الجمهورية الجزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعليم والبحث العلمي
Ministère de l'Enseignement et de la Recherche Scientifique

المدرسة الوطنية للعلوم الهندسية
ECOLE NATIONALE POLYTECHNIQUE D'ALGER

DEPARTEMENT D'ELECTRONIQUE

PROJET DE FIN D'ETUDES

INGENIEUR D'ETAT EN ELECTRONIQUE

THEME

Programmateur de Proms et de Reproms Conversationnel 2 eme Partie : Software

Proposé par :
Mr et Mme HAMAMI

Etudié par :
Moussa BICHARI
Allal LARBI

Promotion Janvier 1984

الجمهورية الجزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعليم والبحث العلمي
Ministère de l'Enseignement et de la Recherche Scientifique

المدرسة الوطنية للعلوم الهندسية
ECOLE NATIONALE POLYTECHNIQUE D'ALGER

DEPARTEMENT D'ELECTRONIQUE

PROJET DE FIN D'ETUDES

INGENIEUR D'ETAT EN ELECTRONIQUE

THEME

Programmateur de Proms et de Reproms Conversationnel 2 eme Partie : Software

Proposé par :
Mr et Mme HAMAMI

Etudié par :
Moussa BICHARI
Allal LARBI

Promotion Janvier 1984

DEDICACES.

Ce modeste travail est dédié :

— A mes chers parents, pour leur sacrifice et leur courage,
— A mes frères et sœur,
— A la mémoire de mon grand-père maternel,
— A tous mes amis et mes collègues de promotion.

Allal LARBI.

o o o

Je dédie cet ouvrage :

— A mes parents, pour leur sacrifice,
— A mes frères et sœurs, pour leur aide morale et matérielle,
— A mes cousins (es),
— A tous mes amis (ies).

Moussa BICHARI.

REMERCIEMENTS

— Nous tenons à remercier vivement notre promoteur Mr. HAMAMI, chef du département de hardware à l'ENSI (ex-CNI), ainsi que Mme. HAMAMI, pour l'aide et les précieux conseils qu'ils nous ont prodigués tout au long de l'élaboration de ce projet.

— Nos remerciements s'adressent aussi à tout le personnel de l'ENSI pour leur assistance et leurs encouragements.

— Que tous les professeurs qui ont contribué à notre formation trouvent dans cet ouvrage l'expression de notre profonde gratitude.

SOMMAIRE

chap.	TITRE	Page
—	BUT DU PROJET	1
—	INTRODUCTION	2
chap.1	RAPPELS SUR LE "HARDWARE" DU SYSTEME	4
1.1.	Carte micro-ordinateur	5
1.2.	Carte alimentation programmable	8
1.3.	Carte parallèle	9
1.4.	Carte mémoire	12
1.5.	Carte support pour les REPROMS (Intel, Tms)	13
1.6.	Carte support pour les PROMs bipolaires	15
chap.2	ETUDE DES PROMS ET REPROMs UTILISEES	19
2.1.	Introduction	19
2.2.	Les REPROMs Intel et Tms	19
2.3.	Les PROMs bipolaires de la série "14/18"	21
2.4.	Les PROMs bipolaires de la série "24/28"	24
chap.3	ORGANISATION DE LA ZONE MEMOIRE DU SYSTEME	28
3.1.	Tableau d'adressage	28
3.2.	organisation des différentes zones mémoires	29
chap.4	PROGRAMMATION	38
4.1.	Introduction	38
4.2.	Méthode de programmation	41
4.2.1	Initialisation du système	42
4.2.2	Réception de la commande	46
4.2.3	Analyse de la commande	50
4.2.4	Exécution de la commande	65
chap.5	ESSAIS ET MISE AU POINT DU PROGRAMME	105
5.1.	Introduction	105
5.2.	Enregistrement du programme	105
5.3.	Phase d'assemblage	105
5.4.	Correction des erreurs de syntaxe	105
5.5.	Exécution du programme	106
—	CONCLUSION	107
—	ANNEXE	
—	REFERENCES BIBLIOGRAPHIQUES	

But du Projet

Le but de notre travail est l'étude et la mise au point de la partie "SOFTWARE" d'un système programmateur de PROMs et de REPROMS conversationnel dont le "HARDWARE" a été conçu et réalisé par deux étudiants de la promotion de juin 1983.
Une fois achevé, le système dont il est question sera capable d'exécuter un bon nombre de commandes dont notamment : la programmation automatique des PROMs et des REPROMS, la vérification des programmes introduits, et dans certains cas, l'émission de messages d'erreur quand les commandes ou les données n'ont pas un format correct.

Le système doit également avoir les caractéristiques suivantes :

- L'autonomie : Le système doit pouvoir être utilisé seul ou en liaison avec un ordinateur.
- La souplesse : Le système doit avoir des possibilités d'extension au point de vue des PROMS qu'on peut programmer, et des liaisons banalisées avec un écran ou avec un ordinateur par jonction RS 232 C.

Introduction

La programmation manuelle des PROMs et des REPROMs étant souvent trop fastidieuse, il est de nos jours beaucoup plus intéressant de penser à réaliser des systèmes capables d'assurer une programmation automatique de celles-ci. Notre travail se situe justement dans ce contexte :

Il s'agit de réaliser un ensemble de programmes, de les tester et de les mettre au point afin de les implanter dans un système programmateur de PROMs et de REPROMs conversationnel qui a été conçu au cours du semestre de juin 1983 par deux étudiants de l'E.N.P. en guise de projet de fin d'études.

Il nous paraît donc indispensable de donner un bref aperçu sur le "hardware" de ce système avant d'entamer l'étude du "software" proprement dit.

Le système en question est bâti autour d'une carte UC à microprocesseur de la famille 6800 de MOTOROLA.

Cette carte dispose de :

* deux voies de communication avec l'extérieur :

— 1 voie sérielle au standard RS 232C pour communiquer soit avec une console de visualisation, soit avec un micro-ordinateur.

— 1 voie parallèle bidirectionnelle pour communiquer soit avec une imprimante, soit avec un dispositif clavier - afficheurs.

* une zône mémoire PROM pour le stockage des programmes de gestion du système. Cette zône a une capacité maximale de 8 K-octets.

* une zône RAM dédiée exclusivement aux opérations du CPU, à la pile de sauvegarde et à la zône de travail; sa capacité est de 2K-octets.

Une étude plus approfondie de cette carte et des autres cartes du système fera l'objet du chapitre suivant.

Chapitre 1 Rappels sur le "Hardware" du Système

Le programmateur étudié se compose essentiellement des cinq cartes suivantes :

_ Carte unité centrale, assurant la gestion du système et permettant la communication avec un ordinateur ou une console de visualisation.

_ Carte support sur laquelle doit se trouver l'organe à programmer : PROM ou REPROM.

_ Carte parallèle permettant, grâce à des circuits d'entrée/sortie, d'accéder à la carte support en écriture ou en lecture.

_ Carte alimentations programmable délivrant, à partir de combinaisons binaires adéquates attribuées aux C.N.A., les tensions d'alimentation nécessaires aux différentes PROMS et REPROMS à programmer.

_ Carte mémoire comportant 4 zônes de RAM statique de 4 k.octets chacune, et permettant de loger temporairement les programmes utilitaires, de les modifier ou de les retoucher avant de les stocker définitivement dans les PROMS ou REPROMS à programmer.

Enfin, un bloc d'alimentation générale sert à délivrer aux différents circuits les tensions nécessaires à leur fonctionnement.

Nous donnons ci-dessous un synoptique simplifié du système :

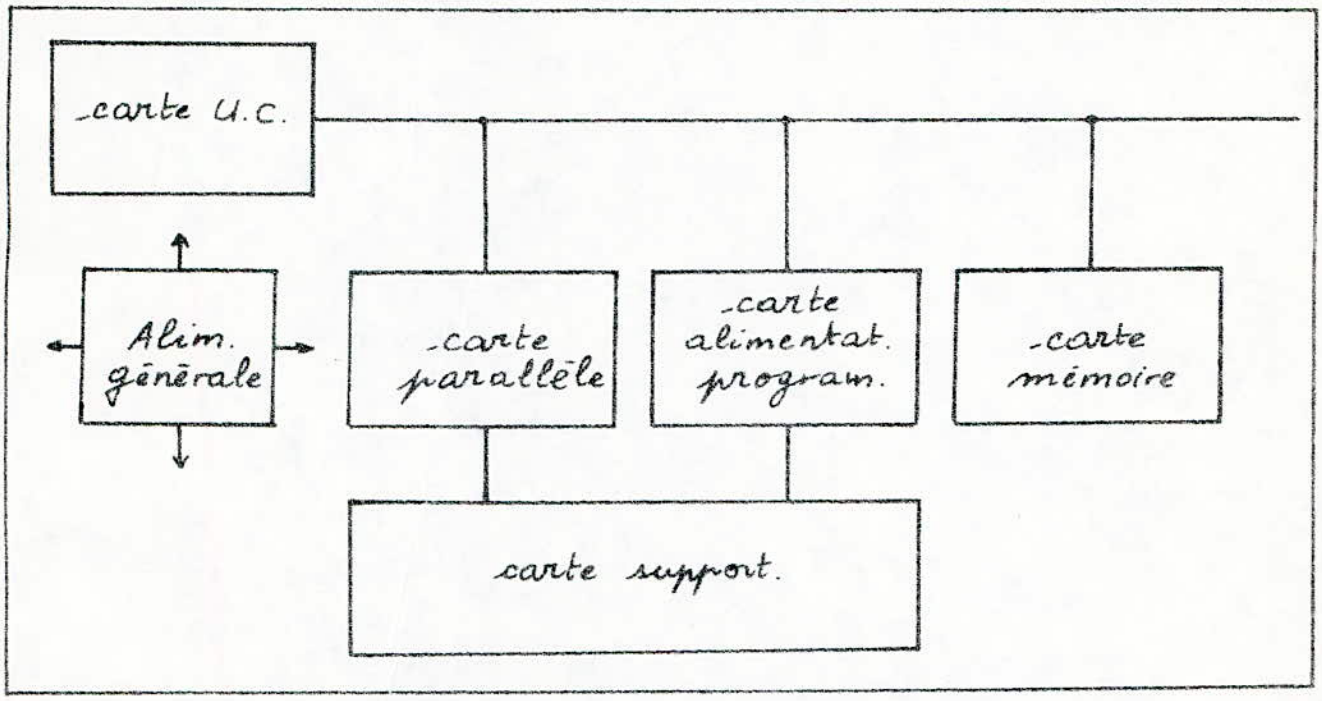


Fig. 1 : synoptique du programmateur.

1.1 : Carte micro-ordinateur (ou unité centrale) :

1.1.1 : Etude théorique :

Cette carte, comme toute carte micro-ordinateur pour un système minimum, comprend en général : une unité centrale et une horloge associée, une logique de décodage d'adresse, des mémoires mortes et des mémoires vives, des circuits d'interface parallèles et séries, ainsi que des circuits permettant d'étendre les possibilités de la carte et assurant le raccordement sur le bus du système où pourront être connectées d'autres cartes.

Le synoptique d'une telle carte est le suivant :

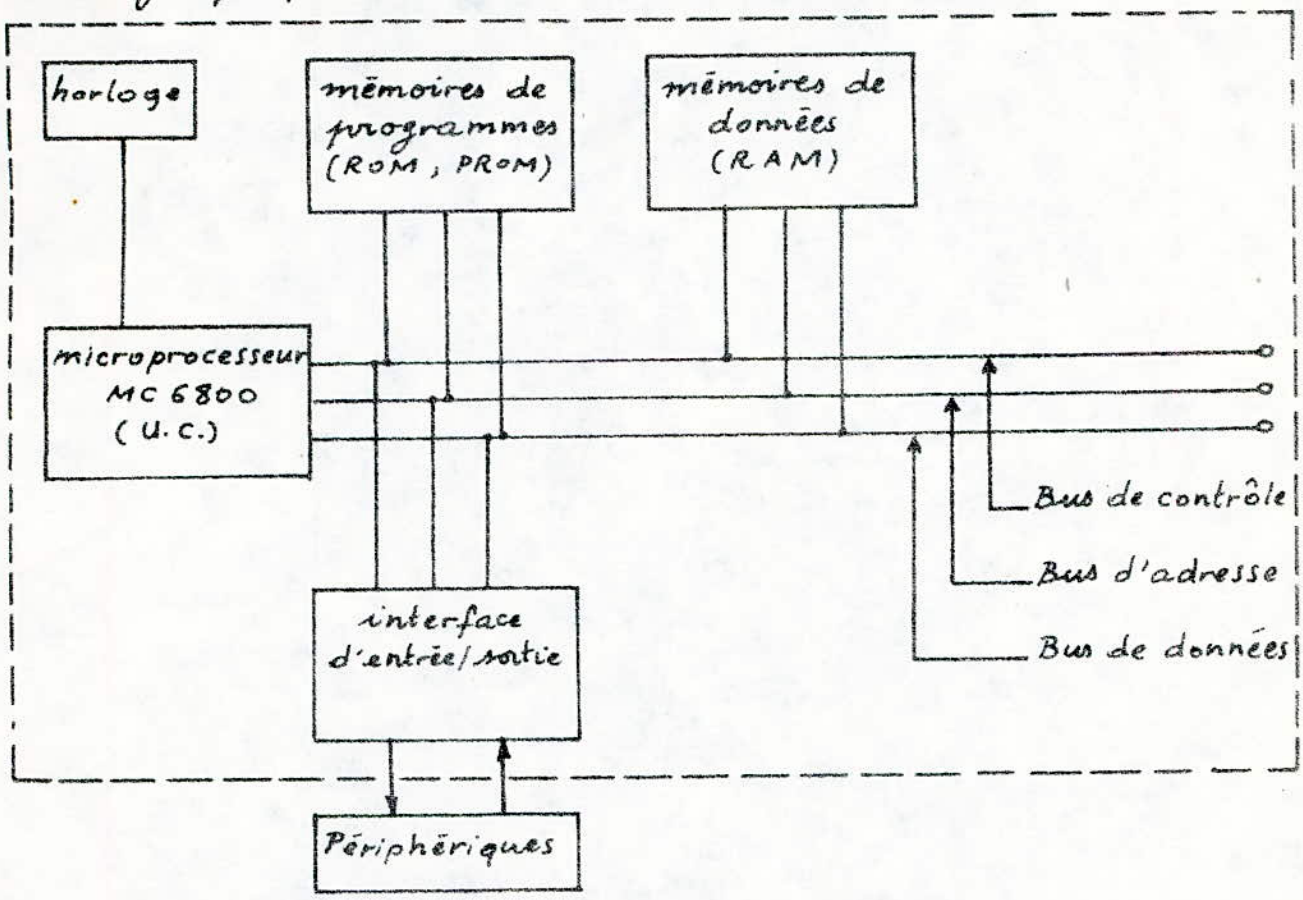


Fig. 2 : synoptique de la carte U.C.

1.1.2 : Composants utilisés :

La carte unité centrale comporte les composants suivants :

- Le microprocesseur MC 6800.
- Une mémoire RAM MB. 8128 de capacité 2 K-octets.
- Deux mémoires EPROM 2732 de capacité 4 K-octets.
- Un interface adaptateur pour communications sérielles asynchrones ACIA (MC 6850).
- Un interface adaptateur pour périphériques PIA (6821)
- Une horloge MC 6875A pour le MC 6800.
- Un générateur d'horloge pour l'ACIA, MC 14411.

- un décodeur SN 74 LS 139 N.
- un circuit NAND SN74LS 00N.
- trois circuits "OU EXCLUSIF", SN 74 LS 86 N.
- un émetteur de ligne MC 1488.
- un récepteur de ligne MC 1489.
- un connecteur G64, et un autre de norme RS232.
- Buffer SN74LS 242N pour les données.
- Buffer SN74LS244N pour les adresses.
- Buffer SN74LS 240N pour les signaux de contrôle.

1.1.3 : Fonctionnement :

Les mémoires EPROM de cette carte contiendront l'ensemble des programmes de gestion que nous nous proposons de réaliser tandis que la zône RAM de 2 k-octets sera réservée à la gestion de la pile et constituera un espace de travail pour le microprocesseur.

Notre carte micro-ordinateur comporte deux circuits d'entrée/sortie assurant l'adaptation entre le microprocesseur et l'"extérieur".
Ainsi, il pourra communiquer en parallèle à l'aide du PIA MC 6821, avec une imprimante ou un clavier hexadécimal par exemple. Il le fait en série à l'aide de l'ACIA MC 6850 et du connecteur RS 232.
Dans notre cas, le microprocesseur sera utilisé aussi pour piloter le C.N/A. de la carte alimentation programmable.

1.2: Carte alimentation programmable :

1.2.1 : Etude théorique :

Les tensions à appliquer à la PROM (ou REPROM) à programmer sont variables d'une mémoire à l'autre. Pour avoir la tension désirée, il suffit de transférer la combinaison binaire requise du MPU vers un convertisseur numérique / analogique (C.N.A). Ce transfert est unidirectionnel. Parmi les PROMs ou REPROMs utilisées, il y en a qui nécessitent deux alimentations, d'autres en demandent trois. Ceci a suggéré de concevoir trois alimentations programmables.

1.2.2 : Composants utilisés :

- Buffer SN 74 LS 240 N (U_1).
- Buffer SN 74 LS 244 N (U_2, U_3).
- Bistable latches SN 74 LS 374 N (U_4, U_5, U_6).
- Amplificateurs opérationnels MC 1747 CL.

1.2.3 : Fonctionnement :

Le MPU transmet à travers son bus de données les combinaisons binaires voulues pour attaquer le convertisseur numérique / analogique. Le décodeur est validé lorsque $V_{PA\,ext}$ est à 0, A_2 et A_3 à 1. Ces niveaux valident aussi le buffer 74LS240 par l'intermédiaire de G1.
Les adresses A_0, A_1 sélectent les sorties Y_0, Y_1 ou Y_2 qui délivrent un niveau logique bas.

Ces dernières sont ensuite inversées et déclenchent une écriture dans le latche correspondant par application du signal formé à l'aide des circuits "NAND", (voir carte alimentation programmable en annexe). Ainsi, suivant A_0 et A_1, ce sera l'un des "clocks" C_1, C_2, ou C_3 qui fonctionnera. Ces latches travaillent sur leur front montant et libèrent ainsi les données qui vont au convertisseur numérique / analogique.

En conclusion, cette carte permet d'obtenir des tensions variables de 0 à 25 V selon notre désir, et ceci par simple changement de la combinaison binaire au cours de la programmation envisagée.

1.3 : Carte parallèle :

1.3.1 : Etude théorique :

Cette carte comporte des circuits d'entrée / sortie permettant d'accéder à la carte support en écriture ou en lecture. Toutes les liaisons sont du type parallèle, leur intérêt réside dans la grande vitesse de transfert autorisée, mais ne permettent pas de liaisons à grande distance compte tenu du nombre de conducteurs.
Une fois bufférisé, le bus de données permettra aux PIA de cette carte de transmettre les données à la PROM qui se trouve sur la carte support, de l'adresser, et de la contrôler.

1.3.2 : Composants utilisés :

- 2 buffers SN74LS240N.
- 2 buffers SN74LS244N.
- 3 PIAs MC68B21P.
- 1 porte NAND SN7400N.
- 1 inverseur SN7404N.

1.3.3 : Fonctionnement :

Cette carte joue le rôle d'un contrôleur de périphérique. Ce dernier, dans notre cas, n'est rien d'autre que la carte support des PROMs à programmer.

Comme nous l'avons mentionné plus haut, cette carte comporte trois PIAs dont le rôle de chacun est le suivant :

- Le PIA3 permet le transfert bidirectionnel des données entre le microprocesseur et la PROM à programmer, ceci se fait par l'intermédiaire de son port A qui sera programmé en entrée ou en sortie selon le cas. Le Port B est prévu pour une éventuelle extension de la carte.
- Le PIA4 nous forme les adresses de la PROM à programmer ou à lire. Le port A contiendra les adresses $A_0 - A_7$, et le port B les adresses $A_8 - A_{15}$. Les lignes d'adresse $A_{12} \div A_{15}$ sont prévues pour une éventuelle extension. Ce PIA sera programmé en sortie.

— Le PIA5 permet de fournir les signaux de contrôle nécessaires à la carte support. Ces signaux sont les suivants :

$PA_0 = \overline{OE}$; $PA_1 = \overline{CE}$: Ces deux signaux sont utilisés pour la PROM étalon en mode lecture ;

$PA_2 = \overline{OE}$ (pr) ; $PA_3 = \overline{CE}$ (pr) : Ces deux derniers signaux sont utilisés pour la PROM à programmer en mode programmation ou en mode lecture.

Ce PIA permet aussi, par son port B, de lire le switch à 8 positions afin de déterminer le type exact de la PROM à programmer.

Le tableau suivant donne les adresses d'implantation de ces trois PIAs dans notre système :

Adr. / PIA	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	adr. hexa.
PIA3	1	1	0	1	1	1	0	1	0	0	0	0	1	0	0	0	DD08
PIA4	1	1	0	1	1	1	0	1	0	0	0	0	0	1	0	0	DD04
PIA5	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	DD00

En conclusion, la carte parallèle sera réalisée une fois pour toutes, et on pourra lui connecter n'importe laquelle des cartes support (carte support pour les Reproms TMS et INTEL, ou carte support pour les Proms bipôlaires), qui contiendra le type de PROM à programmer ou à lire, avec son réseau de contrôle.

1.4 : Carte mémoire :

1.4.1 : Etude théorique :

Cette carte comporte une zône mémoire RAM de capacité 16 K. octets. Pour lire ou écrire un mot de 8 bits dans la RAM, il faut :

— envoyer l'adresse par le bus d'adresses.

— envoyer un signal de lecture ou d'écriture ; c'est le microprocesseur qui effectue cette dernière fonction lorsqu'il a reconnu une instruction d'écriture.

— la donnée est véhiculée par le bus de données.

Dans le cas d'une PROM, le processus est le même, mais la seule opération possible est une lecture.
Toutefois, tout programme nécessite d'être mis au point et donc d'être retouché, cela entraîne deux conséquences pour la phase de mise au point :

— Le programme ne doit pas être stocké dans une PROM ni même dans une REPROM. Pour permettre des modifications de programme faciles et rapides, on le mettra alors provisoirement dans la RAM avec les données et les résultats ; la capacité de la RAM sera prévue en conséquence.

— Il faudra pouvoir, à l'aide d'un clavier, rentrer le programme dans la RAM, en effacer certaines parties, en ajouter d'autres, faire exécuter le programme pour

l'essayer, ... etc.

Cela nécessite un programme spécial appelé moniteur, et qui fait partie du " système de développement " (micro-ordinateur destiné à faciliter la mise au point d'un système à microprocesseur.).

1.4.2 : Description de la carte mémoire :

La carte mémoire utilisée est une carte toute faite. Elle consiste en un module GESRAM-2N, comportant quatre zônes de mémoire statique de 4 K-octets chacune, adressables individuellement. La capacité totale ainsi disponible est de 16 K-octets. 12 lignes d'adresse sont donc nécessaires pour définir l'adresse de chaque mot dans l'un des modules RAM.

1.5 : Carte support pour les REPROMs (intel, Tms) :

1.5.1 : Etude théorique :

Elle servira de support pour les REPROMS (intel, Tms), qui seront étudiées dans le chapitre 2, et sera vue par la carte U.C. comme un périphérique adressable à travers la carte parallèle.

1.5.2 : Composants utilisés :

Elle comporte les éléments suivants :

— deux supports (2x12) pour les REPROMs.
— cinq switches à deux positions.
— un switch à 8 positions (ou 8 éléments).

1.5.3 : Fonctionnement :

• Les supports (2x12) sont utilisés pour supporter les différents boîtiers des REPROMs. Ils sont au nombre de deux pour chaque type de REPROM. Le premier supportera l'une des 2 REPROMs, (REPROM de référence), qui sera en mode lecture ; et le second pour la REPROM en mode programmation.
Il faut noter aussi que tous les boîtiers des REPROMs utilisées dans cette carte sont des boîtiers à 24 broches.

• Vu que les REPROMs utilisées (intel, et Tms) présentent quelques différences au niveau des broches, alors on est amené à utiliser des switches à deux positions afin de sélecter les deux REPROMs, puisque ces dernières ne possèdent pas les mêmes caractéristiques en mode lecture et en mode programmation.

• Le switch à 8 éléments permet d'identifier la REPROM à programmer, en positionnant une valeur binaire sur ses éléments. Ce seront des valeurs binaires choisies arbitrairement. On adoptera les valeurs suivantes :

REPROM :	numéro. switch.	REPROM :	numéro. switch.
TMS 2516	0 0 0 0 0 0 0 0	INTEL 2732	0 0 0 0 0 1 0 0
TMS 2532	0 0 0 0 0 0 0 1	INTEL 2732A	0 0 0 0 0 1 0 1
TMS 2564	0 0 0 0 0 0 1 0	INTEL 2758	0 0 0 0 0 1 1 0
INTEL 2716	0 0 0 0 0 0 1 1	INTEL 2764	0 0 0 0 0 1 1 1

1.6 : Carte support pour les PROMs bipôlaires :

En fait, on a deux cartes supports pour les PROMs bipôlaires : carte support pour la série 14/18; et carte support pour la série 24/28.

1.6.1 : Etude de la carte support des PROMs "14/18":

Elle comporte 4 paires de supports. Dans chaque paire, l'un des supports est prévu pour la PROM de référence, l'autre pour la PROM à programmer, et doivent être identiques. Ainsi, nous aurons :

— une paire de supports (2x8 broches), pour les PROMs 18 SA 030, S030 (organisées en 32 mots de 8 bits).

— une paire de supports (2x8 broches), pour les PROMs 14 SA 10, 14 S 10 (organisées en 256 mots de 4 bits).

— une paire de supports (2x10 broches), pour les PROMs 18 SA 22, 18 S 22 (256 mots de 8 bits) ;
18 SA 42, 18 S 42 (512 mots de 8 bits).
pour cette catégorie de support, on prévoit un nombre de switches à deux positions pour remédier aux différences de brochage des PROMs, liées à la capacité de chacune d'elles, (voir tableau de la page 18).

— une paire de supports (2x12 broches), pour les PROMs 18 SA 46, 18 S 46 (512 mots de 8 bits).

Cette carte comporte également un circuit de commutation constitué de 8 portes "AND" et 8 transistors.

Ce circuit permettra la programmation "bit par bit". En effet, les PROMs bipôlaires de la série 14/18 doivent être programmées bit par bit; les bits qui ne nécessitent pas une programmation doivent être connectés à +5V à travers une résistance de 3,9 KΩ. Ceci est réalisable grâce au circuit de commutation dont nous venons de parler. (pour des détails, voir schéma de la carte support "bipôlaires" en annexe).

1.6.2 : Etude de la carte support des PROMs "24/28":

De même, cette carte comportera un nombre de paires de supports, et éventuellement des switches à deux positions pour remédier aux différences de brochages. Dans chaque paire de supports, l'un sera prévu pour la PROM référence (en mode lecture), l'autre pour la PROM à programmer.
Vu que la série des PROMs 24/28 nécessite une programmation bit par bit, on prévoit ici aussi un circuit de commutation capable de nous offrir cette possibilité.

Notons enfin que pour ces deux cartes supports, on prévoit également un switch à 8 positions sur lequel on positionnera une combinaison binaire dont la lecture par le port B du PIA5 permettra d'identifier le type de PROM.

