? Grega
Signature Not Verified
Digitally signed by Grega
DN: cn=Grega, c=SI
Date: 2002.09.19
14:21:19 +01'00'

OSNOVE RAČUNALNIŠKE ARHITEKTURE

1. *Razložite pojma »računalniška arhitektura« in »organizacija računalnika«.*

Pojem arhitektura računalniških sistemov, ali krajše računalniška arhitektura, *so* skovali pri firmi IBM leta 1964 skupaj s serijo računalnikov IBM 360. Arhitekturo so definirali kot računalnik, kakor ga vidi programer na nivoju strojnega jezika. S tako definicijo so želeli opisati tisto abstraktno bistvo računalnika, ki ni odvisno od načina realizacije. Takrat se je prvič pojavila ideja o fizično različnih strojih z enako arhitekturo, na katerih lahko tečejo isti programi. Računalniška arhitektura je definirana kot stadij zgradbe, delovanja in gradnje računalnika, kot ga vidi programer na nivoju strojnega jezika, je zgradba računalnika, ki jo mora poznati programer, da lahko v strojnem jeziku piše pravilne programe. To pomeni, da z določitvijo ukazov v veliki meri izberemo tudi arhitekturo računalnika.

Pojem **organizacijo računalnika** se nanaša na logično zgradbo in lastnost delov, ki sestavljajo računalnik, ter na njihovo medsebojno povezavo. Neko arhitekturo je mogoče realizirati z različnimi vrstami organizacije. Namesto o organizaciji se pogosto govori o arhitekturi posameznih delov. Tako srečujemo pojme kot *so:* procesorska arhitekturo, mikroprogramska arhitektura, pomnilniška arhitektura, programska arhitektura in podobno. V novejšem času se zgradba in delovanje računalnikov vse pogosteje označuje samo kot arhitektura in ne s starejšim pojmom organizacija in arhitektura. To je v skladu s procesom poenostavljanja terminologije, skozi katerega gredo vsa mlada področja. Pojem računalniška arhitektura je danes širši kot je bil ob nastanku. Kadar je potrebno, se prvotna, ožja definicijo arhitekture označuje kot ukazna arhitektura.

2. *Struktura in funkcije računalniškega sistema*

3. *Zgodovinski razvoj računalnikov (obdobje mehanike, obdobje elektromehanike, obobje elektronskih računalnikov).*

Obdobje mehanike - na začetku 17. stoletja so se v Evropi prvič pojavili stroji, ki so sposobni izvajati osnovne štiri operacije, danes jih označujemo z besedo kalkulatorji. Prvi tak stroj je naredil Wilhelm Shickard leta 1623. Njegov stroj je znal seštevati in odštevati ter z dodatnim ročnim delom tudi množiti in deliti. Dosti bolj znan podoben stroj je tisti, ki ga je leta 1642 naredil Blaise Pascal. Pomembno izboljšavo je naredil Gotfried Leibnitz, ki je leta 1671 zasnoval stroj, ki je poleg seštevanja in odštevanja znal tudi množiti in deliti. Charies Bobbage je izumil stroj, ki je po zasnovi zelo podoben današnjim računalnikom. Zasnoval je in skušal zgraditi dva stroja, in to **diferenčni stroj** in **Analitični stroj**. Diferenčni stroj je bil namenjen za računanje in avtomatično tiskanje matematičnih tabel, poganjal ga je parni stroj. Od aritmetičnih operacij, je uporabljal samo seštevanje, vendar v povezavi z metodo končnih diferenc, ki omogoča izračun poljubnega polinoma. Analitični stroj je bil splošnejši in je lahko izvajal poljubno zaporedje aritmetičnih operacij. Za mehanizem je uporabil luknjane kartice (ukazne in operandne). Analitični stroj je bil prva naprava, ki združuje dve, njegovo delovanje vodi program in lahko rešuje poljubne probleme - prvi računalnik. Ta dva stroja sta bila narejena sicer le teoretično (tehnološko takrat nemogoče zgraditi). Leta 1843 Augusta Ada Byron napise prvi program (za izračun Bernulijeve enačbe) - prvi programer.

Obdobje elektromehanike - razvoj elektrotehnike je odprl nove možnosti pri realizaciji strojev za računanje. Ena od prvih uporab je bila uvedba elektromotorjev za pogon mehanskih kalkulatorjev, drugo pomembno področje uporabe predstavljajo naprave na osnovi luknjanih kartic, s katerimi je bilo mogoče sortirati in tabelirati velike količine podatkov. Herman Hollerith je leta 1887 prvič uspešno uporabil luknjane kartice pri statistični obdelavi podatkov. Holerith je leta 1896 ustanovil podjetje Machine Company, ki se je leta 1911 združilo z dvema drugima v novo podjetje, to pa se je leta 1924 preimenovalo v Internacionol Business Corporation ali IBM. Prvi uspešni poskusi nadaljevanja Babbogevega dela so se pojavili šele konec 1930-tih let neodvisno v Nemčiji in *ZDA*. V Nemčiji je delujoči programsko vodeni računalnik naredil Konrad Zuse leta 1941 imenoval se je Z3 (takšen kot si ga je zamislil Babagge). Vnos je bil preko tipkovnice, rezultat je prikazal poseben zaslon iz žarnic, števila predstavljena z plavajočo vejico. Hkrati je v ZDA IBM po načrtih Howarda Aikena leta 1943 naredil podoben računalnik imenovan Harvard Mark 1, ter ameriška firma BELL po načrtih Georga Stibitza prvi računalnik, ki je znal računati kompleksna števila - prvi računalnik za posebne namene. Gledano *s* stališča računalniške arhitekture so elektromehanski računalniki približno ekvivalentni Babaggeovem analitičnem stroju. Zato lahko rečemo, da je razvoj v tem obdobju bil le tehnološki in ne tudi arhitekturni.

Obdobje elektronskih računalnikov - uporaba elektronike je odpravila dve veliki pomanjkljivosti elektromehanskih strojev:

1.hitrost računanja je zaradi vztrajnosti gibajočih delov omejena,

2.zanesljivost delovanja je slaba zaradi velikega števila mehanskih elementov, kot so zobniki in vzvodi. Ker se elektrika uporablja za prenos informacije pri elektronskih, tako kot pri elektromehanskih strojih, vseeno imamo razliko v hitrosti. Pri preklopu elektromehanskega releja se fizično premakne kovinski del, ki ima svojo vztrajnost - časa za premik ni mogoče zmanjšati. V elektronskih računalnikih se premikajo samo elektroni - čas za preklop je zato veliko krajši. Že pred letom 1940 je bilo znano, da je logične funkcije mogoče realizirati tako z elektromehanskimi releji kot z vakuumskimi elektronkami. Takrat je več skupin raziskovalo možnost uporabe elektronskih vezij namesto elektromehaničnih v svojih napravah. Na University of Pennsylvania so bili razviti in narejeni najprej ENIAC in potem se EDVAC. Delo na ENIAC-u (Electronic Numerical Integrator and Calculator) se je pričelo leta 1943, vodila sta ga J. Mauchly in J. P. Eckert. Denar je zagotovila ameriška vojska in eden od glavnih motivov, je bila potreba po tiskanju balističnih tabel za razne vrste topov in bombnih merilcev v letalih. ENIAC je bil dokončan 1945. Operacije na ENIAC-u so bile približno 1000-krat hitrejše od tistih pri elektromehaničnih strojih in tudi veliko hitrejše od čitalnikov luknjanega traku ali kartic. Izraz programiranje se je prvič uporabljal prav pri ENIAC-u. Za vnos podatkov je služil čitalnik luknjanih kartic, rezultati pa so se izpisovali na tiskalnik ali na luknjalnik kartic. ENIAC, je bil v uporabi do leta 1955. Eden od avtorjev ideje o shranjenem programu (ker je pri ENIAC-u bil problem programiranje) je J. von Neumann. Kvaliteta ideje o shranjenem programu je, da se program postavi v notranjost stroja v obliki v pomnilniku shranjene informacije, ki je preko električnih impulzov vodi stroj. Delo na razvoju EDVAC-a (Electronic biscrete Variable Computer) se je pričelo po objavi von Neumannovega predloga 1945. EDVAC je bil dokončan šele leta 1951 in ga je prehitel v angleškem Cambrigu leta 1949 narejeni EDSAC, ki nosi naslov prvega delujočega računalnika s shranjenim programom. Leta 1946 je von Neumann dal pobudo za razvoj novega računalnika znanega pod imenom IAS (kratica za Institute for Advanced Study - kjer je delo potekalo). IAS je pričel delovati poleti 1951. Bil je dvojiški stroj in je imel pomnilnik na osnovi elektrostatične pomnilniške cevi - takim pomnilnikom danes pravimo, da imajo naključni dostop. Za realizacijo dostopa po naraščajočih naslovih so pri IAS uporabili poseben register z imenom programski števec, ki je prvič bil uporabljen že pri EDSAC-u.

4. Struktura Von Neumannovega računalnika

5. Moorov in Groschev zakon

6. Flinova delitev računalnikov.

Delijo se po dveh kriterijih:

- Število ukazov, ki se izvršujejo naenkrat (tok ukazov);
- Stevilo operandov, ki jih en ukoz obdeluje hkrati (tok podatkov).

Delijo se v štiri razrede:

- SISD (Single Instruction stream, Single data stream),
- SIMD (Single Instruction stream, multiple data stream),
- MISD (Multiple Instruction stream, Single data stream),
- MIMD (Multiple Instruction stream, multiple data stream)

7. Razložite pojme SISD, SIMD, MIMD in VLIW.

SISD (Single Instruction stream, Single data stream) - ti računalniki izvajajo naenkrat en sam ukaz na eni sami ponovitvi operandov (N=1). To je daleč najpogostejša oblika današnjih računalnikov in ustreza osnovnemu von Neumannovemu modelu. Tudi vektorski superračunalniki firme Cray spadajo med SISD.
SIMD (Single Instruction stream, Multiple bata stream) - ti računalniki izvajajo naenkrat en sam ukaz, vendar se ta opravlja naenkrat na več ponovitvah operandov (N>1). Označujejo se z izrazom array ali procesor array (procesorsko polje). Delujejo podobno kot navadni von Neumannovi računalniki. CPE ima eno kontrolno enoto (z enim samim programskim števcem PC) ter N aritmetično-logičnih enot in N množic registrov. Med starejšimi te vrste, je znan ILLIAC IV (N=64), med novejšimi pa Maspar MP-2 (N=512) in Connection Machine 2 *(CM-2,* N=65536). Pri operacijah na strukturah kot so matrike je učinkovitost SIMD računalnikov zelo dobra in tudi programiranje je preprosto. Povsem drugače pa je pri problemih, ki ne vsebujejo veliko takih struktur. Po letu 1985 je razvoj SIMD računalnikov počasi zamrl.
MIMD (MultipJe Instruction stream, Multiple bata stream) - ti računalniki izvajajo naenkrat več ukazov (različnih ali enakih) in vsak se izvaja na svojih operandih. Če so ukazi enaki, se MIMD računalnik spremeni v SIMD. MIMD računalniki so ocitno splošnejši od SIMD in omogočajo, da se izkoristi več vrst paralelizma. Zanje se uporabljata oznaki multiprocesorji in multiračunalniki. V bistvu gre pri MIMD za več med seboj povezanih Neumannovih računalnikov - imajo več procesorjev *(CPE),* ki so bolj ali manj tesno povezani. Pri tako imenovanih tesno povezanih so procesorji povezani preko skupnega pomnilnika, pri rahlo povezanih pa preko vhodno/izhodnih enot. V najbolj preprosti inačici imamo majhno število procesorjev (npr. 2 ali 4), od katerih vsak izvaja svoj program in med seboj skoraj ne komunicirajo. Takih računalnikov je veliko in jih štejemo kar med SISD - bolj natančno, kot več paralelno delujočih SISD. Pravi MIMD računalniki imajo od več deset do več tisoč procesorjev, ki sodelujejo pri reševanju istega problema. Med znanimi računalniki te vrste omenimo BBN (N=256), Connection Machine 5 (CM-5 N=1024) in Intel ASCI Red (N=9216). Narediti dober prevajalnik, ki zna izkoristiti vse zmogljivosti MIMD računalnika, je težak problem (veliko težji kot pri SIMD). Zadovoljivo, je bil resen šele po letu 1985 in takrat so pričeli MIMD računalniki prevladovati nad SIMD.

VLIW (Very Large Instruction Word) - z VLIW pristopom lahko močno zmanjšamo količino logike, ki je potrebna zo to, da se naenkrat izstavi več ukazov. V vsakem dolgem ukazu ima vsaka funkcijska enota svoj ukaz. Tipičen VLIW ukaz ima npr. 3 celoštevilske ukaze, 2 ukaza za operacije v plavojoči vejici, 2 dostopa do pomnilnika in skok. Sskupaj torej 8 ukazov za 8 funkcijskih enot. Zamisel o VLIW procesorjih ,je znana že iz sedemdesetih let in do sedaj ni bila posebno uspešna. Čeprav ima prevajalnik pregled nad celim programom in ima na razpolago vec časa kot procesor, je pri večini programov težko najti dovolj paralelizma za vse funkcijske enote. Za povečanje stopnje paralelizma uporabljajo prevajalniki sredstva, kot je razvijanje zank, ki odpravi veliko skokov in poveča možnosti za zaposlitev funkcijskih enot. Obstajajo pa tudi primeri, ko je paralelnost skoraj vedno dovolj. Na področju digitalnega procesironja signalov se veliko uporabljajo afgoritmi, ki so po svoji zgradbi zelo paralelni. Znana je serija VLIW signalnih procesorjev 32OC6x podjetja Texas Instruments, ki z funkcijskimi enotami dosegajo hitrost preko 3 GFLOPS (Giga FLOPS — Floating Point Operations per second). Z novimi idejami, kot so predikatni ukazi, so VLIW procesorji v zadnjem času postali nekoliko bolj zanimivi. Do sedaj je bila velika ovira uporabi VLIW procesorjev njihova programska nezdružljivost. Vsaka sprememba v številu funkcijskih enot, in tudi v času, v katerem enote izvršijo operacije (ta čas vpliva na medsebojno odvisnost ukazov), namreč zahteva drugačne programe.

