B.V. GNEDENKO
V.S. KOROLIOUK
E.L. IOUCHTCHENKO

éléments de programmation sur ordinateurs

ÉLÉMENTS
DE
PROGRAMMATION
SUR ORDINATEURS

B 10 e

ÉLÉMENTS DE PROGRAMMATION SUR ORDINATEURS

PAR

B. V. GNEDENKO V. S. KOROLIOUK E. L. IOUCHTCHENKO

TRADUIT PAR

Anita GAGNY
Traductrice au C. N. R. S.

AVEC L'AUTORISATION DE

Monsieur le Professeur B. VAUQUOIS
Directeur du Centre d'Etudes pour la Traduction automatique
Grenoble

DUNOD
PARIS
1969

Traduction de l'ouvrage publié en
langue russe sous le titre

ЭЛЕМЕИТЫ ПРОГРАММИРОВАНИЯ

par les ÉDITIONS D'ÉTAT DES PUBLICATIONS
PHYSICO-MATHÉMATIQUES, Moscou

NOTE DES AUTEURS

Cet ouvrage peut servir de manuel de programmation sur calculatrices automatiques. Il reflète les recherches effectuées dans le domaine de la programmation automatique du traitement des problèmes logiques sur les calculatrices ainsi que la méthode opératorielle proposée par A. A. Liapounov.

Il est destiné aux étudiants des Universités, des Etablissements d'Enseignement Technique Supérieur, et peut aussi être utile aux chercheurs des Instituts de Recherche Scientifique qui s'occupent des problèmes de programmation.

On peut l'utiliser comme manuel d'enseignement pour former des programmeurs.

AVANT-PROPOS

Ce livre est fondé sur l'expérience acquise par les auteurs au cours de leur travail sur les calculatrices automatiques universelles, de l'étude des cours de programmation, de l'organisation de séminaires correspondants, de la direction de travaux pratiques pour les étudiants et de la participation au projet de nouvelles calculatrices. Le point de départ de notre travail a été le cours du professeur A. A. Liapounov à l'Université de Moscou. Nous n'avons pas besoin de nous arrêter sur le contenu de notre livre, car la table des matières le rend assez explicite.

Nous serons sincèrement reconnaissants envers les lecteurs qui, éventuellement, nous enverront leurs suggestions, remarques et indications sur les défauts qu'ils auront pu noter.

TABLE DES MATIÈRES

INTRODUCTION

Il y a relativement peu d'années que fut construite la première calculatrice électronique automatique, mais l'on peut déjà affirmer que pour un grand nombre de problèmes scientifiques et techniques contemporains son apparition marque un bond décisif sur la route du progrès.

Pour la première fois s'est ouverte devant la science la possibilité de faire des calculs de grande envergure en un laps de temps extrêmement court. Ce sont les nécessités pratiques qui imposent la production des calculs et, chaque année, les besoins augmentent.

Tout perfectionnement scientifique et technique, toute construction — d'un navire, d'un barrage, d'une turbine, d'un avion, d'une rampe de lancement, d'un réacteur atomique — nécessitent des calculs préalables souvent très compliqués.

Parfois, la seule difficulté, pour obtenir le résultat final, est d'effectuer des millions, ou même des milliards, d'opérations arithmétiques. A titre d'exemple, nous attirerons l'attention sur la résolution des systèmes d'un grand nombre d'opérations algébriques linéaires. Depuis longtemps déjà, on en connaît assez bien le principe, toutefois la seule difficulté est d'effectuer un grand nombre d'additions, de soustractions, de multiplications et de divisions. Pour traiter ce genre de systèmes qui contiennent deux à trois cents inconnues et un nombre correspondant d'équations, il faut des dizaines de milliers d'opérations arithmétiques élémentaires.

Même le meilleur calculateur, qui aurait entre les mains les moyens de calcul les plus simples — machine arithmétique, tables, etc. — passerait des années, voire des dizaines d'années, de travail intense pour résoudre un seul et unique problème d'un type aussi simple. Parfois le problème admet une décomposition en parties que l'on peut résoudre indépendamment les unes des autres. Dans ce cas, en confiant l'exécution des groupes d'opérations indépendants à différents calculateurs, on peut considérablement accélérer la marche du calcul. Cependant une telle division est souvent impossible et le problème entier exige qu'on effectue successivement les opérations.

Il n'y a pas longtemps encore les difficultés que présentaient les calculs étaient telles qu'il paraissait absolument impossible d'exploiter dans la pratique la résolution formelle de nombreux problèmes ; de nos jours, l'apparition de la nouvelle technique de calcul rapide permet d'aller jusqu'aux calculs numériques et, par conséquent, jusqu'aux applications directes.

Dans la technique, les difficultés que présentent les calculs conduisent souvent les ingénieurs à se borner à des supputations et parfois même à se refuser

à faire des calculs, alors que le phénomène correspondant repose sur une théorie satisfaisante. Il en est résulté, et il en résulte en particulier dans la pratique, qu'ils acceptent ces obstacles dont le nombre est excessif sans justification. Dans les cas vraiment importants, ils ont recours à des transformations et à des expériences coûteuses. L'utilisation des calculatrices offre des possibilités extraordinaires à la technique contemporaine. Il suffit de dire que les meilleurs modèles de machines sont actuellement capables d'effectuer à la seconde des dizaines de millions de multiplications de nombres à plusieurs chiffres. Autrement dit, la machine effectue en une seconde plus de calculs que peut en faire un calculateur même expérimenté pendant plusieurs mois. Ces circonstances ne peuvent qu'exercer une profonde influence sur l'ensemble des sciences techniques, économiques et naturelles, en stimulant un large emploi des méthodes de calcul. L'adoption de ces méthodes permet de choisir la solution optimale des problèmes que se posent les ingénieurs, et par conséquent elle a des effets considérables sur l'économie.

Quelle que soit l'importance de la nouvelle technique de calcul pour l'exécution des travaux numériques, la possibilité d'adapter les calculatrices au traitement des problèmes logiques est peut-être encore plus importante. La réussite des essais de traduction automatique que l'on a faits en U. R. S. S., aux U. S. A. et dans d'autres pays, en est un exemple. On s'est rendu compte que l'on pouvait reporter sur la machine certaines fonctions de l'activité intellectuelle et par conséquent libérer l'homme des opérations mentales monotones qui sont effectuées selon des règles déterminées. L'utilisation des calculatrices pour la gestion automatique des procédés de production ouvre des perspectives particulièrement larges. Un grand nombre de savants et d'ingénieurs font de la recherche dans ce sens. Afin de faire apparaître avec assez de netteté la nature de ce genre d'application, nous en donnerons des exemples empruntés aux publications techniques.

La « Sokony Mobil Oil Company » a utilisé la calculatrice électronique pour choisir le régime de distillation du pétrole le plus avantageux. Les données initiales étaient : le coût sur le marché des dérivés du pétrole, de la main-d'œuvre et des différents matériaux, ainsi qu'une série d'autres facteurs. Pour établir le programme de cracking catalytique, d'alkylation, etc., on a préalablement décrit mathématiquement les processus. Le programme qui représente le processus de distillation renferme 6 000 instructions et nombres ; la machine l'exécute en 9 minutes, dont 6 sont perdues pour l'entrée du programme en machine.

La « West Penn Electric » créa une calculatrice électronique pour choisir le régime de travail le plus économique d'un système énergétique d'une puissance de 1 300 mégawatts, rassemblant dix centrales électriques thermiques et une centrale hydraulique. La longueur totale des lignes de transmission est de 2 500 kilomètres pour une tension de 66 et de 132 kilovolts.

La machine sert à calculer le régime optimal, compte tenu des pertes dans le réseau, du coût du combustible pour les stations séparées, des coefficients

d'utilisation des générateurs fournissant l'énergie nécessaire à la charge et d'autres facteurs qui déterminent le coût de l'énergie. Le résultat obtenu est utilisé pour distribuer l'énergie entre les stations.

La presse a récemment publié un nombre considérable de travaux concernant l'application des calculatrices à la commande automatique des découpeuses à métaux et des processus de fusion, aux problèmes de planification nationale, aux problèmes de documentation automatique, etc.

Les rapports, présentés à la session de l'Académie des Sciences de l'U. R. S. S. consacrée aux problèmes scientifiques d'automation industrielle, ont traité de certaines des applications importantes et intéressantes exposées ci-dessus [1].

Il est indiscutable que la recherche, dans ce sens, ne fait que commencer et connaîtra dans un avenir très proche des succès immenses. Mais pour y arriver, il faut entreprendre d'une façon opiniâtre et systématique un travail sur l'étude et la description mathématique des processus d'automation. Jusqu'à un certain degré, ce genre de recherches représente une nouvelle branche des mathématiques, qui exige une formation scientifique et logique toute spéciale. Il est évident que, dès maintenant, l'organisation de l'enseignement universitaire doit en tenir compte.

Jusqu'à ces derniers temps, l'humanité perdait une grande part de son effort créateur en cherchant à mettre à son service les forces de la nature et à utiliser diverses machines afin de libérer l'homme du travail physique, monotone et fatigant. En même temps, mais à un degré moins élevé, elle visait à alléger le travail intellectuel en en rejetant une partie sur différents appareils. Au cours des siècles, elle a atteint ce but en de nombreux points.

Pour s'en convaincre, il suffit de se rappeler l'invention de l'imprimerie, de la photographie, de l'enregistrement du son, des machines et des appareils à compter et de beaucoup d'autres...

Ces inventions ont reporté sur les machines ce qu'auparavant seule l'intelligence humaine était capable d'exécuter : accumuler et transmettre aux autres l'expérience et la connaissance, retenir l'image et le son, effectuer des calculs, etc.

A l'heure actuelle, nous nous trouvons au seuil d'une ère nouvelle où les forces de la nature seront exploitées à leur tour pour augmenter la puissance intellectuelle de l'humanité et pour libérer l'homme d'un travail intellectuel monotone et fatigant, ce qui permettra de concentrer toutes les possibilités créatrices sur des problèmes non encore résolus, sur le développement de nouvelles branches scientifiques.

Les calculatrices électroniques marquent un pas important dans cette voie ; les premiers essais de leur utilisation dans ce sens ont été très concluants. Les possibilités ultérieures nous paraissent littéralement inépuisables.

La révolution apportée par les calculatrices se fait sentir actuellement aussi bien sur la transformation des appareils techniques que sur la création de nouveaux liens entre les disciplines scientifiques qui, il y a très peu de temps, étaient encore très éloignées les unes des autres ; d'autre part, elle a stimulé

l'essor rapide des nouvelles disciplines scientifiques. En premier lieu, il faut citer la cybernétique que l'on peut définir comme la science des moyens de perception, de conservation, de traitement et d'emploi de l'information des machines et des organismes vivants. La création de modèles de certains processus physiologiques et mentaux peut apporter une aide importante à la physiologie ainsi qu'à la technique, par exemple dans la construction des machines de commande. La cybernétique est appelée à réunir les efforts des mathématiciens, biologistes, physiciens, techniciens, psychologues, économistes et linguistes et à favoriser le développement réciproque des disciplines respectives.

Il est évident que les calculatrices ont exercé une grande influence sur certaines branches des mathématiques. Tout d'abord, le premier problème qui a surgi était de préparer des problèmes mathématiques afin qu'ils puissent être traités en machine. Il ne s'agit pas seulement de réduire le problème qui nous intéresse à la résolution de telle ou telle équation ou à une formule déjà toute prête, dans laquelle il n'y ait qu'à donner des valeurs numériques aux inconnues et à effectuer les opérations nécessaires. Pour que, sans intervention humaine, la machine puisse accomplir tout le chemin qui va des conditions initiales du problème à la sortie des résultats numériques et à l'interruption des calculs, il est indispensable de décomposer préalablement le processus de résolution du problème en opérations élémentaires et, par là même, de créer les conditions dans lesquelles elle puisse exécuter automatiquement toutes les opérations nécessaires dans un ordre voulu, pas à pas. Il est très important ici de souligner le mot « automatiquement », car toute intervention humaine retarde énormément l'exécution des calculs. En effet, s'il ne faut à la machine qu'un dix millième et même un cent millième de seconde pour effectuer chaque opération, l'homme, lui, a déjà besoin de minutes, ne serait-ce que pour changer la marche des calculs. Aussi est-il nécessaire que la machine non seulement effectue des additions et des soustractions, mais que le passage d'une opération à la suivante, la « mémorisation » du résultat des calculs intermédiaires, le choix des nombres dans la mémoire, le changement de la nature des calculs, la sortie des résultats finals, et l'interruption opportune du travail, se fassent automatiquement. C'est ce qui s'effectue à l'aide de l'organe de commande à programme enregistré. La nouvelle discipline mathématique qui permet de faire la décomposition indispensable du problème en opérations élémentaires, c'est-à-dire d'établir le programme sur lequel la machine doit travailler, a reçu le nom de théorie de la programmation.

Notre but dans ce livre est d'exposer les idées fondamentales de cette discipline, en nous basant sur des exemples extrêmement simples et aussi sur d'autres plus compliqués. Nous avons en grande partie emprunté ces exemples à la pratique, nous les avons rencontrés en résolvant des problèmes concrets sur des calculatrices. Outre l'exposition des principes de la programmation indispensables à tout programmeur, nous jugeons indispensable de mettre le lecteur au courant des idées de programmation qui sont mises au point actuellement.

En particulier, nous accorderons une grande importance à la méthode de programmation opératorielle proposée par A. A. Liapounov.

Il convient de remarquer qu'à la base des processus de calcul en machine il y a la méthode de la solution par approximation : remplacement d'une intégrale par une somme intégrale, transformation d'une équation différentielle en un système d'équations successives aux différences finies, etc. Tout l'arsenal des méthodes d'analyse par approximation dont on avait l'habitude est actuellement mis en jeu. De plus, on s'est aperçu que quelques méthodes seulement peuvent s'appliquer au calcul automatique. Il faut améliorer et développer les méthodes de calcul approximatif, élaborées autrefois, pour pouvoir les appliquer au travail en machine. Il faut aussi se souvenir des anciens moyens de calcul approximatif depuis longtemps oubliés ; il peut arriver que certains d'entre eux conviennent. Il est, naturellement, indispensable de créer aussi de nouvelles méthodes d'approximation ; il est nécessaire de pousser ces recherches, ne fût-ce que pour la seule raison que les façons habituelles d'aborder une série de problèmes importants conduisent à des calculs qui exigent un travail excessif même pour les machines modernes. Ceci vaut en particulier pour les problèmes multidimensionnels de la physique mathématique. On sait, par exemple, que, lorsqu'on met sous forme d'équations successives les problèmes liés au calcul des chaudières atomiques hétérogènes, on peut avoir à étudier des réseaux contenant des millions de nœuds. Le nombre total des opérations élémentaires qui doivent alors être effectuées atteint des dizaines de milliards. Des problèmes presque jamais rencontrés jusqu'à présent se posent lorsqu'on étudie les méthodes d'approximation. On peut, par exemple, citer la recherche de la stabilité du calcul.

A l'heure actuelle, on n'a pas encore établi de terminologie définitive pour désigner les calculatrices contemporaines qui sont commandées par programme. On les a d'abord appelées calculatrices électroniques, pour souligner que leurs éléments primordiaux étaient des systèmes électroniques. Mais, puisqu'ils sont remplacés actuellement par d'autres éléments, il semble normal d'appuyer non sur le fait que les calculatrices sont des machines analogiques, mais qu'elles donnent des résultats numériques. On a alors proposé comme nom Calculatrice Numérique Automatique Rapide (A. B. T. S. V. M.), cf. A. A. Liapounov et G. A. Chestopal [2]. Nous proposons, nous, l'appellation plus courte de Calculatrice Automatique Numérique (C. A. N.), d'autant plus que la notion de rapidité de la machine est sujette à de nombreux changements. Ainsi les 30 000 à 100 000 opérations à la seconde, qui sont estimées aujourd'hui presque comme la vitesse-record de la machine, ne paraîtront plus une réponse rapide, mais sa vitesse normale, lorsque les machines effectueront de cent mille jusqu'à un million d'opérations à la seconde. C'est pourquoi le nom de la machine évoquera ses deux qualités fondamentales : elles est numérique et elle agit automatiquement.

CHAPITRE I

PRINCIPES DE CONSTRUCTION DES CALCULATRICES ÉLECTRONIQUES

Ce chapitre traite des principes arithmétiques et logiques de fonctionnement des calculatrices. Il existe des publications qui permettent aux lecteurs de se familiariser avec les particularités techniques de la matérialisation de ces principes et avec les idées relatives à la conception des calculatrices (C. A. N.).

Par ailleurs, il convient de remarquer qu'actuellement on n'a pas encore créé une théorie mathématique de construction des calculatrices C. A. N., si bien que l'une des tâches primordiales des chercheurs et des techniciens contemporains est de développer la voie de synthèse des circuits électroniques de calcul et de commande.

1. CHOIX D'UN SYSTÈME DE NUMÉRATION

Pour représenter un nombre on emploie actuellement un principe d'écriture *positionnel* très perfectionné, d'après lequel un même symbole numérique (non chiffré) possède différentes significations selon la place qu'il occupe. Ce système d'écriture des nombres est fondé sur le fait qu'un certain nombre d'unités, par exemple c, s'unissent en une seule unité de seconde position ; l'union c des unités de seconde position constitue l'unité de troisième position etc. Le nombre c porte le nom de *base de numération.*

On peut prendre comme base de numération n'importe quel nombre entier supérieur à 1.

Dans un système de numération à base c, tout nombre a s'écrit sous la forme :

$$a = \pm \sum_{k=-\infty}^{p-1} \alpha_k c^k \qquad (1)$$

où les grandeurs α_k prennent les valeurs 0, 1, 2, ..., $c - 1$, et p est un nombre entier fonction de c et de a. Si $\beta_n = \alpha_{p-n}$, le nombre a peut s'écrire sous une

forme un peu différente de (1) :

$$a = \pm c^p \sum_{n=1}^{\infty} \beta_n c^{-n}.$$

Le nombre p s'appelle *l'ordre* du nombre a (il est entendu que la valeur de p dépend du système de numération adopté). L'ordre du nombre détermine la place de la virgule dans l'écriture du nombre, à savoir : la virgule doit se trouver devant β_{p+1}. Le multiplicateur $\sum_{n=1}^{\infty} \beta_n c^{-n}$ est toujours inclus dans le demi-intervalle [0, 1) ; il porte le nom de *mantisse du nombre A*.

Dans la vie courante, le système le plus employé est le système décimal, dans lequel on prend pour base le nombre 10. Chacun sait que tout nombre peut alors être écrit à l'aide de dix signes numériques rangés dans cet ordre : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Par exemple, le nombre 429,538 est l'écriture abrégée de l'expression :

$$4.10^2 + 2.10^1 + 9.10^0 + 5.10^{-1} + 3.10^{-2} + 8.10^{-3}.$$

Nous emploierons par la suite des écritures abrégées semblables pour représenter des nombres écrits dans d'autres systèmes de numération non décimale. Ainsi, par exemple, dans le système octal l'écriture 321,57 signifie que l'on examine le nombre :

$$3.8^2 + 2.8^1 + 1.8^0 + 5.8^{-1} + 7.8^{-2}.$$

Il arrive que, pour certains problèmes théoriques et pratiques, les systèmes non décimaux présentent des avantages. Ainsi, par exemple, le système binaire est utilisé dans la plupart des C. A. N. actuelles pour la simplicité avec laquelle on effectue techniquement les opérations sur les nombres binaires, et pour les possibilités que ce système offre d'utiliser la logique mathématique (en particulier le calcul des propositions). Dans la pratique, lorsque l'on travaille sur une C. A. N., on applique aussi les autres systèmes de numération, ceux à base 8, à base 16, etc.

Notons que pour représenter la même gamme de nombres, le système binaire a besoin d'un moins grand nombre d'éléments que le système décimal. En effet, la quantité des nombres ayant n positions est égale, dans le système à base c, à $M = c^n$. Un nombre d'éléments proportionnels à $N_c = c.n$ est nécessaire pour représenter ces nombres. Fixons le nombre M et cherchons c pour lequel N_c atteint son minimum. A partir de la première égalité écrite nous trouvons :

$$n = \frac{\ln M}{\ln c},$$

d'où, en reportant cette valeur dans $N_c = cn$, on obtient

$$N_c = \frac{c \ln M}{\ln c}.$$

Il est facile de trouver que le minimum de cette expression est atteint pour $c = e = 2{,}718\,28\ldots$. De ce point de vue, le système de numération à base 3 serait le meilleur. Pour représenter tous les nombres de 1 à 10^6, dans le système décimal il faut 60 éléments, dans le binaire, 40, dans le ternaire, 38. Cependant, l'accroissement du nombre des éléments dans l'écriture binaire n'est pas grand, comparé à celui des éléments de l'écriture ternaire. Si N_2 désigne le nombre des éléments nécessaire à l'écriture en binaire, et N_3 à celle en ternaire, alors :

$$\frac{N_2}{N_3} = \frac{2 \ln 3}{3 \ln 2} = \frac{2 \log_{10} 3}{3 \log_{10} 2} \approx 1{,}056.$$

Le système ternaire n'a pas été largement appliqué dans les calculatrices numériques, car il est difficile de construire des éléments rapides à trois positions stables suffisamment solides, alors qu'il existe un grand nombre d'appareils de physique qui possèdent deux positions stables différentes avec des schémas de combinaisons déterminés : tels sont les condensateurs, les triodes, etc. Les circuits des éléments importants qui composent les calculatrices électroniques sont par essence binaires. Un condensateur est ou n'est pas chargé, la lampe laisse ou ne laisse pas passer le courant. De plus, le système binaire est préférable aux autres grâce à l'extrême simplicité avec laquelle l'on effectue les opérations arithmétiques sur les nombres binaires.

La nécessité de convertir les données initiales du système décimal en binaire et, inversement, les résultats du calcul de binaire en décimal est un inconvénient du système binaire dans les C. A. N. (comme d'ailleurs pour tout autre système de numération différent du système décimal), mais cette conversion est nécessaire car le système décimal est adopté universellement. Cependant, pour une majorité écrasante de problèmes, le volume des opérations numériques dépasse la quantité des opérations de conversion des données et des résultats ; de ce fait le défaut signalé n'a aucune importance. De plus, cette conversion s'opère souvent automatiquement.

Craignant que le lecteur n'ait pas l'habitude des systèmes de numération non décimal, nous donnerons pour l'orienter un petit tableau de la façon d'écrire les nombres dans les différents systèmes.

En outre, nous signalons que, vu que dans la dernière colonne nous avons à écrire les nombres du système à base 16 et que dans ce système on doit avoir des désignations pour 16 chiffres, nous utilisons pour désigner 0 et les 9 premières unités les signes habituels, mais pour 10, 11, 12, 13, 14 et 15, respectivement les signes $\overline{0}$, $\overline{1}$, $\overline{2}$, $\overline{3}$, $\overline{4}$, $\overline{5}$.

Tableau 1. — *Ecriture des nombres dans les différents systèmes.*

Système de numération					
décimal	binaire	ternaire	à base 5	octal	à base 16
1	1	1	1	1	1
2	10	2	2	2	2
3	11	10	3	3	3
4	100	11	4	4	4
5	101	12	10	5	5
6	110	20	11	6	6
7	111	21	12	7	7
8	1000	22	13	10	8
9	1001	100	14	11	9
10	1010	101	20	12	$\overline{0}$
11	1011	102	21	13	$\overline{1}$
12	1100	110	22	14	$\overline{2}$
13	1101	111	23	15	$\overline{3}$
14	1110	112	24	16	$\overline{4}$
15	1111	120	30	17	$\overline{5}$
16	10000	121	31	20	10
17	10001	122	32	21	11
0,5	0,1	0,11...	0,22...	0,4	0,8
0,3	0,01001...	0,02200...	0,122...	0,23146...	0,4$\overline{2}\overline{2}\overline{2}$...
0,(3)	0,0101...	0,1	0,1313...	0,2525...	0,55...

Pour la préparation des problèmes, on utilise souvent les systèmes à base 8 et à base 16. Le passage de ces systèmes au système binaire se fait très facilement. En effet, chaque position du système octal se transforme en un nombre binaire à trois chiffres. Ainsi, par exemple, le nombre écrit dans le système octal 1175, dans le système binaire aura la forme : 1001111101, le nombre 5$\overline{2}$8 (numération de base 16) devient en binaire : 10111001000. Pour passer de l'écriture binaire d'un nombre à l'écriture octale, il faut diviser l'écriture binaire en groupes de trois chiffres de la droite à la gauche et remplacer chaque groupe par le nombre octal, en utilisant le tableau 1. Pour passer de l'écriture binaire à celle à base 16, il convient de diviser l'écriture binaire de droite à gauche en groupes de quatre chiffres et de remplacer chaque groupe par le chiffre à base 16 correspondant. Par exemple, le nombre écrit en binaire 11101111001 sera mis en octal sous la forme : 3571. Pour convertir ce même nombre dans le système à base 16 il faut auparavant le noter par groupe de quatre chiffres 111 0111 1001 ; d'après le tableau 1 nous trouvons que, dans l'écriture à base 16, ce nombre aura la forme : 779.

Ce n'est pas indispensable d'analyser en détail que la simplicité des transformations indiquées tient à ce que les nombres 8 et 16 sont des multiples de 2 ($8 = 2^3$; $16 = 2^4$).

Pour que le système binaire soit mieux assimilable, nous formulerons succinctement sur des exemples les règles d'exécution des opérations arithmétiques sur des nombres écrits en binaire. Ces règles ne diffèrent en rien de celles qui sont exposées dans le cours d'arithmétique scolaire, si ce n'est que dans le système binaire, les tables d'addition et de multiplication sont particulièrement simples.

Table d'addition binaire

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10$

Table de multiplication binaire

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

Prenons un exemple de chaque opération arithmétique sur des nombres binaires :

Addition binaire

$$\begin{array}{r} 1010011{,}111 \\ +\ \ 11001{,}110 \\ \hline 1101101{,}101 \end{array}$$

Soustraction binaire

$$\begin{array}{r} 1100101{,}101 \\ -\ \ 10101{,}111 \\ \hline 1001111{,}110 \end{array}$$

Multiplication binaire

$$\begin{array}{r} 11001{,}01 \\ \times\ \ 11{,}01 \\ \hline 1100101 \\ 1100101 \\ 1100101 \\ \hline 1010010{,}0001 \end{array}$$

Division binaire

$$\begin{array}{r|l} 10100110011 & 1011 \\ \cline{2-2} -\ 1011 & 1111001 \\ \hline -\ 10011 & \\ \ \ 1011 & \\ \hline -\ 10001 & \\ \ \ 1011 & \\ \hline -\ 01100 & \\ \ \ 1011 & \\ \hline -\ 0001011 & \\ \ \ 1011 & \\ \hline \end{array}$$

2. OPÉRATIONS LOGIQUES FONDAMENTALES

L'utilisation du système binaire dans les calculatrices électroniques permet d'utiliser l'appareil de la logique mathématique, en particulier le calcul des propositions, pour analyser et construire les circuits fonctionnels des machines.

En logique mathématique on entend par *proposition* n'importe quelle proposition de laquelle on puisse dire si elle est vraie ou fausse, par exemple : « *Aujourd'hui, nous sommes le* 4 », « *L'homme est immortel* », « *Deux et deux font quatre* ».

Les propositions qui peuvent être à la fois vraies et fausses, ainsi que celles qui ne sont vraies que partiellement, ne sont pas examinées en logique mathématique. Pour évaluer les propositions, nous ne prendrons en considération que leur valeur vraie ou fausse, sans tenir compte de leur contenu concret. On désignera les propositions par les majuscules de l'alphabet latin. Si A est vrai, on écrira $A = 1$, s'il est faux, $A = 0$. Si la proposition dépend d'un paramètre, sa valeur vraie représente une fonction susceptible de ne prendre que deux valeurs : 0 et 1. Une proposition toujours vraie détermine une fonction constante égale à 1. La fonction qui, d'une façon identique, est égale à 0 correspond à une proposition toujours fausse. Dans les exemples cités ci-dessus, la première proposition n'est vraie que pour douze jours de l'année, la seconde est toujours fausse et la troisième est toujours vraie.

Une variable qui n'est susceptible de prendre que deux valeurs s'appelle une *variable binaire*. Conformément à la définition qu'on en a donnée, toute proposition est une variable binaire.

Dans les C. A. N. actuelles les opérations sur les nombres (écrits en binaire), le transfert des codes binaires d'un bloc de la machine dans un autre, la commande de ce transfert sont réalisés au moyen d'organes qui ne peuvent prendre que deux positions (on attribue à ces positions les valeurs 0 et 1). Ainsi, le schéma des circuits de contrôle de l'organe de transfert de l'information pour les C. A. N. est représenté par des fonctions à plusieurs variables, qui elles-mêmes ne prennent que deux valeurs. On les appelle *fonctions binaires*.

La logique mathématique étudie comment représenter et transformer les fonctions binaires de deux arguments binaires au moyen d'opérations logiques appelées relations logiques. A partir de propositions simples, au moyen de relations logiques on peut composer des propositions complexes ayant la valeur « *vrai* » (1) ou « *faux* » (0) par rapport à la valeur des propositions simples composantes. Puisqu'il est possible de considérer les propositions comme des variables binaires, on peut concevoir les relations logiques entre les propositions comme des opérations sur des variables binaires. Nous allons définir les principales opérations logiques.

1. Négation : *La négation* de la proposition A est désignée par le symbole $\overline{A}$ (qui se lit « *non A* »). La valeur « vrai » de la proposition $\overline{A}$ est définie

par le tableau suivant :

Négation

$$\overline{1} = 0$$

$$\overline{0} = 1.$$

Ainsi, la négation de la proposition A est la proposition complexe $\bar{A}$ qui est fausse lorsque A est vraie et vraie lorsque A est fausse.

2. Multiplication logique : La multiplication logique des propositions A et B est désignée par le symbole $A \wedge B$ (qui se lit : « A et B »). La valeur « vrai » du produit logique $A \wedge B$ se définit par rapport aux valeurs « vrai » des propositions A et B selon le tableau suivant :

Multiplication logique

$$0 \wedge 0 = 0$$

$$0 \wedge 1 = 0$$

$$1 \wedge 0 = 0$$

$$1 \wedge 1 = 1.$$

La multiplication logique est appelée *intersection* et le symbole $\wedge$ *signe d'intersection.*

L'intersection $A \wedge B$ de deux propositions représente une proposition complexe qui n'est vraie que si, et seulement si, les propositions A et B sont vraies.

3. Addition logique : L'addition logique de deux propositions A et B est désignée par le symbole $A \vee B$ (qui se lit A ou B). La valeur « vrai » de la somme logique $A \vee B$ se définit par rapport aux valeurs « vrai » des propositions composantes A et B de la façon suivante :

Addition logique

$$0 \vee 0 = 0$$

$$0 \vee 1 = 1$$

$$1 \vee 0 = 1$$

$$1 \vee 1 = 1.$$

L'opération d'addition logique est appelée *union* et le symbole $\vee$ *signe d'union*. L'union de deux propositions A et B est une proposition complexe qui est fausse si, et seulement si, les deux termes A et B sont faux.

4. L'équivalence : L'équivalence s'exprime par le symbole $A \sim B$ (qui se lit : « *A est équivalent à B* »). La valeur « vrai » de la proposition $A \sim B$ est définie par le tableau suivant :

Equivalence

$$0 \sim 0 = 1$$

$$0 \sim 1 = 0$$

$$1 \sim 0 = 0$$

$$1 \sim 1 = 1$$

L'équivalence $A \sim B$ de deux propositions A et B est vraie lorsque les deux propositions A et B ont une valeur identique, elle est fausse dans le cas contraire.

5. Somme modulo 2 : Dans les calculatrices électroniques on utilise souvent l'opération sur deux variables binaires qui se définit comme la négation de l'équivalence et est désignée par le symbole $A \approx B$ (que l'on peut lire : « *A n'est pas équivalent à B* »). La valeur « vrai » de la proposition $A \approx B$ se définit selon le tableau de la somme modulo 2 : effectuée position par position :

$$0 \approx 0 = 0$$

$$0 \approx 1 = 1$$

$$1 \approx 0 = 0$$

$$1 \approx 1 = 0\,.$$

L'addition modulo 2 est exprimée par les opérations d'équivalence et de négation selon la formule :

$$A \approx B = \overline{A \sim B}\,.$$

Les opérations de multiplication logique et d'addition logique citées ci-dessus, l'opération d'équivalence et la somme modulo 2 sont commutatives et associatives, de sorte que :

$$A \odot B = B \odot A$$

$$(A \odot B) \odot C = A \odot (B \odot C)$$

$\odot$ étant le signe d'opération.

En outre, la loi distributive qui concerne la multiplication logique par rapport à l'addition et l'addition par rapport à la multiplication est valable ; on a les formules :

$$A \wedge (B \vee C) = (A \vee B) \wedge (A \vee C)$$

$$A \vee (B \wedge C) = (A \wedge B) \vee (A \wedge C) .$$

Les propriétés de la commutativité et de l'associativité des opérations binaires et la loi de distribution qui concerne la multiplication et la somme logiques sont vérifiées en substituant aux valeurs A, B et C les valeurs 0 ou 1 et en appliquant les définitions des opérations écrites dans les tableaux.

Les liens logiques, énumérés ci-dessus, ne sont pas indépendants. Par exemple, entre les intersections, les unions et les négations, il y a les liaisons :

$$\overline{A \wedge B} = \overline{A} \vee \overline{B}$$

$$\overline{A \vee B} = \overline{A} \wedge \overline{B} .$$

On peut représenter l'équivalence « $\sim$ » et la somme modulo 2 (la non-équivalence) « $\simeq$ » à l'aide des opérations « $-$ » (*non*), « $\wedge$ » (*et*), « $\vee$ » (*ou*). On laisse le soin au lecteur de s'assurer par une vérification directe que les équivalences suivantes sont valables :

$$A \sim B = (A \vee \overline{B}) \wedge (\overline{A} \vee B)$$

$$A \simeq B = (A \wedge \overline{B}) \vee (\overline{A} \wedge B) .$$

Dans le calcul des propositions, on démontre que toute proposition complexe peut être construite à partir de propositions simples à l'aide de la négation, de la multiplication et de l'addition logiques. De surcroît, il existe ce que l'on appelle la *forme normale* des propositions complexes. Le théorème fondamental de la théorie des propositions affirme que :

Toute proposition peut être réduite à une forme normale qui se compose d'une intersection d'unions dans laquelle chaque membre d'une union est soit une proposition simple fondamentale, soit sa négation.

L'expression :

$$(A \vee \overline{B} \vee C) \wedge (\overline{A} \vee \overline{B} \vee C)$$

peut servir d'exemple de proposition complexe à forme normale.

En nous bornant à ces indications sur la logique mathématique, nous ferons remarquer que la connaissance des concepts et des propositions de la logique mathématique est indispensable pour établir les schémas de principe des C. A. N. et pour travailler sur elles.

3. CIRCUITS ÉLECTRONIQUES ÉLÉMENTAIRES

Les relations logiques des variables binaires sont réalisées dans les C. A. N. par des circuits électroniques dont les éléments fondamentaux sont des triodes et des diodes sous forme de lampes électroniques ou d'éléments semiconducteurs. Une variable binaire est réalisée en machine sous forme d'un potentiel ou d'un courant électrique. Le chiffre 1 est représenté par le signal de haute tension, le chiffre 0 par celui de basse tension.

Les schémas électroniques des relations logiques assurent à la sortie du circuit l'émission d'un signal dépendant des signaux d'entrée, conformément au tableau de la relation logique correspondante.

Dans les circuits électroniques les signaux d'entrée et de sortie peuvent être réalisés par des tensions ou des impulsions soit positives, soit négatives. Lorsqu'on étudie les circuits des relations logiques, le potentiel élevé représente la présence du signal, le potentiel bas son absence.

Examinons les circuits électroniques les plus simples, qui réalisent les opérations logiques fondamentales « *non* », « *ou* » et « *et* ».

1. Circuit de la négation. Inverseur.

Le circuit de la négation (Fig. 1) est composé d'une lampe électronique (triode) L et d'une résistance R. S'il n'y a pas de signal à l'entrée A, c'est-à-dire si la tension est faible sur la grille de la lampe, la résistance de la lampe sera grande, donc le potentiel de la sortie C sera élevé, c'est-à-dire qu'à la sortie C il y aura un signal. Lorsque l'entrée A reçoit le signal, la tension s'élève sur la grille de la lampe, la résistance de la lampe tombe brusquement, donc le

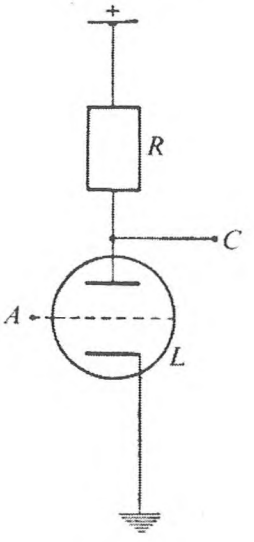


Fig. 1.

potentiel de la sortie C baissera, c'est-à-dire qu'il n'y aura pas de signal à la sortie C. Ainsi le schéma indiqué réalise l'opération de la négation :

$$C = \overline{A}\,.$$

L'appareil électronique qui réalise la négation logique s'appelle un inverseur. Le schéma fonctionnel de l'inverseur est représenté par la figure 2.

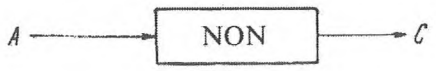


Fig. 2.

2. Circuit « et ».

Le circuit « et » (Fig. 3) réalise le produit logique. Ce schéma représente un groupe de diodes mises en circuit parallèlement (sur la figure 3, les redresseurs d_1, d_2, d_3), successivement réunies à la résistance C. Les diodes ont la propriété d'être des semi-conducteurs, ce qui est conditionné par la nature physique de la substance du redresseur (les redresseurs à germanium, à sélénium et à oxyde de cuivre sont les plus répandus). Des flèches indiquent la direction des courants que les redresseurs sont susceptibles de laisser passer. S'il y a un signal aux entrées A, B, C, c'est-à-dire si la tension y est élevée, les redresseurs sont fermés (ne laissent pas passer le courant) et, par conséquent, le potentiel de la sortie D est élevé, c'est-à-dire qu'un signal apparaît à la sortie. Si ce signal manque ne serait-ce qu'à une entrée, c'est-à-dire que sur cette entrée il y ait une tension basse, le redresseur correspondant laissera passer le courant qui provoquera la chute de la tension sur la résistance R, et à la sortie D il y aura un potentiel bas, c'est-à-dire qu'il n'y aura pas de signal.

Le schéma « et » réalise ce produit logique :

$$D = A \wedge B \wedge C\,.$$

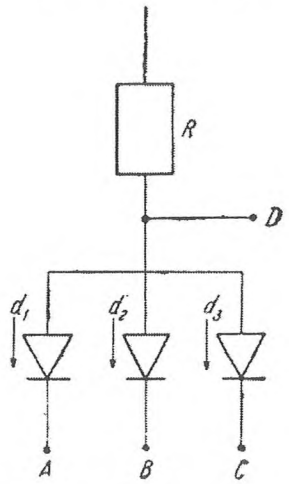


Fig. 3.

La désignation fonctionnelle du schéma « et » est portée sur la figure 4 :

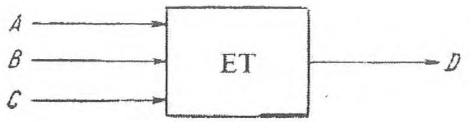


Fig. 4.

3. Circuit « ou ».

Le circuit de la divergence (Fig. 5) réalise l'addition logique. Ce circuit est aussi composé de groupes de redresseurs d_1, d_2, d_3, qui laissent passer le courant dans le sens indiqué par les flèches. S'il n'y a pas de signal aux entrées A, B et C, c'est-à-dire si la tension est basse, le potentiel de sortie D sera bas aussi, c'est-à-dire qu'il n'y aura pas de signal à la sortie. S'il y a un signal, ne serait-ce que sur l'une des entrées, le courant passera au moins à travers l'un des redresseurs, par conséquent le potentiel de la sortie D sera élevé,

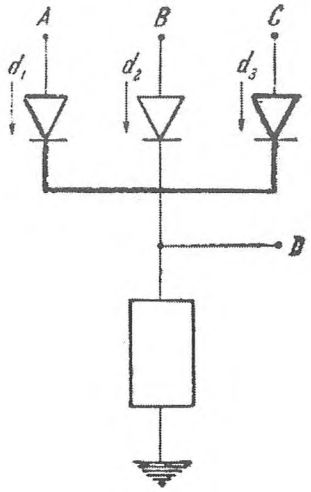


Fig. 5.

et il y aura un signal à la sortie ; c'est de cette façon que le circuit de la figure 5 réalise l'addition logique :

$$D = A \vee B \vee C .$$

La figure 6 représente la désignation fonctionnelle du schéma « ou ».

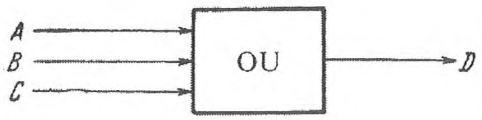


Fig. 6.

Les circuits de contrôle complexes sont définis par des fonctions binaires à variables binaires et peuvent être construits à partir de circuits « NON », de circuits « OU » et de circuits « ET » que nous venons d'examiner.

Par exemple, l'équivalence logique $C = A \sim B$ est représentée par la formule (§ 2.5, page 15).

$$A \sim B = (A \vee \overline{B}) \wedge (\overline{A} \vee B) .$$

Selon cette formule il est facile de construire le circuit qui réalise l'équivalence logique (Fig. 7).

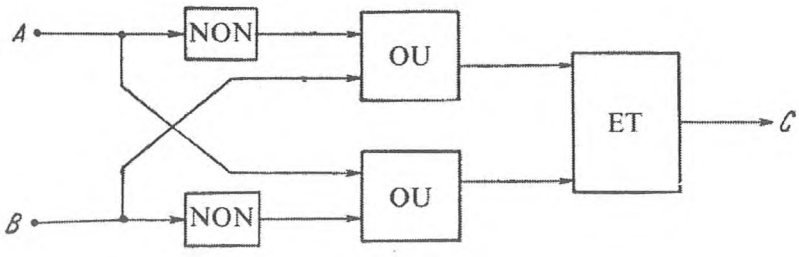


Fig. 7.

4. Circuit de porte binaire.

Le circuit de porte binaire, représentant un réseau de commutation à quatre entrées et une sortie, est illustré par la figure 8. Sur les deux entrées A et B arrivent des signaux qui doivent passer à la sortie C. Aux deux entrées S_1 et S_2 on émet des signaux de contrôle. Selon l'entrée de contrôle, S_1 ou S_2, sur laquelle le signal est appliqué, ce sera soit le signal A, soit le signal B qui sera transmis à la sortie C.

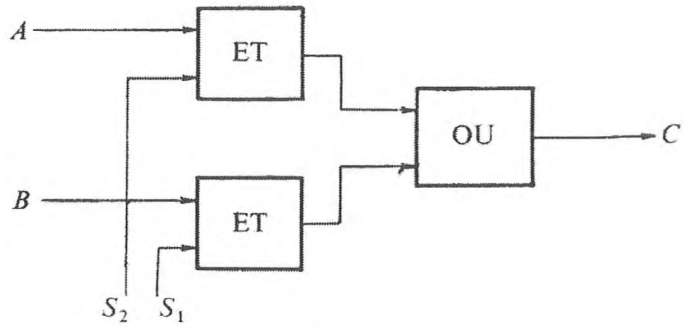


Fig. 8.

5. Circuit décodeur.

Le circuit décodeur se construit avec n entrées et 2^n sorties. A chaque combinaison de signaux d'entrée correspond l'excitation d'une seule ligne de sortie. Sur la figure 9 on a représenté le circuit fonctionnel d'un décodeur à deux

entrées A et B et à quatre sorties P, Q, R, S. Pour chaque combinaison des signaux d'entrée, le signal de sortie ne se présente que sur une seule sortie. Pour les valeurs $A = 0$, $B = 0$ le signal de sortie apparaît sur Q. Pour $A = 0$, $B = 1$ sur P. Pour $A = 1$, $B = 0$ sur S. Pour $A = 1$, $B = 1$ sur R.

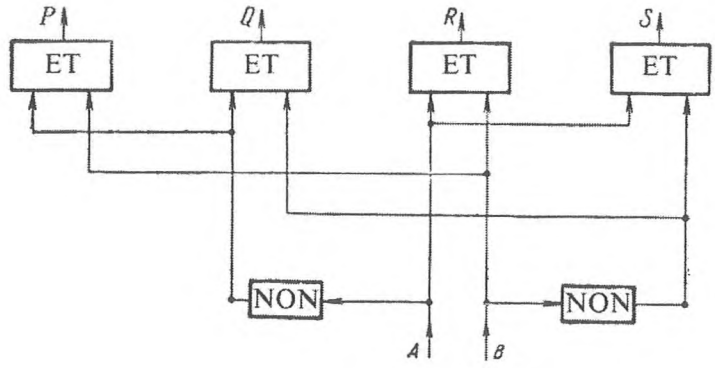


Fig. 9.

6. Additionneur binaire à une position, à deux entrées.

Pour additionner deux nombres, il est nécessaire de pouvoir déterminer la somme de deux chiffres binaires et calculer la retenue de la position précédente. En outre, on peut effectuer une addition de trois chiffres — deux termes et un chiffre de report — soit successivement en additionnant les chiffres deux à deux (on obtiendra la somme tous les deux pas), soit simultanément en additionnant les trois chiffres, ce qui permet de construire un additionneur à deux entrées.

Pour construire le schéma d'un additionneur binaire à une position à deux entrées, avec les éléments logiques « non », « ou », « et » nous établirons le tableau de l'addition de deux nombres d'une seule position A et B avec indication du report pour chaque combinaison possible des valeurs des termes.

Tableau 2. — *Tableau d'addition des nombres binaires à un seul chiffre.*

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Les formules logiques qui définissent la somme et le report R comme fonctions des termes A et B, peuvent être trouvées à partir de ce tableau.

soit par simple sélection, soit en employant le théorème fondamental du calcul des propositions qui concerne la représentation des propositions complexes sous la forme normale.

Pour représenter la somme S et le report R sous forme normale comme fonctions des termes A et B, nous ferons le tableau de toutes les fonctions « ou » possibles à partir de A et B (tableau 3).

Tableau 3. — *Tableau des fonctions « ou » à partir de A et B.*

A	B	$A \vee B$	$A \vee \overline{B}$	$\overline{A} \vee B$	$\overline{A} \vee \overline{B}$
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

D'après le tableau 2, la somme S prend la valeur 0 lorsque les deux termes sont égaux à 0 ou lorsque les termes A et B sont égaux à 1. Selon le tableau 3, on trouve que les fonctions « ou » correspondantes $A \vee B$ et $A \vee \overline{B}$ prennent la valeur zéro, lorsque les termes A et B ont la même valeur. C'est pourquoi la somme S, sous la forme normale, est représentée par la formule :

$$S = (A \vee B) \wedge (\overline{A} \vee \overline{B}) .$$

D'une manière analogue nous trouvons la forme normale du report R :

$$R = (A \vee B) \wedge (A \vee \overline{B}) \wedge (\overline{A} \vee B) .$$

On peut déjà, d'après ces formules, construire les schémas fonctionnels de la somme S et du report R. Cependant, les circuits obtenus ne sont pas les plus simples ; c'est pourquoi il existe un problème de transformation des circuits pour leur donner la forme la plus simple (c'est l'un des principaux problèmes de la synthèse des circuits). Dans le cas donné, les formules obtenues peuvent facilement être simplifiées. Ainsi pour S nous transformons le second membre de l'union selon la formule :

$$\overline{A} \vee \overline{B} = \overline{A \wedge B} .$$

La somme S se met alors sous la forme :

$$S = (A \vee B) \wedge \overline{(A \wedge B)} .$$

Nous transformerons la formule du report de la manière suivante : puisque la somme de tous les *membres de l'union* est identiquement égale à l'unité, on a :

$$\overline{R} = \overline{A} \vee \overline{B}$$

et par conséquent

$$R = A \wedge B .$$

Le circuit fonctionnel de l'additionneur à deux entrées, qui réalise l'addition de deux nombres binaires à une seule position et qui détermine le report, est représenté par la figure 10.

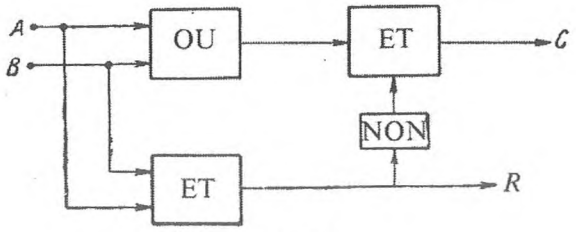


Fig. 10.

7. Additionneur binaire à trois entrées.

On peut effectuer simultanément l'addition de deux nombres binaires et le report de la position précédente. Le circuit de l'opération aura donc trois entrées : les deux termes en A et B et le report de la position précédente en C. La somme S et le report R sont déterminés d'après le tableau 4.

Tableau 4. — *Addition des nombres binaires à une seule position A et B et report de la position précédente C.*

A	B	C	S	R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Comme pour le cas précédent, nous établirons le tableau de toutes les fonctions union possibles à partir de A, B et C.

Tableau 5. — *Fonctions union à partir de A, B et C.*

A	B	C	$A \vee B \vee C$	$A \vee B \vee \overline{C}$	$A \vee \overline{B} \vee C$	$A \vee \overline{B} \vee \overline{C}$	$\overline{A} \vee B \vee C$	$\overline{A} \vee B \vee \overline{C}$	$\overline{A} \vee \overline{B} \vee C$	$\overline{A} \vee \overline{B} \vee \overline{C}$
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

Conformément au tableau 4, la somme S prend la valeur 0 pour les combinaisons des valeurs de A, B et C suivantes :

$$000\ ;\quad 011\ ;\quad 101\ ;\quad 110\ .$$

Relevons les fonctions union qui, pour ces combinaisons, prennent la valeur 0 :

$$A \vee B \vee C\ ;\quad A \vee \overline{B} \vee \overline{C}\ ;\quad \overline{A} \vee B \vee \overline{C}\ ;\quad \overline{A} \vee \overline{B} \vee C\ .$$

Par conséquent :

$$S = (A \vee B \vee C) \wedge (A \vee \overline{B} \vee \overline{C}) \wedge (\overline{A} \vee B \vee \overline{C}) \wedge (\overline{A} \vee \overline{B} \vee C)\ .$$

Par analogie nous trouvons la formule du report R :

$$R = (A \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)\ .$$

Nous laissons le soin au lecteur de simplifier ces formules et d'établir le schéma fonctionnel de l'additionneur binaire à trois entrées.

8. Elément de mémoire. Circuit basculeur.

L'un des principaux éléments des schémas des C. A. N. est *l'élément de mémoire*. Il a les caractéristiques suivantes :

1) Il peut se trouver dans l'un des deux états stables distincts.

2) Il peut influer sur les autres organes, en particulier sur les autres éléments de mémoire, et être influencé par eux. Ce qui est essentiel dans l'élément de

mémoire, c'est le circuit basculeur (Fig. 11). Il est constitué par deux triodes L_1 et L_2. L'anode de la première triode L_1 est reliée à la grille de la seconde triode L_2 par la résistance R_1, et l'anode de la seconde triode L_2 à la grille de la première par la résistance R_2. La principale caractéristique du circuit d'un basculeur est que l'une de ses triodes à l'état stable laisse passer le courant, alors que l'autre ne le laisse pas passer. Lorsque la seconde le laisse passer, c'est la première qui ne le laisse pas passer. Il y a plusieurs moyens de commander le circuit d'un basculeur. Examinons les deux principaux types d'application sur les entrées des signaux de commande.

9. Circuit basculeur utilisé comme compteur.

Sur la figure 11, la lampe qui produit le courant à un moment donné est hachurée. A l'instant où est appliqué sur les cathodes des deux triodes un signal ayant la forme d'une impulsion négative, L_1 se ferme.

Le potentiel de l'anode de L_1 s'accroît brusquement, ce qui produit l'accroissement du potentiel sur la grille de la lampe L_2. Le courant augmente à travers L_2, ce qui provoque la chute du potentiel sur la grille de L_1, qui se maintient à cette valeur jusqu'à l'arrivée du signal suivant. L'impulsion suivante inversera les rôles de L_1 et L_2. A chaque application d'un nouveau signal de commande, le système passera d'un état stable à un autre.

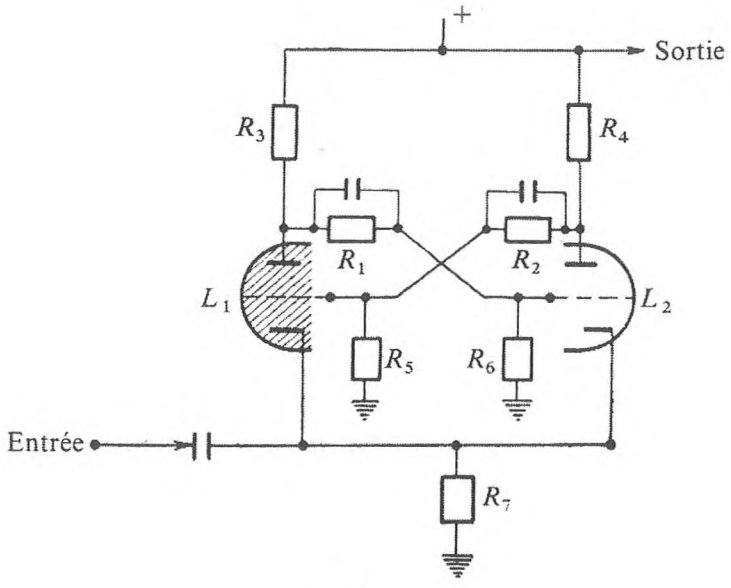
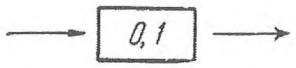


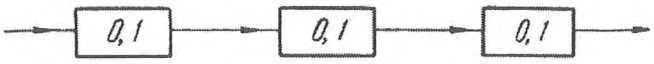
Fig. 11.

Le type de circuit basculeur examiné travaille comme un compteur modulo 2. Les anodes des lampes peuvent être utilisées, en outre, comme sorties du cir-

cuit. L'anode a une tension élevée ou basse suivant l'état de stabilité du circuit. La désignation fonctionnelle du compteur modulo 2 a l'aspect suivant :



Un montage en série de circuits basculeurs (utilisés comme compteurs) donne des compteurs binaires à plusieurs positions. Par exemple le circuit :



effectue le calcul modulo 8, c'est-à-dire de 0 à 7 (en binaire 111).

10. Circuit basculeur utilisé comme registre de mémoire.

Dans la seconde façon d'agencer le circuit basculeur, les signaux sont admis sur deux entrées indépendantes : sur la grille de la lampe L_1 et sur celle de la lampe L_2 (Fig. 12).

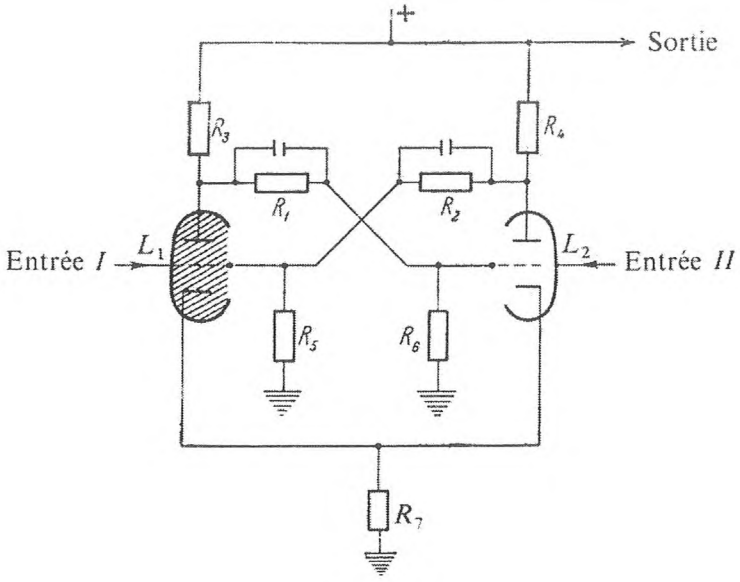
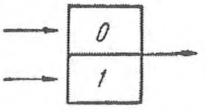


Fig. 12.

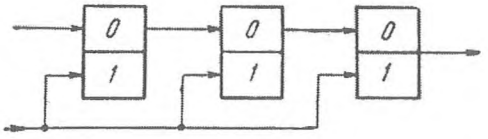
Quand le signal arrive à l'entrée *I*, la lampe L_1 se bloque et la lampe L_2 est passante, ce qui fait tomber brusquement le potentiel de l'anode de L_2. Cependant, lors d'une nouvelle application de signaux sur la grille de L_1, il n'y aura aucun changement dans le circuit, c'est-à-dire que l'action des signaux ne fera que renforcer l'état du circuit. Ce dernier ne changera qu'au moment où un signal sera appliqué à l'entrée *II*, c'est-à-dire sur la grille de L_2. Dans ce cas, le potentiel de l'anode de la lampe L_2 s'accroît brusquement.

Si l'on applique alternativement à l'entrée *I* un code qu'il faut garder et à l'entrée *II* des impulsions qui bloquent le circuit, si 1 représente le signal et 0 son absence, le circuit retiendra chaque fois ce qui a été introduit, à savoir 1 ou 0.

Le circuit représenté sur la figure 12 enregistre la polarité du signal appliqué sur une seule entrée. La désignation fonctionnelle de la mémorisation est la suivante :



Les registres qui permettent d'enregistrer les nombres binaires à plusieurs chiffres se construisent en réunissant des circuits basculeurs qui constituent des registres mémoires. Par exemple, le circuit :



enregistre les nombres binaires de trois positions de 000 à 111, c'est-à-dire de 0 à 7. A une entrée du registre le code binaire d'un chiffre de trois positions est appliqué, à l'autre agissent les impulsions de décalage qui effectuent le report des chiffres dans les éléments correspondants du registre.

4. REPRÉSENTATION DES NOMBRES EN MACHINE

Tout nombre a représenté en binaire sous la forme

$$a = \pm 10^p \sum_{k=1}^{n} a_k . 10^{-k}, \qquad (2)$$

a_k étant soit 0, soit 1 (rappelons que dans le système binaire le symbole 10 désigne le nombre 2) est rendu en machine par une suite donnée de chiffres binaires. Pour figurer les signes des nombres, on utilise aussi les chiffres binaires, le 0 codifie généralement le signe « plus » et le 1 le signe « moins ». C'est commode en ce sens que les signes du produit et du quotient sont déterminés par une addition modulo 2 des codes des facteurs du dividende et du diviseur :

$$0 + 0 = 0; \quad 0 + 1 = 1; \quad 1 + 0 = 1; \quad 1 + 1 = 0.$$

Comme il est facile de le comprendre d'après ce qui précède, une addition de ce genre est réalisée en machine par un simple additionneur à deux entrées.

Deux formes de représentation des chiffres sont employées : elles dépendent des particularités structurales des machines numériques :

1) La forme naturelle, dans les machines à virgule fixe.

2) La forme semi-logarithmique ou normale, dans les machines à virgule flottante.

1. Forme naturelle de représentation des nombres.

Dans la forme naturelle de représentation des nombres comme dans la formule (2), l'exposant est fixé pour tous les nombres. On a le plus souvent $p = 0$; c'est-à-dire que tous les nombres, en valeur absolue, sont plus petits que un. Le cas $p > 0$ ne convient pas, car l'exposant du produit des deux nombres est susceptible d'être plus grand que p. Dans celui où $p < 0$, il se produit une grande perte de chiffres dans la multiplication. Par la suite, nous supposerons que dans les machines à virgule fixe, $p = 0$.

Tout nombre a, dans les machines à virgule fixe, est figuré par la suite $a_0\, a_1\, a_2 \ldots a_n$, a_0 étant le chiffre binaire qui figure le signe du nombre ; $a_1\, a_2 \ldots a_n$ étant les chiffres du nombre a dans la représentation (2). A chaque suite nous attribuerons un nombre :

$$(a) = a_0, a_1\, a_2 \ldots a_n$$

et nous l'appellerons le code du nombre a.

Un nombre positif :

$$a = + \sum_{k=1}^{n} a_k . 2^{-k}$$

est défini par une suite de chiffres binaires

$$0\ a_1\, a_2 \ldots a_n$$

à laquelle correspond un code naturel :

$$(a) = 0, a_1\, a_2 \ldots a_n$$

de sorte que pour les nombres positifs, le code coïncide avec l'écriture binaire du nombre lui-même.

Pour figurer les nombres négatifs, on emploie trois moyens, ou plus précisément trois codes :

1. *Code valeur absolue + signe.* — Soit un nombre négatif dont l'écriture binaire est :

$$a = - 0, a_1\, a_2 \ldots a_n .$$

On lui fait alors correspondre le code suivant :

$$[a]_{\text{dir}} = 1, \quad a_1\, a_2 \ldots a_n = 1 + [a] = 1 - a\,.$$

Le zéro a deux représentations dans le code valeur absolue + signe :

$$[+\,0]_{\text{dir}} = 0{,}00 \ldots 0 \quad \text{et} \quad [-\,0]_{\text{dir}} = 1{,}00 \ldots 0\,.$$

La première représentation s'appellera zéro plus, la seconde zéro moins.

En code valeur absolue + signe, on peut représenter tous les nombres du segment $[-\,(1 - 2^{-n}),\ 1 - 2^{-n}]$.

2. *Code du complément à* 2. — Un nombre binaire négatif $a = -\,0,\ a_1\, a_2 \ldots a_n$ se codifie sous la forme :

$$[a]_{\text{compl.}} = 1\,, \quad a'_1\, a'_2 \ldots a'_n\,,$$

$a'_1\, a'_2 \ldots a'_n$ étant les chiffres du nombre $a' = 0,\ a'_1\, a'_2 \ldots a'_n$ qui complètent $|\,a\,|$ à 1 :

$$|\,a\,| + a' = 1\,.$$

Il s'ensuit que :

$$[a]_{\text{compl.}} = 1 + a' = 10 - |\,a\,|$$

(« 10 » est l'écriture du nombre deux en binaire).

Par exemple, pour $a = -\,0{,}101101$

$$[a]_{\text{compl.}} = 1{,}010011\,.$$

Dans le code du complément à 2, zéro n'a qu'une seule représentation :

$$[0]_{\text{compl.}} = 0{,}00 \ldots 0\,.$$

Le code 1,00 ... 0 figure le nombre $-\,1$.

$$[-\,1]_{\text{compl.}} = 1{,}00 \ldots 0\,.$$

Dans le code du complément à 2, on peut représenter les nombres du segment $[-\,1,\ 1 - 2^{-n}]$.

3. *Code du complément à* 1. — Un nombre binaire négatif $a = -\,0,\ a_1\, a_2 \ldots a_n$ en code du complément à 1 est figuré sous la forme :

$$[a]_{\text{inv.}} = 1\,, \quad \overline{a}_1\, \overline{a}_2 \ldots \overline{a}_n\,,$$

où $\overline{a}_k = 1$ si $a_k = 0$, et $\overline{a}_k = 0$ si $a_k = 1$ $(k = 1, 2, \ldots, n)$, c'est-à-dire que l'on obtient le code du complément à 1 d'un nombre binaire à partir du code

binaire, en remplaçant dans les positions binaires 0 par 1 et 1 par 0, de sorte que :

$$[a]_{\text{inv.}} + |a| = 1{,}11 \ldots 1 = 10 - (10^{-n}),$$

d'où

$$[a]_{\text{inv.}} = 10 - (10^{-n}) + a,$$

$(10^{-n}) = \underbrace{0{,}00 \ldots 01}_{n \text{ positions}}$ étant le code du nombre binaire 10^{-n}.

Dans le code du complément à 1, le zéro est représenté de deux manières :

$$[+\,0]_{\text{inv.}} = 0{,}00 \ldots 0\,; \quad [-\,0]_{\text{inv.}} = 1{,}11 \ldots 1\,.$$

Dans ce code, ainsi que dans le code valeur absolue + signe, on peut représenter les nombres de l'intervalle du segment $[1 - (1 - 2^{-n}), 1 - 2^{-n}]$.

Pour les nombres positifs, nous considérerons que leur représentation est identique dans les trois codes. Pour multiplier et diviser, il est commode d'utiliser la représentation des nombres dans le code valeur absolue + signe ; pour additionner et soustraire, c'est celle des nombres dans les codes du complément à 2 et à 1 qui est intéressante.

Dans les machines à virgule fixe, les positions des nombres sont toutes du même type pour les opérations, ce qui permet de les effectuer très facilement.

Par contre, à cause de l'étendue réduite du nombre de chiffres significatifs, dans les machines à virgule fixe on est obligé de choisir des coefficients de proportionnalité qui permettent de retrouver les résultats corrects ; ceci présente souvent de grandes difficultés.

2. Forme normale de la représentation des nombres.

Dans les machines à virgule flottante, l'ordre du nombre p est variable dans la représentation des chiffres du type (1). Par conséquent, la suite des chiffres qui représente le nombre

$$a = \pm 2^{p} \sum_{k=1}^{n} a_k 2^{-k}$$

doit contenir non seulement la représentation du signe et des chiffres du nombre, mais aussi celle de l'ordre du nombre p, on réserve une position pour y faire figurer le signe de l'ordre.

Si, pour représenter l'ordre, on sépare $r + 1$ positions et pour représenter sa partie numérique $n + 1$ positions, la suite figurant le nombre sous forme normale aura l'aspect :

$$p_0\, p_1 \ldots p_r\, a_0\, a_1 \ldots a_n\,.$$

Dans ce cas, $p_0, p_1, p_2, \ldots, p_r$ est la représentation de l'ordre (p_0 est le signe de l'ordre), $a_0, a_1 \ldots a_n$ est celle de la partie numérique du nombre a (a_0 en est le signe). Par exemple, lorsque $r = 3$, $n = 6$ la suite des chiffres binaires :

$$\underset{\text{ordre}}{0101} \qquad \underset{\text{partie numérique}}{0101110}$$

figure le nombre d'ordre $p = +5$ et de partie numérique $q = +0{,}101110$, c'est-à-dire représente le nombre binaire $a = 10^5 . 0{,}101110$.

On dit que le nombre binaire $a = 10^p . q$ est *normalisé*, si dans la partie numérique du nombre $q = a_0\, a_1\, a_2 \ldots a_n$ le premier chiffre $a_1 = 1$, c'est-à-dire si $\frac{1}{10} \leqslant |q| < 1$ (« 10 » étant le nombre binaire 2).

Le plus petit nombre binaire normalisé, représentable dans une machine à virgule flottante dans laquelle on a séparé $r + 1$ positions pour figurer l'ordre, est égal à

$$a = 10^{-(10^r - 1)} . 0{,}1 = 10^{-10^r};$$

par conséquent, le nombre $a < 10^{-10^r}$ ne peut déjà plus être représenté en machine sous forme normalisée. En général, ces nombres s'identifient à 0. Parfois l'ordre $-2^r + 1$, indépendamment de la valeur de la partie numérique, est identifié à 0.

Les ordres peuvent prendre toutes les valeurs entières (décimales) de $-(2^r - 1)$ à $2^r - 1$. Par exemple, si $r = 5$, ils peuvent prendre les valeurs entières de -31 à $+31$, et réciproquement l'étendue des nombres qui peuvent être représentés par ces ordres comprend les nombres de 2^{-31} à 2^{+31}.

L'application de la forme normale à la représentation des nombres donne la possibilité d'obtenir une gamme assez étendue de nombres représentés en machine, aussi bien très petits que très grands en valeur absolue ; de plus on peut représenter tous les nombres normalisés avec à peu près la même exactitude relative.

Un inconvénient de la forme normale : l'extrême complexité de l'exécution des opérations par rapport à celles que l'on fait sur les nombres sous forme naturelle et, par conséquent, une grande complication dans la construction des circuits de la machine.

5. OPÉRATIONS SUR LES NOMBRES DANS LES CALCULATRICES NUMÉRIQUES

1. Addition et soustraction en machine à virgule fixe.

On rentre les nombres dans l'additionneur en code du complément à 2 ou à 1. Nous supposerons que le résultat de l'opération donne un nombre de valeur absolue inférieure à 1.

Examinons l'addition de nombres en code du complément à 2. Soient deux nombres a et b et supposons que $|a + b| < 1$.

Si $a \geqslant 0$, $b \geqslant 0$, alors $a + b \geqslant 0$ et

$$[a]_{\text{compl.}} + [b]_{\text{compl.}} = a + b = [a + b]_{\text{compl.}}.$$

Si $a < 0$, $b \geqslant 0$, mais $a + b \geqslant 0$, alors

$$[a]_{\text{compl.}} + [b]_{\text{compl.}} = 10 + a + b = 10 + [a + b]_{\text{compl.}};$$

pour obtenir le code du complément à 2 d'une somme il faut négliger le « un » de report que l'on obtient dans la position du signe (dans la position des « uns » du code).

Si $a < 0$, $b \geqslant 0$, mais $a + b < 0$, alors

$$[a]_{\text{compl.}} + [b]_{\text{compl.}} = 10 + a + b = [a + b]_{\text{compl.}}.$$

Si $a < 0$ et $b < 0$, alors

$$[a]_{\text{compl.}} + [b]_{\text{compl.}} = 10 + a + 10 + b = 10 + [a + b]_{\text{compl.}};$$

dans ce cas un « un » de report apparaît dans la position des « uns » du code ; il faut le négliger.

L'addition des nombres en code du complément à 2 peut donc être réalisée sur un additionneur dans lequel les codes des nombres sont rangés comme des nombres ordinaires et où le « un » de report de la position du signe (s'il apparaît) est négligé.

Dans ce cas, la position du signe est considérée comme une position de « uns » entiers.

Exemple 1.

$$\begin{array}{rrrr} x = & 0{,}1101 & [x]_{\text{compl.}} = 0{,}1101 & + \; 0{,}1101 \\ y = & 0{,}0001 & [y]_{\text{compl.}} = 0{,}0001 & \quad\; 0{,}0001 \\ \hline x + y = & 0{,}1110 & [x + y]_{\text{compl.}} = 0{,}1110 \leftarrow & 0{,}1110 \end{array}$$

Exemple 2.

$$\begin{array}{rrrr} x = & -\,0{,}0001 & [x]_{\text{compl.}} = 1{,}1111 & + \; 1{,}1111 \\ y = & +\,0{,}1101 & [y]_{\text{compl.}} = 0{,}1101 & \quad\; 0{,}1101 \\ \hline x + y = & 0{,}1100 & [x + y]_{\text{compl.}} = 0{,}1100 \leftarrow & 10{,}1100 \end{array}$$

Exemple 3.

$$\begin{array}{rrrr} x = & -\,0{,}1101 & [x]_{\text{compl.}} = 1{,}0011 & + \; 1{,}0011 \\ y = & 0{,}0001 & [y]_{\text{compl.}} = 0{,}0001 & \quad\; 0{,}0001 \\ \hline x + y = & -\,0{,}1100 & [x + y]_{\text{compl.}} = 1{,}0100 \leftarrow & 1{,}0100 \end{array}$$

Exemple 4.

$$\begin{array}{rrrr} x = & -\,0{,}1101 & [x]_{\text{compl.}} = 1{,}0011 & + \; 1{,}0011 \\ y = & -\,0{,}0001 & [y]_{\text{compl.}} = 1{,}1111 & \quad\; 1{,}1111 \\ \hline x + y = & -\,0{,}1110 & [x + y]_{\text{compl.}} = 1{,}0010 \leftarrow & 11{,}0010 \end{array}$$

Examinons l'addition de nombres présentés en code du complément à 1 :

— si $a \geqslant 0$ et $b \geqslant 0$, alors

$$[a]_{\text{inv.}} + [b]_{\text{inv.}} = a + b = [a + b]_{\text{inv.}} ;$$

— si $a < 0$, $b \geqslant 0$ et $a + b \geqslant 0$, alors

$$[a]_{\text{inv.}} + [b]_{\text{inv.}} = 10 - (10^{-n}) + a + b = 10 - (10^{-n}) + [a + b]_{\text{inv.}} .$$

Dans ce cas, il faut rejeter le « un » de report qui se trouve dans la position de poids fort (position du signe) et en ajouter un dans la position de poids faible ; ceci est réalisable au moyen d'un report cyclique de la position de poids fort dans celle de poids faible.

Si $a < 0$, $b \geqslant 0$ et $a + b < 0$, alors

$$[a]_{\text{inv.}} + [b]_{\text{inv.}} = 10 - (10^{-n}) + a + b = [a + b]_{\text{inv.}} ;$$

Si $a < 0$, $b < 0$, alors $a + b < 0$ et

$$[a]_{\text{inv.}} + [b]_{\text{inv.}} = 10 - (10^{-n}) + [a + b]_{\text{inv.}} .$$

C'est-à-dire qu'il faut effectuer le report cyclique du « un » de la position du signe à la position de poids faible. Ainsi l'addition des nombres en code inverse est faite par un totalisateur avec report cyclique de la position de poids fort à celle de poids faible ; les codes sont rangés comme des nombres ordinaires et la position du signe est la position de poids fort des « uns » entiers.

Exemple 1.

$$\begin{array}{rrrrr} x = & 0{,}1101 & [x]_{\text{inv.}} = & 0{,}1101 & \\ y = & 0{,}0001 & [y]_{\text{inv.}} = & 0{,}0001 & + \\ \hline x + y = & 0{,}1110 & [x + y]_{\text{inv.}} = & 0{,}1110 & \leftarrow \end{array} \quad \begin{array}{r} \dot{0}{,}1101 \\ 0{,}0001 \\ \hline 0{,}1110 \end{array}$$

Exemple 2.

$$\begin{array}{rrrrr} x = & -\,0{,}0001 & [x]_{\text{inv.}} = & 1{,}1110 & \\ y = & 0{,}1101 & [y]_{\text{inv.}} = & 0{,}1101 & + \\ \hline & & & & \\ x + y = & 0{,}1100 & [x + y]_{\text{inv.}} = & 0{,}1100 & \leftarrow \end{array} \quad \begin{array}{r} 1{,}1110 \\ 0{,}1101 \\ \hline 10{,}1011 \\ \text{report} \\ \text{cyclique} \\ 0{,}1100 \end{array}$$

Exemple 3.

$$\begin{array}{rrrrr} x = & -\,0{,}1101 & [x]_{\text{inv.}} = & 1{,}0010 & \\ y = & 0{,}0001 & [y]_{\text{inv.}} = & 0{,}0001 & + \\ \hline x + y = & -\,0{,}1100 & [x + y]_{\text{inv.}} = & 1{,}0011 & \leftarrow \end{array} \quad \begin{array}{r} 1{,}0010 \\ 0{,}0001 \\ \hline 1{,}0011 \end{array}$$

Exemple 4.

$$\begin{array}{rrrrr} x = & -\,0{,}1101 & [x]_{\text{inv.}} = & 1{,}0010 & \\ y = & -\,0{,}0001 & [y]_{\text{inv.}} = & 1{,}1110 & + \\ \hline & & & & \\ x + y = & -\,0{,}1110 & [x + y]_{\text{inv.}} = & 1{,}0001 & \leftarrow \end{array} \quad \begin{array}{r} 1{,}0010 \\ 1{,}1110 \\ \hline 11{,}0000 \\ \text{report} \\ \text{cyclique} \\ 1{,}0001 \end{array}$$

La soustraction des nombres se réduit à une addition algébrique. En outre, le signe du second terme se transforme en son opposé.

2. Addition et soustraction en machine à virgule flottante.

Dans les machines à virgule flottante, les opérations s'effectuent sur des nombres de forme normale. Pour additionner des nombres binaires donnés sous forme normale $a = 10^p \alpha$ et $b = 10^q \beta$, il faut commencer par comparer les ordres. Si $p > q$, c'est-à-dire si $p = q + k$ $(k > 0)$, alors le terme b est changé en un nombre non normalisé $b_1 = 10^p \beta_1$, où β_1 se déduit de β par un décalage de k positions vers la droite. Le nombre b_1 coïncide avec le nombre b à 10^{p-n} près (tous les chiffres seront donnés en binaire, sauf indication contraire). L'addition des parties numériques α et β_1 est ensuite effectuée selon les règles d'addition des nombres binaires en machine à virgule fixe. Il peut se faire que la somme qui en résulte ne soit pas un nombre normalisé. Lorsque $|\alpha + \beta_1| > 1$, la normalisation est détruite à gauche, lorsque $|\alpha + \beta_1| < \frac{1}{10}$ le résultat est obtenu avec destruction de la normalisation à droite.

Ainsi, après avoir additionné les parties numériques, il faut normaliser le résultat.

La présence de deux chiffres identiques 0,0 (pour les nombres positifs) ou 1,1 (pour les nombres négatifs en convention du complément à 1 ou à 2) dans deux positions voisines (dans la position du signe et dans la position numérique suivante) indique la destruction de la normalisation à droite (c'est-à-dire que la partie numérique de la somme obtenue est inférieure en valeur absolue à 0,1).

Exemple. $a = 10^{10}.0{,}110110$ $\quad b = 10^{10}.(-0{,}101011)$

	ordre	partie numérique
code inverse du nombre a	0 010	0,110110
code inverse du nombre b	0 010	1,010100
		10,001010
		report cyclique
code de la somme	0 010	0,001011

Après la normalisation (décalage de deux positions vers la gauche), nous obtiendrons la partie numérique de la somme 0,101100. La somme est égale à

$$a + b = 10^{10}.10^{-10}.0{,}101100 = 10^{0}.0{,}101100\,.$$

Pour pouvoir décaler un nombre vers la gauche (c'est-à-dire obtenir une partie numérique de la somme supérieure à « un » en valeur absolue), on emploie un nouveau code dans lequel le signe occupe deux positions binaires : les nombres positifs sont représentés sous la forme 00, $a_1\ a_2, \ldots, a_n$, les nombres

négatifs par un code de complément à r modifié :

$$[a]_{\text{compl. mod.}} = 100 + a$$

ou par le code de complément à 1 modifié :

$$[a]_{\text{inv. mod.}} = 100 - (10^{-n}) + a\,.$$

Les nombres de valeur absolue inférieure à un ont, en code modifié, soit deux zéros (00, ...), soit deux « uns » (11, ...) dans les positions du signe. L'addition des nombres, en code de complément à 2 modifié, s'effectue exactement comme en code de complément à 2 ordinaire, si ce n'est que le « un » du report, qui apparaît dans la position de poids fort du signe, y est négligé. L'addition des nombres en code de complément à 1 modifié est effectuée exactement comme en code de complément à 1 ordinaire, si ce n'est que le report cyclique se fait de la position de poids fort du signe à la position numérique de poids faible.

Lorsque s'effectue une addition de mantisses de deux nombres en code modifié, l'apparition de chiffres différents, 10 ou 01, dans les positions du signe indique la destruction de la normalisation à gauche.

Exemple 1 : $a = 10^{10}.0{,}110110$, $b = 10^{10}.0{,}010110$.
Somme des parties numériques en code modifié :

$$\begin{array}{r} 00{,}110110 \\ +\ 00{,}010110 \\ \hline 01{,}001100 \end{array}$$

Le décalage d'une position vers la droite donne la partie numérique normalisée de la somme. Nous obtenons :

$$a + b = 10^{11}.0{,}100110\,.$$

Exemple 2 : $a = 10^{10}(-\,0{,}111011)$, $b = 10^{10}(-\,0{,}010110)$.
Somme des parties numériques en code du complément à 2 modifié :

$$\begin{array}{r} 11{,}000101 \\ 11{,}101010 \\ \hline 10{,}101111 \end{array}$$

Après le décalage d'un rang à droite et l'enregistrement des « uns » dans la position de poids fort du signe, nous obtenons la somme normalisée des parties numériques en code du complément à 2 modifié : 11,010111. Par conséquent :

$$a + b = 10^{11}(-\,0{,}1010001)\,.$$

La soustraction des chiffres écrits sous forme normale revient à l'addition algébrique de leur code de complément à 1 ou à 2, comme dans les machines à virgule fixe.

3. Multiplication et division.

La multiplication des nombres en machine s'effectue habituellement en code valeur absolue + signe. La multiplication des parties numériques des nombres binaires consiste à décaler le multiplicande successivement vers la gauche d'un nombre de positions égal à celui des chiffres significatifs du multiplicateur et à additionner les parties ainsi obtenues. Le signe du produit est obtenu par l'addition des signes des facteurs dans un additionneur binaire à une position, d'après les règles suivantes :

$$0 + 0 = 0\ ;\quad 1 + 0 = 1\ ;\quad 0 + 1 = 1\ ;\quad 1 + 1 = 0\ .$$

Exemple :

$$\begin{array}{rl} & 0{,}110101 \\ \times & 0{,}101101 \\ \hline & \quad 110101 \\ & \quad\ 110101 \\ & \quad\quad 110101 \\ & \quad\quad\ 110101 \\ \hline & 0{,}100101010001 \end{array}$$

La multiplication des chiffres écrits sous la forme normale s'effectue en trois étapes :

1) Addition des chiffres représentant les signes des facteurs, qui donne le signe du produit.
2) Addition algébrique des ordres des facteurs ; elle donne l'ordre du produit.
3) Multiplication des parties numériques des facteurs ; elle donne la partie numérique du produit.

Si nécessaire, on normalise les résultats.

4. Division des nombres en machine à virgule fixe.

En binaire, on effectue la division des nombres en déterminant successivement les chiffres du quotient au moyen de la soustraction du diviseur. Si le résultat de la soustraction est positif, le chiffre correspondant du quotient est égal à « un », si le résultat est négatif, ce chiffre est égal à zéro, et le reste précédent se déplace d'une position à gauche, etc. Le retour au reste précédent se fait en additionnant le diviseur au reste négatif. Ce cycle d'opérations se répète n fois (n est le nombre de chiffres du nombre).

Exemple : Dividende $a = 0{,}100101$, diviseur $b = 0{,}110101$.

```
 _0,1001010   | 0,110101
  ·110101     |----------
--------------| 0,101100
  _0010101
    110101
  ---------
   -001011
   -------
   _101010
     110101
   -------
   _0011111
     110101
   -------
   _001001
     110101
   -------
   -100011
   -------
   _010010
     110101
   -------
   -10001
   ------
    100100
```

Quotient 0,101100 ,
Reste 0,0000001 .

Dans la détermination du deuxième, du cinquième et du sixième chiffre du quotient, on a obtenu un résultat négatif à la soustraction. De plus, on a conservé respectivement les premier, quatrième, cinquième restes déplacés d'une position à gauche.

La division des nombres écrits sous forme normale s'effectue en trois étapes :

1) Détermination du signe du produit par addition des signes du dividende et du diviseur.

2) Soustraction des exposants avec enregistrement des signes des exposants. Le résultat donnera l'exposant du quotient.

3) Division de la partie numérique du dividende par celle du diviseur, ce qui donne la partie numérique du quotient.

Si besoin est, on procède à la normalisation des résultats.

5. Opérations sans report.

Si les opérations que l'on fait sur les chiffres de chaque position des nombres sont indépendantes les unes des autres, elles portent le nom d'opérations sans report.

1. *Complément sans report.*

L'opération transforme la suite $a_0\, a_1\, a_2 \ldots a_n$ en une suite

$$a'_0\, a'_1 \ldots a'_n\,, \quad \text{où} \quad a'_k = 1 - a_k \quad (k = 0, 1, 2, \ldots, n)\,.$$

Dans les machines à virgule fixe, cette opération transforme le nombre a, donné par le code inverse, en nombre $-a$ donné par le même code, puisque

$$[a]_{\text{inv.}} + |a| = 1, 1 \ldots 1 = 10 - (2^{-n})\,.$$

Ainsi, lorsqu'en machine à virgule fixe, les nombres sont donnés en code du complément à 1, l'opération de changement de signe est celle du complément sans report.

2. *Addition sans report.*

L'addition sans report est celle des nombres modulo 2.

$$0 + 0 = 0\,; \quad 0 + 1 = 1 + 0 = 1\,; \quad 1 + 1 = 0\,.$$

A l'aide de l'addition sans report, il est possible de réaliser le complément de chaque chiffre binaire d'un nombre, en l'additionnant avec un nombre qui a dans toutes ses positions le chiffre 1.

3. *Addition logique sans report.*

L'addition logique des chiffres binaires est définie par le tableau du § 2.3. En utilisant cette opération, on peut obtenir à partir d'un nombre, écrit en code valeur absolue + signe, son module négatif au moyen de l'addition de ce nombre avec le code 1, 00 ... 0.

4. *Multiplication logique du report.*

La multiplication logique des chiffres binaires est déterminée par le tableau du § 2.2. Cette opération permet de choisir des positions quelconques dans la suite $a_0\, a_1 \ldots a_n$. Pour cela, il faut multiplier la suite donnée par une suite dans laquelle les positions correspondantes aux positions choisies contiennent 1 et les autres 0. On veut déterminer de cette manière le signe du nombre, la valeur absolue de l'exposant du nombre ou de sa partie numérique, le code de l'opération ou le code d'une adresse. Cette opération peut être utilisée pour séparer certaines parties du code.

5. *Décalage.*

L'opération de décalage consiste dans la translation des chiffres du code d'un nombre fixé de positions à gauche ou à droite. Lorsque l'on décale le

code $a_0\, a_1 \ldots a_n$ de s positions à gauche, le code $a_{s+1}\, a_{s+2} \ldots a_n \underbrace{0 \ldots 0}_{s}$ se substitue à lui. Lorsqu'on le décale de s positions à droite, on obtient le code $\underbrace{0 \ldots 0}_{s}\, a_0\, a_1 \ldots a_{n-s}$.

Les machines à virgule flottante effectuent l'opération de décalage de la partie numérique du nombre. Dans les machines à virgule fixe, l'opération de décalage est équivalente à une multiplication ou à une division par 2^k sans arrondir.

6. Opération de comparaison.

Cette opération détermine quel est de deux nombres a et b le plus grand (le plus petit). Pour comparer deux nombres a et b, on effectue la soustraction $a-b$, et le signe du résultat permet de déterminer le plus grand des deux nombres. Le cas d'égalité de a et de b est interprété d'après le résultat de la soustraction, selon le code employé pour représenter les nombres. Avec le code du complément à 2 le signe de zéro est positif et, par conséquent, l'inégalité $a > b$ est l'interprétation du signe de l'égalité. Avec le code du complément à 1, la soustraction de nombres égaux donne un zéro négatif.

$$[-0]_{\text{inv.}} = 1{,}11 \ldots 1$$

et, par conséquent, l'inégalité $a < b$ est l'interprétation du signe de l'égalité.

7. Précisions dans l'exécution des opérations et arrondi du résultat.

La précision avec laquelle les opérations sur les nombres s'effectuent en machine, est déterminée par la forme de l'opération et par le procédé de réalisation.

1. *Opération de décalage.*

L'exécution de cette opération correspond à une multiplication par 2^k. Lorsque le décalage est possible à gauche, il n'y a pas de perte de chiffres significatifs. Un tel décalage est une multiplication par 2^k exacte, k étant un nombre positif. Lorsque le décalage se fait à droite de p positions, les derniers p des chiffres significatifs sont perdus. Par suite de l'opération de décalage à droite, il y a par rapport à la division par 2^p exacte une erreur inférieure à l'unité dans la dernière position.

La plus grande erreur, égale au chiffre binaire

$$10^{-n}(1 - 10^{-p}),$$

se produit lorsqu'il y a des « uns » dans tous les p des positions rejetées. Conformément à cela, dans les machines à virgule flottante, lorsqu'il y a normalisation de nombres à partie numérique plus grande que 1 en valeur absolue, l'erreur commise ne dépasse pas le « un » de la dernière position de la partie numérique du nombre.

2. *Addition et soustraction.*

Dans les machines à virgule fixe, les opérations d'addition et de soustraction sont exactes. Dans les machines à virgule flottante, il s'y introduit des erreurs dues aux décalages nécessaires pour aligner les positions et normaliser les résultats. L'alignement des positions crée une erreur inférieure à l'unité ; cependant après la normalisation à gauche, l'erreur peut atteindre le chiffre le plus à gauche à 1 près. En effet, soit $n = 6$, $a = 10^{11}.0{,}100000$, $b = 10^{10}.0{,}111111$. Lors de la soustraction, les positions s'alignent :

$$b_1 = 10^{11}.0{,}011111 \quad \text{et} \quad a - b = 10^{11}.0{,}000001 = 10^{-10}.0{,}100000\,.$$

La valeur exacte de la différence est $10^{-10}.0{,}01$. Une telle situation peut se présenter lorsque la différence entre les ordres est de « un ». L'accroissement de l'erreur est éliminé par la présence dans l'additionneur d'une position supplémentaire, c'est-à-dire si le nombre des positions numériques y dépasse d'une unité celui des positions numériques des mémoires de l'O. M. (organe de mémoire). En effet, si les ordres des nombres diffèrent de plus d'une unité, lorsqu'il y a normalisation des résultats, le décalage à gauche ne peut se faire que d'une seule position : ce qui donne une erreur inférieure à l'unité de la dernière position. Une normalisation du résultat supérieure à une position ne peut se produire qu'à condition que les ordres des termes n'aient pas plus d'une unité de différence. Mais, dans ce cas, la présence d'une position supplémentaire dans l'additionneur assure la conservation de tous les chiffres et, par conséquent, l'exactitude du résultat.

3. *Multiplication et division.*

Pour la multiplication et la division en machine, les erreurs ne dépassent pas l'unité.

Pour diminuer les erreurs dans les opérations sur les nombres exécutés en machine, on introduit des opérations avec arrondi ; pour ce faire, on utilise une position supplémentaire dans l'additionneur.

Lorsqu'on effectue des opérations sur les nombres avec arrondi, il est de règle que l'erreur ne dépasse pas la moitié de l'unité de la plus petite position. En outre, au cours de l'exécution des dernières opérations avec arrondi, la compensation des erreurs est possible.

6. PASSAGE D'UN SYSTÈME DE NUMÉRATION A UN AUTRE

Lorsqu'on convertit les nombres du système décimal en binaire et vice versa, on utilise comme étape transitoire l'écriture décimale codée binaire. Dans le système décimal codé binaire, chaque chiffre de l'écriture décimale (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) est représenté sous la forme de nombres binaires à quatre positions 0000, 0001, 0010, ..., 1001. Le système décimal codé binaire est moins économique que le binaire, car les quatre positions binaires sont employées pour la représentation tout au plus de dix chiffres (au lieu de 16 chiffres possibles) ; l'écriture d'un nombre en décimal codé binaire est de 20 % plus longue que celle en binaire pur. Par exemple, le nombre 637 en décimal codé binaire a la forme 0110 0011 0111 (12 chiffres). Alors qu'en binaire il a celle-ci : 1001111101 (10 chiffres).

Afin de trouver l'écriture binaire du nombre $a = \sum_{k=1}^{n} a_k . 10^{m-k}$, dont chaque chiffre a_k est posé sous la forme d'un caractère de quatre chiffres binaires :

$$a_k = a_{k1}\, a_{k2}\, a_{k3}\, a_{k4} \qquad (k = 1, 2, \ldots, n)\,,$$

il faut successivement isoler un caractère de quatre chiffres binaires du nombre a (c'est-à-dire les coefficients a_k) et exécuter le calcul du polynôme $\sum_{k=1}^{m} a_k\, t^{m-k}$, avec $t = 10$ (en binaire $t = 1010$).

Examinons maintenant la conversion du binaire en décimal. Soit $a = 0, a_1\, a_2 \ldots a_n$ — l'écriture binaire d'un nombre ($a_k = 0$ ou $1, k = 1, 2, \ldots, n$). Nous trouverons successivement les chiffres α_k dans l'écriture décimale :

$$a = \sum_{k=1}^{m} \alpha_k\, 10^{-k}$$

de la façon suivante :

1) Calcul de la partie entière du nombre $10a$

$$\alpha_1 = [10a]\ ;$$

2) Calcul de la partie fractionnaire du nombre $10a$

$$b_1 = \{10a\} = \sum_{k=2}^{m} \alpha_k\, 10^{-k}\ ;$$

3) Calcul de la partie entière du nombre $10b_1$

$$\alpha_2 = [10b_1]\ ;$$

etc. Ainsi la succession des chiffres $\alpha_1, \alpha_2, \ldots, \alpha_m$ de l'écriture décimale du nombre a, se trouve d'après le schéma suivant :

$$\alpha_k = [10b_k], \quad b_k = \{10b_{k-1}\}, \quad b_0 = a \qquad (k = 1, 2, \ldots, m).$$

Pour convertir un nombre binaire écrit sous la forme normale $a = 10^p q$ en système de numération décimal, la partie numérique se transforme en système décimal selon la règle précédente et l'ordre du nombre S y est défini à partir de la condition suivante : $10^s > a \geqslant 10^{s-1}$.

CHAPITRE II

PRINCIPES DE LA COMMANDE PAR PROGRAMME SUR LES CALCULATRICES

Pour un mathématicien, toute calculatrice représente un organe capable d'exécuter une série d'opérations arithmétiques et logiques, de conserver et de sortir des informations sur leurs résultats.

En fonction de leurs possibilités, on distingue deux types de calculatrices : celles qui, spécialisées, n'exécutent qu'une suite déterminée d'opérations permettant la résolution d'un genre particulier de problèmes et celles qui, universelles, résolvent les suites d'opérations les plus diverses.

Dans ce chapitre nous allons exposer les bases du traitement automatique (par programme) des problèmes sur calculatrices électroniques universelles.

1. CIRCUITS DE BASE DES C. A. N.

1. Principes généraux de construction.

Comme on l'a déjà noté, les calculatrices électroniques de type universel sont destinées à résoudre des problèmes numériques et logiques variés, exigeant un grand nombre d'opérations et une haute précision.

Les principes généraux de construction des C. A. N. universelles modernes sont, en un certain sens, analogues à ceux des calculatrices déjà connues. Outre l'application des moyens actuels de l'électronique, ce qui est fondamentalement nouveau dans les machines à calculer électroniques, c'est l'utilisation des circuits qui assurent la mémorisation des calculs, la complète automatisation de tout le processus calculatoire et l'extrême rapidité de son exécution.

Après le choix d'une méthode numérique, l'algorithme qui permet de réaliser la résolution d'un problème sur C. A. N. doit être divisé en une suite d'opérations numériques exécutables en machine que nous appellerons ultérieurement opérations élémentaires.

L'exécution rapide des opérations purement numériques n'entre pas seule en jeu pour obtenir une grande rapidité de calcul, ces opérations ne constituent,

en effet, qu'une partie du calcul ; la seconde étant l'indication de l'ordre chronologique du travail : le programme des calculs. Les indications qui concernent le programme doivent pouvoir être communiquées très rapidement à la machine pour que l'automatisation soit complète. L'exécution automatique de tous les calculs d'après un programme donné est réalisée à l'aide d'un dispositif de commande qui, pour cette raison, est encore appelé dispositif de commande par programme.

Une calculatrice universelle est composée d'une série d'organes (blocs) dont chacun possède une affectation spéciale. Voici les principaux :

1) Organe d'entrée des données.
2) Organe de mémoire (OM).
3) Organe de calcul (OA).
4) Organe de commande automatique et manuelle (OC).
5) Organe de sortie des résultats.

Le diagramme fonctionnel d'une C. A. N. est représenté par la figure 13 ; les flèches indiquent les liens qui existent entre les différents organes permettant de transmettre l'information codée (conservée ou élaborée dans un bloc) d'un organe à un autre.

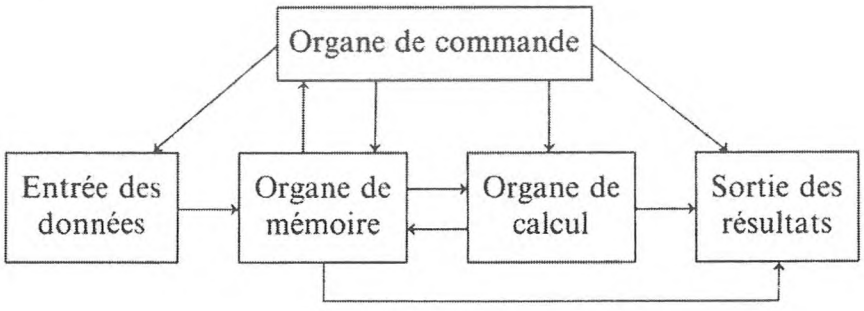


Fig. 13.

Le dispositif d'entrée est destiné à transmettre dans le dispositif de mémoire l'information codée indispensable à la résolution du problème. Le plus souvent le travail de cet organe se fait automatiquement à l'aide de cartes ou de bandes perforées.

Les organes de mémoire, de calcul et de commande sont destinés à effectuer directement les opérations de calcul.

L'organe de mémoire sert à conserver les données introduites en machine et les résultats des calculs intermédiaires et définitifs.

L'organe de calcul d'une calculatrice moderne représente l'ensemble des circuits électroniques effectuant à une vitesse extraordinaire les différentes opérations sur les codes qui y entrent. De plus, chaque machine possède son propre choix d'opérations que l'organe de calcul est capable d'exécuter sur les codes. Quand une opération quelconque est à effectuer, l'organe de calcul

est introduit dans le circuit d'une façon spéciale, à l'aide d'un commutateur d'opérations.

L'organe de commande exécute la suite des calculs conformément à un programme donné.

L'organe de sortie est destiné à extraire de la machine, conformément au programme établi, les résultats définitifs du calcul. Le plus souvent il se présente sous forme d'une machine imprimante, à l'aide de laquelle les résultats cherchés sont transcrits sur une bande par exemple, en une séquence déterminée.

