UNIVERSITE DE NANCY FACULTE DES SCIENCES

Tykka
113 P
Sc N 61/15

CALCULATEUR A ARITHMETIQUE BINAIRE CODA AFFECTATION DE MEMOIRES -

THESE

PRESENTEE A LA FACULTE DES SCIENCES
DE L'UNIVERSITE DE NANCY
POUR L'OBTENTION
DU DOCTORAT DE SPECIALITE
MATHEMATIQUES 3º CYCLE

PAR JACQUELINE LEONARD

17-6

ET SOUTENUE EN JUIN 1961
DEVANT LA COMMISSION D'EXAMEN

UNIVERSITÉ NANCY BIBLIOTHÈQUE

jury	MM. LEGRAS	président
	. HERVE	examinateurs

UNIVERSITE DE NANCY FACULTE DES SCIENCES

CALCULATEUR A ARITHMETIQUE BINAIRE

CODA. AFFECTATION DE MEMOIRES

UNIVERSITE NANCY BIBLIOTHEQUE

PRESENTEE PAR JACQUELINE LEONARD

UNIVERSITE de NANCY - FACULTE des SCIENCES

Doyen : M. ROUBAULT
Assesseur : M. WAHL

Doyens honoraires : MM. CORNUBERT, DELSARTE

Professeurs honoraires : MM. GUTTON, CROZE, RAYBAUD, LAFFITTE, LERAY, JOLY, LAPORTE, EICHHORN, CAPELLE, GODEMENT, DUBREIL, L. SCHWARTZ, DIEUDONNE, de MALLEMANN, LONGCHAMBON, LETORT, DODE, GAUTHIER, GOUDET, OLMER, CORNUBERT, CHAPELLE, GUERIN, CHEVALLIER, REMY.

Maîtres de conférences honoraires : MM. RAUX, LIENHART.

PROFESSEURS

MM.

URION	Chimie biologique		
DELSARTE	Analyse supérieure	FRUHLING	Physique
ROUBAULT	Géologie	LIONS	M.M.P.
VEILLET	Biologie animale	SUHNER	Physique exp.
ECHEVIN	Botanique	HILLY	Géologie
WAHL	Chimie Org. indust.	LE GOFF	Génie Chimique
BARRIOL	Chimie théorique	CHAPON	Chimie biologique
BIZETTE	Physique	HEROLD	Chimie indust.
GUILLIEN	Electronique	SCHWARTZ	Exploit. minière
GIBERT	Chimie Physique	GAYET	Physiologie
HERVE	Calcul diff. et Int.	MANGENOT	Phytopathologie
LEGRAS	Mécanique rationnelle	MALAPRADE	Chimie
DAVID	Chimie organique	HADNI	Physique
BOLFA	Minéralogie	DELAMARE-DEBOUTEVILLE	Zoologie
NICLAUSE	Chimie	BONVALET	Mécanique Physique
FAIVRE	Physique appliquée	BRUHAT	Mathématiques géné.
AUBRY	Chimie minérale	Mme ROIZEN	Physique
DUVAL	Chimie	KERN	Minéralogie
COPPENS	Radiogéologie	BASTICK	Chimie

MAITRES de CONFERENCES

MM.

WERNER	Botanique	CLIN	Paléontologie
GARNIER	Agronomie	ROCCI	Géologie
REGNIER	Physico-chimie	WEISS	Physique M.P.C.
WEPPE	Minéralogie appli.	Mme BASTICK	Chimie M.P.C.
BERNARD	Géologie	VUILLAUME	Psychophysiologie
ARAGNOL	Mathématiques (prop.)	PLAN	Mathématiques
GAY	Chimie biologique	GUDEFIN	Physique
RENARD	Physique théo. et nuc.	MARI	Chimie I.S.I.N.
CONDE	Zoologie	LAFON	Physique I.S.I.N.
GOSSE	Génie chimique	HORN	Physique
CHAMPIER	Physique	PIERRET	Maître de conf. adj.

Secrétaire Principal : M. CARON

L'expression de ma gratitude ira à Monsieur le Professeur LEGRAS, pour sa bienveillante attention, et pour les nombreux conseils qu'il a bien voulu me prodiguer deux années durant. Mes remerciements à Monsieur BRU ingénieur, qui m'a très aimablement guidée dans la première partie de ce travail.

Que Messieurs les Professeurs qui ont accepté de former mon jury soient également remerciés.

S O M M A I R E

Première Partie

CARACTERES GENERAUX DE TYKKA 1

Information 1
Langage machine 3
Registres 5

ARITHMETIQUE BINAIRE 7

Représentation d'un nombre 7
Addition 8
Soustraction I0
Multiplication II
Division I3

SIMULATION SUR ORDINATEUR 650 . . . . . I5

Conventions de numérotation I7
Conversions octal . binaire 23
Opérations simulées 27
Liste des ordres simulés 43
Etude détaillée des ordres TYKKA 45

UTILISATION DE LA SIMULATION 62

Deuxième Partie

PRESENTATION DU LANGAGE SYMBOLIQUE 65

TABLEAU DE CONNEXION 7I

SOUS-PROGRAMMES DE BASE 79

Table des symboles 8I
Table des réservations 85

ETUDE DES PSEUDO-CODES 94

ETUDE D'UN ORDRE SYMBOLIQUE 99

Ordres négatifs I0I
Ordres positifs I05
Adresse instruction suivante I09

UTILISATION DU PROGRAMME CODA II2

P R E M I E R E P A R T I E

D'UNE MACHINE A TRAITER L'INFORMATION.
DE SON ORGANISATION LOGIQUE ET DES
REGLES ARITHMETIQUES QU'ELLE UTILISE.
DE LA SIMULATION OBTENUE SUR ORDINA-
TEUR 650 IBM.

CARACTERES GENERAUX DU CALCULATEUR TYKKA

ORGANISATION DE L'INFORMATION

La caractéristique essentielle de TYKKA est de travailler uniquement sur des informations écrites dans le système binaire, tant pour les entrées et sorties mémoire qu'au cours des opérations internes. Toute donnée est donc représentée par un groupe de digits binaires respectivement de poids "zéro" et "un". Mais le digit binaire n'est pas l'élément directement accessible à l'opérateur, deux types de mots sont utilisés dont nous allons étudier la structure : le caractère et le mot-machine.

Le caractère

C'est l'unité élémentaire d'information traitée à l'intérieur du calculateur. Il est composé de 7 digits binaires dont six portent l'information, le septième est utilisé pour des tests de parité.

Exemple : 010110 | 0
information | parité impaire

Considéré comme combinaison de six digits, zéro ou un, ce caractère représente un des éléments du code alphanumérique TYKKA, qui disposera ainsi de $2^6 = 64$ caractères distincts.

Mais dans la représentation d'un nombre arithmétique, les six digits sont considérés comme six chiffres, de valeur 0 ou 1, et de poids successifs décroissants.

Toute information est u e succession de caractères, en nombre variable.

Le mot-machine

C'est un ensemble ordonné de trois caractères auquel correspond une "adresse en mémoire" ; soit $(X_1 X_2 X_3)$. Suivant le type des caractères XI, X2, X3, cette triade peut représenter une information alphanumérique ou une valeur numérique.

Dans le second cas, on ignore la structure ternaire du mot-machine pour ne considérer que la succession de I8 digits binaires dont il est formé, soit : $d_0\ d_{-1}\ d_{-2} \ldots\ldots\ d_{-I7}$

chacun d'eux est alors un chiffre de valeur 0 ou 1, et leurs poids successifs vont en décroissant.

Nous venons de dire que seul un groupe de trois caractères était adressable. Cependant, le programmeur pourra utiliser ou modifier seulement 1, 2 ou 3 des caractères de la triade grâce à un indicatif joint à son adresse.

Pour noter un caractère ou un mot-machine, la notation binaire est longue et peu maniable pour le programmeur. C'est donc la notation octale qui sera le plus souvent utilisée car la correspondance entre les deux systèmes est directe : si on considère trois digits binaires de poids successifs 2^2, 2^I, 2^0, il leur correspond un chiffre octal, d'où la table suivante :

digits binaires	chiffre octal
000	0
00I	I
0I0	2
0II	3
I00	4
I0I	5

II0	6
III	7

Le caractère s'écrit alors avec deux chiffres octaux, le mot-machine avec six chiffres octaux.

Exemple : $X = (0I0\ 0II)_2 = (2\ 3)_8$

$d_0\ d_{-1} \ldots\ldots\ d_{-I7} = (0I0\ 0II\ III\ 00I\ I0I\ II0)_2 = (23.7I.56)_8$

ORGANISATION LOGIQUE INTERNE

TYKKA est caractérisé par un fonctionnement séquentiel avec une instruction qui occupe deux mémoires consécutives, c'est-à-dire : après exécution de l'ordre placé en N, N+I, il y aura exécution de l'ordre placé en N+2, N+3, et ainsi de suite. Seuls des ordres de saut, conditionnels ou non, peuvent provoquer une rupture de cette séquence. Un compteur d'instructions CI progresse de deux en deux à l'exécution de chaque ordre.

Pendant que s'exécute l'ordre placé en N, N+I, CI contient toujours l'adresse N+2 qui sera "adresse instruction suivante" s'il n'y a pas de rupture de séquence.

Etude du langage machine de TYKKA

L'instruction est formée de deux mots-machines, afin de porter suffisamment de renseignements. Ces deux mots sont obligatoirement rangés dans les triades d'adresses N et N+I. La première mémoire, contenu de N, représente la partie logique de l'ordre, la seconde, contenu de N+I, la partie arithmétique ou partie adresse. Ces instructions sont toujours notées en octal, donc par deux groupes

de six chiffres que nous symboliserons ainsi :

(N)	(N+I)
CØ. X. M. AP.	AI. A2. A3.

Partie logique (N) CØ. X. M. AP.

I° caractère : CØ ou code opération.

Il spécifie laquelle des 56 opérations possibles le calculateur devra effectuer, (une liste en est donnée page).

2° caractère : X ou partie logique de l'adresse.

C'est l'indicatif qui spécifie sur quelle partie de la triade indiquée en N+I doit porter l'opération. X peut prendre 5 valeurs distinctes.

X = 7 = III les 3 caractères sont intéressés
X = 6 = IIO 2 caractères seulement le sont
X = 4 = IOO
X = 2 = OIO 1 seul caractère est intéressé
X = 1 = OOI

Le digit 1 indique la position des caractères intéressés.
Le digit 0 celle des caractères ignorés dans cette opération. Il faut remarquer que X peut être interprété ou non suivant le code opération qu'il accompagne.

2° caractère : M numéro d'un registre d'indices.

TYKKA possède huit registres d'indices numérotés de 0 à 7. Quelle que soit l'opération définie par CØ, la partie arithmétique de l'ordre est modifiée par le contenu du registre indiqué par M : AI A2 A3 est remplacé par AI A2 A3 + contenu de RiM.
Exception : si M=0, il n'y a pas de modification d'adresse quelque soit le contenu du registre d'indice 0.

3° caractère : AP.

C'est également le numéro d'un registre d'indice, il n'est pris en considération que par trois groupes d'ordres :
- les ordres qui demandent une opération sur un RI, AP représente alors le numéro du registre intéressé.
- les ordres de saut : en cas de rupture de séquence le contenu du compteur CI est mémorisé dans RI AP si AP est différent de zéro.
- les ordres d'entrée-sortie : AP représente le numéro de référence de l'organe périphérique appelé.

Partie arithmétique (N+I) AI. A2. A3.

Elle peut avoir des interprétations distinctes suivant le code opération qu'elle accompagne. Dans tous les cas, c'est AI A2 A3 modifié par RiM qui est à considérer.

- ordres de décalage : les deux caractères AI et A2 sont ignorés, A3 représente le nombre de décalage à effectuer.
- ordres d'entrée-sortie d'un message : les caractères AIA2 permettent de construire l'adresse AIA200 où sera rangé le premier mot du message, A3 représente le nombre de mots de ce message.
- pour tous les autres ordres AIA2A3 est une adresse facteur Am : adresse d'une donnée à prendre en mémoire, ou d'une mémoire vers laquelle il faut transferer un résultat.

ROLES JOUES PAR LES DIFFERENTS REGISTRES DU CALCULATEUR

Au cours de l'exécution d'un ordre, différents registres sont mis en jeu. Ils sont internes au calculateur, c'est-à-dire inaccessibles au programmeur, non adressables. Caractère commun : leur capacité est de 1 mot-machine, soit 3 caractères.

CI : compteur d'instruction, il contient toujours l'adresse de l'instruction suivante.

Registre opération : il reçoit la partie logique de l'instruction à exécuter, puis se charge de présenter aux divers décodeurs la partie de l'ordre qui les intéresse.

Registre adresse : il reçoit la partie arithmétique de l'instruction, puis subit la modification d'adresse indiquée par M, il contient alors : A + (RiM) = Am

Registre test T : il n'entre en jeu qu'avec les instructions qui opèrent sur le contenu d'un registre d'indice. C'est une mémoire intermédiaire entre ces registres et les autres organes du calculateur. Quand un nombre est envoyé dans un RI, il passe par T et

s'y inscrit ; de même quand on transfert le contenu d'un Ri vers la mémoire générale, la valeur transférée s'inscrit en T.

Registres de l'unité arithmétique.

Accumulateur : le programme doit y placer les données voulues avant l'exécution d'opérations arithmétiques ou logiques. C'est dans ce registre que se construit le résultat de l'opération au fur et à mesure de son déroulement. En fin d'opération il contient en général le résultat cherché.

Registre multiplicateur-quotient : ce registre complète le précédent et en dépend très étroitement. Il permet de travailler sur des nombres de six caractères en particulier.

Registres adressables : les registres d'indices : ils sont formés par un groupe de huit mémoires, choisi en tête d'une bande par un indicateur du pupitre. Le groupe choisi est alors connecté aux circuits de modification d'adresse, mais ils restent adressables au même titre que des mémoires quelconques.

* * *

R E G L E S D'A R I T H M E T I Q U E B I N A I R E

Les nombres sur lesquels TYKKA calcule sont écrits dans le système binaire et composés de I8 digits que nous noterons :

$$d_o \quad d_{-1} \quad d_{-2} \ldots\ldots\ldots \quad d_{-I7}$$

Les seuls nombres représentés sont, en valeur absolue, inférieurs à l'unité, les digits qui les expriment ont alors pour poids successifs : 2° pour d_o, 2^{-I} pour d_{-1} ..... 2^{-I7} pour d_{-I7}.

On a pu remarquer que le mot-machine TYKKA ne possède pas de position de signe, en effet la convention d'écriture d'un nombre algébrique est la suivante :

- un nombre positif m est représenté par sa valeur exacte, c'est-à-dire, si $M = d_o, d_{-1} \ldots\ldots d_{-I7}$

$$m = d_o 2^o + d_{-1} 2^{-I} + d_{-2} 2^{-2} + d_{-3} 2^{-3} + \ldots\ldots + d_{-I7} 2^{-I7}$$

le digit d_o est alors toujours nul.

- un nombre négatif m est représenté par son complément à 2, c'est-à-dire, si $M = d_o, d_{-1} \ldots\ldots d_{-I7}$

$$m = d_o 2^o + d_{-1} 2^{-1} + \ldots\ldots + d_{I7} 2^{-I7} - 2^I$$

le digit d_o sera alors 1, puisque M est toujours supérieur à 1.

Le digit d_o est caractéristique du signe du nombre représenté

$$\left| \begin{array}{ll} d_o = 0 & \text{nombre} \geqslant 0 \\ d_o = 1 & \text{nombre} < 0 \end{array} \right.$$

La représentation M du nombre m est donnée dans les deux cas par l'égalité

$$M = m + 2 \quad \text{(modulo 2)}$$

La correspondance entre les systèmes binaire et octal se fait rapidement en groupant par trois les digits après la virgule. Le dernier groupe est incomplet, il est complété par un zéro.

Exemples :

le nombre $m_2 = 0,$ IOI IOO OIIOIO OII II/O

s'écrit en octal $m_8 = +\ 0,54\ 32\ 36$

le nombre $m_2 = 1,$ IIO OIO IIO IOI OII IO/O

représente $m'_2 = -\ 0,$ OOI IOI OOI OIO IOO IO/O

et s'écrit en octal $m'_8 = -0,15\ 12\ 44$

Toutes les opérations arithmétiques se font par l'intermédiaire de trois opérations de base : addition, décalage d'une position binaire, prise de l'opposé d'un nombre.

Addition

Soient deux nombres algébriques m_1 et m_2 représentés par M_1 et M_2 dans le calculateur. On veut obtenir la représentation du nombre $(m_1 + m_2)$

Par convention :

$M_1 = m_1 + 2 \qquad M_2 = m_2 + 2 \qquad$ (modulo 2)

d'où $\quad M_1 + M_2 = m_1 + m_2 + 4$

$M_1 + M_2 = m_1 + m_2 + 2 \qquad$ (modulo 2)

Egalité qui nous montre que la somme des représentations est égale à la représentation de la somme algébrique $m_1 + m_2$ dans les conventions adoptées. On applique donc aux représentations les règles d'addition classiques :

0 + 0 = 0	avec	r = 0
0 + I = I	"	r = 0
I + 0 = I	"	r = 0
I + I = 0	"	r = I

L'exécution de la somme modulo 2 s'obtient en négligeant la retenue dans l'ordre 2^I qui ne possède d'ailleurs pas de digit représentatif.

Pour être valable, un résultat d'opération algébrique doit être, en valeur absolue, inférieur à 1, dans le cas contraire la somme $M = M_1 + M_2$ ne représente plus $m_1 + m_2$. Les opérations vont donc s'accompagner de tests de dépassement.

Dépassement en addition : il peut se produire si m_1 et m_2 sont de même signe, l'étude du digit d_o permet de le détecter,

si m_1 et m_2 sont positifs $\left.\begin{array}{l} d_o m_1 = 0 \\ d_o m_2 = 0 \end{array}\right|$ il y a dépassement si $d_o M = 1$

si m_1 et m_2 sont négatifs $\left.\begin{array}{l} d_o m_1 = \bar{I} \\ d_o m_2 = 1 \end{array}\right|$ il y a dépassement si $d_o M = 0$

Opposé d'un nombre

On obtient l'opposé d'un nombre m en prenant son complèment à 2, ce qui correspond en binaire à inverser tous les digits représentant m et à ajouter 2^{-I7} au résultat.

Exemple : X = 0, IOI OOI OII IIO IOO IO

2-X = I OIO IIO IOO OOI OII IO

qui correspond bien à

```
  1, OIO IIO IOO OOI OII OI
+                       ....OI
  1, OIO IIO IOO OOI OII IO
```

Décalages arithmétiques

Un décalage d'une position binaire correspond à une multiplication ou à une division par 2, suivant que le digit de rang -i, qui a pour poids 2^{-i}, prend la place du digit de rang -(i-1) ou -(i+1). Il faut regarder si l'opération effectuée sur les représentations donne le résultat voulu.

Décalage à droite sur m représenté par M

- si m est positif $M = m$ $\quad \frac{M}{2} = \frac{m}{2}$

Le nombre $\frac{m}{2}$ doit être aussi positif, on obtient directement sa représentation en décalant M et en introduisant zéro dans la position d_o

- si m est négatif $M = m + 2$ d'où $\frac{M}{2} = \frac{m}{2} + 1$

Dans ce cas le nombre $\frac{m}{2}$ est également négatif, et sa représentation donnée par $\frac{m}{2} + 2$. Or $\frac{m}{2} + 2 = \frac{M}{2} + I$.

Cette représentation sera obtenue en décalant M d'un rang vers la droite, et en ajoutant 1 dans la position d_o devenue libre.

Règle : dans les deux cas, le digit d_o qui caractérisait le signe du nombre donné, a été reproduit dans la nouvelle position d_o.

<u>Décalage à gauche</u> : que m soit positif ou négatif,

$M = m + 2$ (modulo 2) d'où $2M = 2m + 4$

$2M = 2m + 2$ (modulo 2)

La représentation voulue est obtenue directement, en d_{-I7} s'introduit toujours zéro.

<u>Dépassement en décalage</u> : un dépassement de capacité peut se produire au cours d'un décalage à gauche, qui correspond à une multiplication par 2. Il se caractérise par un changement du digit qui apparait en d_o. Cette position est testée à chaque pas du décalage.

Opérations complexes. Ce sont celles de soustraction, multiplication et division, qui utilisent les trois opérations de base précédentes.

<u>Soustraction</u>

Pour effectuer $m_1 - m_2$, TYKKA cherche la représentation de $(-m_2)$, puis effectue une addition sur les deux représentations obtenues : m_1 et $(-m_2)$.

<u>Multiplication</u>

Soit à effectuer le produit : multiplicande "a" par multiplicateur "m". La règle classique d'opération consiste à construire une suite de produits partiels en testant, digit par digit, le multiplicateur : si ce digit est 1, le produit partiel est décalé d'un rang sur la droite et "a" lui est ajouté, si ce digit est 0, le produit partiel est décalé sans autre modification.

Ce procédé appliqué à des nombres de I7 digits peut donc demander jusqu'à I7 additions successives d'où méthode assez lente. La règle de multiplication appliquée ici a voulu réduire le nombre maximum d'opérations en cherchant en premier une représentation différente du multiplicateur m.

Soit $M = d_o\, d_{-1}\, d_{-2} \ldots\ldots\ldots d_{-I7}$ la représentation du nombre algébrique m, telle que $M = m + 2$ (modulo 2) ; chaque digit d_{-j} a pour valeur 0 ou 1 d'où $m = d_o 2^o + d_{-1} 2^{-1} + \ldots\ldots + d_{-17}\, 2^{-I7}$ mod.2

qui peut s'écrire $m = d_1 2^{+I} + d_o 2^{\prime} + \ldots\ldots\ldots + d_{-17} 2^{-I7}$

avec $d_1 = - d_o$

Nous cherchons une représentation telle que

$m = e_o 2^o + e_{-1} 2^{-1} \ldots\ldots + e_{-I7} 2^{-I7}$ où e_{-j} peut prendre trois valeurs : 0, 1 ou -1.

e_{-j} est obtenu en testant les deux digits consécutifs $d_{-(j-1)}$ et d_{-j} ainsi qu'un digit "report" dans l'ordre -j : r_{-j} .

Soit à considérer le nombre $d_{-(j-1)} S_{-j}$ où $S_{-j} = d_{-j} + r_{-j}$ et à le remplacer par un nombre de même valeur, de la forme $S_{-(j-1)} e_{-j}$

en remarquant que $\left| \begin{array}{l} e_{-j} = 0, 1, \text{ ou } -1 \\ S_{-j} = 0, 1, \text{ ou } 2 \end{array} \right.$

- Si $S_{-j} = d_{-j} + r_{-j} = 0$ c'est que $d_{-j} = r_{-j} = 0$

on conserve alors le groupe $d_{-(j-1)} S_{-j}$ d'où $\left| \begin{array}{l} e_{-j} = 0 \\ r_{-(j-1)} = 1 \end{array} \right.$

- Si $S_{-j} = 2$ il est équivalent en binaire à I0, il sera donc remplacé par $\left| \begin{array}{l} e_{-j} = 0 \\ r_{-(j-1)} = 1 \end{array} \right.$

- Si $S_{-j} = 1$ ou $d_{-(j-1)} = 0$, le groupe $d_{-(j-1)} S_{-j}$ reste inchangé d'où $\left| \begin{array}{l} e_{-j} = 1 \\ r_{-(j-1)} = 0 \end{array} \right.$

ou $d_{-(j-1)} = 1$, on écrit alors $S_{-j} = 2 - 1$ d'où $\left| \begin{array}{l} e_{-j} = -1 \\ r_{-(j-1)} = 1 \end{array} \right.$

Ces différents cas se résument dans le tableau suivant :

S_{-j}	d_{-j} , r_{-j}	$d_{-(j-1)}$	e_{-j}	$r_{-(j-1)}$
0	0 0	0 ou 1	0	$0 = r_{-j}$
2	1 1	0 ou 1	0	$1 = r_{-j}$
1 1	0 1 0 1	0 1	1 -1	0 $1 = r_{-j}$
1 1	1 0 1 0	0 1	1 -1	$0 = r_{-j}$ 1

Remarques : dans les cas $S = 0$ et $S = 2$ il est inutile de tester la valeur de $d_{-(j-1)}$.