8. Delitev SIMD in MIMD računalnikov.

SIMDn MIMD računalniki se pogosto označujejo kot paralelni računalniki, ker se istočasno (paralelno) izvaja več operacij. Vsi najbolj zmogljivi današnji superračunalniki so paralelni.

Delitev SIMD računalnikov:

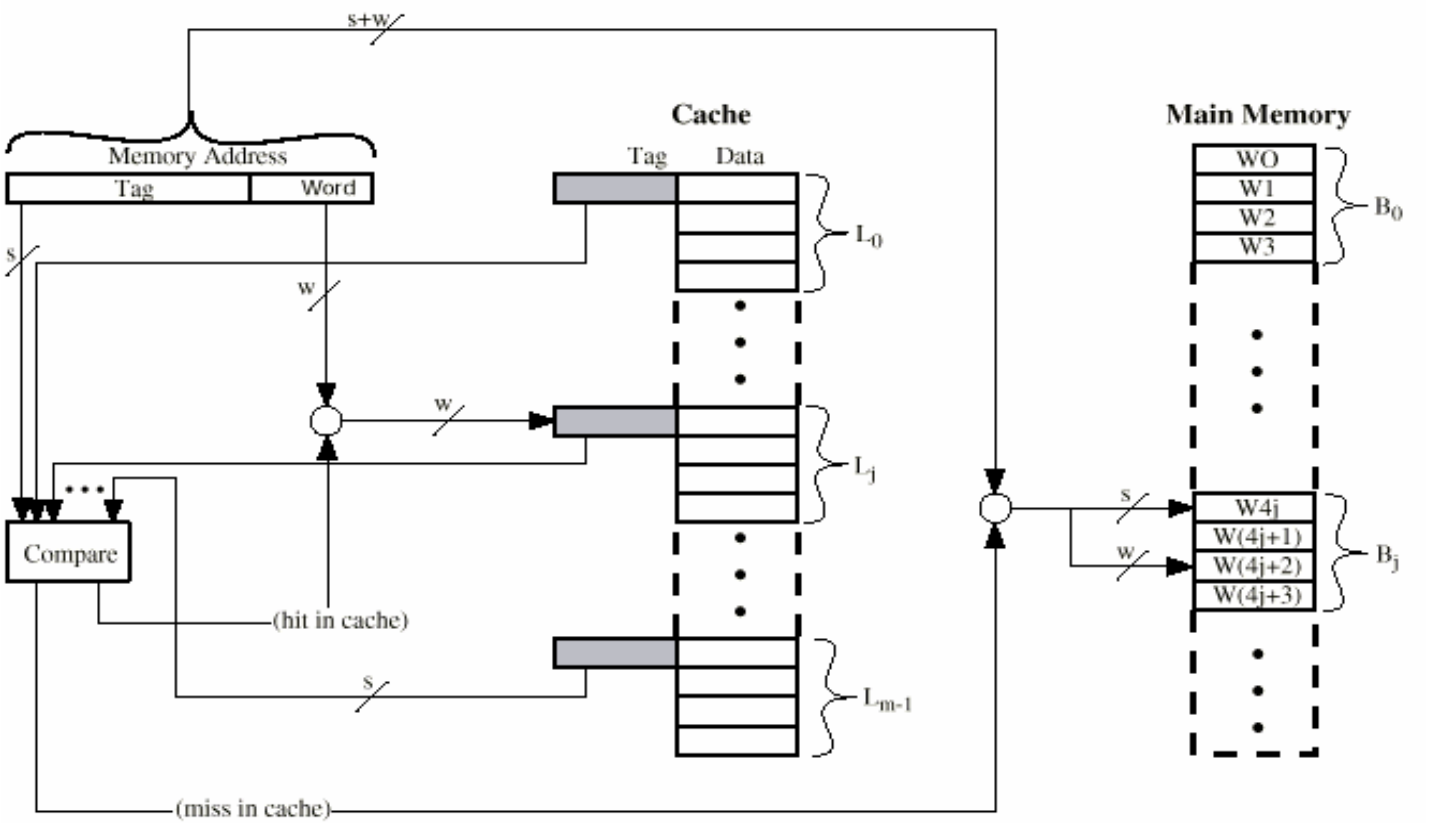
- LM SIMD (Local Memory SIMD) -pomnilniki so zraven procesorja.

- GM SIMD (Global Memory SIMD) - pomnilniki so ločeni od procesorja.

Razlika med LM (NUMA - Nonuniform Memory Access) in GM (UMA - Uniform Memory Access) je čas dostopa do pomnilnika med procesnimi enotami.

Delitev MIMD računalnikov:
- Močno sklopljeni sistemi (Loosely Coupled)
- Šibko sklopljeni sistemi (Tightly Coupled)

9. Principi delovanja asociativnih pomnilnikov.



Zelo pomembna lastnost vsakega pomnilnika je način izbire pomnilniške besede, do katere se želi dostopiti. Glede na to se današnji pomnilniki delijo v dve skupini. V prvi je beseda podana z naslovom. Pri pomnilnikih iz druge skupine besede nimajo naslovov -dostop do njih poteka preko vsebine besede. Pomnilnikom te vrste pravimo asociativni pomnilniki. Dostop do besed v asociativnem pomnilniku poteka preko vsebine besede, ali bolj pogosto preko dela vsebine besede. Asociativni pomnilniki se zato imenujejo tudi vsebinsko-naslovljivi ali tudi paralelno-iskalni. Običajno delujejo tako, da pri dostopu podamo del vsebine besede. Pomnilnik nato poišče besedo, v kateri se deli besede ujema s podanimi biti; to je naslovljena beseda, ki se bere ali v katero se piše. Če besede s podanimi biti v pomnilniku ni, dostop ni možen in pomnilnik to sporoči s posebnim signalom. Primerjava in iskanje besed, ki sta vgrajena v vsak asociativni pomnilnik, sta realizirana elektronsko in potekata zelo hitro. Vhodni biti, ki se podajo pri, dostopu, se naenkrat (paralelno) primerjajo z vsebino vseh besed - ni potrebno, da bi zaporedoma brali vsebino besed in jo primerjali. Besede v asociativnem pomnilniku si lahko nekoliko poenostavljeno predstavljamo, kot množico navadnih pomnilniških celic (enakih kot pri pomnilnikih z naključnim dostopom) organiziranih v besede, v katerem ima vsaka beseda svoj komparator, *s* katerim primerja svojo vsebino s podonim zaporedjem bitov. Taka realizacija zahteva zelo veliko elementov in zato so asociativni pomnilniki običajno majhni - več kot

128 besed se uporablja zelo redko. Očitno je tudi, da čas dostopa do asociativnega pomnilnika ne more biti krajši, kot je čas dostopa do uporabljenih pomnilniških celic, ki so njegova osnova. Zaradi časa, ki ga potrebuje logika za primerjavo, je čas dostopa asociativnega pomnilnika nujno nekoliko daljši od časa dostopa do navadnega pomnilnika v njem. Pač pa postane asociativni pomnilnik zelo hiter v primerih, ko želimo ugotoviti ali in kje v pomnilniku se nahaja določena vsebina. V računalnikih srečujemo asociativne pomnilnike predvsem v predpomnilnikih. Pri njih je namreč ob vsakem dostopu potrebno ugotoviti, ali se dani naslov nahaja v predpomnilniku ali ne. To je operacija, za katero je asociativni pomnilnik kot ustvarjen.

10. Razložite pojme: asinhrona in sinhrona sistolična polja, rekonfigurabilni procesorji.

Sinhrona sistolična polja: po urinem taktu, delujejo na principu sinhronih operacij. Vse dela po taktu, program je zapečen.
Asinhrona sistolična polja: operacija se izvede, ko so prisotni podatki in ne po urinem taktu.
Rekonfigurabilni procesor: struktura se lahko prilagodi določenemu procesu. (povezovanje, določimo kateri procesor bo s katerim deloval)

11. Naloge krmilne enote, aritmetično logične enote in registrov v CPE

CPE si lahko predstavljamo razdeljeno na dva dela **kontrolno enoto** in **podatkovno enoto**. Kontrolna enota generira kontrolne signale, ki vodijo delovanje podatkovne enote in tudi delovanje ostalih enot računalnika. Podatkovna enota vsebuje aritmetično-logično enoto (ALE) in registre. Med registri so tako programsko dostopni registri kot tudi programsko nevidni registri. V podatkovni enoti se opravijo vse operacije, ki jih zahtevajo ukazi. Lastnosti te enote zato močno vplivajo na zmogljivost CPE. Pri tipični CPE predstavlja podatkovna enota približno polovico prostora in logičnih vezij. Delovanje podatkovne enote je zelo preprosto. Za prenos podatkov ima tri vodila S1, S2 in D. Vodili S1 in S2 služita kot vhod v *ALE,* vodilo D pa kot izhod iz ALE. Ker je programsko dostopnih registrov veliko, prihaja do električnih težav če želimo na vodilo priključiti vse. Zato je v večini primerov bolje, če programsko dostopne registre R0 do R31 naredimo v obliki posebne enote, ki ji pravimo registrski blok.
Registrski blok je priključen na vodila preko dveh izhodnih registrov A in B ter vhodnega registra C. Podatkovna enota je pasivna in brez kontrolnih signalov ne dela ničesar. Urin signal je vedno priključen na vse registre, ker se vsa pisanja v register izvršijo ob aktivni fronti ure na koncu urine periode. Tudi delovanje ALE določajo kontrolni signali. Vse ALE operacije se izvršijo v eni urini periodi. Poleg programsko dostopnih registrov R0 do R31 sta za uporabnika vidna še programski števec *PC,* 'exeption' programski števec EPC. Za komunikacijo s pomnilnikom in drugimi deli računalnika imamo programsko nevidna registra MBR (podatkovni) in MAR (naslovni). Odločitev za podatkovno enoto s tremi vodili je posledica zahteve, da se morajo vsi ALE operacije izvršiti v eni urini periodi. Ker so ALE ukazi 3-operandni potrebujemo najmanj 3 vodila. ALE je čisto kombinatorično vezje, ki ne vsebuje nobenega pomnilniškega elementa. Naloga krmilne enote je, da za vsak ukaz ve, kateri koraki so potrebni, in da v vsaki urini periodi aktivira vse tiste signale, ki te korake izvršijo v pravilnem vrstnem redu. Delovanje kontrolne enote lahko podamo z diagramom prehajanja stanj. Diagram ima končno število stanj in povezave, ki določajo prehode med stanji. Prehod iz enega stanja v naslednje se zgodi ob aktivni fronti urinega signala. V vsakem stanju diagrama *sta* definirani dve funkciji: funkcija naslednjega stanja (določa katero je naslednje stanje), funkcija izhodnih signalov (natanko določa kateri izhodni signali so v tem stanju aktivni).

12. Faze izvajanja ukazov.

Faze izvajanja ukazov:

- vstavljanje ukaza;
- dekodiranje ukaza;
- vstavljanje operandov (registri, pomniinik);
- izvajanje ukaza (aritmetično-logična enota);

- shranjevanje rezultata.

13. Harwardska in VonNeumanova računalniška arhitektura.

V von Neumannovem računalniku je glavni pomnilnik videti kot nekakšno skladišče, v katerem so shranjeni ukazi in operandi. Njegovo delovanje je pasivno: naredi samo tisto, kar od njega zahtevata CPE in V/I sistem. Za zmogljivost računalnika je zelo pomembno, da se iz glavnega pomnilnika v CPE in iz CPE v gl. pomnilnik lahko prenese dovolj velika količina informacije. Iz delovanja von Neumannovega računalnika sledi, da ta naredi samo tisto, kar se pred tem prenese v CPE. Prenose med CPE in pomnilnikom pogosto označujemo kot promet. Temu ustrezno se pot med *CPE* in glavnim pomnilnikom, ki je ozko grlo von Neumannovega računalnika, označuje pogosto z izrazom "von Neummanovo ozko grlo". CPE uporablja glavni pomnilnik tako, da poda naslov pomnilniške besede (najmanjše število bitov s svojim naslovom) in vrsto prenosa (smer prenosa informacije), ki je lahko branje (iz pomnilnika v CPE) ali pisanje (iz CPE v pomnilnik). Branju in pisanju rečemo kar dostop do pomnilnika. Naslov pomnilniške besede je nespremenljiv (ista beseda ima glede na CPE vedno isti naslov). Število bitov s katerim je podan naslov imenujemo dolžina naslova. Dolžina naslova določa velikost pomnilniškega prostora - naslovni prostor. Večji je pomnilniški prostor več je lahko glavnega pomnilnika. Vsebina pomnilniške besede se lahko spreminja (npr. 8 bitna pomnilniška beseda ima lahko 256 različnih vsebin).
Harwardska je enaka kot von Neumannova le z eno razliko. Eden od načinov za razširitev ozkega grla je, da glavni pomnilnik razdelimo v dva dela. V prvem so ukazi, v drugem pa operandi. Branje ukaza in operanda lahko poteka istočasno. Če je CPE narejena tako, da traja branje ukaza približno enako dolgo kot njegovo izvrševanje, dosežemo *s* tako razdeljenim pomnilnikom dvakrat večjo hitrost. Računalnikom s tako urejenim pomnilnikom pravimo, da imajo Harvardsko arhitekturo. V skladu s tem potem pravimo računalnikom s pomnilnikom, ki ni razdeljen, da imajo Princetonsko arhitekturo. Obe arhitekturi sta inačici von Neumannove. Pri obeh vrstah von Neumannove arhitekture je glavni pomnilnik videti kot enodimenzionalno zaporedje tako imenovanih pomnilniških besed, od katerih ima vsaka svoj enoličen naslov. Za von Neumannove računalnike je značilno, da iz vsebine pomnilniške besede ni mogoče vedeti, ali je v njej ukaz ali operand (pri Harvardski arhitekturi je to vprašanje trivialno). Podobno pri operandih ni mogoče vedeti, ali je vsebina neke besede število ali npr. črka.

14. Naštejte najpomembnejše lastnosti dobre računalniške arhitekture.

15. ***Naštejte faktorje, ki vplivajo na uspeh računalniške arhitekture***.