2. Organe de mémoire.

L'organe de mémoire se compose d'une série de cellules numérotées : normalement, elles ont un nombre identique de positions qui sont à leur tour numérotées. Dans chaque position, il peut y avoir 0 ou 1. L'ensemble des « zéros » et des « uns » conservés dans une cellule s'appelle *code*.

Normalement, une seule et même cellule peut servir aussi bien à stocker les codes des nombres (avec choix de la position de la virgule) qu'à conserver les codes de commande (appelés ordres ou instructions) qui servent à effectuer dans la machine des opérations élémentaires et à déterminer l'ordre de leur succession. Pour des raisons de commodité on donne aux numéros des cellules correspondantes les mêmes numéros qu'aux instructions ou aux nombres.

Comme on le voit sur le diagramme fonctionnel, l'organe de mémoire est relié à ceux d'entrée et de sortie, de calcul et de commande. L'information de départ du problème à résoudre (les données numériques et les instructions du programme) arrive sous forme codée par le canal de l'organe d'entrée dans les cellules de l'organe de mémoire prévues par le programme. L'organe de mémoire envoie dans l'organe de commande les codes des instructions de commande — instructions du programme — sous l'action desquelles les informations sont extraites de l'organe de mémoire et envoyées dans l'organe de calcul afin d'y être traitées, les résultats du calcul sont transmis à l'organe de mémoire. Conformément au programme, ils sont ensuite transférés dans l'organe de sortie.

En règle générale, l'organe de mémoire comporte deux parties : l'une interne, l'autre externe. *La mémoire interne* se distingue de la mémoire externe par sa plus grande rapidité de travail, c'est-à-dire que les codes en sont extraits ou y sont introduits à une vitesse plus grande. Elle est destinée à l'utilisation automatique directe du dispositif de calcul.

Ses cellules sont de deux sortes : passives (constantes) et actives (variables opératives). Les codes peuvent être extraits (lus) aussi bien des unes que des autres au moyen de l'organe de commande (conformément au programme) pour être traités dans l'organe de calcul. Cependant, les résultats des calculs ne peuvent être envoyés que dans les cellules de la mémoire active, où ils pro-

voquent l'effactement des codes qui s'y trouvaient auparavant. Cela est pris en considération dans l'élaboration des programmes.

Les possibilités d'une calculatrice sont dans une large mesure déterminées par la capacité de la partie intérieure de l'organe de mémoire (le nombre des codes qui peuvent y être simultanément stockés). Cependant, le volume de la mémoire interne est limité par des considérations purement constructives ainsi que par la nécessité de réduire le temps demandé pour en extraire les codes. C'est pourquoi, dans les C. A. N. modernes, *la mémoire externe* (qui se présente parfois sous plusieurs formes), dont la construction est réalisée avec des moyens plus simples, a une grande capacité. Le plus souvent c'est une mémoire magnétique, sur tambours, ferrites ou bandes.

La mémoire externe n'est pas directement reliée à l'organe de calcul. Les codes qui s'y trouvent sont envoyés par groupes, conformément au programme, dans la mémoire active interne ; de la même façon les codes se trouvant dans la mémoire interne sont envoyés dans la mémoire externe. Le temps mis pour extraire les codes de la mémoire externe est relativement long ; cependant, le volume de cette mémoire (le nombre des codes) dépasse en général de beaucoup celui de la mémoire interne.

3. Organe de commande par programme.

L'organe de commande par programme possède un registre dans lequel prennent successivement place les codes de commande (instructions du programme à réaliser).

Le programme d'un problème englobe l'ensemble des instructions indispensables pour le résoudre, numérotées d'après leur répartition en mémoire interne, et les données de départ (dont les constantes peuvent être rencontrées au cours de la résolution du problème), qui sont numérotées en conséquence.

On note dans le programme l'instruction initiale à partir de laquelle le travail débute.

Pour résoudre automatiquement un problème en machine, en plus des instructions destinées au calcul, c'est-à-dire celles qui servent à élaborer, au moyen de l'organe de calcul, des codes-valeurs de fonctions à partir des codes-arguments correspondants, il est indispensable d'avoir des instructions *de commande*, par lesquelles se fait l'appel aux organes externes (d'entrée, de sortie, de mémoire externe) et ainsi s'exécute la suite des opérations prévues par le programme.

Le fonctionnement de l'organe de commande est déterminé, à chaque instant, par l'instruction qui se trouve dans son registre, et cesse par l'envoi dans ce registre du code de l'instruction suivante de l'organe de mémoire. C'est ce qui fait passer automatiquement la machine à l'exécution de l'instruction suivante.

Pour contrôler l'exécution du programme (ou l'une de ses parties) d'après des instructions isolées, les machines possèdent, en plus de la commande automatique, un système de commande manuelle, à l'aide de laquelle, en pressant sur un bouton de mise en marche, on exécute également l'instruction initiale, c'est-à-dire la mise en marche automatique de la machine.

Selon leur composition, les machines sont réparties en machines dans lesquelles l'ordre d'exécution des instructions est *séquentiel (ou naturel)* et machines dans lesquelles cet ordre est *imposé*.

Dans les machines du premier type, après la k-ième instruction (c'est-à-dire l'instruction conservée dans la k-ième cellule), la $(k + 1)$-ième instruction est exécutée (ou, autrement dit, la commande se transmet à la $(k + 1)$-ième instruction), etc.

Font exception les instructions dites *instructions de transfert de commande* ; ce sont celles qui, selon les résultats des calculs précédents, déterminent dans les emplacements correspondants du programme sur quelles parties de celui-ci il est nécessaire de continuer les calculs, et procèdent à l'exécution.

Remarquons que l'énumération successive des opérations numériques élémentaires nécessaire à la résolution d'un problème, conduirait à des programmes d'une longueur démesurée, dont l'élaboration seule ne demanderait pas moins de temps que l'exécution de tous les calculs à la main. En outre, un tel programme chargerait d'une façon démesurée les cellules de l'organe de mémoire. C'est pourquoi la tâche du programmeur est d'établir des programmes sur lesquels on puisse effectuer un grand nombre de calculs indépendants avec un nombre d'instructions relativement petit (grâce à un emploi répété des mêmes instructions, ce qui est facile à réaliser par le transfert de commande).

Dans les machines à ordre séquentiel, chacune des instructions de calcul doit contenir le code chiffré de l'opération en cours et les numéros des cellules sur les codes desquelles doit être effectuée l'opération et dans lesquelles doit être envoyé le résultat des calculs. Il n'est pas obligatoire, pour établir les instructions, de connaître le nombre sur lequel se fait l'opération ; il suffit d'indiquer dans l'instruction le numéro de la cellule — l'adresse — dans laquelle elle a été placée au moment de l'entrée en machine ou après une opération précédente. Ainsi, sans connaître les résultats des calculs précédents, on peut établir à l'avance, les instructions qui permettent d'effectuer sur eux des opérations, en indiquant dans les adresses les numéros des cellules, où ces résultats sont envoyés pour y être conservés.

Comme les adresses des instructions ne comportent pas les nombres, mais leurs numéros seulement, le nombre des positions binaires nécessaires pour coder les instructions peut être très réduit quand la capacité de la mémoire interne de l'organe de mémoire est très faible. En effet, pour coder chaque adresse on peut se contenter du nombre de positions n, si celui des cellules de l'organe de mémoire interne est $M \leqslant 2^n$.

Le nombre d'adresses, qui peuvent être mises dans une seule instruction, définit le type d'adressage de la machine.

Dans les machines à ordre imposé, chaque instruction comporte l'adresse de la cellule où l'instruction suivante est stockée.

D'après ce qui vient d'être dit, il est clair que les machines qui ont une seule adresse doivent travailler selon le principe de l'exécution séquentielle. En outre, il est évident que les machines de ce genre doivent avoir dans l'organe de calcul un ou plusieurs registres pour conserver les résultats intermédiaires. Les opérations arithmétiques se divisent alors en plusieurs phases, de façon que chaque opération élémentaire ne porte que sur l'une des grandeurs. Il s'ensuit que dans les machines à une adresse, la liste des opérations élémentaires doit contenir, par exemple, des opérations telles que :

« l'extraction d'un nombre d'une cellule α et son renvoi au registre de l'organe de calcul, celui-ci devant être au préalable remis à zéro » :

$$P_1 \alpha ;$$

« l'addition d'un nombre contenu dans la cellule α au contenu du registre de l'organe de calcul » :

$$P_2 \alpha ;$$

« l'extraction d'un nombre du registre du dispositif de calcul et son renvoi à la cellule α » :

$$P_3 \alpha ;$$

etc. Ainsi, par exemple, pour effectuer l'opération suivante : « additionner le nombre stocké dans la cellule α à un nombre conservé dans la cellule β et ranger le résultat dans la cellule γ », il convient de mettre dans le programme pour machine à une adresse les trois instructions :

$$P_1 \alpha ,$$

$$P_2 \beta ,$$

$$P_3 \gamma .$$

Dans une machine à trois adresses et à ordre naturel, on n'aura besoin que d'une seule instruction $P\alpha\beta\gamma$. D'autre part, pour additionner les nombres stockés dans les cellules $\alpha + 1$; $\alpha + 2$, ..., $\alpha + n$ et ranger le résultat dans la cellule γ, il faut $n + 1$ instructions dans une machine à une adresse et $n - 1$ dans une machine à trois adresses. En moyenne, les programmes pour machines à trois adresses sont donc à peu près deux fois plus courts que ceux qui leur correspondent pour machines à une adresse.

Un programme établi pour une machine déterminée (ayant un type d'adressage, un mode d'exécution des opérations et leur choix bien fixés) peut être,

sans difficultés particulières, refait à l'usage d'une autre machine. C'est pourquoi, dorénavant, nous exposerons les bases de la programmation pour des machines à trois adresses dans lesquelles les opérations s'accomplissent successivement. Nous ne serons donc plus tenus à une suite déterminée d'opérations élémentaires à effectuer en machine.

L'écriture des instructions pour les machines à trois adresses peut être présentée sous un aspect assez commode :

Code opération	Adresse 1	Adresse 2	Adresse 3

Ainsi, dans la machine à trois adresses « Stréla » (Flèche) qui a des instructions de 43 positions, six positions sont affectées au code d'opération, douze à chacune des adresses, une au contrôle (*).

Adresse 1	Adresse 2	Adresse 3	Position contrôle	Code d'opération
0 – 11	12 – 23	24 – 35	36	37 – 42

Examinons ce qui se passe dans une machine à trois adresses, dans laquelle les instructions se déroulent successivement quand on a admis dans l'organe de commande le code extrait d'une cellule de l'organe de mémoire de numéro n et qui représente une instruction à caractère numérique.

Le travail exigé pour exécuter cette instruction peut être divisé en quatre temps :

Premier temps : Le code opération du registre de l'organe de commande est transmis au commutateur des opérations et prépare l'organe de calcul à effectuer l'opération qui lui correspond.

Deuxième temps : Le contenu de la première adresse de l'instruction (A_1) est transmis au commutateur de l'organe de mémoire (interne) ; la cellule dont le numéro est indiqué en A_1 s'ouvre, ce qui permet à son contenu d'être envoyé sur le registre correspondant de l'organe de calcul.

Troisième temps : Le contenu de la cellule dont le numéro est indiqué en A_2 (dans la deuxième adresse) est envoyé sur le registre correspondant de l'organe de calcul.

Quatrième temps : L'organe de calcul élabore le résultat de l'opération ; le contenu de A_3 du registre de l'organe de commande est transmis au commutateur de l'organe de mémoire ; la cellule, dont le numéro est indiqué en A_3

(*) Le signe de contrôle sert à la vérification du programme.

de l'instruction à exécuter s'ouvre, et le code élaboré dans l'organe de calcul entre dans cette cellule (avec effacement préalable du code qui s'y trouvait) ; le contenu de l'instruction dont le numéro suit (code de la cellule de numéro $n + 1$) est envoyé sur le registre de l'organe de commande.

2. OPÉRATIONS ÉLÉMENTAIRES RÉALISABLES SUR LES C. A. N.

Comme on l'a montré, pour pouvoir résoudre les différents problèmes mathématiques ou logiques, on a prévu la structure de la machine de façon que celle-ci puisse exécuter un choix déterminé d'opérations élémentaires, sous forme d'instructions isolées. Selon le type d'adressage et les autres particularités de la machine, il est possible de fixer les différents assortiments d'opérations élémentaires dont on a besoin. Cependant, le fait que la structure des machines soit limitée par de tels choix allonge énormément les programmes, rend la programmation plus difficile et, par conséquent, encombre la mémoire de la machine et complique le processus d'entrée. C'est pourquoi on a l'habitude de constituer un choix d'opérations élémentaires assez large, en prenant au minimum celles qui sont indispensables.

Nous allons examiner les opérations élémentaires utilisées par les machines dont le code d'instruction a trois adresses. Il est possible de diviser en plusieurs groupes, d'après leurs destinations, les opérations qu'elles peuvent exécuter :

1) Opérations arithmétiques fondamentales.
2) Opérations numériques supplémentaires.
3) Opérations logiques.
4) Opérations d'appel aux organes externes.
5) Opérations sur les instructions (instructions de substitution d'adresse).
6) Opérations de transfert de commande.

1. Opérations arithmétiques fondamentales.

Ce sont l'addition, la soustraction, la multiplication et la division qui sont écrites sous forme des instructions suivantes :

1. *Addition* (+). L'instruction

+	α	β	γ

signifie « additionner le nombre, dont le numéro α est indiqué dans la première adresse A_1, avec le nombre dont le numéro β est indiqué en A_2, et mettre

le résultat dans la cellule de l'organe de mémoire dont le numéro γ est indiqué en A_3 ».

2. *Soustraction* ($-$). L'instruction

$-$	α	β	γ

signifie « soustraire du nombre, dont le numéro α est indiqué en A_1, le nombre dont le numéro β est indiqué en A_2, et mettre le résultat dans la cellule de l'organe de mémoire dont le numéro γ est indiqué en A_3 ». Dans les machines à virgule flottante, avant d'effectuer ces opérations on égalise les ordres des nombres (addition ou soustraction) ; le résultat de l'opération est normalisé et arrondi.

3. *Multiplication* ($\times$). L'instruction

$\times$	α	β	γ

signifie « multiplier le nombre, dont le numéro α est indiqué en A_1, par le nombre dont le numéro β est indiqué en A_2, et mettre le résultat dans la cellule de l'organe de mémoire dont le numéro γ est indiqué en A_3 ».

4. *Division* (:). L'instruction

:	α	β	γ

signifie « diviser le nombre, dont le numéro α est indiqué en A_1, par le nombre dont le numéro β est indiqué en A_2, et mettre le résultat dans la cellule de l'organe de mémoire dont le numéro γ est indiqué en A_3 ».

Les résultats des multiplications et des divisions sont arrondis et, de plus, ils sont normalisés dans les machines à virgule flottante. Sur certaines machines, on a prévu de sortir les résultats des additions sans les arrondir, ceux des multiplications avec un nombre double de positions et ceux des divisions accompagnés du reste.

Après n'importe quelle instruction de ce groupe, l'instruction dont le numéro suit est exécutée (la commande est transmise au moyen de l'organe de commande à l'instruction dont le numéro suit).

2. Opérations numériques supplémentaires.

Il s'agit par exemple de :

1. *Soustraction en valeur absolue* (| − |). L'instruction

| − |	α	β	γ

signifie « du module du nombre, dont le numéro α est indiqué en A_1, soustraire le module du nombre dont le numéro β est indiqué en A_2, et mettre le résultat dans la cellule dont le numéro γ est indiqué en A_3 ».

Dans les machines à virgule flottante le résultat de l'opération est normalisé.

2. *Minimum de deux nombres* (min). L'instruction

min	α	β	γ

signifie « de deux nombres, dont les numéros α et β sont indiqués en A_1 et A_2, choisir le plus petit et le mettre dans la cellule dont le numéro γ est indiqué en A_3 ».

3. *Minimum de deux nombres en valeur absolue* (| min |). L'instruction

| min |	α	β	γ

signifie « de deux nombres, dont les numéros α et β sont indiqués en A_1 et A_2, choisir le nombre qui a le plus petit module et le ranger dans la cellule dont le numéro γ est indiqué en A_3 ».

On peut envisager les instructions suivantes analogues aux deux dernières.

4. *Maximum de deux nombres* (max). L'instruction a la forme :

max	α	β	γ

5. *Maximum de deux nombres en valeur absolue* (| max |). La forme générale de l'instruction est :

| max |	α	β	γ

6. *Extraction de racine carrée* ($\sqrt{\ }$). L'instruction

$\sqrt{\ }$	α		β

signifie « extraire la racine carrée du nombre dont le numéro α est indiqué en A_1 et ranger le résultat dans la cellule β dont le numéro est indiqué en A_3 ». La deuxième adresse de l'instruction n'est pas employée.

La machine peut effectuer des opérations analogues à l'opération 6, telles que :

7. *Formation du module d'un nombre* (mod). L'instruction a la forme :

mod	α		β

8. *Détermination de la partie fractionnaire d'un nombre* ({ }).

{ }	α		β

9. *Détermination de la partie entière d'un nombre* ([ ]).

[ ]	α		β

C'est naturellement en machine à virgule flottante que les deux dernières instructions ont un sens.

10. *Décalage d'un nombre* ($\rightarrow n$). L'instruction

$\rightarrow n$	α	β	γ

signifie « décaler le nombre dont le numéro β est indiqué en A_2 d'un nombre de positions égal à l'ordre du nombre dont le numéro α est indiqué en A_1, à droite ou à gauche selon le signe de l'ordre du nombre α, et mettre le résultat dans la cellule γ dont le numéro est indiqué en A_3 ». Cette opération présente de l'intérêt pour les machines à virgule flottante.

11. *Décalage d'adresse vers la droite* ($\rightarrow$).

$\rightarrow$	α	β	γ

12. *Décalage d'adresse vers la gauche* ($\leftarrow$).

$\leftarrow$	α	β	γ

Les instructions de 11 et 12 ont la même signification que celle de l'opération 10, mais le décalage se fait sur un nombre de positions égal au nombre α placé en A_1, à droite dans l'opération 11 et à gauche dans la 12. A titre exceptionnel, la première adresse n'y joue pas son propre rôle, mais celui d'un nombre.

Sur certaines machines, l'utilisation d'adresses libres est prévue. Par exemple, on peut réussir les instructions 8 et 9 en envisageant que la partie entière du nombre se formera dans la cellule dont le numéro est indiqué en A_2 et la partie fractionnaire dans celle dont le numéro est indiqué en A_3.

13. *Addition cyclique* ($+ c$).

$+ c$	α	β	γ

L'addition 13 diffère d'une addition normale ; s'il y a un report dans la position supérieure (la plus à gauche), il est transféré dans la position inférieure (la plus à droite) du résultat. L'opération 13 est normalement utilisée pour contrôler l'exactitude des calculs et l'entrée des programmes. Comme dans le cas des instructions du premier groupe, la commande est transmise à l'instruction dont le numéro suit, après l'exécution de chaque opération prévue.

3. Opérations logiques (position par position).

Donnons quelques exemples d'opérations logiques qui permettent de résoudre beaucoup plus simplement les problèmes mathématiques ainsi que de programmer très facilement les problèmes logiques sur les C. A. N.

1. *Multiplication logique* (*position par position*)($\times L$). L'instruction

$\times L$	α	β	γ

signifie « multiplier (multiplication logique) le code du nombre dont le numéro α est indiqué en A_1 par le code du nombre dont le numéro β est indiqué en A_2, et mettre le résultat en A_3 ». Autrement dit, 0 ou 1 est enregistré dans chaque position de la cellule selon la présence ou l'absence de 0 dans les positions correspondantes des nombres α et β, à savoir :

α	β	γ
0	0	0
0	1	0
1	0	0
1	1	1

L'opération $\times L$ peut être utilisée pour séparer certaines positions du code donné, pour séparer, par exemple, l'une des adresses de l'instruction. A cet effet, il convient de faire la multiplication logique du nombre donné par le nombre qui a des « uns » aux emplacements des positions à séparer, des « zéros » sur les autres.

2. *Addition logique* (*position par position*) ($+ L$).

$+ L$	α	β	γ

La valeur des positions du nombre dans la cellule est déterminée par le tableau :

α	β	γ
0	0	0
0	1	1
1	0	1
1	1	0

L'opération $+ L$ peut être utilisée pour former un code à partir de ses parties préparées, pour former, par exemple, des instructions à partir d'adresses obtenues à l'aide de l'opération $\times L$.

3. *Contrôle de coïncidence* ($\sim$).

$\sim$	α	β	γ

Chaque position du code A_3 se détermine, position par position, selon les valeurs trouvées dans les positions correspondantes de A_1 et de A_2 d'après la règle :

α	β	γ
0	0	1
0	1	0
1	0	0
1	1	1

De cette façon, si les codes des positions situées dans les cellules α et β coïncident, un « 1 » apparaît dans la position correspondante de la cellule γ ; dans le cas contraire, c'est un « 0 ».

L'opération $\sim$ est commode pour trouver l'égalité de deux nombres (coïncidence des codes), car ce n'est qu'avec une coïncidence totale des codes qu'apparaissent dans toutes les positions de A_3 des « 1 », ce qui, en retour, servira de signe de coïncidence des codes.

4. *Négation logique* ($-L$).

$-.L$	α		γ

Chaque position du nombre situé dans la cellule γ porte 0 s'il y a 1 dans la position correspondante du nombre situé dans la cellule α et réciproquement. La seconde adresse de l'instruction n'est pas utilisée.

Après l'exécution de chacune des instructions de ce groupe, la commande passe à l'instruction suivante.

4. Opérations d'appel aux organes externes.

Ce sont les opérations d'entrée, d'inscription, de lecture des codes, d'impression et d'arrêt.

Les différents transferts de codes sont normalement des opérations de groupe et se produisent dans l'ordre des numéros croissants des cellules. De plus, on code le numéro initial et le numéro final des cellules qui interviennent dans l'opération, ou le numéro initial du groupe de cellules et leur

nombre. Le temps nécessaire pour exécuter les instructions dépend évidemment de la quantité de nombres à transmettre.

1. *Entrée des codes* (*Ec*).

Ec	*n*	$\alpha + 1$	

Le contenu de l'instruction *Ec* est le suivant : « introduire dans *n* cellules $\alpha + 1, \alpha + 2, \ldots, \alpha + n$ de la mémoire opération les codes correspondants issus de l'organe d'entrée (par exemple, de cartes perforées) ».

Il peut y avoir deux types d'appel à la mémoire externe : transfert des codes des cellules de l'organe mémoire interne vers l'organe mémoire externe, c'est « l'écriture », et vice versa, transfert des codes de l'organe mémoire externe vers les cellules correspondantes de l'organe mémoire interne, c'est « la lecture ». Dans les deux cas, la codification de l'opération s'opère en deux instructions.

2. *Ecriture* (*transfert des codes de l'organe mémoire interne vers l'organe mémoire externe*) (*Ea*, *Eb*).

Ea	*n*	α	γ
Eb	*E*	β	

Dans la première instruction on indique, par exemple, en A_1 le numéro du tambour ou de la bande de l'organe de mémoire externe sur lequel se fera l'écriture, en A_2 le numéro α de la cellule du tambour ou de la bande où elle peut débuter et en A_3 le numéro γ de la cellule de l'organe de mémoire interne où l'opération commence.

Dans la deuxième instruction, on indique en A_1 le type de l'opération, soit « écriture », en A_2 le numéro β du nombre dans la mémoire externe jusqu'auquel il est possible d'enregistrer ; on n'utilise pas le A_3 de la deuxième instruction.

3. *Lecture* (*transfert des codes de l'organe mémoire externe vers l'organe mémoire interne*) (*Ea*, *Eb*).

Ea	*n*	α	γ
Eb	*L*	β	

Les adresses ont ici la même signification que dans le cas précédent, sauf A_1 de la deuxième instruction où le type d'instruction indiqué est « lecture ».

Les résultats des calculs en cours doivent, avant d'être imprimés ou sortis par d'autres moyens, être convertis du code binaire en décimal. Sur certaines machines, cette opération s'effectue lorsqu'il y a appel à l'instruction correspondante (*).

4. *Impression* (*I*). L'instruction

I		*n*	α

signifie « imprimer les codes décimaux du groupe de nombres qui contiennent $(n + 1)$ nombres, commençant au numéro α indiqué en A_3 ».

Le fait qu'une machine ait plusieurs organes de sortie oblige à mettre dans la suite des opérations à effectuer un nombre correspondant d'instructions de type 4. Pour distinguer ces instructions, on peut utiliser la première adresse libre.

5. *Arrêt* (*Arr*).

ARR			

Une fois cette instruction effectuée, les calculs s'arrêtent.

Après l'exécution des opérations de ce groupe (sauf celle d'arrêt), la commande passe de même à l'instruction du programme dont le numéro suit.

5. Instructions de substitution d'adresse ou opérations sur les instructions.

La codification des instructions sous forme d'une suite de chiffres élargit énormément les possibilités de la programmation, c'est-à-dire permet, si c'est nécessaire, de transformer les instructions à l'aide d'opérations spéciales pendant les calculs.

Cependant, comme nous l'avons vu, par exemple dans les machines à virgule flottante, avant d'effectuer une addition ou une soustraction, les ordres des nombres sont alignés et le résultat de l'opération est normalisé. Les positions affectées à la codification de l'ordre du nombre y sont normalement utilisées pour codifier le type d'opération de l'instruction. Et comme sur les instructions il n'est pas nécessaire de procéder à l'alignement des ordres et

(*) Dans les circuits des machines où la conversion des nombres en décimal n'a pas été envisagée, on prévoit au cours des programmes avant l'appel à l'organe de sortie, un appel à un programme standard spécial de conversion du binaire en décimal.

à la normalisation du résultat, il est commode d'avoir, dans la suite des opérations élémentaires à exécuter, des instructions spéciales pour les opérations qui les concernent. Lorsque le contenu d'une mémoire est considéré comme une instruction et se divise en code d'opération et en adresses, et que les changements d'adresses sont liés au changement d'une ou plusieurs de ces parties, on a affaire aux instructions de substitution d'adresse.

1. *Addition d'instructions* ($\oplus$).

$\oplus$	K_1	α	K_2

2. *Soustraction d'instructions* ($\ominus$).

$\ominus$	K_1	α	K_2

Le contenu de la première instruction est le suivant : « ajouter, sans égalisation préalable des ordres, au nombre dont le numéro K_1 est indiqué en A_1, le nombre dont le numéro α est indiqué en A_2 ; mettre le résultat, sans le normaliser, dans la cellule dont le numéro K_2 est indiqué en A_3 ». La seconde instruction est analogue à la première.

Il est évident qu'une opération exécutable par une instruction de type 2 peut l'être par une instruction de type 1, si dans la cellule α on place le nombre négatif correspondant.

L'augmentation de A_1, A_2, A_3 de 1, 2, 3, ... consiste à ajouter au nombre qui représente l'instruction un nombre déterminé d'unités dans les positions correspondantes (par exemple pour la machine *Kiev* ces nombres sont respectivement égaux à 2^{-16}, 2^{-28}, 2^{-40}, etc.). Par la suite, nous désignerons ces nombres par « 1 » A_1, « 2 » A_1, « 1 » A_1 + « 2 » A_2, etc., ou même nous les noterons sous la forme :

–	1	–	–
–	2	–	–
–	1	2	–

Comme dans le cas des groupes d'instructions précédents, après exécution des instructions 1 et 2, la commande passe à l'instruction du programme dont le numéro suit.

6. Opérations de transfert de commande.

Les problèmes intéressants à traiter en machine sont ceux dont la résolution peut être divisée en calculs qui se répètent entièrement ou partiellement, ce qui permet d'utiliser à plusieurs reprises une seule instruction ou un groupe d'instructions ; il s'agit de problèmes pour lesquels l'on peut établir des programmes dont le nombre d'instructions est bien inférieur au nombre des opérations élémentaires indispensables pour les résoudre.

En général la possibilité de répéter certaines étapes de calcul en fonction du résultat des opérations est une propriété qu'ont presque tous les problèmes mathématiques (et logiques) complexes. Cependant les instructions des groupes 1 à 5, introduites précédemment (de façon plus ou moins complète), sont nettement insuffisantes pour la programmation de ces problèmes, puisqu'après l'exécution de chacune de ces instructions on passe à celle dont le numéro suit. Ces groupes doivent être complétés par des instructions qui permettent de dévier de l'ordre ordinaire des calculs et transmettent la commande à une instruction qui ne suit pas directement celle qui vient d'être exécutée, ou à l'une de deux (ou de plusieurs) instructions destinées à choisir la suite à donner au calcul d'après des *symboles* déterminés — conditions formulées auparavant ou découlant des résultats des calculs ; aussi est-il indispensable d'avoir la possibilité de vérifier l'exécution des conditions correspondantes — et, d'après cela, à accomplir le transfert de la commande au secteur du programme spécifié. Puisqu'au titre de symboles de ce genre on applique dans différentes machines telles ou telles propriétés des résultats des calculs, des machines différentes peuvent disposer d'instructions différentes, dites de *transfert conditionnel de la commande* (sous une forme ou plusieurs formes).

En outre, remarquons que chaque programme concret est en grande partie déterminé, non seulement par le choix de la méthode employée pour résoudre le problème, mais aussi par les particularités de la machine dont on se sert : en premier lieu son type d'adressage et sa liste d'instructions, parmi lesquelles les instructions de transfert de commande jouent un rôle fondamental.

Citons quelques instructions qui, dans différentes machines, réalisent le transfert de la commande.

1. *Transfert de commande en fonction de la comparaison de deux nombres, compte tenu des signes* ($\leqslant$).

$\leqslant$	α	β	k

Cela signifie « comparer le nombre qui se trouve dans la cellule α à celui de la cellule β et, si le premier est supérieur au second, passer à l'instruction

suivante ; s'il lui est inférieur ou égal, passer à l'instruction dont le numéro est indiqué dans la troisième adresse ».

Ainsi le signe de la différence des nombres placés dans les cellules α et β (*) est pris ici comme signe servant de critère au choix des transferts de la commande à l'instruction dont le numéro est indiqué en A_3 de l'instruction considérée et à celle dont le numéro suit.

2. *Transfert de commande en fonction de la comparaison des modules de deux nombres* ($| \leqslant |$).

$\lvert \leqslant \rvert$	α	β	k

Cette opération est analogue à l'opération 1 à cette différence près qu'elle ne compare pas les nombres, mais leurs modules.

3. *Transfert de commande en fonction de l'égalité de deux nombres* ($=$).

$=$	α	β	k

Autrement dit, comparer le nombre situé dans la cellule α au nombre situé dans la cellule β et, si ces deux nombres sont égaux, passer la commande à l'instruction indiquée dans la troisième adresse, sinon passer à l'exécution de l'instruction suivante.

4. *Transfert de commande en fonction de l'inégalité de deux nombres* ($\neq$).

$\neq$	α	β	k

Cette opération est analogue à l'opération 3 à la seule différence que, dans le cas de l'inégalité des nombres situés en α et β, la commande est transmise à l'instruction indiquée par la troisième adresse, sinon à l'instruction dont le rang est le suivant. Pour transmettre la commande on emploie aussi des instructions qui utilisent d'autres relations entre les nombres ($<$, $>$, etc.).

Notons que si un branchement vers deux instructions différentes k_1 et k_2 est nécessaire, comme il est impossible de les placer toutes les deux immédia-

(*) Nous estimerons toujours dans ce cas-là qu'à la suite du calcul de nombres égaux nous obtenons « -0 » (zéro négatif) $a - a = -0$.

tement après l'instruction en cours, on peut avoir recours au procédé suivant :

$\leqslant$	α	β	k_1
$\leqslant$			k_2

Ainsi, il peut y avoir branchement, par exemple, lorsque l'on fait appel à deux sous-programmes standards différents à la suite des résultats des calculs (cf. chap. III, § 6).

Les instructions 1-4 peuvent aussi être utilisées pour effectuer le transfert obligatoire d'une commande (transfert inconditionnel) à l'instruction indiquée dans la troisième adresse (*). A cet effet, il suffit d'indiquer le même nombre dans les instructions 1, 2, 3 en A_1 et A_2 ; le plus pratique, dans ce cas, est d'utiliser une cellule *nulle* (**). Les instructions de transfert inconditionnel de la commande prennent alors la forme suivante :

$\leqslant$			k
$\lvert \leqslant \rvert$			k
$=$			k

Ainsi, dans les instructions de transfert inconditionnel, les positions A_1 et A_2 restent par définition inemployées. Donc, si on utilise l'opération 4 en qualité d'opération de transfert inconditionnel, il convient de placer dans l'une des adresses (A_1 ou A_2) une cellule qui contienne un nombre α indubitablement différent de + 0 (par exemple une cellule contenant une constante différente de zéro).

5. *Transfert de commande en fonction d'un symbole* (*opération de transfert conditionnel*).

Lorsque sur certaines machines (par exemple : Strela, Oural) on exécute des opérations arithmétiques, logiques et autres en fonction du résultat des calculs, il s'élabore dans un organe de mémoire spécial (le basculeur) un symbole qui y est conservé jusqu'à l'instruction suivante, pour être utilisé si celle-ci

(*) On peut montrer que pour programmer n'importe quel problème, il suffit d'avoir dans la liste des instructions une seule du groupe 6 à côté de la série indispensable de celles des groupes de 1 à 5.

(**) Nous supposons ici que la cellule *nulle* contient « + 0 ».

est une instruction de transfert de commande. Ce symbole peut être le signe moins résultant d'une addition ou d'une soustraction, un résultat de division plus grand que « un » (en machine à virgule flottante), la formation de « uns » dans toutes les positions du résultat d'une opération logique, etc.

Ces machines possèdent une instruction de transfert conditionnel de commande d'après un symbole (*TCC*).

TCC	k_1	k_2	

Cela signifie « transmettre la commande à l'instruction indiquée dans la première adresse si, à la suite de l'opération précédente, il y a un symbole dans le basculeur, sinon à l'instruction indiquée dans la seconde adresse ». La troisième adresse n'est pas utilisée.

Si dans la première et la deuxième adresse on place le numéro d'une même instruction k, l'instruction *TCC* exécutera le transfert inconditionnel de la commande à l'instruction k.

TCC	k	k	

Ainsi, pour utiliser l'opération de transfert de commande d'après un symbole sous la forme d'une opération de transfert conditionnel, il faut prévoir l'élaboration du symbole correspondant dans l'instruction précédente ; mais si cette opération s'exécute comme une opération de transfert inconditionnel, cela reviendra au même qu'un symbole s'élabore ou non dans l'instruction précédente.

L'opération de transfert conditionnel est commode : elle permet lorsqu'il le faut, au moment d'un branchement, à la suite des résultats d'opérations précédentes (ce qui est déterminé par la présence d'un symbole dans l'instruction précédente), de passer à l'exécution des instructions nécessaires.

6. *Transfert de commande en fonction du signe d'un nombre.*

Dans certains problèmes, la détermination des branchements du programme peut être liée aux signes d'un même nombre.

Dans ce cas, lorsqu'on utilise l'opération de transfert de commande de type *TCC*, il est indispensable avant chaque branchement, de répéter dans l'instruction qui précède l'opération *TCC* la composition de l'instruction utilisant le signe du nombre α (par exemple, en répétant chaque fois l'instruction : additionner le nombre α à 0, si l'on prend comme symbole le signe « – » à la suite de l'addition).

En considérant cela, il est commode d'inclure à la place de *TCC* dans la liste des opérations celle de transfert de commande d'après un nombre (*TCN*) :

TCN	α	k_1	k_2

Le contenu de cette opération est le suivant : « si le nombre dont le numéro α est indiqué en A_1 a le signe « + », la commande est transmise à l'instruction indiquée dans la deuxième adresse ; si le signe est « − », elle l'est à l'instruction dont le numéro est indiqué dans la troisième adresse ».

On peut, dans ce cas, réaliser l'opération de transfert inconditionnel vers l'instruction k au moyen de l'opération *TCN* à la première adresse dans laquelle est placé le code « + 0 » (la cellule nulle est utilisée) :

TCN		k	

ou en plaçant dans les deuxième et troisième adresses de cette instruction le numéro de la cellule k :

TCN	α	k	k

à laquelle doit être transmise la commande. Dans ce dernier cas, le contenu de la première adresse est inutilisé.

CHAPITRE III

PROGRAMMATION ÉLÉMENTAIRE

Le traitement des problèmes sur C. A. N. est précédé d'un travail qui expose les motifs et fixe le choix d'une méthode numérique, ce qui permet d'établir des formules de calcul et un programme de travail, compte tenu des particularités d'une machine existante : c'est en somme le but de la programmation.

Les deux choses sont réciproquement liées : l'établissement de programmes complexes exige de fréquentes révisions des formules employées et de la méthode choisie pour résoudre le problème ; par ailleurs, la nécessité de résoudre sur C. A. N. des problèmes dont l'exécution exige souvent des centaines de milliers et même des millions d'opérations élémentaires oblige à élaborer des procédés de programmation rationnels.

Dans ce chapitre, nous allons traiter des éléments de programmation directe.

1. PROGRAMMATION DIRECTE

1. Entrée du programme et sortie des résultats.

Pour exécuter les calculs d'un programme donné, il faut l'introduire dans les emplacements de la mémoire réservés à cet effet. Comme il y a deux types de mémoire interne — la mémoire opérative et la mémoire passive — il existe dans la machine deux procédés d'entrée matérialisés par :

— la codification du programme sur cartes ou rubans perforés,

— la réalisation du programme par un ensemble de fiches de connexion.

L'entrée en mémoire externe est le plus souvent réalisée au moyen de la mémoire interne à l'aide d'instructions de la forme :

Ea	*n*	α	γ
Eb	*M*	β	

qui accomplissent le transfert au groupe de codes, introduits précédemment, des cellules γ, $\gamma + 1$, ... de l'organe mémoire interne vers la n-ième zone de la mémoire externe de α à β.

L'entrée en mémoire opération s'effectue en général automatiquement au moyen de l'organe de commande et d'un sous-programme d'entrée spécial, qui est composé, dans le cas le plus simple, d'une seule instruction :

Ec	$\alpha + 1$	n	

réalisant l'entrée dans les cellules $\alpha + 1$, $\alpha + 2$, ..., $\alpha + n$ de l'organe de la mémoire opération, des codes correspondants issus, par exemple, des cartes perforées.

L'instruction d'entrée sur carte perforée (dans ce cas, cette carte est placée en tête) ou sur le tableau de commande à main, est introduite à son tour dans la mémoire de la machine à l'aide de l'organe d'entrée, puis la machine commence le travail automatique : le traitement du problème.

Il est aussi possible d'entrer manuellement le programme dans la mémoire opération ; les codes sont rassemblés sur le tableau de commande et introduits dans les cellules correspondantes de l'organe mémoire. Mais cela occasionne une grande perte de temps et on ne le fait que pour apporter les corrections nécessaires au cours de la mise au point du programme.

Comme on l'a déjà indiqué, la mémoire passive se compose en général de deux parties : la partie invariable comportant les codes que l'on rencontre souvent au cours des problèmes (ils sont appelés constantes ou sous-programmes standards), et la partie variable qui peut être rapidement composée, le contenu de ses cellules pouvant être, pour chaque problème envisagé, préparé à l'avance pendant le temps de fonctionnement de la machine (généralement en un autre lieu).

Puisque les machines travaillent en code binaire, les données numériques des problèmes (mais non les instructions du programme, ni les codes auxiliaires) doivent, avant d'entrer, être converties de décimal en binaire. Sur certaines machines, ces opérations sont prévues dans leur circuit ; sur d'autres, elles se produisent à l'aide d'un transfert normal des données numériques ou de sous-programmes spéciaux, qui convertissent le binaire en décimal et vice versa. Dans ce dernier cas, il est prévu dans le programme de faire passer chaque nombre par un sous-programme de conversion avant de le sortir. A l'entrée et à la sortie des nombres chaque chiffre décimal est transcrit en binaire (cf. chap. I, § 4).

Par exemple, la sortie des résultats sur bande, sous forme de signes numériques, est prévue en une suite déterminée qui permet de les déchiffrer d'une façon simple.

Par la suite, nous n'envisagerons plus la question de conversion du décimal

en binaire et vice versa, ainsi que celle de l'entrée du programme en machine : ces opérations dépendent beaucoup de la constitution des machines.

2. Programmation d'après des formules.

Les problèmes les plus simples à programmer sont naturellement ceux dans lesquels les calculs se font d'après des formules qui se divisent en opérations élémentaires réalisables par des instructions séparées. La programmation de ces problèmes (ou de ces parties de problèmes) n'offre pas de difficultés particulières et conduit au choix le plus rationnel de la suite des opérations élémentaires et à la répartition des données du problème dans la mémoire, c'est-à-dire à la répartition des données initiales, des constantes auxiliaires, des résultats intermédiaires et finals, ainsi qu'à celle des instructions du programme, dans les cellules de la mémoire.

Pour faciliter l'élaboration des programmes, on utilise généralement des désignations alphanumériques : des adresses conventionnelles. Ainsi, par exemple, pour numéroter les instructions du programme, on peut utiliser les désignations suivantes : $N + 1, N + 2, \ldots$; pour numéroter les données initiales et les constantes auxiliaires, les symboles suivants :

$$\alpha_1, \alpha_2, \ldots \quad \text{ou} \quad \alpha + 1, \alpha + 2, \ldots$$

$$\gamma_1, \gamma_2, \ldots \quad \text{ou} \quad \gamma + 1, \gamma + 2, \ldots$$

Dans ce dernier cas, on souligne la répartition des codes dans les cellules dont les numéros se suivent, ce qui est particulièrement important dans une série de cas. Les cellules, dans lesquelles on conserve les résultats intermédiaires des calculs, les « cellules de travail », seront désignées par $\omega + 1$, $\omega + 2, \ldots$; $r + 1, r + 2, \ldots$; ou $\omega_1, \omega_2, \ldots$; $r_1, r_2, \ldots$ si la répartition séquentielle de ces grandeurs dans les cellules de l'organe mémoire n'est pas essentielle.

L'introduction de ces désignations permet d'établir des programmes, même si au cours de la programmation — avant la répartition de l'information correspondante dans les cellules existantes — le nombre des cellules indispensables à la répartition des instructions, des résultats intermédiaires ou définitifs, etc., ainsi que les numéros des cellules libres de l'organe mémoire sont encore inconnus. Après avoir définitivement établi un programme sous une forme alphabétique, il faut donner une valeur numérique à chacune des lettres $N, \alpha, \gamma, \omega, r$; ce qui permet de résoudre la question de la répartition du programme dans les cellules existantes.

Pour programmer les problèmes les plus simples d'après des formules données, on doit choisir avant tout un ordre déterminé, c'est-à-dire, répartir le calcul en une suite d'opérations élémentaires. Dans ce cas, comme pour les

calculs ordinaires, il convient d'essayer de diminuer autant que possible le nombre des opérations élémentaires, en utilisant à cette fin des transformations identiques des formules. Lorsqu'on résout un problème, la diminution du nombre des opérations élémentaires réduit le nombre des instructions du programme et, partant, abrège le temps de calcul et décharge la mémoire.

D'autre part, il est indispensable de choisir l'ordre des opérations en tenant compte qu'en machine les calculs se font avec *arrondi* et qu'il faut donc essayer de réduire ces erreurs.

Remarquons que la succession des opérations exécutées peut être choisie de plusieurs façons grâce à la possibilité que l'on a de permuter les opérations élémentaires ; en outre, le nombre des cellules de travail peut changer.

Pour effectuer les opérations élémentaires, il faut avoir dans les cellules correspondantes de la mémoire les nombres qui rentrent dans ces opérations. Pour économiser les cellules de la mémoire, il faut placer les résultats des calculs intermédiaires dans les cellules de la mémoire qui contiennent des résultats déjà utilisés et dont on n'aura plus besoin.

Notons qu'en employant les opérations qui sont effectuées en machine on peut construire des programmes pour calculer des grandeurs qui ne sont pas données par des expressions analytiques.

***Exemple* 1.** *Calcul d'une fonction homographique*

$$y = \frac{ax + b}{cx + d}. \tag{1}$$

Nous ferons le calcul de y à l'aide de la suite des opérations élémentaires suivantes :

$$\begin{aligned} &1)\ A_1 = ax; \quad 2)\ A = A_1 + b; \quad 3)\ B_1 = cx; \\ &4)\ B = B_1 + d; \quad 5)\ y = \frac{A}{B}. \end{aligned} \tag{2}$$

Pour cela, nous mettrons en mémoire les nombres a, b, c, d, x, respectivement dans les cellules $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$.

Chacune des opérations élémentaires de (2) est exécutée par une instruction. De plus, nous placerons les résultats des calculs intermédiaires dans les cellules de travail ω_1 et ω_2 (celles de l'organe de mémoire opération).

Ainsi le calcul d'après les formules (2) est réalisé par la suite d'instructions suivantes :

$N + 1$	$\times$	α_1	α_5	ω_1	$A_1 = ax$
$N + 2$	$+$	ω_1	α_2	ω_1	$A = A_1 + b$
$N + 3$	$\times$	α_3	α_5	ω_2	$B_1 = cx$
$N + 4$	$+$	ω_2	α_4	ω_2	$B = B_1 + d$
$N + 5$	:	ω_1	ω_2	ω_1	$y = \frac{A}{B}$

Notons que l'ordre d'exécution des instructions est déterminé par la suite choisie d'opérations (2). Pour terminer les calculs, il est indispensable de prévoir, sur la machine, un arrêt automatique qui est effectué par l'instruction :

ARR			

Dans cet exemple, nous avons utilisé pour la mémorisation de A la cellule dans laquelle se trouvait le résultat de A_1 et pour celle de B, la cellule dans laquelle il y avait B_1 (pour ces grandeurs il faut des cellules différentes, car elles participent au calcul de y). Le résultat cherché peut être placé dans l'une des deux cellules, qui contient A ou B. Conformément à ce qui a été dit dans le paragraphe précédent, on utilise deux cellules de travail : ω_1 et ω_2.

Pour l'entrée automatique des instructions du programme dans la mémoire opération de la machine, prenons comme instruction initiale du programme, l'instruction d'entrée :

Ec	N	$N + 6$	

S'il est nécessaire de donner le résultat des calculs — la valeur y que l'on obtient dans la cellule ω_1 — il convient de placer l'instruction d'impression :

I			ω_1

avant l'instruction d'arrêt. Ainsi, le programme du calcul de y selon la formule (1) peut être enregistré sous la forme :

Programme 1

Nombres

Numéro de la cellule	Contenu initial	Contenu final
α_1	x	
α_2	b	
α_3	c	
α_4	d	
α_5	x	
ω_1		y
ω_2		

Instructions

Numéro de l'instruction	Code de l'opération	A_1	A_2	A_3	Résultat de l'opération
$N + 1$	$\times$	α_1	α_5	ω_1	ax
$N + 2$	$+$	ω_1	α_2	ω_1	$ax + b$
$N + 3$	$\times$	α_3	α_5	ω_2	cx
$N + 4$	$+$	ω_2	α_4	ω_2	$cx + d$
$N + 5$	:	ω_1	ω_2	ω_1	$\frac{ax + b}{cx + d} = y$
$N + 6$	I			ω_1	
$N + 7$	ARR				

De plus, il est sous-entendu qu'avant la mise en route du programme, les codes numériques correspondants sont introduits dans les cellules $\alpha_1, \alpha_2, \ldots, \alpha_5$.

Pour l'exécution d'un programme donné, ces cellules peuvent être aussi bien celles de la mémoire constante que celles de la mémoire opération. Admettons que $\alpha_1 = N + 8$, $\alpha_2 = N + 9, \ldots, \alpha_5 = N + 12$ et prenons pour instruction initiale l'instruction d'entrée :

N	*Ec*	12	$N + 1$

Notre programme s'écrira alors sous la forme :

Programme 1 *a.*

N	*Ec*	12	$N + 1$		
$N + 1$	×	$N + 8$	$N + 12$	ω_1	
$N + 2$	+	ω_1	$N + 9$	ω_1	
$N + 3$	×	$N + 10$	$N + 12$	ω_2	Codes des instructions
$N + 4$	+	ω_2	$N + 11$	ω_2	
$N + 5$	:	ω_1	ω_2	ω_1	
$N + 6$	*I*			ω_1	
$N + 7$	*ARR*				
$N + 8$	*a*				
$N + 9$	*b*				Codes numériques
$N + 10$	*c*				
$N + 11$	*d*				
$N + 12$	*x*				

Par la suite, nous utiliserons l'écriture des programmes sous la première des formes indiquées, en omettant l'instruction d'entrée. Lorsqu'on écrit ainsi, sans indication particulière, la première instruction du tableau compte comme instruction initiale du programme.

Exemple 2. *Opérations sur des nombres complexes.*

On peut fixer dans l'organe mémoire tout nombre complexe $a + ib$ en utilisant deux cellules, par exemple α et $\alpha + 1$ qui contiennent respectivement la partie réelle et la partie imaginaire.

Pour exécuter les opérations arithmétiques sur des nombres complexes, il faut calculer les parties réelles et imaginaires du résultat. Prenons comme exemple une division de nombres complexes.

$$(a + bi) : (c + di) = x + yi\,, \tag{3}$$

où

$$x = \frac{ac + bd}{c^2 + d^2}\,; \qquad y = \frac{bc - ad}{c^2 + d^2}\,. \tag{4}$$

Nous calculerons x et y en nous servant des opérateurs élémentaires suivants :

$$1)\ A = c^2\,;\quad 2)\ B = d^2\,;\quad 3)\ C = A + B\,;\quad 4)\ A_1 = ac\,;$$

$$5)\ A_2 = bd\,;\quad 6)\ A_3 = A_1 + A_2\,;\quad 7)\ x = \frac{A_3}{C}\,; \tag{5}$$

$$8)\ B_1 = bc\,;\quad 9)\ B_2 = ad\,;\quad 10)\ B_3 = B_1 - B_2\,;$$

$$11)\ y = \frac{B_3}{C}\,.$$

Supposons que les nombres complexes $a + ib$ et $c + id$ soient contenus respectivement dans les cellules de l'organe mémoire.

$$\alpha_1\,,\quad \alpha_1 + 1 \quad \text{et} \quad \alpha_2\,,\quad \alpha_2 + 1\,.$$

Dans le programme traité, on utilise les quatre cellules de travail

$$\omega_1, \omega_1 + 1, \omega_2, \omega_3\,;$$

dans les deux premières nous obtiendrons le résultat cherché.

Programme 2.

Nombres		Instructions				
α_1	a	$N+1$	×	α_2	α_2	ω_1	c^2
α_1+1	b	$N+2$	×	α_2+1	α_2+1	ω_1+1	d^2
α_2	c	$N+3$	+	ω_1	ω_1+1	ω_1+1	c^2+d^2

Nombres			Instructions				
α_2+1	d		$N+4$	$\times$	α_1	α_2	ω_1	ac
ω_1		x	$N+5$	$\times$	α_1+1	α_2+1	ω_2	bd
ω_1+1		y	$N+6$	$+$	ω_1	ω_2	ω_1	$ac+bd$
ω_2			$N+7$	$:$	ω_1	ω_1+1	ω_1	x
ω_3			$N+8$	$\times$	α_1+1	α_2	ω_2	bc
			$N+9$	$\times$	α_1	α_2+1	ω_3	ad
			$N+10$	$-$	ω_2	ω_3	ω_2	$bc-ad$
			$N+11$	$:$	ω_2	ω_1+1	ω_1+1	y
			$N+12$	ARR				

***Exemple* 3.** *Calcul des valeurs d'un polynôme à une seule variable.*

$$f(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_n . \tag{6}$$

Examinons comme exemple les deux méthodes de calcul d'une valeur d'un polynôme. Nous sommes convaincus que suivant la suite d'opérations choisies il faut un nombre d'opérations élémentaires différent ; ce qui fait que nous utiliserons un nombre d'instructions et un nombre de cellules opération différents.

1) Calculons d'abord toutes les puissances x^k ($k = 1, 2, \ldots, n$), ensuite tous les produits de la forme $a_{n-k} x^k$ (des valeurs x^k par les coefficients correspondants a_{n-k}) et enfin la somme de ces produits à l'aide de la suite d'opérations élémentaires suivantes :

$$1)\ A_1 = x^2 ;\quad 2)\ A_2 = x^3 ; \ldots ;\quad n-1)\ A_{n-1} = x^n ;$$

$$n)\ A_n = a_0 x^n ;\quad n+1)\ A_{n+1} = a_1 x^{n-1} ; \ldots ;\quad 2n-1)\ A_{2n-1} = a_{n-1} x ;$$

$$2n)\ A_{2n} = a_0 x^n + a_1 x^{n-1} ; \ldots ;$$

$$3n-1)\ A_{3n-1} = a_0 x^n + a_1 x^{n-1} + \cdots + a_n = f(x) .$$

De plus, on a besoin de $3n - 1$ instructions pour les calculs.

2) Représentons préalablement le polynôme $f(x)$ sous la forme :

$$f(x) = \{[(a_0 x + a_1) x + \cdots + a_{n-2}] x + a_{n-1}\} x + a_n . \tag{7}$$

Nous ferons les calculs d'après cette formule (algorithme de Hörner). La suite des opérations sera, dans ce cas; la suivante :

$$1)\ A_1 = a_0 x\ ;\quad 2)\ A_2 = A_1 + a_1\ ;\quad 3)\ A_3 = A_2 x\ ;$$
$$4)\ A_4 = A_3 + a_2\ ;\ \ldots\ ;\quad 2n)\ A_{2n-1} + a_n\ . \tag{8}$$

Remarquons que, outre la diminution des opérations élémentaires à exécuter ($2n$ au lieu de $3n - 1$), donc du nombre des instructions du programme, les calculs d'après la formule (7) ont un autre avantage : ils permettent une plus grande exactitude.

En vertu de ces considérations, nous n'élaborerons un programme que pour le deuxième des procédés de calcul de $f(x)$.

Soit le stockage des valeurs :

$$a_0, a_1, a_2, \ldots, a_n, x$$

respectivement dans les cellules

$$\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_n, \alpha_{n+1}\ .$$

Conformément aux formules (8), pour conserver les résultats intermédiaires, il faut se limiter à une cellule de travail de la mémoire ω, dans laquelle on obtiendra le résultat cherché. En effet, pour calculer chacune des valeurs de A_i, on n'utilise qu'un seul nombre A_{i-1}, que l'on ne rencontrera plus dans les calculs ultérieurs.

Programme 3.

Nombres			Instructions				
α_0	a_0	$N+1$	$\times$	α_0	α_{n+1}	ω	$a_0 x$
α_1	a_1	$N+2$	$+$	ω	α_1	ω	$(a_0 x + a_1)$
α_2	a_2	$N+3$	$\times$	ω	α_{n+1}	ω	$(\ )\ x$
α_3	a_3	$N+4$	$+$	ω	α_2	ω	$(\ )\ x + a_2$
$\vdots$		$\vdots$	$\vdots$				
α_n	a_n	$N+2n-1$	$\times$	ω	α_{n+1}	ω	$\{\ \}\ x$
α_{n+1}	x	$N+2n$	$+$	ω	α_n	ω	$\{\ \}\ x + a_n$
ω		$f(x)$ $N+2n+1$	ARR				

***Exemple* 4.** *Séparation des parties entière et fractionnaire d'un nombre.*

$$E(x)\,, \quad \{\, x \,\}\,.$$

Supposons que dans une machine à virgule flottante, l'ordre le plus grand possible p_0 d'un nombre x ne dépasse pas le nombre des chiffres de la partie numérique du nombre. Soit, dans ce cas, la représentation du nombre sous la forme :

$$x = 2^p.0, x_1\, x_2 \ldots x_n\,,$$

où n est le nombre des positions binaires de la partie numérique du nombre.
Alors

$$\begin{aligned} E(x) &= 0\,, \quad \text{si} \quad p \leqslant 0 \\ &= x_1\, x_2 \ldots x_p\,, \quad \text{si} \quad p > 0\,. \end{aligned}$$

Ainsi, dans une machine à virgule flottante, la séparation de la partie entière du nombre revient à séparer les p premières positions de sa partie numérique, si $p > 0$. Lorsqu'on fait la supposition précédente, on peut exécuter la séparation de $E(x)$ et $\{\, x \,\}$ selon les formules

$$\begin{aligned} E(x) &= (x + 2^{n-1}) - 2^{n-1}\,; \\ \{\, x \,\} &= x - E(x)\,. \end{aligned}$$

En effet, dans l'addition et la soustraction les nombres sont réduits au même ordre et ensuite le résultat de l'opération est normalisé.
Nous avons

$$\begin{aligned} x + 2^{n-1} &= 2^p.0, x_1\, x_2 \ldots x_n + 2^{n-1} \\ &= 2^n.0, \underbrace{0 \ldots 0}_{n-p}\, x_1\, x_2 \ldots x_p + 2^n.0, 1 \\ &= 2^n.0, 10 \ldots x_1\, x_2 \ldots x_p\,. \end{aligned}$$

Puis

$$\begin{aligned} (x + 2^{n-1}) - 2^{n-1} &= 2^n.0,10 \ldots 0\, x_1 \ldots x_p - 2^n.0, 1 \\ &= 2^n.0, 0 \ldots 0\, x_1 \ldots x_p = 2^p.0, x_1 \ldots x_p \underbrace{0 \ldots 0}_{n-p}\,. \end{aligned}$$

Conformément aux formules proposées, nous placerons dans les cellules de l'organe mémoire α_1 et α_2 les valeurs de x et de 2^{n-1}.

Les valeurs déterminées $E(x) + \{\, x \,\}$ seront placées dans les cellules de l'organe de mémoire ω_1 et ω_2.

Programme 4. *Calcul de* $E(x)$.

Nombres

α_1	x	
α_2	2^{n-1}	
ω_1		$E(x)$

Instructions

$N+1$	$+$	α_1	α_2	ω_1
$N+2$	$-$	ω_1	α_2	ω_2
$N+3$	*ARR*			

Programme 5. *Calcul de* $\{ x \}$.

Nombres

α_1	x	
α_2	2^{n-1}	
ω_1		$E(x)$
ω_2		$\{ x \}$

Instructions

$N+1$	$+$	α_1	α_2	ω_1
$N+2$	$-$	ω_1	α_2	ω_1
$N+3$	$-$	α_1	ω_1	ω_2
$N+4$	*ARR*			

Si l'on cherche à calculer seulement $\{ x \}$ on peut envoyer ce résultat dans la cellule ω_1.

CONCLUSIONS

1. Pour obtenir le résultat avec exactitude, il faut que les formules de calcul soient choisies de telle sorte que les erreurs d'arrondi qui entachent le résultat des calculs soient aussi petites que possible.

2. Pour réduire le temps de calcul, au cours d'un programme basé sur des formules, il faut choisir l'ordre des opérations de façon à réduire le plus possible le nombre d'opérations élémentaires, c'est-à-dire le nombre des instructions et celui des cellules de travail utilisées.

3. Afin de décharger la mémoire, il faut placer les calculs intermédiaires dans les cellules de l'organe mémoire qui contiennent des résultats qui ne seront plus utilisés dans le calcul ultérieur.

EXERCICES

Dans les exercices proposés, les valeurs qui figurent dans la partie droite des formules indiquées sont considérées comme données dans l'organe mémoire.

1. Etablir un programme pour calculer

$$y = \frac{ax^2 + bx + c}{a_1 x^2 + b_1 x + c_1}$$

selon les données $a, b, c, a_1, b_1, c_1, x_n$, en se limitant à dix instructions (l'instruction d'arrêt comprise) et à deux cellules opération.

2. Etablir un programme pour calculer

$$(x + \mathrm{i}y) = (a + \mathrm{i}b)(c + \mathrm{i}d) .$$

3. Etablir un programme pour calculer

$$(x + \mathrm{i}y) = (a + \mathrm{i}b)^4 ,$$

en se limitant à dix instructions (l'instruction d'arrêt comprise) et à trois cellules opération.

4. Etablir un programme pour séparer, dans une machine à virgule fixe, les k premières positions d'un nombre.

5. Etablir un programme pour séparer les parties entière et fractionnaire d'un nombre, en machine à virgule fixe, après la k-ième position.

2. PROCESSUS ARBORESCENTS

Dans de nombreux cas, les valeurs des données initiales ou des résultats intermédiaires déterminent le circuit numérique sur lequel les calculs continuent. La question de connaître la direction dans laquelle il faut diriger les calculs ultérieurs peut toujours être ramenée au contrôle de l'exécution d'une condition logique exprimant une propriété particulière des grandeurs employées dans les calculs.

De tels circuits numériques sont naturellement appelés *arborescents* et chaque direction du calcul, *branche du calcul*.

Les programmes *arborescents* correspondent aux processus numériques arborescents.

La programmation des processus arborescents est liée à la répartition dans la mémoire des instructions qui effectuent les calculs suivant toutes les branches de l'aiguillage et de celles qui exécutent le transfert de commande.

Notons que, si le processus numérique se ramifie en plus de deux branches, la condition correspondante peut, à son tour, être divisée en une série de conditions dont chacune sépare une branche de celles qui restent. C'est pourquoi la programmation des processus arborescents est une question qui peut être ramenée à la construction de programmes arborescents dichotomiques.

On peut envisager chaque condition donnée comme une proposition variable, prenant la valeur « vrai » si elle est satisfaite, « faux » dans le cas

contraire. On peut comparer les deux valeurs des propositions variables avec les valeurs correspondantes des variables logiques, c'est-à-dire avec les deux valeurs : 1 si la condition est réalisée, ou 0 si elle ne l'est pas.

Pour conserver les valeurs des variables logiques correspondantes, on incorpore aux machines un registre spécial à position unique (machines *Strela*, *Oural*), ou on utilise une position (celle qui sert habituellement pour le signe) des cellules de la mémoire (machine *Kiev*). Dans ce dernier cas, le contenu des autres positions de la cellule utilisée pour placer la variable logique est indifférent. Autrement dit, si la cellule α est destinée à la conservation de la valeur de la variable logique, celle-ci est choisie de telle sorte que le signe positif qui y est conservé corresponde à la valeur « vrai » de la condition et le signe négatif à la valeur « faux ».

De ce point de vue, *Strela* et *Oural* sont des machines que l'on peut considérer comme ayant un registre unique pour conserver les valeurs logiques : après l'exécution de chaque opération arithmétique ou logique, un symbole s'y élabore en fonction des résultats — valeur de la variable qui se conserve jusqu'à l'exécution de l'instruction suivante. C'est pourquoi, pour réaliser dans ces machines le nœud du branchement, il faut au moins deux instructions consécutives : la première forme définitivement la valeur de la variable logique (élabore le symbole approprié, selon la terminologie adoptée) et la seconde représente l'instruction de transfert conditionnel selon la valeur de la variable logique.

Selon la présence ou l'absence, dans le choix des opérations élémentaires, de telles ou telles opérations de transfert conditionnel de commande, les conditions, dont l'exécution détermine la marche des opérations ultérieures selon des programmes arborescents, doivent être transformées pour se rapprocher de la forme appropriée.

Citons les conditions logiques testées par les opérations de transfert de commande, examinées au paragraphe 2, chapitre II.

1. *Opération de transfert conditionnel de la commande en fonction du résultat de la comparaison de deux nombres, compte tenu des signes* ($\leqslant$).

$\leqslant$	α	β	k

Elle contrôle la condition logique

$$P_1 = \begin{cases} 1, & \text{si} \quad (\alpha) \leqslant (\beta), \\ 0, & \text{si} \quad (\alpha) > (\beta), \end{cases}$$

et transfert la commande à l'instruction k, indiqué en A_3, si la condition

logique est satisfaite (P_1 égal à 1), et à l'instruction en séquence dans le cas contraire. Ici (α) signifie le contenu de la cellule α.

2. *Opération de transfert conditionnel de la commande en fonction de la comparaison des modules de deux nombres* ($| \leqslant |$).

$\lvert \leqslant \rvert$	α	β	k

Elle contrôle la condition logique

$$P_2 = \begin{cases} 1, & \text{si} \quad |(\alpha)| \leqslant |(\beta)|, \\ 0, & \text{si} \quad |(\alpha)| > |(\beta)|, \end{cases}$$

et transfère la commande à l'instruction k, désignée en A_3, dans le premier cas, et à l'instruction qui suit l'opération de transfert conditionnel dans le cas contraire.

3. *Opération de transfert de commande en fonction de la comparaison de deux nombres* ($=$).

$=$	α	β	k

Elle contrôle la condition logique

$$P_3 = \begin{cases} 1, & \text{si} \quad (\alpha) = (\beta), \\ 0, & \text{si} \quad (\alpha) \neq (\beta), \end{cases}$$

et transfère la commande à l'instruction k, si $P_3 = 1$, et à l'instruction suivante dans le cas contraire.

4. *Opération de transfert conditionnel de la commande en fonction de l'inégalité de deux nombres* ($\neq$).

$\neq$	α	β	k

Elle contrôle la condition logique

$$P_4 = \begin{cases} 1, & \text{si} \quad (\alpha) \neq (\beta), \\ 0, & \text{si} \quad (\alpha) = (\beta), \end{cases}$$

et transfère la commande à l'instruction k si $P_4 = 1$, et à l'instruction en séquence dans le cas contraire.

5. *Opération de transfert de commande conditionnel en fonction d'un symbole* (*TCS*).

TCS		k_1	k_2

Elle contrôle la condition logique

$$P_5 = \begin{cases} 1, & \text{si} \quad \omega = 1, \\ 0, & \text{si} \quad \omega = 0, \end{cases}$$

et transmet la commande à l'instruction k_1, indiquée en A_2, si dans l'instruction précédente est élaboré le symbole ω, et à l'instruction k_2, indiquée en A_3, dans le cas contraire.

6. *Opération de transfert conditionnel de commande en fonction du signe d'un nombre* (*TCN*).

TCN	α	k_1	k_2

Elle contrôle la condition logique

$$P_6 = \begin{cases} 1, & \text{si} \quad (\alpha) \leqslant -0, \\ 0, & \text{si} \quad (\alpha) > -0, \end{cases}$$

et transmet la commande à l'instruction k_1, si $P_6 = 1$, à l'instruction k_2, si $P_6 = 0$.

***Exemple* 1.** *Formation du module d'un nombre.*

Sur certaines machines, les opérations qui permettent de prendre le module d'un nombre n'existent pas dans l'ensemble des opérations élémentaires (*).

Etablissons un programme qui permette de prendre le module du nombre x se trouvant dans la cellule α, avec mise du résultat dans la cellule ω. En outre,

(*) On suppose aussi que la position du signe ne participe pas à la multiplication logique.

faisons en sorte qu'indépendamment du signe de x, après avoir obtenu $|x|$, la commande soit transmise à la même instruction. Cette dernière condition est nécessaire lorsque $|x|$ est un résultat intermédiaire.

Suivant le signe du nombre x, on doit transmettre dans la cellule ω :

a) le contenu de la cellule α, si le nombre x est positif ;
b) le contenu de la cellule α, avec le signe contraire, si x est négatif.

Etablissons ce programme pour une machine à opération de transfert conditionnel de commande dépendant d'une comparaison.

Dans le premier cas (*a*), le programme qui forme le module du nombre se compose des instructions :

k	+		α	β
$k+1$	$\leqslant$			p

et dans le deuxième cas :

n	−		α	β
$n+1$	$\leqslant$			p

La $(k + 1)$-ième et la $(n + 1)$-ième opérations de transfert inconditionnel sont introduites pour que, après l'opération qui forme le module d'un nombre, on passe à l'exécution de l'instruction p.

Pour savoir quelle direction prendre dans l'arbre, on utilise la condition logique :

$$P = \begin{cases} 0, & \text{si} \quad 0 > (\alpha), \\ 1, & \text{si} \quad 0 \leqslant (\alpha). \end{cases}$$

L'instruction :

$\leqslant$		α	k

à laquelle il faut attribuer le numéro $n - 1$, contrôle la réalisation de cette

condition et exécute les transferts de commande correspondants. Nous obtiendrons le programme :

$n-1$	$\leqslant$		α	k
n	$-$		α	ω
$n+1$	$\leqslant$			p
k	$+$		α	ω
$k+1$	$\leqslant$			p
p				

Si l'on pose $p = k + 1$, la dernière instruction peut être omise. En outre, on peut poser $k = n + 2$.

Programme 6.

Nombres

α	x
ω	$\lvert x \rvert$

Instructions

$n-1$	$\leqslant$		α	$n+2$
n	$-$		α	ω
$n+1$	$\leqslant$			$n+3$
$n+2$	$+$		α	ω
$n+3$				

Exemple 2. *Résolution d'une équation du second degré.*

$$ax^2 + bx + c = 0 .$$

Pour trouver les racines de l'équation, nous utiliserons la formule :

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} .$$

Selon le signe de la différence $D = b^2 - 4ac$, les racines de l'équation seront ou réelles ou imaginaires. Dans le premier cas, nous considérerons comme résultat des calculs les racines de l'équation, dans le second cas, les parties réelles et imaginaires des racines complexes sorties sur l'imprimante.

Nous conviendrons de noter l'existence des racines réelles en imprimant le signe conventionnel « + 0 », avant la sortie des racines de l'équation.

Dans les cellules $\alpha_1, \alpha_2, \alpha_3, \beta_1$ nous mettrons respectivement les grandeurs $a, b, c, 4$.

Quelles que soient les racines, pour les calculer on a besoin des grandeurs : $-b$, $D = b^2 - 4ac$, $2a$, que nous placerons respectivement dans les cellules $\omega_1, \omega_2, \omega_3$.

La partie commune des calculs, précédant l'aiguillage, est réalisée par les instructions suivantes :

n	×	α_2	α_2	ω_1	b^2
$n+1$	×	α_1	α_3	ω_2	ac
$n+2$	×	ω_2	β_1	ω_2	$4ac$
$n+3$	−	ω_1	ω_2	ω_2	D
$n+4$	−		α_2	ω_1	$-b$
$n+5$	+	α_1	α_1	ω_3	$2a$

Si $D > 0$, il convient d'exécuter les instructions suivantes :

k	I				Imprimer « + 0 »
$k+1$	$\sqrt{}$	ω_2		ω_2	$\sqrt{D}$
$k+2$	+	ω_1	ω_2	ω_4	$-b+\sqrt{D}$
$k+3$	−	ω_1	ω_2	ω_2	$-b-\sqrt{D}$
$k+4$	:	ω_4	ω_3	ω_1	x_1
$k+5$	:	ω_2	ω_3	ω_2	x_2
$k+6$	I			ω_1	
$k+7$	I			ω_2	
$k+8$	ARR				

Si $D < 0$, il convient d'exécuter les instructions suivantes :

m	$-$		ω_2	ω_2	$-D$
$m+1$	$\sqrt{\ }$	ω_2		ω_2	$\sqrt{-D}$
$m+2$	:	ω_1	ω_3	ω_1	$-b/2a$
$m+3$	:	ω_2	ω_3	ω_2	$\sqrt{-D}/2a$
$m+4$	I			ω_1	
$m+5$	I			ω_2	
$m+6$	ARR				

Ici, la m-ième instruction est le changement de signe de la grandeur D (dans le cas donné, négative), ce qui permet d'en extraire les racines en exécutant les instructions suivantes.

Le contrôle de la condition :

$$P = \begin{cases} 0, & \text{si} \quad D < 0, \\ 1, & \text{si} \quad D \geqslant 0, \end{cases}$$

et le transfert de la commande aux branches des calculs suivants peuvent être exécutés par l'instruction :

$k-1$	$\leqslant$	ω_2		m

Puisque les quatre dernières instructions de chacune des branches coïncident, elles peuvent être mises dans une partie de programme commune, réalisée à la suite des calculs suivant l'une ou l'autre des branches de l'aiguillage. Dans ce but, après la $(k+4)$-ième instruction, il convient de placer l'instruction de transfert inconditionnel près de l'instruction $m+3$.

$k+5$	$\leqslant$			$m+3$

Supposons $k-1 = n+6$; $m = k+6 = n+13$ et nous avons le programme :

Programme 7.

Nombres

α_1	a
α_2	b
α_3	c
β	4
ω_1	
ω_2	
ω_3	
ω_4	

résultat

de travail

Instructions

n	$\times$	α_2	α_2	ω_1	b^2
$n+1$	$\times$	α_1	α_3	ω_2	ac
$n+2$	$\times$	ω_2	β	ω_2	$4\,ac$
$n+3$	$-$	ω_1	ω_2	ω_2	D
$n+4$	$-$		α_2	ω_1	$-b$
$n+5$	$+$	α_1	α_1	ω_3	$2\,a$
$n+6$	$\leqslant$	ω_2		$n+13$	
$n+7$	I				
$n+8$	$\sqrt{\ }$	ω_2		ω_2	$\sqrt{D}$
$n+9$	$+$	ω_1	ω_2	ω_4	$-b+\sqrt{D}$
$n+10$	$-$	ω_1	ω_2	ω_2	$-b-\sqrt{D}$
$n+11$	:	ω_4	ω_3	ω_1	x_1
$n+12$	$\leqslant$			$n+16$	
$n+13$	$-$		ω_2	ω_2	$-D$
$n+14$	$\sqrt{\ }$	ω_2		ω_2	$\sqrt{-D}$
$n+15$	:	ω_1	ω_3	ω_1	$-b/2\,a$
$n+16$	:	ω_2	ω_3	ω_2	$x_2\left(\frac{\sqrt{-D}}{2\,a}\right)$
$n+17$	I			ω_1	
$n+18$	I			ω_2	
$n+19$	ARR				

Lorsqu'il y a une opération de transfert conditionnel d'après un nombre *TCN*, l'aiguillage du programme peut être réalisé par l'instruction :

$n + 6$	TCN	ω_2	$n + 7$	$n + 13$

où la grandeur D se trouve dans la cellule de l'organe mémoire ω_2. Après avoir changé de place les adresses 2 et 3 de cette instruction, nous pouvons mettre la deuxième des branches de l'aiguillage immédiatement après l'opération *TCN*. Le nombre d'instructions du programme reste le même.

Lorsqu'il y a une opération de transfert conditionnel de la commande d'après un symbole, il faut une instruction supplémentaire pour former ce dernier, par exemple :

(*)	+		ω_2	ω_2

si le symbole est élaboré à la suite de l'apparition du signe « moins » au cours d'une addition. Il est indispensable de placer cette instruction immédiatement avant l'opération *TCS*. Cependant, après avoir noté que la grandeur D apparaît à la suite de l'instruction $n + 3$ et que les instructions $n + 3$ et $n + 5$ peuvent être interchangées, on peut éviter de compléter le programme par l'instruction supplémentaire (*). En outre, il est sous-entendu que le symbole se forme aussi à l'apparition d'un signe moins, à la suite d'une soustraction.

***Exemple* 3.** *Calcul des valeurs de la fonction*

$$z = \begin{cases} xy & \text{si } x^2 + y^2 \leqslant 1 \quad \text{(Branche I) ;} \\ \sqrt{x} + \dfrac{x + y}{x - y} & \text{si } x^2 + y^2 > 1 \text{ et } x \leqslant 0 \quad \text{(Branche III) ;} \\ \sqrt{2x} + \dfrac{2x + y}{2x - y} & \text{si } x^2 + y^2 > 1 \text{ et } x > 0 \quad \text{(Branche III) .} \end{cases} \tag{9}$$

Relativement à la construction de notre fonction, le calcul se ramifie en trois branches.

Répartissons les grandeurs x, y, 1 dans les cellules α, β, γ.

Les calculs suivant la première branche seront exécutés par les instructions :

n	$\times$	α	β	ω	z
$n + 1$	ARR				

Les calculs suivant la seconde branche, par les instructions :

k	$+$	α	β	ω	$x + y$
$k + 1$	$-$	α	β	ω_1	$x - y$
$k + 2$	$:$	ω	ω_1	ω	$\frac{x + y}{x - y}$
$k + 3$	$\sqrt{\ }$	α		ω_1	$\sqrt{x}$
$k + 4$	$+$	ω	ω_1	ω	z
$k + 5$	ARR				

Les calculs selon la troisième branche peuvent être exécutés à l'aide des instructions k, $k + 1, \ldots, k + 5$ si l'on place d'avance dans la cellule α la grandeur $2x$ à la place de x :

p	$+$	α	α	α	$2x$
$p + 1$	$+$	α	β	ω	$2x + y$
$p + 2$	$-$	α	β	ω_1	$2x - y$
$p + 3$	$:$	ω	ω_1	ω	$\frac{2x + y}{2x - y}$
$p + 4$	$\sqrt{\ }$	α		ω_1	$\sqrt{2x}$
$p + 5$	$+$	ω	ω_1	ω	z
$p + 6$	ARR				

Le passage au calcul de la branche I correspond à l'exécution de la condition suivante :

$$P_1 = \begin{cases} 1, & \text{si} \quad R \leqslant 1, \\ 0, & \text{si} \quad R > 1, \end{cases}$$

où $R = x^2 + y^2$.

Calculons d'avance la grandeur R :

N	$\times$	α	α	ω	x^2
$N + 1$	$\times$	β	β	ω_1	y^2
$N + 2$	$+$	ω	ω_1	ω	R

On peut alors effectuer le contrôle de l'exécution de la condition P_1 et le transfert de commande correspondant :

$N+3$	$\leqslant$	ω	γ	n

Le passage au calcul de la branche II correspond à l'exécution de l'instruction :

$$P_2 = \begin{cases} 1, & \text{si} \quad x \leqslant 0\,, \\ 0, & \text{si} \quad x > 0\,, \end{cases}$$

si $R > 1$ et $x \leqslant 0$, c'est-à-dire lorsque la condition $\overline{P}_1 \wedge P_2$ est satisfaite, $\wedge$ étant le signe de la multiplication logique ($\overline{P}_1$ étant la négation de P_1). L'instruction placée immédiatement après l'instruction $N + 3$ réalise le contrôle de cette condition et les transferts de commande correspondants :

$N+4$	$\leqslant$	α		k

après laquelle il faut placer le programme de calcul sur la branche III.

Ainsi, si l'on admet :

$$N + 5 = p\,,$$

le problème peut être résolu par l'ensemble des instructions :

$$\begin{gathered} N, N+1, \ldots, N+4, N+5 = p, N+6, \ldots, N+11\,, \\ n, n+1, k, k+1, \ldots, k+5\,, \end{gathered} \tag{A}$$

ou

$$\begin{gathered} N, N+1, \ldots, N+4, N+5, \ldots, N+11, \\ k, k+1, \ldots, k+5, n, n+1\,. \end{gathered} \tag{B}$$

Le programme peut être abrégé au profit des instructions de contenu correspondant, qui se trouvent sur les branches II et III. Plaçons en (A) au lieu du groupe d'instructions $N + 6, \ldots, N + 11$ l'instruction de transfert inconditionnel

$N+6$	$\leqslant$			k

ou, au lieu du groupe d'instructions k, $k + 1, \ldots, k + 5$, l'instruction de transfert inconditionnel

k	$\leqslant$			$N + 6$

Dans ce cas, le résultat sera le même si l'on supprime l'instruction k et que, dans l'instruction $N + 4$, à l'adresse A 3, on place $N + 6$ au lieu de k. Notre programme aura alors la forme suivante :

Programme 8.

Nombres

α	x
β	y
γ	1
ω	z
ω_1	

Instructions

N	$\times$	α	α	ω
$N + 1$	$\times$	β	β	ω_1
$N + 2$	$+$	ω	ω_1	ω
$N + 3$	$\leqslant$	ω	γ	$N + 12$
$N + 4$	$\leqslant$	α		$N + 6$
$N + 5$	$+$	α	α	α
$N + 6$	$+$	α	β	ω
$N + 7$	$-$	α	β	ω_1
$N + 8$	$:$	ω	ω_1	ω
$N + 9$	$\sqrt{\ }$	α		ω_1
$N + 10$	$+$	ω	ω_1	ω
$N + 11$	*ARR*			
$N + 12$	$\times$	α	β	ω
$N + 13$	*ARR*			

Si le résultat des calculs de z suivant les formules (9) est intermédiaire, la commande à la fin du programme doit être retransmise à la même instruction, par exemple $N + 13$, indépendamment de l'aiguillage. Dans ce cas, au lieu

de la $(N + 11)$-ième instruction d'arrêt, il convient de placer l'instruction de transfert inconditionnel près de l'instruction $N + 13$ à partir de laquelle se répartit la sortie suivante du programme.

CONCLUSIONS

Pour construire des programmes arborescents, il est indispensable de savoir ce qui suit :

1) La répartition dans la mémoire opération des instructions qui effectuent les calculs suivant toutes les branches de l'aiguillage est simplifiée, si l'on utilise des désignations alpha-numériques pour numéroter les cellules de l'organe mémoire.

2) Dans le but d'économiser les cellules de l'organe mémoire, lorsqu'on construit des programmes à branches séparées, il faut tendre à une répartition matérielle dans la mémoire telle qu'elle permette d'assigner le plus grand nombre possible d'instructions aux parties du programme communes à plusieurs branches.

3) Il convient de transformer les conditions de l'aiguillage de sorte que leur contrôle s'effectue facilement à l'aide des instructions de transfert conditionnel de commande.

4) Lorsque l'aiguillage a plus de deux branches, il convient de sélectionner une suite de conditions, au cours de laquelle une branche se séparera successivement de toutes les autres.

EXERCICES

1. Etablir un programme pour calculer l'intégrale

$$\int_a^b x^\alpha \, dx = \begin{cases} \left[\dfrac{1}{\alpha + 1} x^{\alpha+1}\right]_a^b, & \text{si} \quad \alpha \neq -1, \\ [\ln x]_a^b, & \text{si} \quad \alpha = -1. \end{cases}$$

2. Etablir le programme pour calculer les valeurs de la fonction

$$y = \begin{cases} 2x^2 + \sqrt{x} & \text{si} \quad x > 0, \\ 3x^3 + \sqrt{-x} & \text{si} \quad x \leqslant 0. \end{cases}$$

3. Modifier le programme de l'exercice 2, par l'impression de tous les résultats par une seule instruction.

3. PROCESSUS CYCLIQUES

Comme on l'a vu précédemment, la possibilité d'effectuer un grand nombre d'opérations à l'aide d'un petit nombre d'instructions est une des conditions d'utilisation efficace de la machine à commande programmée. La matérialisation de ce principe est fondée sur le fait que chaque algorithme peut être normalement représenté sous la forme d'une suite d'opérations répétées. Ce genre de processus de calcul s'appelle *cyclique* et les secteurs répétés, *des cycles*.

Le nombre total de cycles indispensables aux calculs peut être, soit indiqué d'avance, soit déterminé au cours des calculs. Au cas où le nombre des cycles est connu d'avance, on peut faire un programme détaillé dans lequel chaque cycle aura son secteur. Cependant, cette façon de résoudre le problème sur C. A. N. exigerait beaucoup de temps pour établir les instructions du programme et conduirait à un encombrement de la mémoire.

Généralement, on ne peut pas établir un programme développé pour les processus cycliques qui n'ont pas un nombre de cycles connu d'avance. Lorsqu'il y a une bonne répartition des données initiales et des résultats des calculs intermédiaires dans l'organe mémoire, on peut établir, pour les processus cycliques, des programmes dits cycliques : ils consistent en instructions indispensables à l'exécution d'un des cycles et en instructions d'importance secondaire.

Le groupe d'instructions qui, au cours de l'exécution des calculs, se répète un certain nombre de fois consécutivement est appelé *programme cyclique* ou *cycle*.

Pour construire un programme cyclique, on sélectionne de façon convenable la condition logique dont l'exécution détermine la fin du processus cyclique, et on complète le programme par des instructions qui réalisent son *test* et le transfert de la commande au début d'un nouveau cycle ou aux instructions suivantes, selon la conclusion de ce *test*. En outre, on peut inclure dans le programme du cycle, des instructions qui préparent le remplissage approprié de la mémoire pour permettre le passage d'un cycle à l'autre.

Lorsqu'on choisit des conditions logiques, on utilise généralement une grandeur qui varie d'un cycle à l'autre, soit qu'elle apparaisse dans le calcul, soit qu'on l'introduise spécialement. En outre, dans l'ensemble des opérations élémentaires que la machine doit effectuer, il faut tenir compte d'instructions de transfert conditionnel de commande.

1. Cycles itératifs.

Dans le cas le plus simple, le processus cyclique se réduit à l'application aux données initiales

$$x_1^0, x_2^0, \ldots, x_n^0$$

de certaines formules :

$$x_1^1 = f_1(x_1^0, \ldots, x_n^0) ; \ldots ; x_n^1 = f_n(x_1^0, \ldots, x_n^0) ; \tag{10}$$

les résultats des calculs selon ces formules, $x_1^1, x_2^1, \ldots, x_n^1$, sont pris à leur tour comme données initiales pour les calculs selon les mêmes formules :

$$x_1^2 = f_1(x_1^1, \ldots, x_n^1) ; \ldots ; x_n^2 = f_n(x_1^1, \ldots, x_n^1), \text{ etc.}$$

Le processus se répète un certain nombre de fois, connu ou déterminé au cours des calculs. L'obtention de $x_1^{k+1}, \ldots, x_n^{k+1}$ à partir de $x_1^k, \ldots, x_n^k$ représente un cycle de calculs. De tels processus cycliques seront appelés *itératifs*.

Pour établir un programme selon un schéma de calcul donné, nous choisirons des cellules spéciales pour les données initiales et, après les calculs selon les formules (10), nous y placerons les grandeurs $x_1^1, \ldots, x_n^1$. Puis, le groupe des instructions qui exécutent les calculs $x_1^1, \ldots, x_n^1$ d'après $x_1^0, \ldots, x_n^0$, peut être utilisé pour calculer $x_1^2, \ldots, x_n^2$ d'après $x_1^1, \ldots, x_n^1$; $x_1^3, \ldots, x_n^3$ d'après $x_1^2, \ldots, x_n^2$, etc.

L'établissement d'un programme cyclique prend fin par l'introduction dans le programme d'instructions qui assurent le transfert de la commande à l'instruction initiale et aux calculs suivants (le nombre de fois indispensable) ou le transfert de la commande à l'instruction arrêt.

On économise des instructions, si, au cours des calculs, on réussit à placer les grandeurs $x_1^{k+1}, \ldots, x_n^{k+1}$ dans les cellules dans lesquelles se trouvent les grandeurs $x_1^k, \ldots, x_n^k$.

Prenons un exemple.

***Exemple* 1.** *Calcul et impression de la table des carrés des termes d'une progression arithmétique.*

Pour faire la table des carrés des termes de la progression arithmétique

$$a, \quad a + c, \quad a + 2c, \ldots$$

de a à $a + cN$, a étant le premier terme de la progression et c la raison de la progression, nous adopterons l'ordre des calculs suivant. En supposant le premier terme de la progression a donné, nous l'élèverons au carré et imprimerons le résultat. Nous préparerons le second terme de la progression ; à cet effet, nous ajouterons la raison c au nombre a ; nous élèverons au carré le nombre obtenu $a + c$ et nous imprimerons ; nous ajouterons au nombre $a + c$ de nouveau la raison, etc. Par cela même, le calcul des carrés des membres d'une progression arithmétique se divise en cycles : formation du carré d'un nombre de la progression arithmétique, impression de ce carré, formation du terme suivant de la progression.

Pour programmer le premier cycle, il est indispensable d'avoir dans l'organe mémoire les nombres a et c que nous placerons respectivement dans les cellules

de l'organe mémoire α_1 et α_2. Une fois que l'on a calculé le terme suivant $a + c$, on le placera dans la cellule α_1 (en effaçant le terme de la progression précédente). Nous placerons le carré du terme dans la cellule opération ω.

Le premier cycle est accompli par une succession d'instructions

					Cycle I	Cycle II
k	$\times$	α_1	α_1	ω	a^2	$(a + c)^2 \ldots$
$k + 1$	I			ω		
$k + 2$	$+$	α_1	α_2	α_1	$a + c$	$a + 2c \ldots$

Il reste à remarquer que le contenu des cellules de l'organe mémoire est préparé par le premier cycle pour le cycle suivant ; c'est donc que les mêmes instructions peuvent exécuter le second cycle et les suivants.

Dans l'exemple cité, le nombre total des cycles est connu d'avance et égal à N.

Pour achever l'établissement d'un programme cyclique, il faut choisir une condition logique qui permette de déterminer la fin du processus.

a) *Soit dans la machine une opération de transfert de commande suivant le résultat de la comparaison* « $\leqslant$ » *ou celui de la comparaison des modules* « $| \leqslant |$ ».

Il est clair, d'après cette condition que le passage d'un cycle à l'autre doit s'effectuer tant que le nombre $a + rc$ (r est le nombre de cycles), obtenu dans la cellule α_1, ne dépasse pas le nombre $a + Nc$. Nous placerons celui-ci dans la cellule de l'organe mémoire α_3.

Supposons

$$P = \begin{cases} 0 & \text{si} \quad a + rc > a + Nc\,, \\ 1 & \text{si} \quad a + rc \leqslant a + Nc\,. \end{cases}$$

Les instructions :

$\leqslant$	α_1	α_3	k

ou

$\lvert \leqslant \rvert$	α_1	α_3	k

permettent d'effectuer le contrôle de la condition P et le transfert de la commande : elles transmettent la commande au cycle suivant jusqu'à ce que dans la cellule α_1 apparaisse le nombre $a + (N + 1)\,c$; après cela il faut placer l'instruction d'arrêt.

Programme 9.

Nombres

α_1	a
α_2	c
ω	
α_3	$a + Nc$

$(a + rc)^2$

Instructions

k	$\times$	α_1	α_1	ω
$k + 1$	I			ω
$k + 2$	$+$	α_1	α_2	α_1
$k + 3$	$\leqslant$	α_1	α_3	k
$k + 4$	ARR			

Examinons en détail le travail du programme. Le travail commence à la k-ième instruction :

DÉBUT DU PREMIER CYCLE

1) Calcul du carré a^2 et envoi dans la cellule ω.
2) Impression de a^2.
3) Calcul du terme suivant $a + c$ de la progression arithmétique et envoi dans la cellule α_1.
4) Comparaison du nombre $a+c$ et du nombre $a+cN$, et si $a+c \leqslant a+cN$ la commande se transmet à l'instruction k.

DÉBUT DU SECOND CYCLE

5) Calcul du carré $(a + c)^2$ et envoi dans la cellule ω.
6) Impression de $(a + c)^2$.
7) Calcul du nombre suivant $a + 2\,c$ de la progression arithmétique et envoi dans la cellule α_1.
8) Comparaison du nombre $a+2\,c$ et du nombre $a+cN$, et si $a+2\,c \leqslant a+cN$, la commande se transmet à l'instruction k et ceci jusqu'à l'apparition dans la cellule α_1 du nombre $a + c(N + 1)$.

DÉBUT DU N-ième CYCLE

$4N - 3$) Calcul de $(a + cN)^2$ et envoi dans la cellule ω.
$4N - 2$) Impression de $(a + cN)^2$.
$4N - 1$) Calcul de $a + c\ (N + 1)$ et envoi dans la cellule α_1.

4 N) Comparaison de $a + c\,(N + 1)$ avec $a + cN$, et puisque

$$a + c\,(N + 1) > a + cN\,,$$

passage à l'exécution de l'instruction $k + 4$, c'est-à-dire :

4 $N + 1$) Arrêt.

Ainsi, à l'aide de cinq instructions, on accomplira $4\,N + 1$ opérations élémentaires.

En outre, on n'utilise pour les calculs que quatre mémoires.

Au début de chaque cycle, il est possible de spécifier s'il faut calculer le cycle suivant ou arrêter les calculs. Au cas où le nombre $a + c\,(N + 1)$ apparaît dans α_1, le transfert de la commande doit se faire à arrêt.

Conformément à cela, le nombre $a + c\,(N + 1)$ se trouve dans la cellule de l'organe mémoire α_3, et l'instruction

$k - 1$	$\leqslant$	α_3	α_1	$k + 4$

qui se trouve au début du cycle, effectue le transfert de commande à arrêt.

Pour passer au cycle suivant, le cycle est complété par l'instruction dite « instruction de transfert obligatoire »

$k + 3$	$\leqslant$			$k - 1$

qui transmet la commande à la $(k - 1)$-ième instruction. Tant que dans la cellule α_1 se trouve un nombre inférieur au nombre $a + c\,(N + 1)$ (de la cellule α_3), les cycles s'effectuent à la suite les uns des autres ; la première instruction après l'apparition du nombre $a + c\,(N + 1)$ dans la cellule α_1 conduit à l'instruction :

ARR			

Notre programme sera alors écrit sous la forme suivante :

Nombres

α_1	a
α_2	c
α_3	$a + (N + 1)c$

Instructions

$k - 1$	$\leqslant$	α_3	α_1	$k + 4$
k	$\times$	α_1	α_1	ω
$k + 1$	I			ω

	Nombres			Instructions			
ω		$(a + rc)^2$	$k + 2$	$+$	α_1	α_2	α_1
			$k + 3$	$\leqslant$			$k - 1$
			$k + 4$	*ARR*			

Dans le dernier cas, le programme a une instruction supplémentaire.

Cependant, cette répartition des instructions peut être nécessaire. Ainsi, le test de l'exécution de la condition logique doit se faire au début du cycle, lorsque pour certaines valeurs des paramètres un tronçon cyclique de calculs peut être supprimé ; le nombre de cycles pour ces valeurs de paramètres est égal à zéro.

b) *La machine possède une opération de transfert de la commande suivant le résultat de l'inégalité* $\neq$.

Le processus cyclique doit se poursuivre jusqu'à ce qu'un nombre égal à $a + (N + 1)\,c$ apparaisse dans la mémoire α_1, nous placerons ce nombre dans la cellule α_3. Supposons

$$P = \begin{cases} 0 & \text{si} \quad a + rc = a + (N + 1)\,c\,, \\ 1 & \text{si} \quad a + rc \neq a + (N + 1)\,c\,. \end{cases}$$

Dans ce cas, on peut réaliser le test de l'exécution de la condition P et le transfert de la commande avec l'instruction

$k + 3$	$\neq$	α_1	α_3	k

c) *Cas de transfert de la commande suivant le résultat d'une égalité* $=$.

Nous utiliserons la cellule α_3 dans laquelle, comme dans le cas précédent, nous garderons le nombre $a + (N + 1)\,c$. Supposons

$$P = \begin{cases} 0 & \text{si} \quad a + rc \neq a + (N + 1)\,c \\ 1 & \text{si} \quad a + rc = a + (N + 1)\,c\,. \end{cases}$$

Maintenant, pour exécuter le processus cyclique, nous complétons le programme d'instructions k, $k + 1$, $k + 2$ avec les instructions :

$k + 3$	$=$	α_1	α_3	$k + 5$
$k + 4$	$=$			k
$k + 5$	*ARR*			

l'instruction $k + 4$ étant une opération de transfert inconditionnel de la commande.

d) *Cas de transfert de la commande suivant un numéro — TCN.*

Le processus cyclique doit se prolonger tant que

$$s = (a + rc) - (a + Nc) \leqslant -0 \quad (*) .$$

Supposons

$$P = \begin{cases} 0 & \text{si} \quad s > -0 \\ 1 & \text{si} \quad s \leqslant -0 . \end{cases}$$

Introduisons dans le programme des instructions k, $k + 1$, $k + 2$ une instruction pour calculer la grandeur s que nous garderons dans la cellule ω :

$k + 3$	$-$	α_1	α_3	ω

Pour le transfert de la commande au début du cycle, on peut alors utiliser l'instruction :

$k + 4$	*TCN*	ω	$k + 5$	k

e) *Cas de transfert de la commande suivant l'instruction TCS.*

Si, à la suite d'une soustraction, il se forme un symbole avec un signe négatif, l'exécution du programme cyclique avec l'instruction *TCS* ne pourra en rien être différenciée du cas précédent. Mais, ici, il importe que l'instruction, après l'exécution de laquelle apparaît nécessairement un symbole, précède immédiatement l'instruction de transfert de commande.

2. Processus de calcul itératif.

La résolution de l'équation

$$x = f(x) \tag{11}$$

(*) Nous supposerons que le résultat de la soustraction de codes semblables est le code « — 0 ».

par la méthode itérative est un exemple d'algorithme se divisant en cycles.

Partant de la grandeur initiale x_0, on calcule successivement

$$x_1 = f(x_0), \qquad x_2 = f(x_1), \ldots, x_n = f(x_{n-1}), \tag{12}$$

etc. Les calculs se prolongent tant que deux grandeurs voisines x_n et x_{n-1} ne coïncident pas à ε près, c'est-à-dire tant que $| x_n - x_{n-1} |$ ne devient pas plus petit que ε, ε étant donné d'avance. Dans le cas envisagé, le nombre global des cycles indispensables pour résoudre l'égalité (II) n'est pas connu à priori, il dépend de la grandeur ε et des conditions de convergence du processus. Prenons un exemple.

***Exemple* 2.** *Extraction d'une racine carrée.*

$$y = \sqrt{x}\,. \tag{13}$$

La recherche de $y = \sqrt{x}$ se réduit à la résolution de l'équation

$$f(y) = y^2 - x = 0\,. \tag{14}$$

Pour résoudre cette équation nous appliquerons la méthode des tangentes :

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)}\,;$$

nous obtiendrons la formule itérative connue :

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right). \tag{15}$$

Pour calculer les nombres y_0, x, $1/2$, ε d'après la formule (15), nous les placerons dans les cellules de l'organe mémoire α_1, α_2, α_3, α_4.

Le calcul d'une itération correspond à un cycle :

1) Calcul de y_{n+1} selon la formule (15), y_n étant donné.
2) Calcul de la différence $\delta_{n+1} = y_{n+1} - y_n$.
3) Préparation de l'organe mémoire pour le cycle suivant.

