CORECTAREA AUTOMATĂ A DEPĂȘIRII UNITĂȚII ÎNTR-O MAȘINĂ CU VIRGULĂ FIXĂ

DE

E. MUTEANU și T. RUS

(Cluj)

Lucrare prezentată la sesiunea asupra problemelor teoretice și practice privitoare la automatizările industriale, noiembrie 1961, București

Mașinile electronice de calcul cifrice, din punctul de vedere al reprezentării numerelor, se împart în două categorii: mașini cu virgulă fixă și mașini cu virgulă mobilă.

În mașinile cu virgulă fixă, numerele se reprezintă sub forma

$$X = \alpha_1\alpha_2 \ldots \alpha_k, \ \alpha_{k+1} \ldots \alpha_n,$$

unde

$$\alpha_i = 0 \text{ sau } 1, \ (i = 1, 2, \ldots, n).$$

În majoritatea mașinilor, virgula se fixează astfel ca

$$|X| < 1.$$

În mașinile cu virgula mobilă, numerele se reprezintă sub forma

$$X = 2^p x_m,$$

unde

$$x_m = \alpha_1\alpha_2 \ldots \alpha_n, \quad \alpha_i = 0 \text{ sau } 1,$$

se numește mantisa numărului X, iar $p = \beta_1\beta_2 \ldots \beta_k$ se numește caracteristica numărului X.

Din punct de vedere constructiv și din punctul de vedere al vitezei de calcul, mașinile cu virgula fixă prezintă avantaje față de mașinile cu virgula mobilă. Din punctul de vedere al pregătirii programului pentru rezolvarea unei probleme, este mult mai complicat să se lucreze pe o mașină cu virgula fixă decît pe una cu virgula mobilă.

Pentru maşinile cu virgula mobilă, nu interesează ordinul de mărime al numerelor care intră în calcul — bineînţeles în limita diapozonului maşinii. Dimpotrivă, în cazul maşinilor cu virgula fixă, programatorul trebuie de la început să prevadă ordinele de mărime ale anumitor rezultate intermediare şi finale şi prin urmare să modifice constantele iniţiale în aşa fel încît ordinele acestor rezultate să nu depăşească o limită bine fixată, determinată de poziţia virgulei. Acest lucru este deseori foarte complicat şi măreşte mult timpul de pregătire a problemei, deoarece, în majoritatea problemelor, determinarea ordinelor de mărime a diferitelor rezultate intermediare se face prin calcule laborioase care pot duce uneori aproape pînă la rezolvarea manuală a problemei. Mai mult decît atît, orice depăşire a limitei stabilite de poziţia virgulei — în cursul calculului la maşină — chiar dacă aceasta este pusă în evidenţă de către maşină, este corectată anevoios şi în majoritatea cazurilor cu pierdere de timp. Pînă în prezent, acest lucru se făcea manual de către programatorul care conducea rezolvarea problemei respective.

Există însă posibilitatea, ca o maşină cu virgula fixă să lucreze într-un regim cu virgula mobilă, cu ajutorul unui program standard. În general, aceste programe standard presupun reprezentarea semilogaritmică a numerelor care intră în maşină. Pentru această reprezentare se poate folosi o celulă a memoriei şi, în acest caz, precizia calculelor se micşorează, sau două celule, ceea ce duce la mărirea numărului de celule folosite pentru rezolvarea problemei.

Oricare ar fi aceste programe standard, ele analizează succesiv fiecare instrucţiune a programului considerat, operînd atît cu mantisa cît şi cu caracteristica numerelor respective, iar dacă numărul este înscris într-o singură celulă, este necesară rotunjirea acestuia.

În cele ce urmează, vom prezenta un algoritm pe baza căruia maşina singură poate să facă automat modificările necesare atunci cînd se realizează o depăşire a limitei stabilite de poziţia virgulei. Acest algoritm diferă de algoritmul folosit pentru programarea în regim cu virgulă mobilă, în primul rînd prin aceea că programul standard corespunzător acestui algoritm intră în lucru numai atunci cînd în decursul calculului se realizează o depăşire a limitei stabilite de poziţia virgulei. El acţionează numai asupra acelor constante iniţiale sau intermediare care duc la o depăşire a limitei stabilite prin poziţia virgulei şi asupra acelor constante a căror modificare este impusă de o modificare anterioară.

