Peter Agh

Princípy počítačov

Úvod

Úvodný text...

Časť I

Matematické základy

Aj keď sa (hlavne v minulosti) hardware a software počítačov chápali oddelene, tieto dva pojmy sú nerozlučne spojené. Poznanie vzájomného vzťahu hardwaru a softwaru, ako aj princípov počítačov je cieľom tejto práce. Potrebný matematický základ pre osvojenie týchto vedomostí čitateľ získa v tejto časti.

Náplňou tejto časti sú niektoré teoretické aspekty súvisiace s činnosťou počítača, ako je kódovanie informácií; špeciálne kódovanie čísel a aritmetika.

Čitateľ dozaista pozná pojmy *informácia, reprezentácia informácie, kód*. Takisto vie, aké druhy informácií sa reprezentujú v počítačoch. Preto tieto pojmy nebudeme formálne zavádzať, postačovať bude ich intuitívna znalosť. Výklad začneme číselnými sústavami. Ďalšia kapitola bude venovaná logike, ktorá –ako neskôr uvidíme– je 'základným stavebným kameňom' dnešných počítačov. Potom sa podrobne budeme venovať kódovaniu čísel a realizácií aritmetických operácií v jednotlivých kódoch – najskôr pohovoríme o spôsoboch kódovania celých čísel so znamienkom i bez znamienka a o aritmetických algoritmoch pre čísla v jednotlivých kódoch. Opíšeme aj spôsoby kódovania reálnych čísel, algoritmy pre reálnu aritmetiku, výpočet zložitejších matematických funkcií a pohovoríme aj o možných 'úskaliach' reálnej aritmetiky realizovanej na počítači.

Kapitola 1

ČÍSELNÉ SÚSTAVY

Čísla sa zapisujú pomocou reťazcov znakov. Význam reťazca pritom závisí od toho, aká konvencia (číselná sústava) sa pri zápise používa.

1.1 Pozičné a nepozičné číselné sústavy

Číselné sústavy možno rozdeliť na pozičné a nepozičné.

- V *pozičnej* číselnej sústave je každá číslica v zápise čísla charakterizovaná svojou polohou vzhľadom na rádovú čiarku. Presnejšie, v sústave základom Z môžeme reálne číslo r zapísať v tvare:

  $$r = a_n \cdot Z^n + a_{n-1} \cdot Z^{n-1} + \ldots + a_0 \cdot Z^0 + a^{-1} \cdot Z^{-1} + \ldots + a^{-m} \cdot Z^{-m},$$

  kde $a_i \in \{0, 1, 2, \ldots, Z-1\}$.

  Hovoríme, že číslica a_i *má váhu* Z^i. Tento zápis sa zvyčajne skracuje na vypísanie koeficientov (číslic), pričom číslice $a_{n-1}, a_{n-2}, \ldots a_0$ sa od číslic $a_{-1}, \ldots, a_{-m}$ sa oddeľujú rádovou čiarkou:

  $$r = a_{n-1}a_{n-2}\ldots a_1 a_0, a_{-1}\ldots a_{-m}$$

  Príklad I.1: v desiatkovej číselnej sústave môžeme vyjadriť čísla 123 a -50.6 ako:

  $$123 = 1.10^2 + 2.10^1 + 3.10^0$$
  $$-50.6 = (-5.10^1) + (-6.10^{-1})$$

- V *nepozičnej* číselnej sústave pozícia číslice v zápise čísla neurčuje jej váhu. Zápis čísla sa skladá zo zreťazenia zápisov niekoľkých čísel a výsledné číslo dostaneme sčítaním týchto čísel.

  Príkladom nepozičnej číselnej sústavy je rímska číselná sústava. V čísle *XIII* síce rozoznáme znak *X* pre číslo 10, no tento znak môže mať aj iný význam (napr. *IX* znamená 9, *XI znamená* 11, *XXX znamená* 30*) a až prečítaním celého zápisu môžeme určiť hodnotu znaku X.*

Príklad I.2: vyjadrenia niektorých čísel v rímskej číselnej sústave: Rímska sústava: $1 = I$, $2 = II$, $3 = III$, $4 = IV$, $5 = V$, $10 = X$, $50 = L$, $100 = C$, $1000 = M$:

$$123 = CXXIII$$

$$244 = CCXXXXIV$$

$$1968 = MLMXVIII$$

Prirodzene, v praxi sa častejšie používajú pozičné sústavy. Ako totiž možno nahliadnuť, v pozičných sústavách možno ľahko vykonávať aritmetické operácie (t.j. sčítať, odčítať, násobiť aj deliť), čo v nepozičných možno len ťažko (skúste vynásobiť dve rímske čísla– najjednoduchšie bude najskôr previesť ich do pozičnej sústavy, vynásobiť ich a previesť výsledok späť na rímske číslo). Pritom každá z uvedených operácií je v pozičnej sústave popísateľná jednoduchým algoritmom.

V bežnom živote používame *dekadickú* (desiatkovú) číselnú sústavu.

Počítače používajú *binárnu* (dvojkovú) číselnú sústavu. Ako neskôr uvidíme, umožňuje jednoduchú technickú realizáciu počítača, pretože operácie nad binárnou sústavou možno popísať pomocou logických operácií a tieto sa dajú realizovať jednoduchými elektrickými obvodmi. Zápis v binárnej sústave je však dlhý, preto sa (napr. v zápise niektorých programov) často využíva *hexadecimálna* (šestnástková) číselná sústava, v ktorej majú čísla kratší zápis a navyše je možné ľahko prevádzať čísla z dvojkovej do šestnástkovej sústavy a naopak.

Venujme sa teraz práve prevodom vyjadrení čísel medzi rôznymi číselnými sústavami. Uvedieme dve metódy, metódu postupného odčítania a metódu delenia.

1.2 Prevody medzi číselnými sústavami

metóda postupného odčítania

Nech je dané číslo r, ktorého zápis v sústave so základom B označíme r_B. Chceme ho vyjadriť v sústave C. Prvou metódou, ako previesť do sústavy so základom C je metóda postupného odčítania. Od čísla r budeme postupne odčítovať násobky[1] stále sa zmenšujúcich mocnín základu C, pričom r hľadáme najväčšie také násobky mocnín, ktoré sú ešte menšie, nanajvýš rovné ako prevádzané číslo.

Poznámka I.1: Z matematického hľadiska je jedno akým spôsobom je číslo vyjadrené, ale v počítači hrá spôsob reprezentácie dôležitú úlohu, má zmysel rozlišovať číslo a jeho reprezentáciu, t.j. hovoriť pre číslo x o jeho reprezentácií r. Kvôli stručnosti zápisu, namiesto výrazu r_B je 'zápis čísla v sústave B' budeme jednoducho hovoriť, že r_B je 'číslo v sústave B'.

Príklad I.3:

[1] nultý až $(C-1)$-vý násobok

$195_{10} \hookrightarrow R_2$

		195	
$2^7 = 128$	$-$	$\underline{128}$	$\longrightarrow 1$
		67	
		67	
$2^6 = 64$		$\underline{64}$	$\longrightarrow 1$
		3	
		3	
$2^5 = 32$	príliš veľké		$\longrightarrow 0$
		3	
$2^4 = 16$	príliš veľké		$\longrightarrow 0$
		3	
$2^3 = 8$	príliš veľké		$\longrightarrow 0$
		3	
$2^2 = 4$	príliš veľké		$\longrightarrow 0$
		3	
$2^1 = 2$		$\underline{2}$	$\longrightarrow 1$
		1	
		1	
$2^0 = 1$		$\underline{1}$	$\longrightarrow 1$
		0	

Odtiaľ $195_{10} = 11000011_2$. Čiže číslo 195 sme vyjadrili ako súčet

$$1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \quad = \quad 195$$

V pozičnej sústave sa dajú vyjadiť aj racionálne čísla. Uvedený postup prevodu možno použiť aj pri prevode čísla so zlomkovou časťou –odčítavame nielen kladné, ale aj záporné mocniny základu C. Najskôr odrátavame mocniny základu s čoraz menším kladným exponentom[2] (čím prevádzame celú časť čísla R) a potom odrátavame (záporné) mocniny základu C s čoraz menším záporným exponentom[3] (čím prevádzame desatinnú časť čísla R).

Príklad I.4: $195.625_{10} = 11000011.101_2$. Čiže číslo 195.625 sme vyjadrili ako súčet

$$1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} \quad = \quad 195$$

[2] t.j. $C^n, C^{n-1}, C^{n-2} \ldots C^1, C^0$
[3] t.j. $C^{-1}, C^{-2}, C^{-3} \ldots$

metóda postupného delenia

Nech je dané (celé) číslo R. Označme R_B jeho vyjadrenie v sústave so základom B a v sústave C ako R_C. Pre R_B a R_C platí:

$$R_B = a_{n-1}B^{n-1} + a_{n-2}B^{n-2} + \ldots + a_2B^2 + a_1B^1 + a_0$$
$$R_C = b_{m-1}C^{m-1} + b_{m-2}C^{m-2} + \ldots + b_2C^2 + b_1C^1 + b_0$$

Na základe vyjadrenia R_B nájdeme vyjadrenie R_C.
Ak vydelíme R_B základom C, dostaneme podiel Q_1 a zvyšok R_1:

$$R_C = Q_1C + R_1$$

Teda

$$R_C = C \cdot [a_{n-1}C^{n_2} + a_{n-2}C^{n_3} + \ldots + a_1C^0] + a_0$$

Zvyšok R_1 predstavuje koeficient a_0.
Ak ďalej vydelíme podiel Q_1 základom C, dostaneme

$$Q_1 = C[a_{n-1}C^{n-3} + a^{n-2}C^{n-4} + \ldots + a_2] + a_1$$

Zvyšok R_2 predstavuje koeficient a_1. Ďalej pokračujeme analogicky.
Príklad I.5:

$$1242_{10} \hookrightarrow R_{16}$$

- $1242/16 = 77$
  - zvyšok 10
- $77/16 = 4$
  - zvyšok 13
- $4/16 = 0$
  - zvyšok 4

Z toho $1358_{10} = 4DA$, čiže 1358_{10} sme vyjadrili ako súčet

$$4 * 256 + 13 * 16 + 10 * 1$$

.

Získané vedomosti o číselných sústavách – o pojme číselných sústav a o prevodoch medzi nimi využijeme pri ďalšom štúdiu kódovania čísel a realizácií aritmetiky.

Kapitola 2

LOGICKÉ FUNKCIE

Cieľom tejto kapitoly je podať prehľad informácií z matematickej logiky, potrebných pre pochopenie činnosti počítača. Našu pozornosť zameriame na jednu konkrétnu oblasť logiky– *výrokovú logiku* (alebo *výrokový počet*). Iné logické teórie (ako napr. *predikátový počet*) pre naše účely nebudú potrebné. Kvôli zjednodušeniu zápisov budeme špecifikáciu 'výroková' vynechávať, t.j. namiesto spojenia 'výroková logika' budeme hovoriť len o 'logike'.

2.1 Logické premenné a základné operátory

Najskôr zopakujme základné pojmy výrokového počtu.

Základným pojmom logiky je *výrok*. Výrok je tvrdenie, ktoré môže byť buď pravdivé alebo nepravdivé. Výroky budeme označovať písmenami P,Q,R ...

Z výrokov možno pomocou *logických spojok* (negácia, konjukcia, disjunkcia, implikácia a iné) vytvoriť nový výrok (formálnu definíciu uvedieme neskôr).

Pretože výrok P nadobúda len dve možné hodnoty (*pravda, nepravda)*, možno ho považovať za premennú p nadobúdajúcu hodnotu z množiny {*pravda, nepravda*}. Takúto premennú budeme nazývať *logická premenná*.

Množinu {*pravda, nepravda*} môžeme reprezentovať aj inou dvojprvkovou množinou - { *true, false* } či $\{0,1\}$, pričom 1 bude znamenať *'pravda'* a 0 *'nepravda'*. Na základe toho môžeme logické premenné pokladať za premenné nadobúdajúce hodnoty z množiny $\{0,1\}$. Takisto funkcie na $\{0,1\}$ nazývame *logické funkcie* (alebo *operácie*) v zmysle nasledovnej definície:

Definícia I.1: Funkciu f, ktorej definičným oborom je množina $\{0,1\}^N$, kde $N \in \mathcal{N}$ a ktorej oborom funkčných hodnôt je množina $\{0,1\}$, nazývame *logickou* (*booleovskou*) *funkciou N premenných*.

Základné unárne a binárne logické funkcie sú:

- logický súčin (AND, označenie $p \cdot q$)
- logický súčet (OR, ozn. $p + q$)
- negácia (NOT, ozn. p', $\neg p$ alebo $\bar{a}$)

x	y	x *AND* y
0	0	0
1	0	0
0	1	0
1	1	1

x	y	x *OR* y
0	0	0
1	0	1
0	1	1
1	1	1

x	*NOT* x
0	1
1	0

Tabuľka 2.1: Logické funkcie *logický súčin, logický súčet* a *negácia*

Vo výrokovej logike sa používajú aj ďalšie funkcie:

- implikácia ($p \Longrightarrow q$)
- ekvivalencia ($p \Longleftrightarrow q$)
- nonekvivalencia (XOR, $p \oplus q$)
- negovaný logický súčet (p *NOR* q)
- negovaný logický súčin (p *NAND* q)

x	y	x *XOR* y
0	0	0
1	0	1
0	1	1
1	1	0

x	y	x *NAND* y
0	0	1
1	0	1
0	1	1
1	1	0

x	y	x *NOR* y
0	0	1
1	0	0
0	1	0
1	1	0

Tabuľka 2.2: Logické funkcie *implikácia, ekvivalencia* a *nonekvivalencia*

Vo výrokovej logike sa tvrdenia (výroky) zapisujú pomocou tzv. *formúl.* V nasledujúcej definícii popíšeme formuly obsahujúce konjukciu, disjunkciu a negáciu[1], ktoré predstavujú zápis zložených výrokov.

Definícia I.2: *Formuly výrokového počtu*[2] definujeme nasledovne:

1. každá logická premenná je (elementárna) formula
2. ak P je formula, tak aj $\neg P$ je formula
3. ak P a Q sú formuly, tak $P + Q$ aj $P \cdot Q$ sú formuly
4. ak P je formula, tak aj (P) je formula
5. formula výrokového počtu je ľubovoľný (konečný) výraz vytvorený pomocou konečnej postupnosti pravidiel 1–4

Dohoda: Slová logická (booleovská) budeme vynechávať, t.j. pokiaľ to nepovedie k nejednoznačnosti, budeme hovoriť len o premenných, výrazoch a funkciách.

Poznámka I.2: Vidíme, že výrokovú logiku možno popísať špeciálnou algebrou nad

[1] analogicky možno definovať aj formuly obsahujúce ďalšie logické funkcie

oborom $\{0,1\}$, ktorú nazývame *Boolova algebra.*

Definícia I.3: Dva výrazy nazývame *ekvivalentné*, ak pre každú kombináciu hodnôt premenných vystupujúcich v tomto výraze sa výsledky výrazov rovnajú.

Cvičenie I.1: Dokážte základné vlastnosti logických funkcií (rovnosť chápeme ako ekvivalenciu ľavej a pravej strany).

a, $x + 0 = x, \quad x \cdot 0 = 0$
b, $x + 1 = 1, \quad x \cdot 1 = x$
c, $x + x = x, \quad x \cdot x = x$
d, $x + \neg x = 1$ (zákon vylúčenia tretieho)
e, $x \cdot \neg x = 0$

Cvičenie I.2: : Dokážte nasledovné vzťahy:

a, $x + y = y + x$ (komutatívnosť)
b, $x \cdot y = y \cdot x$
c, $x + y + z = (x + y) + z = x + (y + z)$ (asociatívnosť)
d, $x \cdot y \cdot z = (x \cdot y) \cdot z = x \cdot (y \cdot z)$
e, $(x + y) \cdot y = x \cdot y + y = x + y$ (distributívnosť)
f, $x \cdot (x + y) = x \cdot y + y = x + y$

Cvičenie I.3: : Dokáže De Morganove zákony

a, $\overline{x + y + z + \ldots} = \overline{x} \cdot \overline{y} \cdot \overline{z} \ldots$

b, $\overline{x \cdot y \cdot z \cdot \ldots} = x + y + z + \ldots \overline{x} + \overline{y} + \overline{z} \ldots$

Cvičenie I.4: Z predchádzajúceho vyplýva nasledovný zákon (zovšeobecnený tvar De Morganových zákonov). Dokážte ho:

Nech $f(x, y, z, +, .)$ označuje výraz obsahujúci len premenné x,y,z

$$\overline{f(x, y, z, \ldots, +, \cdot)} = f(\neg x, \neg y, \neg z, \ldots, \cdot, +)$$

Príklad I.6: Podľa predchádzajúceho cvičenia platí napríklad, že:

$$\overline{((x + y).z + \neg x)} = (\neg x \cdot \neg y + \neg z) \cdot x$$

2.2 Definícia logickej funkcie

Každej formule možno po dosadení hodnôt z množiny $\{0,1\}$ za premenné jednoznačne priradiť hodnotu z množiny $\{0,1\}$. Formula teda predstavuje funkciu s definičným oborom $\{0,1\}$ a oborom hodnôt $\{0,1\}$. Takúto funkciu nazývame *logická funkcia.* Logickú funkciu jednej premennej, ktorej definičným oborom i oborom hodnôt je množina $\{0,1\}$ nazývame *unárnou logickou funkciou.* Logickú funkciu dvoch premenných, ktorej

x	f_0	f_1	f_2	f_3
0	0	0	1	1
1	0	1	0	1

Tabuľka 2.3: Unárne logické funkcie

definičným oborom je množina $\{0,1\}\times\{0,1\}$ a oborom hodnôt $\{0,1\}$ nazývame *binárnou* logickou funkciou. Logickú funkciu troch premenných, ktorej definičným oborom je množina $\{0,1\}^3$, nazývame *ternárnou* logickou funkciou. Analogicky možno definovať *n-árnu* logickú funkciu. Pre $n = 0$ hovoríme o *konštantných* logických funkciách (ktoré sú práve dve, konštanty 0 a 1).

Logická funkcia je *úplne zadaná*, ak je známa jej hodnota pre všetky možné kombinácie hodnôt premenných. Môžeme ju popísať *tabuľkou* (kde je pre každú kombináciu argumentov funkcie uvedený jej výstup), alebo *logickým výrazom* (výraz, ktorý nadobúda hodnotu 0 alebo 1) obsahujúcim len premenné ktoré sú argumentami funkcie.

Príklad I.7: Funkciu *AND* možno popísať tabuľkou (1.3) a výrazom $x \cdot y$, t.j $AND\ (x,y) = x \cdot y$

Neskôr narazíme na prípady, keď nás nebude zaujímať hodnota funkcie pre všetky kombinácie vstupov. Pre 'nezaujímavé vstupy' bude môcť funkcia mať ľubovoľný výstup. Povieme, že výstup je *ľubovoľný*, alebo *nedefinovaný* (výstup nie je určený). O takejto logickej funkcii hovoríme, že je *neúplne zadaná*.

Pozrime sa teraz bližšie na unárne a binárne logické funkcie. Koľko je všetkých možných unárnych a binárnych funkcií? A ako vyzerajú?

Cvičenie I.5: Dokážte, že všetkých n-árnych funkcií je 2^{2^n}.

unárne funkcie

Existujú 4 unárne funkcie. Sú uvedené v pravdivostnej tabuľke 2.3.

Funkcie f_0 a f_3 sú konštanty 0 a 1, funkcia f_1 je *funkcia identity*[3] a funkcia f_2 je *negácia*[4].

binárne funkcie

Binárnych funkcií je 16 a sú uvedené v tab. 2.4.

Medzi funkciami nájdeme niektoré známe funkcie:

- $f_0(x,y) = 0$ a $f_{15}(x,y) = 1$ konštanty *true, false*

[3] pre $(\forall x)\ \ f(x) = x$
[4] pre $(\forall x)\ \ f(x) = \neg x$

x	y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Tabuľka 2.4: Binárne logické funkcie

- $f_3(x,y) = x$ a $f_5(x,y) = y$ identické funkcie
- $f_{12}(x,y) = x'$ a $f_{10}(x,y) = y'$ negácia premennej
- $f_7(x,y) = x + y = x\ OR\ y$ logický súčet
- $f_1(x,y) = xy = x\ AND\ y$ logický súčin
- $f_6(x,y) = xy' + x'y$ logický súčet vo vylučovacom význame[5] - XOR $(x \oplus y)$.
- $f_9(x,y) = x'y' + xy$ ekvivalencia $(x \equiv y)$
- $f_8(x,y) = (x+y)'$ funkcia ani jeden nie je (NOR)
- $f_{14}(x,y) = (xy)'$ funkcia aspoň jeden nie je ($NAND$)

Cvičenie I.6: Vytvorte tabuľku všetkých ternárnych funkcií a vyjadrite ich pomocou výrazov obsahujúcich len spojky *NOT*, *AND*, *OR*.

Cvičenie I.7: Zopakujte predchádzajúce dve cvičenia, ak sa obmedzíme na výrazy obsahujúce len spojku *NAND*.

Cvičenie I.8: Ako predchádzajúce cvičenie, no pre spojku *NOR*.

Disjunktívna a konjunktívna normálna forma

Z prechádzajúcich cvičení vyplýva, že každú n-árnu funkciu možno vyjadriť výrazom obsahujúcim len operácie *AND*, *OR* a *NOT*. Možno však dosiahnuť aj to, aby tento výraz mal špeciálny tvar, o čom hovorí nasledujúca veta:

Definícia I.4: *Literálom* nazývame premennú alebo negáciu premennej.

Definícia I.5: Výraz P je v *disjunktívnej normálnej forme*, ak sa skladá zo súčtu podvýrazov p_i, čo sú súčiny navzájom rozličných literálov.

Definícia I.6: Výraz Q je v *konjunktívnej normálnej forme*, ak sa skladá zo súčinu podvýrazov q_i, čo sú súčty navzájom rozličných literálov.

Príklad I.8:

- výraz $(x\bar{y}z) + (xy\bar{z}) + (xyz)$ je v d.n.f.
- výraz $(x+y+z)(x+y+\bar{z})(x+\bar{y}+z)(\bar{x}+y+z)$ je v k.n.f.

[5] alebo tiež sčítanie modulo 2

Veta I.1: Každá logická funkcia sa dá zapísať výrazom v tvare *disjunktívnej* normálnej formy (základný súčtový tvar) a *konjunktívnej* normálnej formy (základný súčinový tvar).

Disjunktívna normálna forma vyjadruje funkciu ako logický súčet súčinov (resp. disjunkciu konjunkcií) premenných. Jednotlivé súčiny predstavujú tie kombinácie hodnôt premenných, pre ktoré funkcia nadobúda hodnotu 1. Každý z nich je zapísaný ako súčin priamych a negovaných premenných tak, aby sám dával hodnotu 1.

Konjunktívna normálna forma vyjadruje funkciu ako logický súčin súčtov (resp. konjunkciu disjunkcií) premenných. Jednotlivé súčty predstavujú tie kombinácie hodnôt premenných, pre ktoré funkcia nadobúda hodnotu 0. Každý z nich je zapísaný ako súčet priamych a negovaných premenných tak, aby sám dával hodnotu 0.

Príklad I.9:

Nech je daná ternárna funkcia (popísaná nasledovnou tabuľkou)

x	**y**	**z**	**f**(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

a) Disjunktívna normálna forma

Prípady, keď sa funkcia rovná 1	Zodpovedajúce súčiny (elementárne konjunkcie)
x y z	
0 1 1	$\bar{x} \cdot y \cdot z$
1 0 1	$x \cdot \bar{y} \cdot z$
1 1 0	$x \cdot y \cdot \bar{z}$
1 1 1	$x \cdot y \cdot z$

Teda $f = x\bar{y}z + xy\bar{z} + xyz$ (zápis v DNF)

b) Konjunktívna normálna forma

Prípady, keď sa funkcia rovná 0	Zodpovedajúce súčty
x y z	
0 0 0	$x+y+z$
0 0 1	$x+y+\bar{z}$
0 1 0	$x+\bar{y}+z$
1 0 0	$\bar{x}+y+z$

Teda $f=(x+y+z)(x+y+\bar{z})(x+\bar{y}+z)(\bar{x}+y+z)$ (zápis v KNF)

2.3 Zjednodušovanie zápisu logickej funkcie

Vyjadrenie logickej funkcie pomocou formuly nie je jednoznačné. Prirodzenou snahou je spomedzi všetkých možných formúl nájsť najjednoduchšiu formulu, t.j. formulu s najkratším zápisom.

Existuje viacero prístupov hľadania najkratšej formuly (minimalizácie). Spomenieme dve metódy minimalizácie: algebraickú minimalizáciu a minimalizáciu pomocou Karnaughových máp.

algebraická minimalizácia

Zakladá sa na algebraickej úprave výrazov. Využívajú sa pri nej rôzne vzťahy platiace v booleovej algebre, z ktorých najvýznamnejšie sme už uviedli v kapitole 2.3:

$$x+0=x, \qquad x\cdot 0=0$$
$$x+1=1, \qquad x\cdot 1=x$$
$$x+x=x, \qquad x\cdot x=x$$
$$x+\neg x=1 \qquad x.\neg x=0$$
$$(x+y).y=x.y+y=x+y$$
$$x.(x+y)=x.y+y=x+y$$

Príklad I.10: Funkcia f je zadaná v disjunktívnej normálnej forme

$$f=\bar{x}yz+x\bar{y}z+xy\bar{z}+xyz$$

Funkciu môžeme upraviť takto :

$$f=(\bar{x}yz+xyz)+(x\bar{y}z+xyz)+(xy\bar{z}+xyz)=$$

$$=yz(\bar{x}+x)+xz(\bar{y}+y)+xy(\bar{z}+z)=$$

$$= yz.1 + xz.1 + xy.1 = xy + yz + xz$$

Odtiaľ dostávame

$$f = xy + yz + xz$$

Pre funkcie s väčším počtom premenných sa však táto metóda sotva dá použiť; a to preto, lebo táto metóda nie je 'systematická', neposkytuje algoritmus pre nájdenie najkratšej formuly, ale je založená na 'hádaní' skupiny premenných a príslušného vzťahu[6], ktorého použitie v konečnom dôsledku povedie k zjednodušeniu výrazu.

Uľahčenie celého procesu zjednodušovania poskytuje metóda Karnaghových máp. Je založená na vhodnej grafickej reprezentácií logickej funkcie, vďaka čomu sa celý proces minimalizácie stáva jednoduchším a prehľadnejším.

Karnaughova metóda

Pri zjednodušovaní funkcie zväčša spájame súčiny, ktoré sa líšia v jedinej premennej, napr.

$$xy\bar{z}t + xyz\bar{t} = x.y.\bar{z}(\bar{t} + t) = x.y.\bar{z}$$

Karnaughova metóda tento proces 'vizualizuje'. Zakladá sa na vytvorení 'mapy' – tabuľky, v ktorej sú uvedené hodnoty booleovskej funkcie pre všetky možné vstupy (hodnoty vstupných premenných) a to v takom usporiadaní, že sa ľahko nájdu súčiny, ktoré sa líšia v jedinej premennej.

Karnaughova mapa pre funkciu n premenných obsahuje 2^n políčok. Každé políčko má adresu, ktorá predstavuje jednu kombináciu hodnôt vstupných premenných. Pre unárnu funkciu sú možné dve rozličné hodnoty vstupných premenných, pre binárnu 4, ternárnu 8, atď... Je zrejmé, že pre funkciu n-premenných možno ľahko zostrojiť Karnaughovu mapu v n-rozmernom priestore. Postačujúcim je však aj dvojrozmerný priestor, rovina– spôsob reprezentácie bude čitateľovi zrejmý z príkladu:

Na obrázku 2.1 je mapa pre funkciu 4 premenných. Políčku p zodpovedá vektor vstupných hodnôt $x = 1$, $y = 0$, $z = 1$, $t = 0$.

Dôležitým pojmom je tzv. *sused políčka*. Susedia políčka p sú políčka s adresami líšiace sa od adresy políčka p hodnotou práve jednej premennej. Mapy na obrázku 2.1 znázorňujú susedné políčka k danému políčku.

Mapa funkcie sa vytvorí tak, že do každého políčka Karnaughovej mapy sa zapíše hodnota funkcie f pre tú kombináciu premenných, ktorú políčko predstavuje. Napríklad, na obr. 2.2 je tabuľka funkcie a zodpovedajúca Karnaughova mapa.

Z algebraického hľadiska dve navzájom susedné políčka reprezentujú súčiny líšiace sa v práve jednom člene (napr. $xy\overline{zt} + xy\bar{z}t$). Preto pokiaľ je v oboch susedných políčkach 1, formulu môžeme zjednodušiť ($xy\overline{zt} + xy\bar{z}t = x.y.\bar{z}(\bar{t} + t) = x.y.\bar{z}$).

Proces minimalizácie je teda pomerne jednoduchý: hľadáme v mape oblasti 2, 4 alebo 8 susedných políčok tak, aby sa zo skupín súčinov vylúčila jedna, dve alebo tri premenné. Vytváranie oblastí musí využívať všetky políčka obsahujúce 1.

Všimnime si obrázok 2.2, kde je príklad funkcie a k nej prislúchajúcej Karnaughovej mapy.

$zt \backslash xy$	**00**	**01**	**10**	**11**
00		X		
01	X	■	X	
10		X		
11				p

$zt \backslash xy$	**00**	**01**	**10**	**11**
00		X		
01				
10		X		
11	X	■	X	

Uvažované políčko je označené ■ , jeho susedia sú označení znakmi 'X'.

$zt \backslash xy$	**00**	**01**	**10**	**11**
00	X			
01				
10	X			
11	■	X		X

Vyznačenie susedov daného políčka.

Obrázok 2.1: Karnaughove mapy

x	**y**	**z**	**t**	**f**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

$zt \backslash xy$	**00**	**01**	**11**	**10**
00	1	0	0	1
01	0	1	1	1
11	0	0	0	1
10	1	0	0	1

Obrázok 2.2: Funkcia f a jej Karnaghova mapa

$zt \backslash xy$	**00**	**01**	**10**	**11**
00				
01	1	1	–	1
10				
11			–	

Obrázok 2.3: Mapa neúplne zadanej funkcie

V mape môžeme vyznačiť tri oblasti jednotiek. Oblasť s dvoma jednotkami[7]:

$$\bar{x}y\bar{z}t + xy\bar{z}t = y\bar{z}t(\bar{x} + x) = y\bar{z}t$$

Oblasť so štyrmi jednotkami nad sebou[8]:

$$x\bar{y}(\overline{zt} + \bar{z}t) + x\bar{y}(zt + z\bar{t}) = x\overline{\overline{yz}}(\bar{t} + t) + x\bar{y}z(t + \bar{t}) =$$

$$= x\overline{\overline{yz}} + x\bar{y}z = x\bar{y}(\bar{z} + z) = x\bar{y}$$

Oblasť so štyrmi jednotkami v rohoch mapy:

$$\overline{xy}(\overline{zt} + z\bar{t}) + x\bar{y}(\overline{zt} + z\bar{t}) = \overline{xyt}(\bar{z} + z) + x\overline{yt}(\bar{z} + z) =$$

$$= \overline{xyt} + x\overline{yt} = \overline{yt}(\bar{x} + x) = \overline{yt}$$

Odtiaľ výsledný zjednodušený zápis funkcie

$$f = y\bar{z}t + x\bar{y} + \overline{yt} =$$

Čo možno zapísať:

$$f = y\bar{z}t + \bar{y}(x + \overline{t}) =$$

Zjednodušovať môžeme aj neúplne zadané funkcie. Vtedy nedefinované prípady dodefinujeme tak, aby sa v mape dali nájsť čo najväčšie oblasti jednotiek.

Príklad I.11: Neúplne zadanú mapu 2.3 je vhodné dodefinovať (zúplniť) na mapu znázornenú na obr. 2.4

Treba uviesť, že Karnaughova metóda je síce algoritmicky realizovateľná (dokonca jednoduchým algoritmom), no prakticky použiteľná len pre funkcie nanajvýš piatich premenných (skúste odhadnúť počet operácií algoritmu pri minimalizácii funkcie šiestich premenných). Dôvodom nie je to, že by Karnaughova metóda bola neefektívna, príčinou je samotná úloha minimalizácie– čo je problém s veľkou asymptotickou zložitosťou.

[6] napr. z množiny horeuvedených vzťahov

[7] na obrázku je okolo jednotiek tejto oblasti štvorec

[8] štyri jednotky v poslednom stĺpci

$zt \backslash xy$	**00**	**01**	**10**	**11**
00				
01	1	1	1	1
10				
11				

Obrázok 2.4: Mapa dodefinovanej funkcie

Kapitola 3

KÓDOVANIE INFORMÁCIÍ

V počítači sa akákoľvek informácia reprezentuje binárnou abecedou, t.j. v binárnom kóde. Predpokladáme, že kódovanie nečíselnej informácie je čitateľovi dozaista známe[1]. Našu pozornosť preto zameriame na kódovanie čísel binárnou abecedou[2]. V 1.kapitole sme hovorili o binárnej sústave. Je zrejmé, že celé číslo bez znamienka stačí vyjadriť v dvojkovej sústave. Pokiaľ však uvažujeme celé čísla so znamienkom alebo reálne čísla, existuje viacero spôsobov, ako ich kódovať. Dôležité pritom je, aby sa pri danom zápise čísel v zvolenom kódovaní dali ľahko uskutočňovať aritmetické operácie (napríklad, ako sme uviedli, v rímskej sústave to ide ťažko).

Najskôr sa budeme venovať spôsobom kódovania celých čísel. Uvedieme základné spôsoby kódovania a porovnáme ich. Popíšeme algoritmy pre vykonanie základných aritmetických operácií. Ďalej sa budeme venovať spôsobom kódovania reálnych čísel, pričom takisto uvedieme ich základné spôsoby kódovania a aritmetické algoritmy. Porovnáme jednotlivé typy kódov, ako príklady uvedieme niektoré existujúce normy kódov a opíšeme ťažkosti spojené s aritmetikou reálnych čísel. Na záver opíšeme aj niektoré ďalšie spôsoby kódovania čísel, ktoré nachádzajú uplatnenie v špeciálnych úlohách.

Binárne číslo budeme zapisovať v tvare $a_{n-1}, a_{n-2}, \ldots, a_0$. Jednotlivé číslice a_i nazývame *bitmi*.

3.1 Kódovanie celých čísel

Ako sme spomenuli, v prípade *celého čísla bez znamienka* je reprezentácia čísla jednoduchá. Stačí ho vyjadriť bitmi $a_{n-1}, a_{n-2}, \ldots, a_0$, pričom $a_{n-1}a_{n-2}\ldots a_0$ je binárny zápis čísla. Pomocou N bitov možno reprezentovať 2^N čísel v rozsahu $0 \ldots (2^N - 1)$.

Na zápis celých čísel so znamienkom sa najčastejšie používajú tri kódy: *priamy*, *inverzný* a *doplnkový*.

[1] v prípade nečíselnej informácie (napr. textovej, obrazovej, zvukovej) jednotlivé objekty očíslujeme (napr. v prípade textovej informácie: jednotlivým znakom konečnej abecedy, v ktorej je text písaný, priradíme čísla. Takto vieme číslom vyjadriť každý znak textu, text potom zakódujeme súborom čísel). Nečíselnú informáciu teda kódujeme pomocou čísel; resp. počítač samotný nerozlišuje inú než číselnú informáciu a až program ju 'správne' interpretuje ako nečíselnú informáciu určitého typu.

[2] t.j. pomocou dvoch symbolov (0 a 1)

- Pri *priamom* kóde je bit a_{n-1} vyhradený pre znamienko (nula značí kladné a jednotka záporné znamienko) a zvyšné bity $a_{n-2}a_{n-3}\ldots a_0$ predstavujú absolútnu hodnotu čísla. Takýto zápis pripúšťa dve reprezentácie nuly, ako kladnú nulu, alebo ako zápornú nulu. Napríklad, pri 4-bitovom zápise 1000 a 0000 predstavujú -0 a $+0$. Počet reprezentovateľných hodnôt je $2^N - 1$, rozsah je $-(2^{N-1} - 1)\ldots 2^{N-1} - 1$.

  Niektoré aritmetické operácie sa v priamom kóde realizujú pomerne zložito, navyše je kód redundantný (dvojitá reprezentácia nuly). Preto sa zaviedli aj iné kódy, z ktorých najvýznamnejšie sú *inverzný* a *doplnkový kód*. V týchto kódoch je kódovanie *kladných* celých čísel zhodné s kódovaním v priamom kóde (čo znamená, že z N bitov je najvyšší rovný nule a zvyšných $N-1$ vyjadruje absolútnu hodnotu čísla). Odlišné je však kódovanie *záporných* čísel.

- V *inverznom kóde*[3] k číslu a získame číslo $-a$ odčítaním čísla a od čísla $2^N - 1$:

$$-a = (2^N - 1) - a$$

  Ekvivalentným postupom získania čísla $-a$ je, že sa negujú všetky bity čísla a. Možno teda povedať, že inverzný kód je kód, v ktorom sa kladné čísla kódujú priamo (v binárnom tvare) a záporné čísla sa získajú tak, že negujeme všetky bity absolútnej hodnoty čísla.

  Podobne ako v priamom kóde sa najvyšší bit prejavuje ako znamienkový a ostatné bity vyjadrujú samotné číslo.

  Opäť, nula má dve reprezentácie (0000 a 1111). Počet reprezentovateľných hodnôt je takisto len $2^N - 1$ v rozsahu $-(2^{N-1} - 1)\ldots(2^{N-1} - 1)$.

  Operácie v inverznom kóde sa taktiež nevykonávajú bez ťažkostí. Napríklad, pri sčítavaní nie je výsledok vždy priamo v inverznom kóde, niekedy (napr. pri sčítaní čísel s rôznymi znamienkami) je potrebné k výsledku pripočítať 1.

  Príklad I.12: V inverznom kóde platí, že $3_{10} = 0011$ a $-3_{10} = 1100$. Skúsme tieto čísla spočítať:

$$\begin{array}{rll} & 3_{10} & 0011 \\ - & 3_{10} & 1100 \\ & & \overline{1111} \quad = 0 \end{array}$$

  Sčítajme 3 a -2:

$$\begin{array}{rll} & 3_{10} & 0011 \\ - & 2_{10} & 1101 \\ & & \overline{0000} \quad = 0 \end{array}$$

[3] známom aj ako *jednotkový doplnkový kód*

Pretože pracujeme so 4 bitovými číslami, za výsledok považujeme číslo 0000. Tento výsledok však nie je správny, musíme k nemu ešte prirátať 1, aby sme dostali správny výsledok (0001).

Iný príklad:

$$\begin{array}{ll} -\,3_{10} & 1100 \\ -\,3_{10} & 1100 \\ & \overline{11000} \end{array}$$

Opäť, k výsledku 1000 musíme prirátať 1, aby sme dostali správny výsledok 1001 (t.j. -6).

- V *binárnom doplnkovom kóde* opačné číslo získame jeho odčítaním od 2^N. Napr. pri štvorbitovej reprezentácii :

$$\begin{array}{ll} 3_{10} & 0011 \\ & \\ 2^4 & 10000 \\ & -0011 \\ -\,3_{10} & \overline{1101} \end{array}$$

Ekvivalentný postup získania opačného číslo $-a$ k číslu a je negovať všetky jeho bity a k výsledku pripočítať 1:

$$-a = \neg a + 1$$

Počet reprezentovateľných hodnôt je 2^N v rozsahu $-2^{N-1} \ldots (2^{N-1} - 1)$. Všimnime si, že nulu už nevyjadrujeme dvoma rôznymi spôsobmi. Preto v tomto kóde môže N-bitový vektor nadobúdať hodnotu z množiny, ktorá má až 2^N rôznych hodnôt.

Tento kód má spomedzi doteraz uvedených najvyššiu efektivitu. Ako uvidíme, ľahko sa v ňom realizuje sčítanie a odčítanie, pričom výsledok je vždy v doplnkovom kóde. Násobenie a delenie nie je oveľa zložitejšie ako v doterajších kódoch.

Je to tiež pozičný systém - hodnota reprezentovaného čísla sa dá vyjadriť ako

$$-b_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} (b_i \cdot 2^i) + 1,$$

pričom b_i je rovné 0 alebo 1.

Cvičenie I.9: Dokážte uvedenú rovnosť.

Prvý bit sa prejavuje ako znamienkový, pretože ak b_{N-1} (bit, ktorý má najväčšiu váhu) sa rovná 1, potom je číslo záporné, inak je kladné.

$$
\begin{array}{llll}
\ldots z_2 & z_1 & z_0 & \\
\ldots a_3 & a_2 & a_1 & a_0 \\
\underline{\ldots b_3} & \underline{b_2} & \underline{b_1} & \underline{b_0} \\
\ldots \mathbf{c}_3 & \mathbf{c}_2 & \mathbf{c}_1 & \mathbf{c}_0 \\
\ldots z_3 & z_2 & z_1 & z_0
\end{array}
\qquad \text{(smer je od nižších bitov k vyšším)}
$$

Obrázok 3.1: Sčítanie v binárnom kóde

3.2 Binárna aritmetika

Všetky základné operácie sa v dvojkovej sústave realizujú analogicky ako v ('klasickej') desiatkovej aritmetike. Navyše, pretože binárna sústava má len dve číslice, aritmetické algoritmy sa zjednodušia.

Navrhneme príslušné algoritmy; okrem aritmetiky neznamienkových celých čísel (v dvojkovej sústave) aj pre aritmetiku v doplnkovom kóde.

sčítanie

Pri sčítaní v binárnej sústave sa uplatňuje analogický postup ako pri sčítaní v desiatkovej sústave – viď obr. 3.1 (sčítavame postupne od najnižších rádov, pričom rátame aj s prípadným prenosom do vyššieho rádu).

Postup platí nielen pre binárny kód, ale aj pre doplnkový kód. To, že čísla sú v doplnkovom kóde zaručuje, že uvedený algoritmus pre sčítanie čísel v binárnej sústave možno použiť aj na sčítanie čísel v doplnkovom kóde (zamyslite sa, prečo). Výsledok, ktorý dostaneme bude korektný – pokiaľ nedošlo k pretečeniu. Pretečenie sa však indikuje odlišným spôsobom ako pri sčitovaní v binárnom kóde: k pretečeniu došlo, ak je výsledok záporný, pričom sme sčitovali dve kladné čísla; alebo je výsledok kladný a pritom sme sčitovali dve záporné čísla.

Z toho, pre i-tu číslicu (c_i) výsledku C platí:

$$c_i = a_i + b_i + z_{i-1},$$

kde z_{i-1} je *prenos* z predchádzajúceho rádu.

odčítanie

V prípade binárneho kódu používame rovnaký algoritmus ako pre odčítanie dvoch desiatkových čísel.

Nech sú dané dve čísla A a B v doplnkovom kóde. Rozdiel $A - B$ získame sčítaním čísel A a $(-B)$. Znamienko čísla B zmeníme tak, že najskôr negujeme všetky bity čísla B (aj znamienkový) a pripočítame k nemu 1[4].

[4] korektnosť tohto postupu vyplýva priamo z definície doplnkového kódu (viď predch. kapitolu)

násobenie

'Ručné' násobenie čísel $A = a_3a_2a_1a_0$, $B = b_3b_2b_1b_0$ vyzerá takto:

			$\mathbf{a_3}$	$\mathbf{a_2}$	$\mathbf{a_1}$	$\mathbf{a_0}$	
		$\times$	$\mathbf{b_3}$	$\mathbf{b_2}$	$\mathbf{b_1}$	$\mathbf{b_0}$	
			a_3b_0	a_2b_0	a_1b_0	a_0b_0	(b_0A)
		a_3b_1	a_2b_1	a_1b_1	a_0b_1		(b_1A)
	a_3b_2	a_2b_2	a_1b_2	a_0b_2			(b_2A)
a_3b_3	a_2b_3	a_1b_3	a_0b_3				(b_3A)
$\mathbf{c_6}$	$\mathbf{c_5}$	$\mathbf{c_4}$	$\mathbf{c_3}$	$\mathbf{c_2}$	$\mathbf{c_1}$	$\mathbf{c_0}$	

V binárnej sústave sa teda súčin $A \times B$ dá vyjadriť ako:

$$A \cdot B = A \cdot b_3b_2b_1b_0 = A \cdot b_3 \cdot 2^3 + A \cdot b^2 \cdot 2^2 + A \cdot b_1 \cdot 2^1 + A \cdot b_0 \cdot 2^0$$

Výhodné je, že cifry $b_0, \ldots, b_3$ sú z množiny $\{0, 1\}$. Teda ak $b_i = 0$, tak čiastkový súčet $b_i \cdot A \cdot 2^i = 0$, ak $b_i = 1$, tak $b_i \cdot A \cdot 2^i = A \cdot 2^i$. Násobenie sa nám zjednoduší- stačí nám previesť postup uvedený v nasledovnom algoritme násobenia v binárnej sústave:

1. vezmeme poslednú cifru čísla B
2. ak je to 1, tak k celkovému výsledku pripočítame A
3. posunieme A doľava (vynásobíme A dvomi)
4. posunieme B doprava
5. ak $B <> 0$, tak prejdeme k bodu 2, inak skončíme

Koľko je maximálny počet elementárnych krokov (operácií)[5] algoritmu? Ak A aj B sú N-bitové čísla, cyklus sa opakuje nanajvýš N krát. To značí, že algoritmus obsahuje rádovo N krokov.

Cvičenie I.10: Dokážte formálnejšiu formuláciu: Časová zložitosť algoritmu násobenia (na vstupoch A,B) je $O(\log A + \log B)$.

Cvičenie I.11: Ako je to s násobením čísel v doplnkovom kóde?

delenie

Opäť, podiel binárnych čísel sa dá vypočítať pomocou 'štandardného' algoritmu delenia:[6]

Označme číslom A delenca a číslom B deliteľa. Predpokladajme, že $B \neq 0$. Algoritmus delenia je nasledovný:

1. zapíšme číslo B tak, aby číslica s najvyšším rádom B bola pod číslicou s najvyšším rádom čísla A; t.j. číslo B vynásobíme číslom 2^k pre k=(Počet cifier A)–(Počet cifier B). Označme súčin $B \cdot 2^k$ ako C. Použijeme tiež premennú i, do ktorej priradíme hodnotu k

[5] posunov čísla, sčítaniu dvoch čísel, testov bitu alebo čísla na nulu

[6] neuvedieme ho vo všeobecnom tvare, použiteľnom pre akúkoľvek sústavu, ale kvôli jednoduchosti zápisu použijeme jeho prepis pre binárnu sústavu

2. ak je $B \leq A$, i-ty bit výsledku bude 0
3. inak od čísla A odrátame číslo C, i-ty bit výsledku bude 1
4. vydelíme číslo C dvoma (t.j. posuňme C doprava a znížime i o jedna
5. opakujeme od bodu 2, pokiaľ $i \neq 0$

   opakovaním pre hodnoty $i < 0$ dostaneme ako výsledok reálne číslo

V nasledujúcich kapitolách opíšeme ďalšie metódy kódovania celých čísel (napríklad excess kód a BCD kód) i čísel reálnych. Našu pozornosť zameriame aj na aspekty reprezentácie čísel v počítači – napr. akú množinu čísel je v jednotlivých kódoch možné reprezentovať pomocou N bitov, ako možno realizovať aritmetické operácie a iné.

Kapitola 4

Reálne čísla a reálna aritmetika

4.1 Kódovanie reálnych čísel

Ako možno zapísať reálne číslo? Bežne sa stretávame s dvoma spôsobmi. Prvý spôsob zápisu je napísať celú časť čísla, potom desatinnú čiarku[1] a nakoniec desatinnú časť, napríklad 3.1415928. Táto metóda zápisu je však neprehľadná, pokiaľ sa snažíme zapísať veľmi malé alebo veľmi veľké číslo, napr. 'päťsto miliárd'. Vhodnejším môže byť použitie druhého ('vedeckého') spôsobu zápisu: $5 \cdot 10^{11}$.

Reprezentácia reálnych čísel v počítači je založená na rovnakých myšlienkach ako spomenuté metódy. Obmedzujúcim faktorom však je, že v počítači môžeme uchovať len čísla určitého (konečného) rozsahu.

Podľa toho, o ktorú metódu reprezentácie reálnych čísel sa jedná rozlišujeme:

- formát s pevnou rádovou čiarkou
- formát s pohyblivou rádovou čiarkou

- Pri kódovaní v *pevnej rádovej čiarke* je pevne určené, koľko bitov zaberá celá časť a koľko desatinná. Rádová čiarka má teda pevne určenú pozíciu.

  Ak označíme pozíciu rádovej čiarky zprava p, (t.j. za rádovou čiarkou nasleduje p bitov, napr. pre celé čísla je $p = 0$), potom hodnota čísla reprezentovaného v doplnkovom kóde je:

$$-b_{N-1} \cdot 2^{N-p-1} + \sum_{i=1}^{N-1} b_i \cdot 2^{i/p}$$

  Vidíme, že aj toto kódovanie je pozičné. Sčítanie a odčítanie reálnych čísel v pevnej rádovej čiarke sa realizuje rovnako ako pri celých číslach. To isté platí aj pre násobenie a delenie. (Prečo?)

  Môže sa však stať, že výsledok nejakej operácie s číslami je číslo mimo zobraziteľného rozsahu, a preto treba upravovať výsledok. Napríklad, už pri sčítaní a odčítaní celých čísel môže výsledok presahovať zobraziteľný rozsah o jednu číslicu a pri

[1] resp. rádovú čiarku

násobení dvoch N bitových celých čísel až o N číslic. Pri desatinných číslach taktiež môže nastať podobná situácia (napríklad odčítaním dvoch veľmi malých čísel). Dostaneme výsledok popísateľný len väčším počtom bitov ako má daný kód. No v prípade, že 'nadbytočná' informácia sú miesta za desatinnou čiarkou, môžeme výsledok zobraziť do nášho kódu - t.j. nahradiť presnejšie číslo menej presným - či už *odseknutím*, alebo *zaokrúhlením*.

Cvičenie I.12: Napíšte kompletné algoritmy pre spomenuté operácie.

Podľa toho, aké má byť najväčšie zobraziteľné číslo a najmenšia rozlíšiteľná hodnota sa skonštruuje príslušný kód (čiže určí sa n a p). Najmenšia rozlíšiteľná hodnota je najmenší rozdiel medzi dvomi číslami v tomto systéme a budeme ju označovať Δr. Určená je parametrom p - platí, že r je rovné 2^{-p}.

- Pri zápise čísel v *pohyblivej rádovej čiarke* predstavuje počítačovú analógiu spomenutého 'vedeckého' spôsobu zápisu reálnych čísel.

  Pri kódovaní celých čísel (bez znamienka) sme pomocou N bitov mohli vyjadriť čísla z rozsahu $0 \ldots 2^N - 1$, pričom Δr bolo rovné 1. Kódovania celých čísel so znamienkom znížili dolnú aj hornú hranicu intervalov (napr.$-(2^{N-1}-1) \ldots$ $(2^N - 1)$), kódovanie s pevnou rádovou čiarkou znížilo Δr.

  Kódovanie s pohyblivou desatinnou čiarkou umožňuje zapísať pomocou N bitov aj čísla väčšie ako 2^N či menšie ako 2^{-N}. Používame ho vlastne aj v bežnom živote - stačí si všimnúť zápisy: $6.022 * 10^{23}$ či $4.85 * 10^{-54}$. Nemeníme počet reprezentovateľných hodnôt (tých môže nanajvýš 2^N), len spôsob reprezentácie.

  V tomto systéme kódovania (*FPNS - Floating Point Number System*) sa číslo zapisuje v tvare $M * z^E$, kde M je *mantisa*, z je *zaklad* a e je *exponent*. Príslušný kód určujú nasledovné údaje:

  - sústava, v ktorej kódujeme základ (označíme r_b) ('tradične' 10, v počítači preferujeme binárnu sústavu, teda $r_b = 2$)
  - počet cifier použiteľných na reprezentáciu mantisy (m)
  - spôsob kódovania (znamienka) mantisy
  - sústavu, v ktorej je kódovaný exponent r_e (opäť, 'tradične' 10 a pre počítače väčšinou rovné 2)
  - počet cifier použiteľných na reprezentáciu exponentu (e)

Kódové slovo (t.j zápis čísla) obsahuje nasledovné informácie: hodnoty znamienka, exponentu a mantisy. Väčšinou sú uložené tak, že znamienko je najvýznamnejším bitom, po ňom nasleduje exponent a za ním mantisa. Informácie spoločné pre celý kód sa neuchovávajú, ako napríklad pozícia rádovej čiarky (t.j. hodnota p).

Mantisu i exponent reprezentujeme akoukoľvek metódou, ktorá povoľuje zobrazenie kladných a záporných čísel. Často sa na to používa excess kód.

Pri konštrukcií kódu treba určiť aj pozíciu desatinnej čiarky v mantise. Pochopiteľne, nemusela by byť pevne dohodnutá, no potom by každé číslo muselo obsahovať aj informáciu o tom, na ktorej pozícii z m bitov sa čiarka nachádza, čo by vyžadovalo prídavnú informáciu (veľkosti $\log m$ bitov). Preto sa prednosť dáva pevnej pozícii čiarky.

Spomenuté údaje definujú rozsah reprezentovateľných čísel (i Δr). Preto tieto parametre kódu určíme na základe želanej množiny reprezentovateľných čísel.

Dohodneme sa, že rovnako ako už pri racionálnych číslach budeme počet číslic v mantise za desatinnou čiarkou označovať symbolom p. Všimnime si, že to isté číslo môžeme vyjadriť viacerými spôsobmi, napr. $3{,}0.3 * 10^1$,$300 * 10^{-2}$ a tak ďalej...

Preto sa používa tzv. normalizovaná binárna mantisa, čo je mantisa, ktorej prvá cifra je nula (resp. prvý bit je 1), za ňou je umiestnená desatinná bodka, za ktorou nasleduje nenulová číslica. Mantisa má teda tvar 0.xz, kde x je nenulová číslica a z je číslo. Proces úpravy nenormalizovaného čísla na normalizované nazývame normalizácia.

Pretože každé normalizované číslo má na prvom mieste (za desatinnou bodkou) jednotku, nie je dôvod ju ukladať, čím ušetríme jeden bit a zdvojnásobíme priestor ukladateľných čísel. Tento spôsob kódovania nazývame technika skrytého bitu (hidden bit technique).

Otázkou ale je, ako v tomto prípade kódovať nulu. Nulu kódujeme ako číslo s najmenšou absolútnou hodnotou zobraziteľné v danom kóde, t.j. aproximujeme ho s presnosťou Δr.

Uvedieme teraz niekoľko príkladov rôznych kódovaní v pohyblivej čiarke.

Dohodneme sa, že hodnotu mantisy označíme symbolom H_M, jej najmenšiu možnú hodnotu H_{Mmin} a najväčšiu možnú H_{Mmax}. Čísla V_{FPNmin} a V_{FPNmax} udávajú najmenšie, resp. najväčšie číslo zobraziteľné v danom kóde. Pre porovnanie sú uvedené aj čísla NLM_{FPN} a NRV_{FPN} - prvé z nich udáva aké najväčšie binárne číslo vieme zobraziť pomocou m bitov, druhé pomocou $m + e$ bitov.

Jedným z používaných spôsobov zápisu je DEC 32-bitový normalizovaný formát s pohyblivou rádovou čiarkou. V tomto systéme je $r_b = 2$, $r_e = 2$, $m = 24$ so skrytým bitom, $e = 8$ exponent sa ukladá v excess 128 kóde a mantisu považujeme za kladnú. Potom:

$$
\begin{aligned}
H_{Mmin} &= 0.1000\ldots_2 = 1/2 \\
H_{Mmax} &= 0.1111\ldots 2 = 0.999999940395 = 1.0 - 2^{-24} \\
V_{FPNmin} &= 0.1000\ldots 2 * 2^{-127} = 2.9387 * 10^{-39} \\
V_{FPNmax} &= 0.1111\ldots_2 * 2^{+127} = 1.7014 * 10^{38} \\
NLM_{FPN} &= 2^{23} = 8,388,608 \\
NRV_{FPN} &= 2^{23} * (2^8 - 1) = 2.139 * 10^9
\end{aligned}
$$

Okrem uvedeného FPNS sa používajú aj ďalšie systémy pre 32-bitové aj pre 64-bitové formáty.

Príkladom je IBM 32-bitový normalizovaný formát s pohyblivou desatinnou čiarkou. V ňom $r_b = 16$, $r_e = 2$, $m = 6$ so skrytým bitom, $e = 7$ exponent sa ukladá v exces-64 kóde a mantisu považujeme za kladnú.

$$
\begin{aligned}
H_{Mmin} &= 0.1000\ldots 16 = 1/16 \\
H_{Mmax} &= 0.FFFF\ldots\ldots_{16} = 0.999999940395 = 1.0 - 16^{-6} \\
V_{FPNmin} &= 0.1000\ldots 16 * 16^{-63} = 8.636 * 10^{-78} \\
V_{FPNmax} &= 0.FFFF\ldots 16 * 16^{+63} = 7.237 * 10^{75} \\
NLM_{FPN} &= 15 * 16^5 = 15,728,640
\end{aligned}
$$

$$NRV_{FPN} = 15 * 16^5 * (2^7 - 1) = 1.9975 * 10^9$$

Ďalšími príkladmi FPNS sú IEEE 32-bitový normalizovaný formát s pohyblivou desatinnou čiarkou: $r_b = 2$, $r_e = 2$, $m = 24$ so skrytým bitom, ale $p = 23$, $e = 8$ exponent sa ukladá v excess-127 kóde a mantisu považujeme za kladnú

$$\begin{aligned} H_{Mmin} &= 1.000\ldots_2 = 1 \\ H_{Mmax} &= 1.111\ldots_2 = 1.99999988 = 2.0 - 2^{-23} \\ V_{FPNmin} &= 1.000\ldots_2 * 2^{-126} = 1.1755 * 10^{-38} \\ V_{FPNmax} &= 1.111\ldots_2 * 2^{+128} = 3.4028 * 10^{38} \\ NLM_{FPN} &= 2^{23} = 8,388,608 \\ NRV_{FPN} &= 2^{23} * (2^8 - 2) = 2.131 * 10^9 \end{aligned}$$

a IEEE 64-bitový formát s pohyblivou desatinnou čiarkou: $r_b = 2$, $r_e = 2$, $m = 53$, $p = 52$, $e = 11$ a exponent sa ukladá v excess 1023 formáte.

$$\begin{aligned} H_{Mmin} &= 1.000\ldots_2 = 1 \\ H_{Mmax} &= 1.111\ldots_2 = 2.0 - 2^{-52} \\ V_{FPNmin} &= 1.000\ldots_2 * 2^{-1022} = 2.225 * 10^{-308} \\ V_{FPNmax} &= 1.111\ldots_2 * 2^{+1023} = 1.798 * 10^{308} \\ NLM_{FPN} &= 2^{52} = 4.51015 \\ NRV_{FPN} &= 2^{52} * (2^{11} - 2) = 9.214 * 10^{18} \end{aligned}$$

Napokon spomenieme jeden systém používaný pri vedeckých výpočtoch (používaný napr. na superpočítači Cray)– je to 64-bitový formát s pohyblivou rádovou čiarkou. Preň $r_b = 2$, $r_e = 2$, $m = 48$, $p = 48$, $e = 15$, mantisu považujeme za kladnú a exponent sa ukladá v excess 16384 formáte. Pri takom veľkom exponente Cray nepoužíva celý rozsah, ale krajné hodnoty znamenajú pretečenie a 'podtečenie'.

$$\begin{aligned} H_{Emin} &= -8,192 \\ H_{Emax} &= 8,191 \\ VFPN_{min} &= 0.1000\ldots_2 * 2^{-8192} = 4.584 * 10^{-2467} \\ VFPN_{max} &= 0.1111\ldots_2 * 2^{+8191} = 5.4537 * 10^{2465} \\ NLM_{FPN} &= 2^{48} = 2.815 * 10^{2465} \\ NRV_{FPN} &= 2^{48} * (2^{14} - 1) = 4.6114 * 10^{18} \end{aligned}$$

Tento systém má veľmi veľký rozsah a je schopný reprezentovať značne veľké aj značne malé čísla.

4.2 Aritmetické operácie

sčítanie a odčítanie

Nech sú dané čísla A, B reprezentované v pohyblivej rádovej čiarke s mantisami M_a, M_b a exponentami E_a, E_b.

- Ak $E_a = E_b$, potom stačí sčítať mantisy a výsledok upraviť na normalizovaný tvar.
- Vo väčšine prípadov sú však exponenty rôzne. Vtedy musíme čísla upraviť na tvar s rovnakým exponentom:

  Ak $E_a > E_b$, potom možno súčet $a + b$ vyrátať ako:

$$a + b = M_a * 2^{E_a} + M_b * 2^{E_b} = (M_a * 2^{E_a - E_b} + M_b) * 2^{E_b}$$

To znamená, že mantisu M_A musíme posunúť o $E_B - E_A$ pozícií doprava, čím docielime, že na rovnakých pozíciách sú cifry s rovnakou váhou. Takto upravené mantisy už môžeme sčítať. Exponent výsledku je E_B, teda exponent väčšieho čísla.

Výsledok však nemusí byť v normalizovanom tvare, napr.:

0.1101	0.1011
+ 0.1110	− 0.1001
$\overline{1.1011}$ - je potrebný posun vpravo o 1 pozíciu	$\overline{0.0010}$ - je potrebný posun vľavo o 2 pozície

V takomto prípade je potrebné vhodne posunúť mantisu a upraviť exponent. Môže dôjsť k pretečeniu či podtečeniu, čo treba detekovať a ošetriť.

násobenie

Pred vykonaním násobenia v pohyblivej rádovej čiarke nie je potrebné upraviť zápis čísel na jednotný tvar. Platí vzťah

$$A * B = M_a \cdot 2^{E_a} * M_b \cdot 2E_b = (M_a * M_b) \cdot 2^{E_a + E_b}$$

Teda stačí vynásobiť mantisy a sčítať exponenty.

Podobne ako pri sčítavaní je niekedy potrebné výsledok normalizovať. Ako vidieť z nasledujúceho príkladu, v najhoršom prípade je potrebný posun o jednu pozíciu.

Príklad I.13:

- násobenie 'maximálnych' mantís:

  0.1111
  × 0.1111
  $\overline{0.1110}$ - nie je potrebná normalizácia

- násobenie 'minimálnych' mantís:

  0.1000
  × 0.1000
  $\overline{0.0100}$ - je potrebný posun o 1 pozíciu vľavo

delenie

Delenie čísel zapísaných v pohyblivej rádovej čiarke sa realizuje podobne ako násobenie. Platí vzťah:

$$A/B = (M_a * 2^{E_a})/(M_b B * 2^{E_b}) = (M_a/M_b).2^{E_a - E_b}$$

Pri normalizácii je v najhoršom prípade potrebný posun o 1 pozíciu vpravo.

Príklad I.14:

- maximálna mantisa / minimálna

  $0.1111/0.1000 = 1.1110$ - je potrebný posun o 1 pozíciu vpravo

- minimálna mantisa / maximálna

  $0.1000/0.1111 = 0.1000$ - normalizácia nie je potrebná

4.3 Realizácia matematických funkcií

Pomocou základných aritmetických operácií možno vyrátať aj zložitejšie matematické funkcie. Najjednoduchšie je to možné pomocou Taylorovych radov:

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

$$\cos x = -1 + \frac{x^2}{2!} - \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots = \sum_{n=0} -1^{n+1} \frac{x^{2n}}{(2n)!}$$

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots = \sum_{n=1} -1^{n+1} \frac{x^{2n-1}}{(2n-1)}$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{n=0} \frac{x^n}{n!}$$

$$\ln(1+x) = \sum n = 1 - 1^{n+1} \frac{x^n}{n}; x \in (-1, 1)$$

$$\frac{1}{1-x} = \sum_{n=0} x^n; x \in (-1, 1)$$

$$n! = \sqrt{2\pi n} \frac{n}{e}^n (1 + \frac{1}{12n} + \frac{1}{288n^2} - \frac{139}{51840n^3} + O(\frac{1}{n^4}))$$

Pre výpočet týchto funkcií existuje aj množstvo numerických algoritmov, ktoré sú efektívnejšie ako priamočiare použitie Taylorovych radov (sčítanie prvých k členov radu). Čitateľ ich môže nájsť v literatúre z oblasti numerickej matematiky.

4.4 Nepresnosti pri výpočtoch

Ako sme už spomenuli, pri sčítaní dvoch N-bitových mantís sa môže stať, že výsledná mantisa bude mať viac ako N bitov. Napríklad:

$$\begin{array}{r} 101010 \\ +110010 \\ \hline 11011010 \end{array}$$

Vo výsledku máme o dva bity viac ako môžeme zaznamenať a otázkou je, čo s nimi.

Tento problém môžeme formulovať aj ako problém reprezentácie reálnych čísel na počítači. Akýmkoľvek kódom nedokážeme vyjadriť každé reálne číslo– nevieme, ak to nie je racionálne číslo vyjadriteľné v tvare $m \cdot z^e$ pre 'povolené' m, e a dané z. Jediným riešením je aproximovať ho nejakým iným číslom s kratšou mantisou, ktorú už dokážeme reprezentovať. S touto aproximáciou výsledku sa ďalej môže vykonávať množstvo aritmetických operácií; s každou z nich sa celková chyba ďalej zväčšuje. Preto je potrebné rozhodnúť sa pre čo najlepšiu aproximáciu; v závislosti od zanedbávanej časti mantisy a od ďalej vykonávaných operáciách.

Označme presnú hodnotu x (t.j. reálne číslo), jej aproximáciu $\tilde{x}$. *Absolútna chyba* aproximácie $\tilde{x}$ je rozdiel $x - \tilde{x}$. *Relativna chyba* aproximácie $\tilde{x}$ je podiel $\frac{(x-\tilde{x}}{\tilde{x}}, x \neq 0$. Pre rôzne numerické algoritmy (napr. pre rôzne zaokrúhľovania či aritmetické operácie) odhadujeme absolútnu a relatívnu chybu. Snažíme sa dosiahnuť čo najtesnejší horný odhad.

V ďalšom texte budeme študovať rôzne spôsoby zaokrúhľovania. Najjednoduchšia technika je proste nadbytočné bity ignorovať, vypustiť - *truncation* (*usekávanie*); v našom príklade z 1.1011010 odseknutím posledných dvoch bitov dostaneme 1.10110. V prípade aproximácie kladných čísel je aproximácia vždy menšia-nanajvýš rovná ako pôvodné reálne číslo; absolútna chyba je teda vždy kladná. Nech x je číslo a α je jeho aproximácia. Nech x nepatrí do $M(q,t)$, potom

$$x - \alpha = \ sgn\ x(\sum_{k=1}^{t} x_k q^{-k} + \sum_{k=t+1}^{\infty} x_k q^{-k})q^b -$$

$$- \ sgn\ x(\sum_{k=1}^{t} x_k q^{-k})q^b = \ sgn\ x(\sum_{k=t+1}^{\infty} x_k q^{-k})q^b =$$

Odhadnime zhora súčet tohto nekonečného radu- položíme všetky x_k rovné $q-1$:

$$|x - \alpha| \leq |\ sgn\ x(\sum_{k=t+1}^{\infty} q^{-k})q^b| = q^{-t} \cdot q^b = q^{b-t}$$

Na základe toho už ľahko vyrátame odhad relatívnej chyby, čo je q^{-t}.

Ďalším prístupom je *zaokrúhľovanie*. Ním zmenšíme sumu odchýlok[2]. Postup je jednoduchý a čitateľovi známy- vezmeme číslo vzniknuté useknutím a pridáme k nemu

[2]suma odchýlok (pre k-bitové mantisy) sa vytvorí tak, že sa vezmú všetky možné $k+1$-bitové mantisy, zaokrúhlia sa, vyráta sa absolútna hodnota rozdielu výsledku zaokruhlenia (aproximácie) a pôvodného čísla pre všetky uvažované čísla a potom sa všetky abslútne hodnoty rozdielov sčítajú

jednotku, ak prvá číslica odsekávanej časti je $\leq q/2$. V desiatkovej sústave (pri 'ručnom' zaokruhľovaní) to znamenalo, že číslica je aspoň 5, v dvojkovej sústave musí byť 1. Po trochu zložitejšom výpočte dostaneme, že

$$|x - \alpha| = \frac{1}{2} q^{b-t}$$

a

$$\frac{|x - \alpha|}{|\alpha|} \leq \frac{q^{b-t}}{2|\alpha|} \leq \frac{q^{b-t}}{2|q^b|} \leq \frac{1}{2} q^{-t}$$

Odchýlky budú teraz aj záporné, no suma absolútnych hodnôt odchýlok je menšia ako pri usekávaní.

Jednou z metód, ako minimalizovať chybovosť vo výpočtoch je vytvoriť zaokrúhľovaciu schému(tabuľku), ktorej suma odchýlok je rovná nule. Metóda sa volá *zaokrúhľovanie k nule*. Jedna taká tabuľka je uvedená nižšie. Iba dve hodnoty sú zaokrúhľované inak, ako pri bežnom zaokrúhľovaní, celková suma je však 0. Pri množstve výpočtov bude chybovosť takmer nula.

zaokrúhľované číslo	výsledok	chyba
xx0.00	xx0	0.00
xx0.01	xx0	+0.01
xx0.10	xx1	- 0.10
xx0.11	xx1	- 0.01
xx1.00	xx1	0.00
xx1.01	xx1	+0.01
xx1.10	xx1	+0.10
xx1.11	xx0	- 0.01

Inou metódou je *jamming*. Navrhol ju von Neumann a je veľmi jednoduchá - ako posledný bit čísla napíšeme za každých okolností jedničku. Odchýlky pri tejto technike sú väčšie ako pri predchádzajúcich, ale pri veľkom počte výpočtov je celková chybovosť menšia než pri usekávaní, hoci je rovnako rýchla.

