INFORMATIKA

J. GLENN BROOKSHEAR

počítačová architektura
operační systémy
počítačové sítě
algoritmy
programovací jazyky
vývoj softwaru
grafika
umělá inteligence
abstraktní teorie vyčíslitelnosti

computer press
ADDISON WESLEY

J. Glenn Brookshear

Informatika

Computer Press
Brno
2013

Informatika

J. Glenn Brookshear

a přispěvatelé:
David T. Smith
Indiana University of Pennsylvania
Dennis Brylow
Marquette University

Autoři obrázků
Obrázek 0.3: „Abakus". © Wayne Chandler. **Obrázek 0.4:** „Počítač Mark I". Laskavě poskytl archiv společnosti IBM. Neautorizované použití je zakázáno. **Obrázek 10.1:** „Fotografie virtuálního světa, který vznikl pomocí 3D grafiky (z filmu Toy Story: Příběh hraček, který vytvořily společnosti Walt Disney/Pixar Animation Studios)". © Disney/Pixar. **Obrázek 10.6:** „Scéna z filmu Shrek 2, který vznikl ve společnosti Dreamworks SKG". © Dreamworks/Picture Desk Inc./Kobal collection. **Obrázek 11.19:** „Výsledky použití neuronové sítě ke klasifikaci pixelů obrázku". Inspiraci poskytl web www.actapress.com. **Kapitola 11, Historické výkony robotů: a.** „Fotbalový robot kope do míče během turnaje RoboCup German Open 2010 dne 15. dubna 2010 ve východoněmeckém Magdeburgu". © Jens Schlueter/AFP/Getty Images/Newscom. **b.** „Robot Boss firmy Tartan Racing, vítěz závodu Urban Challenge sponzorovaného agenturou DARPA. Vozidla účastnící se tohoto závodu se musela sama pohybovat v městském prostředí". © DARPA. **c.** „Jeden z robotů Rover agentury NASA – robotický geolog, který zkoumá povrch planety Mars". Laskavě poskytla organizace NASA a laboratoř JPL-Caltech.

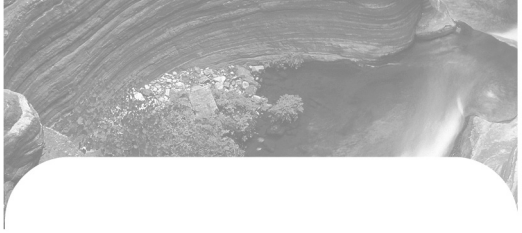
Překlad: Jakub Goner
Odborná korektura: Miroslav Virius
Obálka: Martin Sodomka
Odpovědný redaktor: Libor Pácl
Technický redaktor: Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-3805-2

Vydalo nakladatelství Computer Press v Brně roku 2013 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 16 684.

1. vydání

ALBATROS MEDIA a.s.



Obsah

**Hvězdičky označují doporučené volitelné části.*

Předmluva

Tato kniha představuje úvodní přehled informatiky. Zkoumá celou šíři oboru a jde přitom dostatečně do hloubky, aby mohli čtenáři získat komplexní představu o jednotlivých tématech.

Cílová skupina

Tento text byl napsán nejen pro studenty informatiky, ale také jiných oborů. Většina studentů informatiky nastupuje do školy s iluzí, že informatika se týká programování, webových stránek a sdílení souborů v Internetu, protože to je v zásadě vše, s čím se do té doby setkali. Informatika je však mnohem bohatší. Začínající studenti informatiky se tedy potřebují seznámit s celou šíří oboru, který chtějí absolvovat. O takové seznámení se snaží tato kniha. Poskytuje studentům celkový přehled o informatice. Díky těmto základům mohou později pochopit relevanci a vzájemné souvislosti dalších kurzů ze stejného oboru. Kurz je založen na modelu, který se používá pro úvodní texty v přírodních vědách.

Obecné základy potřebují také studenti jiných disciplín, aby rozuměli technické společnosti, ve které žijí. Kurz informatiky pro tyto studenty by měl poskytnout praktickou a realistickou představu o celém oboru a nikoli pouze úvod k používání Internetu nebo školení, jak pracovat s některými oblíbenými programy. Školení je samozřejmě také důležité, ale tento text má spíše vzdělávat.

Hlavním cílem při psaní knihy proto bylo, aby zůstala přístupná i pro studenty netechnických oborů. Předchozí vydání se díky tomu úspěšně uplatnila v kurzech pro studenty mnoha disciplín na různých vzdělávacích úrovních od středních škol až po magisterská studia. Toto jedenácté vydání by mělo v dosavadní tradici pokračovat.

Struktura textu

Přístup k jednotlivým tématům tohoto textu vychází zdola nahoru. Text postupuje od konkrétního k abstraktnímu, což je pořadí výhodné z hlediska pedagogické prezentace, kde jednotlivá témata vedou k následujícím. Kniha začíná základy kódování informací, ukládání dat a počítačové architektury (kapitoly 1 a 2), pokračuje studiem operačních systémů (kapitola 3) a počítačových sítí (kapitola 4), zkoumá témata algoritmů, programovacích jazyků a vývoje softwaru (kapitoly 5 až 7), analyzuje metody na zdokonalení přístupu k informacím (kapitoly 8 a 9), zabývá se některými důležitými aplikacemi informatiky na poli grafiky (kapitola 10) a umělé inteligence (kapitola 11) a závěrem předkládá úvod do abstraktní teorie výpočtů (kapitola 12).

Text sice postupuje tímto přirozeným způsobem, ale jednotlivé kapitoly a jejich části přitom zůstávají překvapivě nezávislé a obvykle je lze číst jako samostatné jednotky nebo změnit jejich uspořádání tak, aby poskytly alternativní výukový směr. Kniha se skutečně používá v různých kurzech, které s materiálem pracují z různých

hledisek. Jedna taková alternativa začíná materiálem kapitol 5 a 6 (Algoritmy a Programovací jazyky) a poté se podle potřeby vrací k předchozím kapitolám. Na druhou stranu existuje i kurz, který vychází od tématu vyčíslitelnosti z kapitoly 12. V jiných případech se text uplatňuje v kurzech pro pokročilé studenty, kde slouží pouze jako základna, na které staví projekty různého směru. Kurzy pro méně technicky orientované posluchače se mohou zaměřit na kapitoly 4 (Sítě a Internet), 9 (Databázové systémy), 10 (Počítačová grafika) a 11 (Umělá inteligence).

Na úvodní stránce každé z kapitol jsou některé části označeny hvězdičkami jako volitelné. Jedná se o části, které se zabývají speciálnějšími tématy, případně zkoumají tradičnější témata do větší hloubky. Uvedené informace by měly pouze navrhovat alternativní způsoby, jak v rámci textu postupovat. Lze samozřejmě využít i jiné trasy. Konkrétně čtenářům, kteří chtějí knihu zvládnout rychle, lze doporučit následující postup:

Část	Téma
1.1–1.4	Základy kódování a ukládání dat
2.1–2.3	Architektura počítačů a strojový jazyk
3.1–3.3	Operační systémy
4.1–4.3	Sítě a Internet
5.1–5.4	Algoritmy a jejich návrh
6.1–6.4	Programovací jazyky
7.1–7.2	Softwarové inženýrství
8.1–8.3	Datové abstrakce
9.1–9.2	Databázové systémy
10.1–10.2	Počítačová grafika
11.1–11.3	Umělá inteligence
12.1–12.2	Teorie vyčíslitelnosti

Celým textem se vine několik motivů. Jedním z nich je dynamičnost informatiky. Text často představuje témata z historické perspektivy, diskutuje aktuální stav poznání a naznačuje slibné výzkumné směry. Další motiv zdůrazňuje roli abstrakce a způsob, jakým lze pomocí abstraktních nástrojů zvládat složitost. Tento motiv představuje kapitola 0 a poté se opakuje v kontextu architektury operačních systémů, sítí, vývoje algoritmů, návrhu programovacích jazyků, softwarového inženýrství, organizace dat a počítačové grafiky.

Informace pro instruktory

Tento text zahrnuje více materiálu, než se obvykle probírá během jednoho semestru. Neváhejte proto vynechat témata, která nepatří k cílům vašeho kurzu, nebo změnit uspořádání témat, aby vám lépe vyhovovalo. Sami zjistíte, že ačkoli text sleduje určitou linku, vysvětluje témata převážně nezávislým způsobem, který umožňuje vybírat jednotlivé části podle potřeby. Tato kniha je sestavena tak, aby mohla poskytovat obsah kurzů, nikoli aby sloužila jako jejich pevná osnova. Studentům je vhodné doporučit, aby si přečetli i části, které se v příslušném kurzu výslovně neprobírají. Často studenty podceňujeme, když předpokládáme, že je nutné vše vysvětlit při přednášce. Měli bychom jim pomoci, aby se naučili učit sami.

Na tomto místě je nutné věnovat pár slov struktuře textu, který postupuje zdola nahoru od konkrétního k abstraktnímu. Učitelé se mnohdy domnívají, že studenti ocení jejich perspektivu, kterou si učitelé zpravidla tvoří během dlouhých let svého působení v oboru. Autor se domnívá, že by přednášející měli materiál předkládat spíše z hlediska studentů. Tento text proto začíná tématy reprezentace a ukládání dat, architektury počítačů, operačních systémů a sítí. Zmíněná témata studenti snadno pochopí: nejspíše už slyšeli pojmy jako JPEG a MP3, pravděpodobně vypalovali data na disky CD a DVD, kupovali počítačové komponenty, pracovali s operačním systémem a používali Internet. Začne-li kurz uvedenými tématy, studenti dostanou odpovědi na mnoho otázek, které si již léta kladou, a budou kurz považovat spíše za praktický než teoretický. Od těchto základů lze přirozeně přejít k abstraktnějším záležitostem algoritmů, algoritmických struktur, programovacích jazyků, metodik vývoje softwaru, vyčíslitelnosti a složitosti, které mnozí odborníci řadí mezi hlavní témata informatiky. Jak jsme již uvedli, témata jsou představena takovým způsobem, který od přednášejících nevyžaduje, aby k nim přistupoval zdola nahoru, i když je podle názoru autora vhodné to zkusit.

Všichni víme, že studenti se naučí mnohem více, pokud je učíme přímo, a implicitně získané poznatky si často osvojí lépe než ty, kterými se zabývají explicitně. To je důležité při „učení" dovednosti řešit problémy. Studenti se nenaučí řešit problémy tím, že budou studovat metodiky jejich řešení. Naučí se je řešit tím, že to vyzkoušejí, a neomezí se přitom na pečlivě vybrané „učebnicové úlohy". Kniha proto obsahuje mnoho problémů a některé z nich jsou záměrně nejasné, takže nemusí existovat jediný správný přístup či jediné správné řešení. Je vhodné tyto úlohy využít a stavět na nich.

Do kategorie „implicitního učení" patří také témata profesionality, etiky a společenské odpovědnosti. Podle názoru autora by se takový materiál neměl předkládat jako samostatné téma, které je pouze dodatkem celého kurzu. Místo toho by měl být integrální součástí studia, která se projevuje všude tam, kde je relevantní. Právě tento přístup se využívá i v předloženém textu. Části 3.5, 4.5, 7.8, 9.7 a 11.7 se dotýkají témat, jako je bezpečnost, ochrana soukromí, odpovědnost za škodu a povědomí o společenských otázkách v kontextu operačních systémů, sítí, databázových systémů, softwarového inženýrství a umělé inteligence. Část 0.6 navíc představuje tento motiv formou souhrnného přehledu důležitých teorií, které se pokoušejí postavit etické rozhodování na pevné filozofické základy. Na konci každé kapitoly se také nachází soubor dotazů s názvem *Společenské otázky*, které studenty motivují, aby uvažovali o tom, jak problematika příslušné kapitoly souvisí se společností, v níž žijí.

Děkuji, že uvažujete o využití mého textu ve svém kurzu. Ať už pro vaše účely bude vyhovovat, nebo nikoli, doufám, že jej přijmete jako můj příspěvek k informatické vzdělávací literatuře.

Pedagogická hlediska

Tento text vznikl na základě mnoha let výuky. Díky tomu je plný pedagogických prvků. Zásadní význam má dostatek problémů, které pomáhají při zapojení studentů. V 11. vydání je jich více než tisíc. Jsou zařazeny do kategorií Otázky a cvičení, Úlohy na procvičování témat kapitoly a Společenské otázky. Na konci každé části (s výjim-

kou úvodní kapitoly) se nachází pasáž Otázky a cvičení. Umožňuje probrat právě představenou problematiku, rozšířit předchozí diskuzi nebo navrhnout související témata, která se budou probírat později. Odpovědi na tyto otázky obsahuje Příloha F.

Na konci každé kapitoly (kromě úvodní) je umístěna část Úlohy na procvičování kapitoly. Tyto úlohy mají sloužit jako „domácí úkoly", protože se týkají materiálu z celé kapitoly a nejsou v knize zodpovězeny.

Na konci každé kapitoly jsou rovněž uvedeny problémy z kategorie Společenské otázky. Měly by vést k zamyšlení a diskuzi. Mnoho z nich lze zadat jako studentské projekty, které lze ukončit písemnou nebo ústní formou.

Na samém závěru každé kapitoly je také uveden seznam s názvem Další zdroje informací, který shrnuje odkazy na další materiály týkající se tématu příslušné kapitoly. Vhodné související informace lze najít také na webových stránkách, jejichž adresy jsou uvedeny v této předmluvě, v textu a v doplňkových rámečcích.

Informace pro studenty

Jsem poněkud nekonformní (někteří moji přátelé by řekli, že poněkud více). Když jsem se tedy pustil do psaní tohoto textu, pokaždé jsem se neřídil dobrými radami. Mnozí moji kolegové mi například tvrdili, že určité pasáže jsou pro začínající studenty příliš náročné. Domnívám se však, že pokud je téma relevantní, platí to i v případě, že je akademická komunita označí za „pokročilé". Zasloužíte si, aby se vám dostal do rukou text, který bude předkládat úplný obraz informatiky a nikoli vyretušovanou verzi s uměle zjednodušenými popisy pouze těch témat, u nichž autoři usoudili, že jsou pro úvod do studia vhodné. Proto jsem se žádným tématům nevyhýbal. Místo toho jsem se je snažil lépe vysvětlit. Pokusil jsem se dosáhnout dostatečné hloubky, abyste získali nezkreslenou představu o tom, co informatika obnáší. Stejně jako v případě koření v kuchařských receptech se můžete rozhodnout, že některá témata na následujících stránkách vynecháte. Budete-li mít však chuť, máte je k dispozici – a doporučuji vám, abyste je vyzkoušeli.

Měl bych také zdůraznit, že stejně jako pro jiné kurzy související s technologiemi platí, že podrobnosti, které se naučíte dnes, nemusíte již zítra potřebovat. Informatika je dynamická a to patří mezi důvody, proč je tak atraktivní. Tato kniha poskytuje aktuální přehled oboru i historickou perspektivu. Na těchto základech dokážete později své znalosti rozvíjet spolu s rozvojem technologií. Doporučuji vám, abyste začali již nyní a prozkoumali i jiné zdroje informací, než je tato kniha. Naučte se, jak se učit.

Děkuji, že mi důvěřujete a rozhodli jste se přečíst právě mou knihu. Mou povinností jako autora je napsat text, který bude stát za váš čas. Snad usoudíte, že jsem tento úkol úspěšně splnil.

Poděkování

V prvé řadě bych chtěl poděkovat všem, kdo mou knihu podpořili tím, že si přečetli a používali její předchozí vydání. Vaší přízni jsem poctěn.

David T. Smith (Indiana University of Pennsylvania) a Dennis Brylow (Marquette University) sehráli významnou roli při tvorbě tohoto jedenáctého vydání. David se zaměřil na kapitoly 0, 1, 2, 7 a 11 a Dennis se soustředil na kapitoly 3, 4, 6 a 10. Bez jejich tvrdé práce by toto nové vydání nevzniklo. Upřímně jim děkuji.

Jak jsem již zmínil v předmluvě k desátému vydání, jsem vděčný Edu Angelovi, Johnu Carpinellimu, Chrisu Foxovi, Jimu Kurosemu, Garymu Nuttovi, Gregu Riccardimu a Patricku Henrymu Winstonovi za jejich pomoc při práci na tomto předchozím vydání. Výsledek jejich úsilí je viditelný i v jedenáctém vydání.

K dalším lidem, kteří přispěli k tomuto či předchozím vydáním, patří J. M. Adams, C. M. Allen, D. C. S. Allison, R. Ashmore, B. Auernheimer, P. Bankston, M. Barnard, P. Bender, K. Bowyer, P. W. Brashear, C. M. Brown, H. M Brown, B. Calloni, M. Clancy, R. T. Close, D. H. Cooley, L. D. Cornell, M. J. Crowley, F. Deek, M. Dickerson, M. J. Duncan, S. Ezekiel, S. Fox, N. E. Gibbs, J. D. Harris, D. Hascom, L. Heath, P. B. Henderson, L. Hunt, M. Hutchenreuther, L. A. Jehn, K. K. Kolberg, K. Korb, G. Krenz, J. Liu, T. J. Long, C. May, J. J. McConnell, W. McCown, S. J. Merrill, K. Messersmith, J. C. Moyer, M. Murphy, J. P. Myers, Jr., D. S. Noonan, W. W. Oblitey, S. Olariu, G. Rice, N. Rickert, C. Riedesel, J. B. Rogers, G. Saito, W. Savitch, R. Schlafly, J. C. Schlimmer, S. Sells, G. Sheppard, Z. Shen, J. C. Simms, M. C. Slattery, J. Slimick, J. A. Slomka, D. Smith, J. Solderitsch, R. Steigerwald, L. Steinberg, C. A. Struble, C. L. Struble, W. J. Taffe, J. Talburt, P. Tonellato, P. Tromovitch, E. D. Winter, E. Wright, M. Ziegler a jeden anonym. Upřímně jim děkuji.

Jak jsem již uvedl, na doprovodném webu ke knize najdete příručky jazyků Java a C++, které poskytují základy těchto jazyků způsobem, který je kompatibilní se stylem knihy. Napsala je Diane Christieová. Děkuji Ti, Diane. Další poděkování směřuje k Rogeru Eastmanovi, jenž kreativně zpracoval aktivity pro jednotlivé kapitoly, které najdete i na doprovodném webu.

Oceňuji také práci týmu v nakladatelství Addison-Wesley, který se na tomto projektu podílel. Výborně se s nimi spolupracovalo a získal jsem mezi nimi nové přátele. Pokud uvažujete o tom, že byste napsali učebnici, doporučuji vám, abyste se obrátili právě na toto nakladatelství.

Nadále jsem vděčný své ženě Earlene a dceři Cheryl, které mě v uplynulých letech mimořádně povzbuzovaly v mé práci. Cheryl samozřejmě vyrostla a před několika lety odešla z domu. Earlene je však stále se mnou. Jsem opravdu šťastný muž. Ráno 11. prosince 1998 jsem přežil infarkt, protože mě včas dopravila do nemocnice. (Pro mladší z vás bych měl asi vysvětlit, že přežití infarktu je něco podobného, jak když dostanete více času na svůj domácí úkol.)

Nakonec děkuji svým rodičům, kterým tuto knihu věnuji. Na závěr uvádím jedno doporučení, jehož zdroj si přál zůstat v anonymitě: „Knížka našeho syna je opravdu dobrá. Měli by si ji přečíst všichni“.

J. G. B.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu přeložilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:
Computer Press
Albatros Media a.s., pobočka Brno
IBC
Příkop 4
602 00 Brno
nebo
sefredaktor.pc@albatrosmedia.cz
Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Errata

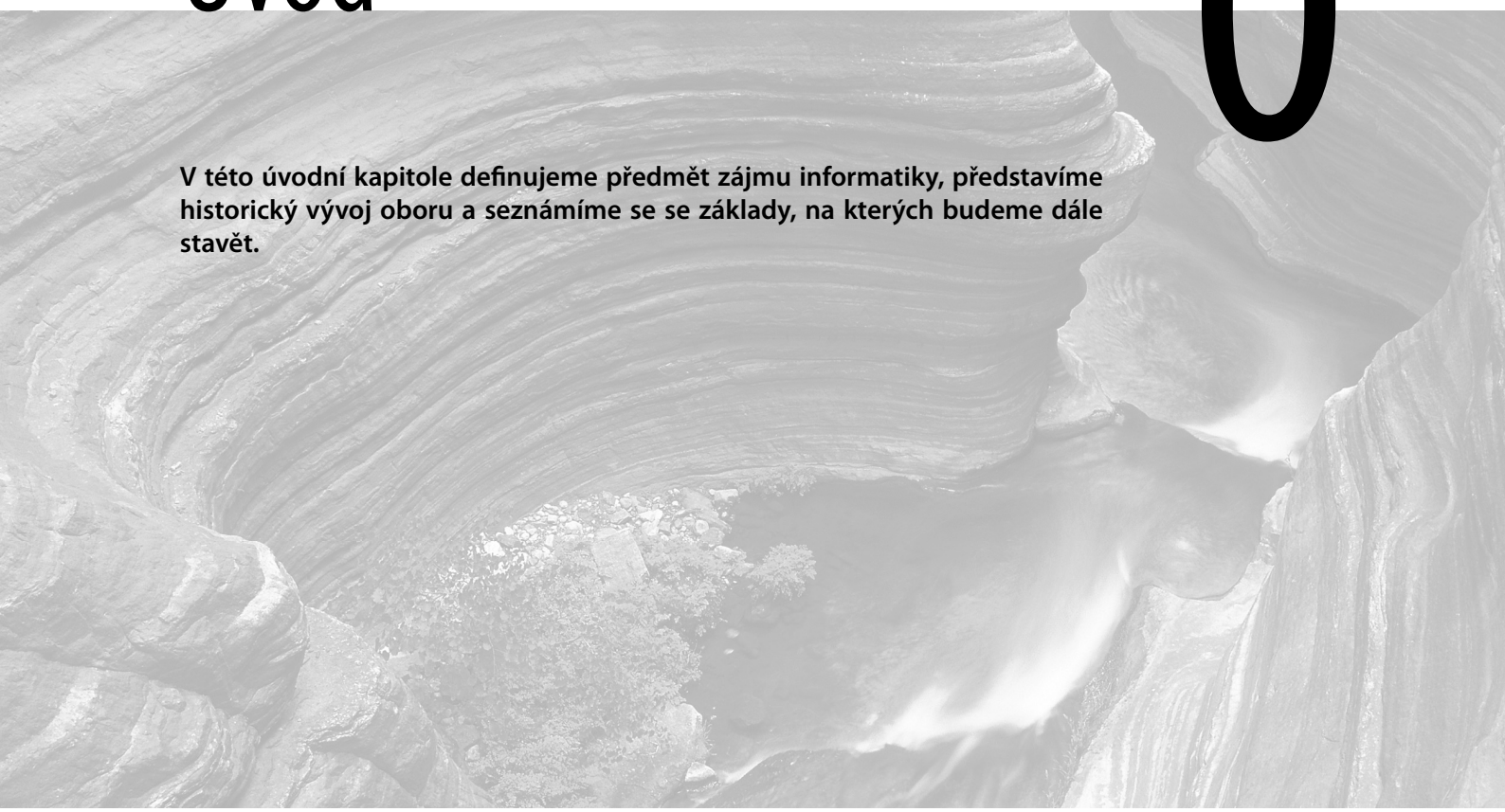
Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nelze. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji oznámíte. Ostatní uživatele tak můžete ušetřit frustrace a pomoci nám zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K2037* po klepnutí na odkaz Soubory ke stažení.

KAPITOLA

0

Úvod



V této úvodní kapitole definujeme předmět zájmu informatiky, představíme historický vývoj oboru a seznámíme se se základy, na kterých budeme dále stavět.

Informatika (computer science) je disciplína, která se snaží o vědecký přístup k tématům, k nimž patří např. počítačový návrh (design), počítačové programování, zpracování informací, algoritmické řešení problémů a vlastní algoritmické procesy. Poskytuje teoretické východisko pro dnešní počítačové aplikace i budoucí počítačovou infrastrukturu.

Tato kniha představuje obecný úvod do informatiky. Prozkoumáme mnoho různých otázek včetně většiny z těch, které obvykle tvoří osnovu univerzitních kurzů tohoto typu. Pokusíme se poznat celý rozsah a dynamiku oboru. Kromě vlastních témat tedy zaměříme svou pozornost i na jejich historický vývoj, aktuální stav výzkumu a vyhlídky do budoucna. Našim cílem je získat praktické znalosti informatiky, které uplatní jak zájemci o další specializované studium, tak studenti jiných směrů. Současná společnost totiž stále více závisí na technice a přehled v oboru informatiky je proto užitečný pro každého.

0.1: Role algoritmů

Začneme od nejzákladnějšího pojmu informatiky, což je algoritmus. Zjednodušeně řečeno představuje **algoritmus** sadu kroků, která definuje postup provedení úkolu. (Přesnější definici uvedeme v kapitole 5.) Existují například algoritmy vaření (říká se jim recepty), hledání cesty v neznámém městě (běžně se označují jako popisy cesty), ovládání praček (zpravidla jsou umístěny na vnitřní straně jejich víka nebo na zdi prádelny), interpretování hudby (mají podobu notové osnovy) či provádění kouzelnických triků (viz obrázek 0.1).

Než může stroj jako počítač splnit nějaký úkol, je nutné najít algoritmus provedení takového úkolu a vyjádřit jej v podobě, která je se strojem kompatibilní. Reprezentace algoritmu se nazývá **program**. Kvůli lepší čitelnosti pro lidské uživatele jsou počítačové programy obvykle vytištěny na papíře nebo zobrazeny na počítačové obrazovce. Z hlediska počítačů je výhodnější program zakódovaný způsobem, který odpovídá technologii daného počítače. Proces vývoje programu, jeho zakódování do kompatibilní podoby a vložení do stroje se označuje jako **programování**. Programy a algoritmy, které reprezentují, používají společný pojem **software**. Samotné fyzické přístroje se oproti tomu nazývají **hardware**.

Studium algoritmů spadá do sféry matematiky. Matematici totiž hledání algoritmů věnovali velké úsilí dlouho předtím, než se objevily dnešní počítače. Cílem bylo najít jednotnou posloupnost pokynů, která by dokázala popsat řešení všech problémů určitého typu. Mezi nejznámější příklady z raných dob tohoto zkoumání se řadí algoritmus písemného dělení, který dovoluje najít podíl dvou vícemístných čísel. Další příklad představuje algoritmus nalezený starořeckým matematikem Euklidem. Tento tzv. Euklidův algoritmus umožňuje zjistit největšího společného dělitele dvou celých kladných čísel (viz obrázek 0.2).

Jakmile je k dispozici algoritmus určitého úkolu, lze příslušný úkol splnit bez znalosti principů, na kterých je algoritmus založen. Provedení úkolu se místo toho omezuje na pouhé dodržování pokynů. (Chceme-li pomocí algoritmu písemného dělení spočítat podíl nebo pomocí Euklidova algoritmu určit největšího společného dělitele, nemusíme rozumět tomu, proč algoritmus funguje.) Inteligence, která je potřebná k vyřešení problému, je v jistém smyslu zakódována do příslušného algoritmu.

Efekt: Účinkující vezme několik karet z balíčku hracích karet, položí je lícem dolů, pečlivě je zamíchá a poté je rozloží po stole. Když pak diváci požádají o červenou nebo černou kartu, kouzelník obrátí kartu příslušné barvy.

Provedení triku:

Krok 1: Z normální sady karet vyberte deset červených a deset černých karet. Položte tyto karty na stůl lícem nahoru do dvou hromádek podle barvy.

Krok 2: Sdělte publiku, že jste vybrali několik červených a několik černých karet.

Krok 3: Zvedněte červené karty. Pod záminkou zarovnání hrany hromádky je podržte ve své levé ruce lícem dolů a palcem a ukazovákem pravé ruky potáhněte oba konce balíčku zpět, abyste všechny karty prohnuli mírně *dozadu*. Potom položte balíček červených karet na stůl lícem dolů a přitom řekněte: „V tomto balíčku jsou červené karty".

Krok 4: Vezměte černé karty. Podobným způsobem jako v kroku 3 prohněte tyto karty mírně *dopředu*. Potom vraťte tento balíček karet na stůl lícem dolů a prohlaste: „A v tomto balíčku máme černé karty".

Krok 5: Okamžitě poté, co položíte černé karty na stůl, promíchejte oběma rukama červené a černé karty (zůstávají lícem dolů) a rozprostřete je po stole. Vysvětlete, že musíte karty pečlivě promíchat.

Krok 6: Dokud na stole leží nějaké karty lícem dolů, opakujte následující kroky:

- 6.1: Zeptejte se obecenstva, zda máte vytáhnout červenou nebo černou kartu.
- 6.2: Pokud chce červenou kartu a na stole leží lícem dolů karta s vydutým tvarem, otočte takovou kartu s komentářem: „Tady máme červenou kartu".
- 6.3: Jestliže diváci požádali o černou kartu a na stole leží lícem dolů karta s vypouklým tvarem, otočte ji a prohlaste: „Tady máme černou kartu".
- 6.4: Jinak řekněte, že na stole nejsou žádné další karty příslušné barvy a abyste své tvrzení dokázali, otočte zbývající karty.

Obrázek 0.1: Algoritmus kouzelnického triku

Popis: Tento algoritmus předpokládá na svém vstupu dvě celá kladná čísla a poté vypočítá největší společný dělitel těchto dvou hodnot.

Postup:

Krok 1: Označte větší vstupní hodnotu jako M a menší vstupní hodnotu jako N.

Krok 2: Vydělte číslo M číslem N a zbytek nazvěte R.

Krok 3: Pokud se R nerovná 0, přiřaďte proměnné M hodnotu N, přiřaďte proměnné N hodnotu R a přejděte zpět ke kroku 2. V opačném případě se největší společný dělitel rovná hodnotě, která je aktuálně přiřazena proměnné N.

Obrázek 0.2: Euklidův algoritmus pro zjištění největšího společného dělitele dvou celých kladných čísel

Díky tomu, že algoritmy umožňují zaznamenat a předat inteligenci (nebo alespoň inteligentní chování), lze konstruovat stroje, které plní užitečné úkoly. Z toho vyplývá, že stroje mohou vykazovat pouze takovou úroveň inteligence, jaká se dá vyjádřit pomocí algoritmů. Stroj, který dokáže provést určitý úkol, můžeme vytvořit jen tehdy, jestliže je k provedení příslušného úkolu dostupný algoritmus. Na druhou stranu pokud na řešení problému žádný algoritmus neexistuje, leží řešení daného problému mimo možnosti strojů.

Analýza omezení možností algoritmů se ustavila jako oblast matematiky ve 30. letech 20. století, kdy Kurt Gödel publikoval svou větu o neúplnosti. Tato věta v zásadě prohlašuje, že v libovolné matematické teorii, která zahrnuje náš tradiční aritmetický systém, existují tvrzení, jejichž pravdivost či nepravdivost není možné dokázat algoritmickými postupy. Stručně řečeno: jakýkoli úplný popis našeho aritmetického systému přesahuje možnosti algoritmických metod.

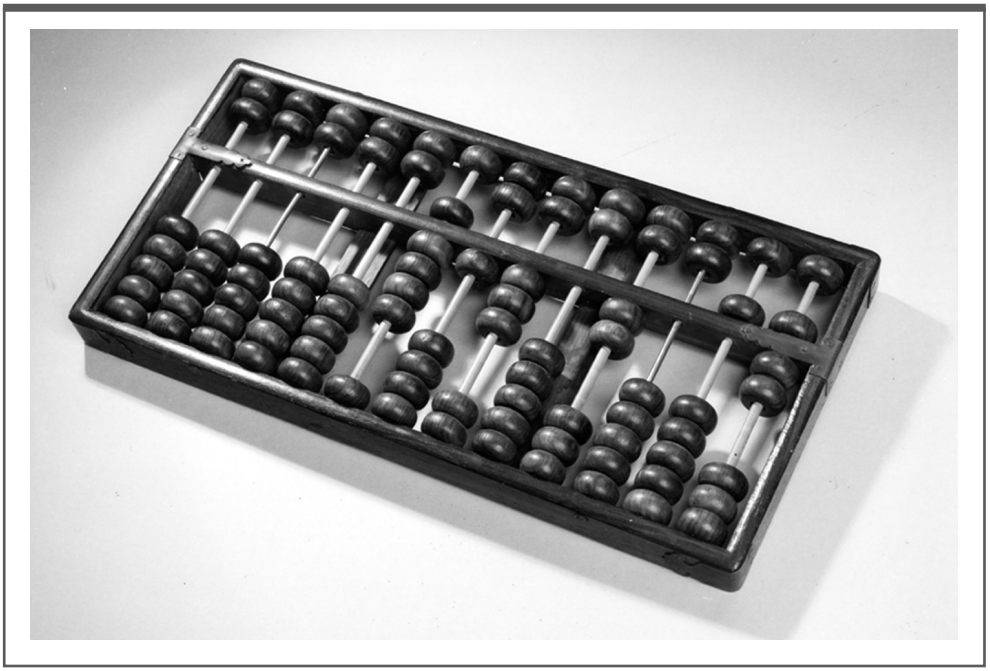
Toto zjištění otřáslo základy matematiky a následné studium algoritmických možností představovalo počátek oboru, který nyní známe jako informatiku. Studium algoritmů skutečně tvoří jádro informatiky.

0.2: Historie počítačů

Dnešní počítače mají dlouhý rodokmen. K nejstarším počítacím zařízením patřil abakus (vlastně počítadlo). Historie svědčí o tom, že abakus pravděpodobně vznikl ve starověké Číně a používaly jej rané řecké i římské civilizace. Tento přístroj je poměrně jednoduchý a sestává z kuliček navlečených na tyčích, které jsou samy usazeny v obdélníkovém rámu (viz obrázek 0.3). Poloha kuliček při jejich pohybu po tyčích do stran označuje uložené hodnoty. Právě pomocí poloh kuliček tento „počítač“ reprezentuje a uchovává data. Při vlastním provádění algoritmu se tento přístroj spoléhá na lidského operátora. Samotný abakus tedy funguje pouze jako systém ukládání dat. Teprve ve spojení s člověkem se mění na úplný výpočetní stroj.

Koncem středověku a na úsvitu moderní éry se objevily první pokusy o důmyslnější výpočetní stroje. Několik vynálezců začalo experimentovat s technologií ozubených převodů. K těmto průkopníkům patřili Francouz Blaise Pascal (1623–1662), Němec Gottfried Wilhelm Leibniz (1646–1716) a Angličan Charles Babbage (1792–1871). Jejich stroje reprezentovaly data polohami ozubených kol. Data se přitom zadávala ručně nastavením počáteční polohy převodů. Výsledky Pascalových a Leibnizových strojů bylo možné odečítat podle konečných poloh převodů. Babbage si oproti tomu představoval stroje, které by tiskly výsledky svých výpočtů na papír, aby se eliminovalo riziko chyb při ručním přepisu.

Z hlediska možností postupovat podle algoritmu lze u těchto strojů sledovat vývoj k vyšší pružnosti. Konstrukce Pascalova stroje umožňovala pouze sčítat. Příslušné pořadí kroků bylo proto nedílnou součástí vlastního stroje. Podobným způsobem byly algoritmy integrovány do architektury Leibnizova stroje, ačkoli tento poskytoval více aritmetických operací, ze kterých mohl operátor vybírat. Babbageův diferenční stroj (byl postaven pouze jako demonstrační model) bylo možné přizpůsobit tak, aby prováděl různé výpočty. Jeho analytický stroj (na jehož stavbu se vynálezci nepodařilo získat prostředky) měl číst instrukce ve formě děr v papírových kartách. Tento analytický stroj tedy umožňoval programování. Za prvního programátora se v současnosti považuje Augusta Ada Kingová, hraběnka z Lovelace, rozená Augusta Ada Byronová. Publikovala totiž článek, kde ukázala, jak lze Babbageův analytický stroj naprogramovat k různým výpočtům.



Obrázek 0.3: Abakus (autorem fotografie je Wayne Chandler)

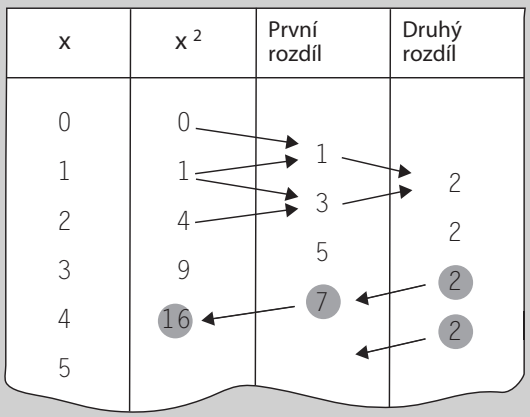
Samotný nápad předávání algoritmu pomocí děr v papíru nepochází od Babbage. Myšlenku si zapůjčil od Josepha Jacquarda (1752–1834). Ten roku 1801 sestrojil tkalcovský stav, u kterého postup tkaní závisel na děrovaných schématech ve velkých tlustých kartách vyrobených ze dřeva (nebo lepenky). Díky tomu bylo možné algoritmus stavu snadno změnit tak, aby vytvářel odlišné tkané vzory. Jacquardovu koncepci využil také Herman Hollerith (1860–1929), který využil myšlenku reprezentování informace děrami v papírových kartách, a tak dokázal urychlit americké sčítání lidu v roce 1890. (Tento Hollerithův projekt vedl ke vzniku společnosti IBM.) Karty uvedeného typu se nakonec rozšířily pod názvem děrné štítky a zachovaly se jako oblíbený způsob komunikace s počítači až do 70. let 20. století. Děrné štítky ve skutečnosti přežily do současnosti, jak se ukázalo při potížích se sčítáním hlasů v amerických prezidentských volbách roku 2000.

Složité stroje s převody, jaké si představovali Pascal, Leibniz a Babbage, nebylo možné s tehdejší technologií vyrobit tak, aby byly ekonomické. Tuto bariéru se však podařilo překonat díky pokrokům v elektronice na začátku 20. století. Jako příklady tohoto pokroku lze uvést elektromechanický stroj George Stibitze[1] dokončený roku 1940 v Bellových laboratořích a počítač Mark I, který roku 1944 na Harvardské univerzitě zkonstruoval Howard Aiken se skupinou techniků ze společnosti IBM (viz obrázek 0.4). Tyto přístroje obsahovaly mnoho elektronicky řízených mechanických relé. V tomto smyslu byly zastaralé téměř ihned po svém vzniku, protože jiní výzkumníci využili technologii elektronek, která umožňovala vytvořit kompletně elektronické počítače. První z těchto strojů pravděpodobně sestavil John Atanasoff se svým asistentem Cliffordem Berrym v období let 1937 až 1941 na Iowa State College (nyní Iowa State University). Dalším zástupcem byl počítač s názvem Colossus, který vybudoval tým pod vedením Tommyho Flowerse v Anglii kvůli dešifrování

1 První elektromechanický reléový počítač Z1 postavil ve skutečnosti Konrad Zuse v roce 1936 v Berlíně (plně dokončen byl v r. 1939). Pozn. odborného korektora.

Babbageův diferenční stroj

Přístroje navržené Charlesem Babbagem představovaly skutečné vzory moderního počítačového návrhu. Kdyby tehdejší technologie umožňovala vyrábět jeho stroje ekonomicky a pokud by podniky a vládní úřady potřebovaly zpracovávat data v takovém měřítku jako nyní, mohly by Babbageovy myšlenky zahájit počítačovou revoluci již v 19. století. Ve skutečnosti však za Babbageova života vznikl pouze ukázkový model jeho diferenčního stroje. Tento stroj určoval číselné hodnoty počítáním „postupných diferencí". Příslušnou metodu si můžeme objasnit na příkladu problému s výpočtem druhých mocnin celých čísel. Vycházíme ze znalosti faktu, že druhá mocnina 0 je 0, druhá mocnina 1 je 1, druhá mocnina 2 je 4 a druhá mocnina 3 se rovná 9. Z toho můžeme dospět ke druhé mocnině čísla 4 následujícím způsobem (viz následující diagram). Nejdříve vypočítáme rozdíl druhých mocnin, které již známe: $1^2 - 0^2 = 1$, $2^2 - 1^2 = 3$ a $3^2 - 2^2 = 5$. Poté stanovíme rozdíly těchto výsledků: $3 - 1 = 2$ a $5 - 3 = 2$. Můžeme si všimnout, že oba tyto rozdíly se rovnají dvěma. Za předpokladu, že toto pravidlo platí i nadále (matematici mohou dokázat, že tomu tak skutečně je), usoudíme, že rozdíl mezi hodnotou $(4^2 - 3^2)$ a hodnotou $(3^2 - 2^2)$ musí být také 2. Z toho vyplývá, že hodnota výrazu $(4^2 - 3^2)$ musí být o 2 větší než hodnota výrazu $(3^2 - 2^2)$, takže $4^2 - 3^2 = 7$ a tedy $4^2 = 3^2 + 7 = 16$. Když nyní známe hodnotu druhé mocniny čísla 4, můžeme stejným postupem pokračovat výpočtem druhé mocniny čísla 5. Vycházíme přitom z hodnot 1^2, 2^2, 3^2 a 4^2. (Podrobnější rozbor postupných diferencí přesahuje rámec této kapitoly. Studenti diferenciálního a integrálního počtu si však mohou povšimnout, že předchozí příklad je založen na faktu, že grafem derivace výrazu $y = x^2$ je přímka se směrnicí 2.)



německých zpráv v pozdějších fázích druhé světové války. (Ve skutečnosti vzniklo až deset strojů tohoto typu, ale kvůli vojenskému utajování a ohledům na národní bezpečnost se nestaly součástí „počítačového rodokmenu".) Brzy následovaly další a flexibilnější přístroje jako ENIAC (z anglického výrazu „electronic numerical integrator and calculator"), který vyvinuli John Mauchly a J. Presper Eckert na Moore School of Electrical Engineering při Pensylvánské univerzitě.

Od této fáze byl vývoj počítacích strojů úzce svázán s technologickými pokroky, mezi něž patří vynález tranzistorů (za který fyzikové William Shockley, John Bardeen a Walter Brattain získali Nobelovu cenu) a následný vývoj kompletních obvodů konstruovaných jako samostatné jednotky, které získaly název integrované obvody (tento vynález zajistil Nobelovu cenu za fyziku Jackovi Kilbymu). Díky těmto novinkám se stroje ze 40. let 20 století o velikosti celé místnosti během několika dekád zmenšily na velikost pouhých skříní. Současně se výpočetní výkon počítačů začal každé dva roky zdvojnásobovat (tento trend pokračuje dodnes). Jak postupovaly práce na vývoji integrovaných obvodů, dostalo se mnoho obvodů používaných v počítačích na volný trh, kde byly dostupné ve formě miniaturních plastových pouzder zvaných čipy.

Obrázek 0.4: Počítač Mark I (Laskavě poskytl archiv společnosti IBM. Neautorizované použití je zakázáno.)

Zásadním krokem k rozšíření výpočetní techniky byl vývoj stolních počítačů. Počátky těchto přístrojů je možné vysledovat až k počítačovým nadšencům, kteří si podomácku stavěli počítače z jednotlivých čipů. Právě mezi tyto „undergroundové" fanoušky počítačů patřili Steve Jobs a Stephen Wozniak, kteří vytvořili komerčně úspěšný domácí počítač a roku 1976 založili společnost Apple Computer, Inc. (nyní Apple Inc.) na výrobu a odbyt svých produktů. Podobné výrobky dodávaly i firmy Commodore, Heathkit a Radio Shack. Jejich produkty měly sice dobrý zvuk u zájemců o počítače, ale podniková komunita je příliš neakceptovala a se svými potřebami v oblasti zpracování dat se obracela hlavně na zavedenou společnost IBM.

V roce 1981 společnost IBM představila svůj první stolní počítač, který se označoval jako osobní počítač („personal computer" neboli PC). Základní software tohoto počítače vyvinula nově vytvořená společnost s názvem Microsoft. Počítače PC se okamžitě setkaly s úspěšným přijetím a podniková sféra díky nim začala stolní počítače brát vážně. V současnosti se termín *PC* obecně používá nezávisle na výrobci pro všechny přístroje, jejichž architektura vychází z původního stolního počítače společnosti IBM. Většina z těchto počítačů se na trh nadále dodává se softwarem od společnosti Microsoft. Termín *PC* se však někdy nahrazuje obecnými názvy *stolní počítač* (pracovní stanice, desktop) nebo *přenosný počítač* (notebook, laptop).

V posledních letech 20. století se mezilidská komunikace zásadně proměnila díky možnostem, které poskytuje připojení jednotlivých počítačů do celosvětové sítě zvané **Internet**. Britský vědec Tim Berners-Lee je autorem návrhu systému, který umožňoval provázat dokumenty uložené v počítačích připojených k Internetu a vytvořit tak pavučinu souvisejících informací. Tento systém získal název **World Wide Web** (výraz se často zkracuje na slovo „web"). Kvůli zpřístupnění informací na webu vznikly softwarové systémy označované jako **vyhledávače** (search engine), jejichž úkolem

Augusta Ada Byronová

Osobnost Augusty Ady Byronové, hraběnky z Lovelace, se v počítačové komunitě těší značné pozornosti. Její nepříliš šťastný život trval jen necelých 37 let (1815–1852). Trpěla kvůli svému chatrnému zdraví a také své nekonformní povaze, protože tehdejší společnost kladla překážky profesionálnímu uplatnění žen. Ada se sice zajímala o mnoho různých vědeckých oborů, ale zaměřovala se hlavně na matematiku. „Vědou o počítání" se začala zabývat poté, kdy na ni v roce 1833 mimořádně zapůsobila demonstrace prototypu diferenčního stroje Charlese Babbage. Její příspěvek k informatice je založen na anglickém překladu francouzského článku, ve kterém Babbage rozebíral svůj návrh analytického stroje. Babbage ji vyzval, aby k překladu připojila vlastní dodatek s popisem aplikací stroje a příklady programování stroje k různým úkolům. Babbage byl prací Augusty Ady nadšen pravděpodobně proto, že doufal ve finanční podporu stavby svého analytického stroje, kterou měla její publikace přinést. (Jako dcera lorda Byrona patřila Augusta Ada mezi tehdejší celebrity, a mohla tak otevřít dveře k mnoha bohatým lidem.) Tuto podporu sice Babbage nikdy nezískal, ale dodatek Augusty Ady se zachoval a obsahuje příklady, které lze považovat za první počítačové programy. Historici diskutují o tom, do jaké míry Babbage ovlivnil práci Augusty Ady. Někteří tvrdí, že k ní Babbage výrazně přispěl, zatímco jiní oponují, že Augustě Adě spíše překážel. Augusta Ada Byronová je každopádně uznávána jako první programátor na světě. Tento titul potvrdilo i Ministerstvo obrany USA, které na její počest zvolilo název Ada pro významný programovací jazyk.

je „procházet" celý web, „roztřídit" zjištěné informace a poté poskytovat výsledky uživatelům, kteří hledají konkrétní téma. V této oblasti mají hlavní slovo společnosti Google, Yahoo a Microsoft. Tyto firmy nadále rozšiřují své webové aktivity. Jejich inovace přitom často narušují náš zaběhlý způsob uvažování.

V době, kdy domácí uživatelé přijali stolní počítače (a později i přenosné notebooky), dále pokračoval trend miniaturizace těchto strojů. Počítače nepatrných rozměrů jsou v současnosti nedílnou součástí různých zařízení. Automobily jsou nyní například vybaveny malými počítači, které vyhodnocují polohu podle navigačního systému GPS (Global Positioning System), sledují funkce motoru a poskytují služby hlasového ovládání, díky nimž lze nastavovat integrované stereosystémy nebo telefonovat.

