3. Osnovni principi delovanja

1. Von Neummanov računalniški model; kaj je zanj značilno?

Von Neumanov računalniški model ima naslednje zahteve:

- sestavljajo ga trije osnovni deli: CPE, glavni pomnilnik in V/I sistem
- ima program, shranjen v glavnem pomnilniku, ki vodi delovanje stroja
- CPE jemlje (FETCH) ukaz za ukazom iz glavnega pomnilnika (iz naslova kamor kaže PC – program counter) in jih izvršuje (EXECUTE) ter pri tem povečuje PC (za 1 ali pa n, če gre za skok)

CPE – Razdelimo jo lahko na:

- *kontrolno enoto* (prevzema ukaze in operande in aktivira ustrezne operacije),
- *podatkovna enota*
  - *aritmetično-logična enota* (izvaja operacije)
  - *registri* (programsko dostopni [hitrejše delovanje] ali nedostopni [ima vsaka CPE za realizacijo delovanja]).

GLAVNI POMNILNIK – Sestavljen je iz več pomnilniških besed, katerih vsaka ima enoličen naslov.

- V njem so ukazi in operandi, ki jih uporablja CPE.
- Pri tem modelu računalnik ne loči med ukazi in operandi.

V/I SISTEM – je namenjen prenosu informacij v/iz zunanjega sveta.

- Sestavljen je iz V/I naprav.
- Nekatere izmed njih služijo tudi kot pomožni pomnilnik. Fizično je to največji del računalnika.

Obstajati mora mehanizem, ki ukaze naloži v glavni pomnilnik – bootstrap loader. Delovanje računalnika potem popolnoma določajo ukazi, v dveh korakih: fetch in execute. PC – PC +1 oz. PC – PC +n.
Von Neumannov model je ukazno pretokovni (ukazi določajo delovanje rač.), obstaja še podatkovno pretokovni.

2. Kaj je prostorska lokalnost pomnilniških dostopov?

Lokalnost pomnilniških dostopov: programi več kot enkrat uporabijo iste ukaze in operande in bolj pogosto tiste, ki so v pomnilniku blizu trenutno uporabljenim.

- Izkustveno pravilo: tipičen program 90% časa uporablja samo 10% ukazov.
- Povzroča jo način pisanja programov in že samo delovanje Von Neumannovih računalnikov (ukazi si zaporedno sledijo).
- Lokalnost omogoča pomnilniško hierarhijo.

Razdelimo jo lahko na:

prostorsko lokalnost – po pojavu naslova A(i) bodo verjetno naslednji naslovi po lokaciji v pomnilniku blizu naslova A(i).

- Če ni skoka, se ukazi jemljejo zaporedno.
- Strukture, kot so polja, so običajno na zaporednih naslovih in se uporabljajo zaporedoma po svojih indeksih
- Program je razdeljen na podprograme in procedure, ki jim je dodeljen tudi majhen zaporedni prostor, ker so večinoma kratke.

časovna lokalnost – program ob času t tvori naslove, ki jih je tvoril malo pred t in ki jih bo nekoliko po t.

- Zanke (isto zaporedje ukazov oz. naslovov se ponovi velikokrat).
- Začasne spremenljivke, zaradi katerih se naslovi določenih operandov pojavljajo večkrat.

Lokalnost lahko kvantificiramo še z delovno množico (Working Set) – V(t,T). Če se v času T pojavi N ukazov in v intervalu od t-T do t V(t,T), je množica V(t,T) veliko manjša od N. Vsebina zaporedno si sledečih množic V(t,T) se spreminja počasi (prekrivanje).

3. Kaj je časovna lokalnost pomnilniških dostopov? WS (Working set). Uporabnost.

Glej 2.

4. Amdahlov zakon in Case/Ambdahlova pravila (kaj pravijo in čemu so pogoj)?

Pove nam, kako močno lahko pohitrimo računalnik s paralelizmom določenih delov.
Če za faktor N pohitrimo delovanje pri vseh operacijah, razen pri f-tem deležu od vseh, potem je povečanje hitrosti računalnika *(ali drugače rečeno: če je delovanje N-krat hitrejše (1-f)-ti del časa izvajanja in enako hitro f-ti del)* enaka:

$$S(N) = \frac{1}{f + \frac{(1-f)}{N}} = \frac{N}{1 + (N-1) * f}$$

Maksimalna pohitritev, ki sledi iz enačbe: S(N) = 1/f.
Za večino problemov je bolje en hiter procesor, kot več počasnejših, razen izjemoma, ko je stopnja paralelnosti (1-f) visoka.

Skupaj s sodelavcem R.P. Case-om, je Amdahl razvil še dve pravili:

- velikost glavnega pomnilnika v Bajtih mora biti najmanj enaka številu ukazov, ki jih v sekundi izvede CPE
- zmogljivost V/I sistema v bitih na sekundo mora biti najmanj enaka številu ukazov, ki jih v sekundi izvede CPE

Če računalnik ustreza tem pravilom, rečemo da je uravnotežen (bolj ekonomičen).

5. Kaj je DMA krmilnik in kako deluje?

- DMA (Direct Memmory Access) označuje neposredno komunikacijo med glavnim pomnilnikom in V/I napravo (nasprotje od programskega V/I, ko z V/I napravo komunicira le CPE). Računalnik lahko uporablja oba.
- Prenos je realiziran s posebno enoto - DMA krmilnikom, ki je ali samostojna ali del krmilnika naprave.
- Videti je kot določeno število registrov, v katere se lahko piše ali se iz njih bere. Na te registre lahko gledamo podobno kot na besede v glavnem pomnilniku – razlika je v tem, da pisanje ali branje pri njih lahko sproži neko operacijo v napravi ali odraža stanje po prejšnji operaciji. Pri vsakem krmilniku je točno določeno kaj se zgodi, če pišemo v nek register ali če beremo iz njega. (disk: pisanje v register -> bralno-pisalna glava se premakne na določeno sled; branje statusnega registra -> ugotovimo, kdaj je premik končan)
- DMA krmilnik sam komunicira z glavnim pomnilnikom in med prenosom sodelovanje CPE ni potrebno.
- Ker za prenos ni potrebno prevzemati ukazov, je hitrejši, vendar zaradi krmilnika tudi dražji.

6. V/I procesorji.

- Posebna izvedba DMA krmilnikov, ki omogoča prenose velikih količin podatkov ob minimalnem sodelovanju CPE.
- Osnovni cilj je razbremenitev CPE pri V/I prenosih, saj CPE samo sporoča svoje zahteve vhodno/izhodnim procesorjem, ki nato poskrbijo za podrobnosti pri izvrševanju prenosov podatkov
- Rešitev pa je še veliko dražja od DMA krmilnikov, zato se pogosto pojavlja predvsem pri večjih računalnikih.

<u>Dva pristopa:</u>

- Ločena V/I vodila: na vsak V/I procesor je priključeno določeno število V/I naprav.
- Koordinatni sistem vodil: vsak V/I procesor lahko komunicira z vsako V/I napravo. Bolj splošen, a dražji.

7 Kaj je to pomnilniško preslikan V/I?

- Pomnilniško preslikan V/I je del naslovnega prostora glavnega pomnilnika, ki je rezerviran za V/I naprave.
- Registri krmilnikov so v pomnilniškem naslovnem prostoru in so iz CPE videti kot pomnilniške besede.
- Za branje/pisanje lahko uporabljamo vse ukaze za dostop do pomnilnika,
  - tako da posebni V/I ukazi niso potrebni.

<u>Poznamo še:</u>

- ločen vhodno/izhodni prostor:
  - registri krmilnikov so v posebnem naslovnem prostoru, ki je ločen od pomnilniškega.
  - Za dostop do registrov so potrebni posebni V/I ukazi -> med izvajanjem teh ukazov CPE aktivira signal, ki pove, da se naslavlja V/I naslovni prostor

- posredno preko vhodno/izhodnih procesorjev:
  - tudi tu so registri krmilnikov so v posebnem naslovnem prostoru, vendar ta prostor iz CPE ni neposredno dostopen -> do njega imajo dostop V/I procesorji
  - glej še vprašanje 6

4. Predstavitev informacije in aritmetika

8. Števila v fiksni vejici. Glavna prednost komplementov in slabosti ostalih predstavitev števil. Katera predstavitev se danes največ uporablja? Prenos, preliv.

- Vejica je fiksna.
  - Številke na levi strani vejice predstavljajo celo število,
  - na desni pa ulomek (decimalni del).
- Vsaka številka ima svojo težo glede na pozicijo – pozicijska notacija.
- V glavnem se uporablja dvojiška predstavitev (desetiška se uporablja izključno za poslovne obdelave – banke, kjer je to celo uzakonjeno). Desetiška potrebuje za predstavitev števil več bitov in pa logična vezja za desetiško aritmetiko so bolj zapletena.

Predstavitev predznačenih števil:

1.Predznak in velikost:

- najbolj levi bit predstavlja predznak → 0 pozitivno in 1 negativno.
- Pri seštevanju in odštevanju je treba bit za predznak obravnavati drugače kot ostale bite.
- Imamo dve ničli: +0, -0.
- Predstavitev je dobra pri množenju in deljenju (ki sta redki operaciji)
- uporablja se za podajanje *mantise* v predstavitvi s plavajočo vejico (IEEE 754)

2.Predstavitev z odmikom:

- vsa števila so pozitivna (k številu se najprej prišteje konstanta - odmik),
- najbolj levi bit tudi tu določa predznak, vendar ga ni potrebno obravnavati drugače od ostalih → 0 negativno
- če so vsi biti 0 predstavlja to najbolj negativen eksponent (število s takim eksponentom pa je po absolutni vrednosti najbližje ničli)
- Slabost: odmik je treba zmeraj popravljati (pri seštevanju odštevati in pri odštevanju prištevati).
- Prednost: vrednost zaporedja bitov odraža velikost števila
- Uporablja se za podajanje *eksponenta* v predstavitvi IEEE 754 (števila v plavajoči vejici)

3.Eniški komplement:

- Tudi tu najbolj levi bit določa predznak → 0 pozitivno in 1 negativno
- Pozitivna števila so predstavljena enako kot pri načinu ''predznak in velikost''
- predstavitev negativnega števila dobimo tako, da najprej zapišemo pozitivnega, zatem pa vse bite invertiramo.
- Invertiranje bitov je ekvivalentno odštevanju števila od $2^n - 1$.
- Prednost: vse bite obravnavamo enako.
- Pri prenosu iz n-1. mesta moramo k rezultatu prišteti 1 (absolutna vrednost rezultata je večja od $2^n - 1$ in je ne moremo predstaviti z n biti+)
- Imamo dve ničli.

4.Dvojiški komplement:

- kot eniški, le da po invertiranju prištejemo še ena (aritmetično je isto kot odštevanje od 2^n)
- Pri prenosu iz n-1. mesta ni treba prištevati enke.
- Imamo samo eno ničlo.
- Najbolj pogosto uporabljen.

Glavna prednost komplementov:

- pri seštevanju in odštevanju (teh operacij je dosti več kot množenja in deljenja) bit za predznak upoštevamo povsem enako kot ostale bite.
- Pri dvojiškem komplementu pa poleg tega nimamo dveh predstavitev za ničlo.

Do prenosa pride pri prekoračitvi obsega NEpredznačenih števil:

ko je rezultat operacije izven področja $0 \leq x \leq 2^n - 1$

Do preliva pride pri prekoračitvi obsega PREDznačenih števil:

ko je rezultat operacije izven področja $-2^{n-1} \leq x \leq 2^{n-1} - 1$

9. Kje se uporablja predstavitev z odmikom?

Predstavitev z odmikom se uporablja npr. pri standardu IEEE 754 (števila v plavajoči vejici) in sicer za predstavitev eksponenta.

10. Predstavitev numeričnih operandov v plavajoči vejici. m, r, e, podliv, preliv, normalizacija, normalizirana števila.

- Omogoča predstavitev tudi zelo majhnih in zelo velikih števil.
- Število razbijemo na tri dele: mantiso m, eksponent e in bazo r.
- Eksponent in mantisa sta predstavljena s fiksno vejico,
  - zaradi praktičnih razlogov za bazo uporabljamo samo 2 ali 10.
  - Eksponent pove, na katerem mestu je decimalna vejica, ki plava levo (e++) ali desno (e--).
- Ker je baza konstantna, lahko rečemo, da je število v plavajoči vejici predstavljeno s parom (m,e).

Preliv:

- glej 8.

Podliv (v fiksni vejici ga ni):

- do njega pride pri številih, ki so po absolutni vrednosti premajhna, da bi jih bilo mogoče predstaviti v normalizirani obliki
- Zato jih denormaliziramo ali zaokrožimo na 0.

Normalizacija:

- Za računanje je zaželeno, da so števila predstavljena na nek enolično definiran način.
- Ta način naj bo tak, da omogoča največjo možno natančnost računanja.
- Število v plavajoči vejici je normalizirano takrat, ko mantisa na levi strani nima ničel.
  - Drugače povedano: prva številka mantise mora biti različna od nič.
- Števila, ki niso normalizirana, so denormalizirana in jih lahko normaliziramo s pomikanjem mantise v levo ali desno

11 Standard IEEE 754. Enojna in dvojna natančnost. Rezervirane kombinacije bitov. 5 vrst števil.

- baza r = 2
- Eksponent je predstavljen v načinu ***»predstavitev z odmikom«***
- Mantisa je predstavljena v načinu ***»predznak in velikost«***
- Uporabljena je ***implicitna predstavitev normalnega bita***
- več formatov: 32 bitni(enojna natančnost), 64 bitni (dvojna) in 80 bitni (1+15+64; samo interno računanje)
- dovoljeno je računanje z denormaliziranimi števili (ena vrednost eksponenta je rezervirana)

Enojna natančnost:

- predznak S
- 8-bitni eksponent E z odmikom 127
- 23-bitna mantisa m
- vrednost: $(-1)^S(1,m)2^{E-127}$

Dvojna natančnost:

- predznak S
- 11 bitni eksponent E z odmikom 1023
- 52-bitna mantisa m
- vrednost: $(-1)^S(1,m)2^{E-1023}$

Standard IEEE 754 določa predstavitev petih vrst števil:

1. *Normalizirana števila:* E nima vseh bitov 0 ali vseh bitov 1.
2. *Denormalizirana števila:* E ima vse bite 0, m pa je različna od 0.
3. *Ničli:* E in m imata vse bite 0, predznak pa določa +0 in -0.
4. *Neskončnosti:* E ima vse bite 1, m vse bite 0, predznak pa določa +neskončno in -neskončno.
5. *Neveljavna števila:* E ima vse bite 1, m pa nima vseh bitov 0.

12. Zaokroževanje števil v računalniku (IEEE).

- Rezultat računanja (interno se uporablja 80 bitni format) mora biti enak matematično točni vrednosti in zaokrožen na mantiso dolžine 23 oz. 52 bitov.
- Zaokrožimo k najbližjemu še predstavljivemu številu, v primeru enake oddaljenosti pa k sodemu številu.
- Zadošča če mantiso podaljšamo za dodatne 3 bite, na osnovi katerih vrednost mantise povečamo za 1 ali pa ne, bite pa odvržemo:
  - **varovalni** (guard): potrebujemo dodatno mesto, ker sta vsota in razlika lahko za 1 mesto daljša od vhodnih operandov
  - **zaokroževalni** (round): 1 mesto ni dovolj za zaokroževanje, v vseh primerih pa zadoščata 2
  - **lepljivi** (sticky): zaradi pravila zaokroževanja k sodemu številu (logična disjunkcija ven padajočih bitov)

5. Ukazi

13. Kam lahko shranjujemo operande v CPE?

- Ni nujno, da obstaja možnost shranjevanja operandov v CPE.
- Kljub temu imajo vsi današnji računalniki v CPE majhen pomnilnik, v katerega je možno shraniti enega ali več operandov
  - programsko dostopni registri.
  - Omogoča večjo hitrost in krajše ukaze.