- la valeur du report dans l'ordre précédent change seulement dans deux cas, dans les autres le report r_{-j} est simplement transmis au rang $-(j-1)$.
- sur deux valeurs consécutives de e_{-j} l'une au moins est toujours nulle, il y a donc au plus huit digits significatifs dans la nouvelle représentation de m.

La règle obtenue précédemment est valable jusqu'à la rencontre du digit d_o, mais le digit d_1 qui est 0 ou -1 ne peut pas la suivre. On doit alors prendre pour e_o une valeur qui en tienne compte, soit : $e_o = 2d_1 + d_o + r_o = r_o - d_o$.

En réalité il est difficile d'appliquer une règle particulière au dernier digit. On partira donc d'une représentation de m donnée par $M = d'_1\ d_o\ d_{-1} \cdots\cdots d_{-17}$, où $d'_1 = d_o$, après avoir vérifié que la règle générale appliquée à d'_1 donnait exactement $e_o = r_o - d_o$

- $S_o = 0$ donne $e_o = 0$
  or $r_o = d_o = 0$ $\quad r_o - d_o = 0$
- $S_o = 2$ donne $e_o = 0$
  or $r_o = d_o = 1$ $\quad r_o - d_o = 0$
- $S_o = 1$ $\quad d_1 = 0$ donne $e_o = 1$
  or $r_o = 1$ $\quad d_o = 0$ $\quad r_o - d_o = 1$

$d_1 = 1$ donne $e_o = -1$
or $r_o = 0$ $\quad d_o = 1$ $\quad r_o - d_o = -1$

A partir de la nouvelle représentation $M = e_o\ e_{-1}\ e_{-2} \cdots\cdots e_{-17}$, nous construisons le produit en testant digit par digit, le multiplicateur. Le nombre maximum d'opérations demandées sera alors de huit.

Division

Soit à effectuer la division algébrique d'un nombre donné x par le nombre y. Comme tous les nombres du calculateur TYKKA, x et y sont, en valeur absolue, inférieurs à l'unité. Nous supposerons d'autre part $|x| \leq |y|$. La division correspond, par définition, à la recherche de deux nombres q et r vérifiant les relations.

$$x = qy + r \qquad \text{et} \qquad |r| \leq 2^{-17}\ |y|$$

(précision imposée par la longueur des nombres utilisés).

Processus utilisé. x est considéré comme le premier reste partiel de la division, le reste partiel suivant est donné par :

$$r_1 = 2r_o + e_1 y \qquad \text{avec } r_o = x$$

où $e_1 = +1$ si r_o et y sont de signe contraire
$e_1 = -1$ si r_o et y sont de même signe
soit : $e(0,0) = e(1,1) = -1$
$e(0,1) = e(1,0) = +1$

Le premier chiffre binaire du quotient, qui correspond à l'opération commandée par e est donné par :

$$z(0,0) = z(1,1) = 1$$
$$z(0,1) = z(1,0) = 0$$

On remarque que $e = 1-2z$ dans tous les cas. On renouvelle ce processus jusqu'à obtenir un résultat z de I7 chiffres binaires. Regardons alors si z_f et r_f ainsi obtenus vérifient les conditions imposées.

Pour l'ordre k nous avons $\begin{cases} r_k = 2\, r_{k-1} + e_k y \\ e_k = 1 - 2\, z_k \end{cases}$

avec dans tous les cas $|r_k| \leq |y|$ puisqu'au départ $|r_o|$ est inférieur ou égal à $|y|$.

Considérons alors : $r_k = 2\, r_{k-1} + (1 - 2\, z_k)\, y$

ou $2^{-k} r_{-k} = 2^{-(k-1)} r_{k-1} + (2^{-k} - 2^{-(k-1)} z_k)\, y$

et effectuons les sommes membre à membre de $k = 1$ à $k = I7$.

Il vient : $2^{-17} r_{I7} = r_o + y \sum_{k=1}^{I7} 2^{-k} - y \sum_{k=1}^{I7} 2^{-(k-1)} z_k$

$$2^{-17} r_{I7} = r_o + y\,(1 - 2^{-I7}) - y \sum_{k=1}^{I7} 2^{-(k-1)} z_k$$

avec $r_o = x$ nous obtenons

$$x = y\,(-1 + 2^{-17} + \sum_{k=1}^{I7} 2^{-(k-1)} z_k) + 2^{-I7} r_{I7}$$

En posant $q = -1 + 2^{-I7} + \sum_{k=1}^{I7} 2^{-(k-1)} z_k$ et $r = r_{17} 2^{-I7}$

nous trouvons bien les deux nombres q et r demandés par les relations de définition.

* * *

SIMULATION SUR ORDINATEUR 650

Le programme qui permet d'interpréter les ordres TYKKA est construit comme une logique extérieure, c'est-à-dire, qu'il sert de maitre-programme vis à vis du langage TYKKA.

Le but principal était de reproduire exactement les opérations arithmétiques faites par TYKKA, pour se familiariser avec le calcul dans le système binaire, et la manipulation de nombres toujours inférieurs à l'unité, dans leur représentation. Nous avons, par contre, ignoré les entrées-sorties et les possibilités pupitre de TYKKA.

Pour que les essais effectués sur 650 soient profitables, nous avons joint à la simulation une analyse ordre par ordre du programme TYKKA.

REMARQUES GENERALES SUR LA SIMULATION

Enregistrement d'un ordre en mémoire

Dans le calculateur TYKKA un ordre occupe deux mémoires consécutives, mémoires à six positions octales. Nous avons conservé cette disposition en plaçant un ordre dans deux mémoires 650 consécutives, sous la forme :

00 00 CØ XMAP 0000 A1 A2 A3

Il aurait été possible de placer l'ordre en une seule mémoire 650 qui est à I0 positions décimales, mais le compteur d'instruction et les appels d'adresse auraient été totalement faussés.

La partie AI A2 A3 indique en général une adresse en mémoire, donnée dans le système octal qui est directement interprété par TYKKA alors que le 650 travaille dans le système décimal.

Nous n'avons pas converti en décimal la partie adresse, plus simplement nous appelons la mémoire tambour 650 qui porte le numéro d'adresse A1 A2 A3. Toutes les modifications d'adresse devront être calculées dans le système octal, en particulier l'indexage par le registre M.

Remarque : les adresses décimales numérotées avec les chiffres 8 ou 9 ne seront jamais appelées par le programme TYKKA, nous pourrons y placer une partie du programme de simulation.

Enregistrement d'un nombre en mémoire

TYKKA enregistre une valeur numérique dans une mémoire à I8 positions binaires. Pour conserver l'encombrement, les valeurs numériques sont placées sous la forme octale dans une mémoire 650 :

00 00 $n_1\ n_2\ n_3\ n_4\ n_5\ n_6$

Mais pour suivre fidèlement les procédés TYKKA, les opérations sont simulées d'après les règles indiquées au chapitre 2, elles opèrent donc sur des nombres de I8 digits binaires.

Pour effectuer une opération il faudra

I° convertir la donnée octale en binaire

2° opérer en binaire

3° convertir le résultat de binaire en octal.

Conversion octal-binaire

Pour convertir un nombre d'un système donné dans un autre, il faut utiliser deux méthodes différentes suivant qu'il est entier ou fractionnaire. Or les calculs d'adresse correspondent à des opérations sur nombres entiers, et les calculs arithmétiques se font sur des nombres fractionnaires, sans aucune distinction possible au moment d'exécuter l'ordre.

Nous avons du adopter une représentation octale symbolique, commune à toutes les expressions. Ce n'est plus l'équivalent du nombre binaire, dans le système octal, que nous enregistrerons, mais une expression qui le représente. Ceci en considérant la succession de I8 digits binaires comme un entier, les digits sont groupés 3 par 3 pour donner 6 nombres.

Exemples : - la donnée numérique

OIO OIO OOI IIO III IOO . . .

sera représentée par 22. I6. 74

alors que sa valeur exacte est :

$+ (0,\ IOO\ IOO\ OII\ IOI\ III\ 000)_2 \equiv + (0,\ 443570)_8$

- la donnée numérique

III OII OIO IOI IIO III

sera représentée par 7 3 2 5 6 7

alors que sa valeur exacte est :

$- (0,\ OOI\ OOI\ OIO\ IOO\ OIO\ OIO) \equiv - (0,II2422)_8$

- l'adresse

000 000 00I 0II I0I 0I0

sera représentée par 00. I3. 52.

mais ce nombre est entier et $(I\ 3\ 5\ 2)_8$ est aussi sa valeur exacte.

Le passage de l'écriture octale à l'écriture binaire n'est plus qu'une simple correspondance entre un chiffre octal et trois digits binaires.

Règle de dépassement arithmétique

Le calculateur TYKKA accepte seulement les nombres x qui vérifient les inégalités.

$$-1 \leq x < 1$$

La représentation binaire de - 1 est donnée par I00.000.....000. = $(40.00.00)_8$

Toute opération arithmétique dont le résultat n'est pas dans ces limites provoque un dépassement de capacité. Un commutateur du pupitre relatif au dépassement, peut prendre deux positions "arrêt" ou "détection".

Si le commutateur est sur "arrêt" : tout dépassement arithmétique provoque un arrêt-machine.

Si le commutateur est sur "détection" : l'arrêt est ignoré. Un indicateur de dépassement est enclenché et le calcul se poursuit (Nous verrons suivant les cas le contenu des registres ACC. MQ au moment d'un dépassement).

Une instruction de "saut si dépassement arithmétique" permet d'utiliser cette indication en fonction du programme déroulé. Une fois excité, l'indicateur ne peut être remis à zéro que par un tel ordre. Ce test ne porte donc pas uniquement sur l'opération qu'il suit immédiatement, mais sur toute opération du programme, qui le précède.

Nous avons simulé cet indicateur de dépassement en lui consacrant une mémoire appelée CTDEP :

si CTDEP = 0 il n'y a pas eu de dépassement

si CTDEP $\neq$ 0 il y a dépassement.

Double précision

Une partie des opérations arithmétiques est effectuée sur des nombres en double précision : la multiplication donne un résultat de 2 x I8 digits, la division accepte un dividende de 2 x I8 digits, de même les décalages en double précision. Donner ainsi, un nombre N a pour valeur $N = N_1 + N_2 2^{-17}$, placé dans les registres, on trouvera N_1 en ACC

N_2 en MQ

Mais N_2 est toujours considéré comme un nombre positif dont on ignore le digit d_o ceci pour que la succession des 35 digits donne une représentation homogène de N. Les poids successifs des digits sont :

ACC	MQ
d_o d_{-1} ..... d_{-17}	d_o d_{-1} d_{-2} ..... d_{-17}
2^{-1} 2^{-17}	2^{-18} 2^{-19} 2^{-35}

Les décalages en double précision doivent amener le digit d_{-1} de MQ à la place du digit d_{-17} de ACC et inversement.

Quand la multiplication donne un résultat significatif sur ACC - MQ, celui-ci est formé de 35 digits, les I7 de MQ sont complétés automatiquement par 0 en position d_o.

Analyse

Elle est construite au cours du décodage de chaque ordre, et perforée sur une carte si un signe (-) est placé au pupitre. Elle indique l'ordre que l'on vient d'exécuter, et le contenu des différents registres, après exécution. Disposition de la carte :

mot 1	mot 2	mot 3	mot 4	mot 5	mot 6	mot 7	mot 8
adresse de l'ordre	2 parties de l'ordre		An	contenu de An	contenu de ACC	contenu de MQ	contenu de TEST

Elle se place en mémoire dans la région PAS∅ = A000I ⟶ A008

Remarque

Les opérations TYKKA les plus fréquentes portent sur le contenu du registre ACC. Pour ne pas effectuer la conversion octal-binaire à chaque ordre, nous gardons en mémoire, d'un ordre sur l'autre, la traduction binaire de ACC : soit ACCI et ACC2.

SOUS-PROGRAMMES PRINCIPAUX

Les conversions octal-binaire et les quatre opérations ont été rédigées sous forme de sous-programme, chaque ordre y fera appel.

Les opérations se font sur des nombres de I8 digits qui seront toujours écrits dans deux mémoires sous la forme :

0 xxx xxx xxx / 0 xxx xxx xxx soit $N = n_1 n_2$

les digits I ou 0 ont été remplacés par les digits 8 ou 9, car le 650 ne peut tester, position par position, que ces deux chiffres.

Exemple : m = 0I0 I00 0II 0II III I0I

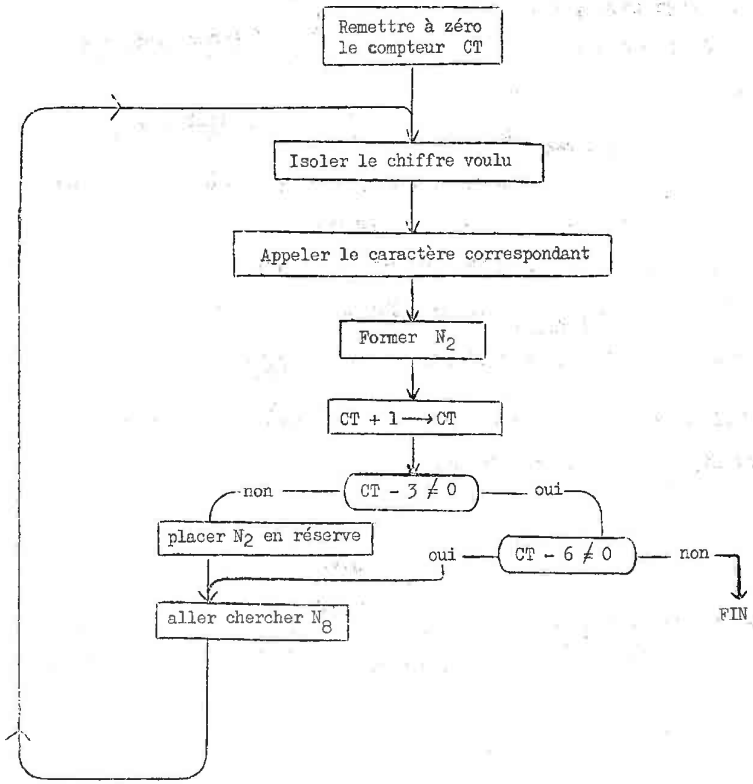
sera représenté par $m_1 m_2$ = 0 989 899 988 0 988 888 898

Nous noterons pour simplifier la rédaction : d8 : digit=8
d9 : digit=9

Les calculs se feront en deux phases successives : calcul de n_2, puis de n_1, ou inversement, ce qui alourdit toutes ces opérations.

Les entrées et sorties placent les données et les résultats en 8003-8002, les ordres de sortie en 800I, de façon à faire se succéder différents sous-programmes avec un minimum d'ordres intermédiaires. Dix mémoires de travail sont utilisées, notées de TR1 à TR10, les mémoires I995-I996 reçoivent les deux parties de la donnée n_1, n_2 .

OCTAL BINAIRE



CONVERSION OCTAL-BINAIRE

But Etant donné un nombre représenté par SIX chiffres octaux, chercher sa représentation dans le système binaire.

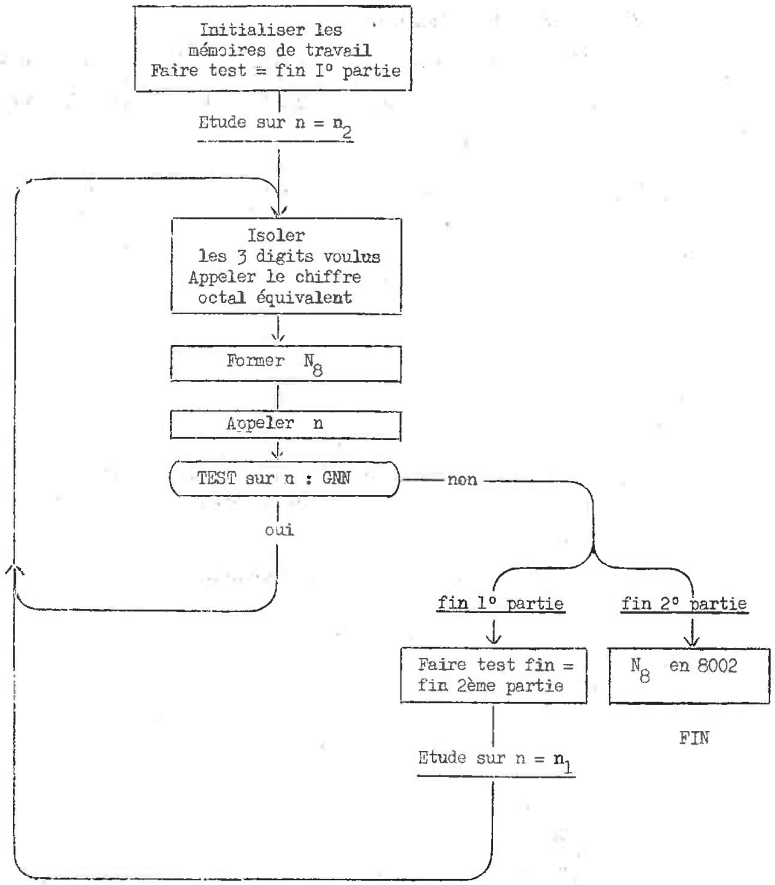
Méthode Un chiffre octal correspond à trois digits binaires. A partir de chacun des chiffres donnés, on construit un ordre d'appel des digits binaires équivalents avec lequels on forme N_2.

Utilisation	8003	8002	800I
données	00.....00	0000—N_8—	ordre fin
résultat	0—n_1—	0—n_2—	0—n_1—

PROGRAMME

```
ØCBI   TDI  TR8                  ordre fin  en TR8
       RAG  8002
       TRD  TR1                  remise à zéro du compteur : TR1
       TDI  1995  ØBA            I995 recevra la partie n du résultat N2
ØBA    DAD  0001
       SAG  8003
       TDI  AMXI
       DAD  0005                 on construit l'ordre : AAD  BOOn8
       AAD  CTØB
       TRD  TR2
       RAD  I995                 construction de N2
       DAD  0003  TR2
CTOB   AAD  BC000
       TRD  I995
       RAG  TR1
       AAG  CT1                  compteur  CT + 1
       TRG  TR1
       SAG  CT3
       CNN        ØB1            tests
       SAG  8001
       CNN  ØB3   ØB4
ØB3    RAG  AMXI  ØBA
ØB1    EDI  I995                 la première partie n2 est formée, elle se
       TDI  I996  ØB3            range en I996
ØB4    RAD  I996
       AAG  I995  TR8            le résultat est placé en 8003  8002     FIN
```

BINAIRE - OCTAL



CONVERSION BINAIRE-OCTAL

But : Etant donné un nombre $N_2 = n_1 n_2$ formé de I8 digits binaires, chercher sa représentation octale N_8.

Méthode Chaque groupe de trois digits appelle le chiffre octal qui doit le représenter. On construit alors N_8 avec six chiffres.

Utilisation		8003	8002	800I
	données	0—n_1—	0—n_2—	ordre fin
	résultats	00.....0	0000—N_8—	

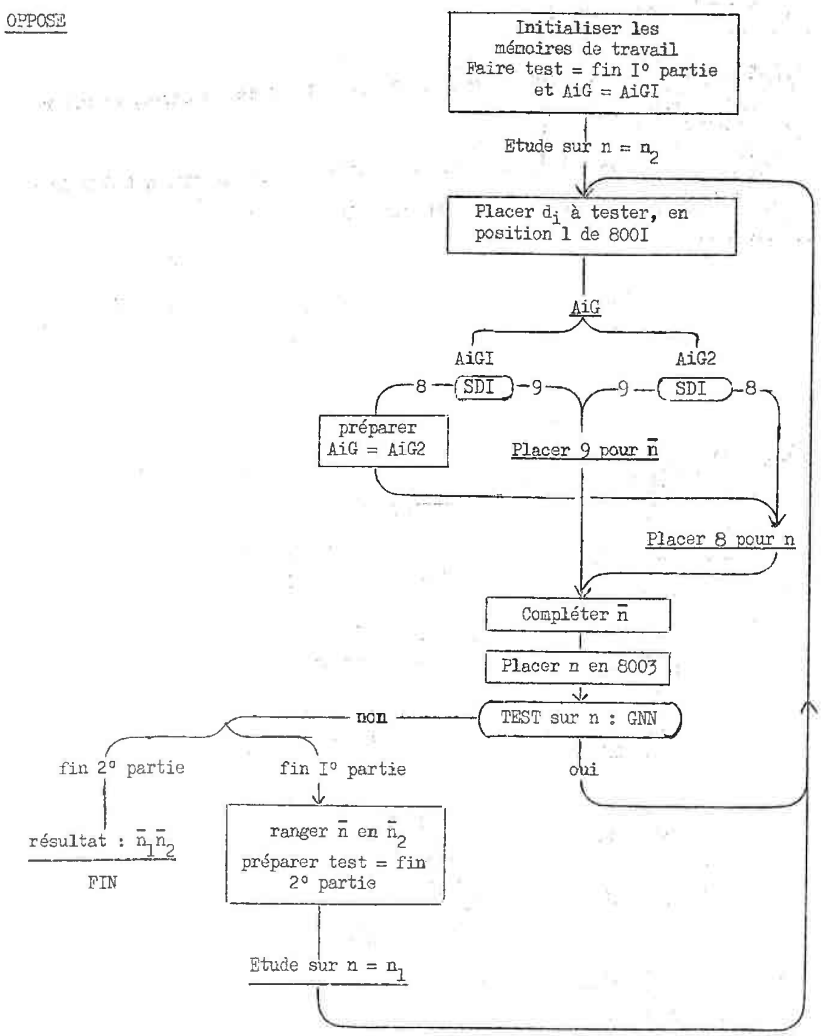
```
BIØC  TDI  TR8
      TRG   I995          données, en mémoire
      RAG   8002
      TRD  AMXI           AMXI reçoit le résultat
      EDI  BØTI   BØ
BØ    TDI  TRI    BØA     prépare test = fin I° partie

BØA   DAD   0003
      SAG   8003
      TDI   I996          isole le caractère boulu
      DAD   0003
      AAD  CTBØ
      TRD  TR2
      RAD  AMXI           construit l'équivalent N8
      DAD   00I  TR2
CTBØ  AAD   I000
      TRD  AMXI
      RAG   I996
      GNN  BØA    TRI     test fin ?

BØTI  RAG   I995          prépare n = n1
      EDI  BØT2   BØ
BØT2  RAD  AMXI   TR8     résultat

                          table des équivalences
 I999  00   0000   0000
 I998  00   I000   0000
 I989  00   2000   0000
 I988  00   3000   0000
 I899  00   4000   0000
 I898  00   5000   0000
 I889  00   6000   0000
 I888  00   7000   0000
```

OPPOSE



OPPOSE

But Etant donné un nombre binaire $N = n_1 n_2$, chercher l'opposé de ce nombre sous forme binaire $\bar{N} = \bar{n}_1 \bar{n}_2$.

