SLOVENSKA TECHNICKA UNIVERZITA V **BRATISLAVE**
Fakulta elektrotechniky a informatiky

Ing. Tibor Krajčovič, CSc.

POČÍTAČE

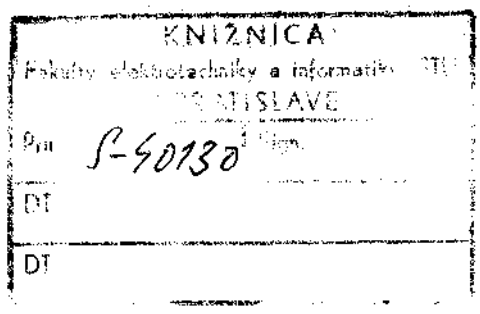
STU
Bratislava

SLOVENSKÁ TECHNICKÁ UNIVERZ1TAV BRATISLAVE
Fakulta elektrotechniky a informatiky

Ing. Tibor Krajčovič, CSc.

POČÍTAČE

2000



Lektori: Prof. Ing. Milan Jelšina, CSc.
Prof. Ing. Pavol Horváth, CSc.

Vydala Slovenská technická univerzita v Bratislave vo Vydavateľstve STU, Bratislava, Vazovova 5.

Schválilo vedenie Fakulty elektrotechniky a informatiky Slovenskej technickej univerzity v Bratislave rozhodnutím č. 5. 10/96 zo dňa 13.2.1996 ako skriptá pre základné štúdium.

ISBN 80-227-1399-6

Uvod

Predkladané skriptá sú základným študijným materiálom predmetu *Počítače,* ktorý sa na *Fakulte elektrotechniky a informatiky STU* v *Bratislave* vyučuje v prvom semestri štúdia. Cieľom tohto predmetu je poskytnúť študentom všetkých študijných odborov základné informácie o technických prostriedkoch počítačov.

Obsahová náplň skriptje rozdelená do šiestich častí, ktoré na seba voľne nadväzujú.

Prvá časť sa zaoberá základnými princípmi počítača riadeného tokom inštrukcií. Je vysvetlená Princetonská a Harvardská architektúra počítačov. Uvedená je tiež základná koncepcia počítača riadeného tokom údajov.

Druhá časť je venovaná problematike zobrazenia informácií v číslicovom počítači. Sú v nej uvedené základné údajové typy, používané číselné a znakové kódy a spôsob realizácie základných aritmetických operácií.

Tretia časť poskytuje základné poznatky o číslicových systémoch. Je vysvetlený pojem číslicového systému, základy Boolovskej algebry a spôsoby zápisu logickej funkcie. Táto časť sa tiež venuje kombinačným a sekvenčným logickým obvodom a základným stavebným prvkom číslicových systémov.

Štvrtá časť sa podrobnejšie venuje výstavbe počítačov s jedným prúdom inštrukcií a jedným prúdom údajov. Je rozobraný prepojovací podsystém počítača. Detailne je vysvetlená základná koncepcia procesora. Uvedené sú spôsoby zvyšovania výkonnosti súčasných procesorov a analyzovaný prerušovací podsystém. Pri pamäťovom podsystéme sa pozornosť venuje okrem hlavnej a vyrovnávacej pamäte aj štandardným vonkajším pamätiam. Rozoberajú sa princípy ochrany a správy hlavnej pamäte. V časti, ktorá sa zaoberá vstupno/výstupným podsystémom počítača, je naznačené pripojenie periférnych zariadení k zbernici a analyzovaná komunikácia procesora s adaptérom periférneho zariadenia. Pozornosť sa venuje spojeniu počítača s technologickým prostredím a analyzujú sa základné metódy vstupno/výstupných prenosov.

Piata časť sa zaoberá počítačmi s jedným prúdom inštrukcií a viacerými prúdmi údajov. Opísaný je princíp maticového procesora.

Šiesta časť sa venuje počítačom s viacerými prúdmi inštrukcií a viacerými prúdmi údajov. Detailnejšie sa tiež zaoberá počítačovými sieťami.

1 Základná koncepcia číslicového počítača

Číslicový počítač je definovaný ako zložitý *univerzálny číslicový systém (automat),* určený na samočinné vykonávanie požadovanej postupnosti *operácií (výpočtov)* nad *údajmi,* zobrazenými *číslicovým kódom,* na základe vopred pripraveného a v *pamäti* uloženého *programu* [14].

Základný koncept počítača, ktorý vykonáva postupnosť operácií na dosiahnutie konečného výsledku, je známy viac ako 150 rokov. Bol použitý v mechanických dekadických počítacích strojoch, ktoré navrhol a čiastočne zostrojil *Charles Babbage*. Jeho *analytický stroj (Analytical Engine)* z r. 1834 obsahuje *centrálnu procesorovú jednotku* (mechanickú, s dekadickou aritmetikou), *pamäť* (mechanickú) a *vstupnú* a *výstupnú jednotku* (pre dierne karty), t.j. všetky základné časti moderných počítačov. *Program* a *údaje* pre tento stroj sú na diernych kartách. Samozrejme, vtedajšia mechanická technológia nedovolila úspešnú realizáciu funkčných zariadení, takže jeho myšlienky zostali viac ako 100 rokov nevyužité. Až vytvorenie *elektronických obvodov* v 40. rokoch 20. storočia ich umožnilo zaviesť do praxe. V tomto čase *John von Neumann* predložil základný princíp *počítača riadeného tokom inštrukcií,* ktorý sa stal základom jednej triedy súčasných počítačov. Táto trieda počítačov býva tiež označovaná ako *von Neumannovské počítače.* Vyznačuje sa tým, že jednotlivé inštrukcie programu sa vykonávajú postupne za sebou, tak ako sú uložené v pamäti.

V súčasnosti existujú aj výpočtové systémy, v ktorých sa inštrukcie nevykonávajú v poradí, v akom sú uložené v pamäti (napr. *počítače riadené tokom údajov, neurónové počítače* a iné). V časti 1.2 je pre ilustráciu opísaný princíp počítača riadeného tokom údajov. V tomto počítači sa vykoná práve tá inštrukcia, ktorá má pripravené údaje, pričom nezáleží od jej poradia.

1.1 Počítače riadené tokom inštrukcií (von Neumannovské počítače)

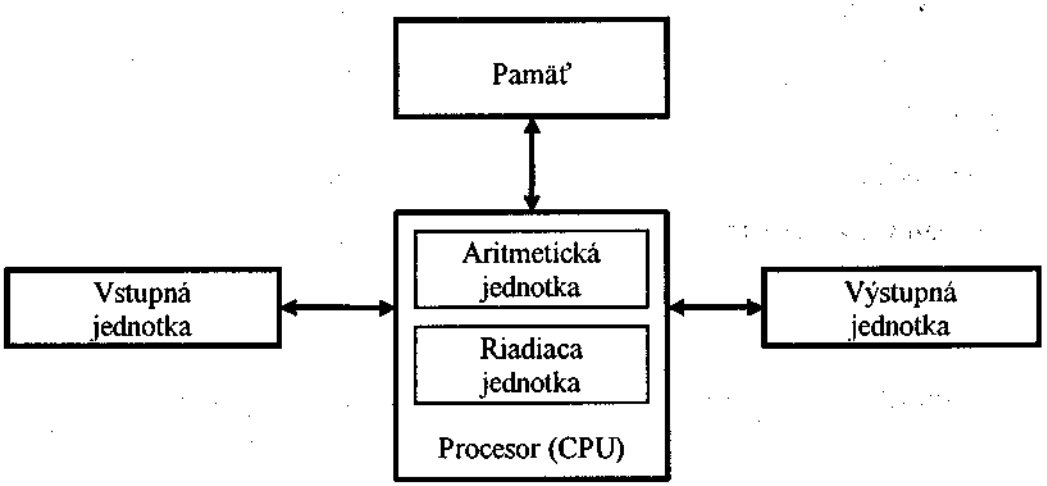
Základné črty von Neumannovského počítača sú [20]:

1. Pamäť je použitá na uloženie *inštrukcií* aj *údajov.*

2. Riadiaca jednotka j e použitá na výber inštrukcií z pamäte.

3. Aritmetická jednotka je použitá na vykonávanie špecifikovaných operácií nad údajmi.

4. Vstupná jednotka je použitá na vstup údajov, *výstupná jednotka* na výstup údajov.



OBR. 1. Bloková schéma von Neumannovského počítača

Pamäť je množina rovnakých buniek, z ktorých každá je samostatne identifikovateľná svojím poradovým číslom - *adresou*. Inštrukcie a údaje, uložené v pamäti, sú zakódované *dvojkovým kódom*.

Inštrukcia (príkaz pre riadiacu jednotku) určuje, aká operácia sa má vykonať a s ktorými údajmi. Inštrukcie sa vykonávajú postupne za sebou, tak ako sú uložené v pamäti. Výnimkou sú *skokové inštrukcie*. Implicitne sa predpokladá *pripravenosť údajov,* ktoré vykonávaná inštrukcia požaduje.

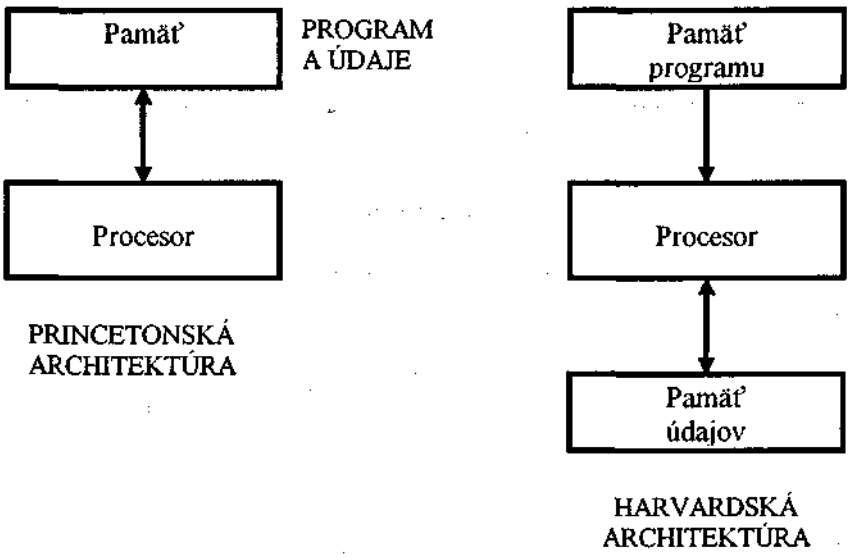
Dvojkovo zakódované inštrukcie sú označované ako *strojové inštrukcie*. Operácie, špecifikované v strojových inštrukciách, sú obyčajne iba jednoduché, napr. *aritmetické a logické operácie, posuvy* atď., čo poskytuje najväčšiu flexibilitu. Zložitejšie operácie potom vytvára používateľ ako *postupnosť inštrukcií*. Z danej množiny strojových inštrukcií *(inštrukčný súbor procesora)* používateľ vyberá inštrukcie na vykonanie požadovaného výpočtu. Táto postupnosť vybraných inštrukcií sa nazýva *strojový program počítača*.

Riadiaca jednotka a *aritmetická jednotka* sú zvyčajne realizované ako jeden funkčný blok, ktorý sa nazýva *centrálna procesorová jednotka (CPU)* alebo skrátene *procesor*.
Ak je procesor integrovaný na jedinom polovodičovom čipe, nazýva sa *mikroprocesor*.
Procesor obsahuje niekoľko *egistrov*, ktoré sú použité na uchovávanie špecifických operandov, použitých pri výpočte, adries a riadiacich informácií. Počet registrov a ich funkcia závisí od konkrétneho procesora, ale niektoré registre sú prítomné v každom procesore von Neumannovského počítača. Typickým registrom je *programové počítadlo (register PC)*. Obsahuje adresu nasledujúcej inštrukcie, ktorá sa bude vykonávať.

1.1.1 Princetonská a Harvardská architektúra

Von Neumannov počítač je predstaviteľom tzv. *Princetonskej architektúry počítačov*, ktorá sa vyznačuje spoločnou pamäťou pre inštrukcie i údaje. Z tohto vyplýva, že je potrebné zabezpečiť, aby procesor neinterpretoval údaj ako inštrukciu a naopak. Prístup procesora k pamäti je totiž rovnaký, či sprístupňuje inštrukciu alebo údaj - používajú sa tie isté *adresové, údajové i riadiace signály*. Takéto usporiadanie pamäte potom umožňuje používať aj *samomodifikujúce sa programy*, t.j. program počas svojho behu môže meniť sám seba. Treba si však uvedomiť, že takáto situácia môže nastať aj neželane, či už nesprávnym programom , alebo vplyvom *poruchy*. Niektoré súčasné typy procesorov už vykonávajú kontrolu správnosti prístupu k pamäti vlastnými technickými prostriedkami. V prípade, že sa niektorý prístup k pamäti vyhodnotí ako nesprávny (napr. procesor sa pokúša zapisovať do oblasti pamäte, ktorá je vyhradená na uloženie inštrukcií), automaticky sa generuje *výnimka (pozri Správa a ochrana pamäte)*. Túto situáciu potom rieši *operačný systém* počítača.
Príkladom počítačov s Princetonskou architektúrou sú napr. počítače s procesormi rodiny *80x86*.
Počítače s *Harvardskou architektúrou* majú oddelený adresový priestor pre program a pre údaje, takže situácia, aby program prepísal sám seba, nemôže nastať. Táto architektúra sa v súčasnosti používa najmä pri niektorých *jednočipových mikropočítačoch*.
Jednočipový mikropočítač sa vyznačuje tým, že všetky štruktúrne prvky počítača (t.j. procesor, pamäť a vstupné a výstupné obvody) sú integrované na jedinom polovodičovom čipe. Procesor používa na adresáciu obidvoch pamätí a na prenos údajov a inštrukcií obyčajne spoločné adresové a údajové vodiče, rozlišovanie medzi prístupom k pamäti programu a k

pamäti údajov vykoná aktiváciou odlišných riadiacich signálov. Príkladom počítača s Harvardskou architektúrou je jednočipový mikropočítač *8051.*



OBR. 2. Organizácia pamäte počítačov s Princetonskou a Harvardskou architektúrou

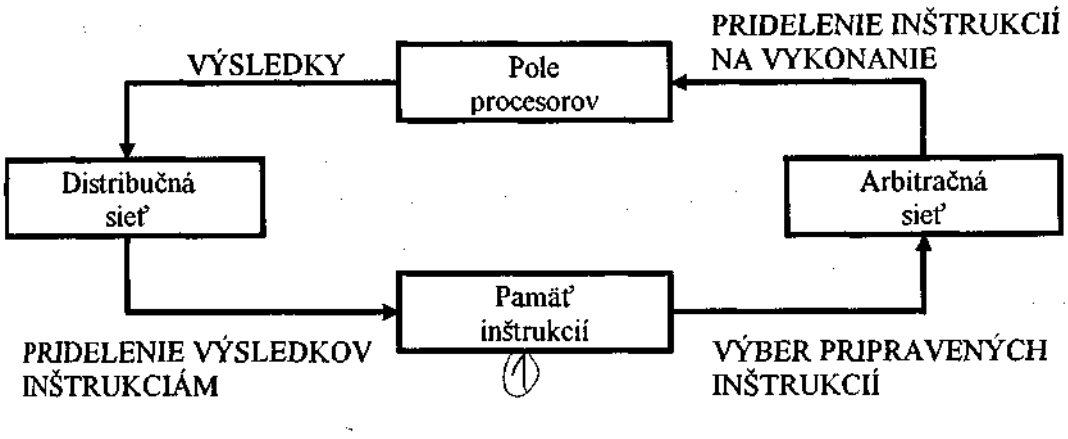
1.2 Počítače riadené tokom údajov (data-flow systémy)

Tieto počítače nevykonávajú inštrukcie postupne za sebou, tak ako sú uložené v pamäti, ale *vykoná sa práve tá inštrukcia, ktorá má pripravené údaje.* Ak má viac inštrukcií pripravené svoje údaje, tieto inštrukcie sa vykonajú paralelne. Treba poznamenať, že je potrebné vždy vytvoriť toľko kópií vstupných údajov, koľko je inštrukcií, ktoré ich budú potrebovať. Počítače riadené tokom údajov predstavujú osobitnú triedu *paralelných počítačov.* Sú to počítače novej generácie s vysokou výkonnosťou. Ďalšou významnou vlastnosťou je, že programu sa prispôsobuje štruktúra technických výpočtových prostriedkov. Na obr. 3 je principiálna bloková schéma počítača, riadeného tokom údajov.

V pamäti inštrukcií sa nachádzajú všetky inštrukcie programu.

Arbitračná sieť zisťuje, ktorá inštrukcia (inštrukcie) je pripravená na vykonanie (t.j. ktorá má pripravené všetky svoje vstupné údaje). Túto inštrukciu potom vyberie a pridelí ju na vykonanie niektorému voľnému procesoru z *poľa procesorov.* Keď procesor vykoná príslušnú inštrukciu, pošle výsledok do distribučnej siete.

Distribučná sieť pridelí výsledok všetkým inštrukciám, ktoré v pamäti inštrukcií na tento údaj čakajú.



OBR. 3. Bloková schéma počítača riadeného tokom údajov

Program pre počítač riadený tokom údajov sa obyčajne zobrazuje ako *orientovaný graf,* v ktorom *uzly* reprezentujú asynchrónne aktívne členy *(operátory, inštrukcie, úlohy)* a *hrany* reprezentujú komunikačné cesty na prenos a smerovanie správ *(údajových balíkov, operandov),* generovaných uzlami počas ich aktivácie alebo prijímaných z externého prostredia počas výpočtu. Tento graf sa nazýva *graf programu*.

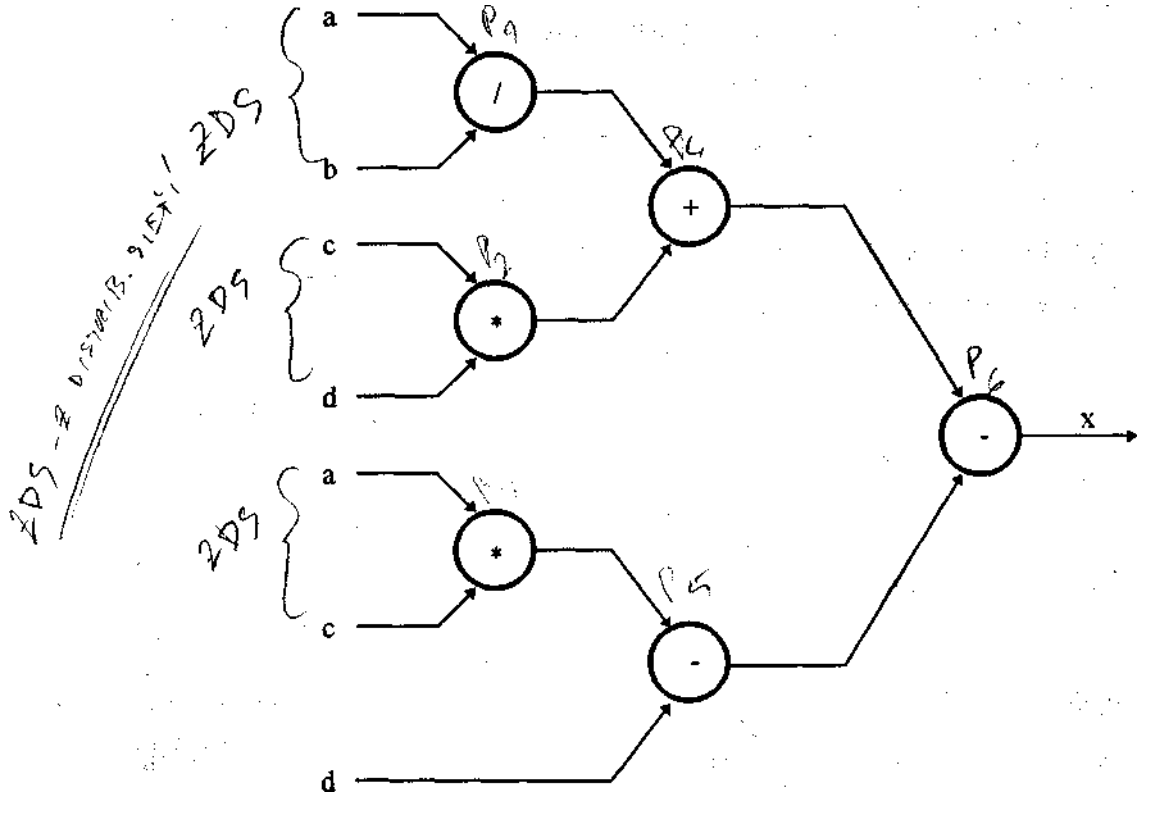
Na obr. 4 je graf programu na realizáciu výpočtu x := (a/b + c*d) - (a*c - d).

1.3 Klasifikácia počítačov

1.3.1 Rozdelenie podľa aplikačného určenia

1. *Univerzálne počítače* predstavujú prevažnú väčšinu systémov, určených na riešenie vedecko-výskumných, inžinierskych, administratívnych a iných úloh. Niektoré triedy úloh riešia s väčšou, iné triedy úloh s menšou efektívnosťou. Tieto počítače používajú štandardné univerzálne procesory s inštrukčným súborom, ktorý je z hľadiska spracovania informácie úplný a štandardné prostriedky na komunikáciu s hlavnou pamäťou a vstupno/výstupným podsystémom.

2. *Problémovo orientované počítače* sú určené na riešenie problémov z danej triedy, napr. na spracovanie signálov. Používajú procesory s prispôsobeným alebo špeciálne navrhnutým súborom inštrukcií a špecializované periférne zariadenia na efektívne riešenie úloh danej triedy.
3. *Aplikačne špecifické počítače* sú svojou architektúrou orientované na riešenie jedného problému (napr. riadenie nejakého technologického procesu). Z danej úlohy vyplýva špecifikácia architektúry systému a jeho implementácia. Neznamená to, že počítače tejto triedy nevyužívajú štandardné procesory. Obyčajne však obsahujú špeciálne, na mieru vytvorené technické prostriedky a programové vybavenie. Do tejto triedy sa zaraďujú aj tzv. *vnorené (embedded) systémy,* v ktorých je počítač neoddeliteľnou súčasťou riadeného zariadenia.



OBR. 4. Graf programu pre výpočet x := (a/b + c*d) - (a*c - d)

1.3.2 Rozdelenie podľa architektonickej koncepcie

Podľa architektonickej koncepcie existuje viacero klasifikácií počítačov. *Flynnova klasifikácia* vychádza z počtu súčasne spracúvaných tokov inštrukcií a tokov údajov v počítači. Na základe tohto kritéria Flynn klasifikuje 4 triedy počítačov:

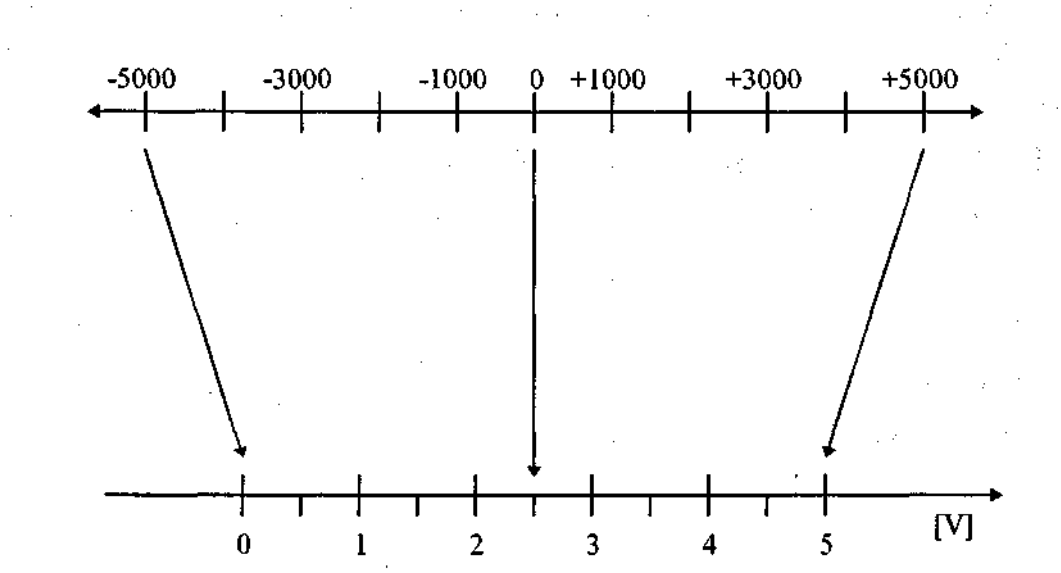
1. *SISD (Single Insíruction stream Single Data stream)*. Jeden tok inštrukcií, jeden tok údajov. Táto architektúra predstavuje architektúru Von Neumannovho počítača, kde jeden procesor interpretuje jeden prúd inštrukcií (strojový program) a v zodpovedajúcom *procese* sa spracúva jeden prúd údajov.
2. *SIMD (Single Instruction stream Multiple Data stream).* Jeden tok inštrukcií, viac tokov údajov. Táto architektúra má významné použitie pri tvorbe paralelných počítačov. Uplatňuje sa najmä pri tvorbe systémov, ktoré vykonávajú *vektorové* a *maticové* operácie, t.j. súčasne sa vykonáva jedna operácia s viacerými údajovými štruktúrami rovnakého typu. Jeden program sa vykonáva súčasne s viacerými prúdmi údajov vo viacerých *procesných elementoch* (v tzv. *pasívnych procesoroch,* ktoré dostanú inštrukciu z riadiacej jednotky a vykonajú ju). Tieto systémy dosahujú podstatne lepšie výsledky ako architektúra SISD pri riešení úloh uvedenej triedy.
3. *MISD (Multiple Instruction stream Single Dala stream).* Niekoľko tokov inštrukcií, jeden tok údajov. Táto trieda predstavuje určitú teoretickú koncepciu, ktorá sa v počítačoch ako v celkoch neaplikuje a zatiaľ sa neobjavil ani jej praktický význam.
4. *MIMD (Multiple Instruction stream Multiple Data stream).* Viacnásobný tok inštrukcií, viacnásobný tok údajov. Architektúra MIMD predstavuje veľmi širokú triedu paralelných systémov, do ktorej zaraďujeme *multiprocesorové* a *multipočítačové systémy* s paralelným spracovaním. Ide o paralelné spracovanie s oddelene prebiehajúcimi *paralelnými procesmi,* riadenými *samostatnými* procesormi. *Multiprocesorový systém* je paralelný počítač, obsahujúci niekoľko procesorov, ktoré majú *vlastnú* alebo *zdieľanú* pamäť a na komunikáciu s okolím používajú *spoločné* vstupné a výstupné zariadenia. *Multipočítačový (distribuovaný) systém* sa skladá z niekoľkých počítačov s možnosťou vzájomnej komunikácie, ktoré sú schopné aj samostatnej činnosti. Patria sem aj *počítačové siete.*

2 Zobrazenie informácií v počítači

Informácie sa v počítači zobrazujú prostredníctvom *hodnôt istých premenných fyzikálnych veličín.* V reálnom počítači môžu tieto fyzikálne veličiny nadobúdať hodnoty iba z *konečnej množiny hodnôt.* Nositeľom hodnoty v elektronických počítačoch je *elektrické napätie.*

2.1 Analógové zobrazenie informácií

Pri *analógovom zobrazení* ide o *spojité zobrazenie,* t.j. každej hodnote vstupnej veličiny zodpovedá nejaká hodnota fyzikálnej veličiny (ktorá je nositeľom informácie) v počítači.
Na obr. 5 je naznačené analógové zobrazenie informácií. Vstupnú veličinu predstavujú čísla v rozsahu *<-5000; +5000>*. Im zodpovedá hodnota napätia v počítači v rozsahu *<0 V; +5 V>*. Tak napr. číslu *-5000* zodpovedá v tomto prípade napätie *0 V,* číslu *0* napätie *2.5 V* a číslu *+5000* napätie *+5 V.* Uvedený spôsob zobrazenia sa používal pri *analógových počítačoch,* s ktorými sa už v praxi nestretneme.



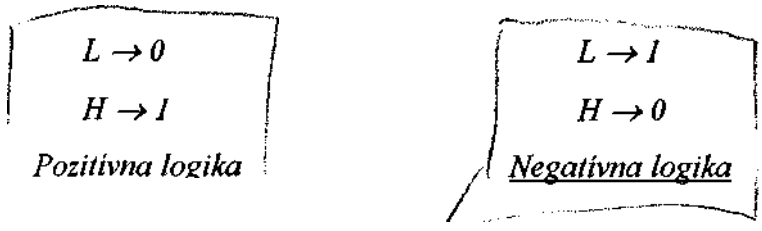
OBR. 5. Princíp analógového zobrazenia informácií

2.2 Číslicové zobrazenie informácií

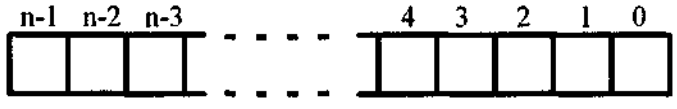
Pri *číslicovom zobrazení* sa údaje v počítači uchovávajú vo vnútorných *pamäťových prostriedkoch,* ktoré sa nazývajú *registre* a *pamäti.*

Register je usporiadaná n-tica základných pamäťových elementov, ktoré sú *dvojhodnotové,* t j. sú schopné uchovávať jednu z dvoch hodnôt *H* resp. *L,* kde *H* označuje vysokú a *L* nízku hodnotu fyzikálnej veličiny, ktorá je nositeľom informácie.

Týmto dvom hodnotám fyzikálnej veličiny je možné priradiť *boolovské (logické)* hodnoty *0* a 7 jedným z týchto spôsobov:



Jedno pamäťové miesto v registri sa nazýva *bit* (skratka z *binary digit - dvojková číslica).* Jednotlivé bity registra budeme označovať číselne sprava doľava, začíname číslom *0.* Tento bit budeme nazývať ako *nultý,* ďalší v poradí bude *prvý* atď. Nultý bit je *najmenej významný, (n-1)*-bite *najvýznamnejší.* Na obr. 6 je znázornený *n-bitový* register (register s dĺžkou *n*) s označením jednotlivých bitov.



OBR. 6. *n-bitový* register

Pamäť je množina rovnakých buniek, z ktorých každá je samostatne identifikovateľná svojou adresou. Pre každú bunku pamäte platí rovnaká špecifikácia, aká už bola uvedená pre register.

2.3 Údajové typy

Údaje v počítači môžu byť vo všeobecnosti rôzne a tak hovoríme o *údajových (dátových) typoch*. V ďalšom uvedieme základné údajové typy a spôsob ich zobrazenia v číslicovom počítači. Pri číslach uvedieme aj spôsob realizácie základných aritmetických operácií.

2.3.1 Boolovské typy

Najjednoduchší údajový typ je *jednoduchý boolovský typ*, pri ktorom má množina hodnôt iba dva prvky, ktoré označujeme symbolmi *T* a *F (True* resp. *False)* alebo *1* a *0.* Tento údajový typ je možné priamo zobraziť do registra s dlžkou *n,* pričom bude zaberať iba jediný bit. Ostatné bity by boli nevyužité a tak sa často do registra s dlžkou *n* zobrazuje *n* booleovských premenných, z ktorých každej je priradený jeden bit.

Zložitejším údajovým typom je *boolovský vektor s dĺžkou k*. Ak je $k < n$, tento vektor je možné priamo zobraziť do registra s dlžkou *n*.

2.3.2 Čísla a základné aritmetické operácie

2.3.2.1 Vyjadrenie čísel v pozičnej číselnej sústave

Každé číslo je možné vyjadriť v pozičnej číselnej sústave v tvare:

$$\pm (a_n . z^n + a_{n-1} . z^{n-1} + \ldots + a_2 . z^2 + a_1 . z^1 + a_0 . z^\circ + a_{-1} . z^{-1} + a_{-2} . z^{-2} + \ldots)$$

kde : z - základ danej číselnej sústavy (prirodzené číslo)

a_i - koeficient (prirodzené číslo), pričom $a < z$

Číslo sa zapisuje v skrátenom tvare: $\pm a_n a_{n-1} a_{n-2} \ldots a_2 a_1 a_0 , a_{-1} a_{-2} \ldots$

Hodnota každej číslice je jednoznačne daná jej pozíciou v rámci celého čísla.

2.3.2.2 Číselné sústavy používané v číslicových počítačoch

Najpoužívanejšia je *dvojková* číselná sústava. *Osmičková* a *šestnástková* číselná sústava sa často používa na zrozumiteľnejší zápis dlhších dvojkových čísel.

Dvojková číselná sústava

Základ : 2

Koeficienty: 0, 1

Osmičková číselná sústava

Základ : 8

Koeficienty: 0, 1, 2, 3, 4, 5, 6, 7

Desiatková číselná sústava

Základ : 10

Koeficienty : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Šestnástková číselná sústava

Základ : 16

Koeficienty : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

2.3.2.3 Prevody medzi číselnými sústavami

Prevod z desiatkovej do inej číselnej sústavy sa najjednoduchšie urobí postupným celočíselným delením desiatkového čísla (resp. jeho podielov) základom požadovanej číselnej sústavy. Zvyšok po delení je jednou číslicou čísla v požadovanej číselnej sústave, pričom prvý zvyšok bude v najnižšom ráde. Delenie pokračuje dovtedy, kým nie je celočíselný podiel

menší ako základ novej číselnej sústavy. Tento posledný podiel je potom poslednou číslicou čísla v novej číselnej sústave.

Príklad:

Prevod desiatkového čísla *13* do dvojkovej číselnej sústavy:

13:2 = 6:2 = 3:2 = 1

Zvyšok p o delení: 1 0 1 1

Číslo *13* v dvojkovej číselnej sústave: *1101*

Prevod z inej číselnej sústavy do desiatkovej je veľmi jednoduchý, váhu každej číslice vyjadríme dekadicky a spočítame.

Príklad:

Vykonajme prevod osmičkového čísla *(3427)* do desiatkovej číselnej sústavy. Ako operátor pre súčin použijeme znak "*" :

$$3*8^3 + 4*8^2 + 2*8^1 + 7*8^0 = 3*512 + 4*64 + 2*8 + 7*1 =$$
$$= 1536 + 256 + 16 + 7 = 1815$$

Priamy prevod z dvojkovej do osmičkovej a šestnástkovej číselnej sústavy využíva skutočnosť, že číslo *8* je treťou a číslo *16* štvrtou mocninou čísla 2, to znamená, že nemusíme robiť medziprevod do desiatkovej číselnej sústavy.

Pri prevode z dvojkovej do osmičkovej číselnej sústavy rozdelíme dvojkové číslo na *trojice bitov sprava*. Jednotlivé trojice potom vyjadríme osmičkovými číslicami a dostaneme výsledok.

Prevod z dvojkovej do šestnástkovej číselnej sústavy je rovnaký, ale namiesto trojíc sa vytvoria *štvorice*.

Príklad:

Vykonajme prevod dvojkového čísla *(100101101110010110)* do šestnástkovej číselnej sústavy:

Rozdelíme číslo na štvorice bitov sprava: 10 0101 1011 1001 0110

Vyjadríme štvorice šestnástkovými číslicami: 2 5 B 9 6

Prevod zo šestnástkovej resp. osmičkovej číselnej sústavy do dvojkovej je opačný - každú číslicu vyjadríme trojicou resp. štvoricou bitov.

2.3.2.4 Prirodzené čísla

Pod *prirodzenými číslami* budeme rozumieť *celé kladné čísla vrátane nuly*. Na ich zobrazenie sa používa *prirodzený dvojkový kód*.

Rozsah zobrazenia pre n-bitový register: $\langle 0, 2^n-1 \rangle$

Základné aritmetické operácie

Uvedieme príklady na operácie sčítania, odčítania, násobenia a delenia s prirodzenými číslami, zobrazenými v prirodzenom dvojkovom kóde. Kvôli jednoduchosti bude použitá šírka slova

Sčítanie

Pravidlá na sčítanie jednotlivých bitov: 0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 (prenos 1 do vyššieho rádu)

Sčítanie začíname sprava (od najnižšieho rádu), ak nastane prenos, musí sa pripočítať k ďalšiemu (vyššiemu) rádu.

Príklad:

```
  0110   (6)
+ 0101   (5)
---------
  1011   (11)
```

V prípade, ak nastane prenos z najvyššieho rádu, nastalo *preplnenie (pretečenie, overflow)* a výsledok je neplatný. Na realizáciu operácie sčítania sa používajú *dvojkové sčítačky*.

Odčítanie

Pravidlá na odčítanie jednotlivých bitov: 0 - 0 = 0

0 - 1 = 1 (prenos 1 do vyššieho rádu)

1 - 0 = 1

1 - 1 = 0

Odčítanie sa začína sprava (od najnižšieho rádu), ak nastane prenos, musí sa odpočítať od ďalšieho (vyššieho) rádu.

Príklad: 0110 (6)

\- 0101 (5)

0001 (1)

V prípade, ak nastane prenos z najvyššieho rádu, nastalo *podtečenie (underflow)* a výsledok je neplatný. Na realizáciu operácie odčítania sa používajú *dvojkové odčítačky.*

Násobenie

Podľa bežne používaného algoritmu sa operácia násobenia vykonáva spočítaním posunutých *čiastočných súčinov*. *Čiastočný súčin* je násobok prvého činiteľa číslicou druhého činiteľa. Násobenie dvojkových čísel je jednoduchšie ako v iných číselných sústavách, pretože čiastočný súčin môže mať buď hodnotu *0* (ak je číslica druhého činiteľa *0*) alebo sa rovná prvému činiteľovi (ak je číslica druhého činiteľa *1*).

Príklad: 0110 (6)

0101 (5)

0110

0000

0110

0000

0011110 (30)

Pri realizácii *násobičky* sa často používajú sčítačky s dvoma vstupnými operandmi a tak sa celkový výsledok dostane postupne spočítaním dvojíc čiastočných súčinov. Tiež sa používa *paralelná násobička,* realizovaná pomocou pamäte.

V prípade, ak sú *n-bitové* operandy, výsledok môže mať až *2n* bitov.

Delenie

Podľa bežne známeho algoritmu delenia sa dve čísla delia odpočítavaním násobkov deliteľa od časti delenca, resp. od jeho *čiastočných zvyškov.* Prvá časť delenca sa vytvorí takým spôsobom, že sa berú číslice delenca zľava dovtedy, kým nedostaneme číslo, ktoré je väčšie ako deliteľ. Od tejto časti sa odpočíta najväčší možný násobok deliteľa. Číslo, ktoré udáva, koľkokrát sa deliteľ nachádza v časti delenca, je súčasne číslicou podielu v danom ráde. K zvyšku po odpočítaní sa pridá ďalšia číslica delenca a tým dostaneme novú časť delenca - *čiastočný zvyšok.* Tento sa porovná s deliteľom. Ak je menší ako deliteľ, číslica podielu v tomto ráde je *0.* V opačnom prípade na určenie číslice podielu platí to, čo už bolo uvedené. Pri *celočíselnom delení* je delenie skončené, ak je posledný čiastočný zvyšok menší ako deliteľ.

Na realizáciu operácie delenia v číslicových sú známe viaceré algoritmy. Delenie dvojkových čísel je jednoduchšie ako delenie čísel v iných číselných sústavách, pretože číslica podielu v danom ráde je *0* alebo *J* a netreba odhadnúť, koľkokrát sa deliteľ nachádza v časti delenca resp. v čiastočnom zvyšku. Sú tu len dve možnosti - deliteľ sa nenachádza v čiastočnom zvyšku (je menší), alebo sa v ňom nachádza (je väčší alebo sa rovná). V prvom prípade je

číslica podielu *0,* v druhom *J.* Výsledkom celočíselného delenia sú dve čísla -*podiel* a *zvyšok po delení.* Zvyšok po delení môže byť nulový, ak je nenulový, je vždy menší ako deliteľ.

Príklad:

(11) (2) (5)

1011 : 10 = 101

\- 10 (deliteľ sa nachádza v časti delenca - prvá číslica podielu je *1)*

001 (čiastočný zvyšok)
00 (deliteľ sa nenachádza v čiastočnom zvyšku - číslica podielu je *0)*

0011 (čiastočný zvyšok)
10 (deliteľ sa nachádza v čiastočnom zvyšku - číslica podielu je *J)*

01 (zvyšok po delení)

2.3.2.5 Celé čísla

Na zobrazenie celých čísel sa využíva niekoľko kódov, ktoré sa líšia tak rozsahom zobrazenia, ako aj svojou vhodnosťou pre rôzne aritmetické operácie. Uvedieme najpoužívanejšie kódy, a to *priamy kód, inverzný kód, doplnkový kód a predpätý kód.*

Priamy kód

V priamom kóde sa najvyšší bit používa ako *znamienkový bit.* Kladné čísla sa zobrazujú rovnako, ako prirodzené čísla, záporné číslo sa líši od kladného tým, že znamienkový bit má jednotkový.

Rozsah zobrazenia pre *n-bitový* register: $<-2^{n-1}+1, 2^{n-1}-1>$. Nevýhodou je, že číslo *0* má dva obrazy.

Príklad: číslo 6 v priamom kóde : 0110
číslo -6 v priamom kóde : 1110

Základné aritmetické operácie

Operácie *sčítania* a *odčítania* je možné vykonávať iba s číslami s rovnakým znamienkom, pričom znamienkový bit do operácie nevstupuje. Prenos do znamienkového bitu signalizuje pretečenie (preplnenie) resp. podtečenie a výsledok je neplatný.

Operácie *násobenia* a *delenia* je možné vykonávať s číslami s ľubovoľným znamienkom, pričom znamienkový bit do operácie nevstupuje. Znamienko výsledku sa určí na základe znamienok obidvoch operandov.

Inverzný kód

V inverznom kóde má najvyšší bit opäť význam znamienka, ale na rozdiel od priameho kódu vstupuje do operácie. Kladné Čísla sa zobrazujú rovnako, ako prirodzené čísla. Záporné číslo sa získa takým spôsobom, že kladné číslo s rovnakou absolútnou hodnotou sa *invertuje* bit po bite. (Invertovať znamená zmeniť hodnotu každého bitu na opačnú.)

Rozsah zobrazenia je rovnaký, ako v priamom kóde. Číslo 0 má opäť dva obrazy.

Príklad: číslo 6 v inverznom kóde : 0110
číslo -6 v inverznom kóde : 1001

Základné aritmetické operácie

V tomto kóde je možné vykonávať operácie *sčítania* a *odpočítania* s číslami s ľubovoľným znamienkom, v niektorých prípadoch je však nutné vykonať tzv. *kruhový prenos* do najnižšieho rádu. Operácia odčítania býva realizovaná ako pripočítanie čísla s opačným znamienkom a potom je možné použiť tak pre sčítanie, ako aj pre odčítanie sčítačku. Nepoužíva sa pre operácie *násobenia* a *delenia.*

Doplnkový kód

V *doplnkovom kóde,* tak ako v inverznom kóde, má najvyšší bit význam znamienka a tiež vstupuje do operácie. Kladné čísla sa zobrazujú rovnako, ako prirodzené čísla. Záporné číslo v doplnkovom kóde získame zo záporného čísla v inverznom kóde tak, že k nemu pripočítame jedničku v najnižšom ráde. Výsledkom je vlastne doplnok absolútnej hodnoty záporného čísla do čísla *2".*

Rozsah zobrazenia pre *n-bitový* register: $<-2^{n-1}, 2^{n-1}-1>$

U tohto kódu má číslo *0* iba jediný obraz.

Príklad: číslo 6 v doplnkovom kóde : 0110
číslo -6 v doplnkovom kóde : 1010

Základné aritmetické operácie

V tomto kóde je možné vykonávať operácie *sčítania* a *odčítania* s číslami s ľubovoľným znamienkom, pričom nie sú nutné žiadne korekcie. *Pretečenie* resp. *podtečenie* je indikované *rôznymi hodnotami prenosov do najvyššieho (znamienkového) bitu a z najvyššieho bitu.*
Operácia odčítania býva realizovaná ako pripočítanie čísla s opačným znamienkom a potom je možné použiť tak pre sčítanie, ako aj pre odčítanie *sčítačku.*

Kód je nevhodný pre priamu realizáciu operácie *delenia*. Je však známy algoritmus pre operáciu *násobenia* [1].