Razlikujemo tri možnosti za shranjevanje operandov v CPE:

1.Akumulator: imamo en sam (ali mogoče dva) register, v katerega lahko shranimo en operand.

- Kratki ukazi, preprosto a staro.
- V ukazih ga ni treba eksplicitno navajati (zelo kratki).
- Vmesne rezultate je treba prenašati nazaj v pomnilnik, ker ni prostora.
- Posledica: počasnost, promet med CPE in gl. pom.

2.Sklad: podoben akumulatorskemu → pri obeh direktno dostopen samo en operand (LIFO),

- promet med CPE in GP je manjši, ker je prostora za več operandov.
- Še vedno kratki ukazi in preprostost.
- Njihova prednost je v računanju dolgih matematičnih izrazov v postfiksni obliki, a je takih izrazov malo – več je prireditev.

3.Množica registrov: število je od 8 do 100.

- Zelo razširjena možnost.
- Ločimo dve rešitvi, glede na svobodo pri uporabi registrov:
  - vsi so ekvivalentni – splošnonamenski registri,
  - registri različnih dolžin: ena skupina za računanje z naslovi, druga za ostalo računanje
- Druga rešitev je bolj toga, vendar omogoča prihranke pri gradnji CPE.
- Ta rešitev je pomembna za cevovodne računalnike, kjer registri morajo shranjevati vmesne rezultate.

14. Kaj je značilno za skladovni računalnik?

- Za skladovni računalnik je značilno, da imajo pomnilnik v CPE realiziran v obliki sklada, zato so **brezoperandni.**
- Operacije se izvajajo nad operandi na vrhu sklada, ki jih ni potrebno navajati eksplicitno.
- Pri vsaki operaciji potrebujemo PUSH in POP.

15. Primer skladovnega računalnika.

Primer skladovnega računalnik: Atlas, računalniki podjetja Burroughs, HP 3000. Po letu 1980 jih ni več.

16. Število eksplicitnih operandov.

- Manjše število pomeni krajše ukaze in tudi manjšo moč ukazov.
- Večje število zahteva bolj zapleteno CPE.
- Na število eksplicitnih operandov vpliva tudi tip operacij, ki jih bo računalnik opravljal.
- Računalnike z m eksplicitnimi ukazi imenujemo ***m-operandni*** ali ***m-naslovni***, kar ne pomeni, da imajo vsi ukazi m operandov, najpomembnejši ukazi (ALE) imajo m operandov.

Imamo pet skupin:

1. **3+1 operandni:**
   - prvi trije operandi so za operacijo (npr. OP3 <- OP2 + OP1),
   - 4. pa za naslov naslednjega ukaza (namesto današnjega PC-ja).
   - Takšnih računalnikov ni več, bil pa je recimo EDVAC.
   - Niso imeli pomnilnika za shranjevanje operandov v CPE, uporabljali pa so GP s krožnim dostopom, zato so ga pohitrili z razporejanjem ukazov in operandov (pohitritev do 200x)

2. **3 operandni:**
   - po letu 1980 praktično vsi, ki imajo operande v registrih (load-store računalniki)
   - OP3 <- OP2 + OP1, PC <-PC + 1.
   - Ker se pojavi pomnilnik z naključnim dostopom, potreba po pametnem razvrščanju ukazov in operandov odpade.

3. **2 operandni:**
   - OP2 <- OP1 + OP2 (rezultat operacije shranimo v prostor, na katerem je eden od vhodnih operandov)
   - Za mnogo primerov velja, da rezultat ene operacije takoj uporabimo kot vhodni operand v naslednji, nato pa ga ne potrebujemo več -> ni potrebe po 3 operandih
   - OP1 in OP2 sta lahko v enem izmed registrov ali v pomnilniku (kot pri 3OP).
   - 1 ½ naslovni računalniki – pogost pojav: en operand v registru (krajši naslov) en v pomnilniku.
   - Do 1980 najpogostejša vrsta, ker opustitev enega eksplicitnega operanda pogosto ne vpliva na hitrost računanja.

4. **1 operandni:**
   - CPE ima 1 ali max. 2 akumulatorja za shranjevanje operandov.
   - AC <- AC + OP1; AC=akumulator, OP1=eksplicitni operand
   - PC <- PC + 1

5. **Brezoperandni (skladovni) računalniki:**
   - SKLAD(vrh) <- SKLAD(vrh) + SKLAD(vrh-1).
   - Operandov v ukazu ni treba eksplicitno navajati, ker se operacije izvajajo nad operandi na vrhu sklada -> krajši ukazi kot kjerkoli drugje
   - Potrebujemo pa 2 ukaza z enim eksplicitnim operandom – PUSH in POP

17. Kaj je značilno za 3+1 operandne računalnike?
Glej 16.

18. Lokacija operandov.
- Problem lokacije se pojavi večinoma samo pri ALE ukazih, saj pri drugih ukazih že sama narava ukaza določa izbiro.
- Poleg tega je ta problem omejen skoraj samo na 2 in 3-operandne računalnike, vendar je ravno teh največ.
- Operandi so lahko shranjeni v CPE, v GP ali v V/I krmilnikih.
- V veliki večini primerov se uporabljata samo prvi dve možnosti.

Ločimo torej tri variante:

1. Registrsko – registrski računalniki:
- vsi operandi ALE ukazov so v registrih CPE.
- Imenujemo jih tudi LOAD/STORE računalniki, ker je treba vsak operand prenesti iz in nazaj v pomnilnik.
- Imajo kratke, preproste ukaze.
- Čas izvrševanja ALE ukazov je vedno enak in ga lahko vnaprej določimo (dobro za paralelizem).
- Slabost je v tem, da je za rešitev istega problema potrebnih več ukazov, kot pri tistih računalnikih, ki imajo ALE ukaze lahko tudi v pomnilniku.
- sem spada večina 3-operandnih računalnikov

2. Registrsko – pomnilniški računalniki:
- Ed operand je v registru ali v pomnilniku, drugi pa so vedno registrih.
- V to množico spadajo računalniki, ki imajo ene ali dva akumulatorja in registrsko-registrskih ukazov nimajo – akumulatorski računalniki.
- Pri ALE operacijah lahko uporabljamo pomnilniške operande, ne da bi jih prej prenesli v registre.
- Ukazi so daljši, vendar jih je potrebno manj.
- Časa za izvajanje ne moremo naprej predvideti, saj je odvisen od lokacije operandov.
- Večina 2-operandnih računalnikov spada v to skupino

3. Pomnilniško – pomnilniški računalniki:
- vsak operand je lahko ali v pomnilniku ali v registru.
- So najbolj splošni in omogočajo veliko število rešitev pri istem problemu.
- Lahko delamo tudi brez uporabe registrov, vendar je to slabo.
- Ukazi so zapleteni.
- V to skupino spadajo t.i. CISC računalniki, med katerimi je posebno znana serija VAX

19. Kakšne načine naslavljanja poznamo? Obseg naslovov in uporaba.
Način naslavljanja je način podajanja operandov v pomnilniku. Lahko jih razdelimo v tri osnovne skupine:

1. **Takojšnje naslavljanje:**
   - operand je podan kar z vrednostjo, ki je del ukaza – dodatni dostop do pomnilnika ni potreben.
   - Temu operandu pravim takojšnji ali literal.
   - Običajno se označuje z znakom #.
   - Korist je največja pri aritmetičnih ukazih, primerjavah in v ukazih za prenos podatkov.
   - Uporabno za konstante.
2. **Neposredno naslavljanje:**
   - operand je podan z naslovom ?1 dodaten dostop do pomnilnika.
   - Dolg pomnilnik zahteva tudi dolge ukaze.
   - Če pomnilniški prostor povečamo – drugačna zgradba ukazov - ni kompatibilnosti za nazaj.
   - Naslov je lahko podan tudi samo z delom naslova.
   - Uporablja se za registrsko naslavljanje (operand je podan z naslovom registra v CPE) in za operande na konstantnih naslovih.
3. **Posredno naslavljanje:**
   - naslov pomnilniškega operanda v ukazu je podan preko neke druge vrednosti – ta je lahko:
   - Ta druga vrednost je lahko v:
     1. glavnem pomnilniku (**pomnilniško posredno naslavljanje**)
        - v ukazu je pomnilniški naslov lokacije na kateri je shranjen pomnilniški naslov operanda

- v primerjavi z neposrednim imamo en dodaten dostop do pomnilnika

2. ali v enem od registrov (**registrsko posredno naslavljanje ali relativno naslavljanje**)
   - v ukazu je naslov registra in t.i. ***odmik***
   - Iz vsebine registra in odmika se izračuna pomnilniški naslov operanda
   - Nima dodatnega dostopa do pomnilnika
   - spreminjanje naslova v registru je lažje kot spreminjanje naslova v pomnilniku
   - Mnogi računalniki zato pomnilniškega neposrednega naslavljanja sploh nimajo
   - Praktično vsi pa imajo registrsko posredno naslavljanje, ki ga lahko razdelimo v več skupin:

     3.1. Bazno naslavljanje (A=R2+D):
     - v ukazu sta podana naslov registra (bazni register) in odmik (variabilna dolžina).
     - Dolžina registra R2 je praviloma enaka ali daljša od dolžine pomnilniškega naslova

     3.2. Indeksno naslavljanje
     - Če je dolžina odmika enaka dolžini pomnilniškega naslova (obsega cel pomnilniški prostor), to imenujemo indeksno naslavljanje.
     - Primerno za dostop do elementov polja.

     3.3. Pred-dekrementno naslavljanje
     - je lahko bazno ali indeksno in avtomatsko zmanjšuje indeks,

     3.4. Po-inkrementno
     - pa avtomatsko zvečuje indeks.
     - Po in Pred-Dekrementno lahko skupaj tvorita strukturo sklad.

     3.5. Velikostno indeksno naslavljanje
     - namesto prištevanja (odštevanja) odmika uporablja množenje z odmikom.
     - Za dostop do elementov polja.

20. Kakšna je razlika med baznim in indeksnim naslavljanjem?

- Pri obeh imamo nek bazni register, ki mu prištejemo odmik in tako dobimo naslov operanda.
- Razlika je v tem, da je dolžina odmika pri indeksnem naslavljanju enaka dolžini pomnilniškega naslova (isto bitov)
  - to lahko dosežemo tudi tako, da kot odmik podamo register+odmik (dolžina tako izračunanega naslova je vedno enaka dolžini pomnilniškega naslova pri vsaki dolžini odmika).

21. Kaj je značilno za pozicijsko neodvisno naslavljanje. Kaj so pozicijsko neodvisni programi?

- Omogoča pozicijsko neodvisne programe → to so programi, ki se pravilno izvajajo na poljubnih naslovih.
- To lahko rešimo tudi v okviru navideznega pomnilnika, sicer pa tako, da uporabljamo le ***takojšnje*** (vrednost operanda ne sme biti naslov) in ***relativno naslavljanje*** (posredno registrsko). Glavno je, da program ne vsebuje ukazov, ki vsebujejo absolutne naslove.
- Pri relativnem naslavljanju moramo ob premestitvi programa samo na začetku spremeniti začetno vsebino prvega registra
- Za to je še posebej primerno indeksno naslavljanje, pri katerem je lahko en register namenjen samo pozicijski neodvisnosti.
- Nekateri računalniki omogočajo celo PC-relativno naslavljanje – indeksni register je kar PC.

22. Operacije.

Niso tako pomembne, ker lahko večino problemov rešimo z zelo osnovnimi operacijami. Razdelimo jih v več skupin:

1. Aritmetične in logične: osnovne računske operacije, Boolove operacije, pomiki.
2. Prenosi podatkov: load (pom->reg), store (reg->pom), push (pom ali reg->sklad), pop (sklad->reg ali pom), move (reg->reg ali pom->pom).
3. Kontrolne: pogojni skoki (beq), brezpogojni skoki (j), klici (call) in vrnitve iz procedur.
4. Operacije v plavajoči vejici: manjši računalniki jih nimajo, v CPE je posebna enota -> enota za operacije v plavajoči vejici.
5. Sistemske: spreminjajo parametre delovanja računalnika (stop).
6. Vhodno/izhodne operacije: ukazi za prenos med GP ali CPE in V/I napravo.

23. Vrsta in dolžina operandov.

- Bit
- Znak: 8 bitov (zanki predstavljeni z abecedo ASCII ali EBCDIC).
- Celo število: 16 ali 32 bitov, predznačena števila v fiksni vejici (dvojiški komplement).
- Realno število: 32 bitov za enojno in 64 bitov za dvojno natančnost v plavajoči vejici (IEEE 754).
- Desetiško število: niz 8-bitnih znakov (pakirano (dve BCD številki v vsakem 8-bitnem znaku) ali nepakirano (po en ASCII ali EBCDIC znak v vsakem 8-bitnem znaku)).

Kako je podan tip operanda v ukazu?

- V operacijski kodi (tip operanda določen z ukazom)
- Z metabiti (vsakemu operandu dodamo nekaj bitov ki povejo, za kakšen operand gre)

24. Problem poravnanosti.

- Sestavljeni operandi: iz več pomnilniških besed.
- Dostop do npr. 8 8-bitnih besed naenkrat je narejen kot dostop do recimo 8 paralelno delujočih pomnilnikov.
- Pri n-bitov dolgem naslovu spodnji 3 biti naslova določajo, v katerem od njih je neka pomnilniška beseda, zgornjih n-3 bitov pa naslov besede znotraj vsakega pomnilnika.
- To pomeni, da je istočasen dostop do S besed nekega operanda možen samo, če je naslov A deljiv z S. Če je deljiv pomeni, da je sestavljeni operand poravnan
- Če to ni izpolnjeno, je do takšnega operanda potrebnih več pomnilniških dostopov, kar seveda upočasni delovanje

Načina za shranjevanje sestavljenih pomnilniških operandov:

- **Pravilo debelega konca –** naslov sestavljenega operanda je enak naslovu besede, ki vsebuje najtežji (največ vreden) del operanda, preostale besede vsebujejo po padajoči teži urejene dele operanda
- **Pravilo tankega konca –** naslov sestavljenega operanda je enak naslovu besede, ki vsebuje najlažji (najmanj vreden) del operanda, preostale besede vsebujejo po naraščajoči teži urejene dele operanda

25. Zgradba ukazov (spremenljiva, fiksna, hibridno)?

Zgradba ukazov je podana s formatom ukaza. Ta je sestavljen iz ***operacijske kode***, ***načina naslavljanja*** in ***naslovnega polja*** (operand).