Acest algoritm nu este deci un algoritm de lucru în regim cu virgulă mobilă a unei maşini cu virgula fixă, ci un algoritm de corectare automată a unor constante ale problemei, după necesitate.

Algoritmul realizează următorul proces :

Presupunem că la o instrucţiune K a programului se realizează o depăşire a limitei stabilite prin poziţia virgulei. În acest caz, se vor corecta constantele respective în mod convenabil iar rezultatului operaţiei i se pune în corespondenţă ordinul modificărilor. În continuare, se analizează dacă instrucţiunea următoare depinde sau nu de modificările făcute anterior. Dacă această instrucţiune depinde de aceste modificări, rezultatului

operaţiei pe care ea o realizează i se pune în corespondenţă noul ordin. Dacă ea nu depinde de aceste modificări, se execută în mod normal de către program. În felul acesta se pune în evidenţă un şir de adrese care corespunde constantelor modificate în decursul calculului, pînă la un moment dat. Fiecare termen al acestui şir este păstrat într-o celulă a memoriei împreună cu ordinul numărului conţinut în adresa care reprezintă termenul respectiv al şirului. Menţionăm că oricare ar fi doi termeni ai şirului, ei sînt întotdeauna diferiţi şi ordinul oricărui termen este diferit de zero.

Dacă problema care urmează să fie rezolvată necesită un volum mare de celule ale memoriei, atunci se poate înscrie într-o celulă a memoriei doi termeni ai şirului respectiv.

Termenii acestui şir sînt în număr finit şi numărul lor poate fi cel mult egal cu numărul constantelor iniţiale şi intermediare care intervin în calcul.

Vom analiza în continuare schema programului acestui algoritm în cazul cînd numerele sînt reprezentate în maşină astfel ca

$$|X| < 1.$$

În acest caz depăşirea unităţii poate să apară numai ca rezultat al operaţiilor de împărţire, adunare sau scădere. Considerăm că maşina dispune de o instrucţiune de transfer care realizează trecerea automată în cursul calculului la programul standard corespunzător algoritmului descris, ori de cîte ori apare o depăşire a unităţii.

Schema programului se compune din trei blocuri pe care le vom nota cu **A**, **B**, **C**. Blocul **A** este realizat de schema operatorială

$$[k \to \mathbf{A}][k \to \mathbf{B}][k \to \mathbf{C}]\ p(c_k \neq c_+) \overset{1}{\uparrow} p(c_k \neq c_-) \overset{2}{\uparrow} \times$$
$$\times\ \mathrm{A}_k^1 p(R = 0) \overset{3}{\uparrow} \mathrm{A}_k^2\, \mathrm{A}^3 \overset{k}{\uparrow} \overset{k}{\downarrow} \mathrm{A}_k^4 \overset{4}{\downarrow} \mathrm{A}^5 \overset{5}{\downarrow} [r_k \to (r_k + h)_0^{10}] \times$$
$$\times\ [(P) \to (r_k + h)_{10}^{16}] \overset{b_1}{\uparrow} \overset{3}{\downarrow} \mathrm{A}_k^6\, \mathrm{A}^7 \overset{k}{\uparrow} \overset{k}{\downarrow} \mathrm{A}_k^8 \overset{4}{\uparrow} \overset{1}{\downarrow} \overset{2}{\downarrow} \mathrm{A}_k^9 \times$$
$$\times\ \mathrm{A}^{10} \overset{k}{\uparrow} \overset{k}{\downarrow} \mathrm{A}_k^8 \overset{5}{\uparrow}.$$

Operatorul $[k \to \mathbf{A}]$ este un operator de transfer care transmite constantele instrucţiunii k în blocul **A**. Acelaşi lucru îl realizează operatorii $[k \to \mathbf{B}]$ şi $[k \to \mathbf{C}]$ pentru blocurile **B**, respectiv **C**. Prin c_k am notat codul instrucţiunii k iar prin c_+ şi c_- am notat codul operaţiei de adunare, respectiv a operaţiei de scădere. Fie instrucţiunea k următoarea :

k	c_k	a_k	b_k	r_k

Operatorul A_k^1 este un operator de calcul care realizează produsul logic dintre numerele (b_k) şi $N = 1000 \ldots 0$. Fie R rezultatul acestei operaţii.