Uviedli sme niekoľko metód aproximovania, a niektoré sme aj analyzovali. Podobne možno analyzovať aj algoritmy pre základné aritmetické operácie, výpočty funkcií či ďalšie numerické algoritmy. Tieto analýzy však prekračujú rozsah tejto práce, a v prípade potreby ich možno nájsť v knihách z oblastí numerickej matematiky. Naším cieľom bolo skôr poukázať na možné úskalia reálnej aritmetiky. Ako si už čitateľ zaisto všimol, nemusí v nej platiť asociatívny či distributívny zákon. Takisto nie je 'jednoznačný' test na nulu či test rovnosti dvoch čísel, veď za nulu považujeme ktorékoľvek číslo menšie ako najmenšie zobraziteľné číslo v našom kóde a podobne, čísla považujeme za rovnaké, ak absolútna hodnota ich rozdielu je menšia ako najmenšie zobraziteľné číslo. Z toho vyplýva, že aj matematicky ekvivalentné algoritmy nemusia dávať rovnaké výsledky. Takisto dva algoritmy nemusia mať ani rovnakú absolútnu či relatívnu odchýlku výsledku. No môže sa stať, že 'menej presný' algoritmus je rýchlejší a 'presnejší' algorimus pomalší; a preto je nutné vybrať si podľa typu úlohy a z toho vyplývajúcich priorít. Navyše, vykonávaním viacerých operácií, či dokonca postupným spúšťaním viacerých algoritmov za sebou, keď

vstupom algoritmu(operácie) je výstup predchádzajúceho algoritmu(operácie) sa chyba stále zväčšuje. Riešenie, ktoré sa v takom prípade používa je, že sa ráta v tzv. *rozšírenej presnosti* - namiesto s N-bitovými číslami (vstup bol N bitový) sa operácie vykonávajú na 2N-bitových číslach a na konci sa z 2N bitového výsledku vykonania operácií ako výsledok procedúry berie (horných) N bitov.

Kapitola 5

Iné spôsoby kódovania čísel

5.1 BCD kód

V niektorých aplikáciach je potrebné veľmi často konvertovať čísla z desiatkovej sústavy do dvojkovej a späť (napríklad keď aplikácia často vypisuje čísla v desiatkovom zápise). V takom prípade môže byť vhodné kódovať čísla odlišne - tzv. *BCD* kódom (*Binary Coded Decimaly*).

Tento kód kóduje každú cifru desiatkovej sústavy pomocou jej dvojkového ekvivalentu vyjadreného štyrmi bitmi. Pomocou štyroch bitov možno totiž vyjadriť 16 hodnôt; BCD kód z nich však využíva len prvých 10 (čísla 0000 až 1001), ostatné sú nevyužité. Ďalšia číslica v desiatkovom zápise čísla sa kóduje pomocou ďalších štyroch bitov, atď...

Príklad I.15: Číslo 729_{10} bude v BCD zakódované takto

7	2	9
0111	0010	1001

Pretože hodnoty 1010 ... 1111 ostávajú nevyužité, počet reprezentovateľných hodnôt pomocou N bitov je $10^{N/4}$ (ak N je deliteľné 4; v opačnom prípade je počet zobraziteľných čísel $10^{\lfloor N/4 \rfloor}$).

Ako realizovať aritmetické operácie? Možno síce vytvoriť algoritmy pre sčítanie a odčítanie BCD čísel, no z dôvodu jednoduchosti používajú procesory iný prístup: najskôr sa dve BCD čísla sčítajú(odčítajú) pomocou inštrukcie pre sčítanie(odčítanie) binárnych čísel bez znamienka, a potom sa prevedie *korekcia* výsledku.

Korekcia pri sčítaní dvoch dvojciferných BCD-čísel znamená že:

- ak došlo pri sčítaní k prenosu medzi tretím a štvrtým bitom, k výsledku je potrebné pripočítať 6 (prečo?)
- ak nižšie štyri bity (reprezentujúce nižšiu cifru) majú binárne hodnotu väčšiu ako 9, tak nastáva prenos do vyššieho rádu- od týchto bitov sa odráta číslo 9 a k vyšším štyrom bitom sa priráta 1
- ak vyššie štyri bity majú binárne hodnotu väčšiu ako 9, výsledok je príliš veľký- došlo k pretečeniu

Čitateľ si ľahko zovšeobecní tieto algoritmy pre korekciu po sčítaní viac ako dvojciferných BCD čísel a takisto pre korekciu po odčítaní N-ciferných BCD čísel.

Cvičenie I.13: Napíšte kompletné algoritmy pre korekcie po sčítaní a odčítaní BCD čísel.

Cvičenie I.14: Napíšte algoritmy pre prevod čísel z BCD tvaru na binárny tvar a naopak, z binárneho tvaru na BCD.

Násobenie a delenie je najjednoduchšie realizovať použitím prevodov: čísla sa prevedú z BCD formátu na binárny, vykoná sa na nich príslušná operácia a výsledok sa opäť prevedie do BCD tvaru. Samozrejme, znova je možné vytvoriť algoritmus pre priame vykonanie týchto operácií na BCD číslach. Toto riešenie si však vyžaduje prídavné obvody pre ďalšiu násobičku(deličku) a preto niektoré procesory nemajú v svojej inštrukčnej sade inštrukcie pre vykonanie spomenutých operácií.

Cvičenie I.15: Napíšte kompletné algoritmy pre násobenie a delenie BCD čísel (bez prevodu na binárne čísla).

5.2 Grayov kód

V niektorých situáciach je potrebné použiť kód, v ktorom sa zápisy (resp. slová kódu) za sebou idúcich číslel odlišujú minimálne. Prirodzene, nemusí sa jednať len o čísla, ale o ľubovoľné usporiadanie kódových slov (napr. máme kódované písmená abecedy s 'tradičným' usporiadaním). Grayov kód je potom kód, v ktorom sa kód každého kódového slova a kód jeho nasledovníka líšia najviac v jednom bite.

Nasledujúca tabuľka ukazuje ekvivalenty čísel v dvojkovom a Grayovom kóde pre prvých 16 čísel.

desiatkovo	**dvojkovo**				**Grayov kód**			
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Z tabuľky vidieť, že v Grayovom kóde sa prechod čísla na nasledujúcu hodnotu deje pomocou zmeny jediného bitu. Platí tiež, že číslo N v priamom binárnom kóde možno previesť na číslo M v Grayovom kóde pomocou vzťahu:

$$M = \frac{N \oplus 2 \cdot N}{2}$$

Uviedli sme dva spôsoby 'netradičných' kódovaní čísel: BCD kód a Grayov kód. Existujú aj rôzne ďalšie spôsoby kódovania číselnej i nečíselnej informácie; napríklad pre prenos informácie v prostredí, ktoré môže prenášanú informáciu poškodiť, sa používajú kódy odhaľujúce chybu pri prenose; resp. kódy odhaľujúce a opravujúce chyby (tzv.*samoopravné kódy*). Iným príkladom sú kódy slúžiace na *kompresiu* informácie. Ako ďalší môžno uviesť kódy *šifrujúce* informáciu. Takisto existuje aj množstvo ďalších kódov pre kódovanie čísel. Príslušný výklad prekračuje rozsah tejto práce; čitateľa odkazujeme na učebnice z oblasti teórie kódovania.

Časť II

Číslicové obvody

V časti I sme hovorili o kódovaní informácie, uviedli sme rôzne spôsoby kódovania čísel a algoritmy pre realizáciu aritmetických operácií. Pri vhodnom kódovaní čísel sme dokázali realizovať aritmetické operácie jednoducho– algoritmy sa skladali z postupnosti logických operácií. Ukázali sme aj to, že každú boolovskú funkciu možno realizovať pomocou malej množiny tzv. základných logických funkcií. V tejto časti (okrem iného) ukážeme aj technickú realizáciu základných logických funkcií.

S využitím týchto poznatkov sa môžeme venovať obvodom počítača, t.j. technických zariadení, ktoré riešia konečné úlohy.

Tematike obvodov je venovaná táto časť, v ktorej budeme hovoriť o základných obvodoch, tvoriacich základ pamätí a procesora. Napriek tomu, že obvody ktoré spomenieme nebudú zložité, s ich znalosťou si čitateľ už dokáže predstaviť realizáciu zložitejších obvodov (napr. pre reálnu aritmetiku), alebo aj samotného procesora (III.časť).

Čitateľ si môže položiť otázku, prečo je tejto problematike venovaná samostatná časť. Obvody majú riešiť konečné úlohy, a každú konečnú úlohu možno transformovať na boolovskú funkciu[1] , zapísať ju trebárs výrazom v DNF a tento výraz realizovať pomocou hradiel pre AND,OR,NOT (označovaných aj ako *základné* hradlá), ktoré už vieme technicky skonštruovať. Principiálne to možné je, avšak už riešenie relatívne jednoduchých problémov (napr. aritmetické operácie) by mohlo viesť k neefektívnym konštrukciám (koľko hradiel by mala sčítačka dvoch štvorbitových čísel?)

Musíme preto zvoliť iný prístup. Navrhneme jednoduché logické obvody a z nich budeme skladať zložitejšie systémy.

Poznámka II.1: Tento prístup vytvárania obvodov (skladaním z jednoduchších) sa nazýva *syntéza logických obvodov*.

Naše obvody budú obsahovať len základné hradlá a prípadne niektoré ďalšie jednoduché prvky. V celej časti budeme navrhovať obvody riešiace určité úlohy, pričom návrhy budú principiálne– nebudeme uvažovať fyzikálne obmedzenia (napr. že počet vetvení výstupu hradla je obmedzený), ktoré by nám schémy zbytočne zneprehľadnili.

Logické obvody môžeme rozdeliť (podľa toho, či ich súčasťou je/nie je pamäť) na:

- *kombinačné obvody* (obvody bez pamäte, ich výstup závisí len od vstupu)
- *sekvenčné obvody* (obvody s pamäťou, výstup je podmienený nielen aktuálnym vstupom, ale môžu ho ovlyvňovať aj predchádzajúce vstupy).

Obom spomenutým kategóriám sa budeme venovať v samostatných kapitolách. Na záver tejto časti sa zmienime aj o tzv. *riadiacich* obvodoch, ktoré tvoria základ procesora a podrobne ich opíšeme v III.časti.

---

[1]

- za prvé, konečná úloha priraďuje vstupu (resp. prvku zo vstupnej množiny) výstup (resp. prvok výstupnej množiny). Teda predstavuje funkciu.
- za druhé, obe množiny sú konečné; preto ich prvky možno zakódovať slovami nejakého binárneho kódu. Teda úlohu môžno transformovať aj na binárnu (boolovskú) funkciu, resp. boolovský operátor.

Kapitola 1

Kombinačné obvody

Kombinačné obvody sú obvody predstavujúce (presnejšie: realizujúce, 'rátajúce') určitú boolovskú funkciu.

Na základe aktuálneho vstupu kombinačný obvod vyráta aktuálny výstup. Nezáleží pritom, aké boli predchádzajúce vstupy, tieto nijako neovplyvnia aktuálny výpočet. Skrátka, výstup závisí len od aktuálnych vstupných hodnôt a nie je ovplyvnený predchádzajúcimi vstupmi.

Každý kombinačný obvod sa dá popísať boolovskou funkciou a pre každú boolovskú funkciu existuje kombinačný obvod, ktorý ju realizuje.

Budeme predpokladať, že čas výpočtu obvodu je nulový, t.j. okamžite po privedení hodnôt na vstup dostaneme výsledok na výstupe. Samozrejme, v skutočnosti je čas výpočtu nenulový, obvod má určité oneskorenie. Vo väčšine prípadov je však čas výpočtu tak malý, že ho môžeme považovať za nulový.

1.1 Základné kombinačné obvody

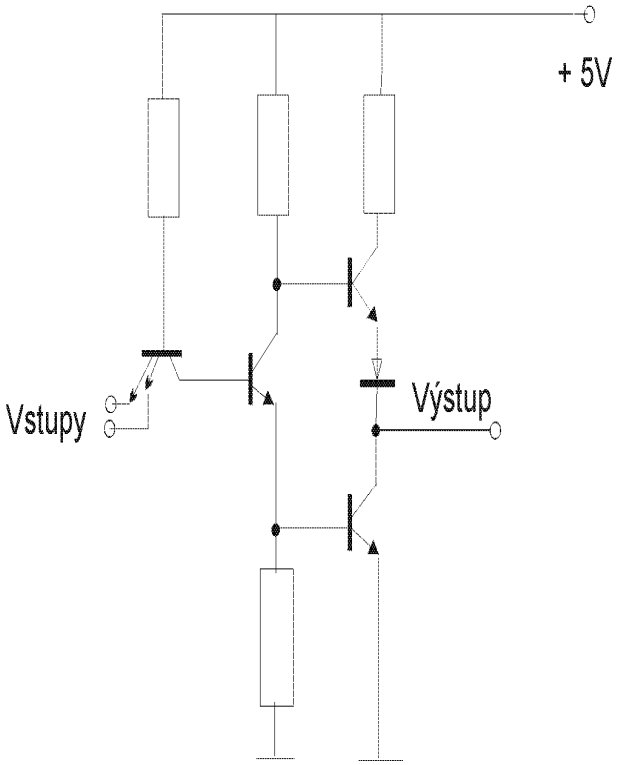
V predchádzajúcej časti sme spomenuli základné logické funkcie AND, OR, NOT, XOR, NAND a NOR, opísali sme ich vlastnosti. Obvody, ktoré tieto funkcie realizujú sa nazývajú *základné kombinačné obvody*, alebo tiež *základné hradlá*.

Možností technickej realizácie obvodov rátajúcich tieto funkcie je viacero. Uvedieme len jednu z nich (iné spôsoby realizácie možno nájsť v odbornej literatúre).

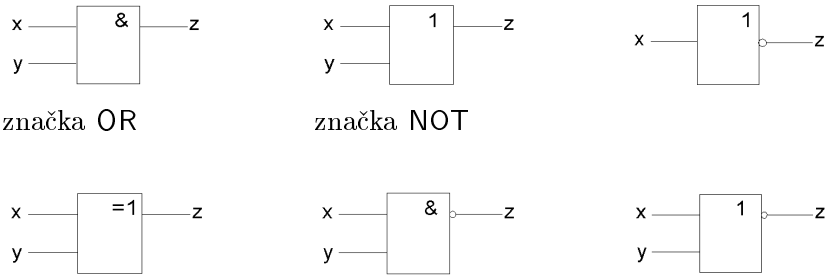
Ako sme spomenuli, binárna informácia je fyzikálne reprezentovaná dvoma úrovňami elektrického napätia. Najčastejšie sa používa tzv. *pozitívna logika* s úrovňami 0V pre symbol '0'[1] a +5V pre symbol '1'[2] (hovoríme o technológií TTL). Z praktických dôvodov je nemožné vždy dosiahnuť presne 0V alebo 5V, preto sa znaky binárnej abecedy reprezentujú *intervalom napäťových hodnôt*; napr. úrovni L zvyčajne prislúcha napätie z intervalu $\langle 0V \ldots 0.8V \rangle$ a úrovni H $\langle 2V \ldots 5V \rangle$. Na obrázku 1.1 je znázornená schéma hradla NAND. Ako už čitateľ vie, hradlá pre ostatné funkcie možno zostaviť vhodným spojením hradiel NAND.

Hradlá pre základné logické funkcie sa v schémach značia nasledovnými značkami (obr. 1.2).

[1] namiesto symbolu '0' sa zvykne hovoriť o úrovni L (low)
[2] úroveň H (high)



Obrázok 1.1: Schéma hradla NAND



značka AND značka OR značka NOT

značka XOR značka NAND značka NOR

Obrázok 1.2: Značky základných logických funkcií

1.2 Viacvstupové logické funkcie

V praxi je často nutné previesť napr. logický súčet či súčin na viac ako dvoch argumentov. Nie je nutné realizovať z polovodičových súčiastok nový člen s potrebným počtom vstupov, ale stačí poskladať takýto obvod z menších obvodov, ako je to ukázané v nasledujúcom príklade.

Príklad: trojvstupové AND so vstupmi x,y a z možno zrealizovať pomocou troch hradiel *and* s dvoma vstupmi napríklad takto: vypočíta sa konjukcia vstupov x a y a výsledok sa logicky vynásobí so vstupom z. Čiže AND(x,y,z) = (x *and* y) *and* z.

Položme si otázku, nakoľko efektívne sa dá táto úloha riešiť. Predpokladajme, že chceme zostrojiť obvod na logický súčin 7 členov. Môžme to urobiť dvoma spôsobmi:

Pri prvom spôsobe realizácie funkciu $AND(A, B, C, D, E, F, G)$ vyjadríme výrazom (H *and* (G *and* (F *and* (E *and* (D *and* (C *and*(B *and* A)))))), druhý spôsob je (X *and* (E *and* Y)), kde X = ((A *and* B)*and* C) a Y = ((F *and* G) *and* H).

V oboch spôsoboch riešenia majú obvody rovnaký počet hradiel AND, no líšia sa hĺbkou (a teda časom výpočtu). Vo všeobecnosti, ak máme N vstupov, tak prvým spôsobom vytvoríme obvod s hĺbkou $N-1$ a v druhom prípade obvod s hĺbkou $\lceil \log N \rceil$.

1.3 Zjednotenie, prienik a doplnok

Nech sú dané dva n-bitové binárne vektory. Na nich možno vykonať logické operácie[3]. Základnými logickými operáciami s binárnymi vektormi sú zjednotenie, prienik a negácia. Uvedieme ich realizáciu a príklady použitia.

a, zjednotenie

Definícia II.1: Nech A,B,C sú binárne veličiny, $A = a_0 \dots a_{n-1}$, $B = b_0, \dots b_{n-1}$ a $C = c_0, \dots b_{n-1}$. Zjednotením veličín A,B nazývame veličinu $C = c_1 c_2 \dots c_n$, pre ktorú platí:

$$c_0 = a_0 + b_0$$

$$\dots$$

$$\dots$$

$$c_n = a_n + b_n$$

Príklad II.1:

$$\begin{array}{r} 10100101 \\ \underline{01011100} \\ 11111101 \end{array}$$

Realizácia:

- na úrovni jedného bitu: jedným hradlom OR

[3] medzi jednotlivými zložkami týchto vektorov, t.j. medzi jednotlivými bitmi

- pre n-bitový vektor: Aby sme realizovali danú funkciu pre celý vektor, realizujeme ju pre každý bit. Príslušný obvod vznikne zlúčením n takýchto obvodov, čím dostaneme obvod so vstupmi $a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1}$ a výstupmi $c_0, c_1, \ldots, c_{n-1}$.

Cvičenie II.1: Navrhnite obvod[4] pre zjednotenie dvoch 4-bitových vektorov.

Príklad použitia: Spájanie informácií, napr. spojenie 2 podslov do jedného slova (nech sú dané podslová A,B a slovo C. Slovu C priraďme hodnotu B, slovo C posunieme vpravo tak, aby sa B dostalo do 'ľavej' časti slova C (na vyššie bity). Zjednotíme slovo C s podslovom A).

b, prienik

Definícia II.2: Nech A,B,C sú binárna veličiny, $A = a_0 \ldots a_{n-1}$, $B = b_0, \ldots b_{n-1}$ a $C = c_0, \ldots b_{n-1}$. Prienikom veličín A,B nazývame veličinu C ak platí:

$$c_0 = a_0 \cdot b_0$$

$$\ldots$$

$$\ldots$$

$$c_n = a_n \cdot b_n$$

Realizácia:

- na úrovni jedného bitu: jedným hradlom AND

Príklad použitia: -opačný proces - rozdelenie informácií, alebo získanie časti informácie.

Použime predchádzajúci príklad, majme slovo C, z ktorého chceme oddeliť podslovo A. Zaveďme slovo M, tzv. *masku*, ktorá bude obsahovať jednotky tam, kde sa v slove C nachádza podslovo B. Urobme prienik slov C a M, výsledkom je podslovo B.

c, doplnok

Definícia II.3: Nech A je binárna veličina $A = a_0 a_1 \ldots a_{n-1}$, Doplnkom veličiny A nazývame veličinu $C = c_1 c_2 \ldots c_n$ ak platí:

$$c_0 = \neg a_0$$

$$\ldots$$

$$\ldots$$

$$c_n = \neg a_n$$

Realizácia:

- na úrovni jedného bitu: jedným hradlom NOT

Príklad použitia: napr. pri binárnej aritmetike, alebo na maskovanie prerušení (viď ďalší text).

[4] t.j. nakreslite schému obvodu

1.4 Výber informácie - výhybka

V praxi sa často stretávame s problémom, keď do jedného miesta (vstupu obvodu) privádzame viacero rôznych údajov, z ktorých nás však zaujíma práve jeden, podľa splnenia istých podmienok.

Napríklad, nech je daný obvod pre aritmetiku na celých číslach, ktorého vstupom sú dva vektory X,Y a príkazový vektor S špecifikujúci operáciu, ktorá sa má vykonať. Chceme, aby za X i Y mohol byť dosadený ktorýkoľvek register počítača, pričom riadiacu informáciu rozšírime o S_X a S_Y, vyberajúcu registre dosadzované za X a Y. Potrebujeme teda použiť dva obvody s funkciou 'výberu'- z viacerých dátových informácií na vstupe chceme (vybrať) zobraziť na výstupe práve jednu, určenú riadiacou informáciou. S využitím tohto obvodu už je riešenie našej úlohy prosté: dátovým vstupom oboch obvodov bude sada registrov počítača[5] , na riadiace vstupy prvého z nich privedieme S_X a na druhého S_Y; a nakoniec, výstup z prvého pripojíme na vstup X a druhého na vstup Y.

Navrhnime spomenutý obvod, realizujúci výberovú funkciu. Tento obvod sa nazýva *výhybka*.

Obvod má dva dátové vstupy (binárne vektory) A, B a dva riadiace vstupy (bity) S_a a S_b. Na výstup sa zobrazí jeden zo vstupných vektorov. Na výber slúžia riadiace premenné S_a, S_b. Ak má premenná S_a hodnotu 1, výstupom je vektor A, ak má premenná S_b hodnotu 1, výstupom je premenná B. Stav, keď sú obe riadiace premenné rovné 1 nepredpokladáme (pretože to odporuje účelu obvodu- vybrať zo vstupov na výstup práve jeden). Táto funkcia je teda neúplne zadaná (viď časť I).

$\mathbf{S_A}$	$\mathbf{S_B}$	**C**
0	0	0
0	1	A
1	0	B
1	1	?

Tabuľka 1.1: Výhybka 3

Výber premennej sa realizuje pomocou vzťahu $Y = A \cdot S_a + A \cdot S_b$.

Realizácia:

- v prípade 1-bitových premenných A,B možno priamo použiť uvedený vzťah a zostrojiť obvod z dvoch hradiel AND a jedného hradla OR (čitateľovi odporúčame nakresliť si príslušnú schému).
- v prípade n-bitových premenných A, B sa tiež využije rovnaký princíp, príslušný obvod vytvoríme spojením n jednobitových obvodov.

Cvičenie II.2: Nakreslite schému výhybky pre dva štvorbitové vektory A, B.
Cvičenie II.3: Nakreslite schému výhybky pre štyri trojbitové vektory A,B,C,D.

[5] presnejšie povedané, na vstup budú privedené obsahy všetkých registrov

1.5 Testovanie parity

Pretože sa informácia prenáša medzi rôznymi zariadeniami, môže vplyvom rušenia dôjsť k poškodeniu informácie. Správny prenos sa kontroluje pomocou pridania dodatočnej informácie k správe, t.j. ďalších bitov. Pri prijímaní správy sa potom kontroluje hodnota týchto bitov. Najjednoduchším spôsobom je pridaním paritného bitu. Pridaný bit má hodnotu 1, ak je počet bitov nepárny a 0 ak je párny. Touto metódou možno odhaliť vznik jednej chyby.

Potrebujeme funkciu (obvod), ktorá umožňuje zistiť paritu danej informácie. S jej pomocou vieme:

- skontrolovať správnosť informácie (príjem)
- určiť hodnotu paritného bitu (vysielanie informácie)

Využijeme vlastnosť funkcie XOR. Táto funkcia nadobúda pre dve i viacej premenných hodnotu:

- **1**, ak je počet jednotiek nepárny
- **0**, ak je počet jednotiek párny

Návrh príslušného obvodu prenechávame na čitateľa.
Cvičenie II.4: Navrhnite obvod pre testovanie parity 4-bitového vektora.

1.6 Dekóder

Dekóder má n vstupov a 2^n výstupov. Výstupy sú označené číslami z intervalu $0 \ldots 2^{n-1}$. Dekóder interpretuje n-bitový vstup ako n-bitové číslo (n-ciferné binárne číslo) x, a na výstupe s číslom x sa objaví 1. Ostatné výstupy majú hodnotu 0.

Funkciu možno popísať tabuľkou 1.2 .

$\mathbf{z_{n-1}}$	...	$\mathbf{z_1 z_0}$	$\mathbf{e_{2^n-1}}$	.....	$\mathbf{e_1 e_0}$
0	...	0 0	0	.....	0 1
0	...	0 1	0	.....	1 0
	.			.	
	.			.	
	.			.	
1	...	1 1	1	.....	0 0

Tabuľka 1.2: Dekóder

Príklad zápisu funkcie pre 2 vstupy a 4 výstupy v DNF:
Majme vstupy z_0,z_1 a výstupy e_0,e_1,e_2, e_3. Funkciu môžeme zapísať:

$$e_0 = \bar{z_0}\bar{z_1}$$

$$e_1 = z_0\bar{z_1}$$

$$e_2 = \bar{z_0}z_1$$

$$e_3 = z_0 z_1$$

Použitie: pri dekódovaní inštrukcií, pri zápise a čítaní do pamäte, zisťovanie adries periférnych zariadení, atď[6]...

Cvičenie II.5: Navrhnite dekóder s dvoma bitmi na vstupe.
Cvičenie II.6: Navrhnite dekóder so štyrmi bitmi na vstupe.

Poznámka II.2: Nulový vektor je reprezentovaný signálom s nulovým napätím. Preto by neprítomnosť signálu na vstupe (obvod nikto nepoužíva) mohla byť interpretovaná ako nulový vektor, nula a dekóder by v tomto prípade dával $e_0 = 1$.

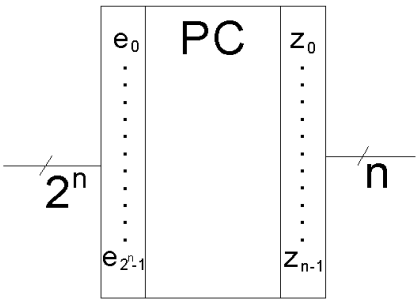
Je viacero možností, ako tomu zabrániť, uvedieme si jeden z nich. Zavedieme nový vstup E (*enable vstup*), udávajúci, či sa s obvodom pracuje. Ak $E = 0$, výstupom obvodu je nula bez ohľadu na hodnoty ostatných vstupov.

Teraz nám stačí jednoducho upraviť náš obvod. Úpravu prenechávame na čitateľa.

Enable signál budeme využívať aj neskôr, napr. pri tzv. synchronizácií obvodov.

1.7 Prioritný kóder

Plní opačnú funkciu ako dekóder. Má 2^n (očíslovaných) vstupov a n výstupov. Na výstup pošle číslo jednotkového vektora s najvyššou prioritou (t.j. vstupu na ktorom je jednotka a spomedzi všetkých takých vstupov má najnižšie číslo).



Obrázok 1.3: Značka prioritného kódera

Popis funkcie obvodu je v tabuľke 1.3.

[6] konkrétny spôsob použitia dekódera napríklad na dekódovanie inštrukcií i ďalšie spomenuté činnosti bude čitateľovi zrejmý neskôr

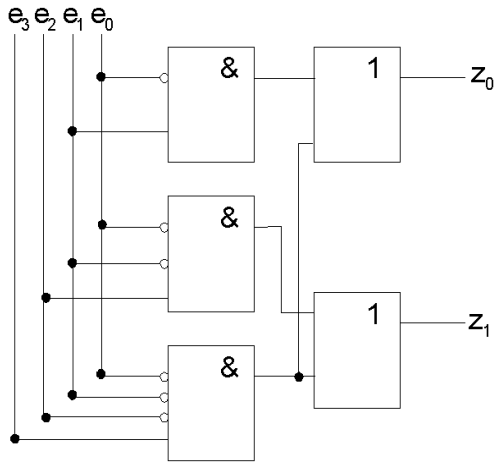
e_{2^n-1}	.....	$e_1 e_0$	z_{n-1}	...	$z_1 z_0$
0	.....	0 1	0	...	0 0
0	.....	1 0	0	...	0 1
	.			.	
	.			.	
	.			.	
1	.....	0 0	1	...	1 1

Tabuľka 1.3: Prioritný kóder

Použitie: napríklad pri mechanizme prerušení procesora– pri spracovaní žiadostí o prerušenie.

Nech naraz vzniklo viacero žiadostí o prerušenie a potrebujeme z nich vybrať prerušenie s najväčšou prioritou. Priveďme ich na vstup PC, na výstupe sa objaví číslo prerušenia s najvyššou prioritou (bol pripojený na vstup PC s najmenším číslom).

Realizácia obvodu:

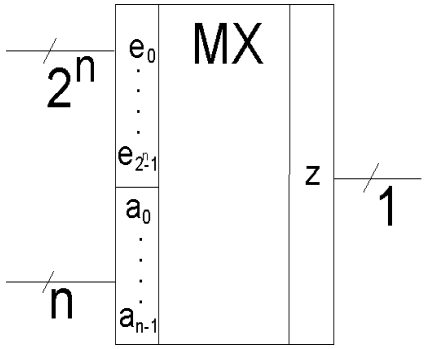


Obrázok 1.4: Schéma prioritného kódera

1.8 Multiplexor

Obsahuje 2^n informačných, n adresových vstupov a 1 výstup. Ak je na adresových vstupoch číslo x, na výstupe sa objaví hodnota toho informačného vstupu, ktorý má číslo x. Multiplexor teda funguje ako n-vstupová výhybka. Môže mať aj enable vstup.

Popis funkcie:



Obrázok 1.5: Značka multiplexora

$\mathbf{e_{2^n-1}}$	.....	$\mathbf{e_1 e_0}$	$\mathbf{a_{n-1}}$	...	$\mathbf{a_1 a_0}$	$\mathbf{z}$
–	.....	$-\ s$	0	...	0 0	s
–	.....	$s\ -$	0	...	0 1	s
	.			.		.
	.			.		.
	.			.		.
s	.....	$-\ -$	1	...	1 1	s

Tabuľka 1.4: Multiplexor

Použitie multiplexora:

- výhybka
- prevod informácie zo sériového na paralelný
- realizácia booleovských funkcií
- pripojenie registrov k ALU
- pripojenie viacerých obvodov k počítaču

Multiplexor sa dá realizovať dvoma spôsobmi:

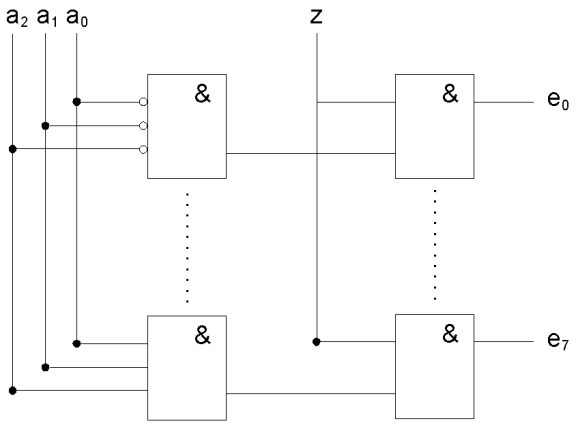
a, prvý spôsob (obr. 1.6)

b, druhý spôsob (realizácia pomocou dekódera– prenechávame na čitateľa v nasledujúcom cvičení:)

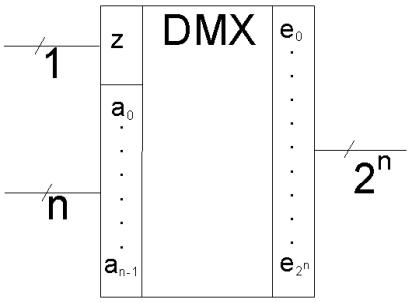
Cvičenie II.7: Navrhnite multiplexor. Využite dekóder.

1.9 Demultiplexor

Plní opačnú funkciu ako multiplexor. Jediný bit umiestňuje na zvolený výstup. Presnejšie, vstupmi demultiplexora sú: dátový bit z a bity $a_0, \ldots, a_{n-1}$ predstavujúce n-bitovú adresu. Hodnota z sa objaví na výstupe s adresou $a_0 a_1 \ldots a_{n-1}$, t.j. na adrese $e_{a_0 a_1 \ldots a_{n-1}}$
.



Obrázok 1.6: Schéma multiplexora

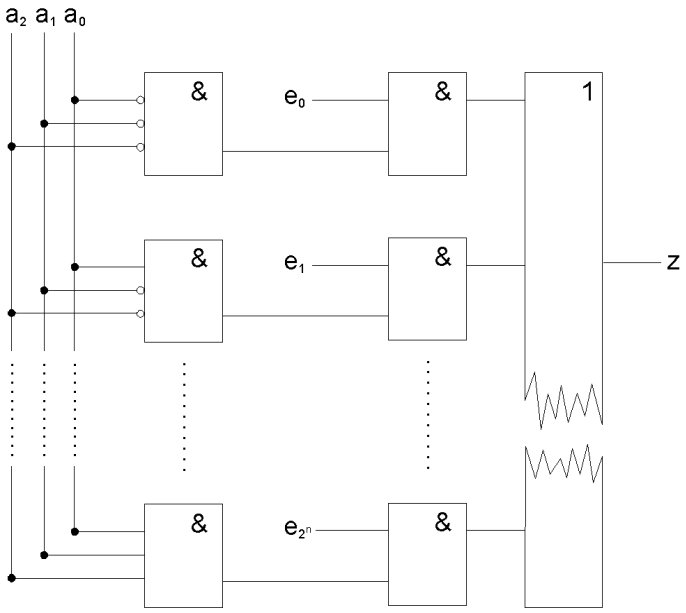


Obrázok 1.7: Značka demultiplexora

$\mathbf{z}$	$\mathbf{a_{n-1}}$	...	$\mathbf{a_1 a_0}$	$\mathbf{e_{2^n-1}}$		.....	$\mathbf{e_1 e_0}$
s	0	...	0 0	0	0	.....	0 s
s	0	...	0 1	0	0	.....	s 0
.		.				.	
.		.				.	
.		.				.	
s	1	...	1 1	s	0	.....	0 0

Tabuľka 1.5: Demultiplexor

Schéma demultiplexora je na obrázku 1.8.

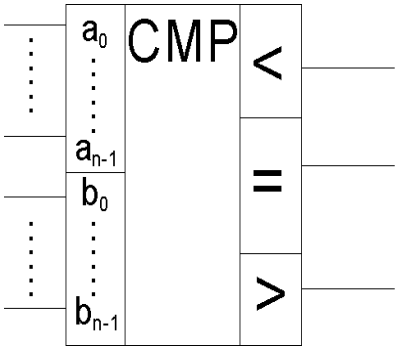


Obrázok 1.8: Schéma demultiplexora

1.10 Porovnávací obvod

Úlohou tohto obvodu je porovnať dve binárne veličiny A, B reprezentujúce celé čísla bez znamienka. Výsledkom by mala byť informácia, či $A = B$, $A > B$ alebo $A < B$. Naviac chceme, aby bolo možné takéto obvody určitým spôsobom spájať tak, aby sme mohli porovnávať čísla väčšieho rozsahu, než je rozsah vstupov A a B.

Na tieto účely slúži porovnávací obvod.

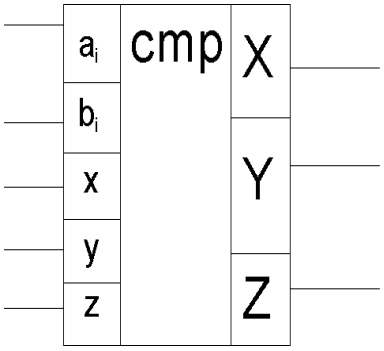


Obrázok 1.9: Značka porovnávacieho obvodu (komparátora)

Aby sme danú funkciu realizovali, pomôžeme si menšími obvodmi. Najskôr navrhneme obvod, ktorý porovnáva jednobitové vstupy. Predpokladajme, že obvodom porovnávame

i-te bity čísel A,B; pričom sme už porovnali všetky ostatné nižšie bity (t.j. od najnižšieho až po j-ty bit, kde j=i-1). Obvod má dátové vstupy a_i, b_i (ktoré predstavujú porovnávané bity) a vstupy x,y, a z (x má význam 'číslo A bolo doteraz menšie ako B', t.j. $a_{i-1} \ldots a_1 a_0 \leq b_{i-1} \ldots b_1 b_0$; vstupy y a z majú analogický význam v zmysle 'rovný', resp. 'väčší'). Výstupy obvodu X, Y a Z sú analógiou vstupov x,y a z – udávajú vzťah čísel $a_i a_{i-1} \ldots a_1 a_0$ a $b_i b_{i-1} \ldots b_1 b_0$.

Činnosť obvodu popisuje tabuľka 1.6.



Obrázok 1.10: Značka jednobitového komparátora

$\mathbf{a_i}$	$\mathbf{b_i}$	**x**	**y**	**z**	**X**	**Y**	**Z**
0	0	1	0	0	1	0	0
0	0	0	1	0	1	1	0
0	0	0	0	1	0	0	1
0	1	–	–	–	1	0	0
1	0	–	–	–	0	0	1
1	1	1	0	0	1	0	0
1	1	0	1	0	0	1	0
1	1	0	0	1	0	0	1

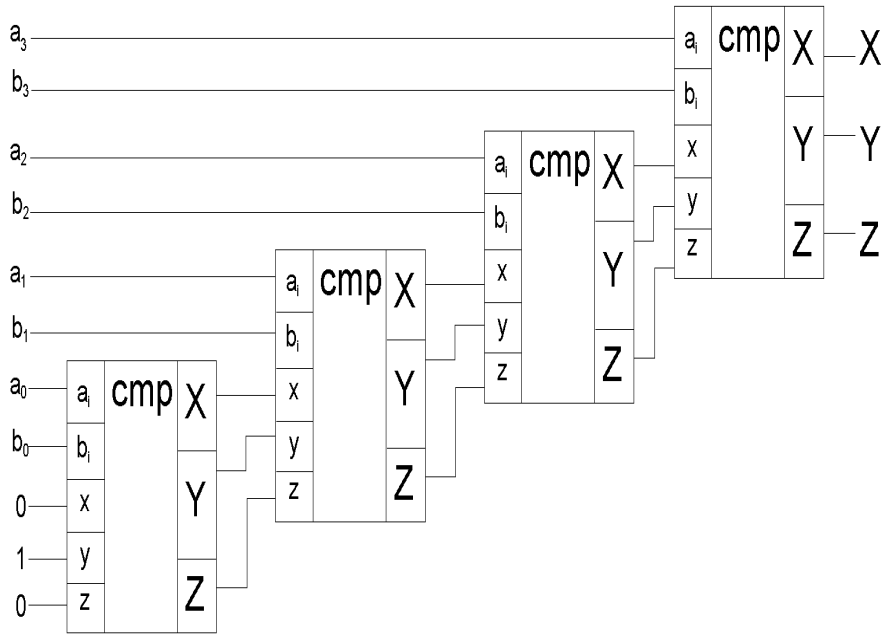
Tabuľka 1.6: Porovnanie jednobitových čísel

Technická realizácia jednobitovej sčítačky podľa tabuľky 1.6 je už triviálna a prenecháme ju na čitateľa.

Kaskádovitým spojením viacerých elementárnych obvodov vznikne viacbitový porovnávací obvod (obr. 1.11). Vstupom elementárnych porovnávacích obvodov môžu byť aj viacbitové čísla, napr. porovnávací obvod pre 16-bitové čísla môžeme vytvoriť spojením štyroch obvodov pre porovnanie štvorbitových čísel.

Cvičenie II.8: Navrhnite obvod cmp porovnávajúci dve štvorbitové čísla

Cvičenie II.9: Sú dané dva obvody CMP pre porovnanie osembitových čísel. Pomocou nich navrhnite obvod pre porovnanie dvoch šestnásťbitových čísel



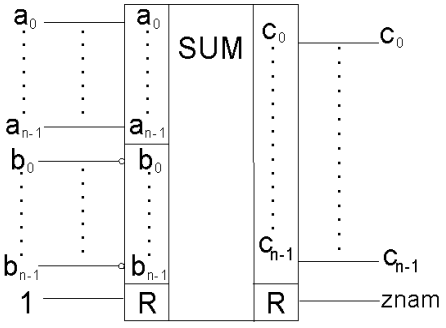
Obrázok 1.11: Schéma komparátora

1.11 Realizácia základných aritmetických operácií

V tejto kapitole navrhneme obvody realizujúce základné aritmetické operácie– sčítanie a odčítanie. Na ich realizáciu postačujú kombinačné obvody. Zložitejšie operácie, ako napr. násobenie a delenie sa vykonávajú pomocou sekvenčných obvodov a budú popísané neskôr.

1.11.1 Sčítačka (sumátor)

Je základný aritmetický obvod. Vstupom sú dve n-bitové binárne čísla $A + B$, výstupom je ich súčet $C = A + B$ a informácia, či došlo k pretečeniu (R).



Obrázok 1.12: Značka sčítačky (sumátora)

Najskôr realizujeme jednobitovú sčítačku. Jednobitová sčítačka sčíta 2 jednobitové

čísla + bit prenosu z predchádzajúceho rádu. Výstupom je výsledná číslica a prenos do vyššieho rádu.

$\mathbf{a_i}$	$\mathbf{b_i}$	$\mathbf{r_i}$	$\mathbf{c_i}$	$\mathbf{r_{i+1}}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabuľka 1.7: Jednobitová sčítačka

Vidíme, že

$$r_i = a_i b_i + a_i r_{i-1} + b_i r_{i-1}$$
$$c_i = a_i \quad XOR \quad b_i \quad XOR \quad r_{i-1}$$

Samotnú sčítačku vytvoríme skladaním menších sčítačiek, napríklad z jedno- alebo štvorbitových. Pre ilustráciu, spojme kaskádovite štyri jednobitové sčítačky (obr. 1.13).

Vidíme, že medzi vstupmi obvodu A, B sa objavil aj 'počiatočný prenos' R_{in}. Ak je rovný nule, na výstupe dostaneme $A + B$, jeho nastavenie na jedna má za následok prirátanie jednotky k výsledku, t.j dostaneme $A + B + 1$. Túto vlastnosť využijeme neskôr, pri realizácií odčítania; v sčitovacom obvode R_{in} nebude súčasťou vstupu a bude konštantne nastavený na 0.

Významný je výstupný prenos R_{out}. Je to vlastne prenos do rádu $n + 1$. Ak je nastavený, tak došlo k pretečeniu.

Cvičenie II.10: Navrhli sme jednobitovú sčítačku. Navrhnite štvorbitovú sčítačku, t.j. obvod *sum* sčitujúci dve štvorbitové čísla.

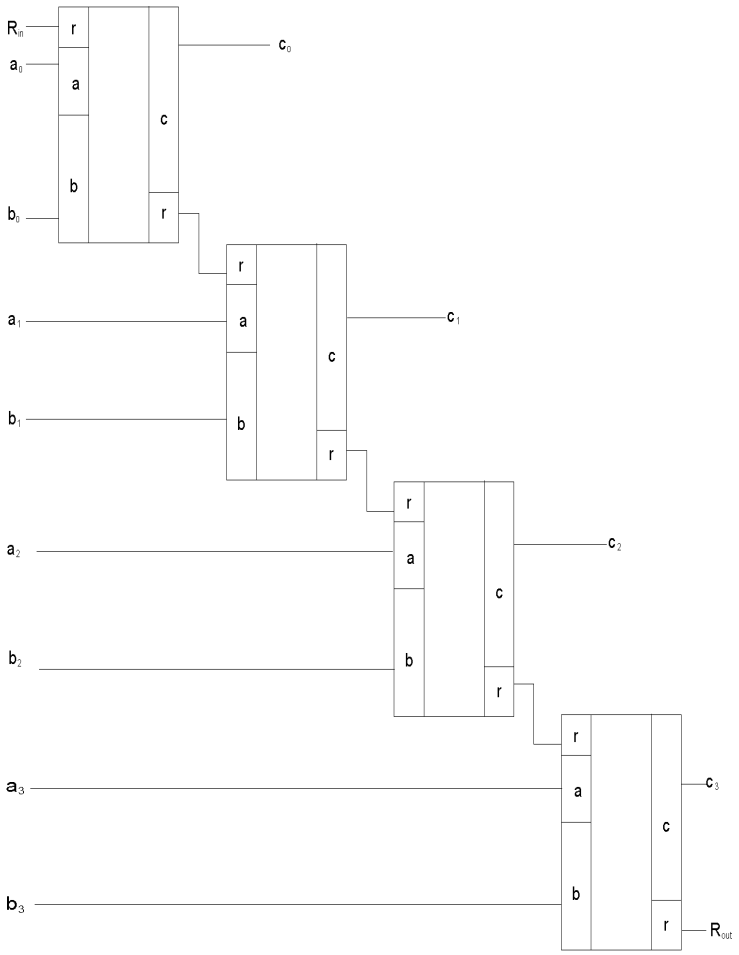
1.11.2 Sčítačka so zrýchleným prenosom

Kaskádovitá sčítačka je ľahko realizovateľná, avšak pomalá. Výpočet totiž musí prebiehať postupne: najprv treba vyrátať súčet číslic $a_i + b_i$ a určiť prenos r_i, až potom môže nasledovať výpočet $a_{i+1} + b_{i+1}$. Sčítačka má lineárnu časovú i priestorovú zložitosť.

Činnosť sčítačky vieme urýchliť tak, že dopredu vypočítame prenos pre každý rád. Tieto prenosy vieme vyrátať a urýchliť tak činnosť sčítačky, avšak na úkor priestorovej zložitosti. Takúto sčítačku nazývame sčítačka so zrýchleným prenosom.

Skúsme odvodiť vzťahy pre prenosy:

$$r_0 = a_0 b_0$$
$$r_1 = a_1 b_1 + a_1 a_0 b_0 + b_1 a_0 b_0$$
$$r_2 = a_2 b_2 + a_2 r_1 + b_2 r_1$$
$$r_k = a_k b_k + r_{k-1} \cdot (a_k + b_k)$$



Obrázok 1.13: Schéma sčítačky so sériovým prenosom

Ak dopredu vyrátame prenosy, časová zložitosť sčítačky so zrýchleným prenosom bude konštantná[7].

Potrebné je však pridať hradlá rátajúce prenosy dopredu. Koľko hradiel bude zrýchlená sčítačka mať? Odhadnime priestorovú zložitosť: nech $R_0, R_1, \ldots, R_n$ označujú zložitosť obvodu na výpočet $r_0, r_1, ..., r_n$. Platia rekurentné vzťahy:

$$R_0 = 1$$

$$R_1 = 7$$

a

$$R_i = 2R_{i-1} + 4 \qquad (pre \quad i \geq 1)$$

Z čoho vidno odhad R_n zdola:

$$R_n \leq 4 * 2^n - 2$$

Priestorová zložitosť je teda exponenciálna.

V praxi sa kombinujú obe metódy, napr. na realizáciu 32 bitovej sčítačky sa použije 8 štvorbitových sčítačiek so zrýchleným prenosom. Rôzne kombinácie a výslednú časovú a priestorovú zložitosť ukazuje nasledovná tabuľka: (údaj **štruktúra** udáva počet elementárnych sčítačiek $\times$ veľkosť vstupných slov elementárnej sčítačky v bitoch).

Štruktúra	**Priestorová zložitosť**	**Časová zložitosť**
1×32	2^{32}	1
2×16	$2 \cdot 2^{16}$	2
4×8	$4 \cdot 2^8$	4
8×4	$8 \cdot 2^4$	8
32×1	322	32

Tabuľka 1.8: Porovnanie časovej a priestorovej zložitosti pri rôznych konštrukciách 32-bitovej sčítačky.

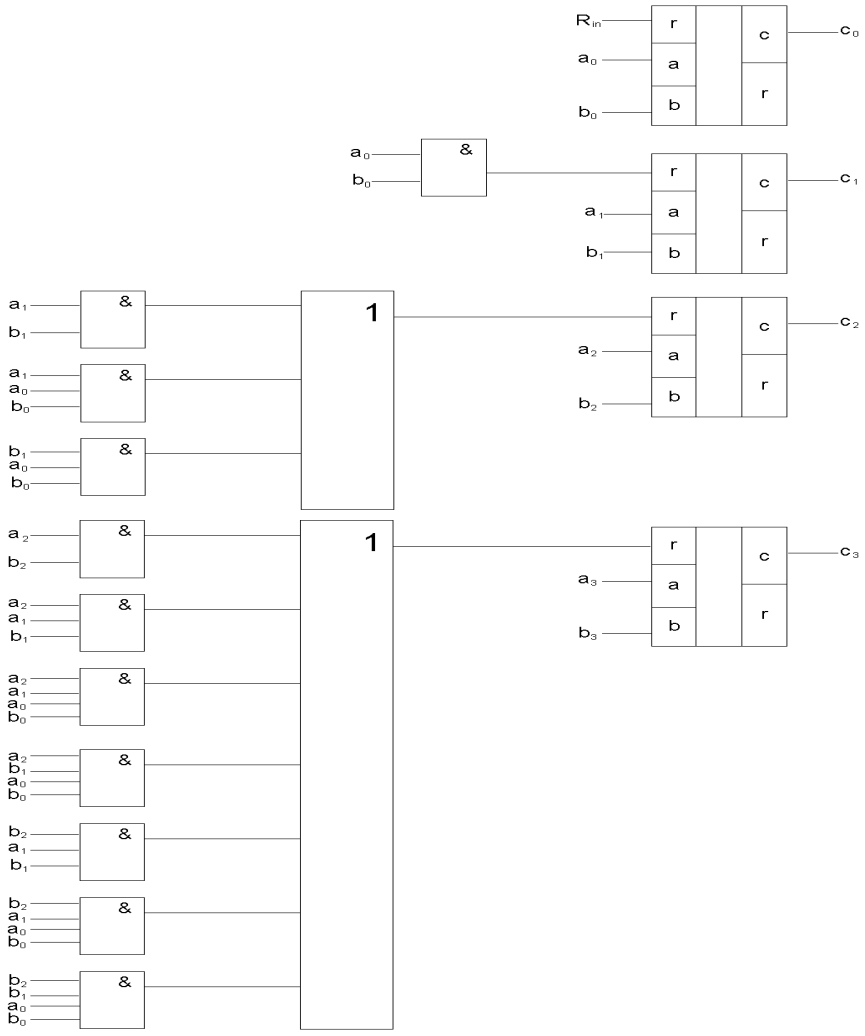
Uvedieme ešte schému sčítačky so zrýchleným prenosom (obr. 1.14).

1.11.3 Sčítačka pre čísla v doplnkovom kóde

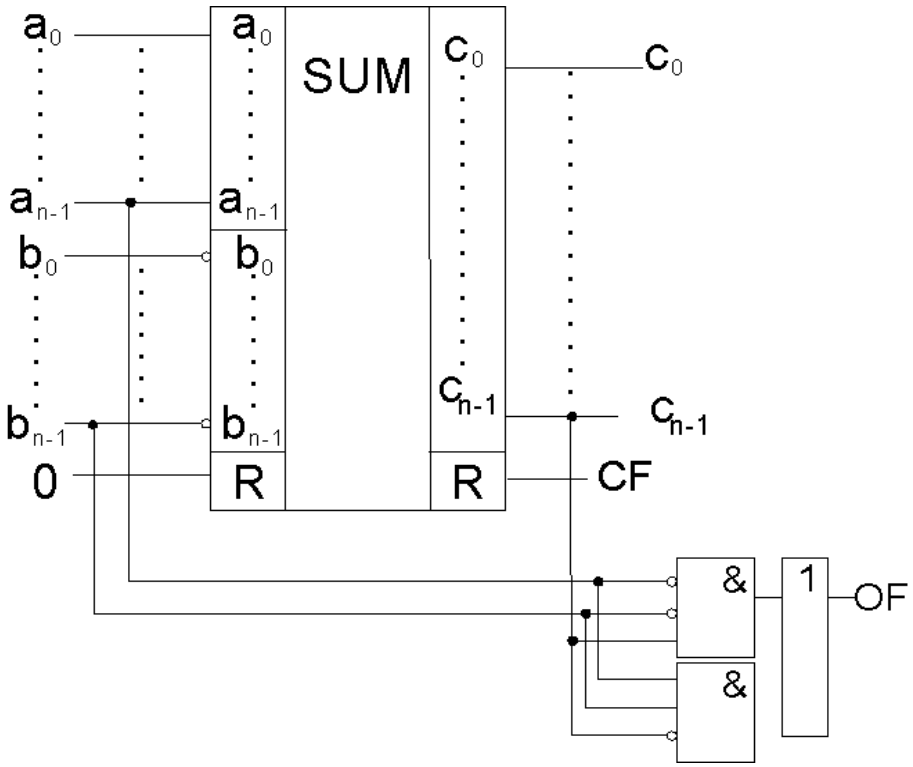
Na sčítavanie môžeme použiť 'klasickú' sčítačku, je však potrebné overiť konzistentnosť výsledku, t.j. či nedošlo k pretečeniu.

Pretečenie pri sčítavaní dvoch čísel v tvare binárnych doplnkov môže nastať len pri sčítavaní dvoch kladných alebo dvoch záporných čísiel. Podľa čoho ho spoznáme? Ak sme sčitovali dve kladné čísla a výsledok je záporný, alebo sme sčitovali dve záporné čísla a výsledok je kladný, tak došlo k pretečeniu. Pretečenie teda možno definovať nasledovne:

$$P = s_a s_b s'_c + s'_a s'_b s_c,$$



Obrázok 1.14: Schéma sčítačky so zrýchleným (paralelným) prenosom



Obrázok 1.15: Schéma sčítačky s určovaním OF

(kde P je paritný bit a s_a, s_b, s_c sú znamienka a, b, c)

1.11.4 Odčítačka

Sčítačku vieme použiť aj vo funkcii odčítačky. Využijeme pritom spomenutý vzťah:

$$A - B = A + \neg B + 1$$

To znamená, že rozdiel dvoch čísel A a B dostaneme tak, že sa číslo A sčíta s logickým doplnkom (negáciou) čísla B a k výsledku sa pripočíta jednotka. Túto jednotku môžeme priamo priviesť na vstup R_{in} sčítačky.

Takto realizovanú odčítačku možno použiť aj pre neznamienkový (binárny) kód a aj pre znamienkový (doplnkový) kód. V prípade binárneho kódu je výsledok rozdielu $A - B$ korektný iba ak $A \geq B$, v prípade doplnkového kódu treba detekovať pretečenie podobným spôsobom ako pri sčítaní.

Realizácia je jednoduchá, nebudeme ju uvádzať.

Cvičenie II.11: Navrhnite odčítačku čísel v binárnom kóde.

Cvičenie II.12: Navrhnite odčítačku čísel v doplnkovom kóde.

[7] nebudeme čakať na výsledok z predchádzajúho rádu - počet krokov nezávisí od počtu číslic sčítancov - preto je počet krokov konštantný

1.12 Aritmeticko-logická jednotka (ALU)

Aritmeticko-logická jednotka je súčasťou procesora. Realizuje aritmetické a logické operácie. Jej vstupom sú operandy a kód operácie, výstupom je výsledok operácie a informácie o priebehu výpočtu (napr. pretečenie).

Jednoduchá ALU môže vyzerať nasledovne:

Na vstup sa zadávajú vstupné údaje a riadiace signály. Vstupné údaje tvoria n-bitové čísla A,B a počiatočný prenos c. Riadiace povely M, $S_0, \ldots S_3$ určujú danú funkciu. Vstup M udáva, či sa jedná o logickú alebo aritmetickú funkciu, slovo $S_0S_1S_2S_3$ je kódom funkcie.

Naša ALU bude vytvárať 16 logických a 16 aritmetických funkcií, zahrnujúcich rôzne variácie sčítania a základných operácií. Uvedieme si len tie zaujímavé:

- *aritmetické*:

$F = A + B$	$F = A + B - 1$	$F = A$
$F = A - B$	$F = A + 1$	$F = A + A$
$F = A + B + 1$	$F = A - 1$	$F = 1$

- *logické*:

$F = A'$	$F = B'$	$F = A$	$F = (AB)'$
$F = AB$	$F = 1$	$F = B$	$F = A \oplus B$
$F = A + B$	$F = 0$	$F = (A + B)'$	$F = (A \oplus B)'$

Na realizáciu jej funkcií stačí sčítačka čísel v doplnkovom tvare a niekoľko základných hradiel. Pre vytvorenie kompletného obvodu ALU potrebujeme ešte niekoľko jednoduchých obvodov. Napríklad dekóder, s ktorým dekódovaním vstupu $MS_0S_1S_2S_3$ vyberieme 'správny' podobvod, t.j. obvod realizujúci požadovanú funkciu.

Nie je ťažké predstaviť si realizáciu tejto ALU, preto nakreslenie daného obvodu prenecháme na čitateľa.

Cvičenie II.13: Navrhnite spomínanú ALU.

Uvedená ALU je dosť jednoduchá; no jednoduchá bola aj jej realizácia. Bez väčších ťažkostí by sme dokázali pridať aj ďalšie operácie:

Cvičenie II.14: Navrhnite ALU s väčším množstvom funkcií (napr. s operáciami porovnania argumentov, sčítanie a odčítanie BCD čísel, s príznakmi *Sign* (znamienko výsledku), *Zero* (výsledok je nula), *Parity* (výsledok má párnu paritu), *Overflow* (pretečenie).

Zostrojenie takejto ALU tiež nebude ťažké; a pritom dostaneme súbor aritmeticko-logických inštrukcií veľmi blízky starším počítačom (napr. Sinclair), ktoré zložitejšie operácie (ako napr. násobenie) nemali zahrnuté v inštrukčnom súbore, ale bolo potrebné realizovať ich algoritmom skladajúcim sa z jedoduchých operácií.

CPU zvyčajne má aj inštrukcie pre zložitejšie operácie: násobenie, delenie, BCD aritmetiku, reálnu aritmetiku, posuvy a rotácie. Tie sa vykonávajú viackrokovými algoritmami a na ich realizáciu potrebujeme obvody s pamäťou, nazývané sekvenčné obvody (napr. už na vykonanie cyklu potrebujeme riadiacu premennú, ktorej hodnota sa číta a mení– musí byť uchovávaná).

Kapitola 2

Sekvenčné obvody

2.1 Všeobecná charakteristika sekvenčného obvodu

Kombinačné obvody realizovali jednoduché logické funkcie, ktorých výsledok záležal len na aktuálnych vstupoch. Výpočet nebol nijako ovplyvňovaný predchádzajúcimi vstupmi. Jednej vstupnej hodnote zodpovedala jedna výstupná hodnota.

Sekvenčné obvody sú obvody, ktoré sú závislé aj na predchádzajúcich vstupoch, resp. predchádzajúce vstupy ovplyvňujú výpočet s aktuálnym vstupom. Hovoríme, že obvod má určitý *vnútorný stav* z danej *konečnej množiny vnútorných stavov*, čo vlastne znamená, že súčasťou obvodu je konečná pamäť.

Preto jednej vstupnej hodnote môže zodpovedať veľa výstupných hodnôt (v závislosti od predchádzajúcich vstupov). Samotný názov sekvenčného obvodu pochádza z toho, že sa spracúvajú postupnosti (sekvencie) vstupov.

Príkladmi sekvenčných obvodov sú napríklad pamäťové členy, posuvné registre, počítadlá impulzov, aritmetické funkčné bloky zobrazujúce čísla sériovým spôsobom. Sekvenčné obvody môžeme tiež použiť ako riadiace obvody počítačov a iných zariadení pri riadení nespojitých procesov (napr. riadiaci obvod automatickej práčky).

Nemožno ich popísať jednoduchou tabuľkou, ktorou sa popisujú kombinačné obvody[1], možno však použiť opisný spôsob pomocou slovných popisov a obrázkov. Matematickým modelom sekvenčného obvodu je *konečný automat*.

Konečný automat je zariadenie, ktoré má v každom okamihu určitý vnútorný stav (prvok z danej konečnej množiny stavov), pracuje v krokoch, pričom v každom kroku prečíta jeden znak zo vstupu (rozoznáva konečnú množinu znakov) a na základe aktuálneho vnútorného stavu a prečítaného znaku prejde na nový vnútorný stav.

Pri kombinačných obvodoch nás čas nezaujímal; resp. predpokladali sme, že obvody pracujú nekonečnou rýchlosťou a teda čas výpočtu je nulový. Pri sekvenčných obvodoch sa však už prejavuje rýchlosť výpočtu jednotlivých hradiel. Napríklad, predstavme si sekvenčný obvod realizujúci nejaký algoritmus, ktorý na vstupe A a v stave B prejde do stavu C. Pritom ale prechod z B do C sa neuskutoční naraz, jedným

[1] napríklad pamäťový člen - nech má ako vstupy povely *Read, Write, Adress, Data* a výstup *Data* - jeho výstup závisí nielen od aktuálneho vstupu (t.j. aktuálneho príkazu pre pamäť), ale aj od uchovávaných dát, t.j. vnútorného stavu obvodu

krokom, ale vykoná sa postupnosť krokov, počas ktorej algoritmus prejde cez stavy $B = B_0, B_1, B_2, \ldots, B_{N-1}, B_N = C$. Predstavme si, že B je jediný stav, keď obvod číta vstup a na základe neho začne vykonávať nejaku činnosť. Ostatné stavy sú pracovné, $C = B$. Ak pred skončením výpočtu zmeníme vstup, obvod sa bude nachádzať v nejakom pracovnom stave B_X, v ktorom sa ho ani nebude pokúšať spracovať, resp. v iných obvodoch, kde aj 'pracovné' stavy čítajú vstup, to spôsobí nesprávny výsledok. Preto je vo všeobecnosti zmena vstupu dovolená len v určitých časových okamihoch, t.j. keď sa obvod nachádza vo 'vhodných' stavoch.

Preto nás budú zaujímať 'významné' časové okamihy, napr. zmeny stavu. Budeme predpokladať, že deje prebiehajú v *diskrétnom čase*. Takýto čas definujeme ako postupnosť *diskrétnych bodov*. Nezaujíma nás absolútny čas, ale dvom časovým okamihom $T_1, T_2 \quad (T_1 < T_2)$ priradíme celé čísla t a $t+1$. Pri takomto navrhnutí nadobúda čas hodnoty tvaru $-1, 0, 1, 2, 3,$ *atď*. Diskrétne body nazývame tiež *kroky* alebo *takty*.

Príklad II.2: (sekvenčného obvodu): Identifikátor výskytu práve jednej jednotky v slovách s dĺžkou 5.

Obvod ID5 bude mať vstupy x_1, x_2 a výstup y. Na vstup x_1 je privádzané sériovým spôsobom vstupné slovo. Vstup x_2 identifikuje začiatok nového slova.

Výstup v čase t je jedna práve vtedy ak $x_2(t) = 1$ a zároveň v postupnosti vstupov $x_1(t-4),\ x_1(t-3),\ x_1(t-2),\ x_1(t-1),\ x_1(t)$ bola práve jedna jednotka.

Z popisu uvedenej funkcie je zrejmé, že ju nemožno popísať jednoduchou funkciou $f: \quad X \longrightarrow Y$, čo v danom prípade predstavuje Booleovskú funkciu

$$f(x_1, x_2): \quad \{00, 01, 10, 11\} \longrightarrow \{0, 1\}$$

Čitateľovi odporúčame overenie na našom príklade.

Z predchádzajúceho príkladu si tiež všimnime schopnosť 'pamätania si', ekvivalentný s pojmom stavu.

Pred začatím štúdia sekvenčných obvodov ešte uvedieme formálny model na ich popísanie– *konečné automaty*.

Konečný automat je modelom výpočtového zariadenia, ktoré má konečnú vnútornú pamäť, pracuje v krokoch (taktoch) podľa (konečného) programu, pričom má na vstupe (konečnú) postupnosť znakov (konečnej) vstupnej abecedy. Presnejšie, v každom kroku prečíta symbol na vstupe, podľa neho a podľa stavu, v ktorom sa nachádza prejde na nový stav. Pravidlá prechodu určuje prechodová funkcia.

Vstupné slovo reprezentuje postupnosť vstupov ktoré prichádzajú do obvodu; jeden znak predstavuje jeden vstup. Stav automatu na konci reprezentuje výstup obvodu.

Formálne sú konečné automaty definované takto:

Definícia II.4: *Konečný automat* je usporiadaná štvorica: $(K, \Sigma, \delta, q_0, F)$, kde K je konečná množina stavov, Σ je konečná vstupná abeceda, $\delta \ : \ K \times \Sigma \longrightarrow K$ je prechodová funkcia, q_0 je počiatočný stav[2] a F je množina koncových stavov.

Príklad II.3: Horeuvedený obvod pre identifikáciu práve 1 jednotky v 5-bitovom slove zapíšeme konečným automatom:

[2] automat sa v ňom nachádza na začiatku výpočtu

$\Sigma = \{0,1\}$, $K = \{q_0, q_1, \ldots q_5, p_2, \ldots p_5, OUT_{false}, OUT_{true}\}$, počiatočný stav je q_0 a prechodová funkcia δ je určená nasledovnou tabuľkou (**q** označuje *pôvodný stav*, **z** označuje *prečítaný znak* (bit $x_1\times$ bit x_2) a $\mathbf{q}^*$ označuje *nový stav*. Znak '-' je použitý na označenie ľubovolného stavu, resp. znaku z množiny $\{0,1\}$):

q	**z**	$\mathbf{q}^*$
$-$	$-\times 1$	q_1
q_1	0×0	q_2
q_1	1×0	p_2
q_2	0×0	q_3
q_2	1×0	p_3
q_3	0×0	q_4
q_3	1×0	p_4
q_4	0×0	q_5
q_4	1×0	p_5
q_5	0×0	OUT_{false}
q_5	1×0	OUT_{true}

q	**z**	$\mathbf{q}^*$
p_1	0×0	p_2
p_1	1×0	OUT_{false}
p_2	0×0	p_4
p_3	1×0	OUT_{false}
p_4	0×0	p_5
p_4	1×0	OUT_{false}
p_5	0×0	OUT_{true}
p_5	1×0	OUT_{false}

Tabuľka 2.1: prechodová funkcia automatu ID5

V ďalšom texte uvedieme delenie sekvenčných obvodov na synchrónne a asynchrónne a ukážeme si základné typy sekvenčných obvodov. Ako pri kombinačných obvodoch, budeme najskôr navrhovať jednoduché obvody a ich syntézou vytvoríme zložitejšie obvody.

2.2 Asynchrónne a synchrónne sekvenčné obvody

Sekvenčné obvody môžeme rozdeliť na *synchrónne* a *asynchrónne*.

- *synchrónny*: má špeciálny krokovací vstup, pomocou ktorého jednotlivé prechody stavov krokujeme. Obvod prejde do nasledujúceho stavu až v prítomnosti krokového signálu. Preto mu môžeme posielať vstupné symboly bez obáv, že by jeden vynechal či započítal viackrát (ako sa môže stať pri asynchrónnom).
- *asynchrónny*: nie je časovo zosúladený so vstupom, neobsahuje krokovací vstup.

Asynchrónne obvody nemusia prechod z jedného stavu do druhého realizovať priamo, môžu mať pri zmene stavu viacero vnútorných (skrytých) stavov. Ak sa počas vnútorného stavu (preklápania) zmení vstup, môže dôjsť k vytvoreniu nevhodného stavu, čím vzniká chyba.

Synchrónne obvody sú riadené zvláštnym zdrojom synchronizačných, tzv. hodinových impulzov. Obvod číta vstup len počas trvania hodinového impulzu. Až do vyslatia ďalšieho hodinového impulzu má čas vstup spracovať a zobraziť výsledok na výstup.

Prechody počas výpočtu môžu byť takisto riadené ďalším hodinovým signálom, taktujúcim jednotlivé kroky výpočtu.

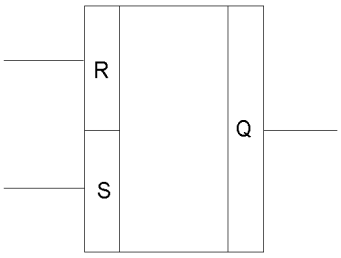
Obvykle sú synchronizačné signály pravidelné, čím sa synchronizuje činnosť celého systému. Na rozdiel od asynchrónneho obvodu nemôže dôjsť ku kritickým postupnostiam a vnútorným nestabilným stavom, časový interval medzi hodinovými impulzami sa volí s ohľadom na časové oneskorenia signálu. Je možné dovoliť aj zmeny vstupných signálov, lebo nemôžu ovplyvniť obvod bez prítomnosti synchronizačného impulzu. Podstatné je zariadiť, aby k zmene vstupných stavov nedochádzalo v dobe trvania hodinového impulzu.