- **Fiksna dolžina:** število eksplicitnih operandov vedno enako, način naslavljanja pa je vsebovan v operacijski kodi. Število formatov je majhno -> lažje realizacija, večja hitrost -> tipično RISC (ARM, HIP – dolžina ukazov je 32 bitov)
- **Spremenljiva dolžina:** poljubno število eksplicitnih operandov, vsak je lahko podan z enim od velikega števila načinov naslavljanja -> veliko formatov (tudi količina dela, ki ga opravi posamezen ukaz, je zelo različna). Z uporabo kratkih formatov za bolj pogoste ukaze se želi doseči, da so programi kratki …(število operandov je vsebovano v operacijski kodi.) (VAX)
- **Hibridni način:** ga dobimo, če na eni strani zmanjšamo spremenljivost dolžine in moči ukazov ter na drugi strani uporabimo več dolžin ukazov (IBM 370)

Dolžina ukaza je odvisna od:

- dolžine pomnilniške besede (dobro je, da je dolžina ukaza mnogokratnik dolžine pomn. besede
- števila eksplicitnih operandov v ukazu (za vsak eksplicitni operand mora biti v ukazu podano, kje je shranjen)
- vrste in števila registrov v CPE (splošno namenski in …)
- dolžine pomnilniškega naslova (pri uporabi neposrednega naslavljanja mora biti v ukazu pomnilniški naslov)

26. Kaj je ortogonalnost ukazov? Kje pride v poštev?.

- Pomeni medsebojno neodvisnost parametrov, ki jih ukaz vsebuje:
  - informacija o operaciji je neodvisna od informacije o operandih
  - informacija o enem operandu je neodvisna od informacije o ostalih operandih
- Za vsak operand lahko uporabimo vse načine naslavljanja in vse dolžine operandov.
  - S tem je lažje programiranje – ker ni izjem, so pravila za podajanje operandov preprosta in enaka pri vseh ukazih.
  - Po drugi strani pa odstopanje od ortogonalnosti poenostavi zgradbo CPE in jo naredi hitrejšo.
- **Pri RISC računalnikih (registrsko - registrski) je ortogonalnost ukazov izgubila svoj pomen:**
  - če so vsi operandi lahko samo v registrih, obstaja za njihovo podajanje še itak samo ena možnost.
  - Ortogonalnost je torej samoumevna in trivialna.
  - Zakomplicira zgradbo CPE in jo upočasni

27. CISC vs. RISC dilema. Prednosti, slabosti. Cevovod.

- Po mnenju nekaterih so boljši računalniki z veliki številom ukazov **CISC računalniki**
- Po mnenju drugih pa je prednost na strani računalnikov z majhnim številom ukazov **RISC računalniki**

1. Razlogi za povečanje števila ukazov-CISC:

1.1. Semantični prepad

- z njim označujemo razliko med računalnikom, kot ga vidi programer v višjem programskem jeziku in med tistim, kar vidi programer v strojnem jeziku
- Včasih so ta prepad označevali kot krivca za prevelike in počasne programe ter za zapletenost prevajalnikov
- Računalniki so zato v razvoju imeli čedalje večje število ukazov, kljub temu da se velik del ukazov rabi zelo redko.

1.2. Mikroprogramiranje

- Omogoča zelo preprosto dodajanje in spreminjanje ukazov
  - Poskus zmanjševanja semantičnega prepada
- Dekodiranje mikroukazov ima za posledico nekoliko počasnejše izvajanje vseh ukazov

1.3. Razmerje med hitrostjo glavnega pomnilnika in CPE

- Jemanje ukazov iz glavnega pomnilnika je bilo počasno.
- En kompleksen ukaz se je zato izvršil hitreje kot ekvivalentno zaporedje preprostih ukazov.

S povečevanjem št. ukazov so proizvajalci torej skušali doseči naslednje cilje:

- Zmanjšati velikost programov
- Poenostaviti gradnjo prevajalnikov
- Povečati hitrost delovanja

2. Razlogi za zmanjševanje števila ukazov-RISC:

- neizpolnjena pričakovanja uvajanja kompleksnih ukazov
- težave pri uporabi kompleksnih ukazov v prevajalnikih (pogosto kompleksni ukazi naredijo več, kot prevajalnik potrebuje, zato jih ne uporabi)
- spremenjeno razmerje med hitrostjo gl. pom in CPE – uvedba predpomnilnikov zmanjsa razliko s faktorja 10 na faktor 2 ➔ ne velja vec, da se en kompleksen ukaz izvede hitreje kot zaporedje preprostejših. Pri CISC so v kontrolnem ROMu zamrznjena zaporedja mikroukazov, pri RISCih pa prevajalnik tvori zaporedja enostavnih ukazov za vsak program posebej => bolj optimizirano
- uvajanje paralelizma v CPE – v podobi cevovoda, ki ga je lažje izvesti pri preprostih ukazih.
- Narediti dober prevajalnik za CISC računalnik je veliko težje kot pri RISC računalnikih.
- Poleg tega je realizacija paralelnega procesiranja (cevovod) veliko lažja pri preprostih ukazih.
- Meritve so pokazale, da so pri približno enaki ceni RISC-i skoraj vedno hitrejši od CISC-ov.
- CPI je pri RISC veliko manjši.

28. Značilnosti RISC računalnikov.

- večina ukazov se izvrši v eni urini periodi CPE
- registrsko-registrski (load/store)
- ukazi realizirani s trdo ožičeno logiko (in ne mikroprogramsko)
- malo ukazov in malo načinov naslavljanja
- vsi ukazi so enako dolgi (fiksna dolžina)
- dobri prevajalniki: upoštevajo zgradbo CPE in po potrebi spreminjajo vrstni red operacij oz. izločajo nepotrebne

29. IBM 370 ukazi. Vrste kanalskih ukazov.

To so računalniki, ki so se pod oznako IBM System/360 začeli proizvajati leta 1964 in so do danes doživeli več izboljšav.

- Način shranjevanja operandov v CPE: množica registrov.
- Število eksplicitnih operandov v ukazu: 2 - operandni.
- Lokacija operandov: registrsko – pomnilniški.
- Načini naslavljanja: takojšnje, bazno, indeksno.
- Operacije in vrsta/dolžina operandov: nadzorni in problemski nivo priviligiranost - multiprogramsko okolje.
- Zgradba ukazov: 14 formatov z 8- ali 16-bitno operacijsko kodo.
- Število ukazov: 1983 se poveča na 208. Ne spada med RISC pa tudi med ekstremne CISC ne.

6. Centralna procesna enota

30. Lastnosti HIP.

- 3-operandni registrsko-registerski (load/store) računalnik s takojšnjim in baznim naslavljanjem
- dolžina pomnilniške besede 8 bitov
- dolžina pomnilniškega naslova 32 bitov
- 32 32-bitnih splošnonamenskih registrov
- vse ALE operacije se izvršijo v eni urini periodi (ukazov za množenje, deljenje in delo v plavajoči vejici ni)
- poravnanost sestavljenih pomnilniških operandov je obvezna
- dolžina operandov: 8, 16 ali 32 bitov - če je manj kot 32, se razširi po pravilu:
  - če je nepredznačen, se zgornji biti postavijo na 0
  - če je predznačen, se zgornji biti postavijo na vrednost najvišjega bita
- pravilo debelega konca: naslov sestavljenega operanda je naslov najtežje besede
- ima predpomnilnik: pri zadetku traja dostop 1 u.p. (pri zgrešitvi pa 11 u.p. – zgrešitvena kazen je 10 u.p.)
- pomnilniško preslikan V/I
- 2 formata pri zgradbi ukazov (če sta bita 31 in 30 enaka 1 je format 2, sicer format 1. Če je bit 30 enak 1, pomeni, da imamo bazno naslavljanje, sicer pa takojšnje. Več je ukazov s formatom 1 (31), s formatom 2 pa 21)
  - Format 1:
    - 31-26 = op. koda,
    - 25-21 = Rs1 (bazni register),
    - 20-16 = Rd (register za preverjanje pogojev),
    - 15-0 = odmik oz. takojšnji operand
  - Format 2:
    - 31-26 = op. koda,
    - 25-21 = Rs1 (bazni register),
    - 20-16 = Rs2 (bazni register),
    - 15-11 = Rd,
    - 10-0 = funkcija (podaljšek op. kode)
- Število ukazov je 52: 8 za prenos podatkov, 34 za ALE, 6 kontrolnih, 4 sistemski

31. Koraki pri izvrševanju ukazov (ukazi trajajo od 3-6 u.p., če je vedno zadetek)

- **IF – prevzem ukaza:** vsebina PC se preko multiplekserja prenese na pomnilniške naslovne signale. Nato se iz pomnilnika preberejo 4 sosednje pomnilniške besede (32 bitov) in se prenesejo v ukazni register IR. Če je v predpomnilniku zadetek traja prenos 1 u.p., drugače pa 11 u.p (1+10). Kontrolna enota čaka dokler je aktiven signal MemWait.
- **ID – dekodiranje ukaza in dostop do registrov:** v registru IR je že ukaz. Vsebina izbranih registrov (glede na format 1 (Rd) oz. format 2(Rs1 in Rs2)) se prebere v vmesna registra A in B. PC se poveča za 4. (trajanje: 1 u.p.)
- **EX – izvrševanje operacije:** računanje naslova ali ALE operacija. Med registri IR, A in B se določi, katera dva bosta šla na vodili S1 in S2. Določi se tudi operacija, ki se bo opravila in rezultat se da na vodilo D in se zapiše v register, ki ga kontrolna enota določi iz operacijske kode
- **MEM – dostop do pomnilnika:** pri LOAD se bere, pri STORE pa piše v pomnilnik. Uporablja se naslov, ki se je v prejšnjem koraku prenesel v register MAR. Še pri TRAP. (trajanje: 1 u.p. + število čakalnih period)
- **WB – shranjevanje rezultata:** vsebina registra C se prenese v register Rd. LOAD, ALE, CALL, MOVER, TRAP.

32. Mikroukazi.

- Za realizacijo kontrolne enote lahko namesto trdo ožičene logike uporabimo mikroprogramiranje.
- Kontrolna enota je v tem primeru narejena kot majhen računalnik, na katerem teče mikroprogram, sestavljen iz mikroukazov.
- Mikroukaz je ekvivalenten enemu stanju diagrama prehajanja stanj:
  - določa, kateri mikroukaz se bo izvršil naslednji,
  - in kateri izhodni signali so aktivni v tem mikroukazu.
- Izvajanje ukazov je počasnejše, vendar pa je spreminjanje in dodajanje novih ukazov zato preprosto.

33. Prekinitve in pasti. Za kaj vse moramo poskrbeti pri njihovi implementaciji?

- Prekinitev označuje dogodek, ki povzroči, da CPE začasno preneha izvajati tekoči program ter začne izvajati prekinitveno servisni program.
- Zahteva za prekinitev pride od zunaj (npr. od neke V/I naprave)
- Prekinitve omogočajo, da drugi deli računalnika pridejo do uslug CPE.
- Posebne vrste prekinitev so pasti, ki jo zahteva CPE sama ali na zahtevo programerja.
- Pasti torej pridejo od znotraj, poleg tega pa so sinhronske na program, medtem ko prekinitve niso.

Dogajanje/delovanje pri prekinitvah (**4 stanja**):

1. normalno izvrševanje ukazov programa
2. shranjevanje stanja CPE ob pojavu prekinitve (če je le-ta omogočena)
3. skok na PSP in izvrševanje njegovih ukazov, tudi ta se lahko prekine (prekinitvam v PSP pravimo **ugnezdene prekinitve**)
4. vrnitev iz prekinitvenega stanja in obnovitev stanja CPE – to mora biti enako tistemu iz točke 2

5 problemov pri implementaciji prekinitev:

1.KDAJ:

- CPE odreagira na prekinitveno zahtevo šele potem, ko dokonča izvajanje trenutnega ukaza (shraniti je potrebno le programsko dostopne registre)
- Takojšnja reakcija je veliko kompleksnejša za realizacijo in se uporablja le, če drugače ne gre (shraniti je potrebno stanja vseh registrov)
- Programer lahko omogoči/onemogoči (maskira) odziv CPE na prekinitvene zahteve
  - Računalniki so narejeni tako, da so po vklopu prekinitve vedno onemogočene.
  - Če pride do nove prekinitve preden PSP shrani registre, bo nova prekinitev povzročila, da se bo izgubil PC, ki se shrani ob prekinitvi – potrebno je onemogočiti nove prekinitve

2.NEVIDNOST prekinitev:

- izvajanje PSP mora biti za uporabnika nevidno (program se po vrnitvi iz PSP nadaljuje, kot da prekinitve ni bilo)
- Vse registre, ki se trenutno uporabljajo je treba shraniti (na sklad ali v registre) in jih zatem obnoviti. Tudi PC.
- Med PSP so prekinitve onemogočene, kasneje se morajo omogočiti.

3.NASLOV PSP: prepoznati je treba napravo, ki je poslala prekinitveno zahtevo (pri pasteh je trivialno, ker je izvor zahteve CPE)
Imamo dve rešitvi:

- **1. Programsko izpraševanje (polling)** – pregledajo se registri vseh naprav (v nekem vrstnem redu glede na prioriteto) v katerih eden od bitov pove ali je zahtevala prekinitev. Ta rešitev je počasna.
- **2. Lahko se v CPE pošlje informacija**, iz katere ta prepozna izvor zahteve. Med najbolj znanimi načini so tako imenovane vektorske prekinitve (glej 35).

4.PRIORITETA: katera od prekinitev se bo izvedla prva (če imamo več prekinitvenih vhodov in na 1 vhod priključenih več naprav)

- Poleg tega je na nekaterih računalnikih dovoljeno, da prekinitev z višjo prioriteto prekine prekinitev z nižjo – ugnezdene prekinitve.
- CPE ima register, s katerim je določen trenutni prioritetni nivo (je v bistvu prioriteta programa, ki se trenutno izvaja)
- CPE se odzove samo na tiste prekinitve, ki imajo višjo prioriteto od lastne (če postavimo prioritetni nivo CPE na najvišjo vrednost, dosežemo onemogočitev vseh prekinitev)
- S pomočjo dodatnega elementa – prekinitvenega krmilnika, lahko to vgradimo tudi v CPE, ki ima samo en bit za omogočanje prekinitev.
- Če je na en prekinitveni vhod priključenih več naprav, lahko problem prioritet rešimo s programskim izpraševanjem ali pa z marjetično verigo (večjo prioriteto ima naprava, ki je bližje CPE):
  - Ideja marjetične verige je, da naprava, ki ni zahtevala prekinitve signal spusti naprej, naprava ki pa jo je zahtevala, pa ga ustavi, da ta ne pride do drugih naprav. – samo pri več priključenih napravah na 1 prek. vhod

5. POTRJEVANJE prekinitev 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 prek. zahtevo) - potrebno, da naprava umakne svojo zahtevo.

- Lahko se naredi programsko (PSP bere ali piše v nek register krmilnika naprave) ali strojno (CPE s posebnim kontrolnim signalom obvesti napravo, da je upoštevana).

34. Kaj mora biti omogočeno za prekinitve?

Za prekinitve v HIP-u služi bit I, ki ga nastavljamo z ukazoma DI (disable) in EI (enable).

35. Kaj so vektorske prekinitve?

- Vektorske prekinitve so skupno ime za vse načine prepoznavanja prekinjajoče naprave, pri kateri naprava pošlje v CPE informacijo o naslovu njenega PSP.
- To pošiljanje se naredi v prekinitvenem prevzemnem ciklu, s katerim CPE pove napravam naj pošljejo informacijo o izvoru prekinitve
- Gledano iz CPE je to posebna vrsta branja. Od drugih branj ga loči kombinacija kontrolnih signalov.
- Pri nekaterih računalnikih je informacija kar naslov, v glavnem pa je samo del naslova PSP.
- Naslovu, kjer je shranjen naslov PSP pravimo prekinitveni vektor ali vektorski naslov, ki se uporablja kot kazalec v tabelo, ki vsebuje naslove PSP.
- Če naprava pošlje samo del naslova, pravimo temu številka prekinitvenega vektorja (ta rešitev je pogostejša, ker poenostavlja krmilnike – namesto 32 bitnega naslova zadošča 8 bitna številka). Ž
- V principu je možno, da ima ista naprava več PSP-jev (naprava sama določa, kateri PSP naj se aktivira)
- Vektorske prekinitve so tipične za večino po letu 1980 razvitih računalnikov.

