina trenutnega bloka spremenila, se mora te prvo prenesti v GP. Če vsebina bloka ni veljavna, se ne zapisi? Zato uporabljam $V=1$. (Roznamo tudi druge metode kot je Veljavnost, gledamo \rightarrow Te najbolj popularne)

→ Danes imamo veliko blokov. Uporabljajo se omejitve pri preslikavi vsebine GP v PP. Ver mora biti primerjava zgornjih $n-b$ bitov velikosti bloka. Alka neka beseda GP se ne more shraniti v poljuben blok PP, temveč v ustrejajo določeno število (majhno) blokov.

Glede na strogoost ločimo:

- asociativne
- set asociativne
- direktrne

8.5.2. PP glede na omejitve pri preslikavi

(22) potrebujemo koliko primerjalnikov blokov, kot je blokov v PP → asociativni PP

Ver potrebujemo ogromno st. komparatorjev, se običajno ne uporablja. So le v primerih, ko imamo do velikih 100 blokov. Npr. pri preslikovanju Nandernih naslovov v fizične.

Set asociativni PP: vpeljemo omejitve

PP je razdeljen na $S = 2^s$ setov, vsak pa je majhen asoc. pp. Št blokov v setu $E = 2^e$... stopnja asociativnosti

Velikost PP: $M_p = S \cdot E = 2^{s+e}$ oz. $M = S \cdot E \cdot B = 2^{s+e+b}$ velikost asoc. pomnilnika v setu obsegno 1-16
pomnilniških besed pomnilnik besed naslov

→ Razlike! - sedaj imamo omejitev pri preslikovanju naslovov.

že vsko besedo je določeno, v kateri set se lahko presliká.

To določajo naslov seta biti → Predpomnilniški INDEX (cache index)

Znotraj seta so bloki ekvivalentni.

Naslov A_i se lahko presliká v enega izmed blokov S_i : Vsi naslovi, pri katerih delujejo $n-b$ bitnega st.

$$S_i = A_i (b \cdot n - 1) \pmod{2^s}$$

$A_i (b \cdot n - 1) \leq S_i < A_i (b \cdot n)$ da isti ostanek, preslikamo v isti set.

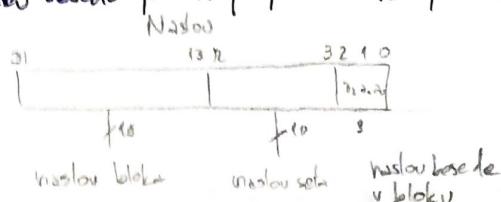
I. skrajnosti ce je stopnja asoc. == št. blokov, dobimo čisti asociativni PP ($s=0$)

↳ ponavljajoča se enota
severno bloku.

II. skrajnosti ce pri enaki velikosti M vzamemo stopnjo asociativnosti $E=1$,

dobimo direktni PP. Potrebujemo samo 1 primerjalnik Preprostnejši. Manj logičnih vredj.

~ za vsako besedo je vsaj ena dolžina, v katerega od blokov (blok je enak setu) se lahko presliko.



to dolžine biti $\geq_0, \geq_{M1}, \dots, \geq_{M+3}$
kjer je 2^k sedaj št. blokov.

Manjša stopnja asoc. E \propto manjša cena PP

- II- \propto manjša verjetnost zadevka

↳ zna se zgoditi, da je potrebno preslikati nek blok v set, ki ima več blokov zadevence, med tem ko je v ostalih setih še prostor.

Pri asociativnem PP tega ni.

Predpomnilniško pravilo: verj. zgrešitve direktne PP velikosti M

je enaka

verj. zgrešitve $\frac{1}{M}$ PP asociativnega PP s stopnjo asoc. = 2 in
velikostjo $M/2$

8.5.3 Kako velik naj biti blok?

- (295) S povzročenjem velikosti bloka se povečuje izkoriscenje prostorske lokalnosti.
 Ker so bloki večji, jih je manj in zato se zmanjšujejo pageji in zahteva lokalnost.
Izkorica - prenos prve besede (vsebitka dolžji od ostalih)

8.5.4 Kateri blok naj se zamenja pri agregativi?

Pri direktnih samo 1 možnost: Zamenjuje se blok, pri kateri je prislo do agregativa.
 22 assoc. in set assoc.: V katerem se prostika beseda,

1. Naključna strategija:

Vsi bloki imajo enako verj., da bodo zamenjeni.

2. LRU:

least recently used

- izkoriscenje zavorne lokalnosti
- prenese/zamenja se blok, do katerega najdlje ni bil nareden dostop

Pri večjih stopnjah asoc. → Naključna strategija

Zamenjalna strategija igra vejzo vlogo pri manjših pp, ker obstaja več možnosti za zamenjavo

8.5.5 Pisalje

(298) Pisalje v PP - tečeje je zaletek.

1. Pisalje skozi (write through):

pise se v PP in GP. Vsebnina ostaja enaka

+ enostavnejše za realizacijo

+ skladnost/kokarentnost podatkov v GP in PP

2. Pisalje nazaj (write back):

pise se le v PP. Vsebnina v bloku je lahko drugačna od vsebine v GP. Le-to je treba prenesti v GP.

+ pisalje s hitrostjo PP

Tato se uporablja umazani bit (dirty bit).

+ pri več pisanjih v isti blok je potreben le 1 pisalje v GP

zato se uporablja umazani bit (dirty bit).

(celotna širina podatkovnih poti in DRAM page nade nazaj)

Pri pisalnih/agregativnih rečitri:

- Pisalna zamenjava (write allocate): - bolj razširjena
 - tgr. se obravnavajo enako kot lokalno
 - v PP se prenese nov blok, ki mu sledi dostop do pp

Pisalne naokrog (writearound):

- pisalna zgr. ne povzroči zamenjave bloka v PP
- ↳ blok se spremeni samo v GP

8.5.6 Viste zgrešitev

300)

3 osnovni vrzoli:

1. Obvezne zgrešitve: (viste zgrešitve/zgrešitve prvega praviza)

↳ ob prvem dostopu do vsebine nekega bloka tega ni več v PP

Velikost pp., asociativnost in zamjenjivo strategija vplivajo vpliva.

Vpliva velikost bloka - pri večjih blokih je dol. zgr. manj, ker se prenese več besed.

(vendar večji bloki lahko ponanijo povečano zgrešitveno koren)

St. zgr. bi bilo enako tudi pri ∞ velikem pp.

Risitev: povečanje bloka. Poneni tudi večjo zgrešitveno koren.

2. Velikostne zgrešitve:

Zaradi konkrete velikosti pp ne more vsebovati vseh blokov.

Prihaja do zamenjave blokov, ki so kmalu \rightarrow tem spet potrebeni.

Večji pp \rightarrow manj zamenjaj \rightarrow manj takih zgrešitev.

Risitev: večji PP (najenostavnježje)

3. Konfliktne zgrešitve:

↳ le pri set-assoc. in direktnih pp.

↳ Vdaj! ko je potrebno zamenjati blok, ki se bo kmalu spet uporabil.

do zamenjave je prišlo, ker se oba bloka prestikata v isti set.