Príklad: Máme *4-bitový* register. Rozsah zobrazenia v doplnkovom kóde je <-*8,* 7>. Skontrolujme platnosť výsledku pri operácii sčítania nasledovných čísel:

```
  0011   (3)
+ 0100   (4)

  0111   (7)
```

Prenos do najvyššieho rádu bol *0,* prenos z najvyššieho rádu bol *0,* takže výsledok je platný.

```
Príklad:  1101   (-3)
        + 1100.  (-4)

        1 1001   (-7)
```

Prenos do najvyššieho rádu bol *J,* prenos z najvyššieho rádu bol takisto *1,* takže výsledok je platný. (Pretože máme iba *4-bitový* register, prenos z najvyššieho rádu sa nezobrazí, je uvedený iba kvôli tomu, aby bolo vidno, že vznikol.)

```
Príklad:  1101   (-3)
        + 0110   (6)

          10011  (3)
```

Prenos do najvyššieho rádu bol 1, prenos z najvyššieho rádu bol takisto 1, takže výsledok je platný.

Príklad: 0110 (6)
+ 0101 (5)

1011

Prenos do najvyššieho rádu bol *1,* prenos z najvyššieho rádu bol *0,* takže výsledok je neplatný (nastalo preplnenie). Naozaj, súčet je *(11)* a to je už mimo intervalu zobrazenia. Okrem toho, *1* v najvyššom ráde signalizuje v danom kóde záporné číslo a nie je možné, aby sme pri sčítaní dvoch kladných čísel dostali výsledok záporný.

Príklad: 1101 (-3)
+ 1010 (-6)

10111

Prenos do najvyššieho rádu bol *0,* prenos z najvyššieho rádu bol *1,* takže výsledok je neplatný (nastalo podtečenie). Súčet je *(-9)* a to je mimo intervalu zobrazenia.

Predpätý kód

Obraz čísla v *predpätom kóde* sa získa pripočítaním *posunutia* tak ku kladnému, ako aj k zápornému číslu. Potom je jednoduché porovnanie veľkosti dvoch čísel bez ohľadu na ich znamienko. Tento kód sa obyčajne používa na zobrazenie exponentov pri číslach zapísaných v *pohyblivej rádovej čiarke.* Napr. pri numerickom koprocesore rodiny *80x87* je pri *8-bitovom* exponente hodnota posunutia *127.*

Príklad: Máme 4-bitový register a nech posunutie = 8.

číslo 6 v predpätom kóde : 1110
číslo -6 v predpätom kóde : 0010

2.3.2.6 Reálne čísla

Reálne čísla je v zásade možné zobraziť dvoma spôsobmi, a to v *pevnej* a v *pohyblivej rádovej čiarke*.

Pri zobrazení v *pevnej rádovej čiarke* je istý počet bitov napevno určený na zobrazenie *celej časti* čísla a istý počet na zobrazenie *zlomkovej časti*. Toto usporiadanie je však pri veľkom rozsahu čísel veľmi nevýhodné, či už z hľadiska rozsahu zobrazenia, ako aj požadovanej presnosti. Z týchto dôvodov sa v súčasnosti používa takmer výhradne zobrazenie v pohyblivej rádovej čiarke.

Číslo *v pohyblivej rádovej čiarke* je zobrazené v tvare: $M \cdot z^E$

kde M je mantisa (obyčajne pravý zlomok),
z - základ použitej číselnej sústavy (obyčajne 2),
E - exponent (celé číslo).

Výhodou tohto zobrazenia reálnych čísel je možnosť dosiahnutia veľkého dynamického rozsahu zobrazenia, pričom je rovnaká presnosť nezávisle od veľkosti zobrazovaného čísla.

Pri súčasných procesoroch sa pritom ešte v závislosti od požadovanej presnosti a rozsahu zobrazenia používa niekoľko formátov pre čísla v pohyblivej rádovej čiarke. Tak napr. pri koprocesoroch rodiny *80x87* existujú tri formáty zobrazenia: *jednoduchá presnosť, dvojnásobná presnosť* a *rozšírená presnosť*.

V prvom prípade je vyhradených pre mantisu *24* a pre exponent *8* bitov, v druhom prípade 55 a *11* bitov a v treťom prípade dokonca *65* a *15* bitov.

Rozsah zobrazenia v treťom prípade predstavuje približne:

$$0.34 \times 10^{4932} < |x| < 1.1 \times 10^{4932}$$

Základné aritmetické operácie

Máme dve čísla *A* a *B,* zobrazené v pohyblivej rádovej čiarke v tvare:

$$A = MA \cdot z^{EA}$$
$$B = MB \cdot z^{EB}$$

$$A \cdot B = MA \cdot z^{EA} \cdot MB \cdot z^{EB} = MA \cdot MB \cdot z^{(EA + EB)}$$

$$A/B = (MA \cdot z^{EA}) / (MB \cdot z^{EB}) = (MA/MB) \cdot z^{(EA - EB)}$$

Aby sme mohli sčítať resp. odčítať mantisy pri operácii *A+B* resp. *A-B,* musíme najskôr upraviť zobrazenie čísel takým spôsobom, aby exponenty mali rovnakú hodnotu. Ak je mantisa pravý zlomok, je zrejmé, že exponenty musia mať hodnotu väčšieho.
V opačnom prípade by prišlo pri úprave k pretečeniu mantisy. Nech v nasledujúcich príkladoch je *EA* väčší ako *EB.*

$$A + B = MA \cdot z^{EA} + MB \cdot z^{EB} = MA \cdot z^{EA} + MB \cdot z^{(EB - EA)} =$$
$$= MA \cdot z^{EA} + MB \cdot z^{EA} \cdot z^{(EB - EA)} = (MA + MB \cdot z^{(EB - EA)}) \cdot z^{EA}$$

$$A - B = (MA - MB \cdot z^{(EB - EA)}) \cdot z^{EA}$$

Realizácia zložitejších matematických funkcií

Zložitejšie funkcie sa obyčajne realizujú *rozvojom funkcie do číselného radu,* v ktorom sa nachádzajú iba základné aritmetické operácie. Doba výpočtu závisí v značnej miere od požadovanej presnosti.

Príklad:
Funkcia sin(x):

$$\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \ldots + (-1)^n \cdot x^{(2n+1)}/(2n+1)! + \ldots$$

Príklad:

Funkcia ln(1 + x):

$$\ln(1+x) = x - x^2/2 + x^3/3 - x^4/4 + \ldots + (-1)^n . x^{(n+1)}/(n+1) +$$

$$x \in (-1, 1)$$

2.3.2.7 Desiatkové čísla

Desiatkové čísla predstavujú špeciálnu triedu celých čísel a často sa spracúvajú priamo, bez prevodu do dvojkovej číselnej sústavy. Z tohto dôvodu sa používajú aj špeciálne kódy na ich zobrazenie, a to *BCD (Binary Coded Decimal) kód* a *zhustený BCD kód.*

Pri zobrazení v *štandardnom BCD kóde* sa do registra dĺžky jednej slabiky *(8* bitov) zobrazí jedna desiatková číslica. Horné štyri bity sú nulové, v dolných štyroch bitoch je zakódovaná jedna číslica desiatkového čísla v prirodzenom dvojkovom kóde.

Príklad: Zobrazme číslo *1463* v BCD kóde.

Je nutné použiť štyri slabiky:

00000001 00000100 00000110 00000011

Pri zobrazení *v zhustenom BCD kóde* sú v jednej slabike zakódované dve desiatkové číslice.

Príklad: Zobrazme číslo *1463* v zhustenom BCD kóde.

Stačia nám dve slabiky:

00010100 01100011

Ak sa používajú aj záporné čísla, na znamienko je vyhradená obyčajne celá jedna slabika.

Základné aritmetické operácie

S číslami v BCD kóde je možné vykonávať operácie sčítania a odčítania rovnako, ako s prirodzenými číslami. Vzhľadom na to, že v každej štvorici bitov je však zobrazené iba číslo v rozsahu *0* až *9,* po vykonaní operácie je nutné v každej štvorici bitov vykonať tzv *desiatkovú korekciu.* Táto korekcia spočíva v pripočítaní čísla *6.* Začína sa vykonávať od najnižšej štvorice a vykoná sa v každej štvorici, v ktorej je zobrazené číslo väčšie ako *9.* Prenos do vyššieho rádu je nutné rešpektovať. Niektoré súčasné typy procesorov majú už priamo vo svojom inštrukčnom súbore inštrukciu pre deiatkovú korekciu (napr. procesory rodiny *80x86).*

Príklad:

Máme *16-bitový* register. V tomto registri je rozsah zobrazenia v zhustenom BCD kóde *0* až *9999.* Spočítajme nasledovné čísla:

```
   0001 0010 0011 0100     (1234)
+  0101 0110 0111 1000     (5678)

   0110 1000 1010 1100     výsledok pred korekciou
+             0110         prvá korekcia

   0110 1000 1011 0010
+        0110               druhá korekcia

   0110 1001 0001 0010     konečný výsledok, ďalšia korekcia už nie je nutná
                           (6912)
```

2.3.3 Znaky

Znakmi budeme v ďalšom rozumieť *číslice, písmená* a tzv. *špeciálne znaky,* napr. *?, !, §, CR, LF* atď. Všetky tieto znaky musia byť nejakým spôsobom zakódované, aby ich bolo možné

zobraziť v číslicovom počítači. *Kódovaním* rozumieme priradenie istého dvojkového čísla každému znaku. V súčasnosti sa používajú niektoré štandardné kódy, pri osobných počítačoch napr. kód *ASCII (American Standard Code for Information Interchange).* Tento kód je *7-bitový* (najvyšší bit je nulový) a obsahuje písmená latinskej abecedy, číslice a niektoré špeciálne znaky (obr. 7). Pretože však neobsahuje všetky znaky, ktoré tá-ktorá národná abeceda potrebuje, používajú sa tzv. *rozšírené (extended) ASCII kódy. Rozšírený ASCII kód* je *8-bitový,* obsahuje všetky znaky štandardného *ASCII* kódu, pridané vlastné špeciálne znaky (napr. písmená s diakritikou) majú najvyšší bit jednotkový.

	0000	0001	0010	0011	0100	0101	0110	0111
0000	NUL	DLE	SPACE	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O		o	DEL

Hodnotu znaku v *ASCII kóde* zistíme z tabuľky takým spôsobom, že spojíme jemu zodpovedajúce dvojkové číslo z prvého riadku (tým získame horné štyri bity) s dvojkovým číslom v prvom stĺpci (tým dostaneme spodné štyri bity).

Tak napríklad znaku + zodpovedá dvojkové číslo 00101011.

OBR. 7. ASCII kód

3 Základy číslicových systémov

3.1 Číslicový systém

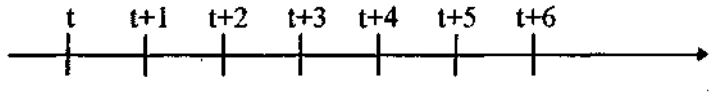
Číslicový systém chápeme ako *formálny model,* ktorý možno definovať nad *reálnym* (napr. elektronickým) *číslicovým zariadením.* Formálny model opisuje toto zariadenie pri danom stupni abstrakcie.

V ďalšom sa nebudeme zaoberať konkrétnou technickou realizáciou číslicových zariadení, t.j. nebudeme zdôrazňovať fyzikálne veličiny, ktorých hodnoty sa menia v spojitom čase a sú v skutočnosti nositeľom informácie. Premenné veličiny budeme prezentovať ako tzv. *číslicové premenné* a spracovanie informácie, ktoré prebieha v spojitom čase, budeme opisovať v tzv. *diskrétnom čase.* Samozrejme, je potrebné venovať neustálu pozornosť tomu, aby použitý model - číslicový systém - dostatočne presne odrážal vlastnosti reálneho zariadenia, fungujúceho v spojitom čase s reálnymi fyzikálnymi veličinami.

Číslicový systém je dynamický systém, ktorý možno charakterizovať týmto spôsobom:

- *Premenné sú číslicové,* čo znamená, že nadobúdajú hodnoty iba z konečných množín hodnôt.

- *Správanie sa systému* je definované v *diskrétnom čase,* ktorý je tvorený usporiadanou postupnosťou diskrétnych bodov na spojitej časovej osi (pozri obr. 8).



OBR. 8. Definícia diskrétneho času

Správanie sa číslicového systému možno opísať takto:

$$s(t+1) = f(s(t), x(t))$$
$$y(t) = g(s(t), x(t))$$

kde $s(t)$ značí stav v danom čase / (presnejšie v bode diskrétneho času t),
$s(t+1)$ značí stav v nasledujúcom bode diskrétneho času,
$x(t)$ značí vstupný vektor v čase t,
$y(t)$ značí výstupný vektor v čase /.

Symbolmi/a g sú označené dve funkcie, ktoré sa obyčajne nazývajú *prechodová* a *výstupná* funkcia systému.

Prechodová funkcia špecifikuje stav, v ktorom sa systém bude nachádzať v nasledujúcom bode diskrétneho času v závislosti od súčasného stavu a vstupného vektora.

Výstupná funkcia špecifikuje výstupný vektor v závislosti od stavu a vstupného vektora.

- *Diskrétny čas,* v ktorom sa "vzorkujú" (vyhodnocujú alebo pozorujú hodnoty premenných) je špecifikovaný pomocou určitých *diskrétnych časovacích udalostí,* ktoré sú predstavované zmenami hodnôt istých premenných číslicového systému. Z pohľadu funkcie číslicového systému nie sú hodnoty premenných mimo bodov diskrétneho času určujúce. Existujú dva základné spôsoby definície diskrétneho času - *synchrónny* a *asynchrónny*.

Synchrónny spôsob, ktorý sa používa pri *synchrónnych číslicových systémoch,* je charakterizovaný osobitnou logickou premennou *CLK,* ktorá má periodický charakter a typ zmeny jej hodnoty špecifikuje body diskrétneho času (napr. zmena z *0* na *1*). V systéme môže byť aj viacej synchronizačných premenných.

Asynchrónny spôsob, ktorý sa používa pri *asynchrónnych číslicových systémoch,* má definovaný diskrétny čas pomocou časovacích udalostí, ktorými sú ľubovoľné zmeny hodnôt ľubovoľnej zo vstupných a stavových premenných.

3.2 Boolovská algebra

Boolovská algebra je algebraický systém, ktorého *obor hodnôt je* množina *(0,1)* a *množina operácií* sa skladá z operácií *AND, OR* a *NEGÁCIA.* Tieto operácie sa tiež nazývajú *logické funkcie* a sú definované takto:

Majme jednoduché boolovské premenné *A, B, Y.*

Funkcia *AND* dvoch premenných, zapísaná do tabuľky:

A	B	Y = A . B
0	0	0
0	1	0
1	0	0
1	1	1

(namiesto *AND* budeme používať operátor .)

Funkcia *OR* dvoch premenných, zapísaná do tabuľky:

A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

(namiesto *OR* budeme používať operátor +)

Funkcia *NEGACIA* jednej premennej, zapísaná do tabuľky:

A Y = A# (NEGÁCIA je vyznačená symbolom # za premennou)

0	1
1	0

Logické funkcie predstavujú základné funkcie, pomocou ktorých sa vyjadrujú iné (napr. aritmetické) funkcie v číslicových systémoch.

3.3 Vektorové logické funkcie

Majme množinu *V* boolovských vektorov s rovnakou dlžkou *k.* Ak známe boolovské funkcie *(AND, OR, NEGÁCIA)* aplikujeme na jednotlivé bity vektorov (pri *NEGÁCII*) alebo na vzájomne zodpovedajúce dvojice bitov *(AND, OR),* dostaneme *boolovskú algebru boolovských vektorov.*

Vektorové logické funkcie sa v číslicových systémoch široko používajú. Vykonávajú sa Často aj nad vektormi, ktoré sa pri iných funkciách interpretujú ako iné údajové typy (napr. čísla, znaky atď.). Popri už definovaných vektorových funkciách možno zaviesť aj ďalšie, ako sú *exkluzívny logický súčet (EXOR,* symbol ©), *logická ekvivalencia* (symbol ≡), *negácia logického súčinu (NAND,* symbol ↑), *negácia logického súčtu (NOR,* symbol ↓) a pod.

3.4 Zápis logickej funkcie

Logické funkcie je možné zapísať viacerými spôsobmi, najčastejšie sa používa zápis pomocou *tabuľky* alebo *Karnaughovej mapy.*

Zápis logickej funkcie pomocou tabuľky

Tabuľka sa interpretuje po riadkoch. V každom riadku kombinácii vstupných premenných zodpovedá príslušná hodnota logickej funkcie.

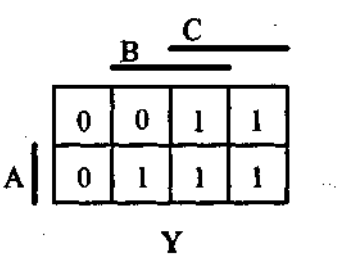
Funkcia $Y = A \,.\, B + C$ zapísaná pomocou tabuľky:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Zápis logickej funkcie do Karnaughovej mapy

Karnaughova mapa pre *n-premenných* obsahuje 2^n políčok, ktoré zodpovedajú všetkým kombináciám vstupných premenných. V každom políčku sa nachádza hodnota logickej funkcie pre danú kombináciu vstupných premenných. V riadku alebo stĺpci, ktorý je vyznačený pruhom, má príslušná vstupná premenná hodnotu *log. 1.*

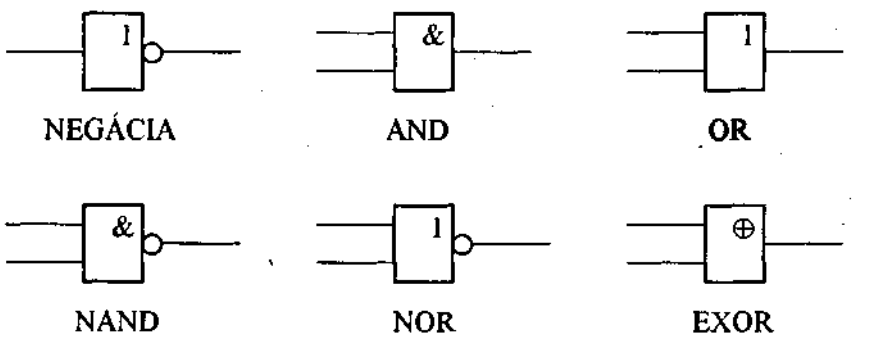
Na obr. 9 je funkcia Y = A . B + C zapísaná do Karnaughovej mapy.



OBR. 9. Zápis funkcie do Karnaughovej mapy

3.5 Logické členy

Logický člen je elementárny číslicový systém, ktorý realizuje niektorú boolovskú funkciu (operáciu) nad vstupnými premennými a jej výsledok poskytuje na svojom výstupe. Na obr. 10 sú uvedené logické členy, realizujúce funkcie *NEGÁCIA, AND, OR, NAND, NOR* a *EXOR*. Ide o typické prvky, ktoré sa nachádzajú v knižniciach (katalógoch) logických členov a implementujú sa ako elektronické obvody.



OBR. 10. Základné logické členy

3.6 Logické obvody

Logické obvody sú štruktúry, obsahujúce ako svoje prvky iba *logické členy* a prípadne *jednobitové pamäťové elementy*.

3.6.1 Kombinačné logické obvody

Kombinačné logické obvody sa vyznačujú tým, že výstupný vektor je jednoznačne daný vstupným vektorom. Ak globálnu vstupnú a výstupnú premennú označíme *X* resp. *Y,* tak pre takéto typy obvodov platí:

$$Y(t) = g(X(t)),$$

kde *g* je funkcia, ktorá každému vstupnému vektoru (hodnote premennej *X*) priradí určitý výstupný vektor (hodnotu premennej *Y*). *Výstupný vektor tu nezávisí od stavu obvodu, ale iba od vstupného vektora.*

Pri reálnej implementácii kombinačného obvodu treba pri realizácii funkcie *g* rátať s istým oneskorením v čase a možno písať:

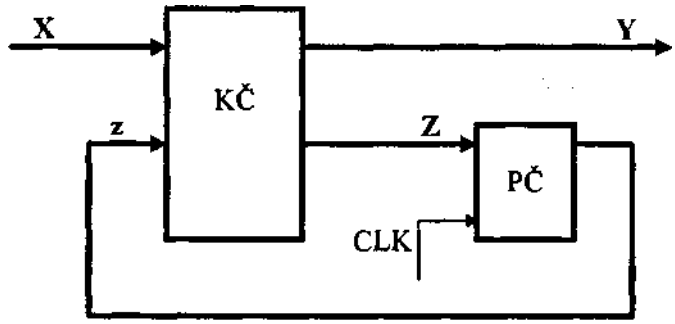
$$Y(t+d) = g(X(t)),$$

kde *t* je bod na osi spojitého času, v ktorom *X* zmení hodnotu a *d* je čas, po ktorom *Y* nadobudne príslušnú hodnou *g(X)*. Symboly *X(t)* a *Y(t+d)* zastupujú príslušné vstupné vektory v čase *t,* resp. *(t+d)*. Parameter *d* sa nazýva *oneskorením obvodu,*
Predpokladáme, že vstupný vektor je v intervale s dĺžkou *d* nemenný. Z toho vyplýva, že bezprostredne po sebe nasledujúce zmeny vstupných hodnôt (ktoré reprezentujú rýchlosť spracovania), sú možné v najlepšom prípade s periódou $> d$, teda najvyššia možná frekvencia zmien vstupov je $1/d$.

3.6.2 Sekvenčné logické obvody

Sekvenčné logické obvody sa vyznačujú tým, že ich správanie nemožno vyjadriť pomocou jednoduchej funkcie *g* ako pri kombinačnom obvode. Výstupný vektor tu závisí od stavu obvodu, t.j. výstupné vektory môžu byť rôzne pri tom istom vstupnom vektore. Závisí od stavu, v akom sa obvod nachádza. *Stav obvodu je daný hodnotami (vnútorných) stavových premenných.*

Správanie sa sekvenčného logického obvodu v definovanom diskrétnom čase opisujú prechodová a výstupná funkcia/ resp. *g,* ktoré boli uvedené pri opise správania sa číslicových systémov. Názov *sekvenčný* pochádza z toho, že výstupný vektor je závislý od sekvencie vstupných vektorov, t.j. nielen od jedného vstupného vektora, ale od predchádzajúcej postupnosti vstupných vektorov.



OBR. 11. Bloková schéma synchrónneho sekvenčného logického obvodu

Okamžitý stav obvodu je daný vektorom tzv. *stavových premenných* (vektor *z*), ktorý sa nachádza na výstupe pamäťovej časti *PČ*. V pamäťovej časti synchrónneho sekvenčného logického obvodu sú použité *synchrónne preklápacie obvody*. Kombinačná časť *KČ* realizuje tzv. *budiacu* a *výstupnú* funkciu.

Vektor Z je vektor *budiacich premenných,* vektor *Y* je vektor *výstupných premenných*. Vektor budiacich premenných sa v nasledujúcom bode diskrétneho času (ktorý je pri synchrónnom obvode daný zmenou synchronizačnej premennej *CLK*) stane vektorom stavových premenných.

Správanie sekvenčného obvodu sa často opisuje pomocou *konečného automatu*. Po *zakódovaní stavov* automatu je možné vyjadriť budiacu a výstupnú funkciu v tvare boolovských výrazov, ktoré sa budú realizovať zapojením logických členov v kombinačnej časti.

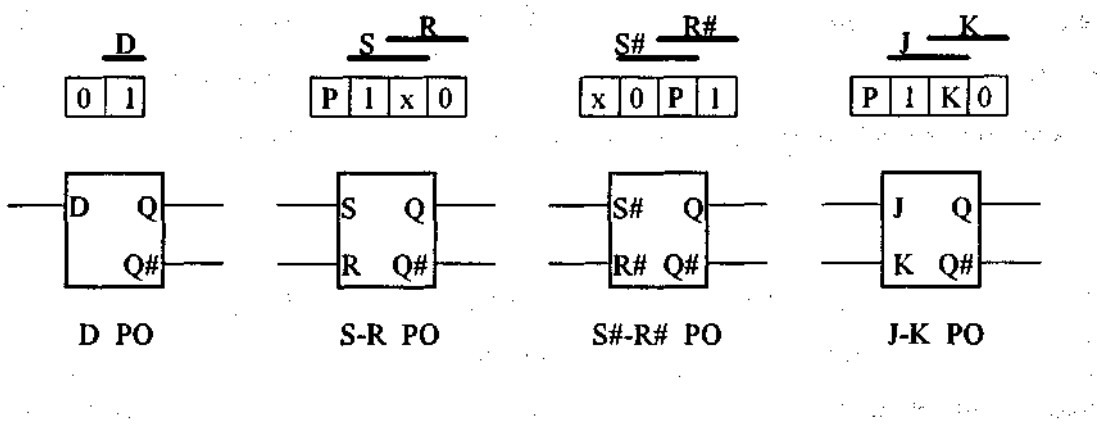
3.6.2.1 Preklápacie obvody

Preklápacie obvody (PO) sú jednobitové pamäťové elementy, ktoré na svojom výstupe uchovávajú logickú hodnotu *0* resp. *1*. Okrem použitia v pamäťovej časti sekvenčných obvodov sú *PO* základom realizácie *registrov* a *pamätí*. Poznáme *synchrónne* a *asynchrónne* preklápacie obvody.

Pri *synchrónnych preklápacích obvodoch* sa zmena stavu obvodu uskutoční pri zmene synchronizačnej premennej *CLK*.

Pri *asynchrónnych preklápacích obvodoch* sa zmena stavu obvodu uskutoční pri zmene niektorej riadiacej premennej.

Najpoužívanejšie preklápacie obvody sú *D* PO, *S-R* PO, *S#-R#* PO a *J-K* PO. Správanie sa týchto obvodov a schematické značky ich asynchrónnej verzie sú na obr. 12.



OBR. 12. Asynchrónne preklápacie obvody

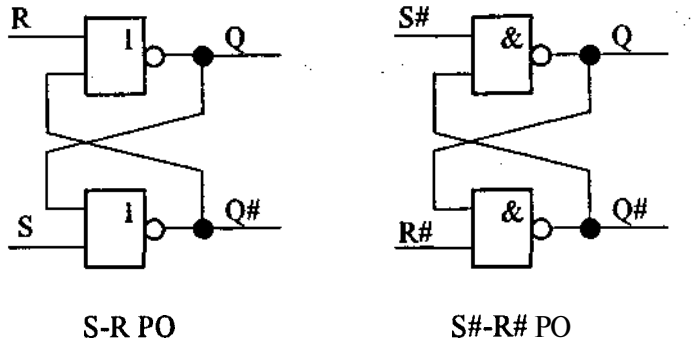
0 resp. *1* v príslušnom políčku Karnaughovej mapy znamená, že pri zodpovedajúcej kombinácii riadiacich vstupov výstup *Q* nadobudne hodnotu *log.0* resp. *log.1*.

P charakterizuje tzv. *pamäťové správanie sa obvodu,* t. j. výstup *Q* si uchová hodnotu, akú mal v predchádzajúcom bode diskrétneho času.

K charakterizuje tzv. *preklopenie obvodu,* t. j. výstup *Q* zmení svoju hodnotu na opačnú, akú mal v predchádzajúcom bode diskrétneho času.

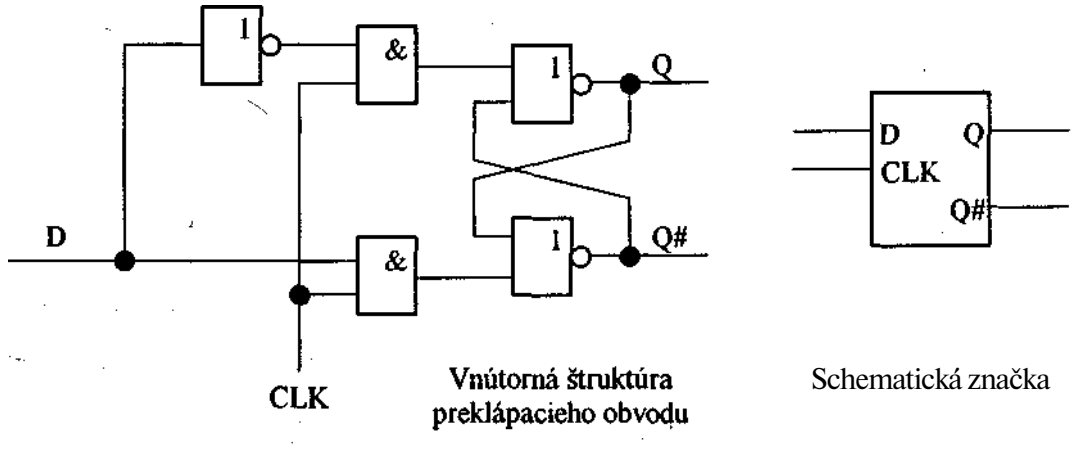
x charakterizuje *zakázanú kombináciu riadiacich vstupov,* kedy správanie sa obvodu nie je definované.

Príklad obvodovej realizácie asynchrónneho S-R a S#-R# PO je na obr. 13.



OBR. 13. Ralizácia asynchrónneho S-R a S#-R# PO

Uvedené typy preklápacích obvodov sa realizujú aj ako synchrónne. Na obr. 14 je naznačený spôsob realizácie synchrónneho *D* PO a jeho schematická značka.



OBR. 14. Synchrónny D PO

Synchrónne preklápacie obvody sa vyhotovujú ako *hladinové* alebo *citlivé na hranu*.

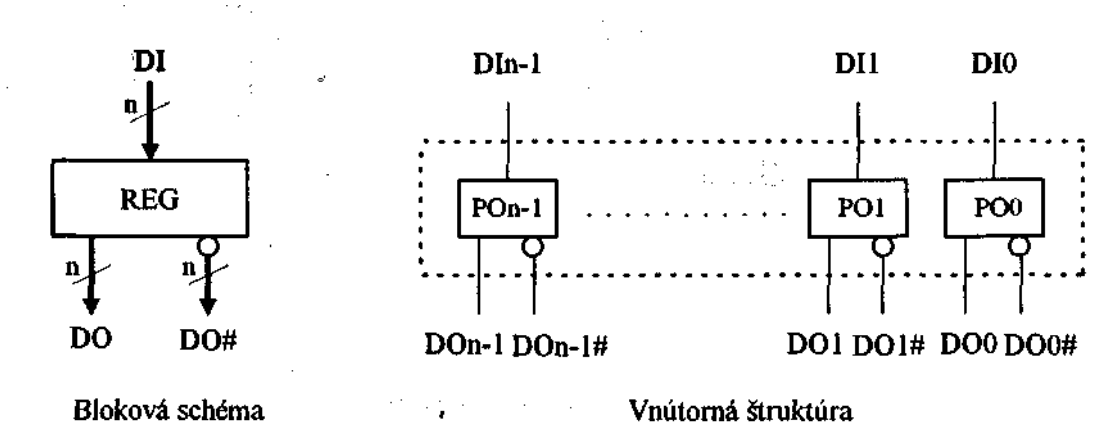
Pri *hladinovom obvode* môže dôjsť k zmene stavu obvodu počas trvania aktívnej úrovne synchronizačnej premennej.

Pri *obvode citlivom na hranu* môže dôjsť k zmene stavu obvodu iba v okamihu aktívnej hrany synchronizačného signálu. Obvod na obr. 14 je hladinový obvod.

3.6.2.2 Registre

Register je základný obvodový prostriedok na uchovávanie údajov. Je schopný uchovávať údaj takého typu, ktorý možno zobraziť prostredníctvom boolovského vektora. Register je realizovaný *n-ticou jednobitových pamäťových elementov* - preklápacích obvodov, preto má *okrem priameho výstupu* (výstupy Q_i obvodov) obyčajne vyvedený aj *komplementárny výstup* (výstupy $Q_i\#$), na ktorom je negovaná hodnota priameho výstupu.

Na obr. 15 je nakreslená bloková schéma a štruktúra *n-bitového* registra; *n-bitový vstup* je označený ako *DI*, *priamy n-bitový výstup* ako *DO*, *negovaný n-bitový výstup* ako *DO#*. *Dĺžka registra* zodpovedá počtu preklápacích obvodov v registri. Preklápací obvod predstavuje špeciálny prípad registra s jednotkovou dĺžkou.



OBR. 15. *n-bitový* register

Register má obyčajne aj určité *vstupné obvody,* ktoré slúžia na zápis údaja zo vstupu *DI* do registra. Zápis môže byť *riadený* alebo *neriadený,* pričom každý z nich môže byť *synchronizovaný* alebo *nesynchronizovaný.*

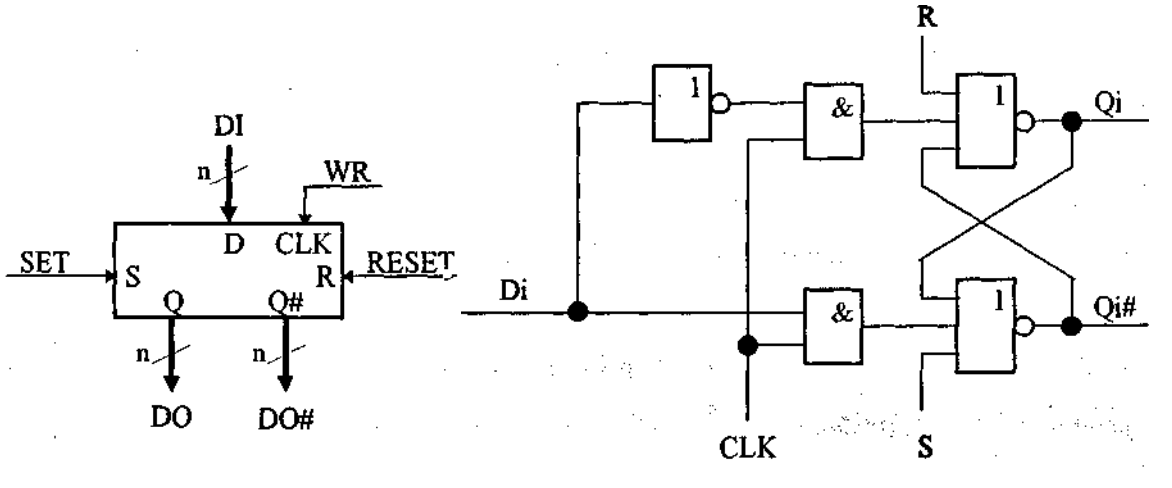
Neriadený nesynchronizovaný zápis sa používa v prípade, ak sú na realizáciu registra použité *asynchrónne* PO. Pri tomto zápise nie je použitá žiadna osobitná *zapisovacia* premenná ani synchronizačná premenná.

Riadený nesynchronizovaný zápis využíva osobitnú *riadiacu premennú WR* na zápis do registra. Zápis sa uskutoční pri aktívnej hodnote tejto premennej.

Neriadený synchronizovaný zápis sa vyznačuje tým, že zápis do registra sa uskutoční pri *každej zmene* synchronizačnej premennej *CLK,* ktorá definuje body diskrétneho času.

Riadený synchronizovaný zápis predstavuje kombinovaný prípad, pri ktorom sa zápis do registra uskutoční pri zmene synchronizačnej premennej, ale iba v tom prípade, ak má zapisovacia premenná aktívnu úroveň.

Okrem *vstupných zapisovacích obvodov* môže mať register aj *ďalšie obvody,* napr. *na posuv, milovanie registra, nastavenie (jednotkovanie) registra* atď.
Na obr. 16 je bloková schéma registra s riadeným nesynchronizovaným zápisom, nastavením a nulovaním. Ako pamäťové elementy sú použité synchrónne *D* PO z obr. 14, ktoré boli doplnené o možnosť *asynchrónneho nastavenia* (vstup *S*) a *asynchrónneho nulovania* (vstup



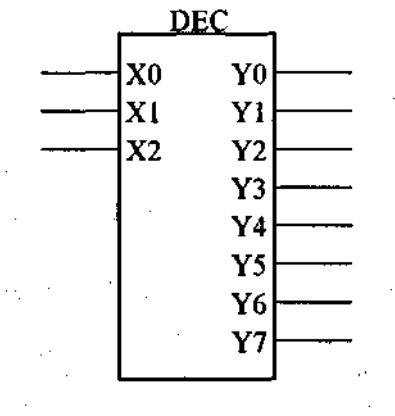
Bloková schéma Štruktúra jedného preklápacieho obvodu

OBR. 16. Register s riadeným nesynchronizovaným zápisom, nastavením a nulovaním

3.7 Ďalšie stavebné prvky číslicových systémov

3.7.1 Dekóder

Dekóder vo všeobecnosti slúži na dekódovanie vstupnej informácie, t.j. na základe nastaveného vstupného boolovského vektora nastaví zodpovedajúci výstupný vektor. Pre naše účely si preberieme špeciálny prípad dekódera, a to dekóder *1 z n*. Schéma dekódera pre prípad *1 z 8* je na obr. 17.



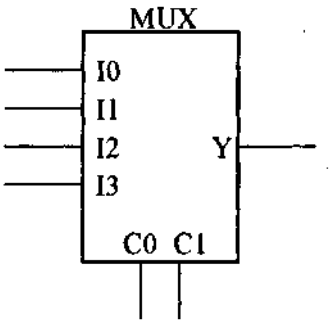
OBR. 17. Dekóder *J z 8*

Dekóder *J z n má k* vstupov a $n=2^k$ výstupov. Ak je na vstupe pripojený vektor *X,* ktorého hodnota v prirodzenom dvojkovom kóde je *i,* potom je výstup *Yi* aktívny.

3.7.2 Multiplexor

Multiplexor je riadený prepínač, ktorý na základe riadiacej informácie prepína niektorý zo vstupných kanálov na jeden výstupný kanál.
Ak je na riadiace vstupy multiplexora C pripojený vektor, ktorého hodnota v prirodzenom dvojkovom kóde je *i,* potom je na výstup *Y* pripojený vstupný kanál /,.

Na obr. 18 je nakreslený multiplexor *4 na 1 (4* vstupné jednobitové kanály, *1* výstupný jednobitový kanál). Riadiaci vstup C preto musí byť dvojbitový.



OBR. 18. Multiplexor *4 na 1*

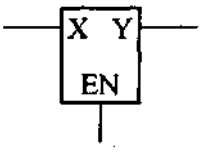
3.7.3 **Demultiplexor**

Demultiplexor je riadený prepínač, ktorý na základe riadiacej informácie prepína jeden vstupný kanál na jeden z niekoľkých výstupných kanálov. V explicitnej forme sa tento stavebný prvok vyskytuje menej často, obyčajne sa používa *demultiplexovanie v skrytej forme,* pri ktorej je priamo viditeľné vlastné rozvetvenie a vlastné demultiplexovanie sa realizuje v príslušnom štruktúrnom prvku. Takýto spôsob demultiplexovania si ukážeme pri realizácii operačných častí procesorov.

3.7.4 Hradlo

Hradlo je riadený spínač. Na základe riadiacej informácie buď prepája vstup na výstup (je *uvoľnené* resp. *priepustné),* alebo informáciu zo vstupu na výstup neprenáša (hradlo je *zablokované* resp. *nepriepustné*). Samozrejme, pri konkrétnej fyzickej realizácii musí byť na výstupe obvodu vždy nejaká hodnota fyzikálnej veličiny. Preto v prípade, ak je hradlo zablokované, sa jeho výstup uvedie do tzv. *tretieho* stavu (stavu *vysokej impedancie).* V tomto stave má výstup vlastnosť vstupu s vysokou impedanciou. Okrem *jednosmerných hradiel* sa používajú aj *obojsmerné hradlá,* kde vstup môže byť v inom okamihu použitý ako výstup a naopak. Hradlá sa využívajú najmä pri *zbernicovej architektúre.*

Na obr. 19 je *nakreslené jednosmerné trojstavové hradlo*. Aktívna úroveň signálu *EN* (t.j. ak *EN=1)* spôsobí, že signál zo vstupu *X* sa prenáša na výstup *Y*. Ak *EN=0,* hradlo je zablokované - výstup *Y* je v stave vysokej impedancie. Na hodnote na vstupe *X* teraz nezáleží.



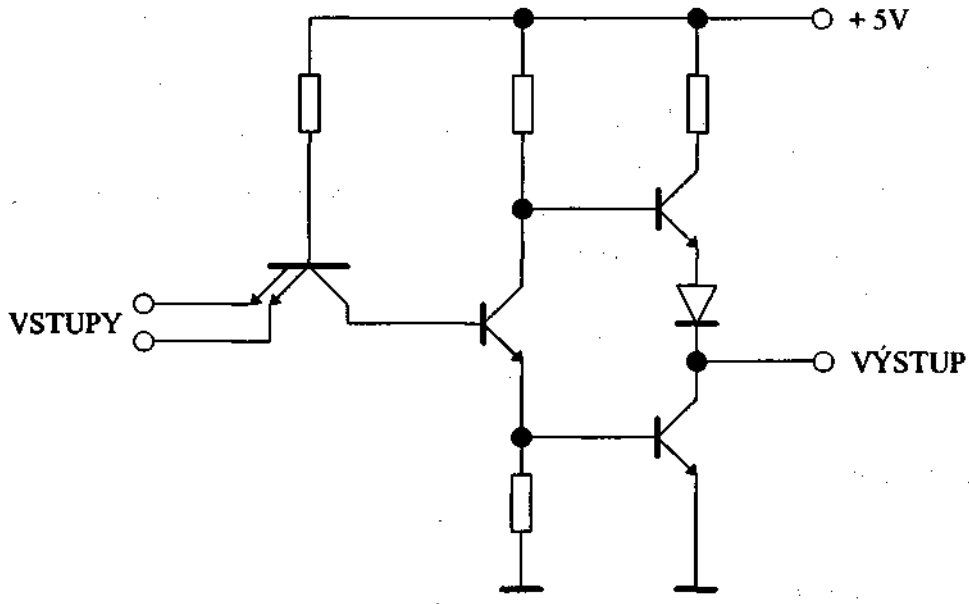
OBR. 19. Jednosmerné trojstavové hradlo

3.8 Fyzická realizácia stavebných prvkov číslicových systémov

Základné logické členy elektronických číslicových systémov sú fyzicky realizované na čipe integrovaného obvodu. Napr. integrovaný obvod typu *7400* obsahuje v jednom puzdre *4* dvojvstupové logické členy *NAND.* Samozrejme, súčasné technológie výroby integrovaných obvodov umožňujú na čip integrovaného obvodu umiestniť aj zložitejšie prvky, napr. multiplexory, dekódery, registre, sčítačky atď. Na jednom čipe sú realizované aj najzložitejšie stavebné prvky číslicových počítačov, procesory a pamäti.

Už sme hovorili o tom, že nositeľom informácie v elektronických číslicových počítačoch je elektrické napätie. Najčastejšie sa používa *pozitívna logika s úrovňami TTL,* kde úrovni *H* zodpovedá napätie *+5 V a* úrovni *L* napätie *0 V.* Toto je samozrejme ideálny prípad, vplyvom rôznych okolností je tieto hodnoty nemožné dodržať. Preto je jednotlivým logickým úrovniam priradená nie jediná hodnota napätia, ale *napätie v istom rozsahu,* napr. obvody vyhodnocujú napätie v intervale *<0 V; 0.8 V>* ako úroveň *L* a napätie v intervale *<2 V; 5 V>* ako úroveň *H.*

Integrované obvody sa v súčasnosti vyrábajú rôznymi technológiami, napr. štandardná technológia *TTL, Schottky TTL (S-TTL), Low Power S-TTL (LS-TTL), HC, HCT* atď. Pre predstavu uvedieme elektrickú schému dvojvstupového logického člena *NAND,* realizovaného technológiou *TTL.*



OBR. 20. Dvojvstupový NAND v technológii TTL

4 Počítače s jedným prúdom inštrukcií a jedným prúdom údajov

V tejto kapitole sa budeme zaoberať von Neumannovským počítačom. Podrobne opíšeme jeho jednotlivé podsystémy a ich vzájomné vzťahy.