36. Realizacija prekinitev pri IBM 370.

- CPE ima štiri prekinitvene vhode, preko katerih prihajajo zahteve za naslednje skupine prekinitev: strojna napaka, eksterna prekinitev, V/I prekinitev, Restart.
- Ima tudi dve skupini pasti: Supervisor Call in Program Interruptions. Najvišjo prioriteto ima strojna napaka, najnižjo pa Restart. Za določitev prioritete več naprav na enem vhodu se uporablja marjetična veriga. Prepoznavanje je strojno, vendar ne na vektorski način. Potrjevanje je strojno.
- Prekinitve (razen Restart) lahko maskiramo v registru PSW (če je bit enak 1, je določena prekinitev omogočena). Ob prekinitvi se starti PSW shrani na enega od šestih fiksnih naslovov, iz enega izmed njih pa se vanj shrani novi PSW (odvisno od vrste prekinitve/pasti).
- Ker PSW vsebuje PC, se z zamenjavo naredi skok v PSP, hkrati pa se na fiksne naslove shrani opis prekinitve iz katerega je razviden vzrok prekinitve. Iz PSP se nazaj v program vrnemo z ukazom LPSW (load PSW), ki vanj vrne staro vrednost.

37. Prekinitve pri CDC Cyber 170 (PPU, EXN).

- CPE dobiva prekinitvene zahteve iz V/I procesorjev imenovanih PPU. V vsaki urini periodi lahko zahteva prekinitev samo en procesor, zato logika za določanje prioritete ob istočasnih zahtevah ni potrebna. CPE nima možnosti za onemogočanje prekinitev. V/I procesor prekine CPE tako, da izvrši ukaz EXN.
- 1. CPE dokonča vse ukaze v besedi, na katero kaže PC (v eni besedi so lahko do 4 ukazi!).
- 2. Iz 16 60-bitnih besed dolgega področja v glavnem pomnilniku, ki je določeno z naslovom v registru A V/I procesorja, se prenese v CPE nova vsebina vseh registrov, istočasno pa se trenutna vsebina shrani na to področje.
- Prepoznavanje prekinjajočega V/I procesorja je doseženo avtomatsko (enolični naslovi). Nevidnost je zagotovljena avtomatsko, ker se zamenjajo vsi registri. Potrjevanje prekinitve ni potrebno, ker so prekinitve vedno omogočene in je čas prekinitev vedno enak.

7. Izkoriščanje paralelizma na nivoju ukazov

38. Cevovodno procesiranje.

V zadnjih desetih letih se je najvišja hitrost logičnih elementov povečala za 10X, število elementov na istem čipu pa za 5000X. Če želimo povečati hitrost CPE, moramo uporabiti večje število logičnih elementov in omogočiti paralelnost. To omogoča cevovod.

- Cevovod je realizacija CPE, ki naenkrat izvršuje več ukazov, tako da se posamezni koraki izvrševanja prekrivajo.
- S tem se zmanjša CPI.
- Izvrševanja ukaza se razdeli na manjše podoperacije – stopnje cevovoda.
- Pomik med stopnjami se dela naenkrat, zato morajo biti podoperacije uravnotežene.
- Cevovod je mogoče narediti na način, ki je za programerja neviden.

39. Stopnje klasičnega RISC cevovoda.

Glej 31.

40. Vrste nevarnosti v cevovodu.

Pospešitev idealnega cevovoda bi bila N-kratna, pri čemer je N stopnja cevovoda, a vendar stopnja ne presega 10, saj z večanjem stopnje cevovodne nevarnosti izničijo pridobitve in je lahko cevovod še slabši kot bi bil, če bi bila stopnja manjša.

Poznamo tri vrste nevarnosti:

1. **Strukturne nevarnosti** nastopijo, če več stopenj cevovoda v isti urini periodi potrebuje isto enoto (registri, ALE, pomnilnik).
   - Npr. če je v eni urini periodi možen samo en dostop do predpomnilnika.
   - Odpravimo jih z zmogljivejšim predpomnilnikom in tako, da vgradimo dovolj veliko število enot.
   - Stopnja EX se lahko naredi v obliki cevovoda.
2. **Podatkovne nevarnosti** lahko imenujemo tudi operandne nevarnosti.
   - Do njih pride, če nek ukaz potrebuje operand, ki še ni dostopen.
   - Podatkovne nevarnosti lahko razdelimo v tri vrste: RAW, WAR, WAW.
     - **RAW (read after write)**
       - Ukaz2 bere operand prede ga ukaz1 shrani in zato dobi napačno vrednost
     - **WAR (write after read)**
       - Ukaz2 piše v register še preden ga ukaz1 prebere.
     - **WAW (write after write)**
       - Ukaz 2 piše v register preden vanj piše ukaz1
3. **Kontrolne nevarnosti** povzročajo v večini bistveno večjo izgubo kot strukturne in podatkovne nevarnosti.
   - Do njih pride pri operacijah, ki spremenijo vsebino PC drugače kot običajno – pri kontrolnih ukazih (pogojni skoki, brezpogojni skoki, klici in vrnitve).

Prvi RISC-i so nevarnosti odpravljali programsko z vstavljanjem NOP. Tako odpravljanje bistveno poenostavi delovanje CPE, a povzroči nekompatibilnost programov na različnih računalnikih in upočasni delovanje.

41. Odpravljanje podatkovnih nevarnosti.

1. **Mehurčki:** v HIP je to ukaz NOP - 32 ničel v registrih IR in IR1.
   - Če ukaz bere operand v stopnji ID, prejšnji ukaz pa ga še ni zapisal (WB),
     - se stopnja cevovoda ID (in tudi IF) ***zaklene*** za 3 urine periode,
     - in čaka dokler se pisanje ne izvrši.
   - Tako dolgo vstavlja tudi mehurčke,
     - stopnje EX, MEM in WB pa morajo delovati naprej
2. **Premoščanje:**
   - sposobnost CPE prenosa rezultata iz stopenj EX, MEM in WB v stopnjo ID drugega ukaza (operand dobimo iz vmesnih registrov cevovoda)
   - Ampak če ukaz takoj za **load** potrebuje operand, ki ga load bere iz pomnilnika, se pri HIP vseeno pojavi nevarnost.
     - Premoščanje je mogoče komaj v stopnji MEM, ker operand pride iz PP šele v naslednji u.p.
3. **Cevovodno razvrščanje:** spreminjanje vrstnega reda ukazov.

42. Odpravljanje kontrolnih nevarnosti.

- Kadar stopnja EX povzroči spremembo PC, je vsebina stopenj IF in ID neveljavna.
  - V teh dveh stopnjah sta ukaza, ki sledita skoku in se ne smeta izvršiti (***kontrolna nevarnost***).
  - Potrebno je počakati 2 u.p. → temu pravimo ***skočna zakasnitev***

1. **Predpostavljanje neizpolnjenega pogoja (skočna zakasnitev??):**
   - Cevovod predpostavlja, da skoka ne bo – čakanje se aktivira samo, če je predpostavka napačna
     - Takrat se v stopnji IF in ID naslednjih ukazov vstavi mehurček.

2. Statična predikcija (z zakasnjenimi skoki):

- Ukazi, ki so v programu takoj za skokom, so v t.i. ***skočnih režah***
  - Ukazi v skočnih režah se vedno izvršijo
  - Pri HIP imamo dve skočni reži
- Cevovod deluje tako, da se bosta ukaza, ki sta v skočnih režah (to je v stopnjah IF in ID) kljub skoku vedno izvršila,
  - zato lahko ukaz za skok prestavimo dva ukaza naprej (če ni ukazov, ki bi jih dali v skočne reže, damo NOP)
  - Pri pogojnih skokih, ukaza, ki vpliva na pogoj ne smemo dati v skočno režo
- **Razveljavitvenimi skoki:** imamo posebne ukaze z napovedjo, da skok bo.
  - Cevovod z zakasnjenimi skoki lahko še izboljšamo z uporabo ***razveljavitvenih skokov***
  - Če je napoved pravilna, se ukazi v skočnih režah izvedejo, sicer se razveljavijo.

3. Dinamična predikcija (prediktorska tabela):

- Dinamična predikcija je boljša, ker se napovedovanje izpolnjenosti pogoja med izvajanjem programa prilagaja.
- ***Prediktorska tabela*** je majhen pomnilnik, ki hrani zgodovino skokov.

1-bitna prediktorska tabela:

- Če je bil pogoj za skok izpolnjen (če je skok v resnici bil), se v prediktorski bit zapiše 1, sicer pa 0 (v stopnji EX, ko je znano ali skok bo ali ne – izpolnjenost pogoja je znana)
  - 0 = ni skoka = false
  - 1 = je skok = true
- Pri branju ukaza v stopnji IF se vedno bere tudi bit iz te tabele (kljub temu, da ne vemo, če ukaz pogojni skok):
  - Naslov, iz katerega se bere, je določen s spodnjimi biti naslova, na katerem je ukaz (če ima tabela npr. 4096 vrednosti, se kot kazalec, ki kaže v tabelo, uporabi spodnjih 12 bitov naslova ukaza)
  - Če se iz tabele prebere (IF) 1, je to napoved skoka, sicer napoved neizpolnjenega pogoja.

2-bitna prediktorska tabela:

- Pri vgnezdenih zankah je boljša **dvobitna tabela** (pri zankah se ob zadnjem obhodu vrednost zmanjša s 3 na 2, kar ob naslednji uporabi iste zanke pomeni, da bo napoved ob prvem obhodu pravilna. Ob zadnjem obhodu pa bo še vedno napačna. Vendar to pomeni, da imamo pri 2-bitni tabeli le 1 napačno napoved v primerjavi z dvema pri 1-bitni tabeli)
- Predstavljamo si jo kot število med 0 in 3
- Če je pogoj za skok izpolnjen, se v stopnji EX vrednost poveča za 1 (če je 3 ostane 3)
- Če pogoj ni izpolnjen se zmanjša za 1 (če je 0 ostane 0)
- Pri branju tabele se v stopnji IF njena vsebina uporabi takole:
  - Če je 2 ali 3, se napove izpolnjen pogoj
  - Sicer pa neizpolnjen

4. Dinamična predikcija (korelacijski prediktorji):

- korelacijski prediktorji (m,n) je naprava, ki uporablja obnašanje prejšnjih m skokov, da izbere eno od n-bitnih prediktorskih tabel (upoštevajo tudi preteklost drugih pogojnih skokov).
- Za vsak skok imamo 2^m prediktorskih tabel (m je število skokov), ki hranijo zgodovino m drugih skokov. (navadno 2-bitno tabelo lahko označimo kot korelacijski prediktor (0,2))
- Na podlagi te »globalne zgodovine pogojnih skokov« se potem izbere ena izmed teh prediktorskih tabel,
  - s spodnjimi naslovnimi biti skočnega ukaza pa ena n-bitna vrednost.
- Običajna n-bitna prediktorska tabela je korelacijski prediktor (0,n).

5. Dinamična predikcija (skočni predpomnilnik):

- Tudi pri pravilni napovedi izpolnjenega skočnega pogoja se, ne glede na vrsto prediktorja, vedno izgubi 1 u.p.
- V stopnji IF namreč ne moremo izračunati skočnega naslova, ker še ne vemo, za kateri ukaz sploh gre
- uporaba predikcije je torej koristna samo, če je znan tudi skočni naslov, **ki pa ga hrani skočni predpomnilnik.**
- V kontrolnem delu so shranjeni naslovi zadnjih skokov, pri katerih je bil pogoj za skok izpolnjen (skočni naslovi se v predpomnilnik shranijo v stopnji EX, če je bil pogoj izpolnjen – če pogoj ni bil izpolnjen, shranjevanje ni potrebno, saj je pri njih skočni naslov že znan -> to je naslednji višji zaporedni naslov)
  - v podatkovnem pa napovedani skočni naslov in predikcijski biti.
- V stopnji IF se istočasno z branjem ukaza naredi tudi dostop do skočnega predpomnilnika. Če imamo v predpomnilniku zadetek, in če tako določajo prediktorski biti, se skočni naslov zapiše v PC. To pomeni, da pri pravilni napovedi ni potrebno čakati 1 u.p. Napoved pa ni vedno pravilna, in tudi skočni naslov včasih ni pravilen. Ko pride ukaz v EX se preveri izpolnjenost pogoja za skok in enakost napovedanega skočnega ukaza z izračunanim.
  - pogoj izpolnjen, izračunani naslov enak napovedanemu – vse v redu
  - pogoj izpolnjen, naslov ni enak izračunanemu – nop v IF in ID, nov naslov v PC, zapise se tudi v skočni predpomnilnik
  - pogoj neizpolnjen – ukaz se izbrise iz skočnega predpomnilnika

6. **Dinamična predikcija (turnirski prediktor):**
   - Paralelno delujeta lokalni in korelacijskemu prediktorju podoben globalni prediktor
   - Imamo še selektorja, ki odloči, ali se bo uporabila napoved iz globalnega ali lokalnega prediktorja

43. Operacije, ki trajajo več urinih period.

- Pri operacijah kot sta množenje in deljenje ni smiselno zahtevati, da se izvedejo v eni u.p., ker bi za to morali upočasniti uro ali pa uporabiti ogromno količino logičnih vezij.
- Oboje je slabo, zato se procesorji gradijo tako, da se večina operacij izvrši v 1 u.p., ne pa vse.
- Pri operacijah, ki trajajo več urinih period, bi se moral cevovod v stopnji EX ustaviti in čakati dokler operacija ni izvršena, kar izniči prednosti cevovoda.
- Ta problem rešujemo s funkcijskimi enotami, ki znajo izvršiti določeno množico operacij (ločimo 4 vrste):
  - Celoštevilska funkcijska enota: ALE ukazi celih števil, skoki, load/store.
  - Enota za seštevanje, odštevanje in pretvorbe v plavajoči vejici.
  - Enota za množenje (vso množenje – s celimi števili in števili v plavajoči vejici).
  - Enota za deljenje (vso deljenje).

Novi problemi:

- kadar se v neki urini periodi konča več ukazov, je število pisanj v register vecje od 1 ➔zato moramo povečati zmogljivosti stopnje WB, če se želimo izogniti strukturnim nevarnostim.
- WAW nevarnosti, ker lahko v WB pridejo ukazi v drugačnem vrstnem redu od prvotnega (kajti vsaka od enot deluje neodvisno ena od druge in vsaka potrebuje različno število stopenj za operacije – posledično lahko nek 'krajši ukaz' pride v stopnjo WB pred nekim 'daljšim' ukazom, kljub temu, da je bil 'daljši' pred 'krajšim'). (WAR ne more biti, ker se bere se vedno le v ID)
- ker traja računanje v funkcijskih enotah več urinih period, pri RAW nevarnostih premoščanje do stopnje MEM ni možno in moramo čakati.

44. Kako doseči CPI < 1?

Cevovod omogoči da se CPI približa 1. Če želimo CPI zmanjšati pod 1, moramo v eni urini periodi prevzeti več kot 1 ukaz in jih tudi več izvršiti – večizstavitveni procesorji. Te procesorje lahko razdelimo v superskalarne in VLIW procesorje. Razlika je v načinu prevzemanja in izstavljanja ukazov, ki ju je mogoče izvajati statično ali dinamično.