Méthode La règle appliquée par TYKKA inverse tous les digits et ajoute 2^{-I7} au résultat obtenu. Ici l'addition de 2^{-I7} dans notre représentation formée de d8 et d9 n'était pas possible. Nous prenons la méthode suivante :

- avant la rencontre du premier digit 9 aucun digit 8 n'est à changer
- le premier digit 8 ne change pas
- après ce premier digit 8, il faut inverser tous les digits rencontrés.

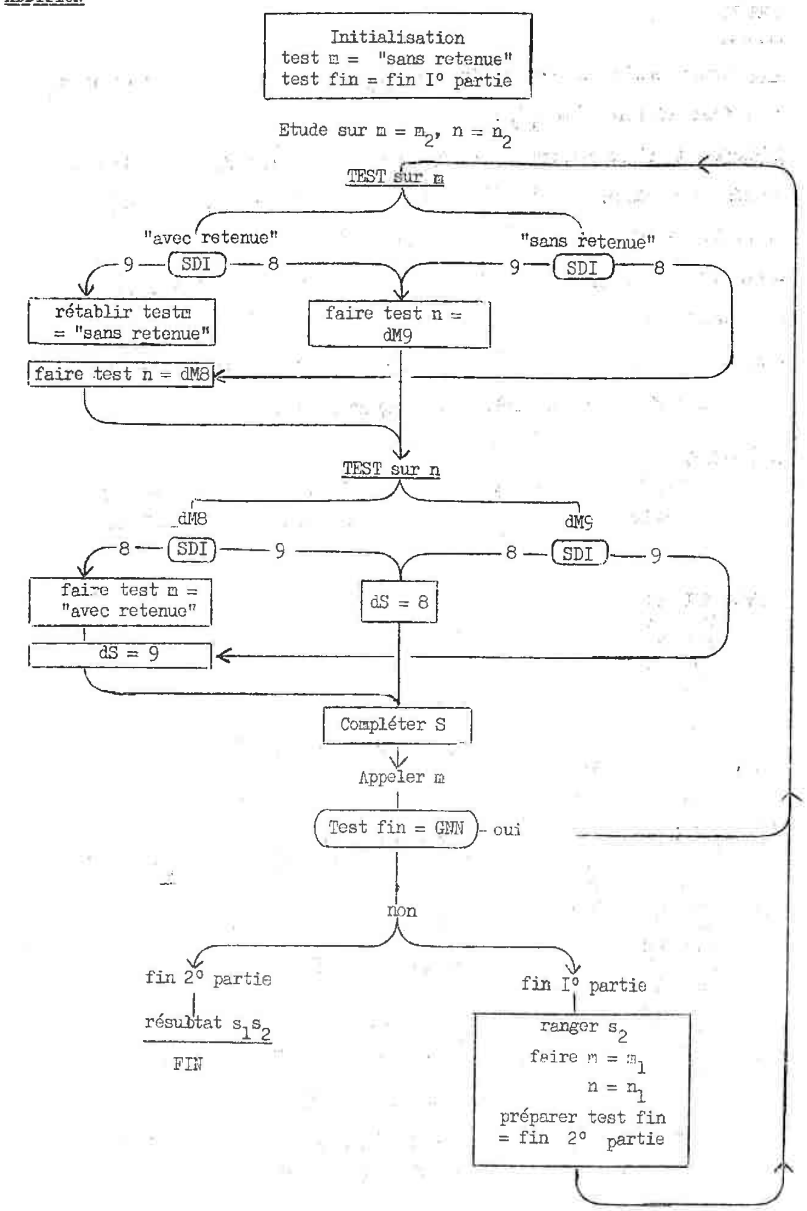
N est construit digit par digit en commençant par d_{-17}

Utilisation	8003	8002	800I
données	0—n_1———	0—n_2———	ordre fin
résultats	0—$\bar{n}_1$———	0—$\bar{n}_2$———	0—$\bar{n}_1$———

```
ØPPØS  TDI  TR8                ordre fin en TR8
       TRG   I995              n1 en I995
       EDI  SDI
       TDI  TR5                AiG = AiGI
       EDI  ØP3
       TDI  TR4                test = fin I° partie
       RAD   8002
       TRG  TR3                TR3 recevra le résultat n̄
       EDI   8002  ØPI

ØPI    DAD   000I  TR5         test sur n
SDI    SDI  SD8    SD9
SD8    EDI  SDII               modifier le test pour
       TDI  TR5    ØP8         inverser tous les digits
ØP8    AAG  CT8    ØP2
SD9    AAG  CT9    ØP2
ØP2    TRD  TR2                n en TR2
       AAG  TR3
       DAD   000I              construction de n̄
       TRG  TR3
       RAD  TR2
       ANN  ØPI    TR4         test fin ?
ØP3    EDI  TR3                la première partie n̄2 est terminée
       TDI   I996
       EDI  ØP4                changer le test fin
       TDI  TR4
       RAD   I995  ØPI         poursuivre avec n1=n
ØP4    RAD   I996              la partie n̄2 est terminée
       AAG  TR3    TR8
```

ADDITION



ADDITION

But étant donnés deux nombres binaires $N = n_1 n_2$, $M = m_1 m_2$, chercher leur somme $S = s_1 s_2$, dans le système binaire.

Méthode on applique directement la règle arithmétique TYKKA. La somme est construite digit par digit en commençant par d_{-17}. Les tests se font en parallèle sur les digits de même rang de N et M d'où on déduit S :

Nd9	+	Md9	⟶ Sd9	sans retenue
Nd8	+	Nd9	⟶ Sd8	- ---
Nd9	+	Md8	⟶ Sd8	- ---
Nd8	+	Md8	⟶ Sd8	avec une retenue

La retenue se traduit par une modification des tests sur M et N, car il n'est pas possible d'effectuer l'addition de 1 à M au rang voulu.

Dans les tests avec retenue :

Md9 devient équivalent à un digit 8
mais le test de retenue est à supprimer

Md8 devient équivalent à un digit 9
mais la retenue est reportée au rang suivant.

Utilisation

données $N = n_1 n_2$ en TR6 TR7
$M = m_1 m_2$ dans les accumulateurs

	800 3	800 2	800 I
sous la forme	0---m_1---	0---m_2---	ordre fin
résultat	0---s_1---	0---s_2---	0---s_2---

```
ADDIT  TDI  TR8                ordre fin
       TRG   I995              m1          en mémoire
       EDI  SADI
       TDI  TR9                prépare test "sans retenue"
       RAG   8002
       TRD  TR5
       EDI  ADGI               préparer fin = fin I° partie
       TDI  TRI
       EDI   8003  AD3
```

```
AD3    DAD   000I  TR9      test sur m
SADI   SDI   SAD8  SAD9
SAD8   EDI   S8D   AD2      préparer le test à faire
SAD9   EDI   S9D   AD2      sur n
AD2    TDI   TR4
       TRG   TR3           range m en TR 3

       RAD   TR7           test sur n
       DAD   000I  TR4
S9D    SDI   SD98  SD99
SD98   AAG   CT8   AD4
SD99   AAG   CT9   AD4
AD4    TRD   TR7
       AAG   TR5           construit s
       DAD   000I
       TRG   TR5
       RAG   TR3
       GNN   AD3   TRI     test fin

ADGI   EDI   TR6           fin I° partie, s1 calculée
       TDI   TR7
       EDI   TR5           placer n2 et m2 en
       TDI   I996          mémoire
       EDI   ADG2          préparer test = fin 2° partie
       TDI   TRI   SD99
       RAG   I995  AD3
ADG2   RAG   TR5
       AAD   I996  TR8     résultat fin

SDIR   SDI   SAD9           test "avec retenue"
       EDI   SADI
       TDI   TR9   SAD8
S8D    SDI   SD88  SD98
SD88   EDI   SDIR
       TDI   TR9   SD99
```


MULTIPLICATION

But Soit à calculer le produit du contenu MQ par le contenu de la mémoire Am = a. Le résultat et demandé en double précision sur ACC-MQ.

Méthode Elle correspond à l'application de la règle arithmétique exposée au chapitre 2. On suppose MQ donné sous la forme $M' = d'_1\, d_o d_{-1} \dots d_{-I7}$, la représentation $m = e_o\, e_{-1} \dots\dots e_{-17}$ définie précédemment ne sera pas explicitement formée. Les produits partiels sont construits dans ACC en double précision.

L'opération s'accomplit en I8 phases successives semblables. Chaque phase teste les trois valeurs $d_{-(j-1)}$, d_{-j}, r_{-j}, dont on déduit les valeurs e_{-j} et r

Suivant la valeur trouvée pour e_{-j}, le multiplicande (Am) sera ajouté au produit partiel ($e_{-j} = +1$), soustrait ($e_{-j} = -1$), ou on passera directement à la phase suivante ($e_{-j}=0$).

Après l'opération (ACC) $\pm$ (Am) le résultat subit un décalage arithmétique en double précision à droite, avant de passer à la phase suivante.

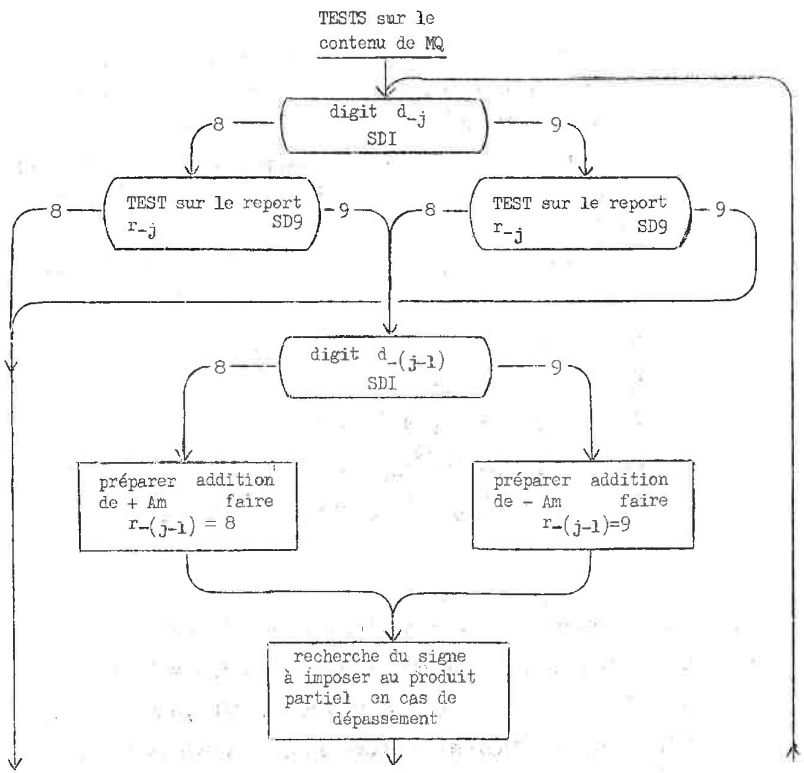
Au cours des I8 phases nous avons formé successivement les produits partiels.

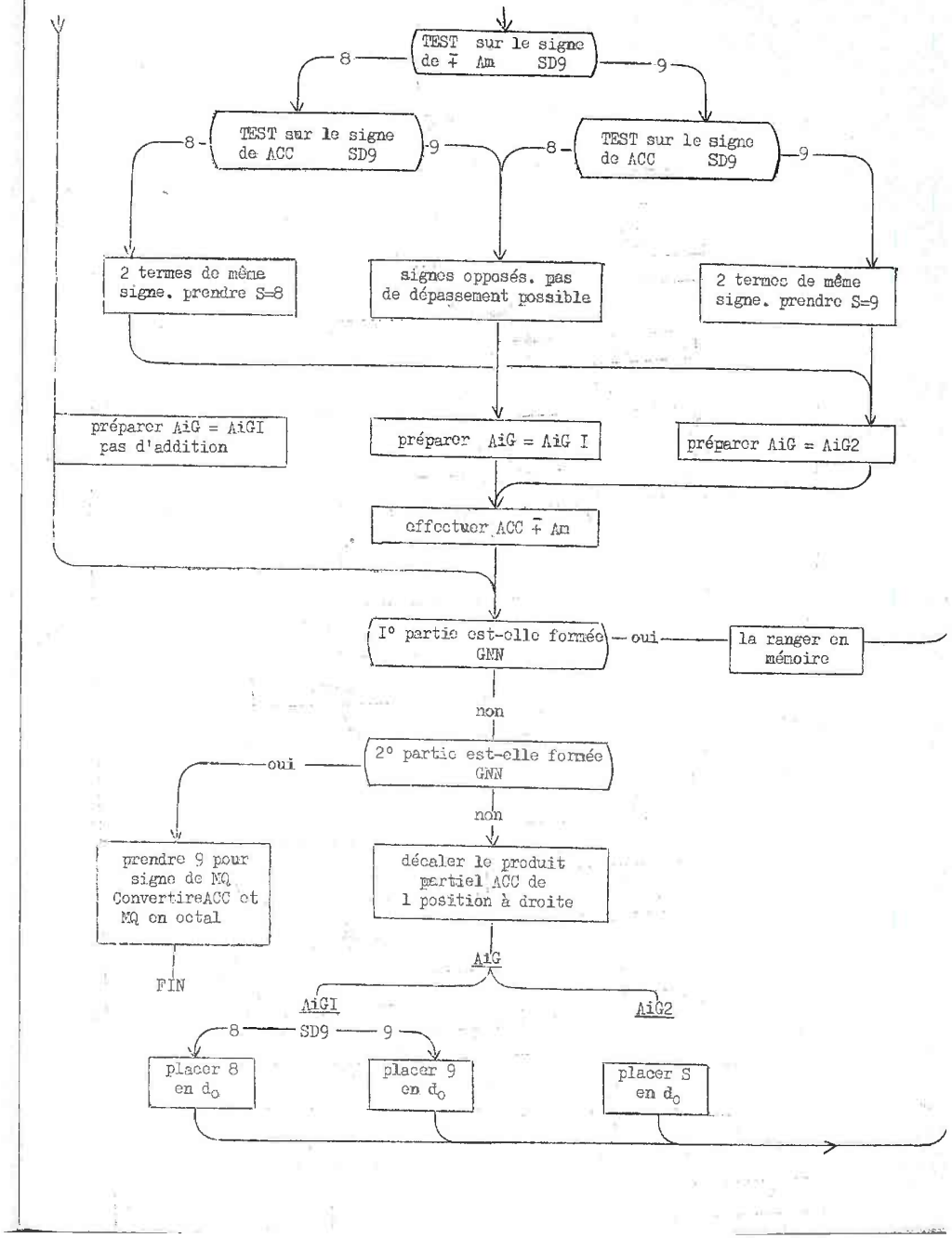
$$P_1 = a\ e_{-17}$$
$$P_2 = a\ (e_{-16} + 2^{-1} e_{-17})$$
$$P_3 = a\ (e_{-15} + 2^{-1} e_{-I6} + 2^{-2} e_{-17})$$
$$P_k = a\ (e_{-(I8-k)} + 2^{-1} e_{-(I8-k-1)} + \dots\dots + 2^{-(k-1)} e_{-17})$$
$$P_{I8} = a\ (e_o + 2^{-1} e_{-1} + 2^{-2} e_{-2} + \dots\dots\dots\dots + 2^{-I7} e_{-17})$$

D'après l'exposé arithmétique P_{18} représente exactement le produit a m. Nous obtenons directement la représentation du produit voulu, quels que soient les signes des données, car les opérations de base : addition, soustraction, décalage arithmétique, sont elles-mêmes effectuées sur les représentations.

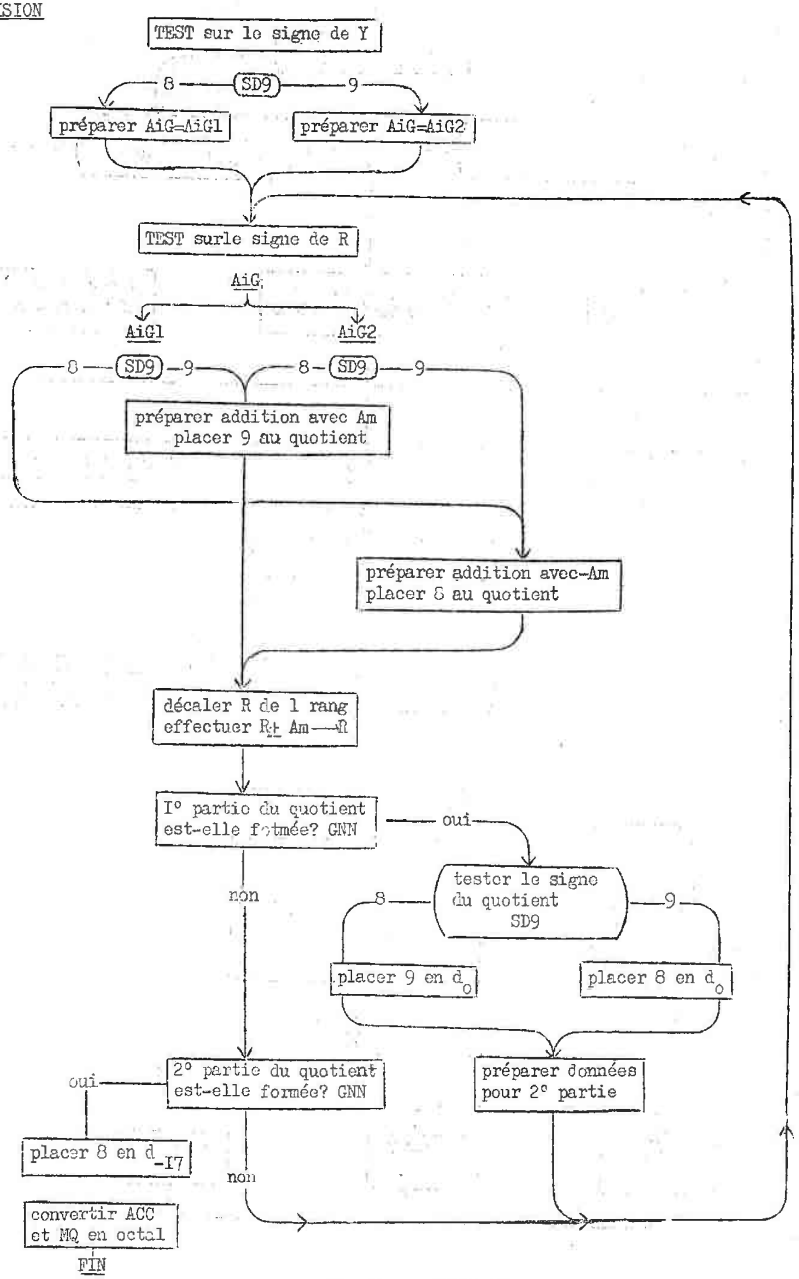
Dépassement : le produit de deux nombres inférieurs à l'unité ne peut donner lieu à aucun dépassement véritable. Cependant il se peut qu'un produit partiel P_k soit supérieur à l'unité, cette valeur étant rectifiée par la phase suivante.
Au cours de l'addition (ACC) $\pm$ (Am), il ne faut pas tester le dépassement, le digit d_o fait alors partie de la valeur numérique du résultat, c'est le décalage à droite qui suit qui devra placer le digit signe voulu en d_o.

Un cas de dépassement est cependant signalé : le produit -1 par -1 donnerait + 1 en résultat, or + 1 n'est pas représenté dans TYKKA.





DIVISION



DIVISION

<u>But</u> calculer q - quotient et r - reste , de la division du nombre contenu dans MQ, par le contenu de Am placé en ACC initialement. On donnera le quotient en MQ, le reste en ACC.

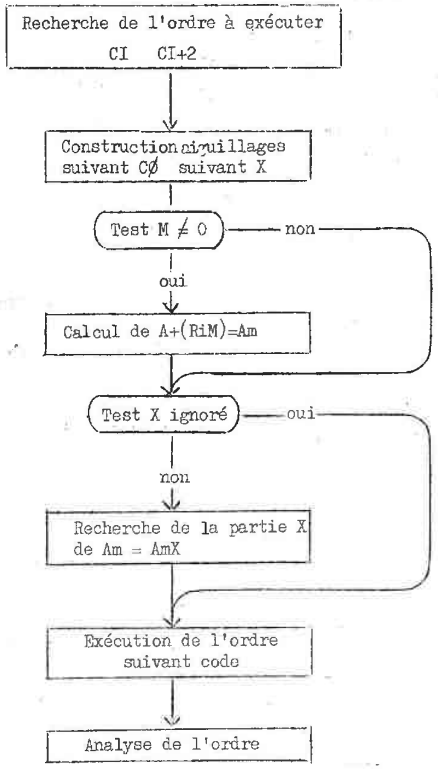
<u>Méthode</u> il résulte directement de la règle exposée au chapitre 2. Le quotient est formé chiffre par chiffre à partir de d_o, au cours de I7 phases successives.

En premier, le signe de y est testé puisqu'il est une constante de l'opération.
Chaque phase teste le signe du reste partiel r_{k-1}, contenu dans ACC, d'où elle déduit e_k et z_k - z_k est introduit dans MQ pour compléter le quotient. Puis r_{k-1} est décalé d'un rang à gauche. (multiplication par 2) et, si $e_k = 1$ on lui ajoute +y, si $e_k = -1$ on lui ajoute -y.

Après l'exécution de I7 phases semblables, ACC contient le reste r, et MQ contient $\sum_{k=1}^{I7} 2^{-(k-1)} z_k$, il faut ajouter $(-1 + 2^{-I7})$ pour obtenir la valeur exacte de q. Pour obtenir la représentation de q, il faut d'après nos conventions, ajouter 2^1 si q est négatif, soit au total ajouter $2 - 1 + 2^{-I7} = 1 + 2^{-I7}$. Si q est positif cette addition de 2, effectuée modulo 2, ne peut pas modifier se résultat, donc : on ajoutera toujours $(1 + 2^{-I7})$ à MQ. Le dernier digit introduit dans MQ l'a été à l'ordre (-I6), nous placerons donc MQ $d_{-I7} = 1$ et un test sera fait sur d_o de MQ pour additionner 1 à ce résultat :

si $d_o = 0$ nous prendrons $d_o = 1$
si $d_o = 1$ - --- $d_o = 0$

ORGANIGRAMME GENERAL



PARTIE COMMUNE

Elle est nécessaire avant l'exécution d'un ordre particulier, et a pour rôle de :

rechercher l'ordre à l'adresse voulue,
ajouter deux au comptour d'instruction CI,
isoler les différentes parties de cet ordre et préparer les constantes ou aiguillages qui seront nécessaires,
calculer l'adresse Am = A1A2A2 + (CIM) si M ≠ 0

Le calcul de CI + 2 devrait se faire en octal. Nous n'avons pas voulu passer en binaire, ni introduire une véritable table d'addition octale pour ce seul cas. Nous avons supposé l'adresse CI toujours paire, et commandé l'addition de 2 dans le système décimal. Des tests permettent de modifier le résultat obtenu si des chiffres 9 ou 8 sont apparus.