Praktickým rozdielom medzi synchrónnymi a asynchrónnymi obvodmi je rýchlosť. V asynchrónnom obvode je určený začiatok každej operácie signálom informujúcim o skončení predchádzajúcej operácie. Časovanie asynchrónneho obvodu je teda riadené signálmi vznikajúcimi v ňom, pričom sa môže využiť maximálna rýchlosť základných obvodov. Rýchlosť asynchrónnych obvodov je preto väčšia než u synchrónnych, v ktorých sa rýchlosť (t.j. taktovacia frekvencia) musí prispôsobiť najpomajšej operácií (resp. obvodu).

Problematikou navrhovania asynchrónnych a synchrónnych obvodov, registrov a pamätí sa hlbšie nebudeme zaoberať, ukážeme si len ich základný princíp a popis. Čitateľa odkazujeme na príslušnú literatúru.

2.3 Klopný obvod SR

Klopný obvod, alebo tiež *pamäťový člen* je najjednoduchší sekvenčný obvod. Dokáže uchovať jednobitovú informáciu. Predstavuje teda *elementárnu* (jednotkovú) *pamäť*.



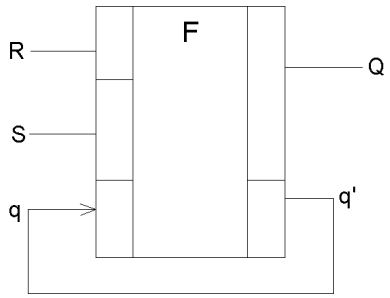
Obrázok 2.1: RS-člen

Klopný obvod SR má dve vstupy R (*reset*) a S (*set*) a výstup Q. Vstup R ho nastaví na hodnotu 0 ($t.j. Q = 0$) a vstup S na hodnotu 1 ($Q = 1$). Výstup Q zobrazuje uchovávanú informáciu.

Pokiaľ sa na vstup neprivedie žiadny signál, obvod nemení svoj stav (t.j. uchovávanú hodnotu).

Súčasné privedienie signálu na vstupy R i S znamená, že sa má obvod preklopiť do stavu 0 i 1, čo je nezmysel a preto je takýto vstup zakázaný.

Pretože klopný obvod je sekvenčným obvodom, tak nadobúda vnútorné stavy (reprezentujúce uloženú informáciu, t.j. hodnotu '0' alebo '1') a je popísateľný funkciou, ktorej



R	S	q	q′	Q
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	–	–
1	1	1	–	–

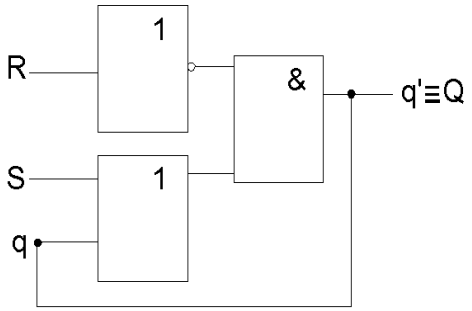
Obrázok 2.2: SR-člen

parametrom je okrem vstupov R, S aj aktuálny vnútorný stav a výstupom prechodovej funkcie je výstup Q a nový vnútorný stav (obr. 2.2).

Z tabuľky vidno vzťah pre vzájomnú závislosť q a Q:

$$Q = SR' + qR' = (S + q)R' = ((S + q)' + R)'$$

Technická realizácia SR člena je znázornená na obrázku 5.1:



Obrázok 2.3: Schéma RS-člena

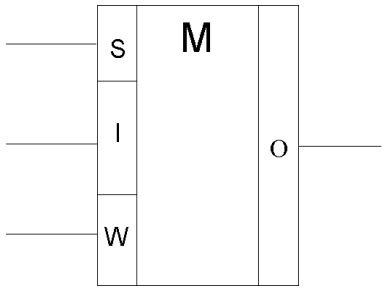
2.4 M-obvod a MEM-obvod

Ide o rozšírenie SR obvodu o selekčný, zapisovací a dátový signál.

Tento sekvenčný obvod je popísaný v tabuľke 2.3.
Z tabuľky po zjednodušení dostaneme vzťahy:

$$O = s \cdot w' \cdot q^*$$

$$q^* = s \cdot q \cdot w' + w \cdot i \cdot s + q \cdot s'$$



Obrázok 2.4: Značka M-obvodu

I	Hodnota na zapamätanie.
W	Povel pre zápis.
	ak $W = 1$, tak sa hodnota I uchová.
S	Výber obvodu.
	musí byť rovný 1, aby sme s obvodom mohli pracovať.
O	Uchovávaná hodnota.

Tabuľka 2.2: Vstupy a výstupy M-obvodu

S	**I**	**W**	**q**	**q***	O
0	–	–	0	0	0
0	–	–	1	1	0
1	–	0	0	0	0
1	–	0	1	1	1
1	0	1	0	0	0
1	1	1	0	1	0
1	0	1	1	0	0
1	1	1	1	1	0

Tabuľka 2.3: Popis M-obvodu

Realizáciu prenechávame na čitateľa.

M-obvod má jednu veľkú nevýhodu: nezaznamená hodnotu *v okamžiku prechodu* signálu W z 0 na 1. Riešením je *MEM-obvod*. Vstupom i výstupom sa zhoduje s M-obvodom, líši sa však konštrukčným riešením. Samotný konštrukčný popis neuvedieme, čitateľa odkazujeme na literatúru.

2.5 Iné klopné obvody

V tejto časti sa stručne oboznámime s troma ďalšími typmi klopných obvodov, ktoré boli vyvinuté:

- dvojstupňový klopný obvod MS-SR
- klopný obvod JK
- klopný obvod D

Ich funkcia je zhodná s funkciou klopného obvodu SR, líšia sa však pravidlami pre čítanie/zápis uchovávanej informácie. Preto uvedieme len ich tabuľky, popis signálov a značenie. Príslušné konštrukcie čitateľ nájde v technickej literatúre.

2.5.1 Dvojstupňový klopný obvod MS-SR

Obsahuje synchrónne i asynchrónne vstupy. S_1 a R_1 sú synchrónne, v závislosti od hodinového impulzu C. S_2 a R_2 sú asynchrónne a majú vyššiu prioritu než synchrónne vstupy (t.j. ak vstupy S_1 a R_1 majú inú hodnotu ako vstupy S_2, R_2, tak sa uvažujú vstupy S_2 a R_2).

2.5.2 Klopný obvod JK

Tento obvod sa riadi dvoma synchrónnymi vstupmi J a K, v závislosti na hodinových impulzoch C a dvoma asynchrónnymi vstupmi S a R. Asynchrónnymi vstupmi S, R možno nastaviť stavy 1 a 0 nezávisle na hodinovom impulze.

2.5.3 Klopný obvod D

Obsahuje vstupné signály D, C, S, R. Asynchrónne vstupné signály R, S klopný obvod nulujú (R) alebo nastavujú (S), nezávisle na hodinových impulzoch. Synchrónny režim je zabezpečený vstupným signálom D a hodinovými impulzami (signál C). Vstupy R, S majú najvyššiu prioritu.

2.6 Čítač

Čítač (angl. *counter*) predstavuje základ jednoduchých riadiacich jednotiek (a ako uvidíme neskôr) aj procesora.

Má jeden jednobitový vstup CP a n-bitový výstup. Na výstupe sa postupne zobrazujú čísla 0 až 2^{n-1}, pričom na výstupe je zobrazené číslo a až do ďalšieho objavenia sa impulzu CP (t.j. vstup $CP = 1$), potom sa zobrazuje číslo $a+1$. Pokiaľ $a = 2^n - 1$, tak $a+1 = 0$. Čítač je teda obvod s n-bitovou pamäťou, ktorý na výstupe zobrazuje uchovávané n-bitové číslo a ktorý po každom inpulze CP inkrementuje svoj obsah (n-bitové číslo) o jedna.

Pre ilustráciu uvedieme tabuľku pre čítač troj-bitových čísel. Čítač pracuje v binárnom kóde.

Postupnosť signálov CP	**Výstupné slovo**
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1
8	0 0 0

Tabuľka 2.4: Čítač (Counter)

Ako realizovať čítač? Stačí si uvedomiť pravidlá pre jednotlivé bity výstupu (a_0 až a_n):

- číslica a_0 zmení svoj stav (teda prechádza z 0 na 1 a naopak) po každom impulze CP
- číslica a_1 zmení svoj stav, keď číslica a_0 je v stave 1
- číslica a_2 sa zmení, keď obe číslice a_0, a_1 sú v stave 1

Pri technickej realizácií možno využiť napr. členy, ktoré obsahujú vstupné signály C a E. Signál C je hodinový, signál E je informačný(ak $C = 0$, tak sa zapíše obsah E). Využitím uvedených vzťahov pre výstupné bity čítača by už realizácia čítača mala byť pre čitateľa zrejmá.

Cvičenie II.15: Navrhnite trojbitový čítač.

V ukážke sme použili priamy binárny kód. Čítače sa však dajú zostrojiť aj pre iný výstupný kód, napríklad Grayov.

Cvičenie II.16: Navrhnite čítač pre čísla v priamom kóde.
Cvičenie II.17: Navrhnite čítač pre čísla v inverznom kóde.

Cvičenie II.18: Navrhnite čítač pre čísla v doplnkovom kóde.
Cvičenie II.19: Navrhnite čítač pre BCD čísla.
Cvičenie II.20: Navrhnite čítač pre čísla v Grayovom kóde.

2.7 Register

2.7.1 Jednoduchý register

Klopný obvod nám umožňuje zapamätať si jednoduchú binárnu informáciu, t.j. jeden bit. Spojením viacerých klopných obvodov je možné zapamätať si čísla väčšej dĺžky. Takto vytvoríme nízkokapacitnú pamäť, schopnú uchovať jedno číslo rozličnej dĺžky. Nazýva sa *register.*

Cvičenie II.21: Navrhnite n-bitový register so vstupmi W a $a_0, \ldots, a_{n-1}$ a výstupmi $c_0, \ldots, c_{n-1}$; pričom výstupy $c_0, \ldots, c_{n-1}$ predstavujú uloženú informáciu a pokiaľ je vstup $W = 1$, tak sa do pamäte zapíše vstup $a_0, \ldots, a_{n-1}$.

Registre môžu realizovať aj jednoduché operácie na svojom obsahu.

2.7.2 Funkcia posúvania

Nech je daná n-bitová binárna veličina $a_0, a_1, \ldots, a_{n-1}$, pričom je uložená v pamäťovom registri zloženého z n pamäťových členov $C_0, \ldots, C_{n-1}$. Posunutie *vľavo* znamená, že bit a_1 nadobudne pôvodnú hodnotu bitu a_0, bit a_2 nadobudne pôvodnú hodnotu bitu a_1 ...až bit a_{n-1} nadobudne pôvodnú hodnotu bitu a_{n-2}. Bit a_0 nadobudne hodnotu 0. Analogickým je posúvanie *vpravo.*

Príklad II.4:

nech register obsahuje binárne číslo	01010111
posunutím vľavo dostaneme	10101110
posunutím vpravo dostaneme	00101011

Podrobnejšie sa zaoberajme posúvaním vľavo. Vidíme, že sa pri ňom 'stráca' najvyšší bit a naopak, treba doplniť najnižší bit. V našom príklade sme ho doplnili nulou. Pokiaľ obsah registra chápeme ako celé číslo bez znamienka, potom posun vľavo predstavuje vynásobenie tohto čísla dvomi a posun vpravo celočíselné delenie dvomi.

Spomenieme ešte ďalšiu funkciu, *rotovanie* obsahu registra. Pri posúvaní sa bity strácajú, jeden bit stratíme a jeden treba doplniť. Pri rotovaní sa ten bit, ktorý by sa mal stratiť vloží do bitu, ktorý treba doplniť. Pri rotovaní vľavo teda urobíme posun doľava a do bitu a_0 vložíme bit a_{n-1}, rotovanie vpravo je analogické.

Príklad II.5: :

nech register obsahuje binárne číslo	01010111
rotovaním vľavo dostaneme	10101110
rotovaním vpravo dostaneme	10101011

2.7.3 Posuvný register

Posuvný register je register s pridanými vstupnými signálmi SL a SR. Ak $SL = 1$, tak obsah registra sa posunie vľavo; ak $SR = 1$ tak obsah registra sa posunie vpravo.

Na realizáciu posuvného registra použijeme rovnaké členy ako pri realizácií čítača. Každý člen má vstupné signály C a E. Signál C je hodinový, signál E je informačný. Samotnú realizáciu prenechávame na čitateľa.

Podobne je možné realizovať aj funkcie rotovania a vytvoriť tak nasledovný register:

$\mathbf{a_0 \ldots a_{n-1}}$	vstupné slovo
RR	rotácia vpravo
RL	rotácia vlavo
SR	posun vpravo
SL	posun vlavo
WE	povolenie k zápisu
	(uloží sa vstupné slovo)
RE	povel k čítaniu obsahu
	(zapíše sa do výstupného slova)
NUL	nulovanie obsahu
$\mathbf{b_0 \ldots b_{n-1}}$	vstupné slovo

Tabuľka 2.5: Vstupné a výstupné signály registra

Cvičenie II.22: Navrhnite register s funkciami z horeuvedenej tabuľky.

2.8 Aplikácie čítačov a posuvných registrov

Uvedieme niektoré možnosti aplikácií čítačov a posuvných registrov.

2.8.1 Násobenie dvoch dvojkových čísel

Metóda násobenia dvojkových čísel bola popísaná v časti I. Teraz si ukážme technickú realizáciu:

Nech v registroch A, B sú uložené dve binárne čísla, nech máme register C, do ktorého uložíme výsledok a register I, ktorý bude udávať polohu práve spracúvaného bitu. Algoritmus násobenia popíšeme nasledovne:

1. Inicializuj príslušné registre A, B, C, I

   $(C = 0, \quad I = 0)$

2. Je najnižší bit v A rovný nule?

   - ak NIE, do C ulož súčet $C + B$
   - ak ÁNO, pokračuj nasledovným krokom algoritmu

3. Posuň A o jeden bit vpravo, B o jeden bit vľavo

4. Inkrementuj I
5. Ak $I =$ Počtu bitov registra A tak skonči
6. Inak opakuj cyklus

Na realizáciu potrebujeme realizovať funkciu sčítania, funkciu posúvania, inkrementáciu čítača počtu posunov a funkciu porovnania dvoch veličín. Technická realizácia by mala byť po prečítaní predchádzajúcich kapitol čitateľovi zrejmá.

2.8.2 Prevod zo sériového na paralelný tvar a naopak

Pri prenose údajov niektoré zariadenia používajú sériový prenos, iné paralelný. Je nutné mať obvody prevádzajúce sériový tvar na paralelný a naopak.

Pri realizácií použijeme čítač C a posuvný register R. Algoritmus vyzerá nasledovne:

1. Inicializuj R a C na nulu
2. Vlož do N paralelnú informáciu
3. Posuň R o krok vpravo
   (výstupný, najnižší bit sa vyšle na výstup)
4. Inkrementuj C
5. Je $C = 0$?
   - ak nie, choď na krok 3
   - ak áno, koniec prenosu
6. Koniec prenosu slova, priprav ďalšie slovo.
   Choď na krok 2.

2.8.3 Iné aplikácie sekvenčných obvodov

Uveďme si ešte niektoré ďalšie možnosti aplikácií:

- hľadanie jednotkového bitu
- výber bitu
- určenie počtu jednotiek v slove
- časové oneskorenie (spomaľovač)

2.9 Realizácia pamäte

V predchádzajúcej kapitole sme zrealizovali register, nízkokapacitnú pamäť. Na podobnom princípe môžeme skonštruovať aj pamäť väčšieho rozsahu, pamäte typu RAM či SAM. Podrobným popisom pamätí a princípmi ich realizácie sa budeme zaoberať v časti V .

Kapitola 3

Riadiace obvody

3.1 Zložitejšie sekvenčné obvody

Na realizáciu zložitejších úloh je potrebné použiť zložitejšie sekvenčné obvody. Kvôli zjednodušeniu návrhu sa obvod rozdelí na dve časti:

1. *aplikačnú časť*, ktorá obsahuje funkčné prostriedky (registre, pamäte)
2. *riadiacu časť* (*radič*), ktorá riadi činnosť funkčných obvodov na základe ich stavu, svojho stavu a hodinového signálu. Realizuje teda príslušný vývojový diagram.

Správne časovanie jednotlivých krokov je riadené hodinovým signálom.

3.2 Radič

Radič je teda obvod, ktorý:

1. riadi činnosť funkčných obvodov (na základe aktuálnych vstupov a svojho vnútorného stavu)
2. riadi svoj vlastný stav (na základe rozhodovacej časti)

Radiče delíme na *obvodové* a na *mikroprogramové*.

Obvodový radič môžeme vytvoriť napríklad z posuvného registra, v ktorom rotuje práve jedna jednotka[1]. Môže sa posunúť, zostať na mieste, alebo prejsť do inej, 'nesusednej' polohy. Určuje stav radiča (a inicializuje príslušnú 'vetvu' obvodov). Namiesto posuvného registra môžeme použiť aj čítač s dekóderom (ktorý dekóduje stav čítača a utvorí výstup rovnakého tvaru ako pri posuvnom registri).

Mikroprogramový radič okrem čítača obsahuje aj (pevnú) pamäť ROM. Výstupy čítača sú pripojené k adresovým vstupom pamäte. Slovo vystupujúce z pamäte obsahuje bity určujúce jednotlivé akcie (napríklad, pre každý riadiaci signál obvodu obsahuje slovo jeden bit určujúci či sa má signál aktivovať alebo nie). Príslušnú činnosť, ktorá sa má vykonať teda rozložíme na postupnosť viacerých krokov, elementárnych akcií, *mikrooperácií*. Hovoríme, že mikroprogramový radič vykonáva činnosť popísanú *mikroprogramom*. Tento spôsob realizácie je výhodnejší najmä pri zložitejších obvodoch. Bližšie ho popíšeme v časti o mikroprocesoroch (časť V).

[1] t.j. register obsahuje binárny vektor, v ktorom sa nachádza práve jedna jednotka

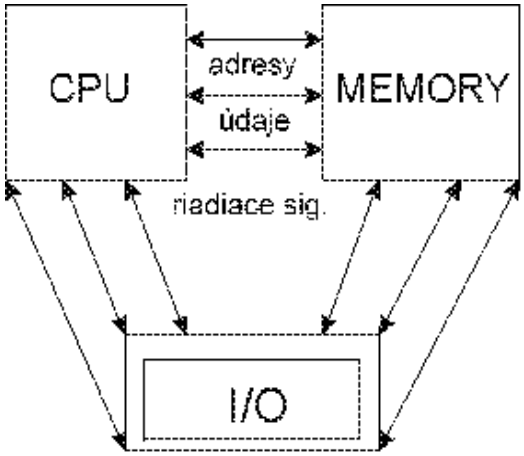
Časť III

Procesor

Počítač vykonáva činnosť popísanú programom. Program je uložený v pamäti. Počítač na základe vstupných údajov, získaných zo vstupných zariadení utvorí výstupné údaje a zapíše ich na výstupné zariadenie.

Počítač je tvorený pamäťou, vstupnými a výstupnými zariadeniami a ďalej časťou schopnou vykonávať program, teda interpretovať príkazy (inštrukcie) programu. Táto časť sa nazýva procesor.

Štruktúru počítača potom môžeme znázorniť nasledovne: (obr. 1)



Obrázok 1: Štruktúra počítača

Aby procesor mohol interpretovať inštrukcie programu, musí obsahovať obvody schopné vykonávať operácie (*aritmeticko-logická jednotka*) a obvody interpretujúce inštrukcie programu (*riadiace obvody- radič*).

V predchádzajúcich častiach sme opísali princípy realizácie všetkých častí procesora: aritmetickej jednotky i riadiacich obvodov– logickej jednotky. Po podrobnom opísaní práce procesora by už čitateľ mal byť schopný principiálne navrhnúť jednoduchý procesor. Takisto by mal byť schopný pochopiť aj princípy práce moderných procesorov, ktoré v snahe dosiahnutia čo najväčšieho výkonu nepoužívajú len nové hardwarové technológie, ale snažia sa napr. paralelne vykonávať viacero inštrukcií v jednom takte, či majú nové inštrukcie pre 'zložité' operácie (napr. reálnej aritmetiky).

Naše rozprávanie o procesoroch rozdelíme na dve časti.

V prvej budeme hovoriť o princípoch činnosti 'jednoduchého', bežného procesora. Najskôr definujeme účel procesora, rozdelíme procesory podľa rôznych kritérií a opíšeme funkcie jednotlivých častí procesora. Potom sa budeme venovať inštrukčnému súboru a vykonávaniu inštrukcií. Opíšeme princípy realizácie procesora. Popíšeme riadiacu jednotku, možné princípy jej realizácie- hardwarovo a mikroprogramo realizovanú riadiacu jednotku.

Ďalšia časť poslúži pre pochopenie princípov moderných procesorov. Moderné procesory sa snažia využiť pre zrýchlenie svojej práce - t.j. zvýšenie výkonu - využiť aj 'iné cesty' ako je rýchlejší hardware. V tejto časti sú opísané niektoré z týchto technológií. Napríklad technológia MMX, paralelné vykonávanie inštrukcií (pipeline, superskalárne vykonávanie) či procesory s architektúrou RISC.

Kapitola 1

Popis procesora

1.1 Funkcia a klasifikácia procesorov

Procesor je jednotka číslicového počítača alebo počítačového systému, ktorá realizuje hlavné operácie pri riadení vykonávania i samostatnom vykonávaní výpočtu, zadaného programom.

Podľa zamerania procesorov môžeme rozlíšiť:

- *Univerzálne procesory*, ktoré sú časťou základnej jednotky číslicového počítača, ktorá vykonáva hlavné operácie a zabezpečuje riadenie jeho ostatných častí prostredníctvom interpretácie inštrukcií programu.
- *Problémovo-orientované procesory*, čo sú špecializované jednotky číslicového počítača, ktoré slúžia na riešenie špeciálnych úloh vo vybratých oblastiach. Sú vybavené prostriedkami pre riešenie problémovo orientovaných úloh. Uvedieme príklady niektorých z nich:
  - *Aritmetický procesor.* Slúži na riešenie vybraných aritmetických operácií. Najčastejšie operáciu nevykonáva priamo, ale ju realizuje postupnosťou viacerých elementárnych krokov- mikroprogramom (viď ďalej). Príkladom operácií môžu byť aritmetické operácie v pohyblivej rádovej čiarke, ktoré vykonáva procesor pohyblivej rádovej čiarky, angl. Floating Point Processor (alebo len FP processor).
  - *Kanálový procesor.* Je jednotka číslicového počítača, ktorá zabezpečuje riadenie vstupno-výstupných operácií. Obdobou kanálového procesora je vstupno-výstupný procesor, ktorý predstavuje jednoúčelový procesor, pracujúci nezávisle od základnej jednotky počítača. Okrem riadenia realizácie vstupno-výstupných inštrukcií vykonáva tiež kontrolu správnosti údajov, úpravu formátov, zmenu kódov a iné.
  - *Videografický procesor.* Je špecializovaný procesor, určený na spracovanie grafickej informácie. Realizuje mnohé grafické operácie, ako napríklad transformáciu súradníc (posúvanie, otáčanie, zväčšovanie, zmenšovanie), filtráciu šumu, označovanie oblastí (separácia a farbenie), operácie nad obrazovými prvkami (aritmetické a logické operácie, korekcie tieňovania) a operácie podporujúce trojdimenzionálnu grafiku. Rýchlosť počítača s týmto procesorom je

pri grafických aplikáciách výrazne väčšia, čo má význam napríklad pri spracovaní obrazových informácií v reálnom čase.

– *Procesor realizujúci algoritmus rozpoznávania.* Je určený na rozpoznávanie obrazcov, písma alebo prirodzenej reči. Procesory tejto triedy zvyčajne majú špecializovanú paralelnú architektúru (navrhnutú napr. na báze neurónových sietí).

Samotné *počítačové systémy* môžeme rozdeliť z hľadiska architektúry na *univerzálne* a *špecializované.*

Rozoznávame tiež *jednoprocesorové* a *viacprocesorové* systémy. Prvá skupina (*monoprocesorové* systémy) má základnú jednotku navrhnutú na báze jedného procesora, v druhej skupine riadiaca jednotka obsahuje viacero procesorov.

V ďalšom sa budeme zaoberať monoprocesorovými počítačmi. Podľa výkonu a úloh, na ktoré sú určené ich môžeme rozdeliť na:

- Mikropočítače
- Personálne počítače
- Pracovné stanice
- Strediskové počítače

Mikroprocesor bol vytvorený v 70.rokoch vďaka zdokonaleniu technológie výroby integrovaných obvodov, ktorá umožnila umiestniť na jeden čip množstvo číslicových obvodov. Mikroprocesor je programovateľný sekvenčný automat určený na riešenie jednoduchých úloh. Môže byť tiež súčasťou jednoúčelových zariadení (napríklad kuchynského robota) ako jeho riadiaci prvok. Počítač na báze mikroprocesora sa nazýva *mikropočítač.* Ich vývoj začala firma Intel, ktorá v roku 1971 uviedla na trh štvorbitový procesor I4004.

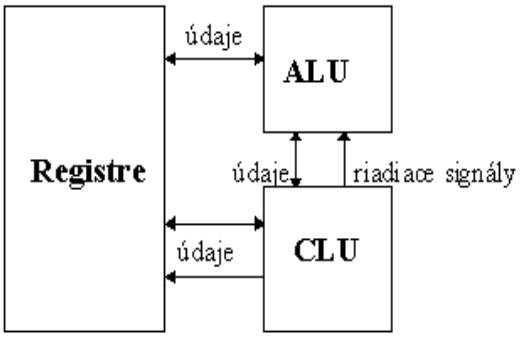
Personálne počítače sú konštruované na báze výkonných mikroprocesorov. Určené sú na jednopoužívateľské aplikácie. Okrem nenáročných aplikácií v domácnosti ich možno využiť aj na profesionálne účely: na vedenie administratívy, vytváranie riadiacich a informačných systémov, riešenie vedecko - technických úloh. Slúžia tiež na vytváranie viacpoužívateľského prostredia v systémoch počítačových sietí a pracovných staníc. Personálne počítače sa objavujú v 80.rokoch. V súčasnosti sú v tejto oblasti štandardom počítače typu IBM PC/AT.

Pracovné stanice a *strediskové počítače* sú počítačové systémy určené pre riešenie náročných úloh. Zväčša sú založené na rovnakých princípoch ako personálne počítače, majú však výkonnejšie komponenty (napr. výkonnejší procesor, väčšiu pamäť), prípadne sa jedná o viacprocesorové systémy. Preto sa im nebudeme venovať osobitne.

1.2 Schéma procesora

Mikroprocesor (*Central processing unit- CPU*) sa skladá z troch častí:

1. *sady registrov* (Register set)
2. *aritmeticko - logickej jednotky* (*ALU*)



Obrázok 1.1: Štruktúra procesora

3. *riadiacej jednotky* (radiča, *CLU - Control Logic Unit*)

Registre slúžia na uchovávanie informácií s ktorými bezprostredne CPU pracuje. Register je rýchla nízkokapacitná pamäť, uchovávajúca jedno slovo. Môžu to byť vstupné operandy, výsledky operácií, prechodné údaje, adresy, kód príkazu, riadiace údaje, príznaky. Práca s registrami je rýchlejšia ako práca s bežnou pamäťou[1].

Aritmeticko - logická jednotka (ALU) je schopná vykonávať aritmetické a logické operácie. Môžu to byť nielen základné operácie sčítanie a odčítanie, ale aj zložitejšie operácie násobenie a delenie. Čísla môžu byť spracované v rôznych formátoch. Súčasťou ALU môže byť aj jednotka na prácu s reálnymi číslami (*FPU- Floating point unit*).

Riadiaca jednotka (CLU) určuje postupnosť operácií, ktoré sa majú vykonať. Riadiaca časť prečíta z pamäte príkazy programu, dekóduje ich a potom riadi ich vykonávanie.

Pretože vykonávanie inštrukcie (resp. pri realizácií operácie určenej inštrukciou) je zložitým procesom, na ktorom sa podieľa množstvo častí (jednotiek) procesora i 'neprocesorových' častí počítača, je nutné zabezpečiť ich vzájomnú synchronizáciu. Využíva sa pritom *časovač*, alebo *generátor hodinových impulzov*, ktorý v pravidelných intervaloch generuje signály (nazývané aj ako *hodinové signály*). Celý proces objasníme v ďalšej kapitole, pojednávajúcej o realizácií procesora.

1.3 Inštrukcie, inštrukčný súbor

Inštrukcia je binárny vektor, ktorý v súlade s dohodnutou konvenciou určuje nejakú operáciu a jej operandy. Je to zápis určujúci počítaču, ktorú operáciu a s ktorými operandami má vykonať.

[1] Dôvodov je niekoľko:

1. Pamäť sa nachádza mimo procesora; procesor s ňou komunikuje po zbernici; podľa predpísaného protokolu– kým k registrom pristupuje 'priamo'.
2. Dôležitým atribútom pamäte je nielen rýchlosť, ale aj cena. Pretože však registre majú oproti pamätiam zanedbateľnú veľkosť, tak na realizáciu registrov možno použiť technológie rýchlejších a drahších pamätí.

Inštrukcie zahŕňajú jednoduché aritmetické operácie, presuny dát, vetvenie programu, komunikáciu so vstupno- výstupnými zariadeniami. Operandami môžu byť konštanty, registre, alebo pamäťové miesta.

Podrobnejší prehľad operácií uvedieme na záver tejto kapitoly.

Množinu všetkých inštrukcií nazývame *inštrukčná sada* (*instruction set*). Bohatá inštrukčná sada, umožňujúca vykonávanie zložitých operácií uľahčuje prácu programátora a zrýchľuje vykonanie programu[2]. Je však náročnejšia na technickú realizáciu, pretože od nej závisí vnútorná organizácia počítača.

1.3.1 Formát inštrukcie

Každá inštrukcia je kódovaná nejakým binárnym reťazcom (vektorom). Formát inštrukcie popisuje, na aké časti je binárny vektor rozdelený a ako sa majú jednotlivé časti interpretovať. Časti binárneho vektora sa nazývajú *polia* (*fields*):

- *operačné pole* (*operačný kód*) určuje operáciu
- *pole operandov* určuje operandy

Operandom inštrukcie môže byť konštanta, register alebo pamäťové miesto. Ak je argumentom pamäťové miesto, jeho adresu určuje *adresové pole*. Pamäťovú bunku však nemusíme špecifikovať (adresovať) len uvedením jej adresy; procesor môže mať implementovaných viacero spôsobov ako má na základe určitých údajov programátora zistiť adresu pamäťovej bunky. Tieto spôsoby sa súhrnne označujú ako *metódy adresácie*. Podrobnejšie sa nimi budeme zaoberať v kapitole 1.5 .

1.3.2 Typy inštrukcií

Základné rozdelenie inštrukcií podľa činností ktoré vykonávajú je nasledovné:

1. presuny dát (prenos informácií medzi registrami, medzi procesorom a hlavnou pamäťou)
2. aritmetické a logické operácie
3. presuny údajov z V/V zariadení
4. vetvenie programu
5. riadiace inštrukcie (organizácia činnosti počítača, koordinácia činnosti jeho podsystémov, organizácia jeho využitia viacerými užívateľmi)

Tieto skupiny zahŕňajú nasledujúce operácie[3]:

- *presun údajov*:

[2] zložitejšiu operáciu je možné vykonať aj sotvérovo, pomocou viacerých jednoduchých operácií, ktoré už sú realizované hardvérovo (t.j. pomocou programu inštrukcií)– táto realizácia operácia je však pomalšia ako hardvérová realizácia (už len preto, lebo nie je potrebné 'strácať čas' procesora nahrávaním inštrukcií programu z pamäte a ich dekódovaniu)

[3] to, s akými argumentami možno nasledovné operácie previesť uvedieme neskôr

  - priradenie hodnoty jedného operandu druhému operandu
  - výmena hodnôt operandov
  - práca s blokmi údajov v pamäti (kopírovanie, presun, priradenie konštantnej hodnoty)
  - operácie so zásobníkom (vkladanie/výber udájov do/zo zásobníka)

- *výpočty*:
  - (aritmetické)
    - základné operácie ($+$, $-$, $*$, $/$) pre čísla bez znamienka i pre čísla so znamienkom
    - inkrementácia a dekrementácia
    - porovnávanie operandov
    - zmena dĺžky operandu (napr. slabiky na slovo)
    - BCD aritmetika, prevod bin. čísel na BCD a naopak
    - reálna aritmetika[4]
    - programové nastavenie aritmetických príznakov
  - (logické)
    - logické funkcie (AND, OR, NOT, XOR a ďalšie)
    - posuvy, rotovania
    - nastavovanie jednotlivých bitov argumentov
- *I/O zariadenia*:
  - vstup a výstup z I/O zariadení
- *riadenie programu* (vetvenia, skoky, cykly):
  - skoky; absolútne i relatívne zadanie adresy skoku podmienené skoky (vykonané len v prípade platnosti určitých príznakov)
  - cykly
  - podprogramy; príkaz pre skok do podprogramu, príkaz pre návrat z podprogramu
- *riadenie programu*
  - prerušenia; vyvolanie prerušenia, maskovanie prerušenia (maskovanie), vytvorenie podprogramu na obsluhu prerušenia a návrat z neho
  - podpora virtuálnej pamäte
  - podpora ochrany; určenie privilégií pre jednotlivé programy. (Každý program (proces) má pridelené určité privilégiá (priority), a program s nižšími právami nemôže zasahovať do programu s vyššími.)
  - inštrukcia nevykonávajúca žiadnu akciu

    (má význam napríklad ak chceme aby program trval presne určený počet krokov– taktov)
  - iné

[4] aritmetické operácie na reálnych číslach

1.3.3 Dĺžka zápisu inštrukcie

Dĺžka inštrukcií inštrukčnej sady môže byť buď jednotná alebo premenlivá. Jednotná dĺžka sa používa pri jednoduchých procesoroch s malou inštrukčnou sadou. Premenlivá dĺžka sa uplatňuje pri rozmanitých spôsoboch adresácie alebo pri dlhších inštrukčných súboroch.

1.3.4 Čas trvania inštrukcie

Je premenlivý, závisí od zložitosti realizácie operácie a od zložitosti získania operandov.

Napríklad, inštrukcie zápisu do registra sú jednoduchšie (a tým aj rýchlejšie) ako inštrukcie sčítania a odčítania a tie sú zase rýchlejšie ako násobenie a delenie[5].

Môžeme to riešiť tak, že zavedieme jednotný čas trvania inštrukcií- predĺžime čas vykonávania inštrukcií na čas potrebný pre vykonanie najpomalšej inštrukcie. Tento spôsob je však výhodný len pri jednoduchých procesoroch s malou inštrukčnou sadou.

Dĺžka trvania inštrukcie sa meria v *taktoch*. Každý takt trvá presne určenú jednotku času, ktorá závisí od taktovacej frekvencie procesora.

1.4 Množina registrov (Register Set)

Register sme už opisovali (v časti o obvodoch). Je to rýchla nízkokapacitná pamäť umožňujúca uchovať n bitov. Rôzne registre môžu mať rozličnú veľkosť[6] a môžu sa konštruovať rozličným spôsobom (registre s paralelným čítaním a zápisom, posuvné registre).

Súbor všetkých registrov sa nazýva *množina* (alebo *sada*) *registrov* (*register set*).

Podľa spôsobu využitia sa registre delia na *všeobecné* a *špeciálne*. Všeobecné sa používajú na rôzne účely (ukladanie operandov, výsledkov), špeciálne slúžia väčšinou na jeden vyhradený účel.

Hoci každý procesor používa špecifické registre so špecifickými menami, niektoré špeciálne registre sa používajú vo väčšine počítačov.

Uveďieme ich, pričom na ich pomenovanie použijeme zaužívané označenie:

- *Program Counter Register (PC)*

  Počítadlo. Obsahuje adresu nasledujúcej inštrukcie, ktorá sa má vykonať. Po vykonaní inštrukcie sa inkrementuje, aby ukazoval na nasledujúcu inštrukciu[7]. Pre vykonanie skokov je potrebné, aby sa doň dalo vložiť číslo (skok na adresu adr sa realizuje priradením PC:=adr).

[5] prirodzene, toto tvrdenie platí len pre jednoducho navrhnuté obvody realizujúce aritmetiku. Zložitým obvodom je aj násobenie možné vykonať v jednom kroku. Vzletne povedané, pre každú operáciu možno navrhnúť (zložitý) obvod, ktorý ju zrealizuje v jednom kroku. Pre jeho realizáciu však môžeme potrebovať také veľké množstvo hradiel, že jeho realizácia sa stane nemožnou. Preto sa v praxi konštruujú jednoduchšie obvody, realizujúce operácie na viac krokov; pri čom sa už prejavia rozdiely medzi jednotlivými typmi operácií.

[6] t.j. rôzne hodnoty n

[7] resp. obsahoval adresu nasledujúcej inštrukcie

- *Instruction Register (IR)*

  Obsahuje (operačný) kód práve vykonávanej inštrukcie. Po začatí vykonávania novej inštrukcie sa prekopíruje obsah pamäťovej bunky na ktorú ukazuje register PC do registra IR.

- *Memory adress register (MAR)*

  Memory buffer register (MBR)

  Registre pre prácu s pamäťou. Do registra MAR sa ukladá adresa pamäťového miesta, ktorého hodnota sa uloží do registra MBR; alebo naopak, hodnota z MBR sa uloží do daného pamäťového miesta.

  Počítač môže mať aj viac registrov slúžiacich na prácu s pamäťou.

- *Akumulátor (A)*

  Väčšinou (pokiaľ nie je určené inak) sa doň ukladajú výsledky aritmetických operácií.

  Staršie počítače mali aritmetické operácie silno viazané na register A: pokiaľ mala inštrukcia 2 operandy, jeden z nich bol register A. Navyše všetky výsledky sa ukladali do registra A. To zjednodušovalo konštrukciu príslušných obvodov pre realizáciu aritmetiky, ktorých súčasťou by inak museli byť selekčné obvody vyberajúce spomedzi viacerých možných argumentov (napr. množiny registrov) jeden. Na druhej strane, je často nutné vymieňať dáta medzi akumulátorom a ostatnými registrami.

  Moderné mikroprocesory obsahujúce veľkú sadu inštrukcií často umožňujú používať ako operandy inštrukcie ľubovoľné kombinácie všeobecných registrov a nie sú viazané len na register A.

- *Flag registers (F)*

  Register obsahujúci príznaky. Príznaky (*flag-y* detekujú rôzne udalosti, ktoré vznikli počas behu programu. Na ich základe dochádza k vetveniu programu.

  Aritmetické príznaky (informujú o výsledku naposledy vykonanej aritmetickej operácie):

  - *CF (Carry Flag)*
    príznak pretečenia neznamienkových čísel
  - *OF (Overlow Flag), AC*
    pretečenie znamienkových čísel
  - *SF (Sign Flag)*
    znamienko čísla (kópia najvyššieho bitu čísla)
  - *ZF (Zero Flag)*
    príznak nuly (vysledok je nulový)
  - *AC (Auxiliary Carry Flag)*
    pretečenie pri číslach BCD
  - *PF (Parity Flag)*
    parita výsledku

Špeciálny význam môžu mať aj kombinácie príznakov, napríklad po vykonaní porovnávania dvoch čísel A,B je výsledkom kombinácia príznakov C a Z:

C	Z	význam
0	1	$A = B$
1	0	$A < B$
0	0	$A > B$

Tabuľka 1.1: Zápis výsledku porovnávania čísel A,B pomocou príznakov C a Z

- *Program Status Word (PSW)*

  Stav vykonávania programu. Obsahuje riadiace príznaky: stav vykonávania programu, detekcia prerušení, vnútorné stavy procesora.

- *Stack Pointer (SP)*

  Vrchol zásobníka. *Zásobník* je pamäťová dátová štruktúra používaná k dočasnému ukladaniu dát. Nachádza sa v operačnej pamäti. Pracuje sa s ňou pomocou *stack pointera*, ktorý ukazuje na *vrchol zásobníka*, čo je aktuálne miesto s ktorým sa pracuje. Bližšie údaje o zásobníku čitateľ nájde v časti o pamätiach.

1.5 Metódy adresácie argumentov

Určujú, ako má počítač interpretovať adresovú časť inštrukcie, teda akým spôsobom má získať operandy pre danú operáciu.

Operandom inštrukcie môže byť konštanta, obsah registra alebo obsah pamäťového miesta. Možnosti ich konkrétneho získania, spôsoby adresácie, sú však značne rozmanité:

spôsob adresácie	spôsob získania operandu inštrukcie
implicitný	argument je implicitne známy
konštantný	argumentom je zadaná konštanta
priamy	z pamäťového miesta, z M[ADR]
nepriamy	nepriamo z pamäť. miesta M[M[ADR]]
register	argumentom je obsah registra
register nepriamy	z pamäť.miesta M[Reg]
inkrementačný	ako predch., ale navyše sa Reg inkrementne
relatívny	M[Adr+Reg]
indexový	M[IX+Adr]
bázový	M[BS+Adr]
bázový + indexový	M[IX+BS+Adr]
zreťazenie	zoberieme dve 8 bitové adresy a vytvoríme z nej 16-bitovú
block adressing	adresácia vzhľadom na blok

Tabuľka 1.2: Rôzne metódy adresácie argumentov

implicitný

Argumenty sú dopredu známe, definuje ich operačný kód.
Príklady využitia:

- pevne daný argument inštrukcie (napríklad aritmetické inštrukcie s povinným argumentom - akumulátorom)
- skok na pevne danú adresu so špeciálnym významom
  (napríklad skok na adresu, ktorá spôsobí RESET počítača)
- čítanie z pevne danej adresy, na ktorej môže byť pevne uložený napríklad vrchol zásobníka.

konštantný

Argumentom je konštanta (uvedená v adresovej časti inštrukcie).

register

Argumentom je obsah registra (špecifikovaného v adresovom poli).

adresovanie pamäte

Argumentom je pamäťová bunka, ktorej adresa je zadaná:

- *priamo* (konštantou)
- *nepriamo* (obsahom registra)

zložitejšie metódy adresovania pamäte

Uvedieme prehľad zložitejších metód adresácie s popisom, ako jednotlivé metódy vyrátajú adresu pamäťovej bunky, ktorá má byť operandom:

- *relatívny*: (k adrese danej inštrukcie –obsah PC– sa priráta obsah registra REG)

$$ADR := PC + REG$$

  Inou možnosťou je prirátavať obsah registra Reg k absolútne zadanej adrese

$$ADR := Konštanta + REG$$

- *bázový* : (k obsahu tzv. *bázového registra* sa priráta register REG)

$$ADR := BS + REG$$

- *indexový*: (rovnaký princíp ako bázový)

$$ADR := IX + REG$$

- *bázový+indexový* :

$$ADR := BS + IX + REG$$

Uvedené spôsoby adresácie sa môžu vyskytovať aj v tvare, keď je namiesto registra REG uvedená konštanta.

Využitie: Práca s údajmi, poľom. Pri relatívnych spôsoboch adresácie inštrukcia neobsahuje absolútne adresy dát. Adresová časť inštrukcie sa interpretuje spôsobom: 'čítaj dáta ktoré sa nachádzajú *sto* pamäťových miest *odtiaľto*'. Vďaka tomu možno vytvoriť program s relatívnou adresáciou. Zápis programu je jednoduchší a prehľadnejší a hlavne je univerzálnejší- funguje aj keď je uložený od ľubovolnej adresy v pamäti; čo o programe s absolútnou adresáciou neplatí.

zreťazenie

Doteraz sme používali vo vytváraní variánt adresových metód operáciu sčítania, keď sme k jednej adrese prirátavali druhú. Inou možnou metódou je zreťazenie. Majme n-bitové registre B a C, 'pripojme' register B k registru C, čím sa vytvorí adresa dĺžky $2n$. Vidno rozdiel medzi súčtom a zreťazením:

súčet: ADR:= B + C
zreťazenie: ADR:=(B)(C)

Príklad III.1:

Nech

$$B = 01101001$$
$$\underline{C = 00100110}$$

potom

$$B + C = 10001111$$
$$(B)(C) = 0110100100100110$$

iné varianty adresovania

Existuje veľké množstvo ďalších spôsobov adresovania– pre nejakú aplikáciu sa môže zmysluplným ukázať určitý (doteraz nepoužívaný) spôsob adresácie. Preto sme sa nesnažili podať vyčerpávajúci prehľad čo najväčšieho počtu metód adresovania; skôr malo zmysel uviesť tie najpoužívanejšie.

Ako príklad ďalších metód uveďme:

autoinkrement:

$$ARGUMENT := MEM[REG], REG := REG + 1$$

autodekrement:

$$ARGUMENT := MEM[REG], REG := REG - 1$$

Autoinkrement je rozšírením nepriameho adresovania, navyše po vykonaní operácie je register inkrementovaný o dĺžku slova. Tým sa dosiahne, že ukazuje na nasledujúce slovo. Analogicky autodekrement.

Využitie: pri cyklických a blokových operáciách, operáciách so zásobníkom.

blokové adresovanie

Využíva adresu na určenie pozície prvého znaku v bloku údajov.

Blok údajov má buď *pevnú dĺžku* (implicitne určenú), alebo má *premenlivú dĺžku*. Vtedy možno ohraničiť blok (určiť začiatok a koniec bloku) tromi spôsobmi, líšiacimi sa spôsobom určenia konca bloku[8]:

- adresou posledného slova
- dĺžkou bloku
- špeciálnym znakom detekujúcim koniec

Využitie: hromadný blokový prenos údajov zo zariadení ako pevný disk, dierna páska...

1.6 Prerušenia

Jednou z významných vlastností procesora je možnosť prerušiť prebiehajúcu činnosť a začať vykonávať inú a potom sa vrátiť k pôvodnej činnosti.

Prerušenie nastáva, keď chce niektoré zariadenie (či už vonkajšie alebo vnútorné) 'prinútiť' procesor, aby sa ním zapodieval (napríklad tlačiareň, ktorá potrebuje oznámiť, že sa počas tlačenia minul papier).

Zariadenie vyšle *žiadosť o prerušenie*. Po prijatí žiadosti procesor preruší vykonávanie programu (zapamätá si svoj vnútorný stav– ktorú akciu vykonával a aké boli v tom okamihu obsahy registrov) a začne vykonávať špeciálny program pre obsluhu prerušenia (riešiaci situácie, ktoré viedli k žiadosti o prerušenie). Po jeho skončení sa procesor vráti k vykonávaniu pôvodného programu (obnovia sa obsahy všetkých registrov a vykonávanie pôvodného programu pokračuje od miesta prerušenia).

Procesor jednoznačne identifikuje žiadateľa– každému pripojenému zariadeniu pridelí číslo (napr. v rozsahu 0..255). Z programátorského pohľadu sa to javí tak, akoby procesor dostal žiadosť o prerušenie s týmto číslom[9]. Tým možno odlíšiť jednotlivých žiadateľov, napr. tlačiareň bude 'posielať' žiadosti s číslom 10 a klávesnica s číslom 20. Procesor má tabuľku v (bežnej) pamäti, v ktorej má pre každý typ žiadostí (resp. pre každé číslo prerušenia) uloženú adresu obslužného programu. Týmto spôsobom možno ľahko pre každé zariadenie napísať tzv. *obslužný* program, riešiaci 'mimoriadne' situácie, ktoré viedli k prerušeniu.

Ako sme spomenuli, nemusí sa jednať len o prerušenie vyvolané vonkajším programom. Napríklad procesory Intel 80x86 vyvolajú prerušenie s číslom 0, pokiaľ sa vykonávaný program pokúšal deliť nulou. Každý proces si môže zaviesť vlastnú procedúru na ošetrenie tejto situácie, napríklad vypísanie varovného hlásenia.

[8] začiatok bloku určujú rovnako- udaním adresy prvého slova v bloku

[9] t.j. akoby toto číslo bolo súčasťou žiadosti o prerušenie

Každé zariadenie má svoju dôležitosť (prioritu) a pokiaľ požiada o prerušenie súčastne niekoľko zariadení naraz, vyberie sa to s najvyššou prioritou.

Prerušenie môže vyvolať aj programátor (t.j. môže byť vyvolané aj softwarovo).

Prerušenia sa delia na *maskovateľné* a *nemaskovateľné*. Maskovateľné sa dajú zamaskovať - t.j. možno ich programovo zakázať a potom ich procesor ignoruje. Nemaskovateľné sa zakázať nedajú.

Zrekapitulujme si činnosť procesora pri detekcii prerušenia:

1. uschová aktuálny stav počítadla inštrukcií (Program-Counter registra) a ostatných registrov (napríklad do pamäte, alebo zásobníka-viď časť o pamätiach)
2. z viacerých žiadostí o prerušenie určí to s najväčšou prioritou
3. zistí adresu obslužného programu a
4. odovzdá mu riadenie
5. po ukončení obslužného programu sa vráti k pôvodnej činnosti (obnoví Program Counter a stav ostatných registrov)

Kapitola 2

Princípy realizácie procesora

2.1 Princíp vykonávania inštrukcií

Úlohou univerzálneho procesora je interpretovať inštrukcie programu uloženého v hlavnej pamäti. Pri tejto činnosti sa jednotlivé inštrukcie postupne vyberajú z hlavnej pamäte, (kde sú uložené binárne zakódované), dekódujú a vykonávajú sa požadované operácie s požadovanými operandami.

Výsledkom interpretácie inštrukcie je teda nejaká *strojová operácia*. Tá však môže byť zložitá na to, aby sa fyzicky vykonala v jednom kroku (napríklad delenie), a preto sa vykoná postupnosť viacerých čiastkových elementárnych operácií– *mikrooperácií*, tak jednoduchých, aby sa už dali vykonať fyzicky, hardwarovo. Inštrukciu potom chápeme ako pomenovanie určitého mikroprogramu, resp. operáciu chápeme ako postupnosť mikrooperácií, ktorej vykonanie trvá istý počet taktov.

Postupnosť elementárnych krokov, počas ktorých sa vykoná operácia definovaná inštrukciou programu sa nazýva *inštrukčný cyklus*.

Skladá sa z viacerých fáz:

- *fetch cycle* (zahrňujúci tiež *address* a *translation cycle*)
- *execute cycle*
- *interrupt cycle*

Fetch cyklus: procesor získa inštrukciu z pamäti, dekóduje ju a určí adresu operandov. Táto časť je rovnaká pri všetkých inštrukciách.

Execute cyklus: procesor získa operandy, vykoná inštrukciu a zapíše výsledok na určené pamäťové miesto (register, hlavná pamäť).

Interrupt cyklus: dôjde k prerušenie vykonávania programu. Tento cyklus nastane len ak procesor zaznamenal požiadavku na prerušenie.

Každá fáza inštrukčného cyklu sa v závislosti od typu inštrukcie vykoná počas jedného alebo viacerých strojových cyklov, v rámci ktorých sa uskutočňuje iba jeden prístup do pamäte. Vo všeobecnosti je *dĺžka strojových cyklov* n-násobkom periódy *hodinových impulzov*, tzv. strojových taktov (resp. *hodinových cyklov*). *Dĺžka strojového taktu* sa definuje *frekvenciou hodinových impulzov*, ktoré z centrálneho zdroja (strojové hodiny)

taktujú činnosť procesora. *Dĺžku inštrukčného cyklu*, t.j. čas vykonania jednej inštrukcie určuje potom frekvencia hodinových impulzov.

V súčasných architektúrach sa inštrukčný cyklus zväčša rozloží na väčší počet fáz, čo umožňuje zvýšiť jeho výkonnosť pomocou ich vzájomného prekrývania (*zreťazenie–pipelining*) pri vykonávaní inštrukcií programu. Napríklad pri päťfázových inštrukčných cykloch môžu byť definované fázy:

1. čítanie inštrukcie
2. dekódovanie inštrukcie
3. čítanie operandu
4. vykonanie inštrukcie
5. zápis výsledku

Prekrytie znamená, že po vykonávaní štvrtej fázy inštrukcie môže paralelne prebiehať vykonávanie prvej fázy pre nasledujúcu inštrukciu. Podrobnejšie túto techniku opíšeme v IV.časti.

Príklad III.2: Uvažujme jednoduchý procesor. Rozlišujme tri fázy vykonávania inštrukcie: fetch, execute, interrupt. Nech hodiny (Clock) generujú signály t_0, t_1, t_2, atď ... Ďalej predpokladajme, že procesor má dva príznakové bity E a F určujúce v ktorom cykle sa (procesor) nachádza:

F	**E**	Význam:
0	0	interrupt cyklus
0	1	execute cyklus
1	0	fetch cyklus
1	1	(nepoužité)

Tabuľka 2.1: Kódovanie cyklov pomocou príznakov F a E

Predpokladajme formát inštrukcie $< FI >=< OP >< IX >< ad >$, kde OP je operačný kód, ad je adresa argumentu a bit IX hovorí, či k tejto adrese prirátať obsah indexového registra IX.

Uveďme výpisy mikroprogramov jednotlivých fáz. Vo výpise programov je používaná štandardná terminológia:

A:=(B)	do registra A zapíš obsah registra B
A:=MEM[adr]	do registra A zapíš obsah pamäťovej bunky s číslom *adr*

Fetch cyklus:

t_0	$F\bar{E}$	MAR:=(PC)
t_1	$F\bar{E}$	MBR:=(M[MAR]), PC:=PC+1
t_2	$F\bar{E}$	IR:=(MBR)
t_3	$F\bar{E}$	If IR[IX]=0 then MAR:=(IR[ad])

t_4 $F\bar{E}$ If IR[IX]=1 then MAR:=(IR[ad] + (IX))
t_5 $F\bar{E}$ F:=0, E:=1

Najprv sa do registra MAR uloží adresa pamäťového miesta, kde sa nachádza ďalšia inštrukcia (je daná hodnotou registra PC). V druhom kroku sa do MBR uloží obsah tejto adresy a obsah PC sa zvýši o 1. Potom sa do registra inštrukcií (IR) uloží obsah MBR (t.j. kód práve spracovávaniej inštrukcie). Podľa hodnoty príznaku IX sa nastaví adresa z ktorej sa získa operand.

Execute cyklus

Tento cyklus je prirodzene pre každú inštrukciu iný. Ako príklad uvedieme inštrukciu ADD X (ktorá prečíta k akumulátoru A operand X):

t_0 $E\bar{F}$ MBR:=(M[MAR])
t_1 $E\bar{F}$ A:= (A) + (MBR)
t_2 $E\bar{F}$ NOP (no operation)[1]
t_3 $E\bar{F}$ If INT=0 then F:=1, E:=0
else F:=0, E:=0

V prvých dvoch taktoch sa vykoná sčítanie, v t_2 sa nerobí nič, a v t_3 ak nie je požiadavka na prerušenie, tak sa prejde opäť do ďalšieho cyklu.

2.2 Aritmeticko- Logická jednotka (ALU)

Aritmeticko-logická jednotka je základným prvkom operačnej časti procesora a je určená na vykonávanie *operácií nad strojovými slovami*, ktorými sú v tvare binárnych vektorov zobrazené údaje, príkazy, prípadne iné zakódované objekty (stavové slová, príznaky), zúčastňujúce sa na procese programového spracovávania informácií.

Škála operácií a číselných formátov je rôzna a závisí od konkrétneho konštrukčného prevedenia ALU.

V ALU je spravidla možné realizovať nasledovné operácie:

- sčítanie a odčítanie
- násobenie, delenie
- logické operácie
- posuvy a rotácie (obsahov registrov)
- porovnania (obsahov registrov)

a to v rôznych číselných formátoch, ako napr.:

- neznamienkové i znamienkové celé čísla
- BCD formát

- reálne čísla (v pevnej alebo pohyblivej rádovej čiarke)

Informácie o vzniku mimoriadnych situácií pri výpočtoch poskytujú príznaky.

Základ ALU tvorí obyčajná sčítačka (viď popis ALU v časti V) upravená tak, že pomocou riadiacich signálov vykonáva aj určitý počet ďalších operácií (*mikrooperácií*). Okrem riadiacich signálov je ešte nutné na vstup sčítačky priviesť operandy. Na výstupe sa objaví *výsledok operácie* a *príznaky* popisujúce výsledok.

Ako skonštruovať obvody realizujúce jednotlivé operácie ALU? V časti o obvodoch (časť II) sme spomenuli princípy realizácie týchto obvodov. Kvôli prehľadu tieto informácie zhrňme - uveďme jednotlivé druhy operácií ALU, princípy ich realizácie a najvýznamnejšie problémy s tým spojené:

1. *sčítanie a odčítanie* - na realizáciu potrebujeme sčítačku. Pomocou nej sa však dá realizovať aj množstvo ďalších funkcií: (vstupy sčítačky sú X,Y,Z)

X	**Y**	**Z**	**mikrooperácia**
A	B	0	$S := A + B$
A	B	1	$S := A + B + 1$
A	B'	0	$S := A + B'$
A	B'	1	$S := A + B' + 1$ (čo je A-B)
A'	B	1	$S := A' + B + 1$)(čo je B-A)
A	1	0	$A - 1$ (DEC A)
1	B	0	$B - 1$
A	0	0	A
A	0	1	$A + 1$ (INC A)

Tabuľka 2.2: Operácie a kódy operácií 'jednoduchej' ALU

2. *convert element*

   Obvod upravujúci vstupný argument (C) na základe riadiacich signálov (S_0,S_1) podľa nasledovnej tabuľky:

S_0	S_1	**mikrooperácia**
0	0	C
0	1	$\bar{C}$
1	0	0
1	1	$\bar{0}$

Tabuľka 2.3: Operácie convert-elementu

3. *násobenie a delenie*

   Sú časovo náročné operácie. Možno ich realizovať viacerými spôsobmi:

   (a) hardwarovo

      i. všetky inštrukcie budú mať rovnaký čas trvania, teda ostatné inštrukcie spomalíme, aby násobenie 'nepredbehli'

      ii. inštrukcie nebudú trvať rovnaký čas

(b) mikroprogramom (využívajúcim posuny, sčítanie a odčítanie)

(c) matematickým koprocesorom

4. *logické operácie ALU*

   Sú rýchle, jednoducho realizovateľné (pomocou základných hradiel). Patrí sem AND, OR, XOR,[2] NOT a ďalšie.

5. *posuny*

   - Patria sem funkcie posunutia a rotácie.

   - Aby nebolo treba použiť posuvné registre, používa sa obvod nazývaný *position scaler*, ktorý umožňuje posúvanie argumentov rôznymi spôsobmi. Má nasledovné funkcie:

   - ponechaj argument bez zmeny
   - posuň ho vľavo
   - posuň ho vpravo
   - rotuj argument vľavo, vpravo
   - prípadne ďalšie

   Výhodné je mať ho mimo ALU, (na vstup ALU privádzať jeho výsledok), čím je možné v jednom takte posunúť operand a vykonať operáciu.

2.3 Control logic unit (CLU)

Riadi vykonanie mikroprogramu (zodpovedajúcemu určitej inštrukcii)– pre každú mikroinštrukciu vygeneruje riadiace signály pre príslušné obvody, ktoré majú mikroinštrukciu realizovať. Okrem toho CPU obsluhuje prerušenia.

CLU je konečný stavový automat. Jeho základné funkčné jednotky sú:

1. *Riadiaca jednotka*, určená na generovanie vnútorných (procesorových) signálov a na vyhodnotenie stavovo-informačných signálov o procesoch prebiehajúcich v počítači.

   Môže sa riešiť *pevnou* alebo *mikroprogramovateľnou* logikou (viď ďalej), prípadne sa oba prístupy skombinujú so snahou dosiahnuť čo najväčšiu efektivitu.

2. *Synchronizačná riadiaca jednotka* je určená na časovanie jednotlivých činností procesora a celého počítačového systému. Jej základným prvkom je *generátor hodinových impulzov*.

3. *Inštrukčná jednotka* (jednotka predvýberu) je určená na výber, resp. predvýber inštrukcií, ich dekódovanie a prípravu na vykonanie príslušnej operácie.

[2] argumentami týchto operácií sú dva vektory privedené na vstup ALU. Najčastejšie to sú dva registre, register a konštantný vektor, alebo register a obsah pamäťového miesta. Prirodzene, z technického hľadiska nie sú registre či pamäť pripojené priamo k ALU– ako sprostredkovateľ sa využíva *zbernica*, ktorú opíšeme neskôr.

4. *Radič požiadaviek prerušenia* je určený na výber a spracovanie požiadaviek o prerušenie. Vykonáva niektoré činnosti, ako napríklad výber z viacerých požiadaviek o prerušenie to s najväčšou prioritou.

Pozrime sa teraz na obe časti počítača (ALU aj CLU, resp. *AP* a *IP*) a určime, ktoré z inštrukčných cyklov vykonáva ktorá časť:

- *AP (aritmetický procesor)* : riadi Execute cyklus
- *IP (inštrukčný procesor)* : riadi Fetch, Adress, Interrupt

Naraz pracuje len jeden z nich (pri obvyklých systémoch). Na ich rozlíšenie zavedieme ďalšiu premennú I,

I=0 značí získavanie inštrukcie
I=1 značí vykonávanie

2.3.1 Realizácia CLU

Napriek popisu funkcií CLU, ktorý mohol vzbudiť u čitateľa zanechať dojem zložitosti, je realizácia CLU pomerne jednoduchá. CLU sa skladá z dekódera a niekoľkých riadiacich obvodov. Dekóder dekóduje inštrukciu. Na jej vykonanie treba vytvoriť a vykonať postupnosť mikrooperácií. Generovanie nie je ťažké, technicky vieme realizovať obvod generujúci v pravidelných intervaloch impulzy. Vykonanie mikro-operácie je založené na nasledovnom princípe:

Stav všetkých obvodov (výkonných, I/O obvodov a iných) je možné určiť. Každý obvod má pripojený vstup aj výstup na dátovú zbernicu. Podstatnou úlohou je zabezpečiť vzájomnú komunikáciu medzi obvodmi. Prinicipiálne možno riešenie popísať nasledovne: nech sú prepojené všetky obvody. Každému spoju priradíme 1 bit N-bitového registra (kde N je počet spojov). Ak je tento bit nula, spoj je neaktívny, ak je jedna, spoj je aktívny. Takýmto spôsobom možno každý mikroprogram zapísať ako postupnosť riadiacich impulzov.

Príklad III.3: Predpokladajme, že inštrukcie sú realizovateľné najviac na 6 mikrokrokov. Uvažujme generátor generujúci signál každých 10ns. Každých 10ns vykonáme jednu mikro-inštrukciu, čiže aktivizujeme príslušné výkonné obvody. Po 60ns vykonáme ľubovoľnú inštrukciu nášho procesora.

Podľa spôsobu realizácie riadiacej jednotky rozoznávame *hardwarovú* a *mikroprogramovú* CLU.

V hardwarovo riešenej CLU je každý algoritmus inštrukcie riešený hardwarovo. V mikroprogramovej CLU sa pre každú inštrukciu vykoná mikroprogram zapísaný v trvalej pamäti, ktorá je súčasťou procesora.

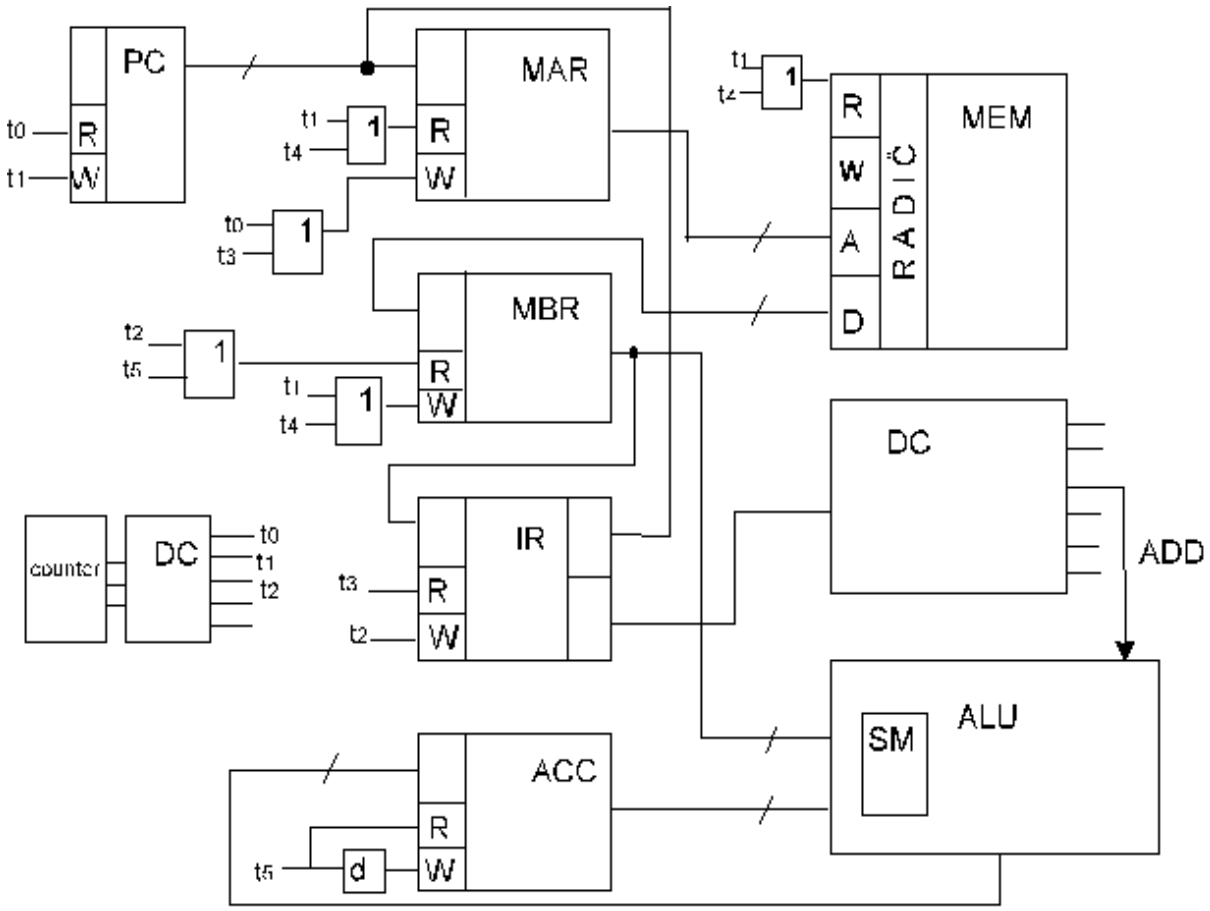
Hardwarová CLU má pevnú logiku, (hard wired - 'pevne zadrôtovaná').
Rozoznávame:

- *synchrónnu* - riadi sa procesorovými hodinami (všetky obvody musia stihnúť vykonať svoju prácu počas trvania príslušného hodinového signálu)
- *asynchrónnu* - po vykonaní operácie obvody vracajú CLU signál, že skončili (až po obdržaní všetkých signálov sa môže pokračovať)

Ako príklad zoberieme inštrukciu ADD X. Táto inštrukcia k akumulátoru priráta obsah pamäťového miesta určeného priamo adresou X. Uvedieme úplný mikroprogram, zahrňujúci operácie od získania argumentu až po uloženie výsledku:

ADD X
t_0 MAR ←(PC)
t_1 MBR ←M[MAR], PC ←(PC) + 1
t_2 IR ←(MBR)
t_3 MAR ←IR [adr]
t_4 MBR ←M[MAR]
t_5 ACC ←(ACC) + 1

Hardwarová realizácia je znázornená na obrázku 2.1.



Obrázok 2.1: Hardwarová CLU pre inštrukciu ADD X

Nevýhody hardwarového prístupu sú: modifikácia inštrukcie alebo zavedenie novej inštrukcie si vyžaduje nový návrh procesora, tiež tvorba veľkého inštrukčného súboru sa stáva problematickou. Výhodou je väčšia rýchlosť.

Alternatívou je mikroprogramovo realizovaná CLU.

2.4 Mikroprogramová CLU a mikroprogramovanie

2.4.1 Mikroprogramovanie

Hardwarovo riešená CLU má značné nevýhody: modifikácia inštrukcie alebo zavedenie novej inštrukcie si vyžaduje nový návrh, takisto problematické je ladenie a hľadanie chýb.[3]

Mikroprogramové riešenie sa objavuje začiatkom päťdesiatych rokov. Konštruktéri sa pokúšali rozdeliť problém vykonávania inštrukcií na viacero úrovní, nájsť kompromis pri optimálnom návrhu zložitého systému. Inštrukcia je zapísaná v pamäti pomocou mikroprogramu. Procesor obsahuje interpreter schopný vykonať ho.

Samotná realizácia však stroskotávala na tom, že v tej dobe neboli k dispozícií rýchle a lacné pamäte na ukladanie mikroprogramov. Až v roku 1964 sa objavuje počítač IBM 360, v ktorom bola použitá technika mikroprogramovania.

Princípy a účel mikroprogramovania sme už vlastne uviedli: inštrukcia sa nerealizuje hardwarovo, ale zapíše sa mikroprogramom. Mikroprogram obsahuje elementárne príkazy - mikroinštrukcie, ktoré sa už realizujú hardwarovo pomocou základných obvodov. Procesor obsahuje dekóder, ktorý dekóduje inštrukciu, pamäť, z ktorej vyberie príslušný mikroprogram a interpreter, ktorý pre každú mikroinštrukciu vygeneruje príslušné riadiace impulzy.

Mikroprogramovanie teda posúva hranicu medzi hardwarom a softwarom. Rozdeľuje proces vykonania inštrukcie na viacero jednoduchších procesov.

Vykonávanie mikroprogramu (mikroprogramové realizovanie inštrukcií) zabezpečuje mikroprogramová CLU.

