HIPOTETIČNI RAČUNALNIK

LASTNOSTI:

- ✓ 32-bitni računalnik;
- ✓ 3-operandni registrsko-registrski (load/store) RISC računalnik;
- ✓ dolžina pomnilniške besede-1bajt;
- ✓ dolžina pomnilniškega naslova je 32 bitov;
- ✓ pomnilniški operandi 8,16 ali 32-bitni (FP ukazov nima);
- ✓ sestavljeni pomnilniški operandi →po pravilu debelega konca;
- ✓ numerični operandi so celoštevilski v fiksni vejici in dvojiškem komplementu;
- ✓ vsi ukazi so dolgi 32 bitov;
- ✓ podatkovno in naslovno vodilo: 32-bitna;
- ✓ vse ALE operacije se izvršijo v eni urini periodi;
- ✓ poravnanost sestavljenih pomnilniških operandov je obvezna;
- ✓ uporablja pomnilniško preslikan vhod/izhod.

NAČINI NASLAVLJANJA:

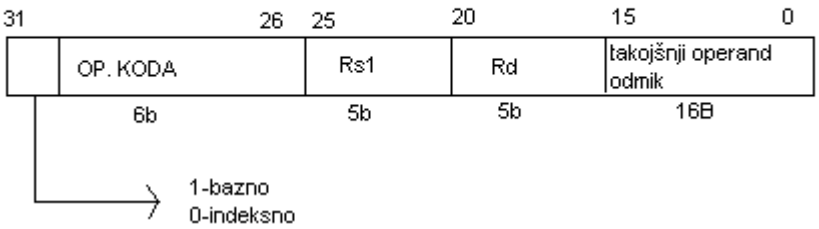
- ✓ **TAKOJŠNJE NASLAVLJANJE:** Takojšnji operand je dolg 16 bitov, predstavljen je kot 16-bitno predznačeno število zapisano v dvojiškem komplementu. Pred uporabo se ta operand z razširitvijo predznaka vedno pretvori v 32-bitnega.

- ✓ **BAZNO NASLAVLJANJE:** Odmik Di je dolg 16 bitov in je enako kot takojšnji operand predstavljen kot 16-bitno predznačeno število zapisano v dvojiškem komplementu. Samo ukaza load/store, odmik je samo pozitiven.

- ✓ **PC-relativno:** pri skočnih ukazih: odmik je spet enak zgornjim dvema, samo da je tukaj lahko pozitiven in negativen.

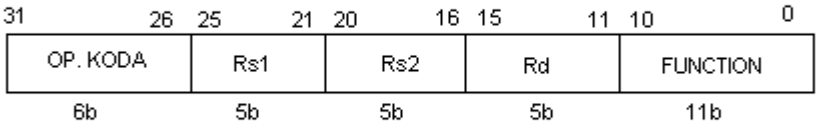
FORMAT STROJNIH UKAZOV

FORMAT 1:



Tak format se uporablja pri ukazih LOAD/STORE, pri skokih in pri klicih (PSP).

FORMAT 2:



Tak format se uporablja pri ALE ukazih.

PREVZEM IN IZVEDBA UKAZA (npr.: 5 korakov)

* **1. korak: prevzem (branje) ukaza iz pomnilnika;** ta korak je potreben pri vseh ukazih. Vsebina PC se preko multiplekserja prenese na pomnilniške naslovne signale. Nato se iz pomnilnika preberejo 4 sosednje pomnilniške besede in prenesejo v ukazni register IR. Pri HIP je poravnanost pri dostopih do pomnilnika obvezna.
* **2. korak: dekodiranje ukaza (ko je v ukaznem registru) in dostop do registrov.** V tem koraku je v registru IR že ukaz. Biti 21-25 in 16-20 določajo registra Rs1 in Rs2 (v formatu 1) oziroma Rd (v formatu 2) in se uporabijo za izbiro registrov v registrskem bloku.Uporaba bitov iz ukaza se običajno označuje kot dekodiranje ukaza. Vsebina izbranih registrov se prebere v vmesna registra A in B.
* **3. korak: izvrševanje operacije.** V tem koraku se naredi računanje dejanskega naslova ali ALE operacije (v obeh primerih so bili operandi pripravljeni v prejšnjih dveh korakih). Zaradi LOAD/STORE zasnove HIP računalnika ne more priti do primera ko bi morali istočasno računati dejanski naslov in izvajati ALE ukaz. Med registri IR, A in B določi kontrolna enota na podlagi operacijske kode dva, katerih vsebina se postavi na vodili S1 in S2. Določi tudi kontrolne signale za ALE operacijo, ki se bo opravila na teh dveh operandih. Rezultat se preko vodila D zapiše v register, ki ga kontrolna enota ravno tako določi iz operacijske kode.
* **4. korak: dostop do pomnilnika.** Ta korak je potreben samo pri ukazih load in store ter pri ukazu TRAP. Pri drugih ukazih se preskoči.
* **5. korak: shranjevanje rezultata.** Ta korak je potreben samo pri ukazih CALL, LOAD, ALE, MOVER in TRAP .

KONTROLNA ENOTA

Podatkovna enota je pasivna-naredi samo tisto, kar od nje zahtevajo kontrolni signali. Njena zgradba sicer določa, kateri signali sprožijo neko operacijo in kako se ta naredi, ne pa tudi kdaj in v kakšnem vrstnem redu. Podatkovna enota je lažji in kontrolna enota je težji del realizacije CPE.
Delovanje kontrolne enote lahko podamo z ***diagramom prehajanja stanj.*** Diagram ima končno število stanj in povezave, ki določajo prehode med stanji. Prehod iz enega stanja

v naslednje (ki je lahko isto) se zgodi ob aktivni fronti urinega signala. Pri vsakem diagramu prehajanja stanj je definirano tako imenovano prvo ali začetno stanje, v katerem se vse skupaj začne. Pri računalnikih je začetno stanje običajno prevzem ukaza.
Naprava, ki prehaja skozi končno število stanj v skladu s pravili, ki jih vsebuje diagram, je v teoretičnem računalništvu znana pod imenom končni avtomat ali stroj s končnim številom stanj.

Za realizacijo kontrolne enote se danes uporabljata dva osnovna načina:

TRDO OŽIČENA LOGIKA
Pri trdo ožičeni kontrolni enoti (hardwired control unit) se diagram prehajanja stanj pretvori v ustrezno povezavo kombinatoričnih in sekvenčnih digitalnih vezij. V najenostavnejši obliki so to vrata flip-flopi. Ime "trdo ožičena« se uporablja zato, ker njeno delovanje določajo žične povezave med elementi. Te povezave so fiksne ali trde in jih je težko spreminjati. Če želimo spremeniti delovanje trdo ožičene kontrolne enote, je potrebno fizično spreminjanje povezav.

MIKROPROGRAMIRANJE
Pri mikroprogramski kontrolni enoti (microprogrammed control unit) je kontrolna enota narejena kot majhen računalnik. Diagram prehajanja stanj se pretvori v program, ki mu pravimo mikroprogram-vsako stanje diagrama je v programu en mikroukaz. Podobno kot pri diagramu prehajanja stanj mora biti tudi tu definiran prvi mikroukaz, s katerim se vse skupaj začne. Drugače kot pri trdo ožičeni kontrolni enoti, je delovanje sedaj določeno s programom, ki je shranjen v pomnilniku kontrolne enote. Temu pomnilniku pravimo mikroprogramski pomnilnik ali kontrolni pomnilnik. Spreminjanje vsebine mikroprogramskega pomnilnika je preprosta zadeva. Z drugačnim programom lahko popolnoma spremenimo izvrševanje ukazov, ne da bi fizično posegli v kontrolno enoto.