En conséquence, les valeurs de y_{n+1} et δ_{n+1} de l'itération suivante sont stockées dans les cellules de travail ω_1 et ω_2.

Le calcul selon la formule (15) est effectué par les instructions suivantes :

$k+1$	:	α_2	α_1	ω_1	$\frac{x}{y_n}$
$k+2$	+	α_1	ω_1	ω_1	$y_n + \frac{x}{y_n}$
$k+3$	×	ω_1	α_3	ω_1	$y_{n+1} = \frac{1}{2}\left(y_n + \frac{x}{y_n}\right)$

Le calcul de δ_{n+1} est effectué par l'instruction

$k+4$	$-$	ω_1	α_1	ω_2

La préparation de l'organe mémoire pour le cycle suivant consiste en l'envoi de la valeur de y_{n+1} dans la cellule α_1, ce qui est accompli par l'instruction

$k+5$	$+$	ω_1		α_1

Le passage d'un cycle à l'autre se produit tant que la différence $| \delta_{n+1} | \geqslant \varepsilon$, et les calculs s'arrêtent lorsque $| \delta_{n+1} | < \varepsilon$, ce qui est exécuté par l'instruction de transfert conditionnel

$k+6$	$\mid \leqslant \mid$	α_4	ω_2	$k+1$

que suit l'instruction :

$k+7$	*ARR*			

Programme 10.

Nombres

α_1	y_0	y_n
α_2	x	
α_3	$\frac{1}{2}$	
α_4	ε	
ω_1		y_n
ω_2		δ_n

Instructions

$k+1$	:	α_2	α_1	ω_1
$k+2$	$+$	α_1	ω_1	ω_1
$k+3$	$\times$	ω_1	α_3	ω_1
$k+4$	$-$	ω_1	α_1	ω_2
$k+5$	$+$	ω_1		α_1
$k+6$	$\mid \leqslant \mid$	α_4	ω_2	$k+1$
$k+7$	*ARR*			

Le programme peut être simplifié si on change de place l'ordre des calculs y_{n+1} et δ_{n+1}. A cet effet, nous emploierons les formules :

$$\left.\begin{aligned} \delta_{n+1} &= \frac{1}{2}\left(\frac{x}{y_n} - y_n\right) \\ y_{n+1} &= y_n + \delta_{n+1} \end{aligned}\right\} \tag{16}$$

Nous stockerons la grandeur δ_{n+1} dans la cellule de l'organe mémoire ω et y_{n+1} directement en α_1.

Programme 10_1.

Nombres

α_1	y_0	y_n
α_2	x	
α_3	$\frac{1}{2}$	
α_4	ε	
ω		δ_n

Instructions

$k+1$	:	α_2	α_1	ω
$k+2$	$-$	ω	α_1	ω
$k+3$	$\times$	ω	α_3	ω
$k+4$	$+$	α_1	ω	α_1
$k+5$	$\lvert \leqslant \rvert$	α_4	ω	$k+1$
$k+6$	*ARR*			

Ainsi, dans le cas donné, le programme a une instruction de moins et n'utilise qu'une cellule de travail.

En général, pour programmer des processus itératifs, il est plus rationnel de faire les calculs selon les formules

$$\left.\begin{aligned} \delta_{n+1} &= f(x_n) - x_n \\ x_{n+1} &= x_n + \delta_{n+1} \end{aligned}\right\} \tag{16'}$$

y_0 peut alors être choisi arbitrairement, mais il dépend en même temps du type de la machine. Par exemple, pour une machine à virgule fixe qui opère sur des nombres de module inférieur à 1, on peut poser $y_0 = 0{,}75$. En outre, on démontre facilement que toutes les itérations obtenues par les formules (16), pour tout $|x| < 1$, n'ont pas un module qui dépasse 1. Dans les machines à virgule flottante on peut, pour accélérer la convergence, choisir y_0 par rapport à la valeur de x.

3. Calcul des fonctions élémentaires.

***Exemple* 3.** *Calcul de la fonction exponentielle* e^x :

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots . \tag{17}$$

Pour que les calculs de la somme (17) aient la propriété d'être cycliques, nous utiliserons les relations de récurrence entre les termes u_n de rang n et les sommes partielles s_n :

$$\left.\begin{aligned} u_{n+1} &= u_n \frac{x}{n+1} \\ s_{n+1} &= s_n + u_{n+1} \end{aligned}\right\} . \tag{18}$$

Un cycle de calculs consiste à calculer d'après les données

$$n\,, \qquad u_n\,, \qquad s_n\,,$$

que nous placerons dans les cellules de l'organe mémoire

$$\alpha_1\,, \qquad \alpha_2\,, \qquad \alpha_3\,,$$

les grandeurs $n + 1$, u_{n+1}, s_{n+1}, qui seront rangées dans ces mêmes cellules α_1, α_2, α_3. Pour effectuer les calculs d'après les formules (18), nous placerons les nombres x et 1 dans les cellules de l'organe mémoire α_4 et α_5. Les calculs s'interrompent dès que le terme calculé u_{n+1} de rang $n + 1$ est inférieur au nombre donné ε, qui est placé dans la cellule α_6.

Pour que le programme soit établi, il reste encore à noter que les valeurs initiales des variables n, u_n, s_n, doivent être les suivantes :

$$n = 0\,, \qquad u_0 = 1\,, \qquad s_0 = 1\,.$$

Programme 11.

Nombres			Instructions				
α_1	0	n	$k+1$	+	α_1	α_5	α_1	$n+1$
α_2	1	u_n	$k+2$	×	α_2	α_4	α_2	$u_n\,x$
α_3	1	s_n	$k+3$	:	α_2	α_1	α_2	u_{n+1}
α_4	x		$k+4$	+	α_3	α_2	α_3	s_{n+1}
α_5	1		$k+5$	$\lvert \leqslant \rvert$	α_6	α_2	$k+1$	
α_6	ε		$k+6$	*ARR*				

***Exemple* 4.** *Calcul des fonctions trigonométriques.*

Prenons, comme exemple, le calcul de la fonction sin x à l'aide d'un développement en série

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots .$$

Comme dans le cas précédent, nous utiliserons les relations entre les nombres de la série et les sommes partielles :

$$\left.\begin{aligned} u_{n+1} &= -x^2 \frac{u_n}{a_{n+1}}, \qquad \text{où} \qquad a_n = 2n(2n+1) \\ s_{n+1} &= s_n + u_{n+1} \qquad (n = 1, 2, \ldots) \end{aligned}\right\}. \tag{19}$$

Pour rendre cycliques les opérations selon les formules (19), il reste à établir les relations de récurrence pour a_n. Fixons $b_n = 2n$, $c_n = 2n + 1$, alors

$$b_{n+1} = c_n + 1 ; \qquad c_{n+1} = b_{n+1} + 1 ; \qquad a_{n+1} = b_{n+1} \cdot c_{n+1} . \tag{20}$$

Par conséquent, chaque cycle de calculs selon les formules (19) et (20) consiste à calculer, à partir de u_n, s_n, c_n donnés que nous placerons dans les cellules de l'organe mémoire α_1, α_2, α_3, les valeurs de u_{n+1}, s_{n+1}, c_{n+1} qui seront stockées dans les mêmes cellules α_1, α_2, α_3.

Nous mettrons les nombres $-x^2$, 1, b_n dans les cellules α_4, α_5, α_7.

Pour le nombre a_n, on peut utiliser la même cellule α_7, puisque la valeur de b_n n'est pas nécessaire dans le calcul du cycle suivant.

La valeur initiale des variables est la suivante :

$$u_0 = x ; \qquad s_0 = x ; \qquad c_0 = 1 .$$

Comme dans le cas précédent, les calculs s'arrêtent lorsque le terme u_{n+1} est inférieur au nombre donné ε, que nous mettrons dans la cellule α_6.

Programme 12.

Nombres			Instructions				
α_1	x	u_n	$k+1$	$+$	α_3	α_5	α_7	b_1
α_2	x	s_n	$k+2$	$+$	α_7	α_5	α_3	c_1
α_3	1	c_n	$k+3$	$\times$	α_7	α_3	α_7	a_1
α_4	$-x^2$		$k+4$	$\times$	α_1	α_4	α_1	$-x^2 u_0$

Nombres

α_5	1	
α_6	ε	
α_7		b_n

Instructions

$k+5$	:	α_1	α_7	α_1	u_1
$k+6$	+	α_2	α_1	α_2	s_1
$k+7$	$\mid \leqslant \mid$	α_6	α_1	$k+1$	
$k+8$	*ARR*				

Pour calculer tg x, nous utiliserons son développement en une fraction continue. A condition que $|x| < \pi/4$, on peut calculer tg x à 10^{-10} près au moyen de

$$\operatorname{tg} x = \frac{x}{y}, \tag{21}$$

où

$$y = 1 - \cfrac{x^2}{3 - \cfrac{x^2}{5 - \cfrac{x^2}{7 - \cfrac{x^2}{9 - \cfrac{x^2}{11 - \cfrac{x^2}{13 - \cfrac{x^2}{15}}}}}}} \tag{22}$$

D'après un algorithme, semblable à celui de Hörner, on écrit la formule (22) sous la forme :

$$y \simeq (1 - x^2 : \{ 3 - x^2 : [\ldots - x^2 : \{ 11 - x^2 : \\ : (13 - x^2 : [15])\} \ldots] \ldots\}). \tag{23}$$

Les calculs de y d'après la formule (23) consistent à calculer successivement les parenthèses u_{2k-1} d'après la formule :

$$u_{2k-1} = (2k - 1) - \frac{x^2}{u_{2k+1}} \quad (k = 7, 6, \ldots, 1) \tag{24}$$

qui représente elle-même le processus cyclique ; en outre $y \simeq u_1$. Un cycle consiste à calculer d'après les données

$$(2k+1), \qquad u_{2k+1},$$

qui sont placées dans les cellules de l'organe mémoire

$$\alpha_1, \qquad \alpha_2,$$

les grandeurs $(2k-1)$, u_{2k-1} qui seront placées dans les mêmes cellules α_1, α_2.

Pour calculer y, on utilise les nombres x^2 et 2 que l'on stockera dans les mémoires α_3 et α_4.

La valeur initiale des variables est la suivante :

$$2k+1 = 2.7+1 = 15;$$

$$u_{2k+1} = u_{15} = 15.$$

Le processus de calcul de y d'après la formule (24) s'achève après le calcul de u_1 ; de plus, dans la cellule α_1, il y aura le nombre 1.

En conséquence, on peut effectuer le passage d'un cycle à l'autre jusqu'à obtention de la valeur de y, à l'aide de l'instruction de transfert conditionnel suivante :

$\leqslant$	α_4	α_1	$k+1$

après quoi, on a le calcul de tg x d'après la formule (21) et l'arrêt des calculs, c'est-à-dire les instructions :

:	α_5	α_2	α_1
ARR			

la cellule de l'organe mémoire α_5 contient x.

Programme 13.

Nombres

α_1	15	$2k+1$
α_2	15	u_{2k+1}
α_3	x^2	

Instructions

$k+1$	—	α_1	α_4	α_1	$2k-1$
$k+2$	:	α_3	α_2	α_2	$\frac{x^2}{u_{2k+1}}$
$k+3$	—	α_1	α_2	α_2	u_{2k-1}

Nombres

α_4	2
α_5	x

Instructions

$k+4$	$\leqslant$	α_4	α_1	$k+1$	
$k+5$	:	α_5	α_2	α_1	tg x
$k+6$	*ARR*				

Le programme 13 est l'exemple le plus simple de la synthèse d'un programme de processus cyclique (4 instructions) et d'un programme de calcul d'après une formule (1 instruction).

***Exemple* 5.** *Réduction d'un argument.*

Dans de nombreux cas, pour calculer les différentes valeurs d'une fonction, il est commode d'utiliser les formules de réduction d'un argument. Il est nécessaire de le faire, par exemple, lorsque dans les machines à virgule fixe on calcule les valeurs élevées de l'argument. La réduction de l'argument, lorsque les fonctions ont été calculées à l'aide de leur développement en série, diminue de toute évidence le nombre des cycles.

Examinons la réduction de l'argument de sin x dans l'intervalle $|x| \leqslant \pi/2$. Soit :

$$\psi(x) = \left| \left\{ \frac{x}{2\pi} - \frac{1}{4} \right\} 2\pi - \pi \right| - \frac{\pi}{2}. \tag{25}$$

Il est évident que si x est quelconque :

$$\sin x = \sin \psi(x) \quad \text{et} \quad |\psi(x)| \leqslant \frac{\pi}{2}.$$

Ainsi le calcul de sin x se divise en deux étapes :

1) Calcul de $\psi(x)$ par la formule (25).
2) Calcul de sin $\psi(x)$.

On peut effectuer la seconde de ces étapes — le calcul de sin $\psi(x)$ — selon le programme 12 du calcul d'un sinus, si le contenu de l'organe mémoire y est prévu d'avance.

En vue de conserver la forme des instructions qui effectuent le calcul du sinus d'après le programme 12, mettons les grandeurs 1, 1, ε dans les cellules de l'organe mémoire α_3, α_5, α_6.

Pour préparer le contenu des cellules α_1, α_2, α_4, on emploie le processus de calcul de $\psi(x)$.

Pour calculer $\psi(x)$ d'après la formule (25), rangeons les grandeurs x, 2π, $1/4$, π, $\pi/2$ respectivement dans les cellules de l'organe mémoire α_1, α_2, α_4, α_7, α_8. Supposons que, parmi les opérations élémentaires effectuées par la machine, il y ait une opération qui serve à prendre la partie décimale du

nombre { } (cf. programme 5) et une autre à prendre le module (cf. programme 6).

Le calcul d'après la formule (25) est alors effectué par les instructions suivantes :

$k+1$	:	α_1	α_2	α_1	$\frac{x}{2\pi}$
$k+2$	—	α_1	α_4	α_1	$\frac{x}{2\pi} - \frac{1}{4}$
$k+3$	{ }	α_1		α_1	$\left\{\frac{x}{2\pi} - \frac{1}{4}\right\}$
$k+4$	×	α_1	α_2	α_1	$\{\}\,2\pi$
$k+5$	—	α_1	α_7	α_1	$\{\}\,2\pi - \pi$
$k+6$	| |	α_1		α_1	$\lvert\{\}\,2\pi - \pi\rvert$
$k+7$	—	α_1	α_8	α_1	$\lvert\ \rvert - \frac{\pi}{2} = \psi(x)$

Pour calculer $\sin \psi(x)$ il reste à préparer le contenu des cellules α_2 et α_4, ce que l'on peut effectuer avec les instructions

$k+8$	+	α_1		α_2
$k+9$	×	α_1	α_1	α_4
$k+10$	—		α_4	α_4

Programme 14.

$$\sin x = \sin \psi(x)\ ; \qquad \psi(x) = \left| \left\{ \frac{x}{2\pi} - \frac{1}{4} \right\} 2\pi - \pi \right| - \frac{\pi}{2}$$

Nombres

α_1	x	U_n
α_2	2π	S_n
α_3	1	c_n
α_4	1/4	$-\psi^2(x)$
α_5	1	

Instructions

$k+1$	:	α_1	α_2	α_1
$k+2$	—	α_1	α_4	α_1
$k+3$	{ }	α_1		α_1
$k+4$	×	α_1	α_2	α_1
$k+5$	—	α_1	α_7	α_1

Nombres

α_6	ε	
α_7	π	b_n
α_8	$\frac{\pi}{2}$	

Instructions

$k+6$	$\|\ \|$	α_1		α_1
$k+7$	—	α_1	α_8	α_1
$k+8$	+	α_1		α_2
$k+9$	×	α_1	α_1	α_4
$k+10$	—		α_4	α_4
$k+11$	+	α_3	α_5	α_7
$k+12$	+	α_7	α_5	α_3
$k+13$	×	α_7	α_3	α_7
$k+14$	×	α_1	α_4	α_1
$k+15$	:	α_1	α_7	α_1
$k+16$	+	α_2	α_1	α_2
$k+17$	$\|\leqslant\|$	α_6	α_1	$k+11$
$k+18$	*ARR*			

Le programme 14 donne un exemple de la réunion de deux programmes : le programme de calcul selon la formule (25) (10 instructions), et le programme cyclique du calcul d'un sinus (8 instructions).

Dans certains cas, le calcul des valeurs de la fonction peut être abrégé, si l'on emploie les relations de récurrence correspondantes. Prenons un exemple :

***Exemple* 6.** *Calcul et impression de la table des valeurs de la fonction* e^{kh}, $k = 1, 2, \ldots, N$.

On peut l'effectuer d'après la formule :

$$e^{kh} = e^{(k-1)h} \cdot e^{h} . \tag{26}$$

Supposons la grandeur e^h connue et placée dans la cellule de l'organe mémoire β_1 ; la valeur e^{kh}, que l'on calcule, sera alors placée dans la cellule β_2.

Un cycle de calculs consiste à faire la multiplication de la formule (26) et à imprimer le nombre qui se trouve dans la cellule β_2 où il y avait initiale-

ment $e^0 = 1$; il est exécuté par les instructions :

k	$\times$	β_1	β_2	β_2
$k+1$	I			β_2

Le nombre de cycles pour un cas donné est égal à N.

Cependant, la N-ième valeur — valeur terminale des quantités rencontrées — n'est pas connue à l'avance. C'est pourquoi, pour passer d'un cycle à l'autre et arrêter les calculs, on a besoin d'utiliser le nombre N, qui indique le nombre de répétitions du même cycle. Ce nombre sera placé en β_3.

Soumettons à l'examen la variable auxiliaire à valeurs entières i, changeant, au cours du passage d'un cycle à l'autre, sa valeur d'une unité et ayant pour valeur finale $i = 0$ (*). Gardons cette variable dans la cellule β_4. Complétons le cycle des calculs par l'instruction suivante que nous placerons au début du cycle :

$k-1$	$+$	β_4	β_5	β_4

β_5 contenant « l'unité ». Alors, à n'importe quel moment, la cellule β_4 contient un nombre d'unités égal au nombre de répétitions du cycle. Il est d'usage d'appeler ces cellules : *compteurs des répétitions d'un cycle*.

Une condition logique, par exemple,

$$P\{ i \neq N \}$$

peut être utilisée pour déterminer la fin du cycle ; l'instruction

$k+2$	$\neq$	β_3	β_4	$k-1$

qui se trouve à la fin du cycle, effectuera le test de cette condition et le transfert de commande correspondant.

Programme 15.

Nombres

β_1	e^h	
β_2	1	e^{kh}

Instructions

$k-1$	$+$	β_4	β_5	β_4
k	$\times$	β_1	β_2	β_2

(*) En outre, on peut, par exemple, prendre comme « unité », l'unité de la position la plus à droite.

Nombres

β_3	N	
β_4	0	i
β_5	1	

Instructions

$k+1$	I			β_2
$k+2$	$\neq$	β_3	β_4	$k-1$
$k=3$	ARR			

A la place de la condition logique $P\{ i \neq N \}$, on peut utiliser la suivante :

$$P\{ i \geqslant N \}.$$

L'instruction

$<$	β_3	β_4	$k+4$

qui se trouve après la $(k-1)$-ième instruction réalise le passage d'un cycle à l'autre et le transfert de la commande à arrêt à la fin des calculs. De plus, le cycle se termine par l'opération de transfert inconditionnel au début du nouveau cycle, c'est-à-dire par l'instruction :

$\leqslant$			$k-1$

Programme 15_1.

Nombres

β_1	e^h	
β_2	1	e^{kh}
β_3	N	
β_4	0	k
β_5	1	

Instructions

$k-1$	$+$	β_4	β_5	β_4
k	$<$	β_3	β_4	$k+4$
$k+1$	$\times$	β_1	β_2	β_2
$k+2$	I			β_2
$k+3$	$\leqslant$			$k-1$
$k+4$	ARR			

L'instruction $k-1$ compte le nombre de cycles, c'est-à-dire accomplit le travail d'un compteur (cellule de l'organe mémoire β_4).

On peut aussi placer l'instruction $k - 1$ après l'instruction k ; mais, dans ce dernier cas, pour accomplir N cycles il convient de placer le nombre $N - 1$ dans la cellule β_3 ou, tout en conservant le contenu antérieur de celle-ci, mettre 1 au début du premier cycle dans le compteur β_4.

L'instruction de transfert inconditionnel $k + 3$ peut être supprimée si on place en fin de cycle l'instruction k, après avoir interverti ses première et deuxième adresses. Puis, l'instruction « < » transmettra la commande à l'instruction $k - 1$ jusqu'à ce que dans la cellule β_4 apparaisse le nombre N, après quoi on s'arrêtera. Au total on aura accompli N cycles.

Programme 15_2.

	Nombres			Instructions			
β_1	e^h		$k+1$	+	β_4	β_5	β_4
β_2	1	e^{kh}	$k+2$	×	β_1	β_2	β_2
β_3	N		$k+3$	I			β_2
β_4	0	k	$k+4$	<	β_4	β_3	$k+1$
β_5	1		$k+5$	ARR			

Supposons maintenant que la quantité e^h ne soit pas donnée, mais doive être calculée d'après h qui, lui, est donné et que l'on placera dans la cellule α_4, en vue du calcul de e^h suivant le programme 11.

Le calcul et l'impression du tableau des valeurs de la fonction e^{kh}, $k = 1, 2, \ldots, N$, dans ce cas, se divisent en deux étapes.

1) Calcul de e^h d'après le programme 11 ; la valeur de e^h conformément à ce programme sera obtenue dans la cellule α_3.

2) Calcul de e^{kh} d'après le programme 15, dans lequel la cellule β_1 doit être remplacée par la cellule α_3, et la troisième adresse de la $(k + 4)$-ième instruction doit changer conformément au nouveau numéro de l'instruction initiale. En outre, on peut utiliser, au lieu de la cellule β_5, la cellule α_5 comme compteur. Le programme aura la forme suivante :

Programme 16.

	Nombres		Instructions			
α_1	0	$k+1$	+	α_1	α_5	α_1
α_2	1	$k+2$	×	α_2	α_4	α_2

Nombres

α_3	1	e^h
α_4	h	
α_5	1	
α_6	ε	
β_2	1	e^{kh}
β_3	N	
β_4	0	

Instructions

$k+3$	:	α_2	α_1	α_2
$k+4$	+	α_3	α_2	α_3
$k+5$	$\lvert \leqslant \rvert$	α_6	α_2	$k+1$
$k+6$	×	α_3	β_2	β_2
$k+7$	I			β_2
$k+8$	+	β_4	α_5	β_4
$k+9$	<	β_4	β_3	$k+6$
$k+10$	ARR			

Le programme 16 représente la réunion de deux programmes cycliques : le programme 11 (5 instructions) et le programme 15 (5 instructions). Comme nous le voyons d'après l'exemple donné, lorsqu'on réunit des programmes contenant des instructions de transfert conditionnel et inconditionnel, leur troisième adresse peut varier.

Le calcul et l'impression du tableau des valeurs des fonctions $\cos(a + kh)$, $\sin(a + kh)$ $(k = 1, 2, \ldots, N)$ peuvent être obtenus en utilisant les formules :

$$\left.\begin{aligned} \cos(a + kh) &= \cos h \cos[a + (k-1)h] - \sin h \sin[a + (k-1)h] \\ \sin(a + kh) &= \sin h \cos[a + (k-1)h] + \cos k \sin[a + (k-1)h] \end{aligned}\right\}. \quad (27)$$

Supposons les grandeurs $\cos h$ et $\sin h$ connues et placées dans les cellules β_1 et β_2. Les valeurs à calculer $\cos(a + kh)$ et $\sin(a + kh)$ seront placées dans les cellules β_3 et β_4.

Un cycle consiste à calculer d'après les formules (27) et à imprimer les nombres de β_3 et β_4 ; il sera réalisé par les instructions :

$k+1$	×	β_2	β_3	ω_1	$\sin h \cos[a + (k-1)h]$
$k+2$	×	β_1	β_3	β_3	$\cos h \cos[a + (k-1)h]$
$k+3$	×	β_1	β_4	ω_2	$\cos h \sin[a + (k-1)h]$
$k+4$	×	β_2	β_4	β_4	$\sin h \sin[a + (k-1)h]$

$k + 5$	$-$	β_3	β_4	β_3	$\cos(a + kh)$
$k + 6$	I			β_3	
$k + 7$	$+$	ω_1	ω_2	β_4	$\sin(a + kh)$
$k + 8$	I			β_4	

Comme dans l'exemple précédent, le nombre de cycles est connu à l'avance et égal à N, mais la valeur finale des variables, qui se rencontrent au cours du calcul, n'est pas connue d'avance. Nous garderons dans la cellule β_5 le nombre $N - 1$ et nous l'utiliserons pour passer d'un cycle à l'autre et pour déterminer la fin des calculs. Dans ce but, nous introduirons un compteur du nombre de cycles dans la cellule β_6, dont le contenu initial sera « 0 », et dans la cellule β_7 nous mettrons « 1 » pour le compteur. De cette façon, un cycle de calcul est complété par les instructions :

$+$	β_6	β_7	β_6
$\leqslant$	β_6	β_5	$k + 1$

que nous placerons à la fin du cycle. Le contenu initial des cellules β_3 et β_4 doit être cos a et sin a. Supposons aussi ces grandeurs connues.

Programme 17.

	Nombres			Instructions			
β_1	$\cos h$		$k + 1$	$\times$	β_2	β_3	ω_1
β_2	$\sin h$		$k + 2$	$\times$	β_1	β_3	β_3
β_3	$\cos a$	$\cos(a + kh)$	$k + 3$	$\times$	β_1	β_4	ω_2
β_4	$\sin a$	$\sin(a + kh)$	$k + 4$	$\times$	β_2	β_4	β_4
β_5	$N - 1$		$k + 5$	$-$	β_3	β_4	β_3
β_6	0	k	$k + 6$	I			β_3
β_7	1		$k + 7$	$+$	ω_1	ω_2	β_4

Nombres

ω_1	
ω_2	

Instructions

$k + 8$	I			β_4
$k + 9$	$+$	β_6	β_7	β_6
$k + 10$	$\leqslant$	β_6	β_5	$k + 1$
$k + 11$	ARR			

CONCLUSIONS

1) Pour les processus de calcul cycliques, il faut établir des programmes composés d'instructions indispensables à l'accomplissement d'un cycle de calcul (le premier) et de quelques instructions secondaires.

2) Les instructions qui préparent le chargement convenable de l'organe mémoire pour permettre le passage d'un cycle à l'autre et les instructions qui fournissent le nombre indispensable de répétitions de cycles sont des instructions secondaires.

3) Quand on établit des programmes cycliques, les valeurs des variables prévues pour accomplir le $(k - 1)$-ième cycle doivent être mises dans les cellules où se trouvaient leur k-ième valeur.

4) Pour former la condition logique qui détermine la fin du processus cyclique, on peut utiliser soit une variable spécialement introduite, soit une variable rencontrée au cours des calculs.

Dans le premier cas, le plus commode est d'introduire la variable à valeurs discrètes k (k est le numéro du cycle accompli), appelée compteur.

De plus, il est indispensable d'avoir :

a) un compteur du nombre de cycles : une cellule qui change de contenu d'un cycle à l'autre selon une loi déterminée ;

b) une unité conventionnelle pour le compteur : une cellule à l'aide de laquelle se produit la modification du compteur ;

c) une cellule pour accomplir le chargement maximal du compteur ;

d) une instruction pour accomplir le travail du compteur (elle entre dans le cycle) ;

e) une instruction pour contrôler le chargement du compteur, afin de pouvoir connaître la fin du processus et exécuter le transfert de la commande à l'instruction initiale ou à celle qui suit le cycle (elle entre aussi dans le cycle) ; et enfin, lorsque le programme cyclique tourne, il faut :

f) assurer le chargement initial du compteur et de toutes les grandeurs changeant d'un cycle à l'autre, conformément à la valeur qu'elles ont avant le passage du premier cycle.

Pour déterminer la fin d'un processus cyclique, il faut utiliser la variable qui rentre dans les calculs si sa valeur finale (ou la condition que cette dernière doit satisfaire) est connue.

5) Si nous admettons une suite de valeurs intervenant dans le calcul des paramètres, suite qui supprime un tronçon cyclique, les instructions contrôlant la condition logique, qui détermine la fin du processus cyclique, doivent être placées au début du cycle. Dans ce cas, le cycle est complété à la fin par une opération de transfert inconditionnel de commande à la première instruction.

EXERCICES

1. Etablir un programme afin de calculer et d'imprimer la table des carrés des nombres impairs de la suite naturelle des nombres.

2. Etablir le programme d'extraction de la racine cubique $y = \sqrt[3]{x}$, en utilisant le processus itératif

$$y_{n+1} = y_n\left(\frac{3}{2} - \frac{y_n^2}{2x}\right).$$

3. Etablir le programme du calcul d'une quantité inverse pour une machine qui ne possède pas la division.

4. Etablir le programme du calcul de ln x à l'aide d'une série.

5. Etablir le programme d'extraction de la racine cubique $y = \sqrt[3]{x}$, en utilisant le processus itératif indiqué dans l'exercice 2, pour une machine dans laquelle il n'y a pas de division.

6. Etablir le programme du calcul de ln $(1 + x)$, $|x| < 1$ à l'aide d'une série.

7. Etablir un programme pour calculer e^x, en utilisant la représentation :

$$e^x = \frac{y + x}{y - x}, \qquad \text{où} \qquad y = 2 + \cfrac{x^2}{6 + \cfrac{x^2}{10 + \cfrac{x^2}{14 + \cfrac{x^2}{18 + \cfrac{x^2}{22}}}}}$$

pour $0 < |x| \leqslant 1$.

8. Etablir un programme pour calculer e^x, $x > 0$, en utilisant la relation

$$e^x = e^{E(x)}\, e^{\{x\}}$$

et en ayant dans l'organe mémoire le nombre e.

9. Etablir un programme pour calculer et imprimer $\sin(a + kh)$ et $\cos(a + kh)$, $k = 0, 1, \ldots, N$, dans le cas où $\cos h$ et $\sin h$ ne sont pas donnés d'avance. Utiliser le programme du calcul de $\sin h$, la formule

$$\cos h = \sqrt{1 - \sin^2 h}$$

et le programme 10. Compter que dans le nombre des opérations élémentaires effectuées par la machine il y a celle de l'extraction d'une racine carrée.

10. Etablir un programme pour calculer et imprimer $\sin(a + kh)$ et $\cos(a + kh)$, $k = 0, \ldots, N$, pour le cas où $\cos h$ et $\sin h$ ne sont pas donnés à priori. Utiliser le programme 16 et le programme de calcul d'un sinus pour calculer $\sin x$ et $\cos x = \sin\left(\frac{2}{\pi} - x\right)$.

11. Etablir un programme pour calculer $\cos x$ à l'aide d'une série :

$$\cos x = 1 - \frac{x^2}{2\,!} + \frac{x^4}{4\,!} - \cdots.$$

12. Etablir un programme pour résoudre l'équation :

$$\tfrac{1}{4}x^3 + x - 1{,}250\,2 = 0$$

par la méthode itérative.

13. Etablir un programme pour calculer le produit scalaire de deux vecteurs.

14. Etablir la commande des processus cycliques dans le programme 15 à l'aide des opérations de transfert de commande « = », « *TCN* » et « *TCS* ».

4. PROCESSUS CYCLIQUES DÉPENDANT DE PARAMÈTRES

Dans le paragraphe précédent nous avons examiné des processus cycliques dans lesquels, lorsqu'on passe d'un cycle à un autre, change seul le contenu des cellules de calcul de l'organe mémoire. Ici, nous allons examiner des processus cycliques dans lesquels, lors du passage d'un cycle à l'autre, changent aussi les adresses des nombres qui participent aux calculs.

***Exemple* 1.** *Calcul d'une table des carrés des termes d'une progression arithmétique avec mise en mémoire externe.*

Supposons le calcul des carrés des termes d'une progression arithmétique de zéro au N-ième inclus et leur transfert dans la mémoire externe. Choisissons les cellules opération

$$\omega, \omega + 1, ..., \omega + N, \tag{28}$$

dans lesquelles nous placerons successivement les carrés calculés. Une fois les calculs des carrés terminés, nous enverrons les codes contenus dans les cellules (28) de l'organe mémoire interne aux cellules qui ont les numéros r, $r + 1$, $r + 2$, ..., $r + N$, de l'organe mémoire externe au p-ième secteur.

Naturellement, les calculs se divisent en $N + 1$ cycles : au n-ième cycle, le carré du n-ième terme de la progression $a + (n - 1)\,c$ se calcule et se conserve.

Nous utiliserons le programme 9 auquel nous ferons subir quelques changements ; nous supprimerons la seconde instruction : imprimer les carrés numériques des nombres et nous donnerons à la première instruction une forme qui assure la mémorisation de $[a + (n - 1)\,c]^2$ dans la cellule $\omega + N$, c'est-à-dire :

$k + 1$	×	α_1	α_1	$\omega + N$

Afin que la quantité calculée $[a + (n - 1)\,c]^2$ ne soit pas supplantée au n-ième cycle par le résultat obtenu au $(n + 1)$-ième cycle, nous compléterons le programme par l'instruction :

$k - N - 1$	+	$\omega + 1$		ω
$k - N$	+	$\omega + 2$		$\omega + 1$
⋮				
k	+	$\omega + N$		$\omega + N - 1$

Pour le groupe d'instructions $k - N - 1, ..., k$, nous introduirons l'instruction symbolique de décalage de groupe « DG » :

k	DG	$\omega + 1$	N	ω

qui effectue le décalage du contenu des cellules $\omega + 1, ..., \omega + N$ d'une cellule vers la gauche. Le chargement initial des cellules $\omega, \omega + 1, ..., \omega + N$

est indifférent. A la suite de N décalages, les quantités se trouvent dans les cellules (28).

Pour envoyer les résultats obtenus dans l'organe mémoire externe, nous introduirons dans le programme les instructions

$k + 4$	Ea	p	$\omega + 1$	$r + 1$
$k + 5$	Eb	M	$\omega + N$	

Nous obtiendrons le programme 18.

Programme 18.

Nombres

α_1	a	
α_2	c	
α_3	$a + Nc$	
ω		a^2
$\omega + 1$		$(a + c)^2$
$\vdots$		
$\omega + N$		$(a + Nc)^2$

Instructions

k	DG	$\omega + 1$	N	ω
$k + 1$	$\times$	α_1	α_1	$\omega + N$
$k + 2$	$+$	α_1	α_2	α_1
$k + 3$	$\leqslant$	α_1	α_3	k
$k + 4$	Ea	p	$\omega + 1$	$r + 1$
$k + 5$	Eb	M	$\omega + N$	
$k + 6$	ARR			

Les instructions de notre programme remplissent les fonctions suivantes : l'instruction $k + 2$ prépare le terme suivant de la progression arithmétique ; l'instruction $k + 1$ de chaque cycle calcule la valeur inconnue du carré dans la même cellule « standard » $\omega + N$; k assure la mémorisation du résultat obtenu — son « évacuation » de la cellule standard. Ces instructions peuvent être changées de place, lorsqu'elles varient de façon adéquate et lorsqu'il y a sélection du chargement initial correspondant de la cellule α_1. Cette même structure de programme est possible pour calculer la table de n'importe quelles autres fonctions pour les valeurs des arguments successivement calculées.

***Exemple* 2.** *Calcul des valeurs d'un polynôme.*

Soit :

$$f(x) = \{ \ldots [(a_0 x + a_1) x + a_2] x + \cdots + a_{n-1} \} x + a_n . \qquad (29)$$

Examinons le programme du calcul d'un polynôme d'après l'algorithme de Hörner noté au § 1. Toutes les instructions de ce programme (programme 3) de numéro impair, à l'exclusion du premier, sont identiques et ont la forme :

$N + 2k + 1$	x	ω	β	ω	$k = 0, 1, \ldots, n.$

Le calcul, d'après la formule (29), peut être divisé en une série de cycles ; dans chaque cycle on retrouve la multiplication de x par la parenthèse correspondante et l'addition du coefficient a_i correspondant. La parenthèse nulle est égale à a_0, par conséquent nous placerons, pour assurer le caractère cyclique, la valeur de la parenthèse à calculer dans la cellule contenant a_0. Les calculs au $(k + 1)$-ième cycle peuvent être alors exécutés par les mêmes instructions qu'au k-ième, si le coefficient a_{k+1} est placé dans la cellule dans laquelle se trouvait le coefficient a_k.

Nous répartirons ensuite les données de la façon suivante : nous placerons les coefficients du polynôme $a_0, a_1, \ldots, a_n$ dans les cellules de travail $\alpha, \alpha + 1, \ldots, \alpha + n$, et la grandeur x dans la cellule β. Au premier cycle, le programme des calculs aura la forme :

$N + 1$	$\times$	α	β	α
$N + 2$	$+$	α	$\alpha + 1$	α

Pour assurer le caractère cyclique nous introduirons les instructions :

$+$	$\alpha + 2$		$\alpha + 1$
$\vdots$	$\vdots$		$\vdots$
$+$	$a + n$		$\alpha + n - 1$

ou l'instruction :

DG	$\alpha + 2$	$n - 1$	$\alpha + 1$

opérant le décalage des coefficients d'une cellule à gauche : ce qui prépare le contenu de l'organe mémoire à réaliser le cycle suivant.

Connaissant le nombre de cycles, qui est égal au nombre n (nous le placerons dans la cellule β_1 de l'organe mémoire), nous pouvons déterminer l'arrêt des calculs. Pour compter le nombre des cycles accomplis, nous introduirons un compteur : la cellule β_2 dans laquelle nous placerons au début « 0 » alors que dans la cellule β_3 nous mettrons « 1 » pour le compteur. Le programme du cycle sera accompli par les instructions :

N	+	β_2	β_3	β_2

$N+4$	$\leqslant$	β_2	β_1	N

dont nous plaçons la première au début du cycle, la seconde à la fin.

Nous obtiendrons le programme :

Programme 19.

Nombres

α	a_0
$\alpha+1$	a_1
⋮	⋮
$\alpha+n$	a_n
β	x
β_1	n
β_2	0
β_3	1

Instructions

N	+	β_2	β_3	β_2
$N+1$	×	α	β	α
$N+2$	+	α	$\alpha+1$	α
$N+3$	*DG*	$\alpha+2$	$n-1$	$\alpha+1$
$N+4$	$\leqslant$	β_2	β_1	N
$N+5$	*ARR*			

A la différence du programme où l'instruction *DG* (ou le groupe d'instructions de transfert correspondant) servait de mémorisation de la suite des résultats obtenus dans une cellule standard, ici cette instruction (ou le groupe d'instructions correspondant) produit l'envoi des données (des coefficients) de la suite à la cellule standard $\alpha + 1$.

Quand on modifie l'instruction *DG* de façon convenable, on peut la déplacer avec les instructions N à $N + 2$.

Les programmes des exemples donnés ont un défaut : on y utilise le transfert improductif de la succession des grandeurs, ce qui fait consommer du temps machine et des cellules de l'organe mémoire. Lorsque les problèmes deviennent de plus en plus complexes, la permutation effective des grandeurs dans les cellules de l'organe mémoire ne peut plus se faire que très difficilement.

Lorsqu'il y a une instruction de substitution d'adresse parmi les opérations élémentaires, la permutation des grandeurs, servant à établir les programmes cycliques, n'est pas obligatoire.

A la place de la permutation des grandeurs, disposées en une suite déterminée, il suffit, au début du nouveau cycle, de changer l'adresse des instructions selon lesquelles se fait l'extraction des grandeurs de la mémoire, ou l'envoi des résultats dans une suite de cellules.

Ainsi, après avoir mis dans la machine la forme initiale du programme cyclique, on le complète au moyen d'instructions préparant les instructions précédentes pour l'exécution répétée du cycle : la machine non seulement exécute les instructions mais les prépare aussi à l'exécution suivante.

De plus, la question de distribution de la mémoire est d'une grande importance.

Il faut distribuer les grandeurs qui participent aux calculs de façon que le calcul par étapes, en cycles indépendants, puisse être accompli selon des programmes qui ne se distinguent que par le changement d'une ou de plusieurs adresses de chacun d'eux, en fonction du numéro du cycle, selon la loi

$$N = \alpha + ip\,,$$

i étant le numéro du cycle, α la valeur initiale de l'adresse (elle peut être différente pour différentes adresses), p étant une constante, identique pour toutes les adresses variables. Il est admis d'appeler ces processus : *processus cycliques dépendant du paramètre i.*

Puis, un cycle de calcul doit être complété par des instructions de substitution d'adresse, changeant de façon appropriée les adresses variables du programme. De plus, si le nombre de cycles est connu à l'avance, pour savoir quand les calculs doivent s'arrêter, on peut utiliser une instruction variable du cycle.

Dans le cas où, le processus cyclique étant achevé, le programme donné rend possibles des calculs ultérieurs, il est indispensable de prévoir le rétablissement de l'état initial des instructions, c'est-à-dire de remettre des instructions variables dans la forme assurant l'exécution du premier cycle.

***Exemple* 3.** *Calcul de la table des carrés des entiers naturels avec mise en mémoire externe.*

Revenons à la première partie de ce paragraphe. Pour calculer le carré du nombre n, nous emploierons le programme 18 que nous changerons un peu : supprimons l'instruction k (ou ce groupe d'instructions d'envoi), et donnons

à la $(k + 1)$-ième instruction la forme qui permette la mémorisation du carré calculé n^2 dans les cellules correspondantes $\omega + n$ c'est-à-dire la forme :

$k + 1$	$\times$	α_1	α_1	$\omega + n$

La troisième adresse de l'instruction $k + 1$ est variable ; elle dépend du numéro du cycle accompli, dans notre cas de celui qui correspond au nombre n. En rapport avec ces modifications, nous compléterons le cycle des calculs avec l'instruction qui prépare la $(k + 1)$-ième instruction pour effectuer le cycle suivant :

$\oplus$	$k + 1$	β	$k + 1$

que nous placerons après l'instruction $k + 1$ devant l'instruction $\leqslant$ (indifféremment avant ou après l'instruction « + » préparant le nombre suivant). Ici, dans la cellule β de l'organe mémoire se place le code correspondant à « 1 » de l'adresse A 3, c'est-à-dire le nombre

β				1

Pour envoyer les résultats obtenus dans les cellules opération de l'organe mémoire $\omega + 1$, $\omega + 2$, ..., $\omega + N$ dans l'organe mémoire externe, nous introduirons dans le programme les instructions

Ea	*p*	$\omega + 1$	$r + 1$
Eb	*M*	$\omega + N$	

Nous obtiendrons le programme.

Programme 20.

Nombres

α_1	1	n
α_2	1	

Instructions

$k + 1$	$\times$	α_1	α_1	$\omega + 1$
$k + 2$	$+$	α_1	α_2	α_1

Nombres

α_3	N	
$\omega+1$		1^2
⋮		
$\omega+N$		N^2
β	«1» $A3$	

Instructions

$k+3$	$\oplus$	$k+1$	β	$k+1$
$k+4$	$\leqslant$	α_1	α_3	$k+1$
$k+5$	Ea	p	$\omega+1$	$r+1$
$k+6$	Eb	M	$\omega+N$	

Pour vérifier la fin des calculs, on peut aussi utiliser la $(k+1)$-ième instruction dont la forme au début du N-ième cycle est connue :

$k+1$	$\times$	α_1	α_1	$\omega+N$

Plaçons maintenant dans la cellule α_3 le code correspondant à l'instruction $(k+1)$, et la $(k+4)$-ième instruction est remplacée par l'instruction :

$k+4$	$\leqslant$	$k+1$	α_3	$k+1$

Nous obtiendrons le programme.

Programme 20_1.

Nombres

α_1	1			
α_2	1			
$\omega+1$		1^2		
⋮				
$\omega+N$		N^2		
β				1
α_3	$\times$	α_1	α_1	$\omega+N$

Instructions

$k+1$	$\times$	α_1	α_1	$\omega+1$
$k+2$	$+$	α_1	α_2	α_1
$k+3$	$\oplus$	$k+1$	β	$k+1$
$k+4$	$\leqslant$	$k+1$	α_3	$k+1$
$k+5$	Ea	p	$\omega+1$	$r+1$
$k+6$	Eb	M	$\omega+N$	
$k+7$	ARR			

***Exemple* 4.** *Calcul des valeurs d'un polynôme avec application de l'instruction de substitution d'adresse.*

Construisons le programme attaché à l'exemple 2 de ce paragraphe, en utilisant l'instruction de substitution d'adresse.

Répartissons maintenant les coefficients du polynôme

$$a_0, a_1, \ldots, a_n \text{ et } x$$

respectivement dans les cellules de l'organe mémoire :

$$\omega, \alpha + 1, \alpha + 2, \ldots, \alpha + n, \alpha .$$

L'instruction qui porte le numéro $N + 2k$ du programme 19 s'écrira :

$N + 2k$	+	ω	$\alpha + k$	ω

et la première instruction prendra la forme de ses instructions impaires.

Il est ainsi possible de diviser le calcul des valeurs d'un polynôme en cycles composés des instructions :

$N + 2k - 1$	×	ω	α	ω
$N + 2k$	+	ω	$\alpha + k$	ω

L'instruction $N + 2k$ a une seconde adresse variable qui doit être augmentée d'une unité lors du passage d'un cycle à l'autre. Pour établir le programme cyclique, nous compléterons les instructions $N + 1$, $N + 2$

$N + 1$	×	ω	α	ω
$N + 2$	+	ω	$\alpha + 1$	ω

par l'instruction de changement d'adresse :

$N + 3$	$\oplus$	$N + 2$	β_4	$N + 2$

où l'on place un nombre auxiliaire dans la cellule β_4 :

β_4			1	

Pour achever les calculs nous utiliserons le même compteur que dans le programme précédent : grâce à quoi le programme du cycle sera terminé.

Cependant, pour pouvoir répéter les calculs selon un programme donné, il convient de le compléter par une instruction (qui ne rentre pas dans le cycle) restaurant l'état initial de la $(N + 2)$-ième instruction variable.

Il est possible de le faire de deux manières :

1) Rétablissement de l'instruction variable du programme à l'aide de l'instruction de substitution d'adresse. Pour rétablir la forme initiale de la $(N + 2)$-ième instruction, nous exécuterons à la fin du programme cyclique l'instruction

$N + 5$	$\ominus$	$N + 2$	β_5	$N + 2$

où dans la cellule β_5 est placé un nombre auxiliaire :

β_5			n	

Nous obtiendrons le programme :

Programme 21.

Nombres

α_1	x	
ω	a_0	$f(x)$
$\alpha + 1$	a_1	
$\alpha + 2$	a_2	
$\vdots$		
$\alpha + n$	a_n	

β_1			1	
β_5			n	

β_2	n
β_3	0
β_4	1

k

Instructions

N	$+$	β_3	β_4	β_3
$N + 1$	$\times$	ω	α_1	ω
$N + 2$	$+$	ω	$\alpha + 1$	ω
$N + 3$	$\oplus$	$N + 2$	β_1	$N + 2$
$N + 4$	$\leqslant$	β_3	β_2	N
$N + 5$	$\ominus$	$N + 2$	β_5	$N + 2$
$N + 6$	ARR			

Le nombre de cellules utilisées dans le programme peut être réduit, si on prend comme unité pour le compteur (qui est en cellule β_4) l'unité de la seconde adresse (qui est en cellule β_1), et de façon analogue, comme constante pour la comparaison (qui est en cellule β_2) la constante qui est en β_5.

Programme 21_1.

Nombres

α_1	x			
ω	a_0	$f(x)$		
$\alpha + 1$	a_1			
⋮	⋮			
$\alpha + n$	a_n			
β_1			1	
β_2			n	
β_3	0			

Instructions

N	+	β_3	β_1	β_3
$N+1$	×	ω	α_1	ω
$N+2$	+	ω	$\alpha+1$	ω
$N+3$	⊕	$N+2$	β_1	$N+2$
$N+4$	⩽	β_3	β_2	N
$N+5$	⊖	$N+2$	β_2	$N+2$
$N+6$	*ARR*			

2) Rétablissement de la commande variable à l'aide de l'envoi à son adresse du code correspondant.

Nous placerons dans la cellule de l'organe mémoire β un nombre auxiliaire

β	+	ω	$\alpha+1$	ω

et nous compléterons le programme des instructions N, $N+1$, ..., $N+4$ par l'instruction de transfert

⊕		β	$N+2$

A la suite de l'exécution de cette instruction, le programme contenant une instruction variable acquiert une forme initiale propre à la répétition des calculs. De plus, il est évident qu'avant le début de la répétition des calculs les nombres doivent être dûment préparés.

Programme 22.

Nombres

α_1	x			
ω	a_0	$f(x)$		
$\alpha + 1$	a_1			
⋮				
$\alpha + n$	a_n			
β_1			1	
β_2			n	
β_3	0			
β	+	ω	$\alpha + 1$	ω

Instructions

N	+	β_3	β_1	β_3
$N+1$	×	ω	α_1	ω
$N+2$	+	ω	$\alpha + 1$	ω
$N+3$	⊕	$N+2$	β_1	$N+2$
$N+4$	⩽	β_3	β_2	N
$N+5$	⊕		β	$N+2$
$N+6$	*ARR*			

Dans le premier cas, comme dans le second, l'instruction de rétablissement peut être placée avant les instructions du cycle. Dans le premier cas, cela oblige à changer correctement l'instruction variable correspondante. Dans le second cas, le choix du code que l'on met dans cette mémoire n'a pas d'importance. Ainsi, d'après la quantité d'information introduite, le second moyen de rétablissement des instructions variables est plus rationnel. En outre, si le travail de la machine a tendance à s'embrouiller, le second moyen s'avère plus efficace : la constante introduite à l'endroit du code altéré le rétablit dans sa forme réelle, tandis que, par le premier moyen, la confusion surgissant dans l'instruction variable ne sera pas éliminée par l'instruction de rétablissement.

Le programme 22 peut être raccourci si l'on utilise le fait que la fin des calculs doit se produire après que l'instruction $N + 2$ a pris la forme :

γ	+	ω	$\alpha + n$	ω

Ainsi, on peut exclure du programme 22 l'instruction N (et la mémoire β_3) ; et pour indiquer la fin des calculs et le transfert de la commande au début du nouveau cycle, on peut employer la comparaison de l'instruction variable avec l'instruction constante correspondant au code γ.

Programme 22_1.

Nombres

α_1	x	
ω	a_0	$f(x)$
$\alpha + 1$	a_1	
⋮		
$\alpha + n$	a_n	

β_1			1	
β_2			n	
γ_3	+	ω	$\alpha + n$	ω

Instructions

$N + 1$	×	ω	α_1	ω
$N + 2$	+	ω	$\alpha + 1$	ω
$N + 3$	$\oplus$	$N + 2$	β_1	$N + 2$
$N + 4$	$\leqslant$	$N + 2$	γ	$N + 1$
$N + 5$	$\ominus$	$N + 2$	β_2	$N + 2$
$N + 6$	*ARR*			

Le problème posé peut être résolu autrement, par le choix de la cellule dite standard.

En particulier, nous choisirons la cellule de l'organe mémoire standard δ, dans laquelle, au début de chaque i-ième cycle de calcul, nous insérerons successivement les coefficients a_i, $i = 1, 2, \ldots, n$, à l'aide de l'instruction de transfert

(*)	+		$\alpha + i$	δ

(l'état initial de la mémoire δ n'a pas d'importance).

La première adresse de cette instruction est variable, ce qui fait que nous compléterons le programme du cycle avec l'instruction de substitution d'adresse :

(**)	$\oplus$	(*)	β_1	(*)

où, dans la cellule β_1, on trouve « 1 » de A_2. L'état initial de l'instruction (*) doit être le suivant :

(*)	+		$\alpha + 1$	δ

Plaçons l'instruction (*) après l'instruction (**) ; dans ce dernier cas, on doit d'abord placer le nombre a_1 dans la cellule $\alpha + 1$ et le numéro $\alpha + 2$ dans la première adresse de l'instruction (*).

Programme 22_2.

Nombres

α	x			
$\alpha + 1$	a_1			
⋮				
$\alpha + n$	a_n			
ω	a_0	$f(x)$		
δ		de travail		
β_1			1	
β_2	+		$\alpha + n$	δ
β_3	+		$\alpha + 1$	δ

Instructions

N	+		$\alpha + 1$	δ
$N + 1$	×	ω	α	ω
$N + 2$	+	ω	δ	ω
$N + 3$	$\oplus$	N	β_1	N
$N + 4$	$\leqslant$	N	β_2	N
$N + 5$	$\oplus$		β_3	N
$N + 6$	*ARR*			

Ici, pour déterminer l'endroit où arrêter le processus cyclique, on utilise une instruction variable. Le dernier programme a une instruction de plus ; cependant, dans de nombreux cas une telle réalisation de programme pour un processus cyclique dépendant d'un paramètre est souvent commode et raccourcit énormément le programme. L'exemple suivant est cité pour illustrer la thèse avancée.

***Exemple* 5.** *Calcul d'une somme.*

$$S = \sum_{i=1}^{n} \frac{x_i^3 + a}{x_i \sqrt{x_i}(x_i - a)}.$$

Les calculs se divisent naturellement en n cycles : dans le k-ième cycle ($k = 1, 2, \ldots, n$) le k-ième terme est calculé

$$A_k = \frac{x_k^3 + a}{x_k \sqrt{x_k}(x_k - a)}$$

et ajouté à la somme des termes précédents s_k. Il est évident que les instructions qui effectuent les calculs dans chacun des cycles dépendront du numéro du cycle.

Pour établir le programme cyclique nous disposerons les données de la manière suivante :

N° de la cellule	$\alpha + 1$	$\alpha + 2$	. . .	$a + n$	β
Son contenu	x_1	x_2		x_n	a

et nous choisirons la cellule opérative ω pour stocker la somme ; au début nous y placerons un « 0 ».

Le programme des calculs au k-ième cycle sera le suivant :

$N + 1$	$\times$	$\alpha + k$	$\alpha + k$	ω_1	x_k^2	A_1, A_2
$N + 2$	$\times$	ω_1	$\alpha + k$	ω_1	x_k^3	A_2
$N + 3$	$+$	ω_1	β	ω_1	$x_k^3 + a$	
$N + 4$	$\sqrt{\ }$	$\alpha + k$		ω_2	$\sqrt{x_k}$	A_1
$N + 5$	$\times$	ω_2	$\alpha + k$	ω_2	$x_k \sqrt{x_k}$	A_2
$N + 6$	$-$	$\alpha + k$	β	ω_3	$x_k - a$	A_1
$N + 7$	$\times$	ω_2	ω_3	ω_2	$x_k \sqrt{x_k}(x_k - a)$	
$N + 8$	:	ω_1	ω_2	ω_1	A_k	
$N + 9$	$+$	ω	ω_1	ω	s_k	

Les adresses des instructions qui dépendent du paramètre — du numéro de cycle — sont indiquées à droite sur le tableau.

La présence d'adresses variables impose que le programme du cycle soit complété par des instructions de substitution d'adresse. De plus, il est facile de remarquer que le nombre des constantes de substitution d'adresse peut être diminué, si, après avoir utilisé la commutativité de la multiplication, on change de place les A_1 et A_2 des instructions $N + 2$ et $N + 5$.

Pour déterminer où se terminent les calculs, nous utiliserons l'une des instructions variables, par exemple la $(N + 2)$-ième pour laquelle on place

dans la mémoire γ la constante auxiliaire correspondante :

γ	$\times$	ω_1	$\alpha + n$	ω_1

Le programme aura la forme :

Programme 23.

Nombres

$\alpha + 1$	x_1	
$\alpha + 2$	x_2	
$\vdots$		
$\alpha + n$	x_n	
β	a	
ω	0	s
ω_1		de travail
ω_2		
ω_3		

γ	$\times$	$\alpha + n$	ω_1	ω_1
γ_1		1	1	
γ_2		1		

Instructions

$N+1$	$\times$	$\alpha + 1$	$\alpha + 1$	ω_1
$N+2$	$\times$	$\alpha + 1$	ω_1	ω_1
$N+3$	$+$	ω_1	β	ω_1
$N+4$	$\sqrt{\ }$	$\alpha + 1$		ω_2
$N+5$	$\times$	$\alpha + 1$	ω_2	ω_2
$N+6$	$-$	$\alpha + 1$	β	ω_3
$N+7$	$\times$	ω_2	ω_3	ω_2
$N+8$	$:$	ω_1	ω_2	ω_1
$N+9$	$+$	ω	ω_1	ω
$N+10$	$\oplus$	$N+1$	γ_1	$N+1$
$N+11$	$\oplus$	$N+2$	γ_2	$N+2$
$N+12$	$\oplus$	$N+4$	γ_2	$N+4$
$N+13$	$\oplus$	$N+5$	γ_2	$N+5$
$N+14$	$\oplus$	$N+6$	γ_2	$N+6$
$N+15$	$\leqslant$	$N+2$	γ_2	$N+1$
$N+16$	*ARR*			

Si, selon les programmes, la répétition des calculs est possible (par exemple au cas où le groupe d'instructions est une étape intermédiaire de calcul à l'intérieur d'un autre cycle) elle doit être exécutée par les cinq instructions de rétablissement, alors que la totalité des nombres doit l'être par les constantes de rétablissement.

Nous utiliserons alors le procédé d'envoi dans une cellule standard. Dans ce cas, nous choisirons la cellule (opération) standard δ et nous compléterons le cycle par l'instruction de transfert

N	+		$\alpha + 1$	δ

Dès lors, les instructions $N + 1$, $N + 2$, $N + 4$, $N + 5$, $N + 6$ deviennent indépendantes du paramètre, et au lieu des cinq instructions de substitution d'adresse, il convient d'inclure l'instruction

$N + 10$	$\oplus$	N	γ_2	N

D'autre part, la constante de substitution d'adresse n'est plus nécessaire. Nous obtenons :

Programme 23_1.

Nombres

$\alpha + 1$	x_1			
$\alpha + 2$	x_2			
⋮				
$\alpha + n$	x_n			
β	a			
ω	0	s		
ω_1		}		
ω_2		} de travail		
ω_3		}		
γ_1			1	
γ_2	+		$\alpha + n$	δ

Instructions

N	+		$\alpha + 1$	δ
$N + 1$	×	δ	δ	ω_1
$N + 2$	×	δ	ω_1	ω_1
$N + 3$	+	ω_1	β	ω_1
$N + 4$	$\sqrt{\ }$	δ		ω_2
$N + 5$	×	δ	ω_2	ω_2
$N + 6$	−	δ	β	ω_3
$N + 7$	×	ω_2	ω_3	ω_2
$N + 8$	:	ω_1	ω_2	ω_1
$N + 9$	+	ω	ω_1	ω
$N + 10$	$\oplus$	N	γ_1	N
$N + 11$	$\leqslant$	N	γ_2	N
$N + 12$	ARR			

On fait encore une plus grande économie de cellules si on rétablit les instructions variables. Notons que, si l'on change l'ordre des calculs dans ce programme, on peut économiser une cellule de travail.

***Exemple* 6.** *Calcul d'une intégrale.*

$$I = \int_0^1 y\,dx, \quad \text{où} \quad y = \frac{\sqrt{1+x^2} - \sqrt{1-x^2}}{1+x}. \tag{30}$$

D'après la formule de Simpson :

$$\int_a^b y\,dx = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + \cdots + 2y_{2n-2} + 4y_{2n-1} + y_{2n}) \tag{31}$$

pour un nombre donné de divisions $2n$; $y_i = y(x_i)$, $x_i = x_0 + ih$,

$$h = \frac{b-a}{n}.$$

Puisque les calculs de y_1 pour toutes les valeurs de $i = 0, 1, 2, \ldots, 2n$ peuvent être exécutés à l'aide d'un même groupe d'instructions, il semble normal de faire le calcul préalable de ces grandeurs en les stockant : il ne restera plus après qu'à faire les calculs selon la formule (31). Cependant, dans ce cas, on a besoin de $2n + 1$ cellules de l'organe mémoire pour la mémorisation des grandeurs y_i.

Programme 24.

Nombres

α	x_i	
β	1	
γ	h	
ω_1		de travail
ω_2		de travail
$\omega + 1$		y_i

Instructions

$k+1$	$\times$	α	α	ω_1	x_i^2
$k+2$	$+$	β	ω_1	ω_2	$1 + x_i^2$
$k+3$	$\sqrt{\ }$	ω_2		ω_2	$\sqrt{1+x_i^2}$
$k+4$	$-$	β	ω_1	ω_1	$1 - x_i^2$
$k+5$	$\sqrt{\ }$	ω_1		ω_1	$\sqrt{1-x_i^2}$
$k+6$	$-$	ω_2	ω_1	ω_2	$\sqrt{1+x_i^2} - \sqrt{1-x_i^2}$
$k+7$	$+$	β	α	ω_2	$1 + x_i$
$k+8$	$:$	ω_1	ω_2	$\omega + 1$	y_i
$k+9$	$+$	α	γ	α	x_{i+1}

(*A*)

Le nombre des cellules indispensables à la mémorisation des grandeurs y_i peut être considérablement réduit si nous prenons le schéma de calcul suivant :

$$\left.\begin{aligned} \Delta s_i &= \frac{h}{3}(y_{2i} + 4\,y_{2i+1} + y_{2i+2})\,, \\ s_{i+1} &= s_i + \Delta s_i\ ;\quad i = 0, 1, \ldots n, n-1\ ;\quad s_0 = 0\,, \\ I &= s_n\,. \end{aligned}\right\}. \qquad (32)$$

Dès lors, pour passer du calcul des grandeurs y_{2i}, y_{2i+1}, y_{2i+2} au calcul de s_{i+1}, il est indispensable de garder en mémoire seulement les grandeurs s_i, y_{2i}, y_{2i+1}, y_{2i+2}. Le calcul selon le schéma (32) peut se diviser en une série de cycles : au i-ième cycle, on calcule la grandeur Δs_i et on l'ajoute à s_i. Le nombre total de cycles est égal à n.

Il faut calculer à leur tour les grandeurs y_{2i}, y_{2i+1}, y_{2i+2}, indispensables au calcul de Δs_i à l'aide d'un programme composé de trois cycles. Nous choisirons effectivement pour les grandeurs la suite de cellules $\omega + 1$, $\omega + 2$, $\omega + 3$. Le programme (A) utilisé pour le calcul de la grandeur y_{2i} peut alors l'être aussi pour calculer y_{2i+1} et y_{2i+2}, si l'on exécute une opération de substitution d'adresse sur l'instruction qui contient l'adresse $\omega + 1$. Ce qui fait que nous compléterons le programme (A) par l'instruction de substitution d'adresse

$k + 10$	+	$k + 8$	δ	$k + 8$

où

δ_1				1

Les instructions

$k + 11$	$\leqslant$	$k + 8$	δ_2	$k + 1$
$k + 12$	+		δ_3	$k + 8$

où

δ_2	:	ω_1	ω_2	$\omega + 3$
δ_3	:	ω_1	ω_2	$\omega + 1$

effectuent la triple répétition du cycle et le rétablissement de l'instruction variable pour les calculs répétés au pas suivant. Puisqu'au pas suivant on a de nouveau besoin de la grandeur y_{2i+1}, nous introduirons dans le programme l'instruction

$k + 13$	$-$	α	γ	α

rétablissant la valeur indispensable de l'argument. Pour déterminer le moment où les calculs s'arrêteront, nous utiliserons la valeur connue de x au dernier cycle $x = 1$.

Nous obtiendrons le programme 24_1.

Programme 24_1.

Nombres

α	x_0	
β	1	
γ	h	
ω_1		
ω_1		
$\omega + 1$		y_{2i}
$\omega + 2$		y_{2i+1}
$\omega + 3$		y_{2i+2}

δ_1				1
δ_2	:	ω_1	ω_2	$\omega + 3$
δ_3	:	ω_1	ω_2	$\omega + 1$
β_1	4			
β_2	$h/3$			
β_3	0			

Instructions

$k + 1$	$\times$	α	α	ω_1
$k + 2$	$+$	β	ω_1	ω_2
$k + 3$	$\sqrt{\ }$	ω_2		ω_2
$k + 4$	$-$	β	ω_1	ω_1
$k + 5$	$\sqrt{\ }$	ω_1		ω_1
$k + 6$	$-$	ω_2	ω_1	ω_1
$k + 7$	$+$	β	α	ω_2
$k + 8$	:	ω_1	ω_2	$\omega + 1$
$k + 9$	$+$	α	γ	α
$k + 10$	$\oplus$	$k + 8$	δ_1	$k + 8$
$k + 11$	$\leqslant$	$k + 8$	δ_2	$k + 1$
$k + 12$	$\oplus$		δ_3	$k + 8$
$k + 13$	$-$	α	γ	α
$k + 14$	$\times$	$\omega + 2$	β_1	ω_1
$k + 15$	$+$	ω_1	$\omega + 1$	ω_1

	Instructions			
$k + 16$	$+$	ω_1	$\omega + 3$	ω_1
$k + 17$	$\times$	ω_1	β_2	ω_1
$k + 18$	$+$	β_3	ω_1	β_3
$k + 19$	$\leqslant$	α	β	$k + 1$
$k + 20$	I			β_3
$k + 21$	ARR			

Le défaut de ce programme réside en ce que la grandeur y_{2i+2} utilisée au i-ième pas (comme contenu de la mémoire $\omega + 3$) et qui se rencontre dans les calculs au $(i + 1)$-ième pas (comme contenu de la mémoire $\omega + 1$) se calcule chaque fois d'une façon itérative pour tout $x_i = x_0 + ih$: ce qui dépense du temps-travail de la machine. En outre, la présence de la $(k + 8)$-ième instruction variable entraîne à sa suite l'exécution des instructions de substitution d'adresse $k + 10$ et de rétablissement $k + 12$, et exige de plus le chargement des cellules δ_1 et δ_3.

Relativement à ce qui a été exposé, nous allons prendre le procédé d'établissement d'un programme, dit procédé de « circulation des grandeurs dans les cellules standards ».

Plaçons la valeur calculée y dans la cellule $\omega + 4$ et introduisons dans le programme les instructions de transfert

$k + 10$	$+$	$\omega + 2$		$\omega + 1$
$k + 11$	$+$	$\omega + 3$		$\omega + 2$
$k + 12$	$+$	$\omega + 4$		$\omega + 3$

où

IG	$\omega + 2$	3	$\omega + 1$

et les instructions

$k + 13$	$-$		δ	δ
$k + 14$	$\leqslant$		δ	$k + 1$

où, au début, on place dans la cellule δ le code « -0 », assurant, à double reprise, la répétition du cycle. Dès lors, l'instruction de substitution d'adresse et celle de rétablissement ne sont plus nécessaires, et l'instruction $k+13$ et les cellules utilisées comme constantes auxiliaires sont libérées.

Nous obtenons le programme 24_2.

Programme 24_2.

Nombres

α	x_0	
β	1	
γ	h	
ω_1		
ω_2		
$\omega+1$		y_{2i}
$\omega+2$		y_{2i+1}
$\omega+3$	y_0	y_{2i+2}
δ	-0	
β_1	4	
β_2	$\frac{h}{3}$	
β_3	0	

Instructions

$k+1$	$\times$	α	α	ω_1
$k+2$	$+$	β	ω_1	ω_2
$k+8$	:	ω_1	ω_2	$\omega+4$
$k+9$	$+$	α	γ	α
$k+10$	$+$	$\omega+2$		$\omega+1$
$k+11$	$+$	$\omega+3$		$\omega+2$
$k+12$	$+$	$\omega+4$		$\omega+3$
$k+13$	$-$		δ	δ
$k+14$	$\leqslant$		δ	$k+1$
$k+15$	$\times$	$\omega+2$	β_1	ω_1
$k+16$	$+$	ω_1	$\omega+1$	ω_1
$k+17$	$+$	ω_1	$\omega+3$	ω_1
$k+18$	$\times$	ω_1	β_2	ω_1
$k+19$	$+$	β_3	ω_1	β_3
$k+20$	$\leqslant$	α	β	$k+1$
$k+21$	I			β_3
$k+22$	ARR			

D'après ce programme la grandeur y_0, indispensable aux calculs du premier pas, se calcule d'avance et se forme dans la cellule $\omega+3$. Pour éviter des

calculs supplémentaires, on peut changer les instructions qui dirigent la répétition du cycle interne de telle sorte que le cycle interne s'exécute trois fois au premier pas, deux fois aux suivants.

Donnons un exemple.

Plaçons dans la mémoire δ la grandeur « 3 » et à la place des instructions $k + 13$, $k + 14$ nous compléterons le programme par les instructions

$k + 13$	$-$	δ	β	δ
$k + 14$	$\leqslant$		δ	$k + 1$
$k + 15$	$+$		δ_2	δ

où dans la mémoire δ_2 sera placée le nombre « 2 ».

L'instruction $k + 15$ rétablit le contenu de la mémoire δ en la forme indispensable pour exécuter les calculs qui suivent le premier cycle. Dans ce cas, le remplissage initial de la mémoire $\omega + 3$ est sans importance.

***Exemple* 7.** *Décalage de la suite des nombres* $\alpha_1, \alpha_2, \ldots, \alpha_n$.

Décaler les nombres $\alpha_1, \alpha_2, \ldots, \alpha_n$ (dans la machine à virgule fixe) d'un certain nombre de positions vers la gauche (le même pour tous les nombres) tel qu'au moins l'un des deux ait un module supérieur ou égal à 1/2.

Plaçons les nombres donnés respectivement dans les cellules opération $\alpha + 1, \alpha + 2, \ldots, \alpha + n$. Le problème se divise en une série de cycles dans chacun desquels on doit vérifier la condition :

$$P\left(a_i < \frac{1}{2};\quad i = 1, 2, \ldots, n\right) = P_1\left(a_1 < \frac{1}{2}\right) \wedge P_2\left(a_2 < \frac{1}{2}\right) \wedge \ldots \wedge P_n\left(a_n < \frac{1}{2}\right), \qquad (33)$$

et si cette condition est satisfaite, le groupe de nombres se déplace d'une position à gauche. Sinon, les calculs s'arrêteront.

Dans l'exemple donné, le nombre de cycles n'est pas connu d'avance. La vérification de la condition qui détermine le nombre de cycles doit être faite au début du cycle, car il peut y avoir des cas où le nombre de cycles est égal à zéro. Comme nous l'avons vu dans le programme 9, dans ce cas, le cycle doit être complété à la fin par l'instruction de transfert d'ordre inconditionnel à sa première instruction.

Plaçons les grandeurs $a_1, \ldots, a_n$ respectivement dans les cellules $\alpha_1, \alpha_2, \ldots, \alpha_n$, le nombre 1/2 dans la cellule β. Le programme qui permet alors de résoudre le problème est représenté par le programme 25.

Programme 25.

Nombres

α_1	a_1
α_2	a_2
⋮	
α_n	a_n
β	$\frac{1}{2}$

Instructions

$N+1$	$\mid \leqslant \mid$	β	α_1	$N+2n+2$
$N+2$	$\mid \leqslant \mid$	β	α_2	$N+2n+2$
⋮				
$N+n$	$\mid \leqslant \mid$	β	α_n	$N+2n+2$
$N+n+1$	:	α_1	β	α_1
$N+n+2$	:	α_2	β	α_2
⋮				
$N+2n$	:	α_n	β	α_n
$N+2n+1$	$\mid \leqslant \mid$			$N+1$
$N+2n+2$	*ARR*			

Les instructions $N+1, \ldots, N+n$ réalisent le test de l'exécution de la condition complexe (33); les instructions $N+n+1, \ldots, N+2n$ exécutent le décalage du groupe de cellules $\alpha+1, \ldots, \alpha+n$ d'une position à gauche. On connaît la relation de la logique mathématique

$$\overline{P_1 \wedge P_2 \wedge \ldots \wedge P_n} = \overline{P}_1 \vee \overline{P}_2 \vee \ldots \vee \overline{P}_n,$$

qui illustre le fait qu'il est possible de ne pas satisfaire la condition complexe $P = \wedge P_i$ seulement si l'une des conditions élémentaires $P_1, P_2, \ldots, P_n$ n'est pas remplie.

Cela est concrètement illustré par le groupe d'instructions $N+1, \ldots, N+n$. Le transfert de la commande à l'instruction $N+2n+2$ n'est possible que si au moins une des conditions élémentaires $P_1, \ldots, P_n$ n'est pas satisfaite.

Si, maintenant, on répartit les grandeurs $a_1, a_2, \ldots, a_n$ dans une suite de cellules dont les numéros constituent une progression arithmétique, par exemple dans la suite $\alpha+1, \alpha+2, \ldots, \alpha+n$, on peut établir pour chacun des groupes d'instructions $N+1, \ldots, N+n$ et $N+n+1, \ldots, N+2n$ des programmes cycliques avec un même nombre de cycles, égal à n.

Programme 25_1.

Nombres

$\alpha + 1$	a_1			
$\alpha + 2$	a_2			
⋮				
$\alpha + n$	a_n			
β	$\frac{1}{2}$			
γ	0			
γ_1	1			
γ_2	n			
δ_1			1	
δ_2		1		1
δ_3	$\mid \leqslant \mid$	β	$\alpha + 1$	$N+13$
δ_4	:	$\alpha + 1$	β	$\alpha + 1$

Instructions

N	+			γ
$N + 1$	+	γ	γ_1	γ
$N + 2$	$\mid \leqslant \mid$	β	$\alpha + 1$	$N + 13$
$N + 3$	$\oplus$	$N + 2$	δ_1	$N + 2$
$N + 4$	$\mid \leqslant \mid$	γ	γ_2	$N + 1$
$N + 5$	+	−	−	γ
$N + 6$	+	γ	γ_1	γ
$N + 7$	:	$\alpha + 1$	β	$\alpha + 1$
$N + 8$	$\oplus$	$N + 7$	δ_2	$N + 7$
$N + 9$	$\mid \leqslant \mid$	γ	γ_2	$N + 6$
$N + 10$	$\oplus$	−	δ_3	$N + 2$
$N + 11$	$\oplus$	−	δ_4	$N + 7$
$N + 12$	$\mid \leqslant \mid$			N
$N + 13$	*ARR*			

L'avantage de ce dernier programme consiste en ce qu'il contient un petit nombre d'instructions qui ne dépendent pas de la quantité des nombres envisagés dans le groupe. Ne dépend de ce nombre que le contenu de la mémoire γ_2, qui contient la constante qui sert à la comparaison du compteur γ. Cependant, à la différence des programmes précédents ce programme a des instructions variables : ce qui fait que pour effectuer le calcul, on doit placer ses instructions dans la mémoire opérative.

***Exemple* 8.** *Contrôle de l'exécution des inégalités* $a_0 > 0$, $a_1 > 0$, ..., $a_n > 0$.

Soit donnée une suite de grandeurs a_i $(i = 0, 1, 2, ..., n)$, fonctions des paramètres k_1 et k_2 de la forme :

$$a_i = \alpha_{i0} + \alpha_{i1} k_1 + \alpha_{i2} k_2 + \beta_i k_1 k_2 + \gamma_i k_1^2 + \delta_i k_2^2 . \qquad (34)$$

Vérifier si les inégalités ci-après sont satisfaites :

$$a_0 > 0, \quad a_1 > 0, \ldots, a_n > 0 . \tag{35}$$

De plus, si la condition (35) n'est pas satisfaite, il convient d'imprimer un signe conventionnel et, si elle l'est, passer la commande à une instruction quelconque.

Distribuons les paramètres donnés dans une cellule de l'organe mémoire

Numéro de la mémoire	Contenu
$r + i$	α_{i0}
$s + i$	α_{i1}
$t + i$	α_{i2}
$p + i$	β_i
$q + i$	γ_i
$v + i$	δ_i

Le contrôle de la condition (35) se divise naturellement en une série de cycles : le signe du coefficient a_i de l'équation se détermine au i-ième cycle, $i = 0$, 1, ..., n. Si $a_i > 0$ et $i < n$ (le contrôle du symbole indispensable n'est pas encore achevé), la commande se transmet au cycle suivant. Si $a_i \leqslant 0$ (ce qui signifie que la condition indiquée n'est pas satisfaite), le signe conditionnel s'imprime et les calculs s'arrêtent. Enfin, si $a_i > 0$, $i = n$, la commande se transmet, conformément à la condition, à l'instruction N.

Le cycle est, naturellement, complété par les instructions de substitution d'adresse appropriées, assurant l'état convenable de la mémoire, lors du passage d'un cycle à l'autre. Pour réduire le nombre des instructions dans les polynômes (34), on commence par faire les mises en facteur possibles, et en particulier le calcul des coefficients a_i est effectué d'après les formules :

$$a_i = \alpha_{i0} + k_1(\alpha_{i1} + \beta_i k_2 + \gamma_i k_1) + k_2(\alpha_{i2} + \delta_i k_2), \quad i = 0, 1, \ldots, n .$$

Programme 26.

Nombres

$r + i$	α_{i0}
$s + i$	α_{i1}

Instructions

$k + 1$	$\times$	p	ε_2	ω_1
$k + 2$	$+$	s	ω_1	ω_1

Nombres

$t+1$	α_{i2}			
$p+i$	β_i			
$q+i$	γ_i			
$v+i$	δ_i			
ε_1	k_1			
ε_2	k_2			
η	signe conditionnel			
Δ	« 1 » A1			
ξ	$\times$	$p+n$	ε_2	ω_1

Instructions

$k+3$	$\times$	q	ε_1	ω_2
$k+4$	$+$	ω_1	ω_2	ω_1
$k+5$	$\times$	ω_1	ε_1	ω_1
$k+6$	$+$	r	ω_1	ω_1
$k+7$	$\times$	v	ε_2	ω_2
$k+8$	$+$	t	ω_2	ω_2
$k+9$	$\times$	ω_2	ε_2	ω_2
$k+10$	$+$	ω_1	ω_2	ω_1
$k+11$	$\oplus$	$k+1$	Δ	$k+1$
$k+12$	$\oplus$	$k+2$	Δ	$k+2$
$k+13$	$\oplus$	$k+3$	Δ	$k+3$
$k+14$	$\oplus$	$k+6$	Δ	$k+6$
$k+15$	$\oplus$	$k+7$	Δ	$k+7$
$k+16$	$\oplus$	$k+8$	Δ	$k+8$
$k+17$	$\leqslant$	ω_1	$-$	$k+20$
$k+18$	$\leqslant$	$k+1$	ξ	$k+1$
$k+19$	$\leqslant$	$-$	$-$	N
$k+20$	I			η
$k+21$	ARR			

C'est un exemple de processus cyclique s'achevant par une condition logique complexe : soit au i-ième cycle, $i = 0, 1, \ldots, n$, d'après un symbole si le coefficient $a_i \leqslant 0$, soit au n-ième cycle d'après le compteur si $a_i > 0$ pour tout $i = 0, 1, \ldots, n$.

De plus, selon le moyen qu'on utilise pour terminer le processus cyclique, la commande est transmise par diverses instructions.

CONCLUSIONS

1) Le caractère cyclique des programmes de *processus de calcul*, dépendant de paramètres, est atteint grâce à une répartition ordonnée dans les cellules de l'organe mémoire des grandeurs dépendant des paramètres, et grâce à l'emploi des instructions de substitution d'adresse.

2) Les programmes des processus cycliques contiennent des instructions qui, à chaque cycle, préparent le chargement, par des nombres, de l'organe mémoire et des instructions variables pour l'exécution du cycle suivant.

3) Pour former les conditions logiques qui déterminent la fin des processus cycliques paramétriques, on peut utiliser les instructions variables du programme.

4) Pour appliquer d'une façon répétée un programme cyclique, il est indispensable de prévoir le rétablissement des instructions variables. Dans ce cas, on introduit dans le programme des instructions de rétablissement (n'entrant pas dans le cycle) qui s'effectuent :

a) à l'aide d'une opération de substitution d'adresse ;

b) à l'aide de l'envoi à une adresse d'instruction variable du code correspondant.

D'ailleurs, le rétablissement des instructions variables peut s'effectuer au moyen de l'opération de l'entrée répétée de tout le programme de la mémoire externe dans la mémoire interne.

5) A la suite de l'application de l'opération d'envoi des grandeurs dépendant d'un paramètre dans les mémoires standards, la partie numérique du programme peut être rendue invariable. De plus, dans beaucoup de cas, le nombre des instructions variables du programme diminue : ce qui réalise une économie dans les cellules de l'organe mémoire.

EXERCICES

1. Etablir un programme cyclique pour calculer la somme :

$$S = \sum_{i=1}^{n} \frac{x_i + a}{x_i - b}.$$

2. Etablir un programme avec transfert de la commande d'un cycle à l'autre par l'une des instructions suivantes : « = », « *TCS* » ou « *TCN* », pour calculer les valeurs d'un polynôme.

3. Etablir un programme pour calculer l'intégrale

$$I = \int_a^b y \, dx, \quad \text{où} \quad y = \frac{x \sin x}{1 + x^2},$$

d'après la formule de Simpson pour les différentes formes des transferts conditionnels de commande.

4. Etablir un programme pour l'exemple 7 du texte (programme 25_1) en utilisant, pour terminer les cycles internes, une forme de commandes variables connue d'avance.

5. Etablir un programme pour l'exemple 7 du texte en tenant compte du nombre des décalages (avec mémorisation de l'échelle) d'abord pour une machine à transfert conditionnel de la commande d'après un nombre et ensuite pour une machine à opération de transfert conditionnel de commande d'après une inégalité.

5. ORGANIGRAMMES ET TRANSFERT DE COMMANDE A DES SOUS-PROGRAMMES

Pour établir un programme pour des problèmes complexes, il est pratique d'utiliser la division en *blocs*.

On appelle « *bloc* », la partie de programme dont seule la première instruction reçoit le transfert de commande venant des autres blocs et dont seule la dernière instruction envoie le transfert de commande vers les autres blocs. A l'intérieur du bloc, on admet des transferts conditionnels aussi bien qu'inconditionnels de commande non liés avec la sortie sur les autres blocs.

En programmation, l'organigramme peut réduire considérablement les programmes grâce à la séparation en blocs des groupes d'instructions qui se répètent au cours du problème. On a adopté l'appellation de sous-programmes pour désigner ces groupes d'instructions. Des instructions spéciales de transfert de commande sur sous-programmes allègent, par leur utilisation répétée, le programme.

Si un groupe d'instructions est valable pour la résolution de nombreux problèmes, il est pratique d'en conserver les sous-programmes, soigneusement vérifiés, sous forme de bibliothèque sur cartes perforées ou sur bandes perforées, ou de les avoir en mémoire sous forme de blocs standardisés (de composition rapide) dans la partie interne de l'organe mémoire ou dans des emplacements de la mémoire externe spécialement assignés. Lorsqu'on compose un nouveau programme, les sous-programmes-bibliothèque peuvent y être insérés comme partie n'exigeant pas de contrôle itératif : ce qui raccourcit beaucoup le temps d'entrée et la vérification du programme.

Si l'on doit, après l'instruction k du programme principal, passer à l'exécution d'un sous-programme qui occupe m instructions et qui est donné dans

les adresses conditionelles 1, 2, ..., m, on peut simplement inclure le sous-programme dans le programme principal, sa première instruction étant devenue la $(k + 1)$-ième instruction du programme principal, la seconde, la $(k + 2)$-ième instruction, etc. De plus, les adresses de « ce » sous-programme dans lesquelles sont indiqués les numéros des instructions (et non des nombres) : adresses des instructions de transferts de commande (inconditionnels et conditionnels) et des instructions de substitutions d'adresse, doivent être changées selon la norme, à savoir augmentées de la quantité k. L'inclusion de sous-programmes dans le programme principal exige en premier lieu la transformation des instructions du sous-programme, et en second lieu, si le sous-programme nécessite des calculs itératifs, conduit à la répétition de toutes les instructions du sous-programme et allonge d'autant le programme.

On peut aussi procéder d'une autre façon : répartissons les instructions du sous-programme dans les cellules (libres) de l'organe mémoire fixées de $n + 1$ à $n + m$, et réservons une cellule de plus, de numéro $n + m + 1$, pour placer l'instruction dite instruction de renvoi au programme principal. Dans la cellule $k + 2$ du programme principal, nous placerons l'instruction de transfert inconditionnel à un sous-programme

TI			$n + 1$

et dans la $(k + 1)$-ième cellule, l'instruction préparant la $(n + m - 1)$-ième instruction au renvoi correct à la $(k + 3)$-ième instruction du programme principal, par exemple l'instruction

$\oplus$	γ		$n + m + 1$

γ étant le numéro de la cellule contenant le code

TI			$k + 3$

Prenons l'exemple d'un programme avec appels fréquemment répétés à des sous-programmes.

Supposons que, d'après un groupe d'instructions Q, nous calculions une fonction issue d'un argument, contenu dans la mémoire α, et, qu'en résolvant le problème, nous ayons à calculer les valeurs de cette fonction venant de deux arguments différents, placés au cours des calculs dans les mémoires β et γ.

Examinons dans le détail l'organigramme de ce programme, dans lequel le groupe des instructions Q sous forme de sous-programme.

Désignons par A le groupe des instructions effectuant les calculs jusqu'au premier appel au sous-programme Q, par B le groupe des instructions effectuant les calculs entre le premier et le deuxième appel au sous-programme et par C les calculs qui suivent le deuxième appel au sous-programme : le schéma du calcul aura alors la forme :

$$AQBQC\,. \tag{36}$$

La composition de chacun des groupes est

pour A, de p instructions ;
B, de r instructions ;
C, de s instructions ;
Q, de m instructions.

Pour faire les calculs sans mettre à part le sous-programme Q, le programme se compose de $p + r + s + 2m$ instructions. (Dans cet examen, nous laissons de côté les transferts conditionnels et inconditionnels de commande, non liés au passage à un sous-programme, que l'on peut rencontrer à l'intérieur de n'importe quel groupe d'instructions).

Pour faire automatiquement les calculs avec séparation du sous-programme Q, il faut les instructions supplémentaires suivantes :

1) Instructions effectuant, avant l'appel au sous-programme, l'envoi dans la cellule α de l'argument se trouvant respectivement dans les cellules β et γ.

2) Instructions effectuant le passage au sous-programme.

3) Instructions effectuant le renvoi du sous-programme vers les calculs ultérieurs du programme (aux groupes d'instructions B, après le premier appel au sous-programme, et C, après le second appel).

En les mettant à part, nous désignerons par a_1 et b_1 les instructions qui réalisent l'envoi de l'argument des cellules β et γ dans la cellule α spécialement affectée à l'argument du sous-programme ; a_2 et b_2 sont les instructions qui assurent le renvoi correct du sous-programme vers le programme principal ; a_3 et b_3 sont les instructions du transfert inconditionnel vers le sous-programme.

En outre, on doit introduire dans le sous-programme l'instruction variable q (instruction de « renvoi » vers le programme principal préparé par les instructions a_2 et b_2).

Le schéma du sous-programme aura la forme :

$$Aa_1\,a_2\,a_3\,Qq\,Bb_1\,b_2\,b_3\,QqC\,. \tag{37}$$

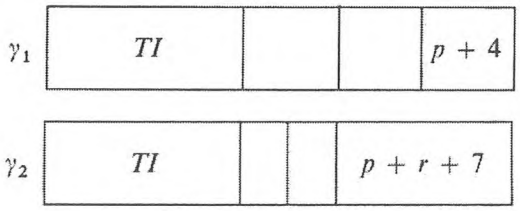
Le nombre d'instructions du programme, conformément à (37), sera égal à $p + r + s + m + 7$.

Pour compléter la préparation de l'instruction de renvoi, on utilise deux constantes dont les codes correspondent aux instructions de passage incondi-

tionnel à la première instruction du groupe B ou C. Nous les placerons dans les mémoires γ_1 et γ_2. Sur l'organigramme (Fig. 14) les flèches indiquent l'ordre d'exécution des instructions.

L'interaction des instructions correspondantes et de l'instruction de « renvoi » du programme vers le programme principal est indiquée par des pointillés. Les instructions mises à part dans le circuit sont développées dans l'une des variantes possibles.

Dans les cellules γ_1 et γ_2 on place les codes :



Dans la répartition que l'on a adoptée pour les instructions, dans les cellules de la mémoire du sous-programme Q on place $N + 1$, $N + 2$, ..., $N + m$, où $N \geqslant p + r + s + 6$.

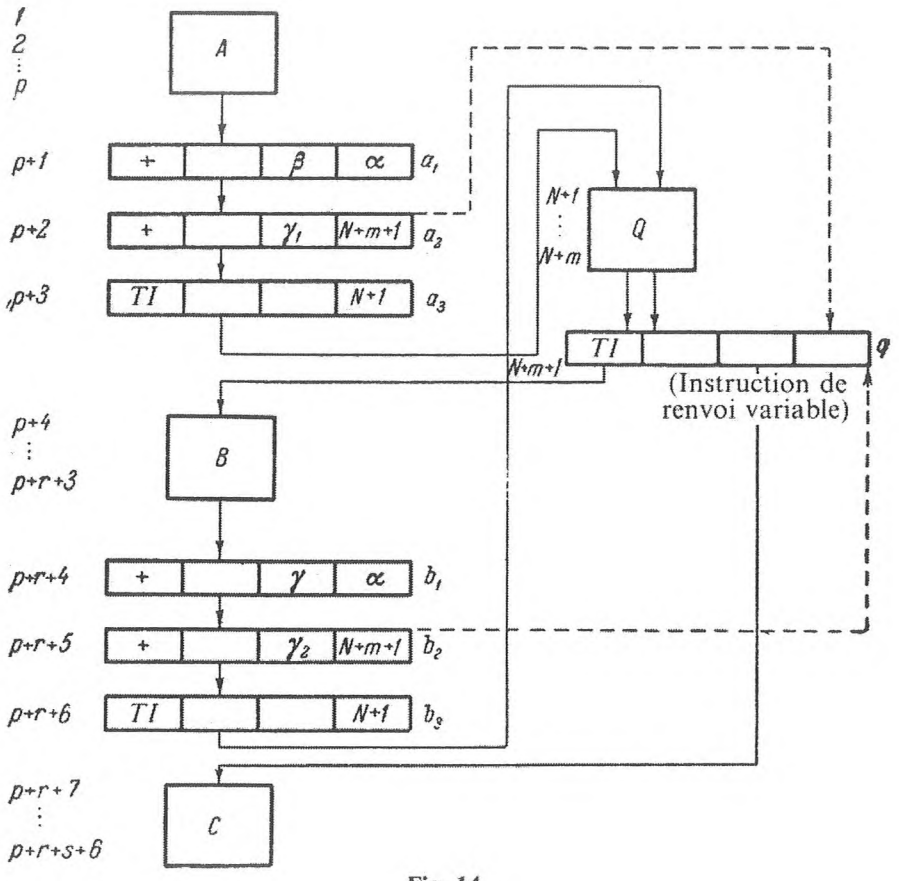


Fig. 14.

Pour que cela soit pratique, il faut terminer tout sous-programme par une instruction de passage inconditionnel à une même mémoire spécialement réservée dans la mémoire opération (« registre de renvoi ») et, lorsqu'on passe au sous-programme, il faut envoyer d'avance dans sa troisième adresse A_3 le numéro de l'instruction du programme principal à laquelle doit être transmise la commande après l'exécution du sous-programme.

En outre, dans le registre de renvoi, il faut placer le code de l'opération de passage inconditionnel.

En plus de ces opérations élémentaires effectuées par la machine, il est commode d'avoir :

1) Transfert inconditionnel aux sous-programmes *TIS* :

TIS	α	k_1	k_2

2) Transfert inconditionnel d'après le registre de renvoi : *TIRR*

TIRR			

La première de ces opérations effectue le transfert inconditionnel vers l'opération dont le numéro k_1 est indiqué en A_2 (le numéro de la première instruction du sous-programme) et envoie dans le registre de renvoi le numéro de l'instruction k_2 indiqué en A_3 (numéro du programme principal), auquel il faut transmettre la commande après exécution du sous-programme.

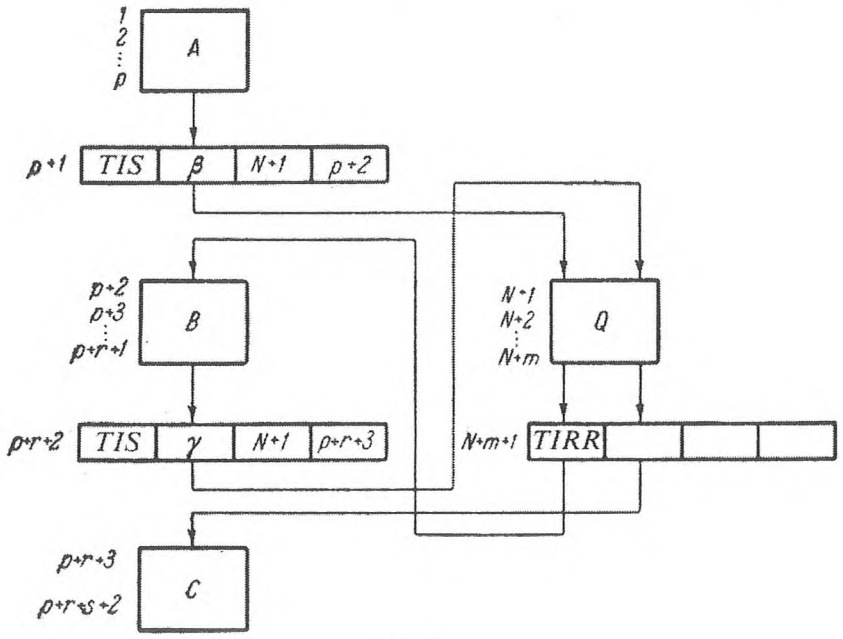


Fig. 15.

En outre, au cours de l'opération *TIS* on peut avoir besoin de l'adresse libre A_1 : le contenu de la cellule dont le numéro est indiqué en A_1 est envoyé dans une cellule « banalisée » (par exemple n° 1) prévue pour conserver « l'argument » de tous les sous-programmes. L'instruction *TIRR* transfère la commande à l'instruction dont le numéro se trouve, à un moment donné, dans le *RR* (registre-renvoi) (Fig. 15). Le nombre d'instructions du programme est égal à $p + r + s + m + 3$.

Parmi les opérations élémentaires effectuées par la machine à la suite des transferts sur sous-programmes, il est pratique d'avoir aussi les instructions :

3) Transfert conditionnel à un sous-programme d'après un « symbole »

TCSS	α	k_1	k_2

Cette opération

a) effectue le transfert de la commande à l'instruction dont le numéro est indiqué en A_2, si le symbole est élaboré dans l'instruction précédente ;

b) envoie dans le registre de renvoi le numéro de l'instruction indiquée en A_3 (numéro de l'instruction à laquelle doit être transmise la commande après l'exécution du sous-programme) ;

c) transmet la commande à l'instruction dont le numéro suit, si le symbole n'est pas élaboré. En outre, lors du transfert de la commande selon A_2, le contenu de la cellule α est envoyé dans la cellule standard prévue pour conserver les arguments du sous-programme.

4) Transfert conditionnel à un sous-programme d'après un nombre (*TCSN*)

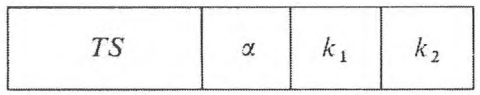
TCNS	α	k_1	k_2

Le contenu de cette instruction est le suivant : si dans la mémoire dont le numéro α est indiqué en A_1 il y a un nombre positif, la commande est transmise à l'instruction dont le numéro k_1 est indiqué en A_2 (à la première instruction du sous-programme), le contenu de A_3 est envoyé dans le registre de renvoi (pour un retour correct du sous-programme), et si ce nombre est négatif, la commande est transmise à l'instruction du programme dont le numéro suit. Si, au cours de l'exécution du sous-programme, on rencontrait à son tour un transfert à un second sous-programme, ce transfert à plusieurs degrés exigerait un registre de renvoi à plusieurs degrés aussi. Pour éviter les difficultés techniques que cela pourrait occasionner, il faut se restreindre à un seul registre de renvoi, ne renfermant pas de transferts vers les autres sous-programmes (sous-sous-programmes) ; mais pour les sous-programmes qui ont des sous-programmes, il faut utiliser les schémas indiqués dans l'exemple page 145 (Fig. 14).

Cependant, comme nous l'avons vu, aux moments des transferts vers des sous-programmes intermédiaires, on introduit dans le programme trois instructions spéciales qui ont chacune besoin d'une constante.

En conséquence, il est souhaitable d'avoir dans l'ensemble des opérations à effectuer par la machine une instruction de la forme :

5) Transfert à un sous-programme (*TS*)



Ici, l'instruction *TS* qui se trouve dans la cellule n transmet la commande à l'instruction dont le numéro k_1 est indiqué en A_2 ; elle envoie en A_3 de l'instruction k_2, indiquée en A_3, le numéro $n + 1$ de l'instruction qui suit. En outre, le contenu de la cellule dont le numéro est indiqué en A_1 est envoyé dans la cellule standard prévue pour conserver les arguments du programme standard. De plus, la cellule de numéro k_1 contient la première instruction du sous-programme, et la cellule de numéro k_2 contient le code de l'instruction de transfert inconditionnel, c'est la dernière du sous-programme.

Remarquons ici que, lors de la programmation, l'appel au sous-programme oblige le programme principal à tenir compte du chargement de l'organe mémoire. Par exemple, certaines cellules doivent être spécialement préparées au travail du sous-programme (envoi des arguments dans les cellules correspondantes, etc.) ; d'autres cellules peuvent être utilisées dans le sous-programme en tant que cellules opération ; par conséquent, on ne doit pas y conserver des grandeurs du programme principal nécessaires à la suite, etc.