Faktorji, ki vplivajo na uspeh računalniške arhitekture:

- zanesljivost (reliability);
- dosegljivost (availability);
- razširljivost (scalability),
- odprtost (openness) : dostop do arhitekture vsem uporabnikom:
- enostavnost uporabe (easy of use);
- zmogljivost (performance);
- *čas* izvajanja programa (execution time);
- čas odziva (response time);
- propustnost (throughput): število obdelanih poslov / sekunda;
- pasovna širina (bandwidth): število prenesenih informacij / sekunda;
- cena (price);
- raztegljivost (malleability): na koliko načinov ga lahko implementiramo;
- prikladnost (applicability): prilagoditev arhitektur aplikacijam;
- učinkovitost (efficiency): uporabnost računalniških komponent;
- splošnost (generality): bolj kot je splošen večji nabor ukazov ima, večnamenski ali specifično namenski

16. ***Amhdalov zakon***.

vzemimo, da za faktor N pohitrimo delovanje pri vseh operacijah razen pri f-temu delu od vseh operacij - drugače povedano, delovanje je N-krat hitrejše (1-f)-ti del časa računanja in ostane enako hitro preostali f-ti del. Potem je povečanje hitrosti računalnika S(N) enako S(N) = 1 / f + (1 — f)/N= N / (1 + (N-1)*f). Amdahlov zakon v bistvu pravi, da je povečanje hitrosti računalnika omejeno s procentom časa, v katerem ostane hitrost nespremenjena. Ugotovitev o težavah pri nadomeščanju enega procesorja z N procesorji je ena od posledic Amdahlovega zakona. Druga posledica se nanaša na hitrost operacij v samem procesorju. Pomembno je, da pohitrimo predvsem tiste operacije, ki so najbolj pogoste in za katere se v povprečju porabi največ časa.
Bistvo Amdahlovega zakona je v tem, da povečevanje hitrosti samo nekaterih operacij ne pomaga veliko. Po Case/Amdahlu je računalnik dobro zasnovan takrat, kadar izpolnjuje zahteve naslednjih dveh pravil:

- Prvo Case/Amdahiovo pravilo: velikost glavnega pomnilnika v bajtih mora biti najmanj enaka številu ukazov, ki jih v eni sekundi izvede CPE.
- Drugo Case/Amdahlovo praviio: zmogljivost V/I sistema v bitih na sekundo mora biti najmanj enaka številu ukazov, ki jih v eni sekundi izvede CPE.

Bistvo Amhdalovega zakona je v tem da povečevanje hitrosti samo nekaterih operacij ne pomaga veliko.

17. ***Metrike in enote za merjenje zmogljivosti računalniške arhitekture***.

$\sum_{i=1}^{N} x_i = \{x_i\}$ - množica meritev	$\sum_{i=1}^{N} w_i = \{w_i\}$ - množica uteži
$\frac{\sum_{i=1}^{N} x_i}{N}$ - aritmetična meritev	$\frac{N}{\sum_{i=1}^{N} \frac{1}{X_i}} =$ - harmonična vrednost
$\frac{\sum_{i=1}^{N} (w_i x_i)}{\sum_{i=1}^{N} w_i}$ - utežna aritmetična vrednost	$\sqrt[N]{\prod_{i=1}^{N} x_i}$ - geometrična vrednost

Enote: - MBS megabyte per s za vodila in V/I enote, - MIPS milion instruction per s niso učinkovite ker imamo različne arhitekture, - MFLOPS milion floating point operation per s operacije so primerljive med različnimi arhitekturami, - SPECFP, SPECINT obstaja SPECMARK test, kjer se izvaja 10 operacij, dajo realno stanje splošno namenskega računalnika, - WAX, IBM MIPS podobno kot MIPS merilo je WAX 780 in IBM 370, - WETHSTONW, DRYSTONE temeljita na vetstonovem in drystonovem testu. Za obod imata trigonometrične funkcije, dobimo moč procesorja, - TPS transaction per s točno se določijo transakcije, osnova je debit/kredit test (zapis in branje podatkovne baze), sodeluje cela struktura, - LINPACK test najbolj neusmiljen do računalnikov, meri hitrost komunikacije med CPE in RAM na podlagi linearnih diferencialnih enačb, koda tega testa je javno dostopna, - LIVERMORE LOOPS test tudi testira CPE/RAM gre za dvodimenzionalna polja, kopira iz enega polja v drugo.

18. Enote računalniškega sistema

19. Funkcije registrov: MAR, MBR, PC, AC, IR, AMASK, SMASK, 0, +1, -1, SP

20. Kako poteka branje in pisanje iz pomnilnika v CPE?

21. Na primeru razložite princip izvajanja ukaza v računalniku?

22. ***Lastnosti strojnega jezika, ki ga uporablja računalniški sistem, ki smo ga spoznali ko smo razlagali mikroprogramiranje.***

23. Katera naslavljanja uporablja računalniška arhitektura, ki smo jo spoznali, ko smo razlagali mikroprogramiranje.

24. Cikel izvajanja mikroukaza.

25. ***Problemi, ki nastopajo pri cevljenem izvajanju ukazov.***

- težko je doseči maksimalno mogočo hitrost zaradi različnega števila operandov, načinov naslavljanj in časovnih razlik ukazov (segmenti se preskočijo - preprosti ukazi pridejo skozi cevovod hitreje kot zapletenejši);
- medsebojna odvisnost ukazov - uporaba operanda, ki je rezultat prejšnjega ukaza (data dorwaeding - kopija rezultata se shrani v *CPE);*
- skočni ukazi ter prekinitve in pasti (del vsebine se zavrže; tudi druge rešitve);
- strukturne nevarnosti - do teh nevarnosti pride, kadar več stopenj cevovoda v neki urini periodi potrebuje isto enoto. Pod eno enoto tu mislimo registre, ALE in pomnilnik. Če imamo samo eno enoto, mora ena od stopenj počakati;
- podatkovne nevarnosti - te nevarnosti bi bilo bolj pravilno imenovati operandne nevarnosti. bo njih pride, kadar ukaz potrebuje kot vhodni operand rezultat prejšnjega, še ne dokončanega ukaza;
- kontrolne nevarnosti - do teh nevarnosti pride pri skokih, klicih in drugih kontrolnih ukazih, ki spreminjajo vsebino programskega števca *PC. Če* se npr. PC spremeni šele v tretji stopnji cevovoda, ni mogoče vedeti, katere ukaze naj cevovod izvršuje v prvih dveh stopnjah.

26. Na katere načine lahko povečamo hitrost procesiranja v računalniku.

- povečanje urinega takta
- povečanje kompleksnosti procesiranja (latch - za vmesno hranjenje podatkov)
- povečanje števila registrov (pridobimo na času dostopa do operandov)
- izboljšanje pasovne širine dostopa do podatkov (izkoriščamo zaporednost dostopa do podatkov ali ukazov)
- povečanje pretoka *s* pomnilniki (izravnalniki - ukazni, podatkovni; dostop je enak kot do registra)
- povečanje podatkovnega vodila (povečanje računalniške arhitekture *(ALU,* registri, sklad...))
- uvedba superskalarnosti (več paralelnih cevovodov)
- optimizacija algoritmov, mikroukazov, prevajalnika... (program prilagojen arhitekturi računalnika)

27. ***Prednosti in slabosti uporabe mikroprogramiranja glede na neposredno izvajanje strojnih ukazov.***

Pri mikroprogramiranju imamo v *CPE* vgrajen še en specializiran računalnik, ki skrbi za izvajanje ukazov. Pri vsakem ukazu se aktivira ustrezno zaporedje ukazov tega specializiranega računalnika - njegovim ukazom pravimo mikroukazi, zaporedju mikroukazov pa mikroprogram. Vsak ukaz aktivira enega od mikroprogramov, ki so hranjeni v t.i. kontrolnem pomnilniku v *CPE.* Mikroukazi so bistveno primitivnejši od strojnih ukazov in jih

izvršuje trdo ožičena logika. Na mikroprogramiranih računalnikih imamo pod običajnim strojnim nivojem še mikroprogramski nivo, ki je pravzaprav pravi strojni nivo. Izvajanje tako realiziranih ukazov je počasnejše kot pri uporabi trdo ožičene logike. Pač pa je preprosto spreminjanje ali dodajanje ukazov, ker zahteva samo spremembo mikroprogramov. Obstajajo tudi računalniki, pri katerih je del ukazov realiziran *s* trdo ožičeno logiko, del pa z mikroprogrami.

28. Horizontalna mikroprogramirana arhitektura

Horizontalno mikroprogramiranje - s tem imenom se označuje uporaba mikroukazov, pri katerih je malo ali nič kodiranja. Pri tej rešitvi je mikroprogramski pomnilnik širok, vendar nizek (ima malo naslovov). Ker ni potrebna logika za dekodiranje, je horizontalno mikroprogramiranje hitrejše, je pa tudi dražje.

29. Vertikana mikroprogramirana arhitektura

Vertikalno mikroprogramiranje - s tem imenom se označuje uporaba močno kodiranih mikroukazov, običajno z več formati. Tu je mikroprogramski pomnilnik ozek, vendar visok (ima več naslovov, ker je lahko potrebnih več mikroukazov). Vertikalno mikroprogramiranje je običajno cenejša in počasnejša rešitev.

30. Razlike med horizontalno in vertikalno mikroprogramirano arhitekturo

Zmanjšanje širine mikroukazov lahko dosežemo z uporabo kodiranja. S kodiranjem lahko naredimo mikroukaze bistveno ožje, vendar kodiranje ni čisto zastonj. Zanj potrebujemo dodatno logiko, ki iz kodirane informacije tvori (dekodira) potrebne kontrolne signale. Te logike je običajno dosti manj kot znaša prihranek pri mikroprogramskem pomnilniku, se pa v njej porabi nekaj časa. To ima lahko za posledico podaljšanje urine periode t. Glede na stopnjo uporabljenega kodiranja so mikroukazi lahko široki ali ozki. Zato razlikujemo možnost horizontalnega in vertikalnega mikroprogramiranja. Ne obstaja splošno veljavna definicija, na osnovi katere bi za neko rešitev lahko rekli, da je horizontalna ali vertikalna. Kljub temu je razlikovanje pomembno: če drugega ne sta z njim označeni obe skrajnosti.

31. Napišite program v zbirniku, ki smo ga spoznali pri razlagi mikroprogramiranja (definicije programov bodo različne).

32. Na različnih primerih mikroukazov opišite strukturo MIR (tako za horizontalno, kot tudi za vertikalno organizacijo).

33. Napišite mikroprogram za horizontalno mikroprogramirano arhitekturo, ki bo znal simulirati ukaz (opis različen za različne ukaze). Z mikroprogramom opišite samo izvedbo ukaza, ne pa tudi faze dekodiranja.

34. Kakšne izboljšave lahko predlagamo pri računalniški arhitekturi, ki uporablja mikroprogramiranje?

35. Predstavitev celih števil in znakov v računalniku.

Znaki:

- ASCII (7 bitni ISO LATIN1 ni šumnikov potrebni gonilniki), (8 bitni ISO LATIN2 uvede se dodatni nabor znakov),
- UNICODE 16 bitov (software) v računalniku je zapisano še 8 bitno zaradi kompatibilnosti,

- EBCDIC 7 bitni Extended Binary Coded Decimal Interchange Code format še danes na IBM main frame.

Cela št:

- BCD koda (4 biti), - število s predznaki (binarni zapis, prvi bit je predznak. Slabost +0,-0),
- eniški komplement (0 zamenjamo z 1, slabost +0,-0, se ni uveljavilo,
- dvojiški komplement (0→1, 1→0 in prištejemo 1. prednost enostavno,
- devetiški komplement (osnova je BCD koda, prištejemo 9 in dobimo negativno,
- desetiški komplement (devetiškemu prištejemo 1),
- predstavitev v EXCESS-N kodi (prednost enostavno pri množenju),
- zapis ulomkov z radix-om.

36. Prednost predstavitve celih števil z dvojiškim komplementom?

37. Seštevanje in množenje celih števil, ki so predstavljena z dvojiškim komplementom?

38. Elementi strojnega ukaza.

39. Ortogonalnost ukazov. Napišite izračun poljubnega aritmetičnega izraza s 3+1 operandnimi ukazi, 3 operandnimi ukazi, 2 operandnimi ukazi, 1 operandnimi ukazi in brezoperandnimi ukazi. Narišite tudi Ganttov diagram za podano strukturo CPE.

40. Predstavitev realnih števil v računalniku.

41. Vrste registrov.

- za programerju nevidni registri - IR (ukazni registri), *MAR* in MBR (za dostop do pomnilnika), maskimi (izločajo kodo operandov);
- programsko dostopni registri (stanje CPE) - podatkovni (shranjujemo podatke, vmesne rezultate), naslovni (shranjevanje naslovov), PC (programski števec, kazalec na trenutno lokacijo ukaza ki se izvaja), SP (kazalec sklada)
- za shranjevanje zastavic (PSW) - stanje posameznih procesnih enot;
- za nadzor prekinitve;
- ura realnega časa;
- za realizacijo navideznega pomnilnika - programi, ki so daljši, se razdelijo in se po blokih prenašajo na naslove v pomnilnik, registri imajo kazalec na začetek tabele kjer so shranjeni naslovi blokov;
- sklad - shranjevanje programsko dostopnih registrov, shranjevanje vmesnih trenutnih spremenljivk, uporaba v primeru klica v podprogramu in prekinitvah, v vsakem trenutku dosegljiv samo vrh sklada (ena beseda), nova beseda se shrani na vrh, LIFO način shranjevanja (zadnji noter, prvi ven), PUSH (prenos iz gl. pomnilnika v sklad), POP (prenos iz sklada), prednost je preprosta realizacija, kratki ukazi, preprosti prevajalniki;
- akumulator - pomnilnik narejen iz samo enega registra (lahko shranimo samo en operand), LOAB (prenos iz gl. pomnilnika v akumulator), STORE (prenos iz akumulatora), kratki ukazi, preprosti prevajalniki, slabost je, da je v njem mesta samo za en operand in je treba vmesne rezultate prenašati v pomnilnik, posledica je počasnejše delovanje in veliko število prenosov med CPE in pomnilnikom;

- množica registrov - dostop možen brez omejitev, *CPE* kot množica akumulatorjev, vsak register ima svoj naslov, število registrov je od 8 do 100 in več, dve rešitvi glede na svobodo pri uporabi registrov v množici - 1.vsi registri ekvivalentni, vsakega lahko uporabimo v vsakem ukazu (splošno namenski), 2.množica registrov razdeljena v dve skupini, prva skupina za aritmetično-logične operande (akumulatorji), druga skupina za računanje z naslovi (bazni ali indeksni), množica registrov je uporabnejša kot akumulator in sklad, aritmetični ukazi se računajo enako dobro kot z skladom, imamo svobodo v vrstnem redu operacij (pomembno za cevovodno procesiranje), registri omogočajo shranjevanje vmesnih rezultatov, več ina podatkov je v registrih in zato je zelo zmanjšan promet med CPE in gl. pomnilnikom (hitrejši dostop do registrov, poveča se hitrost).

