Jiří Hořejš

Principy samočinných číslicových počítačů

Pokroky matematiky, fyziky a astronomie, Vol. 7 (1962), No. 1, 15--33

Persistent URL: http://dml.cz/dmlcz/139814

PRINCIPY SAMOČINNÝCH ČÍSLICOVÝCH POČÍTAČŮ

JIŘÍ HOŘEJŠ, Brno

V tomto článku jsou vysvětleny zásadní možnosti těchto strojů a podán charakter práce matematika s nimi.

§ 1

Samočinný číslicový počítač je zařízení, které vykonává tyto funkce:

I. Zapíše si na místa očíslovaná (prvými N) přirozenými čísly a tím uspořádaná některá reálná čísla a tak je po jistou dobu uchovává a má k dispozici. Vzhledem k této vlastnosti se příslušná část počítače nazývá *paměť*, příslušná místa přesněji *paměťová místa;* na každém paměťovém místě je *zapsáno* (*uloženo*) jedno číslo. Přirozená čísla, jimiž jsou paměťová místa očíslována, jsou jejich *adresy*; N udává *kapacitu* paměti.

Ke každému paměťovému místu přísluší tedy dvě čísla: jeho adresa, udávající jeho pořadí v paměti, a číslo, které je na něm právě zapsáno. Často budeme označovat obě tato čísla stejným písmenem. Bude-li však třeba zdůraznit rozdíl mezi nimi, užijeme této symboliky: číslo zapsané na paměťovém místě o adrese r označíme $<r>$, zatímco adresa libovolného paměťového místa, na kterém je zapsáno číslo x, bude označena $>x<$. Užijeme-li posledního symbolu, budeme vždy předpokládat, že x je skutečně na některém paměťovém místě zapsáno.

II. Provádí jeden z několika (pro daný počítač předem pevně daných) *rozkazů* neboli *instrukcí* (budeme je označovat malými řeckými písmeny), mezi nimiž téměř vždy bývají:

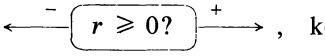
A. Sečti (odečti, znásob, vyděl) čísla, která jsou zapsána na paměťových místech o adresách r a s a výsledek zapiš na paměťové místo o adrese t (bez ohledu na to, co tam bylo původně zapsáno, takže původní číslo se při tom automaticky vymaže; na paměťových místech o adresách r, s zůstanou čísla beze změny). Potom proveď instrukci ξ (*přejdi* k instrukci ξ). Symbolicky celý rozkaz zapíšeme pomocí (*instrukčního*) *bloku*:

$$\boxed{r + s \to t} \longrightarrow \left(\boxed{r - s \to t} \longrightarrow, \quad \boxed{r \,.\, s \to t} \longrightarrow, \quad \boxed{r : s \to t} \longrightarrow \right);$$

při tom šipka končí na bloku, který analogicky zapisuje instrukci ξ.

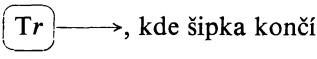
B. Zjisti, zda číslo zapsané na paměťovém místě o adrese r je nezáporné; jestliže

ano, přejdi k instrukci ξ; jestliže ne, k instrukci η. Symbolicky:



, kde šipka $\xrightarrow{+}$ končí na bloku, který zapisuje instrukci ξ; šipka $\xrightarrow{-}$ na bloku, který zapisuje instrukci η.

C. Umožni odečtení (obvykle: vytiskni) čísla zapsaného na paměťovém místě o adrese r a přejdi k instrukci ξ. Symbolicky:



, kde šipka končí na bloku, který zapisuje instrukci ξ.

D. Zastav výpočet. Symbolicky:

Instrukce typu A se někdy nazývají *aritmetickými*, typu B instrukcemi *podmíněného přechodu*, typu C instrukcemi *tisku* a typu D instrukcemi *zastavení*. V dalším budeme někdy jednotlivé instrukce symbolicky zapisovat z typografických důvodů „bloky" bez kontur.

Předpokládejme nyní, že je dán počítač, který je schopen provádět právě to, co je uvedeno v **I** a **II**; budeme o něm mluvit jako o *zjednodušeném počítači.*

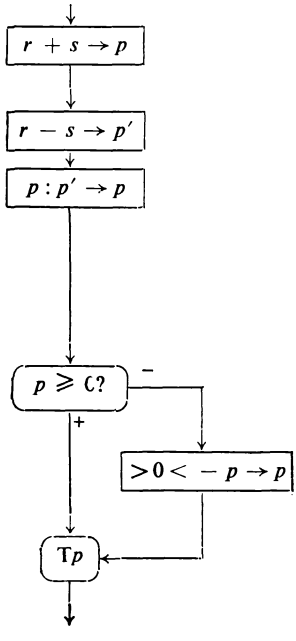
Skutečné počítače jsou obvykle schopny realizovat přímo i další instrukce: provádět aritmetické instrukce nejen s čísly zapsanými na jistých paměťových místech, ale i s jejich absolutními hodnotami (např. $|r| + |s| \to t$), přenášet čísla z jednoho paměťového místa do druhého ($r \to s$), porovnávat dvě libovolná čísla ($r \geqslant s$? či $r = s$?) ap., tedy provádět přímo i takové rozkazy, které uvažovaný zjednodušený počítač může vykonat teprve postupným vyplněním řady jednodušších instrukcí. Na druhé straně některé počítače nejsou schopny přímo provést ani uvedené instrukce; vždy je však obráceně mohou splnit postupným provedením ještě jednodušších (srv. § 4). Existují i další rozdíly (s některými z nich se v dalším seznámíme), žádný z nich však nerozšiřuje ani neredukuje podstatně možnosti počítače ani podstatně neovlivňuje charakter práce s ním (práci samu ovšem ano). V každém případě je užitečné a v zájmu obecnosti i nutné odhlédnout nejprve od speciálních vlastností konkrétních počítačů. Poznamenejme, že některé idealizované zjednodušené počítače dostaly v zahraniční literatuře již i jména, např. UVM (universalnaja vyčislitelnaja mašina) — [1], tom 1, TYDAC (Typical Digital Automatic Computer) — [3].

Od počítače ovšem nikdy nepožadujeme provedení jediné instrukce. Chceme-li symbolicky naznačit, které instrukce mají být při řešení jisté úlohy postupně prováděny, propojíme šipkami bloky zapisující tyto instrukce tak, jak bylo uvedeno v **II**, a vyznačíme blok, který obsahuje prvou (v pořadí výpočtu) instrukci. Při tom instrukcemi typu **B** můžeme *větvit* výpočet, tj. můžeme zachytit různé eventuality dalšího postupu; kterou z nich vybrat, rozhodne stroj v závislosti na hodnotách mezivýsledků. Takto skloubená soustava instrukčních bloků, v níž je symbolicky zachycen postup, tzv. *algoritmus* řešení dané úlohy, se nazývá *instrukční síť.*

Aby se počítač mohl řídit algoritmem zachyceným v dané instrukční síti, tj. aby mohl uskutečnit postupně její jednotlivé instrukce, musí ovšem být celá instrukční síť jistým způsobem vložena do počítače. V následujícím paragrafu poznáme, jak je to možno uskutečnit, aniž bychom na uvažovaný počítač kladli další požadavky podstatně rozdílné od těch, jež byly formulovány v **I** a **II.**

Příklad 1. Nechť v paměti počítače jsou již uložena čísla r, s, $r \neq s$; adresy paměťových míst, na nichž jsou tato čísla zapsána, označme po řadě opět r, s. Chceme, aby počítač určil a vytiskl hodnotu výrazu $|(r + s)/(r - s)|$. Sestavíme instrukční síť zachycující jeden z možných způsobů řešení.