Pravděpodobně nejdůležitější změny může počítačová miniaturizace přinést díky stále větším možnostem přenosných telefonů. Donedávna pouhé telefony se vyvinuly v malé univerzální počítače do dlaně, pro něž se vžilo označení **smartphone**. Telefonování přitom představuje pouze jednu z mnoha jejich aplikací. „Telefony" tohoto typu jsou vybaveny mnoha různými senzory a rozhraními včetně fotoaparátů, mikrofonů, kompasů, dotykových obrazovek a akcelerometrů (které zjišťují orientaci a směr pohybu telefonu). Využívají také různé bezdrátové technologie, které jim dovolují komunikovat s jinými smartphony a počítači. Tyto přístroje mají mimořádný potenciál. Mnozí dokonce prohlašují, že smartphone ovlivní společnost více než osobní počítače.

Díky miniaturizaci počítačů a rozšiřování jejich možností zaujaly počítačové technologie klíčové místo v současné společnosti. Počítačové technologie se nyní uplatňují tak masivně, že bez jejich základní znalosti nelze v moderním světě zaujmout plnohodnotné místo. Tyto technologie ovlivňují schopnost vlád kontrolovat občany, mají mimořádný dopad na globální ekonomiku, umožňují dosáhnout značných pokroků ve vědeckém výzkumu, převratně mění shromažďování, ukládání a zpracování dat, poskytují nové metody mezilidské komunikace a opakovaně narušují společenský status quo. Objevují se proto nové aplikované oblasti informatiky, které se v současnosti vesměs řadí mezi etablované obory samy o sobě. Stejně jako v případě

strojírenství a fyziky je navíc často obtížné nakreslit dělicí čáru mezi novým oborem a obecnou informatikou. Abychom tedy získali vhodnou perspektivu, nebudeme se zabývat pouze ústředními tématy informatiky, ale prozkoumáme také různé další disciplíny a budeme přitom sledovat aplikace i dopady výzkumu. Samotný úvod do informatiky vyžaduje interdisciplinární přístup.

0.3: Studium algoritmů

Faktory, jako byla omezená kapacita datového úložiště a náročné a zdlouhavé programovací postupy, omezovaly složitost algoritmů realizovaných pomocí nejstarších počítačů. Spolu s tím, jak tato omezení postupně mizela, bylo možné počítačům zadávat stále větší a složitější úkoly. Pokusy vyjádřit příslušné úkoly v algoritmické formě začínaly přesahovat schopnosti lidské mysli. Úsilí výzkumníků se proto zaměřilo na studium algoritmů a procesu programování.

V tomto kontextu začínala přinášet plody teoretická práce matematiků. V důsledku Gödelovy věty o neúplnosti matematici zkoumali otázky týkající se algoritmických procesů již před tím, než příslušné odpovědi začal vyžadovat technologický pokrok. Uzrála tedy doba pro ustavení nové disciplíny, která se označuje jako *informatika* (computer science).

V současnosti se informatika definuje jako věda o algoritmech. Informatika má široký záběr a dotýká se natolik rozmanitých oborů, jako je matematika, strojírenství, psychologie, biologie, řízení podniků a lingvistika. Informatici mohou v závislosti na své specializaci definovat svou vědu značně rozdílně. Výzkumník v oblasti architektury počítačů se například může zaměřovat na miniaturizaci integrovaných obvodů a na informatiku může tedy pohlížet z hlediska pokroků a aplikací technologie. Specialista na databázové systémy však oproti tomu může zdůrazňovat, že informatika usiluje o větší užitečnost informačních systémů. Odborník na poli umělé inteligence pak může informatiku považovat za studium inteligence a inteligentního chování.

Úvod do informatiky proto musí zahrnovat různá témata a tomuto úkolu se budeme věnovat v následujících kapitolách. V každém případě bude naším cílem představit hlavní myšlenky příslušné oblasti, aktuální výzkumná témata a některé metody, které se používají k rozvoji znalostí o dané problematice. Vzhledem ke značné šíři té-

Google

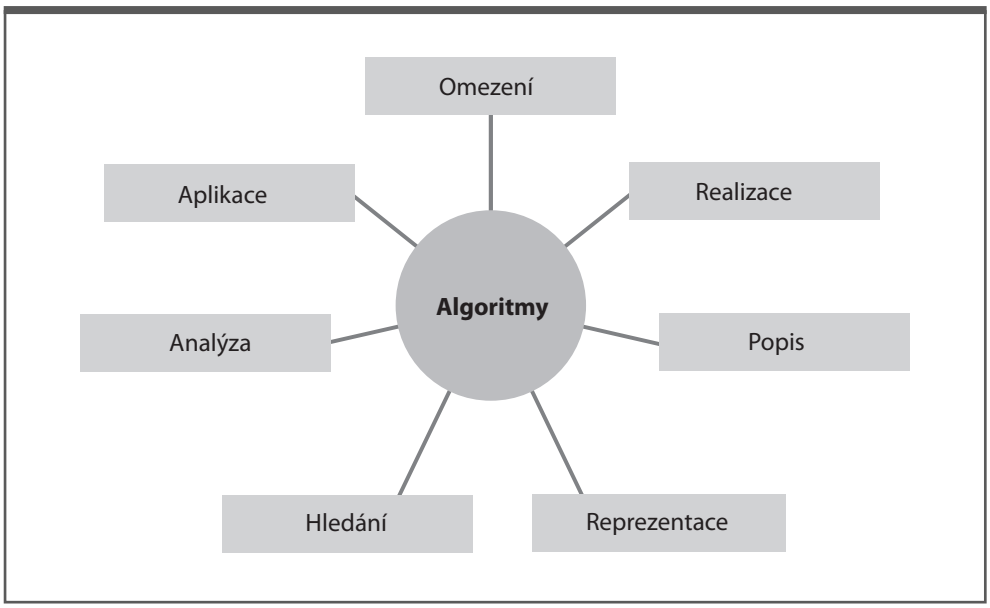
Společnost Google Inc. založená roku 1998 se rychle zařadila mezi nejznámější technologické firmy na světě. Její ústřední službu – vyhledávač Google – používají při hledání dokumentů na webu miliony lidí. Společnost Google kromě toho poskytuje služby elektronické pošty (nazývá se Gmail), internetovou službu sdílení videa (YouTube) a celou paletu dalších internetových služeb (k nimž patří Google Maps, Google Calendar, Google Earth, Google Books a Google Translate).

Ačkoli představuje dokonalou ukázku podnikatelského ducha, nabízí společnost Google také příklady toho, jak šíření nových technologií způsobuje společenské změny. V souvislosti s vyhledávačem Google se například vynořily otázky, do jaké míry by měla mezinárodní společnost vyhovět požadavkům jednotlivých vlád. Služba YouTube vyvolala diskuze o tom, nakolik by měla společnost odpovídat za informace, které jiné subjekty distribuují pomocí jejích služeb, a také ohledně toho, do jaké míry si může společnost nárokovat vlastnictví těchto informací. Služba Google Books rozvířila obavy týkající se rozsahu práv k duševnímu vlastnictví a jejich omezení a službu Google Maps někteří obviňují z porušování práva na soukromí.

mat lze snadno ztratit ze zřetele celkový obrázek. Krátce se proto zastavíme, abychom uspořádali své myšlenky a položili si některé otázky, které nám při studiu informatiky dají správný směr.

- Které problémy lze vyřešit algoritmickým zpracováním?
- Jak lze usnadnit hledání algoritmů?
- Jak lze zdokonalit metody reprezentace a popisu algoritmů?
- Jak lze analyzovat a porovnávat vlastnosti různých algoritmů?
- Jak lze pomocí algoritmů manipulovat s informacemi?
- Jak je možné aplikací algoritmů dospět k inteligentnímu chování?
- Jak použití algoritmů ovlivňuje společnost?

Všechny tyto otázky mají společný motiv, kterým je studium algoritmů (viz obrázek 0.5).



Obrázek 0.5: Ústřední role algoritmů v informatice

0.4: Abstrakce

Princip abstrakce prostupuje studiem informatiky a návrhem počítačových systémů do té míry, že je vhodné, abychom se jím zabývali již v této úvodní kapitole. Termín **abstrakce**, jak jej zde používáme, označuje rozdíl mezi externími vlastnostmi entity a podrobnostmi o interním složení příslušné entity. Právě abstrakce nám umožňuje ignorovat podrobnosti týkající se vnitřního složení komplexního zařízení, jako je počítač, automobil nebo mikrovlnná trouba, a používat dané zařízení jako logickou jednotku. Navíc právě díky abstrakci lze takové složité systémy vůbec navrhovat a vyrábět. Počítače, auta a mikrovlnky se skládají ze součástí a každá z nich obsahuje

menší komponenty. Jednotlivé komponenty představují úrovně abstrakce, na kterých je použití komponenty odděleno od podrobností její vnitřní struktury.

Na základě abstrakce umíme konstruovat, analyzovat a řídit velké a komplexní počítačové systémy, které by se v celkovém pohledu se zahrnutím všech detailů vymykaly možnostem chápání. Na každé úrovni abstrakce popisujeme systém pomocí komponent zvaných **abstraktní nástroje**, jejichž vnitřní složení přitom ignorujeme. Můžeme se tak soustředit na to, jak jednotlivé komponenty interagují s jinými komponentami na stejné úrovni a jak v souhrnu tvoří komponentu vyšší úrovně. Dokážeme tak porozumět části systému, která je relevantní pro příslušný úkol, a neztratíme se přitom v záplavě detailů.

Je potřeba zdůraznit, že abstrakce se neomezuje na doménu vědy a technologie. Představuje důležitou metodu zjednodušení práce, která naší společnosti umožňuje dosáhnout životního stylu, jaký by jinak nebyl reálný. Málokdo z nás ví, jak v praxi fungují různé pomůcky, které nám usnadňují život. Jíme jídlo, které si neumíme sami připravit, a nosíme šaty, které neumíme sami utkat. Pracujeme s elektrickými zařízeními a komunikačními systémy, aniž bychom rozuměli jejich technologickým principům. Využíváme služeb jiných lidí a přitom přesně neznáme, co jejich profese obnášejí. Jakmile se objeví technologická novinka, pokaždé se malá část společnosti rozhodne specializovat na její implementaci, zatímco všichni ostatní se naučí používat výsledky jejich práce jako abstraktní nástroje. Společnost má tak k dispozici stále více abstraktních nástrojů a může na nich založit svůj další pokrok.

Téma abstrakce se vine celou knihou. Naučíme se, že počítačová zařízení se konstruují na jednotlivých úrovních abstraktních nástrojů. Dozvíme se také, že vývoj velkých softwarových systémů probíhá modulárním způsobem, kdy se každý modul používá jako abstraktní nástroj ve větších modulech. Abstrakce kromě toho hraje důležitou roli v pokrocích samotné informatiky, protože vědci mohou zaměřit svou pozornost na konkrétní aspekty složité problematiky. Uvedená vlastnost oboru se projevuje i na uspořádání tohoto textu. Jednotlivé kapitoly, které se soustřeďují na určitou oblast celého oboru, často překvapivě nezávisí na ostatních kapitolách. Přesto však všechny společně přinášejí komplexní přehled o rozsáhlé vědě.

0.5: Osnova knihy

Tato kniha přistupuje ke studiu informatiky zdola nahoru. Začíná praktickými tématy typu počítačového hardwaru a postupuje k abstraktnějším tématům, jako je složitost a vyčíslitelnost algoritmů. Při studiu tedy budeme postupně budovat stále větší abstraktní nástroje spolu s tím, jak budeme rozšiřovat své znalosti problematiky.

Začneme tématy, která souvisejí s návrhem a konstrukcí strojů pro realizaci algoritmů. V kapitole 1 (Ukládání dat) si ukážeme, jak moderní počítače kódují a ukládají informace, a v kapitole 2 (Zpracování dat) budeme zkoumat základní interní operace jednoduchého počítače. Tato kniha se sice částečně zabývá technologiemi, ale hlavní motiv na konkrétní technologii nezávisí. Témata, k nimž patří návrh digitálních obvodů, systémy kódování a komprese dat a počítačová architektura, jsou relevantní v mnoha různých aplikacích a bez ohledu na budoucí technologický vývoj pravděpodobně zůstanou relevantní i nadále.

V kapitole 3 (Operační systémy) budeme studovat software, který řídí základní funkce počítače. Tento software se označuje jako operační systém. Operační systém počítače kontroluje rozhraní mezi strojem a vnějším světem, chrání počítač a data v něm uložená před neoprávněným přístupem, umožňuje uživateli počítače spouštět různé programy a koordinuje interní funkce, které jsou potřebné k realizaci uživatelských požadavků.

V kapitole 4 (Sítě a Internet) se budeme zabývat propojením počítačů do počítačových sítí a propojením jednotlivých místních sítí do Internetu. V této části se dostaneme k tématům jako síťové protokoly, struktura Internetu a principy jeho fungování, web a různá bezpečnostní hlediska.

Kapitola 5 (Algoritmy) uvádí studium algoritmů z formálnějšího hlediska. Prozkoumáme, jak lze hledat nové algoritmy, identifikujeme několik základních algoritmických struktur, vyvineme základní metody reprezentace algoritmů a představíme témata efektivity a správnosti algoritmů.

V kapitole 6 (Programovací jazyky) se zaměříme na téma reprezentace algoritmů a proces vývoje programů. Ukážeme si, že hledání lepších programovacích metod dospělo k různým programátorským metodikám neboli paradigmatům. Každá z těchto metodik přitom používá vlastní sadu programovacích jazyků. Rozebereme tato paradigmata a jazyky a dostaneme se také k otázkám gramatiky a překladu jazyků.

Kapitola 7 (Vývoj softwaru) představuje oblast informatiky označovanou jako softwarové inženýrství (software engineering), která se zabývá problémy souvisejícími s vývojem velkých softwarových systémů. Základní motiv kapitoly lze shrnout tak, že návrh velkých softwarových systémů je náročný úkol, který se potýká s jinými potížemi než tradiční inženýrství. Softwarové inženýrství proto tvoří důležitý obor informatického výzkumu. Čerpá přitom z natolik rozdílných zdrojů, jako je inženýrství, vedení projektů, personalistika, návrh programovacích jazyků nebo dokonce architektura.

V dalších dvou kapitolách se podíváme na způsoby, kterými lze organizovat data v počítačovém systému. V kapitole 8 (Datové abstrakce) představíme metody, které tradičně slouží k organizaci dat v hlavní paměti počítače, a poté budeme sledovat vývoj datových abstrakcí od koncepce primitivních funkcí po dnešní objektově orientované přístupy. V kapitole 9 (Databázové systémy) se budeme zabývat metodami, které se obvykle uplatňují při organizaci dat v hromadném úložišti počítače, a pokusíme se zjistit, jak se implementují mimořádně rozsáhlé a komplexní databázové systémy.

Kapitola 10 (Počítačová grafika) nastiňuje téma grafiky a animace, což je oblast, která se zabývá vytvářením a zobrazováním virtuálních světů. Oblast grafiky a animace, která vychází z pokroků klasičtějších částí informatiky (např. architektury počítačů, návrhu algoritmů, datových struktur a vývoje softwaru), vykazuje značné pokroky a v současnosti se rozvinula do vzrušujícího a dynamického oboru. Může také posloužit jako příklad působivých výsledků, jaké přináší kombinace různých prvků informatiky s jinými disciplínami (v tomto případě například s fyzikou, malířstvím či fotografií).

V kapitole 11 (Umělá inteligence) se dozvíme, že ve snaze o vývoj užitečnějších strojů hledá informatika poučení ve studiu lidské inteligence. Výzkumníci doufají, že když porozumí tomu, jak uvažuje a vnímá lidská mysl, dokáží navrhnout algoritmy, které tyto procesy napodobí a dokáží tedy příslušné schopnosti předat počítačům. Výsledkem je oblast informatiky označovaná jako umělá inteligence, která intenzivně využívá výzkumu v oblastech typu psychologie, biologie a lingvistiky.

Knihu uzavírá kapitola 12 (Teorie výpočtů), která zkoumá teoretické základy informatiky. Tato oblast umožňuje pochopit omezení algoritmů (a tedy i strojů). Identifikujeme zde některé problémy, které není možné řešit algoritmicky (a leží proto mimo možnosti počítačů). Také se dozvíme, že řešení některých jiných problémů klade takové nároky na čas nebo prostor, že jsou tyto problémy z praktického hlediska rovněž neřešitelné. Celá kniha tedy poskytne představu o rozsahu a omezeních algoritmických systémů.

V každé kapitole se budeme snažit o podrobný rozbor, který povede k hlubšímu pochopení tématu. Cílem je dosáhnout praktického zvládnutí informatiky, které umožní porozumět technické společnosti, v níž žijeme, a poskytnout základy, na kterých bude možné při studiu vědy a technologie dále stavět.

0.6: Společenské dopady

Pokrok v oblasti informatiky stírá četné ukazatele, podle kterých se naše společnost v minulosti rozhodovala, a zpochybňuje mnohé dlouho uznávané společenské zásady. Ve sféře práva vede k otázkám ohledně toho, do jaké míry může být duševní vlastnictví majetkem a jaká práva a závazky toto vlastnictví doprovázejí. Z hlediska etiky vyvstávají různé varianty, které narušují tradiční principy, na nichž je založeno společenské chování. V oblasti veřejné správy se objevují debaty týkající se rozsahu regulace počítačové technologie a jejích aplikací. Filozofové musí zvažovat rozpory mezi projevy inteligentního chování a přítomností skutečné inteligence. Společnost jako celek se pak nemůže shodnout, zda nové aplikace technologie znamenají nové svobody nebo nové možnosti kontroly.

I když tato témata přímo do informatiky nepatří, jsou důležitá pro osoby, které uvažují o své kariéře v oblasti informatiky nebo využití počítačů. Vědecké objevy často vedou ke kontroverzním aplikacím, které odmítají sami zúčastnění výzkumníci. Navíc se stává, že etické pochybení může rychle ohrozit i jinak úspěšnou kariéru.

Schopnost řešit dilemata, která jsou důsledky pokroků počítačové technologie, má značný význam i pro lidi mimo vlastní obor. Technologie totiž infiltruje společnost tak rychle, že málokdo si může zachovat nezávislost na jejích důsledcích.

Tento text poskytuje technické znalosti, které umožňují racionálně přistupovat k dilematům, jaká informatika přináší. Samotné technické znalosti oboru však neposkytují řešení všech souvisejících dilemat. S vědomím tohoto faktu budeme několik odstavců následujícího textu věnovat sociálním, etickým a právním otázkám. Patří k nim hlediska zabezpečení, nejasnosti týkající se vlastnictví softwaru a odpovědnosti za škody, společenské dopady databázové technologie a důsledky pokroků v oblasti umělé inteligence.

Na jednotlivé otázky přitom často neexistuje definitivní správná odpověď a mnoho přijatelných řešení představuje kompromisy mezi protikladnými (a často stejně zásadními) pohledy. Hledání řešení v těchto případech často vyžaduje schopnost naslouchat, uznat cizí stanoviska, vést rozumnou debatu a přizpůsobit vlastní názory s ohledem na nově zjištěné poznatky. Všechny kapitoly proto končí částí s názvem „Společenské otázky“, která zkoumá vztah mezi informatikou a společností. Na tyto otázky není nutné najít odpovědi. Jedná se spíše o náměty k úvahám. Odpověď, která může na první pohled vypadat samozřejmě, nás po prozkoumání alternativ často

přestane uspokojovat. Stručně řečeno, tyto otázky nemají čtenáře vést ke „správné" odpovědi, ale spíše je povzbudit k přemýšlení – mj. o různých osobách, na které má otázka vliv, o alternativách a o krátkodobých i dlouhodobých důsledcích těchto alternativ.

Závěrem této části představíme některé přístupy k etice, které kdysi navrhli významní filozofové při svém hledání základních principů, které by mohly vést lidské rozhodování a chování. Většinu z těchto teorií lze zařadit do kategorií etiky následků, etiky povinností, etiky společenské smlouvy a etiky ctností. Tyto teorie mohou nabídnout způsob, jak přistupovat k etickým otázkám naznačeným v textu. Můžeme si například všimnout, že různé teorie vedou k protikladným závěrům a odhalují tedy skryté alternativy.

Etiky následků se pokoušejí analyzovat problémy na základě následků, jaké má volba z různých možností. Nejvýznamnějším příkladem je utilitarismus. „Správná" rozhodnutí nebo akce se podle něj vyznačují tím, že zajišťují největší dobro pro co největší část společnosti. Na první pohled se zdá, že utilitarismus poskytuje spravedlivý způsob, jak řešit etická dilemata. Ve své vyhraněné formě však vede k mnoha nepřijatelným závěrům. Dovolil by například většině společnosti zotročit malou menšinu. Mnozí také argumentují, že přístupy k etickým teoriím založené na následcích, které ze své podstaty kladou důraz na následky činů, zpravidla považují lidi za pouhé prostředky a nikoli za jednotlivce s vlastní hodnotou. Tito kritikové dále prohlašují, že to představuje zásadní vadu všech etických teorií uvedeného typu.

Na rozdíl od etiky následků nehodnotí etiky povinností dopad rozhodnutí a činů, ale místo toho prohlašují, že členové společnosti mají jisté nepominutelné povinnosti či závazky. Na základě těchto povinností je možné následně odpovídat na etické otázky. Pokud například člověk přijme závazek respektovat práva jiných, musí odmítnout otroctví bez ohledu na důsledky tohoto svého postoje. Na druhou stranu odpůrci etik povinností argumentují, že tyto teorie nedokáží řešit problémy, kde dochází ke konfliktu různých povinností. Měli byste říci pravdu, i když tím zradíte důvěru svého kolegy? Měl by se napadený stát bránit, i když válka způsobí smrt mnoha jeho občanů?

Etická teorie společenské smlouvy vychází z hypotetické společnosti bez jakýchkoli etických základů. V tomto „přírodním stavu" je všechno dovoleno. V takové situaci se každý jedinec musí starat sám o sebe a neustále musí být na pozoru před agresí jiných. Etická teorie společenské smlouvy navrhuje, že za uvedených okolností by členové společnosti mezi sebou uzavřeli „smlouvy". Taková smlouva by například zněla: nebudu krást tvé věci, když nebudeš krást moje. Tyto „smlouvy" by pak utvořily základ, který by umožnil definovat etické chování. Všimněme si, že etická teorie společenské smlouvy dává motivaci pro etické chování – „etickou smlouvu" bychom měli dodržovat, protože bychom jinak vedli nepříjemný život. Názoroví oponenti etické teorie společenské smlouvy namítají, že neposkytuje dostatečně široký základ na řešení etických dilemat, protože nabízí vodítko pouze v těch případech, pro které byly uzavřeny smlouvy. (V situacích, které nepopisuje žádná existující smlouva, se mohu chovat jak chci.) Právě nové technologie mohou představovat nezmapované území, na které se stávající etické smlouvy nemusí vztahovat.

Etika ctností, kterou zastávali Platón a Aristoteles, prohlašuje, že „dobré chování" není založeno na uplatnění definovatelných pravidel, ale spíše přirozeným důsledkem „dobré povahy". Zatímco příznivci etik následků, etik povinností a etik společenské smlouvy doporučují, aby lidé řešili etická dilemata na základě otázky „Jaké jsou následky?", „Jaké mám povinnosti?" nebo „Jaké smlouvy jsem uzavřel?", zastánci

etiky ctností navrhují rozhodovat toto dilema pomocí otázky „Jakým člověkem chci být?“. Dobré chování tedy získáme budováním dobré povahy, která obvykle vychází z kvalitní výchovy a rozvoje mravných návyků.

Právě na etice ctností je založen přístup, který se obvykle uplatňuje při seznamování profesionálů různých oborů s etickými otázkami. Místo předkládání jednotlivých etických teorií se začíná od případových studií, které požadují odpověď na různé etické otázky z oboru daného odborníka. Při diskuzi o výhodách a nevýhodách jednotlivých řešení pak experti získávají lepší povědomí, přehled a citlivost ohledně nástrah, které číhají v jejich profesionálním životě, a mohou tímto způsobem rozvíjet svůj charakter. V tomto duchu předkládáme na konci každé kapitoly otázky týkající se společenských hledisek.

Společenské otázky

Následující otázky mají čtenáře provést etickými, společenskými a právními aspekty oboru počítačů. Cílem není jen odpovědět na jednotlivé otázky. Měli byste se také zamyslet nad tím, proč jste vybrali konkrétní odpověď a zda dokážete odpovědi na jednotlivé otázky konzistentně zdůvodnit.

1. Obecně se přijímá předpoklad, že naše společnost je *jiná*, než jaká by byla bez počítačové revoluce. Je naše společnost *lepší*, než kdyby k této revoluci nedošlo? Je naše společnost horší? Lišila by se vaše odpověď, pokud byste ve společnosti zastávali jinou pozici?
2. Je přijatelné, aby členové dnešní technické společnosti nevyvíjeli žádnou snahu porozumět principům použitých technologií? Mají například občané demokratického zřízení, jejichž hlasy často rozhodují o podpoře a nasazení určitých technologií, závazek snažit se dané technologie pochopit? Závisí vaše odpověď na tom, kterou technologii zvažujete? Odpovíte například stejně, jedná-li se o jadernou nebo počítačovou technologii?
3. Díky používání hotovosti při finančních transakcích mohli lidé tradičně spravovat své finanční záležitosti bez poplatků za služby. Jak se však současná ekonomika stále více automatizuje, finanční instituce zavádějí bankovní poplatky, které účtují za přístup k těmto automatizovaným systémům. Existuje určitý limit, kdy tyto poplatky nespravedlivě omezují přístup jednotlivce k ekonomickým transakcím? Předpokládejme například, že zaměstnavatel platí zaměstnancům pouze formou šeku, a všechny finanční instituce požadují za proplácení a ukládání šeků poplatky. Je to vůči zaměstnancům nespravedlivé? Jak se situace změní v případě, že zaměstnavatel trvá na platbách převodem na účet?
4. Do jaké míry by společnost poskytující interaktivní televizní služby měla mít možnost získávat informace od dětí (řekněme pomocí schématu interaktivní hry)? Lze například společnosti povolit, aby od dítěte zjišťovala, co nakupují jeho rodiče? Jaká bude odpověď v případě informací o samotném dítěti?
5. Nakolik by měly vlády regulovat počítačové technologie a jejich aplikace? Zamyslete se například nad problematikou z otázek 3 a 4. Co ospravedlňuje vládní regulaci?
6. Do jaké míry naše rozhodnutí týkající se obecně všech technologií a konkrétně počítačových technologií ovlivní budoucí generace?

7. V souvislosti s technologickými pokroky musí vzdělávací systém neustále přehodnocovat úroveň abstrakce, na které se vyučují jednotlivá témata. Mnohé otázky se vzájemně podobají: zda je určitá dovednost i nadále potřebná, nebo zda lze studentům dovolit, aby se spoléhali na abstraktní nástroj. Studenti trigonometrie se již neučí, jak hledat hodnoty trigonometrických funkcí v tabulkách. Místo toho příslušné hodnoty zjišťují pomocí abstraktních nástrojů, tj. v tomto případě kalkulaček. Někteří lidé se domnívají, že také písemné dělení by mělo ustoupit abstraktním metodám. Kolem kterých dalších témat se objevují podobné spory? Odstraňují moderní textové procesory nutnost učit se správnému pravopisu? Umožní časem technologie videa, aby se lidé obešli bez znalosti čtení?
8. Koncepce veřejných knihoven je založena hlavně na předpokladu, že všichni občané v demokracii musí mít přístup k informacím. Když se stále více informací ukládá a šíří pomocí počítačových technologií, stává se přístup k této technologii právem každého jednotlivce? Pokud ano, měly by veřejné knihovny sloužit jako kanál, který tento přístup zajistí?
9. Jaké etické otázky se objevují ve společnosti, která se spoléhá na použití abstraktních nástrojů? Existují případy, kdy je neetické pracovat s produktem nebo službou bez znalosti toho, jak funguje? Bez informací o tom, jak se produkuje? Nebo bez vědomostí o vedlejších důsledcích používání?
10. Jak se společnost stále více automatizuje, mohou vládní úřady snáze monitorovat aktivitu občanů. Je to dobře nebo špatně?
11. Které z technologií zmíněných v románu George Orwella *1984* se staly skutečností? Používají se způsobem, který Orwell předpovídal?
12. Pokud byste měli stroj času, ve které historické éře byste chtěli žít? Které současné technologie byste si chtěli vzít s sebou? Mohli byste si zvolené technologie vzít samostatně, aniž byste museli přibrat i jiné? Do jaké míry lze jednu technologii oddělit od dalších technologií? Je konzistentní protestovat proti globálnímu oteplování, ale přitom využívat moderní lékařskou péči?
13. Předpokládejme, že vaše práce vyžaduje, abyste se usadili v jiné kultuře. Budete se nadále řídit etickými pravidly kultury svého původu, nebo přijmete etiku hostitelské kultury? Závisí vaše odpověď na tom, zda se otázka týká stylu oblékání nebo přístupu k lidským právům? Které etické standardy by měly převážit, zůstanete-li žít v zemi svého původu, ale budete obchodovat s cizí kulturou?
14. Nestala se naše společnost příliš závislá na počítačových aplikacích pro obchodování, komunikaci a sociální interakci? Jaké důsledky by měl například dlouhodobý výpadek Internetu nebo služeb mobilních telefonů?
15. Většina smartphonů dokáže určit svou polohu pomocí systému GPS. Díky tomu mohou aplikace poskytovat informace závislé na aktuální poloze telefonu (jako například místní zprávy, předpovědi počasí nebo seznamy firem v nejbližším okolí). Tyto navigační funkce však mohou využít i jiné aplikace, které zveřejňují polohu telefonu třetím stranám. Je to dobře? Jak lze znalost polohy telefonu (a tedy jeho uživatele) zneužít?
16. Když vyhodnotíte své počáteční odpovědi na předchozí otázky, ke které etické teorii předložené v oddílu 0.6 máte nejblíže?

Další zdroje informací

Goldstine, J. J. *The Computer from Pascal to von Neumann* (Počítače od Pascala k von Neumannovi). Princeton: Princeton University Press, 1972.

Kizza, J. M. *Ethical and Social Issues in the Information Age* (Etické a společenské otázky v informačním věku), 3. vyd. Londýn: Springer-Verlag, 2007.

Kohák, E.: *Svoboda, svědomí, soužití: kapitoly z mezilidské etiky*, Sociologické nakladatelství 2010, ISBN 978-80-86429-35-9

Mollenhoff, C. R. Atanasoff: *Forgotten Father of the Computer* (Zapomenutý otec počítače). Ames: Iowa State University Press, 1988.

Neumann, P. G. *Computer Related Risks* (Rizika související s počítači). Boston, MA: Addison-Wesley, 1995.

Ni, L. *Smart Phone and Next Generation Mobile Computing* (Smartphone a mobilní počítače nové generace). San Francisco: Morgan Kaufmann, 2006.

Quinn, M. J. *Ethics for the Information Age* (Etika pro informační věk), 2. vyd. Boston, MA: Addison-Wesley, 2006.

Randell, B. *The Origins of Digital Computers* (Počátky digitálních počítačů), 3. vyd. New York: Springer-Verlag, 1982.

Spinello, R. A. a H. T. Tavani. *Readings in CyberEthics* (Čtení z kybernetické etiky), 2. vyd. Sudbury, MA: Jones and Bartlett, 2004.

Swade, D. *The Difference Engine* (Diferenční stroj). New York: Viking, 2000.

Tavani, H. T. *Ethics and Technology: Ethical Issues in an Age of Information and Communication Technology* (Etika a technologie: etické otázky v éře informační a komunikační technologie), 3. vyd. New York: Wiley, 2011.

Woolley, B. *The Bride of Science, Romance, Reason, and Byron's Daughter* (Nevěsta vědy, romantiky a rozumu a Byronova dcera). New York: McGraw-Hill, 1999.

KAPITOLA

1

Ukládání dat

V této kapitole se budeme zabývat tématy, která souvisejí s reprezentací dat a s jejich ukládáním v počítači. Budeme uvažovat o datech různého typu včetně textu, číselných hodnot, obrázků, zvuku a videa. Většina informací z této kapitoly platí i v jiných oborech, než je klasická informatika. Vztahují se například i na digitální fotografii, záznam a reprodukci zvuku či videa a dálkovou komunikaci.

**Hvězdičky označují doporučené volitelné části.*

Na začátku kurzu informatiky se zamyslíme nad tím, jak se informace v počítačích kódují a ukládají. Nejdříve rozebereme principy počítačových zařízení na ukládání dat a poté si ukážeme, jak lze zakódovat informace, které tyto systémy ukládají. Prozkoumáme různé aspekty dnešních systémů na ukládání dat a vysvětlíme, jak lze překonat jejich nedostatky pomocí metod typu komprese dat a kontroly chyb.

1.1: Bity a jejich ukládání

Informace v moderních počítačích jsou zakódovány jako posloupnosti nul a jedniček. Tyto hodnoty se nazývají **bity** (termín vznikl zkrácením anglického výrazu *binary digit* – binární číslice). Bity jsou obvykle považovány za číselné hodnoty, ale ve skutečnosti se jedná o pouhé symboly, jejichž význam závisí na použitém kontextu. Sekvence bitů sice někdy reprezentují číselné hodnoty, ale v jiných případech znaky abecedy a interpunkční symboly, obrázky nebo zvuky.

Logické operace

Chceme-li porozumět tomu, jak počítače ukládají jednotlivé bity a jak s nimi manipulují, je vhodné si představit, že bit 0 označuje hodnotu *nepravda* (false) a bit 1 znamená hodnotu *pravda* (true). Díky tomu můžeme zpracování bitů chápat jako manipulaci s hodnotami pravda a nepravda. Operace, které přijímají hodnoty pravda a nepravda, se označují jako **logické (boolovské) operace**. Jeden z jejich názvů připomíná, že pionýrem matematické oblasti s názvem logika byl George Boole (1815–1864). Mezi tři základní logické operace patří operace AND, OR a XOR (vylučovací nebo), které shrnuje tabulka na obrázku 1.1. Uvedené operace odpovídají aritmetickým operacím KRÁT a PLUS, protože kombinují dvojici hodnot (vstup operace) a poskytují třetí hodnotu (výstup). Oproti aritmetickým operacím však logické operace nepracují s číselnými hodnotami, ale s hodnotami pravda a nepravda.

Logická operace AND zrcadlí pravdivost nebo nepravdivost výrazu, který je vytvořen kombinací dvou menších (jednodušších) výrazů pomocí spojky *a*. Tyto výrazy mají obecný tvar

P AND *Q*

kde *P* označuje jeden výraz a *Q* zastupuje jiný výraz – např.:

Fík je maxipes AND Ája je holčička.

Vstupy operace AND představují pravdivost nebo nepravdivost komponent složeného výrazu. Výstup pak označuje pravdivost či nepravdivost samotného složeného výrazu. Výraz ve tvaru *P* AND *Q* je pravdivý pouze tehdy, jestliže jsou pravdivé obě jeho části. Můžeme tedy usoudit, že výraz 1 AND 1 má hodnotu 1, zatímco všechny ostatní varianty poskytnou hodnotu 0 (viz obrázek 1.1).

Operace OR (tj. „nebo“) je definována obdobně pomocí složených výrazů ve formátu

P OR *Q*

kde *P* opět reprezentuje jeden výraz a *Q* symbolizuje jiný výraz. Tyto výrazy jsou pravdivé tehdy, jestliže je pravdivá alespoň jedna z jejich komponent. To odpovídá znázornění operace OR na obrázku 1.1.

Význam operace XOR nelze vyjádřit jedinou českou spojkou. Operace XOR poskytuje výstup 1 (pravda), pokud má jeden z jejích vstupů hodnotu 1 (pravda) a druhý 0 (nepravda). Například výraz ve tvaru *P* XOR *Q* znamená „buď *P*, anebo *Q*, ale nikoli obojí". (Stručně řečeno, operace XOR dává výstup 1, jestliže jsou její vstupy různé. Jde o „výlučné nebo".)

Další logickou operací je operace NOT. Od operací AND, OR a XOR se liší tím, že má pouze jeden vstup. Její výstup je opakem příslušného vstupu. Pokud je na vstupu operace NOT pravda, výstup má hodnotu nepravda a naopak. Pokud je tedy na vstupu operace NOT pravdivost či nepravdivost výrazu

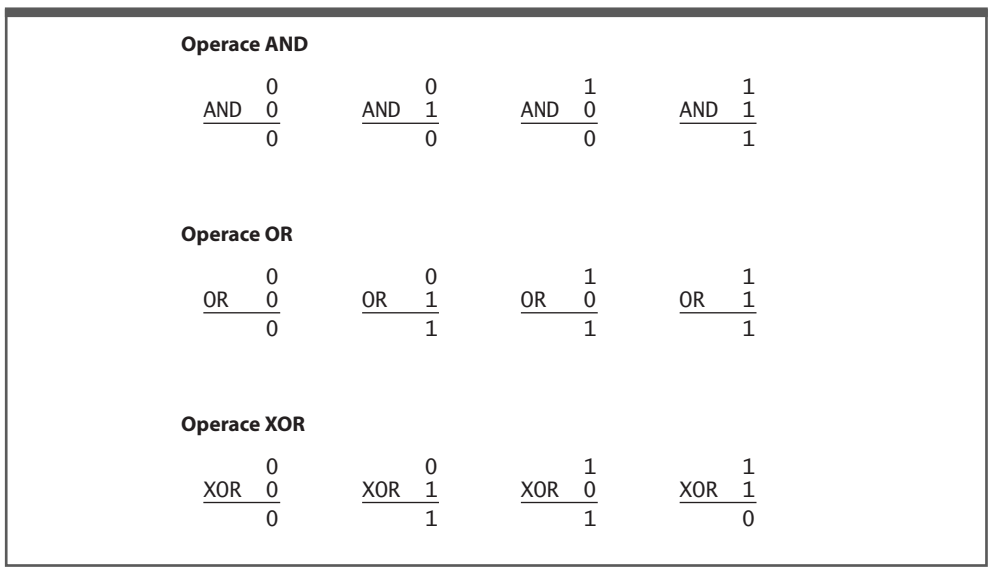
Jonatán je pes.

pak výstup bude označovat pravdivost nebo nepravdivost výrazu

Jonatán není pes.

Hradla a klopné obvody

Zařízení, které poskytuje výstup logické operace, když na ně přivedeme vstupní hodnoty této operace, se nazývá **hradlo** (gate). Hradla je možné konstruovat různými technologiemi, například jako převody, relé nebo optické prvky. V dnešních počítačích se hradla obvykle implementují jako malé elektronické obvody, v nichž jsou číslice 0 a 1 reprezentovány jako úrovně napětí. Těmito podrobnostmi se však nemusíme zabývat. Pro naše účely postačí znázorňovat hradla symbolicky, jak vidíte na obrázku 1.2. Všimněme si, že hradla AND, OR, XOR a NOT mají odlišný tvar svých symbolů. Jejich vstupní hodnoty přitom směřují z jedné strany a výstupní hodnoty vycházejí z druhé strany.

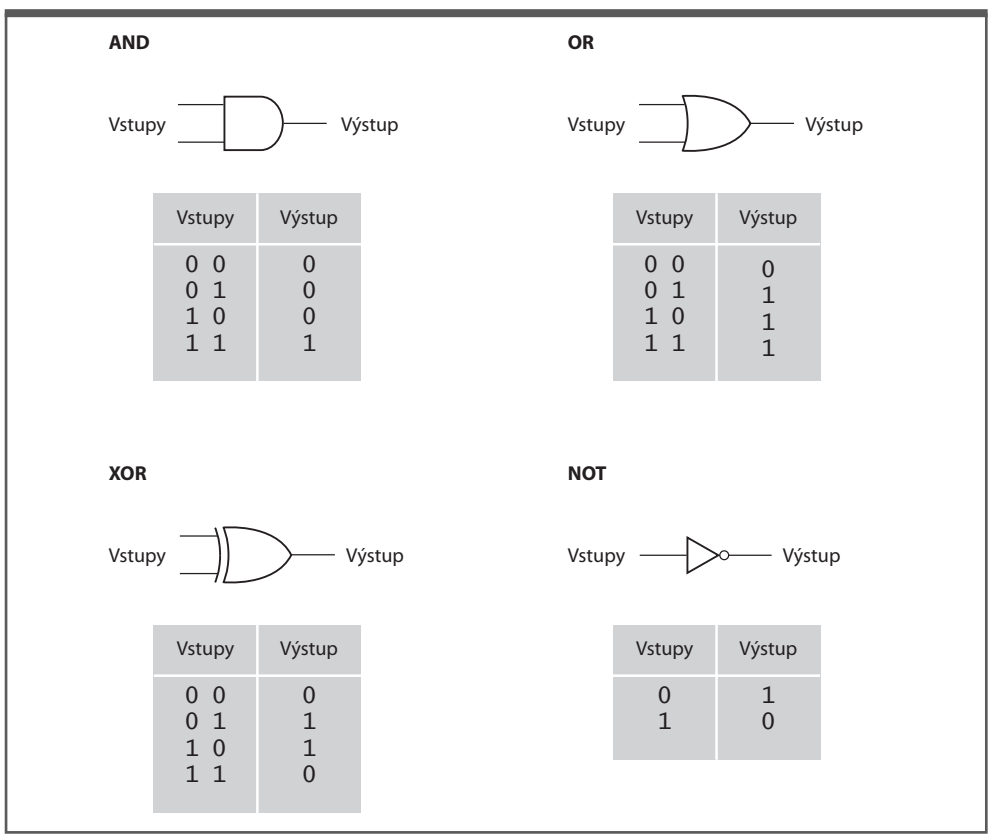


Obrázek 1.1: Logické operace AND, OR a XOR (výlučné nebo)

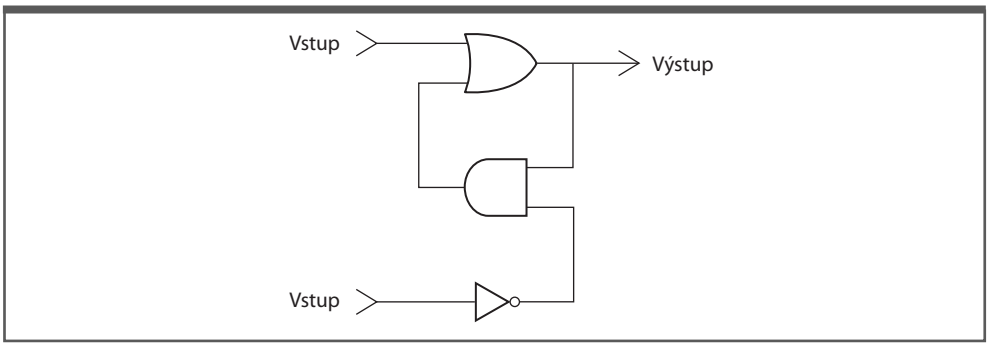
Hradla slouží jako stavební kameny při konstrukci počítačů. Jeden důležitý krok v tomto směru znázorňuje obvod na obrázku 1.3. Jedná se o příklad skupiny obvodů, které se označují jako **klopné obvody** (flip-flop). Klopný obvod produkuje výstupní hodnotu 0 nebo 1. Tato hodnota zůstává konstantní, dokud nedojde ke změně

na jinou hodnotu na základě impulzu (což je dočasná změna signálu na hodnotu 1 s návratem k hodnotě 0) z jiného obvodu. Výstup tedy pod vlivem externího impulzu osciluje mezi dvěma hodnotami. Dokud si oba vstupy obvodu na obrázku 1.3 zachovávají hodnotu 0, výstup (ať už 0 či 1) se nemění. Jestliže se však na horním vstupu dočasně objeví hodnota 1, vynutí změnu výstupu na hodnotu 1, zatímco dočasné nastavení hodnoty 1 na dolním vstupu zaručí, že výstup bude mít hodnotu 0.

Analyzujme toto chování podrobněji. Aniž bychom znali aktuální výstup okruhu na obrázku 1.3, předpokládejme, že horní vstup se změní na hodnotu 1 a dolní vstup si ponechá hodnotu 0 (obrázek 1.4a). V důsledku toho se výstup hradla OR změní na 1 nezávisle na jiném vstupu tohoto hradla. Z toho vyplývá, že hodnota obou vstupů hradla AND se bude nyní rovnat 1, protože druhý vstup tohoto hradla již tuto hodnotu má (výstup poskytuje hradlo NOT vždy, když je na dolním vstupu klopného obvodu hodnota 0). Výstup hradla AND je poté nastaven na hodnotu 1, což znamená, že druhý vstup hradla OR bude mít nyní hodnotu 1 (obrázek 1.4b). Tím je zajištěno, že se výstup hradla OR bude trvale rovnat 1, i když se horní vstup klopného obvodu změní zpět na hodnotu 0 (obrázek 1.4c). Shrnuto: na výstupu klopného obvodu se objevila hodnota 1 a tato výstupní hodnota zůstane v platnosti i poté, kdy se hodnota horního vstupu změní na 0.

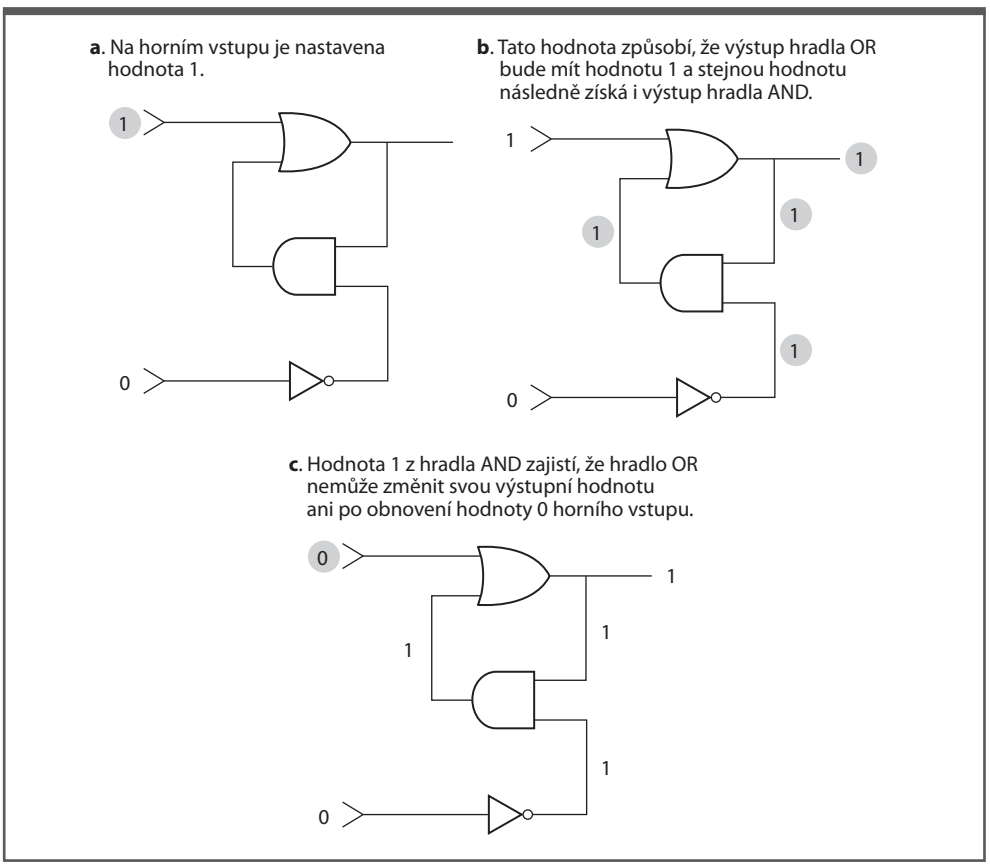


Obrázek 1.2: Grafické znázornění hradel AND, OR, XOR a NOT spolu s jejich vstupními a výstupními hodnotami



Obrázek 1.3: Jednoduchý klopný obvod

Podobně platí, že dočasné nastavení hodnoty 1 pro dolní vstup vynutí hodnotu 0 na výstupu klopného obvodu. Tento výstup pak zůstane zachován i po návratu vstupní hodnoty 0.



