OSNOVE RAČUNALNIŠKE ARHITEKTURE II

11 Navidezni pomnilnik

- Navidezni pomnilnik (virtual memory) je ime za prostor v pomožnem pomnilniku (magnetni disk), ki je za uporabnika videti kot glavni pomnilnik.

- Običajen dostop do pomožnega pomnilnika poteka z V/I ukazi, oziroma z V/I programi in je drugačen kot do glavnega pomnilnika.

- Da je prostor v pomožnem pomnilniku videti kot glavni pomnilnik in so prenosi med njima za uporabnika nevidni ($\Rightarrow$ od tod ime navidezni pomnilnik), je potrebna dodatna logika v CPE.

- Vzrok za pojav navideznega pomnilnika je bil premajhen glavni pomnilnik v šestdesetih letih.
- Programerji so morali programe, ki so bili večji od glavnega pomnilnika, razdeliti na več delov.
- Tak način programiranja je znan pod imenom **prekrivanje (overlay).**
- Avtomatizacija tega postopka naj bi razbremenila programerja in zagotovila hitro izvajanje programov.

- Navidezni pomnilnik ima danes večina računalnikov, vendar razlog zato ni več premajhen glavni pomnilnik kot včasih, temveč:
  - Veliko nižja cena pomnilnika na magnetnem disku
  - Enostavna rešitev pozicijske neodvisnosti programov
  - Zaščita pomnilnika
- Zaradi teh razlogov imajo navidezni pomnilnik tudi računalniki, ki imajo pomožni pomnilnik enak ali malo večji od glavnega pomnilnika.
- Pri skoraj vseh računalnikih je prostor na disku razdeljen na prostor za navidezni pomnilnik in prostor za datoteke, ki je običajno precej večji od navideznega.

11.1 Pomnilniška hierarhija

- S pomnilniško hierarhijo želimo doseči, da CPE vidi velik, počasen in cenen pomnilnik M_n kot hiter (tudi drag in zato majhen) pomnilnik M_1.
- To je možno doseči zaradi lokalnosti pomnilniških dostopov, brez lokalnosti pomnilniška hierarhija ne bi uspešno delovala.
- Pomnilniška hierarhija se od enonivojskega pomnilnika razlikuje v naslednjih lastnostih:

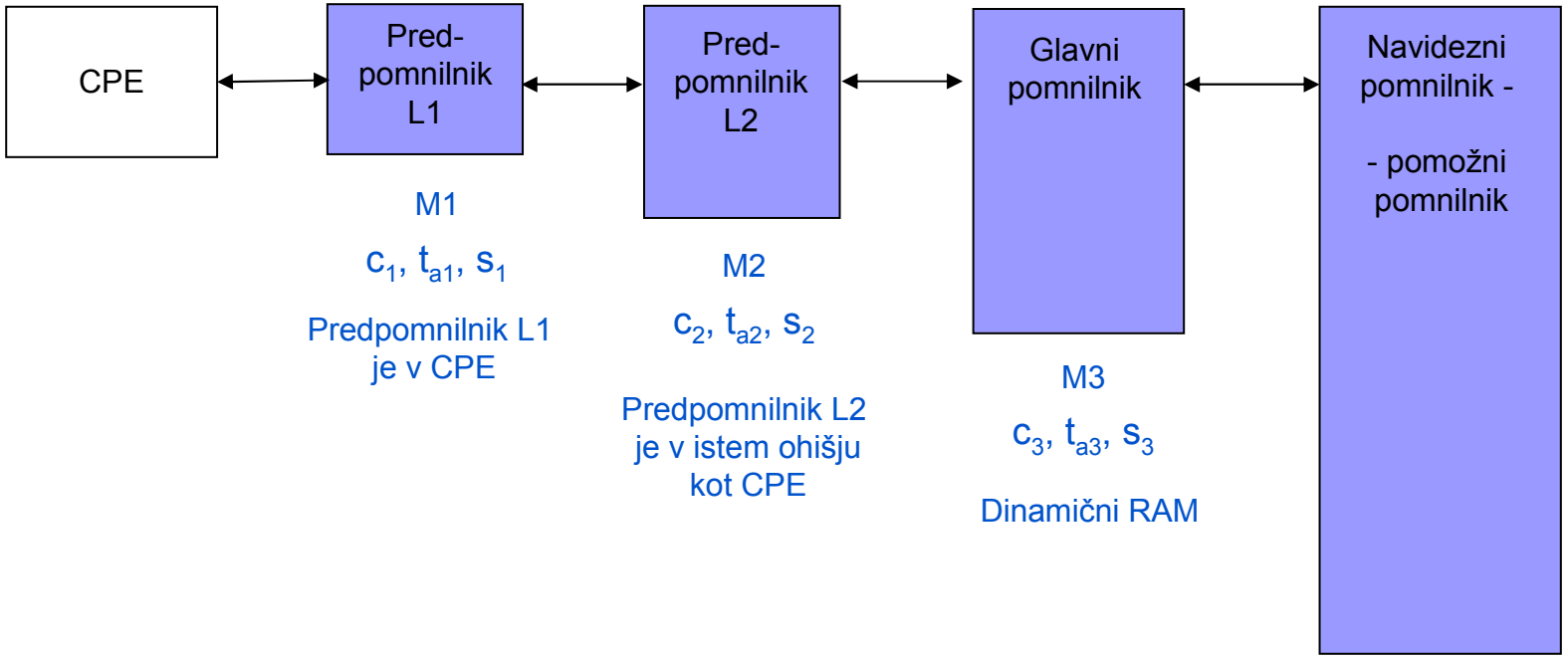
- Čas dostopa ni enak za vse pomnilniške naslove, odvisen je od pomnilniškega nivoja na katerem je trenutno iskana pomnilniška beseda
- Za določen pomnilniški dostop ne moremo predvideti njegovega trajanja, znana je samo statistično določena povprečna vrednost časa dostopa

- CPE pošlje v pomnilniško hierarhijo naslov, ki se nanaša na pomnilnik M_n (zadnji v hierarhiji), vendar to ne pomeni, da bo dostop v resnici opravljen v M_n.
  - Če je beseda, do katere želi CPE dostop, v M_1, se opravi dostop do M_1

- Če besede ni v M_1, se v M_1 prenese iz M_2
- Če želene besede ni tudi v M_2, se v M_2 prenese iz M_3
- . . .
- Pri vsakem dostopu je zahtevana informacija vedno v pomnilniku M_n na zadnjem nivoju

- Na večini današnjih računalnikov je običajna štirinivojska pomnilniška hierarhija z dvema nivojema predpomnilnikov, glavnim pomnilnikom in navideznim pomnilnikom.