Pour l'identification des PROMs bipolaires, nous adopterons les conventions suivantes :

— bit 7 du switch = 1 ⟹ PROM bipolaire.
= 0 ⟹ REPROM (non bipolaire).

— bit 6 du switch = 1 ⟹ bipolaire de la série "14/18".
= 0 ⟹ bipolaire de la série "24/28".

Le tableau suivant donne les combinaisons retenues pour les PROMs bipolaires, série "14/18":

PROM :	combin. switch	PROM :	combin. switch
TBP 18SA030 et TBP 18S030	1 1 0 0 0 0 0 0 ($C0)	TBP 18SA42 et TBP 18S42	1 1 0 0 0 0 1 1 ($C3)
TBP 14SA10 et TBP 14S10	1 1 0 0 0 0 0 1 ($C1)	TBP 18SA46 et TBP 18S46	1 1 0 0 0 1 0 0 ($C4)
TBP 18SA22 et TBP 18S22	1 1 0 0 0 0 1 0 ($C2)	PROMs de la série: "24/28"	1 0 X X X X X X en général.

* Pour la carte support des PROMs bipolaires de la série "14/18", la paire de supports (2x 10 broches) est destinée pour des PROMs à 20 broches (l'une, Etalon, en mode lecture; l'autre, à programmer, en mode programmation).

Cependant, ces PROMs ne possèdent pas toutes le même brochage, et on prévoit par conséquent 4 switches à deux positions pour chaque boîtier.

Le tableau suivant donne les positions des switches qui conviennent au brochage de chaque PROM.

BOÎTIER (20 broches)	numéro de broche :	POSITION DES SWITCHES	
		PROM étalon :	PROM à program.
TBP 18 SA 22 et TBP 18 S 22 (256 x 8 bits)	16	PA1	PA2
	17	A5	A5
	18	A6	A6
	19	A7	A7
TBP 18 SA 42 et TBP 18 S 42 (512 x 8 bits)	16	A5	A5
	17	A6	A6
	18	A7	A7
	19	A8	A8

NOTES :

— PA1 et PA2 sont des lignes venant du port A du PIA5, et fournissant des signaux de contrôle aux broches de validation $\overline{G}$ de chaque PROM.

— A5, A6, A7, A8 : sont des lignes du bus d'adresses.

Chapitre 2 Etude des PROMs et REPROMs utilisées

2.1 : Introdution :

Les mémoires programmables que nous envisagerons sont de deux types :

— REPROMS : Intel 2716 ; 2732 ; 2732A ; 2758 ; 2764.
Tms 2516 ; 2532 ; 2564.

— PROMS : Texas : séries 14/18 ; et 24/28.

Les REPROMs Intel et Tms possèdent presque toutes les mêmes caractéristiques. Elles forment une marge d'application assez importante.

Les PROMs bipôlaires "14/18" et "24/28" constituent une large gamme (capacité très variée), et répondent à des applications très diverses. Celles des séries "14/18" ont une capacité allant de 256 à 4096 bits ; tandis que celles des séries "24/28" de 1024 à 16384 bits.

Les PROMs bipôlaires ont l'avantage d'être rapides mais l'inconvénient de consommer un grand courant.

2.2 : Les REPROMs Intel, Tms :

2.2.1 : Description :

Ce sont des mémoires éffaçables aux ultra-violets, programmables électriquement, de capacité variable. Ce sont des mémoires à lecture seulement, dont les caractéristiques globales sont les suivantes :

— organisation en mots de 8 bits.

— fonctionnement statique.

— tension d'alimentation unique : +5V.

— faible dissipation.

— compatibilité avec les conditions TTL.

2.2.2 : Fonctionnement en mode lecture :

Le bus d'adresse est décodé à l'intérieur du boîtier, pour sélectionner l'un des mots de 8 bits dans la mémoire. La lecture d'un mot se fait après la sélection du boîtier en mode lecture, et la donnée reste sur le bus de données tant que le boîtier reste sélectionné.

Un niveau haut sur l'entrée validation des sorties met le boîtier dans le mode sorties désactivées avec les sorties en état haute-impédance.

2.2.3 : Fonctionnement en mode programmation :

Initialement et après chaque opération d'effacement, tous les bits de la mémoire sont à "1" (état logique haut). Les données sont introduites en programmant sélectivement un "0" dans les positions désirées. Les mots sont adressés de la même façon que pour la lecture.
La programmation d'un "0" ne peut être changée en un "1" que par effacement aux ultra-violets.

Le circuit est en mode programmation en positionnant l'entrée Vpp à +25 volts. La tension Vcc est la même que pour une opération de lecture, (Vcc = +5v).

Les données à programmer sont introduites par mots de 8 bits à travers les broches de données / sorties.
Seuls les "0" seront programmés quand des "0" et des "1" sont présents dans le mot de donnée.

Après le préétablissement des adresses et des données, une impulsion programme TTL de durée 50 ms est appliquée à la broche $\overline{CE}$/prog. Certains boîtiers sont programmés sur le front montant de cette impulsion (ViL à ViH), d'autres sur son front descendant (ViH à ViL).

Toute position mémoire peut être programmée à n'importe quel moment, individuellement ou séquentiellement.

2.3 : Les PROMs bipôlaires de la série "14/18" :

2.3.1 : Description :

Ces PROMs sont destinées pour être programmées par une impulsion de 100 microsecondes.

Toutes les PROMs de cette série sont fournies avec un niveau logique bas "0" en sortie, sauf les PROMs TBP 14S10 et 14SA10 qui, elles, sont fournies avec un niveau logique haut "1" en sortie.
La programmation d'un bit consiste à inverser le niveau logique de celui-ci.

Un niveau logique bas "0" sur les entrées chip-select valide la PROM, tandis qu'un niveau logique haut

sur l'une de ces entrées met hors de fonctionnement les sorties du boîtier.

2.3.2 : Conditions de programmation recommandées :

		min.	nom.	max	unité
Tension d'alimentation, Vcc.	Régime stable	4.75	5	5.25	V
	impulsion progr.	10	10.5	11	V
Tension d'entrée des adresses.	niveau haut, V_{IH}	2.4		5	V
	niveau bas, V_{IL}	0		0.5	V
Tension appliquée à la sortie à programmer, $V_{O(pr)}$.		0	0.25	0.3	V
Durée X, de l'impulsion de programmation, Vcc.		98	100	1000	µs
Température.		0		55	°C

2.3.3 : Séquences de programmation pour les PROMs :

TBP 18SA030 ; TBP 18S030 ; TBP 14S10 ; TBP 14SA10 ; TBP 18SA22 ; TBP 18S22 ; TBP 18SA42, TBP 18S42 ; TBP 18S46 ; TBP 18SA46.

1_ Appliquer une tension Vcc = +5 volts, et adresser le mot à programmer.

2_ vérifier si le bit nécessite une programmation ; sinon passer au bit suivant.

3_ si le bit nécessite une programmation, inhiber les sorties en appliquant un niveau logique haut aux entrées "chip-select".

4_ un seul bit est programmé à la fois ;

les sorties ne devant pas être programmées doivent être connectées à +5 volts à travers une résistance de 3,9 kΩ (voir FIG.3); appliquer à la sortie à programmer la tension spécifiée dans la table; le circuit programmeur doit fournir un courant maximum de 150 mA.

5- augmenter le niveau de Vcc à 10.5 volts (front montant); le courant maximum durant la programmation est de 750 mA.

6- appliquer un niveau logique bas aux entrées "chip-select" ceci doit se passer 1 µs à 1 ms après que Vcc ait atteint la valeur de 10.5 volts.

7- après une durée X, un niveau logique haut est appliqué aux "chip-selects" pour inhiber les sorties.

8- 1 µs à 1 ms après avoir remis les "chip-selects" au niveau logique haut, remettre Vcc au niveau stable de 5 v. pour permettre la vérification du bit courant.

9- le (ou les) "chip-select(s)" peut être maintenu à un niveau logique bas 1 µs ou plus après que Vcc ait atteint la valeur de +5 volts.

10- répéter les étapes (1 à 8) pour chaque bit à programmer.

11- après avoir programmé toute la mémoire, vérifier la programmation de chaque mot en utilisant une valeur de Vcc comprise entre 4.5 et 5.5 volts.

N.B. une seule tentative de programmation est recommandée pour chaque bit.

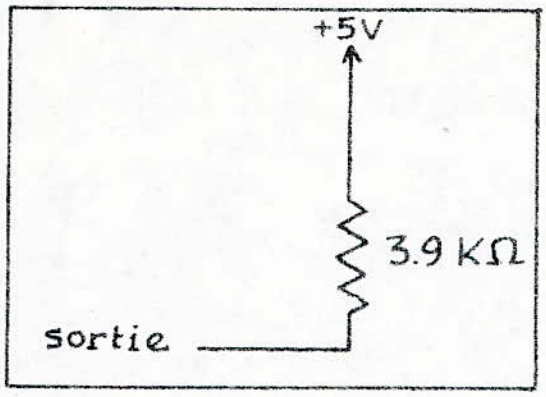


FIG.3 : Circuit de charge pour chaque sortie qui ne nécessite pas une programmation.

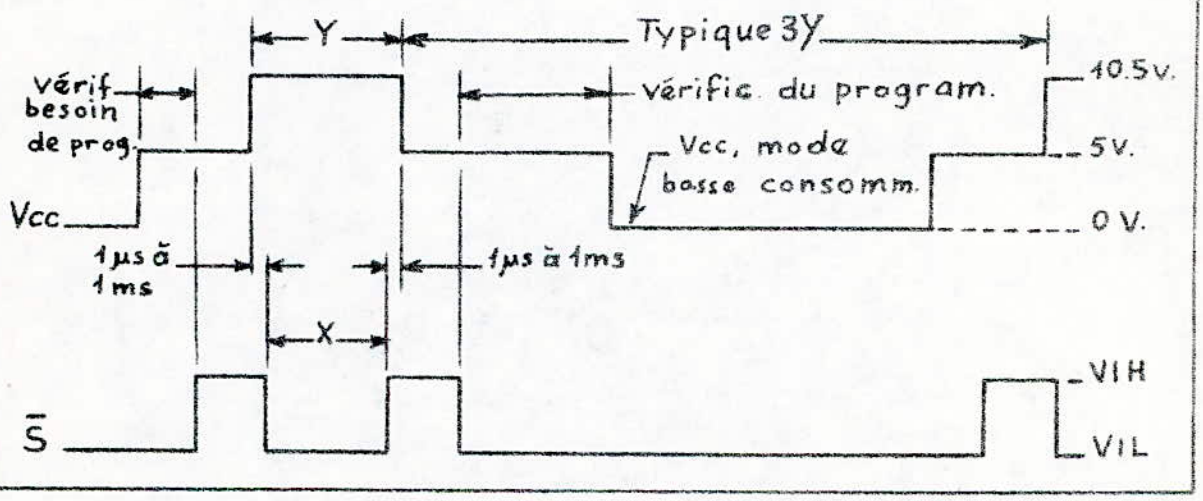


FIG.4 : Forme des tensions de programmation pour les PROMs bipôlaires, série "14/18".

2.4 : Les PROMs bipôlaires de la série "24/28" :

2.4.1 : Description :

Ce sont des PROMS récentes, et constituent 4 groupes :

- PROMs standards.
- PROMs à faible dissipation.

— PROMs à très faible dissipation.

— PROMs à enregistrement.

Toutes ces séries de PROMs possèdent la même technique de programmation.

Initialement, elles sont toutes à "1". La programmation consiste à loger un "0" dans les bits désirés.

L'impulsion programme est de l'ordre de 100 µs.

2.4.2 : Conditions de programmation recommandées :

PARAMETRES		min	typ	max	Unit
Tension Vcc en régime stable.		4.5	5	5.5	V
Tension d'entrée des adresses.	VIH	2.4		5	V
	VIL	0		0.5	
Tension à toutes les sorties, sauf celle à programmer.		0		0.5	V
Impulsion de programmation.	tension, Vcc(pr)	5.75	6	6.25	V
	largeur, tw	1000		2000	µs
	période		25	35	%
Impulsion de programmation à appliquer aux broches "select, Enable"	tension, Vs(pr)	9.75	10	11	V
	VIL	0		0.5	
Impulsion de programmation à appliquer à la sortie à programmer.	tension, Vo(pr)	16.75	17	17.25	V
	temps montée, tr	10		50	µs
	largeur, tw	98	100	1000	µs
	VIL	0		0.5	V
"clock" de vérific. pour PROM à registre.	tw (CH)		20		ns
Température.		0		55	°C

2.4.3 : Séquences de programmation pour les PROMs de la série "24/28" :

1_ Adresser le mot à programmer ; appliquer Vcc = +5V ± 10% et mettre les niveaux actifs à tous les "chip-selects" (S et $\overline{S}$) ou aux entrées de validation (E et $\overline{E}$).

2_ vérifier si le bit nécessite une programmation. Dans le cas des PROMs à enregistrement, une "horloge" doit être connectée à la broche "clock" pour cette vérification.

3_ appliquer un front montant (de Vcc à Vcc(pr)) avec un courant minimal de 200 mA.

4_ appliquer Vs(pr) à toutes les broches $\overline{S}$, $\overline{E}$ ou $\overline{G}$. $Ii \leq 15mA$

5_ connecter toutes les sorties, exceptée celle à programmer, à un niveau logique bas ($0 \leq VIL \leq 0.5V$). Un seul bit est programmé à la fois.

6_ appliquer à la sortie à programmer une impulsion de programmation d'une largeur de 98 microsecondes au moins. Le courant minimal du circuit programmeur doit être de 200 mA.

7_ à la fin de cette impulsion, déconnecter toutes les sorties des conditions VIL.

8_ réduire les tensions à $\overline{S}$, $\overline{E}$ et $\overline{G}$ à VIL.

9_ réduire Vcc au niveau stable et vérifier la programmation du bit. Pour les PROMs à enregistrement, une "horloge" est nécessaire pour cette vérification.

10- répéter les étapes de 3 à 9 pour chaque bit à programmer.

11- après avoir programmé toute la mémoire, vérifier que chaque mot a été correctement programmé, et ce en utilisant des valeurs de Vcc comprises entre 4,5 et 5,5 volts. Pour les PROMS à enregistrement, une "horloge" est nécessaire pour cette procédure.

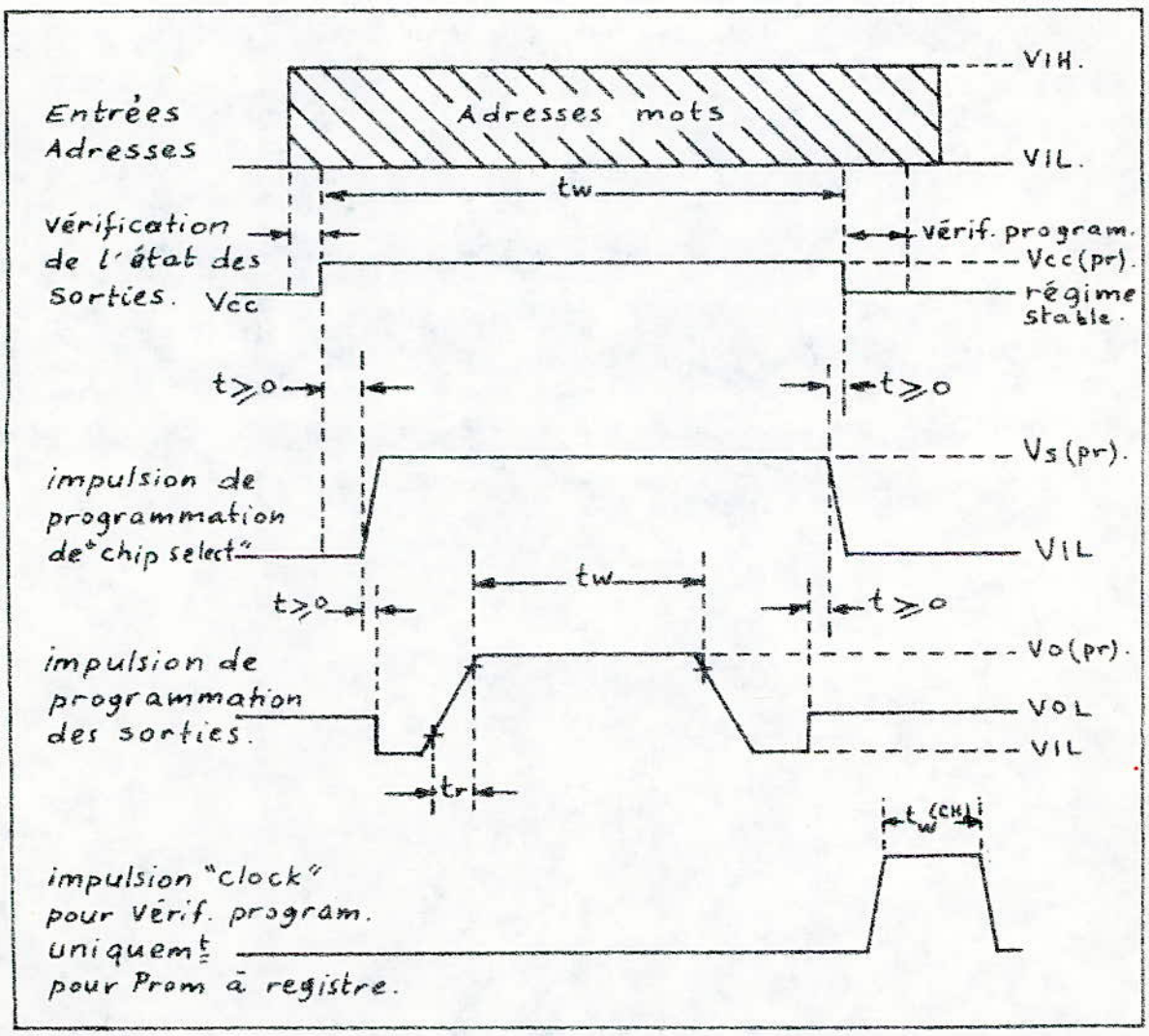


FIG.5 : Forme des tensions de programmation pour les PROMS bipôlaires, série "24/28"

Chapitre 3 : Organisation de la zone mémoire du système :

3.1 : Tableau d'adressage :

De l'étude précédente du "hardware" existant, nous retenons le tableau d'adressage suivant :

Adresses		A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
PROMs. II	FFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	E800	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
PROMs. I	E7FF	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1
	E000	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
INDETERMINÉ		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
ACIA.	DE09	1	1	0	1	1	1	1	0	0	0	0	0	1	0	0	1
	DE08	1	1	0	1	1	1	1	0	0	0	0	0	1	0	0	0
PIA I.	DE07	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1
	DE06	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	0
	DE05	1	1	0	1	1	1	1	0	0	0	0	0	0	1	0	1
	DE04	1	1	0	1	1	1	1	0	0	0	0	0	0	1	0	0
INDETERMINÉ		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
LATCHES	DD0E	1	1	0	1	1	1	0	1	0	0	0	0	1	1	1	0
	DD0D	1	1	0	1	1	1	0	1	0	0	0	0	1	1	0	1
	DD0C	1	1	0	1	1	1	0	1	0	0	0	0	1	1	0	0
PIA 3.	DD0B	1	1	0	1	1	1	0	1	0	0	0	0	1	0	1	1
	DD0A	1	1	0	1	1	1	0	1	0	0	0	0	1	0	1	0
	DD09	1	1	0	1	1	1	0	1	0	0	0	0	1	0	0	1
	DD08	1	1	0	1	1	1	0	1	0	0	0	0	1	0	0	0
PIA 4.	DD07	1	1	0	1	1	1	0	1	0	0	0	0	0	1	1	1
	DD06	1	1	0	1	1	1	0	1	0	0	0	0	0	1	1	0
	DD05	1	1	0	1	1	1	0	1	0	0	0	0	0	1	0	1
	DD04	1	1	0	1	1	1	0	1	0	0	0	0	0	1	0	0
PIA 5.	DD03	1	1	0	1	1	1	0	1	0	0	0	0	0	0	1	1
	DD02	1	1	0	1	1	1	0	1	0	0	0	0	0	0	1	0
	DD01	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	1
	DD00	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0
INDETERMINÉ		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RAM. 16K	47FF	0	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1
	0800	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
RAM. 2K	07FF	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
	0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3.2 : Organisation des différentes zones mémoires :

Le champ mémoire global du système peut être divisé en 4 grandes parties :

— Zone PROM (mémoire morte) : s'étendant de l'adresse hexadécimale E000 à l'adresse FFFF, soit une capacité de 8K-octets (2 PROMS de 4K-octets chacune).

— Zone périphériques : représentant tous les registres adressables des organes périphériques tels que : l'ACIA (2 registres adressables) ; le PIA1 (4 registres) ; le PIA3 (4 registres) ; le PIA4 (4 registres) ; le PIA5 (4 registres) ; les latches des alimentations programmables (3x1 registre). Cette zone s'étend de l'adresse DD00 à l'adresse DD0E, puis de DE04 à DE09.

— Zone RAM de 16K : elle s'étend de l'adresse 0800 à l'adresse 47FF ; son étude a été faite au chapitre 1 (carte mémoire).

— Zone RAM de 2K : s'étendant de l'adresse 0000 à 07FF, et constituant un champ de travail pour le MPU, et assurant la gestion de la pile de sauvegarde.

Un schéma récapitulatif de ces champs est donné ci-après :

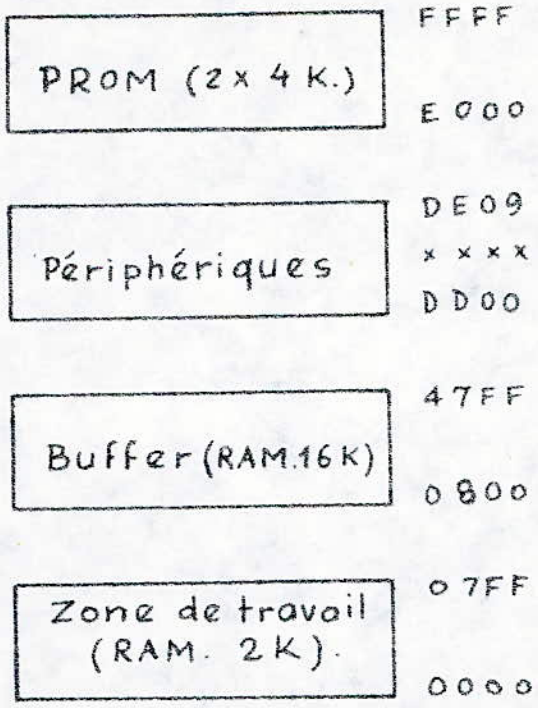
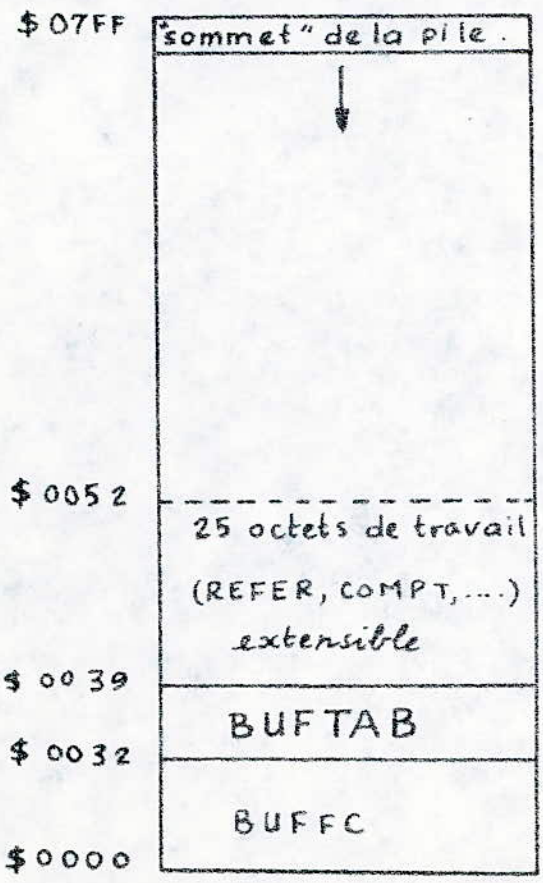


FIG.6 : Organisation des zones mémoires

3.2.1 : RAM de 2 Kilo-octets :



Comme nous l'avons cité précédemment, cette zone est dédiée exclusivement à la gestion de la pile, et aux opérations intermédiaires du CPU. Elle est organisée selon le schéma ci-contre.

N.B. Le symbole $ indique la notation hexadécimale du nombre qui le suit.

— BUFFC : Buffer de réception de la commande ou d'une ligne de donnée à traiter. La capacité de 50 octets a été prévue en conséquence.

— BUFTAB : Zone RAM de 8 octets dans laquelle seront transférées les informations contenues dans la table de la PROM ou de la REPROM à programmer ; cette table étant réservée en zone PROM.

* octets de travail :

— ERROR (1 octet) : contient un nombre indiquant un type précis d'erreur, ~~commise~~ détectée éventuellement pendant l'analyse, le traitement, ou l'éxécution d'une commande.

— REFER (1 octet) : contient un index de branchement à certaines tables.

— COMPT (1 octet) : c'est un octet compteur.

— NP1, NP2 (2 octets) : contiennent respectivement le nombre de caractères ASCII des paramètres P1 et P2 des commandes, ou le nombre de caractères de données ASCII par ligne de donnée (NP1 est utilisé).

— P1H, P1L (2 octets) : contiennent le paramètre P_1 (en héxa) de la commande concernée.
P_1 est une adresse virtuelle.

— P2H, P2L (2 octets) : contiennent le paramètre P2 (héxa) de la commande concernée. P2 peut être une adresse, ou un format (cas de la commande de chargement c).

— OCT (1 octet) : à 00, indique que le BUFFC est vide, à FF, indique que le BUFFC contient la commande à analyser ou la ligne de données à traiter.

— MAXH, MAXL (2 octets) : contiennent la capacité de la PROM à programmer (en nombre de mots).

— ETALON (1 octet) : à 00, si la référence est la RAM, à FF, si la référence est une PROM étalon.

— ZOTRA1, 2 (2 octets) : Zone de travail pour les sous-programmes de conversion et comptage des caractères ASCII des adresses de stockage, et du nombre de données par ligne.

— ZOTRA 3 (1 octet) : Zone de travail du sous-programme de conversion et stockage des données.

— ADRH, ADRL (2 octets) : contiennent, après conversion, l'adresse de stockage du premier caractère de la ligne de données en cours.

— SAUV 1,2 (2 octets) : sauvegardent l'indext du buffer lorsqu'on l'utilise à deux niveaux.

— SAUV 3,4 (2 octets) : sauvegardent l'indext de la PROM lorsqu'on l'utilise à deux niveaux.

— ZONATL (1 octet) : contient, après conversion en hexa, un nombre indiquant la nature de la ligne (format MOTOROLA).

* Pile de sauvegarde :

Le pointeur de pile sera initialisé par l'adresse du début (sommet) de la pile, soit l'adresse $07FF.
Il se décrémentera à chaque stockage dans la pile (opération PUSH), et s'incrémentera à chaque tirage d'une donnée de la pile (opération PULL).

3.2.2 : RAM de 16 kilo-octets :

Cette zone s'étend de l'adresse $0800 à $47FF.
Elle est utilisée pour contenir les données transférées :

- soit de l'extérieur : opération de chargement en vue d'une programmation.
- soit de la PROM : opération de lecture.

On conviendra d'appeler cette zone : Buffer, et sera en tous les cas une "image" de la PROM à programmer.