PREKINITVE IN PASTI

Beseda prekinitev (interrupt) se uporablja za označevanje dogodka, ki povzroči, da CPE začasno preneha izvajati tekoči program in prične izvajati nek drug program, ki mu pravimo prekinitveno servisni program. PSP je običajno kratek in pogosto je trajanje njegovega izvajanja omejeno, da ne moti ostalih programov. Zahteva za prekinitev pride v CPE od zunaj. Prekinitve so osnovni mehanizem, s katerim lahko drugi deli računalnika pridejo do uslug CPE.

Posebna vrsta prekinitve so pasti. Past (trap) je prekinitev, ki jo zahteva sama CPE ob nekem nenavadnem dogodku (deljenje z 0, izvajanje neobstoječega ukaza) ali celo na zahtevo programerja. Pasti, ki jih sproži programer s posebnim ukazom, se označujejo kot programske pasti (ali prekinitve), pasti, ki se sprožijo ob izpolnitvi nekega pogoja, pa so pogojne pasti.

Razlika med prekinitvami in pastmi je v naslednjem: gledano iz CPE pridejo pasti »od znotraj«, prekinitve pa »od zunaj«. Odziv CPE na oba dogodka je podoben, zato nekateri proizvajalci (npr. IBM) sploh ne delajo razlike med njima. Prekinitve in pasti so koristne, ker omogočajo preusmeriti delo CPE, ne da bi ta morala stalno preverjati stanje potencialnih uporabnikov njenih uslug.

Dogajanje pri prekinitvah je simbolično prikazano v knjigi (str. 229)

Iz diagrama sledi, da razlikujemo naslednja štiri stanja:

1. Normalno izvrševanje tekočega programa;
2. Shranjevanje stanja CPE ob pojavu zahteve za prekinitev (če je ta omogočena);
3. Skok na PSP in izvrševanje njegovih ukazov. Tudi ta program se lahko prekine-prekinitvam v PSP pravimo vgnezdene prekinitve;
4. Vrnitev iz PSP-ja in obnovitev stanja CPE – to mora biti enako tistemu na začetku druge točke.

Vključitev sposobnosti za prekinjanje v CPE ni vedno preprosta. Čeprav je prekinitev razmeroma redek dogodek (tipično pride ena vsakih 1000 ali več ukazov), je nemogoče napovedati, kdaj bo do nje prišlo. CPE mora biti zgrajena tako, da je na nek način mogoče rešiti naslednjih pet problemov:

Kdaj CPE reagira na prekinitveno zahtevo?
Najenostavnejšo in tudi najpogostejšo rešitev dobimo, če CPE reagira na prekinitveno zahtevo ali past šele potem, ko dokonča izvrševanje tekočega ukaza (pred prevzemom naslednjega). V tem primeru lahko nevidnost prekinitve zagotovimo tako, da se ohrani samo stanje programsko dostopnih registrov. Ob prevzemu ukaza je kontrolna enota vedno v začetnem stanju 0, ki ga je lahko obnoviti. Drugače je, če dovolimo prekinjanje v poljubnem stanju – takrat moramo shraniti stanje vseh registrov, ne samo programsko dostopnih.

Če se CPE odzove na prekinitveno zahtevo šele po dokončanju tekočega ukaza, potrebujemo tudi sredstvo, s katerim programer lahko onemogoči (maskira) in ponovno omogoči odziv CPE na prekinitvene zahteve. Sposobnost nadzora nad prekinitvami je nujno potrebna zaradi več razlogov. Računalniki so zato narejeni tako, da so po vklopu prekinitve vedno onemogočene. Še bolj očiten razlog za nadzor nad prekinitvami je sam prekinitveno servisni program: če pride do nove prekinitve preden prekinitveno servisni program shrani registre, bo nova prekinitev povzročila, da se bo izgubil programski števec PC.

Kako zagotoviti »nevidnost« prekinitve?
Osnovno pravilo pri prekinitvah je, da se po vrnitvi iz PSP prekinjeni program lahko nadaljuje, kot da prekinitve ni bilo (če zanemarimo čas, v katerem je prekinjeni program stal). Temu pravimo nevidnost ali transparentnost prekinitve. Tako je mogoče doseči, da CPE izmenično izvaja poljubno število programov, ne da bi to vplivalo na njegove rezultate.

CPE mora biti zgrajena tako, da je mogoče zagotoviti nevidnost prekinitev. To zagotovimo tako, da je stanje vseh registrov v CPE ob vrnitvi enako kot je bilo ob vstopu v PSP – samo v tem primeru prekinjeni program ne bo čutil posledic prekinitve. Najmanj kar mora za zagotovitev nevidnosti ob prekinitvi narediti CPE je, da shrani programski števec PC. Če tega ne stori, se seveda ni mogoče vrniti v prekinjeni program. Za shranjevanje in obnavljanje tistih registrov, katerih stanje prekinitveno servisni program spremeni, običajno poskrbi kar PSP sam.

Kje se dobi naslov prekinitvenega servisnega programa?

Pri pasteh, ki imajo izvor v CPE, je to vprašanje trivialno: ker je izvor zahteve CPE, ta lahko tvori naslov za vsako past. Drugače je pri prekinitvah, ki prihajajo od zunaj. Pri njih potrebujemo informacijo, ki pove, kateri prekinitveno servisni program naj se uporabi. Ta informacija pravzaprav pove, katera naprava je zahtevala prekinitev; če to vemo, vemo tudi, kateri servisni program naj se uporabi. Temu problemu pravimo tudi prepoznavanje naprave.

Če ima CPE več prekinitvenih vhodov in je na vsakega vedno priključena samo ena naprava, je ta problem trivialen. CPE ve, od kod je prišla zahteva in ne potrebuje dodatne informacije za razpoznavanje. Drugače je takrat, kadar je na en prekinitveni vhod priključenih več naprav.

Najpogostejša rešitev je PROGRAMSKO IZPRAŠEVANJE (polling). V CPE je vgrajen fiksen naslov prekinitveno servisnega programa. Ta program v določenem vrstnem redu bere ustrezne registre vsake V/I naprave (vsaka naprava ima register, v katerem eden od bitov pove, ali je zahtevala prekinitev). Pri prvi napravi, za katero se ugotovi, da je zahtevala prekinitev, izvrši program skok na njen prekinitveni servisni program. Vrstni red branja registrov določa prioriteto prekinitvenih zahtev. Slabost programskega izpraševanja je počasnost odziva na prekinitveno zahtevo. Dosti boljše so rešitve, pri katerih se v CPE pošlje informacija, iz katere ta prepozna izvor zahteve. Pošiljanje te informacije je mogoče narediti na več načinov. Med najbolj znanimi načini so tako imenovane vektorske prekinitve. To je skupno ime za vse načine prepoznavanja prekinjajoče naprave, pri katerih naprava pošlje v CPE informacijo o naslovu njenega prekinitveno servisnega programa. To pošiljanje se naredi v prekinitvenem prevzemnem ciklu, s katerim CPE obvesti V/I naprave, da naj pošljejo informacijo o izvoru prekinitve.
Pri nekaterih računalnikih je informacija, ki jo pošlje naprava, kar naslov – bolj pogosto pa je del naslova. Naslovu, na katerem je shranjen naslov prekinitveno servisnega programa, pravimo prekinitveni vektor ali vektorski naslov. Naslov se uporablja kot kazalec v tabelo, ki vsebuje naslove prekinitvenih servisnih programov. Če naprava pošlje samo del naslova, iz katerega se po nekem pravilu izračuna vektorski naslov, pravimo temu številka prekinitvenega vektorja.