2.4.2 Mikroprogramová CLU

Mikroprogramové CLU sa delia podľa toho, aká je možnosť používateľa zasahovať do mikroprogramov:

- nemenné mikroprogramy
- sú možné čiastočné zmeny mikroprogramov
- CLU môže používateľ úplne naprogramovať

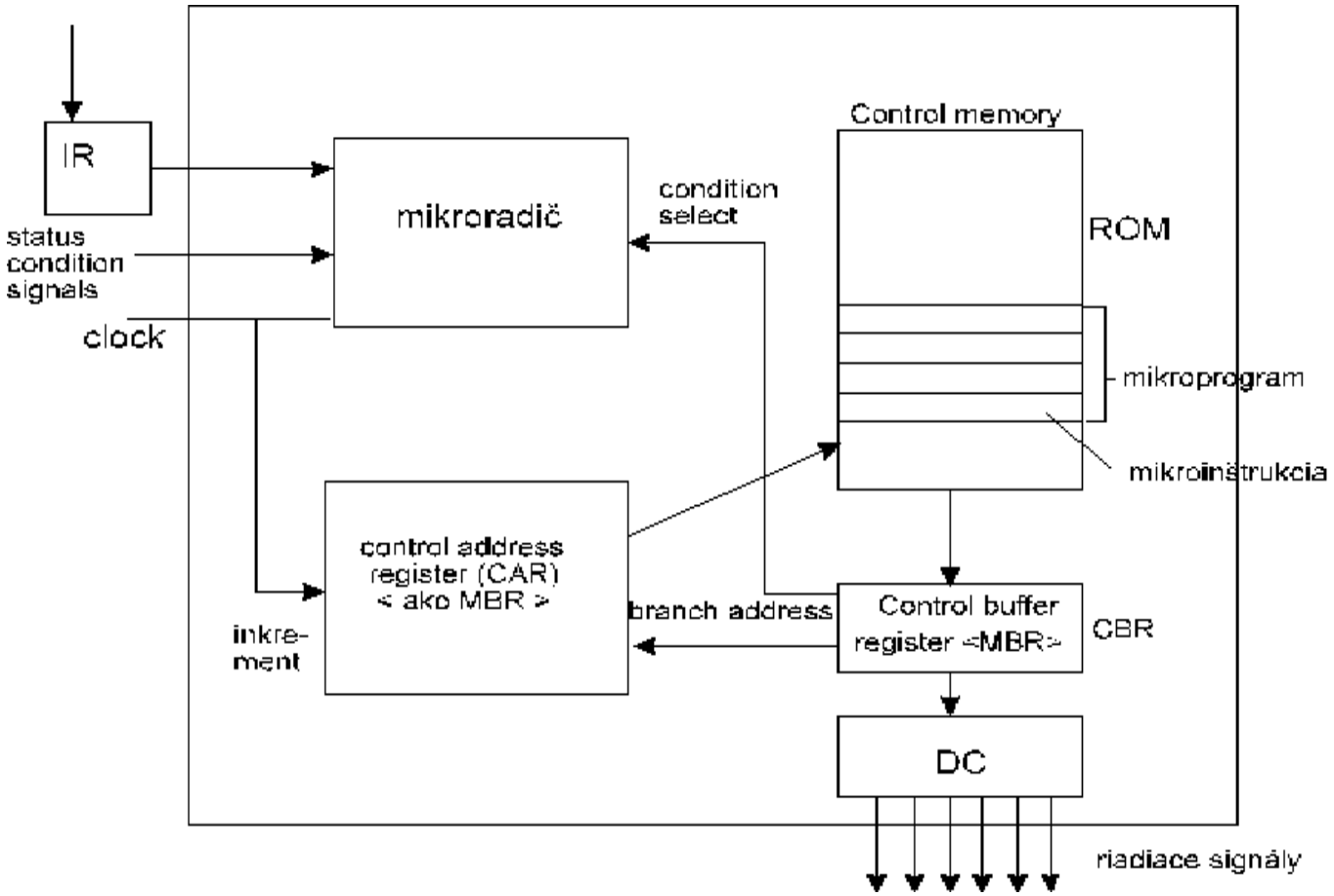
Mikroprogramová CLU je *mikroprogramovateľná*, ak používateľ môže naprogramovať vlastné mikroprogramy. Pre bežné aplikácie však nie je potrebné aby používateľ mal možnosť zasahovania do mikroprogramov.

Organizácia mikroprogramovej CLU vyzerá nasledovne (obr. 2.2).

IR	Instruction register
CAR	Control adress register
CBR	Control buffer register
DC	Decoder

Popíšeme stručne fungovanie mikroprogramovej CLU: v IR je uložená (makro) inštrukcia. Mikroprogramový radič určí príslušný mikroprogram (do CAR vloží jeho začiatočnú adresu) a vykonáva ho:

[3] čitateľ nech si skúsi predstaviť, ako by asi vyzerala CLU pre inštrukčný súbor 200 inštrukcií



Obrázok 2.2: Mikroprogramová CLU

1. do CAR sa uloží adresa mikroinštrukcie, ktorá sa má vykonať
2. obsah miesta mikro-programovej pamäte, ktorého adresa je v CAR sa uloží do CBR
3. začína sa mikrocyklus, počas ktorého CLU vygeneruje riadiace impulzy na vykonanie mikroinštrukcie.
4. CAR sa zvýši o 1 (ak sa nevyskytnú skoky) a cyklus vykonávania pokračuje od 2.kroku, až kým sa mikroprogram neukončí (mikroinštrukciou RET)

2.4.3 Jazyk RTL

Na formálny zápis mikro-programu sa používa jazyk *RTL* (*Register transfer language*). Umožňuje popísať elementárne operácie, ktoré sú súčasťou mikro-jazyka. Uvedieme syntax jazyka a prehľad zápisu jednotlivých typov operácií:

- registre sú označené menom (napr. MAR, PC,...)
- takisto môžeme označiť aj časti registrov (napr. AX[2..6])
- v zápise mikroprogramu je každá mikro-inštrukcia na samostatnom riadku
- za mikro-inštrukciou sa môže nachádzať aj komentár, ktorý sa oddeľuje bodkočiarkou

V jazyku RTL sú definované nasledujúce operácie:

1. vloženie konštanty do registra ($L \leftarrow 5$)
2. vloženie hodnoty iného registra do registra ($A \leftarrow (B)$)
3. vloženie časti registra do iného registra,
   možno popísať dvoma spôsobmi:
   (a) časť sme pomenovali ($PC \leftarrow IR[AD]$)
   (b) označili sme vkladané bity ($PC \leftarrow R[0..3]$)
   - analogicky možno vyjadriť vloženie jedného bitu
4. aritmetické a logické operácie
   (a) operácia sčítania:
   $$A3 \leftarrow (A1) + (A2)$$
   (b) ošetrenie pretečenia (použijeme jednobitový register C):
   $$C \leftarrow (A1) + (A2)$$
   (c) ďalšie aritmetické operácie:

$A \leftarrow (A) + 1$	; inkremenácia registra
$A \leftarrow (A) - 1$	; dekrementácia
$A \leftarrow (A')$	; jednotkový doplnok
$A \leftarrow (A') + 1$	; dvojkový doplnok
$A \leftarrow (A) + (B') + 1$	; odčítanie A-B

(d) logické operácie

$C \leftarrow (A)\ \ AND \quad (B)$
$C \leftarrow (A)\ \ OR \quad\ (B)$
$C \leftarrow NOT\ \ (A)$

(e) operácie posuvov

$A \leftarrow SL(A)$; posuv (o 1 bit) vľavo
$A \leftarrow SR(A)$; posun vpravo
$A \leftarrow RL(A)$; rotácia vľavo
$A \leftarrow RR(A)$; rotácia vpravo

Poznámka III.1: operácie násobenia a delenia už nie sú také jednoduché, nemožno ich realizovať v jednom takte. Preto sa medzi mikroinštrukcie nezaraďujú.

5. presun údajov medzi pamäťou a registrami:

   pamäťové miesto s adresou *adr* zapíšeme ako $M[adr]$. Potom:

   - čítanie z pamäti zapíšeme
     $$A \leftarrow (M[adr])$$
   - zápis do pamäte zapíšeme
     $$M[adr] \leftarrow (A)$$

6. vykonanie operácie ak sú splnené určité podmienky, (napr. pretečenie, parita...)

   if PODMIENKA *then* PRÍKAZ

   (alebo riadiace podmienky programu, ktoré vieme reprezentovať booleovými výrazmi).

Mikrojazyk môže obsahovať aj skoky, podmienené skoky alebo príkazy označujúce koniec mikro-programu. Adresu, kam sa má skákať vyjadríme pomocou návestia:

```
zac     A ← 0          ; začiatok cyklu
        ...
        ...
        ...
        JMP zac        ; skok na začiatok
```

Poznámka III.2: Čitateľovi je zaiste zrejmé, že možnosti zápisov sú dosť variabilné a závisí na programátorovi, ako dané operácie nazve (napr. posuv vľavo možno nazvať *SL, SHL, SHIFTL...*).

Tiež na programátorovi závisí, na akej úrovni zapíše svoj mikroprogram, napríklad či operáciu $A - B$ zapíše ako $A \leftarrow (A) - (B)$, alebo ako $A \leftarrow (A) + (B') + 1$. Podstatnými kritériami zápisu sú korektnosť, prehľadnosť a zrozumiteľnosť.

Príkladom mikroprogramu môže byť mikroprogram inštrukcie ADD X uvedený v kapitole 2.3.

2.4.4 Formáty mikroinštrukcií

Ukázali sme, ako možno symbolicky (formálne) zapísať mikroprogram. Pri konštrukcii by sme zrejme každej elementárnej operácií priradili kód a na zápis mikroinštrukcií by sme použili kódovaciu tabuľku. Aj tu však existujú dve možnosti zápisu, podľa toho aký formát mikroinštrukcie zvolíme.

Formát mikroinštrukcie môže byť horizontálny alebo vertikálny.

Horizontálny formát: binárny vektor, ktorý obsahuje toľko bitov, koľko je všetkých možných riadiacich signálov. Výhodou je, že sa dá vykonať viacero elementárnych operácií naraz. Nevýhodou je veľká dĺžka vektora, pritom vektor obsahuje veľa núl a málo jednotiek, lebo viacero operácií sa navzájom vylučuje.

Vertikálny formát: špecifikuje sa len jedna mikrooperácia. Tento spôsob vyžaduje zložitejší obvod, dekóder. Mikroinštrukcie majú kratší zápis.

Príklad III.4: : uvažujme ALU (argumenty X,Y, výsledok Z)
Nech má 3 operácie NOP (nič), $X+Y$, $X-Y$ s kódmi 00,01 a 10.
Chceme zapísať príkaz: $R_5 := R_3 - R_{11}$

- horizontálne: 10 0011 1011 0101
- vertikálne: (nech 00 značí výber X, 01 výber Y, 10 výber Z a 11 výber ALU).
  Mikroprogram zapíšeme:

  00 0011 (X je R_3; resp. R_3 naplní X)
  01 1011 (Y je R_{11})
  10 0101 (Z je R_5)
  11 0000 (ALU prevedie $Z := X + Y$)
  atď...

V tomto príklade je vertikálny zápis nevýhodnejší (väčšia dĺžka), to však nemusí vždy platiť.

Poznámka III.3: (k *podmienenýM skokom*):
V mikroprograme treba umožniť realizáciu podmienených skokov. Na zjednodušenie zavedieme taký formát, v ktorom podmienené skoky budú mať dve možné cieľové adresy: adresu nasledujúcu inštrukcie alebo adresu uvedenú v adresovom poli. Na ich vzájomné rozlíšenie (určenie cieľovej adresy) použijeme tzv. *condition field*.

Príklad III.5: (2bit cond field)

Cond. f	**význam**
00	skoč na nasledujúcu inštrukciu
01	skoč na adresu ak platí podmienka c_1
10	skoč na adresu ak platí podmienka c_2
11	skok na adresu

Symbolicky to vieme zapísať aj takto:

Cond. f	**význam**
00	CAR:=CAR+1
01	if c_1 then CAR:=ADR
10	if c_1 then CAR:=ADR
11	CAR:=ADRESS FIELD

2.4.5 Výhody a nevýhody mikroprogramovania

Nevýhodou mikroprogramovej CLU je menšia rýchlosť ako hardwarovej CLU, naopak pri mikroprogramovej CLU je možné inštrukčný súbor ľahko modifikovať a dopĺňať, dá sa vytvárať aj veľký inštrukčný súbor.

- *Výhody*:
  - štruktúrovaný návrh CLU
  - jednoduché ladenie, dopĺňanie, opravy
- *Nevýhody*:
  - pomalšia činnosť

V snahe odstránenia nevýhod vznikajú nové typy architektúr, napríklad počítače typu *RISC* (o ktorých pohovoríme v ďalšej časti).

2.4.6 Podporné prostriedky pre mikroprogramovanie

Mikroprogramovanie je dosť náročné, zložitejšie ako bežné programovanie. Na uľahčenie práce slúžia podporné prostriedky. Sem zahrňujeme hardwarové a softwarové prostriedky používané na uľahčenie mikroprogramovania.

- na vytváranie mikro-programu sa používa mikro-assembler
- na ladenie a opravu sa môžu použiť hardwarové simulátory
- veľmi silným prostriedkom sú tiež softwarové emulátory, ktoré umožňujú na jednom počítači simulovať iný počítač.
- vývojové systémy, umožňujú testovanie a editovanie
- pri výrobe alebo pri skúšaní sa používajú pamäte typu ROM, PROM a EPROM.

2.5 Zbernice

Jednotlivé časti počítača (napríklad procesor a pamäť) si medzi sebou musia vymieňať dáta. Prenos údajov zabezpečujú zbernice.

Zbernica je tvorená sústavou vodičov, ktoré spájajú jednotlivé časti počítača. Delí sa na tri časti: *adresovú , dátovú a riadiacu* zbernicu. Po *dátovej zbernici* sa prenášajú dáta. *Adresová zbernica* určuje, pre koho (pre ktoré zariadenie pripojené na zbernicu) sú dáta určené a po *riadiacej zbernici* sa prenášajú rôzne riadiace informácie.

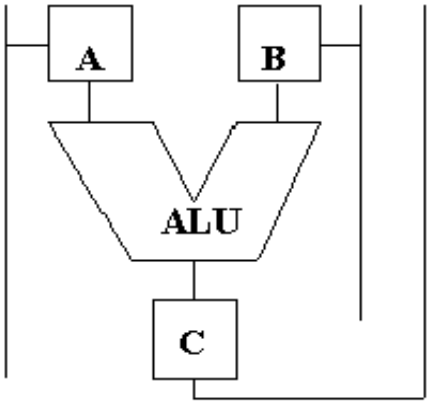
Určitou výnimkou z tejto schémy je pamäť. Ak procesor potrebuje zapísať do pamäte na adresu X hodnotu Y, tak na adresovú zbernicu zapíše adresu pamäťovej bunky– číslo X, na dátovú zbernicu zapíše Y a na riadiacu zbernicu pošle údaj, že sa má zapisovať do pamäte. Pamäť (resp. radič pamäte), prezrie obsah zbernice[4] a zistí, že sa prenáša údaj pre ňu (povel pre zápis), hodnota X a hodnota Y.

[4] presnejšie, vývody zbernice sú pripojené na vstup pamäťového obvodu

Čitateľovi je iste zrejmé, že v jednom okamihu sa po zbernici prenášajú len jedny dáta (zbernica je tvorená sústavou vodičov), teda nemôžeme zapísať dáta na zbernicu, pokiaľ tam už nejaké dáta zapísané sú. Preto, ak chce nejaké zariadenie zapisovať na zbernicu, tak si najskôr overí, či je zbernica 'voľná' (najčastejšie má zbernica riadiaci signál detekujúci, či je 'obsadená' alebo 'voľná'). Taktiež je zrejmé aj to, že dáta, ktoré posiela jedno zariadenie inému je prístupné aj všetkým ostatným zariadeniam pripojeným na zbernici. Každé zariadenie môže údaje čítať - a aj číta, a podľa časti z nich (určujúcej zariadenie, pre ktoré sú určené) ich buď ignoruje (dáta nie sú určené pre neho), alebo ich prečíta a spracuje (dáta sú určené práve pre neho).

Počítač nemusí mať len jednu zbernicu; počet zberníc CPU[5] výrazne ovplyvňuje jeho konfiguráciu.

- *Jednozbernicová organizácia*: máme k dispozícii jednu údajovú zbernicu. Na nej môže byť v jednom okamihu len jeden údaj. Ak napríklad ALU potrebuje dva údaje, tak jeden môže byť na zbernici a druhý sa musí zaviesť do nejakého registra (bufera).



Obrázok 2.3: Jednozbernicová architektúra

Konštrukcia je jednoduchá (obr. 2.3), spracovanie údajov je však zložitejšie. Najprv treba pripraviť operandy (v prvom kroku uložiť do buffera A prvý argument, potom v druhom kroku do buffera B druhý argument), potom vykonať operáciu (výsledok bude v bufri C) a nakoniec výsledok zo C uložiť do požadovaného registra.

- *Viaczbernicová organizácia*: Spracovanie údajov v predošlom prípade bolo dosť komplikované, napríklad príprava operandov sa musela diať v dvoch krokoch. Za cenu zložitejšej štruktúry sa dá uvedený postup zjednodušiť.

  Pri vhodnej štruktúre sa operácia $R_1 = (R_2) + (R_3)$ dá vykonať v jednom takte.

  Návrh vhodnej zbernicovej architektúry s takýmito vlastnosťami prenecháme na čitateľa.

[5]resp. počet zberníc, na ktorých je pripojené CPU

2.6 Parametre CPU

Zloženie, štruktúru a výkonnosť CPU možno charakterizovať nasledujúcimi údajmi:

1. *dĺžka slova*

   určuje, s dátami akého rozsahu je procesor schopný pracovať v jednom takte. Napríklad prvé počítače mali štvorbitovú dĺžku slova– operandy ich inštrukcií boli štvorbitové.

2. *dĺžka adresy*

   určuje aký veľký adresný priestor je počítač schopný adresovať. Napríklad 24 bitová adresa je schopná adresovať 2^{24} pamäťových miest[6].

3. *taktovacia frekvencia*

   je dĺžka trvania jedného taktu, udávaná v herzoch.

4. *počet taktov na vykonanie jednej inštrukcie*

   udáva čas potrebný na vykonanie jednej inštrukcie.

5. *inštrukčný súbor*

   možnosti inštrukčného súboru, rozmanitosť inštrukcií, priamo podmieňujú efektivitu programov.

6. *prítomnosť koprocesora*

   koprocesor umožňuje prevádzať rýchle výpočty s reálnymi číslami.

7. *sériové alebo paralelné spracovanie inštrukcií*

   počítače s paralelným spracovaním inštrukcií sú rýchlejšie a dokážu vykonať viacero inštrukcií v jednom takte.

[6] v prípade, že adresujeme byty to predstavuje pamäť rozsahu 1 megabajt

Časť IV

Zvyšovanie výkonu procesora

V predchádzajúcej časti sme hovorili o princípoch realizácie procesora. Dopracovali sme sa pritom k procesoru s pomerne bohatou sadou inštrukcií. Sústredili sme sa však len na navrhnutie procesora s požadovanou sadou registrov a inštrukcií, pričom nám vyhovovalo akékoľvek správne riešenie. Nezaujímalo nás hľadanie najefektívnejšieho riešenia - čiže navrhnutie čo 'najrýchlejšieho', resp. 'najvýkonnejšieho' procesora, t.j. procesora, na ktorom by programy bežali 'čo najrýchlejšie'. Táto problematika bude témou tejto časti.

Požiadavky trhu sú na čo najrýchlejšie procesory. Úloha dizajnérov, snažiacich sa vyhovieť požiadavkám trhu, však vôbec nie je ľahká. Značne obmedzujúcim faktorom je požiadavka tzv. *spätnej kombatibility* - programy vytvorené pre staršie procesory musia bežať aj na nových procesoroch. Znamená to, že nový procesor musí obsahovať aj všetky registre, inštrukcie a 'časť správania' svojich predchodcov. Dôvod požiadavky spätnej kombatibility je jednoduchý - nové procesory sa objavujú v priemere každých 9 mesiacov, je preto nemysliteľné, aby sa kvôli zmene procesora každých 9 mesiacov musel meniť software. Nový procesor preto zväčša 'naväzuje' na niekoľko starších procesorov, s ktorými je spätne kombatibilný.

Zvýšiť výkon procesora možno dvoma spôsobmi:

- *zvýšením taktovacej frekvencie*, resp. *použitím 'lepšej' technológie* (použitím nových elektronických súčiastok s 'lepšími' vlastnosťami), alebo
- *lepšou architektúrou procesora*

Vyššia taktovacia frekvencia

Spomenuli sme, že *taktovacia frekvencia* podmieňuje rýchlosť procesora - udáva, koľko cyklov vykoná procesor za jednu sekundu. Čím je vyššia, tým viac cyklov (čiže aj inštrukcií) sa stihne vykonať za sekundu, čiže procesor je rýchlejší.

Taktovacia frekvencia je podmienená použitými technológiami. Napríklad výrobná technológia podmieňuje vzdialenosť medzi jednotlivými komponentami, ktorá čím je menšia, tým je procesor rýchlejší. V nedávnej minulosti sa používala 0.35 mikrónová technológia, v súčastnosti sú používané technológie pod 0.25 mikróna. Iným príkladom sú pamäte - existuje viacero fyzikálnych princípov uchovania informácie. Použitý princíp však podmieňuje rýchlosť pamäte (napr. polovodičová pamäť je rýchlejšia ako magnetická).

V roku 1987 mali najrýchlejšie procesory taktovaciu frekvenciu 25 MHz, v súčastnosti je bežných 400 MHz. Nehovoriac o procesoroch pre 'náročnejších'.

Samozrejme, zvyšovanie taktovacej frekvencie sa nemôže uskutočňovať donekonečna. Existujú totiž fyzikálne obmedzenia, napr. rýchlosť šírenia elektrického prúdu alebo vzájomná interakcia dvoch blízkych spojov (vodičov), ktorými preteká prúd. Začíname sa približovať k hraniciam možností elektroniky a čoskoro na ne prudko narazíme. Jediným riešením bude zrejme konštruovať 'neelektronické' počítače, napr. na báze fotoniky (optické procesory). Výskumy týmto aj mnohými ďalšími smermi už prebiehajú.

Vylepšená architektúra

Vývoj nových technológií je zdĺhavý, časovo aj finančne náročný. Návrhár preto nemôže rátať s použitím nových, 'prevratných' technológií, ktoré by zaručili výrazné zvýšenie výkonu, no napriek tomu musí navrhnúť výkonnejší procesor. Ako dosiahnuť zlepšenie oproti 'predchádzajúcim procesorom', postavených na rovnakej technológií? *Lepším využitím tejto technológie*, čiže *lepšou architektúrou*.

Zmena architektúry môže predstavovať napr. zvýšenie počtu registrov, nové inštrukcie, podpora paralelného spracovania atď...

V prípade zvyšovania počtu registrov sa procesoru pridajú nové registre a inštrukcie pre prácu s nimi; pôvodnú aplikáciu treba preprogramovať, aby nové registre využívala. Čo spôsobí zvýšenie počtu registrov? Zvýši sa výkon, pretože aplikácia nemusí tak často pristupovať do hlavnej pamäte, ale načíta potrebné dáta do registrov. Pretože registre sú asi desaťkrát rýchlejšie ako hlavná pamäť, ušetrí sa veľa času.

V ďalšom texte prezentujeme najpoužívanejšie techniky techniky používané v moderných architektúrach:

- *pridanie nových inštrukcií*
- *paralelné spracovanie inštrukcií*
- *realizácia CLU hardvérovo*

pridanie nových inštrukcií – Príkladom je rozšírenie inštrukčnej sady procesora bez reálnej aritmetiky o inštrukcie reálnej aritmetiky. Kým doteraz sa reálna aritmetika musela realizovať softvérovo, programom, teraz je k dispozícií jej harvérová realizácia. To spôsobí rýchlejšie vykonanie týchto operácií, a pretože tieto operácie využíva veľké množstvo aplikácií, tak sa výrazne zvýši výkon. Samozrejme, spolu s novými operáciami je občas vhodné pridať aj nové registre (napr. procesory Pentium majú špeciálne registre určené pre reálnu aritmetiku).

Čiže: procesoru pridáme inštrukcie realizujúce často vykonávané operácie, ktoré doteraz hardwarovo realizované neboli a preto sa museli realizovať softwarovo - čím sa zbytočne strácal čas.

Príkladom použitia tejto metódy je technológia MMX, ktorú opíšeme v nasledujúcej kapitole.

paralelné spracovanie inštrukcií – Súčasné procesory sa snažia počas jedného cyklu paralelne vykonávať viac inštrukcií, čím sa výrazne zvyšuje rýchlosť vykonávania programu. Táto úloha však nie je vôbec jednoduchá, pretože veľa inštrukcií je na sebe závislých a v programe sa vyskytuje veľa vetvení, čo bráni paralelizmu pri vykonávaní inštrukcií. Bolo však vyvinutých veľa techník na prekonávanie týchto problémov. V druhej kapitole sa budeme zaoberať novými technológiami a algoritmami, ktoré umožňujú vykonávať viac inštrukcií v jednom cykle a ktoré posunuli vývoj v oblasti procesorovej architektúry dopredu.

realizácia CLU hardwarovo – Predstavuje prístup do určitej miery 'duálny' k prístupu 'bohatej' inštrukčnej sady. Výrok 'menej môže byť viac' je mottom tohto prístupu, demonštrovaného v poslednej kapitole tejto časti, venujúcej sa RISC-procesorom.

Kapitola 1

MMX

S rozvojom informačných technológií sa v aplikáciách, hrách či komunikácií začali objavovať grafické prvky. Dnes je už nemožné predstaviť si aplikácie bez obrázkov, videa, 3D grafiky, animácií či zvukov. Kladie sa tým väčší výkon na procesor; vyžaduje sa spracovávanie veľkého objemu dát.

Inou cestou, ako zvýšiť výkon procesora 'klasickou' cestou - zvýšením taktovacej frekvencie bola MMX technológia. Technológia MMX bola vyvinutá špeciálne pre zrýchlenie multimediálnych aplikácií.

Analyzovali sa multimediálne aplikácie, presnejšie ich algoritmy z oblastí grafiky, MPEG videa, hudobnej syntézy a kompresie, rozopoznávania hlasu, spracovania obrazu, hier, videokonferencií a iných. Hľadali sa výpočtovo najnáročnejšie rutiny, ktoré boli potom detailne analyzované. Ukázalo sa, že multimediálne aplikácie, hoci nie sú z tej istej 'oblasti' (ako napr. spracovanie zvuku a sporacovanie obrazu), majú viaceré spoločné črty:

- práca s rozsahovo malými celočíselnými dátovými typmi (napr. 8-bitové grafické pixely, 16-bitové audio vzorky)
- krátke cykly, ktoré sa často opakujú (často sa na neveľkom bloku dát vykonajú pre všetky prvky bloku tú istú operáciu)
- časté vykonávanie operácií sčítania a násobenia
- výpočtovo náročné algoritmy, paralelné operácie

Technológia MMX navrhuje nové inštrukcie, podporujúce tieto operácie. Navrhnuté inštrukcie sú dostatočne všeobecné, aby boli využiteľné v širokej škále algoritmov používaných v multimediálnych aplikáciách, a pritom dostatočne efektívne, aby výrazne zvýšili multimediálny výkon.

MMX bola navrhnutá tak, aby sa zachovala kompatibilita s predchádzajúcimi mikroprocesormi Intel rady x86. Staršie programy, vytvorené pre nižšie verzie procesorov bežia správne aj na procesoroch s MMX, takisto existujúce operačné systémy nie je potrebné modifikovať.

Popíšme teraz MMX podrobnejšie.

1.1 Popis technológie MMX

Technológia MMX je rozšírením inštrukčnej sady intelovskej architektúry (IA). Inštrukčná sada MMX pridáva 57 nových inštrukcií, 8 registrov a nový dátový typ 64-bitové štvorslovo (quad- word). Nové MMX registre sú 64-bitové. Pomenované sú MM0 až MM7.

Využíva sa technika *SIMD* (*single instruction, multiple data*), čiže *'jedna inštrukcia, viac dát'*. Konkrétne: MMX inštrukcie pracujú s MMX registrami (ktoré sú 64 bitové). V 64 bitoch môžeme mať uložených 8 bytov a ako 'za sebou napísaných' osem bytov ho aj niektoré MMX inštrukcie chápu. Jednou MMX inštrukciou (ktorej argumenty sú dva MMX registre) potom vieme napríklad uskutočniť osem súčinov dvojíc bajtov- každý z registrov obsahuje 8 bajtov; vynásobíme zodpovedajúce dvojice. Samozrejme, 64 bitový register môžeme interpretovať aj ako napr. štyri 16-bitové slová či dve 32-bitové a takisto, okrem násobenia môžeme vykonať iné operácie.

Zvýšenie výkonu je práve dôsledkom paralelného spracovania 8, 16 a 32-bitových dátových elementov. MMX inštrukcia môže naraz spracovať až 8 bajtov a navyše, v jednom takte procesora môžu byť vykonané až dve MMX inštrukcie. Dosiahneme tak šestnásť dátových elementov spracovaných v jednom takte.

Dátové typy

Základným dátovým typom inštrukčnej sady MMX je 64 bitový blok obsahujúci viac celočíselných slov (zo združovania je odvodené jeho pomenovanie *'zbalený' (packed) celočíselný typ* (*integer*)). Tieto 64-bitové bloky sú presúvané do 64-bitových MMX registrov. MMX podporuje dátové typy - celé čísla, a to bajty (*bytes*), slová (*words*), dvojslová (*doublewords*) a štvorslová (*quadwords*) so znamienkom (*signed*) a bez znamienka (*unsigned*). Novými dátovými typmi teda sú:

- 'zbalený' bajt (packed byte)– osem bajtov združených do jedného 64-bitového bloku
- 'zbalené' slovo (packed word)– štyri 16-bitové slová združené do jedného 64-bitového bloku
- 'zbalené' dvojslovo (packed doubleword)– dve 32-bitové dvojslová združené do jedného 64-bitového bloku
- štvorslovo (quadword)– jedno 64-bitové číslo

Uvedieme príklad využitia 'zbalených' celočíselných typov. Grafické pixely sú bežne reprezentované 8-bitovými celými číslami. S technológiou MMX je osem takýchto bodov uložených ('zbalených') do jedného 64-bitového bloku a presunutých do MMX registra. Vykonávaná inštrukcia MMX vezme naraz všetkých osem bodov z MMX registra, vykoná aritmetickú alebo logickú operáciu na všetkých ôsmich elementoch paralelne a zapíše výsledok do MMX registra.

MMX registre

Technológia MMX zavádza osem 64-bitových všeobecných registrov. Tieto registre prekrývajú registre pohyblivej rádovej čiarky a môžu uchovávať 'zbalené' 64-bitové dátové typy. MMX inštrukcie pristupujú k MMX registrom priamo prostredníctvom ich názvov MM0 až MM7.

MMX registre môžu byť použité na vykonávanie výpočtov na dátach. Nemôžu byť použité na adresovanie pamäte; práca s pamäťou sa realizuje len 'tradičným' spôsobom procesorov rady x86, t.j. MMX nepridáva nové možnosti adresácie či nové registre pre prácu s pamäťou.

Inštrukcie MMX

Inštrukčná sada MMX poskytuje množinu inštrukcií, ktoré môžu spracovávať všetky dátové elementy 'zbaleného' 64-bitového slova paralelne. Ďalej, je možné spracovávať ich so znamienkom (signed) alebo bez znamienka (unsigned).

MMX inštrukcie implementujú dva nové princípy:

- operácie na 'zbalených' (packed) dátach
- aritmetiku so zarovnávaním[1]

Operácie realizovateľné MMX inštrukciami zahŕňajú viaceré funkčné oblasti[2]:

- základné aritmetické operácie (sčítanie, odčítanie, násobenie, aritmetický posun a násobenie so sčítaním[3])
- logické operácie, (napríklad AND, AND NOT, OR a XOR)
- operácie posunu
- porovnávacie operácie
- inštrukcie pre konverziu medzi novými dátovými typmi (t.j. 'zabalenie' a 'rozbalenie' dát)
- inštrukcie pre presun dát (MOV) medzi MMX registrami a tiež 32 a 64-bitové čítanie a ukladanie do pamäte

Aritmetické a logické operácie sú vykonateľné na rôznych 'zabalených' (packed) celočíselných dátových typoch. Preto existuje viacero inštrukcií realizujúcich tú istú operáciu, no nad inými dátovými typmi. Dôsledkom toho je, že technológia MMX má implementovaných 57 nových operačných kódov.

Prehľad inštrukčnej sady
Inštrukcie v nižšie uvedenej tabuľke sú zoskupené podľa kategórií. Ak inštrukcia podporuje viaceré dátové typy-bajt (B), slovo (W), dvojslovo (DW) alebo štvorslovo (QW), tak sú uvedené v hranatých zátvorkách. Pre danú inštrukciu môže byť zvolený len jeden dátový typ. Napríklad, základná inštrukcia PADD (packed add) má nasledujúce variácie: PADDB, PADDW a PADDD. Je uvedený aj počet operačných kódov spojených so základnou inštrukciou.

Opíšme inštrukcie MMX podrobnejšie.

---

[1] objasníme neskôr

[2] argumentami operácií sú spomenuté 'balené' (packed) celočíselné dátové typy; podrobnejšie popíšeme neskôr

[3] podrobnejšie popíšeme neskôr

Kategória	**Názov**	**Počet variant**	**Popis inštrukcie**
Aritmetické	PADD [B,W,D]	3	Sčítanie
	PADDS [B,W]	2	Sčítanie so znamienkom (signed)
	PADDUS [B,W]	2	Sčítanie bez znamienka (unsigned)
	PSUB [B,W,D]	3	Odčítanie
	PSUBS [B,W]	2	Odčítanie so znamienkom (signed)
	PSUBUS [B,W]	2	Odčítanie bez znamienka (unsigned)
	PMULHW	1	'Zbalené' horné násobenie na slovách
	PMULLW	1	'Zbalené' dolné násobenie na slovách
	PMADDWD	1	'Zbalené' násobenie na slovách a sčítanie výsledných párov
Logické	PAND	1	Bitový AND
	PNAND	1	Bitový NAND
	POR	1	Bitový OR
	PXOR	1	Bitový XOR
Porovnávacie	PCMPEQ [B,W,D]	3	'Zbalené' porovnávanie (rovnosť)
	PCMPGT [B,W,D]	3	'Zbalené' porovnávanie (väčší než)
Konverzné	PACKUSWB	1	'Zabalenie' slov do bajtov (bez znamienka)
	PACKSS [WB, DW]	2	'Zabalenie' slov do bajtov, dvojslov do slov (so znamienkom)
	PUNPCKH [BW,WD,DQ]	3	'Rozbalenie' horných bajtov, slov, dvojslov z MMX registra
	PUNPCKL [BW, WD, DQ]	3	'Rozbalenie' dolných bajtov, slov, dvojslov z MMX registra
Posuny	PSLL [W,D,Q]	6	'Zbalený' logický posun vľavo
	PSRL [W,D,Q]	6	'Zbalený' logický posun vpravo
	PSRA [W,D]	4	'Zbalený' aritmetický posun vpravo
Presun dát	MOV [D,Q]	4	Presun do alebo z MMX registra
Stavové	EMMS	1	Vyprázdnenie MMX stavu

Tabuľka 1.1: Prehľad inštrukčnej sady MMX

	Dolná hranica		**Horná hranica**	
So znamienkom	Hexadecimálne	Desiatkovo	Hexadecimálne	Desiatkovo
Bajt (byte)	80h	-128	7Fh	127
Slovo (word)	8000h	-32768	7FFFh	32767
Bez znamienka				
Bajt (byte)	00h	0	FFh	255
Slovo (word)	0000h	0	FFFFh	65535

Tabuľka 1.2: Dátové hranice pre zarovnávanie

1.2 Aritmetika *'so zarovnaním'* a aritmetika *'bez prenosu'*

Technológia MMX podporuje novú aritmetiku, tz. aritmetiku so zarovnaním.

Pri 'klasickej' aritmetike bez prenosu sa pri pretečení (overflow) alebo podtečení (underflow) ako výsledok operácie zoberú len zobraziteľné, nižšie bity a príznak prenosu (carry) a vyššie bity sa ignorujú (odtiaľ názov 'bez prenosu'). Napríklad, nasledovník čísla (číslo vzniknuté prirátaním jednotky) nemusí byť väčšie číslo; napr. nasledovník 0 je 1, nasledovník 1 je 2, no keďže najväčšie číslo zobraziteľné neznamienkovým bytom je 255, jeho nasledovník je 0.

Pri aritmetike so zarovnaním sa výsledky, ktoré pretečú alebo podtečú, zarovnajú na limitnú hodnotu určenú konkrétnym dátovým typom. Výsledok, ktorý presiahne rozsah dátového typu sa zarovná na maximálnu hodnotu; výsledok, ktorý je menší, než rozsah dátového typu sa zarovná na minimálnu hodnotu. V našom príklade inkrementáciou bajtu s hodnotou 255 dostávame 255 a dekrementáciou 0 dostávame 0.

Zarovnávanie je prostriedkom na zamedzenie nežiadúcich výsledkov aritmetiky 'bez prenosu'. Napr. pri výpočtoch farieb zarovnávanie zabezpečí, že môžeme bez obáv zvyšovať – znižovať jas, bez toho, že by sa po istom čase biela farba zmenila na čiernu či čierna na bielu. Toto by sa nám pri aritmetike bez prenosu mohlo stať – čo je síce ošetriteľné, ale s použitím inštrukcií navyše.

MMX inštrukcie neindikujú pretečenie alebo podtečenie[4] výnimkami alebo nastavovaním príznakov.

1.3 Príklady inštrukcií

V nasledujúcej časti stručne popíšme päť príkladov MMX inštrukcií. Inštrukcie budú pracovať s dátovým typom 16-bitové 'zbalené' slovo; väčšina týchto operácií existuje tiež pre 8-bitové alebo 32-bitové 'zbalené' dátové typy.

1. Inštrukcia *'spakovaného' sčítania slov bez prenosu* vykoná štyri sčítania ôsmich 16-bitových elementov, všetky sčítania sú nezávislé a prebehnú paralelne. V tomto prípade výsledok úplne vpravo presiahne maximálnu hodnotu reprezentovateľnú 16-imi bitmi a odreže sa; nenastane teda prenos 17. bitu do príznaku prenosu (carry flag) a výsledok je 7FFFh.

[4] presnejšie, pokus o akciu, ktorá by pri klasickej aritmetike 'bez prenosu' spôsobila pretečenie alebo podtečenie; napr. dekrementáciu registra, v ktorom je nula.

a_3	a_2	a_1	$FFFFh$
$+$	$+$	$+$	$+$
b_3	b_2	b_1	$8000h$
$a_3 + b_3$	$a_2 + b_2$	$a_1 + b_1$	$7FFFh$

Obrázok 1.1: Inštrukcia PADD[W] (Sčítanie 'bez prenosu')

a_3	a_2	a_1	$FFFFh$
$+$	$+$	$+$	$+$
b_3	b_2	b_1	$8000h$
$(a_3 + b_3$	$a_2 + b_2)$	$(a_1 + b_1$	$FFFFh)$

Obrázok 1.2: Aritmetika zarovnávania

2. Ďalším príkladom je *'zbalené' sčítanie čísel bez znamienka so zarovnaním.* Pre demonštrovanie použime čísla z predchádzajúceho príkladu. Sčítanie krajných elementov vpravo (FFFFh (65535) a 8000h (32768)) generuje výsledok, ktorý sa nezmestí do 16-ich bitov; v tomto prípade nastane zarovnanie. V našom prípade došlo k pretečeniu, preto je výsledok zarovnaný na najväčšiu reprezentovateľnú hodnotu - t.j. FFFFh.

   Spomenutú operáciu vykoná inštrukcia pre *'zbalené' sčítanie slov bez znamienka so zarovnaním* (PADDUSW). Kompletná sada operácií ADD existuje tak pre prípady so znamienkom ako aj bez znamienka. Neexistuje žiaden riadiaci bit, ktorého zmenou by sme určili, či sa má alebo nemá zarovnať; namiesto toho sa používajú rozdielne inštrukcie na získanie výsledkov 'bez prenosu' alebo so zarovnaním.

3. Nasledujúci príklad ukazuje kľúčovú inštrukciu pre operácie tzv. násobenia so sčítaním, ktoré sú základom mnohých algoritmov, napr. skalárneho súčinu alebo násobenia matíc. Touto inštrukciou je 'zbalené' násobenie so sčítaním (PMADDWD). Inštrukcia PMADDWD spracuje 16-bitový 'zbalený' dátový typ a generuje 32-bitový 'zbalený' výsledok. Jej činnosť je zrejmá z obrázku.

4. Nasledujúcim príkladom je 'zbalené' paralelné porovnávanie. Tento príklad porovnáva štyri páry 16-bitových slov. V prípade 'klasických' porovnávacích inštrukcií, kde porovnávame jednu dvojicu slov je výsledkom hodnota 'pravdivý' alebo 'nepravdivý'. V tomto prípade porovnávame 4 dvojice slov a máme teda štyri výsledky, 'zbalené' do jedného. Kódovanie je nasledovné: pre jednu dvojicu sa hodnota 'pravdivý' sa kóduje FFFFh, hodnota 'nepravdivý' 0000h. Štyri dvojice sú popísané štyrmi takýmito šestnásťbitovými slovami. Výsledkom je teda 64 bitové

a_3	a_2	a_1	a_0
$*$	$*$	$*$	$*$
b_3	b_2	b_1	b_0
$a_3 * b_3 + a_2 * b_2$		$a_1 * b_1 + a_0 * b_0$	

Obrázok 1.3: Inštrukcia PMADDWD $(16b \times 16b) \rightarrow 32b$–násobenie so sčítaním

35	18	43	16
$<$	$<$	$<$	$<$
12	25	20	7
$FFFFh$	$0000h$	$FFFFh$	$FFFF$

Obrázok 1.4: Paralelné porovnávanie

slovo vytvorené 'zbalením' štyroch výsledkov pre jednotlivé dvojice. Neexistujú žiadne nové porovnávacie stavy a žiadne existujúce stavy touto inštrukciou nie sú ovplyvnené. Nasledujúci príklad ukazuje porovnanie 'väčší než' na 'spakovaných' slovách.

Výsledok 'zbaleného' porovnávania môže byť použitý ako maska na vyberanie elementov z rôznych vstupov použitím logickej operácie, čím sa obíde nutnosť použitia vetviacich inštrukcií. Možnosť vykonať podmienečný pohyb namiesto použitia vetviacich inštrukcií je dôležitým zvýšením výkonu v procesoroch so zreťazením (pipeline) a predpovedaním vetvení (branch prediction). Tieto techniky budeme popisovať neskôr, no teraz aspoň uvedieme, že branch prediction sa snaží s čo najväčšou pravdepodobnosťou uhádnuť výsledok budúceho vetvenia. Ťažko sa to robí, ak je vetvenie založené na výsledku porovnávacej operácie dát, pretože tie môžu byť 'náhodné'. Prostredníctvom MMX inštrukcií s touto technikou možno eliminovať vetvenie, využité len na výber určitej skupiny dát. Preto je táto technika ďalším faktorom zvyšujúcim výkon na moderných procesoroch, ktoré využívajú predpovedanie vetvenia.

5. Ďalej uvedieme príklad 'pakovacej' (pack) inštrukcie. Inštrukcia zoberie štyri 32-bitové hodnoty a 'zabalí' ich do štyroch 16-bitových hodnôt; v prípade, že sa niektorá 32-bitová hodnota nezmestí do 16-ich bitov, vykoná zarovnanie. Existujú aj inštrukcie, ktoré vykonávajú opak-'rozpakovanie' (unpack), napr. 'zbalený' bajt do 'spakovaného' slova.

   Uviedli sme príklad inštrukcií prevádzajúcich dáta medzi 'zabalenými' dátovými typmi s 16 a 32-bitovými dátovými elementami. Samozrejme, inštrukcie 'balenia' a 'rozbalenia' možno použiť aj na vzájomné konverzie medzi ľubovoľnými 'zabalenými' dátovými typmi. Sú dôležité najmä ak algoritmus v istých úsekoch výpočtu potrebuje vyššiu presnosť, ako napríklad pri filtrovaní obrazu. Filter na obraze zvyčajne zahŕňa súbor operácií násobenia medzi koeficientmi filtra a susednými bodmi obrazu, akumulujúc všetky hodnoty. Tieto násobenia a akumulácie vyžadujú väčšiu presnosť než je pôvodných 8 bitov pre bod. Riešením je 'rozbalenie' 8-bitových obrazových bodov do 16-bitových slov, vykonanie výpočtov v 16-bitoch (bez obáv o pretečenie) a následné 'zbalenie' späť do 8-bitových pixelov.

1.4 Rýchlostné testy

Prirodzenou otázkou je, nakoľko použitie MMX technológie zrýchľuje aplikáciu. Prirodzene, existuje niekoľko štúdií na túto tému. Intel uvádza, že pri spracovaní obrazu možno dosiahnuť zvýšenie výkonu o 500%, čo je však skôr dielom naprogramovania testovacieho softwaru 'plne optimalizovateľného' MMX technológiou. V praxi nemožno takýto vzostup

očakávať, no zvýšenie výkonu o 50-100% by malo byť reálne - čo je skok zodpovedajúci novej generácii procesorov. Výrazne obmedzujúcim faktorom môže byť rýchlosť hardisku, ktorá spomaľuje procesor a preto pri častej práci s ním sa tak môže 'stierať' rozdiel medzi procesormi s/bez MMX. Opäť - záleží od aplikácie. Ostatný software môže takisto použiť MMX, no keďže až tak nepotrebuje vykonávať MMX operácie, zlepšenia budú skôr nepatrné - asi o 10%, čo sa však zrejme dosahuje zdvojenou cache prvej úrovne.

1.5 Záver

Technológia MMX zavádza nové všeobecné inštrukcie, ktoré paralelne pracujúcimi výpočtovými jednotkami spracúvajú väčšie množstvo dát, 'spakovaných' do 64-bitových blokov. Je možné vykonať aritmetické a logické operácie na rôznych dátových typoch (byte, slovo...), zavádzajú sa nové operácie (aritmetika s prenosom, násobenie so sčítaním), ktoré sú vhodné pre multimediálne aplikácie. Inštrukčná sada MMX je plne kompatibilná so všetkými IA mikroprocesormi. Všetok existujúci software beží na mikroprocesoroch s MMX správne, nie je potrebné ho modifikovať.

MMX zrýchľuje o 50-100% výkon aplikácií s výpočtovo náročnými algoritmami, ktoré vykonávajú opakované operácie na malých úsekoch dát. Medzi ne patria napr. video, kombinácia grafiky a videa, spracovanie obrazu, audio syntéza, hlasová syntéza a kompresia, 2D a 3D grafika.

Kapitola 2

Paralelné spracovávanie inštrukcií

V tejto kapitole uvedieme algoritmy, ktoré umožňujú vykonávanie viacerých inštrukcií v jednom takte– t.j. umožňujú paralelné vykonávanie inštrukcií. Na ilustráciu použijeme virtuálny procesor, ktorý budeme postupne zlepšovať. Na príklade konkrétneho kódu (obr. 2.1) budeme pozorovať efektivitu jednotlivých zlepšení – o koľko sa nám zrýchli vykonávanie tohto kódu.

Na začiatku predpokladajme procesor s jednoduchou architektúrou (popísanou v časti III). Tento procesor vždy načíta jednu inštrukciu, vykoná ju a až po jej vykonaní začne načítavať novú inštrukciu. Tieto činnosti sú rozdelené do štyroch fáz – cyklov *fetch* (nahratie inštrukcie z pamäte), *decode* (dekódovanie inštrukcie), *dispatch* (odoslatie inštrukcie na vykonanie) a *execute* (samotné vykonanie inštrukcie). Pre jednoduchosť predpokladajme, že cykly fetch, decode a dispatch trvajú 1 cyklus. Execute–fáza inštrukcie, ktorá nepristupuje do pamäte bude trvať 1 cyklus, prístup do pamäte bude trvať 4 cykly. Procesor bude mať k dispozícii registre (R0, ..., R7).

```
I1:   R1 ⇐ Mem[R0]
I2:   R1 ⇐ (R1) + 2
I3:   R2 ⇐ (R2) + (R3)
I4:   R2 ⇐ (R2) + 1
I5:   if R1=R2 then jump I7
I6:   R2 ⇐ Mem[R4]
I7:   R2 ⇐ (R2) + (R1)
```

Obrázok 2.1: Príklad kódu

V prípade, že podmienka v I5 bude splnená, procesor musí vykonať jednu inštrukciu čítania z pamäte (trvá 1+1+1+4 = 7 cyklov) a päť inštrukcií, ktoré do pamäte nepristupujú (každá trvá 1+1+1+1=4 cyklov). Spolu bude na vykonanie tejto vetvy potrebných 27 cyklov. V prípade, že podmienka v I5 nebude splnená, procesor musí navyše ešte raz vykonať inštrukciu pristupujúcu do pamäte a teda vykonanie bude trvať 34 cyklov.

Jeden zo základných problémov počítačového systému je veľmi malá rýchlosť pamäte v porovnaní s rýchlosťou procesora. Zatiaľ čo procesory sa za posledných 10 rokov zrýchlili viac ako 10-krát, rýchlosť pamäte sa zväčšila iba o 60%. Tento beztak už veľký rozdiel v rýchlosti má pritom rastúci trend. Čiastočne ho možno zmenšiť použitím tzv. *CACHE*

pamätí. Podrobnejšie ich opíšeme v časti o pamätiach, uveďme však aspoň, že sa jedná o rýchle nízkokapacitné pamäte, často umiestnené priamo v procesore, do ktorých prístup zvyčajne trvá len jeden inštrukčný cyklus a ktoré slúžia ako istá náhrada hlavnej pamäte– pamäťové miesta v cache obsahujú kópie často používaných pamäťových buniek hlavnej pamäte, resp. kódy inštrukcií, ktoré sa 'onedlho' budú vykonávať. Ak procesor požaduje údaj, ktorý je v cache, pracuje s ňou a nie s hlavnou pamäťou. Stratégie, ako určiť, ktoré pamäťové bunky budú 'často používané', ako aj ďalšie detaily o cache uvedieme v časti V . V procesoroch sa väčšinou nachádza údajová cache (Data Cache), ale aj inštrukčná cache (Instruction Cache), ktorá uchováva kódy 'onedlho vykonávaných' inštrukcií.

Pre jednoduchosť budeme predpokladať, že fetch fáza každej inštrukcie trvá iba jeden cyklus (aj keď v prípade, že sa inštrukcia nenachádza v inštrukčnej cache, tak procesor musí pristupovať do pamäte). Pri inštrukciách s prístupom do pamäte sme predpokladali, že musia naozaj pristupovať do pamäte (potrebné údaje nie sú v cache - resp. ak procesor nemá cache) a že tento prístup im trvá 4 cykly (v skutočnosti však môže trvať výrazne dlhšie).

Toľko úvodom; skúsme teraz vymyslieť rôzne zlepšenia, ktoré by zrýchlili prácu nášho procesora.

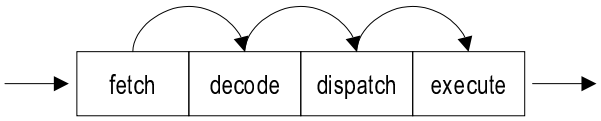
2.1 Pipelining

Náš jednoduchý procesor spracúva inštrukciu v štyroch cykloch: fetch, decode, dispatch a execute. Tieto cykly (fázy) však majú na starosti rôzne funkčné jednotky (obvody) procesora. Teda kým prebieha napr. fetch cyklus, tak pracujú iba príslušné obvody pre fetch fázu a zvyšné sú nevyužité. Metódou na zrýchlenie vykonávania programu je pipelining– ktorý 'neustále využíva' všetky jednotky.

Jeho myšlienka je jednoduchá. Predstavme si, že máme továreň (montujúcu) vyrábajúcu autá. Výroba auta pozostáva z viacerých krokov, napríklad výroba karosérie, namontovanie motora, lakovanie,. . . , výstupná kontrola. Nevyrába sa tak, že počas výroby karosérie ostatné linky čakajú, a keď je hotová, tak sa odovzdá nasledujúcej linke ktorá s ňou pracuje a zase ostatní čakajú so založenými rukami. Nie, po odovzdaní prvej hotovej karosérie nasledovnej linke sa začne vyrábať ďalšia karoséria, paralelne s tým, ako druhá linka spracuváva prvú vyrobenú karosériu (o jej ďalší osud sa už prvá linka nezaujíma). Ako na bežiacom páse.

Pre náš procesor to vyzerá nasledovne: v prvom cykle začne fetch jednotka spracovávať inštrukciu I1 – nahrá ju z pamäte. V druhom cykle je už I1 nahratá a preto môže na I1 začať pracovať jednotka decode. V tom čase však jednotka fetch už môže začať nahrávať ďalšiu inštrukciu – I2. Na konci druhého cyklu už jednotka decode ukončila prácu na I1 a jednotka fetch na I2. Inštrukcie I1 a I2 sa teda môžu 'posunúť o jeden krok ďalej' (I1 do dispatch jednotky a I2 do decode jednotky) a jednotka fetch môže začať nahrávať I3, atď. . . Vykonávanie inštrukcie je teda rozdelené na niekoľko po sebe idúcich fáz (v našom prípade štyri), ktoré sa uskutočňujú v navzájom nezávislých funkčných jednotkách procesora.

Tento spôsob vykonávania sa nazýva pipelining, čiže prúdové spracovanie. Samotný názov pipelining pochádza zo slova pipeline- potrubie, čie pipelining značí *prúdové spracovanie inštrukcií*. Inštrukcie sú akoby na pohyblivom páse, na ktorom ich postupne obsluhujú jednotlivé jednotky procesora. Pipeline-ové vykonávanie ilustračného programu je zobrazené na obrázku 2.3.



Inštrukcia vstúpi do pipelinu a postupne prechádza všetkými fázami. Ak je nasledujúca jednotka voľná, inštrukcia môže prejsť do ďalšej fázy.

Obrázok 2.2: Zobrazenie vykonávania inštrukcie v pipeline.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F	1	2	3	4	4	4	4	5				7			
De		1	2	3	3	3	3	4	5				7		
Di			1	2	2	2	2	3	4	5				7	
E				1	1	1	1	2	3	4	5				7

Podmienka v I5 je splnená

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
F	1	2	3	4	4	4	4	5				6	7						
De		1	2	3	3	3	3	4	5				6	7					
Di			1	2	2	2	2	3	4	5				6	7	7	7	7	
E				1	1	1	1	2	3	4	5				6	6	6	6	7

Podmienka v I5 nie je splnená

Na obrázkoch je zobrazený pipeline a jeho obsah počas jednotlivých cyklov vykonávania kódu. Písmená F, De, Di a E označujú fetch, decode, dispatch a execute fázy pipelinu. Čísla nad stĺpcami označujú cyklus procesora a čísla v políčkach sú čísla inštrukcií, ktoré sa práve vykonávajú. Vykonávanie trvá v prípade (a) 15 cyklov, v prípade (b) 19 cyklov.

Obrázok 2.3: Zobrazenie vykonávania časti kódu v procesore s pipelinom.

Teoreticky by sa beh kódu mal zrýchlil 4-krát (v každom cykle sa začne vykonávať nová inštrukcia). Všimnime si však obrázok 2.3.

Vidíme, že kód sa síce vykonal rýchlejšie, ale zďaleka nie 4-krát rýchlejšie.

Prvým dôvodom spomalenia pipelinu je už spomínaný pomalý prístup do pamäti. Zapríčiní, že inštrukcie I2, I3 a I4 čakať na ukončenie vykonávania I1 3 cykly.

Druhý dôvod bol príčinou 'bubliny' v pipeline, ktorá sa vytvorila medzi I5 a I7 (resp. I6). I5 je vetviaca inštrukcia (angl. branch instruction) – ktorá mení tok programu. Podľa toho, či je splnená podmienka R1=R2 sa pokračuje buď na inštrukcii I7 alebo I6. Aby sme mohli ďalej vykonávať kód (po I5), musíme vedieť výsledok I5 (musíme vedieť, či pokračovať na I7 alebo na I6). Treba si uvedomiť, že inštrukcia I5 sa síce začne vykonávať už v 8. cykle, ale jej výsledok je známy až na konci 11. cyklu. Čiže procesor až na konci 11. cyklu vie, či má jednotka fetch nahrávať I7 alebo I6. V pipeline vznikla 'bublina', počas ktorej niektoré jednotky procesora nepracovali.

Experimentálne sa zistilo, že v priemere je každá piata inštrukcia bežných programov vetviaca inštrukcia (branch inštrukcia). Potom ale v procesore s pipeline-om dĺžky štyri (dĺžka pipelinu je rovná počtu fáz vykonávania inštrukcie v pipeline) po piatich cykloch stále nasleduje prestávka na tri cykly (musíme počkať, kým sa vykoná branch).

V moderných procesoroch je trendom používať pipeline čím väčšej dĺžky. Pri použití pipelinu ako sme ho popísali, by napr. Pentium Pro, ktoré má pipeline dĺžky 12 muselo po každých piatich cykloch čakať ďalších 11 cyklov na vykonanie vetviacej inštrukcie.

Výrazná pomoc by bola, ak by sme dopredu vedeli *predpovedať*, aký výsledok bude

mať I5; teda či bude podmienka splnená alebo nie. Ak by sme totiž vedeli, že vetviaca inštrukcia I5 skočí (splní sa podmienka a vykoná sa skok na I7), tak by sme už v 9. cykle mohli začať vykonávať I7. Takto by v pipeline nevznikla žiadna bublina. Táto metóda sa nazýva *branch prediction* (predpovedanie vetvenia) a je bližšie popísaná v nasledujúcej kapitole.

Spomenuli sme, že inštrukcie pracujúce s pamäťou môžu výrazne spomaliť pipeline. Ďalšími negatívnymi faktormi (spomaľujúcimi pipeline) sú:

- *inštrukcie s premenlivou dĺžkou* - počas dekódovania je nutný viacnásobný prístup do pamäti
- *príliš zložité inštrukcie* - pomalšie než ostatné
- *inštrukcie, ktoré čítajú aj zapisujú do toho istého registra*

Tieto problémy je ale možné vyriešiť. Napríklad vykonanie zložitých inštrukcií možno zrýchliť pridaním ďalších obvodov. Vidíme však, že nevhodná inštrukčná sada môže znemožniť efektívny pipeline. Preto majú procesory s pipeline-om obmedzenú inštrukčnú sadu. Prakticky však tieto obmedzenia nespôsobujú veľké problémy a procesory s pipeline-om sú efektívnejšie ako procesory bez neho.

2.2 Predpovedanie výsledkov vetvenia

Predpovedanie výsledkov vetvenia (*branch prediction*) slúži na určenie vetvy, ktorou sa bude ďalej uberať výpočet; čiže predpovedá splnenie podmienky vo vetviacej inštrukcií.

Význam branch prediction sme naznačili. Umožňuje procesoru 'vidieť dopredu' v toku inštrukcií, t.j. procesor vie v každom kroku výpočtu určiť k nasledujúcich krokov výpočtu. Vďaka tomu potom procesor môže do pipe-u zaraďovať 'správne' inštrukcie, bez toho, že by pipe musel 'stáť' a čakať na výsledok vetviacich inštrukcií.

Branch prediction sa používa aj na iné účely; napr. je potrebný pre realizáciu ďalšej techniky, nazývanej out-of-order execution, ktorú popíšeme neskôr. Takisto sa oboznámime s value prediction, čo je akési 'zovšeobecnenie' branch prediction - hádame výsledky operácií (pričom výsledok môže byť prvkom veľmi veľkej množiny - napr. všetkých šesťdesiat-bitových čísel).

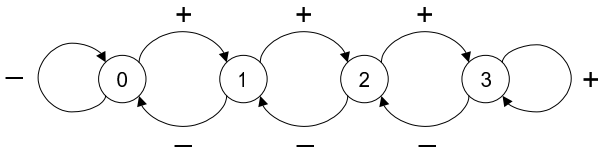
Branch prediction je teda algoritmus, ktorý s veľkou pravdepodobnosťou správne predpovedá výsledky vetvenia. Rôzne algoritmy majú samozrejme rôznu úspešnosť, t.j. rôznu pravdepodobnosť uhádnutia výsledku; pričom na zvýšenie úspešnosti používajú čo 'najprefíkanejšie' metódy. Popíšeme jeden konkrétny algoritmus branch prediction.

Two-level adaptive branch predictor

Je branch predictor, ktorý pri predpovedaní dosahuje až 97% úspešnosť. Tento algoritmus na branch prediction je použitý aj v Pentiu Pro.

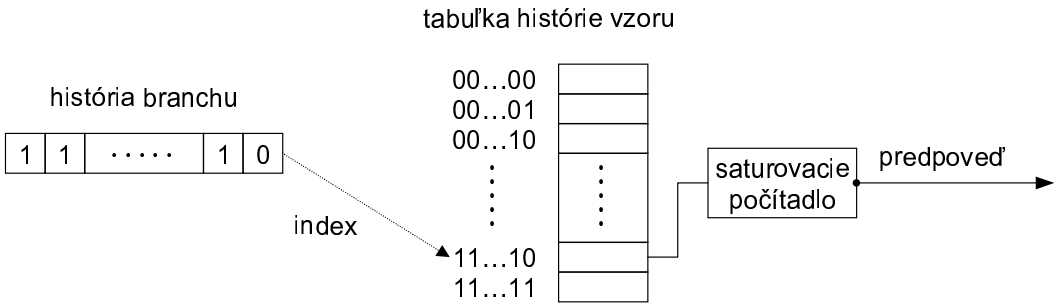
Pre jeho objasnenie najsamprv musíme popísať tzv.*saturovacie počítadlo.*

Saturovacie počítadlo na rozdiel od obyčajného počítadla nikdy nepretečie. Presnejšie: ak je počítadlo nastavené na maximálnu hodnotu a pripočítame 1, tak hodnota v



Na obrázku je zobrazené 2-bitové saturovacie počítadlo, ktoré vie 'počítať' od 0 do 3. Pri pripočítavaní 1 sa pohybujeme po šípkach označených symbolom '+', pri odčítavaní po šípkach označených '−'.

Obrázok 2.4: Dvojbitové saturovacie počítadlo.



História branchu (HB) zaznamenáva, ako sa branch správal pri posledných k opakovaniach. HB slúži ako index pre tabuľku histórie vzoru. Predictor sa pozrie na saturovacie počítadlo, na ktoré ukazuje HB a podľa neho predpovie, či branch skočí alebo nie.

Obrázok 2.5: Two-level branch predictor

počítadle nepretečie na nulu, ale ostane tam maximálna hodnota. Podobne, ak od minimálnej hodnoty (nuly) odrátame 1, tak výsledkom bude nula (počítadlo nepodtečie). V prípade 8-bitového počítadla je teda $255 + 1 = 255$ a $0 - 1 = 0$.

Two-level adaptive branch predictor používa dve dátové štruktúry: históriu vetvení (HB) a tabuľku histórie vzoru (THV).

HB je register, ktorý zaznamenáva históriu výsledkov vetvení, čiže či vetviaca podmienka platila alebo nie. Napríklad, ak nejaký podmienený skok najprv skočil, potom dvakrát nie a potom trikrát opäť skočil, tak jeho HB bude 100111. Ak je HB k-bitový register, tak zaznamenáva históriu posledných K výsledkov branchu.

THV predstavuje druhú, vzletne povedané 'vyššiu úroveň pozorovania' histórie. Každý možný vzor – história– má svoju vlastnú históriu v THV. Ak HB zaznamenáva históriu branchu na k krokov dozadu (t.j. je to k-bitový register), tak existuje 2^k rôznych histórií branchu. Pre každú takú históriu existuje v THV 2-bitové saturovacie počítadlo. Toto saturovacie počítadlo zaznamenáva pre nejakú históriu branchu, ako sa branch správa pri výskyte takejto histórie. Zakaždým, keď sa vyskytne vetviaca inštrukcia (branch), predictor sa pozrie na jeho históriu a potom sa pozrie do THV na počítadlo pre túto históriu. Ak má toto počítadlo hodnotu 0 alebo 1, tak predictor predpovedá, že branch neskočí, ak je väčšie ako jedna tak predpovedá, že skočí. Neskôr, keď je známy výsledok tohto branchu, tak sa toto počítadlo inkrementuje ak branch v skutočnosti skočil a dekrementuje ak neskočil. Two-level branch predictor je zobrazený na horeuvedenom obrázku.

Ak má napr. HB 4 bity, THV bude mať $2^4 = 16$ položiek s počiatočnými hodnotami

01. Predpokladajme, že chceme predpovedať branch, ktorý pri histórii 1101 stále skočí (po 1101 nasleduje stále 1). Pri prvom výskyte histórie 1101 bude mať položka 1101_2 v THV hodnotu 01 a preto sa predpovie, že branch neskočí. Branch však v skutočnosti skočí a položka 1101_2 v THV sa teda inkrementuje na hodnotu 10. Pri ďalších výskytoch histórie 1101 už predictor správne predpovie, že branch skočí.

Autori si však uvedomili, že ak predpovedáme len jedno vetvenie dopredu, tak procesor vidí v priemere len o 5 inštrukcií viac (v priemere je každá piata inštrukcia branch). Päť inštrukcií je však málo, a tiež, čím viac inštrukcií dopredu procesor vidí, tým lepšie. Preto na základe svojho dvojúrovňového adaptívneho predikátora vetvenia navrhli multiple branch predictor, pomocou ktorého predpovedali 2 až 3 vetvenia dopredu. Správnosť predpovedania ich predictora je približne 95%. Procesor je teda pomocou tohto algoritmu schopný vidieť približne 10 až 15 inštrukcií dopredu.

Alternatívou k branch prediction môže byť tzv. *predicative execution*. Pri predicative execution sa dopredu začnú vykonávať obe vetvy vetviacej inštrukcie (na nezávislých výpočtových jednotkách). Kým sa vyhodnotí vetviaca podmienka (resp. vykoná vetviaca inštrukcia), budú pripravené (vykonané) obe vetvy programu a vykonávanie bude môcť pokračovať ďalej.

Pre realizovanie predicative execution procesor musí mať dostatok funkčných (výpočtových) jednotiek a registrov na zapamätanie si výsledkov inštrukcií v oboch vetvách. Za cenu zvýšenia počtu jednotiek a registrov však dokážeme eliminovať zlé predpovedanie branch predictorov.

2.3 Superskalárne vykonávanie

Ďalšia prirodzená myšlienka ako ďalej zrýchliť vykonávanie programu je spracovávať inštrukcie paralelne. Namiesto jednej výpočtovej jednotky ich procesor bude mať viacero. Aplikovaním aj predchádzajúceho odseku o pipeline - nepoužijeme jeden pipeline, ale viacero. Táto metóda sa nazýva superskalárne vykonávanie (*superscaling*).

Rozšírme teraz náš ukážkový jednoduchý procesor o túto techniku. Nech má dva pipeliny. Opíšme, ako pracuje: procesor berie jednu inštrukciu po druhej a ak je nejaká pipeline voľná, začne v nej inštrukciu vykonávať. V ideálnom prípade by mal tento procesor pracovať dvakrát rýchlejšie ako procesor s jedným pipelinom. Na obrázkoch 2.5a 2.6je znázornené, ako by procesor s dvoma pipelinami spracovával našu časť kódu.

Vidíme, že predchádzajúce dva problémy stále ostávajú: pamäť je príliš pomalá (inštrukcie I3 a I5 musia čakať na dokončenie vykonávania I1) a v pipeline vznikajú bubliny kvôli branchom (I6 a I7 musia čakať na výsledok I5). Tieto problémy sú dosť vážne, pretože napriek tomu, že sme pridali jeden pipeline, sa vykonávanie zrýchlilo iba o jeden cyklus (namiesto očakávaných 7 – 9 cyklov).

Je zrejmé, že nie vždy možno ľubovolne prehádzať poradie vykonávania inštrukcií. Hovoríme, že inštrukcia J je závislá na inštrukcii I, ak I nastavuje alebo mení obsah registrov alebo pamäte, ktoré potom J berie ako vstup. J potom môžeme začať vykonávať, až keď sa ukončí vykonávanie I. Napr. inštrukciu I2 nevieme vykonať, kým neskončila inštrukcia I1, pretože I1 načíta do R1 obsah z pamäte a I2 tento *nový* obsah R1 zväčší o 2. Ak by sme I2 vykonali skôr ako I1, tak výsledok (obsah registra R1) by bol nesprávny. Teda inštrukcia I2 je závislá na I1 (kompletný graf závislosti inštrukcií v našej časti kódu

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
F	1	3	5								7			
De		1	3	5	5	5	5					7		
Di			1	3	3	3	3	5	5				7	
E				1	1	1	1	3		5				7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
F	2	4												
De		2	4	4	4	4	4							
Di			2	2	2	2	2	4						
E								2	4					

Obrázok 2.6: Spracovanie kódu procesorom s dvoma pipelinami v prípade, že podmienka v I5 je splnená

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
F	1	3	5								6							
De		1	3	5	5	5	5					6						
Di			1	3	3	3	3	5	5				6					
E				1	1	1	1	3		5				6	6	6	6	

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
F	2	4									7							
De		2	4	4	4	4	4					7						
Di			2	2	2	2	2	4					7	7	7	7	7	
E								2	4									7

Obrázok 2.7: Spracovanie kódu procesorom s dvoma pipelinami v prípade, že podmienka v I5 nie je splnená

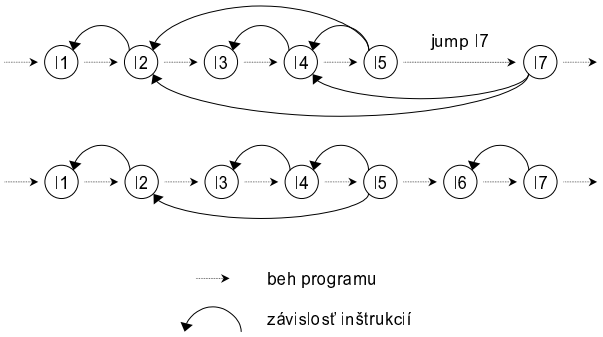
je na nasledovnom obrázku). Naopak inštrukcie I2 a I3 na sebe nie sú závislé, pretože pracujú s rôznymi registrami a môžu byť vykonané v ľubovoľnom poradí.