Obrázek 1.4: Nastavení výstupu klopného obvodu na hodnotu 1

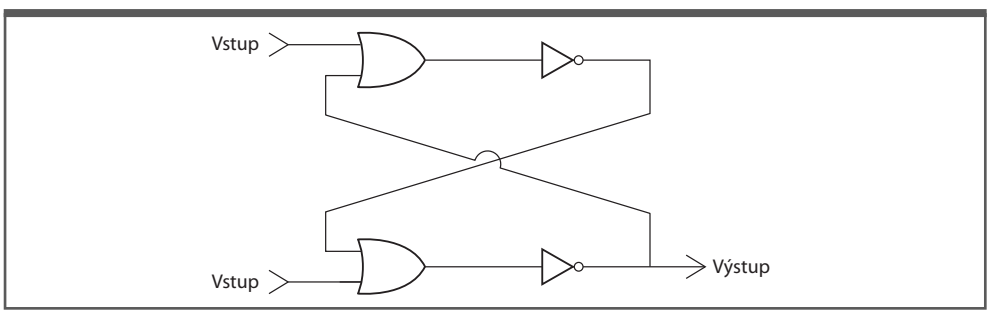
Klopné obvody na obrázcích 1.3 a 1.4 jsme si představili ze tří důvodů. V prvé řadě ukazují, jak lze z hradel sestavovat zařízení. Tento proces se označuje jako návrh digitálních obvodů a jedná se o důležité téma v oboru konstrukce počítačů. Ve skutečnosti představuje klopný obvod pouze jeden z mnoha obvodů, které se používají jako základní součásti při konstrukci počítačů.

Za druhé: koncepce klopného obvodu může posloužit jako příklad abstrakce a použití abstraktních nástrojů. Ve skutečnosti existují i další způsoby, jak klopný obvod zkonstruovat. Jedna alternativa je znázorněna na obrázku 1.5. Budete-li s tímto obvodem experimentovat, zjistíte, že navzdory své odlišné vnitřní struktuře má stejné externí vlastnosti jako obvody na obrázku 1.3. Konstruktér počítačů nemusí vědět, který obvod je v určitém klopném obvodě skutečně zapojen. Pokud chce klopný obvod použít jako abstraktní nástroj, stačí mu znát jeho externí chování. Klopný obvod spolu s dalšími definovanými obvody tvoří sadu stavebních kamenů, z nichž může konstruktér sestavovat složitější schémata. Návrh počítačových obvodů se proto vyznačuje hierarchickou strukturou, kde každá úroveň využívá součásti nižší úrovně jako abstraktní nástroje.

Klopné obvody jsme si představili ještě z třetího důvodu: spolu s jinými metodami umožňují v moderních počítačích ukládat bity. Přesněji řečeno, klopný obvod lze nastavit tak, aby měl výstupní hodnotu 0 nebo 1. Jiné obvody mohou tuto hodnotu upravit odesláním impulzů na vstupy klopného obvodu. Další obvody, které používají výstup klopného obvodu jako svůj vstup, pak mohou na uloženou hodnotu reagovat. Sady klopných obvodů, které jsou vytvořeny z velmi malých elektrických prvků, proto mohou v počítači zaznamenávat informace, které jsou zakódovány jako posloupnosti nul a jedniček. V praxi se uplatňuje technologie **VLSI (very large-scale integration, vysoká integrace)**, která umožňuje na jedné polovodičové destičce (zvané **čip**) sestavit miniaturní zařízení, která obsahují miliony klopných obvodů spolu s jejich řídicími obvody. Tyto čipy se následně uplatňují jako abstraktní nástroje při konstrukci počítačových systémů. V některých případech dovoluje technologie VLSI vytvořit na jediném čipu dokonce celý počítačový systém.

Zápis čísel v šestnáctkové soustavě

Analyzujeme-li vnitřní fungování počítače, musíme se zabývat posloupnostmi bitů, které se nazývají řetězce. Některé z nich mohou být značně dlouhé. Dlouhá sekvence bitů se často označuje jako **datový proud** (stream). Datové proudy jsou bohužel pro člověka těžko srozumitelné. Pouhé přepsání posloupnosti číslic 101101010011 je pracné a nese značné riziko chyb. Kvůli jednodušší reprezentaci takových bitových řad se proto často uplatňuje zkrácený zápis, který se označuje jako **zápis v šestnáctkové soustavě** (hexadecimal notation, šestnáctkový zápis). Využívá toho, že délka posloupností bitů v počítačích je obvykle násobkem čtyř. Šestnáctkový zápis nahrazuje jediným symbolem kombinace čtyřech bitů. Řetězec dvanácti bitů lze například vyjádřit pomocí třech šestnáctkových symbolů.



Obrázek 1.5: Jiný způsob konstrukce klopného obvodu

Bitové schéma	Šestnáctková reprezentace
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

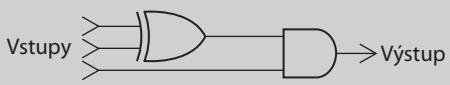
Obrázek 1.6: Systém šestnáctkového kódování

Obrázek 1.6 představuje systém šestnáctkového kódování. V levém sloupci jsou uvedeny všechny možné kombinace o délce čtyř bitů a v pravém sloupci nalezneme symboly, které odpovídají příslušné kombinaci bitů v šestnáctkové soustavě. Sekvenci bitů 10110101 lze v tomto systému vyjádřit jako B5. Výsledek dostaneme tak, že rozdělíme posloupnost bitů na dílčí řetězce dlouhé čtyři bity a každý z nich poté převedeme na jeho šestnáctkový ekvivalent: 1011 odpovídá symbolu B a 0101 můžeme zapsat jako 5. Tímto způsobem můžeme řadu 16 bitů 1010010011001000 zmenšit na čitelnější výraz A4C8.

Šestnáctkovou notaci budeme často používat v další kapitole a přitom oceníme její efektivitu.

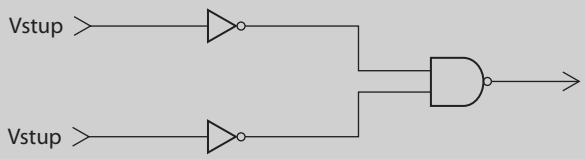
Otázky a cvičení

1. Jaká posloupnost bitů na vstupu zajistí, že následující obvod poskytne výstup 1?

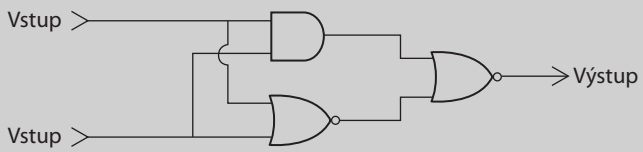


2. V textu jsme uvedli, že pokud se na dolním vstupu klopného obvodu z obrázku 1.3 objeví hodnota 1 (zatímco na horním vstupu zůstane hodnota 0), výstup se změní na hodnotu 0. Popište posloupnost dějů, ke kterým zde v klopném obvodu dochází.
3. Předpokládejme, že oba vstupy klopného obvodu na obrázku 1.5 mají hodnotu 0. Popište posloupnost dějů, které nastávají, když je horní vstup dočasně nastaven na hodnotu 1.

4. **a.** Pokud je výstup hradla AND směrován přes hradlo NOT, vypočítá tato kombinace logickou operaci označovanou jako NAND, která dává výstup 0 pouze tehdy, mají-li oba její vstupy hodnotu 1. Symbol hradla NAND se podobá hradlu AND, ale na výstupu má kroužek. Následující obvod obsahuje hradlo NAND. Jakou logickou operaci tento obvod vyhodnocuje?



b. Jestliže je výstup hradla OR směrován přes hradlo NOT, vyčíslí tato kombinace logickou operaci označovanou jako NOR. Tato operace poskytuje výstup 1 pouze tehdy, mají-li oba její vstupy hodnotu 0. Symbol hradla NOR odpovídá symbolu hradla OR s tím rozdílem, že má na výstupu kroužek. Následující obvod obsahuje hradlo AND a dvě hradla NOR. Jakou logickou operaci tento obvod vyhodnocuje?



5. Vyjádřete v šestnáctkové notaci následující posloupnosti bitů:
   a. 0110101011110010
   b. 111010000101010100010111
   c. 01001000
6. Jaké sekvence bitů zastupují tyto šestnáctkové výrazy?
   a. 5FD97 **b.** 610A **c.** ABCD **d.** 0100

1.2: Operační paměť

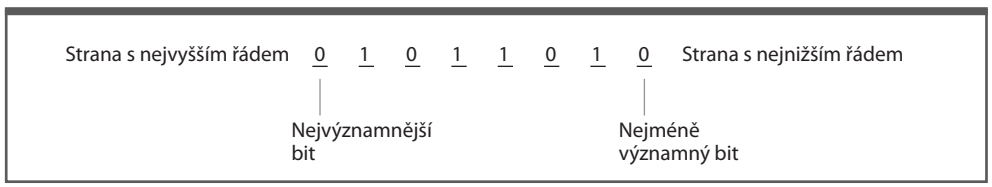
Aby mohl počítač uchovávat data, obsahuje sadu mnoha obvodů (např. klopných obvodů), z nichž každý dokáže uložit jediný bit. Tento zásobník bitů se označuje jako **operační paměť** počítače (uvidíte také označení **operační paměť**).

Organizace paměti

Operační paměť počítače je uspořádána do řiditelných jednotek zvaných **buňky**. Typická buňka má přitom velikost osmi bitů. (Řetězec osmi bitů představuje jeden **bajt**. Paměťové buňky tedy zpravidla mívají kapacitu jednoho bajtu.) Paměť malých počítačů, které se používají v domácích spotřebičích, jako jsou mikrovlnné trouby, se může skládat pouze z několika stovek buněk. Operační paměti velkých počítačů mohou oproti tomu zahrnovat miliardy buněk.

V počítači sice nemají smysl pojmy „vlevo“ ani „vpravo“, ale obvykle si představujeme, že bity uvnitř paměťové buňky jsou uspořádány do řady. Levý konec této řady se označuje jako **strana s nejvyšším řádem** (high-order end) a pravý konec se nazývá

strana s nejnižším řádem (low-order end). Bit úplně vlevo se označuje jako bit s nejvyšším řádem nebo **nejvýznamnější bit**. Vycházíme přitom z toho, že pokud bychom obsah buňky považovali za číselnou hodnotu, jednalo by se o nejvýznamnější číslici daného čísla. Obdobně platí, že zcela vpravo se nachází bit s nejnižším řádem neboli **nejméně významný bit**. Obsah paměťové buňky s kapacitou jednoho bajtu tedy můžeme reprezentovat tak, jak je znázorněno na obrázku 1.7.

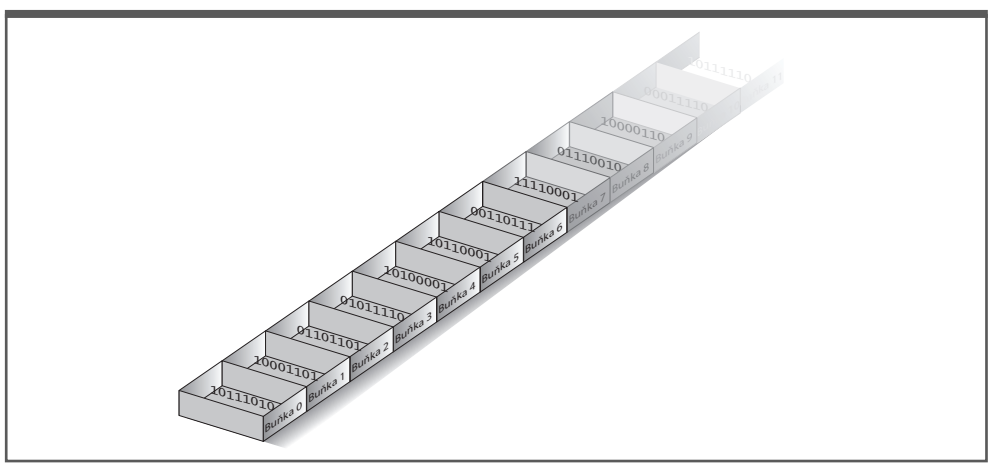


Obrázek 1.7: Organizace paměťové buňky velikosti jednoho bajtu

Aby bylo možné identifikovat jednotlivé buňky v operační paměti počítače, má každá buňka přiřazen jedinečný „název", kterému říkáme **adresa**. Celý systém připomíná schéma poštovních adres. Adresy paměťových buněk však obsahují pouze čísla. Přesněji řečeno, můžeme si představit, že všechny buňky leží v jediné řadě za sebou a v tomto pořadí dostávají čísla počínaje nulou. Tento systém adresování nejen umožňuje jednoznačně identifikovat každou buňku, ale navíc umisťuje buňky do konkrétního pořadí (obrázek 1.8). Lze proto používat výrazy jako „následující buňka" nebo „předchozí buňka".

Z toho, že je určeno pořadí buněk v operační paměti i bitů v rámci každé buňky, vyplývá důležitý závěr: celá kolekce bitů v operační paměti počítače je v zásadě uspořádána do jediné dlouhé řady. Části této dlouhé posloupnosti proto umožňují ukládat sekvence bitů, které jsou delší než jediná buňka. Například řetězec 16 bitů můžeme snadno umístit do dvou paměťových buněk ležících za sebou.

Aby byla operační paměť počítače úplná, je potřeba obvody uchovávající jednotlivé bity zkombinovat s obvody, které umožňují jiným prvkům počítače ukládat a načítat data v paměťových buňkách. Díky tomu mohou jiné komponenty získat data z paměti tak, že se elektronicky dotážou na obsah určité adresy (hovoříme o operaci čtení), nebo mohou zaznamenat informace do paměti, když požádají o umístění určité posloupnosti bitů do buňky na jisté adrese (tato operace se označuje jako zápis).



Obrázek 1.8: Paměťové buňky uspořádané podle adresy

Díky tomu, že je operační paměť počítače rozdělena na jednotlivé adresovatelné buňky, lze k těmto buňkám podle potřeby přistupovat nezávisle. Protože umožňuje přistupovat k buňkám v libovolném pořadí, označuje se operační paměť počítače často jako **paměť s náhodným přístupem** (random access memory – RAM). Touto možností přístupu do kteréhokoli místa se operační paměť zásadně liší od systémů hromadného úložiště, kterými se budeme zabývat v další části kapitoly. Tyto systémy totiž manipulují s bity pouze v nedělitelných blocích.

V předchozím textu jsme ukázali, jak lze ukládat bity pomocí klopných obvodů. Paměť RAM většiny moderních počítačů je však založena na jiných technologiích, které umožňují dosáhnout vyššího stupně miniaturizace a rychlejší doby odezvy. Mnohé z těchto technologií ukládají bity pomocí drobných elektrických nábojů, které se mohou rychle rozptýlit. Taková zařízení proto vyžadují dodatečné obvody označované jako obnovovací, aby bylo možné náboje mnohokrát za sekundu doplnit. Vzhledem k nestálosti se počítačová paměť využívající tuto technologii často nazývá **dynamická paměť** a odtud se odvozuje termín **DRAM** (vyslovuje se „dý–ram"), který znamená dynamickou paměť RAM. Někdy se také setkáme s termínem **SDRAM** (vyslovuje se „es--dý-ram"), který označuje synchronní paměť DRAM. Tento typ paměti DRAM zkracuje pomocí dalších metod dobu, která je potřebná k načítání obsahu paměťových buněk.

Měření kapacity paměti

Jak se dozvíme v další kapitole, systémy operační paměti je praktické navrhovat tak, aby se celkový počet jejich buněk rovnal mocnině dvou. Velikost paměti raných počítačů se proto často měřila v násobcích 1 024 paměťových jednotek (jedná se o 2^{10}). Hodnota 1 024 není daleko od čísla 1 000 a počítačová komunita proto pro uvedenou jednotku přijala předponu *kilo*. Kapacita 1 024 bajtů se proto začala označovat termínem *kilobajt* (zkráceně KB). O počítači s 4 096 paměťovými buňkami jsme tedy mohli říci, že má 4 KB paměti (4 096 = 4 × 1 024). Jak se paměti zvětšovaly, terminologie se postupně obohatila o MB (megabajt), GB (gigabajt) a TB (terabajt). Toto použití předpon *kilo-*, *mega-* apod. bohužel narušuje zavedenou terminologii, protože v jiných oblastech již označují jednotky, které jsou mocninami tisíců. Jednotka *kilometr* například označuje vzdálenost 1 000 metrů a jednotka *megahertz* při měření rádiových frekvencí znamená 1 000 000 hertzů. Při práci s uvedenými termíny je proto nutné dbát opatrnosti. Platí obecné pravidlo, že v kontextu počítačové paměti se termíny *kilo-*, *mega-* atd. vztahují k mocninám dvou, zatímco v jiných kontextech se týkají mocnin tisíců.

Otázky a cvičení

1. Pokud paměťová buňka s adresou 5 obsahuje hodnotu 8, jaký je rozdíl mezi zápisem hodnoty 5 do buňky číslo 6 a přesunem obsahu buňky číslo 5 do buňky číslo 6?
2. Předpokládejme, že chcete zaměnit hodnoty uložené v paměťových buňkách 2 a 3. Co je špatně na následujícím postupu:

   Krok 1: Přesuňte obsah buňky číslo 2 do buňky číslo 3.
   Krok 2: Přesuňte obsah buňky číslo 3 do buňky číslo 2.
   Navrhněte pořadí kroků, které správně zamění obsahy těchto buněk. V případě potřeby můžete použít další buňky.
3. Kolik bitů bude obsahovat paměť počítače se 4 KB paměti?

1.3: Hromadné úložiště

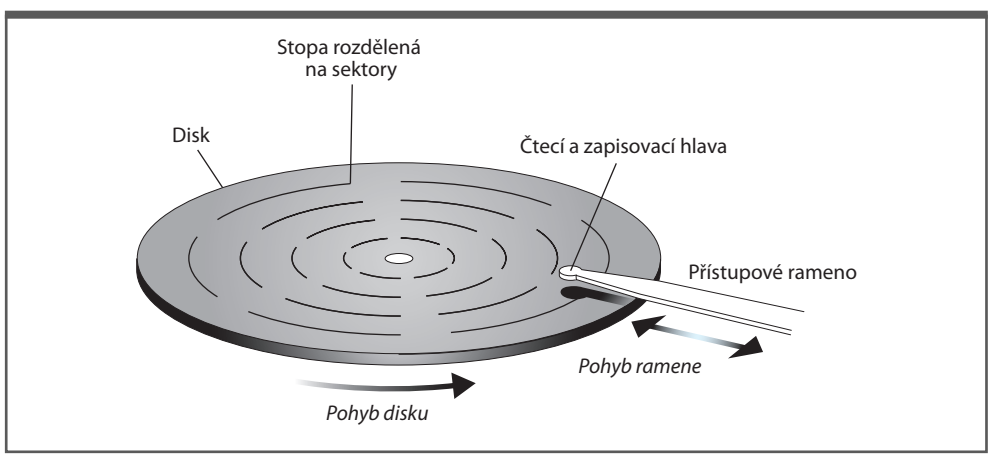
Kvůli nedostatečné trvanlivosti operační paměti a její omezené velikosti je většina počítačů vybavena dodatečnými paměťovými zařízeními, která se označují jako **hromadná úložiště** (neboli sekundární úložiště). Patří k nim magnetické disky, jednotky CD, DVD, magnetické pásky a jednotky flash (všemi uvedenými zařízeními se budeme zakrátko zabývat). K výhodám hromadných úložišť ve srovnání s operační pamětí patří vyšší stabilita dat, větší úložná kapacita a nízká cena. V mnoha případech lze také kvůli archivaci vyjmout úložné médium z počítače.

K popisu zařízení, která lze k počítači připojit a opět od něj odpojit, se často používají termíny *online* a *offline*. **Online** znamená, že zařízení nebo informace jsou připojené a z hlediska počítače snadno dostupné bez zásahu člověka. Termín **offline** oproti tomu popisuje situaci, kdy počítač může s příslušným zařízením nebo informacemi pracovat teprve díky operátorovi – například proto, že je zařízení potřeba zapnout, nebo je nutné do jednotky vložit médium, kde jsou informace uloženy.

Hlavní nevýhoda hromadných úložišť spočívá v tom, že jsou obvykle založena na mechanickém pohybu. Proto vyžadují k ukládání a načítání dat mnohem více času než operační paměť počítače, která všechny operace realizuje elektronicky.

Magnetické systémy

Oblasti hromadného ukládání dat po mnoho let dominovala magnetická technologie. V současnosti je nejvíce rozšířena koncepce **magnetického disku**, kdy jsou data uložena na tenkém otočném disku s magnetickým povlakem (obrázek 1.9). Nad diskem a pod ním jsou umístěny čtecí a zapisovací hlavy. Když se tedy disk otáčí, každá hlava opisuje kružnici označovanou jako **stopa** (track). Posunutím čtecích a zapisovacích hlav je možné přistupovat k různým soustředným stopám. Systémy diskového úložiště často zahrnují několik disků namontovaných na společné hřídeli tak, aby bylo mezi jednotlivými plotnami dostatek místa pro zasunutí čtecích a zapisovacích hlav. V těchto zařízeních se čtecí a zapisovací hlavy pohybují společně. Při každé změně polohy čtecích a zapisovacích hlav jsou zpřístupněny jiné sady stop, které se obecně nazývají **cylindry**.



Obrázek 1.9: Systém diskového úložiště

Stopa může obsahovat více informací, než kolik je obvykle potřeba zpracovat najednou. Proto se každá stopa dělí na malé oblouky zvané **sektory**, v nichž jsou data zaznamenána jako nepřerušený řetězec bitů. Všechny sektory na disku obsahují stejný počet bitů (obvyklé kapacity kolísají mezi 512 bajty a několika málo kilobajty). U nejjednodušších systémů diskového úložiště se každá stopa skládá ze stejného počtu sektorů. Bity v rámci sektoru na stopě blízko vnější hrany disku jsou proto uloženy méně kompaktně než bity na stopách blízko osy, protože vnější stopy jsou delší než vnitřní. V systémech diskového úložiště s vysokou kapacitou proto stopy nedaleko od vnější hrany mohou zahrnovat značně více sektorů než stopy blízko středu disku. Tato vlastnost se často využívá díky technologii, která se označuje jako **ZBR** (zoned-bit recording, zónový záznam bitů). Technologie ZBR označuje několik sousedních stop jako zóny. Typický disk přitom obsahuje přibližně deset zón. Všechny stopy v rámci zóny mají stejný počet sektorů, ale každá zóna má více sektorů na jednu stopu než zóny, které se nacházejí blíže ke středu disku. Tím lze dosáhnout efektivního využití celého povrchu disku. Bez ohledu na podrobnosti se systém diskového úložiště skládá z mnoha jednotlivých sektorů, z nichž ke každému je možné přistupovat jako k nezávislému řetězci bitů.

Umístění stop a sektorů netvoří trvalou součást fyzické struktury disku. Místo toho jsou tyto prvky vyznačeny magneticky během procesu, který se nazývá **formátování** (neboli inicializace) disku. Tento proces obvykle provádí výrobce disku, který dodává tzv. formátované disky. Stejnou operaci dokáže provést i většina počítačových systémů. Když je tedy informace o formátování disku poškozena, lze disk naformátovat znovu, ačkoli tento proces zničí všechny informace, které byly na disku zapsány dříve.

Kapacita systému diskového úložiště závisí na počtu použitých ploten a na hustotě, se kterou jsou na disku rozmístěny stopy a sektory. Systémy s menší kapacitou mohou obsahovat pouze jedinou plotnu. Diskové systémy s vysokou kapacitou, které dokáží uchovávat mnoho gigabajtů nebo dokonce terabajtů, se obvykle skládají ze tří až šesti ploten nasazených na společné hřídeli. Data lze navíc ukládat na horním i dolním povrchu každé plotny.

Při hodnocení výkonu diskových systémů se používá několik parametrů: (1) **doba vyhledání** (seek time – čas potřebný k přesunu čtecích a zapisovacích hlav z jedné stopy na další); (2) **rotační zpoždění** (rotation delay) neboli **doba latence** (latency time – polovina času, který disk potřebuje k úplnému otočení, tj. průměrná doba, za kterou se požadovaný datový záznam otočí pod čtecí a zapisovací hlavy poté, co je hlava umístěna nad příslušnou stopu); (3) **přístupová doba** (access time – součet doby vyhledání a rotačního zpoždění); a (4) **přenosová rychlost** (transfer rate – rychlost, s jakou lze data přenášet na disk a z disku). (Při nasazení technologie ZBR přenáší čtecí a zapisovací hlava během jednoho otočení disku větší objem dat u stop ve vnější zóně než u stop ve vnitřní zóně. Rychlost přenosu dat se proto mění v závislosti na části disku, která se aktuálně používá.)

Přístupová doba a přenosová rychlost závisí na rychlosti, se kterou se diskový systém otáčí. Aby bylo možné zvýšit rychlost rotace, nedotýkají se čtecí a zapisovací hlavy těchto systémů disku, ale místo toho se „vznášejí“ těsně nad jeho povrchem. Mezera je tak úzká, že se mezi hlavu a povrch disku může zaklínit dokonce i jediná částice prachu, která dokáže zničit jak hlavu, tak i povrch s daty (tento jev se označuje jako havárie hlavy). Diskové systémy jsou proto obvykle již z výroby uzavřeny v neprodyšných pouzdrech. Díky této konstrukci mohou diskové systémy rotovat rychlostí několika tisíc otáček za minutu a dosáhnout přenosových rychlostí, které se měří v MB za sekundu.

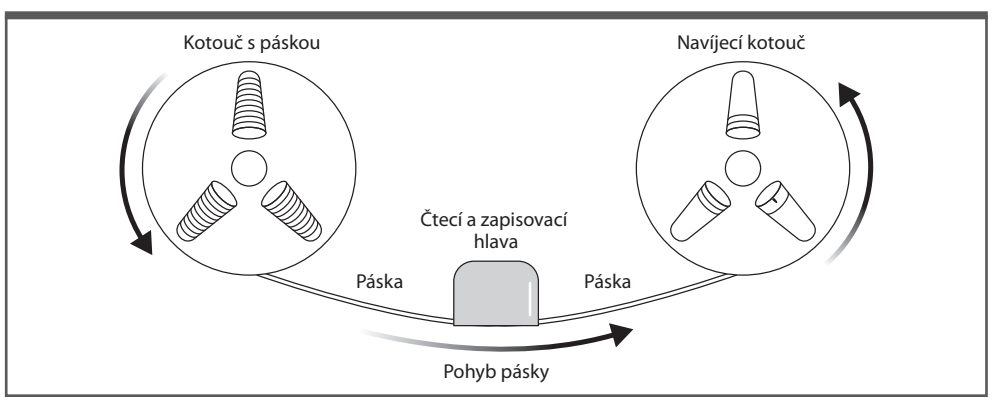
Části diskových systémů se při své činnosti musí fyzicky pohybovat. Tyto systémy se proto nemohou vyrovnat rychlostem elektronických obvodů. Doba zpoždění v elektronických obvodech se měří v nanosekundách (miliardtinách sekundy), zatímco doba vyhledání, doba latence a přístupová doba diskových systémů se pohybuje v milisekundách (tisícinách sekundy). Z hlediska elektronického obvodu, který čeká na výsledek, tedy může čas potřebný k načtení informací z diskového systému vypadat jako celá věčnost.

Diskové úložné systémy nejsou jediná zařízení hromadného úložiště, která využívají magnetickou technologii. Starší variantu hromadného úložiště založeného na magnetické technologii představuje **magnetická páska** (obrázek 1.10). V těchto systémech se informace zaznamenávají na magnetický povlak tenké plastové pásky, která je při uložení navinuta na kotouči. Pokud je potřeba získat přístup k datům, je kotouč vložen do zařízení označovaného jako pásková jednotka, které obvykle podle instrukcí počítače dokáže pásku číst, zapisovat na ni a převíjet ji. Páskové jednotky mohou mít různou velikost od malých kazetových jednotek, které se nazývají streamery (pracují s páskou podobného vzhledu jako stereosystémy), až po starší jednotky s velkými kotouči. Kapacita těchto zařízení sice závisí na použitém formátu, ale většina z nich dokáže uskladnit mnoho GB.

Hlavní nevýhoda magnetické pásky spočívá v tom, že přesun mezi různými pozicemi na pásce může trvat značně dlouho. Z jednoho kotouče na druhý je totiž často potřeba převinout dlouhý úsek pásky. Páskové systémy proto mají mnohem delší přístupovou dobu k datům než systémy magnetických disků, u kterých lze získat přístup k odlišným sektorům jen krátkým pohybem čtecí a zapisovací hlavy. Páskové systémy se tedy nehodí pro ukládání dat online. Technologie magnetických pásek je omezena hlavně na archivní ukládání dat offline, kde se s výhodou uplatní její vysoká kapacita, spolehlivost a ekonomičnost. Díky pokroku alternativních technologií, jako jsou disky DVD a jednotky flash, jsou však magnetické pásky i v jejich posledních aplikacích rychle nahrazovány.

Optické systémy

Jiná třída systémů hromadného úložiště je založena na optické technologii. Jako příklad lze uvést **kompaktní disk (CD)**. Tyto disky mají průměr 12 cm a vyrábějí se z odrazného materiálu, který je pokrytý průhledným ochranným povlakem. Informace se na tyto disky zaznamenávají formou změn jejich odrazných povrchů. Tyto informace je poté možné načítat pomocí laserového paprsku, který detekuje nepravidelnosti odrazného povrchu disku CD během jeho otáčení v mechanice.



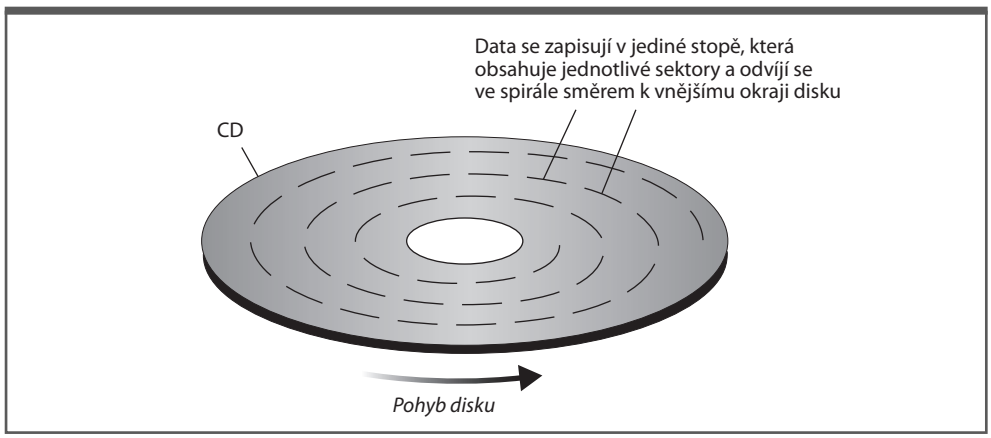
Obrázek 1.10: Mechanismus ukládání na magnetickou pásku

Technologie CD zpočátku sloužila jen k záznamu zvuku, který využíval formát záznamu označovaný jako **CD-DA (compact disk-digital audio)**. Disky CD používané v současnosti k ukládání počítačových dat pracují prakticky se stejným formátem. Informace na těchto discích CD jsou uloženy v jediné stopě, která se vine jako spirála kolem středu disku obdobně jako u starých gramofonových desek. Na rozdíl od klasických vinylových desek však stopa na disku CD probíhá od středu k okraji disku (obrázek 1.11). Tato stopa se dělí na jednotky označované jako sektory. Každý ze sektorů má vlastní identifikátor a dokáže uložit 2 KB dat, které u záznamu hudby odpovídají 1/75 sekundy.

Jedna otočka spirálové stopy poblíž vnějšího okraje disku je pochopitelně delší než ve vnitřní části disku. Aby bylo možné kapacitu disku využít co nejlépe, ukládají se informace po celé spirálové stopě s konstantní lineární hustotou. To znamená, že smyčka při vnějším konci spirály uchovává více informací než smyčka ve vnitřní části. Z toho vyplývá, že když laserový paprsek skenuje vnější část spirálové stopy, přečte při jednom otočení disku více sektorů než v případě skenování vnitřní části stopy. Kvůli dosažení stálé rychlosti přenosu dat jsou proto přehrávače CD-DA navrženy tak, aby měnily svou rychlost otáčení podle toho, ze kterého místa disku laserový paprsek čte. Většina systémů CD používaných k ukládání počítačových dat však používá konstantní vyšší rychlost a musí tedy zvládat změny v rychlosti přenosu dat.

V důsledku těchto konstrukčních voleb poskytují systémy k ukládání na disky CD nejvyšší výkon při manipulaci s dlouhými a souvislými datovými řetězci – obdobně jako při reprodukci hudby. Pokud oproti tomu aplikace vyžaduje přístup k datům v náhodném pořadí, nabízí koncepce ukládání na magnetické disky (s jednotlivými soustřednými stopami, které jsou rozděleny na nezávisle dostupné sektory) vyšší výkon než přístup disků CD založený na jediné spirálové stopě.

Kapacita tradičních disků CD se pohybuje v rozmezí od 600 do 700 MB. Disky **DVD (Digital Versatile Disk)** však nabízejí úložnou kapacitu několik GB. Skládají se z více poloprůhledných vrstev, které při čtení přesně zaostřeným laserem fungují jako samostatné povrchy. Tyto disky umožňují uchovávat dlouhé multimediální prezentace včetně celých filmů. Nedávno se začala prosazovat technologie Blu-ray, která pomocí laseru v modrofialové části světelného spektra (místo červeného konce spektra) dokáže zaostřit svůj laserový paprsek s velmi vysokou přesností. Díky tomu poskytují disky **BD (Blu-ray Disk)** více než pětinásobek místa disku DVD. Tyto mimořádné požadavky na úložnou kapacitu klade video ve vysokém rozlišení.



Obrázek 1.11: Formát ukládání na disky CD

Jednotky flash

Systémy hromadného úložiště založené na magnetické nebo optické technologii se vesměs vyznačují tím, že ke čtení a zápisu dat potřebují fyzický pohyb – například otáčení disku, posun čtecích a zapisovacích hlav nebo zaměření laserového paprsku. To znamená, že ukládání a načítání dat probíhá mnohem pomaleji, než umožňují elektronické obvody. Tuto nevýhodu dokáže překonat technologie **paměti flash**. Systém paměti flash ukládá bity tak, že odesílá elektronické signály přímo úložnému médiu, kde tyto signály vedou k zachycení elektronů v komůrkách oxidu křemičitého a tím se změní vlastnosti malých elektronických obvodů. Vzhledem k tomu, že tyto komory mohou zachycené elektrony uchovávat mnoho let, hodí se tato technologie k ukládání dat offline.

K datům uloženým v systémech paměti flash lze přistupovat v malých jednotkách bajtové velikosti obdobně jako u pamětí RAM. Mazat uložená data je však s ohledem na současné technologie nutné ve velkých blocích. Opakované mazání navíc komory z oxidu křemičitého postupně poškozuje. Dnešní technologie paměti flash se tedy nehodí pro univerzální operační paměť počítače, jejíž obsah se může měnit několikrát za sekundu. V zařízeních s rozumným počtem datových změn, jako jsou digitální fotoaparáty, mobilní telefony a příruční počítače PDA však paměti flash získaly pozici nejpoužívanější technologie pro hromadné ukládání dat. Vzhledem k tomu, že pamětem flash nevadí fyzické otřesy (na rozdíl od magnetických a optických systémů), mají v doméně přenosných zařízeních skutečně velký potenciál.

K ukládání libovolných dat jsou na trhu dostupná zařízení paměti flash, která se označují jako **jednotky flash** (flash drive) s kapacitou až několik set GB. Tyto paměti jsou uzavřeny v malých plastových krytech s délkou několika centimetrů. Na jednom konci mívají krytku, která chrání elektrický konektor zařízení, když není připojeno k počítači. Díky své vysoké kapacitě a snadnému připojování a odpojování od počítače jsou tato přenosná zařízení ideální k ukládání dat offline. Vzhledem k nižší stabilitě svých miniaturních úložných komor však nejsou při skutečně dlouhodobém uchovávání dat natolik spolehlivá jako optické disky.

Další aplikaci technologie flash představují **paměťové karty SD (Secure Digital)**. Tyto karty poskytují úložnou kapacitu do dvou GB a mají plastové pouzdro s kontakty, jehož velikost odpovídá poštovní známce (karty SD jsou rovněž dostupné v menších formátech „mini“ a „mikro“). **Paměťové karty SDHC (High Capacity)** mohou nabídnout až 32 GB místa a kapacita **paměťových karet SDXC (eXtended Capacity)** nové generace může překročit jeden TB. Díky svým kompaktním fyzickým rozměrům se tyto karty snadno vejdou do patic v malých elektronických zařízeních. Jsou proto ideální pro digitální fotoaparáty, smartphony, hudební přehrávače, systémy automobilové navigace a celou řadu dalších elektronických přístrojů.

Ukládání a načítání souborů

Informace uložené v systémech hromadného úložiště jsou logicky seskupeny do větších jednotek zvaných **soubory**. Typický soubor může obsahovat například celý textový dokument, fotografii, program, hudební nahrávku nebo tabulku dat o zaměstnancích společnosti. Již jsme ukázali, že v zařízeních hromadného úložiště je nutné tyto soubory ukládat a načítat v menších jednotkách o velikosti několika bajtů. Se souborem uloženým na magnetickém disku je například potřeba manipulo-

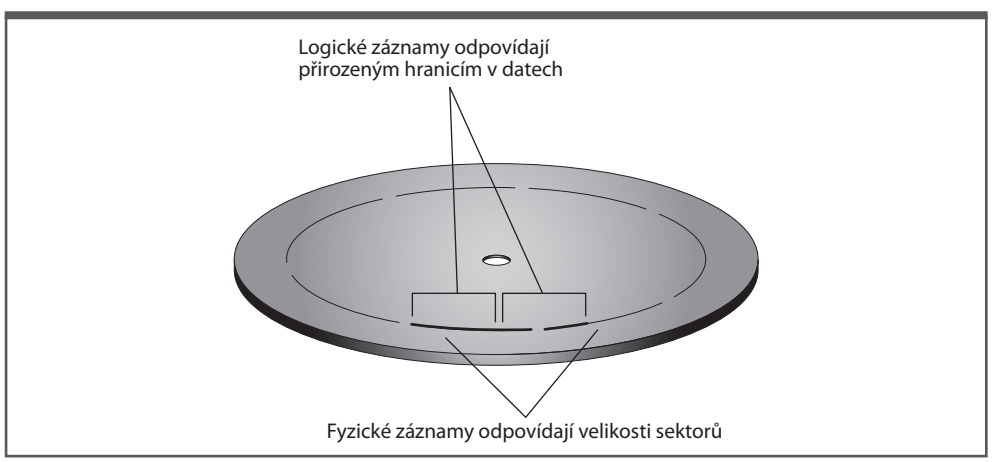
vat po jednotlivých sektorech, které mají předem určenou pevnou velikost. Blok dat, který odpovídá konkrétním parametrům úložného zařízení, se označuje jako **fyzický záznam**. Velký soubor uložený na zařízení hromadného úložiště proto obvykle zahrnuje mnoho fyzických záznamů.

V rozporu s tímto dělením na fyzické záznamy obsahuje soubor často přirozené hranice, které závisejí na reprezentovaných informacích. Soubor s údaji o zaměstnancích společnosti například může obsahovat více jednotek, z nichž každá sestává z informací o jednom zaměstnanci. Soubor s textovým dokumentem se zase může skládat z odstavců nebo stránek. Tyto přirozeně se vyskytující bloky dat se označují jako **logické záznamy**.

Logické záznamy často zahrnují menší jednotky zvané **složky**.[2] Logický záznam s informacemi o zaměstnanci bude například obsahovat složky, jako je jméno, adresa, identifikační číslo zaměstnance atd. Logické záznamy v rámci souboru bývají někdy jednoznačně identifikovány pomocí některé ze svých složek (může se jednat o identifikační číslo zaměstnance, číslo dílu nebo katalogové číslo položky). Identifikační složce tohoto typu se říká **klíčová složka**. Hodnota uložená v klíčové složce se označuje jako **klíč**.

Velikost logických záznamů jen málokdy přesně odpovídá velikosti fyzických záznamů, kterou určují vlastnosti zařízení hromadného úložiště. Několik logických záznamů proto může sdílet jediný fyzický záznam, nebo mohou být některé logické záznamy rozděleny mezi dva či více fyzických záznamů (obrázek 1.12). Data načítaná ze systému hromadného úložiště je tedy nutné poněkud uspořádat. Tento požadavek se obvykle řeší tím, že se vyhradí oblast operační paměti s dostatečnou velikostí na uložení několika fyzických záznamů. Tato oblast potom slouží k přeskupení dat. Mezi touto částí operační paměti a systémem hromadného úložiště lze tedy přenášet bloky dat kompatibilní s fyzickými záznamy, zatímco data umístěná v oblasti operační paměti umožňují odkazování na úrovni logických záznamů.

Oblast paměti, která se používá uvedeným způsobem, se označuje jako **vyrovnávací paměť** (buffer). Vyrovnávací paměť je obecně řečeno oblast paměti, která uchovává dočasná data, obvykle během jejich přenosu mezi různými zařízeními. Moderní tiskárny například obsahují vlastní paměťové obvody a většina z nich slouží jako vyrovnávací paměť k ukládání částí dokumentu, který tiskárna přijala, ale zatím nevytiskla.



Obrázek 1.12: Porovnání logických záznamů s fyzickými záznamy na disku

2 Někdy se hovoří také o **polích**; to je doslovný překlad anglického termínu field. V češtině se však termín „pole" používá v jiném významu, proto dáváme přednost označení „složka". (Pozn. odborného korektora.)

Otázky a cvičení

1. Co získáme zvýšením rychlosti otáčení pevného disku nebo CD?
2. Měli bychom při nahrávání dat do úložného systému s více disky zaplnit celý povrch disku a pokračovat dalším povrchem, nebo je vhodné naplnit nejdříve celý cylindr a poté začít dalším cylindrem?
3. Proč bychom měli data v rezervačním systému, která jsou neustále aktualizována, uchovávat na magnetickém disku a nikoli na disku CD či DVD?
4. Když upravujeme dokument v textovém procesoru, přidání textu někdy nezvětší udávanou velikost souboru v hromadném úložišti. Jindy však může vložení jediného symbolu zvětšit udávanou velikost souboru o několik stovek bajtů. Proč?
5. Jakou výhodu poskytují jednotky flash oproti jiným systémům hromadného úložiště, které jsme představili v této části?
6. Co to je vyrovnávací paměť?

1.4: Vyjádření informací pomocí posloupností bitů

Po seznámení s metodami ukládání bitů se nyní budeme zabývat tím, jak lze do řetězců bitů kódovat informace. Soustředíme se na oblíbené postupy kódování textu, číselných dat, obrazů a zvuku. Každý z uvedených systémů má určité důsledky, které se často projevují i na uživatelské úrovni. Pokusíme se těmto metodám porozumět natolik, abychom chápali jejich důsledky pro chování těchto zařízení.

Reprezentace textu

Informace v podobě textu se obvykle vyjadřují pomocí kódu, který každému z odlišných symbolů v textu (například písmenům abecedy a interpunkčním znaménkům) přiřadí jedinečný vzor bitů. Text je poté reprezentován jako dlouhý řetězec bitů, v němž sousední bitové řetězce zastupují symboly, které po sobě následují v původním textu.

Ve 40. a 50. letech 20. století bylo navrženo mnoho takových kódů, které se používaly v různých zařízeních. To způsobovalo značné potíže při komunikaci. Kvůli zlepšení této situace přijal ústav **American National Standards Institute (ANSI**, vyslovuje se „ansi“) normu **American Standard Code for Information Interchange (ASCII**, vyslovuje se „aski“). Tento kód dokáže pomocí sedmibitových posloupností vyjádřit velká a malá písmena anglické abecedy, interpunkční znaménka, číslice 0 až 9 a některé řídicí informace, jako je přechod na nový řádek, návrat vozíku a tabulátor. Kódování ASCII lze rozšířit do formátu s osmi bity na symbol, když se před nejvýznamnější cifru každého sedmibitového vzoru doplní číslice 0. Tímto postupem získáme kód, ve kterém každý bitový řetězec přesně odpovídá typické paměťové buňce velikosti jednoho bajtu. Kromě toho však také poskytuje 128 dalších bitových vzorů (vzniknou přiřazením hodnoty 1 dodatečnému bitu), které mohou označovat symboly jiných abeced s příslušnou diakritikou.

Část kódu ASCII ve formátu s osmi bity na symbol je znázorněna v Příloze A. Pomocí uvedené přílohy můžeme dekódovat bitovou posloupnost

```
01001000 01100101 01101100 01101100 01101111 00101110
```

jako zprávu „Hello“., jak je patrné na obrázku 1.13.

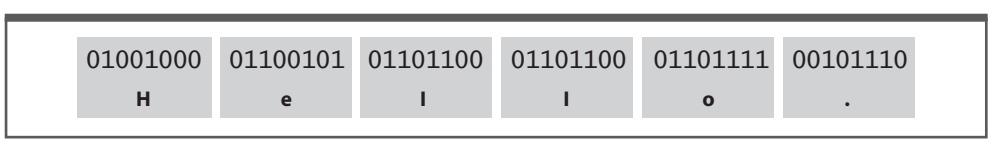
Organizace s názvem **International Organization for Standardization** (je známa i pod svou zkratkou **ISO**, která připomíná řecké slovo *isos* s významem „rovný“) vyvinula několik rozšíření kódu ASCII. Každá z těchto variant umožňuje pracovat s významnou skupinou jazyků. Jeden standard například poskytuje potřebné symboly ke kódování textu ve většině středoevropských jazyků. Dodatečných 128 bitových vzorů obsahuje symboly jako ‰, znaky s diakritickými znaménky jako je á, č nebo ď pro češtinu, slovenštinu, polštinu a další středoevropské jazyky používající latinku.

Rozšíření standardu ASCII o normy ISO představovalo značný pokrok směrem k podpoře veškeré vícejazyčné komunikace ve světě. Objevily se však dva zásadní problémy. Počet dodatečných bitových vzorů, které jsou k dispozici v rozšířeném kódování ASCII, jednoduše nestačí na vyjádření abeced mnoha asijských a některých východoevropských jazyků. Konkrétní dokumenty jsou navíc omezeny na symboly jednoho vybraného standardu. Dokumenty tedy nemohou obsahovat texty ve více jazycích z odlišných skupin. Ukázalo se, že obě uvedené vlastnosti značně omezují mezinárodní komunikaci. Kvůli těmto nedostatkům vznikl ve spolupráci několika významných výrobců hardwaru a softwaru standard **Unicode**, který se v počítačové komunitě rychle rozšířil. Tento kód nahrazuje každý symbol jedinečným vzorem z 16 bitů. Díky tomu kódování Unicode poskytuje 65 536 různých bitových řetězců, což postačuje k reprezentaci textu napsaného v jazycích typu čínštiny, japonštiny či hebrejštiny.

Soubor, který obsahuje dlouhé sekvence symbolů v kódování ASCII nebo Unicode, se často označuje jako **textový soubor**. Je důležité rozlišovat mezi jednoduchými textovými soubory, které lze upravovat v softwarových nástrojích zvaných **textové editory** (často se jednoduše označují jako editory) a komplikovanějšími soubory, jaké generují **textové procesory** (například Microsoft Word). V obou dokumentech se nacházejí textové informace. Textový soubor však obsahuje pouze kódování jednotlivých znaků textu, zatímco soubor vytvořený textovým procesorem zahrnuje mnoho nestandardních kódů, které znamenají změny písma, informace o zarovnání atd.