Prioriteta?
Če ima CPE več prekinitvenih vhodov in tudi, če je na en vhod priključenih več naprav, je potrebno na nek način določiti prioriteto prekinitvenih zahtev. Ta določa, katera od prekinitvenih zahtev se bo upoštevala prva, če je istočasno prisotnih več zahtev.
Nekatere CPE imajo vgrajeno podporo za vgnezdene prekinitve, ki omogoča, da se med izvajanjem prekinitvenega servisnega programa nekatere prekinitvene zahteve upoštevajo, druge pa ne, odvisno od prioritete zahteve. CPE ima register, s katerim je določen trenutni prioritetni nivo CPE; tega je mogoče spreminjati s programom. CPE se odzove samo na tiste prekinitvene zahteve, katerih prioriteta je višja od njene lastne. Če postavimo prioritetni nivo CPE na najvišjo vrednost, dosežemo onemogočitev vseh prekinitev. Tudi če imamo na en prekinitveni vhod priključenih več naprav, je potrebno določiti prioriteto prekinitvenih zahtev. Že omenjeno programsko izpraševanje je ena možnost. Boljša rešitev pa je tako imenovana marjetična veriga. Ideja marjetične verige je v tem, da naprava, ki ni zahtevala prekinitve spusti signal INTAi (interrupt acknowledge-potrditveni signal) naprej v naslednjo napravo, medtem ko naprava, ki je zahtevala prekinitev, signalu zapre pot, tako da ta ne pride do drugih naprav.

Potrjevanje prekinitve?

Potrjevanje prekinitve je obveščanje naprave o tem, da je bila njena prekinitvena zahteva upoštevana. Potrebno je zato, ker ni mogoče natančno napovedati, kako hitro bo CPE reagirala na neko prekinitveno zahtevo – to je odvisno od tekočega programa in od drugih prekinitvenih zahtev. Zato mora naprava držati prekinitveni vhod v aktivnem stanju tako dolgo, dokler njena prekinitev ni potrjena. Če naprava ne dobi potrditve, ostane zahteva stalno aktivna. Pri prekinitvi, ki je bila že servisirana, to pomeni, da se bo servisirala še enkrat, kar ni potrebno. Brez potrjevanja pride do neskončnega zaporedja prekinitev, kar je narobe.

Potrjevanje lahko naredimo programsko ali strojno;
Programsko: tukaj se potrditev naredi tako, da prekinitveni servisni program bere ali piše v nek register krmilnika naprave. To je za krmilnik znak, da je prekinitvena zahteva upoštevana, in da naj umakne svojo prekinitveno zahtevo.
Strojno: realizirano tako, da CPE ob upoštevanju prekinitvene zahteve s posebnim kontrolnim signalom obvesti napravo, da je njena zahteva upoštevana.
Vektorske prekinitve: tukaj naprava, ki v prekinitvenem prevzemnem ciklu pošlje vektor ali številko vektorja, je namreč s tem že obveščena, da je njena zahteva upoštevana.

CEVOVODNA CPE

Z besedo cevovod se označuje način realizacije CPE, pri katerem se naenkrat izvršuje več ukazov tako, da se posamezni koraki izvrševanja prekrivajo. Cevovod izvršuje ukaze na način, ki je podoben tekočemu traku pri proizvodnji avtomobilov. Tako kot pri tekočem traku, se izvrševanje vsakega ukaza razdeli na manjše dele ali podoperacije – za vsako od teh podoperacij je potreben samo majhen del celotnega časa za izvrševanje ukaza. Vsako od podoperacij opravi točno določen del cevovoda, ki mu pravimo stopnja cevovoda ali segment cevovoda. Stopnje so povezane ena z drugo tako, da tvorijo nekakšno cev: ukazi na eni strani vstopajo vanjo, potujejo skozi stopnje, v katerih se obdelujejo in izstopajo na drugi strani. V določeni urini periodi se v CPE izvaja toliko ukazov, kolikor je stopenj, vsak od ukazov v določeni fazi (podoperacij) dodelave.

Pri idealno uravnoteženi cevovodni CPE z N stopnjami je zmogljivost N-krat večja kot pri ne-cevovodni CPE. Vsak posamezen ukaz se sicer ne opravi nič hitreje kot pri ne-cevovodni CPE – zato pa se jih naenkrat obdeluje N. Na izhodu iz cevovoda dobimo v istem času N-krat več izvršenih ukazov kot pri ne-cevovodni CPE. Resnični cevovodi nikoli niso idealno uravnoteženi in tudi samo delovanje cevovoda (prehodi iz ene stopnje v drugo) porabi nekaj časa. Če pustimo vse ostalo enako kot pri ne-cevovodni CPE, povečanje zmogljivosti ne bo N-kratno, čeprav je lahko blizu te vrednosti. V celoti gledano lahko rečemo, da se pri cevovodni CPE število ukazov, ki jih CPE izvrši v danem času, poveča zaradi naslednjih dveh vzrokov:

1. **Manjše število urinih period na ukaz**. Ker pride iz cevovoda N-krat več izvršenih ukazov, je v idealnem primeru povprečno število urinih period na ukaz CPI N-krat manjše kot pri ne-cevovodni CPE. V danem času je izvršenih več ukazov in temu ustrezno hitreje je izveden program.
2. **Krajša urina perioda**. Ta možnost je danes bolj teoretična kot praktična. Če uspemo izvrševanje ukazov razdeliti na zelo enostavne podoperacije, je te v principu mogoče opraviti v krajšem času. Pri resničnih CPE je to redko, ker je število možnosti za delitev izvrševanja ukazov na podoperacije običajno

majhno. Povečevanje števila stopenj N ima namreč za posledico povečanje izgube zaradi tako imenovanih cevovodnih nevarnosti.

CEVOVODNE NEVARNOSTI

Med delovanjem cevovoda prihaja do cevovodnih nevarnosti, zaradi katerih se mora cevovod ustaviti in počakati da nevarnost mine. Razlikujemo tri vrste cevovodnih nevarnosti:

1. ***Strukturne nevarnosti***: Do teh nevarnosti pride, kadar več stopenj cevovoda v neki urini periodi potrebuje isto enoto (registre, ALE, pomnilnik). Če imamo samo eno enoto, mora ena od stopenj počakati.
   Strukturne nevarnosti je mogoče popolnoma odpraviti z bolj zmogljivim predpomnilnikom in tako, da vgradimo dovolj veliko število enot. Pri HIP do strukturnih nevarnosti ne prihaja.
2. ***Podatkovne nevarnosti***: Do njih prihaja kadar nek ukaz potrebuje operand, ki še ni dostopen, ker zaradi izvrševanja ukazov v cevovodu predhodni ukaz še ni dokončan Bolj pravilno ime za njih bi bilo operandne nevarnosti. Do njih prihaja zato, ker izvrševanje v cevovodu ne omogoča vedno enakega vrstnega reda pri dostopu do operandov kot običajno zaporedno izvrševanje. Teh nevarnosti se izognemo s *premoščanjem* (*data forwarding* ali *bypassing*). Bistvo premoščanja je v tem, da je mogoče operand iz vsake stopnje cevovoda pripeljati nazaj v vsako od drugih stopenj. V nekaterih primerih to ni dovolj in potrebno je vgraditi dodatno logiko za *zaklenitev cevovoda* (iz potrebe po čakanju).
3. ***Kontrolne nevarnosti***. Ta vrsta nevarnosti povzroča pri večini računalnikov bistveno večjo izgubo kot strukturne in podatkovne nevarnosti. Do kontrolne nevarnosti pride pri vseh ukazih, ki spremenijo vsebino programskega števca PC. To se dogaja pri kontrolnih ukazih, ki jih delimo v pogojne in brezpogojne skoke, klice in vrnitve. Ker so, gledano s stališča cevovoda, klici in vrnitve enaki brezpogojnim skokom, se kontrolnim ukazom pri obravnavi cevovoda pogosto pravi kar skoki.
4.