Štirinivojska pomnilniška hierarhija



c_i - cena/bit nivoja i
t_{ai} - čas dostopa nivoja i
s_i - velikost pomnilnika nivoja i

Velja:
$c_i > c_{i+1}$
$t_{ai} < t_{ai+1}$
$s_i < s_{i+1}$

- Uspešnost pomnilniške hierarhije ocenjujemo podobno kot pri predpomnilniku z verjetnostjo zadetka H_i, ki podaja verjetnost, da je pri nekem dostopu informacija na nivoju i.

- Ker je na najvišjem nivoju vedno vsa informacija, je verjetnost $H_n = 1$.

- Če je pri skupaj N dostopih, v N_1 primerih informacija v M_1, v N_2 primerih v M_2 in ne v M_1 , v N_3 primerih v M_3 in ne v M_1 in ne v M_2, itn., potem velja:

$$N = N_1 + N_2 + \ldots + N_n$$

$$H_1 = \frac{N_1}{N} \qquad H_2 = \frac{N_1 + N_2}{N} \qquad H_n = \frac{N_1 + N_2 + \ldots + N_n}{N} = 1$$

- Povprečni čas dostopa do *n*-nivojske pomnilniške hierarhije kot ga vidi CPE je:

$$t_a = t_{a1} + (1 - H_1)t_{a2} + \ldots + (1 - H_{i-1})t_{ai} + \ldots + (1 - H_{n-1})t_{an}$$

- Za štirinivojsko pomnilniško hierarhijo velja:

$$t_a = t_{a1} + (1 - H_1)t_{a2} + (1 - H_2)t_{a3} + (1 - H_3)t_{a4}$$

- Ker je navidezni pomnilnik na magnetnem disku, je čas za dostop do prve besede bloka zelo dolg.
- Da se omili vpliv zelo velike zgrešitvene kazni, se pri navideznem pomnilniku uporabljajo naslednje rešitve:
  - Bloki morajo biti veliki, da se zmanjša vpliv dolgega časa dostopa (4KB, 8KB, do 64KB in več)
  - Vsak blok se lahko preslika v poljuben blok glavnega pomnilnika
  - Zamenjava blokov se ob zgrešitvah opravi programsko in ne strojno kot pri predpomnilniku, to pa omogoča uporabo bolj zahtevnih algoritmov za izbiro bloka, ki naj se zamenja
  - Uporablja se pisanje nazaj, zaradi počasnosti diska je pisanje skozi neuporabno

11.2 Preslikovanje navideznih naslovov

- Naslovi, ki jih pošilja CPE v pomnilniško hierarhijo, se vedno nanašajo na najvišji nivo (M_n).
- Če je najvišji nivo navidezni pomnilnik, so to **navidezni naslovi.**
- Pri navideznem pomnilniku je podobno kot pri predpomnilniku, del vsebine navideznega pomnilnika je preslikan v glavni pomnilnik.
- Vedno obstaja preslikava, ki določa podmnožico vsebine navideznega pomnilnika, ki je v glavnem pomnilniku.

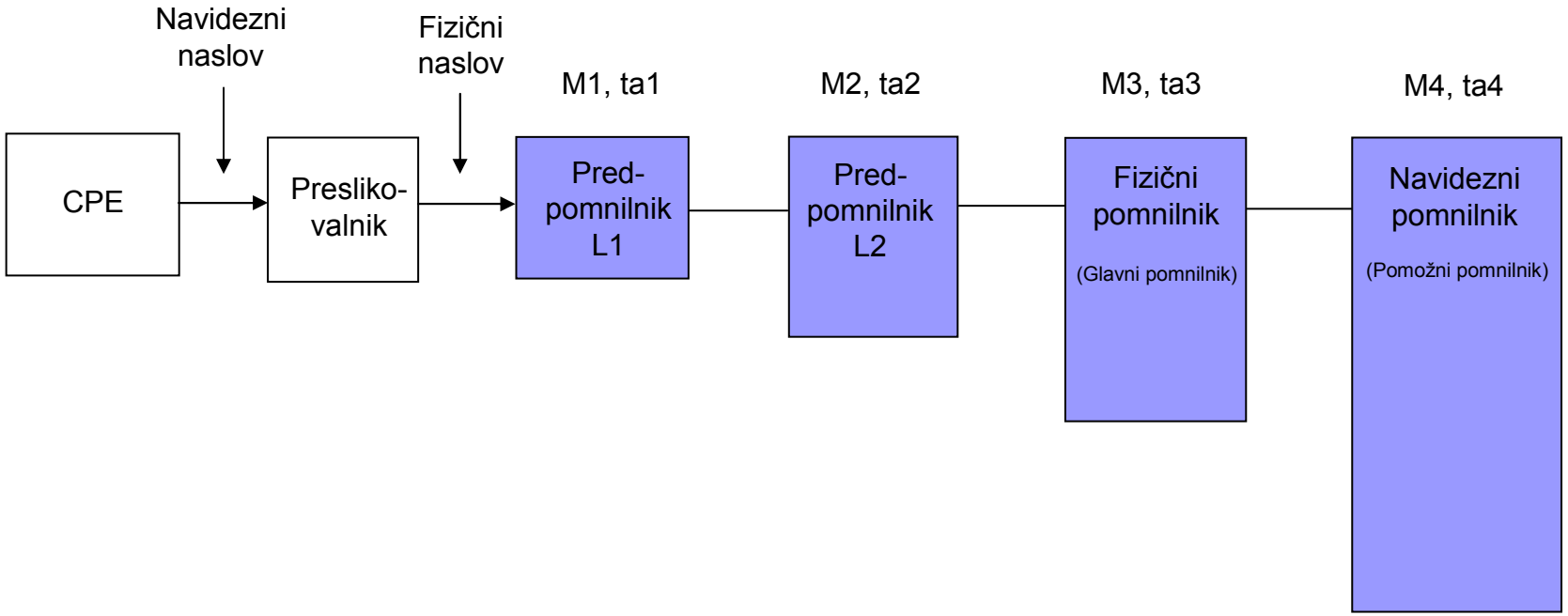
- Z informacijo o preslikavi je iz navideznega naslova vedno mogoče ugotoviti ali je iskana informacija v glavnem pomnilniku ali ne.
- V povezavi z navideznim pomnilnikom imenujemo glavni pomnilnik **fizični pomnilnik.**
- Pri vsakem pomnilniškem dostopu je treba **navidezni naslov**, ki ga pošilja CPE, preslikati v **fizični naslov**.
- Množica navideznih naslovov je običajno večja od množice fizičnih naslovov, ni pa nujno.

- Pri navideznem pomnilniku se preslikovanje navideznih naslovov v fizične večkrat označuje kot **dinamično preslikovanje**, kar pomeni, da se preslikovalna funkcija spreminja s časom.