Ani tuto úlohu bychom neřešili na samočinném počítači samotnou; budeme proto předpokládat, že jde pouze o sestavení části nějaké (podstatně) složitější instrukční sítě, což naznačíme užitím dvou šipek: u prvního bloku sítě šipkou bez určeného začátku, u posledního bez určeného konce. Podobně v dalších příkladech.



... instrukce typu **A**: součet čísel r a s zapíše počítač na pomocné paměťové místo o adrese p (v dalším jen *paměťové místo p* a někdy přímo jen p);

... instrukce typu **A**: rozdíl čísel r a s zapíše počítač na další pomocné paměťové místo p';

... instrukce typu **A**: podíl čísel zapsaných v p a p' (tj. čísel $r + s$ a $r - s$) zapíše počítač na pomocné paměťové místo, kterým může být opět paměťové místo p, neboť číslo až dosud na něm zapsané nebudeme v dalším již potřebovat; je dobré neplýtvat paměťovými místy, kterých je vždy jen konečný počet;

... instrukce typu **B**: počítač zjistí, zda číslo zapsané v p (tj. číslo $(r + s) : (r - s)$) je nezáporné; v závislosti na výsledku větví výpočet;

... instrukce typu **A**: v případě, že číslo v p je záporné, zapíše počítač na toto paměťové místo číslo opačné;

... instrukce typu **C**: počítač tiskne číslo, zapsané na paměťovém místě p; tímto číslem je v každém případě již hledané číslo.

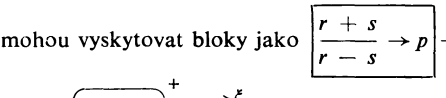
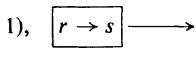
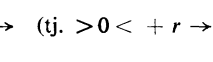
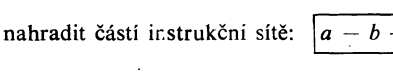
Z toho, co bylo zatím řečeno, lze již usoudit na některé rysy práce se samočinným počítačem:

1. Instrukční síť pro řešení dané úlohy musí vypracovat matematik; samočinnost počítače tedy nespočívá v tom, že by stroj řešil samostatně úlohy ihned po jejich zadání běžnými způsoby, nýbrž v tom, že je schopen na základě vložené instrukční sítě provádět výpočet a jednoduchá rozhodování o postupu výpočtu bez další účasti člověka.

2. Matematik musí rozpracovat řešení až do nejjednodušších aritmetických úkonů (některých výjimek si všimneme později); každá úloha se tedy řeší numericky. Vhodnou numerickou metodu opět musí určit matematik. Použití počítače tedy nečiní práci matematika zbytečnou nebo zcela mechanickou, spíše naopak; z jeho práce jsou odstraněny mechanické početní úkony, kdežto analýza problému musí být podrobnější než při řešení některými běžnými způsoby; předem je nutno uvážit všechny eventuality, které se během výpočtu mohou vyskytnout.

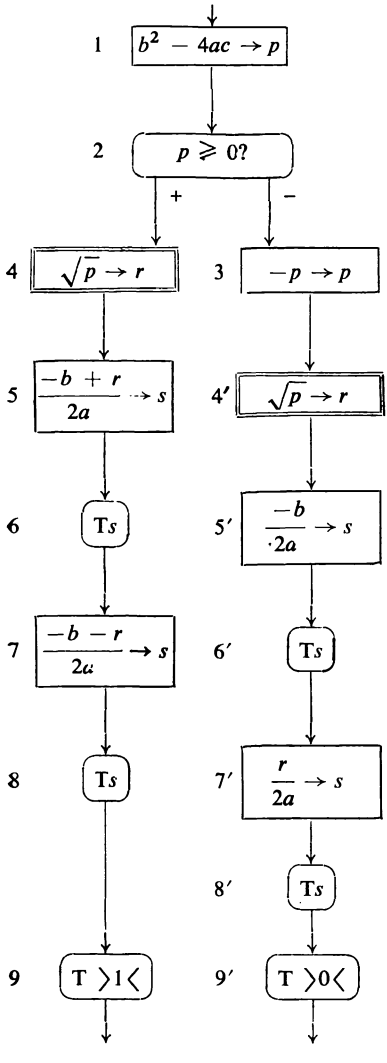
3. V jednoduchosti instrukcí číslicového počítače je jistá nevýhoda: složitější matematické úkony (např. integrování) vyžadují provedení mnoha instrukcí (na rozdíl od některých jiných druhů matematických strojů, které jsou schopny přímo provádět složitější úkony, např. zmíněné integrování v případě tzv. diferenciálních analyzátorů). Avšak v této jednoduchosti je zároveň i velká výhoda číslicového počítače (uvažovaného typu): řešení mnoha úloh požadovaných praxí lze převést na postupné provádění takových jednoduchých instrukcí; říkáme, že číslicový počítač je *univerzální*. Při tom ho lze užít nejen k řešení úloh zřejmého matematického charakteru (vědeckotechnické výpočty, výpočty mezd ap.), ale i mnohých jiných. Jeho schopností pamatovat si velké množství číselného materiálu a rychle porovnávat i rozhodovat se využívá v plánování, v evidenci i v řízení výroby. Dokonce i tam, kde se vyskytují údaje, které nemají charakter čísel, může podstatnou část práce zastat univerzální číslicový počítač. Stačí pouze tyto údaje vhodně *zakódovat*, tj. přiřadit jim jistým způsobem čísla, resp.cifry (srv. se speciálním případem kódování, které je popsáno v následujícím paragrafu), a zpracovat je v tomto tvaru. Tohoto obratu se používá např. při strojovém překladu z jednoho jazyka do druhého, při užití počítačů v diagnostice ap. Vzhledem k velmi širokým možnostem kódování někdy říkáme obecně, že na paměťových místech jsou zapsána *slova*.

Sestavit instrukční síť pro přímé řešení složitějšího problému (třeba soustavy diferenciálních rovnic) by mnohdy bylo prakticky nemožné pro obrovský počet instrukcí a složitost vztahů mezi nimi. Proto se většinou nesestavuje nejdřív instrukční síť sama, nýbrž její *schéma*; ve schématu užíváme stejným způsobem kontur bloků a šipek, avšak uvnitř bloků je dovoleno symbolicky zapsat libovolný požadovaný úkon takový, který je možno splnit postupným provedením skupiny instrukcí. V každém případě musí být jasno, jak ze schématu získáme přímo instrukční síť.

Příklad 2. Ve schématu instrukční sítě se mohou vyskytovat bloky jako  (viz př. 1),  (tj. $>0<+r\to s$),  atd. Poslední blok je možno nahradit částí instrukční sítě:  .

Je zřejmé, že schémata dané instrukční sítě nejsou dána jednoznačně; speciálně mohou být různě podrobná. Při sestavování instrukčních sítí pro řešení velmi složitých úloh je někdy výhodné sestavit předem celou řadu schémat různé podrobnosti. Některé části instrukční sítě, které nesou samostatný charakter (zpracovávají dílčí problém), je výhodné sestavovat odděleně; nazýváme je pak *instrukčními podsítěmi* a ve schématu vlastní instrukční sítě je nahrazujeme bloky s dvojitými konturami. Pro často se opakující úlohy, jakými jsou např. výpočet hodnot základních funkcí e^x, $\ln x$, $\sin x$, $\cos x$, $\sqrt{x}$ ap., bývají instrukční podsítě u počítače trvale k dispozici a není je tedy třeba sestavovat pro každou úlohu zvlášť. Jsou to tzv. *standardní* instrukční podsítě.