Les aiguillages suivant CØ et X sont formés à l'aide d'adresses régionales.

```
CITAC  RAG  CI
       TDI  A000I                    adresse de l'ordre pour analyse : N
       DAG   000I
       AAG        8003
       EDI   0000  S000I             exécuter :  EDI   (N)   S000I
S000I  TDI  A0002                    I° partie de l'ordre
       AAG        8003               exécuter :  EDI  (N+I)  S0002
        00   000I  8001              2° partie
S0002  TDI  A0003
       RAG  CI
.............................        calcul de CI + 2.
       TRG  CI     I006
       RAG  A0002                    I° partie de l'ordre à détailler
       DAD   0004
       GNM         P0000             si CØ = 00  ordre d'arrêt
       AAG  CTP                      si CØ ≠ 00  aiguillage en fonction
       SAG   8003
       TDI  DEBUT                    DEBUT = 00 0000 P000Ø
       DAG   000I
       AAG  CTX                      aiguillage suivant X
       TRG  MEMX                     MEMX = 00 0000 X000x
       AAG  CTXI
       SAG   8003
       TDI  MEMXI                    MEMXI = 00 0000 X00Ix
       DAG   000I
       GNM         NIND              si M = 0  ne pas indexer.
```



```
       DAD   0006                    si M≠0   calculer Am
       EDI  MEMMM
       SBF  MEMMM                    MEMMM =  RAD  M000m  ....
       DAG   0006                    M000m  adresse du registre d'indice n°M
       SAG   8003
       DAD   0004
       TRD  APRIM  INDE              APRIM = 00 00AP  0000
INDE   RAD  A0003
       EDI         ØCBI
       TRD  TR7                      calcul en binaire-octal de
       TRG  TR6    MEMMM
MEMMM  RAD  M0000                    A1A2A2  +  (M000m)  =  Am
       EDI         ØCBI
       EDI         ADDIT
       EDI  FIND   BIØC
NIND   DAD   0004
       TRD  APRIM
       RAD  A0003  FIND

FIND   TDI  AQ004                    Am  en mot 4 d'analyse
       TRD  RADRS                        en registre adresse sous la forme
       DAG   0004                        00 0000 Am
       TRD  RADR4                    prépare Am sous la forme  00 Am 0000
       EDI  FX                       construction des ordres
       SBF  FX                       TDI   Am  FIN
       EDI  MEMXI
       SBF  MEMXI  TEST              RAD   Am  X00Ix
```

FIN : adresse des ordres test sur 8000 pour commander ou non la perforation d'une carte analyse, et retour à CITAC pour exécuter l'ordre suivant.

La carte d'analyse est complétée par appel du contenu de Am. Opération qu'il faut éviter quand A1A2A3 n'est pas l'adresse d'une mémoire (décalages). Pour ces ordres le mot 5 de l'analyse est à ignorer.

```
FIN    RAD  RADR4
       AAD         8002
       EDI   0000                    appel du contenu de Am
       TDI  A0005  PF4
PF4    RAG   8000
       ANG         CITAC             test pupitre
       PFØ  A000I  CITAC
```

L'appel de l'ordre à exécuter est effectué avant le calcul de CI+2. Ceci pour permettre l'utilisation au pupitre du 650, du commutateur "arrêt prédéterminé" pour les ordres du programme TYKKA. En particulier si on désire obtenir l'analyse d'une partie du programme.

Interprétation de la partie X de l'ordre

Les ordres qui tiennent compte de X peuvent être rangés en deux catégories suivant l'interprétation qu'ils donnent à cet indicatif.

1 - Les ordres qui appellent le contenu d'une mémoire An pour le transférer vers un registre : registre arithmétique, ACC ou MQ, ou registres d'indices.

Seule la partie indiquée par X est extraite de la mémoire An et complétée par des zéros de manière à former un mot-machine. L'opération demandée est alors effectuée sur ce nouveau mot, noté AMXI. Si X appelle un seul caractère, soit X = 1, 2 ou 4, celui-ci est complété par 4 zéros à gauche, quelque soit la place qu'il occupait en mémoire.

exemple	si	AM	= a1 a2 a3
	X = 1	AMXI	= 00 00 a3
	X = 2	AMXI	= 00 00 a2
	X = 4	AMXI	= 00 00 a1

Si X appelle deux caractères, ils sont complétés par 00 à droite. Seul le groupe II0 est accepté.

exemple X = 6 AMXI = a1 a2 00

2 - Les ordres qui enregistrent en mémoire un résultat pris dans ces mêmes registres : ACC, MQ ou les registres d'indices.

Si X appelle un seul caractère. C'est le caractère le plus à droite dans le registre qui est considéré et transmis vers la mémoire. Mais il occupera, dans An, la place exacte indiquée par X, les autres caractères de An ne subissent aucune modification.

exemple : enregistrement de ACC en An pour X = 2

$(ACC)_d$ = C1C2C3

$(An)_d$ = a1a2a3 $\longrightarrow$ $(An)_f$ = a1C3a3

Si X appelle deux caractères c'est le groupe CIC2 qui est transmis vers An.

exemple : enregistrement de ACC en An pour X = 6

$(ACC)_d$ = CIC2C3

$(An)_d$ = a1a2a3 $\longrightarrow$ $(An)_f$ = CIC2a3

Simulation

Une suite de tests élimine les ordres qui ignorent X. Pour tous les ordres qui interprétent X on siole le ou les caractères demandés sous deux formes.

exemple : pour X = 2 AMXI du type 00 00 a2
AMX du type 00 a2 00

Les ordres de la classe 1 utiliseront AMX.

Les ordres de la classe 2 isolent , en premier, la partie intéressée de ACC : ACCX, puis construisent $(An)_f$ donné par

$$(An)_f = (An)_d - AMX + ACCX$$

```
TEST   RAG  DEBUT                    MEMXI = RAD  An  X00Ix
       SAG  CTI5
       ANG  MEMXI
       SAG  CTI3                     L'adresse régionale a été construite en ajoutant
       ANG  DEBUT                    à l'indicatif X de l'ordre la constante
       SAG  CT7                      X00I0
       ANG  MEMXI
       SAG  CT3
       ANG  DEBUT  MEMXI

X00II  DAG   0008                    pour X = 1
       SAG   8003
       DAD   0006  X00I7             00 00 00 00 a1 = AMXI = AMX

X00I2  DAG   0006                    pour X = 2
       SAG   8003
       DAD   0008
       TRD  AMXI                     00 00 00 00 a2 = AMXI
       DAG   0002  SX

X00I4  DAD   0004                    pour X = 4
       TRD  AMXI                     00 00 00 00 a3 = AMXI
       DAG   0004  SX                00 00 a3 00 00 = AMX

X00I6  DAD   0002
       DAG   0002  X00I7             00 00 a1 a2 00 = AMXI = AMX

X00I7  TRD  AMXI   SX                pour X = 7
SX     TRD  AMX    DEBUT             00 00 ai a2 a3 = AMXI = AMX
```

ORDRES TYKKA SIMULES

Ordres arithmétiques

01	enregistrement de ACC en mémoire
02	inscription dans MQ
03	enregistrement de MQ en mémoire
04	effacement et addition dans ACC
05	effacement et soustraction dans ACC
06	addition sans effacement dans ACC
07	soustraction sans effacement dans ACC
10	multiplication
11	division en simple précision
12	division en double précision
13	somme logique
14	produit logique
15	décalage à droite logique
16	décalage à gauche logique
17	décalage à droite arithmétique
20	décalage à gauche arithmétique
21	décalage à droite double précision, arithmétique.
22	décalage à gauche double précision, arithmétique.

Ordres de saut relatifs à ACC

23	saut si (ACC) $\geqslant 0$
24	saut si (ACC) > 0
25	saut si (ACC) $= 0$
26	saut si (ACC) $\neq 0$
27	saut si dépassement arithmétique

Ordres relatifs aux registres d'indices

30	inscription dans RiA'
3I	enregistrement de RiA' en mémoire
32	addition, sans effacement, dans RiA'
33	soustraction, sans effacement, dans RiA'
34	comparaison

Ordres de sauts relatifs au registre Test

35	saut si $(T) \geqslant 0$
36	saut si $(T) > 0$
37	saut inconditionnel

Les ordres d'entrée et sortie et les tests intérieurs ne sont pas cités ici. Ils n'ont pas été simulés, car les organes extérieurs de TYKKA sont totalement différents de ceux de 650.

CØ = OI
Enregistrement de ACC

X : interprété
Am : adresse de la mémoire où enregistrer ACC
A' : ignoré

opérations
- la partie spécifiée par X est extraite de ACC, d'où ACCX.
- elle est ensuite transmise à la mémoire Am pour s'inscrire à la place indiquée par X.
- ACC reste inchangé au cours de l'opération.

CØ = 02
Inscription dans MQ

X : interprété
Am : adresse de la mémoire où est inscrit le nombre à placer dans MQ
A' ; ignoré

opérations
- la partie de Am spécifiée par X est extraite de la mémoire, puis transmis au registre MQ en position X. Cette inscription efface le précédent contenu de MQ

CØ = 03
Enregistrement de MQ

X : interprété
Am : adresse de la mémoire où enregistrer MQ
A' : ignoré

opérations
- la partie spécifiée par X est extraite de MQ et transférée vers la mémoire Am. Puis MQ est remis à zéro.

Pour les deux ordres, OI et 03, l'interprétation de X nécessite le calcul de la partie intéressée de ACC et MQ. Ce calcul est commandé par un aiguillage préparé dans la partie commune.

```
P0001  RAG  ACC   SOSOI
SOSOI  DAD  0002  MEMX        MEMX = 00 0000 X000x
X000I  SAG  8003              x = 1
       DAD  0008  SX0         00 0000 00C3
SX0    TRD  TR1   SX2         ACCX  est envoyé en TR1
X0002  SAG  8003              x = 2
       DAD  0006  SX0         00 0000 C300
X0004  SAG  8003              x = 4
       DAD  0004  SX0         00 00C30000
X0006  SAD  8002              x = 6
       DAG  0002  SXI         00 00 C1 C2 00
SX1    TRG  TR1   SX2
X0007  DAG  0002  SX1         00 C1 C2 C3
SX2    RAD  RADR4             calcul de (Am)f = (Am) - AMX + ACCX
       AAD        8002        8002 =  RAD  AM  ....
       RAD  0000
       SAD  AMX
       AAD  TR1   FX          FX   =  TRD  AM  FIN
P0002  EDI  AMXI
       TDI  MQ    FIN
P0003  RAG  MQ    SOSOI       MQ sera transmis vers la mémoire
                              Am par le même processus que pour ACC
```

CØ = 04

Effacement et addition dans ACC

X : interprété

Am : adresse en mémoire d'où est extrait le nombre à additionner.

A' : ignoré

opérations

- la partie Amx de Am est extraite de la mémoire.
- elle est ensuite inscrite dans ACC après effacement du registre.

CØ = 05

Effacement et soustraction dans ACC

X : interprété

Am : adresse en mémoire de la donnée

A' : ignoré

opérations

- la partie Amx est extraite de la mémoire. L'opposé de Amx est inscrit dans ACC après effacement de ce dernier.


```
P0004  RAD  AMXI
       TRD  ACC                la nouvelle valeur de ACC est convertie
       EDI        ØCBI         en binaire
       TRD  ACC2
       TRD  ACCI  FIN

P0005  RAD  AMXI
       EDI        ØCBI         prendre l'opposé de AmX
       EDI  POP05 ØPPØS

POP05  TRG  ACCI
       TRD  ACC2
       EDI  PIP05 BIØC         le convertir en octal

PIP05  TRD  ACC   FIN
```

CØ = 06

Addition à ACC

X : interprété

Am : adresse en mémoire de la donnée à additionner

A' : ignoré

opérations

- la partie AmX est extraite de la mémoire Am.
- l'additionneur effectue alors la somme : AmX + contenu total de ACC.
- le résultat est transmis à ACC.

dépassement

Si le résultat de l'addition est supérieur à l'unité, en valeur absolue, il y a dépassement de capacité. L'indicateur de dépassement est excité. Le résultat obtenu est cependant transmis à ACC même s'il peut paraitre erronné.

CØ = 07

Soustraction sur ACC

X : interprété

Am : adresse en mémoire de la donnée à soustraire

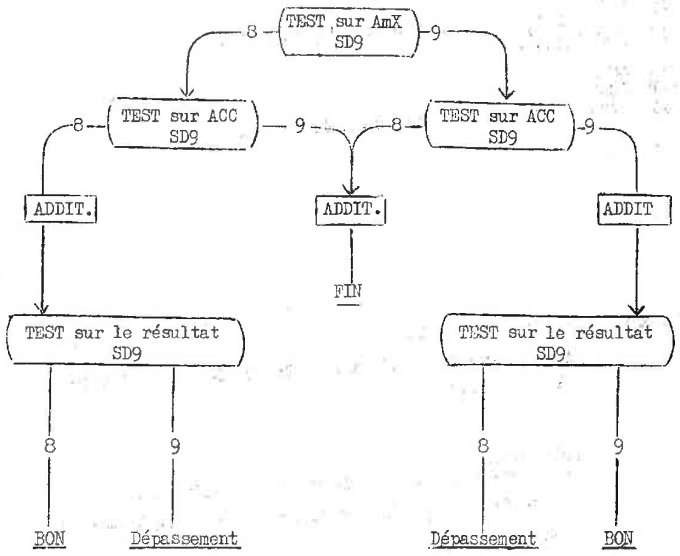
A' : ignoré

opérations

- la partie AmX de la mémoire Am est soustraite du contenu de ACC. Le résultat est réinscrit en ACC.

dépassement
Les conditions sont identiques à celles de l'ordre 06.

Tests de dépassement en addition




```
P0006  RAD  AMXI
       EDI  POP06  ØCBI
POP06  TRD  TR7
       TRG  TR6
       SD9         TST9        test sur le signe des données
       RAG  ACCI
       SD9  TST88  TST99
TST88  AAD  ACC2
       EDI         ADDIT       commande de l'addition
       SD9  POP05  DEPCA       test sur le résultat
TST99  AAD  ACC2
       EDI         ADDIT       commande de l'addition
       SD9  DEPCA  POP05       test sur le résultat
TST89  AAD  ACC2
       EDI  POP05  ADDIT
P0007  RAD  AMXI
       EDI         ØCBI
       EDI  POP06  ØPPØS       prendre l'opposé de AMX puis faire une addition
```


CØ = IO
Multiplication

X : interprété
Am : adresse en mémoire du multiplicande
A' : ignoré

opération
- AmX est extrait de la mémoire. On construit alors le produit de AmX par le contenu de MQ. Le résultat en double précision est placé dans les registres ACC et MQ.
- le contenu initial de ACC est ignoré, il n'est pas utile que ACC soit nul.

dépassement
Seul le cas : MQ = AmX = - 1, peut provoquer un dépassement car le résultat serait +I. L'indicateur de dépassement est enclenché. Le résultat, sous la forme octale $(40.00.00)_8$, est transmis aux registres ACC et MQ.

```
P0010  RAD  AMX
       SAD  04000              AMX est-il égal à (-1) ?
       ANN  SISIO
       RSD  800I
       AAD  MQ                 si oui : MQ est-il égal à (-1) ?
       ANN  SISIO
       ART  00I0
       TDI  ACC                si oui : il y a dépassement
       TDI  CTDEP
       TRD  MQ                 exciter l'indicateur de dépassement
       EDI  89999
       TDI  ACCI
       EDI  99999              placer le résultat en ACC.
       TDI  ACC2     FIN
SISIO  RAD  MQ
       EDI         ØCBI        construire la représentation de MQ
       SD9         MM9
       AAG  CT8    MM          M = d'1 d0 d-1 d-2 ........ d-17
MM9    AAG  CT9    MM
MM     TRG  MQI
       DAG  000I
       TRD  MQ2      SOSIO
SOSIO  RAD  AMX
       EDI         ØCBI        préparer les valeurs binaires de +AMX et -AMX
       TRD  AM2                utilisées par le sous programme de multiplication
       TRG  AMI
       EDI         ØPPØS       s'adresser au calcul du produit.
       TRD  AMO2
       TRG  AMOI   ML
```

CØ = 11
Division en simple précision

X : interprété
Am : adresse, en mémoire, du diviseur
A' : ignoré

opération
Le contenu de ACC forme le dividende, le contenu de Am le diviseur. Si la division est possible le résultat nous donnera : le quotient "q" dans le registre MQ
le reste "r" dans le registre ACC sous la forme $r2^{I7}$

Au départ, le contenu de MQ est ignoré, il sera effacé par le résultat de la division.

Remarques : dans cette opération MQ prend alors un signe propre, indépendant de ACC. D'autre part le reste, contenu de ACC, n'a pas obligatoirement le signe du dividende. Le quotient est obtenu en effet à 2^{-I7} près, mais par excès ou par défaut. Cependant r et q sont tels que la relation de définition reste vérifiée :

$$x = qy + r \quad \text{avec} \quad 1 < |Q| \leqslant 2^{-I7} \quad \text{et} \quad |r| \leqslant |y| \; 2^{-I7}$$

dépassement
Avant l'exécution de la division a lieu une phase de comparaison de x à y. Si la condition $|y| \geqslant |x|$ est réalisée la division est commandée. Si l'on a $|y| < |x|$ il y a dépassement de capacité, l'indicateur de dépassement est enclenché, mais la division n'est pas effectuée, les registres ACC et MQ restent inchangés.

CØ = I2
Division en double précision

X : interprété
Am : adresse, en mémoire, du diviseur
A' : ignoré

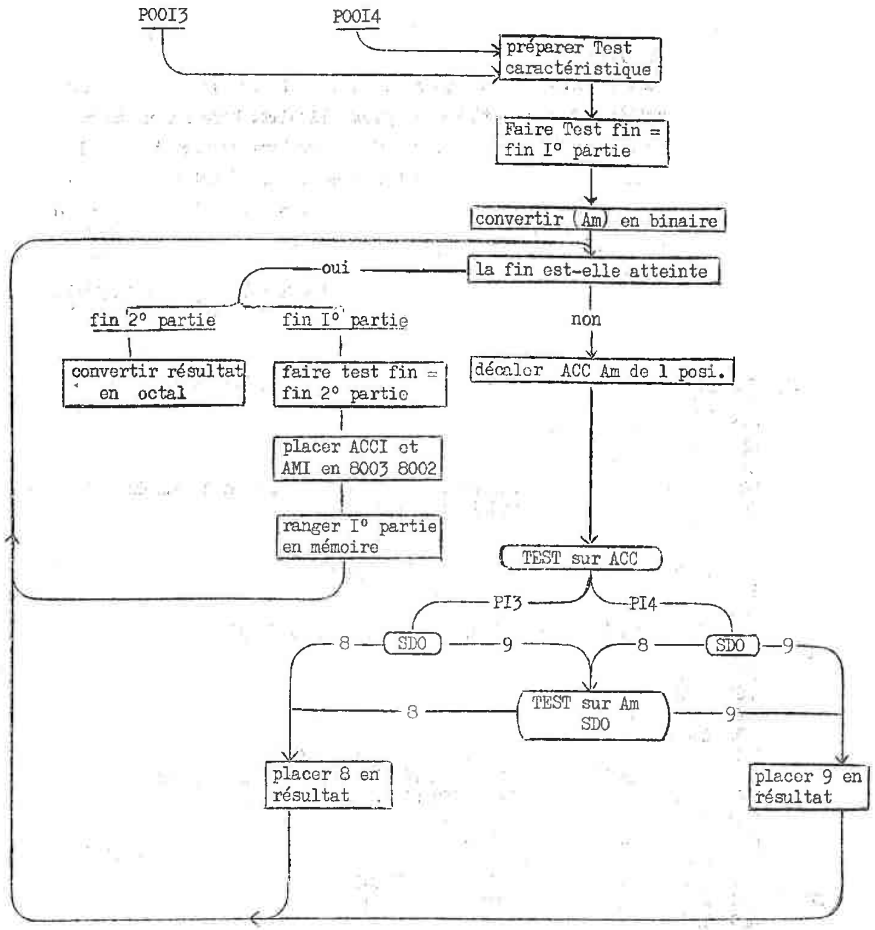
opérations
Le dividende est ici formé par le nombre en double précision placé dans les registres ACC et MQ, MQ est donc considéré com-

me positif. La disposition des résultats est identique à celle de l'ordre 11.

dépassement
La condition de validité des données x et y est alors
$|y| > |x|$
Si $|y| = |x|$ il y a dépassement de capacité.

programme
Test de dépassement : pour calculer $|x|$ et $|y|$ nous devons utiliser les opérations du système binaire. Nous ferons donc la comparaison de ces deux nombres sur leur représentation binaire, soient $R|y|$ et $R|x|$. Sachant que do est représenté par d9, et d1 par d8, la condition $|y| \geqslant |x|$ devient $R\,|x| \geqslant R|y|$
Si $R\,|x| = R\,|y|$ - en simple précision il faut effectuer le quotient
- en double précision il y a dépassement.

```
P0OII  EDI  DPCII                 préparer la réponse au test de dépassement
       TDI  DPC
       RAD  9999   DIVSD
       DAG   000I
       AAG   800I
       DAG   000I                 MQ est considéré comme nul pour la simple
       TRD  MQ2                   précision
       DAG   000I
       TRG  MQI
       RAD  ACC2
       AAG  ACCI
       SD9         VX
       EDI  VX     ØPPØS          valeur absolue de x en TR6 TR7
VX     TRG  TR6
       TRD  TR7
       RAD  AMXI
       EDI         ØCBI
       TRD  AM2
       TRD  AMI                   valeur absolue de y = AMX
       EDI         ØPPØS          mettre en mémoire +AMX et-AMX  pour nécessité
       TRD  AMO2                  de la division.
       TRD  AMOI
       SD9         VXY
       RAD  AM2
       AAG  AMI    VXY            R |y|     - R |x|
VXY    SAD  TR7
       SAG  TR6
       ANG  DIVIS  DPC
DPCCII ANN  DEFC   DIVIS
```




```
P00I2  EDI  DPI2                 même programme que pour P0011, mais MQ
       TDI  DPC                  a alors une valeur donnée
       RAD  MQ
       EDI          ØCBI
       SAG   8003   DIVSD
DPCI2  00   0000    DEPC

DEPC   RAD  A000I                cas de dépassement arithmétique
       DAG   0004
       AAD          8002
       ART   0000   DC           arrêt de la forme
DC     TRD  CTDEP   FIN          OI - adresse de l'ordre - DC
```

CØ = I3
Somme logique (OU)

X : interprété
Am : adresse en mémoire du nombre à comparer à ACC
A' : ignoré

Opérations

La partie X est extraite de la mémoire Am et transmise à l'unité arithmétique. Celle-ci effectue la somme logique de AmX et ACC, le résultat est inscrit dans ACC. L'opération consiste à construire digit par digit un nombre tel que :

$d_{-i} = 1$ si $d_{-i} = 1$ dans ACC OU dans AmX

d'où $d_{-i} = 0$ si $d_{-i} = 0$ dans ACC et AmX

CØ = I4
Produit logique

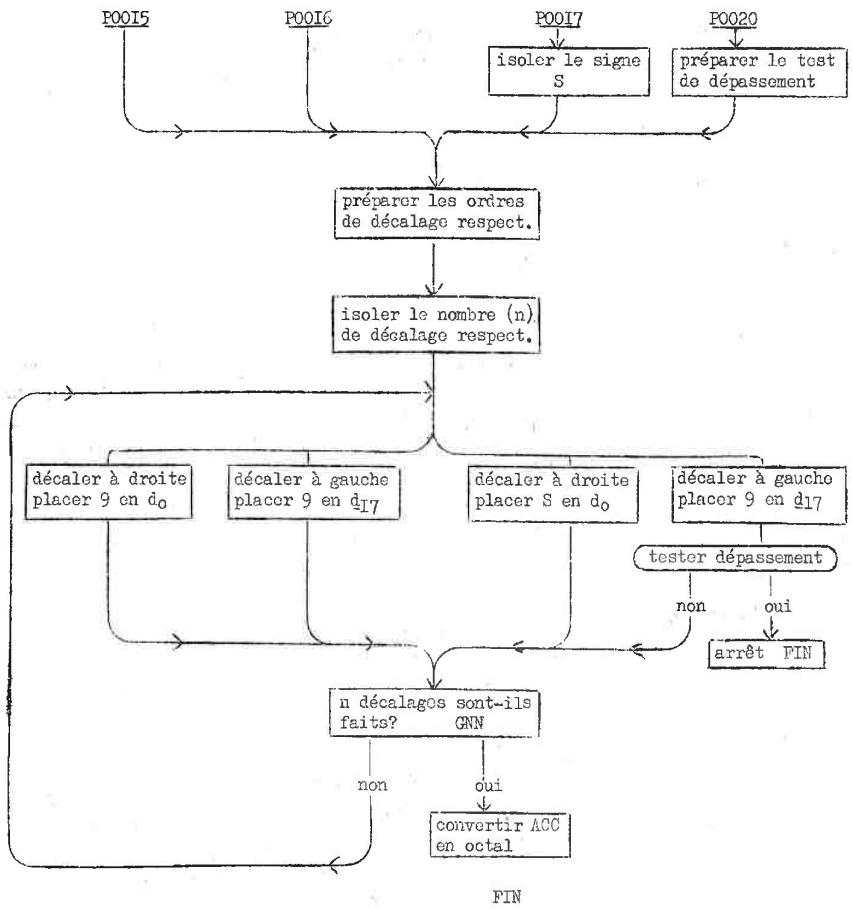
X : interprété
Am : adresse en mémoire du nombre à comparer à ACC
A' : ignoré

Opérations

L'unité arithmétique effectue le produit logique de AmX et ACC. Le résultat inscrit dans ACC est un nombre tel que :

$d_{-i} = 1$ si $d_{-i} = 1$ dans ACC ET dans AmX

d'où $d_{-i} = 0$ si $d_{-i} = 0$ dans l'un ou l'autre.