4.1 Prepojovací podsystém počítača

Prepojovací podsystém počítača slúži na prepojenie jednotlivých častí počítača, t.j. umožňuje procesoru načítanie inštrukcií a údajov z pamäte, zápis výsledkov do pamäte, načítanie údajov zo vstupného zariadenia a výstup výsledkov prostredníctvom výstupného zariadenia. Okrem toho môže umožňovať aj priamy prenos údajov medzi pamäťou a vstupným, resp. výstupným zariadením.

Spôsoby prepojenia jednotlivých častí počítača sú v zásade dva - prepojenie pomocou *prepojovacích kanálov* a prepojenie prostredníctvom *zbernice*.

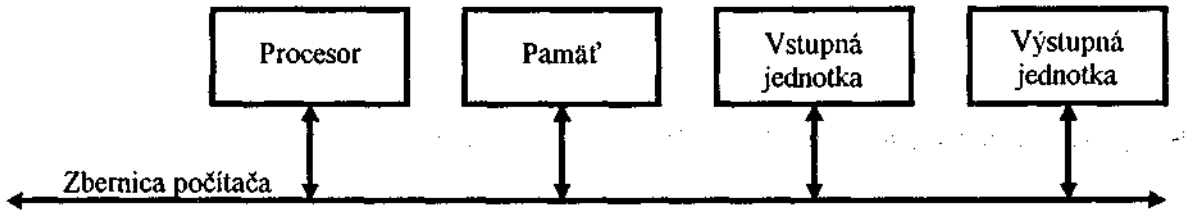
Prepojovací kanál

Prepojovací kanál je špecializované spojenie medzi dvoma blokmi počítača, napr. počítač z obr. 1 má *tri* prepojovacie kanály - prvý slúži na spojenie procesora s pamäťou, druhý na spojenie procesora so vstupným a tretí na spojenie s výstupným zariadením. Kanály pre priamy prenos údajov medzi pamäťou a vstupným, resp. výstupným zariadením tohto konkrétneho počítača nie sú realizované. Kanál predstavuje spojenie dvoch zariadení, kde je jednoznačne určené, ktoré z nich bude riadiť prenos. V našom prípade je týmto prvkom procesor. Vzhľadom na komplikovanosť prepojovacieho systému pri takomto riešení (pripojenie každého ďalšieho zariadenia si vyžaduje realizovať nový kompletný kanál) sa toto riešenie v súčasnosti používa len v špeciálnych prípadoch. V ďalšom sa preto budeme podrobnejšie venovať *zbernicovému prepojovaciemu systému*.

4.1.1 Zbernica

Zbernica je množina liniek (vodičov), ktorá navzájom prepája *všetky prvky na danej štruktúrnej úrovni.* Umožňuje spojenie *každého s každým,* ale v danom okamihu môže údaje na zbernicu vysielať iba *jediné zariadenie.* Ostatné zariadenia, schopné vysielať údaje na zbernicu, musia byť vtedy od zbernice odpojené. Reálne sa to uskutoční takým spôsobom, že uvedú svoje výstupy do stavu vysokej impedancie (pozri časť 3.7.4).

Zbernica počítača navzájom prepája procesor, pamäť, vstupné a výstupné zariadenia. Na obr. 21 je nakreslený počítač so zbernicovou architektúrou.



OBR. 21. Počítač so zbernicovou architektúrou

4.1.1.1 Rozdelenie zberníc

Podľa spôsobu riadenia:

- *Zbernice typu SINGLE-MASTER* - v systéme sa nachádza iba jeden prvok *(zariadenie),* ktorý môže pracovať ako *nadriadený (MASTER),* t.j. môže riadiť zbernicu (v prípade počítačovej zbernice typu SINGLE-MASTER je nadriadeným procesor; pamäť a V/V zariadenia sú *podriadenými).*

- *Zbernice typuMULTI-MASTER* - na zbernicu je pripojených viacero zariadení, z ktorých každé môže riadiť zbernicu. V danom okamihu je však zbernica riadená iba jedným zariadením. Pri zbernici typu MULTI-MASTER je potrebné riešiť problém *prideľovania zbernice* pri viacerých súčasných požiadavkách.

Podľa synchronizácie prenosu:

- *Synchrónne zbernice* - prenos je synchronizovaný spoločným hodinovým signálom.

- *Asynchrónne zbernice* - prenos je synchronizovaný odpoveďou podriadeného zariadenia.

Synchrónne zbernice sú rýchlejšie v dôsledku prísnejších časových pravidiel a používajú sa na spojenie staníc s rovnakou prenosovou rýchlosťou. Asynchrónne zbernice sú pomalšie, pretože sa čaká na potvrdenie prenosu každého údaja. Sú však vhodné na spojenie zariadení s rôznou prenosovou rýchlosťou.

Podľa tvaru prenášaných údajov:

- *Paralelné zbernice* - v jednom *cykle zbernice* sa naraz prenáša viacbitové slovo (obyčajne slabika, prípadne *16, 32, 64,...* bitov).

- *Sériové zbernice* - údaje sa prenášajú v sériovom tvare, t.j. bit po bite.

Podľa časového multiplexu:

- *Multiplexované zbernice* - význam informácie, prenášanej po zbernici, sa mení s časom - v jednom časovom okamihu prenáša zbernica (alebo iba jej časť) jeden typ informácie (napr. *adresu),* v inom okamihu sa prenáša iný typ informácie (napr. *údaje).* V prípade takejto zbernice musia byť k dispozícii signály, ktoré *rozlišujú,* ktorý typ informácie sa po zbernici (časti zbernice) práve prenáša. Obyčajne sa multiplexuje adresová/údajová sekcia zbernice.

- *Nemultiplexované zbernice* - význam signálov, prenášaných po zbernici, sa s časom nemení.

4.1.1.2 Štruktúra typickej počítačovej zbernice

Často používaným typom súčasných počítačových zberníc je *paralelná asynchrónna zbernica*. Skladá sa z *adresovej, údajovej* a *riadiacej* sekcie.

Po *adresovej sekcii* (tiež sa nazýva *adresová zbernica)* sa prenášajú *adresy,* ktoré sú generované nadriadeným prvkom zbernice, t.j. procesorom (alebo *riadiacim obvodom DMA,* pozri časť 4.4.5). Adresa identifikuje bunku v hlavnej pamäti (resp. V/V zariadenie), s ktorou (ktorým) sa bude pracovať.

Po *údajovej sekcii (údajovej zbernici)* sa prenášajú *inštrukcie* (z pamäte do procesora) a *údaje* (medzi procesorom a pamäťou alebo V/V zariadením, prípadne medzi pamäťou a V/V zariadením, pozri časť 4.4.5).

Signály riadiacej sekcie (riadiacej zbernice) sa skladajú z *povelov,* generovaných nadriadeným zbernice (napr. signál *čítania* alebo *zápisu)* a zo *žiadostí,* ktorými sa podriadení obracajú na nadriadeného (napr. žiadosť o predĺženie *cyklu čítania* alebo *zápisu, žiadosť o prerušenie* atď.).

Prenosová rýchlosť zbernice udáva množstvo údajov, ktoré je možné preniesť po zbernici za jednotku času. Obyčajne sa udáva v $MB.s^{-1}$.

V jednom *cykle zbernice* sa vykoná prenos jediného údaja po zbernici (napr. ak je šírka údajovej zbernice *8 bitov,* v jednom cykle zbernice sa prenesie *jedna slabika).*

Pracovná frekvencia zbernice (v *Hz)* je prevrátená hodnota doby trvania jedného cyklu zbernice (v *s).*

Pod *signálovým sledom* rozumieme časové priebehy signálov zbernice, ktoré je potrebné dodržať, aby sa korektne uskutočnil cyklus zbernice. V časti 4.3 a **4.4** uvedieme konkrétne signálové sledy pri prenose údajov po zbernici pri práci s pamäťou resp. V/V zariadením.

Príklad:

Pracovná frekvencia zbernice je *10 MHz* a šírka jej údajovej sekcie je *32 bitov*. Aká je prenosová rýchlosť zbernice ?

Doba trvania jedného cyklu zbernice je $1/(10*10^6\ Hz) = 0{,}0000001\ s = 0{,}1\ \mu s$.

32 bitov sú *4 bajty,* takže prenosová rýchlosť je *4 bajty* za $0{,}1\ \mu s$. V prepočte to predstavuje $40\ MB.s^{-1}$.

4.2 Základná koncepcia procesora

Procesor je jedna zo základných častí počítača. Jeho základná funkcia je *interpretovať inštrukcie programu,* ktorý je uložený v hlavnej pamäti. Pri tejto činnosti sa jednotlivé inštrukcie vyberajú z hlavnej pamäte, vykonávajú sa požadované operácie s operandmi, špecifikovanými v inštrukciách a uskutočňujú sa príslušné prenosy informácií medzi jednotlivými časťami číslicového počítača.

Podľa druhu vykonávanej funkcie rozlišujeme *univerzálne* procesory a *problémovo orientované* procesory.

Univerzálne procesory

Univerzálne procesory realizujú hlavné číselné a nečíselné operácie a zabezpečujú riadenie ostatných častí počítača na základe programu, uloženého v pamäti počítača. Tieto procesory majú vo všeobecnosti inštrukčný súbor, ktorý je z hľadiska spracovania informácie úplný, a preto sa môžu použiť na riešenie úloh ľubovoľného typu. Samozrejme, úlohy istého typu riešia efektívnejšie, úlohy iného typu menej efektívne.

Problémovo orientované procesory

Problémovo orientované procesory vykonávajú v počítači *špecializované funkcie,* pričom sa používajú problémovo orientované programové prostriedky. Sú určené na riešenie úloh

istého typu. Príkladom takýchto procesorov je *vstupno/výstupný procesor, numerický* (aritmetický) *koprocesor, grafický procesor, bitový procesor* atď.

V ďalšom sa budeme zaoberať univerzálnymi procesormi a pod slovom *procesor* budeme rozumieť *univerzálny procesor*.

Hlavné časti procesora:

Procesor sa skladá z dvoch hlavných častí - *operačnej* časti a *riadiacej* časti.

- *Operačná časť vykonáva operácie s operandmi* na základe *povelov* z riadiacej časti. O výsledkoch operácií informuje riadiacu časť prostredníctvom *príznakov*.

- *Riadiaca časť vyberá inštrukcie z pamäte, dekóduje* ich a zabezpečuje ich *vykonanie*. Riadi *spoluprácu procesora s okolím* (komunikácia s pamäťou a vstupno/výstupnými zariadeniami, *obsluha prerušenia* atď.).

4.2.1 Operačná časť procesora

Operačná časť procesora sa skladá z *aritmeticko-logickej jednotky, registrov* a *komunikačných obvodov*.

- *Aritmeticko-logická jednotka (ALU - Arithmetic and Logic Unit)* je určená na bezprostredné vykonanie požadovaných aritmetických, logických a iných operácií s operandmi.

- *Registre (zápisníková pamäť)* slúžia na prechodné uloženie operandov, vstupujúcich do týchto operáciií, ako aj na uloženie výsledku.

- *Komunikačné obvody* umožňujú vykonávanie *medziregistrových prenosov*.

4.2.1.1 Aritmeticko-logická jednotka

Základnou úlohou *ALU* je vykonanie elementárnych *aritmetických* a *logických* operácií, na základe ktorých je možné realizovať ľubovoľné, algoritmicky definované spracovanie *číselných* a *nečíselných* údajov. Okrem týchto elementárnych operácií *ALU* realizuje aj ďalšie typické operácie *(posuvy, predikáty,* atď.).

Realizácia základných aritmetických operácií

Základ *ALU* na úrovni logických obvodov na realizáciu základných aritmetických operácií predstavuje obyčajne *paralelná dvojková sčítačka.* Na jej vstupy sú pripojené *n-bitové* operandy, sčítačka na výstupe poskytuje *n-bitový výsledok* a *výsledný prenos.*

Majme dva ***n-bitové*** operandy *A, B,* zapísané v tvare: $A = A_{n-1}A_{n-2}...A_0$

$B = B_{n-1}B_{n-2}...B_0$

Na základe skôr uvedených pravidiel sčítania jednobitových čísel pre *i-ty* rád platí:

$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = A_i . B_i + A_i . C_i + B_i . C,$$

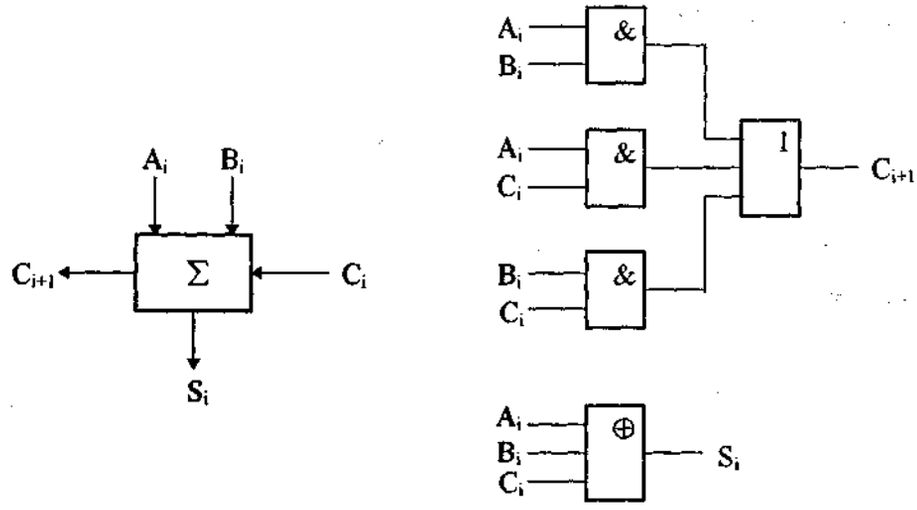
kde: A_i, B_i sú *i-te* rády čísel *A* resp. *B,*

C_i - prenos z predchádzajúceho rádu,

S_i - súčet v *i-tom* ráde,

C_{i+1} - prenos generovaný do nasledujúceho rádu.

Na nasledujúcom obrázku je nakreslená bloková a logická schéma *i-teho* rádu paralelnej dvojkovej sčítačky.



Bloková schéma Vnútorná štruktúra

OBR. 22. *i-ty* rád paralelnej dvojkovej sčítačky

n-bitová sčítačka sa vytvorí jednoducho zreťazením požadovaného počtu uvedených buniek. Je zrejmé, že pri uvedenej realizácii paralelnej sčítačky je nevyhnutné čakať na výsledok, ktorý dostaneme až po prešírení prenosu z najnižšieho do najvyššieho rádu. Na elimináciu tejto skutočnosti boli vytvorené *sčítačky so zrýchleným prenosom*. n-bitovú paralelnú sčítačku je tiež možné jednoducho realizovať pomocou pamäte typu *ROM*.

Okrem paralelných dvojkových sčítačiek existujú aj *sériové sčítačky,* kde sa sčítanie realizuje postupne, spočítavaním zodpovedajúcich bitov operandov a hodnoty prenosu z predchádzajúceho rádu. Spočítavame začína od najnižšieho rádu, kedy je vstupný prenos nulový. V prípade sériovej sčítačky je nutné uchovať v každom kroku hodnotu výsledku a hodnotu prenosu, ktorá vstúpi do operácie v ďalšom kroku.

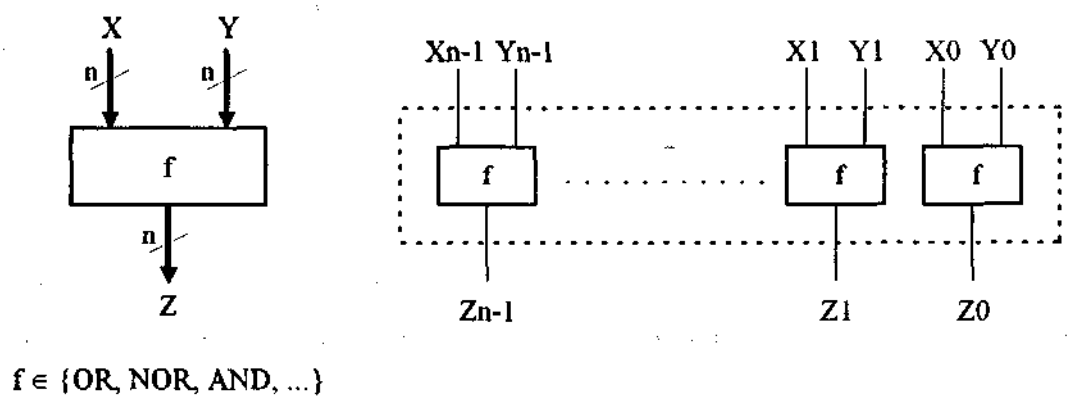
Ukázali sme si, že dvojkovú sčítačku je okrem sčítania možné použiť aj na odčítanie (pozri časť 2.3.2.5). Okrem toho je aj základnou časťou zložitejších obvodov na realizáciu násobenia a delenia.

Realizácia ďalších typických operácií

Ukážeme si, akým spôsobom sa realizujú *logické operácie, posuvy* a *predikáty.*

Logické **operácie**

Na obr. 23 je nakreslená funkčná jednotka na realizáciu niektorej vektorovej logickej funkcie /nad dvoma *n-bitovými* vektormi *X* a *Y.*

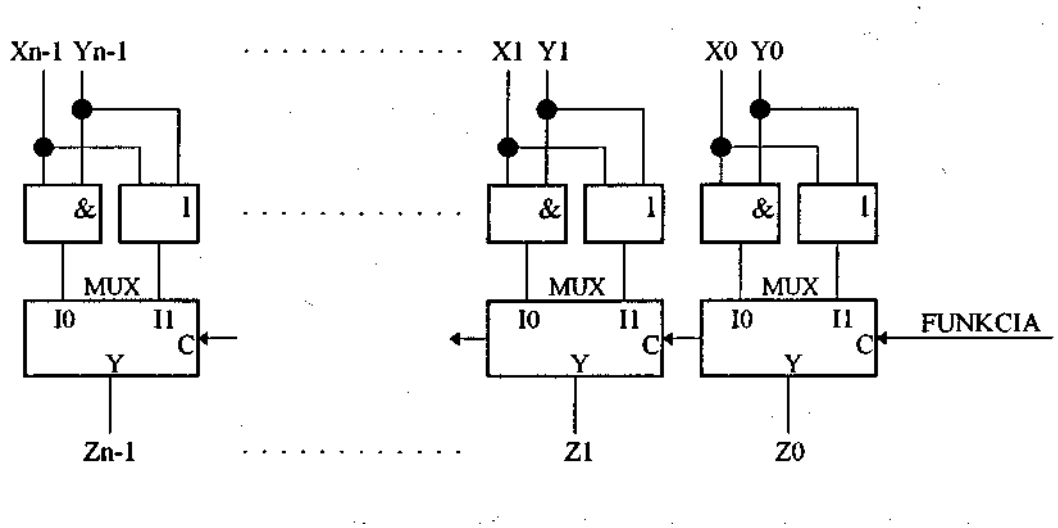


Bloková schéma Vnútorná štruktúra

OBR. 23. Funkčná jednotka na realizáciu vektorovej logickej funkcie

Funkčná jednotka môže byť vyhotovená aj ako *viacfunkčná*, t.j. na základe nastavených *riadiacich premenných* sa realizuje vybraná funkcia. Na obr. 24 je nakreslená logická schéma *dvojfunkčnej* jednotky na realizáciu funkcií *AND* a *OR.*

MUX označuje multiplexor *2 na 1.* Riadiace vstupy všetkých multiplexorov sú navzájom prepojené a sú ovládané premennou *FUNKCIA.*



OBR. 24. Funkčná jednotka pre realizáciu vektorových funkcií AND a OR

Ak *FUNKCIA= log.0,* funkčná jednotka bude realizovať nad vektormi *X* a *Y* logickú funkciu *AND.*

Ak *FUNKCIA=log. 1,* bude realizovaná funkcia *OR.*

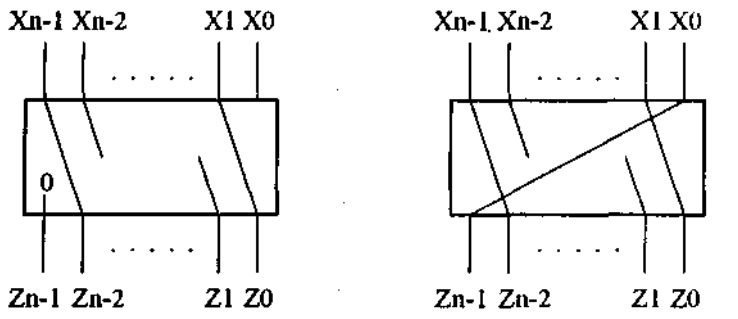
Posuvy

Pri vykonávaní niektorých aritmetických funkcií, ale aj pri iných funkciách sa vyžaduje realizovať *posuvy boolovských vektorov.* Sú to napr. operácie násobenia alebo delenia, násobenie resp. delenie mocninami 2^n atď.

Posuvy sa realizujú *posúvacím obvodom.* Je to kombinačný logický obvod, ktorý realizuje jeden alebo viac typov posuvov. Má *n* vstupov pre operand a *n* výstupov pre výsledok *(n>l).*

Najčastejšie sa používajú tieto *pravé* a *ľavé posuvy* o *n* miest : *logický posuv, kruhový (cyklický) posuv* a *aritmetický posuv.* Kruhový posuv sa často nazýva *rotácia.*

Na obr. 25 sú nakreslené posúvacie obvody na realizáciu logického a kruhového posuvu o jedno miesto vpravo.

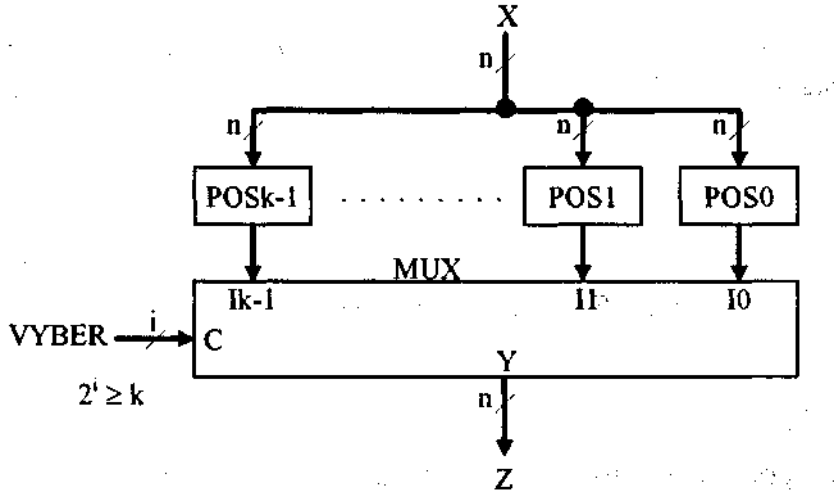


Logický posuv Kruhový posuv

OBR. 25. Posúvacie obvody na realizáciu logického a kruhového posuvu o jedno miesto vpravo

Aj posúvacie obvody sa často vyhotovujú ako viacfunkčné. Potom okrem vstupov pre operand a výstupov pre výsledok obsahujú aj *riadiace vstupy,* na základe ktorých sa realizuje vybraný druh posuvu.

Na obr. 26 je bloková schéma viacfunkčného posúvacieho obvodu.



OBR. 26. Viacfunkčný posúvací obvod

POS označuje posúvacie obvody pre jednotlivé typy posuvov.

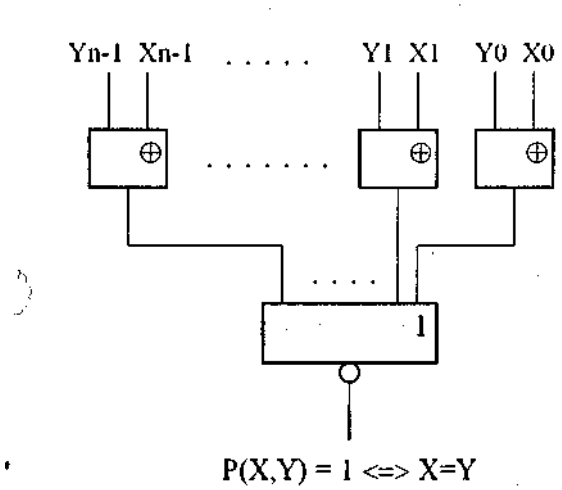
MUX označuje *n-bitový* multiplexor *k na 1,* takže na výstup multiplexora je vždy prenášaný iba výstup z jedného posúvacieho obvodu pre vybraný typ posuvu. Typ posuvu je vybraný vektorovou riadiacou premennou *VYBER.*

Muitiplexor by nebolo potrebné použiť v prípade, ak by sa posúvacie obvody realizovali ako obvody s trojstavovým výstupom.

Predikáty

Predikáty (príznaky) definované nad číslicovými premennými rozličných údajových typov sú *dvojhodnotové funkcie.* Najčastejšie sa implementujú kombinačnými logickými obvodmi. Výstupná boolovská premenná takéhoto obvodu je potom priamym nositeľom hodnoty predikátu. Predikáty sa realizujú nad *jedným operandom* (napr. *rovný nule, väčší ako nula, menší ako nula* atď.), ako aj nad *dvoma* príp. *viacerými operandmi {rovnajú sa, prvý je väčší ako druhý atď.).*

Na obr. 27 je logická schéma obvodu, ktorý realizuje predikát $P(X, Y) = 1 <=> X = Y,$
kde X a Y sú boolovské vektory.



OBR. 27. Predikačný obvod

Predikáty sa v procesore pre ďalšie spracovanie (vyhodnotenie) uchovávajú v špeciálnom registri, ktorý sa nazýva *príznakový (FLAG) register.*

4.2.1.2 Komunikačné obvody

Komunikačné obvody prepájajú jednotlivé štruktúrne prvky operačnej časti a umožňujú realizovať *medziregistrové prenosy.* V zásade sa používajú dva spôsoby realizácie prepojovacích obvodov - prvý je použitie *multiplexorov a demultiplexorov,* druhý spôsob je použitie *zbernice.*

Multiplexory a demultiplexory

Pomocou multiplexorov a demultiplexorov možno v štruktúre operačnej časti vytvoriť *ľubovoľné riadené údajové cesty.*

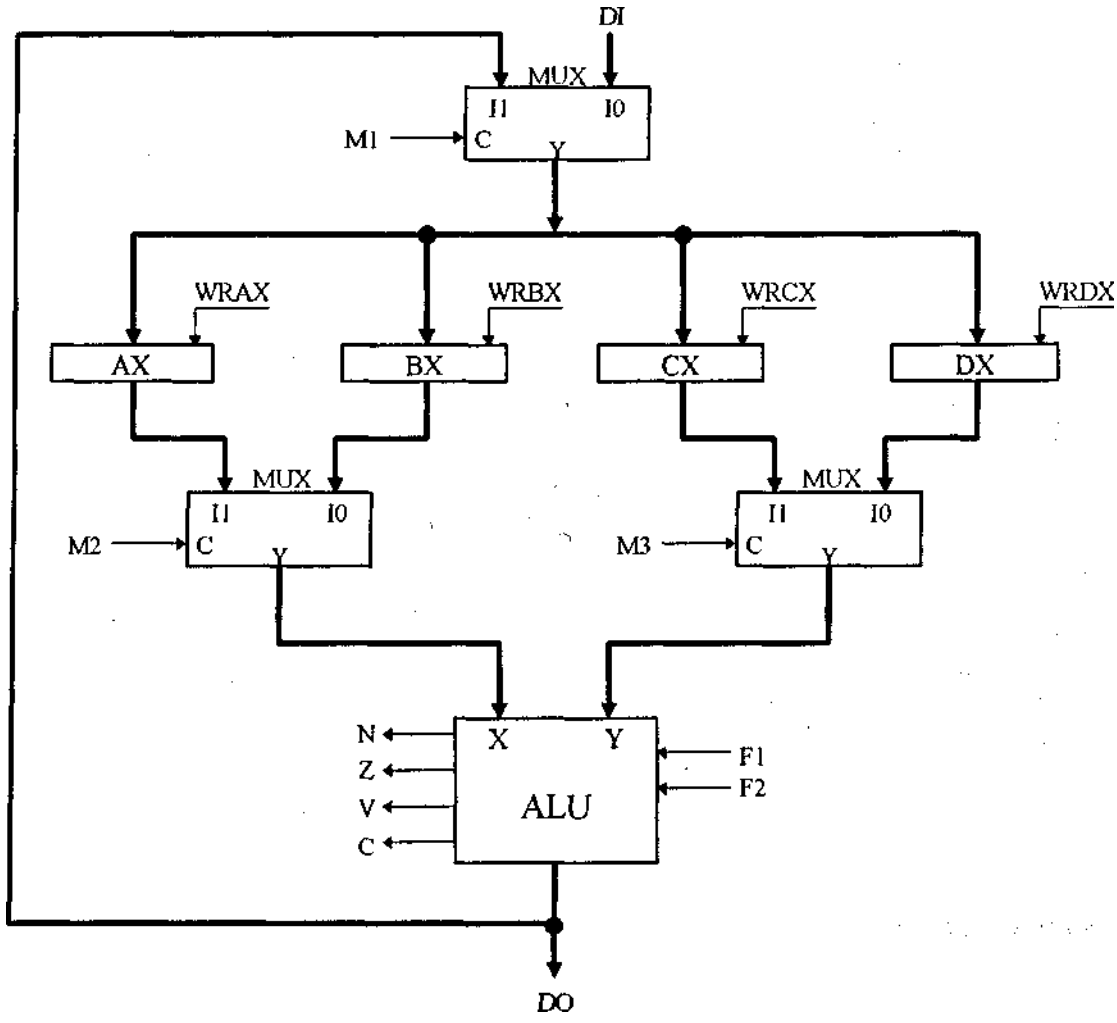
Na obr. 28 je príklad prepojenia registrov s aritmeticko-logickou jednotkou pomocou multiplexorov a demultiplexorov. Na vstup *X ALU* možno pripojiť register *AX* alebo *BX,* na vstup *Y* register *CX* alebo *DX.* Údaj z výstupu *ALU sa.* môže zapísať do ľubovoľného registra. Registre majú riadený zápis (riadiace vstupy *WRiX),* takže rozvetvenie sa realizuje *implicitne* pomocou zapisovacej logiky registrov *{skryté demultiplexovanie,* pozri časť 3.7.3).

DI predstavuje vstupný údajový kanál, *DO* výstupný údajový kanál.
F1 a *F2* sú riadiace vstupy *ALU,* ktoré nastavujú príslušnú požadovanú funkciu.
N, Z, V a *C* sú predikáty, ktoré informujú o výsledkoch vykonania oprácie v *ALU (Negatíve - príznak záporného výsledku, Zero - nulový výsledok, oVerflow -pretečenie, Carry - prenos z najvyššieho rádu).*

Zbernica

Zbernica je často používaný systém prepojenia v operačných častiach procesorov. Prepája navzájom všetky štruktúrne prvky, pričom v danom čase je možné prenášať údaj z jedného východiskového prvku do jedného (alebo niekoľkých) cieľových prvkov. Aby nebolo

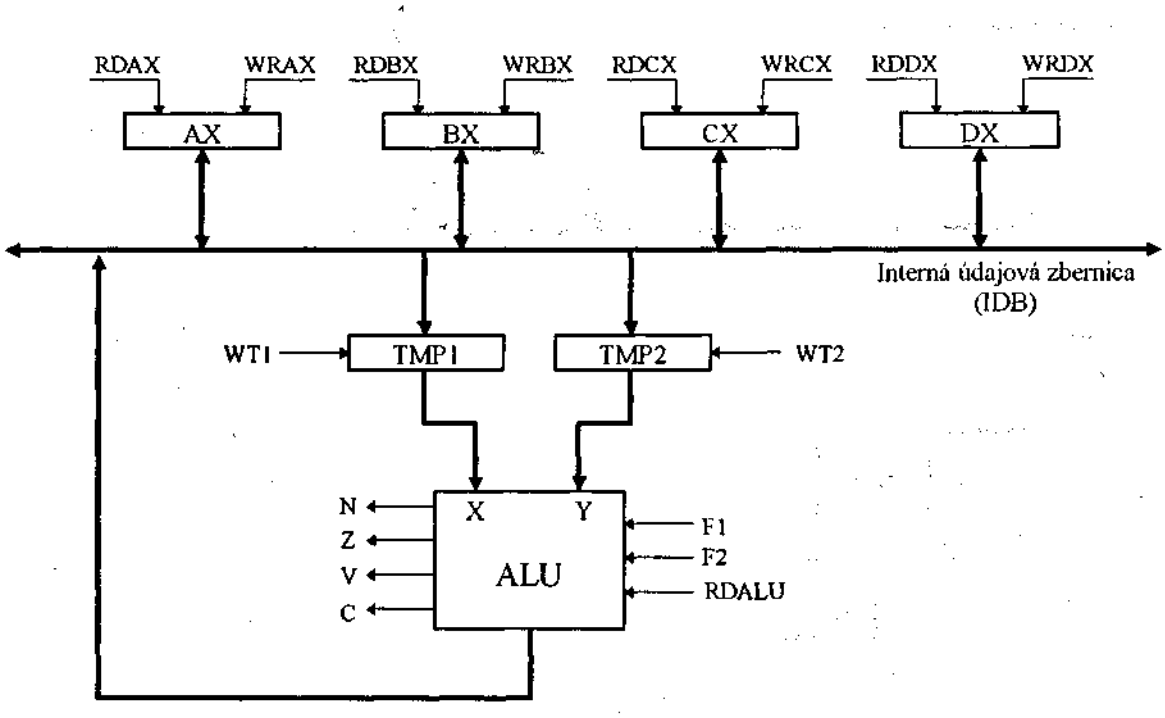
potrebné pripájať výstupy jednotlivých prvkov na zbernicu cez multiplexor, obyčajne sa výstupy pripájajú na zbernicu cez *trojstavové hradlá* alebo sa priamo použijú *prvky s trojstavovým výstupom*.



OBR. 28. Prepojenie registrov s ALU prostredníctvom multiplexorov a demultiplexorov

Na obr. 29 je príklad prepojenia registrov s aritmeticko-logickou jednotkou pomocou zbernice. Použité sú síce ďalšie dva pomocné registre *TMP1* a *TMP2*, ale vzhľadom na štruktúru na obr. 28 je prepojenie jednoduchšie a univerzálnejšie (ako zdroj operandov môžu slúžiť, ľubovoľné registre, jednoduchý presun informácií medzi registrami). Registre *AX* až *DX* majú trojstavové výstupy, ktoré sú ovládané signálmi *RDiX*. *ALU* má takisto trojstavový

výstup, ktorý sa pripojí na zbernicu iba vtedy, ak je požadované vykonanie nejakej operácie (vtedy bude signál *RDALU* aktívny). Interná údajová zbernica je obojsmerná, vstupný, resp. výstupný údajový kanál je možné jednoducho realizovať prostredníctvom trojstavových hradiel (nie sú nakreslené, môžu byť spoločné aj pre riadiacu časť).



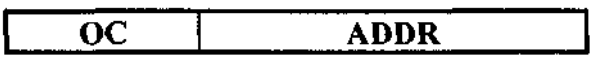
OBR. 29. Prepojenie registrov s ALU prostredníctvom zbernice

4.2.2 Riadiaca časť procesora

Riadiaca časť procesora uskutočňuje *výber a dekódovanie inštrukcií* a zabezpečuje ich *vykonanie*. Okrem toho *riadi spoluprácu procesora s okolím*.

4.2.2.1 Formát inštrukcie

Inštrukcia je príkaz pre procesor, ktorý mu určuje, akú činnosť má vykonať. Vo všeobecnosti sa inštrukcia skladá z viacerých polí (obr. 30).

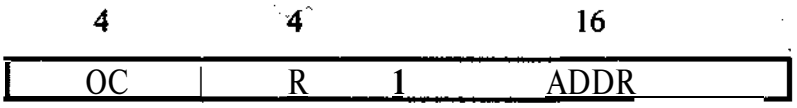


OBR. 30. Všeobecný formát inštrukcie

Pole *OC* je tzv. *operačný kód*. *Operačný kód* špecifikuje *operáciu,* ktorá sa má vykonať. Pole *ADDR* obsahuje *adresu operandu* (operandov) pre príslušnú operáciu. Obyčajne sa používajú *jedno-, dvoj-* prípadne *trojoperandové* inštrukcie. Treba si uvedomiť, že nie všetky inštrukcie však potrebujú pole *ADDR* (napr. inštrukcia pre *povolenie prerušenia).* V takýchto inštrukciách by pole *ADDR* nebolo využité a tento formát by bol neefektívny. Preto sa často používa niekoľko formátov inštrukcií, ktoré majú *rôznu dĺžku* alebo pri inštrukciách *s pevnou dĺžkou* sa nevyužité bity poľa *ADDR* môžu použiť na *rozšírenie operačného kódu.*

Príklad:

Predpokladajme, že máme počítač so *16* registrami, ktorý môže adresovať *64 kB* operačnej pamäte. Použijeme pevný formát inštrukcie, každá inštrukcia bude mať dĺžku *24* bitov a bude obsahovať tieto polia:



Pole *OC* je operačný kód, pole *R* je použité pre adresovanie operandu, ktorý je v registri a pole *ADDR* pre operand, ktorý je v operačnej pamäti.

Pri takomto formáte budeme mať k dispozícii *16 dvoj operandových* inštrukcií. Prvý operand bude v registri a druhý operand v pamäti.

Rozšírenie operačného kódu:

Ak použijeme jednu zo *16* kombinácií poľa *OC* na indikáciu, že operačný kód je *rozšírený* aj o pole *R,* inštrukčný súbor sa bude skladať z *15 dvojoperandových* inštrukcií (prvý operand v registri a druhý v pamäti) a *16 jednooperandových* inštrukcií (operand v pamäti). Opäť

môžeme jednu kombináciu poľa *R* použiť na indikáciu, že operačný kód je rozšírený aj o pole *ADDR* a tým sa nám inštrukčný súbor rozšíri na *15* ***dvojoperandových*** inštrukcií, 75 ***jednooperandových*** inštrukcií a *65536* ***bezoperandových*** inštrukcií. Celá situácia je ilustrovaná na obr. 31.

0000	R	ADDR
0001	R	ADDR
...	...	...
...	...	...
1110	R	ADDR
1111	0000	ADDR
1111	0001	ADDR
...	...	...
...	...	...
1111	1110	ADDR
1111	1111	0000000000000000
1111	1111	0000000000000001
...	...	...
...	...	...
1111	1111	1111111111111110
1111	1111	1111111111111111

OBR. 31. Rozšírenie operačného kódu inštrukcií s pevnou dĺžkou

4.2.2.2 Typy inštrukcií

Podľa *vykonávanej činnosti* delíme inštrukcie na *presunové, výpočtové (operačné), skokové* a *riadiace.* Inštrukcie, uvedené na ilustráciu jednotlivých typov, sú z inštrukčného súboru procesorov rodiny *80x86* a ich detailný opis možno nájsť napr. v [2].

Presunové inštrukcie slúžia na presun údajov medzi registrami procesora, medzi registrom a pamäťou, registrom a V/V zariadením alebo medzi pamäťovými miestami navzájom.

Príklad:

```
MOV     AX,BX
MOV     ALFA,  CX
MOV     BX,  BETA
IN      AL, DX
OUT     10H, AL
MOVSB
```

- *Výpočtové (operačné) inštrukcie* predpisujú vykonanie aritmetických, logických alebo iných operácií s operandmi.

Príklad:

```
ADD     SI,  OFFSET_ARRAY
SUB     CX,  4
AND     AL,  MASK
TEST    AL,  FLAG1
ROL     AX,  CX
```

- *Skokové inštrukcie* slúžia na zmenu lineárneho vykonávania programu, t.j. namiesto inštrukcie, na ktorú ukazuje programové počítadlo, sa vykoná inštrukcia, adresa ktorej je špecifikovaná v práve vykonávanej inštrukcii. Skoky môžu byť *podmienené* alebo *nepodmienené*.

Inštrukcia pre *podmienený skok* testuje nejaký predikát, napr. výsledok porovnania dvoch operandov, príznak nulového výsledku, prenos do vyššieho rádu a pod. Ak je daná podmienka splnená, skok sa uskutoční. V opačnom prípade sa skok neuskutoční a pokračuje sa nasledujúcou inštrukciou.

Príklad:

```
JAE     ABOVEOREQUAL
JZ      ZERO
```

Inštrukcia pre *nepodmienený skok* sa vykoná bez ohľadu na nejaké podmienky.

Príklad: *JMP FAR PTR NEXT_SEG*

Medzi skokové inštrukcie patria aj *inštrukcie volania podprogramu* (sú to *skoky s uchovaním adresy návratu)*. Adresa návratu sa uchová do *zásobníka*. Podprogram je ukončený *inštrukciou návratu z podprogramu*, ktorá zo *zásobníka* vyberie uchovanú adresu návratu a naplní ju do registra PC.

Zásobník sa nachádza v hlavnej pamäti. Je to údajová štruktúra typu *LIFO (Last In First Out)* - posledná vložená položka je prvá vybraná. Pri práci s údajmi v zásobníku sa používa *implicitné zásobníkové adresovanie.* Pri tomto spôsobe adresovania sa adresa pre uloženie/výber údaja nachádza v špeciálnom registri, ktorý sa nazýva *ukazovateľ zásobníka (Stack Pointer - SP).* Okrem odkladania adresy návratu pri volaní podprogramu sa zásobník používa aj na uchovanie *stavu procesora,* pri *obsluhe prerušenia* a na *odovzdávanie parametrov.*

Príklad:
```
CALL    SQRT    ; inštrukcia volania podprogramu
RET     ;       inštrukcia      návratu     z       podprogramu
```

Riadiace inštrukcie sú určené na vykonanie *špeciálnych* operácií, ktoré priamo súvisia s riadiacou alebo operačnou časťou, prípadne s reakciou na externé stimuly. Sú to napr. *inštrukcie na nastavenie* alebo *vynulovanie príznakov* v operačnej časti, *povolenie* alebo *zakázanie externého prerušenia, softvérové prerušenie* atď.

Príklad:
```
CLD
STC
STI
CLI
INT     21H
LMSW    AX
```

4.2.2.3 Spôsoby adresovania operandov

Inštrukcie pracujú s operandmi, ktoré môžu byť adresované rôznymi spôsobmi. Treba si však uvedomiť, že pri ľubovoľnom spôsobe adresovania je výsledkom tzv. *fyzická adresa,* t. j. adresa, ktorú procesor vysiela na adresovú zbernicu. V nasledujúcej tabuľke uvedieme niekoľko najpoužívanejších spôsobov adresovania operandov, pričom ich budeme ilustrovať inštrukciami procesorov rodiny *80x86.*

Tabuľka 1. Spôsoby adresovania operandov

Adresovanie	V inštrukcii	V registri	V pamäti	Príklad
Implicitné : *Registrové* *Nepriame registrové* *Zásobníkové*		 Operand Adresa Adresa	 Operand Operand	 SCASW MOVSB POPF
Bezprostredné	Operand			MOV CX, 10H
Registrové	Register	Operand		MOV BX, DI
Priame	Adresa		Operand	MOV ALFA, DX
Nepriame	Adresa1		Adresa2	JMP WORD PTR DST
Nepriame registrové	Register	Adresa	Operand	JMP WORD PTR [BX]
Indexové	Indexový reg. *Posunutie	Index	Operand	MOV QQQ[DI], AX
Bázovo-indexové	Bázový reg. Indexový reg. *Posunutie	Bázová adresa Index	Operand	MOV AX, QQ[BX][SI]

[1] Posunutie môže, ale nemusí byť. V príklade je uvedené.

4.2.2.4 Riadiaca časť s pevnou logikou

Riadiaca časť s pevnou logikou je implementovaná ako *sekvenčný obvod.* Jeho funkcia je *napevno daná zapojením.*

Vstupný vektor tohto sekvenčného obvodu je tvorený *inštrukciami, signálmi z externého okolia* (napr. žiadosť o prerušenie, signál pripravenosti periférie atď.) a *príznakmi z operačnej časti*.