42. Vrste naslovov.

Kjerkoli je shranjen kakšen podatek, kot npr. pomnilnik, trdi disk, register, se ta podatek nahaja na nekem naslovu, katerega ima ta enota (z naslavljanjem preko tega naslova dostopamo do podatka). Vrste naslovov:

- fizični naslovi - direktna lokacija na kateri je operand
- logični nasiovi - definiraga prevajalnikvčasu prevajanja, lokacijadoločena zodmikom od začetne lokacije
- navidezni naslovi - program razdeljen na bloke, naslov definiran z št. bloka in odmikom
- dejanski nasiovi - pri izvajanju programa so bioki shranjeni v pomnilniku na naslov kje se stvar v resnici nahaja
- relativni naslovi - odmik od vrednosti programskega števca
- indirektni naslovi - posredno dobimo naslov, pomniiniški in registerski (v registrih shranjen naslov nekega podatka)

43. ***Načini naslavljanja***.

- takojšnje (ADD R1, #5; R1=R1+5) - najpreprostejši način za določanje operanda dobimo, če je ta v ukazu podan kar z vrednostjo. V tem primeru je operand del ukaza in se prenese v CPE skupaj z njim - dodatni dostopi do pomnilnika niso potrebni. Takojšnji operandi so posebna vrsta pomnilniških operandov in se uporabljajo za definicijo in vnos konstant. V simbolični obliki se običajno označujejo z znakom #. Koristnost takojšnjih operandov je največja pri aritmetičnih ukazih, primerjavah (predvsem pri povezavi z pogojnimi skoki) in v ukazih za prenos podatkov.
- neposredno - kadar je v ukazu operand podan z naslovom, pravimo temu neposredno, direktno ali tudi absolutno naslavljanje. Čeprav nas registrski operandi tu ne zanimajo je ta način tipičen tudi zanje - takrat mu pravimo registrsko naslavljanje in v ukazu je operand podan z naslovom registra v CPE. Pri pomnilniških operandih pa je v ukazu pomnilniški naslov in pravimo, da ukaz vsebuje naslov lokacije v pomnilniškem prostoru. Ker je ta naslov del ukaza, se ne spreminja in ga zato imenujemo tudi absolutni naslov. Pri prevzemu ukaza pride ta naslov v *CPE,* ki med izvrševanjem ukaza naredi dostop do operanda na tem naslovu
    - registrsko - operandi spravljeni v registre (ADD R1, R2; R1=R1+R2)
    - pomnilniško - vrednost na pomnilniški lokaciji (ADD R1, (1000); R1=RI+M[1000]); page 0 (program se razdeli na bloke, rabimo manj časa in manj bitov)
    - vhodno-izhodno - prek registrov, podobno registerskemu

- posredno - naslov pomnilniškega operanda je v ukazu podan posredno preko neke druge vrednosti. Ta druga vrednost je lahko v glavnem pomnilniku, ali v enem od registrov. V prvem primeru govorimo o pomnilniškem posrednem naslavljanju, v drugem pa o registrskem posrednem naslavljanju. Pri pomnilniškem je v ukazu pomnilniški naslov lokacije, na katerem je shranjen pomnilniški naslov operanda. Pri registrskem pa je v ukazu naslov registra in tako imenovani odmik - iz vsebine registra in odmika se izračuna pomnilniški naslov operanda. Med vsemi načini naslavljanja obstaja daleč največje število inačic ravno pri registrskem posrednem naslavljanju. Za

večino računalnikov velja, da je ta vrsta naslavljanja tudi najpogostejši način za dostop do pomnilniških operandov. Ker je pri vseh načinih registrskega posrednega naslavljanja naslov operanda določen relativno na vsebino najmanj enega registra, mu pravimo tudi relativno naslavljanje.

- registersko - brez odmika *(ADD* R1, (R2); R1=R2+M[R2]); z odmikom (ADD R1, 100(R2); R1=R1+M[R2+100])
- pomnilniško (ADD R1, @(1000); R1=R1+M[M[1000]])
- inkrementalno - predinkrementalno (ADD R1, +(R2); R2=R2+1; R1=R1+M[R2]); poinkrementalno (ADD R1, (R2)+; R1=R1+M[R2]; R2=R2+1)
- dekrementalno - preddekrementalno (ADD R1,-(R2)); podekrementalno (ADD Rl, (R2)-)
- bazno - (ADD R1, 100(R2)) najpogostejša vrsta registrskega posrednega naslavljanja, ki se včasih označuje tudi kot naslavljanje z odmikom. Pri njem sta v ukazu podana naslov registra R1 in odmik D. Naslov operanda (A) je določen z izrazom A = R1 + D. Dobimo ga torej tako da k vsebini registra Rl prištejemo odmik D. Dolžina registra R1 je praviloma enaka ali daljša od dolžine pomnilniškega naslova. Registru R1 pravimo bazni register, izračunanemu naslovu A pa dejanski naslov. Ta pojem se uporablja za označevanje naslova, s katerim CPE zahteva dostop do operanda v glavnem pomnilniku
- indeksno - (ADD R1, (R2+R3)) poleg z odmikom D, ki obsega celoten pomnilniški naslovni prostor, lahko bazno naslavljanje spremenimo v indeksno tako, da v ukazu podamo dva registra. Dejanski naslov je določen z A = R1 + R2 + D = R1 + D1. Ker ima izračunani odmik D1 = R2 + D vedno dolžino, ki je enaka dolžini pomnilniškega naslova, je to naslavljanje indeksno pri vsaki dolžini odmika D. Glavno področje uporabe indeksnega naslavljanja je pri delu z operandi v obliki podatkovnih struktur kot so polja, zapisi in seznami. Posamezni elementi v teh strukturah se pogosto obdelujejo zaporedoma po naraščajočih ali padajočih indeksih, zato sta pri uporabi teh naslavljanj zelo pogosti naslednji dve operaciji R1 ← R1 + x, Rl ←R1 - x kjer je x dolžina operanda merjena v številu pomnilniških besed (1, 2, 4, 8 ali 16) in se označuje kot korak indeksiranja. Operaciji je seveda mogoče narediti z ukazoma za seštevanje in odštevanje. Nekateri računalniki imajo vgrajeno to možnost, ki ji pravimo avtomatsko indeksiranje. Spreminjanje indeksa se lahko naredi pred računanjem dejanskega naslova ali po njem. Če upoštevamo, da je spreminjanje lahko navzgor ali navzdol, imamo skupaj štiri možnosti. Od teh štirih se običajno uporabljate dve komplementarne možnosti: 1) zmanjševanje indeksa pred računanjem in 2) povečanje indeksa po računanju
- preddekrementno - (ADD R1, -(R2)) pri tem načinu je dejanski naslov A določen z R1 ← R1 - x, A = R1 + D ali A = R1 + R2 + D. To naslavljanje je lahko bazno ali indeksno in je, če izvzamemo avtomatsko zmanjševanje indeksa, enako indeksnemu ali baznemu naslavljanju. Avtomatsko indeksiranje naredi to, kar moramo na računalnikih brez njega narediti z dodatnim ukazom za spreminjanje baznega oz. indeksnega registra. Glede števila potrebnih operacij med obema načinoma ni razlike. Pač pa pri naslavljanju z avtomatskim indeksiranjem odpade potreba po prevzemu dodatnega ukaza. Na drugi strani so na računalnikih, ki omogočajo veliko načinov naslavljanja, ukazi daljši
- poinkrementno - (ADD R1, (R2)+) pri tem načinu je dejanski naslov A določen z A = R1 + D ali A = R1 + R2 + D, R1 ← R1 + x. To naslavljanje tvori skupaj *s* preddekrementnim par, s katerim je mogoče, poleg drugega, na zelo preprost način narediti v gl. pomnilniku sklad. Z enim dajemo operande v sklad, z drugim pa jih iz njega jemljemo. To omogoča dostop do elementov polja na način LIFO (zadnji noter, prvi ven), ne da bi bilo treba s posameznimi ukazi spreminjati vsebino registra

- velikostno indeksno - (ADD R1, 100(R2*x+R3)) ta način je nekoliko drugačna inačica avtomatskega indeksiranja. dejanski naslov *A* je določen z A R1 * x + R2 + D. Tudi tu je korak indeksiranja x enak dolžini operanda, le da se množi z indeksnim registrom R1. Za indeksiranje sedaj zadošča povečevanje (ali zmanjševanje) indeksnega registra za 1, ki pa ni vključeno v samo naslavljanje.

44. Vrste neposrednega naslavljanja.

- registrsko - operandi spravljeni v registre (ADD R1, R2; R1=R1+R2)
- pomnilniško - vrednost na pomnilniški lokaciji (ADD R1, (1000); R1=RI+M[1000]); page 0 (program se razdeli na bloke, rabimo manj časa in manj bitov)
- vhodno-izhodno - prek registrov, podobno registerskemu

45. Vrste posrednega naslavljanja.

- registersko
- pomnilniško (ADD R1, @(1000); R1=R1+M[M[1000]])
- inkrementalno - predinkrementalno (ADD R1, +(R2); R2=R2+1; R1=R1+M[R2]); poinkrementalno (ADD R1, (R2)+; R1=R1+M[R2]; R2=R2+1)
- dekrementalno - preddekrementalno (ADD R1,-(R2)); podekrementalno (ADD Rl, (R2)-)
- bazno - (ADD R1, 100(R2))
- indeksno - (ADD R1, (R2+R3))
- preddekrementno - (ADD R1, -(R2))
- poinkrementno - (ADD R1, (R2)+)
- velikostno indeksno - (ADD R1, 100(R2*x+R3)).

46. Od česa je odvisna velikost ukaza?

47. Delitev računalnikov glede na število eksplicitnih naslovov.

Delitev računalnikov glede na število eksplicitnih naslovov:

1. 3+1-operandni računalniki - takih računalnikov danes ni več *(EDVAC)*. Oznaka +1 pomeni, da je poleg treh eksplicitnih operandov v ukazu kot operand tudi naslov naslednjega ukaza. Ti računalniki niso imeli pomnilnika za shranjevanje operandov v CPE (operandi podani z pomnilniškimi naslovi lokacij, v katerih so shranjeni). Iz enačbe OP3←0P2 + OP1; PC← OP4 (OP - operandi, + je operacija, ne nujno seštevanje) sledi, da določa četrti (ali +1) operand naslov naslednjega ukaza. Računalniki te vrste v resnici niso imeli programskega števca. Osnovni razlog za pojav 3+1-naslovnih računalnikov je v tem, da so se v njihovem času (~1950) za glavni pomnilnik uporabljale naprave s krožnim dostopom (magnetni boben, zakasnilna linija). Pri teh napravah informacija stalno kroži in *čas* za dostop do neke lokacije je odvisen od njenega trenutnega položaja glede na dostopni mehanizem. V najboljšem primeru je informacija dostopna takoj, v najslabšem pa je potreben en celi obrat. S pametno razporeditvijo ukazov in operandov je mogoče zelo zmanjšati čakanje zaradi krožne narave dostopa.
2. 3-operandni računalniki - *s* pojavom prvih pomnilnikov z naključnim dostopom je odpadla potreba po pametnem razmeščanju ukazov po pomnilniku in eksplicitnem podajanju naslova naslednjega ukaza. Pri pomnilnikih z naključnim dostopom je čas za dostop neodvisen od zaporedja naslovov (čas za dostop je enak pri naključnem zaporedju naslovov). Pravilo PC←PC + 1 določa implicitni vrstni red ukazov. Ddelovanje 3-operandnega računalnika lahko opišemo z OP3 ← OP2 + OP1; PC ← PC + 1, kjer so vsi trije operandi največkrat v registrih CPE (operandi v gl. pomnilniku se uporabljajo redkeje). Ker imajo osnovne aritmetične in logične operacije tri operande, so ukazi *s* tremi eksplicitnimi operandi najbliže navadam v matematiki. Dbo 1980-tih let so jih imeli predvsem bolj zmogljivi računalniki, ki so namenjeni za računsko

intenzivne operacije. Praktično vsi superračunalniki so bili (in so še) 3-operandni. Med računalnikih razvitih po letu 1980 je največ 3-operandnih, praktično vedno z omejitvijo na operande v registrih (load/store).