Opérations de décalage

La forme de l'instruction est la même pour les six ordres dont TYKKA dispose :

X : ignoré
A1A2 : ignoré
A3 : nombre de décalages à effectuer
A' : ignoré

Dans le calculateur TYKKA, A3 représente un nombre de décalages écrit dans le système octal. Pour ne pas alourdir le décodage de ces ordres, nous avons supposé A3 donné dans le système décimal (le programmeur devra l'exprimer lui-même dans cette base pour des essais sur simulation).

CØ = I5
Décalage à droite logique de (ACC)

Le contenu de ACC est décalé de A3 rangs binaires à droite. Des zéros sont introduits dans les positions de gauche laissées libres par les décalages. Les digits les plus à droite sont perdus.

CØ = I6
Décalage à gauche logique de (ACC)

Le contenu du registre ACC est décalé de A3 rangs binaires à gauche. Les A3 digits les plus significatifs de ACC sont perdus, des zéros sont introduits dans les positions inférieures.

CØ = I7
Décalage à droite arithmétique de (ACC)

Le contenu de ACC est décalé de A3 rangs binaires vers la droite, les derniers digits de ACC sont perdus. Pour respecter la règle arithmétique donnée au chapitre 11, il faut reproduire, dans les positions supérieures, le digit signe du nombre donné. Le résultat, placé dans ACC, représente (ACC) x 2^{-n} .

CØ = 20
Décalage à gauche arithmétique de (ACC)

Le contenu de ACC est décalé de A3 rangs binaires à gauche. Les premiers chiffres binaires sont perdus, des zéros sont introduits dans les positions inférieures. Le résultat, placé dans ACC, représente $(ACC)x2^n$.

Dépassement :
La multiplication par 2 peut produire un dépassement de capacité. Il est caractérisé par un changement de signe du contenu de ACC. Dans ce cas l'indicateur de dépassement est enclenché, cependant le nombre de décalages demandé est effectué.

```
P00I5  EDI          S0SI6
       DAD   000I                ordres de décalages
       AAG   CT09   SISI6        caractéristiques de CØ=I5

P00I6  EDI   POPI6  S0SI6
S0SI6  TDI   TR3    S4SI6
S4SI6  RAG   RADR4
       DAG   0004
       GNN          PF4
       TRG   TRI                 A3 isolé, placé en TRI
       RAD   ACC2                ACC en binaire
       DAG   000I   S2SI6        placé en 8003-8002
S2SI6  AAG   ACCI   TR3
POPI6  DAG   000I                en TR3 : ordres de décalage
       AAD   CT90   SISI6
SISI6  TRG   ACCI   SIS20
SIS20  TRD   ACC2
       RAG   TRI                 a-t-on effectué A3 décalages ?
       SAG   CTOI
       GNN          S3SI6
       TRG   TRI                 si non : boucler pour faire un nouveau décalage
       RAD   ACC2   S2SI6

S3SI6  RAD   ACC2
       DAD   000I   S5SI6
S5SI6  AAG   ACCI
       DAG   000I                cadrer ACCI. ACC2 dans la forme  0xxx xxx xxx
       DAD   000I                                                 0xxx xxx xxx
       TRD   ACC2
       TRG   ACCI
       EDI          BIØC         convertir (ACC)f en octal
       TRD   ACC    PF4

P0020  EDI   ACCI
       SD9          DC9          construction du test de dépassement de capacité
       EDI   DG88   DG
DC9    EDI   DG99   DG
DG88   SD9   SIS20  DEPC2
```



```
DG99   SD9   DEPC2  SIS20

DG     TDI   TR4
       EDI          SOSI6
       DAG   000I
       AAD   CT90                ordres caractéristiques de CØ=20
       TRG   ACCI   TR4

DEPC2  ART   0020                arrêt dépassement
       TDI   CTDEP  SIS20
```

CØ=2I
Décalage à droite en double précision

C'est un décalage arithmétique sur le nombre en double précision formé par le contenu des registres ACC et MQ Ce nombre est décalé de n=A3 positions binaires à droite. Les digits inférieurs de MQ sont perdus, des zéros s'inscrivent à gauche dans ACC. Les n digits sortant de ACC sont introduits dans MQ (en suivant les règles de la double précision). Le résultat placé dans ACC-MQ représente le nombre donné multiplié par 2^{-n}. On effectuera au plus I8 décalages.

CØ=22
Décalage à gauche en double précision

C'est un décalage arithmétique de A3 rangs binaires. Les digits sortant de MQ sont introduits dans les positions inférieures de ACC, des zéros s'inscrivent à gauche dans MQ. L'opération correspond à une multiplication par 2^n.

Dépassement
Il se produit si le résultat de cette multiplication devient supérieur à l'unité, en valeur absolue. L'indicateur de dépassement est enclenché, mais les opérations de décalage se poursuivent.

Opérations sur les registres d'indices

Ces quatre ordres ont une instruction de la forme

X : interprété

A' : numéro du registre d'indices intéressé

A1A2A3 : adresse facteur de l'opération

Nous regarderons dans chaque cas particulier le rôle joué par le registre T : mémoire intermédiaire entre les registres d'indices et la mémoire générale, ou l'additionneur d'adresse. En particulier dans l'interprétation de X : nous avons dit (page40) que les registres RiA' suivaient les mêmes règles que les registres ACC ou MQ. Les opérations commandées sur les RiA' sont effectués dans les circuits "adresse" du calculateur et non dans les circuits arithmétiques. Elles ne détruisent donc, ni ACC, ni MQ, et sont utilisées en parallèle.

CØ=30

Inscription dans RiA'

Le contenu AM de la mémoire d'adresse Am est transmis au registre T. Puis la partie spécifiée parX est inscrite dans le registre voulu, après effacement de celui-ci. Les valeurs initiales de T et de RiA' sont perdues.

CØ=3I

Enregistrement de RiA'

Le contenu du registre A' est transmis au registre T. Puis la partie de T indiquée par X est inscrite dans la mémoire d'adresse Am. Le contenu de RiA' reste inchangé, la valeur initiale de T est perdue.

CØ=32

Addition, sans effacement dans RiA'

Le contenu du registre A' est transmis à T. La partie AMX de la mémoire Am est transférée à l'additionneur des circuits d'adresse. Puis la somme est effectuée entre le registre T et l'additionneur, le résultat se construit dans T, il est ensuite enregistré en RiA'. Aucun test de dépassement n'est effectué au cours de ces opérations. Dans tous les cas un résultat est transféré vers le registre A'.

CØ=33

Soustraction, sans effacement, dans RiA'

De même que pour l'addition, le résultat de la soustraction se construit dans le registre T, avant d'être enregistré en RiA'.

Ces opérations sont effectuées dans le registre A'=0 comme dans les autres, on remarquera que le registre T représente toujours le contenu du RiA' sur lequel a été effectuée la dernière opération. Pour faciliter l'utilisation des adresses indicées, un ordre spécial de comparaison a été créé.

CØ=34

Comparaison

Le contenu du RiA' est extrait de la mémoire et inscrit dans le registre T. La partie AMX de la mémoire Am est envoyée à l'additionneur des circuits d'adresse qui calcule alors : (RiA') - AMX et place le résultat en T. RiA' n'a subi aucune modification. Des ordres de test relatifs à T permettent d'exploiter ce résultat.

```
P0030  RAD  POP30                 former l'ordre :  TDI  RiA'   FIN
       AAD  APRIM
       EDI  AMXI                  appeler AMX dans le distributeur, le placer
       TDI  A0006   8002          dans le registre T
POP30  TDI   0000   FIN

P0031  RAD  SOS3I  SIS3I          former l'ordre :  RAG  RiA'  ....
SIS3I  AAD  APRIM   8002          l'exécuter
SOS3I  RAG   0000
       TDI  A0006  SOSOI          placer RiA' dans T
                                  SOSOI : placer RiA' dans la mémoire Am
P0032  EDI  SIS32
       TDI  S2S32
       RAD  AMXI                  envoyer AMX en mémoire de travail pour
       EDI  SOS32  ØCBI           l'addition
SOS32  TRD  TR7
       TRG  TR6
       RAD         SIS3I          former l'ordre  RAD  RiA' .... l'exécuter
       RAD   0000
       EDI         ØCBI
       EDI         ADDIT          commander l'addition
       EDI  S2S32  BIØC           convertir le résultat en octal
SIS32  AAG  APRIM
       AAG          8003
       TRD   0000  SOS34
SOS34  TRD  A0008  FIN
```

```
P0033  EDI  SIS32  SOS33
SOS33  TDI  S2S32
       RAD  AMXI                 prendre l'opposé de AMX et commander l'addition
       EDI         ØCBI          RiA' +(-AMX) comme pour l'ordre 32.
       EDI  SOS32  ØPPØS

P0034  EDI  SOS34  SOS33         préparer l'ordre de transfert uniquement
                                 vers T.
```

Ordres de rupture de séquence

Ces ordres permettent de dérouler deux parties différentes d'un programme en fonction d'un résultat précédent. Forme de l'instruction :

X : ignoré

AIA2A3 : adresse de l'instruction suivante en cas de rupture de séquence.

A' : numéro du registre d'indice.

Si la condition voulue n'est pas vérifiée, on passe à l'instruction suivante en séquence. Si la condition est réalisée, l'instruction suivante sera prise en mémoire à l'adresse Am. Pour cela le contenu du compteur d'instruction CI est enregistré dans le registre d'indice RiA' en passant par l'intermédiaire de T, puis l'adresse Am est envoyée en CI. Si A'=0, le contenu de CI est perdu, le registre de numéro zéro n'est pas modifié.

Test sur le contenu du registre ACC.

CØ=23 Saut si ACC $\geqslant$ 0

CØ=24 Saut si ACC $>$ 0

CØ=25 Saut si ACC = 0

CØ=26 Saut si ACC $\neq$ 0

Test sur le contenu du registre T.

CØ=35 Saut si T $\geqslant$ 0

CØ=36 Saut si T $>$ 0

CØ=27 Saut si l'indicateur de dépassement est enclenché

CØ=37 Saut inconditionnel, provoque automatiquement une rupture de séquence.

Programme

La condition indiquée est testée dans l'accumulateur.

Pour un test de signe : un nombre positif correspond à d_o=9
un nombre négatif à d_o=8


```
P0023  EDI  ACCI   SOS23
SOS23  SD9  FIN    P0037

P0024  EDI  ACCI   SOS24
SOS24  SD9  P0037  FIN

P0025  RAD  ACC
       ANN  FIN    P0037

P0026  RAD  ACC
       ANN  P0037  FIN

P0027  RAG  CTDEP                placer en 8003 l'indicateur de dépassement
       TRD  CTDEP
       GNN  P0037  FIN

P0035  RAD  A0008                convertir T en binaire
       EDI  SOS23  ØCBI          tester comme pour ACC

P0036  RAD  A0008
       EDI  SOS24  ØCBI

P0037  RAG  APRIM
       GNN         SIS37         si A'=0, ne pas modifier RiA'
       AAG         SOS37         construire l'ordre  TRD  RiA' ....
       TRD  0000   SOS34         SOS34 = TR  A0008  FIN
SOS37  AAD  CI
       EDI  RADRS                placer Am en CI
       TDI  CI      8003
SIS37  AAG         SOS37
       SØP  0000   FIN
```

UTILISATION DE LA SIMULATION

Encombrement

Le programme n'occupe que les mémoires dont l'adresse est formée avec un chiffre 8 ou 9. Toutes les adresses octales comprises entre 0000 et I777 sont donc disponibles pour les ordres TYKKA.

Les mémoires 0000, 000I, ..., 0007, ont été affectées aux registres d'indices TYKKA, pour respecter la règle qui les place en tête d'un bloc de mémoires.

Quelques mémoires restaient libres après l'assemblage. Ceci permet au programme de chargement, n-mots par carte, nécessaire pour placer la simulation sur le tambour, de rester en mémoire à poste fixe.

Essais d'un programme

Il faut indiquer à la logique, l'adresse en mémoire du premier ordre à décoder en la plaçant dans la mémoire I949 sous la forme 00 0000 N. Puis s'adresser à I59I pour exécuter le programme.

Les modifications de données successives pour l'étude d'un problème se feront par cartes n-mots. Le programme de chargement débute en I09I. L'analyse est perforée si on place un signe moins au pupitre.

Différents arrêts

Ils sont caractérisés par le contenu du registre programme.

L'ordre d'arrêt CO=00, placé dans la mémoire N, provoque un arrêt-machine sur OI N 0000 .

Les autres arrêts sont dus à des dépassements de capacité.

OI	0666	xxxx	en cours d'addition, soustraction
OI	00I0	xxxx	en cours de multiplication
OI	N	xxxx	avant une division par un ordre placé en N
OI	0020	xxxx	en cours de décalage, simple précision
OI	0022	xxxx	en cours de décalage, double précision

Dans tous les cas en appuyant sur le bouton "départ-programme", la suite des opérations se déroule comme dans le cas "commutateur de dépassement sur détection".

* * *

D E U X I E M E P A R T I E

D'UN LANGAGE SYMBOLIQUE ALPHABETIQUE, POUR FACILITER L'ECRITURE D'UN PROGRAMME EN " CODE DE PROGRAMMATION ".

E C R I T U R E S Y M B O L I Q U E

Nous disposons, pour faciliter la rédaction d'un programme sur l'ordinateur 650, d'un langage extérieur beaucoup plus maniable que le langage machine : le "Code de Programmation", ou C D P . Logique extérieure qui groupe de nombreux ordres, plus ou moins complexes écrits sous forme numérique.

Cependant, l'expérience du langage PASO nous avait montré les avantages d'une écriture symbolique. Nous avons voulu joindre au C D P numérique une telle possibilité dans la rédaction. Chaque ordre en C D P a reçu un équivalent de forme symbolique, le programme CODA a pour rôle de rétablir la forme numérique nécessaire pour l'exécution en code. C

Chaque adresse peut également s'écrire sous forme symbolique alphabétique, ce qui facilite la lecture d'un programme et sa vérification. Le choix des adresses tambour est laissé à CODA .

Le programme symbolique est perforé à raison d'un ordre par carte. CODA lit une carte donnée, traduit l'ordre symbolique, perfore une carte résultat qui porte l'ordre en code de programmation sous sa forme numérique. Puis il retourne à la lecture de la carte suivante, ceci jusqu'à épuisement des cartes données.

La traduction consiste à rechercher, pour chaque code opération symbolique, son équivalent numérique et, pour chaque adresse non absolue une adresse tambour qui lui sera réservée.

Il ne se présente pas de problème d'optimisation puisque les ordres en code n'ont pas d'adresses optima. CODA donne aux symboles qu'il rencontre des équivalents-tambour placés en séquence à partir de la première mémoire libre. Le nombre de mémoires réservées pour un symbole est variable. Un compteur d'adresse, noté par la suite CTA, contient, avant toute nouvelle recherche d'équivalence, l'adresse de la première mémoire-tambour libre.

Les règles pour les adresses symboliques ou absolues ont été choisies aussi voisines que possible de celles du programme PASO II pour éviter d'apprendre deux jeux de conventions différents.

Initialement toutes les mémoires du bloc 0000-0999 sont disponibles pour l'assemblage. Un pseudo-code de "réservation absolue" permet de fixer l'assemblage dans une certaine zone du tambour.

Exemple

RES	AB	0000	0200
RES	AB	0400	0600

Les blocs de mémoire [0000 - 0200] et [0400 - 0600] sont réservés, bornes comprises.

L'assemblage débutera en 020I, quand la mémoire 0399 sera occupée, CODA passera à l'adresse 060I.

DIFFERENTS TYPES D'ADRESSE

Une adresse est formée de 1 à 5 caractères alphanumériques $|A_I|\ A_2\ A_3\ A_4\ A_5|$

A_I est partie symbolique de l'adresse

$A_2\ A_3\ A_4\ A_5$ partie absolue.

Adresse absolue formée de 4 chiffres placés en partie absolue, de façon que A_I soit blanc. Elle doit être dans une zone au préalable réservée.

Adresse symbolique formée d'un groupe de 1 à 5 caractères alphabétiques ou numériques, placés de façon que A_I soit non blanc.
Le pseudo-code EQIAB "équivalence absolue" permet d'imposer un équivalent absolu à un symbole donné.
exemple |EQI| AB | |A|LPHA| | |0049|
Au symbole ALPHA sera affecté l'adresse absolue 0049. Cette adresse doit être dans une zone réservée.

Les adresses d'un ordre en code sont fréquemment indexées, il faut alors réserver un nombre n de mémoires en séquence à partir de l'équivalent donné au symbole. Le pseudo-code "RESIN - réservation pour indexage" permet d'indiquer n.
exemple |RES|IN |A|LPHA| | |0005|
Si 0060 et l'équivalent de ALPHA, les 5 mémoires 0060 - 0064 seront réservées.
L'emploi de ce pseudo-code est indépendant de celui d'équivalence absolue.

Adresse régionale la partie A_I est une lettre qui définit la région. La partie $A_2\ A_3\ A_4\ A_5$ est un nombre de 4 chiffres qui désigne le numéro d'ordre du mot dans la séquence : le mot 1 correspond à R00I, le mot 5 à R0005.
Une région doit toujours être définie par un pseudo-code "RESRG - réservation régionale". Ce pseudo-code peut être utilisé de deux façons différentes.

exemples

1) pour indiquer l'équivalent du mot 1 et la longueur de la région

|RES|RG| |A|0I0I| | |0I05|

impose à A000I l'équivalent 0I0I, à A0005 l'équivalent 0I05.
Mais la zone 0I0I - 0I05 doit être réservée au préalable.

2) pour indiquer seulement la longueur de la région.

|RES|RG| |A|0000| | |0005|

0000 n'est pas considéré comme l'équivalent du mot 1 de la région A. CODA affectera lui-même un équivalent au mot 1 et réservera alors une zone de 5 mots en séquence.

Si l'assemblage demande plus de mémoires qu'il n'a été prévu, le pseudo-code de "libération" permet de libérer une zone du tambour au moment de la saturation en cours d'assemblage.

exemple |LIB|ER| |0650| | |0700|

L'assemblage se poursuivra à partir de l'adresse 0650.

FORME SYMBOLIQUE D'UN ORDRE. CARTE "CODA"

Le code opération est à 4 ou 5 caractères alphanumériques.

Un ordre peut avoir une, deux ou trois adresses facteurs suivant qu'il correspond à un ordre code, à une mémoire, à un calcul de fonction, ou à une opération algébrique.

L'adresse instruction suivante sera toujours inscrite dans les mêmes positions.

L'adresse instruction suivante peut être en blanc, si l'adresse de l'ordre suivant l'est également. CODA affectera aux deux le même équivalent. De même l'adresse résultat d'un ordre et l'adresse donnée de l'ordre suivant peuvent être en blanc, elles recevront le même équivalent.

PROGRAMME ASSEMBLE

Pour chaque carte lecture, CODA perfore une carte résultat.

C'est une carte chargement du type n mots par carte, qui porte :

- la reproduction de l'ordre symbolique donné
- une numérotation
- l'ordre en code, résultat de l'affectation des mémoires.

Ces cartes seront chargées à la suite de la logique extérieure pour exécution du programme. Il n'est pas nécessaire de supprimer les cartes pseudo-codes, elles sont acceptées par le programme de chargement du code de programmation.

Aspect d'un programme

		RES	AB		0050			0999					
		RES	RG	P	0077			0086					
		RES	RG	A	000I			00I0					
A	LGOR	INI	T									A	LGI
A	LGI	LEC	T	A	000I								
		ADD	VX		I967		A	000I					
		MUL	VX				C	STI2		R	XI2		
		SST	VX	C	STI		R	XI2				A	LG2B
A	LG2B	DIV	VX			I	A	0002					
		ADD	VX				A	000I	I	P	000I		
		BOC	LI	O	0005		A	LG2B			000I		
		PER	F	P	000I							A	LGI

adresse de l'ordre

5 adresses facteurs A, B, C,

code operation

adresse instruction suivante

T A B L E A U D E C O N N E X I O N S

LECTURE

Les cartes CODA sont caractérisées par une perforation I2 en colonne 47, qui commande la lecture par ENTREE A. Soit à placer dans la bande de lecture la valeur transcodée de six symboles alphanumériques, de chacun cinq caractères. Quatre mots de cette bande restent ibres : les valeurs numériques des index i, j et k, et la partie absolue des trois adresses facteurs A, B et C y seront enregistrées.

Pour chaque symbole

- les 5 positions de la LECTURE N°I sont connectées aux 5 positions correspondantes d'un mot LECTURE I ALPHA.
- les 5 positions de la LECTURE N°2 (A) sont connectées aux 5 positions inférieures d'un mot ENTREE A MEMOIRE. Ceci pour les mots 1 à 6 de la bande choisie.