Reprezentace číselných hodnot

Potřebujeme-li zaznamenat čistě číselné informace, není ukládání informací formou kódování znaků příliš efektivní. Můžeme to ukázat na problému uložení hodnoty 25. Jestliže budeme trvat na tom, že ji uložíme jako symboly kódování ASCII, z nich každý vyžaduje jeden bajt, budeme potřebovat celkem 16 bitů. Kromě toho dokážeme pomocí 16 bitů uložit pouze čísla do 99. Jak se však zakrátko dozvíme, pomocí binárního (dvojkového) zápisu můžeme ve stejných 16 bitech uložit libovolné celé číslo v rozsahu od 0 do 65 535. **Binární zápis** (nebo jeho varianty) proto při kódování číselných dat ukládaných v počítači hraje významnou roli.



Obrázek 1.13: Zpráva „Hello“. v kódování ASCII

Institut ANSI

Institut ANSI (American National Standards Institute) založilo roku 1918 malé konsorcium technických společností a vládních úřadů jako neziskovou organizaci, která měla koordinovat vývoj dobrovolných standardů v soukromém sektoru. V současnosti je členy institutu ANSI více než 1 300 podniků, profesionálních organizací, odborů a vládních úřadů. Institut ANSI sídlí v New Yorku a reprezentuje USA při zasedáních organizace ISO. American National Standards Institute má webovou stránku *http://www.ansi.org*.

Obdobné organizace fungují i v jiných zemích. Patří k nim Standards Australia, Standards Council of Canada, Čínský státní úřad kvality a technického dozoru, Deutsches Institut für Normung (Německo), Japonská komise průmyslových standardů, Dirección General de Normas (Mexiko), Státní komise Ruské federace pro standardizaci a metrologii, Švýcarská asociace pro standardizaci a British Standards Institution. V České republice to je Úřad pro technickou normalizaci, metrologii a státní zkušebnictví.

Dvojkový zápis umožňuje reprezentovat číselné hodnoty pouze pomocí číslic 0 a 1 místo číslic 0, 1, 2, 3, 4, 5, 6, 7, 8 a 9 jako v tradiční desítkové soustavě. Dvojkovou soustavou se budeme podrobněji zabývat v části 1.5. Prozatím postačí, když pochopíme základy této soustavy. Jako model nám poslouží klasické počitadlo ujetých kilometrů na tachometru v automobilu, jehož kotouče nenesou číslice 0 až 9 jako obvykle, ale pouze cifry 0 a 1. Počitadlo je nejdříve nastaveno na samé nuly. Když automobil ujede první kilometr, kotouč úplně vpravo se přetočí z hodnoty 0 na 1. Když se pak tato číslice 1 vrátí zpět na 0, objeví se číslice 1 na kotouči vlevo. Počitadlo tedy zobrazuje vzor 10. Nulu napravo poté vystřídá číslo 1 a objeví se číslo 11. Nyní se kotouč zcela vpravo otočí z čísla 1 zpět na 0. Číslice 1 po jeho levé straně se proto rovněž změní na 0. Přitom se rovněž objeví další jednička ve třetím sloupci zprava a počitadlo nyní udává číslo 100. Stručně řečeno, se za jízdy automobilu budou na počitadle postupně zobrazovat následující hodnoty:

```
0000
0001
0010
0011
0100
0101
0110
0111
1000
```

Tuto řadu tvoří celá čísla od nuly do osmi vyjádřená ve dvojkové soustavě. Tento způsob počítání je sice pracný, ale pokud bychom v něm pokračovali, zjistili bychom, že bitový vzor tvořený šestnácti jedničkami odpovídá hodnotě 65 535. Tímto způsobem bychom si ověřili, že pomocí 16 bitů lze skutečně zakódovat libovolné celé číslo v rozsahu od 0 do 65 535.

Vzhledem k této efektivitě se číselné informace ukládají častěji ve tvaru binárního zápisu a nikoli jako zakódované symboly. Úmyslně říkáme „tvar binárního zápisu“, protože jednoduchý binární systém, který jsme právě popsali, představuje pouze základ několika metod ukládání čísel v počítačích. Některými z těchto variant binárního systému se budeme zabývat dále v této kapitole. Prozatím stačí zmínit, že při ukládání čísel se často uplatňuje systém označovaný jako **dvojkový doplněk** (two's complement – viz část 1.6). Tato notace totiž dokáže snadno vyjádřit nejen

kladná, ale i záporná čísla. Při reprezentování čísel se zlomkovou částí, jako je např. $4\frac{1}{2}$ nebo $\frac{3}{4}$, se uplatňuje další metoda zvaná zápis s **plovoucí desetinnou čárkou** (floating-point – viz část 1.7).

Reprezentace obrázků

Jeden ze způsobů reprezentace obrázků je založen na interpretaci obrázku jako kolekce bodů, které se označují jako **pixel** (z anglického „picture element" – prvek obrázku). Následně je zakódován vzhled každého pixelu a celý obrázek je vyjádřen jako kolekce těchto kódovaných pixelů. Kolekce tohoto typu se nazývá **bitová mapa** (nebo také rastrový obrázek). Tento přístup získal svou popularitu díky tomu, že koncepci pixelů využívá mnoho zobrazovacích zařízení, jako jsou tiskárny a monitory. Obrázky ve formátu bitové mapy lze proto snadno prezentovat.

Metoda kódování pixelů v bitové mapě se liší v závislosti na aplikaci. V případě jednoduchého černobílého obrázku je možné každý pixel popsat jediným bitem, jehož hodnota odpovídá tomu, zda je příslušný pixel černý či bílý. Tento přístup používá většina faxů. U černobílých fotografií s plynulými přechody je nutné vyjádřit každý pixel pomocí posloupnosti bitů (obvykle osmi). Díky tomu lze znázornit různé odstíny šedi.

V případě barevných obrázků se každý pixel kóduje složitějším postupem. Obvykle se uplatňuje jeden ze dvou přístupů. U prvního z nich, který se nazývá kódování RGB, je každý pixel reprezentován pomocí tří barevných komponent – červené, zelené a modré, které odpovídají třem primárním barvám světla. Intenzitu každé barevné komponenty zpravidla popisuje jeden bajt. K vyjádření jediného pixelu původního obrázku proto potřebujeme tři bajty úložiště.

Alternativu k prostému kódování RGB představuje použití „jasové" komponenty a dvou barevných komponent. V tomto případě je „jasová" komponenta, která se označuje jako luminance (svítivost) pixelu, ve skutečnosti tvořena součtem červené, zelené a modré komponenty. (Ve skutečnosti se považuje za intenzitu bílého světla v pixelu, ale těmito podrobnostmi se zde nemusíme zabývat.) Další dvě komponenty zvané modrá a červená chrominance (barevnost) závisejí na rozdílu luminance pixelu a intenzity modrého, respektive červeného světla v pixelu. Uvedené tři komponenty pak společně nesou informaci, která postačuje k reprodukci pixelu.

Kódování obrázků pomocí komponent luminance a chrominance se prosadilo v oblasti barevného televizního vysílání, protože tento přístup umožňoval kódovat barevnost způsobem, který byl kompatibilní i se starými černobílými přijímači. Verzi vysílání v šedé škále bylo možné jednoduše získat pouze na základě komponenty luminance zakódovaného barevného obrazu.

ISO (International Organization for Standardization)

Organizace ISO (International Organization for Standardization) byla založena roku 1947 jako celosvětová federace standardizačních orgánů, přičemž každou zemi zastupuje jeden z nich. V současnosti sídlí ve švýcarské Ženevě a sdružuje více než 100 členských institucí a také mnohé korespondenční členy. (Korespondenčním členem je zpravidla standardizační orgán ze země, která nemá celostátně uznávaný standardizační úřad. Tito členové se nemohou přímo podílet na vývoji standardů, ale dostávají informace o činnostech organizace ISO.) Organizace ISO má webovou adresu *http://www.iso.org*.

Nevýhoda reprezentace obrázků jako bitových map spočívá v tom, že nelze snadno změnit jejich měřítko. Chceme-li obrázek zvětšit, nelze to provést bez zvětšení pixelů, takže nový obrázek vypadá zrnitě. (Tato metoda se u digitálních fotoaparátů označuje jako „digitální zoom" na rozdíl od „optického zoomu", kdy se mění nastavení objektivu fotoaparátu.)

Tomuto problému se změnou měřítka se lze vyhnout pomocí alternativního způsobu reprezentace obrázků, který popisuje obrázek jako kolekci geometrických struktur, jako jsou čáry a křivky. Tyto prvky je možné zakódovat metodami analytické geometrie. Díky tomuto popisu může způsob zobrazení geometrických struktur zvolit koncové zobrazovací zařízení a nemusí přitom nuceně reprodukovat určitý bitový vzor. Tento přístup se uplatňuje při návrhu škálovatelných písem, která jsou dostupná v dnešních systémech na zpracování textu. Například technologie TrueType (kterou vyvinuly společnosti Microsoft a Apple) umožňuje geometricky popsat textové symboly. Obdobná technologie s názvem PostScript (se kterou přišla společnost Adobe Systems) dokáže popsat znaky a navíc obecnější obrazová data. Tento geometrický přístup ke znázornění obrázků je zavedený také v systémech **CAD (computer-aided design – počítačový návrh)**, které slouží k zobrazení trojrozměrných objektů a manipulaci s nimi na obrazovkách počítačů.

Rozdíl mezi vyjádřením obrázku pomocí geometrických struktur nebo bitových map je snadno pochopitelný pro každého, kdo pracoval s nástrojem na kreslení obrázků typu Malování v systému Windows, který umožňuje uživateli kreslit obrázky složené z předem vybraných tvarů (např. obdélníků, oválů a jednoduchých křivek). Uživatel jednoduše vybere požadovaný geometrický tvar z nabídky a poté jej vykreslí přetažením myši. Během kreslení uchovává software geometrický popis kresleného tvaru. Na základě pohybu myši dochází k úpravám vnitřní geometrické reprezentace, která je poté převedena do podoby bitové mapy a zobrazena. Tímto způsobem lze snadno měnit velikost a tvar jednotlivých obrazců. Po dokončení celého procesu je však základní geometrický popis zrušen a zůstane zachována pouze bitová mapa. To znamená, že při pozdějších úpravách je nutné pracně modifikovat jednotlivé pixely. Jiné kreslicí systémy však udržují popis geometrických tvarů, které lze později modifikovat. Při práci s těmito systémy je možné snadno měnit velikost tvarů, které si přitom nezávisle na měřítku zachovají hladké obrysy.

Reprezentace zvuku

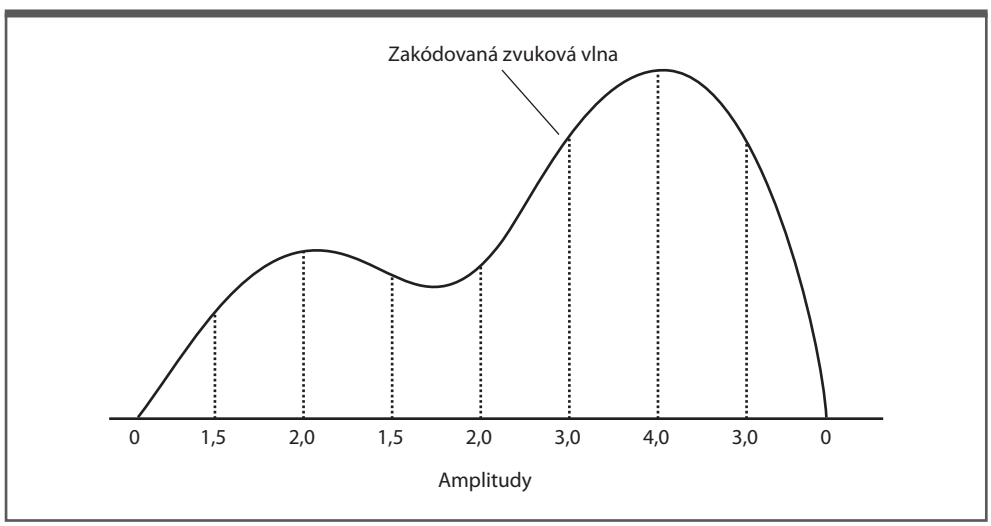
Nejobecnější způsob kódování zvukových informací k jejich ukládání a manipulaci v počítači je založen na vzorkování amplitudy zvukové vlny v pravidelných intervalech a záznamu získané řady hodnot. Například posloupnost 0, 1,5, 2,0, 1,5, 2,0, 3,0, 4,0, 3,0 a 0 by představovala zvukovou vlnu, jejíž amplituda roste, krátce klesá, znovu roste na vyšší úroveň a poté se vrací zpět k 0 (obrázek 1.14). Tato metoda se vzorkovací frekvencí 8 000 vzorků za sekundu se již mnoho let používá u meziměstských telefonních hovorů. Hlas na jedné straně komunikace je zakódován pomocí číselných hodnot, které představují amplitudu hlasu každou osmitisícinu sekundy. Příslušné číselné hodnoty se pak přenášejí po komunikační lince na stranu příjemce, kde umožňují reprodukovat zvuk hlasu.

Frekvence 8 000 vzorků za sekundu je sice zdánlivě vysoká, ale pro záznam hudby ve vysoké kvalitě ve skutečnosti nestačí. Kvality zvuku, jakou poskytují dnešní hudební disky CD, se dosahuje díky vzorkovací frekvenci 44 100 vzorků za sekundu. Získa-

ná data každého vzorku jsou reprezentována pomocí 16 bitů (v případě stereofonní nahrávky 32 bitů). Každá sekunda stereofonní hudební nahrávky tedy vyžaduje více než milion bitů.

Alternativní kódovací systém s názvem MIDI (Musical Instrument Digital Interface, vyslovuje se „midi“) se často používá pro hudební syntetizátory v elektronických klávesových nástrojích, zvuky videoher a zvukové efekty webových stránek. Díky zakódování pokynů k produkci hudby pomocí syntetizátoru namísto zakódování samotného zvuku klade metoda MIDI menší nároky na kapacitu úložiště než vzorkování. Technologie MIDI konkrétně definuje nástroj, který má zaznít, výšku tónu a jeho trvání. To znamená, že klarinet hrající notu D po dobu dvou sekund lze zakódovat pomocí pouhých tří bajtů. Vzorkování s frekvencí 44 100 vzorků za sekundu k tomu přitom potřebuje více než dva miliony bitů.

Stručně řečeno si můžeme systém MIDI představit jako zápis notové osnovy, kterou dokáže číst hudební interpret, nikoli jako samotný koncert. „Nahrávka“ ve formátu MIDI proto může při přehrání na různých syntetizérech znít značně odlišně.



Obrázek 1.14: Zvuková vlna reprezentovaná číselnou řadou 0, 1,5, 2,0, 1,5, 2,0, 3,0, 4,0, 3,0, 0

Otázky a cvičení

1. Následující zpráva je zakódována pomocí standardu ASCII s 8 bity na symbol. Jaký je její obsah? (Viz Příloha A.)

   ```
   01000011 01101111 01101101 01110000 01110101 01110100
   01100101 01110010 00100000 01010011 01100011 01101001
   01100101 01101110 01100011 01100101
   ```

2. Jaký vztah existuje v kódu ASCII mezi kódy pro odpovídající velká a malá písmena? (Viz Příloha A.)
3. Zakódujte tyto věty v kódování ASCII:
   a. „Stop!“ zavolal Petr.
   b. Je 2 + 3 = 5?

4. Popište běžný předmět, který se může nacházet v jednom ze dvou stavů (např. vlajka na stožáru, která buď vlaje, nebo je stažená). Přiřaďte jednomu stavu symbol 1 a druhému symbol 0. Ukažte, jak bude při uložení s těmito bity vypadat reprezentace písmene *b* v kódování ASCII.
5. Převeďte každou z následujících binárních reprezentací na odpovídající číslo se základem deset:
   a. 0101 **b.** 1001 **c.** 1011
   d. 0110 **e.** 10000 **f.** 10010
6. Převeďte každou z následujících číselných reprezentací se základem deset na odpovídající binární číslo:
   a. 6 **b.** 13 **c.** 11
   d. 18 **e.** 27 **f.** 4
7. Jakou největší číselnou hodnotu je možné vyjádřit pomocí tří bajtů, pokud by byla každá číslice zakódována pomocí jednoho znaku na jeden bajt v kódování ASCII? Jaká bude tato hodnota při použití binární notace?
8. Alternativu k šestnáctkovému zápisu při reprezentaci bitových posloupností představuje **tečková desítková notace**, která vyjadřuje každý bajt v posloupnosti jako jeho ekvivalent v desítkové soustavě. Tyto bajtové reprezentace se poté oddělují tečkami. Například zápis 12.5 představuje vzor 0000110000000101 (bajt 00001100 je reprezentován číslem 12 a bajt 00000101 číslem 5) a posloupnost 100010000001000000000111 lze zapsat jako 136.16.7. Vyjádřete v tečkové desítkové notaci všechny následující bitové řetězce.
   a. 0000111100001111 **b.** 001100110000000010000000
   c. 0000101010100000
9. Jakou výhodu oproti bitovým mapám poskytuje reprezentace obrázků pomocí geometrických struktur? Jaké výhody mají naopak metody založené na bitových mapách?
10. Předpokládejme, že stereofonní nahrávka jedné hodiny hudby je zakódována se vzorkovací frekvencí 44 100 vzorků za sekundu, o které jsme se zmínili v textu. Jaká bude velikost zakódované nahrávky v porovnání s úložnou kapacitou disku CD?

1.5: Dvojková soustava

V části 1.4 jsme si ukázali, že dvojkový zápis (binární notace) umožňuje reprezentovat číselné hodnoty pouze pomocí číslic 0 a 1 místo deseti číslic 0 až 9, které se používají v běžnější notaci se základem deset. Nyní se budeme dvojkovou soustavou zabývat podrobněji.

Dvojkový zápis

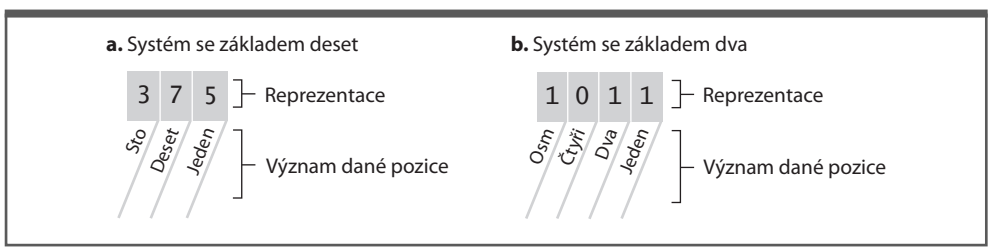
V číselné soustavě se základem deset je každé pozici v reprezentovaném čísle přiřazen určitý význam. V reprezentaci 375 se číslice 5 nachází na pozici přidružené k jednotkám, číslice 7 na pozici přidružené k desítkám a číslice 3 má pozici s vý-

znamem stovek (obrázek 1.15a). Význam každé pozice je desetkrát větší než význam sousední pozice vpravo. Hodnotu, jakou představuje celý výraz, získáme vynásobením hodnoty jednotlivých číslic a jejich významu podle polohy každé číslice a následným sečtením těchto součinů. Uveďme příklad: zápis 375 reprezentuje (3 × sto) + (7 × deset) + (5 × jedna), což lze matematicky rovněž zapsat jako $(3 \times 10^2) + (7 \times 10^1) + (5 \times 10^0)$.

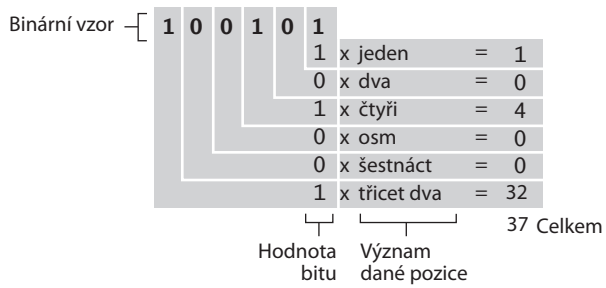
Také pozice každé číslice ve dvojkovém zápisu souvisí s významem, ale v tomto případě každé pozici přísluší dvojnásobný význam oproti pozici, která se nachází napravo od ní. Přesněji řečeno číslice, která je umístěna na pravém konci binární reprezentace, přísluší k jednotkám (2^0), další pozice směrem doleva je přidružena k významu dva (2^1), následující pozice souvisí s významem čtyři (2^2), pozice nalevo od ní s významem osm (2^3) atd. Například v binárním zápisu 1011 se číslice zcela vpravo 1 nachází na pozici přidružené k jednotkám. Číslice 1 vedle ní má pozici s významem dvou, následná číslice 0 je umístěna na pozici související s významem čtyři a číslice 1 zcela vlevo zaujímá pozici s významem osm (obrázek 1.15b).

Chceme-li zjistit hodnotu, kterou představuje binární zápis, postupujeme stejným způsobem jako u čísel v desítkové soustavě – vynásobíme hodnotu každé číslice významem, který odpovídá její pozici, a dílčí výsledky sečteme. Uveďme si příklad: zápis 100101 označuje hodnotu 37, jak je zřejmé z obrázku 1.16. Vzhledem k tomu, že dvojkový zápis pracuje pouze s číslicemi 0 a 1, omezuje se tento postup násobení a sčítání na prosté sečtení významu odpovídajícího pozicím, které jsou obsazeny jedničkami. Binární schéma 1011 tedy označuje hodnotu jedenáct, protože jedničky se nacházejí na pozicích, které mají význam jedna, dva a osm.

V části 1.4 jsme se naučili, jak počítat ve dvojkovém zápisu. Díky tomu jsme dokázali zakódovat malá celá čísla. Při hledání dvojkových reprezentací velkých hodnot může být vhodnější přístup, který popisuje algoritmus na obrázku 1.17. Aplikujme nyní tento algoritmus na hodnotu třináct (viz obrázek 1.18). Nejdříve vydělíme třináct dvěma a získáme podíl šest se zbytkem jedna. Vzhledem k tomu, že podíl není nulový, podle kroku 2 tento podíl (šest) vydělíme dvěma a dostaneme nový podíl tři se zbytkem nula. Aktuální podíl stále není nulový, takže jej dále vydělíme dvěma. Výsledkem je podíl jedna se zbytkem jedna. Ještě jednou vydělíme naposledy získaný podíl (jedna) dvěma a tentokrát dospějeme k podílu nula se zbytkem jedna. Protože jsme nyní získali nulový podíl, přejdeme ke kroku 3. Zjišťujeme, že binární reprezentace původní hodnoty (třináct) má tvar 1101, což vyplývá z posloupnosti zbytků.



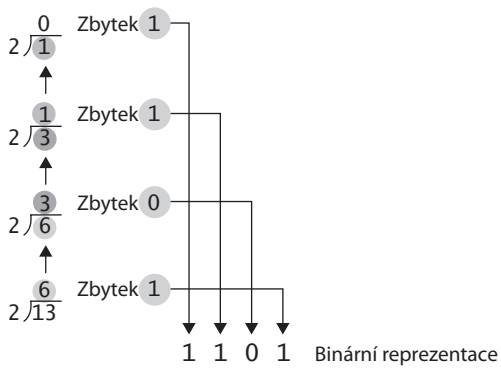
Obrázek 1.15: Systémy se základem deset a dva



Obrázek 1.16: Dekódování binární reprezentace 100101

Krok 1:	Vydělte hodnotu dvěma a zapište zbytek.
Krok 2:	Dokud není získaný podíl nulový, pokračujte v dělení posledního podílu dvěma a zapisujte zbytky.
Krok 3:	Když dostanete nulový podíl, skládá se binární reprezentace původní hodnoty ze zbytků, které v pořadí jejich výpočtu zapíšete zprava doleva.

Obrázek 1.17: Algoritmus nalezení binární reprezentace kladného celého čísla



Obrázek 1.18: Získání binární reprezentace čísla třináct pomocí algoritmu z obrázku 1.17

Binární sčítání

Chceme-li porozumět procesu sčítání dvou celých čísel, která jsou vyjádřena v binárním tvaru, musíme si nejdříve zopakovat postup sčítání hodnot reprezentovaných v tradiční soustavě se základem deset. Zvažme například následující problém:

```
  58
+ 27
```

Začneme tím, že sečteme číslice 8 a 7 v sloupci na pravé straně a získáme hodnotu 15. Pod příslušný sloupec zapíšeme číslici 5 a jedničku přeneseme do dalšího sloupce tímto způsobem:

```
  1
 58
+ 27
  5
```

Nyní sečteme číslice 5 a 2 v dalším sloupci spolu s přenesenou hodnotou 1. Dostaneme součet 8, který zaznamenáme pod daný sloupec. Výsledek vypadá takto:

```
 58
+ 27
 85
```

Shrnuto: postupujeme zprava doleva a přitom sčítáme číslice v každém sloupci, zapisujeme méně významnou číslici součtu pod daný sloupec a přenášíme významnější číslici součtu (pokud existuje) do dalšího sloupce.

Pokud chceme sečíst dvě celá čísla vyjádřená ve dvojkovém zápisu, budeme dodržovat stejný postup s tím rozdílem, že všechny součty budeme počítat podle pravidel na obrázku 1.19 a nikoli tradičním způsobem, který platí pro desítkovou soustavu a který jsme se naučili v základní škole. Chceme-li například vyřešit úlohu,

```
 111010
+ 11011
```

začneme sečtením číslic 0 a 1 úplně vpravo. Získáme hodnotu 1, kterou zapíšeme pod sloupec. Nyní sečteme číslice 1 a 1 v dalším sloupci a dostaneme hodnotu 10. Nulu z této desítky umístíme pod sloupec a jedničku přeneseme na začátek dalšího sloupce. V této fázi vypadá naše řešení následovně:

```
   1
 111010
+ 11011
    01
```

```
  0     1     0     1
+ 0   + 0   + 1   + 1
  0     1     1    10
```

Obrázek 1.19: Pravidla binárního sčítání

V následujícím sloupci sečteme číslice 1, 0 a 0 s výsledkem 1 a tuto jedničku zapíšeme pod sloupec. Součet 1 a 1 ve vedlejším sloupci dává 10. Pod sloupec umístíme 0 a číslici 1 přeneseme do dalšího sloupce. Naše řešení nyní vypadá takto:

```
  1
 111010
+ 11011
  0101
```

Součet číslic 1, 1 a 1 v dalším sloupci se rovná 11 (binární zápis hodnoty tři). Pod sloupec zapíšeme číslici 1 nižšího řádu a druhou jedničku přesuneme na začátek následujícího sloupce. Přičteme tuto jedničku k jedničce, která již v tomto sloupci je, a dostaneme 10. Opět zapíšeme nulu nižšího řádu a přeneseme jedničku do vyššího sloupce. Nyní máme tento stav:

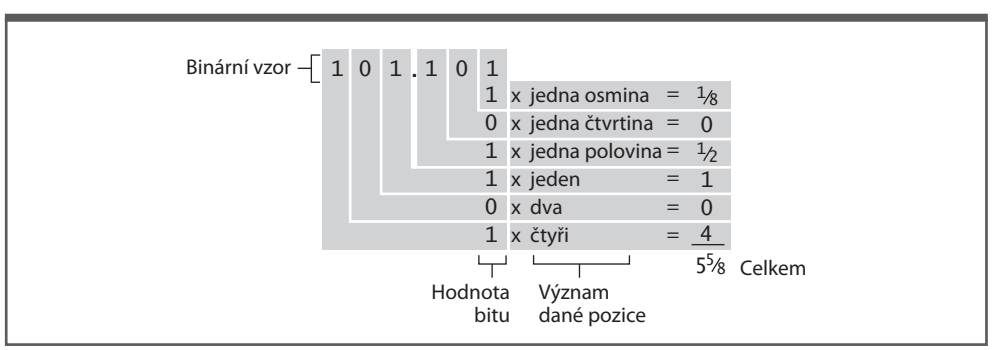
```
  1
   111010
+  11011
  010101
```

Jediná položka v posledním sloupci je číslice 1, kterou jsme přenesli z předchozího sloupce. Zapíšeme ji tedy v rámci výsledku. Naše řešení tedy nakonec vypadá takto:

```
   111010
+  11011
 1010101
```

Zlomky ve dvojkové soustavě

Chceme-li rozšířit binární zápis tak, aby umožňoval pracovat se zlomky, použijeme přitom **řádovou čárku** (radix point), která plní stejnou funkci jako desetinná čárka v desítkové notaci. Číslice vlevo od čárky tedy reprezentují celočíselnou část hodnoty a interpretují se ve dvojkové soustavě, kterou jsme popsali výše. Číslice napravo od řádové čárky představují zlomkovou část hodnoty a interpretují se podobným způsobem jako předchozí bity až na to, že jejich pozice mají přiřazen význam zlomků. První pozici napravo od čárky tedy přísluší význam $\frac{1}{2}$ (což je 2^{-1}), další pozici význam $\frac{1}{4}$ (které se rovná 2^{-2}), další pozici $\frac{1}{8}$ (tj. 2^{-3}) atd. Můžeme si povšimnout, že se jedná jen o pokračování výše uvedeného pravidla: každé pozici je přiřazen dvojnásobný význam oproti pozici napravo od ní. Jakmile jsme pozicím bitů přiřadili tyto významy, postupujeme při dekódování dvojkového zápisu s řádovou čárkou stejně jako u čísel bez této čárky. Konkrétně řečeno vynásobíme každou bitovou hodnotu významem, který je v binární reprezentaci pozici daného bitu přiřazen. Ukážeme to na binárním zápisu 101,101, který lze dekódovat na $5\frac{5}{8}$, jak je patrné na obrázku 1.20.



Obrázek 1.20: Dekódování binárního zápisu 101,101

Analogové versus digitální počítání

Ve 20. století mnozí výzkumníci diskutovali o výhodách a nevýhodách digitální a analogové technologie. V digitálním systému je hodnota zakódována jako řada číslic a poté je uložena pomocí několika prvků, z nichž každý reprezentuje jednu z číslic. Analogový systém uchovává každou hodnotu pomocí jediného prvku, který může vyjádřit libovolnou hodnotu ze souvislého intervalu.

Porovnejme oba přístupy na příkladu kbelíků, které lze naplnit vodou. Chceme-li simulovat digitální systém, můžeme se shodnout na konvenci, že prázdný kbelík bude představovat číslici 0 a plný kbelík bude znamenat číslici 1. Pak můžeme číselnou hodnotu uložit jako řadu kbelíků s použitím zápisu s plovoucí řádovou čárkou (viz část 1.7). Oproti tomu analogový systém můžeme modelovat tak, že naplníme vodou jediný kbelík a úroveň hladiny bude přitom udávat reprezentovanou číselnou hodnotu. Na první pohled se zdá, že analogový systém je přesnější. Netrpí totiž zaokrouhlovacími chybami, které jsou vlastní digitálnímu systému (opět viz část 1.7). Každý pohyb kbelíku v analogovém systému však může způsobit chybu při odečítání hladiny vody, zatímco v digitálním systému by muselo dojít k mimořádně silnému „šplouchání", aby se setřel rozdíl mezi plným a prázdným kbelíkem. Digitální systém je tedy odolnější k chybám než systém analogový. Tato robustnost představuje hlavní důvod, proč na digitální technologii přecházejí mnohá zařízení, která byla dříve založena na analogové technologii (například telefonní komunikace, záznam zvuku a televizní vysílání).

Postupy používané při sčítání v systému se základem deset platí také ve dvojkové soustavě. Chceme-li tedy sečíst dvě dvojkové reprezentace s řádovou čárkou, stačí obě řádové čárky zarovnat pod sebe a aplikovat stejný algoritmus sčítání jako ve výše uvedeném případě. Například součet čísel 10,011 a 100,11 dává 111,001 (viz dále):

```
  10,011
+ 100,110
  111,001
```

Otázky a cvičení

1. Převeďte každou z následujících dvojkových reprezentací na odpovídající číslo se základem deset:

   a. 101010 **b.** 100001 **c.** 10111 **d.** 0110 **e.** 11111

2. Převeďte každou z následujících číselných reprezentací se základem deset na odpovídající dvojkové číslo:

   a. 32 **b.** 64 **c.** 96 **d.** 15 **e.** 27

3. Převeďte každou z následujících dvojkových reprezentací na odpovídající číslo se základem deset:

   a. 11,01 **b.** 101,111 **c.** 10,1 **d.** 110,011 **e.** 0,101

4. Vyjádřete následující hodnoty ve dvojkové soustavě:

   a. $4\frac{1}{2}$ **b.** $2\frac{3}{4}$ **c.** $1\frac{1}{8}$ **d.** $\frac{5}{16}$ **e.** $5\frac{5}{8}$

5. Spočítejte následující součty ve dvojkové soustavě:

   a.
   ```
   11011
   +1100
   ```
   b.
   ```
   1010,001
   +   1,101
   ```
   c.
   ```
   11111
   + 0001
   ```
   d.
   ```
   111,11
   + 00,01
   ```

1.6: Ukládání celých čísel

Matematici se již dávno začali zajímat o systémy zápisu čísel a mnohé jejich myšlenky se později uplatnily při návrhu digitálních obvodů. V této části se budeme zabývat dvěma z těchto systémů, které umožňují reprezentovat celá čísla v počítačích: notací dvojkového doplňku a notací s posunem. Tyto systémy jsou založeny na binární soustavě, ale vyznačují se dalšími vlastnostmi, které zlepšují jejich kompatibilitu s návrhem počítačů. Vedle výhod však mají i určité nevýhody. Pokusíme se nyní porozumět tomu, jak tyto vlastnosti ovlivňují činnost počítačů.

Dvojkový doplněk

V dnešních počítačích se celá čísla nejčastěji reprezentují pomocí **dvojkového doplňku**. Všechny hodnoty v tomto systému mají pevný počet bitů. V současnosti se běžně používá systém dvojkového doplňku, kde je každá hodnota vyjádřena posloupností 32 bitů. Díky svému značnému rozsahu dovoluje takovýto systém reprezentovat mnoho různých čísel, ale pro výukové účely se příliš nehodí. Při seznamování s vlastnostmi systémů dvojkového doplňku se proto zaměříme na jejich jednodušší varianty.

Obrázek 1.21 představuje dva systémy dvojkového doplňku. Jeden z nich je založen na kombinacích tří bitů a druhý na řetězcích s délkou čtyřech bitů. Při tvorbě tohoto systému vycházíme od řetězce příslušné délky se samými nulami a pokračujeme v binárním počítání, dokud nedostaneme vzor s jedinou nulou, po níž následují samé jedničky. Tyto posloupnosti označují hodnoty 0, 1, 2, 3... Schémata, která vyjadřují záporné hodnoty, získáme takto: začneme řetězcem příslušné délky se samými jedničkami a poté počítáme ve dvojkové soustavě pozpátku, až dostaneme vzor s jedinou jedničkou, po které následují samé nuly. Tyto vzory jsou přiřazeny hodnotám –1, –2, –3... (Místo obtížného počítání pozpátku ve dvojkové soustavě stačí začít na samém konci tabulky od bitového schématu, které je tvořeno jednou jedničkou a zbývajícími nulami, a pak počítat nahoru až po vzor, který obsahuje pouze jedničky.)

a. S použitím vzorů délky tří bitů

Bitové schéma	Reprezentovaná hodnota
011	3
010	2
001	1
000	0
111	–1
110	–2
101	–3
100	–4

b. S použitím vzorů délky čtyř bitů

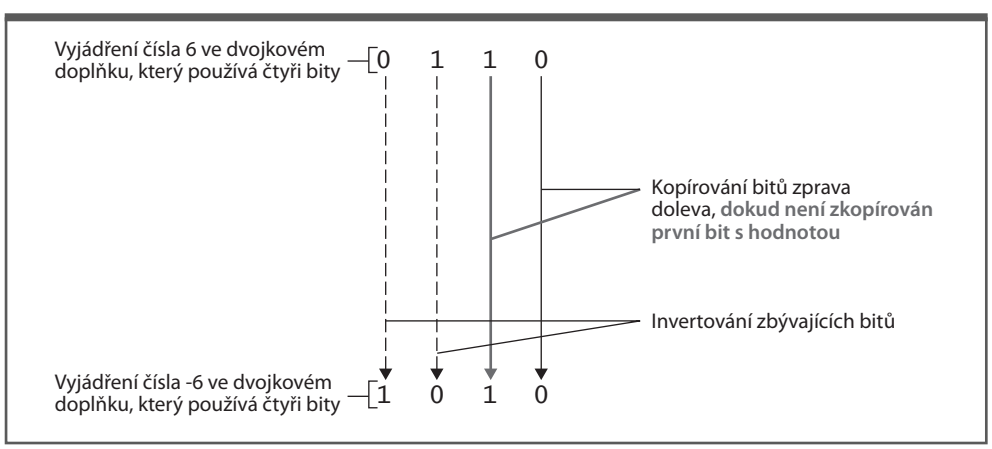
Bitové schéma	Reprezentovaná hodnota
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	–1
1110	–2
1101	–3
1100	–4
1011	–5
1010	–6
1001	–7
1000	–8

Obrázek 1.21: Systémy dvojkového doplňku

Můžeme si všimnout, že bit na levém konci bitového vzoru v systému dvojkového doplňku udává znaménko reprezentované hodnoty. Bit zcela vlevo se proto často označuje jako **znaménkový bit** (sign bit). V systému dvojkového doplňku jsou záporné hodnoty vyjádřeny vzory, které mají znaménkový bit 1. Nezáporné hodnoty lze poznat podle toho, že jejich znaménkový bit má hodnotu 0.

V systému dvojkového doplňku platí jednoduchý vztah mezi vzory, které znamenají kladná a záporná čísla se stejnou absolutní hodnotou. Při čtení zprava doleva se shodují až po první jedničku včetně. Ve své zbývající části mají tyto vzory podobu svých doplňků. (**Doplněk** (complement) vzoru je takový vzor, který získáme záměnou všech hodnot 0 za 1 a všech hodnot 1 za 0. Vzájemně se tedy doplňují například sekvence 0110 a 1001.) V systému se čtyřbitovými posloupnostmi na obrázku 1.21 končí vzory reprezentující čísla 2 i –2 dvojicí bitů 10. Vzor, který označuje číslo 2, však začíná bity 00, zatímco vzor přidružený k číslu –2 má na začátku bity 11. Z tohoto pozorování můžeme odvodit algoritmus na obousměrný převod mezi bitovými vzory, které patří kladným a záporným číslům se stejnou absolutní hodnotou. Stačí kopírovat původní posloupnost zprava doleva až po první jedničku. Zbývající bity poté invertujeme, abychom získali výsledný bitový vzor (obrázek 1.22).

Díky znalosti základních vlastností systémů dvojkového doplňku můžeme také dospět k algoritmu na dekódování reprezentací ve dvojkovém doplňku. Pokud má dekódovaný vzor znaménkový bit s hodnotou 0, stačí přečíst hodnotu, jako by se jednalo o běžnou binární reprezentaci. Posloupnost bitů 0110 například znamená hodnotu 6, protože 110 je binární zápis čísla 6. Jestliže potřebujeme dekódovat vzor se znaménkovým bitem 1, vycházíme z toho, že reprezentované číslo je záporné. Musíme jen zjistit jeho absolutní hodnotu. Přitom použijeme postup „kopírování a invertování“ z obrázku 1.22 a získaný vzor následně dekódujeme stejně, jako by se jednalo o prostou binární reprezentaci. Chceme-li například dekódovat posloupnost 1010, nejdříve si uvědomíme, že se vzhledem ke znaménkovému bitu 1 jedná o zápornou hodnotu. Metodou „kopírování a invertování“ tedy získáme vzor 0110, vyhledáme, že jde o binární reprezentaci čísla 6, a z toho odvodíme, že původní vzor znamenal číslo –6.

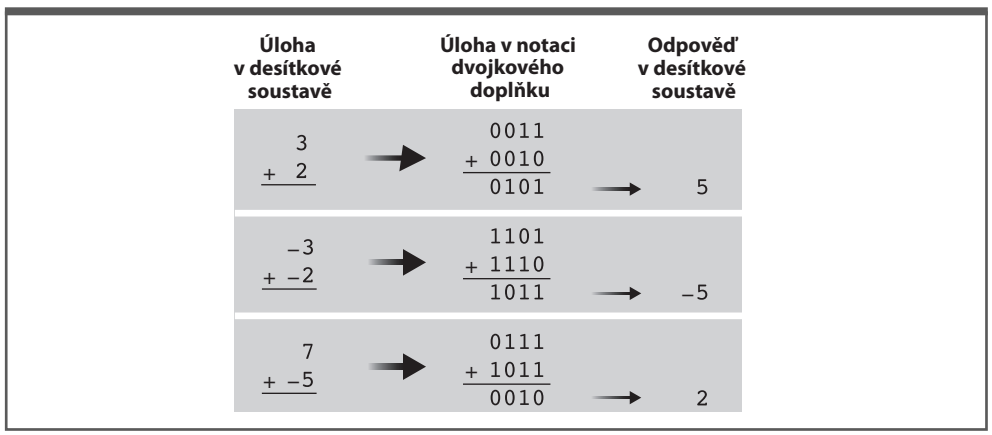


Obrázek 1.22: Zakódování hodnoty –6 ve dvojkovém doplňku se čtyřmi bity

Sčítání ve dvojkovém doplňku – Chceme-li sečíst hodnoty vyjádřené ve dvojkovém doplňku, uplatníme stejný algoritmus, který jsme použili při binárním sčítání. Jediný rozdíl spočívá v tom, že všechny bitové vzory včetně výsledku mají nyní stejnou délku. To znamená, že při sčítání ve dvojkovém doplňku je nutné zahodit každý dodatečný bit, který se objeví při posledním přenosu nalevo od první číslice výsledku. „Sečtením" posloupností 0101 a 0010 tedy dostaneme 0111 a „součet" sekvencí 0111 a 1011 dává výsledek 0010 (0111 + 1011 = 10010, což je nutné zkrátit na 0010).

Pomocí těchto pravidel můžeme vyřešit tři úlohy sčítání na obrázku 1.23. Ve všech případech převedeme zadání do dvojkového doplňku (s použitím posloupností s délkou čtyř bitů), sečteme výše popsaným postupem a dekódujeme výsledek zpět do běžného zápisu se základem deset.

Jak si můžeme všimnout, třetí problém na obrázku 1.23 vyžaduje přičíst kladné číslo k zápornému. Zde se ukazuje hlavní výhoda dvojkového doplňku: libovolnou kombinaci čísel se znaménkem lze sečíst pomocí stejného algoritmu a tedy i stejného elektronického obvodu. Počítače tedy při aritmetických operacích postupují úplně jinak než lidé. Žáci základní školy se nejdříve učí sčítat a poté odečítat. Na druhé straně počítač, který dokáže pracovat s dvojkovým doplňkem, si vystačí se sčítáním.



Obrázek 1.23: Úlohy sčítání převedené do dvojkového doplňku

Úloha odčítání 7 – 5 je například ekvivalentní sčítání 7 + (–5). Pokud by tedy počítač dostal úkol odečíst číslo 5 (uloženo jako 0101) od čísla 7 (uloženo jako 0111), nejdříve by převedl číslo 5 na –5 (reprezentováno jako 1011) a následně by sečtením vzorů 0111 + 1011 získal posloupnost 0010, která odpovídá číslu 2:

```
 7          0111          0111
-5    →   - 0101    →   + 1011
                         0010    →    2
```

Vidíme tedy, že když se číselné hodnoty reprezentují pomocí dvojkového doplňku, stačí ke sčítání i odečítání jediný obvod pro sčítání, který je doplněn obvodem na inverzi bitů. (Obvody tohoto typu jsou popsány v Příloze B.)

Problém přetečení – v předchozích příkladech jsme obešli jeden problém. Každý systém dvojkového doplňku má omezený rozsah hodnot, které umí reprezentovat. Při použití dvojkového doplňku se čtyřbitovými vzory lze vyjádřit nejvyšší celé číslo 7 a nejmenší záporné číslo -8. Není možné popsat například hodnotu 9, což znamená, že nedokážeme získat správnou odpověď na zadání 5 + 4. V tomto případě by

se například zobrazil výsledek –7. Tento jev se označuje jako **přetečení** (overflow). Přetečení tedy nastává, když výpočet vede k hodnotě, která nepatří do rozsahu hodnot použité notace. U dvojkového doplňku k tomu může dojít při sčítání dvou kladných nebo dvou záporných hodnot. V každém případě lze přetečení zjistit kontrolou znaménkového bitu odpovědi. Přetečení se projeví tím, že součet dvou kladných čísel poskytne vzor odpovídající záporné hodnotě, nebo že součet dvou záporných hodnot je zdánlivě kladný.

Vzhledem k tomu, že většina počítačů používá systémy dvojkového doplňku s mnohem delšími bitovými vzory, než jaké jsme ukázali v příkladech výše, je možné bez rizika přetečení manipulovat i s většími hodnotami. V současnosti se hodnoty ve dvojkovém doplňku běžně ukládají pomocí vzorů s 32 bity, které umožňují bez výskytu přetečení dosáhnout kladných hodnot do 2 147 483 647. Jestliže je potřeba počítat ještě s většími hodnotami, lze použít delší bitové vzory nebo změnit měrnou jednotku. Pokud například řešení úlohy místo v centimetrech vyjádříme v kilometrech, můžeme pracovat s menšími čísly a přitom často zachovat dostatečnou přesnost.

Jak je zřejmé, chybovat mohou i počítače. Operátor počítače si tedy musí příslušná rizika uvědomovat. Problém spočívá v tom, že programátoři a uživatelé počítačů ztratili opatrnost a ignorují fakt, že součtem mnoha malých čísel může být značně velké číslo. V minulosti se například k reprezentaci hodnot ve dvojkovém doplňku běžně používaly vzory s 16 bity. To znamenalo, že přetečení nastalo již při dosažení hodnoty $2^{15} = 32\,768$. Dne 19. září 1989 náhle selhal počítačový systém nemocnice, který do té doby léta fungoval spolehlivě. Podrobná analýza ukázala, že datum leželo 32 768 dní po 1. lednu 1900 a počítač byl naprogramován tak, aby určoval aktuální datum podle tohoto počátečního data. Vzhledem k přetečení se tedy 19. září 1989 objevila záporná hodnota a počítačový program na takovou situaci nebyl připraven.

Zápis s posunem

Další metoda reprezentace celočíselných hodnot se nazývá **zápis s posunem** (excess notation). Stejně jako u dvojkového doplňku i u zápisu s posunem platí, že všechny hodnoty jsou vyjádřeny bitovým vzorem stejné délky. Chceme-li vytvořit systém s posunem, nejdříve zvolíme délku jeho vzoru a poté zapíšeme všechny kombinace bitů dané délky v pořadí, v jakém by po sobě následovaly při počítání ve dvojkové soustavě. První vzor, jehož nejvýznamnější bit má hodnotu 1, nalezneme přibližně v polovině seznamu. Tento vzor bude označovat nulu a vzory následující po něm budou reprezentovat čísla 1, 2, 3... a předcházející vzory přiřadíme číslům –1, –2, –3... Výsledný kód, který zahrnuje posloupnosti délky čtyř bitů, je znázorněn na obrázku 1.24. Jak je zřejmé, hodnotu 5 reprezentuje vzor 1101 a hodnota –5 je vyjádřena jako 0011. (Rozdíl mezi systémem s posunem a systémem dvojkového doplňku zjevně spočívá v tom, že znaménkové bity mají opačnou hodnotu.)