Z obrázkov 2.5 a 2.6 vidno, že inštrukcia I2 čaká na dokončenie I1 a tak blokuje inštrukciu I4. Existujú dve riešenia tohto problému:

- inštrukcie, ktoré na sebe nie sú závislé, môžeme preusporiadať a začať ich vykonávať v inom poradí. Ak začneme inštrukcie vykonávať vo vhodnom poradí, môžeme sa zbaviť niektorých závislostí. Táto metóda sa volá *out-of-order execution* (vykonávanie mimo poradia).

- ďalšou možnosťou je *speculative execution* ('špekulatívne' vykonávanie). V tomto prípade sa snažíme odhadnúť – predpovedať – obsah registrov dopredu a vykonať aj závislé inštrukcie skôr (s predpovedanými hodnotami).

Treba si uvedomiť, že pri použití superscalingu výrazne vzrastá potreba predpovedania vetvení. V nových procesoroch sa totiž vyskytuje až 5 pipeline-ov, čo by znamenalo, že priemerne sa v každom cykle vyskytne jedna vetviaca inštrukcia. Ak majú navyše tieto pipeliny hĺbku 12, tak to znamená 1 cyklus práce a potom 11 cyklov čakania na výsledok jediného branchu.



Obrázok 2.8: Zobrazenie závislostí inštrukcií v našom kóde. Hore – podmienka v I5 je splnená, dole – podmienka v I5 nie je splnená

2.4 Vykonávanie mimo poradia

Vykonávanie mimo poradia (*out-of-order execution*) je metóda využívajúca fakt, že ak nejaká inštrukcia, ktorá nie je závislá na žiadnej inštrukcii ,(ktorá je v poradí pred ňou), potom ju môžeme (v prípade, že sa nám to hodí) vykonať v predstihu, skôr než príde na rad.

Využitie tejto metódy môžeme ilustrovať na našej časti kódu. Inštrukcia I4 je závislá na inštrukcii I3. Táto závislosť môže v superskalárnom počítači spôsobiť zablokovanie vykonávania I4 (kým sa neskončí vykonávanie I3). Jednotky procesora, ktoré by normálne paralelne vykonávali ďalšiu inštrukciu (I4) sú nevyužité. Preto, keď bude musieť procesor stáť – napr. čakanie na ukončenie load inštrukcie – môže využiť čas užitočnejšie a vykonať inštrukciu I3 mimo poradia (I3 nie je závislá na I1 ani I2). Neskôr, keď sa začne vykonávať I4, bude už I3 ukončená a teda nebude blokovať I4.

Inštrukcia I3 sa vykoná v jednom cykle (nepristupuje do pamäte). Preto zablokuje I4 iba na jeden cyklus. Predpokladajme, že podmienka v inštrukcii I5 nebude splnená. Potom sa musia vykonať inštrukcie I6 a I7. I7 je závislá na I6, ktorá pristupuje do pamäte, preto bude I7 zablokovaná na veľmi dlhý čas (4 cykly v našom procesore, ale v skutočnosti oveľa viac; viď obr. 5). Preto, ak by sme vedeli pomocou prediktora správne predpovedať výsledok vetvenia I5, tak môžeme inštrukciu I6 vykonať skôr (I6 nezávisí na predchádzajúcich inštrukciách). I7 potom nebude zablokovaná a procesor ušetrí veľa cyklov.

Vidíme, že niekedy je výhodné použiť out-of-order execution. Čiastočne kompenzuje omeškanie pamäte. Pre efektívnu realizáciu out-of-order execution však potrebujeme výkonný branch prediction, inak totiž 'nevidíme' tok programu dosť ďaleko dopredu a nevieme naplánovať, ktoré inštrukcie sa majú vykonať skôr.

Každý predpovedaný branch sa samozrejme musí neskôr overiť. Keď príde na branch rad vo vykonávaní, tak sa vykoná a výsledná hodnota sa porovná s predpovedanou. Ak sme predpovedali správne, tak vieme, že sa vykonávali správne inštrukcie. Ak však branch predictor predpovedal zle, vykonávali sme inštrukcie z nesprávnej vetvy – z vetvy, do ktorej by sa tok programu nedostal. Preto musíme na všetky tieto 'zlé' inštrukcie 'zabudnúť' a začať znovu vykonávať program od zle predpovedaného branch-u. V prípade zlej predpovede sa musí vyprázdniť celý pipeline, pretože sa v ňom nachádzajú inštrukcie, ktoré nepotrebujeme vykonať. Ak je pipeline dlhý, stojí takáto zlá predpoveď procesor

veľmi veľa času.

Uvažujme chvíľu, že máme k dispozícii dokonalý branch predictor, ktorý správne predpovie každé vetvenie. Na obrázku 2.8vidno, ako by sa v tom prípade vykonával náš kód, keby sme použili out-of-order execution. V prípade (a) branch predictor predpovie, že podmienka je splnená a inštrukcia I5 skočí. Vieme teda, že po I5 nasleduje I7, preto ju môžeme začať vykonávať skôr. Po skončení vykonávania I5 procesor overí, či bol branch predpovedaný správne. V prípade (b) predictor predpovie, že branch neskočí. Po I5 teda nasledujú inštrukcie I6 a I7. Obe sa začnú vykonávať skôr, aby procesor nemusel čakať na pomalý prístup do pamäte.

Naopak, čo sa stane pri zlom predpovedaní? Na obrázku 2.9vidíme, ako zlé predpovedanie branch inštrukcie I5 predĺži vykonávanie programu. Branch predictor predpovedal, že podmienka v I5 bude splnená a teda, že po I5 sa bude vykonávať I7. Po vykonaní I5 však procesor zistil, že podmienka v I5 nebola splnená a skok sa neuskutočnil. Musia sa teda ešte vykonať inštrukcie I6 a I7.

2.5 Špekulatívne vykonávanie

Špekulatívne vykonávanie (*speculative execution*) rieši problém závislosti inštrukcií inak ako out-of-order execution. Ak je nejaká inštrukcia závislá na iných inštrukciách, tak sa procesor pokúsi uhádnuť jej vstupné hodnoty (t.j. výsledky inštrukcií, na ktorých závisí) a inštrukcia je potom spustená s predpovedanými hodnotami. Neskôr, keď sú známe jej skutočné vstupné hodnoty, tak sa porovnajú s 'tipovanými' (t.j. predpovedanými)

	1	2	3	4	5	6	7	8	9
F	1	2	5						
De		1	2	5	5	5	5		
Di			1	2	2	2	2	5	
E				1	1	1	1	2	5

	1	2	3	4	5	6	7	8	9
F	3	4	7						
De		3	4	7					
Di			3	4	7	7	7	7	
E				3	4				7

Podmienka v I5 je splnená

	1	2	3	4	5	6	7	8	9	10
F	1	2	5							
De		1	2	5	5	5	5			
Di			1	2	2	2	2	5		
E				1	1	1	1	2	5	

	1	2	3	4	5	6	7	8	9	10
F	3	4	6	7						
De		3	4	6	7					
Di			3	4	6	7	7	7	7	
E				3	4	6	6	6	6	7

Podmienka v I5 nie je splnená

Obrázok 2.9: Zobrazenie spracovania kódu procesorom s dvoma pipelinami a out-of-order execution

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
F	1	2	5							6							
De		1	2	5	5	5	5				6						
Di			1	2	2	2	2	5				6					
E				1	1	1	1	2	5				6	6	6	6	

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
F	3	4	7							7							
De		3	4	7							7						
Di			3	4	7	7	7	7				7	7	7	7	7	
E				3	4				7								7

Branch inštrukcia I5 bola zle predpovedaná. Branch predictor predpovedal skok, ale I5 neskočila. Musia sa vykonať inštrukcie I6 a I7.

Obrázok 2.10: Zlá predpoveď branch inštrukcie

hodnotami. Ak bola predpoveď správna, ak inštrukciu už nemusíme vykonať, jej výsledok je správny. Ak však predpoveď správna nebola, potom musíme vykonať inštrukciu znova, so skutočnými vstupnými hodnotami.

Napríklad, v našom prípade môže procesor predpovedať hodnotu, ktorú z pamäte načíta inštrukcia I1. Potom môže s touto predpovedanou hodnotou spustiť inštrukciu I2. Obdobne ako pri out-of-order execution, aj pri speculative execution musí byť každá predpovedaná hodnota neskôr overená. Ak bola predpoveď správna, môže program bežať ďalej. Ak však bola predpovedaná nesprávna inštrukcia, znamená to, že všetky inštrukcie, ktoré sú na nej závislé dostali zlé vstupné hodnoty. Teda všetky inštrukcie, ktoré sú

	1	2	3	4	5	6	7	8	9	10	11
F	1							7			
De		1							7		
Di			1							7	
E				1	1	1	1				7

	1	2	3	4	5	6	7	8	9	10	11
F	2	3	4	5							
De		2	3	4	5						
Di			2	3	4	5					
E				2	3	4	5				

Podmienka v I5 je splnená

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
F	1							6						
De		1							6					
Di			1							6				
E				1	1	1	1				6	6	6	6

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
F	2	3	4	5				7						
De		2	3	4	5				7					
Di			2	3	4	5				7				
E				2	3	4	5				7			

Podmienka v I5 nie je splnená

Obrázok 2.11: Zobrazenie spracovania časti kódu procesorom s dvoma pipelinami a speculative execution

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
F	1							2					7			
De		1							2					7		
Di			1							2					7	
E				1	1	1	1				2					7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
F	2	3	4	5				5								
De		2	3	4	5				5							
Di			2	3	4	5				5	5					
E				2	3	4	5					5				

Inštrukcia I1 bola predpovedaná zle. Inštrukcie I2 a I5 sa musia vykonať znova.

Obrázok 2.12: Zlé predpovedanie výsledku inštrukcie pri speculative execution

závislé na inštrukcii so zle predpovedanou vstupnou hodnotou musia byť vykonané znova. Pri zlej predpovedi musí procesor aj v prípade speculative execution stráviť veľa času na jej opravení.

Na obrázku 2.10 je zobrazené vykonávanie časti kódu (obr. 2.1) v procesore, ktorý používa speculative execution a dokonalý value predictor (všetky predpovedané vstupné hodnoty sú správne). V prípade (a) je predpovedaná hodnota, ktorú nahrá inštrukcia I1 z pamäte. I2 sa teda môže začať vykonávať už v prvom cykle. Po skončení vykonávania I1 (v 7. cykle) sa overí predpovedaná hodnota a keďže bola správna, tak sa pokračuje ďalej. V prípade (b) sa predpovie aj hodnota, ktorú nahrá z pamäte I6.

Naopak, na nasledujúcom obrázku (obr. 2.11) je zobrazené, čo sa stane, ak value predictor nesprávne predpovie hodnotu inštrukcie I1. Keď procesor zistí, že hodnota bola predpovedaná zle, musia sa opätovne vykonať inštrukcie, ktoré sú na nej závislé. V našom prípade sa teda vykonajú inštrukcie I2 a I5.

2.6 Predpovedanie hodnôt

Predpovedanie hodnôt (*value prediction*) je dôležitá technika, ktorá umožňuje vykonávanie speculative execution.

Najčastejšie sa zvyknú predpovedať branche (branch prediction). Pri branch prediction sa musí predpovedať iba jeden bit – či branch skočí alebo nie. Value prediction, predpovedá celý obsah registrov (t.j. napr. 32 bitov). Zdá sa, že pri predpovedaní 32 bitov nie je veľká šanca na správnu predpoveď. Ukázalo sa však , že hodnoty počítané inštrukciami sa zvyknú väčšinu času opakovať alebo tvoria rôzne postupnosti, ktoré sa dajú predpovedať (opäť, poznatok je štatistický a platí v priemernom prípade).

Popíšme si postupnosti, ktoré môžu vznikať pri počítaní inštrukcie. Môžu to byť:

- konštantné postupnosti (2, 2, 2, 2, ...)
- postupnosti posunuté o konštantu (1, 2, 3, 4, 5, 6, ...) alebo
- 'náhodné' postupnosti (24, -2, 13, 7, ...)

Konštantné postupnosti sú najjednoduchšie a vznikajú z inštrukcií, ktoré stále produkujú rovnaký výsledok. Postupnosti posunuté o konštantu vznikajú napr. v cykloch,

alebo pri práci s dátovými štruktúrami ako napr. pole. Dôležité postupnosti sú aj tie, ktoré vznikajú opakovaním rôznych hodnôt (napr. 1, 2, 3, 1, 2, 3, ... alebo 1, -13, 29, 7, 1, -13, 29, 7, ...). Takéto opakované postupnosti vznikajú hlavne vo vnorených cykloch, kde práve vnútorný cyklus vytvára opakované hodnoty.

Typický value predictor vezme ako vstup stav procesora, pozrie sa do svojej tabuľky a predpovie hodnotu. Potom tabuľku aktualizuje novými informáciami. Predictor môže mať ako vstup rôzne stavové informácie ako napr. hodnoty registrov, hodnotu PC registra, kód aktuálnej inštrukcie (ktorej hodnotu chceme predpovedať), kontrolné bity v rôznych stavoch pipelinu, atď ....

Vo všeobecnosti existujú dva rôzne spôsoby predpovedania:

- pri *výpočtovom predpovedaní* je predpovedaná hodnota funkciou predchádzajúcich hodnôt. Príkladom je *stride predictor*, ktorý k posledne vypočítanej hodnote priráta konštantu.
- *kontextové predpovedanie* sa učí, aká hodnota (či hodnoty) nasleduje po určitej postupnosti hodnôt a potom, keď sa daná postupnosť zopakuje, predpovie jednu z naučených hodnôt.

Výpočtové predpovedanie

Last value predictor

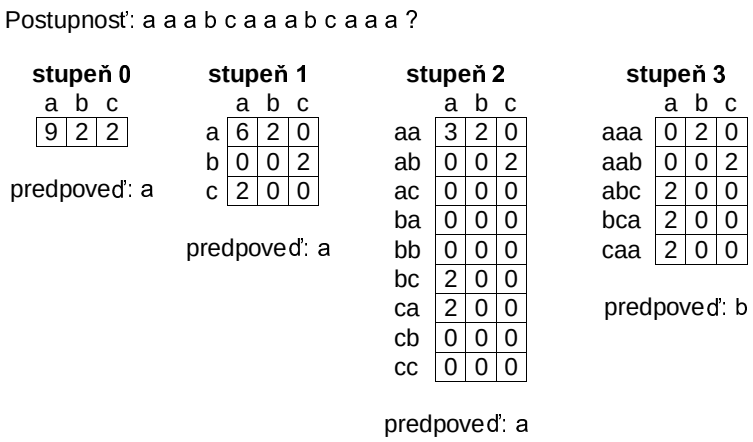
(predpovedanie poslednej hodnoty) vo svojej najjednoduchšej podobe vracia ako predpoveď poslednú hodnotu, ktorú inštrukcia vypočítala. Existujú však aj lepšie metódy, ktoré zvyšujú schopnosť správne predpovedať. Príkladom je použitie saturovacieho počítadla pre každú inštrukciu. Počítadlo je inkrementované (resp. dekrementované) pri každej správnej (nesprávnej) predpovedi. Predictor nezmení predpovedanú hodnotu dovtedy, kým sa nová hodnota nevyskytne niekoľko krát za sebou.

Stride predictor

(predpovedanie postupností posunutých o konštantu) môže vyzerať napríklad nasledovne: procesor si bude pamätať posledné dve hodnoty vypočítané inštrukciou. Pri predpovedaní pripočíta k poslednej hodnote rozdiel medzi poslednými dvoma hodnotami. Ak teda boli posledné dve hodnoty vypočítané inštrukciou v_{n-2} a v_{n-1}, tak predpovedaná hodnota bude $v_{n-1} + v_{n-1} - v_{n-2}$.

Rovnako ako pri last-value predictoroch aj tu existuje veľa modifikácií. Napr. je použité saturovacie počítadlo, ktoré je inkrementované/dekrementované pri každej správnej, resp. nesprávnej predpovedi a diferencia (rozdiel hodnôt v postupnosti posunutých o konštantu) sa zmení, iba ak je toto počítadlo pod určitou hranicou. Táto zmena zmenší počet zlých predpovedí pri opakovaných postupnostiach z dvoch na jednu.

Ďalšia metóda je tzv. *two-delta method*, pri ktorej si predictor pamätá dve diferencie. Prvá diferencia (d_1) je vždy rozdielom medzi poslednými dvoma hodnotami vypočítanými inštrukciou. Druhá diferencia (d_2) je používaná na predpovedanie hodnôt. Ak sa diferencia d_1 vyskytne dvakrát za sebou, d_2 sa nastaví na hodnotu d_1. Metóda two-delta tiež zníži počet zlých predpovedí pri opakovaných postupnostiach - mení diferenciu, iba ak sa nová diferencia vyskytne dva krát.



V riadkoch sú uvedené histórie (nejakej inštrukcie), ktoré si pamätá daný fcm predictor. Ku každej histórii je v tabuľke uvedené, koľkokrát sa po danej histórii vyskytla hodnota (písmeno) a, b a c. Predictor si pamätá svoju aktuálnu históriu a z a, b, c vyberie písmeno, ktoré sa po tejto histórii vyskytovalo najčastejšie. Predictor stupňa 2 vybral písmeno a, pretože po aa sa najčastejšie vyskytuje a. Až predictor stupňa 3 vybral správne b, pretože sa po aaa vyskytovalo najčastejšie.

Obrázok 2.13: Tabuľky pre fcm predictory rôznych stupňov

Kontextové predpovedanie

Pri kontextovom predpovedaní predictor vníma, aké hodnoty nasledujú po určitej skupine predchádzajúcich výsledkov. Z toho pochádza jeho názov- výsledok inštrukcie sa chápe v určitom kontexte – konečnej postupnosti predchádzajúcich hodnôt inštrukcie.

Najväčšou skupinou kontextových predictorov sú tzv. *finite context method* predictory (fcm). Fcm predictory predpovedajú nasledujúcu hodnotu podľa konečného počtu predchádzajúcich hodnôt vypočítaných inštrukciou (história inštrukcie). Fcm predictor stupňa k používa k predchádzajúcich hodnôt. Využíva tabuľku, kde sa pre každú históriu inštrukcie (kontext) a danú hodnotu pamätá, koľkokrát sa po tejto histórii vyskytla daná hodnota. Predictor si teda pamätá aktuálnu históriu inštrukcie a predpovie hodnotu, ktorá sa po danej histórii vyskytovala najčastejšie. Na obrázku 2.12 je príklad fcm predictorov rôznych stupňov.

Pri realizácií nie je možné pamätať si v tabuľke presné počty výskytov, pretože na reprezentáciu počítadla je použitý neveľký počet bitov. Čiastočne to možno riešiť napr. tak, že ak niektoré počítadlo dosiahne maximálnu hodnotu, hodnoty vo zvyšných počítadlách pre rovnakú históriu sa predelia dvoma.

Existuje veľa variantov fcm predictorov. Na predpovedanie sa napr. môže použiť n rôznych fcm predictorov stupňov 0 až $n-1$, pričom každý predictor si pamätá iba niekoľko 'vlastných' histórií. Pri predpovedaní sa potom vyberie predictor s najvyšším stupňom, ktorý si pamätá aktuálnu históriu. Kvôli ušetreniu pamäte si môže fcm pamätať pre každú históriu iba jednu hodnotu.

Ukázalo sa , že úspešnosť last value predictorov je v priemere 40% a úspešnosť stride predictorov okolo 55%. Čím väčší je stupeň fcm predictorov, tým presnejšie predpovedajú. Fcm predictor stupňa 3 má priemernú úspešnosť 78%. Taktiež sa ukázalo, že viac ako

polovica statických inštrukcií generuje iba jednu hodnotu a tiež, že viac ako 90% statických inštrukcií generuje menej ako 64 hodnôt. Tieto výsledky ukazujú, že predpovedanie hodnôt (value prediction) je použiteľné v praxi.

2.7 Záver

V predchádzajúcich sekciách sme opísali niekoľko metód zvyšujúcich výkon procesora zavádzaním paralelizmu - vykonávania viacerých inštrukcií v jednom takte. Pipelining rozdelí vykonávanie inštrukcie na niekoľko častí, z ktorých každú vykonáva nezávislá jednotka procesora. Inštrukcia potom 'tečie potrubím' a je postupne vykonávaná. Superscaling používa viac pipeline-ov, čím umožňuje vykonávanie viacerých nezávislých inštrukcií naraz. Out-of-order execution pozerá dopredu v toku inštrukcií a vykonáva inštrukcie skôr ako sú na rade. Speculative execution predpovedá výsledky inštrukcií a vykonáva inštrukcie špekulatívne – s predpovedanými hodnotami. Superscaling, out-of-order execution a speculative execution sú na sebe nezávislé techniky, ktoré môžu byť kombinované medzi sebou. Speculative execution využíva na predpovedanie výsledkov inštrukcií value prediction. Value prediction predpovedá výsledok inštrukcie na základe jej predchádzajúcich výsledkov. Branch prediction je špeciálna časť value prediction, ktorá predpovedá výsledok vetviacich inštrukcií.

procesor	**a**	**b**
pôvodný procesor	27	34
+ pipeline	15	19
+ superscaling (2 pipeliny)	14	18
+ superscaling + out-of-order execution	9	10
+ superscaling + speculative execution	11	14

Stĺpec (a) udáva počet taktov potrebných na vykonanie kódu 2.1 v prípade, že podmienka v I5 bude splnená, v stĺpci (b) je udaný počet taktov v prípade, že splnená nebude.

Tabuľka 2.1: Porovnanie rýchlostí procesorov s rôznymi zlepšeniami

Na začiatku sme skonštruovali virtuálny procesor, v ktorom sme vykonávali istú časť kódu. Tento procesor sme postupne vylepšovali spomenutými technikami. Výsledky dosiahnuté pri jednotlivých zlepšeniach sú uvedené v tabuľke 2.1. Všimnime si, že procesor, ktorý používal dva pipeliny a out-of-order execution s dokonalou branch prediction bol oproti pôvodnému procesoru v prípade (a) rýchlejší o 300%. V prípade (b) bolo toto zrýchlenie takmer 350%.

Kapitola 3

RISC

V predošlých odsekoch sme spomenuli inštrukčnú sadu procesorov INTEL. Potom sme zaviedli nové inštrukcie MMX.

Pripomeňme, ako sme sa k MMX inštrukciám 'dostali': analýzou multimediálnych aplikácií sme zistili, že často vykonávajú operácie na maticiach 2×2 a 3×3, napr. sčítanie či násobenie matíc. Príslušný program teda opakovane na jednotlivých prvkoch vykonal niekoľko základných operácií. Opakované volanie inštrukcií, síce jednoduchých, však zbytočne plytvalo časom- opakovane prebiehalo načítanie týchto inštrukcií z pamäte a dekódovanie; navyše, niektoré z operácií by sa mohli vykonávať paralelne. Preto sa výhodnejšie ukázalo zostrojiť pre často používané operácie obvody realizujúce ich. MMX inštrukcia potom aktivovala príslušný obvod. Ušetrili sme čas na dekódovanie a čítanie z pamäte pre viacero inštrukcií, prípadne sme viacero operácií vykonali paralelne, čím sme sa mohli dostať až k veľmi rýchlym inštrukciám a naše programy sa výrazne zrýchlili.

Toto teda bola jedna cesta zvýšenia rýchlosti procesora- nahradenie softwarového programu 'hardwarovým programom', resp. hardwarovo realizovať často používaný kód. Nemusia to byť len MMX inštrukcie, takéto inštrukcie má už sada 80286 - napr. inštrukcia PUSHA ukladajúca do zásobníka obsahy *všetkých* registrov.

Znie to lákavo, no má to aj svoje nevýhody. Inštrukcie sa stávajú priveľmi zložitými a preto namiesto hardwarovej CPU treba použiť mikroprogramovú. Zložitú inštrukciu nahradíme postupnosťou mikroinštrukcií - ktoré však jednak treba dekódovať, a tiež je treba taktovať mikrokroky v rovnakej frekvencií, aj keď niektoré sú rýchlejšie.

Navyše, experimentálne sa zistilo, že v programoch sa používajú len v 20% prípadoch zložité inštrukcie a až v 80% prípadoch inštrukcie jednoduché. Komfortnosť sa teda aj tak dostatočne nevyužila a pritom spôsobovala nutnosť pomalej mikroprogramovej CLU. Na tomto poznatku je založená architektúra RISC (Reduced Instruction Set Computer), ktorá uvažuje jednoduchú sadu inštrukcií, vykonateľných v jednom takte a realizovateľných bez používania mikrokódu.

3.1 Inštrukčná sada procesorov RISC

Všetky inštrukcie majú rovnakú dĺžku, čo umožňuje načítať ju z pamäte v jednom hodinovom cykle.

Inštrukčná sada zahŕňa:

- *inštrukcie, ktoré pracujú s pamäťou*

procesory RISC používajú na prácu s pamäťou len inštrukcie LOAD a STORE. Žiadna iná inštrukcia nemôže ako vstupný či výstupný argument použiť pamäťovú bunku.

- *aritmetické inštrukcie*

  zahrňujú logické operácie, posuvy a celočíselnú aritmetiku (aj násobenie a delenie).

- *inštrukcie vetvenia a skoku*

  - testovanie registrov ($= 0$, < 0, > 0, $\neg = 0$, párny, nepárny, a pod ...)
  - inštrukcie skoku (podmienený a nepodmienený)
  - volanie a návrat z procedúry

- *riadiace inštrukcie* a prípadne aj

- *inštrukcie reálnej aritmetiky*

- *MMX inštrukcie*

  a ďalšie.

Vidíme, že RISC-sada môže zahŕňať aj komplexnejšie inštrukcie; ako MMX či reálnu aritmetiku. Tieto, hoci sú popísané sekvenčným algoritmom sa dajú aspoň čiastočne sparalelniť a 'napevno' zadrôtovať. Taktiež, významné je, že inštrukcie (s výnimkou LOAD a STORE) nepracujú s pamäťou. Čítanie alebo zápis do pamäťe trvá vždy niekoľko hodinových cyklov.

3.2 Porovnanie RISC a CISC

3.2.1 Filozofia CISC

Protikladom k 'RISC-filozofii' je až doteraz spomínaná filozofia s názvom CISC - mať čo najkomplexnejšiu sadu inštrukcií, realizujúcich aj tie najzložitejšie operácie. Názov CISC je skratkou slov Complet Instruction Set Computer.

3.2.2 Porovnanie RISC a CISC

RISC

1. jednoduché inštrukcie vykonateľné v jednom cykle - realizovateľné bez použitia mikroprogramovej riadiacej logiky
2. komunikácia s pamäťou len pomocou inštrukcií LOAD a STORE
3. vysoko zreťazené (highly pipelined)
4. inštrukcie sú vykonávané harvérovo - inštrukcie sú jednoduché, preto sa dajú ľahko hardwarovo realizovať, bez použitia mikroprogramovania
5. pevný formát inštrukcií
6. málo adresných módov - súvisí s používaním pamäte len cez vyhradené inštrukcie LOAD a STORE

7. zložitosť v kompilátore - RISC-ovské procesory používajú rôzne techniky vykonávania programu (ako napr. pipelining, špekulatívne vykonávanie a pod.). Kompilátor musí vzhľadom na ne optimalizovať kód, aby bežal čo najrýchlejšie
8. veľa registrov
   - registre sú všeobecné
   - pristupuje sa k nim rýchlejšie ako k pamäti

CISC

1. zložené inštrukcie vykonávané na viac cyklov - dôsledok snahy zrýchliť a sprehľadniť program v strojovom kóde zahrnutím zložitejších operácií do inštrukčnej sady
2. každá inštrukcia môže komunikovať s pamäťou
3. málo zreťazené - ak sa používa mikrokód, je ťažké zabezpečiť, aby sa v jedna inštrukcia vykonala v jednom cykle
4. inštrukcie sú vykonávané mikroprogramom - ľahko sa realizuje spätná kombatibilita
5. variabilný formát inštrukcií, premenlivá dĺžka
6. veľa adresných módov
7. zložitosť v procesore - inštrukcie popísané mikroprogramami uloženými v pamäti ROM
8. málo registrov
   - nie je ich potrebné mať veľa, lebo každý vie pracovať s pamäťou
   - väčšina z nich je špecializovaná

3.3 Výhody a nevýhody procesorov RISC

Výhody RISC architektúry

- *Rýchlosť.* Jednoduchá inštrukčná sada, pipelining a superskalárny dizajn RISC procesora spôsobuje výrazné zvýšenie výkonu oproti 'porovnateľne zložitému' CISC procesoru (t.j. pracujúcom na tej istej frekvencii a skladajúcom sa z rovnakého počtu hradiel za použitia rovnakej polovodičovej technológie).
- *Jednoduchý hardware.* Pretože inštrukčná sada RISC procesora obsahuje iba jednoduché inštrukcie pomerne ľahko realizovateľné a nevyžadujúce mikrokód, realizačné obvody sú jednoduché a na čipe zaberajú málo miesta.
- *Kratšia doba vývoja.* RISC procesory sú jednoduchšie ako CISC, preto sa môžu vyvíjať oveľa rýchlejšie, konštruktéri môžu v kratšej dobe začať využívať nové technológie– čo vedie k väčším výkonnostným skokom medzi generáciami procesorov.

Nevýhody a možné problémy

- *Kvalita kódu.* Výkon RISC procesora závisí najmä od kódu, ktorý sa vykonáva. Keď programátor alebo kompilátor neoptimalizuje kód, procesor bude často stáť, napríklad čakať na výsledok inštrukcie, kým bude môcť spracovať nasledujúcu.

  Pretože pravidlá pre optimalizáciu kódu môžu byť veľmi zložité, väčšina programátorov používa vyššie programovacie jazyky (C++, Java) a necháva optimalizáciu kódu na kompilátor.

- *Ladenie (debugging).* Optimalizácia kódu môže viesť k ťažkostiam pri ladení. Ak je instruction shedulling ako aj ďalšia optimalizácia kódu vypnutá, inštrukcie sa vykonávajú tak, ako za sebou nasledujú v programe. V opačnom prípade sa nemusia vykonávať v tom istom poradí (superskalárna architektúra, vykonávanie mimo poradia...)

- *Zväčšovanie kódu (Code expansion).* Kým CISC procesory vykonajú nejakú väčšiu, komplexnejšiu akciu jednou inštrukciou, na tú istú akciu môžu RISC procesory potrebovať viac jednoduchých inštrukcií.

- *Dizajn systému.* RISC systémy potrebujú veľmi rýchle pamäťové systémy, aby mali včas dostatok inštrukcií, ktoré by mohli spracovať (pipeline). RISC systémy preto majú veľké CACHE-pamäte, obyčajne umiestnené na čipe procesora. Pripomeňme, že tieto sú známe ako first-level cache (CACHE prvej úrovne). Niektoré systémy majú tiež CACHE-pamäte, ktoré už nie sú zabudované priamo v čipe procesora, nazývané second-level cache (CACHE druhej úrovne).

3.4 Využitie RISC procesorov

Ešte v nedávnej minulosti neboli RISC procesory veľmi rozšírené. Dôvodom bola vysoká cena a nekompatibilita s najrozšírenejšou platformou Intel x86. Preto sa využívali hlavne v pracovných staniciach.

Nekompatibilnosť však pomaly prestáva byť problémom – jednak sa RISC-ovské procesory presadzujú vďaka svojmu výkonu a môžeme sa s nimi stretnúť čoraz častejšie; a ďalej je to nový trend v operačných systémoch znižujúci závislosť od hardwaru. Príkladom môže byť Windows NT, ktorý obsahuje HAL (hardware abstract layer - vrstva pre abstrakciu na hardvéri). Táto vrstva do značnej miery znižuje prácu nutnú k preneseniu operačného systému na novú architektúru tak, že nahrádza závislosť na hardvéri za neveľké softvérové rozhranie. Kód aplikácie sa tak stáva izolovaný od hardvéru.

RISC procesory stále majú 20% náskok v oblasti vyžadujúcich vysoký výkon v aplikáciách s pohyblivou rádovou čiarkou, ako napr. finančné a obchodné aplikácie, strojárske aplikácie, vedecké aplikácie a podobne. Môžme sa s nimi stretnúť aj v grafických (napr. firmy SGI) či v hracích konzolách (napr. Nintendo).

Ceny RISC procesorov klesajú. Zároveň, výrobcovia CISC procesorov sa inšpirujú RISC architektúrou. Nové procesory sú vlastne 'kríženci' medzi RISC a CISC.

Časť V

Pamäte

V tejto kapitole sa budeme zaoberať ďalšou základnou časťou počítača– pamäťou.

Najskôr uvedieme, čo pamäť je a na čo slúži. Ukážeme príklad jednoduchej realizácie pamäte. V ďalších troch kapitolách uvedieme parametre, ktorými možno pamäte charakterizovať a rozdelíme ich podľa viacerých kritérií. V piatej kapitole sa budeme bližšie zaoberať polovodičovou pamäťou, fyzikálnymi princípmi uchovania informácie, realizáciou pamäťových členov i pamäte a tiež princípmi 'pevných' pamätí ROM.

V šiestej kapitole opíšeme najčastejšie používané (či už dnes alebo v minulosti) technológie pamätí, ktorými sú: dierové, magnetické, feritové, bublinové, magnetooptické, optické a ďalšie pamäte.

V siedmej kapitole spomenieme 'laboratórne' technológie, t.j. technológie, ktoré sú v štádiu vývoja a nájdeme ich skôr v laboratóriách. Spomedzi mnohých spomenieme pamäte kryogénne a holografické.

Na záver podrobnejšie opíšeme niektoré pamäťové štruktúry: zásobník a frontu, asociatívnu pamäť a CACHE.

Kapitola 1

Pojem pamäti

Pamäť (alebo *pamäťové zariadenie*) je časť počítača, ktorá slúži na uchovávanie informácie. Ukladáme sem program, údaje, medzivýsledky, výsledky. Na základe programu a vstupných dát dokáže procesor vykonávať činnosť popísanú programom.

V II.časti sme najskôr zostrojili *klopný obvod*, ktorý vedel uchovať informáciu veľkosti jedného bitu, a potom sme spojením viacerých klopných obvodov zostrojili *register*, ktorý dokázal uchovať jedno slovo dĺžky n bitov. Podobnou metódou, spojením m - registrov, by sme z klopných obvodov dokázali zostrojiť pamäť o veľkosti $m * n$ bitov.

Pamäť pozostáva z *pamäťových členov*. Pamäťovým členom nazveme ľubovoľné zariadenie, ktoré dokáže uchovávať informáciu. Môže teda nadobúdať viacero vnútorných stavov, pričom vieme testovať, či je v určitom stave (čítanie informácie) a tiež ho vieme ho určitého stavu uviesť (zápis informácie). V svojom stave zotrváva až do ďalšej požiadavky na zmenu stavu.

Najjednoduchšou informáciou je jeden *bit*, ktorý môže mať hodnotu 0 alebo 1. Skupinu n bitov nazývame *slovo*. Pamäť je členená na m buniek z ktorých každá má veľkosť n bitov. Tieto bunky tiež budeme označovať ako slová a hovoriť, že pamäť obsahuje m n-bitových slov. Kapacita pamäte, t.j. celkový počet bitov zapamätateľnej informácie je udaná súčinom $m * n$.

Slová sú najjednoduchšie prvky (bunky) pamäte, s ktorými procesor môže pracovať (čítať a zapisovať)[1].

[1] prirodzene, procesor má aj inštrukcie pracujúce s jednotlivými bitmi slova, no tieto inštrukcie procesor realizuje pomocou operácií čítania/zápisu celého slova z/do pamäte.

Kapitola 2

Parametre pamätí

Jednotlivé pamäťové zariadenia možno charakterizovať nasledovnými parametrami:

1. *celková kapacita* - veľkosť informácie ktorú v pamäti možno uchovať.
2. *organizácia* - počet bitov v slove, počet slov.
3. *rýchlosť* - rozlišujeme *vybavovaciu dobu*, čo je doba od povelu čítania až po obdržanie informácie a doba cyklu, čo je minimálna doba medzi dvoma po sebe idúcimi povelmi pre čítanie alebo zápis do pamäte.

   Tieto dve údaje sa nemusia zhodovať, doba cyklu môže byť dlhšia ako vybavovacia doba. Ak výrobca udáva 'rýchlosť pamäte', treba zistiť, o ktorý z uvedených parametrov sa jedná. Aby sme posúdili výkon pamäte, potrebujeme oba: jeden udáva rýchlosť splnenia požiadaviek 'ojedinelých' a druhý 'častých'.
4. *energetická nezávislosť* - je na uchovávanie informácie potrebný energetický zdroj, alebo je informácia energeticky nezávislá?
5. *cena* - vyjadrená v cene za jednotku informácie[1].

Pamäťové obvody majú tiež nasledovné vlastosti:

- *prístupový čas vzhľadom na adresu* (*adress acces time*) je oneskorenie od ustálenia adresy na vstupe pamäte až po ustálenú výstupnú hodnotu.
- *prístupový čas vzhľadom na odblokovanie* (*chip enable acces time*) je oneskorenie od ustáleného odblokovania vstupu až po ustálenú výstupnú hodnotu.

[1] napr. cena za jeden megabajt

Kapitola 3

Rozdelenie pamätí

Pamäte možno rozdeliť podľa viacerých kritérií:

1. Podľa *prístupu k uloženým informáciám* delíme pamäte na:

   - *RAM* (*Random Acces Memory*) - pamäť s ľubovoľným prístupom. Každé slovo pamäti má priradenú adresu, pomocou ktorej je jednoznačne určený. Vyhľadanie ľubovoľného slova trvá rovnaký čas.
   - *SAM* (*Sequential Access Memory*) - pamäť so sekvenčným prístupom. Prvky majú tiež svoje adresy, ale čas prístupu nie je rovnaký pre všetky prvky, závisí od umiestnenia prvku v pamäti.
     Ako príklad nám môže poslúžiť čítanie údajov z diskety. Prečítanie údajov z bloku nad ktorým sa práve nachádza čítacia hlava je rýchlejší ako prístup z iného bloku, nad ktorý sa hlava musí najskôr presunúť, čo trvá určitý čas.
   - *CAM* (*Content Access Memory*) - pamäť s asociatívnym prístupom. Pamäťový prvok sa vyhľadáva podľa určitej vlastnosti hľadanej informácie, najčastejšie podľa známej časti jeho obsahu. Práve odtiaľto pochádza názov - podobnosť s pamäťou človeka, ktorá je tiež asociatívna. Napríklad, na informácie ktoré vieme o paradajke si 'spomenieme' rovnako po počutí slova 'paradajka' ako po slovách 'červený plod zeleniny, veľmi chutný'.
     Asociatívnej pamäti sa zadá obsah niektorých bitov hľadaného prvku, pamäť nájde prvok, ktorý má na zadaných pozíciách bity zhodné so zadanými, na ostatných môže mať ľubovoľný obsah. Tento typ pamätí má uplatnenie v niektorých špeciálnych aplikáciách (napr. realizácia CACHE) a bližšie si ho opíšeme v samostatnej kapitole.

2. Podľa *stálosti uložených údajov* delíme pamäte na:

   - *trvalé* alebo *permanentné pamäte* - na zachovanie údajov nie je potrebný žiadny vonkajší zdroj energie. Príkladom sú pevné disky alebo gramofónové platne. Sem zaraďujeme aj pamäte, ktoré svoje údaje 'stratia' až po 'veľmi dlhom čase' (napr. po niekoľkých rokoch), ako napr. prepisovateľné CD-ROM.
     - Bežné pamäte ROM (nazývané aj *M-ROM*): programujú sa priamo počas výroby mechanickými maskami (odtiaľ názov), ich obsah je teda určený výrobcom. Ich výroba je rentabilná až pri výrobe väčšieho počtu kusov (cez tisíc).

- *elektricky programovateľné pamäte PROM*: môže si ich naprogramovať priamo užívateľ, zapisuje sa pomocou elektrického signálu, zapísať je možné len raz. Sú výhodné pri menšom počte kusov (špeciálne úlohy).
- *pamäte EPROM*: možno ich podobne ako elektricky programovateľné pamäte naprogramovať, čítať z nich; navyše je možné ich vymazať pomocou silného ultrafialového žiarenia a opätovne do nich zapisovať. Stálosť údajov nezávisí od napájacieho napätia.
- *pamäte REPROM* (tiež *EAROM- Eletricity Alterable ROM*): sú elektricky reprogramovateľné pamäte ROM. Niektoré uchovajú údaje len na istý čas, iné na neobmedzene dlhú dobu.
- *programovateľné logické polia PLA* (*Programmable Logic Arrays*): sú pamäte typu ROM, upravené na realizáciu logických funkcií. Obsahujú programovateľné matice pre členy AND, pre členy OR a pre preklápacie obvody umožňujúce generovať sekvenčnú logiku. Logické funkcie vieme tiež realizovať pomocou pamätí ROM, avšak pri nich pre n-vstupov popíšeme všetky možné výstupy (ktorých je 2^n. Naproti tomu PLA dokážu skupinu vstupov ignorovať, dať viacerým vstupom rovnaký výsledok.

Na záver treba dodať, že väčšinou sa uvedené typy realizujú polovodičovými obvodmi a preto toto delenie patrí skôr do už spomínanej samostatej časti o polovodičových pamätiach. Uvádzame ich však tu, pretože uvedené pojmy sú všeobecné, nemusia sa vzťahovať len na polovodičové pamäte.

- *dočasné pamäte* - po vypnutí elektrického prúdu (alebo iného zdroja energie) sa uložená informácia 'stratí'. Ako príklad môžu slúžiť polovodičové pamäte RWM.

  Môžu byť *statické* alebo *dynamické*. *Dynamické pamäte* sú schopné uchovávať informáciu len veľmi krátky čas (niekoľko milisekúnd) a po tomto čase treba obsah pamäte obnoviť, t.j. znova zapísať do pamäťových buniek ich obsah. Súčasťou pamäte sú obvody obnovujúce niekoľkokrát za sekundu jej obsah. V čase obnovovania obsahu nemožno s pamäťou pracovať, procesor i zbernica sú blokované, navyše, je pomalšia ako statická. Napriek tomu počítače bežne používajú dynamickú polovodičovú pamäť. Má dve veľké výhody. Jednak na menšiu plochu dokážeme sústrediť viac pamäťových obvodov, jednak je lacnejšia. Preto ju používame na bežné a statickú polovodičovú pamäť na špeciálne účely (kde potrebujeme rýchlu pamäť s neveľkou kapacitou).

  Najčastejšie o tomto delení hovoríme v súvislosti s polovodičovými pamäťmi. Ale aj iné typy pamätí môžu byť dočasné, buď statické alebo dynamické.

3. Podľa *možnosti čítania a zapisovania údajov*:

- do pamätí typu *RWM* (*Read-Write Memory*) možno údaje zapisovať aj čítať. Používajú sa na bežné účely. Rozlišujeme:
  (a) *pamäte s rýchlym čítaním a rýchlym zápisom* - využívajú sa najmä v hlavných operačných pamätiach.
  (b) *pamäte s rýchlym čítaním a pomalým zápisom* - používame, ak je nutné rýchle čítanie údajov, pričom sa do pamäte zapisuje zriedkavo. (Označujú sa RMM- Read Mostly Memories).

- z pamätí typu *ROM* (*Read-Only Memory*) možno len čítať, informácia ktorá je v nich uložená sa už nedá zmeniť. Zachovávajú si svoj obsah trvalo. Ukladajú sa do nich informácie, ktoré sa často používajú: grafický tvar znakov, matematické tabuľky funkcií, generátory logických funkcií, prípadne základný operačný systém.

Kapitola 4

Triedy pamätí

Ideálna pamäť by mala mať čo najväčšiu kapacitu, najvyššiu rýchlosť, čo najpohodlnejší spôsob manipulácie s údajmi, energetickú nezávislosť informácie (informácia sa nestratí pri neprítomnosti vstupného napätia) a čo najnižšiu cenu. Pretože niektoré požiadavky sa navzájom vylučujú (napr. veľká kapacita a nízka cena), konštrukcia takej pamäte je nereálna.

Konštrukcia ideálnej pamäte, ktorá by zrýchlila beh všetkých aplikácií je síce nereálna, napriek tomu je možné zostrojiť pamäťový systém zrýchľujúci prácu veľkej časti aplikácií.

Potrebné je uvedomiť si, že pamäte sa používajú na rôzne úlohy. Jednotlivé skupiny úloh kladú rozdielne požiadavky na parametre pamätí, napríklad zálohovanie údajov do archívu nepotrebuje ani tak rýchlu pamäť, ako skôr pamäť energeticky nezávislú a s veľkou kapacitou. Naopak, pamäť kde je uložený vykonávaný program a potrebné dáta nemusí byť energeticky nezávislá, ani mať veľkú kapacitu (v porovnaní s predchádzajúcou), ale mala by byť čo najrýchlejšia.

Existuje viacero technológií pamätí a každá z nich 'vyniká' určitými parametrami na úkor iných.

Jednotlivé pamäťové zariadenia možno teda rozdeliť do viacerých skupín (tried), ktoré sa líšia svojim účelom a z toho vyplývajúcich požiadaviek na kapacitu a rýchlosť. Pre tieto triedy pamätí použijeme odlišné technológie.

Základné triedy sú: *registre procesora, vyrovnávacia pamäť, hlavná* (operačná) *pamäť* a *vonkajšie* (periférne) *pamäte*.

- *Registre procesora* sú súčasťou procesora. Ukladajú sa sem základné údaje potrebné na vykonávanie programu. Ich funkcia je bližšie opísaná v kapitole o procesoroch. V porovnaní s ostanými skupinami má najvyššiu rýchlosť a najnižšiu kapacitu.
- *Vyrovnávacia pamäť* slúži na dočasné prechovávanie údajov, ktoré by sa inak čítali z hlavnej pamäte. Je asi 10 krát rýchlejšia ako hlavná. Jej kapacita je zlomkom hlavnej pamäte. Bližšie si o nej povieme v samostatnej časti tejto kapitoly.
- *Operačná pamäť*, nazývaná tiež *hlavná pamäť* (ang. *main memory*) tvorí súčasť počítača. Obsahuje práve vykonávaný program a jeho pracovné údaje. Vyžaduje sa rýchlosť a primeraná kapacita.

  Hlavné pamäte súčasných počítačov majú kapacitu od 64 Megabajtov. Realizujú sa polovodičovými obvodmi. Väčšinou sa jedná o pamäť typu RAM, čiže pamäť s priamym prístupom (na každú adresu sa dá dostať v rovnakom čase).

Buď celá alebo takmer celá je typu RWM (časť z nej môže byť typu ROM). Ako sme uviedli, v pamäti ROM sa môže nachádzať jednoduchý operačný systém, programy na obsluhu periférií, alebo často používané funkcie.

Procesor pracuje s operačnou pamäťou priamo. Obsahuje inštrukcie pre zápis a čítanie z operačnej pamäte. S periférnou pamäťou procesor priamo nepracuje, považuje ju za V/V zariadenie.

Realizácia polovodičovými obvodmi spôsobuje, že pamäť má značnú rýchlosť, teda aj výkonnosť. Pre ilustráciu, pri bežných typoch je možné dosiahnuť vybavovaciu dobu pod 70 ns. Ďalším faktorom podmieňujúcim výkonnosť je kapacita pamäte.

- *Periférna pamäť* je pamäťové zariadenie priamo spojené so základnou jednotkou počítača, umožňujúce čítanie a zapisovanie veľkého množstva údajov. Sem ukladáme dáta, s ktorými program nepracuje, alebo pracuje zriedka. Objem dát je príliš veľký, aby sa zmestil do hlavnej pamäte. Pamäť má mať veľkú kapacitu, nemusí byť taká rýchla ako hlavná pamäť, má byť energicky nezávislá a samozrejme, má mať čo najnižšiu cenu. Môže mať *vymeniteľný nosič údajov* (napr. disketová jednotka, nosiče - diskety) a vtedy sa označuje aj ako *vstupno - výstupná jednotka.*

  Existuje množstvo technológií periférnych pamätí, s odlišnými parametrami a dobou vzniku. Medzi najznámejšie patria: páskové, diskové, disketové pamäte, bublinové pamäte, optické pamäte a iné. Podrobne sa im budeme venovať v ďalšom texte.

Kapitola 5

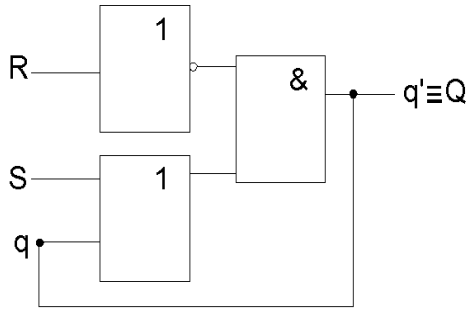
Polovodičové pamäte

5.1 Pamäťové členy polovodičovej pamäte

Polovodičové pamäte sú integrované obvody zložené z tranzistorov.

Pamäťovú bunku uchovávajúcu jeden bit môžeme vytvoriť pomocou už spomenutého klopného obvodu, vytvoreného prepojením dvojice tranzistorov (čím vytvoríme *statickú pamäťovú bunku*) alebo pomocou jediného tranzistora MOS-FET (*dynamická pamäťová bunka*). Ich ďalším spájaním môžeme vytvoriť registre uchovávajúce n-bitové slová a spojením registrov pamäťové členy s veľkou kapacitou a organizáciou RAM.

Statická pamäťová bunka (klopný obvod) bola opísaná v časti o obvodoch. Klopný obvod vytvorený pomocou hradiel je na nasledovnom obrázku.



Obrázok 5.1: Schéma RS-člena

Dynamická pamäťová bunka je realizovaná MOS-FET tranzistorom. Skratka MOS (metal-oxid-semiconductor) popisuje jeho jednotlivé vrstvy- kov (Al), izolant (SiO_2) a polovodič (Si) a skratka FET (field-effect-tranzistor) udáva, že tento tranzistor je riadený elektrickým poľom. Tento tranzistor je unipolárny. Jeho schému a popis vlastností čitateľ môže nájsť v literatúre z oblasti elektroniky (viď zoznam literatúry).

5.2 Realizácia pamäte RAM

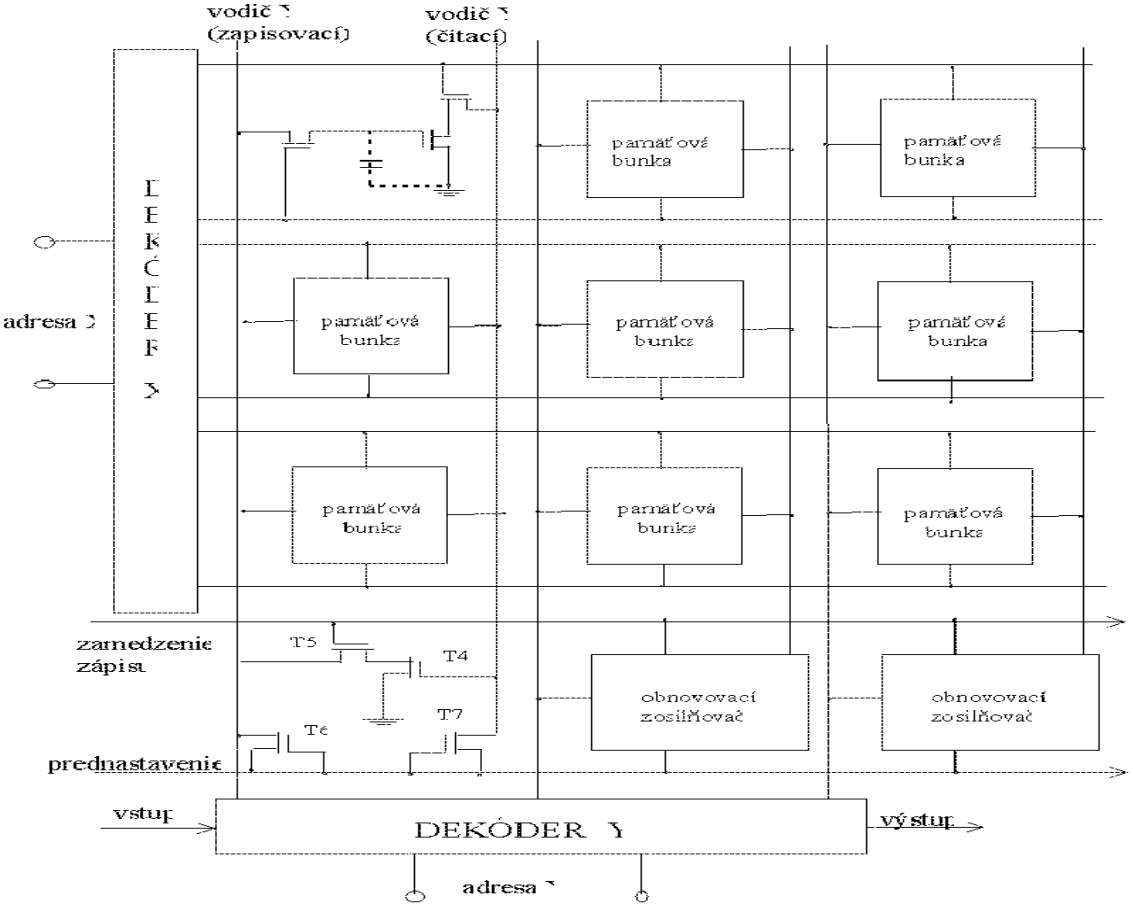
Predpokladajme, že chceme realizovať pamäť o veľkosti k slov po l bitov. Navrhneme príslušný obvod. K dispozícií máme základné pamäťové členy (na uchovávanie jedného bitu), pričom nás nezaujíma, ako sú realizované a či sú statické alebo dynamické.

Naša pamäť by mala umožniť čítanie a zapisovanie informácie– pamäťový obvod môže mať napr. nasledovné vstupy:

S	Selekcia pamäťového obvodu
R	Čítanie
W	Zápis
I	Údaj na zápis
$a_{n_1} \dots a_0$	Adresa pamäťovej bunky

Princíp realizácie je čitateľovi dozaista zrejmý: súčasťou obvodu bude dekóder s toľkými výstupmi, koľko má obvod pamäťových členov[1]. Na jednotlivé výstupy dekódera sú pripojené jednotlivé pamäťové členy. Celá adresa je pretransformovaná dekóderom a vyberá práve jeden MEM obvodov. Návrh príslušného obvodu prenechávame na čitateľa.

Pre vysokokapacitné pamäte je však priestorová zložitosť dekóderu exponenciálna,dekóder sa stáva zložitým. Skúsme preto pamäťový obvod realizovať nasledovne:



Obrázok 5.2: Realizácia pamäte pomocou dvoch dekóderov

Ako vidno zo schémy, druhé riešenie určuje polohu príslušného pamäťového člena pomocou dvoch dekóderov.

[1] t.j. koľko slov má pamäť

Kvôli prehľadnosti znázorňujeme len princíp adresácie pomocou dvoch selektorov, aj keď prirodzene, môžeme pridávať ďalšie dekódery[2], čím ďalej znížime nároky na ich priestorovú zložitosť.

Pri doterajších úvahách sme vytvárali pamäte, kde jednou adresou sme určovali jeden bit. Ľahko však vytvoríme pamäť, kde adresa bude určovať jedno slovo (a teda aj namiesto vstupu I bude mať pamäť vstupy I_1 až I_n). Návrh opäť ponechávame na čitateľa.

[2] a vytvoriť tak n-rozmernú pamäť

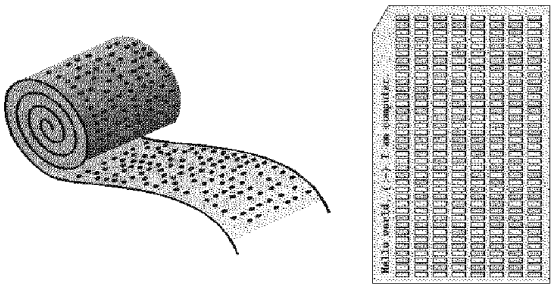
Kapitola 6

Ďalšie technológie pamätí

V predchádzajúcom texte sme sa venovali najmä polovodičovým pamätiam. V tejto kapitole popíšeme iné fyzikálne princípy uchovávania informácie a pamäťové zariadenia, ktoré ich využívajú.

6.1 Mechanický záznam

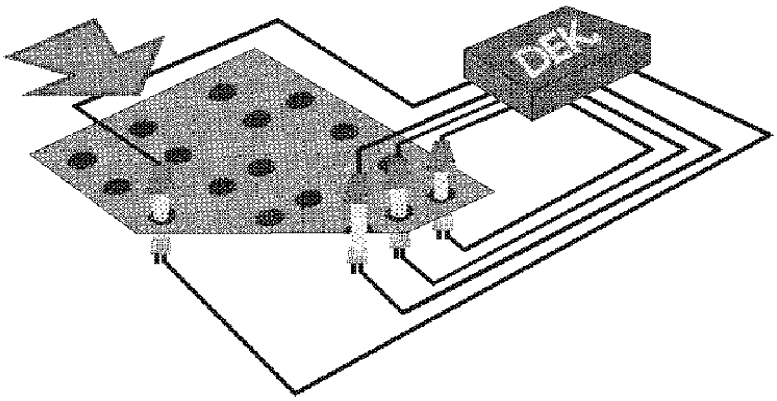
Jedny z prvých vysokokapacitných pamätí boli pamäte založené na veľmi jednoduchom princípe, technicky realizovateľnom aj mechanicky.



Obrázok 6.1: Dierna páska a dierny štítok

Pamäťovým médiom je papier (dierny štítok alebo dierna páska). Princíp zaznamenania informácie je jednoduchý: na určitých miestach pásky/štítku možu byť vyrezané otvory (diery). Tým je možné kódvať binárnu informáciu – prítomnosť či neprítomnosť značí (podľa dohody) buď jednotku – nulu, alebo nulu – jednotku. Médium je snímané pomocou fotoelektrickej diódy a zdroja svetla, pričom sa umiestni medzi nich. Dióda i zdroj sú umiestnené oproti sebe. Ak sa v médiu na mieste medzi nimi nachádza otvor, tak lúč zo zdroja ním prejde a zasiahne svetlocitlivú diódu, ktorá vyšle elektrický impulz. Ak tam otvor nie je, lúč diódu nezasiahne a tá preto elektrický impulz nevytvorí. Médium sa pohybuje tak, aby čítacia sústava mohla otestovať všetky miesta, kde podľa dohody môže byť vyrezaný otvor.

Konštrukčne sú takéto pamäte ľahko realizovateľné, no hustota záznamu je dosť nízka a navyše, pamäte sú značne pomalé. V dobe svojho vzniku však nebolo potrebné ukladať také množstvo dát ako v súčastnosti, navyše, používali sa zväčša len na archiváciu



Obrázok 6.2: Proces čítania z dierného štítku

dát. Kvôli týmto faktorom a nízkej cene zariadení i médií predstavovali najvýhodnejšiu (externú) pamäť. Používali sa až do začiatku 80.rokov, kedy sa namiesto nich začali používať magnetické pamäte.

6.2 Magnetický záznam

Medzi magnetické pamäte patria magetické pásky, bubny, pevné disky, diskety a karty. Ich spoločným znakom je fyzikálny princíp zápisu využívajúci na reprezentáciu informácie dve rozličné magnetické orientácie na magnetizovateľnej vrstve.

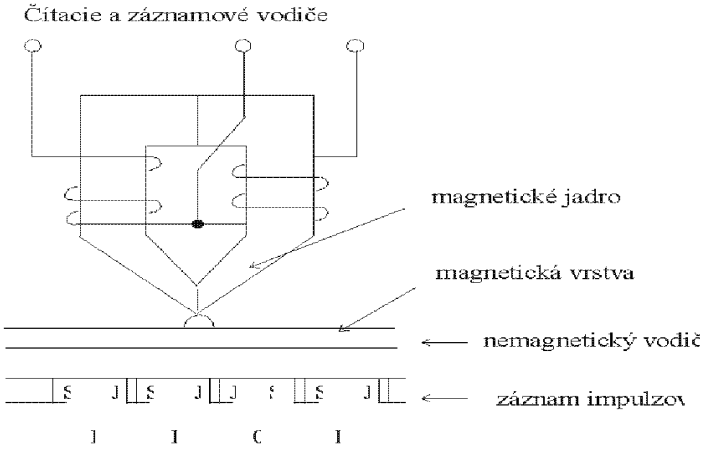
Fyzikálny princíp magnetických pamätí je nasledovný: záznamové médium je pokryté homogénnou vrstvou (Fe, Fe_2O_3 alebo CrO_2). Na zápis a čítanie slúži elektromagnetická záznamová hlava. Je to malý elektromagnet magnetizujúci úzku oblasť média. Hlava je tvorená magnetickou a elektrickou časťou- magnetickým jadrom a cievkou navinutou na jadre. V mieste dotyku s povrchom magnetického média je jadro prerušené, je tu vzduchová štrbina. V tejto oblasti dokáže vytvoriť silné magnetické pole. Hlava i štrbina majú veľmi malé rozmery, preto je plocha ktorú je schopná zmagnetizovať veľmi malá (viď animácie a obr.6.3).

Magnetická hlava magnetizuje (resp. zapisuje) pozdĺžne, v smere zápisu úzke oblasti. Na zápis jedného bitu použije dve takéto oblasti. Ak majú obe súhlasnú orientáciu, predstavujú zápis nuly. Ak majú opačnú orientáciu, zápis jednotky. Nasledujúci bit sa začína zapisovať s opačnou orientáciou (obr.6.4 (a)).

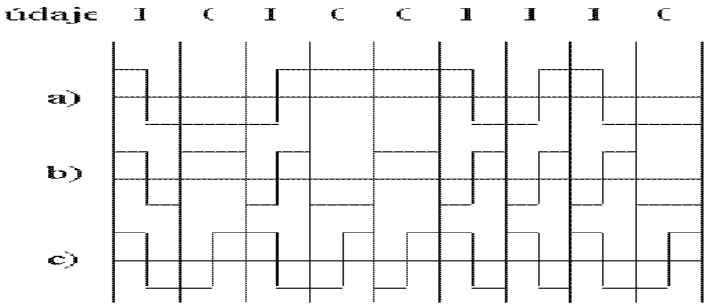
Pri čítaní spôsobujú miesta kde sa mení magnetický tok (*magnetické reverzácie*) zmenu magnetického poľa. To spôsobí vznik prúdového impulzu na cievke, ktorý sa ďalej zosilní elektronickými zosilňovačmi. Hlavu teda treba nastaviť na začiatok záznamu, pohybovať médiom a sledovať, či sa uprostred dvoch oblastí pre záznam bitu zmenil alebo nezmenil indukovaný prúd, čo predstavuje kód jednotky, resp. nuly (viď animácie).

Tento spôsob záznamu sa nazýva *pozdĺžna magnetizácia* (alebo *pozdĺžny záznam*).

Okrem tejto metódy existuje aj metóda kolmého zápisu. Tiež, existujú rôzne metódy kódovania údajov, napríklad FM, MFM, M2FM a RLL.



Obrázok 6.3: Čítacia a záznamová hlava



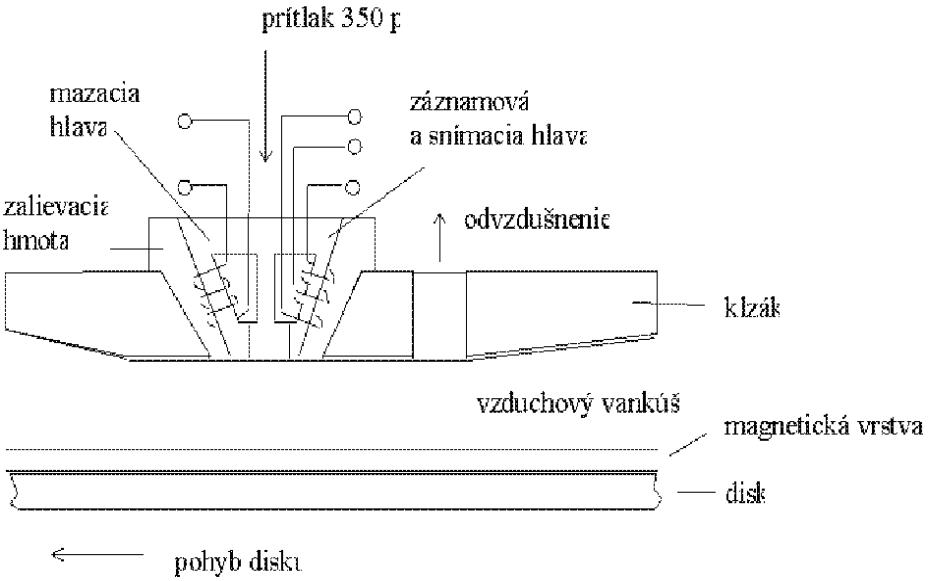
Obrázok 6.4: Kódovanie údajov

Médium rotuje (u diskov, diskiet,...) alebo sa posúva (páska) konštantnou uhlovou rýchlosťou.

Aj keď opísaný postup vyzerá komplikovane, má niekoľko výhod. Pri čítaní nastane zmena polarity po každom bite (t.j. dvoch oblastiach). V praxi rýchlosť čítania môže mierne kolísať a tiež je potrebné identifikovať začiatok záznamu. Kvôli tomu sa bežne používa synchronizačný signál– v našom systéme ho však nepotrebujeme, zosynchronizovať sa dá priamo zo záznamu (pretože po každých dvoch prečítaných oblastiach určite musí nastať zmena polarity). Na začiatok záznamu zapíšeme pevne určený počet nulových bitov, pomocou ktorých čítacie obvody správne nastavia hlavu na začiatok záznamu. Ďalšou výhodou je, že pri zázname väčšieho počtu rovnakých bitov vieme na základe zmien polarity bezchybne určiť ich počet.

Táto technológia vyžaduje úplne sterilné (bezprašné) prostredie. Pokiaľ by sa medzi záznamové médium a hlavu dostalo zrnko prachu, zápis i čítanie by bolo znemožnené. Buď sa teda vyžaduje bezprašné prostredie, alebo sa celé zariadenie ukladá do vzduchotesnej schránky.

Ďalšou požiadavkou je pritlačenie hláv k médiu. Magnetické siločiary vytvárajú v mieste štrbiny úzky zväzok, ale pokiaľ hlavu vzdialime od média, vytvoria sústredné kruhy siločiar ktoré magnetizujú väčšiu oblasť ako chceme. Nevyhnutné je, aby sa hlava dotýkala média (alebo aspoň bola len nepatrne vzdialená). U páskových mechaník páska elasticky obopína hlavu. Pružné diskety majú hlavy jemne pritláčané k diskete Pevné disky nemôžu hlavu pritláčať k médiu, lebo to sa otáča priveľkými rýchlosťami (okolo 3600 otáčok/min.) a aj ľahký dotyk s povrchom by spálil hlavu aj povrch. Namiesto toho sa disk najprv roztočí a až nad roztočený disk (tanier) sa presunú hlavy. Jeho rotáciou sa nad ním vytvorí tenká vrstva vzduchu. Hlava je na elastickom držiaku s aerodynamickým krídelkom, ktoré je navrhnuté tak, aby sa vyrovnala vztlaková sila odtláčajúca hlavu od disku s nepatrnou silou nosného ramienka pritláčajúceho hlavu k disku (viď nasl. obrázok). Takto vieme nastaviť vzdialenosť hlavu od povrchu na niekoľko mikrometrov.

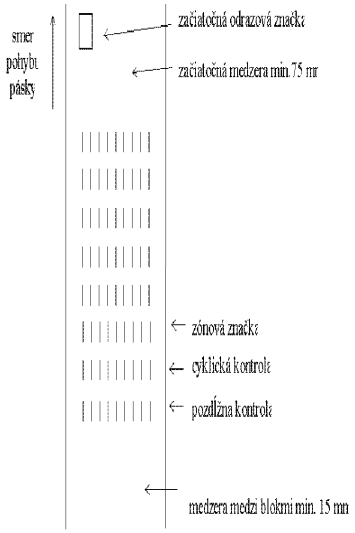


Obrázok 6.5: Pritláčanie hláv k médiu - úplná štruktúra hlavy

6.2.1 Magnetické páskové pamäte

Používali sa v počítačoch druhej a tretej generácie.

Záznamovým nosičom (záznamovým médiom) je magnetická páska. Na základnej, nosnej fólií pásky je tenká magnetizovaná vrstva, kde sa ukladajú údaje. Zapisujú a čítajú sa zvyčajne pomocou 9 hlavičiek, uložených vedľa seba. Údaje sa zaznamenajú v 9 stopách.



Obrázok 6.6: Magnetická páska (štruktúra média)

Používajú sa najmä tam, kde treba čítať a spracovávať nepretržité sledy údajov. Magnetické pásky možno ľahko vymieňať, uchovávať a transportovať.