Dans ce but, on utilise maintenant dans tous les sous-programmes chaque fois les mêmes cellules comme cellules opération, chaque fois les mêmes cellules comme emplacement des arguments, etc.

Les instructions de sous-programmes qui changent au cours du calcul doivent être rétablies pour être employées de façon itérative. C'est pourquoi les sous-programmes dans lesquels entrent des instructions variables commencent ou finissent par une instruction de rétablissement.

6. FORMATION DU CONTENU DE L'ORGANE MÉMOIRE

Dans les paragraphes précédents, l'unique condition qui déterminait le contenu de l'organe mémoire, était que les processus cycliques du calcul soient assurés.

D'autre part, il est souhaitable de répartir les nombres et les instructions dans les mémoires de façon à rendre le travail du programme (y compris l'entrée du programme en machine) le plus rapide et le plus sûr possible.

De plus, il convient de tenir compte de la corrélation entre les différentes parties de l'organe mémoire.

L'entrée des programmes — nombres et instructions initiaux — ne se faisant pas de façon rapide dans les machines construites jusqu'à présent, il faut essayer à tout prix de diminuer le volume des programmes.

Sur certaines machines, on a la possibilité d'entrer des parties de programme dans une partie fixe de l'organe mémoire interne et dans l'organe mémoire externe, en dehors du temps-travail de la machine. Les cellules de cette mémoire fixe, existant sous forme de blocs à composition rapide, ainsi que les blocs de l'organe mémoire externe sous forme de bandes magnétiques, etc., se préparent d'avance pour chaque programme (dans un emplacement particulier) avant de réaliser le programme pour la machine. Ordinairement, dans ces parties de l'organe mémoire, il y a des secteurs spécialement réservés pour conserver les constantes types et les sous-programmes standards, et selon la nécessité on introduira dans l'organe mémoire l'un ou l'autre bloc.

Ces programmes standards soigneusement vérifiés, en plus d'accélérer le processus d'entrée et de simplifier la programmation, rendent le travail de la machine plus sûr.

Il est utile de perfectionner et de compléter la bibliothèque des sous-programmes dans le processus d'utilisation de la machine.

Lors de la distribution des programmes standards entre les organes mémoire interne et externe, on tient compte, d'une part, du fait que l'appel à la mémoire externe est une opération relativement lente, et que, par conséquent, l'utilisation des sous-programmes qui y sont disposés augmente le temps des calculs, et d'autre part, du fait que l'organe mémoire externe est techniquement plus simple à réaliser. Comme il a déjà été indiqué dans l'introduction, pour exécuter les calculs, le programme ou la partie du programme dans laquelle doivent entrer les instructions doit se trouver dans l'organe mémoire interne (rapide). C'est pourquoi, il est plus pratique de placer dans l'organe mémoire externe les sous-programmes standards qui contiennent des instructions variables. Lorsque des calculs sont exécutés sur de tels sous-programmes, ces derniers seront placés dans l'organe mémoire opération, ce qui sera fait de la façon la plus simple à l'aide de l'opération « lecture » qui accomplira le transfert du sous-programme correspondant de l'organe mémoire externe à l'organe mémoire opération.

Il convient d'introduire dans l'organe mémoire externe les opérations, même de petits programmes, avant leur répartition intégrale dans l'organe mémoire interne.

En effet, lorsqu'on travaille sur machine, il peut y avoir des interruptions de différentes sortes (coupure de courant dans la machine, etc.) à la suite desquelles le contenu de la partie opérative de l'organe mémoire peut être altéré. L'existence d'une forme initiale du programme dans l'organe mémoire externe permet à l'aide de la « lecture » de rétablir le programme dans la mémoire opération sans appel aux organes d'entrée. C'est ainsi qu'à l'entrée et pendant

la mise au point du programme, on peut découvrir des fautes au moment où certaines instructions variables ont changé de forme. « Après avoir lu » le programme dans l'organe mémoire externe et après avoir fait les corrections nécessaires, il convient « d'écrire » le programme corrigé dans la mémoire externe. Cela allège la correction ultérieure du programme et le travail qu'elle donne.

D'autre part, pour de petits programmes répartis dans la mémoire opération il n'y a pas, au cours du travail, de possibilité d'appel à l'organe mémoire externe : le programme se transporte en entier dans l'organe mémoire interne, ensuite commencent les calculs.

Si le volume de la partie interne de l'organe mémoire n'est pas suffisant pour répartir le programme, l'appel à l'organe mémoire externe au cours du travail du programme devient inévitable. Dans ce cas, il est plus avantageux de placer en mémoire externe les parties du programme qui, au cours des calculs, sont assez rarement utilisées, car, comme nous l'avons déjà noté, les opérations relatives à l'appel à l'organe mémoire externe sont assez lentes. De plus, la masse des cartes et des bandes perforées, qui représente le programme codé, se divise en groupes correspondant aux groupes de codes qui sont introduits l'un après l'autre dans la mémoire opération et de là dans une place spécialement affectée de l'organe mémoire externe. Le programme prévoit au moment voulu, un appel (« lecture ») dans la mémoire opération des groupes de codes qui sont nécessaires au travail.

Certaines difficultés sont liées à la division du programme en de tels groupes.

En effet, on ne peut pas exécuter le transfert de la commande à une instruction qui ne serait pas contenue à un moment donné dans la mémoire opération. C'est pourquoi chaque groupe d'instructions placé dans l'organe mémoire interne doit posséder la particularité suivante : « la lecture » de l'instruction de ce groupe doit précéder chaque instruction de transfert de la commande à une instruction d'un autre groupe. De plus, on tient compte de ce qu'après l'opération « lecture » les mémoires dans lesquelles s'est faite « la lecture » conservent les codes lus.

L'utilisation de la mémoire externe est aussi recommandée pour un programme qui change au cours du travail tout en servant à plusieurs reprises, par exemple, pour un programme de processus cycliques dépendant de paramètres. La présence de la forme initiale d'un tel programme dans la mémoire externe permet de renouveler le programme pour répéter les calculs à l'aide de l'opération « lecture ». De plus, le volume du programme peut être réduit de façon à libérer des cellules pour conserver les constantes nécessaires aux instructions de rétablissement. En outre, le temps pris en machine par les calculs directs peut être un peu augmenté (l'opération « lecture » peut prendre beaucoup plus de temps que l'exécution des instructions de rétablissement) ; cependant, il y a une économie de temps à l'entrée du programme, de sorte que le volume de l'information d'entrée sur le programme est réduit.

La formation du contenu de l'organe mémoire dépend essentiellement de la place occupée par le programme dans le schéma général de résolution du problème. Pour des programmes de formes différentes, le contenu de l'organe mémoire se forme différemment.

Pour former le contenu de l'organe mémoire pour un programme qui est auxiliaire dans la résolution d'un problème, c'est-à-dire un sous-programme du programme principal, il est indispensable de tenir compte de ce que certaines données lui sont fournies par le programme principal, et se trouvent par conséquent dans les mémoires opération de l'organe mémoire.

Si un sous-programme est utilisé plusieurs fois, il est indispensable de prévoir le rétablissement du contenu de l'organe mémoire à la forme initiale, les instructions variables incluses.

Ici, nous nous bornerons à examiner la formation du contenu de l'organe mémoire seulement pour les programmes cités ci-dessus.

***Exemple* 1.** *Programme du calcul de la table des carrés des termes d'une progression arithmétique avec sortie sur l'imprimante* (cf. § 3, programme 9).

Le contenu de l'organe mémoire pour le programme 9 est composé de deux nombres constants c et $a + cN$, qui sont placés dans les cellules α_2 et α_3, de deux cellules de travail α_1 et ω, dans lesquelles se trouvent les termes de la progression arithmétique et leurs carrés, et d'instructions que l'on peut placer à n'importe quel endroit de l'organe mémoire.

Si le premier terme de la progression arithmétique a se trouve dans la cellule β, on doit avoir, au début du programme, l'instruction de transfert de a de la mémoire β dans la mémoire de travail α_1

$k-1$	+	β	−	α_1

***Exemple* 2.** *Extraction d'une racine carrée* (cf. § 3, programme 10).

Le programme de l'extraction d'une racine carrée est généralement un programme standard et se place dans les cellules passives. Les nombres y_0 et x doivent être dans les cellules de travail. D'ailleurs, pour une machine à virgule fixe, par exemple, il faut choisir y_0 indépendant de x et, par conséquent, le placer dans la mémoire passive β, après avoir prévu dans le programme une instruction d'envoi de y_0 de cette mémoire dans la mémoire de travail nécessaire.

***Exemple* 3.** *Calcul des valeurs d'un polynôme* (cf. § 4, programme 21).

Il est naturel de supposer que les coefficients du polynôme et la valeur de l'argument sont obtenus à la suite du calcul sur le programme principal et,

par conséquent, sont répartis dans les cellules de travail. Le mieux est de remettre à zéro le contenu de β_3, comme on l'a déjà mentionné ci-dessus, en introduisant au début du programme l'instruction

$N-1$	+	−	−	β_3

Les nombres $1(\beta_4)$ et 1 dans l'adresse $A_2(\beta_1)$ sont, en principe, dans toutes les machines, dans la mémoire passive. Le nombre n d'unités dans l'adresse A_2 et le nombre n — constants pour un programme donné — peuvent, soit se placer dans les mémoires passives, soit être formés dans les mémoires actives du programme principal. Les instructions du programme sont réparties dans les cellules actives, puisque le programme a des instructions variables.

Si le programme est placé en mémoire externe il faut, pour exécuter les calculs sur lui, des instructions appropriées qui assurent le transfert du programme dans l'organe mémoire interne (« lecture ») : ce qui fait que, d'une façon générale, la numérotation des instructions peut changer. Conformément à la nouvelle numérotation des instructions dans la mémoire interne, on doit changer les instructions qui contiennent dans leurs adresses les numéros des instructions. La 4e et la 6e instructions, dont les 1re et 3e adresses doivent être changées conformément au nouveau numéro de l'instruction variable, sont dans le programme 21 des instructions de ce genre.

***Exemple* 4.** *Formation du contenu de l'organe mémoire pour le programme* 12 (§ 3).

Le programme qui permet de calculer sin x est aussi un programme standard caractéristique. La valeur de l'argument x est préparée par le programme principal, par exemple, dans la cellule active α_1. Les nombres x de la cellule α_2 et x^2 dans la cellule α_4 doivent être formés dans le programme donné ; c'est-à-dire le programme 12 doit être complété par les instructions

$k-2$	+	α_1	−	α_2
$k-1$	×	α_1	α_1	α_4
k	−	−	α_4	α_4

La répartition initiale des nombres dans les mémoires est simplifiée :

α_1	x
α_2	—
α_3	1
α_4	—
α_5	1
α_6	ε

Le chargement initial des cellules α_2 et α_4 est indifférent.

***Exemple* 5.** *Réunion des programmes du calcul de* $\sin x$ *et* $\cos x$.

Les programmes du calcul de $\sin x$ et $\cos x$ ne se différencient que par le contenu initial des cellules numériques dans l'organe mémoire. C'est pourquoi on peut calculer $\sin x$ et $\cos x$ sur un même programme ; il faut seulement prévoir la formation correcte des cellules numériques dans l'organe mémoire. Comparons le contenu initial des cellules numériques dans l'organe mémoire pour les programmes de $\sin x$ et $\cos x$:

Contenu initial des nombres dans l'organe mémoire pour le programme de $\sin x$.

α_0	x	
α_1	x	u_n
α_2	x	s_n
α_3	1	c_n
α_4	$-x^2$	
α_5	1	
α_6	ε	

Contenu initial des nombres dans l'organe mémoire pour le programme de $\cos x$.

α_0	x	
α_1	1	u_n
α_2	1	s_n
α_3	0	c_n
α_4	$-x^2$	
α_5	1	
α_6	ε	

Les programmes de $\sin x$ et $\cos x$ se distinguent par le chargement des mémoires α_1, α_2 et α_3. La formation des nombres x et 1 est exécutée pour le

programme de sin x par les instructions suivantes :

+	α_0	−	α_1
+	α_0	−	α_2
+	α_5	−	α_3

Pour programmer cos x les nombres qui se trouvent dans les cellules $\alpha_1, \alpha_2, \alpha_3$ sont formés par les instructions

+	α_5	−	α_1
+	α_5	−	α_2
+	−	−	α_3

On peut donc établir un programme unique pour sin x et cos x :

Programme 27.

Nombres			Instructions				
α_0	x		$N+1$	+	α_0	−	α_1	début du calcul de sin x
α_1	−	u_n	$N+2$	+	α_0	−	α_2	
α_2	−	s_n	$N+3$	+	α_5	−	α_3	
α_3	−	c_n	$N+4$	$\mid \leqslant \mid$	−	−	$N+8$	
α_4	−	$-x^2$	$N+5$	+	α_5	−	α_1	début du calcul de cos x
α_5	1		$N+6$	+	α_5	−	α_2	
α_6	ε		$N+7$	+	−	−	α_3	
α_7	−		$N+8$	×	α_0	α_0	α_4	
			$N+9$	−	−	α_4	α_4	

Instructions

$N+10$	+	α_3	α_5	α_7
$N+11$	+	α_7	α_5	α_3
$N+12$	×	α_7	α_3	α_7
$N+13$	×	α_1	α_4	α_1
$N+14$	:	α_1	α_7	α_1
$N+15$	+	α_2	α_1	α_2
$N+16$	$\lvert \leqslant \rvert$	α_6	α_1	$N+10$
$N+17$	*ARR*			

CONCLUSIONS

1) Pour former le contenu de l'organe mémoire, il est indispensable de tenir compte des différentes formes de la mémoire dans la machine et de la place du programme en cours dans le schéma général de la résolution du problème.

2) Pour une utilisation réitérée du programme au cours du calcul, il est indispensable de prévoir le rétablissement du contenu de l'organe mémoire, les instructions variables incluses.

3) Le rétablissement du contenu de l'organe mémoire peut être réalisé de deux façons :

a) Transfert du contenu initial de l'organe mémoire d'une espèce de mémoire à l'autre.

b) Rétablissement du contenu de la mémoire à l'aide d'opérations élémentaires (remise à zéro des mémoires, addition, soustraction, etc.).

4) Lorsqu'on utilise les sous-programmes, on doit mettre le contenu de l'organe mémoire dont on a besoin en correspondance avec le programme principal.

5) Quand on forme le contenu de l'organe mémoire, il est souhaitable d'utiliser au maximum la mémoire passive, si elle existe.

6) Pour diviser le programme en parties, qui se trouvent dans la mémoire externe, il est indispensable de coordonner convenablement le transfert de la commande d'un groupe d'instructions à l'autre.

EXERCICES

1. Analyser la formation du contenu de l'organe mémoire pour les programmes :

a) Division (§ 2, programme 8).
b) Calcul de tg x (§ 3, programme 13).
c) Calcul d'un polynôme (§ 4, programmes 21, 22_1, 22).

7. OPÉRATIONS DE GROUPE

Dans la résolution de problèmes sur C. A. N., le contrôle a une grande importance, et en particulier celui de l'exactitude des programmes. Si une erreur dans le code d'un nombre détermine une inexactitude dans la résolution de certaines variantes du problème posé, l'erreur dans l'une des positions du code d'une instruction peut causer la détérioration de tout le programme. D'autre part, l'expérience du travail sur C. A. N. montre que des programmes même soigneusement vérifiés peuvent, en principe, contenir des fautes non dévoilées au cours de leur établissement, de leur codification et de leur entrée en mémoire, fautes découvertes seulement au moment où l'on fait des calculs de contrôle spéciaux.

Si, pour résoudre le même problème selon une méthode choisie, on peut proposer deux programmes différents, du point de vue commodité d'entrée et rapidité de mise au point, la préférence doit être accordée au programme qui occupe le plus petit nombre de cellules, c'est-à-dire contient l'information du problème sous la forme la plus condensée. On doit préférer ce programme, même lorsque les calculs qui y sont exécutés font perdre beaucoup de temps. D'autre part, un programme qui utilise un grand nombre de cellules peut être préférable, si ses instructions ne changent pas au cours du travail : ce qui est plus pratique du point de vue entrée et contrôle.

Il s'ensuit que l'on doit faire très attention pour choisir la suite des opérations élémentaires à effectuer en machine et leur codification rationnelle sous forme d'instructions.

Les exemples de programmes cycliques dépendant de paramètres nous ont montré qu'on peut arriver à économiser des instructions du programme, si l'on inclut dans l'ensemble des opérations élémentaires l'opération de substitution d'adresse et si l'on réglemente convenablement la répartition des grandeurs dans l'organe mémoire.

Dans le cas général, les processus cycliques sont, en principe, complexes (réitérés) et contiennent des paramètres qui peuvent déterminer une série de grandeurs rencontrées dans le cycle. Ce qui fait que le nombre total d'instructions n'ayant pas un caractère numérique s'accroît considérablement. Les moyens que donne la programmation de réduire parfois ce genre de pro-

gramme ont été indiqués au paragraphe 3. On peut aller plus loin et poser la question de la réduction des programmes cycliques, aux dépens non plus de la réussite du programme, mais de l'introduction en machine d'opérations spéciales, appelées *opérations de groupe*, moyennant des complications dans son schéma.

Remarquons ici que l'introduction des opérations de groupe donne, en principe, des possibilités plus larges pour construire des programmes invariables.

Puisqu'on peut construire des programmes de processus cycliques paramétriques de sorte que toutes les adresses variables, lors du passage d'un cycle à l'autre, varient d'une même quantité p (p étant le pas de substitution d'adresse), on pense naturellement à réaliser des changements d'adresses à l'extérieur de l'organe de mémoire, en insérant un dispositif approprié dans l'organe de commande.

Appelons le paramètre, qui caractérise le numéro du cycle accompli, index (ou constante) de modification.

Les informations indispensables pour le cycle qui dépend d'un paramètre sont :

1) La liste des opérations (instructions) du cycle sous leur forme initiale.
2) L'indication des adresses variables (symboles de la variabilité des adresses).
3) L'information sur l'index de modification (autrement dit de variabilité).

Les quatre grandeurs suivantes :

a) α, valeur initiale de l'index,
b) p, pas de variabilité (*) de l'index,
c) α_k, valeur finale de l'index,
d) α_t, valeur courante de l'index,

composent l'information sur l'index de modification.

La grandeur α_k, valeur finale de l'index de modification, sert à indiquer la fin du processus cyclique ; la grandeur α_t, valeur courante de l'index, est indispensable pour réaliser les processus cycliques réitérés, dépendant de paramètres externes et internes.

Les adresses variables $a_{(t)}$ doivent se former selon la loi :

$$\left.\begin{aligned} \alpha_t &= \alpha_{t-1} + p \\ a_{(t)} &= a_{(0)} + \alpha_t \end{aligned}\right\}; \tag{38}$$

où a_0 est le contenu initial de l'adresse variable.

En règle générale, on donne l'information 1 à l'aide des instructions de calcul correspondantes. Pour choisir les adresses variables, c'est-à-dire pour

(*) Il n'est pas exclu que le pas p soit à son tour une grandeur variable dépendant du numéro d'un cycle, par exemple, calculée selon une règle à l'intérieur du cycle.

savoir lorsqu'il faut former une adresse conformément à (38), il convient de choisir dans chaque adresse une position spéciale, le code « unité », dans lequel on aura comme symbole de dépendance de l'adresse par rapport au paramètre « le symbole d'opération de groupe » (l'adresse ne subit de changement que s'il y a un symbole d'opération de groupe). La position indiquée est complémentaire des positions dans lesquelles sont codées les adresses des cellules de l'organe mémoire.

Pour modifier les adresses variables conformément à (38) (si le symbole d'opération de groupe s'y trouve), il faut prévoir dans les machines des organes spéciaux. Cela peut, par exemple, être réalisé de la manière suivante :

Pour conserver la valeur courante α_t de l'index de modification, on prévoit un registre spécial de modification d'adresse.

L'opération

$$a_{(t)} = a_{(0)} + \alpha_t$$

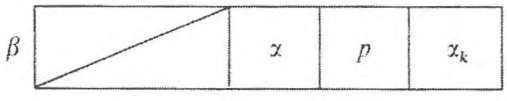
peut s'effectuer, soit sur l'organe arithmétique principal, soit, en raison de la simplicité de cette opération (tous les codes ont la dimension d'une adresse) sur un sommateur d'adresses spécial. On peut y réaliser l'opération de formation de la valeur suivante de l'index de modification :

$$\alpha_t = \alpha_{t-1} + p \,,$$

avec envoi du résultat obtenu au registre de modification.

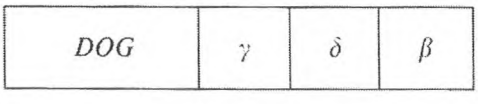
Pour conserver α_k, valeur finale de l'index de modification, un registre de cycles spécial est prévu. En outre, dans le circuit de la machine, on inclut un organe de coïncidence à l'aide duquel, pour savoir où doit s'arrêter l'exécution du cycle, on compare les grandeurs α_t et α_k par coïncidence (contenu du registre de modification des adresses et du registre des cycles).

Plaçons les grandeurs α, p, α_k, qui caractérisent le paramètre, dans les adresses d'une seule mémoire



Complétons l'ensemble des opérations élémentaires par les opérations suivantes :

1) opération *DOG*, début d'une opération de groupe



2) opération *FOG*, fin d'une opération de groupe

FOG	β	k_1	k_2

Suivant l'opération *DOG*, on exécute ce qui suit :

a) Le contenu du registre de modification d'adresses, conservant la valeur courante de l'index de modification du cycle précédent, est transféré dans la mémoire γ (dans l'une de ses adresses, par exemple, la première). Si le cycle externe (par rapport à ce qui est donné) ne dépend pas d'un paramètre, à la place de γ dans la première adresse de l'instruction *DOG* on met zéro ; mais s'il dépend d'un paramètre, la mémoire γ conservera l'information nécessaire qui sera utilisée de manière analogue lors d'un second appel au cycle externe.

b) Le contenu de la mémoire δ (de sa première adresse) est envoyé au registre de modification des adresses (A_1 δ conserve la grandeur α_t).

c) Le contenu de l'adresse A_3 de la mémoire β est envoyé au registre des cycles (valeur finale de l'index α_k).

Suivant l'opération *FOG* :

a) Addition au contenu du registre de modification d'adresses (α_{t-1}) du contenu de la deuxième adresse de la cellule β, pas de substitution d'adresse de l'index :

$$\alpha_t = \alpha_{t-1} + p \, .$$

b) Comparaison du contenu du registre des cycles et du registre de modification d'adresses ; s'il y a coïncidence, transfert de la commande à l'instruction dont l'adresse k_1 est indiquée dans l'adresse A_2 de l'instruction *FOG* ; sinon à l'instruction k_2 dont l'adresse est indiquée en A_3 de l'instruction *FOG*.

Toutes les adresses variables sont marquées du symbole de l'opération de groupe, c'est-à-dire que dans la position affectée à un symbole, on codifie l'unité.

Quand on exécute des instructions dont les adresses ont le symbole de l'opération de groupe, ces dernières sont modifiées, c'est-à-dire augmentent de la grandeur contenue dans le registre de modification d'adresses (la forme des instructions en mémoire reste en outre invariable).

Ainsi lorsqu'il y a des opérations de groupe de ce type, les programmes deviennent plus compacts et peuvent être entièrement invariables. Prenons des exemples.

***Exemple* 1.** *Calculer la somme* :

$$s = \sum_{j=1}^{m} \sum_{i=1}^{n} \frac{a_i x_j + b_i y_j}{c_i x_j + d_i y_j} .$$

Répartissons les données, dépendant des indices i et j dans les suites de mémoires correspondantes. Le calcul de la somme s peut alors être ramené à un processus cyclique binaire. Il est possible, j étant fixé, de faire le calcul de chacun des termes de la somme interne d'après un programme cyclique (cycle interne) : au i-ième cycle, on calcule le i-ième membre de la somme et on l'ajoute à la somme des termes précédents ; le programme du cycle interne est complété par les adresses de substitution d'adresse nécessaires. Les grandeurs x_j et y_j dépendant de j, seront envoyées dans les mémoires ω_1 et ω_2 avant l'appel au cycle interne. La construction ultérieure du programme n'a pas besoin d'être expliquée.

Dans l'exemple donné, avant chaque appel à un programme de cycle interne (instructions $N + 5 \ldots N + 17$), on a besoin d'établir les instructions dépendant du paramètre i (instructions $N + 5$, $N + 6$, $N + 8$, $N + 9$) : les suites de cellules avec des numéros qui dépendent du numéro du cycle du paramètre i, utilisées dans le cycle interne, sont fixées comme suit :

$$\alpha + 1, \ldots, \alpha + n ;$$
$$\beta + 1, \ldots, \beta + n ;$$
$$\gamma + 1, \ldots, \gamma + n ;$$
$$\delta + 1, \ldots, \delta + n ;$$

en conséquence, le programme est complété par les instructions de rétablissement $N + 18 \ldots N + 21$.

Programme 28.

Nombres

$\alpha + 1$	a_1
$\alpha + 2$	a_2
⋮	
$\alpha + n$	a_n
$\beta + 1$	b_1
$\beta + 2$	b_2
⋮	
$\beta + n$	b_n

Instructions

$N + 1$	+		$\tau + 1$	ω_1
$N + 2$	+		$\rho + 1$	ω_2
$N + 3$	$\oplus$	$N + 1$	Δ_1	$N + 1$
$N + 4$	$\oplus$	$N + 2$	Δ_1	$N + 2$
$N + 5$	$\times$	ω_1	$\alpha + 1$	ω_3
$N + 6$	$\times$	ω_2	$\beta + 1$	ω_4
$N + 7$	+	ω_3	ω_4	ω_3
$N + 8$	$\times$	ω_1	$\gamma + 1$	ω_4

Nombres

$\gamma + 1$	c_1
$\gamma + 2$	c_2
⋮	
$\gamma + n$	c_n
$\delta + 1$	d_1
$\delta + 2$	d_2
⋮	
$\delta + n$	d_n
$\tau + 1$	x_1
⋮	
$\tau + m$	x_m
$\rho + 1$	y_1
⋮	
$\rho + m$	y_m
ω_1	x_j
ω_2	y_i
ω	0 s

Δ_1			1	
Δ_2	+		$\tau + m$	ω_1
Δ_3			n	
Δ_4	×	ω_1	$\alpha + n$	ω_3

Instructions

$N + 9$	×	ω_2	$\delta + 1$	ω_5
$N + 10$	+	ω_4	ω_5	ω_4
$N + 11$	:	ω_3	ω_4	ω_3
$N + 12$	+	ω_3	ω	ω
$N + 13$	⊕	$N + 5$	Δ_1	$N + 5$
$N + 14$	⊕	$N + 6$	Δ_1	$N + 6$
$N + 15$	⊕	$N + 8$	Δ_1	$N + 8$
$N + 16$	⊕	$N + 9$	Δ_1	$N + 9$
$N + 17$	⩽	$N + 5$	Δ_4	$N + 5$
$N + 18$	⊖	$N + 5$	Δ_3	$N + 5$
$N + 19$	⊖	$N + 6$	Δ_3	$N + 6$
$N + 20$	⊖	$N + 8$	Δ_3	$N + 8$
$N + 21$	⊖	$N + 9$	Δ_3	$N + 9$
$N + 22$	⩽	$N + 1$	Δ_2	$N + 1$
$N + 23$	*ARR*			

En utilisant les opérations de groupe, nous obtenons le programme

Programme 29.

Nombres

$\alpha + 1$	a_1	
⋮		
$\alpha + n$	a_n	
$\beta + 1$	b_1	
⋮		
$\beta + n$	b_n	
$\gamma + 1$	c_1	
⋮		
$\gamma + n$	c_n	
$\delta + 1$	d_1	
⋮		
$\delta + n$	d_n	
$\tau + 1$	x_1	
⋮		
$\tau + m$	x_m	
$\rho + 1$	y_1	
⋮		
$\rho + m$	y_m	
ω_1		x_j
ω_2		y_j
S	0	

Instructions

$N + 1$	*DOG*	$-$	Δ_2	Δ_2
$N + 2$	$+$	$-$	$1/\tau+1$	ω_1
$N + 3$	$+$	$-$	$1/\rho+1$	ω_2
$N + 4$	*DOG*	Δ_2	$N+15$	$N + 5$
$N + 5$	*DOG*	Δ_2	Δ_1	Δ_1
$N + 6$	$\times$	$1/\alpha+1$	ω_1	ω_3
$N + 7$	$\times$	$1/\beta+1$	ω_2	ω_4
$N + 8$	$+$	ω_3	ω_4	ω_3
$N + 9$	$\times$	ω_1	$1/\gamma+1$	ω_4
$N + 10$	$\times$	ω_2	$1/\delta+1$	ω_5
$N + 11$	$+$	ω_4	ω_5	ω_4
$N + 12$	:	ω_3	ω_4	ω_3
$N + 13$	$+$	ω_3	S	S
$N + 14$	*DOG*	Δ_1	$N + 1$	$N + 6$
$N + 15$	*ARR*			

	Nombres			
Δ_1	–	–	1	n
Δ_2	–	–	1	m

Ici, et par la suite, le symbole opération de groupe dans l'adresse est indiqué par un « un » avant un trait vertical.

Suivons le travail du programme. D'après l'instruction $N + 1$, on envoie dans le registre de modification d'adresses le contenu de la première adresse de la mémoire Δ_2 qui contenait initialement dans cette adresse zéro. En outre, dans le registre des cycles on envoie le contenu de la troisième adresse de la cellule Δ_2 : la grandeur m.

Puisque le registre de modification d'adresses contient 0 lorsqu'on remplit pour la première fois les adresses $N + 2$, $N + 3$, ces instructions seront exécutées sous la forme sous laquelle elles sont notées dans le programme, sans tenir compte des symboles d'opération de groupe dans les deuxièmes adresses de ces instructions.

Selon l'instruction $N + 4$, on ajoutera au contenu du registre de modification d'adresses le contenu de la seconde adresse de la cellule Δ_2 (le pas de substitution d'adresse, égal à l'unité), c'est-à-dire que le registre de modification d'adresses contiendra déjà l'unité.

Puisque le contenu du registre de modification et celui des cycles ne coïncident pas, l'instruction $N + 4$ cette fois-ci transférera la commande à l'instruction $N + 5$.

Selon l'instruction $N + 5$, le contenu du registre de modification (l'unité) sera envoyé dans la première adresse de la cellule Δ_2, les contenus de la première et de la troisième adresses de la cellule Δ_1 (zéro et le nombre n) seront envoyés respectivement dans le registre de modification et celui des cycles. De plus, les instructions $N + 6$ ... $N + 13$ seront de nouveau accomplies sous la forme sous laquelle elles ont été écrites dans le programme.

Selon l'instruction $N + 14$, le contenu du registre de modification augmente d'une unité (du contenu de la deuxième adresse de la cellule Δ_1) et, puisque ce registre et le registre des cycles contiennent des codes différents, on enverra la commande à l'instruction $N + 6$.

Lorsqu'on exécutera ensuite les instructions $N + 6$ ... $N + 13$, il y aura une modification dans les adresses qui ont le symbole d'opération de groupe (la première adresse des instructions $N + 6$, $N + 7$ et la seconde adresse des instructions $N + 9$, $N + 10$), à savoir que ces adresses seront augmentées du contenu du registre de modification, contenu, qui est, dans ce cas, égal à l'unité.

L'instruction $N + 14$ augmente de nouveau le contenu du registre de modification d'une unité et transfère la commande à l'instruction $N + 6$. Ensuite, les adresses des instructions variables seront augmentées de 2 et ainsi de suite

jusqu'à ce que, selon l'instruction $N + 14$, à la suite de l'addition d'une unité (du contenu de la deuxième adresse de la cellule Δ_1) apparaisse le nombre n dans le registre de modification. (Notons que cela signifie que dans le cycle précédent les adresses variables ont été augmentées du nombre $n - 1$, c'est-à-dire, par exemple, que la première adresse de l'instruction $N+6$ était égale à :

$$(\alpha + 1) + (n - 1) = \alpha + n ,$$

c'est-à-dire que le dernier cycle du programme est accompli.) Dès lors, les contenus des registres de modification et des cycles sont égaux et la commande passe à l'instruction $N + 1$.

Selon l'instruction $N + 1$ on enverra dans le registre de modification le contenu de la première adresse de la cellule Δ_2, égal maintenant à l'unité, et dans le registre des cycles, le nombre m (contenu de A_3 de la cellule Δ_2). Finalement les instructions $N + 2$ et $N + 3$ exécutent l'envoi de x_2 et y_2 (cellules $\tau + 2$ et $\rho + 2$) dans les cellules standards ω_1 et ω_2. Selon l'instruction $N + 4$, le contenu du registre de modification augmentera d'une unité (et devient égal à 2) et la commande sera transmise au cycle interne à l'instruction $N + 5$. Cela se poursuivra ainsi jusqu'à ce qu'en A_1 de la cellule Δ_2 (à la suite de l'exécution de l'instruction $N + 4$) on ait le nombre $m - 1$. Ce nombre, d'après l'instruction $N + 1$, sera envoyé au registre de modification, et après exécution des instructions $N + 2$ et $N + 3$ (les deuxièmes adresses étant égales à $\tau + m$ et $\rho + m$), l'instruction $N + 4$ augmentera le contenu du registre de modification d'une unité et transmettra la commande à l'instruction $N + 15$, qui est l'arrêt (puisque les registres de modification et de cycles contiendront des codes égaux.)

***Exemple* 2.** *Calculer le produit* :

$$\pi = \sum_{i=1}^{n} \frac{a_i x + b_i y}{c_i x + d_i y} \sum_{i=n+1}^{2n} \frac{a_i x + b_i y}{c_i x + d_i y} \cdots \sum_{i=(m-1)n+1}^{mn} \frac{a_i x + b_i y}{c_i x + d_i y} .$$

Comme dans l'exemple précédent, rangeons les données qui dépendent du paramètre i dans des suites de cellules. Le calcul se ramène aussi à un processus cyclique binaire : dans le cycle interne, on calcule le terme suivant du produit ; dans le cycle externe, on multiplie le résultat obtenu par le produit des facteurs précédents.

A la différence de l'exemple précédent, chaque fois que l'on fait appel aux instructions du cycle interne, elles ne se rétablissent pas : les suites de cellules, qui ont un numéro dépendant du numéro du cycle de paramètre i et qui sont utilisées dans le cycle interne, se suivent.

Programme 30.

Nombres

$\alpha + 1$	a_1
$\alpha + 2$	a_2
⋮	
$\alpha + n$	a_n
$\alpha + n + 1$	a_{n+1}
$\alpha + n + 2$	a_{n+2}
⋮	
$\alpha + 2n$	a_{2n}
$\alpha + 2n + 1$	a_{2n+1}
⋮	
$\alpha + 3n$	a_{3n}
⋮	
$\alpha + mn$	a_{mn}
$\beta + 1$	b_1
⋮	
$\beta + mn$	b_{mn}
$\gamma + 1$	c_1
⋮	
$\gamma + mn$	c_{mn}
$\delta + 1$	d_1
⋮	

Instructions

N	$+$	$-$	$-$	ω_1
$N + 1$	$\times$	$\alpha + 1$	α_1	ω_3
$N + 2$	$\times$	$\beta + 1$	α_2	ω_4
$N + 3$	$+$	ω_3	ω_4	ω_3
$N + 4$	$\times$	$\gamma + 1$	α_1	ω_4
$N + 5$	$\times$	$\delta + 1$	α_2	ω_5
$N + 6$	$+$	ω_4	ω_5	ω_4
$N + 7$	:	ω_3	ω_4	ω_3
$N + 8$	$+$	ω_3	ω_1	ω_1
$N + 9$	$\oplus$	$N + 1$	Δ_1	$N + 1$
$N + 10$	$\oplus$	$N + 2$	Δ_1	$N + 2$
$N + 11$	$\oplus$	$N + 4$	Δ_1	$N + 4$
$N + 12$	$\oplus$	$N + 5$	Δ_1	$N + 5$
$N + 13$	$\leqslant$	$N + 1$	Δ_2	$N + 1$
$N + 14$	$\times$	ω_2	ω_1	ω_2
$N + 15$	$\oplus$	Δ_2	Δ_3	Δ_2
$N + 16$	$\leqslant$	$N + 1$	Δ_4	N
$N + 17$	ARR			

Nombres

$\delta + mn$	d_{mn}	
ω_1	0	Σ
ω_2	1	Π
α_1	x	
α_2	y	

Δ_1		1		
Δ_2	×	$\alpha + n$	α_1	ω_3
Δ_3		n		
Δ_4	×	$\alpha + mn$	α_1	ω_3

Le programme de cet exemple sur des opérations de groupe peut être noté sous la forme :

Programme 30_1.

Nombres

Δ_1	–	–	1	n
Δ_2	–	n		n
Δ_3	–	mn		–

Instructions

$N + 1$	DOG	–	Δ_1	Δ_1
$N + 2$	+	–	–	ω_1
$N + 3$	×	$1/\alpha+1$	α_1	ω_3
$N + 4$	×	$1/\beta+1$	α_2	ω_4
$N + 5$	+	ω_3	ω_4	ω_3
$N + 6$	×	$1/\gamma+1$	α_1	ω_4
$N + 7$	×	$1/\delta+1$	α_2	ω_5
$N + 8$	+	ω_4	ω_5	ω_4
$N + 9$	:	ω_3	ω_4	ω_3
$N + 10$	+	ω_3	ω_1	ω_1

	Instructions			
$N + 11$	FOG	Δ_1	$N + 12$	$N + 2$
$N + 12$	$\times$	ω_2	ω_1	ω_2
$N + 13$	$+$	Δ_1	Δ_2	Δ_1
$N + 14$	$\leqslant$	Δ_1	Δ_3	$N + 1$
$N + 15$	ARR			

Nous recommandons au lecteur de faire un contrôle suivi des changements de contenu des registres de modification et de cycles au cours du travail du programme.

***Exemple* 3.** *Conversion des nombres de binaire en décimal.*

Dans le circuit de certaines machines, il n'y a pas de transformation des codes des nombres de binaire en décimal. Dans de tels cas, on inclut un sous-programme de conversion spécial dans la suite des sous-programmes standards de la machine, et avant la sortie des nombres on prévoit, dans les programmes, d'y faire obligatoirement appel.

Examinons, pour plus de précision, une machine à virgule fixe et à 40 positions significatives binaires.

Supposons qu'une cellule α soit celle d'entrée pour le sous-programme, c'est-à-dire pour la transformation du code binaire du nombre A en décimal. Le code binaire doit être placé dans la cellule α. Supposons qu'une cellule β soit celle de sortie, c'est-à-dire que dans la cellule β se forme le code transformé du nombre A.

Désignons par a_i les caractères décimaux correspondants du nombre A, c'est-à-dire :

$$A = \frac{a_1}{10^1} + \frac{a_2}{10^2} + \cdots .$$

Pour écrire chacun des nombres de a_i, il faut 4 positions binaires, c'est-à-dire que 40 positions de la cellule β permettent d'écrire 10 codes décimaux.

Le problème de la transformation du code binaire d'un nombre peut être considéré comme un problème de construction d'un code

$$B = \frac{a_1}{16^1} + \frac{a_2}{16^2} + \cdots + \frac{a_{10}}{16^{10}},$$

où chacune des 4 positions binaires permet d'écrire le code décimal correspondant.

Le programme peut être représenté sous la forme :

Programme 31.

Nombres

α_1	$\frac{10}{16}$			
α	A			
β		B		
Δ	−	−	1	10

Instructions

N	DOG	−	−	Δ
$N+1$	+	−	−	β
$N+2$	×	α	α_1	α
$N+3$	$\rightarrow n$	36	α	r
$N+4$	$\leftarrow n$	4	β	β
$N+5$	+	r	β	β
$N+6$	$\leftarrow n$	36	r	r
$N+7$	−	α	r	α
$N+8$	$\leftarrow n$	4	α	α
$N+9$	FOG	Δ	$N+2$	k

déplacement de 36 positions vers la droite

déplacement de 4 positions vers la gauche

Ici, k est le numéro de l'instruction dont la commande est transmise après l'exécution du programme.

Par $\rightarrow n$ et $\leftarrow n$, on désigne le code de l'opération de déplacement à droite et à gauche du nombre de positions indiquées dans la première adresse de l'instruction correspondante.

Pour les processus cycliques dont on connaît le nombre de cycles et qui ne sont composés que d'une seule instruction, il convient d'avoir une opération de groupe spéciale *OG* I codifiée sous la forme de l'instruction

OG I	α	p	d_k

Les adresses de l'instruction *OG* I contiennent directement l'information sur le paramètre : dans la première adresse, la valeur initiale du paramètre de l'index de modification ; dans la deuxième adresse, le pas ; dans la troisième adresse, la valeur finale du paramètre.

Une fois l'instruction *OG* I exécutée, celle qui la suit est effectuée itérativement ; les adresses qui ont un symbole d'opération de groupe changent tant que le paramètre n'a pas atteint sa valeur finale. La commande est ensuite transmise à l'instruction suivante. La forme de l'instruction *OG* I dans la mémoire reste inchangée.

Ainsi, à l'aide de l'opération de groupe *OG* I, on peut exécuter les formes d'opérations suivantes :

$$1)\ [A] \odot a\,,$$

$$2)\ \ a \odot [A]\,,$$

$$3)\ [A] \odot [B]\,,$$

$$4)\ [A] \odot\,,$$

où $\odot$ est n'importe quelle opération des groupes I – III, V ; A et B sont des vecteurs à n dimensions ; a est un scalaire (qui peut être égal à zéro). Le cas 4) se rapporte à une instruction dans laquelle la deuxième adresse n'est pas utilisée.

Le symbole de l'opération de groupe est placé pour les opérations 1) et 4) en A_1 et A_3, pour l'opération du type 2) en A_2 et A_3, pour l'opération 3) dans toutes les adresses de l'instruction.

Pour certaines opérations relatives à un appel aux organes externes et habituellement employées en même temps pour les groupes de codes, on introduit des opérations de groupe spéciales, codifiées sous la forme d'instructions par des codes d'opérations spéciaux. Des exemples de ces opérations sont cités dans le chapitre II (opérations d'entrée, de sortie, d'appel à l'organe mémoire externe).

Pour compléter, mentionnons encore les opérations de groupes suivantes :

1) Conversion de n nombres $a + 1, a + 2, \ldots, a + n$ de décimal en binaire et leur enregistrement dans les cellules de l'organe mémoire $b + 1, b + 2, \ldots, b + n$

Code I	$a + 1$	n	$b + 1$

2) Conversion de n nombres $a + 1, a + 2, \ldots, a + n$ de binaire en décimal et enregistrement dans les cellules $b + 1, b + 2, \ldots, b + n$

Code II	$a + 1$	n	$b + 1$

3) Conversion de la zone « a » de l'organe mémoire externe de n nombres et leur enregistrement dans les cellules $b + 1, b + 2, \ldots, b + n$

Code III	a	n	$b + 1$

Le déplacement de groupe est le transfert des codes des cellules $a + 1, \ldots, a + n$ dans les cellules qui ont les numéros $b + 1, \ldots, b + n$

DG	$a + 1$	n	$b + 1$

Remarquons enfin que, pour exécuter l'opération de groupe *OG* I à l'aide d'instructions de substitution d'adresse, il faudrait déjà au moins quatre instructions : instruction de l'opération, instruction de substitution d'adresse de ses adresses variables, instruction effectuant le contrôle de fin de cycle et le transfert de commande correspondant. En outre, il faut aussi, en principe, des instructions pour le rétablissement des instructions variables, des constantes de rétablissement et de substitution d'adresse. D'autre part, la quantité des cycles effectués en machine est diminuée, c'est-à-dire que le temps pour les calculs est réduit aux dépens de la réduction de la quantité d'instructions qui exécutent les opérations sur les nombres donnés et des instructions auxiliaires.

Ainsi, les opérations de groupe, sous leurs différentes formes, simplifient la programmation, libèrent des cellules de l'organe mémoire et, par conséquent, réduisent le volume des informations introduites en machine et aussi, en principe, le temps d'exécution des opérations.

En ce qui concerne la programmation à l'aide des opérations de groupe de processus cycliques complexes dépendant de plusieurs paramètres, nous nous bornerons à la remarque ci-après.

Pour codifier « les symboles » de dépendance des adresses vis-à-vis des différents paramètres, on peut dans les adresses mettre à part plusieurs positions, et compliquer d'autant la commande par des opérations de groupe, de sorte que les adresses variables changent suivant la suite de symboles qui s'y trouvent. Cependant, une telle résolution du problème complique sérieusement l'organe de commande.

On peut se limiter à une seule position pour codifier le symbole de l'opération de groupe. En effet, provoquons la dissociation suivante des paramètres : supposons par exemple que nous ayons un processus cyclique binaire, le cycle externe dépendant du paramètre i, le cycle interne du paramètre j. Si dans le cycle interne il n'y a pas d'adresse dépendant du paramètre i, nous considérerons les paramètres comme déjà dissociés. En la présence de ces paramètres, nous agirons de la manière suivante : toutes les grandeurs entrant dans les adresses I et II du cycle interne et dépendant du paramètre i, seront envoyées dans les cellules standards, avant le programme du cycle interne. Les adresses III du cycle interne, dépendant du paramètre i, seront remplacées par les cellules standards correspondantes et, après le programme du cycle interne, seront transférées de ces cellules dans les cellules correspondantes (dépendantes de i).

EXERCICES

En utilisant les opérations de groupe, établir les programmes :

1. Calcul des valeurs d'un polynôme.

2. Calcul du produit scalaire de deux vecteurs.

3. Multiplication de la matrice par un vecteur.

4. Multiplication des matrices.

5. Résolution d'un système d'équations algébriques linéaires.

6. Résolution d'un système d'équations algébriques linéaires par la méthode d'élimination des inconnues.

7. Calcul des déterminants.

CHAPITRE IV

PROGRAMMATION OPÉRATORIELLE

L'analyse des programmes du chapitre III montre que, pour programmer des problèmes différents, on emploie des moyens standards que l'on peut formuler sous l'aspect de règles générales, non liées au contenu concret de chaque problème. D'autre part, ces règles doivent être formulées comme des indications pour programmer des fonctions algorithmiques déterminées sous forme de groupes d'instructions. Ces groupes d'instructions se divisent généralement en blocs séparés. La description des programmes par organigrammes permet de les examiner plus facilement et de programmer indépendamment chaque bloc isolé. Ainsi la classification des blocs de programmes selon leur destination algorithmique (fonctionnelle) dans le schéma général de la résolution du problème allège beaucoup le travail du programmeur.

La méthode de programmation fondée sur l'utilisation de la classification des parties des programmes d'après leur destination fonctionnelle a été proposée par A. A. Liapounov et a reçu le nom de méthode de programmation opératorielle.

Les fondements de cette méthode sont exposés dans ce chapitre.

1. SCHÉMAS DE CALCUL

Le matériel initial pour résoudre tout problème numérique est l'ensemble des données numériques initiales, que nous désignerons par le symbole $x_{\text{déb.}}$. Le but final d'un problème est d'obtenir les valeurs numériques des grandeurs qui intéressent le chercheur, ce que nous désignerons par $x_{\text{fin.}}$.

Le processus de résolution d'un problème numérique se présente comme un algorithme de remaniement des données initiales $x_{\text{déb.}}$ en données finales x_{fin}. Cet algorithme peut être représenté sous la forme d'une suite d'étapes de calcul plus ou moins indépendantes. A chaque étape, une succession déterminée d'opérations élémentaires effectue le remaniement des données numériques obtenues à l'étape précédente.

On a l'habitude d'appeler l'ensemble des opérations qui transforment les données numériques x en y, *opérateur de calcul*. Si l'on note **A** l'opérateur de

calcul, cette transformation s'écrira $x\mathbf{A} = y$. Les opérateurs seront dans la suite désignés par des lettres grasses.

Habituellement, l'opérateur de calcul est défini par des formules qui contiennent des données numériques x et des indications sur la façon dont doivent se succéder les opérations à effectuer sur ces données. En général, ces indications ne sont pas équivalentes, mais elles doivent garantir l'unicité du résultat. Le passage d'une étape à une autre peut être déterminé à l'avance, ou au cours du calcul par la vérification des différentes caractéristiques de l'information numérique. La vérification des caractéristiques de l'information est effectuée par le calcul des variables logiques. Les conditions qui fixent le choix de l'ordre des calculs s'appellent *les conditions logiques* et les opérateurs correspondants qui réalisent ce choix, *les opérateurs logiques*. La condition logique et l'opérateur qui lui correspond seront désignés respectivement par les lettres P et $\mathbf{P}$. On peut former le tableau des conditions logiques dans lequel il y aura les corrélations d'après lesquelles se calculent leurs valeurs. Dans le cas le plus simple, le contenu d'une condition logique s'écrit entre parenthèses, après sa désignation, par exemple : $P(a \leqslant b)$ ou $P(a \neq b)$, etc. Si une condition logique contrôlée est satisfaite, on estime sa valeur égale à l'unité ; dans le cas contraire on place, après l'opérateur logique, une flèche qui indique l'opérateur vers lequel se transmet la commande. Si la condition logique n'est pas satisfaite, on estime sa valeur égale à zéro, et la commande se transmet à l'opérateur qui suit l'opérateur logique.

Ainsi, l'algorithme de résolution d'un problème numérique est composé d'une succession d'opérateurs de calcul qui transforment l'information numérique, et d'opérateurs logiques qui effectuent le passage à l'application des opérateurs de calcul suivants, conformément à la valeur des conditions logiques.

La succession complète des opérateurs, qui définit tout le processus de transformation des problèmes numériques donnés $x_{\text{déb.}}$ en $x_{\text{fin.}}$, s'appelle le *schéma de calcul*. L'utilisation successive des opérateurs s'écrit sous la forme de leur produit :

$$\mathbf{A}_1 . \mathbf{A}_2 ..... \mathbf{A}_n = \prod_{k=1}^{n} \mathbf{A}_k .$$

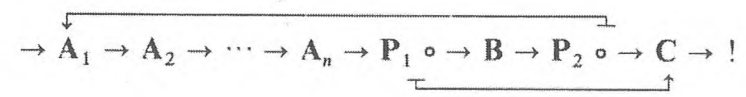
Pour comprendre le schéma de calcul, il faut savoir que l'opérateur agit sur l'information numérique à sa gauche et qu'ensuite, c'est à l'opérateur qui est situé à sa droite d'agir.

La valeur de la condition logique est déterminée par l'application de l'opérateur logique : si elle est égale à zéro, on applique l'opérateur d'ordre suivant ; si elle est égale à un, il y a passage à l'application de l'opérateur dont l'indication contient une flèche qui se trouve après l'opérateur logique. Par exemple, l'ordre d'action des opérateurs dans le schéma de calcul :

$$\prod_{k=1}^{n} \mathbf{A}_k \, \mathbf{P}_1 \quad \overset{\mathbf{C}}{\uparrow} \quad \mathbf{BP}_2 \quad \overset{\mathbf{A}_1}{\uparrow} \quad \mathbf{C} \, !$$

est le suivant : on applique d'abord successivement les opérateurs de calcul $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$, ensuite le calcul se ramifie : si la condition logique $\mathbf{P}_1$ à laquelle correspond l'opérateur logique $\mathbf{P}_1$ est égale à zéro, on applique l'opérateur de calcul **B** et si la condition logique $\mathbf{P}_2$ à laquelle correspond l'opérateur logique $\mathbf{P}_2$ est égale à 1 on applique de nouveau l'opérateur $\mathbf{A}_1$, etc. ; mais si la condition logique $\mathbf{P}_1$ est égale à 1 ou si la condition logique $\mathbf{P}_2$ est égale à zéro, on applique l'opérateur de calcul **C** et le calcul s'arrête — ce qui est indiqué par un point d'exclamation !

On peut renoncer à une notation linéaire du schéma et indiquer le passage d'un opérateur à l'autre par des flèches : on part de l'opérateur de calcul par une seule flèche, des opérateurs logiques par deux flèches, l'une avec le signe 1 ($1 \rightarrow$), l'autre avec le signe 0 ($0 \rightarrow$) conformément à la valeur de la condition logique. Le schéma de calcul, représenté ci-dessus sous l'aspect d'un graphique, a la forme :



Pour faciliter l'orientation dans le schéma de calcul, il faut mettre, avant l'opérateur auquel on transmet une commande issue de l'opérateur logique, une flèche avec le numéro de l'opérateur logique correspondant qui est à l'origine du transfert de commande : par exemple, le schéma de calcul que nous avons ci-dessus peut être représenté sous la forme :

$$\begin{array}{ccccccccc} 2 & & & & \mathbf{C} & & \mathbf{A}_1 & 1 & \\ \downarrow & \prod_{k=1}^{n} \mathbf{A}_k \mathbf{P}_1 & \uparrow & \mathbf{B}\mathbf{P}_2 & \uparrow & \downarrow & \mathbf{C}\,! & & \end{array}$$

La construction du schéma de calcul s'effectue après avoir choisi une méthode déterminée pour résoudre le problème. La division du calcul en opérateurs de calcul n'est en général pas unique. La pratique pour construire rationnellement les circuits de calcul s'acquiert au cours du travail.

Dans le chapitre III, la construction des programmes se fait après l'analyse du problème et la construction du schéma de calcul. En outre, la forme du programme dépend du choix de l'ordre d'exécution des opérations et du degré de précision de cet ordre à l'intérieur même du schéma de calcul.

Dans l'exemple 1 du calcul d'une fonction homographique

$$y = \frac{ax + b}{cx + d} \qquad \text{(chap. III, § 1)},$$

cette formule elle-même peut servir de schéma de calcul. Cependant, le schéma de calcul peut être précisé en établissant un ordre pour exécuter les opérations : ce qui est fait lorsqu'on traite cet exemple (cf. les formules de (2)).

Dans l'exemple 3 du calcul des valeurs d'un polynôme à une variable $f(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_n$ (chap. III, § 1), on a les deux schémas de calcul (6) et (8). Comme le montre la comparaison de ces schémas, le calcul des valeurs d'un polynôme à une variable d'après l'algorithme de Hörner (8) abrège considérablement le nombre des opérations nécessaires. L'analyse que nous allons faire permet de construire le schéma de calcul suivant. Introduisons les opérateurs :

$$\begin{aligned}
&\mathbf{B}_1 : A_1 = a_0 x, \qquad A_2 = A_1 + a_1, \\
&\mathbf{B}_2 : A_3 = A_2 x, \qquad A_4 = A_3 + a_2, \\
&\vdots \\
&\mathbf{B}_n : A_{2n-1} = A_{2n-2} x, \qquad A_{2n} = A_{2n-1} + a_n = f(x).
\end{aligned}$$

Le schéma de calcul (8) sous la forme opérationnelle a alors l'aspect

$$\prod_{k=1}^{n} \mathbf{B}_k .$$

Les opérateurs $\mathbf{B}_k$ sont définis par les formules :

$$A_{2k} = A_{2k-2} x + a_k, \qquad k = 1, 2, \ldots, n .$$

Ces formules sont du même type : ce qui donne la possibilité de construire un programme pour calculer les valeurs d'un polynôme à une variable qui soit rationnel (programme de l'exemple 2, § 4, chap. III). Pour garantir la rationalité du programme, il faut compléter le schéma de calcul par des opérateurs qui tiennent compte du caractère spécifique aussi bien du schéma que de la C. A. N. en question.

2. LES SCHÉMAS DE PROGRAMME

Un schéma de calcul permet d'effectuer un problème suivant un algorithme choisi. Cependant, pour que les calculs se fassent automatiquement, c'est-à-dire pour les exécuter sur C. A. N., il ne suffit pas de construire un schéma de calcul. Pour programmer un problème sur C. A. N., on doit tenir compte des possibilités et des particularités de la commande automatique des calculs sur C. A. N. Dans la mesure où les numéros des cellules de l'organe mémoire contenant les nombres sur lesquels doivent porter les opérations, sont codifiés dans les instructions effectuant les opérations élémentaires, une seule instruction peut effectuer les opérations élémentaires sur des nombres différents. Par exemple, dans le programme 9 du chapitre III, la $(k + 2)$-ième instruction calcule successivement les nombres de la suite naturelle.

Rappelons que les instructions elle-mêmes sont en C. A. N. des nombres sur lesquels il est aussi possible de faire des opérations ; c'est pourquoi, si on les change de façon appropriée et si on applique la réitération, on peut résoudre de nouveaux problèmes, exécuter de nouvelles opérations sur les mêmes nombres, ou accomplir les mêmes opérations sur des nombres contenus dans les autres cellules de l'organe mémoire. Par exemple, dans le programme 19, paragraphe 4, la $(N + 2)$-ième instruction variable utilise pour les calculs, successivement les nombres contenus dans les cellules $\alpha + 1$, $\alpha + 2, \ldots, \alpha + n$.

Pour programmer des problèmes sur C. A. N., il faut aussi essayer d'utiliser le plus rationnellement possible l'organe mémoire. A l'aide d'un nombre restreint d'instructions, on doit pouvoir faire un grand nombre d'opérations. Enfin, il faut prévoir une entrée et une sortie des données, une conversion des nombres d'un système numérique dans un autre, le contrôle des calculs, etc. Pour programmer un problème numérique sur C. A. N., le schéma de calcul doit être exécuté par des opérateurs spéciaux dits *opérateurs de commande*. Les opérateurs de commande préparent le contenu de l'organe mémoire et assurent la gestion la plus rationnelle du processus numérique sur le C. A. N.

Le schéma de calcul exécuté par les opérateurs de commande qui permettent de représenter un algorithme sous la forme d'un programme (ensemble d'instructions), s'appelle *schéma de programme* ou *schéma de travail de la machine*.

Dans un schéma de programme, chaque opérateur représente un groupe déterminé d'instructions du programme. Pour simplifier le schéma du programme, il est à souhaiter que chaque opérateur contienne le nombre maximal d'instructions. En même temps, l'ensemble des instructions de chaque opérateur indépendant a une destination fonctionnelle unique, d'après laquelle on divise les opérateurs en opérateurs logiques et opérateurs de commande. De même, on peut distinguer quelques types d'opérateurs de commande :

1) les opérateurs de substitution d'adresse,
2) les opérateurs de mise en forme,
3) les opérateurs de rétablissement,
4) les opérateurs d'envoi,
5) les opérateurs de transfert,
6) les opérateurs de circulation.

Au fur et à mesure que l'on acquiert de l'expérience en matière de programmation, on peut améliorer la classification des opérateurs.

Les types d'opérateurs énumérés permettent de construire des programmes très dissemblables.

La méthode de programmation opératorielle est constituée de la manière suivante. On construit d'abord le schéma de calcul du problème : il contient les opérateurs de calcul et les conditions logiques (l'information sur les opérateurs de calcul se présente sous la forme qui assure le plus rationnellement

possible l'établissement du programme). On introduit ensuite dans le schéma de calcul les opérateurs de commande qui permettent la réalisation du schéma de calcul sur C. A. N. Enfin, on fait la description de la structure du schéma du programme, c'est-à-dire que l'on donne l'information sur chaque opérateur du schéma. D'après l'information donnée, la programmation des opérateurs est faite en utilisant des règles bien déterminées. Mentionnons les avantages attachés à la méthode de programmation opératorielle.

Elle permet de construire par morceaux des programmes pour des problèmes complexes. De plus, la programmation des différentes parties du programme peut être faite après une interruption, sans connaissance préalable de tout le problème, et même par différentes personnes. L'utilisation de la méthode de programmation opératorielle facilite la mise au point du programme en C. A. N., l'introduction des corrections dans le programme, et crée les conditions qui permettent d'automatiser le processus de programmation.

Examinons les programmes du chapitre III qui ont différents opérateurs de commande.

Dans l'exemple 3 du calcul de la table des carrés des entiers naturels avec mise en mémoire externe (chap. III, § 4), la $(k + 1)$-ième instruction du programme a sa troisième adresse variable : elle change d'une unité d'un cycle à l'autre. Le changement de la troisième adresse de la $(k + 1)$-ième instruction du programme est accompli par la $(k + 3)$-ième instruction en ajoutant à la $(k + 1)$-ième instruction la constante contenue dans la cellule β, c'est-à-dire que la $(k + 3)$-ième instruction du programme est une instruction de substitution d'adresse avec une substitution d'adresse constante contenue dans la cellule β.

Dans l'exemple 4 (chap. III, § 4), du calcul des valeurs d'un polynôme, le programme contient aussi une instruction de substitution d'adresse (la $(N + 3)$-ième instruction). Chaque instruction de substitution d'adresse peut changer en même temps une ou plusieurs adresses d'une seule et même instruction. Pour changer les adresses de différentes instructions du programme, on a besoin de différentes instructions de substitution d'adresse. Aussi, dans l'exemple 5 (chap. III, § 4) du calcul de la somme

$$s = \sum_{i=1}^{n} \frac{x_i^3 + a}{x_i \sqrt{x_i}(x_i - a)}$$

le programme contient cinq instructions de substitution d'adresse, de la $(N + 10)$-ième à la $(N + 14)$-ième incluse.

Dans les exemples cités, les programmes des opérateurs de calcul contiennent des instructions variables qui changent lors du passage d'un cycle à un autre. Si l'opérateur contient des instructions qui changent selon une règle déterminée lorsqu'on l'applique de façon répétée, on dit que *l'opérateur dépend d'un paramètre*. Nous noterons la dépendance de l'opérateur **A** du paramètre k par un indice inférieur $\mathbf{A}_k$, et nous l'établirons à l'aide *du tableau*

spécial des variations des adresses en fonction d'un paramètre. Ce tableau contient les numéros des instructions variables, le pas de substitution d'adresse et les numéros des adresses dépendant d'un paramètre.

L'opérateur de commande effectuant la substitution d'adresse des instructions des opérateurs dépendant de paramètres en correspondance avec le tableau des variations des adresses en fonction d'un paramètre est appelé *opérateur de changement d'adresse* et est désigné par $\mathbf{F}(pk)$, où p est le pas de substitution d'adresse, c'est-à-dire le nombre d'unités sur lequel le paramètre est adressé dans chaque cycle. Le programme de l'opérateur de substitution d'adresse se construit selon le tableau des variations des adresses en fonction d'un paramètre.

Dans l'exemple 3 (chap. III, § 4) : calcul de la table des carrés des entiers naturels avec mise en mémoire, l'opérateur de substitution d'adresse $\mathbf{F}(n)$ est réalisé par la $(k+3)$-ième instruction de substitution d'adresse. L'information initiale pour construire le programme d'un opérateur de substitution d'adresse $\mathbf{F}(n)$ — tableau des variations des adresses des instructions en fonction d'un paramètre (en abrégé *TVP*) — n'a, dans cet exemple, qu'une seule ligne.

Tableau 6. — *Variations des adresses en fonction d'un paramètre.*

Numéro de l'instruction variant en fonction du paramètre	Variation de l'adresse 1 en fonction du paramètre	Variation de l'adresse 2 en fonction du paramètre	Variation de l'adresse 3 en fonction du paramètre
$k+1$	—	—	1

Dans l'exemple 4 (chap. III, § 4) : calcul des valeurs d'un polynôme, le *TVP* n'a aussi qu'une seule ligne

$N+2$	—	1	—

d'après laquelle on construit la constante de substitution d'adresse

β_1	—	—	1	—

et le programme de l'opérateur de substitution d'adresse $\mathbf{F}(k)$, c'est-à-dire l'instruction de substitution d'adresse

$N+3$	$\oplus$	$N+2$	β_1	$N+2$

Dans l'exemple 5 (chap. III, § 4) du calcul de la somme

$$s = \sum_{i=1}^{n} \frac{x_i^3 + a}{x_i \sqrt{x_i}(x_i - a)}$$

le tableau des variations des adresses en fonction du paramètre est le suivant :

TVP

$N + 1$		1	1	–
$N + 2$		1	–	–
$N + 4$		1	–	–
$N + 5$		1	–	–
$N + 6$		1	–	–

D'où l'on construit deux constantes de substitution d'adresse :

γ_1		1	1	
γ_2		1		

et le programme de l'opérateur de substitution d'adresse :

$N + 10$	$\oplus$	$N + 1$	γ_1	$N + 1$
$N + 11$	$\oplus$	$N + 2$	γ_2	$N + 2$
$N + 12$	$\oplus$	$N + 4$	γ_2	$N + 4$
$N + 13$	$\oplus$	$N + 5$	γ_2	$N + 5$
$N + 14$	$\oplus$	$N + 6$	γ_2	$N + 6$

Si l'on doit préparer le programme du problème en vue d'un emploi répété, il faut prévoir un rétablissement des instructions transformées au cours du travail.

L'opérateur de commande qui prépare le contenu initial de l'organe mémoire pour que le programme soit répété s'appelle *opérateur de formation* et est désigné par la lettre Φ. L'information sur les opérateurs de formation doit contenir les numéros des cellules variables de l'organe mémoire qui demandent le rétablissement, et l'état initial de ces cellules.

Dans l'exemple 4 (chap. III, § 4) du calcul des valeurs d'un polynôme il faut prévoir, pour que le programme puisse être répété, un rétablissement à l'état initial de l'instruction variable $N + 2$

$N + 2$	$+$	ω	$\alpha + 1$	ω

ce qui, dans le programme 22, est réalisé par l'instruction

$N + 5$	$\oplus$		α	$N + 2$

où α est le numéro de la cellule de l'organe mémoire contenant l'état initial de la $(N + 2)$-ième instruction.

Dans l'exemple cité, l'aspect final de la $(N + 2)$-ième instruction est connu d'avance :

$N + 2$	$+$	ω	$\alpha + n + 1$	ω

c'est pourquoi le rétablissement de son état initial peut aussi être réalisé à l'aide de l'instruction de substitution d'adresse

$N + 5$	$\ominus$	$N + 2$	β_2	$N + 2$

avec la substitution constante d'adresse

β_2			n	

Ainsi, l'opérateur de formation Φ dans cet exemple est réalisé par l'instruction $N + 5$.

Dans l'exemple 5 (chap. III, § 4) l'opérateur de formation Φ doit rétablir l'état initial des instructions $N + 1$, $N + 2$, $N + 4$, $N + 5$ et $N + 6$.

$N + 1$	$\times$	$\alpha + 1$	$\alpha + 1$	ω_1
$N + 2$	$\times$	$\alpha + 1$	ω_1	ω_1
$N + 4$	$\sqrt{\ }$	$\alpha + 1$		ω_2
$N + 5$	$\times$	$\alpha + 1$	ω_2	ω_2
$N + 6$	$-$	$\alpha + 1$	β	ω_2

Plaçons l'état initial de ces instructions respectivement dans les cellules δ_1, δ_2, δ_3, δ_4 et δ_5. L'opérateur de rétablissement est alors réalisé par le groupe de commandes

$\Phi + 1$	$\oplus$		δ_1	$N + 1$
$\Phi + 2$	$\oplus$		δ_2	$N + 2$
$\Phi + 3$	$\oplus$		δ_3	$N + 4$
$\Phi + 4$	$\oplus$		δ_4	$N + 5$
$\Phi + 5$	$\oplus$		δ_5	$N + 6$

L'opérateur de calcul dépendant d'un paramètre peut devenir indépendant après l'introduction d'un opérateur spécial d'envoi des données variables dans des cellules standards.

L'opérateur d'envoi est un opérateur de commande réalisant le transfert successif des nombres dans les cellules standards de l'organe mémoire. Désignons-le par la lettre **Z**. L'information sur l'opérateur **Z** est formée d'un couple de numéros de cellules de l'organe mémoire. Le premier nombre du couple détermine la cellule à partir de laquelle se fait l'envoi, le second, la cellule vers laquelle l'envoi des nombres est exécuté.

Dans l'exemple 4 (chap. III, § 4), le programme 22_2 contient l'opérateur d'envoi $\mathbf{Z}\ \{\ \alpha + 1 \Rightarrow \delta\ \}$ réalisé par l'instruction

N	$+$		$\alpha + 1$	δ

Dans l'exemple 5 (chap. III, § 4), l'introduction de l'opérateur d'envoi $\mathbf{Z}\{\alpha + 1 \Rightarrow \delta\}$ est réalisée par l'instruction

N	+		$\alpha + 1$	δ

ce qui améliore beaucoup le programme du problème (cf. programmes 23 et 23_1).

L'opération d'envoi qui ne dépend pas d'un paramètre s'appelle *opérateur de transfert.*

Dans l'exemple 2 (chap. III, § 3) de l'extraction de racine $y = \sqrt{x}$, le programme contient un opérateur de transfert $\mathbf{Z}\{\omega_1 \Rightarrow \alpha_1\}$ réalisé par l'instruction

$k + 5$	+	ω_1		α_1

qui prépare le contenu de l'organe mémoire pour le cycle itératif suivant.

Une forme d'envoi spéciale est *la circulation* de la succession des grandeurs dans les cellules de l'organe mémoire.

L'opérateur de circulation produit la permutation (la circulation) des données dans les cellules de l'organe mémoire. La succession des numéros des cellules de l'organe mémoire, dans lesquelles se produisent la circulation des nombres et l'ordre de permutation (substitution), sert d'information à l'opérateur de circulation.

Dans le programme de l'exemple 6 (chap. III, § 4) du calcul de l'intégrale $\int_0^1 y\,dx$ selon la formule de Simpson, la circulation des grandeurs y_{2i}, y_{2i+1}, y_{2i+2} dans les cellules $\omega + 1$, $\omega + 2$, $\omega + 3$ se fait à l'aide du groupe d'instructions

$k + 10$	+	$\omega + 2$		$\omega + 1$
$k + 11$	+	$\omega + 3$		$\omega + 2$
$k + 12$	+	$\omega + 4$		$\omega + 3$

qui déterminent aussi l'opérateur de circulation dans un programme donné.

Passons à l'analyse des schémas des programmes étudiés dans le chapitre précédent.

Examinons l'exemple 1 (chap. III, § 3) du calcul et de l'impression de la table des carrés des termes d'une progression arithmétique. Dans cet exemple, le schéma de calcul sous la forme opératorielle a l'aspect $\prod_{k=0}^{N} \mathbf{A}_k$, où les opérateurs de calcul $\mathbf{A}_k$ sont déterminés par la formule :

$$a_k^2 = b_k\,, \qquad a_k + c = a_{k+1} \qquad (k = 0, 1, 2, \ldots, N)\,,$$

de plus $a_0 = a$. Puisque les nombres a_k et b_k $(k = 0, 1, \ldots, N)$ ne sont pas mémorisés dans chaque cycle, l'indice k ne désigne que la différence de leur valeur dans chaque cycle, et non une place différente dans les cellules de l'organe mémoire.

Dans le cas où, dans la suite $a_0, a_1, \ldots, a_N$ les nombres diffèrent par leur grandeur, mais se placent dans une seule et même cellule de l'organe mémoire, l'indice qui désigne ces grandeurs doit être mis entre parenthèses $(a_{(k)})$.

Ainsi, les opérateurs de calcul $\mathbf{A}_k$, dans l'exemple montré, conformément à la distribution des cellules de l'organe mémoire, se définissent par la formule :

$$a_{(0)} = a\,; \quad a_{(k)}^2 = b_{(k)}\,, \quad a_{(k)} + c = a_{(k+1)} \qquad (k = 0, 1, \ldots, N)\,.$$

Il est alors évident que les opérateurs $\mathbf{A}_k$ se définissent par des formules identiques, et c'est pourquoi ils se réalisent en machine par des groupes d'instructions identiques.

Pour construire le schéma, il reste à introduire l'opérateur « $\mathbf{\Pi}$ » pour l'impression des nombres $b_{(k)}$, et l'opérateur $\mathbf{P}$ pour le transfert de la commande d'après la condition logique :

$$P = \begin{cases} 1\,, & a_{(k)} \leqslant a + cN\,, \\ 0\,, & a_{(k)} > a + cN\,. \end{cases}$$

Le schéma du programme de l'exemple 1 (chap. III, § 3) a la forme :

$$\begin{array}{ccccc} \mathbf{P} & & & & \mathbf{A} \\ \downarrow & \mathbf{A}_{(k)} & \mathbf{\Pi P} & \uparrow & ! \end{array}$$

C'est le schéma typique d'un programme de processus itératif et de calcul récurrent (cf. programmes 7, 8, 9, 10, 11 du chap. III).

Dans le paragraphe 4 du chapitre précédent, des exemples de programmes dont les opérateurs de calcul dépendent d'un paramètre ont été examinés.

Dans l'exemple 2 (chap. III, § 4), le schéma de calcul des valeurs d'un polynôme, comme nous l'avons établi (chap. IV, § 2), se présente sous la forme :

$$\prod_{k=1}^{n} \mathbf{B}_k\,;$$

les opérateurs $\mathbf{B}_k$ sont définis par les formules :

$$A_0 = a_0 \; ; \quad A_{(2k-2)}\, x + a_k = A_{(2k)} , \quad k = 1, 2, \ldots, n \; ; \quad A_{(2n)} = f(x) ,$$

qui ne se distinguent que par le nombre a_k, c'est-à-dire que les opérateurs $\mathbf{B}_k$ contiennent des grandeurs qui dépendent d'un paramètre.

Disposant les nombres $a_0, a_1, \ldots, a_n, x$ respectivement dans les cellules de l'organe mémoire $\omega, \alpha + 1, \ldots, \alpha + n, \alpha$, construisons les programmes des opérateurs $\mathbf{B}_k$ et notons la variation des adresses des instructions des programmes en fonction d'un paramètre :

	Instructions				Variation des adresses en fonction d'un paramètre		
$B + 1$	×	ω	α	ω			
$B + 2$	+	ω	$\alpha + k$	ω		1	

Introduisons l'opérateur de substitution d'adresse $\mathbf{F}(k)$ qui change d'une unité la seconde adresse de la $(B + 2)$-ième instruction, c'est-à-dire qui réalise le passage du programme de l'opérateur $\mathbf{B}_k$ au programme de l'opérateur $\mathbf{B}_{k+1}$; en répétant alors de façon cyclique les instructions de calcul $\mathbf{B}_k$ et en préparant ces instructions à accomplir le cycle suivant, on peut réaliser le calcul des valeurs d'un polynôme de degré n avec un petit nombre d'instructions. Pour diriger le processus de calcul cyclique, l'opérateur de transfert de commande $\mathbf{P}$ est introduit, à condition que la deuxième adresse de la $(B + 2)$-ième instruction contienne un code inférieur à $\alpha + n + 1$.

A cet effet, la constante de comparaison est représentée par le code :

β	+	ω	$\alpha + n$	ω

et se place dans la cellule β. La condition logique est déterminée par la relation

$$P = \begin{cases} 1 , & (B + 2) \leqslant (\beta) , \\ 0 , & (B + 2) > (\beta) . \end{cases}$$

Pour rétablir la variable de la $(B + 2)$-ième instruction du programme, on introduit un opérateur de formation $\mathbf{\Phi}$, en utilisant la forme initiale de la $(B + 2)$-ième instruction :

$B + 2$	+	ω	$\alpha + 1$	ω

Alors au schéma de calcul : $\prod_{k=1}^{n} \mathbf{B}_k$ correspondra le schéma de programme

$$\begin{array}{llll} \mathbf{P} & & & \mathbf{B} \\ \downarrow & \mathbf{B}_k\, \mathbf{F}(k)\, \mathbf{P} & \uparrow & \mathbf{\Phi}\, ! \end{array}$$

C'est le schéma typique d'un programme fait pour des processus cycliques qui dépendent d'un seul paramètre (cf. programmes 6, 7, 8, 9 du chap. III).

Les schémas des programmes que nous venons de voir sont les schémas fondamentaux des programmes à processus cycliques les plus simples. Des schémas analogues peuvent différer par la présence ou l'absence d'opérateurs de formation **Φ**, par l'emplacement dans le programme des opérateurs de commande et des opérateurs logiques. En principe, il y a un opérateur de formation dans chaque programme, il se trouve très souvent au début. La condition logique qui assure le processus cyclique peut se trouver avant l'opérateur de calcul de façon qu'en fin de programme se place l'instruction de transfert inconditionnel de la commande au début du programme. Désignons par P^0 la condition logique correspondant à l'instruction de transfert inconditionnel de la commande. Le schéma du programme 6 du chapitre III donne ceci :

$$\begin{array}{llllll} 2 & & 1 & & \mathbf{P}_1\ 1 & \\ \downarrow & \mathbf{P}_1 & \uparrow & \mathbf{AP}_2^0 & \uparrow\ \downarrow & ! \end{array}$$

Il convient de ne pas oublier que, lors de l'inversion des opérateurs dans le schéma du programme, les instructions qui leur correspondent peuvent changer (cf. programmes 6 et 6_1 du chap. III).

3. PROGRAMMATION DE PROCESSUS CYCLIQUES COMPLEXES

Les schémas de processus cycliques très simples ne suffisent pas pour de nombreux problèmes techniquement importants. Par conséquent, il faut étudier la façon de programmer les processus cycliques complexes. Dans ce paragraphe nous allons examiner des processus cycliques qui ont des schémas de programmes compliqués.

Les processus cycliques du chapitre précédent sont simples (« unaires ») : leur réalisation dépend d'une seule condition logique. Lorsqu'il y a plus d'une condition logique pour organiser les processus cycliques, le schéma du programme en est compliqué.

En combinant les schémas de programmes des processus cycliques qui sont simples, on peut obtenir les différentes formes des processus cycliques binaires.

Si les résultats des calculs d'un processus cyclique simple $\mathbf{AP}\ \overset{\mathbf{A}}{\uparrow}\ !$ ou

$\mathbf{A}_k\,\mathbf{F}(k)\,\mathbf{P}\ \overset{\mathbf{A}}{\uparrow}\ !$ sont des données initiales pour un nouveau processus cyclique avec l'opérateur de calcul **B**, le processus numérique devient un processus cyclique binaire ayant, par exemple, un schéma de programme de forme :

$$\mathbf{\Phi A P}_1\ \overset{\mathbf{A}}{\uparrow}\ \mathbf{B P}_2\ \overset{\mathbf{\Phi}}{\uparrow}\ !\qquad \text{ou}\qquad \mathbf{\Phi}(k)\,\mathbf{A}_k\,\mathbf{P}_1\ \overset{\mathbf{A}}{\uparrow}\ \mathbf{B P}_2\ \overset{\mathbf{\Phi}}{\uparrow}\ !$$

L'opérateur de calcul **B** peut aussi dépendre d'un paramètre. Le processus cyclique sera aussi binaire, si l'opérateur de calcul dépend à la fois de deux paramètres. Le schéma du programme aura alors, par exemple, la forme :

$$\mathbf{\Phi}_1\,\mathbf{\Phi}_2(k)\,\mathbf{A}_{ki}\,\mathbf{F}(k)\,\mathbf{P}_1\ \overset{\mathbf{A}}{\uparrow}\ \mathbf{F}(i)\,\mathbf{P}_2\ \overset{\mathbf{\Phi}_2}{\uparrow}\ !$$

Le nombre des cycles binaires selon k peut, en général, dépendre d'un second paramètre i.

Lorsqu'on programme des processus cycliques multiples, il est indispensable de surveiller l'exactitude de la préparation des données initiales et du programme lors du passage d'un cycle ayant un paramètre à un cycle ayant deux paramètres.

Il est évident qu'on peut ensuite compliquer le schéma des programmes à processus cycliques qui ont des cycles multiples.

1. Processus cyclique binaire sans paramètre.

1. *Exemple de solution numérique d'une équation différentielle ordinaire du premier degré* :

$$\frac{dy}{dx} = f(x, y)$$

sur le segment $[a, b]$ avec la condition initiale $y(a) = y_0$. On peut prendre le schéma numérique suivant : divisons le segment $[a, b]$ en intervalles de longueur h. Notons $y_h = y(x_n)$ et intégrons l'équation sur l'intervalle (x_{n-1}, x_n). Comptons la valeur de l'intégrale d'après la formule des trapèzes. Nous obtiendrons les formules suivantes :

$$\Delta y_n = \frac{h}{2}\left[f(x_{n-1}, y_{n-1}) + f(x_n, y_n)\right],$$

$$y_n = y_{n-1} + \Delta y_n .$$

Nous emploierons la méthode itérative pour faire les calculs d'après cette formule. Les données initiales sont $x_0 = a$, $y_0^{(0)} = y_0$; $y_1^{(0)} = y_0$,

$$y_n^{(k+1)} = y_{n-1} + \frac{h}{2}\left[f(x_{n-1}, y_{n-1}) + f(x_n, y_n^{(k)})\right].$$

Le processus itératif ne se termine que lorsque $| y_n^{(k)} - y_n^{(k-1)} | < \varepsilon$. Supposons que les valeurs numériques de la fonction $\frac{h}{2} f(x, y)$ se traitent d'après un sous-programme dont les données initiales sont placées respectivement dans les cellules de l'organe mémoire α_1 et α_2, et le résultat $\frac{h}{2} f(x, y)$ se détermine dans la cellule ω_1. Pour en abréger l'écriture, nous désignerons cette supposition sous la forme de l'instruction symbolique spéciale

$\frac{h}{2} f(\alpha_1, \alpha_2)$			ω_1

Prenons le schéma de calcul suivant. Les données initiales seront : $x_0 = a$, $y(0) = y_0$, $y_1^{(0)} = y_0$.

$$x_n = x_{n-1} + h ,$$

$$y_n^{(k+1)} = \left[y_{n-1} + \frac{h}{2} f(x_{n-1}, y_{n-1}) \right] + \frac{h}{2} f(x_n, y_n^{(k)}) ,$$

$$l_k = y_n^{(k+1)} - y_n^{(k)} .$$

Conformément à cela, l'opérateur de calcul **A** dans le cycle primaire est programmé d'après les formules :

$$y_n^{(k+1)} = \varphi_{n-1} + \frac{h}{2} f(x_n, y_n^{(k)}) ,$$

$$l_k = y_n^{(k+1)} - y_n^{(k)} .$$

L'opérateur de calcul **B** dans le cycle binaire se compose de :

$$\varphi_{n-1} = y_{n-1} + \frac{h}{2} f(x_{n-1}, y_{n-1}) ,$$

$$x_n = x_{n-1} + h .$$

L'opérateur de formation préparant les données initiales pour l'opérateur **A** a la forme :

$$y_n^{(0)} = y_{n-1} .$$

Le schéma du sous-programme est le suivant :

$$\begin{array}{cc} & \mathbf{A} \quad \mathbf{B} \\ \mathbf{B\Phi AP}_1 & \uparrow \mathbf{P}_2 \uparrow ! \end{array}$$

Les données initiales pour l'opérateur **B** sont les nombres x_{n-1}, y_{n-1}, h, que nous placerons respectivement dans les cellules de l'organe mémoire α_1, α_2, α_3.

L'opérateur **B** est réalisé par les instructions

$B+1$	$\frac{h}{2}f(\alpha_1, \alpha_2)$			ω_1
$B+2$	$+$	ω_1	α_2	α_4
$B+3$	$+$	α_1	α_3	α_1

Les données initiales pour l'opérateur **A**, les nombres x_n, $y_n^{(0)} = y_{n-1}$, φ_{n-1}, sont placées respectivement dans les cellules α_1, α_2, α_4. L'opérateur **A** est réalisé par les instructions

$A+1$	$\frac{h}{2}f(\alpha_1, \alpha_2)$			ω_1
$A+2$	$+$	α_4	ω_1	ω_1
$A+3$	$-$	ω_1	α_2	α_5
$A+4$	$+$	ω_1		α_2

Au moment du choix de l'approximation initiale $y_n^{(0)} = y_{n-1}$ dans le programme ci-dessus, l'opérateur de formation $\mathbf{\Phi}$ est absent. Il reste à remarquer que pour réaliser les conditions logiques P_1 et P_2, il est indispensable d'avoir les nombres ε et b que nous placerons dans les cellules α_6 et α_7.

Programme 1.

Nombres

α_1	x_0	x_n
α_2	y_0	$y_n^{(k)}$
α_3	h	
α_4		φ_{n-1}
α_5		l_n
α_6	ε	
α_7	b	
ω_1		$\frac{h}{2}f$

Instructions

$k+1$	$\frac{h}{2}f(\alpha_1, \alpha_2)$			ω_1
$k+2$	$+$	ω_1	α_2	α_4
$k+3$	$+$	α_1	α_3	α_1
$k+4$	$\frac{h}{2}f(\alpha_1, \alpha_2)$			ω_1
$k+5$	$+$	α_4	ω_1	ω_1
$k+6$	$-$	ω_1	α_2	α_5
$k+7$	$+$	ω_1		α_2
$k+8$	$\lvert \leqslant \rvert$	α_6	α_5	$k+4$

Instructions

$k + 9$	I	α_2		
$k + 10$	$\leqslant$	α_1	α_7	$k + 1$
$k + 11$	ARR			

Si, dans l'exemple précédent, l'intégrale est remplacée par une formule plus exacte à laquelle participent des valeurs de plus de deux points, le schéma du programme est complété par un opérateur de circulation qui prépare l'organe mémoire pour l'opérateur de calcul **B**.

Utilisons par exemple la formule d'intégration numérique (formule parabolique) suivante :

$$\int_{x_{n-2}}^{x_n} f(x)\,\mathrm{d}x = \frac{h}{6}\left[f(x_{n-2}) + 4\,f(x_{n-1}) + f(x_n)\right].$$

Le schéma de calcul pour résoudre le problème de l'exemple précédent a alors l'aspect :

$$\mathbf{B}\begin{cases} \varphi_{n-1} = y_{n-1} + \dfrac{h}{6}[f_{n-2} + 4\,f_{n-1}], \\ x_n = x_{n-1} + h, \end{cases}$$

$$\mathbf{A}\begin{cases} y_n^{(k+1)} = \dfrac{h}{6} f(x_n, y_n^{(k)}) + \varphi_{n-1}, \\ l_k = y_n^{(k-1)} - y_n^{(k)}, \end{cases}$$

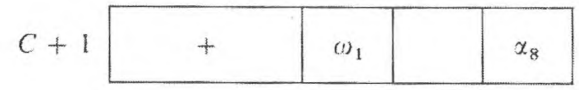
où $f_k = f(x_k, y_k)$.

Nous allons organiser le travail de l'opérateur **B** de la manière suivante : nous estimerons que $\dfrac{h}{6} f_0$, y_1 et le nombre 4 sont donnés dans les cellules α_8, α_9, et α_{10}.

Le programme de l'opérateur **B** est alors complété par deux instructions :

$B + 1$	$\dfrac{h}{6} f(\alpha_1, \alpha_2)$			ω_1
$B + 2$	$\times$	ω_1	α_{10}	α_4
$B + 3$	$+$	α_4	α_8	α_4
$B + 4$	$+$	α_4	α_9	α_4
$B + 5$	$+$	α_1	α_3	α_1

Pour que les calculs soient répétés au $(n + 1)$-ième point suivant, il est indispensable de placer la valeur f_{n-1} dans la cellule α_8 à la place de f_{n-2}. L'opérateur de décalage **C** (le cas le plus simple de circulation) est réalisé par l'instruction



et se place après l'opérateur **B.**

Le schéma du programme a l'aspect :

A B
BCAP$_1$ ↑ P$_2$ ↑ !

2. *Exemple de résolution des équations numériques*

$$f(x) = 0, \tag{1}$$

$f(x)$ étant une fonction continue, algébrique ou transcendante, sur un intervalle (a, b) donné.

Supposons que l'équation $f(x) = 0$ n'a pas de racines 2 n-uples. Nous supposerons, de plus, que les racines sont distinctes et que leur distance est supérieure à $d = 1/2^p$ (p entier).

Utilisons la méthode suivante d'approximations successives qui permet de déterminer les racines de l'équation $f(x) = 0$ à $\Delta = 1/2^n$ près ; ces racines seront rangées par ordre croissant sur l'intervalle (a, b).

Divisons ce segment en intervalles de longueur égale à $\Delta = 1/2^N$ où le nombre entier $N \leqslant N_0$, N_0 étant le nombre de positions binaires choisies pour fixer la mantisse des nombres. Pour plus de facilité, nous supposerons que les points a et b font partie des nœuds de ce réseau.

Suivant le signe de $f(x)$ qui se trouve en chacun d'eux, plusieurs petits domaines se divisent en sous-domaines. Si x^* est une racine de l'équation (1), nous obtiendrons dans les calculs soit $f(x^*) = +0$, soit $f(x) = -0$.

Convenons de considérer comme racines de l'équation les nœuds de droite de chacune des mailles élémentaires du réseau.

Désignons par x_i^* ($i = 1, 2, \ldots, s$; $x_{i-1}^* < x_i^*$) les racines inconnues de l'équation (1), et par x_i^0 la valeur initiale de la racine x_i^* :

$$x_i^{(0)} = \begin{cases} a & \text{si } i = 1 \\ x_{i-1}^* & \text{si } i \neq 1 . \end{cases} \tag{*}$$

Chaque valeur est donnée par la règle de récurrence

$$x_i^{(k+1)} = x_i^{(k)} + \Delta x_k , \quad \text{avec} \quad \Delta x_0 = \frac{1}{2^p} \quad \text{et} \quad k > 0$$

$$\Delta x_k = \begin{cases} \Delta x_{k-1}, & \text{si sign } f(x_i^{(k-1)}) = \text{sign } f(x_i^{(k)}), \\ -\dfrac{\Delta x_{k-1}}{2}, & \text{si sign } f(x_i^{(k-1)}) \neq \text{sign } f(x_i^{(k)}) \quad \text{et} \quad \Delta x_k > 2^{-N}. \end{cases}$$

On arrête les approximations successives lorsque :

$$\text{sign } f(x_i^{(k-1)}) \neq \text{sign } f(x_i^{(k)})$$

et

$$\Delta x_{k-1} = 2^{-N}.$$

Il est facile de voir que cette valeur est égale à la racine de x_i^*, s'il y a identité de la parité des nombres p et N.

En effet, pour passer de la k-ième à la $(k+1)$-ième vérification tous les pas accomplis à droite auront l'aspect $1/2^l$, où l a la même parité que p, et tous les pas à gauche auront l'aspect $-1/2^l$, où l a une parité opposée à celle de p. Aussi, le dernier pas de valeur $|1/2^N|$ sera-t-il accompli à droite si p a la même parité que N.

La valeur de x_i^* que l'on obtient est considérée comme une vérification nulle pour la racine x_{i+1}^* conformément à (*), etc. Les calculs ne s'arrêtent qu'après avoir obtenu la valeur x qui satisfasse à l'inégalité

$$x > b + \Delta x_0 .$$

De plus, il est clair que la différence maximale entre la vraie valeur de la racine et la valeur calculée peut égaler $1/2^N$, si on estime que le processus de calcul de $f(x)$ est tel que la valeur calculée $f(x)$ donne toujours un signe exact à $f(x)$.

Conformément à l'algorithme formulé, nous avons le schéma de calcul :

1. $x_i^{(k+1)} = x_i^{(k)} + \Delta x_k \quad (x_i^0 = a\,;\ \Delta x_0 = \Delta)\,;$ (A)

2. $y_{k+1} = f(x_i^{(k+1)})\,;$ (B)

3. $S_{k+1} = \text{sign } y_{k+1}\,;$ (C)

4. $$\Delta x_{k+1} = \begin{cases} \Delta x_k, & \text{si } s_k = s_{k+1}\,; \quad (D_1) \\ -\dfrac{\Delta x_k}{2}, & \text{si } s_k \neq s_{k+1}\,. \quad (D_2) \end{cases}$$

5. Si $\Delta x_k > 2^{-N}$ l'échantillonnage s'arrête, on a $x_i^* = x_i^{(k)}$ et l'on passe au calcul de la racine suivante.

6. Si $x_i^{k+1} > b + \Delta$ les calculs s'arrêtent.

Nous allons analyser les conditions logiques rencontrées au cours du problème.

1) La condition de fin des calculs P_1 :

$$P_1 = P_1 \{ x_i^{(k+1)} \geqslant b + \Delta \}$$

dépend de $x_i^{(k+1)}$; par conséquent nous placerons l'opérateur $\mathbf{P}_1$ qui assure la vérification et les transferts de commande après l'opérateur de calcul (**A**) qui change la valeur de x.

2) Condition de passage au calcul de Δx_{k+1} selon la formule (D_1) ou (D_2) :

$$P_2 = P_2 \{ s_k = s_{k-1} \} .$$

La condition P_2 (coïncidence des signes de la fonction lors de deux vérifications successives) est équivalente à la condition suivante, mieux adaptée au calcul :

$$P_2 \sim P_2' = P_2' \{ f(x_i^{(k+1)}) f(x_i^{(k)}) \geqslant +0 \} .$$

Pour tester la condition P_2' il faut la grandeur :

$$s = f(x_i^{(k+1)}) f(x_i^{(k)}) , \qquad (C')$$

pour laquelle il est nécessaire d'avoir à la fois les valeurs de la fonction au $(k+1)$-ième point donné et au k-ième point précédent.

En rapport avec ce qui précède, il convient d'inclure dans le schéma du programme l'opérateur d'envoi $\mathbf{Z}_1$ retenant la valeur de la fonction dans la vérification « donnée » en tant que valeur de la vérification « précédente » et qui servira dans le cycle suivant.

Puisque cette mémorisation doit avoir lieu indépendamment de la conclusion de la vérification de la condition $\mathbf{P}_2'$, il faut placer l'opérateur $\mathbf{Z}_1$ après le calcul de s et avant l'opérateur logique $\mathbf{P}_2'$.

3) La condition de fin de l'échantillonnage

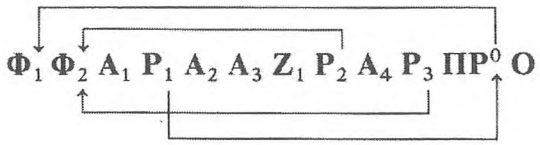
$$P_3 = P_3 \{ \Delta x_{k+1} \geqslant 2^{-N} \}$$

dépend de Δx ; c'est pourquoi il convient de placer l'opérateur **P** du schéma logique du programme après l'opérateur réalisant le changement Δx (D).

Introduisons les désignations suivantes :

L'opérateur $\mathbf{A}_1$ exécute les calculs d'après la formule (A),
— $\mathbf{A}_2$ — — — — (B),
— $\mathbf{A}_3$ — — — — (C'),
— $\mathbf{A}_4$ — — — — (D_2),
— $\mathbf{\Phi}_1$ forme les données initiales $x_0, f(x)_0$,
$\mathbf{\Phi}_2$ — les valeurs initiales $\Delta x_0 = \Delta$,
— $\mathbf{\Pi}$ est « imprimer x_i^* » (en convertissant en système décimal),
— $\mathbf{O}$ est « arrêt » .

Conformément à la description que nous avons faite de l'algorithme et des opérateurs pris séparément, nous obtenons le schéma de programme logique suivant :



Dans cet exemple, nous nous bornons à donner le schéma du programme. Nous laissons au lecteur le soin de décrire le programme avec sa modification préalable de manière à ce que la valeur de la fonction qui y est employée dans la vérification nulle ($f(x_0)$) soit calculée d'avance au moyen d'un même opérateur $\mathbf{A}_3$.

Remarquons aussi que la méthode exposée de recherche de la racine peut être facilement adaptée aux cas de la recherche de la racine la plus grande ou de la racine la plus petite ou encore des racines disposées sur un sous-intervalle de l'intervalle (a, b).

3. *Choix automatique du pas d'après un paramètre.*

Nous examinerons le problème du calcul des coordonnées des points d'une courbe définie paramétriquement :

$$\left.\begin{aligned} x &= x(t) \\ y &= y(t) \end{aligned}\right\} \quad (t_0 \leqslant t \leqslant T)\,;$$

où x et y sont les fonctions numériques et continues d'un paramètre t. Exigeons que la suite des coordonnées déterminées, c'est-à-dire, calculées pour l'impression des points (x_k, y_k) de la courbe examinée satisfasse aux conditions suivantes :

$$\begin{aligned} |\,\Delta x_k\,| &= |\,x_{k+1} - x_k\,| < \Delta\,, \\ |\,\Delta y_k\,| &= |\,y_{k+1} - y_k\,| < \Delta\,, \end{aligned} \tag{2}$$

et qu'au moins une des grandeurs Δx_k, Δy_k soit plus grande que δ :

$$|\,\Delta x_k\,| > \delta \quad \text{ou} \quad |\,\Delta y_k\,| > \delta\,. \tag{3}$$

Ici les grandeurs Δ et δ sont données et satisfont par exemple à la condition $4\,\delta < \Delta$.

La condition (2) signifie que les points dont les coordonnées sont imprimées ne doivent pas être situés trop loin les uns des autres sur la courbe ; au contraire, la condition (3) signifie que ces points ne doivent pas y être situés trop près les uns des autres.

Ces deux conditions sont entièrement naturelles, puisque l'accomplissement de la première permet d'obtenir une suite de points qui donne une représentation assez complète de la courbe. Le non-accomplissement de la seconde conduirait à un encombrement superflu du volume de l'information à sortir.

Admettons le résultat $x_k(t_k)$, $y_k(t_k)$ sur un certain pas ; alors pour calculer x_{k+1}, y_{k+1} nous prendrons le schéma de calcul suivant :

1) $t^i_{k+1} = t_k + \Delta t^i_{k+1}$ où $\Delta t^i_{k+1} = \Delta t_k$; $i = 1, 2, \ldots$;

2) $x^i_{k+1} = x(t^i_{k+1})$;
$y^i_{k+1} = y(t^i_{k+1})$;

3) $\Delta x^i_{k+1} = x^i_{k+1} - x_k$;
$\Delta y^i_{k+1} = y^i_{k+1} - y_k$;

4) $\Delta t^{i+1}_{k+1} = \dfrac{\Delta t^i_{k+1}}{2}$, si $P_1 = 0$;
$\Delta t^{i+1}_{k+1} = 2\,\Delta t^i_{k+1}$, si $P_2 = 0$;
$\Delta t^{i+1}_{k+1} = \Delta t_{k+1}$, si $P_1 = 1$ et $P_2 = 1$;

5) $t_{k+1} = t_k + \Delta t_{k+1}$; $x_{k+1} = x(t_{k+1})$; $y_{k+1} = y(t_{k+1})$.