Příklad 3. Schéma instrukční sítě pro řešení kvadratické rovnice $ax^2 + bx + c = 0$, $a \neq 0$, a, b, c jsou reálná čísla zapsaná na paměťových místech o stejných adresách. Předpokládáme, že máme k dispozici standardní instrukční podsíť pro výpočet druhé odmocniny z reálného čísla.



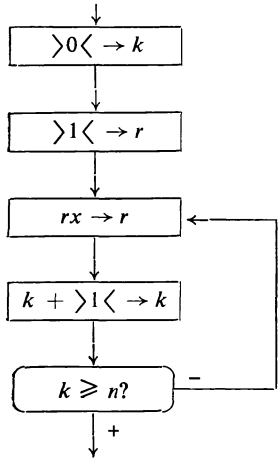
Jak se rozepíší do instrukcí bloky 1, 3,5,5′,7,7′, je zřejmé; bloky 2,6,6′,8,8′,9,9′ jsou přímo instrukčními bloky, kdežto bloky 4 a 4′ zastupují instrukční podsíť pro výpočet druhé odmocniny. Po skončení výpočtu budou vytisknuta tři čísla: je-li posledním z nich číslo 1, pak daná rovnice má reálné kořeny — jejich hodnoty předcházejí. Je-li poslední z nich číslem 0, jsou kořeny komplexně sdružené; prvé číslo z trojice udává jejich reálnou část, druhé imaginární část.

Poznamenejme, že vysvětlení významu vytisknutých čísel — tak, jak bylo právě na příkladě provedeno — je užitečné a ve složitějších případech i nutné sestavit na základě instrukční sítě ještě před započetím vlastního výpočtu (někdy se mluví o *objasnění tisků*).

Zapsat instrukci dané sítě dá mnohdy více práce, než ji přímo provést, např. pomocí kalkulačního stroje. Proto se skutečné výhody a přednosti samočinného počítače projeví teprve tehdy, jestliže v instrukční síti budou existovat instrukce, které se budou během výpočtu provádět víckrát. To může nastat např. tehdy, když na bloku zapisu-

jícím takovou instrukci nebo na některém bloku, který předchází, končí víc než jedna šipka dané instrukční sítě. Dojde-li skutečně k takovému opakování instrukce, řekneme, že proběhl *cyklus*. Poznamenejme, že při sestavování příslušné instrukční sítě je třeba vždy zajistit, aby při vlastním výpočtu proběhl cyklus jen tolikrát, kolikrát je třeba, to je pokud není splněna jistá podmínka, tzv. *podmínka cyklu* (týkající se např. počtu cyklů).

Příklad 4. Schéma instrukční sítě pro výpočet x^n, kde x, n jsou čísla zapsaná na paměťových místech o stejných adresách, n přirozené.



... číslo, zapsané na paměťovém místě k bude udávat, kolikrát proběhl cyklus;

... na paměťovém místě r se budou postupně tvořit mocniny x: 1, x, x^2, ... x^n;

... exponent mocniny čísla x, zapsané na paměťovém místě r, vzrostl o 1;

... zaregistrovalo se proběhnutí cyklu;

... podmínka cyklu.

§ 2

Pro pochopení problematiky programování je nutno seznámit se aspoň stručně se zápisem čísel ve stroji.

Ve většině počítačů – a hlavně takové budeme mít v dalším na mysli – jsou čísla zapsána ve svém *dvojkovém kódu*, tj. ve svém vyjádření pouze pomocí cifer 0 a 1 (tzv. *bitů*), a to tak, že na každém paměťovém místě se zapisuje týž počet těchto bitů. Dvojkový kód bývá často v úzkém vztahu k zápisu čísla v soustavě o základu 2.

Připomeňme, že zápisem čísla a v *soustavě o základu* g (g přirozené, $g \geqslant 2$) rozumíme posloupnost $\pm a_n \dots a_1 a_0, a_{-1} \dots a_{-k} \dots$, kde $0 \leq a_i \leq g - 1$, je-li $a = \pm (a_n g^n + \dots + a_1 g^1 + a_0 g^0 + a_{-1} g^{-1} + \dots + a_{-k} g^{-k} + \dots)$. Je tedy např. 7,25 zapsáno v soustavě o základu 2 jako 111,01, neboť $7{,}25 = 2^2 + 2^1 + 2^0 + 2^{-2}$. Běžně užívaná je soustava o základu 10.

Dvojková soustava je nepřehledná. Proto užíváme pro pomocný zápis všeho číselného materiálu, který má být zapsán do paměti počítače, soustavy o základu 8 (někdy též 16); tato soustava je již přehledná a navíc – na rozdíl od soustavy o základu 10 – má k dvojkové úzký vztah.

Skutečně ze zápisu čísla v soustavě o základu 8 dostaneme jeho zápis v soustavě o základu 2 tak, že každou cifru prvého zápisu chápanou samu jako číslo zapíšeme pomocí tří cifer v soustavě

druhé. Je tedy např. číslo zapsané v soustavě o základu 8 posloupností 507, 2 zapsáno v soustavě o základu 2 posloupností 101 000 111, 010 (a v běžné soustavě posloupností 327,25).

V následujících příkladech jsou všechna čísla, jež mají být zapsána v paměti stroje, zapsána v soustavě o základu 8. Tato skutečnost nekomplikuje podstatně práci s počítačem: existují jednoduchá pravidla pro převod zápisů čísel v různých soustavách. Tuto práci můžeme často svěřit i stroji. Jindy je možno celé úvahy provádět v soustavě o základu 8 (která je ve skutečnosti mnohem přirozenější a má po matematické stránce lepší vlastnosti než soustava o základu 10, jejíž jedinou předností je tradice).

K zápisu čísel do paměti ještě dodejme: i když — jak bylo řečeno — je dvojkový kód většinou v blízkém vztahu k zápisu téhož čísla v soustavě o základu 2, nemůže s ním splývat, neboť v dvojkovém kódu je třeba zachytit navíc (aspoň) znamení čísla, resp. polohu řádové čárky. Zatímco znamení lze pohodlně zapsat jediným bitem (cifra 0 např. zapisuje +, cifra 1 zapisuje −), je situace s řádovou čárkou složitější. Řádová čárka může být umístěna pevně za určitým bitem paměťových míst (počítač má *pevnou řádovou čárku*). Stroj pak může zapsat pouze čísla v jiném rozmezí (např. pouze ta, která jsou v absolutní hodnotě menší než 1, je-li řádová čárka umístěna před prvým bitem). Ostatní čísla zpracovává pouze pomocí vhodných obratů, např. měřítek, které zaručují, že ani v zadání ani během výpočtu se nevyskytnou čísla vybočující z uvedeného rozsahu, tj. že nedojde k tzv. *přeplnění*. Jiné počítače pracují v systému *pohyblivé řádové čárky*, tj. dovolují přímo zapsat a zpracovávat čísla s libovolnou polohou řádové čárky. Všechny příklady uvedené v článku předpokládají, že problém řádové čárky je vyřešen; detailněji se jím zde zabývat nebudeme.

Jak již bylo řečeno, k tomu, aby se počítač mohl řídit instrukcemi sestavené instrukční sítě, a to bez dalšího zásahu člověka, je třeba, aby i tato síť byla do něho jistým způsobem vložena. Většina moderních počítačů si instrukční síť, předem vhodně upravenou, zapisuje do paměti, a to na základě obratu, který je popsán v následujících dvou bodech:

1. Podle jistých, pro daný stroj předem pevně daných pravidel přiřadíme každé instrukci jisté číslo (toto přiřazení je při tom vzájemně jednoznačné), které nazveme jejím *kódem*. Kódované instrukce jsou tedy čísla a jako taková je lze zapsat do paměti. Uvažujme, že zjednodušený počítač zapíše každou instrukci na jedno paměťové místo. Je-li kód instrukce ξ zapsán na paměťovém místě o adrese r, říkáme, že r je *adresa instrukce* ξ.