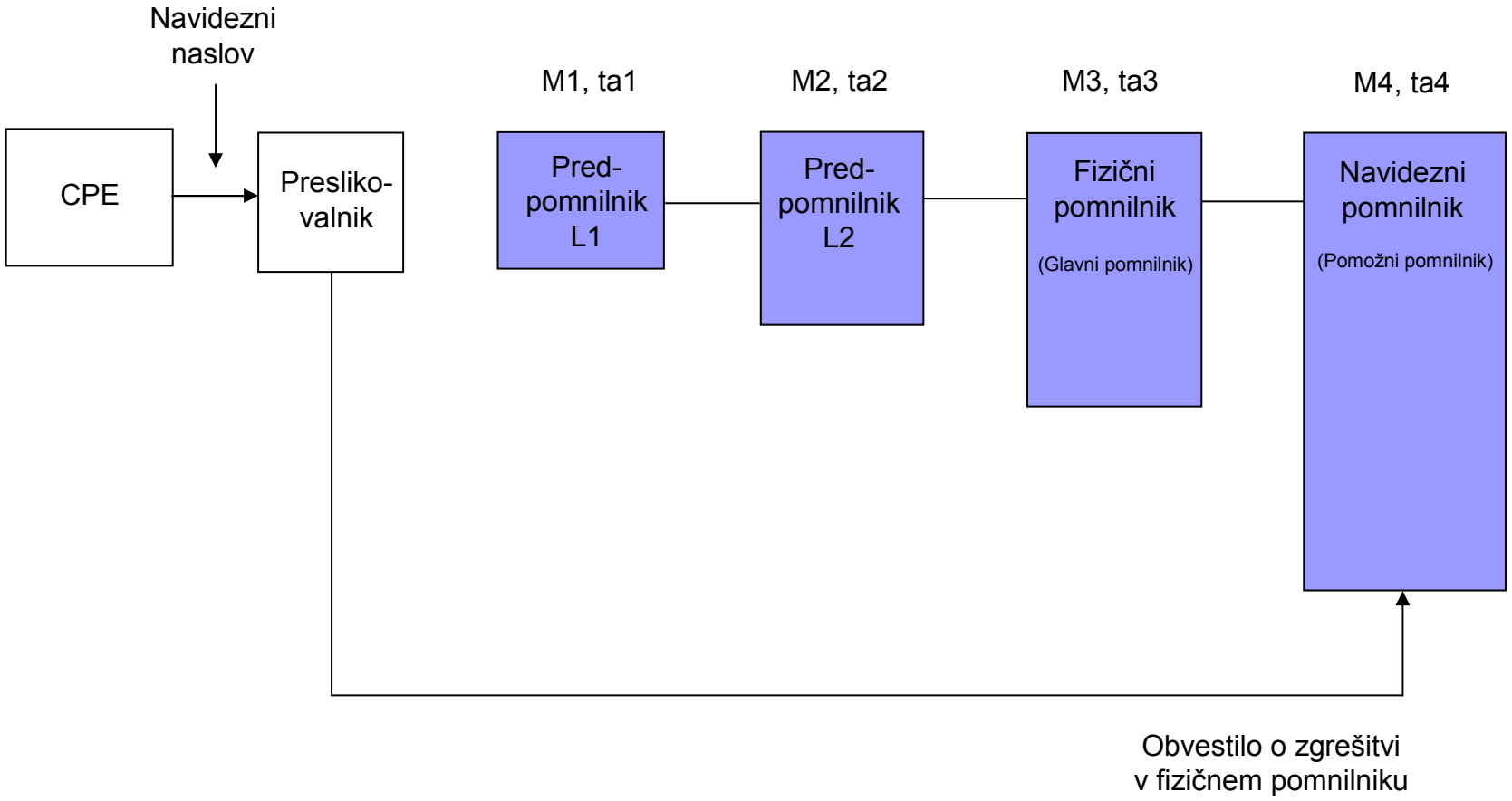
- Enota, ki izvaja to preslikavo, ima različna imena:
  - Naprava za dinamično preslikovanje naslovov (Dynamic Address Translation Facility - IBM 370)
  - Enota za upravljanje s pomnilnikom (Memory Management Unit - MMU pri procesorjih Intel in Motorola)

- Pri večini računalnikov se navidezni naslov preslika v fizičnega, nato pa se fizični naslov (ne navidezni) uporabi za dostop do predpomnilnika.

Preslikovanje navideznih naslovov



Preslikovanje navideznih naslovov



- Preslikovanje je možno tudi izklopiti, takrat je navidezni naslov enak fizičnemu.

- Preslikovalna funkcija se vzpostavi programsko, zato je na začetku pri vklopu računalnika preslikovanje navideznih naslovov v fizične potrebno izklopiti.

11.3 Vrste navideznih pomnilnikov

- Navidezni pomnilnik je možno realizirati na več načinov:
  - z bloki fiksne velikosti - strani
  - z bloki spremenljive velikosti - segmenti
- Osnovne razlike med obema vrstama navideznih pomnilnikov:

	Stran	Segment
Naslov	Enodimenzionalen	Večdimenzionalen
Vidno za programerja	Ne	Lahko
Zamenjava bloka	Enostavna (enako veliki bloki)	Zapletena (potreben zvezen prostor spremenljive dolžine)
Izguba pomnilniškega prostora	Notranja fragmentacija	Zunanja fragmentacija
Učinkovitost prenosov	Da - velikost strani se lahko prilagodi	Ne vedno

- Čisti segmenti se redko uporabljajo, pač pa se veliko uporablja kombinacija obeh načinov, ko so segmenti razdeljeni na strani.

- V tem primeru za segment ni več potreben zvezen prostor v glavnem pomnilniku, niti ni potrebno, da je v glavnem pomnilniku cel segment.

- Načini realizacije navideznega pomnilnika so:
  - Ostranjevanje
  - Segmentacija
  - Segmentacija z ostranjevanjem

11.4 Ostranjevanje

- Najstarejša vrsta navideznega pomnilnika je ostranjevanje (paging).
- Pomožni pomnilnik je razdeljen na bloke enakih velikosti - **strani (pages)**. Vse strani sestavljajo navidezni pomnilnik.
- Glavni pomnilnik je razdeljen prav tako na bloke enake velikosti - **okvirje strani (page frames).**
- Vsako stran je možno prenesti v poljuben okvir.