Če je pogoj za skok izpolnjen: se pri teh ukazih v PC prenese tako imenovani skočni naslov. Skočni naslov se izračuna v stopnji EX, ki ga zapiše v PC. Kadar stopnja EX povzroči spremembo PC, je vsebina stopenj IF in ID neveljavna. V teh dveh stopnjah sta namreč ukaza, ki sledita skoku in se ne smeta izvršiti – to je kontrolna nevarnost.
Najpreprostejši način za reševanje te nevarnosti je, da se v urini periodi, v kateri stopnja EX spremeni PC, v stopnji IF in ID vstavi mehurček. To je ekvivalentno čakanju v trajanju dveh urinih period, ki se označuje z izrazom skočna zakasnitev.

Rešitve:

- ✓ **Zaklepanje cevovoda**
- ✓ **Statična predikcija in skočne reže**: Pri tej rešitvi sodeluje prevajalnik na podoben način kot pri cevovodnem razvrščanju. Prevajalnik skuša za pogojne skoke napovedati bolj verjeten rezultat preverjanja pogoja za skok. Ker to napoved naredi prevajalnik, se med izvrševanjem programa ne spreminja. Zato se ta rešitev označuje z izrazom statična predikcija. Pri rešitvi z zakasnjenimi skoki je pomemben pojem tako imenovanih skočnih rež. Ukazi, ki so v programu takoj za skokom, so v skočnih režah

– število rež je enako številu stopenj cevovoda, ki so pred stopnjo v kateri se povzroči prenos skočnega naslova v PC. Pri HIP se to zgodi v stopnji EX, kar pomeni, da imamo dve skočni reži.

- ✓ **Dinamična predikcija**: Napovedovanje izpolnitve pogojev za skok, ki ga dela prevajalnik pri zakasnjenih in razveljavitvenih skokih, je statično. Določi se pred začetkom izvrševanja programa in se med izvrševanjem ne spreminja. Rešitve, pri katerih se napovedovanje spreminja med izvrševanjem programa v skladu z »obnašanjem« skočnega ukaza, se označuje kot dinamična predikcija. Dinamična predikcija je očitno boljša od statične, ker se prilagaja dogajanju v programu.

KAKO ZMANJŠATI CPI POD ENA (CPI<1)?

Cevovod in načini za odpravljanje cevovodnih nevarnosti, omogočajo, da se CPI približa vrednosti 1. Če želimo CPI zmanjšati pod 1, lahko to dosežemo le tako, da se v vsaki urini periodi prevzame več kot en ukaz. Ker se mora v vsaki urini periodi več ukazov tudi izvršiti (ne samo prevzeti), se procesorji s to sposobnostjo označujejo z izrazom več-izstavitveni procesorji. S tem se želi povedati, da se v vsaki urini periodi izstavi v stopnje EX več ukazov. Te procesorje danes delimo v dve vrsti:

Superskalarni procesorji: To so računalniki, ki opravijo v urini periodi več kot en skalaren ukaz.CPE je narejena tipično tako, da prevzame v vsaki urini periodi 2 do 4 ukaze, preveri ali so med seboj neodvisni in izvrši naenkrat samo tiste, ki so neodvisni (vsakega v svojem cevovodu). Ostali morajo čakati. Pogoje za paralelizem preverja CPE in ne prevajalnik – osnovna značilnost superskalarnih računalnikov.

VLIW procesorji: Preverjanje neodvisnosti med ukazi za razliko od prejšnjega opravi prevajalnik (ki ima za to več časa) in ne CPE. V vsakem ukazu je podanih več operacij (do 8 ali več).

Razlika med superskalarnimi in VLIW procesorji je v načinu prevzemanja in izstavljanja ukazov. Prevzemanje ukazov in njihovo izstavljanje sta ločeni operaciji, ki ju je mogoče izvajati statično ali dinamično. Pri prevzemanju je razlika v vrstnem redu, po katerem se ukazi berejo iz pomnilnika. Tu gre pravzaprav za razvrščanje ukazov.

Razlikujemo:

Statično razvrščanje: pri katerem prevzema procesor ukaze v natanko takem vrstnem redu, kot so v programu. Temu načinu se pravi tudi in-order. Spreminjanje vrstnega reda, če je to možno, lahko opravlja prevajalnik in s tem omogoča boljše izkoriščanje procesorja. Njegova slabost je, da ne pridejo v istočasno izvrševanje ukazi, pri katerih se med izvajanjem programa pokaže, da bi bilo to mogoče, vendar prevajalnik tega ne more ugotoviti in zato ne razvrsti ukazov.

Dinamično razvrščanje: pri katerem prevzema procesor ukaze v vrstnem reku, ki je lahko drugačen od tistega v programu. Temu načinu se pravi tudi out-of-order. Spreminjanje vrstnega reda opravlja procesor; kadar je neka od enot procesorja prosta, išče ukaze, ki jih je mogoče poslati v izvrševanje. Prednost dinamičnega razvrščanja se

pokaže pri zgrešitvah v ukaznem (in tudi operandnem) predpomnilniku; med čakanjem na prenos bloka v predpomnilnik lahko procesor prevzame in izvrši nek drug ukaz. To je mogoče samo, če je temu ustrezno narejen predpomnilnik. Pri iskanju ukazov je seveda potrebno uporabljati dinamično predikcijo skokov, brez nje bi bilo skoraj nemogoče najti primeren ukaz. Kar predikcija ni vedno pravilna, ni zanesljivo, da bo tako prevzet in izvršen ukaz v resnici potreben. Dinamično razvrščanje, v kombinaciji s predikcijo skokov, se zato imenuje špekulativno izvrševanje.

Podobna razlika obstaja tudi pri izstavljanju ukazov, kjer razlikujemo:

Statično izstavljanje je način, pri katerem je vrstni red izvrševanja ukazov (njihovega izstavljanja v stopnjo EX) določen že pri prevzemu ukaza. Procesor glede vrstnega reda, po katerem gredo ukazi v izvrševanje, ne odloča o ničemer. Procesorji s statičnim izstavljanjem imajo običajno tudi statično razvrščanje, čeprav to ni nujno.

Dinamično izstavljanje je način, pri katerem vrstni red izstavljanja ukazov določa logika v procesorju. Procesor pregleduje ukaze od trenutne točke izvrševanja naprej in išče take, ki niso odvisni od trenutno izvršujočih; če tak ukaz najde, ga izstavi v izvrševanje. Procesorji z dinamičnim izstavljanjem imajo lahko statično ali dinamično razvrščanje ukazov.

MERJENJE ZMOGLJIVOSTI CPE

Kako merimo zmogljivost CPE? Edini zanesljivi odgovor je čas izvrševanja programa, ki ga merimo v sekundah na program (10-krat krajši čas izvrševanja pomeni 10-krat večjo zmogljivost). Ta čas izrazimo kot

$$CPE_{čas} = \text{število ukazov} * CPI * t_{CPE}$$

Kjer CPI pomeni povprečno število urinih period na ukaz. Zmogljivost CPE je torej odvisna od treh lastnosti. Te tri lastnosti so med seboj odvisne – če spremenimo eno, to vpliva tudi na druge dve.

1. **Urina perioda t_{CPE}.** Nanjo vpliva hitrost in število digitalnih vezij, iz katerih je narejena CPE, ter tudi zgradba kontrolne in podatkovne enote.
2. **CPI.** Na povprečno število urinih period, ki so potrebne za izvršitev ukaza, vpliva zgradba kontrolne in podatkovne enote ter število in vrsta ukazov.
3. **Število ukazov, v katere se prevede program.** Nanj vpliva število in vrsta ukazov ter lastnosti prevajalnika.

Vedno se moramo zavedati, da je edino merilo za zmogljivost CPE čas, ne pa katerakoli od teh treh lastnosti. Če npr. povečamo število ukazov in spremenimo njihovo zgradbo, da bi zmanjšali število ukazov, v katere se prevede ukaz, ima to lahko za posledico tako kontrolno in podatkovno enoto, ki zahteva daljšo urino periodo.