Výstupný vektor sa skladá z *povelov pre operačnú časť* (napr. signál zápisu do registra, nastavenie funkcie aritmeticko-logickej jednotky atď.) *a pre externé okolie* (napr. signál čítania z pamäte, zápisu do výstupného zariadenia, potvrdenia prerušenia atď.).

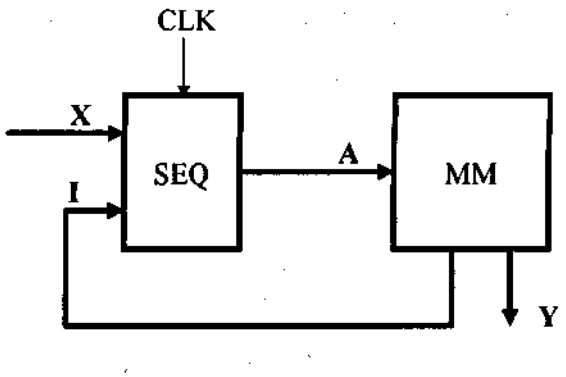
Iba málo riadiacich jednotiek je implementovaných ako asynchrónne sekvenčné obvody, pričom ide obyčajne iba o malé riadiace jednotky. Dôvody spočívajú najmä pri obťažnom odstraňovaní dôsledkov asynchrónne prebiehajúcich udalostí (zmien hodnôt premenných) v logických obvodoch, ktoré môžu vyvolať nesprávnu funkciu systému. Oveľa viac sa používajú riadiace jednotky, implementované ako synchrónne sekvenčné obvody.

4.2.2.5 Mikroprogramová riadiaca jednotka

Mikroprogramové riadiace jednotky sú *synchrónne systémy,* v ktorých je inštrukcia realizovaná *vykonaním mikroprogramu*.

Mikroprogram je postupnosť *mikroinštrukcií,* ktoré sú uložené v *pamäti mikroinštrukcií*. Mikroprogramovú riadiacu jednotku môžeme teda charakterizovať ako špecializovaný procesor, ktorý priamo interpretuje mikroprogramy, uložené v svojej pamäti mikroprogramov a vykonáva ich na danej operačnej časti. Je zrejmé, že zmenou obsahu pamäte mikroinštrukcií je možné dosiahnuť zmenu inštrukčného súboru procesora. Takýmto spôsobom je potom možné na jednom procesore vykonávať programy pre iný typ procesora. Vtedy hovoríme o *mikroprogramovej emulácii.*

Základná štruktúra mikroprogramovej riadiacej jednotky je na obr. 32.



OBR. 32. Koncepcia mikroprogramovej riadiacej jednotky

V *pamäti mikroprogramov (MM) sa nachádzajú mikroprogramy.* Časť pamäte mikroprogramov môže byť realizovaná ako *prepisovateľná,* čo potom umožňuje dynamickú zmenu (doplnenie) inštrukčného súboru procesora.

Úlohou *sekvenčnej jednotky (SEQ)* je *vypracovať adresu A* nasledujúcej mikroinštrukcie. Vstupný vektor *X* predstavujú *príznaky z operačnej časti* a *vonkajšie riadiace vstupy,* vektor */*je informácia z *interných polí* mikroinštrukcie a výstupný vektor *Y* je *riadiacim vektorom pre operačnú časť* a sú z neho odvodené tiež *vonkajšie riadiace výstupy.*

Diskrétny čas je definovaný zmenami synchronizačnej premennej (hodinový signál) *CLK.*

Činnosť mikroprogramovej riadiacej jednotky

Činnosť mikroprogramovej riadiacej jednotky je *cyklická.* V každom *mikrocykle* vyšle sekvenčná jednotka na výstupe *A* adresu mikroinštrukcie, mikroinštrukcia sa načíta, vyšle sa výstupný vektor *Y,* sekvenčná jednotka otestuje hodnoty vstupov *X* a / a vytvorí novú adresu, ktorá sa použije v nasledujúcom mikrocykle.

Príklad:

Máme k dispozícii operačnú časť z obr. 29. Nech *ALU* realizuje operácie +, -, AND a OR, ktoré sú zakódované nasledovným spôsobom:

F1	F2	Operácia
0	0	X + Y
0	1	X - Y
1	0	X *AND* Y
1	1	X *OR* Y

Uvažujme teraz o spôsobe implementácie nasledovných inštrukcií v mikroprogramovej riadiacej jednotke:

MOV AX, ADDR — načítanie údaja z pamäte z adresy *ADDR* do registra *AX*

MOV ADDR, AX — presun obsahu registra *AX* do pamäte na adresu *ADDR*

JMP ADDR — nepodmienený skok na adresu *ADDR*

JZ ADDR — ak príznak Z = *1*, skok na adresu *ADDR*
— ak príznak *Z = 0*, pokračuje sa nasledujúcou inštrukciou

JNC ADDR — ak príznak C = *0*, skok na adresu *ADDR*
— ak príznak C = *J*, pokračuje sa nasledujúcou inštrukciou

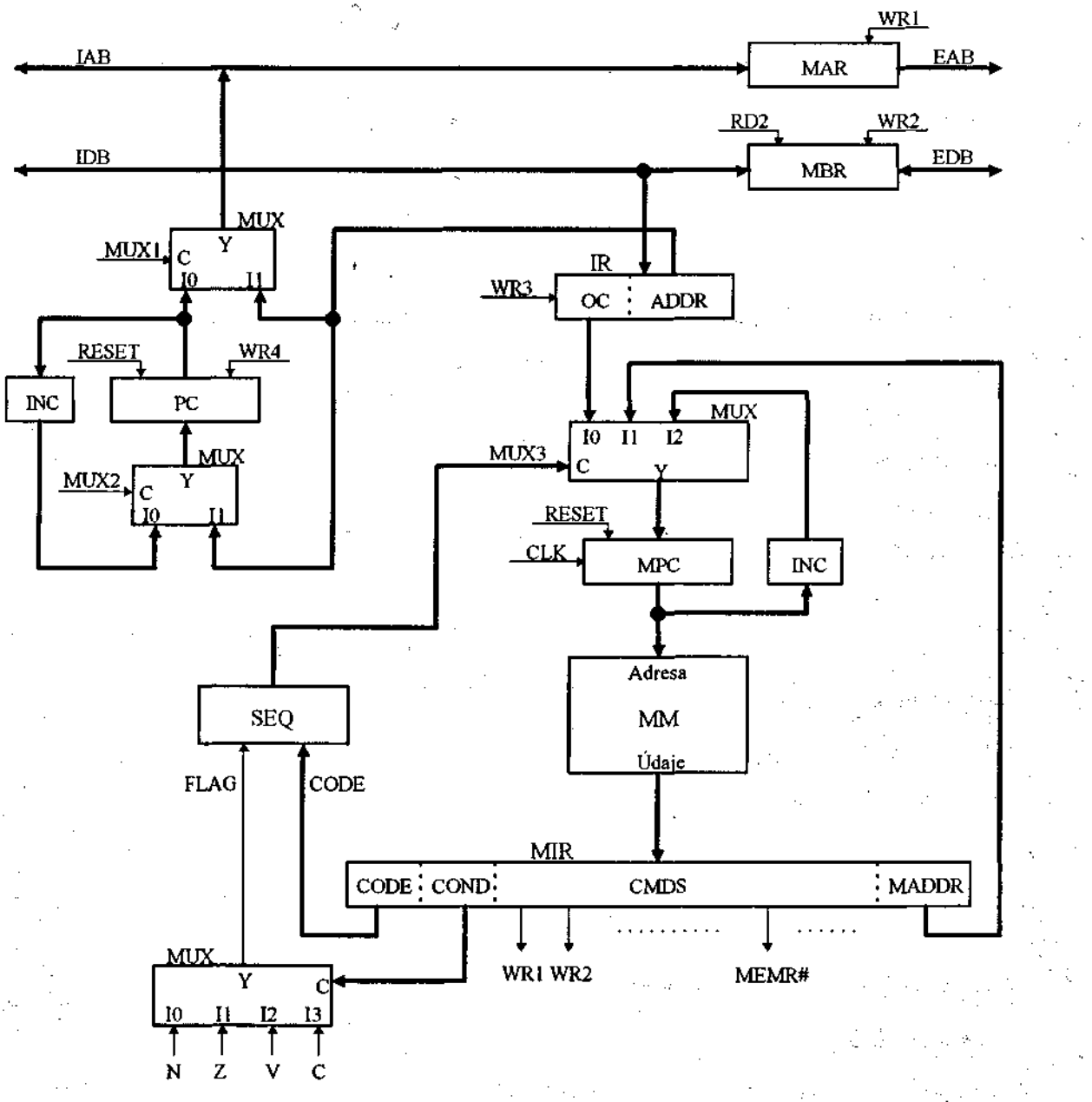
IN AX, ADDR — načítanie údaja zo vstupného zariadenia z adresou *ADDR*

OUT ADDR, AX — zápis obsahu registra *AX* do výstupného zariadenia s adresou *ADDR*

ADD AX, BX — spočítanie obsahu registrov *AX* a *BX*, výsledok sa uloží do *AX*

Na zakódovanie inštrukcií použijeme kvôli jednoduchosti formát inštrukcie z obr. 30.

Podľa koncepcie z obr. 32 teraz navrhneme detailnejšiu štruktúru mikroprogramovej riadiacej jednotky.



OBR. 33. Jednoduchá mikroprogramová riadiaca jednotka

Oddeľovač údajovej zbernice (MBR) oddeľuje *vnútornú údajovú zbernicu* od okolia. Ak *WR2 = 1,* je prepnutý v smere $IDB \rightarrow EDB$, ak *RD2 =1,* v smere $EDB \rightarrow IDB$.

Vnútorná údajová zbernica (IDB) slúži na prenos údajov medzi procesorom a vonkajším okolím (pamäť, resp. V/V zariadenia), medzi registrami v operačnej časti, prípadne na prenos inštrukcií z pamäte do riadiacej časti. Výstupy, ktoré sú na ňu pripojené, musia byť preto trojstavové.

Register adresy (MAR) obsahuje adresu bunky v hlavnej pamäti alebo adresu V/V zariadenia, s ktorým sa v danom cykle zbernice pracuje. Súčasne oddeľuje *internú adresovú zbernicu* od vonkajšieho okolia.

Na *internú adresovú zbernicu* sa vysiela buď adresa inštrukcie, ktorá sa bude v nasledujúcom cykle vyberať z pamäte alebo adresová časť práve vykonávanej inštrukcie (pole *ADDR* z registra *IR*).

Register inštrukcie (IR) obsahuje práve vykonávanú inštrukciu. Všimneme si, že na základe operačného kódu (pole *OC)* sa odštartuje príslušný mikroprogram.

Programové počítadlo (PC) obsahuje adresu nasledujúcej inštrukcie.

Mikroprogramové počítadlo (MPC) obsahuje adresu nasledujúcej mikroinštrukcie.

Register mikroinštrukcie (MIR) obsahuje práve vykonávanú mikroinštrukciu.

Funkcia *pamäte mikroprogramov (MM)* a *sekvenčnej jednotky (SEQ)* už bola opísaná.

Formát mikroinštrukcie

Mikroinštrukcia sa skladá z viacerých polí:

Pole *CODE* určuje typ mikroinštrukcie, je to vlastne pole / z obr. 32.

Pole *COND* vyberá jeden predikát z operačnej časti (príznak *N, Z, V* alebo *C*).

Pole *MADDR* obsahuje adresu mikroinštrukcie, ktorá sa použije ako cieľ pri *skokovej mikroinštrukcii.*

Pole *CMDS* obsahuje povely pre riadiacu časť *(MUX1, MUX2, WR1, WR2, WR3, WR4, RD2),* pre operačnú časť *(WRAX, RDAX, WRBX, RDBX, WT1, WT2, F1, F2, RDALU)* a externé povely *(MEMR#, MEMW#, IOR#, IOW#).* Význam povelov pre riadiacu a operačnú časť je zrejmý z obr. 29 resp. z obr. 33. Externé povely majú tento význam:

MEMR# - signál čítania z pamäte,

MEMW# - signál zápisu do pamäte,

IOR# - signál čítania zo vstupného zariadenia,

IOW# - signál zápisu do výstupného zariadenia.

(# označuje negáciu, t.j. príslušný signál má aktívnu úroveň *log.0*).

Určenie adresy nasledujúcej mikroinštrukcie

V nasledujúcej tabuľke si ukážeme, ako sekvenčná jednotka *SEQ* na základe poľa *CODE* a príznaku *FLAG* určuje adresu nasledujúcej mikroinštrukcie:

Tabuľka 2. Určenie adresy nasledujúcej mikroinštrukcie

CODE	**Adresa nasledujúcej mikroinštrukcie**	*MUX3*	**Poznámka**
0	obsah poľa *OC*	*0*	dekódovanie inštrukcie
1	obsah poľa *MADDR*	*1*	skoková mikroinštrukcia
2	*<MPC> + J*	*2*	obyčajná mikroinštrukcia
3	ak *FLAG = 0* ak *FLAG = 1*	*1* *2*	podmienená mikroinštrukcia

Obsah pamäte mikroprogramov

V tabuľke 3 je uvedený obsah pamäte mikroprogramov na realizáciu požadovaných inštrukcií. V stĺpci "Poznámka" je vyznačený začiatok mikroprogramu pre realizáciu každej inštrukcie (napr. mikroprogram inštrukcie *MOV ADDR,AX* začína na adrese 5).

Znak "-" v poliach *COND* a *MADDR* mikroinštrukcie znamená, že príslušné pole v mikroinštrukcii nie je využité a teda na jeho obsahu nezáleží. V poli *CMDS* reprezentuje situáciu, že žiaden povel nie je v aktívnej úrovni.

Tabuľka 3. Obsah pamäte mikroprogramov

Adresa v MM	*CODE*	*COND*	*CMDS*	*MADDR*	**Poznámka**
0	*2*		*MUX1=0, WR1 =1, MEMR# =0*		Výber inštrukcie z pamäte
1	*2*		*MEMR#* =0, *RD2* =7, *WR3=1, MUX2 =0, WR4* =7		<PC> := <PC>+1
2	*0*	-	-	-	Dekódovanie
3	*2*		*MUX1 =1, WR1*=7, *MEMR#* =0		*MOV AX, ADDR*
4	*1*		*MEMR#* =0, *RD2 =1, WRAX=1*	*0*	
5	*2*		*MUX1=1, WRI =1, RDAX=1, WR2=1, MEMW# =0*		*MOV ADDR, AX*
6	*1*	-	*RDAX =1, WR2=1*	*0*	
7	*1*	-	*MUX2=1, WR4 =7*	*0*	*JMP ADDR*
8	*'3*	7	-	*0*	*JZADDR*
9	*1*	-	*MUX2=1, WR4 =7*	*0*	
10	*3*	*3*	-	*7*	*JNCADDR*
11	*1*	-	-	*0*	
12	*2*		*MUX1=1, WRI =7, IOR# =0*	'	*IN AX, ADDR*
13	*1*		*IOR# =0, RD2=1, WRAX =1*	*0*	
14	*2*		*MUX1=1, WRI =7, RDAX =1, WR2 =1, IOW#=0*		*OUT ADDR, AX*
15	*1*	-	*RDAX =1, WR2=1*	*0*	
16	*2*	-	*RDAX =1,WT1 =1*	-	*ADD AX, BX*
17	*2*	-	*RDBX =1, WT2=1*	-	
18	*1*		*F1 =0, F2* =0, *RDALU*=7,	*0*	

Predpokladáme, že po inicializácii sú registre *PC* a *MPC* vynulované, takže procesor začína činnosť mikroprogramom pre výber a dekódovanie inštrukcie *(MPC=0)*. Program bude takisto začínať na adrese *0 (PC=0)*. Všimneme si, že mikroprogram na realizáciu každej inštrukcie končí mikroskokom na adresu *0* (v pamäti mikroinštrukcií), t.j. nasleduje výber ďalšej inštrukcie z pamäte.

Určenie **operačných kódov inštrukcií**

Ak by sme nepoužili žiadnu *mapovaciu logiku,* priamo z pamäte mikroinštrukcií môžeme určiť operačné kódy jednotlivých inštrukcií - budú to *štartovacie adresy zodpovedajúcich mikroprogramov.*

Inštrukcia	**Operačný kód**
MOV AX, ADDR	*3*
MOV ADDR,AX	***5***
JMP ADDR	*7*
JZ ADDR	*S*
MC ADDR	***10***
IN AX, ADDR	*12*
OUT ADDR, AX	***14***
ADD AX, BX	***16***

Mapovacia logika slúži na prevod operačného kódu inštrukcie na štartovaciu adresu zodpovedajúceho mikroprogramu, takže v prípade jej použitia môžeme pre každú inštrukciu zvoliť kód podľa vlastného výberu.

4.2.2.6 CISC a RISC procesory

- *Procesor s architektúrou CISC (Complex Instruction Set Computer)* sa vyznačuje *zložitým inštrukčným súborom,* ktorý je navrhnutý tak, aby priamo podporoval preklad z

vyšších programovacích jazykov do strojového kódu procesora. Používa sa *veľa spôsobov adresácie operandov,* inštrukcie realizujú aj *zložitejšie,* napr. reťazcové *operácie,* je implementovaná *priama podpora niektorých funkcií operačného systému,* dodržiava sa *kompatibilita inštrukčného súboru v rodinách procesorov* zdola nahor atď. Zabezpečenie všetkých týchto vlastností si vyžaduje zložitú *mikroprogramovú riadiacu jednotku.* Tým sú samozrejme dané aj niektoré nevýhody takýchto procesorov - zložité inštrukcie si vyžadujú pre svoju realizáciu veľký súbor *mikroinštrukcií* a podporných technických prostriedkov. Technické prostriedky na podporu mikroprogramovania zaberajú na čipe veľkú plochu, čím je daná aj vyššia cena. Realizácia inštrukcií vzhľadom na použitie mikroprogramovej riadiacej jednotky trvá dlhšiu dobu. Iba malá podmnožina inštrukčného súboru tvorí väčšiu časť programov, pričom zložitejšie inštrukcie sa využívajú málo. Zložité inštrukcie dokonca môžu nepriaznivo ovplyvňovať optimalizáciu prekladu zložitejších jazykových konštrukcií vyšších programovacích jazykov. Predstaviteľmi procesorov *CISC* sú napr. procesory počítačov radu *IBM 43xx, VAX 11/780* alebo mikroprocesory rodiny *Intel 80x86.*

- *Procesor s architektúrou RISC (Reduced Instruction Set Computer)* sa vyznačuje *redukovaným inštrukčným súborom.* Inštrukcie sú *jednoduché,* preto ich vykonanie trvá veľmi krátko (typicky sa vykonávajú v *jednom strojovom cykle),* používa sa *málo spôsobov adresácie operandov* (obyčajne iba inštrukcie typu *čítanie/zápis z/do hlavnej pamäte), väčší počet univerzálnych registrov* (desiatky až stovky). Riadiaca jednotka je jednoduchšia, obyčajne je to riadiaca jednotka *s pevnou logikou.* Na čipe sa uvoľnila značná časť plochy, ktorá môže byť použitá napr. na implementáciu *numerického koprocesora* a *vyrovnávacej pamäte.* Vykonanie inštrukcie, uloženej v tejto vyrovnávacej pamäti, je rýchlosťou porovnateľné s vykonaním mikroinštrukcie. Program, v ktorom sa nachádzajú aj zložitejšie inštrukcie, je síce pri procesore *CISC* kratší, ako pri procesore *RISC* (procesor *RISC* musí tieto zložitejšie inštrukcie nahradiť postupnosťou svojich jednoduchých inštrukcií), ale vzhľadom na réžiu procesora *CISC* (musí inštrukciu dekomponovať na postupnosť mikroinštrukcií) je jeho vykonanie procesorom *RISC* rýchlejšie. Predstaviteľmi procesorov *RISC* sú napr. procesory *SPARC, Motorola 88000, Transputer* ív *INMOS* atď.

4.2.3 Zvyšovanie výkonnosti procesorov

4.2.3.1 Zvyšovanie pracovnej frekvencie

Zdokonaľovanie technológie výroby integrovaných obvodov viedlo k zmenšovaniu fyzických rozmerov jednotlivých súčiastok na čipe a tým k ich priblíženiu. Zmenšenie oneskorenia šírenia signálov umožnilo podstatné zvýšenie pracovnej frekvencie procesorov. Ak pracovné frekvencie prvých mikroprocesorov boli rádovo *jednotky MHz,* pracovné frekvencie súčasných procesorov sa pohybujú už v *stovkách MHz* (napr. *PENTIUM Pro* pracuje na frekvencii *200 MHz, ALPHA 21164-400* na frekvencii *400 MHz* a ohlasované je ďalšie zvyšovanie).

Vzhľadom na fyzikálne obmedzenia (stavebné prvky musia mať isté minimálne rozmery, obťažný odvod tepla z integrovaného obvodu s veľmi veľkou hustotou integrácie atď.) sú možnosti ďalšieho prudkého zvyšovanie pracovnej frekvencie obmedzené. Ďalej, zvýšením pracovnej frekvencie procesora vznikol nový zásadný problém - preklenutie rádových *rozdielov* v *rýchlosti procesora* a *prístupovej doby pamätí a V/V zariadení.*

4.2.3.2 Výpočty v pohyblivej rádovej čiarke

Štandardný univerzálny procesor má obyčajne iba inštrukcie pre celočíselnú aritmetiku a operácie v pohyblivej rádovej čiarke sa preto musia vykonávať pomocou programu. Toto programové riešenie však veľmi zdržuje celý výpočet, preto boli vytvorené špecializované procesory pre výpočty v pohyblivej rádovej čiarke. Tieto procesory sú obyčajne *pasívne*, to znamená, že samotné si nedokážu vybrať inštrukciu z pamäte a musí to za nich vykonať hlavný (univerzálny) procesor. Program pre tento špecializovaný procesor je potom súčasťou programu hlavného procesora. Špecializovaný procesor pre operácie v pohyblivej rádovej čiarke sa potom označuje ako *numerický koprocesor*. Okrem základných aritmetických operácií vie koprocesor realizovať aj napr. niektoré *transcendentálne* a iné funkcie. Niektoré súčasné výkonné procesory už majú numerický koprocesor integrovaný priamo na čipe spolu s univerzálnym procesorom (napr. *PENTIUM).*

4.2.3.3 Predvýber inštrukcií

Z obsahu pamäte mikroinštrukcií (tab. 3) vidíme, že vykonávanie inštrukcií si nevyžaduje, aby procesor v každom cykle pristupoval k zbernici. To znamená, že zbernica nie je využitá počas všetkých hodinových cyklov. Na druhej strane, počas výberu a dekódovania inštrukcií sú aj niektoré časti procesora nevyužité. Ďalšou skutočnosťou je rozdiel medzi rýchlosťou procesora a dobou prístupu pamäte. Od týchto poznatkov bol už iba krok k myšlienke oddeliť výberovú a výkonnú fázu inštrukcií takým spôsobom, aby jednotky, ktoré tieto činnosti realizujú, mohli pracovať nezávisle.

Jednotka predvýberu inštrukcií (Prefetch Unit) vyberá inštrukcie z pamäte a ukladá ich do *frontu predvýberu (Prefetch QueueJ* v procesore zakaždým, keď je zbernica voľná (t.j. neprenášajú sa údaje). Inštrukcie na dekódovanie sa potom vyberajú už nie z hlavnej pamäte, ale z tohto frontu predvýberu. Týmto sa dosiahli súčasne dve dôležité výhody. Prvou je to, že rýchly procesor nemusí čakať na pomalú pamäť pri výbere inštrukcií (front predvýberu je rýchlosťou porovnateľný s rýchlosťou procesora), druhou výhodou je efektívne využitie vonkajšej zbernice počítača.

Predvýber inštrukcií robí *lineárne,* t. j. inštrukcie sa predvyberajú z pamäte v poradí, v akom nasledujú za sebou. Je zrejmé, že v prípade, ak sa narazí na skokovú inštrukciu, front predvýberu je potrebné vyprázdniť a začať inštrukcie predvyberať z adresy, uvedenej v skokovej inštrukcii. Tým môže dôjsť k istému zníženiu efektívnosti. Tento problém vzniká najmä pri *podmienených skokových inštrukciách,* kedy vopred nie je známe, či sa skok uskutoční alebo nie.

4.2.3.4 Prúdové spracovanie inštrukcií

Inštrukčný cyklus sa vo všeobecnosti skladá z týchto fáz:

J. výber inštrukcie z pamäte,

2. dekódovanie inštrukcie,

3. výber operandov (ak sú v pamäti alebo vo vstupnom zariadení. Ak sú operandy v *registroch, táto fáza odpadá),*

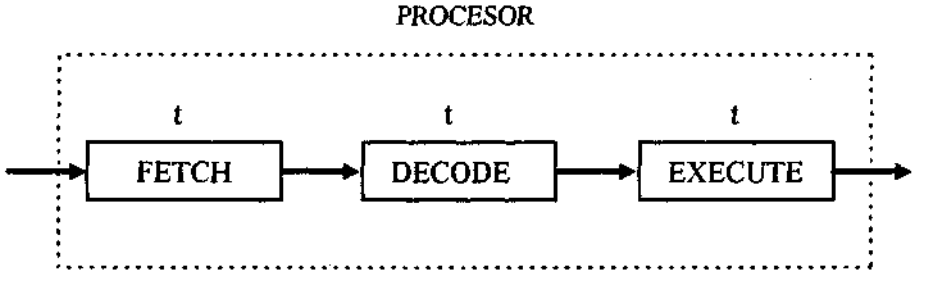
4. *vykonanie požadovanej operácie nad operandmi,*
5. *zápis výsledku do pamäte alebo výstupného zariadenia (ak má výsledok zostať v registri, táto fáza odpadá).*

Pre zjednodušenie predpokladajme, že všetky inštrukcie, ktoré sú v procesore spracúvané, majú svoje operandy uložené v registroch procesora a tu budú uložené aj výsledky. Potom sa inštrukčný cyklus bude skladať z týchto fáz:

1. *výber inštrukcie z pamäte,*
2. *dekódovanie inštrukcie,*
3. *vykonanie operácie.*

Nech každá fáza trvá rovnako dlho a tento čas označme *t*. V nami navrhnutom procesore potom vykonanie jednej inštrukcie bude trvať *3t*. Vykonanie dvoch inštrukcií bude trvať *6t,* troch *9t* atď. Grafické zobrazenie tejto situácie je na obr. 35a. Symboly *I1, I2, I3* v spodnej časti obrázku vyznačujú okamih, kedy príslušná inštrukcia vstupuje do procesora na spracovanie, v hornej časti obrázku tieto symboly vyznačujú okamih, kedy sú na výstupe procesora poskytnuté výsledky príslušnej inštrukcie.

Nech je však každá fáza realizovaná nezávislou funkčnou jednotkou, pričom tieto jednotky sú zapojené za sebou, ako je naznačené na obr. 34. Fázu *J* realizuje jednotka *FETCH,* fázu *2* jednotka *DECODE* a fázu *3* jednotka *EXECUTE.*

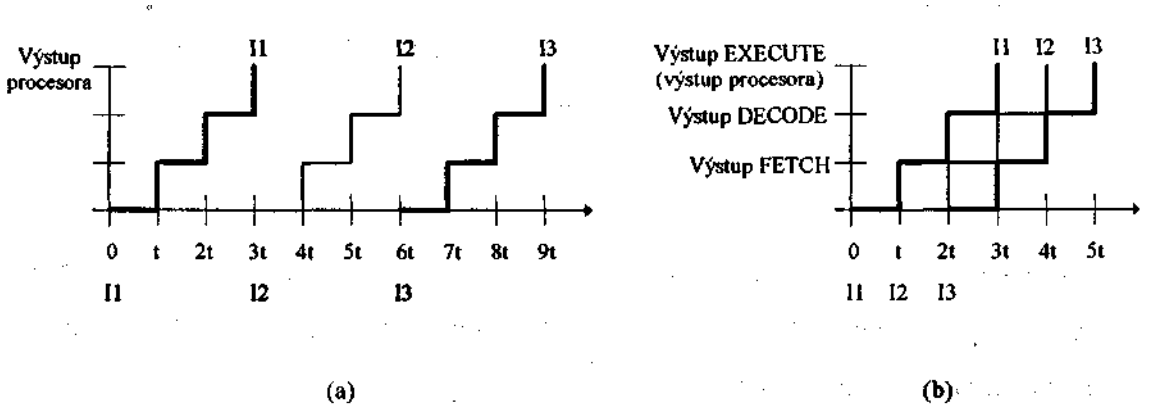


OBR. 34. Procesor so zreťazenými funkčnými jednotkami

Keď sa budú teraz inštrukcie spracúvať, prvá inštrukcia *I1* sa za čas / jednotkou *FETCH* vyberie a dostane sa na výstup, ktorý je súčasne vstupom jednotky *DECODE.* Pretože jednotka *FETCH* sa uvoľnila, môže vybrať druhú inštrukciu *I2.* Za čas *t* sa inštrukcia *12* vyberie a v tomto istom čase sa inštrukcia *II* dekóduje a dostane sa na vstup jednotky *EXECUTE.* Za ďalší čas *t* jednotka *FETCH* vyberie už tretiu inštrukciu *I3,* inštrukcia *I2* je dekódovaná a *I1* je už vykonaná.

Teraz na obr. 35b vidíme, že vykonanie prvej inštrukcie *II* trvá síce rovnako ako v predchádzajúcom prípade čas *3í,* ale vykonanie každej nasledujúcej inštrukcie už iba *t,* čím sa rýchlosť spracovania podstatne zvýšila.

Samozrejme, v reálnom procesore treba ešte riešiť problémy, ktoré vznikajú s prístupom procesora k operandom v pamäti resp. V/V zariadeniach.



OBR. 35. Porovnanie rýchlosti spracovania inštrukcií v procesore bez zreťazenia (a) a v zreťazenom procesore (b)

4.2.3.5 Paralelné vykonávanie inštrukcií

Treba poznamenať, že stále ide o program pre procesor s jedným prúdom inštrukcií a jedným prúdom údajov. Programátor sa aj pri tomto spracovaní stále díva na inštrukcie tak, ako keby boli spracúvané iba sekvenčne. Paralelne je možné vykonať iba tie inštrukcie, ktoré za sebou bezprostredne nasledujú a pritom nie sú od seba závislé, t.j. jedna nemôže poskytovať výsledok, ktorý iná používa ako operand *(údajová nezávislosť)* a takisto nesmú byť zviazané

cez príznaky *(riadiaca nezávislosť)*. Tie inštrukcie, ktoré nie je možné vykonať paralelne, sú vykonané sekvenčne.

Vnútorný mechanizmus procesora rozhoduje o tom, ktoré inštrukcie budú vykonané paralelne. Procesor musí mať samostatné kompletné jednotky pre paralelné vykonanie inštrukcií. Reálne sa používa napr. *párovanie inštrukcií* (procesor *PENTIUM)* **[2].**

4.2.4 Prerušovací systém procesora

Charakteristickým znakom súčasných procesorov je vyspelý *prerušovací systém,* ktorý umožňuje efektívnu implementáciu viacpoužívateľských a viacprogramových *operačných systémov* a rýchlu odozvu na *externé udalosti.*

Predstavme si situáciu, že procesor vykonáva program a zrazu nastane požiadavka okamžitej obsluhy novej *udalosti.* Procesor musí *prerušiť* vykonávanie práve bežiaceho programu a začať vykonávať nový program - *obslužný program prerušenia.* Po skončení obslužného programu bude procesor pokračovať v pôvodnom - *prerušenom* programe.

Prerušenie sa teda skladá z týchto krokov:

1. prijatie požiadavky na prerušenie,
2. odloženie stavu procesora,
3. zistenie zdroja prerušenia,
4. vykonanie zodpovedajúceho obslužného programu prerušenia,
5. obnovenie pôvodného stavu procesora,
6. pokračovanie v prerušenom programe.

K bodu 1.

Požiadavka na *externé* prerušenie môže prísť v ľubovoľnom okamihu, t.j. aj uprostred vykonávania inštrukcie. S obsluhou prerušenia (t.j. odloženie stavu procesora atď.) sa však začne až po dokončení práve vykonávanej inštrukcie.

K bodu 2.

Okamžitý stav procesora je charakterizovaný obsahom všetkých registrov procesora. Jedným z registrov je *programové počítadlo,* ktoré obsahuje adresu nasledujúcej inštrukcie. Inštrukciou, ktorá sa nachádza na tejto adrese, sa bude pokračovať po skončení obsluného programu prerušenia. *Stav procesora sa odkladá do zásobníka* (pozri časť 4.2.2.2). Použitie zásobníka umožňuje aj *hniezdenie prerušení,* to znamená, že počas obsluhy jedného prerušenia môže prísť k akceptovaniu ďalšieho prerušenia *s vyššou prioritou.*

K bodu 3.

V počítači môže byť viac zdrojov prerušenia, ktoré musia byť *samostatne identifikovateľné.* Procesor musí zistiť, ktorý zdroj prerušenia požaduje obsluhu, aby vedel odštartovať zodpovedajúci obslužný program.

Synchrónne prerušenia majú *pevne určené štartovacie adresy* obslužných programov, pri *asynchrónnych prerušeniach* sa obyčajne štartovacia adresa určí prostredníctvom *prerušovacieho vektora.*

Prerušovací vektor je ukazovateľ do tabuľky štartovacích adries obslužných programov prerušení. Tento vektor je načítaný procesorom z údajovej zbernice po prijatí prerušenia v špeciálnom *cykle potvrdenia prerušenia (Interrupt Acknowledge Cycle).* Každému zdroju prerušenia je priradený vlastný prerušovací vektor.

K bodu 5.

Obslužný program prerušenia je ukončený *inštrukciou návratu z prerušenia,* ktorá zo zásobníka *obnoví pôvodný stav procesora.*

4.2.4.1 Asynchrónne prerušenie

Asynchrónne prerušenie je prerušenie, ktoré priamo nesúvisí s vykonávanými inštrukciami a môže nastať kedykoľvek. Je to tzv. *externé (hardvérové) prerušenie* a typicky je požadované niektorým vstupno/výstupným zariadením, keď je toto *pripravené na prenos.*

Procesor má zvyčajne dva prerušovacie vstupy pre externé prerušenie:

- *Vstup maskovateľného prerušenia*. Inštrukčný súbor procesora obsahuje v tomto prípade špeciálne inštrukcie, ktoré umožňujú *povoliť* resp. *zakázať* prijatie požiadavky z tohto vstupu.

- *Vstup nemaskovateľného prerušenia*. Toto prerušenie nie je možné zakázať a typicky sa používa pri obsluhe katastrofických situácií (napr. *výpadok napájacieho napätia)*.

4.2.4.2 Synchrónne prerušenie

Synchrónne (interné) prerušenie priamo súvisí s vykonávanými inštrukciami a nie je ho možné zakázať. Synchrónne prerušenie je dvojaké:

- *Softvérové prerušenie*. Softvérové prerušenie je generované po vykonaní špeciálnej *riadiacej inštrukcie*. Parametrom tejto inštrukcie je *číslo prerušenia,* ktoré sa má obslúžiť. Toto prerušenie sa typicky používa pri *volaní funkcií operačného systému*.

- *Výnimka (Exception)*. Výnimka sa generuje automaticky, ak nastane nejaká chyba pri vykonaní inštrukcie (napr. *nedefinovaný operačný kód, delenie nulou, pokus o zápis do oblasti, kde sú uložené inštrukcie, sprístupňovaný segment sa nenachádza v hlavnej pamäti* atď.).

Využitie prerušovacieho systému procesora pri prenose údajov medzi počítačom a vstupno/výstupným zariadením si ukážeme v časti 4.4.5.

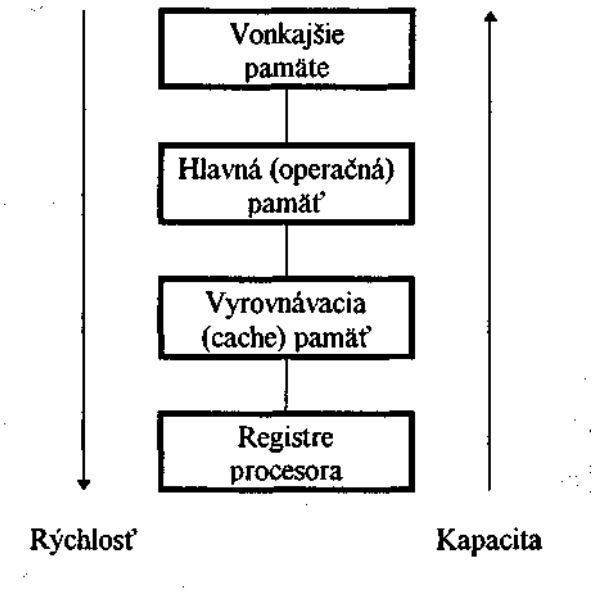
4.3 Pamäťový podsystém počítača

Pamäťový podsystém počítača slúži jednak na uloženie programu a údajov, ktoré sa práve používajú, ako aj na ich archiváciu.

4.3.1 Hierarchická organizácia pamäťového podsystému počítača

Pamäťové prostriedky počítača je možné rozdeliť do niekoľkých úrovní, ktoré sa líšia spôsobom použitia v procese spracovania informácií. Jednotlivé úrovne majú pritom aj rozdielnu kapacitu a operačnú rýchlosť.

Na obr. 36 je nakreslená hierarchická organizácia pamäťového podsystému počítača. Je vyznačené, ako sa pre jednotlivé úrovne zvyšuje rýchlosť a kapacita.



OBR. 36. Hierarchická organizácia pamäťového podsystému počítača

Charakteristika jednotlivých úrovní

- *Registre procesora* sa nachádzajú priamo na čipe procesora. Slúžia na prechodné uchovanie informácií počas ich spracovávania v procesore. Registre sú najrýchlejšie zo všetkých častí pamäťového podsystému, t.j. majú najkratšiu *dobu prístupu*. Pod *dobou prístupu* rozumieme čas, ktorý uplynie od nastavenia požiadavky na pamäť (register) po poskytnutie požadovaného údaja. Počet registrov v procesore je rôzny pri rôznych typoch procesorov, sú ich jednotky až desiatky. Doba prístupu registrov je rádovo *jednotky nanosekúnd (ns)*.

- *Hlavná (operačná) pamäť* obsahuje práve vykonávaný program a spracúvané údaje. Ideálne by bolo, keby sa aj hlavná pamäť nachádzala na čipe procesora a dosahovala rýchlosť registrov. Toto však vo všeobecnosti technicky nie je realizovateľné vzhľadom na požadovanú *kapacitu pamäte*. *Kapacitou pamäte* sa rozumie množstvo informácií, ktoré je schopná pamäť uchovať. Treba si uvedomiť, že programy sa bežne skladajú z niekoľko tisíc inštrukcií, pričom ešte vôbec nehovoríme o množstve spracúvaných údajov. Z tohto vyplýva, že hlavná pamäť je realizovaná ako samostatný funkčný blok, ktorý sa skladá z jedného alebo niekoľkých integrovaných obvodov. Tým, že je hlavná pamäť vzdialená od procesora, je dané prvé nevyhnutné predĺženie doby prístupu, zapríčinené konečnou dobou šírenia signálu medzi procesorom a pamäťou. Jednotlivé bunky pamäte by bolo možné realizovať podobne, ako registre procesora, pamäť by však bola veľmi rozsiahla, drahá, s veľkou spotrebou energie. Z týchto dôvodov je hlavná pamäť realizovaná odlišnými spôsobmi ako registre procesora (napr. používajú sa *dynamické pamäti,* ktoré pri nepatrných rozmeroch a nízkej spotrebe energie majú veľkú kapacitu). Na druhej strane však za zvýšenie kapacity platíme znížením rýchlosti. Doba prístupu hlavnej pamäte je rádovo *desiatky ns*. Kapacita sa pohybuje od *jednotiek megabajtov* až do *niekoľko gigabajtov (GB)*.

- *Vonkajšie pamäti* slúžia na uchovanie informácií, ktoré sa momentálne nepoužívajú a na archiváciu informácií. Na rozdiel od hlavnej pamäte, k vonkajším pamätiam procesor pristupuje ako k *vstupno/výstupným zariadeniam*. Ako typické vonkajšie pamäti sa používajú *pevné disky, diskety, kazetovo-páskové jednotky, CD-ROM* atď. Doba prístupu je rôzna, záleží od typu vonkajších pamätí, napr. *pevné disky* majú v súčasnosti dobu prístupu *jednotky milisekúnd (ms)* a ich kapacita dosahuje *niekolko gigabajtov (GB)*.

- *Vyrovnávacia pamäť (cache)* slúži na preklenutie v podstate rádového rozdielu medzi prístupovou dobou registrov procesora a hlavnej pamäte. Je to rýchla pamäť rádovo menšej kapacity, akú má hlavná pamäť, *umiestnená medzi procesor a hlavnú pamäť*. Do vyrovnávacej pamäte sa *presunie časť obsahu hlavnej pamäte* a procesor potom sprístupňuje informácie z vyrovnávacej pamäte vyššou rýchlosťou. Samozrejme, ak nastane situácia, že požadovaná informácia sa vo vyrovnávacej pamäti nenachádza, procesor musí túto informáciu sprístupniť z hlavnej pamäte. Vyrovnávaciu pamäť riadi vlastný *riadiaci obvod,* ktorý zabezpečuje nielen presun požadovaných informácií z

hlavnej pamäte do vyrovnávacej pamäte, ale rieši aj ďalšie problémy, napr. situáciu, že procesor modifikoval nejaký údaj vo vyrovnávacej pamäti. Túto zmenu je nutné kvôli zhode informácií vykonať aj v hlavnej pamäti. Kapacita vyrovnávacej pamäte býva rádovo *desiatky až stovky kilobajtov (kB)* a doba prístupu je rádovo *nanosekundy (ns).* Vzhľadom k pokroku technológie výroby integrovaných obvodov sa už objavila aj vyrovnávacia pamäť priamo na čipe procesora, napr. mikroprocesor *PENTIUM* má priamo na čipe integrovanú vyrovnávaciu pamäť s kapacitou *16 kB.* Aj v tomto prípade má však počítač realizovanú externú vyrovnávaciu pamäť s väčšou kapacitou a potom hovoríme o *dvojúrovňovej* vyrovnávacej pamäti.

Pre úplnosť treba poznamenať, že aj rôzne periférne zariadenia (napr. *tlačiareň* alebo *pevný disk)* môžu mať vlastnú vyrovnávaciu pamäť. Táto pamäť však nesúvisí s vyššie uvedenou hierarchickou organizáciou pamäťového podsystému počítača.

4.3.2 Rozdelenie pamätí

Podľa spôsobu prístupu k informáciám:

- *Pamäti s náhodným prístupom (RAM - Random Access Memory).* Pri tomto type pamätí je doba prístupu pre jednotlivé bunky rovnaká, nezáleží od ich umiestnenia y pamäti. Typickým príkladom je *hlavná pamäť počítača* von Neumannovského typu.

- *Pamäti so sekvenčným prístupom (SAM - Sequential Access Memory).* Pri tomto type sa adresované miesto sprístupní až po systematickom prehľadaní predchádzajúcich buniek, takže doba prístupu záleží od umiestnenia adresovanej bunky v pamäti. Je to typické napr. pre *páskové* a *diskové pamäti.*

- *Pamäti s asociatívnym prístupom* alebo *pamäti adresované obsahom (CAM - Content Access Memory).* Sprístupnenie pamäťového miesta sa pri asociatívnej pamäti uskutoční nie na základe adresy, ale porovnaním všetkých buniek s tzv. *výberovým kľúčom.* Tento typ pamätí sa používa v špecializovaných počítačových architektúrach *(asociatívne počítače, data-flow počítače* a pod.) a pri realizácii *vyrovnávacích pamätí.*

Podľa **možnosti čítania a zápisu:**

- *Pamäti pre čítanie a zápis (RWM - Read/Write Memory).* Do týchto pamätí je možné v priebehu činnosti kedykoľvek informáciu zapísať a kedykoľvek ju čítať. Typickou pamäťou *RWM* je napr. *hlavná pamäť počítača.* Niektoré typy pamätí *RWM* si po vypnutí napájacieho napätia svoj obsah uchovajú (napr. *pamäti s magnetickým záznamom),* iné svoj obsah stratia *(polovodičové pamäti).*

- *Pamäti iba pre čítanie (ROM - Read Only Memory).* Z pamäte typu *ROM* je možné informáciu iba čítať. Prvotný zápis informácie sa vykoná buď pri výrobe pamäte, alebo si ich môže v špeciálnom zariadení *naprogramovať* používateľ. Typickou vlastnosťou pamätí *ROM je* to, že si uchovajú svoj obsah aj po vypnutí napájacieho napätia.