3.2.3 : Zone "périphériques" :

Grâce à leurs registres internes adressables.

les organes périphériques tels que l'ACIA, le PIA, ... etc, peuvent être considérés par le microprocesseur comme de simples positions mémoires où l'on pourra lire et écrire. Dans notre système, nous avons la structure suivante : (périphériques avec adresses d'implantation de leurs registres internes).

* Remarque : Dans le cas des PIAs, on remarquera que les registres "sens de transfert des données" DDRA (ou DDRB), et les registres de sortie ORA (ou ORB) auront respectivement la même adresse d'implantation ; pour les différencier, on positionnera le bit 2 du registre de commande concerné CRA (ou CRB) : à "0" ⇒ accès à DDRA (ou DDRB).

à "1" ⇒ accès à ORA (ou ORB).

organe périphérique	Adresse du registre	bit 2 du registre CRA(B)	registre adressé
PIA 5	DD00	0	DDRA5
	DD00	1	ORA5
	DD01	X	CRA5
	DD02	0	DDRB5
	DD02	1	ORB5
	DD03	X	CRB5

Organe périphérique	Adresse du registre	bit 2 du registre CRA(B)	registre adressé
PIA4	DD04	0	DDRA4
	DD04	1	ORA4
	DD05	X	CRA4
	DD06	0	DDRB4
	DD06	1	ORB4
	DD07	X	CRB4
PIA3	DD08	0	DDRA3
	DD08	1	ORA3
	DD09	X	CRA3
	DD0A	0	DDRB3
	DD0A	1	ORB3
	DD0B	X	CRB3

LATCH1	DD0C
LATCH2	DD0D
LATCH3	DD0E

PIA1	DE04	0	DDRA1
	DE04	1	ORA1
	DE05	X	CRA1
	DE06	0	DDRB1
	DE06	1	ORB1
	DE07	X	CRB1

Organe périphérique	Adresse du registre	signal R/$\overline{W}$	registre adressé
ACIA	DE08	0	CR
	DE08	1	SR
	DE09	0	TDR
	DE09	1	RDR

CR : Control Register (registre de contrôle).

SR : Status Register (registre d'état).

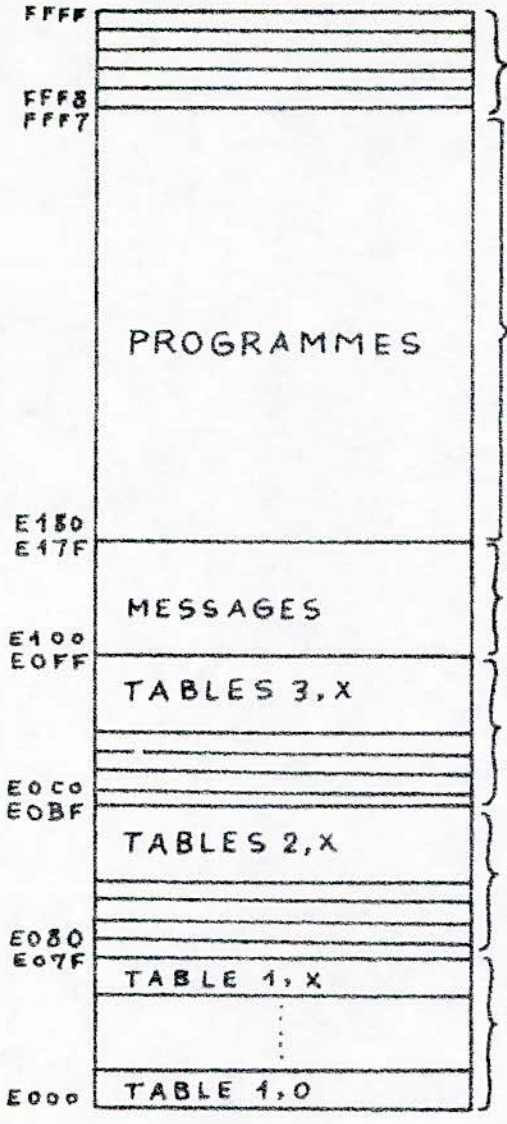
TDR: Transmit Data Register (registre de transmission des données).

RDR: Receive Data Register (registre de réception des données).

3.2.4 : Zone PROM (mémoire morte) :

Elle est destinée à contenir les programmes de gestion du système, les tables des REPROMs et des PROMs bipôlaires à programmer, et des messages éventuels.

Sa capacité maximale est de 8 kilo-octets, organisée selon le schéma suivant :



vecteurs d'initialisation (8 octets)

zone disponible = 7800 octets.

zone disponible = 128 octets.

TABLES 3,X des PROMs bipôlaires de la série "24/28" (64 octets), (note 3).

TABLES 2,X des PROMs bipôlaires de la série "14/18" (64 octets), (note 2).

TABLES 1,X des REPROMS (128 octets) (voir note 1).

NOTES:

—1— Chacune des tables 1,X comporte 5 octets contenant respectivement (ordre croissant) :
- combinaison pour avoir 25 volts.
- temporisation de 50 ms (2 octets sans le bit 7 du 1er octet).
- niveau de l'impulsion program. (bit 7 du 1er octet tempor.)
- capacité en mots de la REPROM (2 octets).

—2— Les 4 octets du "bas" de la zone des tables 2,X contiennent les paramètres communs des PROMS de la série "14/18" : respectivement : combin. 5 V ; combin. 10,5 V ; tempo. 10 µs ; 100 µs
Les tables 2,X comporteront chacune 3 octets, respectivement : état vierge (1 octet) ; capacité (2 octets).

—3— Identique à 2, sauf qu'on réserve 5 octets pour les paramètres communs de la série "24/28", respectivement : 3 octets pour les tensions ; 2 octets pour la temporisation.

Chapitre 4 : Programmation

4.1 : Introduction :

Le programmateur étudié doit être conversationnel, c'est-à-dire qu'il doit permettre un « dialogue » entre le microprocesseur et un ordinateur (configuration de notre système en mode périphérique) ; ou entre le microprocesseur et un opérateur par l'intermédiaire d'un ensemble "clavier - console de visualisation" (configuration de notre système en mode U.C.).

Dans cette optique, nous nous proposons de formuler un certain nombre de commandes que le système pourra comprendre, analyser, puis éxécuter. Ces commandes sont constituées de séquences ESCAPE ; c'est-à-dire du caractère ASCII "ESC" représenté en hexadécimal par 1B, suivi d'autres caractères ASCII représentant la commande concernée et, le cas échéant, les paramètres séparés par des virgules.

La structure la plus générale d'une de ces commandes est la suivante :

ESCA P1, P2 (RC).

ESC ⟶ $1B : entête de la commande.

A : caractère alphabétique déterminant la commande.

P1, P2 : paramètres dépendant de la commande, séparés entre eux par une virgule.

RC : "retour chariot", termine la séquence de commande.

nous formulons ci-dessous les commandes suivantes :

1 — ESCZ (RC) : mise à zéro du buffer si la PROM vierge contient des 00, à FF si la PROM vierge contient FF.

2 — ESCE (RC) : indique que le buffer est remplacé par une PROM étalon installée sur la carte support. Toutes les opérations de programmation et de comparaison se feront en prenant cette PROM comme référence.
Par défaut, c'est le buffer qui sera pris pour référence.

3 — ESCR (RC) : remise à l'état initial de tous les paramètres de travail.

4 — ESCP P1, P2 (RC) : cette commande permet de lancer la programmation de la PROM.
P1, P2 : des adresses allant de 0000 à FFFF.
P1 : adresse de début de la zone à programmer dans la PROM.
P2 : adresse de fin de la zone à programmer dans la PROM.

— Par défaut de P1, on le prendra = 0000. et l'on écrira : ESCP , P2 (RC).

— Par défaut de P2, on prendra le maximum et l'on écrira : ESCP P1 (RC).

— Par défaut de P1 et P2, on programmera toute la PROM (du 0000 à son maximum), et la commande s'écrira : ESCP (RC).

5 — ESCL P1, P2 (RC) : cette commande permet la lecture de la PROM dans le buffer, entre les adresses P1 et P2.
L'absence des paramètres P1 ou/et P2 aura les mêmes conséquences que pour la commande P.

6 — ESCV P1, P2 (RC) : cette commande permet de comparer le contenu de la PROM (de l'adresse P1 à l'adresse P2) avec le contenu du buffer ; ce dernier étant préalablement une image de cette PROM.
L'absence de P1 ou/et P2 aura les mêmes conséquences que pour la commande P.

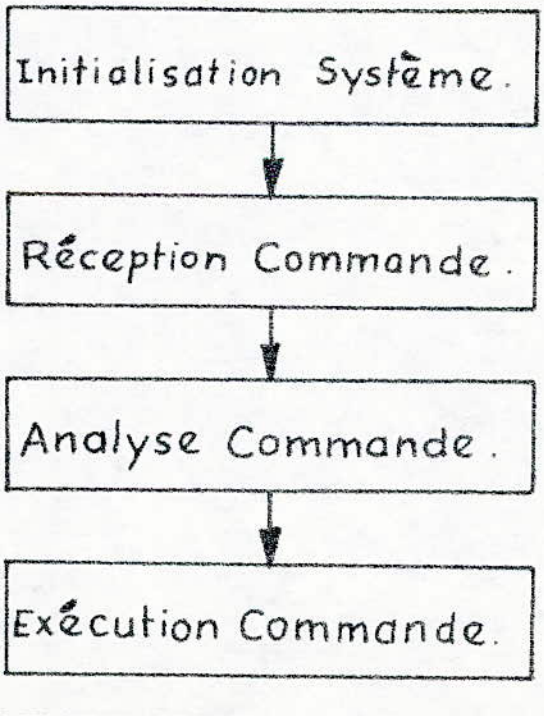
7 — ESCC P1, P2 (RC) : cette commande permet de charger le buffer par des données reçues de l'extérieur.
P1, étant l'adresse virtuelle de la PROM à programmer, c'est-à-dire son adresse de début dans le système auquel elle est destinée.

L'adresse du début de chargement dans le buffer doit être calculée en conséquence pour que ce dernier soit une image de la PROM à programmer.
P2, un nombre binaire de 8 bits indiquant le format de la ligne de données : $00 ⇒ format INTEL.
$01 ⇒ format MOTOROLA.

8 — ESCI P1, P2 (RC) : commande permettant de sortir les données sur une imprimante.
P1 et P2, ayant exactement les mêmes définitions que pour la commande C.

4.2 : Méthode de programmation :

Le programme principal comprendra les étapes suivantes:



4.2.1: INITIALISATION DU SYSTEME

L'initialisation du système comportera les étapes suivantes:

- Initialisation du microprocesseur.
- Initialisation des interfaces : ACIA, PIA1, PIA3, PIA4, PIA5.
- Mise à zéro de toutes les alimentations programmables.
- Attribution des valeurs aux symboles des registres programmables des PIAs, de l'ACIA. et des LATCHES pour les alimentations programmables, ainsi qu'à certains symboles ASCII utilisés dans la suite du programme.
- Réservation d'octets dans la zone RAM de 2 Kilo-octets : (voir chapitre 3, paragraphe 3.2.1).
- Réservations des tables pour les PROMs à programmer ; 1,X pour les PROMs non bipolaires; 2,X pour les PROMs bipolaires de la série "24/18"; 3,X pour celles de la série "24/28" (pour les détails, voir chapitre 3, paragraphe 3.2.4).

1 : Initialisation du microprocesseur :

L'entrée $\overline{\text{RESET}}$ du microprocesseur MC 6800 est utilisée pour mettre à l'état initial et démarrer le MPU après mise sous tension ou après une panne d'alimentation. Le passage à l'état haut de cette entrée conduit le MPU à exécuter la séquence de démarrage, pendant laquelle le contenu des deux derniers octets de la mémoire, (octets d'adresse FFFE et FFFF) est chargé dans le compteur programme pour adresser le début du programme.

2 : Initialisation des interfaces :

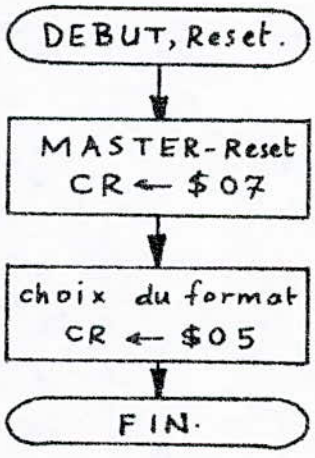
a- L'ACIA : l'ACIA (EF6850) est un circuit d'interface entre le µp EF6800 et un périphérique travaillant en mode série asynchrone. Il possède 4 registres internes. L'initialisation de l'ACIA revient à initialiser son registre de contrôle (CR) par une valeur déterminant un certain nombre de choix :

* (CR0 = 1, CR1 = 0) ⇒ division d'horloge par 16.
* (CR2 = 1, CR3 = 0, CR4 = 0) ⇒ format des mots comme suit : 7 bits + parité impaire + 2 bits d'arrêt (choix arbitraire).
* (CR5 = 0, CR6 = 0) ⇒ interruptions de transmission inhibées.
* (CR7 = 0) ⇒ interruptions du recepteur inhibées.

N.B. Après une mise sous tension, les bits CR0 et CR1 du registre de contrôle (CR) de l'ACIA doivent être mis à 1 ; ceci s'appelle : MASTER-RESET.

Après cette opération, on pourra programmer le registre de contrôle selon la configuration ci-dessus.

on aura donc, pour l'ACIA, l'organigramme suivant :



b_Les PIAS: Le PIA (EF6821) est un moyen universel d'interface des appareils périphériques avec un microprocesseur EF6800. Il possède 6 registres internes (3 pour le port A; 3 pour le port B), vus par le MPU comme 4 simples positions mémoires adressables.

Les PIAs seront tous initialisés en entrée pour éviter, à la mise sous tension du système, qu'une tension intempestive trop forte détruise les PROMs de la carte suppot.

_PIA3 : mettre $00 dans CRA3 (⇒ accès à DDRA3).
mettre $00 dans DDRA3 (⇒ port A en entrée).
mettre $04 dans CRA3 (⇒ accès à ORA3).
le port B du PIA3 n'est pas utilisé ici.

_PIA4
_PIA5
_PIA1
} procéder de la même façon pour les ports A et B.

3 : Mise à zéro des alimentations programmables:

Ceci se résume à mettre la combinaison $00 dans tous les latches de la carte alimentation programmable.

_ mettre $00 dans LATCH1, LATCH2, LATCH3.

4 : Attribution des valeurs aux registres programmables :

Cette opération consiste à donner l' « équivalence » d'un symbole (ASCII, ou autre) à une valeur que l'on donne. Ceci est fait par la directive : EQU (voir listing).

Exemple: "RC" EQU $0D (RC = retour chariot).

5 : Réservation d'octets :

(voir organisation de la RAM de 2K, paragraphe 3.2.1, chap.3).
Notons seulement que cette réservation se fait par la directive RMB. ("Reserve Memory Byte").

Exemple : ERROR RMB 1 veut dire :
réserver un octet pour "l'ERROR".

6 : Réservation des tables :

(voir chapitre 3, paragraphe 3.2.4).
On note ici également que la réservation de ces tables se fait par la directive "RMB", et que les tables seront chargées par les informations nécessaires à la programmation des PROMs auxquelles elles correspondent.

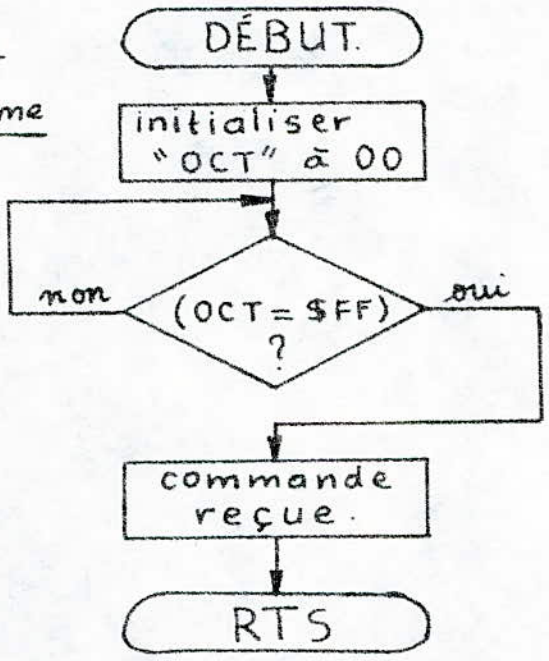
N.B.

Avant de terminer ce paragraphe, il est à noter qu'avant la mise sous tension du système, on recommande d'enlever toutes les PROMs de la carte support afin d'éviter leur destruction imprévisible : en effet à l'état initial, les alimentations programmables pourraient fournir des tensions trop élevées vers la carte support, d'où le risque de destruction des PROMs qu'elle porte.
Après cette initialisation, les PROMs peuvent être alors remises en place.

4.2.2 : RECEPTION DE LA COMMANDE

La réception de la commande est faite sous interruption ($\overline{IRQ}$). Dans le programme principal, pour savoir qu'une commande est entièrement reçue dans le buffer de réception BUFFC, on réalise une « boucle d'attente » qui consiste à tester l'octet « OCT » initialisé à zéro, et attendre son passage à la valeur $FF. (voir organigram.)

Boucle d'attente dans le programme principal.

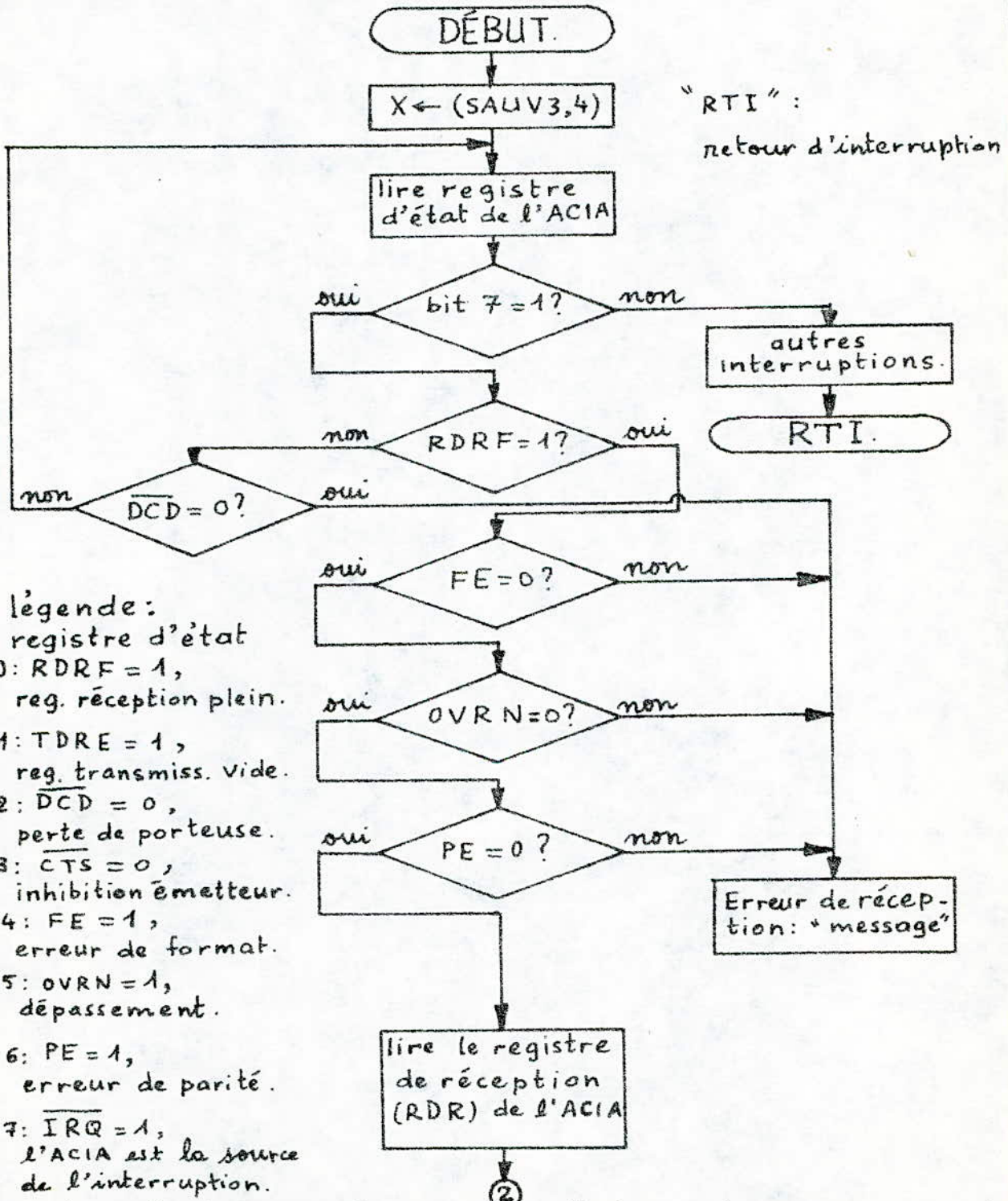


Quant à la routine de réception (RECEPT) proprement dite, elle consiste à : (à travers l'ACIA),

- Initialiser l'index avec l'adresse de stockage dans BUFFC.
- Tester le registre d'état et attendre que le registre de réception (RDR) contienne un caractère correct (sans erreur).
- stocker chaque caractère reçue, dans BUFFC à l'adresse indexée, et faire un écho du caractère si notre système est en liaison avec un écran.

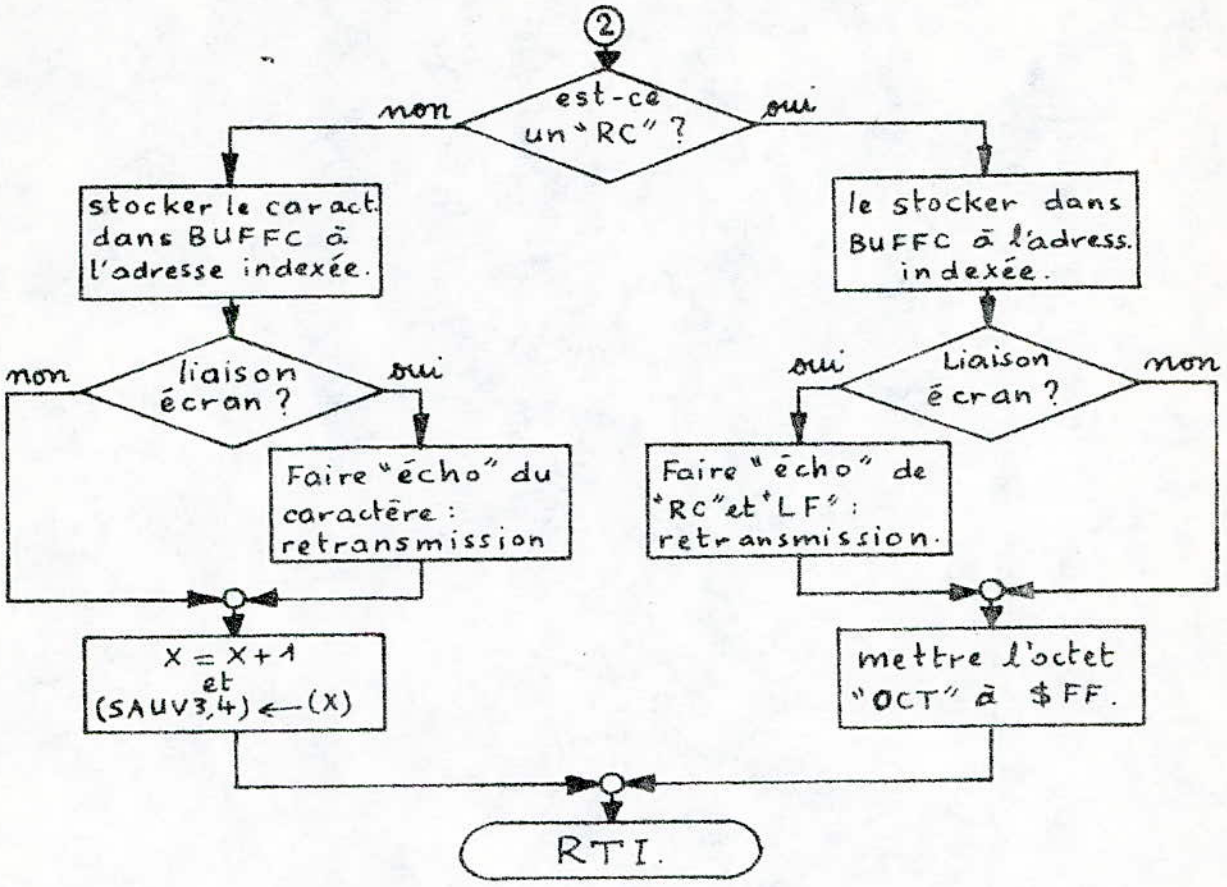
— Quand le caractère ("RC": retour-chariot) est reçu, on met l'octet "OCT" à $FF pour dire que BUFFC contient la commande complète, et que l'analyse peut commencer.

* Organigramme de la routine de réception (RECEPT):



légende:
registre d'état
0: RDRF = 1, reg. réception plein.
1: TDRE = 1, reg. transmiss. vide.
2: $\overline{DCD}$ = 0, perte de porteuse.
3: $\overline{CTS}$ = 0, inhibition émetteur.
4: FE = 1, erreur de format.
5: OVRN = 1, dépassement.
6: PE = 1, erreur de parité.
7: $\overline{IRQ}$ = 1, l'ACIA est la source de l'interruption.

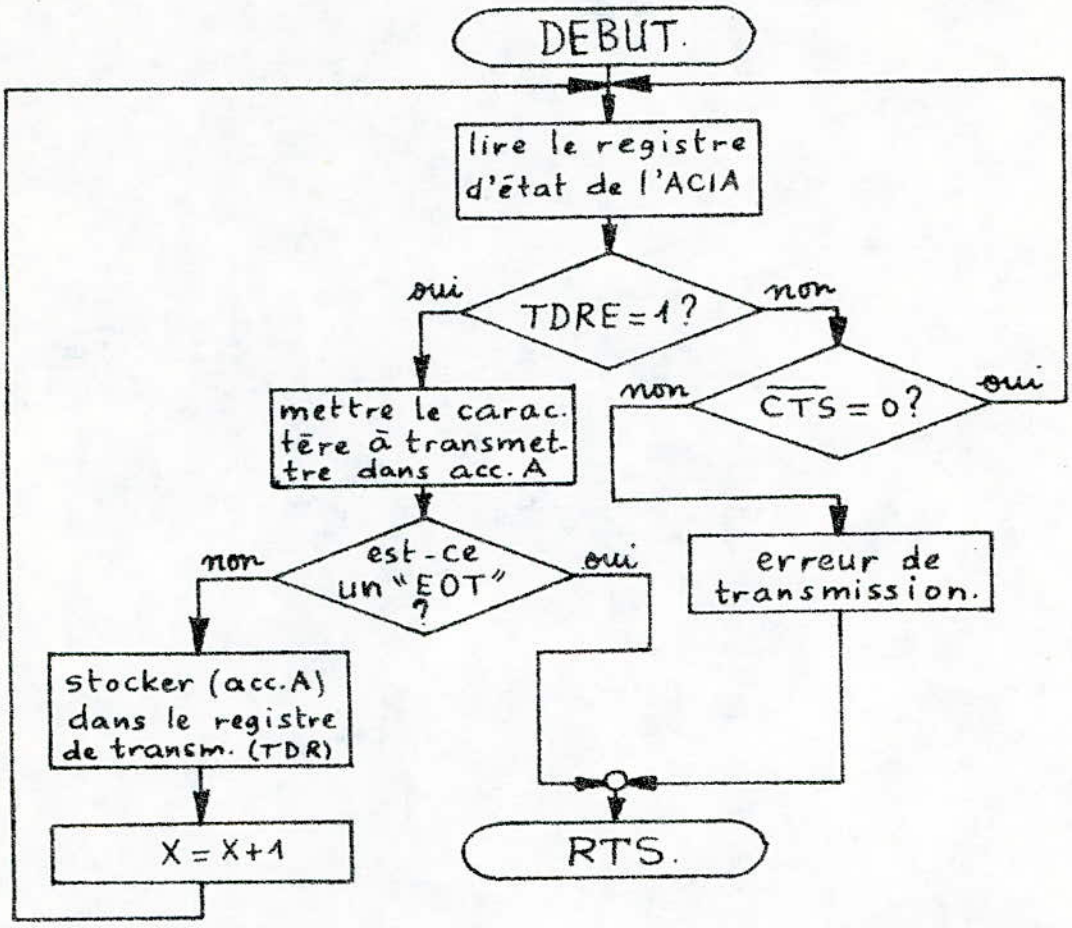
en résumé: $83 dans le registre d'état ⇒ interrup. prise en compte.



