Návrh počítačových systémů
INP

Studijní opora

Řešené a neřešené příklady

Lukáš Sekanina verze 1.2006

Tento učební text vznikl za podpory projektu „Zvýšení konkurenceschopnosti IT odborníků – absolventů pro Evropský trh práce“, reg. č. CZ.04.1.03/3.2.15.1/0003. Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

Použité piktogramy

Počítačové cvičení, příklad	Správné řešení
Otázka, příklad k řešení	Obtížná část
Příklad	Důležitá část
Slovo tutora, komentář	Cíl
Potřebný čas pro studium, doplněno číslicí přes hodiny	Definice
Reference	Zajímavé místo (levá, pravá varianta)
Souhrn	Rozšiřující látka, informace, znalosti. Nejsou předmětem zkoušky.

1 Úvod a motivace

Tento text je součástí studijní opory pro kurz Návrh počítačových systémů, který je vyučován ve druhém ročníku bakalářského studijního programu Informační technologie na Fakultě informačních technologií VUT v Brně.

Cílem textu je poskytnout studentům sadu příkladů, které jim pomohou ověřit a prohloubit znalosti získané v průběhu kurzu.

Text je rozčleněn do kapitol, které obsahují řešené i neřešené příklady týkající se určité problematiky studované v kurzu. Na začátku každé kapitoly je odstavec „Zopakujte si“, který obsahuje stručný seznam základních pojmů a dovedností, které jsou nutné k úspěšnému vyřešení příkladů. Předpokládá se, že student získal tyto znalosti a dovednosti z primárního výkladu látky kurzu (přednáška, učebnice apod.). Řešení neřešených příkladů je často možné najít v textu přednášek nebo v prezentacích používaných na demonstračních cvičeních.

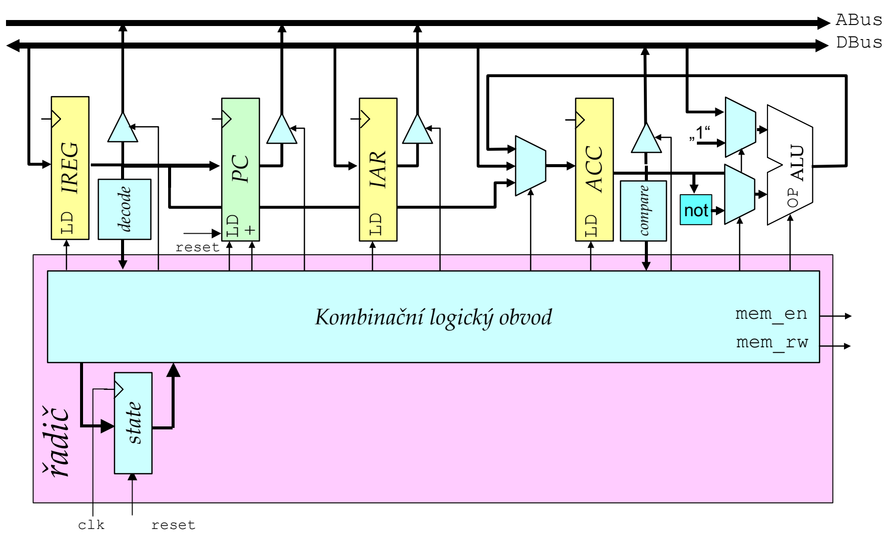
2 Jednoduchý procesor

Zopakujte si: Pojem algoritmu, vztah mezi programovou a obvodovou realizací algoritmu, princip činnosti procesoru, fáze zpracování instrukce, připojení registrů IREG, IAR, střadače a programového čítače v procesoru.

Příklad 2.1 Demonstrujte příklady číslicových obvodů, které jsou schopny realizovat základní výpočetní konstrukce, které znáte z programovacích jazyků.

Příklad 2.2 Popište jednotlivé fáze zpracování instrukce.

Příklad 2.3 Vysvětlete činnost jednotlivých komponent procesoru, který je uveden na obrázku a který byl prezentován na přednášce.



Příklad 2.4 Zapište posloupnost signálů, které musí řadič aktivovat a deaktivovat, pokud se má vykonat instrukce *ADD addr. 25h* (sečti obsah střadače s obsahem paměťové buňky s adresou 25h).

Příklad 2.5 Zapište posloupnost signálů, které musí řadič aktivovat a deaktivovat, pokud se má vykonat instrukce *ILOAD addr. 25h* (nepřímé čtení - načti do střadače obsah paměťové buňky, která je určena adresou definovanou jako obsah paměťové buňky s adresou 25h).

Příklad 2.6 Navrhněte řadič uvedeného procesoru. Uvažujte Mealyho automat.

Příklad 2.7 Popište hlavní slabá místa (s ohledem na výkonnost) uvedeného procesoru.

3 Zobrazení čísel v počítači

Zopakujte si: Převody mezi soustavami. Zobrazení čísel v přímém kódu, inverzním kódu, doplňkovém kódu, v kódu „sudý posun“ a v kódu „lichý posun“. Rozdíl mezi přenosem a přetečením. Zobrazení reálných čísel v pevné a pohyblivé řádové čárce. Chyby při zobrazování čísel v počítači.

Příklad 3.1 Zobrazte čísla –7 a +7 na 8 bitech v přímém kódu, inverzním kódu, doplňkovém kódu, v kódu „sudý posun“ a v kódu „lichý posun“. Vysvětlete, proč se nejčastěji používá doplňkový kód, když chceme pracovat s čísly se znaménkem.

Příklad 3.2 Vysvětlete rozdíl mezi přetečením (overflow) a přenosem (carry) při sčítání. Demonstrujte tyto jevy při sčítání binárních čísel 11101110 a 11100100.

Příklad 3.3 Vyjádřete binárně v pevné řádové čárce desetinné číslo 140,15625. Celá část je uložena v přímém kódu na 10 bitech. Desetinná část je uložena v přímém kódu na 8 bitech. V jakém intervalu a s jakou přesností je možné ukládat čísla v uvažovaném kódu?

Příklad 3.4 Uvažujte číslo 1 1000 0001 0100 0000 0000 0000 0000 0000. Číslo je uloženo v pohyblivé řádové čárce v kódu Intel (krátká reálná čísla). Určete jeho dekadickou hodnotu.

Dekadická hodnota $N = M*B^E$ (M je mantisa, B = 2 je základ, E je exponent).
V uvedeném kódu určuje první bit znaménko, následuje 8b exponent v kódu „lichý posun“ a 24b mantisa v přímém kódu.

$N = (-1)^S\ (2^{E-BIAS})\ (1+mantisa)$

$N = (-1)\ \ (2^{129-127})\ (1.25) = -5$

Příklad 3.5 Sledujte přenos do nejvyššího bitu (Cp) výsledku a z nejvyššího bitu (Cv) výsledku při sčítání následujících osmibitových čísel: (a) 11110100 + 00001001, (b) 10010010 + 10101000, (c) 10110101 + 01001111, (d) 01000101 + 01100111. Kdy došlo k přetečení? Pomocí které logické funkce F(Cp, Cv) je možné automaticky detekovat přetečení? Uvažujte doplňkový kód na 8 bitech.

Příklad 3.6 Doplňte následující tabulku:

Binárně (8 bitů)	Hexa	Dekadicky -bez znaménka	Dekadicky - dvojkový doplněk	Dekadicky - přímý kód	Dekadicky - sudý posun	Dekadicky - lichý posun
00000000		0	0	0	-127	-128
00000001		1	1	1		
00000010		2	2	2		
00000011		3				
00000100		4				
....	....	....	....	....		
		125	125	125		
		126				
		127				
10000000	80	128	-128	-0		
		129		-1		
....	....	....	....	....		
		253				
		254				
11111111	FF	255	-1	-127		
Interval		0..255	-128..127	-127..127		

Příklad 3.7

Vysvětlete rozdíl mezi následujícími chybami při zobrazování čísel: chyba měření, chyba stupnice, chyba zanedbáním a zaokrouhlováním. Jak se určí absolutní chyba při zobrazení čísel v pohyblivé řádové čárce?

Příklad 3.8 Uvažujte osmibitové číslo 11001111. Navrhněte a zdůvodněte deset různých („rozumných“) interpretací tohoto čísla.

Příklad 3.9 Vysvětlete princip sčítání čísel v pohyblivé řádové čárce. Je toto sčítání asociativní?

x+y

4 Výkonnost

Zopakujte si: Definice výkonnosti. Poměr výkon/cena. Měření výkonnosti na základě počtu provedených instrukcí, frekvence procesoru, středního počtu cyklů na instrukci. MIPS a MFLOPS. Amdahlův zákon. Koncept relativní výkonnosti.
Poznámka: V této kapitole jsou použity následující symboly: P značí výkon, T_e je doba výpočtu, I je počet instrukcí a T_C je perioda hodinového signálu.

Příklad 4.1 Na počítačích A a B jsme spustili program R. Naměřili jsme doby výpočtu t_A=10s, t_B=15s. O kolik je A výkonnější než B (v %)? Bude vypočtené procento platit i pro jiný program Q spuštěný na počítačích A a B?