SINTETIČNI BENCHMARK PROGRAMI: programi, s katerimi primerjamo več računalnikov za merjenje zmogljivosti.

GLAVNI POMNILNIK IN PREDPOMNILNIK

Z izrazom glavni pomnilnik označujemo prostor, iz katerega CPE jemlje ukaze in operande in kamor shranjuje rezultate. Velikost je pomembna, ker majhen pomnilnik omejuje programerje in zmanjšuje število problemov, ki jih je na danem računalniku mogoče rešiti. Hitrost je pomembna, ker mora pri počasnem pomnilniku CPE čakati; s tem se seveda podaljšuje čas reševanja problemov. Osnovna težava pri gradnji glavnega pomnilnika je, da si zahtevi po velikosti in hitrosti nasprotujeta: velikost zmanjšuje hitrost in obratno. Velik in hiter glavni pomnilnik je polet tega drag in ga je pri današnjem stanju pomnilniške tehnologije tudi težko narediti. Zato se je že v 1960-tih letih pojavila zamisel, da se podmnožica glavnega pomnilnika shrani v majhen in hiter pomnilnik. Temu pomnilniku pravimo predpomnilnik (cache).

Seveda se moramo takoj vprašati o smiselnosti te zamisli. Če bi bili naslovi, ki jih tvori CPE in V/I naprave, porazdeljeni naključno, bi bil iskani naslov redko v predpomnilniku in z njim ne bi pridobili nič. Vendar ni tako. Pri računalnikih naslovi niso porazdeljeni naključno, temveč velja zanje princip lokalnosti. Lokalnost pomnilniških dostopov omogoča, da se že pri razmeroma majhnih predpomnilnikih doseže bistveno povečanje hitrosti. Če izvzamemo zelo majhne računalnike, imajo zato danes vsi predpomnilnike. Še več: pogosto imajo celo dva nivoja predpomnilnikov. Manjši in hitrejši nivo L1 je vgrajen v CPE, večji in običajno nekoliko počasnejši L2 pa izven nje.

Pri današnjih računalnikih imamo namesto enega glavnega pomnilnika opraviti z več nivojsko pomnilniško hierarhijo. Ta ne obsega samo predpomnilnikov, temveč se pogosto razširja od glavnega pomnilnika naprej in vključuje tudi pomožni pomnilnik.

Glavni pomnilnik je tisti pomnilnik, do katerega ima CPE *neposreden dostop* tako, da poda naslov pomnilniške besede. Pri pomožnih pomnilnikih je dostop *posreden* preko V/I ukazov, ki najprej prenesejo zahtevano besedo v glavni pomnilnik (običajno se poleg zahtevane besede prenese še več sosednjih), šele nato je možen neposreden dostop.

Razlog za delitev pomnilnika je danes ekonomski, včasih pa je bil tehnološki.

Danes se glavni pomnilniki gradijo skoraj izključno iz elektronskih pomnilniških elementov, ki so narejeni v obliki v obliki majhnih silicijevih ploščic ali čipov. Vedno ni bilo tako in razvijalci so v preteklosti uporabljali veliko različnih tehnologij. Čeprav te niso več pomembne, imajo pomnilniški elementi nekatere lastnosti, ki so skupne vsem.

1) **CENA:** Cena pomnilnika se danes običajno podaja v dolarjih na Megabajt ($/MB). Dobimo jo tako, da ceno celotnega pomnilnika delimo z velikostjo pomnilnika, merjeno v MB. Pri tem moramo v ceno celotnega pomnilnika vključiti poleg cene pomnilniških celic tudi ceno vse dodatne elektronike in/ali mehanike, ki je potrebna za delovanje pomnilnika.
2) **HITROST DOSTOPA:** Zmogljivost nekega pomnilnika je določena s hitrostjo branja in pisanja informacije v pomnilnik. Kot mera za hitrost se običajno uporablja povprečni čas, ki je potreben za branje določene količine informacije. Ta čas imenujemo bralni čas dostopa, ali bolj pogosto samo čas dostopa (access time) in ga označujemo s t_a. Pri večini današnjih pomnilniških elementov je čas za pisanje približno enak času za branje, tako da razlikovanje pogosto ni potrebno. Čas dostopa je definiran kot čas, ki preteče od trenutka,

ko pomnilnik dobi naslov, do trenutka, ko informacija na vhodu pomnilnika (pisanje) ni več potrebna. Pri nekaterih vrstah pomnilnikov (taki so DRAM elementi, s katerimi se danes gradijo pomnilniki) mora po vsakem dostopu preteči nek čas (mrtev čas), preden se lahko prične naslednji dostop.

3) **NAČIN DOSTOPA:** Zelo pomembna lastnost vsakega pomnilnika je način izbire pomnilniške besede, do katere želi dostopiti. Glede na to se današnji pomnilniki delijo v dve skupini.

   - Prva skupina, ki je veliko večja in se uporablja daleč najbolj pogosto, je beseda podana z naslovom. Dostop z naslovom je tako običajen, da ti pomnilniki nimajo posebnega imena in bi jim lahko rekli **običajni pomnilniki.** Vsaka pomnilniška beseda ima svoj naslov, ki je nespremenljiv: ista beseda ima vedno isti naslov;
   - Pri pomnilnikih iz druge skupine besede nimajo naslovov – dostop do njih poteka preko vsebine besede. Pomnilnikom te vrste pravimo **asociativni pomnilniki.**

Običajne pomnilnike delimo glede na odvisnost časa dostopa od vrstnega reda naslovov, s katerimi se bere ali piše. Razlikujemo naslednje štiri primere:

a). **NAKLJUČNI DOSTOP** (random access). Pri tem načinu je čas dostopa do poljubne besede neodvisen od naslova pred tem naslovljenih besed. Drugače povedano, čas dostopa t_a je konstanten in znan vnaprej pri naključnem zaporedju naslovov. Konstanten čas dostopa ne glede na fizični položaj naslovljene besede in ne glede na prejšnje naslove, je glavna lastnost naključnega dostopa. (polprevodniški pomnilniki).

b). **ZAPOREDNI** (sekvenčni) **DOSTOP** (serial access). Pri tem načinu je čas za dostop do neke besede odvisen od naslova besede, do katere je bil narejen dostop tik pred tem. Če je bil npr. A naslov prejšnje besede, je takoj dostopna samo beseda z naslovoma A+1. Za dostop do poljubnega naslova B pa moramo najprej izvršiti dostop do vseh besed med A in B. Očitno je pri tem načinu čas dostopa t_a močno odvisen od zaporedja naslovov. Tipičen predstavnik pomnilnikov z zaporednim dostopom je magnetni trak

c). **KROŽNI DOSTOP** (rotational access). To je pravzaprav posebna vrsta zaporednega dostopa. Lahko si ga predstavljamo kot magnetni trak, ki je zlepljen v zanko. Povprečen čas dostopa t_a je pri krožnem načinu enak ½ periode vrtenja.

d). **DIREKTNI DOSTOP** (direct access). Gre za kombinacijo zaporednega in krožnega načina dostopa. Direktni dostop srečamo pri magnetnih in optičnih diskih s premičnimi glavami. Deluje tako, da se bralno-pisalna glava najprej premakne nad ustrezno sled (to je zaporedni dostop), nato pa imamo že opisani krožni način dostopa. Ker poteka premik glave preko sledi hitro, je pri enaki velikosti pomnilnika povprečen čas dostopa veliko krajši kot pri zaporednem ali krožnem dostopu.