4.3.3 Hlavná pamäť počítača

Hlavná pamäť počítača obsahuje práve vykonávaný program a spracúvané údaje. Je to pamäť s náhodným prístupom (RAM) a obyčajne sa skladá z dvoch častí, z ktorých jedna je typu *ROM* a druhá *RWM.* Z hľadiska *fyzickej realizácie* je hlavná pamäť vytvorená z *polovodičových pamätí.*

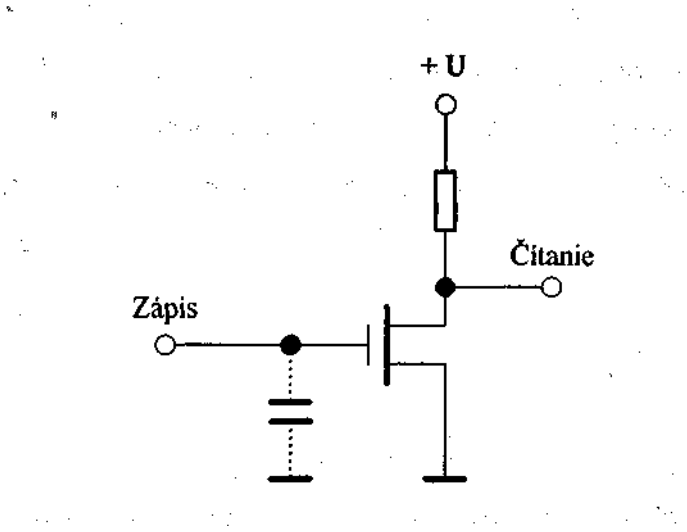
Polovodičové pamäti

Polovodičové pamäti typu RWM sú realizované na polovodičovom čipe a vyhotovujú sa ako *statické* alebo *dynamické.*

- Pri *statických polovodičových pamätiach RWM (SRAM)* je základný pamäťový element realizovaný ako *preklápací obvod* (pozri časť 3.6.2.1). Preklápací obvod po zápise informácie zostáva v stabilnom stave, ktorý sa zmení až zápisom novej hodnoty.

- Pri *dynamických pamätiach RWM (DRAM)* je základný pamäťový element realizovaný pomocou *parazitnej kapacity,* ktorá sa pri zápise nabije. Vplyvom zvodového prúdu sa

však elektrický náboj postupne vybíja a po istom čase by prišlo k jeho strate, a tým aj k strate zaznamenanej informácie. Z tohto dôvodu je nutné pri dynamických pamätiach periodicky obnovovať náboje na parazitných kapacitách. Táto činnosť sa nazýva *občerstvovanie dynamickej pamäte (Refresh)*. Na obr. 37 je naznačený spôsob realizácie jedného bitu v dynamickej pamäti. Kondenzátor, ktorý predstavuje parazitnú kapacitu, je nakreslený čiarkovane, pretože nejde o samostatný prvok, ale parazitná kapacita je iba dôsledkom špeciálnej technologickej realizácie tranzistora.



OBR. 37. Realizácia jedného bitu v dynamickej pamäti

Obsah polovodičových pamätí *RWM,* nezáleží či sú statické alebo dynamické, sa po vypnutí napájacieho napätia *stratí.*

Polovodičové pamäti ROM si *uchovajú* svoj obsah aj po vypnutí napájacieho napätia. Okrem polovodičových pamätí *ROM*, u ktorých je *prvotný zápis informácie vykonaný už pri výrobe pamäte* (poslednou technologickou maskou), existujú aj *používateľom programovateľné polovodičové pamäti ROM.*

- *Pamäť typu PROM (PROM - Programmable ROM)* je jedenkrát naprogramovateľná polovodičová pamäť *ROM.* Naprogramovanie je *trvalé,* to znamená, že zmena obsahu pamäte po naprogramovaní už nie je možná.

Pamäti typu EPROM sú programovateľné pamäti *ROM* s možnosťou vymazania a opätovného naprogramovania *(EPROM - Erasable PROM.* Pamäť sa maže pôsobením *ultrafialového žiarenia* na pamäťové elementy. Aby sa to dalo uskutočniť, pamäť má špeciálne puzdro, kde nad pamäťovými elementmi je *okienko z kremičitého skla.).*

Pamäti typu EEPROM sú elektricky mazateľné pamäti *PROM (EEPROM - Electrically Erasable PROM).*

4.3.3.1 Stavebné prvky hlavnej pamäte

Ako bolo uvedené v predchádzajúcej časti, hlavná pamäť je realizovaná z polovodičových pamätí. Tieto pamäte sú fyzicky realizované ako *integrované obvody.*

Organizácia pamäťových buniek v rámci jedného integrovaného obvodu je v súčasnosti v podstate dvojaká:

- *Statické pamäti RWM a pamäti ROM (PROM, ...)* majú obyčajne *slabikovú organizáciu,* t.j. do jednej bunky pamäte je možné uložiť *slabiku (Bajt).* Šírka údajovej zbernice pamäte je potom samozrejme *8 bitov. Kapacita pamäte* sa pri slabikovej organizácii udáva v *bajtoch (B)* a vypočíta sa ako mocnina *2",* kde *n* označuje šírku adresovej zbernice pamäte.

- *Dynamické pamäti RWM* majú obyčajne *bitovú organizáciu,* t.j. do jednej bunky pamäte je možné uložiť *1 bit.* Šírka údajovej zbernice pamäte je potom iba *1 bit. Kapacita pamäte* sa pri bitovej organizácii udáva v *bitoch (b)* a vypočíta sa ako mocnina *2",* kde *n* označuje šírku adresovej zbernice pamäte.

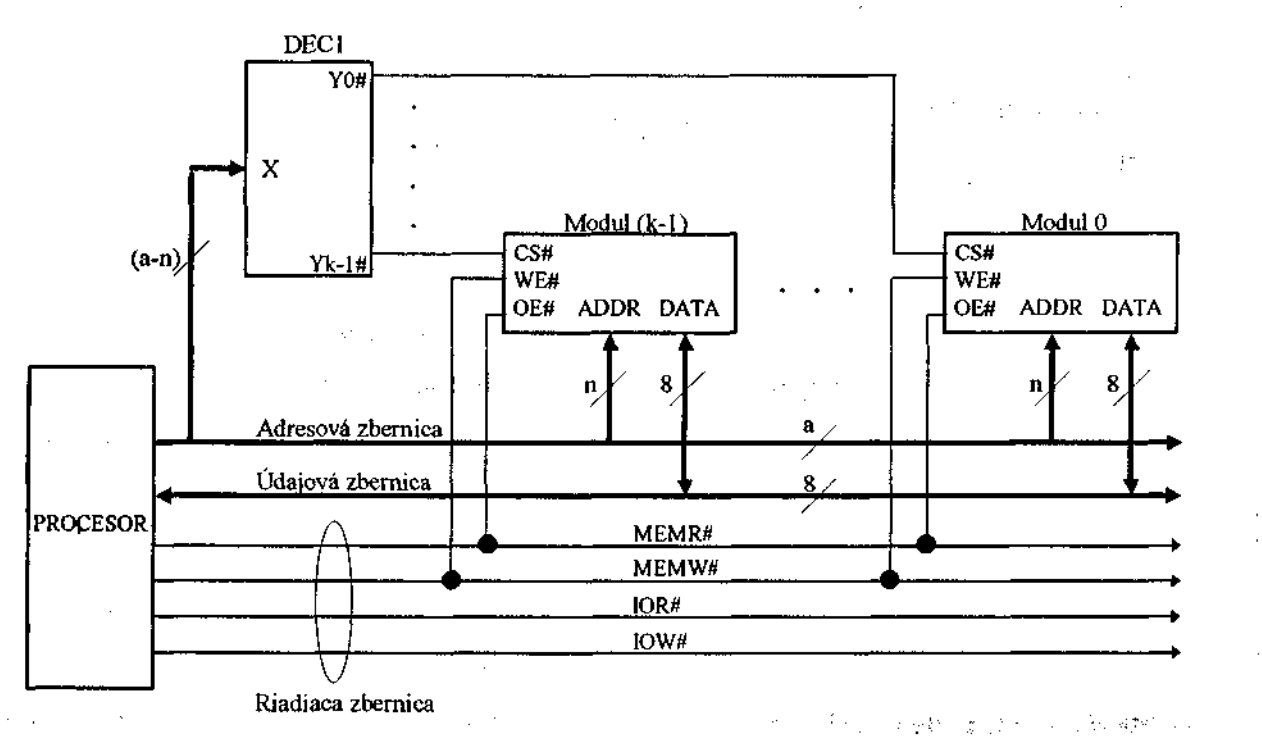
Reálne má hlavná pamäť minimálne *slabikovú organizáciu.* Preto ak sa na jej výstavbu použijú *pamäti s bitovou organizáciou,* pri slabikovej organizácii hlavnej pamäte je potrebné použiť *osmice integrovaných obvodov,* ktoré majú *spoločný výberový signál a spoločné*

riadiace a adresové signály. Údajový vodič každej pamäte v osmici je pripojený na *iný bit* údajovej zbernice počítača.

4.3.3.2 Pripojenie pamäte k zbernici počítača

Zbernica počítača sa skladá z *adresovej, údajovej* a *riadiacej sekcie.* Teraz si ukážeme, akým spôsobom sa pripoja jednotlivé obvody, ktoré vytvárajú hlavnú pamäť, k zbernici. Kvôli jednoduchosti budeme uvažovať obvody typu *RWM so slabikovou* organizáciou.

Na obr. 38 je naznačené pripojenie viacerých pamäťových obvodov k zbernici počítača.



OBR. 38. Pripojenie pamäťových obvodov k zbernici počítača

Šírka údajovej zbernice počítača je *8 bitov,* šírka adresovej zbernice *a-bitov.* Riadiaca zbernica sa skladá zo *4* signálov, ktoré sú riadené procesorom. Majú tento význam:

MEMR# - signál čítania z pamäte,

MEMW# - signál zápisu do pamäte,

I0R# - signál čítania zo vstupného zariadenia,

lOW# - signál zápisu do výstupného zariadenia.

Počas práce s pamäťou sú riadiace signály *IOR#* a *IOW#* v neaktívnej úrovni.

Všetky pamäťové obvody *Modul0, ... Modul(k-1)* sú pripojené na zbernicu počítača *paralelne.* Všimneme si, že na adresové vstupy pamäťových obvodov je pripojených *spodných n bitov* adresy, ktoré slúžia na adresovanie *jednej konkrétnej bunky* vo vybranom pamäťovom obvode.

Dekóder *DEC1* na základe *zvyšnej časti adresy* na adresovej zbernici počítača určuje, s ktorým pamäťovým obvodom sa bude pracovať. Je typu *1 z n,* takže vždy iba jediný pamäťový obvod môže byť aktívny.

Typické vstupy a výstupy pamäťových obvodov typu *RWM*

ADDR - na tento vstup je vyvedená *n-bitová adresová zbernica* pamäte.

DATA- obojsmerná (v našom prípade *8-bitová*) *údajová zbernica* pamäte.

0E# - vstup, slúžiaci na *uvoľnenie trojstavových výstupov* údajovej zbernice pamäte pri čítaní údajov.

WE# - *zapisovací vstup.* Aktívna úroveň na tomto vstupe *(log.O)* zapisuje údaje z údajovej zbernice do pamäťových elementov pamäte.

CS# - *výberový vstup* pamäte. Tak čítanie, ako aj zápis z/do pamäte sa môže uskutočniť iba vtedy, ak je na tomto vstupe aktívna úroveň, t.j. *log.O.* Výberový vstup slúži na to, aby sa pri každom prístupe k hlavnej pamäti pracovalo vždy iba s jedným pamäťovým obvodom.

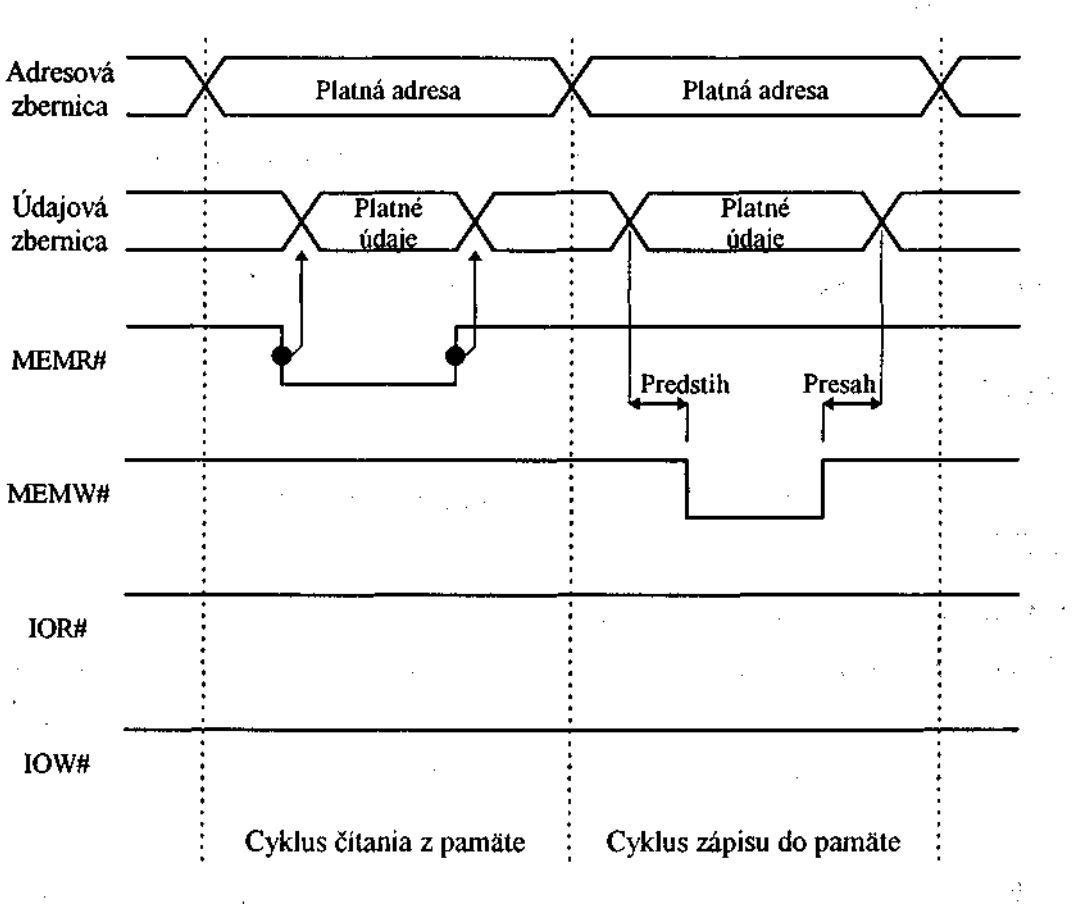
Typické vstupy a výstupy pamäťových obvodov typu *ROM*

Obvody typu *ROM* majú rovnaké signály ako obvody typu *RWM,* chýba im však signál *WE#* (do týchto obvodov sa nedá zapisovať).

4.3.3.3 Komunikácia procesora s pamäťou

- *Čítanie z pamäte*. Procesor musí nastaviť na adresovej zbernici platnú adresu pamäťovej bunky, z ktorej chce načítať informáciu a nastaviť aktívnu úroveň signálu pre čítanie z pamäte *MEMR#*. Signál musí byť v aktívnej úrovni dostatočne dlhý čas. Po istom čase *(doba prístupu pamäte)* pamäť vyšle na údajovú zbernicu platnú informáciu.

- *Zápis do pamäte*. Procesor musí nastaviť na adresovej zbernici adresu bunky pamäte, do ktorej chce zapísať a na údajovú zbernicu vyslať platné údaje. Potom môže nastaviť do aktívnej úrovne signál pre zápis do pamäte *MEMW#*. Je potrebné dodržať jednak istú minimálnu *dobu trvania* signálu *MEMW#*, ako aj dostatočný *predstih* a *presah údajov* voči tomuto signálu, tak ako je naznačené na obr. 39.



OBR. 39. Signálové sledy pre čítanie a zápis z/do pamäte

4.3.4 Správa a ochrana hlavnej pamäte

Ako sme už uviedli v časti 4.2.2.3, existuje veľa spôsobov adresácie operandov. Pri ľubovoľnom spôsobe adresácie je výsledkom tzv. *efektívna* alebo *absolútna adresa,* t.j. adresa, ktorú procesor vysiela na adresovú zbernicu. Hlavná pamäť počítača sa navonok javí ako *lineárny priestor pamäťových buniek,* ktoré nasledujú spojite za sebou a líšia sa iba svojou adresou. Z tohto dôvodu sa nasledujúca rovnosť javí ako celkom prirodzená:

logický adresový priestor = fyzický adresový priestor

pričom *logický adresový priestor* je priradený *programu* a *fyzický adresový priestor* zodpovedá *hlavnej pamäti počítača*. Nie vždy sa však uvedené priradenie dá realizovať.

Zavedenie *správy pamäte* má tri hlavné príčiny:

1. *Program môže vyžadovať pre svoje vykonanie väčší logický adresový priestor, ako je fyzický adresový priestor, ktorý má počítač k dispozícii.*
2. *Vyžaduje sa priradenie a rozdelenie fyzického adresového priestoru viacerým používateľským programom súčasne.*
3. *Požaduje sa ochrana fyzického adresového priestoru, priradeného jednému používateľskému programu, pred ovplyvňovaním iným používateľskými programom.*

K bodu 1.

Táto požiadavka sa rieši vytvorením tzv. *virtuálnej pamäte.* Jej vytvorenie je viazané na existenciu rýchlej vonkajšej pamäte s dostatočnou kapacitou, napr. pevný disk.

K bodu 2.

Musí byť k dispozícii jednoduchý mechanizmus pre *dynamickú relokáciu,* t.j. možnosť premiestňovania oblastí s danou logickou adresou na rôzne fyzické adresy.

K bodu 3.

Je nutné, aby každá oblasť fyzickej pamäte bola vybavená tzv. *prístupovými právami,* ktoré definujú, *kto* ju môže používať a *akým spôsobom (privilegované úrovne, spôsob použitia - čítanie, čítanie aj zápis, vykonanie atď.)*

Preklad logickej adresy na fyzickú adresu a *kontrolu prístupových práv* pri zavedení správy pamäte vykonáva špeciálny blok procesora, ktorý sa *nazýva jednotka správy pamäte (MMU - Memory Management Unit).*

4.3.4.1 Virtuálna pamäť

Princíp *virtuálnej pamäte* spočíva v tom, že nie celý program alebo všetky údaje sa nachádzajú naraz v hlavnej pamäti počítača. Nachádza sa tam iba tá časť, ktorá sa práve používa, zvyšná časť programu alebo údajov je uložená na rýchlej vonkajšej pamäti, napr. na pevnom disku. V prípade, ak procesor chce vybrať inštrukciu alebo sprístupniť údaj, ktorý sa v hlavnej pamäti nenachádza, treba zabezpečiť, aby sa príslušná časť programu alebo údajov preniesla z vonkajšej do hlavnej pamäte.

Najpoužívanejšími spôsobmi realizácie virtuálnej pamäte je *segmentovanie a stránkovanie.*

Segmentovanie

Podstatou *segmentovania* je to, že program a údaje sa rozdelia na bloky - *segmenty.* Inštrukcie, resp. údaje, ktoré sú v jednom segmente, *spolu logicky súvisia.* Veľkosť segmentov môže byť rôzna. Napríklad v jednom segmente sa môže nachádzať *hlavný program,* v ďalšom *podprogramy,* v inom *údaje* atď.

Logická adresa sa v prípade *segmentovania* skladá z dvoch častí, ktoré sa spracúvajú samostatne:

Selektor : Posunutie

Posunutie (offset) reprezentuje *vzdialenosť* inštrukcie alebo údaja od *začiatku segmentu*. Iba *posunutie* vystupuje ako parameter v *adresovej časti* inštrukcií.

Selektor je ukazovateľ do *tabuľky deskriptorov segmentov*. Selektor sa nachádza v *špeciálnom* registri a manipuluje sa s ním nezávisle od posunutia *Tabuľka deskriptorov segmentov* sa nachádza v hlavnej pamäti a sú v nej *deskriptory* všetkých segmentov.

Deskriptor segmentu je záznam, ktorý tieto informácie o segmente:

1. *Bázová adresa segmentu (Base)*. Je to adresa, od ktorej je segment uložený v hlavnej pamäti *(začiatok segmentu)*.
2. *Veľkosť segmentu (Limit)*. Pre každú inštrukciu alebo údaj v danom segmente musí platiť, že *Offset < Limit*.
3. *Atribúty segmentu (Attributes)*. *Patrí sem:*

* *Informácia o prítomnosti segmentu v hlavnej pamäti*. Ak sa sprístupňovaný segment nenachádza v hlavnej pamäti, použije sa položka *4.* a segment sa načíta z vonkajšej pamäte.

• *Informácia o type segmentu (vykonateľný segment, vykonateľný segment s možnosťou čítania, údajový segment len pre čítanie, údajový segment pre čítanie i zápis, zásobníkový segment atď.).*

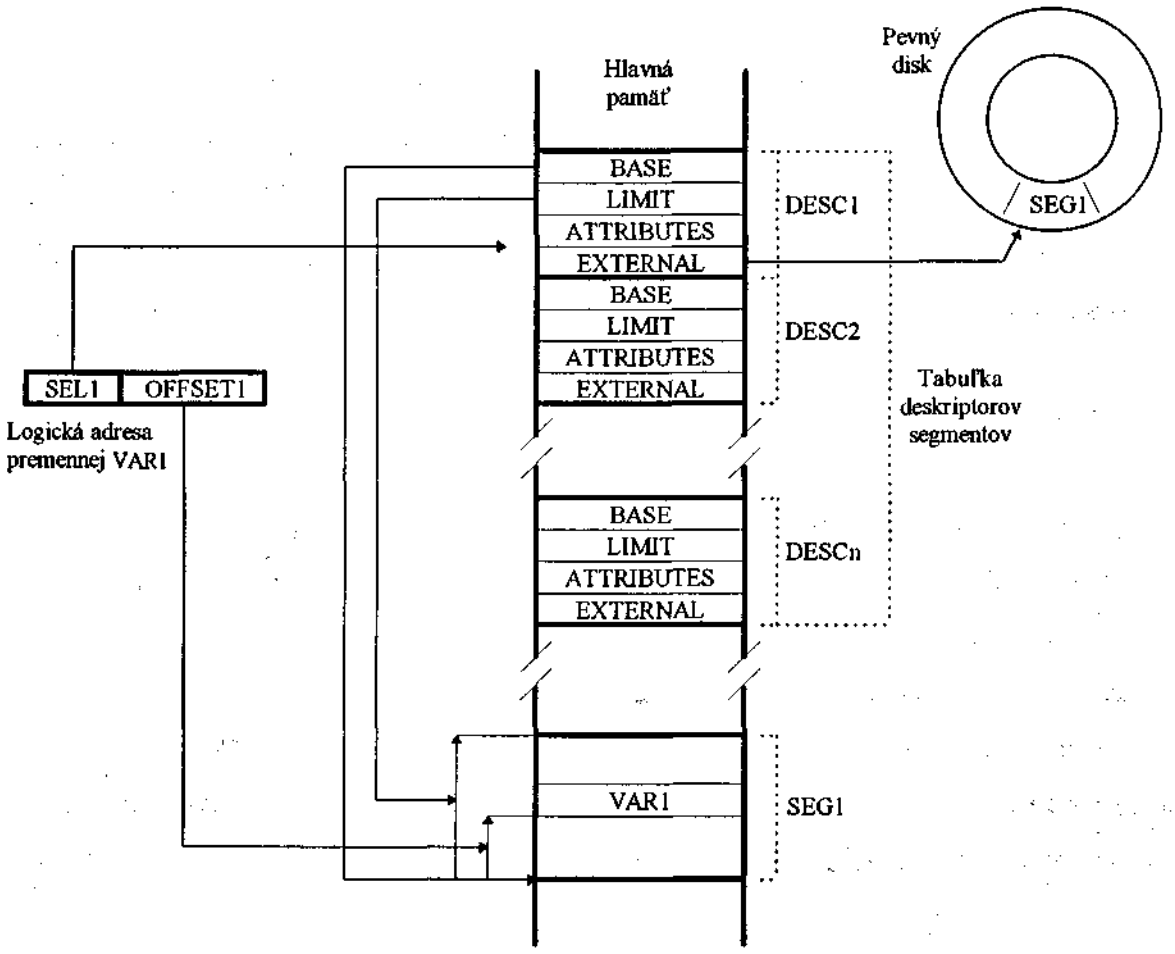
* *Informácia o privilegovanej úrovni segmentu* (má význam v prípade, ak *viacero programov* môže používať ten istý segment)

4. *Adresa segmentu vo vonkajšej pamäti (External).*

Na obr. 42 je zobrazený vzťah medzi deskriptorom segmentu a segmentom. Premenná *VAR1* patrí do segmentu *SEG1* a má logickú adresu *SEL1:OFFSET1*. Segment *SEG1* je opísaný deskriptorom *DESC1*.

Fyzická adresa sa vypočíta tak, že k hodnote *bázy* sa pripočíta hodnota *posunutia*.

Uvedený spôsob adresovania je tzv. *relatívne adresovanie*. Jeho výhodou je to, že segment je možné bez zmeny uložiť od ľubovoľnej adresy v hlavnej pamäti, stačí iba *zmeniť hodnotu bázy*.



OBR. 42. Vzťah medzi deskriptorom segmentu a segmentom

Segmentácia a využitie hlavnej pamäte

V prípade segmentácie prichádza k tzv. *externej fragmentácii*. Táto spočíva v tom, že po niekoľkonásobnom načítaní a vylúčení rôznych segmentov z hlavnej pamäte môže nastať situácia, že segmenty sú v hlavnej pamäti rozmiestnené tak nepriaznivo, že nemôžeme už načítať do pamäte ďalší segment, aj keď súčet voľnej pamäte je väčší, ako veľkosť segmentu, ktorý chceme načítať.

V tomto prípade je potrebné urobiť *preusporiadanie segmentov (kondenzácia)* - segmenty sa popresúvajú v hlavnej pamäti tak, aby voľná pamäť bola *spojitá*. Táto činnosť si však vyžaduje istý čas procesora, ktorý sa nedá využiť pre užitočný výpočet.

Stránkovanie

Pri *stránkovaní* je hlavná pamäť rozdelená na úseky rovnakej dĺžky, ktoré sa nazývajú *stránkové rámy (page frames)*. Stránkové rámy sú očíslované. Program i údaje sú takisto rozdelené na *úseky rovnakej dĺžky* - *stránky*. Veľkosť stránky je rovnaká, ako veľkosť stránkového rámu.

Logická adresa sa v prípade *stránkovania* skladá z dvoch častí, ktoré sú však na rozdiel od segmentovania spracovávané spoločne - *stránka* a *posunutie*.

Posunutie (offset) reprezentuje *vzdialenosť* inštrukcie alebo údaja od *začiatku* stránky.

Stránka (page) je ukazovateľ do *tabuľky deskriptorov stránok*.

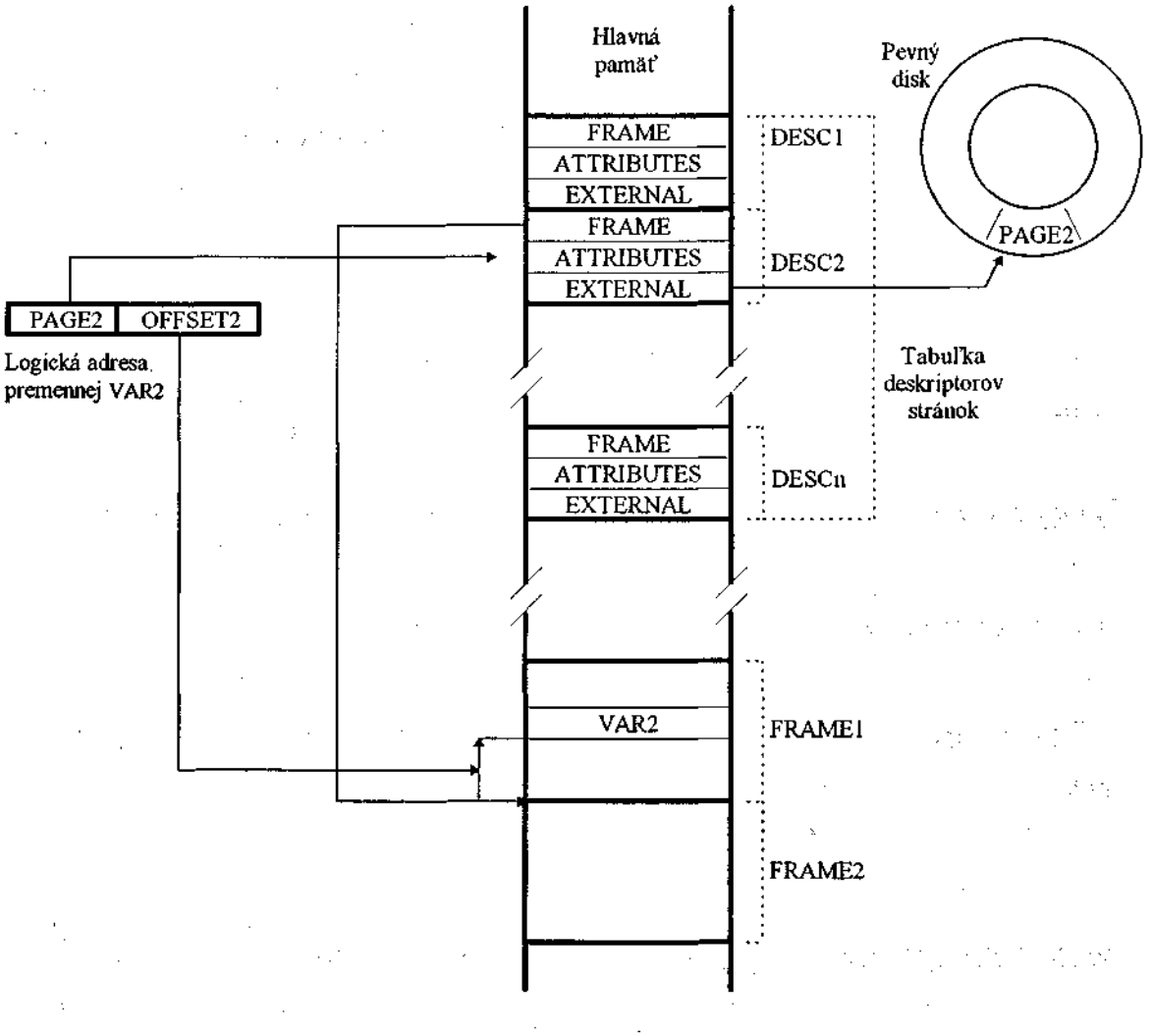
Tabuľka deskriptorov stránok sa nachádza v hlavnej pamäti a sú v nej *deskriptory* všetkých stránok.

Deskriptor stránky je záznam, ktorý obsahuje tieto informácie o stránke:

1. *Číslo stránkového rámu (Frame)*. Pretože všetky stránkové rámy majú rovnakú veľkosť, číslo stránkového rámu jednoznačne určuje umiestnenie stránky v hlavnej pamäti, t.j. adresu, od ktorej je stránka v hlavnej pamäti uložená *(začiatok stránky)*.
2. *Atribúty stránky (Attributes)*. Tieto sú rovnaké, ako atribúty segmentu.
3. *Adresa stránky vo vonkajšej pamäti (External)*.

Pretože všetky stránky majú rovnakú veľkosť, deskriptor stránky informáciu o veľkosti stránky *nemusí obsahovať*.

Na obr. 43 je zobrazený vzťah medzi deskriptorom stránky a stránkou. Premenná *VAR2* má logickú adresu *PAGE2 OFFSET2*.Patrí do stránky *PAGE2*, ktorá je opísaná deskriptorom *DESC2*. Táto stránka je momentálne uložená v stránkovom ráme *FRAMEJ*.



OBR. 43. Vzťah medzi deskriptorom stránky a stránkou

Situáciu, keď procesor chce sprístupňovať stránku, ktorá sa nenachádza v hlavnej pamäti, označujeme ako *výpadok stránky*.

Stránkovanie a využitie hlavnej pamäte

V prípade stránkovania prichádza k tzv. *internej fragmentácii*. Táto spočíva v tom, že veľkosť programov alebo údajov nie je vo všeobecnosti celočíselným násobkom veľkosti stránky. Posledná stránka potom nie je plne využitá. Čím viac programov alebo údajových štruktúr sa v pamäti nachádza, tým viac sa prejavuje nevyužitie hlavnej pamäte. Čím je väčšia

veľkosť stránky, tým je problém vypuklejší. Na druhej strane, ak zmenšíme veľkosť stránky, výrazne rastie tabuľka deskriptorov. Potrebné je zvoliť vhodný kompromis.

Okrem jednoduchého stránkovania sa používa aj *viacúrovňové stránkovanie,* prípadne sa kombinuje *segmentovanie a stránkovanie*. Vtedy hovoríme o *stránkovaných segmentoch.*

4.3.4.2 Technické prostriedky na podporu ochrany pamäte

Problém *ochrany pamäte* sa čiastočne rieši vo všetkých architektúrach, ktoré podporujú segmentovanie a stránkovanie. Vo všeobecnosti sa väčší stupeň ochrany poskytuje v systémoch so segmentovaním, kde *mechanizmus ochrany môže byť riadený programátorom,* ktorý rozdeľuje program a údaje do segmentov podľa vlastného uváženia.

Komplexné riešenie problému ochrany pamäte si vyžaduje spoluprácu technických a programových prostriedkov počítača.

Technické prostriedky súčasných procesorov *(napr. 80x86 v privilegovanom režime) bez účasti programátora kontrolujú,* či sa sprístupňovaný segment *nachádza* v *hlavnej pamäti,* či *adresa* sprístupňovanej inštrukcie alebo údaja *nepresiahla limit segmentu,* či je *použitie daného typu segmentu korektné (napr. či nedochádza k výberu inštrukcie z údajového segmentu)* a či je *oprávnené použitie daného segmentu (či má žiadateľ dostatočnú privilegovanú úroveň).*

Vykonávanie týchto činností priamo technickými prostriedkami umožňuje implementovať efektívny spôsob ochrany pamäte. Ak by sa totiž tieto činnosti vykonávali programovými prostriedkami, prichádzalo by k neúnosnému spomaleniu činnosti celého systému.

V prípade, ak technické prostriedky zistia porušenie ochrany pamäte, je vygenerovaná *výnimka* (pozri *Prerušovací systém procesora)* a úlohou *operačného systému* je daný problém vyriešiť.

4.3.5 Vyrovnávacia pamäť (cache)

Vyrovnávacia pamäť (cache) je rýchla pamäť rádovo menšej kapacity, akú má hlavná pamäť. *Slúži na preklenutie v podstate rádového rozdielu medzi prístupovou dobou registrov procesora a hlavnej pamäte.* Do vyrovnávacej pamäte sa presunie časť obsahu hlavnej pamäte a procesor potom sprístupňuje informácie z vyrovnávacej pamäte vyššou rýchlosťou ako z hlavnej pamäte.
Činnosť vyrovnávacej pamäte vychádza z predpokladu, že počas vykonávania programu sa inštrukcie i údaje vyberajú postupne za sebou, tak ako sú umiestnené v hlavnej pamäti. Z tohto dôvodu, ak procesor sprístupňuje nejakú bunku v hlavnej pamäti, do vyrovnávacej pamäte sa presunie nie iba táto jediná bunka, ale *celá skupina po sebe nasledujúcich buniek*. Táto skupina buniek, označovaná ako *riadok (line)*, tvorí základnú alokačnú jednotku vyrovnávacej pamäte.

Počas práce procesora s riadkom, ktorý je umiestnený vo vyrovnávacej pamäti, môže nastať situácia, že procesor modifikoval niektorú bunku v riadku. V tomto okamihu nastala *nekonzistencia* informácií medzi vyrovnávacou a hlavnou pamäťou. Túto situáciu rieši *riadiaci obvod vyrovnávacej pamäte (cache controller)* dvoma spôsobmi:

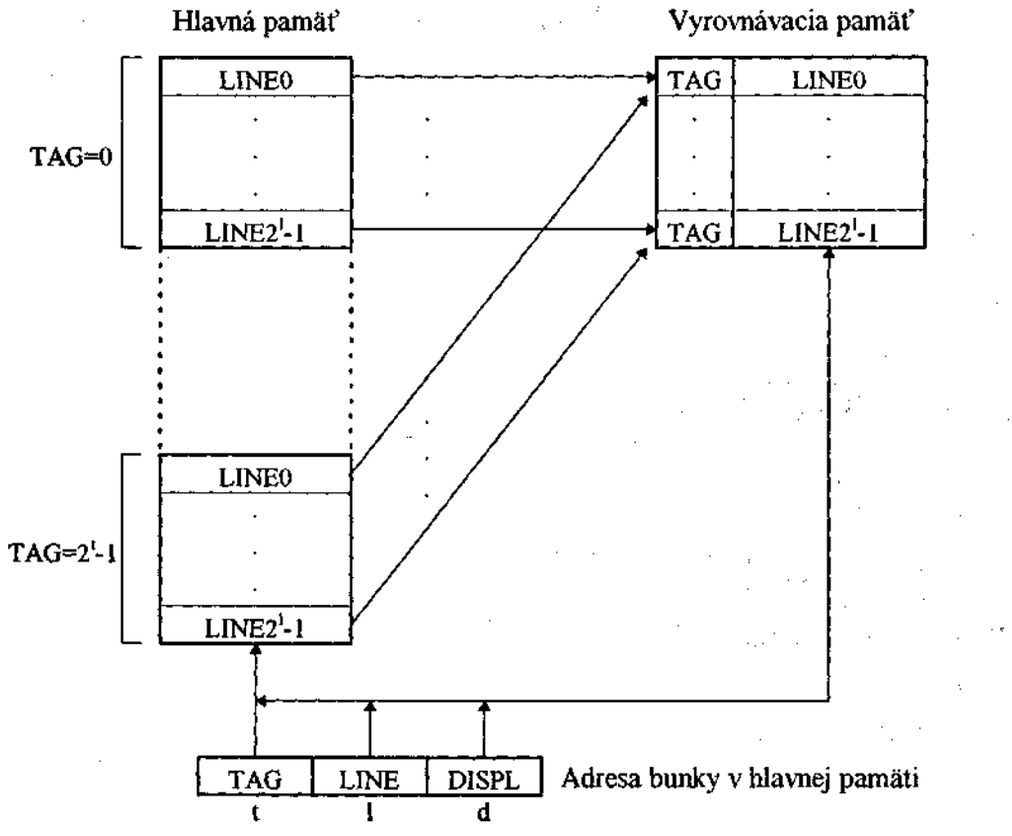
- Prvý spôsob, označovaný ako *zápis späť (copy-back)* spočíva v tom, že ak sa riadok vo vyrovnávacej pamäti už stane neaktuálny a do vyrovnávacej pamäte je treba naplniť nový riadok, celý modifikovaný starý riadok je prepísaný do hlavnej pamäte.

- Druhý spôsob, označovaný ako *zápis cez vyrovnávaciu pamäť (write-through)* rieši problém nekonzistencie tak, že kedykoľvek procesor modifikuje niektorú bunku vo vyrovnávacej pamäti, táto zmena sa súčasne uskutoční aj v hlavnej pamäti.

4.3.5.1 Mapovanie hlavnej pamäte do vyrovnávacej pamäte

Ako už bolo uvedené, vyrovnávacia pamäť má menšiu kapacitu ako hlavná pamäť. Treba preto nájsť spôsob, ako ľubovoľný riadok z hlavnej pamäte umiestniť do niektorého riadku vyrovnávacej pamäte. Tomuto spôsobu sa hovorí *mapovanie*.

Na obr. 40 vidíme princíp mapovania hlavnej pamäte do vyrovnávacej pamäte.



OBR. 40. Princíp mapovania hlavnej pamäte do vyrovnávacej pamäte

Hlavná pamäť je rozdelená na *2 ' blokov,* ktoré majú rovnakú veľkosť a organizáciu, ako vyrovnávacia pamäť.

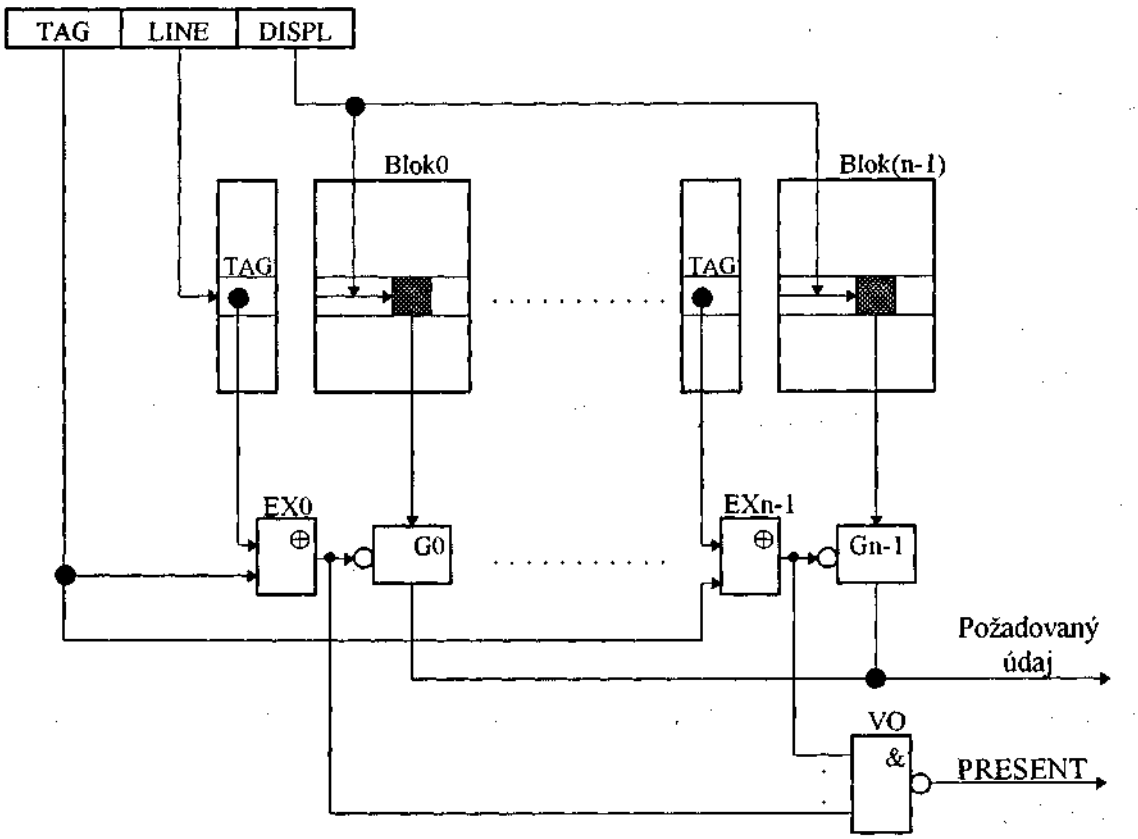
Adresa každej bunky v hlavnej pamäti je teraz rozdelená do *troch polí:*

I. Pole *DISPL* má *d* bitova udáva *posunutie bunky* v *riadku* od začiatku riadku.
2. Pole *LINE* má/ bitov a udáva, *do ktorého riadku bunka patrí.*
3. Pole *TAG* má *t* bitov a udáva, *do ktorého bloku bunka patrí.*

Modely vyrovnávacích pamätí

V súčasnosti sa používa niekoľko modelov vyrovnávacích pamätí, napr. *vyrovnávacia pamäť s priamym mapovaním (Direct Mapping Cache Memory), s množinou blokov (Set Associaíive Cache Memory), plne asociatívna vyrovnávacia pamäť (Fully Associative Cache Memory).*

Na obr. 41 je nakreslená vyrovnávacia pamäť s množinou blokov, ktorá sa skladá z *n* blokov.