Risitev: večja stopnja asociativnosti (lahko pa poveča dostop do pp)

8.5.7 (ne) homogenost

homogen - en sam pp

nehomogen - razdeljen v ukazne in operanduega

8.5.8 vpliv pp na hitrost delovanja CPE

čas za izvrševanje programa (CPE) zdele:

- čas, ko CPE deluje

- čas, ko CPE dela na posnutek zaradi zgrešitev v pp

$$\text{CPE}_{\text{zas}} = (\text{CPE}_{\text{periode izvrševanje}} + \text{CPE}_{\text{zgrešitev}}) \cdot t_{\text{cpe}}$$

perioda CPE zde

$$\text{CPE}_{\text{periode izvrševanje}} = N \cdot (1-H) \cdot \text{zgreš. koten}$$

Et vsak
št. down dostopov porpr. verjetnost
zgrešitev
(pričakanje in pisanje)

Na prvem nivoju je CPE razdeljen na ukazni in operandni del.

Povprečna verjetnost zgr. $1-H$ je rezultat zgrešitev v oba:

$$(1-H) = (1-H_{\text{ukazi}}) + M_I \cdot (1-H_{\text{operandi}})$$

(povprečno št.
posnutek vseh
dostopov
na ukaz)

→ posimistrično: ne upoštevana, da se v primusu
istoznačne zgr. v dveh pp izvršuje
ne sešteva redno

N... št. dostopov

I... št. izvršenih ukazov

Namreč N imamo ponavadi I.

$$\text{CPE}_{\text{zas}} = I \cdot (\text{CPI}_{\text{zgrešitev}} + (1-H) \cdot \text{zgrešitvena koten}) \cdot t_{\text{cpe}}$$

kjer

$$\text{CPI}_{\text{zgrešitev}} = \frac{\text{CPE}_{\text{periode izvrševanje}}}{I}$$

↳ clocks per instruction

povpr. št. vrstic porad., ki so pri danem prog. potrebne za izvršitev 1 ukaza (prez ustrezačna izkajanje na posn.)

$$\text{CPI}_{\text{zgrešitev}} = \text{CPI}_{\text{zgrešitev}} + (1-H) \cdot \text{zgrešitvena koten}$$

pri manjših vrednostih CPI brez zgrešitev pride do veliko večjih upočasitev.

Toda frekvenca ima vpliv. CPE ima visjo frekvenco ure pri enakih hitreih GP vsej po zgrešitveno kazen (le-ta se meri v st. UP-vrinih periodih)

Počasni DRAM-i : vzrok za veliko zgrešitveno kazen . z L2 PP želimo kazen zmanjšati
Imamo dvourojški PP.

Razlikujemo 2 vrsti verjetnosti zadevna / zgrešitev:

~ Lokalna verjetnost zgrešitev :

to je število: $\frac{\text{st. zgrešitev v PP}}{\text{st dostopov do tega PP}}$

za prvi nivo ... $1-H_{L1}$

za drugi nivo ... $1-H_{L2}$

~ Globalna verjetnost zgrešitev :

z st. zgrešitev v PP

za prvi nivo (L1) je lokalna verj. == globalna verj.

z st. vseh dostopov do PP

za drugi nivo (L2) je globalna: $(1-H_{L1})(1-H_{L2})$

za prvi nivo:

$$\text{CPE period} \cdot \text{časovje na ponu.} = N \cdot (1-H_{L1}) \cdot \text{zgrešitvena kazen}_{L1}$$

spremeni se:

$$\text{zgrešitvena kazen}_{L1} = t_{BL1} + (1-H_{L2}) \cdot \text{zgreš. kazen}_{L2}$$

če prenos bloka iz L2 v L1
pri zadetku v L2 <UP>

Dvečji del lokalnosti
izkoristi že 1. PP

za dvourojški PP:

$$\text{CPI} = \text{CPI}_{\text{brez zgrešitev}} + (1-H_{L1}) \cdot \text{zgrešitvena kazen}_{L1} =$$

$$= (\text{CPI}_{\text{brez zgrešitev}} + (1-H_{L1}) \cdot (t_{BL1} + (1-H_{L2}) \cdot \text{zgrešitvena kazen}_{L2}))$$

8.5.10 Načini za zmanjševanje zgrešitvene kogni

(310)

- vpliv za zmanjšanje / izgubo hitrosti:
- verjetnost zgrešitve → Restri:
 - zgrešitveni koren
 - Restri
 - ~ parametri vrstnega reda prenosilje besed v bloku
 - ~ z več nivoji PP

- ~ učenje PP
- ~ večja stopnja asoc.
- ~ pravilna izbiro velikosti bloka
- ~ dobra zamengovalna strategija

Že več opcij (kot zgornji 2):

(1) Vnaprejšnji prevzem bloka

pri prenosu bloka $k \rightarrow pp$, se prenese tudi blok $k+1$. Ta se shrani v **bralni bufer** (read buffer). Če CPE naslovni blok $k+1$ je to veliko hitreje, kot ōe bi moral v GP. Pri locirjenem ukazem in operandu en pp sta potrebna 2 b. itarnalnika.

Povečanje verjetnosti, da bo nro 2. blok ($\in RB$) rabljen → s povečanjem globine RB

(2) NEBLOKIRajoči PP (nonlocking cache)

ideja: CPE se do zgrešitv v ukazem ali operandu en pp ne ustvari, temveč špekulativno izvršuje druge ukaze (ne samo v običajnem vrstnem redu). Če se rezultati teh ukazov lahko koristijo porabijo, izgubite zaradi zamenjave bloka sploh ni več! Običajni PP se pri zgrešitvi ustavlja in ne omogoča novih dostopov (blokirajo)

↳ Potrebujemo drugačen PP

potrebujemo:
način delovanja: zadetek pod zgrešitvijo (hit under miss)

↳ kaj narediti, če pride do še ENE zgrešitve?

- CPE se ustavi in čaka na 1

- zadetek pod večkratno zgrešitvijo (hit under multiple miss / miss under miss)

↳ smiselno le, če povezova CPE ↔ GP in ogroža prenosilje več blokov hitrosti (ali pp L2)

↳ zapletje zgradbo logike za krmiljenje PP (komunikator PP oz. cache controller)

8.6 POMNILNIŠKO PREPLETANJE

(312)

Hitrost DRAMov je nizka in se počasi vira. Uporaba page mode pomaga (vendar nedovolj)

Preostane nam - povečanje št. bitov, ki se prenesejo naenkrat

↳ 2 načina:

① Sirse podatkovne poti do GP

Istogre je dostop do sestavljenih pomn. besed. (do 8, 16, 32 ali več sosednjih besed)
 Gоворимо о sirini pomnilnika (če je sirina pomn. = velikosti bloka) →
 Blok se lahko prenese v 1 dostopu

② pomnilniško prepletanje (memory interleaving)

pomnilnik je razdeljen na m samostojnih delov M_0, M_1, \dots, M_{m-1} imenovani: **MODULI**
m-kratno prepletanje

Modul - samostojen pomnilnik, deluje neodvisno od ostalih

Istočasno lahko poteka do m dostopov istočasno.