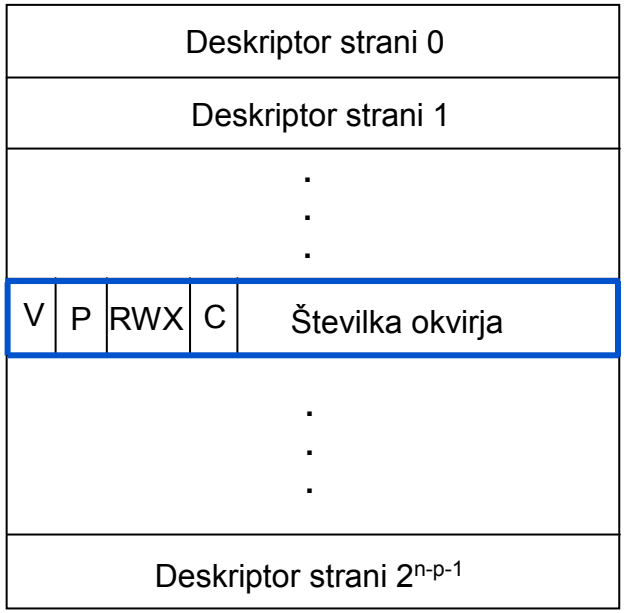
- Velikost strani je potenca števila 2, tipične velikosti strani so 4KB, 8KB do 64KB ali tudi več.
- Vsak program zaseda določeno število strani, zadnja stran običajno ni cela zasedena, ker velikost programa največkrat ni enaka mnogokratniku velikosti strani.
- V povprečju je pri vsakem programu polovica zadnje strani, ki se prenese v glavni pomnilnik, neizkoriščena.
- Ta neizkoriščenost glavnega pomnilnika pri ostranjevanju se imenuje **notranja fragmentacija**.

- Preslikava navideznega naslova v fizični naslov (preslikovalna funkcija) je definirana s pomočjo posebne tabele - **tabele strani (page table)**.

- Vsaka stran navideznega pomnilnika ima v tabeli strani eno polje, ker je število strani fiksno, je določena tudi največja velikost tabele strani.

- Polje v tabeli strani, ki pripada določeni strani, imenujemo **deskriptor strani**.

Zgradba tabele strani in deskriptorja



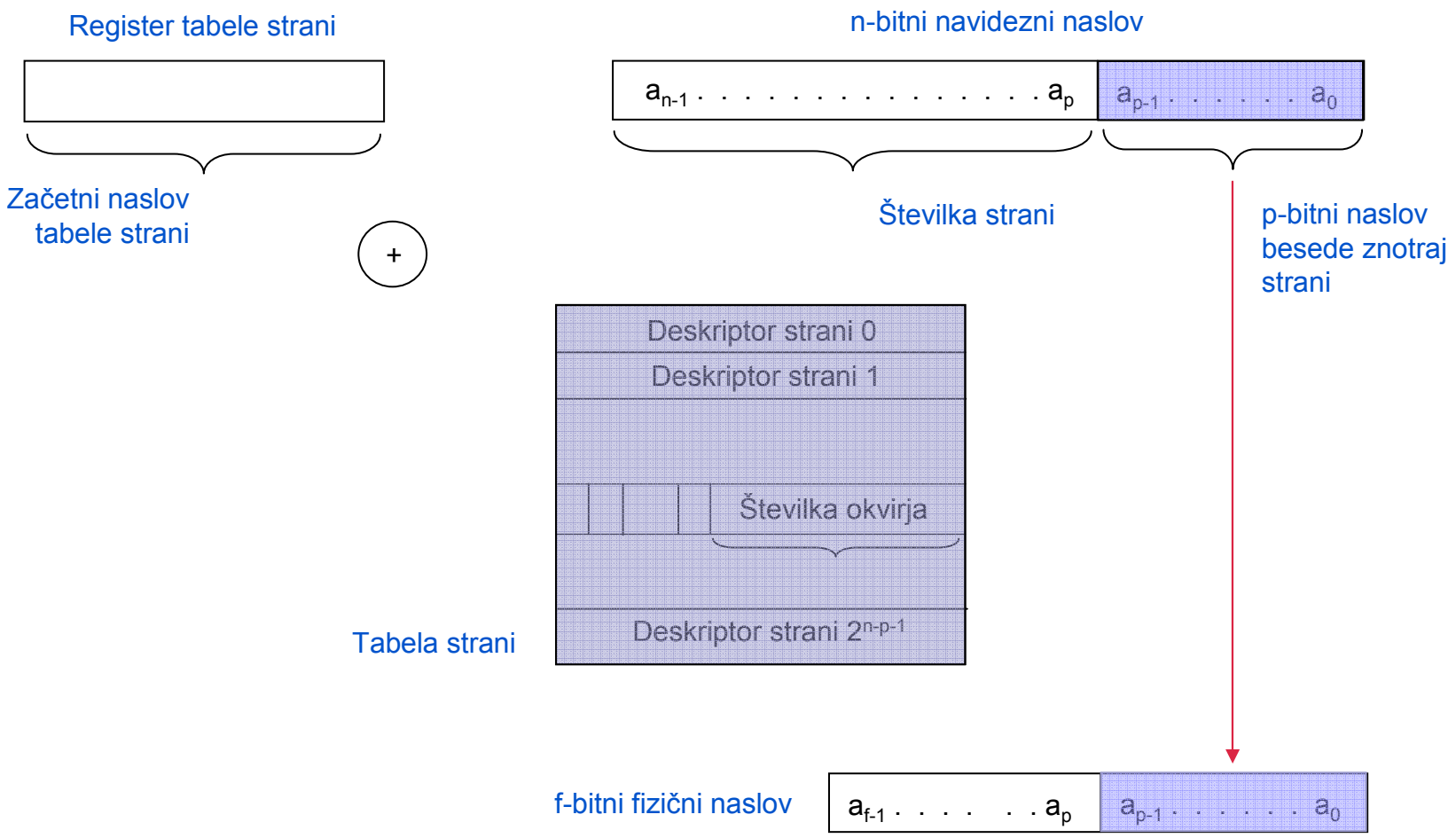
Deskriptor strani

V - veljavni bit (Valid)
P - bit prisotnosti (Present)
RWX - zaščitni ključ (Read,Write,
C - umazani bit (Change)