OBR. 41. Vyrovnávacia pamäť s množinou blokov

Vyhodnocovací obvod (VO) zisťuje, či sa riadok s adresovanou bunkou nachádza v niektorom bloku vyrovnávacej pamäte. V prípade, ak tomu tak je, signál *PRESENT* je aktívny a príslušná bunka je cez zodpovedajúce hradlo *G* sprístupnená. Pre uloženie príznakov *TAG* sa používa *asociatívna pamäť (CAM),* čím je vyhodnotenie veľmi jednoduché a rýchle. Ak požadovaný

riadok vo vyrovnávacej pamäti nie je, *riadiaci obvod vyrovnávacej pamäte* musí zabezpečiť jeho načítanie z hlavnej pamäte. Ak vo vyrovnávacej pamäti nie je nijaký voľný riadok, ďalšou úlohou riadiaceho obvodu je vybrať, ktorý z riadkov bude zrušený.

Pre výber sa používajú najčastejšie dva algoritmy:

- *LRU (Least-Recently Used)* - bude vylúčený ten riadok, ktorý sa *najdlhší čas nepoužil.*

* *LFU (Least-Frequently Used)* - bude vylúčený *najmenej často používaný* riadok.

4.3.6 Vonkajšie pamäti

Vonkajšie pamäti slúžia na prechodné uchovanie informácií počas výpočtu a na archiváciu informácií. Nie sú súčasťou operačnej (hlavnej) pamäte počítača a procesor k nim pristupuje ako k *V/V zariadeniam*.
V ďalšom uvedieme základné informácie o najpoužívanejších vonkajších pamätiach.

4.3.6.1 Pružné disky (diskety)

Disketa je výmenné pamäťové médium. Je to tenký plastový kotúč, na povrchu ktorého je nanesená vrstva *magnetického materiálu.*

Údaje sa pri *zápise* na magnetické médium prostredníctvom *magnetickej hlavičky* zaznamenávajú ako *elementárne magnety,* ktoré sú istým spôsobom orientované. Pri *čítaní* sa pri otáčaní média indukujú v hlavičke *napäťové impulzy.*

Hlavička, ktorá slúži pre záznam a čítanie informácií, je pri čítaní alebo zápise v *priamom kontakte* s povrchom diskety. Informáciu je možné kedykoľvek zmazať alebo prepísať novou informáciou.

Priemer diskiet, používaných v súčasnosti, je *5.25"* a *3.5"*. Diskety s priemerom *5.25"* sú uložené v pružnom ochrannom obale, diskety s priemerom *3.5"* majú pevné plastové puzdro. Ide o ochranu pred mechanickým poškodením a prachom.
Disketa sa vkladá do *disketovej mechaniky*.

Organizácia údajov

Počet *strán* (povrchov) môže byť *1* alebo *2*. V súčasnosti sa používajú *dvojstranné* diskety, to znamená, že na obidvoch stranách majú nanesený magnetický materiál. V disketovej mechanike sú dve hlavičky, pre každú stranu diskety jedna.

Stopa je kruhová dráha na povrchu diskety, po ktorej sa kíže magnetická hlavička. Stopy sú číslované od *0,* pričom číslo *0* je pridelené krajnej stope na vonkajšom okraji diskety. Počet stôp závisí od *typu* diskety.

Všetky stopy sú rozdelené na úseky, ktoré sa nazývajú *sektory*. Údaje sú organizované po slabikách, počet slabík býva obyčajne v každom sektore rovnaký. Tak napr. v operačnom systéme *MS DOS* je v jednom sektore uložených *512 Bajtov*.

Pred použitím je potrebné disketu *naformátovať*, t.j. rozmiestniť na ňu značky, ktoré vyznačia začiatky jednotlivých sektorov na všetkých stopách. Diskety môžu byť už naformátované od výrobcu alebo sa použije špeciálny *formátovací program*. Pri rovnakom priemere môžu byť diskety naformátované na *rôznu* kapacitu.

Výpočet kapacity diskety

Kapacita diskety sa vypočíta ako súčin:

(počet strán) .(počet stôp) .(počet sektorov) .(počet bajtov na sektor)

V nasledujúcej tabuľke sú uvedené kapacity diskiet o priemere *5.25"* a *3.5"* pri zázname *512 B/sektor*. Označenie *DS/DD (Double Sided/Double Density)* značí obojstrannú disketu s dvojnásobnou hustotou záznamu, *DS/HD (Double Sided/High Density)* obojstrannú disketu s vysokou hustotou záznamu. Každú disketu je potrebné naformátovať na kapacitu, na akú je určená. Vnútiť diskete naformátovaním inú kapacitu, než na akú je určená (hoci i menšiu), môže znamenať stratu údajov v dôsledku nečitateľnosti záznamu.

Tabuľka 4. Kapacita diskiet

Priemer	Označenie	Počet stôp	Sektorov/stopu	Kapacita
5.25"	DS/DD	40	9	360 kB
5.25"	DS/HD	80	15	1.2 MB
3.5"	DS/DD	80	9	720 kB
3.5"	DS/HD	80	18	1.44 MB

4.3.6.2 Pevné disky

Pri pevných diskoch sa používa rovnaký princíp záznamu a čítania informácií, ako pri disketách. Na rozdiel od disketovej mechaniky je však mechanika pevného disku úplne *uzatvorená,* takže pamäťové médium, ktoré sa skladá z jedného alebo niekoľkých pevných diskov, *nie je výmenné.* V prípade viacerých diskov hovoríme o *diskovom zväzku.*

Magnetické hlavičky *sa nedotýkajú* priamo povrchu disku, ale plávajú vo vzdialenosti *niekoľkých mikrometrov* nad jeho povrchom. Pri vysokých prevádzkových otáčkach disku *(napr. 3600 . min^{-1})* by prítomnosť mechanickej nečistoty (v tomto prípade i prachovej častice) mala katastrofálne následky. Pevné disky sú tiež chúlostivé na otrasy počas prevádzky.

Magnetické hlavičky sú upevnené na držiaku, ktorému sa hovorí *vystavovacie ramienko* a pohybujú sa všetky spoločne nad rovnakým *valcom,* t.j. jednotlivé hlavičky nemôžu byť vystavované nezávisle.

Priemer používaných diskov je rovnaký, ako pri disketách, preto sú vonkajšie rozmery pevného disku v podstate rovnaké ako rozmery disketovej mechaniky.

Rýchlosť disku charakterizujú dva údaje. Prvý udáva, ako dlho trvá vyhľadanie určitej informácie na disku *(doba prístupu),* druhý udáva, ako rýchlo je disk schopný informáciu preniesť *(rýchlosť prenosu údajov).*

Doba prístupu sa skladá z dvoch častí. Najprv sa premiestni magnetická hlavička na príslušnú stopu *(doba vyhľadávania)* a potom sa čaká, kým sa pod hlavičkou bude nachádzať príslušný sektor *(rotačná čakacia doba).*

Doba prístupu je pri súčasných pevných diskoch menšia ako *10 ms*, rýchlosť prenosu údajov dosahuje okolo *1200 kB.s'.*

Organizácia údajov

Organizácia údajov je podobná, ako na disketách. Naviac sa tu používa výraz *valec (cylinder).*

Pod *valcom* rozumieme všetky stopy rovnakého priemeru, ktoré sú umiestnené pod sebou. Počet stôp pri pevných diskoch je podstatne väčší, ako pri disketách, býva ich aj *viac ako 1000.* Počet sektorov na stopu sa pohybuje *od 17* až do *viac ako 50.* Počet bajtov na sektor býva obyčajne rovnaký ako pri disketách, t.j. *512.*

Výpočet kapacity pevného disku

Kapacita disku sa vypočíta ako súčin:

(počet hlavičiek) .(počet valcov) .(počet sektorov) .(počet bajtov na sektor)

Krokovacie motorčeky a vychyľovacie cievky

Pre pohyb vystavovacieho ramienka v mechanike pevného disku sa používajú dva rôzne spôsoby. Prvý z nich je *krokovací motorček,* druhý spôsob je *vychyľovacia cievka.*

Krokovací motorček je napájaný elektrickými impulzmi a po každom impulze sa jeho hriadeľ pootočí o uhol, ktorému sa hovorí *krok*. Jedna celá otáčka hriadeľa sa môže skladať z *niekoľko sto* krokov. Okolo hriadeľa je namotaný pružný tenký kovový pásik, spojený s vystavovacím ramienkom. Keď sa hriadeľ otáča, pásik sa buď navíja alebo odvíja, čím sa pohybuje aj vystavovacie ramienko. Jeden impulz z riadiaceho obvodu disku pootočí hriadeľ motorčeka o jeden krok a hlavičky sa tým presunú o jeden valec.

Tento systém má však niekoľko nevýhod. Prvou z nich je to, že pásik sa časom môže natiahnuť, druhou je to, že krokovací motorček sa opotrebúva a môže niekedy preskočiť aj o viac ako jeden krok.

Vychyľovacia cievka má vo svojom strede kovové jadro. Keď sa cievke dodá elektrická energia, jadro sa vysúva alebo zasúva. Pretože je s ním spojené vystavovacie ramienko, súčasne sa pohybujú aj magnetické hlavičky. Ako však riadiaci obvod disku vie, ako ďaleko má jadro vysunúť ?
Pri krokovacom motorčeku je situácia jasná - jeden impulz znamená posun o jeden valec. Pri systéme s vychyľovacou cievkou je informácia o pozícii hlavičiek zakódovaná na disku spoločne s údajmi. Niektoré disky venujú tejto informácii jeden celý povrch, takže potom v údajoch o disku nájdeme *nepárny počet hlavičiek*. Neuvedená hlavička je použitá iba pre polohovacie informácie. Tomuto sa hovorí *jednoúčelový servomechanizmus*. Iné disky majú informáciu o polohe hlavičiek roztrúsené po celom disku a tak pre údaje používajú všetky povrchy a pri nich je udávaný *párny počet hlavičiek*. Vtedy sa hovorí o *vloženom servomechanizme*.

Systém s vychyľovacou cievkou je perspektívnejší. Je rýchlejší, a okrem toho ide o tzv. *uzavretý riadiaci systém*. Vystavovacie ramienko sa pohybuje dovtedy, kým sa nezíska informácia o správnej polohe hlavičiek. Naproti tomu systém s krokovacím motorčekom vychádza z predpokladu, že valec *20* bude vždy *20* krokov od valca *0*. Časom sa však vplyvom mechanického opotrebovania jeho pozícia stále vzďaľuje. Ďalšou významnou výhodou systému s vychyľovacou cievkou je to, že po vypnutí napájacieho napätia pružina automaticky presunie hlavičky k stredu disku - disk sa sám *zaparkuje.*

Kódovanie informácie

Ako už bolo uvedené, údaje sú na magnetickom médiu zaznamenané ako elementárne magnety. Pri otáčaní média sa v magnetickej hlavičke indukujú napäťové impulzy. Na reprezentáciu logických hodnôt na magnetických médiách sa teda používa *impulz (I)* resp. *neprítomnosť impulzu,* ktorú označíme ako *medzera (M).*

Najjednoduchším kódovaním by teda zrejme bolo kódovanie, keby hodnote *log. J* zodpovedal *impulz* a hodnote *log.0 medzera,* t.j.:

$$log.1 = I$$
$$log.O = M$$

Môže však nastať situácia, že za sebou nasleduje veľa hodnôt *log.O.* Toto by potom predstavovalo dlhý čas bez impulzov, čím vzniká vážny problém. Impulzy totiž okrem toho, že reprezentujú údaje, slúžia aj na synchronizáciu vnútorných hodín *riadiaceho obvodu disku* s údajmi. Údaje sa získavajú z disku v závislosti od času, takže doba, ktorá bola použitá na *záznam údajov,* musí byť rovnaká, ako doba na *čítanie údajov.* Impulzy pomáhajú údaje a riadiaci obvod disku synchronizovať. Preto je potrebná taká kódovacia schéma, ktorá zabezpečí, že pri ľubovoľných údajoch nebude riadiaca jednotka disku veľmi dlho bez impulzu. Riešením je vloženie *synchronizačných bitov* medzi *údajové bity.*

Kódovanie MFM

Modifikovaná frekvenčná modulácia (MFM - Modified Frequence Modulation) kóduje takto:

log.1 = MI
log.O = ak je pred ňoulog.O, tak IM
log.O = ak je pred ňou log. 1, tak MM

V kódovaní *MFM* je minimálny počet medzier *J* a maximálny počet medzier *3.* Hovoríme, že *MFM* má *1,3 obmedzenú dĺžku chodu (1,3 RLL - Ruň Lengíh Limited). MFM* sa používa pri kódovaní *pružných diskov* a pri starších pevných diskoch.
V súčasnosti sa používa kódovanie *2,7 RLL* (alebo skrátene iba *RLL),* ktoré umožňuje ešte väčšiu hustotu údajov a tým lepšie využitie diskového média. Kódovacia schéma je zložitejšia. V kódovaní *RLL* majú disky okrem väčšej kapacity aj vyššiu prenosovú rýchlosť.

4.3.6.3 Kazetovo-páskové pamäti

Kazetovo-páskové pamäti využívajú magnetický spôsob záznamu informácií. Údaje sa zaznamenávajú na kazetu s magnetickou páskou *(cartridge).* Používa sa viac typov kaziet, ktoré sa líšia svojou veľkosťou a kapacitou. Kapacita jednej kazety sa bežne pohybuje v *stovkách MB* a kazetovo-páskové pamäti sa typicky používajú ako veľkokapacitné archívne pamäti.

Nevýhodou kazetovo-páskových pamätí je *nízka záznamová rýchlosť* a najmä *sériový prístup k údajom,* ktorý tieto pamäti vylučuje z bežného použitia počas normálnej prevádzky počítača.

4.3.6.4 CD-ROM pamäti

Médium *CD-ROM pamäte* je rovnako ako disketa *výmenné.* Na základnej podložke je nanesená *odrazová (reflexná) vrstva* z hliníka, ktorá je prekrytá *maskovacou vrstvou (land).* Na tejto sa nachádza vrchná priehľadná *ochranná vrstva.* Informácie sú zaznamenané pomocou *jamiek (pits)* v maskovacej vrstve. Pri čítaní informácie *laserový lúč* šírky asi *1.6 nm* sleduje *stopu* na médiu. V prípade, ak je v maskovacej vrstve jamka, lúč sa odrazí a je prijatý *optickým snímačom.* Ide o bezkontaktné snímanie, takže ani médium, ani snímač sa neopotrebúvajú.

Médium sa vkladá do *mechaniky CD-ROM.* Pôvodne sa *CD (Kompaktný disk - Compact Disc)* používal pre *záznam* zvuku a až potom sa stal veľkokapacitným nosičom údajov vo

výpočtovej technike. V súčasnosti sa používa ako inštalačné médium veľkých softvérových produktov, na archiváciu údajov a v *multimediálnych* aplikáciách na uloženie textu, digitalizovaného zvuku a obrazu.

Okrem médií *CD-ROM,* na ktorých je informácia zaznamenaná priamo u výrobcu, existujú jedenkrát programovateľné médiá *CD-WORM (Write Once Read Many)* a viackrát prepisovateľné *magneticko-optické disky,* ktoré si môže naprogramovať používateľ vlastnými údajmi v špeciálnych mechanikách. Na čítanie obidvoch uvedených *CD* nosičov je pritom možné použiť štandardné *CD-ROM* mechaniky.

Organizácia údajov

Na *CD,* ktorý má štandardný priemer *12 cm,* sa nachádza *jediná špirálovitá stopa* dĺžky zhruba *5 km,* ktorá má asi *20 000* závitov.

Organizácia údajov na *AUDIO-CD* a *CD-ROM* je prakticky rovnaká. Uložené údaje je možné prehrávať asi *J hodinu = 60 minút.* Každá minúta záznamu sa delí na *60 sekúnd,* každá sekunda má *75 blokov,* očíslovaných *0* až *74.* Každý blok má *2352 bajtov.* Kým pri *A UD10-CD* je celá táto kapacita venovaná údajom, pri *CD-ROM (Mód 1)* je pre údaje v bloku venovaných iba *2048 bajtov* a zvyšok slúži na synchronizáciu a opravy chýb.

Údaje sa prenášajú rovnakou rýchlosťou, či ide o blok z vonkajšieho alebo vnútorného závitu. Pri konštantnej rýchlosti prenosu sa preto *mení rýchlosť otáčania média.* Otáčky sa menia v rozmedzí *200* až *530* za minútu.

Výpočet kapacity CD-ROM

Kapacitu CD-ROM vypočítame ako súčin:

(celkový počet sekúnd). (počet blokov). (počet bajtov na blok)

Pri uvedených parametroch je kapacita viac ako *600 MB.*

4.4 **Vstupno/výstupný** podsystém počítača

Vstupno/výstupný podsystém počítača slúži na vstup, resp. na výstup údajov. Musí umožniť komunikáciu procesora s rôznymi *vstupnými a výstupnými (periférnymi) zariadeniami,* napr. typickým vstupným zariadením je *klávesnica,* typickým výstupným zariadením *monitor* alebo *tlačiareň.* Niektoré periférne zariadenia slúžia na vstup aj výstup údajov, napr. *pevné disky.*

4.4.1 Pripojenie periférnych zariadení k zbernici počítača

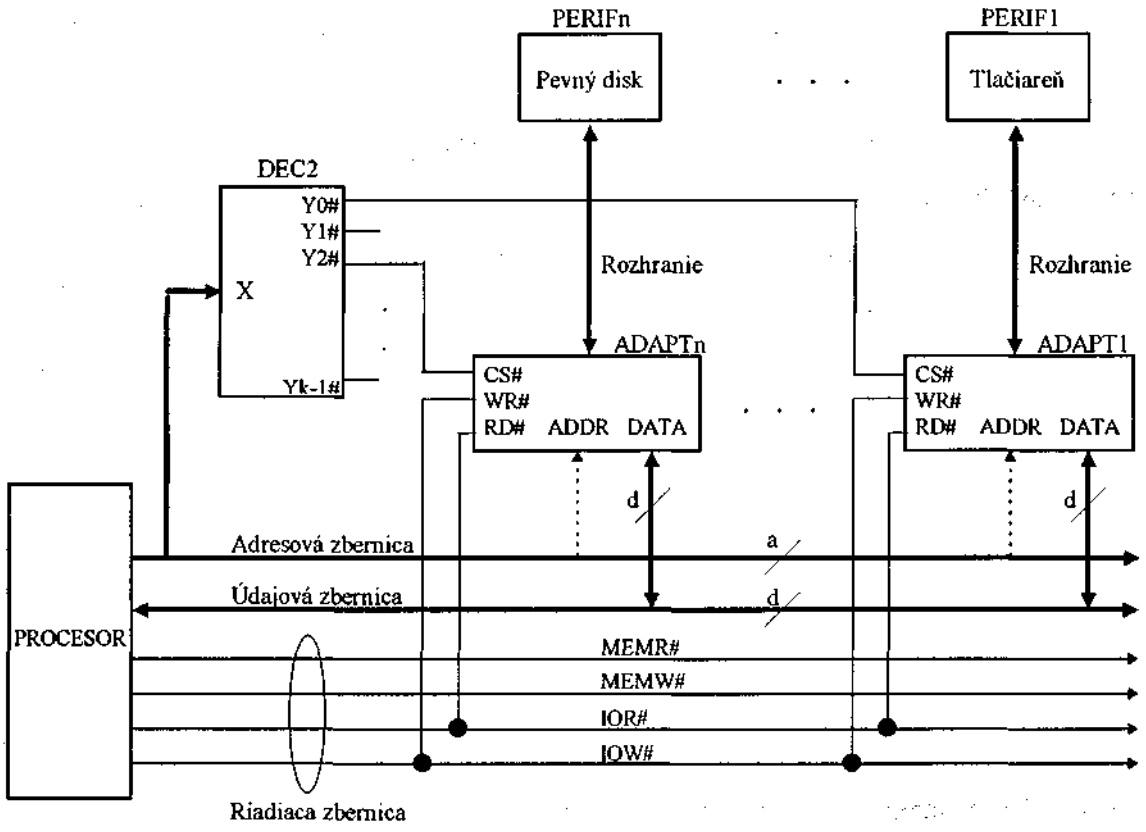
Periférnych zariadení, podobne ako pamäťových obvodov, môže byť k zbernici počítača pripojených niekoľko. Z toho vyplýva, že procesor musí byť schopný rozlíšiť, s ktorým periférnym zariadením bude pracovať. Pri vlastnej práci s periférnym zariadením nesmie prísť ku konfliktom, či už s pamäťou, alebo s inými periférnymi zariadeniami.

Na obr. **44** je naznačené principiálne pripojenie periférnych zariadení k zbernici počítača.

Ako z obrázku vidíme, periférne zariadenia *(PERIF1 ... PERIFn)* sú pripojené k zbernici prostredníctvom svojich *adaptérov (ADAPT1 až ADAPTn).*

Dekóder *DEC2* na základe aktuálnej adresy na adresovej zbernici počítača určuje, s ktorým adaptérom sa práve pracuje. Jeho úloha je podobná, ako úloha dekódera z obr. 38 - vyberá práve jeden aktívny adaptér, čím zabraňuje vzájomným konfliktom adaptérov na údajovej zbernici. Pripojenie adresy k adaptérom je naznačené čiarkovane - využíva sa iba v prípade, ak adaptér obsahuje *viacero interných registrov* na výber práve jediného registra, s ktorým sa pracuje.

Vylúčenie konfliktov medzi adaptérmi a pamäťou je zabezpečené tým, že adaptéry a pamäťové obvody majú *vlastné riadiace signály,* ktoré nikdy nie sú aktívne súčasne. Pamäťové obvody používajú na zápis resp. čítanie signály *MEMW#* resp. *MEMR#,* adaptéry používajú signály *IOW#* a *IOR#.*



OBR. 44. Pripojenie periférnych zariadení k zbernici počítača

4.4.2 Komunikácia procesora s adaptérom periférneho zariadenia

Adaptér vytvára nevyhnutné *rozhranie* medzi zbernicou počítača a periférnym zariadením, pretože priame pripojenie periférneho zariadenia na zbernicu počítača vo všeobecnosti nie je možné (odlišné napäťové úrovne, výkonové požiadavky, spôsob činnosti, spôsob prenosu údajov, bezpečnosť atď.).

Adaptér môže byť jednoduchý (napr. na pripojenie signalizačného prvku nám môže stačiť obyčajný *register),* ale môže to byť aj značne zložitý obvod (napr. *videoadaptér* na pripojenie *grafického monitora).*

Adaptéry sú obyčajne *programovateľné,* takže procesor pred vlastným prenosom údajov vyšle najprv do adaptéra *riadiace slová* a z adaptéra okrem údajov môže načítať *stavové slová.*

Treba si preto uvedomiť, že nie každý zápis/čítanie do/z adaptéra znamená zápis alebo čítanie z pripojeného periférneho zariadenia.

Rozhranie je vo veľkej väčšine prípadov v súčasnosti *štandardizované,* čo umožňuje k počítaču pripojiť periférne zariadenia od ľubovoľného výrobcu, ktorý toto rozhranie rešpektuje.
Neštandardné rozhrania sa obyčajne vytvárajú pri použití počítača na špeciálne aplikácie, napr. *na riadenie technologických procesov, vo vnorených systémoch* a pod.

Komunikácia procesora s adaptérom na najnižšej úrovni

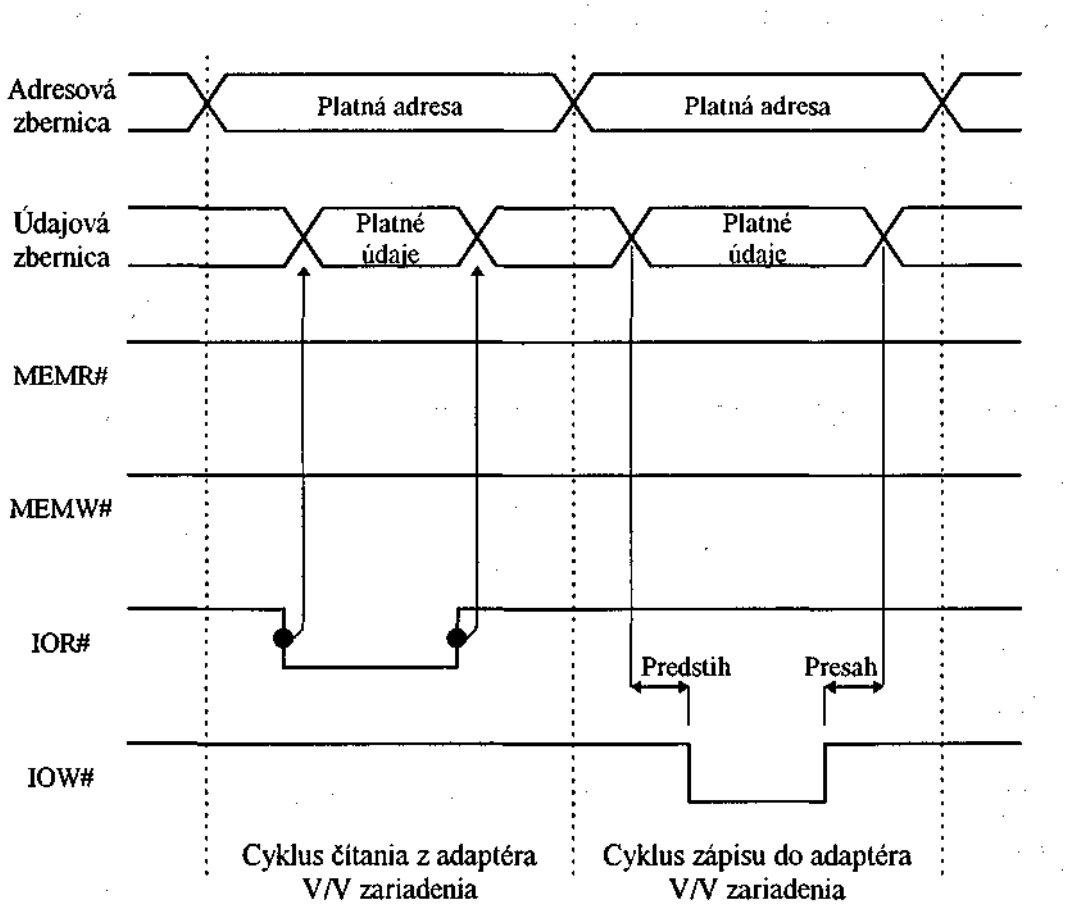
- *Čítanie z adaptéra*. Procesor musí vyslať na adresovú zbernicu adresu adaptéra, z ktorého chce načítať informáciu a nastaviť aktívnu úroveň signálu pre čítanie *IOR#*. Aktívna úroveň signálu musí trvať dostatočne dlhý čas. Po istom čase (doba prístupu) adaptér vyšle na údajovú zbernicu platnú informáciu.

- *Zápis do adaptéra*. Procesor musí vyslať na adresovú zbernicu adresu adaptéra, do ktorého chce zapísať a na údajovú zbernicu vyslať platné údaje. Potom môže nastaviť do aktívnej úrovne signál pre zápis *IOW#*. Je potrebné dodržať jednak istú minimálnu dobu trvania tohto signálu, ako aj dostatočný predstih a presah údajov voči tomuto signálu.

Na obr. 45 sú nakreslené typické signálové sledy pre cyklus čítania a zápisu z/do adaptéra periférneho zariadenia *(PZ)*.

4.4.3 Štandardné rozhrania na pripojenie periférnych zariadení

Pre štandardné rozhranie sú definované *použité napäťové úrovne, konektory, spôsob prenosu, protokoly,* atď. V ďalšom sa budeme stručne zaoberať dvoma typickými štandardnými rozhraniami - paralelným rozhraním *CENTRONICS* a sériovým rozhraním *RS232C*.



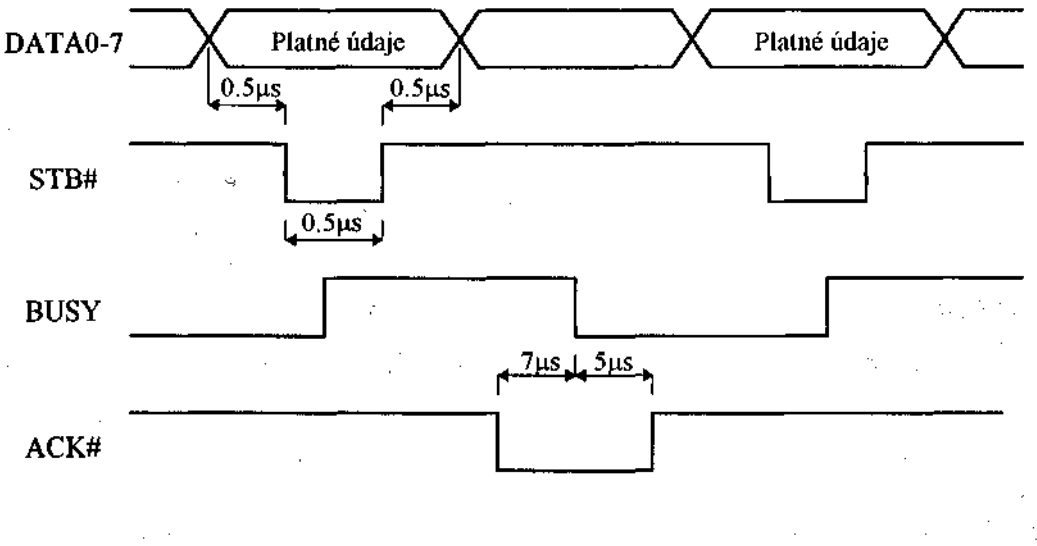
OBR. 45. Signálové sledy pre čítanie a zápis z/do adaptéra PZ

4.4.3.1 Paralelné rozhranie CENTRONICS

Rozhranie *CENTRONICS* je štandardné *paralelné rozhranie.* Šírka prenášaných údajov je *8 bitov.* Okrem prenosu údajov medzi *vysielačom* (adaptérom) a *prijímačom* (periférnym zariadením) sa pri komunikácii používajú aj *riadiace signály.* Tieto slúžia na vzájomnú *synchronizáciu* vysielača a prijímača.

Rozhranie *CENTRONICS* sa typicky používa pre pripojenie *tlačiarní.*

Na obr. 46 je nakreslený principiálny signálny sled prenosu jednej slabiky prostredníctvom rozhrania *CENTRONICS.* Vyznačené sú aj minimálne doby trvania, predstihu a presahu jednotlivých signálov.



OBR. 46. Prenos jednej slabiky cez rozhranie CENTRONICS

Vysielač vyšle na údajové vodiče *(DATA0-7)* slabiku, ktorú chce preniesť do prijímača a vygeneruje impulz definovanej dĺžky *STB# (STROBE - navzorkovanie údajov).* Týmto impulzom sa vysielané údaje zapíšu do záchytného registra v prijímači.

Prijímač reaguje takým spôsobom, že nastaví do aktívnej úrovne signál *BUSY (Obsadenosť prijímača),* ktorým oznamuje vysielaču, že je zamestnaný spracúvaním prijatého údaja a nie je schopný prijať ďalší údaj. Keď prijatý údaj prijímač spracuje (napr. tlačiareň vytlačila prijatý znak), vráti signál *BUSY* do neaktívnej úrovne a potvrdí prenos vygenerovaním impulzu definovanej dĺžky *ACK# (Acknowledge - potvrdenie).* Až potom môže nasledovať prenos ďalšej slabiky.

Použité napäťové úrovne všetkých signálov sú úrovne *TTL,* čím je daná nevyhnutnosť krátkej vzdialenosti medzi vysielačom a prijímačom (rádovo *jednotky metrov).*

4.4.3.2 Sériové rozhranie RS232C

Rozhranie *RS232C* je štandardné *sériové rozhranie.* Údaje sa medzi vysielačom a prijímačom prenášajú v sériovom tvare ako *postupnosť bitov* po jedinom vodiči. Pretože po zbernici

počítača sa údaje prenášajú v paralelnom tvare, adaptér musí realizovať pri vysielaní z počítača prevod údajov z paralelného na sériový tvar a pri príjme prevod údajov zo sériového na paralelný tvar. Na synchronizáciu vysielača a prijímača sú opäť použité riadiace signály.

Používajú sa dva typy prenosu - *synchrónny* a *asynchrónny*.

Pri *synchrónnom prenose* sa každý prenášaný bit *vzorkuje špeciálnym hodinovým impulzom,* slabiky sa prenášajú *bez prerušenia* jedna za druhou a nevykonáva sa žiadna kontrola správnosti prenosu na úrovni slabiky.

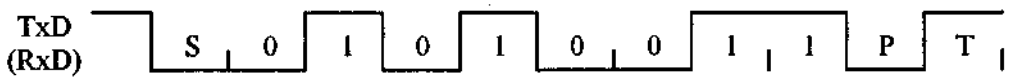
Pri *asynchrónnom prenose* sa po údajovom vodiči okrem údajových bitov prenášajú aj *špeciálne bity,* ako sú *štart-bit, paritný bit* a *stop-bit.* Rýchlosť prenosu je štandardizovaná (napr. *1200, 2400, 4800, 9600, 19200 bit.s'),* takže prenos každého bitu trvá presne určený časový okamih. Neaktívna úroveň na údajovom vodiči je *log. 1.*

Prenos slabiky je odštartovaný *štart-bitom,* ktorý začína zmenou z *log. 1* do *log. 0* a úroveň *log.0* potom trvá dobu prenosu jedného bitu. Potom nasledujú *údajové bity,* najskôr *nultý bit,* potom *prvý bit* atď. Používa sa 7 alebo *8* údajových bitov. Za posledným údajovým bitom nasleduje *paritný bit,* a to buď pre *párnu* alebo *nepárnu* paritu.

Párna parita znamená, že vysielač doplní paritným bitom počet jednotiek v slabike na *párny,* pri *nepárnej parite* na *nepárny.* Prijímač po prijatí údajovej slabiky nezávisle určí paritu tejto slabiky a porovná ju s prijatým paritným bitom. Paritný bit je voliteľný, ak chceme kontrolovať správnosť prenosu už na úrovni jednotlivých slabík.

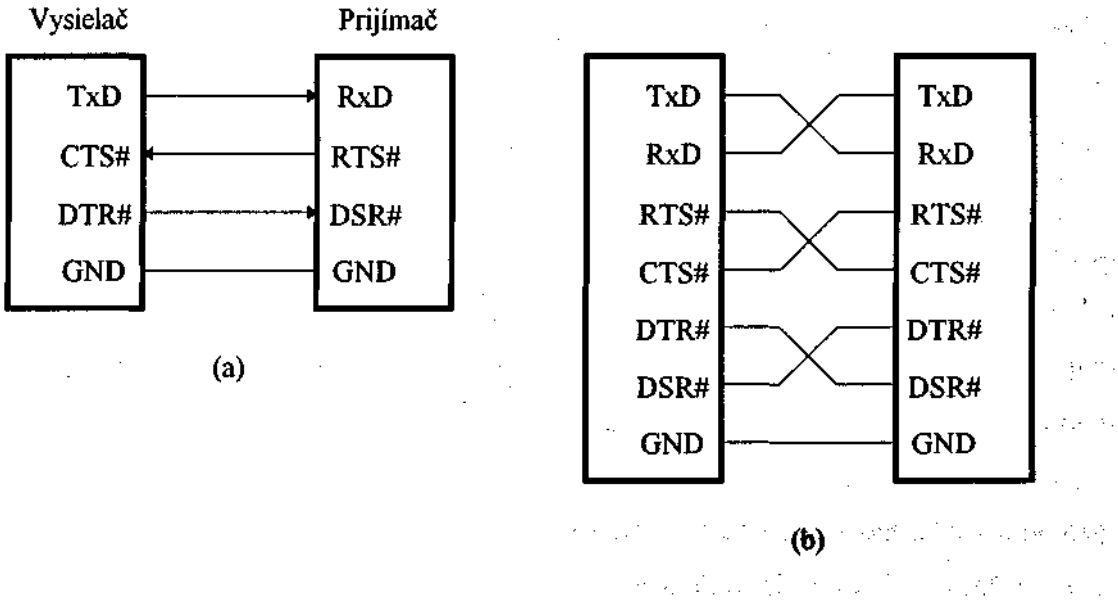
Prenos slabiky je ukončený *stop-bitom,* čo je vlastne *podržanie* úrovne *log.1* na údajovom vodiči na dobu prenosu jedného bitu. Používa sa *jeden, jeden a pol* alebo *dva* stop-bity. Až za stop-bitom môže nasledovať prenos ďalšej slabiky, t.j. nový štart-bit.

Na obr. 47 je naznačený prenos slabiky *11001010,* ak je použitá párna parita a jeden stop-bit. Štart-bit je označený *S,* paritný bit *P* a stop-bit *T.* Pretože máme párnu paritu, paritný bit je nulový (prenášaná slabika má párny počet jednotkových bitov).



OBR. 47. Prenos jednej slabiky pri asynchrónnom prenose

Vzájomné prepojenie vysielača a prijímača je nakreslené na obr. 48a. Pre ich synchronizáciu pri asychrónnom prenose sú použité špeciálne riadiace signály.



OBR. 48. Prepojenie vysielača a prijímača rozhraním RS232C

Signál *TxD (Transmit Data - vysielané údaje)* vysielača je pripojený na signál *RxD (Receive Data - prijímané údaje)* prijímača. Prijímač žiada vysielač o vyslanie údajov signálom *RTS (Request To Send - výzva k vysielaniu),* ktorý je pripojený na signál *CTS (Clear To Send)* vysielača. Vysielač oznamuje svoju pripravenosť k prenosu signálom *DTR (Data Terminal Ready),* ktorý je pripojený na signál *DSR (Data Set Ready)* prijímača

Prenos sa uskutoční iba vtedy, ak je vysielač aj prijímač pripravený. Tým je zabezpečené, že žiadne údaje sa počas prenosu nestratia (napr. ak prijímač nestíha spracovať údaje tak rýchlo

za sebou, ako ich je schopný vysielač vysielať, jednoducho po prijatí údaja prestane žiadať ďalší údaj a vysielač musí čakať).

Uvedený spôsob prepojenia sa používa pri *simplexnom prenose,* kedy je jednoznačne určené, kto je vysielačom a kto prijímačom a toto usporiadanie nie je možné zmeniť. Rozhranie *RS232C* však býva často realizované ako *obojsmerné,* t.j. zariadenie môže údaje nielen vysielať, ale aj prijímať. Samozrejme, potom obsahuje všetky signály vysielača aj prijímača. Ak sú navzájom prepojené dve takéto zariadenia, je možné vykonávať dva spôsoby prenosu - *poloduplexný (half duplex)* a *duplexný (full duplex, plný duplex)* prenos.

Poloduplexný prenos sa vyznačuje tým, že každé zariadenie môže tak vysielať, ako aj prijímať údaje, ale nie naraz - v istom časovom úseku jedno zariadenie pracuje ako vysielač a druhé ako prijímač, v inom časovom úseku tomu môže byť naopak.

Duplexný prenos sa vyznačuje tým, že zariadenie môže naraz pracovať aj ako vysielač, aj ako prijímač. Na obr.48 b je nakreslené prepojenie pri duplexnom prenose.

Napäťové úrovne rozhrania *RS232C* nie sú úrovne *TTL,* ale používa sa napätie *oboch polarít* a *väčších hodnôt.* Adaptér teda musí obsahovať *prevodník napäťových úrovní.*

Úrovni *log.0* zodpovedá napätie z intervalu *<+3 V; + 15 V>,* úrovni *log.1* napätie z intervalu *<-3 V; -15 V>.* Z tohto dôvodu je možné prepojiť vysielač a prijímač na väčšie vzdialenosti *(15 až 20 m).*

4.4.4 Spojenie počítača s technologickým prostredím

V prípade, že sa počítač používa na riadiacu aplikáciu alebo je súčasťou vnoreného systému, je priamo spojený s technologickým prostredím. Výstupné periférne zariadenia sú v tomto prípade *akčné členy (Actuators),* vstupné periférne zariadenia sú *senzory (Sensors).* Prostredníctvom akčných členov počítač *vstupuje* do prostredia, prostredníctvom senzorov *načítava stavové informácie* z prostredia. Niekedy počítač obsahuje iba senzory (napr. ak je použitý na zber údajov na ďalšie spracovanie), inokedy iba akčné členy (riadenie *bez spätnej*

väzby). Iba v prípade, ak počítač obsahuje akčné členy spolu so senzormi, je možné vykonávať *riadenie so spätnou väzbou*, ktoré sa vyznačuje tým, že počítač môže na základe stavovej informácie sledovať výsledok svojho riadiaceho zásahu a korigovať ho.

4.4.4.1 Vstup a výstup logických a číslicových signálov

Ako *logický signál* budeme označovať signál, ktorý nesie iba *dvojstavovú informáciu,* takže na jeho prenos stačí *jednobitový kanál.*

Číslicový signál nesie *viacstavovú informáciu,* takže na jeho paralelný prenos je nutné použiť *viacbitový kanál.* Počet stavov, ktoré je možné zakódovať do *n-bitov,* je *2".*

Realizovať samostatný adaptér na prenos každého logického signálu je veľmi neefektívne, preto sa logické signály združujú do skupín, ktoré prenáša jeden adaptér. Ak máme *n-bitový* adaptér, môžeme ním prenášať *n* logických signálov. Rovnakým adaptérom je potom možné prenášať *n-bitový* číslicový signál. Z tohto dôvodu budeme preto v ďalšom hovoriť už iba o prenose číslicových signálov.

Kvôli jednoduchosti budeme predpokladať, že senzor má na svojom (číslicovom) výstupe, ktorý je pripojený na vstup adaptéra, signály o úrovni *TTL*. Podobne nech aj akčný člen vyžaduje pre svoje ovládanie signály o úrovni *TTL*.

Vstup číslicových signálov

V prípade, ak je šírka slova senzora *menšia* alebo *rovnaká,* ako šírka údajovej zbernice počítača, adaptér môže byť veľmi jednoducho realizovaný *oddeľovačom s troj stavovým výstupom*.
Na vstup oddeľovača budú pripojené výstupy zo senzora, výstup adaptéra bude pripojený na údajovú zbernicu počítača.

J

Ak sa výstup zo senzora môže počas načítavania zmeniť, vhodnejšie je použiť *záchytný register* a načítanie riešiť v dvoch krokoch. V prvom kroku procesor najprv navzorkuje hodnotu zo senzora do registra a v druhom kroku načíta z registra platný údaj.

Ak je šírka slova senzora *väčšia* ako šírka údajovej zbernice počítača, adaptér je potrebné realizovať viacerými registrami *so spoločným vzorkovaním*. Procesor potom v prvom kroku navzorkuje celý údaj zo senzora do týchto registrov a v ďalších krokoch z nich postupne načíta platné údaje.

Výstup číslicových signálov

Ak je šírka slova akčného člena *menšia* alebo *rovnaká,* ako šírka údajovej zbernice počítača, adaptér môže byť veľmi jednoducho realizovaný prostredníctvom *registra*. Výstupy registra sú pripojené priamo na vstup akčného člena.

Ak je šírka slova akčného člena *väčšia* ako šírka údajovej zbernice počítača, adaptér je potrebné realizovať viacerými registrami *so spoločným ovládaním výstupu*. Procesor najskôr požadovaný údaj postupne zapíše do všetkých registrov a potom uvoľní spoločný výstup registrov, čím sa celé slovo pripojí k akčnému členu naraz.