Kód instrukce sestává z několika čísel zapsaných vedle sebe (tvořících tak posloupnost cifer, a tím výsledné číslo — kód), z nichž prvé udává druh operace, která má být splněním instrukce provedena (sečti, odečti, znásob, vyděl, zjisti, vytiskni, zastav), zbývající adresy paměťových míst, na nichž jsou zapsána čísla, jichž se má instrukce týkat. V našem případě přesněji:

v případě instrukcí typu **A** po řadě adresy r, s, t;
v případě instrukcí typu **B** po řadě adresu r, adresu instrukce ξ, adresu instrukce η;
v případě instrukce typu **C** adresu r;
(instrukce typu **D** nevyžaduje podobných údajů).

Příklad 5. U mnoha počítačů je počet druhů prováděných operací roven řádově desítkám (v našem zjednodušeném případě ovšem pouze sedmi), počet paměťových míst tisícům. Kódujeme tedy druh operace dvěma ciframi, adresy obsahují po čtyřech cifrách. Kód celé instrukce sestává pak ze 14 cifer; při tom nechť i instrukce typu **C** a **D** mají kódy o 14 cifrách, kde v prvém případě nezáleží na posledních osmi, v posledním dokonce na posledních dvanácti cifrách.

Nechť operace jsou kódovány takto: sečti 01, odečti 02, znásob 03, vyděl 04, zjisti 21, vytiskni 12, zastav 31. Pak

instrukce	tj.	má kód
2040 + 1120 → 0030	sečti čísla zapsaná na paměťových místech 2040 a 1120 a součet zapiš na paměťové místo 0030;	01 2040 1120 0030
1100 ≥ 0? (+ → ξ, − → η)	zjisti, zda číslo zapsané na paměťovém místě 1100 je nezáporné;	21 1100 2000 2006
(adr. instr. ξ = 2000) (adr. instr. η = 2006)	je-li, proveď instrukci, jejíž kód je zapsán na paměťovém místě 2000, jinak tu s kódem na paměťovém místě 2006;	
T 3000 →	vytiskni číslo zapsané na paměťovém místě o adrese 3000;	12 3000 0000 0000
Z	zastav výpočet	31 0000 0000 0000

a podobně

V tomto příkladě jsme uvedli způsob zakódování (některých) instrukcí skutečného počítače, a to sovětského počítače BESM (bystrodějstvujuščaja elektronnaja sčotnaja mašina; instrukce typu B je v příkladě trochu přizpůsobena).

V příkladě jsme narazili na otázku rozsahu paměti číslicových počítačů; věnujme proto v poznámce několik vět kvantitativní charakteristice strojů, i když tím poznámka vybočí z celkové koncepce paragrafu. Předně: vedle vlastní *vnitřní* paměti, popsané v § 1, mají počítače ještě obvykle pomocnou *vnější* paměť o kapacitě často mnohonásobně větší. S čísly zapsanými ve vnější paměti není možno přímo pracovat; zvláštní instrukce však zprostředkují přepis čísel (skupin čísel) z jedné paměti do druhé. Druhá poznámka se týká rychlosti práce stroje: elektronické počítače provádějí stovky až statisíce instrukcí za vteřinu.

2. Zakódované instrukce dané instrukční sítě zapíšeme postupně na libovolná (ovšem dosud neobsazená) místa paměti, ale vždy tak, aby instrukce následující po instrukci ξ typu **A** nebo **C** byla zapsána na následujícím paměťovém místě (to znamená, má-li ξ adresu r, pak adresa instrukce, na níž končí šipka jdoucí od bloku zapisujícího instrukci ξ, je $r + 1$). Počítač je tedy zařízen tak, že po vykonání jisté instrukce provede instrukci zapsanou na následujícím paměťovém místě. Výjimku tvoří instrukce typu **B**, u nichž jsou adresy následujících instrukcí přímo zapsány, a ovšem instrukce typu **D**.

Existují i počítače (např. americký IBM 650), u nichž jednotlivé instrukce sítě mohou být zapsány do paměti na zcela libovolná místa; pak ovšem i instrukce typu A a C musí obsahovat údaje o adrese následující instrukce.

Instrukční síť je tedy připravena k zápisu do paměti, jakmile: 1. každá její instrukce je zakódována, 2. u každé instrukce je udána její adresa (je dáno její umístění v paměti).

Takto zpracovanou instrukční síť nazveme *programem*; instrukční podsíť je analogicky *podprogramem*.

Program sestavuje *programátor* na základě instrukční sítě nebo jejího schématu. Zkušenému programátorovi stačí i málo detailní schéma, u jednoduchých úloh je dokonce možné sepsat program přímo. Vedle schématu instrukční sítě, ve kterém adresy jsou obvykle označeny jestě písmeny, potřebuje programátor znát konkrétní adresy paměťových míst, na nichž mají být zapsány počáteční hodnoty úlohy, konstanty potřebné k řešení, adresy pomocných paměťových míst a konečně umístění samotného programu v paměti; musí tedy znát *obsazení paměti*. Rozhodnout o vhodném obsazení paměti nebývá těžké, pokud úloha neklade velké nároky na paměť (větší počet paměťových míst zůstane nevyužit) nebo pokud se nesnažíme co nejvýhodněji sestavit program podle jistých kritérií (*optimální programování*).

Příklad 6. Uvažujme počítač se způsobem zakódování z příkladu 5 a vraťme se k příkladu 1. Nechť čísla r, s, 0 jsou po řadě uložena na paměťových místech o adresách 6000, 6002, 5000; zvolme dále adresy pomocných paměťových míst p a p': 6010 a 6012. Zapíšeme-li konečně prvou instrukci na paměťové místo o adrese 0100, bude program vzniklý z instrukční sítě z příkladu 1 vypadat takto:

Adresa	Kódovaná instrukce	Odpovídající blok instrukční sítě
0100	01600060026010	$r + s \to p$
0101	02600060026012	$r - s \to p'$
0102	04601060126010	$p : p' \to p$
0103	21601001050104	$\overset{+}{-} p \geqslant 0?$
0104	02500060106010	$> 0 < -p \to p$
0105	12601000000000	$\to Tp$

Je-li dále např. $r = 30000000000000$, $s = 40000000000000$ vypadají některá následující paměťová místa takto:

Adresa	Číslo
5000	00000000000000
6000	30000000000000
6002	40000000000000

Často se vyskytuje otázka, která již vlastně byla zodpovězena tím, co bylo řečeno, ale na kterou je dobré odpovědět explicitně: Jak rozezná počítač zakódované instrukce od „obyčejných" čísel? Jedině jejich umístěním v paměti. Jakmile je veškerý číselný materiál zapsán před započetím vlastní práce stroje do paměti, vyznačí obsluha stroji jistou adresu. Číslo zapsané na paměťovém místě o této adrese chápe stroj jako zakódovanou instrukci (existují ovšem čísla, která nejsou kódy žádných instrukcí; jak reaguje počítač na taková čísla, zapíšeme-li je na místa instrukcí, závisí na typu počítače; většinou je „přeskočí", aniž by cokoliv jiného provedl),

rozšifruje ji, provede a přejde k instrukci o adrese o 1 větší, nebo byla-li právě prováděná instrukce typu **B**, přejde k instrukci, jejíž adresa je zapsána v instrukci prováděné. „Přejít k instrukci" pak v obou případech opět znamená vyhledat paměťové místo o uvedené adrese, číslo zapsané na tomto místě chápat opět jako kód instrukce a postup opakovat, pokud nedojde k instrukci typu **D**. Byla by to chyba programátora, kdyby byl stroj nucen považovat za kód instrukce číslo jiného významu. Zároveň je vidět, že malá chyba v zápisu instrukce (zejména u adresné části instrukce typu **B**) může úplně znehodnotit celou práci stroje. Není ovšem takovou chybou zničena celá práce matematikova, resp. programátorova: stačí instrukci opravit a znovu zavést veškerý číselný materiál do paměti.