6) Les calculs s'arrêtent lorsque $P_3 = 1$:

$$P_3 = P_3 \{ t \geqslant T \} .$$

Les conditions désignées par P_1 et P_2 sont :

$$P_1 = P_1 \{ |\Delta x^i_{k+1}| < 0 \quad \text{et} \quad |\Delta y^i_{k+1}| < \Delta \} ;$$
$$P_2 = P_2 \{ |\Delta x^i_{k+1}| > \delta \quad \text{ou} \quad |\Delta x^i_{k+1}| > \delta \} .$$

Désignons par :

$\mathbf{A}_1$ l'opérateur du calcul selon la formule 1 ,
$\mathbf{A}_2$ — — selon la formule 2-3 ,
$\mathbf{A}_3$ — — selon la formule 4-I ,
$\mathbf{A}_4$ — — selon la formule 4-II ,

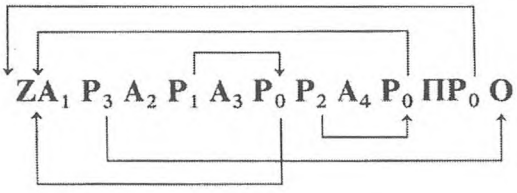
(la valeur Δt^{i+1}_{k+1} est placée dans la cellule qui contenait auparavant Δt^i_{k+1}),

$\mathbf{Z}$ l'opérateur d'envoi :

$$t_{k+1} \to t_k ,$$
$$x_{k+1} \to x_k ,$$
$$y_{k+1} \to y_k ;$$

П l'opérateur de sortie sur imprimante de x_{k+1} et y_{k+1} ;
О l'opérateur d'arrêt.

Nous obtenons le schéma logique du programme :



Désignons par

x	α_1		ω_3

y	α_1		ω_4

les instructions généralisées qui accomplissent le calcul x^i_{k+1} et y^i_{k+1} et rangent les résultats dans les cellules ω_3 et ω_4 selon la donnée t^i_{k+1} contenue dans la cellule α_1.

Le programme peut être établi sous la forme :

Nombres			Instructions					
α	t_0		$N+1$	+	−	α_1	α	
α_1		t_{k+1}	$N+2$	+	−	ω_3	ω_1	**Z**
ω_1	x_0		$N+3$	+	−	ω_4	ω_2	
ω_2	y_0		$N+4$	+	α	β	α_1	$\mathbf{A}_1$
ω_3		x_{k+1}	$N+5$	$\leqslant$	δ_1	α_1	$N+21$	$\mathbf{P}_3$
ω_4		y_{k+1}	$N+6$	x	α_1		ω_3	
β	Δt_0	Δt	$N+7$	y	α_1		ω_4	$\mathbf{A}_2$
δ_1	T		$N+8$	−	ω_1	ω_3	γ_1	
δ_2	Δ		$N+9$	−	ω_2	ω_4	γ_2	

Nombres

δ_3	2	
δ_4	δ	
γ_1		Δx
γ_2		Δy

Instructions

$N+10$	$\lvert \leqslant \rvert$	δ_2	γ_1	$N+12$	$\mathbf{P}_1$
$N+11$	$\lvert \leqslant \rvert$	γ_2	δ_2	$N+14$	
$N+12$	:	β	δ_3	β	$\mathbf{A}_3$
$N+13$	$\leqslant$	–	–	$N+4$	$\mathbf{P}_0$
$N+14$	$\lvert \leqslant \rvert$	δ_4	γ_1	$N+18$	$\mathbf{P}_2$
$N+15$	$\lvert \leqslant \rvert$	δ_4	γ_2	$N+18$	
$N+16$	$\times$	β	δ_3	β	$\mathbf{A}_4$
$N+17$	$\leqslant$	–	–	$N+4$	$\mathbf{P}_0$
$N+18$	I			ω_3	Π
$N+19$	I			ω_4	
$N+20$	$\leqslant$	–	–	$N+1$	$\mathbf{P}_0$
$N+21$	ARR				$\mathbf{O}$

La programmation des processus cycliques binaires à un paramètre ne présente pas de nouvelles difficultés et peut être faite par le lecteur (cf. exercice 2).

2. Processus cyclique binaire à deux paramètres.

1. *Résolution d'une équation de conductibilité de la chaleur.*

L'équation de conductibilité de la chaleur

$$\frac{\partial U}{\partial t} = a^2 \frac{\partial^2 U}{\partial x^2}$$

dans le domaine défini par $0 \leqslant t \leqslant T$, $0 \leqslant x \leqslant X$, avec les conditions initiales et finales

$$U(0, x) = f(x) \; ; \quad U(t, 0) = \varphi(t) \; ; \quad U(t, X) = \psi(t) \, ,$$

peut être résolue d'après le schéma suivant :

supposons $$\Delta x = h \; ; \quad \Delta t = \frac{h^2}{2\,a^2}$$

et désignons

$$K = \frac{2\,a^2}{h^2}\,T\,; \quad J = \frac{X}{h}\,,$$

$$U_{ki} = U(k\,\Delta t,\, ih) \qquad (k = 0, 1, \ldots, K\,; \quad i = 0, 1, \ldots, J)\,.$$

Substituons :

$$\frac{\partial U}{\partial t} \simeq \frac{1}{\Delta t}\,(U_{k+1,i} - U_{k,i})\,; \quad \frac{\partial^2 U}{\partial x^2} \simeq \frac{1}{h^2}\,(U_{k,i+1} - 2\,U_{k,i} + U_{k,i-1})\,.$$

A l'équation de conductibilité correspondra alors l'équation différentielle :

$$U_{k+1,i} = \frac{1}{2}\,(U_{k,i-1} + U_{k,i+1})\,, \qquad \begin{matrix} k = 0, 1, \ldots, K-1\,, \\ i = 0, 1, \ldots, J-1\,. \end{matrix}$$

Ces formules déterminent l'opérateur de calcul $\mathbf{A}_{ik}$ qui dépend de deux paramètres : i et k. Le schéma du programme correspond à un processus cyclique binaire à deux paramètres :

$$\mathbf{\Phi}_{(k)}\,\mathbf{A}_{ik}\,\mathbf{F}(i)\,\mathbf{P}_1\,\overset{\mathbf{A}}{\uparrow}\,\mathbf{F}(k)\,\mathbf{\Phi}(i)\,\mathbf{P}_2\,\overset{\mathbf{A}}{\uparrow}\,!$$

La forme du programme dépend de la situation des nombres U_{ki} dans les cellules de l'organe mémoire. Supposons que les nombres U_{ki} soient placés dans les cellules $\alpha + i + kJ$

$$(k = 0, 1, 2, \ldots, K\,; \quad i = 0, 1, 2, \ldots, J)\,.$$

Pour programmer l'opérateur $\mathbf{A}_{ik}$ nous placerons le nombre 1/2 dans la cellule β.

L'opérateur $\mathbf{A}_{ik}$ est réalisé par les instructions :

$\mathbf{A}_{ik}$

$A+1$	+	$\alpha + (i-1) + kJ$	$\alpha + (i+1) + kJ$	ω
$A+2$	×	ω	β	$\alpha + i + (k+1)\,J$

La forme initiale de l'opérateur $\mathbf{A}_{ik}$, pour $i = k = 1$, est la suivante :

$\mathbf{A}_{11}$

$A+1$	+	$\alpha + 0 + J$	$\alpha + 2 + J$	ω
$A+2$	×	ω	β	$\alpha + 1 + 2\,J$

La variation des adresses de l'opérateur $\mathbf{A}_{ik}$ en fonction des paramètres est déterminée par le tableau

$A + 1$	$(i - 1) + kJ$	$i + 1 + kJ$	
$A + 2$			$i + (k + 1)\,J$

Il résulte de ce tableau que, pour faire varier d'une unité le paramètre i, il est indispensable d'avoir deux constantes de substitution d'adresse. L'une contient un « un » dans la première et la seconde adresse que nous avons convenu de désigner :

	1	1	

et l'autre ne contient qu'un seul « un » dans la troisième adresse, c'est-à-dire qu'elle a la forme :

			1

Plaçons les constantes de substitution d'adresse

	1	1	

			1

dans les cellules δ_1 et δ_2. Le programme de l'opérateur de substitution d'adresse prendra la forme

$\mathbf{F}(i)$

$F + 1$	$\oplus$	$A + 1$	δ_1	$A + 1$
$F + 2$	$\oplus$	$A + 2$	δ_2	$A + 2$

Pour programmer la fin du processus cyclique interne, il faut utiliser, soit un compteur du nombre de cycles spécial, soit l'une des instructions variables de l'opérateur A_{ik}, par exemple $A + 2$.

Examinons le deuxième moyen. La forme définitive de l'instruction $A + 2$ au dernier cycle selon le paramètre i est la suivante :

$A + 2$	$\times$	ω	β	$\alpha + J + (k + 1)J$

Nous plaçons le code correspondant en δ_3 et nous l'utilisons comme constante de la comparaison, ayant remarqué que ce code dépend de k :

δ_3			$(k + 1)J$

La condition logique P_1 est alors réalisée par l'instruction :

$P_1 + 1$	$\leqslant$	$A + 2$	δ_3	$A + 1$

Pour programmer $\mathbf{F}(k)$ et $\mathbf{\Phi}(i)$, il faut connaître la forme finale de l'opérateur $\mathbf{A}_{ik}$ une fois que le processus cyclique initial est réalisé selon le paramètre i, c'est-à-dire $\mathbf{A}_{J,k}$, et la forme initiale de cet opérateur $\mathbf{A}_{i,k}$ avant que ne soit réalisé le processus cyclique selon le paramètre k, c'est-à-dire $\mathbf{A}_{1,k+1}$.

$\mathbf{A}_{1,k+1}$

$A + 1$	$+$	$\alpha + 0 + (k + 1)J$	$\alpha + 2 + (k + 1)J$	ω
$A + 2$	$\times$	ω	β	$\alpha + 1 + (k + 2)J$

En comparant $\mathbf{A}_{1,k+1}$ et $\mathbf{A}_{J,k}$, nous trouvons que pour transformer $\mathbf{A}_{J,k}$ en $\mathbf{A}_{1,k+1}$ il faut dans l'instruction $A + 1$ faire varier d'une unité la première et la seconde adresse, et dans l'instruction $A + 2$ faire de même pour la troisième adresse. En outre, il faut que le changement soit fait en fonction du paramètre k de la constante de δ_3, ce qui s'effectuera à l'aide du nombre constant « J unités dans la troisième adresse », que nous placerons dans la cellule δ_7. Le programme des opérateurs $\mathbf{F}(k)$, $\mathbf{\Phi}(i)$ est :

$F + 1$	$\oplus$	$A + 1$	δ_1	$A + 1$
$F + 2$	$\oplus$	$A + 2$	δ_2	$A + 2$
$F + 3$	$+$	δ_3	δ_7	δ_3

Pour programmer la condition logique P_2, on peut utiliser la forme finale de l'instruction $A + 2$ si $k = K$ et $i = J$:

$A + 2$	$\times$	ω	β	$\alpha + (K + 2)\,J$

Plaçons la constante :

$\times$	ω	β	$\alpha + (K + 2)\,J$

dans la cellule δ_8. La condition logique est alors réalisée par l'instruction :

$\leqslant$	$A + 2$	δ_8	$A + 1$

Il reste à établir un programme pour l'opérateur $\Phi(k)$ rétablissant la première forme de l'opérateur $\mathbf{A}_{ik}$ pour que l'on puisse l'appliquer de manière répétée. Le passage de la forme finale de l'opérateur $\mathbf{A}_{ik}$ après la fin des processus cycliques selon les paramètres i et k à la forme initiale $\mathbf{A}_{11}$ peut être effectué, soit à l'aide de l'envoi des premières instructions $A + 1$ et $A + 2$ des cellules fixes de l'organe de mémoire δ_5 et δ_6 dans les emplacements correspondants du programme, soit à l'aide du calcul des constantes appropriées des instructions $A + 1$ et $A + 2$ que nous obtiendrons à partir de la comparaison de $\mathbf{A}_{1,k}$ et $\mathbf{A}_{11}$, et particulièrement des constantes :

$(k - 1)\,J$	$(k - 1)\,J$	

		$(k - 1)\,J$

Nous placerons ces constantes dans les cellules de l'organe mémoire δ_5 et δ_6. Conformément à ceci l'opérateur $\Phi(k)$ peut être réalisé soit par les instructions :

$\Phi + 1$	$\oplus$	δ_5		$A + 1$
$\Phi + 2$	$\oplus$	δ_6		$A + 2$

Programme 2.

Nombres

$\alpha + i + kj$ $(i = 0, 1, \ldots, J)$ $(k = 0, 1, \ldots, K)$	u_{ik}			
β	$\frac{1}{2}$			
δ_1		1	1	
δ_2				1
δ_3	$\times$	ω	β	$\alpha + 3J - 1$
δ_4	$\times$	ω	β	$\alpha + (k+1)J$
δ_5	$+$	$\alpha + J$	$\alpha + 2 + J$	ω
δ_6	$\times$	ω	β	$\alpha + 1 + 2J$
δ_7				J
ω	de travail			

		$(k+1)J$

Instructions

$k + 1$	$\oplus$	δ_5		$k + 3$
$k + 2$	$\oplus$	δ_6		$k + 4$
$k + 3$	$+$	$\alpha + J$	$\alpha + 2 + J$	ω
$k + 4$	$\times$	ω	β	$\alpha + 1 + 2J$
$k + 5$	$\oplus$	$k + 3$	δ_1	$k + 3$
$k + 6$	$\oplus$	$k + 4$	δ_2	$k + 4$
$k + 7$	$\leqslant$	$k + 4$	δ_3	$k + 3$
$k + 8$	$\oplus$	$k + 3$	δ_1	$k + 3$
$k + 9$	$\oplus$	$k + 4$	δ_2	$k + 4$
$k + 10$	$+$	δ_3	δ_7	δ_3
$k + 11$	$\leqslant$	$k + 4$	δ_4	$k + 3$
$k + 12$	*ARR*			

Variantes

$k + 3$		$i - 1 + kJ$	$i + 1 + kJ$	
$k + 4$		$-$	$-$	$i + (k + 1)J$

soit par les instructions :

$\Phi + 1$	$\ominus$	$A + 1$	δ_5	$A + 1$
$\Phi + 2$	$\ominus$	$A + 2$	δ_6	$A + 2$

Puisque le volume de la mémoire employée est identique, la première variante est préférable à la seconde. Cependant, si les constantes de soustraction des cellules δ_5 et δ_6 dans la seconde variante avaient été formées au cours du calcul selon le programme, cette variante aurait été meilleure.

Nous avons remarqué ci-dessus que pour programmer les conditions logiques P_1 et P_2 il est indispensable de faire le calcul du nombre de cycles d'après les paramètres i et k. Le calcul est réalisé à l'aide des instructions variables de l'opérateur $\mathbf{A}_{ik}$. Cependant, on peut utiliser le compteur du nombre de cycles dans la cellule spéciale de l'organe mémoire δ_3, dans laquelle on met zéro pour débuter. Comme unité de calcul, on utilise la constante de substitution d'adresse de δ_2. On peut réaliser le compteur à l'aide de l'instruction :

$C + 1$	$+$	δ_2	δ_3	δ_3

Le nombre des cycles selon le paramètre i est égal à $J - 1$. Par conséquent, pour satisfaire à la condition logique P_1 il est indispensable d'avoir le nombre « $J - 1$ unités dans la troisième adresse », que nous placerons dans la cellule δ_8. Alors la condition logique peut être réalisée par l'instruction

$P_1 + 1$	$\leqslant$	δ_3	δ_8	$A + 1$

Pour réaliser la condition logique P_2 on a besoin d'avoir la constante « $(k + 1)\,J$ unités dans la troisième adresse », que nous placerons dans la cellule δ_4 :

$P_2 + 1$	$\leqslant$	δ_3	δ_4	$A + 1$

Remarquons qu'à présent la constante δ_4 dépend du paramètre k, et, par conséquent, doit être remplacée par l'opérateur $\mathbf{F}(k)\,\mathbf{\Phi}(i)$. Dans ce cas, nous laissons au lecteur le soin de changer le programme des opérateurs $\mathbf{F}(k)\,\mathbf{\Phi}(i)$.

2. *Développement du polynôme* $f(x)=a_n x^n+a_{n-1} x^{n-1}+\cdots+a_1 x+a_0$ selon les puissances de $(x-a)$.

Il nous est nécessaire de déterminer les coefficients b_k $(k = 0, 1, 2, \ldots, n)$ du développement du polynôme selon les puissances de $(x - a)$

$$f(x) = b_n(x - a)^n + b_{n-1}(x - a)^{n-1} + \cdots + b_1(x - a) + b_0 ;$$

nous les déterminerons en appliquant l'algorithme de Hörner de division du polynôme $f(x)$ par $(x - a)$. Les coefficients à la i-ième ligne du schéma de Hörner se calculent suivant la formule de récurrence :

$$a_k^{(i)} = aa_{k+1}^{(i)} + a_k^{(i-1)},$$

où

$$k = n - 1, n - 2, \ldots, i - 1 ; \quad i = 1, 2, \ldots, n .$$

De plus $a_k^{(0)} = a_k$ $(k = 0, 1, 2, \ldots, n)$; $a_{i-1}^{(i)} = b_{i-1}$ sont les nombres cherchés. L'opérateur de calcul $\mathbf{A}_{ki}$ dépend des deux paramètres : k et i ; le nombre de cycles intérieurs selon le paramètre k dépend de i. Le schéma du programme a la forme :

$$\mathbf{A}_{ki}\ \mathbf{F}(k)\ \mathbf{P}_1\ \uparrow\ \mathbf{F}(i)\ \Phi(k)\ \mathbf{P}_2\ \uparrow\ !$$

Plaçons les données initiales qui permettent de calculer l'opérateur $\mathbf{A}_{ki}$ — les nombres $a_k = a_k^{(0)}$ $(k = 0, 1, \ldots, n)$ et le nombre a — dans les cellules de l'organe mémoire $\alpha + k$ $(k = 0, 1, \ldots, n)$ et β. Les résultats des calculs sont aussi placés dans les cellules $\alpha + k$. L'opérateur $\mathbf{A}_{ki}$ est programmé par les instructions :

$\mathbf{A}_{ki}$

$A + 1$	×	$\alpha + n$	β	ω	$k + 1$		
$A + 2$	+	$\alpha + n - 1$	ω	$\alpha + n - 1$	k		k

Pour réaliser l'opérateur $\mathbf{F}(k)$, il faut avoir une constante de substitution d'adresse que nous placerons dans les cellules δ_1 et δ_2 :

δ_1	–	1	–	–

δ_2	–	1	–	1

$F_1 + 1$	$\ominus$	$A + 1$	δ_1	$A + 1$
$F_2 + 2$	$\ominus$	$A + 2$	δ_2	$A + 2$

Pour programmer la condition logique P_1, il faut utiliser l'instruction $A + 1$ qui, à la fin du cycle initial, a la forme

$A + 1$	$\times$	$\alpha + 0$	β	ω	$i - 1$		

La constante qui sert à la comparaison

$\times$	$\alpha + 0$		

et qui dépend du paramètre i se place dans la cellule δ_3 :

δ_3	$\times$	$\alpha + 0$			$i - 1$		

Pour réaliser l'opérateur $\mathbf{F}(i)$ il ne faut changer que la constante de δ_3, ce qui est réalisé par l'instruction

$F_2 + 1$	$+$	δ_3	δ_1	δ_3

Puisque la forme de l'opérateur $\mathbf{A}_{ki}$, après le processus cyclique selon le paramètre k, dépend du paramètre i, il convient de réaliser le rétablissement de l'opérateur $\mathbf{A}_{ki}$ par le transfert des codes

δ_4	$\times$	$\alpha + n$	β	ω

δ_5	$+$	$\alpha + n - 1$	ω	$\alpha + n - 1$

dans les cellules $A + 1$ et $A + 2$. On place naturellement l'opérateur $\mathbf{\Phi}_{(k)}$ au début du programme. Pour programmer la condition logique P_2 il faut uti-

liser la constante contenue dans la cellule δ_3, dont la forme finale est :

×	$\alpha + n$	–	–

Nous placerons la constante dans la cellule δ_6

×	$\alpha + n - 1$	–	–

Il faut alors réaliser la condition P_2 par l'instruction :

$\leqslant$	δ_3	δ_6	$\Phi + 1$

Programme 3.

Nombres

$\alpha + k$ $0 \leqslant k \leqslant n$	a_k		b_k	
β	a			
δ_1	–	1	–	–
δ_2	–	1	–	1
δ_3	×	$\alpha + 0$	–	–
δ_4	×	$\alpha + n$	β	ω
δ_5	+	$\alpha+n-1$	ω	$\alpha+n-1$
δ_6	×	$\alpha+n+1$	–	–
ω	de travail			

i	–	–

Instructions

$k + 1$	+	δ_4	+	$k + 3$
$k + 2$	+	δ_5	+	$k + 4$
$k + 3$	×	$\alpha + n$	β	ω
$k + 4$	+	$\alpha+n-1$	ω	$\alpha+n-1$
$k + 5$	$\ominus$	$k + 3$	δ_1	$k + 3$
$k + 6$	$\ominus$	$k + 4$	δ_2	$k + 4$
$k + 7$	$\leqslant$	$k + 3$	δ_3	$k + 3$
$k + 8$	+	δ_3	δ_1	δ_3
$k + 9$	$\leqslant$	δ_3	δ_6	$k + 1$
$k + 10$	ARR			

3. Processus cyclique ternaire.

1. *Résolution des systèmes d'équations linéaires par une méthode itérative.*

Soit le système d'équations linéaires $Ax = b$, où $A = \{ a_{ik} \}^m$ est la matrice carrée du m-ième degré et $b = (b_1, b_2, \ldots, b_m)$, un vecteur à m dimensions. Si

la matrice A satisfait à des conditions connues, ce système peut être résolu par la méthode itérative. Prenons le schéma de calcul suivant :

$$\Delta x_k^{(n)} = x_k^{(n)} - x_k^{(n-1)} = \sum_{i=1}^{k-1} a_{ki}\, x_i^{(n)} + \sum_{i=k}^{m} a_{ki}\, x_i^{(n-1)} + b_k\,,$$

$$x_k^{(n)} = x_k^{(n-1)} + \Delta x_k^{(n)}\,,$$

$$k = 1, 2, \ldots, m\,.$$

Les itérations cessent quand :

$$\| \Delta x^{(n)} \| = \sqrt{\sum_{k=1}^{m} (x_k^{(n)})^2} < \varepsilon\,,$$

où ε est un nombre donné. Le schéma de calcul contient trois paramètres : k, i et m. Introduisons les opérateurs de calcul en tenant compte de leur dépendance des paramètres. Au début agit l'opérateur $\mathbf{A}_{ik}$

$$\sum_{i=1}^{k-1} a_{ki}\, x_i^{(n)} + \sum_{i=k}^{m} a_{ki}\, x_i^{(n-1)} = s_k^{(n)}\,,$$

ensuite l'opérateur $\mathbf{B}_k$:

$$\Delta x_k^{(n)} = s_k^{(n)} + b_k\,; \quad x_k^{(n)} = x_k^{(n-1)} + \Delta x_k^{(n)}\,;$$

$$d_0 = 0\,, \quad d_k = d_{k-1} + (\Delta x_k^{(n)})^2\,, \quad k = 1, 2, \ldots, m\,.$$

Le schéma du programme est le suivant :

$$\mathbf{A}_{ik}\, \mathbf{F}(i)\, \mathbf{P}_1 \uparrow \mathbf{B}_k\, \mathbf{F}_1(k)\, \mathbf{\Phi}_1(i)\, \mathbf{P}_2 \uparrow \mathbf{F}_2(n)\, \mathbf{\Phi}_2(k)\, \mathbf{P}_3 \uparrow !$$
$$\mathbf{A} \qquad\qquad \mathbf{A} \qquad\qquad \mathbf{A}$$

Plaçons les coefficients du système a_{ik} dans les cellules $\alpha + i + (k-1)\,m$ $(i = 1, 2, \ldots, m\,;\, k = 1, 2, \ldots, m)$, les parties droites des équations b_k dans les cellules $\beta + k$ $(k = 1, 2, \ldots, m)$ et les valeurs initiales des inconnues $x_k^{(0)}$ dans les cellules $\gamma + k$ $(k = 1, 2, \ldots, m)$. Dans ces mêmes cellules on placera $x_k^{(n)}$. Affectons aux nombres $\Delta x_k^{(n)}$ une seule et même cellule ω_1, qu'il est indispensable de remettre à zéro lorsque le paramètre k change. Le programme de l'opérateur $\mathbf{A}_{ik}$ est

$\mathbf{A}_{ik}$

$A+1$	$\times$	$\alpha + 1 + 0.m$	$\gamma + 1$	ω_2
$A+2$	$+$	ω_1	ω_2	ω_1

	$i + (k-1)\,m$	k	$-$

Le programme de l'opérateur $\mathbf{F}_1(i)$ est facile à établir :

$\mathbf{F}_1(i)$	$F_1 + 1$	$\oplus$	$A + 1$	δ_1	$A + 1$

δ_1	–	1	–	–

Pour programmer la condition logique P_1, on utilise l'instruction $A + 1$ et la constante qui est définie par la forme finale de l'instruction $A + 1$ lorsque $i = m + 1$.

$A + 1$	$\times$	$\alpha + m + 1$	$\gamma + 1$	ω_2		$(k - 1)\,m$	k	–

Plaçons cette constante dans la cellule δ_2, après avoir noté sa variation en fonction du paramètre k.

Le programme de l'opérateur $\mathbf{B}_k$

$\mathbf{B}_k$									
$B + 1$	$+$	ω_1	$\beta + 1$	ω_1	–	–	k	–
$B + 2$	$+$	ω_1	$\gamma + 1$	$\gamma + 1$	–	–	k	k
$B + 3$	$\times$	ω_1	ω_1	ω_1				
$B + 4$	$+$	ω_3	ω_1	ω_3				

utilise la cellule ω_3 dans laquelle est placé $\| \Delta x^{(n)} \|$; par conséquent ω_3 dépend du paramètre n. Les opérateurs $\mathbf{F}_2(k)$ $\mathbf{F}_1(i)$ modifient l'instruction $A + 1$, remettent à zéro la cellule ω_1 et changent la constante δ_2. Nous obtiendrons la constante qui permet le changement de l'instruction $A + 1$ en soustrayant de la $(A + 1)$-ième instruction (lorsque la valeur des paramètres est $i = 1$ et $k = k + 1$) sa valeur lorsque $i = m + 1$ et $k = k$.

	$1 + km$	$k + 1$	–

	$m + 1 + (k - 1)\,m$	k	–

δ_3	–	–	1	–

Pour modifier δ_2 on utilise la constante :

–	m	–	–

qui est placée dans la cellule δ_4. Le programme $\mathbf{F}_2(k)\,\mathbf{\Phi}_1(i)$ est :

	$F_2 + 1$	$\oplus$	$A + 1$	δ_3	$A + 1$
$\mathbf{F}_2(k)\,\mathbf{\Phi}_1(i)$	$F_2 + 2$	$+$	δ_2	δ_4	δ_2
	$F_2 + 3$	$+$			ω_1

Comme dans l'exemple précédent, grâce à la relation entre les paramètres i et k, les opérateurs $\mathbf{F}_2(k)$ et $\mathbf{\Phi}_1(i)$ sont programmés en même temps. La condition logique P_2 est réalisée par la comparaison de δ_2 avec la constante :

δ_5	$\times$	$\alpha + m^2 + 1$	–	–

Les opérateurs $\mathbf{F}_3(n)\,\mathbf{\Phi}_2(k)$ remettent à zéro la cellule ω_3 et rétablissent la forme initiale des opérateurs $\mathbf{A}_{ik}$ et $\mathbf{B}_k$ à l'aide des constantes de substitution d'adresse

δ_6	–	$m^2 - 1$	m	–
δ_7	–	–	m	–
δ_8	–	–	m	m

La condition logique P_3 est réalisée par la comparaison de la constante $(\delta_9) = \varepsilon^2$ avec $(\omega_3) = \| \Delta x^{(n)} \|^2$.

Il vaut mieux effectuer la remise à zéro de ω_3 au début du programme.

2. *Traitement du problème de Dirichlet pour le rectangle.*

L'équation de Laplace

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0\,,$$

avec les conditions limites sur les côtés du rectangle $0 \leqslant x \leqslant a$, $0 \leqslant y \leqslant b$

Programme 4.

Nombres

$\alpha + i + (k-1)m$ $1 \leqslant i \leqslant m$ $1 \leqslant k \leqslant m$	a_{ki}				
$\beta + k$ $1 \leqslant k \leqslant m$	b_k				
$\gamma + k$ $1 \leqslant k \leqslant m$	$x_k^{(0)}$				
δ_1	$-$	1	$-$	$-$	
δ_2	$\times$	$\alpha + m + 1$	$\gamma + 1$	ω_2	$(k-1)m$ | k | $-$
δ_3	$-$	$-$	1	$-$	
δ_4	$-$	m	$-$	$-$	
δ_5	$\times$	$\alpha + m^2 + 1$	$-$	$-$	
δ_6	$-$	$m^2 - 1$	m	$-$	
δ_7	$-$	$-$	m	$-$	
δ_8	$-$	$-$	m	m	
δ_9		ε^2			
ω_1		0	$s_k^{(n)}$		
ω_2	de travail				
ω_3	0		$a_k^{(n)}$		

Instructions

$k+1$	$+$	$-$	$-$	ω_3	
$k+2$	$+$	$-$	$-$	ω_1	
$k+3$	$\times$	$\alpha+1$	$\gamma+1$	ω_2	$i + k.m$ | k | $-$
$k+4$	$+$	ω_1	ω_2	ω_1	
$k+5$	$\oplus$	$k+3$	δ_1	$k+3$	
$k+6$	$\leqslant$	$k+3$	δ_2	$k+3$	
$k+7$	$+$	ω_1	$\beta+1$	ω_1	$-$ | k | $-$
$k+8$	$+$	ω_1	$\gamma+1$	$\gamma+1$	$-$ | k | k
$k+9$	$\times$	ω_1	ω_1	ω_1	
$k+10$	$+$	ω_3	ω_1	ω_3	
$k+11$	$\oplus$	$k+3$	δ_3	$k+3$	
$k+12$	$+$	δ_2	δ_4	δ_2	
$k+13$	$\leqslant$	δ_2	δ_5	$k+2$	
$k+14$	$\ominus$	$k+3$	δ_6	$k+3$	
$k+15$	$\ominus$	$k+7$	δ_7	$k+7$	
$k+16$	$\ominus$	$k+8$	δ_8	$k+8$	
$k+17$	$\leqslant$	δ_9	ω_3	$k+1$	
$k+18$	$!$				

par l'application de la méthode des quadrillages, peut être réduite à la résolution du système d'équations linéaires

$$u(kh, ih) = \tfrac{1}{4}[u(kh, \quad (i+1)h) + u(kh, \quad (i-1)h) + \\ + u((k-1)h, \quad ih) + u((k+1)h, \quad ih)],$$

où $k = 0, 1, 2, \ldots, K$, $i = 0, 1, 2, \ldots, J$, que l'on peut résoudre facilement par la méthode des itérations :

$$u_{ki}^{(s+1)} = \tfrac{1}{4}[u_{k,i+1}^{(s)} + u_{k,i-1}^{(s)} + u_{k-1,i}^{(s)} + u_{k+1,i}^{(s)}],$$
$$\Delta_s = \max_{k,i} | u_{ki}^{(s+1)} - u_{k,i}^{(s)} | .$$

On peut juger du degré d'exactitude de la solution d'après la différence maximale entre deux approximations successives Δ_s. Les données initiales du problème (les valeurs limites) et les approximations initiales des points du quadrillage $u_{ki}^{(0)}$ seront placées dans les cellules

$$\alpha + k + iK, \quad k = 0, 2, \ldots, K; \quad i = 0, 1, \ldots, J.$$

L'opérateur de calcul selon les formules $\mathbf{A}_{kis}$ dépend de trois paramètres. Il est facile d'établir le schéma du programme :

$$\overset{\mathbf{A}}{\Phi_1(s)\,\mathbf{A}_{kis}\,\mathbf{F}_1(k)\,\mathbf{P}_1}\uparrow \overset{\mathbf{A}}{\mathbf{F}_2(i)\,\Phi_2(k)\,\mathbf{P}_2}\uparrow \overset{\mathbf{A}}{\mathbf{F}_3(s)\,\Phi_3(i)\,\mathbf{P}_3}\uparrow !$$

Nous laissons le lecteur écrire le programme instruction par instruction.

3. *Calcul du déterminant d'une matrice carrée non dégénérée.*

Soit $A = \{ a_{ik} \}$ $(i = 1, 2, \ldots, n ; k = 1, 2, \ldots, n)$ la matrice carrée du n-ième degré. Le déterminant de la matrice D sera calculé en réduisant la matrice A à la forme diagonale. Par soustraction des lignes, nous remettons à zéro tous les éléments de la matrice qui se trouvent au-dessous de la diagonale principale. Supposons que dans le processus de calcul, les éléments diagonaux ne soient pas remis à zéro. Introduisons les opérateurs de calcul suivants :

$\mathbf{C}_{ir}$ calcul de $C_{ir} = \dfrac{a_{ir}}{a_{rr}}$; $r = 2, 3, \ldots, n$; $i = r+1, r+2, \ldots, n$;

$\mathbf{A}_{kir}$ calcul de $a_{ik}^{(r)} = a_{ik}^{(r-1)} - a_{rk}^{(r-1)} C_{ir}$; $k = r, r+1, \ldots, n$;

$\mathbf{D}_r$ calcul de $d_r = d_{r-1}\, a_{rr}^{(r+1)}$, $d_0 = 1$; $d_n = D_A$.

Le schéma du programme a la forme :

$$\overset{\mathbf{A}}{\Phi_1(r)\,\mathbf{D}_r\,\mathbf{C}_{ir}\,\mathbf{A}_{kir}\,\mathbf{F}_1(k)\,\mathbf{P}_1}\uparrow \overset{\mathbf{C}}{\mathbf{F}_2(i,k)\,\mathbf{P}_2}\uparrow \overset{\mathbf{D}}{\mathbf{F}_3(r,i)\,\mathbf{P}_3}\uparrow !$$

Remarquons qu'il est possible de réaliser le calcul selon le paramètre k, en commençant par $k = 1$; aussi lorsque $k < r$, il n'y aura que des soustractions de zéro, ce qui ne change nullement les résultats du calcul. Mais l'opérateur qui forme le paramètre k peut se simplifier. Le calcul selon le paramètre i doit débuter par $i = r + 1$, puisque le début $i = r$ ferait soustraire la r-ième ligne d'elle-même, ce qui est impossible. Nous savons déjà établir un programme sur un schéma donné.

EXERCICES

1. *Etablir le programme de résolution de l'équation différentielle* $\frac{dy}{dx} = f(x, y)$ *selon le schéma de programme suivant :*

$$\begin{array}{c} \mathbf{A} \quad \mathbf{\Phi} \\ \mathbf{\Phi}\ \mathbf{AP}_1 \uparrow \mathbf{BP}_2 \uparrow\ !\ \ \text{(cf. § 1)}. \end{array}$$

2. *Etablir le schéma et le programme de calcul et d'impression de la table des valeurs du polynôme* $f(x + ih)$, $i = 0, 1, 2, \ldots, J$.

3. *Etablir le programme du développement du polynôme :*

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

d'après les puissances de $(x - a)$ avec le calcul des coefficients du développement b_k dans les cellules $\beta + k$, $k = 0, 1, 2, \ldots, n$.

4. *Comment transformer le programme de l'exercice* 3 *si l'on place les coefficients* a_k *du polynôme dans les cellules* $\alpha + n - k$, $k = 0, 1, 2, \ldots, n$?

5. *Etablir le programme de la multiplication de deux matrices carrées de rang* n :

$$A = \{ a_{ik} \}_1^n \quad \text{et} \quad B = \{ b_{ik} \}_1^n .$$

4. Processus cyclique complexe.

1. *Résolution graphique d'une fonction de deux variables.*

Examinons la courbe de la fonction continue de deux variables :

$$f(x, y) = c . \tag{4}$$

Supposons que la courbe (4) divise le plan en deux régions liées dans lesquelles la fonction

$$\varphi(x, y) = f(x, y) - c \tag{5}$$

a des signes différents. Traçons dans le plan xOy un réseau de droites de coordonnées différant de Δx et Δy. Appelons « courants » les rectangles du réseau qui ont des points communs avec la courbe (4). Remarquons que, étant donné

que sur toute C. A. N. le zéro a aussi un signe, chacun des nœuds, y compris ceux pour lesquels $\varphi(x, y) = 0$, a un signe déterminé.

Il est évident que la question de la détermination d'une courbe en machine opérant par valeurs discrètes consiste à trouver les sommets des rectangles « courants » pour des Δx et Δy suffisamment petits.

La manière la plus simple d'aborder un problème sur C. A. N. est le procédé dans lequel, sur chaque horizontale (ou verticale) d'un rectangle assez grand (les dimensions en sont dictées lorsqu'on traite un problème concret), le signe de φ est successivement déterminé à chaque nœud du réseau ; au cas où, lors du passage d'un nœud à un autre, le signe du nœud change, nous considérerons ces sommets comme connus et nous les fixerons (les imprimerons). Ensuite, après avoir traversé l'une des verticales (ou horizontales) il faudra prévoir le passage à la suivante, etc., jusqu'au complet épuisement du rectangle examiné, après quoi les calculs s'arrêteront. Bien entendu, dans une machine à virgule fixe, la région des valeurs des variables qui nous intéresse se transforme par le choix des échelles à l'intérieur du carré $(|x| < 1, \quad |y| < 1)$.

Cependant, dans ce cas, le volume du travail accompli dépasse de beaucoup le volume des calculs liés directement au résultat cherché (dans un carré avec n-intervalles le calcul s'effectue sur n^2 nœuds, alors que, par exemple, pour la courbe d'une fonction donnée ayant une seule valeur pour une des coordonnées, le nombre de paires de sommets voisins des rectangles « courants » ne dépasse pas n). Nous citerons deux méthodes pour résoudre le problème posé, en excluant la possibilité de calculer les valeurs de la fonction $\varphi(x, y)$ à chaque nœud du quadrillage.

1) Supposons que l'équation (4) détermine y comme étant une fonction de x sur l'intervalle (a, b). La courbe (4) détermine deux régions dans lesquelles la fonction $z = \varphi(x, y)$ a des signes opposés. Pour faire cette détermination, il faut calculer ce qui est au-dessus de la courbe (5) ($z < 0$), et au-dessous ($z > 0$).

L'ensemble de tous les nœuds des rectangles « courants » peut être entièrement reconstitué, si l'on détermine sur chacune des verticales un seul nœud du rectangle « courant » alternativement négatif ou positif (Fig. 16).

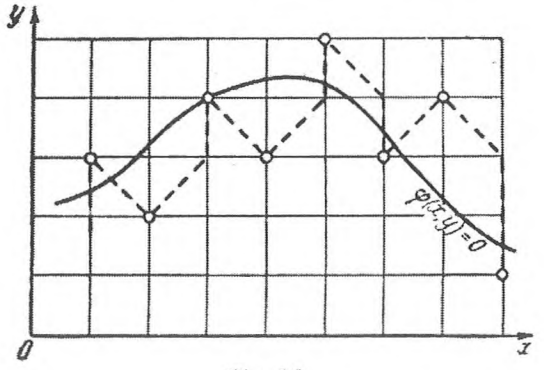


Fig. 16.

Pour distinguer les nœuds correspondants des rectangles « courants », faisons le schéma de calcul suivant : calculons les valeurs de la fonction $\varphi(x, y)$ à l'un des nœuds du rectangle et considérons l'intervalle « initial » Δy_0 le long de l'axe Oy comme positif si le nœud choisi est positif et vice versa.

Les calculs des valeurs de la fonction $\varphi(x, y)$, qui vont suivre, se feront aux sommets où se trouve un point qui se déplace dans le système de nœuds conformément aux règles de *a*) à *c*) :

a) Chaque pas successif Δx_{n+1} d'un point qui se déplace le long de l'axe Ox est égal à zéro si les signes du n-ième nœud donné et du sens précédent du déplacement le long de l'axe Oy coïncident ; dans le cas contraire ce pas Δx_{n+1} est égal à Δx.

b) Le pas Δy_{n+1} le long de l'axe Oy est égal au pas précédent si le pas le long de l'axe Ox est égal à zéro ; dans le cas contraire, ce pas a un sens inverse du précédent et lui est égal en grandeur.

c) Lorsque la limite $y = y_{\max}$ est atteinte par le point et que le signe du nœud ne change pas, le point se déplace vers le nœud $(x + \Delta x, y)$; Δy dans ce cas conserve son signe.

d) Lorsque la verticale $x = x_{\max}$ est coupée, les calculs s'arrêtent.

e) Les coordonnées de chaque nœud dont le pas le long de l'axe Ox n'est pas égal à zéro sont fixées.

Il est clair qu'en partant d'un des nœuds du rectangle examiné et en continuant les calculs conformément aux règles de *a*) à *e*), on fixe l'un après l'autre sur chaque verticale les nœuds du rectangle « courant ».

Pour établir le programme nous allons analyser les conditions dans lesquelles se fait le déplacement.

Conformément aux règles de *a*) à *e*), le déplacement d'un point sur le n-ième pas doit être déterminé par le signe du sens du déplacement précédent le long de l'axe Oy et par le signe de la fonction $\varphi(x, y)$ au n-ième point $s_n = \text{sign } \varphi(x_n, y_n)$. Pour accomplir le déplacement sur le n-ième pas il faut déterminer les grandeurs Δx et Δy selon les valeurs de sign Δy_{n-1} et s_n en conformité avec les règles de *a*) à *e*).

Examinons le tableau des variantes possibles du déplacement :

sign Δy_{n-1}	s_n	Δx_n	Δy_n	$\Delta y_{n-1} \cdot s_n$
+	+	0	Δy_{n-1}	+
−	−	0	Δy_{n-1}	+
+	−	Δx	$-\Delta y_{n-1}$	−
−	+	Δx	$-\Delta y_{n-1}$	−

Ce tableau montre que les valeurs Δx et Δy et aussi la fixation des coordonnées des nœuds des rectangles « courants » sont choisies par rapport au signe du produit $d_n = \Delta y_{n-1} \cdot s_n$.

Répartissons les données dans les cellules de l'organe mémoire :

α_1	α_2	α_3	α_4	ω_1	ω_2	ω_3	ω_4	β_1
Δx_0	Δy_0	$x_{\max}$	$y_{\max}$	$x_0(x_n)$	$y_0(y_n)$	d_n	s_n	$y_{\min}$

Le tableau 7 représente l'organigramme.

Remarquons que, conformément à la condition qui concerne les calculs sur un programme donné, il faut déterminer préalablement le sens du déplacement nul le long de l'axe Oy, c'est-à-dire le signe de $\varphi(x_0, y_0)$. Cette détermination préalable peut être facilement introduite dans le programme (cf. § 4).

2) La construction de la courbe d'après la méthode précédente permet de déterminer en un seul processus une branche de la courbe seulement. C'est pourquoi il est normal d'étudier un procédé qui permette de déterminer la courbe d'une fonction arbitraire suffisamment régulière.

Supposons que la courbe $\varphi(x, y) = 0$ soit continue, n'ait pas de points d'intersection avec elle-même et divise le plan en deux régions liées dans lesquelles la fonction $z = \varphi(x, y)$ ait des signes contraires.

Dans le plan xOy construisons un quadrillage qui fasse un angle de 45° par rapport aux axes de coordonnées. D'après les conditions posées, il s'ensuit que lorsque h est assez petit le périmètre de chaque carré a seulement deux points d'intersection avec la courbe (ces points peuvent coïncider dans un nœud du réseau).

Examinons les règles suivantes qui concernent le déplacement sur les côtés des rectangles « courants » du réseau.

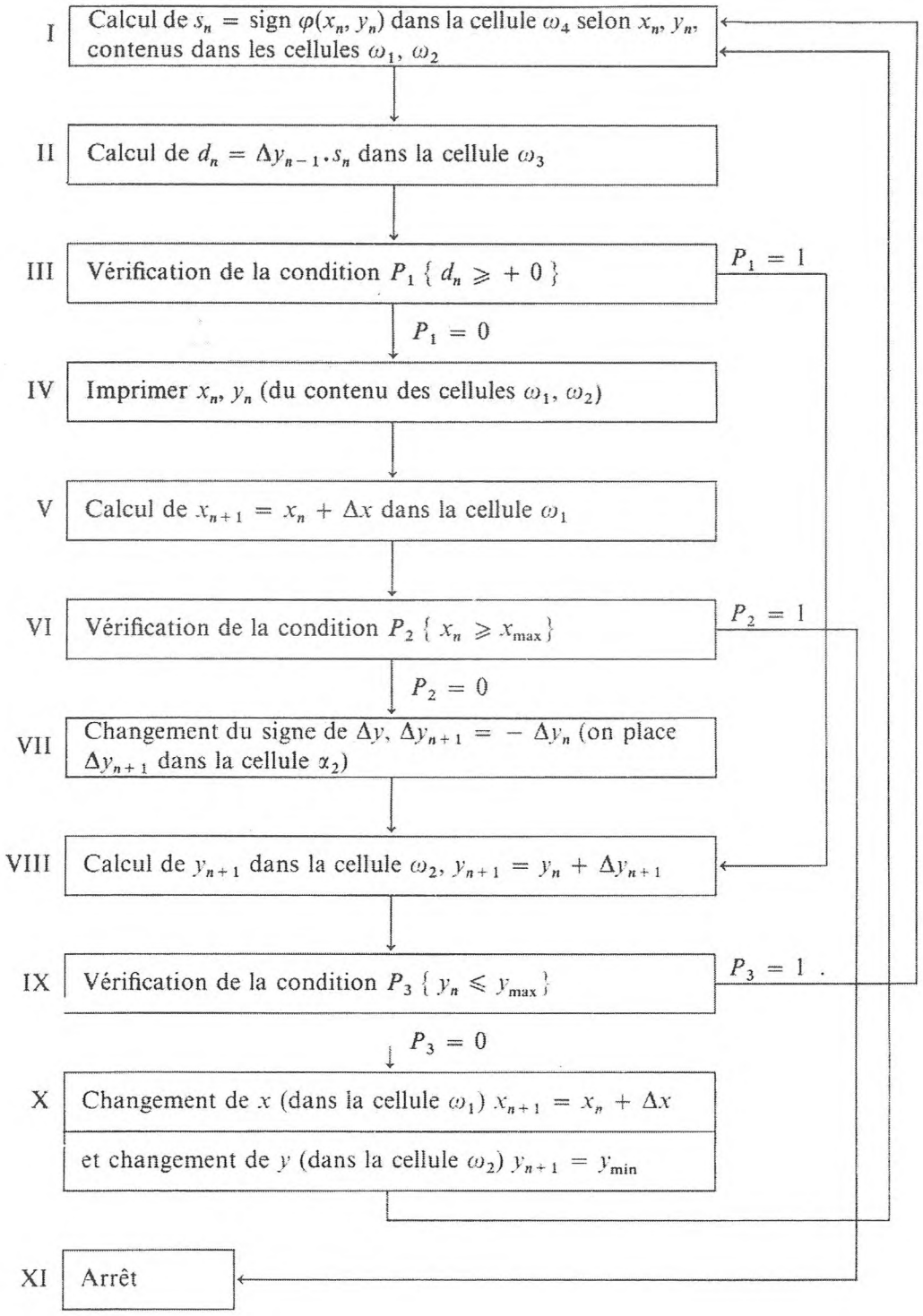
a) Le pas initial est le parcours arbitraire d'un côté d'un carré du réseau avec des nœuds de valeurs différentes. La direction du déplacement sur chaque pas successif varie de 90° à droite ou à gauche par rapport à la direction précédente.

b) Si au pas précédent le point a coïncidé avec un nœud ayant le signe +, il tourne à droite.

c) Si au pas précédent le point a traversé un nœud ayant le signe −, il tourne à gauche.

Il est facile de comprendre que, lorsqu'il se déplace sur les côtés d'un carré « courant », le point traverse obligatoirement le côté d'un carré « courant » voisin. Maintenant, remarquons qu'en vertu de la cohérence des parties qui composent le plan d'une courbe donnée pour un h assez petit, deux nœuds d'une des parties peuvent être unis par une ligne brisée passant par les côtés

Tableau 7


I
Calcul de s_n = sign $\varphi(x_n, y_n)$ dans la cellule ω_4 selon x_n, y_n, contenus dans les cellules ω_1, ω_2
II
Calcul de $d_n = \Delta y_{n-1} . s_n$ dans la cellule ω_3
III
Vérification de la condition P_1 { $d_n \geqslant +0$ }
$P_1 = 1$
$P_1 = 0$
IV
Imprimer x_n, y_n (du contenu des cellules ω_1, ω_2)
V
Calcul de $x_{n+1} = x_n + \Delta x$ dans la cellule ω_1
VI
Vérification de la condition P_2 { $x_n \geqslant x_{max}$ }
$P_2 = 1$
$P_2 = 0$
VII
Changement du signe de Δy, $\Delta y_{n+1} = -\Delta y_n$ (on place Δy_{n+1} dans la cellule α_2)
VIII
Calcul de y_{n+1} dans la cellule ω_2, $y_{n+1} = y_n + \Delta y_{n+1}$
IX
Vérification de la condition P_3 { $y_n \leqslant y_{max}$ }
$P_3 = 1$.
$P_3 = 0$
X
Changement de x (dans la cellule ω_1) $x_{n+1} = x_n + \Delta x$
et changement de y (dans la cellule ω_2) $y_{n+1} = y_{min}$
XI
Arrêt

des carrés du réseau qui se trouvent dans cette partie. Nous dirons qu'un tel ensemble de nœuds de même signe est *lié*.

On peut démontrer que tout ensemble lié, ayant des nœuds négatifs dont le complément est aussi lié, peut être décrit par un déplacement dans les conditions de *a*) à *c*).

Pour programmer un déplacement autour de l'ensemble des sommets négatifs, nous ferons l'analyse logique des conditions de *a*) à *c*).

Conformément aux conditions de *a*) à *c*) le déplacement des points au n-ième pas est déterminé :

1) par le sens précédent du déplacement le long de Ox, sign Δx_{n-1} ;
2) par le sens précédent du déplacement le long de Oy, sign Δy_{n-1} ;
3) par le signe de la fonction $\varphi(x, y)$ au n-ième point, s_n.

Pour déterminer le déplacement sur le n-ième pas, il est indispensable de déterminer les grandeurs Δx_n et Δy_n d'après les valeurs de sign Δx_{n-1}, sign Δy_{n-1}, s_n.

Examinons le tableau des variantes possibles du déplacement (cf. tableau 8).

Tableau 8

sign Δx_{n-1}	sign Δy_{n-1}	s_n	sign Δx_n	sign Δy_n	sign Δx_{n-1} · .sign $\Delta y_{n-1} \cdot s_n$
+	+	+	+	−	+
+	−	−	+	+	+
−	+	−	−	−	+
−	−	+	−	+	+
+	+	−	−	+	−
+	−	+	−	−	−
−	+	+	+	+	−
−	−	−	+	−	−

D'après le tableau 8, nous obtenons les formules suivantes :

$$\text{sign } \Delta x_n = (\text{sign } \Delta x_{n-1} \cdot \text{sign } \Delta y_{n-1} \cdot s_n) \text{ sign } \Delta x_{n-1} = \text{sign } \Delta y_{n-1} \cdot s_n ;$$

$$\text{sign } \Delta y_n = -(\text{sign } \Delta x_{n-1} \cdot \text{sign } \Delta y_{n-1} \cdot s_n) \text{ sign } \Delta y_{n-1} = -\text{sign } \Delta x_{n-1} \cdot s_n .$$

Pour établir le programme de calcul de Δx_n et Δy_n d'après ces formules, nous notons qu'à chaque pas le signe d'une seule grandeur Δx ou Δy change

$$\Delta y_{n+1} = -\Delta y_n , \quad \Delta x_{n+1} = \Delta x_n$$

si sign $\Delta x_{n-1} \cdot$sign $\Delta y_{n-1} \cdot s_n$ a le signe +, et

$$\Delta y_{n+1} = \Delta y_n , \quad \Delta x_{n+1} = -\Delta x_n$$

dans le cas contraire.

Traitons maintenant seulement la partie du programme qui effectue les calculs de Δx_n et Δy_n

Nombres

α_1	Δx_{n-1}
α_2	Δy_{n-1}
α_3	s_n
α_4	sign $\Delta x_{n-1} \cdot s_n$
γ_1	-0

Instructions

$N+1$	$\times$	α_1	α_2	α_4	$\Delta x_{n-1} \cdot \Delta y_{n-1}$
$N+2$	$\times$	α_3	α_4	α_4	
$N+3$	$\leqslant$	α_4	γ_1	$N+6$	
$N+4$	$-$		α_1	α_1	
$N+5$	$-$		α_2	α_2	
$N+6$	$-$		α_1	α_1	

Le programme doit être complété par des instructions qui permettent d'obtenir le premier carré « courant » (par exemple selon la méthode indiquée au § 1), par des instructions qui effectuent le calcul de s_n, l'impression des coordonnées des sommets des rectangles « courants », lorsqu'il y a intersection de la courbe, et par des commandes déterminant l'arrêt des calculs. Nous laissons aux lecteurs le soin de faire le schéma opératoriel.

2. *Résolution d'un problème aux limites concernant une équation aux différences de type parabolique.*

La méthode optimale de contrôle statistique de réception consiste en ce qui suit : on a un domaine de contrôle (*) (Fig. 17) ; si le résultat du contrôle est valable, un segment unitaire est reporté à droite, s'il ne l'est pas, en haut et à droite, etc.

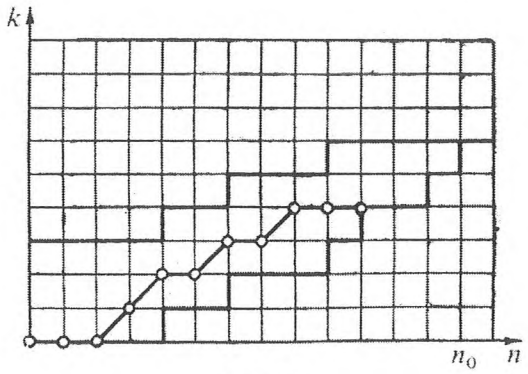


Fig. 17.

(*) Cf. la réalisation du problème dans l'article de N. S. Mikhalevitch « Résolutions successives de Baeyer et méthodes optimales pour le contrôle statistique de réception ». *Théorie des probabilités et ses applications.* Tome 1, 4e édition, 1956, 437-465.

Si l'on atteint la frontière supérieure du domaine, cela signifie la nécessité de refuser tout le calcul, si l'on atteint la frontière inférieure, de l'accepter. Si l'on se trouve parmi les points intérieurs du domaine les essais continuent.

La frontière du domaine de réception a les caractéristiques géométriques suivantes :

1) Elle est composée des graphes de deux fonctions en escalier monotones non décroissantes (qui sont les limites inférieures et supérieures).
2) Les sauts de ces deux fonctions ne dépassent pas l'unité.
3) Il n'y a pas de points intérieurs au domaine à droite de la verticale n_0.
4) Si $n < n_0$, sur toute la verticale il y a un point intérieur au domaine.

Les calculs des tableaux pour les méthodes optimales de contrôle statistique de réception permettent de résoudre l'équation hétérogène aux différences suivante :

$$\rho(k, n) = M(k, n)\,\rho(k + 1, n + 1) + (1 - M(k, n))\,\rho(k, n + 1) + c \quad (6)$$

avec une frontière inconnue, et de déterminer cette frontière par la condition que la solution $\rho(k, n)$ de l'équation (6) coïncide avec les solutions connues de l'équation aux différences correspondante :

$$\rho(k, n) = M(k, n) \text{ sur la limite inférieure du domaine}$$

$$\rho(k, n) = p_0 \text{ sur la limite supérieure du domaine .}$$

Nous considérons comme connues les valeurs de $M(k, n)$, lorsque k et n sont donnés, ainsi que $k_{\max}$ et n_0.

Achevons de déterminer pour les points du rectangle Π ($0 \leqslant n \leqslant n_0$; $0 \leqslant k \leqslant k_{\max}$) la fonction $\rho(k, n)$, déterminée aux points intérieurs du domaine cherché et sur sa frontière en supposant :

$$\rho(k, n) = M(k, n)$$

pour les points « inférieurs » situés au-dessous de la frontière inférieure, et

$$\rho(k, n) = p_0$$

pour les points « supérieurs » situés dans le rectangle au-dessus de la frontière supérieure.

Examinons la fonction $\rho_1(k, n)$ satisfaisant, pour tous les points du rectangle Π, à l'équation :

$$\rho_1(k, n) = M(k, n)\,\rho(k + 1, n + 1) + (1 - M(k, n))\,\rho(k, n + 1) + c\,,$$

et désignons par $m(k, n)$ la grandeur

$$m(k, n) = \min\,(M(k, n), p_0)\,. \quad (7)$$

Il est alors évident que pour tous les points du rectangle Π

$$\rho(k, n) = \min(m(k, n), \rho_1(k, n)) . \tag{8}$$

Dans ce cas, on a les relations :

a) si $\rho = \rho_1 < m$ le point est à l'intérieur
b) si $\rho = m$, le point est à l'extérieur, et de plus
c) si $\rho = M < \rho_0$, le point est au-dessous
d) et si $\rho = \rho_0$, le point est au-dessus.

Ainsi, si les conditions *a*) à *d*) sont satisfaites, le problème consiste à déterminer les coordonnées (k, n) des points intérieurs et les valeurs de la fonction $\rho(k, n)$ pour eux, à l'aide des relations de récurrence :

1) $M(k, n)$,
2) $\rho_1(k, n) = M(k, n)\, \rho(k + 1, n + 1) + (1 - M(k, n))\, \rho(k, n + 1) + c$,
3) $m(k, n) = \min(M(k, n), \rho_0)$,
4) $\rho(k, n) = \min(m(k, n), \rho_1(k, n))$.

La méthode de calcul la plus naturelle pour accomplir les calculs indiqués d'après ces formules est de procéder de la façon suivante : sur chaque verticale, décrite de bas en haut, on calcule aux points d'ordonnées entières la valeur $\rho(k, n)$ de la fonction, à l'aide de $\rho(k, n + 1)$ et de $\rho(k - 1, n - 1)$, à partir de $k = 0$ et de $n = n_0$ jusqu'au premier point supérieur : on passe ensuite à la verticale située immédiatement à gauche. Au début du calcul, n_0 est la verticale pour laquelle les valeurs de la fonction $\rho(k, n)$ sont données (ou spécialement calculées) et placées dans des cellules spécialement affectées :

$$\alpha + 0, \alpha + 1, \alpha + 2, \ldots, \alpha + k_{\max}.$$

Le nombre de cellules indispensables à la conservation de ρ est déterminé par la « hauteur » maximale de la région $(k_{\max} + 1)$.

Le caractère cyclique des calculs est assuré par la mise en place des valeurs calculées de la fonction $\rho(k, n)$ dans la cellule $\alpha + k$, dans laquelle se trouve la valeur de la grandeur $\rho(k, n + 1)$ déjà utilisée au cours des calculs.

Les calculs se terminent sur la verticale nulle $(n = 0)$.

La catégorie de points à laquelle appartient le point (k, n) est déterminée en chaque point sur la base des conditions de *a*) à *d*) :

si le point (k, n) est inférieur, nous conserverons la valeur $\rho(k, n)$ pour les calculs sur $n - 1$ verticales et nous passerons aux calculs au point $(k + 1, n)$ (à un « nouveau point ») ;

si le point (k, n) est intérieur, nous sortirons (imprimerons) ses coordonnées et la valeur de la fonction $\rho(k, n)$, laquelle est en outre conservée, et nous passerons aux calculs au point $(k + 1, n)$ (à un « nouveau point ») ;

si le point (k, n) est supérieur, nous passerons aux calculs au point $(0, n - 1)$, à une « nouvelle verticale » ; la valeur de la fonction $\rho(k, n)$ est alors égale à p_0 et une mémorisation spéciale n'est pas nécessaire.

3. *Economie de cellules de la mémoire et de temps de calcul.*

Cependant, la méthode de calcul employée exige une mémorisation des valeurs de la fonction $\rho(k, n)$ à tous les points de chaque verticale jusqu'aux premiers points supérieurs et englobe le calcul sur le domaine des points inférieurs qui ne nous intéresse pas et qui est situé entre l'axe des abscisses et la limite inférieure du domaine cherché.

En utilisant les caractéristiques géométriques de la frontière d'un domaine on peut proposer la méthode de calcul suivante qui est plus économique.

Le calcul se fait sur chaque n-ième verticale en commençant par le k-ième point (k^*, n) où k^* est l'ordonnée des sommets des points inférieurs sur la verticale $n + 1$ et se continue jusqu'au premier point supérieur, après quoi l'on passe à la verticale $n - 1$. Les calculs commencent au point $(k_{\max}, n_0)$ et se terminent au point (0, 0).

Dans ce cas, pour les calculs d'après les formules récurrentes 1 à 4, il est indispensable de conserver la valeur de la fonction $\rho(k, n)$ seulement aux points intérieurs. Le programme du calcul de la fonction $\rho_1(k, n)$ au point (k, n) se construit dans les cellules standards dans lesquelles on envoie, à l'aide des opérateurs d'envoi Z_1 et Z_2, la valeur $\rho(k + 1, n + 1)$ calculée sur la verticale précédente et la valeur $\rho(k, n + 1)$ utilisée au cours des calculs au point précédent ; cette dernière se calcule au point initial (k^*, n) de chaque verticale par la formule :

$$\rho(k^*, n + 1) = M(k^*, n + 1),$$

puisque le point $(k^*, n + 1)$ est inférieur.

La mémorisation de la fonction ρ sur chaque verticale est faite en commençant par le premier point intérieur, dans la cellule $\alpha + 0$, etc.

La quantité de cellules de mémorisation des valeurs $\rho(k, n)$ est désormais égale à la « largeur » maximale de la région.

4. *Réduction du volume des données à la sortie.*

Au cours des calculs sur C. A. N., sans rien perdre de l'information indispensable, il faut faire en sorte que les données soient réduites de la façon suivante :

1) s'il n'y a pas de points intérieurs sur la verticale, imprimer le signe conventionnel « vertical » ;

2) n'imprimer l'ordonnée que du premier point intérieur ;

3) sur chaque verticale, aux points intérieurs, la valeur est imprimée par ordre croissant des ordonnées ;

4) la sortie de l'information sur chaque verticale est successivement réalisée à partir de la verticale n_0 jusqu'à la verticale nulle.

La façon dont il faut satisfaire aux exigences 3)-4) est évidente. C'est pourquoi nous nous arrêterons seulement sur les exigences 1)-2).

Pour satisfaire à la première exigence, il est indispensable d'élaborer un symbole qui permette de distinguer les verticales qui ont des points intérieurs de celles qui n'en ont pas. Pour la seconde alternative, il faut un symbole pour différencier le premier point intérieur sur une verticale donnée des autres points intérieurs.

Puisque la présence sur la verticale de points intérieurs entraîne la rencontre avec un premier point intérieur et vice versa, il est suffisant pour réaliser le transfert de commande conformément à 1) et à 2) de placer « -0 » dans une certaine cellule β au début de chaque verticale et de conserver cette valeur inchangée jusqu'à la rencontre avec le premier point intérieur, après quoi, on y place « $+0$ ». Cette propriété des points qui nous intéresse se déterminera alors par le signe de la cellule β :

— si le point donné est le premier point intérieur, β contient -0 ;
— s'il ne l'est pas, β contient $+0$;
— si la verticale traversée ne contient pas de points intérieurs, β contient -0 ;
— si la verticale traversée contient des points intérieurs, β contient $+0$.

Introduisons les désignations :

I. Opérateurs de calcul :

$\mathbf{A}_1$, calcul de la fonction M aux points $(k^*, n+1)$,
$\mathbf{A}_2$, celui de la fonction M au point (k, n),
$\mathbf{A}_3$, celui des fonctions $\rho_1(k, n)$, $m(k, n)$, $\rho(k, n)$,
$\mathbf{A}_4$, préparation au calcul du point suivant $(k+1)$,
$\mathbf{A}_5$, préparation de k^* pour le début du calcul sur la $(n-1)$-ième verticale qui suit.

II. Opérateurs d'envoi :

$\mathbf{Z}_1$, envoi de $\rho(k+1, n+1)$ dans la cellule standard (pour l'opérateur $\mathbf{A}_3$, c'est le calcul de $\rho_1(k, n)$),
$\mathbf{Z}_2$, mémorisation de $\rho(k, n)$ dans une cellule standard pour les calculs sur la verticale suivante,
$\mathbf{Z}_3$, envoi de « -0 » dans la cellule β,
$\mathbf{Z}_4$, envoi de $\rho(k, n+1)$ dans une cellule standard (pour l'opérateur $\mathbf{A}_3$).

III. Opérateurs de substitution d'adresse :

$\mathbf{\Pi}_1$, substitution d'adresse de l'opérateur Z_1,
$\mathbf{\Pi}_2$, substitution d'adresse de l'opérateur Z_2.

IV. Conditions logiques :

1. Répartition des points en points intérieurs et extérieurs :

$$P_1 = \begin{cases} 1, \text{ si le point est extérieur } (\rho = m) \text{ ; la commande se transmet à l'opérateur } \mathbf{P}_3 \text{ ;} \\ 0, \text{ si le point est intérieur ; la commande se transmet à l'opérateur } \mathbf{P}_2. \end{cases}$$

2. Répartition des points intérieurs en premier point et points suivants :

$$P_2 = \begin{cases} 0, \text{ si le point intérieur est le premier ; la commande est transmise à l'opérateur } \mathbf{B}_1, \beta = -0 ; \\ 1, \text{ sinon ; la commande est transmise à l'opérateur } \mathbf{B}_2, \beta = +0. \end{cases}$$

3. Répartition des points extérieurs en points supérieurs et inférieurs :

$$P_3 = \begin{cases} 0, \text{ si le point extérieur est supérieur, la commande est transmise à l'opérateur } \mathbf{P}_4 ; \\ 1, \text{ sinon ; la commande est transmise à l'opérateur } \mathbf{A}_4. \end{cases}$$

4. Répartition des verticales en verticales qui ont des points intérieurs et verticales qui n'en ont pas.

$$P_4 = \begin{cases} 0, \text{ s'il n'y a pas de points intérieurs sur la verticale ; la commande est transmise à l'opérateur } \mathbf{B}_3, \beta = -0 ; \\ 1, \text{ sinon ; la commande est transmise à l'opérateur } \mathbf{P}_5, \beta = +0. \end{cases}$$

5. Détermination de la fin des calculs.

$$P_5 = \begin{cases} 0, \text{ si } n = 0 ; \text{ la commande est transmise à l'instruction « arrêt » ;} \\ 1, \text{ si } n > 0 ; \text{ la commande est transmise à l'opérateur } \mathbf{\Phi}_2. \end{cases}$$

V. Opérateurs de formation :
$\mathbf{\Phi}_1$, formation des données initiales,
$\mathbf{\Phi}_2$, formation des opérateurs $\mathbf{Z}_2$, $\mathbf{Z}_4$ et β.

VI. Sortie des données :
$\mathbf{B}_1$, imprimer k,
$\mathbf{B}_2$, imprimer ρ,
$\mathbf{B}_3$, imprimer le signe conventionnel « verticale sans points intérieurs ».

Lorsqu'on l'écrit avec les désignations adoptées, le schéma du programme a la forme :

$$\mathbf{\Phi}_1\,\mathbf{\Phi}_2\,\mathbf{A}_1\,\mathbf{A}_2\,\mathbf{Z}_1\,\mathbf{A}_3\,\mathbf{Z}_2\,\mathbf{P}_1\ \underset{\mathbf{P}_3}{\downarrow}\ \mathbf{P}_2\ \underset{\mathbf{B}_2}{\downarrow}\ \mathbf{B}_1\,\mathbf{B}_2\,\mathbf{Z}_3\,\mathbf{\Pi}_1\ \underbrace{\mathbf{\Pi}_2\,\mathbf{Z}_4\,\mathbf{A}_4\,\mathbf{P}^0}_{(2)}\ \underset{\mathbf{A}_2}{\downarrow}\ *$$

$$*\ \mathbf{P}_3\ \underset{\mathbf{A}_4}{\downarrow}\ \mathbf{P}_4\ \underset{\mathbf{P}_5}{\downarrow}\ \mathbf{B}_3\,\mathbf{P}_5\ \underset{\mathbf{\Phi}_2}{\downarrow}\ \text{« } ARR \text{ »}\,\mathbf{A}_5\ \underbrace{\mathbf{\Pi}_2\,\mathbf{Z}_4\,\mathbf{A}_4\,\mathbf{P}^0}_{(1)}\ \underset{\mathbf{A}_2}{\downarrow}$$

où * indique que la formule continue à la ligne suivante sans coupure.

Dans l'examen du schéma du programme, il faut remarquer que le groupe d'opérateurs (1) mis en évidence à la fin, s'achevant par l'opérateur de transfert inconditionnel à l'opérateur $\mathbf{A}_2$, coïncide avec le groupe d'opérateurs (2), se terminant aussi par l'opérateur de transfert inconditionnel à l'opérateur $\mathbf{A}_2$; c'est pourquoi il faut simplifier le schéma du programme en plaçant après

l'opérateur $\mathbf{A}_5$ l'opérateur de transfert inconditionnel à l'opérateur $\mathbf{\Pi}_2$. Le schéma du programme prend la forme :

$$\mathbf{\Phi}_1 \mathbf{\Phi}_2 \mathbf{A}_1 \mathbf{A}_2 \mathbf{Z}_1 \mathbf{A}_3 \mathbf{Z}_2 \mathbf{P}_1 \underset{\mathbf{P}_3}{\downarrow} \mathbf{P}_2 \underset{\mathbf{B}_2}{\downarrow} \mathbf{B}_1 \mathbf{B}_2 \mathbf{Z}_3 \mathbf{\Pi}_1 \mathbf{\Pi}_2 \mathbf{Z}_4 \mathbf{A}_4 \mathbf{P}^0 \underset{\mathbf{A}_2}{\downarrow} \mathbf{P}_3 \underset{\mathbf{A}_5}{\downarrow} *$$

$$* \mathbf{P}_4 \underset{\mathbf{P}_5}{\downarrow} \mathbf{B}_3 \mathbf{P}_5 \underset{\mathbf{\Phi}_2}{\downarrow} \text{« } ARR. \text{ »} \mathbf{A}_5 \mathbf{P}^0 \underset{\mathbf{\Pi}_2}{\downarrow}$$

4. CONDITIONS LOGIQUES DANS LE SCHÉMA DES PROGRAMMES

Dans un programme, le changement de place d'une condition logique par rapport à l'opérateur dont cette condition définit le travail est important. Ainsi, si la condition logique se trouve après l'opérateur, ce dernier agira tout au plus une fois ; mais si la condition logique se trouve avant l'opérateur, celui-ci ne peut pas s'appliquer.

Examinons un schéma qui a un branchement. Supposons que l'on ait à résoudre une équation différentielle du premier degré du type :

$$\frac{dy}{dx} = \varphi(x) ,$$

$$\varphi(x) = \begin{cases} f_1(x) , & \text{si } x \leqslant a \\ f_2(x) , & \text{si } x > a \end{cases} \qquad (y(x_0) = y_0 , \quad x_0 \leqslant x < X) .$$

Le schéma du calcul est :

1) Calcul de $\varphi(x_n)$ d'après la première formule ($f_1(x_n)$), ou d'après la seconde ($f_2(x_n)$) ;

2) $y_{n+1} = y_n + h\varphi(x_n) ; \qquad x_{n+1} = x_n + h .$

Pour réaliser ce schéma de calcul, nous devons introduire la condition logique P_1 qui assurera le travail de l'opérateur de calcul $\mathbf{A}_1$ (calcul de $f_1(x_n)$), ou celui de l'opérateur de calcul $\mathbf{A}_2$ (calcul de $f_2(x_n)$).

Introduisons la condition logique $P_1(x)$ qui prend deux valeurs : « oui » désignée par le nombre 1 et « non » désignée par le nombre 0, selon la règle :

$$P_1(x) = \begin{cases} 0 , & x \leqslant a , \\ 1 , & x > a . \end{cases}$$

Le schéma $\mathbf{P}_1(x)\ \mathbf{A}_1 \underset{\mathbf{P}_1}{\downarrow}$ assurera alors le travail de l'opérateur $\mathbf{A}_1$ conformément au schéma de calcul. Pour assurer le travail de l'opérateur $\mathbf{A}_2$ confor-

mément à ce schéma, introduisons la condition logique $P_2(x)$ d'après la règle :

$$P_2(x) = \begin{cases} 0, & x > a, \\ 1, & x \leqslant a. \end{cases}$$

La condition logique $P_2(x)$ coïncide avec la négation de la condition logique P_1 que nous écrirons ainsi : $P_2(x) = \overline{P}_1(x)$. Nous noterons $\overline{1}$ la flèche de la condition logique $\overline{P}_1$.

Ainsi le schéma du programme du problème envisagé aura la forme :

$$\begin{array}{cccccccccccc} 0 & & \overline{\mathbf{P}}_1 & & 1 & & \mathbf{B} & & \overline{1} & & \mathbf{P}_1 & \\ \downarrow & \mathbf{P}_1 & \uparrow & \mathbf{A}_1 & \downarrow & \overline{\mathbf{P}}_1 & \uparrow & \mathbf{A}_2 & \downarrow & \mathbf{BP}_0 & \uparrow & !. \end{array}$$

où **B** est l'opérateur du calcul de x_{n+1} et de y_{n+1} et où P_0 est la condition logique, déterminant la possibilité de répéter N fois un cycle :

$$\left(N = \frac{X - x_0}{h}\right).$$

Le schéma du programme peut être transformé. Il est possible de placer la flèche $\overset{1}{\downarrow}$ après la condition logique $\overline{P}_1$, puisque l'on aura toujours $\overline{P}_1 = 1$, lorsque la valeur de la condition logique $P_1 = 0$. Mais il faut alors changer la condition logique $\overline{P}_1$ par la condition logique identique P_2^0, c'est-à-dire $P_2^0 \equiv 1$.

Ainsi le schéma du programme prend la forme :

$$\begin{array}{cccccccccccc} 0 & & \mathbf{A}_2 & & & \mathbf{B} & 1 & & 2 & & \mathbf{P}_1 & \\ \downarrow & \mathbf{P}_1 & \uparrow & \mathbf{A}_1 & \mathbf{P}_2^0 & \downarrow & \uparrow & \mathbf{A}_2 & \downarrow & \mathbf{BP}_0 & \uparrow & !. \end{array}$$

L'établissement du programme selon le schéma donné est simple :

Programme 5.

Nombres

α_1	x_0	x_n
α_2	y_0	y_n
α_3	a	
α_4	h	

Instructions

$k+1$	$\leqslant$	α_1	α_3	$k+4$	$\mathbf{P}_1$
$k+2$	$f_1(\alpha_1)$	–	–	ω	$\mathbf{A}_1$
$k+3$	$\lvert \leqslant \rvert$	–	–	$k+5$	$\mathbf{P}_2$
$k+4$	$f_2(\alpha_1)$	–	–	ω	$\mathbf{A}_2$

Nombres

α_5	X
ω	de travail

Instructions

$k+5$	$\times$	α_4	ω	ω	**B**
$k+6$	$+$	ω	α_2	α_2	
$k+7$	$+$	α_1	α_4	α_1	
$k+8$	$\leqslant$	α_1	α_5	$k+1$	$\mathbf{P}_0$
$k+9$	*ARR*				

Les instructions $k+2$ et $k+4$ sont ici des instructions symboliques pour calculer respectivement les fonctions f_1 et f_2.

Examinons la résolution d'une équation différentielle du premier degré en machine à virgule fixe :

$$\frac{dy}{dx} = \frac{\varphi(x, y)}{\psi(x, y)},$$

avec comme condition initiale $y(x_0) = y_0 \quad (x_0 \leqslant x \leqslant X)$. Adoptons le schéma de calcul suivant :

I. Si $\varphi_k = \varphi(x_k, y_k) > \psi_k = \psi(x_k, y_k)$, les calculs mèneront aux formules :

$$A^{\mathrm{I}} \quad y_{k+1} = y_k + h, \quad x_{k+1} = x_k + 2h\frac{\psi_k}{2\varphi_k}.$$

II. Si $\varphi_k \leqslant \psi_k$, nous calculerons selon les formules :

$$A^{\mathrm{II}} \quad x_{k+1} = x_k + h, \quad y_{k+1} = y_k + 2h\frac{\varphi_k}{2\psi_k}.$$

Introduisons la condition logique P_1 :

$$P_1 = \begin{cases} 0, & \varphi_k \leqslant \psi_k, \\ 1, & \varphi_k > \psi_k. \end{cases}$$

Introduisons les désignations suivantes : $\mathbf{\Phi}$ est l'opérateur de calcul $\varphi_k = \varphi(x_k, y_k)$; $\mathbf{\Psi}$ est l'opérateur de calcul $\psi = \psi(x_k, y_k)$; P_2 est la condition logique réalisant le processus cyclique selon le paramètre k sur l'intervalle $x_0 \leqslant x \leqslant X$:

$$P_2 = \begin{cases} 0, & x \leqslant X, \\ 1, & x > X. \end{cases}$$

Nous obtenons un schéma de programme analogue au précédent :

$$\begin{array}{cccccccccccc} 2 & & \mathbf{A}^{\mathrm{II}} & & \mathbf{P}_2 & 1 & & 3 & & \Phi & & \\ \downarrow & \Phi\Psi\mathbf{P}_1 & \uparrow & \mathbf{A}^{\mathrm{I}}\,\mathbf{P}_3^0 & \uparrow & \downarrow & \mathbf{A}^{\mathrm{II}} & \downarrow & \mathbf{P}_2 & \uparrow & ! & . \end{array}$$

Nous laissons le soin au lecteur d'en établir le programme.

Il faut faire attention à ce que dans le problème traité le travail de la condition logique P_1 soit préparé au cours du calcul d'après le programme. Ainsi les conditions logiques, en utilisant les simples données initiales du problème, traitent un processus de calcul complexe.

Examinons le calcul du déterminant d'une matrice carrée non dégénérée avec choix d'éléments maximaux (cf. chap. IV, § 3.3.1).

Si au cours du calcul d'un déterminant (cf. chap. IV, § 3.3.2) les éléments de la diagonale principale se transforment en zéros ou peuvent être petits en valeur absolue, le calcul du déterminant d'après le programme du paragraphe 3 est impossible ou fait perdre beaucoup d'exactitude. Pour éliminer ce défaut, nous déterminerons à chaque ligne l'élément de module maximal et nous transformerons en zéros tous les éléments qui lui sont inférieurs, en soustrayant des lignes suivantes les quantités proportionnelles aux éléments de la ligne donnée.

Introduisons l'opérateur $\mathbf{H}_{kr}$ déterminant $m_{kr} = \max \{ m_{k-1,r}, | a_{kr} | \}$, $m_{0r} = 0$ dans la cellule v_1.

Dans le schéma de sortie du calcul les opérateurs $\mathbf{D}_r$ et $\mathbf{C}_{ir}$ changent. Pour réaliser l'opérateur $\mathbf{D}_r$, il est indispensable :

1) De stocker l'élément de module maximal de la ligne a_{rk_r} dans une certaine cellule fixe v_2.

2) De former le nombre $(-1)^k$ et de stocker $(-1)^k$ dans une certaine cellule v_3.

3) De calculer $d_0 = (-1)^{n(n+1)/2}$. L'opérateur $\mathbf{D}_r$ calcule alors $d_r = d_{r-1}\, a_{rk_r} \cdot (-1)^k\, r$; $d_0 = (-1)^{n(n+1)/2}$; $d_n = D_{\mathrm{A}}$; il est réalisé par les deux instructions :

$D+1$	×	v_0	v_3	v_3
$D+2$	×	v_2	v_3	v_0

d_0 se trouve dans la cellule v_0.

L'opérateur $\mathbf{C}_{ir}$ fait le calcul de

$$C_{ik_r} = \frac{a_{ik_r}}{a_{rk_r}}.$$

Pour le réaliser il est indispensable d'avoir le nombre :

–	k_r	–	–

dans la cellule v_4. Le programme de l'opérateur $\mathbf{C}_{i_r}$ a la forme :

$c+1$	$\oplus$	$c+2$	v_4	$c+2$
$c+2$	:	$a+n+0$	v_2	ρ

Introduisons l'opérateur $\mathbf{U}_{k_r}$, qui stocke a_{rk_r} dans la cellule v_2, $(-1)^{k_r}$ dans la cellule v_3 et

–	k_r	–	–

dans la cellule v_4, et l'opérateur $\mathbf{B}_k$ qui calcule $(-1)^k$ et les nombres

–	k	–	–

respectivement dans les cellules v_5 et v_6.

Le schéma du programme est alors réalisé par le schéma du choix des éléments maximaux et de la préparation du travail des opérateurs $\mathbf{D}_r$ et $\mathbf{C}_{i_r}$:

$$\overset{5}{\downarrow}\ \mathbf{B}_k\, \mathbf{H}_{kr}\, \mathbf{U}_{k_r}\, \mathbf{F}(k)\, \mathbf{P}_5\ \overset{\mathbf{B}}{\uparrow}$$

En définitive, le schéma du programme de calcul du déterminant de la matrice carrée non dégénérée avec choix d'éléments maximaux a l'aspect :

$$\mathbf{\Phi}_1(r) \overset{\mathbf{Z}}{\downarrow} \mathbf{\Phi}_2(k) \overset{5}{\downarrow} \mathbf{B}_k\, \mathbf{H}_{kr}\, \mathbf{U}_{k_r}\, \mathbf{F}_1(k)\, \mathbf{P}_5 \overset{\mathbf{B}}{\uparrow} \mathbf{D}_r \overset{2}{\downarrow} \mathbf{C}_{i_r} \overset{1}{\downarrow} \mathbf{A}_{k\,i_r}\ *$$

$$*\ \mathbf{F}_2(k)\, \mathbf{P}_1 \overset{\mathbf{A}}{\uparrow} \mathbf{F}_3(i, k)\, \mathbf{P}_2 \overset{\mathbf{C}}{\uparrow} \mathbf{F}_4(r, i)\, \mathbf{P}_3 \overset{\mathbf{\Phi}_1}{\uparrow}\ !. \quad (9)$$

Etablissons le programme seulement d'après le schéma complémentaire. Introduisons l'opérateur Q, lequel détermine le module du nombre a qui se trouve dans la cellule α, et envoie ce module dans la cellule β. Supposons qu'il soit possible de réaliser l'opérateur Q par une seule instruction :

| |	α	–	β

Le programme $\mathbf{H}_{kr}$ prend alors la forme

$H+1$	$\| \leqslant \|$	$\alpha+n+1$	v_2	$H+3$		$r.n+k$	–	–
$H+2$	| |	$\alpha+n+1$	–	v_1		$r.n+k$	–	–

La condition logique $P_{\mathbf{H}}$ satisfaite par la première instruction de l'opérateur $\mathbf{H}_{kr}$ prend un sens d'après la règle :

$$P_{\mathbf{H}} = \begin{cases} 0, & m_{kr} < |\, a_{k+1,r}\,| \,, \\ 1, & m_{kr} \geqslant |\, a_{k+1,r}\,| \,. \end{cases}$$

A l'aide de cette condition logique la conservation de l'élément du module maximal de la ligne est assurée. On comprend facilement que cette même condition logique $P_{\mathbf{H}}$ est à utiliser pour conserver a_{rk_r} dans la cellule v_2. De plus, le programme de l'opérateur $\mathbf{U}_r$ peut être entièrement construit en utilisant la condition logique $P_{\mathbf{H}}$, à savoir :

Nombres

v_2	0	a_{rk_r}
v_3	–	$(-1)^{k_r}$
v_4	0	– k_r – –
v_5	-1	$(-1)^k$
v_6	0	– k – –

Instructions

$k+1$	$\| \leqslant \|$	$\alpha+n+1$	v_2	$k+6$		$r.n+k$	–	–
$k+2$	| |	$\alpha+n+1$	–	v_2		$r.n+k$	–	–
$k+3$	+	$\alpha+n+1$	–	v_1		$r.n+k$	–	–
$k+4$	+	v_5	–	v_3				
$k+5$	+	v_6	–	v_4				

L'établissement d'un programme selon le schéma (9) n'offre pas de difficultés (cf. programme 6).

Programme 6.

Nombres

v_0	d_0					
v_1	$-$	a_{rk_r}				
v_2	0	$m_{rk_r}^{k_r}$				
v_3	$-$	$(-1)^{k_r}$				
v_4	$-$	$-$	k_r	$-$	$-$	
v_5	1	$(-1)^k$				
v_6	0	$-$	k	$-$	$-$	
v_7	-1					
v_8	$-1--$					
v_9	$-n--$					

Instructions

B	$k+1$	$\times$	v_5	v_7	v_5		
	$k+2$	$+$	v_6	v_8	v_6		
H	$k+3$	$\lvert \leqslant \rvert$	$\alpha+n+1$	v_2	$k+8$	$r.n+k$	$-$ $-$
	$k+4$	$\lvert \quad \rvert$	$\alpha+n+1$	$-$	v_2	$r.n+k$	$-$ $-$
U	$k+5$	$+$	$\alpha+n+1$	$-$	v_1	$r.n+k$	$-$ $-$
	$k+6$	$+$	v_5	$-$	v_3		
	$k+7$	$+$	v_6	$-$	v_4		
$\mathbf{F}_1(k)$	$k+8$	$\oplus$	$k+3$	v_8	$k+3$		
	$k+9$	$\oplus$	$k+4$	v_8	$k+4$		
	$k+10$	$\oplus$	$k+5$	v_8	$k+5$		
$\mathbf{P}_5$	$k+11$	$\leqslant$	v_6	v_9	$k+1$		

Le programme 27 (Chap. III, § 6, 5) permet de calculer soit sin x, soit cos x, selon l'instruction que l'on emploie pour commencer le travail du programme.

A l'aide de la variable logique on peut établir le programme de calcul de sin x et cos x ayant une seule entrée. Soit la condition P_1 qui prend les valeurs selon la règle :

$$P_2 = \begin{cases} 1, & \text{s'il est nécessaire de calculer sin } x \\ 0, & \text{— — — cos } x. \end{cases}$$

Le schéma du programme :

$$\begin{array}{c} \mathbf{\Phi_C} \quad \mathbf{A\,2} \quad \mathbf{0\,1} \quad \mathbf{A} \\ \mathbf{P_2 \uparrow \Phi_s P_0 \uparrow \downarrow \Phi_C \downarrow \downarrow AP_1 \uparrow !}. \end{array} \tag{10}$$

a un début commun pour le calcul de sin x et de cos x.

Pour réaliser la condition logique P_2, avant chaque appel au sous-programme de calcul de sin x et cos x, nous placerons dans la cellule β le nombre -0 s'il faut calculer sin x, et $+0$ s'il faut calculer cos x. Alors la condition logique P_2 est réalisée par l'instruction :

P_2	$\leqslant$	$-$	β	Φ_C

Le nombre dans la cellule β détermine d'une façon univoque la valeur de la condition logique P_2. Dorénavant, nous appellerons *variable logique* le contenu de la cellule β qui ne prend que deux valeurs et qui détermine la valeur de la condition logique.

EXERCICES

1. Construire la variable logique correspondant à la condition logique P_H du programme dans la cellule de l'organe mémoire.

2. Réunir le calcul de sin x et cos x dans le programme sans employer la condition logique identique P_0.

3. Etablir le programme de calcul du module du nombre se trouvant dans la cellule α, avec mise en place du résultat dans la cellule β, en n'utilisant que les instructions d'opérations arithmétiques et de comparaison.

5. PROGRAMMATION DES OPÉRATEURS LOGIQUES

Les programmes de certains opérateurs logiques consistent en des instructions de transfert de commande distinctes. Enumérons les conditions logiques

dont les programmes représentent les instructions de transfert de commande distinctes, citées paragraphe 3.

Condition logique	Programme
1) $P\{(a) \leqslant (b)\}$	$\leqslant \quad a \quad b \quad A$
2) $P\{\lvert(a)\rvert \leqslant \lvert(b)\rvert\}$	$\lvert\leqslant\rvert \quad a \quad b \quad A$
3) $P\{(a) \neq (b)\}$	$\neq \quad a \quad b \quad A$
4) $P\{(a) = (b)\}$	$= \quad a \quad b \quad A$
5) $P\{(a) \geqslant (b)\}$	$\geqslant \quad a \quad b \quad A$
6) $P\{(a) \geqslant +0\}$	$TCN \quad a \quad A \quad B$

Ici, A est le code de l'opérateur auquel est transmise la commande lorsque la condition logique correspondante est satisfaite, B est le code de l'opérateur auquel elle est transmise si la condition logique n'est pas satisfaite. En outre nous appellerons code de l'opérateur le numéro de son instruction initiale. Le nombre contenu dans l'adresse a est désigné par (a).

Comme on l'a déjà remarqué, on effectue d'ordinaire des opérations de transfert de commande différentes sur des machines différentes. Toutefois, on peut établir le programme de n'importe quel opérateur logique en ne se bornant qu'à l'une des opérations citées. Dans ce but, utilisons par exemple les opérations de transfert de commande d'après un nombre « TCN », et examinons-en quelques exemples.

1) Le programme de la condition logique $P_1\{(a) \leqslant -0\}$, c'est-à-dire la négation de la condition logique $P\{(a) \geqslant +0\}$, a la forme :

TCN	a	B	A

2) La condition logique $P_6\{(\alpha) > -0\}$ est équivalente à la condition $P\{(\alpha) \geqslant +0\}$ et réalisée par le programme

TCN	α	A	B

3) La condition logique $P_2\{(a) \leqslant (b)\}$ est équivalente à la condition

logique $P\{(a) - (b) \leqslant -0\}$, dont le programme a la forme :

N	$-$	a	b	c
$N+1$	TCN	c	B	A

4) La condition logique $P_4\{|(a)| \leqslant |(b)|\}$ est équivalente à $P\{|(a)| \leqslant |(b)| \leqslant -0\}$ dont le programme a la forme :

N	$\|-\|$	a	b	α
$N+1$	TCN	α	B	A

5) La condition logique $P_5\{(a) \neq (b)\}$ est équivalente à la condition logique $P\{(a-b)(b-a) \leqslant -0\}$ et peut être réalisée par le programme :

N	$-$	a	b	α
$N+1$	$-$	b	a	β
$N+2$	$\times$	α	β	α
$N+3$	TCN	α	B	A

6) La condition logique $P_6\{(a) > (b)\}$ est la négation de la condition logique $P\{(a) \leqslant (b)\}$, elle est programmée par les ordres :

$N+1$	$-$	a	b	c
$N+2$	TCN	c	A	B

7) La condition logique $P_7\{(a) = (b)\}$ est équivalente à la négation de la condition $P\{(a) \neq (b)\}$ et réalisée par le programme :

$N+1$	$-$	a	b	α
$N+2$	$-$	b	a	β
$N+3$	$\times$	α	β	α
$N+4$	TCN	α	A	B

Les opérateurs logiques déterminés par la condition logique $P\{(\alpha) \leqslant -0\}$ ou par sa négation $P\{(\alpha) \geqslant +0\}$ où α est donné par une formule arithmétique, seront appelés *opérateurs logiques simples.*

Les programmes des opérateurs logiques simples consistent en des instructions de calcul de la valeur (α) selon une formule arithmétique donnée et s'achèvent par l'instruction de transfert conditionnel de la commande suivant le nombre α à l'opérateur **A**, si la condition logique est satisfaite, sinon à l'opérateur **B**.

Les opérateurs logiques déterminés par une formule logique qui contient des opérations logiques simples seront dits *opérateurs logiques complexes.*

La programmation des opérateurs complexes peut être réalisée par deux méthodes.

La première consiste à établir un programme de calcul de la condition logique complexe qui définit l'opérateur logique correspondant d'après n'importe quelle formule qui contient des conditions logiques simples. Les programmes de calcul des conditions logiques sont construits comme des programmes arithmétiques. Examinons quelques exemples :

8) La condition logique $P_8\{(\alpha_1) \leqslant -0 \text{ ou } (\alpha_2) \leqslant -0\}$ est exprimée par la formule :

$$\alpha = \operatorname{sign}(\alpha_1) \vee \operatorname{sign}(\alpha_2).$$

Le programme est :

$\vee$	α_1	α_2	α
TCN	α	B	A

9) La condition logique $P_9\{(\alpha_1) \geqslant +0 \text{ et } (a) - (b) \leqslant -0\}$ est exprimée par la formule :

$$\alpha = -\operatorname{sign}(\alpha_1) \wedge \operatorname{sign}[(a) - (b)].$$

Le programme est :

$k+1$	—	—	α_1	α
$k+2$	—	a	b	β
$k+3$	$\vee$	α	β	α
$k+4$	*TCN*	α	*B*	*A*

La seconde méthode de programmation est fondée sur l'emploi des trois règles ci-après.

I. *Règle de programmation de la négation d'un opérateur logique* **P**.

Admettons que l'opérateur logique **P** transmette la commande à l'opérateur **A** si la condition logique correspondante est vraie et à l'opérateur **B** dans le cas contraire. Le programme d'un tel opérateur contient une instruction de transfert de commande en fonction d'un nombre aux opérateurs **A** et **B**. Le programme de la négation de cet opérateur ($\overline{\mathbf{P}}$) est obtenu par le programme de l'opérateur initial **P** en substituant, dans les instructions de transfert de commande, au code de l'opérateur **A** (*) le code de l'opérateur **B** et vice versa.

Il est facile de se convaincre que les programmes de la négation des opérateurs logiques qui ont plus d'une instruction de transfert de commande sont obtenus conformément à la règle citée.

Prenons quelques exemples.

10) L'opérateur logique complexe $\mathbf{P}_{10}$ déterminé par le programme

$k+1$	*TCN*	α_1	*A*	$k+2$
$k+2$	*TCN*	α_2	*A*	*B*

transmet l'instruction à l'opérateur **A** à la condition qu'au moins un des nombres α_1 ou α_2 ait la valeur $\geqslant +0$, et à l'opérateur **B** dans l'autre cas.

Le programme de la négation de l'opérateur donné, conformément à la règle formulée, aura la forme :

$\overline{\mathbf{P}}_{10}$

$k+1$	*TCN*	α_1	**B**	$k+2$
$k+2$	*TCN*	α_2	**B**	**A**

(*) Le code de l'opérateur qui suit immédiatement l'opérateur donné peut être pris égal au nombre qui a une unité de plus que le nombre de toutes les instructions de l'opérateur donné.

Conformément à la signification logique du signe de négation l'opérateur déterminé par ce programme passe la commande à l'opérateur **A** seulement si $\alpha_1 \leqslant -0$ et $\alpha_2 \leqslant -0$.

11) L'opérateur logique déterminé par le programme

$\mathbf{P}_{11}$

$k+1$	*TCN*	α_1	$k+2$	**B**
$k+2$	*TCN*	α_2	**A**	**B**

transmet la commande à l'opérateur **A** à condition que $\alpha_1 \geqslant +0$ et $\alpha_2 \geqslant +0$, et à l'opérateur **B** dans le cas contraire. En appliquant la règle de négation de l'opérateur, nous obtiendrons le programme de l'opérateur $-\mathbf{P}$ qui transmet la commande à l'opérateur **A** à la condition que $\alpha_1 \leqslant -0$ ou $\alpha_2 \leqslant -0$,

$\overline{\mathbf{P}}_{11}$

$k+1$	*TCN*	α_1	$k+2$	**A**
$k+2$	*TCN*	α_2	**B**	**A**

12) La négation de l'opérateur logique $\mathbf{P}_{12}$ déterminé par le programme

$\mathbf{P}_{12}$

$k+1$	*TCN*	α_1	**A**	$k+2$
$k+2$	*TCN*	α_2	**B**	**A**

est obtenue en inversant les opérations

$\overline{\mathbf{P}}_{12}$

$k+1$	*TCN*	α_1	**B**	$k+2$
$k+2$	*TCN*	α_2	**A**	**B**

II. *Règle de programmation de la somme de deux opérateurs logiques* $\mathbf{P}_1 \vee \mathbf{P}_2$.

Le programme de la somme logique de deux opérateurs $\mathbf{P}_1$ et $\mathbf{P}_2$, qui transmettent la commande au même opérateur **A** si l'une des conditions logiques

correspondantes est vraie et à l'opérateur **B** si les deux conditions logiques sont fausses, est obtenu en attribuant à l'instruction de l'opérateur $\mathbf{P}_1$ les instructions de l'opérateur $\mathbf{P}_2$. Dans les instructions de transfert de la commande de l'opérateur $\mathbf{P}_1$ le code de l'opérateur **B** se remplace par le code de l'opérateur $\mathbf{P}_2$.

Examinons des exemples.