Vsako UP se poslje naslov enemu modulu, ti shranijo naslov in opravijo branje.

Po končanem branju re upaka UP prebrani podatek poslje v CPE.

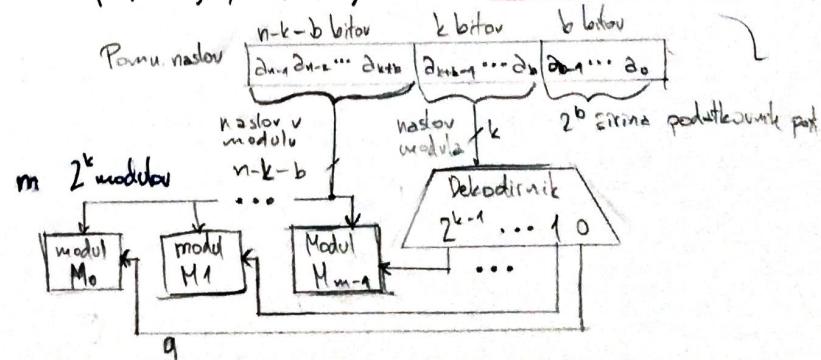
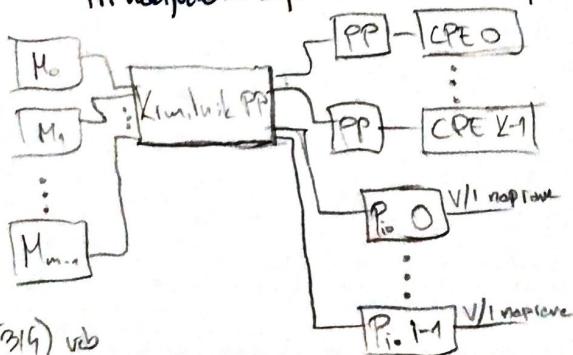
Pri pišanju se modulu poslje še podatek.

Krmilnik PP - ugotovi, na kateri modul se nanaša naslov, ki jih razlikujejo procesorji in
 ① vzpostavi vezbo med podatkovnimi potnimi procesorji in moduli.
 ② vzpostavi vezbo med podatkovnimi potnimi procesorji in moduli.

Če je naslovljeni modul trenutesko zaseden, krmilnik postavi, da zaključi početek ③
 Istočasni dostopi: **KONFLIKTNI DOSTOPI** (ki se jih morec izogniti)

početo: **spadajo prepletanje** - modul jedoločen s spodnjihi biti pomn. naslov.

Pri naključnem zap. naslovov modulom pri m-kratnem prepletanju je uporajmo možnih funkcij istočasnih dostopov.



8.7 UPORABA in DODELJEVANJE POMNILNIKA

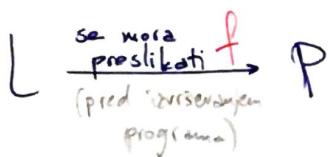
Lahko def. večdimensionalne podatkovne strukture (višji prog. jeziki)

obazi in operandi strogo ločeni;

L - logični naslovni prostor (programerjev prostor)

P - fizični naslovni prostor (zaposreduje besed 0, 1, ..., $2^n - 1$)

ividijo CPE in drugi procesorji



f - preslikovalna funkcija: v večini primerov skupni rezultat preslikav, ki te iteracijo v več korakih.
za programerja običajno nemanj.

4 koraki preslikave:

① **pisanje programa** - programer določi relativni položaj ukazov in podatkov znotraj programa / procedur.
Fizikalni ukaz in podatek \rightarrow na procedure, konstante in spremenljivke.

② **prevajanje** - prevajalnik ~ pretvori ime v dvojiske naslove.
~ določi, kje bodo podatki in kje ukazi.

Usplošno: rezultat prevajanja: množica ukazov in podatkovnih blokov, od katerih je vsak zaposredovan v posamezne besede.

Beseda je v nekem bloku - enolično določena z logičnim naslovom (vsebuje začetni naslov bloka in relativni naslov besede glede na začetek bloka)

③ povezovanje in nalačanje

- **POVEZOVALNIK** (linker) - poveže vse programme (in podprogramme) v celoto, ki jo vloži v pomnilnik.
- dodatni programi** - lahko razteza programer
 - potrebnost ugotoviti povezalnik (npr. procedure za računanje mat. funkcij, prog. za izvajanje V/I operacij, diagnosticni prog, ...)

Ko je nalačanje končano, je preslikava zaključena,
Vendar se na mnogih rač. med izvajanjem spreminja:

statično dodeljevanje (static allocation) - se ne spreminja

dinamično dodeljevanje (dynamic alloc.) - se spreminja

④ izvajanje programa

pri dinamičnem dodeljevanju imamo zaporedje preslikovalnih funkcij
naslovi (fizični) se od časa do časa spremenijo.

2 razloga za uporabo dinamičnega načina dodelj.:

I ob začetku programa je tekmo določiti velikost te-tega. **Rekurzija** - podati je potrebno max velikost
slabo: npr. prekoračitvi → napaka
večina časov pomnilnik neizkorističen

II multiprogramski način ! - kar kaže programov ta pomnilniški prostor
Dodeljevanje prostora vodi OS. Prestavlja programme poučni pomn.
več istosimno izvajajočih programov si deli pomnilniški prostor
V GP je samo "aktivni" del programa (trenutno)

Boljši izkoristek: programi se prenesejo v poljuben del prostora in po možnosti:
v več neveznih delov.

Pomnilnik se dodeljuje tako, da CPE zim manj stoji.

Pri dodeljevanju GP pomaga ideja o neskončno velikem navideznem pomnilniku

(31a)

9. NAVIDEZNI POMNILNIK

(Virtual memory) prostor v pomočnem pomnilniku. Skoraj vedno magnetni disk.

Veseli: prekrivanje - program razbit na dele programa ali prekrivki.

✓ GP naložene globinske sprem. in zacetek programa. Ko te opravi, klici iz PM drugi del, ...

NP danes - ekonomični razlogi, pravilno rešitev poticajske neodvisnosti, (razlogi) rešitev problema začete pomnilnika.

NP veseli - premalo GP (npr < 8K besed)

9.1 POMNILNIŠKA HIERARHIJA

↳ zap. pomn., kjer vsak pomn. komunicira s svojimi sosedoma

$$M_1, M_2, \dots, M_n$$

↳ vse informacije (vedno) $M_{i-1} \in M_{i+2}$

S pomn. hier. želimo doseg, da CPE vidi velik, cenen in pocasen M_n kot hiter, drag M_1 .

Nastav CPE daje, je nastav besede v M_n (najvišji nivo) ↳ močno zaradi lokalnosti pomn. dostopov.