Z definičního vztahu pro výkonnost určíme $P_A/P_B = t_B/t_A = 15/10 = 1,5$
Tudíž počítač A je o 50 % výkonnější než B.
Tento výsledek platí jen pro daný konkrétní program R.

Příklad 4.2

CPUa potřebuje pro podmíněný skok 2 instrukce: CMP (porovnej) a BC (skok). CPUb potřebuje pro podmíněný skok 1 instrukci: BC (skok). V obou případech zabere vykonání instrukce BC 2 cykly, ostatní instrukce pouze 1 cyklus. Pro CPUa tvoří instrukce BC 20% všech instrukcí v programech. Který CPU je rychlejší, pokud (a) $T_{cb} = 1.25 \cdot T_{ca}$, (b) $T_{cb} = 1.1 \cdot T_{ca}$. ($T_c$ je perioda hodinového signálu.)

Určíme střední počet taktů na provedení instrukce pro CPUa: $CPI_{a,stř} = 2 \cdot 0,2 + 1 \cdot 0,8 = 1,2$ (ve 20% případů se provádí 2 takty (skok), jinak pouze jeden takt).
Určíme střední počet taktů na provedení instrukce pro CPUb: $CPI_{b,stř} = 2 \cdot 0.25 + 1 \cdot 0.75 = 1.25$ (Pozor! Zde máme kratší programy oproti CPUa – není v nich ve 20% případů instrukce CMP. Proto četnost CMP relativně vzroste na 25%. Uvažte, například, typický program o 100 instrukcích. U CPUa tam bude 20 x BC, 20 x CMP a 60 x ostatní instrukce. U CPUb tam bude jen 20x BC a 60x ostatní instrukce).
Doba výpočtu CPUa: $t_{ea} = I \cdot CPI_{a,stř} \cdot T_{ca} = 1.2 \cdot I \cdot T_{ca}$
Určíme dobu výpočtu CPUb pro oba požadované případy.
(a) $t_{eb} = 0.8 \cdot I \cdot CPI_{b,stř} \cdot T_{cb} = 0.8 \cdot I \cdot 1.25 \cdot 1.25 \cdot T_{ca} = 1.25 \cdot I \cdot T_{ca}$ (koeficient 0.8 představuje uvažovanou korekci)
$t_{eb} / t_{ea} = 1.25/1.2 = 1.04$ (CPUa je rychlejší)
(b) $t_{eb} = 0.8 \cdot I \cdot CPI_{b,stř} \cdot T_{cb} = 0.8 \cdot I \cdot 1.25 \cdot 1.1 T_{ca} = 1.1 \cdot I \cdot T_{ca}$
$t_{eb} / t_{ea} = 1.1/1.2 = 0.9$ (CPUb je rychlejší)

Příklad 4.3

Předpokládejme, že navrhujeme nový procesor. V rámci předběžných testů jsme naměřili: střední CPI operací pracujících v pohyblivé řádové čárce (FP) je 4, četnost FP operací je 25%, střední CPI ostatních operací je 1.33, frekvence použití instrukce FPSQR je 2% (odmocnina), CPI_FPSQR = 20. Existují dvě alternativy zlepšení návrhu: (a) redukovat CPI_FPSQR (z 20) na 2 nebo (b) redukovat střední CPI všech FP operací (ze 4) na 2. Srovnejte tyto dvě alternativy pomocí rovnice o výkonnosti CPU.

Rovnice o výkonnosti říká, že $te = I \cdot CPIstř \cdot T_c$. Uvážíme-li, že počet instrukcí a Tc se v naší úloze nemění, stačí analyzovat pouze vliv CPI.
S využitím informace, že CPIFP, stř = 4, vypočteme CPIFP, stř, bez FPSQR:
$CPI_{FP,\ stř} = 4$, což můžeme rozepsat jako $4 = 0.98 \cdot n + 0.02 \cdot 20 \Rightarrow n = 3.67 = CPI_{FP,\ stř,\ bez\ FPSQR}$
Vypočteme původní $CPI_{pův,\ stř} = 4 \cdot 0.25 + 1.33 \cdot 0.75 = 2$
Ohodnotíme alternativy:
(a) $CPI_{a,\ stř} = (0.98 \cdot n + 0.02 \cdot 2) \cdot 0.25 + 1.33 \cdot 0.75 = (3.6 + 0.04) \cdot 0.25 + 0.9975 = 1.91$
(b) $CPI_{b,\ stř} = 2 \cdot 0.25 + 1.33 \cdot 0.75 = 1.5$ (toto je lepší alternativa)
Zrychlení: $t_{e,pův}/t_{e,b} = CPI_{pův,\ stř} / CPI_{b,\ stř} = 1.33$

Příklad 4.4

Optimalizující kompilátor odstraní 50% instrukcí ALU, ostatní instrukce ponechá. T_c=2ns. Vypočtěte $CPI_{pův}$, CPI_{opt}, $P_{MIPSpův}$, $P_{MIPSopt}$. Souhlasí poměr P_{MIPS} a t_e? Co je tu zvláštní? V tabulce jsou uvedeny četnosti a CPI jednotlivých instrukcí. V nejpravějším sloupci jsou potom četnosti přepočítány pro optimalizovaný případ.

Instrukce	Četnost (%)	CPI	Četnost po optimalizaci (%)
ALU	43	1	21,5
Load	21	2	21
Store	12	2	12
JMP	24	2	24
Celkem	100		78,5

$CPI_{pův} = 0.43·1+0.21·2+0.12·2+0.24·2 = 1.57$

$CPI_{opt} = (0.215·1+0.21·2+0.12·2+0.24·2)/0.785 = 1.73$ (změnil se počet instrukcí, proto přepočet s koeficientem 0.785)

$P_{MIPS,pův}= f_c/(CPI_{pův}·10^6) = 10^3/(CPI_{pův}·T_c) = 318.4$ [MIPS]

$P_{MIPS,opt}= f_c/(CPI_{opt} ·10^6) = 10^3/(CPI_{opt}·T_c) = 289.0$ [MIPS]

$t_{e,pův} = I·CPI_{pův}·T_c =3.14·10^{-9}·I$ [s]

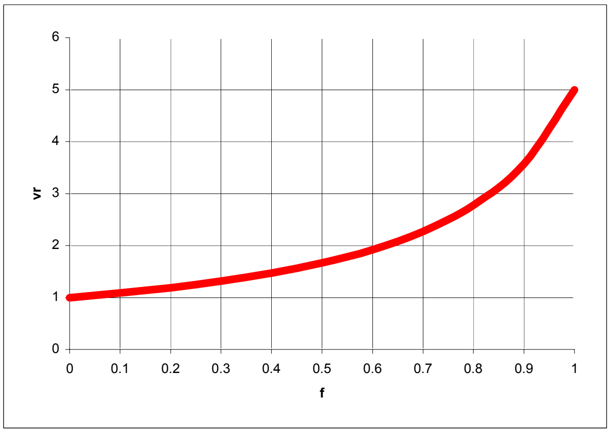
$t_{e,opt} = 0.785·I·CPI_{opt}·T_c = 2.7·10^{-9}·I$ [s]

Po optimalizaci se nám přirozeně zkrátila doba výpočtu (kompilátor odstranil některé instrukce). Ale snížil se i ukazatel výkonnosti – je to dáno metodikou výpočtu. P_{MIPS} se chápe jako ukazatel pro celý systém včetně programu. Směrodatný je poměr dob výpočtu.

Příklad 4.5

Paměť cache je 5x rychlejší než hlavní paměť a využívá se v 90% výpočetního času. Jaké se dosáhne zrychlení použitím této cache? Nakreslete závislost graficky.

Použijeme Amdahlův zákon: f = 0.9 reprezentuje část, kterou můžeme urychlit. r = 5 určuje, kolikrát umíme urychlit část f. Zrychlení dle Amdahlova zákona je dáno:
$V_r = 1/((1-f)+f/r) = 1/((1-0.9) + 0.9/5) = 3.57$



Celkové zrychlení v závislosti na f (f je část, kterou umíme zrychlit pětkrát).

Příklad 4.6

CPU je využit z 50% doby výpočtu, zbytek doby výpočtu čeká na pomalé V/V operace. Cena CPU je 1/3 ceny počítače. Je výhodné koupit 5x rychlejší CPU za 5x vyšší cenu?

f = 0.5 (část, kterou umíme urychlit)
r = 5 (urychlení části f)
Celkové zrychlení: V_r = 1/((1-f)+f/r) = 1/((1-0.5) + 0.5/5) = 1.66
Násobek ceny počítače s novým CPU: 2/3+5*1/3=2.33 (pozor, 2/3 ceny ne nemění!)
Počítač je 2.33x dražší, ale jen 1.66x výkonnější. => Není obvykle výhodné koupit takový CPU.

Příklad 4.7

Vysvětlete koncept relativní výkonnosti. Demonstrujte metodiku založenou na SPEC.