NOTES:

- lorsqu'on reçoit un caractère, et si le système travaille en liaison avec un écran, on doit faire un "écho" de ce caractère, c'est-à dire le retransmettre à travers l'ACIA pour le visualiser sur écran.
- Quand on reçoit "Retour-chariot", il faut renvoyer "RC" et "Line-Feed" pour que le curseur revienne au début de la ligne suivante.
- l'"ECHO" doit faire appel à une routine de transmission notée "TRANSM", dont nous donnons l'organigramme à la page suivante :

Organigramme de la routine de Transmission noté: TRANSM.



NOTES:

- Avant d'appeler ce sous-programme, il faut programmer l'ACIA en sorte que les interruptions du transmetteur soient autorisées (bits CR5 = 1, CR6 = 0; du registre de contrôle).
- Si ce sous-programme est appelé par le programme de traitement d'erreur, l'index doit être initialisé à l'adresse du début du buffer de transmission "BUFTRA" qui contient le message à transmettre.
- Pour transmettre un caractère, on teste si le registre de transmission est vide (TDRE = 1?); sinon il faut vérifier, avant de tester de nouveau TDRE, que $\overline{CTS}$ n'est pas à 1: ce qui inhiberait TDRE.

4.2.3: ANALYSE DE LA COMMANDE

Avant d'entamer cette partie, il est indispensable de connaître la structure du programme principal dont l'organigramme est donné aux pages suivantes:

* Commentaires sur le programme principal:

A. Fonction:

Il initialise le système (matériel: ACIA, PIAs, ... etc); initialise les paramètres de travail des autres sous-programmes; teste l'octet "OCT" pour savoir si la commande est entièrement reçue. Si oui, il envoie successivement à des sous-programmes de détermination, d'analyse, et d'exécution de la commande. Au retour de chaque sous-programme, il teste s'il y a eu ou pas d'erreur; si oui, il envoie un message spécifiant l'erreur sinon, il émet un message indiquant que la commande a été correctement exécutée. Après chaque étape, il revient en état d'Attente d'une nouvelle commande.

B. Paramètres d'entrée:

— BUFFC: Zone RAM de 50 octets, contenant la commande à traîter.

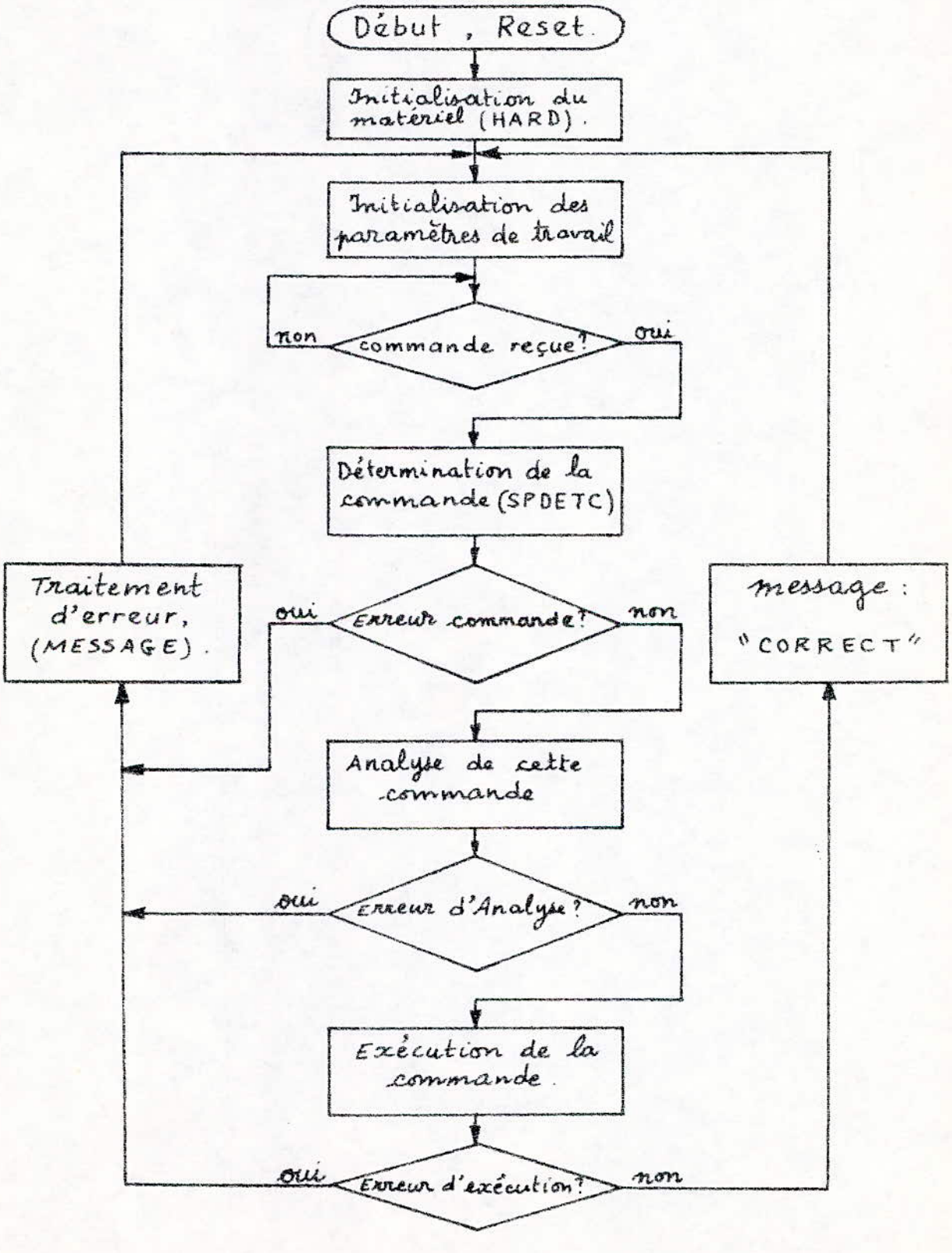
C. Zone de travail:

— Registres adressables des périphériques (ACIA, PIAs, ... etc.)
— Paramètres préalablement réservés dans la RAM de 2 kilo-octets.

D. Paramètres de sortie :

— ERROR = 00 ; pas d'erreur ; message " CORRECT".

— ERROR ≠ 00 ; erreur commise ; message d'erreur.



4.2.3.1 : Sous-Programme de DETermination de la Commande : SPDETC.

A. Fonction :

Il vérifie la présence du caractère ESCAPE "ESC" au début de la séquence de commande stockée dans BUFFC ; sinon, il s'agit d'une erreur n°01 ; si oui, il détermine le type exact de commande, s'il la trouve correcte, il envoie au sous-programme d'analyse de celle-ci ; sinon, il s'agira d'une erreur n°02 ; puis retourne au point d'appel.

B. Paramètres d'entrée :

— OCT = FF, indiquant que la commande est reçue.

— ERROR = 00 initialement.

— BUFFC, contenant la séquence de commande.

C. Zone de travail :

— COMPT, initialement à zéro.

— TABCOM, table contenant les commandes acceptées.

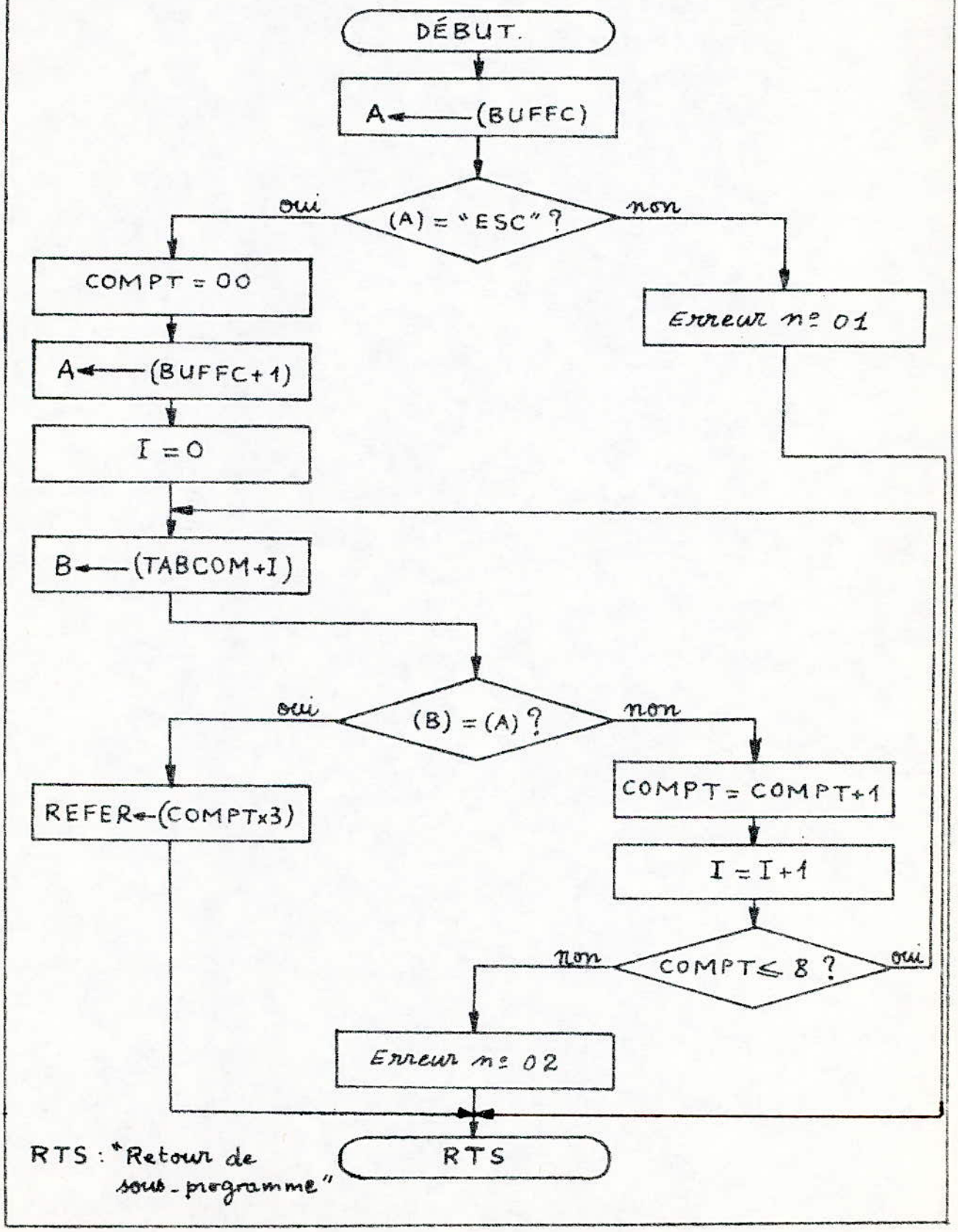
— REFER, contiendra l'indexe de la table des sauts aux sous-programmes d'analyse.

D. Paramètres de sortie :

— ERROR = 00, si pas d'erreur d'analyse.
= 01, si "ESC" n'existe pas en tête de commande
= 02, si la commande est autre que celles qui sont acceptées par le système.

— REFER, contenant l'indexe de la table des sauts vers les sous-programmes d'analyse.

Organigramme du Sous-Programme de DETermination de la Commande : SPDETC.



4.2.3.2 : Sous-Programme 1, relatif aux commandes Z, E, ou R; noté : SP1.

Au retour de SPDETC, si pas d'erreur, le nombre contenu dans l'octet REFER permet de se brancher au sous-programme d'analyse de la commande concernée. Cette procédure est faite au niveau du programme principal par un ensemble d'instructions qui n'apparaîtront pas dans l'organigramme mais qui figureront dans le programme imprimé sur "listing".

Commentaires sur SP1 :

A. Fonction :

Il vérifie la présence du "Retour chariot" (RC) immédiatement après la commande ; si oui : retour de sous-programme vers le programme principal ; sinon, il s'agira d'erreur n° 03, et retour au programme principal pour "traiter" cette erreur.

B. Paramètres d'entrée :

— ERROR = 00, initialement.

— BUFFC, contenant la séquence de commande à analyser.

C. Zone de travail :

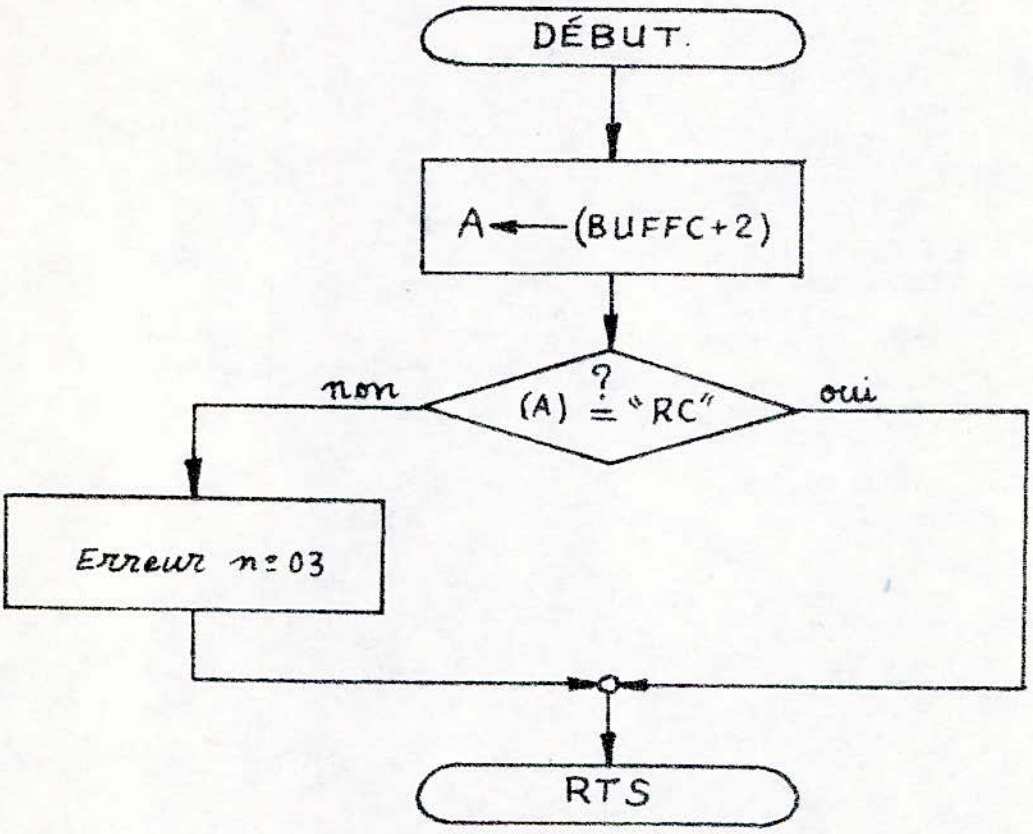
— Accumulateur A.

D. Paramètres de sortie :

— ERROR = 00, si pas d'erreur (RC existe).

= 03, si RC n'existe pas, c'est-à-dire que la commande est mal formulée.

Organigramme du Sous-Programme 1, relatif aux commandes Z, E, ou R ; il est noté : SP1.



RTS : "Retour de sous-programme"

4.2.3.3 : Sous-Programmes SP21, relatif aux commandes P,L,V ; SP22 relatif aux commandes C,I ; et SP2, commun à P,L,V, C,I.

A. Fonction :

— SP21, initialise les paramètres NP1 et NP2 à 04, nombre maximal de caractères ASCII que peut comporter chacun des paramètres P1 et P2 (pour les commandes : P,L,V).

— SP22, initialise les paramètres NP1 à 04, NP2 à 02, nombre maximal de caractères ASCII que peuvent comporter respectivement les paramètres P1 et P2 (commandes C,I).

Les sous-programmes SP21 et SP22 ne font que préparer les paramètres NP1 et NP2 pour renvoyer ensuite à SP2.

— SP2, vérifie la formulation correcte de la commande concernée (présence ou absence du "blanc" entre la commande et ses paramètres), et envoie à des sous-programmes pour comptage et conversion des caractères ASCII des paramètres P1 et P2. À chaque retour, il teste l'erreur et revient dans tous les cas au programme principal.

B. Paramètres d'entrée :

— ERROR, initialement à zéro.

— NP1, NP2, initialisés à leur maximum.

— P1H, P1L ; P2H, P2L, initialement à zéro.

— MAXH, MAXL, contenant la capacité de la PROM à programmer.

— BUFFC, contenant la séquence de commande à analyser.

— Index, initialisé à l'adresse (BUFFC+2).

C. Zone de travail :

— Accumulateurs A et B.

— P1H, P1L ; P2H, P2L, contiennent les paramètres convertis.

— NP1, NP2, modifiés selon le nombre exact de caractères ASCII des paramètres P1 et P2.

D. Paramètres de sortie :

— ERROR = 00, si pas d'erreur.

≠ 00, si des erreurs sont commises pendant l'analyse.

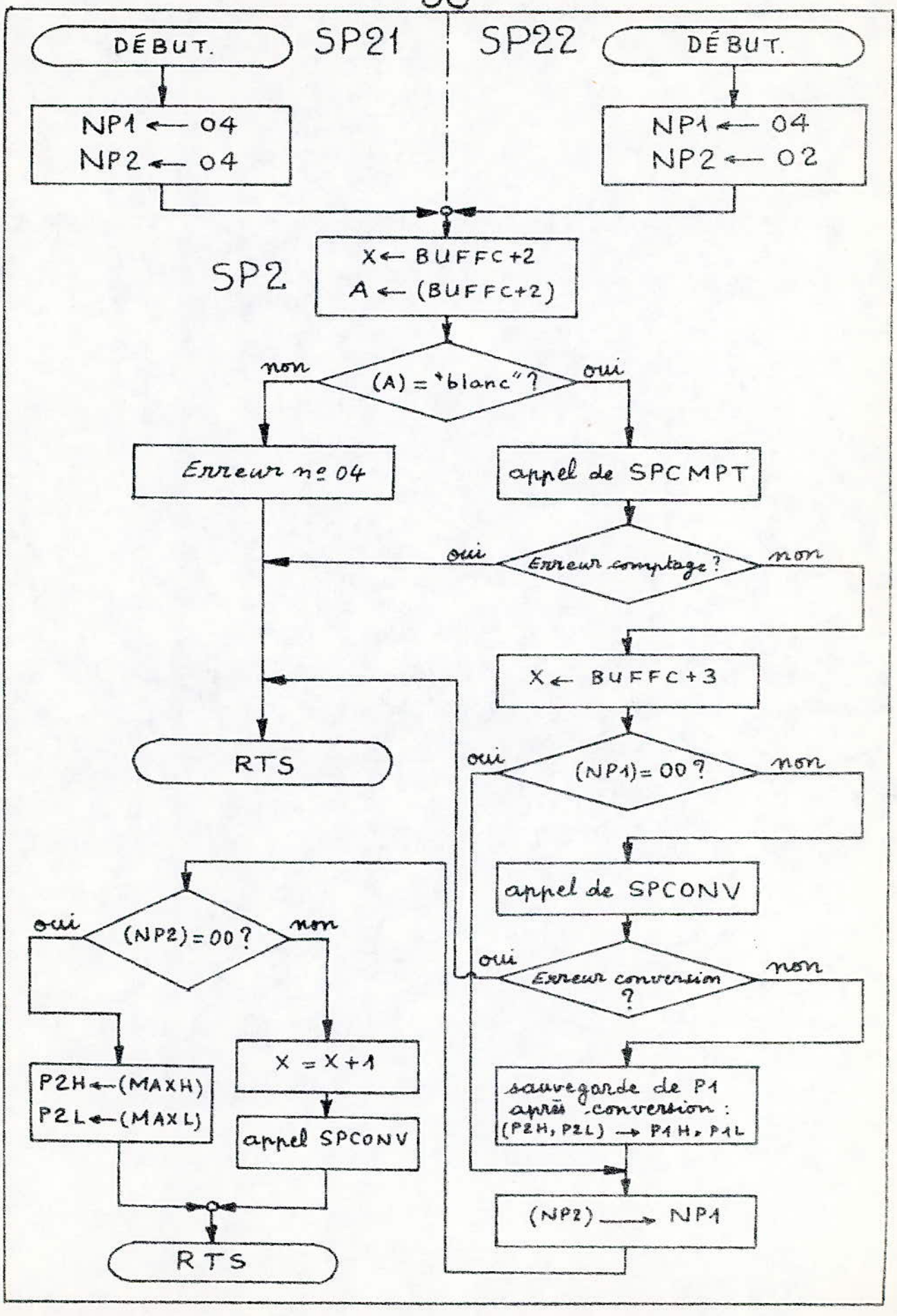
— P1H, P1L ; P2H, P2L, contiennent les paramètres P1 et P2 après les avoir convertis.

— NP1, NP2, contiennent le nombre exact de caractères ASCII qui composent les paramètres P1 et P2 respectivement. Tout ceci, s'il n'y a pas eu d'erreur pendant l'analyse ; sinon le contenu des octets ci-dessus seront sans importance.

E. Sous-programmes appelés par SP2 :

— SPCMPT.

— SPCONV (appelé à deux reprises).


DÉBUT.
SP21
NP1 ← 04
NP2 ← 04
SP22
DÉBUT.
NP1 ← 04
NP2 ← 02
SP2
X ← BUFFC+2
A ← (BUFFC+2)
non
(A) = "blanc" ?
oui
Erreur n° 04
appel de SPCMPT
oui
Erreur comptage ?
non
X ← BUFFC+3
RTS
oui
(NP1) = 00 ?
non
appel de SPCONV
oui
Erreur conversion ?
non
oui
(NP2) = 00 ?
non
P2H ← (MAXH)
P2L ← (MAXL)
X = X+1
appel SPCONV
sauvegarde de P1 après conversion : (P2H, P2L) → P1H, P1L
(NP2) → NP1
RTS

4.2.3.4: Sous-Programme de COMPTage des caractères ASCII composant les paramètres P1 et P2;

Il est noté : SPCMPT :

A. Fonction :

Il compte le nombre exact de caractères ASCII qui composent les paramètres P1 et P2 (pour les commandes : P, L, V, I, C); Il est appelé par le sous-programme SP2.

B. Paramètres d'entrée :

— ERROR, initialement à zéro.

— COMPT, initialement à zéro.

— NP1, NP2, initialisés à leur maximum (04;04) ou (04;02) selon la commande à analyser.

— Indexe, pointant l'adresse où se trouve le premier le caractère qui vient immédiatement après le "blanc".

C. Zone de travail :

— COMPT, servant à compter les caractères ASCII qui composent chaque paramètre.

— Accumulateurs A et B.

D. Paramètres de sortie :

— ERROR = 00, si pas d'erreur.

≠ 00, en cas d'erreur de comptage des caractères.

— NP1, NP2, contenant respectivement le nombre exact de caractères ASCII composant les paramètres P1 et P2.

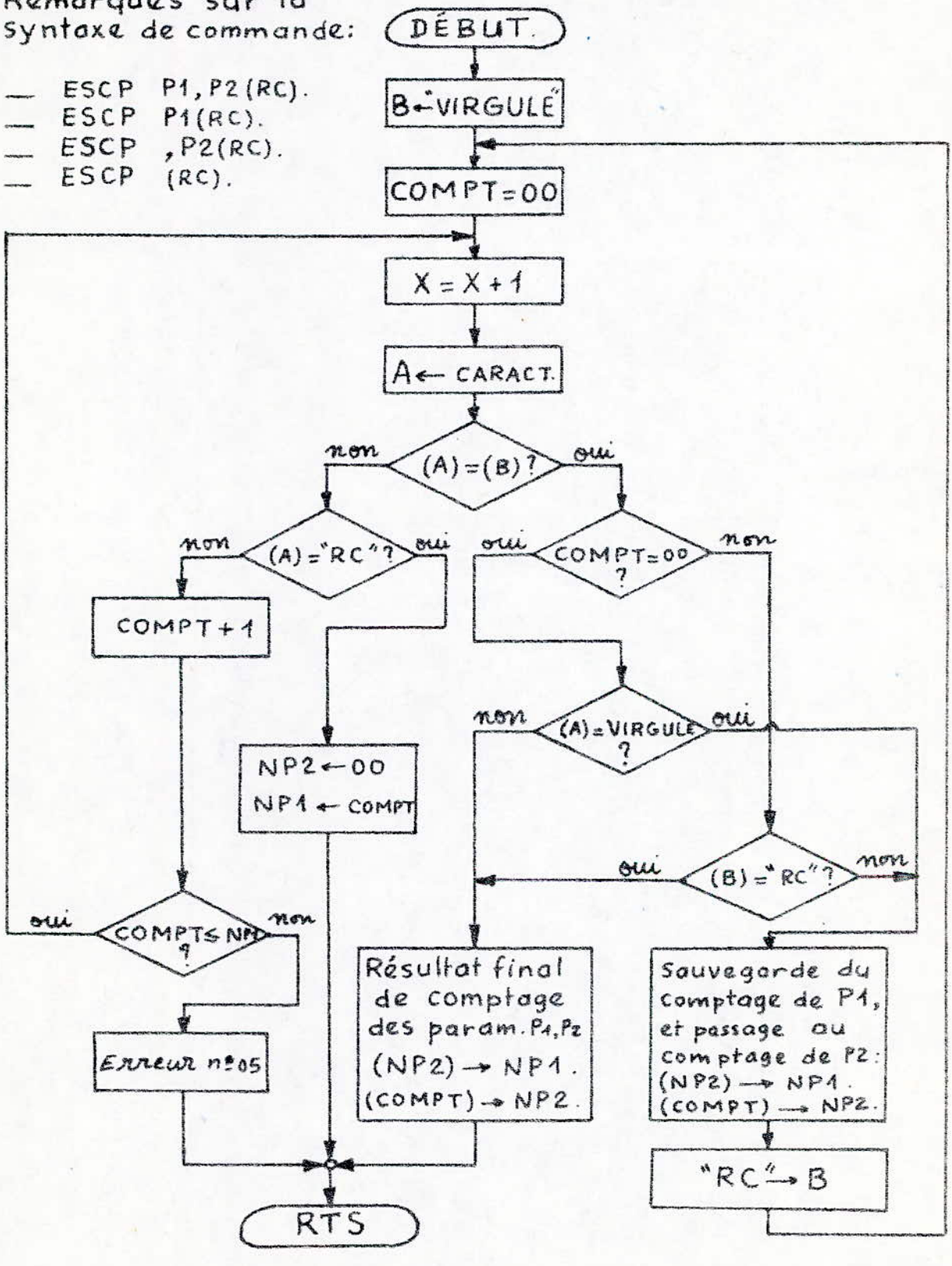
— Indexe, pointant le dernier caractère compté dans le paramètre P2.