§ 3

Formální shoda mezi čísly a zakódovanými instrukcemi rozšiřuje podstatně možnosti počítače: umožňuje měnit během výpočtu jednotlivé instrukce programu.

Instrukce, kterými konáme změnu jiných instrukcí, vyznačujeme pro jejich poněkud zvláštní postavení mezi ostatními instrukcemi bloky s dvojitou levou konturou; nazýváme je instrukce *formování*. Instrukce formování ztrácejí smysl, není-li současně užito cyklu: instrukci má smysl měnit jedině, počítáme-li s jejím opětovným užitím.

Změnu instrukce je možno provádět různými instrukcemi — podle vymezení instrukcí typu **A** je např. obecně možné přičítat k dané (rozumí se zakódované) instrukci číslo zapsané na libovolném paměťovém místě, speciálně dokonce kód další instrukce. Totéž platí i o odečítání, násobení a dělení. Jen některé z těchto operací nad instrukcemi však dávají přirozeným způsobem opět instrukci: je sice pravda, že je dost velká pravděpodobnost, že např. podíl kódů dvou instrukcí bude opět kódem nějaké instrukce, ale jen málokdy bude vztah mezi všemi třemi instrukcemi takový, aby se ho dalo při programování využít. Z nejčastěji užívaných instrukcí provádějících změny jiných instrukcí si blíže všimneme jen dvou nejběžnějších:

1. Přičítání (odečítání) čísla, které je zapsáno na paměťovém místě o adrese p ke kódu instrukce ξ_0. Číslo zapsané na paměťovém místě p je při tom takové, že v instrukci ξ_0 se změní pouze její adresová část (tj. některé z adres, které tvoří její kód). Obecně ke každé z adres r, s, t bude po řadě přičteno při této změně některé celé číslo p_r, p_s, p_t.

Příklad 7. Přičtu-li k instrukci 01204011200030 (viz příklad 5) číslo 00002000000002, dostanu kód instrukce 01206011200032.

Zároveň si všimněme, že někdy může mít smysl i obecnější úprava, než vyšetřujeme (týkající se nejen adresové části): přičteme-li k instrukci o adrese 0100 z příkladu 6 číslo 01000000000002, dostaneme instrukci o adrese 0101.

Při provedení nově vytvořené instrukce — označme ji ξ_1 — bude provedeno totéž, tj. táž operace jako při provádění instrukce ξ_0, avšak s čísly zapsanými na paměťových místech o jiných adresách (lišících se od původních o p_r, p_s a p_t).

Jestliže instrukci ξ_1 podrobíme téže změně jako instrukci ξ_0 (přičteme k ní totéž číslo) a nově vzniklou instrukci ξ_2 opět téže změně a pokračujeme-li tak dále až k in-

strukci ξ_k, je výhodné chápat instrukce $\xi_0, \xi_1, \ldots, \xi_k$ jako proměnnou instrukci ξ, $\xi = \xi_i$ závisící na celočíselném parametru $i\,(i = 0, \ldots, k)$.

Jindy uskutečňujeme nad instrukcemi dva druhy změn; v tomto případě je opět výhodné chápat vzniklé instrukce jako proměnnou instrukci, jež je funkcí dvou parametrů i, j: $\xi = \xi_{i,j}$. Prvému druhu změny odpovídá změna parametru i, druhému změna parametru j. Podobně dospějeme k pojmu instrukce, která závisí na více parametrech.

Vzhledem k tomu, že jsme se omezili jen na takové změny, které se týkají adres, je možno parametry připisovat jako indexy přímo k adresám v symbolickém zápisu instrukcí (s případným vynecháním toho indexu u té adresy, která se s jeho změnou nemění).

Příklad 8. Nechť při změně prvého druhu vzroste prvá adresa instrukce 01204011200030 (viz př. 7) o 20, třetí o 2; při změně druhého druhu všechny adresy o 1. Označíme-li $\xi_{i,j} = r_{ij} + s_j \to t_{ij}$, je např.:

$$\begin{array}{lll} \xi_{1,0} = r_{10} + s_0 \to t_{10} & \text{a její kód je} & 01206011200032, \\ \xi_{0,1} = r_{01} + s_1 \to t_{01} & & 01204111210031, \\ \xi_{1,3} = r_{13} + s_3 \to t_{13} & & 01206311230035. \end{array}$$

Pojem instrukce závislé na parametrech umožňuje zjednodušení schémat některých instrukčních sítí: ve schématu totiž nemusíme uvádět, jakým způsobem mají být instrukce konkrétně změněny, tj. neurčujeme čísla, která se mají k jednotlivým adresám přičítat, ale jako instrukce provádějící změnu zapíšeme nikoliv instrukce měnící adresy, nýbrž *pseudoinstrukce* měnící parametry. (Pseudoinstrukcí zde rozumíme výraz, jenž má formální tvar instrukce a vyjadřuje tedy jistý rozkaz, při tom však splnění tohoto rozkazu vyžaduje provedení speciální posloupnosti vlastních instrukcí.) Otázkou vhodných změn adres je ovšem nutno zabývat se před rozepisováním schématu instrukční sítě do programu.

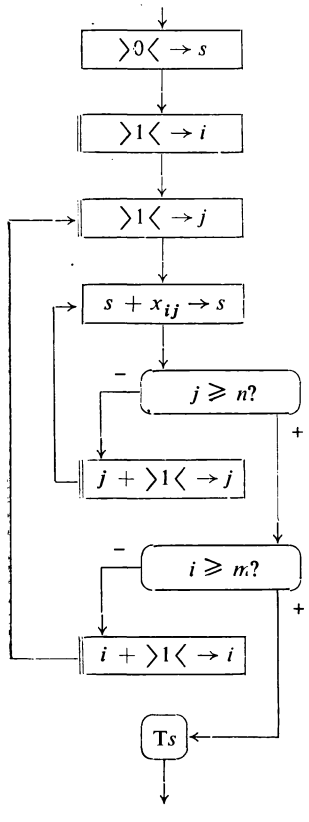
V následujícím příkladu ukážeme užití takových pseudoinstrukcí; poněkud zjednodušeným opakováním úlohy bude příklad 12, v kterém ukážeme řešení ve skutečných instrukcích (viz př. 9 na následující stránce).

V tomto odstavci jsme se omezili na přičítání (odčítání) čísel ke kódům (od kódů) instrukcí, snadno je však možné některé úvahy zobecnit. Také je na místě zmínit se o tom, že některé počítače mají speciální instrukce formování, které obvykle nahrazují — aspoň částečně — obraty, jež byly právě popsány. Tyto instrukce mohou značně ulehčit programátorovu práci, nejsou však nezbytné.

2. Zápis instrukce η na místo instrukce ξ. Jde tedy o přepis čísla na jednom paměťovém místě na číslo zapsané na druhém paměťovém místě. Obecně je smysl a provedení tohoto jednoduchého obratu jasné (viz příklad 2); budeme se proto poněkud podrobněji zabývat jen speciálním případem, ve kterém se současně seznámíme s dalšími užívanými pojmy.