1) Zvolíme referenční počítač.
2) Zvolíme sadu testovacích, tzv. benchmarkových, úloh.
3) Spočítáme relativní doby výpočtu úloh na daném počítači (tj. vzhledem k referenčnímu počítači)
4) Z relativních dob můžeme spočítat např.: aritmetický průměr, vážený průměr, geometrický průměr. Obvykle je používán geometrický průměr (viz sady testovacích úloh SPEC, Whetstone, Dhrystone, …) protože jeho hodnota nezávisí na volbě referenčního počítače.

Ilustrační příklad: Sada benchmarků SPEC89 sestává ze 4 programů pracujících v pevné řádové čárce a z 6 programů pracujících v pohyblivé řádové čárce. Ohodnocení výkonnosti se určí pro každou skupinu dle následujících vztahů (t_r je relativní doba výpočtu vzhledem k referenčnímu počítači):

$$spec_{\mathrm{int}\ 89} = \sqrt[4]{\prod_{i=1}^{4} t_{r_i}}$$

$$spec_{fp\ 89} = \sqrt[6]{\prod_{i=1}^{6} t_{r_i}}$$

Následující tabulka demonstruje způsob porovnání počítačů DEC3100 a SPARC (počítač VAX11/780 je zvolen jako referenční)

Počítač / Úloha	VAX11/780 (s)	DEC3100 (s)	DEC3100 (rel. k VAX)	SPARC (s)	SPARC (rel. k VAX)
GCC	1482	145	10,22	138,9	10,67
Espresso	2266	194	11,68	254	8,92
Li	6206	480	12,93	689,5	9,08
Eqntott	1101	99	11,12	113,5	9,7
Spice	23951	2500	9,58	2875,5	8,33
Doduc	1863	208	8,96	374,1	4,98
NASA7	20093	1646	12,21	2308,2	8,71
Matrix300	4525	749	6,04	409,3	11,06
Fpppp	3038	292	10,4	387,2	7,85
Tomcatv	2649	260	10,19	469,8	5,64
SPEC-FX	Referenční počítač		11,45		9,57
SPEC-FP			9,36		7,49

Dle SPEC89 je DEC3100 výkonnější než SPARC pro úlohy pracující v pevné i pohyblivé řádové čárce.

5 Kódy pro odstranění redundance a zabezpečení informace

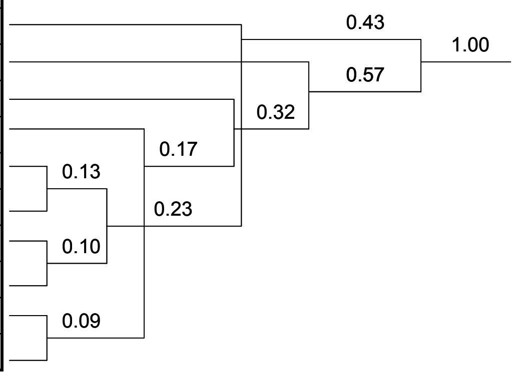
Zopakujte si: Typy kódů pro odstranění redundance a zabezpečení informace. Huffmanův kód – konstrukce a vlastnosti. Hammingova vzdálenost. Parita, ztrojení bitu, Hammingův kód, rozšířený Hammingův kód (včetně obvodové realizace). Kód zbytkových tříd.

Příklad 5.1

Pomocí Huffmanova kódování zakódujte znaky 0-9 vyskytující se s uvedenou četností: 0 (0.2), 1 (0.25), 2 (0.15), 3 (0.08), 4 (0.07), 5 (0.06), 6 (0.05), 7 (0.05), 8 (0.05), 9 (0.04).

Vytvoříme strom tak, že budeme spojovat uzlem ty hodnoty, pro které získáme nejmenší součet pravděpodobností. Do uzlů postupně vepisujeme součty (obr. A).

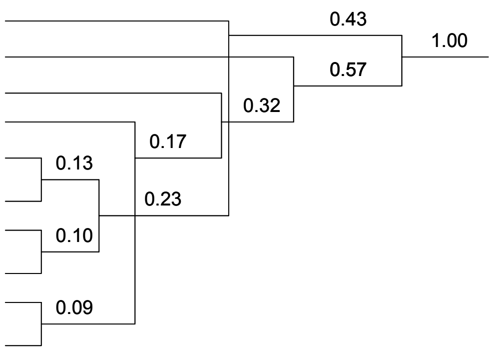
znak	četnost
0	0.2
1	0.25
2	0.15
3	0.08
4	0.07
5	0.06
6	0.05
7	0.05
8	0.05
9	0.04



Obrázek A

Systematicky (od kořene) ohodnotíme 0 (horní) a 1 (dolní) jednotlivé podstromy. Průchod stromem od kořene k listu udává zakódování znaků (obr. B).

znak	četnost
0	0.2
1	0.25
2	0.15
3	0.08
4	0.07
5	0.06
6	0.05
7	0.05
8	0.05
9	0.04



Obrázek B

Pokud je číslování systematické, je kód prefixový, tj. začátek každé značky je unikátní a značky jsou jednoznačně dekódovatelné.

Příklad 5.2 Uvažujte Huffmanův kód sestrojený v předchozím příkladu.

(a) Dekódujte následující posloupnost 20 bitů 00100010111001100000. O kolik bitů je zpráva kratší než kdybychom kódovali každou hodnotu 4 bity (tj. nepoužili Huffmanův kód)?

(b) Jaké dekadické číslo představuje v uvažovaném kódu posloupnost dvaceti jedniček? O kolik bitů je zpráva kratší než kdybychom kódovali každou hodnotu 4 bity (tj. nepoužili Huffmanův kód)?

Příklad 5.3 Pro uvedený Huffmanův kód vypočtěte průměrnou délku značky, střední délku značky, teoretickou optimální délku značky a redundanci kódu.

kód	znak	Li	fi
00	0	2	0.2
10	1	2	0.25
110	2	3	0.15
1110	3	4	0.08
0100	4	4	0.07
0101	5	4	0.06
0110	6	4	0.05
0111	7	4	0.05
11110	8	5	0.05
11111	9	5	0.04

Nejdříve pro každou kódovou značku určíme její délku Li (tj. počet symbolů 0 a 1, viz tabulka).

průměrná délka značky: $Lp = \Sigma Li/n = 3.7$ $(i=0\ldots9,\ n = 10)$

střední délka značky: $Lstř = \Sigma Li*fi = 2*0.2+\ldots+5*0.04 = 3.04$

teoretická optimální délka značky: $Lopt = -\Sigma\ Li* \log_2 fi = 3.01$

Redundance kódu: $R = (Lstř-Lopt)/Lstř = 0.98\%$

Příklad 5.4 Vysvětlete princip kódu sudá parita. Demonstrujte jeho činnost. Demonstrujte selhání.

Příklad 5.5 Vysvětlete princip SEC kódu „ztrojení bitu“. Demonstrujte jeho činnost. Demonstrujte selhání.

Vzniká ztrojením každého bitu: 0→000, 1→111. Při výskytu jedné chyby v trojici dokáže odvodit původní hodnotu z majority.

Kódové kombinace: 000, 111

Nekódové kombinace: 001, 010, 011, 100, 101, 110

Opravy: (001, 010, 100) →0, (011, 101, 110) →1

Příklad:

Zpráva: 0 1 1 0 1

Zakódováno: 000 111 111 000 111

Při chybě (označené podtržením) 000 111 1$\underline{0}$1 00$\underline{1}$ 111 dojde ke správné opravě na 0 1 1 0 1.

Při jiné chybě 000 100 111 000 111 dojde k chybné opravě na 0 0 1 0 1, tj. kód nedokáže detekovat dvojchybu.

Příklad 5.6

Vysvětlete princip činnosti Hammingova kódu (7,4). Zakódujte v HK(7,4) číslo 1001.

Konstrukce HK(7,4) je uvedena v textu výkladu problematiky (přednášky, skripta). Pro zakódování 1001 musíme vypočítat tzv. kontrolní bity C1, C2 a C4 (I3, I5, I6, I7 jsou informační bity).
I3 = 1, I5 = 0, I6 = 0, I7 = 1 (získáme ze zadání)
C1 = I3 xor I5 xor I7 = 0
C2 = I3 xor I6 xor I7 = 0
C4 = I5 xor I6 xor I7 = 1
Zakódováním 1001 dostaneme 1001100 (pozice jednotlivých bitů v HK(7, 4) je následující: I7, I6, I5, C4, I3, C2, C1).

Příklad 5.7

Navrhněte obvod, který realizuje dekódování a opravu jednochyb na základě syndromu v HK(7,4). Nápověda: použijte dekodér a log. členy XOR.

Příklad 5.8

Demonstrujte činnost HK(7,4) v situaci, kdy byla data 1001 zakódována na 1001100 a příjemce přečetl (a) 1001100 a (b) 1001000.