Sous-Programme de COMPTage des caractères ASCII de P1 et P2

-SPCMPT-

Remarques sur la syntaxe de commande:

- ESCP P1,P2(RC).
- ESCP P1(RC).
- ESCP ,P2(RC).
- ESCP (RC).



4.2.3.5: Sous-Programme de CONVersion, en hexadécimal, des caractères ASCII des paramètres P1, P2 ; il est noté : SPCONV :

A. Fonction :

Il fait appel au sous-programme SPAH pour convertir, un par un, les caractères ASCII constituant les paramètres P1 et P2. Le résultat de chaque conversion est stocké dans les positions mémoires (P2H, P2L).

B. Paramètres d'entrée :

— ERROR, initialement à zéro.

— COMPT, initialement à zéro.

— P2H, P2L, initialement à zéro (0000).

— NP1, contenant le nombre de caractères ASCII du paramètre à convertir.

— Indexe, pointant le premier caractère à convertir.

C. Zone de travail :

— P2H, P2L, où se trouve le paramètre après conversion.

— NP1, contenant le nombre de caractères ASCII constituant le paramètre à convertir.

D. Paramètres de sortie :

— ERROR = 00, si pas d'erreur de conversion.

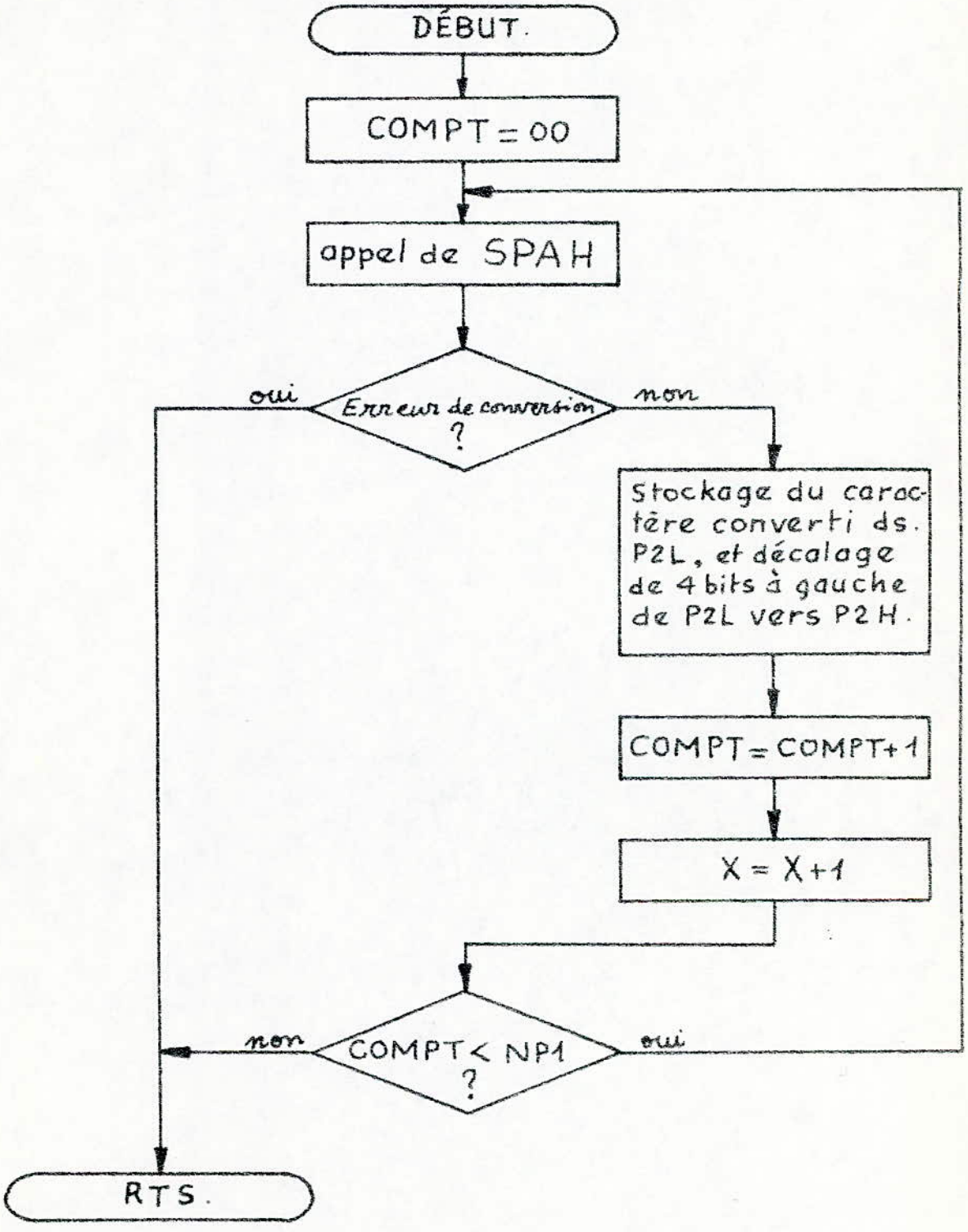
≠ 00, en cas d'erreur.

— P2H, P2L contenant le paramètre converti.

— Indexe, pointant le dernier caractère converti.

E. Sous-programmes appelés : SPAH.

organigramme du Sous-Programme de CONVersion des caractères ASCII de P1 et P2 en caractères héxa : SPCONV.



4.2.3.6: Sous-Programme de conversion ASCII/HEXA d'un des caractères suivants :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Il est noté : SPAH.

A. Fonction :

Il convertit un caractère ASCII en un des caractères hexadécimaux suivants : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Ceci étant fait si et seulement si le caractère X à convertir est tel que : $30 \leq X \leq 39$ ou bien : $41 \leq X \leq 46$; autrement, il s'agira d'une erreur de conversion.

B. Paramètres d'entrée :

— ERROR, initialisé à zéro.

— Index, pointant l'adresse du caractère à convertir.

C. Zone de travail :

— Accumulateur A.

D. Paramètres de sortie :

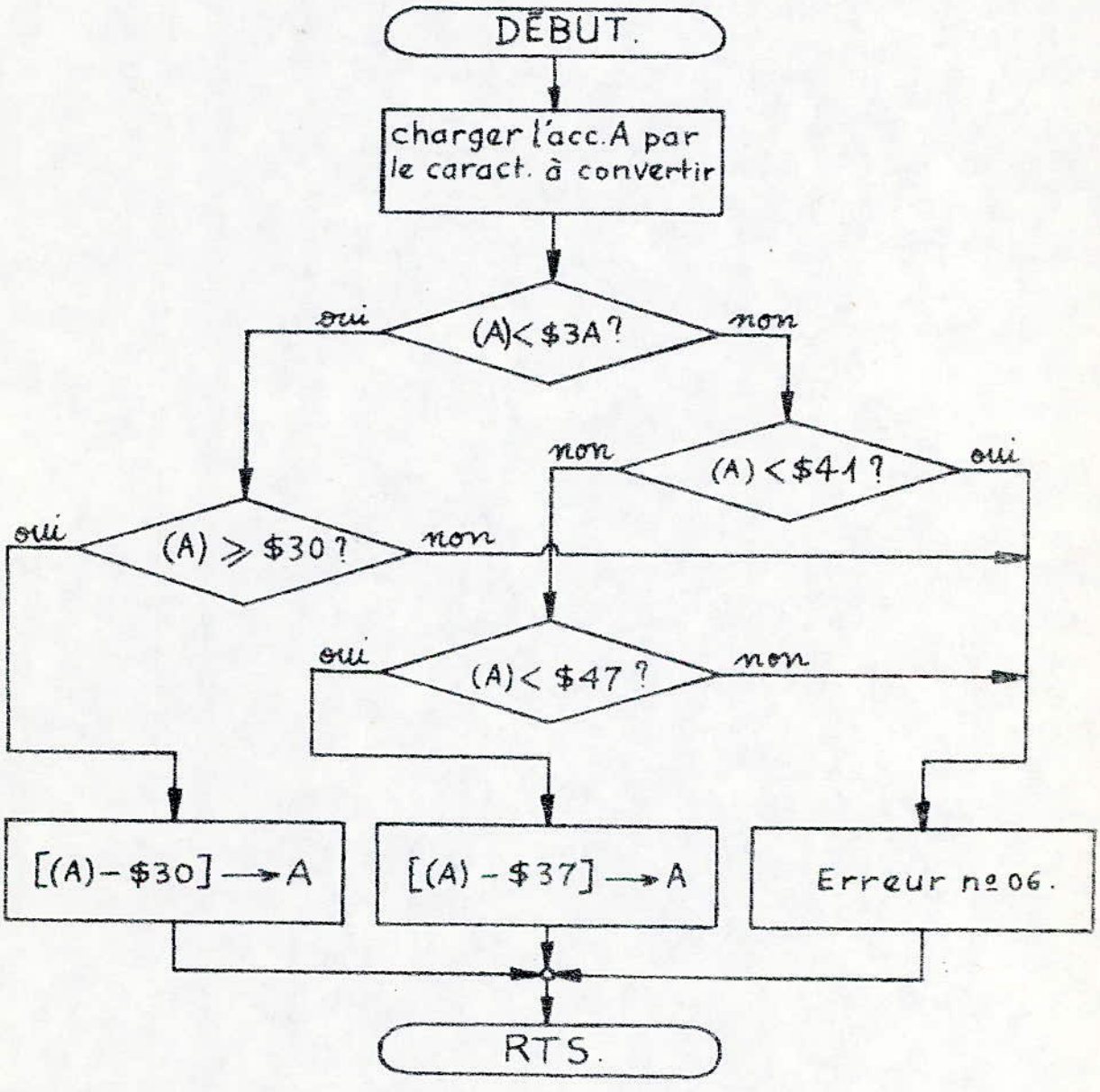
— ERROR = 00, si pas d'erreur de conversion.
= 06, si le caractère est inconvertible.

— Accumulateur A contenant le caractère converti s'il n'y a pas eu d'erreur. En cas d'erreur, le contenu de A sera sans importance.

— Index, pointant l'adresse du caractère qui vient d'être converti.

Organigramme du Sous-Programme de conversion Ascii/Hexa d'un des caractères : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Il est noté : SPAH.



4.2.4 : EXECUTION DE LA COMMANDE

___ RAPPEL :

Dans le paragraphe précédent, nous avons donné les organigrammes des sous-programmes qui accomplissent l'analyse de la commande reçue. Après cette étape, si la commande a été correctement formulée, nous avons toutes les informations nécessaires à l'éxécution de la commande. Au retour de l'analyse de la commande, une série d'instructions, dans le programme principal, permettent un branchement au sous-programme d'éxécution de la commande concernée.

Dans ce qui suit, nous allons voir ces différentes éxécutions :

4.2.4.1 : EXECution de la commande Z, EXECZ :

A. Fonction :

Elle consiste à faire du buffer (RAM de 16 K-octets) une image de l'état vierge de la PROM à programmer.
Ainsi, ce sous-programme "lit" le numéro d'identification de la PROM pour en déduire son état vierge :
Si celle-ci est fournie avec un niveau logique bas "0", on mettra tout le buffer à zéro.
Si, au contraire, elle est fournie (à l'état vierge) avec un niveau logique haut "1", on mettra le buffer à "1".

B. Paramètres d'entrée :

— Valeur lue sur le switch à 8 positions, donnant une combinaison binaire pour identifier la PROM à programmer et en déduire son état vierge.

— index pointant l'adresse de début du buffer ($0800).

C. Zone de travail :

— Accumulateur B, pour lecture du switch.

— Accumulateur A, contenat la valeur à stocker ($00, ou $FF).

D. Paramètres de sortie :

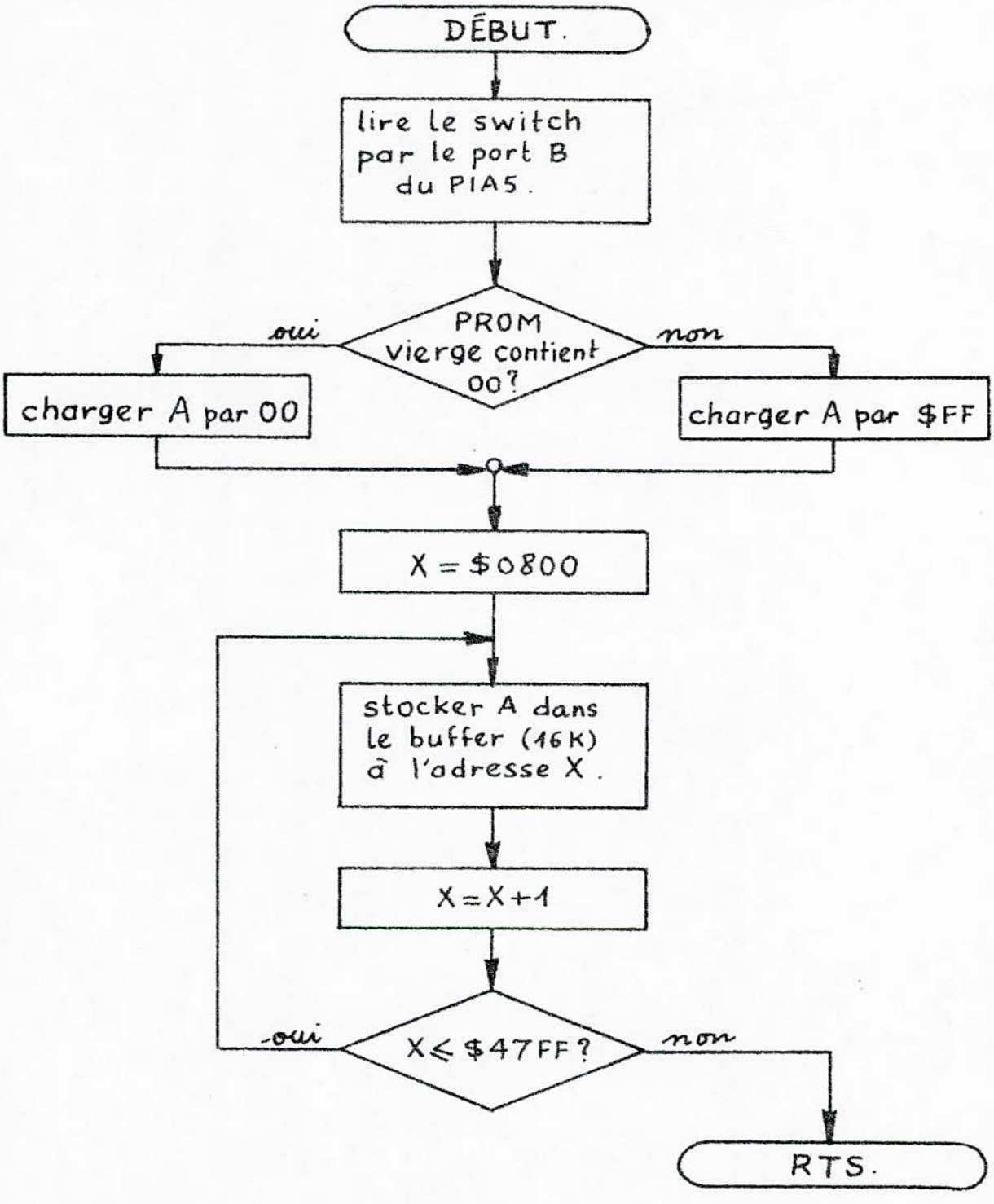
— Index pointant l'adresse de fin du buffer ($47FF).

— Accumulateur A contenant $00 ou $FF.

— Accumulateur B contenant le numéro d'identité de la PROM.

— Buffer contenant dans toutes ses positions mémoires des zéros ($00) ou des "un" ($FF).

Organigramme d'EXECution de la commande Z, EXECZ :



4.2.4.2: EXECution de la commande E, EXECE :

A. Fonction :

Elle consiste à mettre dans l'octet ETALON (réservé en RAM de 2K) la valeur binaire $FF, indiquant ainsi que le buffer est remplacé par une PROM étalon, et que, par conséquent, toutes les opérations de programmation ou de vérification doivent se faire en prenant cette PROM comme référence.

B. Paramètres d'entrée :

— ETALON, initialement à zéro.

C. Zone de travail :

— Accumulateur A.

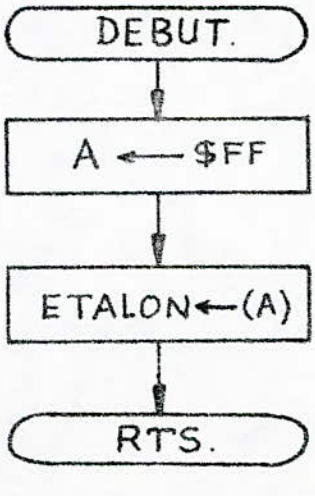
— Octet ETALON.

D. Paramètres de sortie :

— ETALON, contenant la valeur $FF.

— Accumulateur A, contenant aussi $FF.

Organigramme de 'EXECE'



4.2.4.3 : EXECution de la commande R , EXECR :

A. Fonction :

Consiste à réinitialiser les paramètres de travail du programme, et remettre le pointeur de pile à la valeur indiquant l'adresse du "sommet" de la pile (soit : $07FF).

B. Paramètres d'entrée :

- OCT,
- ETALON,
- ERROR,
- Pointeur de pile (SP)

} contenant des valeurs quelconques.

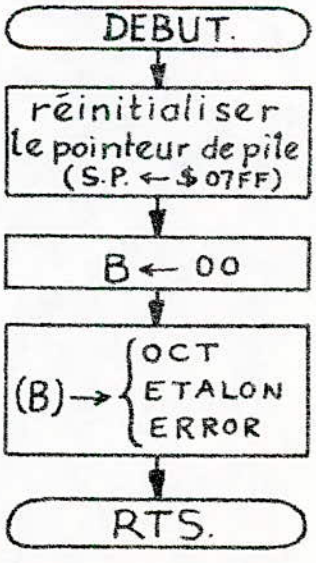
C. Zone de travail :

- Ce sont les paramètres d'entrée + accumulateur B.

D. Paramètres de sortie:

- Accumulateur B à zéro.
- Pointeur de pile indiquant le "sommet" de la pile ($07FF).
- OCT, à zéro.
- ERROR, à zéro.
- ETALON, à zéro.

Organigramme de 'EXECR' :



4.2.4.4 : EXECution de la commande de chargement, C, notée : EXECC,

— Introduction :

La commande de chargement consiste à transférer les données à programmer du buffer de réception, BUFFC, vers le buffer de 16 K, (RAM). Les données sont reçues de l'extérieur, ligne par ligne, à travers l'ACIA (caractère par caractère). Une ligne de données est une suite de caractères, ayant un format déterminé. Ceci nous incite donc à donner un bref aperçu sur les formats utilisés. Dans notre programme, nous en avons prévu deux :

— FORMAT INTEL, noté : 00 :

Dans ce format, une ligne de données s'écrira, d'une façon générale, comme suit :

:	N_1 N_2	A_1 A_2 A_3 A_4	L_1 L_2	D_1 D_2 ..... D_N	C S	RC	LF

: — un caractère ASCII représentant l'entête de la ligne.

$N_1 N_2$, — 2 caractères ASCII dont la conversion donne un octet hexa. La valeur hexa de $N_1 N_2$ indique le nombre d'octets hexa de données par ligne. Donc $N_1 N_2 \times 2$ donne le nombre de caractères de données ASII par ligne sans tenir compte des caractères de CS.

$A_1 A_2 A_3 A_4$, — 4 caractères ASCII représentant, après conversion en hexa, l'adresse de début de stockage de la ligne courante.

$L_1 L_2$, — 2 caractères ASCII indiquant la nature de la ligne courante :
(30, 30) ⇒ ligne de données.
(30, 31) ⇒ ligne de fin de fichier.

$D_1 D_2 \ldots D_N$ — caractères ASCII représentant les données ; la conversion de deux caractères ASCII donne un octet hexa.

CS — "Check Sum", 2 caractères ASCII permettant de vérifier si le nombre total de caractères par ligne est correct.

RC — un caractère ASCII qui veut dire : "retour-chariot" (retour à la ligne).

LF — un caractère ASCII qui signifie "Line-feed" c'est-à-dire : " interligne ".

— FORMAT MOTOROLA, notée : 01 (hexa).

Dans ce format, une ligne de données aura la forme générale suivante :

S	L	$N_1 N_2$	$A_1 A_2 A_3 A_4$	$D_1 D_2 \ldots\ldots D_{N-1}$	C S	RC	LF

S — un caractère ASCII représentant l'entête de la ligne.

L — un caractère ASCII déterminant la nature de la ligne courante :

(30) ⇒ début de fichier.

(31) ⇒ ligne de donnée.

(39) ⇒ ligne de fin de fichier.

N_1N_2 — deux caractères ASCII dont la conversion en hexa donne le nombre d'octets hexa de données par ligne, y compris les caractères CS.

$A_1A_2A_3A_4$ — 4 caractères ASCII dont la conversion en hexa donne l'adresse de stockage du 1er octet de donnée de la ligne courante dans la PROM à programmer.

$D_1D_2 \ldots D_{N-2}$, caractères ASCII de données.

CS — "Check-Sum" ayant le même rôle que pour le format Intel.

RC — "Retour-Chariot".

LF — "Line-Feed", (interligne).

Remarque :

Vu la souplesse de notre programme, d'autres formats peuvent être admis, à condition de prévoir les modifications adéquates au niveau du programme d'éxécution.

Commentaires sur : EXECC,

A. Fonction :

Il "lit" le numéro du format, calculé pendant l'analyse de la commande C, dans la position mémoire P2L :

— 00 ⇒ format INTEL, et renvoie au sous-programme correspondant, noté : SPINT ;

— 01 ⇒ format MOTOROLA, et renvoie au sous-programme correspondant, noté : SPMOT ;

— autre valeur ⇒ format inadmis, et erreur n° 07, avec possibilité d'extension à d'autres formats, dans l'avenir.

B. Paramètres d'entrée :

— P2L, contenant le numéro du format utilisé ($00 ou $01).

— P1H, P1L : contenant l'adresse virtuelle de la PROM à programmer (début de programmation).

— Index, indiquant l'adresse de début de TABFOR, table des sauts vers les sous-programmes correspondant aux formats utilisés.

— ERROR, initialisé à zéro.

C. Zone de travail :

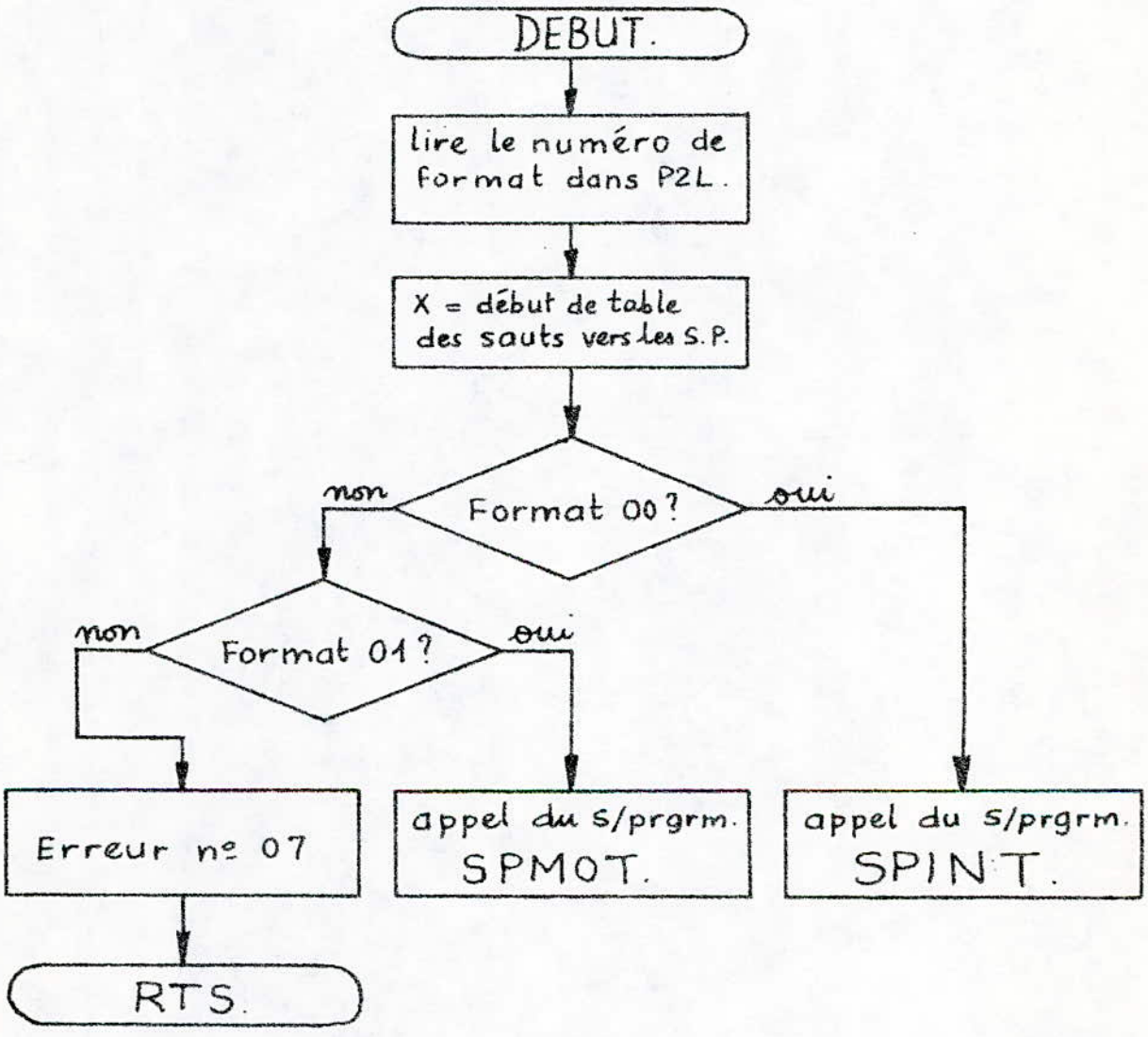
— Accumulateur A, pour lecture du numéro de format.

D. Paramètres de sortie :

— ERROR, = 00 si pas d'erreur sur le numéro de format.
= 07, si le format est autre que celui de Intel, ou Motorola.

— Index, pointant, dans TABFOR, l'adresse de début du sous-programme de traitement correspondant au format choisi; ceci si pas d'erreur; sinon, l'indexe sera sans importance.

Organigramme de : EXECC ;



RTS : "Retour de Sous-programme".

* Sous-Programme de traitement d'une ligne de données ayant le format INTel : SPINT.

A. Fonction:

— teste l'octet OCT pour savoir si une ligne de données est reçue.
— si oui, il vérifie si l'entête de la ligne (:) est conforme au format; dans le cas contraire, il s'agira d'une erreur n° 08.
— si l'entête est correct, il appelle des sous-programmes de traitement de la ligne.
— au retour de chaque sous-programme, il teste s'il n'y a pas eu d'erreur, sinon, il renvoie au traitement d'erreur.