Systém, který vidíme na obrázku 1.24, se označuje jako zápis s posunem osm. Smyslu tohoto názvu porozumíme, když interpretujeme všechny vzory v kódu pomocí tradičního binárního systému a poté výsledky porovnáme s hodnotami, které

zápis s posunem reprezentuje. Ve všech případech zjistíme, že binární interpretace odpovídá číslu, které je oproti notaci s posunem větší právě o hodnotu 8. Vzor 1100 v binárním zápisu například patří hodnotě 12, ale v systému s posunem označuje hodnotu 4. Jiný příklad: posloupnost 0000 v binárním zápisu znamená číslo 0, ale v systému s posunem se jedná o ekvivalent čísla -8. Systém s posunem, který je založen na vzorech délky pěti bitů, bychom obdobně nazvali zápis s posunem šestnáct, protože například vzor 10000 by namísto hodnoty 16, jako je tomu obvykle, reprezentoval nulu. Podobně si můžeme ověřit, že v systému s posunem, který obsahuje tříbitové vzory, se bude jednat o notaci s posunem čtyři (obrázek 1.25).

Bitové schéma	Reprezentovaná hodnota
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	-1
0110	-2
0101	-3
0100	-4
0011	-5
0010	-6
0001	-7
0000	-8

Obrázek 1.24: Převodní tabulka notace s posunem osm

Bitové schéma	Reprezentovaná hodnota
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

Obrázek 1.25: Systém notace s posunem, který používá vzory délky tří bitů

Otázky a cvičení

1. Převeďte každou z následujících reprezentací ve dvojkovém doplňku na odpovídající číslo se základem deset:

   a. 00011 **b.** 01111 **c.** 11100
   d. 11010 **e.** 00000 **f.** 10000

2. Převeďte každou z následujících číselných reprezentací se základem deset do dvojkového doplňku s délkou 8 bitů:

   a. 6 **b.** –6 **c.** –17
   d. 13 **e.** –1 **f.** 0

3. Předpokládejme, že následující bitové posloupnosti představují hodnoty uložené ve dvojkovém doplňku. Najděte ve dvojkovém doplňku reprezentaci opačného čísla každé z těchto hodnot:

   a. 00000001 **b.** 01010101 **c.** 11111100
   d. 11111110 **e.** 00000000 **f.** 01111111

4. Předpokládejme, že počítač ukládá čísla ve dvojkovém doplňku. Jaká největší a nejmenší čísla lze uložit, pokud počítač používá bitové vzory s následující délkou?

   a. čtyři **b.** šest **c.** osm

5. Každý bitový vzor v následujících úlohách představuje hodnotu uloženou ve dvojkovém doplňku. Najděte odpověď jednotlivých úloh postupem sčítání ve dvojkovém doplňku, který jsme popsali v textu. Potom zkontrolujte své výpočty tak, že zadání i svůj výsledek převedete do zápisu se základem deset.

   a. 0101 + 0010 **b.** 0011 + 0001 **c.** 0101 + 1010 **d.** 1110 + 0011 **e.** 1010 + 1110

6. Vyřešte následující úlohy ve dvojkovém doplňku, ale tentokrát kontrolujte přetečení a označte odpovědi, které kvůli tomuto jevu nejsou správné.

   a. 0100 + 0011 **b.** 0101 + 0110 **c.** 1010 + 1010 **d.** 1010 + 0111 **e.** 0111 + 0001

7. Převeďte všechny následující úlohy ze zápisu se základem deset do dvojkového doplňku s použitím posloupností délky čtyř bitů, poté převeďte každou úlohu na ekvivalentní problém sčítání (jak by postupoval počítač) a čísla sečtěte. Zkontrolujte své odpovědi tak, že je zpětně převedete do zápisu se základem deset.

   a. 6 −(−1) **b.** 3 −2 **c.** 4 −6 **d.** 2 −(−4) **e.** 1 −5

8. Může teoreticky dojít k přetečení při sčítání jedné kladné a jedné záporné hodnoty ve dvojkovém doplňku? Vysvětlete svou odpověď.

9. Převeďte každou z následujících reprezentací v zápisu s posunem osm na odpovídající číslo se základem deset, aniž byste použili tabulku v textu:

   a. 1110 **b.** 0111 **c.** 1000
   d. 0010 **e.** 0000 **f.** 1001

10. Převeďte každé z následujících čísel se základem deset na odpovídající zápis s posunem osm, aniž byste použili tabulku v textu:

a. 5 **b.** –5 **c.** 3
d. 0 **e.** 7 **f.** –8

11. Lze v zápisu s posunem osm reprezentovat hodnotu 9? Jak je tomu u reprezentace čísla 6 v notaci s posunem čtyři? Vysvětlete svou odpověď.

1.7: Ukládání zlomků

Na rozdíl od ukládání celých čísel nestačí při ukládání hodnot se zlomkovou částí zaznamenat vzor nul a jedniček, který udává binární reprezentaci čísla, ale je potřeba uložit také polohu řádové čárky. Přitom se často používá varianta vědeckého zápisu,[3] která se označuje jako zápis s **plovoucí desetinnou čárkou** (floating-point notation).

Zápis s plovoucí řádovou čárkou

Zápis s plovoucí desetinnou čárkou vysvětlíme na systému, který uchovává čísla pouze v jediném bajtu. Počítače obvykle pracují s mnohem delšími posloupnostmi, ale tento osmibitový formát zastoupí skutečné systémy a umožní ukázat důležité principy, aniž bychom museli složitě počítat s dlouhými bitovými řadami.

Nejdříve označíme bit s nejvyšším řádem v bajtu jako znaménkový bit. Opět platí, že hodnota 0 znaménkového bitu znamená, že uložená hodnota není záporná, a hodnota 1 označuje zápornou hodnotu. Poté rozdělíme zbývajících 7 bitů bajtu do dvou skupin neboli polí: **pole exponentu** a **pole mantisy**. Vyhraďme tedy 3 bity za znaménkovým bitem pro pole exponentu a zbývající 4 bity ponechejme jako pole mantisy. Způsob rozdělení bajtu je znázorněn na obrázku 1.26.

Význam polí je možné ukázat na následujícím příkladu. Předpokládejme, že bajt obsahuje bitový vzor 01101011. Když analyzujeme tento vzor pomocí výše uvedeného formátu, vidíme, že znaménkový bit má hodnotu 0, exponent je roven 110 a mantisa 1011. Chceme-li tento bajt dekódovat, nejdříve extrahujeme mantisu a umístíme řádovou čárku na její levou stranu. Dostaneme[4]

```
,1011
```

Dále extrahujeme obsah pole exponentu (110) a interpretujeme je jako celé číslo, které je uloženo tříbitovou metodou s posunem (opět viz obrázek 1.25). Vzor v poli exponentu našeho příkladu tedy označuje kladné číslo 2. To znamená, že řádovou čárku řešení je potřeba posunout o dva bity doprava. (Záporný exponent by ukazoval, že je nutné řádovou čárku posunout doleva.) Proto získáváme

```
10,11
```

což je binární reprezentace čísla $2\frac{3}{4}$. V další fázi si povšimneme, že znaménkový bit v našem příkladu má hodnotu 0. Reprezentovaná hodnota tedy není záporná. Došli jsme k závěru, že bajt 01101011 označuje číslo $2\frac{3}{4}$. Kdyby se jednalo o vzor 11101011 (který se liší pouze znaménkovým bitem), rovnala by se reprezentovaná hodnota $-2\frac{3}{4}$.

3 V češtině se zpravidla používá označení „semilogaritmický tvar".
4 Je-li před řádovou čárkou pouze 0, vynecháme ji.

Jako další příklad uveďme bajt 00111100. Extrakcí mantisy dostaneme

,1100

a přesuneme řádovou čárku o 1 bit doleva, protože pole exponentu (011) představuje hodnotu –1. Máme tedy

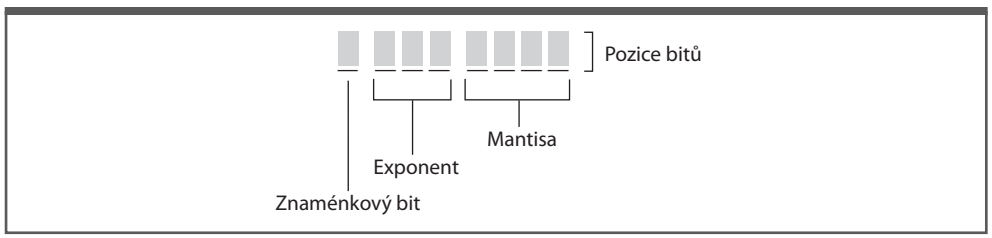
,01100

což odpovídá hodnotě $^3/_8$. Vzhledem k tomu, že původní vzor má znaménkový bit 0, kóduje nezápornou hodnotu. Usoudíme, že vzor 00111100 reprezentuje číslo $^3/_8$.

Chceme-li uložit hodnotu pomocí zápisu s plovoucí řádovou čárkou, obrátíme předchozí postup. Jestliže například potřebujeme zakódovat číslo $1^1/_8$, nejdříve je vyjádříme ve dvojkové soustavě a získáme 1,001. Dále zkopírujeme bitový vzor do pole mantisy zleva doprava. Přitom začínáme jedničkou, která se v binární reprezentaci nachází zcela vlevo. Bajt aktuálně vypadá takto:

_ _ _ _ 1 0 0 1

Nyní musíme zaplnit pole exponentu. Za tímto účelem si představíme, že obsah pole mantisy má řádovou čárku z levé strany, a zjistíme, o kolik bitů a v jakém směru je nutné řádovou čárku posunout, abychom získali původní binární číslo.



Obrázek 1.26: Součásti zápisu s plovoucí desetinnou čárkou

V našemu příkladu vidíme, že řádovou čárku ve výrazu ,1001 je nutné posunout o 1 bit doprava, abychom dostali 1,001. Exponent by tedy měl být kladný. Do pole exponentu proto zapíšeme 101 (což je kladná jednička v notaci s posunem čtyři – viz obrázek 1.25). Nakonec nastavíme znaménkový bit na hodnotu 0, protože ukládaná hodnota není záporná. Výsledný bajt vypadá takto:

0 1 0 1 1 0 0 1

Při vyplňování pole mantisy je nutné dodržet pravidlo, které požaduje kopírovat bitový vzor z binární reprezentace zleva doprava počínaje jedničkou zcela vlevo. Vysvětlíme to na příkladu uložení hodnoty $^3/_8$, která má v binárním zápisu tvar ,011. V tomto případě bude mít mantisa hodnotu

_ _ _ _ 1 1 0 0

Nebude

_ _ _ _ 0 1 1 0

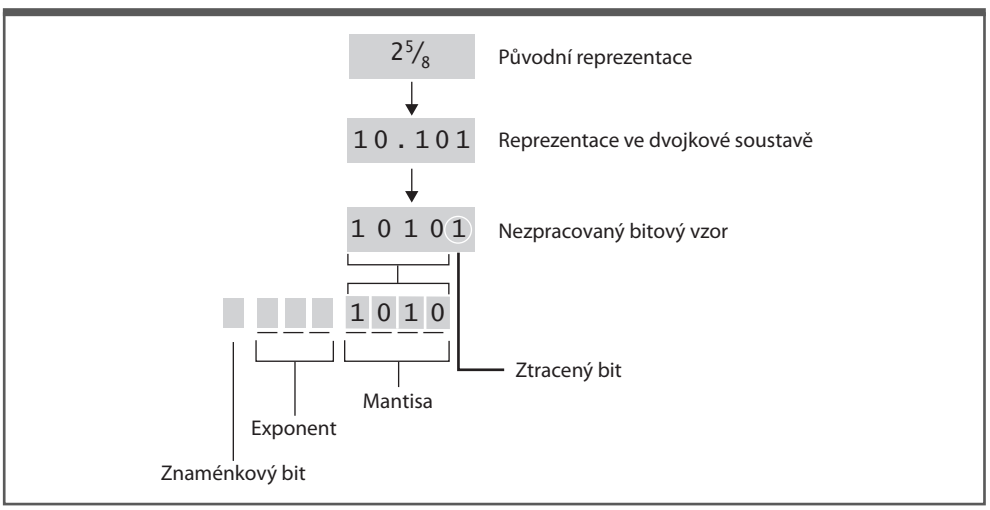
To vyplývá z faktu, že pole mantisy plníme *od jedničky, která se v* binární reprezentaci nachází na levé straně. Zápisy, které vyhovují uvedenému pravidlu, se označují jako zápisy v **normalizovaném tvaru**.

Díky normalizovanému tvaru má každá hodnota pouze jedinou reprezentaci. Například zápisy 00111100 i 01000110 bychom mohli dekódovat na hodnotu $^3/_8$, ale pouze první vzor je v normalizovaném tvaru. Soulad s normalizovaným tvarem také

znamená, že mantisy reprezentací všech nenulových hodnot budou začínat číslicí 1. Hodnota nula však představuje zvláštní případ: bitový vzor nuly v zápisu s plovoucí desetinnou čárkou obsahuje pouze číslice 0.

Chyby vzniklé odříznutím části čísla

Nyní se budeme zabývat nepříjemným problémem, který se objeví, pokusíme-li se v našem jednobajtovém systému s plovoucí řádovou čárkou uložit hodnotu $2\frac{5}{8}$. Nejdříve číslo $2\frac{5}{8}$ zapíšeme ve dvojkové soustavě a dostaneme 10,101. Když však tuto hodnotu přeneseme do pole mantisy, narazíme na nedostatek místa. Číslice 1 na pravém konci (která reprezentuje poslední $\frac{1}{8}$) se proto ztratí (viz obrázek 1.27). Budeme-li tento problém prozatím ignorovat a budeme pokračovat ve vyplňování pole exponentu a znaménkového bitu, získáme bitový vzor 01101010, který označuje číslo $2\frac{1}{2}$ a nikoli $2\frac{5}{8}$. Došlo zde k takzvané **chybě při odříznutí části čísla** (truncation error) neboli **zaokrouhlovací chybě** (round-off error). Při této chybě se část ukládané hodnoty ztratí, protože pole mantisy není dostatečně dlouhé.



Obrázek 1.27: Zakódování hodnoty $2\frac{5}{8}$

Vliv těchto chyb lze omezit pomocí delšího pole mantisy. Ve skutečnosti většina dnešních počítačů ukládá hodnoty v zápisu s plovoucí desetinnou čárkou do záznamů s délkou alespoň 32 bitů, nikoli osm jako v našem příkladu. Tyto delší záznamy dovolují zároveň prodloužit i pole exponentu. I u těchto delších formátů se však občas stává, že jejich přesnost nedostačuje.

Chyby při zkrácení se také objevují kvůli jevu, který již dobře známe z desítkové soustavy. Jedná se o problém periodických čísel, se kterým se například setkáme, když se pokusíme v desítkové soustavě vyjádřit zlomek $\frac{1}{3}$. Některé hodnoty nelze přesně vyčíslit bez ohledu na to, kolik číslic použijeme. Rozdíl mezi naším tradičním zápisem se základem deset a binárním zápisem spočívá v tom, že ve dvojkové soustavě má periodický rozvoj mnohem více hodnot než v desítkové soustavě. Ve dvojkové soustavě například nelze přesně zapsat hodnotu jedné desetiny. Představme si, jaké potíže to může způsobit neznalé osobě, která se v zápisu s plovoucí řádovou čárkou pokouší ukládat finanční částky a zpracovávat je. Jestliže jako měrnou jednotku

zvolíme korunu, nedokážeme přesně uložit hodnotu deseti haléřů. Řešení v tomto případě spočívá v manipulaci s daty v jednotkách haléřů. Všechny hodnoty pak lze vyjádřit pomocí celých čísel, která je možné přesně ukládat metodami typu dvojkového doplňku.

Chybami způsobenými odříznutím a souvisejícími problémy se neustále zabývají lidé, kteří pracují v oblasti numerické analýzy. Toto odvětví matematiky se zaměřuje na otázky praktických výpočtů, které často bývají mimořádně rozsáhlé a vyžadují značnou přesnost.

Uveďme příklad, který by se jistě líbil každému numerickému analytikovi. Předpokládejme, že máme za úkol sečíst v našem jednobajtovém zápisu s plovoucí řádovou čárkou, který jsme definovali výše, následující tři hodnoty:

$$2^1/_2 + {}^1/_8 + {}^1/_8$$

Jestliže budeme tyto hodnoty sčítat v pořadí, v němž jsou uvedeny, nejdříve přičteme číslo $2^1/_2$ k číslu $^1/_8$ a získáme mezivýsledek $2^5/_8$, který lze binárně zapsat jako 10,101. Vzhledem k tomu, že tuto hodnotu nelze uložit přesně (jak jsme již ukázali výše), bude výsledek tohoto sčítání nakonec uložen jako $2^1/_2$ (což se rovná jednomu ze sčítanců). V dalším kroku potřebujeme k této hodnotě přičíst $^1/_8$. Zde se opět vyskytne chyba způsobená odříznutím a konečný výsledek dá nesprávnou odpověď $2^1/_2$.

Nyní se pokusme hodnoty sečíst v opačném pořadí. Nejdříve přičteme $^1/_8$ k $^1/_8$ a dostaneme $^1/_4$. Ve dvojkové soustavě lze toto číslo zapsat jako 0,01. Výsledek prvního kroku tedy můžeme přesně uchovat v jednom bajtu 00111000. Dále tuto hodnotu $^1/_4$ sečteme s další hodnotou v původním seznamu $2^1/_2$ a výsledkem je číslo $2^3/_4$, které můžeme přesně uložit jako bajt 01101011. Tentokrát jsme dospěli ke správnému řešení.

Když to shrneme, platí, že při sčítání číselných hodnot reprezentovaných v zápisu s plovoucí řádovou čárkou může být důležité, v jakém pořadí hodnoty sčítáme. Při sčítání čísel, jejichž řády se hodně liší, může dojít k odříznutí malého čísla. Jestliže tedy potřebujeme sečíst více hodnot, řídíme se obecným pravidlem, které doporučuje nejdříve spolu sečíst menší hodnoty. Doufáme přitom, že tato čísla v úhrnu dají hodnotu, která bude při sčítání s většími hodnotami dostatečně významná. Právě tento jev jsme pozorovali v předchozím příkladu.

Zápis s plovoucí řádovou čárkou a s jednoduchou přesností

Zápis s plovoucí řádovou čárkou, který jsme si představili v této kapitole (část 1.7), je příliš primitivní, než aby ho bylo možné používat ve skutečných počítačích. S pouhými 8 bity totiž dokáže vyjádřit pouze 256 hodnot z celé osy všech reálných čísel. Systém s osmi bity jsme zvolili proto, abychom příklady příliš nekomplikovali a přesto mohli ukázat důležité základní principy.

Mnohé dnešní počítače podporují 32bitovou formu tohoto zápisu, který se označuje jako **zápis s plovoucí řádovou čárkou a s jednoduchou přesností** (Single Precision Floating Point). Tento formát používá 1 bit pro znaménko, 8 bitů pro exponent (v zápisu s posunem) a 23 bitů pro mantisu. Zápis s plovoucí řádovou čárkou a s jednoduchou přesností tedy umožňuje vyjádřit značně velká čísla (v řádu 10^{38}) a zároveň velmi malá čísla (řádově 10^{-37}) s přesností na 7 desetinných míst. To znamená, že lze s poměrně vysokou přesností uložit prvních 7 číslic daného desetinného čísla (může ovšem zbývat malá chyba). Všechny číslice za prvními sedmi budou nevyhnutelně ztraceny kvůli chybě způsobené odříznutím (ačkoli zůstane zachován řád čísla). Další forma s názvem **zápis s plovoucí řádovou čárkou a s dvojitou přesností** (Double Precision Floating Point) používá 64 bitů a poskytuje přesnost na 15 desetinných míst.

Autoři moderních počítačových programů se účinně snaží nepoučené uživatele před potížemi tohoto typu chránit. V běžném tabulkovém procesoru dostaneme chybnou odpověď při sčítání jen v případech, kdy se sčítané hodnoty liší v řádu 10^{16} nebo více. Pokud se tedy rozhodneme přičíst jednotku k číslu

```
10 000 000 000 000 000
```

můžeme dostat výsledek

```
10 000 000 000 000 000
```

namísto správné hodnoty

```
10 000 000 000 000 001
```

Tyto problémy je nutné ošetřit v aplikacích (jako jsou navigační systémy), kde se drobné chyby mohou kumulovat v dalších výpočtech a způsobit významné odchylky. Běžným uživatelům počítačů však přesnost, kterou nabízí většina komerčních programů, postačuje.

Otázky a cvičení

1. Dekódujte následující bitové posloupnosti na základě formátu s plovoucí řádovou čárkou, který jsme vysvětlili v textu:
   a. `01001010` **b.** `01101101` **c.** `00111001` **d.** `11011100` **e.** `10101011`
2. Zakódujte následující hodnoty do formátu s plovoucí řádovou čárkou (viz text výše). Označte výskyt chyb způsobených odříznutím.
   a. $2\frac{3}{4}$ **b.** $5\frac{1}{4}$ **c.** $\frac{3}{4}$ **d.** $-3\frac{1}{2}$ **e.** $-4\frac{3}{8}$
3. Uvažujeme-li formát s plovoucí řádovou čárkou popsaný v textu, který ze vzorů 01001001 a 00111101 představuje větší hodnotu? Popište jednoduchý postup, jak určit, který ze dvou vzorů reprezentuje větší číslo.
4. Jakou největší hodnotu je možné vyjádřit pomocí formátu s plovoucí řádovou čárkou, který je popsán v předchozím textu? Jaké nejmenší kladné číslo lze vyjádřit?

1.8: Komprese dat

Při ukládání nebo přenosu dat je často užitečné (a někdy dokonce nutné) omezit velikost zpracovávaných dat a přitom zachovat obsaženou informaci. Příslušná metoda se nazývá **komprese dat** (data compression). Tuto část zahájíme přehledem některých obecných metod komprese dat. Poté se budeme zabývat některými speciálními přístupy.

Obecné metody komprese dat

Schémata komprese dat patří do dvou kategorií. Některé jsou **bezeztrátové** (lossless) a jiné **ztrátové** (lossy). U bezeztrátových schémat nedochází při kompresi ke ztrátě informací. Ztrátová schémata oproti tomu mohou ztrátu části informací způsobit. Ztrátové metody často poskytují účinnější kompresi než bezeztrátové a jsou tedy

oblíbené v oblastech, kde lze tolerovat drobné chyby, což platí například u obrázků a zvuku.

V situacích, kde komprimovaná data obsahují dlouhé sekvence stejných hodnot, se často nasazuje metoda komprese s názvem **RLE** (run-length encoding, kódování podle délky). Jedná se postup, kdy se posloupnosti stejných datových prvků nahrazují kódem, který udává opakovaný element a počet jeho následných opakování. Bitová posloupnost, která se skládá z 253 jedniček následovaných 118 nulami a poté 87 jedničkami, například v kódování RLE zabírá mnohem méně místa než původních 458 bitů.

Další metoda bezeztrátové komprese dat se označuje jako **frekvenční kódování** (frequency-dependent encoding). Délka bitového vzoru, který reprezentuje určitou datovou položku, v tomto systému nepřímo závisí na frekvenci používání dané položky. Tyto kódy představují příklady kódů s proměnnou délkou. Jednotlivé položky jsou tedy zastoupeny vzory s odlišnými délkami; stojí v protikladu ke kódům, jako je Unicode, kde jsou všechny symboly vyjádřeny pomocí 16 bitů. Algoritmus, který se běžně používá při vývoji frekvenčních kódů, původně vyvinul David Huffman. Kódy vytvořené tímto způsobem se proto běžně označují jako **Huffmanovy kódy**. Většina v současnosti používaných frekvenčních kódů tedy patří mezi Huffmanovy kódy.

Jako příklad frekvenčních kódování můžeme uvést zakódování textu v přirozeném jazyce. Písmena *e*, *o*, a a i se v češtině vyskytují mnohem častěji než písmena *x*, *w* a *q*. Když tedy tvoříme kód pro český text, můžeme ušetřit místo, když budeme běžnější písmena reprezentovat kratšími bitovými vzory a pro méně obvyklá písmena zvolíme delší bitové vzory. Tímto způsobem dostaneme kód, který dokáže český text zakódovat pomocí menšího počtu bitů, než by bylo možné u kódu s jednotnou délkou znaků.

V některých případech se datový proud pro kompresi skládá z jednotek, které se navzájem liší jen mírně. Příkladem mohou být následná políčka filmu. V těchto situacích jsou užitečné metody, které používají **relativní kódování** (zvané také **rozdílové kódování**). Tyto metody nezaznamenávají celé následné datové jednotky, ale pouze rozdíly mezi nimi. Každá jednotka se tedy kóduje ve vztahu k předchozí jednotce. Relativní kódování lze implementovat v bezeztrátové nebo ztrátové formě. Závisí to na tom, zda jsou rozdíly mezi následnými datovými jednotkami zakódovány přesně, nebo pouze přibližně.

Jiné populární systémy komprese jsou založeny na metodách **slovníkového kódování**. Termín *slovník* (dictionary) zde označuje kolekci stavebních bloků, z nichž je možné sestavit komprimovanou zprávu. Samotná zpráva je zakódována jako posloupnost odkazů na slovník. Systémy slovníkového kódování si obvykle představujeme jako bezeztrátové. Jak si ale ukážeme v rozboru komprese obrázků, v některých případech jsou položky slovníku pouhým přiblížením ke správným datovým prvkům, takže dostáváme systém ztrátové komprese.

Slovníkové kódování lze použít při kompresi textových dokumentů v textových procesorech. Slovníky, které se v těchto programech již používají ke kontrole pravopisu, totiž představují vynikající kompresní slovníky. Ve skutečnosti můžeme celé slovo zakódovat jako jediný odkaz na tento slovník, místo abychom jej vyjadřovali jako posloupnost jednotlivých znaků zakódovaných pomocí systému typu ASCII nebo Unicode. Běžný slovník v textovém procesoru obsahuje přibližně 25 000 položek. To znamená, že jednotlivý záznam lze identifikovat jako celé číslo z intervalu od 0 do 24 999. Konkrétní položku ve slovníku je tedy možné identifikovat pomocí

posloupnosti pouhých 15 bitů. Jestliže předpokládáme, že odkazované slovo obsahuje šest písmen, vyžadovalo by oproti tomu jeho zakódování po jednotlivých znacích 48 bitů v případě osmibitového kódování ASCII nebo 96 bitů při použití standardu Unicode.

Variantou slovníkového kódování je **adaptivní slovníkové kódování** (známé také pod názvem dynamické slovníkové kódování). V systému adaptivního slovníkového kódování se slovník může během kódování měnit. Často se používá například kódování **LZW (Lempel-Ziv-Welsh)**, které získalo název podle příjmení svých autorů (Abraham Lempel, Jacob Ziv a Terry Welsh). Při kódování zprávy vychází algoritmus LZW od slovníku se základními stavebními kameny, ze kterých je zpráva sestavena. Když jsou však ve zprávě nalezeny větší jednotky, algoritmus je přidá do slovníku. To znamená, že každý budoucí výskyt těchto jednotek je možné zakódovat jako jediný odkaz do slovníku namísto více odkazů na základní jednotky. Při kódování českého textu lze například začít od slovníku, který obsahuje jednotlivá písmena, číslice a interpunkční znaménka. Jak se však ve zprávě objevují jednotlivá slova, lze je přidat do slovníku. Při kódování zprávy se tedy slovník zvětšuje. Jak slovník roste, je možné kódovat více slov (nebo opakované posloupnosti slov) ve zprávě jako jediné odkazy na slovník.

Výsledkem bude zpráva zakódovaná pomocí poměrně velkého slovníku, který je pro danou zprávu jedinečný. Při dekódování zprávy však nemusí být tento velký slovník k dispozici. Stačí pouze původní malý slovník. Proces dekódování může ve skutečnosti začít se stejným malým slovníkem, na kterém byl založen proces kódování. V průběhu dekódování pak algoritmus zjistí stejné jednotky jako během kódování a dokáže je proto přidat do slovníku pro budoucí potřebu stejným způsobem, jaký se uplatňuje v procesu kódování.

Ukážeme si to na postupu kódování následující zprávy algoritmem LZW:

xyx xyx xyx xyx

kdy začínáme se slovníkem, který obsahuje tři položky. První z nich je *x*, druhou *y* a třetí mezera. Nejdříve zakódujeme řetězec *xyx* jako 121. To znamená, že zpráva začíná vzorem, který obsahuje první položku slovníku následovanou druhou položkou a poté opět položkou první. Následně je zakódována mezera a dostáváme posloupnost 1213. Když jsme však narazili na mezeru, víme, že předchozí řetězec znaků tvoří slovo. Přidáme tedy vzor *xyx* do slovníku jako čtvrtou položku. Když budeme pokračovat tímto způsobem, zakódujeme celou zprávu jako 121343434.

Jestliže bychom nyní dostali úkol zprávu dekódovat pomocí původního slovníku se třemi položkami, nejdříve bychom dekódovali počáteční posloupnost 1213 na řetězec *xyx* následovaný mezerou. V této fázi bychom si povšimli, že řetězec *xyx* tvoří slovo, a přidali bychom jej do slovníku jako čtvrtý záznam – stejně jako během procesu kódování. Při dekódování zprávy bychom dále zjistili, že hodnota 4 ve zprávě odkazuje na tuto novou čtvrtou položku, a dekódovali bychom ji na slovo *xyx*. Dostali bychom tedy posloupnost

xyx xyx

Uvedeným postupem bychom řetězec 121343434 nakonec kompletně dekódovali jako

xyx xyx xyx xyx

což je původní zpráva.

Komprese obrázků

V části 1.4 jsme viděli, jak lze zakódovat obrázky pomocí metod založených na bitových mapách. Generované bitové mapy však bohužel bývají značně velké. Vzniklo proto mnoho kompresních schémat, která jsou určena speciálně k popisu obrázků.

Jeden systém označovaný jako **GIF** (akronym slov Graphic Interchange Format, vyslovuje se zpravidla „gif“) je systém slovníkového kódování, který vyvinula společnost CompuServe. Problém komprese zjednodušuje tím, že umožňuje přiřadit pixelu jednu z nejvýše 256 barev. Kombinace červené, zelené a modré složky každé z těchto barev je zakódována pomocí tří bajtů a těchto 256 kombinací je uloženo do tabulky (slovníku) označované jako paleta. Každý pixel obrázku lze poté reprezentovat jediným bajtem, jehož hodnota určuje, která z 256 položek v paletě popisuje barvu daného pixelu. (Připomeňme, že jediný bajt může obsahovat libovolný z 256 odlišných bitových vzorů.) Je potřeba poznamenat, že pokud se kódování GIF aplikuje na náhodné obrázky, jedná se o systém ztrátové komprese, protože barvy v paletě nemusí být shodné s barvami v původním obrázku.

Formát GIF dále zlepšuje kompresi tím, že tento jednoduchý slovníkový systém rozšiřuje o adaptivní slovníkový systém založený na metodě LZW. Konkrétně přidává tento algoritmus do slovníku vzory pixelů, které nalezne během kódování. Díky tomu lze efektivněji zakódovat jejich budoucí výskyty. Výsledný slovník proto obsahuje původní paletu a kolekci vzorů pixelů.

Jedna z barev v paletě GIF má obvykle přiřazenu „průhlednou“ hodnotu. To znamená, že v oblastech obrázku, které mají nastavenu tuto „barvu“, se může zobrazit pozadí. Díky této možnosti a zároveň své relativní jednoduchosti představuje systém GIF logickou volbu pro jednoduché animace, kdy se na obrazovce počítače střídá několik obrázků. Na druhou stranu se formát GIF vzhledem k možnosti kódovat pouze 256 barev nehodí v případech, kdy je nutné zajistit vyšší přesnost barevného podání, jako například v oblasti fotografie.

Další populární systém komprese obrázků se označuje jako **JPEG** (vyslovuje se „džejpeg“). Jedná se o standard, který vyvinula skupina **Joint Photographic Experts Group** (odtud jeho název) v rámci organizace ISO. Formát JPEG se osvědčuje při kompresi barevných fotografií a v tomto oboru se široce používá, jak je zřejmé z toho, že většina digitálních fotoaparátů má JPEG jako standardní metodu komprese.

Ve skutečnosti zahrnuje standard JPEG několik metod komprese obrázků, z nichž každá má vlastní účel. V případech, kdy je nutné zajistit maximální přesnost, poskytuje standard JPEG takzvaný bezeztrátový režim. Tento bezeztrátový režim formátu JPEG však nenabízí vysoké úrovně komprese jako jiné varianty standardu. Jiné verze formátu JPEG se navíc velmi osvědčily, takže se bezeztrátový režim JPEG používá jen zřídka. Standardní nastavení v mnoha aplikacích místo toho představuje možnost, která se označuje jako základní standard JPEG (nazývá se také ztrátový sekvenční režim JPEG).

Komprese obrázku pomocí základního standardu JPEG zahrnuje řadu kroků, z nichž některé využívají omezení lidského zraku. Lidské oko je například citlivější na změny jasu než na změny barvy. U výchozího obrázku zakódovaného pomocí hodnot luminance a chrominance tedy první krok spočívá ve zprůměrování hodnot chrominance pro čtverce velikosti dvakrát dva pixely. Rozsah informace o chrominanci se tím zmenší čtyřikrát a původní jasové informace přitom zůstanou zachovány. Výsledkem je výrazný stupeň komprese bez pozorovatelné ztráty kvality obrázku.

Dalším krokem je rozdělení obrázku na bloky velikosti osm krát osm pixelů a komprese informací po jednotlivých blocích. Přitom se aplikuje matematická metoda zvaná diskrétní kosinová transformace, jejímiž podrobnostmi se zde nebudeme zabývat. Důležité přitom je, že tato transformace převádí původní bloky o hraně osm pixelů na jiné bloky, které již neobsahují vlastní hodnoty pixelů, ale záznamy o tom, jak spolu pixely v původním bloku souvisejí. Hodnoty, které v těchto nových blocích nedosahují předem určeného prahu, jsou pak nahrazeny nulami. Vychází se přitom z faktu, že změny popsané těmito hodnotami jsou příliš malé, než aby je lidské oko zaregistrovalo. Pokud by například původní blok obsahoval šachovnicový vzor, mohl by mít nový blok jednotnou průměrnou barvu. (Blok s hranou osmi pixelů obvykle představuje velmi malou část obrázku, takže by lidské oko šachovnicovou strukturu stejně nezpozorovalo.)

V další fázi se aplikují tradičnější metody kódování RLE, relativního kódování a kódování s proměnnou délkou, které dále zvýší stupeň komprese. Kombinace jednotlivých metod zajišťuje, že základní standard JPEG obvykle komprimuje barevné obrázky alespoň desetinásobně (často až třicetinásobně) bez znatelné ztráty kvality.

Další systém komprese dat, který se uplatňuje při zpracování obrazových dat, se nazývá **TIFF** (akronym slov Tagged Image File Format). Systém TIFF se však nepoužívá primárně ke kompresi dat, ale zejména jako standardizovaný formát ukládání fotografií spolu se souvisejícími informacemi, jako je datum, čas a nastavení fotoaparátu. Za tímto účelem se vlastní obrazová data obvykle ukládají s červenými, zelenými a modrými komponentami jednotlivých pixelů a přitom se neuplatňuje žádná komprese.

Sada standardů TIFF zahrnuje metody komprese dat, které jsou většinou určeny ke kompresi obrázků textových dokumentů ve faxových aplikacích. Jsou založeny na variantách kódování RLE, které těží z toho, že textové dokumenty obsahují dlouhé posloupnosti bílých pixelů. Možnost komprese barevných obrázků, která je součástí standardů TIFF, je založena na podobných metodách jako u standardu GIF. V oblasti fotografie se tedy příliš nepoužívá.

Komprese zvuku a videa

Nejrozšířenější standardy kódování a komprese zvuku a videa vyvinula skupina **MPEG (Motion Picture Experts Group)** pod záštitou organizace ISO. Název MPEG tedy získaly i samotné standardy.

Systém MPEG zahrnuje řadu standardů pro různé aplikace. Požadavky na vysílání technologie HDTV (High Definition Television) se například liší od potřeb videokonferencí, kde musí vysílaný signál procházet různými komunikačními trasami, které mohou mít omezenou přenosovou kapacitu. Obě tyto aplikace se pak liší od ukládání videa takovým způsobem, který umožní opakované přehrávání nebo vynechávání jednotlivých částí.

Popis metod, které se uplatňují v kódování MPEG, značně přesahuje rozsah tohoto textu. Obecně řečeno však metody komprese videa vycházejí z toho, že se video skládá z řady obrázků, které připomínají políčka na filmovém pásu. Při kompresi těchto sekvencí se kompletně kódují pouze některé obrázky, které se označují jako rámce I (I-frame). Obrázky, které leží mezi rámci I, se kódují pomocí metod relativního kódování. To znamená, že místo kódování celého obrázku se zaznamenávají pouze rozdíly oproti předchozímu snímku. Samotné rámce I se zpravidla komprimují podobnými metodami, jaké využívá formát JPEG.

Nejznámější systém komprese zvuku se nazývá **MP3** a vznikl v rámci standardů MPEG. Akronym *MP3* mimochodem vznikl ze slov *MPEG layer 3*. Kromě jiných kompresních metod využívá formát MP3 vlastností lidského ucha a odstraňuje detaily, které lidé nedokáží vnímat. Jedna z těchto vlastností s názvem **časové maskování** (temporal masking) spočívá v tom, že krátce po zaznění hlasitého zvuku nedokáže lidské ucho zaznamenat tišší zvuky, které by jinak byly slyšitelné. Další vlastnost s názvem **frekvenční maskování** (frequency masking) se projevuje tím, že zvuk na jedné frekvenci zpravidla maskuje slabší zvuky na podobných frekvencích. S využitím těchto vlastností může formát MP3 dosáhnout vysoké komprese zvuku a zachovat přitom kvalitu zvuku, která se blíží CD.

Pomocí kompresních metod MPEG a MP3 mohou videokamery nahrát i hodinu videa do paměti s kapacitou 128 MB a přenosné hudební přehrávače dokáží uložit až 400 populárních skladeb v jediném gigabajtu paměti. Na rozdíl od jiných typů dat však cílem komprese zvuku a videa nutně nemusí být úspora úložného prostoru. Stejně důležité je najít kódování, které umožní dostatečně rychlý přenos informací pomocí současných komunikačních systémů, aby bylo možné zajistit plynulou prezentaci. Pokud by uložení každého rámce videa vyžadovalo jeden megabajt a k přenosu rámců by sloužila komunikační linka s přenosovou rychlostí jeden kilobajt za sekundu, nedalo by se o úspěšné videokonferenci vůbec uvažovat. Kromě dostupné kvality reprodukce se tedy systémy komprese zvuku a videa často posuzují podle přenosových rychlostí, jaké jsou potřebné k interaktivní datové komunikaci. Tyto rychlosti se obvykle měří v **bitech za sekundu – b/s** (bits per second – bps). Běžně se používají tyto násobky základní rychlosti: **kb/s** (kilobity za sekundu, tj. tisíc b/s), **Mb/s** (megabit za sekundu, tj. milion b/s) a **Gb/s** (gigabit za sekundu, tj. miliarda b/s). Díky algoritmům MPEG lze úspěšně přenášet video po komunikačních linkách, které dosahují přenosových rychlostí 40 Mb/s. Pro nahrávky MP3 zpravidla postačují přenosové rychlosti do 64 kb/s.

Otázky a cvičení

1. Uveďte čtyři obecné metody komprese.
2. Jak bude vypadat zakódovaná verze zprávy
   `xyx yxxxy xyx yxxxy yxxxy`
   při použití komprese LZW s počátečním slovníkem, který bude obsahovat znaky *x*, *y* a mezeru (popis metody naleznete v textu)?
3. Proč je formát GIF při kódování barevných komiksů vhodnější než JPG?
4. Předpokládejme, že váš tým má navrhnout vesmírnou loď, která poletí k jiným planetám a bude zpět na Zemi zasílat fotografie. Je rozumné fotografie komprimovat pomocí standardu GIF nebo základního standardu JPG, aby se omezily nároky na ukládání a přenos obrázků?
5. Jaké vlastnosti lidského oka využívá základní standard JPEG?
6. Jaké vlastnosti lidského ucha využívá formát MP3?
7. Uveďte nežádoucí jev, který se vyskytuje při kódování číselných informací, obrázků i zvuku pomocí bitových vzorů.

1.9: Komunikační chyby

Při přenosu informací mezi různými částmi počítače či ze Země na Měsíc a zpět nebo dokonce při pouhém uložení dat se může stát, že výstupní bitový vzor nebude přesně odpovídat původnímu vzoru. Kvůli částicím prachu či mastnoty na magnetickém záznamovém povrchu nebo nesprávně fungujícímu elektronickému obvodu může dojít k chybě při záznamu nebo čtení dat. Část informace může také při přenosu poškodit statická elektřina. V případě některých technologií se také mohou bitové vzory uložené v operační paměti počítače změnit působením normálního radiačního pozadí.

Pro řešení těchto problémů bylo vyvinuto mnoho kódovacích metod, které umožňují chyby zjišťovat a dokonce i opravovat. Vzhledem k tomu, že tyto metody jsou často integrovány do návrhu komponent počítačových systémů, uživatelé počítačů o nich vůbec nevědí. Přesto se jedná o značně důležité techniky, které rovněž představují významný výzkumný přínos na poli informatiky. Je tedy vhodné, abychom se nyní zabývali některými z těchto metod, na nichž závisí spolehlivost dnešních počítačových zařízení.

Paritní bity

Jednoduchá metoda detekce chyb je založena na tom, že pokud má každý zpracovávaný bitový vzor lichý počet jedniček a objeví se vzor se sudým počtem, muselo dojít k chybě. Chceme-li využít tento princip, potřebujeme systém kódování, ve kterém všechny bitové vzory obsahují lichý počet jedniček. Snadno toho dosáhneme tak, že nejdříve ke každému existujícímu vzoru v systému kódování přidáme další bit označovaný jako **paritní bit** (řekněme, že ho přidáme na stranu s nejvyšším řádem). V každém případě tomuto novému bitu přiřadíme hodnotu 1 nebo 0 tak, aby měla celá výsledná posloupnost lichý počet jedniček. Jakmile systém kódování upravíme tímto způsobem, můžeme při výskytu vzoru se sudým počtem jedniček usoudit, že nastala chyba a zpracovávaný vzor není správný.

Obrázek 1.28 ukazuje, jak lze přidat paritní bity ke kódům ASCII pro písmena A a F. Můžeme si všimnout, že kód písmene A se změní na 101000001 (paritní bit 1) a kód ASCII pro písmeno F je upraven do podoby 001000110 (paritní bit 0). Původní osmibitový vzor pro písmeno A má sice sudý počet jedniček a původní osmibitový vzor pro písmeno F obsahuje lichý počet jedniček, ale obě nové posloupnosti s devíti bity již mají lichý počet jedniček. Pokud bychom tuto metodu aplikovali na všechny kódy osmibitového standardu ASCII, získali bychom devítibitový systém kódování, v němž by na chybu ukazoval každý vzor se sudým počtem jedničkových bitů.

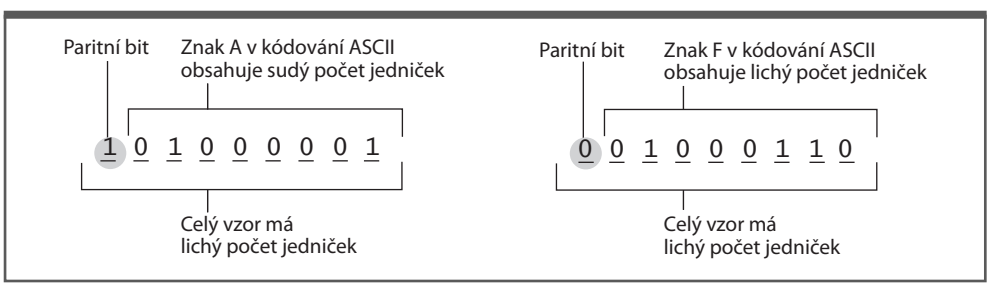
Právě popsaný paritní systém se nazývá **lichá parita**, protože jsme jej navrhli tak, aby každý správný vzor zahrnoval lichý počet jedniček. Jiná metoda se označuje jako **sudá parita**. V systému sudé parity je každý vzor sestaven tak, aby obsahoval sudý počet jedniček. Chyby lze tedy zjistit na základě výskytu vzoru s lichým počtem jedniček.

V současnosti se paritní bity často používají v operační paměti počítače. Obvykle si sice představujeme, že počítače mají paměťové buňky s kapacitou 8 bitů, ale ve skutečnosti mají kapacitu 9 bitů, z nichž jeden bit slouží jako paritní. Pokaždé, když paměťový obvod přijme osmibitový vzor, doplní k němu paritní bit a uloží výslednou posloupnost s devíti bity. Při pozdějším čtení dat paměťový obvod kontroluje paritu devítibitového vzoru. Jestliže výsledek neukazuje na chybu, paměť odebere

paritní bit a vrátí důvěryhodnou sekvenci zbývajících 8 bitů. Jinak paměť předá 8 datových bitů s upozorněním, že vrácený vzor se nemusí shodovat se vzorem, který byl do paměti původně umístěn.

Toto přímé použití paritních bitů je jednoduché, ale má svá omezení. V případě, že původní vzor měl lichý počet jedniček a dojde ke dvěma chybám, bude mít i nadále lichý počet jedniček a paritní systém proto chyby nezjistí. Lze odvodit, že při jednoduchém nasazení paritních bitů není možné detekovat žádný sudý počet chyb v příslušném bitovém vzoru.

Uvedený problém je možné minimalizovat způsobem, který se někdy aplikuje na dlouhé bitové vzory, jako jsou řetězce bitů zapsané v sektoru magnetického disku. V tomto případě je vzor doprovázen kolekcí paritních bitů, které dohromady tvoří **kontrolní bajt** (checkbyte). Kontrolní bajt obsahuje výhradně paritní bity a každý z těchto bitů souvisí s určitou posloupností bitů rozloženou v rámci celého vzoru. Jeden paritní bit může být například přidružen ke každému osmému bitu ve vzoru počínaje prvním bitem, zatímco jiný bit může patřit každému osmému bitu počínaje druhým bitem. Tímto způsobem lze zvýšit pravděpodobnost, že bude odhalena řada chyb koncentrovaných v jedné oblasti původního vzoru, protože se projeví u několika paritních bitů. Úpravami této metody kontrolního bajtu vznikla schémata detekce chyb, která se označují jako **kontrolní součty** (checksum) a **CRC** (cyclic redundancy check).



Obrázek 1.28: Kódy ASCII pro písmena A a F s úpravami na lichou paritu

Samoopravné kódy

Paritní bity sice dovolují zjistit chybu, ale neposkytují informace, které by ji umožnily opravit. Mnohé překvapí, že lze vyvinout **samoopravné kódy** (error-correcting code), které dokáží chyby nejenom detekovat, ale také opravovat. Intuice nám přece napovídá, že nemůžeme opravit chyby v přijaté zprávě, pokud ještě informaci ve zprávě neznáme. Na obrázku 1.29 je však znázorněn jednoduchý kód s takovými vlastnostmi.

Abychom porozuměli fungování tohoto kódu, musíme nejdříve definovat **Hammingovu vzdálenost**. Odkazuje na R. W. Hamminga, který ve 40. letech začal hledat kódy na opravu chyb, protože jej trápila nespolehlivost prototypů komunikačních zařízení. Hammingova vzdálenost mezi dvěma bitovým vzory je dána počtem bitů, v nichž se vzory liší. Příklad: Hammingova vzdálenost mezi vzory, které v kódu

na obrázku 1.29 představují písmena A a B, je rovna čtyřem, a Hammingova vzdálenost mezi kódy pro písmena B a C se rovná třem. Kód na obrázku 1.29 se vyznačuje důležitou vlastností: libovolné dva kódy dělí Hammingova vzdálenost tři nebo více.