3. 2-operandni računalniki - dolgo so bili najbolj pogosta vrsta. Osnovni razlog za to je spoznanje da opustitev enega eksplicitnega operanda pogosto ne vpliva veliko na hitrost računanja (rezultat ene operacije se takoj porabi kot vhodni operand za naslednjo, nato pa ga shranimo v prostor, na katerem je eden od vhodnih operandov). Z opustitvijo enega operanda dobimo 2-operandni računalnik, ki je opisan z OP2← 0P2 + OP1; PC ← PC + 1. Občasno se tudi zgodi, da pri kasnejših operacijah potrebujemo oba vhodna operanda. V tem primeru je treba z dodatnim ukazom enega od operandov najprej shraniti na drugo mesto (drug register). 2-operandni računalnik je v nekaterih primerih počasnejši od 3-operandnega. Operanda OP1 in OP2 sta lahko v enem od registrov v CPE ali v gl. pomnilniku. Obstaja veliko 2-operandnih računalnikov, kjer sta oba operanda v pomnilniku. Še pogosteje pa je eden vedno v registru (1 1/2-naslovni računalniki, kjer je z 1/2 označen naslov registra). 2-operandni računalniki so bili do leta 1980 najpogostejša vrsta.
4. 1-operandni računalniki - računalniki, ki imajo v CPE za shranjevanje operandov en sam (včasih dva) akumulator. Eden od operandov se vedno nahaja v akumulatorju, zato zadošča en eksplicitni operand. Delovanje se lahko simbolično opiše z *AC* ← *AC* + *AC; PC*←PC + 1, kjer je AC akumulator. Čisti 1-operandni računalniki, ki ne bi poleg akumulatorja imeli se kakšnega registra se niso nikoli gradili. Ti drugi registri so namenjeni za posebne namene in se ne morejo uporabljati pri večini aritmetično-logičnih operacij. Zaradi tehnoloških omejitev so bili "nečisti" 1-operandni računainiki skoraj vsi mikroprocesorji iz 1970-tih let. 1-operandni računalniki so zelo razširjeni in se uporabljajo še danes (gospodinjski stroji, avtomobili, igrače). Čisti 1-naslovni računalnik se kot teoretični model von Neumannovega računalnika včasih uporablja pri študiju izračunljivosti.
5. brez-operandni (skladovni) računalniki - računalniki ki imajo pomnilnik narejen v obliki sklada. Delovanje lahko opišemo z Sklad(vrh) ←Sklad(vrh) + Sklad(vrh-1); PC ← PC + 1. Operacije se izvajajo med operandi na vrhu sklada in operandov v ukazu ni treba navajati eksplicitno. Ukazi so zato krajši kot pri drugih vrstah računalnikov. Pri vsakem potrebujemo najmanj dva ukaza z enim eksplicitnim operandom. Ukaza PUSH in POP, omogočata prenos operanda iz pomnilnika na sklad in obratno. Pri večini skladovnih računalnikov imamo v CPE poleg sklada še nekaj registrov, ki olajšujejo dostop do operandov v pomnilniku. Noben brez-operandni računalnik ni popolnoma brez ukazov z eksplicitnimi operandi. Med računalnikih, ki so bili razviti po letu 1980, skladovnih ni več.

48. Ukazi (ddiitev po tipu, po namenu, po številu operandov)

Delitev ukazov po tipu:

- ukaz za prenos podatkov
- aritmetični ukazi (increment, decrement, negacija, absolutna vrednost, vektorski ukaz)
- logični ukazi (delo z biti, boolove operacije)
- ukaz za nadzor nad programom (primerjave, pogojni skoki, nadzor nad zankami, ukazi za klic procedur)
- ukazi za nadzor nad delovanjem računalnika (privilegirani, namenjeni sistemskemu programiranju)
- V/I ukazi

Delitev ukazov po namenu:

- operacijski ukazi (aritmetične operacije, logične operacije, delo z nizi, nop, pomnilni ukazi)
- ukazi za dostop do pomnilnika (load/store ukazi, ukazi za sinhronizacijo procesov, podatkovna sinhronizacija, aromatni ukazi (ni prekinitve, izvesti se mora od začetka do konca, sicer se sploh ne izvede))
- test and set - binarni semafor (0, 1)
- clear - odklepanje
- match and replace *(C* - primerjalna vrednost, R - vrednost ki jo bomo primerjali, 5 -binarni semafor *(* npr. if(C==R) S=R))

- inkrementirajoči semafor (fetch and increment, fetch and decrement, fetch and add)

Delitev ukazov po številu operandov:

- 3+1-operandni (zadnji operand določa naslov naslednjega ukaza)
- 3-operandni
- 2-operandni
- 1-operandni
- brez-operandni (skladovni)

49. Vrste ukazov.

- splošno namenski
- specifično namenski
- ortogonalni
- operacijski - aritmetično-logični, pomikalni, kontrolni, za delo *s* skladom in registri, NOP (no operation)
- za prenos podatkov
- za sinhronizacijo procesov
- za spreminjanje poteka izvajanja programa
- vhodno/izhodni
- za delo s prekinitvami in pastmi
- privilegirani ukazi
- vektorski ukazi - operacijski, vstavljanje in shranjevanje vektorjev, selekcija elementov vektorjev, združevanje dveh vektorjev, kompresija in ekspanzija vektorjev.

50. Sinhronizacijski ukazi.

Sinhronizacijski ukazi skrbijo za medsebojno izključevanje dostopov do vira (RAM, printer), lahko pa tudi skrbijo za komunikacijo med procesi ter za sprožanje in končanje izvajanja procesov. Semaforji so binarni (samo en tip) in inkrementalni (omejeni so na celo število, ne more biti manjše od 0 in ne večje od maksimalnega). Lastnosti ukazov za sinhronizacijo procesov so, da jih ni mogoče prekiniti in drugi proces, ne more doseči operandov dokler se ukaz izvaja.
test and set - testira vrednost in če je enaka 0 jo postavi na 1
clear - vrednost binarnega semaforja postavi na 1
match and replace - primerja vrednosti in jih zamenja (C-primerjalna vrednost, R- vrednost ki jo bomo primerjali in vpisali v S če je *C=R, S*-binarni semafor)
fetch and increment - vrednost inkrementalnega semaforja poveča za 1
fetch and add - vrednost inkremntalnega semaforja lahko poveča za večkot 1.

51. Razložite pojme CISC, RISC in superskalarne arhitekture.

- CISC (Compleks Instruction Set Computer) računalnik z velikim št ukazov. Pri teh računalnikih se nekateri ukazi izvršijo hitro drugi pa precej počasneje, uporablja se cca 15% ukazov ostali pa zelo redko. Ukazi so različno dolgi (16, 32, 64, 128 bit) itd.
- RISC (Reduced Instruction Set Computer) računalnik z majhnim št ukazov. Večina ukazov se izvrši v eni urini periodi CPE, je registrsko-registrska zasnova (load-store). Ukazi so realizirani s trdo ožičeno logiko in ne mikoprogramsko, vsebujejo malo ukazov in naslavljanj (dekodiranje in izvrševanje ukaza je zelo hitro), vsi ukazi so iste dolžine, imajo dobre prevajalnike.
- Super skalarni računalniki. So računalniki, ki opravijo v eni urini periodi več kot en skalarni ukaz. CPE prevzame v vsaki urini periodi 2-4 skalarne ukaze, preveri ali so med seboj neodvisni in izvrši

naenkrat samo tiste, ki so neodvisni (vsakega v svojem cevovodu), ostali morajo čakati. Pogoje za paralelizem preverja CPE in ne prevajalnik kar je njihova osnovna značilnost.

52. Napišite formule s katerimi lahko analiziramo strojne programe.

53. Razlike med CISC in RISC

CISC

- računalniki z velikim številom ukazov
- ukazi so realizirani mikroprogramsko
- več operacij, naslavljanj ter novi ukazi

RISC

- računalniki z malim številom ukazov
- registrsko registrska(load/store) zasnova
- malo ukazov in načinov naslavljanja
- vsi ukazi imajo isto dolžino

54. Vodila

Vodilo je skupek žic po kateri se pretakajo naslovi, podatki, ukazi itd. skrbi za prenos podatkov med različnimi deli računalnika. Lahko so statična (obroč, mreža) ali dinamična (v času izvajanja lahko vzpostavljamo različne povezave med različnimi enotami (omega mreža).

Delitev (struktura) :

- lokalna znotraj CPE,
- sistemska (povezovalne enote znotraj računalnika ISA; PCI) CPE pomnilnik,
- razširjena lokalna
- hitre enote (SCSI, grafična kartica)
- počasne enote (fax, modem),
- V/I vodila za povezovanje.

Linije vodila:

- podatkovna (prenašajo podatke in ukaze, velikost št podatkovnih linij vpliva na hitrost računalnika (8, 16, 32, 64)),
- naslovna (definirajo izvor in ponor prenosa podatkov po podatkovnih linijah, velikost naslovnih linij definira največjo kapaciteto pomnilnika),
- kontrolna (zahteva po vodilu, delitev vodila, naslov je na voljo, podatki so na voljo, potrditev prenosa, prekinitvene linije, kontrola parniosti, izbira naprave, ura. Določajo smer in število prenesenih besed).

Tipi vodil:

- namenska (ločene podatkovne in naslovne linije),
- multipleksirana (po isti liniji se prenašajo naslovi in podatki dodatne kontrolne linije, ki povejo, ali se prenašajo podatki ali naslovi. Prednost manj linij, slabost kompleksnejši nadzor, zmanjšanje zmogljivosti).

55. Dodeljevanje vodila (arbitracija)

Po prenosni poti lahko poteka naenkrat samo en prenos. Prenos zahteva in vodi gospodar. Gospodar generira naslovne in kontrolne signale. Gospodar je lahko v nekem trenutku samo eden. Včasih želi dostopati do vodila več enot naenkrat, zato se uporablja mehanizem, s pomočjo katerega se gospodarji dogovorijo, kdo bo imel nadzor nad vodilom ARBITRAŽA. Poznamo CENTRALIZIRANO in DECENTRALIZIRANO arbitracijo vodila. CENTRALIZIRANA: - enonivojska ena naprava kontrolira dostop do vodila (krmilnik vodila, arbiter). Ta naprava je lahko del CPE ali pa je samostojna naprava. Enota pošlje zahtevo po vodilu, arbiter pošlje zagotovitev vodila, - dvonivojska arbiter najprej obravnava zahteve na prvem nivoju, če ni zahteve gre na drugi nivo itd.

DECENTRALIZIRANA: - WAX enota postavi zahtevo po vodilu, če ni zasedene linije. Ko vspostavi zahtevo po vodilu postavi zasedeno linijo, ko konča postavi zasedeno linijo na 0.

56. Sinhrona in asinhrona vodila.

Glede na način določanja začetka in konca prenosa razlikujemo dve osnovni vrsti prenosa: sinhroni in asinhroni prenos.

- Sinhroni: čas prenosa T je vedno enak, vrednost T je vedno enaka celemu številu urinih period gospodarja. Gospodar na začetku generira naslovne signale, za tem se vzpostavijo kontrolni signali, ki predstavljajo smer in število prenesenih besed (branje). Pri pisanju se med generiranjem prejšnjih dveh signalov generirajo še podatkovni signali. Po preteku časa T se kontrolni in naslovni signali umaknejo ali zamenjajo z novimi in do takrat morajo biti vse operacije zaključene. Če imamo na vodilo priključenih več naprav mora biti T izbran tako da ustreza najpočasnejši (vse enote delujejo z hitrostjo najpočasnejše) - slabost sinhronskega prenosa.
- Asinhroni: čas prenosa ni vnaprej določen. Začetek prenosa je enak kot pri sinhronem, konec pa drugačen. Gospodar ne umakne ali spremeni naslovnih ali kontrolnih signalov po preteku časa T ampak šele ko dobi od naslovljene enote potrditveni signal. Če enota ne obstaja (napaka v programu) se potrditveni signal ne bo aktiviral, zato imajo asinhrona vodila posebno časovno vezje, ki zaključi predolg prenos in javi napako.

57. Vodila, ki jih uporabljajo osebni računalniki, ki temeljijo na Intel X86 in Pentium računalniški arhitekturi.

- PCI Peripheral Component Interconnection
- ISA Industry Standard Architecture
- USB Universal Serial Bus
- SCSI Small Computer Systems Interface
- AGP Accelerated Graphics Port

58. Hierarhija pomnilnikov (zakaj pomnilnik razbili v hierarhijo)? Opišite povezavo med ceno/bit, časom dostopa, velikostjo pomnilnika in verjetnostjo zadetka).

Osnovni cilj pomnilniške hierarhije je doseči, da je velik počasen in cenen pomnilnik videti kot majhen, drag in hiter pomnilnik. Čas dostopa do podatka je odvisen od tega, na katerem nivoju pomnilniške hierarhije se nahaja želena beseda. Potrebna je logika za prehajanje in zamenjavanje vsebine po hierarhiji navzgor ali navzdol. Za CPE je pomnilniška hierarhija videti kot glavni pomnilnik. Če informacija do katere želi dostopiti CPE ni v M1, se ta prenese iz M2 v M1, če je ni v M2 se prenese iz M3 v M2, itd. Uspešnost te rešitve je posledica lokalnosti. Imamo množico pomnilnikov {M1, M2,…Mn}. Za to množico velja: cena/bit (Ci<Ci+1), čas dostopa (tai<tai+1), velikost (Si<Si+1).

-registri (najhitrejši, najdaržji)
-predpomnilnik (cache)
-RAM
-HD (počasen, poceni)

Pomnilnik razbijemo v hierarhijo zaradi hitrejšega delovanja – optimizacije izvajanja ukazov – programov.
Verjetnost zadetka je največja (100%) pri registru in se spreminja (zmanjšuje) enako kot hierarhija pomnilnika do HD (najmanjša verjetnost zadetka), prav tako se hierarhično zmanjšuje hitrost dostopa (reg najhitrejši, HD najpočasnejši) in cena/bit (1 reg najdražje, 1 bit HD najcenejše – večji je pomnilnik, dražji je).

59. Lastnosti pomnilniške hierarhije?

60. Vrste lokalnosti.

- prostorska lokalnost - pojav, da je po pojavu naslova *A(i)* verjetno, da bodo naslednji naslovi, npr. naslov *A(i-i-n),* blizu naslova A(i) - drugače povedano, verjetno je, da bo absolutna vrednost razlike A(i+n) - *A(i)* majhna. Glavni razlog za prostorsko lokalnost je, da von Neumannov računalnik, če ni skoka, jemlje ukaze enega za drugim po naraščajočih naslovih. Drugi razlog je, da v programih pogosto nastopajo podatkovne strukture (polja), ki se največkrat uporabljajo zaporedoma po naraščajočih ali padajočih indeksih. Iz teh dveh razlogov prostorsko lokalnost pogosto označujemo kot zaporedno lokalnost. Tretji razlog je razdelitev programa na procedure ali podprograme. Med izvajanjem ukazov neke procedure se večina dostopov nanaša na pomnilniški prostor, ki je dodeljen proceduri, ker so procedure pogosto kratke je ta prostor majhen.
- časovna lokalnost - pojav, da program ob času t pogosto tvori naslove, ki jih je tvoril malo pred t, in ki jih bo tvoril tudi nekoliko po t. Vzroka za ta pojav sta predvsem v lastnosti programov, da uporabljajo zanke in začasne spremenljivke. Zaradi zank se isto zaporedje ukazov, in s tem isti naslovi, velikokrat ponovi. Zaradi začasnih spremenljivk se naslovi nekaterih operandov pojavijo večkrat.