IO MEMOIRES BANDE DE LECTURE 1951-1960

1951				DEBUT							
1952				CODE OPERATION							
1953				ADRESSE A							Transcodé des symboles alphanumériques de l'ordre donné
1954				ADRESSE B							
1955				ADRESSE C							
1956				SUITE							
1957	0	0	0	i	0	0	0	0	j	k	index
1958	0	0	0	0	0	0	A	A	A	A	partie absolue des 3 adresses A, B et C
1959	0	0	0	0	0	0	B	B	B	B	
1960	0	0	0	0	0	0	C	C	C	C	

La dimension IO de l'émetteur dimension mot est connectée aux 6 mots M1 à M6 pour l'entrée alphanumérique A.

De plus, pour les symboles A B C

- les 4 positions, partie absolue du symbole, sont enregistrées à partir de LECTURE N°2 (B), dans les quatre positions inférieures des mots 8. 9 et IO de l'ENTREE A MEMOIRE. Des zéros les complètent.

Index i, j et k

Le mot 7 leur est réservé. Si i est "non blanc" sa valeur numérique est enregistré. Si i est "en blanc" un zéro est enregistré. De même pour j et k. Ceci par l'intermédiaire des sélecteurs-pilotes 1. 2 et 3.

Connexions correspondantes

Symbole à enregistrer	Positions de lecture émettrices	6 fois 5 positions		3 fois 4 posit.
		LECTURE N°I vers ENTREE ALPHA	LECTURE N°2 (A) vers ENTREE A MEMOIRE	LECTURE N°2 (B) vers ENTREE A MEMOIRE
"Début"	48/49.50.51.52	mot 1	mot 1	
code opération	53.54.55/56.57	mot 2	mot 2	
adresse A	59/60.61.62.63.	mot 3	mot 3	mot 8
adresse B	65/66.67.68.69.	mot 4	mot 4	mot 9
adresse C	71/72.73.74.75.	mot 5	mot 5	mot IO
"Suite"	76/77.78.79.80.	mot 6	mot 6	

L'impulsion I2 en colonne 47 commande la lecture A par l'intermédiaire de l'éliminateur d'X, et l'entrée alpha pour les 6 mots par l'intermédiaire d'un sélecteur pilote, excitation immédiate.

LECTURE N°I position 47 ⟶ ELIMINATEUR d'X Commun LECTURE I2-x ⟶ SELECTEUR PILOTE N°4 EXC-X EXC-IM ↓ M1...M6

Trois sélecteurs pilotes construisent le mot 7 sous la forme 000i000jk.

Ils sont excités par digit depuis la LECTURE N°I. Ils reçoivent en position

- Transférée, la valeur de l'index voulu de la LECTURE N°2 (B)
- Normale, zéro, par IMPULSION LECTURE ©
- Commun, transmet la valeur T ou N dans la position choisie du mot 7, ENTREE A MEMOIRE.

Ils sont maintenus en lecture par une impulsion MAINT-LEC.

	Sel. Pil.	EXC. D	T	N	C
i	1	position 58	58	zéro	position 7
j	2	position 64	64	zéro	position 2
k	3	position 70	70	zéro	position 1

Les zéros nécessaires dans les positions de gauche des mots 7. 8. 9 et IO sont placés par les émetteurs de "dimension mots".

EMET. DIMENSION. MOTS | 4 en M8. M9. MIO / 3 en M7 | ENTREE. DIMENSION A.

PERFORATION

A partir de la bande mémoire de sortie donnée par le programme, les connexions doivent établir la carte CODA résultat. Elle porte : - l'ordre symbolique de la carte donnée - l'ordre numérique présenté sous forme n mots par carte. CODA utilise la perforation A.

Ordre symbolique

Les IO positions des mots 1 à 6 pris en SORTIE MEMOIRE A sont connectées au 5 positions voulues de PERFORATION CARTE A.
Les index i, j et k sont rétablis. Le numéro d'ordre de la carte mis en place,

Symbole	SORTIE MEMOIRE A	Positions de PERFORATION CARTE A
"Début"	mot 1	48/49.50.5I.52.
code-opération	mot 2	53.54.55/56.57.
adresse A	mot 3	59/60.6I.62.63.
adresse B	mot 4	65/66.67.68.69.
adresse C	mot 5	7I/72.73.74.75.
"Suite"	mot 6	76/77.78.79.80.
numéro carte	mot 9 5. 4. 3.	4I.42.43.

la perforation A ainsi que la sortie alphanumérique de ces symboles sont commandées par la position 3 du mot IO.

INFORMATION DE CONTROLE position 3 ⟶ M1. M2. M3. M4. M5. M6. de SORTIE ALPHA ⟶ PERF. A.

Ordre numérique

Chacune des cinq adresses résultats est envoyée en perforation A par l'intermédiaire d'un coselecteur. Lequel est commandé par une position du MOT IO, information de contrôle.

- Si 8 : le co-sélecteur est transféré et transmet le résultat voulu.
- Si 0 : le contact est normal, l'adresse sort en blanc.

IO MEMOIRES BANDE DE PERFORATION 1977 - 1986

1977	DEBUT										
1978	CODE OPERATION										
1979	ADRESSE A										Transcodé des symboles donnés
1980	ADRESSE B										
1981	ADRESSE C										
1982	SUITE										
1983	C	Ø	N	N	N	I	A	A	A	A	ordre assemblé
1984	j	k	B	B	B	B	C	C	C	C	
1985	D	D	D	D	n		n°				adresse de l'ordre numérotation
1986	8/0	8/0	8/0	8/0	8/0	8/0		8/0	8/0	8/0	commande de perforation

si 8, adresse en blanc pour

DDDD

CØ

AAAA

jBBBB

kCCCC

NNNi

si 8
perfo A

si 8
perfo pour n=4

8 multiplication

0 division

mot 1	mot 2	mot 3	mot 4
00 DDDD 000n	CØ NNNi AAAA	jBBBB kcccc	
← ordre à 2 mémoires →			
00 DDDD 000n	CØ UUUU VVVV		
← ordre à 1 mémoire →			
00 DDDD 000n	CØ NNNi AAAA	jBBBB kcccc	
← ordre à 3 mémoires →		multiplication division →	3λ 000 μ 1939 3λ 000 μ 1991

Les index j et k sont joints aux adresses B et C.
L'index i est joint à l'adresse NNN, pour être en accord avec la forme des ordres négatifs : dans le cas non perforation de l'adresse U.

Des impulsions "zéro" complètent le mot 1 de la carte.

Le code opération est tributaire d'un sélecteur pilote.

Pour multiplication et division le mot 4 est construit par les émetteurs digits, seules les variables λ et μ sont prises en SORTIE MEMOIRE A.

Sont nécessaires : les co-sélecteurs 1, 2, 3 - les sélecteurs pilotes 9. IO et 11.

Leurs contacts sont fonction des positions 1 et 2 du mot IO, information de contrôle.

La multiplication est caractérisée par 80
La division — — par 88 en position 1 et 2

8 en position 2 : excite les co-sélecteurs 1, 2 et 3.
8 en position 1 : excite le sélecteur pilote 3.
Dans ce cas, le sélecteur pilote 11 est à son tour excité par l'intermédiaire du co-sélecteur 3, il placera 91 pour compléter le mot 4, sinon, le sélecteur pilote IO est excité, il placera 39 pour compléter 4.

Adresse	Co. Sel.	SORTIE MEMOIRE vers Transféré	Normal	vers PERFORATION de Commun
DDDD	IO	Mot 9 IO.9.8.7.	blanc	3. 4. 5. 6.
NNNi	5	mot 7 8.7.6.5.	blanc	I3.I4.I5.I5.
AAAA	8	mot 7 4.3.2.I.	blanc	I7.I8.I9.20.
jBBBB	7	mot 8 IO.8.7.6.5.	blanc	2I.22.23.24.25.
kcccc	6	mot 8 9.4.3.2.I.	blanc	26.27.28.29.30.
CØ	Sel. Pil. 11	mot 7 I.2.	blanc	II.I2.

* * *

E T U D E D E S S O U S - P R O G R A M M E S

Ils sont au nombre de huit, le programme CODA y fera appel pour l'étude de chaque donnée. Voici leurs rôles respectifs.

Ss. P - 1	calcul de l'adresse relative à un symbole
Ss. P - 2	consultation de la table des symboles
Ss. P - 3	mise en mémoire des réservations absolues
Ss. P - 4	recherche du type d'une adresse
Ss. P - 5	recherche de la partie absolue d'une adresse
Ss. P - 6	étude de la validité de l'adresse CTA
Ss. P - A	recherche de l'équivalent d'une adresse régionale
Ss. P - B	recherche de l'équivalent d'une adresse symbolique.

Ces sous-programmes prennent leurs renseignements dans quatre tables qui occupent la première moitié du tambour - ce sont :

la table de codes opérations

la table des adresses régionales

la table des symboles

la table des réservations absolues.

Nous allons étudier en détail la façon dont CODA les consulte, soit directement, soit par un sous-programme approprié.

TABLE DE CODES OPERATIONS

Les valeurs transcodées des codes symboliques, rangés par ordre alphabétique, sont rangées dans les mémoires 0600 à 0699. Leurs équivalents sont placés dans les mémoires 0700 à 0799, à raison d'un code par mémoire accompagné d'indicatifs 8 ou 9 nécessaires pour l'étude de l'ordre. Ces équivalents sont de trois types :

XX 989 00000 - pour un ordre classique en code de programmation, XX est l'équivalent propre du code symbolique. Trois positions sont suffisantes pour l'indicatif.

00 989 FFFF - pour un ordre de calcul de fonction en virgule flottante. Le cade opération est 00, la partie FFFF est caractéristique de la fonction à étudier et se placera dans la partie adresse A de l'ordre CDP.

00 0000 OPPPP - pour les pseudo-codes propres au programme CODA, aucune équivalence n'est nécessaire. L'adresse PPPP est celle du premier ordre du sous-programme relatif au pseudo-code donné. Elle permet l'aiguillage.

La consultation de cette table est immédiate grace à l'ordre CDT de l'ordinateur 650. Une vérification du résultat élimine les codes abhérents. Si le code opération est donné sous forme absolue, la table des équivalents est consultée pas à pas, il faut en effet disposer de l'indicatif "989" avant de poursuivre.

TABLE DES ADRESSES REGIONALES

A chaque lettre de l'alphabet CODA fait correspondre une mémoire d'adresse A_R telle que l'on ait 080 A_R 0829 . A_R est obtenue en ajoutant 0740 à la valeur transcodée de la lettre. Cette mémoire contient l'équivalent du mot 1 de la région et le nombre n de mots dont elle est formée, sous la forme :

00 n équiv. 1

La consultation de cette table se ramène à l'appel de la mémoire A_R.

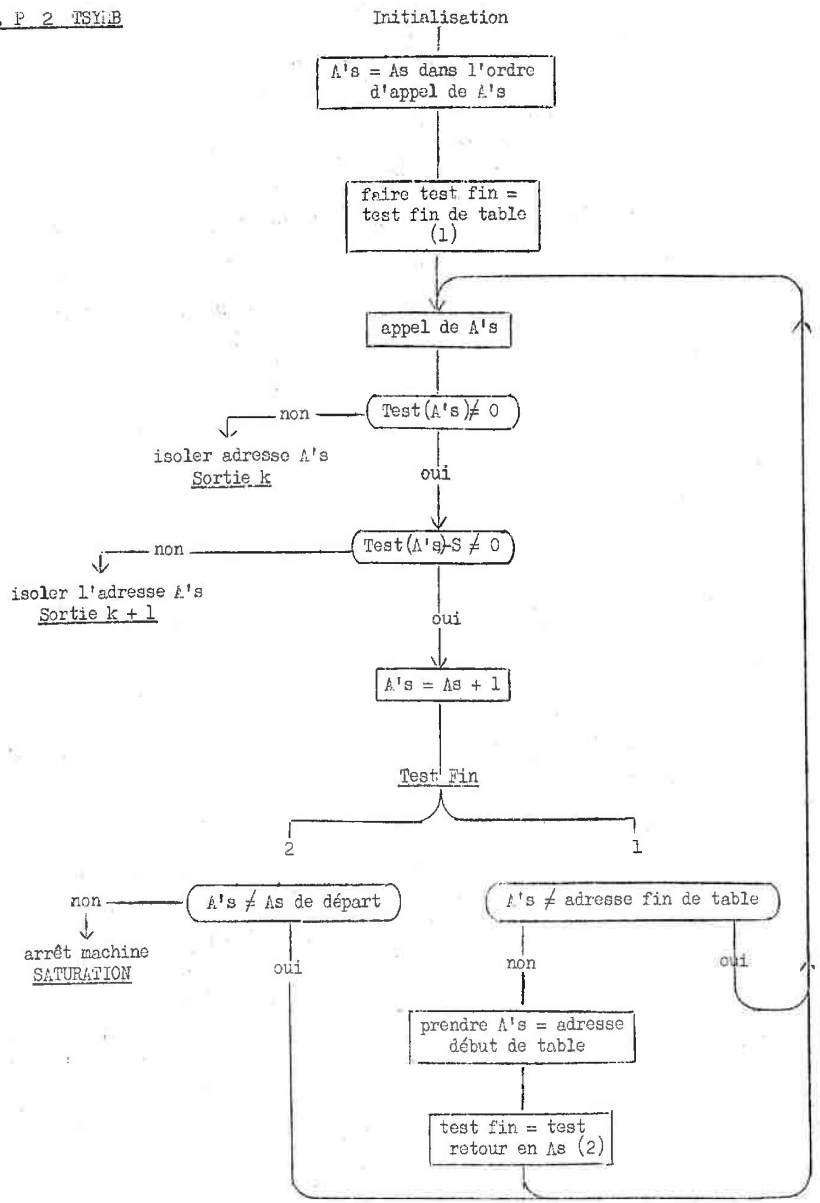
TABLE DES SYMBOLES

Elle est formée de deux blocs de 300 mémoires qui recevront les symboles et leurs équivalents au fur et à mesure de leur affectation par CODA. De 000 à 0299 se trouveront ces symboles, de 0300 à 0599 leurs équivalents. Le principe d'utilisation de cette table est en grande partie celui de PASO II.

A un symbole donné S, CODA fait correspondre une adresse As inférieur à 0300. S doit être rangé dans la table à cette adresse, et son équivalent à l'adresse As + 0300. Il se peut que deux symboles différents donnent la même valeur As, si l'un d'eux est déjà en mémoire, nous placerons l'autre à l'adresse A's = As+1 si elle est libre, ou nous passerons à l'adresse suivante. La consultation de cette table forme un sous-programme de base dans l'affectation des mémoires.
Quand le symbole S est dans la table, la mémoire As + 0300 contient son équivalent tambour et, si besoin est, le nombre n de mémoires qu'il faut lui réserver, sous la forme :

00 équiv. n

Ss. P 2 TSYMB



L'une ou l'autre de ces indications peut s'y trouver seule à la suite d'un pseudo-code RESIN ou RQIAB.

Sous-programmes relatifs à cette table :
(1) pour le calcul de l'adresse As.
(2) pour la consultation de table proprement dite.

Sous-programme 1. noté ADSYM

As est calculé à partir de Ts valeur transcodée du symbole S. C'est un nombre de IO chiffres dont le plus à gauche, au moins, n'est pas nul. Il est multiplié par la constante k = IOO IOO IOO I pour obtenir une combinaison des chiffres qui caractérisent Ts. Le résultat occupe vingt positions d'accumulateur :

x —— xAAA | x ——— x

La partie AAA est isolée, elle vérifie les inégalités 000 ≤ AAA ≤ 999 .
Pour obtenir un nombre compris entre 0000 et 0299, AAA est multiplié par 3 et décalé d'un rang à droite, le nombre obtenu est pris comme adresse As.
Ce procédé explique comment deux symboles différents peuvent donner la même adresse As.

Sous-programme 2. noté TSYMB

Recherche dans la table, à partir de l'adresse As, la première mémoire d'adresse A's, telle que : (A's) = 0 ou (A's) = S donné

utilisation :	8003	8002	8001
données	- symbole S-	00 -As- 0000	00 0000 -k -
résultats	- symbole S-	00 -A's 0000	- symbole S-

Sortie en k : si (A's) = 0

Sortie en k + 1 : si (A's) = symbole S donné

Ss. P 3 LIM

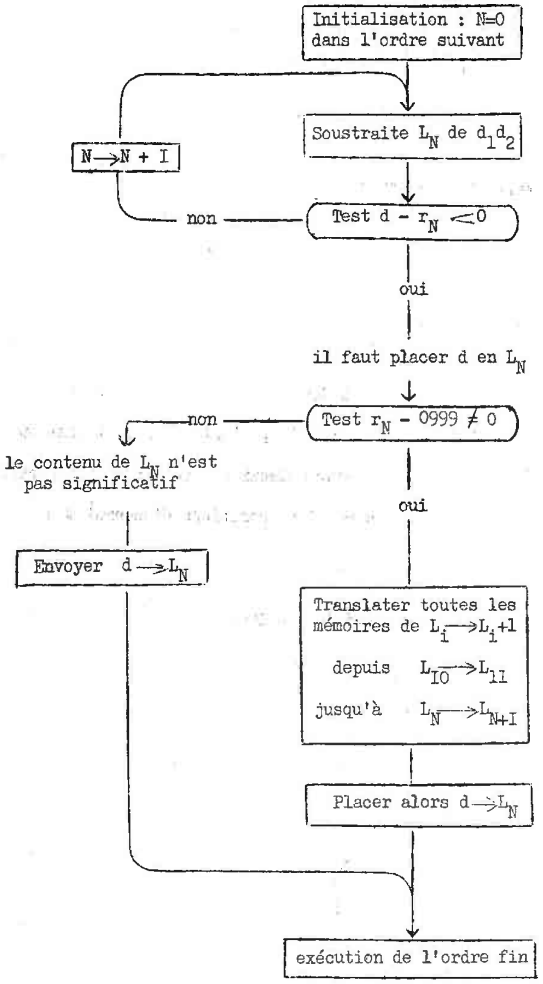


TABLE DES RESERVATIONS

Dix mémoires en séquence LI à LIO forment cette table, elles contiennent les données des zones réservées. Pour interdire les adresses du bloc $[r_1 - r_2]$ nous placerons dans une mémoire L l'indicatif 00 $-r_1$ - $-r_2$ -

Les différents indicatifs sont ordonnés suivant les valeurs croissantes de r_1, puis rangés de LI en LIO. Les zones ainsi définies ne doivent avoir aucune mémoire en commun. L'initialisation place dans ces mémoires : 00 0999 0000, où r_1 est borne supérieure des adresses libres en code de programmation.

Sous-programme 3. noté LIM

Il est chargé de placer, dans une mémoire L, la donnée d'une réservation 00 $-d_1$- $-d_2$- en fonction de sa valeur.

utilisation :	8003	8002	800I
données	00 $-d_1$- d_2	00 0000 0000	- ordre fin-
résultats	———	sans intérêt	———

L'organigramme ci-contre résume les tests nécessaires.

Remarques - Le transfert de la mémoire LIO vers LII a lieu sans test sur le contenu de LII, si celui-ci est significatif, il sera perdu. La limite imposée d'un maximum de IO zones réservées peut paraitre génante. En réalité, CODA fait un assemblage très compact puisqu'il place en séquence toutes les mémoires utiles. Il suffira d'indiquer l'adresse de la première mémoire libre sur le tambour et une limite à ne pas dépasser si une région doit être protégée. L'assemblage est d'autant plus rapide que chaque bloc de mémoires libres est plus grand.

ETUDE D'UNE ADRESSE

Sous-programme 4. noté TYPAD

Etant donnée la valeur transcodée d'une adresse quelconque, non blanche, il nous en indique la nature : régionale, symbolique, absolue. Les tests d'adresse blanche sont particuliers à chaque cas.

utilisation :	8003	8002	800I
données	00 0000 0000	A. transcod.	00 0000 -k -
résultats			
A absolue sortié k+2	A. transcod.	00 0000 0000	00 0000 k +2
A symbolique sortie k+I	- symbole S-	00 0000 0000	- symbole S-
A régionale sortie k	A. transcod.	00 0000 0000	- quelconque

Dans ce dernier cas, 8003 contient seulement la transcodée de la partie absolue de l'adresse. La lettre caractéristique de la région est dans la mémoire de travail TR3.

Test sur A_1

A_1 en blanc ⟶ A. absolue (k+2)

A_1 numérique ⟶ A. symbolique (k+1)

sinon

Test sur les Ai suivants

Si les quatre Ai sont numériques ⟶ A. régionale (k)

sinon

L'un d'eux est une lettre ou un blanc ⟶ A. symbolique (k+1)

Le commutateur dépassement de capacité doit être positionné sur "détection" pour cette étude.

Sous-programme 5

Le tableau de connexion permet seulement l'enregistrement de la partie absolue de trois adresses de l'ordre. Pour les deux autres, nous sommes obligés de déduire la donnée de sa valeur transcodée. Par décalages et éliminations de 9, ce sous-programme regroupe les quatre chiffres significatifs.

utilisation :	8003	8002	800I
données	9x 9x 9x 9x 00	00 0000 0000	- ordre fin-
résultats	00 0000 0000	00 xxxx 0000	- quelconque

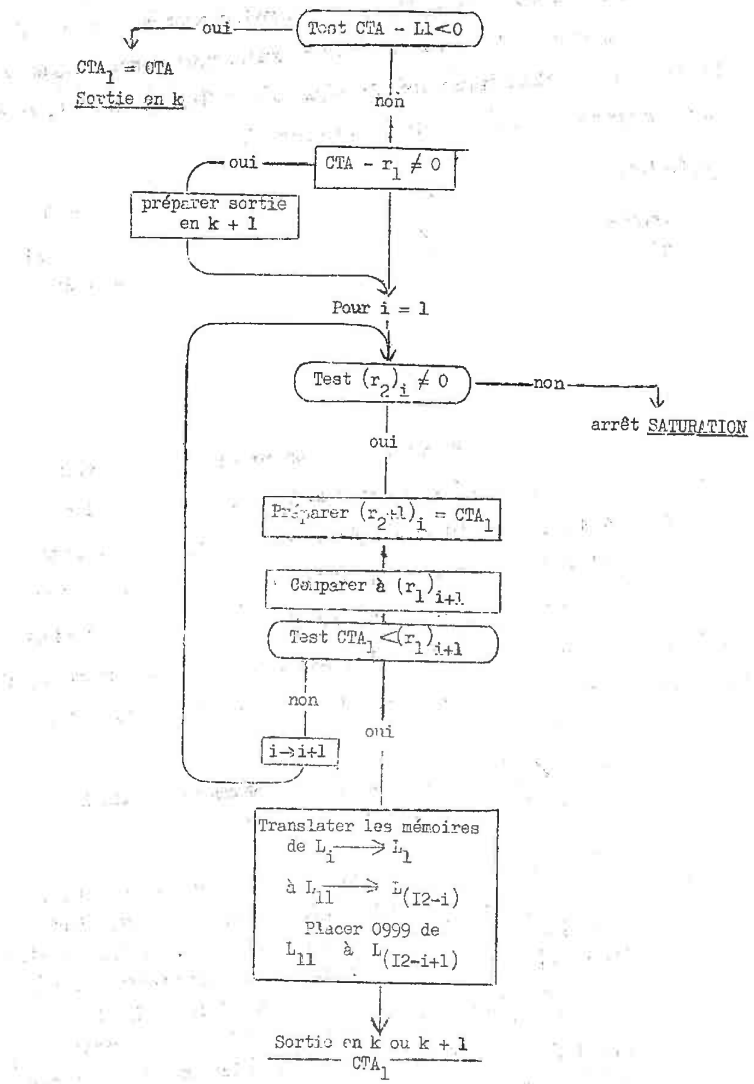
Sous-programme 6. noté VALID

Quand un certain équivalent Es est affecté à un symbole, nous lui réservons en séquence un nombre n, variable, de mémoires. La première mémoire libre a alors pour adresse Es + n , cette valeur est placée dans le compteur CTA. Avant de passer à l'affectation du symbole suivant, il faut vérifier que Es + n n'est pas dans une zone réservée, c'est-à-dire comparer le contenu de CTA aux indicatifs placés dans les mémoires L. Ces indicatifs sont rangés par ordre croissant, il suffit donc de comparer CTA au contenu, 00 $-r_1-$ $-r_2-$, de la mémoire LI.