Magnetické páskové pamäte sú konštruované tak, že magnetické pásky možno takmer ihneď zastaviť, a to aj napriek vysokým rýchlostiam posuvu; a rovnako rýchlo je možné uviesť ich do pohybu. Páska sa z cievky na cievku neprevíja priamo, ale cez pomocné zariadenie.

Údaje sú uložené v blokoch. Bloky majú priradené adresy, pomocou ktorých možno určiť blok s ktorým chceme pracovať. Rovnako, blok je najmenšia adresovateľná jednotka, pracovať je možné len s celým blokom naraz (t.j. celý blok buď čítať alebo celý blok zapísať), nemožno adresovať údaje v ňom. Na páske sú jednotlivé bloky sú oddelené medziblokovou medzerou.

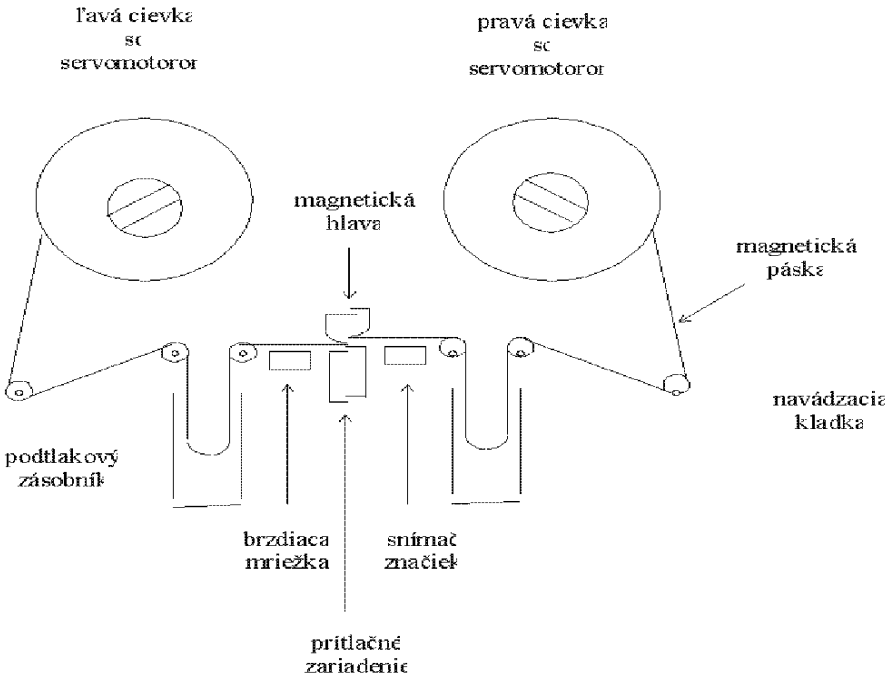
6.2.2 Kazetové páskové pamäte

Kazetová pásková pamäť používa ako nosič údajov osobitnú formu magnetickej pásky-kazetu s magnetickou páskou, ktorá je veľmi dobre známa zo spotrebnej elektroniky.

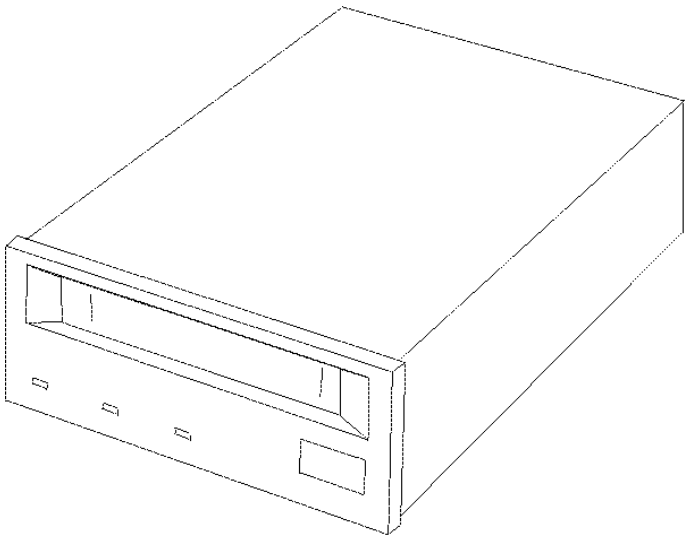
Tento spôsob zaznmenávania údajov sa presadil pri malých výpočtových zariadeniach pre jednoduchú cenu zariadenia a média.

Ako príklad uveďme páskovú jednotku DAT, ktorá bola vyvinutá pre kvalitný záznam zvuku a neskôr sa uplatnil v počítačovom priemysle. Na jedno médium (jedna záznamová páska) je možné uložiť až 2.5 GB dát.

Tento spôsob zaznamenávania údajov sa presadil pri malých výpočtových zariadeniach pre jednoduchú cenu zariadenia a média. V súčasnosti sa používa jej varianta (tzv. *streamre*) pre zálohovanie veľkého množstva dát.



Obrázok 6.7: Štruktúra páskovej jednotky

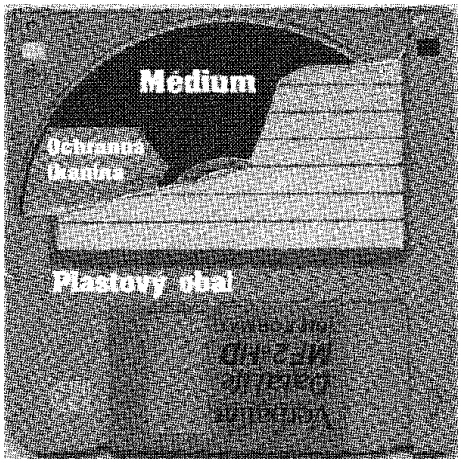


Obrázok 6.8: Pásková jednotka

6.2.3 Disketové pamäte

Prvý návrh disketovej pamäte predložila firma IBM v roku 1967. Disketová pamäť mala mnoho výhod a tak sa čoskoro začala hromadne používať.

Ako záznamové médium používa diskety. *Disketa* (alebo *pružný disk – floppy disk*) je osobitná forma magnetického disku. Je to vlastne pružná, okrúhla platňa, ktorá je na jednej alebo na obidvoch stranách pokrytá magnetizovateľnou vrstvou. Táto vrstva môže byť ešte pokrytá ochrannou vrstvou. Disketa je uložená v plastikovom obale.



Obrázok 6.9: Disketa (pružný disk). Štruktúra média

Disketa sa vkladá do čítacieho a zapisovacieho zariadenia- *disketovej jednotky*. Obsahuje jednu univerzálnu (čítaciu i záznamovú) hlavu.

Poznáme viacero druhov diskiet a disketových mechaník, podľa *priemeru diskiet* a *hustoty záznamu*.

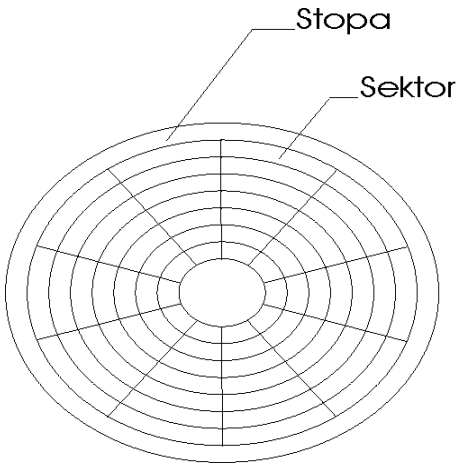
Priemer diskiet používaných v osobných počítačoch môže byť $5^1/_4$ palcové alebo $3^1/_2$ palcové; čo sa zapisuje $5^1/_4$ ″, resp. $3^1/_2$ ″ Historicky najstaršie diskety, dnes už nepoužívané, majú priemer 8 ″. Takisto sa prestali používať $5^1/_4$ palcové diskety. Tieto sú umiestnené v polotvrdom, pružnom ochrannom obale a diskety $3^1/_2$ ″ majú pevné plastikové púzdro.

Zápis a čítanie sa realizujú pomocou magnetickej hlavy, keď je disketa umiestnená do disketovej jednotky.

Disketa má jeden alebo dva povrchy (*strany*) a disketová mechanika má dve hlavy. Hlava sa pohybuje po sústredných kružniciach (*stopách* (resp. *trace*-och). Do každej z nich sa dajú zapisovať údaje. Každá stopa je rozdelená na úseky (čo sú vlastne kruhové výseky)- *sektory* (viď. nasl. obr.). Každý sektor predstavuje súvislý blok dát. Všetky stopy sa delia na rovnaký počet sektorov a všetky sektory obsahujú rovnaký počet bytov (čo je zvyčajná dĺžka uchovávateľných slov). Celkovú kapacitu diskety vyrátame ako:

$$počet\ strán * počet\ stôp * počet\ sektorov * počet\ bytov\ v\ sektore$$

Sektor je najmenšia 'jednotka', s ktoru možno operovať (zapisovať, čítať). Ak chceme zmeniť jeden byte záznamu, musíme načítať do hlavnej pamäte príslušný sektor, zmeniť v ňom jeden byte a potom ho opäť celý zapísať.



Obrázok 6.10: Logické členenie diskety

To, že disketa má v každej stope rovnaký počet sektorov, nie je najrozumnejšie využitie priestoru. Musíme počet sektorov a ich kapacitu prispôsobiť najvnútornejšej stope, ktorá má najmenší priemer. Smerom od stredu tak zanedbávame priestor, ktorý by sme mohli využiť. Dôvodom je zvolená konštantná rýchlosť otáčania média a ukladanie dát do kružníc, čo sa ľahko realizuje. Racionálnejšie využitie média majú optické mechaniky CD-ROM, ktoré ukladajú dáta 'do špirály' (viď optické pamäte).

V rámci označovania kapacít diskiet sa stalo štandartom niekoľko označení, ktoré si teraz uvedieme.

- Podľa toho, či sa informácie zapisujú na jednej strane alebo na obidvoch stranách, rozoznávame diskety:

SS	(Single Sided)	jednostranné
DS	(Double Sided)	obojstranné

- Hustota záznamu sa označuje:

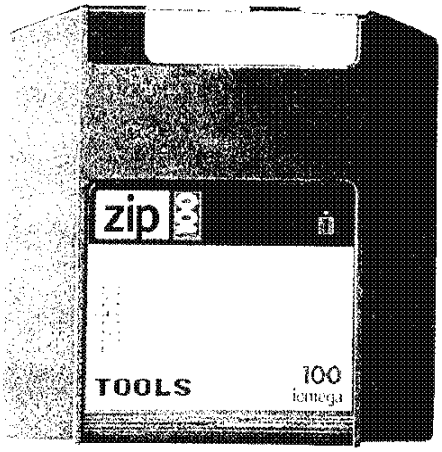
SD	(Single Density)	jednoduchá hustota
DD	(Double Density)	dvojnásobná hustota
HD	(High Density)	vysoká hustota

Štandardná kapacita diskiet je:

	DD	**HD**
$5^1/_4$ "	360 Kb	1.2 MB
$3^1/_2$ "	720 Kb	1.44 MB

Existujú aj diskety $3^1/_2$ " s kapacitou 2 MB až 4 MB (čo vyžaduje aj špeciálne mechaniky). Ich hustota sa označuje *ED* (*Extra Density*). Používajú sa už aj výmenné diskové mechaniky (s vymeniteľným médiom), kde disk je rozmeru $3^1/_2$ " a má kapacitu až 100 MB (napr. mechaniky a médiá ZIP).

Pamäť na diskete má podstatne menšiu kapacitu a nižšiu prenosovú rýchlosť ako magnetická disková pamäť. Avšak presadila sa vďaka nízkej cene a ľahkej prenositeľnosti diskiet. Preto sa uplatnila v osobných počítačoch, v malých zariadeniach na spracovanie údajov, textových systémoch a pri získavaní a príprave údajov. Stala sa súčasťou štandardného vybavenia počítača.

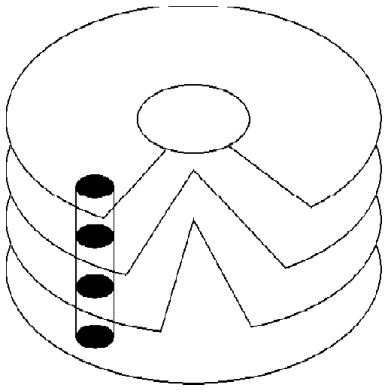


Obrázok 6.11: ZIP-médium

6.2.4 Magnetické diskové pamäte

Diskové pamäte (angl. *hard disk*-y) sú obľúbenou periférnou pamäťou osobných počítačov. Majú veľkú kapacitu a rýchly vybavovací čas, vďaka čomu sa rýchlo presadili a stali sa štandardnou súčasťou počítačových zostáv.

Nosičom údajov je sada kruhových diskov, platní pokrytých magnetickou vrstvou. Zariadenie má niekoľko hláv; disky sú umiestnené nad sebou a zaznamenávať údaje možno na obe strany diskov. S každou stranou pracuje samostatná hlava. Princípy a spôsob programovania sú podobné ako u diskiet. Novým pojmom je cylinder, ktorý označuje sektory nachádzajúce sa na rovnakých polohách no na rozdielnych diskoch (viď nasl. obr.).



Obrázok 6.12: Členenie disku. Cylindre

Čas potrebný na zápis alebo prečítanie jednej stopy je vďaka rýchlemu otáčaniu veľmi malý (rádovo desiatky milisekund).

Magnetické diskové pamäte môžu mať pevné alebo vymeniteľné disky.

6.3 Magnetické bublinové pamäte

Magnetická bublinová pamäť (*Magnetic Buble Memory*) je tiež nemechanickou pamäťou. Zvonku vyzerá rovnako ako polovodičové integrované obvody a zvyčajne sú takisto naspájkované na platničkách plošných spojov.

Ich princíp je v tom, že na tenkej vrstve magnetického materiálu sú vytvorené mikroskopicky malé 'ostrovčeky'- *magnetické domény*. Pôsobením umelo vytvoreného magnetického poľa rotujú magnetické bubliny v kruhu a prechádzajú pritom okolo záznamovej a snímacej (univerzálnej) hlavy, pomocou ktorej sa údaje môžu zapísať alebo prečítať. Existencia magnetickej bubliny značí '1', neexistencia značí '0'.

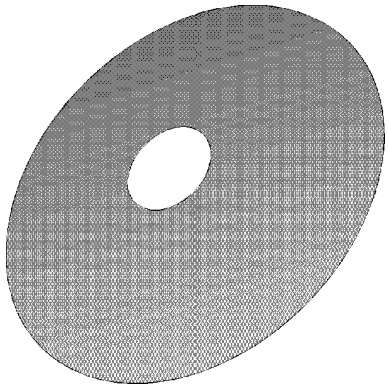
Magnetické bublinové pamäte sa môžu skladať z viacerých stoviek bublinových slučiek, z ktorých každá má vyše tisíc bublín. Magnetické bubliny sú také malé, že v jednom prvku s rozmermi 3x3 cm môže byť zaznačených niekoľko miliónov bitov.

Čas výberu údajov v magnetickej bublinovej pamäte je síce o niečo kratší ako pri najrýchlejších mechanických pamätiach, je však podstatne dlhší ako čas výberu údajov v polovodičových pamätiach. Ich výhodou je však veľká hustota zápisu údajov a tiež zachovanie informácie aj po prerušení napájania. Bublinové pamäte majú mechanickú i radiačnú odolnosť.

6.4 Optický záznam

Optický disk (označovaný skratkou CD - *compact disc*) je v súčasnosti jedným z najpoužívanejších médií. Táto technológia umožňuje 100-násobne hustejší záznam ako majú magnetické médiá, čo spôsobuje veľkú kapacitu optických médií. Dáta je možné čítať veľkou rýchlosťou. Médium i čítacia mechanika sú lacné. Majú však aj veľkú nevýhodu: sú typu ROM, t.j. údaje na ne zapísané sa nedajú prepísať. Na odstránenie tohto nedostatku však vzniklo viacero riešení. Predsa však ešte pevné disky nevytlačili a štandardnou výbavou počítača je pevný disk spolu s CD mechanikou.

V tejto kapitole porozprávame o vzniku CD, ich rozdelení, protokoloch pre jednotlivé typy CD, princípoch čítania i zápisu, podrobnejšie sa budeme venovať audio-CD a CD-ROM a spomenieme ďalšie typy. Taktiež uvedieme princípy prepisovateľných CD (CD-R, CD-RW) a spomenieme najnovšiu optickú technológiu DVD.



Obrázok 6.13: Kompaktný disk

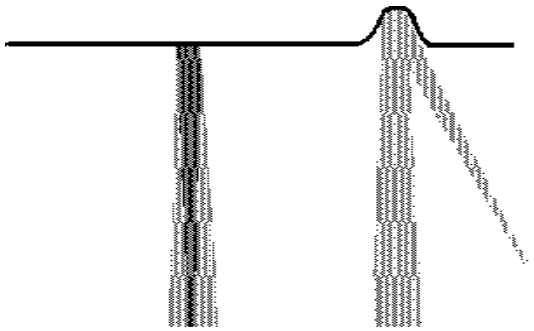
6.4.1 Vznik CD

Začiatkom 80.rokov prišla firma Philips, s revolučnou technológiou digitálneho optického záznamu videa. Neskôr spolu s firmou Sony publikovali formát pre záznam digitálneho zvuku (nazvaný Red Book), ako technológiu pre nahradenie vinylových platní. Výhodou tejto technológie oproti iným je veľká kapacita a malá chybovosť pri čítaní (viď ďalej). Táto technológia sa presadila a stala sa veľmi populárnou.

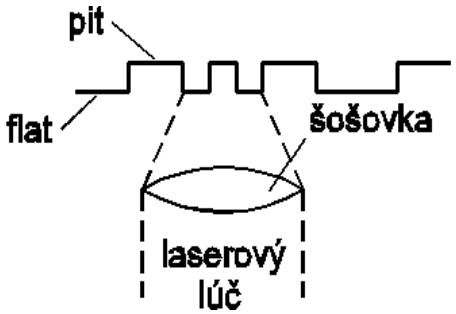
Jej výhody zaujali aj výrobcov počítačov, ktorí sa rozhodli adaptovať CD pre záznam počítačových dát. Zrodil sa nový typ média nazvaný CD-ROM (ktorého špecifikácia bola uverejnená v Yellow Book). Táto špecifikácia bola v roku 1986 rozšírená a vzniklo CD-Interactive. Vznikli aj iné, ktoré uvedieme v ďalšom texte. V roku 1995 sa objavuje technológia DVD.

6.4.2 Základné princípy

Kompaktný disk je zložený z niekoľkých vrstiev. Informácie sú zaznamené na hliníkovej odrazovej (reflexívnej) ploche. Presnejšie, nachádzajú sa na nej priehlbinky (ang. *pits*) a (rovné) plôšky (angl. *lands*). Na čítanie informácie sa používa laserový lúč, ktorý sa odráža od povrchu rôznou intenzitou podľa toho či sa lúč odrazil od priehlbinky alebo od rovnej vrstvy. Pokiaľ dopadne na plôšku, odrazí sa späť s rovnakou intenzitou. Pokiaľ dopadne na priehlbinku, rozptýli sa a späť sa odrazí lúč takmer nulovej intenzity. Odrazený lúč sa sníma fotodetektormi a na základe jeho intenzity sa vie určiť, či bola snímaná priehlbinka alebo plôška.

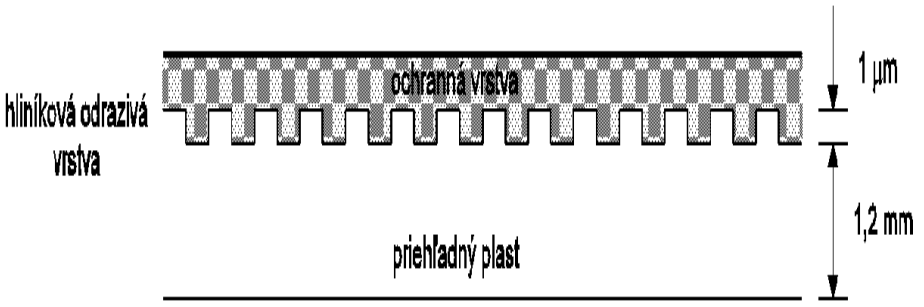


Obrázok 6.14: Čítanie - odraz lúča pri dopade na plôšku (a) a priehlbinku (b)



Obrázok 6.15: Snímacia sústava

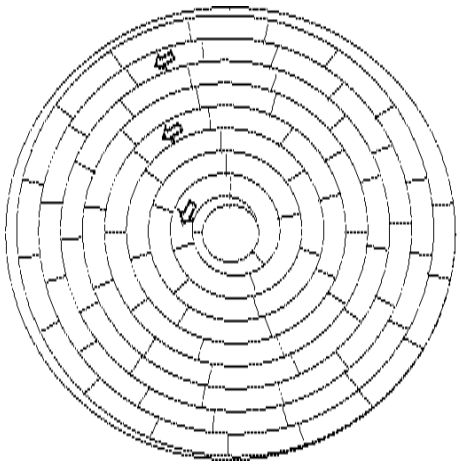
Základ média tvorí už spomínaná hliníková odrazová plocha. Tá je zo spodnej strany pokrytá priehľadnou plastikovou vrstvou, ktorá má ochrannú funkciu.



Obrázok 6.16: Vrstvy kompaktného disku

Aké sú výhody tejto technológie? Za prvé, veľká kapacita dát. Taktiež, pri načítavaní dát nedochádza ku kontaktu čítacej hlavy s povrchom média. Preto nedochádza k opotrebovávaniu média a zároveň je možné vyvinúť vysoké otáčky a tým aj vysokokú prenosovú rýchlosť. Dáta nie sú ovplyvniteľné magnetickým poľom. Plastová vrstva ich chráni pred mechanickým poškodením (napr. pred poškriabaním či dotykom), zároveň je možné vhodným kódovaním dát odstrániť prípadné chyby. O spôsoboch kódovania dát povieme v ďalšom texte.

Dáta sú usporiadané v špirále. Tým sa dosahuje lepšie využitie priestoru, ako keď by boli usporiadané v kruhoch (viď diskety).



Obrázok 6.17: Logické rozdelenie CD - na sektory

Ako nuly sa interpretuje pravidelné striedanie priehlbiniek s rovinkami a akákoľvek nepravidelnosť je interpretovaná ako jednotka. Táto informácia sa potom spracúva v ďalších elektronických obvodoch a jej interpretácia sa líši podľa funkcie, ktorú má dané médium vykonávať.

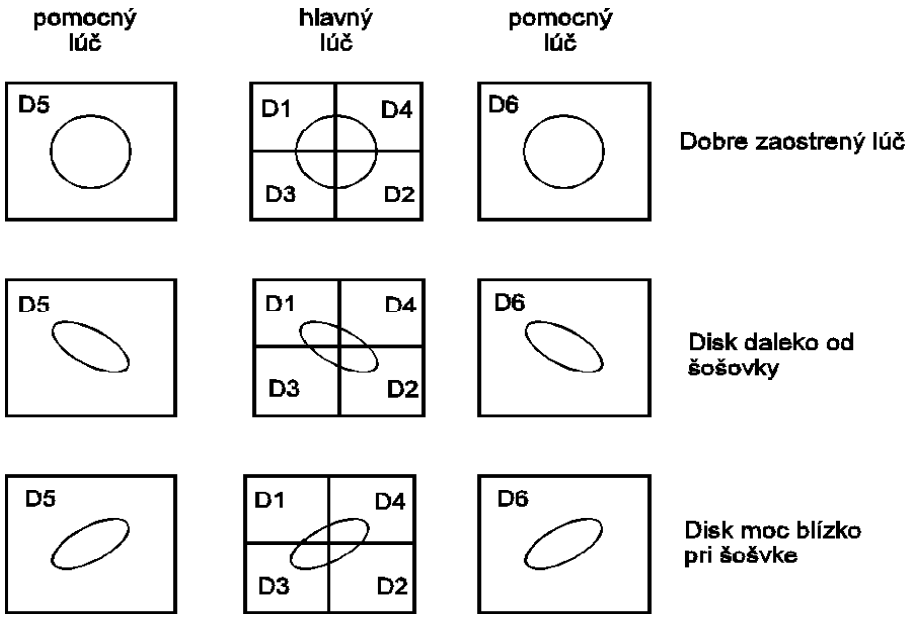
6.4.3 Optická sústava

V tomto odseku podrobnejšie opíšeme optické čítanie a problémy s ním súvisiace (zaostrovanie a držanie stopy).

Optická sústava je pripevnená na pohyblivom ramene, ktoré môže byť otočné alebo posuvné. Pri otočnom je optická sústava umiestená na konci otočného ramena. Servo motor otáča ramenom a tým mení polohu celej optickej sústavy voči disku. Toto rameno bolo postupne nahradené posuvným. Na ňom sa optická sústava pohybuje medzi stredom a okrajom disku.

V optickej sústave sa náchadza polovodičová dióda, vyrobená na báze Hliník-Gálium-Arzenidu. Vyžaruje neviditeľné infračervené svetlo. Sústava šošoviek dokáže zamerať lúč na bod v veľkosti $1\mu m$. To umožnuje vytvárať stopy vzdialených od seba $2\mu m$, pričom priehlbinka - pit je široká od $0,4\mu m$ po $0.5\mu m$ a hlboká $0.1\mu m$.

Čítacia hlava sa skladá z laseru, sústavy šošoviek a zrkadiel, fotodiód a mechanických častí (slúžiacich na pohyb hlavy a zaostrovanie). Správne zaostrenie sa kontroluje pomocou štyroch fotodiód (viď nasledovný obrázok), na ktoré dopadá odrazený laserový lúč. Ak je dopadajúci lúč správne zaostrený, má tvar kruhu. Vtedy majú všetky štyri diódy signál. Inak má tvar elipsy a dve z diód strácajú signál. Podľa toho, ktoré, sa pohne sústavou šošoviek k povrchu alebo od neho, čím sa zaostrí.

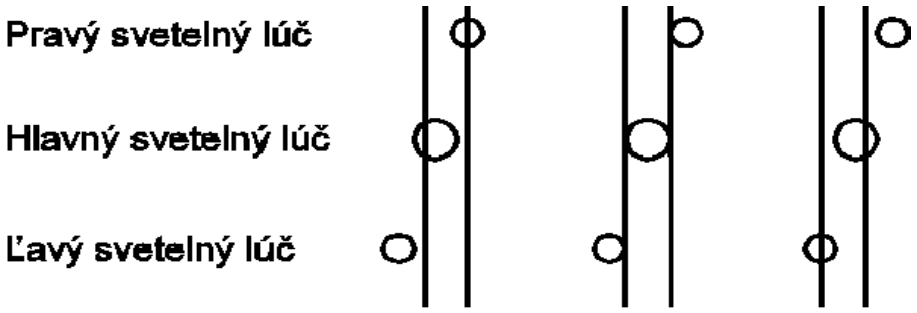


Obrázok 6.18: Testovanie správneho zaostrenia

Kontrolovať treba aj to, či lúč pri čítaní nevybočil zo stopy. Na to slúžia dva pomocné bočné lúče, ktorých odraz opäť testujeme dvoma fotodiódami. Ak lúč vybočí zo stopy, stratí sa signál z ľavého alebo pravého pomocného lúča, podľa čoho vieme, ktorým smerom treba hlavu posunúť.

6.4.4 Typy médií

Na CD disky možno ukladať informácie rôznych typov (napr. zvuk, obraz, video, počítačové dáta). Tieto typy majú odlišné nároky na kapacitu/rýchlosť/presnosť. Napríklad, pri snímaní záznamu videa môžu nastať chyby, pretože neveľa chybne zobrazených bodov si na výslednom obraze ani nevšimneme (obraz má skoro milión bodov) a teda na celé CD môžeme tolerovať tisícky chýb. Na druhej strane, vyžadujeme extrémnu rýchlosť



Obrázok 6.19: Testovanie udržania stopy

čítania (desiatky Mb za sekundu). Presne opačnou je situácia pri zázname počítačových dát, tu nesmie nastať ani jedna chyba čítania a čítanie dát nemusí byť až také rýchle. Preto vzniklo viacero typov CD médií:

CD-DA	compact disk digital audio
CD-ROM	compact disk read only memory
CDV	compact disc video
CD-I	compact disk interactive
PhotoCD	na ukladanie digitálnych fotografií

Okrem spomínaných CD diskov, ktoré predstavovali pamäte typu ROM, vznikli aj médiá:

- *CD-R* - umožňujúci jedenkrát informáciu zapísať a ľubovoľne veľa krát ju čítať
- *CD-RW* - umožňujúci ľubovolne veľa zápisov i čítaní. V závere tejto časti spomenieme aj novší typ optického média – DVD.

Vráťme sa ale k spomenutým typom CD. Treba spomenúť, že viaceré z nich majú aj svoje 'podtypy' (napr. niekoľko možných formátov CD-ROM). Jednotlivé formáty uvedených médií sú definované v takzvaných *color book* (alebo farebných knihách):

- *Red Book* - fyzický formát audio CD (známy aj ako CD-DA)
- *Yellow Book*- fyzický formát dátových CD (CD-ROM)
- *Green Book*- fyzický formát CD-i
- *Orange Book* - fyzický formát zapisovateľných CD. Má tri časti: CD-MO (Magneto-Optical, magneto-optické), CD-WO (Write-Once, zapisovateľné; vrátane PhotoCD), CD-RW (ReWritable, prepisovateľné)
- *White Book*- formát VideoCD
- *Blue Book* - CD Extra (jedno CD obsahuje dve sekcie, prvá je CD-DA, druhá dátová; známe aj ako CD Plus)

6.4.5 CD Digital Audio

Ako jedno z prvých diskových médií sa objavuje CD-DA, na ktorý možno ukladať zvukový (audio) záznam.

Bežný audio disk má okolo 12,5 cm v priemere a je možné na ňom zaznamenať od 60 do 70 minút stereofónneho digitálneho zvukového záznamu. Vyšší rozsah (okolo 75 minút) záznamu už maximálne využíva toleranciu hustoty záznamu a je 'zaplatený' vyššou chybovosťou.

Používa sa vzorkovacia frekvencia 44 kHz.

Záznam sa začína pri strede kotúča a končí pri vonkajšom okraji. Dáta sú uložené v špirále. Diskety a pevné disky ukladali dáta do sústredných kružníc, pričom každá kružnica obsahovala rovnaké množstvo dát. Preto okrajové kružnice neboli naplno využité. Ukladanie do špirály je oveľa ekonomickejšie, sústredné kruhy obsahujú nerovnaké množstvo dát (a okrajové najviac), čím sa viac využije kapacita média.

Informácie na CD-DA sú rozdelené na úseky - *sektory* (tiež *large frame– veľké rámce*). Tieto sa delia na 98 (*malých*) *rámcov*. Tie už predstavujú najmenšiu jednotku údajov. Obsahujú 24 bajtov dát a 8 bajtov opravného kódu (viď obr. 6.21). Ďalej, vzorky (rámce) nie sú ukladané za sebou (t.j. podľa poradia), ale *prekladane* (napr. v poradí $1, 11, 21, \ldots, 2, 12, 22, \ldots$), čo má tú výhodu, že ak sa poškodí (napr. škrabancom) neveľká súvislá časť disku, chyba zasiahne zvukové vzorky rôznych periód. Z jednej periódy sa tak 'nestratí' priveľa údajov a pretože zvukový záznam sa nemení príliš prudko, je možné zo susedných údajov[1] interpolovať stratený údaj (napr. ako priemer susedných údajov).

6.4.6 CD-ROM

Nový typ CD, označený ako CD-ROM, uspôsobený pre záznam počítačových dát bol predstavený v roku 1984. Disk má rovnaké rozmery ako 'klasický' CD-disk používaný na záznam zvuku. Zmestí sa naň 650 MB dát. Z dôvodov kompatibility s CD-DA (aby čítacie CD-ROM mechaniky ľahko dokázali prehrávať aj CD-DA disky) sú dáta tiež zapísané v špirále.

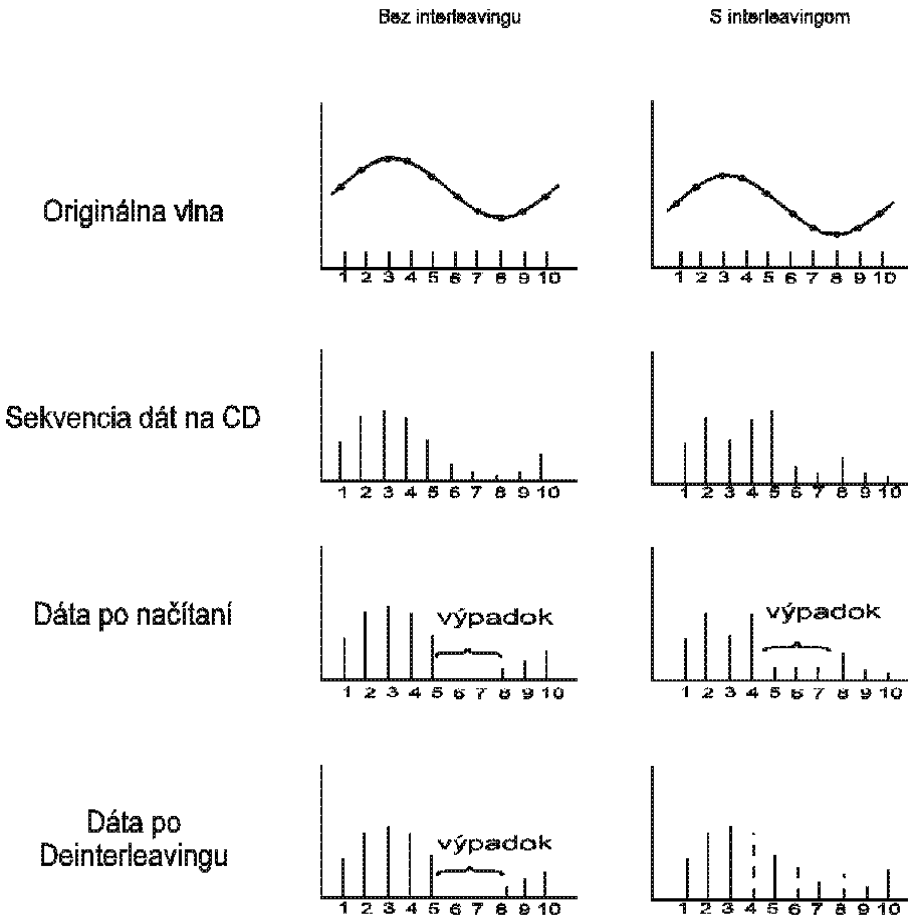
Technika zápisu a čítania CD-ROM je podobná CD-DA. Využíva sa tu tiež EFM kódovanie, avšak dáta sú uložené v sektoroch dĺžky 2325 bytov. Sektor obsahuje hlavičku, 2048 bytov dát a 288 bytov informácie pre viacúrovňový opravný kód (*layered ECC*).Podobne ako pri CD-DA sa sektory delia na rámce, ktorých je 98 na sektor.

Opravný kód má nasledovnú štruktúru informácie: 4 byty sú paritné a pomocou nich sa testuje, či vôbec došlo k chybe. Zvyšných 276 už slúži na opravu poškodenej informácie. Oprava je dvojúrovňová: prvá na úrovni bytov, druhá na úrovni rámcov.

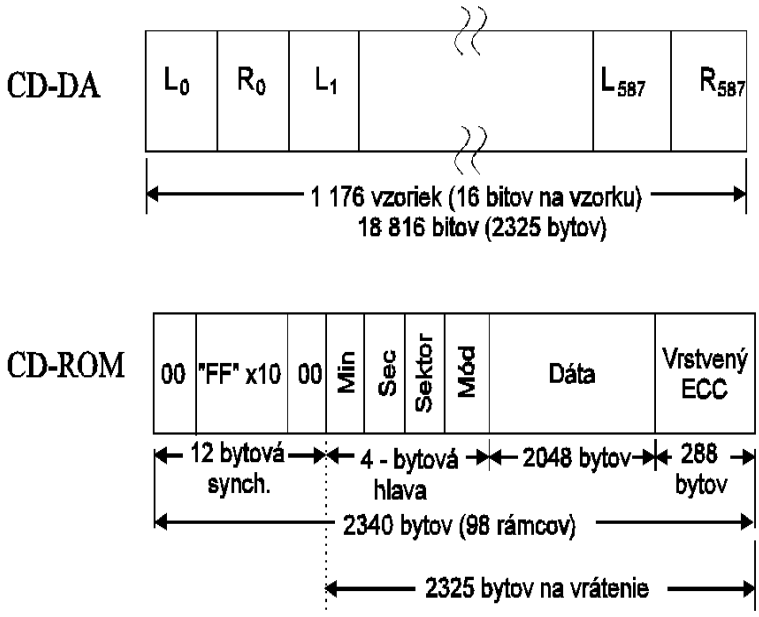
Súborový systém

Štruktúra systému súborov a adresárov je definovaná normou ISO 9660. Má tri časti. Prvá definuje súborový systém kompatibilný s MS-DOS-om (súbory majú 8-znakové mená a 3-znakovú príponu, dovolených je osem vnorení podadresárov). Druhá časť povoľuje dlhé názvy súborov a 32 vnorení podadresárov. S týmto formátom už MS-DOS nedokáže pracovať. Ani s tretím, ktorý povoľuje aj nesúvislé súbory. to... Vznikli aj iné

[1] susedných k poškodenému údajmu



Obrázok 6.20: Obnova poškodenej informácie interpoláciou



Obrázok 6.21: Sektory CD-ROM a CD-DA

súborové systémy (napr. Unix, Windows 95), pri ktorých je problém s prenositeľnosťou medzi jednotlivými systémami.

Rýchlosť mechaník

Rýchlosť čítania (*prenosová rýchlosť*) CD-ROM mechaník sa označuje násobkom prenosovej rýchlosti štandardného CD-DA prehrávača, ktorá je okolo 150 kB/s. Napríklad, *16-rýchlostná mechanika* číta rýchlosťou 2400 kB/s.

Pôvodná rýchlosť prenosu čoskoro prestala stačiť. Vznikli multimediálne aplikácie a na CD-ROM sa začali ukladať multimediálne dáta, ako napr. zvuk, obraz a video. Potrebný bol značne rýchly prenos (digitálne video - megabajty za sekundu). Za počiatočnou 1-rýchlostnou mechanikou čoskoro nasledovali $2, 4, 8, 16, 24$ i viac rýchlostné. Na trhu sa ďalej objavujú čoraz rýchlejšie mechaniky.

CD-ROM mechanika môže dáta čítať v dvoch módoch: *konštantnou uhlovou* a *konštantnou lineárnou* rýchlosťou. Pri konštantnej uhlovej rýchlosti (Constant Angular Velocity – CAV) sa médium otáča rovnakou (uhlovou) rýchlosťou, preto má mechanika vzrastajúcu rýchlosť čítania dát smerom k okraju (napr. od 1200 kB/s na vnútornej strane disku až do 2400 kB/s na vonkajšej strane). Druhý mód čítania je čítať stálou lineárnou rýchlosťou (CLV - Constant Linear Velocity), pri ktorom sa plynule mení rýchlosť otáčok podľa vzdialenosti od stredu disku tak, aby mechanika zakaždým mala rovnakú rýchlosť čítania dát[2]. Staršie mechaniky pracovali v móde CLV, novšie používajú obe techniky. Používaním oboch módov čítania napríklad možno opravovať chyby– pri čítaní poškodeného miesta mechanika spomalí a pokúsi sa prečítať dáta ešte raz aby mohla ľahšie opraviť chybu.

Na trhu sa objavili aj mechaniky s označením typu *24max*. Aký je význam tohto označenia? Uviedli sme, že existujú dva spôsoby čítania (konštantná uhlová a konštantná lineárna rýchlosť), pričom novšie mechaniky zvládajú obe. Pri CAV môže byť rýchlosť prenosu pri strede disku napr. 2400 kb/s a pri okraji 4800 kb/s. Táto mechanika môže byť označená ako 40max, hoci v skutočnosti je 20-rýchlostná. Takisto, reálna rýchlosť čítania dát nezávisí len od maximálnej rýchlosti čítania dát, ale aj na množstve ďalších faktorov – napríklad ako sa mechanika správa pri čítaní poškodených miest[3]. Preto mnohé mechaniky sú v praxi rýchlejšie ako iné s dvojnásobným koeficientom.

Ochrana proti kopírovaniu CD-ROM médií

Pretože všetky dáta na CD musí čítacia mechanika vedieť prečítať, neexistuje všeobecná a účinná metóda ako zabrániť duplikácii média. Existujú však spôsoby, ako kopírovanie 'sťažiť'.

Jednoduchou a častou technikou je zmeniť informácie o súboroch a predstierať, že niektoré z nich majú dĺžku stoviek Mb. Informáciu o dĺžke zmeníme v hlavičke súboru. Náš softvér s týmito súbormi pracuje správne, pretože ich skutočnú dĺžku pozná. No ak sa pokúsime tieto súbory kopírovať, skopírujeme aj nezmyselné dáta. Táto ochrana je však neúčinná, ak urobíme po sektoroch kópiu celého CD.

[2] t.j. čítala rovnaký objem dát (za jednotku času)

[3] hneď vyhlási chybu, alebo sa pokúsi sektor čítať znovu? Raz, alebo viackrát?

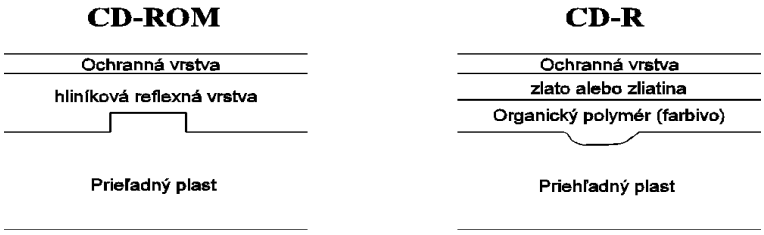
'Špeciálnejšie metódy' sú napríklad: Vylisovať CD, ktoré obsahuje dáta aj za hranicou zapisovateľnosti bežných CD-R médií. Také sa už duplikuje ťažšie, potrebné sú špeciálne CD-R médiá a špeciálny software. Inou možnosťou je zapísať do sektora chybné dáta (napr. keď vytvoríme rámec s nesprávnou paritou). Program číta v režime 'bez opravy chýb' (sám si zaisťuje detekciu a opravu chýb pri čítaní 'našich' sektorov). Pokiaľ čítame štandardne, dáta sa opravia automaticky (samoopravným kódom), čím sa poškodia a na kópii sa už nenachádza taká informácia, akú mal aj originál. Nevýhodou tohto spôsobu je, že mechanika musí vedieť čítať z CD v režime 'bez opravy chýb'. Pretože to nedokáže každá CD-ROM mechanika, pre ochranu počítačových dát ho nemôžeme použiť. Môžme ho však uplatniť napr. na herných konzolách. Na konzolách sa používajú aj ďalšie metódy používajúce upravené mechaniky alebo médiá. Napríklad používanie médií s natoľko nízkou reflexivitou, že bežné CD-ROM mechaniky ich nedokážu prečítať.

6.4.7 CD-Recordable

Štruktúra média

Štruktúra média CD-R je veľmi podobná štruktúre klasického CD-ROM. Médium obsahuje nasledovné vrstvy: potlač, špeciálna nepoškriabateľná ochranná vrstva (nemajú všetky médiá), ochranná vrstva, odrazová vrstva (hrubá 50 až 100 nm)– organické farbivo (polymér) a naspodu priehľadný plast.

Podstatným rozdielom medzi CD-ROM a CD-R je zloženie odrazovej vrstvy. Plochy vypálené pri zápise do vrstvy farbiva pohlcujú svetlo, rovnako ako pity lisovaného CD. Ako organické farbivo sa spravidla používa cyanín (zelená, modrá farba) alebo ftalocyanín (zlatá farba). Oba typy majú obmedzenú životnosť (t.j. dobu uchovania informácie) a navyše, svetlo sa od ich povrchu odráža s menšou intenzitou ako u lisovaných CD-ROM. Preto potrebujeme citlivejšiu čítaciu mechaniku.



Obrázok 6.22: Porovnanie štruktúra CD-ROM a CD-R

Multisession disky

Nevýhodou pôvodných formátov CD-R diskov bola nemožnosť neskoršieho zápisu dát na nevyužité miesto. Hoci bolo zapisované len na tretinu disku, nebolo možné na disk opätovne zapisovať a využiť tak nevyužitú časť.

Riešením je členenie disku na bloky dát - *sessions*. Session obsahuje jednu alebo viac stôp ľubovolného typu. Nemusí byť napálená v jednom zápise, môžeme ju po častiach vytvárať vo viacerých zápisoch. Čítať ju bežnou mechanikou je možné až keď ju uzavrieme. Vtedy však už nie je možné do nej znova pripisovať nové dáta. Uzatvorením disku sa zakáže vytváranie nových sessions.

Pri čítaní z média sa najskôr nájde posledná zatvorená session a prečíta sa jej adresár. Tento môže obsahovať aj odkazy na súbory v predchádzajúcich sessions– je možné zlučovať súbory viacerých session, simulovať vymazanie súborov starej verzie či ich prepísanie novými súbormi (namiesto odkazu zapíšeme nový súbor).

Prehrávače CD-audio prezerajú len prvú session, čo umožňuje vytvárať disky CD-Extra. V audio-prehrávači sa médium bude správať ako obyčajné CD-DA a až po vložení do CD-ROM mechaniky sa objavia aj dátové session.

Technika mulitsessions bola po prvý krát použitá pre PhotoCD disky. Dnes sa využíva aj pre CD-R disky.

Záznam po stopách

Alternatívnou možnosťou k zapisovaniu súborov na CD je priamo určiť obsahy jednotlivých stôp CD. Záznam po stopách umožňuje k už zapísaným stopám pripisovať ďalšie stopy.

6.4.8 CD ReWritable

V roku 1988 Tandy Corporation vyvinula prepisovateľný CD disk. Pre vysoké výrobné náklady sa nikdy neobjavil na trhu. Až v roku 1995 predstavila firma Philips technológiu CD Erasable. Koncom roku 1996 boli na trh uvedené CD-E disky, známejšie pod označením *CD-ReWritable*.

Odrazová plocha sa skladá zo zliatiny striebra, irídia, antimónu a telúria. Ak je zliatina v kryštalickom skupenstve, dobre odráža svetlo a naopak v amorfnom svetlo rozptyľuje a pohlcuje. Laser sa používa na zmenu materiálu z amorfného na kryštalický a naopak. Pri zápise vysokovýkonný laser zahreje kryštalickú zliatinu až na taviacu teplotu 600 ^{o}C. Keď materiál vychladne, zmení svoju štruktúru na amorfnú. Naopak, zahriatím na 200 ^{o}C zliatina kryštalizuje. Pri čítaní sa takisto používa laser, no slabší, ktorý ešte naviac pulzuje, aby nedochádzalo k zahrievaniu zliatiny.

Technológia CD-E je kompatibilná s predchádzajúcimi štandardami, no tieto disky nie sú čitateľné na bežných mechanikách, pretože majú v porovnaní s bežnými CD príliš malú odrazivosť.

Jeden disk môže byť prepisovaný 1000 až 10 000 krát.

6.4.9 DVD disky

Na jedno CD sa zmestí len hodina záznamu videa. Mnoho firiem sa pokúšalo vylepšiť technológiu CD a nájsť spôsob, ako uložiť viac digitálnych dát. V roku 1995 sa objavuje štandart novej technológie DVD, ktorá je ďalším stupňom vývoja optickej technológie. Médium má rovnaké rozmery i vonkajší výzor ako 'klasické' CD-čko. No vieme naň zapísať viac dát, mechaniky majú vyššiu prenosovú rýchlosť a dokážu čítať dáta z CD-ROM a CD-DA.

DVD je skratka anglických slov *Digital Verstile Disc* (*digitálny univerzálny disk*). V septembri 1995 sa dohodli dve skupiny firiem, obe presadzujúce vlastné štandardy, na základných črtách nového vysokokapacitného média, pôvodne určeného pre záznam videa. Dali mu názov DVD - Digital Video Disc. Neskôr, keď sa začalo uvažovať o jeho využití

pri ukladaní počítačových dát, význam skratky sa zmenil na Digital Versatile Disc, čiže digitálny univerzálny disk.

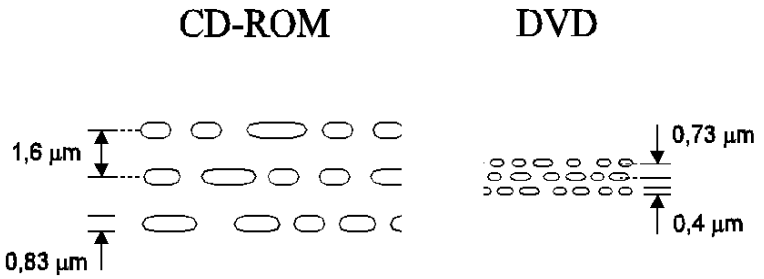
Každopádne, DVD bolo predstavované ako univerzálne médium. Jeden disk môže súčasne obsahovať viacero typov dát (zvuk, fotografie, video, digitálne dáta) pre viacero zariadení (audio-zariadenie, video, počítač či hraciu konzolu). Odpadajú tak ťažkosti s prenosom dát medzi platformami, bežné u tradičných CD, kde boli definované nekompatibilné štandardy pre každý typ zariadenia a prípadne sa definovali hybridy obsahujúce viacero typov dát (ako napr. CD-Extra). Túto univerzálnosť DVD dosahuje jednak tým, že DVD špecifikácia zavádza všeobecný formát údajov a tiež tým, že je umožnený náhodný prístup.

Princípy DVD a rozdiely medzi CD-ROM a DVD

	CD-ROM	**DVD**
Hrúbka disku	1.2mm	0.6 mm (jednostranný) 1.2 mm (obojstranný)
Veľkosť pitu	0.83 μm	0.4 μm
Rozostup stopy	1.6 μm	0.74 μm
Základná prenosová rýchlosť	150 KB/s	11 MB/s

Tabuľka 6.1: Rozdiely medzi CD-ROM a DVD

CD-ROM používa infračervený laser. DVD používa červený laser v oblasti viditeľného spektra, ktorý má menšiu vlnovú dĺžku– čím je možné ho lepšie zaostriť a tým aj zvýšiť hustotu dát na disku (zmenšia sa rozmery pitov a rozostup stopy). Ďalšou prednosťou DVD technológie je vysoká prenosová rýchlosť.



Obrázok 6.23: Porovnanie hustoty záznamu CD-ROM a DVD

Záznamová vrstva je polopriehľadná. To môžeme využiť a na seba môžeme uložiť dve záznamové vrstvy. Nastavením vhodnej vlnovej dĺžky vieme čítací laser zaostriť na jednu či druhú vrstvu. Laser sa potom odráža z tejto vrstvy, z nej číta informácie.

Ďalšou možnosťou zvýšenia kapacity je zapisovať na obe strany. Pri dvoch vrstvách na každej strane tak dostaneme DVD so štyrmi vrstvami. Celkovo je možné na jeden DVD disk uložiť objem dát zodpovedajúci 7 až 25 štandardným CD-ROM-om. Kapacity rôznych médií sú znázornené v tabuľke.

Priemer	*SL/SS*	*DL/SS*	*SL/DS*	*DL/DS*
12 cm	**4.7 GB**	**8.5 GB**	**9.4 GB**	**17 GB**
8 cm	**1.4 GB**	**2.6 GB**	**2.9 GB**	**5.3 GB**

SL/SS znamená *Single layer – Single sided*, čiže jednovrstvový jednostranný záznam, DL/SS je *Dual layer – Single sided*, čiže dvojvrstvový jednostranný záznam, SL/DS je *Single layer – Double sided*, t.j. jednovrstvový dvojstranný záznam a DL/DS je *Dual layer – Double sided*, teda dvojvrstvový dvojstranný záznam.

Tabuľka 6.2: Kapacity rôznych typov DVD diskov

Pre ilustráciu uveďme, že na jeden disk DVD o kapacite 4,3 Gb sa dá nahrať (bez kompresie) dvojhodinový film a na dvojvrstvovom obojstrannom médiu skoro 9 hodín videa. Čitateľ nech si skúsi vyrátať objem informácií iných typov, ktoré je možné na DVD disky uložiť– napr. koľko hodín hudby, koľko fotografií alebo počet kníh. Najnovšie správy pritom hovoria o možnosti zápisu s ešte vyššou hustotou. Má byť zachovaná kompatibilita s klasickými DVD diskami. Máme sa teda na čo tešiť...

Kapitola 7

Vyvíjané technológie pamätí

V tejto kapitole spomenieme ďalšie pamäťové technológie, ktoré sú v štádiu výskumu. Aj keď sa v praxi nepoužívajú, lebo ich súčasné metódy realizácie sú príliš nákladné, v budúcnosti sa môžu nájsť úspornejšie riešenia. Spomenieme dve technológie, holografickú a kryogénnu. Prvá z nich vyniká najmä vysokou hustotou záznamu; druhá rýchlosťou čítania i zápisu. A preto aj keď nie sú bežne používané, pri niektorých špeciálnych úlohách nachádzajú uplatnenie.

Holografické pamäte

Holografická technológia umožňuje trojrozmerné zobrazenie predmetov, možno ju však použiť aj na zapamätanie číslicových údajov. V tomto prípade sú jednotlivé bity uložené po celej ploche hologramu zo svetlocitlivej vrstvy.

Holografické pamäte sú veľmi odolné voči poruchám, pretože pri poruche na jednom mieste sa nezničí celý údaj, iba sa zmenší kontrast medzi jednotlivými vzorkami bitov. Umožňujú dosiahnuť vysokú hustotu ukladania údajov, dosahujúcu miliardu bitov na štvorcový centimeter.

Kryogénne pamäte

Alebo tiež *pamäte využívajúce hlboké podchladenie*, sú pamäte využívajúce efekt supravodivosti.

Štandardný preklápací obvod rýchleho počítača je schopný vykonať miliardu preklopení za sekundu, rýchlosti pamätí sa teda pohybujú v nanosekundách. Vo vývoji sú obvody schopné vykonať až sto miliard preklopení za sekundu. Výskumníci ale narážajú na fyzikálne hranice a obmedzenia. Elektrický signál môže za 1 nanosekundu prekonať vzdialenosť niekoľkých centimetrov. Ak by sme aj chceli využiť preklápacie časy kratšie ako 1 nanosekunda, tak by všetky obvody základnej jednotky museli byť od seba vzdialené len niekoľko centimetrov. Znamená to ďalšiu miniaturizáciu; pri činnosti obvodov však vzniká teplo, ktoré by pri zvýšenej hustote prvkov nemohlo byť spoľahlivo odvádzané.

Preto bolo nutné vyvinúť polovodičové obvody, ktoré by vyžadovali málo energie na svoju činnosť (čím by vyvíjali aj menej tepla). Takéto obvody sú už vyvinuté (známe pod menom *Josephsonove obvody*). Pri konštrukcii týchto obvodov sa využívajú supravodivé kovové vrstvy– sú teda konštruované z kovu, ktorý, ak je ochladený na teplotu blízku

absolútnej nule (-273 ^{o}C), tak nekladie prechodu elektrického prúdu žiadny odpor. Preto možno pracovať v týchto obvodoch s mimoriadne malou intenzitou elektrického prúdu.

Aby bolo možné vyvinúť potrebnú teplotu, obvody sa ukladajú do tekutého hélia. V súčasnosti sa vyvíjajú obvody, ktoré využívajú keramické materiály a u ktorých sa aj pri kladných teplotách prejavuje efekt supravodivosti.

Pamäte, ktoré sú zostavené z týchto alebo podobných obvodov, čiže pamäte s hlbokým podchladením sa výhodne využívajú pri riešení úloh pri ktorých sa vyžaduje rýchly prístup k údajom.

Kapitola 8

Rôzne pamäťové štruktúry

Doteraz sme sa venovali najmä technologickým princípom uchovania informácie. Uvažovali sme pritom len jeden spôsob práce s pamäťou, jeden model pamäte: Pamäť obsahuje bunky, každá bunka obsahuje informáciu. S bunkami vieme pracovať (čítať informáciu uloženú v bunke, alebo informáciu do bunky zapisovať), pričom v jednom kroku pracujeme len s jednou bunkou– pamäti zadáme adresu príslušnej bunky a povel pre čítanie alebo zápis (v prípade zápisu zadáme aj dáta, ktoré sa majú zapísať). Tento model platí nielen pre pamäte RAM, ale aj pre SAM– s tým rozdielom, že v prípade SAM je bunkou pamäte blok dát.

Uplatnenie však nachádzajú aj iné pamäťové štruktúry, s ktorými pracujeme odlišným spôsobom. S najznámejšími sa oboznámime v tejto kapitole. Najskôr spomenieme CACHE pamäte, potom asociatívne pamäte (ktorých jeden spôsob využitia je práve pri realizovaní CACHE pamätí), zmienime sa o modulárnych pamätiach a na záver popíšeme zásobník a frontu.

Treba však upozorniť, že štruktúry ktoré uvedieme nebudú nutne radikálne odlišné od 'klasickej pamäti' (ako napr. asociatívne pamäte). Zameriame sa na technickú realizáciu pamäťových štruktúr spĺňajúcich požiadavky z praxe, napr. rýchly prístup k často používaným dátam (pamäte CACHE), vyhľadávanie informácie podľa kľúča (asociatívne pamäte) a ďalšie. Niektoré zo štruktúr ktoré spomenieme sa často realizujú jednoduchou modifikáciou RAM-pamätí, prípadne sa dajú realizovať aj softvérovo (zásobník, fronta). Cieľom tejto kapitoly však ani nie je podať vyčerpávajúci prehľad komplikovaných, čo 'najexotickejších' pamätí, ale skôr ukázať, že pre niektoré významné úlohy je možné zostrojiť špeciálnu pamäťovú štruktúru a zároveň podať prehľad najvýznamnejších takých štruktúr.

8.1 CACHE

Výkonnosť počítača neovplyvňuje len rýchlosť procesora, ale aj rýchlosť pamäte. Pamäť je značne pomalšia než mikroprocesor a možnosti ďalšieho zvyšovania jej rýchlosti sú obmedzené. Vyšší výkon však môžeme dosiahnuť aj *optimalizáciou práce*.

Všimnime si niektoré štatistické údaje o programoch:

Dáta programu sú zväčša usporiadané tak, že program pracuje s malou lokálnou oblasťou dát. Príkazy programu sa zase zväčša vykonávajú za sebou, alebo sa dokonca

(v cykle) viackrát opakuje vykonávanie skupiny (za sebou idúcich) inštrukcií (tela cyklu). Tieto vlastnosti vyjadrujú princípy *časovej* a *miestnej lokality*:

- *Časová lokalita* vyjadruje skutočnosť, že adresa, ktorá bola práve vyvolaná (tj. pracovalo sa s pamäťovou bunkou určenou danou adresou) bude v krátkej dobe vyvolaná znova.
- *Miestna lokalita* vyjadruje skutočnosť, že okrem údaju z aktuálne čítanej adresy sa bude v krátkej dobe vyžadovať údaj aj z jej okolia.

CACHE je názov rýchlej nízkokapacitnej pamäte. Jej kapacita je oveľa nižšia ako kapacita operačnej pamäte, no na druhej strane má oveľa vyššiu rýchlosť ako hlavná pamäť (asi 5 až 10 krát). Väčšinou je súčasťou procesora.

Slúži na ukladanie najpotrebnejších a najčastejšie používaných údajov. Do CACHE sa prenesie blok často používaných údajov a pokiaľ si tieto údaje program znova vyžaduje, procesor ich nemusí vyvolávať z operačnej pamäte, ale priamo z rýchlej pamäte cache. CACHE teda slúži ako vyrovnávacia pamäť (buffer) operačnej pamäte.

Ako sa dosahuje rýchlosť CACHE ? Existuje viacero metód jej konštrukcie:

1. cache = nízkokapacitná RAM

   (doba prístupu je funkciou počtu slov v pamäti)

2. asociatívna pamäť

   (doba prístupu je funkcia dĺžky slova)

3. bipolárne pamäte (namiesto 'tradičných' MOS)

4. kombinácie predošlých spôsobov

Dosť podstatné je určenie 'veľmi často' používaných dát. Využijeme už uvedené štatistické vlastnosti časovej a miestnej lokality.

Ak procesor potrebuje pracovať s určitým slovom pamäte, najskôr ho hľadá v cache. Ak ho nenájde v cache, vezme ho z pamäte a uloží do cache. Spolu s ním však zoberie a uloží aj jeho lokálne okolie. Rovnaký princíp platí pre akýkoľvek prístup k pamäti, či už za účelom čítania dát programu alebo inštrukcií programu. Vďaka princípom 'časovej a miestnej lokality' sa dosahuje 85 - 95 % úspešnosť pri hľadaní údajov v cache.

Pamäť cache má nízku kapacitu, rýchlo sa zaplní. Preto treba vedieť vyradiť z cache najmenej používané údaje. Spôsob detekcie a 'vyraďovania' závisí od konkrétneho návrhu systému, najčastejšie prístupy sú:

- LFU (*Least-Frequently Used*) - vylúči sa bunka, ktorá sa používala najmenší počet krát (Realizácia: spolu so slovom uchovávame aj informáciu o tom, koľkokrát sa s ním pracovalo).
- LRU (*Least-Recently Used*) - vylúči sa bunka, ktorá sa nepoužívala najdlhší čas.

Ak sa hľadané údaje našli v cache, ich čítanie je rovnaké ako u bežnej pamäte. Zápis do CACHE možno realizovať dvoma spôsobmi:

1. *write though method*– pri každom zápise do cache sa uskutoční aj zápis do operačnej pamäte

2. *write back*– ak už nie je potrebné určitú pamäťovú bunku uchovávať v cache, tak predtým ako sa vymaže z cache – ak bola jej kópia v cache modifikovaná – sa obsah kópie zapíše do hlavnej pamäte. Každá bunka v cache má flag určujúci, či sa do bunky v cache zapisovalo, alebo nie.

8.1.1 Organizácia CACHE

Keď do CACHE ukladáme dáta z hlavnej pamäte, musíme tiež neskôr vedieť určiť ich pôvodné miesto v pamäti (tj. určiť ich adresu). Na to existujú 3 metódy:

- direct mapping
- associative mapping
- set-associative mapping

direct mapping

Adresa slova v hlavnej pamäti sa delí na dve časti: na *index* (nižšie bity) a *tag* (vyššie bity). Do CACHE sa uloží dátová časť slova + tag. V cache sú teda slová s dĺžkou (dĺžka dát + dĺžka tag-u).

Aby sme zrekonštruovali pôvodnú adresu, potrebujeme ešte určiť index. Ten sa rovná indexu daného slova v CACHE.

Príklad V.1: Majme v cache slovo (s indexom 0001), ktoré obsahuje dátovú zložku s hodnotou X a tag 110111. Potom táto položka predstavuje pamäťovú bunku s adresou 1101110001.

Nevýhoda uvedenej metódy je značná : do CACHE nemožno uložiť dve slová, ktorých adresa má rozdielne tag-y a zhodné indexy.

associative shaping

Do cache sa ukladá adresa + slovo.

Výhodou tejto metódy je možnosť ukladať slová s ľubovolnými adresami (na rozdiel od predošlej), nevýhodou je potreba väčšej cache.

set-associative shapping

Je kombináciou predošlých metód.

Na každej adrese v cache je uložených niekoľko slov. Každé slovo má svoj tag + dátovú časť. Pretože je viacero slov na jednej adrese v cache, môže mať viacero slov rovnaký index, a rozličné tag-y. Tým sa čiastočne odstráni nevýhoda prvej metódy.

Keď chceme nájsť v cache slovo s určitou adresou, najprv pomocou indexu nájdeme príslušnú skupinu slov. Potom porovnávame tag-y týchto slov s tag-om našej adresy a pokiaľ nastane zhoda, hľadané slovo sme našli.

Uviedli sme problémy spojené s realizáciou cache, pričom sme naznačili niekoľko spôsobov riešenia.

Pri návrhu cache (a procesora) sa zvolia konkrétne prístupy a riešenia. Podstatné je určenie veľkosti lokálnej oblasti (okolia) a určenie veľkosti cache.

8.2 Asociatívna pamäť

V bežných pamätiach (RAM, ROM) boli údaje dostupné pomocou adresy.

V *asociatívnej* pamäti sú údaje prístupné na základe asociácií. Asociatívna pamäť pracuje podobne ako mozog, ktorý pri vyhľadaní jednej informácie nájde informácie s ňou súvisiace na základe rôznych kritérií- *asociácií*. Najčastejším testovacím kritériom (vyhľadávacím kľúčom) je časť obsahu hľadanej bunky. Údaje uložené v pamäti sa porovnávajú so zadanou vzorkou (kľúčom) a indikuje sa, na ktorých adresách došlo k zhode. Testuje sa paralelne, pamäť je veľmi rýchla. Má však vysokú zložitosť a z toho vyplývajúcu vysokú cenu.

Asociatívna pamäť sa využíva pri niektorých špeciálnych aplikáciách, v umelej inteligencii, expertných systémoch a tiež CACHE.

Údaje uchovávané v cache sú zložené z adresovej a dátovej zložky. Ak chceme prečítať hodnotu pamäťovej bunky s určitou adresou, túto adresu dáme ako vyhľadávací kľúč. Pokiaľ sa hľadaná bunka nachádza v cache, výstupom asociatívnej pamäte je práve jedno slovo obsahujúce hodnotu uchovávanú touto bunkou. Zapisovanie i stratégie obhospodarovania CACHE sa robia rovnako ako s bežnými typmi pamätí.

Komunikácia z asociatívnou pamäťou vyzerá nasledovne:

1. Súčasťou m-bitovej asociatívnej pamäte sú registre A a M.

   Do registra A vložíme hľadanú vzorku.

2. Obsah registra A sa porovná so všetkými slovami pamäte.

3. Ak pri porovnávaní s i-tou vzorkou nastala zhoda (pričom zhodu nemusíme definovať ako rovnosť) sa do príslušného bitu M-registra zapíše 1, inak sa zapíše 0.

4. Ďalej sa bude pracovať len s tými pamäťovými miestami, ktorých zodpovedajúci bit v registri M je nastavený na 1.

   Obvod obsahuje m porovnávacích obvodov.

Zvyčajne nie je klúčom celý uchovávaný obsah, ale len niektoré bity slova. To, ktoré sú to, určuje tzv. maska. Bity, ktoré chceme porovnávať sú v maske označené 1, ostatné (na ktorých obsahu nám nezáleží) majú nastavenú 0.

Zápis a čítanie

čítanie– ak register M obsahuje viacero jednotiek (našli sme viacero slov zhodných so zadanou vzorkou), potom treba príslušné slová čítať postupne.

Môžeme napríklad pripojť register M na zariadenie postupne generujúce riadiaci signál read pre slová s jednotkou v registri M.

zápis– zvyčajne sa pri využití cache predpokladá, že špecifikovaný údaj je len jeden (t.j. zhoda s kľúčom nastala len v jednom prípade).

8.3 Modulárna pamäť

Program sa vykonáva tak, že sa postupne čítajú inštrukcie z pamäte a vykonávajú sa. Vykonávanie by sa však dalo urýchliť, ak by sa niekoľko operácií mohlo vykonávať naraz. To znamená vedieť vykonávať naraz aj viacero operácií s pamäťou.

Možností realizácie je viacero. Napríklad, pamäť môže mať viacero vstupov/výstupov, čo je však drahé riešenie. Iným riešením sú *modulárne organizované* pamäte. Pamäť sa rozdelí na viacero nezávislých častí (modulov).

K modulárnym pamätiam sa môže pristupovať dvojako:

1. Vyššie bity určujú pamäťový modul, nižšie- slovo v module.

   (za sebou nasledujúce slová sú v jednom module)

2. Vyššie bity určujú slovo v module, nižšie- pamäťový modul.

   (za sebou idúce slová sú v rozličných moduloch, inštrukcie možno spracovať paralelne)

8.4 Zásobník a fronta

Na dočasné uchovávanie pracovných údajov slúžia dátové štruktúry *zásobník* a *fronta*. Obe sú čitateľovi dozaista známe.

Do zásobníka je možné dáta ukladať i vyberať. Pri uložení sa zásobník predĺži o jednu položku (ukladanú) smerom nahor. Smerom nahor znamená, že naposledy ukladaná položka je na nižšej adrese ako najskôr ukladaná položka. Pri výbere sa vyberie údaj z najvrchnejšej pozície a vyradí sa zo zásobníka (zásobník sa zníži o jednu položku, smerom nadol).

So zásobníkom sa pracuje pomocou dvoch príkazov uloženie (PUSH), výber (POP) a dvoch booleovských funkcií: test, či je zásobník plný (FULL) a test či je prázdny (EMPTY).

Príklad V.2: Do zásobníka sme vložili najskôr údaj (číslo) 3, potom 7 a nakoniec 9. Po príkaze výberu (POP) obdržíme údaj 9. Ďalší príkaz POP vráti číslo 7. Po vložení čísla 14 (príkazom PUSH) dostaneme operáciou POP výsledok 14 a ďalším povelom POP číslo 3.

Zásobník vyberie ako prvú tú položku, ktorá bola uložená ako posledná. Nazýva sa aj LIFO (Last In - First Out).

Zásobník možno realizovať hardwarovo, softwarovo i kombinovane.

Pracuje sa s ním pomocou premenných Stack Pointra, ktorá ukazuje na naposledy uloženú položku, tzv. vrchol zásobníka, a premenných Stack Base a Stack Limit, ktoré udávajú začiatok a koniec pamäte vyhradenej pre zásobník (pri sofvérovej realizácii).

hardwarová realizácia

1. Pomocou k n-bitových registrov s paralelným zápisom a čítaním.

2. Pomocou n posuvných k-bitových registrov, z ktorých každý predstavuje jeden bit všetkých slov uchovaných v zásobníku. Operácie PUSH a POP sa realizujú pomocou posuvov SL a SR.