B. Paramètres d'entrée:

— BUFFC : contenant la ligne de données à traiter.
— OCT mis à $FF ⟹ ligne de données reçue dans BUFFC.
— ERROR, initialement à zéro.
— Index, pointant l'adresse de l'en-tête de ligne dans BUFFC.

C. Zone de travail:

— Accumulateurs A et B.
— COMPT, servant de compteur pour la suite des S./program.
— RAM de 16K. où seront stockées les données traitées.

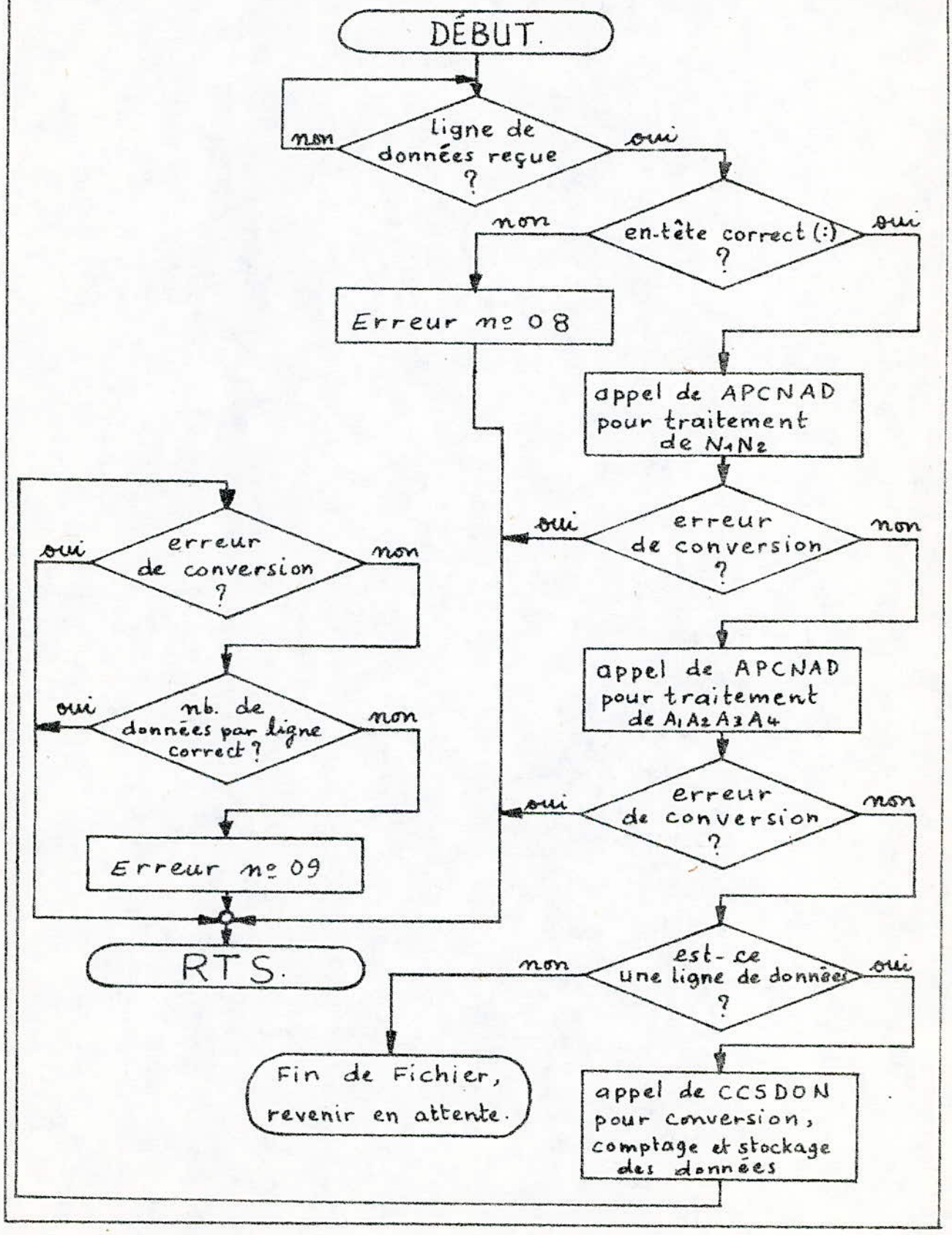
D. Paramètres de sortie:

— ERROR = 00, si pas d'erreur.
≠ 00, si une erreur est commise pendant le traitt.
— Buffer (RAM de 16K) contenant les données converties.

E. Sous-programmes appelés par SPINT:

— APCNAD, — CNAD, — CCSDON.

Organigramme du Sous-Programme de traitement d'une ligne de données de format INTel, noté : SPINT :



* Sous-Programme de traitement d'une ligne de données de format MOTorola, noté SPMOT :

A. Fonction :

— teste l'octet OCT pour savoir si une ligne de données est reçue.
— si oui, il vérifie si l'en-tête (S) est conforme au format choisi; autrement, il s'agira d'une erreur n° 08.
— si l'en-tête est correct, il appelle des sous-programmes pour le traitement de la ligne courante.
— Au retour de chaque sous-programme, il teste s'il n'y a pas eu d'erreur ; sinon, il renvoie au "traitement d'erreur".

B. Paramètres d'entrée :

— BUFFC, contenant la ligne de données à traiter.
— OCT contenant $FF ⇒ ligne de données reçue dans BUFFC.
— ERROR, initialement à Zéro.
— Index, pointant l'adresse de l'en-tête de ligne dans BUFFC.

C. Zone de travail :

— Accumulateurs A et B.
— COMPT, servant de compteur dans la suite des s./program.
— RAM (buffer) de 16K. où seront stockées les données traitées.

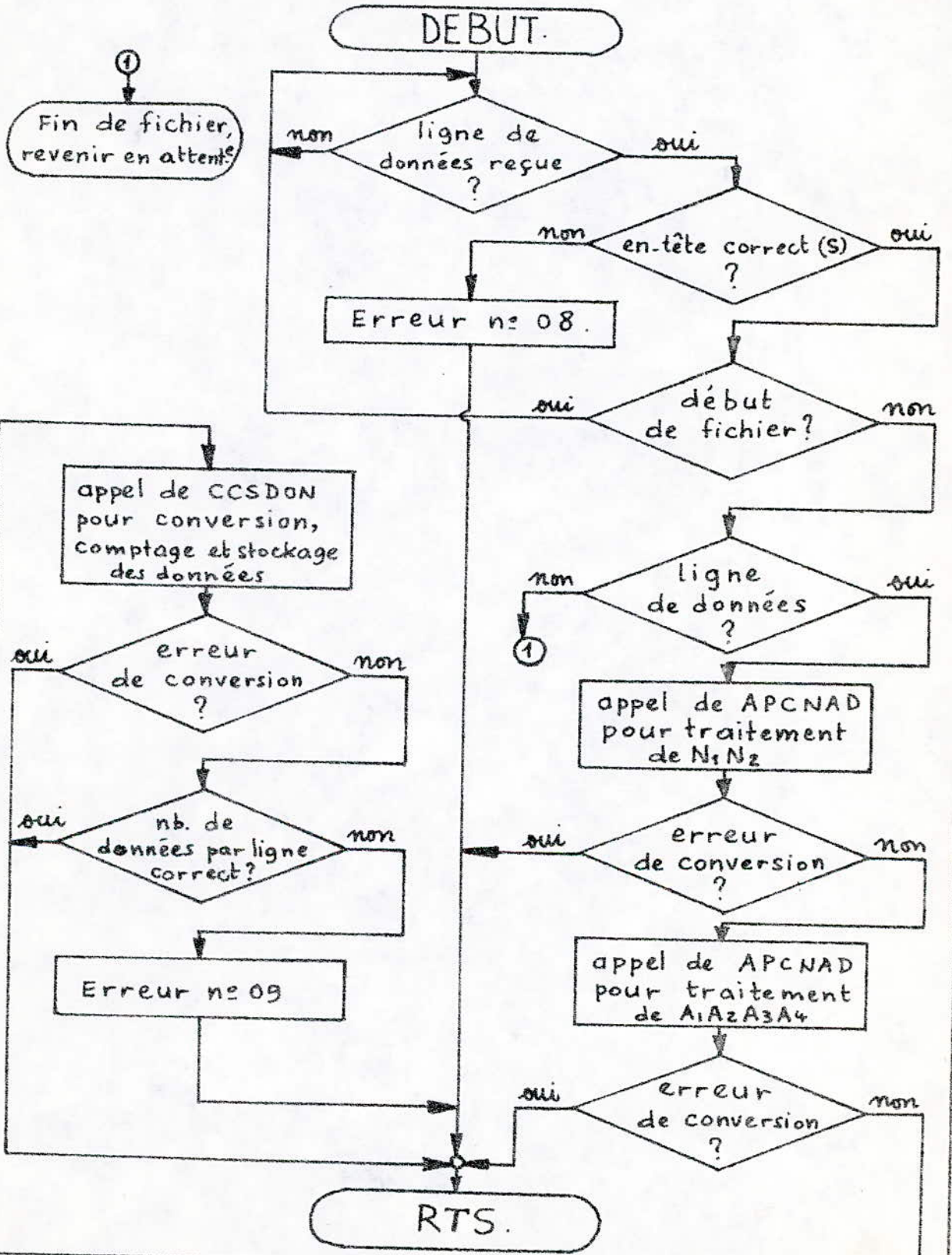
D. Paramètres de sortie :

— ERROR = 00, si pas d'erreur.
≠ 00, si une erreur est commise pendant le trait^{t}.
— RAM (de 16K), contenant les données converties (si pas d'erreur)

E. Sous-programmes appelés par SPMOT :

— NATLIN, — APCNAD, — CCSDON.

Organigramme du Sous-Programme de traitement d'une ligne de données de format MOTorola : SPMOT.



* Sous-Programme d'APpel de CNAD : APCNAD.

A. Fonction :

Il charge d'abord le compteur par la valeur 02 et appelle le sous-programme CNAD pour convertir en hexa les caractères ASCII, N_1N_2 précédemment définis (voir rappels sur les formats Intel, et Motorola).

Puis il charge le compteur par la valeur 04 et appelle, pour la 2ème fois, le sous-programme CNAD pour la conversion des caractères adresses : $A_1A_2A_3A_4$.

Au retour du 1er appel, il sauvegarde le résultat de la conversion si pas d'erreur; sinon RTS.

Au retour du 2ème appel, il sauvegarde aussi le résultat de la conversion si pas d'erreur, puis renvoie au sous-programme CALADS pour le calcul de l'adresse de stockage.

B. Paramètres d'entrée :

- Index pointant, dans BUFFC, l'adresse du 1er caractère de N_1N_2.
- COMPT, initialisé à la valeur 02.
- ERROR, initialisé à zéro.
- BUFFC, contenant la ligne de données à traiter.

C. Zone de travail :

- NP1, où l'on sauvegarde le résultat de conversion de N_1N_2
- ADRH, ADRL, 2 octets dans lesquels sera sauvegardé le résultat de conversion de l'adresse reçue : $A_1A_2A_3A_4$.

D. Paramètres de sortie :

— ERROR = 00 , si pas d'erreur.

≠ 00, si erreur de conversion.

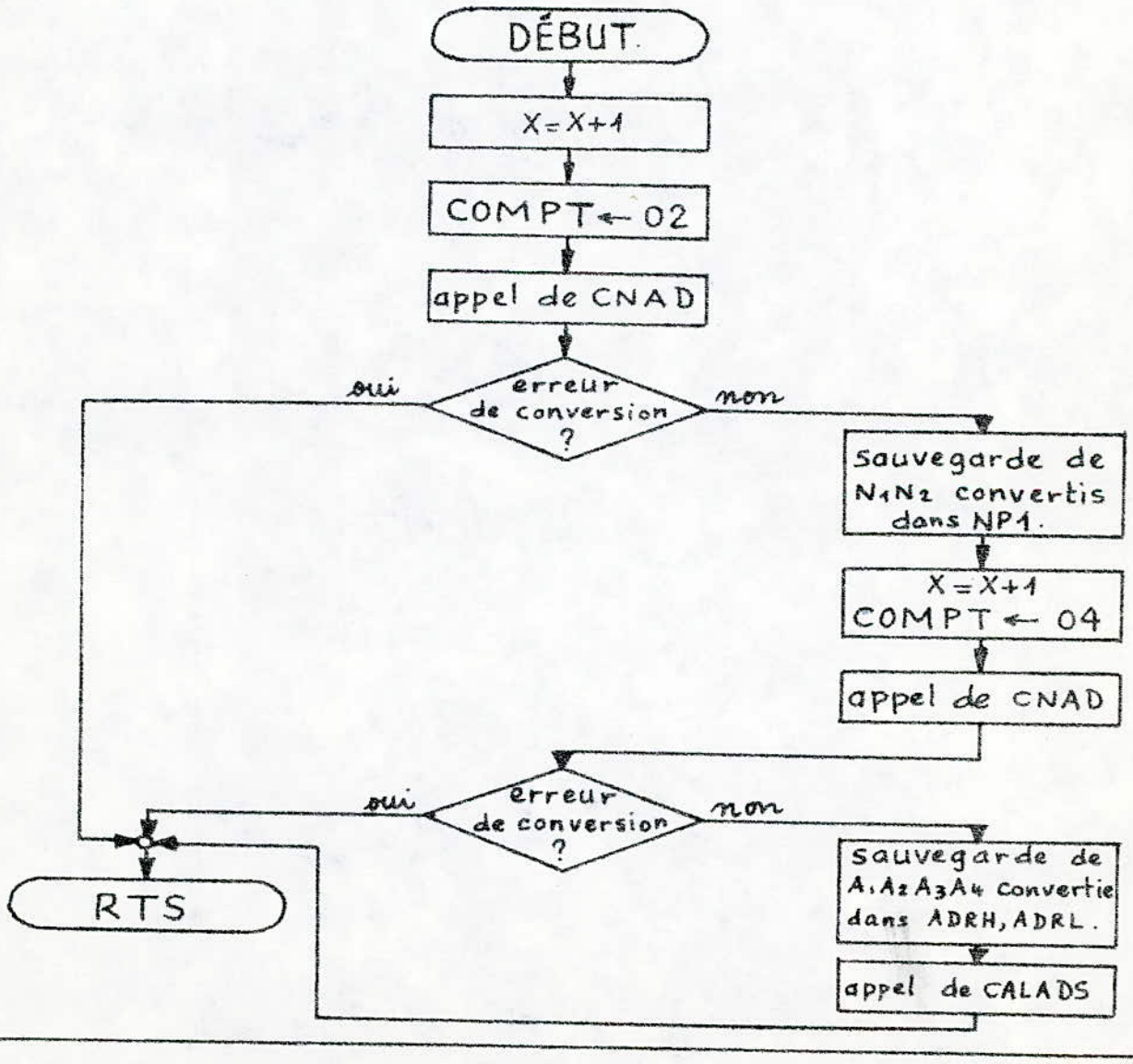
— NP1, contenant la valeur hexa de N_1N_2 converti.

— ADRH, ADRL : contenant l'adresse de stockage du 1er octet de la ligne courante, dans la RAM de 16K.

E. Sous-programmes appelés :

— CNAD (2 fois), — CALADS.

* Organigramme du sous-programme APCNAD :



* Sous-Programme de Conversion de N_1N_2 et de l'ADresse, CNAD

A. Fonction :

Il appelle le sous-programme SPAH (déjà défini), pour convertir les caractères ASCII de N_1N_2 et $A_1A_2A_3A_4$, en hexa. À chaque retour de SPAH, il teste s'il y a erreur de conversion, sinon, il stocke le résultat de conversion dans deux octets ZOTRA1, ZOTRA2, puis retourne au point d'appel.

B. Paramètres d'entrée :

— Index, pointant le 1er caractère à convertir.

— ERROR, initialisé à zéro.

— ZOTRA1, ZOTRA2, initialisés à zéro.

— COMPT, contenant le nombre de caractères ASCII à convertir.

— BUFFC, contenant la ligne à traiter.

C. Zone de travail :

— ZOTRA1, ZOTRA2, servant à contenir le résultat de conversion après chaque retour de SPAH.

D. Paramètres de sortie :

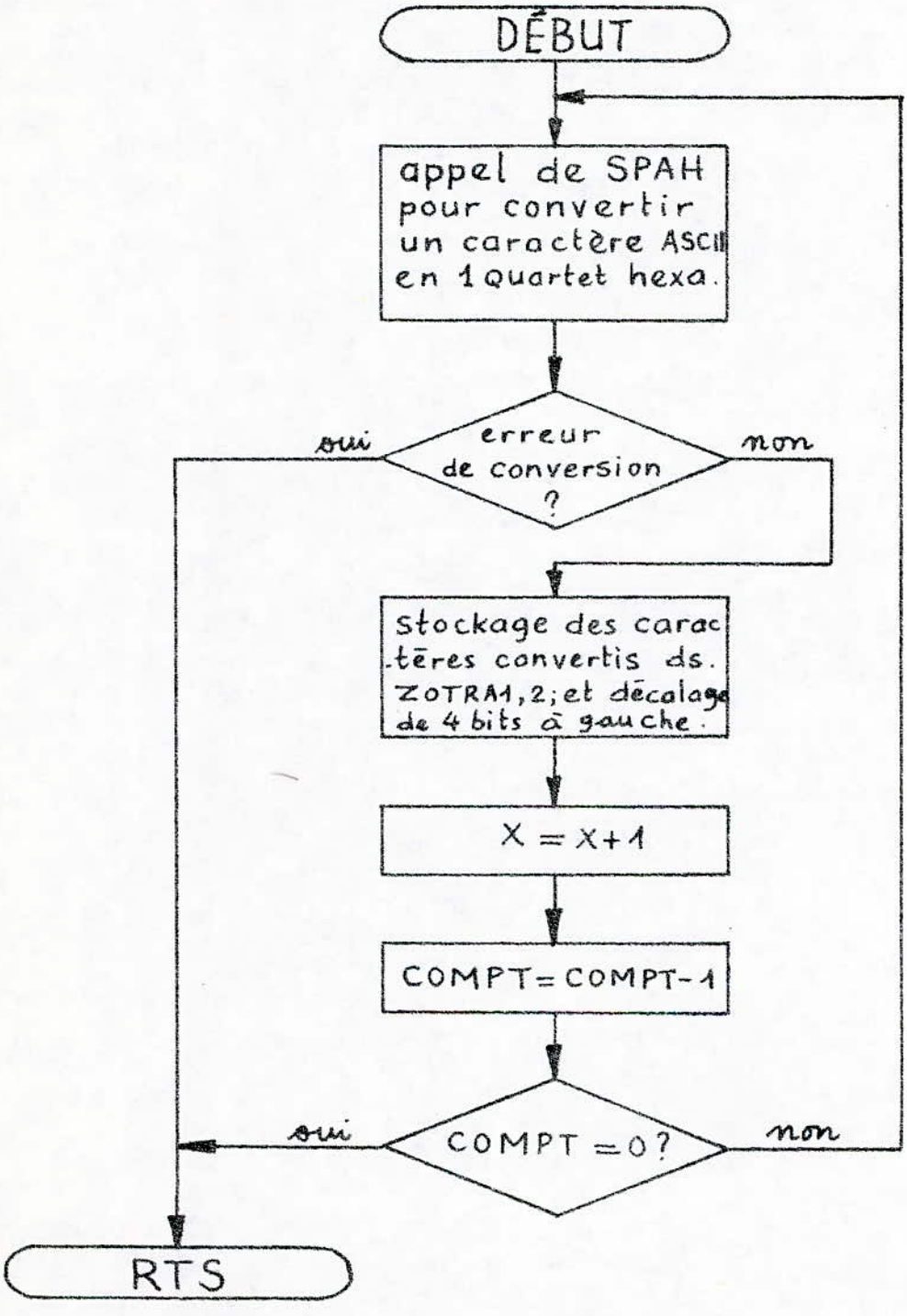
— ZOTRA1, ZOTRA2, contenant le résultat de conversion.

— Index pointant l'adresse du dernier caractère converti.

E. Sous-programmes appelés :

— SPAH.

Organigramme du sous-programme de Conversion de N_1N_2 et des ADresses, noté : CNAD.



* sous-programme de CALcul d'ADresse de Stockage : CALADS.

A. Fonction :

Il commence par vérifier si l'adresse de stockage de la ligne de données courante ne dépasse pas la capacité de la PROM à programmer ; autrement, il s'agira d'une erreur n° $0A.
Si pas de dépassement, il effectue le calcul d'adresse de début de stockage de la ligne courante comme suit :

Adr. de Stock. = [$A_1A_2A_3A_4$ − (P1H, P1L) + $0800] ⟶ (ADRH, ADRL).

où : . $A_1A_2A_3A_4$ est l'adresse reçue dans la ligne (et convertie).
. (P1H, P1L) contiennent l'adresse virtuelle de début de programmation de la PROM à programmer.
. $0800, adresse de début du buffer de stockage (RAM de 16K).

Le calcul ci-dessus vise à rendre le buffer, à partir duquel se fera la programmation (voir EXECP), une image directe de la PROM à programmer.

B. Paramètres d'entrée :

— ERROR, initialisé à zéro.
— ADRH, ADRL : 2 octets contenant, après conversion en hexa, l'adresse $A_1A_2A_3A_4$ reçue dans la ligne de données.
— NP1, contenant le résultat de conversion de N_1N_2.
— MAXH, MAXL : 2 octets contenant la capacité de la PROM à programmer.

C. Zone de travail :

— ADRH, ADRL : 2 octets dans lesquels sera placé le résultat du calcul de l'adresse de stockage cité ci-dessus.

D. Paramètres de sortie :

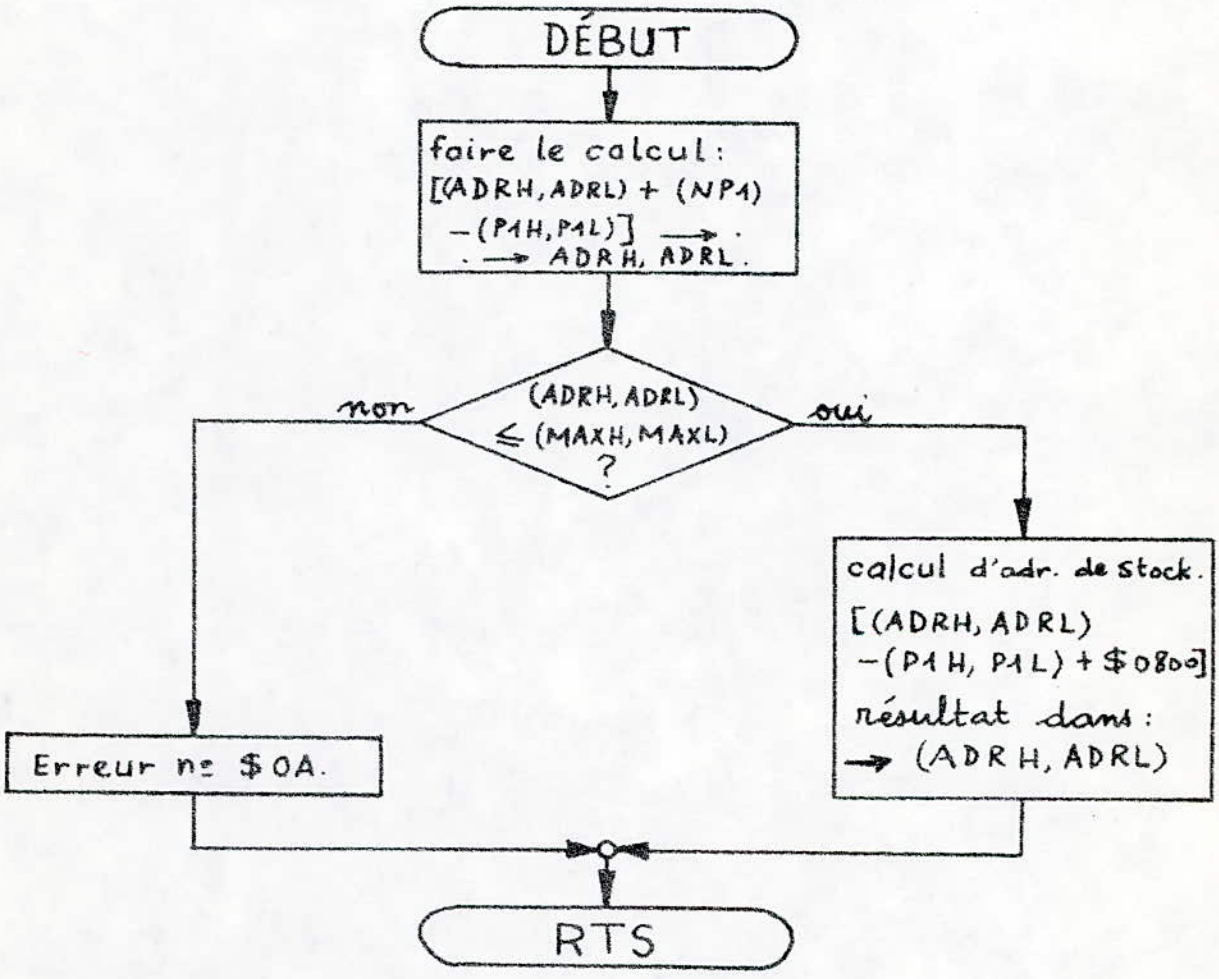
— ERROR = 00, si pas d'erreur.

≠ 00, si dépassement de capacité de la PROM.

— ADRH, ADRL, contenant le résultat du calcul de l'adresse de stockage précité.

Ceci, si pas d'erreur; sinon ces deux octets auront un contenu sans signification.

* Organigramme du sous-programme CALADS :



* sous-programme de conversion en hexa des deux caractères ASCII indiquant la NATure de la LIgNe de données : NATLIN.

A. Fonction :

Il appelle le sous-programme SPAH pour convertir les 2 caractères L_1L_2 (précédemment définis) et en faire un octet hexa dont la valeur indique le type de ligne courante : est-elle une ligne "début de fichier", une ligne de données, ou une ligne "fin de fichier" ? (voir rappels sur les formats des données).

B. Paramètres d'entrée :

— ERROR, initialisé à zéro.

— ZONATL, initialisé à zéro.

— Index, pointant l'adresse du caractère à convertir.

C. Zone de travail :

— ZONATL, destiné à contenir le caractère "nature de ligne" après sa conversion en hexadécimal.

D. Paramètres de sortie :

— ERROR = 00, si pas d'erreur de conversion.

≠ 00, autrement.