61. Verjetnost zadetka in verjetnost zgrešitve v hierarhično organizirani pomnilniški strukturi.

Verjetnost zadetka pove kakšna je verjetnost, da bomo podatke našli na hierarhičnem nivoju. Če so bloki veliki začne verjetnost padati, ker se izvrši veliko menjav. Uspešnost delovanja pomnilnika merimo z verjetnostjo zadetka in jo označimo s črko H. Kadar je naslov do katerega želi dostopiti CPE, v pomnilniku imamo zadetek, če ga ni imamo zgrešitev oziroma imamo verjetnost zgrešitve. Ker imamo pri vsakem dostopu zadetek ali zgrešitev je verjetnost zgrešitve enaka 1-H. Verjetnost H lahko izmerimo tako da štejemo koliko pomnilniških dostopov se nanaša na naslove, ki so v predpomnilniku. Če je pri Np od skupno N dostopov informacija v predpomnilniku, je verjetnost zadetka enaka: H=Np/N.

62. Vrste dostopov do pomnilniške strukture.

- Naključni dostop -čas dostopa do poljubne besede je neodvisen od naslova predtem naslovljenih besed. Čas dostopa t je konstanten in znan vnaprej pri naključnem zaporedju naslovov. Konstanten čas dostopa, ne glede na fizični položaj naslovljene besede in ne glede na prejšnje naslove, je glavna lastnost naključnega dostopa. DRAM elementi, s katerimi so danes narejeni glavni pomnilniki, so naprave z naključnim dostopom.
- Zaporedni dostop - čas za dostop do neke besede je odvisen od naslova besede, do katere je bil narejen dostop tik pred tem. Če je bil npr. *A* naslov prejšnje besede, je takoj dostopno samo beseda z naslovoma *A* + 1. Za dostop do poljubnega naslova B pa moramo najprej izvršiti dostop do vseh besed med *A* in B. Očitno je pri tem načinu čas dostopa t močno odvisen od zaporedja naslovov. Tipična predstavnika pomnilnikov z zaporednim dostopom sta magnetni trak in pomikalni register.
- Krožni dostop - posebna vrsto zaporednega dostopa. Predstavljamo si ga lahko kot magnetni trak, ki je zlepljen v zanko. Srečamo ga pri magnetnih diskih s fiksnimi glavami, magnetnih bobnih, zakasnilnih linijah in magnetnih mehurčkih. Povprečen čas dostopa t je pri krožnem načinu enak 1/2 periode vrtenja. Pri diskih in bobnih je vrtenje mehansko, pri zakasnilnih linijah in magnetnih mehurčkih pa je elektronsko, brez gibajočih delov.
- Direknti dostop - pri tem načinu gre za kombinacijo zaporednega in krožnega načina dostopa. Direktni dostop srečamo pri magnetnih in optičnih diskih s premičnimi glavami. Deluje tako, da se bralno-pisalna glava najprej pomakne nad ustrezno sled (zaporedni dostop), nato pa imamo že opisani krožni način dostopa. Ker poteka premik glave preko sledi hitro, je pri enaki velikosti pomnilnika povprečen čas dostopa veliko krajši kot pri zaporednem ali krožnem dostopu.

63. Razložite pojme »pisanje skozi«, »pisanje nazaj«,»LRU«, »LFU«

- Pisanje skozi - strategija za pisanje v predpomnilnik. Pri tem načinu se vedno piše v predpomnilnik in v glavni pomnilnik. Vsebina bloka v predpomnilniku in v glavnem pomnilniku je vedno enaka. Prednost pisanja skozi je da je enostavno za realizacijo, čeprav zahteva uporabo pisalnega izravnalnika. Vsebina bloka v predpomnilniku je vedno enaka tisti v glavnem pomnilniku. Pravimo da sta vsebini skladni ali koherantni.

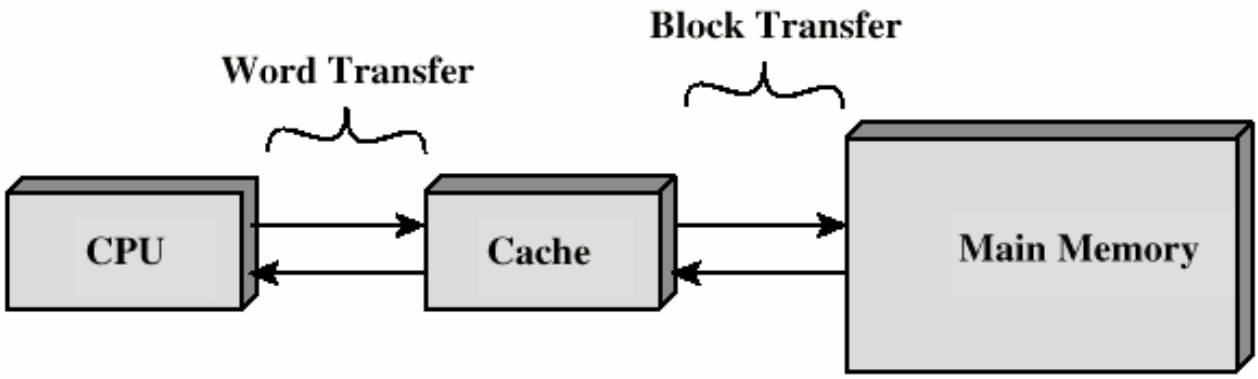
- Pisanje nazaj - pri tem načinu piše se samo v predpomnilnik. To pomeni da je vsebina bloka v predpomnilniku lahko drugačna od vsebina v glavnem pomnilniku. Pri zamenjavi je spremenjeni blok zato potrebno prenesti nazaj v glavni pomnilnik. Ti predpomnilniki so običajno narejeni tako, da imajo v kontrolni informaciji vsakega bloka poseben bit, ki mu pravimo umazani bit. Ta bit se ob prenosu bloka v predpomnilnik postavi na 0. Če pride do pisanja v blok, se umazani bit postavi na 1. To omogoča razlikovati med bloki, ki so se v predpomnilniku spremenili in tistim, ki se niso. Pri zamenjavi bloka se nazaj v glavni pomnilnik zapišejo samo tisti bloki, ki imajo umazani bit enak 1. Prednost pri pisanju nazaj je da pisanje poteka s hitrostjo ki jo ima predpomnilnik, pri več pisanjih v isti blok je potrebno samo eno pisanje v glavni pomnilnik, ko se ob zamenjavi blok piše nazaj v glavni pomnilnik se lahko izkoristi največja hitrost glavnega pomnilnika ker se prenese ves blok.
- LRU (Least Recently Used) - zamenjevalna strategijo za določanje blokov. Pri tem načinu se zamenja blok do katerega najbolj dolgo ni bil narejen dostop. Ta strategija je boljša od naključne, ker zmanjšuje nevarnost za zamenjavo bloka, ki se je uporabil pred kratkim. Tu gre za izkoriščanje časovne lokalnosti, po kateri je bolj verjetno, da se dalj časa neuporabljen blok ne bo več uporabil.
- LFU (Least Frequently Used) - zamenja blok, ki ima najmanj dostopov.

64. Strategije zamenjave in nameščanja.

Zamenjevalna strategija določa blok kateri se naj zamenja z novim. Pri predpomnilniku z direktno preslikavo poznamo 4 načine:

- RAND (naključno izbere blok)
- FIFO (first in first out)
- LRU (zamenja blok, ki najdalj ni bil uporabljen)
- LFU (zamenja blok, ki ima najmanj dostopov)

65. Kako delujejo predpomnilniki.



Predpomnilnik je majhen in hiter pomnilnik. Nahaja se med CPE in glavnim pomnilnikom. V nekaterih primerih se uporablja tudi med glavnim pomnilnikom in V/I napravami. Pri pravilni uporabi lahko dosežemo bistveno povečanje hitrosti dostopa, ker je hitrost dostopa do predpomnilnika 5 do 10 krat večja od hitrosti glavnega pomnilnika. To je možno samo če se bo informacija pri večini dostopov nahajala v predpomnilniku. Uspešnost delovanja predpomnilnika merimo z verjetnostjo zadetka. Če se zahtevani naslov nahaja v predpomnilniku, imamo zadetek - če ga ni, pa zgrešitev. Pri dovolj velikih in kvalitetnih predpomnilnikih je verjetnost zadetka večja od 0,9. Visoka verjetnost zadetka je zelo pomembna za učinkovitost predpomnilnika. Za vse predpomnilnike praviloma velja da so nevidni za programerja. Pri večini današnjih računalnikov je predpomnilnik vgrajen v CPE. Lahko je tudi zgrajen kot posebna enota ki jo je mogoče vzeti iz sistema. Predpomnilnik mora vsebovati, skupaj z informacijo, tudi naslove ki povedo katerim besedam glavnega pomnilnika ustreza njegova trenutna vsebina. Predpomnilnik je sestavljen iz dveh delov:

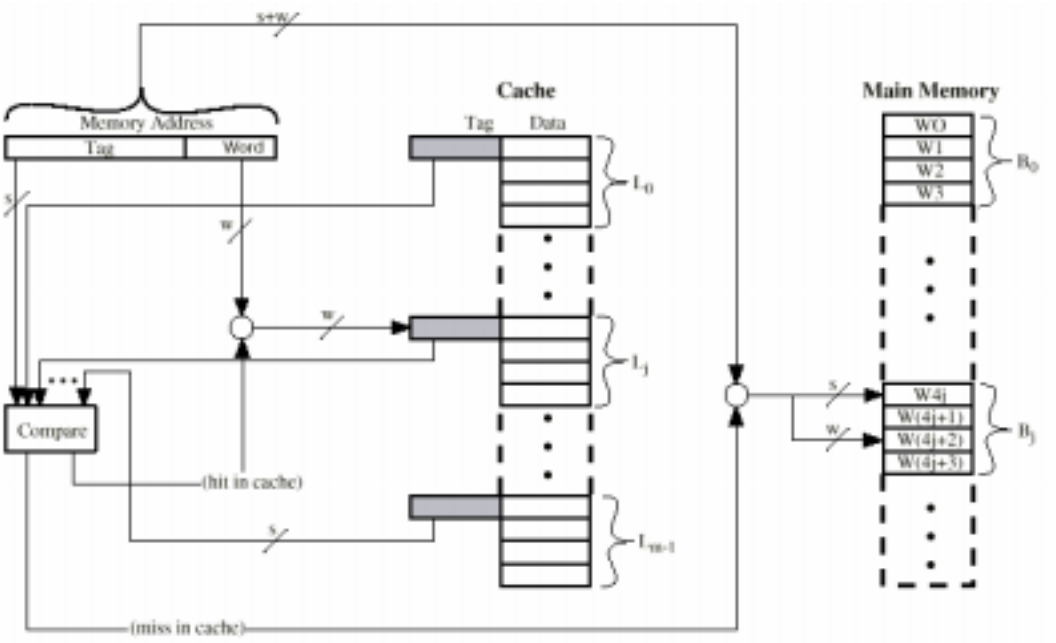
- kontrolni del
- pomnilniški del

Pomnilniški del je razdeljen v enote enake velikosti - bloki. Kontrolni del vsebuje kontrolno informacijo ki identificira vsak blok. Kontrolna informacija vsebuje trenutni pomnilniški naslov bloka, ter še nekatere dodatne bite (umazani bit, veljavni bit...). Med delovanjem CPE generira zaporedje naslovov ki gredo naprej v predpomnilnik. Predpomnilnik primerja zgornjih n-bitov naslova z naslovi v kontrolni informaciji vseh blokov. Če pri nekem bloku imamo enakost in je istočasno veljavnostni bit 1, imamo zadetek in dostop se izvrši do vsebine v pomnilniškem delu bloka. Če zadetka ni je potreben dostop do glavnega pomnilnika ter polnjenje predpomnilnika. Glede na vrsto preslikave pomnilniških naslovov v bloke, danes poznamo 3 vrste predpomnilnika:

- asociativne
- set asociativne
- direktne.

66. Vrste predpomnilnikov.

- asociativni: (vsak blok glavnega pomnilnika se lahko preslika v katerikoli blok predpomnilnika, najboljši možen izkoristek, slabost je visoka cena).

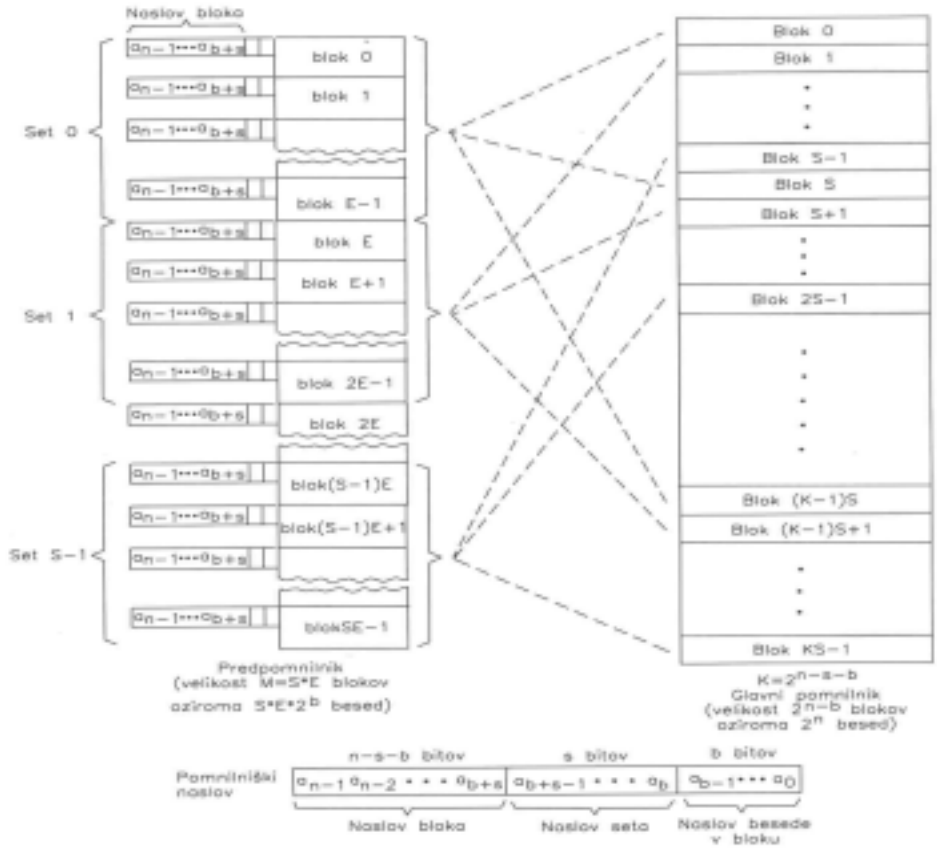


- set asociativni: (predpomnilnik razdeljen na več setov, ti pa na več blokov, set je majhen asociativni pomnilnik, nek blok glavnega pomnilnika se lahko preslika v točno določeni set predpomnilnika, potem pa v katerikoli blok v setu, optimalna velikost seta je med 2 in 16 blokov).
- direktni: (vsak blok glavnega pomnilnika se preslika v točno določen blok predpomnilnika, najcenejši najpreprostejši).