• zas. dostopa do neke besede zdaj ni enak za vse nastave.

ekskluzivna verj. zadetka
$$h_i = \frac{N_i}{N}$$
 (ne vivojih $1, 2, \dots, i-1$ niz detektor na i pa)

lokalna verj. t. (zaznanje med st. zadetkov na vivoju i in st. dostopov do tega nivoja)

$$h_i = h_n$$

$$h_i = h_i - h_{i-1} \quad i=1, 2, \dots, n$$

$$\begin{aligned} \text{st. dost. } N &= N_1 + N_2 + \dots + N_n \\ \text{verjetnost zad. } \rightarrow h_1 &= \frac{N_1}{N} \\ h_2 &= \frac{N_1 + N_2}{N} \\ &\vdots \\ h_n &= \frac{N_1 + N_2 + \dots + N_n}{N} = 1 \end{aligned}$$

Pri registrirju v nav. pomn. so kazni zelo vecje kot pri PP.

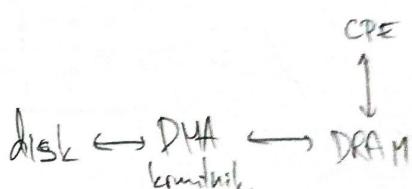
Rešujemo jih na drugačen način.

~ večji bloki za prenos

~ asociativna preslikava (vsakek blok se lahko preslika v poljubeni blok GP)

~ prog. zmanjšava blokov do registrirali (omogoča uporabo bolj zahtevenih algor. za izbiro bloka)

~ pisalne napot (zaradi počasnosti diska je pisanje skozi BV)



(32h) 9.2 PRESLIKOVANJE NAVIDEZNIH NASLOVON

za vsak nivo obstaja preslikovalna funkcija

↓ fāc NAV poun.

Naslov, ki jih daje CPE se vedno nenešo na najvišji nivo → **navidezni naslov**
(virtual addresses)

za del GP velja: nikoli ne pride do zgresitev.

GP - fizični pounnik (physical address)

pred vsakim pounniliškim dostopom je potreben preslikati NAV → FIZ

↳ tu imamo lahko zgresitev

preslikovanje: $A_f = f(A_n)$

fizični naslov → preslikovalna funkcija → navidezni naslov

↳ se med delovanjem spreminja (prilagaja se lastnosti programa, da je rež. zadevra) ČIM VEČJA

Pri NAV. Poun. pravimo tenu: **dinamično preslikovanje** (fui se spreminja s časom)

Preslikovalnik - naprava za dinamično preslikovanje naslovov

- enote za upravljanje s pounnikom (MMU - memo. management unit)

pri PP: - logika za preslikovanje naslovov

- logika za izmenjovanje blokov

pri NP: - logika za preslikovanje bl.

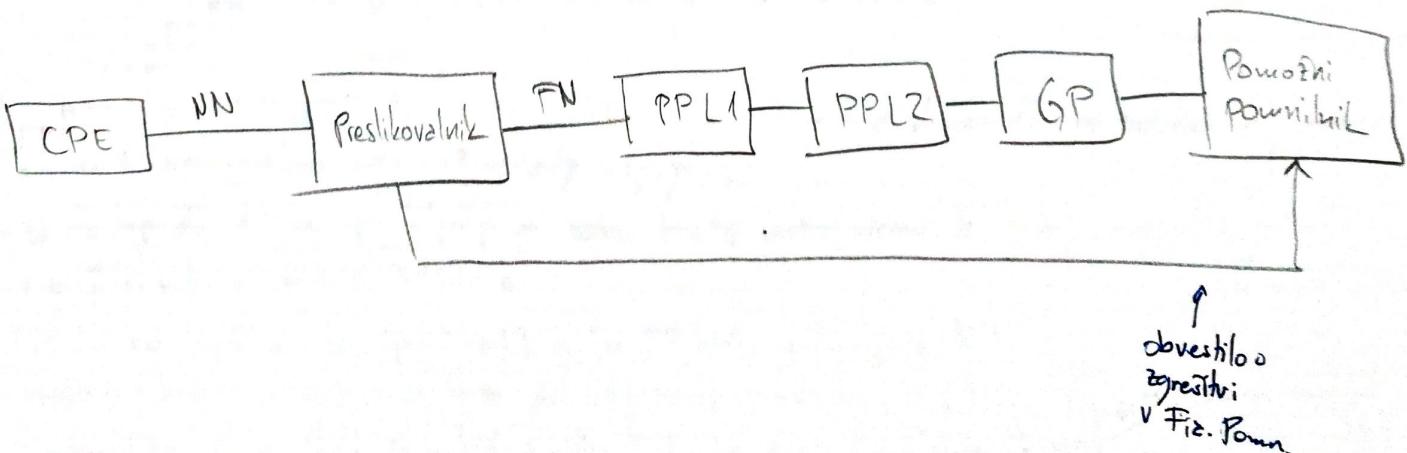
- izmenjovanje blokov: **PROGRAMSKO**

občajno: zgresitev

- sproži se prekinitev

- PSP postribi za prenos bloka v GP

Pri večini primerov se **NN** preslika v **FN**, kateri se uporablja za nastavljanje pp in GP



(326) Zakaj tako realitacije?

① Želimo možnost, da se doda več različnih NN lahko preslika v isti FN.

Ti naslove, alias / sinonimi.

↳ Tako lahko razl. programi uporabljajo iste procedure ali podtuke.

Če pp uporablja NN, je lahko ista fiz. beseda na večih mestih \rightarrow SLABO in

↳ če jo en prog. spremeni, drugi prog. tega ne vidijo

② I/O naprave: lažje jim je pri prenosih v/iz pomo. uporabljati FN.

(če bi uporabljale NN, bi ta slabost potrebovali preslikovanje FN \rightarrow NN)

zdržimo prednosti NN in FN.

Dostop do 1. nivoja PP se opravlja istočasno s preslikovanjem NN \rightarrow FN

(če preden je FN znan)

2 načina:

① Fizični indeks - fizični naslov bloka

naslov besede znova strani doloz spodnjih p bitov NN. (pri preslikavi se ne spremeni)

spodnji p bitov - spodni naslov besede v PP bloku

zgorajji biti - indeks, ki kaže na enega od setov PP ob koncu dostopa

↳ pp naredi dostop do vseh blokov v tem setu. Vsi naslovi vseh pp blokov se primarjo s FN, v katerem se je preslikal NN

Zadelek - če imamo pri katerem od blokov enakost. Sicer agresiv

Umejitev: velikost PP < stopnja asoc. \times velikost strani;

npr. stran 64B
stopnja asoc.: 8
32KB ukazi PP
32KB operandni OP

② Navigacijski indeks - fizični naslov bloka

- drugi dostopi (ker pri zadetkih ni potrebno preverjanje)

(če na 1. stopnji želimo večji PP ali večji stopnji asoc.)

- za dostop do PP se uporablja polog naslova besede znova strani še vekaj sosednjih višjih bitov,

- z višjimi biti se izbere/indeksira set.

ki so del NN.

- ob koncu naslova se (podobno!) naslov PP bloka primarje z FN.

- oključek (zaredi uporabe bitov, ki so del NN, lahko pride do ujega)

- agresiv se pregleda, če je blok s istim FN. Če prisoten v drugih setih \Rightarrow se ga tam naveže (vse enač-bakrat)

non-translatable področje. Programsko def. področje, kjer so NN vedno enaki FN.

Preglogi: (200) ?