- **Statično razvrščanje:** procesor prevzema ukaze v natanko takem vrstnem redu, kot so v programu,
  - spreminjanje vrstnega reda pa lahko opravlja prevajalnik.
  - Slabost: med izvajanjem programa se pokaže, da bi v istočasno izvrševanje lahko prišli nekateri ukazi, a prevajalnik tega ne more ugotoviti.

- **Dinamično razvrščanje:** procesor prevzema ukaze po vrstnem redu, ki je drugačen od tistega v programu:
  - ko je ena od enot procesorja prosta išče ukaze, ki jih je mogoče poslati vanjo.
  - Prednost: pri zgrešitvah v predpomnilniku lahko med čakanjem izvrši nek drug ukaz.
  - Pri iskanju ukazov moramo uporabljati dinamično predikcijo skokov, ki pa ni vedno pravilna, zato prevzet ukaz morda ne bo potreben: dinamično razvrščanje+ predikcija skokov = špekulativno izvrševanje.

- **Statično izstavljanje:** je način pri katerem je vrstni red izstavljanja ukazov v stopnjo EX določen že pri prevzemu ukaza.

- **Dinamično izstavljanje:** vrstni red izstavljanja določa logika v procesorju.
  - Pregleduje ukaze, ki se izvršujejo in išče take, ki niso odvisni od trenutno izvršujočih.
  - Če takega najde, ga izstavi v izvrševanje.
  - Pri špekulativnem izvrševanju se rezultati ukazov ne smejo shraniti v programsko dostopne registre ali pomnilnik tako dolgo, dokler ni gotovo, da je bila predikcija skokov pravilna.

- **Superskalarni procesorji:** predstavljamo si jih kot običajni cevovod, ki je sposoben prevzemati, dekodirati, izvrševati in shranjevati rezultate več ukazov istočasno.
  - To pomeni, da potrebujemo dodatne vmesne izravnalnike in dodatne stopnje za izstavljanje ukazov in umikanje rezultatov.
  - Pri superskalarnih procesorjih je izstavljanje ukazov vedno dinamično.
  - Po letu 1995 je začelo prevladovati tudi dinamično razvrščanje.

- **VLIW:** pri VLIW imamo dolge ukaze, ki so razdeljeni na polja, v katerih so ukazi za posamezne funkcijske enote.
  - Če vseh polj ne moremo zapolniti s koristnimi ukazi, vstavimo NOP.
  - Pri VLIW procesorjih ne prevzemamo enega ampak več ukazov naenkrat.
  - Vsak ukaz ima svojo cevovodno enoto.
  - Uporablja statično razvrščanje in izstavljanje.
  - Težko je najti probleme, ki imajo dovolj veliko količino paralelnosti.

8. Glavni pomnilnik in predpomnilniki

45. Kako vemo ali je v predpomnilniku prišlo do zadetka ali zgrešitve?

- Pri vsakem pomnilniškem dostopu imamo n-bitni naslov (od tega spodnjih b bitov za naslov besede v bloku)
- Zgornjih n-b bitov naslova se v predpomnilniku primerja z naslovi v kontrolnem delu vseh blokov.

1. **ZADETEK**
   - Če se ugotovi enakost pri nekem bloku, in če je veljavni bit V = 1, imamo zadetek.
2. **ZGREŠITEV**
   - Če enakosti naslovov ni ali če je bit V=0 imamo zgrešitev
   - potreben je dostop do glavnega pomnilnika

46. Katere vrste zgrešitev poznamo? Kako zmanjšamo število obveznih zgrešitev in kaj se s tem poveča?

1. **Obvezne zgrešitve**:
   - So ob prvem dostopu do vsebine nekega bloka (neke besede), ker tega seveda ni v predpomnilniku.
   - Če bi imeli večje bloke, bi bilo teh zgrešitev manj (ker se prenese več besed), vendar s tem lahko povečamo zgrešitveno kazen – ni vredno.
2. **Velikostne zgrešitve**:
   - Zaradi končne velikosti predpomnilnik ne more vsebovati vseh blokov, ki jih med izvrševanjem potrebuje program.
   - Rešitev je povečanje predpomnilnika, s tem je tudi manj velikostnih zgrešitev
3. **Konfliktne zgrešitve**:
   - Pojavljajo se samo pri set asociativnih in direktnih predpomnilnikih
   - Do konfliktne zgrešitve pride ko je v predpomnilniku sicer še dovolj prostora, a ima blok določeno preslikavo na že zaseden set.
   - Konfliktnih zgrešitev ne bi bilo, če bi bil predpomnilnik čiste asociativne vrste
   - Zmanjšamo jih lahko s povečanjem stopnje asociativnosti
     - To ni enostavno in lahko poveča čas dostopa do pomnilnika

47. Kako lahko zmanjšamo število obveznih zgrešitev?

- Število obveznih zgrešitev zmanjšamo z večjimi bloki, vendar s tem lahko povečamo zgrešitveno kazen
- Pri večjih blokih je obveznih zgrešitev manj, ker se prenese več besed.

48. Kako se ugotovi zadetek v predpomnilniku (rabimo primerjalnike)? Koliko primerjalnikov potrebujemo v predpomnilniku?

- Problem primerjave zgornjih n-b bitov naslova z naslovi vseh blokov rešujemo z uporabo velikega števila primerjalnikov
- Potrebujemo toliko primerjalnikov, kolikor je blokov v setu (pri set asociativnem predpomnilniku).
- Če imamo čisti asociativni predpomnilnik potrebujemo toliko primerjalnikov, kolikor je blokov v predpomnilniku

49. Kaj je zgrešitvena kazen?

Zgrešitvena kazen predstavlja dodatno število urinih period, ki se pri zgrešitvi v predpomnilniku prištejejo času dostopa. Odvisna od zgradbe predpomnilnika (predvsem od velikosti bloka), širine podatkovnih poti in morebitnega drugega nivoja L2.

50. Kako je zgrajen predpomnilnik (kontrolni, podatkovni del)?

Zgradba predpomnilnika:

Predpomnilnik je sestavljen iz dveh delov: kontrolnega dela in pomnilniškega dela.

1. **Pomnilniški del** je razdeljen v enote enake velikosti, ki jim pravimo bloki.
   - Velikost bloka B = 2^b sosednjih pomnilniških besed.
2. **Kontrolni del** vsebuje informacijo, ki enolično opisuje vsak blok.
   - Naslov v kontrolni informaciji pove, kateri del glavnega pomnilnika je trenutno v bloku
   - Običajno pa so v njej še nekateri dodatni kontrolni biti, kot sta veljavni bit in umazani bit.
     - **Veljavni bit** – V → če je V=0, je vsebina bloka neveljavna
     - **Umazani bit** – Na 1 se postavi, če pride do pisanja v blok

51. (Pred)Pomnilniško pravilo 2:1

Verjetnost zgrešitve direktnega predpomnilnika velikosti M je približno enaka verjetnosti zgrešitve set asociativnega predpomnilnika s stopnjo asociativnosti 2 in velikostjo M/2.

52. Kako deluje predpomnilnik(+kontrolni del)? Razlogi zanj.

- Predpomnilnik izrablja lokalnost pomnilniških dostopov: je majhen in hiter. Navzven deluje kot da je velik.
- PP je podmnožica glavnega pomnilnika in ga priključimo med CPE in glavni pomnilnik
- Zgrajen je iz SRAMov in je običajno na istem vezju kot CPE.
- Služi za premoščanje vrzeli med hitrostjo DRAMov in hitrostjo CPE.

To vrzel lahko še zmanjšamo tako da:

1. Predpomnilnik razdelimo na dva dela (heterogen - Harvardski predpomnilnik) → en za ukaze in en za operande, tako da se lahko v isti urini periodi dostopa do obeh (zaradi cevovoda)
2. Uporabimo tudi širše podatkovne poti iz CPE do predpomnilnika in iz predpomnilnika do glavnega pomnilnika.
3. Dodamo lahko tudi drugi, tretji, … nivo predpomnilnika.

53. Kaj je asociativnost?

- Število blokov v setu E = 2^e imenujemo *stopnja asociativnosti ali kar asociativnost* in ni nič drugega, kot velikost asociativnega pomnilnika (v setu) → ta velikost pa je enaka številu primerjalnikov, ki jih tak predpomnilnik potrebuje
- Pri današnjih PP je asociativnost tipično med 1 in 16

54. Predpomnilniki glede na omejitve pri preslikavi=Kakšne vrste predpomnilnikov poznamo? (+stopnja asociativnosti za vsako)?

Problem je v primerjavi naslova, ki ga da CPE z zgornjimi n-b biti v kontrolnem delu bloka. Primerjavo je treba opraviti z vsemi bloki v eni urini periodi. Zato uvedemo določene omejitve pri preslikavi.

1. Asociativni predpomnilnik (čisti asociativni p.p.):

- Dostop do besede poteka preko vsebine besede (oz.bolj pogosto preko dela vsebine besede)
  - Asociativnim pomnilniki se zato imenujejo tudi vsebinsko-naslovljivi ali paralelno-iskalni
  - Pri dostopu podamo del vsebine besede. Pomnilnik nato poišče besedo, v kateri se del besede ujema s podanimi biti in to je naslovljena beseda.
- kontrolni del je narejen kot asociativni pomnilnik (dostop do besede poteka preko njene vsebine)
- zato lahko predpomnilnik na osnovi naslova, ki ga CPE zahteva, takoj ugotovi, ali imamo zadetek ali zgrešitev.
- Pri zadetku se naredi dostop do tiste besede v bloku, ki je določena s spodnjimi *b* biti.
- Vsak blok predpomnilnika lahko sprejme katerokoli besedo iz glavnega pomnilnika

Kljub dobrim lastnostim, se asociativni predpomnilniki ne uporabljajo veliko, ker potrebujemo veliko število n-b bitnih primerjalnikov, ki zahtevajo zelo veliko št. logičnih elementov

2. Set asociativni predpomnilnik:

- Dobimo ga, če namesto enega velikega asociativnega pomnilnika uporabimo večje število majhnih, kar je lažje in ceneje.
- Set-asociativni predpomnilnik je 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* in ni nič drugega, kot velikost asociativnega pomnilnika v setu
- Velikost predpomnilnika je enaka M_b=S*E=2^{s+e} blokov oziroma M=S*E*B=$^{2s+e+b}$ pomnilniških besed.
- Pri set asociativnem predpomnilniku imamo omejitev pri preslikovanju naslovov.
  - Za vsako besedo glavnega pomnilnika je vnaprej določeno, v katerega od setov se lahko preslika: to določajo naslovni biti a_b, a_{b+1}, …, a_{b+s-1}.
  - Vsak blok predpomnilnika ne more sprejeti poljubne besede iz glavnega pomnilnika.
  - Znotraj vsakega seta pa so bloki še vedno ekvivalentni.

3. Direktni predpomnilnik:

- Če stopnjo asociativnosti zmanjšamo na ena (E = 1 oz. v vsakem setu je en blok), potem dobimo direktni predpomnilnik, torej asociativnega pomnilnika sploh nimamo več
- Pri direktnem predpomnilniku je omejitev pri preslikovanju 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

4. Pseudo asociativni predpomnilnik:

- Poskuša združiti prednosti direktnega in set-asociativnega pomnilnika.
- Če imamo zadetek deluje identično kot direktni predpomnilnik.
  - Pri zgrešitvi pa je razlika → namesto, da se v predpomnilnik prenese nov blok, se pri pseudo asociativnem naredi še en poskus dostopa, tokrat do pseudo bloka.
- Pseudo asociativni predpomnilnik je podoben set asociativnemu z dvema blokoma v setu. Razlika je v tem, da je pri zadetku čas dostopa do pseudo bloka daljši od časa dostopa do prvega bloka. Poleg zgrešitvene kazni imamo torej en hiter in en počasen zadetek. Pojavi se nevarnost, da bo velik del zadetkov v počasnih pseudo blokih. Rešitev je, da se ob zadetku v pseudo bloku oba bloka preprosto zamenjata: pseudo blok postane prvi in obratno.

55. Pomen asociativnega pomnilnika (zakaj mu pravimo tudi vsebinsko naslovljiv, paralelno iskalni)?

- Asociativni pomnilnik se uporablja, ko želimo ugotoviti ali in kje v pomnilniku se nahaja določena vsebina.
- To se zgodi zelo hitro in je kot nalašč za predpomnilnike, kjer je potrebno hitro ugotoviti ali se dani naslov nahaja v predpomnilniku ali ne.

56. Načini zaščite (glavnega) pomnilnika (par registrov z zgornjo in spodnjo mejo, zaščitni ključ pri vsakem bloku v okviru navideznega pomnilnika).

Potrebujemo jih zato, da en program ne posega (namerno ali nenamerno) v pomnilniški prostor drugega → večopravilni in večuporabniški OS.

Načini zaščite:

a) **Par registrov**
   - …,ki vsebujeta zgornjo in spodnjo mejo (ali pa začetni naslov in dolžino programa).
   - Vsak pomnilniški naslov A se pred dostopom do pomnilnika preveri.
     Naslov je veljaven, če je izpolnjen pogoj: **spodnja meja ≤ A ≤ zgornja meja**.
   - Slabost te vrste zaščite je v tem, da mora program zasedati zvezen prostor v pomnilniku.
   - Poleg tega so vse besede zaščitene na enak način. Npr. ni možno, da bi bilo do nekaterih delov dovoljeno branje, do drugih pa samo pisanje.

b) **Metabiti**
   - vsaka beseda ima metabite, ki povejo kateremu programu pripada ta beseda.
   - Slabost je potratnost pomnilnika in časa (pri spreminjanju bitov pri spremembi programa)

c) **Rešitve danes**
   - Glavni pomnilnik je razdeljen na dele določene velikosti, ki so vsak zase zaščiteni.
   - Tem delom pravimo bloki ali strani (ki pa niso strani v smislu navideznega pomn.!!).
   - Vsakemu programu je dodeljen pomnilniški prostor, ki je enak celemu številu strani.
   - Vsaka stran ima svoje zaščitne metabite – **zaščitni ključ**, ki se shrani v poseben pomnilnik, do katerega ne moremo dostopati s pomnilniškimi ukazi.

57. Kako je zaščiten (glavni) pomnilnik pri seriji IBM 370 (zaščitni ključi, PSW register)?

- Glavni pomnilnik je razdeljen na bloke velikosti 4096 Bajtov;
- Vsak blok ima svoj 7-bitni zaščitni ključ.
- Biti 0-3 ključa določajo, kateremu programu pripada blok,
- Bit 4 (5. bit po vrsti) pa določa kako zaščita deluje.
  - Če je enak 0, deluje zaščita pri pisanju v pomnilnik (branje je dovoljeno)
  - Če je enak 1, deluje zaščita pri branju in pisanju
- Če do bloka dostopa CPE, se biti 0-3 ključa primerjajo z biti 8-11 PSW registra v CPE in če imamo enakost, je dostop dovoljen. Če enakosti ni se sproži ***past zaščite***
- Podobno je pri dostopih V/I naprav, le da imamo tam namesto registra PSW, register ORB.