(a) Přijaté slovo: 1001100. Vypočteme syndrom S.
$S_1 = C'_1$ xor I'_3 xor I'_5 xor I'_7= 0 xor 1 xor 0 xor 1 = 0
$S_2 = C'_2$ xor I'_3 xor I'_6 xor I'_7= 0 xor 1 xor 0 xor 1 = 0
$S_4 = C'_4$ xor I'_5 xor I'_6 xor I'_7= 1 xor 0 xor 0 xor 1 = 0
S = 000 => nedošlo k chybě
(b) Přijaté slovo: 1001000
$S_1 = C'_1$ xor I'_3 xor I'_5 xor I'_7= 0 xor 0 xor 0 xor 1 = 1
$S_2 = C'_2$ xor I'_3 xor I'_6 xor I'_7= 0 xor 0 xor 0 xor 1 = 1
$S_4 = C'_4$ xor I'_5 xor I'_6 xor I'_7= 1 xor 0 xor 0 xor 1 = 0
S = 011 = 3 => došlo k chybě na pozici 3, po opravě 1001100

Příklad 5.9

Vysvětlete princip rozšířeného HK(8, 4). Co získáme rozšířením? Jak se definuje syndrom chyby? Co je možné detekovat a co opravit?

Příklad 5.10

Obecně vysvětlete jak se tvoří kód zbytkových tříd.

Příklad 5.11

S využitím modulů (2, 3, 5) zakódujte čísla 0 - 29 v kódu zbytkových tříd. Demonstrujte sčítání, odčítání a násobení v tomto kódu. Proč nefunguje dělení? Proč používáme kód zbytkových tříd pro realizaci aritmetických operací?

Příklad 5.12

Navrhněte sčítačku pracující v kódu zbytkových tříd (2, 3, 5).

Příklad 5.13

Proč se Huffmanův kód používá při návrhu zakódování instrukcí?

x+y

6 Sčítačky

Zopakujte si: Poloviční sčítačka, úplná sčítačka, sčítačka s postupným přenosem, sériová sčítačka. Sčítačka s uchováním přenosu, sčítačka s CLA. Výpočet zpoždění a počtu hradel nutných pro realizaci. Aritmeticko-logická jednotka (ALU).

Příklad 6.1 Nakreslete značku a vnitřní strukturu poloviční sčítačky.

Příklad 6.2 Nakreslete značku a vnitřní strukturu úplné sčítačky (použijte logické členy XOR). Jaké má sčítačka zpoždění?

Příklad 6.3 Nakreslete vnitřní strukturu n-bitové sčítačky s postupným přenosem. Odvoďte počet hradel potřebných pro konstrukci a zpoždění.

Příklad 6.4 Definujte rozšířenou sčítačku (se signály G – generate a P – propagate). Nakreslete schéma zapojení a odvoďte zpoždění.

Příklad 6.5 Nakreslete schéma zapojení 4b sčítačky s 4b generátorem CLA. Odvoďte zpoždění obvodu.

Příklad 6.6 Nakreslete schéma zapojení 16b sčítačky využívající 4b generátory CLA zapojené do stromu. Odvoďte zpoždění obvodu. Proč nepoužíváme 16b generátor CLA?

Příklad 6.7 Nakreslete vnitřní schéma 4b ALU, která provádí operace A+B, A-B, A and B a A xor B.

x+y

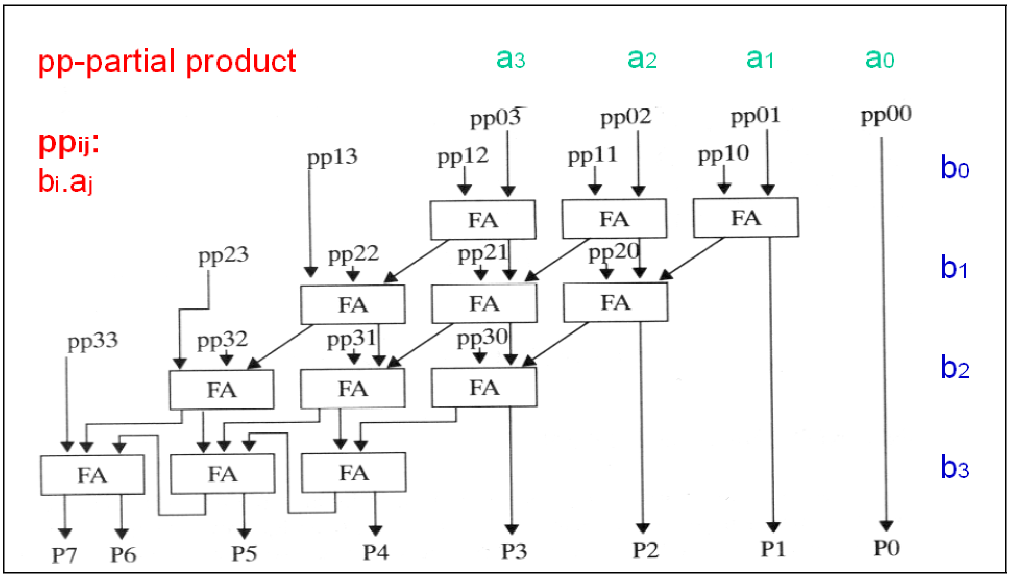
7 Násobičky

Zopakujte si: Násobení čísel bez znaménka a se znaménkem. Sekvenční a kombinační násobičky. Boothovo překódování (odvození tabulky). Obvody kombinační násobičky. Vroubení znaménka .Wallaceův strom.

Příklad 7.1 Vynásobte 4b čísla bez znaménka: 1011 x 1101. Kolik bitů potřebujeme pro uložení výsledku?

Příklad 7.2 Nakreslete schéma 4b násobičky s uchováním přenosu. Odvoďte zpoždění.

Násobička bude sestavena z 12 sčítaček (viz obrázek). Kromě posledního řádku není přenos není šířen vodorovně, ale přímo do následujícího součtu. V posledním řádku je realizována sčítačka s postupným přenosem. Jednotlivé součiny (pp) jsou realizovány hradlem AND.
Předpokládejme násobení N-bitů x M-bitů:
Plocha: Potřebujeme (N-1)*M sčítaček (některé mohou být poloviční).
Zpoždění: předpokládejme, že zpoždění jedné úplné sčítačky odpovídá zpoždění 2 hradel. Do celkového zpoždění musíme zahrnout tato zpoždění:
1 x hradlo AND (první částečný součin)
M-1 řádků sčítaček
N-1 sčítaček v posledním řádku
Celkem 2*(M-1 + N-1) + 1 [zpoždění hradla]
V našem případe: 2(4-1 + 4-1) + 1 = 13 x zpoždění hradla



Obrázek násobičky s uchováním přenosu

Příklad 7.3

Vynásobte čísla se znaménkem: -4 x -14 na 4b (+1b na znaménko).

```
(-4)                1111111100
(-14)               1111110010
                         00000
                  1111111100
                       00000
                      00000
            1111111100
           1111111100
          1111111100
         1111111100
        1111111100
       1111111100
                    0000111000b      (+56d)
```

Příklad 7.4

S využitím Boothova překódování s radixem 2 (tj. po jednom bitu) vynásobte –12 x –7 (4b + 1b znaménkový). Jak se zredukuje počet sčítání, která musíme provést?

7 = 00111, -7 = 11001, -12 = 10100

Při překódování násobitele –7, tj. 11001, si na nejpravějším místě představíme nulu, tj. získáme 11001<u>0</u>. Dále násobitele překódujeme podle tabulky s relativními číslicemi, tj. překódovaný násobitel bude: 0-101-1.

Tabulka relativních číslic pro Boothovo překódování s radixem 2.

Bit násobitele	Bit vpravo	Relativní číslo
0	0	0
0	1	1
1	0	-1
1	1	0

Vlastní násobení: Celkem tedy musíme provést pouze tři součty.

```
      10100 (-12)
    0-101-1 (-7)
-----------------
      01100 (+12)
   11110100 (-12)
          0 (0)
     001100 (+12)
-----------------
 1001010100 (+84)
```

Příklad 7.5

S využitím Boothova překódování s radixem 4 (tj. po dvou bitech) vynásobte 13 x (–6).

Překódujeme násobitele. Musíme přejít na 6 bitů, protože budeme potřebovat dvojnásobky, které není možné zobrazit na původním počtu bitů.

```
13 x -6, 13=001101, 6=000110, -13=110011,
-26=100110  (musíme přejít na 6 bitů!!)
 -6=111010 => 1110100 => 0-1-2 (překódovaný násobitel)
```

Při násobení nezapomeneme šířit znaménko a posunovat každý další sčítanec o 2 bity doleva.

```
             01101 (13)
             0-1-2 (-6)
---------------------------
         111100110 (-26)
         1110011   (-13)  posun o 2bity
---------------------------
         110110010 (-78)
```

Příklad 7.6

Vysvětlete princip vroubení znaménka pro Boothovo překódování s radixem 4 na násobení čísel 13 x (-6).

Příklad 7.7

Odvoďte tabulku, podle které se překóduje násobitel pro Boothovo překódování s radixem 8.