Obe realizácie vytvárajú zásobník o k slovách s dĺžkou slova n bitov. Príslušné schémy si už čitateľ dokáže navrhnúť.

softwarová realizácia

Ako príklad možného riešenia si uveďme najjednoduchšie riešenie pomocou poľa. Zásobník budeme vytvárať v poli, na ukazovateľ vrchola použijeme premennú; operácie a funkcie so zásobníkom sa už naprogramujú jednoducho.

kombinovaná realizácia

Hardwarová realizácia zásobníka je rýchla, ale drahá a preto má tento zásobník menšiu kapacitu. Softwarová realizácia (pomocou RAM) je síce s väčšou kapacitou, ale je pomalšia.

Kompromisom môže byť riešenie, pri ktorom je horná časť zásobníka v registroch a dolná časť zásobníka v pamäti.

Fronta

Fronta je štruktúra, ktorá ako prvý vyberie ten údaj, ktorý bol do nej prvý vložený. Označuje sa ako FIFO (First In - First Out). Nebudeme sa ňou hlbšie zapodievať, pretože na základe uvedených údajov o pamätiach LIFO (zásobníka) by si už mal čitateľ uvedomiť príslušné analógie s pamäťami FIFO (frontou).

Časť VI

I/O komunikácia

Počítač má význam len v prípade, že je spojený s okolitým svetom, odkiaľ získava vstupné údaje a kam oznamuje výsledky svojej práce.

Kontakt s vonkajším svetom mu zabezpečujú vonkajšie zariadenia. Úlohou vonkajších zariadení (periférií) je získať dáta pre počítač (napr. klávesnica), resp. dáta získané od počítača ďalej spracovať (napr. tlačiareň). Vzletne povedané, sú pre počítač vonkajším svetom, pretože 'čo oni nevidia, nevidí ani on'.

Princípmi týchto zariadení sa budeme zaoberať neskôr; v tejto časti pohovoríme o komunikácií (t.j. výmenou dát) medzi počítačom a perifériami. Nazývať ju budeme Vstupno/Výstupná, resp. Input/Output komunikácia (skrátene len I/O komunikácia). Periférne zariadenie budeme skrátene označovať I/O zariadenie, alebo len I/O.

Pri vstupno/výstupnej komunikácií sa objavuje niekoľko problémov:

- CPU pracuje s binárne kódovanou info, je preto potrebné informácie získané z vonkajšieho sveta (obraz, zvuk, stlačenú klávesu) kódovať binárne
- treba zabezpečiť fyzický prenos dát medzi perifériou a počítačom, niekedy treba vedieť detekovať vznik chyby (napr. pomocou kontroly parity), prípadne chybu aj opraviť (samoopravné kódy)
- informáciu je potrebné preniesť do počítača (na systémovú zbernicu), no zariadenia nemôžu byť na zbernicu pripojené priamo (dôvody uvedieme neskôr)
- CPU a I/O zariadenie obvykle nemožno synchronizovať (majú rozličné rýchlosti) - preto treba koordinovať všetky I/O operácie (inicializovať spojenie, preniesť dáta a ukončiť prenos). Komunikácia prebieha podľa určitým dohodnutým (štandardným) spôsobom, teda podľa určitého protokolu

Vráťme sa ešte k dôvodom, prečo periférne zariadenia nemôžu byť k systémovej zbernici pripojené priamo. Dôvodov je niekoľko:

- procesor využíva zbernicu na komunikáciu s pamäťou a ďalšími blokmi, pričom s nimi komunikuje istým prísne dodržiavaným spôsobom. Periféria priamo pripojená na zbernicu by mohla do tohto procesu elektricky zasahovať a narušiť ho (ak by napríklad počas inštrukčného cyklu fetch klávesnica zapísala na zbernicu kód prečítaného klávesu, procesor by tento kód vnímal ako kód inštrukcie, ktorú má vykonať)
- takisto, konflikt môže nastať medzi dvoma zariadeniami, ktoré sa súčasne pokúšajú zapísať na zbernicu svoje dáta - dôjde k ich zmiešaniu, čo môže viesť k nepredvídaným situáciám
- k rýchlemu procesoru patrí aj rýchla zbernica, ktorá má kratší čas 'vybavovania' požiadaviek - signálov na zbernici. Môže sa preto stať, že riadiace obvody periférie nebudú 'stíhať' priamo komunikovať so zbernicou
- pokiaľ by došlo k poškodeniu periférie, periféria priamo pripojená k zbernici by mohla elektrickým výbojom poškodiť procesor i ostatné bloky pripojené k zbernici
- pre každý typ systémovej zbernice by sa museli vyrábať osobitné druhy klávesníc, tlačiarní a iných periférnych zariadení, alebo by tieto museli mať nadbytočné prispôsobovacie obvody pre rôzne zbernice

Z týchto dôvodov je výhodnejšie dohodnúť niekoľko štandardných pripájaní periférií k počítaču (napr. RS232C, Centronics) a ku každému typu zbernice vyrobiť špecifický obvod (nazývané V/V obvody) majúci úlohu prispôsobovacieho článku medzi zbernicou a perifériou.

Dostávame sa tak k nasledovnej hierarchii I/O systému, opísanej v nasledovnej kapitole:

Kapitola 1

Zloženie I/O systému

Celý vstupno/výstupný systém (alebo Input/Output, skrátene I/O systém) sa skladá z niekoľkých častí:

- *I/O zariadenia* (alebo *periférie*), ktoré 'zberajú' údaje z okolitého sveta. Môžme ich rozdeliť na:
  - vstupné (len získavajú dáta pre počítač, napr. myš)
  - výstupné (len spracúvajú dáta z počítača, napr. tlačiareň)
  - vstupno/výstupné (aj zber dát, aj ich spracovanie, napr. modem)

  Ich princípy si ukážeme v ďalšej časti, v tejto časti pre nás budú periférie len objekty, ktoré chcú komunikovať s počítačom (čítať aj zapisovať).
- *radiče I/O zariadení* (alebo device controllery), prostredníctvom ktorých zariadenie komunikuje s počítačom (presnejšie, s procesorom alebo pamäťou). Komunikácia prebieha dopredu určeným spôsobom (scenáru komunikácie hovoríme protokol)
- *spoje*, po ktorých sa prenášajú dáta (medzi radičmi zariadení a radičmi počítača - vstupnými bránami)
- *obslužný software*

Kapitola 2

Prístup k I/O zariadeniam (I/O accesing)

Opíšme, akým spôsobom môže program komunikovať s periférnym zariadením. Existujú 2 prístupy k I/O portom:

1. memory mapped I/O
2. I/O mapped I/O

V prvom prípade sú I/O porty pripojené k adresovej zbernici. Každé I/O zariadenie má priradené jedno alebo viac čísel, ktoré sa chápu ako adresy pamäťových buniek. Presnejšie, periféria *prekryje* niektoré pamäťové miesta svojími vstupmi, resp. výstupmi. Potom, vstupné I/O sa správa ako pamäť ROM (môžeme z nej len čítať), vstupno/výstupné I/O sa správa ako pamäť RAM. Procesor nemusí mať špeciálne inštrukcie pre prácu s I/O - každá inštrukcia pracujúca s pamäťou môže zároveň pracovať s I/O. Stačí pritom, aby ako adresu pamäťovej bunky udala adresu prislúchajúcu I/O.

Týmto spôsobom sa správa napríklad videopamäť počítačov PC. Z programátorského hľadiska je videopamäť súvislý úsek pamäte začínajúci od adresy A000 (hexadecimálne). Teda videopamäť, ktorá je fyzicky uložená na videokarte, sa správa, akoby bola súčasťou hlavnej pamäte, nachádzajúcej sa na hlavnej doske.

V druhom prípade sú I/O porty nezávislé na pamäti. CPU odlišuje, či sa jedná o operáciu s pamäťou alebo s I/O zariadením. Ak chceme pracovať s I/O zariadením, musíme použiť špeciálne inštrukcie vstupu a výstupu z I/O zariadenia (resp. z I/O portov)- IN a OUT. Pri prenose dát sa po zbernici prenáša aj riadiaci signál rozlišujúci, či sa komunikuje s pamäťou alebo I/O zariadením.

Porovnanie oboch prístupov:

- *memory mapped I/O*:
  - netreba špeciálne operácie I/O vstupu - výstupu
  - pomalšie
  - zmenšuje sa adresový priestor (jeho časť sa využíva pre adresovanie I/O portov)
- *I/O mapped I/O* : má presne opačné vlastnosti v porovnaní s predchádzajúcim prístupom

Kapitola 3

Prenos dát

3.1 Prenos dát na fyzickej úrovni

Fyzicky sa prenos dát medzi perifériami a počítačom najčastejšie uskutočňuje 'tradične', t.j. elektrickým signálom po drôte. Opäť, kódovou abecedou je najčastejšie binárna abeceda - na drôte je v každej chvíli určité napätie, pričom napätie od 0 po X voltov kóduje nulu a napätie od Y do Z voltov kóduje jednotku.

Tento 'tradičný' spôsob prenosu však nie je jediný. Napríklad bezdrôtové myši komunikujú s počítačom pomocou infračerveného svetla; prenos dát sa teda uskutočňuje na 'neelektrickom' princípe, bez použitia 'hmatateľného' média. Takisto, kódová abeceda nemusú byť binárna. Môže sa použiť viacero neprekrývajúcich sa 'hladín' napätia, z ktorých každá kóduje inú informáciu. Napríklad modemy používajú viacej hladín prenosového signálu.

O fyzickom prenose sa však teraz nebudeme podrobnejšie zmieňovať; 'netradičné' spôsoby prenosu sú totiž skôr doménou počítačových sietí. V ďalšom texte budeme predpokladať 'tradičný' model prenosu, resp. budeme hovoriť len o prenose údajov, abstrahujúc od fyzickej realizácie prenosu.

3.2 Módy prenosu dát

Podľa *formátu prenášaných dát* môže byť prenos dát:

- sériový
- paralelný

A podľa *prenosvého módu*:

- synchrónny
- asynchrónny

sériový a paralelný prenos

- Pri sériovom prenose sa dáta prenášajú jednou komunikačnou linkou, čiže správa sa prenáša bit po bite.

- Pri paralelnom prenose máme k dispozícií viac liniek, čiže môžeme naraz prenášať niekoľko bitov.

Paralelný prenos je síce rýchlejší, no vyžaduje viacej komunikačných liniek (napr. najčastejšie sa paralelne prenášajú znaky, čo znamená paralelne prenášať 8 bitov = 8 liniek). Použitie viacerých liniek však znásobuje cenu spojenia, a pre väčšie vzdialenosti je už použitie paralelného spojenia neúnosne drahé. Preto sa používa na krátke vzdialenosti, ak je potrebné rýchlo prenášať veľké množstvá dát (napr. prenos dát medzi počítačom a pevným diskom).

Sériový prenos je pomalší, no lacnejší. Pretože proces prenosu znaku je: odosielateľ postupne vysiela jednotlivé bity a príjemca ich prijíma a skladá do výsledného bytu (resp. znaku), tak sa vyžadujú obvody konvertujúce znak z paralelného tvaru na sériový a naopak. Sériový prenos sa používa na spojenie vzdialených miest, resp. v prípade, že prenosová rýchlosť zariadenia je malá (napr. spojenie počítača a myši - myš prenáša malé množstvo dát, ktoré nie je nutné spracovať 'rýchlo' ((stačí niekoľko krát za sekundu)). Preto stačí tieto dáta prenášať sériovo).

synchrónny a asynchrónny prenos

Prenos dát môže prebiehať dvoma spôsobmi: synchrónne a asynchrónne.

Oba spôsoby majú vlastnú filozofiu riešenia problému 'rozlične rýchlych' periférií.

- Pri synchrónnom prenose si vysielateľ a príjemca 'dohodnú' rovnakú 'rýchlosť práce' - rýchlosť vysielania a prijímania. CPU na adresovú zbernicu pošle adresu zariadenia, na dátovú dáta a nastaví signál WRITE na 1. Zariadenie musí prečítať dáta, kým je WRITE=1. Signál WRITE je generovaný s istou pevne zvolenou frekvenciou a má pevne zvolenú dĺžku.

  Zariadenia majú rozličné rýchlosti. Ako teda určiť dĺžku signálu WRITE? V zásade sú dve možnosti:

  - rôzna dĺžka synchronizačných impulzov WRITE (ktorú si CPU a zariadenie dohodnú na začiatku komunikácie (podľa určitého protokolu)).
  - dĺžka jeho trvania je zvolená tak, aby komunikáciu 'stíhalo' aj najpomalšie zariadenie (z množiny uvažovanovaných periférií).

- Odlišný prístup má asynchrónny prenos. Nenastavuje sa rovnaká rýchlosť vysieľateľa a príjemcu, obaja môžu vysielať rozličnými rýchlosťami. Obaja posielajú po riadiacich linkách množstvo riadiacich signálov (správ). Komunikácia môže vyzerať napríklad takto (čitateľ si môže predstaviť napr. ako CPU posielanie dáta tlačiarni): vysielateľ pošle správu (request), ktorou sa pýta či je zariadenie pripravené. Ak je príjemca pripravený, odpovie (acknowledge). Potom vysielateľ začne posielať dáta. Dáta sa nepošlú naraz, ale po menších častiach (nazývaných rámce). Príjemca potvrdí príjem dát (data received). Vysielateľ oznámi koniec prenosu a preruší spojenie. V prípade CPU a nejakej periférie (napr. tlačiarne) to môže vyzerať napríklad takto: CPU umiestni na dátovú zbernicu údaje, na adresovú adresu zariadenia a nastaví WRITE na 1. Zariadenie dáta prečíta a vyšle riadiaci signál (data recived). CPU potom nastaví WRITE na 0 a zmaže údaje z adresnej a dátovej zbernice. Zariadenie potom nastaví Data recived na 0 a celý cyklus prenosu sa môže opakovať.

Scenár komunikácie špecifikuje protokol.

Vynára sa tu však jeden problém. Prenášajme dáta sériovo, napr. po slovách (slovo nemusí byť len 16 bitov, chápme ho ako určitý 'malý' úsek dát), pričom príjemca a vysieľateľ majú rozličné rýchlosti. K správnemu prečítaniu vysielaného slova však príjemca musí poznať rýchlosť vysielania, inak sa môže stať, že jeden bit započíta viackrát (ak bude čítať rýchlejšie ako bolo vysielané), resp. niekoľko bitov neprečíta (ak bude čítať pomalšie). Preto treba zosynchronizovať vysielateľa a príjemcu aspoň na dobu vysielania slova. K tomu existuje viacero techník, ktoré čitateľ môže nájsť v literatúre (viď zoznam literatúry). Uveďme však jeden - pošle sa niekoľko striedajúcich sa núl a jednotiek. Prechody signálu medzi 0-1 a 1-0 slúžia na nastavenie správnej rýchlosti.

Porovnajme synchrónny a asynchrónny prenos:

1. synchrónny
   - rýchlejší
   - jednoduchšie riadenie (hodinový signál)
   - problémy s rozličnými rýchlosťami periférií
2. asynchrónny
   - pomalší
   - komplikované riadenie (viac riadiacich signálov)
   - flexibilnejší (zariadenia s rozličnými rýchlosťami)

Kapitola 4

Riadenie prenosu dát

I/O operácie môžeme rozdeliť podľa toho, ako sa riadi prenos údajov. Rozoznávame tri základné typy:

1. programom riadené I/O
2. I/O riadené pomocou prerušení
3. DMA (priamy prístup do pamäte)

4.1 I/O riadené programom

Alebo programové I/O. Predstavuje hardwerovo najjednoduchšiu metódu. Nepotrebujeme zložitý I/O hardware, pretože inicializácia, prenos a ukončenie spojenia - resp. implementácia zložitejších I/O operácií či I/O protokolov je softwérová, t.j. popísaná programom. Prenos údajov teda prebieha prostredníctvom CPU, podľa špeciálneho programu.

I/O hardware obsahuje niekoľko registrov, pomocou ktorých sa I/O prenos programuje. Typické registre sú:

- status register
- buffer register
- data counter
- buffer pointer

Status register obsahuje informáciu o aktuálnom stave I/O zariadenia (napr. či sa pracuje v synchrónnom alebo asynchrónnom režime, či je zariadenie pripravené, či sa má zapisovať alebo čítať, a pod.) a informáciu o stave prenášaných dát (napr. typ prenášaných údajov (byte, slovo), alebo informácia o parite prijatej informácie).

Buffer register slúži na dočasné uloženie údajov, ktoré treba preniesť, resp. na dočasné uloženie prijatých údajov (kým sa nespracujú).

Data counter udáva, koľko údajov treba preniesť (udané napríklad v bytoch). Pri prenášaní informácie sa postupne znižuje a ak je rovné 0, prenos sa ukončí.

Buffer pointer uchováva adresu pamäťového miesta, kam sa majú ukladať informácie z buffer registra (resp. odkiaľ sa majú zapisovať do buffer registra).

Prenos potom vyzerá nasledovne (pre ilustráciu, nech CPU dáta zapisuje): nastaví sa buffer pointer na začiatok prenášaného bloku dát a do registra data counter sa zapíše veľkosť bloku. Potom sa cyklicky opakuje prenos jednotlivých slov bloku, až kým sa neprenesie celý blok. Prenos znaku vyzerá nasledovne: CPU overí, či je zariadenie pripravené (prečíta obsah Status registra). Pokiaľ áno, z pamäťového miesta určeného pomocou buffer pointra sa načíta slovo a zapíše do buffer registra. Hneď po zápise dát do buffer registra V/V obvod sám spustí ich vysielanie. Ďalej CPU zníži obsah data countera a zvýši buffer pointer. Pokiaľ je data counter$\geq$0, cyklus sa opakuje, inak prenos končí.

Táto metóda je neefektívna, plytvá časom procesora, pretože:

- značnú časť výpočtu zaberajú rôzne testovania
- čas zaberie aj dekódovanie inštrukcií prenosu

Zaťaženie CPU sa prejaví najmä pri prenosoch väčšieho bloku dát. Za účelom 'odbremenenia' CPU vznikli podporné špecializované procesory (tzv. I/O procesory), podriadené hlavnému (univerzálnemu) procesoru, ktoré sa venujú I/O prenosu, kým CPU sa môže venovať inej činnosti. Okrem nich možno využiť aj iné techniky prenosu dát:

4.2 I/O riadené pomocou prerušení

Predstavuje z hľadiska riadenia odlišnú techniku ako programové I/O. Pri programovom I/O sa akákoľvek komunikácia inicializuje a uskutočňuje prostredníctvom procesora. Preto sa procesor 'raz za čas' musí 'pozrieť' na každé zariadenie, či nechce komunikovať a v kladnom prípade komunikáciu uskutoční. Opäť si uvedomme, že vo väčšine prípadov sú tieto testy negatívne a teda sa 'márni' čas procesora.

Pri I/O riadenom pomocou prerušení majú 'prvotnú iniciatívu' zariadenia a nie procesor. Ak majú nejakú požiadavku, hneď to oznámia procesoru. Pri existencii požiadavky procesor okamžite preruší svoju činnosť, vybaví požiadavky zariadenia a vráti sa k pôvodnej činnosti. Ako je tento mechanizmus realizovaný? Ako inak, než pomocou už spomínaných prerušení (časť III). V prípade nejakej požiadavky zariadenie vygeneruje signál INTR (interrupt request), čím nastane prerušenie a spustí sa obslužná procedúra pre dané zariadenie.

CPU má žiadateľov rozdelených na dve skupiny: na tých, ktorí môžu počkať (*maskovateľné*) a tých, ktorí musia byť vybavení okamžite (*nemaskovateľné prerušenia*). Maskovateľné prerušenia sú také, ktoré program môže buď povoliť alebo zakázať, zamaskovať (prerušenia sa budú ignorovať)- buď špeciálnymi inštrukciami, alebo nastavením určitých bitov v určitom riadiacom registri procesora. Nemaskovateľné prerušenia zakázať nemožno, musia sa vykonať okamžite - aj počas vykonávania iného prerušenia [1]. Sú priradené zariadeniam vyžadujúcim rýchle a neprerušené vybavenie svojich požiadaviek - napríklad disketová jednotka, kde by prerušenie procesora behom zápisu dát mohlo pôsobiť deštrukčne.

V prípade viacerých žiadostí o prerušenie sa vyberie to s najväčšou prioritou (opäť, viď časť III). A ako je to s prerušením počas iného prerušenia (t.j. počas vykonávania procedúry pre obsluhu iného prerušenia)? Uviedli sme (časť III), že je to možné iba ak má

[1] presnejšie povedané, počas vykonávania obslužného programu pre dané prerušenie

nové prerušenie vyššiu prioritu ako pôvodné. Prirodzene, je možné zamaskovať maskovateľné prerušenia. Počas vykonávania nemaskovateľného prerušenia sa maskovateľné prerušenia aj zakážu.

4.3 Direct memory access (DMA)

DMA (čiže *priamy prístup do pamäte*) je ďalšia metóda, spočívajúca v prenose bloku dát bez účasti procesora. Programmed i Interrupt I/O sú nevhodné na prenos väčších blokov dát, ktoré vyžadujú niektoré periférie (disk, disketa, CD).

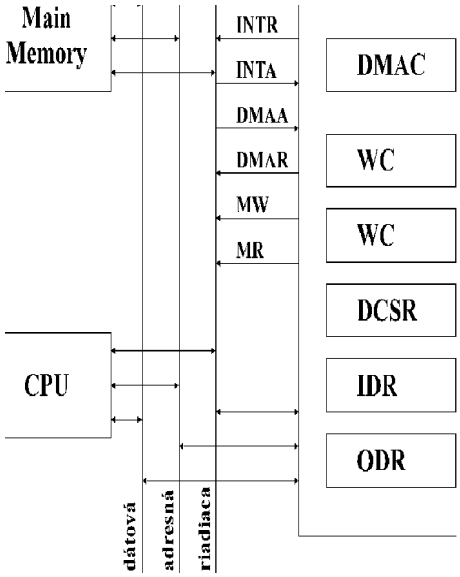
Pre ne sa používa iná I/O schéma - dáta sa prenášajú priamo medzi pamäťou a perifériou, bez sprostredkovania procesora (ktorý sa zatiaľ môže venovať inej činnosti). Uvedená schéma sa nazýva DMA (Direct Memory Access).

I/O alebo pamäť prenášajú veľký blok údajov počas jednej súvislej operácie (DMA block transfer). CPU spustí operáciu tak, že inicializuje DMA-kanál - potom je už prenos riadený DMA-radičom. Vďaka vykonávaniu 'mimo procesora' sa dosiahne rádové zvýšenie rýchlosti prenosu.

Môže sa stať, že počas prenosu chce CPU robiť s pamäťou. Keďže je však k dispozícii len jedna sada registrov MAR - MBR (resp. s pamäťou nemôžu v tom istom čase pracovať dve zariadenia, vždy len jedno), musia sa nejako dohodnúť. Obyčajne má prioritu DMAC (DMA controller), pretože je dôležité, aby prenos dát bol neprerušovaný.

DMA controller
Riadi prenos údajov v 'móde' DMA. Môže obsluhovať jedno alebo viac I/O zariadení.

DMAC pozostáva z niekoľkých registrov a riadiacich obvodov (obr. 4.1).



Obrázok 4.1: Schéma DMA-radiča

- WC (Word counter): počet prenášaných slov. Automaticky sa po prenesení slova dekrementne o 1.

- DAR (DMA adress register): adresa ďalšieho slova, ktoré sa má preniesť (adresa pamäť. miesta, kam sa má zapisovať, resp. odkiaľ sa má čítať). Automaticky sa po prenesení slova inkrementne o 1.
- ODR (Output data register): obsahuje slovo, ktoré sa má poslať I/O zariadeniu.
- IDR (Input data regiter): obsahuje slovo, ktoré prišlo z I/O zariadenia.
- DCSR (control/status register) popisuje stav DMAC a stav zariadení pripojených k DMAC. Obsahuje:
  - device enable flag
  - done/redy flag (WC=0)
  - interrupt enable flag
  - error bits
  - device status bits

Na inicializáciu DMA procesu sa používa INTR a INTA:
- CPU 'prečíta' INTR a pokiaľ je možný DMA-prenos, inicializuje ho (nastaví registre WC, DAR a DCSR) a pošle signál INTA (INT acknowledge). DMAC vyšle DMA-R (DMA request) signál. CPU odpovie DMAA (DMA acknowledge) a uvoľní riadenie zbernice. DMAC podľa požadovanej činnosti aktivuje MR (Memory read) alebo MW (Memory write). Postupne prebieha prenos jednotlivých slov, pričom sa znižuje WC. Ak je rovný nule, prenos sa skončí.

Kapitola 5

Rozhranie (Interface)

Periférne zariadenia nemôžeme pripojiť priamo k jeho zbernici počítača, pretože parametre CPU a periférie môžu byť dosť odlišné. Na prekonanie rozdielov slúži špeciálny obvod, nazývaný *rozhranie* alebo *interface*.

Interface umožňuje:

1. oddelenie V/V zariadení od zbernice a selektívny výber medzi nimi
2. prispôsobenie z hľadiska spôsobu prenosu, napríklad:
   - sériový alebo paralelný prenos
   - synchrónny alebo asynchrónny, a ďalšie ...
3. prispôsobenie z elektronického hľadiska, napríklad:
   - signálových úrovní (typicky 0/5V, 0/3.3V, alebo 24V a 20/40mA)
   - polarity signálov (invertované alebo neinvertované)
   - počtu riadiacich a dátových vodičov periférie a počtu riadiacich, adresných a dátových vodičov systémovej zbernice
   - prenosová rýchlosť, a ďalšie ...

Interface vykonáva nasledovné činnosti (má nasledovné funkcie):

- sprístupňuje procesoru stav periférie
- má schopnosť prerušovať alebo vykonať DMA (prípadne obe)
- signalizuje CPU ukončenie operácie, či operácia prebehla úspešne alebo vznikla chyba
- prenáša povely CPU perifémemu zariadeniu
- používa buffer na dočasné ukladanie dát (pri čítaní alebo zápise)
- kóduje a dekóduje údaje
- testuje paritu, resp. môže mať aj iné metódy na odhalenie chyby počas prenosu, prípadne aj opravy poškodenej informácie (samoopravné kódy)
- konvertuje medzi sériovým a paralelným tvarom, prípadne umožňuje vysielanie v synchrónnom alebo asynchrónnom móde

Časť VII

Periférne zariadenia

V predchádzajúcich častiach sme hovorili o princípoch činnosti počítača. Pod počítačom, presne povedané, rozumieme 'jadro počítačovej zostavy', t.j. procesor, vnútorné pamäte, zbernice a vstupno - výstupné obvody. Program vykonávaný v počítači však potrebuje vstupné dáta a vytvára výstupné. Informácia je v počítači reprezentovaná elektricky, pomocou rôznych úrovní napätia. Preto súčasťou počítačovej zostavy musia byť aj zariadenia, ktoré:

1. Získavajú informácie buď od užívateľa alebo z prostredia a prevádzajú ich na adekvátny elektrický signál (napr. klávesnica); a/alebo

2. Majú za úlohu znázorniť, zviditeľniť výsledky výpočtu alebo programu. Tieto zariadenia prevádzajú teda výstupnú informáciu z počítača, taktiež v elektrickom tvare, na iný tvar (napr. monitory do obrazovej podoby). Prípadne na základe výstupnej informácie realizujú nejakú činnosť (napr. riadenie sústruhov počítačom).

Vonkajšie pamäte a zariadenia slúžiace na vstup a na výstup údajov nazývame *periférne zariadenia*. Vonkajšími pamäťovými zariadeniami sme sa zaoberali v V .časti, v tejto časti sa budeme venovať ostatným perifériám.

Existuje veľké množstvo vstupných a výstupných zariadení, pretože pre mnohé aplikácie potrebujeme špecifické periférne zariadenia. V tejto kapitole porozprávame o najpoužívanejších. Opíšeme displeje, tlačiarne, klávesnice a rôzne grafické snímače a ovládače, pričom uvedieme nielen ich využitie, ale aj fyzikálne princípy na ktorých sú tieto zariadenia založené. Ich poznanie je dôležité aj z hľadiska bežného užívateľa, pretože priamočiaro podmieňuje možnosti a ohraničenia použitia toho-ktorého zariadenia.

Kapitola 1

Rozdelenie periférnych zariadení

Vstupné zariadenia môžeme rozdeliť podľa charakteru snímanej informácie (ktorou môže byť napr. text, obraz, zvuk, video, fyzikálne veličiny snímané z prostredia). Ďalej, ako sme už spomenuli, niektoré reagujú na podnety užívateľa a iné zasa snímajú prostredie. Z tohto hľadiska sú najrozšírenejšie:

- *tlačidlové ovládače*
  - klávesnica
- *grafické ovládače*
  - myš
  - joystick
  - svetelné pero
  - dotyková obrazovka
- *grafické snímače*
  - tablet
  - scanner
  - videokamera
  - snímače čiarkového kódu
- *snímače fyzikálnych veličín z prostredia*
  - mechanických veličín (napr. rýchlosti, tlaku)
  - elektrických veličín (napr. U,I,R)
  - chemických veličín (napr. hustoty)
  - snímanie zvuku (zvukový vstup)

Výstupné zariadenia, ako sme už spomenuli, prevádzajú informáciu z elektrického tvaru do iného (taktiež text, obraz, zvuk, video, atď.). Výstupná informácia prípadne predstavuje riadiace signály slúžiace na ovládanie nejakých procesov. Opäť ich môžeme rozčleniť do niekoľkých skupín:

- *zariadenia pre dočasné zobrazenie informácie*
  - displej, monitor
  - projekčné LCD panely a videosystémy
- *zariadenia pre trvalé zobrazenie informácie*
  - *tlačiarne*
    - typové
    - mozaikové
    - tepelné
    - tryskové a sublimačné
    - laserové
    - termotransferové
    - plazmové
  - *súradnicové zapisovače*
    - s valcovým posunom
    - s pohyblivým mostom (kresliace stoly)
- *počítačom riadené prístroje*
  - NC frézy, laserové obrábacie stroje
  - roboty
  - vyrezávací ploter
- *zvukový výstup*
  - hudobné syntetizátory
  - rečové syntetizátory

Vstupno - výstupné zariadenia môžu slúžiť jednak na vstup a jednak na výstup údajov.

Okrem vonkajších pamätí je ich predstaviteľom napríklad aj modem, ktorý umožňuje komunikáciu počítačov cez sieťové a telekomunikačné spoje.

Kapitola 2

Displeje

V nasledujúcej časti sa budeme zaoberať *displejmi*, výstupnými zariadeniami ktoré sa využívajú na dočasné zobrazenie informácie.

Najskôr popíšeme čo je a ako sa vytvára obrazová informácia v počítači. Výstupné zariadenia kategorizujeme podľa toho, akým spôsobom výstup (či už obraz alebo text) popisujú a uvedieme výhody a nevýhody jednotlivých spôsobov.

V druhej časti sa budeme podrobne venovať problematike vytvárania farieb a farebného výstupu.

V tretej časti popíšeme najpoužívanejšie (takpovediac 'štandardné') výstupné zariadenie počítačov- monitor. Opíšeme princípy činnosti monochromatického i farebného monitora, ich štruktúru a funkcie základných častí, grafickú kartu a rôzne videorežimy. Budeme tiež hovoriť o príčinách najčastejších porúch. Na záver sa zmienime o riadiacej časti monitora – grafickej karte.

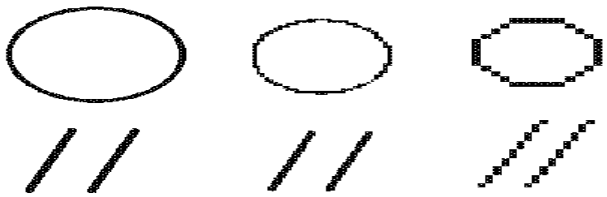
2.1 Režimy zobrazovania

Ľudské oko je vynikajúci optický systém, má však určité obmedzenia. Jednou z jeho nedokonalostí je, že nedokáže na vzdialenosť jedného metra rozlíšiť body vzdialené od seba menej ako tri desatiny milimetra. Body vzdialené menej ako 0.3 mm (pri uvedenej vzdialenosti) človek vníma ako jeden bod.

Túto nedokonalosť môžeme výhodne využiť pri vytvorení ilúzie 'verného' obrazu skutočnosti pomocou počítačového displeja. Na nasledovnom obrázku je znázornený geometrický útvar (kruh) poskladaný zo štvorčekov. Je znázornený vo viacerých veľkostiach, so stále sa zmenšujúcimi veľkosťami štvorčekov. Posledný obrázok (najmenšej veľkosti) sa skladá z tak malých štvorčekov, že už nie sme schopní rozpoznať jeho 'hranatosť' a obrázok -krúžok- sa nám javí dokonale okrúhly.

Podobne ako v prípade kruhu, každý obrázok vieme znázorniť pomocou matice skladajúcej sa z X*Y bodov. Bodom nazývame elementárny útvar (v našom príklade to bol štvorec), ktorého tvar závisí od fyzikálneho princípu daného zariadenia. Na tvare bodu však nezáleží, pokiaľ je dostatočne malý. Tiež, čím je bodov matice viac (hovoríme o vyššom *rozlíšení*), teda čím sú čísla X,Y väčšie, tým je náš obraz 'vernejší'.

Obraz na displeji môžeme vytvoriť viacerými spôsobmi – záleží od toho, o aký typ obrazu sa jedná (text, jednoduchý obraz, fotografia) a hlavne, aké elementárne obrazy dokáže displej zobraziť (body, písmená, grafické znaky). Podľa toho, akým spôsobom



Obrázok 2.1: Obraz kruhu v rôznych veľkostiach

vytvárajú obraz môžeme displeje rozdeliť na: *numerické*, *alfanumerické*, *semigrafické* a *grafické*.

- *Numerický* (alebo *číslicový*) *displej* slúži na zobrazenie číslic. Používa sa napríklad v kalkulačkách a meracích prístrojoch. Dokáže zobraziť číslice 0-9, desatinnú bodku, znamienka mínus a niektoré písmená (ktoré?).

  Ďalší, konštrukčne jednoduchší spôsob, ako môžeme znázorniť čísla, je znázornenie binárnych čísiel v normálnom alebo BCD formáte pomocou radu svetielok. Tento spôsob sa používal pri počítačoch druhej generácie a využíva sa pri jednoduchých (napr. niektorých meracích) zariadeniach.

  Pri numerickom displeji je zobrazovanou informáciou číslo. Informáciou, ktorú displej ptrebuje sú *kódy jednotlivých číslic čísla* (resp. znamienko).

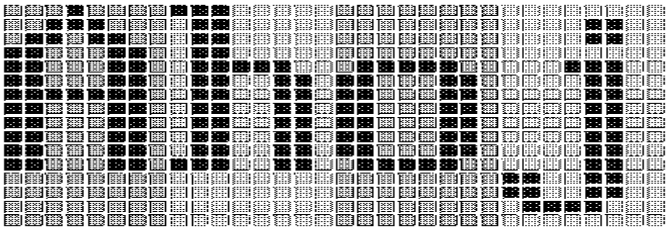
- *Alfanumerický* (alebo *abecedno-číslicový*) *displej* dokáže zobraziť písmená, číslice a niektoré symboly používané v textoch (napr. ! ? , ; . : ' + - * / = % ( ) [ ] ¡ ¿).

  Alfanumerický displej má obrazovku rozdelenú na pevne definované, neprekrývajúce sa riadky a stĺpce. Displej teda predstavuje pravouhlú maticu (tabuľku), do ktorej je možné zapisovať znaky. Displeju zadávame kódy znakov na jednotlivých pozíciách tabuľky. Nevieme posunúť znaky o bod dole či hore, môžeme udať jeho súradnice len v tvare číslo riadka-číslo stĺpca. Bežne používané rozlíšenie je 25 riadkov a 80 znakov na riadok.

  Alfanumerický displej zobrazuje text. Prenáša sa informácia- kódy znakov na jednotlivých pozíciách. Buď popisujeme celú obrazovku, alebo len určité miesto na nej - v tom prípade sa prenášajú udaje: *riadok*, *stĺpec* a *kód znaku*.

  Displej (zobrazovacie zariadenie) má vlastnú pamäť ROM, kde má uložené matice uchovávajúce obraz jednotlivých znakov. Rozmer matice je rovnaký ako rozmer znaku v bodoch (t.j. ako šírka a výška znaku). Bežné rozmery sú 5×7, 8×8 alebo 9×14 bodov. Matica obsahuje hodnoty nula a jedna; na mieste, kde má byť v obraze znaku bod je jedna a kde nemá byť bod je nula. Počítač vysiela kódy znakov, ktoré sa majú zobraziť. Displej podľa kódov vyberie príslušné matice a vykresľuje body podľa nich.

  V rastroch 5×7 alebo 8×8 sme schopní zobraziť znaky abecedy, číslice a symboly, ale nemôžeme zobraziť znaky s diakritikou. Na to je výhodný raster 9×14 bodov.

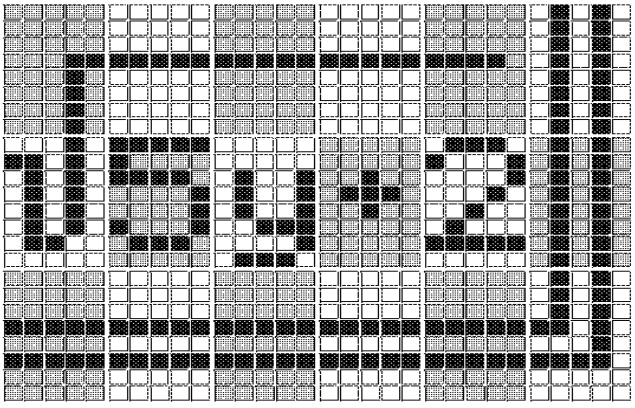


Obrázok 2.2: Príklady matíc niektorých znakov

Pozrime sa napríklad na priebeh vykresľovania textu 'AHOJ': počítač vyšle kód prislúchajúci tomuto textu displeju (napr. v ASCII je to 65,72,79,74). Displej na základe kódov určí príslušné matice znakov. Pomocou nich určí obsahy (či na danom mieste v riadku má byť bod farby pozadia, alebo bod farby pera) pre jednotlivé riadky bodov obrazovky a vykreslí ich na obrazovku.

Alfanumerický displej má obmedzenú znakovú sadu (najčastejšie 128 alebo 256 znakov). To nás výrazne obmedzuje, keď chceme písať viacerými druhmi písiem, používať slovenské znaky alebo kresliť jednoduché obrázky. Jednou z možností je, aby nám alfanumerický displej umožňoval definovať si vlastné znaky. Iným riešením je semigrafický displej.

- *Semigrafický displej* sa od alfanumerického líši len tým, že má pridané niektoré neštandardné znaky, z ktorých možno skladať vodorovné a zvislé čiary, rámiky, tiene, okienka, tiene okienok.



Obrázok 2.3: Semigrafika

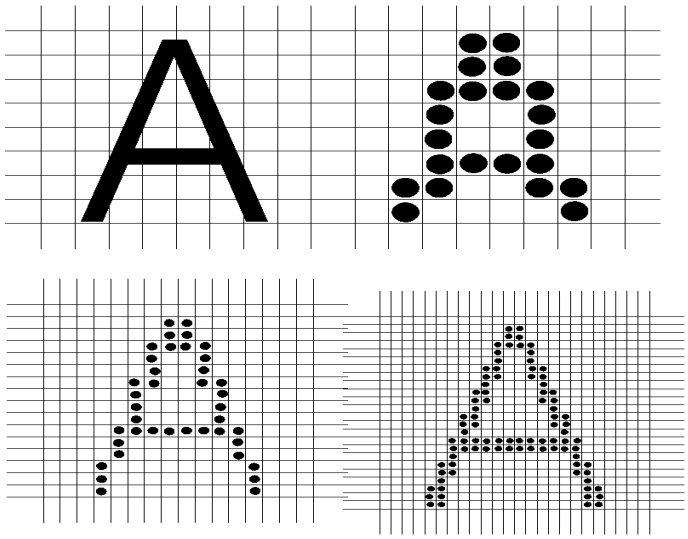
- *Grafický displej rastrový* umožňuje na rozdiel od alfanumerického a semigrafického ovládanie každého z bodov celej obrazovky. Principiálne povedané, displeju môžeme udať polohu bodu a akou farbou má byť zafarbený.

  Grafický displej umožňuje zobrazovať text i grafiku, obrazovou informáciou je v

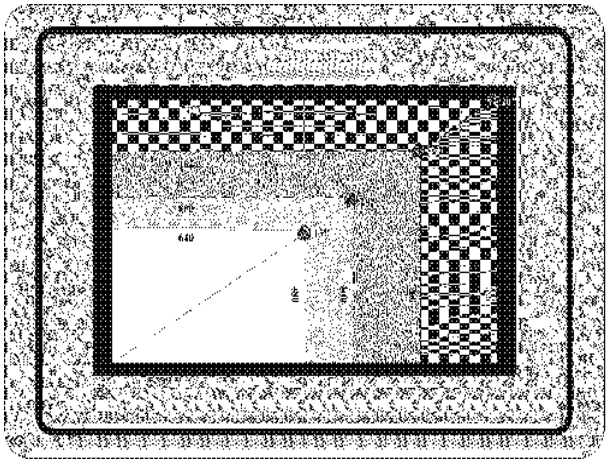
tomto prípade *farba každého bodu obrazovky.*

Samozrejme, alfanumerický displej je určitým 'variantom' grafického. Grafickým displejom tiež možno zobrazovať znaky, zobrazovaním bodov na príslušných, vhodných miestach. Z hľadiska fyzikálneho princípu sú alfanumerický, semigrafický i grafický displej rovnaké. Líšia sa len v 'logickom pohľade', v spôsobe ich programovania. Z toho vyplýva, že displej môže mať viacero možností (režimov) práce: zobrazovanie iba textu, prípadne semigrafiky– textový režim (čím sa správa ako semigrafický displej), alebo zobrazovanie grafiky– grafický režim. Môže tiež povoľovať viacero variantov týchto módov, napr. pre textový režim sa jednotlivé módy môžu mať rozdielny počet znakov v riadku–v stĺpci, rozličnú veľkosť rastra pre jednotlivé znaky, rozličný počet farieb, rôzne sady znakov. Displej tiež môže umožňovať užívateľovi definovať vlastné znaky. Pre grafické režimy sú obdobné parametre: počet riadkov a stĺpcov (rozlíšenie), počet zobraziteľných farieb.

Prečo však nevytvoriť displej iba s jedným, grafickým režimom? K čomu je dobré mať toľko rôznych režimov, a k čomu je vôbec dobrý textový režim? Odpoveď je zrejmá, ak si uvedieme jeden údaj z fyzikálnych princípov všetkých dočasných zobrazovacích zariadení: obraz treba niekoľkokrát za sekundu obnoviť (teda opäť vysvietiť tie body, ktoré majú byť vysvietené). Preto niekde musíme mať uloženú informáciu, ako obraz vytvoriť. Grafický mód 640×480 monochramatických bodov potrebuje 640×480 bitov pamäte. Naproti tomu textový mód 25×80 znakov potrebuje 25×80 bajtov pamäte (ak nezobrazujeme viac ako 256 rozličných znakov). Kvôli úspore pamäte volíme podľa druhu činnosti programu čo najvhodnejší režim práce displeja. Podobne, pre grafické režimy: ak máme videopamäť s veľkosťou 1 MB, môžeme mať režimy 640×480 bodov v 16,7 mil. farieb, 1024×768 bodov v 256 farbách, alebo 1280×1024 v 16 farbách (pozri príklady PC režimov v predposlednej kapitole). Prvý režim je vhodný na zobrazenie fotografií, posledný pri CAD-aplikáciách (kde potrebujeme veľké rozlíšenie a stačí nám malé množstvo farieb).



Obrázok 2.4: Rôzne druhy grafických režimov

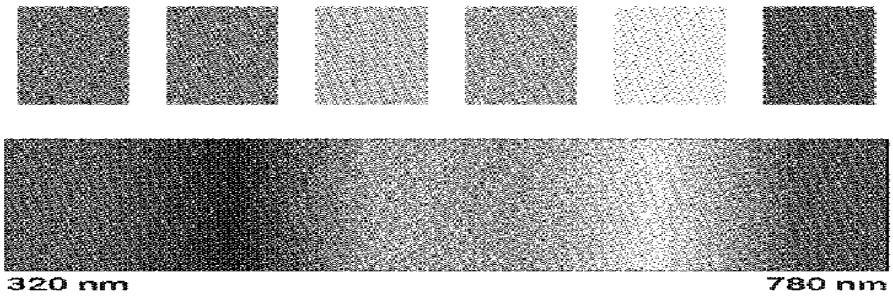


Obrázok 2.5: Doporučené rozlíšenia pre rozlične veľké uhlopriečky monitorov

2.2 Farebné zobrazovanie

V predchádzajúcej časti sme popísali, ako možno vytvoriť jednofarebný (monochrómny) obraz pomocou mozaiky bodov. Teraz povieme niečo o vytváraní farebného (polychrómneho) obrazu.

Najskôr uvedieme niekoľko základných poznatkov o svetle. Viditeľné svetlo je časťou elektromagnetického vlnenia, v rozsahu od 380nm-780nm. V prírode existuje viacero zdrojov svetla, ktoré buď vyžarujú žiarenie jednej vlnovej dĺžky, alebo vyžarujú celé spektrum vlnových dĺžok (slnko). Ľudské oko na toto žiarenie reaguje a mozog mu prisudzuje vnem určitej farby. V úseku viditeľnej časti svetla (často označovaného len ako *spektrum*) sa nachádzajú farby: červená, oranžová, žltá, zelená, modrá a purpurová (nazývaných aj ako *základné farby spektra*). Hodnota 380 nm prislúcha fialovej, 780 nm červenej. Farby sa menia plynule. Pod 380 nm je už ultrafialové žiarenie a nad 780 nm infračervené (viď nasledujúci obrázok). Viditeľné svetlo je len úzkou časťou celého spektra elektromagnetického vlnenia.



Obrázok 2.6: Farby spektra

Denné biele svetlo, ktoré vnímame, je súhrnom celého spektra farieb viditeľného svetla s približne rovnakou intenzitou. Skladá sa teda zo žiarenia všetkých farieb. Dokázal to Newton, ktorý pomocou skleneného hranola rozložil biele svetlo na farby, z ktorých sa skladá– farby spektra (viď animácie).

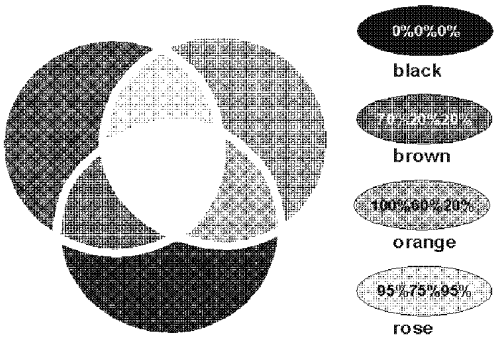
Samozrejme, nie je možné dosiahnuť aby zdroj vyžaroval všetky farby spektra a aby mali všetky rovnakú intenzitu. To ani nie je potrebné, na vnem bieleho svetla stačí vnímanie základných farieb spektra približne rovnakých intenzít. Na ilustráciu opäť uvedieme jeden pokus– 'opačný' k predchádzajúcemu. Majme kruh, ktorý je rovnomerne vyfarbený základnými farbami spektra . Ak kruh začneme otáčať veľkou rýchlosťou, pozorovateľovi sa bude zdať, že kruh je biely (viď animácie).

V spektre sa nenachádzajú všetky farby. Napríklad, nie je tu fialová farba. Fialová farba vzniká, ak naraz vnímame červené a modré svetlo.

Poznámka VII.1: V ďalšom texte budeme stotožňovať pojmy *farba* a *svetlo danej farby* , tiež budeme slovom *spektrum farieb* označovať spektrum farieb viditeľného svetla.

Ak vnímame svetlá viacerých rozličných vlnových dĺžok (rozličných farieb spektra) naraz, vnímame ich ako jednu, novú farbu. Ako sme už uviedli, vnímaním všetkých farieb spektra naraz (ich opätovným zložením) dostaneme opäť biele svetlo. Ale na znovuvytvorenie bieleho svetla nepotrebujeme úplné spektrum, stačia nám tri farby: červená, zelená a modrá.

Tieto tri farby nazývame tiež *základné farby.* Kombináciami týchto farieb pri ich rozličných intenzitách vieme 'vytvoriť' všetky ostatné farby.



Obrázok 2.7: Miešanie farieb z červenej, zelenej a modrej

Ako vidíme na obrázku, kombináciou červenej, zelenej a modrej farby môžeme dostať 8 farieb: červenú (Č), zelenú (Z), modrú (M), fialovú (Č+M), tyrkysovú (B+G), žltú (Č+Z), bielu (Č+Z+M) a tiež čiernu (neprítomnosť žiadnej zo základných farieb).

Ďalšie farby dostávame, ak kombinujeme rozličné intenzity základných farieb (príklady niektorých sú na obrázku).

Opíšme teraz ďalší pokus (viď animácie). Na bielom kartóne, ktorý osvetľujeme bielym svetlom, máme zobrazený biely, modrý, červený a zelený štvorec. Dajme pred náš svetelný zdroj červený filter. Osvetľujeme obraz červeným svetlom. Biely štvorec sa zmenil na červený, červený zostal červený, ale zelený a modrý štvorec zčerneli. Prečo? Predmet je modrý, ak odrazí modré svetlo a svetlá ostatnej farby pohltí. Ak osvetlíme obraz červeným svetlom, modrý štvorec ho celé pohltí a žiadne svetlo neodráža. Predmety, ktoré pohlcujú svetlo všetkých vlnových dĺžok svetelného zdroja majú *čiernu farbu.* Naopak, keď predmet odráža svetlo všetkých vlnových dĺžok tak má takú farbu ako svetelný zdroj (teda pri osvetlení bielym svetlom má bielu farbu).

Opísali sme si dve metódy vytvárania (miešania) farieb, ktoré sa nazývajú *aditívne* a *substraktívne.*

Aditívna (sčítacia) *metóda* sa používa pri zdrojoch svetla. Súčtom (súčasným vyžarovaním) svetiel viacerých farieb dostaneme svetlo novej farby.

Substraktívna (alebo *odčítacia*) metóda sa používa pri telesách, ktoré nie sú zdrojom svetla. Predmet sám síce nie je zdrojom svetla, ale ak naň svieti zdroj svetla, predmet určitú časť vlnenia odráža a dráždi oko rovnako, ako keby sám bol zdrojom svetla. Predmet má určitú farbu spektra, ak odráža svetlo tejto farby a svetlo iných farieb pohlcuje (presnejšie, predmet pohlcuje všetko svetelné žiarenie okrem jedného alebo viacerých intervalov spektra, ktoré odráža).

Aditívne miešanie farieb sa používa pri vytváraní farebného výstupu na monitore.

Substraktívne miešanie farieb budeme využívať pri tvorbe farebných dokumentov tlačiarňami.

Uvedieme ešte niekoľko poznatkov o ľudskom oku. Oko obsahuje niekoľko častí. Očná šošovka má meniteľnú vypuklosť (očnými svalmi), čím sa nastavuje zaostrenie. Obraz sa prenáša šošovkou na sietnicu. Sietnica je svetlocitlivá vrstva. Na svetlo reaguje elektrickými impulzmi, ktoré vysiela do mozgu. Obsahuje dva druhy buniek citlivých na svetlo– tyčinky a čapíky. Tyčinky reagujú na intenzitu svetla (jas), čapíky slúžia na vnímanie farieb.

Čapíky rozoznávajú farbu len ak má určitú intenzitu, preto pri slabom osvetlení (napr. za šera) pracujú len tyčinky. Existujú tri druhy čapíkov. Jedny sú citlivé na červené, druhé na zelené a tretie na modré svetlo.

Tyčinky síce nevnímajú farbu, no nie sú rovnako citlivé na svetlá rôznych vlnových dĺžok. Najväčšiu citlivosť majú pre zelené až žlté svetlo, asi polovičnú pre červené a veľmi malú pre modré. Celková intenzita prijímaného svetla je udaná pomerom: 59% intenzity zeleného, 30% červeného a 11% modrého svetla. Preto sa modrá plocha javí ako najtmavšia, červená je jasnejšia a zelená a žltá ako najjasnejšie.

Oko vníma farebné detaily s menšou presnosťou ako čiernobiele. Farebný obraz teda môže mať menšie rozlíšenie ako čiernobiely.

Farby môžeme charakterizovať troma veličinami: *tónom, jasom* a *sýtosťou.*

Tón farby (odtieň) je určený vlnovou dĺžkou (farbou) prevládajúcou v spektrálnom diagrame. *Jas farby* (intenzita) je určená množstvom svetelnej energie. Farby môžu mať rovnaký tón, ale zmenou intenzity dostávame nové farby. *Sýtosť farby* udáva stupeň zriedenia tejto farby s bielym svetlom. Pridávaním bielej farby sýtosť znižujeme, odoberaním zvyšujeme. Sýta farba, t.j. farba so sýtosťou 100% nemá primiešanú bielu farbu. Čierna farba má nulovú sýtosť. Sýtosť nezávisí od intenzity, zvýšením intenzity farby nezvýšime aj jej sýtosť. Ružové svetlo (ružová farba) vzniká zmiešaním červeného a bieleho svetla (červenej a bielej farby). Ak však máme len svetlo červenej farby, zvýšením alebo znížením jeho intenzity nevytvoríme ružové svetlo.

Uvedené veličiny môžeme určiť zo spektrálneho diagramu. Vrchol krivky udáva tón farby, výška krivky určuje jas a konštantná úroveň (určuje množstvo bieleho svetla v pomere k vrcholu krivky) sýtosť farby.

Poznámka VII.2: Tón a jas možno zlúžiť do jedného parametra, tzv. *farebnosti.*

Poznatky, ktoré sme v tejto kapitole uviedli sa využívajú pri vytváraní farebného výstupu, či už dočasného alebo trvalého.

2.3 Princíp práce monitora

Monitor, druh displeja, je najznámejšie a najpoužívanejšie výstupné zariadenie počítača. Jeho základom je obrazovka.

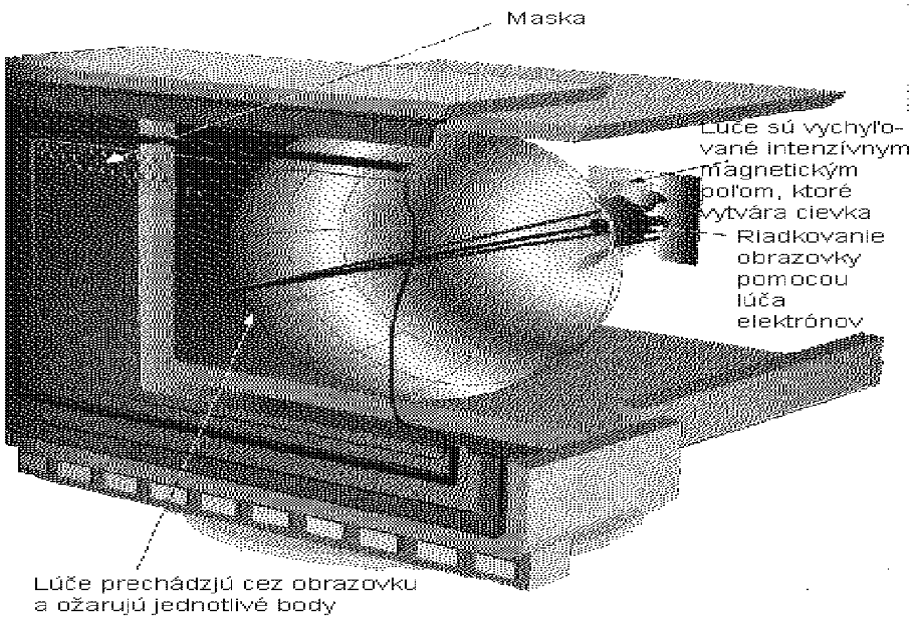
Obrazovka je zariadenie meniace elektrickú energiu na svetelnú. Napriek zdanlivej zložitosti je jej princíp jednoduchý a rovnaký ako u televíznych prijímačov.

Obrazovka je z vnútornej strany pokrytá luminoscenčnou vrstvou (tzv. luminoforom). Ak na luminofor dopadnú elektróny, na okamih sa miesto dopadu (resp. jeho určité okolie) rozžiari a istý čas vyžaruje svetelné žiarenie.

Aj keď je čas vyžarovania veľmi malý, postačí nám niekoľko krát (napr. 50 krát) za sekundu rozžiariť určitý bod, aby sme vytvorili ilúziu, že svieti stále. Opäť využijeme jednu z nedokonalostí ľudského oka, využívanú aj v kinematografii: oko má určitú 'zotrvačnosť', presnejšie, pohyb alebo iné javy odohrávajúce sa pod 1/25 sekundy nevníma. Stačí, aby kamera nasnímala za sekundu aspoň 25 obrázkov nejakého pohybu. Pri ich opätovnom vykresľovaní rovnakou rýchlosťou, akou boli spúšťané, sa vytvorí ilúzia plynulého pohybu.

Obrazovku si môžeme predstaviť ako maticu rozmerov X×Y bodov. Jednofarebný (monochromatický) monitor má celú vnútornú stranu pokrytú rovnakým luminoforom svietiacou určitou farbou.

Monitor ďalej obsahuje elektrónové delo, ktoré má schopnosť vytvoriť elektrónový lúč. Tiež obsahuje dvojicu vychyľovacích (elektromagnetických) cievok, ktoré vedia generovať magnetické pole a teda nimi môžeme vychyľovať lúč v x-ovej a y-ovej osi. Podrobnejšie sa technikými detailami nebudeme zaoberať, čitateľ si môže nájsť ďalšie fakty (napr. o tom, na akom princípe funguje elektrónové delo) nájsť jednak v stredoškolskej fyzike, alebo v odbornejšej literatúre.



Obrázok 2.8: Základné časti monitora

Riadenie činnosti monitora zabezpečuje grafická karta. Obsahuje *pamäť* (tiež nazývanú *videopamäť*), ktorá obsahuje popis obrazu. Bez újmy na všeobecnosti predpokladajme, že je v nej uložený obraz ako matica bodov. Vykreslenie obrazu sa deje nasledovne: elektrónový lúč je na začiatku nasmerovaný do ľavého horného rohu. Lúč sa začne pohybovať po hornom riadku bodov smerom vpravo. Prechádza cez jednotlivé body a z pamäte dostáva informáciu, či daný bod svieti. Ak áno, elektrónové delo vyšle impulz a rozžiari bod. Keď prejde na koniec riadka, vráti sa na začiatok nasledujúceho riadku (lúč je samozrejme vypnutý). Z pravého dolného rohu sa vracia do ľavého horného rohu. Tento proces niekoľko krát za sekundu (zväčša aspoň 50 krát) opakujeme.

Prirodzene, keď hovoríme o presúvaní lúča, nemáme na mysli fyzické presúvanie elektrónového dela (čo by trvalo príliš dlho), ale zmenu elektromagnetického poľa generovaného vychyľovacími cievkami (tým určíme, do ktorého bodu sa vyšle impulz elektrónového dela).

Opísali sme základné princípy monitora. Ďalej, podľa podrobnejších delení monitorov, napríklad podľa toho, či sa jedná o farebné či monochromatické zobrazenie, alebo podľa pohybu elektrónového lúča (viď ďalej), sa jednotlivé skupiny technicky odlišujú. Popíšeme ich. Najskôr rozdelíme monitory na *rastrové* a *vektorové*, potom na *monochromatické*, *gradované* a *farebné*.

Monitory môžu byť rastrové, alebo vektorové.

Rastrové monitory pracujú už spomenutým spôsobom: vykresľujú obraz posúvaním lúča cez všetky body obrazovky, bez ohľadu na to, či na danom mieste je alebo nie je bod.

Odlišne pracujú *vektorové monitory*. V pamäti sú uložené súradnice úsečiek (vektorov). Elektrónový lúč nevykresľuje obraz ako raster bodov, ale vykresľuje jednotlivé vektory. Najskôr sa presunie na začiatočný bod vektora. Potom sa pohybuje až do koncového bodu, pričom lúč je zapnutý. Monitor obsahuje obvody rátajúce smer vychyľovania lúča smerom ku koncovému bodu. Tento spôsob je výhodný pre určitý počet vektorov (rádovo do 10 000) a využíva sa napríklad pri CAD aplikáciach.

Uviedli sme princíp fungovania *monochromatického* monitora. Ďalšie spôsoby zobrazovania (kategorizujeme podľa množiny zobraziteľných farieb) sú: *gradované monochromatické, polychromatické* a *polychromatické gradované*.

Pri gradovanom monochromatickom monitore nerozlišujeme len či bod svieti alebo nie, ale tiež udávame jeho jas, intenzitu. Technické riešenie je jednoduché, pretože čím väčší prúd elektrónov necháme dopadať na luminofor, tým intenzívnejšie bude žiariť.

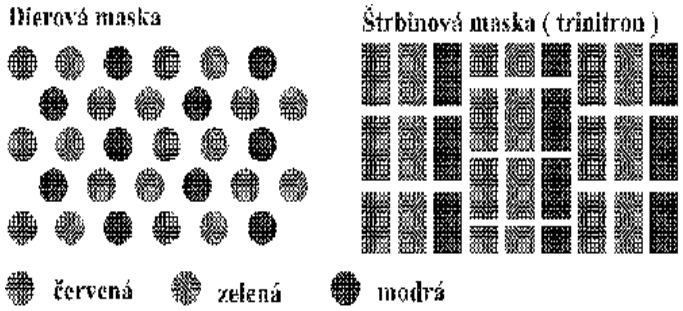
Zamerajme našu pozornosť na *farebné* monitory. V predchádzajúcom odseku sme hovorili o farbách, ich skladaní a o vytváraní farebného obrazu. Tieto poznatky sa uplatňujú pri realizácii farebného monitora.

Farebný monitor má na vnútornej strane mozaiku farebných luminoforov (Č,Z,M). Každý bod obrazovky sa skladá z troch luminoforových bodov (farebých zložiek), červenej, zelenej a modrej farby. Monitor obsahuje tri elektrónové delá, z ktorých každé osvetľuje len luminofory jednej farby.

Podľa spôsobu rozmiestnenia farebných zložiek a elektrónových diel možno obrazovky rozdeliť na obrazovky typu:

- *delta*

- *in-line*
- *trinitron*



Obrázok 2.9: Usporiadanie farebných zložiek a elektr.diel u jednotlivých typov obrazoviek

Delta obrazovky majú zložky rozmiestnené do vrcholov rovnostranného trojuholníka, in-line a trinitron obrazovky ich majú rozmiestnené v riadku. Rovnako sú rozložené elektrónové delá, aktivizujúce jednotlivé farebné zložky obrazovky (viď uvedený obrázok).

Farebná obrazovka vykresľuje obraz rovnako ako čiernobiela: Tri delá naraz vystrelia elektrónové lúče, ktorých intenzity určí grafická karta na základe toho, aký jas majú mať jednotlivé farebné zložky. Lúče sa vychyľujú magnetickým poľom v horizontálnom i vertikálnom smere. Postačuje jeden vychyľovací systém pre všetky tri lúče.

Napriek zdaniu, konštrukcia farebnej obrazovky nie je jednoduchá a naráža na niekoľko problémov. Na tienidle obrazovky sa priemerne nachádza aspoň 1 800 000 luminoforov, čo predstavuje 600 000 farebných bodov. Elektrónový lúč má šírku viacerých luminoforových bodov. Kvôli vytvoreniu správneho obrazu– správnych farieb pre jednotlivé body je nutné zabezpeiíť aby elektrónový lúč dopadal len na luminofory svojej farby. Na to slúži *maska*.

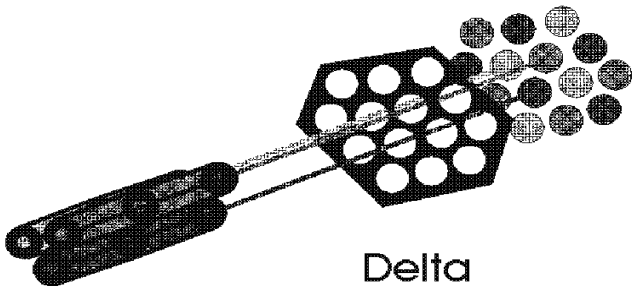
U *delta obrazoviek* je maskou tenká kovová fólia s vyleptanými otvormi. Materiál z ktorého je vyrobená (zliatina železa a niklu) má veľmi malú tepelnú rozťažnosť.

Maska je umiestnená pred vrstvou luminoforov. Pre každý bod obrazovky (t.j tri luminofory) sa v maske nachádza jeden otvor. Jedným otvorom teda prechádzajú tri lúče, ktoré sa na tomto mieste križujú. Pre každý lúč vieme určiť miesto jeho dopadu nastavením uhla, ktorým prechádza cez otvor v maske. Takže elektrónové delá nastavíme tak, aby lúč z jedného dela dopadal len na zelené luminofory, z druhého len na červené a z tretieho na modré. Pre daný bod potom budeme pre delá vyrábať analogové signály zodpovedajúce intenzitám jednotlivých farebných zložiek.

čistota farieb

Otvor masky je o čosi menší ako luminofor, ostáva nám 'rezerva' pre nasmerovanie lúča, ktorý nesmie zasahovať luminofory iných farieb. Pokiaľ však lúč zasahuje nesprávny luminofor, prejaví sa to nesprávnou reprodukciou farieb obrazu. Najzreteľnejšie sa chyba prejaví pri zobrazovaní bielej farby, ktorá sa zobrazí so stopami červenej, zelenej alebo modrej farby (viď animácie).

Delta obrazovky nastavujú čistotu farieb pomocou dvojice magnetických krúžkov, ktorými sa jemne dolaďujú uhly dopadov lúčov cez otvory masky



Obrázok 2.10: Prechod lúčov dierovou maskou

geometria obrazu

Nevýhodou delta obrazovky je deformácia obrazu na okrajoch. Lúč dopadajúci do stredu obrazovky má kruhový tvar, ale lúč dopadajúci na okraj má už tvar elipsy Podobne, rovné čiary sa nezobrazia ako rovné, celý obraz má 'poduškové' skreslenie (viď animácie).

Chyba sa odstraňuje viacerými pomocnými obvodmi obrazovky, zároveň je obrazovka tvorená povrchom gule. Súčasným trendom je však plochá obrazovka.

konvergencia

Ďalšia možná chyba v reprodukcii obrazu vzniká, ak lúče síce dopadajú na správne luminofory ('svojej farby'), ale nesprávnych bodov *statickú konvergenciu* (zbiehavosť lúčov v strede obrazovky) a *dynamickú konvergenciu* (zbiehavosť na okraji). Príčinou nesprávnej konvergencie je magnetické pole vychyľovacieho systému obrazovky. Pri vychyľovaní lúčov sú ich dráhy rôzne (napr. na obr. je dráha modrého lúča dlhšia) a preto ani uhly odchýlenia od pôvodného smeru nie sú rovnaké. Následkom toho sa lúče nekrižujú v otvore masky, ale pred ňou, prechádzajú cez rôzne otvory a rozsvecujú nesusedné luminofory (obr.24). (viď animácie).Tento jav sa najviac prejavuje na okrajoch obrazu, v strede sa neprejavuje vôbec. Závada sa odstraňuje zložitou sústavou tzv. konvergenčných obvodov.

trinitron

Obrazovka *trinitron* má luminofory umiestnené v jednej rovine, v tvare zvislých prúžkov, pričom zelený luminofor je v strede, zľava je červený a zprava je modrý.

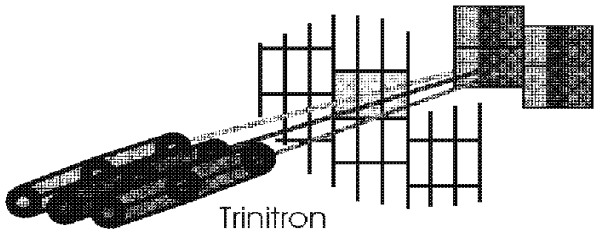
Maska je vytvorená z kovových, veľmi tenkých vertikálnych vlákien (spevnených priečnymi drôtikmi). Maskou prejde viac elektrónov (má vyššiu priepustnosť), preto je aj obraz jasnejší. Zároveň má vyšší kontrast.

Obrazovka má tvar povrchu valca.

Zabrániť deformácii bodov je jednoduché - stačí znížiť vzdialenosti vlákien. Výsledný bod nemá kruhovitý tvar, čo prispieva k vyššej ostrosti obrazu. Navyše, vertikálne rozlíšenie závisí len od presnosti zamerania lúčov.

Čistota farieb sa nastavuje ľahko, pretože delá sú v jednom riadku, stačí nastavovať jeden uhol. Zároveň sú odchýlky lúčov v zvislom smere minimálne (oproti delta obrazovke) a teda aj konvergenčné obvody sú jednoduchšie.

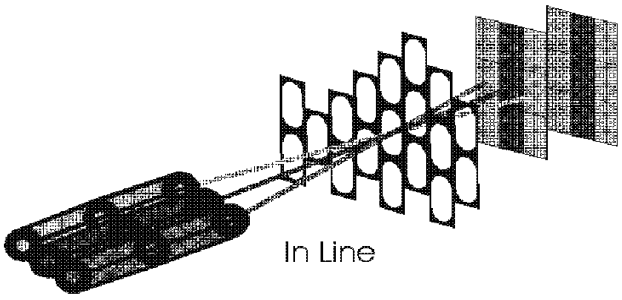
Nevýhodou trinitronu je, že oproti 'klasickej' diernej maske je jeho maska veľmi mäkká a ľahko podlieha deformácií. Magnetické pole ju dokáže trvalo poškodiť.