4. **SPREMENLJIVOST VSEBINE:** Pri nekaterih vrsta pomnilnikov lahko pisalno operacijo izvršimo samo enkrat. Ko je informacija vpisana, je pri

normalni uporabi pomnilnika ne moremo več spremeniti. Danes so taki npr. CD-ROMi. Pomnilnike, katerih vsebine ne moremo spreminjati pri normalni uporabi, imenujemo **bralni pomnilniki.** (ROM).
Pri tako imenovanih **programabilnih ROMih** lahko s posebno napravo, ki ji pravimo programator, vpišemo v pomnilnik poljubno vsebino.
Nekatere vsebine (**EPROM**) je mogoče brisati (brisanje z UV svetlobo) in s programatorjem ponovno vpisati.
Pri najnovejših vrstah bralnih pomnilnikov (**EEPROM**) je mogoče električno brisanje.

Pomnilnike, pri katerih ob normalni uporabi z enako lahkoto beremo in pišemo imenujemo **bralno-pisalni pomnilniki**. Zanje se danes uporablja oznaka RAM (random access memory). Imamo **DRAM** (dinamični) in **SRAM** (statični) pomnilnik.

FLASH POMNILNIKI – te vrste pomnilnikov so varianta EEPROMA-brisanje celotne vsebine. Flash kartica je porazdeljena na bloke, brisanje poteka po posameznih blokih, ki so različne velikosti.

5. **OBSTOJNOST VSEBINE:** Fizikalni pojav, ki se izkorišča za hranjenje informacije, ki je pogosto nestabilen – shranjena informacija se s časom lahko izgubi, če tega z ustreznimi ukrepi ne preprečimo. Razlogov, zaradi katerih v današnjih pomnilnikih prihaja do izgube informacije je več. Med njimi so najpomembnejši naslednji trije:

   - **Destruktivno branje:** Primer pomnilnika z destruktivnim branjem so DRAMi. Pri njih je informacija shranjena v obliki naboja na majhnem kondenzatorju, ki je del pomnilniške celice. Ob branju se kondenzatorji vseh stolpcev izbrane vrstice izpraznijo in informacija se izgubi. Vsaki bralni operaciji mora zato slediti pisalna operacija, ki vzpostavi prejšnje stanje. Pisalna operacija se izvrši avtomatsko.
   - **Dinamično shranjevanje:** Tak pomnilnik ima lastnost, da se shranjena vrednost s časom sama spremeni iz stanja 1 v stanje 0 ali obratno. Tukaj je vsak bit informacije shranjen v obliki električnega naboja v majhnem kondenzatorju. Pri takih pomnilnikih se bo informacija izgubila, če je ne obnovimo. Postopku pravimo osveževanje (refresh). Pri današnjih DRAM čipih velikosti 64 Mbit je npr. potrebno vsak bit osvežiti najmanj enkrat na vsakih 64 milisekund. Osveževanje DRAMov je realizirano podobno kot že opisano obnavljanje zaradi destruktivnega branja. Vrstica, ki jo želimo osvežiti, se prebere in nato zapiše nazaj. Ker med osveževanjem dostop do DRAMa ni možen, se lahko izgubi nekaj časa.
   - **Odklop vira energije:** Ker je vir energije danes praktično vedno električna napetost, govorimo o odklopu napajanja. Pomnilniškim napravam, ki ohranijo informacijo tudi po odklopu napajanja pravimo, da so obstojne. Pomnilniki na osnovi magnetnega shranjevanja so običajno obstojni, medtem ko so polprevodniški DRAMi in SRAMi neobstojni.

6. **ZANESLJIVOST:** zelo pomemben podatek pri vsakem pomnilniku je njegova zanesljivost, ki jo merimo z verjetnostjo za pojav napake.

   Enota za merjenje zanesljivosti: BER (bit error rate)
   Npr.: pri disku BER=10^{-14} => 1 napačno prebran bit pri 10^{14} prebranih bitih.

Ločimo:

-mehka napaka (soft error), kjer se pomnilniška celica ne poškoduje, spremeni se vsebina;

-trda napaka (hard error), kjer se pomnilniška celica poškoduje in ostane neuporabna. Temu lahko rečemo tudi trajna napaka.

METABITI

Z imenom metabiti označujemo tiste bite v pomnilniški besedi, ki opisujejo pomen ostalih bitov te besede. Večina današnjih računalnikov metabitov nima, ali pa jih uporablja samo za detekcijo in korekcijo napak v glavnem pomnilniku – to so tako imenovani paritetni biti. Ker so pri današnjih računalnikih glavni pomnilniki veliki, verjetnost za pojav mehke ali trde napake ni več zanemarljiva in paritetni biti se pogosto uporabljajo.

PARITETNI BITI:

		P
SODA PARITETA=število vseh enic je sodo	1 0 1 0 1 1 0 1	1
LIHA PARITETA=število vseh enic je liho	1 0 1 0 1 1 0 1	0

ZAŠČITA GLAVNEGA POMNILNIKA

Najpreprostejši zaščitni mehanizem je par registrov, ki vsebuje spodnjo in zgornjo mejo naslova, ki pripada nekemu programu.
Nekoliko drugačno obliko takega načina zaščite dobimo, če je v enem registru začetni naslov programa, v drugem pa dolžina programa.
Enega od prvih zaščitnih mehanizmov te vrste predstavlja uporaba metabitov, ki za vsako pomnilniško besedo določajo, kateremu programu pripada.

DANAŠNJE REŠITVE ZAŠČITE

Rešitve, ki se uporabljajo danes, ne uporabljajo zaščite posameznih pomnilniških besed. Namesto tega je glavni pomnilnik razdeljen na dele velikosti 512B, 1024B, 2048B, 4096B, 8192B, ki so vsak zase zaščiteni. Tem delom običajno pravimo **bloki** ali **strani**. Vsakemu programu je dodeljen pomnilniški prostor, ki je enak celemu številu strani. Vsaka stran ima svoje zaščitne **metabite**, ki jim pravimo **zaščitni ključ**. Vsebina bitov zaščitnega ključa neke strani velja za vse pomnilniške besede v strani. Zaščitni ključi so shranjeni v posebnem pomnilniku, ki ni del glavnega pomnilnika in dostop do njih z običajnimi pomnilniškimi ukazi ni možen (dostopen samo za operacijski sistem).

HITROST DOSTOPA DO GLAVNEGA POMNILNIKA

Hitrost glavnega pomnilnika lahko ustrezno povečamo samo s kombinacijo več različnih rešitev:
-hitrejši pomnilniški elementi (pomnilniški čipi, krajši t_a),
-večje število naenkrat prenesenih bitov med CPE in glavnim pomnilnikom,
-predpomnilnik z več nivoji (L1, L2, L3),
-pomnilniško prepletanje.

PREDPOMNILNIK

Je majhen, hiter pomnilnik med CPE in glavnim pomnilnikom. Vsebina predpomnilnika je vedno podmnožica vsebine glavnega pomnilnika.

Pri cevovodno narejenih CPE, ki so danes v večini, imamo običajno dva predpomnilnika: v enem so ukazi, v drugem pa operandi. Razlog za to je potreba po istočasnem dostopu do ukazov in operandov, ki se pojavi pri cevovodu. Za računalnike z ločenim ukaznim in operandnim pomnilnikom smo rekli, da imajo Harvardsko arhitekturo. Računalnikov z ločenima predpomnilnikoma je danes več, kot tistih z enim samim. Če je pomnilnik en sam pravimo, da je homogen, sicer pa nehomogen.

DELOVNA MNOŽICA

Množico naslovov, ki jih v časovnem intervalu od t – T do t generira nek program, imenujemo delovna množica V(t,T). Vzemimo, da v tem času pride v glavni pomnilnik N naslovov, ki tvorijo zaporedje A(1) A(2)… A(N). Zaradi lokalnosti velja naslednje:

- velikost množice V(t,T) je veliko manjša od N;
- vsebina zaporedno si sledečih množic V(t,T), V(t + T,T), V(t + 2T,T),… se spreminja počasi.