4.4.4.2 Vstup a výstup analógových signálov

Priame spracovanie analógových hodnôt v číslicovom počítači nie je možné, pretože číslicový počítač vie pracovať iba s číslicovými premennými. Analógové údaje je preto potrebné transformovať na číslicové.
Pretože hodnota analógového údaja sa v čase mení, treba si uvedomiť, že číslicový počítač bude pracovať iba s *okamžitými hodnotami analógových údajov,* t.j. hodnotami, ktoré získal v istých bodoch *diskrétneho času*. Týmto hodnotám hovoríme *vzorky.*

Pre vstup analógových údajov do číslicového počítača sú realizované *analógovo/číslicové (A/D) prevodníky,* pre výstup analógových údajov z počítača *číslicovo/analógové (D/A) prevodníky*.

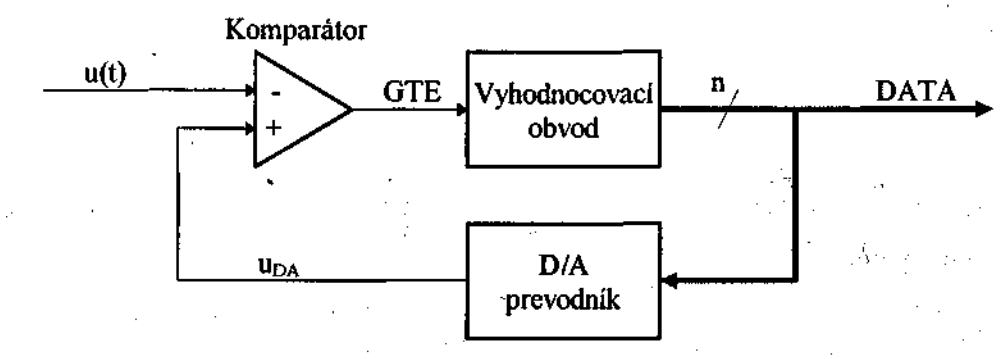
Analógovo/číslicové prevodníky

Na ilustráciu uvedieme dva typické *A/D* prevodníky - *prevodník s postupným prevodom* a j *paralelný prevodník*.

Prevodník s postupným prevodom

Prevodník s postupným prevodom sa vyznačuje tým, že prevod analógového údaja na číslicový sa vykonáva v niekoľkých krokoch.

Na obr. 49 je bloková schéma A/D prevodníka s postupným prevodom.



OBR. 49. A/D prevodník s postupným prevodom

u(t) - hodnota vstupnej analógovaj veličiny (v našom prípade *napätie).* Kvôli jednoduchosti predpokladajme, že počas prevodu sa táto hodnota nemení. Nech všetky hodnoty *u(t) > 0.*

u_{DA} - výstupné napätie D/A prevodníka.

DATA - vzorka, získaná konkrétnym prevodom.

Používajú sa dve základné metódy prevodu:

- *Metóda postupných prírastkov* sa vyznačuje tým, že vyhodnocovací obvod začína od *najnižšej hodnoty* (v našom prípade číslo *0)* a postupne po *najnižšom prírastku* (pripočítava číslo *J)* zvyšuje hodnotu *DATA.* Komparátor porovnáva veľkosť vstupného napätia *u(t)* s napätím u_{DA} , ktoré sa získalo číslicovo/analógovým prevodom číslicového údaja *DATA.* V okamihu, keď $u_{DA} \geq u(t)$ (signál *GTE* nadobudne aktívnu úroveň), je prevod ukončený a hodnota *DATA* reprezentuje *číslicovú hodnotu vstupného napätia.* Výhodou tejto metódy je jednoduchý vyhodnocovací obvod, nevýhodou je to, že ak máme *n-bitový* prevodník, prevod sa môže v najnepriaznivejšom prípade skladať až z 2" krokov.

- *Metóda postupnej aproximácie* pracuje takým spôsobom, že vyhodnocovací obvod začína od *najvyššieho bitu,* ktorý nastaví na 7 a kontroluje sa výstup komparátora. V prípade, ak je signál *GTE* aktívny, vyhodnocovací obvod vráti hodnotu tohto bitu na *0.* Ak je signál *GTE* neaktívny, hodnota bitu sa ponechá *J.* Vyhodnocovací obvod potom nastaví ďalší bit v poradí na *J,* otestuje signál *GTE* atď. Po *n* krokoch je prevod ukončený.

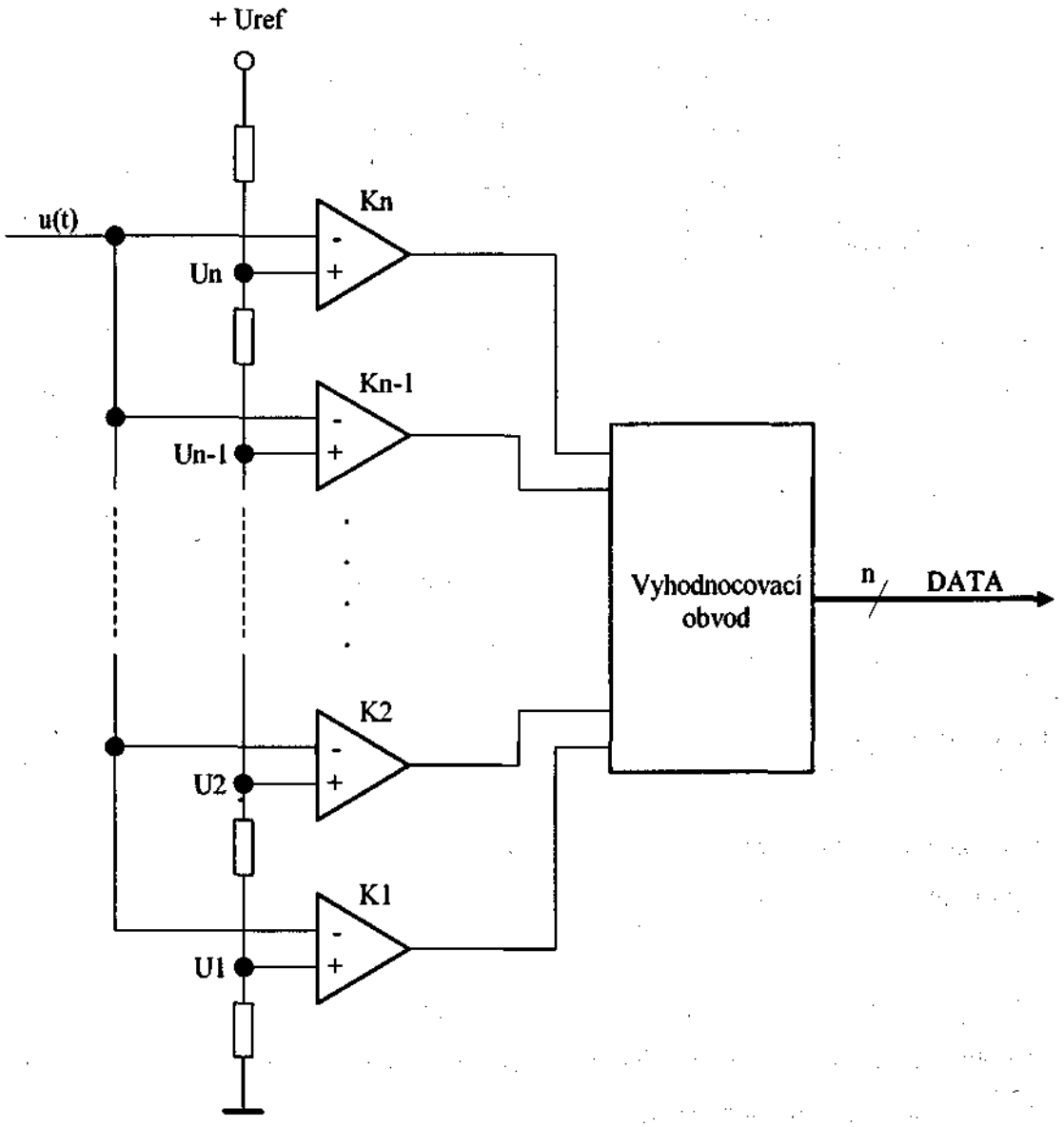
Paralelný A/D prevodník

Paralelný A/D prevodník sa vyznačuje tým, že prevod sa vykoná v jedinom kroku. Tento prevodník je najrýchlejší. Jeho bloková schéma je na obr. 50.

Nevýhodou tohto prevodníka je jeho zložitá realizácia - n-bitový prevodník obsahuje 2^n komparátorov. Vyhodnocovací obvod slúži iba na prevod z jedného kódu do druhého.

Číslicovo/analógové prevodníky

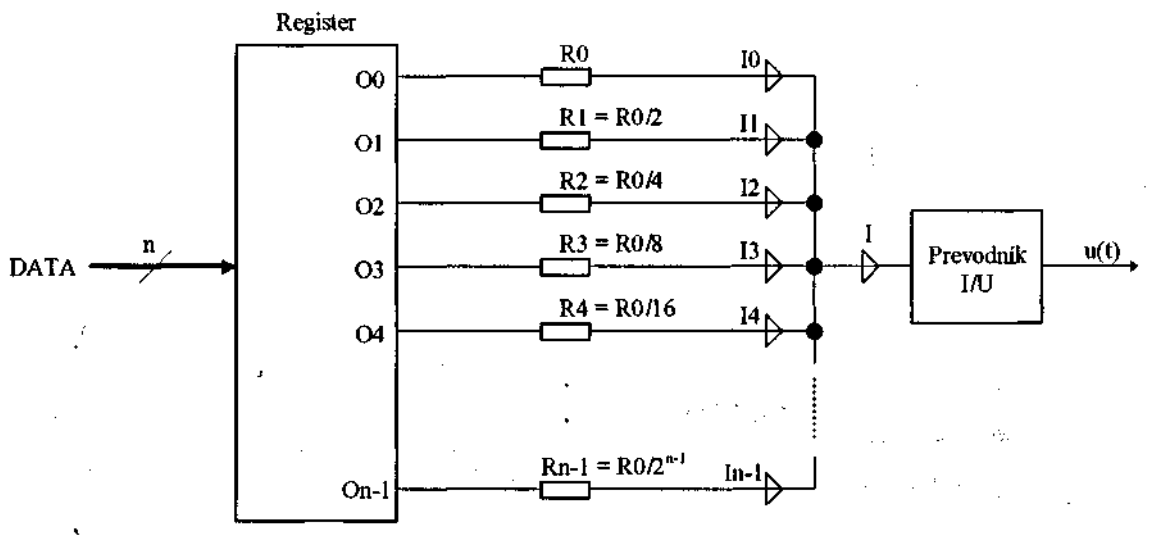
Na ilustráciu uvedieme jednoduchý *D/A prevodník s váhovými odpormi.* Jeho bloková schéma je na obr. 51.



OBR. 50. Paralelný A/D prevodník

Princíp činnosti prevodníka s váhovými odpormi vidno priamo z obrázku. Prevodník *I/U* transformuje celkový výstupný prúd /, ktorý je daný súčtom jednotlivých čiastkových prúdov, na výstupné analógové napätie *U*.

Okrem prevodníka s váhovými odpormi sa často používa aj *D/A prevodník* s *impulznou štrkovou moduláciou (PWM).*



OBR. 51. D/A prevodník s váhovými odpormi

4.4.4.3 Galvanické oddelenie

Doteraz sme implicitne predpokladali, že počítač je ***galvanicky*** *(vodivo) spojený s prostredím.* V niektorých prípadoch však takéto riešenie nevyhovuje:

- *Rušivé vplyvy prostredia na činnosť počítača alebo na komunikáciu*. Elektromagnetické rušenie môže spôsobovať problémy pre správnu činnosť počítača alebo pri komunikácii počítača so vzdialenými zariadeniami.

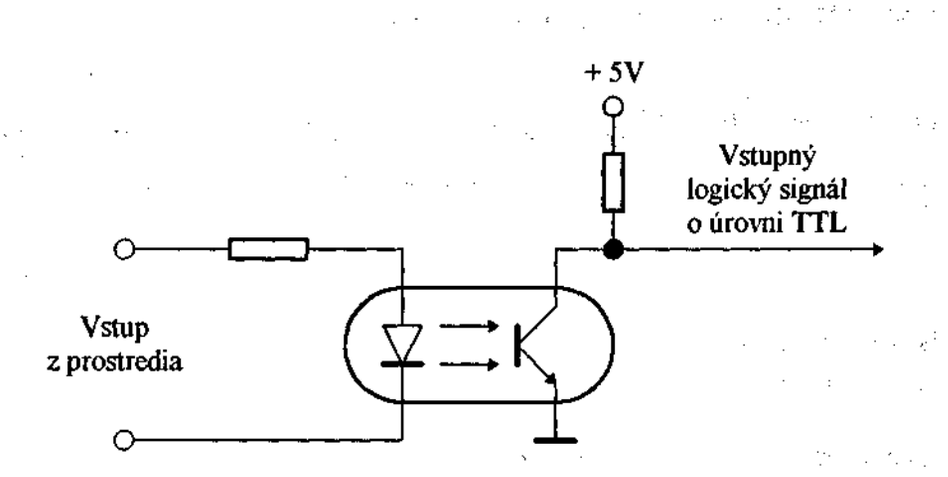
- *Možnosť ohrozenia obsluhy alebo zničenia počítača zo strany prostredia*. V prípade poruchy v prostredí môže byť priamo k počítaču pripojené nebezpečné (napr. sieťové) napätie, ktoré spôsobí zničenie počítača alebo úraz obsluhy.

- *Možnosť ohrozenia prostredia zo strany počítača*. V prípade poruchy v počítači (napr. zničenie napájacieho zdroja) môže byť k prostrediu pripojené sieťové napätie, ktoré môže zničiť riadené zariadenie alebo spôsobiť úraz.

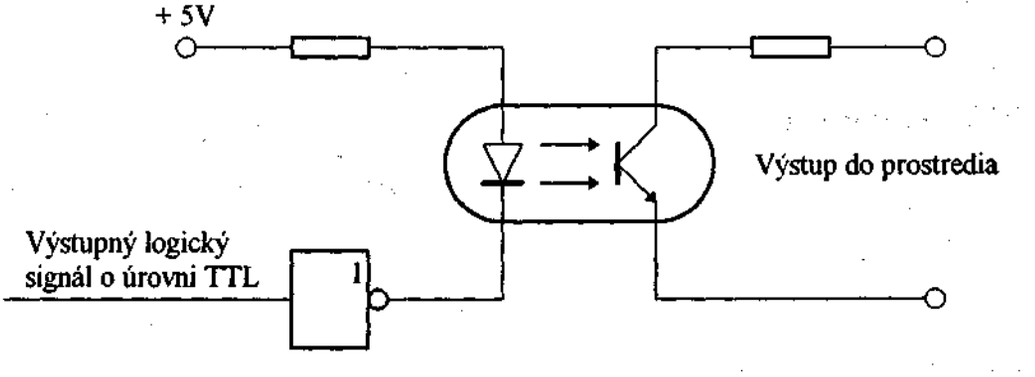
Všetky uvedené problémy je možné riešiť *galvanickým oddelením* počítača od prostredia, čo znamená, že počítač už s prostredím nebude vodivo spojený. Samozrejme, oddeľovacie prvky musia byť dostatočne napäťovo a výkonovo dimenzované.

V súčasnosti sa na galvanické oddelenie používajú *elektronické prvky (optočleny)* a *elektromechanické prvky (relé, stýkače)*.

Na obr. 52 je nakreslená principiálna schéma galvanického oddelenia vstupného a výstupného logického signálu počítača od prostredia optočlenom. V prípade číslicového signálu musí byť realizované *samostatné oddelenie pre každý bit.*



Galvanické oddelenie vstupného signálu počítača



Galvanické oddelenie výstupného signálu počítača

OBR. 52. Principiálna schéma galvanického oddelenia optočlenom

Zabezpečenie samotného počítača proti rušivým vplyvom prostredia a naopak predstavuje samostatnú a rozsiahlu problematiku *elektromagnetickej kompatibility číslicových zariadení,* ktorá už presahuje rámec tejto publikácie.

4.4.5 Metódy vstupno/výstupných prenosov

Pod *vstupno/výstupným (V/V) prenosom* rozumieme prenos údajov medzi periférnym zariadením a procesorom alebo medzi periférnym zariadením a pamäťou.

Riadenie zbernice počítača počas počas V/V prenosu

Podľa toho, *kto riadi zbernicu počítača počas prenosu* údajov z/do periférneho zariadenia, rozdeľujeme V/V prenosy na *prenosy s účasťou procesora* a *prenosy bez účasti procesora*.

4.4.5.1 Prenosy s účasťou procesora

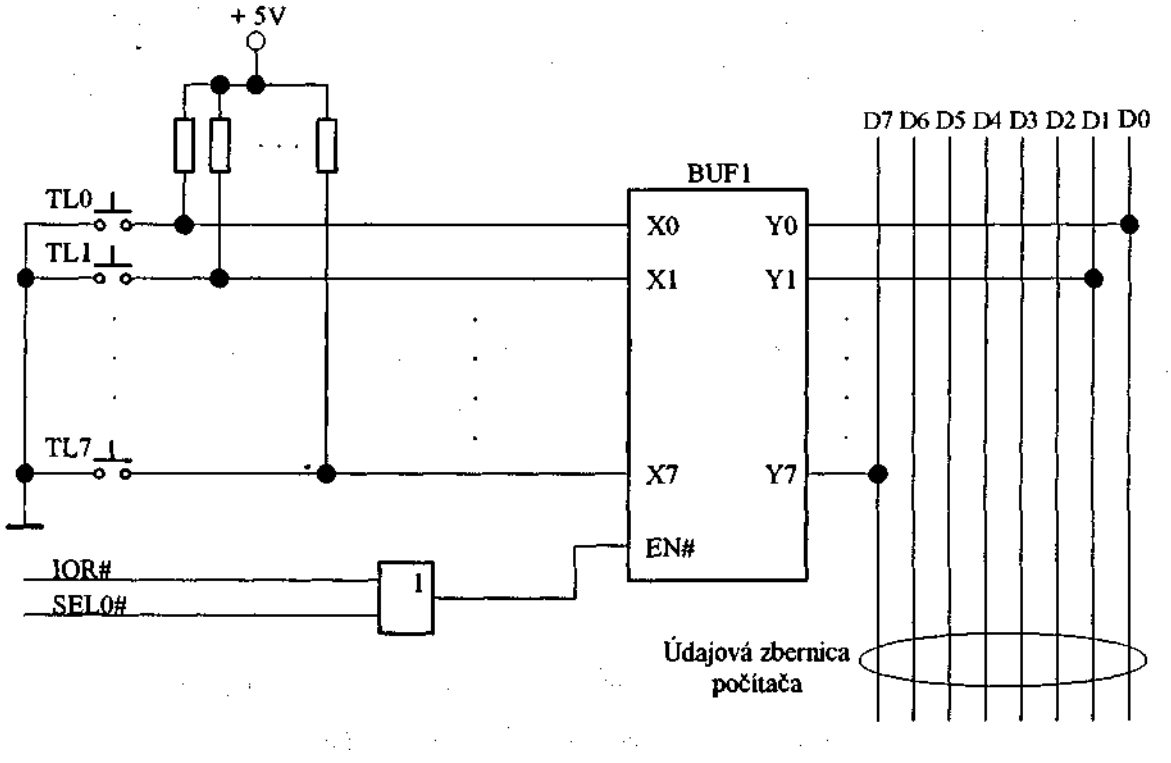
Pri V/V prenosoch s účasťou procesora generuje riadiace signály zbernice procesor. Týmto spôsobom sa typicky vykonáva prenos *jednotlivých údajov* a údaje sa prenášajú *medzi procesorom a V/V zariadením*.

Vykonať prenos údajov z/do periférneho zariadenia nemusí byť možné v každom okamihu (zariadenie je schopné prijať/vyslať údaj iba v istom čase a istou rýchlosťou). Podľa toho, *akým spôsobom sa rozhodne o okamihu odštartovania prenosu údajov*, rozlišujeme tieto V/V prenosy:

Nepodmienený V/V prenos sa vyznačuje tým, že procesor implicitne považuje V/V zariadenie v ktoromkoľvek okamihu *pripravené na prenos,* t. j. že kedykoľvek môže zo vstupného zariadenia údaj načítať a do výstupného zariadenia kedykoľvek údaj zapísať. Prenos je veľmi rýchly, pretože sa vykoná pracovnou rýchlosťou procesora. Na druhej

strane, iba málo periférnych zariadení je schopných prenos takouto rýchlosťou uskutočniť. Typicky sa tento spôsob používa iba na *ovládanie indikačných prvkov, načítanie stavových slov* zariadenia a pod.

Na obr. 53 je nakreslené pripojenie jednoduchého vstupného periférneho zariadenia k zbernici počítača. Toto zariadenie sa skladá z 8 tlačidiel.



OBR. 53. Vstupné periférne zariadenie s nepodmieneným prenosom

Adaptér je vytvorený trojstavovým oddeľovacím obvodom *BUF1.* Na vstup tohto obvodu sú pripojené jednotlivé tlačidlá, výstupy obvodu sú pripojené na zbernicu počítača. Výstupy sú aktivované iba v prípade, ak je na adresovej zbernici nastavená adresa *300H* (signál *SEL0#* z dekódera periférnych zariadení má vtedy aktívnu úroveň) a súčasne je aktívny signál *IOR#* (vykonáva sa cyklus čítania zo vstupného zariadenia). Konkrétna adresa *300H* je zvolená iba pre ilustráciu nášho príkladu.

Vstupy obvodu *BUF1* sú ošetrené odpormi, pripojenými druhým koncom na napájacie napätie, čím je zabezpečená úroveň *log. I* na vstupe obvodu v prípade, ak tlačidlo nie je stlačené.

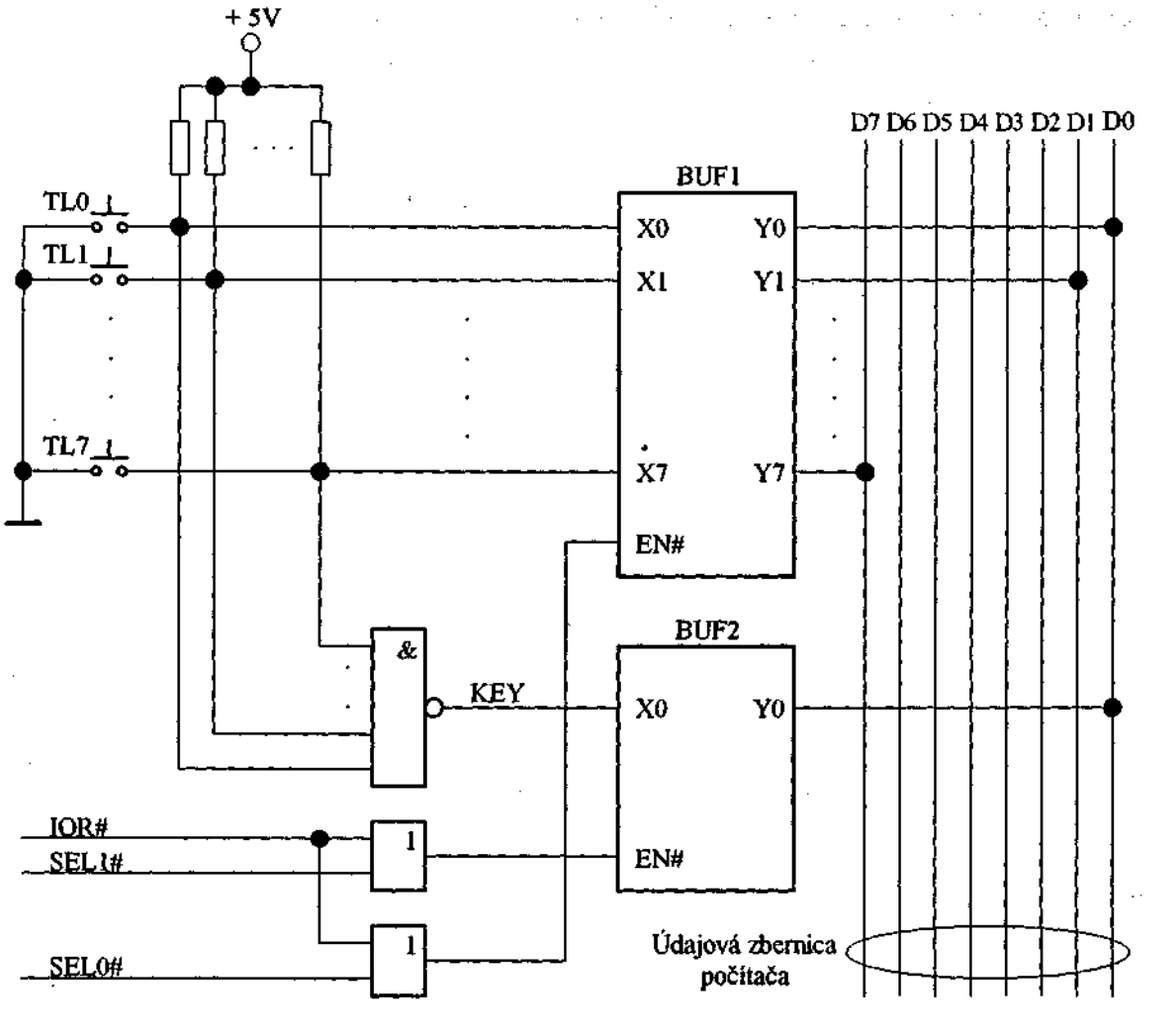
Nasledujúci úsek programu v jazyku symbolických inštrukcií procesorov rodiny *80x86* zisťuje, ktoré tlačidlá sú stlačené:

```
MOV   DX, 300H    ; Adresa adaptéra do registra DX
IN    AL, DX      ; Načítanie vstupného údaja z adaptéra, adresa ktorého je v registri
                  ; DX, do registra AL. Načíta sa boolovský vektor, hodnota jedno-
                  ; tlivých bitov záleží od toho, či je tlačidlo stlačené (log.0) alebo
                  ; nestlačené (log. 1).
```

Netestovali sme pripravenosť tlačidiel, pretože implicitne sme predpokladali, že v tom okamihu, kedy čítame vstupný údaj, sú pripravené na prenos. Je zrejmé, že môže nastať aj situácia, kedy žiadne z tlačidiel nie je stlačené, ale údaj sa napriek tomu načíta (hodnota *0FFH)*.

- Pri *podmienenom V/V prenose* procesor pred vlastným vykonaním prenosu údajov najskôr testuje *pripravenosť* zariadenia prijať, resp. vyslať údaje. Procesor zisťuje pripravenosť takým spôsobom, že z adaptéra načíta *stavové slovo,* v ktorom jeden alebo niekoľko bitov nesie *informáciu o pripravenosti zariadenia*. Len v prípade, ak je zariadenie k prenosu pripravené, procesor vykoná vlastný prenos údajov. Tento prenos rešpektuje pracovnú rýchlosť zariadenia, jeho nevýhodou je však to, že v prípade pomalého periférneho zariadenia (napr. tlačiareň) procesor strávi podstatne viac času čakaním, ako vlastným prenosom údajov.

Na obr. 54 je nakreslené upravené zapojenie z obr. 53. V tomto prípade chceme vykonať prenos iba v tom prípade, keď je niektoré tlačidlo stlačené. Procesor bude čakať dovtedy, kým nezistí pripravenosť zariadenia k prenosu, t.j. stlačenie niektorého tlačidla.



OBR. 54. Upravené zapojenie pre podmienený prenos

Vidíme, že do adaptéra je pridaný ďalší oddeľovací obvod *BUF2,* na vstup ktorého je pripojený signál *KEY,* indikujúci stlačenie niektorého tlačidla. Výstup obvodu je pripojený na *bit 0* údajovej zbernice. Prostredníctvom tohto obvodu sa načíta *stavové slovo.*

Obvod *BUF2* je aktivovaný adresou *320H (signál SEL1# má vtedy aktívnu úroveň)* a aktívnou úrovňou signálu *IOR#* (vykonáva sa cyklus čítania zo vstupného zariadenia). Signál *KEY* je vtedy pripojený na *bit 0* údajovej zbernice počítača. Konkrétna adresa *320H* je zvolená iba pre účely ilustrácie príkladu.

Nasledujúci úsek programu v jazyku symbolických inštrukcií procesorov rodiny *80x86* vykoná načítanie údaja iba v prípade, ak je niektoré tlačidlo stlačené:

```
MOV   DX, 320H      ; adresa oddeľovača stavového signálu KEY do registra DX

NAV1:               ; návestie

IN    AL, DX        ; načítanie stavového slova do registra AL
TEST  AL,1          ; otestovanie hodnoty bitu 0 (signál KEY)
JZ    NAV1          ; ak je hodnota tohto bitu log.0, nie je žiadne tlačidlo stlačené,
                    ; skok sa vykoná a pokračuje sa inštrukciou na návestí NAV1, t.j.
                    ; opätovné načítanie stavového slova
                    ; ak je hodnota tohto bitu log. J, niektoré tlačidlo je stlačené,
                    ; skok sa nevykoná a pokračuje sa nasledujúcou inštrukciou
MOV   DX, 300H      ; už známy nepodmienený prenos z predchádzajúceho príkladu
IN    AL, DX
```

Z programu vidíme, že procesor bude čakať dovtedy, kým nebude niektoré tlačidlo stlačené. Počas čakania nevykonáva žiadnu užitočnú činnosť.

- Nevýhody podmieneného prenosu odstraňuje *V/V prenos s prerušením*. Pri takomto spôsobe prenosu procesor netestuje pred prenosom údajov pripravenosť V/V zariadenia. V/V zariadenie totiž v prípade svojej pripravenosti k prenosu vygeneruje *žiadosť o prerušenie*. Procesor preruší práve prebiehajúci program a v rámci obslužného programu prerušenia bez otestovania pripravenosti zariadenia vykoná vlastný prenos údajov. Po jeho skončení pokračuje v prerušenom programe. Keď je periférne zariadenie schopné

prijať/vyslať ďalší údaj, opäť vygeneruje novú žiadosť o prerušenie atď. Čas, ktorý pri podmienenom prenose procesor strávil neefektívnym čakaním, je pri prenose s prerušením využitý na užitočný výpočet.

Aby sme mohli použiť *V/V prenos s prerušením* pre periférne zariadenie z obr. 54, signál *KEY* pripojíme na *prerušovací vstup procesora*. Oddeľovač *BUF2* teraz nie je potrebný, pretože pripravenosť zariadenia k prenosu nebudeme testovať.

Obslužný podprogram prerušenia môže vyzerať takto:

OBSLUHA:

```
PUSH AX          ; odloženie obsahu registra AX do zásobníka
PUSH DX          ; odloženie obsahu registra DX do zásobníka
MOV  DX, 300H    ; už známy nepodmienený prenos
IN   AL,DX
MOV  UDAJ, AL    ; odloženie načítaného údaja z registra AL do premennej UDAJ
POP  DX          ; obnovenie obsahu registra DX zo zásobníka
POP  AX          ; obnovenie obsahu registra AX zo zásobníka
IRET             ; návrat do prerušeného programu
```

Poznámka.
Obsah registrov AX a DX sme uchovali preto, lebo pri obsluhe prerušenia boli modifikované. V prípade, ak by sme ich neuchovali a prerušený program by ich po svojom pokračovaní použil, prišlo by k jeho nekorektnému správaniu sa.

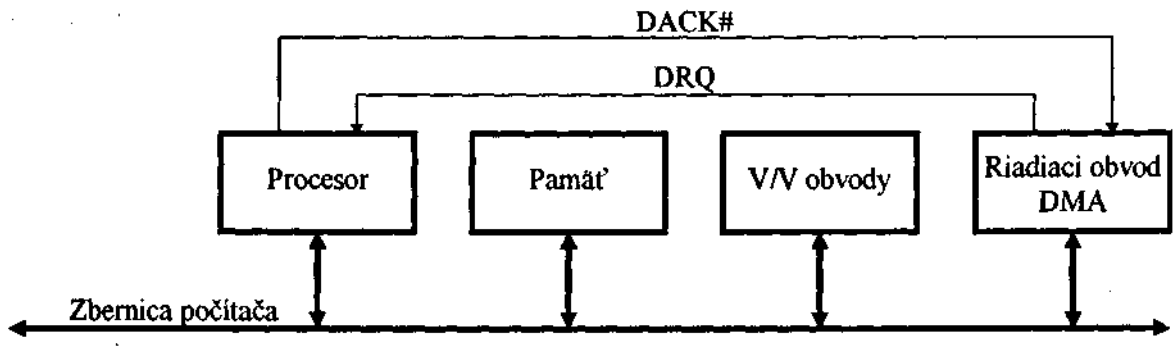
4.4.5.2 Prenosy bez účasti procesora

Prenos bez účasti procesora sa vyznačuje tým, že *počas prenosu údajov riadi zbernicu počítača riadiaci obvod DMA a procesor je od zbernice odpojený* (má svoje výstupy v stave vysokej impedancie). Údaje sa prenášajú *medzi pamäťou a V/V zariadením*. Tento spôsob

prenosu sa tiež nazýva *priamy prístup do pamäte (DMA - Direct Memory Access)* a typicky sa používa na *blokový prenos údajov,* napr. pri práci s pevným diskom.
Ako riadiaci obvod *DMA* je použitý buď *špecializovaný programovateľný obvod,* alebo *špeciálny V/V procesor.*
Tento spôsob prenosu sa vykonáva vtedy, ak riadiaci obvod *DMA* vykonáva prenos údajov rýchlejšie ako procesor. Ďalej, program je podstatne jednoduchší, pretože procesor iba odovzdá riadiacemu obvodu *DMA* požiadavky na prenos a ďalej už len čaká na jeho vykonanie.

Treba si uvedomiť, že aj keď vlastný prenos údajov je bez účasti procesora, V/V prenos ako taký bol *inicializovaný procesorom.* Pred vlastným uskutočnením prenosu údajov musí procesor oznámiť riadiacemu obvodu *DMA* požiadavky na prenos *(odkiaľ a kam sa majú údaje prenášať* a *koľko ich má byť).* Až potom riadiaci obvod *DMA požiada procesor o pridelenie zbernice* a keď mu ju procesor pridelí, vykoná vlastný prenos údajov. Po jeho ukončení *vráti riadenie zbernice späť procesoru,* ktorý môže pokračovať v činnosti.

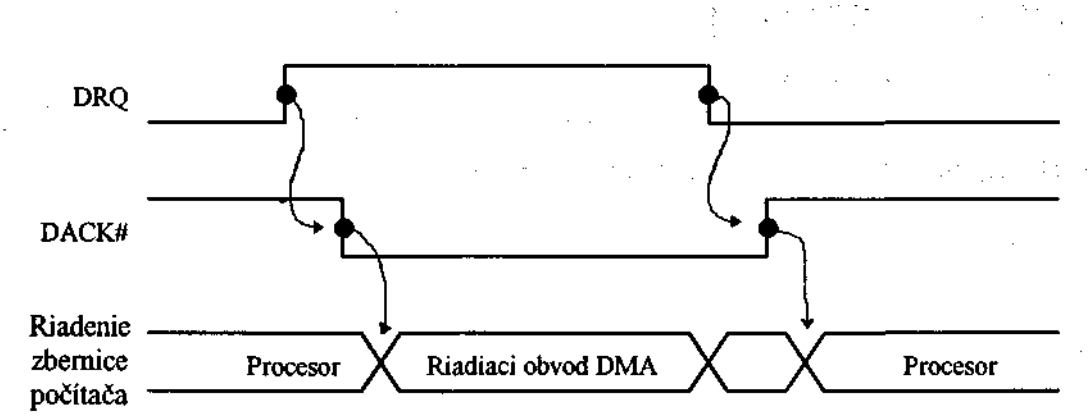
Na obr. 55 je typická konfigurácia počítača so zbernicovou architektúrou. Je vyznačené pripojenie riadiaceho obvodu *DMA,* pripojenie signálu *žiadosti o zbernicu DRQ (DMA Request)* z riadiaceho obvodu *DMA* do procesora, ako aj pripojenie signálu *o pridelení zbernice DACK# (DMA Acknowledge)* z procesora do riadiaceho obvodu *DMA.*



OBR. 55. Počítač s riadiacim obvodom DMA

Na obr. 56 je nakreslený typický signálny sled pri prenose *DMA*. Riadiaci obvod *DMA* žiada procesor o pridelenie zbernice nastavením signálu *DRQ* do aktívnej úrovne. Procesor zbernicu uvoľní a oznámi jej pridelenie riadiacemu obvodu *DMA* tak, že nastaví signál *DACK#* do aktívnej úrovne. Potom riadiaci obvod *DMA* prevezme riadenie zbernice a vykoná vlastný prenos údajov. Po jeho ukončení uvoľní zbernicu a procesoru to oznámi nastavením signálu *DRQ* do neaktívnej úrovne. Procesor reaguje nastavením potvrdzovacieho signálu *DACK#* do neaktívnej úrovne a opäť prevezme riadenie zbernice.

V obrázku sú vyznačené časové úseky, v ktorých riadi zbernicu procesor resp. riadiaci obvod *DMA*. Všimneme si, že istú dobu nemusí byť zbernica riadená. Aby neprišlo k neželanému zápisu do pamäte alebo V/V zariadenia, riadiace signály, aktívne v úrovni *log.0*, sa ošetrujú pripojením cez odpor *(pull-up resistor)* na napätie *+5V*. Tým je zabezpečená ich neaktívna úroveň v čase, keď zbernica nie je riadená ani procesorom, ani riadiacim obvodom DMA.



OBR. 56. Signálny sled pri prenose DMA

J

5 Počítače s jedným prúdom inštrukcií a viacerými prúdmi údajov

Táto kapitola sa zaoberá počítačmi vo Flynnovej klasifikácii označovanými ako *SIMD*. Sú to *paralelné počítače,* použité najmä pre vykonávanie *vektorových* a *maticových operácií,* t. j. súčasne sa vykonáva jedna operácia s viacerými údajovými štruktúrami rovnakého typu. Jeden program sa vykonáva súčasne s viacerými prúdmi údajov vo viacerých *procesných elementoch (Processing Elements).*

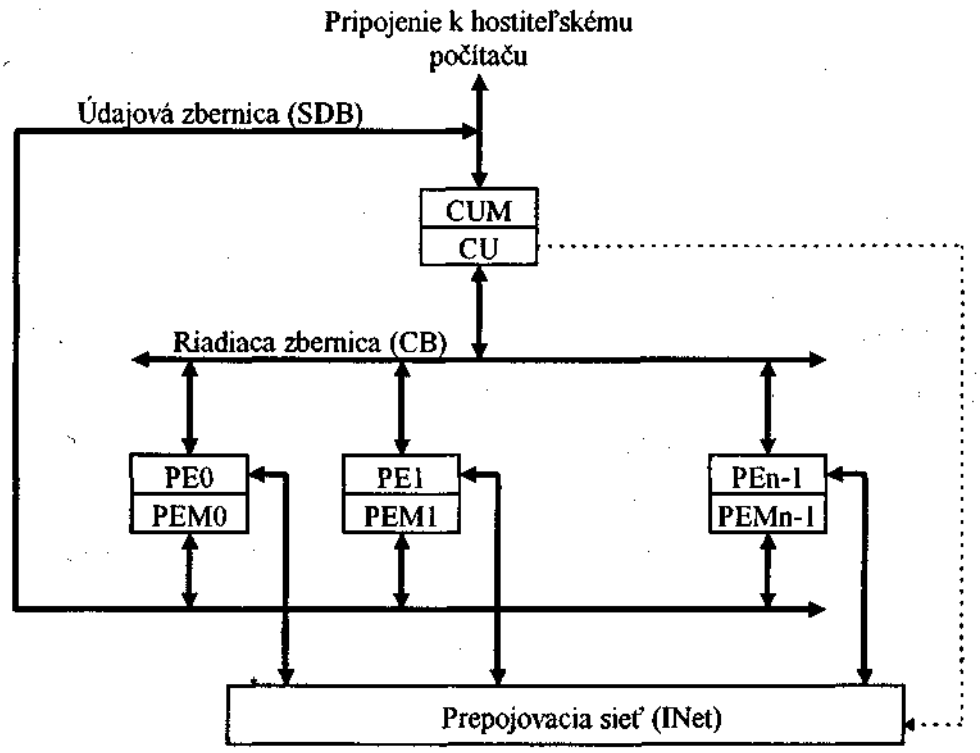
SIMD počítače používajú buď *maticový procesor (Array Processor)* alebo *asociatívny procesor (Associative Processor).* V prvom prípade je použitá pamäť s náhodným prístupom, v druhom prípade pamäť adresovaná obsahom (asociatívna pamäť).

Asociatívny procesor predstavuje špeciálny typ maticového procesora, v ktorom slová asociatívnej pamäte zodpovedajú jednotlivým procesným elementom.

5.1 Maticové procesory

Na obr. 57 je principiálna schéma maticového procesora počítača Illiac IV., ktorý je typickým predstaviteľom počítačov s architektúrou SIMD.

Maticový procesor sa skladá z *N* synchronizovaných *procesných elementov (PE),* ktoré sú riadené jedinou spoločnou *riadiacou jednotkou (CU).*
Každý *procesný element* obsahuje *aritmeticko-logickú jednotku* s pripojenými *pracovnými registrami* a má vlastnú *lokálnu pamäť (PEM)* pre uloženie spracúvaných údajov.
Aj k *riadiacej jednotke* je pripojená *pamäť (CUM),* v tejto je však uložený program. Funkciou riadiacej jednotky je *dekódovať inštrukcie* a *rozhodnúť, kde budú dekódované inštrukcie vykonané.*
Riadiace inštrukcie a inštrukcie na vykonanie *skalárnych operácií* sú vykonáva priamo riadiaca jednotka, *inštrukcie na vykonanie vektorových operácií* sú vyslané do jednotlivých



OBR. 57. Bloková schéma maticového procesora

procesných elementov. Všetky procesné elementy vykonajú rovnakú operáciu *synchrónne* pod riadením riadiacej jednotky. Vektorové operandy sú naplnené do lokálnych pamätí procesných elementov ešte pred vykonaním vektorovej inštrukcie z externého zdroja buď cez *systémovú údajovú zbernicu SDB* alebo cez riadiacu jednotku použitím *riadiacej zbernice CB.* Pre riadenie stavu každého procesného elementu je použitý *maskovací vektor.* Každý *PE* môže byť v danom inštrukčnom cykle buď *aktívny (Active)* alebo *zablokovaný (Disabled).* Iba tie elementy, ktoré sú aktívne, sa zúčastnia na vykonaní danej vektorovej inštrukcie. Výmena údajov medzi procesnými elementmi sa vykonáva cez *prepojovaciusieť INet,* ktorú riadi riadiaca jednotka.

Maticový procesor je pripojený k *hostiteľskému počítaču* prostredníctvom riadiacej jednotky *CU.* Hostiteľský počítač je obyčajne univerzálny počítač, ktorý slúži ako manažér celého systému. Napĺňa programy a údaje do maticového procesora a preberá z neho výsledky. Zabezpečuje tiež komunikáciu celého systému s okolím.

6 Počítače s viacerými prúdmi inštrukcií a viacerými prúdmi údajov

Táto kapitola sa zaoberá počítačmi a počítačovými systémami, vo Flynnovej klasifikácii označovanými ako *MIMD*. Niekoľko programov, vykonávaných rôznymi procesormi, paralelne spracúva vlastný prúd údajov - ide o paralelné spracovanie s oddelene prebiehajúcimi *paralelnými procesmi,* riadenými *samostatnými* procesormi.
Patria sem ***multiprocesorové*** a ***multipočítačové*** *systémy*. Počítače typu *MIMD* sa používajú na riešenie úloh, ktoré je možné *rozložiť na niekoľko paralelných procesov,* alebo na *paralelné riešenie nezávislých* úloh. V obidvoch prípadoch je však typická vzájomná *komunikácia* medzi procesormi, a to buď prostredníctvom špecializovaných *komunikačných kanálov* alebo **cez** *zdieľanú pamäť*.