67. Direktni predpomnilnik

je posebna vrsta set asociativnega predpomnilnika, v primeru ko je velikost seta 1. V tem primeru je v vsakem samo en blok in asociativni pomnilnik postane trivialen primerjalnik naslovov. Slabost direktnega pomnilnika je v tem, da se vsaki blok glavnega pomnilnika preslika v točno določen (vedno isti) blok predpomnilnika. V nasprotju s set asociativnim predpomnilnikom, kjer se vsak naslov lahko preslika v poljubnega od blokov v setu, ni pri direktnem predpomnilniku nobene izbire. Ker se v delovni množici nahaja več naslovov, ki se preslikajo v isti blok, je razumljivo da bo tu verjetnost zadetka manjša. Verjetnost zadetka je pri direktnih predpomnilnikih med 0,7 in 0,8, odvisno od velikosti predpomnilnika. direktni predpomnilnik se uporablja danes predvsem pri manjših računalnikih.

68. Set asociativni predpomnilnik



Set asociativni predpomnilnik se uporablja veliko pogosteje. Pri njemu je predpomnilnik razdeljen na 2^s setov, vsak set pa je majhen čisti asociativni predpomnilnik. Vsak set ima 2^e blokov, tako da je skupna vrednost 2^s+e blokov (besed). Osnovna razlika v primerjavi z čistim asociativnim predpomnilnikom je da sedaj obstaja omejitev pri preslikovanju pomnilniških naslovov v predpomnilnik. Za vse bloke glavnega pomnilnika velja da se lahko preslikajo samo v določeni set predpomnilnika. Vsi naslovi, ki pri deljenju z modulom 2^s+b dajo isti ostanek, se preslikajo v isti set. Zaradi te omejitve se pri set asociativnem predpomnilniku dogaja da je potrebno nek naslov preslikati v set, ki ima vse bloke zasedene, v ostalih setih so še pa prosti bloki. Set asociativni predpomnilnik je očitno slabši od čistega asociativnega. Cena set asociativnega predpomnilnika je veliko nižja od cene enako velikega čistega asociativnega, in zato so tudi danes set asociativni predpomnilniki najpogosteje uporabljana vrsta predpomnilnikov.

69. Asociativni predpomnilnik

Asociativni predpomnilnik predstavlja najboljšo rešitev problema. Uporabljen je za shranjevanje kontrolne informacije, iz zgornjih n-bitov pomnilniškega naslova takoj ugotovi, če je dani naslov trenutno v predpomnilniku. Vse lokacije asociativnega pomnilnika so ekvivalentne, vsak blok predpomnilnika lahko sprejme kateregakoli od blokov glavnega pomnilnika. To omogoča najboljši možen izkoristek predpomnilnika ne glede ne zaporedje naslovov. V praksi se čisti asocativni pomnilniki malo uporabljajo, zaradi zapletenosti in visoke cene. Danes se čisti asociativni predpomnilniki uporabljajo izključno pri majhnih predpomnilnikih za posebne namene (npr.preslikovanje navideznih naslovov v fizične, na računalnikih z navideznim pomnilnikom).

70. Zgradba diskov.

Disk se sestoji iz bralno/pisalne glave in pomnilniškega magnetnega medija, ki je nanesen na nemagnetni nosilec. Bralno/pisalna glava je narejena iz visoko permeabilnega mehko magnetnega materiala običajno toroidne oblike, ki ima tanko zračno rezo. Informacije so na disku shranjene v koncentričnih sledeh (tracks), ki jih je odvisno od modela od nekaj 10 pa do nekaj 1000 na eni strani diska. Sledem, ki jih hkrati dosežemo z bralno/pisalnimi

glavami, pravimo cilinder. Vsaka sled je razdeljena na sektorje (med 8 in 25 na sled). Vsi sektorji imajo enako kapaciteto, zato se gostota zapisa povečuje proti središču. Sektorji so nadalje razdeljeni v skupine/grozde. Med sektorji so tako imenovane vrzeli. Vrzeli so potrebne zato, ker zaradi nenatančnosti vrtenja ni mogoče zagotoviti, da se bo nek sektor zapisal natanko na isto mesto. V principu disk ni zapletena naprava. Sestoji se iz ravnih, krožnih plošč - diskov (nameščenih v stolpič) iz aluminij, lahkega nemagnetnega materiala. Premer diskov je običajno 3,5 ali 5,25 palcev, najdemo pa tudi manjše, 2,5 palčne, primerne za prenosne računalnike, pa tudi takšne s premerom 30 ali več za večje računalniške sisteme.

71. Razložite pojme, premetavanje, »zunanja« in »notranja« fragmentacija.

72. Ostranjevanje.

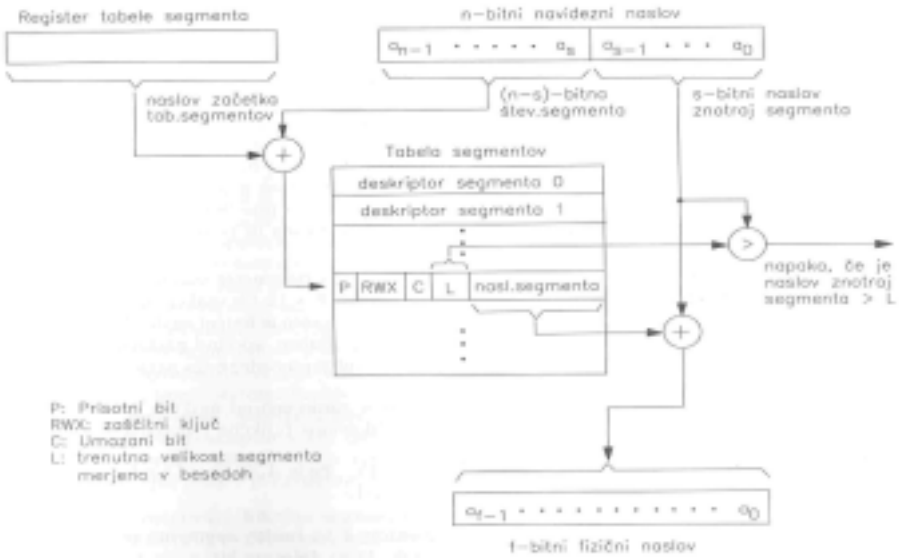
Ostranjevanje - najstarejša vrsta navideznega pomnilnika. Pomožni pomnilnik je razdeljen na bloke enake velikosti, ki jim pravimo strani. Vse strani skupaj sestavljajo navidezni pomnilnik. Na enako velike bloke, ki im pravimo okviri strani, je razdeljen tudi glavni pomnilnik. Vsako stran navideznega pomnilnika je mogoče prenesti v poljubnega od okvirov. Pri dani velikosti pomožnega in glavnega pomnilnika sta največje število strani in število okvirov fiksna in znana naprej. Velikost strani je vedno potenca števila 2. Preslikovalna funkcija je pri odstranjevanju definirana s pomočjo posebne tabele, ki se imenuje tabela strani. Vsaki strani navideznega pomnilnika pripada v tabeli strani eno polje. Ker je število strani fiksno, je vnaprej določena tudi največja velikost tabele strani. Nekateri računalniki imajo za preslikovanje več tabel strani. Vsak program zaseda določeno število strani. Ker je velikost programa samo izjemoma enaka mnogokratniku velikosti strani, bo v povprečju zadnja stran izkoriščena samo polovično. V povprečju je pri vsakem programu polovica ene strani neizkoriščena. Tej neizkoriščenosti pravimo notranja fregmentacija. Notranja fregmentacija narašča z velikostjo strani. Vsak navidezni naslov se pri odstranjevanju obravnava, kot da je sestavljen iz dveh delov: spodnjih p bitov določa naslov besede znotraj strani, zgornjih n-p bitov pa številko strani. Pri preslikovanju ostane prvi del nespremenjen. drugi del pa se preko tabele strani preslika v številko okvira strani. Vsako polje v tabeli strani pripada točno določeni strani in vsebuje informacijo o njej. Polja v tabeli strani se označujejo z besedo deskriptor strani. Informacijo v deskriptoru predstavlja pet parametrov: V - veljavni bit, P - prisotni bit, RWX - zaščitni ključ, C - umazani bit, FN -številka okvira. Stanje računalnika ko se ne dela nič drugega kot prenašajo programi -uporabniški programi praktično stojijo, označujemo z izrazom premetavanje, ker je njegovo delovanje reducirano na premetavanje informacije med glavnim in pomožnim pomnilnikom. bo premetavanja lahko pride pri vseh vrstah navideznega pomnilnika. Na računalnikih z navideznimi pomnilniki je potrebno vgraditi mehanizme ki preprečujejo premetavanje. Gre za pravilno izbiro parametrov (velikost strani, velikost glavnega pomnilnika, strategija za zamenjavanje strani, število naenkrat izvajajočih programov). Dobra lastnost odstranjevanja je njegova preprostost in zaradi tega je precej razširjeno in ga lahko srečamo pri današnjih računalnikih. Slabost je, da ne omogoča uporabe boljših načinov za zaščito pomnilnika in tega, da si več programov deli isti fizični prostor.

73. Načini dodeljevanja pomnilniškega prostora segmentom.

Načini dodeljevanja pomnilniškega prostora segmentom: Vzemimo da je v danem trenutku v glavnem pomnilniku lukenj, ki jih označimo s sl, s2, … sk, kjer je s/velikost i-te luknje. Če želimo prenesti v glavni pomnilnik segment velikosti S in so luknje razvrščene po njihovi naraščajoči velikosti tako, da velja s1<=s2<=..<=sk, potem lahko podamo naslednje tri algoritme:

- Najboljše ujemanje - pri tem algoritmu poiščemo najmanjše število t za katero velja S-rst V luknjo si prenesemo segment. Razlika med velikostjo segmenta in luknje je v tem primeru najmanjša.
- Najslabše ujemanje - pri tem algoritmu razvrstimo luknje po njihovi padajoči velikosti, sicer pa je enak najboljšemu ujemanju. Razlika med velikostjo segmenta in luknje je v tem primeru največja.
- Prvo ujemanje - pri tem algoritmu so luknje razvrščene po njihovih naraščajočih začetnih naslovih ne glede na velikost. Luknja z najmanjšim indeksom t pri kateri velja S~=st je tista, v katero se prenese segment. Razlika med velikostjo segmenta in luknje je v tem primeru naključna.

74. Segmentacija



Proces je razdeljen v bloke različne dolžine (programski segment, sistemski segment, podatkovni segment). Bloki predstavljajo logično celoto. Slabost ostranjevanja je v tem, da je delitev pomnilniškega prostora na strani povsem mehanična in ne upošteva modularnosti, ki je prisotna v programih. Vsak program je sestavljen iz več logičnih delov - segmentov.

Vrste segmentov:

- javni (do globalnih segmentov lahko dostopajo vsi segmenti. Sistemski programi so običajno definirani kot globalni segmenti).
- lokalni (segmenti pripadajo enemu programu. Dostop do lokalnih segmentov drugim procesom ni dovoljen).

Navidezni pomnilnik, ki kot bloke uporablja segmente, se imenuje segmentacija. Št in velikost segmentov nista fiksni, sta pa omejeni. Spodnji biti navideznega naslova določajo relativni naslov besede v segmentu, zgornji biti št segmenta. Št segmenta se preko tabele segmentov preslika v pomnilniški naslov v glavnem pomnilniku. Vsako polje v tabeli segmentov pripada določenemu segmentu in vsebuje informacijo o njem. Pri prenosu segmenta iz pomožnega v glavni pomnilnik prihaja do zunanje fragmentacije – različno velike luknje v pomnilniku.

Elementi tabele segmentov:

- C umazani bit (ko se segment prenese v glavni pomnilnik se vedno postavi na 0. Če pri izvajanju programa pride do pisanja na katerikoli od naslovov tega segmenta, se bit C postavi na 1),
- L velikost segmenta (trenutna velikost segmenta. Velikost L ni fiksna in se med izvajanjem programa lahko spreminja),
- RWX biti zaščite dostopa do strani (read / write x),
- S naslov v sekundarnem pomnilniku,
- naslov segmenta v primarnem pomnilniku (kje se nahaja segment v primarnem pomnilniku),
- R rezidentni bit (0 če segment ni v primarnem pomnilniku → segment fault. Napaka segmenta sproži past, ki aktivira servisni program za servisiranje napake segmenta; prenese segment iz pomožnega v glavni pomnilnik).

75. Segmentacija z ostranjevanjem.

Z navideznim pomnilnikom, ki uporablja segmentacijo z odstranjevanjem, želimo izkoristiti prednosti segmentacije, vendar brez problemov, ki se pojavljajo pri čisti segmentaciji. Vsak segment je razdeljen na strani. Glavni pomnilnik je razdeljen na okvire strani. Spodnji biti navideznega naslova določajo naslov znotraj strani, srednji biti določajo številko strani, zgornji pa št segmenta.