Si CTA $< r_1$: Es est valable. La recherche des équivalents peut se poursuivre à partir de CTA_1 = Es + n

Si CTA = r_1 : les n mémoires Es à Es + (n-1) étaient libres ; Es est valable. La première mémoire libre, à la suite, n'est plus Es + n, mais r_2 + I que nous placerons en CTA. Le bloc de mémoire $[r_1 - r_2]$ est alors réservé. L'indicatif correspondant sera supprimé, le contenu des mémoires L2 à LIO translaté de Ll à L9 pour qu'une comparaison suivante porte toujours sur Ll. Avant de prendre CTA_1 = r_2 + 1 . Nous vérifierons que le second bloc réservé ne commence pas en r_2 + 1 .

Ss.P 6 VALID



Si CTA $>$ r_1 : les n mémoires Es à Es+(n-1) n'étaient pas toutes libres : E(s) n'est pas valable. Partant de la valeur $CTA_1 = r_2+1$ nous devrons recommencer l'affectation précédente, qu'il nous faudra à son tour vérifier. La translation des mémoires L sera faite comme dans le cas précédent.

Dans ces deux cas si nous trouvons r_2 = 0000, il y a saturation du tambour de 0000 à 0999. En effet ceci n'est possible que pour L_1 = 00 0999 0000.

L'arrêt saturation est de la forme 0I 2222 2222

utilisation :	8003	8002	800I
données	00 0000 CTA-	00 0000 0000	00 0000 -k -
résultats	00 0000 CTA_1	00 0000 0000	00 0000 CTA_1

Sortie en k : si E(s) est valable

Sortie en k + 1 : s'il faut recommencer la recherche de E(s).

RECHERCHE D'UN EQUIVALENT-SYMBOLE

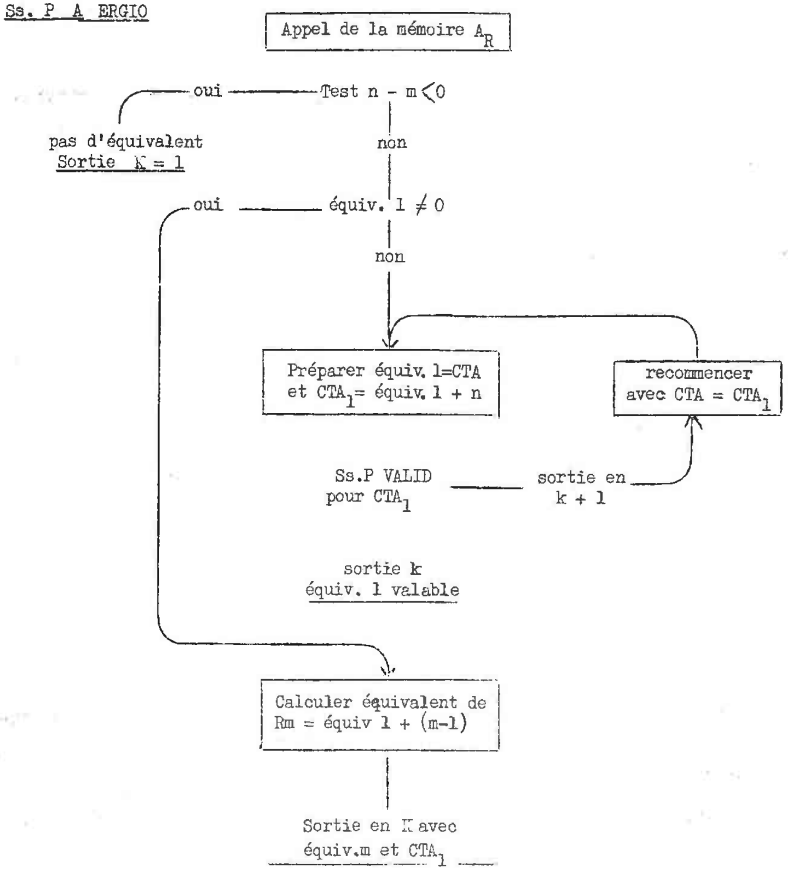
Le processus est différent suivant que l'adresse donnée est régionale ou symbolique, d'où les deux sous-programmes A et B. Ils sont utilisés à la suite du sous-programme 4 .

Sous-programme A. noté ERGIO

La mémoire A_R relative à la région R doit contenir au moins une indication : longueur de la région, soit n, avec ou sans l'équivalent du mot 1 sous la forme

00	équiv. 1	n

Ss. P A ERGIO



si $A_R = 0$ aucun équivalent n'est choisi pour R000m, c'est une erreur-adresse.

si $A_R \neq 0$ il faut vérifier que m est inférieur ou égal à n. Sinon, l'affectation n'a pas lieu. Si oui, l'équivalent de R000m est calculé.

si l'équivalent du mot 1 est donné E(Rm) = E(R1) + (m-1)

si l'équivalent du mot 1 n'est pas en mémoire, on prendra E(R1) égal à CTA en vérifiant que la région [CTA - CTA+n-1] est libre.

utilisation :	8003	8002	800I
données	00 0000 CTA-	00 00-R 00-n	00 0000 -k -
résultats			
équivalent donné sortie en k	00 0000E(Rm)	00 0000 CTA_1	00 0000 000I
pas d'équivalent sortie k+1	00 0000 0000	00 0000 k +I	00 0000 000I

Ci-contre organigramme de cette étude.

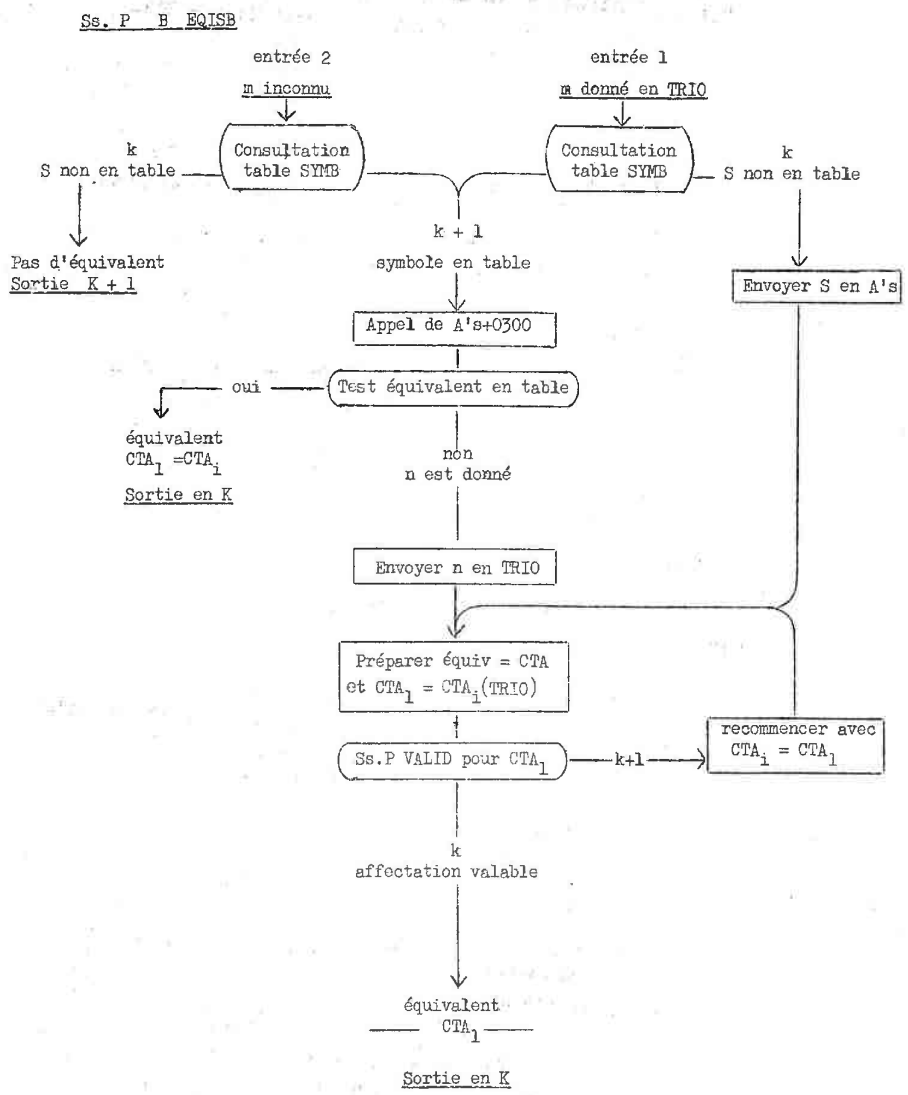
Sous-programme R noté EQISB

Deux entrées différentes sont prévues dans ce sous-programme.

Entrée 1. Nous connaissons, par le maitre programme, le nombre n de mémoires qu'il faut réserver au symbole S.

- Une consultation de la table des symboles nous indique si S est en mémoire ou non.

- S est en table à l'adresse A's, la mémoire A's+0300 contient : ou son équivalent, ou le nombre m de mémoires demandées par le programme, ou ces deux indications. Si l'équivalent E(s) n'est pas donné, nous prendrons E(s) = CTA et nous vérifierons que le grou-



pe de mémoires [CTA - CTA+m-1] est libre.

- Le nombre m a ici priorité sur n qui n'était qu'un minimum.
- Si S n'est pas en table, il sera placé à l'adresse A's et son équivalent E(s) en A's + 0300 par le même processus.

Entrée 2. Nous ne connaissons pas encore le nombre m.

- Si le symbole est dans la table nous revenons au cas précédent.
- Si le symbole n'est pas dans la table, nous ne plaçons rien ni en A's, ni en A's + 0300. Nous prendrons dans le maitre programme CTA pour équivalent et nous attendrons de connaitre n pour faire une vérification de validité et placer le résultat définitif dans la table des symboles.

utilisation :

	8003	8002	800I
données	- symbole S-	00 0000 CTA	00 0000 -k -
résultats			
équivalent trouvé sortie en k	00 0000 éqi.	00 0000 CTA_1	00 0000 0000
pas équival. trouvé, sortie en k+1	00 0000 CTA-	00 0000 0000	00 0000 k +I

Dans le cas entrée 1, le nombre n doit se trouver dans la mémoire TRIO sous la forme 00 0000 -n -

ETUDE DES PSEUDO-CODES

A chaque pseudo-code correspond un sous-programme particulier, mais tous perforent une carte chargement du même type. Pour ne pas avoir à retirer du programme assemblé ces cartes devenues inutiles, elles porteront :

mot 1 : 00 I960 000I

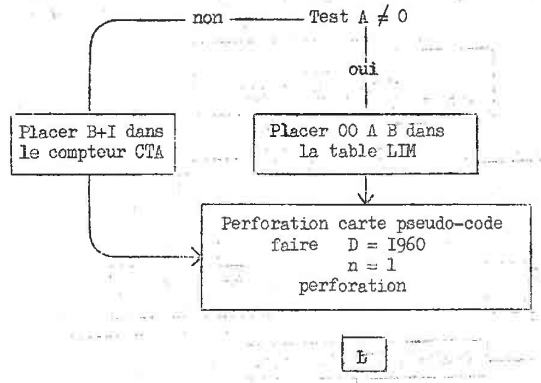
mot 2 : xx xxxx xxxx

... .

Lues par le programme de chargement du code de programmation, elles placeront en I960 le contenu du mot 2. Cette valeur sera effacée par une lecture suivante de toute façon cette mémoire n'est pas utilisée par la logique.

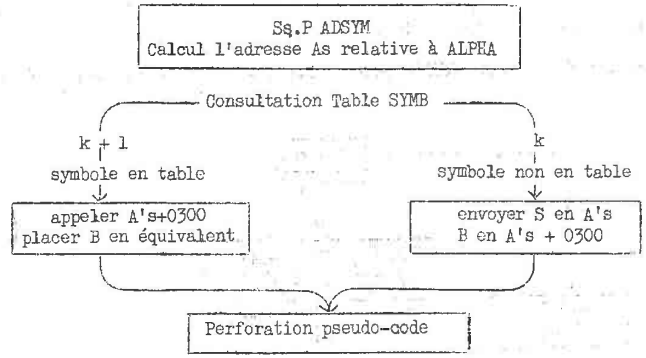
Les cartes pseudo-codes sont en général placées avant le programme à assembler, mais, à l'exception d'une seule, elles sont acceptées à tout moment par le programme CODA.

RESAB - Réservation absolue du bloc A - B



L : lecture de la carte suitante.

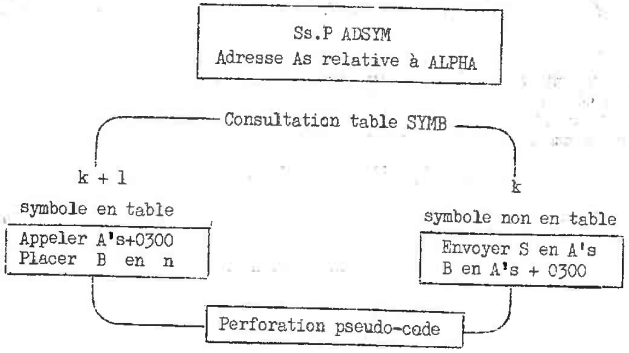
EQIAB - Equivalence absolue - Symbole ALPHA équivalent à B



N.B. - Le symbole A est déjà en table s'il a reçu une indication d'index n ou d'équivalence. L'équivalence sera remplacée par B et n conservé.

RESIN - Réservation pour indexage - Symbole ALPHA, n = B

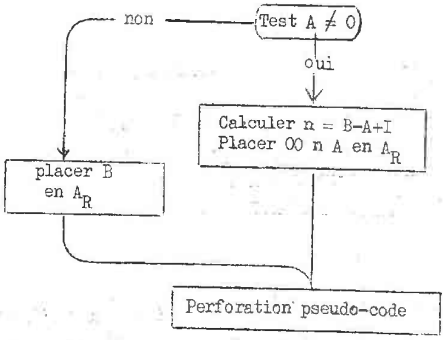
De même pour ce pseudo-code, A est en table s'il a reçu un équivalent, il sera conservé.



RESRG - Réservation régionale

Pour une donnée de type RESRG R0000 0005 , il faut placer B = 0005 dans la mémoire A_R sous la forme 00 0005 0000.

Pour une donnée du type RESRG R0I0I 0I05 , il faut placer en A_R : 00 0005 0I0I



LIBER - Libération du bloc de mémoire A - B

Ce pseudo-code ne peut être utilisé qu'en cours d'assemblage, s'il y a saturation par rapport aux réservations initiales. L'adresse A, première mémoire libre est envoyée dans le compteur d'adresse CTA. La borne supérieure B est introduite dans la table des limites sous la forme 00 -B - 0000 . Deux cartes LIBER ne peuvent se succéder immédiatement. Quand cette bande sera occupée, la rencontre de r_2 = 0000 provoquera un nouvel arrêt "saturation" . A ce moment une seconde carte de libération pourra être fournie à CODA.

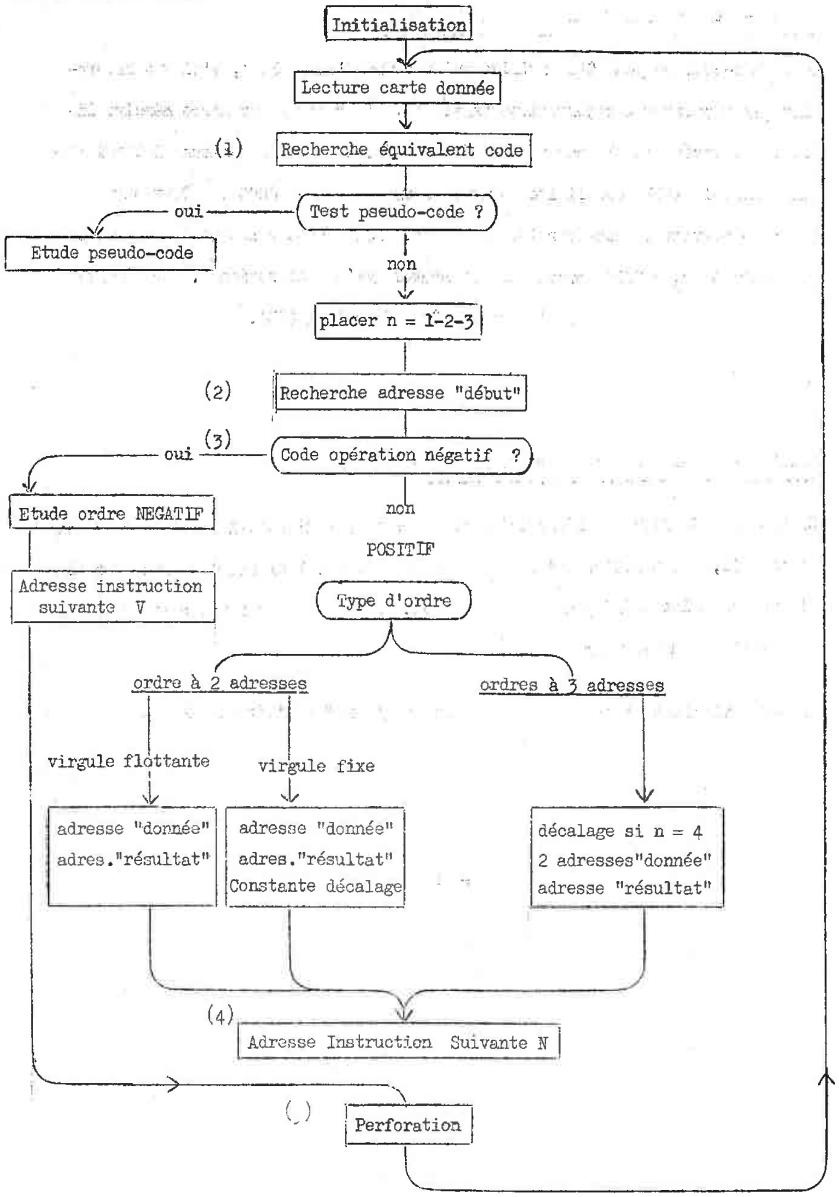
RZ.ASS - Remise à zéro pour assemblage

Ce pseudo-code provoque l'initialisation des tables d'équivalence et des mémoires de travail, avant lecture d'un programme symbolique. Pour faire se succéder plusieurs assemblages indépendants, il suffit, alors, de placer une carte RZASS devant chaque nouveau programme.

Il est équivalent d'effectuer un "départ programme" à l'adresse I900 -

* * *

Organigramme général



E Q U I V A L E N T D' U N O R D R E S Y M B O L I Q U E

Nous adopterons les notations suivantes pour les différentes parties de l'ordre symbolique donné.

DEBUT	CØDØP	i ALPHA	j BETTA	k GAMMA	SUITE
adresse de l'ordre	code operation	--- adresses facteurs ------			adresse instruction suivante

Pour chaque ordre donné, CODA doit former l'ordre numérique équivalent et commander la perforation de la carte correspondante. La présentation d'un ordre en code de programmation varie avec l'opération demandée. Les conventions sont plus uniformes en langage symbolique, mais la construction de l'ordre équivalent dépend du code opération.

L'organigramme de la page ci-contre nous indique les différentes parties de cette étude et leur enchainement. Leurs roles respectifs font l'objet de ce chapitre.

INITIALISATION

Elle prépare les différentes mémoires de travail avant l'assemblage d'un nouveau programme.

Sont remises à zéro, les trois cents mémoires 0000 à 0299 de la table des symboles, et les vingt quatre autres de la table régionale. Excepté pour la mémoire A_I qui contient toujours : 00 0009 I968 . La région I est en effet réservée à l'appel des registres d'index du code de programmation, ils sont au nombre de dix et le registre N°I a pour adresse tambour I968.

Les dix mémoires de la table des réservations reçoivent la constante 00 0999 0000

Le compteur CTA prend la valeur 000I considérée par CODA comme première mémoire libre du tambour. Il n'est pas possible de partir de l'adresse 0000 car les différents tests de zéro utilisés seraient ici en défaut.

Après lecture de la nouvelle carte donnée, le digit 8 qui commande la perforation A est envoyé en MOT I0.

ETUDE I - Code opération

Code donné symbolique. Un ordre de consultation de table nous indique l'adresse de ce symbole et de son équivalent dans la table des codes. Une comparaison des symboles donné et rencontré, détecte les erreurs-codes. L'étude ne peut pas se poursuivre indépendamment du code d'où arrêt-machine. La carte erronée est corrigée puis remise en place, en appuyant sur départ programme l'assemblage se poursuit.

Code donné absolu. La table des équivalents code est consultée pas à pas. Dans les deux cas, l'équivalent est placé en MOT 7 pour construire l'ordre numérique et dans une mémoire de travail pour les tests suivants.

Si le code trouvé correspond à un pseudo-code on caractérise une constante, CODA s'adresse directement au sous-programme voulu.

Si c'est un code opération, CODA recherche l'encombrement de l'ordre : codes négatifs n=1 ; codes positifs n=2 ; codes 05 et 06 n=3.