Představme si, že od jistého bloku schématu instrukční sítě vychází ne jedna, ale více šipek $\alpha_1, \alpha_2, \ldots, \alpha_n$ a že se ve schématu vyskytují bloky $\boxed{\alpha_i \to \alpha}\longrightarrow$; každý ten-

Příklad 9. Schéma instrukční sítě pro výpočet $\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}$, kde m, n, x_{ij} $(i = 1, \ldots, m;\ j = 1, \ldots, n)$ jsou čísla zapsaná na paměťových místech o stejných adresách.



... na paměťovém místě s budou postupně zapisovány částečné součty;

... pseudoinstrukce ustavující počáteční hodnotu parametru i instrukce ξ;

... pseudoinstrukce ustavující počáteční hodnotu parametru j instrukce ξ;

... instrukce ξ: k částečnému součtu na s přičte se další člen;

... podmínka („vnitřního") cyklu;

... pseudoinstrukce měnící hodnotu parametru j v instrukci ξ;

... podmínka („vnějšího") cyklu;

... pseudoinstrukce měnící hodnotu parametru i v instrukci ξ;

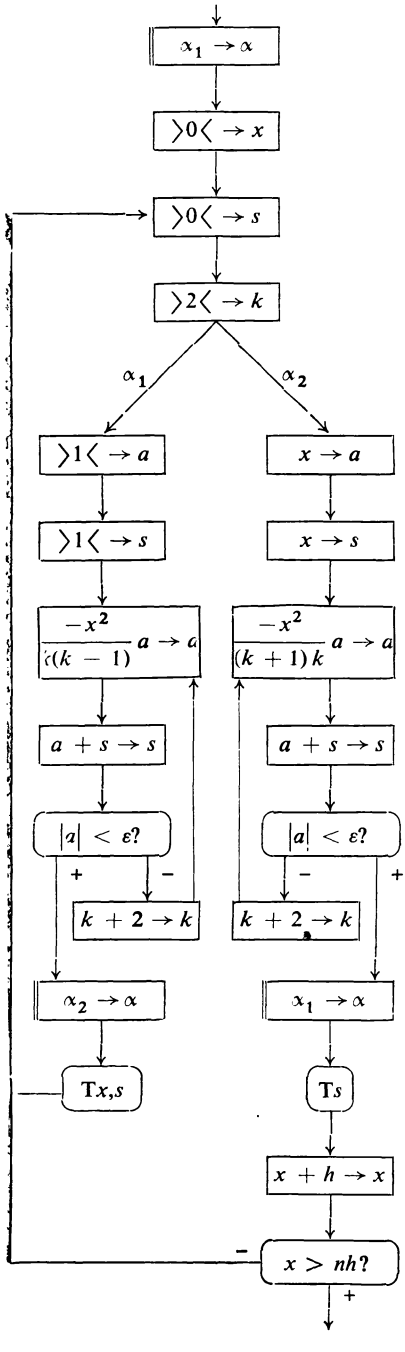
... tisk výsledku.

to blok zapisuje pseudoinstrukci, na jejímž základě počítač považuje za platnou šipku α_i, a to tak dlouho, pokud nedojde k provedení pseudoinstrukce $\boxed{\alpha_j \to \alpha} \longrightarrow$ pro $j \neq i$. Užijeme-li tohoto obratu, řekneme, že jsme užili *výhybky*. Snadno uznáme, že tento zvláštní případ zapadá pod obecný případ vytyčený na začátku odstavce, uvědomíme-li si, že speciálním případem instrukčního typu **B** je instrukce: přejdi k instrukci ξ (stačí položit např. $r = \rangle 1 \langle$). Užít výhybky znamená nahrazovat v příslušném místě schématu vzájemně instrukce tohoto typu; z jejich symbolického zápisu však ponecháváme pouze šipky.

Popsané instrukce nesou název instrukcí *nepodmíněného přechodu*, a jsou tedy speciálním případem instrukcí podmíněného přechodu. U mnoha počítačů existují jako samostatné (mají svůj zvláštní kód).

Příklad 10. Schéma instrukční sítě pro tabelaci funkcí $\cos x$, $\sin x$ pro $x = 0, h, 2h, \ldots, nh$. Výpočet provádíme pomocí rozvoje v Taylorovu řadu: $\cos x = 1 - x^2/2! + x^4/4! - \ldots$, $\sin x =$

$= x - x^3/3! + x^5/5! - \ldots$, přičemž výpočet považujeme za dostatečně přesný, jakmile poslední přičítaný člen je v absolutní hodnotě menší než dané $\varepsilon > 0$.



... počáteční hodnota výhybky

... počáteční hodnota parametru

... na paměťovém s budou uchovány částečné součty řad

... k určuje přičítaný člen (viz další bloky)

... výhybka: platí-li α_1, počítá se $\cos x$; platí-li α_2, počítá se $\sin x$

... na paměťovém místě a budou uchovány jednotliví členové řad

... výpočet $(\frac{1}{2}k + 1)$-ního členu řady pomocí předcházejícího členu (rekurencí)

... tvoření částečných součtů

... podmínka cyklu (je přesnost dostačující?)

... přechod k dalšímu členu řady

... změna výhybky

... tisk argumentu a výsledků

... přechod k další hodnotě argumentu

... podmínka cyklu (je výpočet skončen?)

Objasnění tisků: po skončení uvedené etapy výpočtu budou vytisknuta postupně tato čísla: 0, cos 0, sin 0, h, cos h, sin h, $2h$, cos $2h$, sin $2h$, ..., nh, cos nh, sin nh (s uvažovanou přesností).

Sestavování schémat instrukční sítě není tak obtížné, jak se může na prvý pohled zdát. Základním úkolem je vždy nalezení vhodného algoritmu řešení. Jeho formální zachycení do instrukční sítě se děje obvykle zcela mechanicky; většinou tak, že zachytíme nejprve jádro algoritmu a schéma postupně doplňujeme a zpřesňujeme.

V tomto příkladě je přirozené sestavit nejprve bloky, tvořící částečně součty (typický obrat sumace), potom bloky pro výpočet $(k/2 + 1)$-ních členů, v nichž je zachyceno jádro algoritmu — cyklus rekurence. Ostatní bloky se tvoří snadno.

Viděli jsme, že instrukce mohou měnit nebo vytvářet jiné instrukce. Obecněji si lze představit instrukční síť, resp. program, sestavený nikoliv k řešení předložené úlohy přímo, nýbrž k sestavení programu řešení této úlohy. Takový program vyššího typu, tzv. *kompilační (programující) program* se může sestavovat mnohem pohodlněji než přímý program sám. Užívání kompilačních programů a tím různých typů pseudoinstrukcí, bohaté užívání podprogramů všech druhů a v současné době i snaha po sjednocení symboliky projevující se sestavením jakéhosi „strojového esperanta" (tzv. ALGOLu), to vše přispívá k jisté *automatizaci programování*, jejímž posledním cílem je vyloučit z programátorovy práce všechnu mechanickou práci. Ani nejdokonalejší prostředky automatizace nevyřadí ovšem rozhodující matematikovu práci — stanovení základní myšlenky algoritmu pro řešení dané úlohy; umožní jen její pohodlnější, obecnější formulaci. Automatizace programování není také jen záležitostí teorie a praxe práce na počítačích; přispívá k ní všeobecný rozvoj matematiky, která nachází stále obecnější (zahrnující větší třídu úloh) i účinnější algoritmy a podrobuje preciznímu zkoumání i sám pojem algoritmu, vymezujíc třídu úloh, které vůbec — principiálně — mohou být algoritmicky řešitelné (viz např. [5]). Zajímavé je, že do této třídy rozhodně nepatří všechny matematické problémy.

§ 4

Instrukce vyšetřovaného zjednodušeného počítače měly tu charakteristickou vlastnost, že obsahovaly maximálně 3 adresy a některé z nich (typu **A** a **B**) tolik adres skutečně obsahovaly — počítač byl *tříadresový*.