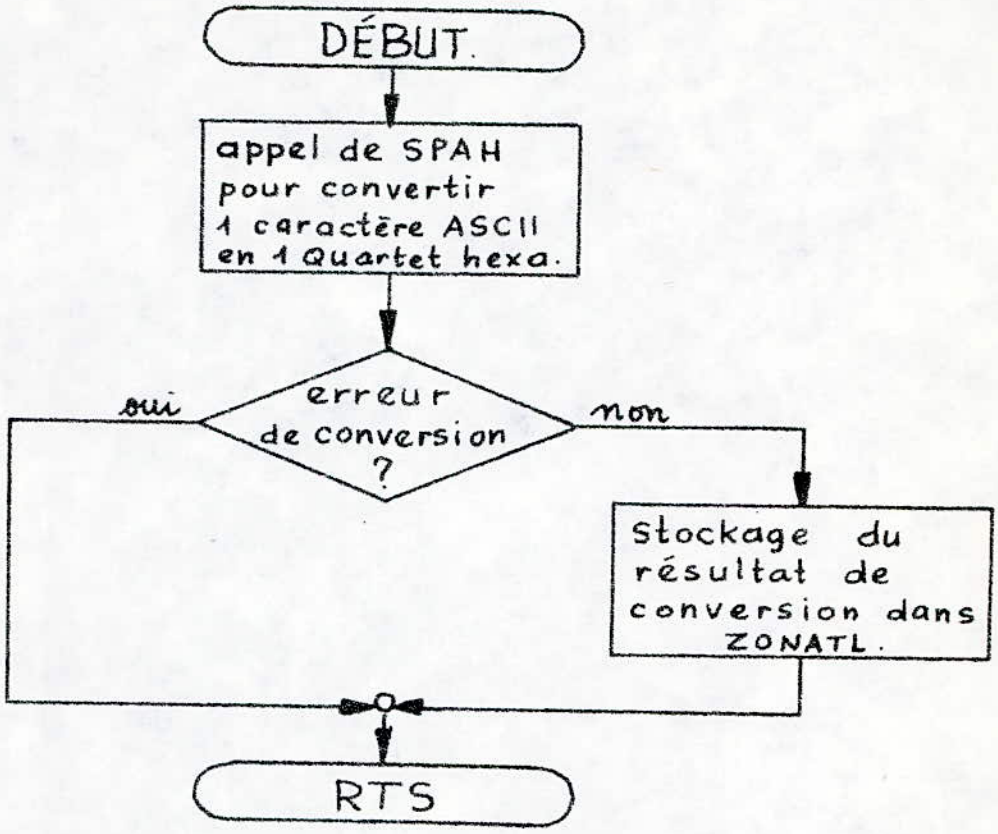
— ZONATL, contenant la valeur définissant le type de ligne, si pas d'erreur ; sinon, sans importance.

— Index, pointant l'adresse du caractère converti.

E. Sous-programmes appelés :

— SPAH.

organigramme du sous-programme noté : NATLIN.



* Sous-programme de Comptage, Conversion et Stockage des caractères de DONnées d'une ligne ; noté : CCSDON :

A. Fonction :

Il "prend" les caractères ASCII des données d'une ligne, 2 à 2, il en fait un octet hexa et le stocke dans le buffer à l'adresse indiquée par l'index (ce dernier est chargé de ADRH, ADRL). Tous les caractères de données d'une ligne sont convertis et stockés. Les deux caractères du "check-Sum" (CS) sont convertis mais ne sont pas stockés dans le buffer, car le compteur des caractères à stocker est comparé à (NP1) qui contient le nombre de données sans tenir compte du (CS).

B. Paramètres d'entrée :

— ZOTRA3, à zéro.

— COMPT, à zéro.

— ERROR, à zéro.

— NP1, contenant le nombre de caractères de données.

— BUFFC, contenant la ligne de données à stocker.

— Index pointant, dans BUFFC, l'adresse du 1er caractère de donnée à stocker.

— ADRH, ADRL : contenant l'adresse image où sera stocké le 1er caractère de donnée dans le buffer de 16 k.

C. Zone de travail :

— ZOTRA3, contenant l'octet de donnée converti (à stocker).

— Buffer (RAM de 16 k) dans lequel seront stockées les données après leur conversion en hexa.

D. Paramètres de sortie :

— ERROR = 00, si pas d'erreur.

≠ 00, si erreur.

— COMPT, indiquant le nombre de caractères de données convertis et stockés.

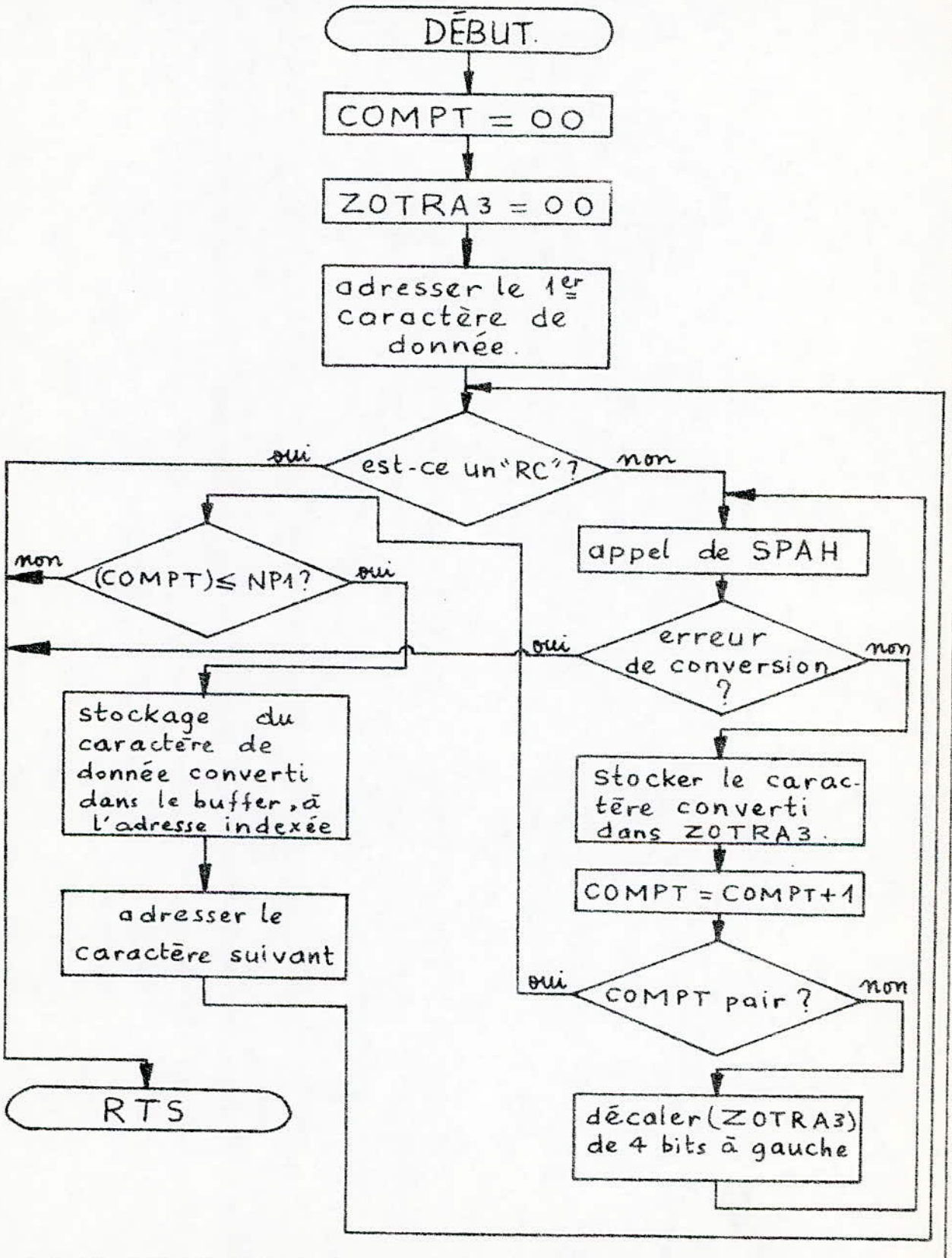
— Index, pointant l'adresse du dernier caractère de la ligne (c'est-à-dire RC : "retour-chariot").

— Buffer, contenant les données, si pas d'erreur ; sinon il n'a pas d'importance.

E. Sous-programmes appelés :

— SPAH, pour la conversion en hexa de chaque caractère de donnée.

Organigramme du sous-programme de Comptage, Conversion et Stockage des DONnées : CCSDON.



4.2.4.5 : EXECution de la commande de Programmation : EXECP.

A. Fonction :

— Il programme le PIA5 à l'état requis (port A en sortie, port B en entrée) ;

— Il fait le transfert des informations contenues dans la table de la PROM à programmer vers une zone RAM de 8 octets (adresse de début : BUFTAB).

— Il teste l'octet ETALON pour savoir si la programmation va se faire à partir du buffer ou d'une PROM-étalon.

— Dans ce dernier cas, il fait appel au sous-programme de lecture pour transférer le contenu de la PROM vers le buffer à partir duquel se fera la suite de la programmation.

— Il teste si la PROM à programmer est vierge, sinon, il s'agira d'une erreur n° $0B.

— Enfin, il met le PIA3 en sortie et fait appel à un sous-programme de PROGRAMMATION correspondant à la PROM considérée (bipolaire ou non bipolaire).

— La vérification de la totalité du programme est faite immédiatement à la fin de l'opération de programmation.

B. Paramètres d'entrée :

— Périphériques (PIA3, PIA4, PIA5) tous en entrée par initialisation ; et alimentations programmables à Zéro.

— ERROR, initialisé à Zéro.

— TABLES des PROMs (ou REPROMs) contenant les informations nécessaires à leur programmation.
— octets (P1H, P1L) et (P2H, P2L) contenant les paramètres P1 et P2 de la commande P (convertis au cours de l'Analyse).
— ETALON à 00 ou FF selon que l'on désire une programmation à partir de la RAM ou de la PROM étalon. (dans le 1er cas, la RAM doit contenir les données à programmer préalablement chargées par EXECC).
— Combinaison se trouvant sur le switch à 8 positions, spécifiant le type de la PROM à programmer.

C. Zone de travail:

— PROM à programmer sur sa carte support.
— si le cas se présente, PROM de référence (ETALON).
— RAM de 16K contenant les données à programmer.

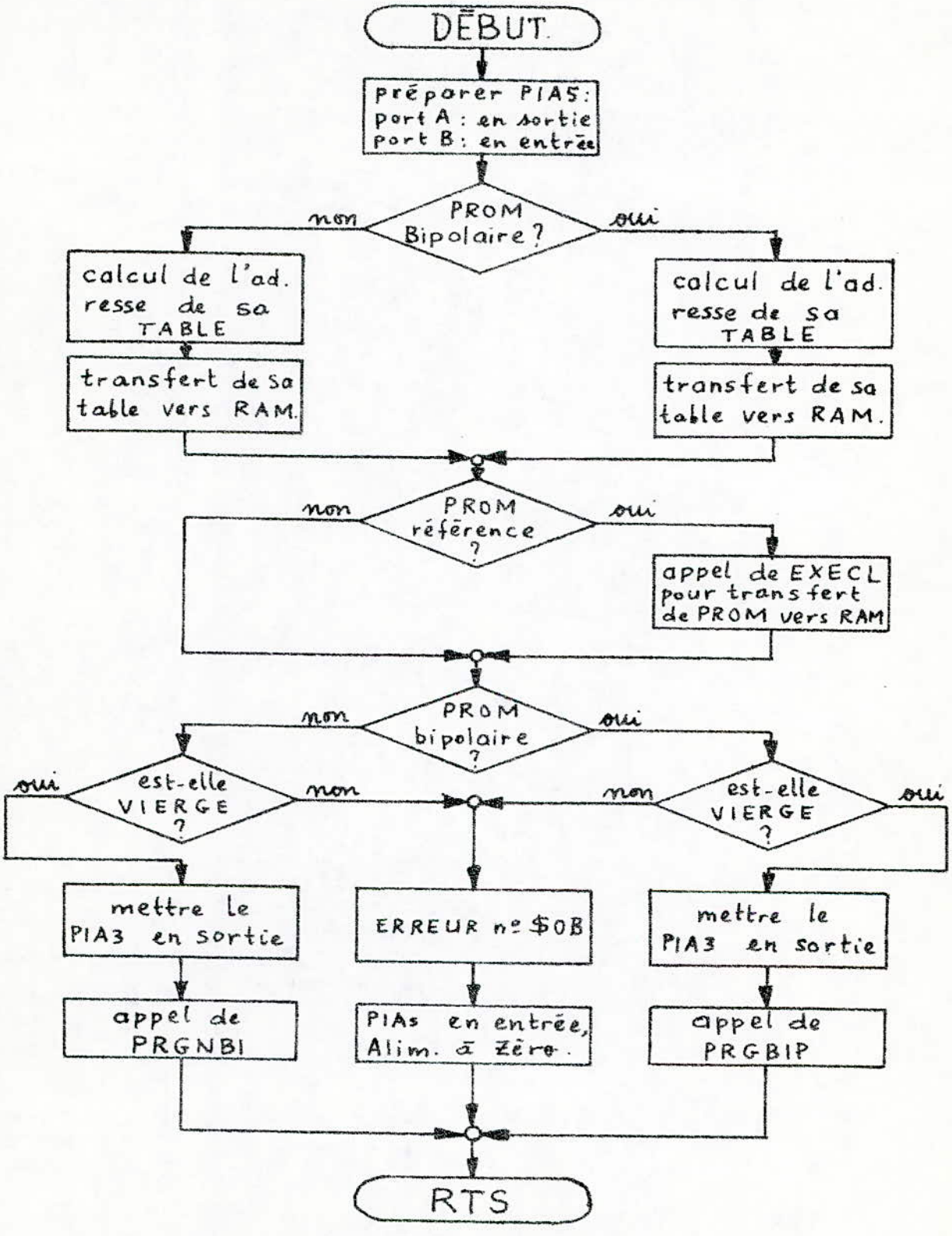
D. Paramètres de sortie:

— ERROR = 00, si pas d'erreur.
≠ 00, si une erreur est commise dans EXECP.
— PROM à programmer contenant les données qu'on voulait y introduire, si pas d'erreur; sinon sans importance.
— Index, pointant la dernière position mémoire programmée.
— tous les PIAs remis à l'état initial (en entrée.). et les alimentations remises à zéro.

E. Sous-programmes appelés:

— PRGBIP, ou — PRGNBI, — EXECV (vérification),
— EXECL (pour lecture de la PROM étalon).

Organigramme d'EXECution de la Programmation, EXECP :



* sous-programme assurant la PRoGrammation des PROMs Non-BIpolaires, noté: PRGNBI.

A. Fonction:

— Il prépare toutes les tensions nécessaires à la programmation et met l'impulsion de programmation à l'état inactif.
— Adresse le mot à programmer, et établit la donnée.
— Applique le front actif de l'impulsion de programmation à la broche concernée de la PROM.
— Fait une temporisation de 50 ms nécessaire à la programmation, puis remet l'impulsion à l'état inactif.
— Refait les pas précédents pour chaque mot à programmer.
— Vérifie, enfin, si le programme est correct; sinon, erreur n°: $0C.

B. Paramètres d'entrée:

— Buffer de 16 k. contenant les données à programmer.
— ERROR, initialisé à zéro.
— PIA5 (A en sortie; B en entrée); PIA3 et PIA4 en entrée (initial.).
— P1H, P1L, P2H, P2L, contenant les paramètres de la commande P.
— BUFTAB, contenant les informations de la table.

C. Zone de travail:

— LATCHES des alimentations programmables.
— SAUV1,2: sauvegardent l'index du buffer.
— SAUV3,4: sauvegardent l'index de la PROM à programmer.

D. Paramètres de sortie:

— ERROR = 00, si pas d'erreur; ≠ 00 autrement.

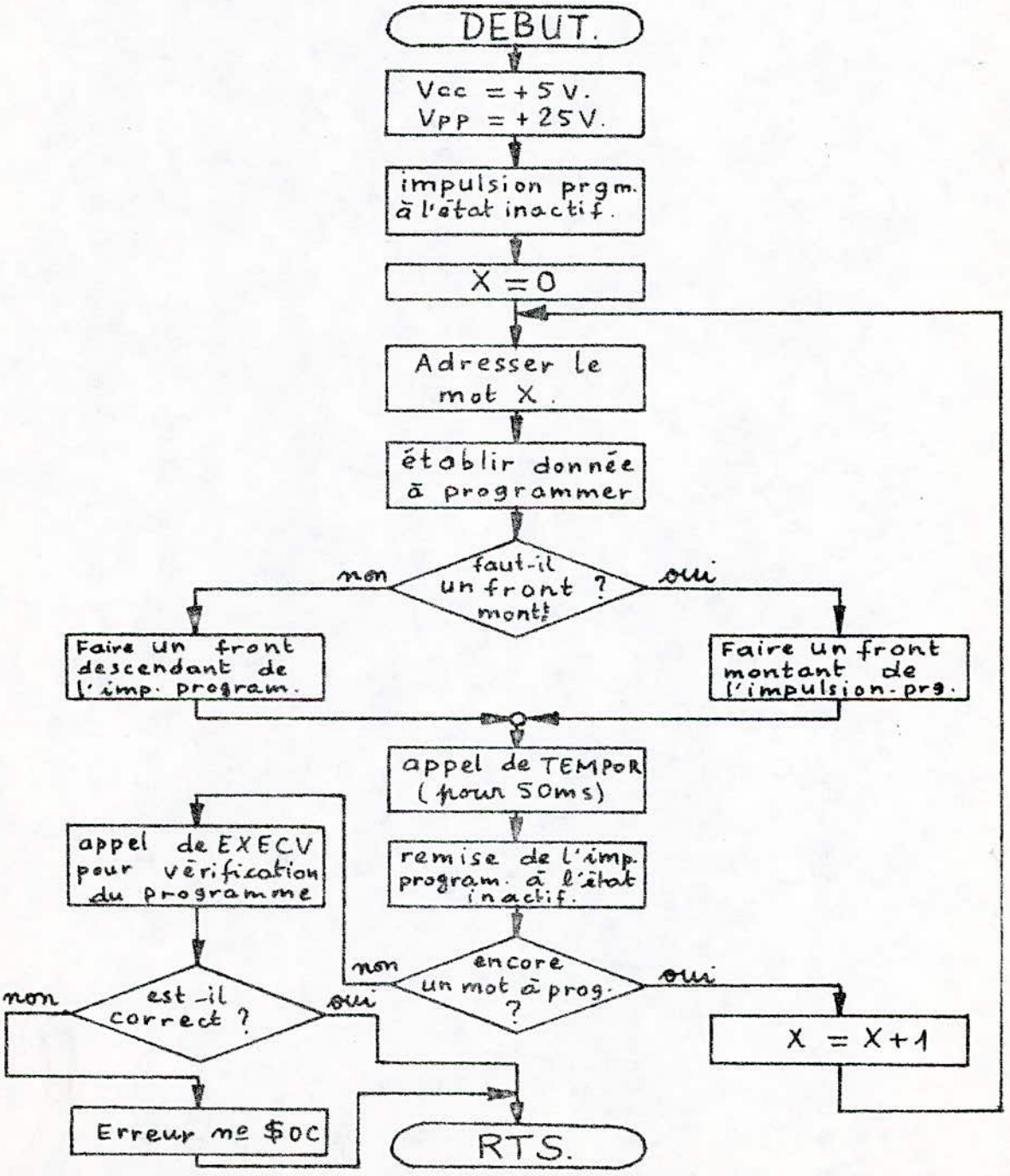
— PROM (ou REPROM) contenant les données, si pas d'erreur ; autrement, elle sera sans importance.

E. Sous-programmes appelés :

— TEMPO (pour temporisation de 50 ms).

— EXECV, à la fin de la programmation, (pour vérification).

* Organigramme de PRGNBI :



* sous-programme assurant la PROGrammation des PROMs BIPolaires ; noté : PRGBIP.

A. Fonction :

— Il teste (par l'intermédiaire du port B du PIA5) la combinaison du switch à 8 éléments pour savoir s'il s'agit de la série "14/18" ou "24/28".

— Il prépare, selon le cas, les tensions nécessaires à la programmation, et met la PROM étalon à l'état inactif.

— Assure la programmation " bit par bit " en respectant les étapes citées aux paragraphes (2-3-3 ; et 2-4-3).

— À la fin de la programmation, il vérifie si tous les mots programmés sont corrects (EXECV) ; sinon, erreur n°$0C.

B. Paramètres d'entrée :

— Les mêmes que pour PRGNBI.

C. Zone de travail :

— Les mêmes que pour PRGNBI.

D. Paramètres de sortie :

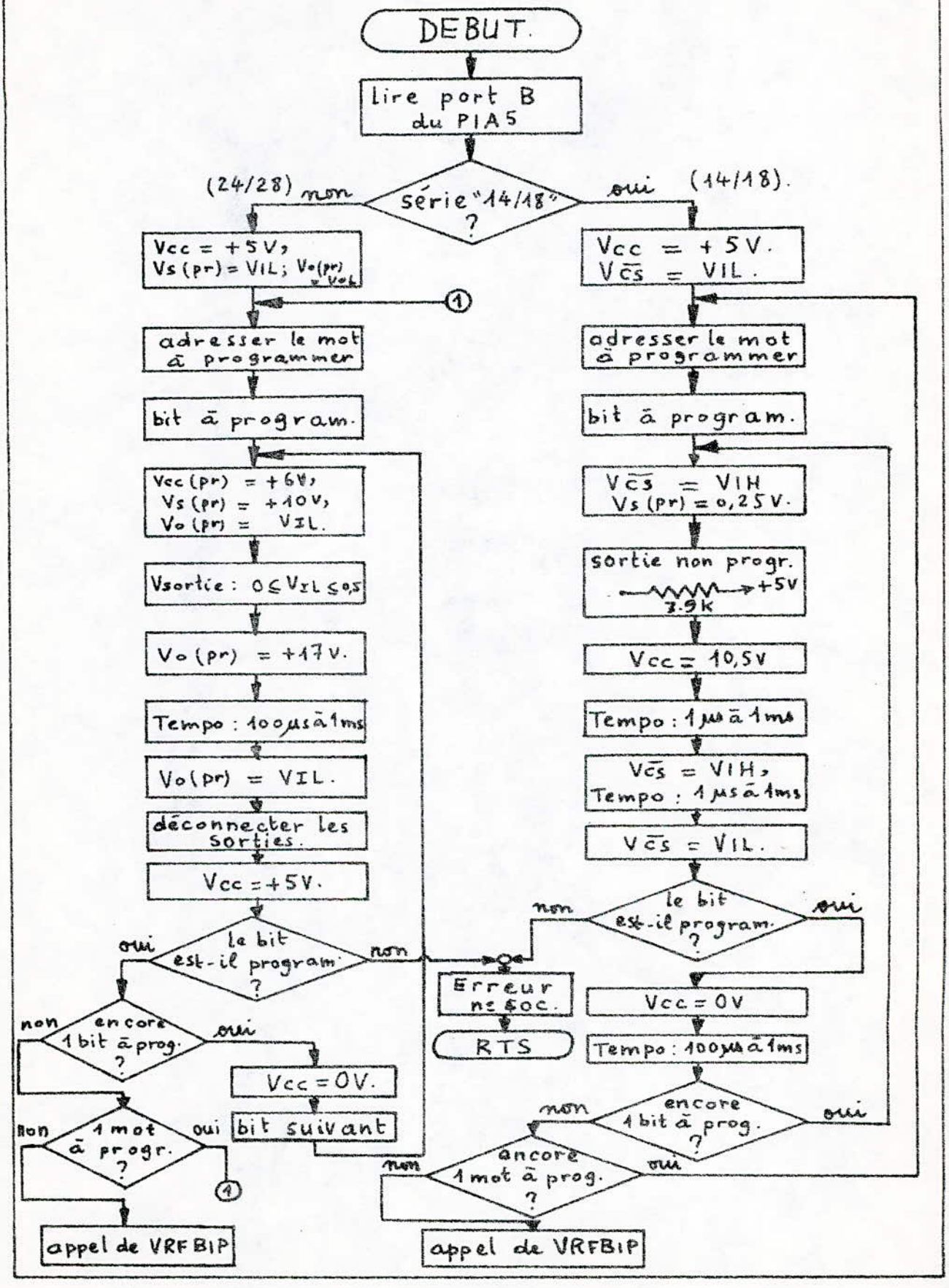
— Les mêmes que pour PRGNBI.

E. Sous-programmes appelés :

— TEMPOR, (pour les temporisations nécessaires).

— EXECV, pour vérification finale du programme.

Organigramme du sous-programme noté : PRGBIP.



4.2.4.6 : EXECution de la commande de Lecture : EXECL

A. Fonction :

— Il transfère le contenu d'une PROM (ou REPROM) vers la RAM de 16 K-octets (buffer).

— Il peut être appelé soit indépendamment, soit comme sous-programme au cours de l'éxecution : EXECP.

— Il met le PIA4 en sortie (pour adressage) ; PIA3 en entrée.

— Il détermine s'il s'agit d'1 Prom bipolaire ou non (en lisant le port B du pr

— Il renvoie, enfin, au sous-programme de lecture correspondant au type de PROM, après avoir mis la PROM à programmer à l'état inactif pour éviter des conflits éventuels sur le bus de données.

B. Paramètres d'entrée :

— PIA5 (port A en sortie, B en entrée).

— PIA4 en sortie, PIA3 en entrée, alimentations à Zéro.

— ERROR, initialisé à zéro.

— Combinaison sur le switch spécifiant la PROM à lire.

— P1H, P1L ; P2H, P2L, contenant les paramètres P1 et P2.

— Buffer dans lequel sera faite la "lecture".

C. Zone de travail :

— PROM à lire sur la carte support.

— Buffer vers lequel se fait le transfert.

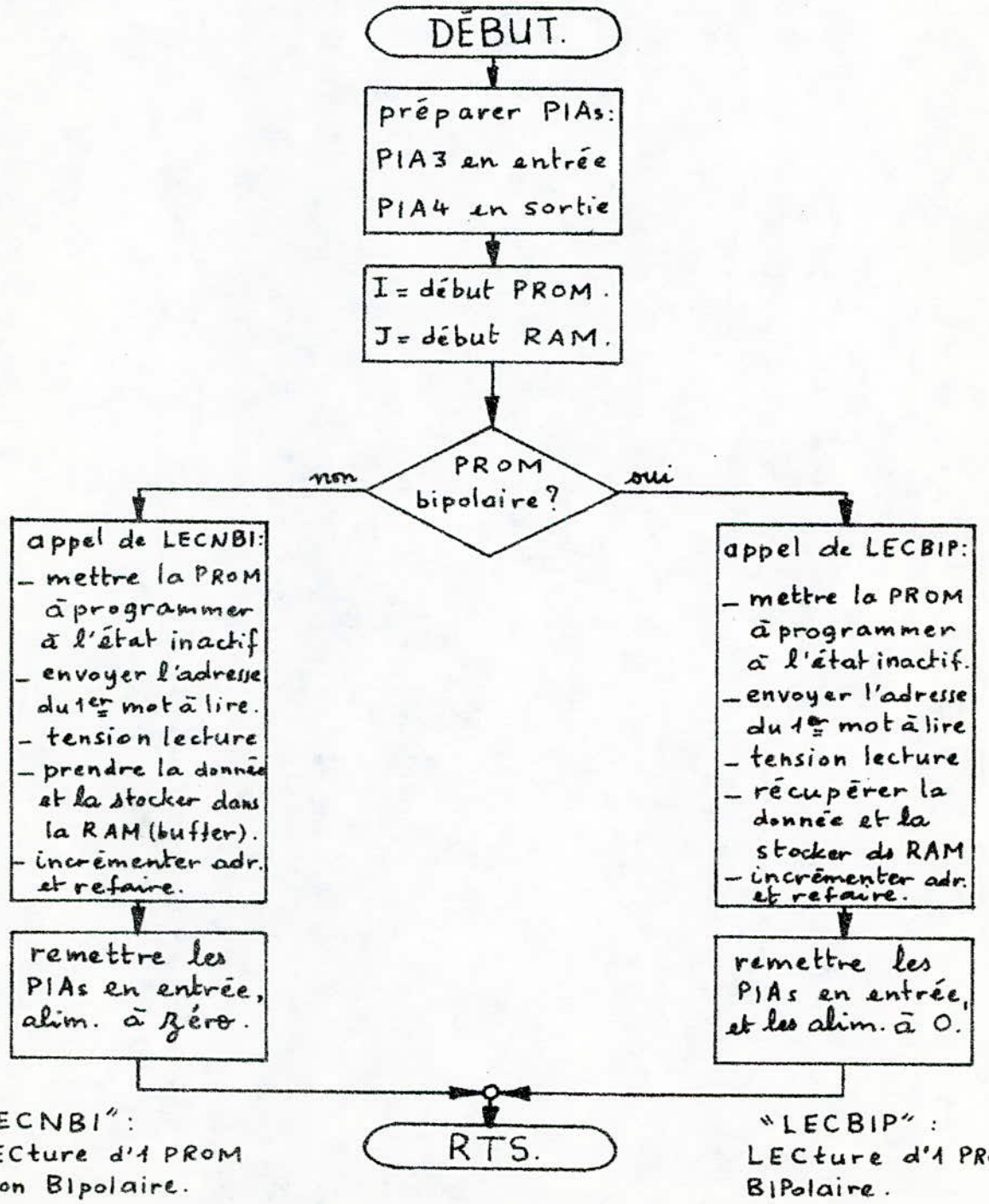
— LATCHES des alimentations programmables.