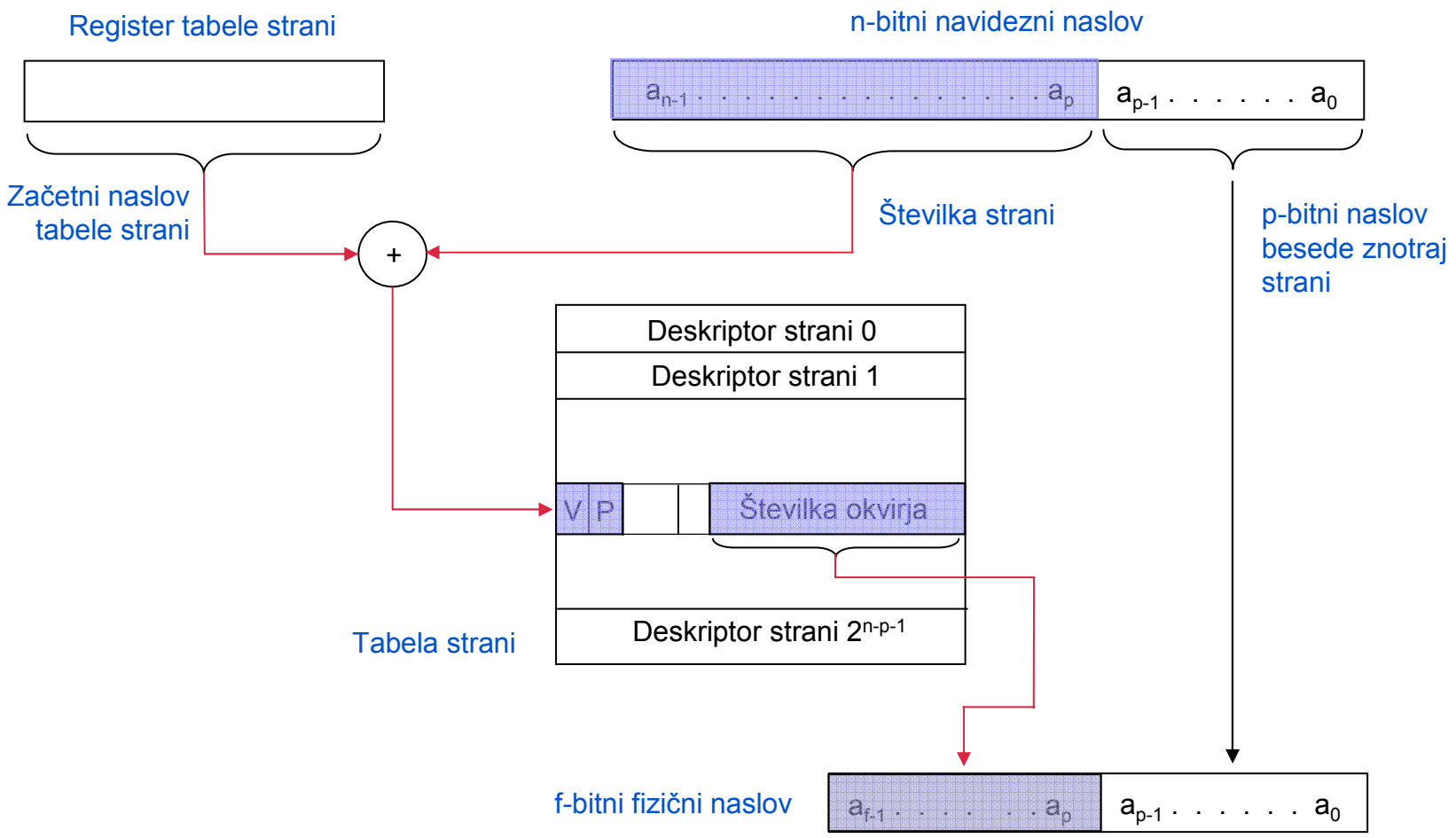
- Navidezni pomnilniški naslov dolžine n-bitov lahko razdelimo na dva dela:
  - Spodnji biti v naslovu (p-bitov a_0 - a_{p-1}) določajo naslov besede znotraj strani
  - Preostali zgornji biti (n-p bitov a_p - a_{n-1}) pa dolčajo številko strani

- Pri preslikavi navideznega naslova v fizični se preko deskriptorja v tabeli strani številka strani zamenja s številko okvirja v katerem je ta stran v fizičnem pomnilniku, spodnji biti pa ostanejo nespremenjeni.

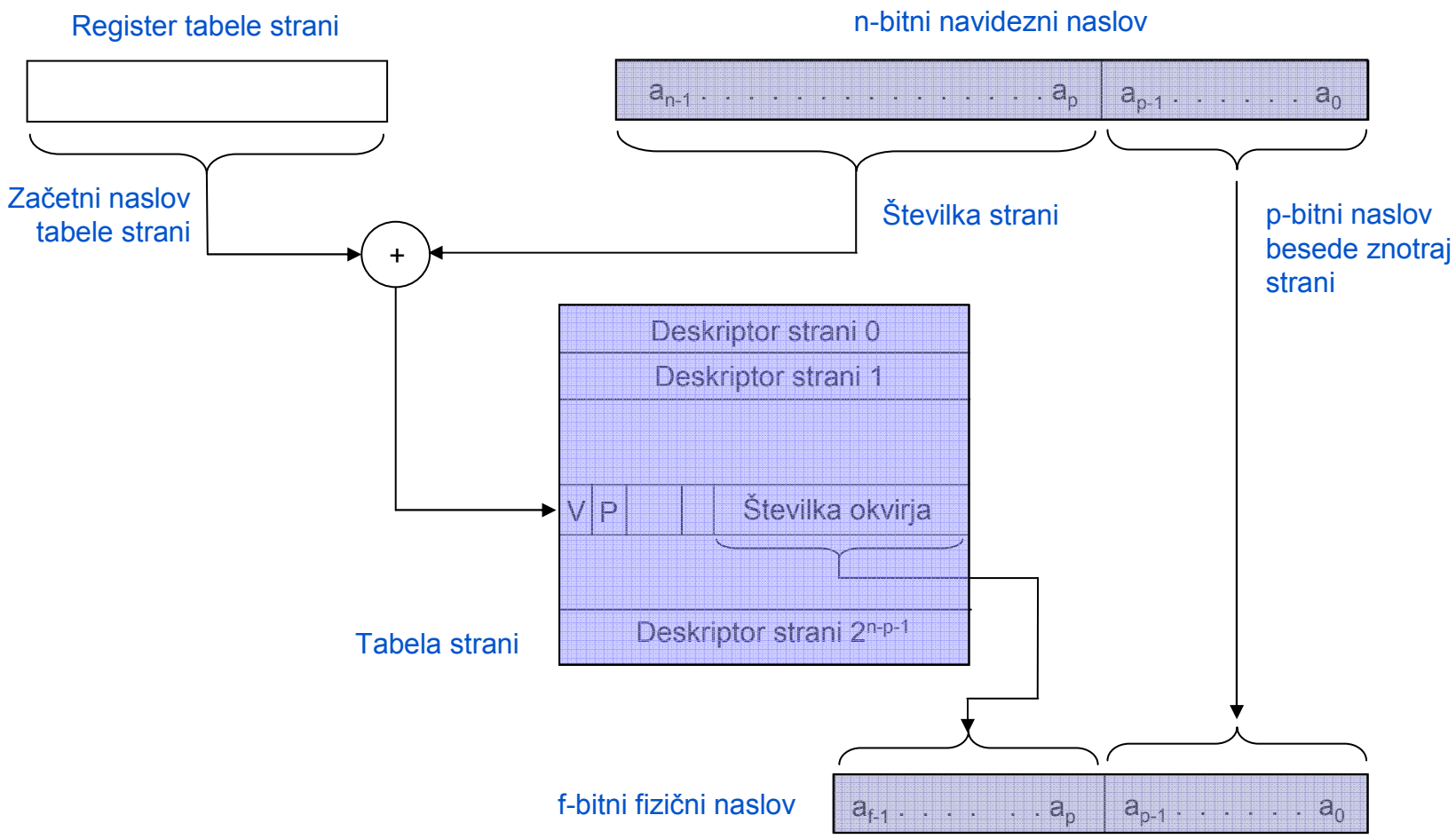
Preslikovanje navideznih naslovov v fizične pri ostranjevanju