Příklad 7.8

Popište obvody, ze kterých se skládá kombinační násobička pracující s Boothovým překódováním. Nakreslete schéma takové násobičky.

Příklad 7.9

Porovnejte kombinační a sekvenční násobičky z pohledu zpoždění a plochy, kterou zabírají na čipu.

x+y

8 Dělení a iterační algoritmy

Zopakujte si: Úplná odčítačka. Sekvenční a kombinační děličky. Algoritmus dělení s restaurací a bez restaurace nezáporného zbytku. Algoritmus SRT. Iterační algoritmy dělení (Newton, Goldschmidt). Rychlost konvergence. Výpočet odmocniny a dalších funkcí. CORDIC.

Příklad 8.1

Vysvětlete základní princip sekvenční realizace dělení v HW.

Hledáme: podíl a zbytek dělení D/d
Platí: D = Q .d + R
D – dělenec (2n bitů), d – dělitel (n bitů), Q – podíl (n bitů), R – zbytek (n bitů)

Princip: v každém kroku (i) se pokoušíme odečíst od průběžného zbytku Ri posunutý dělitel ($2^{-i}d$ – posuv o i bitů vpravo)

Příklad 8.2 Vydělte D=100110 / d=101, n = 3.

Příklad 8.3 Vydělte 30:7 algoritmem s restaurací nezáporného zbytku

Snažíme se odečíst dělitele od průběžného zbytku. Protože pracujeme v doplňkovém kódu, tak vlastně pořád sčítáme. Pokud získáme záporné číslo, poznamenáme si do výsledku 0 a provedeme korekci přičtením dělitele. Pokud získáme kladné číslo, zapíšeme do výsledku 1. Posuneme průběžný zbytek o jeden bit doleva. Postup opakujeme dokud je možné provádět odčítání. C0 – C4 značí bity výsledku.

```
30=00011110, 7=0111, -7=1001
 00011110
+1001         -d
 10101110     <0, => c4=0
+0111                +d (korekce)
 00011110     posuv <-
 0011110x
+1001         -d
 1100110x     <0 => c3=0
+0111         +d (korekce)
 0011110x     posuv
 011110xx
+1001         -d
 000010xx     >0 => c2=1
 00010xxx     posuv
+1001         -d
 10100xxx     <0 => c1=0
+0111         +d (korekce)
 00010xxx     posuv
 0010xxxx
+1001         -d
 1011xxxx     <0 => c0=0
+0111         +d (korekce)
 0010xxxx     zbytek 2
```

Příklad 8.4 Vydělte 30:7 algoritmem bez restaurace nezáporného zbytku

Oproti předchozímu příkladu příkladu nepřičítáme v průběhu výpočtu dělitele pokud je průběžný zbytek záporný.

```
30=00011110, 7=0111, -7=1001
 00011110
+1001         -d
 10101110     <0 => c4 = 0
 0101110x     posuv <-
+0111         +d
 1100110x     <0 => c3 = 0
 100110xx     posuv
+0111         +d
 000010xx     >0 => c2 = 1
 00010xxx     posuv
+1001         -d
 10100xxx     <0 => c1 = 0
 0100xxxx     posuv
+0111         +d
 1011xxxx     <0 => c0 = 0
+0111         +d (korekce na kladný zbytek)
 0010xxxx     zbytek 2
```

Příklad 8.5 Vydělte -49:7 algoritmem SRT

Postupujeme podobně jako v předchozích příkladech. Následující akci však vždy provádíme na základě nejvyšších tří bitů průběžného zbytku dle následující tabulky.

Ri	d>0 bit podílu	d>0 operace	d<0 bit podílu	d<0 operace
000 111	0	posuv vlevo	0	posuv vlevo
001 010 011	1	-d, posuv vlevo	-1	+d, posuv vlevo
101 110 100	-1	+d, posuv vlevo	1	-d, posuv vlevo

49=00110001, 7=0111, -7=1001
-49=11001111 (d>0)

```
-1      11001111
        0111    +d
        00111111       <-
+1      0111111x
        1001    -d
        0000111x
0       000111xx       <-
+1      00111xxx       <-
        1001    -d
        11001xxx       <-
-1      1001xxxx
        0111    +d
        0000xxxx
        zbytek 0
        Q = -1 1 0 1 -1 => -16+8+0+2-1 = -7
```

Příklad 8.6 Vydělte 53:6 algoritmem SRT.

Příklad 8.7 Vydělte 53 : (-6) algoritmem SRT.

53=00110101, 6=0110, -6=1010

```
-1      00110101
        1010    +d
        11010101
1       1010101x       <-
        0110    -d
        0000101x
0       000101xx       <-
-1      00101xxx       <-
        1010    +d
        11001xxx       <-
1       1001xxxx
        0110    -d
        1111xxxx       záporný zbytek
        0110    +d (korekce)
        0101    zbytek 5

        Q = -1 1 0 -1 1 =
        -16+8+0-2+1=-9
        po korekci -9+1=-8
```

Příklad 8.8 Vydělte 51 : 7 algoritmem SRT. Je výsledek správný? Zdůvodněte.

Příklad 8.9 Odvoďte vztah pro Newtonův iterační algoritmus pro dělení

Příklad 8.10 Vydělte 1/20 pomocí Newtonova iteračního algoritmu

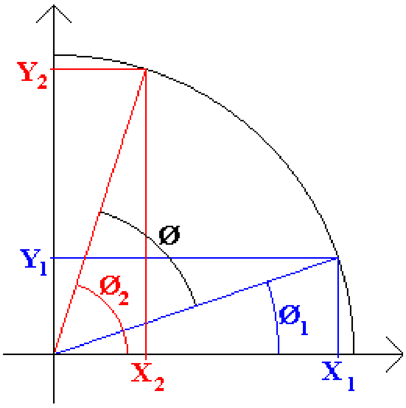
b = 10100. Posuneme desetinnou čárku o 4b vlevo, abychom se dostali do intervalu <1,2>, tj. b = 1.0100.
Zvolíme $x_0 = 1$
$x_1 = x_0 (2 – bx_0) = 1(10 – 1.01 \text{ x } 1) = \underline{0.11}$
$x_2 = x_1 (2 – bx_1) = 0.11(10 – 1.01 \text{ x } 0.11) = 0.11(10 – 0.1111) = 0.11 \text{ x } 1.0001 = \underline{0.110011}$
$x_3 = \underline{0.11001100110011}$
atd.
Posuneme desetinnou čárku zpět, tj. o 4b vlevo.
Výsledek po 3 iteračních krocích: 1/20 = 0.000011001100110011b = 0.051952362d

Příklad 8.11 Pro Newtonův iterační algoritmus dělení ukažte, že se relativní chyba s každým krokem sníží na polovinu, tj. že se počet správných (přesných) bitů p s každým iteračním krokem zdvojnásobuje.

Určíme absolutní chybu: $x_i – 1/b$.
Dále nechť $A = (x_i – 1/b)/(1/b) = 2^{-p}$ je relativní chyba *i.* kroku.
Hledáme tvar pro krok *i*+1, tj. pro x_{i+1}
Vyjádříme $x_i = 1/b\ (2^{-p}) + 1/b$ a dosadíme do $x_{i+1} = x_i (2 – bx_i)$, kde dostáváme
$x_{i+1} = 1/b\ (2^{-p}) + 1/b\ (2 – b\ (1/b\ (2^{-p}) + 1/b)) =$
$1/b – 1/b(2^{-2p})$
což upravíme na tvar pro relativní chybu pro krok *i*+1:
$B = (x_{i+1} – 1/b)/(1/b) = 2^{-2p}$
Porovnáním B/A dostaneme požadovanou přesnost, tj. $B/A = 2^{-2p} / 2^{-p} = (1/4)/(1/2) = 2/4 = 1/2$
Ukázali jsme, že se chyba s každým provedeným krokem sníží na polovinu.

Příklad 8.12 Vysvětlete princip algoritmu Cordic na úloze výpočtu funkce sinus a kosinus.