Preslikovanje poteka preko dveh tabel:

- tabela segmentov,
- tabela strani.

Izboljšavi :

- odstranitev zunanje segmentacije. Odpade potreba po zamudnem stmjevanju pomnilnika,
- boljša izkoriščenost prostora, ki ga zasedajo segmenti. Zaradi delitve segmentov na strani ni več potrebno, da je v glavnem pomnilniku vedno cel segment. Namesto tega zadošča, da so v glavnem pomnilniku samo tiste strani segmenta, ki se trenutno uporabljajo.

76. Vhodno-izhodne naprave (naštejte nekaj primerov V/I naprav; razložite pojma vmesnik in krmilnik)

Vhodno/izhodne naprave lahko razdelimo v dve skupini. V prve spadajo naprave, ki so namenjene za prenos informacije med CPE im glavnim pomnilnikom na eni strani ter zunanjim svetom na drugi strani. Standardne naprave te vrste so npr. tipkovnica, miška, zasloni raznih vrst, tiskalniki, risalniki, telekomunikacijske linije, itd. Obstaja še veliko V/I naprav, ki so specifične za neko področje uporabe računalnika. Pri računalnikih, ki vodijo procese so to npr. senzorji za temperaturo, pritisk, število vrtljajev in podobno ter akumulatorji za npr. spreminjanje hitrosti motorjev, odpiranje ventilov, vključevanje grelcev in podobno. Skupna značilnost vseh naprav je, da sluzijo kot sredstvo, preko katerega računalnik komunicira z zunanjim svetom.
V drugo skupino spadajo naprave ki smo jih označevali z izrazom pomožni pomnilnik. Predvsem magnetni diski raznih vrst. Zaradi tehničnih značilnosti so priključene na CPE in glavni pomnilnik kot V/I naprave, čeprav se uporabnik tega ne zaveda. Skupna značilnost teh naprav je, da služijo za shranjevanje informacije, čeprav se včasih uporabljajo tudi za komuniciranjem z drugimi računalniki. Razlike med V/I napravami so zelo velike; tako glede zapletenosti pravil za komuniciranje z njimi, kot glede njihove hitrosti. Vsaka naprava je priključena preko krmilnika naprave. Krmilnik je fizično običajno v istem ohišju kot CPE in glavni pomnilnik, pogosto celo na isti plošči tiskanega vezja. Pri preprostih napravah je krmilnik trivialen in si ga lahko predstavljamo kot navadna logična vrata ali register brez vsake sposobnosti za procesiranje informacije. Pri zapletenih napravah pa si lahko krmilnik predstavljamo kot specializiran računalnik s fiksnim programom, ki je prilagojen posebnostim te naprave, in skrbi za podrobnosti pri prenosu podatkov. V opisu vsakega krmilnika je točno določeno kaj se zgodi po pisanju v nek register, ali po branju nekega registra. Fizične realizacije krmilnikov se od naprave do naprave in od proizvajalca do proizvajalca razlikujejo. Na nek računalnik lahko priključujemo naprave različnih proizvajalcev, to omogočajo standardizirani vmesniki, s katerim so definirani signali, ki vodijo v in iz V/I naprav. Če je krmilnik narejen v skladu z nekim standardom, je nanj mogoče priključiti vse za ta standard narejene naprave. Gledano iz CPE ali iz V/I procesorja za priključevanje V/I naprav velja naslednje: dostop do naprave poteka preko registrov krmilnika; izvajanje operacij dosežemo tako, da v ustrezen register krmilnika vpišemo ustrezno vrednost, z branjem drugih registrov lahko spreminjamo izvajanje operacije.

77. Naslavljanje V/I naprav.

Namesto izraza izbira naprave se uporablja izraz naslavljanje V/I naprave. Podobno kot pri pomnilniku je tudi tu nujno, da ima vsak register svoj lasten naslov, ki mora biti različen od naslovov vseh ostalih registrov. V splošnem je način naslavljanja, ki ga uporablja nek računalnik, odvisen od uporabljene povezovalne strukture. Število in vrsta vodil ter prisotnost oziroma odsotnost V/I procesorjev v veliki meri določata tudi način naslavljanja. Obstajata tri načina naslavljanja V/I naprav katere uporabljajo praktično vsi današnji računalniki:

- pomnilniški preslikan vhod/izhod - pri njih lahko programer za J dostop do registrov krmilnika uporablja vse tiste ukaze, ki imajo enega od operandov v pomnilniku. Naslovi so vedno v fizičnem pomnilniku ker

zgrešitve pri dostopu do njih ne sme biti;
- ločen vhodno/izhodni prostor - registri krmilnikov naprav so v posebnem naslovnem prostoru, ki je ločen od pomnilniškega;
- posredno preko vhodno/izhodnih procesorjev - CPE nima neposrednega dostopa do registrov naprav.

78. Načini reševanja prenosa podatkov iz in v Vhodno/Izhodne enote.

Osnovna V/I operacija je prenos določene količine podatkov iz glavnega pomnilnika v napravo ali obratno. Količina podatkov in hitrost je odvisna od naprave. Zaradi velikih razlik so se med razvojem izoblikovale različne rešitve. Običajna delitev teh rešitev je glede na stopnjo sodelovanja CPE v v/i prenosih. Uporabljane rešitve delimo na štiri skupine:

- programski vhod/izhod;
- programski vhod/izhod z uporabo prekinitev:
- neposreden dostop do pomnilnika;
- vhodno/izhodni procesorji.

79. Programski Vhod/Izhod.

Programski vhod/izhod - pri tem načinu je delovanje V/I naprave pod neposredno kontrolo CPE. Količina dela, ki ga mora opraviti CPE, je pri tem načinu največja. CPE izvaja program, ki prične V/I operacijo, nadzoruje njeno izvajanje, prenaša podatke in zaključi operacijo. Vsi podatki iz ali v V/I napravo gredo skozi CPE (iz nekega CPE registra v register krmilnika ali obratno). Prenos enega podatka zahteva izvedbo najmanj dveh CPE ukazov: ukaza za prenos iz naprave v CPE (in obratno) in ukaza za prenos v glavni pomnilnik (in obratno). Veliko računalnikov omogoča uporabo programskega vhoda/izhoda, ker je strojno zelo preprost. CPE lahko porabi veliko časa pri preverjanju stanja V/I naprave, ko čaka na naslednji podatek. Ker je veliko V/I naprav za nekaj razredov velikosti počasnejših od povprečnega časa za izvedbo enega CPE ukaza, se že pri razmeroma majhni količini prenesenih podatkov število koristnih CPE ukazov lahko močno zmanjša. Temu ustrezno slaba je izkoriščenost računalnika.

80. Programski Vhod/Izhod z uporabo prekinitev.

izkoriščenost računalnika lahko izboljšamo, če za obveščanje CPE o spremembah v stanju V/I naprave uporabimo prekinitve. Namesto da CPE izvaja program, ki stalno bere register stanja naprave, lahko uporabimo naslednjo rešitev: CPE pošlje napravi ukaz za prenos; namesto da čaka na podatek, CPE nadaljuje z izvajanjem programa: ko je podatek pripravljen, pošlje naprava v CPE zahtevo za prekinitev, vključi se prekinitveni servisni program naprave, ki prenese podatek, po prenosu CPE nadaljuje z izvajanjem programa. Čeprav se tudi pri tem načinu vsi prenosi izvršijo s programom, odpade čakanje, in izkoriščenost računalnika je lahko bistveno večja. To velja še posebej za počasnejše V/I naprave, ki prenašajo podatek s hitrostjo en podatek na nekaj sto ukazov ali počasneje. V takih primerih V/I prenosi zelo malo obremenjujejo CPE in videti je, kot da bi potekali istočasno z izvajanjem programa. V resnici gre seveda za strogo zaporedno delovanje. Ta način delovanja je možen na vseh računalnikih, ker imajo vsi v CPE vgrajeno neko vrsto mehanizma za prekinitve. Vse kar še potrebujemo, je sposobnost krmilnika, da postavlja zahteve za prekinitev v skladu s pravili, ki jih uporablja dana CPE.

81. Prekinitve in pasti (razložite oba pojma, nemaskirane in maskirane prekinitve, prekinitveni vektor, potrjevanje prekinitve, servisiranje prekinitev in pasti).

Prekinitev: dogodek ko CPE prekine izvajanje tekočega programa, in začne izvajati program, ki servisira zahtevo po prekinitvi. Prekinitve so sprožene asinhrono (lahko se zgodijo kadarkoli). Tudi servisni program se lahko prekine (ugnezdene prekinitve). Pasti: predstavljajo izjemne dogodke znotraj CPE (deljenje z 0). Pasti so prožene sinhrono (zgodijo se vedno na istem mestu v programu). Servisiranje izjem: - pojavitev zahteve po servisiranju izjeme, - prepoznavanje izjeme in naprave, ki je izjemo generirala, - shranjevanje stanj CPE na sklad ali v posebne registre ali v rezerviran del pomnilnika (saved area), - izvrševanje programa za servisiranje izjeme (upošteva se prioriteta izjeme), - obveščanje naprave da je njena izjema servisirana, - obnovitev starega stanja CPE. Če ima

CPE več prekinitvenih vhodov in tudi če je na en vhod priključenih več naprav, je potrebno na nek način določiti prioriteto prekinitvenih zahtev. To določa, katera od prekinitev se bo upoštevala prva. Ena možnost za ugotavljanje prioritete je izpraševanje naprave (programsko ali strojno). Boljša rešitev je uporaba marjetične verige (signal potuje iz vhoda na izhod prve naprave, nato iz vhoda na izhod druge naprave in tako naprej). Potrjevanje prekinitve je obveščanje naprave, o tem da je bila njena prekinitvena zahteva upoštevana. Lahko je strojno ali programsko. Prekinitveni prevzemni cikel: čas ko CPE reagira na prekinitev in od naprave z najvišjo prioriteto zahteva identifikacijo naprave in vrsto prekinitve. Naprave lahko samo pošljejo naslov prekinitvenega servisnega programa. Naslov na katerem je shranjen naslov prekinitvenega servisnega programa, pravimo prekinitveni vektor ali vektorski naslov. Vektorski naslov se uporablja kot kazalec v tabelo, ki vsebuje naslove prekinitvenih servisnih programov. Če naprava pošlje samo del naslova iz katerega se po nekem pravilu izračuna vektorski naslov, pravimo temu št prekinitvenega vektorja. Maskirane prekinitve: druge prekinitve jih lahko prekinejo. Nemaskirane prekinitve: druge prekinitve jih ne morejo prekiniti (reset).

82. Direktni dostop do pomnilnika (DMA).

Direktni (neposreden) dostop do pomnilnika - pri napravah, ki zahtevajo hiter prenos velike količine podatkov, postane programski vhod/izhod neuporaben. Tudi če je računalnik dovolj hiter, je tak način prenašanja skoraj vedno nesprejemljiv, ker preveč zasede CPE. V tem primeru tudi prekinitve ne pomagajo. Pri neposrednem prenosu namesto da gredo podatki skozi CPE pod kontrolo programa, jih krmilnik naprave prenaša neposredno v glavni pomnilnik ali iz glavnega pomnilnika. To je možno če sta izpolnjena dva pogoja:
1) obstajati mora neposredna povezava med krmilnikom V/I naprave in glavnim pomnilnikom:
2) krmilnik mora biti sposoben tvoriti pomnilniške naslove in kontrolne signale, ki so potrebni za komuniciranje z glavnim pomnilnikom, poleg tega mora imeti logiko, ki šteje prenesene besede in ki avtomatsko izvrši prenos vsakič, ko je pripravljen naslednji podatek. DMA je krmilnik, ki tvori naslovne in kontrolne signale in opravlja vse operacije, ki jih izvaja program. DMA krmilnik je lahko samostojna enota, na katero so priključeni krmilniki V/I naprav. Lahko pa so njegove sposobnosti vgrajene v krmilnik naprave. Realizacija povezave med DMA krmilnikom in glavnim pomnilnikom je odvisna od števila in vrste vodil, ki sestavljajo povezovalno strukturo danega računalnika. Če želimo z uporabo neposrednega dostopa do pomnilnika opraviti neko V/I operacijo, mora CPE izvesti program, ki naredi naslednje:

- vpiše začetni naslov polja v glavnem pomnilniku v naslovni register DMA krmilnika:
- vpiše število besed, ki naj se prenesejo v števni register DMA krmilnika:
- vpiše v ustrezne registre vrednosti, ki določajo operacijo in način delovanja naprave;
- pošlje ukaz za začetek izvajanja operacije.

Med prenosom podatkov lahko pride do primerov, ko CPE in DMA krmilnik istočasno zahtevata dostop do glavnega pomnilnika. Krmilnik pomnilnika omogoča, da do glavnega pomnilnika istočasno dostopata CPE in DMA krmilnik. Pogoj za to je, da je glavni pomnilnik narejen z uporabo pomnilniškega prepletanja.

83. Vhodno-izhodni procesorji.

Vhodno/izhodni procesorji - pri večini računalnikov, ki običajno delujejo na multiprogramski način postane tudi majhna stopnja sodelovanja CPE neekonomična. Visoka cena CPE, veliko število V/I naprav in velike zahteve po vhodno/izhodnih prenosih opravičujejo vlaganje v dodatne zmogljivosti, ki povečajo izkoriščenost CPE in s tem celotnega računalnika. Te dodatne zmogljivosti so vhodno/izhodni procesorji, ki se pri nekaterih proizvajalcih označujejo tudi kot periferni procesorji ali V/I kanali. Nekateri V/I procesorji so podobni običajnim CPE, drugi so pa tako močno prilagojeni V/I prenosom, da podobnosti ni več. Za vse velja da so to pravzaprav DMA krmilniki, ki imajo sposobnost za izvrševanje zaporedja ukazov, ki lahko obsega veliko število prenosov v ali ‘z glavnega pomnilnika in ne samo en prenos, kot je to pri navadnih DMA krmilnikih. Osnovni cilj je razbremenitev CPE pri V/I prenosih.