ETUDE 2 - Adresse de l'ordre

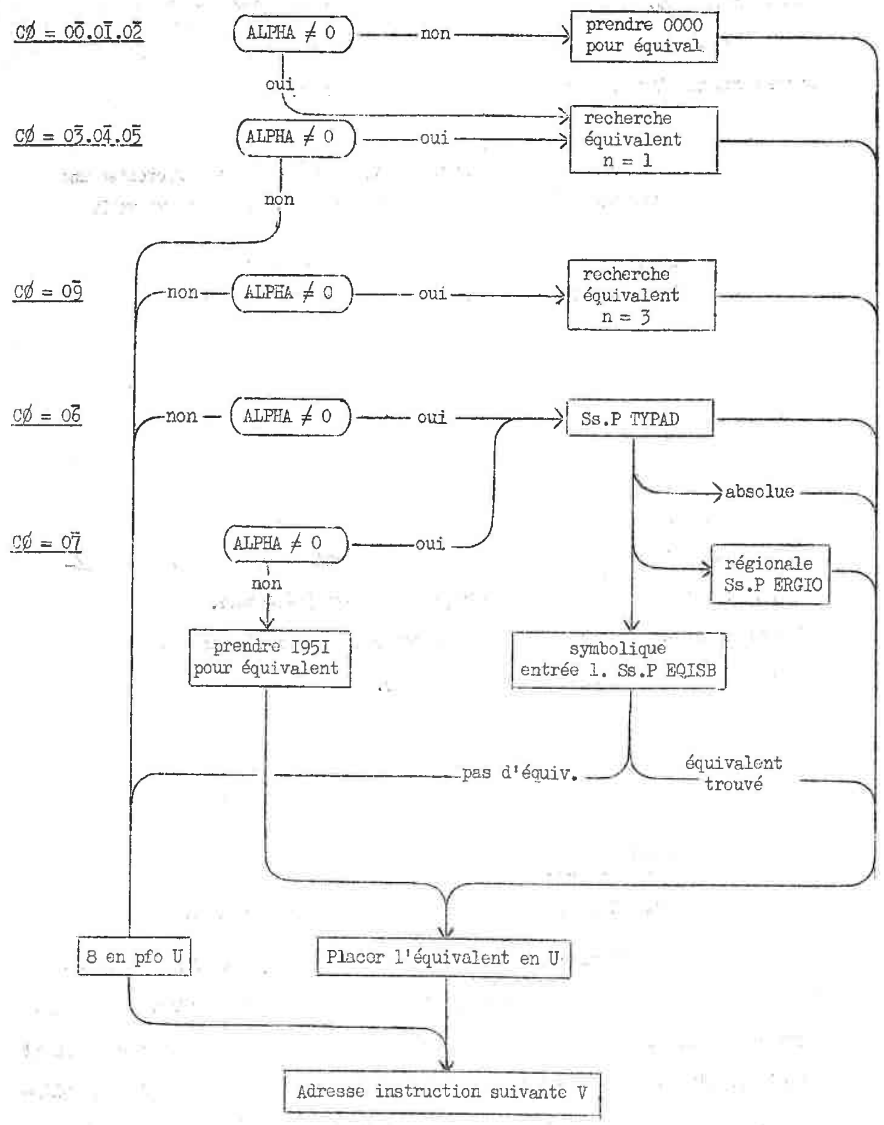
Elle a pour symbole DEBUT. CODA lui cherche un équivalent-tambour, et lui réserve le nombre n de mémoires demandé par les tests précédents. Cette étude peut dépendre de la carte précédente, nous indiquerons ces différents cas après étude de l'adresse instruction suivante. (page I09).

ETUDE 3 - Adresses facteurs

Une première distinction est faite par le signe du code opération.

Ordres négatifs - Ils occupent une seule mémoire et sont de la forme CO -U - -V - U : adresse facteur à quatre chiffres. V : adresse instruction suivante également à quatre chiffres. Les adresses en blanc sont acceptées mais les tolérances varient avec la signification de l'ordre. Leur étude est commandée par un aiguillage multiple à l'aide d'adresses régionales construites sur CØ. Toute erreur adresse place

Ordres négatifs



un 8 dans le mot de commande I986 pour que l'adresse voulue sorte en blanc.

- CØ = 0$\bar{0}$ sans opération
  CØ = 0$\bar{1}$ arrêt inconditionnel
  CØ = 0$\bar{2}$ initialisation des compteurs et registre d'index.

  Opérations logiques pour lesquelles l'adresse U n'est pas significative. ALPHA peut rester en blanc, CODA placera 0000 en U.

- CØ = 0$\bar{3}$ remise à zéro de la mémoire U
  CØ = 0$\bar{4}$ entrée dans le distributeur du contenu de U
  CØ = 0$\bar{5}$ entrée en langage machine à l'adresse U.

  U est une adresse facteur significative. ALPHA doit être "non blanc". L'absence de symbole est considérée comme une erreur-adresse.

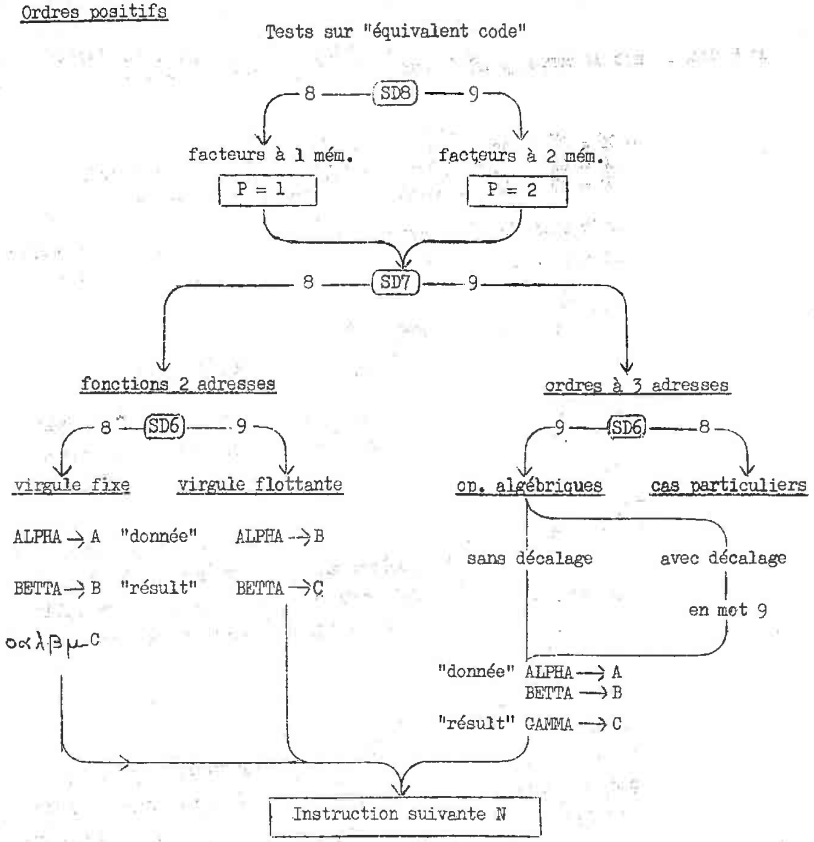
- CØ = 0$\bar{6}$ perforation dans la bande U
  U indique la zone de mémoires grises à perforer par l'ordre langage machine 7I-U- . ALPHA doit être non blanc et au préalable affecté. Ces conditions non réalisées entrainent une erreur - adresse.

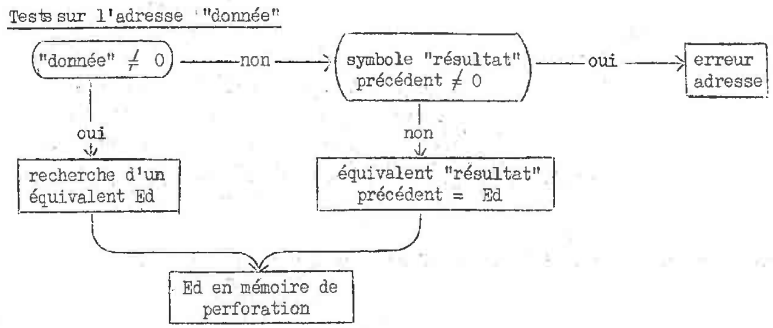
- CØ = 0$\bar{7}$ lecture dans la bande U
  En code de programmation le programme peut utiliser la bande I95I pour les lectures en cours d'exécution. Si ALPHA est blanc, ou symbolique, non affecté au préalable, CODA lui donnera pour équivalent l'adresse I95I. Ceci n'était pas possible pour l'ordre de perforation, car la bande I977 n'est pas disponible.

- CØ = 0$\bar{9}$ fin de séquence. Aller à l'adresse U en logique, à l'adresse V en autoperforation. Le symbole ALPHA est l'adresse d'un ordre en code. S'il n'a pas encore reçu d'équivalent, nous devons lui réserver n= 3 mémoires, encombrement maximum d'un ordre CDP. ALPHA en blanc provoque une erreur adresse.

Ci-contre organigramme d'exécution qui regroupe les différents cas.

Ordres positifs





Ordres positifs - Ils forment la partie essentielle du code de programmation et groupent : les calculs de fonction, les opérations algébriques en virgule fixe et flottante. Les digits 8 et 9 de l'indicatif occupent les positions 6.7. et 8 de la mémoire "équivalent code". Chacun d'eux correspond à une condition particulière.

Position 8

8 - ordre en virgule fixe ou flottante, en simple précision. Une seule mémoire sera réservée pour les adresses facteurs A, B et C.

9 - ordre en virgule flottante double précision. Deux mémoires seront réservées pour chaque facteur.

Position 7

8 - calcul de fonction. Deux adresses facteurs seulement. ALPHA : adresse de l'argument. BETTA : adresse de la fonction ou adresse "résultat".

9 - opérations logiques ou algébriques qui comportent trois adresses facteurs significatives.

Position 6

Elle est testée en parallèle après la position 7 d'où quatre embranchements.

8 8 - Le calcul de fonction demandé est en virgule fixe. La position d'adresse C contient des indications de décalage :

L'ordre numérique à former a pour structure :

CØ NNN i A j B 0

A : argument B : fonction

8 9 - Le calcul de fonction est en virgule flottante. L'adresse C reste en blanc. L'ordre numérique a pour structure :

00 NNN OIxxx j A k B

OIxxx : indicatif de fonction
A : argument B : fonction
La mise en place des équivalents diffère dans ces deux cas.

9 9 - Opération algébrique : addition, soustraction, multiplication, division. Structure numérique :

CØ NNN i A j B k C

L'adresse "résultat" correspond ici au symbole C. A et B sont les deux adresses "donnée".
Les deux cas d'opérations avec décalage prolongent cette partie. Il faut construire le troisième mot de l'ordre et commander sa perforation :

3λ 000μ I939 multiplication
3λ 000μ I99I division

Les codes opérations symboliques sont MULλμ et DIVλμ
Le mot 3 est construit par le tableau de connexion au moment de la perforation. L'aiguillage voulu a été préparé par le test sur CØ qui indiquait n = 3. Ces deux ordres seuls occupent trois mémoires.

Pour tous ces ordres il est possible de laisser les adresses facteurs en blanc. A condition, toutefois, de respecter l'ordre des cartes symboliques. Une adresse "donnée" en blanc recevra le même équivalent que l'adresse "résultat" de l'ordre précédent si celle-ci était en blanc ; si elle ne l'était pas, il y a erreur sur l'adresse "donnée".

9 8 - Certains ordres logiques sont étudiés de façon isolée, ils ne correspondent exactement à aucune des structures précédentes.

- Ordre de bouclage : se traduit en numérique par

9i NNN 00-n 0 B 0 00-n

Il signifie faire n fois la séquence d'ordre qui débute en B, uti-

liser le compteur numéro α, modifier le registre d'index i de la quantité m. Le nombre n peut être indiqué au cours de l'exécution du programme par un ordre de modification de bouclage. Au cours de l'affectation : ALPHA doit être absolu ou en blanc, dans ce dernier CODA inscrira A = 0000. BETTA doit être non blanc, si son équivalent n'est pas en mémoire, lui réserver trois mémoires. GAMMA est absolu, ou blanc pour un bouclage sans indexation, CODA prendra C = 0000.

- Ordre de test : de forme numérique

07 NNN 0 A 0 B 0 C

aiguille le programme en N, A ou B suivant que le contenu de C est négatif, nul, ou positif. Pour l'affectation : ALPHA et BETTA sont des adresses "instruction suivante", trois mémoires en séquence leur seront réservées. GAMMA non blanc recevra une seule mémoire. Pour équivalent, GAMMA adresse blanche, prendra l'équivalent de l'adresse "résultat" précédente si celle-ci était en blanc.

- Test de 8 et de 9 : de forme numérique

08 NNN 0 A 0000n 0 C

aiguille le programme en N, ou A suivant la présence d'un 8 ou d'un 9 en position n de la mémoire C. ALPHA reçoit trois mémoires, BETTA est absolu. Les conditions sur GAMMA sont les mêmes que pour l'ordre précédent. (L'exécution de l'affectation également).

- Perforation : de forme numérique

49 NNN 0000n j B k C

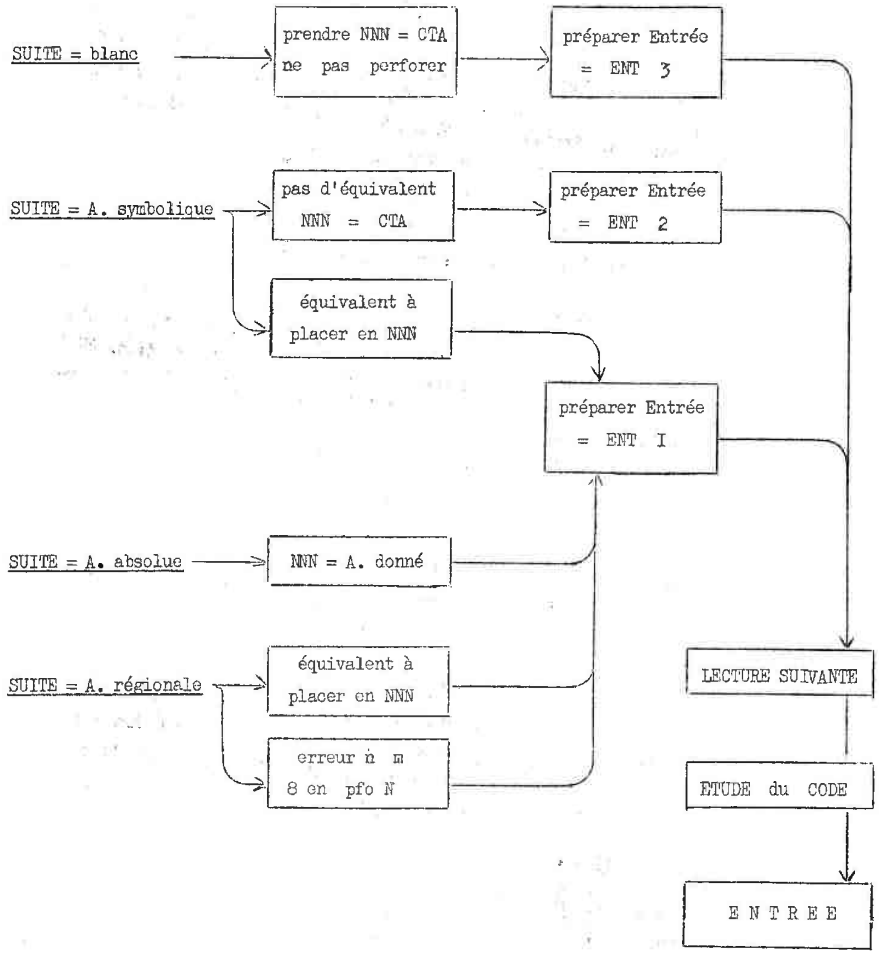
Commande la perforation du bloc B - C en n mots par carte.
Si ALPHA est en blanc, CODA imposera n = 5 .

- Modification ordre de bouclage : de forme numérique

00 NNN OI937 j B k C

Remplace le nombre n de l'ordre de bouclage placé à l'adresse C par le contenu de la mémoire B. CODA réserve une mémoire à BETTA non blanc, une adresse blanche recevra l'équivalent de l'adresse "résultat" précédente. Trois mémoires seront réservées pour C, non blanc.

Adresse instruction suivante



Pour tous les ordres traduits par CODA, le symbole de l'adresse "résultat" et son équivalent sont mis en mémoire pour être disponibles au cours de l'étude suivante.

ETUDE 4 - Adresse instruction suivante

Les conventions suivantes sont les mêmes pour tous les ordres. Pour un ordre positif l'équivalent se place en NNN, pour un ordre négatif il se place en V. Deux programmes distincts sont rédigés pour cette seule raison. L'adresse peut être ou non en blanc. Trois cas sont à distinguer pour la recherche de son équivalent.

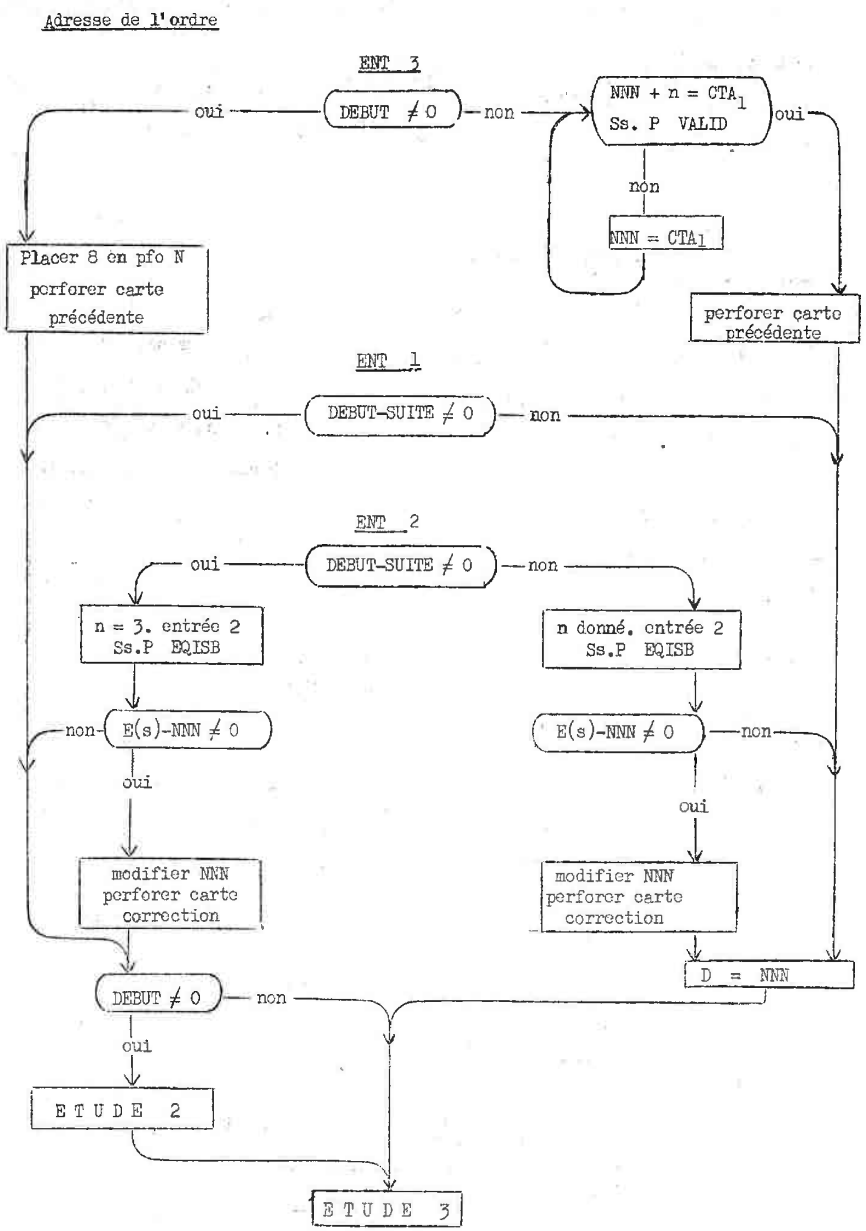
1 - SUITE non blanc - Un équivalent est déjà affecté à ce symbole.
Il est placé en mémoire de perforation en NNN. La carte résultat complète est perforée

2 - SUITE non blanc - Le symbole n'est pas encore affecté.
Nous ne pouvons pas prévoir le nombre exact de mémoire à lui réserver, et lui accorder systèmatiquement trois mémoires perdrait trop de place sur le tambour. Alors, CODA place en N l'adresse inscrite dans CTA, et perfore la carte résultat sans autre vérification. La carte suivante est lue, son code opération étudié, d'où "n" nombre de mémoires que cet ordre occupera.
Le symbole SUITE et le nouveau symbole DEBUT sont alors comparés.

- Ces symboles sont identiques. Leur affectation est commandée par l'entrée 2 du sous-programme EQISB avec pour données : CTA = NNN et "n". Soit E(s) l'équivalent obtenu.

  Si E(s) = NNN la carte précédente est exacte ; E(s) est placé en adresse D de l'ordre actuel. On passe à l'étude 3, adresses facteurs.

  Si E(s) $\neq$ NNN la carte précédente est erronnée. Une seconde car-



te sera perforée pour la rectifier. Elle portera E(s) en adresse instruction suivante.

- Ces symboles sont différents, n = 3 sera imposé pour l'affectation du symbole SUITE, avec possibilité de rectification comme dans le cas précédent. On passera à l'étude 2, équivalent de DEBUT.

3 - SUITE en blanc - Le problème se rapproche du précédent.

CODA place en NNN l'adresse inscrite dans le compteur CTA, mais la carte résultat n'est pas perforée immédiatement. Une telle adresse en blanc appelle obligatoirement une carte suivante à adresse DEBUT également en blanc. Cette carte sera lue et son code opération testé pour connaitre l'encombrement "n" de l'ordre.

Alors le sous-programme VALID peut vérifier que le groupe de mémoires CTA, CTA+n-1 est libre. Si oui, la carte relative à l'ordre précédent est perforée. Sinon, l'adresse NNN est rectifiée avant cette perforation.

La valeur perforée de NNN devient adresse D du dernier ordre lu, CODA passe à l'étude 3 de cet ordre.

Si DEBUT est non blanc à la lecture, une erreur adresse est détectée pour NNN (qui sort en blanc). CODA passe à l'étude 2, adresse de l'ordre.

Chacun des trois cas notés, prépare l'aiguillage qui lui est propre, pour l'exécution de l'assemblage après lecture de la carte suivante. Ces ordres sont également rédigés deux fois : l'une pour permettre des modifications sur NNN, l'autre sur V.

* * *

UTILISATION DE CODA

Le programme CODA est disponible sous forme n-mots par carte, précédé de son propre programme de chargement. Pour effectuer un assemblage :

- placer le tableau de connexions voulu,
- placer en lecture : le paquet CODA suivi du programme symbolique.

Commander un départ programme par l'ordre 70.I95I.XXX affiché au pupitre.
Une fois le programme CODA lu, il prendra en charge la lecture des cartes symboliques et la perforation des cartes résultats .

Pour effectuer un nouvel assemblage, CODA étant chargé. Faire précéder les ordres symboliques d'une carte pseudo-code RZASS et s'adresser en I950 (par "restauration générale" et "départ programme") ou, les placer directement dans le magasin de lecture et s'adresser en I900.

Différents arrêts possibles

Erreur code opération : arrêt par OI.0333.I950, pour un code symbolique erroné ou qui n'existe pas dans la table CODA utilisée.
Retirer la carte et la rectifier, le programme se poursuit en I950 en appuyant seulement sur départ programme.

Saturation de la table des symboles : arrêt par OI.9999.9999.
Il est possible de poursuivre l'assemblage en conservant seulement quelques équivalents de symboles. Perforer pour chaque symbole une carte pseudo-code EQIAB, ou RESIN et la placer devant la partie du programme symbolique non assemblée. Effectuer un départ programme en "I420". La table des symboles sera initialisée

et CODA passera à la lecture des pseudo-codes donnés.

Saturation du tambour : arrêt par OI.6666.6666.
C'est une saturation par rapport aux reservations données, ou à 0999 considérée comme dernière mémoire libre du tambour. Pour que l'assemblage se poursuive : perforer une carte pseudo-code LIBER qui indique à CODA une région libre sur le tambour, la placer devant les cartes symboliques non lues, et effectuer un départ programme en I950.

Remarques

CODA accepte toutes les adresses absolues. Mais, seuls les codes opérations positifs sont acceptés sous forme absolue.
Le commutateur "Dépassement de capacité" doit être placé sur "détection" .

* * *

UNIVERSITÉ NANCY BIBLIOTHÈQUE

VU et APPROUVE

NANCY le 8 JUIN 1961

Le Doyen de la Faculté
des Sciences :

M. ROUBAULT

VU et
PERMIS D'IMPRIMER
NANCY le 10 Juin 1961

Le Recteur

Président du Conseil de
l'Université de NANCY

P. IMBS