58. Strategije zamenjave blokov pri zgrešitvah (naključna, LRU strategija) = Kako je realizirana zamenjava bloka v predpomnilniku

a) **Direktni predpomnilnik**
   - Zamenja se blok, v katerega se preslika beseda, pri kateri je prišlo do zgrešitve

b) **Čisti asociativni in Set asociativni**

   Če je blok ki ga mečemo iz predpomnilnika umazan, ga moramo najprej prenesti nazaj v glavni pomnilnik. Za določanje blokov za prenos se uporabljata dve strategiji:
   - naključna strategija: zamenja se naključni blok
   - LRU: zamenja se blok, do katerega najbolj dolgo ni bil narejen dostop.
     - Ta strategija je boljša, ker zmanjšuje nevarnost za zamenjavo bloka, ki se je uporabil pred kratkim (izkoriščanje časovne lokalnosti – bolj verjetno je, da se dalj časa neuporabljeni blok ne bo več uporabil)
     - pri LRU za asociativnost ≥ 4 postane logika težka za realizacijo (Če je pomnilnik velik in je tudi asociativnost velika, med strategijama sploh ni več razlike.)

59. Načini dostopa do glavnega pomnilnika (naključni page mode, zaporedni, krožni, direktni). Kaj je to page mode način dostopa?

Današnje pomnilnike delimo v dve skupini:

- Navadni pomnilniki kjer je beseda podana z naslovom (ista beseda ima vedno isti naslov)
- Asociativni pomnilniki, kjer besede nimajo naslova. Dostop do njih poteka preko vsebine besede

Navadne pomnilnike delimo še glede na vrstni red naslovov do katerih dostopa

1. **Naključni dostop**:
   - Čas dostopa do poljubne besede je neodvisen od naslova pred tem naslovljenih besed (je konstanten);
   - To so DRAM pomnilniki, ki imajo tudi page mode način dostopa, ki pa ni naključni; DRAM si lahko predstavljamo kot 2-dimenzionalno polje. Če želimo brati ali pisati v neko celico, je potrebno podati naslov vrstice in stolpca (to pa podajamo ločeno – pri naključnem dostopu se najprej poda naslov vrstice, nato pa še stolpca. Pri page-mode dostopu pa se v register vrstice shranijo vsi biti naslovljene vrstice – dostop do bitov vrstice se sedaj lahko naredi bistveno hitreje. Ko je vrstica shranjena v registru vrstice, lahko samo spreminjamo naslov stolpca in tako dostopamo do bitov vrstice)
   - Ta način pokaže svoje prednosti pri uporabi predpomnilnika, saj se vanj vedno prenese cela vrstica (ena ali več).
2. **Zaporedni dostop**
   - čast za dostop je odvisen od naslova besede, do katere je bil narejen dostop tik pred tem
   - Izvršiti moramo dostop do vseh vmesnih besed.
   - Magnetni trak, pomikalni register
3. **Krožni dostop**
   - Posebna vrsta zaporednega dostopa
   - Predstavljamo si ga kot magnetni trak zlepljen v zanko
   - Tipični predstavniki: magnetni disk s fiksnimi glavami, magnetni boben, zakasnilne linije, magnetni mehurčki
4. **Direktni dostop**
   - Kombinacija zaporednega in krožnega načina dostopa
   - Srečamo pri magnetnih in optičnih diskih s premičnimi glavami
   - Bralno pisalna glava se najprej premakne nad ustrezno sled (zaporedni dostop), nato pa imamo krožni način dostopa

60. Problemi pri pisanju v predpomnilnik? Pisalne strategije, pisanje skozi/nazaj, pisalni izravnalnik. Problem skladnosti (prenosi skozi PP, razveljavitev, dodatna logika). Kako ravnamo, ko pri pisanju pride do zgrešitve v predpomnilniku?

- Ker je veliko več dostopov do pomnilnika bralnih, je dobro da ga optimiziramo za branje. A vendar moramo zaradi Amdahlovega zakona realizirati tudi hitro pisanje.
- Ločimo dve osnovni vrsti pisalnih strategij:
  1. **Pisanje skozi**:
     - Vedno se piše v predpomnilnik in v glavni pomnilnik.
     - Ker je pri pisanju skozi pisanje v glavni pomnilnik bistveno počasnejše od pisanja v predpomnilnik, mora CPE čakati
     - To čakanje lahko v veliki meri odpravimo, če v CPE vgradimo ***pisalni izravnalnik***
     - Namesto v GP piše CPE v pisalni izravnalnik, ta pa poskrbi, da se njegova vsebina prenese naprej v GP, medtem ko CPE deluje naprej

     Prednosti pisanja skozi:
     - Enostavnejše za realizacijo
     - Vsebina bloka v predpomnilniku je vedno enaka tisti v glavnem pomnilniku (vsebini sta skladni)
  2. **Pisanje nazaj**:
     - Piše se samo v predpomnilnik.
     - Ko pride do zgrešitve in se blok, v katerega smo prej pisali, zamenja, se mora ta blok zapisati nazaj v GP
     - Ali ga je treba prenesti ali ne označimo z umazanim bitom v vsakem bloku (pisanje v blok → U=1=umazan).

     Prednosti pisanja nazaj:
     - Pisanje poteka s hitrostjo, ki jo ima predpomnilnik
     - Pri več pisanjih v isti blok, je potrebno samo eno pisanje v GP (ob zamenjavi bloka)
     - Ob prenosu bloka nazaj v GP se lahko izkoristi maksimalna hitrost prenosa, ker se prenese ves blok

61. Pisalni izravnalnik

- Pisalni izravnalnik se uporablja pri pisanju skozi.
- Problem je, da je pisanje v predpomnilnik mnogo hitrejše od pisanja v glavni pomnilnik, zato se podatek shrani najprej v pisalni izravnalnik (ta je v bistvu majhen vmesni pomnilnik), kasneje pa CPE poskrbi, da se podatki iz pisalnega izravnalnika prenesejo v GP.

62. Kaj so to pisalne zgrešitve in kje nastopajo?

- Pri bralnih zgrešitvah se blok vedno prenese v PP
- Do pisalnih zgrešitev pa pride, ko želimo na nek naslov pisati, tega bloka pa ni v predpomnilniku.

Kaj narediti pri pisalni zgrešitvi:

1. Pisalna zamenjava:

- Pisalne zgrešitve se obravnavajo enako kot bralne
- V predpomnilnik se prenese nov blok, ki mu sledi dostop do predpomnilnika
- Uporablja se pri predpomnilnikih, ki uporabljajo pisanje skozi ali pisanje nazaj

2. Pisanje naokrog:

- Pisalna zgrešitev ne povzroči zamenjave bloka v predpomnilniku
- Blok se spremeni samo v GP

63. Kaj je to neblokirajoči predpomnilnik?

- Neblokirajoči predpomnilnik je tak PP, kjer se ob zgrešitvi do njega lahko normalno dostopa
- CPE se ob zgrešitvi v ukaznem ali operandnem predpomnilniku ne ustavi, temveč vnaprej ali pa špekulativno izvršuje druge ukaze (medtem, ko se vrši zamenjava);
  - če se rezultati teh ukazov lahko koristno porabijo, izgube zaradi zamenjave bloka sploh ni več.

64. Načini za zmanjševanje zgrešitvene kazni pri predpomnilniku (neblokirajoči predpomnilnik, bralni izravnalnik, vnaprejšnji prevzem bloka).

Zmanjšamo jo lahko s pametnim vrstnim redom prenašanja besed v bloku in z več nivoji predpomnilnika. Imamo pa še dve zelo pomembni rešitvi:

1. **Vnaprejšnji prevzem bloka:**
   - Zaradi zaporedne lokalnosti obstaja verjetnost, da bo poleg bloka v katerem je prišlo do zgrešitve, potreben tudi naslednji višji blok
   - Ob prenosu bloka *k* v predpomnilnik se iz glavnega pomnilnika prebere še blok *k + 1*, ki se shrani v ***bralni izravnalnik*** *(lahko ukazni in operandni ločena ali pa skupaj).* Če nato CPE naslovi blok k+1, se le ta prenese v predpomnilnik, kar je bistveno hitreje, kot pa prenos iz glavnega pomnilnika (zgrešitvena kazen je majhna)
   - To je učinkovito le, če je prenos dveh blokov približno enako hiter kot prenos enega ali če med prenašanjem dodatnega bloka CPE deluje
2. **Neblokirajoči predpomnilnik:**
   - Glej odgovor na prejšnje vprašanje, št.63

65. Kaj je to pomnilniško prepletanje? (dva načina za povečevanje hitrosti prenosov do glavnega pomnilnika)

- Ne glede na to, kakšen PP uporabimo, bo podatke v njem vedno potrebno prebrati iz glavnega pomnilnika in zapisati nazaj v glavni pomnilnik → zmanjšanje št. dostopov do GP ima spodnjo mejo.
- Preostane pa nam še to, da povečamo število naenkrat prenesenih besed. To lahko naredimo tako da:
  1. **Razširimo podatkovne poti do GP**
     - Glavni pomnilnik naredimo, da lahko dostopa do 8,16,32 ali več sosednjih pomnilniških besed (gre za dostop do sestavljenih pomnilniških besed)
  2. **S pomnilniškim prepletanjem:**
     - Glavni pomnilnik razdelimo na m samostojnih delov - modulov (lahko delujejo neodvisno od ostalih)
     - Vsak modul je samostojen pomnilnik, pri m modulih lahko poteka največ m dostopov istočasno.
     - Pomnilniško prepletanje je zelo koristno tudi zaradi tega, ker omogoča istočasne dostope do glavnega pomnilnika na računalnikih, ki imajo več CPE in/ali V/I procesorjev.
     - Pomembno nalogo ima tukaj krmilnik pomnilnika, ki iz pomnilniških naslovov, ki jih dajo procesorji ugotovi na kateri modul se nanašajo.
     - Dostop se opravi takoj, ko postane modul prost. Če želi dostopati prej, imenujemo to konfliktni dostop.

66. Kako pri pisanju v predpomnilnik dosežemo, da traja eno urino periodo (shranjevalni register)? =Kako se zagotovi, da se lahko v eni u.p. naredi bralni in pisalni dostop?

- Ker se zadetek ugotavlja istočasno z branjem, lahko branje brez težav naredimo v eni urini periodi.
- Pri pisanju pa je za ugotavljanje zadetka najprej potrebno branje → če imamo zadetek, mu sledi pisanje. To pomeni, da rabimo 2 urini periodi.

Kako realiziramo PISANJE v predpomnilnik v eni urini periodi:

1. Ločimo kontrolni in podatkovni del predpomnilnika

Zato, da lahko dostopamo do vsakega posebej

2. V predpomnilnik vgradimo shranjevalni register

- Pri pisanju se ugotavlja zadetek, podatek pa se skupaj z naslovom zapiše v shranjevalni register
- Če imamo zgrešitev se ob koncu urine periode vsebina registra razveljavi
- Ob naslednjem pisanju se tako podatek, med ugotavljanjem zadetka, iz shranjevalnega registra shrani na ta naslov.

67. Kakšna naj bo velikost bloka

Predpomnilnik lahko povečujemo tako, da povečujemo stopnjo asociativnosti, št. setov ali velikost bloka (najlažje in najceneje).

- S povečevanjem bloka se ustvarjajo pogoji za boljšo izkoriščenost prostorske lokalnosti,
  - ker pa se s povečevanjem pri isti velikosti predpomnilnika zmanjšuje število blokov, se s tem slabšajo pogoji za izkoriščanje časovne lokalnosti → od tod sledi, da lahko pri dani velikosti PP pričakujemo, da se bo s povečevanjem bloka verjetnost zgrešitve najprej zmanjševala, od neke vrednosti naprej pa bo naraščala
- Meritve kažejo, da so pri večjem predpomnilniku boljši večji bloki, pri manjšem pa manjši.
- Osnovno pravilo je, da mora biti število blokov pri dani velikost predpomnilnika dovolj veliko
  - Drugače se bloki stalno zamenjujejo in verjetnost zadetka je majhna (vedno pa velja, da če povečamo velikost PP pri dani velikosti bloka, to vedno izboljša verjetnost zadetka)
- Na velikost bloka vpliva tudi zgrešitvena kazen (je enaka času potrebnemu za prenos bloka v PP). Ta čas pa je sestavljen iz časa za prenos prve besede (temu času pravimo latenca) in iz časa za prenos ostalih besed bloka (ta čas je pogosto krajši od latence – page mode dostop) – zgrešitvena kazen bo večja, če povečamo blok, ne pohitrimo pa prenosa bloka.

68. Problem skladnosti pri predpomnilniku, centralni imenik, vohunjenje, MESI skladnostni protokol=Problem skladnosti – kaj je in kako ga rešujemo? Opiši vohunjenje! = Skladnost predpomnilnika + kje se nahaja centralni imenik?

Skladnost predpomnilnika:

- Da se vsebina nekega bloka v predpomnilniku ne razlikuje od vsebine istega bloka v glavnem pomnilniku in v drugih predpomnilnikih, če jih je več
- Če skladnost ni zagotovljena lahko pride do napačnega delovanja

Obstajata dva vzroka, zaradi katerih neskladnost povzroča napačno delovanje:

1. *Prenosi med V/I napravami in glavnim pomnilnikom*

Podatki iz diska se prenesejo neposredno v glavni pomnilni. Če je med naslovi, na katere so se pisali podatki iz diska tudi naslov, ki je v predpomnilniku, bo CPE pri branju dobil staro vrednost (problem tudi pri pisanju skozi)

2. *Pisanje na disk*

Če imamo predpomnilnik, ki uporablja pisanje nazaj. Na disk se bo zapisala stara vrednost iz glavnega pomnilnika in ne nova iz predpomnilnika

Probleme, ki jih povzročajo V/I prenosi, lahko rešimo na enega od naslednjih načinov:

1. V/I naprave priključimo tako, da gredo prenosi skozi predpomnilnik

- zapletena rešitev, verjetnost zadetka se občutno poslabša počasnejše delovanje računalnika

2. Pred izvrševanjem V/I prenosov se razveljavi vsebina predpomnilnika

- ali pa se vsi umazani bloki prenesejo.
- To je programski način in ne zahteva dodatne logike. To deluje, ker so V/I prenosi redki.

3. Selektivno razveljavljanje ali praznjenje predpomnilnika

Z dodatno logiko naredimo mehanizem, ki selektivno razveljavlja ali prazni predpomnilnik, imamo dva mehanizma:

3.1. Centralni imenik

- Informacija o naslovih vseh blokov, ki so trenutno v enem od predpomnilnikov je shranjena v centralnem imeniku
- Ta imenik je vgrajen v krmilnik pomnilnika
- Če pride do pisanja v blok, se to registrira v imeniku.
- Če je ta blok še v drugih predpomnilnikih, krmilnik poskrbi, da se blok zamenja/razveljavi.

3.2. Vohunjenje

- Ni centralnega imenika
- Uporablja se kadar so procesorji (ena ali več CPE in V/I procesorji) in glavni pomnilnik priključeni na isto vodilo, po katerem gredo vsi dostopi do GP
- Vsi procesorji oz. njihovi predpomnilniki znajo vohuniti za naslovi na vodilu
- Procesor, ki spremeni besedo v svojem predpomnilniku, pošlje na vodilo njen naslov skupaj z novo vsebino in signalom, da je prišlo do spremembe
- ***Vohunski zadetek:*** imamo kadar nek procesor ugotovi, da se spremenjena beseda, nahaja v njegovem PP
- Kadar imamo vohunski zadetek se:

  a) Pri pisanju se nova vrednost zapiše v predpomnilnik (ali pa se samo razveljavi blok, ki vsebuje to besedo)

  b) Pri branju bo procesor naredil naslednje:

  1. s posebnim signalom obvesti drug procesor, da naj opusti branje
  2. zapiše spremenjeni blok v GP
  3. obvesti procesor, ki bere, da naj ponovi branje

Skladnostni protokoli:

- S problemom skladnosti na večprocesorskih sistemih se ukvarjajo tudi skladnostni protokoli, ki določajo kaj naj se zgodi z bloki v predpomnilniku ob dostopu do njih.
- Med njimi je danes najbolj znan **MESI** (Modified, Exclusice, Shared, Invalid).
- Njegovo ime je okrajšava za 4 stanja bloka, ki jih predvideva.
- Zasnovan je na osnovi vohunjenja. Za programerja je neviden in deluje avtomatsko. Uporablja se v L2 in v operandnem predpomnilniku.

xx. Homogeni, nehomogeni in več-vhodni predpomnilnik

Če imamo en sam predpomnilnik za ukaze in operande, pravimo da je homogen, če pa je razdeljen v ukaznega in operandnega, pa je nehomogen.