Preslikovanje navideznih naslovov v fizične pri ostranjevanju



Preslikovanje navideznih naslovov v fizične pri ostranjevanju



- Tak način preslikovanja navideznih naslovov v fizične imenujemo **linearno preslikovanje**, ker je navidezni naslovni prostor linearen in pri računanju z navideznimi naslovi ni nobenih omejitev.

- Delitev pomnilniškega prostora na strani je za uporabnika nevidna.

- Tabela strani je običajno v glavnem pomnilniku, če je ena sama, je preslikava navideznega naslova v fizični **enonivojska preslikava**.

- Tabela strani lahko v pomnilniku zasede veliko prostora, zato je preslikovanje običajno realizirano preko več nivojev tabel strani.
- Prednost večnivojskega preslikovanja je, da se zmanjša prostor, ki ga zasedajo tabele strani v glavnem pomnilniku.
- Največkrat se uporablja trinivojska preslikava preko treh nivojev tabel strani.

- Primerjava lastnosti predpomnilnika in navideznega pomnilnika z ostranjevanjem

	Predpomnilnik	Ostranjevanje
Dostop	Predpomnilniška vrstica (blok)	Okvir strani
Blok	64B	8KB
Verjetnost zgrešitve (1-H)	1% do 20%	< 0,001%
Zadetek	1 urina perioda	~ 10 do 100 urinih period
Zgrešitev	~ 10 do 100 urinih ciklov	~ 10M urinih ciklov
Zamenjava bloka	Strojno (hardware)	Programsko (software)

11.5 Segmentacija

- Pri segmentaciji je navidezni pomnilnik razdeljen na **segmente**, segmenti so različnih dolžin, vsebine segmentov pa imajo za programerja določen pomen.
- Med izvajanjem programov se dolžine in število segmentov lahko spreminjajo.
- Vsak segment se mora med izvajanjem seveda prenesti v glavni pomnilnik, kamor se lahko prenese na poljuben naslov.

- Vsak program je pri segmentaciji videti kot nekaj med seboj povezanih segmentov.
- Pri prehajanju iz enega segmenta v drugega so običajno predpisane omejitve, prehod je možen samo na vnaprej določen način.
- Vsak segment lahko obravnavamo kot svoj lasten pomnilniški prostor, ki je na predpisan način povezan z drugimi.
- Segmentacija se zato označuje tudi kot večdimenzionalni navidezni pomnilnik.

- Preslikava je realizirana preko tabele segmentov, ki vsebuje deskriptorje segmentov.
- V deskriptorju je fizični naslov segmenta v glavnem pomnilniku (namesto številke-naslova okvirja strani pri ostranjevanju) in tudi velikost segmenta.
- Ker število segmentov ni fiksno, velikost tabele ni vnaprej določena. Običajno ima vsak program svojo tabelo segmentov.

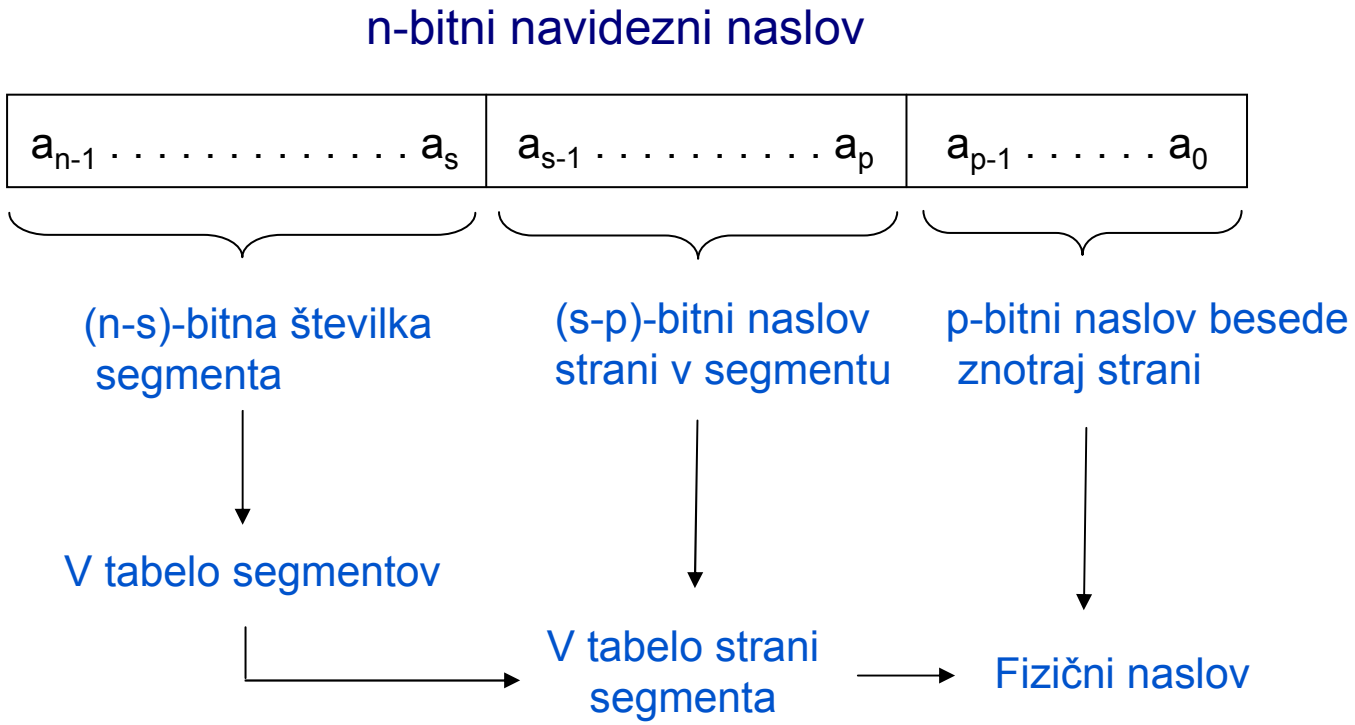
- Pri prenosu segmentov iz navideznega v glavni pomnilnik je treba v glavnem pomnilniku najti ustrezno velik zvezen prostor, ki mora biti najmanj tako velik kot segment.
- V večini primerov je ta prostor večji kot je segment in po nekaj prenosih imamo tako v pomnilniku različno velike prazne dele - luknje.
- Ti prazni deli so lahko skupaj dovolj veliki za nov segment, vsak zase pa ne.
- Ta neizkoriščenost oziroma drobljenje pomnilniškega prostora v glavnem pomnilniku pri segmentaciji se imenuje **zunanja fragmentacija**.