— Zones de sauvegarde des index (SAUV1, 2 : index buffer
SAUV3, 4 : index PROM).

D. Paramètres de sortie :

- ERROR = 00 (pas de possibilité d'erreur).
- BUFFER, contenant les données lues.
- Index, pointant l'adresse de la dernière donnée lue.

* Organigramme de EXECL :



"LECNBI" :
LECture d'1 PROM
Non BIpolaire.

"LECBIP" :
LECture d'1 PROM
BIPolaire.

4.2.4.7: EXECution de la commande de Vérification; noté : EXECV.

A. Fonction :

— Il compare le contenu de la PROM (ou REPROM) à celui du buffer à partir duquel a été faite la programmation.

— Il lit la combinaison sur le switch pour déterminer s'il s'agit d'une PROM bipolaire ou non.

— Selon le cas qui se présente, il renvoie au sous-programme de vérification qui lui correspond.

— Si un mot de la PROM diffère de celui qui lui correspond dans le buffer, il s'agira d'une erreur n° $0C.

B. Paramètres d'entrée:

— Les mêmes que pour EXECL.

C. Zone de travail:

— PROM à vérifier sur la carte support.

— Buffer auquel elle doit être comparée.

— Zones de sauvegarde de l'index :

SAUV1,2 : pour index du buffer.

SAUV3,4 : pour index de la PROM.

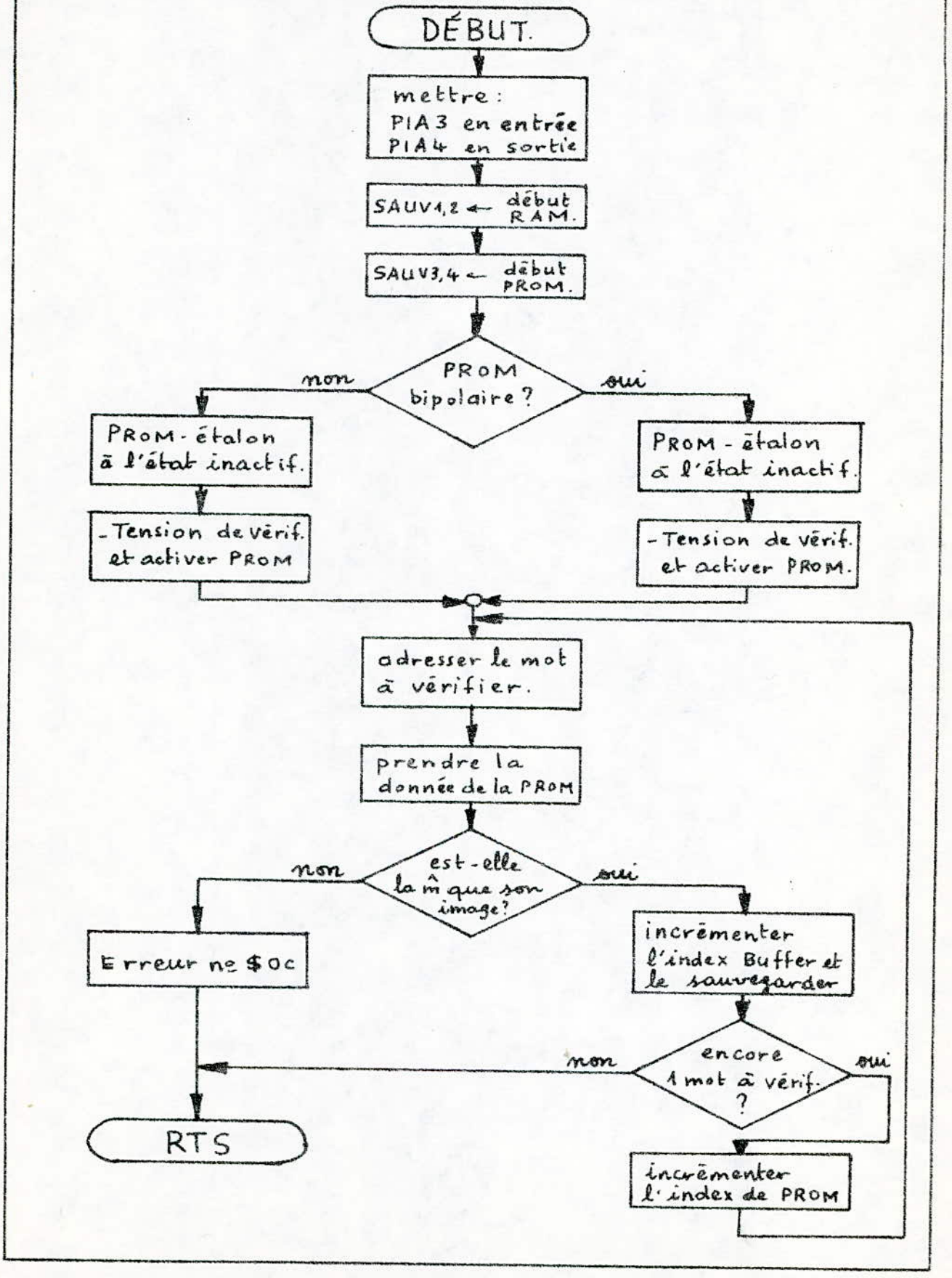
D. Paramètres de sortie :

— ERROR = 00 si la vérification est correcte.

≠ 00, autrement : (1 mot est différent de son "image").

— Index, pointant l'adresse du dernier mot vérifié.

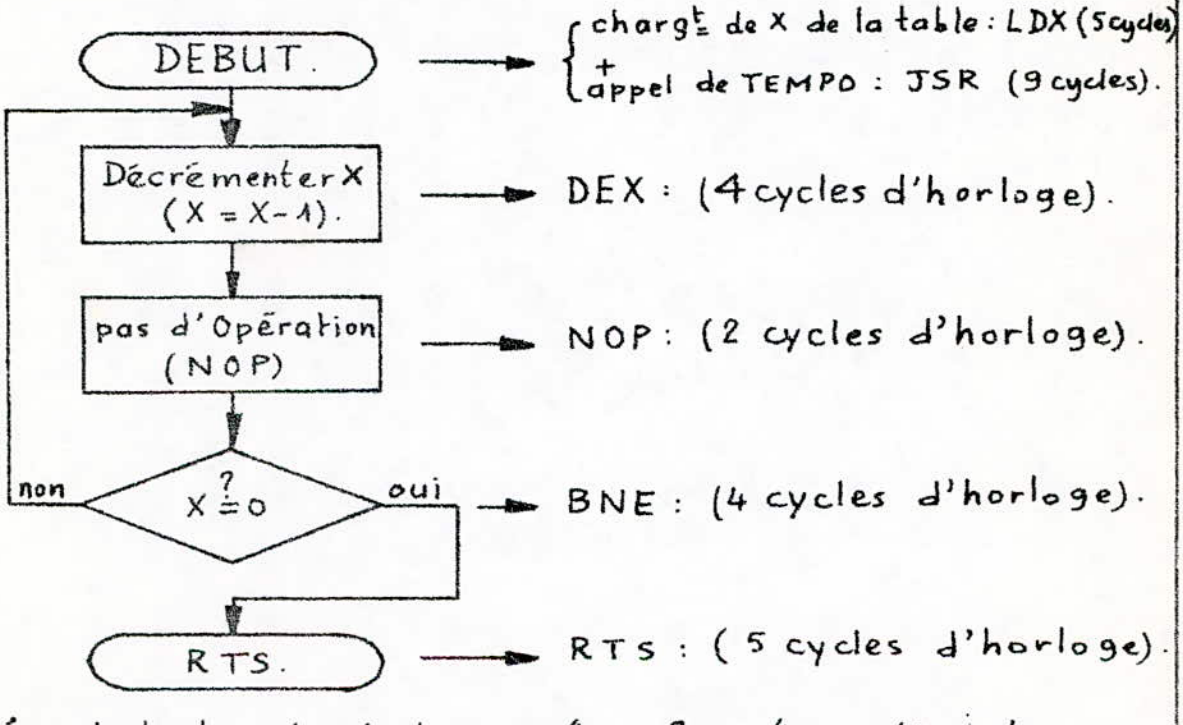
organigramme du sous-programme noté : EXECV.



* sous-programme de TEMPOrisation, noté : TEMPO.

Il consiste à réaliser une temporisation dont la durée dépend du nombre hexadécimal contenu dans le registre d'index. Avant d'appeler ce sous-programme, le registre d'index doit être chargé, à partir de la table des PROMs, par la valeur correspondant à la temporisation désirée; puis sera décrementé jusqu'à zéro.

* Organigramme de TEMPO :



. durée de la boucle de base : 4c + 2c + 4c = 10 cycles.

. si la fréquence d'horloge est de 1MHz ⟹ Tc = 1µs.

. la durée d'1 boucle de base sera donc : 10 x 1µs = 10µs.

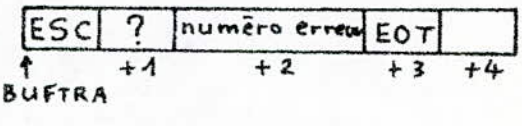
. Pour avoir 50ms = 50.000µs, il faut charger X par : 5000.

. En tenant compte du temps mis pour exécuter "début et RTS" (19c), puis la remise de l'impulsion de programmation à l'état inactif (EOR : 2 cycles), cela fera 2 boucles de moins ⟹ (X) = 4998 = $1386.

. de même, pour avoir 100µs ⟹ (X) = 10 - 2 = 8.

* sous-programme de TRAITement d'erreur : "TRAIT".

- Certaines erreurs peuvent être détectées pendant la réception, l'analyse, ou l'exécution de la commande. Elles sont repérées par des nombres (numéros) que l'on mettra dans l'octet ERROR et qui seront traités par ce sous-progr.
- De plus, si notre système est en liaison avec un ordinateur, on doit émettre le message suivant : ESC ?(numéro d'erreur). si, la liaison est avec écran : $?(numéro d'erreur)(en ASCII).
- Donc on appelle le sous-programmme de traitement d'erreur après avoir initialisé le buffer de transmission "BUFTRA" par le message :



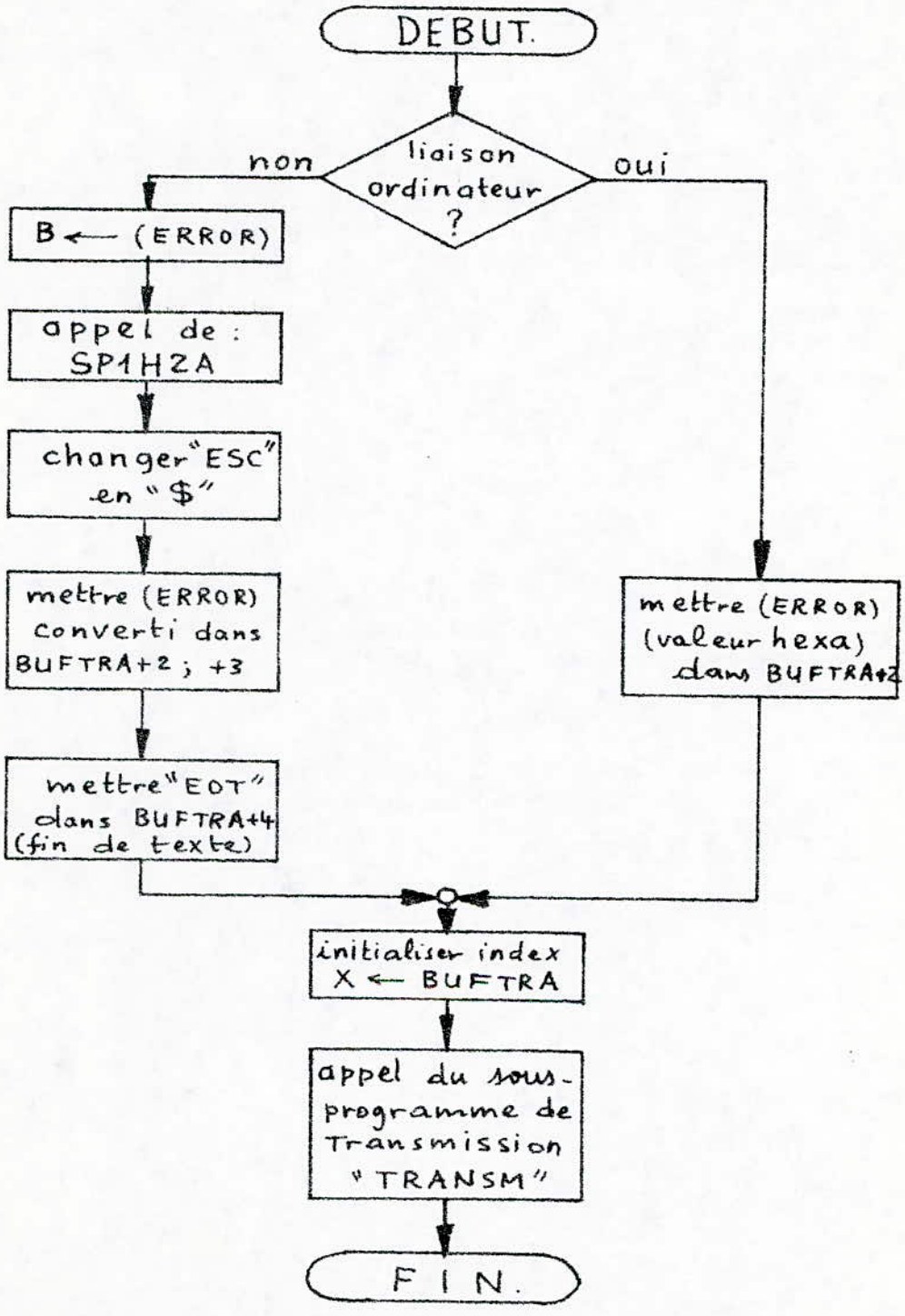
comme si le système était en liaison avec un ordinateur, puis on changera le contenu de BUFTRA si la liaison est avec écran.

- notons enfin que si le système est en liaison avec un écran, le numéro d'erreur (1 octet hexa) doit être converti en 2 caractères ASCII d'où la nécessité de faire appel à un sous-programme qui fait cette conversion (on le notera : SP1H2A).
- Dans ce qui suit, nous donnerons les organigrammes de " TRAIT " et " SP1H2A ".

Remarque :

la liste des numéros d'erreurs avec leur signification est donnée en annexe.

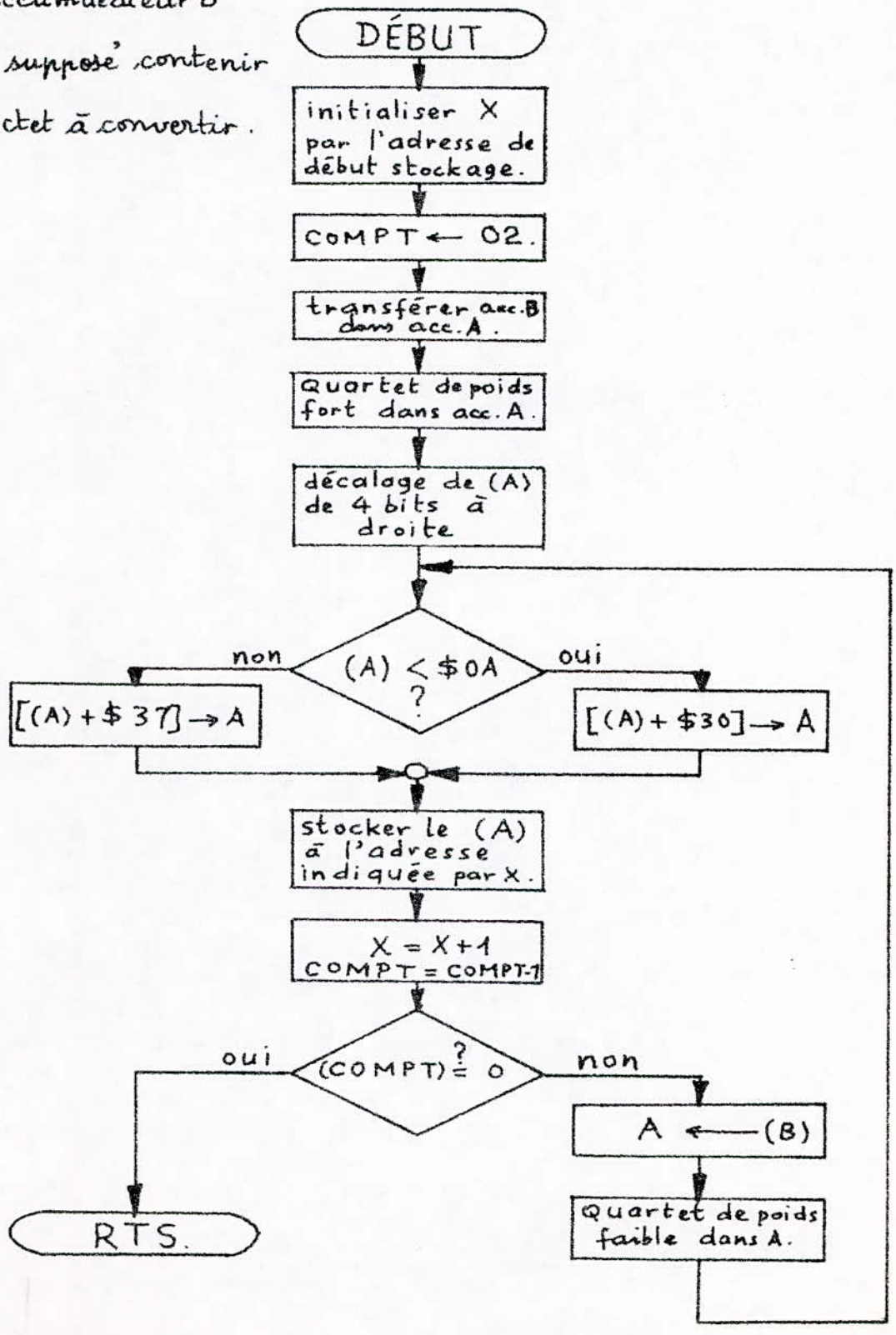
Organigramme du sous-programme de Traitement d'erreur, noté : "TRAIT".



N.B. pour le sous-programme de transmission, voir chapitre "réception de la commande" (page 49).

* sous-programme de conversion d'1 octet Hexa en 2 caractères ASCII, noté : SP1H2A.

l'accumulateur B est supposé contenir l'octet à convertir.



Chapitre 5: Essais et mise au point du programme.

5.1: Introduction:

Avant de charger définitivement un programme dans une mémoire morte, celui-ci a besoin d'être vérifié, mis au point, et si besoin est, modifié ou retouché.

Dans ce chapitre, nous présentons, en bref, les étapes de mise au point et d'essai du programme.

5.2: enregistrement du programme:

Après avoir écrit les instructions du programme sur du papier, on procèdera à son enregistrement sur une disquette souple. Ceci est fait grâce au système de développement offert par MOTOROLA, par l'intermédiaire de l'ensemble "clavier-écran-unité disque". Cette opération fait appel au programme dit: "Editeur".

5.3: phase d'assemblage:

Cette étape consiste à traduire les instructions écrites en langage mnémonique, en langage machine, en vue de l'éxécution du programme. Cette opération est accomplie grâce à un programme approprié, appelé: "ASSEMBLEUR".

5.4: Correction des erreurs de syntaxe:

Après assemblage et impression du programme sur listing, on procède à la correction des erreurs de syntaxe détectées par l'ASSEMBLEUR, telles que: "étiquettes indéfinies", "instructions mal écrites", "non respect des espaces entre champs"...

5.5 : Exécution du programme :

Le système d'aide à la mise au point (EXORTERM) offert par MOTOROLA permet seulement de tester les parties du programme qui ne mettent pas en jeu les organes du système étudié : (ACIA, PIA, alimentations programmables ... etc). La totalité du programme peut cependant être testée grâce à la technique d'"EMULATION" offerte par l'"EXORCISER".

Pour les tests préliminaires, nous avons pu suivre le déroulement correct du programme d'ANALYSE DE LA COMMANDE, ainsi que celui d'EXECUTION de la commande de chargement : "EXECC".

Concernant l'ANALYSE DE LA COMMANDE, nous avons chargé le buffer de réception "BUFFC" par quelques unes des commandes prévues, et le déroulement correct du programme a donné tous les paramètres de celles-ci, convertis et sauvegardés, en attente d'exécution.

Des erreurs sur les formats des commandes ont été glissées exprès pendant leur formulation, et le programme d'analyse les a systématiquement détectées et mis dans l'Octet "ERROR" un numéro pour les repérer (voir ANNEXE).

Quant à la commande de chargement "EXECC", on a chargé le "BUFFC" par une ligne de données, et le programme a fait leur stockage dans la RAM de 16k. comme c'était prévu. Ici aussi, les erreurs de format sont détectées.

CONCLUSION

. En conclusion, nous pouvons dire que le système dont nous venons d'étudier le logiciel, permet de résoudre définitivement la programmation des mémoires mono-, bi-, ou tri-tensions. Il sera d'autant plus efficace quand on lui aura adjoint un ensemble "clavier-écran" et une imprimante qui le rendent entièrement autonome. Ceci pourra, d'ailleurs, faire l'objet d'un projet de fin d'études pour les promotions futures.

. Nous étions obligés, dans cet ouvrage, d'omettre la partie du programme relative à l'éxécution de la commande I (impression), vu que celle-ci nécessite toute une étude des interfaces mis en jeu, et que le temps qui nous a été attribué ne l'aurait pas permis.

. Pour en terminer, on voudrait mettre l'accent sur les possibilités d'extension de notre système du point de vue : augmentation du nombre de commandes acceptées ; ou des formats de données ; addition d'autres PROMs ou REPROMs à programmer.
La souplesse de nos programme a été prévue à cet effet.

- Pour l'addition d'une ou plusieurs commandes, des octets "libres" ont été laissés dans la table des commandes "TABCOM".
- Pour programmer d'autres PROMs, il suffit de donner les informations nécessaires dans les tables correspondantes.

REFERENCES BIBLIOGRAPHIQUES

— Catalogue "microprocesseurs et mémoires".
—EFCIS— 1980

— The bipolar microcomputer components data book. —Texas Instruments— 1979

— M6800 programming reference manual.
— MOTOROLA — 1976

— Initiation à la logique programmée et au microprocesseur. — J. COUDERC — 1980

— Projet de fin d'études sur le "hardware" du programmateur. —promotion de juin 83—

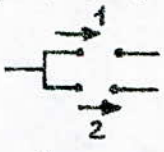
ANNEXE

Tableau d'attribution des numéros aux erreurs :

numéro d'erreur	SIGNIFICATION
$00	pas d'erreur.
$01	absence du caractère "ESCAPE" en tête de la séquence de commande.
$02	erreur de commande : le caractère de la commande est autre que : Z, E, R, P, L, V, I, C.
$03	absence du caractère "Retour-Chariot" en fin de la séquence de commande.
$04	absence de "blanc" entre la commande et ses paramètres : (elle concerne les commandes : P, L, V, I, C).
$05	nombre de caractères ASCII constituant les paramètres P_1 et P_2, supérieur au maximum admis (4 pour P_1 et P_2, pour P, L, V ; 4 pour P_1 et 2 pour P_2, cas de : I, C).
$06	erreur de conversion ASCII / Hexa : (caractère non convertible).
$07	Format de la ligne de données non admis.
$08	erreur sur l'en-tête du format : "S" pour MOTOROLA ; ":" pour INTEL.
$09	nombre de caractères ASCII de données par ligne, différent du nombre prévu.
$0A	dépassement de la capacité de la PROM à programmer.
$0B	PROM (ou REPROM) à programmer non vierge.
$0C	programme vérifié incorrect.

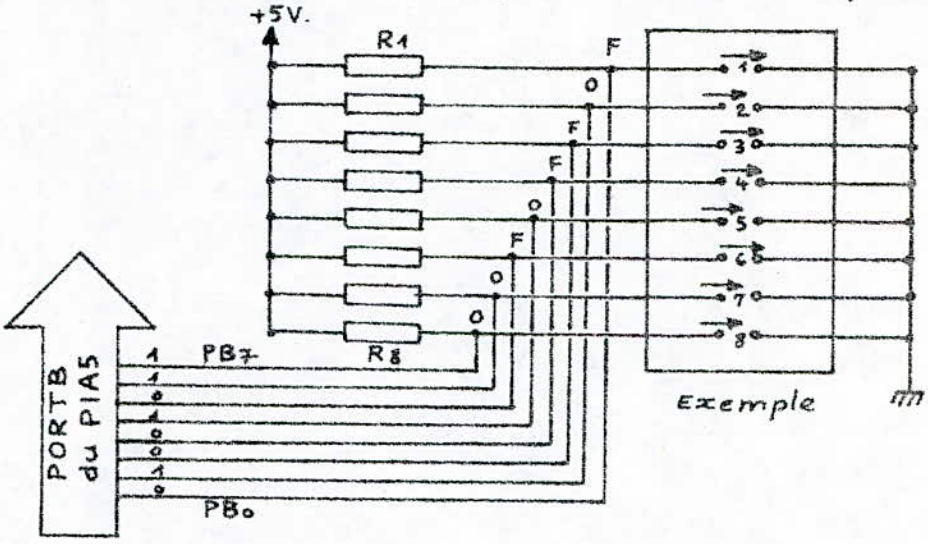
Tableau donnant la position des différents switches pour les REPROMs de la carte support (non-bipolaires).

explication : * switch à 2 positions.



position 1 ouverte ⇒ O.
position 2 fermée ⇒ F.

* même convention pour switch à 8 positions.



TYPE DE REPROM	Switch à 8 positions							
	8	7	6	5	4	3	2	1
TMS 2516	F	F	F	F	F	F	F	F
TMS 2532	F	F	F	F	F	F	F	O
Intel 2716	F	F	F	F	F	F	O	O
Intel 2732	F	F	F	F	F	O	F	F
Intel 2758	F	F	F	F	F	O	O	F

Switches à 2 positions									
SW.5		SW.4		SW.3		SW.2		SW.1	
1	2	1	2	1	2	1	2	1	2
F	O	F	O	O	F	F	O	F	O
F	O	F	O	O	F	F	O	F	O
F	O	F	O	O	F	F	O	F	O
F	O	O	F	O	F	F	O	O	F
F	O	F	O	O	F	F	O	F	O