L1 predpomnilniki so skoraj vedno nehomogeni (zaradi potrebe cevovoda, da se hkrati dostopa do ukaza in operanda), **L2 pa homogeni** (imajo večjo verjetnost zadetka, ker program sam razdeli predpomnilnik v ukazni in operandni po velikosti). L2 predpomnilniki imajo tako majhno verjetnost zadetka (80%) zaradi tega, ker že L1 izkorišča del lokalnosti.

V ukazni predpomnilnik se običajno ne piše, ker je to slab način programiranja, vendar današnji računalniki omogočajo tudi to. V večini računalnikov je ukazni predpomnilnik samo bralni. Problem se pojavi, če imamo tak program, ki spreminja svoje ukaze. Po navadi je rešen tako, da CPE razveljavi blok v ukaznem predpomnilniku, če se ugotovi pisanje vanj. Naredi se pisanje direktno v glavni pomnilnik (to je seveda zelo počasno). Če je beseda tudi v operandnem predpomnilniku, se blok v operandnem predpomnilniku razveljavi.

Danes se pri večini računalnikov zahteva (npr. pri superskalarnih računalnikih, kjer se v isti urini periodi izvršuje več ukazov), da CPE dostopa do več ukazov in operandov naenkrat, zato rabimo predpomnilnik, ki je sposoben v eni urini periodi narediti več dostopov do različnih naslovov. To lahko dosežemo na 2 načina:

- Za predpomnilnik uporabimo več-vhodno zgradbo, ki omogoča več istočasnih dostopov do pomnilniških celic. Težava je, da pri danem številu tranzistorjev ta način zmanjšuje predpomnilnik
- Predpomnilnik se razdeli na več samostojno-delujočih modulov (predpomnilniško prepletanje – ki je podobno pomnilniškemu prepletanju iz vprašanja 65). Omogoča več istočasnih dostopov pod pogojem, da se nanašajo na različne module. V nasprotnem primeru je potrebno čakanje.

V ukaznem je manj zgrešitev, ker je stopnja lokalnosti večja.

9. Navidezni pomnilnik

69. Pomen navideznega pomnilnika

- V preteklosti se je zaradi premajhnega glavnega pomnilnika delalo s ***prekrivki***.
- Program se je v glavni pomnilnik nalagal po delih. Ko je nek del opravil delo, se je poklical naslednji del iz pomožnega pomnilnika. Ta del je v celoti ali delno prekril prvi del programa v glavnem pomnilniku → od ***tu prekrivki***.
- Programerju se dandanes ni potrebno ukvarjati s prekrivki.
- Zanj je pomnilniški prostor v pomožnem pomnilniku videti, kot glavni pomnilnik.
- Prenosi med njima so za programerja nevidni, opravlja jih OS.

70. Zakaj se pri naslavljanju predpomnilnika uporabljajo fizični naslovi in ne navidezni?

Navidezni naslov se najprej preslika v fizičnega in ta se nato uporablja za naslavljanje predpomnilnika in glavnega pomnilnika.

- Čas za preslikovanje se neposredno prišteva k času dostopa

Razlogi za to so naslednji:

1. Sinonimi ali aliasi

- Na računalnikih z navideznim pomnilnikom želimo imeti možnost, da se več navideznih naslovov preslika v isti fizični naslov
- Te naslove imenujemo sinonimi ali aliasi
- Na ta način je mogoče doseči, da različni programi uporabljajo iste procedure ali podatke
- Pri predpomnilniku, ki uporablja navidezne naslove, je ista fizična beseda v predpomnilniku lahko na več mestih
  - če jo en program spremeni, drugi te spremembe ne vidijo in lahko pride do napake

2. Problem pri V/I prenosih

- Za V/I naprave je lažje, če pri prenosih v ali iz pomnilnika uporabljajo fizične pomnilniške naslove.
- Če predpomnilnik uporablja navidezne naslove, bi za zagotavljanje skladnosti potrebovali preslikovanje fizičnih v navidezne naslove

3. Multiprogramski način delovanja

- Ker si isti navidezni prostor deli več programov, se ob preklopu programov zgodi, da se isti navidezni naslov preslika v fizični naslov, ki je drugačen kot pri prejšnjem programu.
- Če se predpomnilnik naslavlja z navideznim naslovom, pride do dostopa, ki ustreza za prejšnji program veljavni preslikavi, kar je seveda narobe.

71. Osnovne vrste navideznega pomnilnika, prednosti/slabosti posamezne vrste (ostranjevanje, segmentacija, segmentacija z ostranjevanjem, prednosti segmentacije)
= Navidezni pomnilnik (katere dva poznamo, v čem se ločita hotel je slišati da imajo segmenti pomen, to je glavno)

1. Ostranjevanje:

- Pomožni pomnilnik je razdeljen na enako velike bloke, ki jim pravimo ***strani (pages).***
- Vse strani skupaj sestavljajo navidezni pomnilnik.
- Na enako velike bloke, ki jim pravimo ***okviri strani***, je razdeljen tudi GP.
  - Vsako stran navideznega pomnilnika je mogoče prenesti v poljubnega od okvirov.

Preslikovanje iz navideznega v fizični prostor poteka s pomočjo ***tabele strani*** *(vsaki strani navideznega pomnilnika pripada eno polje – deskriptor strani). Deskriptor je* sestavljen iz petih parametrov: V,P,RWX, umazani bit C, številka okvira FN)

  - V: veljavni bit → kadar je enak 1, so parametri v deskriptorju veljavni, drugače stran ni definirana in parametri nimajo pomena (običajno pri prvem dostopu do nekega naslova v strani)
  - P: prisotni bit → kadar je ta bit enak 1, je stran v enem od okvirov v glavnem pomnilniku (parameter FN pove v katerem). To ustreza zadetku v predpomnilniku in take strani imenujemo aktivne strani. Pri ostranjevanju se zgrešitvi pravi napaka strani.
  - RWX: zaščitni ključ → sestavlja ga več bitov, ki povedo, kakšna vrsta dostopa je dovoljena do te strani
  - C: umazani bit → ko se stran prenese v gl. pomn., se ta bit vedno postavi na 0. Če pride med izvajanjem programa do pisanja na kateregakoli od naslovov te strani, se postavi na 1. S tem je označeno, da se je stran, medtem ko je bila v gl. pomn. spremenila.
  - FN: številka okvira → ta parameter podaja številko okvira, v katerem je stran (FN je veljaven samo, če je P=1)

- Ker je število polj fiksno, je vnaprej določena tudi največja velikost tabele strani.
- Pri ostranjevanju lahko pride do Notranje fragmentacije → Glej 72
- …in premetavanja → Glej 73
- Slabost je, da ne omogoča uporabe boljših načinov za zaščito pomnilnika

2. Segmentacija:

- Navideznemu pomnilniku, ki ima pomnilniški prostor razdeljen na segmente, pravimo segmentacija.
- Segmenti so različnih dolžin, vsebina vsakega segmenta pa ima za program svoj **pomen**.
- Število in velikost segmentov se med izvajanjem programov lahko spreminjata
- Več uporabnikov lahko uporablja isti segment
- Segmenti je v bistvu ***strojni modul***, ki je preveden iz programskih modulov(podatkovne strukture, procedure, podprogrami).
  - V GP se lahko preslikajo na poljuben naslov.
  - Segmenti so med seboj sicer povezani, vendar je prehod iz enega v drugega mogoč samo z majhnim številom točno določenih načinov,
  - zato segmente smatramo kot večdimenzionalen prostor(vsak segment je ena dimenzija), za razliko od ostranjevanja kjer je prostor enodimenzionalen.
  - Preslikovanje poteka preko tabele segmentov, ki hranijo deskriptorje segmentov
- Parametri v deskriptorju:
  - P: prisotni bit → je ekvivalenten bitu P pri ostranjevanju. Kadar je 1, je segment v glavnem pomnilniku in parameter naslov segmenta pove, na katerem naslovu segment je. Dostop se izvrši samo, če to dovoljuje zaščitni ključ RWX.
  - RWX: zaščitni ključ → podobno kot pri ostranjevanju, vendar pa, ker ima vsak segment svoj pomen, je lahko zaščita popolnejša kot pri ostranjevanju.
  - C: umazani bit → isto kot pri ostranjevanju
  - L: velikost segmenta → vsebuje trenutno velikost segmenta. Največja možna velikost je 2^S.
  - SA: naslov segmenta → vsebuje fizični naslov segmenta v glavnem pomnilniku (če je bit P=1)
- Pri segmentaciji lahko pride do Zunanje fragmentacije → Glej 72

Segmente lahko razdelimo v dve vrsti:

1. **lokalni segmenti**, ki pripadajo enemu programu, dostop do lokalnih segmentov drugim programom ni dovoljen
2. **globalni segmenti**, do katerih lahko dostopajo vsi programi. Sistemski programi so običajno definirani kot globalni segmenti.

3. Segmentacija z ostranjevanjem:

- Vsak segment razdelimo na strani na način, ki je enak tistemu pri ostranjevanju.
- Vsak program ima poleg tabele segmentov tudi množico tabel strani – po eno za vsak segment.

Glavni izboljšavi segmentacije z ostranjevanjem sta:

1. Odstranitev zunanje fragmentacije

- Odpade potreba po zamudnem strnjevanju pomnilnika

2. Boljša izkoriščenost prostora, ki ga zasedajo segmenti.

- Zaradi delitve segmentov na strani ni več potrebno, da je v GP vedno cel segment. Zadošča, da so v njem samo tiste strani segmenta, ki se trenutno uporabljajo

72. Razlika med zunanjo in notranjo fragmentacijo

Notranja fragmentacija:

- vsak program zaseda določeno število strani.
- Ker pa je velikost program redko večkratnik velikosti strani, je v povprečju neizkoriščene pol strani na program.
- Notranja fragmentacija narašča z velikostjo strani

Zunanja fragmentacija:

- Vsak segment se mora prenesti iz pomožnega pomnilnika v zadosti velik zvezen prostor v GP
- Ponavadi ta prostor ne bo enako velik kot segment, zato bodo po nekaj prenosih v GP nastale luknje
- Pogosto se zgodi, da je vsota vseh lukenj dovolj velika za segment, vsaka posamezna pa ne
- Algoritmi za odpravljanje: najboljše ujemanje, najslabše ujemanje, prvo ujemanje (strnjevanje pomnilnika)

73. Kaj je premetavanje (trashing)?

- Trashing ali premetavanje je stanje računalnika, kjer je njegovo delovanje zreducirano na premetavanje informacije med glavnim in pomožnim pomnilnikom.
- Do njega pride pri stalnem velikem številu napak strani (do nove napake pride, še preden je končan prenos prejšnje)
- V tem primeru se koristno delovanje računalnika skoraj ustavi, saj se večino časa ne dela nič drugega kot prenašanje strani – uporabniški programi pa praktično stojijo.
- Za preprečevanje premetavanja je potrebno imeti nek mehanizem (potrebno je pravilno izbrati parametre, ki vplivajo na verjetnost za napako strani). Ti parametri so: velikost strani, velikost glavnega pomnilnika, strategija za zamenjevanje strani in pa število naenkrat izvajajočih programov (stopnja multiprogramiranja).

74. Kaj označuje rwx zaščitni ključ in kje se uporablja?

- RWX ključ je oblika zaščite strani/segmentov pri navideznemu pomnilniku (podobno zadevo imamo tudi pri GP).
- Pove kakšna vrsta dostopa je dovoljena (branje, pisanje) in kdo ima dovoljen dostop.
- Kadar iz zaščitnega ključa sledi, da dostop ni dovoljen, se sproži past zaščite

75. Kako ugotovimo ali je stran v navideznem pomnilniku prisotna?

- V tabeli strani za nek navidezni naslov poiščemo ustrezen deskriptor strani in pogledamo, če je bit P = 1.
- Če je imamo zadetek, drugače pa napako strani.

76. Kaj je TLB in zakaj ga rabimo?

Pri vsakem dostopu (tudi če podatek ni v navideznem pomnilniku) je potrebno narediti preslikavo navideznega naslova v fizični naslov.

- To pomeni, da sta pri vsakem pomnilniškem dostopu potrebna najmanj dva dostopa:
  - dostop do tabele strani v glavnem pomnilniku
  - in dostop do fizičnega naslova kjerkoli že je.
- Če imamo segmentacijo z ostranjevanjem ali večnivojske tabele se to poveča na 3 oz. 4 dostope.

Zaradi tega imajo računalniki z navideznim pomnilnikom v CPE vgrajen mehanizem, ki skrajša čas za preslikovanje

- Ta mehanizem je majhen predpomnilniku, ki vsebuje nazadnje uporabljene deskriptorje iz tabel strani / segmentov,
  - pravimo mu ***preslikovalni predpomnilnik*** ali ***TLB*** *(translation lookaside buffer)*.
- Dolžina bloka je običajno enaka 1 (vsak blok vsebuje en deskriptor)
- Pri segmentaciji z ostranjevanjem imamo običajno dva TLB-ja.
  - Eden vsebuje deskriptorje segmentov, drug pa deskriptorje strani.
- Če je preslikava večnivojska je dovolj, da se v TLB prenese samo deskriptor najnižjega nivoja.
- Če imamo ločen operandni in ukazni predpomnilnik moramo imeti tudi dva TLB-ja.

- Pri dostopu do nekega navideznega naslova se lahko zgodijo 3 vrste zgrešitev: v TLB, v predpomnilniku in v glavnem pomnilniku (napaka strani oz. segmenta)

Tipične lastnosti TLB-jev na današnjih računalnikih:

- Velikost TLB: 32 do 2048 deskriptorjev
- 1 do 2 deskriptorja na blok
- Čas dostopa pri zadetku 0,5 – 1 urina perioda
- Zgrešitvena kazen: 10-100 urinih period
- 99 do 99,9% verjetnost zadetka
- Dostop do TLB in preslikovanje tečeta paralelno z dostopom do predpomnilnika.
- Pri TLB se poleg set asociativnih predpomnilnikov uporabljajo tudi čisti asociativni, ker je mogoče doseči visoko verjetnost zadetka v TLB pri dosti manjši velikosti predpomnilnika

77. Kam se shranjujejo tabele strani (TLB)?

- V splošnem so tabele strani shranjene v GP
- Ker so lahko zelo velike, obstaja možnost shranjevanja tudi v navideznem pomnilniku, v tem primeru morajo biti deli OS v GP:
  - predvsem tisti program, ki opravlja zamenjavo strani
  - in tiste tabele ki preslikujejo naslove na katerih je OS
- Deskriptorji iz tabel od nazadnje dostopanih strani pa so v TLB-ju.

78. Kako še drugače rečemo zgrešitvi v navideznem pomnilniku?

Napaka strani.

79. Kako pospešiti preslikovanje navideznih naslovov v fizične v navideznem pomnilniku (TLB)?

S preslikovalnim predpomnilnikom(TLB).