Brez lokalnosti bi bil predpomnilnik neuporaben, ker bi bila verjetnost za to, da je iskana informacija v njem, veliko premajhna. Ker pa lokalnost obstaja, lahko predpomnilnik naredimo tako, da vsebuje delovno množico V(t,T) in se prilagaja njenemu spreminjanju. Ker je delovna množica veliko manjša od velikosti glavnega pomnilnika, zadošča že razmeroma majhen predpomnilnik.

Uspešnost delovanja predpomnilnika merimo z **verjetnostjo zadetka** (hit ratio) in jo označujemo s črko H. Kadar je naslov, do katerega se želi dostopiti v predpomnilniku, imamo **zadetek** – če ga ni, imamo **zgrešitev** (miss). Pogosto se uporablja tudi pojem **verjetnost zgrešitve** (miss rate). Ker imamo pri vsakem dostopu zadetek ali zgrešitev, je verjetnost zgrešitve seveda enaka 1-H. Verjetnost zadetka H lahko izmerimo tako, da štejemo, koliko pomnilniških dostopov se nanaša na naslove, ki so v predpomnilniku.

Predpomnilnik je sestavljen iz dveh delov:

- kontrolni del (vsebuje pomnilniški naslov informacije)
- podatkovni del (vsebuje podatke;operande/ukaze)

Te informacije so v obliki blokov iz glavnega pomnilnika. Velikost bloka danes je od 4B do 256B.

ASOCIATIVNI PREPDPOMNILNIK
Če damo na vhod asociativnega pomnilnika kombinacijo bitov, se ta naenkrat primerja z delom vsebine (lahko tudi s celotno vsebino) vsake od besed, ki sestavljajo pomnilnik. Kadar obstaja enakost, dobimo na izhodu celotno vsebino besede – če enakosti ni, se to sporoči s posebnim signalom.

Pri predpomnilniku predstavljajo vsebino naslovi, ki so shranjeni v kontrolnem delu. Če je kontrolni del narejen kot asociativni pomnilnik, lahko predpomnilnik na osnovi naslova, ki ga da CPE, takoj ugotovi, ali imamo zadetek ali zgrešitev. Pri zadetku se lahko naredi dostop do tiste besede v bloku, ki je določena s spodnjimi b biti. Takemu predpomnilniku pravimo **asociativni predpomnilnik** ali tudi **čisti asociativni predpomnilnik.** Ker so vse besede asociativnega pomnilnika ekvivalentne, ni pri preslikavanju pomnilniških naslovov v predpomnilnik **nobenih omejitev.** Drugače povedano, vsak blok predpomnilnika lahko sprejme katerokoli besedo iz glavnega pomnilnika. To omogoča najboljši možen izkoristek predpomnilnika. Pri dani velikosti je verjetnost zadetka H pri tem predpomnilniku zato večja kot pri vseh drugih.

SET-ASOCIATIVNI PREDPOMNILNIK
Velik predpomnilnik lahko naredimo samo, če v preslikavanje naslovov vpeljemo omejitve. Namesto enega velikega asociativnega pomnilnika lahko uporabimo večje število majhnih, kar je bistveno lažje in ceneje. Tako dobimo tako imenovani **set-asociativni predpomnilnik**. Pri set-asociativnem predpomnilniku je predpomnilnik razdeljen na $S = 2^s$ setov, vsak set pa je majhen asociativni predpomnilnik. Število blokov v setu $E = 2^e$ imenujemo **stopnja asociativnosti** ali kar **asociativnost**. Stopnja asociativnosti ni nič drugega kot velikost asociativnega pomnilnika, v katerem so shranjeni naslovi blokov; pri današnjih predpomnilnikih je tipično med 2 in 8. (število blokov v setu določa število kontrolnih delov v setu).

Pomembna razlika v primerjavi s čistim asociativnim predpomnilnikom je, da imamo sedaj omejitev pri preslikovanju naslovov. Za vsako besedo glavnega pomnilnika je vnaprej določeno, v katerega od setov se lahko preslika.

DIREKTNI PREDPOMNILNIK
Vzemimo, da je velikost predpomnilnika M besed, kjer je $M = S*E*B = 2^{s+e+b}$. Pri set-asociativnem predpomnilniku lahko s spreminjanjem stopnje asociativnosti E vplivamo na njegove lastnosti. Kadar je asociativnost E enaka številu blokov, dobimo čisti asociativni predpomnilnik (S = 1). Drugo skrajnost dobimo, če pri enaki velikosti M vzamemo stopnjo asociativnosti E = 1 (oziroma e = 0). V tem primeru je v vsakem setu samo en blok. Takemu predpomnilniku pravimo **direktni predpomnilnik**. Pri njem je stopnja asociativnosti enaka 1 – to pomeni, da asociativnega pomnilnika sploh nimamo več. Pri direktnem predpomnilniku je omejitev pri preslikavanju najhujša, ker ni več nikakršne svobode. Za vsako besedo glavnega pomnilnika je vnaprej določeno, v katerega od blokov (blok je sedaj enak setu) se lahko preslika.

Predpomnilniško pravilo: **2:1**
Verjetnost zgrešitve (1-H) direktnega predpomnilnika velikosti M je približno enaka verjetnosti zgrešitve set-asociativnega predpomnilnika velikosti M/2 (če sta v vsakem setu samo dva bloka).

POVEČANJE PREDPOMNILNIKA

- povečamo število setov;
- povečamo stopnjo asociativnosti (več blokov) E;
- povečamo velikost bloka (najcenejša varianta).

VRSTE ZGREŠITEV V PREDPOMNILNIKU

- **Obvezne zgrešitve;** predpomnilnik je v začetku prazen. Take vrste zgrešitve so obvezne, ker se jim v nobenem primeru ni mogoče izogniti. Pravimo jim tudi mrzle zgrešitve ali zgrešitve prvega dostopa.
- **Velikostne zgrešitve;** zaradi končne velikosti predpomnilnik ne more vsebovati vseh blokov, ki jih med izvrševanjem potrebuje program. Zato prihaja do zamenjave blokov, ki so kmalu za tem spet potrebni. Z večanjem predpomnilnika se število takih zamenjav zmanjšuje, s tem pa je tudi manjše število zgrešitev.
- **Konfliktne zgrešitve;** obstajajo pri set-asociativnih in direktnih predpomnilnikih. Do take zgrešitve pride, ko je potrebno zamenjati blok, ki se bo kmalu spet uporabil. Do zamenjave je prišlo, ker se oba bloka preslikata v isti set. Konfliktnih težav ne bi bilo, če bi bil predpomnilnik čiste asociativne vrste.

ZAMENJALNA STRATEGIJA

Če je v predpomnilniku zgrešitev je predpomnilnik poln – zamenjava bloka.
Kateri blok zamenjati? To določa zamenjalna strategija (pri direktnem predpomnilniku ni potrebna)

- naključno izbran blok;
- fifo zamenjalna strategija (zamenja se blok, ki je najbolj dolgo v predpomnilniku;
- LRU-least recently used (zamenja se blok, ki je najbolj dolgo neuporaben.

BRANJE/PISANJE

BRANJE: 1 urina perioda pri zadetku

PISANJE: -branje bloka
-spreminjanje vsebine (pisanje)
-shranjevanje bloka

Pri zadetku traja pisanje vsaj 2x več kot branje.
Pri zgrešitvi - prenos bloka iz glavnega pomnilnika v predpomnilnik =>10 urinih period (zgrešitvena kazen).

POMNILNIŠKO PREPLETANJE

Osnovna ideja pomnilniškega prepletanja je v razdelitvi glavnega pomnilnika na m samostojnih delov M_0, M_1,…M_{m-1}. Te dele imenujemo **moduli**, tako prepletanje pa **m-kratno** prepletanje. V osnovni izvedbi se širina podatkovnih poti do modulov ne poveča. Vsak modul je samostojen pomnilnik, ki lahko deluje neodvisno od ostalih. Pri branju poteka delovanje približno takole: vsako urino periodo se pošlje naslov enemu od modulov, moduli shranijo naslov in opravljajo branje, po končanem branju se vsako urino periodo prebrani podatek posreduje v CPE. Pri pisanju je podobno, le da se skupaj z naslovom modulu pošlje še podatek, ki se piše.