Operatorul A_k^2 este de asemenea un operator de calcul care realizează o deplasare la stînga a lui (b_k) cu o unitate. Presupunem că în celula P_b vom păstra ordinul numărului (b_k).

Operatorul A^3 adună o unitate la conţinutul celulei P_b care iniţial este zero.

Operatorul A_k^4 reface constanta (b_k) sub forma ei iniţială.

Operatorul A^5 este un operator de calcul care realizează diferenţa $(P_a) - (P_b)$ şi înscrie acest rezultat într-o celulă P.

Operatorul $[r_k \to (r_k + h)_0^{10}]$ este un operator de transfer care înscrie adresa r_k în primele 10 ordine ale celulei $r_k + h$, unde h este o constantă dinainte stabilită. Considerăm că instrucţiunile programului pentru rezolvarea unei probleme oarecare ocupă primele n celule ale memoriei, următoarele p celule conţin datele iniţiale şi intermediare, iar următoarele q celule sînt ocupate de programul standard. Atunci $h = p + q$. Evident că dacă M este numărul celulelor memoriei, va trebui să avem

$$M \geqq n + q + 2p.$$

Dacă într-o celulă a memoriei se înscriu doi termeni ai şirului considerat, atunci este suficient ca M să satisfacă relaţia

$$M \geqq n + q + \frac{3p}{2}.$$

Operatorul A_k^6 este un operator de calcul care realizează o deplasare la dreapta cu o unitate a numărului (a_k).

Operatorul A^7 scade o unitate din conţinutul celulei P_a, care iniţial este zero.

Operatorul A_k^8 reface constantele (a_k) şi (b_k) sub forma lor iniţială.

Operatorul A^9 deplasează pe (a_k) şi (b_k) la dreapta cu o unitate.

Operatorul A^{10} scade o unitate din conţinutul celulei P, care iniţial este zero.

În concluzie blocul A în urma depăşirii unităţii, ca rezultat al executării instrucţiunii k, realizează modificările necesare constantelor (a_k) şi (b_k), trece la executarea instrucţiunii k în program, reface constantele modificate la valoarea lor iniţială şi formează termenul corespunzător al şirului. Acest bloc predă conducerea calculului instrucţiunii b_1 din blocul B.

Blocul B este realizat de schema operatorială

$$\overset{b_1}{\downarrow}\overset{1}{\downarrow}1\overset{k}{\uparrow}\overset{k}{\downarrow}p(a_{k+1} = a_{k+1} + h)_0^{10})\overset{2}{\uparrow}p(b_{k+1} = (b_{k+1} + h)_0^{10})\overset{3}{\uparrow}\times$$

$$\times\, p(c_{k+1} \neq c_+)\overset{4}{\uparrow}p(c_{k+1} = c_-)\overset{5}{\uparrow}\overset{11}{\downarrow}\overset{8}{\downarrow}A_k^1\overset{c}{\uparrow}\overset{4}{\downarrow}\overset{5}{\downarrow}\times$$

$$\times\, [(a_{k+1} + h)_{10}^{16} \to P_a]\,[(b_{k+1} + h)_{10}^{16} \to P_b]p((P_a) = (P_b))\overset{6}{\uparrow}\times$$

$$\times\, [(P_a) \to P]\overset{c}{\uparrow}\overset{6}{\downarrow}A^2\,A_k^3\,[(P_a) \to P]\overset{k+1}{\uparrow}\overset{k+1}{\downarrow}A_k^4\overset{c_1}{\uparrow}\times$$

$$\times\overset{3}{\downarrow}p(c_{k+1} \neq c_+)\overset{7}{\uparrow}p(c_{k+1} = c_-)\overset{8}{\uparrow}\overset{7}{\downarrow}[(a_{k+1} + h)_{10}^{16} \to P_a]\,A_k^5\times$$

$$\times\overset{k+1}{\uparrow}\overset{k+1}{\downarrow}A_k^6\,[(P_a) \to P]\overset{c_1}{\uparrow}\overset{2}{\downarrow}p(b_{k+1} = (b_{k+1} + h)_0^{10})\overset{9}{\uparrow}p(c_{k+1} \neq c_+)\overset{10}{\uparrow}\times$$

$$\times\, p(c_{k+1} = c_-)\overset{11}{\uparrow}[(b_{k+1} + h)_{10}^{16} \to P_b]A_k^7\overset{k+1}{\uparrow}\overset{k+1}{\downarrow}A_k^8[(P_b) \to P]\times\overset{c_1}{\uparrow}\overset{9}{\downarrow}F(k)\overset{1}{\uparrow}.$$

În această schemă, 1 este condiţia logică identic satisfăcută.