80. Kako zmanjšati velikost tabele strani?

= Načini za zmanjšanje prostora za tabele strani / segmentov (Večnivojske t., Invertirana tabela strani, shranjevanje t. v NP). = Razpršitvena (hash) funkcija pri PowerPC.
= Kaj delamo če imamo prevelike tabele strani podrobno invertirane(hoče slišat hash funkcija:))

Prostora, ki ga zasedajo tabele strani in/ali segmentov, ni mogoče uporabiti za programe. Ta prostor potrebuje računalnik zato, da je mogoče opravljati preslikovanje naslovov. Gledano s stališča uporabnika je ta prostor neizkoriščen. Potrebujemo pametnejšo organizacijo tabel, in sicer:

1. večnivojske tabele strani:

- namesto ene zelo velike strani imamo več majhnih tabel, ki so lahko v različnih delih pomnilnika (ali celo v pomožnem pomnilniku). (Intel 80x86)

2. invertirana tabela strani:

- v tabeli so samo tiste strani, ki so v nekem trenutku v glavnem pomnilniku
- Problem je v tem, da iz številke strani v navideznem naslovu ni razvidno, ali in kje stran v glavnem pomnilniku je, zato moramo uporabit razpršev alno funkcijo.
- ***Razpršev alna (hash) funkcija*** preko razpršev alne tabele preslika številko strani v številko okvira strani
  - Če za neko številko strani ta preslikava obstaja, se dobljena številka okvira uporabi, kot indeks za tabelo strani → če ne obstaja, imamo napako strani
  - To rešitev uporablja PowerPC

3. shranjevanje tabel v navideznem pomnilniškem prostoru

- ni nujno, da so vse tabele strani v glavnem pomnilniku
- če zagotovimo, da so nekateri programi in nekatere tabele vedno v GP, je večina tabel strani lahko v navideznem pomnilniku
  - v GP morajo biti predvsem tiste, ki preslikujejo naslove, na katerih je operacijski sistem
- pri večnivojskem preslikovanju (vključno s segmentacijo z ostranjevanjem) je npr. tabela strani najvišjega nivoja vedno v glavnem pomnilniku. Tabele nižjih nivojev so brez težav lahko v navideznem pomnilniku.
- Shranjevanje tabel v navidezni prostor seveda nekoliko poveča število napak strani, vendar se ta rešitev vseeno veliko uporablja.

81. Kakšna je razpršilna funkcija hash?

- Razpršilna funkcija HASH iz nekih vhodnih podatkov izračuna neke majhne številke (oz. kaj drugega), ki jim pravimo HASH SUM.
- To pride prav npr. pri HASH tabelah, kjer se iz npr. besed izračunajo indeksi, ki kažejo na opise teh besed, kar pomeni, da je iskanje hitro. Lahko pride do kolizij → različne besede se preslikajo v isti HASH SUM.

82. Zakaj imamo večnivojske tabele strani? = Zakaj večnivojske tabele strani zasedejo manj prostora?

Večnivojske tabele strani se uporabljajo za dosego zmanjšanja velikosti tabele strani, saj mora bita v glavnem pomnilniku samo najvišji nivo, za ostale pa to ni nujno potrebno.

83. Servisiranje napake strani/segmenta?

Zgrešitev v TLB pomeni eno od naslednjih dveh možnosti:

1. **stran (ali segment) je v glavnem pomnilniku.**
   V TLB se it tabele strani ali segmenta prenese nov deskriptor in po prenosu se naredi dostop
2. **stran (ali segment) ni v glavnem pomnilniku.**
   Sproži se past za napako strani (ali segmenta) in OS poskrbi za prenos strani z diska in za spremembo parametrov deskriptorja te strani v tabeli strani.

<u>Kako vemo katera od teh dveh možnosti se je zgodila?:</u>

Pri TLB zgrešitvi CPE vedno naredi dostop do deskriptorja v tabeli strani ali tabeli segmentov

1. Če je v deskriptorju bit P enak 1, se deskriptor prenese v TLB.

2. Če je bit P enak 0, se sproži past za napako strani ali segmenta = ***past napake*** *(sproži se med izvajanjem trenutnega ukaza)*

(

<u>Dva načina za reševanje problema pasti napake:</u>

1. Ponovitev ukaza

- Ukaz v katerem je prišlo od napake se v celoti ponovi
- Ker je bil v trenutku nastopa napake del ukaza že izvršen, mora biti zagotovljeno, da že izvršeni del ne povzroči spremembe vsebine registrov
  - Pri RISC je to preprosto, ker se ukaz bere v stopnji IF – če ga prekinemo, ni naredil nič. Pri load/store samo preprečimo, da se ukaz konča. Pri load to pomeni, da ne piše v register, pri store pa preprečimo pisanje v pomnilnik
  - Pri CISC je bolj zapleteno, zato se pri ukazih, ki lahko pri ponovitvi povzročijo napako, ponavadi naredi preverjanje ali bo prišlo do napake strani in prične z izvajanjem samo, če napake ne bo.

2. Nadaljevanje ukaza

- CPE poskrbi, da se vsebina CPE pred pastjo prenese v sklad in po pasti nazaj v CPE

)

V knjigi:

- Servisni program napake mora shraniti stanje programa. To stanje obsega poleg PC še vsebino registra tabele strani in vse programsko dostopne registre. Nato mora ugotoviti navidezni naslov, pri katerem je prišlo do napake strani ali segmenta. Ta naslov je odvisen od tega, ali je do napake prišlo pri prevzemu ukaza ali pri dostopu do operanda. Pri ukazu je naslov v shranjenemu PC. Dobro pa je, da CPE ob odzivu na past shrani še naslov operanda, ki je v registru MAR.

- Ko OS pozna navidezni naslov, ki je sprožil past, naredi naslednje:
  1. Z uporabo številke strani v navideznem naslovu se v tabeli dobi naslov te strani na disku.
  2. Določi se, kateri od okvirov v glavnem pomnilniku se bo zamenjal. Če je umazani bit okvira postavljen, se mora prej v njem shranjeno stran zapisati nazaj na disk.
  3. Stran se prenese z diska v izbrani okvir. Vsebina deskriptorja v tabeli strani se spremeni tako, da odraža novo stanje te strani.

Zadnja 2 koraka trajata skupaj tipično 15-20 ms. To je veliko urinih period. Zato ta čas CPE raje izvaja nek drug program. Ko se izvedejo vsi trije koraki lahko CPE nadaljuje z izvajanjem prejšnjega programa takoj ali pa počaka, da pride pri sedaj izvršujočem programu do napake strani ali nekega drugega dogodka (to je pogosteje).

84. Kako velika naj bo stran v pomnilniku?
= Kaj vpliva na velikost strani pri navideznem pomnilniku (velikost sektorja diska, število napak strani, optimalni izkoristek [formule, odvajamo, dobimo ekstrem]). Izbira glede na verjetnost zadetka.
= (bolj podrobno sem moral opisat izkoriščenost pomnilnika ter napisat formulo za optimalno velikost strani)
=Kaj vpliva na izkoriščenost pomnilnika?
= kako izbrati velikost strani (velikost sektorja, izkoriščenost pomn. [omeniš notranja fragmentacija + tabele strani + enačba], št. napak)

Na odločitev o velikosti strani vplivajo trije faktorji:

1. **velikosti sektorja na magnetnem disku** (pomožni pomnilnik je na današnjih rač. skoraj vedno trdi magnetni disk)
   - Pri prenosih z diska ali na disk se vedno prenese najmanj en sektor (512 bajtov).
   - To pomeni, da je izkoristek najboljši, če je velikost strani večkratnik velikosti sektorja.
2. **izkoriščenost pomnilnika**
   - Pri navideznem pomnilniku imamo vedno opraviti z deli pomnilnika, ki jih uporabnik ne more uporabiti za svoje programe
   - Ti neizkoriščeni deli pomnilnika so dveh vrst:
     - notranja fragmentacija zaradi delitev na strani
     - prostor, ki ga zasedajo tabele strani
   - Z zmanjšanjem strani se zmanjšuje tudi notranja fragmentacija, obenem pa se povečuje velikost tabele strani.

   Enačba Optimalne velikost strani:
   - $S_{opt} = \sqrt{2cS_s}$ S_{opt} → Optimalna velikost strani
     S_s → Povprečna velikost segmenta (programa pri čistem ostranjevanju)
     c → dolžina deskriptorja je c besed
3. **število napak strani** (pomemben zaradi premetavanja)
   - je pomembnejši parameter kot izkoriščenost pomnilnika

   - Če povečujemo velikost strani, se do neke vrednosti število napak strani manjša, nato pa se začne večati. Seveda to velja za programe, kjer niso vse strani programa v glavnem pomnilniku istočasno. Tudi glavni pomnilnik mora imeti končno velikost.

   Opazujmo 2 zaporedna naslova A(i) in A(i+1), ki jih tvori CPE ob izvajanju nekega programa. Ta dva naslova sta med seboj oddaljena za neko število d. Vrednosti d-ja ne poznamo, vemo pa, da bo za oba naslova veljal eden od teh treh primerov:
   1. naslova A(i) in A(i+1) sta oba naslova ukazov → ukazi z operandi v registrih ali ukazi s takojšnjim naslavljanjem → d je običajno majhen, izjema so le skoki na oddaljene ukaze
   2. naslova A(i) in A(i+1) sta oba naslova operandov → pri ukazih, ki prenašajo polja, in pri prenosih v V/I naprave in iz njih → d bo običajno majhen, ker se polja skoraj vedno prenašajo po zaporednih naslovih
   3. naslov A(i) je ukaz, A(i+1) je operand ali obratno → ukazi, ki naslavljajo operande v pomnilniku → d bo običajno velik, ker je v večini programov operandom dodeljen prostor, ki je ločen od operandov

   - Sklepamo lahko, da bo v primerih 1 in 2 verjetnost zgrešitve 1-H s povečevanjem velikosti strani padala. Ker je v teh dveh primerih bolj verjetno, da je razdalja d majhna, je bolj verjetno tudi, da bosta A(i) in A(i+1) v isti strani, kar pomeni manjše število napak strani.
   - Drugače je v primeru 3. Tam se s povečevanjem velikosti strani veča tudi verjetnost za napako strani. Razdalja d bo namreč velika in zato tudi naslova A(i) in A(i+1) ne bosta v isti strani.

   Najti moramo pravo razmerje med tema dvema primeroma.

85. Algoritmi za optimalno uporabo navideznega pomnilnika

Na izkoriščenost računalnika vplivamo z izbiro pravil, s katerimi določimo naslednje:

1. **Dodeljevalna strategija**
   - Koliko okvirov strani v glavnem pomnilniku naj ima nek program
2. **Polnilna strategija**
   - Kdaj, katere in koliko strani naj se prenese iz pomožnega v glavni pomnilnik
3. **Zamenjevalna strategija**
   - Katere strani naj se prenesejo iz glavnega pomnilnika nazaj v pomožni pomnilnik

Cilj je zmanjšanje števila napak strani in s tem zmanjšanje nevarnosti za premetavanje.
Vse tri strategije upravljanja s pomnilnikom izvajajo algoritmi, ki so realizirani programsko in ne strojno.

- **Dodeljevalna strategija** je lahko:
  - fiksna (programu je dodeljeno fiksno število okvirov neodvisno od časa. Ne moremo dodeljevat okvirov drugim programom),
  - spremenljiva (če še imamo proste okvire, jih lahko dodelimo drugim programom),
  - lokalna (upošteva samo trenutne lastnosti rezidenčne množice)
  - ali globalna (upošteva tudi zgodovino rezidenčne množice vseh programov).
    - Rezidenčna množica so okvirji v glavnem pomnilniku, ki pripadajo enemu programu

- **Polnilna strategija**: polnjenje na zahtevo (zamenjava ob napaki strani) in vnaprejšnje polnjenje (prenese se še ena ali več drugih strani v pričakovanju da jih bo program potreboval kasneje). Te strani se lahko prenašajo istočasno z izvajanjem programa.

- Pri izbiri ustreznega algoritma moramo najprej analizirati dogajanje med izvajanjem programa. Stopnja lokalnosti ni konstantna, ampak jo lahko razdelimo na faze, v katerih se delovna množica spreminja počasi, saj je stopnja lokalnosti visoka in na prehode, za katere je značilna hitra sprememba delovne množice – stopnja lokalnosti je majhna. Ker je čas v fazah veliko daljši, kot prehodih, predstavlja število napak v prehodih njihovo večino. Torej je vnaprejšnje polnjenje boljše.

Algoritmi za dodeljevanje pomnilniškega prostora in zamenjevanje blokov:
1.WS (working-set) algoritem:
Glej → 86
2.PFF (page-fault frequency) algoritem:
Glej → 87

86. WS (Workingset) algoritem (moraš mu povedat samo bistvo tega algoritma) = Primerjava WS (Working set algoritma z) PFF (Page Freq. Fault) algoritmom?

To sta algoritma za dodeljevanje pomnilniškega prostora in zamenjevanje blokov:

1. WS (working-set) algoritem:

- WS (Working Set) algoritem uporablja pojem delovne množice strani W(t,T),
  - ki je definirana kot množica tistih strani, ki jih je program naslovil v časovnem intervalu (t-T,t),
  - pri čemer je t navidezni čas, ki ne teče ko program stoji,??
  - T pa parameter.?? (navidezni čas – parameter)
- Algoritem se aktivira za vsak program, ko je intervala konec (preteče navidezni čas T).
- Ob prvem aktiviranju dodeli programu neko fiksno število strani.
- Kasneje dodeljuje programu toliko strani, kot je okvirov v delovni množici.

- Če povzamemo, delujejo dodeljevalna, polnilna in zamenjevalna strategija, ki so potrebne pri vsakem navideznem pomnilniku, pri WS algoritmu na naslednji način:
  1. Dodeljevalna strategija: število okvirov strani se za vsak program določi ob vsakokratnem aktiviranju WS algoritma na način, ki je opisan zgoraj
  2. Polnilna strategija: ob vsaki napaki strani se v enega od prostih okvirov prenese stran iz pomožnega pomnilnika. Če prostega okvira ni, se program izloči iz množice aktivnih programov (stopnja multiprogramiranja pa se zmanjša za 1). Okviri, ki jih je zasedal izločeni program, so sedaj del množice prostih okvirov.
  3. Zamenjevalna strategija: ob aktiviranju WS algoritma se strani v okvirih, ki niso v množici W(t,T), prenesejo nazaj v pomožni pomnilnik. Pripadajoči okvirji so sedaj prosti in postanejo del množice prostih okvirov.ž

- Ko število prostih okvirov preseže določeno vrednost se stopnja multiprogramiranja spet poveča.

- Torej algoritem upošteva spremembe v lokalnosti,
  - zato daje zelo dobre rezultate,
  - vendar je njegova realizacija nerodna, saj ga je treba aktivirati vsakič, ko za nek program preteče čas T;
  - za vsak program teče T drugače.

87. Primerjava WS (Working set algoritma z) PFF (Page Freq. Fault) algoritmom?

PFF (page-fault frequency) algoritem:

- Ta algoritem je poenostavljena inačica WS algoritma.
- Še vedno imamo parameter T, vendar se algoritem aktivira samo ob napakah strani.
- Tako je tukaj čas med dvema aktiviranjema algoritma enak t(i) - t(i-1).
  - Če je ta čas < T, algoritem ne naredi ničesar (razen prenosa strani, pri kateri je prišlo do napake)
  - Če je ta čas > T, se programu (enako kot pri WS algoritmu) dodeli toliko okvirov strani, kot jih je v množici W. Ostali okviri postanejo del množice prostih okvirov.