Največ programsko izključitve preslikovanja.

↳ pomembno pri vključu rečunalnika!

(327)

9.3 VRSTE NAVIDEZNIH POMNILNIKOV

2 očnove vrsti realizacije NAV. P.: (na tiste, ki imajo:)

- bloki fiksne velikosti: STRANI (1KB, 8KB, 16KB, ...)

↳ podobne blokom v PP

→ drugačni

- bloki spremenljive velikosti: SEGMENTI (od 1 besede do zg. mgle - odvisno od rač.)

2^{16} - 2^{32} besed

(328)

	STRAN	SEGMENT
naslov	enodimensionalen	večdim. (št. segmenta in naslov znotraj segm.)
viduo za programir?	ne	lakko
zamenjava bloka	prosto (vsi bloki so enake veliki)	zapletena (v GP je treba najti pravilen rezon prostor spremenljive dolžine)
izguba pom. prostora	notranja fragmentacija (neuporabljene deli strani)	zunanja fragmentacija (neuporabljene deli GP)
večkorivost prenosov na disk	Dz. (velikost strani se lahko prilagodi lastnostim diska)	Ne vedno. (Pri majhnih segmentih je prenos nevečkoriv)

Uporablja se kombinacije: Segment, ki je razdeljen na celo število strani.

↳ odpravi problem pri zamenjovanju blokov

→ prostor v GP ne rablji več tvezov
→ v GP ne rablji bit celoten segment

9.3.1 ostranjevanje - najstarejši oblik NP

NAV

ideja: ROM. pomoč. je razdeljen na bloke enake velikosti, ki jih pravimo strani (pages)

Vse strani skupaj: NAVIDEZNI POMNINKER.

velikost: potenca št. 2 (4KB-16KB)

GP je razdeljen na enako velike bloke: okviri strani (page frames) — FIZ

Vsaka STRAN je mogoče prenesti v poljuben okvir

pomočna beseda: 8b

preslikovalna fun.: s pomočjo tabeli strani (page table)

Vsaki strani NP pripada v tabeli eno polje. Vnaprej določena fiksna velikost tabele (max)

Imamo lahko VEC TABEL. Vsak program zaseda določeno št. strani.

Redko je program velik toliko kot mnogokratnik velikosti strani.

V povprečju je zadnja stran itkorisena polovično.

notranja fragmentacija (narašča + velikost strani)

preslikovanje:

~ spodnjih p bitov: določa naslov strani znotraj strani → pri preslikovanju nespremenjen

~ zgorajih n-p bitov: št. strani → se preslika (skoti tabelo)

Polja v tabeli strani se imenujejo: deskriptorji strani (page descriptor)

Vsebujejo: V, P, RWK, C, FN (št. okvira)

To parametru

- ① **Veljavni bit V** - kadar je 1, so parametri v deskriptorju VELJAVNI
 Valid
 0, stran ni definirana, parametri nimajo pomena
 (običajno pri 1. dostopu do nekoga naslova strani)
Kreiranje strani: definiranje parametrov v deskriptorju
 Loparji OS

↳ itogniti ^{mnz} se je mogoče, ob tačetku izvajanja programa vidijo se gen. vse vsegora strani.
 ↳ to ni dobro (veliko prog. ned izvajajočih ve uporabi vseh strani)

- ② **Prisotni bit P** - kadar je 1, je stran v enem od okvirjev v GP. Parameter FN pove v Present
 ↳ to ustreza tačetku prve pp in take strani imenujeno **katerem.**
AKTIVNE STRANI

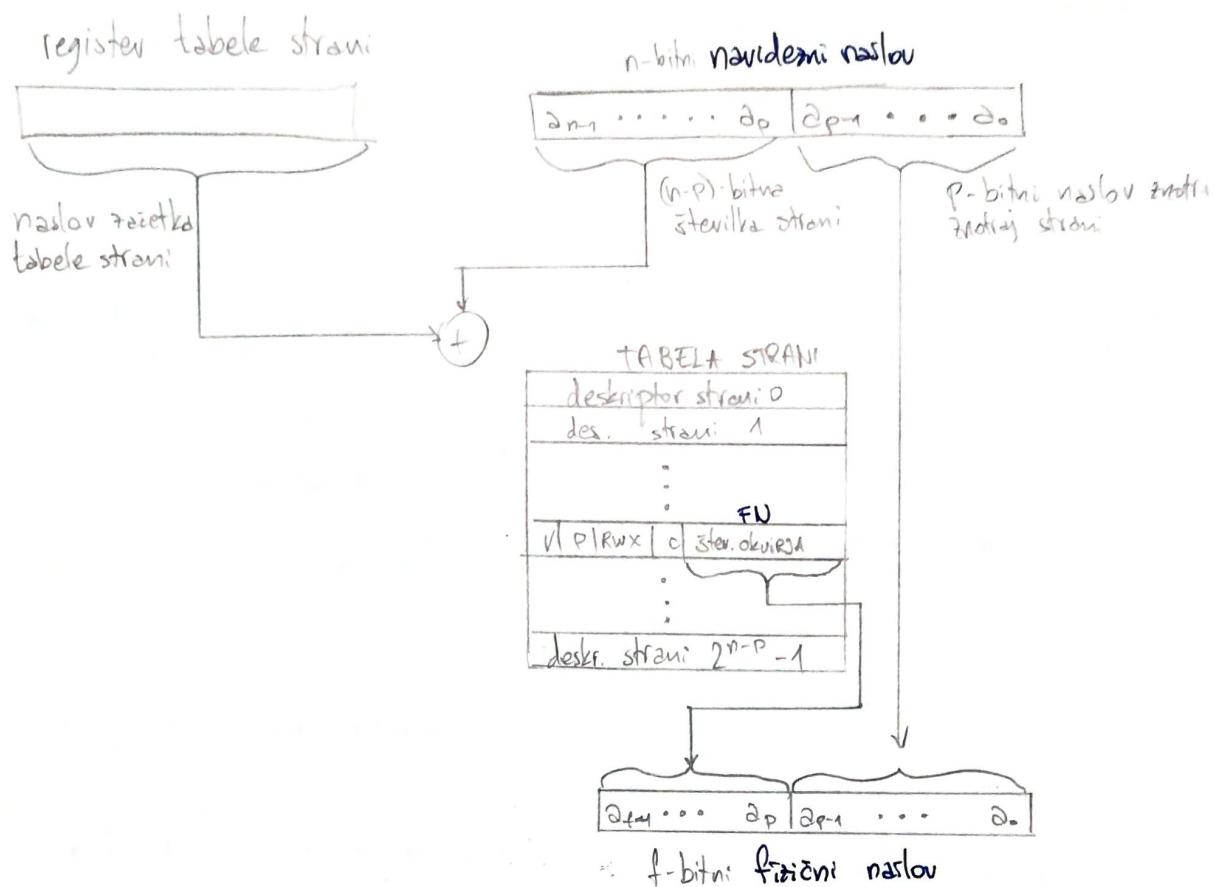
0, strani ni v GP - zgreditev. Pri ostranjanju je to: **NAPAKA STRANI**
 (page fault)
 ↳ FN ni veljaven, stran je potrebno prenesti iz diska v enega od okvirjev GP. → to naredi servisni prog. Pasti.