Pokud se ve vzoru na obrázku 1.29 změní jediný bit, lze chybu detekovat, protože výsledek neodpovídá žádnému platnému vzoru. (Ve vzoru se musí změnit alespoň 3 bity, aby vypadal jako jiný platný vzor.) Navíc můžeme také zjistit, jaký byl původní vzor. Modifikovaný vzor bude mít totiž od své originální formy pouze jednotkovou Hammingovu vzdálenost, ale od libovolného jiného platného vzoru bude tato vzdálenost minimálně dvojnásobná.

Chceme-li tedy dekódovat zprávu, která byla původně zakódována systémem podle obrázku 1.29, jednoduše porovnáme všechny přijaté vzory se vzory v kódu, dokud nenalezneme kód, který se od přijaté posloupnosti liší pouze jedním bitem. Usoudíme, že se jedná o symbol, který chceme dekódovat. Jestliže bychom například přijali bitový vzor 010100 a porovnali jej se schématy v kódu, získali bychom tabulku na obrázku 1.30. Došli bychom tedy k závěru, že musel být přenesen znak D, protože je zjištěnému vzoru nejblíže.

Symbol	Kód
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

Obrázek 1.29: Kód na opravu chyb

Znak	Kód	Přijatý vzor	Vzdálenost mezi přijatým vzorem a kódem
A	0 0 0 0 0 0	0 **1** 0 **1** 0 0	2
B	0 0 1 1 1 1	0 **1 0** 1 **0 0**	4
C	0 1 0 0 1 1	0 1 0 **1 0 0**	3
D	0 1 1 1 0 0	0 1 **0** 1 0 0	**1** — Nejmenší vzdálenost
E	1 0 0 1 1 0	**0 1** 0 1 **0** 0	3
F	1 0 1 0 0 1	**0 1 0 1** 0 **0**	5
G	1 1 0 1 0 1	**0** 1 0 1 0 **0**	2
H	1 1 1 0 1 0	**0** 1 **0 1 0** 0	4

Obrázek 1.30: Dekódování vzoru 010100 pomocí kódu z obrázku 1.29

Můžeme si povšimnout, že uvedená metoda založená na kódu z obrázku 1.29 umožňuje detekovat až dvě chyby v jednom vzoru a opravit jednu chybu. Pokud bychom kód navrhli tak, aby každý vzor ležel od každého jiného vzoru v Hammingově vzdálenosti alespoň 5, dokázali bychom detekovat až čtyři chyby v jednom vzoru a opravit dvě. Návrh efektivních kódů s velkými Hammingovými vzdálenostmi samozřejmě není jednoduchý. Ve skutečnosti tento úkol patří do odvětví matematiky, které se nazývá teorie algebraického kódování. Toto odvětví přitom vychází z lineární algebry a teorie matic.

Díky intenzivnímu používání metod na opravu chyb lze zvýšit spolehlivost počítačového hardwaru. Tyto metody se například často uplatňují v magnetických pevných discích s vysokou kapacitou, kde snižují pravděpodobnost poškození dat kvůli vadám magnetického povrchu. Co se týče disků CD, hlavní rozdíl mezi původním formátem určeným pro zvukové disky a novějším formátem, který se používá k ukládání počítačových dat, spočívá právě v úrovni chybové korekce. Formát CD-DA zahrnuje funkce na opravu chyb, které redukují frekvenci chyb na jednu chybu na dva disky CD. To sice postačuje pro záznam zvuku, ale společnost, která dodává svým zákazníkům software na discích CD, by 50 % vadných disků nemohla tolerovat. U disků CD, které jsou určeny k ukládání dat, se proto používají další metody na opravu chyb, které pravděpodobnost chyby omezují na pouhou jednu chybu na 20 000 disků.

Otázky a cvičení

1. Následující bajty byly původně zakódovány pomocí liché parity. U kterých z nich víte, že došlo k chybě?

   a. `100101101` **b.** `100000001` **c.** `000000000`
   d. `111000000` **e.** `011111111`

2. Mohly by se v bajtech z otázky 1 vyskytnout chyby, které byste nedokázali zjistit? Vysvětlete svou odpověď.

3. Jak by se změnily vaše odpovědi na otázky 1 a 2, pokud byste se dozvěděli, že byla místo liché parity použita sudá?

4. Zakódujte tyto věty v kódování ASCII pomocí liché parity tak, že ke každému kódu znaku přidáte paritní bit na stranu s nejvyšším řádem:

   a. „Stop!“ zavolal Petr. **b.** Je 2 + 3 = 5?

5. Pomocí kódu na opravu chyb z obrázku 1.29 dekódujte následující zprávy:

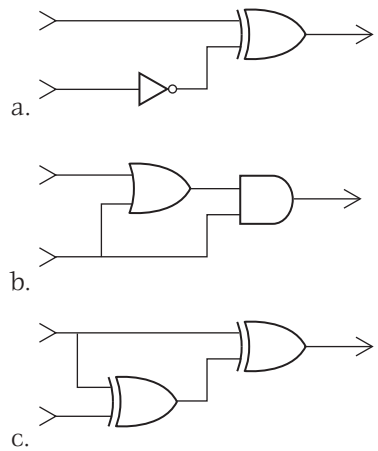
   a. `001111 100100 001100` **b.** `010001 000000 001011`
   c. `011010 110110 100000 011100`

6. Vytvořte kódy pro znaky A, B, C a D, které budou mít délku pěti bitů a Hammingova vzdálenost mezi libovolnými dvěma z nich bude větší nebo rovna třem.

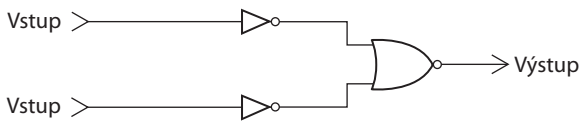
Úlohy na procvičování témat kapitoly

(Problémy označené hvězdičkou patří k volitelným částem kapitoly.)

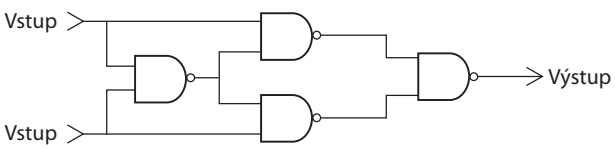
1. Určete výstup každého z následujících obvodů. Předpokládejte přitom, že horní vstup má hodnotu 1 a dolní vstup 0. Jaký bude výstup, pokud má horní vstup hodnotu 0 a dolní 1?



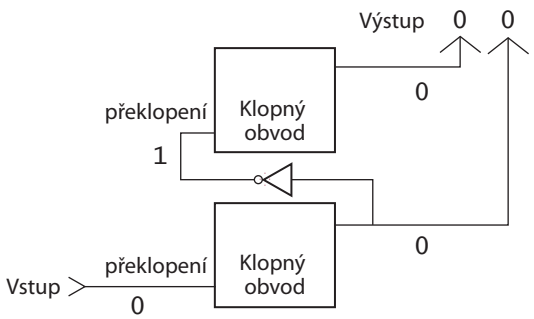
2. **a.** Jakou logickou operaci tento obvod vyhodnocuje?



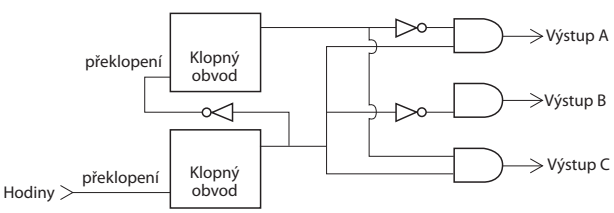
b. Jakou logickou operaci tento obvod vyhodnocuje?



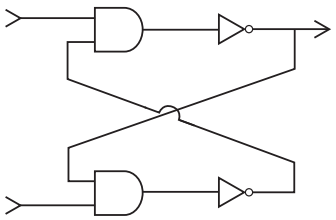
***3.** **a.** Zakoupíte-li v obchodě s elektronickými součástkami klopný obvod, můžete zjistit, že obsahuje dodatečný vstup zvaný *překlopení* (flip). Když tento vstup změní hodnotu z 0 na 1, obrátí se stav na výstupu (pokud měl hodnotu 0, bude nyní vykazovat hodnotu 1 a naopak). Jestliže se však vstup překlopení změní z 1 na 0, nestane se nic. Ačkoli zatím nemusíte vědět, jak je toto chování v elektronickém obvodu implementováno, můžete toto zařízení přesto použít jako abstraktní nástroj v jiných obvodech. Uvažujte zapojení, které obsahuje dva následující klopné obvody. Pokud na vstup obvodu vyšlete pulz, změní se stav dolního klopného obvodu. Stav druhého klopného obvodu však zůstane stejný, protože jeho vstup (přijatý z výstupu hradla NOT) změnil hodnotu z 1 na 0. Tento obvod proto nyní poskytuje výstupy 0 a 1. Druhý pulz by překlopil stav obou klopných obvodů a výsledkem by byly výstupní hodnoty 1 a 0. Jaký by byl výstup po třetím pulzu? A po čtvrtém?



b. Činnost různých komponent počítače je často potřeba koordinovat. To lze zajistit připojením pulzujícího signálu (který se označuje jako *hodiny* – clock) k obvodu, který vypadá podobně jako v části a. Další hradla (viz obrázek) poté odesílají sladěné signály jiným připojeným obvodům. Když tento obvod prozkoumáte, měli byste ověřit, že při 1., 5., 9. ... pulzu hodin bude na výstup A odeslána hodnota 1. Při kterých pulzech hodin bude jednička odeslána na výstup B? Při kterých pulzech hodin bude jednička odeslána na výstup C? Na kterém výstupu se jednička objeví při 4. pulzu hodin?



4. Předpokládejte, že oba vstupy následujícího obvodu mají hodnotu 1. Popište, co se stane, když se horní vstup dočasně změní na hodnotu 0. Popište, co se stane, když se dolní vstup dočasně změní na hodnotu 0. Překreslete obvod pomocí hradel NAND.



5. Následující tabulka obsahuje adresy několika buněk v operační paměti počítače a jejich obsah (v šestnáctkové notaci). Na základě této paměťové konfigurace postupujte podle pokynů a zaznamenejte výsledný obsah všech paměťových buněk:

Adresa	Obsah
0	AB
1	53
2	D6
3	2

Krok 1: Přesuňte obsah buňky s adresou 03 do buňky s adresou 00.
Krok 2: Přesuňte hodnotu 01 do buňky s adresou 02.
Krok 3: Přesuňte hodnotu uloženou na adrese 01 do buňky s adresou 03.

6. Kolik buněk se vejde do operační paměti počítače, jestliže lze adresu každé buňky vyjádřit dvěma šestnáctkovými číslicemi? Jak tomu bude při použití čtyř šestnáctkových číslic?

7. Jaké sekvence bitů zastupují tyto šestnáctkové výrazy?

   a. CD **b.** 67 **c.** 9A

   d. FF **e.** 10

8. Jakou hodnotu má nejvýznamnější bit v sekvencích bitů, které jsou vyjádřeny těmito šestnáctkovými výrazy?

   a. 8F **b.** FF

   c. 6F **d.** 1F

9. Vyjádřete následující bitové vzory v šestnáctkové notaci:

   a. 101000001010
   b. 110001111011
   c. 000010111110

10. Předpokládejte, že paměť digitálního fotoaparátu má kapacitu 256 MB. Kolik fotografií lze ve fotoaparátu uložit, pokud je každý snímek tvořen na šířku i na výšku 1024 pixely a uložení každého pixelu vyžaduje tři bajty?

11. Předpokládejte, že obrázek je na obrazovce reprezentován obdélníkovým polem, které obsahuje 1 024 sloupců a 768 řádek pixelů. Jestliže každý pixel vyžaduje 8 bitů k zakódování barvy a dalších 8 bitů k zakódování intenzity, kolik paměťových buněk velikosti jednoho bajtu je potřeba k uchování celého obrázku?

12. **a.** Uveďte dvě výhody, které poskytuje operační paměť oproti úložišti na magnetickém disku.

    b. Uveďte dvě výhody, které poskytuje úložiště magnetického disku oproti operační paměti.

13. Předpokládejte, že ze 120 GB kapacity pevného disku osobního počítače je volných pouze 50 GB. Je rozumné k zálohování veškerých dat na pevném disku zvolit disky CD? Jaká bude odpověď v případě disků DVD?

14. Pokud každý sektor na magnetickém disku obsahuje 1 024 bajtů, kolik sektorů bude potřeba k uložení jediné stránky textu (řekněme 50 řádků po 100 znacích), je-li každý znak reprezentován v kódování Unicode?

15. Kolik bajtů úložného místa bude potřeba k uložení románu o 400 stranách, kdy každá stránka zahrnuje 3 500 znaků v kódování ASCII? Kolik bajtů by spotřebovalo uložení stejného textu ve formátu Unicode?

16. Jaká je doba latence běžného pevného disku, který se otáčí rychlostí 360 otáček za sekundu?

17. Jaká je průměrná přístupová doba pevného disku, který se otáčí rychlostí 360 otáček za sekundu a má dobu vyhledání 10 milisekund?

18. Předpokládejte, že písař dokáže den co den trvale psát 60 slov za minutu. Za jak dlouho by dokázal zaplnit disk CD s kapacitou 640 MB? Při výpočtu uvažujte, že slova mají průměrnou délku pět znaků a každý znak vyžaduje jeden bajt úložiště.

19. Následuje zpráva v kódování ASCII. Jaký je její obsah?

```
01010111 01101000 01100001 01110100
00100000 01100100 01101111 01100101
01110011 00100000 01101001 01110100
00100000 01110011 01100001 01111001
00111111
```

20. Následuje zpráva v kódování ASCII, která využívá jeden bajt na jeden znak a následně je vyjádřena v šestnáctkové soustavě. Jaký je obsah zprávy?

```
686578616465636696D616C
```

21. Zakódujte následující věty v kódování ASCII s jedním bajtem na jeden znak.

a. Je 100 / 5 = 20?

b. Cena je 7,25 $.

22. Vyjádřete své odpovědi na předchozí problém v šestnáctkové soustavě.

23. Uveďte binární reprezentace celých čísel od 8 do 18.

24. **a.** Napište číslo 23 tak, že vyjádříte číslice 2 a 3 v kódování ASCII.

b. Napište číslo 23 ve dvojkové soustavě.

25. Jaké hodnoty mají binární reprezentace, z nichž pouze jeden bit se rovná 1? Uveďte binární reprezentace nejmenších šesti hodnot s touto vlastností.

***26.** Převeďte všechny následující čísla ve dvojkové soustavě na odpovídající čísla v desítkové soustavě:

a. 1111 **b.** 0001 **c.** 10101
d. 1000 **e.** 10011 **f.** 000000
g. 1001 **h.** 10001 **i.** 100001
j. 11001 **k.** 11010 **l.** 11011

***27.** Převeďte každou z následujících reprezentací čísel v desítkové soustavě na reprezentaci téhož čísla ve dvojkové soustavě:

a. 7 **b.** 11 **c.** 16
d. 17 **e.** 31

***28.** Převeďte všechny následující reprezentace v zápisu s posunem 16 na odpovídající čísla v desítkové soustavě:

a. 10001 **b.** 10101 **c.** 01101
d. 01111 **e.** 11111

***29.** Převeďte každou z následujících reprezentací čísel v desítkové soustavě na odpovídající zápis s posunem čtyři:

a. 0 **b.** 3 **c.** –2
d. –1 **e.** 2

***30.** Převeďte každou z následujících reprezentací v systému dvojkového doplňku na odpovídající reprezentaci v desítkové soustavě:

a. 01111 **b.** 10100 **c.** 01100
d. 10000 **e.** 10110

***31.** Převeďte každou z následujících reprezentací čísel v desítkové soustavě do dvojkového doplňku, kde je každá hodnota vyjádřena sedmi bity:

a. 13 **b.** –13 **c.** –1
d. 0 **e.** 16

***32.** Proveďte všechna následující sčítání za předpokladu, že bitové řetězce představují hodnoty zapsané do dvojkového doplňku. Označte všechny případy, kde výsledek není správný kvůli přetečení.

a. 00101 + 01000 **b.** 11111 + 00001 **c.** 01111 + 00001
d. 10111 + 11010 **e.** 11111 + 11111 **f.** 00111 + 01100

***33.** Vyřešte všechny následující úlohy tak, že převedete hodnoty do dvojkového doplňku (se vzory o délce 5 bitů), převedete případné odečítání na odpovídající operace sčítání a nakonec toto sčítání provedete. Zkontrolujte své odpovědi tak, že je zpětně převedete do desítkové soustavy. (Kontrolujte případné přetečení.)

a. 5 + 1 **b.** 5 − 1 **c.** 12 − 5
d. 8 − 7 **e.** 12 + 5 **f.** 5 − 11

***34.** Převeďte všechny následující binární reprezentace na odpovídající čísla v desítkové soustavě:

a. 11,11 **b.** 100,0101 **c.** 0,1101
d. 1,0 **e.** 10,01

***35.** Vyjádřete všechny následující hodnoty ve dvojkové soustavě:

a. $5\frac{3}{4}$ **b.** $15\frac{15}{16}$ **c.** $5\frac{3}{8}$
d. $1\frac{1}{4}$ **e.** $6\frac{5}{8}$

***36.** Dekódujte následující bitové posloupnosti ve formátu s plovoucí řádovou čárkou (popis na obrázku 1.26):

a. 01011001 **b.** 11001000
c. 10101100 **d.** 00111001

***37.** Zakódujte následující hodnoty pomocí osmibitového formátu s plovoucí řádovou čárkou, který je popsán na obrázku 1.26. Označte všechny případy, v nichž dochází k chybě při odříznutí.

a. $-7\frac{1}{2}$ **b.** $\frac{1}{2}$ **c.** $-3\frac{3}{4}$
d. $\frac{7}{32}$ **e.** $\frac{31}{32}$

***38.** Za předpokladu, že se nemusíte omezovat na normalizovaný tvar, uveďte všechny bitové vzory, které mohou reprezentovat hodnotu $\frac{3}{8}$ ve formátu s plovoucí řádovou čárkou dle obrázku 1.26.

***39.** Jaké nejlepší přiblížení odmocniny ze dvou lze vyjádřit v osmibitovém formátu s plovoucí řádovou čárkou, který je popsán na obrázku 1.26? Kterou hodnotu ve skutečnosti dostanete, když počítač toto přibližné číslo umocní při použití tohoto formátu s plovoucí řádovou čárkou?

***40.** Jaké nejlepší přiblížení hodnoty jedné desetiny lze reprezentovat v osmibitovém formátu s plovoucí řádovou čárkou, který je popsán na obrázku 1.26?

***41.** Vysvětlete, jak může dojít k chybám, když jsou měření v metrickém systému zapsána pomocí notace s plovoucí řádovou čárkou. Co se například stane, jestliže zaznamenáte 110 cm v jednotkách metrů?

***42.** Jeden z bitových vzorů 01011 a 11011 představuje hodnotu uloženou v zápisu s posunem 16 a druhý zastupuje stejnou hodnotu, avšak vyjádřenou ve dvojkovém doplňku.

a. Co lze říci o této společné hodnotě?

b. Jaký vztah má vzor, který představuje hodnotu uloženou ve dvojkovém doplňku, a vzor, který zastupuje stejnou hodnotu, avšak vyjádřenou v zápisu s posunem 16, když oba systémy pracují se stejnou bitovou délkou?

***43.** Tři bitové vzory 10000010, 01101000 a 00000010 znamenají stejnou hodnotu ve dvojkovém doplňku, v zápisu s posunem a osmibitovém formátu s plovoucí řádovou čárkou (viz obrázek 1.26), ovšem nikoli nutně v uvedeném pořadí. Jaká je společná hodnota a který vzor patří ke které notaci?

***44.** Které z následujících hodnot není možné přesně vyjádřit ve formátu s plovoucí řádovou čárkou, který jsme představili na obrázku 1.26?

a. $6\frac{1}{2}$ **b.** $\frac{13}{16}$ **c.** 9
d. $\frac{17}{32}$ **e.** $\frac{15}{16}$

***45.** Pokud se délka bitových řetězců, které slouží k binární reprezentaci celých čísel, změní ze čtyř na šest bitů, jakou změnu to znamená v hodnotě největšího celého čísla, které lze vyjádřit? Jak tomu bude při použití dvojkového doplňku?

***46.** Jaká bude šestnáctková reprezentace největší paměťové adresy v paměti o velikosti 4 MB, jestliže má každá paměťová buňka kapacitu jednoho bajtu?

***47.** Jak bude vypadat zakódovaná verze zprávy

```
xxy yyx xxy xxy yyx
```

při použití komprese LZW s počátečním slovníkem, který bude obsahovat znaky *x*, *y* a mezeru (popis metody naleznete v části 1.8)?

***48.** Následující zpráva byla komprimována algoritmem LZW pomocí slovníku, jehož první, druhou a třetí položkou jsou *x*, *y* a mezera. Jak vypadá dekomprimovaná zpráva?

```
22123113431213536
```

***49.** Pokud zpráva

```
xxy yyx xxy xxyy
```

bude komprimována metodou LZW s počátečním slovníkem, který jako první, druhou a třetí položku obsahuje *x*, *y* a mezeru, jak budou vypadat záznamy ve výsledném slovníku?

***50.** Jak se dozvíte v následující kapitole, bity lze pomocí klasické telefonní sítě vysílat tak, že se bitové vzory převedou na zvuk, tento zvuk je přenesen po telefonní lince a následně konvertován zpět na původní posloupnost bitů. Pomocí těchto metod lze dosáhnout maxi-

mální přenosové rychlosti 57,6 kb/s. Stačí to k videokonferencím, jestliže je video komprimováno systémem MPEG?

*51. Zakódujte následující věty v kódování ASCII pomocí sudé parity tak, že ke každému kódu znaku přidáte paritní bit na stranu s nejvyšším řádem:

a. Je 100/5 = 20?

b. Cena je 7,25 $.

*52. Následující zpráva byla původně vyslána s lichou paritou v každém dílčím bitovém řetězci. Ve kterých řetězcích určitě došlo k chybám?

11001 11011 10110 00000 11111 10001
10101 00100 01110

*53. Představte si 24bitový kód, který je generován zřetězením tří kopií reprezentace příslušného znaku v kódování ASCII (například písmeno A bude reprezentováno bitovým řetězcem 010000010100000101000001). Jaké samoopravné vlastnosti tento nový kód poskytuje?

*54. Pomocí samoopravného kódu, který je znázorněn na obrázku 1.30, dekódujte následující slova:

a. 111010 110110

b. 101000 100110 001100

c. 011101 000110 000000 010100

d. 010010 001000 001110 101111 000000 110111 100110

e. 010011 000000 101001 100110

Společenské otázky

Následující otázky mají čtenáře provést etickými, společenskými a právními aspekty oboru počítačů. Cílem není jen odpovědět na jednotlivé otázky. Měli byste se také zamyslet na tím, proč jste vybrali konkrétní odpověď a zda dokážete odpovědi na jednotlivé otázky konzistentně zdůvodnit.

1. V kritickém systému se vyskytla chyba při odseknutí, která způsobila rozsáhlé škody a ztráty na životech. Kdo je za to odpovědný (pokud vůbec někdo)? Návrhář hardwaru? Architekt softwaru? Programátor, který příslušnou část programu napsal? Osoba, která rozhodla o použití softwaru ke konkrétnímu účelu? Jak by tomu bylo v případě, že by původní dodavatel softwaru chybu opravil, ale uživatel by tuto aktualizaci kritické aplikace nezakoupil a nenainstaloval? Co v situaci, kdy byl software pořízen nelegálně?
2. Je přípustné, aby uživatel počítače při vývoji vlastních aplikací ignoroval možnost chyb při odseknutí a jejich důsledků?
3. Bylo v 70. letech 20. století etické vyvíjet software, který reprezentoval rok pomocí pouhých posledních dvou číslic (například 76 pro rok 1976), a ignorovat přitom fakt, že takový program bude na přelomu století fungovat chybně? Je v současnosti etické kódovat rok pomocí pouhých tří číslic (jako např. 982 pro 1982 a 015 místo 2015)? Jak je tomu při použití čtyř číslic?
4. Mnoho lidí tvrdí, že kódováním se informace často ředí nebo jinak zkreslují, protože je přitom nutné informace kvantifikovat. Dotazníky, kde je při vyplňování názorů nutné odpovídat na stupnici od jedné do pěti, jsou podle jejich názoru ze své podstaty vadné. Do jaké míry je možné informace vyjádřit číselně? Lze kvantifikovat výhody a nevýhody různých umístění spalovny? Dá se kvantifikovat diskuze o využívání jaderné energie a ukládání jaderného odpadu? Je nebezpečné přijímat rozhodnutí na základě průměrů a jiných statistických analýz? Je etické, když zpravodajské agentury oznamují výsledky průzkumů, aniž by ke zprávě připojily přesné znění otázek? Je možné vyčíslit hodnotu

lidského života? Je přípustné, aby společnost přestala investovat do zlepšování produktu, i když by dodatečné investice mohly snížit pravděpodobnost smrtelných nehod při jeho používání?

5. Mělo by se v případě práv na shromažďování a šíření dat přihlížet k typu dat? Měla by tedy práva týkající se shromažďování a publikování fotografií, zvuků nebo videí odpovídat právům na shromažďování a šíření textu?
6. Ať už úmyslně či neúmyslně, příspěvky novinářů často odrážejí jejich zaujatost. Změnou pouhých několika slov lze mnohdy zcela změnit vyznění článku. (Porovnejte větu: „Většina dotázaných s referendem nesouhlasí." s větou „Významná část dotázaných referendum podporuje.") Je rozdíl mezi přizpůsobením textu (vynecháním určitých aspektů nebo pečlivým výběrem slov) a úpravou fotografie?
7. Předpokládejte, že kvůli použití systému komprese dat je ztracena malá, ale podstatná část informace. Jaké otázky ohledně odpovědnosti by to mohlo vyvolat? Jak by na ně bylo možné odpovědět?

Další zdroje informací

Drew, M. a Z. Li. *Fundamentals of Multimedia* (Základy multimédií). Upper Saddle River, NJ: Prentice-Hall, 2004.

Halsall, F. *Multimedia Communications* (Multimediální komunikace). Boston, MA: Addison-Wesley, 2001.

Hamacher, V. C., Z. G. Vranesic a S. G. Zaky. *Computer Organization* (Organizace počítačů), 5. vyd. New York: McGraw-Hill, 2002.

Knuth, D. E. *The Art of Computer Programming*. sv. 2, 3. vyd. Boston, MA: Addison-Wesley, 1998. (Česky *Umění programování,* sv. 2, Brno: Computer Press 2010)

Long, B. *Complete Digital Photography* (Úplný přehled digitální fotografie), 3. vyd. Hingham, MA: Charles River Media, 2005.

Miano, J. *Compressed Image File Formats* (Komprimované souborové formáty obrázků) New York: ACM Press, 1999.

Petzold, C. *CODE: The Hidden Language of Computer Hardware and Software* (Kód: skrytý jazyk počítačového hardwaru a softwaru). Redman, WA: Microsoft Press, 2000.

Salomon, D. *Data Compression: The Complete Reference* (Komprese dat: kompletní referenční příručka), 4. vyd. New York: Springer, 2007.

Sayood, K. *Introduction to Data Compression* (Úvod do komprese dat), 3. vyd. San Francisco: Morgan Kaufmann, 2005.

KAPITOLA

2

Zpracování dat

V této kapitole se naučíme, jak počítač manipuluje s daty a jak komunikuje s periferními zařízeními, jako jsou tiskárny a klávesnice. Přitom prozkoumáme základy architektury počítačů a dozvíme se, jak se počítače programují pomocí zakódovaných instrukcí, které se označují jako instrukce strojového jazyka.

****Hvězdičky označují doporučené volitelné části.***

V kapitole 1 jsme se zabývali tématy, která souvisela s ukládáním dat v počítači. V této kapitole zjistíme, jak počítač data zpracovává. Přitom musí počítač data přesunovat mezi různými umístěními a provádět operace typu aritmetických výpočtů, úprav textu a manipulace s obrázky. Nejdříve se seznámíme s architekturou počítačů nad rámec systémů ukládání dat.

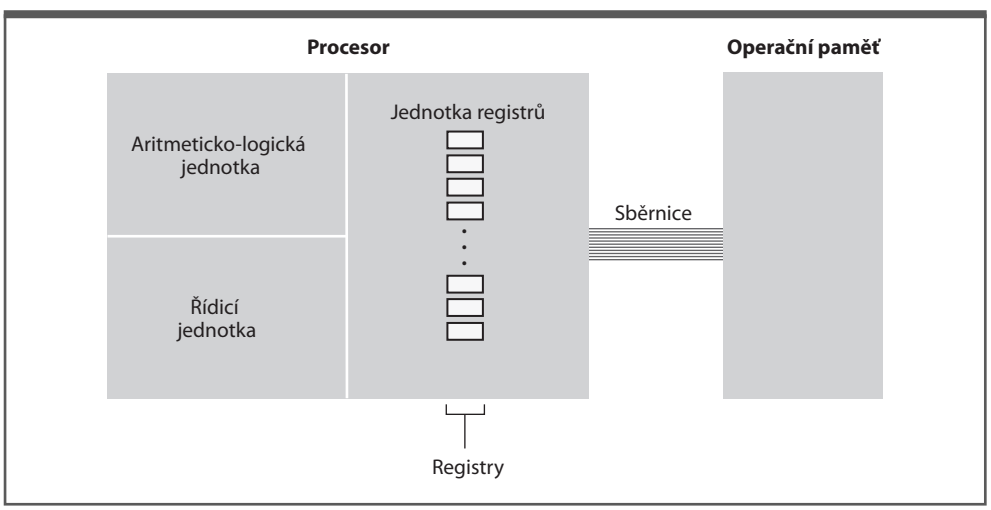
2.1: Architektura počítačů

Elektronický prvek v počítači, který řídí manipulaci s daty, se nazývá **procesor** nebo **CPU** (z anglického termínu „central processing unit"). V polovině 20. století se jednalo o velké jednotky, které se mohly skládat až z několika regálů elektronických obvodů, takže je nebylo možné přehlédnout. Díky technologickému pokroku se však tato zařízení výrazně zmenšila. Procesory, které jsou součástí dnešních stolních počítačů a notebooků, mají podobu malých plochých čtverečků (velikosti přibližně pět krát pět centimetrů). Jejich konektory se zasunují do patice na **základní desce** počítače (motherboard). Procesory ve smartphonech, miniaturních noteboocích a dalších **mobilních internetových zařízeních** (MID – Mobile Internet Device) jsou velké přibližně jako polovina poštovní známky. Vzhledem ke svým malým rozměrům se tyto procesory označují jako **mikroprocesory**.

Úvod do procesorů

Procesor sestává ze třech částí (viz obrázek 2.1): **aritmeticko-logické jednotky** s obvody, které mohou provádět různé operace s daty (např. sčítání a odčítání), **řídicí jednotky** s obvody pro koordinaci činností počítače a **jednotky registrů**, která obsahuje buňky k ukládání dat (podobné buňkám operační paměti). Tyto buňky označované jako **registry** slouží k dočasnému uchovávání informací uvnitř procesoru.

Některé registry z jednotky registrů patří mezi **obecné** (general-purpose register), zatímco jiné jsou **speciální** (special-purpose register). Některými speciálními registry se budeme zabývat v části 2.3. Prozatím nás budou zajímat jen obecné registry.



Obrázek 2.1: Propojení procesoru a hlavní paměti pomocí sběrnice

Obecné registry fungují jako dočasné úložiště dat, která procesor zpracovává. Tyto registry obsahují vstupy obvodů aritmeticko-logické jednotky a poskytují místo k uchování výsledků této jednotky. Pokud má procesor provést operaci s daty uloženými v operační paměti, přenese řídicí jednotka příslušná data z paměti do obecných registrů, informuje aritmeticko-logickou jednotku o tom, ve kterých registrech se data nacházejí, aktivuje příslušné obvody aritmeticko-logické jednotky a sdělí jí, do kterého registru má umístit výsledek.

Kvůli přenosu bitových posloupností je procesor počítače propojen s jeho hlavní pamětí sadou vodičů, které se označují jako **sběrnice** (angl. „bus" – viz opět obrázek 2.1). Pomocí této sběrnice procesor extrahuje (načítá) data z operační paměti. Přitom uvede adresu příslušné paměťové buňky a doplní ji elektronickým signálem paměťovému obvodu, aby načetl data uložená v dané buňce. Podobným způsobem procesor také umisťuje (zapisuje) data do paměti. V tomto případě poskytne adresu cílové buňky a data, která do ní budou uložena. Příslušným elektronickým signálem přitom hlavní paměti oznámí, že má přijatá data uložit.

Z uvedeného návrhu vyplývá, že k sečtení dvou hodnot uložených v hlavní paměti nestačí provést pouze operaci sčítání. Data je nutné přenést z hlavní paměti do registrů v procesoru, hodnoty je potřeba sečíst a výsledek umístit do jiného registru a následně tento výsledek uložit do paměťové buňky. Celý proces shrnuje pět kroků, které jsou uvedeny na obrázku 2.2.

Koncepce uloženého programu

První počítače zrovna nevynikaly pružností. Kroky, které prováděl konkrétní stroj, byly při jeho návrhu zabudovány do řídicí jednotky. Kvůli vyšší flexibilitě byly některé z raných elektronických počítačů navrženy tak, aby bylo možné procesor jednoduše přepojovat. Tuto variabilitu zapojení poskytovaly přepínače zasunované do otvorů, které připomínaly dírkovanou desku starých telefonních ústředen.

Krok 1: Načti jednu sčítanou hodnotu z paměti a ulož ji do registru.

Krok 2: Načti druhou sčítanou hodnotu z paměti a ulož ji do jiného registru.

Krok 3: Aktivuj obvody pro sčítání. Jako jejich vstupy nastav registry z kroků 1 a 2 spolu s dalším registrem, který bude uchovávat výsledek.

Krok 4: Ulož výsledek do paměti.

Krok 5: Stop.

Obrázek 2.2: Sčítání hodnot uložených v paměti

Zásadní myšlenkový přelom (který se zřejmě omylem připisuje Johnu von Neumannovi) spočíval ve zjištění, že program lze stejně jako data zakódovat a uložit do hlavní paměti. Pokud řídicí jednotka dokáže extrahovat program z paměti, dekódovat jeho instrukce a spustit je, lze program vykonávaný počítačem změnit pouhou změnou obsahu počítačové paměti a není nutné měnit zapojení procesoru.

Princip uložení počítačového programu do operační paměti počítače se označuje jako **koncepce uloženého programu** (stored-program concept). Tento standardní přístup se prosadil natolik, že nám v současnosti připadá jako samozřejmý. Původně

Mezipaměť

Prvky počítačové paměti je vhodné analyzovat s ohledem na jejich funkci. Registry uchovávají data, která jsou nezbytná pro aktuální operaci. Operační paměť se používá k uložení dat, která budou potřebná v blízké budoucnosti, a hromadné úložiště obsahuje data, která bude počítač pravděpodobně zpracovávat teprve později. Návrh mnoha počítačů zahrnuje další paměťovou úroveň, která se označuje jako mezipaměť (cache memory). **Mezipaměť** je malá vysokorychlostní paměť (s kapacitou kolem několika set KB) umístěná přímo v procesoru. V této speciální paměťové oblasti se počítač pokouší udržovat kopii části hlavní paměti, která je momentálně nejaktuálnější. V této konfiguraci lze některá data, která by se normálně musela přenášet mezi registry a hlavní pamětí, přesunovat mezi registry a mezipamětí. Všechny změny mezipaměti se poté přenášejí do hlavní paměti zároveň v čase, který je k tomu nejvhodnější. Procesor tohoto typu může realizovat své strojové cykly rychleji, protože jej nezdržuje komunikace s operační pamětí.

tato změna vypadala obtížně, protože programy a data se obecně považovaly za odlišné prvky: data se ukládala do paměti a programy byly součástí procesoru. Na tuto situaci se dokonale hodí přirovnání o stromech, pro které není vidět les. Je snadné upadnout do takových myšlenkových pastí, a nebýt uvedené změny paradigmatu, mohl se vývoj informatiky značně zpomalit. Počítačová věda je vzrušující také proto, že nové myšlenky neustále otevírají dveře novým teoriím a aplikacím.

Otázky a cvičení

1. Jaké pořadí kroků je podle vašeho názoru nutné k přesunu obsahu jedné paměťové buňky počítače do jiné paměťové buňky?
2. Které informace musí procesor poskytnout obvodům operační paměti, aby bylo možné zapsat hodnotu do paměťové buňky?
3. Hromadné úložiště, operační paměť i obecné registry patří mezi úložné systémy. Jak se liší svým použitím?

2.2: Strojový jazyk

S ohledem na koncepci uloženého programu jsou procesory navrženy tak, aby rozpoznávaly instrukce zakódované pomocí bitových vzorů. Soubor takových instrukcí spolu se systémem kódování se označuje jako **strojový jazyk** nebo **strojový kód** (machine language). Instrukce vyjádřená v tomto jazyce se nazývá instrukce na strojové úrovni nebo běžněji **strojová instrukce** (machine instruction).

Sada instrukcí

Seznam strojových instrukcí, které musí běžný procesor dekódovat a provádět, je poměrně krátký. Jakmile počítač dokáže realizovat několik vhodně vybraných základních úkolů, přidávání dalších funkcí již ve skutečnosti nezvyšuje teoretické možnosti počítače. Jinými slovy: za určitou mezí sice mohou dodatečné funkce například usnadnit používání, ale ničím nepřispívají k rozsahu základních schopností počítače.

Dva základní přístupy k návrhu procesorů se liší právě tím, nakolik by měl návrh počítače vycházet z výše uvedeného faktu. Jedna koncepce je založena na tom, že procesor by měl pracovat s minimální sadou strojových instrukcí. Tento přístup vede k počítačům typu **RISC** (reduced instruction set computer – počítač s omezenou instrukční sadou). Ve prospěch architektury RISC mluví fakt, že takové počítače jsou efektivní, rychlé a levnější na výrobu. Na druhou stranu existují také zastánci procesorů, které dokáží spouštět větší počet složitějších instrukcí, ačkoli mnoho z těchto instrukcí je v praxi nadbytečných. Od uvedené filozofie se odvozují počítače **CISC** (complex instruction set computer – počítač s úplnou instrukční sadou). Ve prospěch architektury CISC mluví fakt, že složitější procesory dokáží lépe zvládat požadavky stále složitějších moderních programů. U procesorů CISC mohou programy pracovat s bohatou sadou instrukcí, z nichž mnohé by v návrhu RISC bylo nutné realizovat dlouhou řadou základních instrukcí.

V 90. letech 20. století a na začátku 21. století spolu soupeřily komerčně dostupné procesory CISC a RISC o dominanci v oblasti stolních počítačů. Procesory společnosti Intel používané v počítačích PC představují příklady architektury CISC. Oproti tomu procesory PowerPC (vyvíjené v alianci společností Apple, IBM a Motorola) jsou založeny na architektuře RISC a tvořily jádro počítačů Apple Macintosh. S postupem času se výrobní náklady procesorů CISC dramaticky snížily. Procesory společnosti Intel (či jejich ekvivalenty od společnosti AMD – Advanced Micro Devices, Inc.) se nyní nacházejí téměř ve všech stolních počítačích a noteboocích (dokonce i společnost Apple nyní vyrábí počítače postavené na procesorech Intel).

Procesory CISC si sice zabezpečily své místo v oblasti stolních počítačů, ale mají vysoké nároky na odběr energie. Společnost Advanced RISC Machine (ARM) však navrhla architekturu RISC, která je navržena zejména s ohledem na nízkou spotřebu. (Společnost Advanced RISC Machine se původně nazývala Acorn Computers a nyní nese název ARM Holdings.) Procesory podle návrhu ARM, které produkují různí výrobci včetně společností Qualcomm a Texas Instruments, se proto často objevují v herních konzolách, digitálních televizorech, navigačních systémech, automobilových modulech, mobilních telefonech, smartphonech a jiné spotřební elektronice.

Nezávisle na tom, zda se jedná o architekturu RISC nebo CISC, lze strojové instrukce zařadit do třech skupin: (1) skupiny přenosu dat, (2) aritmeticko-logické skupiny a (3) řídicí skupiny.

Kdo to vlastně vynalezl?

Často nelze s jistotou označit jednoho konkrétního vynálezce nebo objevitele. Thomas Edison je považován za vynálezce žárovky, ale podobné lampy v té době vyvíjeli i jiní výzkumníci. Do jisté míry měl Edison štěstí, že právě on získal patent. Bratrům Wrightovým se připisuje vynález letadla těžšího než vzduch. Soupeřili však s mnoha konkurenty a zároveň těžili z jejich práce. Všechny je pak svým způsobem předešel Leonardo da Vinci, který si pohrával s myšlenkou létajících strojů již v patnáctém století. Dokonce i Leonardovy návrhy byly podle všeho založeny na dřívějších nápadech. V těchto případech má samozřejmě uznávaný vynálezce na své ocenění legitimní nárok. V jiných situacích však historie přiřkla zásluhu neoprávněně – příkladem je právě koncepce uloženého programu. John von Neumann byl bezpochyby geniální vědec, který si zaslouží obdiv za značný přínos informatice. Jeden z příspěvků na poli poznání, který se mu obecně připisuje – koncepci uloženého programu – patrně vyvinul J. P. Eckert na Moore School of Electrical Engineering při Pensylvánské univerzitě. John von Neumann jednoduše jako první publikoval práci, kde byla tato myšlenka zmíněna, a v počítačové tradici mu tedy zůstala role autora.

Přenos dat – skupina přenosu dat (data transfer group) zahrnuje instrukce, které požadují přesun dat z jednoho umístění do jiného. Do této kategorie patří kroky 1, 2 a 4 na obrázku 2.2. Je potřeba si povšimnout, že termíny typu „přenos" nebo „přesun" jsou pro tuto skupinu ve skutečnosti zvoleny nevhodně. Málokdy dochází k tomu, že jsou přenášená data z původního umístění vymazána. Proces popsaný instrukcí přenosu se spíše než přesunutí dat podobá jejich kopírování. Akce prováděné touto skupinou instrukcí by tedy lépe popisovaly termíny jako *kopírování* nebo *klonování*.

Když jsme se dostali k terminologickým otázkám, měli bychom zmínit, že pro přenos dat mezi procesorem a hlavní pamětí se používají speciální pojmy. Požadavek na naplnění obecného registru obsahem paměťové buňky se běžně označuje jako instrukce LOAD. Opačný požadavek na přenos obsahu registru do paměťové buňky se obecně nazývá instrukce STORE. Kroky 1 a 2 na obrázku 2.2 jsou instrukce LOAD a v případě kroku 4 se jedná o instrukci STORE.

Důležitá skupina instrukcí z kategorie přenosu dat zahrnuje příkazy pro komunikaci se zařízeními mimo kontext procesoru a operační paměti (tiskárny, klávesnice, zobrazovací zařízení, diskové jednotky atd.). Vzhledem k tomu, že tyto instrukce řídí vstupně-výstupní činnosti počítače (input/output – I/O), označují se jako **vstupně-výstupní instrukce** a podle některých klasifikací tvoří vlastní kategorii. Na druhou stranu v části 2.5 popíšeme, jak lze tyto vstupně-výstupní operace kontrolovat stejnými instrukcemi, které požadují přenosy dat mezi procesorem a operační pamětí. Proto je vhodnější považovat vstupně-výstupní instrukce za podmnožinu skupiny přenosu dat.

Aritmeticko-logické instrukce – aritmeticko-logická skupina zahrnuje instrukce, pomocí kterých řídicí jednotka požaduje provedení operací v aritmeticko-logické jednotce. Do této skupiny lze zařadit krok 3 na obrázku 3.3. Jak vyplývá z názvu, aritmeticko-logická jednotka dokáže provádět i jiné operace než základní aritmetické úkony. Mezi tyto dodatečné operace patří logické operace AND, OR a XOR, které jsme představili v kapitole 1 a kterými se budeme podrobněji zabývat v této kapitole.

Většina aritmeticko-logických jednotek je také vybavena další sadou operací, které umožňují interně posunout obsah určitého registru doprava nebo doleva. Tyto operace se označují jako operace SHIFT či ROTATE podle toho, zda bity, které „se nevejdou na konec", jsou jednoduše zahozeny (SHIFT), nebo se použijí k vyplnění prázdného místa na druhém konci (ROTATE).

Instrukce s proměnnou délkou

Kvůli jednoduššímu vysvětlení problematiky mají všechny instrukce strojového jazyka z příkladů v této kapitole (který je popsán v Příloze C) pevnou velikost dvou bajtů. Při načítání instrukce tedy procesor vždy vyvolá obsah dvou po sobě následujících paměťových buněk a dvakrát inkrementuje svůj programový čítač. Takový jednoduchý formát zvyšuje efektivitu operací načítání instrukcí a je typický pro architekturu RISC. Počítače typu CISC však pracují se strojovým jazykem, jehož instrukce mají různou délku. Dnešní procesory Intel například používají instrukce v rozsahu od jediného bajtu až po několik bajtů. Délka instrukcí přitom závisí na jejich funkci. Procesor s takovým strojovým jazykem určuje délku příchozí instrukce podle jejího operačního kódu. Procesor tedy nejdříve získá operační kód instrukce a poté na základě přijatého bitového vzoru zjistí, kolik dalších bajtů má při načítání zbytku instrukce z paměti ještě vyžádat.

Řídicí instrukce – řídicí skupina obsahuje instrukce, které neslouží k manipulaci s daty, ale kontrolují provádění programu. Do této kategorie spadá i krok 5 na obrázku 2.2, i když se jedná o nejjednodušší příklad. Tato skupina zahrnuje mnoho zajímavějších instrukcí ve strojovém repertoáru, jako např. sadu instrukcí JUMP (či BRANCH), které zajišťují, aby procesor provedl jinou instrukci než tu, která v seznamu bezprostředně následuje. Tyto instrukce JUMP se vyskytují ve dvou verzích: **nepodmíněné skoky** a **podmíněné skoky**. Příkladem první může být „přejdi na krok 5" a jako příklad druhé skupiny můžeme uvést třeba „pokud je získaná hodnota rovna 0, přejdi na krok 5". Rozdíl spočívá v tom, že podmíněný skok mění směr programu pouze v případě, že je splněna určitá podmínka. Lze to ukázat na obrázku 2.3, jehož instrukce popisují algoritmus dělení dvou hodnot a krok 3 představuje podmíněný skok, který chrání před možností dělení nulou.

Krok 1: Načti (LOAD) do registru hodnotu z paměti.
Krok 2: Načti (LOAD) do registru další hodnotu z paměti.
Krok 3: Je-li tato druhá hodnota nulová, přeskoč (JUMP) na krok 6.
Krok 4: Vyděl obsah prvního registru obsahem druhého registru a ponechej výsledek ve třetím registru.
Krok 5: Ulož (STORE) obsah třetího registru do paměti.
Krok 6: Stop.