Základní myšlenka algoritmu Cordic: Být schopen počítat většinu matematických funkcí vyčíslováním funkce ve tvaru $a \pm b.2^{-i}$, tj. pomocí obvodově nenáročných součtů, rozdílů, posunů a vyhledáním v lookup tabulce. Vyjdeme z následujícího obrázku:



Je patrné, že $[X_2, Y_2]$ můžeme vyjádřit pomocí $[X_1, Y_1]$ takto:
$X_2 = X_1 * \cos(\varnothing) - Y_1 * \sin(\varnothing)$
$Y_2 = X_1 * \sin(\varnothing) + Y_1 * \cos(\varnothing)$
V algoritmu Cordic je důležitý úhel $\varnothing_2$ – jeho sinus a kosinus chceme vypočítat. Počáteční úhel $\varnothing_1$ je nastaven na nějakou konvenční hodnotu, např. 0. Přechod od $\varnothing_1$ k $\varnothing_2$ se provádí v krocích. Pokud vhodně zvolíme tyto kroky, stačí k výpočtu pouze sčítání a rotace. Předchozí vztahy mohou být přepsány následovně:

$X_2 = \cos(\varnothing) * [X_1 - Y_1 * \tan(\varnothing)]$
$Y_2 = \cos(\varnothing) * [X_1 * \tan(\varnothing) + Y_1]$

Hodnoty pro $\varnothing$ se volí tak, že $\tan(\varnothing)$ nabývá hodnot 1/(mocina čísla 2):

$\tan(\varnothing_{21}) = 1/1$	$\varnothing_{21} = 45°$	$\cos(\varnothing_{21}) = 0.707107$
$\tan(\varnothing_{32}) = 1/2$	$\varnothing_{32} = 26.5650°$	$\cos(\varnothing_{32}) = 0.894427$
$\tan(\varnothing_{43}) = 1/4$	$\varnothing_{43} = 14.0362°$	$\cos(\varnothing_{43}) = 0.970142$
$\tan(\varnothing_{54}) = 1/8$	$\varnothing_{54} = 7.12502°$	$\cos(\varnothing_{54}) = 0.992278$
$\tan(\varnothing_{65}) = 1/16$	$\varnothing_{65} = 3.57633°$	$\cos(\varnothing_{65}) = 0.998053$
$\tan(\varnothing_{76}) = 1/32$	$\varnothing_{76} = 1.78991°$	$\cos(\varnothing_{76}) = 0.999512$
$\tan(\varnothing_{87}) = 1/64$	$\varnothing_{87} = 0.895174°$	$\cos(\varnothing_{87}) = 0.999878$
$\tan(\varnothing_{98}) = 1/128$	$\varnothing_{98} = 0.447614°$	$\cos(\varnothing_{98}) = 0.999969$

To nám dovoluje nahradit násobení $\tan(\varnothing)$ rychlou operací posun vpravo. Je však ještě třeba vypořádat se s $\cos(\varnothing)$. Ukažme si několik po sobě jdoucích iterací:

První iterace (z [X1, Y1] do [X2, Y2]) znamená otočení o úhel $\varnothing_{21}$
$X_2 = \cos(\varnothing_{21}) * (X_1 - Y_1 * \tan(\varnothing_{21}))$
$Y_2 = \cos(\varnothing_{21}) * (X_1 * \tan(\varnothing_{21}) + Y_1)$

Druhá iterace (z [X2, Y2] do [X3, Y3]) znamená otočení o úhel $\varnothing_{32}$
$X_3 = \cos(\varnothing_{32}) * (X_2 - Y_2 * \tan(\varnothing_{32}))$
$Y_3 = \cos(\varnothing_{32}) * (X_2 * \tan(\varnothing_{32}) + Y_2)$

Dosazením získáme:
$X_3 = \cos(\varnothing_{32}) * \{ \cos(\varnothing_{21}) * [X_1 - Y_1 * \tan(\varnothing_{21})] - \cos(\varnothing_{21}) * [X_1 * \tan(\varnothing_{21}) + Y_1] * \tan(\varnothing_{32}) \} = \cos(\varnothing_{32}) * \cos(\varnothing_{21}) * \{ [X_1 - Y_1 * \tan(\varnothing_{21})] - [X_1 * \tan(\varnothing_{21}) + Y_1] * \tan(\varnothing_{32}) \}$

Je patrné, že kosiny se před závorkami násobí, tj.

$\cos(Ø_{21}) * \cos(Ø_{32}) * \cos(Ø_{43}) ... * \cos(Ø_{nn})$

Vyjádřením hodnot Ø jako inverzních hodnot tangenty dostaneme řadu, ze které potom hodnotu:

$$\prod_{N=0}^{\infty} \frac{2^N}{\sqrt{1 + 2^{2N}}} = 0.607253$$

Tato hodnota se nazývá *agregační konstanta*. Ve výpočtech můžeme cos(Ø) ignorovat a jednoduše násobit agregační konstantou před nebo po každé iteraci. Výše uvedené iterační výpočty provádíme dokud nezískáme výsledek s dostatečnou přesností. Po vynásobení agregační konstantou získáme požadovaný sinus a kosinus:

$\sin(Ø_2) = 0.607253 * Y$

$\cos(Ø_2) = 0.607253 * X$

Příklad 8.13 S využitím algoritmu Cordic vypočtěte sin (28.027°) a cos (28.027°).

Položíme: Ø = 0°

cos(Ø) = 1	X = 1
sin(Ø) = 0	Y = 0

Otoč z 0° do 45° ($Ø_{21}$ = 45°)

X' = X - Y / 1	= 1 - 0 / 1	= 1
Y' = X / 1 + Y	= 1 / 1 + 0	= 1

Otoč z 45° do 18.435° ($Ø_{32}$ = -26.565°).

Protože se jedná o záporný úhel a protože je tangens funkce lichá, změníme znaménko u čísel, která se posunují:

X' = X + Y / 2	= 1 + 1 / 2	= 1.5
Y' = -X / 2 + Y	= -1 / 2 + 1	= 0.5

Agregační konstanta se nemění, protože závisí na kosinech. Kosinus je sudá funkce.

Rotuj z 18.435° do 32.471° ($Ø_{43}$ = 14.036°)

X' = X - Y / 4	= 1.5 - 0.5 / 4	= 1.375
Y' = X / 4 + Y	= 1.5 / 4 + 0.5	= 0.875

Otoč z 32.471° do 25.346° ($Ø_{54}$ = -7.125°)

X' = X + Y / 8	= 1.375 + 0.875 / 8	= 1.484375
Y' = -X / 8 + Y	= -1.375 / 8 + 0.875	= 0.703125

Otoč z 25.346° do 28.922° ($Ø_{65}$ = 3.576°)

X' = X - Y / 16	= 1.484375 - 0.703125 / 16	= 1.440429
Y' = X / 16 + Y	= 1.484375 / 16 + 0.703125	= 0.795898

Otoč z 28.922° do 27.132° ($Ø_{76}$ = -1.790°)

X' = X + Y / 32	= 1.440429 + 0.795898 / 32	= 1.465300
Y' = -X / 32 + Y	= -1.440429 / 32 + 0.795898	= 0.750884

Otoč z 27.132° do 28.027° ($Ø_{87}$ = 0.895°)

X' = X - Y / 64	= 1.465300 - 0.750884 / 64	= 1.453567
Y' = X / 64 + Y	= 1.465300 / 64 + 0.750884	= 0.773779

Předpokládejme, že nám přesnost výsledku již postačuje. Tudíž můžeme výpočet ukončit. Následuje násobení agregační konstantou a získání požadovaných hodnot:

sin(28.027°) = 0.607253 * Y = 0.46988

cos(28.027°) = 0.607253 * X = 0.88268

Abychom se vyhnuli tomuto násobení, můžeme počáteční hodnotu X nastavit na 0.607253 místo 1. Znamená to vlastně, že budeme otáčet vektor (mezi osami X a Y) o délce 0.607253 místo o délce 1. Při každé iteraci je třeba rozhodnout, zda přičíst nebo odečíst následující hodnotu Ø. To se provede porovnáním Ø s cílovou hodnotou.

Příklad 8.14

Navrhněte obvodovou realizaci výše uvedeného algoritmu.

Příklad 8.15

Pomocí algoritmu Cordic vypočtěte úhel, který svírá vektor $[x_A, y_A]$ s osou X.

9 Řetězené zpracování

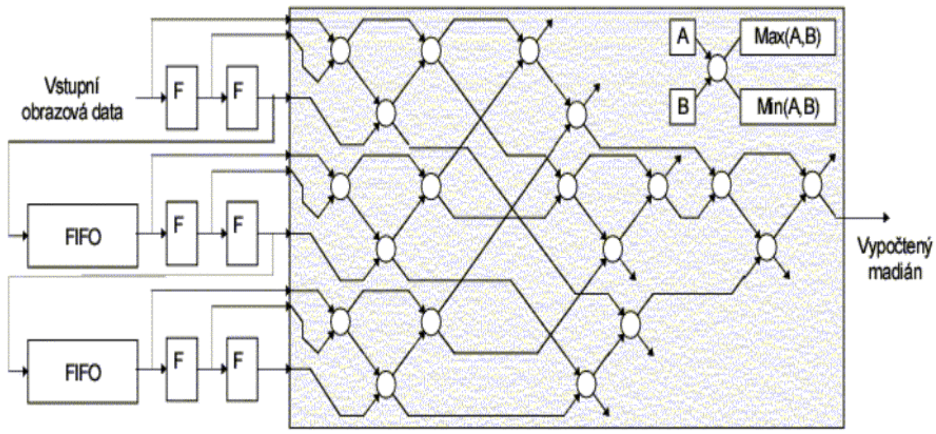
Zopakujte si: Princip řetězeného zpracování. Teoretické urychlení výpočtu. Optimální počet stupňů řetězené linky. Řetězené zpracování instrukcí. Konflikty a jejich odstranění.

Příklad 9.1

Vysvětlete princip zřetězeného zpracování.