- ③ **Tačitni ključ RWX** - zasečite pomicniku. Določena vrsta dostopa (branje/pisanje)
 Read/write/execute nedovoljenem dostopu se P bit ignorira. **izvajanje**

- ④ **Umarani bit C** - ko se stran prenese v GP, gre na 1.
 Change
 Če pride do pisanja na katerikoli naslov te strani → 1
 Označuje spremembro vsebine. Če je C=1 in je stran potrebno zamenjati, se stran prenese nazaj na disk.

- ⑤ **Stevilka okvirja FN** - podaja št. okvirja v kateri je stran. $\underline{FN \times 2^P \dots}$ naslov okvirja v Frame number G.P.
 ↳ veljaven le, če je $(P=1)$
 ↳ FN $\times 2^P$ se prišteje naslov tuotraj strani → **PRIZNI NASLOV** → dobimo

Če ima naslov okvirje spodnjih p bitov enakih 0, so ti naslovni biti fiz. naslova vedno enaki bitom bitom NAV. naslova.



to do sedaj je bilo:

linearno preslikovanje - navideni prostor je linearen / raven (flat) \neq "floufjer" / vezdimenzionalen

Ljubot da ne budi navidenega pommilnika

segmentacija

delitev prostora je za uporabnike nevidna

~ vezmivojsko preslikovanje:

- zmanjšuje prostor
- če je napak strani veliko, lahko pride do naslednje napake strani, če preden je končan prenos projinge.

↳ takšne se postavijo v vrsto

↳ prenetovanje (thrashing) - premetavanje / premazuje strani.

GP $\xleftarrow{\text{informacije}}$ Pomozni P.

lakšo pride pri velikih vrtih NAV.P.

Mehanizem, ki preprečuje prenetovanje: (izbira pravilnih parametrov, ki vplivajo na verjetnost napake strani)

~ velikost strani

} fiksna

~ velikost GP

~ strategije za temenje strani

} del OS

~ st. naenkrat izvajajočih programov (stopnja multiprog.)

} Med izkajanjem se SPREMINJATA

OSTRANJEVANJE

⊕ proprostost

⊖ ne najboljša zaščita pomm.

⊖ več programov si ve more deliti isti prostor (pogoj za to je NP, pri katerem ima vsebuha blokov svoj pomen).

Delitev pommilnika na strani je mehanizmo.
Tega ostr. NiMA. Ne upošteva modularnosti

sledijo

NP, ki upoštevajo zgradbo programov : segmentacija → (segmentation)

(333) 9.3.2 segmentacija

6. Na prostor, ki ima pomen prostor razdeljen na segmente: segmentacija

Razlike od strani:

- segmenti so ravno veliki
- vsebuje segmenta ima z program svoj pomen
- v GP nimata nujna podobnega okvirnega prostora: (ker se segm. med izvajanjem resita)
- (- vsak segm. se lahko premese v poljuben prostor GP)

Programski modul - vsebine / podatkovne zgradbe

↳ daber prog. je napisan izvirke jasno def. modulov, ki imajo def. vezjo med sabo.

Naslov besede: naslov segm. + relativni naslov besede v seg.

Na vsak segm. gledeamo kateri je svoj lasten pomen prostor, vezivam od ostalih (z njimi je povezan)

Vedimentionalni NP - vsak seg. je svoja dim.

Simbolica / čista segmentacija - vsak seg. ima svoje simbolično ime, s katerim je znan OS.
(ali programerju)

Segmenti med prevajanjem prog.:

- UKAZNI (ukazi procedur prevajalnika, velikost segm. je fiksna)
- PODATKOVNI
 - ~ tekst izvornega prog., se hrani ta izpisovanje
 - ~ tabela simbolov
 - ~ tabela konstant
 - ~ sintaktično drevo (sintaktična anal. prog.)
 - ~ sklad; program uporablja za kljuc svojih PROCEDUR

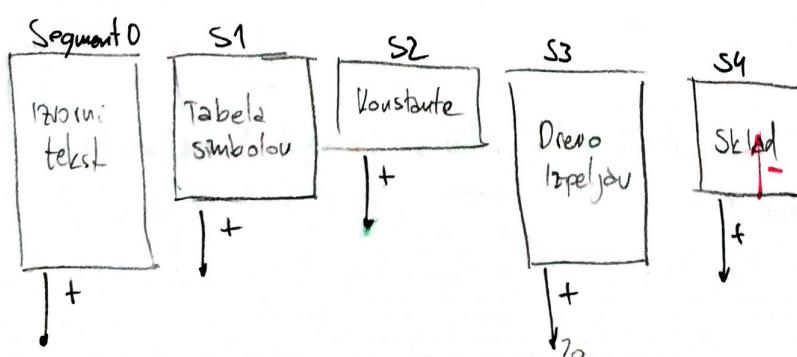


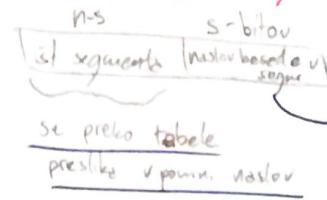
tabela segmentov (segment table) - preko nje je org. preslikovalna funkcija

↪ segm. ni fiksno; velikost tabele ni znana vnaprej

Rešitev: vsak prog. ima svojo TABELO SEG.

Vsek segment ima v tabeli s. svoje polje: **deskriptor segmenta**

n-bitni nov. naslov, ki ga tvori CPU:



ti se pri preslikavi
NE okrajinijo.

① Prisotni bit P - ekvivalenten P bitu pri ustranjevanju

↪ [1]: seg. je v GP, naslov segm. nam pove KJE (nakaterem naslov)

dostop odgovoren od RWX IN je relativni naslov \leq velikost segmenta

↪ [2]: seg. ni v GP, poskus dostopa - zgrešitev, napaka segmenta

↪ Past, PSP prenese segment iz Pmm. P v GP

↪ SA (naslov segmenta) ni veljavен

② Zaščitni ključ RWX = ostri

↪ zaščita lahko popolnejša kot pri ostri.

↪ lahko pomeni, ali se bo seg. spremnil (velikostno),
ali ga lahko uporablja več uporabnikov

...

③ Umetani bit C (change) == ostri.

↪ ko gre seg. v GP, se nastavi na 0, če se piše v segm, gre na 1.

④ Velikost segmenta L (length) ~ ni fiksna velikost seg.

↪ vsebuje trenutno velikost seg. (merjeno v št besed); max je 2^S besed.

↪ dolg S bitov

↪ pri vsakem dostopu se s-bitni naslov znotraj seg. primerja z L

↪ naslov > L: napaka (past) → detekcija prog. napak in nedovoljenih posegov.