Ze skutečných tříadresových počítačů uveďme vedle již citovaného BESM sovětskou Strelu a americké počítače MIDAC a NORC.

Uvažme nyní (zjednodušený) počítač s trochu odlišnými instrukcemi:

A — 1. Sečti (odečti, znásob, vyděl) čísla, z nichž prvé je zapsáno na zvláštním paměťovém místě bez adresy, tzv. *registru R*, druhé na paměťovém místě o adrese r a výsledek zapiš na registr R. Potom proveď instrukci ξ. Symbolicky:

$$\boxed{R + r \to R} \longrightarrow \left(\boxed{R - r \to R} \longrightarrow, \ \boxed{R \,.\, r \to R} \longrightarrow, \ \boxed{R : r \to R} \longrightarrow \right);$$

při tom šipka končí na bloku, který zapisuje instrukci ξ.

A – 2. Zapiš na registr R číslo zapsané na paměťovém místě o adrese r a přejdi k instrukci ξ. Symbolicky:

$\boxed{r \to R} \longrightarrow$, kde šipka končí na bloku, který zapisuje instrukci ξ.

A – 3. Zapiš na paměťové místo o adrese r číslo zapsané na registru R a přejdi k instrukci ξ. Symbolicky:

$\boxed{R \to r} \longrightarrow$, kde šipka končí na bloku, který zapisuje instrukci ξ.

B. Zjisti, zda číslo zapsané na registru R je nezáporné; jestliže ano, přejdi k instrukci ξ; jestliže ne, k instrukci η. Symbolicky:

$\overset{-}{\longleftarrow}\boxed{R \geqslant 0?}\overset{+}{\longrightarrow}$, kde šipka $\overset{+}{\longrightarrow}$ končí na bloku, který zapisuje instrukci ξ, šipka $\overset{-}{\longrightarrow}$ na bloku, který zapisuje instrukci η.

C, **D** stejně jako dříve.

Kód instrukce tohoto počítače nechť sestává ze dvou čísel zapsaných vedle sebe: prvé udává druh operace, druhé adresu r u instrukcí typu **A** – 1, 2, 3, **C** či adresu instrukce η u instrukce typu **B**. Nechť konečně pro počítač s těmito instrukcemi platí totéž, co bylo uvedeno v § 2 odst. 2 s tím jediným rozdílem, že u instrukce typu **B** musí být instrukce ξ zapsána na následujícím paměťovém místě (zatímco adresa instrukce η je zachycena v kódu).

Takovýto počítač má tu vlastnost, že každá jeho instrukce obsahuje maximálně jednu adresu – je to počítač *jednoadresový*.

Příklady jednoadresových počítačů jsou sovětský URAL I, anglický ELLIOTT 803 a západoněmecký LGP (všechny pracují i v ČSSR).

Instrukce jednoadresového počítače mají podstatně jednodušší strukturu než instrukce počítače tříadresového, je jich však potřeba k řešení téhož úkolu obecně víc (na druhé straně snadno nahlédneme, že každou instrukci tříadresového počítače lze splnit postupným provedením nejvýše tří instrukcí jednoadresového – speciálně schémata instrukčních sítí pro tříadresový počítač jsou zároveň schématy sítí i pro počítač jednoadresový).

Příklad 11. Ukážeme, jak vypadá instrukční síť a program řešení úlohy formulované v příkladu 1 na počítači URAL I. Adresy počátečních hodnot i pomocných paměťových míst volíme tytéž jako v příkladě 6, se kterým řešení porovnáme (viz schéma na str. 30).

Poznamenejme, že tato část programu by se dala poněkud zjednodušit, kdybychom ho nechtěli srovnávat s programem z př. 6 a že zacházení s instrukcí 21*a* je ve skutečnosti poněkud složitější, než je uvedeno; její užití zde je však zcela v pořádku. U tohoto příkladu je snad též na místě připomenout, že veškerý číselný materiál je zapsán v soustavě o základu 8 (v které po čísle 0107 následuje 0110 apod.).

Blok instr. sítě	Adresa	Kód instrukce	Program Vysvětlivka		Odpovídá blok z př. 6
$r \to R$	0100	02 6000	02 a je kódem instr. $a \to R$		
$R + s \to R$	0101	01 6002	01 a	$a + R \to R$	$r + s \to p$
$R \to p$	0102	16 6010	16 a	$R \to a$	
$r \to R$	0103	02 6000			
$R - s \to R$	0104	03 6002	03 a	$R - a \to R$	$r - s \to p'$
$R \to p'$	0105	16 6012			
$p \to R$	0106	02 6010			
$R : p' \to R$	0107	07 6012	07 a	$R : a \to R$	$p : p' \to p$
$R \to p$	0110	16 6010			
$R \geqq 0?$	0111	21 0113	21 a $R \geqq 0?$ $\overset{+}{\to}$ k násl. instr. $\overset{-}{\to}$ k instr. o adr. a		$p \geqq 0?$
	0112	22 0115	22 a nepodm. přech. k instr. o adr. a		$-$ $+$
$>0< \to R$	0113	02 5000			$>0< - p \to p$
$R - p \to R$	0114	03 6010			
$\to TR$	0115	32 0000	32 a TR (na a nezávisí)		Tp

Vedle jednoadresových a tříadresových počítačů můžeme podobně vyšetřovat počítače dvouadresové, čtyřadresové a pětiadresové, které mají skutečně své reálné předlohy.

Z dvouadresových počítačů je v ČSSR instalován sovětský MINSK; prvý čs. počítač SAPO je pětiadresový.

V principu je jasné, jakým způsobem mohou být modifikovány instrukce u těchto počítačů; podrobněji ukážeme činnost dvouadresového počítače na příkladě, na kterém zároveň bude demonstrován způsob změny instrukcí, popsaný v § 3, odst. 1 (srv. př. 9).

Příklad 12. Program pro výpočet $\sum_{i=1}^{n} x_i$ na počítači MINSK, kde $>x_i< = \alpha + i$. Sestavme nejprve *program v obecných adresách*, tj. program, v němž některá čísla mohou být ještě označena písmeny:

Adresa	Kód instrukce			Blok instr. sítě	Vysvětlivka
$k + 0$	05	$>0<$	s	$>0< \to s$	05 rs je kód instr. $r \to s$
1	05	$k + 6$	$k + 3$	$>0< \to i$	na pam. místo $k + 3$ se zapíše 00 αs
2	00	$>1'<$	$k + 3$	$i + 1 \to i$	00 rs je kód instr. $r + s \to s$
3	—	—	—	$x_i + s \to s$	na počátečním zápisu vzhledem k instr. na $k + 1$ nezáleží
4	11	$k + 3$	$k + 7$	$n - i \to R$	11 rs je kód instr. $s - r \to R$
5	54	$k + 2$	$k + 10$	$R \geqq 0?$	54 rs „ $R \geqq 0?$ $\overset{-}{\to}$ k instr. o adr. r $\overset{+}{\to}$ k instr. o adr. s
6	00	α	s		pam. místa zapisující pomocně nultou
7	00	$\alpha + n$	s		a poslední instr. cyklu na $k + 3$

Změnu počáteční instrukce, zapsané na adresu $k + 3$, která mohla být původně libovolná, obstarávají: instrukce zapsaná na adrese $k + 1$ (která tam přenese nultou instrukci z adresy $k + 6$ — srv. § 3, odst. 2) a instrukce zapsaná na adrese $k + 2$, která k instrukci na adrese $k + 3$, a to k její prvé adrese, jak dále uvidíme (viz adresa 1400), přičte číslo 1. Všimněme si, že podmínku cyklu tvoříme (instrukcí $k + 4$) pomocí odečítání instrukcí, přesněji řečeno jejich kódů; to je ovšem možné, jak pochopíme, zapíšeme-li kódy těchto instrukcí. Na příkladě je také vidět, že pseudoinstrukce (konkrétně: $i + 1 \to i$) je možné někdy vyjádřit jedinou vlastní instrukcí (zde: $k + 3 + >1' < \to k + 3$).