Obrázek 2.3: Dělení hodnot uložených v paměti

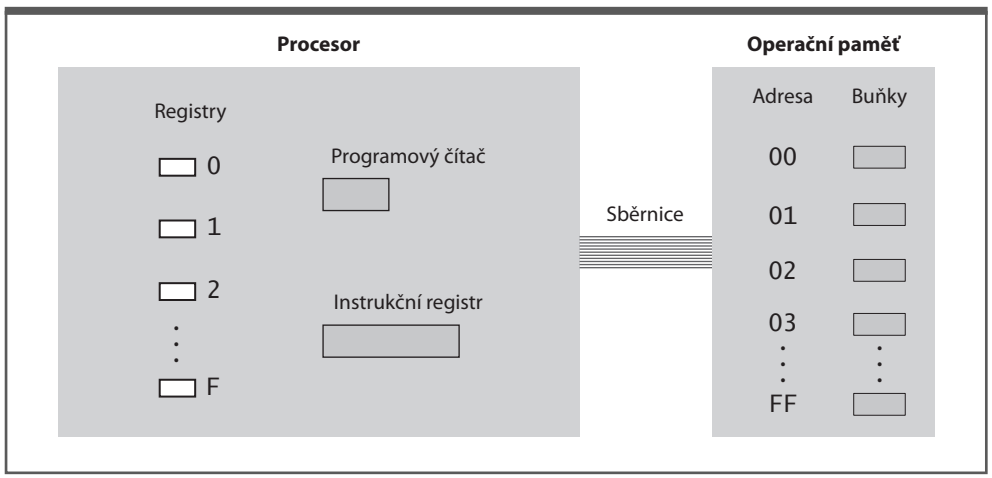
Ukázkový strojový jazyk

Zamysleme se nyní nad tím, jak jsou zakódovány instrukce typického počítače. V naší analýze budeme používat stroj, který je popsán v Příloze C. Souhrnné informace naleznete na obrázku 2.4. Tento počítač je vybaven 16 obecnými registry a 256 buňkami operační paměti, z nichž každá má kapacitu 8 bitů. Abychom se na ně mohli odkazovat, označíme registry hodnotami od 0 do 15 a paměťové buňky budeme adresovat hodnotami 0 až 255. Pro zjednodušení si tyto jmenovky a adresy budeme představovat jako hodnoty reprezentované ve dvojkové soustavě a výsledné bitové posloupnosti zkrátíme pomocí šestnáctkového zápisu. Registry tedy označíme 0 až F a paměťové buňky budou mít adresy 00 až FF.

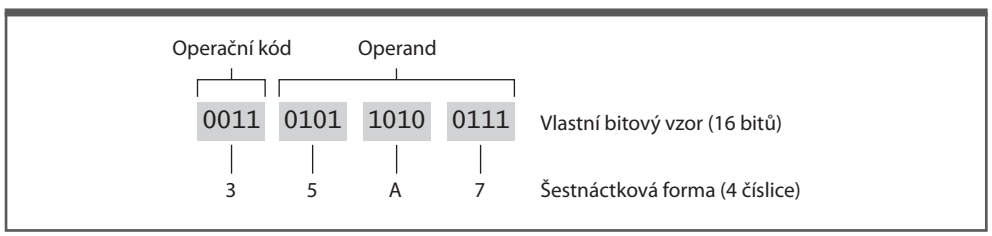
Zakódovaná verze strojové instrukce obsahuje dvě části: **operační kód** (op-code) a **operand**. Bitový vzor obsažený v operačním kódu říká, kterou základní operaci (např. STORE, SHIFT, XOR a JUMP) instrukce požaduje. Bitový vzor v operandu poskytuje podrobnější informace o operaci, která je určena operačním kódem. V případě operace STORE například hodnota složky operandu udává, který registr obsahuje ukládaná data a do které paměťové buňky se mají data uložit.

Úplný strojový jazyk ukázkového počítače (Příloha C) obsahuje pouze dvanáct základních instrukcí. Každá z těchto instrukcí je zakódována pomocí celkem 16 bitů, které lze vyjádřit pomocí čtyřech šestnáctkových číslic (obrázek 2.5). Operační kód

každé instrukce je tvořen prvními čtyřmi bity, tj. první šestnáctkovou číslicí. V Příloze C si můžete povšimnout, že tyto operační kódy jsou vyjádřeny šestnáctkovými číslicemi 1 až C. Z tabulky v Příloze C je patrné, že pokud instrukce začíná šestnáctkovou číslicí 3, jedná se o instrukci STORE, a počáteční šestnáctková číslice A označuje instrukci ROTATE.



Obrázek 2.4: Architektura počítače popsaného v Příloze C

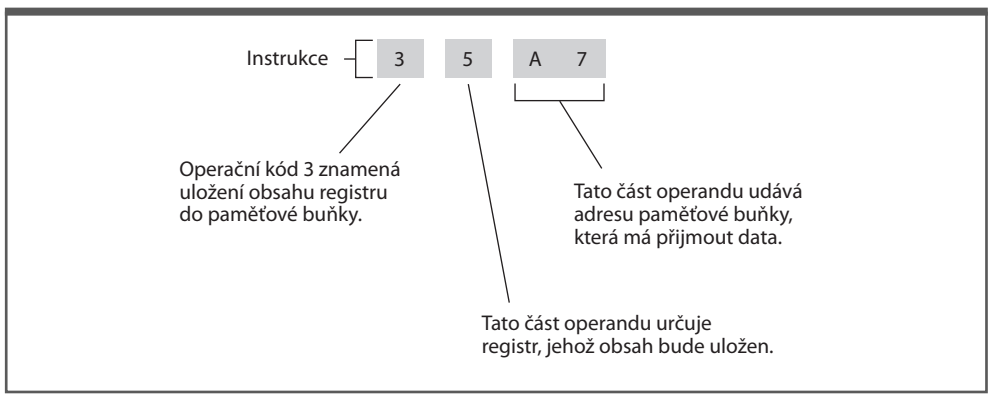


Obrázek 2.5: Struktura instrukce počítače z Přílohy C

Složka představující operand v každé instrukci ukázkového počítače obsahuje tři šestnáctkové číslice (12 bitů). S výjimkou instrukce HALT (která nevyžaduje další upřesnění) vždy poskytuje podrobnější informace týkající se obecné instrukce definované operačním kódem. Jestliže například instrukce začíná šestnáctkovou číslicí 3 (operační kód pro uložení obsahu registru), bude další šestnáctková číslice instrukce určovat ukládaný registr a poslední dvě šestnáctkové číslice budou udávat paměťovou buňku, která má data přijmout. Instrukci 35A7 (šestnáctkově) lze tedy přeložit takto: „ulož (STORE) bitový vzor zjištěný v registru 5 do paměťové buňky s adresou A7“. (Je zřejmé, jak šestnáctkový zápis zjednodušuje popis. Instrukce 35A7 je ve skutečnosti vyjádřena bitovou posloupností 0011010110100111.)

(Instrukce 35A7 také zřetelně dokládá, proč kapacita operační paměti zpravidla odpovídá mocninám dvou. Vzhledem k tomu, že 8 bitů v instrukci je vyhrazeno pro určení paměťové buňky, se kterou bude instrukce pracovat, lze odkazovat přesně

na 2^8 různých paměťových buněk. Je tedy vhodné konstruovat operační paměť s tolika buňkami, které budou mít adresy od 0 do 255. Kdyby a operační paměť obsahovala více buněk, nedokázali bychom mezi nimi pomocí instrukcí rozlišit. Jestliže by operační paměť měla buněk méně, mohli bychom napsat instrukce, které by odkazovaly na neexistující buňky.)



Obrázek 2.6: Dekódování instrukce 35A7

Jako další příklad toho, jak může složka představující operand upřesnit obecný operační kód instrukce, můžeme uvést instrukci s operačním kódem 7 (šestnáctkově), která požaduje provést operaci OR s obsahem dvou registrů. (Přesný význam operace OR se dvěma registry vysvětlíme v části 2.4. Prozatím nás zajímá pouze způsob kódování instrukcí.) V tomto případě další šestnáctková číslice určuje registr, do kterého má být výsledek umístěn, zatímco dvě poslední šestnáctkové číslice definují, které dva registry bude operace OR zpracovávat. Instrukci 70C5 lze tedy slovně popsat takto: „Proveď operaci OR s obsahem registru C a registru 5 a ulož výsledek do registru 0".

Dvě strojové instrukce LOAD se mírně liší. Jak je patrné, operační kód 1 (šestnáctkově) určuje instrukci, která do registru načítá obsah paměťové buňky, zatímco operační kód 2 (šestnáctkově) identifikuje instrukci, která do registru zapisuje určitou hodnotu. Rozdíl spočívá v tom, že složka operandu v instrukci prvního typu obsahuje adresu, avšak u druhého typu instrukce se ve stejné složce nachází konkrétní bitový vzor, který má být načten.

Můžeme si všimnout, že počítač je vybaven dvěma instrukcemi ADD: jedna umožňuje sčítat reprezentace ve dvojkovém doplňku a druhá slouží ke sčítání reprezentací s plovoucí řádovou čárkou. Tento rozdíl vychází z faktu, že sčítání bitových vzorů reprezentujících hodnoty, které jsou vyjádřeny ve dvojkovém doplňku, vyžaduje od aritmeticko-logické jednotky odlišný postup než sčítání hodnot vyjádřených v zápise s plovoucí řádovou čárkou.

Tuto část uzavřeme obrázkem 2.7, který znázorňuje zakódovanou verzi instrukce z obrázku 2.2. Předpokládáme, že sčítané hodnoty jsou uloženy ve dvojkovém doplňku na paměťových adresách 6C a 6D a součet bude umístěn do paměťové buňky s adresou 6E.

Zakódované instrukce	Překlad
156C	Do registru 5 načti bitový vzor, který se nachází v paměťové buňce na adrese 6C.
166D	Do registru 6 načti bitový vzor, který se nachází v paměťové buňce na adrese 6D.
5056	Sečti obsah registrů 5 a 6 způsobem, který se používá pro reprezentace dvojkového doplňku, a ulož výsledek do registru 0.
306E	Ulož obsah registru 0 do paměťové buňky na adrese 6E.
C000	Stop.

Obrázek 2.7: Zakódovaná verze instrukce z obrázku 2.2

Otázky a cvičení

1. Proč můžeme tvrdit, že označení *přesun* pro operaci přesunu dat mezi umístěními dat v počítači není správné?
2. Přímo v rámci instrukcí JUMP, které jsme představili výše, byl explicitně identifikován cíl svým názvem nebo číslem kroku (například „přejdi na krok 6“). Nevýhoda této metody spočívá v tom, že pokud se název (nebo číslo) instrukce později změní, musíme vyhledat všechny skoky na tuto instrukci a změnit název i v nich. Popište jiný způsob, jak by bylo možné vyjádřit instrukci JUMP, aby neobsahovala explicitní název cíle.
3. Je instrukce „Pokud se 0 rovná 0, přejdi na krok 7“ podmíněný nebo nepodmíněný skok? Vysvětlete svou odpověď.
4. Zapište ukázkový program na obrázku 2.7 formou skutečných bitových řetězců.
5. Následující instrukce jsou zapsány ve strojovém jazyce, který je popsán v Příloze C. Vyjádřete je slovně.

   a. 368A **b.** BADE **c.** 803C **d.** 40F4
6. Jaký je rozdíl mezi instrukcemi 15AB a 25AB ve strojovém jazyce z Přílohy C?

7. Následuje několik slovních instrukcí. Přeložte je do strojového jazyka z Přílohy C.
   a. Načti (LOAD) do registru číslo 3 šestnáctkovou hodnotu 56.
   b. Posuň (ROTATE) obsah registru číslo 5 o tři bity doprava.
   c. Proveď operaci AND s obsahem registru A a registru 5 a ulož výsledek do registru 0.

2.3: Provádění programu

Když se počítač řídí programem uloženým ve své paměti, kopíruje podle potřeby jeho instrukce z paměti do procesoru. Jakmile jsou instrukce přeneseny, procesor je dekóduje a provede. Pořadí načítání instrukcí z paměti odpovídá pořadí, ve kterém jsou instrukce v paměti uloženy, jestliže toto pořadí nezmění instrukce JUMP.

Chceme-li porozumět tomu, jak celý proces realizace programu probíhá, musíme se zmínit o dvou speciálních registrech procesoru: **instrukčním registru** a **programovém čítači** (viz opět obrázek 2.4). Instrukční registr uchovává prováděnou instrukci. Programový čítač obsahuje adresu další vykonávané instrukce a počítač tedy díky němu dokáže sledovat, ve kterém místě programu se právě nachází.

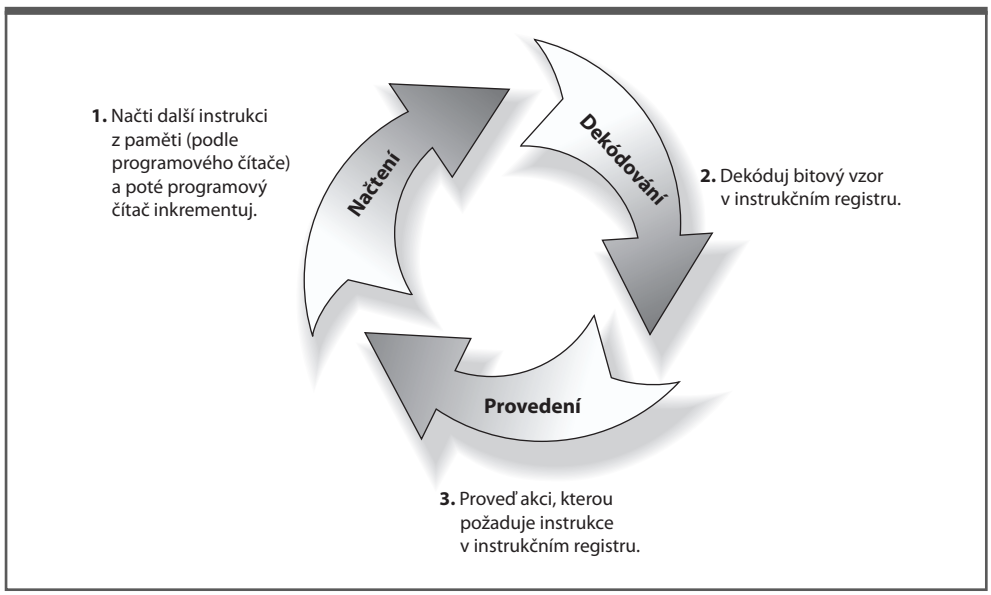
Procesor při své činnosti neustále opakuje algoritmus, který popisuje trojfázový proces označovaný jako **strojový cyklus**. Strojový cyklus obsahuje následující kroky: načtení, dekódování a provedení instrukce (obrázek 2.8). V kroku načtení si procesor z hlavní paměti vyžádá instrukci, která je uložena na adrese uvedené v programovém čítači. Vzhledem k tomu, že instrukce našeho počítače mají délku dvou bajtů, vyžaduje tento proces načtení obsahu dvou buněk operační paměti. Procesor poté umístí instrukci získanou z paměti do svého instrukčního registru a dvakrát inkrementuje programový čítač, aby obsahoval adresu další instrukce uložené v paměti. Programový čítač je tedy připraven na načtení další instrukce.

Když se instrukce nachází v instrukčním registru, procesor ji dekóduje. Přitom je nutné v závislosti na operačním kódu instrukce rozdělit složku operandu na příslušné komponenty.

Procesor poté instrukci provede tak, že aktivuje elektronické obvody specializované na konkrétní úkol. Jestliže se například jedná o instrukci načtení z paměti, procesor odešle příslušné signály operační paměti, vyčká, dokud paměť neodešle data, a poté tato data umístí do požadovaného registru. Když jde o instrukci aritmetické operace, procesor aktivuje obvod aritmeticko-logické jednotky s vhodnými registry jako vstupy a vyčká, než jednotka vypočítá odpověď a uloží ji do určeného registru.

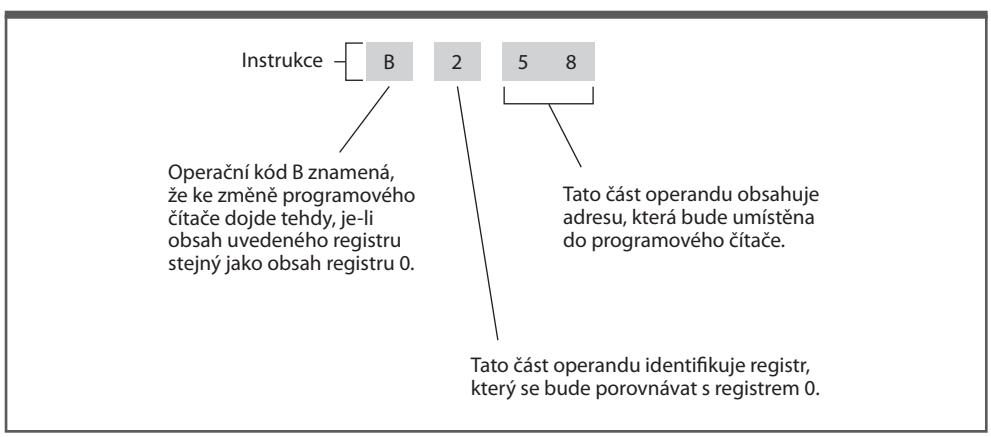
Po provedení instrukce v instrukčním registru procesor opět zahájí strojový cyklus načtením další instrukce. Díky tomu, že byl programový čítač na konci předchozího načtení inkrementován, opět nyní procesoru poskytne správnou adresu.

Do jisté míry speciální případ představuje spuštění instrukce JUMP. Uvažujme například instrukci B258 (obrázek 2.9), která znamená „skoč (JUMP) na instrukci na adrese 58 (šestnáctkově), jestliže se obsah registru 2 rovná obsahu registru 0". V tomto případě krok provádění strojového cyklu začíná porovnáním registrů 2 a 0. Obsahují-li odlišné bitové posloupnosti, je krok provádění ukončen a začne další strojový cyklus. Jestliže se však oba registry rovnají, počítač během kroku provádění umístí do svého programového čítače hodnotu 58 (šestnáctkově). V této situaci další krok načtení nalezne v programovém čítači hodnotu 58, takže další instrukce, kterou počítač načte a provede, se bude nacházet na uvedené adrese.



Obrázek 2.8: Strojový cyklus

Pokud by se jednalo o instrukci B058, programový čítač by se změnil v závislosti na tom, zda by se obsah registru 0 rovnal obsahu registru 0. Tyto registry se ovšem shodují a musí tedy obsahovat stejný obsah. Každá instrukce ve tvaru B0XY tedy způsobí skok na paměťové umístění XY bez ohledu na skutečný obsah registru 0.



Obrázek 2.9: Dekódování instrukce B258

Porovnání výkonu počítačů

Při nákupu osobního počítače si můžeme všimnout, že se k porovnání různých modelů často používají taktovací frekvence. Počítačové **hodiny** (clock) jsou realizovány jako obvod zvaný oscilátor. Tento obvod generuje pulzy, které umožňují koordinovat činnost počítače. Čím rychleji obvod oscilátoru generuje pulzy, tím rychleji počítač provádí svůj strojový cyklus. Taktovací frekvence se měří v hertzech (zkratka Hz), kde jeden Hz odpovídá jednomu cyklu (či pulzu) za sekundu. Typické hodinové frekvence pracovních stanic se pohybují od několika set MHz (starší typy) až po několik GHz. (Zkratka MHz označuje megahertz, což je milion Hz. GHz je zkratka gigahertzu, který se rovná 1 000 MHz.)

Různé procesory však mohou v závislosti na své konstrukci za jeden hodinový cyklus provést různě složité operace. Samotná hodinová frekvence tedy k porovnání počítačů s různými procesory nestačí. Při porovnání počítače s procesorem Intel s počítačem, který obsahuje procesor ARM, je rozumnější porovnat výkon pomocí **srovnávacího testování** (benchmarking). Přitom se měří výkon počítačů, které provádějí stejný program označovaný jako srovnávací test (benchmark). Vybereme-li srovnávací testy, které reprezentují různé typy aplikací, získáme v jednotlivých tržních segmentech porovnání s vypovídací hodnotou.

Příklad provedení programu

Projděme nyní strojový cyklus programu na obrázku 2.7, který načte dvě hodnoty z operační paměti, spočítá jejich součet a uloží výsledek do buňky operační paměti. Nejdříve musíme program umístit někam do paměti. V našem příkladu budeme předpokládat, že program je uložen na adresách následujících po sobě počínaje adresou A0 (šestnáctkově). Pokud je program uložen uvedeným způsobem, můžeme jej v počítači spustit tak, že umístíme adresu první instrukce (A0) do programového čítače a poté aktivujeme počítač (obrázek 2.10).

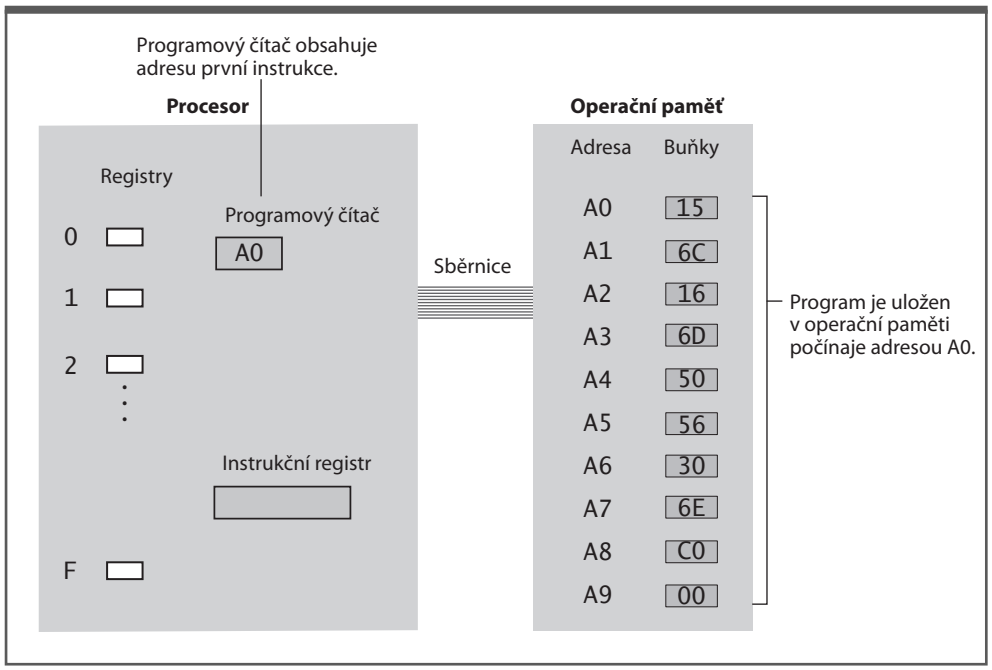
Procesor zahájí strojový cyklus načtením instrukce. Zjistí instrukci, která je uložena na adrese A0 operační paměti, a umístí tuto instrukci (156C) do svého instrukčního registru (obrázek 2.11a). Jak jsme již uvedli, instrukce našeho počítače mají délku 16 bitů (dva bajty). Celá načítaná instrukce tedy zabírá paměťové buňky s adresami A0 a A1. Konstrukce procesoru s tím počítá. Procesor tedy načte obsah obou buněk a umístí přijaté bitové posloupnosti do instrukčního registru s délkou 16 bitů. Procesor následně zvýší hodnotu programového čítače o dvě, aby registr obsahoval adresu další instrukce (obrázek 2.11b). Na konci kroku načítání v prvním strojovém cyklu budou v programovém čítači a instrukčním registru umístěna následující data:

```
Programový čítač: A2
Instrukční registr: 156C
```

Procesor pak analyzuje instrukci ve svém instrukčním registru a zjistí, že má do registru 5 načíst obsah paměťové buňky na adrese 6C. K vlastnímu načtení dojde v kroku provádění strojového cyklu a procesor vzápětí spustí další strojový cyklus.

Tento cyklus začíná načtením instrukce 166D ze dvou paměťových buněk, z nichž první leží na adrese A2. Procesor vloží tuto instrukci do instrukčního registru a zvýší programový čítač na hodnotu A4. Programový čítač a instrukční registr tedy aktuálně obsahují tyto hodnoty:

```
Programový čítač: A4
Instrukční registr: 166D
```



Obrázek 2.10: Program z obrázku 2.7 uložený v operační paměti a připravený ke spuštění

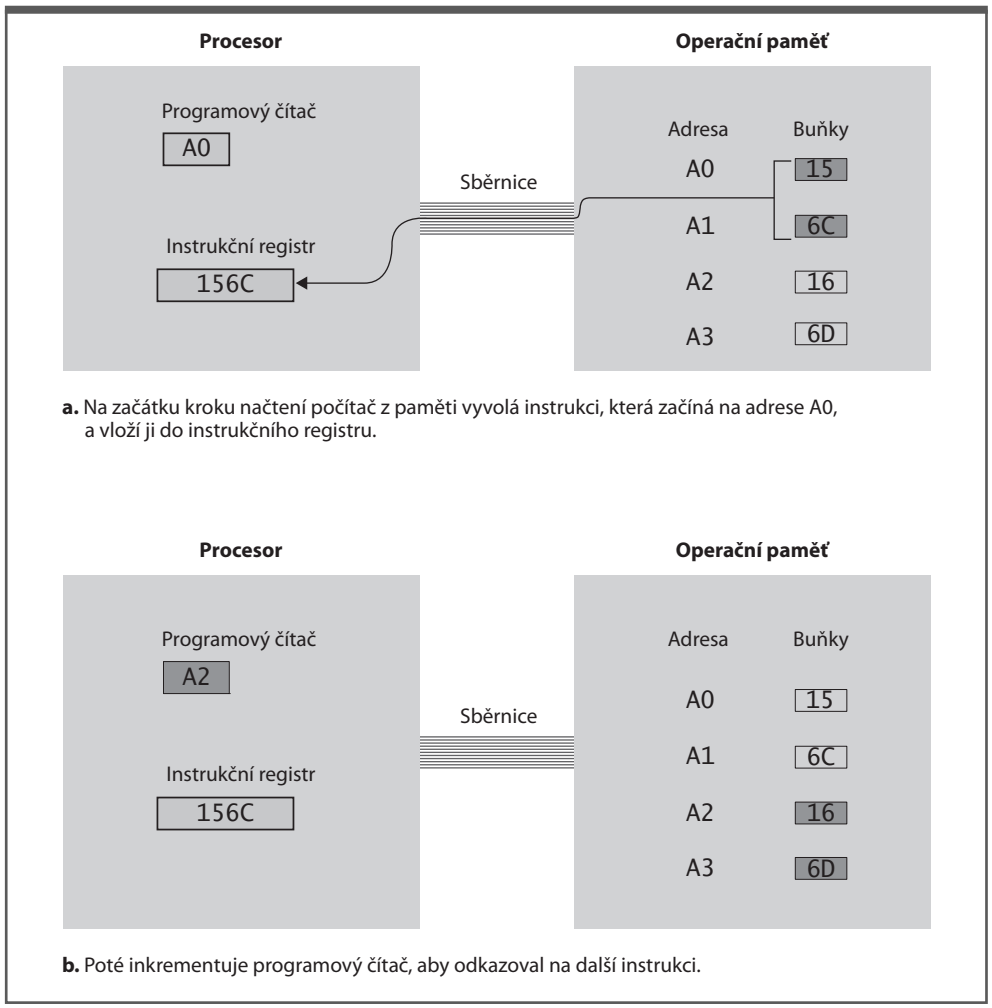
Procesor nyní dekóduje instrukci A66D a určí, že má do registru 6 načíst obsah paměťové adresy 6D. Následně instrukci provede. V této fázi obsahuje registr 6 načtenou hodnotu.

Vzhledem k tomu, že v programovém čítači se nyní nachází hodnota A4, procesor extrahuje další instrukci, která začíná na této adrese. V důsledku toho bude do instrukčního registru umístěna hodnota 5056 a programový čítač je inkrementován na A6. Následně procesor dekóduje obsah svého instrukčního registru a provede požadovanou operaci. Přitom aktivuje obvod pro sčítání čísel ve dvojkovém doplňku a jako vstupy operace použije registry 5 a 6.

V tomto kroku aritmeticko-logická jednotka provede příslušné sčítání, umístí výsledek do registru 0 (podle pokynu řídicí jednotky) a oznámí řídicí jednotce, že svou práci dokončila. Procesor poté zahájí další strojový cyklus. Opět pomocí programového čítače vyvolá další instrukci (306E) ze dvou paměťových buněk, které začínají v umístění A6, a inkrementuje hodnotu programového čítače na A8. Uvedená instrukce je poté dekódována a spuštěna. V této fázi je součet uložen do paměťového umístění 6E.

Další instrukce je získána z umístění v paměti, které začíná adresou A8, a hodnota programového čítače se zvýší na AA. Vzápětí procesor dekóduje, že instrukční registr obsahuje instrukci stop (C000). Počítač se tedy zastaví v kroku provádění strojového cyklu a program skončí.

Lze shrnout, že provádění programu uloženého v paměti využívá stejný postup, jakým lidé sledují podrobný seznam pokynů. Zatímco lidé kontrolují svůj postup tím, že si splněné kroky odškrtávají, procesor dosaženou pozici sleduje pomocí programového čítače. Když zjistíme, který další pokyn je potřeba provést, přečteme příslušnou instrukci a přeložíme význam textu. Poté provedeme úkol, který instrukce požaduje, a vrátíme se k další instrukci seznamu. Stejně postupuje i procesor, který provede instrukci ve svém instrukčním registru a potom do registru načte následující instrukci.



a. Na začátku kroku načtení počítač z paměti vyvolá instrukci, která začíná na adrese A0, a vloží ji do instrukčního registru.

b. Poté inkrementuje programový čítač, aby odkazoval na další instrukci.

Obrázek 2.11: Provedení kroku načtení strojového cyklu

Programy versus data

Do operační paměti počítače je možné současně uložit mnoho programů za předpokladu, že se nacházejí na odlišných adresách. Program, který počítač provede po svém spuštění, lze určit patřičným nastavením programového čítače.

V operační paměti se však zároveň nacházejí data, která jsou také zakódována pomocí nul a jedniček. Počítač proto nedokáže zjistit, co jsou data a co programy. Kdybychom programovému čítači místo adresy požadovaného programu přiřadili adresu nějakých dat, procesor by bez váhání extrahoval bitové vzory dat, jako by se jednalo o instrukce, a začal je spouštět. Výsledek by závisel na obsahu příslušných dat.

Neměli bychom však z toho vyvozovat, že jednolitá struktura programů i dat v paměti je něco nežádoucího. Ve skutečnosti nabízí užitečné možnosti, protože dovoluje, aby jeden program manipuloval jinými programy (nebo dokonce měnil sama sebe) stejně, jako by pracoval s daty. Můžeme si například představit program, který se upravuje v závislosti na své interakci s prostředím a má tedy schopnost učení, nebo jiný program, který řeší předkládané problémy tak, že píše a spouští jiné programy.

Otázky a cvičení

1. Paměťové buňky na adresách 00 až 05 operační paměti počítače popsaného v Příloze C obsahují (šestnáctkově zapsané) bitové vzory, které jsou uvedeny v následující tabulce:

Adresa	Obsah
00	14
01	02
02	34
03	17
04	C0
05	00

   Spustíme-li počítač, jehož programový čítač obsahuje hodnotu 00, který bitový vzor se bude po zastavení počítače nacházet v paměťové buňce s šestnáctkovou adresou 17?

2. Paměťové buňky na adresách B0 až B8 operační paměti počítače popsaného v Příloze C obsahují (šestnáctkově zapsané) bitové vzory, které jsou uvedeny v následující tabulce:

Adresa	Obsah
B0	13
B1	B8
B2	A3
B3	02
B4	33
B5	B8
B6	C0
B7	00
B8	0F

   a. Jestliže programový čítač při spuštění obsahuje hodnotu B0, který bitový vzor bude v registru číslo 3 po provedení první instrukce?

   b. Který bitový vzor se bude nacházet v paměťové buňce B8 po provedení instrukce stop?

3. Paměťové buňky na adresách A4 až B1 operační paměti počítače popsaného v Příloze C obsahují (šestnáctkově zapsané) bitové vzory, které jsou uvedeny v následující tabulce:

Adresa	Obsah
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02

AE	B0
AF	AA
B0	C0
B1	00

Při hledání odpovědí na následující otázky předpokládejte, že při spuštění počítače obsahuje jeho programový čítač hodnotu A4.

a. Co je uloženo v registru 0 po prvním provedení instrukce na adrese AA?

b. Co je uloženo v registru 0 po druhém provedení instrukce na adrese AA?

c. Kolikrát je do zastavení počítače provedena instrukce na adrese AA?

4. Paměťové buňky na adresách F0 až F9 operační paměti počítače popsaného v Příloze C obsahují (šestnáctkově zapsané) bitové vzory, které jsou uvedeny v následující tabulce:

Adresa	**Obsah**
F0	20
F1	C0
F2	30
F3	F8
F4	20
F5	00
F6	30
F7	F9
F8	FF
F9	FF

Spustíme-li počítač, jehož programový čítač obsahuje hodnotu F0, co počítač provede po dosažení instrukce na adrese F8?

2.4: Aritmeticko-logické instrukce

Jak jsme již uvedli, skupina aritmeticko-logických instrukcí obsahuje instrukce, které požadují provedení aritmetických, logických a posunových operací. V této části se na uvedený typ operací zaměříme podrobněji.

Logické operace

Logické operace AND, OR a XOR (vylučovací nebo) jsme představili v kapitole 1. Jedná se o operace, které kombinují dva vstupní bity a poskytují jeden výstupní bit. Tyto operace lze rozšířit také na kombinaci dvou řetězců bitů, ze kterých získáme jeden výstupní řetězec tak, že základní operace aplikujeme na jednotlivé sloupce. Výsledek operace AND se vzory 10011010 a 11001001 například bude

```
  10011010
AND 11001001
  10001000
```

kde výsledek operace AND s dvojicí bitů v každém sloupci vidíte v posledním řádku. Podobně operace OR a XOR s těmito bitovými vzory poskytnou výsledky

```
    10011010        10011010
OR  11001001    XOR 11001001
    11011011        01010011
```

Operace AND se často používá k umístění nul do jedné části bitového řetězce, aniž by se změnila druhá část. Zamysleme se například nad tím, co se stane, pokud bude bajt 00001111 prvním operandem operace AND. Bez znalosti obsahu druhého operandu můžeme s jistotou říci, že čtyři nejvýznamnější bity výsledku budou nulové. Navíc je jasné, že čtyři nejméně významné bity výsledku budou beze změny kopírovat příslušnou část druhého operandu, jak je zřejmé z následující ukázky:

```
    00001111
AND 10101010
    00001010
```

Toto použití operace AND představuje příklad procesu, který se nazývá **maskování**. Jeden operand označovaný jako **maska** zde určuje, která část druhého operandu ovlivní výsledek. V případě operace AND poskytuje maskování výsledek, který částečně replikuje jeden z operandů. Zbývající pozice výsledku jsou pak vyplněny nulami.

Taková operace je užitečná při manipulaci s **bitovou mapou**, tj. řetězcem bitů, kde každý bit znamená přítomnost či nepřítomnost určitého objektu. S bitovými mapami jsme se již setkali v kontextu reprezentace obrázků, kde každý bit souvisí s pixelem. Jako jiný příklad můžeme uvést řetězec 52 bitů, kde každý bit odpovídá konkrétní hrací kartě. Tento řetězec může reprezentovat rozdané karty – stačí nastavit jedničku pro 5 bitů, které znamenají karty v ruce, a nulu pro všechny ostatní karty. Podobně bitová mapa s 52 bity, z nichž třináct má hodnotu 1, může představovat rozdané karty v bridži, nebo lze pomocí bitové mapy s 32 bity znázornit, kolik z celkem 32 druhů zmrzliny je v prodeji.

Předpokládejme tedy, že 8 bitů v paměťové buňce slouží jako bitová mapa. Chceme přitom zjistit, zda je přítomen objekt, který souvisí s třetím bitem zleva. Stačí provést operaci AND s celým bajtem a maskou 00100000. Tímto způsobem dostaneme bajt se samými nulami tehdy a právě tehdy, jestliže je třetí bit zleva v samotné bitové mapě také nulový. Program tedy může využít výsledek operace AND v instrukci podmíněného větvení. Navíc platí, že pokud má třetí bit z levého konce bitové mapy hodnotu 1 a chceme změnit hodnotu tohoto bitu na 0, aniž bychom ovlivnili jiné bity, můžeme provést operaci AND s bitovou mapou a maskou 11011111 a výsledek následně uložit na místo původní bitové mapy.

Operace AND tedy dovoluje okopírovat část bitového řetězce a umístit do nekopírované části nuly. Naproti tomu pomocí operace OR lze kopírovat část řetězce a jeho zbývající část vyplnit jedničkami. Přitom se opět používá maska, ale tentokrát se duplikované bitové pozice označují nulami a pomocí jedniček jsou určeny pozice, které duplikovány nebudou. Operace OR s libovolným bajtem a řetězcem 11110000 například poskytne výsledek, který bude ve čtyřech nejvýznamnějších bitech obsa-

hovat jedničky, zatímco zbývající bity budou kopírovat čtyři nejméně významné bity druhého operandu, jak je patrné z následující ukázky:

```
   11110000
OR 10101010
   11111010
```

Z toho je zřejmé, že zatímco maska 11011111 umožňuje s operací AND vynutit nulu ve třetím bitu z významnější strany bajtu, pomocí masky 00100000 a operace OR lze na dané pozici vynutit hodnotu 1.

Operace XOR se obvykle používá ke generování doplňku bitového řetězce. Operace XOR, která má na vstupu libovolný bit a masku se samými jedničkami, poskytne doplněk daného bajtu. Všimněme si například vztahu mezi druhým operandem a výsledkem následující ukázky:

```
   11111111
XOR 10101010
   01010101
```

Ve strojovém jazyce popsaném v Příloze C se pro logické operace OR, AND a XOR používají operační kódy 7, 8 a 9. Každá z těchto instrukcí požaduje provedení příslušné logické operace s obsahem dvou uvedených registrů a umístění výsledku do dalšího definovaného registru. Instrukce 7ABC například znamená, že výsledek operace OR s obsahem registrů B a C bude uložen do registru A.

Rotační a posunové operace

Operace ze skupiny rotačních a posunových operací umožňují přemístit bity v rámci registru. Často umožňují řešit problémy se zarovnáním hodnot. Tyto operace lze rozdělit podle směru posunu (doprava či doleva) a na základě toho, zda se jedná o kruhový proces. V této klasifikaci se vyskytují různé odchylky a používají se odlišné termíny. Podívejme se nyní zběžně na základní principy.

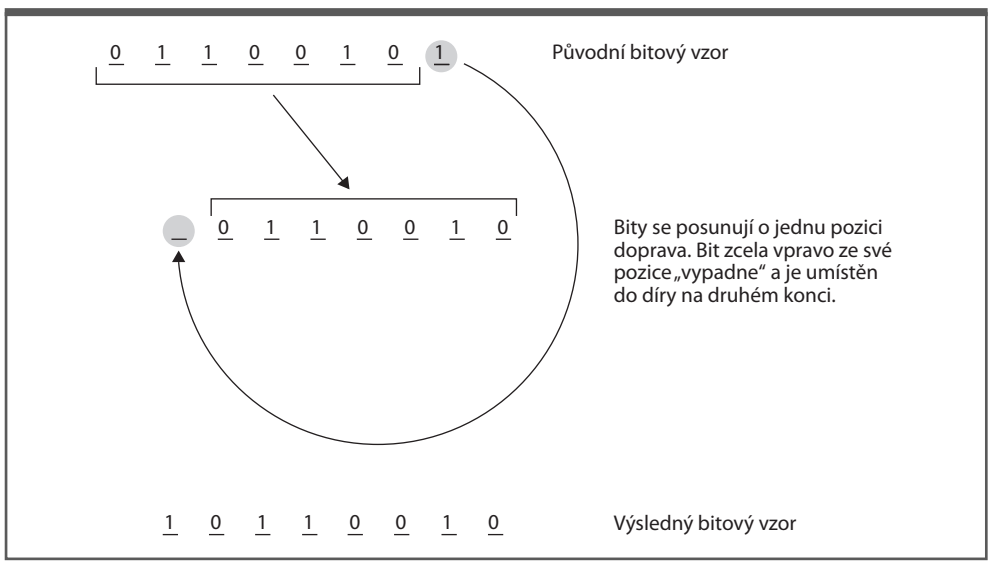
Uvažujme registr, který obsahuje bajt s řadou bitů. Jestliže posuneme obsah o 1 bit doprava, může pravý bit jakoby „odpadnout“ a na levém konci se může objevit „díra“. Podle toho, co se stane s tímto přebytečným bitem a nově vzniklou dírou, se rozlišují různé skupiny posunových operací. Jedna metoda vyplňuje díru (v tomto případě na levém konci) bitem, který se nevešel na druhý konec. Výsledkem je **kruhový posun**, který se také označuje jako **rotace**. Jestliže tedy osmkrát provedeme pravý kruhový posun se vzorem délky jednoho bajtu, získáme stejnou posloupnost bitů, jakou jsme měli na začátku.

Jiný postup zahazuje bit, který se nevešel na konec, a mezeru na druhém konci vždy vyplňuje nulou. Operace tohoto typu se často označují termínem **logický posun**. Pomocí takových posunů doleva lze čísla v dvojkovém doplňku násobit dvěma. Posun binárních číslic doleva totiž odpovídá násobení dvěma, podobně jako posun číslic v desítkové soustavě znamená násobení deseti. Obdobně je možné dělit dvěma pouhým posunem binárního řetězce doprava. V každém posunu je nutné dbát na zachování znaménkového bitu, který se používá v některých systémech číselné reprezentace. Často se tedy můžeme setkat s posuny doprava, které vždy vyplňují díru (která se objevuje na pozici znaménkového bitu) původní hodnotou tohoto bitu. Posuny, které nemění hodnotu znaménkového bitu, se někdy nazývají **aritmetické posuny**.

Z různých možností posunových a rotačních instrukcí nalezneme ve strojovém jazyce z Přílohy C pouze pravý kruhový posun, který je označen operačním kódem A. V tomto případě první šestnáctková číslice operandu určuje posouvaný registr a zbytek operandu udává, o kolik bitů má obsah registru rotovat. Instrukce A501 tedy znamená „Posuň (ROTATE) obsah registru 5 doprava o 1 bit". Jestliže registr 5 původně obsahoval například bitový vzor 65 (šestnáctkově), bude se v něm po provedení této instrukce nacházet hodnota B2 (obrázek 2.12). (Pomocí kombinace instrukcí strojového jazyka z Přílohy C lze vytvořit i jiné instrukce posunových a rotačních operací. Vzhledem k tomu, že registr má délku 8 bitů, poskytuje například pravý kruhový posun o 3 bity stejný výsledek jako levý kruhový posun o 5 bitů.)

Aritmetické operace

O aritmetických operacích sčítání, odečítání, násobení a dělení jsme se sice již zmiňovali, musíme ale ještě vyjasnit některé souvislosti. V prvé řadě jsme si ukázali, že odečítání lze simulovat pomocí sčítání a negace. Kromě toho násobení je pouhé opakované sčítání a dělení lze provést jako opakované odečítání. (Šest děleno dvěma jsou tři, protože od šesti je možné odečíst tři dvojky.) Z tohoto důvodu se některé jednoduché procesory navrhují pouze s instrukcemi sčítání, případně sčítání a odečítání.



Obrázek 2.12: Rotace bitového vzoru 65 (šestnáctkově) o jeden bit doprava

Neměli bychom také zapomenout, že všechny aritmetické operace mají různé varianty. S tímto jevem jsme se již setkali v souvislosti s operacemi sčítání, které nabízí ukázkový počítač z Přílohy C. Pokud mají být například sčítané hodnoty uloženy ve dvojkovém doplňku, je nutné sčítání provést prostě po jednotlivých sloupcích. Jestliže jsou však operandy uloženy jako hodnoty s plovoucí řádovou čárkou, musí proces sčítání nejdříve extrahovat mantisu každého čísla, v závislosti na složkách exponentu je posunout doprava nebo doleva, zkontrolovat znaménkové bity, sečíst a převést výsledek do formátu s plovoucí řádovou čárkou. I když se tedy u obou operací jedná o sčítání, počítač pokaždé postupuje jinak.

Otázky a cvičení

1. Proveďte uvedené operace.

a.
```
  01001011
AND 10101011
```

b.
```
  10000011
AND 11101100
```

c.
```
  11111111
AND 00101101
```

d.
```
  01001011
OR 10101011
```

e.
```
  10000011
OR 11101100
```

f.
```
  11111111
OR 00101101
```

g.
```
  01001011
XOR 10101011
```

h.
```
  10000011
XOR 11101100
```

i.
```
  11111111
XOR 00101101
```

2. Chcete izolovat prostřední 4 bity z bajtu tak, že umístíte do zbývajících čtyřech bitů nuly, aniž byste změnili hodnotu prostředních 4 bitů. Kterou masku a operaci přitom musíte použít?

3. Chcete získat doplněk prostředních 4 bitů z bajtu a ponechat zbývající 4 bity beze změny. Kterou masku a operaci přitom musíte použít?

4. **a.** Provedete operaci XOR s prvními 2 bity řetězce bitů a poté pokračujete dalšími bity řetězce. Operace XOR přitom bude přijímat aktuální a předchozí bit řetězce. Jak výsledek souvisí s počtem jedniček, které řetězec obsahuje?

b. Jak tato úloha souvisí se zjišťováním vhodného paritního bitu při kódování zprávy?

5. Často je výhodné použít logickou operaci namísto numerické. Logická operace AND například kombinuje 2 bity stejným způsobem jako násobení. Která logická operace je téměř stejná jako sečtení dvou bitů a jaký problém se přitom objevuje?

6. Pomocí které logické operace spolu s jakou maskou lze změnit kódy ASCII malých písmen na kódy velkých písmen? Jak je tomu při změně z velkých písmen na malá?

7. Jaký je výsledek provedení pravého kruhového posunu o 3 bity u následujících bitových řetězců:

a. 01101010 **b.** 00001111 **c.** 01111111

8. Jaký je výsledek provedení levého kruhového posunu o 1 bit u následujících bajtů, které jsou vyjádřeny v šestnáctkovém zápisu? Svou odpověď zapište v šestnáctkové formě.

a. AB **b.** 5C **c.** B7 **d.** 35

9. Pravý kruhový posun o 3 bity v řetězci s 8 bity je ekvivalentní levému kruhovému posunu o kolik bitů?

10. Který bitový vzor vyjadřuje součet čísel 01101010 a 11001100, jestliže tyto vzory reprezentují hodnoty uložené v zápisu dvojkového doplňku? Jak je tomu v případě, že vzory reprezentují hodnoty uložené ve formátu s plovoucí řádovou čárkou, který je popsán v kapitole 1?

11. Pomocí strojového jazyka z Přílohy C napište program, který umístí hodnotu 1 do nejvýznamnějšího bitu paměťové buňky s adresou A7, aniž by změnil zbývající bity této buňky.
12. Pomocí strojového jazyka z Přílohy C napište program, který zkopíruje prostřední 4 bity z paměťové buňky E0 do nejméně významných čtyř bitů paměťové buňky E1 a zároveň uloží nuly do čtyřech nejvýznamnějších bitů buňky na adrese E1.

2.5: Komunikace s jinými zařízeními

Operační paměť a procesor tvoří jádro počítače. V této části se budeme zabývat tím, jak toto jádro, které budeme označovat jako počítač, komunikuje s periferními zařízeními, k nimž patří systémy hromadného úložiště, tiskárny, klávesnice, myši, zobrazovací jednotky, digitální fotoaparáty a dokonce i jiné počítače.