Operatorul A_k^1 este un operator de calcul care formează în cazul operaţiilor de înmulţire sau împărţire ordinul numărului (r_{k+1}) şi îl înscrie în celula P.

Operatorul A^2 este un operator de calcul care în cazul operaţiei de adunare sau scădere egalează ordinele celor doi termeni pentru ca instrucţiunea $k + 1$ să se poată executa.

Operatorul A_k^3 modifică pe (a_{k+1}) sau (b_{k+1}) în mod corespunzător cu rezultatul operatorului A^2.

Operatorul A_k^4 reface la valoarea iniţială numerele (a_{k+1}) sau (b_{k+1}).

Operatorul A_k^5 modifică numărul (b_{k+1}) prin deplasare la dreapta sau a stînga cu un număr de ordine egal cu (P_a).

Operatorul A_k^6 reface la valoarea iniţială constanta (b_{k+1}).

Operatorul A_k^7 modifică numărul (a_{k+1}) prin deplasarea la dreapta sau la stînga cu un număr de unităţi egal cu ordinul lui (b_{k+1}).

Operatorul A_k^8 reface numărul (a_{k+1}) la valoarea lui iniţială.

Operatorul $F(k)$ este un operator de readresare care măreşte cu o unitate parametrul k în blocurile **A**, **B** şi **C**.

Blocul **B** realizează deci analiza instrucţiunii $k + 1$ din punctul de vedere al faptului că a_{k+1} şi b_{k+1} sînt sau nu termeni ai şirului considerat. În cazul în care a_{k+1} sau b_{k+1} sînt termeni ai şirului considerat, face modificările necesare pentru executarea instrucţiunii $k + 1$ şi reface constantele a_{k+1} şi b_{k+1} la valoarea lor iniţială. În cazul în care a_{k+1} şi b_{k+1} nu sînt termeni ai şirului, trece la executarea instrucţiunii $k + 1$ în program şi analizează instrucţiunea următoare. Acest bloc predă conducerea calculului instrucţiunilor c sau c_1 al blocului **C**.

Blocul **C** are schema operatorială

$$\overset{c}{\downarrow}1\overset{k+1}{\uparrow}\overset{k+1}{\downarrow}\overset{c_1}{\downarrow}\,p((P) \neq 0)\overset{b_2}{\uparrow}[r_{k+1} \to (r_{k+1} + h)_0^{10}]\times$$

$$\times\,[(P) \to (r_{k+1} + h)_{10}^{16}]F(k)\overset{b_2}{\uparrow}.$$

Acest bloc realizează deci un termen al şirului în cazul cînd acesta nu provine dintr-o depăşire a unităţii, ci dintr-o modificare făcută de blocul **B**.

АВТОМАТИЧЕСКАЯ КОРРЕКЦИЯ ПРЕВЫШЕНИЯ ЕДИНИЦЫ В МАШИНЕ С ФИКСИРОВАННОЙ ЗАПЯТОЙ

РЕЗЮМЕ

В работе предлагается алгоритм для автоматической коррекции превышения единицы в машине с фиксированной запятой. Схема программы алгоритма записана с помощью операторов и состоит из описываемых в работе блоков **A**, **B** и **C**.

CORRECTION AUTOMATIQUE DU DÉPASSEMENT DE L'UNITÉ DANS UNE MACHINE À VIRGULE FIXE

RÉSUMÉ

Les auteurs présentent un algorithme pour la correction automatique du dépassement de l'unité dans une machine à virgule fixe. Le schéma du programme de cet algorithme est établi à l'aide d'opérateurs, et se compose des blocs **A**, **B** et **C**, décrits dans le travail.

BIBLIOGRAFIE

1. Mc Cracken D. D., *Digital Computer Programing*. John Wiley a. Sons, Inc., New York, 1957.
2. Leapunov A. A., *Despre schemele logice de programe*. Probleme de cibernetică, Biblioteca Analelor Rom.-Sov., Ser. tehnică, **69—70** (1959).
3. Шура—Бура М. Р., *Система стандартных подпрограмм*, Физматгиз, Москва, 1958.

Primit la 14. III. 1961.