6.1 Multiprocesorové systémy

Multiprocesorový systém je paralelný počítač, obsahujúci niekoľko procesorov, ktoré majú *vlastnú* alebo *zdieľanú* pamäť a pre komunikáciu s okolím používajú *spoločné vstupné* a *výstupné zariadenia*.

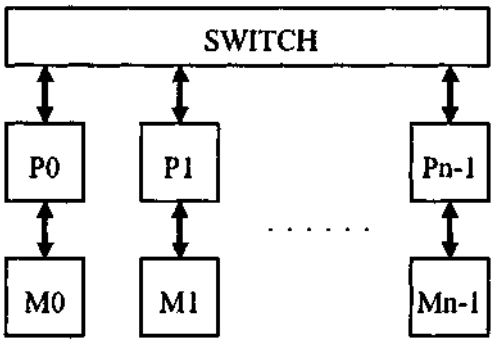
Podľa toho, či procesory *majú vlastnú alebo zdieľanú pamäť*, rozdeľujeme multiprocesorové systémy na:

1. *voľne viazané (Loosely Coupled),*
2. *tesne viazané (Tightly Coupled),*
3. *hybridné (Hybrid).*

Voľne viazané multiprocesorové systémy

Voľne viazané multiprocesorové systémy sa vyznačujú tým, že každý z procesorov má vlastnú pamäť, do ktorej má prístup iba on a na vzájomnú komunikáciu je použitý mechanizmus

posielania správ (Message Passing). Tento mechanizmus používa špecializované *komunikačné kanály* jednotlivých procesorov. Na obr. 58 je naznačená štruktúra voľne viazaného multiprocesorového systému. Prostredníctvom *prepojovacieho podsystému,* ktorý spája komunikačné kanály jednotlivých procesorov, môžu procesory navzájom komunikovať. *Prepojovaci podsystém* býva realizovaný buď ako *krížový prepínač* alebo ako *viacstupňová prepojovacia sieť.*



P označuje procesor, *M* pamäťový modul, *SWITCH* prepojovaci podsystém

OBR. 58. Voľne viazaný multiprocesorový systém

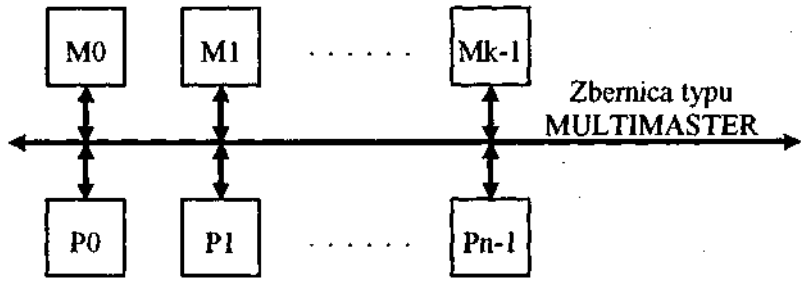
Tesne viazané multiprocesorové systémy

Tesne viazané multiprocesorové systémy sa vyznačujú *zdieľanou pamäťou,* do ktorej majú prístup všetky procesory.

Zdieľaná pamäť je realizovaná jedným alebo viacerými modulmi, pričom počet procesorov a počet pamäťových modulov nemusí byť rovnaký. Pri týchto systémoch je nutné zabezpečiť *vzájomné vylučovanie* pri *súčasnej požiadavke o prístup do rovnakej oblasti pamäte.*

V tesne viazaných multiprocesorových systémoch procesory komunikujú cez *vyhradenú oblasť* v zdieľanej pamäti, kam zapisujú svoje správy.

Na obr. 59 je naznačená štruktúra tesne viazaného multiprocesorového systému, v ktorom je prepojovaci podsystém realizovaný ako *zbernica typu MULTI-MASTER.*

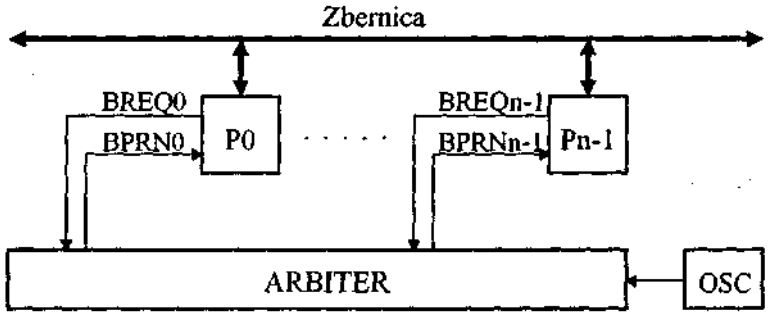


OBR. 59. Tesne viazaný multiprocesorový systém

Pretože v danom okamihu môže byť zbernica riadená iba jedným procesorom, je potrebné riešiť problém prideľovania zbernice pri viacerých súčasných požiadavkách.
Používa sa niekoľko metód riešenia tohto problému, napr. *paralelná* alebo *sériová arbitrácia žiadostí o zbernicu, cyklické prideľovanie zbernice, náhodný prístup na zbernicu.*

Paralelná arbitrácia žiadostí

Pri *paralelnej arbitrácii* je použitý paralelný arbiter, znázornený na obr. 60. Každý procesor môže požadovať pridelenie zbernice prostredníctvom signálu *BREQ*. Pridelenie zbernice príslušnému procesoru je oznamované signálom *BPRN*.



OBR. 60. Paralelný arbiter zbernice

Ak dva alebo viac procesorov požaduje súčasne pridelenie zbernice, paralelný arbiter pridelí zbernicu len jednému z nich, a to podľa *priority*. Zbernica bude pridelená pre nasledujúci cyklus zbernice procesoru *s najvyššou prioritou.*

Priorita môže byť pri paralelnej arbitrácii daná *napevno (staticky)* alebo sa môže aj dynamicky *meniť:*

- Pri *statickom pridelení priorít* je priorita jednotlivých žiadostí definovaná vopred a je nemenná. Takéto riešenie je vhodné používať vtedy, ak je jasne odstupňovaná dôležitosť obsluhy jednotlivých procesorov.

- Pri *dynamickom pridelení priorít* sa úroveň priority jednotlivých procesorov mení podľa potreby z pohľadu celého systému, napr. najnižšiu prioritu bude mať práve obslúžený procesor a najvyššiu ten, ktorý najdlhšie nemal pridelenú zbernicu.

Osobitným konštrukčným problémom návrhu arbitra je *časovanie akceptovania* prichádzajúcich žiadostí. Predstavme si, že prišli dve žiadosti o pridelenie zbernice a arbiter vyberie podľa priority jednu z nich. Vo chvíli vzniku signálu *BPRN* pre jeden z procesorov príde tretia žiadosť, s vyššou prioritou ako obe predchádzajúce. Čo urobí arbiter ? Odloží ďalšiu arbitráciu na nasledujúci cyklus alebo prehodnotí rozhodnutie ? Ak prehodnotí rozhodnutie a príde žiadosť s ešte vyššou prioritou, čo urobí ?

Tento problém sa rieši *synchronizáciou žiadostí.* Napríklad tak, že v prvej etape arbiter *zbiera žiadosti* a v druhej etape *prideľuje zbernicu.* Striedanie týchto etáp sa deje na základe *synchronizačného* (hodinového) *signálu,* takže čas na prihlásenie žiadosti je obmedzený. Ide o synchronizáciu *asynchrónnych* dejov. Preto je na obr. 60 vyznačený blok *OSC,* ktorý poskytuje synchronizačný signál pre arbiter aj pre zbernicu.

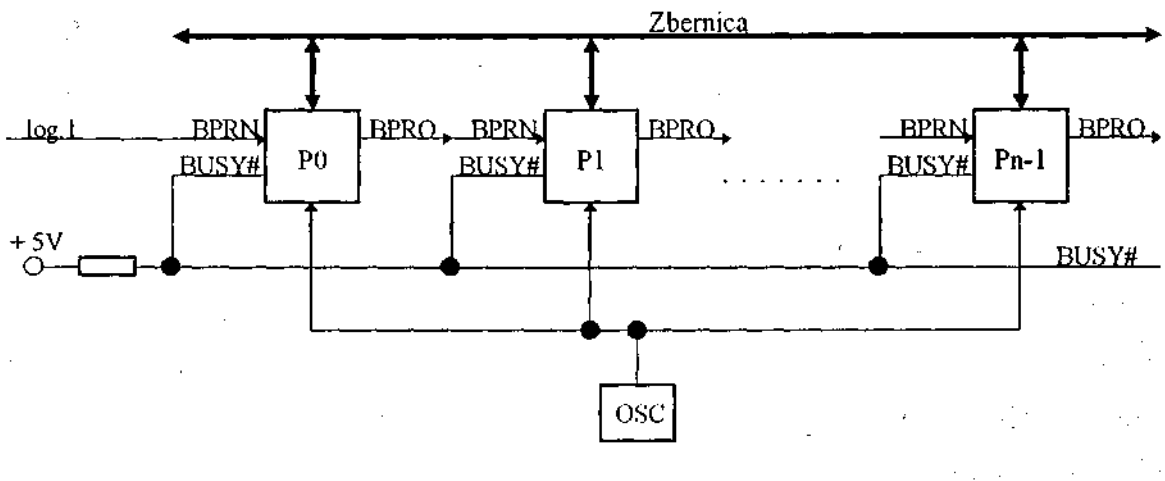
Sériová arhitrácia žiadostí

Pri *sériovej arbitrácii* priorít sú všetky procesory sériovo prepojené prostredníctvom signálov *BPRN* a *BPRO.* Okrem toho je obsadenosť zbernice indikovaná aktívnou úrovňou signálu *BUSY#.* Procesor sa môže pripojiť k zbernici iba v prípade, ak je zbernica voľná (signál *BUSY#=1)* a súčasne má svoj vstupný signál *BPRN* v aktívnej úrovni. Signál obsadenosti zbernice *BUSY#* uvedie do aktívnej úrovne ten procesor, ktorému bola zbernica pridelená, ostatné procesory udržiavajú svoj výstup *BUSY#* v stave vysokej impedancie. Pri pridelení

zbernice nastaví súčasne procesor aj svoj výstup *BPRO* do neaktívnej úrovne, čím zablokuje žiadosti ostatných procesorov s nižšou prioritou. Ak procesor o zbernicu nežiada, prenáša signál zo svojho vstupu *BPRN* na výstup *BPRO*. Arbiter je takto distribuovaný do jednotlivých procesorov a *priority sú jednoznačne dané zapojením procesora v prioritnom reťazci*. Procesor s najvyššou prioritou (prvý procesor v prioritnom reťazci) má svoj vstupný signál *BPRN* trvale aktívny.

Aj v prípade sériovej arbitrácie je nutné činnosť synchronizovať pomocou hodinového signálu.

Na obr. 61 je naznačená sériová arbitrácia priorít. Všimneme si, že signál *BUSY#* je *ošetrený* (cez odpor je pripojený na napájacie napätie), takže v prípade, ak je zbernica voľná (t.j. všetky procesory majú svoj výstup *BUSY#* v stave vysokej impedancie), má neaktívnu úroveň *(BUSY#=log.1)*.



OBR. 61. Sériová arbitrácia priorít

Cyklické prideľovanie zbernice

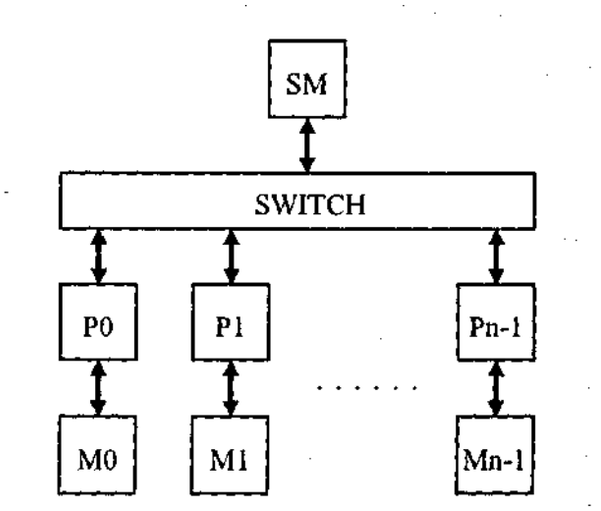
Cyklické prideľovanie zbernice jednotlivým staniciam je ďalšou stratégiou prideľovania zbernice. Pri cyklickom prideľovaní arbiter postupne kruhovo prideľuje zbernicu jednotlivým procesorom, takže každý procesor má rovnakú možnosť získať zbernicu. Ak procesor nepožaduje v danom okamihu pridelenie zbernice, arbiter pridelí zbernicu ďalšiemu v poradí.

Náhodné ***prideľovanie zbernice***

Náhodné prideľovanie zbernice prebieha tak, že procesor, ktorý potrebuje použiť zbernicu, počká na ukončenie práve prebiehajúcej komunikácie a potom sa zbernicu pokúsi použiť. Ak žiaden iný procesor neurobil to isté, daný procesor vykonal úspešný cyklus zbernice. Ak vznikla kolízia, procesor sa od zbernice odpojí a pokúsi sa opätovne použiť zbernicu o chvíľu. Čas, ktorý bude každý z kolidujúcich procesorov čakať, je daný náhodne, čím sa znižuje pravdepodobnosť opätovnej kolízie.

Hybridné multiprocesorové systémy

V hybridných multiprocesorových systémoch má síce každý procesor svoju vlastnú pamäť, ale v systéme existuje buď spoločná (zdieľaná) pamäť alebo do pamäte každého procesora môžu mať prístup aj ostatné procesory. Na obr. 62 je naznačená štruktúra hybridného multiprocesorového systému, v ktorom existuje zdieľaná pamäť.



SM označuje zdieľanú pamäť

OBR. 62. Hybridný multiprocesorový systém

6.2 Multipočítačové (distribuované) systémy

Multipočítačový (distribuovaný) systém sa skladá z niekoľkých počítačov s možnosťou vzájomnej komunikácie, ktoré sú schopné aj samostatnej činnosti. Typickým predstaviteľom distribuovaných systémov sú napr. *informačné systémy pracujúce* v *reálnom čase,* do ktorých je možné súčasne vstupovať z viacerých miest, od seba značne vzdialených. Medzi distribuované systémy patria aj *počítačové siete,* ktorým sa budeme v ďalšom venovať podrobnejšie.

6.2.1 Počítačové siete

Počítačové siete slúžia na vzájomné prepojenie rôznych výpočtových systémov za účelom *zdieľania informácií* alebo *technických prostriedkov.*
Pretože pri rozsiahlych sieťach, ktoré spájajú počítače v rámci veľkých oblastí, fyzicky nie je možné prepojiť každý počítač s každým, vzniká *polygonálna sieť,* v ktorej je spojenie sprostredkované - niektoré počítače majú počas spojenia funkciu *spojovacích uzlov.* Ako fyzické spoje sa používajú buď *hlasové kanály telefónnej siete* (sú pomalé a málo spoľahlivé) alebo *zvláštne dátové linky.*

Na *sprostredkovanie spojenia* sa používajú dva princípy:

- *Prepájanie okruhov* je analógiou telefónnej siete. Počas celého spojenia je vytvorená cesta medzi koncovými uzlami, ktorá slúži iba pre toto spojenie. Výhodou je minimálne oneskorenie prenášanej informácie, nevýhodou vysoká cena spojenia v prípade, ak nie je využitá plná kapacita spoja. Efektívnejšie využitie je možné dosiahnuť vtedy, ak sieť umožňuje dostatočne rýchle vytvorenie spoja iba pre prenos určitého zhluku správ.

- *Prepájanie paketov* pracuje s blokmi údajov s maximálnou prípustnou dĺžkou *(pakety).* Vysielajúca stanica vysiela pakety spolu s cieľovou adresou do komunikačnej siete. V najbližšom prepájacom uzle je paket dočasne uložený, preskúma sa jeho adresa a potom sa vyšle najvýhodnejším smerom ďalej. Postupným opakovaním týchto krokov sa dostane

paket k adresátovi. Tento typ siete pracuje pri náhodnom rozložení zaťaženia siete efektívnejšie ako predchádzajúci.

6.2.1.1 Rozdelenie počítačových sietí

Počítačové siete môžeme deliť podľa viacerých kritérií, my si uvedieme rozdelenie podľa územnej rozľahlosti, podľa typu pospájaných počítačov a podľa topológie.

Podľa územnej rozľahlosti:

- *Lokálne (LAN - Local Area Network).* Prepojenie v rámci budovy alebo areálu jedného podniku na vzdialenosť *stoviek metrov až niekoľko km.*

- *Mestské (MAN - Metropolitan Area Network).* Vytvorenie siete v rámci mesta, prepojenie do vzdialenosti *niekoľko desiatok km.*

- *Globálne (WAN - Wide Area Network).* Tieto siete spájajú používateľov vo viacerých mestách, vo viacerých štátoch, prípadne medzi kontinentmi na vzdialenosti *stoviek až tisícov km.*

Podľa typu počítačov zapojených do siete:

- *Homogénne* - všetky počítače sú rovnakého typu.
- *Heterogénne* - prepojenie počítačov rôzneho typu s rôznymi operačnými systémami.

Podľa topológie:

- *Polygonálne.*
- *Strom.*

- *Kruh.*
- *Zbernica.*
- *Hviezda.*
- *Kombinované.*

6.2.1.2 Základné pojmy

Správa je jednotka, ktorá obsahuje ucelenú informáciu. Jej obsahom môžu byť tak údaje nejakého diskového súboru, ako aj žiadosť o ich zaslanie, potvrdenie ich správneho príjmu alebo iné dátové a riadiace informácie. Z toho vyplýva, že i dĺžka správy môže byť veľmi rozdielna. Toto je z hľadiska prevádzky siete nevhodné, a preto sa správy pri väčšine sietí prenášajú po úsekoch pevnej alebo obmedzenej dĺžky, ktoré sa nazývajú *pakety*. Dlhšie správy sa delia na viac kratších paketov, veľmi krátke správy môžu byť doplňované na minimálnu dĺžku. Aby mohol byť paket prenesený po sieti, musí byť doplnený o ďalšie údaje.

Rámec je paket doplnený o *synchronizačnú postupnosť, cieľovú a zdrojovú adresu a kontrolný znak.*

Synchronizačná postupnosť slúži na označenie začiatku rámca a sfázovanie generátora hodín na prijímacej strane.
Cieľová adresa informuje o tom, ktorá stanica má paket prijať a *zdrojová adresa* identifikuje vysielaciu stanicu.
Kontrolný znak umožňuje prijímacej strane rozhodnúť, či bol paket prenesený bez chyby. Obyčajne sa používa *CRC kód,* čo je v podstate zvyšok po delení rámca stanoveným polynómom.

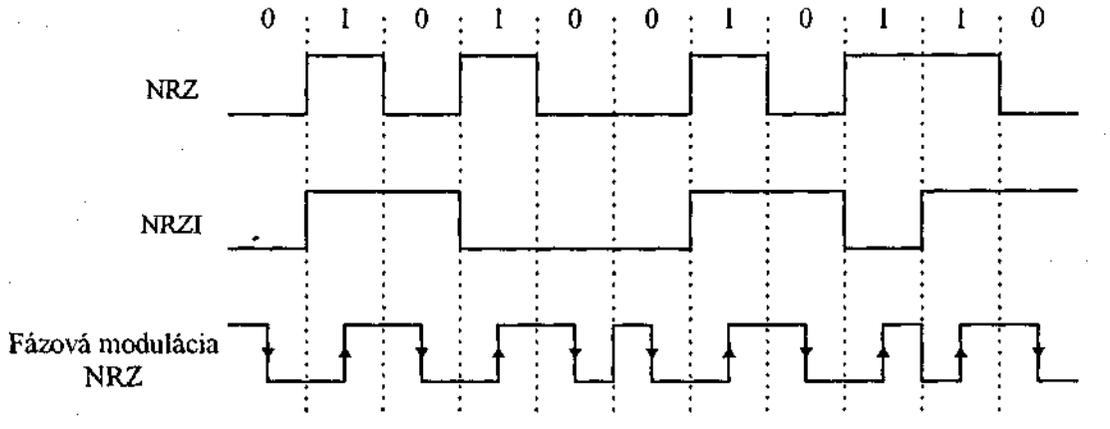
6.2.1.3 Kódovanie signálu

Informácia sa prenáša po prenosovom médiu siete prostredníctvom vhodných fyzikálnych veličín. Tak napr. pri klasických kábloch je to *elektrický signál,* pri optických kábloch

svetelné žiarenie. Pred vlastným prenosom je potrebné informáciu nejakým spôsobom zakódovať - prideliť logickým hodnotám úrovne fyzikálnej veličiny.

Rozlišujeme dva základné typy prenosu - *prenos v základnom pásme* a *prenos v preloženom pásme (modulácia):*

- Pri *prenose v základnom pásme* sa používa niekoľko typov kódovania, napr. *NRZ, NRZI, fázová modulácia NRZ* (častejšie sa nazýva *kód Manchester II), diferenciálna fázová modulácia (diferenciálny Manchester II), fázová modulácia RZ* atď. Na obr. 63 sú niektoré typy kódovania binárneho signálu, použité pri prenose v základnom pásme.



OBR. 63. Niektoré typy kódovania binárneho signálu

- *Prenos v preloženom pásme (modulovaný prenos)* sa v lokálnych sieťach s klasickými medenými káblami používa zriedkavejšie. Modulácia je však nutná na prenos po *optických kábloch.*

  Používajú sa tieto základné typy modulácie:

  - *Amplitúdová* - ľahko realizovateľná, ale málo odolná proti nelineárnemu skresleniu i proti rušeniu. Vhodná je najmä na moduláciu zdrojov svetla na optických spojoch.

- *Kmitočtová* - oproti amplitúdovej je odolnejšia proti skresleniu i rušivým signálom, vyžaduje však väčšiu šírku pásma. Používa sa pri prenose údajov telefónnymi spojmi a rádiovými kanálmi.

- *Fázová* - dobre využíva kmitočtové pásmo. Používa sa pri prenose údajov telefónnymi kanálmi. Niekedy môže byť kombinovaná s amplitúdovou moduláciou.

6.2.1.4 Potvrdzovanie správ

Pri prenose paketov po sieti môže dochádzať v zásade k dvom chybovým situáciám:

V prvom prípade cieľová stanica síce paket prijme, ale kontrolou kontrolného znaku zistí, že prenos neprebehol správne.
V druhom prípade cieľová stanica paket vôbec neprijme, a to buď preto, že bolo poškodené jeho adresné pole, alebo počas vysielania cieľová stanica nebola pripravená na príjem.

Ak je požiadavka na zabezpečenie bezchybného prenosu informácií, musí byť vysielacia strana obe tieto chyby schopná detekovať a vyslanie príslušného paketu zopakovať. Pre tento účel sa najčastejšie používa *metóda potvrdzovania správ.*

Existuje niekoľko metód, ktoré sa v praxi môžu kombinovať a modifikovať:

- *Pozitívne potvrdzovanie.* Prijímacia strana v prípade, že prevzala dátový paket bez chyby, vyšle špeciálny potvrdzovací paket, väčšinou označovaný ako *ACK.* Na chybne prijatý paket (a samozrejme na neprijatý) stanica nereaguje. Vysielacia strana po vyslaní paketu musí čakať istú dobu na potvrdzovací paket *(timeout).* Až po jeho príjme môže pokračovať vyslaním ďalšieho paketu. Ak nepríde potvrdenie do istého času, opakuje vysielanie predchádzajúceho paketu znovu. Uvedený postup dokáže vyriešiť aj stratu paketu - táto vedie k jeho opätovnému vyslaniu. Pozitívne potvrdzovanie je základom v praxi používaných metód. V svojej čistej forme má však nevýhody vo veľkom náraste časových oneskorení pri raste chybovosti prenosu a v relatívne veľkom zaťažení siete pri prenose krátkych paketov.

- *Negatívne potvrdzovanie*. Vysielacia strana je upozornená iba na chybne prijaté pakety prijatím špeciálneho paketu, označovaného *NACK:* Tento mechanizmus rýchle reaguje na chybne prijaté pakety, nedokáže však vyriešiť stratu paketu, preto sa kombinuje s pozitívnym potvrdzovaním. V tejto kombinácii potom rýchle reaguje na chybne prijatý paket, stratený paket je detekovaný uplynutím časového limitu. Strata potvrdenia vedie k zdvojeniu paketu na prijímacej strane.

- *Číslovanie paketov*. Číslovanie paketov je mechanizmus, ktorý umožňuje prijímacej strane detekovať zdvojený paket. K tomuto dochádza pri strate potvrdzovacieho paketu. Vysielacia strana nie je schopná rozpoznať, či prišlo k strate paketu alebo iba k strate jeho potvrdenia a opakuje vysielanie. Aby prijímacia strana rozpoznala, že ide o už prijatý paket, všetky pakety sú vzostupne očíslované. Z praktických dôvodov sa používa modulárne číslovanie, napr. cyklicky čísla v rozsahu *0* až *127*. Potom je možné detekovať nielen opakovanú správu (rovnaké číslo), ale aj stratu správy (vynechané číslo). Stratu je možné detekovať za predpokladu, že prepájacia sieť zaručuje dodržanie postupnosti vysielaných paketov. Okrem vysielaných správ môžu byť číslované aj potvrdzovacie pakety.

- *Skupinové potvrdzovanie*. Potvrdzovaním každého paketu samostatne môžu značne narastať časové straty, najmä pri väčších oneskoreniach v prenosovej ceste. Preto je výhodnejšie potvrdzovať prijatie celej skupiny paketov. Vysielacia strana očakáva potvrdenie až po odvysielaní *n* paketov. Prijímacia strana potvrdzuje číslom posledného' správne prijatého paketu.

- *Nesamostatné potvrdzovanie*. Ďalšie zvýšenie využiteľnosti prenosovej cesty sa dosahuje tým, že v každom vysielanom rámci sa vyhradí zvláštne pole na informáciu o čísle potvrdzovaného paketu. Ak potom prichádza k obojstrannej výmene správ medzi stanicami, potvrdenie prv prijatého paketu môže byť súčasťou teraz vysielanej správy. Ak prijímacia strana nemá pripravené vlastné dáta na vysielanie, použije samostatný potvrdzovací paket. Väčšina komunikačných protokolov lokálnych sietí používa niektorú formu skupinového nesamostatného potvrdzovania.

6.2.1.5 Datagramy a virtuálne spojenia

Komunikácia účastníkov siete s prepájaním paketov prebieha obojsmerným vymieňaním správ - postupností paketov. Ich prenos je možné realizovať buď *datagramovou službou (connectionless service)* alebo ako *virtuálny spoj (connection-oriented)*.

- *Datagram* je paket, ktorý obsahuje kompletnú adresu vysielacej a cieľovej stanice a ktorý *cestuje po sieti nezávisle od ostatných paketov,* prenášaných medzi rovnakými stanicami. Pretože pri veľkých sieťach, v ktorých paket prechádza viacerými spojovacími uzlami, môže byť pre každý práve prenášaný datagram vybraná iná cesta, nemusí datagramová služba zaručovať poradie, v ktorom sú pakety doručované. Prenášanie paketov je však jednoducho realizovateľné a umožňuje posielanie správ na všeobecnú alebo skupinovú adresu.

- *Virtuálny spoj* je vytvorený medzi koncovými stanicami pred vlastným začiatkom prenosu správ prostredníctvom výmeny *špeciálnych* paketov. Ak ide o veľkú sieť s viacerými prepájacími uzlami, vytvorí sa v nich pred nadviazaním virtuálneho spoja informácia o ceste, ktorou budú prechádzať všetky pakety, prenášané v rámci tohto virtuálneho spoja. Potom je zabezpečené, že pakety sú prenášané v rovnakom poradí, v akom boli vysielané a je možné použiť všetky skôr opísané metódy potvrdzovania. Pri vytváraní virtuálneho spoja sa tiež rýchlo reaguje na neprítomnosť alebo nepripravenosť protistanice k príjmu, pri uzatváraní virtuálneho spoja sa zaručí dokončenie prenosu všetkých správ.

6.2.1.6 Princípy prístupových metód

Ak je na vytvorenie počítačovej siete použité zdieľané elektrické alebo optické vedenie, prípadne i rádiový kanál, je potrebné vytvoriť metódu, ktorá umožní prenášať údaje medzi ľubovoľnými účastníkmi bez toho, aby boli rušení vysielaním inej stanice.
Používajú sa tieto prístupové metódy:

- *Statické prideľovanie*. Kapacita prenosového spoja je pevne rozdelená do častí, pridelených jednotlivým účastníkom. Rozdelenie môže byť *buď frekvenčné* alebo *časové*.

Frekvenčný multiplex (FDMA) rozdelí celkovú kmitočtovú šírku kanála do niekoľkých pásiem, pridelených jednotlivým staniciam. Tento spôsob je skôr typický pre prenos rôznych typov informácie (hlas, údaje). *Časový mulíiplex (TDMA)* prideľuje prenosový kanál na istý čas jednej stanici. Jeden úsek musí byť venovaný synchronizačnej správe, nutnej k jednoznačnej identifikácii príslušných úsekov pre jednotlivé stanice. Časový multiplex sa využíva napr. aj družicovými sieťami. Obidve metódy nedokážu plne využiť prenosovú kapacitu spoja a vedú k oneskoreniu paketov zúžením šírky pásma jednotlivého kanála pri *FDMA* alebo nutnosťou čakať na príslušný časový úsek pre *TDMA.*

Centrálne prideľovanie. Ak existuje v sieti jedna centrálna stanica, táto môže byť poverená úlohou prideľovať kapacitu prenosových kanálov tým podriadeným staniciam, ktoré ju skutočne potrebujú. Výhodou je celkové lepšie využitie prenosovej cesty, časť kapacity je však potrebné venovať na prenos požiadaviek. Existujú dva varianty - *prideľovanie na žiadosť* a *prideľovanie na výzvu (pooling).* Pri *prideľovaní na žiadosť* má každá stanica pre seba vyhradenú malú časť prenosovej kapacity kanála, v ktorej môže kedykoľvek žiadať centrálnu stanicu o pridelenie voľného prenosového kanála. Po potvrdení potom uskutoční prenos údajov, uvoľnenie kanála oznámi opäť centrálnej stanici. *Prideľovanie na výzvu (pooling)* plne riadi centrálna stanica, ktorá sa periodicky spytuje všetkých podriadených staníc, či nemajú údaje na vyslanie. Ak áno, dopytovaná stanica ich hneď odošle. Ak nemá údaje pripravené na vysielanie, odpovie iba potvrdzovacím paketom alebo neodpovie vôbec.

Náhodný prísíup. Metódy, ktoré využívajú náhodný prístup, nepotrebujú činnosť žiadnej centrálnej stanice. Okamih svojho prístupu na kanál určujú všetky stanice samostatne na základe vlastného odhadu. Nedokážu síce vylúčiť situácie, keď je vysielanie jednej stanice znehodnotené súčasným vysielaním inej stanice, ale snažia sa počet týchto kolízií minimalizovať a udržať čo najväčšiu priepustnosť dát aj pri veľkom zaťažení kanálov. Metóda *CSMA (Carrier Sense Multiple Access)* znižuje počet kolízií tým, že pred vysielaním testuje stanica obsadenosť kanála. Ak je kanál voľný, začne vysielať, ak nie, čaká na jeho uvoľnenie. Táto metóda náhodného prístupu sa často uplatňuje v lokálnych sieťach, kde je test obsadenosti kanála ľahký (napr. sieť *Ethernet).* Je zrejmé, že ani tento prístup nemôže vylúčiť vznik kolízie. Tá vzniká ako dôsledok stavu, ak dve stanice testujú

stav kanála v časovom odstupe menšom, ako je doba šírenia signálu medzi nimi. Preto sa používajú rozšírenia základnej metódy (*CSMA/CA*, *CSMA/CD*).

- *Distribuované prideľovanie.* Metódy, ktoré používajú distribuované prideľovanie, zaručujú bezkonfliktné prideľovanie zdieľaného kanála jednotlivým staniciam tak, že je možné zaručiť aj maximálnu dobu oneskorenia paketu. Pritom nie je potrebná existencia centrálnej stanice, ktorej výpadok by spôsobil zrútenie celej siete. Algoritmus prideľovania kanála sa vykonáva na všetkých zúčastnených staniciach *súčasne (synchrónne metódy, logický kruh, kruhové siete)*.

6.2.1.7 Prenosové médiá počítačových sietí

V súčasnosti najpoužívanejšie prenosové médiá počítačových sietí sú:

- *Telefónne linky* - malá prenosová rýchlosť (bežne do *9600 bit.s'*, výnimočne až do *512 kbit.s'')*, malá spoľahlivosť prenosu, prenos na veľké vzdialenosti.

- *Symetrické vedenia* - rýchlosť prenosu do $10Mbit.s^{-1}$, spojenie na vzdialenosť < *1 km*.

- *Nesymetrické vedenie (koaxiálny kábel)* - rýchlosť prenosu *10* až $20Mbit.s^{-1}$, spojenie na vzdialenosť rádovo *stovky m*. Je to typické prenosové médium s dobrou odolnosťou proti rušeniu.

- *Pozemné všesmerové rádiové kanály* - rýchlosť prenosu do *9600 bit.s'*. Používané frekvencie *100 MHz až 1 GHz*. Používajú sa na pripojenie mobilných terminálov resp. počítačov.

- *Pozemné smerové rádiové kanály* - rýchlosť prenosu do $10\ Mbit.s^{-1}$, spojenie do vzdialeností *desiatok km*. Používané frekvenčné pásma *4/6 GHz* a *12/14 GHz*.

- *Družicové spoje* - prenosové rýchlosti a pásma ako v predchádzajúcom prípade. Spojenie na veľmi veľké vzdialenosti *(tisícky km).* Majú *značné dopravné oneskorenie (~270 ms)*, zapríčinené veľkou vzdialenosťou družice od zemského povrchu.

- *Optické vlákna* - vysoké prenosové rýchlosti *(do 200Mbit.s^{-1})*, spojenie na vzdialenosti *rádovo km.*

6.2.1.8 Normalizácia počítačových sietí

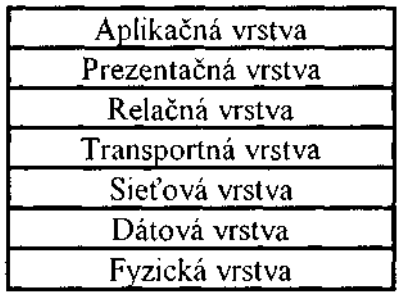
Rozvoj komunikácie medzi počítačmi uviedol do praxe niekoľko typov počítačových sietí, veľa z nich však vzniklo ako uzavretá sieť pre počítače jedného výrobcu - *SNA* pre sálové počítače *IBM, DNA* pre počítače firmy *DEC* atď. Pre používateľov na príslušných počítačoch je sieť úplne transparentná, ak je však potrebné pripojiť iný systém, je nutné použiť špeciálne prostriedky, ktoré však môžu znižovať výkonnosť.

Podobné problémy vznikli aj pri osobných počítačoch, kde existuje jednak veľa rôznych technických riešení siete a jednak rôzne typy programového vybavenia siete. Z hľadiska používateľa by bolo ideálne, keby na jednu sieť mohli byť štandardne pripojené rôzne počítače s rôznymi operačnými systémami - napr. osobné počítače triedy *PC* s operačným systémom *DOS,* pracovné stanice s *UNIX-om*, strediskové počítače atď. a mohli vzájomne zdieľať údaje bez použitia špeciálnych komunikačných prostriedkov.

V tejto súvislosti si je potrebné uvedomiť, že nestačí iba pripojiť všetky počítače na rovnaké technické vyhotovenie siete, ale je nutné použiť aj zodpovedajúce komunikačné programy. V opačnom prípade by si síce počítače rôznych systémov mohli vymieňať po sieti pakety, ale nerozumeli by ich obsahu.

Medzinárodný normalizačný úrad (International Standards Organization - ISO) vypracoval preto tzv. *referenčný model OSI (Reference Model for Open System Interconnection).* Jeho základom je rozdelenie všetkých činností pri výmene dát do 7 *častí - vrstiev,* ktoré začínajú *fyzickou definíciou spoja* a končia *aplikačnými programami.* Každá vrstva definuje vlastnosti svojich dvoch rozhraní - t.j. *služby poskytované* vyššej vrstve **a** *služby požadované* od nižšej vrstvy.

Referenčný model OSI je nakreslený na obr. 64.



OBR. 64. Referenčný model OSI

6.2.1.9 Lokálne počítačové siete

S hromadným rozšírením osobných počítačov vznikla potreba ich prepojenia tak, aby mohli *zdieľať spoločné údaje* a niektoré *periférne zariadenia.* Obyčajne ide o prepojenie na malé vzdialenosti - v rámci budovy, podniku atď. Použitie siete s prepojovacími uzlami je preto neefektívne. Naopak, výhodným sa javí prepojenie pomocou kábla s vysokou prenosovou rýchlosťou.

Lokálne počítačové siete (LAN - Local Area Network) preto väčšinou používajú mnohonásobný prístup k *zdieľanému prenosovému médiu.* Všetky počítače, ktoré pracujú ako stanice v lokálnej sieti, sú pripojené na spoločný kábel a vysielajú po ňom správy ostatným staniciam. Pritom trvale sledujú činnosť v sieti a tým získavajú im adresované správy. Vysielať v danej chvíli môže len jedna stanica - toto právo sa prideľuje rôznymi metódami podľa typu siete. Vzhľadom k vysokým prenosovým rýchlostiam *(rádovo desiatky Mbit/s)* je práca v *LAN* z pohľadu účastníkov prakticky bez zdržania.
V súčasnosti je rozšírených viac typov lokálnych počítačových sietí, napr. *Ethernet, IBM Token Ring, AppleTalk, STARLAN, ISN, ISDN, ARCnet, 10Net, Vines, LANtastic* atď.

Servery a pracovné stanice

Technické prostriedky bežných *LAN* považujú všetky pripojené počítače za rovnocenné, žiadny z nich nemá pridelenú riadiacu úlohu. Programové vybavenie však už zavádza dve triedy počítačov - *servery* a *pracovné stanice.*

Servery sú počítače, ku ktorým sú pripojené *zdieľané zariadenia* alebo ktoré obsahujú *zdieľané údaje*. Servery potom *poskytujú svoje služby* ostatným počítačom, pripojeným do siete - *pracovným staniciam*.

Na serveroch sa vykonáva *špeciálne sieťové programové vybavenie*. Toto môže, ale nemusí umožniť využitie servera aj ako pracovnej stanice. Ak áno, hovoríme o *nevyhradenom (non-dedicated) serveri,* v opačnom prípade o *vyhradenom (dedicated) serveri*. Nevyhradený server síce znamená o jednu pracovnú stanicu naviac, zhoršuje sa však výkonnosť počítača ako servera. Okrem toho pri spustení chybného programu môže prísť k zablokovaniu činnosti servera pre celú sieť.

Používa sa niekoľko typov serverov:

- *Súborový server (File Server)* je základný typ servera. Pracuje so službami typu *otvor súbor, čítaj zo súboru, zapíš do súboru*. Z hľadiska používateľa pracovnej stanice sa zdieľaný disk na súborovom serveri správa podobne, ako keby išlo o disk lokálny - je mu pridelené meno zariadenia (napr *F:),* je možné *zisťovať obsah jeho adresárov, kopírovať súbory, mazať* atď. Súborový server ďalej zaisťuje *zdieľanie súborov* a *ochranu prístupových práv*.

- *Tlačový server (Print Server)* poskytuje ostatným účastníkom v sieti svoju lokálnu tlačiareň, príp. iné grafické výstupné zariadenie. Používatelia siete môžu posielať svoje súbory na tlačiareň buď pomocou špeciálnych príkazov alebo môžu trvale presmerovať všetky požiadavky na tlač na tlačový server.

- *Komunikačný server (Com Server)* poskytuje prostredníctvom svojich sieťových adaptérov možnosť prístupu na vonkajšiu spojovaciu sieť a cez ňu prístup k vzdialeným počítačom resp. sieťam.

- *Server pre zálohovanie (Back-up Server)* má pripojenú veľkokapacitnú vonkajšiu pamäť, na ktorú je možné zálohovať obsah pevných diskov v sieti (napr. *streamer, prepisovateľný optický disk* atď.).

Na *pracovných staniciach* sú zavedené *rezidentné sieťové programy,* ktoré transformujú požiadavky používateľov na vzdialené zariadenia na komunikáciu po sieti.
Niektoré štandardné sieťové operačné systémy (napr. *Novell NetWare)* uprednostňujú komunikáciu medzi serverom a pracovnou stanicou, komunikácia medzi dvoma pracovnými stanicami býva obmedzená napr. iba na posielanie jednoduchých správ.

Siete typu peer-to-peer

Okrem sietí, v ktorých sa nachádzajú uvedené dve triedy počítačov, sa používajú aj siete typu *peer-to-peer.* Ako *peer-to-peer* sa označujú siete, ktoré *nemusia obsahovať centrálny súborový server.* Skladajú sa z rovnocenných pracovných staníc, ktoré navzájom zdieľajú svoje disky a periférne zariadenia.
Príkladom siete typu *peer-to-peer* je sieť *LANtastic.*

Zoznam použitej literatúry

[1] Blatný, V. a kol.: Číslicové počítače. ALFA, Bratislava 1982

[2] Brandejs, M.: Mikroprocesory INTEL. Pentium a spol. GRADA, Praha 1994

[3] Burger, L: Stykové obvody mikropočítačov. ALFA, Bratislava 1990

[4] De Blasi, M.: Computer Architecture. Addison - Wesley Publishing Company, Wokingham 1990

[5] Frištacký, N. - Jelšina, M.: Číslicové počítače. ALFA, Bratislava 1993

[6] Frištacký, N. - Kolesár, M. - Kotočová, M.: Číslicové počítače. Logický návrh číslicových počítačov. EF SVŠT, Bratislava 1988

[7] Hlavička, J.: Číslicové počítače II. ČVUT, Praha 1987

[8] Horváth, P. - Linhart, M.: Periférne zariadenia číslicových počítačov. ALFA, Bratislava 1990

[9] Horváth, P. - Kotočová, M.: Dátové siete, protokoly, služby a ochrana prístupu. Vydavateľstvo STU, Bratislava 1996

[10] Hwang, K. - Briggs, F. A.: Computer Architecture and Parallel Processing. McGraw - Hill Book Company, London 1992

[11] Irwin, G. W. - Fleming, P. J.: Transputers in Real-Time Control. Research Studies Press Ltd., Taunton, Somerset 1992

[12] Jelšina, M.: Mikropočítače a počítačové systémy. EF VŠT, Košice 1987

[13] Minasi, M.: Pevné disky od A po Z. GRADA, Praha 1992

[14] Náučný slovník elektrotechnický (4.zv.). Kybernetické systémy. ALFA, Bratislava 1987

[15] Schatt, S.: Počítačové sítě LAN od A po Z. GRADA, Praha 1994

[16] Sokolowsky, P. - Šedivá, Z.: Multimédia. GRADA, Praha 1994

[17] Šnorek, M. - Richta, K.: Připojování periferií k PC. GRADA, Praha 1996

[18] Šubrt, V.: Jednočipové mikropočítače INTEL 8048 - 8096. GRADA, Praha 1992

[19] Transputer Databook. INMOS. Redwood Burn Ltd., Trowbridge 1989

[20] Wilkinson, B.: Computer Architecture. Design & performance. Prentice Hall Int. Ltd., London 1991

Obsah

EDICIA SKRIPT

Ing. Tibor Krajčovič, CSc.

POČÍTAČE

2. vydanie
Náklad 600 výtlačkov
157 strán, 64 obrázkov, 9,897 AH, 10,140 VH
Edičné číslo 4845
Tlač Vydavateľstvo STU v Bratislave
Rok vydania 2000

| 85-258-2000 Sk 26,-- [

ISBN 80-227-1399-6
ISBN 80-227-0924-7 (1. vyd., r. 1997)