13) Soient deux opérateurs :

$\mathbf{P}_1$	*TCN*	α_1	**A**	**B**

$\mathbf{P}_2$	*TCN*	α_2	**A**	**B**

Le programme de la somme logique de ces opérateurs, conformément à la règle, a la forme :

$\mathbf{P}_{13} = \mathbf{P}_1 \vee \mathbf{P}_2$

$k+1$	*TCN*	α_1	**A**	$k+2$
$k+2$	*TCN*	α_2	**A**	**B**

Nous sommes assurés par une vérification directe que le programme obtenu transmet la commande à l'opérateur **A** si, soit la condition $(\alpha_1) \geqslant +0$, soit la condition $(\alpha_2) \geqslant +0$ est satisfaite, et à l'opérateur **B** dans le cas contraire.

14) Formons la somme logique des opérateurs :

$\mathbf{P}_1$	*TCN*	α_1	**B**	**A**

$\mathbf{P}_2$	*TCN*	α_2	**A**	**B**

En réunissant les programmes, nous obtiendrons le programme de l'opérateur qui transmet la commande à l'opérateur **A** aux conditions suivantes :

$P\{(\alpha_1) \leqslant -0 \quad \text{ou} \quad (\alpha_2) \geqslant +0\}$

$k+1$	*TCN*	α_1	$k+2$	**A**
$k+1$	*TCN*	α_2	**A**	**B**

III. *Règle de construction du programme du produit logique de deux opérateurs logiques* $\mathbf{P}_1 \wedge \mathbf{P}_2$.

Ce programme, qui transmet la commande au même opérateur **A** si les conditions logiques correspondantes sont vraies, et à l'opérateur **B** si seule-

ment une seule d'entre elles est fausse, peut être construit selon la formule

$$\mathbf{P}_1 \wedge \mathbf{P}_2 = \overline{(\overline{\mathbf{P}}_1 \vee \overline{\mathbf{P}}_2)},$$

c'est-à-dire que l'établissement du programme du produit logique des opérateurs mène à l'application des règles I et II de construction de la négation et de la somme logique de deux opérateurs.

Il est facile de voir que le programme du produit logique de deux opérateurs logiques $\mathbf{P}_1$ et $\mathbf{P}_2$ peut être construit d'après la règle suivante : on applique aux instructions de l'opérateur $\mathbf{P}_1$ les instructions de l'opérateur $\mathbf{P}_2$; dans les instructions de transfert de commande de l'opérateur $\mathbf{P}_1$ le code de l'opérateur **A** se remplace par le code de l'opérateur $\mathbf{P}_2$. On comprend facilement que les formules des deux règles soient équivalentes.

Examinons des exemples.

15) Soient deux opérateurs :

$\mathbf{P}_1$	*TCN*	α_1	**A**	**B**	$\mathbf{P}_2$	*TCN*	α_2	**A**	**B**

Le programme de leur négation a la forme

$\overline{\mathbf{P}}_1$	*TCN*	α_1	**B**	**A**	$\overline{\mathbf{P}}_2$	*TCN*	α_2	**B**	**A**

Le programme de leur produit logique est le suivant :

$\mathbf{P}_1 \wedge \mathbf{P}_2 = \overline{\overline{\mathbf{P}}_1 \vee \overline{\mathbf{P}}_2}$

$k+1$	*TCN*	α_1	$k+2$	**B**
$k+2$	*TCN*	α_2	**A**	**B**

conformément à la deuxième formulation de la règle de programmation du produit logique.

16) Soient deux opérateurs

$\mathbf{P}_1$	*TCN*	α_1	**A**	**B**	$\mathbf{P}_2$	*TCN*	α_2	**B**	**A**

Le programme du produit logique a la forme :

$\mathbf{P}_1 \wedge \mathbf{P}_2$

$k+1$	*TCN*	α_1	$k+2$	**B**
$k+2$	*TCN*	α_2	**B**	**A**

Examinons la commande de répétition du travail des opérateurs du programme.

Dans les programmes des processus cycliques, les opérateurs individuels ou les groupes d'opérateurs sont normalement appliqués de façon répétée conformément à la valeur de la condition logique dont l'argument est une variable qui change d'un cycle à l'autre. Comme on le sait, une telle condition s'appelle en logique un « prédicat à une seule variable ». Par exemple, on commande le processus cyclique de l'application N-fois répétée d'un opérateur **A** à l'aide de la condition logique **P** $\{ n \leqslant N \}$, n étant la variable qui augmente, à partir de zéro, d'une unité de cycle en cycle (cf. chap. III, § 4).

Dans le cas général, la commande de l'application répétée de certains opérateurs est réalisée, lors du passage d'un cycle à l'autre, par la condition logique $P(n)$, dont l'argument parcourt la suite naturelle des nombres conformément au numéro de répétition du cycle. La valeur de $P(n)$ peut être représentée d'avance sous forme du tableau :

n	0	1	2	...
$P(n)$	$P(0)$	$P(1)$	$P(2)$	...

Ici, $P(n)$ est égal à 0 ou à 1 selon les opérateurs auxquels la commande est transmise. Le programme de l'opérateur logique correspondant peut alors être construit à l'aide du masque logique proposé par M. R. Choura-Boura. Dans le cas le plus simple, le masque logique occupe deux cellules de l'organe mémoire : α et β. La cellule α contient le tableau des valeurs de la condition logique $P(n)$, la cellule β un « 1 » dans la position du signe et des « 0 » dans les autres positions. C'est l'index du masque. Le produit logique de ces deux cellules détermine la présence de « 0 » ou de « 1 » à la division du masque logique correspondant et produit le transfert de la commande selon la valeur obtenue. Après chaque transfert de commande, l'index du masque se déplace soit d'une division à droite, soit d'une division à gauche.

Examinons des exemples :

17) L'opérateur logique est déterminé par la condition $P(n)$ donnée par le tableau :

n	0	1	2	3	4	5	6	7	8	9	10
$P(n)$	0	1	1	1	0	1	1	1	1	0	1

La commande est transmise à l'opérateur **A** si $P(n) = 0$ et à l'opérateur **B** si $P(n) = 1$.

Le programme de l'opérateur logique avec application du masque logique a la forme

Nombres

α_1	01110111101
α_2	10000000000

Instructions

$k + 1$	$\wedge$	α_1	α_2	α
$k + 2$	$\rightarrow$	1	α_1	α_1
$k + 3$	*TCN*	α	**A**	**B**

ou (*)

α_1	0 1110111101 ...

$k + 1$	*TS* +	α_1	α_1	α_1
$k + 2$	*TCN*	α_1	**A**	**B**

Si le nombre de positions d'une cellule est insuffisant pour contenir le tableau de valeurs de la condition logique, le masque peut être disposé dans plusieurs cellules. Le programme de l'opérateur correspondant alors se complique.

18) L'opérateur logique **P**(n) est déterminé par le tableau

n	0	1	2	3	4	5	6	7	...	57	58	59	60
$P(n)$	0	1	1	1	0	1	1	1		0	1	1	1

(*) Dans le second cas le masque de sortie est placé à partir de la première position numérique, l'opération *TS* + réalise le déplacement du masque dans la position du signe.

que nous plaçons dans deux cellules de l'organe mémoire (en écriture à base seize)

$\alpha + 1$	777.777.777
$\alpha + 2$	777.777.000

Le programme de l'opérateur a la forme :

Nombres

$\alpha + 1$	777777777			
$\alpha + 2$	777777000			
α_3	800000000			
α_4	800000000			
α_5	−	1	−	−

Instructions

$k + 1$	$\wedge$	$\alpha + 1$	α_3	α
$k + 2$	$\rightarrow$	1	α_3	α_3
$k + 3$	−	α_3	−	β
$k + 4$	−	α	−	α
$k + 5$	*TCN*	β	$k + 8$	$k + 6$
$k + 6$	$\oplus$	$k + 1$	α_5	$k + 1$
$k + 7$	+	α_4	−	α_3
$k + 8$	*TCN*	α	**B**	**A**

Les masques logiques peuvent être utilisés pour transmettre la commande dans plus de deux directions. Dans ce cas, la condition logique correspondante n'a pas deux, mais plusieurs valeurs, et chaque valeur de la condition logique est par conséquent codifiée à l'aide d'un nombre binaire à plusieurs chiffres. Par exemple, pour transmettre la commande dans trois ou quatre directions on utilise une condition logique à trois ou quatre valeurs ; chaque valeur en est codifiée par un nombre binaire à deux positions, ce qui fait que l'index du masque contient autant d'unités que de positions affectées à la codification des valeurs de la condition logique.

19) L'opérateur de commande est donné par la condition logique

n	0	1	2	3	4	5	...
P(n)	00	01	11	00	01	11	...

où la commande est transmise aux opérateurs $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, si $\mathbf{P}(n)$ est égal respectivement à 00, 01, 11.

Le programme de l'opérateur, lorsqu'on utilise le masque logique, est :

α_1	000	111 000	111 ...
α_2	1 1 0 ...		
α_3	0 0 ... 0		
α_4	0 1 0 ...		

$k+1$	$\wedge$	α_1	α_2	α
$k+2$	$\leftarrow$	α_1	α_1	α_1
$k+3$	$\leqslant$	α	α_3	$\mathbf{A}_1$
$k+4$	$\leqslant$	α	α_4	$\mathbf{A}_2$
$k+5$	*TI*	—	—	$\mathbf{A}_3$

Si les valeurs de la condition logique $P(n)$ se répètent périodiquement, il ne faut construire que le masque d'une seule période des valeurs $P(n)$. De plus il faut prévoir, dans le programme de l'opérateur correspondant, le rétablissement du masque (ou de l'index du masque) à la fin de la période.

20) La condition logique est donnée sous la forme

$$P(3\,n) = 1\,,$$
$$P(3\,n \pm 1) = 0\,.$$

Le programme d'un tel opérateur de transfert de commande conformément aux valeurs périodiquement répétées de $P(n)$ est établi de la manière suivante :

α_1	001 ...
α_2	001 ...
α_3	001 ...

$k+1$	$\wedge$	α_1	α_2	α
$k+2$	$\leqslant$	—	α_1	$k+5$
$k+3$	+	α_3	—	α_1
$k+4$	$\leqslant$	—	—	$k+6$
$k+5$	$\leftarrow$	1	α_1	α_1
$k+6$	*TCN*	α	**A**	**B**

6. TRANSFORMATION DU CONTENU DU SCHÉMA DES PROGRAMMES

Comme on l'a noté plus haut, le programme selon un schéma de calcul ne se détermine pas d'une façon univoque. Sur un même schéma de calcul, on peut établir des programmes différents. On peut poser la question de la transformation des programmes sous une forme plus simple et mieux adaptée au traitement en machine (d'après le volume de l'information à l'entrée ou le nombre des mémoires de travail de la machine, etc.). La transformation du schéma des programmes peut être réalisée en tenant compte de leur contenu ou, simplement, formellement d'après un algorithme précédemment élaboré (cf. Faddeieva [1]).

Dans ce paragraphe, nous examinerons quelques exemples de transformation du contenu du schéma des programmes.

1. *Transformation du schéma des programmes avec changement des opérateurs de substitution d'adresse.*

Soit le schéma d'un programme de processus cyclique binaire à deux paramètres :

$$\begin{array}{cccccccccc} & & 2 & 1 & & & \mathrm{A} & & & \mathrm{A} \\ \Phi_1(k) & & \downarrow & \downarrow & \mathbf{A}_i\,\mathbf{F}_1(i)\,\mathbf{P}_1 & & \uparrow & \mathbf{B}_k\,\mathbf{F}_2(k)\,\Phi_2(i)\,\mathbf{P}_2 & & \uparrow \quad ! \,. \end{array} \tag{11}$$

Dans l'exemple concret de la résolution de l'équation de la conductibilité de la chaleur (cf. programme 2, § 3), les programmes des opérateurs $\mathbf{F}_1(i)$ et des opérateurs $\mathbf{F}_2(k)\ \Phi_2(i)$ contiennent une partie commune. Cette particularité peut être utilisée pour transformer le schéma du programme. Puisque le programme des opérateurs $\mathbf{F}_2(k)\Phi_2(i)$ contient deux instructions semblables à celles qui représentent l'opérateur $\mathbf{F}_1(i)$, il est possible d'exclure du programme les deux instructions de l'opérateur $\mathbf{F}_2(k)$ en transmettant la commande aux instructions de l'opérateur $\mathbf{F}_1(i)$. Ainsi, le schéma transformé du programme prend la forme :

$$\begin{array}{cccccccccc} & & 1 & & 2 & & \mathrm{A} & & & \mathrm{F} \\ \Phi_1(k) & & \downarrow & \mathbf{A}_i & \downarrow & \mathbf{F}(i)\,\mathbf{P}_1 & \uparrow & \mathbf{B}_k\,\Phi(i)\,\mathbf{P}_2 & & \uparrow \quad ! \,. \end{array} \tag{12}$$

L'opérateur $\Phi(i)$ prépare le travail de la condition logique P_1.

Dans le programme qui résout le problème de Dirichlet concernant le rectangle (cf. chap. IV, § 3, 2), il est aussi possible d'apporter un changement grâce à la relation existant entre les paramètres k et i : la variation de k de k unités correspond à la variation de i d'une unité. Les opérateurs $\mathbf{F}_2(i)\ \Phi_2(k)$ peuvent être réalisés par la double application de l'opérateur $\mathbf{F}(k)$, ce que l'on peut faire par l'introduction d'une condition logique spéciale P_4. Le schéma du

programme sera le suivant :

$$\overset{3}{\Phi_1(s)} \downarrow \Phi_2(i) \overset{2}{\downarrow} \overset{1}{\downarrow} \mathbf{A}_{kis} \overset{4}{\downarrow} \mathbf{F}_1(k)\,\mathbf{P}_1 \overset{\mathbf{A}}{\uparrow} \mathbf{P}_4 \overset{\mathbf{F}_1}{\uparrow} \mathbf{P}_2 \overset{\mathbf{A}}{\uparrow} \mathbf{F}_3(s)\,\mathbf{P}_3 \overset{\Phi_2(i)}{\uparrow} ! \,. \quad (13)$$

Pour établir le programme selon ce schéma, il faut prêter attention au travail de la condition logique P_1.

La complexité de la substitution d'adresse dans le programme de calcul du déterminant d'une matrice carrée, non dégénérée, est due à la variabilité du nombre de cycles en fonction du paramètre i. On peut éliminer cet inconvénient en introduisant la condition logique P_4 qui exclut du travail l'opérateur de calcul $\mathbf{A}_{kir}$ pour $i \leqslant r$, c'est-à-dire

$$P_4 = \begin{cases} 0, & i > r, \\ 1, & i \leqslant r. \end{cases}$$

Le schéma du programme a désormais la forme

$$\overset{3}{\Phi(r)} \downarrow \overset{2}{\mathbf{D}_r} \downarrow \overset{1}{\mathbf{C}_{ir}} \downarrow \overset{\mathbf{F}_2}{\mathbf{P}_4} \uparrow \mathbf{A}_{kir}\,\mathbf{F}_1(k)\,\mathbf{P}_1 \overset{\mathbf{P}_4 4}{\uparrow \downarrow} \mathbf{F}_2(i, k)\,\mathbf{P}_2 \overset{\mathbf{C}}{\uparrow} \mathbf{F}_3(r, i)\,\mathbf{P}_3 \overset{\mathbf{D}}{\uparrow} !. \quad (14)$$

2. *Transformation des opérateurs de calcul.*

Le programme qui résout une équation différentielle du premier ordre sur une machine à virgule fixe possède deux opérateurs de calcul $\mathbf{A}^{\mathrm{I}}$ et $\mathbf{A}^{\mathrm{II}}$, sur chaque cycle desquels n'agit qu'un opérateur en fonction de la condition logique P_1.

La comparaison des programmes des opérateurs $\mathbf{A}^{\mathrm{I}}$ et $\mathbf{A}^{\mathrm{II}}$ montre que les instructions qui les réalisent ne se distinguent que par l'adresse des nombres ; ce qui permet de transformer l'opérateur de calcul $\mathbf{A}^{\mathrm{I}}$ en opérateur de calcul $\mathbf{A}^{\mathrm{II}}$ et vice versa.

Introduisons l'opérateur **H** de transformation de $\mathbf{A}^{\mathrm{I}}$ en $\mathbf{A}^{\mathrm{II}}$ et l'opérateur **M** de transformation de $\mathbf{A}^{\mathrm{II}}$ en $\mathbf{A}^{\mathrm{I}}$. Le schéma du programme peut alors être ramené à la forme :

$$\overset{2}{\downarrow} \Phi\Psi\mathbf{P}_1 \overset{\mathbf{A}}{\uparrow} \mathbf{H} \overset{1}{\downarrow} \mathbf{A}^{\mathrm{II}}\,\mathbf{P}_3 \overset{\mathbf{P}_2}{\uparrow} \mathbf{M} \overset{3}{\downarrow} \mathbf{P}_2 \overset{\Phi}{\uparrow} !. \quad (15)$$

La programmation des opérateurs de transformation **H** et **M** peut être réalisée de deux façons différentes : soit que les opérateurs exécutent le transfert des données nécessaires dans les cellules standards, dont les numéros entrent dans les adresses des instructions de l'opérateur **A**, soit que les opérateurs **H** et **M** réalisent respectivement la substitution d'adresse nécessaire des instructions des opérateurs $\mathbf{A}^{\mathrm{I}}$ et $\mathbf{A}^{\mathrm{II}}$.

Citons le programme des opérateurs de transformation **H** et **M** dans les deux cas. Dans le premier cas, l'opérateur **H** doit changer de place les nom-

bres x_k et y_k, φ_k et ψ_k dans les cellules de l'organe mémoire, ce qui est réalisé par l'instruction

$H + 1$	+	$\alpha + 1$	−	ω
$H + 2$	+	$\alpha + 2$	−	$\alpha + 1$
$H + 3$	+	ω	−	$\alpha + 2$
$H + 4$	+	$\alpha + 3$	−	ω
$H + 5$	+	$\alpha + 4$	−	$\alpha + 3$
$H + 6$	+	ω	−	$\alpha + 3$

Le programme de l'opérateur **M** rétablit la répartition initiale de ces données dans l'organe mémoire pour le travail de l'opérateur **A** et a la même forme que le programme de l'opérateur **H**.

Dans le second cas, pour substituer les adresses des instructions de l'opérateur **A**, il faut les constantes suivantes :

δ_1	−	1	−	−
δ_2	−	−	1	−
δ_3	−	−	1	1
δ_4	−	1	−	1

Le programme de l'opérateur **H**, qui réalise la substitution d'adresse des instructions de l'opérateur $\mathbf{A}^{\mathrm{I}}$ en instructions de l'opérateur $\mathbf{A}^{\mathrm{II}}$, a la forme :

$H + 1$	$\oplus$	$A + 1$	δ_1	$A + 1$
$H + 2$	$\ominus$	$A + 2$	δ_2	$A + 2$
$H + 3$	$\ominus$	$A + 4$	δ_3	$A + 4$
$H + 4$	$\oplus$	$A + 5$	δ_4	$A + 5$

Le programme de l'opérateur **M** qui rétablit les instructions de l'opérateur **A**, a la forme :

$M + 1$	$\ominus$	$A + 1$	δ_1	$A + 1$
$M + 2$	$\oplus$	$A + 2$	δ_2	$A + 2$
$M + 3$	$\oplus$	$A + 4$	δ_3	$A + 4$
$M + 4$	$\ominus$	$A + 5$	δ_4	$A + 5$

EXERCICES

1. Transformer le schéma du programme 4 d'une manière analogue à la transformation du programme 2.

2. Introduire la condition logique dans le programme 3 qui élimine le nombre variable des cycles internes selon le paramètre k.

CHAPITRE V

PROGRAMMES A ADRESSES

Lorsqu'on a des problèmes complexes à traiter, il y a une différence fondamentale entre le schéma de calcul et le schéma des programmes. Le poids spécifique des opérateurs arithmétiques qui effectuent le calcul d'après les formules est tout à fait insignifiant dans les schémas des programmes des problèmes complexes. Les opérateurs de commande (substitution d'adresse, transfert de commande, etc.), qui assurent l'accomplissement d'un schéma logique déterminé de calcul, occupent la place principale.

Le calcul arithmétique est presque complètement absent des algorithmes des problèmes non arithmétiques. La description de tels algorithmes est faite sous forme d'un système de règles de transformation de l'information initiale et habituellement énoncée sous forme verbale. La formulation verbale de l'algorithme n'est pas, en principe, assez précise et ne contient pas d'indications sur les moyens de le réaliser en programme. La programmation des problèmes non arithmétiques dans les C. A. N. devient difficile si ces indications n'existent pas. Un moyen efficace de décrire les processus algorithmiques, pratique pour la lecture et son exécution en machine, doit répondre aux exigences suivantes :

1. Il doit être voisin de la description alphabétique et symbolique adoptée en mathématique pour les processus algorithmiques.

2. La description des algorithmes ne doit pas dépendre des particularités techniques matérielles des C. A. N. et doit laisser aussi la possibilité d'un transfert automatique à *la description par programme, dans les instructions des C. A. N. existantes.*

En même temps, la description par programme de l'algorithme, dans les instructions des C. A. N., contient, outre le schéma logique de l'algorithme, des particularités techniques spécifiques aux C. A. N. existantes.

Le présent chapitre expose un moyen de programmer qui a pour base deux particularités caractéristiques de la réalisation programmée des processus numériques sur C. A. N. : le principe de l'adressage et le principe de la commande par programme (cf. Koroliouk V. S. [1] et Koroliouk V. S., Iouchtchenko E. L. [1]).

1. CONCEPT D'UN PROGRAMME A ADRESSES

1. Remarques préliminaires.

Chaque machine a son choix d'opérations et une série de particularités techniques spécifiques qui doivent entrer en ligne de compte pour la programmation. Pour résoudre un même problème, le programme a différentes formes qui dépendent des différentes C. A. N. C'est pourquoi, la programmation des problèmes est habituellement exposée dans les instructions des diverses C. A. N. existantes. Il est alors clair que les principes logiques fondamentaux de programmation ne doivent pas dépendre des particularités techniques concrètes de chaque C. A. N.

Le principe d'adressage est un principe général d'étude des programmes pour toutes les C. A. N. L'information sur le problème est codifiée et répartie dans un ordre déterminé dans les cellules de l'organe mémoire de la machine à adresses. La répartition des données initiales dans les cellules de l'organe mémoire est l'étape de début de la programmation. Le programme du processus numérique est enregistré sous *forme d'adresses*, c'est-à-dire que, dans le programme, on indique non les nombres sur lesquels on doit faire l'opération, mais les adresses des cellules de l'organe mémoire contenant ces nombres. Exécuter un programme sur C. A. N. conduit à résoudre un problème concret avec un chargement donné des cellules de l'organe mémoire. Grâce à la description du programme sous forme d'adresses, il est possible d'après un seul et même programme de résoudre différents problèmes concrets, en changeant le contenu des cellules de l'organe mémoire (en changeant les paramètres des problèmes). Cela assure le caractère « ramassé » de l'algorithme décrit sous forme d'un programme.

Le principe même de l'adressage est à la base des autres méthodes de description des processus numériques. Par exemple, lorsqu'on décrit le processus numérique sous la forme littérale, il est entendu que pour réaliser concrètement ce processus on met à la place des lettres les nombres qui leur correspondent.

Une autre particularité de description par programme des processus numériques commune à toutes les C. A. N. est *le principe de la commande par programme*. L'ordre de réalisation des différentes étapes de calcul est rigoureusement déterminé dans le programme par l'ordre des instructions et par les instructions de transfert de commande, compte tenu des résultats des calculs intermédiaires.

Nous maintiendrons ces principes de programmation dans la définition du concept de programme à adresses, tout en renonçant, dans la description des processus numériques, aux autres limitations liées aux particularités techniques de construction des C. A. N. De plus, le langage de la programmation à adresses est naturellement, autant que possible, conservé dans les machines :

ce qui fait qu'on peut facilement automatiser la transformation des programmes à adresses en programmes pour les C. A. N. existantes. Nous remarquerons aussi que le perfectionnement technique des machines permet un allègement très important de la transformation du programme à adresses pour de telles C. A. N.

2. Système des objets de codification.

Le système des objets S, entre lesquels des relations sont établies, sert de matériel initial pour codifier l'information d'un problème. Les objets du système S seront appelés *les codes*. Les relations entre les codes établies dans le système des codes S seront appelées *opérations sur les codes*. Pour désigner les codes et les opérations, nous utiliserons différents symboles. En principe, les codes sont désignés par des lettres (latines ou grecques) et des nombres. Pour les opérations, on utilise des symboles spéciaux empruntés aux mathématiques :

opérations arithmétiques $+$, $\times$, x, $:$;
opérations fonctionnelles $\sqrt{\ }$, sin, Log. etc.
opérations logiques $\vee$, $\wedge$, $\neg$ (opération-négation) ;
opérations de relations (elles s'appellent aussi opérations de prédicat) $=$, $<$, $\leqslant$, etc.

Exemples.

1. L'ensemble fini S des objets (symboles), pour lesquels seule une relation de différence est établie. Pour toute paire d'objets de S il est déterminé s'ils sont égaux ou non. Ce système d'objets est normalement appelé alphabet.

2. La série des nombres naturels constitue le système S $(N, O, ')$. N est l'ensemble des nombres naturels, 0 est un élément de l'ensemble N, $'$ est l'opération de succession des nombres naturels.

3. L'ensemble D des nombres binaires à n positions, pour lesquels des opérations à une place (*) (par exemple, la négation) et à deux places (par exemple, une opération arithmétique) sont données, constitue le système d'objets utilisé pour codifier l'information des problèmes en C. A. N.

Les relations entre les codes, établies dans le système des codes, permettent de construire, selon les règles habituelles, les expressions (les fonctions) dont les valeurs sont des codes obtenus après avoir effectué les opérations indiquées dans l'expression. Remarquons qu'à la suite de l'opération de relation, on ne peut avoir que deux codes, que nous désignerons par 1 si la relation est satisfaite et par 0 sinon. Autrement dit, les opérations de relation déterminent les fonctions qui ne prennent que deux valeurs (1 ou 0). De telles fonctions

(*) On appelle « à une place » les fonctions à un argument, « à deux places », les fonctions à deux arguments.

sont appelées prédicats. Convenons d'inscrire les prédicats sous la forme $P\{\ \}$, où l'expression qui détermine le prédicat est enregistrée entre accolades.

3. Opérations algorithmiques dans le système des codes.

Dans le système des codes S, outre les opérations qui déterminent les fonctions arithmétiques et les opérations de prédicats, nous introduirons *l'opération de séparation du contenu de l'adresse* ou plus brièvement *l'opération adresse* désignée par le symbole $'$ qui se place en haut et à gauche avant le code. Le contenu de cette opération est le suivant :

A chaque code a de S on fait correspondre un code 'a de S. Le code a est appelé adresse du code 'a. Le code 'a est appelé contenu de l'adresse a. L'application répétée de l'opération de séparation du contenu de l'adresse conduit au concept de rang de l'adresse. Soit b le contenu de l'adresse a, c'est-à-dire que $'a \equiv b$, et c le contenu de l'adresse b, c'est-à-dire $'b \equiv c$; alors a est l'adresse de l'adresse du code c.

Nous appellerons a adresse du deuxième rang du code c ou, plus brièvement, fixateur du code c et nous noterons :

$$^2a \equiv '('a) \equiv 'b \equiv c\,.$$

On détermine d'une façon analogue *l'adresse du k-ième rang* (k est un nombre entier positif). Le contenu de l'adresse a du k-ième rang est déterminé par la relation :

$$^ka \equiv '(^{k-1}a)\,.$$

Dans le système des codes S, la définition de l'opération de séparation du contenu de l'adresse détermine l'état du système des codes S que nous appellerons *répartition des adresses dans S* (le chargement des cellules de l'organe mémoire de la machine détermine de façon analogue l'état de l'organe mémoire de la machine). Introduisons ici l'opération algorithmique de *transfert selon l'adresse* (avec effacement) qui change le contenu des adresses, c'est-à-dire transforme l'état du système S. Pour désigner l'opération de transfert selon l'adresse nous utiliserons le symbole $\Rightarrow$.

L'application des opérations de transfert selon l'adresse aux codes a et b de S

$$a \Rightarrow b\,,$$

indique que $'b \equiv a$, c'est-à-dire que a devient le contenu de l'adresse b (effaçant le contenu précédent de b).

Comme les valeurs de la fonction f des codes de S sont aussi des codes provenant de S, l'opération adresse qui est la valeur de la fonction $'f$ a un sens. De plus, $'f$ désigne le code qui est le contenu d'une adresse, celle-ci étant la

valeur de la fonction f. Dans l'opération $a \Rightarrow b$, on peut considérer a et b comme des fonctions. On peut alors formuler ainsi le résultat de l'opération $a \Rightarrow b$: la valeur de la fonction a devient le contenu de l'adresse qui est le résultat du calcul de la valeur de la fonction b ($'b \equiv a$).

Nous appellerons l'expression $a \Rightarrow b$ *formule de transformation* ou *opérateur de fonctionnement*.

Pour construire les programmes nous utiliserons les formules de prédicat que nous déterminerons de la manière suivante :

Soit P, la fonction de prédicat dans S, c'est-à-dire la fonction qui peut seulement prendre deux valeurs (1 ou 0). Soient α et β les formules (ou la désignation des formules).

L'expression $P\alpha\beta$ s'appelle formule de prédicat.

Signalons que, dès lors, α et β, dans la définition de formule de prédicat, peuvent être, soit des formules de transformation, soit des formules de prédicat.

Les formules de prédicat ne changent pas l'état de S (c'est-à-dire ne changent pas le contenu des adresses), mais déterminent l'ordre d'action des formules selon la règle suivante : *si la fonction de prédicat P prend la valeur* 1, *on applique la formule* α ; *si P prend la valeur* 0, *on applique la formule* β (*).

Les formules de prédicat s'appellent encore *opérateurs de choix*.

Notons encore que si, pour désigner les formules dans la formule de prédicat, on utilise les codes de S, il est possible de déterminer l'ordre d'application des formules selon l'adresse. Par exemple, la notation $P'\alpha^2\ \beta$ indique que, lorsque la valeur de P est 1, on emploie la formule $'\alpha$, c'est-à-dire la formule dont la désignation est le contenu de l'adresse α ; si P prend la valeur 0 on utilise la formule dont la désignation est le contenu de l'adresse du deuxième rang β. Si la fonction de prédicat P est identiquement égale à 1, le second exposant de la formule de prédicat correspondante s'abaisse.

Convenons d'abaisser le second exposant aussi lorsque, avec la valeur de $P = 0$, la commande se transmet à la formule de rang suivant.

Nous noterons $P\alpha$ le transfert inconditionnel de la commande à la formule α.

Enfin, convenons d'indiquer la fin du travail du programme par « *ARR* ».

Passons maintenant à la définition du concept de programme à adresses.

Le programme à adresse est défini par la distribution initiale des adresses en S (état du système des codes S) et par la succession des formules d'adresses avec l'indication de l'ordre de leur application.

Ce qui donne l'indication de l'ordre de fonctionnement des formules est, soit les formules de prédicat, soit l'ordre ordinaire de succession : de gauche à droite lorsqu'on écrit sur une ligne, de haut en bas lorsqu'on écrit dans une colonne.

Il peut arriver que dans le programme des formules puissent agir dans un ordre arbitraire. Alors, au moment de l'inscription du programme dans une

(*) Quelquefois nous écrirons les formules de prédicat sous la forme P^{α}_{β}.

colonne, nous écrirons ces formules sur une ligne, et lors de l'inscription sur une ligne, nous les écrirons dans une colonne.

Illustrons la définition d'un programme à adresses par des exemples concrets très simples.

***Exemple* 1.** *Calcul d'une somme de n nombres* :

$$S_n = \sum_{k=1}^{n} a_k .$$

Introduisons les adresses $\alpha_k = \alpha_0 + k$ pour les termes ($'\alpha_k = a_k$), les sommes $S('S = 0)$ et les adresses $\alpha('\alpha = \alpha_0 + 1)$, $\beta('\beta = \alpha_0 + n)$.

Le programme à adresses a la forme suivante :

Répartition des adresses

α		β		k_1 $'S + {}^2\alpha \Rightarrow S$
$\alpha_0 + 1$	...	$\alpha_0 + n$	S	$'\alpha + 1 \Rightarrow \alpha$
a_1		a_n	0	$P\{'\alpha \leqslant '\beta\}\, k_1$ *ARR*

***Exemple* 2.** *Calculer et mettre en mémoire N nombres de la progression géométrique*

$$a_{k+1} = a_k . q,\ k = 0, 1, 2, \ldots, N .$$

Introduisons les adresses $\alpha_k = \alpha_0 + k$ pour les membres de la progression $'\alpha_0 = \alpha_0$ (contenu initial des adresses α_k, où $1 \leqslant k \leqslant N$ est arbitraire) et les adresses $\delta('\delta = q)$.

Le programme à adresses a la forme suivante :

Répartition des adresses

α			β		k_1 $\quad {}^2\alpha \times '\delta \Rightarrow '\alpha + 1$
α_0	$\alpha_0 + 1$	...	$\alpha_0 + N$	δ	$'\alpha + 1 \Rightarrow \alpha$
a_0				q	$P\{'\alpha \leqslant '\beta\}\, k_1$ *ARR*

4. Organigramme des programmes.

Si l'on introduit la désignation « → » pour déterminer le passage d'une formule à l'autre et si, de plus, on représente les formules de prédicat sous

la forme $P_{0\to\beta}^{1\to\alpha}$, nous arriverons à déterminer l'organigramme du programme (cf. Kaloujnine, L. A. [1]), qui représente concrètement le déroulement du programme.

Soit $D_1, D_2, \ldots, D_m$ le système des opérateurs de fonctionnement qui transforment l'information concernant le problème, et $P_1, P_2, \ldots, P_r$ le système des opérateurs de choix. Le système des points unis par des flèches s'appelle l'organigramme $D - P$. Il y a un point qui s'appelle entrée de l'organigramme et duquel ne part qu'une flèche ; désignons ce point par la lettre B. Il y en a un qui s'appelle sortie de l'organigramme ; désignons-le par le signe $\mathfrak{A}$; nous désignerons parfois ce point par « ARR ». Les autres points de l'organigramme peuvent être, soit des points D, soit des points P. Une seule flèche part de chaque point D, deux flèches de chaque point P : l'une avec la marque 1, l'autre avec la marque 0. A n'importe quel point de l'organigramme peut aboutir un nombre quelconque de flèches. On fait correspondre un opérateur de fonctionnement D_k ($1 \leqslant k \leqslant m$) à chaque point D et un test P_s ($1 \leqslant s \leqslant r$) à chaque point P.

L'action de l'organigramme $D - P$, comme celle d'un algorithme, se déroule de la façon suivante : à partir du point d'entrée, en suivant la flèche, le signal d'action passe au point de l'organigramme suivant (nous le désignerons par T) ; si le point T est un point D, l'information concernant le problème est transformée conformément à la description de l'opérateur de fonctionnement qui lui correspond, après quoi le signal d'action passe au point suivant de l'organigramme selon la flèche qui part du point T ; si le point T est un point P, on détermine dans l'information une particularité conformément à la description du test qui lui correspond ; s'il y a dans l'information la particularité cherchée, le signal d'action suit la flèche marquée 1, sinon le signal d'action est transmis suivant la flèche marquée 0 etc. L'algorithme termine le travail lorsque le signal d'action arrive au point de sortie $\mathfrak{A}$ de l'organigramme. L'organigramme $D - P$ ne détermine que l'ordre des opérations sans restreindre la complexité du schéma ni le caractère des opérateurs et des tests.

5. Programmation des formules à adresses pour les C. A. N.

La transformation des programmes à adresses en programmes pour des types existants de C. A. N., compte tenu des particularités techniques, ne pose pas de grandes difficultés et peut être automatisée.

Prenons des exemples de la façon de programmer des formules à adresses très simples, qui contiennent des adresses de second rang dans les codes des C. A. N. à trois adresses, avec les quelques opérations décrites dans le chapitre II, à savoir

$$a) \quad {}^2\alpha \Rightarrow \beta\,, \qquad b) \quad {}'\alpha \Rightarrow {}'\beta\,, \qquad c) \quad {}^2\alpha \Rightarrow {}'\beta\,.$$

Il est évident que les programmes de ces formules dépendent du procédé de codification. Nous examinerons le cas le plus simple où les fixateurs des codes

contiennent dans les premières adresses les numéros des cellules de l'organe mémoire qui conservent ces codes, à condition que l'on ne garde dans chaque cellule qu'un seul code. Ces hypothèses permettent de construire très facilement les programmes des formules à adresses.

a) Soient les codes a et b contenus dans les cellules qui ont respectivement les numéros γ et ρ, c'est-à-dire $'\gamma = a$, $'\rho = b$, et soient les numéros des cellules γ et ρ contenus dans la première adresse respectivement des cellules α et β ; on met des zéros dans les autres positions de ces cellules.

Ainsi $'\alpha = \gamma$, $'\beta = \rho$. Le programme de transfert ${}^{2}\alpha \Rightarrow \beta$ a la forme suivante :

$k+1$	$\wedge$	δ_1	$k+3$	$k+3$
$k+2$	$+$	α	$k+3$	$k+3$
$k+3$	$+$			β

Ici, la cellule δ_1 contient la constante qui a des « uns » dans les positions du code opération et dans la troisième adresse et des « zéros » dans les positions de la première et de la seconde adresses.

Conventionnellement nous la représenterons sous la forme

δ_1	1...1	0...0	0...0	1...1

L'instruction $k+1$ donne à l'instruction $k+3$ la forme indiquée dans le programme. Lorsque l'instruction $k+2$ est accomplie, l'instruction $k+3$ prend la forme

$+$	γ		β

b) Le programme du transfert $'\alpha \Rightarrow '\beta$ a la forme suivante :

$k+1$	$\wedge$	δ_2	$k+4$	$k+4$
$k+2$	$\rightarrow 2\,A$	β		c
$k+3$	$+$	c	$k+4$	$k+4$
$k+4$	$+$	α		

où

δ_2	1...1	1...1	0...0	0...0

L'instruction $k + 1$ donne à l'instruction $k + 4$ la forme indiquée dans le programme. L'instruction $k + 2$ transfert le contenu de la première adresse de la cellule β dans la troisième adresse de la cellule c (Il se produit un déplacement de deux adresses à droite du contenu de l'adresse β). Une fois que l'instruction $k + 3$ est exécutée. l'instruction $k + 4$ a la forme

$k + 4$	+	α		ρ

c) Le programme du transfert ${}^2\alpha \Rightarrow {}'\beta$ est enregistré sous la forme suivante :

$k + 1$	$\wedge$	δ_3	$k + 5$	$k + 5$
$k + 2$	+	α	$k + 5$	$k + 5$
$k + 3$	$\rightarrow 2\,A$	β		c
$k + 4$	+	c	$k + 5$	$k + 5$
$k + 5$	+			

Ici

δ_3	1...1	0...0	0...0	0...0

L'instruction $k + 1$ donne à l'instruction $k + 5$ la forme indiquée dans le programme. Après l'exécution des instructions $k + 2$ à $k + 4$, l'instruction $k + 5$ a la forme suivante :

$k + 5$	+	γ		ρ

Rappelons que ${}'\alpha = \gamma$, ${}'\beta = \rho$. La programmation des programmes à adresses plus compliquées conduit à la programmation des opérateurs de transfert déjà examinés et à une programmation ordinaire.

L'introduction en C. A. N. d'opérations spéciales peut simplifier considérablement la programmation des formules à adresses (cf. description de la machine *Kiev*).

2. SCHÉMAS D'EXAMEN DES CODES

1. Concept de l'examen de l'information.

Comme il a été noté au début du paragraphe précédent, dans les programmes des problèmes traités sur C. A. N., on indique les adresses des cellules de l'organe mémoire qui ont pour contenu les codes de l'information sur le problème. Cependant, ce n'est que pour des problèmes très simples que le programme contient les adresses de tous les codes de cette information. Par exemple, le programme du calcul d'après une formule donnée contient les adresses de toutes les cellules dans lesquelles on conserve les nombres participant au calcul. En principe, il n'y a qu'une partie des adresses des codes de l'information qui rentre dans le programme ; les adresses des autres codes sont utilisées au cours du travail. Dans la machine décrite au chapitre II, on y parvient grâce aux instructions de substitution d'adresse.

Nous dirons *qu'un code donné n'est examiné par le programme que si son adresse d'un certain rang est contenue dans le programme.*

L'application des adresses de rangs supérieurs permet de mieux examiner les codes du programme.

L'adresse a de second rang du code c ($^2a \equiv c$) sera appelée *fixateur* du code c.

Dans le programme de nombreux problèmes, les codes de l'information initiale sont examinés en une suite déterminée.

Nous appellerons *schéma d'examen de la suite des codes*

$$a_0, a_1, \ldots, a_n \tag{1}$$

le programme à adresses cycliques, au i-ième cycle duquel on examine le i-ième élément de la suite.

Pour construire les programmes d'examen des codes, nous introduirons l'opération successeur C dans la suite des codes (1)

$$Ca_i = a_{i+1} .$$

Le code a_{i+1} s'appelle code *suivant* le code a_i ; le code a_i s'appelle le code *précédant* (antérieur) le code a_{i+1}. Il est évident que l'opération C a sa réciproque $\overline{C}$

$$\overline{C}a_{i+1} = a_i .$$

La suite des codes (1) est mise en ordre par l'opération successeur C.

Exemples.

1) Dans la suite des nombres entiers

$$n + 1, n + 2, \ldots, n + m$$

l'opération successeur consiste à ajouter une unité

$$C(n + k) = (n + k) + 1 .$$

2) Dans la suite des nombres qui forment la progression arithmétique de raison d, l'opération successeur est définie par addition du nombre d ; pour les nombres qui forment la progression géométrique de raison q, elle l'est par multiplication par q.

Pour mettre en ordre une suite quelconque de codes (1), répartissons-les dans les adresses

$$\alpha_0, \alpha_1, \ldots, \alpha_n , \tag{2}$$

pour lesquelles l'opération successeur C est déterminée :

$$C\alpha_i = \alpha_{i+1} \quad (0 \leqslant i \leqslant n) , \tag{3}$$

en outre

$$'\alpha_i = a_i \qquad (0 \leqslant i \leqslant n) .$$

Alors, l'opération successeur C_1 des codes (1) est ainsi énoncée :

$$C_1 a_i = '(C\alpha_i) ,$$

puisque

$$'(C\alpha_i) = '\alpha_{i+1} = a_{i+1} .$$

Pour construire le programme qui examine la suite des codes (1), introduisons l'adresse α comme adresse du premier code :

$$'\alpha = \alpha_0 ,$$

c'est-à-dire α est le fixateur du code α_0 :

$$^2\alpha = a_0 .$$

Afin que ce soit évident, nous représenterons la répartition initiale des adresses par le tableau

α			
α_0	α_1	...	α_n
a_0	a_1		a_n

(4)

Les codes de la suite (1) sont répartis sur la dernière ligne ; les adresses des codes disposés sur cette dernière ligne sont indiquées sur la ligne qui précède.

Le programme à adresses d'examen des codes (1) a la forme suivante :

Répartition des adresses *Programme* 1

α			
α_0	α_1	...	α_n
a_0	a_1		a_n

$k_1.\ C'\alpha \Rightarrow \alpha\ ;$

$k_2.\ P\{\ '\alpha \neq \alpha_n\ \}\ k_1\ ARR.$

La première formule change le contenu du fixateur α conformément à (3), de sorte que les codes de (1) sont successivement examinés par le programme 1. Sur le tableau de la répartition des adresses, l'action de la formule k_1 est représentée comme le déplacement du fixateur α d'une seule adresse vers la droite sur une même ligne. Par la suite, nous appellerons les formules de forme k_1 *formules de déplacement du fixateur*.

Examinons les différentes modifications du programme 1. Introduisons l'adresse β, où $'\beta = \alpha_n$; la forme du programme sera alors standard pour n'importe quelle suite de codes (1) et quelle que soit leur répartition dans les adresses.

Répartition des adresses *Programme* 2

α		...	β
α_0	α_1	...	α_n
a_0	a_1	...	a_n

$k_1.\ C'\alpha \Rightarrow \alpha$

$k_2.\ P\{\ '\alpha \neq '\beta\ \}\ k_1\ ARR.$

Comme nous l'avons déjà mentionné, l'opération successeur C peut avoir une interprétation différente qui déterminera la forme de la formule k_1.

Exemples.

1) Si

$$C\alpha_i = \alpha_i + 1\ , \tag{5}$$

alors

$$k_1.\quad '\alpha + 1 \Rightarrow \alpha\ ; \tag{5'}$$

2) Si

$$\alpha_{i+1} = C\alpha_i = \alpha_i + d \tag{6}$$

(d est le pas de substitution d'adresse), alors

$$k_1. \quad '\alpha + d \Rightarrow \alpha ; \tag{6'}$$

3) Si

$$\alpha_{i+1} = C\alpha_i = \alpha_i + '\delta , \tag{7}$$

où $'\delta = d$, c'est-à-dire que le pas de substitution d'adresse est contenu dans l'adresse δ, alors

$$k_1. \quad '\alpha + '\delta \Rightarrow \alpha . \tag{7'}$$

Introduisons l'adresse a dans le programme 2 de la manière suivante :

Répartition des adresses *Programme* 3

α			β	$k_1.$ $^2\alpha \Rightarrow a$
α_0	α_1	...	α_n	$k_2.$ $C'\alpha \Rightarrow \alpha$
a_0	a_1	...	a_n	$k_3.$ $P\{'\alpha = '\beta\}\, k_1\, ARR.$

Les codes de la suite (1) sont examinés par le programme 3 : les codes (1) forment, successivement d'un cycle à l'autre, le contenu de l'adresse a. On appelle l'adresse *a standard*, et le programme 3, *programme d'examen des codes dans l'adresse standard.*

Une fois le programme 2 exécuté, l'adresse α_0 du code initial a_0 ne sera pas examinée.

Dans beaucoup de problèmes, il est souhaitable de conserver fixes les codes initiale et finale de la suite (1). Il est possible d'introduire un fixateur de travail supplémentaire γ.

Répartition des adresses *Programme* 4

α			β	$k_1.$ $'\alpha \Rightarrow \gamma$
α_0	α_1	...	α_n	$k_2.$ $C'\gamma \Rightarrow \gamma$
a_0	a_1	...	a_n	$k_3.$ $P\{'\gamma \neq '\beta\}\, k_2\, ARR.$

Il est possible de proposer une variante du programme d'examen des codes (1) avec un compteur. pour la formule de décalage du fixateur α de forme $C'\alpha = '\alpha + 1$. Introduisons l'adresse δ (au début $'\delta = 0$).

Répartition des adresses *Programme 5*

α					$k_1.$ $'('\alpha + '\delta) \Rightarrow a$
α_0	$\alpha_0 + 1$	...	$\alpha_0 + n$	δ	$k_2.$ $'\delta + 1 \Rightarrow \delta$
a_0	a_1	...	a_n	0	$k_3.$ $P\{'\delta \leqslant n\} k_1$ *ARR.*

Dans ce programme le fixateur α du début de la suite (1) ne se déplace pas. L'adresse δ joue le rôle de compteur du nombre de cycles.

2. Exemples de programme avec sous-programme d'examen de codes.

On rencontre souvent le programme qui examine la suite des codes dans les programmes de problèmes différents. Examinons-en quelques exemples :

***Exemple* 1.** *Multiplier par le scalaire b le vecteur de composantes* :

$$a_1, a_2, \ldots, a_n .$$

Répartition des adresses *Programme 6*

α			α_1		$k_1.$ $^2\alpha \times '\beta \Rightarrow '\alpha$
$\alpha_0 + 1$	$\alpha_0 + 2$	...	$\alpha_0 + n$	β	$k_2.$ $'\alpha + 1 \Rightarrow \alpha$
a_1	a_2	...	a_n	b	$k_3.$ $P\{'\alpha \leqslant '\alpha_1\} k_1$ *ARR.*

***Exemple* 2.** *Calculer la valeur de la fonction* $f(x)$ *pour* $x = x_0 + kh$ $(0 \leqslant k \leqslant n)$ *et placer dans les adresses* $\alpha_0 + k (0 \leqslant k \leqslant n)$.

Répartition des adresses *Programme 7*

α			α_1			$k_1.$ $f('\beta) \Rightarrow '\alpha$
α_0	$\alpha_0 + 1$	...	$\alpha_0 + n$	β	δ	$k_2.$ $'\beta + '\delta \Rightarrow \beta$
$f(x_0)$	$f(x_1)$	...	$f(x_n)$	x_0	h	$k_3.$ $\alpha + 1 \Rightarrow \alpha$
						$k_4.$ $P\{'\alpha \leqslant '\alpha_1\} k_1$ *ARR.*

***Exemple* 3.** *Relever les nombres positifs de la suite*

$$a_1, a_2, \ldots, a_n$$

et les placer dans la suite des adresses commençant à l'adresse β_0.

Répartition des adresses

α			α_1	β		
$\alpha_0 + 1$	$\alpha_0 + 2$	...	$\alpha_0 + n$	β_0	$\beta_0 + 1$	...
a_1	a_2	...	a_n	a_{i_1}	a_{i_2}	...

Programme 8

k_1. $P\{{}^2\alpha > 0\}\, k_2\, k_3$
k_2. ${}^2\alpha \Rightarrow {}'\beta$
${}'\beta + 1 \Rightarrow \beta$
k_3. ${}'\alpha + 1 \Rightarrow \alpha$
k_4. $P\{{}'\alpha \leqslant {}'\alpha_1\}\, k_1\ ARR$.

Citons l'exemple de l'utilisation des adresses du troisième rang pour examiner la succession des codes.

***Exemple* 4.** *Soient les codes*

$$a_1, a_2, \ldots, a_n$$

rangés dans les adresses

$$\alpha_1, \alpha_2, \ldots, \alpha_n,$$

pour lesquelles l'opération successeur n'est pas déterminée.

Introduisons alors la répartition des adresses suivante :

γ			γ_1	
$\beta + 1$	$\beta + 2$	...	$\beta + n$	
α_1	α_2	...	α_n	α
a_1	a_2	...	a_n	

Le programme d'examen des codes, dans l'adresse standard a, a la forme suivante :

Programme 9

k_1. ${}^3\gamma \Rightarrow a$
k_2. ${}'\gamma + 1 \Rightarrow \gamma$
k_3. $P\{{}'\gamma \leqslant {}'\gamma_1\}\, k_1\ ARR$.

***Exemple* 5.** *Calcul de la somme* $s = \sum_{i=1}^{n} \sum_{j=i}^{n} a_{ij}$.

On suppose que $a_{ij} = {}'(\alpha + (i-1)\,n + j)$ (rangement des éléments de la matrice en ligne). Le sens logique du problème consiste à examiner (ou à totaliser) les éléments « sus-diagonaux » de la matrice rectangulaire, c'est-à-dire les éléments disposés sur la diagonale et au-dessus de celle-ci. Introduisons les adresses : φ pour l'examen des éléments diagonaux de la matrice, ψ pour l'examen des éléments « sus-diagonaux » des lignes, χ pour les éléments de la dernière colonne de la matrice.

Pour débuter, nous supposerons que $'\varphi = \alpha + 1$,

$$'\chi = \alpha + n,\ '\gamma = 0,\ '\delta = n,\ '\overline{\chi} = \alpha + n^2 .$$

Répartition des adresses

$'(\alpha + (i-1)n + j) = a_{ij}$
$'\varphi = \alpha + 1$
$'\chi = \alpha + n$
$'\gamma = 0 \mid s$
$'\delta = n$
$'\overline{\chi} = \alpha + n^2$
ψ, Adresse de travail

Programme 10

k_1. $'\varphi \Rightarrow \psi$;
k_2. ${}^2\psi + '\gamma \Rightarrow \gamma$;
$'\psi + 1 \Rightarrow \psi$;
$P\{'\psi \leqslant '\chi\}\, k_2$;
$'\varphi + '\delta + 1 \Rightarrow \varphi$; $'\chi + '\delta \Rightarrow \chi$;
$P\{'\chi \leqslant '\overline{\chi}\}\, k_1\ ARR$.

Si, maintenant, on veut placer l'élément sus-diagonal de plus grand module de la matrice donnée (a_{ij}) dans la cellule γ, il est suffisant de remplacer la formule k_2 par la formule $\max\{'\gamma, {}^2\psi\} \Rightarrow \gamma$ dans le programme cité.

***Exemple* 6.** *Placer respectivement dans les cellules* $\beta+1, \beta+2, \ldots, \beta+k$ $(\beta + 1 > \alpha + k)$ *le m-ième degré de la permutation* $i_1, i_2, \ldots, i_k$ — *des nombres* $1, 2, \ldots, k$, *placés initialement dans les cellules* $'(\alpha + j) = i_j$ $(j = 1, 2, \ldots, k)$.

Introduisons les adresses φ pour l'examen des éléments de la substitution initiale, ψ pour les éléments de son m-ième rang. Supposons pour commencer que $'\varphi = \alpha + 1$, $'\overline{\varphi} = \alpha + n$, $'\psi = \beta + 1$.

Répartition des adresses

$'(\alpha + j) = i_j \quad (j = 1, 2, \ldots, k)$

Substitution initiale

$'\varphi = \alpha + 1$
$'\overline{\varphi} = \alpha + n$
$'\psi = \beta + 1$
$'(\beta + 1)$, …, $'(\beta + k)$ — m-ième rang de la substitution initiale .

Programme 11

k_1. $\underbrace{'(\ldots '('(}_{m \text{ fois}}{}^2\varphi + \alpha) + \alpha) + \cdots \cdots + \alpha) \Rightarrow '\psi$;
$'\varphi + 1 \Rightarrow \varphi$; $'\psi + 1 \Rightarrow \psi$;
$P\{'\varphi \leqslant '\overline{\varphi}\}\, k_1\ ARR.$

3. PROGRAMMES DE PROBLÈMES D'ALGÈBRE LINÉAIRE

Dans ce paragraphe nous illustrerons par une série d'exemples la méthode qui permet de construire des programmes à adresses pour résoudre des problèmes d'algèbre linéaire.

Dans les problèmes les plus divers, l'examen de l'information peut être réduit aux schémas de l'examen de l'information caractéristique des problèmes d'algèbre linéaire.

Les scalaires, les vecteurs ou les matrices peuvent servir de données d'entrée ou de sortie dans les problèmes d'algèbre linéaire. Dans le cas de données vectorielles, on considérera que leurs composantes sont données sous forme de suites, c'est-à-dire réparties dans des suites d'adresses. Dans le cas de données matricielles, on estimera que les éléments des matrices sont répartis sur des lignes et dans des colonnes. Dans le premier cas, la grandeur

$$\alpha + (i - 1)\, n + j\,,$$

où n est le nombre des colonnes de la matrice et α une constante définissant l'adresse initiale du bloc, servira d'adresse à l'élément placé à la i-ième ligne de la j-ième colonne. Dans le second cas, cette adresse est définie par

$$\alpha + (j - 1)\, m + i\,,$$

m étant le nombre de lignes de la matrice.

Puisqu'en algèbre linéaire, on peut réduire la résolution des problèmes complexes à l'emploi de sous-programmes qui représentent des programmes permettant de résoudre des problèmes plus élémentaires, il est rationnel que ces derniers aient des exigences standards. Ces exigences sont celles déjà formulées sur la répartition des données sous forme vectorielle et matricielle. Imposons encore que les fixateurs de travail, qui sont utilisés dans des sous-programmes séparés, leur soient communs : $\psi_1, \psi_2, \psi_3, \ldots$.

Par exemple si à l'entrée

le premier bloc est	$\alpha + 1, \alpha + 2, \ldots$
le second bloc,	$\beta + 1, \beta + 2, \ldots$

et à la sortie

le troisième bloc	$\gamma + 1, \gamma + 2, \ldots,$

on considérera pour le travail du sous-programme que les fixateurs $\psi_1, \psi_2, \psi_3, \ldots$ doivent fixer le début de ces blocs :

$$'\psi_1 = \alpha + 1\,,$$

$$'\psi_2 = \beta + 1\,,$$

$$'\psi_3 = \gamma + 1\,.$$

Dans le cas particulier où l'un des blocs représente une cellule (le scalaire est une grandeur d'entrée ou de sortie), nous considérerons que cette grandeur est directement conservée dans l'adresse ψ_i.

Ainsi, nous ne donnerons que des fixateurs standards ou des adresses (dans le cas de grandeurs scalaires) pour les blocs d'entrée et de sortie ; les mêmes blocs fixés par ces fixateurs peuvent être répartis en suites arbitraires.

Nous introduirons aussi des cellules standards $\xi_1, \xi_2, \ldots$ pour les paramètres qui caractérisent les dimensions des vecteurs ou des matrices.

***Exemple* 1.** *Soustraction de vecteurs*

$$X - Y = Z .$$

Soit ψ_1 le fixateur de la première composante de X, ψ_2 le fixateur de la première composante de Y, ψ_3 contenant l'adresse affectée à la première composante de Z, $'\xi = n$ la dimension des vecteurs X, Y, Z.

Programme 1

$$k_{11}. \quad 0 \Rightarrow \delta ;$$

$$k_{12}. \quad '('\psi_1 + '\delta) - '('\psi_2 + '\delta) \Rightarrow '\psi_3 + '\delta ;$$

$$'\delta + 1 \Rightarrow \delta ;$$

$$P \{ '\delta < '\xi \} k_{12} \text{ Я} .$$

***Exemple* 2.** *Multiplication du vecteur par un scalaire*

$$Xa = Y .$$

Soit ψ_1 le fixateur de la première composante de X, $'\psi_2 = a$, $''\xi = n$ la dimension des vecteurs X, Y, ψ_3 contenant l'adresse affectée à la première composante de Y.

Programme 2

$$k_{21}. \quad 0 \Rightarrow \delta ;$$

$$k_{22}. \quad '('\psi_1 + '\delta)\, '\psi_2 \Rightarrow '\psi_3 + '\delta ;$$

$$'\delta + 1 \Rightarrow \delta ;$$

$$P \{ '\delta < '\xi \} k_{22} \text{ Я} .$$

***Exemple* 3.** *Calcul du carré du module d'un vecteur*

$$\| X \|^2 = a .$$

Soit ψ_1 le fixateur de la première composante du vecteur X, ψ_2 l'adresse dans laquelle est rangée le résultat, $'\xi = n$ la dimension du vecteur X.

Programme 3

$$k_{31}. \quad 0 \Rightarrow \delta ; \quad 0 \Rightarrow \psi_2 ;$$

$$k_{32}. \quad '('\psi_1 + '\delta)^2 + '\psi_2 \Rightarrow \psi_2 ;$$

$$'\delta + 1 \Rightarrow \delta ;$$

$$P \{ '\delta < '\xi \} k_{32} \text{ Я} .$$

***Exemple* 4.** *Multiplication de la matrice carrée A d'ordre n par le vecteur X.*

$$AX = Y .$$

Soit ψ_1 le fixateur du premier élément de la matrice A rangée ligne par ligne, ψ_2 le fixateur du premier élément du vecteur X, ψ_3 contenant l'adresse affectée à la première composante de Y, $'\xi = n$.

Programme 4

$$k_{41}.\quad 0 \Rightarrow \delta_1 \ ;\quad 0 \Rightarrow \delta_3 \ ;$$

$$k_{42}.\quad 0 \Rightarrow \delta_2 \ ;\quad 0 \Rightarrow '\psi_3 + '\delta_3 \ ;$$

$$k_{43}.\quad '('\psi_1 + '\delta_1) \times '('\psi_2 + '\delta_2) + '('\psi_3 + '\delta_3) \Rightarrow '\psi_3 + '\delta_3 \ ;$$

$$'\delta_1 + 1 \Rightarrow \delta_1 \ ;\quad '\delta_2 + 1 \Rightarrow \delta_2 \ ;$$

$$P\{\ '\delta_2 < '\xi\ \}\, k_{43} \ ;$$

$$'\delta_3 + 1 \Rightarrow \delta_3 \ ;$$

$$P\{\ '\delta_3 < '\xi\ \}\, k_{42}\ \mathfrak{A} .$$

***Exemple* 5.** *La matrice A (carrée, d'ordre n) est rangée ligne par ligne. Multiplier la matrice A* transposée de A par le vecteur X :*

$$A^* X = Y .$$

Nous utiliserons les désignations de l'exemple 4.

Programme 5

$$k_{51}.\quad 0 \Rightarrow \delta_1 \ ;\quad 0 \Rightarrow \delta_3 \ ;$$

$$k_{52}.\quad 0 \Rightarrow \delta_2 \ ;\quad 0 \Rightarrow '\psi_3 + '\delta_3 \ ;$$

$$k_{53}.\quad '('\psi_2 + '\delta_1) \times '('\psi_2 + '\delta_2) + '('\psi_3 + '\delta_3) \Rightarrow '\psi_3 + '\delta_3 \ ;$$

$$'\delta_1 + '\xi \Rightarrow \delta_1 \ ;\quad '\delta_2 + 1 \Rightarrow \delta_2 \ ;$$

$$P\{\ '\delta_2 < '\xi\ \}\, k_{53} \ ;$$

$$'\delta_3 + 1 \Rightarrow \delta_3 \ ;\quad '\delta_1 - '\xi('\xi - 1) + 1 \Rightarrow \delta_1 \ ;$$

$$P\{\ '\delta_3 < '\xi\ \}\, k_{52}\ \mathfrak{A} .$$

Dans tous les exemples cités, on réalise l'examen de l'information en utilisant des compteurs ; le contenu des fixateurs employés dans le programme ne change pas.

Introduisons maintenant un exemple de problèmes plus complexes, dont la résolution est presque intégralement exprimée par les programmes 1 à 5 cités.

***Exemple* 6.** *Résolution d'un système d'équations algébriques linéaires par la méthode de Friedman* (cf. Friedman, V. M. [1]).

Le système d'équations algébriques linéaires

$$AX = F,$$

où $A = (a_{ij})$ est la matrice des coefficients du système, $F = (f_1, \ldots, f_n)$ le vecteur libre et X le vecteur cherché, peut être résolu au moyen du processus itératif suivant.

Soit X^0 un vecteur (arbitraire) des approximations d'ordre zéro. Alors le vecteur de la $(k+1)$-ième approximation X^{k+1} est défini par les formules :

$$X^{k+1} = X^k - a_{k+1} A^*(AX^k - F),$$

où

$$a_{k+1} = \frac{\| AX^k - F \|^2}{\| A^*(AX^k - F) \|^2},$$

A^* est la matrice transposée. Le processus itératif s'arrête lorsque

$$\| AX^{k+1} - F \|$$

devient inférieur à un nombre donné ε (soit $'\omega = \varepsilon$), et les valeurs de X^{k+1} sont prises comme solutions.

Soit la matrice initiale A rangée ligne par ligne :

$$'(\alpha + (i-1)n + j) = a_{ij} \quad (i, j = 1, 2, \ldots, n) ;$$

le vecteur F est donné sous forme de suite

$$'(\theta + i) = f_i \quad (i = 1, 2, \ldots, n).$$

Le vecteur des approximations d'ordre zéro X^0 est aussi donné sous forme de suite

$$'(\beta + i) = x_i^0 .$$

Séparons n cellules de travail $\delta + 1, \ldots, \delta + n$, dans lesquelles on placera au cours du calcul les composantes des vecteurs :

$$AX^k, AX^k - F, \quad A^*(AX^k - F).$$

Pour examiner l'information, introduisons les adresses de second rang suivantes :

$$'\gamma_1 = \alpha + 1 ; \quad '\gamma_4 = \theta + 1 ;$$

$$'\gamma_2 = \beta + 1 ; \quad '\overline{\varphi} = \delta + n ;$$

$$'\gamma_3 = \delta + 1 ; \quad \text{de plus} : \quad '\xi = n ; \quad '\omega = \varepsilon .$$

Introduisons le concept de *formule d'entrée à adresses*

$$\pi\alpha\beta\,.$$

Par *formules d'entrée* nous comprenons le passage à l'exécution de la formule à adresses, à laquelle l'index α sert de désignation. Lorsque le signe $\mathfrak{A}$ apparaît, il y a retour à l'exécution de la formule à laquelle l'index β sert de désignation. Autrement dit, la formule d'entrée signifie que l'on passe au sous-programme auquel l'instruction α sert d'instruction initiale. Après son exécution, on retourne à l'instruction β.

Convenons, en outre, de supprimer l'index β s'il désigne l'instruction du programme suivant.

Programme 6

	Chargement des fixateurs de travail des sous-programmes			Transfert de la commande à un sous-programme
	ψ_1	ψ_2	ψ_3	
N_1. $'\gamma_1 \Rightarrow \psi_1\,;\ '\gamma_2 \Rightarrow \psi_2\,;\ '\gamma_3 \Rightarrow \psi_3$ πk_{41}	$'\gamma_1$	$'\gamma_2$	$'\gamma_3$	$AX = Y$
$'\gamma_3 \Rightarrow \psi_1\,;\ '\gamma_4 \Rightarrow \psi_2$ πk_{11}	$'\gamma_3$	$'\gamma_4$	$'\gamma_3$	$X - Y = Z$
N_2. $P\{\,\lvert\,^2\psi_3\,\rvert > '\omega\,\}\,N_3$ $'\psi_3 + 1 \Rightarrow \psi_3$ $P\{\,'\psi_3 \leqslant '\overline{\varphi}\,\}\,N_2\,N_4$ N_3. πk_{31}	$'\gamma_3$	$\|\ \ \|^2$		$\|\,X\,\|^2$
$'\psi_2 \Rightarrow r$ $'\gamma_1 \Rightarrow \psi_1\,;\ '\gamma_3 \Rightarrow \psi_2\,;\ '\gamma_3 \Rightarrow \psi_3$ πk_{51}	$'\gamma_1$	$'\gamma_3$	$'\gamma_3$	$A^*\,X = Y$
$'\gamma_3 \Rightarrow \psi_1$ πk_{31}	$'\gamma_3$	$\|\ \ \|^2$		$\|\,X\,\|^2$
$'r : '\psi_2 \Rightarrow \psi_2$ πk_{21}	$'\gamma_3$	a_k	$'\gamma_3$	$Xa = Y$
$'\gamma_2 \Rightarrow \psi_1\,;\ '\gamma_3 \Rightarrow \psi_2\,;\ '\gamma_2 \Rightarrow \psi_3$ $\pi k_{11}\,N_1$	$'\gamma_2$	$'\gamma_3$	$'\gamma_2$	$X - Y = Z$
N_4. $\pi k_{61}\,\mathfrak{A}$	$'\gamma_2$			à un sous-programme de transfert de groupe et d'impression.

Lorsqu'il y a des sous-programmes standards pour les opérations fondamentales d'algèbre linéaire, la programmation des problèmes d'algèbre linéaire revient à construire des programmes pour diriger le travail des sous-programmes. Comme l'exemple le montre, le programme principal est composé du transfert de la commande vers un sous-programme. De plus, la préparation du travail des sous-programmes réside dans le chargement correspondant des fixateurs de travail des sous-programmes.

Ici, on suppose que tous les sous-programmes utilisent la cellule ξ qui contient l'ordre du système initial n.

Outre les programmes 1 à 5 que nous avons cités, on utilise dans le programme 6 un sous-programme de transfert de groupe et d'impression dont l'instruction initiale est k_{61}.

Examinons l'exemple rencontré dans les problèmes d'algèbre linéaire, duquel le calcul arithmétique est entièrement absent.

***Exemple* 7.** *Trouver le mineur* $C = (C_{ij})$ *de l'élément* a_{ks} *de la matrice carrée* $A = (a_{ij})$. *Supposons que la matrice soit rangée ligne par ligne* :

$$'(\alpha + (i-1)n + j) = a_{ij} \quad (i, j = 1, 2, \ldots, n),$$

les nombres k et s sont donnés. Le mineur obtenu est rangé ligne par ligne dans les cellules $\beta + 1, \beta + 2, \ldots, \beta + (n-1)^2$.

Si φ examine les éléments de la matrice en commençant par $'\varphi = \alpha + 1$, $\overline{\varphi}$ examine les éléments de la dernière colonne en commençant par $'\overline{\varphi} = \alpha$, $\overline{\overline{\varphi}}$ fixe le dernier élément de la matrice :

$$'\overline{\overline{\varphi}} = \alpha + n^2,$$

χ examine les éléments de la colonne supprimée initialement $'\chi = \alpha + s - n$, ω fixe le premier élément de la ligne supprimée :

$$'\omega = \alpha + (k-1)n + 1,$$

ψ examine les adresses du mineur de la matrice.

Prenons l'ordre suivant pour le travail de l'algorithme : les éléments de la matrice donnée sont successivement examinés ligne par ligne ; les éléments qui n'appartiennent ni à la ligne ni à la colonne supprimées sont transférés dans la suite correspondante.

Répartition des adresses

$'(\alpha + (i-1)n + j) = a_{ij}$ $(i, j = 1, 2, \ldots, n)$

$'\varphi = \alpha + 1$

Programme 7

R_1. $P\{'\varphi \neq '\omega\} R_2$

$'\varphi + n \Rightarrow \varphi$; $'\overline{\varphi} + n \Rightarrow \overline{\varphi}$; $'\chi + n \Rightarrow \chi$.

$$'\overline{\varphi} = \alpha$$
$$'\overline{\overline{\varphi}} = \alpha + n^2$$
$$'\chi = \alpha + s - n$$
$$'\omega = \alpha + (k-1)n + 1$$
$$'\psi = \beta + 1$$
$$'(\beta + (i-1)n + j) = c_{ij} \qquad (i, j = 1, 2, \ldots, n-1)$$

R_2. $P\{\,'\varphi > '\overline{\overline{\varphi}}\,\}\,\mathfrak{A}$
$'\varphi + n \Rightarrow \varphi$; $'\chi + n = \chi$;
R_3. $P\{\,'\varphi = '\chi\,\}\,R_4$
$^2\varphi \Rightarrow '\psi$
$'\psi + 1 \Rightarrow \psi$
R_4. $'\varphi + 1 \Rightarrow \varphi$
$P\{\,'\varphi > '\overline{\varphi}\,\}\,R_1\,R_3$.

EXERCICES

Etablir les programmes à adresses :

1. Calcul du cosinus de l'angle formé par deux vecteurs.

2. Multiplication de matrices.

3. Calcul de la trace d'une matrice.

4. Calcul du carré d'une matrice.

5. Résolution d'un système d'équations linéaires par la méthode de Seidel.

6. Résolution du système par la méthode de la simple itération.

7. Résolution du système dans le cas d'une matrice symétrique par la méthode des racines carrées (En mémoire, on ne conservera ligne par ligne que les éléments de la diagonale principale et les éléments de la diagonale située immédiatement au-dessus de la diagonale principale).

8. Résolution d'un système d'équations algébriques linéaires par la méthode de l'élimination avec choix de l'élément principal (cf. Faddeieva, V. N. [1]).

9. Calcul du déterminant en substituant des zéros dans la colonne qui contient l'élément de plus grand module (cf. Faddeieva, V. N. [1]).

4. PROBLÈMES NON ARITHMÉTIQUES

Nous étudierons dans ce paragraphe les problèmes logiques : les schémas qui permettent de les résoudre sont bien connus ou facilement définis, mais difficilement précisés sous forme de programme. Si la logique employée par l'homme pour résoudre un problème est connue, encore cela ne signifie-t-il pas que l'algorithme de résolution du problème soit défini au point de pouvoir l'exécuter en machine.

Seule une mise en forme exacte de toutes les étapes de la résolution des problèmes peut assurer le succès de la programmation en C. A. N. C'est par la programmation à adresses que l'on peut atteindre ce degré de mise en forme

du processus de résolution des problèmes. De plus, comme cela a été montré dans le chapitre précédent, le passage des programmes à adresses aux programmes pour les C. A. N. actuelles ne présente pas de sérieuses difficultés et peut être automatisé. Dans ce paragraphe nous examinerons les problèmes suivants :

1) Conversion des formules arithmétiques en une écriture sans parenthèses.
2) Exécution des algorithmes normaux (algorithmes de Markov) sur C. A. N. (cf. Markov, A. A. [1]).
3) Différentiation des expressions dans les fonctions élémentaires.
4) Réduction des formules de calcul des propositions à la forme normale.

1. Conversion des formules arithmétiques qui ont des opérations à deux places en écriture sans parenthèses.

Enoncé du problème. Examinons des expressions contenant des grandeurs que nous désignerons par des minuscules latines a, b, ..., et des opérations à deux positions (binaires) (par exemple, l'addition, la soustraction, la multiplication, etc.) que nous désignerons par la lettre θ ou par les symboles $+$, $-$, $\times$, etc. .

Les formules sont définies dans l'écriture avec parenthèses ouvertes [le signe en est « (»] et fermées [le signe en est «) »] de la manière suivante :

Définition 1. L'expression $(a)\ \theta\ (b)$, dans laquelle a et b sont des grandeurs, est appelée formule élémentaire. L'expression $(A)\ \theta\ (B)$, dans laquelle A et B sont des formules, est appelée formule.

Dans l'écriture sans parenthèses les formules sont ainsi définies :

Définition 2. L'expression θab, dans laquelle a et b sont des grandeurs, est appelée formule élémentaire. L'expression θAB, dans laquelle A et B sont des formules, est appelée formule.

Par exemple, l'expression

$$(((a) + (b)) \times (c)) - ((a) \times ((b) + (c)))$$

est une formule arithmétique qui a des opérations à deux places.

La même formule dans l'écriture sans parenthèses a la forme :

$$- \times + abc \times a + bc\,.$$

L'ordre d'application des opérations dans l'écriture de la formule sans parenthèses est facile à décrire verbalement. Dans notre exemple, la formule dans l'écriture sans parenthèses peut se lire : différence entre le produit de la somme de a et de b par c et le produit de a par la somme de b et de c. Remarquons que la définition de l'écriture des formules avec parenthèses que nous avons adoptée conduit à des expressions plus volumineuses que les règles habituelles de l'écriture des formules arithmétiques. Au contraire, les formules

selon la définition donnée plus haut sont plus simples pour analyser et construire l'algorithme de conversion en écriture sans parenthèses.

La conversion de l'écriture des formules avec parenthèses en écriture sans parenthèses peut être effectuée par deux procédés.

Le premier procédé consiste à rechercher successivement dans la formule écrite avec parenthèses la première formule élémentaire à gauche de forme $(a)\,\theta\,(b)$, et de l'écrire sans parenthèses θab. Ensuite, on répète ce processus jusqu'à ce que toute la formule soit écrite sans parenthèses.

Dans l'exemple cité, la succession des transformations se présentera ainsi :

1er pas $((+\ ab) \times (c)) - ((a) \times ((b) + (c)))$

2e pas $(\times\ +\ abc) - ((a) \times ((b) + (c)))$

3e pas $(\times\ +\ abc) - ((a) \times (+\ bc))$

4e pas $(\times\ +\ abc) - (\times\ a\ +\ bc)$

5e pas $-\ \times\ +\ abc \times a\ +\ bc$, c'est le résultat cherché.

Le second procédé consiste à considérer la formule donnée ou écrite avec parenthèses comme l'expression $(A)\,\theta\,(B)$; on recherche l'opération correspondant à une telle représentation de la formule et cette formule compliquée est transcrite en écriture sans parenthèses θAB. Ensuite, chaque expression A et B qui apparaît à son tour comme une formule de forme $(A_1)\,\theta\,(A_2)$ et $(B_1)\,\theta\,(B_2)$ est transcrite en écriture sans parenthèses etc.

L'exemple de la formule cité est, dans ce cas, transformé en écriture sans parenthèses de la façon suivante :

1er pas $-\ ((a) + (b)) \times (c)\,(a) \times ((b) + (c))$

2e pas $-\ \times\ (a) + (b)\ c \times a(b) + (c)$

3e pas $-\ \times\ +\ abc \times a\ +\ bc$, c'est le résultat cherché.

Construisons l'algorithme de conversion des formules en écriture sans parenthèses suivant la deuxième règle. Il est indispensable avant tout de préciser le contenu du travail de l'algorithme. A cet effet, il est nécessaire de savoir trouver dans l'expression examinée l'opération qui la définit comme formule $(A)\,\theta\,(B)$. La règle pour la découvrir résulte directement de la définition de la formule dans l'écriture avec parenthèses : *avant l'opération qui définit la formule, le nombre de parenthèses ouvertes est égal au nombre de parenthèses fermées.*

Codification de l'information et programmation. L'information initiale pour l'algorithme de conversion des formules dans l'écriture sans parenthèses est constituée des éléments suivants : parenthèses ouvertes « (». parenthèses

fermées «) », grandeurs a, b, ..., opérations θ. Le code des parenthèses doit contenir deux symboles : le symbole de la parenthèse (le signe /) et le symbole de la forme de la parenthèse, ouverte ou fermée [respectivement les signes « (» et «) »]. Les codes des grandeurs et les codes des opérations contiennent aussi chacun deux symboles. Les codes des grandeurs contiennent le symbole de la grandeur et le symbole du numéro d'ordre de la grandeur ; les codes d'opérations contiennent le symbole de l'opération et le numéro d'ordre de l'opération.

Pour effacer les parenthèses, nous introduirons le code « vide » Λ. La formule, c'est-à-dire la suite des codes des éléments de la formule qui sont rangés dans un ordre déterminé, est l'objet initial de l'algorithme. Comme cela a été montré au paragraphe 2 du présent chapitre, nous fixerons l'ordre de succession des codes des éléments de la formule en les répartissant dans les adresses

$$\alpha_1, \alpha_2, \alpha_3, \ldots ,$$

pour lesquelles l'opération successeur C est déterminée :

$$C\alpha_i = \alpha_{i+1} \quad (i = 1, 2, \ldots) .$$

Nous fixerons le début de la formule par l'adresse de second rang (le fixateur du début) $\varphi_{\text{déb.}}$. A la fin de la formule nous mettrons le signe ! pour marquer la fin de la formule. Introduisons de plus dans l'adresse α un compteur du nombre de parenthèses ouvertes et de parenthèses fermées. De cette manière, la répartition initiale des adresses a la forme suivante :

Répartition des adresses

$\varphi_{\text{déb.}}$					
α_1	α_2	α_3	...		α
Codes des éléments de la formule				!	0

L'algorithme est constitué de deux parties :

1) Recherche de l'opération qui détermine la formule.
2) Passage à l'écriture sans parenthèses de la formule.

Programme 1. *Conversion des formules en écriture sans parenthèses.*

k_1. $\quad '\varphi_{\text{déb.}} \Rightarrow \varphi_0$

k_2. $\quad P\{^2\varphi_0 \Rightarrow (\}\, k_3$

$\quad C'\varphi_0 \Rightarrow \varphi_0$

$\quad P\{^2\varphi_0 \neq !\}\, k_2\, k_9$

Le début du travail de l'algorithme est de rechercher la première parenthèse ouverte de la formule. On examine successivement avec le fixateur φ_0 les éléments de la formule jusqu'à ce qu'on trouve la première parenthèse ouverte, qui est alors fixée

k_3. $'\varphi_0 \Rightarrow \varphi_1$
k_4. $P\{{}^2\varphi_1 = /\}\, k_6$
k_5. $C'\varphi_1 \Rightarrow \varphi_1$
Pk_4
k_6. $P\{{}^2\varphi_1 = (\}\, k_7$
$'\alpha - 1 \Rightarrow \alpha$
$P\{'\alpha \neq 0\}\, k_5$
$C\,'\varphi_1 \Rightarrow \varphi_1$
Pk_8
k_7. $'\alpha + 1 \Rightarrow \alpha$
Pk_5

par φ_0. Une fois qu'elle est fixée, la commande passe à l'opérateur k_3. La deuxième étape de l'algorithme consiste à calculer le nombre des parenthèses ouvertes et fermées en commençant à ${}^2\varphi_0$. On examine successivement les éléments de la formule après φ_0 et on cherche les parenthèses avec le fixateur φ_1. Si φ_1 fixe une parenthèse ouverte, la commande est transmise à l'opérateur k_7, on ajoute 1 à α. Si φ_1 fixe une parenthèse fermée, on retranche 1 de α et le contenu de α est comparé à 0. Tant que $\alpha \neq 0$ le calcul des parenthèses se poursuit ; mais si $'\alpha = 0$ il y a passage à la seconde partie de l'algorithme dont le programme commence à l'opérateur k_8.

Avant de construire le programme de la deuxième partie de l'algorithme, remarquons que le passage de l'écriture $(A)\,\theta\,(B)$ à l'écriture θAB signifie qu'il faut transférer le signe de l'opération à la place de la première parenthèse ouverte et effacer les parenthèses qui se trouvent à gauche et à droite de ce signe. En outre, il faut trouver la *dernière* parenthèse fermée de la formule et l'effacer aussi. Cette étape du travail de l'algorithme — effacement de la dernière parenthèse fermée de la formule — est assez compliquée puisqu'il faut, en fait, refaire une recherche analogue à celle de l'opération qui définit la formule. Mais, notons que, s'il reste une dernière parenthèse fermée dans la formule, la règle de recherche de l'opération qui définit la formule ne change pas. C'est pourquoi, on peut simplifier l'algorithme en passant en deux fois à l'écriture sans parenthèses. Il faut d'abord passer de l'expression $(A)\,\theta\,(B)$ à l'expression θAB). Après avoir écrit toutes les opérations sous la forme sans parenthèses, il faut effacer toutes les parenthèses fermées qui restent. L'absence dans la formule de parenthèses ouvertes peut servir de signe de fin d'écriture de toutes les opérations, ce qui est défini par le prédicat $P\{{}^2\varphi_0 = !\}$.

k_8. ${}^2\varphi_1 = '\varphi_0$
$C\,'\varphi_1 \Rightarrow \varphi_2$
$\Lambda \Rightarrow '\varphi_2$
$\overline{C}\,'\varphi_1 \Rightarrow \varphi_2$
$\Lambda \Rightarrow '\varphi_2$
$\Lambda \Rightarrow '\varphi_1$
Pk_2
k_9. $'\varphi_{\text{déb.}} \Rightarrow \varphi$
k_{10}. $P\{{}^2\varphi = !\}\, ARR$
$P\{{}^2\varphi \neq /\}\, k_{11}$
$\Lambda \Rightarrow '\varphi$
k_{11}. $C\,'\varphi \Rightarrow \varphi$
Pk_{10}

Construisons donc le programme de transfert de l'écriture $(A)\,\theta\,(B)$ à l'écriture θAB.

Le code de l'opération, fixé par φ_1, est transféré à la place de la première parenthèse ouverte, fixée par φ_0. Les parenthèses, les opérations de code voisin et le code de l'opération sont ensuite effacés. L'étape finale de l'algorithme — effacement des parenthèses fermées qui restent — débute par l'opérateur k_9. Des explications supplémentaires concernant ne sont pas utiles.

Nous recommandons au lecteur d'établir le programme qui transcrit la formule obtenue en écriture sans parenthèses en une nouvelle suite d'adresses sans code (vide) Λ.

2. Réalisation sur C. A. N. des algorithmes normaux de Markov.

Commençons par expliquer le sens du concept d'*algorithme normal* introduit par A. A. Markov (pour plus de détails nous renverrons le lecteur à la monographie de A. A. Markov : *Théorie de l'algorithme*. Travaux de l'Institut de Mathématiques, Académie des Sciences de l'U. R. S. S., tome 42, 1954). Les mots, suites de symboles élémentaires appelés lettres d'un certain alphabet, servent d'information initiale à l'algorithme normal. L'algorithme remanie le mot initial pour en faire un autre mot, résultat de son action. L'opération de *substitution* est adoptée comme opération élémentaire de la transformation des mots. Soient un mot X et un mot A. On dit que le mot A *entre* dans le mot X, si X est représenté sous la forme $X = YAC$. Pour les mots X et A une telle représentation peut, naturellement, avoir plusieurs valeurs. L'*entrée* du mot A dans le mot X est appelée *première*, si le mot Y dans la représentation $YAC = X$ est le plus court. Ici, l'opération de substitution est définie ainsi : Soient le mot X et la substitution $A \to B$. Si $X = YAC$, l'*application de l'opération de substitution* $A \to B$ au mot X transfert le mot X dans le mot YBC.

L'*algorithme normal* est défini par le schéma constitué par la suite de formules du type :

$$(\mathfrak{A}) \quad \begin{array}{l} A_1 \to B_1 \\ A_2 \to B_2 \\ \cdots\cdots\cdots \\ A_n \to B_n \,. \end{array}$$

Le processus des substitutions successives, appliquées sur la base du schéma d'algorithme donné $(\mathfrak{A})$ au mot X, est réalisé par la règle suivante : (1) On cherche la première formule (d'en haut) de substitution du schéma $(\mathfrak{A})$ telle que sa partie gauche entre dans le mot examiné X, par exemple la substitution $A_k \to B_k$; (2) B_k est mis à la place de la première entrée A_k dans le mot X, ce qui donne le mot X' ; (3) la règle qui vient d'être formulée est appliquée de nouveau au mot X', etc. Si, à une étape, aucune des parties gauches des formules du schéma n'entre dans X, le processus s'arrête et le mot que l'on a obtenu est considéré comme le résultat de l'application de l'algorithme normal avec le schéma considéré $(\mathfrak{A})$. La fin de l'algorithme peut avoir lieu aussi dans un autre cas, lorsque la substitution du schéma $(\mathfrak{A})$ notée comme formule finale est appliquée. De telles formules finales sont inscrites avec une flèche suivie d'un point : marque de fin de mot : $A \to\cdot B$.

Examinons, par exemple, l'algorithme normal pour calculer la valeur absolue de la différence de deux nombres naturels. Les nombres naturels n et m sont notés dans un alphabet ne contenant qu'un seul signe 1, c'est-à-dire que chaque nombre naturel est représenté par le nombre de signes égal au nombre d'unités qui s'y trouvent. Le couple des nombres n et m s'écrit avec le signe disjonctif $*$: $n * m$. L'algorithme normal correspondant est donné par le schéma

$$| * | \rightarrow *$$

$$* \rightarrow$$

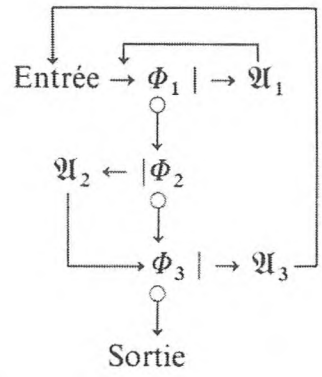
La formulation alphabétique du contenu du travail d'un algorithme normal peut être précisée à l'aide d'un organigramme. Introduisons les opérateurs $\mathfrak{A}_k$ qui effectuent la substitution $A_k \rightarrow B_k$, c'est-à-dire qui, dans l'application au mot X, substituent B_k à la première partie A_k du mot X. Introduisons les tests Φ_k qui déterminent l'entrée du mot A_k dans le mot X examiné. L'organigramme de l'algorithme normal

$$A_1 \rightarrow B_1$$

$$A_2 \rightarrow \cdot B_2$$

$$A_3 \rightarrow B_3$$

prendra alors la forme :



Passons maintenant à la structure du programme à adresses pour la forme normale de l'algorithme de A. A. Markov. Le mot examiné X, constitué d'une succession ordonnée de lettres, et les formules de substitution $\{ A_k \rightarrow B_k \}$, données dans un ordre déterminé, servent d'information initiale. Conformément à l'organigramme de l'algorithme normal, le programme est constitué par le test Φ_k de la première entrée du mot A_k dans le mot X examiné et par les opérateurs de substitution $\mathfrak{A} \{ A_k \rightarrow B_k \}$ de B_k à la place de la première entrée du mot A_k dans le mot X de la partie gauche de la formule.

L'algorithme qui teste la première entrée du mot A dans le mot X est réalisé de la façon suivante : on cherche dans le mot X la première lettre du mot A, on contrôle ensuite la concordance du mot A et du mot qui se compose de la suite des lettres de X commençant par la première lettre du mot A. S'il y a concordance, on a trouvé la première entrée du mot A dans le mot X, sinon on cherche à nouveau dans le mot X l'entrée suivante de la première lettre du mot A, etc.

Pour décrire, au moyen d'un programme, l'algorithme qui teste la première entrée du mot A dans le mot B nous introduirons les fixateurs :

$\varphi_{\text{déb.}}$ fixe le début du mot X (à la fin du mot, comme c'est normal, il y a le signe !).

$\varphi_{\text{déb.}}$ fixe le début du mot A.

Nous disposerons les codes des lettres du mot X et les codes de substitution $A \to B$ dans les adresses mises en ordre par l'opération successeur C.

Programme 2.1. *Test de la première partie du mot A dans le mot X.*

k_1. $'\psi_{\text{déb.}} \Rightarrow \psi_1$
$'\varphi_{\text{déb.}} \Rightarrow \varphi_1$

k_2. $P\{{}^2\varphi_1 = {}^2\psi_1\}k_3$
$C\,'\varphi_1 \Rightarrow \varphi_1$
$P\{{}^2\varphi_1 \neq !\}k_2\ ARR.$

k_3. $'\varphi_1 \Rightarrow \varphi_2$
$'\psi_1 \Rightarrow \psi_2$

k_4. $C\,'\varphi_2 \Rightarrow \varphi_2$
$C\,'\psi_2 \Rightarrow \psi_2$
$P\{{}^2\psi_2 = \to\}k_6$
$P\{{}^2\varphi_2 = {}^2\psi_2\}k_4$

k_5. $C\,'\varphi_1 \Rightarrow \varphi_1$
$P\{{}^2\varphi_1 = {}^2\psi_1\}k_3$
$P\{{}^2\varphi_1 \neq !\}k_5\ ARR.$

Le travail de l'algorithme débute par la recherche et la fixation φ_1 de la première lettre du mot A dans le mot X. Si, dans le mot X, il y a la première lettre du mot A, la commande est transmise à l'opérateur k_3 qui commence la vérification (lettre par lettre) de la première entrée du mot A dans le mot X. Si le mot A entre dans X, la commande est transmise au programme suivant (à l'opérateur k_6) fixant par ψ_2 le signe $\to$ qui se trouve après le mot A. Si à une certaine lettre ${}^2\varphi_2 \neq {}^2\psi_2$, nous passerons à l'opérateur k_5 qui commencera la recherche de la première lettre du mot A dans le mot X après la lettre fixée par φ_1. Si elle est trouvée, la commande est transmise à l'opérateur k_3 et on vérifie de nouveau la première entrée du mot A dans le mot X. S'il n'y a, dans le mot X, que la première lettre du mot A, l'algorithme s'arrête.

Le programme de substitution qui remplace la première entrée du mot A dans le mot X par le mot B est alors construit en fonction des résultats du travail du programme 1.

ψ_2 fixe le signe $\to$ après lequel se trouve le mot B,
φ_1 fixe le début du mot A dans le mot X,
φ_2 fixe la lettre qui suit le mot A dans le mot X.

Ainsi, le programme qui effectue la substitution $\{ A \to B \}$ doit remplacer dans le mot X les lettres de ${}^2\varphi_1$ à ${}^2\varphi_2$ par les lettres du mot B qui se trouvent après ${}^2\psi_2$ et qui se terminent par le signe qui est à la fin du mot B, par exemple, le signe $*$. Puisqu'en général les mots A et B sont d'une longueur arbitraire, nous rangerons le résultat de la substitution en une nouvelle suite d'adresses : le fixateur $\omega_{\text{déb.}}$ en examinera le début. Conformément à ces remarques, le contenu succinct du travail du programme qui effectue la substitution $\{ A \to B \}$ est le suivant : le début du mot X est transcrit de la lettre ${}^2\varphi_{\text{déb.}}$ à la lettre ${}^2\varphi_1$ dans la succession des adresses selon le fixateur $\omega_{\text{déb.}}$; ensuite on transcrit le mot B dont les lettres sont placées entre ${}^2\psi_2$ et le signe $*$; la fin du mot X est transcrite de ${}^2\varphi_2$ au signe ! ; les fixateurs $\varphi_{\text{déb.}}$ et $\omega_{\text{déb.}}$ échangent leurs rôles.

Il est facile maintenant de construire le programme à adresses correspondant.

Programme 2.2. *Exécution de la substitution* $\{ A \to B \}$.

k_6. $'\omega_{\text{déb.}} \Rightarrow \omega$
$'\varphi_{\text{déb.}} \Rightarrow \varphi$

Transcription du début du mot X de ${}^2\varphi_{\text{déb.}}$ à ${}^2\varphi_1$ (l'opérateur k_7. ${}^2\varphi \Rightarrow {}'\omega$).

k_7. ${}^2\varphi \Rightarrow {}'\omega$
$C\,{}'\varphi \Rightarrow \varphi$
$C\,{}'\omega \Rightarrow \omega$
$P\{ {}'\varphi \neq {}'\varphi_1 \}\, k_7$
$C\,{}'\varphi_2 \Rightarrow \psi$

k_8. $P\{ {}^2\psi = * \}\, k_9$
${}^2\psi \Rightarrow {}'\omega$
$C\,{}'\omega \Rightarrow \omega$
$C\,{}'\psi \Rightarrow \psi$
Pk_8

Transcription du mot B (après ${}^2\psi_2$) d'après le fixateur ω.

k_9. ${}'\varphi_2 \Rightarrow \varphi$

k_{10}. ${}^2\varphi \Rightarrow {}'\omega$
$C\,{}'\varphi \Rightarrow \varphi$
$C\,{}'\omega \Rightarrow \omega$
$P\{ {}'\varphi \neq ! \}\, k_{10}$

Le programme de transcription de la fin du mot X de ${}^2\varphi_2$ au signe ! commence à l'opérateur k_{10}.

k_{11}. ${}^2\varphi \Rightarrow {}'\omega$
${}'\omega_{\text{déb.}} \Rightarrow \varphi$
${}'\varphi_{\text{déb.}} \Rightarrow \omega_{\text{déb.}}$
${}'\varphi \Rightarrow \omega_{\text{déb.}}$
ARR.

Programme de changement de rôle des opérateurs $\varphi_{\text{déb.}}$ et $\omega_{\text{déb.}}$. De plus, à la fin du mot on inscrit le signe ! (opérateur ${}^2\varphi \Rightarrow {}'\omega$).

Maintenant, il n'est plus difficile d'établir un programme pour réaliser un algorithme normal donné conformément à l'organigramme de l'algorithme donné, en ayant des programmes de tous les tests d'entrée de la partie gauche des formules de substitution dans le mot examiné et des programmes qui exécutent la substitution $\{ A_k \to B_k \}$. Il faut construire en même temps un programme universel qui permette de réaliser sur C. A. N. n'importe quel algorithme normal de A. A. Markov. En effet, le programme qui teste l'entrée de la partie gauche de la substitution dans le mot examiné ainsi que celui qui réalise la substitution $\{ A \to B \}$ ne sont pas communs à toutes les formules. Il faut seulement en organiser la commande de façon concordante.

Le mot X, qui est examiné pour être traité et dont le début est fixé par le fixateur $\varphi_{\text{déb.}}$, et la suite des formules de substitution

$$A_1 \to B_1 * A_2 \to B_2 * \cdots * A_n \to B_n \,!\,,$$

dans laquelle le signe $*$ sépare les formules de substitution voisines, servent d'information initiale à un tel programme universel.

Pour construire le programme universel qui réalise les algorithmes normaux sur C. A. N., il faut établir le programme du passage d'une formule de substitution à la suivante. Ce programme est simple.

Programme 2.3. *Passage à la formule de substitution suivante.*

k_{13}. $C\,'\psi_1 \Rightarrow \psi_1$
$P\,\{\,{}^2\psi_1 = !\,\}\,ARR.$
$P\,\{\,{}^2\psi_1 \neq *\,\}\,k_{13}$
$C\,'\psi_1 \Rightarrow \psi_1$
ARR

Rappelons que le fixateur ψ_1 du programme 2.1 fixe la première lettre du premier mot de la formule de substitution suivante ; il faut rechercher le signe $*$ après lequel on trouve la lettre cherchée.

En conclusion, notons que l'arrêt de l'algorithme peut être provoqué par l'un des deux signes : soit que le fixateur ψ_1 fixe le signe final ! de la suite des formules de substitution, soit que la formule de substitution accomplie soit finale, c'est-à-dire contienne le signe $\to *$. En rappelant (cf. programme 2.1) que le signe qui sépare le premier et le deuxième mot de la formule de substitution est fixé par ψ_2, nous parvenons à l'arrêt de l'algorithme selon le prédicat

$$P\,\{\,{}^2\psi_2 = \to\cdot\,\}$$

Nous donnerons le programme commun sans explications.