- Pri ostranjevanju se zunanja fragmentacija ne more pojaviti, ker so vse strani enako velike, imamo pa notranjo fragmentacijo (povprečno pol strani na program).
- Nasprotno pa se pri segmentaciji ne pojavi notranja fragmentacja, ker je segment tako dolg kot ustrezen programski modul.
- Pri notranji fragmentaciji poznamo njeno velikost, dočim se zunanja fragmentacija stalno spreminja, zato je pojav zunanje fragmentacije hujši problem.

11.6 Segmentacija z ostranjevanjem

- Pri segmentaciji z ostranjevanjem želimo izkoristiti prednosti segmentacije in se izogniti težavam, ki jih povzroča.
- Vsak segment je razdeljen na strani enako kot pri ostranjevanju. To imenujemo **linearna segmentacija**.
- Vsakemu programu pripada tabela segmentov, poleg tega pa še po ena tabela strani za vsak segment tega programa.

- V n-bitnem navideznem pomnilniškem naslovu zgornji biti določajo številko segmenta, srednji biti številko strani in spodnji biti naslov besede znotraj strani.



- Segmentacija z ostranjevanjem je izboljšava čiste segmentacije:
  - Ni več zunanje fragmentacije, zato odpade potreba po strnjevanju (defragmentaciji) pomnilnika
  - Prostor, ki ga zasedajo segmenti, je bolje izkoriščen. Ker so segmenti razdeljeni na strani, ni potrebno da je v glavnem pomnilniku cel segment, temveč samo tiste strani segmenta, ki se trenutno uporabljajo

11.7 Pohitritev preslikovanja

- Pri preslikavi navideznega naslova v fizični naslov je potreben dostop do tabele strani ali segmentov.
- Te tabele so velike, zato so shranjene v glavnem pomnilniku ali celo v navideznem pomnilniku.
- Pri vsakem dostopu do pomnilnika sta zato potrebna najmanj dva dostopa do glavnega pomnilnika (če odmislimo predpomnilnik):
  - Prvi dostop do glavnega pomnilnika je do deskriptorja v tabeli strani
  - Drugi dostop je do želene besede na fizičnem naslovu v glavnem pomnilniku

- To število dostopov se pri večnivojski preslikavi navideznega naslova v fizični ali pri segmentaciji z ostranjevanjem poveča na 3 do 4 dostope do glavnega pomnilnika.

- Tak računalnik za uporabnika ne bi bil dovolj hiter, zato imajo vsi računalniki z navideznim pomnilnikom vgrajen poseben predpomnilnik, ki vsebuje nekaj nazadnje uporabljenih deskriptorjev.

- Delovanje tega predpomnilnika je enako kot pri običajnem predpomnilniku, le da vsebuje vedno samo deskriptorje nikoli operandov ali ukazov.

- Za ta posebni predpomnilnik se uporabljajo oznake:
  - **Preslikovalni predpomnilnik** (translation cache)
  - **TLB** (Translation Lookaside Buffer)
- Dolžina bloka v preslikovalnem predpomnilniku je enaka dolžini deskriptorja, v kontrolnem delu pa je številka strani, ki ji deskriptor pripada.
- Visoko verjetnost zadetka (99% do 99,9%) se lahko doseže že z nekaj deskriptorji, zato je preslikovalni predpomnilnik lahko majhen in je večkrat čisti asociativni.

- Pri zadetku v preslikovalnem predpomnilniku dostop do tabele ni potreben
  - Deskriptor iz preslikovalnega predpomnilnika se uporabi za tvorbo fizičnega naslova
  - Fizični naslov pa se uporabi za dostop do predpomnilnika
  - Če imamo predpomnilnik razdeljen na ukazni in operandni predpomnilnik, sta potrebna tudi dva preslikovalna predpomnilnika (ukazni in operandni)

11.8 Strategije in algoritmi

- Delovanje navideznega pomnilnika vodi operacijski sistem, s ciljem doseči največjo izkoriščenost računalnika.
- Kot velika izkoriščenost se večinoma smatra, da se dana množica programov izvrši v najkrajšem možnem času.
- Na izkoriščenost računalnika vpliva izbira pravil, ki določajo:
  - Koliko okvirov strani naj ima v glavnem pomnilniku nek program

- Kdaj, katere in koliko strani naj se prenese iz pomožnega v glavni pomnilnik
- Katere strani naj se prenesejo iz glavnega pomnilnika nazaj v pomožni pomnilnik

- Ta pravila se imenujejejo **dodeljevalna, polnilna in zamenjevalna strategija**.

- Pri navideznem pomnilniku so strategije realizirane programsko in ne strojno kot pri predpomnilniku.

- Vse tri strategije izvajajo algoritmi s skupnim imenom **upravljanje s pomnilnikom** (memory management).

- Najpreprostejša je polnilna strategija, ob vsaki napaki strani se v glavni pomnilnik prenese zahtevana stran:
  - **Polnjenje na zahtevo** - v glavni pomnilnik se prenese nova stran, ko pride do zgrešitve
  - **Vnaprejšnje polnjenje** - poleg strani zaradi katere je prišlo do napake, se v glavni pomnilnik prenese še ena ali več sosednjih strani
- Dodeljevalne strategije določajo koliko okvirjev strani zaseda v glavnem pomnilniku en program:
  - **Fiksna razdelitev** - število okvirjev, ki so dodeljeni enemu programu je fiksno in se s časom ne spreminja
  - **Spremenljiva razdelitev** - število okvirjev, ki so dodeljeni programu se sčasom spreminja (upoštevanje sprememb lokalnosti v programih)

- Zamenjevalne strategije za izbiro strani, ki naj se zamenja, imajo običajno za osnovo LRU algoritem.