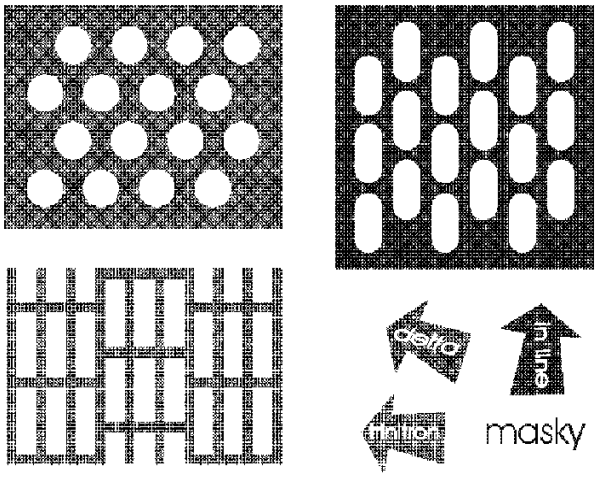
Obrázok 2.11: Prechod lúčov maskou obrazovky trinitron

in-line

In-line obrazovky sa podobajú obrazovkám trinitron. Majú luminofory i elektrónové delá umiestnené v rovine. Maska je tiež oceľová fólia s vyleptanými otvormi (pásikmi). In-line obrazovky nemajú problémy klasických delta obrazoviek s konvergenciou a vedia poskytnúť väčšie rozlíšenie ako delta obrazovky. Pre potreby počítačového výstupu sa používa in-line obrazovka s nižšími pásikmi, menšími vzdialenosťami medzi jednotlivými bodmi a jemnejším rastrom.

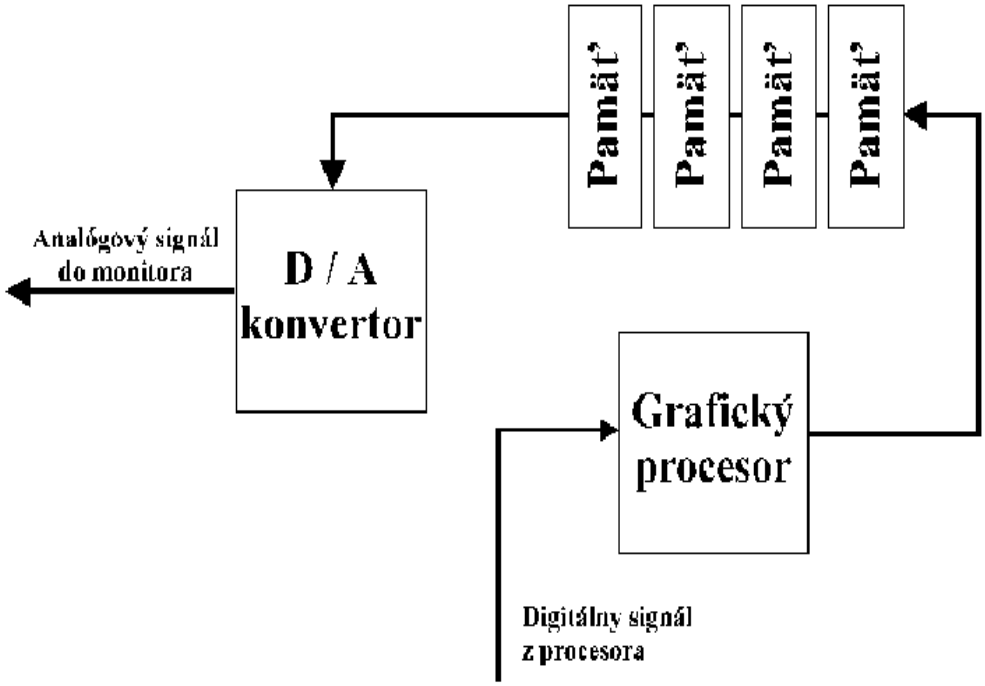


Obrázok 2.12: Prechod lúčov maskou obrazovky in-line



Obrázok 2.13: Masky delta, inline, trinitron

2.4 Grafická karta



Obrázok 2.14: Jednotlivé časti grafickej karty

Grafická karta predstavuje riadiaci obvod monitora. Skladá sa z niekoľkých častí. *Pamäť* (označovaná aj ako *videopamäť*) obsahuje informácie o jednotlivých bodoch obrazu (farbu, resp. jas). Pamäť sa skladá z dynamických pamäťových buniek a z jedného alebo viacerých posuvných registrov, ktorými možno po bitoch (t.j. sériovo) prečítať obsah celej videopamäti. Pokiaľ je posuvných registrov viacero, možno paralelne čítať navzájom disjunktné úseky pamäte. Bity na výstupe sú pripojené na *D/A konvertor*, ktorý z digitál-

nej informácie určujúcej jas, resp. farbu jednotlivých bodov vytvára analógový signál pre monitor. Výstup je synchronizovaný spolu s monitorom pomocou *synchronizačného obvodu*, ktorý obsahuje vlastný generátor hodinových impulzov. Súčasťou karty je aj *grafický procesor*, ktorý dokáže realizovať určitú sadu grafických operácií. Okrem jednoduchých primitív (napr. zmena grafického režimu, vykreslenie bodu na zadanú pozíciu či vykreslenie znaku) môžu byť implementované aj zložitejšie operácie podporujúce 2D a 3D grafiku (napr. vykreslenie štvorca a iných geometrických útvarov, vypĺňanie vzorkou, alebo rôzne algoritmy 3D grafiky).

Kapitola 3

Tlačiarne a súradnicové zapisovače

Tlačiarne a súradnicové zapisovače sú najpoužívanejšie výstupné zariadenia na trvalé (permanentné) zobrazenie informácie.

V predchádzajúcej kapitole o displejoch sme hovorili o princípoch zobrazovania textu i obrazu a tiež o tom, ako možno popísať ich vytvorenie. Myšlienky a princípy tam uvedené sú využívané aj pri tlačiarňach. Preto čitateľa odkazujeme na tieto texty, ku ktorým sa v prípade potreby môže vrátiť. Vytváranie farebného obrazu používa trocha odlišné pricípy, spôsobené odlišnými fyzikálnymi vlastnosťami pojmov 'farby svetla' a 'farby hmoty'. Preto problematike farebnej tlače budeme venovať samostatnú časť tejto kapitoly.

Najskôr popíšeme najstaršie používané typy tlačiarní- *mechanické typové* a *mechanické mozaikové* tlačiarne. Spomenieme *laserové* a *atramentové*, *sublimačné* a *voskové*. Opíšeme princípy vytvárania farebného výstupu a ako sa tieto princípy realizujú na spomenutých typoch. Záver kapitoly bude patriť súradnicovým zapisovačom.

3.1 Typové tlačiarne

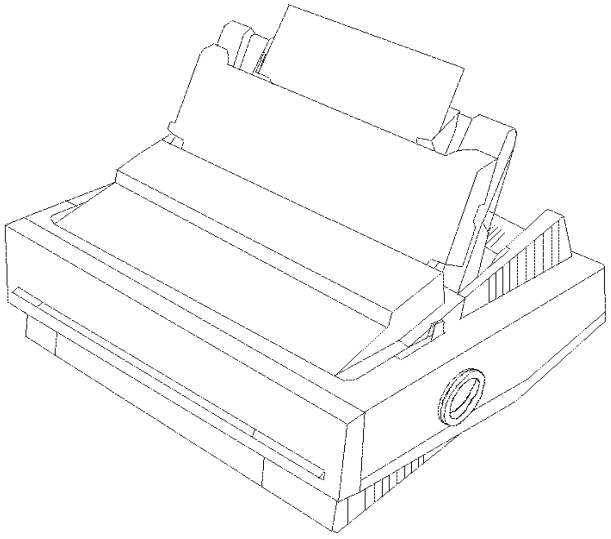
Typové tlačiarne sú prvými tlačiarňami vôbec. Princíp tlače je jednoduchý a podobný ako na písacích strojoch. Tlačiaca hlava obsahuje kovové (alebo gumenné) predlohy znakov. Počas tlače sa hlava posúva nad všetkými potencionálnymi pozíciami, kde môže byť zobrazený znak. Pre každú takú pozíciu tlačiareň obdrží kód znaku, ktorý sa má zobraziť, vyberie predlohu znaku (zodpovedajúceho danému kódu) a vytlačí ho (pritlačí predlohu na farebnú pásku, ktorá sa nachádza pred papierom).

Prevedení je niekoľko– tlačiaca hlava môže byť guľa alebo ružica (rovnako ako na písacích strojoch), valec alebo gumenný pás (takéto tlačiarne obsahujú niektoré kalkulačky s možnosťou tlače). Pre pochopenie princípov spomenutých prevedení čitateľa odkazujeme na animácie.

Typové tlačiarne majú zjavné nevýhody: sada 'tlačiteľných' znakov je obmedzená (ak ju chceme zmeniť, musíme vymeniť celú hlavu), nie je možná tlač grafiky. Na druhej strane, typové tlačiarne sú veľmi rýchle (až desiatky riadkov za sekundu pri valcových tlačiarňach), prípadne je celé zariadenie veľmi jednoduché (jednoduchšie ako pri ostatných typoch tlačiarní– čo je jeden z dôvodov, prečo sú pri vreckových kalkulačkách najvýhodnejším typom). Preto mali (a v určitých aplikáciách aj majú) svoj význam.

3.2 Mozaikové tlačiarne

Staršie typy tlačiarní používali na tlačenie textu kovové predlohy znakov (podobne ako písací stroj). Neskôr vznikla *mozaiková tlačiareň* (známa aj ako *ihličková tlačiareň*), ktorá tlačí dokument bod po bode.



Obrázok 3.1: Ihličková tlačiareň

Ihličková tlačiareň produkuje výstup na papieri pomocou tlačiacej hlavy, ktorá obsahuje skupinu kovových ihličiek. Medzi papierom a ihličkami je vložená textilná páska napustená farbou. Údery ihličiek (vyvolané elektromagneticky) spôsobujú, že sa atrament prenáša z pásky na papier (viď animácie).

Povedzme, že obraz tlačíme po bodoch. Na čo je to dobré? Každý znak možno rozložiť na body - znak nakresliť v matici M × N. Proces tlače môže vyzerať napríklad nasledovne: program požiada o vytlačenie textu a dodá kódy príslušných znakov. Obslužný program tlače má uschované v pamäti maticové obrazy všetkých znakov. Z nich vygeneruje maticový obraz riadku. Tento údaj sa pošle tlačiarni, ktorá daný riadok vytlačí. To sa opakuje do vytlačenia všetkých riadkov.

Samozrejme, pomocou ihličkových tlačiarní možno tlačiť dokumenty s ľubovolnými druhmi a veľkosťami písma a tiež tlačiť obrázky. Ale tlač obrázkov trvá dlhšie a nie je veľmi kvalitná.

Existuje viacero tried podľa počtu ihličiek v tlačiacej hlave. Prvotné, jednoihličkové tlačiarne vymizli. Najčastejšie používanými sú 9 a 24 ihličkové tlačiarne.

Čiastočne je *rýchlosť tlače* ovplyvnená i počtom ihličiek v tlačiacej hlave. Štandardná rýchlosť 9 ihličkových tlačiarní je asi 150 zn/s a 24 ihličkových cez 400 zn/s. Ovlyvnená je aj kvalita tlače. Tlačiareň s 9 ihličkami môže teoreticky poskytnúť rovnako kvalitnú tlač ako tlačiareň s 24 ihličkami, ale bude musieť prejsť každý riadok bodov vozíkom s hlavou 3 až 4 krát. Rýchlosť tlače sa tým výrazne spomalí.

Kvalita tlače závisí od *rozlíšenia*. Je to parameter udávaný v DPI, čo je angl. skratka označujúca počet bodov na palec. Priemerné ihličkové tlačiarne majú rozlíšenie okolo 150-200 dpi.

Kvalita znakov súvisí aj s tým, do akej veľkej mriežky zobrazujeme znaky.

Horizontálne rozlíšenie kolíše podľa rýchlosti opakovaného úderu ihličky vo vzťahu k rýchlosti pohybu vozíka s hlavou. Nekvalitné tlačiarne majú pomalé hlavy. Aby sa vytlačili body, ktoré sa horizontálne prekrývajú, musí sa rýchlosť hlavy výrazne spomaliť. Na to používajú pomalšie tlačiarne techniku, ktorá spočíva v tom, že pri prvom prechode vozíka po riadku vytlačia body na párnych a pri druhom prechode na nepárnych pozíciách.

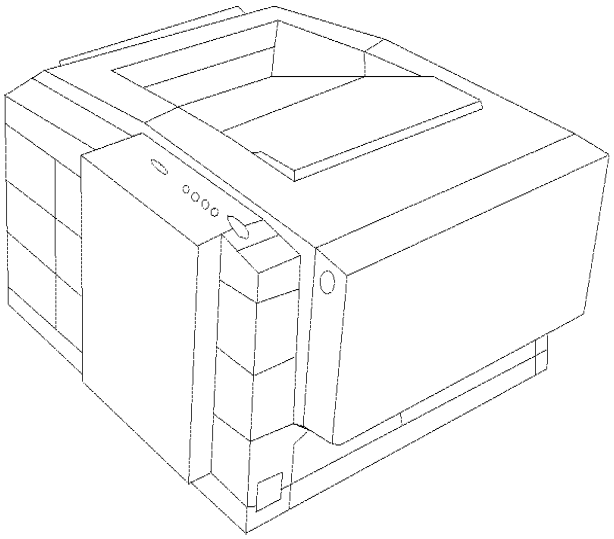
Pri hlave s 9 ihličkami, ktoré sú usporiadané do jedného stĺpca, sú získané body navzájom oddelené. Na získanie prekrývajúcich sa bodov bude treba jeden riadok tlačiť na dvakrát. Pri druhom prechode vozíka sa musí hlava zdvihnúť o polovicu bodu.

Pre kvalitu tlače sa zaviedli niektoré pojmy: *draft mode* je tzv. 'nečistopis', jednoduchý a rýchlo vytlačený koncept a *letter quality mode*, čo je kvalitnejšia tlač vyššej, 'listovej kvality', ovšem dvakrát pomalšia.

Ihličkové tlačiarne majú značne veľa nevýhod: sú hlučné, tlačia text len priemernou kvalitou a grafiku slabou kvalitou. Sú však dostatočne rýchle a pomerne lacné, vďaka čomu sa z trhu nevytratili ani po vzniku dokonalejších spôsobov tlače.

3.3 Laserové tlačiarne

Ihličkové tlačiarne tlačia text po riadkoch. Priemerná rýchlosť tlače je okolo 150 znakov za sekundu. Napriek tomu, že ihličková tlačiareň je na tlačenie textu vyhovujúca, grafiku už v požadovanej kvalite vytlačiť nevie. Tlačí ju pomaly a so slabou kvalitou, čo je spôsobené najmä nemožnosťou vytlačiť rovné čiary. S nástupom grafických prostredí sa začali dokumenty písané písmom v rôznych fontoch a v rôznych veľkostiach a neskôr sa začal text kombinovať s obrázkami. Tento problém vyriešil príchod laserových tlačiarní, umožňujúcich vysokú kvalitu tlače.



Obrázok 3.2: Laserová tlačiareň

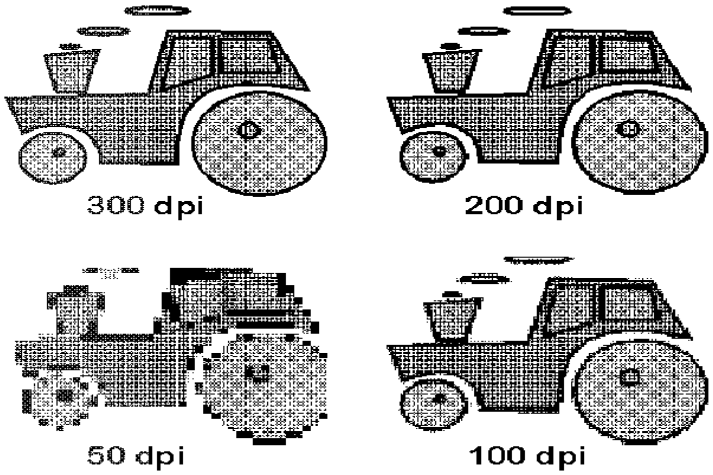
V laserových tlačiarňach sa obraz tvorí s použitím elektrostatického procesu. Laserový lúč dopadá na povrch valca cez zrkadlo. Tam, kde dopadne lúč, sa vytvorí elektrostatický

náboj. Povrch valca je aspoň taký veľký ako povrch stránky. Kde na stránke má byť bod, tam zasvietime lúčom a vytvoríme náboj. Kde nemá byť bod, to miesto 'preskočíme'. Takto vytvoríme celý obraz stránky.

Na osvietený valec sa nenesie suchý atramentový prášok, nazývaný *toner*, ktorý priľne na miesta s nábojom. Potom prejdeme papierom okolo valca, tým sa prášok prenesie na papier. Príslušná strana papiera potom prejde medzi dvoma horúcimi valčekmi. Toner sa teplom roztaví a tlak valcov ho vtlačí do papiera. Po vytlačení sa valec očistí od zbytkov toneru (viď animácie).

Tlačiareň vyžaduje papier, ktorý má určitú tepelnú odolnosť.

Laserové tlačiarne tlačia oveľa rýchlejšie ako ihličkové. Dosahujú *rýchlosť* niekoľko ppm[1]. Poskytujú aj oveľa vyššiu kvalitu. *Rozlíšenie* laserových tlačiarní býva okolo 200-600 dpi. Laserové tlačiarne poskytujú *kvalitu* veľmi blízku 'tlačiarenskej kvalite' (dokumenty teda vyzerajú ako tlačené na tlačiarenských strojoch).



Obrázok 3.3: Rozlične jemné rozlíšenia

Pri údajoch o rýchlosti však treba rozlíšiť medzi rýchlosťou motorčeka a reálnou rýchlosťou tlače dokumentu.

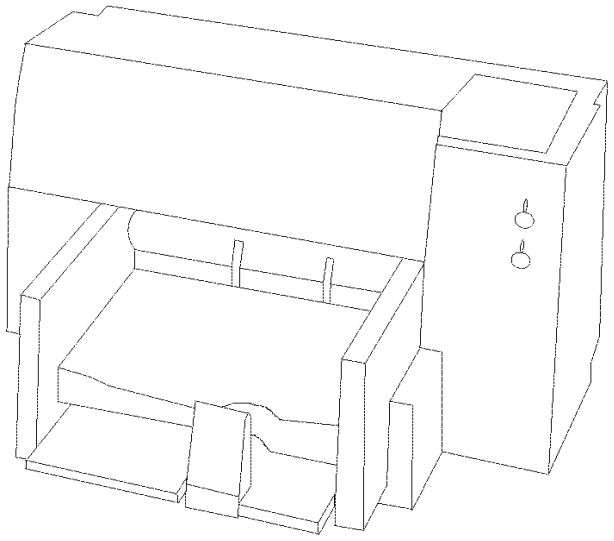
Laserové tlačiarne sú často používané na tlač zložito usporiadaných stránok, ktoré obsahujú grafiku, text a dokonca aj fotografie. Okrem toho, že počítač vysiela tlačiarni binárny popis dokumentu (čo je výhodné najmä pri obrázkoch), môžeme komunikácia prebiehať aj na úrovni povelov tvaru napríklad: 'nakresli kruh s takýmito súradnicami a vyfarbi ho' alebo 'sem napíš toto písmeno takýmto fontom a takouto veľkosťou'. Tlačiareň obsahuje program, ktorý na základe týchto povelov určí polohu príslušných bodov a vytvorí obraz stránky. Zoznamy týchto príkazov sa nazývajú *jazyky popisu stránky.* Používané sú napr. PostScript alebo PCL. Výpočet toho, kde majú byť body však môže trvať omnoho dlhšie ako samotná tlač. Preto môže vytlačenie zložitého dokumentu trvať aj niekoľko minút, hoci výrobca udáva, že tlačiareň je schopná tlačiť rýchlosťou 8 strán za minútu. V skutočnosti tento údaj hovorí, že tlačiareň je schopná vytlačiť osem identických, už pripravených strán. Keď sa stránka vytvorí, jej obraz sa uloží do vnútornej pamäte tlačiarne, podľa neho už laser môže nabiť valec hocikoľko krát.

[1] pages per minute, čiže stránok za minútu

Lacnejšie laserové tlačiarne neobsahujú jazyk popisu stránky. V tomto prípade musí počítač vyrátať pozície bodov na stránke, čo spomaľuje vykonávanie ostatných programov.

3.4 Atramentová tlačiareň

Laserové tlačiarne sú výhodné a príťažlivé pre svoju rýchlosť a kvalitu tlače. Vyhovujú aj po ergonomickej stránke, lebo sú úplne tiché. Ich nákupné ceny sú však relatívne vysoké, čo obmedzuje ich rozšírenie. Výrobcovia sa snažili vyvinúť technológiu, ktorá by poskytla podobnú kvalitu tlače, ale pri nižšej cene. *Atramentové tlačiarne* majú uspokojivú kvalitu, veľmi podobnú laserovým tlačiarňam, rýchlosť je výrazne nižšia, tlač je bezhlučná. Cena tlačiarne i náklady na tlač sú oveľa nižšie.



Obrázok 3.4: Atramentová tlačiareň

Princíp tlače je nasledovný: trysková hlava je pripevnená na pohyblivý vozík podobne ako pri ihličkových tlačiarňach. Hlava obsahuje niekoľko trysiek (otvorov). Za otvorom je *atramentová dutina*. Atrament sem tečie kanálikmi. Za atramentovou dutinou je zahrievací odpor. Je schopný zahriať priestor s atramentom a priviesť atrament do varu. Vytvorí sa plynová bublina, ktorá sa pri ďalšom vzraste teploty začína 'nafukovať' a zvyšovať svoj objem, až tlak plynu vytryskne atrament cez trysku na papier. Vytvorí sa malá čierna bodka (viď animácie).

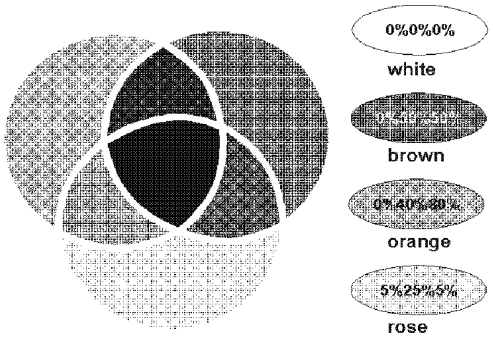
Niektoré tlačiarne používajú na vystreknutie atramentu *pizoelektrický* systém. Elektrický prúd vyvoláva vibrácie v kúsku kremíka, ktoré dokážu vystreknúť atrament.

Atramentové tlačiarne môžu tlačiť aj na obyčajný papier. Ale výsledná kvalita tlače veľmi záleží na kvalite papiera, ktorý sa v atramentových tlačiarňach používa. Vzhľadom k tomu, že atrament vytryskne z trysky ako kvapalina, spôsobuje príliš savý papier jeho rozpíjanie, čo znižuje ostrosť bodu. Aby sa dosiahlo čo najlepších výsledkov, používa sa špeciálny papier. Tento je však drahší.

3.5 Farebná tlač, voskové a sublimačné tlačiarne

Na vytváranie farieb pri farebných monitoroch sme využili optický princíp, *aditívne skladanie farieb*. Pri farebnom tlačení využijeme iný optický princíp, *substraktívne skladanie farieb*.

Substraktívne skladanie farieb je miešanie farieb z troch základných farieb: tyrkysovej, purpurovej a žltej. Tento spôsob sa označuje aj CMY (cyan, magenta, yellow). Žltá je farba, ktorá z dopadajúceho svetla odráža zelené a červené svetlo a pohlcuje svetlo ostatných farieb. Podobne tyrkysová a purpurová. Zmiešaním všetkých troch dostaneme čiernu farbu. Rôznymi kombináciami vieme vytvoriť všetky ďalšie farby.



Obrázok 3.5: Substraktívne skladanie farieb

Ako sme uviedli, zmiešaním všetkých troch základných farieb by sme mali dostať čiernu farbu. Avšak, v praxi táto čierna farba nie je 'dokonalá' a pri detailnom pohľade sa v nej objavujú farebné škvrny. Príčinou je, že nevieme namiešať ani 'dokonálé' základné farby; teda také aby odrážali svetlo presne určenej vlnovej dĺžky. Riešením je, že k trom základným farbám CMY sa pridáva štvrtá farba, čierna. Tento spôsob sa označuje CMYB (cyan, magenta, yellow, black).

Farebná tlač sa v zásade nelíši od čiernobielej. Rozdielom je, že na papier treba postupne naniesť základné tri (prípadne štyri) farebné zložky. Spôsoby realizácie farebných tlačiarní však môžu byť značne rozdielne.

Najskôr teda, na akých princípoch môžu tlačiť farebne už uvedené typy tlačiarní?

Ihličková tlačiareň sa štandartne nezvykne používať na farebnú tlač. Tlačením farebne sa rýchlosť tlače ešte viac spomalí. Princíp farebnej tlače je, že sa tlačí cez viacero farebných pások.

Atramentová tlačiareň používa farebné atramenty (CMY alebo CMYB). Náklady nie sú oveľa vyššie ako pri čiernobielej tlači. Farebná atramentová tlačiareň nie je omnoho drahšia ako čiernobiela. Dá sa prepínať farebný mód/čiernobiely mód, prípadne sa dá vymeniť farebná hlava za čiernobielu. Existujú tiež tlačiarne, ktoré majú pripojené obe hlavy, teda tlačia spôsobom CMYB. Ak pre tlač farebných dokumentov/obrázkov použijeme špeciálny papier, dosiahneme výbornú kvalitu.

Laserová tlačiareň tlačí každú stranu trikrát (opakuje popísaný proces pre každú farebnú zložku). Poskytuje vynikajúcu kvalitu.

Farebné laserové tlačiarne sú ešte dosť drahé zariadenia. Lepší pomer Výkon/Cena dávajú atramentové tlačiarne, ktoré sú cenovo dostupné.

Na dosiahnutie fotografickej kvality farebnej tlače bolo vypracovaných množstvo technológií, ktoré u čiernobielej tlače nemajú obdoby. Napríklad tlačiarne *voskové*, s *termickým prenosom* alebo *sublimačné*.

Farebné voskové tlačiarne

Podobajú sa atramentovým tlačiarňam. Používajú pevné atramenty- vosky. Ak sa tieto vosky nahrejú, premenia sa na kvapalinu a tlačiareň s nimi pracuje rovnako ako obyčajná atramentová tlačiareň. Výsledok je kvalitnejší, lebo vosk nezaschýňa vyparovaním, ale okramžite chladom tuhne. Naviac sa obrazy vytvorené pevnými atramentami nerozpíjajú.

Sublimačné tlačiarne

Využívajú sublimáciu- premenu pevnej látky do plynného stavu bez toho, že by sa premenila na kvapalinu. Silné a náhle zohriatie špeciálnych atramentov (na teplotu nad 500 ^{o}C) spôsobí vznik plynného atramentu.

3.6 Súradnicové zapisovače

Sú určené pre kreslenie schém, resp. vektorových obrázkov. Kresliacou časťou hlavy je pero, ktoré hlava presúva nad papierom, resp. jeho priblížením k papieru naň pero kreslí. Je možné kresliť nielen objekty zložené z čiar, ale aj kruhy či písmená (prirodzene, tlač textu je pomalšia ako na bežných tlačiarňach).

Existujú dva typy: *s otočným valcom* a *stolné*. Pri *stolných zapisovačoch* je súčasťou zapisovača kresliaca plocha rovná veľkosti papiera. Nad ňou sa hýbe hlava. Pri *zapisovačoch s otočným valcom* sa hlava hýbe len vo vodorovnom smere, namiesto pohybu v zvislom smere sa hýbe papier. Tým je možné kresliť aj na veľkoplošné výkresy. Oba spomenuté typy sú znázornené animáciami.

Kapitola 4

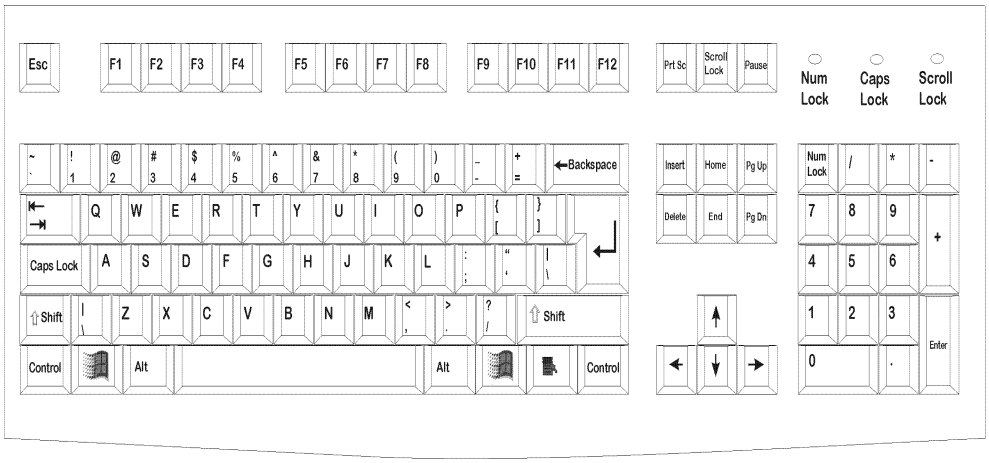
Klávesnica

Klávesnica je jedno z najpoužívanejších vstupných zariadení. Slúži na ručné vkladanie údajov. Je súčasťou mnohých zariadení: kalkulátorov, elektrických písacích strojov, monitorov, rôznych periférií a samozrejme počítačových zostáv.

Z funkčného hľadiska rozoznávame:

- *Číslicovú (Numerickú) klávesnicu* pomocou ktorej vkladáme číslice. Je súčasťou napríklad kalkulačiek.
- *Abecedno-číslicovú (Alfanumerickú) klávesnicu*, pomocou ktorej vkladáme písmená, číslice a ďalšie znaky. Je súčasťou napr. písacích strojov.
- *Funkčnú klávesnicu*, ktorej stlačenie klávesy či kombinácie klávesov môže program (alebo dané zariadenie) detekovať a vykonať príslušnú činnosť. Je súčasťou napr. tlačiarní, kde je napr. kláves na zrušenie tlače.

Klávesnica počítača obvykle združuje spomenuté funkcie – možno pomocou nej nielen vkladať alfanumerické znaky, ale aj spúšťať funkcie programu (viď nasledujúci obrázok – klávesnicu počítačov PC).



Obrázok 4.1: Klávesnica

Aj keď pre klávesnice počítačov neexistuje jednotný štandart určujúci ako má klávesnica vyzerať, používajú sa určité dohodnuté konvencie. Napríklad číslicové a znakové klávesy sú obvykle umiestnené (usporiadané) v rovnakom poradí ako na písacom stroji.

Súčasťou klávesnice môžu byť led-diódy. Štandartné PC klávesnice majú tri led diódy označené *Num Lock, Caps Lock* a *Scroll Lock*. Ich význam je čitateľovi zaiste známy– informujú o prepnutí klávesnice do zvláštneho módu (napr. *Caps Lock*: dávanie veľkých písmen).

Niektoré klávesnice majú zvukovú signalizáciu; t.j. pri stlačení klávesu sa ozve krátke pípnutie.

Jednotlivé skupiny klávesov môžu byť farebne oddelené (t.j. určitým skupinám klávesov priradíme osobitnú farbu).

V súčasnosti klávesnice zvyčajne obsahujú aj špeciálne funkčné klávesy pre podporu operačného systému a multimédií (t.j. niektoré funkčné klávesy spúšťajú určitú funkciu operačného systému či multimediálnych aplikácií). Prirodzene, operačný systém či aplikácia musia 'poznať' príslušný štandard klávesníc, aby vedeli rozpoznať a správne interpretovať aj kódy detekujúce stlačenia 'špeciálnych' kláves. Súčasťou klávesníc tiež môže byť aj varianta myši – trackball (ktorý v ďalšom texte popíšeme podrobnejšie).

Spomenuli sme niekoľko prvkov (vylepšení 'klasickej' klávesnice), ktoré môžu skvalitniť prácu s počítačom. Najdôležitejším rysom klávesnice však je, aby na nej bolo možné písať čo najpohodlnejšie a najrýchlejšie – vyžaduje sa, aby klávesnica bola *ergonomická*. Klávesy musia byť jednak citlivé na dotyk (treba zvoliť silu potrebnú na stlačenie klávesy optimálnu ľudskej ruke– klávesa nesmie klásť ani príliš veľký, ani príliš malý odpor) a tiež klávesy musia byť na klávesnici optimálne rozmiestnené.

Počítač dokáže určiť, ktorá klávesa či kombinácia kláves bola v danom okamihu stlačená. Každá klávesa má priradený nejaký kód, ktorý pri jej stlačení klávesnica pošle počítaču. Vykonávaný program tento kód môže prečítať a na základe neho vykonať nejakú činnosť.

4.1 Realizácia klávesnice - detekcia stlačenia kláves

Zamerajme teraz našu pozornosť na realizáciu klávesnice. V prvom rade musíme vedieť detekovať stlačenie jedného klávesu. Aj keď vyriešiť túto úlohu je zaiste ľahké, treba si uvedomiť, že klávesnica musí spĺňať vysoké nároky na spoľahlivosť, dlhú životnosť a zároveň musí mať čo najnižšiu cenu.

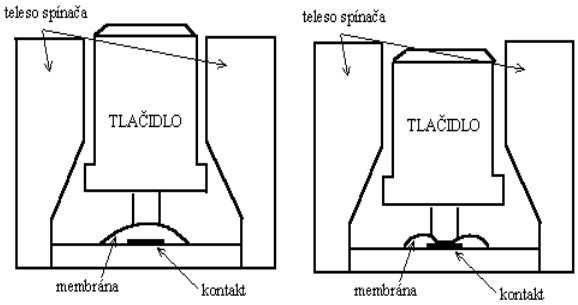
Na dosiahnutie čo najlepších výsledkov sa objavilo viacero typov spínačov. Môžme ich rozdeliť na dve skupiny, *kontaktné* a *bezkontaktné*. Uveďme niekoľko najbežnejších spôsobov realizácie kláves oboch skupín.

kontaktné spínače

Na nasledujúcom obrázku je znázornený bežný mechanický kláves. Tlačidlo je umiestnené na pružnej podložke, pod ňou sú dva od seba oddelené (ohybné) kontakty. Jeden z nich je pripojený na zdroj elektrického prúdu. Pri stlačení tlačidla sa druhý spoj prehne a dotkne sa prvého – vytvorí sa spojenie a aj cez druhý kontakt začne tiecť prúd.

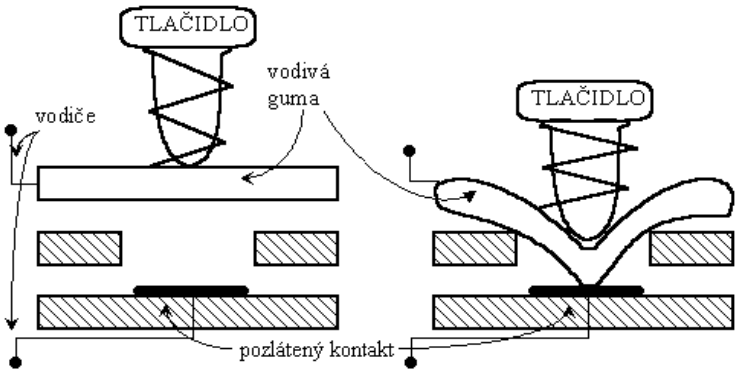
Variantou prepínacieho klávesu je aj *plochý prepínací kláves* (známy aj ako *membránový kláves*). Pri stlačení klávesu sa prehne vodivá guma (ktorou neustále preteká elektrický prúd) a dotkne sa vodivej doštičky spojenej s testovaným kontaktom.

Tieto typy spínačov sú používané pri vreckových kalkulačkách. Ich skonštruovanie na takomto princípe umožňuje, aby zaberali málo miesta.



Obrázok 4.2: Mechanický kláves

Nevýhodou je, že kladú malý odpor pri stlačení[1].



Obrázok 4.3: Membránový kláves

bezkontaktné spínače

Kontaktné spínače sú síce konštrukčne jednoduché, ale nemajú dlhú životnosť. Riešením sa ukázali bezkontaktné spínače.

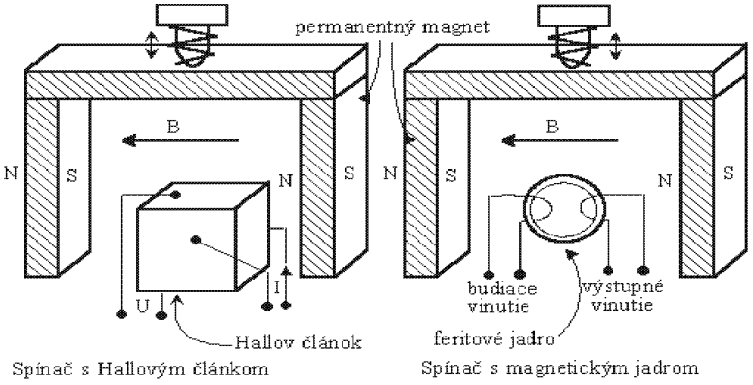
Príkladom bezkontaktného spínača je *kondenzátorový kláves*. Dve vyznačené plochy (doštičky) tvoria kondenzátor. Keď sa stlačí kláves, stredový kolík sa priblíži k doštičkám. Náboj kondenzátora sa zmení a vytvorí sa slabý elektrický prúd.

Iné dva typy sú znázornené na nasledujúcom obrázku. Bezkontaktný spínač obsahujúci *feritové jadro* pracuje ako transformátor. Na tlačidle sú umiestnené permanentné magnety. Ak tlačidlo nie je stlačené, tak týmto magnetickým poľom sa udržuje jadro v nasýtenom stave a transformátorová väzba medzi vstupným a výstupným napätím je zanedbateľne malá. Vstupné vinutie je napájané vysokofrekvenčným prúdom, pri stlačení tlačidla prestane na jadro pôsobiť magnetické pole a na výstupe sa objaví signál transformovaný zo vstupného vinutia. Výstupný signál sa ešte usmerňuje a tvaruje na tvar diskrétnych signálov. Spínač s Hallovým článkom pracuje podobne (Hallov článok je citlivý na magnetické pole).

porovnanie

- *kontaktné spínače* sú konštrukčne jednoduché a lacné. Ich výstupné signály netreba

[1] po istom čase je pre ruku únavné pracovať s takouto klávesnicou



Obrázok 4.4: Klávess (a) Halovým článkom (b) magnetickým jadrom

zosilňovať. No nemajú dlhú životnosť.

- *bezkontaktné spínače* majú neobmedzenú životnosť. Realizácia je však zložitejšia – výstupné signály treba upravovať.

V praxi sa môžeme stretnúť s oboma typmi klávesov. V periférnych zariadeniach počítačov sa používajú najčastejšie bezkontaktné klávesnice.

4.1.1 Komunikácia počítača s klávesnicou

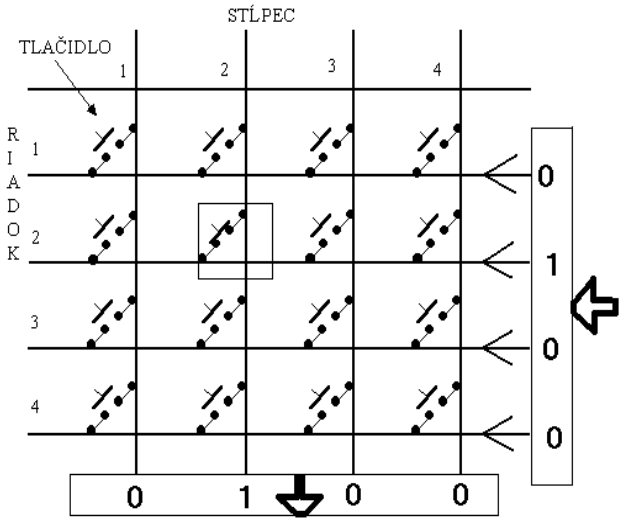
Ako zariadiť, aby klávesnica 'poznala' stlačený kláves?

Už vieme otestovať, či bol konkrétny kláves (spínač) stlačený. Uvažujme teraz bežnú klávesnicu (ktorá má viac ako 100 klávesov). Otázkou je, ako poslať počítaču údaje o tom, ktoré klávesy boli v danom okamihu stlačené.

Najjednoduchší spôsob je výstup každého klávesu pridať do celkového výstupu klávesnice. Ale potom bude na výstupe najmenej sto spojov.

Efektívnejší spôsob je mať pre viacero klávesov jeden spoj. Princíp je znázornený na nasledujúcom obrázku. Je na ňom znázornených šestnásť klávesov, usporiadaných do dvojrozmernej matice so štyrmi riadkami a štyrmi stĺpcami. Upozorňujeme, že aj keď sa spoje riadkov a stĺpcov na obrázku pretínajú, neznačí to fyzický kontakt vodičov, t.j. vetvenie prúdu (ako pri schémach obvodov, vetvenie je označené plným krúžkom). Spoje v stĺpcoch pošlime na výstup klávesnice. Na spoje v riadkoch pošlime vektor 0100 (na 1.spoj pošleme nulu, na 2.spoj jednotku, atď. . .). Ak bol stlačený niektorý kláves, dôjde k vodivému spojeniu medzi príslušným riadkom a stĺpcom. V našom prípade, ak bol stlačený kláves v druhom riadku, tak v príslušnom stĺpci sa objaví jednotka (napr. na obrázku bol stlačený jeden kláves v 2.riadku a v 2.stĺpci, preto sa v druhom stĺpci (resp. druhom bite výstupu klávesnice) objaví jednotka. Stlačenie klávesy v inom než druhom riadku nespôsobí žiadny efekt, pretože síce dôjde k prepojeniu príslušného riadka a stĺpca, no daným riadkom preteká nula. Upozorňujeme čitateľa na to, že usporiadavame jednotlivé spoje kláves, samotné klávesy môžu byť umiestnené kdekoľvek na ploche klávesnice.

Klávesnica má vlastný jednoduchý procesor (radič klávesnice) ktorý spomenutým spôsobom zisťuje, ktorá klávesa bola stlačená: kódy kláves, ktoré boli stlačené bude zisťovať postupne– po riadkoch. Postupne bude posielať na vstupy klávesnice hodnoty (1,0,. . . ,0), (0,1,0,. . . ,0) ,. . . , (0,. . . ,0,1). Po vyslaní hodnoty otestuje výstupy $x_1, \ldots, x_n$,



Obrázok 4.5: Dvojrozmerné detekovanie stlačenia klávesu

čím vie zistiť polohu stlačeného klávesu (riadok a stĺpec). Túto informáciu potom radič klávesnice posiela počítaču v jednom slove– vo forme tzv. polohového kódu[2]. Obvykle postačuje polohový kód reprezentovať bytom. Klávesnica sa pripája k sériovému portu.

Obslužný program klávesnice uloží kódy stlačených kláves na pevne určené miesto (do osobitého buffera), odkiaľ ich bežiaci program môže prečítať a interpretovať (t.j. na základe nich vykonať nejakú činnosť). Do buffera sa nemusia ukladať len polohové kódy– obslužný program môže ihneď prekladať polohové kódy do kódov im zodpovedajúcich znakov, napr. v kódovaní ASCII.

Nielen každý kláves má vlastný kód, ale aj súčasné stlačenie viacerých klávesov môže mať svoj vlastný (tzv. *polohový*) kód. (napr. súčasné stlačenie klávesu ALT a iného klávesu má priradené osobitý polohový kód).

V prípade detekcie stlačenia klávesy klávesnica posiela procesoru žiadosť o prerušenie. Obslužný program prečíta polohový kód stlačeného klávesu. Obslužný program tiež môže polohové kódy automaticky prevádzať do nejakého textového kódu, napr. ASCII.

[2]foriem zakódovania môže byť viacero, napr. ak je Riadok n-bitový vektor a Stĺpec m-bitový, tak Polohový kód vytvoríme zreťazením týchto dvoch vektorov. Iným spôsobom je Polohový kód=Riadok*(Počet Stĺpcov riadku) + (Stĺpec-1).

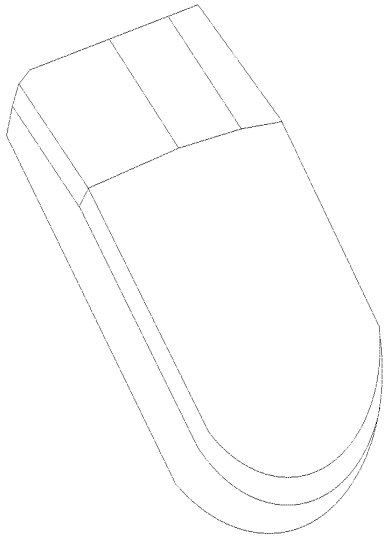
Kapitola 5

Grafické ovládače

Pôvodne sa počítače ovládali len pomocou klávesnice. Nástup grafických prostredí tento stav zmenil. Programy dostali 'novú tvár'– štandardom sa stalo symbolické znázornenie funkcií programov či samotných programov graficky, pomocou obrázkov(ikon). Takisto, objavili sa aplikácie pre vytváranie a spracovanie obrázkov i ďalšej grafiky. Manipulácia s obrázkami je však pomocou klávesnice dosť náročná. Preto sa na ovládanie počítačov začali používať okrem klávesnice aj tzv. *polohovacie zariadenia*. Sú to zariadenia, pomocou ktorých možno plynule ovládať pohyb kurzora (alebo iného objektu) po obrazovke. Medzi najznámejšie patrí *myš* a jej varianty (napr. *trackball*), *jojstyck*, *svetelné pero* a *dotyková obrazovka*, ktoré podrobnejšie opíšeme v tejto kapitole.

5.1 Myš

Najpoužívanejším grafickým ovládačom je myš. Prvú vyrobila firma Xerox. Stala sa populárnou najmä vďaka operačnému systému Windows, na ktorom sa ukázala vysoká komfortnosť ovládania grafických prostredí myšou oproti klávesnici.

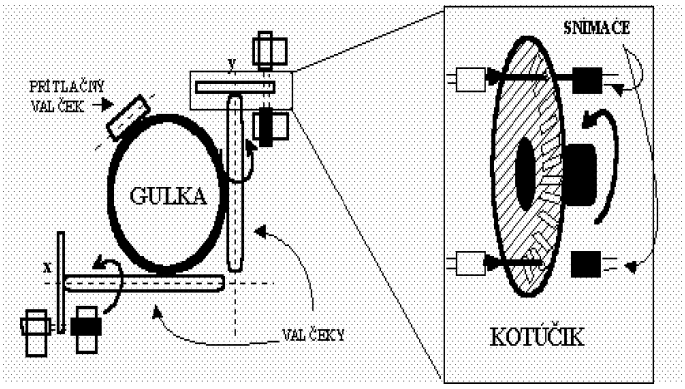


Obrázok 5.1: Myš

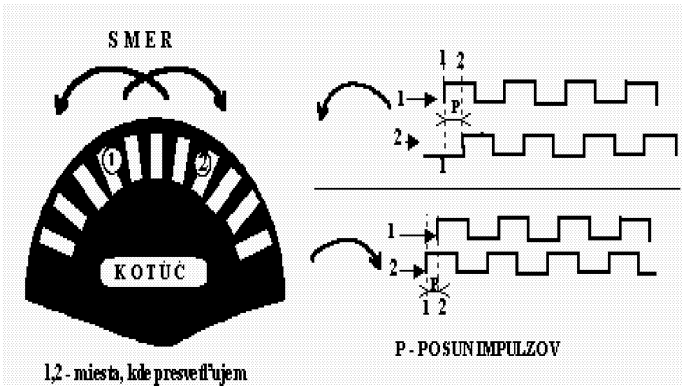
Myš sa pohybuje po podložke a od jej pohybu sa odvodzuje pohyb kurzora po obrazovke. Máva dve alebo tri tlačidlá, ktorých využitie (čiže interpretácia stlačenia) závisí

od daného programu.

Opíšeme princíp fungovania *mechanickej myši* (nasl. obrázok). Myš sa pohybuje po podložke. Na spodu má pohyblivú guličku. Pohyb myši sa prenáša na pohyb guličky, pohyb guličky sa prenáša na dva kolmo postavené valčeky, v x-ovom a y-ovom smere (doleuvedeny obrazok). Ak posúvame myšou v x-ovom smere, gulička otáča x-ový valček a po y-ovom valčeku sa šmýka (obvodová rýchlosť gulôčky je v dotykovom bode s valcom Y nulová) . Veľkosť otáčania valčeku je priamo úmerná obvodovej rýchlosti guličky (teda čím rýchlejšie pohybujeme myšou, tým rýchlejšie sa otáča valček). Podobne, ak hýbeme myšou v iných smeroch, smer pohybu sa rozloží na dva vektory, x-ový a y-ový (ktoré sú na seba navzájom kolmé) a úmerne veľkosti týchto vektorov sa natočia valčeky x,y. S osou každého z nich je spojený kotúčik s otvormi. Presvetľujeme ho dvojicou led-diód. Teleso kotúčika pri pohybe prerušuje ich svetlo, dopadajúce na protiľahlé fototranzistory. Takto dokážeme určiť veľkosť posunutia. Ako však určiť smer posunutia (čiže smer otáčania kotúčika)? Na vyriešenie tohto problému použijeme dve fototranzistory. Sú posunuté od seba na pol okienka, takže ak sa jeden fototranzistor zatemňuje, druhý sa odkrýva. Zo začiatku signálov (vzájomného fázového posunu) vieme určiť smer otáčania.



Obrázok 5.2: Princípy myši



Obrázok 5.3: Určenie smeru pohybu myši

Počítaču sa potom pošle relatívna zmena polohy- veľkosť vychýlenia (v smere osi X a Y).

Väčšina myší je káblom spojená s počítačom. Existujú však aj myši, ktoré komunikujú s počítačom pomocou infračerveného svetla alebo rádiových vĺn. Nazývame ich *bezdrôtové myši.*

Pri počítačoch PC pripájame myš na sériový alebo paralelný port. Ekonomickejšie je pripojiť myš na sériový port, pretože objem prenášaných dát je pomerne malý a sériový port je preto plne postačujúci. Pripájanie na sériový port je najrozšírenejšie, no možno sa stretnúť aj s pripájaním na port paralelný. Myš pripojená na paralelný port sa nazýva *zbernicová myš.* Táto myš nemôže využívať port vyhradený pre paralelné tlačiarne, ale vyžaduje svoju vlastnú IO kartu. Niektoré myši môžeme pripájať do oboch portov, sériového aj paralelného, lebo majú zabudovaný konverzný obvod.

Okrem mechanických myší existujú aj myši *nemechanické*, určujúce zmenu polohy myši na podložke na inom, ako na mechanickom princípe. Príkladom takýchto zariadení je *optická myš*, ktorej princíp je nasledovný: na spodu myši je umiestnený svetelný zdroj, ktorý (pod vhodným uhlom) vyžaruje infračervený lúč. Ten dopadá na špeciálnu podložku a odráža do snímača umiestneného v zadnej časti myši. Podložka je najčastejšie kovová a je pokrytá hustou mriežkou čiernych čiar. Odraz do snímača nastane iba ak nebola prerušená žiadna z čiar. Takto vieme určiť veľkosť pohybu. Podobne určujeme aj smer.

5.2 Joystick

Známy aj ako *pákový ovládač*, je pomôcka určená pre ovládanie hier. Má tvar zvislo postavenej páky, ktorú možno vychýliť do určitého smeru. Podľa smeru vychýlenia, prípadne aj veľkosti vychýlenia sa uskutoční pohyb objektu v hre. Súčasťou joysticku je aj jedno alebo viacero tlačidiel, ktoré v danej hre môžu mať rôzny význam.

Existujú dva typy joystickov: *analógové* a *digitálne*. *Digitálne* pákové polohovacie zariadenia pracujú v pomerne jednoduchom móde 'áno alebo nie'. Rozpoznáva sa iba smer pohybu, ale nie veľkosť pohybu. Navyše, joystick rozoznáva len osem smerov vychýlenia. *Analógové joysticky* majú proporcionálne ovládanie, t.j. malé vychýlenie páky vyvolá malý pohyb objektu na obrazovke (pohyb na krátku vzdialenosť), väčší pohyb pákou väčší pohyb objektu. Používa sa napríklad v leteckých simulátoroch.

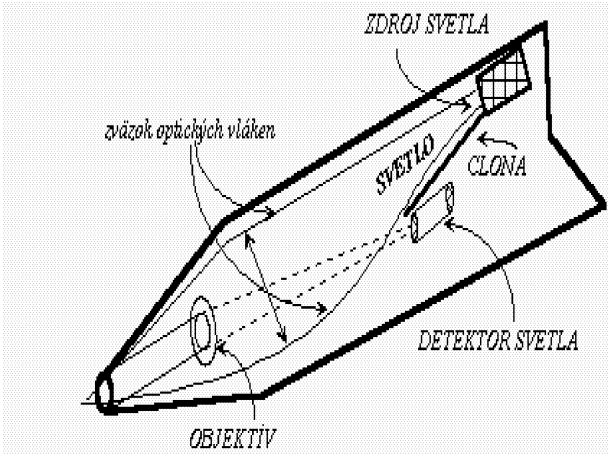
Princíp joysticku je jednoduchý. *Digitálny joystick* sa skladá z jednoduchého štvorpólového snímača a jedného prídavného tlačidla. Tieto sa 'pripoja' k piatim bitom vstupnej brány niektorého z V/V obvodov počítača. Ak vychýlime joystick do jedného zo štyroch smerov (vľavo, vpravo, hore, dole) tak vytvoríme spojenie jedného z týchto štyroch pólov spínača s hrotom páky, ktorý je uzemnený. Preto sa na príslušnom vodiči objaví nulové napätie (čiže logická nula) a preto príslušný bit vstupu V/V obvodu bude nulový. Ak vychýlime joystick do jedného zo 4 šikmých smerov, napr. vpravo hore, uzemnia sa súčasne pól spínača vpravo a pól spínača hore, a na vstupe V/V brány bude slovo s dvoma nulovými bitmi na mieste bitu horného pólu a pravého pólu (viď animácie).

Analógový joystick obsahuje dva potenciometre (čo sú prvky meniace svoj odpor v závislosti od ich vychýlenia). Odpor je priamo úmerný veľkosti vychýlenia a možno ho zmerať napríklad ADC prevodníkom. Pomocou jedného sa detekuje vektor vychýlenia v x-ovom smere, druhým v y-ovom smere.

5.3 Svetelné pero

Svetelné pero je pomôcka, ktorou môžeme takisto ovládať pohyb kurzora. Na rozdiel od myši ním ukazujeme priamo na jednotlivé body obrazovky.

Svetelné pero má tvar pera, káblom spojeného s počítačom. Jeho priblížením sa k určitému miestu na obrazovke svetelné pero počítaču vyšle absolútne súradnice 'dotykového' bodu[1] na obrazovke. Využitie pera opäť závisí od bežiaceho programu– môže to byť výber z množiny ponúk, kreslenie či písanie. Používanie pera na kreslenie či písanie je pre človeka prirodzenejšie ako používanie myši, pretože práca so svetelným perom je v podstate analógiou práce s 'klasickým' perom, s ktorým pracujeme s veľkou jemnosťou a presnosťou.



Obrázok 5.4: Svet.pero

Princíp svetelného pera je pritom veľmi jednoduchý. Pero je tvorené jedným fotoelektrickým snímačom - v telese pera sa nachádza fotodióda alebo fototranzistor a šošovka sústreďujúca dopadajúci lúč na ich svetlocitlivé plochy. Ak pero dostatočne priblížime k obrazovke, tak sníma jas bodu pred sebou. Ako sme spomenuli, aj keď monitor vytvára ilúziu, že všetky body obrazu svietia nezmeneným jasom, v skutočnosti ich jas klesá a obraz musí byť neustále obnovovaný elektrickým lúčom. Okamžitú polohu lúča dokáže určiť grafická karta. V okamihu keď lúč osvieti bod na ktorý ukazuje svetelné pero, s pomocou fototranzistora sa detekuje zmena jasu a vyšle sa impulz grafickej karte, ktorá z aktuálnej polohy lúča určí polohu bodu, na ktorý pero ukazuje. Túto polohu si zapamätá, prípadne vygeneruje prerušenie a odovzdá polohu obslužnému programu. Obmedzením tohto princípu je, že bod na ktorý chceme 'ukázať' perom nesmie byť úplne čierny, musí mať nenulový jas. Postačujúcim však je aj malá zmena čiernej farby, napr. na sivú.

5.4 Dotyková obrazovka

Svetelné pero umožňuje veľmi pohodlným spôsobom vyberať z ponúk znázornených na displeji. 'Ukázanie' na objekt na displeji je totiž prirodzenejšie ako jeho určovanie pomocou myši či klávesnice. Ešte prirodzenejšie je však ukazovať prstom.

[1] t.j. bodu, v ktorom došlo k dotyku, resp. dostačnému priblíženiu pera k obrazovke

Dotyková obrazovka vyzerá zvonku ako filter monitora a rovnako ako filter sa aj ona pripevňuje pred obrazovku. Podobne ako svetelné pero, aj ona počítaču oznamuje absolútne súradnice označeného ('ukázaného') bodu obrazovky.

Existuje niekoľko druhov dotykových obrazoviek. *Finger-screen* reaguje na priblíženie prsta k povrchu obrazovky. Zariadenie obsahuje dva rady zdrojov svetla (infračervených LED-diód) a dva rady fotosnímačov umiestnených oproti sebe (t.j. oproti každej LED-dióde sa nachádza fotosnímač). Rady LED-diódy sú umiestnené kolmo na seba a teda nad celým povrchom obrazovky vytvárajú sieť (mriežku) vodorovných a zvislých infračervených lúčov. Fotosnímače sú umiestnené oproti diódam. Ak sa prst priblíži k obrazovke, preruší niektorý zvislý a vodorovný lúč vysielaný LED-diódami do protiľahlých tranzistorov. Počítaču sa pošlú súradnice 'ukázaného' bodu - t.j. poradové číslo LED-diód vysielajúcich prerušený zvislý a vodorovný lúč. Pre svoju veľkosť má však prst 'malú rozlišovaciu schopnosť'. Preto sa používa aj druhá varianta dotykovej obrazovky nazývaná *touch screen*.

Touch screen používa pre výber špeciálne ukazovátko v tvare ceruzky, podobné svetelnému peru. Touch screen však používa iný fyzikálny princíp (ktorý bližšie popíšeme pri tablete - povrch obrazovky je pokrytý jemnými, okom nepostrehnuteľnými vodičmi, ktoré sú od seba vzájomne oddelené nevodivou vrstvou. Po priblížení ukazovátka s elektromagnetickým hrotom sa v okolí styčného bodu indukuje elektromagnetické pole, pričom najsilnejšie je práve v tomto bode.

Touch screen sa využíva najmä v diároch s LCD displejmi. Jedným zo súčasných trendov elektronických diárov je využiť čo najviac možnosti touch screenu a vytvoriť tak diár novej generácie. Diár s touch-screenom nepotrebuje klávesnicu, ukazovátkom je možné 'vyberať', resp. 'stláčať' klávesy virtuálnej klávesnice znázornenej na displeji. Takisto je možné vyberať z rôznych ponúk, prípadne kresliť obrázky a písať. Rukou písané písmo je automaticky rozpoznávané a prevádzané do digitálnej formy, napr. ASCII kódu. Touch screen teda nahrádza klávesnicu i svetelné pero.

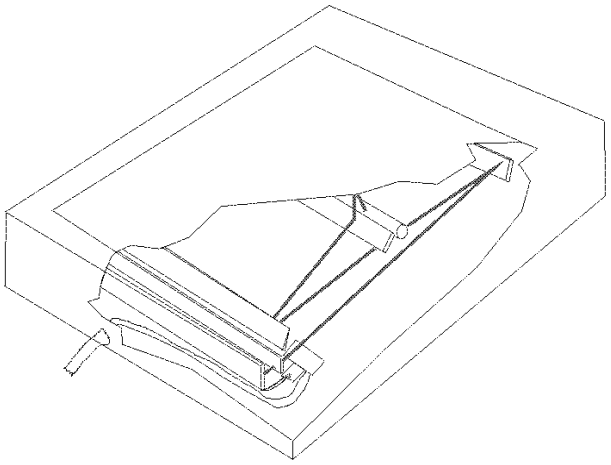
Kapitola 6

Grafické snímače

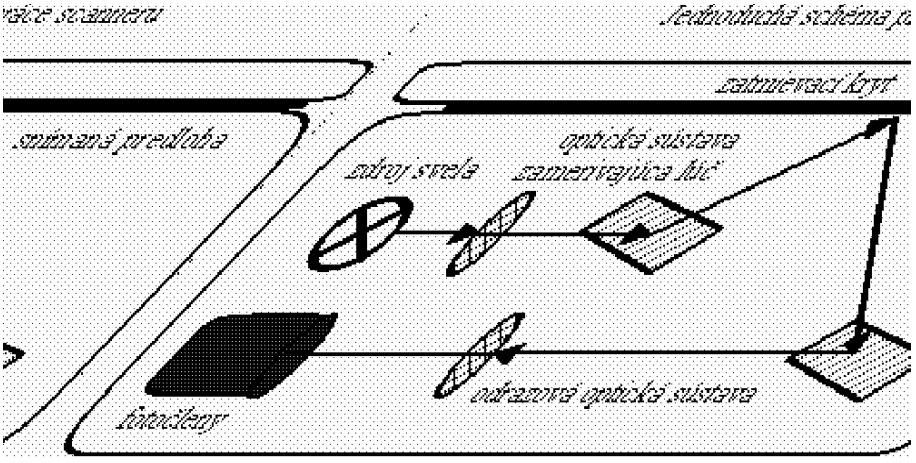
6.1 Scanner

Scanner umožňuje načítanie obrazových predlôh do pamäte počítača. Prevádza ich na digitálny tvar, teda do číselného tvaru obsahujúceho nuly a jednotky. Nasnímané obrazové predlohy potom možno pomocou rôznych grafických programov prezerať, upravovať či tlačiť. Druhý spôsob využitia je, že nasnímame text a potom použijeme špeciálny program na rozoznávanie písma.

Scanner pracuje nasledovne: obrazová predloha sa umiestni do scannera obrazom dole. Svetelný zdroj (fluorescenčná trubica) osvetľujúca predlohu je umiestnená na pohyblivej rampe. Svetlo sa odráža od obrazu a od systému pohybujúcich sa zrkadiel. Využíva sa vlastnosť, že tmavé oblasti (čierne body dokumentu) odrážajú len málo svetla, zatiaľ čo svetlé plochy odrážajú viac svetla. Odrážané svetlo je zrkadlami smerované na rampu fotodetektorov. Tieto detektory konvertujú svetlo na elektrický prúd. Čím je intenzita svetla väčšia, tým väčšie napätie generujú. Každý detektor je pripojený k osobitému kondenzátoru, všetky kondenzátory sú spojené a tvoria analógový posuvný register, v ktorom možno posúvať napätia uložené v jednotlivých kondenzátoroch na nasledovnú pozíciu– až ku krajnému prvku, pripojenému k vyhodnocovacím obvodom. Krajný prvok je pripojený na vstup analógovo-digitálnych konvertorov, ktoré príslušnému napätiu priradia zodpovedajúcu číselnú hodnotu.



Obrázok 6.1: Práca scannera



Obrázok 6.2: Scanner (princíp)

V jednom kroku sa sníma celý riadok bodov. Na vstup ďalšieho riadku krokovací motorček posunie zrkadlovú plochu o hodnotu zodpovedajúcu *rozlíšeniu scanera*.

Bežný je interval 256 hodnôt pri čiernobielom snímaní (udáva sa ako 256 odtieňov šedi).

Farebné scannery pracujú podobným spôsobom. Pre každý bod snímajú intenzitu červenej, zelenej a modrej zložky odrazeného svetla a prevádzajú intenzity do digitálnej formy. Čiže pre každý bod dostaneme vektor s troma zložkami, vyjadrujúcimi intenzitu jeho farebných zložiek (R,G,B).

Najjednoduchším spôsobom realizácie je obrazovú predlohu snímať trikrát– najskôr umiestniť pred fotodetektory červený, potom zelený a nakoniec modrý filter. Rýchlejšie pracuje scanner, ktorý používa tri snímacie rampy– pred každou je umiestnený osobitý farebný filter a svetlo z predlohy je odrážané na každú z nich. Týmto spôsobom je možné predlohu zosnímať v jednom kroku.

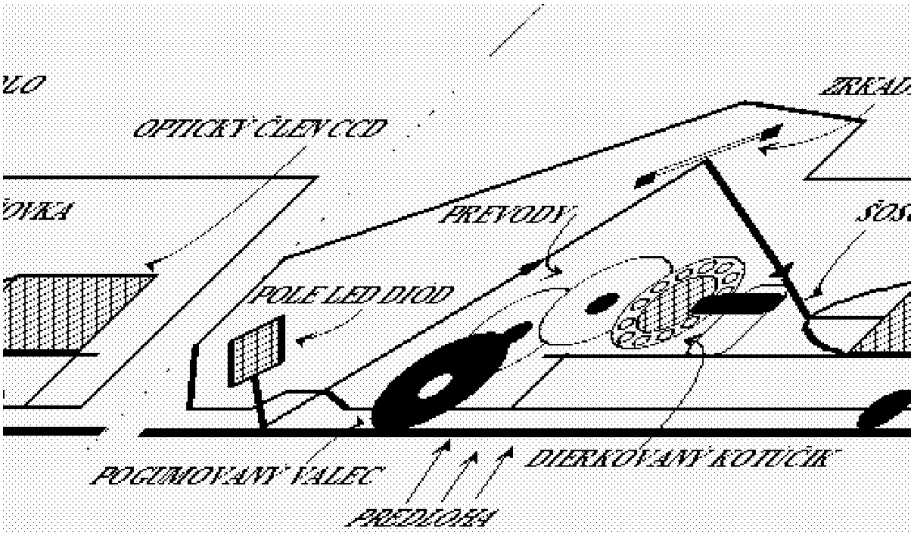
Scannery môžu byť buď *ručné* alebo *stolné*. Ručné sú lacnejšie, no keďže majú malé zorné pole (asi 12cm), tak sa skôr hodia na snímanie menších predlôh (šírky 12 cm), do stolných scannerov možno vkladať aj predlohy väčších formátov - najčastejšie A4 až A3.

6.2 Tablet

Tablet, nazývaný aj *digitizér* je vstupné zariadenie často využívané v stavebnom, strojnom a elektrotechnickom inžinierstve. Možno pomocou neho prekresľovať schémy a výkresy– používa sa CAD-aplikáciách (t.j. v počítačovom návrhu dizajnu).

Najväčšia časť tabletu má tvar plochej dosky rozmerov A5, A4, alebo A3, ku ktorej je pripojené kresliace pravítko s vyznačeným bodom a tlačidlami. Po stlačení tlačidla tablet vyšle počítaču súradnice bodu, na ktorý ukazuje vyznačený bod pravítka. Súradnice už každý program interpretuje osobitne: napríklad označením dvoch bodov sa nakreslí úsečka spájajúca tieto dva body; alebo sa nakreslí kruh so stredom v prvom bode prechádzajúci cez druhý bod a podobne....

Povrch kresliacej plochy je pokrytý radmi nepretínajúcich sa zvislých a vodorovných elektrických vodičov. Uložené sú tesne pod povrchom kresliacej plochy, zaliate do plastu



Obrázok 6.3: Rucny Scanner (princíp)

(hmoty, z ktorej je zložená kresliaca plocha). Hustota pokrytia kresliacej plochy vodičmi zodpovedá rozlíšeniu tabletu. Po stlačení tlačidla na pravítku pravítko generuje elektromagnetický impulz (vo vyznačenom bode pravítka). Ten spôsobí, že sa vo vodičoch nachádzajúcich sa v okolí vyznačeného bodu generuje el. prúd. Jeho intenzita je úmerná vzdialenosti od vyznačeného bodu pravítka; najväčšia je v tom zvislom a vodorovnom vodiči, ktorých priesečník je bod najbližší vyznačenému bodu pravítka (viď animácie).

Súčasťou tabletu je niekoľko tlačidiel, ktorými môžeme vyvolať rôzne funkcie bežiacej aplikácie. Môže to byť nakreslenie objektu na danej pozícií vyznačeného bodu pravítka (napr. vykreslenie štvorca, kruhu, elektronickej súčiastky), grafická operácia (napr. vyplnenie objektu) a iné. Na plochu s tlačidlami sa umiestni šablóna pokrývajúca povrch všetkých tlačidiel s grafickými symbolmi v mieste tlačidiel. Šablóna je špecifická pre danú aplikáciu, grafické symboly na tlačidlách popisujú príslušné funkcie aplikácie, ktoré stlačením vyvoláme.

Opísali sme niektoré najznámejšie a najpoužívanejšie periférie: monitory, tlačiarne, klávesnice, myši, joysticky a ďalšie polohovacie a snímacie zariadenia. Je nemožné (a aj zbytočné) opísať všetky periférie, pretože je ich nepreberné množstvo– na špeciálne úlohy sa často používajú špecifické periférie. Takisto do tejto publikácie neboli z rozsahových dôvodov zaradené aj niektoré ďalšie používané periférie, napríklad digitálne fotoaparáty a videokamery, alebo zvukové karty. Informácie o nich čitateľ nájde v ďalšej literatúre (viď zoznam literatúry).

Záver

Zoznam literatúry

...

Obsah