Programme universel 2.4. *qui exécute l'algorithme normal sur C. A. N.*

k_1. $'\psi_{\text{déb.}} \Rightarrow \psi_1$

k_2. $'\varphi_{\text{déb.}} \Rightarrow \varphi_1$

k_3. $P\{{}^2\varphi_1 = {}^2\psi_1\}k_4$
$C\,'\varphi_1 \Rightarrow \varphi_1$
$P\{{}^2\varphi_1 \neq\ !\}k_3\,k_7$
$'\varphi_1 \Rightarrow \varphi_2$
$'\psi_1 \Rightarrow \psi_2$

k_5. $C\,'\varphi_2 \Rightarrow \varphi_2$
$C\,'\psi_2 \Rightarrow \psi_2$
$P\{{}^2\psi_2 = \rightarrow\}k_8$
$P\{{}^2\varphi_2 = {}^2\psi_2\}k_5$

k_6. $C\,'\varphi_1 \Rightarrow \varphi_1$
$P\{{}^2\varphi_1 = {}^2\psi_1\}k_4$
$P\{{}^2\varphi_1 \neq\ !\}k_6\,k_7$

k_7. $C\,'\psi_1 \Rightarrow \psi_1$
$P\{{}^2\psi_1 = !\}ARR.$
$P\{{}^2\psi_1 \neq *\}k_7$
$C\,'\psi_1 \Rightarrow \psi_1$
Pk_2

k_8. $'\omega_{\text{déb.}} \Rightarrow \omega$
$'\varphi_{\text{déb.}} \Rightarrow \varphi$

k_9. ${}^2\varphi \Rightarrow {}'\omega$
$C\,'\varphi \Rightarrow \varphi$
$C\,'\omega \Rightarrow \omega$
$P\{'\varphi \neq {}'\varphi_1\}k_9$
$C\,'\psi_2 \Rightarrow \psi$

k_{10}. $P\{{}^2\psi = *\}k_{11}$
${}^2\psi \Rightarrow {}'\omega$
$C\,'\omega \Rightarrow \omega$
$C\,'\psi \Rightarrow \omega$
Pk_{10}

k_{11}. $'\varphi_2 \Rightarrow \varphi$

k_{12}. ${}^2\varphi \Rightarrow {}'\omega$
$C\,'\varphi \Rightarrow \varphi$
$C\,'\omega \Rightarrow \omega$
$P\{{}^2\varphi \neq\ !\}k_{12}$
${}^2\varphi \Rightarrow {}'\omega$
$'\omega_{\text{déb.}} \Rightarrow \varphi$
$'\varphi_{\text{déb.}} \Rightarrow \omega_{\text{déb.}}$
$'\varphi \Rightarrow \varphi_{\text{déb.}}$
$P\{{}^2\psi_2 \neq \rightarrow\}k_1$ $ARR.$

3. Différentiation des expressions dans les fonctions élémentaires.

Examinons les expressions des fonctions élémentaires représentées en écriture sans parenthèses. Supposons que les expressions contiennent : des opérations à deux places — addition, soustraction, multiplication — et des opérations à une place — exp, ln, sin, cos, etc. L'élévation à une puissance entière (positive ou négative) que nous désignerons sous la forme χ_n (l'indice n indiquant la puissance), est une opération à une place. L'expression des fonctions élémentaires dans l'écriture sans parenthèses contient des formules de la forme αAB, où α est le signe d'une opération à deux places, et des formules de la forme βA, où β est une opération à une place.

Par exemple, l'expression

$$(a + \sin y)^n \times \ln (b + y^2) \tag{1}$$

a. en écriture sans parenthèses. la forme

$$\times\ \chi_n + a \sin y \ln + b\chi_2\, y\,. \tag{2}$$

Nous désignerons l'opération de différentiation par la lettre D. L'écriture des expressions qui contiennent l'opération de différentiation est régie par la règle suivante : le signe de l'opération de différentiation se rapporte à la formule qui est déterminée par l'opération qui se trouve immédiatement après le signe de différentiation. Ainsi, l'écriture $D\beta A$ signifie qu'il faut différentier la formule βA. L'écriture $D\alpha AB$ signifie qu'il faut le faire pour la formule αAB. La différentiation de l'expression (2) s'écrit ainsi :

$$D \times \chi_n + a \sin y \ln + b\chi_2\, y\,. \tag{3}$$

Pour établir un algorithme de différentiation, il est nécessaire de formuler les règles de différentiation, connues grâce à l'enseignement de l'analyse, sous une forme qui assure leur réalisation formelle.

Ecrivons les règles de différentiation des formules avec des opérations à une et à deux places en écriture sans parenthèses.

1) $D \pm uv = \pm DuDv$

2) $D \times uv = + \times Duv \times uDv$

3) $D\chi_n u = \times \times n\chi_{n-1}\, uDu$

4) $D \exp u = \times \exp uDu$

5) $D \ln u = \times \chi_{-1}\, uDu$

6) $D \cos u = \times - 0 \sin uDu$

7) $D \sin u = \times \cos uDu$

etc.

8) $Dy = 1$

9) $Da = 0\,.$

Remarquons que les formules de différentiation citées ne sont pas des formules de substitution, puisque u et v peuvent être à leur tour des formules et par là même compliquées à volonté. Ces formules déterminent les algorithmes de différentiation des opérations à une ou à deux places. Il est facile de comprendre que l'algorithme de différentiation d'une somme et d'une différence est le même, et que celui d'un produit diffère de celui d'une somme et d'une différence. En même temps, tous les algorithmes de différentiation des opérations à une place peuvent être rendus identiques si l'on introduit une

table spéciale des formules de substitution

$$\chi_n \to \times \times n\chi_{n-1} * \exp \to \times \exp * \ln \to \times \chi_{-1} *$$
$$\sin \to \times \cos * \cos \to \times - 0 \sin * ; \qquad (4)$$

lorsque l'on codifie, il est évident que l'on peut omettre le signe $\to$.

Enfin, l'utilisation des formules de différentiation d'une variable libre et des valeurs des constantes est aussi englobée dans un algorithme particulier.

Ainsi, l'algorithme de différentiation des expressions des fonctions élémentaires doit contenir les algorithmes suivants :

1) Algorithme de différentiation d'une somme et d'une différence.
2) Algorithme de différentiation d'un produit.
3) Algorithme de différentiation d'opérations à une place.
4) Algorithme de différentiation d'une variable libre et de constantes.

Les algorithmes de différentiation d'une somme, d'une différence et d'un produit sont normalement appelés règles de différentiation, et l'algorithme de différentiation des opérations à une place, c'est-à-dire celui des opérations élémentaires, est appelé application de la table de différentiation. Il est normal que l'algorithme des opérations élémentaires soit appelé tabulaire, puisqu'il contient la table des formules de substitution (4).

1. *Classification et codification de l'information.*

Conformément à ce qui a été exposé ci-dessus, il faut classer les éléments de l'information selon leurs propriétés. Les expressions des fonctions élémentaires contiennent des opérations, des nombres, une variable indépendante et un signe de différentiation D. On distingue les opérations à une place de celles à deux places. Les opérations à deux places ont deux formes : l'addition (la soustraction) et la multiplication. Les opérations à une place ont autant de formes qu'il y en a dans la table de substitution (4).

2. *Contenu du travail de l'algorithme.*

1) Recherche du signe de différentiation.
2) Détermination de la forme de l'opération qui se trouve après le signe de différentiation D et, selon cette application de l'un ou de l'autre algorithme de différentiation de la formule qui est définie par l'opération qui se trouve après D.

L'expression est chaque fois examinée du début à la fin, et les algorithmes de différentiation sont appliqués autant de fois qu'il y a de signes D contenus dans l'expression. L'algorithme s'arrête lorsqu'il n'y a plus de D dans l'expression.

Dans l'exemple (2) la différentiation se fait de la manière suivante :

Premier examen de l'expression : le signe D se trouve avant la multiplication. Appliquons la règle de différentiation du produit :

$$+ \times \underline{D\chi_n} + a \sin y \ln + b\chi_2 y \times \chi_n + a \sin y \underline{D \ln} + b\chi_2 y .$$

Deuxième examen de l'expression : le signe D se trouve avant l'opération χ_n d'élévation à la puissance n de l'expression $+ a \sin y$. Appliquons la règle de la différentiation de la puissance :

$$+ \times \times \times n\chi_{n-1} + a \sin \underline{yD} + a \sin y \ln + b\chi_2 y \times \chi_n + \\ + a \sin \underline{yD \ln} + b\chi_2 y .$$

Plus loin, le signe D se trouve encore avant l'opération ln. Appliquons l'algorithme de différentiation :

$$+ \times \times \times n\chi_{n-1} + a \sin yD + a \sin y \ln + b\chi_2 y \times \chi_n + \\ + a \sin y \times \chi_{-1} + b\chi_2 yD + b\chi_2 y .$$

Troisième examen de l'expression : le signe D se trouve avant l'addition des expressions a et $\sin y$ et avant l'addition des expressions b et $\chi_2 y$. Appliquons la règle de différentiation de la somme :

$$+ \times \times \times n\chi_{n-1} + a \sin y + DaD \sin y \ln + b\chi_2 y \times \chi_n + a \sin y \\ \times \chi_{-1} + b\chi_2 y + DbD\chi_2 y .$$

Quatrième examen de l'expression : le signe D se trouve avant le nombre a, avant le nombre b, avant l'opération sin et avant celle d'élévation à la puissance de la variable libre. L'algorithme de différentiation nous donne l'expression :

$$+ \times \times \times n\chi_{n-1} + a \sin y + 0 \times \cos yDy \ln + b\chi_2 y \times \chi_n + a \sin y \times \chi_1 \\ + b\chi_2 y + 0 \times \times 2\chi_1 yDy .$$

Cinquième examen de l'expression : le signe D se trouve avant la variable y. En définitive nous avons :

$$+ \times \times \times n\chi_{n-1} + a \sin y + 0 \times \cos y \ln + b\chi_2 y \times \chi_n + \\ + a \sin y \times \chi_{-1} + b\chi_2 y + 0 \times \times 2\chi_1 y 1 .$$

L'expression donnée peut être écrite plus simplement si l'on tient compte des opérations triviales, telles que l'addition de 0, la multiplication par 1, l'élévation à la puissance 1 etc. Cependant, un tel problème de simplification est un nouvel algorithme. On laisse au lecteur le soin d'établir l'algorithme

qui simplifie le résultat. En simplifiant l'expression et en passant à l'écriture normale, nous obtenons à la suite de la différentiation de l'expression (2) la solution :

$$+ \times \times \times n\chi_{n-1} + a \sin y \cos y \ln + b\chi_2\, y \times \chi_n + a \sin y \times \chi_{-1}' +$$
$$+ b\chi_2\, y \times 2\, y = n(a + \sin y)^{n-1} \cos y \ln (b + y^2) +$$
$$+ (a + \sin y)^n (b + y^2)^{-1}\, 2\, y\,.$$

3. *Programmation de la différentiation des expressions dans les formules élémentaires.*

L'algorithme de différentiation est composé des étapes suivantes :

$\mathfrak{A}_1$, recherche du signe D dans l'expression.
$\mathfrak{A}_2$, détermination de la forme de l'opération qui se trouve après le signe D.

Les étapes :

$\mathfrak{A}_3$, détermination de la formule,
$\mathfrak{A}_4$, transcription de la formule,
$\mathfrak{A}_5$, différentiation des opérations à une place,
$\mathfrak{A}_6$, différentiation de la somme et de la différence,
$\mathfrak{A}_7$, différentiation du produit,
$\mathfrak{A}_8$, application des formules de substitution,

font partie intégrante de l'algorithme de différentiation.

Passons à la construction des programmes de chaque étape de l'algorithme.

Programme $\mathfrak{A}_1$. *Recherche du signe D dans l'expression.* Supposons le début de l'expression donnée par le fixateur φ_0, la fin par le signe final !. La recherche est alors effectuée d'après le programme suivant :

Programme $\mathfrak{A}_1$.

k_{11}. $\quad '\varphi_0 \Rightarrow \varphi_1$
k_{12}. $\quad P\{{}^2\varphi_1 = D\}\, k_{21}$
$\quad C\,'\varphi_1 \Rightarrow \varphi_1$
$\quad P\{{}^2\varphi_1 \neq !\}\, k_{12}\, \mathfrak{A}$

L'opérateur k_{12} transmet la commande à l'étape suivante, s'il y a dans l'expression le signe D. Sinon, la sortie dans le programme donne le dernier prédicat.

Programme $\mathfrak{A}_2$. *Détermination de la forme de l'opération qui se trouve après le signe D.*

k_{21}. $\quad C\,'\varphi_1 \Rightarrow \varphi_2$
$\quad P\{{}^2\varphi_2 = \alpha\}\, k_{22}$
$\quad P\{{}^2\varphi_2 = \beta\}\, \mathfrak{A}_5\, \mathfrak{A}_8$
k_{22}. $\quad P\{{}^2\varphi_2 = \times\}\, \mathfrak{A}_7\, \mathfrak{A}_6$

Ce programme a quatre sorties selon l'élément qui se trouve après le signe D : opération à une place, nombre ou variable libre, signe de multiplication, signe d'addition ou de soustraction.

Programme $\mathfrak{A}_3$. *Détermination de la formule qui se trouve après une opération donnée.* L'opération fixée après le signe D détermine la formule à laquelle doit être appliquée la règle de différentiation correspondante. Si l'opération est à deux places, les deux formules u et v auxquelles elle s'applique la suivent. Si l'opération est à une place, elle est suivie de la formule u à laquelle elle s'applique. La fin de la formule u est déterminée sur la base de la règle suivante : dans la formule, le nombre d'opérations à deux places (code α) est d'une unité inférieur au nombre de grandeurs (code a, y) qu'elle contient. Pour indiquer la fin de la formule, un compteur est introduit dans l'adresse δ (au début $'\delta = 0$) : il détermine la différence entre le nombre des grandeurs et le nombre d'opérations à deux places qui figurent dans l'expression donnée. L'apparition d'un « un » dans le compteur δ annonce la fin de la formule.

Programme $\mathfrak{A}_3$.

k_{31}. $0 \Rightarrow \delta$
$C\,'\varphi_2 \Rightarrow \varphi_3$

k_{32}. $P\{\,^2\varphi_3 = \alpha\}\,k_{34}$
$P\{\,^2\varphi_3 = a \vee y\}\,k_{35}$

k_{33} $C\,'\varphi_3 \Rightarrow \varphi_3$
Pk_{32}

k_{34}. $'\delta - 1 \Rightarrow \delta$
Pk_{33}

k_{35}. $'\delta + 1 \Rightarrow \delta$
$P\{\,'\delta \neq 1\}\,k_{33}\,\mathfrak{A}$

Si le fixateur φ_3 fixe une opération à deux places, on soustraira « un » dans δ. Si φ_3 fixe une grandeur, on ajoutera « un » dans δ et on regardera ensuite si le contenu de δ est égal à « un ». Si cette condition est satisfaite, le programme arrête le travail (sortie $\mathfrak{A}$).

Nous ne citerons pas le programme $\mathfrak{A}_4$ de transcription de la formule d'une suite d'adresses dans une autre, puisque nous avons eu l'occasion de le rencontrer dans les précédents problèmes de ce paragraphe.

Programme $\mathfrak{A}_5$. *Différentiation des opérations à une place.* Ce programme est composé de quelques éléments.

Sous-programme $\mathfrak{A}_{5.1}$. *Recherche dans la table des substitutions d'une opération analogue à celle fixée par* φ_2.

k_{51}. $'\rho_0 \Rightarrow \rho$

k_{52}. $P\{\,^2\varphi_2 = {}^2\rho\}\,k_{54}$

k_{53}. $C\,'\rho \Rightarrow \rho$
$P\{\,^2\rho \neq *\}\,k_{53}$
$C\,'\rho \Rightarrow \rho$
Pk_{52}

Le début de la table est indiqué par le fixateur ρ_0. Si les éléments fixés par φ_2 dans l'expression $u\rho$ de la table coïncident, la recherche sera arrêtée par le transfert de la commande à l'étape suivante (à l'opérateur k_{54}). Les opérateurs qui débutent à k_{53} fixent la substitution suivante de la table.

Sous-programme $\mathfrak{A}_{5.2}$. *Application de la substitution.*

k_{54}. $C'\rho \Rightarrow \rho$

k_{55}. $^2\rho \Rightarrow '\psi$

$C'\rho \Rightarrow \rho$

$C'\psi \Rightarrow \psi$

$P\{\,^2\rho \neq *\}\, k_{55}\, \mathfrak{A}_3$

Le programme précédent fixe (par le fixateur ρ) l'opération qui définit la substitution. Le fixateur ψ examine l'adresse à partir de laquelle la substitution doit être transférée. A la fin du transfert, la commande est transmise au programme $\mathfrak{A}_3$ qui détermine la fin de la formule u qui se trouve après le signe de l'opération à une place.

Le programme de différentiation d'une opération à une place contient des sous-programmes de transcription de la formule u en une nouvelle suite, l'inscription du signe D après la formule et, de nouveau, la transcription de la formule u. Rappelons la règle de différentiation à une place :

$$D\beta u = \beta' uDu\,.$$

Ici $\beta \to \beta'$ est la substitution correspondante de la table (4). Convenons d'indiquer les fixateurs nécessaires au programme entre accolades placées après la désignation du programme. Le schéma du programme de différentiation des opérations à une place aura alors la forme suivante :

Programme $\mathfrak{A}_5$.

$\mathfrak{A}_{5.1}$ $\{\varphi_2, \rho_0, \rho\}$

$\mathfrak{A}_{5.2}$ $\{\psi, \rho\}$

$\mathfrak{A}_3$ $\{\varphi_2, \varphi_3\}$

$\mathfrak{A}_4$ $\{\varphi_2, \varphi_3, \psi\}$

$D \Rightarrow '\psi$

$C'\psi \Rightarrow \psi$

$\mathfrak{A}_4$ $\{\varphi_2, \varphi_3, \psi\}$

On recherche dans la table des substitutions (fixateur ρ_0) l'opération (fixateur ρ) qui coïncide avec $^2\varphi_2$ dans l'expression. Ensuite on applique la substitution, c'est-à-dire que la substitution est transcrite en une suite (fixateur ψ). Puis on détermine la fin de la formule (fixateur φ_3) qui se trouve après le signe de l'opération (fixateur φ_2). Cette formule est transcrite en la suite (ψ) ; le signe D y est aussi inscrit et la formule de nouveau transcrite.

Programme $\mathfrak{A}_6$. *Différentiation d'une somme et d'une différence.*

k_{61}. $^2\varphi_2 \Rightarrow '\psi$

$C'\psi \Rightarrow \psi$

$D \Rightarrow '\psi$

$C'\psi \Rightarrow \psi$

$\mathfrak{A}_3$ $\{\varphi_2, \varphi_3\}$

Le contenu du travail du programme conformément à la règle de différentiation

$$D \pm uv = \pm DuDv$$

est le suivant : Le signe de l'opération ($^2\varphi_2$) et le signe D sont transcrits en une nouvelle suite

$\mathfrak{A}_4$ $\{ \varphi_2, \varphi_3, \psi \}$
$D \Rightarrow {}'\psi$
$C\,{}'\psi \Rightarrow \psi$

d'adresses (ψ) ; la formule $u(\varphi_2, \varphi_3)$ est déterminée et le signe D est ajouté à la suite (ψ). La formule $v(\varphi_3, \varphi_4)$ est déterminée et se réécrit (ψ).

$\mathfrak{A}_3$ $\{ \varphi_3, \varphi_4 \}$

$\mathfrak{A}_4$ $\{ \varphi_3, \varphi_4, \psi \}$

Programme $\mathfrak{A}_7$. *Différentiation d'un produit.*

k_{71}. $+ \Rightarrow {}'\psi$
$C\,{}'\psi \Rightarrow \psi$
$\times \Rightarrow {}'\psi$
$C\,{}'\psi \Rightarrow \varphi$
$D \Rightarrow {}'\psi$
$C\,{}'\psi \Rightarrow \psi$

Conformément à la règle de différentiation

$$D \times uv = + \times Duv \times uDv$$

le contenu du travail du programme est le suivant : les signes $+$, $\times$ et D se réécrivent (ψ). La première formule $u(\varphi_2, \varphi_3)$ est définie et réécrite (ψ). La seconde formule $v(\varphi_3, \varphi_4)$ est définie et réécrite (ψ). Le signe $\times$, la formule u, le signe D et la formule v sont ajoutés successivement.

$\mathfrak{A}_3$ $\{ \varphi_2, \varphi_3 \}$

$\mathfrak{A}_4$ $\{ \varphi_2, \varphi_3, \psi \}$

$\mathfrak{A}_3$ $\{ \varphi_3, \varphi_4 \}$

$\mathfrak{A}_4$ $\{ \varphi_3, \varphi_4, \psi \}$
$\times \Rightarrow {}'\psi$
$C\,{}'\psi \Rightarrow \psi$

$\mathfrak{A}_4$ $\{ \varphi_2, \varphi_3, \psi \}$
$D \Rightarrow {}'\psi$
$C\,{}'\psi \Rightarrow \psi$

$\mathfrak{A}_4$ $\{ \varphi_3, \varphi_4, \psi \}$

Programme $\mathfrak{A}_8$. *Application des formules de substitutions* $Dy \Rightarrow 1$ *et* $Da \Rightarrow 0$. Le déroulement en est évident.

Outre les programmes construits, l'algorithme de différentiation des expressions dans les fonctions élémentaires comprend : un programme qui réécrit le début de l'expression, de φ_0 au signe D ; un programme de réécriture de l'expression, de φ_3 au signe D suivant, après avoir appliqué l'algorithme de différentiation de l'opération à une place ; un programme de réécriture de l'expression, de φ_4 jusqu'au signe D suivant, après avoir appliqué l'algorithme de différentiation des opérations à deux places. Si, après l'application de l'algorithme de différentiation, il n'y a pas, plus loin dans l'expression, de signe D, la fin de l'expression se réécrit après φ_3 ou après φ_4 en une suite.

Le schéma général du programme de différentiation des expressions des fonctions élémentaires est illustré par la figure 18 (rappelons que le début de l'expression est fixé par φ_0 ; le début de la suite des adresses dans lesquelles est réécrit le résultat de la différentiation est fixé par ω_0 ; le début de la table des substitutions est fixé par ρ_0).

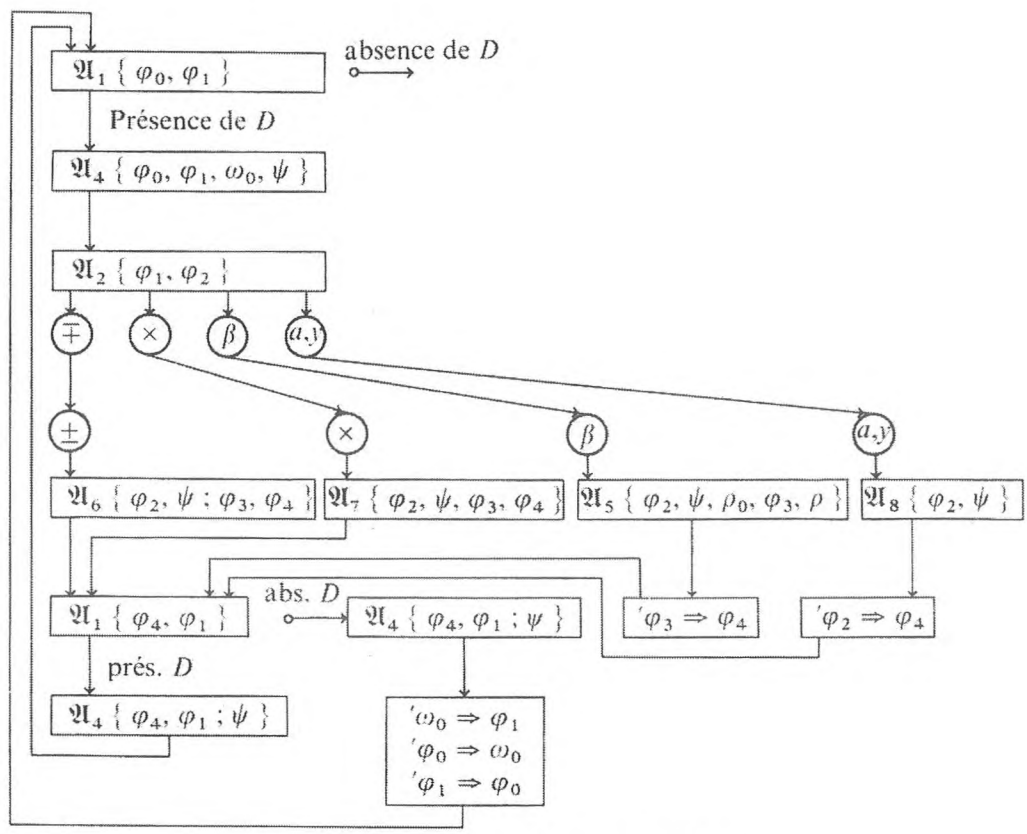


Fig. 18. Schéma général de la différentiation.

4. Réduction des formules du calcul des propositions à la forme normale.

Dans le chapitre I, on a indiqué comment la forme normale de l'expression des propositions complexes était définie au moyen des propositions élémentaires à l'aide des opérations logiques fondamentales : addition logique (signe $\vee$), multiplication logique (signe $\wedge$) et négation.

La représentation d'une proposition complexe sous la forme normale a été utilisée pour construire la formule d'après la table des valeurs de vérité d'une proposition complexe. Il est évident que le problème inverse : la construction de la table des valeurs de vérité d'une proposition complexe d'après une formule donnée, se résout très facilement si l'on écrit la formule sous la forme normale, en particulier, le problème important de la théorie des propositions est facilement résolu sous cette forme : vérification du fait qu'une proposition

complexe est identiquement vraie. Il est facile de comprendre que, pour résoudre ce problème, il suffit de vérifier que chaque terme connecté de la forme normale soit identiquement vrai. Chacun d'eux est à son tour identiquement vrai, si, et seulement si, il est composé ne serait-ce que d'une proposition élémentaire et de sa négation.

Pour construire l'algorithme de réduction des formules de calcul des propositions à la forme normale, nous adopterons l'écriture sans parenthèses pour les formules. Nous savons déjà (cf. différentiation des expressions dans les formules élémentaires) que dans l'écriture sans parenthèses, chaque opération à deux places détermine une formule de forme θAB, où A et B sont des formules. Puisque la négation est une opération à une place et qu'elle s'applique toujours à une formule, il est possible de marquer la négation d'une formule par la négation de l'opération qui définit cette formule. Par exemple, l'écriture $\overline{\wedge}\ AB$ signifie la négation de la formule $\wedge\ AB$, c'est-à-dire que l'écriture $\overline{\wedge}\ AB$ est équivalente à l'écriture avec parenthèses suivante :

$$\overline{[A] \wedge [B]}\,.$$

De façon analogue, l'écriture $\overline{\vee}\ AB$ est équivalente à l'écriture avec parenthèses $\overline{[A] \vee [B]}$. Ainsi, les formules de calcul des propositions dans l'écriture sans parenthèses, avec la convention adoptée sur la façon d'exprimer la négation, sont notées au moyen de propositions élémentaires à l'aide des opérations binaires à deux places $\wedge$ et $\vee$ et à l'aide de ces mêmes opérations avec négation, ce qu'indique la négation de la formule correspondante que définit l'opération à deux places données. Examinons un exemple de proposition complexe :

$$\wedge\ \overline{\wedge}\ \overline{\vee}\ xy \vee x\overline{y} \vee \overline{xy}\,.$$

Dans l'écriture habituelle avec parenthèses, ces formules ont la forme suivante :

$$\overline{\overline{((x \vee y) \wedge (x \vee \overline{y}))}} \wedge (\overline{x} \vee \overline{y})\,.$$

Pour les propositions complexes, la réduction des formules à la forme normale est fondée sur l'utilisation des règles de transformation suivantes :

1. $(\overline{\overline{A}}) \equiv A$.
2. $\overline{\wedge}\ AB \equiv \vee\ \overline{A}\,\overline{B}$.
3. $\overline{\vee}\ AB \equiv \wedge\ \overline{A}\,\overline{B}$.
4. $\vee\ A \wedge BC \equiv \wedge \vee AB \vee AC$.
5. $\vee \wedge ABC \equiv \wedge \vee AC \vee BC$.

Les règles 2 et 3 sont dites règles de De Morgan ; les règles 4 et 5 sont la distributivité de la multiplication logique par rapport à l'addition.

Dans l'exemple de proposition complexe que nous venons d'examiner, la réduction à la forme normale par étapes se présente de la façon suivante :

$$\text{Formule initiale} \quad \wedge \underline{\overline{\wedge}\ \overline{\vee}\ xy \vee x\overline{y}} \vee \overline{\overline{xy}}.$$

Appliquons la règle 2 à la formule soulignée

$$\wedge \vee \vee xy \underline{\overline{\vee}\ x\overline{y}} \vee \overline{\overline{xy}}.$$

Appliquons la règle 3 à la formule soulignée

$$\wedge \vee \vee xy \underline{\wedge\ \overline{xy}} \vee \overline{\overline{xy}}.$$

L'application des règles 2 et 3 réduit la formule à l'écriture dans laquelle la négation ne se rapporte qu'aux propositions élémentaires. Appliquons maintenant la règle 4 à la formule soulignée ($A = \vee xy$, $B = \overline{x}$, $C = y$)

$$\wedge \wedge \vee \vee xy\overline{x} \vee \vee xyy \vee \overline{\overline{xy}},$$

la formule est réduite à la forme normale.

Passons maintenant à la description de l'algorithme qui réduit les propositions complexes à la forme normale. La formule en écriture sans parenthèses qui utilise les deux opérations binaires $\vee$ et $\wedge$, avec ou sans négation sur les propositions élémentaires et sur les opérations, sert d'information initiale. La formule est définie par la propriété suivante : le nombre des propositions de la formule est supérieur d'une unité au nombre des opérations $\wedge$ ou $\vee$.

Contenu du travail de l'algorithme :

I. Application des règles 2 et 3 (règles de De Morgan) jusqu'à ce que les signes des opérations soient tous négatifs, en tenant compte de la règle 1.

II. Application de la règle de distributivité, soit sous la forme 4 si avant $\wedge$ on a la formule, soit sous la forme 5 si avant $\wedge$ on a le signe $\vee$.

Plus en détail, les principales étapes du travail de l'algorithme sont les suivantes :

I.1. Recherche du premier élément qui porte une négation en partant du début de la formule.

I.2. Détermination de la forme de l'élément fixé qui porte la négation. Si c'est une proposition, on applique I.3 ; si c'est une opération on applique I.4.

I.3. Recherche de l'élément suivant qui porte une négation.

I.4. Application des règles 2 et 3 de transformation de la formule et passage à I.3.

Si dans I.1 et I.3, on ne trouve pas dans la formule d'élément qui porte une négation, l'étape I de l'algorithme est terminée et nous passons à II.

II.1. Recherche du signe $\wedge$ après $\vee$. Si après $\vee$ il y a le signe $\wedge$, on exécute II.2; sinon la formule est réduite à la forme normale, l'algorithme a terminé son travail.

II.2. Choix de la règle de distributivité 4 ou 5.

II.3. Application de la règle 4 et passage à II.1.

II.4. Application de la règle 5 et passage à II.1.

Il est facile de construire des programmes à adresses pour I.1, I.2, I.3, II.1, II.2 des principales étapes du travail de l'algorithme que l'on vient de citer. Nous noterons le début de la formule par le fixateur φ_0; la fin en est normalement déterminée par le signe !.

Programme I.1. *Recherche du premier élément qui porte une négation.*

k_{11}. $\quad '\varphi_0 \Rightarrow \varphi_1$

k_{12}. $\quad P\{{}^2\varphi_1 = \bar{e}\}\, k_{13}$

$\quad C\,'\varphi_1 \Rightarrow \varphi_1$

$\quad P\{{}^2\varphi_1 \neq !\}\, k_{12}\, \mathfrak{A}$

Un élément quelconque de la formule est désigné par le signe e. Si, dans la formule, aucun élément ne porte de négation, l'algorithme arrête son travail.

Programme I.2. *Détermination de la forme de l'élément fixé par* φ_1.

k_{13}. $\quad P\{{}^2\varphi_1 = \bar{A}\}\, k_{14}\, k_{15}$

Ici, le signe A désigne une proposition. Le programme a deux sorties : si φ_1 fixe une proposition qui porte la négation, il y a passage à k_{14}; si φ_1 fixe un signe d'opération qui porte la négation, il y a passage à k_{15}.

Programme I.3. *Recherche de l'élément qui porte une négation qui suit* φ_1.

k_{14}. $\quad C\,'\varphi_1 \Rightarrow \varphi_1$

$\quad P\{{}^2\varphi_1 = \bar{e}\}\, \mathfrak{A}_{\bar{e}}$

$\quad P\{{}^2\varphi_1 \neq !\}\, k_{14}\, \mathfrak{A}_e$

Ce programme a deux sorties selon la présence ou l'absence de $\bar{e}$.

Programme I.4. *Application des règles de transformation des deuxième et troisième formules.*

k_{15}. $P\{\,^2\varphi_1 = \overline{\wedge}\,\}(\vee \Rightarrow {}'\varphi_1)\ (\wedge = {}'\varphi_1)$
$C\,{}'\varphi_1 \Rightarrow \varphi_2$

L'opérateur k_{15} remplace le signe $\overline{\wedge}$ par $\vee$ et le signe $\overline{\vee}$ par $\wedge$. L'opérateur k_{16} change $\bar{e}$ en e et vice versa, c'est-à-dire que l'on rend négative la première formule après le signe de l'opération.

k_{16}. $P\{\,^2\varphi_2 = \bar{e}\,\}(e \Rightarrow {}'\varphi_2)(\bar{e} \Rightarrow {}'\varphi_2)$
$0 \Rightarrow \delta$
${}'\varphi_2 \Rightarrow \varphi_3$

k_{17}. $P\{\,^2\varphi_3 = A\,\}({}'\delta + 1 \Rightarrow \delta)\ ({}'\delta - 1 \Rightarrow \delta)$
$C\,{}'\varphi_3 \Rightarrow \varphi_3$
$P\{\,{}'\delta \neq 1\,\}k_{17}$

Introduisons dans l'adresse δ un compteur pour calculer la différence entre le nombre des propositions élémentaires et celui des opérations qui se trouvent dans la formule. Comme nous l'avons déjà noté précédemment, si le début de la formule est connu, sa fin sera déterminée par la condition ${}'\delta = 1$.

k_{18}. $P\{\,^2\varphi_3 = \bar{e}\,\}(e \Rightarrow {}'\varphi_3)\quad(\bar{e} \Rightarrow {}'\varphi_3)$

Le calcul est exécuté par l'opérateur k_{17}. L'opérateur k_{18} opère la négation de la deuxième formule après le signe d'opération.

Programme II.1. *Recherche du premier signe* $\wedge$ *après le signe* $\vee$.

k_{21}. ${}'\varphi_0 \Rightarrow \varphi_1$

k_{22}. $P\{\,^2\varphi_1 = \vee\,\}k_{23}$
$C\,{}'\varphi_1 \Rightarrow \varphi_1$
$P\{\,^2\varphi_1 \neq !\,\}k_{22}\quad ARR$

Ce programme a trois sorties correspondant aux cas suivants : quand il n'y a pas dans la formule de signe $\vee$, quand il n'y a pas de signe $\wedge$ après $\vee$, quand il y a le signe $\wedge$ après le signe $\vee$.

k_{23}. ${}'\varphi_1 \Rightarrow \varphi_2$

k_{24}. $P\{\,^2\varphi_2 = \wedge\,\}k_{25}$
$C\,{}'\varphi_2 \Rightarrow \varphi_2$
$P\{\,^2\varphi_2 \neq !\,\}k_{24}\quad ARR$

Dans ce dernier cas, la commande est transmise au programme II.2 (à l'opérateur k_{25}).

Programme II.2. *Choix des règles* 4 *ou* 5.

k_{25}. $\overline{C}\,{}'\varphi_2 \Rightarrow \varphi_3$
$P\{\,^2\varphi_3 = A\,\}k_{26}\,k_{27}$

Si avant le signe $\wedge$, il y a une proposition, on appliquera la règle 4 (opérateur k_{26}) ; si avant $\wedge$ il y a $\vee$, on appliquera la règle 5 (opérateur k_{27}).

Lors de la construction des programmes II.3 et II.4 il faut réécrire la formule, puisque l'application des règles de distributivité l'élargit (augmentation du nombre de ses éléments). Pour le programme de transfert de la suite des éléments qui va du début marqué par le fixateur φ_0 jusqu'à l'élément noté par le fixateur φ_1, introduisons la désignation abrégée $\Pi \{ \varphi_0, \varphi_1, \psi \}$, dans la suite des adresses dont le début est noté par ψ. Nous laisserons au lecteur le soin de faire le programme correspondant, à la fin duquel le fixateur ψ fixe l'élément $C\,'\varphi_1$.

Le transfert est effectué, incluant l'élément fixé par φ_0 et excluant celui fixé par φ_1. Remarquons que φ_2, une fois le programme II.2 accompli, fixe le signe $\wedge$ avant lequel il y a le signe $\vee$. Pour construire le programme II.4, il faut déterminer le signe $\vee$ dans la formule à laquelle on applique la règle 4. Il est facile de comprendre que le signe $\vee$ de la formule $\vee\ A \wedge BC$ est déterminé par la condition suivante : la différence entre le nombre des propositions élémentaires et celui des opérations qui se trouvent avant le signe $\wedge$ doit être égale à zéro. En effet, A est la formule dans laquelle cette différence est égale à 1, la présence du signe $\vee$ avant A rend cette différence égale à zéro.

Programme II.4. *Application de la règle* 4.

$k_{26}.$ $\quad 0 \Rightarrow \delta$

$\quad\quad {}'\varphi_3 \Rightarrow \varphi_1$

$k_{27}.$ $\quad P \{ {}^2\varphi_1 = A \} ('\delta + 1 \Rightarrow \delta)$

$\quad\quad\quad\quad\quad\quad\quad\quad ('\delta - 1 \Rightarrow \delta)$

$\quad\quad P \{ '\delta = 0 \}\, k_{28}$

$\quad\quad \overline{C}\,'\varphi_1 \Rightarrow \varphi_1$

$\quad\quad Pk_{27}$

$k_{28}.$ $\quad {}'\psi_0 \Rightarrow \psi$

$\quad\quad \Pi \{ \varphi_0, \varphi_1, \psi \}$

$\quad\quad C\,'\varphi_1 \Rightarrow \varphi_3$

$\quad\quad C\,'\varphi_2 \Rightarrow \varphi_4$

$\quad\quad 0 \Rightarrow \delta$

$\quad\quad {}'\varphi_4 \Rightarrow \varphi_5$

$k_{29}.$ $\quad P \{ {}^2\varphi_5 = A \} ('\delta + 1 \Rightarrow \delta)$

$\quad\quad\quad\quad\quad\quad\quad\quad ('\delta - 1 \Rightarrow \delta)$

$\quad\quad C\,'\varphi_5 \Rightarrow \varphi_5$

$\quad\quad P \{ '\delta \neq 1 \}\, k_{29}$

$\quad\quad C\,'\varphi_5 \Rightarrow \varphi_5$

D'abord, recherche du signe $\vee$ qui détermine la formule

$$\vee\ A \wedge BC$$

avant le dernier élément de la formule A fixé par φ_3 (jusqu'à l'opérateur k_{28}).

Ensuite, réécriture du début de la formule (φ_0, φ_1) en une nouvelle suite d'adresses d'après le fixateur ψ ; φ_3, φ_4, φ_5 fixent le début des formules A, B, C. Le résultat de l'application de la règle 4 est inséré dans la nouvelle suite d'adresses (ψ) :

$$\vee\ A \wedge BC =$$

$$= \wedge \vee AB \vee AC\,.$$

$$\wedge \Rightarrow {}'\psi$$
$$C\,{}'\psi \Rightarrow \psi$$
$$\vee \Rightarrow {}'\psi$$
$$C\,{}'\psi \Rightarrow \psi$$
$$\Pi\{\varphi_3, \varphi_2, \psi\}$$
$$\Pi\{\varphi_4, \varphi_5, \psi\}$$
$$\vee \Rightarrow {}'\psi$$
$$C\,{}'\psi \Rightarrow \psi$$
$$\Pi\{\varphi_3, \varphi_2, \psi\}$$
$${}'\varphi_5 \Rightarrow \varphi$$
$$k_{2.10}\cdot \quad {}^2\varphi \Rightarrow {}'\psi$$
$$C\,{}'\varphi \Rightarrow \varphi$$
$$C\,{}'\psi \Rightarrow \psi$$
$$P\{{}^2\varphi \neq !\}\, k_{2.10}\, k_{2.11}$$
$${}'\psi_0 \Rightarrow \varphi$$
$${}'\varphi_0 \Rightarrow \psi_0$$
$${}'\varphi \Rightarrow \varphi_0$$
$$Pk_{21}\cdot$$

La construction du programme d'application de la règle 5 est laissée au soin du lecteur.

CHAPITRE VI

AUTOMATISATION DE LA PROGRAMMATION

Les difficultés qui se présentent au moment de la préparation des problèmes à résoudre sur C. A. N. obligent à développer des méthodes de programmation qui admettent l'automatisation, c'est-à-dire la possibilité d'effectuer la plus grande partie du travail de programmation en utilisant cette même machine.

Ce chapitre donne un bref aperçu des méthodes qui automatisent la programmation. Nous examinerons plus en détail les questions d'automatisation liées à la méthode opératorielle de programmation.

1. APERÇU GÉNÉRAL DES MÉTHODES

La préparation des problèmes à résoudre sur une calculatrice universelle à commande par programme se divise naturellement en plusieurs étapes :

1) Choix d'une méthode numérique, d'une évaluation de l'erreur et choix d'un moyen pour contrôler la justesse et la précision du résultat.

2) Etablissement d'un schéma de calcul, c'est-à-dire division en étapes du processus de calcul et détermination de leur ordre d'exécution.

3) Programmation, c'est-à-dire représentation des calculs sous forme d'une succession d'opérations élémentaires exécutées par différentes instructions de la machine.

Les deux premières étapes exigent un spécialiste hautement qualifié pour mener à bien la préparation du problème à résoudre sur C. A. N. La dernière étape, la programmation, est un travail plus ou moins technique qui demande beaucoup de travail et occupe un gros volume : il peut être effectué par quelqu'un de beaucoup moins qualifié. Pour écrire un programme il faut beaucoup de travail, un soin et une attention extrêmes. Lors de la répartition du schéma de calcul dans les instructions, les fautes techniques sont généralement inévitables : fautes d'écriture, emploi incorrect de l'organe mémoire, erreurs dans la programmation des transferts de commande, etc. Les sous-programmes

préparés à l'avance doivent concorder avec le programme général du problème pour pouvoir être utilisés ; des fautes techniques sont aussi à prévoir. Dans le cas où l'on découvre des fautes dans un programme, ou que l'on ait besoin de le compléter, il faut apporter les modifications et les compléments nécessaires en déplaçant les différentes parties du programme dans l'organe mémoire, et en répartissant à nouveau les cellules. Pour rectifier les fautes d'un programme, il faut aussi y introduire les changements adéquats. Malgré cela, on laisse encore souvent passer des fautes. Pour arriver à résoudre sans fautes un problème sur C. A. N., il faut faire d'avance la mise au point du programme par des calculs d'essai. Toute la préparation du travail (établissement du programme d'après un organigramme, recherche et rectification des fautes, changement et introduction de compléments, calculs d'essai sur machine) fait perdre beaucoup de temps-travail aux programmeurs, aux mathématiciens, et aussi beaucoup de temps-machine. Souvent il en résulte que la préparation du problème à résoudre sur C. A. N. demande beaucoup plus de temps que la résolution elle-même, sur la machine. Pour assurer sur C. A. N. un travail à plein rendement, il faut avoir un grand nombre de collaborateurs scientifiques qualifiés, de programmeurs et de mathématiciens (cent à deux cents personnes pour assurer le travail d'une C. A. N. de puissance moyenne).

Au fur et à mesure qu'on acquiert de l'expérience en programmation, on réussit à mettre au point des moyens et des méthodes qui allègent et ordonnent le travail de préparation des programmes. La systématisation de ces règles de programmation rend possible l'automatisation de l'établissement des programmes.

A l'heure actuelle, les méthodes suivantes qui accomplissent le travail d'après des programmes établis et permettent leur automatisation sont très largement répandues.

1) Méthode des sous-programmes-bibliothèques.
2) Méthode des adresses symboliques.
3) Méthode opératorielle élaborée sous la direction du professeur A. A. Liapounov.
4) Méthode des programmes d'interprétation.
5) Méthode de programmation à adresses.

Dans ce chapitre, nous allons donner une courte description des deux premières méthodes et nous ne nous arrêterons en détail que sur les questions d'automatisation de la programmation fondées sur la méthode opératorielle d'établissement des programmes. Nous exposerons brièvement aussi la méthode des programmes d'interprétation et celle des programmes à adresses.

1. Méthode des sous-programmes-bibliothèque.

L'emploi de programmes standards, élaborés pour des schémas de calcul souvent rencontrés dans des problèmes différents, abrège énormément le travail de la programmation. Lorsque la bibliothèque des programmes standards est

assez riche, les blocs standards permettent d'établir presque complètement le programme de nombreux problèmes complexes. Dans ce cas, la programmation d'un problème consiste à diviser les calculs en étapes standards dont on a déjà les programmes, et à coordonner ces programmes standards entre eux. La programmation du passage d'une partie standard à l'autre est un travail proprement technique et facile à formaliser. On peut formuler des règles assez simples et entièrement définies, dont l'application permet d'automatiser le travail par l'introduction de programmes standards dans le programme général du problème.

Au cours de cette introduction de programmes standards, on peut rencontrer les cas suivants :

1) Le programme standard est placé dans la partie interne de l'organe mémoire à une place déterminée d'avance. De tels programmes standards sont dits *fermés.*

2) Le programme standard (placé dans la partie externe de l'organe mémoire) est mis pour le travail dans la partie interne de l'organe mémoire à une place déterminée par le programmeur. De tels programmes standards sont dits *ouverts.*

Dans le premier cas (lorsqu'on utilise des programmes standards fermés), il est nécessaire de calculer dans le programme principal quelles cellules de l'organe mémoire sont les cellules d'entrée, de sortie et de travail du programme standard.

Dans le programme principal, il faut prévoir : 1° des instructions de transfert des données, indispensables au travail du programme standard, vers les cellules d'entrée, et 2° des instructions de transfert des résultats des calculs, fournies par le programme standard, des cellules de sortie vers des cellules réservées à cet effet dans le programme principal. Le passage au programme standard est codifié dans le programme principal par une instruction particulière (ou plusieurs instructions) à l'aide de codes spéciaux. Un programme d'interprétation spécial déchiffre ce code en le remplaçant par le transfert de la commande à la première instruction du programme standard, et, dans le cas où cela serait impossible, prépare le retour après le travail du programme standard à l'emplacement correct du programme principal.

Dans le second cas (lorsqu'on emploie les programmes standards ouverts), outre ce qui a été montré ci-dessus, il est indispensable de réduire le programme standard en fonction de l'emplacement où il sera exécuté. Si le programme standard contient des instructions, elles doivent être changées. Un nombre, égal à la différence entre le numéro effectif de la première instruction du sous-programme et un numéro conventionnel de l'instruction sur laquelle a été calculé le programme standard, sert d'information initiale à la modification des instructions de transfert et de la commande du programme standard. Ce nombre est indiqué par le programmeur dans l'instruction de transfert au programme standard.

Outre le travail que nous avons examiné, le programme d'interprétation peut automatiquement changer les autres paramètres qui caractérisent les sous-programmes (le nombre de cycles, les coefficients, la précision des calculs, etc.). Les données pour ces changements sont indiquées dans le programme principal.

Le moyen d'utiliser les programmes standards que l'on a examiné augmente naturellement le volume du programme et son temps d'exécution. C'est pourquoi, dans de nombreux cas, il est plus rationnel que le programme standard fasse partie intégrante du programme principal. De plus, non seulement l'emplacement du sous-programme, mais aussi l'adresse des cellules d'entrée, de sortie et de travail varient.

2. Méthode des adresses symboliques.

Ce qui est essentiel dans cette méthode, c'est qu'on établit entièrement à la main le programme du problème ; cependant, à la place des adresses effectives dans les instructions et des numéros des instructions elles-mêmes, on utilise des nombres conventionnels : les codes symboliques. De plus, il n'est nullement nécessaire que les nombres conventionnels représentent un ordre. Il importe seulement que des codes symboliques différents soient enregistrés dans des adresses et des instructions différentes. Il est nécessaire aussi que les codes effectifs soient différents des codes conventionnels. Les codes symboliques sont choisis de façon à pouvoir facilement et simplement introduire des compléments et faire des rectifications dans le programme. Ce n'est qu'une fois que le programme a été soigneusement vérifié et qu'il est prêt à être introduit en machine que l'on change les codes symboliques en codes effectifs. La transformation des codes conventionnels en numéros d'instructions effectifs et en adresses effectives des cellules de l'organe mémoire peut être faite automatiquement par un programme de transformation spécial. Le programme du problème en codes symboliques et la table de répartition des cellules de l'organe mémoire servent d'information à un tel programme. Lors de la répartition des numéros effectifs des instructions, le programme de transformation répartit les instructions du programme donné dans l'ordre de leur entrée en machine. Les instructions qui ont un numéro effectif depuis le début font exception : elles se placent dans l'organe mémoire au numéro indiqué. Les instructions suivantes, qui ont des codes symboliques, sont placées les unes après les autres dans l'ordre de leur entrée en machine. D'après la table de répartition des cellules de l'organe mémoire, le changement des codes symboliques des instructions en numéros effectifs a lieu dans les instructions de transfert de commande. Le changement des adresses symboliques des nombres se fait d'après la table de répartition de l'organe mémoire de la façon suivante : le programme de transformation examine le programme donné et choisit l'adresse symbolique du nombre dans l'instruction, la recherche dans la table de répartition des adresses effectives et la remplace par l'adresse effective correspondante indiquée dans la table. Si le code symbolique du

nombre existe dans la table, il est remplacé par le numéro effectif libre, dont c'est le tour, dans la cellule de l'organe mémoire et, le code symbolique et l'adresse effective, qui lui correspondent, sont insérés dans la table de répartition des adresses. La machine sort en fin de compte le programme transformé et la table de répartition des cellules de l'organe mémoire d'après laquelle on détermine la correspondance entre les codes symboliques et effectifs.

La méthode des adresses symboliques peut être appliquée en combinaison avec celle des programmes-bibliothèque standards. De plus, les sous-programmes peuvent être établis en codes symboliques, ce qui élargit les possibilités de leur emploi.

Le programme de transformation qui change les codes symboliques en codes effectifs est assez compliqué. Cependant ce programme est, en fait, la partie fondamentale de la méthode de programmation exposée ci-dessous.

3. Méthode opératorielle d'automation de la programmation.

Le schéma opératoriel du programme qui est constitué par n'importe quel ensemble d'opérateurs sert d'information initiale pour établir le plan de travail du problème à l'aide d'un programme générateur de programmes. Lorsqu'on établit le schéma du programme, on a l'habitude d'utiliser les formes suivantes d'opérateurs : arithmétiques, logiques, de substitution d'adresse, d'envoi, de formation (de rétablissement), de circulation. Ces opérateurs sont appelés opérateurs *standards*. S'il s'avère que, pour un problème donné, il est impossible d'établir un schéma de programme satisfaisant en utilisant les opérateurs standards, on introduira des opérateurs dits *non standards* qui ont une affectation fonctionnelle quelconque. Le programme qui les concerne est construit à la main. En particulier, un sous-programme standard mis au point auparavant peut servir d'opérateur non standard dans le schéma de programme.

L'information concernant les opérateurs qui rentrent dans le schéma de programme est généralement codifiée (avec des nombres conventionnels) de manière à assurer sa conversion automatique par le programme générateur de programmes en programme du problème.

Le programme de génération peut être exécuté en plusieurs étapes qui se succèdent automatiquement.

1) Les opérateurs qui rentrent dans le schéma de programme sont programmés en codes symboliques. Les opérateurs d'un seul type sont programmés les uns après les autres dans un ordre déterminé, par exemple, les opérateurs arithmétiques, les opérateurs logiques, les opérateurs d'envoi, de circulation, de substitution d'adresse et de rétablissement, ou dans l'ordre de leur succession dans le schéma.

2) On économise des cellules de travail. Lorsqu'on programme des formules arithmétiques, les résultats des calculs intermédiaires sont placés dans des cellules de travail de l'organe mémoire dont le numéro n'est pas connu d'avance. Toutefois, en pratique, cela peut mener à une utilisation irration-

nelle des cellules opératives de l'organe mémoire ; c'est pourquoi, il est nécessaire de programmer les calculs selon des formules qui les utilisent le moins possible pour conserver les résultats intermédiaires. Le plus commode est de faire l'économie des cellules de travail après la programmation des opérateurs arithmétiques et des autres opérateurs dans lesquels se trouvent les résultats intermédiaires du calcul.

3) Les programmes des divers opérateurs sont placés conformément à leur ordre dans le schéma de programme.

4) Les codes symboliques des nombres et des instructions sont changés en adresses effectives conformément à la répartition des adresses de l'organe mémoire, etc.

Chaque étape du programme est effectuée par un bloc spécial. Les blocs du programme travaillent indépendamment les uns des autres ; cependant le résultat du travail d'un bloc peut servir d'information initiale à un autre bloc. On établit en conséquence l'ordre dans lequel travaillent les blocs du programme :

A bloc arithmétique,
P bloc logique,
Γ bloc de transfert,
Z bloc d'envoi,
Φ bloc de rétablissement,
H bloc de travail des opérateurs non standards,
E bloc d'économie des cellules de travail,
N bloc de disposition des opérateurs d'après leur numéro d'ordre dans le schéma du programme,
D bloc d'attribution des adresses effectives aux nombres et aux instructions du programme établi.

Le volume énorme, aussi bien du programme que de l'information traitée, force à placer le programme des blocs du programme dans la mémoire externe de la machine et à l'appeler au fur et à mesure en mémoire interne au moment où l'on traite l'information. Cette dernière peut aussi être introduite partie par partie par l'organe d'entrée. Si l'on ne connaît pas d'avance le volume du programme, qu'on ne puisse donc pas attribuer les adresses effectives, c'est-à-dire remplacer les adresses symboliques par les adresses véritables, alors il est possible de prévoir une sortie du programme en adresses symboliques afin d'établir le tableau de répartition des codes effectifs.

Pour qu'il soit plus facile de s'orienter dans le programme établi, le bloc de répartition des opérateurs constitue, conformément au schéma du programme, la table des caractéristiques dans laquelle est indiqué l'endroit où se trouve chaque opérateur du schéma. Ainsi le travail du programme de génération donne le programme du problème conforme au schéma de programme en codes symboliques et effectifs, la table des correspondances entre les codes symboliques et effectifs et celle des caractéristiques.

La préparation du schéma de programme qui produira le programme de génération est différente de celle que l'on aurait pour un programme écrit à la main. Dans le schéma de travail établi par un programmeur, le schéma joue un rôle auxiliaire : il facilite l'élaboration du schéma de calcul et permet de diviser le travail en parties suivant la composition du programme du problème. Un programmeur qui connaît le schéma de calcul et les particularités de programmation que présente la machine sur laquelle il travaille peut combler les lacunes du schéma fait à la main. Il peut aussi relever les fautes commises dans le schéma de programme au moment de la répartition dans les instructions et les corriger au cours du travail.

Dans la programmation automatique qui utilise un programme de génération, le schéma de programme doit inclure une information exhaustive sur le problème, selon laquelle le programme puisse être établi à l'aide d'algorithmes entièrement définis. Ainsi, le schéma de programme y joue un rôle beaucoup plus important et le spécialiste doit accorder une plus grande attention à son exactitude. Une faute qu'on laisse passer dans un schéma de programme que l'on prépare pour traiter un programme de génération ne sera pas corrigée et passera dans le programme du problème. Pour diminuer le nombre des fautes dans la préparation du schéma et pour renforcer l'efficacité du travail du programme de génération, on élabore des règles de codification de l'information qui la rendent simple et commode à utiliser pour les programmeurs. La quantité d'information introduite en machine doit être aussi petite que possible. L'information complète du problème est établie partie par partie : séparément pour chaque opérateur qui rentre dans le schéma de programme ; en outre on indique l'information qui se rapporte au programme dans son entier.

Décrivons l'information qu'il convient de donner pour les différents types d'opérateurs.

1) Pour l'opérateur arithmétique, on indique les formules suivant lesquelles on doit faire les calculs. Elles sont dans l'ordre d'exécution des calculs.

2) Pour l'opérateur logique, on indique la formule logique

$$f(P_1, P_2, ..., P_n)$$

et les numéros des opérateurs auxquels la commande transmet l'opérateur logique donné. Ici, $P_1, P_2, ..., P_n$ sont les conditions logiques élémentaires.

3) Pour l'opérateur de substitution d'adresse on indique : le paramètre suivant lequel on effectue la substitution d'adresse et le pas de substitution d'adresse, les grandeurs qui dépendent du paramètre donné, et les numéros des opérateurs qui contiennent les grandeurs.

4) L'information concernant les opérateurs d'envoi contient les grandeurs qui doivent être envoyées dans les cellules standards ou les grandeurs qui doivent être transférées des cellules standards dans une suite, et les numéros des

opérateurs qui doivent être utilisés par les grandeurs indiquées se trouvant dans les cellules standards.

5) L'information sur l'opérateur de rétablissement contient les numéros des opérateurs qui doivent être ramenés à l'état initial.

6) L'information sur un opérateur non standard peut avoir une valeur arbitraire, mais est inscrite sous la forme d'un programme.

L'information sur chaque opérateur contient des nombres conventionnels suivant lesquels on détermine le type de l'opérateur et son numéro d'ordre dans le schéma du programme.

L'information qui se rapporte au programme dans son entier est mise sous forme de tables.

1) Table du contenu des conditions logiques élémentaires. Elle peut être commune à de nombreux problèmes, mais peut aussi être propre à un problème particulier.

2) Table du stock des cellules de travail. On y indique les blocs de cellules de l'organe mémoire qui peuvent être utilisés comme cellules de travail.

3) Table de répartition de l'organe mémoire pour les nombres. Elle contient les numéros des cellules dans lesquelles sont placées les grandeurs qui participent au calcul ; elle contient en particulier les numéros des cellules dans lesquelles il y a les constantes supplémentaires dont la valeur numérique est déterminée d'avance (avant les calculs).

4) Table de répartition de l'organe mémoire pour les instructions. Elle contient les numéros des cellules de l'organe mémoire dans lesquelles on place le programme actif.

Une fois toute l'information sur le schéma de programme établie et codée sous la forme qui convient, on l'introduit en machine dans un ordre déterminé. On introduit d'abord l'information sur les opérateurs qui rentrent dans le schéma de programme, ensuite les tables énumérées ci-dessus. La commande est transmise au programme de génération qui, dans le résultat du travail, sort le programme de travail du problème et la table des nombres conventionnels avec les adresses qui leur correspondent.

La méthode opératorielle de programmation automatique décrite ici a de grands avantages sur celles qui ont été décrites auparavant. La méthode des sous-programmes-bibliothèque et celle des adresses symboliques ont considérablement moins de possibilités ; elles n'englobent que les étapes particulières de l'exécution d'un programme de génération universel. En particulier, le bloc d'attribution des adresses effectives fait, au fond, le même travail que le programme qui emploie la méthode des adresses symboliques. Les programmes standards peuvent être utilisés au cours de l'exécution du programme de génération, comme opérateurs non standards.

La programmation opératorielle allège et accélère considérablement la préparation d'un programme. Il arrive qu'un programmeur ait affaire à un

schéma de programme qui ne contienne qu'un petit nombre d'opérateurs et non à un programme qui contienne des centaines d'instructions. La codification de l'information peut être faite parallèlement : ce qui accélère le travail et rend possible un contrôle rigoureux.

Ainsi, l'emploi du programme de génération élève la productivité du travail en économisant le temps d'établissement, de mise au point et de vérification du programme.

La méthode opératorielle d'automatisation de la programmation est le début d'une nouvelle programmation. Compte tenu des possibilités de la machine dont on se sert. l'établissement du schéma de programme selon un schéma de calcul sera l'étape d'automatisation suivante.

Il convient de dire, en conclusion, que l'on pousse le travail sur l'automatisation de la préparation des problèmes à résoudre sur C. A. N., dans le sens de la simplification maximale et de la réduction du travail préliminaire consacré à cette préparation par le programmeur. Les possibilités de la C. A. N. font espérer de grands succès dans ce secteur.

4. Méthode des programmes d'interprétation.

La réunion des méthodes énumérées ci-dessus, qui permettent d'automatiser la programmation des problèmes, crée la possibilité de réduire au maximum le travail de préparation des problèmes à résoudre sur C. A. N. Lorsqu'on utilise des bibliothèques de programmes standards pour des schémas de calcul d'un usage général, la majeure partie du travail qui prépare l'information initiale du problème à résoudre en machine est tout de même faite par le programmeur. Par exemple, pour le calcul de la valeur du polynôme

$$f(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n ,$$

on utilise généralement sur C. A. N. un programme standard spécial. Pour l'appliquer, il faut lui donner l'information initiale dont il a besoin : répartir les coefficients du polynôme a_0, a_1, ..., a_n, le degré n du polynôme et l'argument x dans leurs cellules respectives sur lesquelles le travail du programme a été prévu.

Le programme a un travail beaucoup plus compliqué à faire pour préparer les données initiales qui permettent de résoudre le système des équations différentielles ordinaires, par exemple selon la méthode Adams-Stenner. Outre la répartition du contenu de l'organe mémoire, il faut établir un sous-programme pour calculer les valeurs des fonctions qui rentrent dans la partie droite des équations du système et les inclure dans le programme standard général qui résout le système des équations différentielles. Enfin, si le programme standard qui résout une classe entière de problèmes numériques prévoit la possibilité de faire varier son propre schéma conformément aux données initiales du problème, il faut préparer, pour l'utiliser, les conditions qui permettent le choix automatique du schéma de calcul.

Dans tous les cas, la préparation des données initiales d'un problème par le programme standard qui doit le résoudre peut être complètement déterminée de façon algorithmique (admettant donc l'automatisation). Nous appellerons *programme d'interprétation* le programme de l'algorithme qui prépare les données initiales d'un programme standard. Il ne reste qu'à remarquer que les programmes d'interprétation peuvent être construits, non seulement pour des schémas numériques (arithmétiques), mais aussi pour des schémas non arithmétiques (logiques).

Les programmes d'interprétation, en utilisant l'information initiale concernant le problème, dirigent le processus de résolution automatique du problème sur C. A. N. dans laquelle rentrent la programmation des différentes étapes de calcul, l'analyse de l'information initiale et de celle que l'on obtient au cours des calculs, et le choix du schéma de calcul.

A côté des programmes qui interprètent les données initiales du problème pour effectuer les calculs standards, il faut construire des programmes qui interprètent les résultats de ces calculs pour les utiliser ultérieurement ou pour les sortir de la C. A. N. sous une forme habituelle pour son emploi par le programmeur.

2. CONSTRUCTION DES ALGORITHMES D'UN PROGRAMME DE PROGRAMMATION OPÉRATORIELLE

Examinons plus en détail comment se construisent les algorithmes d'un programme de génération. En outre, nous nous bornerons à ne considérer que les questions de principe sur l'exemple de l'élaboration de l'algorithme de programmation des formules arithmétiques.

On commence à élaborer l'algorithme en analysant l'information et en choisissant le moyen de la codifier sur la base du contenu de l'algorithme. Il est clair que l'algorithme de l'élaboration de la codification dépendra du moyen de codifier l'information. Au cours de la construction de l'algorithme, on peut s'apercevoir que l'on est obligé d'imposer à la codification de l'information des conditions supplémentaires.

Lorsqu'on choisit les moyens de donner et de codifier l'information, il faut au départ poser les conditions suivantes :

1) L'information concernant le problème doit avoir autant que possible la forme qu'utilise en général le programmeur pour faire le travail à la main.

2) La quantité de cette information doit être aussi minime que possible.

3) L'algorithme de programmation doit être aussi simple que possible.

L'algorithme de programmation des formules arithmétiques se présente comme suit :

1. Analyse de l'information et choix du procédé de codification.

Les formules arithmétiques contiennent des nombres, des opérations et des parenthèses. Il peut y avoir deux sortes de nombres : des constantes, dont

la valeur est connue avant les calculs par les formules, et des variables, dont la valeur numérique est déterminée au cours des calculs. Pour simplifier, nous ne ferons pas, dans l'information, de différence entre les constantes et les variables, estimant que toutes les grandeurs qui rentrent dans la formule sont des nombres égaux en droits. Les résultats, qui sont obtenus au cours des calculs et qui ne sont pas fixés, sont appelés intermédiaires. Ils sont placés dans les cellules de travail de l'organe mémoire.

Les opérations qui entrent dans les formules peuvent être des opérations à une place, par exemple ln, exp, sin, etc., ou à deux places, par exemple l'addition, la soustraction, la multiplication, la division, etc. La fixation du résultat des calculs est faite à l'aide du signe de l'égalité.

Pour simplifier l'algorithme et l'écriture de l'information, nous examinerons l'écriture des formules sans parenthèses. Par la suite, nous utiliserons les désignations suivantes : nous désignerons les nombres par la lettre a avec un indice, les opérations à une place par la lettre β et celles à deux places par la lettre α.

Détermination des formules dans l'écriture sans parenthèses :

a) Les expressions

$$\beta a_1, \alpha a_1 a_2 ,$$

(où a_1 et a_2 sont des nombres) sont des formules élémentaires ;

b) Les expressions

$$\beta A_1, \alpha A_1 A_2 ,$$

(où A_1, A_2 sont des nombres, des formules élémentaires ou des formules) sont des formules.

Il est évident que n'importe quelle formule en écriture normale (avec parenthèses) peut être transcrite en écriture sans parenthèses. Il faut seulement observer l'ordre des opérations. Par exemple, la formule :

$$(a_1 + a_2) \times \frac{a_3}{\sin a_4} = a_5$$

a, dans l'écriture sans parenthèses, la forme :

$$= \times + a_1 a_2 : a_3 \sin a_4 a_5 .$$

Nous sommes ainsi arrivés à la classification des éléments de l'information en formules arithmétiques (Fig. 19).

Conformément à cette classification, il faut maintenant formuler la règle de codification des éléments de l'information qui entrent dans les formules arithmétiques. Le code de chaque élément de l'information comprend deux parties :

— la première partie contient le code de la forme de l'information : nombre, opération ou signe égal ;

— la seconde partie du code du nombre contient le numéro d'ordre du nombre ; la seconde partie du code de l'opération se divise à son tour en deux parties : l'une contient le code d'une opération à une place (β) ou le code d'une opération à deux places (α), l'autre, le numéro de l'opération ; la seconde partie du signe d'égalité est libre. Ainsi, le code du nombre comprend deux groupes de chiffres, ce que nous désignerons par an. Le code de l'opération comprend trois groupes de chiffres, ce que nous désignerons par $\theta\alpha n$ ou $\theta\beta n$.

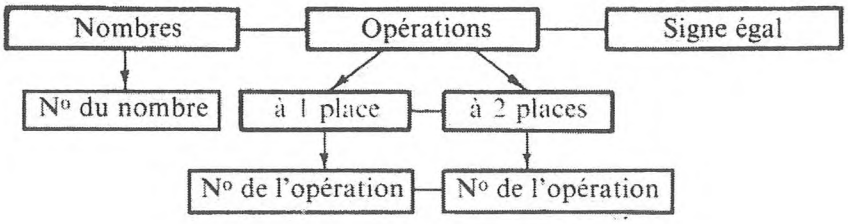


Fig. 19.

2. Formulation de l'algorithme de programmation des formules arithmétiques.

Pour construire un algorithme donné, il faut avant tout formuler nettement son contenu. Procédons de la façon suivante : essayons d'abord de simplifier à l'extrême le contenu de l'algorithme, et ensuite, en le complétant peu à peu, nous nous occuperons de le préciser le mieux possible.

Pour programmer des calculs suivant une formule, l'algorithme doit procéder de la façon suivante : programmer les calculs des formules élémentaires qui entrent dans la formule donnée, en fixant les résultats dans les cellules de travail libres de l'organe mémoire, et remplacer les formules élémentaires de l'information par les résultats des calculs que l'on a effectués sur elles. De plus, au début, pour simplifier, nous n'inclurons pas dans notre examen le signe égal.

En conséquence, on peut diviser l'algorithme de programmation des formules en étapes :

1) Recherche de la première formule élémentaire.

2) Etablissement d'un programme de calcul d'après une formule élémentaire et remplacement de la formule élémentaire par le résultat du calcul effectué sur elle.

La répétition cyclique de l'algorithme permet d'établir un programme de calcul suivant une formule donnée.

Prenons en un exemple : la formule a la forme suivante :

$$\times + \times + a_1\, a_2 : a_3 \sin a_4\, a_5 + a_1\, a_2 .$$

Appliquons l'algorithme de programmation de la formule que nous avons énoncé. La première formule élémentaire est $+ a_1 a_2$. Le programme des calculs que l'on effectue sur elle :

+	a_1	a_2	r_1

Après avoir remplacé cette formule par le résultat, nous obtenons dans l'information la formule :

$$\times + \times \ldots r_1 : a_3 \sin a_4 a_5 + a_1 a_2 .$$

Appliquons de nouveau l'algorithme. La première formule élémentaire est sin a_4, son programme :

sin	a_4	–	r_2

L'information transformée est :

$$\times + \times \ldots r_1 : a_3 \ldots r_2 a_5 + a_1 a_2 ,$$

la première formule élémentaire est $a_3 r_2$, son programme :

:	a_3	r_2	r_3

L'information transformée est :

$$\times + \times \ldots r_1 \ldots r_3 a_5 + a_1 a_2$$

etc.

L'algorithme de programmation des formules que nous avons énoncées ci-dessus est fait pour être utilisé par le programmeur, non pour être réalisé en machine : sa formalisation n'indique pas comment il peut l'être. Il est, avant tout, impossible de comprendre comment s'effectue la recherche de la première formule élémentaire. La description de l'algorithme ne montre pas comment, étant donnée une formule élémentaire, on construit sur elle un programme de calcul et comment cette formule est remplacée par le résultat des calculs. L'analyse de l'exemple examiné nous donnera une plus juste description de l'algorithme.

Appelons *opération exécutable*, l'opération qui entre dans la formule élémentaire. On peut alors formuler les règles suivantes : une opération à une place est exécutable si un nombre la suit immédiatement ; une opération à deux places l'est, si deux nombres consécutifs la suivent immédiatement.

Avant de passer à l'établissement des programmes à adresses, nous remarquerons que, dans l'algorithme que nous avons examiné, le concept d'adresse d'un programme à adresses et celui d'un programme pour une C. A. N. à trois adresses ne concordent pas toujours. Ainsi nous avons besoin de donner une désignation aux parties indépendantes des cellules de l'organe mémoire qui correspondent au code de l'opération et aux première, deuxième et troisième adresses. Nous les désignerons de la façon suivante : $(a)_i$ où $i = 0$, I, II, III. Ainsi, $(a)_0$ est l'adresse de la partie de la cellule d'adresse a qui correspond au code des opérations ; $(a)_{\text{I}}$, $(a)_{\text{II}}$ et $(a)_{\text{III}}$ sont les adresses des parties de la cellule a qui correspondent à la première, deuxième et troisième adresses.

Programme à adresses I. *Recherche de la première opération exécutable.*

k_{11}. $'\varphi_0 \Rightarrow \varphi_1$

k_{12}. $P\{{}^2\varphi_1 = !\}\ ARR$
$C\ '\varphi_1 \Rightarrow \varphi_1$
$P\{{}^2\varphi_1 \neq \theta\}\ k_{12}$
$'\varphi_1 \Rightarrow \varphi_2$

k_{13}. $C\ '\varphi_2 \Rightarrow \varphi_2$
$P\{{}^2\varphi_2 = 0\}\ k_{13}$
$P\{{}^2\varphi_2 \neq a\}\ k_{12}$
$P\{{}^2\varphi_1 = \beta\}\ k_{21}$
$'\varphi_2 \Rightarrow \varphi_3$

k_{14}. $C\ '\varphi_3 \Rightarrow \varphi_3$
$P\{{}^2\varphi_3 = 0\}\ k_{14}$
$P\{{}^2\varphi_3 \neq a\}\ k_{12}\, k_{22}$

φ_0 fixe le début de la formule ; ψ désigne l'adresse de la cellule libre suivante, dans la suite affectée au programme de travail à établir ; ω désigne l'adresse de la cellule de travail libre suivante.

On recherche d'abord le premier code d'opération à gauche (il fixe φ_1).

On cherche ensuite le premier élément de la formule qui ne soit pas nul. Si ce n'est pas un nombre, l'opération n'est pas exécutable. La commande passe à l'opérateur k_{12}. Si après le code de l'opération, il y a un nombre, la forme de l'opération est déterminée. Si l'opération n'a qu'une place, elle est exécutable et la commande passe à la seconde étape (à l'opérateur k_2). Si elle a deux places, on cherchera le prochain élément non nul de la formule. Si cet élément n'est pas un nombre, la commande passe de nouveau à k_{12} — recherche du code de l'opération suivant. Si c'est un nombre, l'opération à deux places est exécutable, la commande passe à la seconde étape k_{22}.

Ici, ! est le symbole de la fin de formule, θ le symbole de l'opération, β le symbole d'une opération à une place, a le symbole de la grandeur.

Programme à adresses II. *Programmation de la formule élémentaire et son remplacement par le résultat du calcul fait sur elle.*

k_{21}. $^{2}\varphi_1 \Rightarrow (\psi)_0$
$^{2}\varphi_2 \Rightarrow (\psi)_I$
$'\omega \Rightarrow (\psi)_{III}$
$0 \Rightarrow '\varphi_1$
$'\omega \Rightarrow '\varphi_2$
Pk_{23}.

Nous remarquerons que le code de l'opération exécutable de la formule fixe φ_1 et que le code du nombre fixe φ_2. Si l'opération exécutable a deux places, le code du second nombre fixe φ_3.

L'opérateur k_{21} commence le programme qui établit l'instruction pour une opération exécutable à une place et qui remplace la formule élémentaire correspondante par l'adresse de la cellule de travail.

k_{22}. $^{2}\varphi_1 \Rightarrow (\psi)_0$
$^{2}\varphi_2 \Rightarrow (\psi)_I$
$^{2}\varphi_3 \Rightarrow (\psi)_{II}$
$'\omega \Rightarrow (\psi)_{III}$
$0 \Rightarrow '\varphi_1$
$0 \Rightarrow '\varphi_2$
$'\omega \Rightarrow '\varphi_3$

L'opérateur k_{22} commence le programme qui établit l'instruction pour une opération exécutable à deux places et qui remplace la formule correspondante par l'adresse de la cellule de travail libre.

k_{23}. $C\,'\omega \Rightarrow \omega$
$C\,'\varphi \Rightarrow \psi$
Pk_{11}.

L'opérateur k_{23} déplace les fixateurs ω et ψ, c'est-à-dire prépare l'adresse de la cellule de travail libre et l'adresse de l'instruction suivante du programme de travail.

En conclusion, nous remarquerons que les complications d'un algorithme de programmation des formules arithmétiques ne soulèvent pas de sérieuses difficultés. Par exemple, admettons que la formule contienne le signe égal. Il est évident que l'égalité peut être considérée comme une opération à deux places qui détermine la formule de forme :

$$= ab\ .$$

Le programme qui construit l'instruction correspondant à une telle formule

+	a	−	b

peut être facilement établi par le lecteur.

3. MÉTHODE DE LA PROGRAMMATION A ADRESSES

La programmation à adresses, exposée dans le chapitre V, représente aussi une des méthodes d'automatisation de la programmation.

Le langage algorithmique est considérablement simplifié et les algorithmes peuvent être écrits sous une forme facile à examiner, si l'on utilise des adresses

de rangs supérieurs et des schémas qui examinent l'information d'après eux, notamment, comme nous l'avons montré (Chapitre V), lorsqu'il s'agit de problèmes logiques complexes. La programmation à adresses utilise le langage courant des formules mathématiques et logiques, ce qui explique la simplicité de son application.

La vérification des algorithmes à adresses est d'autant plus facilitée que les programmes à adresses conservent une écriture invariable tout au long de leur exécution.

Au niveau de développement que les mathématiques ont atteint actuellement en machine, le langage à adresses possède le grand avantage de pouvoir être utilisé sous la forme de programmes à adresses pour un échange d'informations entre des organismes qui se servent de machines de types différents. C'est, en effet, un langage algorithmique universel puisque l'on ne tient pas compte des particularités individuelles de la machine, mais seulement des principes généraux de résolution des problèmes ; le principe d'adressage et celui de commande automatique (à programmes).

D'autre part, la programmation à adresses peut servir de base à la construction d'algorithmes de conversion automatique des programmes à adresses en programmes de machines existantes — programmes pour des machines qui utilisent l'algorithme à adresses comme information (cf. Iouchtchenko E. L., Bistrov L. P., [1]).

Pour construire de tels programmes, on peut utiliser les idées exposées dans la description des programmes de génération fondés sur la méthode opératorielle. De plus, il est nécessaire d'inclure dans le programme de génération des blocs spéciaux, par exemple un bloc qui diminue le rang de l'adresse, et un qui programme les opérations qui ont des adresses de second rang (*), s'il s'agit de la programmation pour une machine qui possède la gamme d'opérations décrite dans le chapitre II, c'est-à-dire d'une machine qui n'accomplit que des opérations sur des adresses de premier rang. Au fur et à mesure du développement de ce langage, les programmes fondés sur la programmation à adresses peuvent être perfectionnés comme ceux fondés sur la programmation opératorielle.

(*) Il est possible de montrer que le programme à adresses, qui inclut une adresse de rang supérieur à 2, peut être transformé en un programme équivalent, qui ne contient que des adresses qui n'ont pas un rang supérieur à 2 (cf. Iouchtchenko E. L., [1]).

CHAPITRE VII

ORGANISATION DU TRAVAIL SUR C. A. N.

1. MISE EN FORME DES PROGRAMMES SUR LES FEUILLES DE PROGRAMMATION

Il est très important pour résoudre un problème sur C. A. N. d'organiser l'élaboration, le contrôle et l'entrée des programmes et de contrôler l'exactitude de l'exécution des programmes par la machine.

Déjà, pour un calcul à la main, les programmes sur papier doivent être présentés avec le maximum de soin et d'exactitude. Il en est de même lorsqu'on travaille à l'aide de programmes de génération ; l'information des problèmes doit être mise en forme d'une façon particulièrement consciencieuse. Après l'établissement des programmes en écriture alpha-numérique, la mémoire de la machine doit être répartie entre le programme et les nombres, et réécrite sous forme d'adresses réelles, ce qui la rend au travail.

Il ne faut pas perdre de vue que les adresses des cellules de l'organe mémoire de la machine sont numérotées en binaire. Pour raccourcir l'écriture, on utilise généralement les systèmes numériques à base huit et à base seize. Ce qui fait que, sur les feuilles de programme, les codes d'opération, les adresses des instructions et les nombres conditionnels (les constantes de substitution d'adresse, les codes des instructions de comparaison, etc.) sont aussi notés dans le système à base huit ou à base seize. De plus, le code est divisé en groupes de chiffres à base huit ou à base seize : l'un d'eux correspond au code de l'opération, les autres aux adresses ; en outre les positions de poids fort qui manquent dans chaque groupe ne sont pas omises, mais représentées par des zéros. Le système décimal n'est utilisé que pour les nombres décimaux. Pour écrire les programmes de travail ou les informations d'un problème, en programmation automatique, il convient d'utiliser des feuilles de programmation normales et d'affecter une ligne particulière à chaque instruction ou nombre. Sur les feuilles de programmation il y a, outre les colonnes destinées aux codes des instructions et des nombres, une colonne affectée à l'écriture alphabétique des grandeurs dans la partie numérique du programme (dans la notation adoptée dans le schéma de calcul) et des résultats des opérations dans les

Tableau 9. *Modèle d'une feuille de programmation standard pour machines à trois adresses.*

Numéro d'identification du problème	Feuille n°	Nombre total de feuilles	Numéro de la section	Etabli par	Vérifié par	Validé par

Numéro de la cellule	Code opération	A_1	A_2	A_3	Impression sur organe de contrôle	Résultat du contrôle	Notation alphabétique	Variation en fonction des paramètres	Commentaires

instructions. Dans les instructions de transfert de commande peuvent être inscrites les conditions logiques appropriées. Ensuite, une colonne, utilisée au cours de la mise au point des programmes, est affectée à l'indication du résultat des calculs, par exemple lorsque certaines instructions « de branchement » sont exécutées pour la première fois ; il y a aussi une colonne pour indiquer la variation des adresses des instructions en fonction des paramètres. En outre, on peut désigner d'une façon spéciale les instructions variables, par exemple entourer le numéro de l'instruction d'un trait rouge.

Lorsqu'il y a un organe de contrôle imprimant, on réserve sur les feuilles de programmation une colonne que le programme sur le clavier remplit au cours de sa perforation sur rubans ou cartes.

Une ligne en haut des feuilles de programmation sert à indiquer le numéro d'identification du problème, le numéro de la feuille, le nombre total de feuilles dans le programme, ainsi que les numéros de la section de l'organe mémoire dans laquelle le programme doit être enregistré.

Il faut deux personnes pour vérifier l'exactitude des programmes : le programmeur et un vérificateur qui apposeront leur signature sur chaque feuille de programmation.

Les feuilles de programmation préparées et le schéma de calcul seront rangés ensemble dans une chemise.

Il est utile d'inscrire toutes les corrections des fautes que l'on a découvertes dans le programme sur des feuilles de programmation séparées et de n'introduire dans le programme que les renvois correspondants.

2. ENTRÉE DES PROGRAMMES DANS L'ORGANE MÉMOIRE ET CONTRÔLE D'ENTRÉE

Comme on l'a déjà noté, sur la plupart des C. A. N. actuelles, on utilise des rubans ou des cartes perforées pour introduire les programmes dans la mémoire opération ou dans l'organe mémoire externe. La perforation du programme de travail sur cartes ou sur rubans est faite sur des perforatrices spéciales. Pour éviter d'y introduire des fautes, on a mis au point une série de procédés spéciaux. Ainsi, sur la plupart des machines la perforation est-elle faite par deux personnes qui travaillent indépendamment sur des perforatrices différentes. Ensuite, l'ensemble des programmes est vérifié sur un appareil spécial : le contrôleur. Les non-coïncidences découvertes sont éliminées. En outre, il existe des imprimantes : le programme en sort perforé et on le vérifie d'après ce qui a été initialement inscrit sur les feuilles de programmation.

Sur certaines perforatrices, il y a un appareil de contrôle spécial : la feuille de programmation écrite est placée sur le chariot de la perforatrice et le contenu de la première ligne est recueilli sur le clavier ; ensuite, on appuie sur une touche spéciale « imprimer sur l'organe de contrôle » et cette première ligne est imprimée, dans la colonne prévue à cet effet ; on compare le contenu de la

feuille de programmation à celui de l'ensemble imprimé ; s'il y a concordance, on appuie sur le bouton « exécution » qui déclenche la perforation correspondante sur bande ou sur feuille de programmation. De plus, on imprime sur cette dernière un signe conventionnel : il indique que cette ligne a été codée ; on passe ensuite à la perforation de la ligne suivante.

Lorsqu'on transfère les codes sur des cartes perforées ou des rubans, c'est par un moyen purement mécanique que les codes octaux (à base seize) ou décimaux sont transformés en codes binaires ou décimaux codés binaires.

Au moment de l'entrée des instructions en mémoire, les numéros des cellules qui font partie de leurs adresses et les codes des opérations sont ainsi déjà écrits en binaire. Pour faire les calculs, les codes des nombres décimaux codés binaires sont convertis en binaire soit au cours de l'entrée, soit à l'aide d'un sous-programme spécial.

Pour distinguer nombres et instructions au cours de l'entrée, on les groupe en blocs séparés et on les introduit à l'aide d'instructions différentes « entrée des nombres » ou « entrée des instructions » ou, lorsqu'on les code, on prévoit un signe spécial « symbole de nombre décimal ».

Lorsque le programme est introduit en machine, il peut y avoir des défauts dans les organes de lecture, de commande et de mémoire. A cette étape, il peut déjà y avoir des altérations dans le résultat. Pour vérifier si l'entrée est exacte, on utilise le plus souvent l'addition cyclique, qui s'appelle aussi totalisation de contrôle à cause de la fonction qu'elle remplit. Etant donnés deux ensembles pour un seul et même programme, ils sont introduits indépendamment, et tous les codes du programme sont totalisés les deux fois à l'aide de l'addition cyclique. Les totaux de contrôle sont ensuite comparés. Lorsqu'il n'y a pas concordance des programmes, l'égalité des totaux de contrôle est très peu probable, c'est donc une garantie de l'introduction correcte du programme. Mais si les totaux de contrôle ne coïncident pas, il faut vérifier le travail. Si l'on n'a qu'un seul exemplaire du programme, on l'introduit deux fois et l'on compare les totaux de contrôle exactement de la même façon. La perte de temps que nécessite l'organisation d'un tel contrôle est justifiée par une entrée plus sûre.

3. VÉRIFICATION DE L'EXACTITUDE DU TRAVAIL DE LA MACHINE

1. Vérification de la machine dans des buts « prophylactiques ».

On soumet périodiquement la machine à un contrôle « prophylactique » pour exclure les fautes systématiques, dues au mauvais état de ses circuits et à leur fonctionnement défectueux. Dans ce but, on résout ce que l'on appelle des *problèmes-tests*, ou des *tests*, lorsque les modes de contrôle se sont parti-

culièrement altérés. En principe, on fait subir à la machine une vérification par des tests avant la mise au point ou la résolution de tout nouveau problème.

Les problèmes-tests sont faits de telle sorte que leur résolution intéresse tous les éléments de la machine. Le plus souvent, on établit des tests spéciaux pour contrôler les différents organes de la machine.

Il convient de placer les programmes-tests dans l'organe mémoire externe, tandis que la mémoire interne conserve les instructions initiales dans des cellules spéciales (qui ne peuvent pas être occupées par d'autres instructions), car on peut avoir besoin d'appeler les tests chaque fois qu'il y a dans un problème une faute systématique ou un travail incorrect. Cela permet d'aiguiller la machine sur un travail d'après les tests sans perdre de temps sur leur entrée.

Ce n'est qu'après l'élimination des défauts révélés par la résolution des problèmes-tests que l'on procède à la mise au point des programmes.

Par exemple, pour vérifier le totalisateur de l'organe arithmétique, le programme-test peut être construit de la façon suivante : à partir de certains nombres x_0, ... et a_0, ... on forme les suites de nombres x_1, x_2, ... et a_1, a_2, ... (par exemple, à l'aide de l'instruction de l'addition cyclique) et l'on effectue les opérations de multiplication et de division $a_k x_k$ et $\frac{a_k x_k}{x_k}$. De plus, on forme les nombres x_k et a_k de façon que, lorsqu'on multiplie, on ne dépasse pas une position (pour la machine à virgule flottante). Si

$$\left| a_k - \frac{a_k x_k}{x_k} \right| > \varepsilon$$

(les erreurs d'arrondi entrant dans ε lorsque l'on fait la multiplication et la division), alors il y a arrêt : cela signifie que le défaut est dû au totalisateur de l'organe arithmétique. Pour en localiser ensuite l'endroit précis, on emploie des opérations plus simples (par exemple, l'addition) avec des codes spécialement choisis ; si cela ne donne aucun résultat, on utilisera des moyens techniques pour faire la vérification.

Le programme-test, qui permet de vérifier l'organe mémoire, effectue l'envoi des différents nombres dans toutes les cellules de cet organe pour les comparer ultérieurement aux nombres qui sont formés dans une des cellules standards.

Dans les machines à tubes à rayons cathodiques, des appels réitérés à une seule cellule ou à des cellules voisines peuvent causer des erreurs : ce qui oblige à construire des programmes pour détecter ce genre d'inexactitudes.

2. Contrôle de l'exactitude du travail de la machine au cours de la résolution des problèmes.

Même lorsque le programme est contrôlé, mis au point et correctement introduit dans l'organe mémoire, et que le travail de la machine sur les tests est valable, on ne peut pas garantir que la résolution du problème soit correcte.

Des pannes de la machine peuvent provoquer des erreurs fortuites qui altèrent les résultats.

C'est pourquoi il est très important d'utiliser le contrôle automatique (par programme) pour empêcher ces erreurs de nuire aux calculs et à leur résultat. Il est indispensable de prévoir un tel contrôle pour résoudre un calcul qu'il soit compliqué ou non.

Comme on le fait lorsqu'on contrôle l'entrée des programmes en machine, on peut aussi utiliser l'addition cyclique pour contrôler la marche des calculs.

Dans ce but, on prévoit, à des étapes déterminées du programme, une totalisation cyclique des codes des instructions et des nombres ou de certains résultats de calcul. Le total de contrôle obtenu est mis en mémoire et les calculs de cette même étape sont répétés. Les totaux de contrôle du premier et du second calcul sont comparés pour savoir s'il y a coïncidence totale. Si oui, on passe à l'étape suivante, etc. Si non, on prévoit un arrêt ou la répétition des calculs. Dans le dernier cas, le troisième total de contrôle est comparé à chacun des deux précédents. S'il y a concordance de l'un des couples de totaux de contrôle, on passe aux calculs suivants. La non-concordance témoigne qu'une panne s'est produite et qu'il est nécessaire de prendre des mesures pour éliminer les erreurs.

L'organigramme des calculs et des calculs de contrôle est représenté par la figure 20.

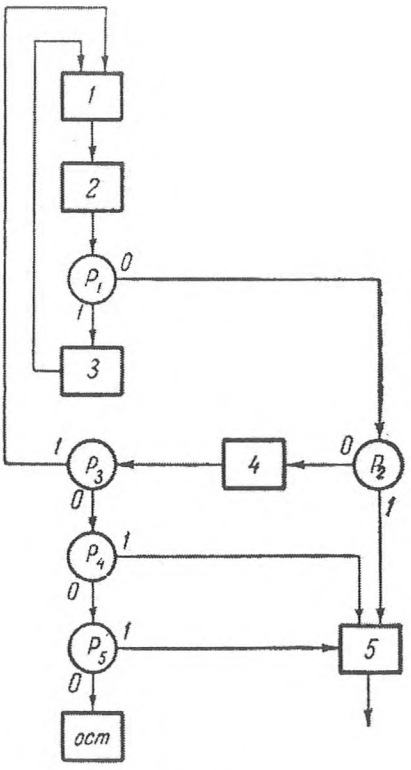


Fig. 20.

On y introduit la notation suivante : 1 est l'étape numérique ; 2 la totalisation de contrôle ; $P_1 = P\{n = 1\}$, où n est le numéro de l'étape de calcul en cours ; 3 la mémorisation du total de contrôle S_1 ; $P_2 = P\{S_1 = S_2\}$ la condition de concordance des deux premiers totaux de contrôle ; 4 la mémorisation du total de contrôle S_2 ;

$$P_3 = P\{n = 2\}; \quad P_4 = P\{S_3 = S_1\}; \quad P_5 = P\{S_3 = S_2\};$$

5 la continuation des calculs. Dans les schémas qui n'ont qu'une erreur de calcul binaire, on met un arrêt à la place de 4 et on supprime P_3, P_4, P_5.

Bien entendu, le doublage automatique des calculs étape par étape augmente le nombre des instructions et utilise des cellules supplémentaires de la mémoire opération ou de la mémoire externe pour conserver les données initiales à chaque étape. On peut appliquer les schémas analogiques de ce doublage, puis au lieu de procéder à une totalisation de contrôle, on teste la coïncidence totale de certains résultats de calcul. Lorsqu'une panne altère une instruction du programme, les calculs suivants peuvent donner des résultats qui coïncident lors du doublage de contrôle. Il en ressort qu'on peut compliquer quelque peu la méthode de contrôle par doublage, par exemple, en exécutant séparément une totalisation de contrôle des instructions, disons, au moment où les instructions prennent leur forme initiale.

Le schéma de contrôle décrit ci-dessus peut ne pas donner de résultat, si, lors d'une panne, il y a des erreurs qui se compensent l'une l'autre. En outre, il peut y avoir au cours de la résolution, une erreur systématique qui ne sera pas non plus découverte. C'est pourquoi, si c'est possible, on doit appliquer des méthodes de contrôle qui utilisent la signification du contenu des résultats des calculs, par exemple, effectuer le calcul par deux méthodes différentes ou vérifier les différentes relations auxquelles doivent satisfaire les résultats.

a) Par exemple, dans le calcul de la table des sinus et des cosinus, on peut vérifier si les résultats que la machine obtient satisfont à la relation

$$\sin^2 x + \cos^2 x = 1 .$$

L'égalité est, bien entendu, vérifiée à ε près, ε étant l'erreur admissible.

Dans certains cas, lorsqu'on résout des équations algébriques et différentielles, il est utile de prévoir dans le programme une vérification des solutions au moyen de la substitution, dans les équations initiales, des valeurs numériques des grandeurs cherchées (et de leurs dérivées dans le cas d'équations différentielles) ; les calculs ne se poursuivent que si les résultats de la substitution sont satisfaisants.

b) Dans certains cas, on parvient à introduire des grandeurs de contrôle auxiliaires dont on connaît d'avance la liaison avec les résultats. Les grandeurs de contrôle se calculent en même temps que les autres grandeurs et on vérifie, à des étapes numériques déterminées, si la liaison entre les grandeurs

initiales et les grandeurs auxiliaires est satisfaite, ce qui est prévu par le programme.

Par exemple, lorsqu'on résout un système d'équations algébriques linéaires par la méthode de Gauss, on examine, outre le système initial, le système composé par la même matrice de coefficients et par les termes libres, égaux aux totaux des éléments des lignes de la matrice des coefficients, y compris les termes libres. Chaque racine du système auxiliaire sera d'une unité supérieure à la racine correspondante du système initial, ce qui est aussi un moyen de contrôle.

Quelquefois, on peut juger de la marche correcte des calculs par la monotonie ou l'absence de modifications soudaines dans les différents résultats du calcul.

Le contrôle, s'il est assez complet, c'est-à-dire s'il englobe toutes les étapes de calcul, donne presque toujours la possibilité de fixer le moment où se produit la panne ; celle-ci découverte, il est indispensable de réintroduire le programme en machine et de continuer les calculs sur le programme corrigé. Il est évident qu'il est utile de poursuivre les calculs à partir du moment où les totaux de contrôle concordent (ou les conditions de contrôle sont satisfaites) pour la dernière fois.

A cette fin, lorsqu'il y a concordance des totaux de contrôle, il est indispensable de prévoir dans le programme la mémorisation des valeurs des paramètres variables du programme qui déterminent la marche ultérieure des calculs.

La résolution des problèmes qui ont un grand nombre de variantes est liée, lors du passage automatique d'une variante à l'autre, aux difficultés que l'on a de déterminer à quelles valeurs des paramètres du problème qui déterminent les variantes a lieu la panne. Pour ces problèmes, il est possible de prévoir la mémorisation des valeurs des paramètres déterminant la variante suivante après l'erreur certaine de la variante qui précède. En cas de panne, le programme est réintroduit, mais on utilise les valeurs indiquées des paramètres pour continuer les calculs. On peut mémoriser les valeurs des paramètres en les sortant sur des cartes ou des rubans perforés. Sur la *Strela* on a l'habitude d'appeler *carte de panne* la carte qui contient l'ensemble des valeurs des paramètres qui déterminent la variante altérée par le mauvais fonctionnement.

4. MISE AU POINT DES PROGRAMMES

L'expérience que l'on a acquise pour résoudre des problèmes sur machines numériques automatiques prouve que la vérification la plus soigneuse laisse encore passer des fautes dans le programme. La recherche de ces fautes, la préparation définitive du programme du point de vue calcul, ainsi que la détermination du temps dont on a besoin pour résoudre entièrement le problème, entrent toutes dans la mise au point du programme en machine. En

outre, on peut, pendant la mise au point, résoudre des questions particulières au programme en cours, par exemple le choix du pas d'intégration qui assure l'exactitude indispensable, si le choix automatique de celui-ci rencontre des difficultés, ou le choix des échelles.

En principe, la mise au point consiste à exécuter successivement les opérations des différentes parties du programme et à sortir à chaque étape du calcul les résultats intermédiaires et le contenu de certaines cellules de la mémoire. En comparant les résultats obtenus en machine aux calculs faits à la main, on peut juger de l'exactitude des opérateurs arithmétiques. Le contenu des différents compteurs et des instructions variables peut attirer l'attention sur des fautes qui portent atteinte à la logique du programme. On peut exécuter la sortie des résultats sur l'imprimante à partir du pupitre de commande après les arrêts de contrôle prévus à la fin de chaque partie contrôlée. Les arrêts de contrôle sont effectués soit à l'aide d'un symbole spécial (signal de contrôle), s'il y en a un qui a été prévu sur la machine, soit à partir du pupitre de commande (*Oural*), soit par d'autres moyens.

Il est possible d'automatiser le processus de mise au point en introduisant dans le programme des instructions qui permettent, sans qu'il y ait d'arrêt, de contrôler successivement les parties du programme et la sortie des résultats. Etablir un tel programme nécessite souvent un gros travail supplémentaire ; c'est pourquoi, il serait expédient pour la mise au point d'utiliser des programmes de contrôle universels.

L'idée d'un programme universel de mise au point est la suivante : on extrait successivement les instructions du programme à contrôler et on les exécute à l'intérieur du programme de mise au point. Après avoir exécuté certaines instructions fixées d'avance, on fait la comparaison des résultats intermédiaires et, en cas de non-concordance, on corrige le résultat erroné, on imprime les fautes et on passe au contrôle des instructions suivantes. En outre, on a la possibilité, dans le programme de mise au point, d'omettre des parties du programme à contrôler, de raccourcir certains cycles, d'imprimer des instructions, etc.

L'ordre de la mise au point est donné par le programme de contrôle sous la forme d'une information codée de façon spéciale.

Un programme dont l'ordre est bien agencé permet de découvrir rapidement les fautes et d'économiser du temps-machine.

APPENDICE

I. B. E. S. M.

Les travaux de réalisation de la Calculatrice Electronique Rapide (B. E. S. M.) ont été dirigés par l'Académicien S. A. Lebedev en 1951 à l'Académie des Sciences d'U. R. S. S. Cette machine possède 39 positions pour le code en système à virgule flottante. Pour coder la caractéristique, on utilise 6 positions (la première correspond au signe de la caractéristique) et pour la mantisse 33 positions (la première correspond au signe de la mantisse), comme le montre la figure 21.