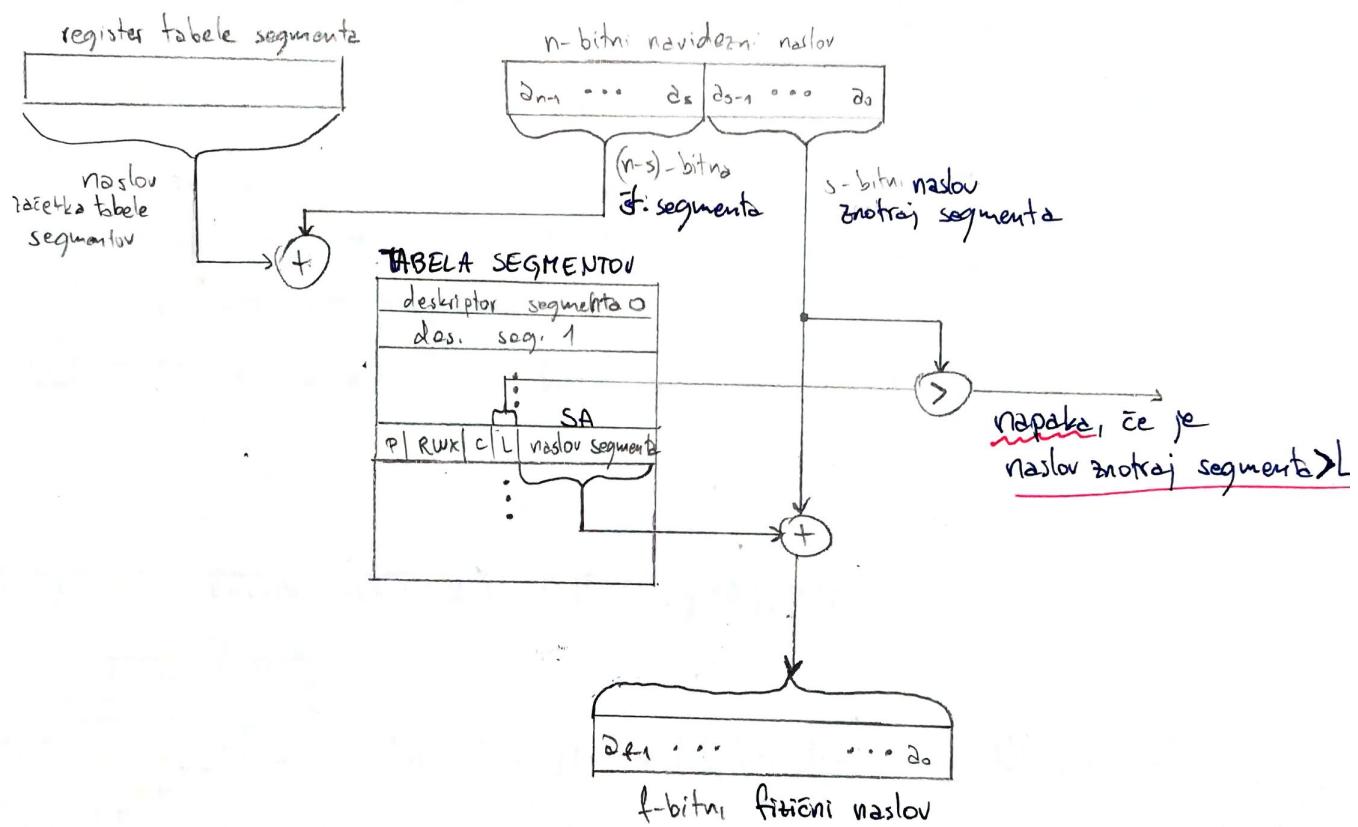
(b) Naslov segments SA (Segment Address)

↳ vsebuje fizični naslov v GP, kjer je segment.

↳ do dostopa se fizični naslov besede prične s naslovom znotraj segmenta

↳ kar je naslov segm. SA poljuben, spodnji naslovni biti: fizične n. **NISO ENAKI** bitom **NN**.

Prostor ni linearen, oz je le znotraj segmenta.



Pri racunanju z NN se št. segmenta (zgojnih n-simetov NN) ne more spremeniti, če bi to želeli, se sproži pao.

Obstojeci segmenti - seg. ki se trenutno izvaja

Tabela obstoječih seg - za vsak obstoječi s vsebuje par [ime seg., št. seg.]

2 možnosti delovanja rač s seg.:

simbolično. Pod tem, je ravn programerju
izvajator pa

① - podoben sistemu rač. ki nimata nov. P. (339)

② - **dinamično povezovanje** (dynamic linking)

↳ v tabelo obstoječih s in v tabelo s. prog. se upiše samo seg., ki vsebuje

začetek programa (pri ① se prenesejo vri segmenti)
pied začetkom

↳ to mod izvajanjem prvič klize vsek drug seg., **povezovalnik** poišče datoteko s tem seg. in ju preko tabele obst. seg (TOS) dodeli et. in nov descriptr v tabeli seg. programa (TSP).

④ povezijo se samo tisti seg., ki jih prog. res uporablja

KAKO LAHKO VEČ PROG. UPORABLJA ISTI SEGMENT?

Razdelitev seg. v 2 vrsti:

① **Lokalni segmenti** - pripadajo 1 programu (dostop drugim prog Ni dovoljen)

② **Globalni segmenti** - lahko dostopajo vsi prog. Npr. sistemski prog.

↳ descriptori vsebujejo se bit **G** - **1**-globalen. Vsebuje **kotalec na descriptor v tabeli GS** (globalnih segmentov) kot naslov segmenta se uporablja parameter iz **GS** - **0** - preslikovanje ostane enako

TABELA SEGMENTOU PROG. X

deskriptor segmenta X _i			
G=1			
P	RWx	G	L
P	RWx	G	L
			nastav deskriptorje
:			

TABELA SEGMENTOU PROG. Y

deskriptor segmenta Y _j			
G=1			
P	RWx	G	L
P	RWx	G	L
			nastav deskriptorje
:			

TABELA GLOBALNIH SEGMENTOU

deskriptor segmenta G _o			
deskri. segm. G _o			
G=o			
P	RWx	G	L
			nastav segmenta
:			

Lahko pride (pri segm.) do prenetavanja.

preskrivanje

↳ zaradi različne velikosti seg., je v GP potrebljao najti ustrezen velik prostor.

↳ imamo luknje

↳ ce so luknje vecje od velike za segment, vsake posamezne pa ne:

SUM

Zunanja fragmentacija (precejšnji problem)
 (pri ostranjevanju je ni), (pri segmentaciji ni notranje fragmentacije)

obc resiti glede na izkoriscenost enako škodljivi

endar pri notranji resaj vemo max in avg. velikost.

Zunanj se stavno spreminja n

Algoritmi za izvajanje fragmentacije: → imamo k luknji, označimo jih: s_1, s_2, \dots, s_k

s_i ... velikost luknje i;

$s_1 \leq s_2 \leq \dots \leq s_k$

① Najboljše ujemanje: poiščemo min i, da bo $s \leq s_i$

↳ segment prenesemo v luknjo s_i : Razlika najmanjša

↳ rezultat so zelo majhne luknje, ki skoraj zanerljivo ne bodo uporabne

s ... segment velikosti s_i , ki ga želimo prenesti

② Najslabše ujemanje: Luknje razvrstimo po njihovi padajoči velikosti.