Příklad 9.2

Vypočtěte dobu, za kterou bude upraven obraz filtrem, který se nazývá medián (viz šedý blok na obrázku A). Tento filtr pracuje tak, že nahradí daný pixel novou hodnotou, která se vypočítá jako medián z 9 hodnot (8 pixelů sousedících s daným pixelem a původní hodnota pixelu). Jednotlivé pixely prochází obvodem sestaveným z komponent (kolečka), které na jednom výstupu dávají maximum a na druhém výstupu minimum ze svých dvou vstupů. Předpokládejte zpoždění komponenty 10 ns a obraz velikosti 256 x 256 pixelů. Neuvažujte řetězené zpracování.



Obrázek A k příkladu

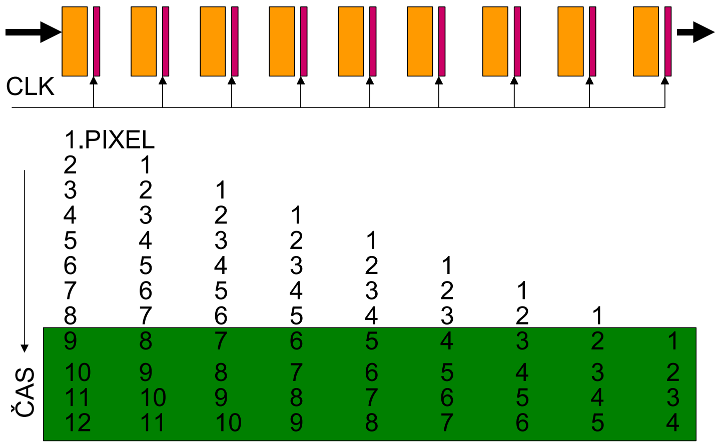
Na cestě od vstupu k výstupu je max. 9 komponent, tj. jeden pixel bude vyfiltrován za 9 x 10 ns = 90 ns. Celkem 256 x 256 x 90 ns = 5898240 ns, tj. filtrace bude trvat cca 5.9 ms.

Příklad 9.3

Vypočtěte zrychlení, které získáme zavedením řetězeného zpracování do předchozí úlohy.

Vzhledem k tomu, že je na cestě od vstupu k výstupu max. 9 komponent, můžeme filtr rozdělit na 9 stupňů. V každém stupni se budou vykonávat na sobě nezávislé operace (např. v prvním stupni se souběžně provedou operace v komponentách nakreslených v nejlevějším sloupci obrázku – v šedém bloku).

Po té, co první pixel opustí první stupeň řetězené linky, bude tento stupeň využit pixelem č. 2. Po té, co první pixel opustí druhý stupeň linky, bude tento stupeň obsazen pixelem č. 1 a první stupeň obsadí pixel č. 3 atd., viz obrázek B. První pixel bude vyfiltrován za 9 x 10 ns, tj. za 90 ns. 2., 3. až 65636. pixel budou vyfiltrovány každý za 10 ns. Celkem: 90 ns + 65535 x 10 ns = 656440 ns = 0.7 ms. Oproti předchozímu příkladu úlohu urychlíme 8.985 krát. Maximálního zrychlení (9, v našem případě) dosáhneme až po naplnění linky (zvýrazněná část obrázku B). Jednotlivé stupně linky musíme oddělit registry. Celý výpočet bude synchronizován signálem CLK.



Obrázek B k příkladu

Příklad 9.4

Odvoďte obecný vztah udávající zrychlení, když zavedeme řetězené zpracování využívající k stupňů. Předpokládejte, že se zpracovává N datových jednotek a jeden stupeň má zpoždění t. Kdy má smysl zavádět řetězené zpracování?

Příklad 9.5

Odvoďte optimální počet stupňů k_{opt} řetězené linky z pohledu maximalizace poměru výkon/cena. Uvažujte zpoždění a cenu registrů, které se musí do původního systému dodat.

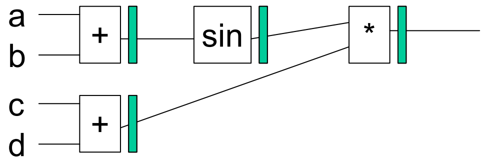
Původní systém má zpoždění p a jeho cena je C.

Zavedením řetězení musíme vložit k registrů. Potom je zpoždění stupně $p/k + r$, kde r je zpoždění registru. Celková cena vzroste na $C + kL$, kde L je cena registru. Maximalizujeme poměr Q = výkon/cena = V / (C + kL). Výkon V odpovídá maximálnímu kmitočtu, na kterém můžeme systém provozovat, tj. $fmax = 1/T = 1 / (p/k + r)$. Hledáme extrém funkce $Q(k)$,
$Q = 1 / [(p/k + r)(C + kL)]$. Tento extrém je

$$k_{opt} = \sqrt{\frac{pC}{rL}}$$

Příklad 9.6

Uvažujte řetězený systém sestávající ze tří stupňů na obrázku, který má počítat výraz $(c + d) * \sin(a + b)$
Úzké obdélníčky reprezentují synchronizační registry. Bude tento systém pracovat správně? Vysvětlete.



Příklad 9.7

Předpokládejte řetězené zpracování instrukcí se stupni F, D, E, M a W. Nakreslete diagram vytížení vstupů. Kdy bude linka naplněna? Uvažujte ideální případ.

Čas	1	2	3	4	5	6	7	8
F	i1	i2	i3	i4	i5	i6	i7	i8
D		i1	i2	i3	i4	i5	i6	i7
E			i1	i2	i3	i4	i5	i6
M				i1	i2	i3	i4	i5
W					i1	i2	i3	i4

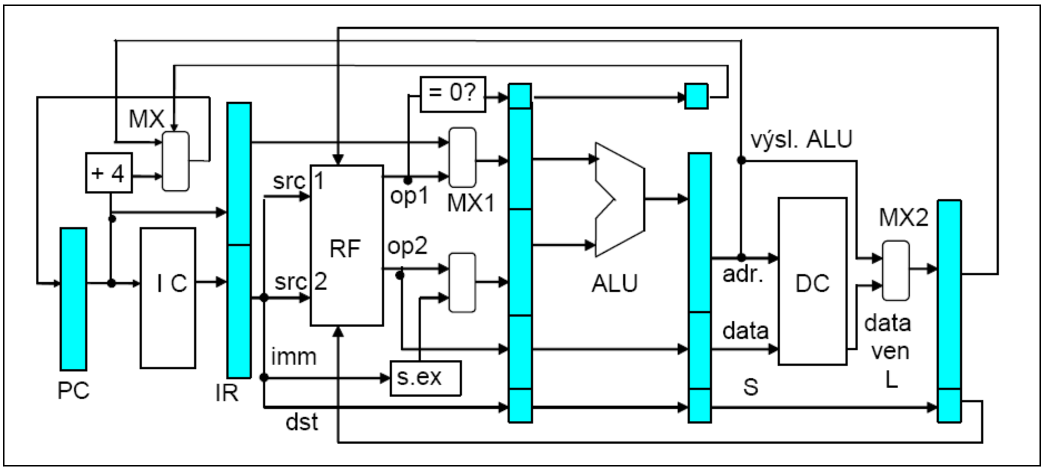
Linka bude naplněna po 5 krocích a bude urychlovat výpočet 5krát, pokud nebude docházet ke konfliktům.

Příklad 9.8

Předpokládejte řetězené zpracování instrukcí se stupni F, D, E, M a W. Nakreslete zpracování instrukcí. Uvažujte ideální případ.

Čas	1	2	3	4	5	6	7	8	9
I1	F	D	E	M	W				
I2		F	D	E	M	W			
I3			F	D	E	M	W		
I4				F	D	E	M	W	
I5					F	D	E	M	W
I6						F	D	E	M
I7							F	D	E
I8								F	D
I9									F

Příklad 9.9 Vysvětlete činnost zřetězeného procesoru DLX, který je uveden na následujícím obrázku.



Příklad 9.10 Které konflikty při zřetězeném zpracování označujeme jako strukturní? Jak se řeší? Uveďte příklady.

Příklad 9.11 Které konflikty při zřetězeném zpracování označujeme jako datové? Jak se řeší? Uveďte příklady.

Příklad 9.12 Které konflikty při zřetězeném zpracování označujeme jako řídicí? Jak se řeší? Uveďte příklady.

x+y

10 Řadiče a organizace počítače

Zopakujte si: Princip činnosti řadiče. Hlavní registry počítače. Metody návrhu řadičů. Návrh sekvenčních logických obvodů. Mikroprogramový řadič a formát mikroinstrukcí. Konstrukce řadiče ve vztahu k obecné koncepci počítače, způsobu adresování, adresovým módům a přerušení.