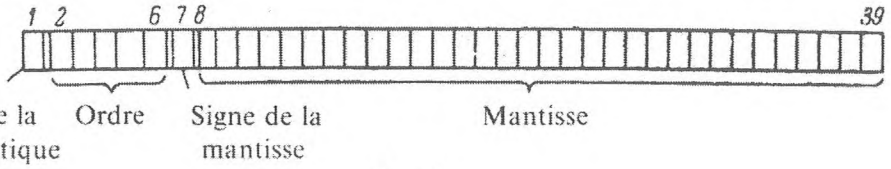


Fig. 21.

Pour les instructions, on utilise un système à 3 adresses : 6 positions sont réservées au code du type d'opération, 11 positions à chaque adresse.

Les calculatrices rapides font de 8 000 à 10 000 opérations à la seconde. Les mémoires internes sont composées de 2 047 cellules, les mémoires externes sont composées de 2 tambours et de 4 bandes magnétiques. La capacité d'un tambour est de 5 120 codes, celle des bandes de 30 000 (les bandes sont interchangeables). L'entrée, ainsi que la sortie, se fait par cartes ou bandes perforées.

Contrairement aux autres machines soviétiques (Strela, Kiev, Oural) la B. E. S. M. est une machine à 2 commandes, centrale et locale, ce qui représente un certain avantage au cours de l'exploitation des sous-programmes.

Dans la B. E. S. M., en plus du registre d'instruction K, il y a 2 compteurs d'instructions (*) :

C_1, compteur d'instructions de la commande centrale,
C_2, compteur d'instructions de la commande locale.

De plus, il y a un basculeur de commande T. La présence d'un « un » dans le basculeur T correspond au travail de la commande centrale, celle d'un « zéro » correspond au travail de la commande locale.

(*) Ici, nous n'indiquerons que les registres et les interrupteurs qui nous seront utiles dans la description des opérations élémentaires effectuées par cette machine.

Le cycle de base est défini par l'exécution d'une instruction $'K$, (c'est-à-dire d'une instruction dont le code est contenu dans le registre d'instructions K) et par le transfert, dans ce registre, du code de l'instruction suivante.

Pour décrire les opérations en B. E. S. M., nous diviserons le registre d'instructions K comme le montre le tableau 10.

Tableau 10

Positions du registre d'instructions de la B. E. S. M. (numérotées de gauche à droite)	Notation et dénomination des groupes de positions (des registres)
1-6	K_0, registre du code opération
7-17	K_1, registre de l'adresse 1
18-28	K_2, registre de l'adresse 2
29-39	K_3, registre de l'adresse 3

De plus, quand nous aurons besoin de la i-ième position du registre K, nous la désignerons par ε_i.

La figure 22 montre cette division du registre en parties (et les cellules où on inscrit l'instruction).

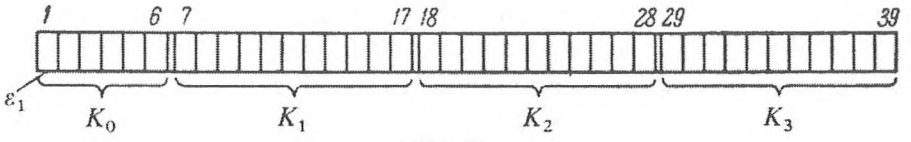


Fig. 22.

Pour écrire $'K_0$ nous utiliserons le système binaire. $'K_0$ pourra varier de 000000, 000001, ... à 111111.

Nous noterons T_1 l'interrupteur d'arrêt conditionnel qui est dans la machine.

En général, l'exécution des instructions (arithmétiques, logiques, auxiliaires qui servent dans les calculs) comprend :

1) la transformation des codes et leur transfert suivant l'adresse, c'est-à-dire l'exécution d'une opération du type

$$f({}^2K_1, {}^2K_2) \Rightarrow {}'K_3 ;$$

2) l'augmentation d'une unité dans le compteur C_1 si $'T = 1$, ou dans le compteur C_2 si $'T = 0$:

$$'C_1 + 'T \Rightarrow C_1 ;$$

$$'C_2 + (1 - 'T) \Rightarrow C_2 ;$$

3) le passage à l'instruction suivante dont le numéro se trouve dans le compteur C_1 si $'T = 1$. ou dans C_2 si $'T = 0$:

$$'('C_1\ 'T + 'C_2(1 - 'T)) \Rightarrow K\,.$$

Ainsi les opérations particulières à la B. E. S. M. sont celles qui changent l'état du basculeur T.

I. Opérations de transfert de la commande locale à la centrale et vice versa

1. $'K_0 = 011000$, transfert vers la commande locale (TCL) :

a) $0 \Rightarrow T$;

b) $^2C_2 \Rightarrow K$.

Ainsi les instructions sont exécutées à partir du numéro conservé dans le compteur de la commande locale C_2.

2. $'K_2 = 011001$, transfert vers la commande centrale (TCC) :

a) $1 \Rightarrow T$;

b) $^2C_1 \Rightarrow K$.

Ainsi les instructions sont exécutées à partir de l'adresse conservée dans le compteur de la commande centrale C_1.

Le contenu des registres K_1, K_2, K_3 importe peu.

3. $'K_0 = 011010$, changement du compteur de la commande locale ($CCCL$) :

a) $0 \Rightarrow T$;

b) $'K_3 \Rightarrow C_2$;

c) $^2C_2 \Rightarrow K$.

Ainsi, si le travail s'effectuait sur la commande centrale, il continue sur la commande locale, en commençant par l'adresse de $'K_3$. S'il s'effectuait sur la commande locale, il continue à partir de l'adresse de $'K_3$. Les contenus de K_1 et K_2 n'influent en rien sur le résultat de l'opération.

4. $'K_0 = 011011$, changement du compteur de la commande centrale ($CCCC$) :

a) $'C_1\ 'T + 'C_2(1 - 'T) + 1 \Rightarrow ('K_2)_3$; $011011 \Rightarrow ('K_2)_0$;

b) $1 \Rightarrow T$;

c) $'K_3 \Rightarrow C_1$;

d) $^2C_1 \Rightarrow K$.

L'opération s'effectue de la même manière que la précédente ; cependant l'instruction, dont le code opération est *CCCC* (*), est envoyée à l'adresse de $'K_2$, alors qu'il y a, dans la troisième adresse, l'adresse de l'instruction en cours de traitement, augmentée d'une unité (*a*). Le contenu du registre K_1 n'influe en rien sur le résultat de l'opération. L'opération *CCCL* est pratique pour passer à un sous-programme. Dans ce but, le programme principal est exécuté par la commande centrale et les sous-programmes par la commande locale. Dans l'adresse A_3 de cette instruction l'on a l'adresse de l'instruction initiale du sous-programme.

Pour retourner au programme principal l'on utilise l'opération *TCCI* qui permet de le reprendre à l'endroit où il a été interrompu (à l'instruction qui suit l'instruction *CCCL* qui a permis le passage au sous-programme).

Il est donc évident que les instructions qui font passer de la commande locale à la commande centrale doivent, au cours de l'exécution d'un sous-programme, être considérées à part. Ainsi, si au cours d'un sous-programme l'on a besoin de revenir au programme, on peut utiliser l'instruction *CCCC*. Cette instruction permet de fixer dans la troisième adresse de la cellule dont l'adresse est indiquée en K_2 le contenu du compteur de la commande centrale des instructions (endroits de sortie du programme principal vers le sous-programme).

Voici maintenant la liste des autres opérations que peut exécuter la machine B. E. S. M..

II. Opérations de transformation et de transfert de codes

1. $'K_0 = 000001$ (+), addition :

a) $^2K_1 + {}^2K_2 \Rightarrow {}'K_3$;
b) $'C_1 + 'T \Rightarrow C_1$; $'C_2 + (1 - 'T) \Rightarrow C_2$;
c) $'('C_1\ 'T + 'C_2(1 - 'T)) \Rightarrow K$.

Avant l'addition des nombres, on égalisera leur ordre ; le résultat est normalisé et arrondi.

2. $'K_0 = 100001$ (, +), même opération que pour $'K_0 = 000001$, mais avec normalisation bloquée.

Les instructions suivantes se déroulent de manière identique :

3. $'K_0 = 000010$ (−), soustraction.

4. $'K_0 = 100010$ (, −), soustraction avec normalisation bloquée.

(*) Rappellons que la notation $(\ )_3$ indique les positions qui correspondent à la troisième adresse de la cellule dont l'adresse est indiquée entre parenthèses ( ).

5. $'K_0 = 000011$ ($\times$), multiplication.

6. $'K_0 = 100011$ (, $\times$), multiplication avec normalisation bloquée.

7. $'K_0 = 000100$ (:), division.

8. $'K_0 = 000101$ ($+ C$), addition des caractéristiques. La caractéristique de $^2K_2$ est ajoutée à la caractéristique de $^2K_1$, et le résultat normalisé est envoyé à l'adresse de $'K_3$.

9. $'K_0 = 100101$ (, $+ C$), même opération, mais avec normalisation bloquée.

10 et **11.** $'K_0 = 000110$ et $'K_0 = 100110$ ($- C$) et (, $- C$), soustraction des caractéristiques. Même chose que pour $'K_0 = 000101$ et $'K_0 = 100101$ si ce n'est qu'il s'agit d'une soustraction.

12. $'K_0 = 000111$ (CCA), changement de caractéristique suivant adresse. A la caractéristique du nombre $^2K_1$ on ajoute le nombre $'K_2$ et le résultat normalisé est envoyé à l'adresse de $'K_3$. La sixième position de K_2 est utilisée pour le signe.

13. $'K_0 = 100111$ (, CCA), même opération, mais avec normalisation bloquée.

14 et **15.** $'K_0 = 001000$ ($\times a$) et $'K_0 = 001001$ ($\times b$), multiplication, résultat donné avec un nombre de positions doublé. Deux instructions servent à exécuter cette opération : premièrement $'K_0 = 001000$, qui est la multiplication des nombres $^2K_1$ et $^2K_2$ sans arrondi, la normalisation et l'envoi des 32 premières positions du résultat avec leur caractéristique à l'adresse de $'K_3$, et deuxièmement $'K_0 = 001001$ (instruction qui suit immédiatement la précédente), qui est la normalisation des 32 positions du totalisateur qui correspondent aux poids les plus faibles (avec la caractéristique obtenue après l'instruction précédente), et l'envoi à l'adresse de $'K_3$ ($'K_1$ et $'K_2$ ne sont pas utilisés dans cette dernière instruction).

16 et **17.** $'K_0 = 101000$ et $'K_0 = 101001$, même opération que précédemment, mais avec normalisation bloquée.

18 et **19.** $'K_0 = 001010$ ($: a$) et $'K_0 = 001011$ ($: b$), analogue à l'opération précédente, effectue la division avec sortie du reste.

20. $'K_0 = 001100$ (TN), transfert normal d'un nombre. Le nombre $^2K_1$ est normalisé, puis envoyé à l'adresse de $'K_3$.

a) $(^2K_1)^n \Rightarrow {}'K_3$.

Les points *b*) et *c*) comme pour l'addition ; $'K_2$ n'est pas employé, $(^2K_1)^n$ désigne le code $^2K_1$ normalisé.

21. $'K_0 = 101100$ (, *TN*), transfert normal d'un nombre avec normalisation bloquée.

22. $'K_0 = 101100$ et $'\varepsilon_{19} = 1$ (, *TNI*), transfert d'un nombre vers l'imprimante, $^2K_1$ est imprimé.

23. $'K_0 = 001100$ et $'\varepsilon_{26} = 1$ ou $'\varepsilon_{27} = 1$ ou $'\varepsilon_{28} = 1$ (*TNR*), transfert d'un nombre à partir des registres du tableau de commande.

Si $'\varepsilon_{26} = 1$, transfert du code à partir du registre de contrôle du basculeur.
Si $'\varepsilon_{27} = 1$, transfert du code du second registre alimenté par diodes.
Si $'\varepsilon_{28} = 1$, transfert du code du premier registre alimenté par diodes.

24. $'K_0 = 101100$ et $'\varepsilon_{26} = 1$ ou $'\varepsilon_{27} = 1$ ou $'\varepsilon_{28} = 1$ (, *TNR*), même chose que (*TNR*), mais avec normalisation bloquée.

25. $'K_0 = 001101$ (*TN* −), transfert d'un nombre avec changement de signe. Le nombre $^2K_1$ normalisé, pourvu du signe opposé est transféré à l'adresse de $'K_3$. $'K_2$ n'est pas utilisé

$$(-\,{}^2K_1)_n \Rightarrow {}'K_3\,.$$

26. $'K_0 = 101101$ (, *TN* −), même chose que (*TN* −), mais avec normalisation bloquée.

27. $'K_0 = 001110$ (| *TN* |), transfert du module d'un nombre. Le module du nombre normalisé $^2K_1$ est envoyé à l'adresse de $'K_3$

$$|\,{}^2K_1\,|^n \Rightarrow {}'K_3\,.$$

28. $'K_0 = 101110$ (, | *TN* |), (| *TN* |) avec normalisation bloquée.

29. $'K_0 = 001111$ (*TN*±), transfert d'un nombre avec changement de signe en fonction du signe d'un autre nombre. Le nombre transféré $^2K_1$ est normalisé, puis le signe de ce nombre est inversé si $^2K_2$ est négatif, sinon il est conservé. Le résultat est envoyé à l'adresse de $'K_3$.

30. $'K_0 = 101111$ (, *TN*±), même chose que précédemment, mais avec normalisation bloquée.

31. $'K_0 = 010000$ (↓), transfert de la caractéristique d'un nombre. La caractéristique du nombre $^2K_1$ est représentée sous forme d'un nombre normalisé ayant sa propre caractéristique et est transféré à l'adresse de $'K_3$. $'K_2$ n'est pas utilisé.

32. $'K_0 = 110000$ (↓), même chose que précédemment, mais avec normalisation bloquée.

33. $'K_0 = 010001$ ($\leftarrow$), décalage avec caractéristiques bloquées. Décalage du nombre $^2K_1$ d'un nombre de positions égal au nombre de $'K_2$ vers la gauche si sign $'K_2 = 0$, vers la droite si sign $'K_2 = 1$. Dans ce cas, $'\varepsilon_{22}$ (à la septième position du registre K_2) est pris comme position de signe (le code de la caractéristique n'est ni décalé, ni transféré) ; le résultat est envoyé à l'adresse de $'K_3$.

34. $'K_0 = 110001$ (, $\leftarrow$), décalage à toutes les positions. Même instruction que la précédente, mais avec décalage à toutes les positions.

35. $'K_0 = 011101$ (L), multiplication logique. Multiplication logique position par position des codes $^2K_1$ et $^2K_2$ avec envoi du résultat à l'adresse de $'K_3$.

36. $'K_0 = 010010$ (AI), addition des instructions. Addition des parties numériques des nombres $^2K_1$ et $^2K_2$ (signes compris) ; la caractéristique du second nombre n'est pas prise en considération. Le résultat (sans normalisation) est envoyé à l'adresse de $'K_3$.

37. $'K_0 = 110010$ (, AI), addition cyclique. Les codes $^2K_1$ et $^2K_2$, considérés comme un seul entier, sont additionnés (avec transfert à partir de la position du signe du nombre à la première position de la caractéristique et à partir de la position du signe de la caractéristique à la position de poids faible de la partie numérique du nombre). Le résultat est envoyé à l'adresse de $'K_3$.

38. $'K_0 = 010011$ ou 110011 (PE). Extraction de la partie entière d'un nombre. La partie entière du nombre $^2K_1$ est envoyée, avec son signe, à l'adresse de $'K_3$ (sous forme d'un nombre avec virgule fixe après la position du poids le plus faible) ; la partie décimale, ramenée à l'ordre zéro, est envoyée à l'adresse de $'K_2$.

III. Opérations de transfert de commande

Dans la machine B. E. S. M., en plus des opérations introduites au début, il existe les opérations suivantes qui maintiennent le régime de travail de la commande locale ou centrale.

1. $'K_0 = 010100$ ($<$), comparaison en tenant compte des signes :

a) $P\{\,^2K_1 < {}^2K_2\}$ $\quad 'K_3(1 - 'T) + 'C_2\,'T \Rightarrow C_2$; $\quad 'K_3\,'T + 'C_1(1 - 'T) \Rightarrow C_1$; $\quad 'C_1 + 'T \Rightarrow C_1$; $\quad 'C_2 + (1 - 'T) \Rightarrow C_2$;

b) $'('C_1\,'T + 'C_2(1 - 'T)) \Rightarrow K$.

Ainsi, si ${}^2K_1 < {}^2K_2$, le transfert de la commande se fait vers l'adresse de $'K_3$,
si ${}^2K_1 \geqslant {}^2K_2$, le transfert se fait vers l'instruction dont le numéro suit.

De façon analogue, on peut utiliser les deux opérations suivantes.

2. $'K_0 = 110100$ (, <), comparaison de l'égalité des codes.

$$a)\ P\{\ {}^2K_1 \neq {}^2K_2\ \}\quad \begin{matrix} 'K_3(1-'T) + 'C_2\,'T \Rightarrow C_2\ ; \\ 'K_3\,'T + 'C_1(1-'T) \Rightarrow C_1\ ; \\ 'C_1 + 'T \Rightarrow C_1\ ;\quad 'C_2 + (1-'T) \Rightarrow C_2\ ; \end{matrix}$$

$$b)\ '('C_1\,'T + 'C_2(1-'T)) \Rightarrow K\,.$$

3. $'K_0 = 010101$ (| < |), comparaison des modules.

$$a)\ P\{\ |\ {}^2K_1\ | < |\ {}^2K_2\ |\ \}\quad \begin{matrix} 'K_3(1-'T) + 'C_2\,'T \Rightarrow C_2\ ; \\ 'K_3\,'T + 'C_1(1-'T) \Rightarrow C_1\ ; \\ 'C_1 + 'T \Rightarrow C_1\ ;\quad 'C_2 + (1-'T) \Rightarrow C_2\ ; \end{matrix}$$

$$b)\ '('C_1\,'T + 'C_2(1-'T)) \Rightarrow K\,.$$

IV. Opérations d'appel aux organes périphériques

1 et **2.** $'K_0 = 010110$ (*IMa*) et $'K_0 = 010111$ (*IMb*), appel aux organes de mémoire externes (enregistrement magnétique). Ces opérations sont effectuées par deux instructions. Dans les adresses, on note le premier et le dernier numéro du groupe de codes de l'organe mémoire externe concernés par l'opération, et la première adresse de la cellule de l'organe mémoire interne à laquelle est liée l'opération. Le caractère de l'opération est défini de façon complémentaire par la première adresse de l'instruction *IMa* (enregistrement, lecture ou déroulement ; tambour ou bande ; numéro du tambour ou de la bande). On prévoit aussi l'appel aux bandes perforées, tambours ou bandes magnétiques.

3. $'K_0 = 011100$ (ARR. COND.) arrêt conditionnel (par interrupteur). Le programme s'arrête si l'interrupteur spécial T_1 est enclenché ($'T_1 = 1$), sinon le programme passe à l'instruction suivante, c'est-à-dire :

$$a)\ P\{\ 'T_1 = 1\ \}\ \text{arr.}\quad 'C_1 + 'T \Rightarrow C_1\ ;\quad 'C_2 + (1-'T) \Rightarrow C_2\ ;$$

$$b)\ '('C_1\,'T + 'C_2(1-'T)) \Rightarrow K.$$

4. $K_0 = 011111$ (*ARR.*) arrêt.

L'arrêt s'effectue au signal approprié.

II. STRELA

La C. A. N. Strela effectue des calculs avec des codes à 43 positions. Les nombres sont en virgule flottante ; on a prévu 36 positions binaires pour la mantisse (la première est réservée au signe de la mantisse) et 7 positions binaires pour la caractéristique (la première est réservée au signe de la caractéristique) (Fig. 23).

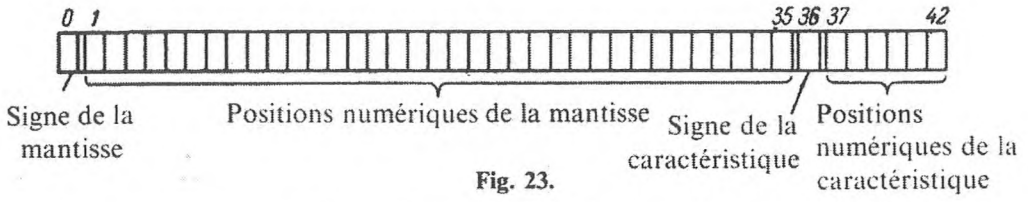


Fig. 23.

Nous appellerons registre de la mantisse les positions de 0 à 35 et registre de la caractéristique les positions de 36 à 42, et nous les désignerons par un numéro de cellule suivi de l'indice *m* ou *C*.

Pour la mise en mémoire de nombres en décimal codé binaire on a besoin de 37 positions pour la mantisse (dont une pour le signe) et de 6 pour la caractéristique (dont une pour le signe). Les 36 positions numériques de la mantisse sont divisées en 9 quadruplets, un pour chaque signe en décimal.

Le système d'instructions employé est celui à 3 adresses avec l'ordre naturel d'exécution. Au moment de l'enregistrement de l'instruction en mémoire, chaque adresse occupe 12 positions, le code opération en occupe 6 et une position est réservée au contrôle ; la figure 24 montre comment se répartissent les positions de la cellule dans laquelle est écrite l'instruction.

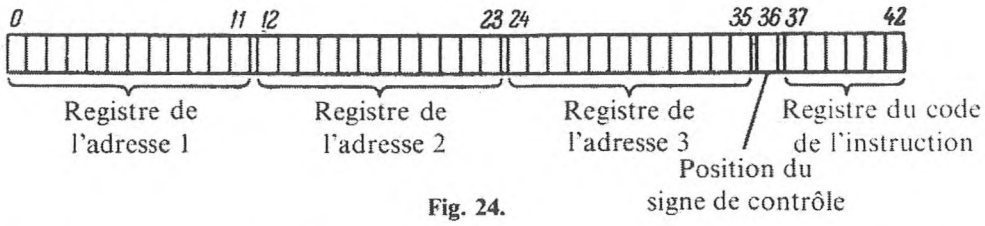


Fig. 24.

La Strela effectue 2 000 à 3 000 opérations à la seconde. La mémoire interne est composée de 2 048 cellules ; elle est constituée de tubes cathodiques. En plus de la mémoire opération, la machine possède une mémoire permanente. Il faut introduire ici ce qu'on appelle l'organe de distribution des constantes (O. D. C.), dans lequel on garde les constantes fréquemment employées ; on affecte les numéros de 7 400 à 7 777 à ces cellules. Dans la mémoire permanente, on entre un accumulateur de sous-programmes standards (A. S. S.) dans lequel sont stockés les sous-programmes qui calculent les fonctions élémentaires $\left(\frac{1}{x}, e^x, \sin x, \ln x, \text{etc.}\right)$.

La mémoire externe est composée de deux bandes ferro-magnétiques. Ces bandes sont divisées en zones : une bande peut contenir jusqu'à 511 zones. Dans chaque zone on peut inscrire 2 048 nombres. Les zones des bandes magnétiques ont respectivement les numéros octaux de 4 001 à 4 777 et de 5 001 à 5 777. La lecture des codes de chaque zone est possible à n'importe quel groupe qui commence par le premier code.

L'entrée est matérialisée par les cartes perforées, il en est de même pour la sortie. Un organe spécial permet de transférer l'information des cartes perforées à l'imprimante. Chaque code (nombre ou instruction) est perforé sous forme d'une ligne ; une carte peut contenir 12 codes.

A la perforation des instructions, chaque chiffre en octal est converti en un triplet de chiffres binaires, le signe de contrôle en un chiffre binaire. A la perforation des nombres décimaux, chaque chiffre en décimal est converti en un quadruplet de chiffres binaires, le signe en un nombre binaire. Ces conversions sont automatiques.

Pour décrire les opérations, nous aurons besoin des notations suivantes :

1) c, compteur des instructions,
2) K, registre des instructions,
3) G, registre des instructions de l'opération de groupe.
4) A, registre de modification d'adresses.
5) ω, basculeur du symbole qui sert au transfert conditionnel.
6) T, interrupteur du régime de contrôle.

Nous diviserons le registre des instructions K, conformément à la répartition des instructions (voir fig. 24) en groupes de positions (du registre) (voir le tableau 11).

Tableau 11

Groupe de positions du registre des instructions	Notation et dénomination des groupes de positions (des registres)
0-11	k_1, registre de l'adresse 1
12-23	k_2, — 2
24-35	k_3, — 3
36	α, position du signe de contrôle
37-42	k_0, registre du code instruction

Le registre des instructions de l'opération de groupe G (et de la cellule) peut, au besoin, être divisé de façon analogue. On les désigne par « G » (ou par le numéro de la cellule) avec l'indice correspondant : (0, 1, 2, 3).

Les registres indiqués peuvent contenir des codes à base 8

$$'k_1,\ 'k_2,\ 'k_3 = 0000\ ;\ 0001\ ;\ \ldots\ ;\ 7777\ ;$$

$$'\alpha = 0\ ;\ 1\ ;$$

$$'k_0 = 00\ ;\ 01\ ;\ \ldots\ ;\ 77\ .$$

Toutefois $'k_1$, $'k_2$, $'k_3$, dans tous les cas (sauf convention spéciale) sont inférieurs à 4 000 (2048 en décimal), nombre qui correspond à l'organe mémoire interne.

Dans la machine, on a prévu un régime dit d'arrêt de contrôle : pour s'y transférer on branche un interrupteur T ($'T = 1$). Dans ce cas après l'exécution de chaque instruction dans laquelle $'\alpha = 1$, la machine s'arrête. Autrement dit, après l'exécution de chaque instruction est exécutée en plus l'opération :

$$P\{\ '\alpha \times 'T = 1\ \}\quad ARR\ .$$

En fonction du code $'k_0$, les opérations suivantes sont effectuées.

I. Opérations arithmétiques

1. $'k_0 = 01$, addition. La machine effectue les opérations suivantes :

a) $^2k_1 + {}^2k_2 \Rightarrow 'k_3$;

b) $P\{\ ^2k_3 \leqslant -0\ \}$ $\begin{matrix} 1 \Rightarrow \omega\ ; \\ 0 \Rightarrow \omega\ ; \end{matrix}$

c) $'c + 1 \Rightarrow c$;

d) $^2c \Rightarrow K$.

a) désigne l'addition des codes $^2k_1$ et $^2k_2$ avec normalisation du résultat et son envoi à l'adresse de $'k_3$; *b*) l'élaboration du signal ω (utilisé dans les instructions de transfert de commande conditionnel, voir ci-dessous) ; *c*) et *d*) le passage à l'instruction suivante (ils sont identiques pour les opérations arithmétiques). En outre, il existe un arrêt dit de dépassement de capacité lorsque la caractéristique du résultat est supérieure à 77 ; si elle est inférieure au minimum admis -77, on prend zéro comme résultat. C'est pour cela que *a*) doit être, ici comme pour les autres opérations arithmétiques, écrit d'une façon plus explicite :

a) $P\{\ (^2k_1 + {}^2k_2)_c > 77\ \}\quad ARR$;

$P\{\ (^2k_1 + {}^2k_2)_c > -77\ \}$ $\begin{matrix} ^2k_1 + {}^2k_2 \Rightarrow 'k_3\ ; \\ 0 \Rightarrow k_3. \end{matrix}$

2. $'k_0 = 03$, soustraction.

3. $'k_0 = 04$, soustraction des modules.

Ces opérations s'effectuent comme l'addition (point *a*)), mais avec signes appropriés.

4. $'k_0 = 05$, multiplication :

a) $^2k_1 \times {}^2k_2 \Rightarrow {}'k_2$;

b) $P\{\,|\,{}^2k_3\,|\geqslant 1\,\}$ $\begin{matrix} 1 \Rightarrow \omega\,; \\ 0 \Rightarrow \omega\,; \end{matrix}$

c) et *d*).

5. $'k_0 = 06$, addition des caractéristiques et

6. $'k_0 = 07$, soustraction des caractéristiques :

a) $'('k_1)_m \Rightarrow ('k_3)_m$;
$'('k_1)_c \pm {}'('k_2)_c \Rightarrow ('k_3)_c$;

b) $P\{\,('k_3)_c \geqslant 1\,\}$ $\begin{matrix} 1 \Rightarrow \omega\,; \\ 0 \Rightarrow \omega\,; \end{matrix}$

c) et *d*).

7. $'k_0 = 10$, transfert d'un nombre accompagné du signe d'un autre nombre :

a) $|\,{}^2k_1\,| \vee \operatorname{sign} {}^2k_2 \Rightarrow {}'k_3$;

b) $P\{\,'('k_3)_c \geqslant 1\,\}$ $\begin{matrix} 1 \Rightarrow \omega\,; \\ 0 \Rightarrow \omega\,; \end{matrix}$

c) et *d*).

8. $'k_0 = 12$, addition des nombres sans arrondi :

a) $^2k_1 + {}^2k_2 \Rightarrow {}'k_3$;

b) $P\{\,{}^2k_3 = 0\,\}$ $\begin{matrix} 1 \Rightarrow \omega\,; \\ 0 \Rightarrow \omega\,; \end{matrix}$

c) et *d*).

9. $'k_0 = 17$, addition de contrôle des codes $^2k_1$ et $^2k_2$ considérés comme

seuls codes binaires, avec transfert de la position de poids fort vers celle de poids faible :

a) ${}^2k_1(C+)\,{}^2k_2 \Rightarrow {}'k_3$;
b) $0 \Rightarrow \omega$;
c) et *d)*.

10. ${}'k_0 = 02$, addition spéciale (pour les instructions) et

11. ${}'k_0 = 15$, soustraction spéciale (pour les instructions) :

a) ${}'({}'k_1)_i \pm {}'({}'k_2)_i \Rightarrow ({}'k_3)_i$; où $i = 1, 2, 3$;
${}'({}'k_1)_0 \Rightarrow ({}'k_3)_0$;
b) $0 \Rightarrow \omega$;
c) et *d)*.

II. Opérations logiques

1. ${}'k_0 = 11$, multiplication logique position par position :

a) $({}^2k_1) \wedge ({}^2k_2) \Rightarrow {}'k_3$, position par position ;

b) $P\{{}^2k_3 = 0\}$ $\begin{matrix} 1 \Rightarrow \omega ; \\ 0 \Rightarrow \omega ; \end{matrix}$

c) et *d)*.

2. ${}'k_0 = 13$, addition position par position :

a) $({}^2k_1) \vee ({}^2k_2) \Rightarrow {}'k_3$, position par position ;

b) $P\{{}^2k_3 = 0\}$ $\begin{matrix} 1 \Rightarrow \omega ; \\ 0 \Rightarrow \omega ; \end{matrix}$

c) et *d)*.

3. $k'_0 = 14$, décalage : 2 possibilités pour cette opération :
Si ${}'k_2 < 4\,000$, c'est-à-dire s'il est l'adresse d'une cellule opération, le code ${}^2k_1$ se décale d'un nombre de positions égal à la caractéristique du nombre ${}^2k_2$, vers la gauche si le signe de la caractéristique est $+$, vers la droite s'il est $-$.
Si ${}'k_2 = 4\,000 + l$ ou ${}'k_2 = 5\,000 + l$, le décalage se fait de l positions vers la gauche ; de même, si ${}'k_2 = 4\,100 + l$ ou ${}'k_2 = 5\,100 + l$ le décalage se fait vers la droite de l positions ($l < 77$).
En représentant par $\beta \leftrightarrow \alpha$ le décalage du code β de α positions (vers la gauche si $\alpha > 0$, vers la droite si $\alpha < 0$) nous noterons l'opération sous forme d'adresses :

a) 1) $P\{\ 'k_2 < 4\,000\ \}$ $\quad {}^2k_1 \leftrightarrow {}'('k_2) \Rightarrow {}'k_3$;

2) $P\{\ 'k_2 < 4\,100\ \}$ $\quad {}^2k_1 \leftrightarrow ('k_2 - 4\,000) \Rightarrow {}'k_3$;

3) $P\{\ 'k_2 < 5\,000\ \}$ $\quad {}^2k_1 \leftrightarrow (4\,100 - {}'k_2) \Rightarrow {}'k_3$;

4) $P\{\ 'k_2 < 5\,100\ \}$ $\quad {}^2k_1 \leftrightarrow ('k_2 - 5\,000) \Rightarrow {}'k_3$;

$\quad {}^2k_1 \leftrightarrow (5\,100 - {}'k_2) \Rightarrow {}'k_3$;

b) $P\{\ {}^2k_3 = 0\ \}$ $\quad 1 \Rightarrow \omega$;

$\quad 0 \Rightarrow \omega$;

c) et *d*).

4. $'k_0 = 16$, opération de « position par position non-équivalence » (cf. chapitre II) :

a) $({}^2k_1) \cong ({}^2k_2) \Rightarrow {}'k_3$ (position par position) :

b) $P\{\ {}^2k_3 \neq 0\ \}$ $\quad 1 \Rightarrow \omega$;

$\quad 0 \Rightarrow \omega$;

c) et *d*).

5. $'k_0 = 26$, comparaison et arrêt en cas de non-coïncidence : *a*), *b*) et *d*) sont les mêmes que pour 4.

c) $P\{\ \omega = 1\ \}$ $\quad ARR$;

$\quad 'c + 1 \Rightarrow c$.

III. Instructions de transfert conditionnel et instructions préliminaires d'opérations de groupe

1. $'k_0 = 20$, transfert conditionnel de type I :

a) $P\{\ '\omega = 0\ \}$ $\quad {}'k_1 \Rightarrow c$;

$\quad {}'k_2 \Rightarrow c$;

b) $0 \Rightarrow {}'k_3$;

c) ${}^2c \Rightarrow K$.

2. $'k_0 = 27$, transfert conditionnel de type II :

$$a)\ P\{\ '\omega = 0\ \}\quad \begin{matrix} 'k_1 \Rightarrow c\ ; \\ 'k_2 \Rightarrow c\ ; \end{matrix}$$

$$b)\ \begin{matrix} 'c + 1 \Rightarrow ('k_3)_1\ ; \\ 'c + 1 \Rightarrow ('k_3)_2\ ; \\ 'k_3 \Rightarrow ('k_3)_3\ ; \\ 20 \Rightarrow ('k_3)_0\ . \end{matrix}$$

a) est le transfert conditionnel déterminé par la présence de ω ; *b*) est l'envoi de l'instruction de transfert inconditionnel à l'adresse $('k_3)$.

$$c)\qquad {}^2c \Rightarrow K\ .$$

3. $'k_0 = 30, 31, 32, 33, 34, 35, 36, 37$, instructions préliminaires d'opérations de groupe.

Les opérations de groupe sont composées de deux instructions : la préliminaire et l'effective (le code de cette dernière est un code arbitraire d'opération arithmétique ou logique).

Dans l'adresse 2 de l'instruction préliminaire, on note un nombre inférieur d'une unité au nombre de fois que l'instruction effective doit être exécutée.

Le code opération de l'instruction préliminaire contient de l'information sur la règle de modification des adresses de l'instruction effective. De plus, dans l'adresse 3 de cette instruction, on indique l'adresse où on envoie 0.

Désignons respectivement par ε_1, ε_2, ε_3, les 40-ième, 41-ième, 42-ième positions du registre K (c'est-à-dire les trois positions de poids faible du registre du code opération k_0). On peut décrire ainsi les opérations de groupe :

$a)\ 0 \Rightarrow A\ ;\quad 0 \Rightarrow 'k_3\ ;$
$b)\ '('c + 1) \Rightarrow G\ ;$
$c)\ '('G_1 + '\varepsilon_1 . 'A)\ \theta\ '('G_2 + '\varepsilon_2\ 'A) \Rightarrow 'G_3 + '\varepsilon_3\ 'A\ ;$
$d)\ 'A + 1 \Rightarrow A\ ;$
$e)\ P\{\ 'A \leqslant 'k_2\ \}\ b)\ ;$
$f)\ 'c + 2 \Rightarrow c\ ;$
$g)\ {}^2c \Rightarrow k\ .$

Par θ on désigne n'importe quelle opération arithmétique ou logique. *a*) est la remise à zéro du registre de modification A ainsi que celle d'une cellule de $'k_3$ (si cette dernière remise à zéro est inutile, on met zéro dans l'adresse 3). *b*) est l'envoi de l'instruction effective dans le registre des opérations de groupe G. *c*) est l'exécution de l'instruction contenue dans le registre G avec modification d'adresses conformément aux trois dernières positions du code opération de l'instruction préliminaire ; l'adresse G_1, pour laquelle $\varepsilon_i = 1$

($i = 1, 2, 3$), est augmentée du contenu du registre A (puisqu'à la première exécution de c) le registre A contient zéro, la première fois que l'instruction est exécutée, elle le sera sous la forme sous laquelle elle a été codée). d) est l'augmentation d'une unité du contenu du registre A. e) est le test de la fin de l'opération de groupe. Finalement l'opération c) sera répétée $('k_2 + 1)$ fois. f) et g) sont le transfert vers l'instruction qui suit l'instruction effective.

IV. Instructions d'appel aux organes externes

1. $'k_0 = 25$, amène la zone $'k_1$ de la bande magnétique sous la tête de lecture.

2. $'k_0 = 43$, transfert des codes de la bande magnétique vers les cellules de la mémoire. Le code $('k_2 + 1)$ de la zone $'k_1$ de la bande magnétique est transféré vers les cellules $'k_3, 'k_3 + 1, \ldots, 'k_3 + 'k_2$. Par convention nous noterons :

$$'[('k_1)\, BM] \Rightarrow \begin{pmatrix} 'k_3 \\ 'k_3 + 'k_2 \end{pmatrix}.$$

3. $'k_0 = 46$, transfert des codes de la mémoire interne vers la bande magnétique :

$$\text{le code } ('k_2 + 1) \qquad 'k_1, 'k_1 + 1, \ldots, 'k_1 + 'k_2$$

est transféré de la mémoire vers la zone $'k_3$ de la bande magnétique.

Nous noterons par convention :

$$'\begin{pmatrix} 'k_1 \\ 'k_1 + 'k_2 \end{pmatrix} \Rightarrow ('k_3)\, BM\,.$$

$('k_3)\, BM$ désigne la zone $'k_3$ de la bande magnétique.

4. $'k_0 = 41$, transfert des codes des cartes perforées vers la mémoire interne. Le code $('k_2 + 1)$ est transféré des cartes vers les cellules :

$$'k_3, 'k_3 + 1, \ldots, 'k_3 + 'k_2\,;$$

$$'BM \Rightarrow \begin{pmatrix} 'k_3 \\ 'k_3 + 'k_2 \end{pmatrix}.$$

5. $'k_0 = 44$, transfert des codes des cellules de l'organe mémoire vers les cartes. Le code $('k_2 + 1)$ est envoyé des cellules $'k_1, 'k_1 + 1, \ldots, 'k_1 + 'k_2$ vers les cartes :

$$\begin{pmatrix} 'k_1 \\ 'k_1 + 'k_2 \end{pmatrix} \Rightarrow CP .$$

6. $'k_0 = 45$, transfert de codes de certaines cellules dans d'autres. Le code $('k_2 + 1)$ est envoyé des cellules $'k_1$, $'k_1 + 1$, ... $'k_1 + 'k_2$ vers les cellules $'k_3$, $'k_3 + 1$, ..., $'k_3 + 'k_2$:

$$'\begin{pmatrix} 'k_1 \\ 'k_1 + 'k_2 \end{pmatrix} \Rightarrow \begin{pmatrix} 'k_3 \\ 'k_3 + 'k_2 \end{pmatrix}.$$

7. $'k_0 = 60$, transfert de groupe avec contrôle. Variantes possibles de cette opération :

Si $$'k_1 = 0000, 0001 \leqslant 'k_3 \leqslant 3\,777 ,$$

transfert des cartes perforées vers la mémoire.

Si $$0001 \leqslant 'k_1 \leqslant 3\,777, 'k_3 = 0000 ,$$

transfert de la mémoire vers les cartes.

Si $$0001 \leqslant 'k_3 \leqslant 3\,777 ,$$

$$4\,001 \leqslant 'k_1 \leqslant 4\,777 \text{ ou } 5\,001 \leqslant 'k_1 \leqslant 5\,777 ,$$

transfert de la bande magnétique vers la mémoire.

Si $$0001 \leqslant 'k_1 \leqslant 3\,777 ,$$

$$4\,001 \leqslant 'k_3 \leqslant 4\,777 \text{ ou } 5\,001 \leqslant 'k_3 \leqslant 5\,777 ,$$

transfert de la mémoire vers la bande magnétique.

Si $$0001 \leqslant 'k_1 \leqslant 3\,777, 0001 \leqslant 'k_3 \leqslant 3\,777 ,$$

transfert d'une série de cellules vers d'autres. Le transfert est accompagné d'une addition cyclique ($'k_0 = 17$) des codes avant et après le transfert de groupe avec contrôle. Les sommes obtenues sont comparées et, en cas de non-coïncidence, la machine s'arrête.

8. Instruction d'arrêt, $'k_0 = 40$. Les nombres $^2k_1$ et $^2k_2$ apparaissent sur le pupitre.

V. Programmes standards de la mémoire permanente

Les instructions décrites ci-dessous sont des opérations de groupe qui appellent les sous-programmes standards qui sont dans la mémoire perma-

nente. Après l'exécution de ces sous-programmes, on passe à l'instruction suivante du programme principal. La dernière ne doit pas être une instruction de transfert conditionnel (codes 20 et 27), car, dans ces opérations, le signe ω n'est pas recherché.

1. $'k_0 = 62$, calcul de l'inverse d'une grandeur :

$$\frac{1}{'('k_1 + i)} \Rightarrow 'k_3 + i, \quad \text{où} \quad i = 0, 1, ..., 'k_2 .$$

2. $'k_0 = 63$, calcul d'une racine carrée :

$$\sqrt{'('k_1 + i)} \Rightarrow 'k_3 + i, \quad i = 0, 1, ..., 'k_2 .$$

De même, les opérations suivantes sont exécutées :

3. $'k_0 = 64$, calcul de la fonction exponentielle.

4. $'k_0 = 66$, calcul du logarithme.

5. $'k_0 = 67$, calcul du sinus.

6. $'k_0 = 73$, calcul de la fonction arc tg.

7. $'k_0 = 74$, calcul de la fonction arc sin.

Pour chacune de ces opérations, il peut y avoir arrêt s'il y a dépassement de capacité de la cellule $'k_3 + i$ ou si la fonction ne peut pas s'appliquer au code $'k_1 + i$ dans le domaine des nombres réels.

8. $'k_0 = 70$, conversion des codes de binaire en décimal codé binaire :

$$'('k_1 + i)_{\text{déc.}} \Rightarrow 'k_3 + i , \quad \text{où} \quad i = 0, 1, ..., 'k_2 .$$

9. $'k_0 = 72$, opération inverse :

$$'('k_1 + i)_{\text{bin.}} \Rightarrow 'k_3 + i , \quad i = 0, 1, ..., 'k_2 .$$

De même, on effectue :

$$'c + 1 \Rightarrow c ; \qquad "c \Rightarrow K .$$

III. OURAL

La calculatrice Oural opère à partir de codes longs à 36 positions ou courts à 18 positions. On utilise pour les instructions le système à une adresse ; elles sont codées dans les cellules courtes : 5 positions pour le code opération, 12 pour l'adresse, une pour la marque de l'instruction de modification d'instruction.

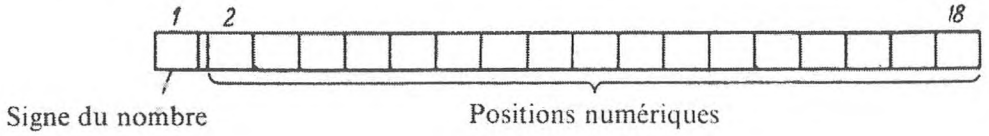


Fig. 25.

Les nombres sont en virgule fixe, la première position est celle du signe. Pour le stockage des nombres dans les cellules courtes les positions sont distribuées comme l'indique la figure 25. La répartition des positions pour le stockage des nombres dans ce qu'on appelle les cellules longues (pleines) est représentée par les figures 26 et 27 pour le décimal codé binaire.

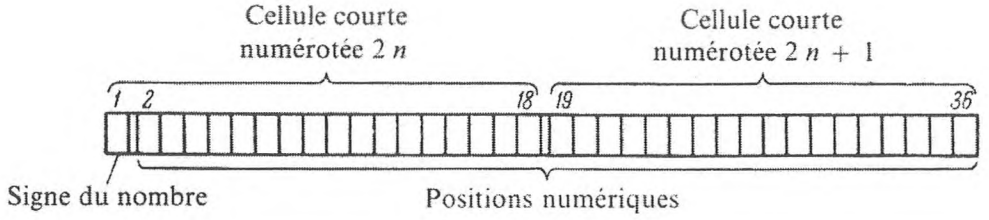


Fig. 26.

Ces machines sont rapides : 100 opérations à la seconde. La mémoire interne (sur tambour magnétique) contient 2 048 cellules courtes ou 1 024 longues. Les cellules longues n'ont que des numéros d'adresses pairs. La mémoire externe est composée de bandes magnétiques. L'entrée se fait par bandes perforées, la sortie par bandes perforées ou imprimées.

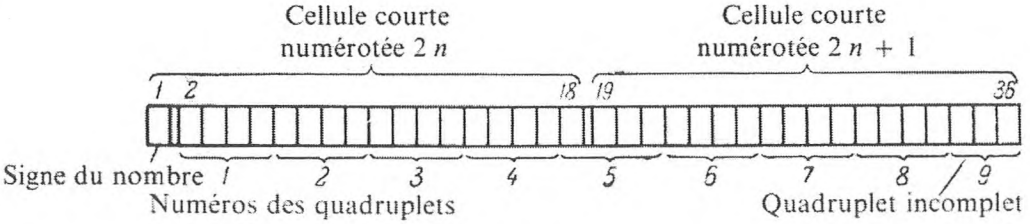


Fig. 27.

Les registres et les manipulateurs spéciaux à l'Oural sont :

1) S, totalisateur.
2) r, registre de l'organe arithmétique.
3) C, compteur des instructions.
4) K, registre des instructions.
5) ω, basculeur du symbole de transfert conditionnel de la commande.
6) Cy, registre des cycles.
7) K_i ($i = 1, ..., 7$), commutateurs.
8) T_1, interrupteur 1.
9) T_2, interrupteur 2.

Ayant adopté les désignations indiquées, passons à la description des opérations élémentaires effectuées par l'Oural.

Un cycle de travail de la machine Oural se compose de :

1) l'exécution de l'instruction K, c'est-à-dire l'instruction dont le code se trouve à un moment donné dans le registre des instructions ;
2) l'envoi du code de l'instruction qui suit dans le registre des instructions.

Pour décrire les opérations élémentaires sur Oural il faut isoler des groupes de positions (registres) dans le registre des instructions (cf. tableau 12).

Tableau 12

Positions du registre des instructions, numérotées de gauche à droite	Notation et dénomination des groupes de positions (registres)
1	ε_0, position du symbole de modification d'instruction
2-6	k_0, registre du code opération
7	ε, position du symbole de cellule longue
8-18	k, registre d'adresse d'instruction

La figure 28 montre la division du registre des instructions (et de la cellule qui contient l'instruction) en groupes de positions.

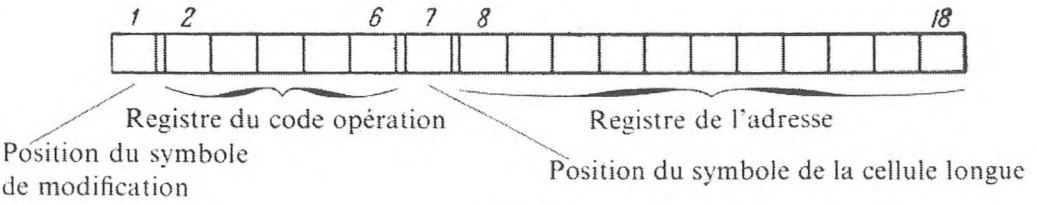


Fig. 28.

Les ensembles des codes (en octal) susceptibles d'être conservés dans les registres sont les suivants :

$$'\varepsilon_0, '\varepsilon = 0 ; 1 ;$$
$$'k_0 = 00, 01, ..., 37 ;$$
$$'k = 0000, 0001, ..., 3777 .$$

I. Opérations arithmétiques

1. $'k_0 = 01$, addition. Cette instruction effectue les opérations suivantes :

a) $'S + '('k - '\varepsilon_0\ 'Cy) \Rightarrow S$;
sign $'S \vee | '(C\ 'k - '\varepsilon_0\ 'Cy) | \Rightarrow r$;

b) $P\{\text{sign } 'S = 1\}$ $\begin{matrix} 1 \Rightarrow \omega ; \\ 0 \Rightarrow \omega ; \end{matrix}$

c) $'C + 1 \Rightarrow C$;

d) $^2C \Rightarrow K$.

Si $'\varepsilon_0 = 0$, c'est-à-dire s'il n'y a pas de symbole de modification d'instruction, l'addition $'S + {}^2k \Rightarrow S$ est exécutée.

Si $'\varepsilon_0 = 1$, l'adresse est modifiée, c'est-à-dire qu'elle est diminuée du nombre contenu dans le registre des cycles.

De plus, si $'\varepsilon = 0$, on prend le contenu de la cellule courte d'adresse $'k$; si $'\varepsilon = 1$, celui de la cellule longue. Dans ce dernier cas, $'k$ doit être pair.

b) signifie l'envoi dans le basculeur du symbole, 1 ou 0 suivant le signe du nombre qui se trouve dans le totalisateur, symbole de transfert conditionnel de la commande ω.

c) et *d)* correspondent à la préparation de l'instruction dont le numéro suit. Les opérations *b)*, *c)* et *d)* accompagnent toutes les autres instructions de ce groupe. On peut dire la même chose pour la position ε. Nous n'en parlerons plus dans les descriptions suivantes.

2. $'k_0 = 02$, transfert vers le totalisateur :

a) $'('k - '\varepsilon_0\ 'Cy) \Rightarrow S$; $'('k - '\varepsilon_0\ 'Cy) \Rightarrow r$;
b), *c)* et *d)*, comme pour $'k_0 = 01$.

3. $'k_0 = 03$, soustraction :

a) $'S - '('k - '\varepsilon_0\ 'Cy) \Rightarrow S$;
sign $'S \vee | '('k - '\varepsilon_0\ 'Cy) | \Rightarrow r$.

4. $'k_0 = 04$ (même chose que pour $'k = 03$, la seule différence est que l'opération se fait en valeur absolue) :

$a)$ $|\,'S\,| - |\,'(k - {}'\varepsilon_0\,'Cy)\,| \Rightarrow S$;
sign $'S \vee |\,'('k - {}'\varepsilon_0\,'Cy)\,| \Rightarrow r$.

5. $'k_0 = 05$, multiplication avec accumulation dans le totalisateur :

$a)$ $'r \times '('k - {}'\varepsilon_0\,'Cy) + 'S \Rightarrow S$; $0 \Rightarrow r$;

le contenu de la cellule est multiplié par le contenu du registre du totalisateur et le résultat est ajouté au contenu du totalisateur.

6. $'k_0 = 06$, multiplication :

$a)$ $'S \times '('k - {}'\varepsilon_0\,'Cy) \Rightarrow S$;
$0 \Rightarrow r$.

7. $'k_0 = 07$, division :

$a)$ $'S : '('k - {}'\varepsilon_0\,'Cy) \Rightarrow S$; $0 \Rightarrow r$.

Dans la machine, en plus des registres énumérés, il y a un registre de dépassement de capacité du réseau φ et les interrupteurs φ_1 et φ_2 qui lui sont associés.

Au cours des opérations arithmétiques 01, 03, 04, 05, 06, 07,

si $|\,'S\,| < 1$, on envoie 1 dans le basculeur φ :

$$1 \Rightarrow \varphi\ ;$$

si $('S) < 1$, on envoie 0 dans φ :

$$0 \Rightarrow \varphi\ .$$

De plus, si $'\varphi = 1$ et

$a)$ $'\varphi_1 = 1$
$'C + 1 \Rightarrow C$ est effectué, ce qui signifie le passage à l'instruction suivante ;

si $b)$ $'\varphi_1 = 0$, $'\varphi_2 = 1$, la machine s'arrête :

si $c)$ $'\varphi_1 = 0$, $'\varphi_2 = 0$ c'est $'C + 2 \Rightarrow C$ qui est effectuée, c'est-à-dire que l'on saute une instruction.

8. $'k_0 = 26$, opération d'addition cyclique du contenu du totalisateur avec le contenu de la cellule $'k - {}'\varepsilon_0\,'Cy$ (avec transfert de la position de poids fort vers celle de poids faible). Le rôle de ε_0 et ε est le même que dans les autres opérations. $b)$, $c)$ et $d)$ s'accomplissent de la même façon.

II. Opérations logiques

9. $'k_0 = 10$, attribution du signe d'un nombre au contenu du totalisateur.

a) $|\,'S\,| \vee \text{sign}\,'('k - '\varepsilon_0\,'Cy) \Rightarrow S\,(\Rightarrow r)$;

b) $P\{\,\text{sign}\,'S = 1\,\}\ \begin{matrix} 1 \Rightarrow \omega\,; \\ 0 \Rightarrow \omega\,; \end{matrix}$

c) $'C + 1 = C$;

d) $^2C \Rightarrow K$.

10. $'k_0 = 11$, décalage du contenu du registre r du totalisateur (position du signe y compris) d'un nombre de positions correspondant au module du nombre écrit dans les positions 13 à 18 du totalisateur S :

a) vers la gauche, si sign $'S = 0$, et vers la droite si sign $'S = 1$. Le résultat du décalage est placé dans le totalisateur : $0 \Rightarrow r$:

b) $P\{\,'S = 0\,\}\ \begin{matrix} 1 \Rightarrow \omega\,; \\ 0 \Rightarrow \omega\,; \end{matrix}$

c) $'C + 1 \Rightarrow C$;

d) $^2C \Rightarrow K$.

L'opération est indépendante de $'\varepsilon_0$, $'\varepsilon_1$ et $'k$.

11. $'k_0 = 12$, opération de multiplication logique position par position.

a) $'S \wedge '('k - '\varepsilon_0\,'Cy) \Rightarrow S\,(\Rightarrow r)$, dans ce cas $'k$ est pair.

b) $P\{\,'S = 0\,\}\ \begin{matrix} 1 \Rightarrow \omega\,; \\ 0 \Rightarrow \omega\,; \end{matrix}$

c) $'C + 1 \Rightarrow C$;

d) $^2C \Rightarrow K$.

12. $'k_0 = 13$, même chose que pour $'k_0 = 12$. mais l'opération accomplie est une addition logique position par position.

13. $'k_0 = 14$, opération logique de « non-équivalence » $\cong$ position par position ; le tableau 13 montre les positions du résultat.

Tableau 13

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

a) $['S \cong '('k - '\varepsilon_0\ 'Cy)] \Rightarrow S\ (\Rightarrow r)$;

b) $P\{\ 'S = 0\ \}$ $\begin{matrix} 0 \Rightarrow \omega\ ; \\ 1 \Rightarrow \omega\ ; \end{matrix}$

c) $'C + 1 \Rightarrow C$;

d) $^2C \Rightarrow K$.

14. $'k_0 = 15$, normalisation.

a) Le contenu du totalisateur est décalé vers la gauche d'un nombre de positions égal au nombre de 0 contenus entre la virgule et le premier chiffre significatif, si $|\ 'S\ | < \frac{1}{2}$, et vers la droite d'une position si $|\ 'S\ | > 1$. Après le décalage, le nombre est inscrit en $'k - '\varepsilon_0\ 'Cy$, et on met dans le totalisateur la caractéristique du nombre fixée par le décalage, dans le premier cas ce nombre est négatif, dans le second, il est positif. Le signe de la caractéristique est mis en position signe du totalisateur, la 19-ième position du totalisateur étant celle de plus faible poids.

b) $P\{\ '('k - '\varepsilon_0\ 'Cy) = 0\ \}$ $\begin{matrix} 1 \Rightarrow \omega\ ; \\ 0 \Rightarrow \omega\ ; \end{matrix}$

c) $'C + 1 \Rightarrow C$;

d) $^2C \Rightarrow K$.

15. $'k = 16$, envoi du contenu du totalisateur dans l'adresse :

a) $'S \Rightarrow 'k - '\varepsilon_0\ 'Cy$;

b) $P\{\ \text{sign}\ 'S = 1\ \}$ $\begin{matrix} 1 \Rightarrow \omega\ ; \\ 0 \Rightarrow \omega\ ; \end{matrix}$

c) $'C + 1 \Rightarrow C$;

d) $^2C \Rightarrow K$.

16. $'k_0 = 17$, envoi dans le registre du totalisateur :

a) $'('k - '\varepsilon_0\ 'Cy) \Rightarrow r$;

b) $P\{\ 'r = 0\ \}$ $\begin{matrix} 1 \Rightarrow \omega\ ; \\ 0 \Rightarrow \omega\ ; \end{matrix}$

c) $'C + 1 \Rightarrow C$;

d) $^2C \Rightarrow K$.

17. $'k_0 = 20$, envoi dans le totalisateur :

a) $'k - {}'\varepsilon_0\,'Cy \Rightarrow S\ (\Rightarrow r)$.

Dans le totalisateur (de la 7-ième position à la 18-ième) on porte le nombre $'k - {}'\varepsilon_0\,'Cy$ et non le contenu de cette adresse comme dans l'opération 02. On considère comme signe du nombre $'k - {}'\varepsilon_0\,'Cy$ le contenu ε, et $'\varepsilon$ est envoyé dans la position signe du totalisateur.

b) $P\{\text{ sign } 'S = 1\}$ $\begin{matrix} 1 \Rightarrow \omega\,; \\ 0 \Rightarrow \omega\,; \end{matrix}$

c) $'C + 1 \Rightarrow C$;

d) ${}^2C \Rightarrow K$.

III. Opérations de transfert de commande

Les opérations de transfert de commande ne changent pas le contenu des adresses de la machine et ne concernent que le contenu de quelques registres spéciaux.

18. $'k_0 = 21$, transfert conditionnel de la commande :

a) $P\{'\omega = 1\}$ $\begin{matrix} 'k - {}'\varepsilon_0\,'Cy \Rightarrow C\,; \\ 'C + 1 \Rightarrow C\,; \end{matrix}$

b) ${}^2C \Rightarrow K$.

Si dans le basculeur ω, il y a 1, le transfert se fait vers l'instruction qui se trouve à l'adresse $'k - {}'\varepsilon_0\,'Cy$, sinon vers l'instruction dont le numéro suit.

19. $'k_0 = 22$, transfert inconditionnel de la commande :

a) $'k - {}'\varepsilon_0\,'Cy \Rightarrow C$;
b) ${}^2C \Rightarrow K$.

20. $'k_0 = 23$, opération de transfert de commande par commutateur ; si le commutateur $'k$ est fermé, il laisse passer une instruction, sinon, le transfert se fait à l'instruction dont le numéro suit, c'est-à-dire :

a) $P\{'K_k = 1\}$ $\begin{matrix} 'C + 2 \Rightarrow C\,; \\ 'C + 1 \Rightarrow C\,; \end{matrix}$

b) ${}^2C \Rightarrow K$.

21. $'k_0 = 25$, début d'une opération de groupe. Cette opération prépare le contenu du registre des cycles Cy :

a) $'k \Rightarrow Cy$;
b) $'C + 1 \Rightarrow C$;
c) $^2C \Rightarrow K$.

22. $'k_0 = 24$, fin d'opération de groupes.

a) $P\{\, 'Cy \neq 0 \,\}$ $\quad 'k - '\varepsilon_0\, 'Cy \Rightarrow C$; $\quad 'Cy - '\varepsilon - 1 \Rightarrow Cy$; $\quad 'C + 1 \Rightarrow C$;
b) $^2C \Rightarrow K$.

a) signifie le transfert de la commande à un cycle (instruction conservée à l'adresse $'k - '\varepsilon_0\, 'Cy$) si $'Cy \neq 0$, et la diminution d'une unité du contenu du registre Cy si $'\varepsilon = 0$ (pour une opération de groupe des cellules courtes), et de deux unités si $'\varepsilon = 1$ (pour une opération de groupe des cellules longues) ; dans le cas contraire, sortie du cycle.

23. $'k_0 = 30$, opération de modification d'instructions. Cette instruction effectue :

a) $'k - '\varepsilon_0\, 'Cy \Rightarrow K$;
b) $'C + 1 \Rightarrow C$;
c) $'K + {}^2C \Rightarrow K$.

Ainsi, l'instruction « suivante », modifiée par le contenu de l'adresse $'k - '\varepsilon_0\, 'Cy$ est envoyée dans le registre des instructions.

24. $'k_0 = 37$. Cette instruction indique l'arrêt de la machine et, en plus, envoie dans le totalisateur le contenu de la cellule $'k - '\varepsilon_0\, 'Cy$:

$$'('k - '\varepsilon_0\, 'Cy) \Rightarrow S\,.$$

IV. Opérations d'appel aux organes externes

25. $'k_0 = 31$, instruction préparatoire du transfert des codes ; $'k$ est l'adresse de la première cellule de la mémoire interne (du tambour magnétique), à partir de laquelle commence l'opération. Les opérations d'appel aux organes externes sont codées par une suite de trois instructions :

$$3\ 1\ \alpha\,;$$

$$\beta_1\ \beta\,;$$

$$00\gamma\,.$$

Ici, 3 1, β_1, 00 sont les codes qui correspondent au registre k_0 ; α, β, γ les codes qui correspondent au registre k ; β_1 peut prendre les valeurs suivantes :

$$\beta_1 = 01 ; 02 ; 03$$

qui déterminent le type d'opération ; β est le numéro de zone de la bande magnétique ou perforée.

1) $\beta_1 = 01$, exécute le transfert des codes de l'organe d'entrée (bande perforée) dans l'organe mémoire interne (tambour magnétique).

2) $\beta_1 = 02$, exécute le transfert des codes de l'organe externe (bande magnétique) dans l'organe mémoire interne.

3) $\beta_1 = 03$, exécute le transfert des codes de l'organe mémoire interne vers l'organe externe.

Dans chacun de ces cas, le code γ détermine l'adresse de la dernière cellule de l'organe mémoire interne jusqu'à laquelle l'opération s'étend.

Après l'opération de transfert de code, on effectue l'instruction dont le numéro suit.

26. $'k_0 = 32$, impression du contenu du totalisateur. Si $'T_1 = 1$ la sortie se fait sur imprimante ; si $'T_1 = 0$ la sortie se fait sur bande perforée ; si $'T_2 = 1$ l'impression des codes est donnée en octal (sortie du programme), si $'T_2 = 0$ en décimal (sortie des valeurs numériques).

27. $'k_0 = 34$, saut d'une ligne sur la bande en papier.

IV. M-3

La calculatrice automatique M-3 travaille avec des codes à 31 positions. Les nombres sont représentés dans le système à virgule fixe. Une position est celle du signe, les 30 autres sont numériques (Fig. 29).

Lors de l'inscription des nombres dans le système décimal codé binaire, les deux dernières positions ne sont pas utilisées, et on peut rentrer 7 chiffres décimaux dans les 28 qui restent.

Le système des instructions de M-3 est celui à 2 adresses avec exécution séquentielle. Le code opération occupe 6 positions, les adresses 12 positions chacune.

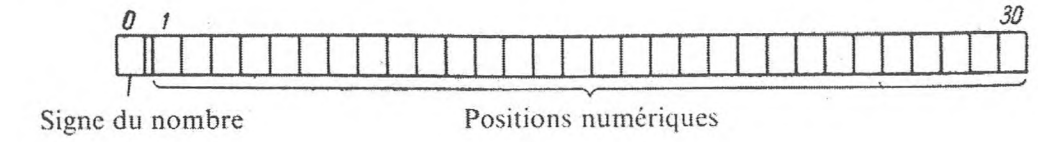


Fig. 29.

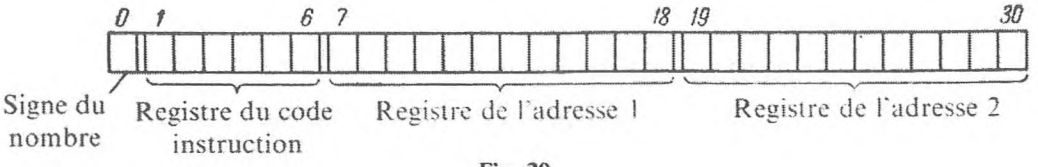


Fig. 30.

Le schéma de la répartition des positions, lorsque celles-ci contiennent une instruction, est donné figure 30.

La mémoire opération de la machine se compose de 2 048 cellules (numérotées de 0000 à 3 777 en système octal) réparties sur le tambour magnétique. La rapidité de la machine est de 30 opérations à la seconde. Les organes de calcul et l'organe de commande sont très rapides. C'est pourquoi le remplacement du tambour magnétique par un organe de mémoire à tores magnétiques augmente la rapidité jusqu'à 1 500 opérations à la seconde. L'entrée se fait par bandes perforées, la sortie par imprimante. La cellule 0000, contrairement aux machines Strela, B. E. S. M. et Kiev, est une cellule ordinaire (qui ne contient pas le code + 0).

Pour décrire les opérations, introduisons les désignations suivantes :

1) C, compteur des instructions,
2) K, registre des instructions,
3) r, registre de l'organe de calcul,
4) ω, basculeur du symbole de transfert conditionnel.

Le registre d'instructions K sera divisé en groupes de positions (les registres) (cf. tableau 14). La valeur de $'k_{02}$ détermine le type d'opération (cf. tableau 15).

Tableau 14

Groupes de positions du registre des instructions	Notation et dénomination des groupes de positions (registres)	
1-3	k_{01}, premier registre du code instruction	k_0 registre du code
4-6	k_{02}, deuxième registre du code instruction	
7-18	k_1, registre de la première adresse	
19-30	k_2, registre de la deuxième adresse	

Tableau 15

Type d'opérations	$'k_{02}$
Addition	0
Soustraction	1
Multiplication	3
Division	2
Multiplication logique position par position	6

La valeur de $'k_{01}$ détermine les particularités de l'opération à effectuer. Notons par θ n'importe quelle opération indiquée dans le tableau. En fonction de $'k_{01}$ la machine effectuera les opérations suivantes.

I. Opérations arithmétiques

1. $'k_{01} = 0$.

a) $^2k_1\,\theta^2\,k_2 \Rightarrow r$; $'r \Rightarrow 'k_2$;

b) $P\{'r \leqslant -0\}$ $\begin{matrix} 1 \Rightarrow \omega\,; \\ 0 \Rightarrow \omega\,; \end{matrix}$

c) $'C + 1 \Rightarrow C$;

d) $^2C \Rightarrow K$.

En d'autres termes, les adresses des arguments de l'opération sont codées dans les adresses de l'instruction : le résultat de l'opération est contenu dans la deuxième adresse de $'k_2$ et dans r, registre de l'organe de calcul. *b)* est l'élaboration du symbole utilisé dans les instructions de transfert conditionnel de la commande.

b) c) et *d)*, qui sont le transfert à l'instruction suivante, sont les mêmes pour toutes les instructions de ce groupe.

2. $'k_{01} = 1.$

a) $^2k_1\ \theta^2\ k_2 \Rightarrow r.$

L'opération ne change pas le contenu de l'adresse de $'k_2$, contrairement au cas où $'k_{01} = 0.$

3. $'k_{01} = 2.$

a) $'r\ \theta^2\ k_1 \Rightarrow r\ ;$
$'r \Rightarrow 'k_2.$

(L'opération θ s'effectue sur le contenu du registre r de l'organe de calcul et sur celui de la cellule $'k_1$; le résultat est envoyé dans le registre r et à l'adresse de $'k_2$.)

4. $'k_{01} = 3.$

a) $'r\ \theta^2\ k_1 \Rightarrow r.$

5. $'k_{01} = 4.$

Dans ce cas, ce sont les instructions suivantes qui sont effectuées :

a) $^2k_1\ \theta^2\ k_2 \Rightarrow r\ ;$
$'r \Rightarrow 'k_2\ ;$

imprimer $'r.$
($'r$ est imprimé sur papier).

6. $'k_{01} = 5.$

a) $|\ ^2k_1\ |\ \theta\ |\ ^2k_2\ | \Rightarrow r.$

7. $'k_{01} = 6.$

$'r\ \theta^2\ k_1 \Rightarrow r\ ;$
$'r \Rightarrow 'k_2\ ;$

imprimer $'r.$

8. $'k_{01} = 7.$

$|\ 'r\ |\ \theta\ |\ ^2k_1\ | \Rightarrow r.$

II. Instructions de transfert de la commande

1. $'k_0 = 24$, transfert inconditionnel à la 1ère adresse :

a) $'r \Rightarrow {}'k_2$;
b) $'k_1 \Rightarrow C$;
c) $^2C \Rightarrow K$.

En même temps, le contenu de *r*, registre de l'organe de calcul, est envoyé à l'adresse de $'k_2$.

2. $'k_0 = 64$, transfert inconditionnel à la première adresse avec impression du contenu de *r* :

a) $'r \Rightarrow {}'k_2$;
b) *imprimer* $'r$;
c) $'k_1 \Rightarrow C$;
d) $^2C \Rightarrow K$.

3. $'k_0 = 74$, transfert inconditionnel à la 2-ième adresse ($'k_1 = 0$) :

a) $| \, 'r \, | \Rightarrow r$;
b) $'k_2 \Rightarrow C$;
c) $^2C \Rightarrow K$.

4. $'k_0 = 34$, transfert conditionnel :

a) $P\{ \, '\omega = 1 \, \}$ $\begin{matrix} 'k_1 \Rightarrow C ; \\ 'k_2 \Rightarrow C ; \end{matrix}$
b) $^2C \Rightarrow K$.

5. $'k_0 = 37$, *arrêt* :

a) $^2k_1$ *sort sur le pupitre* ;
b) *arrêt*.

III. Opérations d'appel aux organes d'entrée (de sortie) et opération de transfert

1. $'k_0 = 07, 27$, *entrée par bande perforée* ($'k_1 = 0$) :

a) $'BP \Rightarrow {}'k_2$;
b) $'C + 1 \Rightarrow C$;
c) $^2C \Rightarrow K$.

Un seul code est transféré de la bande perforée vers la cellule de la mémoire $'k_2$.

2. $'k_0 = 05, 15,$ *transfert* :

a) $^2k_1 \Rightarrow {}'k_2$;
b) $'C + 1 \Rightarrow C$;
c) $^2C \Rightarrow K$.

3. $'k_0 = 45, 55,$ *transfert et impression* :

a) $^2k_1 \Rightarrow {}'k_2$;
b) *imprimer* $^2k_1$;
c) $'C + 1 \Rightarrow C$;
d) $^2C \Rightarrow K$.

V. KIEV

La calculatrice automatique universelle Kiev (*) utilise des codes à 41 positions. Pour une instruction, dans le système à trois adresses employé, on dispose 5 positions pour coder l'opération et 12 pour chaque adresse.

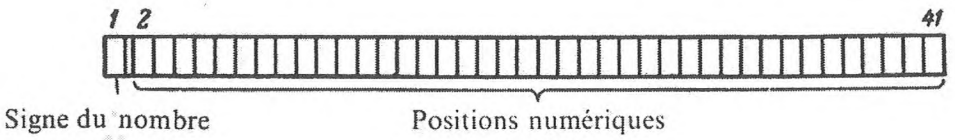


Fig. 31.

La première position de chaque adresse est réservée au signe de modification de l'adresse. Les nombres sont représentés dans le système à virgule fixe, la première position est réservée au signe.

La répartition des positions de la cellule, lorsque celle-ci contient un nombre binaire, est représentée par la figure 31.

Les machines rapides effectuent 15 000 à 20 000 opérations à la seconde du type addition et 3 000 à 4 000 du type multiplication ou division, en moyenne 9 000 opérations à la seconde.

La mémoire interne est composée de 2 048 cellules, la mémoire externe de tambours magnétiques et de bandes. La mémoire interne contient des cellules opératives et des cellules passives. Il y a trois types de cellules passives :

a) Les cellules à accès rapide.
b) Les cellules à accès lent.
c) Les cellules ajoutées au fur et à mesure des besoins.

La mémoire *a*) (à accès rapide) sert à conserver les constantes et les sous-programmes les plus fréquemment rencontrés. La mémoire *b*) (à accès lent) est composée de blocs connectés destinés à mettre en réserve les différents sous-programmes standards dont on pourra avoir besoin au cours du travail. Les huit cellules de la mémoire *c*) sont composées d'un ensemble d'interrupteurs qui permettent d'introduire manuellement les codes. La conversion des codes des nombres de décimal en binaire se fait en même temps que l'entrée et ne prend pas de temps supplémentaire. L'entrée est réalisée sur bande perforée, la sortie sur bande perforée et imprimante ; la possibilité d'utiliser des cartes perforées est prévue.

(*) Cf. B. V. Gnedenko, V. M. Glouchkov, E. L. Iouchtchenko [1] et L. N. Dachevski, S. V. Pogrebinski, E. A. Chkabara [1].

En plus des cellules de la mémoire interne, dont les adresses vont de 0000 à 3777 dans le système octal, la machine Kiev possède les registres spéciaux suivants :

1) C, compteur des instructions (11 positions),
2) K, registre des instructions (41 positions),
3) P, registre de renvoi (11 positions),
4) Cy, registre des cycles (10 positions),
5) A, registre de modification des adresses (10 positions),
6) T, registre-basculeur d'arrêt de panne (1 position).

Pour décrire les opérations qui font intervenir ces registres nous nous bornerons à les désigner par des lettres, en supposant que les adresses ainsi formulées sont différentes des codes des adresses de l'organe mémoire interne.

Conformément au principe de la commande par programme (cf. chap. II), un cycle de travail de la machine consiste à exécuter les instructions K, c'est-à-dire les instructions dont le code est présent dans le registre des instructions, et à transférer le code de l'instruction suivant au registre K. L'exécution de l'instruction K, son tour venu, dépend de son contenu (code).

Dans la machine Kiev, comme dans beaucoup d'autres, on utilise, pour conserver le numéro (l'adresse) de l'instruction dont c'est le tour, le compteur des instructions C.

Pour faciliter la description de l'ensemble des opérations élémentaires effectuées par les différentes instructions sur la machine Kiev, nous isolerons dans le registre des instructions K des groupes de positions (registres), comme il est indiqué dans le tableau 16. Au besoin une telle division est aussi opérée dans les cellules de la mémoire interne.

Tableau 16

Positions du registre des instructions de la Kiev (numérotées de gauche à droite)	Notation et dénomination des groupes de positions (registres)
1-5	k_0, registre du code opération
6	ε_1, position du signe de modification de l'adresse I
7-17	k_1, registre de l'adresse I
18	ε_2, position du signe de modification de l'adresse II
19-29	k_2, registre de l'adresse II
30	ε_3, position du signe de modification de l'adresse III
31-41	k_3, registre de l'adresse III

Le registre des instructions (et la cellule de la mémoire qui contient l'instruction) est divisé comme le montre la figure 32.

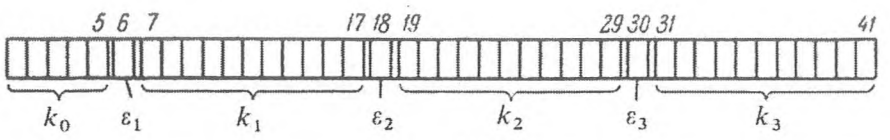


Fig. 32.

Dans les registres indiqués on peut conserver les codes octaux correspondants :

$$'k_0 = 00, 01, 02, \ldots, 37 ;$$

$$'\varepsilon_1, '\varepsilon_2, '\varepsilon_3 = 0, 1 ;$$

$$'k_1, 'k_2, 'k_3 = 0000, 0001, \ldots, 3777 .$$

En fonction de $'k_0$ (code opération), la machine effectue les opérations suivantes.

I. Opérations arithmétiques

1. $'k_0 = 01$. Addition « + ». Lors de cette opération, la machine effectue :

a) addition des nombres qui sont contenus dans les adresses $'k_1 + '\varepsilon_1 \, 'A$ et $'k_2 + '\varepsilon_2 \, 'A$, et renvoi du résultat de l'addition à l'adresse $'k_3 + '\varepsilon_3 \, 'A$:

$$'('k_1 + '\varepsilon_1 \, 'A) + '('k_2 + '\varepsilon_2 \, 'A) \Rightarrow 'k_3 + '\varepsilon_3 \, 'A ;$$

b) augmentation du contenu du compteur des instructions C d'une unité :

$$'C + 1 \Rightarrow C ;$$

c) envoi de l'instruction suivante dans le registre des instructions K :

$$^2C \Rightarrow K .$$

Le point *a*) signifie que si $'\varepsilon_i = 1$ ($i = 1, 2, 3$) l'adresse correspondante est modifiée, c'est-à-dire qu'elle est augmentée de la valeur contenue dans le registre de modification d'adresses A, et si $'\varepsilon_i = 0$ l'opérateur s'effectue directement sur l'adresse. Les points *b*) et *c*) préparent l'exécution de l'instruction suivante.

Ces particularités vont se rapporter à toutes les opérations de ce groupe.

2. $'k_0 = 02$. Soustraction « − ». Même chose que pour l'addition, mais l'opération est une soustraction :

a) $'('k_1 + '\varepsilon_1 \, 'A) - '('k_2 + '\varepsilon_2 \, 'A) \Rightarrow 'k_3 + '\varepsilon_3 \, 'A ;$

b) $'C + 1 \Rightarrow C ;$

c) $^2C \Rightarrow K.$

3. $'k_0 = 03$. Addition d'instructions « $+_1$ » :

a) $|\,'('k_1 + '\varepsilon_1\,'A) + |\,'('k_2 + '\varepsilon_2\,'A)\,|\,| \vee \text{sign}\,'('k_2 + '\varepsilon_2\,'A) \Rightarrow$
$\Rightarrow\, 'k_3 + '\varepsilon_3\,'A$;

b) $'C + 1 \Rightarrow C$;

c) $^2C \Rightarrow K$.

L'opération « $+_1$ » diffère de l'opération « + » en ce que le contenu de la cellule $'('k_1 + '\varepsilon_1\,'A)$, constante de substitution d'adresse, est ajouté au contenu de la cellule $'('k_2 + '\varepsilon_2\,'A)$ considéré comme code d'adresse, c'est-à-dire qu'il est ajouté à son module ; d'autre part le signe de l'instruction $'('k_2 + '\varepsilon_2\,A)$ est ajouté au résultat.

4. $'k_0 = 06$. Soustraction des modules « $|-|$ ». Même chose que pour « − », mais les deux termes de la soustraction sont les modules des codes correspondants.

5. $'k_0 = 07$. Addition cyclique « $Cy +$ » — addition des codes avec transfert de la position de rang le plus élevé à la position de rang le moins élevé. Comme dans les opérations précédentes, on admet la possibilité de modifier les adresses. De façon analogue on a :

6. $'k_0 = 10$. Multiplication sans arrondi « × ».

7. $'k_0 = 11$. Multiplication avec arrondi « ⊗ ».

8. $'k_0 = 12$. Division « : ».

En plus de ce qui a été exposé, au moment de l'exécution de « + », « − », « : », on envoie 1 dans le registre T si le résultat de l'opération en valeur absolue est supérieur à l'unité, et 0 dans le cas contraire. Dans le montage de l'interrupteur, l'arrêt dû à une panne correspond à $1 \Rightarrow T$.

II. Opérations logiques

9. $'k_0 = 35$. Normalisation.

a) Le nombre $'('k_1 + '\varepsilon_1\,'A)$ est normalisé, le rang du nombre est placé dans les 6 positions de poids faible de la cellule $('k_2 + '\varepsilon_2\,'A)$ et la mantisse à l'adresse $k_3 + '\varepsilon_3\,'A$;

b) $'C + 1 \Rightarrow C$;

c) $^2C \Rightarrow K$.

10. $'k_0 = 13$. Décalage logique.

a) Le code $'('k_1 + 'ε_1\ 'A)$ (y compris la position du signe) est déplacé du nombre de positions indiqué à l'adresse III de la cellule $'k_2 + 'ε_2\ 'A$, vers la droite ou la gauche suivant le signe de ce nombre ; le résultat est envoyé à l'adresse $'k_3 + 'ε_3\ 'A$;

b) $'C + 1 \Rightarrow C$;

c) $^2C \Rightarrow K$.

11. $'k_0 = 14$. Addition logique position par position « $\vee$ » (cf. chap. II).

a) $'('k_1 + 'ε_1\ 'A) \vee '('k_2 + 'ε_2\ 'A) \Rightarrow ('k_3 + 'ε_3\ 'A)$;

b) $'C + 1 \Rightarrow C$;

c) $^2C \Rightarrow K$.

Les deux opérations suivantes sont effectuées de manière analogue.

12. $'k_0 = 15$. Multiplication logique position par position « $\wedge$ ».

13. $'k_0 = 17$. Non-équivalence logique position par position « $\cong$ ».

III. Opérations de transfert de commande

Toutes les opérations de transfert de commande ne changent pas le contenu des cellules de l'organe mémoire ; leur résultat ne concerne que quelques registres spéciaux.

14. $'k_0 = 16$. Opérations de transfert conditionnel en fonction de l'égalité des modules « = ».

L'opération « = » exécute le programme suivant :

a) $P\{'('k_1 + 'ε_1\ 'A) = '('k_2 + 'ε_2\ 'A)\}$ $\begin{matrix} 'k_3 + 'ε_3\ 'A \Rightarrow C ; \\ 'C + 1 \Rightarrow C ; \end{matrix}$

b) $^2C \Rightarrow K$.

Les trois opérations suivantes sont effectuées de façon analogue.

15. $'k_0 = 04$. Transfert conditionnel de la commande en fonction de « inférieur ou égal $\leqslant$ » :

a) $P\{'('k_1 + 'ε_1\ 'A) \leqslant '('k_2 + 'ε_2\ 'A)\}$ $\begin{matrix} 'k_3 + 'ε_3\ 'A \Rightarrow C ; \\ 'C + 1 \Rightarrow C ; \end{matrix}$

b) $^2C \Rightarrow K$.

16. $'k_0 = 05$. Transfert conditionnel de la commande en fonction de « inférieur ou égal » en valeur absolue « $| \leqslant |$ » :

$$a)\ P\{\,|\,'('k_1 + '\varepsilon_1\,'A)\,| \leqslant |\,'('k_2 + '\varepsilon_2\,'A)\,|\,\}\quad \begin{matrix} 'k_3 + '\varepsilon_3\,'A \Rightarrow C\,; \\ 'C + 1 \Rightarrow C\,; \end{matrix}$$

$$b)\ {}^2C \Rightarrow K.$$

17. $'k_0 = 31$. Transfert conditionnel de la commande en fonction du signe du nombre. « T. C. N. ».

$$a)\ P\{\,'('k_1 + '\varepsilon_1\,'A) \leqslant -0\,\}\quad \begin{matrix} 'k_3 + '\varepsilon_3\,'A \Rightarrow C\,; \\ 'k_2 + '\varepsilon_2\,'A \Rightarrow C\,; \end{matrix}$$

$$b)\ {}^2C \Rightarrow K.$$

18. $'k_0 = 32$. Transfert conditionnel à un sous-programme « T. C. S. ».

$$a)\ P\{\,'('k_1 + '\varepsilon_1\,'A) \leqslant -0\,\}\quad \begin{matrix} 'k_2 + '\varepsilon_2\,'A \Rightarrow P\,;\ 'k_3 + '\varepsilon_3\,'A \Rightarrow C\,; \\ 'C + 1 \Rightarrow C\,; \end{matrix}$$

$$b)\ {}^2C \Rightarrow K\,;$$

où $'k_3 + '\varepsilon_3\,'A$ est l'adresse de l'instruction initiale du sous-programme et $'k_2 + '\varepsilon_2\,'A$ le numéro de l'instruction à laquelle la commande sera transmise à la fin du sous-programme. Si cette condition n'est pas satisfaite, le transfert se fait vers l'instruction suivante.

Le sous-programme se termine par une instruction spéciale.

19. $'k_0 = 30$. Transfert en fonction du registre de renvoi. Cette instruction exécute les opérations suivantes :

$a)\ 'P \Rightarrow C\,;$
$b)\ {}^2C \Rightarrow K.$

Le contenu des registres k_1, k_2, k_3 n'influe pas sur le résultat.

IV. Opérations d'appel aux organes externes

Toutes ces opérations sont des opérations de groupe, c'est-à-dire qui concernent des suites de codes.

20. $'k_0 = 20$. Entrée des nombres « *EN* » à partir de bandes perforées (*BP*). Cette instruction permet de placer les codes des bandes perforées, préalablement convertis de décimal codé binaire en binaire pur, dans les

cellules de la mémoire opération aux adresses $'k_1$, $'k_1 + 1, \ldots, 'k_2$. Par convention, nous écrivons cette opération de groupe sous la forme suivante :

a) $'(BP)$ code converti en binaire $\Rightarrow \begin{pmatrix} 'k_1 \\ 'k_2 \end{pmatrix}$.

De plus, par cette instruction on effectue le transfert :

b) $'C + 1 \Rightarrow C$;
c) $^2C \Rightarrow K$.

21. $'k_0 = 21$. Entrée des instructions par bandes perforées « *EI* ». Même chose que « *EN* », mais le transfert des codes se fait sans conversion.

a) $'(BP) \Rightarrow \begin{pmatrix} 'k_1 \\ 'k_2 \end{pmatrix}$;

b) $'C + 1 \Rightarrow C$;
c) $^2C \Rightarrow K$.

22. $'k_0 = 22$. Sortie des codes sur bandes perforées « *SB* ». Opération inverse : le contenu des cellules $'k_1, 'k_1 + 1, \ldots, 'k_2$ est perforé sur bandes :

a) $'\begin{pmatrix} 'k_1 \\ 'k_2 \end{pmatrix} \Rightarrow BP$;

b) $'k_3 \Rightarrow C$;
c) $^2C \Rightarrow K$.

23. $'k_0 = 23$. Echange des codes de la mémoire opération et de la mémoire externe (ME) en position « écriture » : « *MEE* ». Les codes contenus dans la suite des cellules de la mémoire opération $'k_1, 'k_1 + 1, \ldots, 'k_2$ sont transférés en ME :

a) $'\begin{pmatrix} 'k_1 \\ 'k_2 \end{pmatrix} \Rightarrow ME$;

de plus,

b) $'k_3 \Rightarrow C$;
c) $^2C \Rightarrow K$.

24. $'k_0 = 24$. Echange des codes de la mémoire opération et de la mémoire externe en position « lecture » « *MEL* ». Les codes de la ME

sont transférés dans la suite des cellules de la mémoire opération $'k_1$, $'k_1 + 1, \ldots, 'k_2$, c'est-à-dire :

$$a)\ '(ME) \Rightarrow \begin{pmatrix} 'k_1 \\ 'k_2 \end{pmatrix};$$

de plus,

$b)\ 'k_3 \Rightarrow C$;
$c)\ {}^2C \Rightarrow K$.

25. $'k_0 = 25$. Opération préparatoire aux opérations « *MEE* » et « *MEL* » qui assure l'alimentation du tambour magnétique. Ici,

$$'k_1 = \begin{cases} 0 \text{ pour l'opération 23,} \\ 1 \text{ pour l'opération 24,} \end{cases}$$

$'k_2$, numéro de *ME* à partir duquel s'effectue l'échange des codes.
$'k_3$, numéro du nombre de *ME* à partir duquel il faut commencer l'opération correspondante.

Dans ce cas, il en est de même pour les opérations suivantes :

$$'C + 1 \Rightarrow C\ ;\quad {}^2C \Rightarrow K\ .$$

26. $'k_0 = 33$. Arrêt.

V. Opérations de modification d'adresses

Les opérations de modification d'adresses sont des opérations de groupes en ce sens que le contenu des registres de modification formé par ces opérations peut être utilisé par un groupe d'instructions.

27. $'k_0 = 26$. « Début d'une opération de groupe » *DOG* :

a) On envoie un nombre qui caractérise le nombre de cycles du processus cyclique dans le registre des cycles (*Cy*)

$$'k_1 \Rightarrow Cy\ ;$$

b) On envoie la constante de substitution d'adresse dans le registre de modification d'adresses *A*

$$'k_2 \Rightarrow A\ ;$$

c) On réalise la formule prédicative :

$$P\{\,'Cy = 'A\,\}\quad \begin{matrix} 'k_3 \Rightarrow C\,; \\ 'C + 1 \Rightarrow C\,; \end{matrix}$$

d) $^2C \Rightarrow K$.

Ainsi, l'instruction *DOG* prépare le contenu du registre de substitution d'adresse et, par là-même, assure la modification convenable des adresses substituées.

Si le nombre de cycles N est connu d'avance et que p soit le pas de substitution d'adresse, nous supposerons :

$$'k_1 = {}'k_2 + Np\,.$$

Alors à chaque répétition du cycle le contenu du registre A est augmenté de la grandeur p. Tant que $'Cy \neq 'A$ les calculs cycliques continuent ; quand $'Cy = 'A$ on sort du cycle et on va à l'instruction de numéro $'k_3$.

L'augmentation du contenu du registre A du pas de substitution d'adresse p peut être obtenue par substitution d'adresse sur l'instruction *DOG* correspondante (augmentation de sa seconde adresse de la quantité p) et de la répétition de ce travail, ou à l'aide de l'instruction *FOG* suivante.

28. $'k_0 = 27$. Fin de l'opération de groupe *FOG*. Dans cette instruction :

a) le contenu du registre A augmente du pas de substitution d'adresse $'k_1 + 'A \Rightarrow A$ ($'k_1 = p$ est le pas de substitution d'adresse) ;

b) on réalise la formule prédicative :

$$P\{\,'Cy = 'A\,\}\quad \begin{matrix} 'k_3 \Rightarrow C\,; \\ 'k_2 \Rightarrow C\,; \end{matrix}$$

c) $^2C \Rightarrow K$.

Ici, $'k_3$ est le numéro de l'instruction qui transmet la commande à la fin du cycle, $'k_2$, le numéro de l'instruction qui transmet la commande au cours des calculs cycliques. Nous remarquerons que $'k_2$ n'est pas un numéro d'instruction *DOG*, car la dernière instruction ne se répète plus au moment du passage d'un cycle à un autre, sa répétition aurait restitué au registre A son contenu initial.

29. $'k_0 = 34$. Chargement du registre A par fixateur (appel par fixateur) « Φ ».

En plus de l'opération « *DOG* », qui assure le chargement du registre de modification d'adresses, l'opération « Φ » qui remplit aussi cette fonction est réalisée dans la machine. Tandis que dans l'instruction *DOG* la grandeur envoyée dans le registre A est donnée en clair comme $('k_2)$, dans l'instruction Φ cette grandeur n'est donnée que par son adresse. Dans l'instruction Φ :

a) $'('k_1 + '\varepsilon_1\ 'A)_2 \Rightarrow A$;
b) $'('('k_1 + '\varepsilon_1\ 'A)_2) \Rightarrow 'k_3$;
c) $'C + 1 = C$;
d) $^2C \Rightarrow K$;

$'k_2$ n'est pas utilisé pendant l'exécution de cette instruction.

Appelons $'\alpha_2$ le contenu de l'adresse II de la cellule α. Si

$$'k_1 = \alpha \qquad \text{et} \qquad '\alpha_2 = \beta\ ,$$

alors l'opération Φ devient :

a) $\beta \Rightarrow A$;
b) $'\beta \Rightarrow 'k_3$;
c) $'C + 1 \Rightarrow C$;
d) $^2C \Rightarrow K$.

Il est commode d'utiliser l'opération Φ, ce qui est clair d'après le chapitre V.

On peut montrer que l'opération Φ permet aux processus cycliques à paramètres arbitraires d'établir des programmes qui ne changent pas en cours de travail (sans instructions de substitution d'adresse).

BIBLIOGRAPHIE

[1] Session de l'Académie des Sciences de l'U. R. S. S. sur les problèmes d'automatisation de la production, 15-20 octobre 1956, T. 1-7. Editions de l'Académie des Sciences de l'U. R. S. S., 1957, en particulier T. I.

[1] BOOTH, A. et BOOTH, K., *Les calculatrices automatiques*, traduit de l'anglais par V. M. KOUROTCHKINE. Editions physico-mathématiques, 1959.

[1] BROOK, I. C., *La calculatrice automatique rapide M-2*, éditions techniques d'Etat, 1957.

[1] *Les calculatrices automatiques rapides*, traduit de l'anglais par D. I. PANOV. Editions étrangères, 1952.

[1] HILBERT, D. I. ACKERMAN, B., *Bases d'une logique théorique*. Editions étrangères, 1954.

[1] GNEDENKO, B. V., GLOUCHKOV, V. M., IOUCHTCHENKO, E. L., *Les paramètres mathématiques de la calculatrice automatique universelle Kiev*. Publication du Centre de calcul de l'Académie des Sciences de la République Socialiste d'Ukraine, T. II, 1960.

[1] DACHEVSKI, L. N., POGREBINSKI, S. V., CHKABARA, E. L., *Les paramètres techniques de la calculatrice automatique universelle Kiev*. Publication du Centre de calcul de l'Académie des Sciences de la République Socialiste d'Ukraine, T. II, 1960.

[1] ERCHOV, A. P., *Programme de génération pour une calculatrice rapide électronique*. Editions de l'Académie des Sciences de l'U. R. S. S., 1958.

[2] ERCHOV, A. P., *Algorithmes opératoriels*. Communication faite à l'Académie des Sciences de l'U. R. S. S., 122, n° 6 (1958), 967-970.

[1] KAGAN, B. M. et TER-MIKAELIAN, G. M., *Résolution des problèmes techniques sur les calculatrices automatiques*. Editions des Sciences et Techniques énergétiques, 1958.

[1] KALOUYNINE, L. A., *Algorithmes associés à des problèmes mathématiques. Problèmes de cybernétique*, 2e édition. Editions physico-mathématiques, 1959, 51-67.

[1] KITOV, A. P. et KRINITSKI, N. A., *Calculatrices électroniques et programmation*. Editions physico-mathématiques, 1959.

[1] KOROLIOUK, V. S., *Méthode de programmation*. Supplément aux publications de l'Académie des Sciences de la République Socialiste d'Ukraine, n° 12 (1958), 1292-1295.

[1] KOROLIOUK, V. S. et IOUCHTCHENKO, E. L., *Questions de théorie et de pratique de la programmation*. Publication du Centre de calcul de l'Académie des Sciences de la République Socialiste d'Ukraine, T. 1, 1960.

[1] LIAPOUNOV, A. A., *Schémas logiques des algorithmes, problèmes de cybernétique*, 1re édition. Editions physico-mathématiques, 1958, 46-74.

[1] LIAPOUNOV, A. A. et CHESTOPAL, G. A., *Introduction à la résolution des problèmes sur calculatrices électroniques*. Enseignement mathématique. Editions techniques, 1957, 57-74.

[2] LIAPOUNOV, A. A. et CHESTOPAL, G. A., *Description algorithmique des processus de commande*. Enseignement mathématique, 2e édition, 1957, 81-95.

[1] LEBEDEV, C. A. et MELNIKOV, V. A., *Description générale de la B. E. S. M. et méthode d'exécution des opérations*. Editions physico-mathématiques, 1959.

[1] MARKOV, A. A., *Théories des algorithmes*. Travaux de l'Institut mathématique, Académie des Sciences de l'U. R. S. S., publication nº 42, 1954.

[1] NOVIKOV, P. S., *Introduction à la logique mathématique*. Editions Dunod, 1964.

[1] RICHARDS, R. K., *Opérations arithmétiques sur les calculatrices*. Editions étrangères, 1957.

[1] CHOURA-BOURA, *Systèmes de sous-programmes standards*. Editions physico-mathématiques, 1958.

[1] TRAHTENBROT, B. A., *Algorithmes et machines à calculer*. Edition Dunod, 1963.

[1] WILKES, M., WHILLER, D., HILL, S., *Ecriture de programmes pour une calculatrice numérique électronique*. Traduit de l'anglais en russe par D. I. PANOV. Editions étrangères, 1953.

[1] FADDEIEV, D. K. et FADDEIEVA, B. N., *Résolution numérique de l'algèbre linéaire*, éditions physico-mathématiques, 1960.

[1] FRIEDMAN, V. M., *Nouvelles méthodes de résolution d'une équation linéaire opératorielle*. Communication faite à l'Académie des Sciences de l'U. R. S. S., 128, nº 3 (1959), 482-484.

[1] IOUCHTCHENKO, E. L., *Algorithmes à adresses pour une calculatrice automatique*, Publication du Centre de calcul de l'Académie des Sciences de la République Socialiste d'Ukraine, T. II, 1960.

[1] IOUCHTCHENKO, E. L., BYSTROVA, L. P., *Programme de génération, algorithmes à adresses*. Publication du Centre de calcul de l'Académie des Sciences de la République Socialiste d'Ukraine, T. III (1960).

[1] IANOV, I. I., *Schémas logiques des algorithmes. Problèmes de cybernétique*, 1re édition. Editions physico-mathématiques, 1958, 75-127.

Imprimé en France.

Photocomposition : Imprimerie JOUVE, 12, Rue de Tournon, PARIS (6e).

Tirage Offset par Joseph FLOCH, Maître-Imprimeur,
8, Rue Charles-de-Blois, 53-Mayenne.

Dépôt légal : Nº 6013. — 1er trimestre 1969.