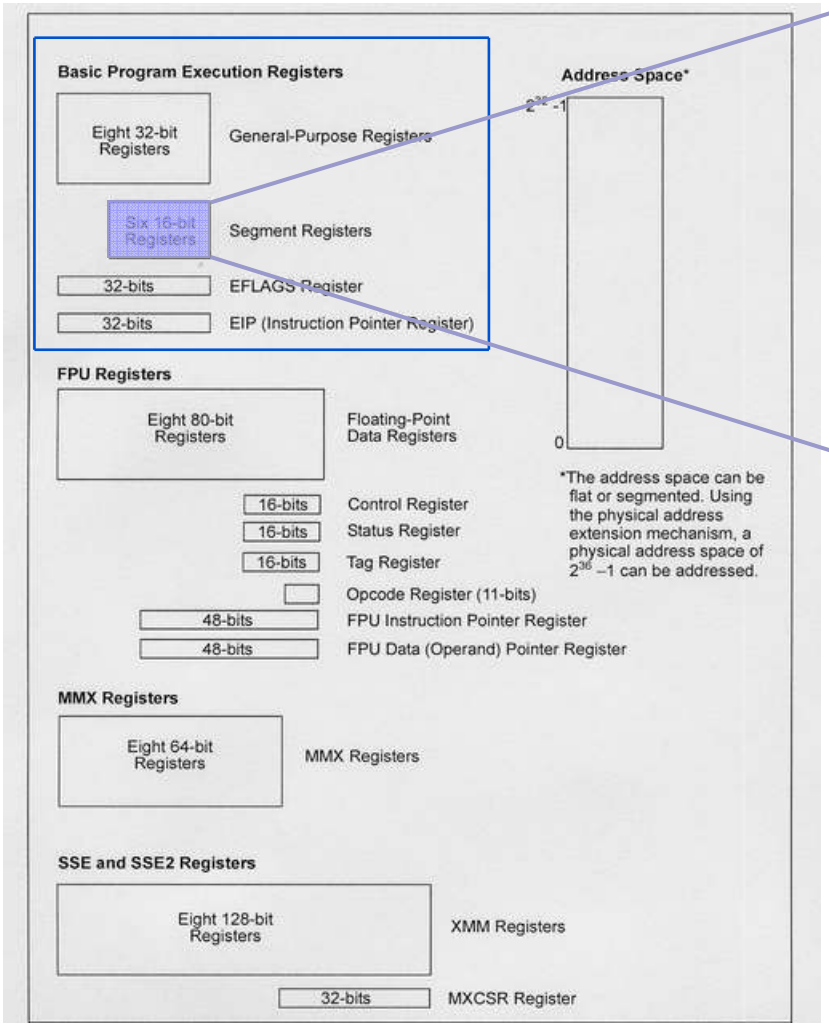
- Dodeljevalna in zamenjalna strategija sta bolj zapleteni in sta vgrajeni v operacijski sistem. Za dodeljevanje in zamenjevanje strani se največ uporabljata dva algoritma:
  - WS (working-set) algoritem
  - PFF (page-fault frequency) algoritem

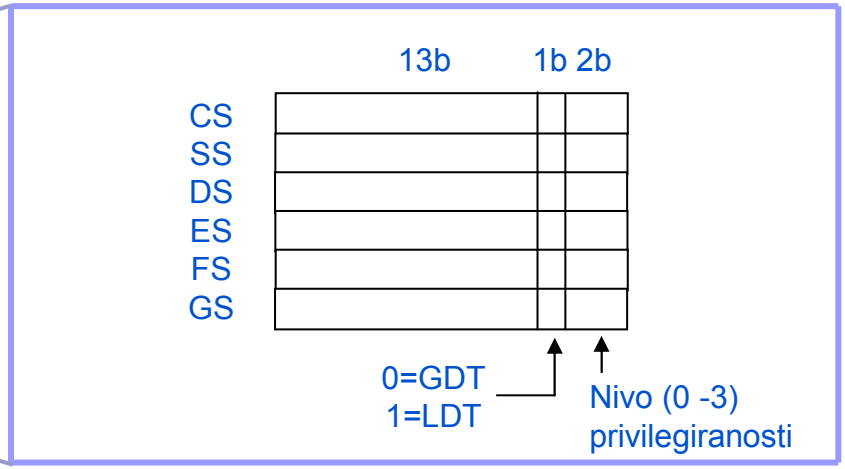
11.9 Navidezni pomnilnik pri procesorju Pentium 4

- Pentium 4 ima vgrajeno podporo za zmogljiv navidezni pomnilnik, ki omogoča vse tri načine realizacije: ostranjevanje, čisto segmentacijo in segmentacijo z ostranjevanjem.
- Velikost strani je lahko 4KB ali 4MB, izbira je programska.
- Največja velikost segmenta je 1MB (v inkrementih po 1B) ali 4GB (v inkrementih po 4KB, kar pomeni do 1M strani v segmentu)

- Dolžina navideznega naslova je 46 bitov, dolžina fizičnega naslova pa 32 bitov ali 36 bitov(programska izbira).
- Preslikava navideznega naslova v fizični naslov poteka preko tabele segmentov, v kateri ima vsak segment 64-bitni deskriptor, v katerem je naslov segmenta in njegova dolžina ter kontrolni biti.
- Dostop do segmenta določa zgornjih 13 bitov v enem od 6 segmentnih registrov (CS-Code Segment, DS-Data Segment, …)

Registri procesorja Intel P4 v 32-bitnem načinu delovanja





CS - Kazalec na ukazni segment (Code segment)
SS - Kazalec na skladovni segment
DS - Kazalec na podatkovni segment
ES - Kazalec na dodatni podatkovni segment
FS - Kazalec na podatkovni segment 2
GS - Kazalec napodatkovni segment 3

GDT - Globalni segment (uporabi se globalna tabela deskriptorjev - ena sama tabela)
LDT - Lokalni segment (uporabi se lokalna tabela deskriptorjev - vsak program ima svojo)

- V segmentnem registru je tudi informacija o nivoju privilegiranosti in bit za izbiro med globalnimi in lokalnimi segmenti.
- Preslikava poteka preko globalne tabele segmentov, ki je ena sama, ali preko lokalne tabele segmentov, ki jo ima vsak program.
- Pri preslikavi preko tabele segmentov dobimo 32-bitni linearni naslov, ki se preko dveh nivojev tabel strani preslika v 32-bitni fizični naslov.

- Sistem zaščite pomnilnika je pri procesorju Pentium 4 sestavni del navideznega pomnilnika in podpira štiri nivoje privilegiranosti (pri večini procesorjev 2 nivoja).
- Najbolj privilegiran je nivo 0, najmanj pa nivo 3, ki se imenuje tudi uporabniški nivo.
- Nivo privilegiranosti programa, ki se izvaja, je določen z dvema bitoma v segmentnem registru CS, ki je del navideznega naslova. Nivo privilegiranosti v ostalih segmentnih registrih se uporablja pri dostopih do operandov.

- Vsak segment ima v svojem deskriptorju določen svoj nivo privilegiranosti, ki se pri dostopih primerja z nivoji privilegiranosti v segmentnih registrih.

- Dostop je možen samo, če so izpolnjeni določeni pogoji, sicer se sproži past GP (General Protection).