Příklad 10.1 Popište činnost řadiče typického skalárního procesoru při vykonání instrukcí LOAD addr (načti do střádače A obsah paměťové buňky s adresou addr), ADD A, B (sečti obsahy reg. A a B a výsledek ulož do A) JNZ addr (proveď podmíněný skok na adresu addr pokud je obsah střádače A nenulový). Využijte tyto pojmy: programový čítač, adresová sběrnice, datová sběrnice, paměť, registr instrukce, dekódování instrukce, registr příznaků, ALU, nastavení funkce ALU, hodinový cyklus apod. Nakreslete blokové schéma řadiče. Jak se řadič obvykle implementuje? Jaké jsou výhody a nevýhody jednotlivých přístupů?

Příklad 10.2 Vysvětlete princip činnosti mikroprogramového řadiče. Jakou má základní výhodu? Nakreslete blokové schéma.

Příklad 10.3 Vysvětlete rozdíly mezi koncepcí těchto počítačů: zásobníkový počítač, registrový počítač a střadačový počítač.

Příklad 10.4 Jaký je rozdíl mezi von neumannovskou a harvardskou koncepcí počítače?

Příklad 10.5 Jaké jsou možnosti adresování v procesorech (tzv. adresové módy)?

Příklad 10.6 Vysvětlete koncepci přerušení. Na jakých mechanismech může být mechanismus přerušení založen? Vysvětlete.

Příklad 10.7 Popište formáty mikroinstrukcí.

Příklad 10.8 Co to je nanoprogramování?

Příklad 10.9 Navrhněte část řadiče (automat a obvodovou realizaci), který vykonává několik vámi zvolených instrukcí. Zvolte si rovně procesor, pro který řadič navrhujete.

Příklad 10.10 Vysvětlete koncept výjimek.

x+y

11 Další podpůrné obvody

Zopakujte si: Pomocné speciální obvody v procesorech – válcový posunovač, bezpečný čítač, sada registrů, paměť typu FIFO apod.

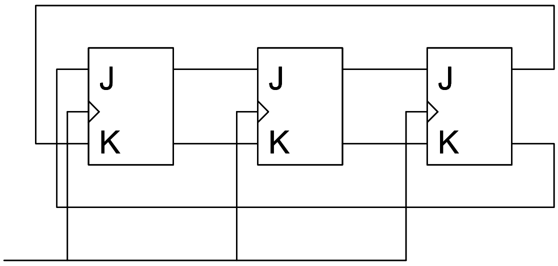
Příklad 11.1 Jakým způsobem se implementují obvody pro posun a rotace (tzv. válcový posunovač)? Demonstrujte princip na 4b obvodu, který provádí rotace o 0-3 bity vlevo.

Příklad 11.2 Navrhněte 16b válcový posunovač (vlevo) se strukturou (2, 2, 2, 2) (tj. využívající 4 stupně sestavené z dvouvstupových multiplexorů).

Příklad 11.3 Navrhněte 16b válcový posunovač (vlevo) se strukturou (2, 4, 2).

Příklad 11.4 Navrhněte 4b válcový posunovač provádějící rotace vpravo i vlevo o 0-3 bity.

Příklad 11.5 Zjistěte, zda je Johnsonův čítač uvedený na obrázku bezpečný.

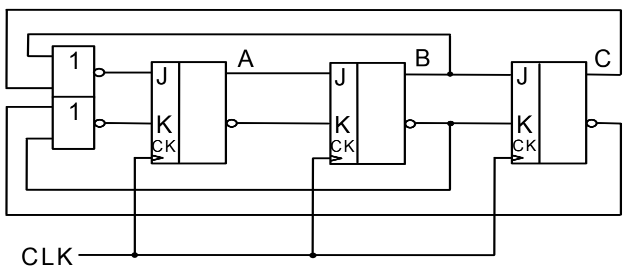


Čítač je bezpečný, pokud se dostaneme z jakéhokoliv počátečního stavu do hlavního cyklu. S využitím tabulky přechodů klopného obvodu JK budeme analyzovat činnost čítače. Předpokládejme, že jsou všechny klopné obvody ve stavu log. 0.

0	0	0	(0)
1	0	0	(1)
1	1	0	(3)
1	1	1	(7)
0	1	1	(6)
0	0	1	(4)
0	0	0	(0)
0	1	0	(2)
1	0	1	(5)
1	0	1	(5)
0	1	0	(2)

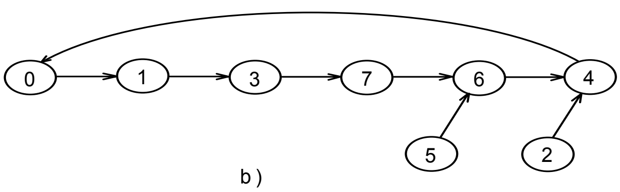
Je zřejmé, že čítač bezpečný není – pokud se dostane do stavu 2 nebo 5, tak se nevrátí do hlavního cyklu.

Příklad 11.6 Zjistěte, zda je čítač uvedený na obrázku bezpečný.



a)

Čítač (na obr. a) je bezpečný, protože je možné se vždy dostat do hlavního cyklu (viz obr. b).



b)

Příklad 11.7 Navrhněte 4b bezpečný Johnsonův čítač.

Příklad 11.8 Nakreslete schéma sady registrů s využitím standardních součástek. Sada bude obsahovat osm 16b registrů. Vstupy obvodu: 16b datový port, hodinový signál, 3b výběr registru (*sel*), zapisovací signál (*WE*), výběr čipu (*CE*). Výstupy: 16b datový výstup. Pokud budou signály *CE* i *WR* aktivní, dojde s náběžnou hranou hodinového signálu k zápisu vstupních dat do registru, který je určen hodnotou *sel*. Pokud je signál *CE* aktivní, je na výstupu k dispozici hodnota registru, který je určen hodnotou sel.

Příklad 11.9 Jakým způsobem se obvodově realizuje generování náhodných čísel?

12 Paměti

Zopakujte si: Principy fyzické realizace pamětí. Paměti ROM, RAM, SRAM, DRAM, SSRAM. Parametry pamětí. Vnitřní organizace pamětí. Paměťová hierarchie, lokalita odkazů. Rychlá vyrovnávací paměť. Virtuální paměť. Segmentování a stránkování. Překlad virtuální adresy. Překladové tabulky.

Příklad 12.1 Co to je lokalita odkazů?

Příklad 12.2 Co to je paměťová hierarchie?

Příklad 12.3 Jaké jsou parametry pamětí? Klasifikujte paměti podle těchto parametrů.

Příklad 12.4 Vysvětlete princip činnosti a vnitřní organizaci statické paměti. Nakreslete časový diagram čtení a zápisu.

Příklad 12.5 Vysvětlete princip činnosti a vnitřní organizaci dynamické paměti.

Příklad 12.6 Vysvětlete princip činnosti čtecího zesilovače DRAM.

Příklad 12.7 Vysvětlete jak funguje a proč se používá rychlá vyrovnávací paměť (cache).

Příklad 12.8 Co to je virtuální paměť. Co to je virtuální a fyzická adresa. Jak probíhá překlad virtuální adresy?

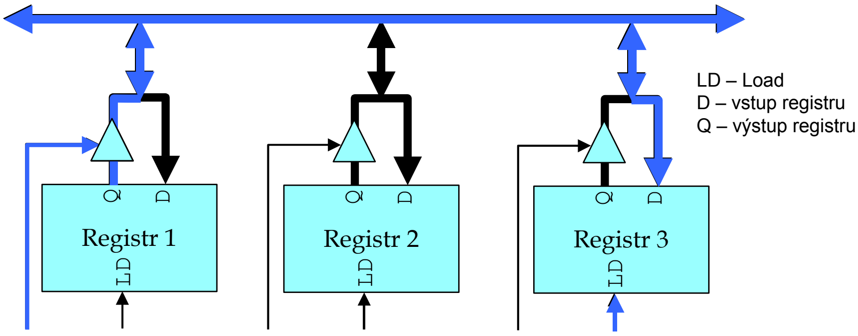
Příklad 12.9 Jaký je rozdíl mezi stránkováním a segmentováním?

13 Sběrnice a periferie

Zopakujte si: Typy sběrnic. Řízení sběrnice. Synchronní a asynchronní přenos dat. Připojování periferních zařízení.

Příklad 13.1 Vysvětlete princip a realizaci třístavového budiče sběrnice.

Příklad 13.2 Na uvedeném obrázku vysvětlete princip činnosti sběrnice. Proč se používají třístavové budiče?



Příklad 13.3 Proč se používá sběrnicová hierarchie?

Příklad 13.4 Na časových diagramech demonstrujte základní operace při synchronním a asynchronním přenosu dat.

Příklad 13.5 Popište mechanismy rozhodování, které se používají pro přidělování sdílených sběrnic.

Příklad 13.6 Vysvětlete možnosti ovládání periferních zařízení (programově, využití přerušení, DMA, využití I/O procesorů apod.).

14 Použitá literatura

Drábek, V.: Výstavba počítačů. Skriptum VUT v Brně, 1995
Hennessy, J., Patterson, A.: Computer Architecture: A Quantitative Approach, 2nd edition, Morgan Kaufmann Publ., 1996
Giese, Ch.: Cordic http://my.execpc.com/~geezer/embed/cordic.htm, 2005

Obsah