Role řadičů

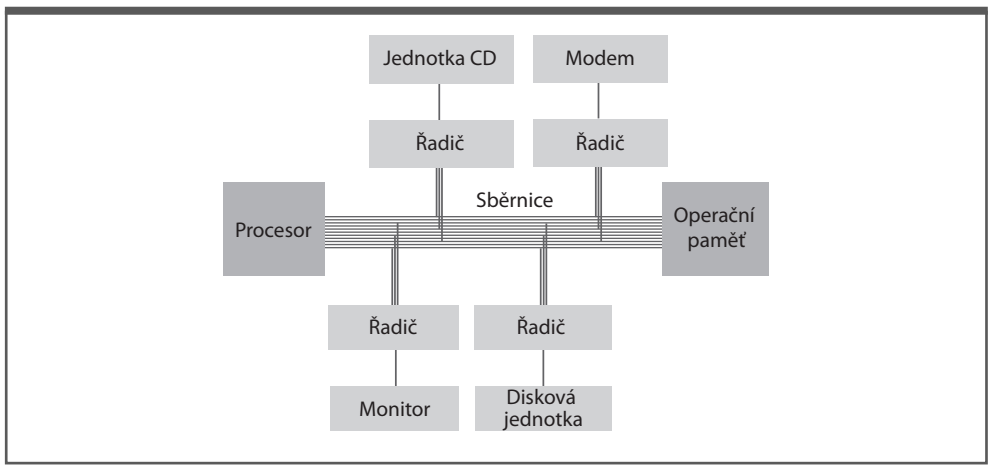
Komunikaci mezi počítačem a jinými zařízeními obvykle zajišťuje prostředník, který se označuje jako **řadič** (controller). V případě osobního počítače může být řadič tvořen elektronickými obvody, které jsou nedílnou součástí základní desky, nebo se kvůli vyšší flexibilitě může jednat o přídavnou kartu, kterou lze zasunout do patice na základní desce. V každém případě je konektor kabelově připojen buď k periferním zařízením uvnitř počítače, nebo ke konektoru označovanému jako **port** na zadní straně počítače, kam lze připojit externí zařízení. Tyto řadiče mohou být malé počítače samy o sobě a mohou obsahovat vlastní paměťové obvody a jednoduchý procesor, který realizuje program řídicí činnost řadiče.

Řadič převádí zprávy a data mezi různými formáty, které jsou kompatibilní s interními parametry počítače a připojeného periferního zařízení. Řadiče se původně navrhovaly pro konkrétní typ zařízení. Při nákupu nového periferního zařízení tedy bylo často nutné pořídit i nový řadič.

V posledních letech se v oblasti osobních počítačů prosazují standardy jako např. **USB (universal serial bus)** a **FireWire**, kde jediný řadič dokáže obsluhovat různá zařízení. Jediný řadič USB lze například použít jako rozhraní mezi počítačem a libovolnou sadou zařízení, která jsou kompatibilní se standardem USB. Seznam zařízení, která jsou v současnosti na trhu a dokáží komunikovat s řadiči USB, zahrnuje myši, tiskárny, skenery, zařízení hromadného úložiště, digitální fotoaparáty i smartphony.

Každý řadič komunikuje se samotným počítačem pomocí připojení ke stejné sběrnici, která spojuje procesor a operační paměť počítače (viz obrázek 2.13). Díky této pozici dokáže monitorovat signály přenášené mezi procesorem a operační pamětí a také na sběrnici odesílat vlastní signály.

Uvedené uspořádání umožňuje, aby procesor komunikoval s řadiči připojenými ke sběrnici stejným způsobem, jakým komunikuje s pamětí. Když procesor potřebuje odeslat zprávu řadiči, nejdříve příslušný bitový řetězec sestaví v jednom ze svých obecných registrů. Následně procesor provede instrukci podobnou instrukci STORE, která „uloží“ bitový řetězec do řadiče. K příjmu zprávy od řadiče se obdobně používá instrukce podobná instrukci LOAD.



Obrázek 2.13: Řadiče připojené ke sběrnici počítače

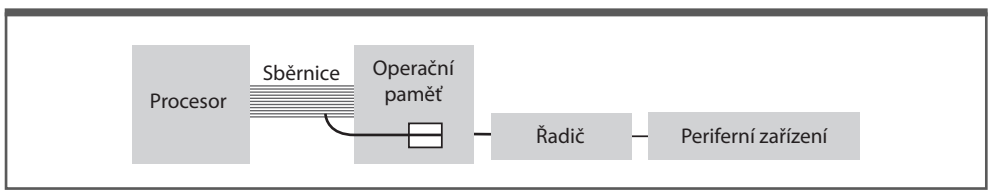
V některých počítačových architekturách je přenos dat do řadičů a zpět řízen stejnými operačními kódy LOAD a STORE, které již slouží ke komunikaci s operační pamětí. V těchto případech jsou řadiče zkonstruovány tak, aby reagovaly na odkazy na jedinečné sady paměťových adres, zatímco operační paměť odkazy na stejné adresy ignoruje. Když tedy procesor odešle sběrnici zprávu, která požaduje uložení bitové posloupnosti do paměťového umístění přiřazeného řadiči, není tento řetězec bitů ve skutečnosti „uložen" do operační paměti, ale do řadiče. Obdobně platí, že pokusí-li se procesor načíst data z takového paměťového umístění (pomocí instrukce LOAD), dostane bitovou posloupnost od řadiče a nikoli z paměti. Komunikační systém tohoto typu se označuje jako **vstup a výstup** zobrazený do paměti („mapovaný do paměti", memory-mapped I/O), protože vstupně-výstupní zařízení počítače se zdánlivě nacházejí v různých místech v paměti (viz obrázek 2.14).

Alternativu vstupu a výstupu zobrazeného do paměti představují speciální operační kódy strojového jazyka počítače, které směrují přenosy mezi řadiči a procesorem. Instrukce s těmito operačními kódy se nazývají vstupně-výstupní instrukce. Jestliže by například autoři jazyka popsaného v Příloze C zvolili tento přístup, mohl by jazyk obsahovat instrukce typu F5A3 s významem „Ulož (STORE) obsah registru 5 do řadiče identifikovaného bitovou posloupností A3".

Přímý přístup do paměti

Vzhledem k tomu, že řadič je připojen k počítačové sběrnici, může využít nanosekund, kdy sběrnici nepoužívá procesor, k vlastní komunikaci s operační pamětí. Tato schopnost řadiče přistupovat k operační paměti se označuje jako **přímý přístup do paměti** (DMA – direct memory access) a výrazně zvyšuje výkon počítače. Při čtení dat ze sektoru na disku může například procesor posílat bitové řetězce požadavků řadiči, který je připojen k disku. Procesor takto zajistí, aby řadič přečetl konkrétní sektor a umístil data do určené oblasti operační paměti. Procesor může poté pokračovat v jiné činnosti a řadič mezitím provádí operaci čtení a ukládá data do hlavní paměti s využitím přímého přístupu. Obě aktivity tedy mohou probíhat současně. Procesor spouští program a řadič dohlíží na přenos dat mezi diskem a operační pamětí. Díky tomu počítač neplýtvá kapacitou procesoru na relativně pomalý přenos dat.

Použití přímého přístupu do paměti však také komplikuje komunikaci, kterou přenáší sběrnice počítače. Bitové posloupnosti je nutné přepravovat mezi procesorem a operační pamětí, mezi procesorem a jednotlivými řadiči a mezi každým řadičem a operační pamětí. Koordinace všech těchto činností na sběrnici představuje zásadní konstrukční problém. Dokonce i v těch nejlépe navržených architekturách může centrální sběrnice snižovat výkon, když procesor soupeří o přístup ke sběrnici s řadiči. Tato výkonnostní brzda je známa jako **von Neumannův úzký profil**, protože se jedná o důsledek základní **von Neumannovy architektury**, ve které procesor načítá své instrukce z paměti prostřednictvím centrální sběrnice.



Obrázek 2.14: Koncepční reprezentace vstupu a výstupu zobrazeného do paměti

Handshaking

Přenos dat mezi dvěma počítačovými komponentami zpravidla neprobíhá jednosměrně. Tiskárnu si sice můžeme představovat jako zařízení, které přijímá data, ve skutečnosti ale tiskárna také odesílá data zpět do počítače. Počítač totiž dokáže generovat znaky a odesílat je do tiskárny mnohem rychleji, než je tiskárna dokáže vytisknout. Pokud by počítač trvale vysílal data do tiskárny vlastní rychlostí, tiskárna by mu rychle přestala stačit a skončilo by to ztrátou dat. Proces typu tisku dokumentu tedy vyžaduje neustálý dialog, který se označuje jako **proces handshaking** („potřásání rukama"). Počítač přitom od periferního zařízení dostává informace o jeho stavu a může s ním koordinovat svou činnost.

Proces handshaking často využívá **stavové slovo**, což je bitový vzor generovaný periferním zařízením a odesílaný řadiči. Stavové slovo je bitová mapa, jejíž bity odrážejí stav zařízení. V případě tiskárny může například hodnota nejméně významného bitu stavového slova sdělovat, že v tiskárně došel papír, zatímco další bit může znamenat, že tiskárna je připravena přijímat další data. Následující bit pak může třeba informovat o tom, že v tiskárně uvízl papír. V závislosti na systému pak řadič může na tyto stavové informace reagovat sám, nebo je zpřístupnit procesoru. V každém případě stavové slovo poskytuje mechanismus, který umožňuje koordinovat komunikaci s periferním zařízením.

Oblíbená komunikační média

Komunikace mezi počítačovými zařízeními probíhá dvěma cestami: paralelně a sériově. Tyto termíny se vztahují ke způsobu, kterým se přenášejí signály vzhledem k sobě navzájem. V případě **paralelní komunikace** se přenáší několik signálů současně – každý z nich po samostatné „lince". Tato metoda dokáže zajistit rychlý pře-

USB a FireWire

USB (universal serial bus) a FireWire jsou standardy sériových komunikačních systémů, které zjednodušují připojování nových periferních zařízení k osobnímu počítači. Vývoj sběrnice USB vedla společnost Intel. Při vývoji sběrnice FireWire měla hlavní slovo společnost Apple. Obě technologie vycházejí z požadavku, aby jediný řadič dokázal poskytnout externí porty pro připojení různých periferních zařízení. V této konfiguraci řadič překládá interní signály charakteristické pro počítač na příslušné standardní signály sběrnice USB nebo FireWire. Jednotlivá zařízení připojená k řadiči pak převádějí své interní specifické signály na stejný standard USB či FireWire, aby se mohly s řadičem dorozumět. Kvůli připojení nového zařízení k počítači tedy není nutné instalovat nový řadič. Stačí pouze připojit zařízení kompatibilní se standardem USB nebo FireWire k příslušnému portu.

Sběrnice FireWire sice poskytuje vyšší přenosovou rychlost než USB, ale díky nižším cenám zařízení se na spotřebitelském trhu více prosazuje technologie USB. V současnosti se prodávají například tato zařízení vyhovující standardu USB: myši, klávesnice, tiskárny, skenery, digitální fotoaparáty, smartphony a zařízení hromadného úložiště, která jsou určena k zálohování dat. Standard FireWire se uplatňuje spíše u zařízení, která vyžadují vysoké přenosové rychlosti, jako jsou videokamery a systémy hromadného úložiště online.

nos dat, ale vyžaduje relativně složité komunikační trasy. K příkladům patří interní sběrnice počítače, která dovoluje pomocí několika vodičů souběžně přenášet velké bloky dat a jiné signály.

Oproti tomu **sériová komunikace** je založena na následném přenosu signálů po jediné lince. K sériové komunikaci proto postačuje jednodušší datová trasa než v případě paralelní komunikace a právě proto je tak oblíbená. Jako příklady sériových komunikačních systémů, které nabízejí poměrně vysoké přenosové rychlosti na krátké vzdálenosti pouhých několika metrů, lze uvést standardy USB a FireWire. V kategorii poněkud delších vzdáleností (v rozsahu bytových nebo kancelářských domů) můžeme zmínit oblíbené připojení Ethernet (část 4.1), buď pomocí kabelu, nebo bezdrátově.

Co se týče komunikace na větší vzdálenosti, po mnoho let v oboru osobních počítačů dominovaly tradiční hlasové telefonní linky. Tyto komunikační trasy tvořené jedním vodičem, který přenáší jeden tón za druhým, patří ze své podstaty mezi sériové systémy. Přenos digitálních dat po těchto linkách probíhá tak, že nejdříve jsou bitové posloupnosti převedeny na slyšitelné tóny **modemem** (zkratkové slovo od *modulátor-demodulátor*), tyto tóny jsou sériově přeneseny pomocí telefonní sítě a následně zpětně konvertovány na bity pomocí jiného modemu v cílovém umístění.

Kvůli urychlení dálkové komunikace po klasických telefonních linkách nabízejí telefonní operátoři službu, která je známa pod zkratkou **DSL** (Digital Subscriber Line). Tato technologie využívá toho, že stávající telefonní linky dokáží přenášet širší frekvenční spektrum, než jaké se používá při tradiční hlasové komunikaci. Přesněji řečeno technologie DSL pracuje při přenosu digitálních dat s frekvencemi nad slyšitelným rozsahem a nižší frekvenční spektrum ponechává pro hlasové hovory. Technologie DSL je sice velmi úspěšná, ale telefonní společnosti rychle modernizují své systémy na optické linky, které se k digitální komunikaci hodí lépe než klasické telefonní kabely.

S technologiemi DSL a optických kabelů soupeří například operátoři kabelových televizních systémů a poskytovatelé satelitního připojení, které je založeno na vysokofrekvenčním rádiovém vysílání.

Komunikační rychlosti

Rychlost přenosu bitů mezi počítačovými komponentami se měří v **bitech za sekundu (b/s)**. Běžně se používají také jednotky **Kb/s** (kilobit za sekundu, tj. tisíc b/s), **Mb/s** (megabit za sekundu, tj. milion b/s) a **Gb/s** (gigabit za sekundu, tj. miliarda b/s). (Nepřehlédněme rozdíl mezi bity a bajty – 8 Kb/s se tedy rovná 1 KB za sekundu. Ve zkratkách malé „b" obvykle znamená *bit*, zatímco velké „B" označuje *bajt*.)

U komunikace na krátkou vzdálenost poskytují standardy USB a FireWire přenosové rychlosti několika set Mb/s, což pro většinu multimediálních aplikací vyhovuje. Díky této vlastnosti a zároveň snadnému používání a relativně nízké ceně příslušných zařízení se často uplatňují ke komunikaci mezi domácími počítači a místními periferiemi, jako jsou tiskárny, externí disky a fotoaparáty.

Pomocí kombinace **multiplexingu** (kódování či prokládání dat, aby jedna komunikační trasa dokázala fungovat jako několik tras) a metod komprese dat dosahovaly tradiční hlasové telefonní systémy přenosových rychlostí nejvýše 57,6 Kb/s, což nestačí nárokům dnešních multimediálních a internetových aplikací, jako je YouTube či Facebook. Přehrávání hudebních záznamů ve formátu MP3 vyžaduje přenosové rychlosti asi 64 Kb/s a k přehrávání videoklipů (i když v nízké kvalitě) jsou potřebné přenosové rychlosti, které se měří v Mb/s. Z tohoto důvodu musely tradiční hlasové telefonní systémy ustoupit alternativám, jako jsou technologie DSL, technologie kabelových televizí a satelitních spojů. (Technologie DSL například nabízí přenosové rychlosti do 54 Mb/s.)

Maximální dostupná rychlost v určité lokalitě závisí na komunikační trase a na konkrétní technologii, která se používá při její implementaci. Maximální rychlost se tedy volně přirovnává k **šířce pásma** (bandwidth) komunikační trasy, i když termín *šířka pásma* souvisí nejen s přenosovou rychlostí, ale i s kapacitou linky. Řekneme-li tedy, že komunikační trasa má velkou šířku pásma (či poskytuje **širokopásmovou** službu – broadband), máme na mysli, že komunikační trasa dokáže přenášet bity vysokou rychlostí a zároveň má kapacitu přenášet velký objem informací současně.

Otázky a cvičení

1. Předpokládejme, že počítač popsaný v Příloze C používá vstup a výstup zobrazený do paměti a adresa B5 označuje umístění v oblasti portu tiskárny, kam je nutné odesílat tisková data.
   a. Jestliže registr 7 obsahuje kód ASCII pro písmeno A, která instrukce strojového jazyka umožňuje vytisknout toto písmeno na tiskárně?
   b. Jestliže počítač provádí milion instrukcí za sekundu, kolikrát během jedné sekundy dokáže tento znak na tiskárnu odeslat?
   c. Umožňuje-li tiskárna vytisknout pět běžných stránek textu za minutu, dokáže držet krok s přenosem znaků podle bodu (b)?
2. Předpokládejme, že pevný disk osobního počítače se otáčí s rychlostí 3 000 otáček za minutu, každá stopa zahrnuje 16 sektorů a každý sektor obsahuje 1 024 bajtů. Jakou přibližnou rychlostí musí komunikovat disková

jednotka s řadičem disků, má-li řadič přijímat bity z diskové jednotky tak rychle, jak je rotující disk čte?

3. Odhadněte, jak dlouho trvá při přenosové rychlosti 54 Mb/s přenos románu s 300 stranami, který je uložen v kódování Unicode.

2.6: Jiné architektury

Abychom rozšířili své obzory, budeme se nyní zabývat alternativami k tradiční počítačové architektuře, kterou jsme zatím popsali.

Pipelining

Elektrické impulzy nemohou vodičem putovat rychleji než světlo. Vzhledem k tomu, že světlo urazí za nanosekundu (miliardtinu sekundy) přibližně 30 cm, potřebuje procesor na načtení instrukce z paměťové buňky vzdálené asi 30 cm přibližně 2 nanosekundy. (Požadavek čtení je nutné odeslat do paměti, což trvá minimálně 1 nanosekundu, a instrukci je potřeba přenést zpět do procesoru, což vyžaduje alespoň další nanosekundu.) Načtení a spuštění instrukce v takovém počítači proto vyžaduje několik nanosekund. To znamená, že zvyšování pracovní rychlosti počítačů se v důsledku mění na problém miniaturizace.

Výkon počítačů lze však zlepšit i jinými způsoby než zvýšením pracovní rychlosti. Skutečným cílem je zlepšit **propustnost** (throughput) počítače. Tento pojem označuje celkový objem práce, který počítač dokáže zvládnout v daném časovém úseku.

Jako příklad toho, jak lze zlepšit propustnost počítače, aniž by bylo nutné zvyšovat pracovní rychlost, lze uvést metodu **pipelining**. Tato technologie umožňuje, aby se jednotlivé kroky strojového cyklu překrývaly. Konkrétně to funguje tak, že v době, kdy počítač provádí jednu instrukci, už načítá další instrukci. V pracovním „kanálu“ (pipe) tedy může být v libovolný okamžik více instrukcí a každá z nich se přitom nachází v jiném stadiu zpracování. Díky tomu se zlepšuje celková propustnost počítače, i když čas potřebný k načtení a provedení každé jednotlivé instrukce zůstává stejný. (Při dosažení instrukce JUMP se samozřejmě žádný zisk díky předběžnému načítání neprojeví, protože místo instrukcí v „kanálu“ jsou nakonec potřeba jiné.)

Vícejádrové procesory

Jak technologie umožňuje směstnat na jeden křemíkový čip stále více elektronických obvodů, zmenšují se i fyzické bariéry mezi počítačovými komponentami. Jediný čip může například obsahovat procesor i operační paměť. Jedná se o příklad přístupu „systém na jediném čipu“, jehož cílem je zahrnout všechny komponenty do jediného zařízení, které lze použít jako abstraktní nástroj v návrzích vyšší úrovně. V jiných případech obsahuje jediné zařízení více kopií stejných obvodů. Tento přístup se původně objevil ve formě čipů, které nesly několik nezávislých hradel nebo více klopných obvodů. Současné nejmodernější technologie dovolují na jediný čip umístit více kompletních procesorů. Na této architektuře jsou založeny komponenty označované jako vícejádrové procesory, které na jednom čipu obsahují dva nebo více procesorů spolu se sdílenou mezipamětí. (Vícejádrové procesory, které se skládají ze dvou procesorových jednotek, se obvykle označují jako dvoujádrové procesory.) Tato zařízení zjednodušují stavbu systémů MIMD a v současnosti jsou lehce dostupná i pro domácí počítače.

Návrh moderních počítačů aplikuje koncepci pipeliningu nad rámec tohoto jednoduchého příkladu. Současné počítače často dokáží načítat několik instrukcí najednou a v praxi současně spouštějí více instrukcí, které na sobě navzájem nezávisí.

Víceprocesorové počítače

Technologii pipelining lze považovat za první krok směrem k **paralelnímu zpracování**, které spočívá v provádění několika činností zároveň. Skutečné paralelní zpracování však vyžaduje více procesorů. Vznikají tak počítače, které se označují jako víceprocesorové stroje.

Na tomto principu jsou v současnosti založeny různé počítače. Jedna strategie spočívá v připojení několika procesorů, z nichž každý připomíná procesor z jednoprocesorového počítače, ke stejné operační paměti. V této konfiguraci mohou procesory pracovat nezávisle, ale zároveň koordinovat svou činnost tak, že si předávají zprávy ve sdílených paměťových buňkách. Když například jeden procesor musí vyřešit složitou úlohu, může do společné paměti umístit program pro část dané úlohy a požádat jiný procesor, aby jej spustil. Výsledný počítač dokáže provádět různé sekvence instrukcí s různými sadami dat, což se označuje jako architektura **MIMD** s několika posloupnostmi instrukcí a více datovými proudy (multiple-instruction stream, multiple-data stream) architecture. Protikladem je tradičnější architektura **SISD** s jedinou posloupností instrukcí a datovým proudem (single-instruction stream, single-data stream).

Jiná varianta víceprocesorové architektury propojuje procesory navzájem, aby mohly provádět stejnou sekvenci instrukcí současně – každý z nich s vlastní sadou dat. Taková architektura s jednou posloupností instrukcí a více datovými proudy se nazývá **SIMD** (single-instruction stream, multiple-data stream). Uvedené počítače jsou vhodné tam, kde je potřeba stejným způsobem zpracovávat více sad podobných položek z velkého bloku dat.

Další přístup k paralelnímu zpracování spočívá v konstrukci velkých počítačů, které se skládají z mnoha menších počítačů, z nichž každý je vybaven vlastní pamětí a procesorem. V takové architektuře jsou jednotlivé malé počítače spárovány se svými sousedy, aby bylo možné jednotlivým počítačům přidělovat části úkolů, které má celý systém zpracovat. Jestliže lze tedy úkol přiřazený jednomu z interních počítačů rozdělit na nezávislé dílčí úkoly, může daný počítač požádat své sousedy, aby tyto části úlohy prováděly souběžně. Původní úkol je tak možné dokončit mnohem rychleji, než by to dokázal jednoprocesorový počítač.

Otázky a cvičení

1. Když se vrátíme k otázce 3 z části 2.3, co bude v „kanálu“ při provedení instrukce AA, jestliže bude počítač využívat metodu pipeliningu popsanou v textu? Za jakých podmínek by pipelining v této fázi programu neposkytl žádnou výhodu?
2. Které konflikty je nutné vyřešit při spouštění programu z otázky 4 části 2.3 v počítači s technologií pipelining?
3. Předpokládejme, že ke stejné paměti jsou připojeny dva „hlavní“ procesory, které spouštějí jiné programy. Dále předpokládejme, že jeden z těchto procesorů potřebuje přičíst jedničku k obsahu paměťové buňky přibližně

ve stejnou dobu, kdy jiný potřebuje od obsahu stejné buňky jedničku odečíst. (Výsledek by měl být takový, že buňka bude nakonec obsahovat stejnou hodnotu jako původně.)

a. Popište pořadí kroků, kdy tyto operace procesorů způsobí, že v buňce bude uložena hodnota o jedničku nižší než hodnota počáteční.

b. Popište pořadí kroků, kdy tyto operace procesorů způsobí, že v buňce bude uložena hodnota o jedničku vyšší než hodnota počáteční.

Úlohy na procvičování témat kapitoly

(Problémy označené hvězdičkou patří k volitelným částem kapitoly.)

1. **a.** V čem se podobají obecné registry procesoru a buňky operační paměti?

b. V čem se obecné registry procesoru a buňky operační paměti liší?

2. Odpovězte na následující otázky v terminologii strojového jazyka, který je popsán v Příloze C.

a. Napište instrukci 2304 (šestnáctkově) jako řetězec 16 bitů.

b. Napište operační kód instrukce B2A5 (šestnáctkově) jako řetězec 4 bitů.

c. Napište složku operandu instrukce B2A5 (šestnáctkově) jako řetězec 12 bitů.

3. Předpokládejme, že je v paměťových buňkách počítače popsaného v Příloze C uložen blok dat na adresách od 98 do A2 včetně. Kolik paměťových buněk tento blok obsahuje? Uveďte jejich adresy.

4. Jakou hodnotu bude mít programový čítač počítače popsaného v Příloze C ihned po provedení instrukce B0CD?

5. Předpokládejme, že paměťové buňky na adresách 00 až 05 operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
00	22
01	11
02	32
03	02
04	C0
05	00

Pokud programový čítač původně obsahoval hodnotu 00, zaznamenejte obsah programového čítače, instrukčního registru a paměťové buňky na adrese 02 na konci každé fáze načítání strojového cyklu až do zastavení počítače.

6. Předpokládejme, že jsou v paměti počítače uloženy hodnoty x, y a z. Popište pořadí událostí (načítání registrů z paměti, ukládání hodnot do paměti atd.), ke kterým dochází při výpočtu $x + y + z$. Jak je tomu v případě výpočtu $(2x) + y$?

7. Následující instrukce jsou zapsány ve strojovém jazyce, který je popsán v Příloze C. Vyjádřete je slovně.

a. 7123 **b.** 40E1 **c.** A304
d. B100 **e.** 2BCD

8. Předpokládejme, že návrh strojového jazyka používá čtyřbitovou složku operačního kódu. Kolik různých typů instrukcí může jazyk obsahovat? Jaký bude výsledek, pokud se složka operačního kódu prodlouží na 6 bitů?

9. Převeďte následující popisy instrukcí do strojového jazyka, který je popsán v Příloze C.

a. Načti (LOAD) do registru číslo 6 šestnáctkovou hodnotu 77.

b. Načti (LOAD) do registru číslo 7 obsah paměťové buňky 77.

c. Přeskoč (JUMP) na instrukci v paměťovém umístění 24, pokud se obsah registru 0 rovná hodnotě obsažené v registru A.

d. Posuň (ROTATE) obsah registru číslo 4 o tři bity doprava.

e. Proveď operaci AND s obsahem registru E a registru 2 a ulož výsledek do registru 1.

10. Přepište program na obrázku 2.7 za předpokladu, že sčítané hodnoty nejsou zakódovány v zápisu dvojkového doplňku, ale pomocí zápisu s plovoucí řádovou čárkou.

11. U každé z následujících instrukcí (ve strojovém jazyce z Přílohy C) určete, zda její spuštění změní obsah paměťové buňky v umístění 3B, načte obsah paměťové buňky v umístění 3C nebo na obsahu paměťové buňky v umístění 3C nezávisí.

a. 353C **b.** 253C **c.** 153C
d. 3C3C **e.** 403C

12. Předpokládejme, že paměťové buňky na adresách 00 až 03 operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
00	26
01	55
02	C0
03	00

a. Vyjádřete první instrukci slovně.

b. Spustíme-li počítač, jehož programový čítač obsahuje hodnotu 00, který bitový vzor se bude po zastavení počítače nacházet v registru 6?

13. Předpokládejme, že paměťové buňky na adresách 00 až 02 operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
00	12
01	21
02	34

a. Která instrukce bude provedena jako první, spustíme-li počítač, jehož programový čítač obsahuje hodnotu 00?

b. Která instrukce bude provedena jako první, spustíme-li počítač, jehož programový čítač obsahuje hodnotu 01?

14. Předpokládejme, že paměťové buňky na adresách 00 až 05 operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
00	12
01	02
02	32
03	42
04	C0
05	00

Při hledání odpovědí na následující otázky předpokládejte, že při spuštění počítače obsahuje jeho programový čítač hodnotu 00.

a. Vyjádřete prováděné instrukce slovně.

b. Který bitový vzor se bude po zastavení počítače nacházet v paměťové buňce na adrese 42?

c. Který bitový vzor se bude po zastavení počítače nacházet v programovém čítači?

15. Předpokládejme, že paměťové buňky na adresách 00 až 09 operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
00	1C
01	03
02	2B
03	03
04	5A
05	BC
06	3A
07	00
08	C0
09	00

Předpokládejte, že při spuštění počítače obsahuje jeho programový čítač hodnotu 00.

a. Co se bude po zastavení počítače nacházet v paměťové buňce na adrese 00?

b. Který bitový vzor se bude po zastavení počítače nacházet v programovém čítači?

16. Předpokládejme, že paměťové buňky na adresách 00 až 07 operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
00	2B
01	07
02	3B
03	06
04	C0
05	00
06	00
07	23

a. Uveďte adresy paměťových buněk, které obsahují spouštěný program, spustíme-li počítač, jehož programový čítač obsahuje hodnotu 00.

b. Uveďte adresy paměťových buněk, které uchovávají data.

17. Předpokládejme, že paměťové buňky na adresách 00 až 0D operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
00	20
01	04
02	21
03	01
04	40
05	12
06	51
07	12
08	B1
09	0C
0A	B0
0B	06
0C	C0
0D	00

Předpokládejte, že při spuštění počítače obsahuje jeho programový čítač hodnotu 00.

a. Který bitový vzor se bude po zastavení počítače nacházet v registru 0?

b. Který bitový vzor se bude po zastavení počítače nacházet v registru 1?

c. Který bitový vzor se bude po zastavení počítače nacházet v programovém čítači?

18. Předpokládejme, že paměťové buňky na adresách F0 až FD operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
F0	20
F1	00
F2	22
F3	02
F4	23
F5	04
F6	B3
F7	FC
F8	50
F9	02
FA	B0
FB	F6
FC	C0
FD	00

Spustíme-li počítač, jehož programový čítač obsahuje hodnotu F0, jaká bude hodnota v registru 0 poté, co počítač nakonec provede instrukci stop na adrese FC?

19. Jestliže počítač v Příloze C provádí jednu instrukci každou mikrosekundu (miliontinu sekundy), za jak dlouho bude dokončen program z otázky 18?

20. Předpokládejme, že paměťové buňky na adresách 20 až 28 operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
20	12
21	20
22	32
23	30
24	B0
25	21
26	24
27	C0
28	00

Předpokládejte, že při spuštění počítače obsahuje jeho programový čítač hodnotu 20.

a. Které bitové vzory se budou po zastavení počítače nacházet v registrech 0, 1 a 2?

b. Který bitový vzor se bude po zastavení počítače nacházet v paměťové buňce na adrese 30?

c. Který bitový vzor se bude po zastavení počítače nacházet v paměťové buňce na adrese B0?

21. Předpokládejme, že paměťové buňky na adresách AF až B1 operační paměti počítače popsaného v Příloze C obsahují následující bitové vzory:

Adresa	Obsah
AF	B0
B0	B0
B1	AF

Co se stane, spustíme-li počítač, jehož programový čítač obsahuje hodnotu AF?

22. Předpokládejme, že paměťové buňky na adresách 00 až 05 operační paměti počítače popsaného v Příloze C obsahují následující (šestnáctkové) bitové vzory:

Adresa	Obsah
00	25
01	B0
02	35
03	04
04	C0
05	00

Spustíme-li počítač, jehož programový čítač obsahuje hodnotu 00, kdy se počítač zastaví?

23. V každém z následujících případů napište krátký program ve strojovém jazyce popsaném v Příloze C, který provede požadované činnosti. Předpokládejte, že všechny programy jsou v paměti umístěny od adresy 00.

- **a.** Přesuňte hodnotu z paměťového umístění D8 do paměťového umístění B3.
- **b.** Zaměňte hodnoty uložené v paměťových umístěních D8 a B3.
- **c.** Pokud se hodnota uložená v paměťovém umístění 44 rovná 00, umístěte hodnotu 01 do paměťového umístění 46. Jinak uložte hodnotu FF do paměťového umístění 46.

24. Mezi počítačovými nadšenci byla kdysi v oblibě hra zvaná „core wars", která připomínala známou hru „lodě". (Termín *core* pochází z raných dob paměťové technologie, kdy se jedničky a nuly zaznamenávaly jako magnetická pole v malých kroužcích magnetického materiálu. Těmto prstencům se říkalo „core".) Hry se účastní dva soupeřící programy, které jsou uloženy v jiných umístěních paměti stejného počítače. Počítač mezi oběma programy přepíná a po provedení instrukce jednoho programu provede jednu instrukci druhého. Cílem každého programu je způsobit selhání protivníka tím, že přes něj zapíše jiná data. Ani jeden z programů však nezná pozici toho druhého.

- **a.** Napište ve strojovém jazyce z Přílohy C program, který bude využívat defenzivní herní strategii a bude co nejmenší.
- **b.** Napište ve strojovém jazyce z Přílohy C program, který se bude snažit vyhnout všem útokům soupeřícího programu tím, že se bude přesunovat do jiných umístění. Přesněji řečeno napište program, který se z původního umístění 00 zkopíruje do umístění 70 a poté přeskočí na umístění 70.
- **c.** Rozšiřte program z bodu (b) tak, aby pokračoval v přesunech do nových paměťových umístění. Konkrétně program přizpůsobte tak, aby se přesunul do umístění 70, poté do umístění E0 (= 70 + 70), následně do umístění 60 (= 70 + 70 + 70) atd.

25. Napište ve strojovém jazyce z Přílohy C program, který bude počítat součet hodnot v zápisu s plovoucí řádovou čárkou uložených v paměťových umístěních A0, A1, A2 a A3. Program by měl uložit součet do paměťového umístění A4.

26. Předpokládejme, že paměťové buňky na adresách 00 až 05 operační paměti počítače popsaného v Příloze C obsahují následující (šestnáctkové) bitové vzory:

Adresa	Obsah
00	20
01	C0
02	30
03	04
04	00
05	00

Co se stane, spustíme-li počítač, jehož programový čítač obsahuje hodnotu 00?

27. Co se stane, pokud paměťové buňky na adresách 08 a 09 počítače popsaného v Příloze C obsahují bitové vzory B0 a 08 a programový čítač obsahuje při spuštění počítače hodnotu 08?

28. Předpokládejme, že následující program zapsaný ve strojovém jazyce z Přílohy C je uložen v operační paměti počínaje adresou 30 (šestnáctkově). Jakou funkci bude plnit spuštěný program?

```
2003
2101
2200
2310
1400
3410
5221
5331
3239
333B
B248
B038
C000
```

29. Shrňte kroky, které provádí počítač popsaný v Příloze C, když spustí instrukci s operačním kódem B. Vyjádřete svou odpověď v posloupnosti pokynů, jako byste instruovali procesor, co má dělat.

***30.** Shrňte kroky, které provádí počítač popsaný v Příloze C, když spustí instrukci s operačním kódem 5. Vyjádřete svou odpověď v posloupnosti pokynů, jako byste instruovali procesor, co má dělat.

***31.** Shrňte kroky, které provádí počítač popsaný v Příloze C, když spustí instrukci s operačním kódem 6. Vyjádřete svou odpověď v posloupnosti pokynů, jako byste instruovali procesor, co má dělat.

***32.** Předpokládejme, že registry 4 a 5 počítače popsaného v Příloze C obsahují bitové vzory 3A a C8. Který bitový vzor zůstane v registru 0 po provedení každé z následujících instrukcí:

a. 5045 **b.** 6045 **c.** 7045
d. 8045 **e.** 9045

***33.** Pomocí strojového jazyka popsaného v Příloze C napište programy, které budou provádět následující úkoly:

a. Zkopírujte bitový vzor uložený v paměťovém umístění 44 do paměťového umístění AA.

b. Změňte čtyři nejméně významné bity v paměťové buňce v umístění 34 na nuly a ostatní bity ponechejte beze změny.

c. Zkopírujte čtyři nejméně významné bity z paměťového umístění A5 do čtyřech nejméně významných bitů v umístění A6, ale zbývající bity v umístění A6 ponechejte beze změny.

d. Zkopírujte čtyři nejméně významné bity z paměťového umístění A5 do čtyřech nejvýznamnějších bitů v umístění A5. (První čtyři bity v umístění A5 budou tedy stejné jako poslední 4 bity.)

***34.** Proveďte uvedené operace:

a.
```
  111001
AND 101001
```
b.
```
  000101
AND 101010
```
c.
```
  001110
AND 010101
```
d.
```
  111011
AND 110111
```
e.
```
  111001
OR 101001
```
f.
```
  010100
OR 101010
```
g.
```
  000100
OR 010101
```
h.
```
  101010
OR 110101
```
i.
```
  111001
XOR 101001
```
j.
```
  000111
XOR 101010
```
k.
```
  010000
XOR 010101
```
l.
```
  111111
XOR 110101
```

***35.** Identifikujte masku i logickou operaci, které jsou potřebné k dosažení každého z následujících cílů:

a. Umístěte jedničky do horních 4 bitů osmibitového vzoru, aniž byste narušili zbývající bity.

b. Změňte nejvýznamnější bit osmibitového vzoru na jeho doplněk, aniž byste změnili zbývající bity.

c. Vytvořte doplněk vzoru s 8 bity.

d. Umístěte nulu do nejméně významného bitu osmibitového vzoru, aniž byste narušili zbývající bity.

e. Umístěte jedničky do všech bitů osmibitového vzoru kromě nejvýznamnějšího bitu, aniž byste změnili hodnotu tohoto bitu.

***36.** Identifikujte logickou operaci (spolu s odpovídající maskou), která při aplikaci na osmibitový vstupní řetězec poskytne výstupní řetězec se samými nulami tehdy a právě tehdy, když se vstupní řetězec rovná 0000001.

***37.** Popište pořadí logických operací (spolu s odpovídajícími maskami), které při aplikaci na osmibitový vstupní řetězec poskytnou na výstupu bajt se samými nulami, když vstupní řetězec začíná a končí jedničkami. Jinak by měl výstup obsahovat alespoň jednu jedničku.

***38.** Jaký je výsledek provedení levého kruhového posunu o 4 bity u následujících bitových vzorů?

a. 10101 **b.** 11110000 **c.** 001

d. 101000 **e.** 00001

***39.** Jaký je výsledek provedení pravého kruhového posunu o 2 bity u následujících bajtů, které jsou vyjádřeny v šestnáctkovém zápisu (svou odpověď uveďte rovněž v šestnáctkovém zápisu)?

a. 3F **b.** 0D

c. FF **d.** 77

***40.** **a.** Jakou jedinou instrukcí strojového jazyka z Přílohy C lze zajistit pravý kruhový posun obsahu registru B o 5 bitů?

b. Jakou jedinou instrukcí strojového jazyka z Přílohy C lze zajistit levý kruhový posun obsahu registru B o 2 bity?

***41.** Napište ve strojovém jazyce z Přílohy C program, který stranově obrátí obsah paměťové buňky na adrese 8C. (To znamená, že výsledný bitový vzor na adrese 8C se bude při čtení zleva doprava shodovat z původním vzorem čteným zprava doleva.)

***42.** Napište ve strojovém jazyce z Přílohy C program, který odečte hodnotu uloženou v umístění A1 od hodnoty uložené na adrese A2 a umístí výsledek na adresu A0. Předpokládejte, že hodnoty jsou zakódovány v zápisu dvojkového doplňku.

***43.** Video s vysokým rozlišením může mít frekvenci 30 snímků za sekundu, kde každý snímek má rozlišení 1920 × 1080 pixelů a využívá 24 bitů na pixel. Lze nekomprimovaný datový proud videa v tomto formátu přenášet po sériovém portu standardu USB 1.1? Jak je tomu u sériového portu standardu USB 2.0? Jak je tomu u sériového portu standardu USB 3.0? (Poznámka: Sériové porty standardu USB 1.1, USB 2.0 a USB 3.0 poskytují maximální rychlosti postupně 12 Mb/s, 480 Mb/s a 5 Gb/s.)

***44.** Předpokládejme, že písař dokáže na klávesnici psát čtyřicet slov za minutu. (Slovo má v průměru pět znaků.) Jestliže počítač provádí 500 instrukcí každou mikrosekundu (miliontinu sekundy), kolik instrukcí počítač spustí v době mezi dvěma následnými stisknutími kláves?

***45.** Kolik bitů za sekundu musí klávesnice přenášet, aby stačila písaři, který píše čtyřicet slov za minutu? (Předpokládejme, že každý znak je vyjádřen v kódování ASCII a každé slovo obsahuje šest znaků.)

***46.** Předpokládejme, že počítač popsaný v Příloze C komunikuje s tiskárnou metodou vstupu a výstupu zobrazeného do paměti. Předpokládejme dále, že adresa FF slouží k odesílání znaků do tiskárny a adresa FE je určena pro příjem informací o stavu tiskárny. Konkrétně předpokládejme, že nejméně významný bit na adrese FE určuje, zda je tiskárna připravena přijmout další znak (kdy 0 znamená „nepřipraveno“ a 1 „připraveno“). Napište rutinu ve strojovém jazyce, která začíná na adrese 00. Tato rutina bude vyčkávat, až bude tiskárna připravena přijmout další znak, a poté tiskárně odešle znak, který je určen bitovým vzorem v registru 5.

***47.** Napište ve strojovém jazyce z Přílohy C program, který umístí nuly do všech paměťových buněk od adresy A0 do C0, ale zároveň je dostatečně malý, aby se vešel do paměťových buněk od adresy 00 do 13 (šestnáctkově).

***48.** Předpokládejme, že počítač má 200 GB dostupného volného místa na pevném disku a přijímá data pomocí širokopásmového připojení s rychlostí 15 Mb/s. Za jak dlouho se při této rychlosti dostupné úložné místo zaplní?

***49.** Předpokládejme, že se používá satelitní systém, který přijímá sériový datový proud o rychlosti 250 Kb/s. Pokud po dobu 6,96 sekundy dochází k atmosférickému rušení, kolik datových bitů to ovlivní?

***50.** Předpokládejme, že máte k dispozici 32 procesorů a každý z nich dokáže zjistit součet dvou vícemístných čísel za miliontinu sekundy. Popište, jak lze pomocí metod paralelního zpracování najít součet 64 čísel za pouhých šest miliontin sekundy. Za jak dlouho spočítá stejný součet jediný procesor?

***51.** Shrňte rozdíly mezi architekturami CISC a RISC.

***52.** Uveďte dva přístupy, které umožňují zvýšit propustnost.

***53.** Popište, jak lze pomocí víceprocesorového počítače určit průměr sady čísel rychleji než pomocí jednoprocesorového počítače.

Společenské otázky

Následující otázky mají čtenáře provést etickými, společenskými a právními aspekty oboru počítačů. Cílem není jen odpovědět na jednotlivé otázky. Měli byste se také zamyslet nad tím, proč jste vybrali konkrétní odpověď a zda dokážete odpovědi na jednotlivé otázky konzistentně zdůvodnit.

1. Předpokládejme, že výrobce počítačů vyvine novou architekturu. Do jaké míry by měla mít tato společnost právo tuto architekturu vlastnit? Jaké zásady by byly pro společnost nejlepší?
2. Roku 1923 se v určitém smyslu zrodila koncepce, která se nyní často označuje jako *plánované zastarávání* (planned obsolescence). V tomto roce společnost General Motors pod vedením Alfreda Sloana zavedla do automobilového průmyslu princip modelových let. Cílem bylo zvýšit prodej nikoli díky lepšímu automobilu, ale pouhou změnou stylu. Cituje se Sloanův výrok: „Chceme, abyste se svým současným autem přestali být spokojeni a koupili si proto nové“. V jakém rozsahu se tento marketingový trik používá v současném počítačovém průmyslu?
3. Často uvažujeme o tom, jak počítačová technologie změnila naši společnost. Mnozí však tvrdí, že tato technologie často zabraňovala nástupu změn, protože umožnila starým systémům přetrvat a v některých případech dokonce jejich pozici upevnila. Mohla by například bez počítačové technologie přežít společenská role ústřední vlády? Do jaké míry by mohla v současnosti existovat centralizovaná správa, kdyby nebyla počítačová technologie k dispozici? Nakolik bychom na tom byli bez počítačové technologie lépe nebo hůře?
4. Je etické, aby se jednotlivci rozhodli, že nepotřebují nic vědět o vnitřních podrobnostech stroje, který používají, protože jej konstruuje, udržuje v provozu a opravuje někdo jiný? Závisí vaše odpověď na tom, zda se jedná o počítač, automobil, atomovou elektrárnu nebo opékač topinek?
5. Předpokládejme, že výrobce vyprodukuje počítačový čip a později v jeho návrhu najde chybu. Dále předpokládejme, že výrobce chybu opraví v nově vyráběných čipech, ale rozhodne se původní vadu utajit a již vyrobené čipy nestáhne, protože žádný z nich se nepoužívá na místech, kde by tato chyba měla vážné následky. Poškodí výrobce někoho svým rozhodnutím? Je rozhodnutí výrobce oprávněné, když nikdo nedošel újmy a zároveň díky tomuto postupu výrobce neztratí příjmy a nemusí tedy propouštět zaměstnance?

6. Přináší technologické pokroky léčbu srdečních nemocí, nebo vedou k sedavému životnímu stylu, který je jednou z příčin těchto nemocí?
7. Snadno si můžeme představit finanční nebo navigační nehody, které by mohly nastat kvůli aritmetickým chybám v důsledku přetečení nebo odříznutí cifer při výpočtu. Jaké následky by mohly mít chyby v systémech ukládání obrázků způsobené ztrátou detailů na obrázcích (řekněme v oblastech jako je letecký průzkum nebo lékařská diagnostika)?
8. ARM Holdings je malá společnost, která navrhuje procesory pro mnoho různých výrobků spotřební elektroniky. Sama žádné procesory nevyrábí, ale poskytuje licence jiným společnostem s továrnami na zpracování polovodičů (jako je Qualcomm, Samsung a Texas Instruments), které za každý vyrobený kus hradí licenční poplatky. Tento obchodní model rozptyluje vysoké náklady na výzkum a vývoj počítačových procesorů na celý trh spotřební elektroniky. V současnosti více než 95 % všech mobilních telefonů (nejen smartphony), přes 40 % všech digitálních fotoaparátů a 25 % digitálních televizorů obsahuje procesor ARM. Procesory ARM se také nacházejí v miniaturních noteboocích, MP3 přehrávačích, herních konzolách, čtečkách elektronických knih, navigačních systémech a seznam by mohl pokračovat. Lze s ohledem na tyto informace společnost ARM považovat za monopol? Vysvětlete svou odpověď. Jak spotřební zařízení získávají v dnešním světě stále větší význam, je závislost na této málo známé firmě dobrá, nebo vzbuzuje obavy?

Další zdroje informací

Carpinelli, J. D. *Computer Systems Organization and Architecture* (Organizace a architektura počítačových systémů). Boston, MA: Addison-Wesley, 2001.

Comer, D. E. *Essentials of Computer Architecture* (Základy počítačové architektury). Upper Saddle River, NJ: Prentice-Hall, 2005.

Dandamudi, S P. *Guide to RISC Processors for Programmers and Engineers* (Příručka procesorů RISC pro programátory a techniky). New York: Springer, 2005.

Furber, S. *ARM System-on-Chip Architecture* (Architektura systémů ARM na jednom čipu), 2. vyd. Boston, MA: Addison-Wesley, 2000.

Hamacher, V. C., Z. G. Vranesic a S. G. Zaky. *Computer Organization* (Organizace počítačů), 5. vyd. New York: McGraw-Hill, 2002.

Knuth, D. E. *Umění programování*, *1. díl. Základní algoritmy*. Brno: Computer Press 2008.

Murdocca, M. J. a V. P. Heuring. *Computer Architecture and Organization: An Integrated Approach.* (Architektura a organizace počítačů: integrovaný přístup), New York: Wiley, 2007.

Stallings, W. *Computer Organization and Architecture* (Organizace a architektura počítačů), 7. vyd. Upper Saddle River, NJ: Prentice-Hall, 2006.

Tanenbaum, A. S. *Structured Computer Organization* (Struktura a organizace počítačů), 5. vyd. Upper Saddle River, NJ: Prentice-Hall, 2006.