$s_1 \geq s_2 \geq \dots \geq s_k$. Velja: $s \leq s_i$ (isto kot pri ①). Razlika je največja.

↳ Luknje so vecje, vendar jih hitro zmanjša

③ Prvo ujemanje: Luknje razvrstene po naraščajočih začetnih naslovih: ne glede na velikost

Luknja z min indeksom i, ko je $s \leq s_i$, je tista, v katero gre segment.

Razlika vključna.

④ najboljši performance na meritrah

strukturirane povišljosti - počasna operacija

↳ izvede se, ko uporaba poviš. zaradi fragmentacije ni več možna

↳ postopek, ko vse lukuje strumento v prazem prostor

↳ vmes izvajanje programov stoji

“ ↳ slabo. čisti oblika segmentacije se uporablja redko

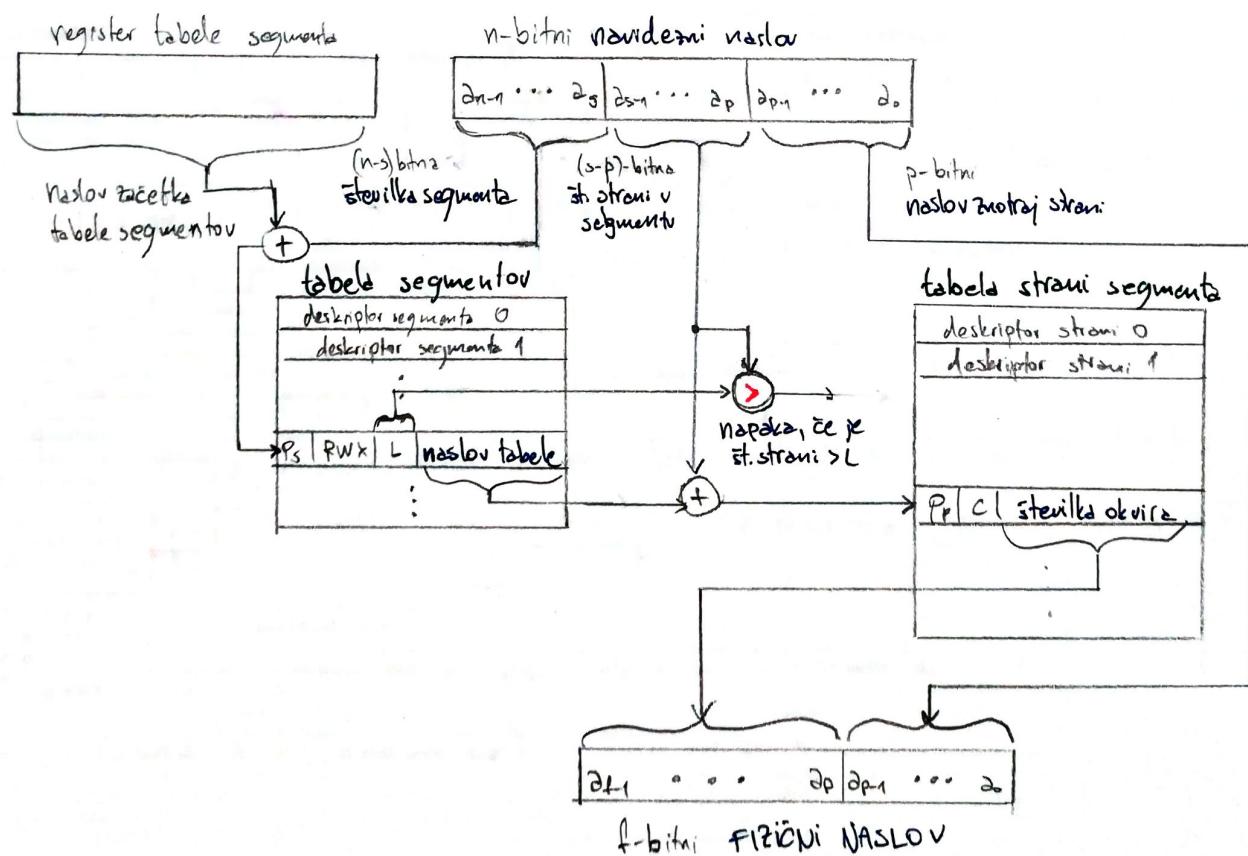
(339) 9.3.3 segmentacija z ostranjevanjem

Vsek segment razdelimo na strani z nacinom ostranjevanja.

To je: linearna segmentacija (linear segmentation)

Vsek prog. ima poleg TS (tabela segmentov) tudi mnogico tabel strani → po 1 za vsek segment.

GP razdeljen na okvire strani:



zadnje 2 dela naslova sta v bistvu naslov besede v segmentu, ki je podan v 2 delih.

① Prisotni segment bit Ps (Present)

- ↳ $\boxed{1}$: tabela strani tega segm. je v GP. PTA pove na katerem naslovu.
- ↳ $\boxed{0}$: TS ni v GP. Postopek dostopa do besede v takem segmentu: napaka segmenta. PSP prenese tabelo strani tega segmenta iz Pm.P. v GP.
Dostop odvisen od RWX (inče st. strani
ni veja od segmenta L [meri se v st. strani])

② Zaščitni kljuc RWX == čisti segm

③ Velikost segmenta L - trenutna velikost segmenta merjena v st. strani

- ↳ max.: 2^{3-P} strani in je določena v napravi. L mora biti dolg s-p bitov
- ↳ če je st. strani $> L \rightarrow napaka (detekcija prog. napak in nedovoljenih dostopov)
(s-p bitov določa to st.)$
- ↳ velikost L se spreminja

④ Naslov tabele strani PTA (Page table address)

- ↳ fiz. naslov v GP na katerem je tabela strani za ta segment (če je $P_s = 1$)
- ↳ ob dostopu se PTA pristeje st. strani (v segmentu) $\xrightarrow{\text{SUM}}$ fizični naslov deskriptorja v tabeli strani

Ko je $P_s = 1$, se prelikovanje nadaljuje preko tabeli strani.

Naslov te tabele je glavni rezultat ki ga opravi Tabela Segmentov.

① Prisotna stran bit Ps (present) \rightarrow ni potreben, da babil segm. vedno v celoti v GP. (Drugače kot pri čisti segmentaciji)

- ↳ $\boxed{1}$: stran je v enem izmed okvirov v GP. Fiz N. = st. okvirja + spodnji p bitov NN.
- ↳ $\boxed{0}$: strani M v GP: napaka strani \rightarrow PSP poservisira in prenese stran iz Pm.P v GP

② Umarani bit C == ostrijanje.

- ↳ nekateri sistemi imajo bit spremembre tudi v tabeli segmentov.

③ Četrtikna okvirja FN (Frame number) == ostrijanje

seg. z ostr. 12BOLJEAVE:

⊕ odstranitev zmanjšje fragmentacije. Ni več zamudnega strnjevanja polnem.

⊕ boljša iskoriščenost prostora, ki ga zasedajo segmenti. V GP ne rabijo biti celi segmenti.

Lahko so le tiste, ki se trenutno uporabljajo.
(dovolj je)