Konfliktni dostopi: so istočasni dostopi do istih modulov in se jim ni mogoče izogniti. Pri konfliktnih dostopih morajo prizadeti procesorji čakati in to seveda zmanjšuje učinkovitost prepletanja.

Najpogostejši pravili za pomnilniško prepletanje:

1. **Spodnje prepletanje**. Modul je določen s spodnjimi biti pomnilniškega naslova. Do vsakega modula se prenese naenkrat več sosednjih pomnilniških besed. Omogoča izkoristek zaporedne lokalnosti. Stopnja prepletanja ni nujno potenca števila 2, lahko je tudi praštevilo (računalniki firme Burrougs). Osnovni cilj izbire stopnje prepletanja je zmanjšati število konfliktnih dostopov.
2. **Zgornje prepletanje**. Naslov je določen z zgornjimi biti pomnilniškega naslova. Slabše se izkorišča lokalnost in glede na hitrost ni tako dobro kot spodnje prepletanje. Prednost je da okvara v enem od modulov pri zgornjem prepletanju računalnika ne prizadene bistveno. Občuti se le zmanjšanje velikosti pomnilnika. Pri spodnjem prepletanju je prizadet celotni naslovni prostor. Tudi razširjanje pomnilnika je lažje. Bolj primerna je tudi za več CPE.

NAVIDEZNI POMNILNIK (virtual memory)

Vrste navideznih pomnilnikov

Navidezni pomnilnik je mogoče realizirati na več načinov, ki jih lahko razdelimo v dve osnovni vrsti:

- v tiste z bloki fiksne velikosti, ki jim pravimo **strani**,
- v tiste z bloki spremenljive velikosti, ki jim pravimo **segmenti**.

Na današnjih računalnikih se uporabljajo strani velikosti 4KB, 8KB in 16KB, v bližnji prihodnosti pa pričakujemo tudi 32KB in 64KB strani. Strani so podobne blokom pri predpomnilniku, medtem so so segmenti drugačni.

OSTRANJEVANJE

Ostranjevanje je najstarejša vrsta navideznega pomnilnika. Osnovna ideja ostranjevanja je preprosta. Pomožni pomnilnik je razdeljen na bloke enake velikosti, ki jim pravimo **strani**. Vse strani skupaj sestavljajo navidezni pomnilnik. Na enako velike bloke, ki jim pravimo **okviri strani**, je razdeljen tudi glavni pomnilnik. Vsako stran navideznega pomnilnika je mogoče prenesti v poljubnega od okvirov. Pri dani velikosti pomožnega in glavnega pomnilnika sta največje število strani in število okvirov fiksna in znana vnaprej.

Preslikovalna funkcija je pri ostranjevanju definirana s pomočjo posebne tabele, ki se imenuje **tabela strani**. Vsaki strani navideznega pomnilnika pripada v tabeli strani eno polje. Ker je število strani fiksno, je vnaprej določena tudi največja velikost tabele strani.

Vsak program zaseda določeno število strani. Ker je velikost programa samo izjemoma enaka mnogokratniku velikosti strani, bo v povprečju zadnja stran izkoriščena samo polovično. Z drugimi besedami: v povprečju je pri vsakem programu polovica ene strani neizkoriščena. Temu pravimo notranja fragmentacija, ki narašča z velikostjo strani.

Preslikovanje navideznih naslovov v fizične poteka na nasljednji način: vsak navidezni naslov se pri ostranjevanju obravnava, kot da je sestavljen iz dveh delov; spodnjih p bitov določa **naslov besede znotraj strani**, zgornjih n – p bitov pa **številko strani**. Pri preslikovanju ostane prvi del (naslov znotraj strani) nespremenjen. Drugi del (številka strani) pa se preko tabele strani preslika v številko okvira strani. Vsako polje v tabeli strani pripada točno določeni strani in vsebuje informacijo o njej. Polja v tabeli strani se pogosto označujejo z besedo **deskriptor strani**.

SEGMENTACIJA

Navideznemu pomnilniku, ki ima pomnilniški prostor razdeljen na segmente pravimo segmentacija. Prva razlika v primerjavi s stranmi, ki so vse enako velike, je v različni dolžini segmentov. Druga razlika je v tem, da ima vsebina segmenta za program svoj pomen. Število in velikost segmentov nista fiksna in se med izvajanjem programov lahko spreminjata. V tem se segmentacija bistveno razlikuje od ostranjevanja. Vsak segment je med izvajanjem seveda potrebno prenesti v glavni pomnilnik, podobno kot stran. Ker pa so segmenti različno veliki, v glavnem pomnilniku nimamo ničesar, kar bi ustrezalo okvirom strani. Namesto tega se vsak segment lahko prenese na poljuben naslov v glavnem pomnilniku.

SEGMENTACIJA Z OSTRANJEVANJEM

Z navideznim pomnilnikom, ki uporablja segmentacijo z ostranjevanjem, želimo izkoristiti prednosti segmentacije, vendar brez problemov, ki nastajajo pri čisti segmentaciji. To storimo tako, da razdelimo vsak segment na strani na način, ki je enak tistemu pri ostranjevanju. Vsak program ima poleg tabele segmentov tudi množico tabele strani – po eno za vsak segment.

Pri segmentaciji z ostranjevanjem je glavni pomnilnik, podobno kot pri navadnem ostranjevanju, razdeljen na okvire strani. Vsak n-bitni navidezni naslov, ki ga med izvajanjem tvori CPE, se obravnava, kot da je sestavljen iz treh delov: zgornjih n-s bitov določa **število segmenta**, srednjih s-p ibtov določa **številko strani** in spodnjih p bitov določa **naslov znotraj strani**.

Parametri v deskriptorju strani:

- **FN** (frame No.) vsebuje številko okvirja v katerem je trenutno stran (ki ji pripada deskriptor) v glavnem pomnilniku.
- **V** (valid – bit veljavnosti) če je 1 = vsebina deskriptorja veljavna; če je 0 = ta stran ni definirana (sistem je ne pozna).

- **P** (present – bit prisotnosti) če je 1 = je ta stran v enem od okvirov v glavnem pomnilniku; če je 0 = te strani ni v glavnem pomnilniku (FN nima pomena).
- **Rwx** (read/write X) zaščitni ključ; običajno ga sestavlja več bitov, ki povedo, kakšna vrsta dostopa je dovoljena (branje/pisanje) in kdo ima dovoljen vstop.
- **C** (change – bit spremembe) če je 0 = ob prenosu strani iz navideznega pomnilnika v okvir glavnega pomnilnika. Vsebina strani v navideznem je enaka vsebini okvirja v glavnem pomnilniku. Če je 1 = če pride med izvajanjem programa do pisanja v kateregakoli od naslovov te strani.

Problemi pri preslikavi:

1. velika tabela strani (zasede veliko glavnega pomnilnika),
2. čas za preslikavo navideznega naslova v fizični (čas dostopa do tabele strani),

Če imamo eno tabelo strani (enonivojska preslikava – cela tabela mora biti v glavnem pomnilniku), lahko razdelimo preslikavo na več nivojev (tabelo strani razdelimo na več manjših tabel v več nivojih).

Za vsak dostop do glavnega pomnilnika (do strani v nekem okvirju gl. pomnilnika), ločimo:

- enonivojska preslikava: **2** dostopa do gl. pomnilnika
  1 dostop do tabele
  2 dostopa do okvirja
- trinivojska preslikava: **4** dostopi do gl. pomnilnika
  1,2,3 do tabel, 4. dostop do okvirja (do informacije)