Přejděme k sestavení vlastního programu a volme $k = 1000$, $s = 1200$. Je-li nyní např. $n = 20$, $\alpha = 1400$, $>0< = 0000$, $>1'< = 1400$, bude příslušná část paměti před začátkem výpočtu vypadat takto:

Program		Konstanty	
Adresa	Kód instrukce	Adresa	Zapsané číslo
1000	05 0000 1200	0000	00 0000 0000
1001	05 1006 1003	1400	00 0001 0000
1002	00 1400 1003	1401	x_1
1003	00 0000 0000	1402	x_2
1004	11 1003 1007	.	
1005	54 1002 1010	.	
1006	00 1400 1200	1420	x_{20}
1007	00 1420 1200		Pom. pam. místo
1010	pokrač. programu	1200	s

Adresnost počítače je jednou z jeho důležitých charakteristik a má vliv na některé konstrukční zvláštnosti, které se projevují i v programování. Tak např. u jednoadresových počítačů bývá porušena zásada, kterou jsme přijali pro náš zjednodušený počítač: každé číslo je zapsáno právě na jednom paměťovém místě, neboť instrukce ke svému zápisu potřebují méně cifer, než je zapotřebí k dostatečně přesnému zápisu čísel (srv. př. 6 a 11). Proto se zapisují čísla do dvojic po sobě jdoucích paměťových míst (u URALu má prvé z nich vždy sudou adresu — viz př. 11). S opačným jevem se setkáváme u pětiadresového počítače SAPO, ve kterém naopak každá instrukce je rozepsána na dvě sousední paměťová místa.

§ 5

Na programování mají částečně vliv i některé zvláštnosti práce se strojem. Popíšeme je spolu se stručným popisem počítače samého.

Vedle paměti je pro práci počítače nejdůležitější ústrojí, které řídí podle vložené instrukční sítě celou práci stroje, tzv. *řadič*, a kromě něho *operační jednotka*, provádějící všechny instrukce. Zápis do paměti obstarává *vstupní jednotka*, sestávající většinou z *pásky* (perforované, zmagnetované) nebo ze *štítků* a ze *snímače* (pásky, štítků). *Výstupní jednotka*, spojená většinou s *tiskárnou*, umožňuje publikaci výsledků.

Práci stroje je možno kontrolovat na *panelu*, na nějž jsou vyvedeni ukazatelé (např. pomocí doutnavek) signalizující stavy některých důležitých částí počítače:

část řadiče, která přechodně zaznamenává právě prováděnou instrukci i její adresu, registr apod.

Do práce stroje je možno zasahovat během výpočtu z *řídicího stolu*; vedle zavádění pásky a spouštění stroje lze zastavovat stroj buď přímo stiskem tlačítka, nebo nastavením adresy, na níž je zapsána instrukce, po jejímž provedení chceme práci stroje přerušit. Dále je možno provádět čtení nebo zápis libovolného čísla na libovolné paměťové místo (po případném předchozím zastavení stroje). Lze také seřídit stroj tak, aby jeho práce byla podstatně pomalejší a dala se kontrolovat. Je možno rozhodovat o různých variantách činnosti stroje po přeplnění a podobně. Těchto možností se využívá hlavně při prověrce programu, *odstraňování chyb*, které předchází vždy vlastnímu výpočtu. Ovšem i při něm je možno částečně řídit jinak samočinnou práci počítače, např. rozhodovat o dalším postupu v závislosti na veličinách, které stroj není schopen přímo vzít v úvahu (doba potřebná k řešení části programu apod.). V každém případě programátor musí být podrobně obeznámen s těmito možnostmi zasahování do práce stroje.

Technická realizace paměti, operačního ústrojí i řadiče — všimneme-li si hlavně strojů se způsobem zápisu popsaným v § 2 — je založena na výběru vhodných *dvojkových prvků*, tj. fyzikálních elementů, z nichž každý je schopen nalézat se právě v jednom ze dvou možných — dobře odlišitelných — stavů. Např.: vodič nebo elektronka (protéká proud — neprotéká), relé (obvod propojen — rozpojen), magnetický element (zmagnetováno — nezmagnetováno). Vhodným propojením takových elementů (detaily viz monografie [4]) získáme obvody realizující hlavní činnost počítače. Speciálně některé z těchto dvojkových prvků realizují již zmíněné bity paměti. Vstupní zařízení navíc přibírá fotobuňky nebo mechanické snímače a servomotory.

Shrňme nakonec chronologický postup práce matematika, programátora a obsluhy stroje (rovnost mezi nimi může, ale nemusí nastat), a to před řešením a při řešení úlohy na samočinném číslicovém počítači.

Po předložení úlohy počíná si matematik takto:

1. Provede rozbor úlohy, z něhož vyplyne hlavní idea algoritmu, tj. stanovení vhodné numerické metody vzhledem k požadované přesnosti a omezujícím podmínkám počítače (omezená kapacita paměti, existence pevné řadové čárky, nepřesnost počítače ap.). Metoda má být volena tak, aby umožňovala co největší využití cyklů a kontrolu výpočtu.
2. Sestaví schéma (schémata) instrukční sítě a sepíše potřebný číselný materiál.
3. Určí obsazení paměti.
4. Je-li nutno během výpočtu zasahovat do práce stroje, popíše činnost obsluhy.
5. Zapíše objasnění vytištěných výsledků, resp. jejich vyhodnocení.

Programátor dále

6. rozepíše program na blanket, na který přepíše i ostatní číselný materiál (podle 2,3),
7. zařídí přepis na blanketu na pásku.

Obsluha stroje pak

8. zapíše (podle 3) veškerý materiál do paměti stroje,
9. spustí stroj a pracuje podle pokynů (4),
10. předá vytištěné hodnoty spolu s (5) zadavateli úlohy.

Závěrečné poznámky: Tento článek nemá ovšem za úkol naučit programovat, chce jen popsat charakter programování a seznámit s jeho problémy. Proto také text ani příklady nezabíhají do všech podrobností a někde v zájmu obecnosti zjednodušují skutečnost, zatímco jinde, kde zobecnění nelze dobře provést, si všímají podrobněji zvláštního případu. Bylo snahou autora, aby takové zvláštní případy byly dostatečně typické nebo alespoň ilustrační. Terminologie souhlasí v podstatě s ČSN 01 6928, jsou však zavedeny i některé termíny navíc. Z odchylek uvádím pouze rozlišení instrukční sítě a programu, což se v normě nerozlišuje (navíc se užívá termínu kódovaný program). Terminologie není ještě ustálena, proto se vyskytují synonyma: paměťové místo – buňka; vymaže – nuluje; přechod – předání řízení; schéma instrukční sítě – blokové schéma, vývojový diagram; šipka instrukčního bloku – vývojový znak; odstranění chyb (z programu) – ladění.

Literatura

[1] Problemy kibernetiki, sborníky.
[2] A. J. KITOV, *Elektronické číslicové počítače.*
[3] D. D. MAC CRACKEN, *Digital Computer Programming* (též v ruském překladu).
[4] R. K. RICHARDS, *Arithmetic Operations in Digital Computers* (též v ruském překladu).
[5] TRACHTENBROT, *Algoritmy i mašinnoje rešenije zadač.*